PROGRAMIRANJE II

2. RAZRADA EFIKASNIH ALGORITAMA

U ovoj glavi uvode se neke osnovne strukture podataka (recimo liste) i uvode se neke osnovne tehnike programiranja (recimo rekurzija i dinamičko programiranje).

2.1. STRUKTURE PODATAKA: LISTE, REDOVI I STEKOVI

Struktura podataka znači vrsta podataka ili vrsta promjenljive ili složena promjenljiva. Za strukturu podataka kaže se da je visokog nivoa ako je to vrsta podataka koja nije ugrađena vrsta podataka u višem programskom jeziku, kakav je recimo Paskal. Na primjer, u Paskalu postoji vrsta podataka skup (`set`), ali ne postoji (nije ugrađena) vrsta podataka graf.

Osnovni primjer strukture podataka visokog nivoa jeste lista. Kaže se lista ili povezana lista ili spisak. Lista se definiše kao konačan niz izvjesnih objekata, u oznaci recimo $x_1, x_2, \ldots, x_n$. Primjer liste je Item1, Item2, Item3, Item4 i vidi se da ova lista ima četiri člana.

Kako se lista čuva u memoriji računara? Drugim riječima, kakvo je memorijsko predstavljanje liste? Postoje dva načina za predstavljanje: a) pomoću pokazivača i b) pomoću dva niza.

Za a), u memoriji se drži onoliko cjelina koliko lista ima članova. Još se drži i jedan poseban pokazivač za koji se kaže da je glava liste. Taj pokazivač ukazuje na prvu cjelinu. Svaka cjelina sastoji se iz dva dijela. U prvom dijelu nalazi se upisan sami podatak tj. član liste. U drugom dijelu nalazi se upisan pokazivač na iduću cjelinu. Ako u drugom dijelu u nekoj cjelini piše vrijednost `nil` (adresa koja ne postoji) onda to znači da se sa tom cjelinom lista završava (pokazivač nas upućuje na nepostojeće mjesto). Na slici 1 dato je predstavljanje liste Podatak1, Podatak2, Podatak3, Podatak4. Vidi se da je Prvi glava liste.

Za b), u memoriji se drže dva niza čija su imena recimo NAME i NEXT. U prvom nizu NAME drže se sami članovi liste, a pomoću drugog niza NEXT definiše se redosljed članova. Ako je i indeks niza onda NAME[i] predstavlja podatak, a NEXT[i] predstavlja indeks sljedećeg člana liste. NEXT[0] stalno upućuje na prvog člana liste. Ako je NEXT[j] = 0 onda to znači da je j indeks posljednjeg člana liste. Na slici 2 dato je predstavljanje liste Podatak1, Podatak2, Podatak3, Podatak4.

Ponekad koristimo englesku riječ, a ponekad našu, u smislu: name – ime, next – sljedeći, item – podatak, newitem – novipodatak, first – prvi.

Prvo se govori o memorijskom predstavljanju, a poslije toga se govori o tome kako da se izvede neka operacija (o algoritmima za operacije). Dvije osnovne operacije sa listom su: umetanje člana i brisanje člana.

Neka Novipodatak treba da bude umetnut u našu listu i to poslije Podatak2. Na slici 3 prikazana je situacija poslije izvršavanja te operacije.

Na slici 4 predstavljena je (na nestrogom paskalu) procedura za umetanje u listu. Smisao parametara procedure je: FREE – indeks neiskorišćenog mjesta u dva niza (NAME i NEXT), ITEM – sami podatak koji se umeće i POSITION – indeks člana liste poslije koga treba da dolazi podatak koji se umeće. Uporedimo sa našim primjerom: FREE = 5, ITEM = NOVIPODATAK, POSITION = 3 (jer PODATAK2 ima indeks 3).

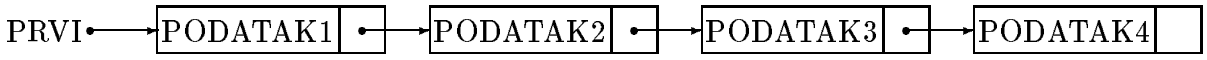
Svaki razuman prevod procedure INSERT u kod za RAM pokazuje da vrijeme potrebno za izvršavanje ove procedure (ovog potprograma) ne zavisi od veličine liste.

Da bi se izbrisao član koji dolazi poslije člana na mjestu i, treba da NEXT[i] dobije vrijednost NEXT[NEXT[i]]. Pored toga, indeks izbrisanog člana može da se smjesti u posebnu listu neiskorišćenih indeksa.

Često je nekoliko različitih listi upisano u isti veliki niz. Tada obično jedna od tih listi služi za pamćenje slobodnih mjesta (slobodnih indeksa) u tom velikom nizu. Ta posebna lista zove se free list. Tada procedura za umetanje preuzima informaciju o slobodnom mjestu (FREE) iz free list, pa free list odsad ima jedan član manje. Dualno kod brisanja.

Navedimo još dvije osnovne operacije sa listama. 1 Konkatenacija ili nadovezivanje dvije liste. 2 Presijecanje liste: da svi članovi poslije određenog člana čine drugu listu. Dvije operacije su uzajamno inverzne.

Lista može da se učini dvostruko povezanom ako se uvede i treći niz PREVIOUS. PREVIOUS[i] označava položaj podatka koji je neposredno ispred podatka čiji je položaj i. Previous – prethodni. Sada imamo tri niza: NAME, NEXT i PREVIOUS. Mi smo dogradili strukturu podataka, tj. mi smo ojačali predstavljanje. Zato sada neke operacije možemo da izvršimo efikasnije.



Slika 1 Predstavljanje liste

	IME	SLJEDECI
0	—	1
1	PODATAK1	3
2	PODATAK4	0
3	PODATAK2	4
4	PODATAK3	2

Slika 2 Predstavljanje liste

	IME	SLJEDECI
0	—	1
1	PODATAK1	3
2	PODATAK4	0
3	PODATAK2	5
4	PODATAK3	2
5	novipodatak	4

Slika 3 Lista sa umetnutim novipodatak

```
PROCEDURE insert(item,free,position);
name[free] ← item;
next[free] ← next[position];
next[position] ← free
```

Slika 4 Procedura insert

Dva važna specijalna slučaja liste su stek (stack) i red (queue). Za stekove i redove je karakteristično da su ograničene mogućnosti za manipulaciju sa članovima liste.

Uzmimo da podaci mogu da budu dodati ili izbrisani samo sa kraja liste. Za takvu listu kaže se da je stek. Analogija: naslagani tanjiri ili metalni novčići u fišeku. Pristupni kraj steka zove se njegovim vrhom, a drugi kraj zove se dno. Samo jedan član steka je dostupan u datom trenutku (gornji član). Kako se stek realizuje? Za držanje članova steka koristi se jedan niz NAME. Neka niz NAME ima ℓ članova i neka se indeksi u nizu NAME kreću u granicama od 0 do $\ell - 1$. Koristi se još samo i jedna cjelobrojna promjenljiva TOP. Njena vrijednost jednaka je indeksu gornjeg člana steka. Na slici 5 prikazan je primjer steka. Vidi se da stek ima tri podatka: ITEM1, ITEM2 i ITEM3. Sada ćemo da nabrojimo operacije sa stekom. a) PUSH. Da bi se novi podatak dodao steku treba povećati TOP za jedan i onda upisati taj podatak

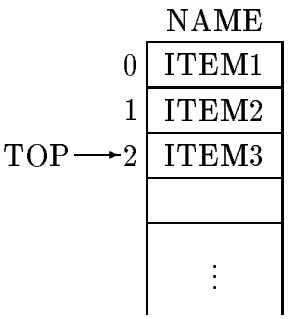
u NAME[TOP]. Prethodno treba provjeriti da li ima mjesta za dodavanje, da ne bi došlo do prekoračenja (overflow). b) Ispitivanje da li je stek pun. Stek je pun ako je TOP $= \ell - 1$. c) POP. Da bi se podatak izbrisao sa steka dovoljno je TOP smanjiti za jedan. Prilikom pokušaja brisanja iz praznog steka, procedura POP treba da saopšti poruku o greški "underflow". POP briše člana i vraća izbrisanog člana kao svoj rezultat. d) Da bi se utvrdilo da li je stek prazan treba provjeriti da li je TOP $= -1$. e) Čitanje gornjeg člana steka bez njegovog brisanja. f) Da se stek inicijalizuje treba izdvojiti prostor od ℓ riječi i dodijeliti ga nizu NAME i treba TOP $\leftarrow -1$.

LIFO znači "last in first out" ili u prevodu posljednji upisan prvi pročitan. To je način pristupa steku ili način rada sa stekom.

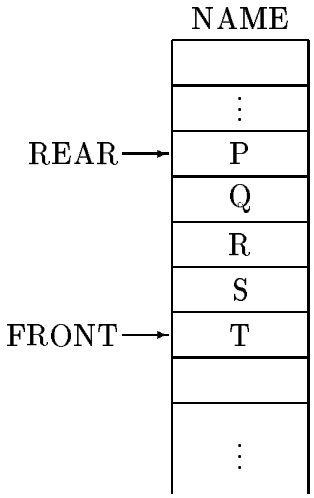
U slučaju reda, podaci se uvijek dodaju na jednom kraju, a brišu sa drugog kraja. Analogija: red čekanja pred šalterom. Red se realizuje pomoću jednog niza NAME i dvije cjelobrojne promjenljive FRONT i REAR, čelo i začelje reda. Na slici 6 prikazan je primjer liste P, Q, R, S, T. Ako želimo da dodamo podatak U ovom redu onda treba FRONT $\leftarrow$ FRONT $+ 1$ i NAME[FRONT] $\leftarrow$ U. Ako želimo da izbrišemo podatak P iz ovog reda onda treba uraditi REAR $\leftarrow$ REAR $+ 1$.

FIFO znači "first in first out" ili u prevodu prvi upisan prvi pročitan. Tako se manipuliše sa članovima reda.

Vidimo da se tokom izvršavanja programa vrijednosti i FRONT i REAR stalno povećavaju. Treba smatrati da poslije NAME[$\ell-1$] slijedi NAME[0]. Drugim riječima, kada se i NAME[$\ell-1$] popuni onda iduće upisivanje izvršiti u NAME[0], ako nije zauzeto, zatim u NAME[1], itd.



Slika 5
Realizacija steka



Slika 6
Realizacija reda

2.2. PREDSTAVLJANJE SKUPOVA

Neka je U univerzalni skip. Neka je card$(U) = n$ i neka je $U = \{a_1, a_2, \ldots, a_n\}$. Navedimo tri moguća načina da se u memoriji računara predstavi skup $S \subset U$. Prvi način: pomoću liste koja sadrži članove skupa S. Drugi način: pomoću vektora bitova v, v. ranije III. Skupu S pridruži se njemu odgovarajući tzv. karakteristični vektor v, gdje je i–ta komponenta vektora $= 1$ ili $= 0$, u zavisnosti od toga da li $a_i \in S$ ili pak suprotno $a_i \notin S$, za $1 \leq i \leq n$. Treći način: pomoću niza cijelih brojeva $A[1..n]$, gdje je $A[i] = 1$ ili $A[i] = 0$ kada $a_i \in S$ odnosno $a_i \notin S$.

Osnovne operacije nad skupovima su računanje unije, presjeka i slično. Algoritmi za osnovne operacije opisuju se različito u zavisnosti od izabranog načina za predstavljanje. A različita je i složenost tih algoritama. Biće rađeno kasnije.

2.3. GRAFOVI

U ovom naslovu uvodi se pojam grafa i uvode se obične strukture podataka za predstavljanje grafova.

Definicija. Neka je V konačan neprazan skup. Za člana skupa V kaže se da je vrh (engl. vertex) ili da je čvor. Neka je $E \subset V \times V$. Za člana skupa E kaže se da je ivica (engl. edge) ili da je grana. Uređeni par skupova $G = (V, E)$ naziva se usmjerenim grafom.

Vidimo da su članovi skupa E uređeni parovi vrhova. Tako da (a, a) može da bude ivica u usmjerenom grafu. Usmjereni graf sa n vrhova ima najviše n^2 ivica.

Neka je (v, w) ivica u usmjerenom grafu. Za v se kaže da je početak te ivice, a za w se kaže da je njen kraj. Kaže se da ta ivica vodi od vrha v ka vrhu w. Takođe se kaže da je vrh w susjedni vrhu v. Stepen ili izlazni stepen vrha v jednak je broju vrhova koji su susjedni tom vrhu v. Slično se definiše ulazni stepen vrha.

Mi smo povukli strelicu od jedne tačke do druge tačke ili od tačke ka njoj samoj. Umjesto toga možemo dvije tačke da povežemo sa jednom duži. Tada to moraju da budu dvije različite tačke. Naravno da duž, za razliku od strelice, nije orijentisana.

Definicija. Neka je V konačan neprazan skup vrhova. Neka je E skup čiji su članovi dvočlani podskupovi skupa V. Za članove skupa E kaže se da su ivice. Uređeni par skupova $G = (V, E)$ naziva se neusmjerenim grafom.

Neusmjereni graf sa n vrhova ima najviše $\binom{n}{2} = \frac{1}{2}n(n-1)$ ivica.

Neka je $\{v, w\}$ ivica u neusmjerenom grafu (znači da je $v \neq w$). Tada se kaže da je vrh v susjedni vrhu w. Isto se kaže da je vrh w susjedni vrhu v. Ili se prosto kaže da su ta dva vrha jedan drugom susjedni. Kada se radi o neusmjerenom grafu, stepen nekog vrha definiše se kao broj njemu susjednih vrhova.

Neka je $\{v, w\}$ ivica u neusmjerenom grafu. Obično se ta ivica označava kao (v, w). Šta se dešava ako se usvoji takav način označavanja? Tada su (v, w) i (w, v) dvije različite oznake za jednu te istu ivicu. Već je rečeno da (a, a) ne može biti ivica u neusmjerenom grafu.

U nastavku se definišu pojmovi put, dužina puta, jednostavni put i ciklus.

U usmjerenom ili neusmjerenom grafu, niz ivica oblika $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$ naziva se putem. Kaže se da put vodi od vrha v_1 ka vrhu v_n i kaže se da je dužina puta jednaka $n-1$. Put se obično predstavlja kao niz svojih ivica, tj. kao $v_1, v_2, v_3, \ldots, v_{n-1}, v_n$. Posebno, pojedini vrh predstavlja put koji vodi od tog vrha ka njemu samom i čija je dužina jednaka 0.

Za put se kaže da je jednostavan ako su sve njegove ivice međusobno različite i ako su još i svi njegovi vrhovi (izuzimajući možda prvi i posljednji vrh) međusobno različiti. Jednostavni put koji počinje i završava se u jednom te istom vrhu i čija dužina iznosi barem 1 naziva se ciklusom.

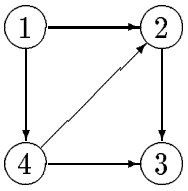
Uočavamo da dužina ciklusa u neusmjerenom grafu iznosi najmanje 3.

Prelazimo na strukture podataka za predstavljanje grafova. Biće izložene dvije mogućnosti za predstavljanje: matrica susjednosti (ili matrica koincidencije) i liste susjednosti. Jedna i druga mogućnost biće definisane za slučaj usmjerenog grafa, a slične okolnosti važe i za neusmjereni graf.

Razmotrimo usmjereni graf $G = (V, E)$. Neka graf ima n vrhova, tj. neka je $\operatorname{card} V = n$, ponekad se piše $\|V\| = n$. Matrica susjednosti grafa G je kvadratna matrica oblika $n \times n$, u oznaci recimo A. Članove matrice označavamo kao $A[i, j]$, gdje je $1 \leq i \leq n$ i $1 \leq j \leq n$. Ako u grafu postoji ivica od vrha i ka vrhu j onda se stavlja da je $A[i, j] = 1$, a inače se stavlja da je $A[i, j] = 0$. Vidimo da se u ovoj definiciji implicitno uzima da su vrhovi grafa uređeni (da je definisan redosljed vrhova), tj. uzima se da je $V = \{v_1, v_2, \ldots, v_n\}$ ili da je $V = \{1, 2, \ldots, n\}$.

Usmjereni graf sa n vrhova može da bude predstavljen i pomoću svojih tzv. listi susjednosti. Svakom vrhu odgovara jedna lista, tako da se graf predstavlja sa n listi. Lista susjednosti određenog vrha sadrži sve vrhove koji su njemu susjedni.

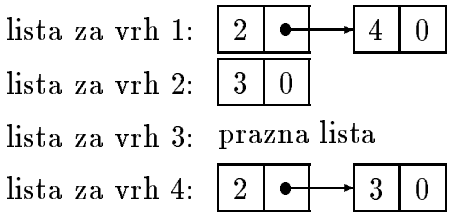
Pogledajmo primjer. Na slikama su prikazani redom: a) jedan usmjereni graf sa četiri vrha, b) njegova matrica susjednosti A, c) njegove liste susjednosti i d) tabelarno predstavljanje listi susjednosti.



a)

	1	2	3	4
1	0	1	0	1
2	0	0	1	0
3	0	0	0	0
4	0	1	1	0

b)



c)

	END	NEXT
1		5
2		7
3		0
4		8
5	2	6
6	4	0
7	3	0
8	2	9
9	3	0

d)

Za d), niz NEXT sadrži na pozicijama od $i = 1$ do $i = n$ (od $i = 1$ do $i = 4$) glave pojedinih listi susjednosti. Na pozicijama $i = n + 1$, $i = n + 2$, ... sadržane su u dva niza END i NEXT same liste susjednosti. END – "završetak ivice je".

Nije osobito efikasno pomoću matrice susjednosti predstavljati "rijetke" grafove (one koji imaju srazmjerno mali broj ivica). Za njihovo prikazivanje bolje je koristiti liste susjednosti.

2.4. DRVETA

U ovom naslovu uvode se pojmovi drvo i binarno drvo. Još se govori o jednom načinu za predstavljanje binarnog drveta. Kaže se drvo ili stablo ili engl. tree.

Def. 1. Usmjereni graf u kome nema ciklusa zove se usmjereni aciklični graf. Drvo ili usmjereno drvo jeste usmjereni aciklični graf koji ima sljedeća tri svojstva. 1 Postoji tačno jedan vrh koji nije završetak nijedne ivice, taj vrh naziva se korijenom, engl. root. 2 Svakom vrhu, osim korijena, odgovara tačno jedna ivica čiji je završetak taj vrh. 3 Postoji put od korijena ka svakom vrhu.

Uslov 1 govori da je ulazni stepen korijena jednak nula, a uslov 2 govori da je ulazni stepen svih ostalih vrhova jednak jedan. Lako se pokazuje da je put od korijena ka vrhu o kome se govori u uslovu 3 – jedinstven. Na slici 1 prikazan je primjer drveta. Usmjereni graf koji se sastoji od jednog ili više drveta naziva se šumom.

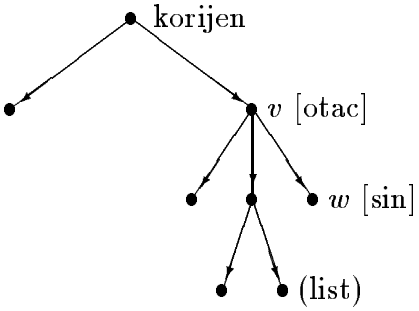
Def. 2. Neka je $F = (V, E)$ graf koji je šuma. Ako uređeni par vrhova (v, w) pripada skupu E onda se kaže da je vrh v otac za vrh w (ili neposredni prethodnik) i kaže se da je w sin od v (ili neposredni sljedbenik ili neposredni nasljednik). Ako postoji put od vrha v ka vrhu w onda se kaže da je v predak za w (ili prethodnik) i kaže se da je w potomak od v (ili sljedbenik). Ako je osim toga $v \neq w$ onda se kaže da je v pravi predak za w i da je w pravi potomak od v. Vrhovi koji nemaju pravih potomaka nazivaju se listovima. Vrh v zajedno sa svim svojim pravim potomcima i odgovarajućim ivicama naziva se poddrvetom od F, vrh v naziva se korijenom tog poddrveta.

Def. 3. Razmotrimo jedno drvo i razmotrimo jedan vrh v u tom drvetu. Dubina vrha v jeste dužina puta od korijena ka v. Visina vrha v jeste dužina najdužeg puta od v ka nekom listu. Visina drveta jeste visina korijena. Nivo vrha v jeste razlika između visine drveta i dubine vrha v.

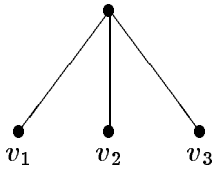
Recimo, na slici 1, dubina vrha v jednaka je 1, njegova visina jednaka je 2, a visina čitavog drveta jednaka je $h(T) = 3$, tako da je nivo vrha v jednak $h(T) - 1 = 3 - 1 = 2$.

Def. 4. Za drvo se kaže da je uređeno ako su sinovi svakog vrha uređeni. Drugim riječima, u skupu svih sinova jednog vrha, prisutna je relacija potpunog uređenja, ovo za svaki vrh.

Kada crtamo uređeno drvo, mi uzimamo da su sinovi svakog vrha uređeni slijeva nadesno. Recimo, na slici 2, $v_1 \preceq v_2 \preceq v_3$.



Slika 1



Slika 2

Def. 5. Binarno drvo jeste drvo koje ima sljedeća dva svojstva. 1 Svaki sin nekog vrha označen je ili kao njegov lijevi sin ili kao njegov desni sin. 2 Nijedan vrh nema više od jednog lijevog sina, niti više od jednog desnog sina. Binarno drvo je uređeno drvo. Naime, smatra se da je lijevi sin lijevo od ili ispred desnog sina.

Def. 6. Razmotrimo jedno binarno drvo i razmotrimo u njemu jedan vrh v. Lijevim poddrvetom T_l vrha v naziva se poddrvo čiji je korijen lijevi sin vrha v, u slučaju da ono postoji, tj. da njegov skup vrhova nije prazan, tj. da v ima lijevog sina. Desnim poddrvetom T_r vrha v naziva se (u slučaju da to poddrvo postoji) poddrvo čiji je korijen – desni sin vrha v. Za svaki vrh iz T_l smatra se da se taj vrh nalazi lijevo od ili ispred bilo kog vrha iz T_r.

Binarno drvo predstavlja se obično pomoću dva niza LEFTSON i RIGHTSON. Neka drvo ima n vrhova. Tada se indeksi u jednom i drugom nizu kreću od $i = 1$ do $i = n$. Indeks $i = 1$ odgovara korijenu. Važi LEFTSON$[i] = j$ ako i samo ako je vrh j lijevi sin vrha i, a ako i nema lijevog sina onda je LEFTSON$[i] = 0$. Slično se definišu i vrijednosti RIGHTSON$[i]$.

Pogledajmo primjer. Na slici 3 prikazani su: a) jedno binarno drvo i b) njegovo predstavljanje.

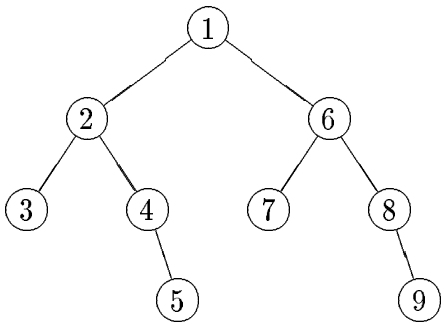
U nastavku se govori o potpunom binarnom drvetu i o obilasku drveta.

Def. 7. Za binarno drvo se kaže da je potpuno ako za neki cio broj k važi da svaki vrh čija je dubina manja od k ima i lijevog i desnog sina i da je svaki vrh čija je dubina k – list.

Potpuno binarno drvo čija je visina k ima tačno $2^{k+1} - 1$ vrhova.

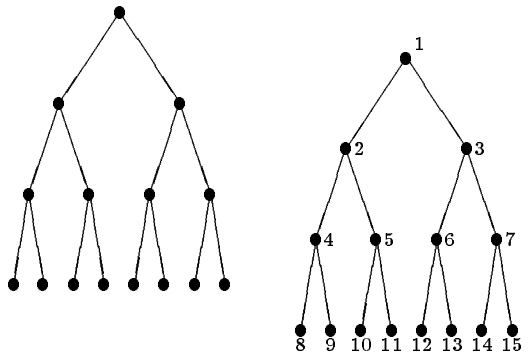
Potpuno binarno drvo obično se predstavlja pomoću jednog niza. Prvi član niza odgovara korijenu. Za vrh koji se nalazi na poziciji i, njegov lijevi sin nalazi se na poziciji $2i$, a njegov desni sin nalazi se na poziciji $2i + 1$. Na taj način, ako je neki vrh na poziciji j onda je njegov otac na poziciji $[j/2]$, ovdje je $j > 1$.

Pogledajmo primjer. Na slici 4 prikazani su: a) potpuno binarno drvo visine $k = 3$ i b) skica njegovog predstavljanja pomoću jednog niza (skica njegove numeracije). Indeksi u nizu kreću se u granicama od $i = 1$ do $i = 2^{k+1} - 1 = 15$.



	LE	RI
1	2	6
2	3	4
3	0	0
4	0	5
5	0	0
6	7	8
7	0	0
8	0	9
9	0	0

Slika 3 a) b) LE = leftson, RI = rightson



Slika 4 a) b)

Algoritmi u kojima se pojavljuju drveta obično zahtijevaju da se izvrši pregledanje ili obilazak drveta, tj. oni zahtijevaju da se svaki vrh drveta jednom posjeti (samo jednom posjeti). Postoji nekoliko načina da se drvo sistematično pregleda, a mi ćemo da navedemo tri obična načina.

Def. 8. Neka je T drvo i neka je r njegov korijen. Uzmimo da je drvo T uređeno i označimo sa $\preceq$ relaciju potpunog uređenja u skupu svih sinova jednog bilo kog vrha drveta. Neka su $v_1, v_2, \ldots, v_k$ sinovi korijena i neka važi $v_1 \preceq v_2 \preceq \ldots \preceq v_k$, ovdje je $k \geq 0$. U slučaju $k = 0$ drvo se sastoji od jedinog vrha r (pitanje obilaska je trivijalno).

Obilazak drveta T po prednjem redosljedu definiše se kako slijedi: 1 Posjeti korijen r i 2 Posjeti po prednjem redosljedu poddrveta čiji su korijeni $v_1, v_2, \ldots, v_k$, tim redom.

[Prvo korijen, zatim poddrvo čiji je korijen v_1 (i to po prednjem redosljedu), onda poddrvo čiji je korijen v_2 (i to po prednjem redosljedu), ..., na kraju poddrvo čiji je korijen v_k (i to po prednjem redosljedu).]

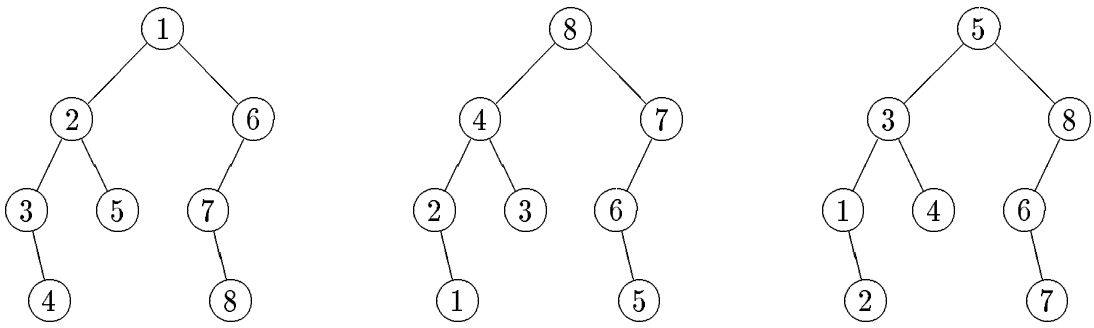
Obilazak drveta T po zadnjem redosljedu definiše se kako slijedi: 1 Posjeti po zadnjem redosljedu poddrveta čiji su korijeni $v_1, v_2, \ldots, v_k$, tim redom i 2 Posjeti korijen r.

Za binarno drvo T, obilazak po srednjem redosljedu definiše se kako slijedi: 1 Posjeti po srednjem redosljedu lijevo poddrvo korijena r (ako to poddrvo postoji), 2 Posjeti r i 3 Posjeti po srednjem redosljedu desno poddrvo korijena r (ako to poddrvo postoji).

Umjesto obilazak po prednjem, zadnjem i srednjem redosljedu mi ćemo govoriti preorder, postorder i inorder obilazak. Nazivi su slični sa sljedećim nazivima. Primjer aritmetičkog izraza zapisanog na prefiks, postfiks, infiks način je $+\,a\,b$, $a\,b\,+$, $a + b$.

Ponovimo da je srednji redosljed definisan samo za binarno drvo.

Pogledajmo primjer. Vrhove jednog te istog binarnog drveta numerisali smo: a) po prednjem, b) po zadnjem i c) po srednjem redosljedu. V. sliku 5.



Slika 5 a) b) c)

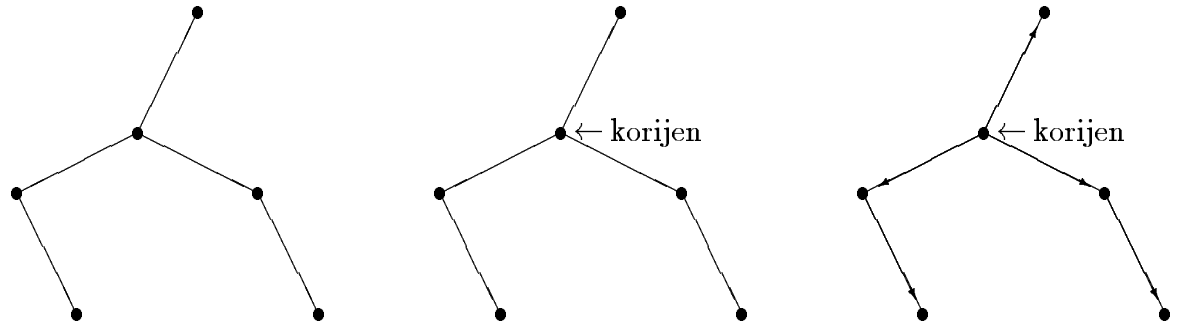
Zapaziti da je u bilo kojoj od tri numeracije – lijevo poddrvo ispred desnog.

Na kraju, kažimo o još dva moguća načina da se definiše drvo.

Def. 9. Za neusmjereni graf koji je povezan (između svaka dva vrha postoji put) i acikličan (nema ciklusa) kaže se da je neusmjereno drvo. Za neusmjereno drvo u kome je jedan vrh izdvojen da bude korijen kaže se da je korijeno neusmjereno drvo.

Na slici 6 dati su primjeri za: a) neusmjereno drvo i b) korijeno neusmjereno drvo.

Postoji uzajamno jednoznačna korespondencija između drveta i korijenih neusmjerenih drveta. Ako se u korijenom neusmjerenom drvetu sve ivice orijentišu u smjeru od korijena onda se tako dobija jedno drvo. Za ovo, v. sliku 6 c).



Slika 6 a) b) c)

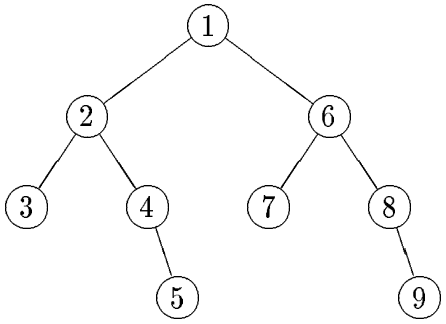
Drvo može da se definiše i sljedećom rečenicom. Skup vrhova V je konačan, u skupu vrhova postoji jedan poseban vrh – korijen r, ostali vrhovi mogu da se podijele u $n \geq 0$ disjunktnih podskupova $V_1, \ldots, V_n$. Svakom V_k odgovara jedno drvo. Ono je poddrvo korijena.

2.5. REKURZIJA

Zadatak o inorder numeraciji vrhova binarnog drveta biće riješen sa upotrebom rekurzije i bez upotrebe rekurzije. Biće još riječi i o ostvarenju rekurzije u slučaju modela RAM.

Potprogram koji poziva samog sebe (neposredno ili posredno) naziva se rekurzivnim potprogramom.

Neka je binarno drvo predstavljeno pomoću dva niza LEFTSON i RIGHTSON na običan način, što čini ulazne podatke. Razmotrimo zadatak o inorder numeraciji njegovih vrhova (o numeraciji po srednjem redosljedu). Program treba svakom vrhu da dodijeli njegov broj (njegov redni broj). Tako da niz NUMBER koji se obrazuje tokom izvršavanja čini izlaz, gdje je NUMBER[i] broj koji odgovara vrhu i. Pogledajmo primjer. Prikazani su: binarno drvo i njegovo predstavljanje i rezultat NUMBER koji treba da bude dobijen. Znamo da inorder zakon za neki vrh glasi: prvo lijeva strana vrha, onda vrh i na kraju desna strana vrha, s tim da se lijeva strana obilazi po inorder zakonu, a takođe se i desna strana obilazi po inorder zakonu. U nastavku će biti data dva algoritma koji služe za postavljeni zadatak.



i	LE	RI	NU
1	2	6	5
2	3	4	2
3	0	0	1
4	0	5	3
5	0	0	4
6	7	8	7
7	0	0	6
8	0	9	8
9	0	0	9

LE = leftson[i], RI = rightson[i], NU = number[i]

Prvo rješenje (sa upotrebom rekurzije). U ovom algoritmu očito se postupa po zakonu: lijevo poddrvo vrha, onda taj vrh i na kraju desno poddrvo vrha.

U programu je prisutna globalna promjenljiva COUNT čiji je smisao – redni broj koji se trenutno dodjeljuje. Ona startuje od vrijednosti 1. Slijedi tekst potprograma:

```
PROCEDURE inorder(vertex);
IF leftson[vertex] ≠ 0 THEN
    inorder(leftson[vertex]);
number[vertex] ← count;
count ← count + 1;
IF rightson[vertex] ≠ 0 THEN
    inorder(rightson[vertex])
```

Slijedi tekst glavnog programa, takođe na neformalnom jeziku. U primjeru je ROOT = 1:

```
count ← 1;
inorder(root)
```

Drugo rješenje (bez upotrebe rekurzije).
Slijedi tekst programa, a zatim objašnjenja.

```
        count ← 1;
        vertex ← root;
        stack ← prazno;
  left: WHILE leftson[vertex] ≠ 0 DO
            BEGIN
            upiši vertex u stack;
            vertex ← leftson[vertex]
            END;
center: number[vertex] ← count;
        count ← count + 1;
        IF rightson[vertex] ≠ 0 THEN
            BEGIN
            vertex ← rightson[vertex];
            GOTO left
            END;
        IF stack ≠ prazno THEN
            BEGIN
            vertex ← gornji član od stack;
            skrati stack;
            GOTO center
            END
```

Kaže se da se neki vrh dodaje steku ili stavlja na stek ili stavlja na vrh steka, čime se očigledno broj članova steka povećava za jedan. Takođe se kaže da se iz steka uzima član, to je njegov gornji član, čime se očigledno broj članova steka za jedan smanjuje.

U programu se pojavljuju promjenljive VERTEX, COUNT i STACK. Promjenljiva VERTEX odnosi se na tekući vrh drveta, a njena početna vrijednost je ROOT – korijen drveta. Cjelobrojna promjenljiva COUNT ima smisao rednog broja koji se trenutno dodjeljuje, a njena početna vrijednost je 1. Stek STACK sadrži vrhove drveta čije je postojanje evidentirano, a koji čekaju da budu numerisani. U početku je stek prazan.

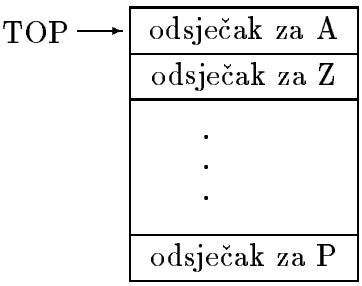
Ako VERTEX ima lijevog sina onda se VERTEX šalje u stek, a lijevi sin postaje nova vrijednost promjenljive VERTEX. A ako pak VERTEX nema lijevog sina onda to znači da je VERTEX došao na red da bude numerisan, nakon čega desni sin od VERTEX preuzima ulogu. Međutim, ako desnog sina nema onda ulogu VERTEX preuzima gornji član steka, s tim da smo sigurni da je njegova lijeva strana već numerisana. Najzad, ako se prilikom obraćanja steku vidi da je stek prazan, ne može se uzeti iz steka, onda to znači da su svi vrhovi drveta već numerisani, pa stop.

Objasnićemo još u ovom naslovu – kako se rekurzivni algoritmi prevode u kod za RAM, odnosno RASP. Objasnićemo slučaj RASP, jer je tako lakše, a ranije je rečeno da se program za RASP može prevesti u program za RAM, kao uostalom i obrnuto.

Prvo nekoliko riječi o programskim jezicima. Fortran ne dopušta rekurzivne potprograme, kao uostalom ni pokazivače (promjenljive koje se generišu dinamički). Paskal i C dopuštaju i jedno i drugo. U slučaju Fortrana, plan raspodjele ili upotrebe memorije (za buduću fazu izvršavanja programa) pravi se statički, tj. čitav bude napravljen u fazi prevođenja programa. U slučaju Paskala ili jezika C nije tako, raspodjela je dinamička, tj. nova upotreba traži se i tokom izvršavanja programa. Zato je moguće da u datom trenutku tokom izvršavanja programa

budu aktivna dva poziva ili više poziva jednog te istog potprograma, recimo potprograma A. Kako se to ostvaruje? Slično kao u opisu koji slijedi.

Glavnu ulogu u realizaciji u slučaju modela RASP ima jedan stek za koji se kaže da je centralni stek ili stek za pozive (call stack). U taj stek bivaju upisani svi lokalni podaci bilo kog potprograma koga je pozvao (tokom svog izvršavanja) neki drugi potprogram ili je svejedno pozvan od samog sebe. Stek je podijeljen na odsječke. Jedan blok uzastopnih memorijskih riječi (memorijskih lokacija) predstavlja jedan odsječak. Svako pozivanje stvara novi odsječak, čija dužina zavisi od konkretnog potprograma koji je pozvan. Donji član steka odnosi se na glavni program P. Uzmimo da se trenutno izvršava potprogram A. Njega je ranije pozvao neki potprogram Z. Trenutni izgled steka prikazan je na slici. Promjenljiva TOP upućuje na gornji odsječak. Umjesto "odsječak za Z" bolje je reći odsječak za tu aktivaciju potprograma Z. Isto tako, umjesto odsječak za A bolje je reći odsječak za tekuću aktivaciju potprograma A.



Uzmimo da A poziva potprogram B i opišimo detaljno sve korake koji treba da budu urađeni (radi ostvarivanja radnje CALL B, ova radnja nalazi se u potprogramu A). Recimo unaprijed da sve potpuno isto važi ako A poziva samog sebe, samo smo mi izabrali lakši način izražavanja.

U fazi zaduživanja (call) izvrše se sljedeći koraci:

a) Na vrh steka postavlja se odsječak odgovarajuće dužine (odsječak za B). Sada stek izgleda ovako, odozgo nadolje: odsječak za B, odsječak za A, odsječak za Z, ..., odsječak za P.

b) Dio prostora u tom odsječku služi za pokazivače na stvarne parametre te aktivacije potprograma B. U okviru toga, ako je stvarni parametar neki izraz onda vrijednost izraza bude prethodno izračunata u okviru A, a pokazivač na izračunatu vrijednost upisuje se u odsječak za B. U slučaju da je stvarni parametar neki složeni podatak, recimo da je niz, dovoljan je jedan pokazivač na prvu memorijsku riječ tog složenog podatka.

c) U tom odsječku predviđen je i prazan ili slobodan prostor za lokalne promjenljive koje koristi B.

d) U tom odsječku je i prostor za povratnu adresu, return address. To je adresa RASP–ove naredbe koja treba da bude prva izvršena kasnije, kada se iz B vratimo u A.

Uzmimo da B vraća jednu vrijednost, tj. uzmimo da je B funkcija. Tada ima i sljedeće: e) U tom odsječku je i prostor za adresu vrijednosti (value address), tj. pokazivač na mjesto u odsječku za A gdje će ta vrijednost biti upisana.

Kontrola se predaje prvoj naredbi potprograma B. Tokom rada potprograma B, adrese njegovih parametara i lokalnih promjenljivih u njegovom odsječku računaju se uz pomoć relativnog adresiranja, po formuli:

$$\text{adresa} = \text{bazna adresa} + \text{offset (address} = \text{base} + \text{offset)},$$

tj. adresa početne (nulte) riječi odsječka + udaljenost od početka odsječka, offset je isti za svaku aktivaciju. B uradi svoje, odnosno neka smo stigli do radnje RETURN (ova instrukcija RETURN nalazi se u potprogramu B).

U fazi razduživanja (return) vrše se redom sljedeći koraci:

a) U slučaju da je B funkcija, izračunata vrijednost upisuje se u odsječak za A na odgovarajuće mjesto, saglasno adresi vrijednosti.

b) Iz steka se spašava povratna adresa.

c) Odsječak steka za B uklanja se iz steka, čime na vrh steka opet dolazi odsječak za A.

d) Spašena vrijednost šalje se u programski brojač LC. Time se kontrola predaje potprogramu A, čiji rad se obnavlja, čiji rad se nastavlja od odgovarajućeg mjesta, očito saglasno tekućoj vrijednosti programskog brojača.

Povratna adresa je adresa neke naredbe, očito. Time se objašnjava zašto smo govorili o RASP–u, a ne o RAM–u.

Na kraju, algoritam u kome se pojavljuju rekurzivni pozivi potprograma može da bude opisan i sredstvima Fortrana. Samo što programeru sleduje čitav posao oko zaduživanja i razduživanja. Ako Paskal ili C onda veći dio posla obavlja računar, odnosno (samim tim) programeru ostaje manje posla.

2.6. PODIJELI PA VLADAJ

U ovom naslovu govori se o tehnici programiranja "podijeli pa vladaj". Taj princip govori da zadatak treba podijeliti na nekoliko manjih zadataka i onda riješiti te manje zadatke. Kombinacijom njihovih rješenja dobija se rješenje polaznog zadatka. Ako su mali zadaci istog tipa kao i polazni zadatak onda je postupak rješavanja rekurzivan. Ilustrovaćemo na dva primjera.

Prvi primjer (zadatak o najvećem i najmanjem članu skupa).

Razmotrimo sljedeći zadatak. Dat je skup S. Sastaviti program za određivanje najvećeg člana skupa i najmanjeg člana skupa. Složenost programa (algoritma) neka se mjeri brojem izvršenih operacija poređenja.

Neka n označava broj članova skupa. Razmotrimo prvo tzv. obični postupak ili algoritam A_1 za rješavanje postavljenog zadatka. Prvo se izvrši $n-1$ poređenje da se odredi najveći član skupa. Zatim se izvrše još $n-2$ poređenja (među preostalih $n-1$ članova skupa) da se odredi i najmanji član skupa. Tako da je složenost:

$$T(n) = n - 1 + n - 2 = 2n - 3 \text{ ili } T(n) \sim 2n.$$

Prelazimo na drugi algoritam A_2 za rješavanje istog zadatka. U A_2 se koristi pristup "podijeli pa vladaj". Pretpostavlja se da je broj n stepen broja 2, tj. pretpostavlja se da je $n = 2^k$ za neki cio broj k. Sada se skup S podijeli na dva dijela S_1 i S_2. Podskupovi S_1 i S_2 imaju po $n/2$ članova. Za S_1 se riješi postavljeni zadatak, takođe i za S_2, i to ponovo pomoću dijeljenja na dva podskupa, itd. Slijedi tekst rješenja na nestrogom paskalu:

```
PROCEDURE maxmin(S);
IF ||S|| = 2 THEN
    BEGIN LET S = {a, b};
    IF a < b THEN RETURN (b, a) ELSE RETURN (a, b) END;
podijeli S na dva podskupa S1 i S2 jednake veličine;
(max1, min1) ← maxmin(S1);
(max2, min2) ← maxmin(S2);
IF max1 < max2 THEN c ← max2 ELSE c ← max1;
IF min1 < min2 THEN d ← min1 ELSE d ← min2;
```

RETURN (c, d)

Iz gramatike nestrogog paskala. LET znači neka bude ili uvedimo oznaku. RETURN izraz znači povratak, s tim da izraz predstavlja rezultat (predstavlja ono što potprogram vraća).

O programskoj realizaciji. Skup S može da bude predstavljen kao niz dužine n. Tada, djelimični skupovi su podnizovi tog niza. Djelimični skup je definisan ako se odaberu početni i krajnji indeks odgovarajućeg podniza. Ta dva indeksa mogu da posluže kao parametri potprograma.

Kolika je složenost $T(n)$ algoritma A_2, tj. koliko se operacija poređenja izvrši tokom izvršavanja algoritma? Važi jednakost:

$$T(n) = \begin{cases} 1, & \text{ako je } n = 2 \\ 2T(n/2) + 2, & \text{ako je } n = 4, 8, 16, \ldots \end{cases}$$

Zaista, iz teksta potprograma maxmin vidi se da on poziva samog sebe na dva mjesta, na jednom i drugom mjestu sa prepolovljenim dimenzionim brojem $n/2$, zato $2T(n/2)$. Još IF max1 < max2 i IF min1 < min2, pa tako $2T(n/2) + 2$.

Vidimo da su vrijednosti $T(n)$ definisane. Želimo da dobijemo zatvoreni izraz za te vrijednosti (da riješimo rekurentnu relaciju). Uvedimo smjenu nezavisno promjenljive $n = 2^k$ i uvedimo drugu oznaku za zavisno promjenljivu $T(n) = T(2^k) = a_k$. Prepisujemo rekurentnu relaciju u novim oznakama: $a_1 = 1$, $a_k = 2a_{k-1} + 2$ za $k = 2, 3, 4, \ldots$. Lako nalazimo sljedeće:

$$a_k = 2a_{k-1} + 2 = 2(2a_{k-2} + 2) + 2 = 2^2 a_{k-2} + 2^2 + 2 = 2^3 a_{k-3} + 2^3 + 2^2 + 2$$

$$= \ldots = 2^{k-1} a_1 + 2^{k-1} + 2^{k-2} + \ldots + 2^2 + 2 = 2^{k-1} + 2^k - 2 = \frac{3}{2} \cdot 2^k - 2$$

(znamo da je $1 + 2 + 4 + \ldots + 2^{k-1} = 2^k - 1$). Tako da je $T(n) = \frac{3}{2}n - 2$. Vidimo da je složenost algoritma A_2 manja od složenosti algoritma A_1, tj. vidimo da je A_2 bolji od A_1.

Ako dimenzioni broj zadatka n nije oblika 2^k onda se postavljenom zadatku može pridružiti zadatak istog tipa većeg dimenzionog broja n' s tim da n' bude oblika 2^k, pa se riješi pridruženi zadatak. Bolje je da se za rješavanje postavljenog zadatka primijeni postupak gdje se na svakom koraku zadatak dijeli na dva manja zadatka čiji se dimenzioni brojevi poklapaju ili se za jedan razlikuju.

Drugi primjer (zadatak o množenju dva broja).

Sastaviti program za množenje dva binarno prikazana broja.

Neka je n broj cifara jednog i drugog broja (broj bita). Neka se složenost programa mjeri po kriterijumu *II*. Računanja bit po bit, tj. neka se broji koliko ima operacija nad bitovima.

Ako se primijeni obični algoritam A_1 onda složenost iznosi $T(n) = O(n^2)$. Uzastopno se šiftuje prvi činilac i uzastopno se vrše sabiranja.

Prelazimo na izlaganje drugog algoritma A_2 u kome se koristi pristup "podijeli pa vladaj". Sada se pretpostavlja da je broj n stepen broja 2. Postupak se sastoji u tome da se svaki od dva činioca podijeli na dva dijela, jedan i drugi dio od po $n/2$ binarnih cifara. Onda se množe ti djelovi, itd. rekurzivno.

Za pripremu, neka je $a(x) = a_1 x + a_0$, $b(x) = b_1 x + b_0$ i $c(x) = a(x)b(x) = c_2 x^2 + c_1 x + c_0$. Tada su koeficijenti polinoma $c(x)$ očito jednaki $c_2 = a_1 b_1$, $c_1 = a_1 b_0 + a_0 b_1$ i $c_0 = a_0 b_0$. Za njihovo računanje treba četiri operacije množenja. Međutim, oni mogu da budu izračunati i sa svega tri operacije množenja, i to: $c_2 \leftarrow a_1 b_1$, $c_0 \leftarrow a_0 b_0$, $d \leftarrow (a_1 + a_0)(b_1 + b_0)$, $c_1 \leftarrow d - c_2 - c_0$.

Za pripremu, ako je $x = (x_{n-1}x_{n-2}\ldots x_1x_0)_2$ onda je očito $x = 2^p(x_{n-1}\ldots x_p)_2 + (x_{p-1}\ \ldots\ x_0)_2$, gdje je $p = n/2$.

Uvedimo oznake. Neka je $k \geq 0$ cio broj i neka je $n = 2^k$. Neka su x i y dva broja koji imaju po n binarnih cifara i neka treba izračunati proizvod $z = xy$. Imamo da je $x = 2^{n/2}a + b$ i $y = 2^{n/2}c + d$, gdje su a, b, c i d brojevi koji imaju po $n/2$ binarnih cifara. U programu se računa po sljedećim formulama:

$$u \leftarrow ac,\ \ v \leftarrow bd,\ \ w \leftarrow (a+b)(c+d),\ \ z \leftarrow 2^n u + 2^{n/2}(w-u-v) + v.$$

Koliko ukupno treba operacija nad bitovima $T(n)$? Za tri množenja trošak iznosi $3T(n/2)$, jer će ta množenja da se obave po istom algoritmu. Množenje nekog broja sa osnovom brojnog sistema na stepen (kao $2^n u$) samo je prividno množenje, odnosno ostvaruje se pomoću šiftovanja (pomjeranja), tako da je trošak reda n. Trošak za sabiranja i oduzimanja takođe je reda n. Zapaziti da trošak za računanje w može da bude veći od $T(n/2)$ zato što $a+b$ može da ima $n/2+1$ cifara (može da ima jednu cifru više), a takođe i $c+d$. Ipak, taj trošak može da bude veći od $T(n/2)$ samo za izraz koji je reda n. Ukupno, $3T(n/2)$ + linearni izraz. Prema tome, važe relacije:

$$T(1) \leq c \quad \text{i} \quad T(n) \leq 3T(n/2) + cn \ \text{ za } \ n = 2, 4, 8, \ldots,$$

za neku konstantu $c > 0$.

Želimo da riješimo rekurentnu relaciju. Ako pišemo $n = 2^k$ i $a_k = T(2^k)$ onda je dato $a_0 \leq c$ i $a_k \leq 3a_{k-1} + 2^k c$ za $k = 1, 2, 3, \ldots$. Tako da imamo:

$$a_k \leq 3a_{k-1} + 2^k c \leq 3(3a_{k-2} + 2^{k-1}c) + 2^k c = 3^2 a_{k-2} + 3\cdot 2^{k-1}c + 2^k c \leq$$

$$3^2(3a_{k-3} + 2^{k-2}c) + 3\cdot 2^{k-1}c + 2^k c = 3^3 a_{k-3} + 3^2\cdot 2^{k-2}c + 3\cdot 2^{k-1}c + 2^k c \leq$$

$$\ldots \leq 3^k a_0 + 3^{k-1}\cdot 2c + 3^{k-2}\cdot 2^2 c + \ldots + 3\cdot 2^{k-1}c + 2^k c =$$

$$3^k c\left(1 + \frac{2}{3} + \left(\frac{2}{3}\right)^2 + \ldots + \left(\frac{2}{3}\right)^{k-1} + \left(\frac{2}{3}\right)^k\right) \leq$$

$$3^k c\left(1 + \frac{2}{3} + \left(\frac{2}{3}\right)^2 + \ldots\right) = 3^k c\cdot 3 = \text{const}\cdot 3^k$$

(zbir beskonačne geometrijske progresije: $1 + q + q^2 + \ldots = \dfrac{1}{1-q}$ za $-1 < q < 1$),

$$a_k \leq \text{const}\cdot 3^k,\ \ T(n) \leq \text{const}\cdot 3^{\log_2 n} = \text{const}\cdot n^{\log_2 3}.$$

Možemo se uvjeriti da je zaista $3^{\log_2 n} = n^{\log_2 3}$ tako što ćemo primijeniti $\log_2$ na lijevu i desnu stranu, čime dobijamo $\log_2 n\cdot\log_2 3 = \log_2 3\cdot\log_2 n$.

$$T(n) \leq \text{const}\cdot n^{\log_2 3} = \text{const}\cdot n^a, \ \text{ gdje je } \ a = \log_2 3 = 1{,}59$$

$$T(n) = O(n^a)$$

Algoritam A_1 ima složenost reda n^2, dok A_2 ima složenost reda n^a. A_2 ima manji red složenosti, tako da je bolji.

2.7. BALANSIRANJE, ALGORITAM MERGESORT

Princip balansiranja govori da prilikom podjele zadatka na dva manja zadatka ti manji zadaci treba da budu svi iste veličine ili približno iste veličine. Ilustrovaćemo na primjeru koji slijedi. Zadatak: dat je niz od n članova, urediti niz u rastući oblik. Pogledajmo dva algoritma A_1 i A_2 koji mogu da posluže. Obični algoritam A_1 predviđa da se uradi kako slijedi. Prvo se među svih n članova odredi najmanji. Za ovo se potroši $n-1$ operacija poređenja. Zatim se rješava zadatak o uređivanju ostalih $n-1$ članova. I to se rješava na isti način. Upravo, među tih preostalih $n-1$ članova odredi se najmanji, itd. Izračunajmo složenost algoritma A_1. Označimo sa $T(n)$ ukupan broj izvršenih operacija poređenja. Imamo da je $T(1)=0$ i $T(n)=n-1+T(n-1)$ za $n>1$,

$$x_n = n-1+x_{n-1} = n-1+n-2+x_{n-2} = \ldots = n-1+n-2+\ldots+1.$$

Izlazi $T(n)=\binom{n}{2}=\frac{1}{2}n(n-1)$, tj. $T(n)\sim\frac{1}{2}n^2$. Vidimo da se u algoritmu A_1 zadatak veličine n dijeli na dva manja zadatka čije su veličine 1 i $n-1$. Bolje je da se primijeni algoritam u kome se zadatak dijeli na dva manja zadatka veličine po $n/2$ približno. Prelazimo na algoritam A_2. To je mergesort ili uređivanje pomoću spajanja (uređivanje ujedinjavanjem). Ulazni podaci čine niz $x_1, x_2, \ldots, x_n$. Izlazni podatak ili rezultat treba da bude niz $y_1, y_2, \ldots, y_n$ koji je permutacija ulaznog niza i koji zadovoljava uslov $y_1 \le y_2 \le \ldots \le y_n$. Pretpostavimo da je n oblika 2 na stepen, $n=2^k$ za neko $k\ge 1$.

Opišimo A_2. Neka je $p=n/2$ i $q=p+1$. Zamislimo da su dva podniza $x_1, x_2, \ldots, x_p$ i $x_q, x_{q+1}, \ldots, x_n$ već uređeni. Tada se ukupno rješenje dobija spajanjem ta dva podniza. Spajanje počinje time što se uporede x_1 i x_q. Manji od ta dva broja šalje se na izlaz. Veći od ta dva broja uporedi se sa sljedećim članom suprotnog podniza. Itd. Kada se jedan podniz iscrpi onda na izlaz poslati preostale članove suprotnog podniza. Za realizaciju spajanja treba $\le n-1$ operacija poređenja.

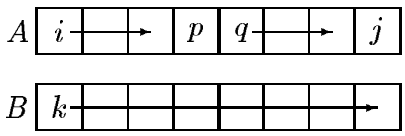
A kako postići da dva podniza budu uređena? Na isti način, tj. algoritam A_2 je rekurzivan.

U nastavku je algoritam mergesort izražen na jeziku nestrogi paskal. Dat je glavni program i data su dva potprograma MERGE i SORT koji se koriste. Prvo potprogrami:

```
    PROCEDURE MERGE(i,j);
    p ← (i+j-1)/2; q ← p+1; k ← i-1;
2:  k ← k+1;
    IF k = j+1 THEN GOTO 1;
    IF i = p+1 THEN BEGIN B[k] ← A[q]; q ← q+1; GOTO 2 END;
    IF q = j+1 THEN BEGIN B[k] ← A[i]; i ← i+1; GOTO 2 END;
    IF A[i] < A[q] THEN BEGIN B[k] ← A[i]; i ← i+1 END
    ELSE BEGIN B[k] ← A[q]; q ← q+1 END; GOTO 2;
1:  prepiši B[i], B[i+1], ..., B[j] u A[i], A[i+1], ..., A[j];
    RETURN

PROCEDURE SORT(i,j);
IF i = j THEN RETURN;
p ← (i+j-1)/2; q ← p+1;
SORT(i,p);
SORT(q,j);
MERGE(i,j);
RETURN
```

upiši članove niza tj. upiši ulazne podatke u $A[1], A[2], \ldots, A[n]$;
SORT(1,n);
štampaj rezultat tj. štampaj $A[1], A[2], \ldots, A[n]$



U nastavku se računa složenost $T(n)$ algoritma mergesort. Neka $T(n)$ označava ukupan broj izvršenih operacija poređenja i to u najgorem slučaju. Imamo da je $T(1) = 0$ i

$$T(n) = 2T(n/2) + n - 1 \text{ za } n = 2, 4, 8, \ldots.$$

Želimo da dobijemo zatvoreni izraz za funkciju složenosti. Stavimo $n = 2^k$ i $a_k = T(n)$. Dato je $a_0 = 0$ i

$$a_k = 2a_{k-1} + 2^k - 1 \text{ za } k = 1, 2, 3, \ldots.$$

Zapažajući da je $a_1 = 1$, kao i $k = \log_2 n$, imamo redom:

$$a_k = 2a_{k-1} + 2^k - 1 = 2(2a_{k-2} + 2^{k-1} - 1) + 2^k - 1 = 2^2 a_{k-2} + 2^k - 2 + 2^k - 1 =$$

$$2^3 a_{k-3} + 2^k - 4 + 2^k - 2 + 2^k - 1 = \ldots =$$

$$2^{k-1} a_1 + 2^k - 2^{k-2} + \ldots + 2^k - 4 + 2^k - 2 + 2^k - 1 =$$

$$2^{k-1} + 2^k(k-1) - (2^{k-2} + \ldots + 4 + 2 + 1) = 2^{k-1} + 2^k(k-1) - (2^{k-1} - 1) =$$

$$2^k(k-1) + 1$$

$$a_k \sim 2^k \cdot k \quad T(n) \sim n \log_2 n$$

Odgovor je $T(n) \sim n \log_2 n$ ili $T(n) = O(n \log_2 n)$, ako je broj n oblika 2 na stepen. $n \log_2 n$ je znatno manje od pređašnjeg $\frac{1}{2}n^2$, tako da je algoritam A_2 bolji od algoritma A_1.

Slijede dopune.

Može se pokazati da za složenost algoritma A_2 važi procjena $T(n) \sim n \log_2 n$ za svaki broj $n \geq 1$. Ako je n neparan onda se niz podijeli na dva manja niza čije se dužine razlikuju za jedan.

Može se izvršiti sljedeća dogradnja algoritma. Vidimo da algoritam ima k prolaza, gdje je $n = 2^k$. Vidimo da jedno aktiviranje potprograma MERGE obavi spajanje dva podniza (lijevog i desnog). U pojedinom prolazu može se postupiti na sljedeći način: sve lijeve podnizove držati u jednom fajlu f_1, a sve desne u drugom fajlu f_2.

2.8. DINAMIČKO PROGRAMIRANJE

U ovom naslovu govori se o tehnici programiranja – dinamičko programiranje. Kažimo prvo opisno o toj tehnici programiranja. Nacrtajmo na x–osi duž od $x = 1$ do $x = n$ i neka tačke $x = 1, x = 2, \ldots, x = n$ budu čvorovi. Rješavanju zadatka odgovara prelazak puta od početnog do krajnjeg čvora. Manjem zadatku ili podzadatku odgovara kraći put od $x = i$ do $x = j$ čija je dužina jednaka $l = j - i$. Redom se rješavaju podzadaci, po rastućim vrijednostima razlike l. Ova tehnika može da bude ilustrovana pomoću sljedećeg primjera.

Neka je n prirodan broj i neka su još dati i prirodni brojevi $r_0, r_1, \ldots, r_n$. Razmotrimo proizvod n matrica $M_1 \times M_2 \times \ldots \times M_n$. Ovdje je matrica M_i oblika $r_{i-1} \times r_i$. Znamo da množenje matrica nije komutativna operacija. Međutim, jeste asocijativna operacija. Tako da susjedne matrice mogu da se udružuju po želji u djelimične proizvode. Vidjećemo i na primjeru da raspored udruživanja utiče na ukupan vremenski trošak računanja matrice $M = M_1 \times M_2 \times \ldots \times M_n$. Razmatra se zadatak o određivanju optimalnog rasporeda udruživanja.

Vremenski trošak izražava se brojem aritmetičkih operacija, odnosno izražava se brojem zahtijevanih operacija množenja tipa $x_1 x_2$, gdje $x_1 \in R$ i $x_2 \in R$. Za računanje skalarnog proizvoda dva vektora $(x_1, x_2, \ldots, x_q)$ i $(y_1, y_2, \ldots, y_q)$ potroši se q operacija množenja. Za množenje matrice oblika $p \times q$ matricom oblika $q \times r$ potroši se pqr operacija množenja (matrica koja predstavlja rezultat imaće oblik $p \times r$). Mi želimo da izgradimo efikasan algoritam za određivanje optimalnog rasporeda udruživanja, tj. za određivanje minimalne cijene po kojoj može da bude izračunata matrica–proizvod M.

Primjer. Neka bude $r_0 = 10$, $r_1 = 20$, $r_2 = 50$, $r_3 = 1$ i $r_4 = 100$ i neka je $M = M_1 \times M_2 \times M_3 \times M_4$. Ako se računa kao $M = M_1 \times (M_2 \times (M_3 \times M_4))$ onda se izvrši 125.000 množenja. A ako se računa kao $M = (M_1 \times (M_2 \times M_3)) \times M_4$ onda samo 2.200 množenja,

M_1	$\times$	M_2	$\times$	M_3	$\times$	M_4
$[10 \times 20]$		$[20 \times 50]$		$[50 \times 1]$		$[1 \times 100]$

Ako bi se razmatrali svi mogući rasporedi udruživanja onda bi algoritam koji tako postupa imao eksponencijalnu složenost. Dinamičko programiranje daje algoritam čija je složenost reda n^3.

Opišimo algoritam. Označimo sa m_{ij} **minimalnu** cijenu (broj množenja $x_1 x_2$) računanja djelimičnog proizvoda $M_i \times M_{i+1} \times \ldots \times M_j$, gdje je $1 \leq i \leq j \leq n$. Stavimo $m_{ij} = 0$ kada je $i = j$. Važi sljedeća osnovna formula:

$$m_{ij} = \underset{i \leq k < j}{\text{MIN}} \left(m_{ik} + m_{k+1,j} + r_{i-1} r_k r_j \right) \text{ kada je } i < j.$$

U ovoj formuli, m_{ik} odnosi se na $M_i \times M_{i+1} \times \ldots \times M_k = M'$, $m_{k+1,j}$ odnosi se na $M_{k+1} \times M_{k+2} \times \ldots \times M_j = M''$, a $r_{i-1} r_k r_j$ odnosi se na množenje $M' \times M''$. Drugim riječima, poslužimo se malim primjerom, djelimični proizvod $M_4 \times M_5 \times \ldots \times M_9$ biće u optimalnom rasporedu izračunat na neki od sljedećih načina:

$(M_4) \times (M_5 \times M_6 \times M_7 \times M_8 \times M_9)$ ili

$(M_4 \times M_5) \times (M_6 \times M_7 \times M_8 \times M_9)$ ili itd.

gdje su i M' i M'' već ranije izračunati i to svakako na optimalni način.

Brojevi m_{ij} računaju se prvo za $l = j - i = 0$, zatim za $l = j - i = 1$, itd. tj. razlika $l = j - i$ raste tokom izvršavanja algoritma.

Algoritam. Algoritam zasnovan na dinamičkom programiranju za računanje minimalne cijene (m_{1n}) množenja niza od n matrica $M_1 \times M_2 \times \ldots \times M_n$. Ulaz. Brojevi $r_0, r_1, \ldots, r_n$ gdje matrica M_i ima dimenzije r_{i-1} i r_i. Izlaz. Minimalna cijena računanja proizvoda M, uzimajući da se potroši pqr operacija prilikom množenja matrice oblika $p \times q$ matricom oblika $q \times r$.

Metod. U nastavku je dato rješenje, tj. u nastavku je dat tekst programa na nestrogom paskalu:

FOR $i \leftarrow 1$ UNTIL n DO $m_{ii} \leftarrow 0$;
FOR $l \leftarrow 1$ UNTIL $n-1$ DO
    FOR $i \leftarrow 1$ UNTIL $n-l$ DO
    BEGIN
    $j \leftarrow i+l$;
    $m_{ij} \leftarrow \text{MIN}\{m_{ik} + m_{k+1,j} + r_{i-1}r_k r_j,\ i \leq k < j\}$
    END;
WRITE m_{1n}

Primijenimo algoritam na prethodni primjer $r_0, r_1, \ldots, r_4$. Djelimični rezultati i rezultat prikazani su u sljedećoj tabeli. Prvo se računaju brojevi iz prvog reda tabele, zatim oni iz drugog reda, itd. Rezultat je $m_{14} = 2.200$.

$m_{11} = 0$	$m_{22} = 0$	$m_{33} = 0$	$m_{44} = 0$
$m_{12} = 10.000$	$m_{23} = 1.000$	$m_{34} = 5.000$	
$m_{13} = 1.200$	$m_{24} = 3.000$		
$m_{14} = 2.200$			

Razmotrimo optimalni put koji vodi od čvora $x = 1$ do čvora $x = n$. Razmotrimo dio tog puta od $x = i$ do $x = j$. Taj dio predstavlja optimalno rješenje za podzadatak. Drugim riječima, svaki dio optimalnog puta je optimalni put. Važi i obrnuto: put koji je sastavljen sve iz optimalnih djelova jeste optimalni ukupni put. Posljednja rečenica predstavlja skicu za dokaz ispravnosti (za dokaz korektnosti) predloženog programa. Da program saopštava stvarno minimalnu cijenu.

Spoljašnja petlja po l ponavlja se reda n puta. Unutrašnja petlja po i ponavlja se reda n puta. Za jedno njeno ponavljanje treba izračunati jedno MIN. Za računanje jednog MIN treba reda n operacija. Tako $n \cdot n \cdot n = n^3$ ili $O(n) \cdot O(n) \cdot O(n) = O(n^3)$. Tako da predloženi program ima složenost reda n^3.

Mi ne možemo da smanjimo veličinu m_{1n}, niti da na nju utičemo na bilo kakav način. Mi želimo da tu veličinu saznamo na neki efikasan način (koji troši malo vremena), mi želimo da smanjimo vrijeme potrebno da se ta veličina sazna.

Predloženo rješenje treba dopuniti tako da saopštava (pored m_{1n}) i optimalni redosljed udruživanja.

3. UREĐIVANJE

Algoritmi za uređivanje (sortiranje) mogu da se podijele u dvije klase. U prvoj klasi koristi se struktura objekata koji se uređuju, recimo ti objekti su riječi nad određenom azbukom, v. naslov 3.2. U drugoj klasi osnovna operacija jeste upoređivanje dva objekta, da li je $a_1 \leq a_2$, algoritmi heapsort i quicksort.

3.1. POSTAVKA ZADATKA O UREĐIVANJU

Zadatak o uređivanju odnosi se na niz $a_1, a_2, \ldots, a_n$ čiji članovi pripadaju skupu S u kome je definisana relacija linearnog (potpunog) uređenja $\leq$.

Govorimo o unutrašnjem uređivanju ako se članovi niza nalaze upisani u memoriji sa slučajnim pristupom. Međutim, moguće je da su članovi niza smješteni u memoriji sa sekvencijalnim pristupom, tj. da su smješteni na spoljašnjoj memoriji (recimo na disku ili na magnetnoj traci). U ovom drugom slučaju govorimo o spoljašnjem uređivanju.

3.2. RADIKALNO UREĐIVANJE

U ovom naslovu razmatra se zadatak o uređivanju datog skupa riječi, odnosno razmatra se jedan algoritam za rješavanje tog zadatka. Prvo se definiše jedna relacija potpunog uređenja u skupu svih riječi nad određenom azbukom S, tzv. leksikografsko uređenje (tako su uređeni rječnici). Neka su $w_1 = s_1 s_2 \ldots s_p$ i $w_2 = t_1 t_2 \ldots t_q$ dvije riječi nad azbukom S. Kažemo da $w_1 \preceq w_2$ (č. prethodi) ako je ispunjen jedan od uslova a) i b) koji slijede. a) Za neko j je $s_j < t_j$ i za svako $i \in \{1, \ldots, j-1\}$ je $s_i = t_i$. b) $p \leq q$ i za svako $i \in \{1, \ldots, p\}$ je $s_i = t_i$. Vidimo da je relacija označena kao $\preceq$. Vidimo da se implicitno uzima da je definisan redosljed slova (članova azbuke). Primjeri: algebra $\preceq$ algoritam i algoritam $\preceq$ algoritamski, podrazumijeva se da je redosljed slova $a < b < c < \ldots < z$.

Prelazimo na opis algoritma. Uvedimo potrebne oznake. Neka azbuka S ima m članova, možemo smatrati da je $S = \{0, 1, \ldots, m-1\}$. Označimo sa W dati skup riječi. Neka W ima n članova. U ovom algoritmu pretpostavlja se da sve riječi $w \in W$ imaju jednu te istu dužinu (imaju jedan te isti broj slova). Neka je dužina jedne bilo koje riječi $w \in W$ označena sa k, $w = x_1 x_2 \ldots x_k$, gdje $x_i \in S$ za svako $i \in \{1, \ldots, k\}$.

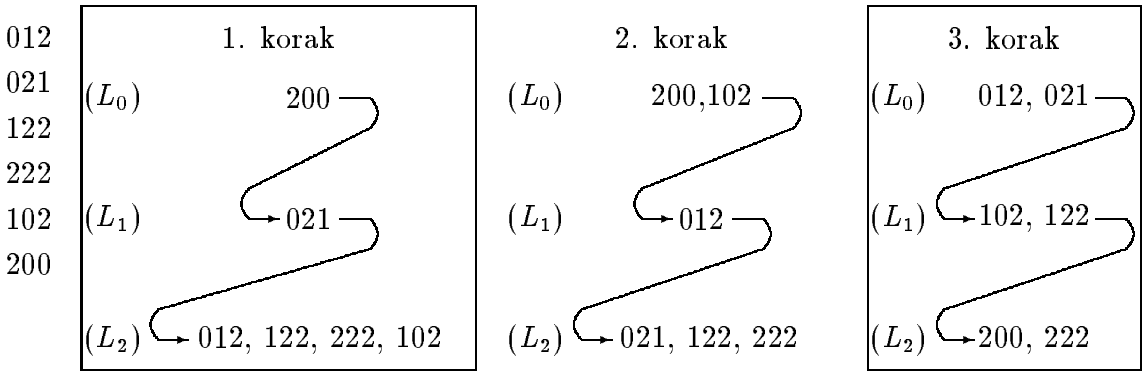
Na početku algoritma, sve date riječi upišu se u jednu veliku listu L. Na početku prvog koraka otvori se m listi $L_0, L_1, \ldots, L_{m-1}$, sve one su u datom trenutku prazne. Redom se čita jedan po jedan član liste L i upućuje se u jednu od listi $L_0, L_1, \ldots . L_{m-1}$. U prvom koraku, pročitana riječ w upućuje se u neku od m listi saglasno svom posljednjem slovu x_k. Drugim riječima, ako je $x_k = 0$ onda pošalji u L_0 i slično. Riječ w dodaje se na kraj njoj odgovarajuće liste, tj. upiše se iza svih riječi koje se već nalaze u toj listi. Razvrstava se na opisani način sve dok se L ne iscrpi. Sada se izvrši nadovezivanje djelimičnih listi $L_0, L_1, \ldots, L_{m-1}$ u jednu listu, upravo u veliku listu L opet. Naravno da se nadovezivanje uradi kao: prvo redom kako su u listi svi članovi liste L_0, zatim redom svi članovi liste L_1, itd. Sada je prvi korak algoritma završen. Opišimo drugi korak. Opet se otvori m praznih listi $L_0, L_1, \ldots, L_{m-1}$. Riječi se redom čitaju iz L i razvrstavaju se u djelimične liste, ovog puta saglasno svom pretposljednjem slovu x_{k-1}. Na kraju drugog koraka izvrši se nadovezivanje djelimičnih listi $L_0, L_1, \ldots, L_{m-1}$ slično kao u prethodnom koraku, da to sada bude lista L. U nastavku se postupa slično. U trećem koraku gleda se slovo x_{k-2} riječi. Algoritam ima ukupno k koraka. Kada se i k–ti korak završi onda će lista L da sadrži svih n datih riječi u leksikografskom redosljedu.

Programska realizacija. Za djelimične liste $L_0, L_1, \ldots, L_{m-1}$ može se preciznije reći da su redovi (queue), budući da za njih važi disciplina FIFO, first–in first–out.

Dokaz ispravnosti (dokaz korektnosti) izloženog algoritma jasan je iz njegove konstrukcije.

Za liste $L_0, L_1, \ldots, L_{m-1}$ koje se pojavljuju kaže se da su korpe. Za algoritme za uređivanje ovog tipa kaže se da oni vrše uređivanje pomoću korpi, bucket sort. Isto se kaže da oni vrše radikalno uređivanje ili uređivanje počev od korijena, jer se polazi od posljednjeg slova u riječi.

U nastavku je naveden primjer. Izabrali smo da bude $m = 3$, tj. da azbuka bude $S = \{0, 1, 2\}$. Izabrali smo da svaka riječ ima dužinu $k = 3$, tj. da bude oblika $w = x_1x_2x_3$. Izabrali smo da bude $n = 6$, treba šest riječi da se uredi.



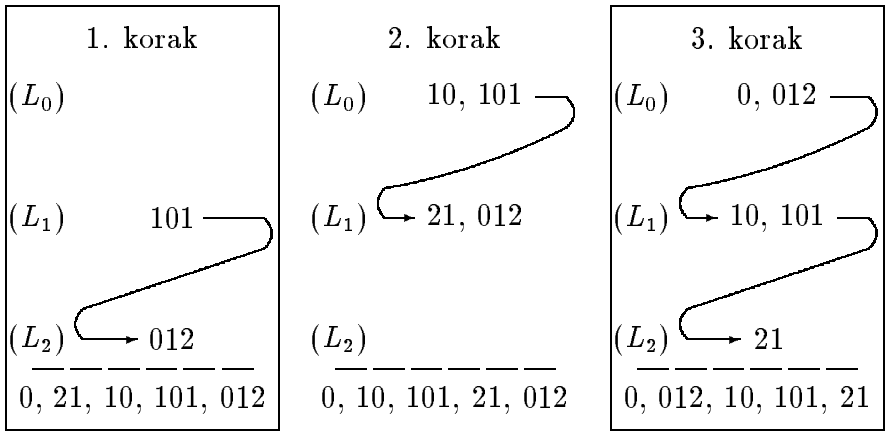
U nastavku ćemo se osloboditi uslova da je $|w|$ konstanta. Mi smo izložili algoritam za uređivanje riječi jednake dužine. Sada ćemo izložiti i drugi algoritam koji obavlja leksikografsko uređivanje riječi promjenljive dužine. Predstavlja malu modifikaciju prvog algoritma, pa izložimo ukratko.

Neka bude $l = \max_{w \in W} |w|$, $|w|$ znači dužinu ili broj slova riječi w. U pripremnom koraku algoritma date riječi urede se po rastućim dužinama, tako da će riječi čija dužina iznosi l da dođu na kraj liste L. U prvom koraku algoritma razvrstava se na osnovu slova x_l pojedine riječi. U tom koraku operiše se samo sa riječima w čija je dužina $|w| = l$. Napominje se da kraće riječi ostaju tokom ovog koraka na svojim mjestima u listi L. U drugom koraku razvrstava se na osnovu slova x_{l-1}, s tim da se operiše samo sa riječima w čija je dužina $|w| \geq l - 1$. Napominje se da riječi kraće od $l - 1$ slova ostaju tokom ovog koraka na svojim mjestima u listi L. Itd. Ukupno ima l koraka.

U nastavku je naveden primjer za drugi algoritam.

data lista: 101, 012, 0, 21, 10

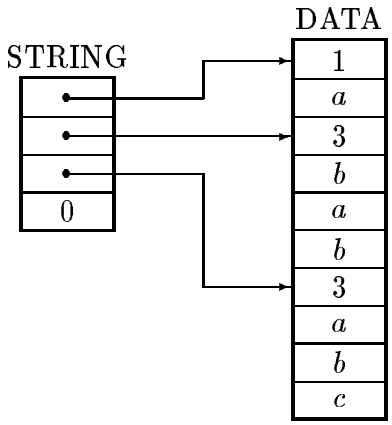
pripremni korak: 0, 21, 10, 101, 012



rezultat: 0, 012, 10, 101, 21

Neka bude $w = x_1x_2\ldots x_j$ i $j < l$. Mi možemo da riječ w dopunimo do dužine l, da na kraju riječi dodamo $l-j$ slova α, tj. da w pretvorimo u $x_1x_2\ldots x_j\underbrace{\alpha\ldots\alpha}_{l-j}$. Sa ovom izmjenom, drugi algoritam se svodi na prvi algoritam. Ovdje je α fiktivno slovo takvo da važi $\alpha < a < b < c < \ldots < z$.

Kakve strukture podataka su pogodne za držanje ulaznih podataka prvog, odnosno drugog algoritma? Obična struktura podataka za predstavljanje niza dužine n riječi, gdje dužina svake riječi iznosi k slova, jeste matrica oblika $n \times k$ čiji su članovi tipa `char`. Na sljedećoj slici skicirana je jedna moguća struktura podataka za predstavljanje niza riječi promjenljive dužine (riječi koje pripadaju nizu imaju svaka svoju dužinu). Skica se odnosi na slučaj tri riječi: a, bab i abc.



3.3. UREĐIVANJE POMOĆU UPOREĐIVANJA (POMOĆU POREĐENJA)

Neka je $n \geq 1$ i neka je dato n brojeva $x_1, x_2, \ldots, x_n$. Date brojeve treba urediti po veličini. U algoritmu za rješavanje tog zadatka, elementarnu operaciju predstavlja (po pravilu) operacija upoređivanja: da li je $a < b$. Tako da se složenost $T = T(n)$ takvog algoritma A definiše kao broj izvršenih operacija upoređivanja. Ima se u vidu najgori slučaj: $T(n)$ je maksimalni broj izvršenih operacija, po svim primjercima zadatka čiji je dimenzioni broj jednak n. U ovom naslovu funkcija $T(n)$ biće ocijenjena sa donje strane: mi ćemo naći konkretnu funkciju $T_0 = T_0(n)$ takvu da važi

$$T(n) \geq T_0(n)$$

za svaki prirodan broj n, gdje $T = T(n)$ označava složenost ma kog algoritma oblika A.

Za pripremu, poznata je Stirlingova formula koja daje dobru procjenu za članove niza $a_n = n!$ kako za bilo koje $n \in N$ tako isto i kada $n \to \infty$. Formula se može prikazati u tri različita oblika, kako slijedi, svaki idući oblik je precizniji, odnosno daje više informacija od prethodnog oblika:

$$n! \geq \left(\frac{n}{e}\right)^n, \quad n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n, \quad n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\theta_n/12n},$$

gdje je $0 < \theta_n < 1$. U sva tri oblika: $n \in N$. Mi ćemo koristiti prvi oblik.

Vratimo se algoritmu A. S jedne strane, mogućih rasporeda ili permutacija datih brojeva $x_1, x_2, \ldots, x_n$ ima $n!$ a samo jedan od tih rasporeda je pravilan, samo u jednoj permutaciji su brojevi poređani po veličini, u slučaju da su dati brojevi međusobno različiti. S druge strane, poslije jedne operacije poređenja imamo najviše dvije grane ili dvije mogućnosti, poslije dvije operacije najviše četiri grane, poslije tri najviše osam, itd. Vidimo kakva je zavisnost između broja poređenja $T(n)$ i broja mogućnosti. Jasno je da broj mogućnosti mora da bude $\geq a_n = n!$

pa tim prije mora biti $2^{T(n)} \geq a_n$. Govoreći prostije, mi se pitamo: koliko daleko treba ići u nizu $1, 2, 4, 8, 16, 32, \ldots$ da član niza premaši a_n.

Izvedimo posljednju formulu $2^{T(n)} \geq a_n$ i na drugi način. Ranije je rađeno *IV*. Drveta odlučivanja. Algoritmu oblika A odgovara jedno drvo odlučivanja D. To je binarno drvo čiji svaki vrh ima dva sina, osim ako je taj vrh – list. Pojedini list predstavlja jedan mogući odgovor za zadatak o uređivanju, odnosno pojedinom listu odgovara permutacija u kojoj su dati brojevi poređani po veličini. Tako da D mora da ima bar $a_n = n!$ listova. Označimo sa $h = h(D)$ visinu drveta D. Lako se vidi da je $T(n) = h(D)$. Lako se vidi da postoji sljedeći odnos između h i l: $l \leq 2^h$, gdje l označava broj listova u drvetu. Imamo:

$$T(n) = h \quad 2^h \geq l \quad l \geq n! \quad \Rightarrow \quad 2^{T(n)} \geq n!$$

$$2^{T(n)} \geq n! \quad \Big/ \log_2$$

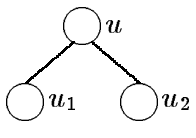
$$T(n) \geq \log_2 n!$$

$$\text{Stirling} \quad \Rightarrow \quad T(n) \geq \log_2 \left(\frac{n}{e}\right)^n = n \log_2 n - n \log_2 e$$

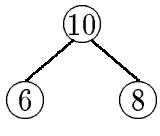
Prema tome, možemo staviti da je $T_0(n) = n \log_2 n - n \log_2 e$. Znamo da je $\log_2 e = 1{,}44$. Vidimo da važi $T_0(n) \sim n \log_2 n$ kad $n \to \infty$.

3.4. HEAPSORT

Razmotrimo zadatak o uređivanju n datih brojeva $x_1, \ldots, x_n$, gdje $x_i \in Z$ ili $x_i \in R$ za $1 \leq i \leq n$. U ovom naslovu predlaže se algoritam heapsort za rješavanje tog zadatka, heap – gomila, č. hip. Prethodno, šta je to heap i kada se za heap kaže da je sređen. Razmotrimo potpuno binarno drvo koje ima dovoljnu visinu, odnosno čiji je broj vrhova $\geq n$. Njegove vrhove $v_1, v_2, \ldots$ pogodno je označiti tako da bude: v_1 – korijen drveta, v_2 i v_3 – lijevi, odnosno desni sin korijena, v_4 – lijevi sin od v_2, itd. Razmotrimo poddrvo koje čine vrhovi $v_1, \ldots, v_n$. To poddrvo i jeste heap H_n. Neka je sada svakom vrhu drveta v_i $(1 \leq i \leq n)$ pridružen po jedan cio ili realan broj $\ell(v_i) = x_i$. U vezi buduće programske realizacije, podaci o pridruženim brojevima mogu da se čuvaju u jednom nizu A čiji se indeksi kreću od 1 do n i čiji su članovi tipa INTEGER ili tipa REAL, $A[i] = \ell(v_i)$ za $1 \leq i \leq n$. Za vrh v_i, njegov lijevi i desni sin su na poziciji $2i$ odnosno $2i + 1$. Slično, za vrh j njegov otac je u nizu A na poziciji $[j/2]$. Neka je u ma koji vrh heapa i neka su u_1 i u_2 njegovi lijevi, odnosno desni sin (ako postoje). Ako je ispunjen par uslova $\ell(u) \geq \ell(u_1)$ i $\ell(u) \geq \ell(u_2)$ onda se kaže da je heap sređen na mjestu u. Ako je heap sređen na svakom mjestu onda se kaže da je heap čitav sređen. Lako se vidi da sređeni heap posjeduje svojstva 1)–3) kako slijedi. 1) Ako je $u_1, u_2, \ldots, u_j$ bilo koji niz vrhova heapa, gdje je u_1 korijen, u_{i-1} je otac za u_i $(2 \leq i \leq j)$ i u_j je list onda važi $\ell(u_1) \geq \ell(u_2) \geq \ldots \geq \ell(u_j)$. 2) Za bilo koji vrh, posmatrajmo sve vrhove u poddrvetu čiji je on korijen (sve vrhove u njegovom poddrvetu). Tada je najveći među brojevima $\ell(u)$, po svim vrhovima u iz poddrveta, pridružen upravo korijenu poddrveta. 3) Među svim pridruženim brojevima po čitavom drvetu, najveći je broj koji je pridružen korijenu drveta. Lako se vidi da je 3) specijalan slučaj od 2). Prelazimo na opis algoritma. Dati brojevi $x_1, \ldots, x_n$ na početku se rasporede na slučajan način po drvetu H_n: neka bude $\ell(v_i) = x_i$ za svako $i \in \{1, \ldots, n\}$. Tokom izvršavanja algoritma, vrše se određena premještanja ili zamjene mjesta pridruženih brojeva, da bi se na kraju dobio uređeni oblik niza od n brojeva koji je na početku bio dat. U prvom koraku algoritma vrše se premještanja, da bi heap postao sređen. Heap će postati sređen kada se izvrše sljedeće radnje: izvršiti sređivanje za vrh v_n, zatim za vrh v_{n-1}, itd. na kraju izvršiti sređivanje za korijen drveta v_1, tim redom.



$\ell(u) = 10$
$\ell(u_1) = 6$
$\ell(u_2) = 8$



A šta znači izvršiti sređivanje za jedan vrh u drvetu u? Ova radnja sa svoje strane razlaže se na više manjih radnji. Upravo, prvo se izvrši sređivanje na mjestu u. Upravo i bliže, ispita se da li važi par nejednakosti $\ell(u) \geq \ell(u_1)$ i $\ell(u) \geq \ell(u_2)$, gdje su u_1 i u_2 lijevi, odnosno desni sin vrha u. Ako je tako onda je sređivanje za vrh u završeno. Ako nije tako onda treba izvršiti pogodnu zamjenu mjesta tri pridružena broja $\ell(u)$, $\ell(u_1)$ i $\ell(u_2)$ da poslije toga bude istinito $\ell(u) \geq \ell(u_1)$ i $\ell(u) \geq \ell(u_2)$. Na mjesto u kao rezultat premještanja dođe broj iz u_1 ili u_2. U slučaju da su $\ell(u)$ i $\ell(u_1)$ zamijenili mjesta, u nastavku treba izvršiti sređivanje za vrh u_1. A u slučaju da su $\ell(u)$ i $\ell(u_2)$ zamijenili mjesta, u nastavku treba izvršiti sređivanje za vrh u_2, na sličan način. Itd. Sve dok se tako ne stigne do nekog lista drveta. Kada se stigne do lista, tada je sređivanje za vrh u okončano. Uočava se sljedeće. Od vrha u do lista prođe se više vrhova. Može se desiti da je za neki od tih prolaznih vrhova bila sama po sebi istinita i jedna i druga nejednakost, za taj vrh ne vrši se premještanje. U tom slučaju, sređivanje za vrh u već u tom trenutku je okončano, nije potrebno da se dalje ide do lista. Ovo će se uočiti iz konstrukcije algoritma. Pogledajmo drugu stvar, u vezi budućeg računanja složenosti algoritma. Koliko treba potrošiti operacija poređenja c da bi se obavilo sređivanje na jednom mjestu. Dovoljno je

$c = 2$. Zaista, uporedi lijevog sina protiv desnog sina, a onda uporedi većeg od njih protiv oca.

Rekli smo da se prvi korak algoritma sastoji od sređivanja za vrhove v_i, gdje je $i = n, n-1, \ldots, 1$. Ako je $i > n/2$ onda je v_i list. Za list nema šta da se sređuje. Dakle, može se reći $i = [n/2], \ldots, 1$.

Pri kraju prvog koraka, heap H_n je sređen. Najveći broj pridružen je korijenu. Taj najveći broj saopštava se na izlaz, $\ell(v_1)$ šalje se na izlaz. Pored toga, $\ell(v_1) \leftarrow \ell(v_n)$, sada se korijenu pridružuje broj koji je dosad bio pridružen vrhu v_n. Tako da je posljednji vrh v_n postao suvišan, taj vrh uklanja se iz drveta. Tako da se sada prelazi na novi heap H_{n-1}.

Opišimo drugi korak algoritma. Okolnosti su povoljnije nego u prvom koraku. Heap H_{n-1} već je donekle sređen, tj. nastao je malim remećenjem u sređenom heapu H_n. Želimo da i H_{n-1} postane sređen. Dovoljno je da se izvrši sređivanje za njegov korijen v_1, tj. sada nije potrebno da se vrši sređivanje za vrhove $i = n-1$, ..., $i = 2$. Dakle, za drugi korak, pređe se put od korijena pa naprijed, dok se negdje ne izađe do lista, a tokom puta se ispituje da li treba vršiti premještanja. Ako se na određenom mjestu ne premješta onda i ne treba dalje ići prema listovima. Pri kraju drugog koraka, heap H_{n-1} je sređen. Najveći broj u njemu pridružen je korijenu i taj najveći broj $\ell(v_1)$ sada se šalje na izlaz. Još se uradi $\ell(v_1) \leftarrow \ell(v_{n-1})$ i zatim se vrh v_{n-1} izbaci iz drveta, tako da se prelazi na novi heap H_{n-2} koji je još kraći. Time je okončan drugi korak algoritma. Iskoristimo priliku da još nešto kažemo o programskoj realizaciji. Kada se sa H_n prelazi na H_{n-1}, u tom istom trenutku se jedan broj saopštava na izlaz. Za H_n su potrebni $A[i]$, gdje je $1 \leq i \leq n$. Za H_{n-1} su potrebni $A[i]$, gdje je $1 \leq i \leq n-1$. U nizu A oslobađa se mjesto $A[n]$. U oslobođeno mjesto upiši broj koji se saopštava na izlaz. Slično kada se drugi broj saopštava na izlaz, upiši ga u $A[n-1]$. Itd. Dakle, čitava programska realizacija uradi se lako i efikasno pomoću jednog niza A.

Treći korak algoritma je po svemu sličan drugom koraku. Itd. Algoritam ukupno ima n koraka. Ako se brojevi koji se šalju na izlaz zapisuju redom jedan za drugim onda oni (tako zapisani) upravo i čine uređeni oblik ulaznog niza brojeva $x_1, \ldots, x_n$. Time je algoritam heapsort opisan. Dokaz ispravnosti algoritma sadržan je u dosadašnjem tekstu, odnosno jasan je iz njegovog izvođenja (jasan je iz njegove konstrukcije).

Tekst glavnog programa:

```
BUILDHEAP;
FOR i ← n STEP −1 UNTIL 2 DO
{ BEGIN
  zamijeni A[1] i A[i];
  HEAPIFY(1, i − 1)
  END
```

Tekst prvog potprograma:

```
PROCEDURE HEAPIFY(i, j);
IF i nije list u H_j & neki sin od i sadrži
veći broj nego što ga sadrži i THEN
{ BEGIN
  neka je k sin od i sa većim brojem;
  zamijeni A[i] i A[k];
  HEAPIFY(k, j)
  END
```

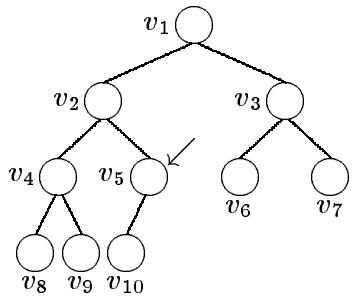
Tekst drugog potprograma:

PROCEDURE BUILDHEAP;
FOR $i \leftarrow n$ STEP -1 UNTIL 1 DO HEAPIFY(i, n)

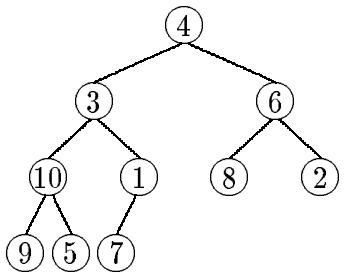
U nastavku: program za heapsort na nestrogom paskalu, primjer i računanje složenosti algoritma.

U programu na nestrogom paskalu koristi se oznaka n za broj objekata koji su predmet uređivanja i koristi se oznaka A za niz koji sadrži te objekte. Pored glavnog programa imamo i dva potprograma HEAPIFY i BUILDHEAP. Potprogram HEAPIFY vrši sređivanje za jedan vrh. On ima dva parametra i i j čiji je smisao kako slijedi: i – vrši se sređivanje za vrh v_i, j – u datom trenutku u nizu A j je krajnji desni indeks koji se odnosi na heap, veći indeksi ne odnose se na heap. Drugim riječima, u datom trenutku radi se sa drvetom H_j. Vidi se da je potprogram HEAPIFY rekurzivan: ako je došlo do zamjene onda istu proceduru treba ponoviti za $i = k$. Drugi potprogram BUILDHEAP odgovara prvom koraku algoritma. Dakle, pomoću njega se prvi put postigne da heap postane čitav sređen. Vidimo da je u BUILDHEAP uzeto da početna vrijednost promjenljive i bude $i = n$. Može se staviti da ta početna vrijednost bude jednaka $i = [n/2]$, kao što je ranije već rečeno. Može se reći da se pomoću BUILDHEAP izvrši opšte uređivanje heapa.

prije početka izvršavanja:

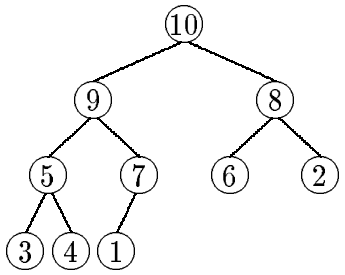


na početku prvog koraka:

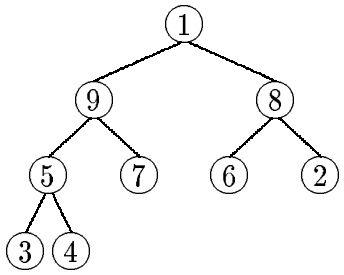


tokom prvog koraka:
za v_5: 7 i 1 zamijene mjesta
za v_4: ništa se ne mijenja
za v_3: 8 i 6 zamijene mjesta
za v_2: 10 i 3 zamijene mjesta
9 i 3 zamijene mjesta
za v_1: 10 i 4 zamijene mjesta
9 i 4 zamijene mjesta
5 i 4 zamijene mjesta

na kraju prvog koraka:

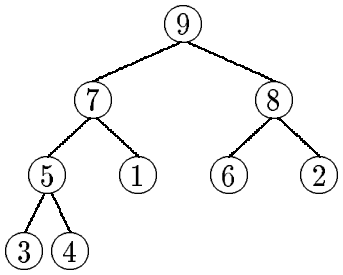


na početku drugog koraka:



na izlazu: 10

na kraju drugog koraka:



na izlazu: 10

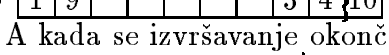
na početku trećeg koraka:
heap: ...
na izlazu: $9, 10$

poslije kraja izvršavanja:
heap: nema
na izlazu: $1, 2, \ldots, 10$

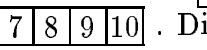
U primjeru je uzeto $n = 10$. Tako da drvo ima vrhove od v_1 do v_{10}, s tim da je v_1 korijen. Na prvoj slici prikazan je izgled drveta, a u vrh v_5 uperena je strelica. Zato što se u prvom koraku algoritma vrši sređivanje za vrhove v_5, ..., v_1, tim redom. U drugom, trećem, itd. koraku algoritma vrši se sređivanje samo za jedan vrh (samo za korijen v_1). Uzeto je da dati brojevi koji treba da budu uređeni glase, odnosno uzeto je da početno stanje u nizu A glasi

4	3	6	10	1	8	2	9	5	7

. Drugim riječima, uzeli smo da je $x_1 = 4$, $x_2 = 3$, ...,

$x_{10} = 7$. Na sljedećim slikama prikazano je stanje u strukturi podataka u nekim karakterističnim trenucima tokom izvršavanja algoritma, odnosno prikazani su izgled drveta i stanje na izlazu. Recimo, na kraju prvog koraka u nizu A imaćemo 

. Na kraju drugog koraka imaćemo 

. A kada se izvršavanje okonča biće 

. Dio niza A koji se odnosi na heap je lijevo od znaka } a desno od tog znaka je dio niza A koji se odnosi na ono što je poslato na izlaz.

Označimo sa $T(n)$ složenost algoritma, odnosno (najveći mogući) broj izvršenih operacija poređenja. Želimo da odredimo niz a_n takav da važi $a_n \sim T(n)$ kad $n \to \infty$. Označimo sa $T_1(n)$ i $T_2(n)$ procjene vremenskog troška za prvi korak algoritma, odnosno za sve ostale korake algoritma zajedno. Neka su koraci numerisani kao $i = 1, \ldots, i = n$. Za korake od $i = 2$ do $i = n$, tj. za $T_2(n)$ imamo sljedeće. U pojedinom koraku treba izvršiti sređivanje za korijen. Broj elementarnih operacija za jedno takvo sređivanje jednak je trenutnoj visini drveta. Neka bude $k = \log_2 n$. Za visinu imamo sljedeću dobru procjenu: tokom prvih $n/2$ koraka visina je jednaka k, tokom idućih $n/4$ koraka visina je jednaka $k-1$, $n/8$ $k-2$, itd. Još, cijena jedne elementarne operacije iznosi $c = 2$, kao što je ranije utvrđeno. Za $T_2(n)$ pišemo:

$$\left(\frac{n}{2} \cdot k + \frac{n}{4} \cdot (k-1) + \frac{n}{8} \cdot (k-2) + \ldots\right) \cdot c \leq$$

$$\left(\frac{n}{2} \cdot k + \frac{n}{4} \cdot k + \frac{n}{8} \cdot k + \ldots\right) \cdot c = nk \cdot c = 2nk = T_2(n)$$

Za $T_1(n)$, treba izvršiti sređivanje za vrhove $i = [n/2], \ldots, i = 1$, tim redom. Ostali vrhovi su listovi, tako da pojedinom takvom vrhu sljeduje 0 elementarnih operacija. Ima približno $n/4$ vrhova koji su skoro–listovi (koji su na jednu ivicu od lista), pojedinom takvom vrhu sljeduje 1 elementarna operacija. Slično, približno $n/8$ 2. Itd. Za jednu elementarnu operaciju $c = 2$ upoređivanja. Prema tome, za $T_1(n)$ pišemo:

$$\left(\frac{n}{4} \cdot 1 + \frac{n}{8} \cdot 2 + \frac{n}{16} \cdot 3 + \ldots\right) \cdot c =$$

$$\left(\left(\frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \ldots\right) + \left(\frac{n}{8} + \frac{n}{16} + \ldots\right) + \left(\frac{n}{16} + \ldots\right) + \ldots\right) \cdot c =$$

$$\left(\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \ldots\right) \cdot c = n \cdot c = 2n = T_1(n)$$

Znamo da je $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \ldots = 2$. Konačno:

$T(n) \sim a_n$

$a_n = T_1(n) + T_2(n)$

$T_1(n) + T_2(n) = 2n + 2nk = 2n + 2n \log_2 n \sim 2n \log_2 n$

$T(n) \sim 2n \log_2 n$

Uporediti sa teorijskom donjom granicom složenosti $n \log_2 n$ koja je utvrđena u prethodnom naslovu. Tako da imamo sljedeći zaključak. Složenost algoritma heapsort malo odstupa od teorijske granice. Algoritam je vrijedan.

3.5. QUICKSORT

Neka je dato n brojeva $x_1, \ldots, x_n$, gdje je $n \geq 1$. Razmotrimo zadatak o uređivanju datog niza brojeva. Za rješavanje tog zadatka u ovom naslovu predlaže se algoritam quicksort, quick – brz. Opišimo algoritam. Iz datog niza brojeva S uzmemo na slučajan način jedan njegov član x. Zatim se svaki broj x_i $(1 \leq i \leq n)$ uporedi sa x. Biće $x_i < x$ ili $x_i = x$ ili $x_i > x$. Svi članovi x_i koji su $< x$, $= x$ ili $> x$ upišu se u djelimični niz S_1, S_2, odnosno S_3. Uvedimo oznake za dužine podnizova (za broj članova u podnizu): $|S_1| = n_1$, $|S_2| = n_2$ i $|S_3| = n_3$. Očito je $n_1 + n_2 + n_3 = n$. U podnizu S_2, svi njegovi članovi su međusobno jednaki (jednaki su x), tako da u tom podnizu više nema šta da se uređuje. Ako bi podnizovi S_1 i S_3 bili uređeni onda bi čitav posao oko uređivanja već bio okončan, odnosno bili bismo u mogućnosti da saopštimo odgovor. Upravo, odgovor glasi: prvo članovi iz S_1 (članovi iz uređenog oblika podniza S_1), za njima članovi od S_2 a zatim nadovezati još i članove od S_3 (članove iz uređenog oblika od S_3). Jedino još treba da se kaže na koji način će biti uređeni S_1 i S_3. Odgovor je: na sličan način, istim postupkom. Drugim riječima, algoritam quicksort je rekurzivan. Time je algoritam opisan. U nastavku je dat tekst programa (tekst algoritma) na nestrogom paskalu:

Tekst glavnog programa:

ulazne podatke $x_1, \ldots, x_n$ upiši u niz S;
QUICKSORT(S);
štampaj članove niza S

Tekst potprograma:

PROCEDURE QUICKSORT(S);
IF $|S| \leq 1$ THEN RETURN S;
izaberi iz S na proizvoljan način član x;
formiraj podnizove S_1, S_2 i S_3 niza S koje čine članovi koji su $< x$, $= x$, odnosno $> x$;
$S_1 \leftarrow$ QUICKSORT(S_1);
$S_3 \leftarrow$ QUICKSORT(S_3);
$S \leftarrow$ članovi od S_1, zatim članovi od S_2 i na kraju članovi od S_3;
RETURN S

Prelazimo na računanje složenosti algoritma. Neka $T(n)$ znači broj izvršenih operacija poređenja. Razlikujemo dva slučaja.

Očekivani slučaj (prosječna složenost). Prvo treba da uvedemo pretpostavku o raspodjeli brojeva koji čine dati niz S. Uvedimo prirodnu pretpostavku koja se i sama po sebi podrazumijeva: broj x_i može da bude sa jednakom vjerovatnoćom bilo koji broj sa realne ose, vrijednost x_i ne utiče na x_j. Znamo da S ima n članova (znamo da je $|S| = n$). Koliko članova ima S_2, tj. čemu je jednako $n_2 = |S_2|$? Matematičko očekivanje za veličinu n_2 očito je jednako 1 (srednja vrijednost za n_2). Drugim riječima, vjerovatnoća događaja $\{n_2 > 1\}$ jednaka je 0. Napominje se da prisustvo međusobno jednakih članova u datom nizu brojeva samo olakšava posao oko uređivanja, odnosno samo smanjuje složenost algoritma. Dakle, $n_2 = |S_2| = 1$. A čemu su jednaki $|S_1| = n_1$ i $|S_3| = n_3$? Za n_1 postoji n mogućnosti. Upravo, $n_1 = 0$ ili $n_1 = 1$ ili $\ldots$ ili $n_1 = n - 1$. Onda se n_3 izračuna iz jednakosti $n_1 + n_3 = n - 1$. Za veličinu n_1 postoji n mogućnosti, mogućnosti su međusobno jednako–vjerovatne (zbog uvedene pretpostavke). Tako da je vjerovatnoća pojedine mogućnosti jednaka $\frac{1}{n}$. Numerišimo mogućnosti kao $i = 1, 2, \ldots, n$, gdje i–toj mogućnosti odgovara $n_1 = i - 1$ i $n_3 = n - i$. Ako je nastupila i–ta mogućnost onda je

vremenski trošak izvršavanja algoritma quicksort jednak $n+T(i-1)+T(n-i)$. Prvi sabirak n odnosi se na razvrstavanje u tri podniza, drugi sabirak $T(i-1)$ odnosi se na uređivanje podniza S_1, a treći sabirak $T(n-i)$ napisan je na račun troška za uređivanje podniza S_3. Dok je $T(n)$ jednako srednjoj vrijednosti tog zbira:

$$T(n) = \frac{1}{n}\sum_{i=1}^{n}\Big(n + T(i-1) + T(n-i)\Big).$$

Sada je niz $\{T(n)\}_{n=1}^{\infty}$ definisan, jer stavljamo $T(0) = T(1) = 0$. Treba riješiti rekurentnu relaciju, sami za vježbu. Može se pokazati da važi $T(n) \leq 2n\ln n$ za $n \in N$. Dakle, očekivani slučaj (average case): $T(n) = O(n\log_2 n)$ kad $n \to \infty$.

Najgori slučaj. Neka sada $T(n)$ označava složenost algoritma po kriterijumu najgoreg slučaja. Ostaje $n_2 = 1$. Lako se vidi da će najviše posla oko uređivanja biti ako se ispostavi da je $n_1 = 0$ i samim tim $n_3 = n-1$ ili obrnuto. Tako da je $T(n) = n + T(n-1)$. Ako prilikom svakog dijeljenja podniz S_1 ili podniz S_3 ostane prazan onda nastupa najgori slučaj. Računamo:

$$T(n) = n + n - 1 + T(n-2) = n + (n-1) + (n-2) + \ldots + 1 = \frac{n(n+1)}{2},$$

$$T(n) \sim \frac{1}{2}n^2 \quad \text{kad} \quad n \to \infty.$$

Dakle, najgori slučaj (worst case): $T(n) = O(n^2)$ kad $n \to \infty$.

Izvršimo dogradnju opisanog algoritma. Izdvajanje podniza S_2 (čiji su svi članovi jednaki x) kao posebnog podniza ne doprinosi efikasnosti algoritma. Pripojiti članove podniza S_2 podnizu S_1. Dakle, ispitivali bismo da li važi $x_i \leq x$.

Izvršimo još jednu dogradnju. Kako odrediti x? Prva mogućnost: izdvoj iz S uzorak, odnosno izdvoj iz S jedan mali podniz dužine recimo pet članova. Neka x bude medijana tog kratkog podniza (treći po veličini u njemu). Druga mogućnost: neka x bude aritmetička sredina kratkog podniza, sada može da prestane da bude $x \in S$.

Dva algoritma heapsort i quicksort služe za rješavanje jednog te istog zadatka. Po kriterijumu najgoreg slučaja, heapsort ima složenost reda $n\log_2 n$, a quicksort reda n^2, heapsort ima manju složenost. Tako da bismo rekli da je heapsort bolji od quicksort. Zapaziti da quicksort ima složenost reda veličine $n\log_2 n$ po kriterijumu očekivanog slučaja. U prilog algoritmu quicksort idu sljedeće dvije činjenice. 1) Kriterijum očekivanog slučaja daje realniju predstavu o vrijednosti algoritma od kriterijuma najgoreg slučaja. 2) Quicksort ima vrlo jednostavnu programsku realizaciju. Zaključak: u praksi se najviše koristi quicksort, za uređivanje brojeva, riječi i drugog.

3.6. ORDER STATISTICS

U ovom naslovu razmatra se zadatak o određivanju k–tog najmanjeg člana nekog niza. Drukčije se kaže zadatak o selekciji ili statistika poretka, engl. order statistics. Pogledajmo mali primjer. Neka bude $n = 5$ i $x_1, \ldots, x_5 = 10, 50, 40, 20, 30$. Ako je $k = 2$ onda je rezultat $x = 20$. Ako je $k = 3$ onda je rezultat $x = 30$, za $x = 30$ još se kaže i da predstavlja medijanu datog niza.

Pogledajmo dvije jednostavne definicije. Neka bude $n \geq 1$ i $k \geq 1$. Neka je S niz brojeva $x_1, x_2, \ldots, x_n$. Kaže se da je član niza x k–ti po veličini u nizu ako je $\text{card}\{i|\, x_i < x\} = k-1$ i

$\mathrm{card}\{i|\, x < x_i\} = n - k$. Ovo u slučaju da su članovi niza svi međusobno različiti dva–po–dva, a slično se definiše i kada među članovima niza ima ponavljanja. Razumije se da je $1 \leq k \leq n$. Kaže se da je član niza x srednji po veličini u nizu ili se kaže da je on medijana niza ako je x k–ti po veličini u nizu, gdje je $k = [n/2] + 1$ u slučaju neparnog n, odnosno $k = [n/2]$ ili $k = [n/2] + 1$ u slučaju parnog n. Znamo da se za niz $y_1, y_2, \ldots, y_n$ kaže da predstavlja uređeni oblik niza $x_1, x_2, \ldots, x_n$ ako je $y_1, y_2, \ldots, y_n$ jedna permutacija od $x_1, x_2, \ldots, x_n$ i ako je još $y_1 \leq y_2 \leq \ldots \leq y_n$.

Razmotrimo dva algoritma A_1 i A_2 za rješavanje zadatka o statistici poretka. Kao mjera složenosti algoritma $T(n)$ uzima se broj operacija poređenja i to se ima u vidu najgori slučaj. Obični algoritam A_1 predviđa kako slijedi. Uredi dati niz i onda pročitaj k–ti član uređenog oblika. Za složenost imamo $T(n) = O(n \log_2 n)$, jer je složenost algoritma A_1 očito jednaka složenosti algoritma primijenjenog za uređivanje niza. U ovom naslovu biće konstruisan algoritam A_2 čija je složenost jednaka $T(n) = O(n)$. Riječima, složenost algoritma A_2 je linearna. Potprogram SELECT definiše algoritam A_2. Slijedi tekst potprograma na nestrogom paskalu. SELECT vrši selekciju k–tog po veličini u nizu S koji ima n članova:

$L1$ PROCEDURE SELECT(k, S);
$L2$ IF $|S| < 50$ THEN BEGIN uredi S; RETURN k–ti član iz uređenog oblika od S END;
$L3$ podijeli S na nizove od po 5 članova ($p \leq 4$ člana niza S ostaju van podjele);
$L4$ uredi svaki petočlani podniz;
$L5$ pročitaj medijanu svakog petočlanog niza i obrazuj od pročitanih niz M;
$L6$ $m \leftarrow$ SELECT(a, M), gdje je $a = [b/2] + 1$, gdje je $b = |M|$;
$L7$ uporedi svaki $x \in S$ protiv m i tako obrazuj nizove S_1, S_2, S_3 ($x < m$, $x = m$ i $x > m$);
$L8$ $n_1 \leftarrow |S_1|$; $n_2 \leftarrow |S_2|$; $n_3 \leftarrow |S_3|$ (tako da je $n_1 + n_2 + n_3 = n = |S|$);
$L9$ IF $1 \leq k \leq n_1$ THEN rezultat $\leftarrow$ SELECT(k, S_1);
$L10$ IF $n_1 + 1 \leq k \leq n_1 + n_2$ THEN rezultat $\leftarrow m$;
$L11$ IF $n_1 + n_2 + 1 \leq k \leq n_1 + n_2 + n_3$ THEN rezultat $\leftarrow$ SELECT$(k - n_1 - n_2, S_3)$;
$L12$ RETURN rezultat

Objašnjenja za tekst potprograma. Ako je $n < 50$ onda se primijeni obični postupak (A_1). U slučaju $n \geq 50$ radi se kako slijedi. Kratkih petočlanih nizova ima očito $[n/5]$. Medijana petočlanog niza je naravno treći po veličini u nizu. Od tih medijana obrazuje se niz M. A m je medijana niza M. Ako je $x < m$ onda se x šalje u S_1 i slično. Naravno da i članovi niza koji su bili ranije privremeno odbačeni (ima ih p) učestvuju sada u ovom razvrstavanju svih članova niza S u tri djelimična niza S_1, S_2 i S_3. Očito je $0 \leq p \leq 4$. Sljedeći korak algoritma zavisi od odnosa veličina k, n_1 i n_2. Zapaziti da niz S_1 nije uređen, ali znamo da su svi njegovi članovi manji od m. U nizu S_2, svi članovi su jednaki m. Niz S_3 nije uređen, svi njegovi članovi su veći od m. Postoje tri mogućnosti za sljedeći korak. Primjera radi, treća mogućnost nastupa ako je $k > n_1 + n_2$. Tada treba tražiti u nizu S_3, ali sa prilagođenim startnim brojem. Zaista, k–ti u S je ℓ–ti u S_3, gdje je $\ell = k - n_1 - n_2$. Naime, uređeni oblik niza S očito jeste: uređeni oblik niza S_1 plus niz S_2 plus uređeni oblik niza S_3.

Prelazimo na računanje složenosti $T(n)$ algoritma A_2. Smatraćemo da su u datom nizu svi njegovi članovi međusobno različiti. Naime, prisustvo jednakih članova samo bi olakšalo posao oko rješavanja postavljenog zadatka, odnosno samo bi smanjilo složenost.

Razmotrimo primjer $n = 105$. Tada M ima 21 član i m je 11–ti po veličini u M. Tada će sigurno biti $32 \leq n_1 \leq 72$ i isto tako $32 \leq n_3 \leq 72$. Zaista, imamo sljedeću šemu:

$\alpha_i < \beta_i < \gamma_i < \delta_i < \varepsilon_i$ za $i = 1, \ldots, 21$, $\gamma_1 < \gamma_2 < \ldots < \gamma_{21}$, $\gamma_{11} = m$.

Među n brojeva $\alpha_1, \ldots, \varepsilon_{21}$ lako uočavamo one koji su u svakom slučaju manji od m. To su $\alpha_1, \ldots, \alpha_{11}, \beta_1, \ldots, \beta_{11}, \gamma_1, \ldots, \gamma_{10}$. Ukupno 32.

Slične okolnosti važe i u slučaju bilo kog n. Upravo, $n_1 \leq \frac{3n}{4}$ a isto tako i $n_3 \leq \frac{3n}{4}$.

Tako da za funkciju $T(n)$ važe sljedeće dvije rekurentne relacije:

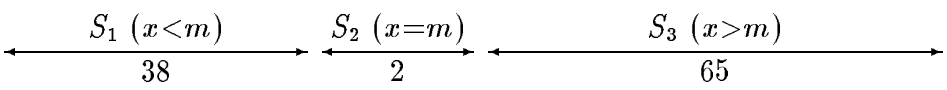
$$\begin{cases} \text{ako je } n < 50 \text{ onda je } T(n) \leq cn, \\ \text{ako je } n \geq 50 \text{ onda je } T(n) \leq cn + T(\frac{n}{5}) + T(\frac{3n}{4}) \end{cases}$$

za neku konstantu $c \geq 1$. Ovdje se cn iz prve relacije odnosi na $L2$ "uredi S", cn iz druge relacije obuhvata trošak za $L4$ i $L7$, $T(\frac{n}{5})$ odnosi se na $L6$, $T(\frac{3n}{4})$ odnosi se na $L9$ ili $L11$. Treba riješiti par rekurentnih relacija. Rješavanjem će se dobiti sljedeće. Postoji konstanta $const \geq 1$ takva da je $T(n) \leq const \cdot n$, za svako $n \in N$. Znači da je $T(n) = O(n)$ kad $n \to \infty$.

U zaključku, zapaziti da se na desnoj strani rekurentne relacije pojavljuje $T(\frac{n}{5}) + T(\frac{3n}{4})$, što i omogućuje da složenost bude linearna, budući da je $\frac{1}{5} + \frac{3}{4} < 1$. Ova okolnost manifestuje se prilikom rješavanja rekurentne relacije.

Govoreći o istoj stvari na drugi način, algoritam A_2 je efikasan zahvaljujući sljedećoj okolnosti. Potprogram SELECT je rekurzivan, a za njegov parametar S važi $|S| = n$. On će pozvati samog sebe ili u $L9$ ili u $L11$, gdje će dužina parametra biti $\leq \frac{3n}{4}$. Dakle, selekcija se sužava na manji niz čiji je broj članova sveden garantovano na $\frac{3}{4}$ dosadašnjeg broja članova ili je još bolje sveden.

Slika ilustruje linije $L9$–$L11$, odnosno podešavanje startnih brojeva: $n = 105$, $n_1 = 38$, $n_2 = 2$, $n_3 = 65$. Ako se tražilo SELECT$(20, S)$ onda će biti pozvano SELECT$(20, S_1)$. Ako se traži SELECT$(40, S)$ onda će biti saopšteno m kao rezultat. Ako se traži SELECT$(60, S)$ onda će biti pozvano SELECT$(20, S_3)$.



3.7. ORDER STATISTICS, PRODUŽETAK

PROCEDURE SELECT(k, S);
IF $|S| = 1$ tj. $n = 1$ THEN RETURN jedini član niza S;
slučajno izaberi iz niza S član a;
uporedi a sa svakim $x \in S$ i obrazuj nizove S_1, S_2, S_3 da sadrže $x < a$, $x = a$ i $x > a$;
$n_1 \leftarrow |S_1|$; $n_2 \leftarrow |S_2|$; $n_3 \leftarrow |S_3|$ (tako da je $n_1 + n_2 + n_3 = n = |S|$);
IF $1 \leq k \leq n_1$ THEN rezultat $\leftarrow$ SELECT(k, S_1);
IF $n_1 + 1 \leq k \leq n_1 + n_2$ THEN rezultat $\leftarrow a$;
IF $n_1 + n_2 + 1 \leq k \leq n_1 + n_2 + n_3$ THEN rezultat $\leftarrow$ SELECT$(k - n_1 - n_2, S_3)$;
RETURN rezultat

Razmatra se isti zadatak kao u prethodnom naslovu: odrediti k–ti po veličini u nizu S.

U ovom naslovu predlaže se novi algoritam A_3 za rješavanje tog zadatka. Potprogram SELECT opisuje A_3. Moglo se potprogramu dati neko drugo ime, da se bolje razlikuje od onog iz prethodnog naslova.

Vidimo da je ideja algoritma A_3 bliska ideji algoritma quicksort iz naslova 3.5.

Može se pokazati da je složenost algoritma A_3 linearna: $T(n) = O(n)$ po kriterijumu očekivanog slučaja.

4. STRUKTURE PODATAKA ZA SKUPOVE

Mnogi zadaci mogu da budu formulisani u terminima osnovnih matematičkih objekata, kakvi su recimo skupovi. U ovoj glavi razmatraju se zadaci u kojima se vrše operacije nad skupovima. S jedne strane, definisaćemo sedam osnovnih operacija nad skupovima. S druge strane, definisaćemo i osnovne strukture podataka za skupove: heš–tabela, drvo binarnog pretraživanja i 2–3 drvo. Razmatraćemo kako se operacije izvode u slučaju primjene određene strukture. Izgradnja dobre strukture podataka ide ruku pod ruku sa izgradnjom dobrog algoritma.

4.1. OSNOVNE OPERACIJE NAD SKUPOVIMA

U ovom naslovu jednostavno se nabrajaju, odnosno definišu osnovne operacije, a ima ih sedam na broju.

1. MEMBER(a, S). Izvršavanjem operacije ili naredbe MEMBER(a, S) utvrđuje se da li objekat a pripada ili ne pripada skupu S. Ako $a \in S$ onda se štampa "da", a ako $a \notin S$ onda se štampa "ne".

2. INSERT(a, S). Skup S zamjenjuje se skupom $S \cup \{a\}$. Zapaziti da, u trenutku kada je naredba INSERT(a, S) došla na red da bude izvršena, može biti bilo $a \in S$ bilo $a \notin S$. Podrazumijeva se da se, tokom izvršavanja programa σ, vodi evidencija o jednom skupu S ili o nekoliko skupova. Stanje u skupu očito se mijenja tokom izvršavanja. Program σ sastoji se od naredbi sedam vrsta.

3. DELETE(a, S). Skup S zamjenjuje se skupom $S \setminus \{a\}$. Pred ovu naredbu bilo je $a \in S$ ili $a \notin S$.

4. UNION(S_1, S_2, S_3). Skupovi S_1 i S_2 zamjenjuju se skupom $S_1 \cup S_2 = S_3$. Drugim riječima, S_1 i S_2 se brišu iz evidencije, a S_3 se dodaje evidenciji. Pretpostavlja se da su skupovi S_1 i S_2 disjunktni (kada se izvršava ova operacija). Tako da ne moramo da brišemo eventualne duplikate u uniji.

5. FIND(a), pronađi a. Štampa se ime onog skupa kome objekat a trenutno pripada. Ako je a član više skupova (više skupova iz evidencije) onda djelovanje naredbe FIND(a) nije definisano.

6. SPLIT(a, S). Skup S dijeli se na dva skupa S_1 i S_2, gdje je $S_1 = \{b \in S |\ b \leq a\}$ i $S_2 = \{b \in S |\ b > a\}$. Pretpostavlja se da je prisutna relacija potpunog uređenja $\leq$ za članove skupa S.

7. MIN(S). Štampa se najmanji član skupa S (najmanji u odnosu na relaciju $\leq$).

Imamo skup od sedam vrsta naredbi. Za njegov podskup kaže se da predstavlja jedan rječnik. Na primjer, kod algoritama za pretraživanje koristi se rječnik od tri naredbe i to: MEMBER, INSERT i DELETE.

Navedimo jedan primjer programa σ:
INSERT(a, S_1); INSERT(a, S_2); INSERT(b, S_1); MEMBER(b, S_2)

Program σ sastavljen je od sedam nabrojanih vrsta naredbi. Vidimo da je jezik na kome je σ sastavljen višeg nivoa od recimo programskog jezika paskal. To znači da jednoj naredbi programa σ odgovara više naredbi u programu na paskalu. Uzimamo da se σ izvršava pomoću recimo paskala. Drugim riječima, uzimamo da se σ izvršava pomoću nekog višeg programskog jezika. Znamo da su glavni primjeri viših programskih jezika paskal i jezik C.

U ovom naslovu još se definišu dva moguća načina za izvršavanje programa σ, on–line i off-line. U slučaju on–line izvršavanja, naredbe koje čine σ izvršavaju se pojedinačno, jedna po jedna, odvojeno tj. nezavisno jedna od druge (redom kako su napisane u nizu). U slučaju off–line izvršavanja programa σ dozvoljava se da čitavi program σ bude pregledan prije nego što

se bilo kakav odgovor da, naredbe mogu da se izvršavaju "zajedno". Pogledajmo na primjeru. Pogledajmo program σ koji se sastoji od dvije naredbe:

DELETE(a, S); DELETE(b, S)

U slučaju on–line, prvo se izvrši prva naredba, pa tek poslije toga počinje izvršavanje druge naredbe DELETE(b, S). U slučaju off-line, imamo pravo da dvije naredbe izvršavamo "zajedno". Drugim riječima, sada možemo da pregledamo redom članove skupa S, za svaki član ispitujemo da li je jednak a ili b i vršimo brisanje ako jeste.

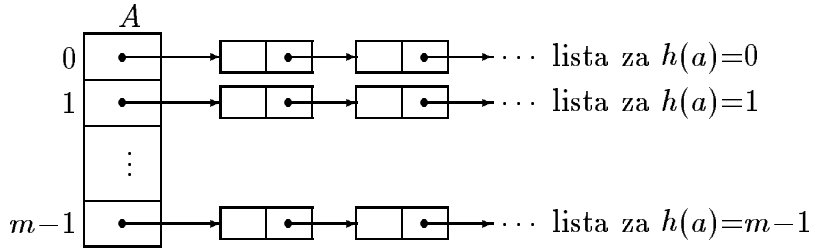
Svaki on–line algoritam može da se koristi kao off–line algoritam, a obrnuto ne važi.

4.2. HEŠOVANJE

U ovom naslovu razmatra se zadatak o održavanju ili memorisanju promjenljivog skupa S. Skupu S mogu da se dodaju novi članovi, iz skupa S mogu da se odstranjuju stari članovi i, s vremena na vrijeme, treba odgovoriti na pitanje – da li u datom trenutku objekat x pripada skupu S. Dakle, traži se struktura podataka na kojoj se lako realizuje rječnik: INSERT, DELETE i MEMBER. Pretpostavlja se da se objekti, koji mogu da se pojave u skupu S, biraju iz jednog univerzalnog skupa U koji je izuzetno velik, tako da nije ostvarljivo da se skup S predstavi kao vektor bitova. U ovom naslovu razmatra se tehnika hešovanja ili rasutog adresiranja, koja može da posluži za postavljeni zadatak.

Primjer. Kompajler drži u tabeli simbola (symbol table) podatke o svim identifikatorima koje je uočio u programu koji se prevodi, o svim imenima, posebno o imenima promjenljivih. Za većinu programskih jezika, skup svih mogućih identifikatora U je izuzetno velik. Na primjer, u fortranu ima oko $1{,}62 \times 10^9$ mogućih identifikatora; dužina imena ≤ 6 znakova. Zato tabela simbola S ne može da bude predstavljena pomoću jednog niza (pomoću vektora bitova). Operacije koje kompajler izvodi nad tabelom simbola su dva tipa. Prvo, novi identifikatori dodaju se tabeli kada se na njih naiđe. Dodavanje znači priključivanje prostora tabeli. U prostor se upisuje sami identifikator i upisuju se njegovi atributi. Atributi su recimo: predviđena fizička adresa promjenljive, njena vrsta (cjelobrojna ili realna ili ...) i slično. Drugo, kompajler s vremena na vrijeme traži neku informaciju o identifikatoru, npr. – da li je on cjelobrojnog tipa. Potrebne su operacije INSERT i MEMBER. Struktura podataka koja se razmatra u ovom naslovu često se koristi za realizaciju tabele simbola.

Postoji više varijacija teme hešovanja, a mi ćemo razmotriti samo njenu osnovnu ideju. Šema hešovanja predstavljena je na slici. Neka je $m \geq 1$. Prisutna je tzv. funkcija hešovanja h. Oblast definisanosti funkcije h je univerzalni skup U. Njene vrijednosti su cijeli brojevi u granicama od 0 do $m-1$. Pretpostavlja se da vrijednost $h(a)$ može da bude izračunata za konstantno vrijeme, za svaki objekat $a \in U$, zato što želimo da algoritam bude efikasan. Svi članovi a skupa S raspoređeni su u m listi, zavisno od vrijednosti $h(a)$. Svi članovi $a \in S$ takvi da je $h(a) = i$ pripadaju jednoj listi, za svako $i \in \{0, 1, \ldots, m-1\}$. Svaka lista ima svoju glavu. Glave su okupljene u jedan niz. Dakle, prisutan je niz A veličine m (indeksi u nizu A kreću se od 0 do $m-1$). Članovi niza A su poenteri na liste članova skupa S. Lista poentirana sa $A[i]$ sastoji se od svih članova a skupa S takvih da je $h(a) = i$.



Vratimo se primjeru o tabeli simbola. Navedimo jednu mogućnost za funkciju h. Neka $h(a)$ bude prvo slovo u imenu a. Recimo, $a = \text{PROMJ}$, $h(a) = \text{P}$. Ili svejedno neka $h(a)$ bude redni broj prvog slova. Recimo, $h(\text{PROMJ}) = 15$. Redni brojevi se kreću od 0 do 25; $m = 26$.

Da bi se izvršila naredba INSERT(a, S), mi izračunamo $h(a)$ i zatim pretražujemo listu poentiranu sa $A[h(a)]$. Ako u toj listi nema objekta a onda se a uključuje u tu listu (na njenom kraju). Da bi se izvršila naredba DELETE(a, S), mi takođe pretražujemo listu $A[h(a)]$ i brišemo a, ako otkrijemo a u listi. Slično, odgovor na naredbu MEMBER(a, S) dobija se pregledanjem liste $A[h(a)]$.

Analiza. Osnovnu teškoću predstavlja pronalaženje funkcije hešovanja koja će članove skupa S da rasporedi po tabeli (po listama) ravnomjerno ili približno ravnomjerno. Drugim riječima, dužine listi trebalo bi da budu približno izjednačene.

Za kompajlera je najgori slučaj kada u izvornom programu na fortranu (čije prevođenje se vrši) sva imena promjenljivih koja se pojavljuju imaju jedno te isto prvo slovo. Nije postignuto rasipanje imena po listama.

Kakav je odnos između broja m i broja članova skupa S? Listi treba da bude približno koliko i članova skupa. Ili listi ima više nego što ima članova skupa. Onda je u većini slučajeva lista prazna ili sadrži samo jedan član. Na stranu što se broj članova skupa S mijenja tokom izvršavanja programa σ. I na stranu što, po pravilu, nije unaprijed poznato koliku najveću vrijednost može da dostigne broj članova skupa S. Ako je svaka lista prazna ili sadrži samo jedan član onda se kaže da nema kolizija u heš–tabeli.

Već je rečeno da je hešovanje dobro kada je $1 \ll \operatorname{card} U$ i $\operatorname{card} S \ll \operatorname{card} U$, $\operatorname{card} U$ je ogroman i $\operatorname{card} S$ je znatno manji od $\operatorname{card} U$. Naravno da je i $m \ll \operatorname{card} U$. Oznaka $\ll$ znatno manje od. Oznaka card kardinalni broj skupa (broj članova skupa).

O složenosti hešovanja. Neka se program σ sastoji od n naredbi INSERT. U najgorem slučaju, dešava se da svih n članova koji se umeću treba da se upišu u jednu te istu od m listi. Tako da je vrijeme potrebno za ostvarivanje j–te naredbe srazmjerno sa j, gdje je $1 \leq j \leq n$. Sabrati $1 + \ldots + n$. Tako da je ukupno vrijeme potrebno za ostvarivanje programa σ srazmjerno sa n^2, jednako $O(n^2)$. U očekivanom slučaju, vremenski trošak za pojedinu naredbu je konstantan, tako da se za ostvarivanje čitavog programa σ potroši vrijeme koje je jednako $O(n)$. Ovdje smo pretpostavili da je $n \leq m$. Još, ovdje smo pretpostavili i da su vrijednosti $h(a)$ raspoređene ravnomjerno od 0 do $m - 1$. Ako σ sadrži naredbe sva tri tipa INSERT, DELETE i MEMBER onda važi isti zaključak.

Slijede dvije dopune.

Dopuna. Izbor funkcije hešovanja h je važan. Navedimo jednu jednostavnu i često korišćenu mogućnost za njeno definisanje. Može se staviti da je $h(a) = a \bmod m$, $h(a)$ je ostatak kod dijeljenja $a : m$. Ovdje je U skup cijelih brojeva ili neki dio skupa cijelih brojeva.

Dopuna. Uzmimo da se broj članova skupa S povećava tokom izvršavanja programa σ. Neka se radi sa heš–tabelom T_0 koja se sastoji od m listi. Neka je u datom trenutku broj članova skupa dostigao m. Mi tada definišemo novu funkciju hešovanja čije su vrijednosti cijeli brojevi u granicama od 0 do $2m - 1$. Umjesto stare tabele T_0 izgradi se nova tabela T_1 sa $2m$ listi. Od datog trenutka pa ubuduće, radi se sa tom novom funkcijom hešovanja i sa tom novom heš–tabelom T_1. Skup S nastavlja da raste. Kada broj članova u tabeli dostigne $2m$ onda se prelazi na tabelu T_2 kojoj odgovara $4m$ listi. Itd. Dakle, konstruiše se serija heš–tabela. Dakle, mi smo izvršili dogradnju algoritma, zato što želimo da algoritam bude efikasan.

O tzv. paradoksu rođendana. Neka je $n \geq 1$. Razmotrimo skup X sačinjen od n osoba. Za $x \in X$, stavimo $h(x) = y$, gdje je $1 \leq y \leq 365$, y je datum i mjesec rođenja. Ima li preklapanja? Razmotrimo događaj A: neke dvije osobe imaju isti rođendan. Uvedimo oznaku $p_n = P(A)$ za

vjerovatnoću. Da li je $p_n < 1/2$ ili $p_n > 1/2$? Ako je $n = 22$ onda je $p_n < 1/2$. Međutim, već za $n = 23$ imamo $p_n > 1/2$.

Pojam "heš" (hash) koristi se u programiranju i u drugom značenju. Dajemo definiciju funkcije hešovanja h. Razmotrimo binarnu azbuku $A = \{0, 1\}$. Označimo sa $X = \Omega(A)$ skup svih riječi u toj azbuci. Neka je $n \geq 1$. Neka je $Y \subset \Omega(A)$, gdje skupu Y pripadaju samo riječi y takve da je $|y| = n$ (dužina riječi). Za funkciju h: $X \to Y$ kaže se da je funkcija hešovanja.

Kada je dato $x \in X$, treba izračunati $y = h(x)$. U algoritmu za računanje $y \in Y$, kombinuju se bitovi (slova) riječi x, po pravilu se kombinuju sva slova. Nastoji se da svako slovo utiče na vrijednost rezultata y, iako uslov jednoznačnosti za funkciju h naravno nije ispunjen.

Funkcija se koristi za provjeru prilikom prenošenja fajla f iz jednog čvora računarske mreže u drugi čvor. Poslati f i $h(f)$. Naime, ako je prepisivanje izvršeno tačno onda se heš vrijednost originala $h(f)$ poklapa sa onom kopije.

Za funkciju hešovanja h: $X \to Y$ kaže se da je kriptografska funkcija hešovanja ako je ispunjen sljedeći uslov. Neka je data riječ $y \in Y$, s tim da y pripada skupu vrijednosti funkcije (y predstavlja nečiju sliku). Tada, ne postoji efikasan algoritam da se nađe riječ $x \in X$ takva da je $y = h(x)$. Ili: nije poznat efikasan algoritam.

Kao teorijski primjer kriptografske funkcije hešovanja mogla bi da posluži sljedeća funkcija $h = h(x)$, definisana na skupu prirodnih brojeva: $h(x) = a^x \bmod p$, gdje su fiksirani prost broj p i broj a u granicama $2 \leq a \leq p - 1$. U praksi, h se sastoji od permutacija, transpozicija, funkcije XOR i afinih funkcija.

Kriptografske funkcije hešovanja široko se koriste (u kriptografiji). Kod tih funkcija, obično je $n = 128$ ili $n = 160$ (koliko ima bita u y). Primjeri takvih funkcija su MD5 i SHA–1.

Po pravilu, digitalni potpis se stavlja na heš vrijednost poruke (na $h(m)$), a ne na samu poruku m (message).

4.3. BINARNO PRETRAŽIVANJE

Razmotrimo zadatak o pretraživanju ili traženju, search. Neka je dat skup S od n članova. Članovi pripadaju izvjesnom velikom univerzalnom skupu U. Treba izvršiti program σ koji se sastoji od samih naredbi MEMBER. Kako? Navedimo tri različite mogućnosti. Prvo rješenje: članovi skupa S upišu se u jednu listu. Svaka naredba MEMBER(a, S) izvršava se tako što se redom pregledaju članovi liste, sve dok se a ne otkrije ili se ne dođe do kraja liste. Ovaj način zahtijeva (i u najgorem i u očekivanom slučaju) vrijeme reda veličine n (drukčije zapisano $O(n)$), da se izvrši jedna naredba MEMBER. Drugo rješenje: članovi skupa S upišu se u heš–tabelu koja se sastoji od $\geq \operatorname{card} S = \|S\|$ listi. Naredba MEMBER(a, S) izvršava se pregledanjem liste $h(a)$. Ako može da se odredi dobra funkcija hešovanja h onda je vrijeme potrebno za ovu naredbu: $O(1)$ u očekivanom slučaju i $O(n)$ u najgorem slučaju. O trećem rješenju (o binarnom pretraživanju) biće riječi u ovom naslovu. Ovo rješenje moguće je kada je prisutna relacija potpunog uređenja $\leq$ među članovima skupa U.

Opišimo algoritam. U pripremnom koraku uradi se sljedeće. Članovi skupa S smjeste se u niz A i zatim se niz A uredi tako da postane $A[1] < A[2] < \ldots < A[n]$. Dakle, u slučaju binarnog pretraživanja vrši se traženje u uređenom obliku niza. Opišimo u nastavku kako se izvršava pojedina naredba MEMBER(a, S). Prvo se objekat a uporedi sa članom skupa koji se u nizu A nalazi na poziciji $[(n + 1)/2]$, neka je taj član skupa označen sa b. Ako je $a = b$ onda mi saopštavamo odgovor "da" i zaustavljamo se. Inače, usmjeravamo se na prvu polovinu niza A ako je $a < b$ ili na drugu polovinu niza A ako je $a > b$ i ponavljamo postupak. Itd. Područje na koje se odnosi pretraživanje stalno se sužava. Mi tako nastavljamo, da bismo otkrili a ili zaključili da a nije u S.

Slijedi tekst rekurzivnog potprograma SEARCH(a, f, l) na nestrogom paskalu. Parametri potprograma imaju sljedeći smisao: a – objekat koji se traži i f i l – prvi, odnosno posljednji član niza A u području na koje se odnosi pretraživanje. Drugim riječima, pregledaju se članovi na mjestima $f, f+1, \ldots, l$.

PROCEDURE SEARCH(a,f,l);
IF $f > l$ THEN RETURN "ne";
$m \leftarrow [(f+l)/2]$; $b \leftarrow A[m]$;
IF $a = b$ THEN RETURN "da";
IF $a < b$ THEN RETURN SEARCH(a,f,$m-1$)
ELSE RETURN SEARCH(a,$m+1$,l)

Glavni program za binarno pretraživanje svodi se na jednu naredbu pozivanja potprograma SEARCH(a,1,n).

Kolika je složenost za izvršavanje jedne naredbe MEMBER(a, S) po potprogramu SEARCH, mjereno brojem zahtijevanih operacija upoređivanja.

Iz koraka u korak, dužina područja se prepolovi ili se broj članova koji još uvijek dolaze u obzir još bolje smanji (još malo bolje smanji). Poslije prvog koraka dolazi u obzir $n/2$ članova, poslije drugog koraka preostaje svega $n/4$ članova, itd., poslije k koraka biće svedeno na svega $n/2^k$ članova. Ako napišemo $n/2^k = n/n = 1$ onda $k = \log_2 n$, pišući približno. Složenost je približno jednaka $\log_2 n$. Imamo u vidu najgori slučaj.

Ne predstavlja teškoću da se zahtijevani broj koraka (broj operacija upoređivanja) sasvim tačno izračuna. Kada se račun sprovede onda će se dobiti kako slijedi. Važi nejednakost $T(n) \leq \lceil \log_2(n+1) \rceil$. Imamo u vidu najgori slučaj. Zaključak: search $\log_2 n$.

Na uređivanje niza A, u pripremnom koraku, očito se odnosi određeni vremenski trošak. Kada smo računali složenost izvršavanja jedne naredbe MEMBER, nismo uzeli u obzir taj vremenski trošak.

Predlaže se mala modifikacija algoritma, da piše: if $a \leq b$ then return search(a, f, m) else return search$(a, m+1, l)$.

4.4. DRVETA BINARNOG PRETRAŽIVANJA

Razmotrimo sljedeći zadatak. Imamo skup S u koji mogu da se umeću objekti i iz koga mogu da se brišu članovi. S vremena na vrijeme želimo da saznamo da li određeni objekat pripada skupu S ili da saznamo koji je trenutno najmanji član u S. Pretpostavlja se da članovi skupa S potiču iz jednog velikog univerzalnog skupa U u kome je prisutna relacija linearnog uređenja $\leq$. Ovaj zadatak može da bude apstraktno formulisan kao zadatak o izvršavanju programa σ sastavljenog od naredbi sljedeće četiri vrste: INSERT, DELETE, MEMBER i MIN. Na primjer, niz naredbi σ može da glasi:

INSERT(a, S); INSERT(b, S); INSERT(c, S); MIN(S);
MEMBER(c, S); MEMBER(d, S); DELETE(a, S)

Struktura podataka koja je pogodna za ove četiri naredbe jeste drvo binarnog pretraživanja.

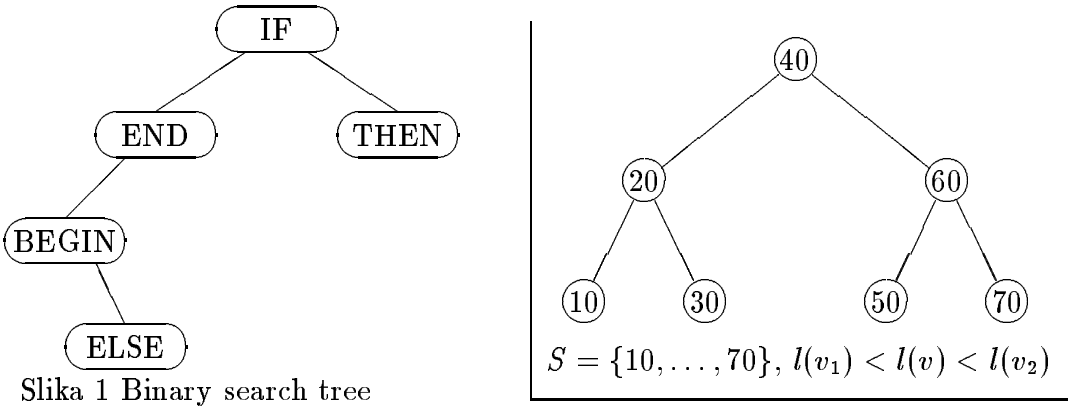
Definicija. Za binarno drvo T kažemo da predstavlja drvo binarnog pretraživanja za skup S ako je T labelisano ili označeno drvo: svaki vrh $v \in T$ označen je ili obilježen je jednim objektom $l(v) \in U$ i to za sve labele ili imena važi $l(v) \in S$, s tim da su još ispunjena sljedeća tri uslova. 1. Za svaki vrh v_1 iz lijevog poddrveta vrha v važi $l(v_1) < l(v)$. 2. Za svaki vrh v_2 iz desnog poddrveta vrha v važi $l(v) < l(v_2)$. 3. Za svaki član $a \in S$, postoji vrh v (postoji tačno jedan vrh v) takav da je $l(v) = a$.

Govoreći prostije, članovi skupa S raspoređeni su po vrhovima drveta T bez ponavljanja i T nema drugih objekata kao ime vrha. Još, imena $l(v) \in S$ raspoređena su tako da za njih važi srednji redosljed (inorder zakon).

Jednom skupu odgovara više mogućih drveta binarnog pretraživanja. Kao primjer, na slici 1 prikazano je jedno moguće drvo binarnog pretraživanja za rezervisane riječi paskala BEGIN, ELSE, END, IF i THEN. Linearno uređenje u ovom slučaju jeste leksikografsko uređenje.

Zapaziti da naredbe MEMBER i MIN ne mijenjaju strukturu drveta T, dok je naredbe INSERT i DELETE mijenjaju, uopšte uzev. Svaka naredba programa σ koristi, tokom svog izvršavanja, dvije okolnosti: da su članovi skupa S raspoređeni po vrhovima $v \in T$ i da za $l(v) \in S$ važi inorder zakon. Drugim riječima, naredba se oslanja na te dvije okolnosti, da bi izvršavanje bilo efikasno, da bi algoritam bio efikasan. S druge strane, naredbe koje mijenjaju strukturu drveta obavezne su sljedeće. One nemaju pravo da poremete dvije okolnosti. Kada se njihovo izvršavanje okonča, T treba da i dalje bude drvo binarnog pretraživanja za skup S. Da bi se sljedeća naredba iz niza σ takođe izvršila na efikasan način. U nastavku će biti data četiri algoritma, za svaku od četiri moguće vrste naredbi u σ.

Napominje se sljedeće. Naša definicija drveta predviđa da je skup vrhova neprazan, drvo ima barem korijen. U ovom naslovu, iz tehničkih razloga, mi ćemo govoriti i o praznom drvetu (drvetu čiji je broj vrhova jednak nuli) kao o binarnom drvetu.



Slika 1 Binary search tree

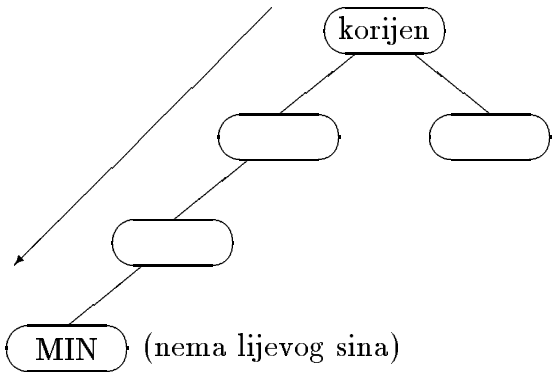
Dakle, neka je T drvo binarnog pretraživanja za skup S i neka treba da se izvrši naredba MEMBER(a, S). Za ovo imamo sljedeći algoritam. Označimo sa r korijen drveta. Tako da je $b = l(r)$ član koji je pridružen korijenu. Uporedimo a i b. Ako se poklapaju onda je jasno da objekat a pripada skupu S. Ako je $a < b$ onda pregledamo lijevo poddrvo korijena ako ono postoji, ako korijen ima lijevog sina, ako lijevo poddrvo korijena nije prazno. Ako je $a > b$ onda pregledamo desno poddrvo korijena ako ono postoji. Itd. Ovim postupkom, ako a pripada S, a će biti otkriven, inače se proces završava kada bi trebalo da se pregleda nepostojeće lijevo ili desno poddrvo. U nastavku je dat tekst algoritma za MEMBER na nestrogom paskalu. Tekst se sastoji od glavnog programa i od potprograma SEARCH(a, v). Smisao parametara potprograma: a – objekat koji se traži, v – vrh do koga smo došli tokom pregledanja, pregleda se njegovo poddrvo. U početku je taj vrh jednak upravo korijenu drveta, $v = r$.

Za MEMBER(a, S):
glavni program:
IF T = prazno THEN "ne" ELSE
SEARCH(a, r), gdje je r korijen od T
potprogram:
PROCEDURE SEARCH(a, v);

IF $a = l(v)$ THEN RETURN "da";
IF $a < l(v)$ i v nema lijevog sina THEN RETURN "ne";
IF $a < l(v)$ i v ima lijevog sina w THEN RETURN SEARCH(a, w);
IF $a > l(v)$ i v nema desnog sina THEN RETURN "ne";
IF $a > l(v)$ i v ima desnog sina w THEN RETURN SEARCH(a, w)

Na redu je drugi algoritam. Skup S predstavljen je naravno i dalje pomoću drveta binarnog pretraživanja T. Algoritam za naredbu INSERT(a, S) dat je na nestrogom paskalu, u nastavku. Sastoji se od teksta glavnog programa i teksta potprograma ADD. Smisao parametara potprograma ADD(a, v) je sljedeći: a – objekat koji se umeće, v – tekući vrh drveta, vrh drveta do koga smo došli tokom pregledanja. Znamo da se INSERT može tražiti i za objekat koji pripada i za objekat koji ne pripada skupu. Opišimo algoritam riječima. Zapaža se sličnost dva algoritma, za MEMBER i za INSERT. Ako je objekat već prisutan u skupu (MEMBER bi dao odgovor "da"), izvršavanje programa treba prosto okončati. A ako $a \notin S$ onda: kada treba da bude pregledano nepostojeće poddrvo, lijevo ili desno, tada se zaključuje da objekat ne pripada skupu. Treba drvetu dodati jedan novi vrh i pridružiti objekat tom vrhu kao ime tog vrha. Ne može se drvo dopuniti novim vrhom na bilo kom mjestu u drvetu. Bili smo upućeni na lijevu ili desnu stranu na kojoj nema ništa – upravo na tom mjestu dodati vrh, da postane lijevi odnosno desni sin, drvo ostaje inorder. Mali primjer: dodavanje riječi DO drvetu sa slike 1. Postaće lijevi sin od ELSE.

Za INSERT(a, S):
glavni program:
IF T = prazno THEN
BEGIN generiši vrh r da postane korijen od T; $l(r) \leftarrow a$ END
ELSE ADD(a, r), gdje je r korijen od T
potprogram:
PROCEDURE ADD(a, v);
IF $a = l(v)$ THEN RETURN;
IF $a < l(v)$ i v nema lijevog sina THEN
BEGIN generiši vrh w da postane lijevi sin od v; $l(w) \leftarrow a$; RETURN END;
IF $a < l(v)$ i v ima lijevog sina w THEN RETURN ADD(a, w);
IF $a > l(v)$ i v nema desnog sina THEN
BEGIN generiši vrh w da postane desni sin od v; $l(w) \leftarrow a$; RETURN END;
IF $a > l(v)$ i v ima desnog sina w THEN RETURN ADD(a, w)



Slika 2 Kako da odredimo MIN(S)

Treći algoritam, za MIN(S). Prosto rečeno, najmanji član skupa pridružen je vrhu koji se nalazi na završetku tzv. lijeve dijagonale drveta (ona polazi iz korijena). Mali primjer: najmanji

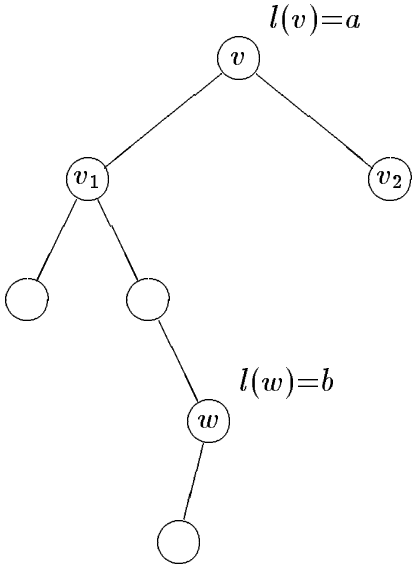
član u drvetu sa slike 1 jeste BEGIN. Dakle, najmanji član drveta binarnog pretraživanja T biće pronađen ako se slijedi put $v_0, v_1, \ldots, v_p$. Ovdje je v_0 korijen drveta T, v_i je lijevi sin vrha v_{i-1} za $1 \leq i \leq p$, a v_p nema lijevog sina. Tada je ime vrha v_p jednako najmanjem članu skupa S, $l(v_p) = \text{MIN}(S)$. Ove okolnosti prikazane su na slici 2.

Četvrto, za DELETE(a, S). Izrazimo algoritam riječima. Uzmimo da je član a koji treba da bude izbrisan pridružen vrhu v, $l(v) = a$. Moguća su tri slučaja, kako slijedi.

Prvi slučaj: vrh v je list drveta. Tada jednostavno treba vrh v ukloniti iz drveta, čime očito član a biva izbrisan iz skupa i inorder zakon ostaje sačuvan.

Drugi slučaj: vrh v ima tačno jednog sina w. Označimo sa u oca od v. Tada treba da w zauzme mjesto od v. Drugim riječima, vrh u treba da postane otac vrha w (premosti se). U okviru ovog, vrh v treba da bude efektivno uklonjen iz drveta. (Ako je v korijen onda će w da postane novi korijen.)

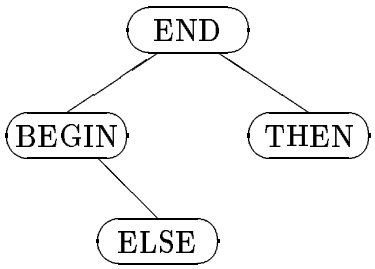
Treći slučaj: vrh v ima dva sina. U lijevom poddrvetu vrha v pronađi vrh w koji sadrži najveći član b. Uočiti da je $b < a$ i da je $b = \max\{x \in S |\ x < a\}$. Uočiti da se w nalazi na kraju desne dijagonale koja polazi iz vrha v_1, gdje je v_1 lijevi sin vrha v. U datom trenutku je $l(v) = a$ i $l(w) = b$. Uraditi $l(v) \leftarrow b$ i još ukloniti w iz drveta. Uklanjanje vrha w vrši se na način prvi slučaj ili na način drugi slučaj, budući da w sigurno nema desnog sina. V. sliku 3.



Slika 3 DELETE(a, S):
T drži S, $a \in S$, $v \in T$, $l(v)=a$
Lijevo od v: svi su $< a$, b je
najveći među njima, $b < a$
Desno od v: svi su $> a$

Pokazuje se da je, nakon brisanja na izloženi način, preostalo drvo takođe drvo binarnog pretraživanja (zadovoljava uslove definicije), za novi sastav skupa S.

Pogledajmo kroz primjere koji se odnose na raniju sliku 1. Ako treba da se izbriše THEN onda je to prvi slučaj, ako treba da se izbriše END onda je to drugi slučaj, a ako treba da se izbriše IF onda nastupa treći. Na slici 4 prikazano je drvo nakon izvršavanja DELETE(IF, S).



Slika 4 Nakon DELETE

4.5. OPTIMALNA DRVETA BINARNOG PRETRAŽIVANJA

U tački 4.4. bio je dat skup $S = \{a_1, a_2, \ldots, a_n\}$ koji je podskup izvjesnog velikog univerzalnog skupa U, a trebalo je da se izradi struktura podataka koja omogućava da se efikasno izvrši niz σ koji se sastoji od samih naredbi MEMBER. Opet razmatramo taj zadatak, ali sada pretpostavljamo da, kada je dat skup S, imamo datu i vjerovatnoću da se naredba MEMBER(a, S) pojavi u σ, za svaki element a univerzalnog skupa U. Sada želimo da izradimo drvo binarnog pretraživanja za S tako da niz σ naredbi MEMBER može da se izvrši on–line (tj. naredbe se izvršavaju jedna po jedna) sa najmanjim mogućim očekivanim brojem upoređivanja.

Neka su $a_1, a_2, \ldots, a_n$ elementi skupa S u rastućem redosljedu i neka je p_i vjerovatnoća naredbe MEMBER(a_i, S) u σ. Neka je q_0 vjerovatnoća da se u σ pojavi naredba oblika MEMBER(a, S), gdje je $a < a_1$. Neka je q_i vjerovatnoća da se u σ pojavi naredba oblika MEMBER(a, S), gdje je $a_i < a < a_{i+1}$. Najzad, neka je q_n vjerovatnoća da se u σ pojavi naredba oblika MEMBER(a, S), gdje je $a > a_n$. Važi $\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$. Da bi se definisala cijena drveta binarnog pretraživanja, pogodno je da se binarnom drvetu doda $n+1$ fiktivnih listova koji odražavaju elemente iz $U \setminus S$. Mi ćemo te listove zvati $0, 1, \ldots, n$. Na slici je prikazano ranije drvo binarnog pretraživanja sa ovim fiktivnim listovima. Na primjer, list imenovan 3 predstavlja elemente a za koje je END $< a <$ IF.

Treba da definišemo cijenu drveta binarnog pretraživanja $c = c(T)$. Ako je element a ime $\ell(v)$ nekog vrha v onda je broj vrhova koji se posjete kada se izvršava naredba MEMBER(a, S) za jedan veći od dubine vrha v. Znamo da se dubina vrha v u drvetu definiše kao dužina puta od korijena ka v. Ako $a \notin S$, neka je $a_i < a < a_{i+1}$, onda je broj vrhova koji se posjete prilikom izvršavanja naredbe MEMBER(a, S) jednak dubini fiktivnog lista i. Tako da cijena drveta binarnog pretraživanja može da se definiše kao

$$c = \sum_{i=1}^{n} p_i(DEPTH(a_i) + 1) + \sum_{i=0}^{n} q_i DEPTH(i),$$

gdje $DEPTH$ označava dubinu.

Kada jednom dobijemo drvo binarnog pretraživanja T minimalne cijene onda mi možemo da izvršimo niz naredbi MEMBER sa najmanjim očekivanim brojem posjeta vrhovima jednostavno koristeći algoritam za T za izvršavanje svake naredbe MEMBER.

Kada su p_i i q_i dati, kako da odredimo drvo minimalne cijene? Pristup podijeli pa vladaj preporučuje da se prvo odredi element a_i koji treba da bude korijen. Tada bi se zadatak podijelio na dva manja podzadatka – izgradnja lijevog poddrveta i izgradnja desnog poddrveta. Ipak, nema jednostavnog načina da se odredi korijen, kraćeg od rješavanja cijelog zadatka. Zato bismo pokušali sa svakim a_i kao korijenom. Tako bismo morali da razmatramo $2n$ podzadataka, po dva za svaki mogući korijen. Tako prirodno dolazimo do rješenja pomoću dinamičkog programiranja – posljednje se određuje korijen, prije toga se riješe svi podzadaci, polazi se od najmanjih podzadataka.

(Podijeli pa vladaj odbacujemo, dinamičko programiranje prihvatamo.)

Neka je T_{ij} drvo minimalne cijene za podskup $\{a_{i+1}, a_{i+2}, \ldots, a_j\}$, gdje je $0 \leq i < j \leq n$. Neka je c_{ij} cijena drveta T_{ij} i neka je r_{ij} korijen drveta T_{ij}. Neka je težina w_{ij} drveta T_{ij} definisana kao $q_i + (p_{i+1} + q_{i+1}) + \ldots + (p_j + q_j)$. Drukčije bi se težina drveta mogla definisati kao broj njegovih vrhova.

Neka je a_k korijen drveta T_{ij}. Tada je $T_{i,k-1}$ njegovo lijevo poddrvo, a T_{kj} je njegovo desno poddrvo (zato što je $a_1 < a_2 < \ldots < a_n$ i inorder). Lako se pokazuje da je to lijevo poddrvo – drvo minimalne cijene za podskup $\{a_{i+1}, a_{i+2}, \ldots, a_{k-1}\}$, a desno poddrvo je drvo minimalne cijene za $\{a_{k+1}, a_{k+2}, \ldots, a_j\}$, vidi sliku. Ako je $k = i+1$ onda nema lijevog poddrveta, a ako je

$k = j$ onda nema desnog poddrveta. Radi lakšeg zapisivanja, tretiraćemo T_{ii} kao prazno drvo. Njegova težina w_{ii} iznosi q_i, a njegova cijena c_{ii} iznosi 0.

Znamo da je $c_{ij} = c(T_{ij}) = \sum_{\nu=i+1}^{j} p_\nu (DEPTH(a_\nu) + 1) + \sum_{\nu=i}^{j} q_\nu DEPTH(\nu)$. U drvetu T_{ij}, kada je $i < j$, dubina svakog vrha iz lijevog ili desnog poddrveta uvećana je za jedan u odnosu na njegovu dubinu u $T_{i,k-1}$ odnosno T_{kj}. Tako da se cijena c_{ij} tog drveta T_{ij} može izraziti kao

$$c_{ij} = c_{i,k-1} + w_{i,k-1} + p_k + c_{kj} + w_{kj},$$

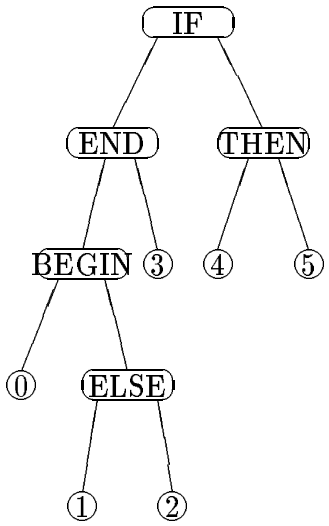
gdje prva dva sabirka na desnoj strani dolaze od lijevog poddrveta, p_k dolazi od korijena, a posljednja dva sabirka od desnog poddrveta. Objašnjenje za ovu formulu: doprinos ovoj zbirnoj cijeni vrhova koji potiču od lijevog poddrveta jednak je: težinski faktor (p_ν ili q_ν) puta udaljenost od korijena, zbir po svim vrhovima; svaka nova udaljenost od korijena je za jedan veća od stare udaljenosti od korijena.

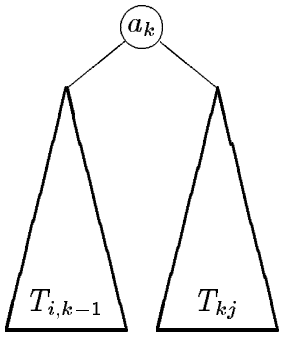
Imamo da je

$$c_{ij} = c_{i,k-1} + c_{kj} + w_{ij}.$$

Prema tome, bira se ono k koje minimizuje zbir $c_{i,k-1} + c_{kj}$ (jer treći sabirak w_{ij} ne zavisi od k). Dakle, da bismo odredili optimalno drvo T_{ij}, mi računamo cijenu drveta čiji je korijen a_k, čije je lijevo poddrvo $T_{i,k-1}$ i čije je desno poddrvo T_{kj} (računamo za svako k takvo da je $i < k \leq j$), pa među njima izaberemo drvo sa minimalnom cijenom.

Time je algoritam konstruisan (posljednja formula plus dinamičko programiranje). Prvo se računa za $j - i = 1$, zatim za $j - i = 2$, ..., jer se primjenjuje dinamičko programiranje.





Poddrvo T_{ij}

Algoritam. Konstrukcija optimalnog drveta binarnog pretraživanja. Ulaz. Brojevi q_0, p_1, q_1, ..., p_n, q_n čiji smisao je ranije opisan (zbir svih ovih brojeva iznosi 1). Izlaz. Drvo binarnog pretraživanja minimalne cijene.

Metod. 1. Mi računamo c_{ij} i r_{ij} za $0 \leq i < j \leq n$, redom po rastućim vrijednostima razlike $j - i = \ell$. Slijedi tekst ovog potprograma COST na nestrogom paskalu. 2. Kada smo odredili sve r_{ij} (treba sastaviti samo drvo), mi pozivamo BUILDTREE$(0, n)$, čime se rekurzivno konstruiše optimalno drvo T_{0n} za skup S. Slijedi tekst ovog potprograma BUILDTREE na nestrogom paskalu. Parametri potprograma i i j odnose se na poddrvo T_{ij}, tj. na podskup $\{a_{i+1}, a_{i+2}, \ldots, a_j\}$.

Tekst prvog potprograma:

PROCEDURE COST;

FOR $i \leftarrow 0$ UNTIL n DO BEGIN $w_{ii} \leftarrow q_i$; $c_{ii} \leftarrow 0$ END;
FOR $\ell \leftarrow 1$ UNTIL n DO FOR $i \leftarrow 0$ UNTIL $n - \ell$ DO BEGIN $j \leftarrow i + \ell$;
$w_{ij} \leftarrow w_{i,j-1} + p_j + q_j$;
neka je m ona vrijednost k $(i < k \leq j)$ za koju je $c_{i,k-1} + c_{kj}$ minimalno;
$c_{ij} \leftarrow c_{i,m-1} + c_{mj} + w_{ij}$; $r_{ij} \leftarrow a_m$ END

Tekst drugog potprograma:

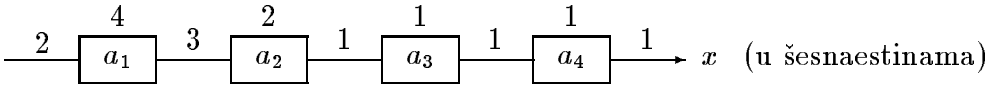
PROCEDURE BUILDTREE(i, j);
unesi novi vrh v_{ij} da postane korijen od T_{ij}; neka v_{ij} ima ime r_{ij};
neka je m indeks od r_{ij} (tj. $r_{ij} = a_m$);
IF $m > i + 1$ THEN neka BUILDTREE$(i, m - 1)$ postane lijevo poddrvo od v_{ij};
IF $m < j$ THEN neka BUILDTREE(m, j) postane desno poddrvo od v_{ij}

Tekst glavnog programa:

učitaj ulazne podatke p_i i q_i;
COST;
BUILDTREE$(0, n)$;
štampaj rezultat $T = T_{0n}$ (optimalno drvo)

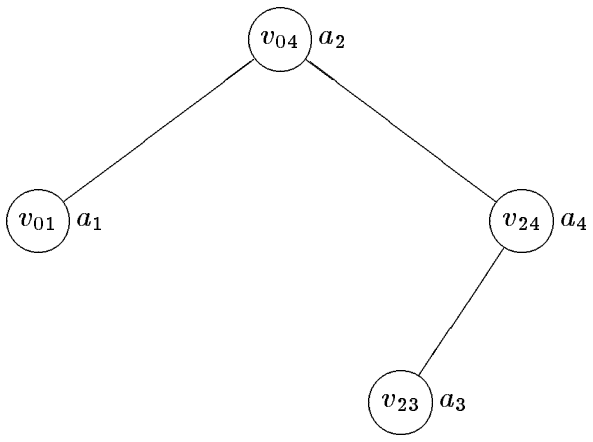
Vidimo da se ukupno rješenje sastoji od glavnog programa i dva potprograma. Matrice W, C i R su globalne promjenljive. Za drvo T_{0n} mogu da se koriste dva niza LEFTSON i RIGHTSON, kako je ranije rađeno.

Primjer: $n = 4$, $q_0 = \frac{1}{8}$, $q_1 = \frac{3}{16}$, $q_2 = q_3 = q_4 = \frac{1}{16}$ i $p_1 = \frac{1}{4}$, $p_2 = \frac{1}{8}$, $p_3 = p_4 = \frac{1}{16}$.



Data je tabela koja nastaje prilikom rada po potprogramu COST i data je definitivna slika – rješenje – izgrađeno optimalno drvo binarnog pretraživanja. Njegova cijena iznosi $\frac{33}{16}$. Vrijednosti w_{ij} i c_{ij} su u tabeli pomnožene sa 16.

	i=0	i=1	i=2	i=3	i=4
ℓ=0	w_{00}=2 c_{00}=0	w_{11}=3 c_{11}=0	w_{22}=1 c_{22}=0	w_{33}=1 c_{33}=0	w_{44}=1 c_{44}=0
ℓ=1	w_{01}=9 c_{01}=9 r_{01}=a_1	w_{12}=6 c_{12}=6 r_{12}=a_2	w_{23}=3 c_{23}=3 r_{23}=a_3	w_{34}=3 c_{34}=3 r_{34}=a_4	
ℓ=2	w_{02}=12 c_{02}=18 r_{02}=a_1	w_{13}=8 c_{13}=11 r_{13}=a_2	w_{24}=5 c_{24}=8 r_{24}=a_4		
ℓ=3	w_{03}=14 c_{03}=25 r_{03}=a_1	w_{14}=10 c_{14}=18 r_{14}=a_2			
ℓ=4	w_{04}=16 c_{04}=33 r_{04}=a_2				



4.6. JEDNOSTAVNI ALGORITAM ZA UNIJU DVA DISJUNKTNA SKUPA

Pisaćemo EX_NAME umjesto EXTERNAL_NAME (spoljašnje ime) i pisaćemo IN_NAME umjesto INTERNAL_NAME (unutrašnje ime).

Na početku sljedeće glave (pete) radićemo algoritam za određivanje (za dati graf) drveta koje povezuje (spanning tree) minimalne cijene. Operacije sa skupovima koje se pojavljuju u tom algoritmu imaju sljedeće tri osobine:

1. Kada se dva skupa ujedinjuju, tada su ta dva skupa disjunktna.
2. Elementi skupova mogu da se smatraju cijelim brojevima od 1 do n.
3. Operacije su UNION i FIND.

U ovoj tački razmotrićemo jednu strukturu podataka za zadatak ovog tipa. Pretpostavimo da je dato n elemenata i da su ti elementi cijeli brojevi $1, 2, \ldots, n$. Uzimamo da je na početku svaki element – skup za sebe. Treba izvršiti niz naredbi UNION i FIND. Ponovimo da naredba UNION ima oblik UNION(A, B, C) i označava da treba dva disjunktna skupa A i B zamijeniti njihovom unijom i nazvati tu uniju C. U primjenama je često nevažno kako su izabrana imena skupova, tako da ćemo pretpostavljati da su skupovi imenovani cijelim brojevima od 1 do n. Više od toga, pretpostavljaćemo da se nikad ne dešava da dva skupa imaju jedno te isto ime.

Struktura podataka koju ćemo opisati u ovoj tački ima sljedeće svojstvo. Neka treba da se izvrši niz σ koji sadrži do $n - 1$ naredbi UNION (to je najveći mogući broj takvih naredbi, u vezi pretpostavki) i sadrži $O(n \log_2 n)$ naredbi FIND. Vrijeme za koje se ovaj niz σ izvrši iznosi $O(n \log_2 n)$. Osim toga, ova struktura je pogodna i za izvršavanje naredbi INSERT, DELETE i MEMBER.

Zapazimo da smo kod algoritama pretraživanja, tačke 4.2. do 4.5., pretpostavljali da elementi potiču iz univerzalnog skupa koji je znatno veći od broja naredbi koje se izvršavaju. U ovoj tački pretpostavljamo da je veličina univerzalnog skupa približno jednaka dužini niza σ naredbi koje treba da se izvrše.

Možda najjednostavnja struktura podataka za UNION–FIND zadatak jeste niz pomoću koga je predstavljena kolekcija skupova prisutna u svakom trenutku. Neka je R niz veličine n takav da je R[i] ime skupa koji sadrži element i. S obzirom da su imena skupova nebitna, može u početku da bude $R[i] = i$, $1 \leq i \leq n$, što odražava činjenicu da na početku kolekcija skupova jeste $\{\{1\}, \{2\}, \ldots, \{n\}\}$ i skup {i} zove se i.

U ovom slučaju, naredba FIND(i) izvršava se time što se štampa tekuća vrijednost R[i]. Tako je cijena izvršavanja naredbe FIND konstantna, što je najbolje što smo mogli da očekujemo. (Može se napisati program za RAM ili program na paskalu.)

U ovom slučaju, naredba UNION(A, B, C) izvršava se tako što se po redu pregleda niz R i svaki put kada se pročita A ili B – promijeni se na C. Tako da je cijena izvršavanja jedne naredbe UNION O(n), a cijena izvršavanja n takvih naredbi iznosi $O(n^2)$, što nije poželjno.

Izloženi jednostavni algoritam može da se poboljša na više načina. Jedno poboljšanje jeste uvođenje povezanih listi. Drugo je da se primijeti da je efikasnije priključiti manji skup većem skupu (nego obrnuto). Da bi se ovo ostvarilo, mora se razlikovati "unutrašnje ime" (koje se koristi za identifikovanje skupova u nizu R) od "spoljašnjeg imena" (koje se pojavljuje u naredbi UNION). Smatraćemo da su oba ta imena – cijeli brojevi od 1 do n, ali ne moraju da se poklope.

Opišimo strukturu podataka. Kao i prije, koristimo niz R tako da je R[i] "unutrašnje ime" skupa koji sadrži element i. Za svaki skup koristimo i povezanu listu koja sadrži elemente skupa. Za implementaciju ovog posljednjeg, dovoljna su dva niza LIST i NEXT. LIST[A] jeste neki cio broj j, što znači da je j prvi element skupa čije je unutrašnje ime A. NEXT[j] jeste sljedeći element skupa A, NEXT[NEXT[j]] jeste sljedeći (dalji) element skupa A, itd.

Dalje, koristimo niz SIZE, gdje je SIZE[A] broj elemenata skupa A.

Na kraju, zbog postojanja dvije vrste imena, uvodimo i dva niza IN_NAME i EX_NAME da povežemo unutrašnja i spoljašnja imena. I to, EX_NAME[A] jeste stvarno ime (tj. ime koje figuriše u narebi UNION) skupa čije je unutrašnje ime A. Slično, IN_NAME[j] jeste unutrašnje ime skupa čije je spoljašnje ime j. U nizu R koriste se unutrašnja imena.

Primjer. Pretpostavimo da je n = 8 i da imamo kolekciju od tri skupa i to $\{1, 3, 5, 7\}$, $\{2, 4, 8\}$ i $\{6\}$ čija su spoljašnja imena redom 1, 2 i 3. Na slici je prikazana struktura podataka za ova tri skupa, pri čemu je uzeto da skupovi 1, 2 i 3 imaju unutrašnja imena redom 2, 3 i 1. V. prvu sliku.

Razjasnimo sada kako se izvršava naredba FIND(i) u slučaju primjene ove strukture podataka. Izvršava se slično kao prije. Prvo se pročita R[i], čime se sazna unutrašnje ime skupa kome element i trenutno pripada. Zatim EX_NAME[R[i]] daje stvarno ime, tj. daje odgovor.

U prethodnom primjeru (prva slika), npr. FIND(5) izaziva štampanje broja 1.

Razjasnimo i kako se izvršava naredba UNION(I, J, K). Možemo rastaviti na pet koraka:

1. Određujemo unutrašnja imena za skupove I i J.
2. Iz niza SIZE saznajemo veličine tih skupova.
3. Prelazimo listu elemenata manjeg skupa i mijenjaju se odgovarajući članovi niza R – dobijaju vrijednost unutrašnjeg imena većeg skupa.
4. Lista elemenata manjeg skupa poveže se sa početkom liste elemenata većeg skupa, čime je završeno ujedinjavanje.
5. Povežu se unutrašnje i spoljašnje ime nastalog skupa.

Dat je tekst procedure na nestrogom paskalu. Implementacija naredbe UNION:

```
procedure UNION(I, J, K);
A ← IN_NAME[I];
B ← IN_NAME[J];
wlg pretpostavimo SIZE[A] ≤ SIZE[B]
otherwise zamijenimo uloge A i B in
begin
ELEMENT ← LIST[A];
while ELEMENT ≠ 0 do
        begin
        R[ELEMENT] ← B;
        LAST ← ELEMENT;
        ELEMENT ← NEXT[ELEMENT]
        end;
NEXT[LAST] ← LIST[B];
LIST[B] ← LIST[A];
SIZE[B] ← SIZE[A] + SIZE[B];
IN_NAME[K] ← B;
EX_NAME[B] ← K
end
```

Konstrukcija iz nestrogog paskala: wlg pretpostavimo ... otherwise ... in naredba. Izraz wlg (without loss of generality) znači ne umanjujući opštost, otherwise znači inače, in znači u.

Primjer. Ako se u situaciji prikazanoj na prvoj slici traži UNION(1, 2, 4) onda poslije izvršavanja te naredbe nastaje situacija koja je prikazana na drugoj slici.

Proučimo na kraju složenost algoritma UNION–FIND zasnovanog na razmatranoj strukturi podataka.

Teorema 1. Algoritam zasnovan na proceduri UNION može da izvrši $n-1$ (to je najveći mogući broj) naredbi UNION za $O(n \log_2 n)$ koraka.

Dokaz. Za prebacivanje jednog elementa iz jedne liste u drugu listu potroši se konstantan broj koraka c (u WHILE–petlji). Odredimo koliko puta jedan fiksirani element može da bude prebacivan, pa ćemo ukupan broj koraka dobiti sabiranjem po svakom elementu. Kako se svaki put prebacuju elementi iz manjeg od dva skupa (u veći) to je taj broj prebacivanja $\leq \log_2 n$ (jer se taj element uključuje u listu koja je bar duplo duža od liste u kojoj se prije nalazio). Dakle, ukupan broj koraka se majorira brojem $c \cdot \log_2 n \cdot n$ plus neki broj koji je srazmjeran broju pozivanja procedure UNION (dio van WHILE–petlje), a broj tih pozivanja je $\leq n$. Zaključak: $O(n \log_2 n)$ koraka. Teorema je dokazana.

Teorema 2. Ako se niz σ sastoji od najviše $n-1$ nredbi UNION i $O(n \log_2 n)$ naredbi FIND onda se σ izvršava za $O(n \log_2 n)$ koraka.

Dokaz. Prema prethodnoj teoremi je $u = O(n \log_2 n)$. Jedna naredba FIND troši konstantan broj koraka, konstanta $\times$ $O(n \log_2 n) = O(n \log_2 n) = f$. Zaključak se dobija po formuli $u + f = O(n \log_2 n) + O(n \log_2 n) = O(n \log_2 n)$. Sabirak u odnosi se na UNION, a drugi sabirak f odnosi se na FIND. Teorema je dokazana.

U teoremi 1. i teoremi 2. imamo u vidu složenost odgovarajućeg programa za RAM, uniformna cijena, tj. jedan korak znači jedna naredba za RAM. Već je ranije rečeno da se ovaj kriterijum i inače podrazumijeva (ako nije posebno naveden neki drugi kriterijum složenosti). Takođe se podrazumijeva "vremenska složenost" i "najgori slučaj".

Ukupan broj naredbi UNION je $\leq n-1$ jer je u početku broj skupova jednak n, a svaka naredba UNION smanjuje za jedan broj skupova.

Dvostruka imena su neophodna zato što se uvijek manji skup uključuje u veći.

	R	NEXT
1	2	3
2	3	4
3	2	5
4	3	8
5	2	7
6	1	0
7	2	0
8	3	0

	LIST	SIZE	EX_NAME	IN_NAME
1	6	1	3	2
2	1	4	1	3
3	2	3	2	1

Skupovi
(sa spoljašnjim imenima):
$1 = \{1, 3, 5, 7\}$,
$2 = \{2, 4, 8\}$,
$3 = \{6\}$

Struktura podataka za UNION–FIND algoritam
(ovo je prva slika)

	R	NEXT
1	2	3
2	2	4
3	2	5
4	2	8
5	2	7
6	1	0
7	2	0
8	2	1

	LIST	SIZE	EX_NAME	IN_NAME
1	6	1	3	–
2	2	7	4	–
3	–	–	–	1
4	–	–	–	2

Skupovi
(sa spoljašnjim imenima):
$3 = \{6\}$,
$4 = \{1, 2, 3, 4, 5, 7, 8\}$

Struktura podataka poslije naredbe UNION
(ovo je druga slika)

4.7. ŠEME (ALGORITMI) KOJE KORISTE BALANSIRANA DRVETA

Samo skica ove tačke, samo pojam.

Kažemo balansirano drvo za drvo čija je visina približno jednaka logaritmu od broja vrhova drveta.

Balansirano drvo se lako konstruiše na početku. Problem je što drvo može da postane debalansirano prilikom izvršavanja niza naredbi INSERT i DELETE. Na primjer, ako mi izvršavamo niz naredbi DELETE koje uklanjaju vrhove stalno iz lijeve strane drveta, doći će do toga da drvo postane nagnuto nadesno, ako se periodično ne vrši rebalansiranje.

Postoji nekoliko mehanizama koji se koriste za rebalansiranje drveta, kada je to potrebno. Više metoda dopušta da struktura drveta bude gipka (elastična), recimo dopušta da broj vrhova u drvetu visine h bude između 2^h i 2^{h+1} (ili 3^h). Takve metode dopuštaju da se najviše udvostruči broj vrhova u poddrvetu, prije nego što se izvrši promjena koja se tiče i korijena tog poddrveta.

Definicija 1. 2–3 drvo jeste drvo u kome svaki vrh koji nije list ima 2 ili 3 sina i u kome svaki put od korijena ka listu ima jednu te istu dužinu. Drvo koje se sastoji od jednog vrha jeste 2–3 drvo.

Na slici je prikazano jedno 2–3 drvo.

Lema. Neka je T 2–3 drvo visine h. Tada je broj vrhova u drvetu T između $2^{h+1}-1$ i $\frac{1}{2}(3^{h+1}-1)$, a broj listova je između 2^h i 3^h.

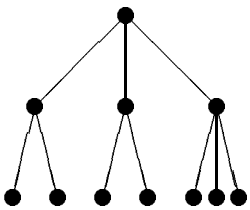
Dokaz. Indukcijom po h.

Izvršavanje naredbe DELETE sastoji se, u zavisnosti od slučaja, ili u prostom uklanjanju jednog vrha ili se još izmijeni struktura jednog dijela drveta (da bi se sačuvala ravnoteža, tj. da bi ostalo 2–3 drvo). Slično i za INSERT.

Definicija 2. AVL drvo je binarno drvo koje ispunjava: visina lijevog i desnog poddrveta određenog vrha se razlikuju najviše za jedan – ovo za svaki vrh.

Definicija 3. Drvo sa ograničenim balansom, drugi naziv α–balansirano drvo, to je binarno drvo koje ispunjava uslov: balans svakog vrha je između α i $1-\alpha$. Balans jednog vrha je po definiciji jednak $(L+1)/(L+R+2)$, gdje su L i R broj vrhova u lijevom, odnosno u desnom poddrvetu.

Svaka od ove tri definicije daje po jedan primjer balansiranog drveta.



2–3 tree

5. ALGORITMI NA GRAFOVIMA

Algoritmi za zadatke koji se odnose na grafove.

U ovoj glavi razmatramo nekoliko glavnih zadataka koji se odnose na grafove, a koji imaju polinomska rješenja (vrijeme potrebno za izvršavanje u slučaju RAM po uniformnoj cijeni), izraženo kao funkcija od broja vrhova grafa.

Šema: obim ulaza $= n$, vremenska složenost $= f(n)$, $f(n) \leq$ polinom.

5.1. DRVO KOJE POVEZUJE MINIMALNE CIJENE

Neka je $G = (V, E)$ povezan neusmjeren graf sa funkcijom cijene c koja preslikava ivice u nenegativne realne brojeve.

Def. Drvo koje povezuje za graf G jeste neusmjereno drvo koje povezuje sve vrhove iz skupa V. Kaže se drvo koje povezuje ili obuhvatno drvo ili engl. spanning tree. Def. Cijena drveta koje povezuje jeste zbir cijena svih njegovih ivica.

Algoritam 5.1. Konstrukcija drveta koje povezuje čija je cijena minimalna, Kruskalov algoritam. Ulaz. Povezan neusmjeren graf $G = (V, E)$ sa funkcijom cijene c: $E \to [0, +\infty)$. Izlaz. Drvo koje povezuje $S = (V, T)$ čija je cijena najmanja moguća, gdje je $T \subset E$.

Metod. Sastavimo prvo listu (red prioriteta, priority queue) svih ivica grafa, tako da u listi cijene ne opadaju. Na početku zamišljamo podgraf $(V, \emptyset)$. Prvi element liste uključuje se u T jer on sigurno poveže dva vrha. Drugi element liste uključuje se u T ako i samo ako time dolazi do nekog povezivanja. Dalje na isti način za treći element. Itd. Sve dok se svi vrhovi ne povežu, kada i imamo rezultat $S = (V, T)$.

Slijedi algoritam na nestrogom paskalu. Promjenljiva VS predstavlja trenutno stanje (tokom izgradnje drveta T) povezanih komponenti. U početku se VS sastoji od n jednoelementnih skupova, gdje je $n = \|V\|$. Kada se dva skupa iz VS povežu onda se oni u VS zamijene njihovom unijom. Drugim riječima, uključivanjem jedne ivice u drvo T, broj članova u familiji skupova VS smanjuje se za jedan. Algoritam se završava kada se broj članova u familiji svede na 1 (kada postane $\|VS\| = 1$). Lako vidimo da je $\|T\| = n - 1$ (u drvetu S ima $n - 1$ ivica).

```
T ← ∅;
VS ← ∅;
FOR svaki vrh v ∈ V DO dodaj {v} skupu VS;
konstruiši red prioriteta Q koji sadrži sve ivice iz E;
WHILE ||VS|| > 1 DO
BEGIN
izaberi (v,w), najjeftiniju ivicu u Q;
izbriši (v,w) iz Q;
IF v i w su u različitim skupovima W1 i W2 u VS THEN
      BEGIN
      zamijeni W1 i W2 u VS sa W1 ∪ W2;
      dodaj (v,w) skupu T
      END
END
```

Sigurni smo da će rezultujući graf biti drvo, tj. da će biti jedan acikličan graf. Zaista, razmotrimo neki ciklus i ukinimo iz njega jednu ivicu. Tada to prestaje da bude ciklus, a vrhovi koji učestvuju ostaju i dalje povezani međusobno. Pored toga, ukidanjem ivice zbirna cijena se samo smanjuje. Zato naziv "drvo koje povezuje" (a ne eventualno "podgraf koji povezuje").

Preskačemo dokaz ispravnosti (pravilnosti) Kruskalovog algoritma.

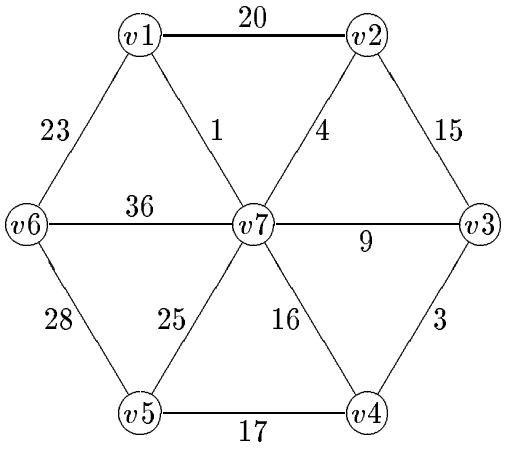
Preskačemo i računanje složenosti ovog algoritma.

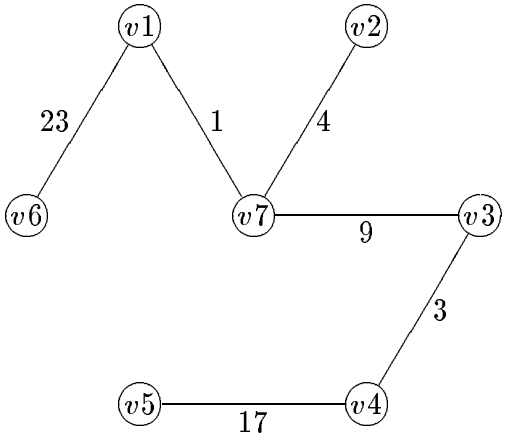
Primjer. Ulazni podatak je graf $G = (V, E)$, a izlazni podatak (rezultat) je drvo $S = (V, T)$ čija je cijena najmanja moguća. Prikazani su: tablica i slika za G i tablica i slika za S. U tablici za G – već je formiran red prioriteta Q, tj. cijene su uređene (da se povećavaju). Cijena drveta S iznosi 57.

Tablica za G: ivica cijena: $(v1, v7)$ 1 $(v3, v4)$ 3 $(v2, v7)$ 4 $(v3, v7)$ 9 $(v2, v3)$ 15 $(v4, v7)$ 16 $(v4, v5)$ 17 $(v1, v2)$ 20 $(v1, v6)$ 23 $(v5, v7)$ 25 $(v5, v6)$ 28 $(v6, v7)$ 36.

Tablica za S: ivica cijena: $(v1, v7)$ 1 $(v3, v4)$ 3 $(v2, v7)$ 4 $(v3, v7)$ 9 $(v4, v5)$ 17 $(v1, v6)$ 23.

Tokom rada programa imamo redom. Na početku je $VS = \{\{v1\}, \{v2\}, \{v3\}, \{v4\}, \{v5\}, \{v6\}, \{v7\}\}$. Prvo se u S uključuje ivica $(v1, v7)$ čime postaje $VS = \{\{v1, v7\}, \{v2\}, \{v3\}, \{v4\}, \{v5\}, \{v6\}\}$. Zatim se dodaje ivica $(v3, v4)$ čime postaje $VS = \{\{v1, v7\}, \{v2\}, \{v3, v4\}, \{v5\}, \{v6\}\}$. Itd. Na kraju će biti $VS = \{\{v1, v2, v3, v4, v5, v6, v7\}\}$.





5.2. OBILAZAK GRAFA PO DUBINI

U nastavku je prepisan jedan naslov iz knjige A.V. Aho, J.E. Hopcroft and J.D. Ullman: The design and analysis of computer algorithms, Addison–Wesley publishing company, Reading, Massachusetts, 1974. Prepisane su strane od 176 do 179. Možda bi trebalo prevesti sa engleskog jezika.

5.2. DEPTH–FIRST SEARCH

Consider visiting the vertices of an undirected graph in the following manner. We select and "visit" a starting vertex v. Then we select any edge (v, w) incident upon v, and visit w. In general, suppose x is the most recently visited vertex. The search is continued by selecting some unexplored edge (x, y) incident upon x. If y has been previously visited, we find another new edge incident upon x. If y has not been previously visited, then we visit y and begin the search anew starting at vertex y. After completing the search through all paths beginning at y, the search returns to x, the vertex from which y was first reached. The process of selecting unexplored edges incident upon x is continued until the list of these edges is exhausted. This method of visiting the vertices of an undirected graph is called a *depth–first search* since we continue searching in the forward (deeper) direction as long as possible.

Depth–first search can be applied to a directed graph as well. If the graph is directed, then at vertex x we select only edges (x, y) directed out of x. After exhausting all edges out of y, we return to x even though there may be other edges directed into y which have not yet been searched.

If depth–first search is applied to an undirected graph which is connected, then it is easy to show that every vertex will be visited and every edge examined. If the graph is not connected, then a connected component of the graph will be searched. Upon completion of a connected component, a vertex not yet visited is selected as the new start vertex and a new search is begun.

A depth–first search of an undirected graph $G = (V, E)$ partitions the edges in E into two sets T and B. An edge (v, w) is placed in set T if vertex w has not been previously visited when we are at vertex v considering edge (v, w). Otherwise, edge (v, w) is placed in set B. The edges in T are called *tree edges*, and those in B *back edges*. The subgraph (V, T) is an undirected forest, called a *depth–first spanning forest for* G. In the case that the forest consists of a single tree, (V, T) is called a *depth–first spanning tree*. Note that if G is connected, the depth–first spanning forest will be a tree. We consider each tree in the depth–first spanning forest to be rooted at that vertex at which the depth–first search of that tree was begun.

An algorithm for the depth–first search of a graph is given below.

Algorithm 5.2. Depth–first search of an undirected graph. *Input.* A graph $G = (V, E)$ represented by adjacency lists $L[v]$, for $v \in V$. *Output.* A partition of E into T, a set of tree edges, and B, a set of back edges.

Method. The recursive procedure SEARCH(v) in Fig. 5.6. adds edge (v, w) to T if vertex w is first reached during the search by an edge from v. We assume all vertices are initially marked "new". The entire algorithm is as follows:

```
6.  T ← ∅;
7.  for all v in V do mark v "new";
8.  while there exists a vertex v in V marked "new" do
9.  SEARCH(v)
```

All edges in E not placed in T are considered to be in B. Note that if edge (v, w) is in E, then w will be on $L[v]$ and v will be on $L[w]$. Thus we cannot simply place edge (v, w) in B if we are at vertex v and vertex w is marked "old" since w might be the father of v.

```
     procedure SEARCH(v);
1.   mark v "old";
2.   for each vertex w on L[v] do
3.   if w is marked "new" then
          begin
4.        add (v,w) to T;
5.        SEARCH(w)
          end
```

Fig 5.6. Depth–first search

Example 5.2. We shall adopt the convention of showing the tree edges in T solid and back edges in B dashed. Also, the tree (or trees) will be drawn with the root (the starting vertex selected at line 8) at the top, and with the sons of each vertex drawn from left to right in the order in which their edges were added at line 4 of SEARCH. Consider the graph of Fig. 5.7.a. One possible partition of this graph into tree and back edges resulting from a depth–first search is shown in Fig. 5.7.b.

Initially, all vertices are "new". Suppose v_1 is selected at line 8. When we perform SEARCH(v_1), we might select $w = v_2$ at line 2. Since v_2 is marked "new", we add (v_1, v_2) to T and call SEARCH(v_2). SEARCH(v_2) might select v_1 from $L[v_2]$, but v_1 has been marked "old". Then suppose we select $w = v_3$. Since v_3 is "new", we add (v_2, v_3) to T and call SEARCH(v_3). Each of the vertices adjacent to v_3 is now "old", so we return to SEARCH(v_2).

Then, proceeding with SEARCH(v_2), we find edge (v_2, v_4), add it to T, and call SEARCH (v_4). Note that v_4 is drawn to the right of v_3, a previously found son of v_2 in Fig. 5.7.b. No "new" vertices are adjacent to v_4, so we return to SEARCH(v_2). Now we find no "new" vertices adjecent to v_2, and so return to SEARCH(v_1). Continuing, SEARCH(v_1) finds v_5, and SEARCH(v_5) finds v_6. All vertices would then be on the tree and marked "old", and thus the algorithm would end. If the graph were not connected, the loop of lines 8–9 would repeat, once for each component.

Theorem 5.2. Algorithm 5.2. requires $O(\text{MAX}(n, e))$ steps on a graph with n vertices and e edges.

Proof. Line 7 and the search for "new" vertices at line 8 require $O(n)$ steps if a list of vertices is made and scanned once. The time spent in SEARCH(v), exclusive of recursive calls to itself, is proportional to the number of vertices adjacent to v. SEARCH(v) is called only once for a given v, since v is marked "old" the first time SEARCH(v) is called. Thus the total time spent in SEARCH is $O(\text{MAX}(n, e))$, and we have the theorem. □

Part of the power of depth–first search is contained in the following lemma, which says that each edge of an undirected graph G is either an edge in the depth–first spanning forest or connects an ancestor to a descendant in some tree of the depth–first spanning forest. Thus all edges in G, whether tree or back, connect two vertices such that one vertex is an ancestor of the other in the spanning forest.

Lemma 5.3. If (v, w) is a back edge, then in the spanning forest v is an ancestor of w or vice versa.

Proof. Suppose without loss of generality that v is visited before w, in the sense that SEARCH(v) is called before SEARCH(w). Thus when v is reached, w is still labeled "new". All "new" vertices visited by SEARCH(v) will become descendants of v in the spanning forest. But SEARCH(v) cannot end until w is reached, since w is on the list $L[v]$. □

There is a natural order which depth–first search imposes on the vertices of a spanning forest. Namely, vertices can be labeled in the order they are visited if we initialize COUNT to 1 between lines 6 and 7 of Algorithm 5.2. and insert

DENUMBER[v] $\leftarrow$ COUNT;

COUNT $\leftarrow$ COUNT + 1;

at the beginning of procedure SEARCH. Then the vertices of the forest would be labeled 1, 2, ..., up to the number of vertices in the forest.

The labels can clearly be assigned in $O(n)$ time for a graph of n vertices. The order corresponds to a preorder traversal of each tree in resulting spanning forest. We shall subsequently assume that all depth–first spanning forests are so labeled. We shall often treat these vertex labels as though they are the vertex names themselves. It thus makes sense to say, e.g., "$v < w$", where v and w are vertices.

Example 5.3. *The depth–first order of the vertices* of the graph in Fig. 5.7.a. is

v_1, v_2, v_3, v_4, v_5, v_6,

which can be ascertained either by tracing the order in which SEARCH was initiated for the various vertices or traversing the tree of Fig. 5.7.b. in preorder.

Note especially that if v is a proper ancestor of w, then $v < w$. Also, if v is to the left of w in a tree, then $v < w$.

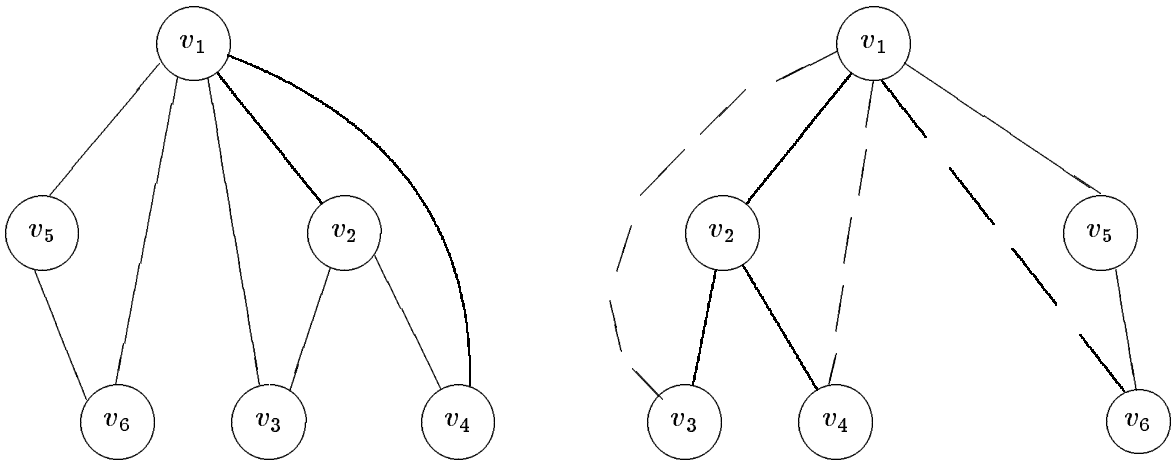


Fig. 5.7. A graph and its depth–first spanning tree

5.3. ALGORITAM ZA TRANZITIVNO ZATVORENJE

Neka su v i w dva vrha u grafu. Razmotrimo sljedeći zadatak. Odgovoriti sa "da" ili "ne" na pitanje: da li postoji put koji vodi od v ka w. U ovom naslovu razmatra se jedan algoritam koji služi za rješavanje tog zadatka, Warshallov algoritam.

Uvedimo potrebne oznake i pojmove. Neka je $G = (V, E)$ usmjeren graf, gdje skup V ima $n \geq 1$ članova. Numerišimo na proizvoljan način članove skupa V brojevima $1, 2, \ldots, n$ ili svejedno stavimo da je $V = \{v_1, v_2, \ldots, v_n\}$. Neka $v \in V$ i $w \in V$. Definišimo $\ell(v, w)$ labelu ili oznaku tog para vrhova na sljedeći način:

$\ell(v, w) = 1$ ili svejedno $\ell(v, w) =$ `true` ako je (v, w) ivica grafa, tj. ako $(v, w) \in E$,

$\ell(v, w) = 0$ ili svejedno $\ell(v, w) =$ `false` ako (v, w) nije ivica grafa, tj. ako $(v, w) \notin E$.

Definicija. Za graf G^* koji ima isti skup vrhova kao i graf G a koji ima ivicu od v ka w ako i samo ako u grafu G postoji put (dužine nula ili više) od vrha v ka vrhu w kaže se da predstavlja tranzitivno zatvorenje grafa G.

Prema tome, pomoću G^* pregledno se pokazuje da li u polaznom grafu G postoji ili ne postoji put koji počinje u vrhu v i čiji je kraj vrh w, samo treba pogledati da li je (v, w) ivica u grafu G^* ili nije. Mi ćemo koristiti sljedeću oznaku: neka je $C = [c_{ij}]_{i,j=1}^{n}$ matrica oblika $n \times n$ definisana na sljedeći način:

$c_{ij} = 1$ ako je (v_i, v_j) ivica u grafu G^*, tj. ako u grafu G postoji put od vrha v_i ka vrhu v_j,

$c_{ij} = 0$ ako (v_i, v_j) nije ivica u grafu G^*, tj. ako u grafu G ne postoji put od vrha v_i ka vrhu v_j.

Drukčije se može reći da mi razmatramo zadatak o određivanju refleksivnog i tranzitivnog zatvorenja date binarne relacije $\rho = E$. Refleksivno zatvorenje neke relacije nastaje kada se toj relaciji dodaju svi parovi oblika (v, v). Što se tiče pojma tranzitivnog zatvorenja relacije, poslužimo se primjerom. Ako $(v_2, v_4) \in \rho$ i $(v_4, v_6) \in \rho$ i $(v_6, v_1) \in \rho$ i $(v_1, v_3) \in \rho$ onda (v_2, v_3) pripada tranzitivnom zatvorenju relacije ρ i slično.

Mi želimo da riješimo sljedeći zadatak. Ulazni podaci programa su podaci o usmjerenom grafu G. Treba sastaviti program koji računa i štampa njegovo tranzitivno zatvorenje.

Pogledajmo ideju rješenja. Prethodno, ako posmatramo put v_2, v_4, v_6, v_1, v_3 onda se kaže da su v_4, v_6 i v_1 međutačke tog puta. Razmotrimo matrice $C^k = [c_{ij}^k]_{i,j=1}^{n}$ za $k = 0, 1, \ldots, n$ čiji su članovi definisani na sljedeći način: $c_{ij}^k = 1$ ako i samo ako postoji put koji polazi od v_i i završava se u v_j i čije sve međutačke pripadaju skupu $\{v_1, \ldots, v_k\}$, a inače je $c_{ij}^k = 0$. Algoritam se temelji na sljedećem jednostavnom zapažanju: $c_{ij}^k = c_{ij}^{k-1} \vee (c_{ik}^{k-1} \,\&\, c_{kj}^{k-1})$. Riječima, postoji put od v_i ka v_j čije su međutačke iz skupa $\{v_1, \ldots, v_k\}$ ako i samo ako: a) postoji put od v_i ka v_j čije su sve međutačke iz $\{v_1, \ldots, v_{k-1}\}$ ili b) postoji put od v_i ka v_k sa indeksima međutačaka do $k-1$ i postoji put od v_k ka v_j sa indeksima međutačaka do $k-1$. Ove okolnosti prikazane su na slikama 1 i 2.

Vidimo da relacija $c_{ij}^k = c_{ij}^{k-1} \vee (c_{ik}^{k-1} \,\&\, c_{kj}^{k-1})$ izražava članove matrice C^k preko članova matrice C^{k-1}. Ona omogućava da svi članovi matrice C^k budu izračunati, ako su u datom trenutku poznati članovi matrice C^{k-1}. Tokom rada algoritma, mi napredujemo po k. Kada se dostigne najveća vrijednost $k = n$ onda su svi vrhovi grafa dopušteni da budu međutačke. Drugim riječima, važi $C^n = C$, odnosno matrica C^n predstavlja rezultat. Time je algoritam konstruisan, jer se lako definišu vrijednosti u polaznoj matrici C^0, kada nijedan vrh nije dopušten kao međutačka. Naime, očito treba da bude $c_{ij}^0 = 1$ ako i samo ako je (i, j) ivica, tj. ako i samo ako je $\ell(i, j) = 1$, u slučaju $i \neq j$. U slučaju $i = j$ treba staviti $c_{ij}^0 = 1$ imajući u vidu put dužine nula od vrha i ka tom istom vrhu i.

Na redu je Warshallov algoritam na nestrogom paskalu:

učitaj podatke o grafu $G = (V, E)$;

for $i,j = 1,\ldots,n$ do if $(i,j) \in E$ then $\ell(i,j) \leftarrow 1$ else $\ell(i,j) \leftarrow 0$;
for $i,j = 1,\ldots,n$ do $c_{ij}^0 \leftarrow \ell(i,j)$;
for $i = 1,\ldots,n$ do $c_{ii}^0 \leftarrow 1$;
for $k = 1,\ldots,n$ do
    for $i,j = 1,\ldots,n$ do
    $c_{ij}^k \leftarrow c_{ij}^{k-1} \vee (c_{ik}^{k-1} \,\&\, c_{kj}^{k-1})$;
for $i,j = 1,\ldots,n$ do $c_{ij} \leftarrow c_{ij}^n$;
štampaj matricu $C = [c_{ij}]$

Primjer. Neka bude $n = 3$, tj. $V = \{v_1, v_2, v_3\}$ i neka bude $E = \{(1,1), (1,2), (2,1), (1,3), (3,2)\}$ i razmotrimo graf $G = (V,E)$. Primijeniti Warshallov algoritam na dati graf, odnosno odrediti tranzitivno zatvorenje datog grafa.

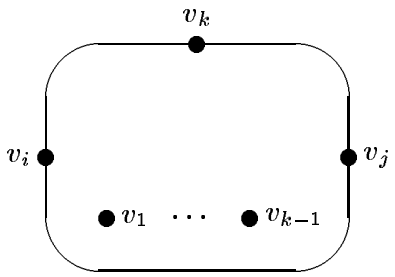
U nastavku su prikazani redom matrica $L = [\ell(i,j)]_{i,j=1}^n$ koja predstavlja ulazni podatak, zatim među–rezultati C^0, C^1 i C^2 i na kraju rezultat C^3 ili svejedno C.

$$L = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad C^0 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \quad C^1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \quad C^2 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

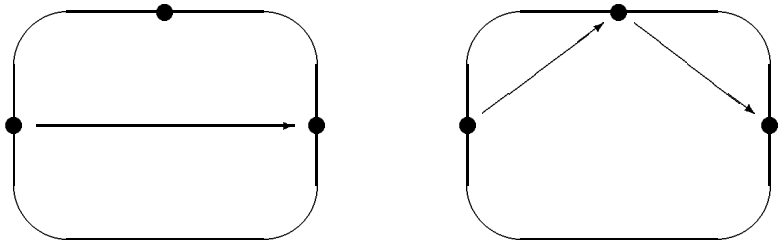
$$C^3 = C = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Vidi se da rezultat (graf G^*, tranzitivno zatvorenje grafa G) ima svih devet mogućih ivica, tj. da je to potpun graf. To znači da od bilo kog vrha ka bilo kom vrhu postoji put u polaznom grafu G. Na slici 3 prikazana su dva grafa G i G^*.

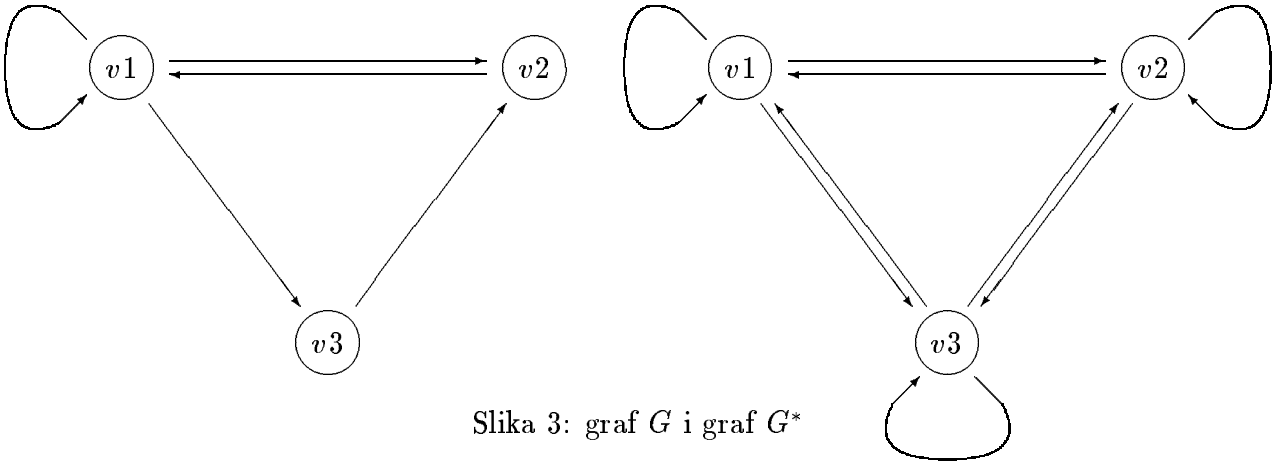
U zaključku, vremenska složenost predloženog algoritma jednaka je $T(n) = O(n^3)$, jer se računa n matrica oblika $n \times n$. Algoritam treba dopuniti tako da u slučaju pozitivnog odgovora (postoji put od i ka j) još saopštava i dionice puta, niz ivica koje čine put, redom. Slično se rješava zadatak kada je polazni graf G jedan neusmjeren graf.



Slika 1: putevi



Slika 2: a) ili b)



Slika 3: graf G i graf G^*

5.4. ALGORITAM ZA NAJKRAĆI PUT

Ako postoje putevi od grada A ka gradu B onda želimo da pronađemo najkraći među njima, želimo da saznamo kolika je dužina najkraćeg puta. Ovom pitanju odgovara u teoriji grafova tzv. zadatak o najkraćem putu od vrha v ka vrhu w. U ovom naslovu razmatra se jedan algoritam koji služi za rješavanje tog zadatka, Floydov algoritam.

Uvedimo potrebne oznake i pojmove. Neka je $G = (V, E)$ usmjeren graf, gdje skup V ima $n \geq 1$ članova. Numerišimo na proizvoljan način članove skupa V brojevima $1, 2, \ldots, n$ ili svejedno stavimo da je $V = \{v_1, v_2, \ldots, v_n\}$. Dodatno, neka je graf G ponderisan (težinski). To znači da je svakoj ivici $e \in E$ pridružena njena cijena (ili dužina ili težina) u oznaci $c(e)$, gdje je $c(e) \geq 0$ za svako $e \in E$.

($c(v, w) \geq 0$ za svako $(v, w) \in E$.)

Neka $v \in V$ i $w \in V$. Definišimo labelu ili oznaku uređenog para vrhova (v, w) u oznaci $\ell(v, w)$ na sljedeći način:

$\ell(v, w) = c(v, w)$ ako $(v, w) \in E$ a

$\ell(v, w) = +\infty$ ako $(v, w) \notin E$.

Tako da je $\ell(v, w) \geq 0$ ili $\ell(v, w) = +\infty$.

Uzmimo da niz ivica $e_1, e_2, \ldots, e_m$ čini jedan put. Cijena ili dužina tog puta definiše se kao zbir $c(e_1) + c(e_2) + \ldots + c(e_m)$. Od interesa su što kraći, odnosno što jeftiniji putevi.

Razmotrimo dva vrha $i \in V$ i $j \in V$. Razmotrimo sve moguće puteve u datom grafu G koji vode od i ka j. Sagledajmo dužinu svakog od tih puteva i onda ćemo izdvojiti put čija je ukupna dužina najmanja od svih. Uvodi se oznaka c_{ij} za dužinu najkraćeg puta koji vodi od vrha i ka vrhu j. Ovdje je očito $i \in \{1, 2, \ldots, n\}$ i $j \in \{1, 2, \ldots, n\}$. Takođe, stavlja se da je $c_{ij} = +\infty$ ako u datom grafu ne postoji put koji bi vodio od vrha i ka vrhu j.

Mi želimo da riješimo sljedeći zadatak. Ulazni podaci programa su podaci o ponderisanom usmjerenom grafu G. Sastaviti program koji računa i štampa vrijednosti c_{ij}.

Pogledajmo ideju rješenja. Razmotrimo matrice $C^k = [c_{ij}^k]_{i,j=1}^n$ za $k = 0, 1, \ldots, n$ čiji su članovi definisani na sljedeći način: c_{ij}^k jeste dužina najkraćeg od svih puteva koji vode od v_i ka v_j a čije sve međutačke pripadaju skupu $\{v_1, \ldots, v_k\}$, ako ne postoji nijedan takav put onda neka bude $c_{ij}^k = +\infty$. Algoritam se temelji na sljedećem jednostavnom zapažanju: $c_{ij}^k = \text{MIN}(c_{ij}^{k-1}, c_{ik}^{k-1} + c_{kj}^{k-1})$. Riječima, najkraći put od v_i ka v_j čije sve međutačke pripadaju skupu $\{v_1, \ldots, v_k\}$ jeste kraći od sljedeća dva puta: prvi put i drugi put. Prvi put je najkraći mogući put koji vodi od v_i ka v_j a kao njegove međutačke dopušteni su samo vrhovi iz skupa $\{v_1, \ldots, v_{k-1}\}$. Drugi put nastaje kada se na najkraći mogući put od v_i ka v_k čije međutačke imaju indeks $k - 1$ ili manje nadoveže najkraći mogući put od v_k ka v_j čije međutačke imaju

indeks $k-1$ ili manje. Vidimo da relacija $c_{ij}^k = \text{MIN}(c_{ij}^{k-1}, c_{ik}^{k-1} + c_{kj}^{k-1})$ izražava članove matrice C^k preko članova matrice C^{k-1}. Tokom rada algoritma, broj k se postepeno povećava, odnosno promjenljiva k uzima redom vrijednosti $k = 1$, $k = 2$, ..., $k = n$. Tokom rada algoritma, postepeno se dopušta da sve više i više vrhova grafa mogu biti međutačke puta. Ograničenje na dopuštene međutačke slabi sve više i više. Kada je $k = n$ onda je ograničenje potpuno iščezlo. Drugim riječima, važi jednakost $c_{ij} = c_{ij}^n$, odnosno matrica $C = C^n$ predstavlja rezultat.

Time je algoritam konstruisan, jer se lako definišu vrijednosti c_{ij}^0 u polaznoj matrici C^0, kada nijedan vrh nije dozvoljen kao međutačka. Naime, očito treba staviti $c_{ij}^0 = c(i,j)$ ako $(i,j) \in E$, imamo u vidu put dužine jedan, tj. put koji se sastoji od samo jedne ivice, a treba staviti $c_{ij}^0 = +\infty$ ako $(i,j) \notin E$, već ranije smo vidjeli da izraz "dužina puta jednaka je beskonačno" predstavlja zamjenicu za izraz "put ne postoji", sve ovo u slučaju $i \neq j$. U slučaju $i = j$ treba staviti $c_{ij}^0 = 0$ imajući u vidu put dužine nula od vrha i ka tom istom vrhu i, naravno da je dužina takvog puta nula.

Na redu je Floydov algoritam na nestrogom paskalu:

učitaj podatke o ponderisanom grafu $G = (V, E)$;
for $i, j = 1, \ldots, n$ do if $(i,j) \in E$ then $\ell(i,j) \leftarrow c(i,j)$ else $\ell(i,j) \leftarrow +\infty$;
for $i, j = 1, \ldots, n$, $i \neq j$ do $c_{ij}^0 \leftarrow \ell(i,j)$;
for $i = 1, \ldots, n$ do $c_{ii}^0 \leftarrow 0$;
for $k = 1, \ldots, n$ do
    for $i, j = 1, \ldots, n$ do
    $c_{ij}^k \leftarrow \text{MIN}(c_{ij}^{k-1}, c_{ik}^{k-1} + c_{kj}^{k-1})$;
for $i, j = 1, \ldots, n$ do $c_{ij} \leftarrow c_{ij}^n$;
štampaj rezultate c_{ij}

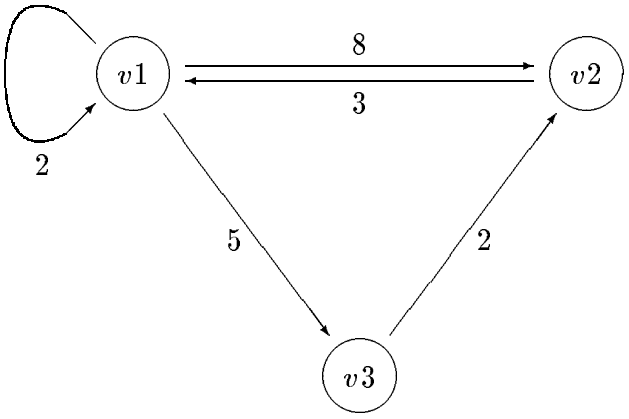
Primjer. Neka bude $n = 3$, tj. $V = \{1,2,3\}$ ili svejedno $V = \{v_1, v_2, v_3\}$ i neka bude $E = \{(1,1), (1,2), (2,1), (1,3), (3,2)\}$ i razmotrimo ponderisani usmjereni graf $G = (V, E)$, gdje je $c(1,1) = 2$, $c(1,2) = 8$, $c(2,1) = 3$, $c(1,3) = 5$ i $c(3,2) = 2$. V. sliku. Primijeniti Floydov algoritam na dati graf, odnosno odrediti dužinu najkraćeg puta od v ka w, za svaki uređeni par vrhova $(v,w) \in V \times V$.

U nastavku su redom prikazani matrica $L = [\ell(i,j)]_{i,j=1}^3$, zatim matrice C^0, C^1 i C^2 koje predstavljaju među–rezultate i na kraju matrica $C^3 = [c_{ij}^3]_{i,j=1}^3$ ili svejedno $C = [c_{ij}]_{i,j=1}^3$, to je rezultat izvršavanja programa.

$$L = \begin{bmatrix} 2 & 8 & 5 \\ 3 & +\infty & +\infty \\ +\infty & 2 & +\infty \end{bmatrix}, \quad C^0 = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & +\infty \\ +\infty & 2 & 0 \end{bmatrix}, \quad C^1 = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ +\infty & 2 & 0 \end{bmatrix},$$

$$C^2 = \begin{bmatrix} 0 & 8 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix}, \quad C^3 = C = \begin{bmatrix} 0 & 7 & 5 \\ 3 & 0 & 8 \\ 5 & 2 & 0 \end{bmatrix}.$$

Kao što je već rečeno, posljednja napisana matrica $\begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$ sadrži rezultate. Upravo, c_{ij} je dužina najkraćeg puta od vrha i ka vrhu j.



Slika: graf G

5.5. ZADATAK O JEDNOM IZVORU

Neka nas interesuju samo najkraći putevi koji svi polaze iz jednog određenog vrha, *izvora*. Ustvari, mogao bi se posmatrati i drugi (još manji) zadatak o određivanju najkraćeg puta od jednog određenog vrha ka drugom određenom vrhu. Međutim, nije poznat algoritam za taj manji zadatak koji bi bio bolji (po kriterijumu najgoreg slučaja) od najboljeg algoritma za zadatak o jednom izvoru.

Algoritam. Najkraći put za jedan izvor, Dajkstrin algoritam (Dijkstra). *Ulaz.* Usmjereni graf $G = (V, E)$, izvor $v_0 \in V$ i funkcija ℓ: $E \to [0, +\infty)$, $\ell(e) \geq 0$ je dužina ili oznaka ivice e. Uzimamo $\ell(v_i, v_j) = +\infty$ ako $(v_i, v_j) \notin E$ i $v_i \neq v_j$. Uzimamo $\ell(v, v) = 0$. *Izlaz.* Za svako $v \in V$, minimum, preko svih puteva P od v_0 ka v, zbira oznaka svih ivica puta P.

Metod. Koriste se dvije osnovne promjenljive: skup $S \subset V$ i niz $D[v]$, gdje je v vrh grafa ($v \neq v_0$). U početku je $S = \{v_0\}$. U svakom koraku skup S se uvećava za jedan element. Algoritam se završava kada postane $S = V$. U početku je $D[v]$, za svako v, jednak cijeni puta (dužine jedan) od v_0 ka v. Uopšte, tokom izvršavanja algoritma, $D[v]$ znači minimalnu cijenu svih puteva od v_0 ka v, a čije su sve međutačke iz skupa S. Jedino ostaje da se kaže kojim redom se elementi uključuju u skup S. Svaki put kada skup S treba da se uveća za jedan element, mi među svim elementima skupa $V \setminus S$ biramo onaj za koji je tekuća vrijednost $D[v]$ minimalna. Time je algoritam opisan.

Slijedi algoritam na nestrogom paskalu:

učitaj ulazne podatke;
definiši vrijednosti $\ell(v_i, v_j)$ za svako $v_i \in V$ i $v_j \in V$;
$S \leftarrow \{v_0\}$;
$D[v_0] \leftarrow 0$;
FOR svako $v \in V \setminus \{v_0\}$ DO $D[v] \leftarrow \ell(v_0, v)$;
WHILE $S \neq V$ DO
    BEGIN
    izaberi u $V \setminus S$ vrh w takav da je $D[w]$ minimalno;
    uključi w u S;
    FOR svako $v \in V \setminus S$ DO
        $D[v] \leftarrow$ MIN$(D[v], D[w] + \ell(w, v))$
    END;
štampaj rezultate $D[v]$, za svako $v \in V$

U trećem redu odozdo ovog teksta može da dođe do promjene $D[v]$. Do promjene dolazi ako je raniji najkraći put od v_0 ka v (kroz stari skup S) duži od puta koji nastaje kada se na najkraći put od v_0 ka w nadoveže ivica (w, v). Do ove promjene može da dođe za $v \in V \setminus S$ jer je $D[w] \leq D[v]$. Do ove promjene ne može da dođe za $v \in S$ (pa se na te vrhove i ne odnosi ovo preračunavanje veličine $D[v]$), u vezi načina na koji se skup S širi. D – distanca. Stavljeno je $D[v_0] = 0$.

Na slici 1 prikazan je izbor vrha w. Na slici 2 prikazano je preračunavanje veličine $D[v]$, v je proizvoljan član skupa $V \setminus S$, gleda se koji od dva puta je kraći.

Primjer. Na slici 3 nacrtan je usmjereni graf čije su ivice označene, tj. čija svaka ivica ima svoju cijenu. U tabeli su prikazani među–rezultati, u posljednjem redu tabele nalaze se rješenja (odgovor). *Odgovor.* Najkraći put od v_0 ka v_1 ima dužinu 2, ka v_2 – 5, ka v_3 – 9 i ka v_4 – 9.

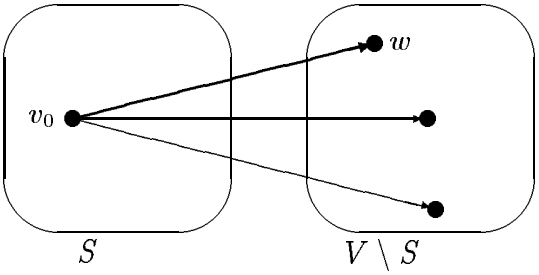
(U primjeru, slučajno se pogodilo da skupu S pristupaju vrhovi u redosljedu "redom" vrh čiji je indeks 1, vrh čiji je indeks 2, vrh čiji je indeks 3, vrh čiji je indeks 4.)

Teorema 1. Dajkstrin algoritam je pravilan, odnosno on saopštava upravo dužinu najkraćeg puta od fiksiranog vrha v_0 ka vrhu v, za svaki vrh v.

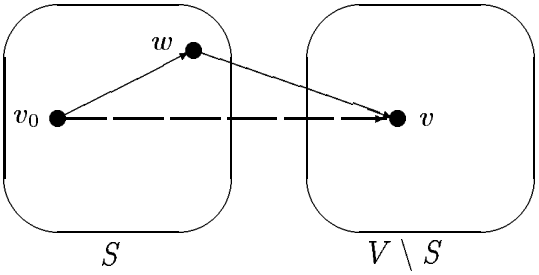
Glavno mjesto u *dokazu* je da se pokaže sljedeće. (a) za $v \in S$, $D[v]$ je dužina najkraćeg puta od v_0 ka v, (b) za $v \in V \setminus S$, $D[v]$ je dužina najkraćeg puta od v_0 ka v koji ima svojstvo da su mu sve tačke iz S, izuzimajući krajnju tačku v. Ovo se dokazuje indukcijom po veličini skupa S, po $\|S\| = \text{card}(S)$.

Teorema 2. Vremenska složenost (najgori slučaj, model RAM, uniformna cijena) razmatranog algoritma iznosi $O(n^2)$, gdje je sa n označen broj vrhova grafa.

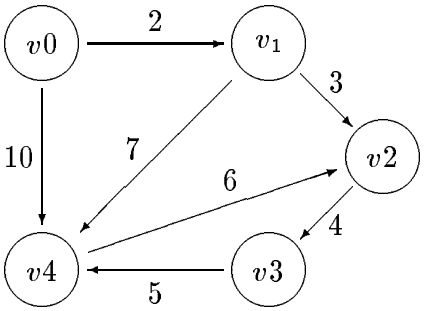
Dokaz. Najviše se troši za petlju FOR pri kraju algoritma. Sama petlja izvrši se reda n puta. U toku jednog izvršavanja petlje obavi se reda n operacija $D[v] \leftarrow \ldots$. Ova operacija zahtijeva konstantno vrijeme. To konstantno vrijeme računa se tako što se saberu: vrijeme potrebno za sabiranje, vrijeme potrebno za MIN i vrijeme potrebno za pronalaženje određenih registara u memoriji (naredbe LOAD i STORE), uključujući indirektno adresiranje za rukovanje sa nizom. Zatim: promjena brojača ciklusa, provjera da li treba da se izađe iz ciklusa i slično. Dokaz je završen.



Slika 1: koji vrh se uključuje u S



Slika 2: $D[v]$ ili $D[w] + \ell(w, v)$



Slika 3: ponderisani graf

Tabela:

iteracija	S	w	$D[w]$	$D[v_1]$	$D[v_2]$	$D[v_3]$	$D[v_4]$
0	$\{v_0\}$			2	$+\infty$	$+\infty$	10
1	$\{v_0, v_1\}$	v_1	2	2	5	$+\infty$	9
2	$\{v_0, v_1, v_2\}$	v_2	5	2	5	9	9
3	$\{v_0, v_1, v_2, v_3\}$	v_3	9	2	5	9	9
4	$\{v_0, v_1, v_2, v_3, v_4\}$	v_4	9	2	5	9	9

5.6. ZADATAK O NEZAVISNOM SKUPU I BACKTRACKING

Neka je $G = (V, E)$ usmjeren graf, gdje je $V = \{v_1, \ldots, v_n\}$ skup koji ima $n \geq 1$ članova. Za skup $S \subset V$ kaže se da je nezavisan ako $v_i \in S$, $v_j \in S \Rightarrow (v_i, v_j) \notin E$. Zadatak o nezavisnom skupu glasi: odrediti nezavisan skup čiji je kardinalni broj najveći mogući.

Uvode se provizorni termini. Za skup $S \subset V$ kaže se da je dopustivo rješenje. Dopustivo rješenje je pravilno ako ono zadovoljava izlazni kriterijum (ako je S nezavisan skup), a inače je nepravilno. Pravilno dopustivo rješenje predstavlja optimalno rješenje ako ono maksimizuje ciljnu funkciju (ako je kardinalni broj $\|S\|$ najveći mogući).

Kao što znamo, varijacija sa ponavljanjem n–te klase od dva elementa jeste bilo koja uređena n–torka $(i_1, i_2, \ldots, i_n)$, gdje $i_k \in \{0, 1\}$ za svako k, dva elementa označeni su kao 0 i 1. Može se kazati da je $(i_1, \ldots, i_k)$ djelimična varijacija; što se tiče k, ovdje je $0 \leq k \leq n$. Postoji uzajamno jednoznačna korespondencija između varijacija i podskupova: odgovarajući podskup za varijaciju $(i_1, i_2, \ldots, i_n)$ je $S = \{v_j | \, 1 \leq j \leq n,\ i_j = 1\}$. Podskupova ima 2^n, a i varijacija isto. Recimo, $(i_1, \ldots, i_n) = (i_1, i_2, i_3, i_4) = (1, 0, 1, 0) \leftrightarrow S = \{v_1, v_3\} \subset \{v_1, v_2, v_3, v_4\} = \{v_1, \ldots, v_n\}$.

Sastaviti program koji štampa sve varijacije n–te klase od dva elementa 0 i 1. U nastavku je dato rješenje (grubi algoritam) na nestrogom jeziku:

for $i_1 = 0$ to 1 do
for $i_2 = 0$ to 1 do
etc.
for $i_n = 0$ to 1 do
write $(i_1, i_2, \ldots, i_n)$

Sastaviti program koji rješava zadatak o nezavisnom skupu. Program treba da odštampa kardinalni broj optimalnog rješenja. Neka se program temelji na tzv. metodi iscrpnog pregledanja, engl. exhaustive search. U nastavku je dat grubi algoritam:

učitaj podatke o grafu: broj n i matricu koincidencije; $max \leftarrow 0$;
for $i_1 = 0$ to 1 do
for $i_2 = 0$ to 1 do
etc.
for $i_n = 0$ to 1 do
if dopustivo rješenje S je pravilno & $\|S\| \geq max$ then $max \leftarrow \|S\|$;
štampaj rezultat max

U pretposljednjem redu, "S je pravilan" znači: ako $v \in S$ i $w \in S$ onda (v, w) nije ivica grafa. Već je rečeno o korespondenciji: n–torki $(i_1, \ldots, i_n)$ odgovara skup S, gdje $v_j \in S$ ako i samo ako je $i_j = 1$ (za svako j).

Sastaviti program 3 koji štampa sve varijacije sa ponavljanjem i sve djelimične varijacije sa ponavljanjem. U nastavku je dat grubi algoritam:

write ();
for $i_1 = 0$ to 1 do begin
write (i_1); for $i_2 = 0$ to 1 do begin
write (i_1, i_2); for $i_3 = 0$ to 1 do begin
write (i_1, i_2, i_3); for $i_4 = 0$ to 1 do begin
etc.
write $(i_1, i_2, i_3, \ldots, i_{n-1})$; for $i_n = 0$ to 1 do
write $(i_1, i_2, i_3, \ldots, i_n)$ ... end end end end

Sastaviti program koji rješava zadatak o nezavisnom skupu. Neka se program temelji na tzv. metodi vraćanja unazad (ili pokušavam i vraćam se), engl. backtracking. Grubi algoritam dobija se kada se u posljednjem programu 3 izvrše prepravljanja.

Iz gramatike programskog jezika: učinak naredbe `skip` (ili `skip one instruction`) jeste da se jedna naredba u nizu naredbi preskoči. Recimo, ako u programu piše naredbaa; `skip`; naredbab; naredbac onda će naredbab biti preskočena (ona neće biti izvršena).

U programu broj 4 uvedena je skraćenica d d r za djelimično dopustivo rješenje:

učitaj podatke o grafu: broj n i matricu koincidencije; $max \leftarrow 0$;
for $i_1 = 0$ to 1 do begin
if d d r (i_1) nije pravilno then skip; for $i_2 = 0$ to 1 do begin
if d d r (i_1, i_2) nije pravilno then skip; for $i_3 = 0$ to 1 do begin
if d d r (i_1, i_2, i_3) nije pravilno then skip; for $i_4 = 0$ to 1 do begin
etc.
if d d r $(i_1, i_2, i_3, \ldots, i_{n-1})$ nije pravilno then skip; for $i_n = 0$ to 1 do
if d r $(i_1, \ldots, i_n)$ je pravilno & $\|S\| > max$ then $max \leftarrow \|S\|$ ... end end end end;
štampaj rezultat max

Kao što je već rečeno, za skup S kaže se da je pravilan (kaže se da skup S čine međusobno nezavisni vrhovi) ako: $v \in S$ i $w \in S$ povlači da (v, w) nije ivica grafa.

Vidimo da se vrši "skip" kada podskup koji odgovara k–torki $(i_1, \ldots, i_k)$ nije pravilan, tj. kada postoji ivica u grafu između neka dva člana tog podskupa. Zato što i svaki širi (srodan) podskup nije pravilan, imamo u vidu podskup kome odgovara n–torka oblika $(i_1 \ldots., i_k, \ldots)$. Preskače se petlja po i_{k+1}, tj. preskaču se petlje po $i_{k+1}, \ldots, i_n$.

Backtracking predstavlja nešto usavršeni oblik od exhaustive search. U slučaju exhaustive search, redom se generišu sva dopustiva rješenja i gleda se izlazni kriterijum (a eventualno i ciljna funkcija). U slučaju backtracking, generišu se i djelimična dopustiva rješenja. Već za djelimično dopustivo rješenje ispituje se da li ono zadovoljava uslov, odnosno da li je ono pravilno. Ako nije pravilno onda odbaciti sva dopustiva rješenja koja su sa njim srodna (koja su mu slična). Tako da očito dolazi do smanjivanja ukupnog broja pregledanja koja treba da budu izvršena.

Ako je $n = 4$ onda program 3 odštampa ukupno 31 riječ. Sve te riječi prikazane su na sljedećoj slici. Pored svake riječi stavljen je broj (1 ili 2 ili ... ili 31) da prikaže redosljed u štampanju. Sve riječi organizovane su u jedmo drvo. To je tzv. backtracking drvo.

Na slici je prikazano drvo mogućnosti i djelimičnih mogućnosti.

Neka je $(i_1, \ldots, i_k)$ jedno djelimično dopustivo rješenje. Njemu odgovara jedan vrh u drvetu. Ako su okolnosti takve da u tom vrhu nije zadovoljen uslov onda nema svrhe da se ide naprijed (da se ispituju srodna rješenja). Na tom mjestu, drvo se podsječe, skrati. Treba odstupiti za jedan korak, treba se vratiti nazad u drvetu. Od ove predstave dolazi riječ backtracking. Isto se kaže i engl. branch–and–bound method, metoda grananja i granica.

Predloženi algoritam (tipa backtracking) za zadatak o nezavisnom skupu može da bude poboljšan, odnosno njegova vremenska složenost može da bude smanjena time što će uslov koji se ispituje biti postrožen, drvo će biti podsječeno još više. Prilikom formiranja uslova, neka se uzimaju u obzir i okolnosti koje su tokom izvršavanja programa postale poznate. Dakle, označimo sa $(i_1, \ldots, i_k)$ jedno djelimično dopustivo rješenje. Odgovara mu podskup S za čiji broj članova važi $\|S\| \leq k$. Uzmimo da je podskup S pravilan (nezavisan), tj. da za bilo koja dva njegova člana, u datom grafu, ne postoji ivica koja bi spajala ta dva člana. Treba dodati još $n - k$ komponenti $i_{k+1}, \ldots, i_n$ da se djelimično dopustivo rješenje dopuni do dopustivog rješenja. Ako $n - k$ članova mogu da budu dodati skupu S bez narušavanja pravilnosti onda bi nastao podskup čiji je kardinalni broj jednak $\|S\| + n - k$. Uporediti sa tekućom vrijednošću max. Ako je $\|S\| + n - k \leq max$ onda nema svrhe da se ispituje odgovarajuće poddrvo (da se ispituju srodna rješenja).

Tehnika ili metoda ili opšta metoda backtracking može da bude primijenjena na raznorazne zadatke. Koriste se razne ideje da se uoče (i eliminišu) nekorisna pregledanja.

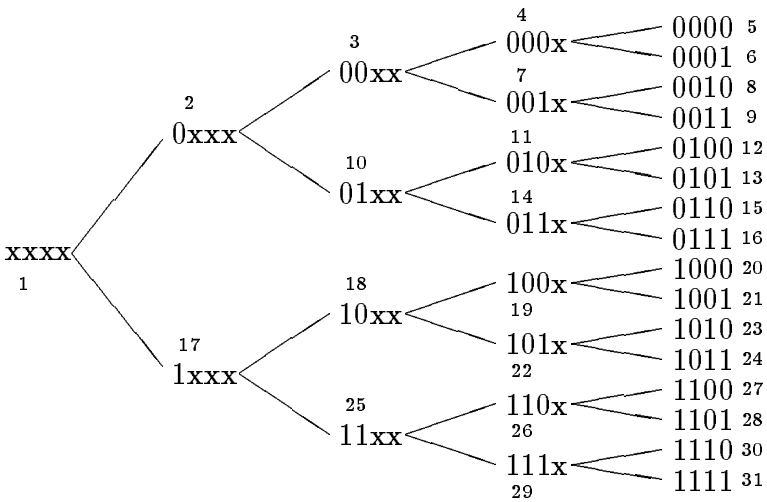
U zaključku, i exhaustive search algoritam i backtracking algoritam imaju eksponencijalnu složenost, po pravilu. A backtracking ipak ima znatno manju složenost od exhaustive search. Umjesto exhaustive search češće se kaže metoda grube sile, engl. brute–force, sa ekvivalentnim značenjem.

Ako upoređujemo vremenske složenosti u tri slučaja onda možemo kazati: složenost efikasnog algoritma $\ll$ složenosti backtracking algoritma $\ll$ složenosti algoritma tipa brute–force.

Tako da se tehnika backtracking primjenjuje po pravilu kao krajnje sredstvo, kada nema nikakvog boljeg izlaza (iole prihvatljivog izlaza, metode). Kaže se da je riječ backtracking skoro sinonim za program koji je spor, čije izvršavanje traje puno vremena.

Obično se tehnika backtracking primjenjuje na NP–kompletne zadatke.

Vraćajući se zadatku o nezavisnom skupu, može se pokazati da je broj nezavisnih skupova u grafu sa n vrhova jednak $O(n^{\ln n})$, ako se pretpostavi da su za bilo koja dva vrha jednaki izgledi (šanse) da za ta dva vrha postoji ili ne postoji ivica. Zatim se na osnovu toga može pokazati da je vremenska složenost backtracking algoritma (program broj 4) jednaka takođe $T(n) = O(n^{\ln n})$, očekivani slučaj. S druge strane, vremenska složenost exhaustive search algoritma jednaka je $T(n) = O(2^n)$. Lako se vidi da je $n^{\ln n} \ll 2^n$ ("znatno manje") kad $n \to \infty$, u smislu $\lim_{n\to\infty} n^{\ln n}/2^n = 0$.



Da sve bude razjašnjeno, u slučaju primjera radi $n = 6$, program broj 4 glasi kako slijedi:
učitaj matricu koincidencije; $max \leftarrow 0$;
for $i_1 = 0$ to 1 do begin
if not (i_1) je pravilno then skip; for $i_2 = 0$ to 1 do begin
if not (i_1, i_2) je pravilno then skip; for $i_3 = 0$ to 1 do begin
if not (i_1, i_2, i_3) je pravilno then skip; for $i_4 = 0$ to 1 do begin
if not (i_1, i_2, i_3, i_4) je pravilno then skip; for $i_5 = 0$ to 1 do begin
if not $(i_1, i_2, i_3, i_4, i_5)$ je pravilno then skip; for $i_6 = 0$ to 1 do
if $(i_1, i_2, i_3, i_4, i_5, i_6)$ je pravilno & $\|S\| > max$ then $max \leftarrow \|S\|$ end end end end end;
štampaj rezultat max

Znamo da $(i_1, \ldots, i_k)$ je pravilno znači: ako $v \in S$ i $w \in S$ onda (v, w) nije ivica. Znamo da S znači sljedeći skup vrhova: $S = \{v_j | \, 1 \le j \le k, \, i_j = 1\}$. Djelimična varijacija i podskup odgovaraju jedno drugom.

5.7. ZADATAK O TRGOVAČKOM PUTNIKU I BACKTRACKING

Razmotrimo neusmjeren graf $G = (V, E)$ sa $n \geq 1$ vrhova, $V = \{v_1, v_2, \ldots, v_n\}$. Neka je graf potpun, tj. neka su bilo koja dva vrha povezani ivicom. Uzmimo još da je graf težinski (ponderisan): svakoj ivici (i, j) pridružena je njena težina ili dužina ili cijena u oznaci $c(i, j)$, gdje je $c(i, j) \geq 0$. Zadatak o trgovačkom putniku glasi: dat je graf, odrediti optimalnu maršrutu (optimalan put, optimalan ciklus) ili barem odrediti cijenu (dužinu) optimalnog ciklusa. Naime, trgovački putnik treba da krene iz nekog vrha i onda da obiđe sve vrhove (da tačno jednom posjeti svaki vrh, svaki grad) i još da se na kraju vrati u vrh iz koga je krenuo.

Znamo da se permutacija od n elemenata definiše kao bilo koja uređena n–torka $(i_1, i_2, \ldots, i_n)$, gdje je $i_j \neq i_k$ za $j \neq k$ i $i_j \in \{1, 2, \ldots, n\}$ za svako j, uzeto je da n elemenata budu $1, 2, \ldots, n$. Znamo da takvih permutacija $p = (i_1, i_2, \ldots, i_n)$ ima na broju $n!$ ukupno.

Postoji uzajamno jednoznačna korespondencija između skupa svih mogućih permutacija i skupa svih mogućih maršruta (ciklusa) za trgovačkog putnika. Zaista, permutaciji $p = (i_1, i_2, \ldots, i_n)$ neka odgovara obilazak kako slijedi: kreni iz vrha i_1, zatim pređi u vrh i_2 uz korišćenje ivice (i_1, i_2) očito, itd., zatim pređi u vrh i_n uz korišćenje ivice (i_{n-1}, i_n) i na kraju se preko (i_n, i_1) vrati u početni vrh i_1. $p \leftrightarrow$ cycle. Cijena ciklusa u oznaci c(cycle) definiše se kao $c(\text{cycle}) = c(i_1, i_2) + c(i_2, i_3) + \ldots + c(i_{n-1}, i_n) + c(i_n, i_1)$, n sabiraka. Već je rečeno da treba pronaći ciklus cycle čija je ukupna cijena c(cycle) najmanja moguća.

Uvode se provizorni termini. Za svaki ciklus koji obuhvata sve vrhove grafa kaže se da je dopustivo rješenje. Za dopustivo rješenje kaže se da je optimalno rješenje ako ono minimizuje ciljnu funkciju (ako je njegova ukupna dužina najmanja moguća).

Mogu se posmatrati i djelimične permutacije $p = (i_1, \ldots, i_k)$. Što se tiče k, ovdje je $0 \leq k \leq n$. Djelimičnoj permutaciji na očigledan način odgovara jedan djelimični ciklus (djelimična maršruta): kreni iz i_1, zatim pređi u i_2, itd., zatim pređi u i_k. Prirodno je definisati cijenu tog djelimičnog ciklusa (tog djelimičnog rješenja) kao $c(\text{cycle}) = c(i_1, i_2) + \ldots + c(i_{k-1}, i_k)$.

Na primjer, djelimična permutacija $p = (i_1, i_2, i_3)$ dopuniće se do potpune permutacije kada se dopišu $i_4, \ldots, i_n$, gdje je $(i_4, \ldots, i_n)$ jedna permutacija elemenata skupa $\{1, 2, \ldots, n\} \setminus \{i_1, i_2, i_3\}$.

Sastaviti program koji štampa sve permutacije od n elemenata, kao i sve djelimične permutacije. U nastavku je dat grubi algoritam:

učitaj n;
write ();
for $i_1 = 1$ to n do begin
write (i_1); for $i_2 = 1$ to n, $i_2 \neq i_1$ do begin
write (i_1, i_2); for $i_3 = 1$ to n, $i_3 \neq i_1, i_2$ do begin
write (i_1, i_2, i_3); for $i_4 = 1$ to n, $i_4 \neq i_1, i_2, i_3$ do begin
etc.
write $(i_1, i_2, \ldots, i_{n-1})$; for $i_n = 1$ to n, $i_n \neq i_1, i_2, \ldots, i_{n-1}$ do
write $(i_1, \ldots, i_n)$ ... end end end end;
štampaj da nema više, da je završeno

Ako je $n = 4$ onda program odštampa ukupno 65 riječi ili 65 redova ili 65 permutacija i djelimičnih permutacija. Sve te riječi prikazane su na sljedećoj slici u obliku tzv. backtracking drveta. Pored svake riječi stavljen je broj (1 ili 2 ili ... ili 65) da se pokaže redosljed štampanja. Na slici piše recimo 23xx. Time je označena djelimična permutacija $(i_1, i_2) = (2, 3)$. Pored je stavljen broj 24.

Na slici je prikazano drvo mogućnosti i djelimičnih mogućnosti.

Backtracking algoritam za rješavanje zadatka o trgovačkom putniku temelji se na sljedećem jednostavnom zapažanju. Neka je izvršavanje algoritma u toku. To znači da su neka dopustiva rješenja već pregledana a neka će tek doći na red. Među svim dosad pregledanim dopustivim rješenjima, neko od njih je najbolje (privremeno optimalno rješenje). Njegova ukupna cijena $c = c(\text{cycle})$ je naravno izračunata. U datom trenutku imamo $min = c(\text{cycle})$. Dakle, u programu je prisutna promjenljiva min čija je tekuća vrijednost uvijek jednaka dužini najkraćeg dopustivog rješenja dosad otkrivenog (dužini najkraćeg među dosad pregledanim dopustivim rješenjima). Toj promjenljivoj kao početnu vrijednost treba dodijeliti $min = +\infty$. Kada se pronađe bolje (kraće) dopustivo rješenje onda se vrijednost min promijeni (smanji). Neka je tokom izvršavanja programa u datom trenutku na red došla djelimična permutacija $p = (i_1, \ldots, i_k)$, odnosno na red je došlo odgovarajuće djelimično rješenje (djelimični ciklus) cycle. Izračunaj $c(\text{cycle}) = c(i_1, i_2) + \ldots + c(i_{k-1}, i_k)$ i uporedi po veličini sa min. Ako je uslov $c(\text{cycle}) < min$ ispunjen onda nastavi sa pregledanjem, djelimičnu permutaciju uvećaj za jednu komponentu, odnosno djelimičnoj permutaciji dodaj–izaberi njenu $(k+1)$–vu komponentu. A ako je pak suprotno $c(\text{cycle}) \geq min$ onda očito treba odbaciti tekuće djelimično rješenje, kao i sva potencijalna rješenja (djelimična i potpuna) koja su mu srodna, koja su sa njim slična, tj. imaju sa njim prvih k komponenti zajedničkih. Jer je njihova ukupna cijena (n sabiraka) očito veća ili jednaka od cijene tekućeg djelimičnog rješenja, od $c(\text{cycle})$, a tim prije je veća ili jednaka od cijene tekućeg (privremenog) optimalnog rješenja, od min. Dakle, u slučaju $c(\text{cycle}) \geq min$ treba zamijeniti k–tu komponentu (posljednju komponentu) u tekućem djelimičnom rješenju njenim sljedećim izborom, a prvih $k-1$ komponenti ostaviti po starom, po mogućnosti, tj. ako sljedeći izbor postoji. Drugim riječima, treba odstupiti za jedno mjesto, backtrack. Dakle, vidimo da sljedeći korak (kako će program da nastavi) zavisi od tekućih okolnosti koje su se stekle. Kriterijum za izbor sljedećeg koraka glasi: da li je $c(\text{cycle}) < min$ ili je pak suprotno $c(\text{cycle}) \geq min$.

Ukupno, vidimo da tokom izvršavanja programa neće biti pregledano čitavo backtracking drvo, već će neki njegovi djelovi (poddrveta) biti odbačeni, neke grane se podsijecaju. Tako da je vremenski trošak izvršavanja programa manji, u odnosu na program koji bi bio temeljen na iscrpnom pregledanju, exhaustive search.

U grubom algoritmu koji slijedi uvedena je oznaka $c(i_1, \ldots, i_k) = \sum_{j=1}^{k-1} c(i_j, i_{j+1})$. Slično, $c(i_1, i_2, \ldots, i_n, i_1) = \sum_{j=1}^{n-1} c(i_j, i_{j+1}) + c(i_n, i_1)$, ima n sabiraka.

Sastaviti program koji rješava zadatak o trgovačkom putniku. Neka se program temelji na tzv. metodi vraćanja unazad, backtracking. U nastavku je dat grubi algoritam.

Znamo da naredba `skip` ima smisao: preskoči jednu naredbu. Neka je $1 \leq k \leq n-1$. Naredba koja će eventualno biti preskočena (neće biti izvršena) jeste naredba for $i_{k+1} = \ldots$. Zapaziti da ta naredba obuhvata naredbe for $i_{k+2} = \ldots, \ldots$, for $i_n = \ldots$, one su njen "podskup". Tako da će u tom slučaju i sve te naredbe biti preskočene.

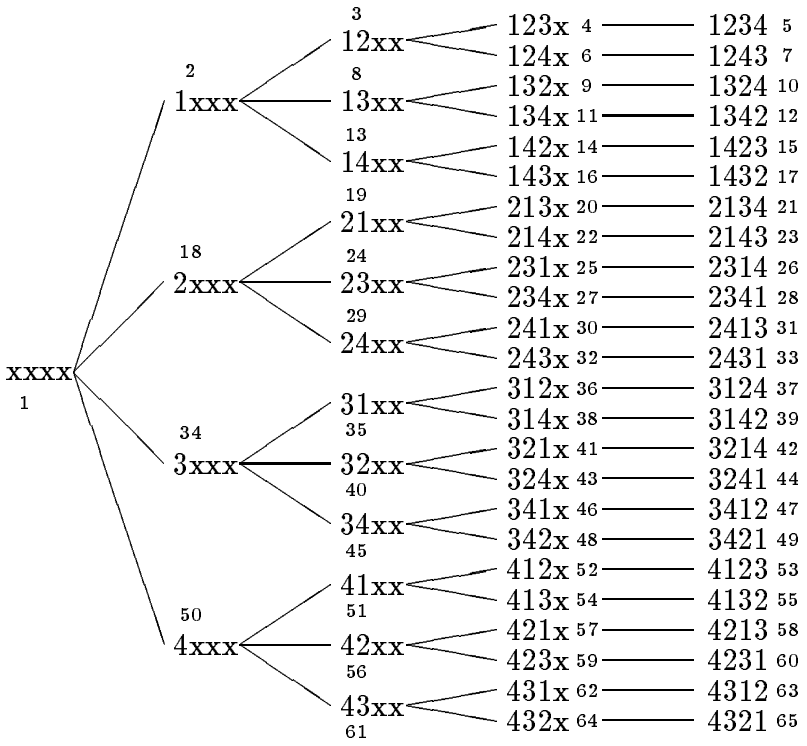
učitaj podatke o grafu: broj n i cijene ivica $c(i,j)$;
$min \leftarrow +\infty$;
for $i_1 = 1$ to n do begin
if $c(i_1) \geq min$ then skip; for $i_2 = 1$ to n, $i_2 \neq i_1$ do begin
if $c(i_1, i_2) \geq min$ then skip; for $i_3 = 1$ to n, $i_3 \neq i_1, i_2$ do begin
if $c(i_1, i_2, i_3) \geq min$ then skip; for $i_4 = 1$ to n, $i_4 \neq i_1, i_2, i_3$ do begin
etc.
if $c(i_1, i_2, \ldots, i_{n-1}) \geq min$ then skip; for $i_n = 1$ to n, $i_n \neq i_1, i_2, \ldots, i_{n-1}$ do
if $c(i_1, i_2, \ldots, i_n, i_1) < min$ then $min \leftarrow c(i_1, i_2, \ldots, i_n, i_1)$ ... end end end end;
štampaj rezultat min
Petlja po i_2 ima $n-1$ iteracija, petlja po i_3 ima $n-2$ iteracija, itd.

Na redu je dopuna.

Predloženi algoritam može da bude poboljšan ako bi se u ranoj fazi izvršavanja programa naišlo na ciklus koji je prilično jeftin, čija je dužina dosta kratka. Tada bi podsijecanje drveta bilo više izraženo, vremenski trošak se smanjuje. U tom cilju treba na početku programa malo dodati. Predvidjeti neki prilično grub i prost postupak za pronalaženje jednog dosta dobrog dopustivog rješenja. Označimo to potencijalno rješenje kao cycle i označimo sa c(cycle) njegovu dužinu. U programu, naredbu $min \leftarrow +\infty$ zamijeniti sa $min \leftarrow c$(cycle).

Na redu je zaključak.

Poznato je da je zadatak o trgovačkom putniku jedan NP–kompletan zadatak. U slučaju rješavanja NP–kompletnog zadatka, mi ne tražimo puno od programa koji se predlaže. Ne postavljamo velike zahtjeve u pogledu vremenskog troška. U tom smislu se ovdje predloženi program zasnovan na metodi "backtracking" može smatrati prihvatljivim.



Da sve bude razjašnjeno, na primjer u slučaju $n = 6$, "backtracking" program glasi kako slijedi:

učitaj podatke o grafu: broj n i cijene ivica $c(i, j)$;
$min \leftarrow +\infty$;
for each $i_1 \in \{1, \ldots, 6\}$ do begin
if $c(i_1) \geq min$ then skip; for each $i_2 \in \{1, \ldots, 6\} \setminus \{i_1\}$ do begin
if $c(i_1, i_2) \geq min$ then skip; for each $i_3 \in \{1, \ldots, 6\} \setminus \{i_1, i_2\}$ do begin
if $c(i_1, i_2, i_3) \geq min$ then skip; for each $i_4 \in \{1, \ldots, 6\} \setminus \{i_1, i_2, i_3\}$ do begin
if $c(i_1, i_2, i_3, i_4) \geq min$ then skip; for each $i_5 \in \{1, \ldots, 6\} \setminus \{i_1, i_2, i_3, i_4\}$ do begin
if $c(i_1, i_2, i_3, i_4, i_5) \geq min$ then skip; for each $i_6 \in \{1, \ldots, 6\} \setminus \{i_1, i_2, i_3, i_4, i_5\}$ do
if $c(i_1, i_2, i_3, i_4, i_5, i_6, i_1) < min$ then $min \leftarrow c(i_1, i_2, i_3, i_4, i_5, i_6, i_1)$ end end end end end;
štampaj rezultat min

Iz gramatike neformalnog jezika znamo da (na primjer) dio teksta programa begin naredba1; naredba2 end predstavlja jednu naredbu, slično sastavljenoj naredbi u paskalu.

6. ALGORITMI U TEORIJI BROJEVA I U KRIPTOGRAFIJI

6.1. EUKLIDOV ALGORITAM I NJEGOVA SLOŽENOST

Možda najstariji algoritam koji postoji u matematici (iz godine 300 p.n.e. približno), a i dan–danas se koristi. Euklidov algoritam služi za nalaženje najvećeg zajedničkog djelioca (NZD) dva prirodna broja a i b. Biće izložen algoritam. Zatim će biti pokazana ispravnost (korektnost) algoritma: biće dokazano da se u rezultatu primjene predloženog postupka kao odgovor dobija upravo NZD(a, b). Na kraju će biti određena vremenska složenost Euklidovog algoritma: vidjećemo da algoritam ima polinomsku složenost, tj. da je efikasan.

Na nestrogom paskalu algoritam se izražava kako slijedi:

read(a, b);
if $a < b$ then neka a i b zamijene mjesta;
1: $c \leftarrow a \bmod b$;
if $c > 0$ then begin $a \leftarrow b$; $b \leftarrow c$; goto 1 end;
if $c = 0$ then write(b)

Riječima, treba naći NZD = NZD(a, b) za date brojeve a i b, gdje je $a \geq 1$, $b \geq 1$ i $a \geq b$. U prvom koraku izvrši se dijeljenje $a : b$ čime se dobijaju količnik k i ostatak c. Važi jednakost $a = kb + c$. Očito je $0 \leq c \leq b - 1$. Razlikujemo dva slučaja $c > 0$ i $c = 0$. Uzmimo da je $c > 0$. Euklidov algoritam temelji se na relaciji NZD(a, b) = NZD(b, c) čija će istinitost biti dokazana u nastavku. Tako da se onda po istom postupku nastoji da se pronađe NZD(b, c), algoritam je rekurzivan. Budući da je par brojeva koji stoji pod znakom NZD iz koraka u korak sve manji i manji to će algoritam dati rezultat u konačno mnogo koraka. Kada će biti obustavljeno uzastopno računanje količnika i ostatka (manji broj b preuzima ulogu većeg broja a, dok ostatak c preuzima ulogu manjeg broja b)? Biće obustavljeno kada se ispostavi da je ostatak jednak nuli. Time je završena analiza slučaja $c > 0$. Uzmimo sada da je $c = 0$ (drugi slučaj). Znači da je a djeljivo sa b bez ostatka, tj. da je $a = kb$ za neko $k \geq 1$. Očito je NZD$(a, b) = b$. Tako da se b saopštava kao rezultat i računar se zaustavlja. Algoritam je izložen.

Pogledajmo relaciju $a = kb + c$. Za pripremu, pogledajmo sljedeće jednostavne iskaze. Ako su a i b parni onda je i c paran. Ako su b i c parni onda je i a paran. Ako je NZD(a, b) paran onda je i NZD(b, c) paran, kao i obrnuto.

Umjesto a je paran piše se $2 \mid a$, ima smisao a je djeljivo sa 2 (bez ostatka). Č. 2 dijeli a.

Iz $a = kb + c$ imamo. Neka je x ma koji broj (maločas je bilo $x = 2$). Ako $x \mid a$ i $x \mid b$ onda $x \mid c$. Ako $x \mid b$ i $x \mid c$ onda $x \mid a$. Prema tome, par brojeva (a, b) i par brojeva (b, c) imaju iste činioce, imaju sve činioce zajedničke. To upravo znači da je NZD(a, b) = NZD(b, c). Korektnost algoritma je dokazana.

Navedimo primjer. Naći NZD$(120, 26)$. Izrada: $a = 120$, $b = 26$

120 : 26 = količnik 4, ostatak 16 ($a_0 = 120$, $b_0 = 26$, $c_0 = 16$)
26 : 16 = količnik 1, ostatak 10 ($a_1 = 26$, $b_1 = 16$, $c_1 = 10$)
16 : 10 = količnik 1, ostatak 6 ($a_2 = 16$, $b_2 = 10$, $c_2 = 6$)
10 : 6 = količnik 1, ostatak 4 ($a_3 = 10$, $b_3 = 6$, $c_3 = 4$)
6 : 4 = količnik 1, ostatak 2 ($a_4 = 6$, $b_4 = 4$, $c_4 = 2$)
4 : 2 = količnik 2, ostatak 0 ($a_5 = 4$, $b_5 = 2$, $c_5 = 0$)

NZD = 2 (NZD = b_5) Odgovor glasi NZD$(120, 26) = 2$.

Na redu je vremenska složenost algoritma. U našem primjeru je $a_0 > a_1 > a_2 > \ldots$. Očito je da isto važi i u opštem slučaju. U našem primjeru je uvijek $a_{i+2} \leq a_i/2$. Pokažimo da i u opštem

slučaju važi $a_{i+2} \leq a_i/2$ za svako i. Zaista, razlikujemo dvije mogućnosti $k_i = 1$ naspram $k_i > 1$ ili svejedno $b_i > a_i/2$ naspram $b_i \leq a_i/2$. Podrazumijeva se relacija $a_i = k_i b_i + c_i$.

Ako je $k_i = 1$ onda imamo $a_i = b_i + c_i$ i $b_i > c_i$, tako da je $c_i < a_i/2$. U slučaju $k_i > 1$, tj. u slučaju $b_i \leq a_i/2$ imamo $c_i < a_i/2$ jer je ostatak c_i sigurno manji od djelioca b_i.

Pokazano je da važi $a_{i+2} \leq a_i/2$. Poslije 2, 4, 6, 8, ... koraka djeljenik se svede na $a/2$, $a/4$, $a/8$, $a/16$, ... ili se još bolje svede. Lako je odrediti kada će se on svesti do jedinice i u najgorem slučaju. Broj koraka u algoritmu za računanje NZD(a, b), gdje je $a \geq b$, je manji ili jednak od $2[\log_2 a] + 1$.

Broj a ima $[\log_2 a] + 1$ binarnih cifara, odnosno $[\log a] + 1$ dekadnih. Pravilno je da se dimenzioni broj zadatka ili mjera obima ulaza n definiše kao broj cifara broja a ($a \geq b$) u binarnom ili u dekadnom sistemu. Eventualno bi moglo n = broj cifara broja a + broj cifara broja b.

Vidjeli smo da je broj koraka algoritma jednak $2n$. Vremenska složenost $T(n)$ dobiće se kada se $2n$ pomnoži sa troškom za pojedini korak.

Glavnina troška u jednom koraku vezana je za operaciju računanja ostatka $c_i \leftarrow a_i \bmod b_i$. Pravilno je da se trošak za tu operaciju smatra jednakim broju cifara prvog argumenta a_i plus broj cifara drugog argumenta b_i, što je u skladu sa logaritamskim kriterijumom za RAM. Dakle $n + n = 2n$. Sa povećavanjem i broj cifara opada, ali mi ocjenjujemo složenost sa gornje strane, mi želimo da dobijemo $T(n) \leq \ldots$.

Sve u svemu, $2n$ koraka, $2n$ ili eventualno $const \cdot n$ vremenskih jedinica po koraku, tako da je $T(n) = 2n \cdot 2n = 4n^2$ ili $T(n) = O(n^2)$. Zaključak: složenost algoritma je kvadratna, složenost algoritma je polinomska, algoritam je efikasan.

(Def. NZD$(n_1, n_2) = \max\{k |\ k \mid n_1 \ \&\ k \mid n_2\}$ $(n_1, n_2 \geq 1)$.)

6.2. NEKE TEOREME IZ TEORIJE BROJEVA

Na početku navodimo osnovne definicije iz teorije brojeva. Za broj $k < n$ kaže se da je svojstveni činilac broja n ako se prilikom dijeljenja $n : k$ kao ostatak dobija nula. Za prirodan broj $n \geq 2$ kaže se da je prost ako on nema svojstvenih činilaca osim $k = 1$. U suprotnom slučaju kaže se da je n složen. Za $n = 1$ ne kaže se ni da je prost ni da je složen.

Na početku navodimo dva temeljna tvrđenja iz teorije brojeva. Osnovna teorema aritmetike: prirodan broj n može na jedinstven način da se prikaže kao proizvod svojih prostih činilaca. Upravo, jedinstveno su određeni prosti brojevi $p_1 < p_2 < \ldots < p_i$ i brojevi $e_1 \geq 1$, $e_2 \geq 1$, ..., $e_i \geq 1$ takvi da je $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_i^{e_i}$. Recimo, $120 = 2^3 \cdot 3 \cdot 5$.

Teorema o raspodjeli prostih brojeva. Neka je $\pi(x)$ po definiciji broj prostih brojeva koji su manji ili jednaki od x. Tada važi relacija $\pi(x) \sim \frac{x}{\ln x}$ kad $x \to \infty$. Recimo, $\pi(1000) = \text{card}\{p |\ p \text{ je prost i } p \leq 1000\} \approx \frac{1000}{\ln 1000} = 145$.

U ovom naslovu biće riječi o dvije teoreme iz teorije brojeva: o maloj Fermaovoj teoremi i o teoremi o ostacima. Te teoreme služe kod zasnivanja algoritama koji se koriste u kriptografiji.

Mala Fermaova teorema (Fermat). Neka je p prost broj i neka je a broj koji nije djeljiv sa p. Tada važi $a^{p-1} \equiv 1 \pmod p$.

Znamo da oznaka $x \equiv y \pmod n$ po definiciji ima smisao: ostatak pri dijeljenju $x : n$ jednak je ostatku pri dijeljenju $y : n$, kaže se da je x kongruentno sa y po modulu n ili da su x i y ekvivalentni po modulu n. Na primjer, $45 \equiv 17 \pmod 7$ je istinito.

Pogledajmo primjer. Uzmimo da je $p = 7$. Teorema tvrdi da se prilikom dijeljenja $a^6 : 7$ uvijek dobije 1 kao ostatak, za bilo koje $1 \leq a \leq 6$, $8 \leq a \leq 13$, ... Provjerimo za $1 \leq a \leq 6$:

$a = 1$, $a^6 - 1 = 0$ je djeljivo sa 7, $a = 2$, $a^6 - 1 = 63$ je djeljivo sa 7, $a = 3$, $a^6 - 1 = 728$ je djeljivo sa 7, $a = 4$, $a^6 - 1 = 4095$ je djeljivo sa 7, $a = 5$, $a^6 - 1 = 15624$ je djeljivo sa 7, $a = 6$, $a^6 - 1 = 46655$ je djeljivo sa 7.

Dokaz teoreme. Posmatrajmo brojeve $a, 2a, 3a, \ldots, (p-1)a$. Podijelimo svaki broj iz tog spiska sa p i uočimo odgovarajuće ostatke $a \bmod p, 2a \bmod p, 3a \bmod p, \ldots, (p-1)a \bmod p$. Svaki ostatak je $\neq 0$ i $< p$. Ukupno ima na broju $p-1$ ostataka. Dokažimo da su dva po dva ostatka međusobno različiti. U tom cilju, dopustimo da postoje brojevi r i s ($1 \leq r \leq p-1$, $1 \leq s \leq p-1$, $r \neq s$) takvi da je $ra \bmod p = sa \bmod p$, slijedi $ra - sa = kp$ za neko k,

$$(r-s)a = kp.$$

Broj $r - s$ nije djeljiv sa p. Kako a nije djeljiv sa p po uslovu, to znači da njihov proizvod $(r-s)a$ nije djeljiv sa p (jer je p prost). Lijeva strana $(r-s)a$ nije djeljiva sa p, a desna kp jeste, kontradikcija. Mi smo dokazali da su dva po dva ostatka međusobno različiti. Znači da su ostaci jednaki $1, 2, \ldots, p-1$ u nekom redosljedu. Napišimo odgovarajuće relacije i zatim ih pomnožimo:

$$a \bmod p = \ldots,\ 2a \bmod p = \ldots,\ 3a \bmod p = \ldots,\ \ldots,\ (p-1)a \bmod p = \ldots$$

$$a \cdot 2a \cdot 3a \cdot \ldots \cdot (p-1)a \bmod p = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot (p-1) \bmod p$$

$$a^{p-1} \cdot (p-1)! \bmod p = (p-1)! \bmod p,\ \text{ drukčije zapisano } (a^{p-1} \cdot (p-1)! - (p-1)!) \equiv 0 \,(\bmod p)$$

$$a^{p-1} \bmod p = 1 \bmod p,\ \text{ drukčije zapisano } (a^{p-1} - 1) \equiv 0 \,(\bmod p)$$

U posljednjem koraku, mi smo skratili lijevu i desnu stranu jednakosti sa $(p-1)!$ što je ispravno, budući da je broj $(p-1)!$ uzajamno prost sa p (jer je p prost). Mi smo dokazali da je $a^{p-1} \bmod p = 1$. Teorema je dokazana.

Mala Fermaova teorema predstavlja specijalan slučaj sljedeće teoreme:

Ojlerova teorema (Euler). Neka je $n \geq 2$ prirodan broj i neka je a ma koji prirodan broj koji je uzajamno prost sa n. Tada važi $a^{\varphi(n)} \equiv 1 \,(\bmod n)$.

Dva broja su uzajamno prosti znači po definiciji da oni nemaju zajedničkih činilaca, odnosno znači da je njihov NZD $= 1$.

Veličina $\varphi(n)$ je tzv. Ojlerova funkcija, a definiše se za $n \geq 1$ na sljedeći način: $\varphi(n)$ govori koliko ima brojeva $k \leq n$ takvih da su k i n uzajamno prosti. Primjera radi $\varphi(6) = 2$ (jer $k = 1$ i $k = 5$). Primjera radi $\varphi(120) = 32$. Ako je n prost onda je $\varphi(n) = n - 1$. Definicija iskazana formulom: $\varphi(n) = \#\{k | \, 1 \leq k \leq n, \text{NZD}(k, n) = 1\}$. Piše se card ili $\#$. Specijalno, $\varphi(1) = 1$.

Ovu teoremu navodimo bez dokaza.

Prelazimo na tzv. kinesku teoremu o ostacima. Razmotrimo sistem od k jednačina po nepoznatoj x:

$$x \equiv r_1 \,(\bmod n_1),\ x \equiv r_2 \,(\bmod n_2),\ \ldots,\ x \equiv r_k \,(\bmod n_k), \tag{1}$$

gdje je $0 \leq r_1 \leq n_1 - 1$, $0 \leq r_2 \leq n_2 - 1$, ..., $0 \leq r_k \leq n_k - 1$. Dakle, poznati su ostaci $r_1, r_2, \ldots, r_k$ koji se dobijaju prilikom dijeljenja nekog broja x sa brojevima $n_1, n_2, \ldots, n_k$ redom. Treba odrediti x. Govoreći puno tačnije, treba ispitati da li rješenje x postoji. Vidjećemo da rješenje postoji pod određenim uslovima za $n_1, n_2, \ldots, n_k$.

Teorema o ostacima. Neka je $k \geq 1$. Neka su $n_1, n_2, \ldots, n_k$ prirodni brojevi. Neka je $0 \leq r_1 \leq n_1 - 1$, $0 \leq r_2 \leq n_2 - 1$, ..., $0 \leq r_k \leq n_k - 1$. Neka je $N = n_1 \cdot n_2 \cdot \ldots \cdot n_k$. Ako su brojevi $n_1, n_2, \ldots, n_k$ dva po dva uzajamno prosti (to znači da je $\text{NZD}(n_i, n_j) = 1$ za $i \neq j$) onda sistem (1) ima jedinstveno rješenje u skupu $\{0, 1, \ldots, N-1\}$.

Dokaz teoreme. Mi ispitujemo postojanje i jedinstvenost rješenja x sistema jednačina (1) s tim da je $0 \leq x \leq N-1$. Pogledajmo iz suprotnog ugla: od x prema ostacima $r_1, r_2, \ldots, r_k$. Dakle, neka je izabran broj $x \in \{0, 1, \ldots, N-1\}$. Podijelimo x sa n_i i registrujmo čemu je jednak ostatak r_i, $r_i = x \bmod n_i$ za svako i. Za tako dobijenu k–torku $(r_1, r_2, \ldots, r_k)$ očito postoji rješenje sistema. Svako x predviđenog oblika produkuje jednu k–torku predviđenog oblika. Za neke k–torke rješenje postoji. Mogućih vrijednosti x ukupno ima očito N. Mogućih k–torki ukupno ima takođe $N = n_1 \cdot n_2 \cdot \ldots \cdot n_k$. Prema tome, dovoljno je pokazati da dva različita x neće produkovati jednu te istu k–torku. U tom cilju, neka je $0 \leq x \leq N-1$ i $0 \leq y \leq N-1$ i dopustimo da važi

$$x \equiv r_1 \,(\mathrm{mod}\, n_1),\ \ldots,\ x \equiv r_k \,(\mathrm{mod}\, n_k),\ y \equiv r_1 \,(\mathrm{mod}\, n_1),\ \ldots,\ y \equiv r_k \,(\mathrm{mod}\, n_k).$$

Stavimo $z = y - x$ i oduzmimo slične relacije:

$$z \equiv 0 \,(\mathrm{mod}\, n_1),\ z \equiv 0 \,(\mathrm{mod}\, n_2),\ \ldots,\ z \equiv 0 \,(\mathrm{mod}\, n_k),$$

z je djeljivo sa n_i za svako i. Rješenja po z u skupu cijelih brojeva su $z = 0, \pm N, \pm 2N, \ldots$. Iz $0 \leq x \leq N-1$, $0 \leq y \leq N-1$ i $z = y - x$ slijedi $z = 0$. Dakle, ni za koju k–torku $(r_1, r_2, \ldots, r_k)$ ne mogu postojati dva rješenja. Drugim riječima, ako se x promijeni onda se i k–torka promijeni. Već je rečeno da se broj mogućnosti za x poklapa sa brojem k–torki. Teorema je dokazana.

U specijalnom slučaju $k = 2$ teorema o ostacima glasi kako slijedi:

Teorema. Neka su n_1 i n_2 prirodni brojevi i neka su r_1 i r_2 cijeli brojevi, gdje je $0 \leq r_1 \leq n_1 - 1$ i $0 \leq r_2 \leq n_2 - 1$. Razmotrimo sistem jednačina

$$x \equiv r_1 \,(\mathrm{mod}\, n_1),\ x \equiv r_2 \,(\mathrm{mod}\, n_2)$$

po nepoznatoj $x \in \{0, 1, \ldots, N-1\}$, gdje je $N = n_1 n_2$. Ako je $\mathrm{NZD}(n_1, n_2) = 1$ onda sistem ima jedinstveno rješenje.

Očito je od interesa jedino slučaj $n_1 \geq 2$ i $n_2 \geq 2$. Razmotrimo dva količnika $k_1 = \left[\frac{x}{n_1}\right]$ i $k_2 = \left[\frac{x}{n_2}\right]$. Slijedi $x = k_1 n_1 + r_1$ i $x = k_2 n_2 + r_2$. Oduzimanjem imamo $k_1 n_1 - k_2 n_2 = -r_1 + r_2$.

Prema tome, ako je $\mathrm{NZD}(n_1, n_2) = 1$ onda jednačina $k_1 n_1 - k_2 n_2 = -r_1 + r_2$ po nepoznatim $k_1 \in Z$ i $k_2 \in Z$ ima rješenja. Primjer takve jednačine je $17k_1 - 12k_2 = 1$. O postupku za nalaženje rješenja govori se u idućem naslovu.

Mi razmatramo $y = x \bmod n$ samo za $n \in N$. Svakako je $y \geq 0$. Npr. $(-2) \bmod 10 = 8$.

Za binarnu operaciju $x \bmod n$ važi: $(x + y) \bmod n = (x \bmod n + y \bmod n) \bmod n$, $xy \bmod n = (x \bmod n \cdot y \bmod n) \bmod n$, $x^k \bmod n = (x \bmod n)^k \bmod n$, gdje je $n \geq 2$, $x, y \in Z$ i $k \geq 1$. Slično tome, ako je $x_1 \equiv x_2 \,(\mathrm{mod}\, n)$ i $y_1 \equiv y_2 \,(\mathrm{mod}\, n)$ tada važi $(x_1 + y_1) \bmod n = (x_2 + y_2) \bmod n$, kao i $x_1 y_1 \bmod n = x_2 y_2 \bmod n$, a takođe i $x_1^k \bmod n = x_2^k \bmod n$. Tako da se na primjer $x^{21} \bmod n$ može računati kao $(x^{16} \bmod n \cdot x^4 \bmod n \cdot x \bmod n) \bmod n$. Isto tako, npr. $y = x^6 \bmod n = (x^3 \bmod n)^2 \bmod n$.

Neka je broj n fiksiran. Znamo da binarna relacija $x \equiv y \,(\mathrm{mod}\, n)$ predstavlja jednu relaciju ekvivalencije na skupu Z.

6.3. UOPŠTENI EUKLIDOV ALGORITAM

Uopšteni Euklidov algoritam služi za rješavanje sljedećeg zadatka. Razmotrimo jednačinu

$$a\alpha + b\beta = f \tag{1}$$

gdje su a i b dati brojevi ($a \geq 2$ i $b \geq 2$) i f je dati cio broj ($f \neq 0$). Mi ćemo radi prostijeg pisanja smatrati da je $a > b$, čime se ne gubi na opštosti. Smatramo da je jednačina (1) po nepoznatim α i β koje se traže u skupu cijelih brojeva Z. Pitamo se da li postoji rješenje (α, β). Ili: da li postoji linearna kombinacija datih brojeva a i b koja daje datu vrijednost f. Uopšteni Euklidov algoritam rješava sljedeća dva pitanja. a) Daje potvrdan ili odrečan odgovor na pitanje postoji li rješenje jednačine. I b) ako je odgovor potvrdan onda još i saopštava jedno rješenje (α, β). Pored toga, kao usputni rezultat, c) saopštava i vrijednost NZD(a, b). Biće izložen algoritam i biće dokazana njegova korektnost.

Znamo da se prilikom izvršavanja Euklidovog algoritma (za NZD) u pojedinom koraku vrši dijeljenje $a : b$ čime se dobijaju količnik k i ostatak c, formule

$$\frac{a}{b} = k + \frac{c}{b} \quad \text{ili} \quad a = kb + c.$$

Očito je $c < b$. Tamo je pokazano da važi NZD(a, b) = NZD(b, c). Tako da ulogu para brojeva (a, b) preuzima manji par brojeva (b, c). Pogledajmo da iskoristimo za postavljeni zadatak. Izvršimo supstituciju iz posljednje relacije u relaciju (1):

$$(kb + c)\alpha + b\beta = f \quad \text{ili} \quad b(k\alpha + \beta) + c\alpha = f \quad \text{ili} \quad b\alpha' + c\beta' = f, \tag{2}$$

gdje je stavljeno

$$\alpha' = k\alpha + \beta, \;\; \beta' = \alpha.$$

Zaključujemo sljedeće: ako postoje cijeli koeficijenti (α i β) za linearnu kombinaciju brojeva a i b onda postoje i koeficijenti (α' i β') za linearnu kombinaciju manjeg para brojeva b i c. Važi i obrnut iskaz, budući da α i β mogu da se izraze preko α' i β':

$$\alpha = \beta', \;\; \beta = \alpha' - k\beta'. \tag{3}$$

Odgovor na pitanje postoje li koeficijenti za a i b poklapa se sa odgovorom na pitanje postoje li koeficijenti za b i c. Time je algoritam utemeljen. Dakle, mi ćemo iz koraka u korak svoditi pitanje na pitanje istog oblika koje se tiče para brojeva koji je sve manji i manji. Sve dok ne svedemo na pitanje koje se tiče para brojeva oblika $(a_j, 0)$. Zatim se u drugoj fazi rada algoritma krećemo u suprotnom smjeru. Upravo, na osnovu α' i β' koji odgovaraju jednom paru brojeva lako izračunamo α i β. Brojevi α i β odgovaraju sljedećem (većem) paru brojeva. Drugim riječima, uzastopno se primjenjuje formula (3).

V. u nastavku uopšteni Euklidov algoritam izražen na nestrogom jeziku.

Prilikom prelaska iz prve faze u drugu fazu pojavljuje se pitanje: da li postoje $\alpha_j \in Z$ i $\beta_j \in Z$ takvi da je $a_j\alpha_j + b_j\beta_j = f$, tj. da je $a_j\alpha_j = f$ (budući da je $b_j = 0$). Odgovor je očito da postoje ako i samo ako važi $a_j \mid f$, tj. ako i samo ako je f djeljivo sa a_j (podrazumijeva se, djeljivo bez ostatka).

Dvije ključne formule u drugoj fazi algoritma su $\alpha_i \leftarrow \beta_{i+1}$ i $\beta_i \leftarrow \alpha_{i+1} - k_i\beta_{i+1}$. To su očito preslikane formule (3).

Iz algoritma vidimo (u vezi $a_j\alpha_j = f$) da neophodan i dovoljan uslov za postojanje rješenja jednačine (1) glasi $\mathrm{NZD}(a,b) \mid f$. Specijalno, ako je $f = \pm 1$ onda je neophodno i dovoljno da a i b budu uzajamno prosti, tj. $\mathrm{NZD}(a,b) = 1$.

Dodajmo na kraju da se slično postupa ako je $a > 0$, $b < 0$.

Primjer. Da li postoje cijeli brojevi α i β takvi da važi jednakost $213\alpha + 90\beta = 6$. Ako postoje, naći bar jedno rješenje (α, β). V. u nastavku redosljed među–rezultata tokom rada algoritma. Odgovor. Rješenje postoji. Jedno rješenje je $(\alpha, \beta) = (22, -52)$.

Uopšteni Euklidov algoritam
$a \geq 2$, $b \geq 2$, $a > b$, $f \neq 0$
$a\alpha + b\beta = f$

read(a, b, f);
$a_0 \leftarrow a$; $b_0 \leftarrow b$;
$i \leftarrow 0$;
1: $k_i \leftarrow a_i$ div b_i; $c_i \leftarrow a_i$ mod b_i;
$a_{i+1} \leftarrow b_i$; $b_{i+1} \leftarrow c_i$;
$i \leftarrow i + 1$;
if $b_i > 0$ then goto 1;
$j \leftarrow i$; write('NZD(a,b)=', a_j);
if $a_j \mid f$ then goto 2;
write('ne postoje α i β'); stop;
2: write('postoje α i β');
$\alpha_j \leftarrow f$ div a_j; $\beta_j \leftarrow 0$;
for $i \leftarrow j - 1$ downto 0 do
begin $\alpha_i \leftarrow \beta_{i+1}$; $\beta_i \leftarrow \alpha_{i+1} - k_i\beta_{i+1}$ end;
write('α=', α_0, 'β=', β_0); stop

Primjer
$a = 213$ $b = 90$ $f = 6$
$213\alpha + 90\beta = 6$

djeljenik, djelilac, količnik, ostatak
$a_0 = 213$ $b_0 = 90$ $k_0 = 2$ $c_0 = 33$
$a_1 = 90$ $b_1 = 33$ $k_1 = 2$ $c_1 = 24$
$a_2 = 33$ $b_2 = 24$ $k_2 = 1$ $c_2 = 9$
$a_3 = 24$ $b_3 = 9$ $k_3 = 2$ $c_3 = 6$
$a_4 = 9$ $b_4 = 6$ $k_4 = 1$ $c_4 = 3$
$a_5 = 6$ $b_5 = 3$ $k_5 = 2$ $c_5 = 0$
$a_6 = 3$ $b_6 = 0$
$(j = 6)$, $\mathrm{NZD}(a,b) = 3$, $3 \mid 6$ jeste
$\alpha_6 = 2$ $\beta_6 = 0$
$\alpha_5 = 0$ $\beta_5 = 2$
$\alpha_4 = 2$ $\beta_4 = -2$
$\alpha_3 = -2$ $\beta_3 = 6$
$\alpha_2 = 6$ $\beta_2 = -8$
$\alpha_1 = -8$ $\beta_1 = 22$
$\alpha_0 = 22$ $\beta_0 = -52$
provjera $213 \cdot 22 - 90 \cdot 52 = 6$

Komentar (za primjer). Polazna jednačina $213\alpha_0 + 90\beta_0 = 6$ svodi se preko smjene $\alpha_1 = 2\alpha_0 + \beta_0$, $\beta_1 = \alpha_0$ na $90\alpha_1 + 33\beta_1 = 6$. Koeficijenti smjene i novi oblik jednačine zavise od dijeljenja, tj. od izraza $213 = 2 \cdot 90 + 33$. Itd. Jednačina $6\alpha_5 + 3\beta_5 = 6$ svodi se preko smjene na $3\alpha_6 = 6$. Sada je završena prva faza rada programa. Imamo $\alpha_6 = 2$ i β_6 po želji, recimo $\beta_6 = 0$. U drugoj fazi ostvaruje se postepeni povratak od (α_6, β_6) ka (α_0, β_0) saglasno uvedenim smjenama. Rezultat je $(\alpha, \beta) = (\alpha_0, \beta_0) = (22, -52)$.

Komentar (za primjer). Postavljena jednačina ima i drugih rješenja, recimo $\alpha = 22 - 30$, $\beta = -52 + 71$ znači $\alpha = -8$, $\beta = 19$ i slično.

Napomena. Preporučuje se da se razmatrana jednačina (1) na početku podijeli sa $\mathrm{NZD}(a,b)$, pa tek nakon toga da se primijeni algoritam.

Drugi primjer. Postavka: odrediti a,
$11a \equiv 1 \pmod{42}$, $1 \leq a \leq 41$
Izrada: $11a - 42b = 1$ za neko b
$11a - (44b - 2b) = 1$
$11(a - 4b) + 2b = 1$, smjena $c = a - 4b$
$11c + 2b = 1$
$2b + (10c + c) = 1$
$2(b + 5c) + c = 1$, smjena $d = b + 5c$
$2d + c = 1$
Izberimo $d = 0$
Počinje povratak $c = 1$
iz smjene $b = -5$
iz smjene $a = -19$
Možemo a–u dodati ili oduzeti 42 ili 84 ili sl., budući da $y = kn + x \Rightarrow y \equiv x \pmod{n}$
$(k = \pm 1, \pm 2, \ldots)$
Prema tome $42 - 19 = 23$
tako da je $1 \leq a \leq n - 1$ $(n = 42)$
Odgovor: $a = 23$.

6.4. KRIPTOGRAFIJA POMOĆU JAVNOG KLJUČA: NAČIN RSA

Kriptografija znači tajno pisanje. Neka dvije osobe žele da razmjenjuju poruke preko računarske mreže. Savremeni protokoli koji se koriste temelje se na tzv. sistemu javnog ključa. Postoji nekoliko načina (postupaka), međusobno su slični, svi su utemeljeni na značajnim matematičkim činjenicama, upravo na rezultatima iz teorije brojeva. Način RSA je među prvim predloženim načinima, a i danas je veoma rasprostranjen.

Biće izloženi neophodni matematički pojmovi, zatim opšta šema načina RSA i zatim njegova analiza u pogledu vremenske složenosti i bezbjednosti.

⋈ Pogledajmo prvo matematičke činjenice.

Teorema. Neka su p i q prosti brojevi ($p \neq q$) i neka bude $n = pq$ i $\varphi(n) = (p-1)(q-1)$. Neka je d bilo koji broj ($1 < d < \varphi(n)$) takav da je d uzajamno prost sa $\varphi(n)$. Tada postoji jedan jedini broj e ($1 < e < \varphi(n)$) takav da je $de \equiv 1 \,(\mathrm{mod}\, \varphi(n))$. Dalje, neka je M ma koji broj ($0 < M < n$) i neka bude $C = M^e \bmod n$, tako da je očito $0 < C < n$. Tada važi $M = C^d \bmod n$.

Znamo da $de \equiv 1 \,(\mathrm{mod}\, \varphi(n))$ znači da je, prilikom dijeljenja broja de sa brojem $\varphi(n)$, ostatak jednak 1. Znamo da $M^e \bmod n$ jeste ostatak prilikom dijeljenja broja M^e sa brojem n. Ovu teoremu navodimo bez dokaza.

⋈ Sada je na redu ukupna šema načina RSA.

Pođimo od osobe B Bob. On želi da mu se šalju poruke, od strane raznih lica. Bob izabere p i q i odmah izračuna $n = pq$. On još izabere broj d da ispunjava gore navedeni uslov. Na kraju, on izračuna broj e, kako je gore definisano. Bob na neki način objavi svakome n i e koji predstavljaju javni dio ključa. Isto tako objavi da on koristi RSA algoritam za prenošenje poruka, tj. za kodiranje i dekodiranje. S druge strane, samo on zna vrijednost d. To je njegova tajna. To je njegov tajni ključ. Sada će razna lica da šalju puno poruka Bobu, držeći se predložene šeme.

Uzmimo da osoba A Alice želi da pošalje poruku M Bobu. Njena poruka predstavlja broj u gore naznačenim granicama za M. Svakako da ona kroz računarsku mrežu neće uputiti M, već će uputiti kodirani oblik poruke. Upravo, ona će uputiti $C = M^e \bmod n$.

Prema tome, Bob će dobiti C. On će izračunati $C^d \bmod n = M$. Tako će on moći da pročita šta mu ona kaže.

M message, C ciphertext, e encrypt, d decrypt.

Brojeve p i q treba izabrati tako da imaju po 100 dekadnih cifara ili više. Tako da broj n ima 200 dekadnih cifara ili više. Veći broj dekadnih cifara znači veću sigurnost.

U nastavku ćemo vidjeti da je čitava šema izvodljiva. Isto tako, vidjećemo da je šema i bezbjedna. Drugim riječima, realno je pretpostaviti da nepoželjni učesnik u komunikaciji može da sazna C (on još naravno zna i Bobov javni ključ). Ipak, on neće moći da otkrije M. Dakle, tajnost poruke je obezbijeđena. Takođe, poruka u kodiranom obliku C tokom svog puta kroz računarsku mrežu ne može da bude prepravljena jer ona ima svoju nevidljivu unutrašanju strukturu, u rezultatu dekodiranja prepravljenog oblika Bob će dobiti neki tekst koji nema smisla. Međutim, treći učesnik može da pošalje Bobu poruku u kojoj piše "ovo ti šalje Alice". Sredstva kojima se ovaj problem prevazilazi nazivaju se digitalni potpis. O ovome će biti riječi u sljedećem naslovu. Još ostaje samo jedan problem. Naime, Marko želi da pošalje poruku Bobu. Kako će Marko biti siguran da je (n, e) upravo Bobov javni ključ? Ovo je tzv. pitanje autentičnosti. I ovaj problem može da bude prevaziđen: neka Bob objavi svoj javni ključ u malim oglasima u novinama. Ili pomoću nekoga ko već ima sistem.

⋈ Izvršiti rekapitulaciju svih računskih radnji koje treba da budu izvršene, koje se pojavljuju.

Prilikom konstrukcije svog sistema, Bob treba da uradi tri radnje a)–c). a) Da ptonađe dva velika prosta broja p i q. Vidjećemo u nastavku da računska složenost ovog posla nije uopšte za zanemarivanje! A ipak posao može da bude urađen. Zadatak "prime" glasi: dat je prirodan broj, odgovoriti sa "da" ili "ne" na pitanje: da li je taj broj prost. Samo se napominje da za pitanje a) postoji niz prihvatljivih rješenja.

b) Zatim treba da pronađe jedan broj d da je uzajamno prost sa $\varphi(n) = (p-1)(q-1)$. Postoje jednostavna rješenja za ovo pitanje. Znamo da se NZD dva broja može izračunati po Euklidovom algoritmu. Na primjer, kao rješenje d može da posluži bilo koji prost broj koji je veći od $\max(p,q)$. Savjetuje se da d bude veliki broj.

c) Još treba da riješi po nepoznatoj e jednačinu $de \equiv 1 \pmod{\varphi(n)}$. Za ovo postoji efikasan algoritam. Primijeniti uopšteni Euklidov algoritam: $\exists k$, $de + k\varphi(n) = 1$, nepoznate su e i k.

d) Prilikom slanja poruke, Alice treba da izračuna $M^e \bmod n$. Za ovo postoji efikasan algoritam. Opišimo ga u glavnim crtama. Izračunati $M^2, M^4, M^8, M^{16}, \ldots$ i onda pomnožiti neke od izračunatih brojeva da izađe M^e. Ustvari, izračunati $M^2 \bmod n, M^4 \bmod n, M^8 \bmod n, M^{16} \bmod n, \ldots$ i onda pomnožiti $\bmod n$ neke od izračunatih brojeva da izađe $M^e \bmod n$.

e) Za dekodiranje, Bob treba da izračuna $C^d \bmod n$. Vidimo da je riječ o istom zadatku kao pod d).

f) Neko ko želi da probije šifru treba da rastavi na proste činioce broj n. Bolje rečeno, ako rastavi n onda je on probio šifru. Broj n je svima poznat. Vidjećemo da dosad nije pronađen algoritam koji bi bio makar i blizu efikasnom za rješavanje sljedećeg zadatka "factor": dat je prirodan broj, kako glasi njegovo rastavljanje u proizvod prostih brojeva. A moguće je da efikasan algoritam postoji (i da će biti efektivno konstruisan). Ipak, zasad takav algoritam nije pronađen. Dakle, sigurnost cijelog algoritma bazira se na tome što zasad nema upotrebljivog algoritma za zadatak "factor"! Jedino na tome se bazira!

Zaista, pretpostavimo da je Carol uspjela da rastavi broj n na proste činioce, odnosno da ona raspolaže sa p i q. Njoj tada samo još ostaje da riješi po nepoznatoj d jednačinu $de \equiv 1 \pmod{\varphi(n)}$.

⋈ Navedimo primjer. Kroz primjer se ilustruje kako se poruci pridružuje broj M i kako se dugačka poruka rastavlja na blokove, pa se onda blok po blok kodira.

Razmotrimo slučaj $p = 47$, $q = 59$, $n = pq = 47 \cdot 59 = 2773$, $\varphi(n) = (p-1)(q-1) = 46 \cdot 58 = 2668$ i $d = 157$. Izračuna se da je tada $e = 17$.

Svakom slovu pridružuje se dvocifren broj: blanko = 00, A = 01, B = 02, ..., Z = 26 a jedan blok neka obuhvata dva slova, imamo u vidu da je kod nas $n = 2773$.

Neka poruka glasi `ITS ALL GREEK TO ME`. Znači da se poruka prikazuje kao `0920 1900 0112 1200 0718 0505 1100 2015 0013 0500`. Prvi blok $M = 920$ kodira se kao $C = M^e \bmod n = 920^{17} \bmod 2773 = 948$. Drugi blok $M = 1900$ kodira se kao $C = M^e \bmod n = 1900^{17} \bmod 2773 = 2342$. Itd. Čitava poruka kodira se sa `0948 2342 1084 1444 2663 2390 0778 0774 0219 1655` (ovo se šalje preko računarske mreže). Čitalac se može uvjeriti da dekodiranje radi: $948^{157} \bmod 2773 = 920$, itd.

⋈ Koliko vremena treba da se riješi zadatak "factor"?

Najbrži algoritam koji danas postoji za rastavljanje broja n na proste činioce potroši $T(n) = \exp\sqrt{\ln n \ln\ln n} = (\ln n)^{\sqrt{\ln n/\ln\ln n}}$ koraka. U tabeli je dat broj koraka (operacija) potrebnih da se broj n rastavi po tom algoritmu, kao i vrijeme koje se traži ako jedna operacija troši jednu mikro–sekundu za razne dužine broja n, tj. za razne slučajeve broja dekadnih cifara broja n.

cifara	broj operacija	vrijeme
50	$1{,}4 \cdot 10^{10}$	3,9 časova
75	$9{,}0 \cdot 10^{12}$	104 dana
100	$2{,}3 \cdot 10^{15}$	74 godine
200	$1{,}2 \cdot 10^{23}$	$3{,}8 \cdot 10^{9}$ godina
300	$1{,}5 \cdot 10^{29}$	$4{,}9 \cdot 10^{15}$ godina
500	$1{,}3 \cdot 10^{39}$	$4{,}2 \cdot 10^{25}$ godina

Lako se pokazuje da zadatak "factor" pripada klasi $\mathcal{NP}$. Nije poznato da li je NP–kompletan. V. kasnije o teoriji NP–kompletnih zadataka.

⋈ Koliko vremena treba da se riješi zadatak "prime"?

Poznat je algoritam (da se ispita da li je dati broj prost) čija je vremenska složenost $T(n) = (\ln n)^{c \ln\ln\ln n}$ za neko $c > 0$. Postoji algoritam koji potroši $T(n) = (\ln n)^{1+\ln\ln\ln n}$ koraka, gdje postoje neka ograničenja na kojima se ovdje nećemo zadržavati, v. tabelu.

cifara	broj operacija	vrijeme
50	$1{,}9 \cdot 10^{5}$	0,2 sekunde
75	$8{,}0 \cdot 10^{5}$	0,8 sekundi
100	$2{,}3 \cdot 10^{6}$	2 sekunde
200	$3{,}1 \cdot 10^{7}$	30 sekundi
300	$1{,}5 \cdot 10^{8}$	2 minuta
500	$1{,}1 \cdot 10^{9}$	18 minuta

Lako se pokazuje da zadatak "prime" pripada klasi $\mathcal{NP}$. Prije nekoliko godina pronađen je algoritam čija složenost iznosi $T(n) = c(\ln n)^{12}$ koraka (bit–operacija), odnosno dokazano je da "prime" pripada klasi $\mathcal{P}$, 2002. godine.

Laički posmatrano, rekli bismo da je zadatak "prime" nešto lakši od zadatka "factor", kada uporedimo vremensku složenost algoritma za jedan i drugi zadatak. Složenost APR algoritma manja je od složenosti Pomeranceovog algoritma. Formulom iskazano, važi relacija $(\ln n)^{c \ln\ln\ln n} \ll (\ln n)^{\sqrt{\ln n / \ln\ln n}}$ kad $n \to \infty$.

⋈ Sljedeće, u kriptoanalizi se ispituje da li predloženi sistem za šifrovanje ima slabih tačaka ili je pak (suprotno) neprobojan za napade. Pronađeni su uslovi koje treba poštovati da bi RSA sistem bio sasvim neprobojan. Brojevi p i q treba da se razlikuju po dužini za nekoliko cifara. Oba $p-1$ i $q-1$ treba da imaju velike proste činioce. NZD$(p-1, q-1)$ treba da bude mali.

⋈ RSA: Rivest, Shamir i Adleman, 1978. godine. APR algoritam za zadatak "prime": Adleman, Pomerance i Rumely, 1980. godine. Algoritam za zadatak "factor": Pomerance, 1982. godine.

6.5. DIGITALNI POTPIS

Biće izložen pojam digitalnog potpisa i biće izložen postupak po kome se (u slučaju primjene sistema RSA) ostvaruje digitalni potpis, elektronski potpis.

U slučaju obične ili papirne komunikacije (nekodirane), osoba B može da dobije papir sa otkucanim tekstom, gdje na početku teksta piše, takođe otkucano, da je papir poslala osoba A. Bolje bi bilo da stoji svojeručni potpis osobe A. Vidjećemo da je prilikom kodiranog (šifrovanog) slanja poruka po sistemu javnog ključa, preko komunikacione linije čija je zaštita od nepoželjnog učesnika pod znakom pitanja, moguće ostvariti ono što je ekvivalentno "svojeručnom potpisu". Ideja koja se koristi je vrlo jednostavna, koliko je jednostavna toliko je i zanimljiva i korisna. Vidjećemo da treba da i jedna i druga osoba imaju svoje javne ključeve.

Dakle, treba da bude obezbijeđeno sljedeće: da osoba A šalje poruke osobi B u kodiranom obliku, s tim da još: a) osoba B je sigurna da je upravo A poslala tu poruku i b) A kasnije ne može da negira (iz nekog razloga) da je upravo ona poslala tu poruku. Prema tome, kao što

je već rečeno: c) treće lice ne može da razumije poruku makar i da na računarskoj mreži (na komunikacionoj liniji) pročita kodirani oblik poruke i d) treće lice ne može da pošalje poruku u tuđe ime, tj. ne može da pošalje poruku osobi B u ime osobe A.

Ovaj naslov predstavlja nastavak prethodnog naslova (kao što digitalni potpis poruke predstavlja dogradnju slanja poruke), tako da se oznake i ostalo prenose. Ipak ćemo ponoviti ukratko, vršeći pritom malu dogradnju oznaka, jer će trebati.

Bob je izabrao proste brojeve p i q ($p \neq q$), kao i broj d takav da je $\mathrm{NZD}(d, \varphi(n)) = 1$, gdje je $n = pq$ i $\varphi(n) = (p-1)(q-1)$, $d < \varphi(n)$. On je još našao broj $e < \varphi(n)$ takav da $\varphi(n) \mid de-1$, $\varphi(n)$ dijeli $de-1$ ili svejedno broj $de-1$ djeljiv je bez ostatka sa brojem $\varphi(n)$. Kao $23 \mid 460$. Vrijednost e je jedinstvena. Time je Bob definisao svoj sistem za kriptografiju. Naravno da se primjenjuje način RSA. Njegov javni (opštepoznati) ključ je (n, e). A njegov tajni (privatni) ključ je broj d, samo njemu je poznat. Znamo da n i e služe kada se vrši enkripcija, kada se originalna poruka M prevodi u kodirani oblik C koji će biti poslat kroz komunikacionu liniju (kroz računarsku mrežu). Znamo da d zajedno sa n služi za obrnut proces, služi kada se vrši dekripcija: na osnovu C izračuna se M. Tako da Alice koja šalje poruku zna, pored n i e, svoju poruku u nekodiranom obliku M i u kodiranom obliku C. A nepoželjni učesnik u komunikaciji zna samo n i e i još eventualno može da sazna C, što mu je sve skupa nedovoljno. Dok Bob očito zna sve podatke: n, e, d, C i M. Uvedimo oznake $n = n_B$, $e = e_B$ i $d = d_B$. Razmotrimo skup $S = S_B = \{1, 2, \ldots, n-1\}$. Uočili smo da poruci odgovaraju dva algoritma: algoritam za enkripciju (primjenjuje ga Alice) i zatim algoritam za dekripciju (primjenjuje ga Bob). Uočili smo da su dva algoritma jedan drugom inverzni. Umjesto o dva algoritma možemo govoriti o dvije funkcije f i f^{-1}. Dakle, neka bude $f(M) = M^e \bmod n$ za svaki broj M takav da je $1 \leq M \leq n-1$, tj. za svaki broj $M \in S = S_B$. Pisaćemo svejedno f ili f_B ili E_B. Funkcija f je uzajamno jednoznačna (1–1) i "na". Njen domen (njena oblast definisanosti) je skup $S = \{1, 2, \ldots, n-1\}$, kao što je već rečeno. Kao što je u prethodnom naslovu konstatovano, za inverznu funkciju f^{-1} važi formula $f^{-1}(C) = C^d \bmod n$ za svaki broj $C \in S$. Funkcija f^{-1} je takođe 1–1 i "na", $f^{-1}\colon S \to S$. Pisaćemo svejedno f^{-1} ili f_B^{-1} ili E_B^{-1} ili D_B. Ponovimo: E_B je uobičajena radnja koju vrši onaj koji šalje poruku Bobu, enkripcija po Bobovom sistemu ili ključu, D_B je uobičajena radnja koju vrši Bob kada dobije poruku (kada od nekog primi kodirani oblik poruke), dekripcija po Bobovom sistemu. Na osnovu izloženog znamo da je $D_B \circ E_B = I$ (kompozicija preslikavanja), kao uostalom i $E_B \circ D_B = I$, gdje je I identičko preslikavanje na skupu S. Drukčije zapisano $D_B(E_B(x)) = x$, kao uostalom i $E_B(D_B(x)) = x$ za svako $x \in S$ (za svako $1 \leq x \leq n-1$).

Neka i Alice ima svoj ključ: javni dio n_A i e_A i tajni dio d_A, a definiše se naravno kao u opštem slučaju (slično kao za Boba). Slično se definišu i dvije odgovarajuće funkcije: njena funkcija enkripcije E_A i njena funkcija dekripcije D_A. Obe funkcije definisane su na skupu $S_A = \{k \mid 1 \leq k \leq n_A - 1\}$. One očito posjeduju slična svojstva kao dvije funkcije čiji je indeks B. Posebno, važi jednakost $D_A^{-1} = E_A$, kao uostalom i jednakost $E_A^{-1} = D_A$.

Pogledajmo sada kako se vrši digitalni potpis. U najkraćem kazano, važi jednakost

$$E_A(D_B(E_B(D_A(M)))) = M.$$

Šematski prikazano: Alice $M \xrightarrow{D_A} S \xrightarrow{E_B} C$, Bob $C \xrightarrow{D_B} S \xrightarrow{E_A} M$. Uvedena je oznaka S za potpisanu poruku (signed message).

Ispričajmo korake i njihov redosljed riječima. Razlikujemo sljedeća četiri koraka: $S = D_A(M)$, $C = E_B(S)$, $S = D_B(C)$ i $M = E_A(S)$.

1. Na svoju poruku M Alice prvo primijeni svoju funkciju za dekripciju D_A, čime ona dobija S, potpisani oblik poruke. Dakle, izvršila je radnju koju uobičajeno uradi kada ona od nekoga dobije poruku u kodiranom obliku (misli se: u slučaju RSA bez potpisa). 2. Zatim ona kodira (enkriptuje) S saglasno Bobovom ključu. Tako ona dobija C. Ona pošalje C preko mreže. 3. Kada C pristigne kod Boba onda on prvo izvrši dekripciju saglasno svom ključu. Tako je on odredio S. 4. Najzad, Bob na S primijeni enkripciju na Alisin način. Drugim riječima, izvrši radnju koja je uobičajena prilikom slanja poruke Alisi (misli se: u slučaju sistema RSA bez digitalnog potpisa). Sada Bob ima M i sada je sve završeno. On može da pročita šta mu ona poručuje.

Ispravnost predloženog postupka proizilazi iz ranije navedenih činjenica.

Izvršiti analizu i uvjeriti se da su ranije na početku postavljeni zahtjevi (uslovi) ispunjeni u cjelini. U okviru analize, Alisa nije u mogućnosti da otkrije D_B, kao što ni Bob nije stekao podatke po kojima bi mogao da otkrije njen tajni ključ d_A.

Time je izložena ukupna šema digitalnog potpisa prilikom kriptografije temeljene na javnom ključu (u slučaju RSA). Do kraja nam još ostaju dvije dopune.

Izvršimo malu dogradnju izložene šeme. Kako Bob zna ko mu je uputio poruku? Vidimo da on u četvrtom koraku upotrebljava funkciju E_A. Dobro bi bilo da on preko mreže zajedno sa S dobije i neki tekst oblika približno "ovu poruku Alice upućuje". Rješenje: neka ulogu S preuzme tekst + S, sa očitim izmjenama u ukupnom algoritmu. Dakle, u drugom koraku Alice vrši enkripciju od tekst + S. Na kraju trećeg koraka Bob će raspolagati sa "ovu poruku Alice upućuje" + S. Sada on zna da u posljednjem (četvrtom) koraku treba da primijeni upravo funkciju E_A na S.

Zapaziti da je $n_A \neq n_B$ uopšte uzev, pretpostavimo određenosti radi da je $n_A > n_B$. Zapaziti da dvije funkcije imaju kao domen (oblast definisanosti) skup S_A a dvije funkcije imaju kao domen manji skup S_B. Time se stvara jedan manji problem. Naime, može se desiti da bude $D_A(M) \geq n_B$, odnosno da izraz $E_B(D_A(M)) = E_B(S)$ nema smisla, nije definisan. Rješenje: tada treba podijeliti S na dva bloka, na blokove čija je veličina dozvoljena, čija je veličina $< n_B$. Tek nakon toga izvršiti enkripciju, naravno blok po blok. Drugim riječima, tek nakon toga primijeniti E_B na pojedinačne blokove. Onda preko mreže poslati.

(Ili Alice šalje M i $S = D_A(M)$, a onda Bob računa $E_A(S)$, izlazi M očito. Kaže se da je Alice potpisala poruku M, a da Bob ili neko drugi vrši provjeru, može da izvrši provjeru.) Ako autoritet ili agencija stavi potpis na izjavu oblika "Alice koristi RSA sa parametrima $n =$ toliko i $e =$ toliko" onda se kaže da su oni izdali jedan digitalni sertifikat.

6.6. KRIPTOGRAFIJA POMOĆU JAVNOG KLJUČA: METODA KVADRATNOG OSTATKA

Razmotrimo funkciju $y = y(x)$ definisanu jednakošću $y = x^2 \bmod n$. Vidjećemo da ova funkcija ima jedno zanimljivo svojstvo (pod određenim uslovima na broj n). Vidjećemo da na osnovu tog svojstva može da bude izgrađen jedan dobar kriptografski sistem. Za taj sistem koristi se naziv kvadratni ostatak ili eventualno ostatak od potpunog kvadrata, isto Rabinov sistem. Kažimo unaprijed da x ima ulogu poruke (message), a y ima ulogu kodiranog oblika poruke (ciphertext).

⋈ Pogledajmo prvo matematičke činjenice na kojima se sistem temelji.

Teorema. Neka su p i q prosti brojevi ($p \neq q$) takvi da je $p \equiv 3 \,(\mathrm{mod}\, 4)$ i $q \equiv 3 \,(\mathrm{mod}\, 4)$, to znači da p prilikom dijeljenja sa 4 daje ostatak 3 i da q prilikom dijeljenja sa 4 daje ostatak

3. Stavimo $n = pq$ (tzv. Blumov broj). Neka je x prirodan broj, $1 \leq x \leq n-1$. Stavimo $y = x^2 \bmod n$. Tada je $1 \leq y \leq n-1$. Stavimo

$$y_1 = (y \bmod p)^{(p+1)/4} \bmod p \ \text{ i } \ y_2 = (y \bmod q)^{(q+1)/4} \bmod q.$$

Tada x zadovoljava sistem jednačina

$$x \equiv \pm y_1 \,(\mathrm{mod}\, p), \quad x \equiv \pm y_2 \,(\mathrm{mod}\, q). \tag{1}$$

Teorema (nastavak prethodne teoreme). Ako je polazni broj x oblika cio broj puta p ili oblika cio broj puta q onda sistem (1) ima dva rješenja. Za ostale x ($1 \leq x \leq n-1$) sistem ima četiri rješenja. Rješenja se naravno traže u skupu $\{1, 2, \ldots, n-1\}$.

Ove dvije teoreme navodimo bez dokaza. Lako se vidi da je $(n-k)^2 \bmod n = (n^2 - 2nk + k^2) \bmod n = k^2 \bmod n$. Lako se vidi da $k \equiv y_1 \,(\mathrm{mod}\, p) \Rightarrow n - k \equiv -y_1 \,(\mathrm{mod}\, p)$.

Da ponovimo. Izračuna se $y = x^2 \bmod n$. Izračunaju se y_1 i y_2. Formira se sistem (1), tako da je $x = ap \pm y_1$ za neko a i $x = bq \pm y_2$ za neko b, ustvari formiraju se četiri sistema po šablonu $(+,+)$, $(+,-)$, $(-,+)$, $(-,-)$. Riješi se sistem (1) po nepoznatoj x. Ima dva ili četiri rješenja. Među rješenjima će se svakako naći i polazni broj x.

U velikoj većini slučajeva, sistem ima četiri rješenja. Za potrebe kriptografije, bolje bi bilo da rješenja ima što manje (da je rješenje jedinstveno), da bi se otkrio polazni broj x.

Uzmimo da smo se ograničili na brojeve x takve da je $1 \leq x \leq \frac{n-1}{2}$ i uzmimo da rješenje sistema (1) tražimo samo u skupu $\left\{1, 2, \ldots, \frac{n-1}{2}\right\}$. Tada postoji jedno rješenje ili postoje dva rješenja. U "većini" slučajeva postoje dva rješenja, a samo u nekim izuzetnim slučajevima (vrijednosti y_1 i y_2) postoji samo jedno rješenje. Postojanje dva rješenja predstavlja mali problem, sa stanovišta kriptografuje. U nastavku će biti pokazano da se taj problem lako prevazilazi.

Neka je $X = \{1, \ldots, n-1\}$, $f(x) = x^2 \bmod n$, $Y = \{f(x) |\ x \in X\}$. U drugoj teoremi razmatra se slučaj $y \in Y$. Ako $y \notin Y$ onda će se rješavanjem sistema (1) dobiti broj x koji ne zadovoljava $x^2 \bmod n = y$.

$\bowtie$ Na redu je ukupan opis sistema kvadratnog ostatka.

Osoba B (Bob) želi da prima poruke preko računarske mreže. On će kao prvo da izgradi svoj sistem utemeljen na principu javnog ključa. U tom cilju, on izabere dva prosta broja p i q (da su različiti), takve da je $p \equiv 3 \,(\mathrm{mod}\, 4)$ i $q \equiv 3 \,(\mathrm{mod}\, 4)$. On pomnoži $n = pq$. Broj n predstavlja njegov javni ključ. Brojevi p i q čine njegov tajni ključ. Vrijednost n on objavi svakome. Brojeve p i q on čuva kao svoju tajnu (samo on zna p i q). On još objavi da se koristi metoda (algoritam) kvadratnog ostatka. To što je objavljeno saznali su osoba A (Alice) koja namjerava da šalje poruke, a isto i osoba C (Carol) koja se zanima pronalaženjem eventualnih slabih tačaka u kriptografskim sistemima.

Poruka koju Alice želi da saopšti Bobu prikaže se kao jedan broj $x \in \{1, 2, \ldots, n-1\}$. Ona naravno neće uputiti x kroz računarsku mrežu. Ona samo izračuna $y = x^2 \bmod n$ i onda uputi y Bobu preko računarske mreže.

Bobu je stiglo $y \in \{1, 2, \ldots, n-1\}$. On će riješiti sistem (1) i tako će saznati x, otkriće šta Alice želi da mu saopšti. Prilikom formiranja sistema (1) i rješavanja tog sistema, njemu pomažu vrijednosti p i q (njegov tajni ključ).

Za Carol se pretpostavlja da raspolaže javnim ključem n i da je saznala kodirani oblik poruke y. Nije dovoljno da bi mogla da otkrije x.

Brojeve p i q treba izabrati da imaju po 100 ili više dekadnih cifara. Tako da onda broj $n = pq$ ima 200 dekadnih cifara ili više.

⋈ Na redu je rekapitulacija. Koje računske operacije treba da budu izvršene i kolika je njihova vremenska složenost.

a) Prilikom konstrukcije svog kriptografskog sistema Bob treba da pronađe dva prosta broja p i q. Znamo da složenost ovog posla nije za potcjenjivanje. Ipak, postoje algoritmi koji će na računaru za svega nekoliko sekundi da utvrde da li je dati broj p prost, gdje p ima 1000 ili čak 3000 dekadnih cifara. Plus Bob treba da izvrši jednostavnu radnju $n = pq$.

b) Pošiljalac poruke Alice treba da izvrši samo jednostavnu radnju $y = x^2 \bmod n$.

c) Primalac Bob treba da uradi sljedeće. Da izračuna y_1 i y_2, složenost je prihvatljiva. Da nađe sva rješenja sistema (1), on će primijeniti uopšteni Euklidov, složenost je prihvatljiva. I još samo da od nađenih rješenja (od četiri rješenja) odabere ono pravo, jednostavno.

Ako je $x \equiv y_1 \pmod p \Rightarrow x = ap + y_1$ za neko a, $x \equiv y_2 \pmod q \Rightarrow x = bq + y_2$ za neko b, onda $ap + y_1 = bq + y_2$ i onda uopšteni Euklidov algoritam da se nađu a i b, samim tim i x.

d) Pred Carol stoji zadatak o rastavljanju broja n na proste činioce. Nepremostiva prepreka. Budući da danas nije poznat algoritam koji bi na bilo kom računarskom sistemu našao proste činioce broja n od 200 ili više dekadnih cifara za vrijeme koje bi bilo kraće od .... Nije lako izračunzti. Možda godinu dana ili deset godina.

Ko rastavi n, taj je provalio sistem (otključao je). Pokušaji da se nešto postigne (na planu dekodiranja) sa pretpostavkom da se raspolaže sa puno y takvih da $\sqrt{y}$ postoji i predstavlja smislenu poruku, gdje svi y odgovaraju jednom te istom n, nisu dali rezultata.

⋈ Još nešto o bezbjednosti sistema.

Korišćenjem najmoćnijih sredstava može da bude faktorisan broj od $N = 120$ dekadnih cifara za razumno vrijeme. Ni korišćenjem najmoćnijih sredstava danas ne može da bude faktorisan za iole prihvatljivo vrijeme broj veličine $N = 150$ dekadnih cifara. Čini se da je $\Delta N = 3$ ili $\Delta N = 4$ na godišnjem nivou, najviše. Da se ovo pokaže, treba izvršiti računicu. Računica se zasniva na analizi vremenske složenosti najboljeg danas poznatog algoritma za faktorisanje velikih prirodnih brojeva (Pomerance). Računica se zasniva i na tzv. Murovom zakonu (Moore) koji govori da se broj logičkih kola na čipu udvostruči kada prođu dvije godine, da se ukupna performansa računara za vrijeme od otprilike dvije godine udvostruči.

Već je rečeno da se bezbjednost većine sistema sa javnim ključem temelji na tome što do danas nije pronađen efikasan algoritam (algoritam čija je složenost polinomska) za rješavanje zadatka o faktorizaciji prirodnog broja (o rastavljanju datog prirodnog broja na njegove proste činioce). Dimenzioni broj tog zadatka definiše se naravno kao broj binarnih ili dekadnih cifara tog prirodnog broja. Dosad nije pronađen, a moguće je da postoji; moguće je da postoji i da će biti pronađen. Dalja priča o ovoj temi vodi prema teoriji NP–kompletnih zadataka.

⋈ Kako riješiti mali problem što je rješenje x sistema (1) nejednoznačno (najviše četiri rješenja mogu da postoje)? Neka Alice na kraju poruke ponovi nekoliko njenih posljednjih karaktera (to je x). Bob će naći sva rješenja i izdvojiće kao tačno rješenje ono rješenje koje posjeduje takvo svojstvo: svojstvo da se na kraju poruke dvaput pojavljuje jedan te isti string.

⋈ Jedan blok poruke ima veličinu 660 bita približno, ako je kao n poslužio broj koji ima 200 dekadnih cifara. Znamo da se dugačka poruka rastavlja na blokove i da se svaki blok poruke posebno kodira.

⋈ Primjer. Neka bude $p = 7$ i $q = 11$. Ispunjeni su uslovi $p \equiv 3 \pmod 4$ i $q \equiv 3 \pmod 4$. Imamo

da je $n = pq = 77$. Neka promjenljiva x prolazi kroz skup $\{1, 2, \ldots, n-1\} = \{1, 2, \ldots, 76\}$. Stavimo $y = x^2 \bmod 77$, tako da y takođe pripada skupu $\{1, 2, \ldots, 76\}$. Za razne y, treba riješiti jednačinu $y = x^2 \bmod 77$. Nepoznata x traži se u tom istom skupu. Tu jednačinu ćemo riješiti puno lakše ako se oslonimo na formule (1).

Pogledajmo prvo jedan poseban slučaj. Stavimo da je $x = 2$. Tada je $y = x^2 \bmod 77 = 4 \bmod 77 = 4$. Zaboravimo polaznu vrijednost x i prosto razmotrimo jednačinu $x^2 \bmod 77 = 4$. Sada, formule (1) u slučaju $y = 4$ govore da je

$$y_1 = (y \bmod 7)^2 \bmod 7 = (4 \bmod 7)^2 \bmod 7 = 4^2 \bmod 7 = 16 \bmod 7 = 2,$$

$$y_2 = (y \bmod 11)^3 \bmod 11 = (4 \bmod 11)^3 \bmod 11 = 4^3 \bmod 11 = 64 \bmod 11 = 9.$$

Sistem (1) glasi

$$x \equiv \pm 2 \,(\mathrm{mod}\, 7), \quad x \equiv \pm 9 \,(\mathrm{mod}\, 11). \tag{2}$$

Postoje ukupno četiri rješenja i to $x = 2$, $x = 9$, $x = 68$ i $x = 75$. Treba izvršiti provjeru.

x=1 ... x=10	y=1 y=4 y=9 y=16 y=25	y=36 y=49 y=64 y=4 y=23
x=11 ... x=20	y=44 y=67 y=15 y=42 y=71	y=25 y=58 y=16 y=53 y=15
x=21 ... x=30	y=56 y=22 y=67 y=37 y=9	y=60 y=36 y=14 y=71 y=53
x=31 ... x=40	y=37 y=23 y=11 y=1 y=70	y=64 y=60 y=58 y=58 y=60
x=41 ... x=50	y=64 y=70 y=1 y=11 y=23	y=37 y=53 y=71 y=14 y=36
x=51 ... x=60	y=60 y=9 y=37 y=67 y=22	y=56 y=15 y=53 y=16 y=58
x=61 ... x=70	y=25 y=71 y=42 y=15 y=67	y=44 y=23 y=4 y=64 y=49
x=71 ... x=76	y=36 y=25 y=16 y=9 y=4	y=1

y=1	x=1, x=34, x=43, x=76	y=42	x=14, x=63
y=4	x=2, x=9, x=68, x=75	y=44	x=11, x=66
y=9	x=3, x=25, x=52, x=74	y=49	x=7, x=70
y=11	x=33, x=44	y=53	x=19, x=30, x=47, x=58
y=14	x=28, x=49	y=56	x=21, x=56
y=15	x=13, x=20, x=57, x=64	y=58	x=17, x=38, x=39, x=60
y=16	x=4, x=18, x=59, x=73	y=60	x=26, x=37, x=40, x=51
y=22	x=22, x=55	y=64	x=8, x=36, x=41, x=69
y=23	x=10, x=32, x=45, x=67	y=67	x=12, x=23, x=54, x=65
y=25	x=5, x=16, x=61, x=72	y=70	x=35, x=42
y=36	x=6, x=27, x=50, x=71	y=71	x=15, x=29, x=48, x=62
y=37	x=24, x=31, x=46, x=53		

Sada pogledajmo opšti slučaj. Za svako $x = 1, 2, \ldots, 76$ izračunati y po formuli $y = x^2 \bmod 77$. Vrijednosti koje je saopštio računar prikazane su u tabeli.

Za $x = 2$ odgovara $y = 4$. Pogledajmo iz suprotnog ugla. Za $y = 4$ odgovara $x = 2$, a možda još i neke druge vrijednosti x odgovaraju. Dakle, sada ćemo da izvršimo "inverziju" tabele. U drugoj tabeli prikazane su sve vrijednosti $y \in \{1, 2, \ldots, 76\}$ za koje jednačina $x^2 \bmod 77 = y$ (po nepoznatoj $x \in \{1, 2, \ldots, 76\}$) ima rješenja. Još su prikazana i sva odgovarajuća rješenja.

Ako je $x^2 \bmod 77 = 4$ i $1 \leq x \leq 76$ onda je $x = 2$ ili $x = 9$ ili $x = 68$ ili $x = 75$ (četiri rješenja). Jednačina $x^2 \bmod 77 = 5$ nema rješenja u skupu $\{1, 2, \ldots, 76\}$. Završen primjer.

Rješenje sistema $x \equiv 2 \,(\mathrm{mod}\, 7)$, $x \equiv 9 \,(\mathrm{mod}\, 11)$ glasi $x = 9$. Rješenje sistema $x \equiv 2 \,(\mathrm{mod}\, 7)$, $x \equiv -9 \,(\mathrm{mod}\, 11)$ glasi $x = 2$. Rješenje sistema $x \equiv -2 \,(\mathrm{mod}\, 7)$, $x \equiv 9 \,(\mathrm{mod}\, 11)$ glasi $x = 75$. Rješenje sistema $x \equiv -2 \,(\mathrm{mod}\, 7)$, $x \equiv -9 \,(\mathrm{mod}\, 11)$ glasi $x = 68$.

⋈ Dopuna. Slučaj dugačke poruke. Pretpostavimo da broj n ima $b = 660$ (ili $b = 1024$) binarne cifre. Tada i x može da ima isto toliko binarnih cifara, najviše. Isto važi i za y. Uzmimo da poruka ima $2b$ bita. Tada se poruka rastavi na dva bloka x_1 i x_2. Kroz računarsku mrežu poslati prvi mail y_1 i malo kasnije drugi y_2, gdje je $y_1 = x_1^2 \bmod n$ i $y_2 = x_2^2 \bmod n$.

VIŠAK (NE TREBA)

DOPUNA: SLOŽENOST ALGORITMA ZA FAKTORIZACIJU

Izvršimo još jednom analizu bezbjednosti (stepena zaštite) kriptografskih sistema tipa javnog ključa. Procijenimo izglede da neko uspije da probije zaštitu. Da uspije da faktoriše broj n. Najbolji danas poznati algoritam za rješavanje zadatka o faktorizaciji jeste Pomeranceov algoritam. Njegova vremenska složenost je $T(n) = c \exp \sqrt{\ln n \ln \ln n}$ operacija, očekivani slučaj. Za taj algoritam koristi se naziv metoda kvadratnog sita, engl. quadratic sieve.

Pogledajmo prvo sa kakvim hardverom raspolažemo. Neka je na raspolaganju računar čija se brzina (tempo, učestanost časovnika) opisuje sa vrijednošću 10^k MIPS. MIPS znači million instructions per second, 10^6 naredbi u sekundi. Neka broj n koji treba da bude faktorisan ima N dekadnih cifara. Po formuli za $T(n)$ lako se izračuna koliko se vremena potroši (koliko vremena se zahtijeva) da se izvrši Pomeranceov algoritam na tom hardveru sa brojem n kao ulaznim podatkom. To je zavisnost vrijeme = vrijeme(N). Pogledajmo inverznu zavisnost.

Neka je raspoloživo vrijeme za rad hardvera (računara) fiksirano. Uzmimo da je to fiksirano vrijeme jednako jednoj sekundi. U tabeli je prikazano najveće N za koje možemo da saznamo odgovor (za koje će nam hardver saopštiti rezultat, faktorizaciju broja n). Drugim riječima, za brojeve sa N ili manje dekadnih cifara mi možemo da saznamo kako glasi njihovo rastavljanje na proste činioce (na proste faktore). A za brojeve sa preko N dekadnih cifara, hardver je bespomoćan, za dato fiksirano raspoloživo vrijeme. Možda nekome jedna sekunda izgleda kao isuviše stroga granica. Neka umjesto jedna sekunda bude jedan dan. Jedan dan = 86.400 sekundi. Približno jedan dan = 10^5 sekundi. Odgovarajuću graničnu vrijednost (do koliko N može za jedan dan na 10^k MIPS hardveru) takođe imamo u tabeli. Pogledati šta u tabeli piše malo dalje, na mjestu $k + 5$.

Tabela: Pomerance za jednu sekundu uspijeva do N dekadnih cifara na 10^k MIPS hardveru. Istabelirano je za $k = 6$, $k = 7$, ..., $k = 18$.

Murov zakon (Moore's law): za dvije godine udvostruči se performansa računara. Slijedi: za šest godina se udesetostruči, grubo računato. Broj k poveća se za jedan svakih šest godina.

Vidimo da gornja granica N iz godine u godinu sporo napreduje. Napredak tehnologije ne doprinosi puno. Ima li nešto što bi doprinijelo? Kada bi se pronašao bolji (efikasniji) algoritam.

Prethodni dio tabele:

$k =$	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5
$N =$	1	2	4	7	11	16	21	27	34	41	49	57

Tabela:

$k =$	6	7	8	9	10	11	12	13	14	15	16	17	18
$N =$	66	76	86	96	107	119	131	143	156	170	184	199	214

Nastavak tabele:

$k=$	19	20	21	22	23	24
$N=$	230	246	262	279	297	315

Intel Pentium III 2000. godine 10^3 MIPS.

DOPUNA: MUROV ZAKON (MOORE'S LAW)

Tzv. Murov zakon ili Murova procjena glasi: performansa računara (broj bit operacija u sekundi) udvostruči se kada prođe vrijeme Δt, gdje je Δt neka vrijednost u granicama 2 godine $\leq \Delta t \leq 2{,}5$ godina. Znači da se efikasnost računara (broj operacija izvršenih u jedinici vremena) udesetostruči svakih 8 godina. Za 64 godine efikasnost se pomnoži sa 10^8. Ako je računaru iz 1938. godine bila potrebna jedna sekunda vremena da sabere dva cijela broja onda računar iz 2002. godine za jednu sekundu vremena obavi 10^8 operacija integer sabiranja.

O empirijskim temeljima Murovog zakona. Prvim računarima smatraju se Z3 iz 1938. godine (Zuse), ABC iz 1941. godine (Atanasoff), ENIAC iz 1946. godine (Eckert) i Manchester Mark I iz 1948. godine (Williams). Sve do Pentium 2GHz iz 2002. godine. Itd. Za razne računare iz prošlosti, u literaturi se lako mogu pronaći podaci o njihovim karakteristikama, kakve su na primjer: koliko bit operacija u sekundi, koliko integer sabiranja u sekundi, koliko elektronskih lampi ili tranzistora ima u procesoru (u računaru), koliki je kapacitet memorije.

O matematičkim temeljima Murovog zakona. Veličina $y = y(t)$ mijenja se kako vrijeme t protiče. Kao što znamo, trenutni tempo promjene iznosi $y' = y'(t)$. Za mnoge pojave y važi $y' = ky$, gdje je $k > 0$ konstanta. Što je k veće to je razvoj brži. Opšte rješenje diferencijalne jednačine $y' = ky$ glasi $y(t) = Ce^{kt}$. Eksponencijalna funkcija $y(t) = Ce^{kt}$ posjeduje sljedeće svojstvo: postoji konstanta $\Delta t > 0$ takva da važi relacija $\frac{y(t+\Delta t)}{y(t)} = 2$ bez obzira na C i t, već je rečeno da je k fiksirano. Lako se vidi da je Δt obrnuto srazmjerno sa k.

Neka podatak iz literature ima ulogu $y = y(t)$. Izvršiti računicu i odrediti Δt.

Postoji više mogućnosti za izbor veličine y. Na primjer, opšta performansa računara ili bit operacija u sekundi ili integer sabiranja u sekundi. Isto tako, broj logičkih elemenata u procesoru ili broj tranzistora na čipu. Takođe, koliko ima memorijskih elemenata (flipflopova) u memoriji ili koliki je kapacitet memorijskog čipa (izraženo u broju bita). Iz godine u godinu, razne veličine y mijenjaju se po dosta sličnom zakonu.

Treba reći da se i sama veličina Δt pomalo mijenja, gledano od prvih računara do naših dana.

U tabeli je prikazana gruba procjena vremena udvostručavanja Δt za razne decenije.

od 1960. godine do 1970. godine bilo je $\Delta t = 16$ mjeseci
od 1970. godine do 1980. godine bilo je $\Delta t = 18$ mjeseci
od 1980. godine do 1990. godine bilo je $\Delta t = 20$ mjeseci
od 1990. godine do 2000. godine bilo je $\Delta t = 24$ mjeseca
od 2000. godine do 2010. godine bilo je $\Delta t = 30$ mjeseci

Primjer: UNIVAC I 1951. godine 1.900 sabiranja u sekundi, HP 9000 1991. godine 50.000.000 sabiranja u sekundi. Slijedi $e^{40k} = 26.000$. Uvijek važi $e^{k\Delta t} = 2$. Izračunati Δt.

Primjer: Intel 8086 1978. godine 28.000 tranzistora, Intel Pentium 1993. godine 3.000.000 tranzistora. Uvijek važi $e^{k\Delta t} = 2$. Izračunati Δt.

6.7. ISPITIVANJE PRIMALNOSTI: MILLEROV POSTUPAK

Za konstrukciju RSA sistema treba naći dva velika prosta broja (isto za sistem kvadratnog ostatka). Ispitati primalnost datog prirodnog broja znači utvrditi da li je taj broj prost ili složen. Dakle, mi želimo da sastavimo algoritam (program) čiji je ulazni podatak jedan prirodan broj, a koji će da saopšti "dati broj je prost" ili pak "dati broj nije prost". Takođe želimo da algoritam bude efikasan ili blizak efikasnom.

U ovom naslovu govori se o Millerovom postupku (o Millerovom testu). Ponekad se naziva – Fermaovim testom. Vidjećemo da razmatrani postupak samo **djelimično** rješava zadatak o primalnosti.

Napišimo prvo kako glasi Mala Fermaova teorema, u nešto prilagođenom obliku. Neka je $n \geq 2$ i $2 \leq b \leq n-1$. Neka je $(b,n) = 1$ (ili neka je $\mathrm{NZD}(b,n) = 1$, druga oznaka), tj. neka su b i n uzajamno prosti. Stavimo $y = b^{n-1} \bmod n$, tako da je očito $0 \leq y \leq n-1$. Teorema glasi: ako je n prost onda je $y = 1$.

Zapaziti da je $y \neq 0$. Prilagođavajući dalje našim potrebama, možemo reći da je $n \geq 20$. Naime, teorema će služiti da se vidi da li je broj n – prost broj, a za vrlo male brojeve to se ionako zna. Isto tako, uzimaćemo obično $b = 2$ ili $b = 3$ ili $b = 5$ ili $b = 7$. Drugim riječima, u algoritmu, obično se uzima da je baza b – neki vrlo mali prost broj.

Pogledajmo primjere. Primjer: $b = 2$ i $n = 21$ (znamo da je n složen). Tada je $y = b^{n-1} \bmod n = 2^{20} \bmod 21 = 4$. Imamo $y \neq 1$ pa se kaže da n nije prošao test (da n nije prošao test za bazu $b = 2$). Imamo da je $y \neq 1 \Rightarrow n$ je složen.

Primjer: $b = 2$ i $n = 23$ (znamo da je n prost). Možemo pisati $y = b^{n-1} \bmod n = 2^{22} \bmod 23 = 1$ i bez računanja. Imamo $y = 1$ pa se kaže da je n prošao test.

Primjer: $b = 2$ i $n = 341$ (znamo da je n složen). Tada je $y = b^{n-1} \bmod n = 2^{340} \bmod 341 = 1$. Imamo $y = 1$ pa se kaže da je n prošao test. Završeni su primjeri.

Prilikom stepenovanja $y = b^{n-1} \bmod n$, radi lakšeg računanja, može se vršiti redukcija po modulu n djelimičnih proizvoda (čim pređe n) saglasno formuli $x_1 x_2 \bmod n = (x_1 \bmod n \cdot x_2 \bmod n) \bmod n$.

Definicija. Neka je b prirodan broj ($2 \leq b \leq n-1$). Neka je n složen broj. Ako je $b^{n-1} \bmod n = 1$ onda se kaže da je broj n – pseudo–prost broj u bazi b. Ili svejedno: ako je $b^{n-1} \equiv 1 \,(\mathrm{mod}\ n)$ onda se kaže da je broj n – pseudo–prost broj u bazi b.

Sigurno je $y = 1$ ili $y \neq 1$. Na stranu $y = 1$ dolaze prosti brojevi i pseudo–prosti brojevi (na stranu $y = 1$ dolaze prosti brojevi i brojevi koji su pseudo–prosti u bazi b). A na drugu stranu $y \neq 1$ dolaze svi ostali složeni brojevi.

Poznato je da je $\pi(10000) = 1229$, to je koliko ima prostih brojeva manjih od 10^4. Poznato je da do 10^4 ima 22 pseudo–prosta broja u bazi 2, ima toliko složenih brojeva koji prolaze test. Najmanji među njima je $n = 341$. Šta se dešava u bazi $b = 3$? Poznato je da postoje 23 broja koji su manji od 10^4 (svejedno: koji su manji ili jednaki od 10^4) i koji su peudo–prosti u bazi 3. Najmanji među njima je $n = 91$. Ima svega 7 brojeva do 10^4 koji su pseudo–prosti u obe baze $b = 2$ i $b = 3$, to je kardinalni broj presjeka dva skupa od maločas. Najmanji među njima je $n = 1105$. Dakle, ako je broj prošao onda je vjerovatnoća (onda su šanse) da je taj broj lažnjak jednake $\frac{7}{1229+7}$.

Ponovimo da su pseudo–prosti brojevi – složeni brojevi. Vidjeli smo da prosti brojevi i pseudo–prosti brojevi prolaze test. Što je udio pseudo–prostih brojeva u brojevima koji prolaze test manji, to su okolnosti povoljnije. Zbog postojanja pseudo–prostih brojeva, Millerov postupak samo nepotpuno rješava razmatrani zadatak o ispitivanju primalnosti datog broja.

Millerov postupak je vrlo jednostavan. Neka je n broj čija se primalnost ispituje. Izabere se baza b i izračuna se $y = b^{n-1} \bmod n$. Ako je izašlo $y \neq 1$ onda je broj n sigurno složen i to se saopštava kao odgovor. A ako je izašlo $y = 1$ onda je moguće i jedno i drugo, s tim da su veće šanse da je n prost. Šanse da složen broj prođe smanjuju se ako se upotrebi više baza.

Neka je $b_1 = 2$, $b_2 = 3$, $b_3 = 5$, $b_4 = 7$, $b_5 = 11$, $b_6 = 13$, $b_7 = 17$, $b_8 = 19$, ..., tj. neka je $\{b_n\}_{n=1}^{\infty}$ niz svih prostih brojeva (uređenih po veličini).

Algoritam: Millerov postupak:

```
      read(n, k);
      write "testira se po bazama b_1, ..., b_k";
      for i ← 1 to k do
      begin
      b ← b_i;
      if (b, n) > 1 then goto out;
      if b^{n-1} mod n ≠ 1 then goto out
      end;
      write "broj n je prošao test";
      write "vjerovatno, broj n je prost";
      stop;
out:  write "broj n nije prošao test";
      write "broj n je složen";
      stop
```

Ako je $(b, n) > 1$, tj. ako b i n nisu uzajamno prosti onda je n složen.

Obično se uzima da je $k = 4$ ili $k = 8$.

Kolika je vremenska složenost? Program se izvrši za čas–posla, algoritam je efikasan.

Definišimo događaje $A = \{n \text{ je složen}\}$ i $B = \{n \text{ je prošao}\}$ i uvedimo oznaku za uslovnu vjerovatnoću: $p = P(A|B)$, č. vjerovatnoća da je n složen, pod uslovom da je n prošao. Mogao bi da se da precizan izraz za p, što se ovdje izostavlja. Kao što je već rečeno, veličina p je bliska nuli, ali je ipak $p > 0$. Bilo bi lijepo da je $p = 0$ jer bi tada predloženi postupak rješavao u pravom smislu riječi zadatak o primalnosti.

Millerov postupak se ne koristi u praksi, zato što postoji dograđeni algoritam koji ima bolje karakteristike. To je tzv. Miller–Rabinov postupak. On se koristi u praksi. Ima bolje karakteristike – to znači da je veličina p još manja (još bliža nuli). Ipak, i kod Miller–Rabinovog postupka imamo da je $p > 0$, tako da i taj drugi postupak samo djelimično rješava zadatak o primalnosti.

6.8. ISPITIVANJE PRIMALNOSTI: MILLER–RABINOV POSTUPAK

U ovom naslovu biće definisan Miller–Rabinov postupak za ispitivanje primalnosti i biće navedene njegove karakteristike. Predstavlja poboljšanje Millerovog postupka i često se koristi u praksi.

Neka je n prirodan broj čija se primalnost ispituje. Slobodno možemo smatrati da je $n \geq 20$, kao i da je n neparan. Razmotrimo jednačinu $x^2 \bmod n = 1$ po nepoznatoj $x \in \{0, 1, \ldots, n-1\}$. Odmah se vidi da su $x = 1$ i $x = -1$ rješenja te jednačine (da su $x = 1$ i $x = n - 1$ rješenja te jednačine). Uslov $x = 1$ možemo zapisati kao $x \equiv 1 \pmod{n}$. Uslov $x = n - 1$ možemo zapisati kao $x \equiv n - 1 \pmod{n}$ ili svejedno kao $x \equiv -1 \pmod{n}$. Slično, polaznu jednačinu možemo zapisati kao $x^2 \equiv 1 \pmod{n}$.

Teorema. Neka je broj n prost. Razmotrimo jednačinu $x^2 \bmod n = 1$ po nepoznatoj $x \in \{0, 1, \ldots, n-1\}$. Jedina dva rješenja te jednačine su $x = 1$ i $x = n - 1$.

Dokaz teoreme se izostavlja, mada je dokaz prilično jednostavan.

Neka je b baza (baza je obično prost broj). Neka je n neparan broj čija se primalnost ispituje. Odredimo brojeve s i t iz jednakosti $n - 1 = 2^s t$, gdje je t neparan. Definišimo brojeve $x_0, x_1, \ldots, x_s$:

$$x_0 = b^t \bmod n, \;\; x_1 = x_0^2 \bmod n, \;\; x_2 = x_1^2 \bmod n, \;\; \ldots, \;\; x_s = x_{s-1}^2 \bmod n.$$

Umjesto $x_r = n - 1$ ponekad ćemo pisati $x_r = -1$ jer se ta dva broja poklapaju mod n. Zapazimo odmah da je $x_s = b^{n-1} \bmod n$. Ako je n prost ili pseudo–prost onda je $x_s = 1$. Ako je $x_s \neq 1$ onda je n složen.

Pogledajmo detaljnije slučaj kada je $x_s = 1$.

Ako je neki član u nizu $(x_0, x_1, \ldots, x_s)$ jednak -1 onda je idući član u nizu sigurno jednak 1 jer je $(-1)^2 = 1$. Slično, poslije $x_r = 1$ sigurno dolazi $x_{r+1} = 1$ jer je $1^2 = 1$.

Pogledajmo brojeve $x_s, x_{s-1}, x_{s-2}, \ldots$ i potražimo gdje se prvi put pojavljuje broj koji je različit od 1. Dakle, dopustimo da je za neko r: $x_{r-1} \neq 1$ i $x_r = 1$. Ako je broj n prost onda mora biti $x_{r-1} = -1$, po prethodnoj teoremi. Dakle, ako je $x_{r-1} \neq -1$ onda je broj n sigurno složen. A ako je $x_{r-1} = -1$? Tada je veoma vjerovatno da je n prost. Jednostavno, tada se kaže da je broj n prošao Miller–Rabinov test za bazu b. Drukčije rečeno, u skladu sa definicijom koja slijedi, tada je n prost ili je n jaki pseudo–prost broj u bazi b.

Definicija. Neka je b prirodan broj ($2 \leq b \leq n - 1$). Neka je n neparan složen broj. Ako je $(x_0, x_1, \ldots, x_s) = (1, 1, \ldots, 1)$ ili postoji r da je $x_{r-1} = n - 1$ i $x_r = 1$ onda se kaže da je broj n jaki pseudo–prost broj u bazi b.

Druga formulacija: Definicija. Za neparan složen broj n kaže se da je jaki pseudo–prost broj u bazi $b \in \{2, 3, 5, 7, \ldots\}$ ako je bilo $b^t \equiv 1 \pmod{n}$ ili postoji r, $0 \leq r \leq s - 1$, takav da je $b^{2^r t} \equiv -1 \pmod{n}$, gdje je $n - 1 = 2^s t$ i t je neparan.

Lako se vidi da važi tvrđenje: n je jaki pseudo–prost broj u bazi $b \Rightarrow n$ je pseudo–prost broj u bazi b.

Izvodeći test za razne baze b (za jedno te isto n) smanjuju se šanse da lažnjak prođe (smanjuju se šanse da složen broj n prođe test). Naime, očito je da će broj n koji je prost – proći test za bilo koju bazu. To isto ne može se reći za broj n koji je složen.

Neka je $b_1 = 2$, $b_2 = 3$, $b_3 = 5$, $b_4 = 7$, $b_5 = 11$, $b_6 = 13$, $b_7 = 17$, $b_8 = 19$, ..., tj. neka je $\{b_n\}_{n=1}^{\infty}$ niz svih prostih brojeva (uređenih po veličini).

U programu se ispituje da li n prolazi u svim bazama $b_1, \ldots, b_k$.

Algoritam: Miller–Rabinov postupak:

```
read(n, k);
write "testira se po bazama b_1, ..., b_k";
for i ← 1 to k do
begin
b ← b_i;
if (b, n) > 1 then goto out;
if b^{n-1} mod n ≠ 1 then goto out;
izračunaj niz brojeva x_0, x_1, ..., x_s;
if niz brojeva nema predviđeni oblik then goto out
end;
```

write "broj n je prošao test";
write "veoma vjerovatno, broj n je prost";
stop;
out: write "broj n nije prošao test";
write "broj n je složen";
stop

Ponovimo na primjeru $s = 5$ šta znači da niz brojeva ima predviđeni oblik. To znači da taj niz brojeva $(x_0, x_1, x_2, x_3, x_4, x_5)$ glasi $(1,1,1,1,1,1)$ ili $(-1,1,1,1,1,1)$ ili $(x,-1,1,1,1,1)$ ili $(x,x,-1,1,1,1)$ ili $(x,x,x,-1,1,1)$ ili $(x,x,x,x,-1,1)$. Ovdje je $x \neq -1$ $(x \neq n-1)$, x je generička oznaka. Jasno je da je $x \neq 1$ jer je $1^2 = 1$.

Kao primjer, pogledajmo primjenu Miller–Rabinovog postupka na brojeve $n < 10000$. Pogledajmo prvo slučaj baze $b = 2$. U literaturi piše da ima 22 pseudo–prosta broja do 10^4 i da je među njima najmanji $n = 341$ (nije jaki pseudo–prost u toj bazi). Znamo da ne može biti jaki pseudo–prost ako nije pseudo–prost, kada se radi o jednoj te istoj bazi. U literaturi piše da među ta 22 broja – njih 5 su jaki pseudo–prosti brojevi. Drugim riječima, ima 5 jakih pseudo–prostih brojeva do 10^4 u bazi $b = 2$. Tih 5 brojeva su: 2047, 3277, 4033, 4681, 8321. U tabeli je prikazan niz $(x_0, x_1, \ldots, x_s)$ za $n = 341$ i za ovih pet. Prelazimo na slučaj baze $b = 3$, isto posmatramo do 10^4. U literaturi piše da ima 23 pseudo–prosta broja (među njima je najmanji $n = 91$ i nije jaki), a od njih su jaki pseudo–prosti: 121, 703, 1891, 3281, 8401, 8911. Ima ih 6 na broju. U tabeli je prikazano za $n = 91$ i za ovih šest. Pogledajmo sada šta se dešava kada ukrstimo baze $b = 2$ i $b = 3$, odnosno kada primijenimo Miller–Rabinov postupak sa parametrom $k = 2$ (testira se po bazama b_1 i b_2) u slučaju da je ulazni podatak $n < 10^4$. Lako vidimo da petočlani i šestočlani skup nemaju zajedničkih članova. Prema tome, svi brojevi n za koje program saopštava "veoma vjerovatno, broj n je prost" – istinski su prosti (kada je $k = 2$ i $n < 10^4$).

U tabeli se koriste oznake koje su već uvedene: $n - 1 = 2^s t$, t neparan, $x_0 = b^t \bmod n$, $x_r = x_{r-1}^2 \bmod n$ za $1 \leq r \leq s$, tako da je $x_1 = b^{2t} \bmod n$, $x_2 = b^{4t} \bmod n$, $x_3 = b^{8t} \bmod n$, ... U tabeli, $x = -1$ je zamjena za $x = n - 1$.

	b	n	t	s	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	zaključak
1	2	341	85	2	32	1	1	.....	.....	.....	.....	.....	nije prošao
2	2	2047	1023	1	1	1	.....	.....	.....	.....	.....	.....	prošao je
3	2	3277	819	2	128	-1	1	.....	.....	.....	.....	.....	prošao je
4	2	4033	63	6	3521	-1	1	1	1	1	1	.....	prošao je
5	2	4681	585	3	1	1	1	1	.....	.....	.....	.....	prošao je
6	2	8321	65	7	8192	-1	1	1	1	1	1	1	prošao je
7	3	91	45	1	27	1	.....	.....	.....	.....	.....	.....	nije prošao
8	3	121	15	3	1	1	1	1	.....	.....	.....	.....	prošao je
9	3	703	351	1	-1	1	.....	.....	.....	.....	.....	.....	prošao je
10	3	1891	945	1	-1	1	.....	.....	.....	.....	.....	.....	prošao je
11	3	3281	205	4	3038	3272	81	-1	1	.....	.....	.....	prošao je
12	3	8401	525	4	-1	1	1	1	1	.....	.....	.....	prošao je
13	3	8911	4455	1	-1	1	.....	.....	.....	.....	.....	.....	prošao je

Ponovimo da su svi brojevi n koji se u tabeli pojavljuju – složeni brojevi. Vidimo da u tabeli preovlađuju "negativni primjeri" (preovlađuju jaki pseudo–prosti brojevi, brojevi koji

prolaze test za jednu datu bazu b). To ne treba smatrati slabošću tabele. Naime, nešto lakše je naći brojeve n za koje je $x_{r-1} \neq -1$ i $x_r = 1$ (takvi brojevi n su složeni).

Kako se ponaša Miller–Rabinov test kada je u pitanju broj n sa 10 ili 20 ili više dekadnih cifara (malo treba parametar k povećati)? Primjera radi, pokazano je sljedeće: neka je n najmanji prirodan broj koji je jaki pseudo–prost broj u svih osam najmanjih baza (u svim bazama od $b_1 = 2$ do $b_8 = 19$). Za taj broj n važi ocjena $n > 10^{14}$.

Sada je na redu finalna priča o Miller–Rabinovom postupku, koja će pokazati da je taj postupak ustvari bolji od onog kako je dosad rečeno (da je možda bolji od onog kako je dosad rečeno). Prethodno, dimenzioni broj instance (obim ulaza) definiše se kao broj cifara ulaznog podatka n (u binarnom ili dekadnom zapisu), znamo da je broj cifara reda $\ln n$. Može se pokazati da vremenska složenost testa za jednu bazu ima red veličine $\ln^3 n$. To znači da je ta složenost – polinomska. Jasno, vremenska složenost testa za k baza ima red veličine $k \ln^3 n$. Važi sljedeća teorema:

Teorema. Neka je n prirodan broj. Neka n prolazi test za sve baze b takve da je $b < 2\ln^2 n$. Ako je tzv. uopštena Rimanova hipoteza tačna onda je broj n – prost broj.

Drugim riječima, ako je n složen onda n nije jaki pseudo–prost broj bar u jednoj bazi $b < 2\ln^2 n$ pod uslovom da je istinita uopštena Rimanova hipoteza. Prema tome, dopustimo da važi uopštena Rimanova hipoteza i uzmimo da se test sprovodi za sve navedene baze, tj. da u tekstu programa umjesto read(k) piše: izaberi k tako da važi $b_k > 2\ln^2 n$. Tada, u tekstu programa, umjesto write "veoma vjerovatno, broj n je prost" može se staviti write "broj n je prost". Vremenska složenost ostaje polinomska jer je $2\ln^2 n \cdot O(\ln^3 n) = O(\ln^5 n)$, prostih brojeva (baza) do $2\ln^2 n$ ima manje od $2\ln^2 n$. Dakle, složenost je $T(m) = O(m^5)$, $m \geq 1$.

Množenje dva broja čije su veličine po m bita uzima m^2 taktova.

U zaključku, Miller–Rabinov algoritam je efikasan algoritam koji u potpunosti rješava zadatak o primalnosti, opet: ako važi uopštena Rimanova hipoteza (Riemann).

Sasvim na kraju, uslov Miller–Rabinovog testa očito je strožiji od uslova Millerovog. Ključnu ulogu kod Millerovog ima tvrđenje: ako je b prost, $b < n$, nije $b \mid n$, $y = b^{n-1} \bmod n$ i $y \neq 1$ onda je n složen. A kod Miller–Rabinovog: ako je b prost, $b < n$, nije $b \mid n$, $y = b^{n-1} \bmod n$ i $y \neq 1$ ili $y = 1$ i $(*)$ onda je n složen. Uslov $(*)$ glasi: u nizu $x_0 = b^t \bmod n$, $x_1 = b^{2t} \bmod n$, $x_2 = b^{4t} \bmod n$, …, $x_{s-1} = b^{(n-1)/2} \bmod n$, $x_s = b^{n-1} \bmod n$ nisu svi brojevi $= 1$ i u tom nizu nema broja koji bi bio $= -1$ (koji bi bio $= n-1$); t neparan. Znamo da $b \mid n$ znači b dijeli n.

6.9. STREAM CIPHER

Kriptografski sistemi dijele se u dvije kategorije: sa javnim ključem i sa tajnim ključem. U ovom naslovu govori se (makar površno) o sistemima sa tajnim ključem. Kod sistema sa tajnim ključem, koriste se razne ideje, a mi ćemo razmotriti jednu mogućnost: engl. stream cipher ili u prevodu – kodiranje pomoću niza. Izlaganje se sastoji iz dva dijela: slučaj kada je niz unaprijed dat i slučaj kada se niz definiše zajedno sa porukom.

Prvi dio izlaganja: list za jednokratnu upotrebu ili engl. one–time pad.

Najprostiji i najsigurniji način. Koristi se Bulova funkcija XOR (isključivo ILI) ili svejedno $\oplus$ (sabiranje po modulu 2). Da izložimo kriptografski sistem. Alisa i Bob se sastanu i na jednom listu papira napišu $N \geq 1$ bita (N binarnih cifara) na slučajan način. Označimo niz kao $s_1, s_2, \ldots, s_N$, gdje $s_i \in \{0, 1\}$ za $1 \leq i \leq N$, kao što je već rečeno. Navedeni niz predstavlja tajni ključ. Oni isto prepišu na drugi list, tako da sada imaju dva jednaka primjerka. Svakome

po jedan primjerak. Time su oni izvršili sve pripreme da Alisa može (za nekoliko dana) da pošalje Bobu jednu poruku (dužine N bita) u kodiranom obliku. Šta će se desiti kada prođe nekoliko dana? Alisa prvo sastavi poruku kao niz bita $x_1x_2\ldots x_N$. Zatim izračuna $y_i = x_i \oplus s_i$ za $1 \le i \le N$. Ona uputi $y_1y_2\ldots y_N$ preko računarske mreže. Čim dobije kodirani oblik poruke, Bob će da uradi sljedeće. Izračunaće $z_i = y_i \oplus s_i$ za $1 \le i \le N$. Lako se vidi da je on time ostvario dekodiranje (saznao originalnu poruku), odnosno da važi $z_i = x_i$. Zaista, $z_i = y_i \oplus s_i = (x_i \oplus s_i) \oplus s_i = x_i$. Znamo da je $(x + 2s) \bmod 2 = x \bmod 2$.

Ako je $N = 1000$ onda poruka ima 125 bajta, odnosno poruka se sastoji od 125 karaktera (po propisu 8–bitni ASCII) ili manje.

Analiza bezbjednosti sistema. Pretpostavimo da je treća osoba saznala kodirani oblik poruke $y_1y_2\ldots y_N$. To joj ništa ne vrijedi, na osnovu toga ne može ništa da se zaključi o originalnoj poruci $x_1x_2\ldots x_N$. Jednake su šanse da je $x_i = 0$ odnosno da je $x_i = 1$ za svako i. Dakle, razmatrani kriptografski sistem je potpuno siguran, totalno neprobojan.

Drugi dio izlaganja: slučaj kada se niz $s_1, s_2, \ldots, s_N$ generiše ili LFSR.

Postoje razne ideje da tajni ključ bude puno kraći od N bita, a mi ćemo razmotriti jednu mogućnost: engl. linear feedback shift register (LFSR) ili u prevodu – šift registar (pomerački registar). Neka $n \ge 1$ označava veličinu šift registra (broj flipflopova). Sada se tajni ključ sastoji od n bita, u oznaci $s_1, \ldots, s_n$. Recimo, može da bude $n = 4$ ili $n = 8$ ili $n = 16$. Vidjećemo da se pomoću tajnog ključa lako generiše niz $s_1, s_2, \ldots, s_N$ koji služi za enkripciju i dekripciju, gdje je $N = 2^n - 1$. Kažimo unaprijed da se postupak enkripcije poklapa sa prethodnim slučajem lista za jednokratnu upotrebu ($y_i = x_i \oplus s_i$), a isto tako se poklapa i postupak dekripcije ($z_i = y_i \oplus s_i$). Suvišno je ponavljati, $x_1x_2\ldots x_N$ je originalna poruka (nekodirani oblik poruke), $y_1y_2\ldots y_N$ je kodirani oblik poruke, a $z_1z_2\ldots z_n$ je dekodirana poruka. Važi $z_i = x_i$ za $i = 1, 2, \ldots, N$.

Treba objasniti kako se tajni ključ $s_1, \ldots, s_n$ produžava do niza koji ima $N = 2^n - 1$ članova.

Poslužimo se primjerom ($n = 4$). Razmotrimo šift registar od 4 bita i označimo njegovo tekuće stanje (stanje u taktu $t = i$) kao $\boxed{a_4}\boxed{a_3}\boxed{a_2}\boxed{a_1}$, a njegovo naredno stanje ($t = i + 1$) sa $\boxed{a'_4}\boxed{a'_3}\boxed{a'_2}\boxed{a'_1}$. Radi se o registru u kome se vrši pomijeranje udesno. Tada je $a'_3 = a_4$, $a'_2 = a_3$ i $a'_1 = a_2$. Znamo da je $a'_4 = a_1$ ako je pomijeranje kružno, odnosno da je $a'_4 = 0$ ako nije kružno. Isto tako znamo da se a_1 saopštava. U svakom taktu, na izlaz se šalje sadržaj krajnjeg desnog flipflopa. Za naše potrebe, izvršimo modifikaciju registra. Neka bude $a'_4 = a_1 \oplus a_2$, v. sliku. U tabeli su prikazani stanje u registru iz takta u takt i vrijednosti s_i koje se redom saopštavaju na izlazu, a uzeto je da je početno stanje (takt $i = 1$) $s_4s_3s_2s_1 = a_4a_3a_2a_1 = 1000$. Vidimo da se stanje kada je $i = 16$ poklapa sa početnim stanjem.

takt i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
stanje $a_4a_3a_2a_1$	1000	0100	0010	1001	1100	0110	1011	0101	1010	1101	1110	1111	0111	0011	0001	1000
izlaz $s_i = a_1$	0	0	0	1	0	0	1	1	0	1	0	1	1	1	1	0

Opet: ako je u i–tom taktu stanje (sadržaj registra) $\boxed{a_4}\boxed{a_3}\boxed{a_2}\boxed{a_1}$ onda je u $(i + 1)$–vom taktu $\boxed{a_1 \oplus a_2}\boxed{a_4}\boxed{a_3}\boxed{a_2}$. U svakom taktu (u i–tom taktu) na izlaz se daje a_1. Kaže se da a_1 i a_2 utiču i da odgovarajući polinom glasi $p(x) = x^4 + x^3 + 1$.

Primjer ($n = 8$) $\boxed{a_8}\boxed{a_7}\boxed{a_6}\boxed{a_5}\boxed{a_4}\boxed{a_3}\boxed{a_2}\boxed{a_1}$, $a'_j = a_{j+1}$ za $j = 1, \ldots, 7$, $a'_8 = a_1 \oplus a_5 \oplus a_6 \oplus a_7$, $p(x) = x^8 + x^4 + x^3 + x^2 + 1$.

Primjer ($n = 16$) $a'_{16} = a_1 \oplus a_5 \oplus a_{14} \oplus a_{16}$, $p(x) = x^{16} + x^{12} + x^3 + x + 1$.

U opštem slučaju (ma koje n):

$$s_{n+i} = (c_1 s_{n-1+i} + c_2 s_{n-2+i} + \ldots + c_n s_i) \bmod 2, \quad i \geq 1,$$

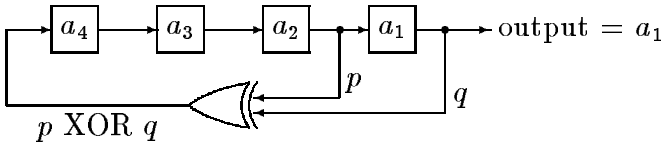
$c_j \in \{0,1\}$, $p(x) = c_n x^n + \ldots + c_2 x^2 + c_1 x + 1$, polazi se od $s_1, \ldots, s_n$. Na izlazu registra imamo redom iz takta u takt jedan po jedan bit, jedan po jedan član niza $s_1, s_2, \ldots, s_N$ koji služi za enkripciju, kao uostalom i za dekripciju.

U sva tri navedena primjera, perioda je maksimalna moguća, ona iznosi $N = 2^n - 1$. Maksimalnu periodu imamo samo za neke polinome. Takve polinome nazivamo primitivnim. Za većinu polinoma, perioda je puno kraća. Oni su slabiji sa stanovišta kriptografije jer je kraća poruka koja može da bude kodirana. Matematički dio izlaganja koji je sada potreban (algebra) – izostavlja se. Gleda se da se izabere primitivni polinom p koji ima mali broj sabiraka, da bi računanje po formuli $s_{n+i} = \ldots$ kraće trajalo. Dalje, može se pokazati da generisani niz $s_1, s_2, \ldots, s_N$ ima dobra svojstva u pogledu distribucije nula i jedinica, što je značajno za bezbjednost kriptografskog sistema. Zato se smatra da je to jedan niz pseudo–slučajnih brojeva. Samo ne treba kao početno stanje izabrati $(s_1, \ldots, s_n) = (0, \ldots, 0)$.

Ako je $n = 16$ onda pseudo–slučajni niz ima 2^{16} bita (ustvari, ima jedan bit manje), tako da se može kodirati poruka veličine do $2^{13} = 8192$ karaktera.

Prelazimo na zaključke. Vidi se da su postupci za enkripciju i dekripciju jednostavni. Oni se sprovode brže nego u slučaju javnog ključa. Predložena metoda (šift registra) može lako da bude i hardverski implementirana. Time se svakako dobija na brzini.

Analiza bezbjednosti sistema? Imamo sljedeće okolnosti. Opšte–poznat je algoritam (šift registra), kao i formula oblika s_{n+i} po kojoj se računa. Takođe se pretpostavlja da je treća osoba saznala kodirani oblik poruke $y_1 y_2 \ldots y_N$ (jedino ne zna tajni ključ $s_1, \ldots, s_n$). U takvoj situaciji, sistemi sa tajnim ključem imaju slabija svojstva bezbjednosti (sigurnosti) od sistema sa javnim ključem. Završen je drugi dio izlaganja.



Vidjeli smo da je potrebno da Alisa i Bob imaju zajednički (podijeljeni) tajni ključ $s_1, \ldots, s_n$. Mogu li oni da dođu do tog ključa, a da se ne sastaju? Postoji način. Treba koristiti neku metodu tipa javnog ključa (recimo RSA). Dakle, Alisa će prvo sama da izabere tajni ključ $s_1, \ldots, s_n$ ($s_j \in \{0,1\}$ za $j = 1, \ldots, n$), recimo $n = 16$. Zatim će da pošalje u prvom fajlu, naravno u kodiranom obliku, recimo pomoću RSA načina, ključ koji je izabrala. Onda će u drugom fajlu da pošalje i $y_1 y_2 \ldots y_N$. Zaključak: vidimo da mogu da se kombinuju kriptografski sistemi tipa tajnog ključa i tipa javnog ključa. Za kriptografske sisteme tipa javnog ključa ponekad se kaže da su sistemi koji služe za razmjenu ključeva.

Stoljećima unazad koristili su se sistemi sa tajnim ključem. Njihova bezbjednost bazira se na tajnosti podataka koji služe za enkripciju i dekripciju. Tek sedamdesetih godina pojavili su se sistemi sa javnim ključem: u javnom domenu ima dovoljno podataka da bi se sistem "otključao", samo što treba puno puno vremena.

6.10. PRATTOV SERTIFIKAT PRIMALNOSTI

U ovom naslovu uvodi se pojam sertifikata uopšte i zatim se još govori o jednom mogućem sertifikatu za zadatak o ispitivanju da li je dati broj prost (Prattov rezultat). Vidjećemo da se ovdje ne razmatra zadatak: dat je prirodan broj, odgovoriti sa "da" ili "ne" na pitanje – da li je taj broj prost, već se razmatra sljedeći problem: dat je prost broj n, a želim da provjerim da li je n zaista prost, želim da budem siguran da je n zaista prost broj.

Razmotrimo skup svih prirodnih brojeva N i razmotrimo njegov podskup $\{2,3,5,7,\ldots\}$ – skup svih prostih brojeva. Uvedimo neko kodiranje za članove skupa N, recimo pomoću dvočlane azbuke $A_2 = \{0,1\}$, tj. pomoću binarnog brojnog sistema. Tada imamo jezik $L = \{10,11,101,111,\ldots\} \subset \Omega(A_2)$.

Pogledajmo primjer za Prattov sertifikat primalnosti. Neka je $n = 7919$ (ovaj broj je prost). Neka je $p_1 = 2$, $p_2 = 37$, $p_3 = 107$, ovi brojevi su prosti i važi $p_1 p_2 p_3 = n - 1$. Neka je $a = 7$. Važi

$$a^{n-1} \equiv 1 \,(\mathrm{mod}\ n) \text{ i } a^{(n-1)/p_1} \not\equiv 1 \,(\mathrm{mod}\ n),\ a^{(n-1)/p_2} \not\equiv 1 \,(\mathrm{mod}\ n),\ a^{(n-1)/p_3} \not\equiv 1 \,(\mathrm{mod}\ n).$$

Veličine a, p_1, p_2, p_3 čine sertifikat za n, uz pozivanje na jednu teoremu iz teorije brojeva. Te veličine čine sertifikat za izjavu "broj $n = 7919$ je prost". Razumije se da te veličine treba kodirati po nekoj šemi za kodiranje kao riječ c u azbuci A_2 ili u nekoj većoj azbuci, isto kao što se broj n kodira pomoću riječi $w \in \Omega(A_2)$. Dakle, ulazni podaci algoritma su dvije riječi c i w. Drukčije rečeno, početna pozicija na Tjuringovoj traci je $]\star c \star w \underset{\uparrow}{\star} \ldots$, gdje je $w = (7919)_2$. Za navedene c i w. program (algoritam) će saopštiti kao rezultat "da", tj. saopštiće da c sertifikuje da je n prost (c *certifies*). Ustvari, c treba dopuniti sa sličnim sertifikatima za izjave "$p_1 = 2$ je prost", "$p_2 = 37$ je prost", "$p_3 = 107$ je prost". Za p_3, to su veličine 2, 2, 53. Smatramo da nikakva potvrda ne treba za proste brojeve manje od 100. Dakle, u primjeru se govori o jednoj mogućoj proceduri za provjeru (*checking procedure*) da $w \in L = \{10,11,101,111,\ldots\}$.

Navedimo maločas spomenutu teoremu iz teorije brojeva.

Teorema. Neka je $n \geq 3$ prirodan broj. Uočimo rastavljanje broja $n-1$ na proste činioce: $n - 1 = p_1 \cdot p_2 \cdot \ldots \cdot p_k$, gdje među prostim brojevima $\{p_i\}_{i=1}^{k}$ može da bude ponavljanja. Broj n je prost ako i samo ako postoji broj a $(2 \leq a \leq n-1)$ takav da važi $a^{n-1} \equiv 1 \,(\mathrm{mod}\ n)$ i da za svako $i \in \{1,2,\ldots,k\}$ važi $a^{(n-1)/p_i} \not\equiv 1 \,(\mathrm{mod}\ n)$. Za a se kaže da je svjedok (*witness*).

Definicija. Neka je $t \geq 1$, neka je $A_t = \{a_1,\ldots,a_t\}$ azbuka sa t članova i neka je $\Omega(A_t)$ skup svih riječi u azbuci A_t. Neka je L jezik u azbuci A_t, tj. neka je $L \subset \Omega(A_t)$. Razmotrimo algoritam $\mathcal{A}$, tj. razmotrimo Tjuringovu mašinu čija je radna azbuka nadskup od A_t i koja posjeduje sljedeća dva svojstva. Prvo svojstvo: za svaku riječ $w \in L$ postoji riječ c takva da važi

$$]\star c \star w \underset{\uparrow}{\star} \ldots \quad \Rightarrow \quad]\sim a_0\, a_1\, \underset{\uparrow}{a_0}.$$

Isto je kazati da će završna pozicija biti $]\sim a_0\, a_1\, \underset{\uparrow}{a_0}$ (ili svejedno da će se mašina zaustaviti poslije riječi a_1) ili kazati da će mašina saopštiti "da" kao rezultat (takvu konvenciju uzimamo). Za riječ c kaže se da predstavlja sertifikat. Pored toga, drugo svojstvo: za svaku riječ $w \in \Omega(A_t) \setminus L$, ne postoji riječ c u azbuci Tjuringove mašine za koju bi važila prethodna implikacija ($\Rightarrow$). Drugim riječima, za bilo koju riječ $w \notin L$ i bilo koju riječ c, u slučaju da je početna pozicija mašine $]\star c \star w \underset{\uparrow}{\star} \ldots$, biće da mašina vječno radi ili da se zaustavlja ali ne poslije neke riječi ili da se

zaustavlja poslije riječi koja je različita od a_1. Za algoritam $\mathcal{A}$ kaže se da predstavlja *proceduru za provjeru* za jezik L.

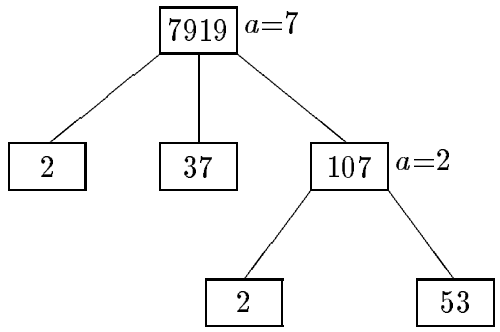
Uz korišćenje navedene teoreme iz teorije brojeva, lako se konstruiše procedura za provjeru $\mathcal{A}$ za jezik L kome pripadaju prosti brojevi, odnosno binarni (ili dekadni) prikazi–kodovi prostih brojeva. Ne obraćamo puno pažnje na šemu za kodiranje jer se samo po sebi razumije da je uvijek potreban neki sistem za prikazivanje podataka i da je to pitanje jednostavno. O toj proceduri za provjeru $\mathcal{A}$ već je bilo riječi kroz primjer $n = 7919$.

Njen sertifikat $c = c(w)$ sastoji se iz tri dijela: svjedok a, prosti činioci $p_1, p_2, \ldots, p_k$ broja $n-1$ i sertifikati za p_i $(i = 1, 2, \ldots, k)$. Jasno je da sertifikat za p_i ima sličan oblik, tako da obuhvata i proste činioce broja $p_i - 1$, itd. Tako da se c može prikazati kao jedno drvo.

Koje računske radnje izvodi procedura za provjeru $\mathcal{A}$ tokom svog rada? Početna pozicija na traci je $]\star c \star w \underset{\uparrow}{\star} \ldots$. Ona prvo izračuna proizvod $y = p_1 \cdot p_2 \cdot \ldots \cdot p_k$ i uvjeri se da je zaista $y = n - 1$. Ako bi došlo do odstupanja (do kratkog spoja), tj. ako bi se ispostavilo da je $y \neq n-1$ onda neka mašina (recimo) izbriše čitavu traku i zaustavi se na samom početku trake. Zatim se uvjeri da je zaista $a^{n-1} \equiv 1 \,(\mathrm{mod}\ n)$. Zatim se uvjeri da je $a^{(n-1)/p_i} \not\equiv 1 \,(\mathrm{mod}\ n)$ za svako $i = 1, 2, \ldots, k$. Zatim prelazi na verifikaciju primalnosti broja p_1 (kasnije će za $p_2, \ldots, p_k$). Ta verifikacija vrši se na sličan način. Uzmimo da se tako čitavo drvo potrošilo i da nigdje nije bilo kratkog spoja. Tada neka algoritam $\mathcal{A}$ saopšti "da" i zaustavi se.

Na slici je prikazano drvo c koje odgovara instanci $n = 7919$. Pored slike prikazane su računske radnje koje $\mathcal{A}$ uradi tokom svog izvršavanja.

$2 \cdot 37 \cdot 107 = 7918$, $7^{7918} \bmod 7919 = 1$
$7^{7918/2} \bmod 7919 \neq 1$, $7^{7918/37} \bmod 7919 \neq 1$, $7^{7918/107} \bmod 7919 \neq 1$
$2 \cdot 53 = 106$, $2^{106} \bmod 107 = 1$
$2^{106/2} \bmod 107 \neq 1$, $2^{106/53} \bmod 107 \neq 1$



Slika: Verifikacija primalnosti

U nastavku se govori o procedurama za provjeru za neke druge jezike, o definiciji klase $\mathcal{NP}$ i o Prattovom rezultatu.

Razmotrimo jezik $L \subset N$ kome pripadaju složeni brojevi. Jedna moguća procedura za provjeru za taj jezik predviđa da, kada je dat složen broj n, njegov sertifikat čine dva broja n_1 i n_2 takvi da je $n_1 > 1$, $n_2 > 1$ i $n = n_1 n_2$. Druga procedura za provjeru za taj jezik može lako da se konstruiše na osnovu sljedećeg poznatog tvrđenja:

Tvrđenje: Neka je n neparan složen broj i neka je $n - 1 = 2^s t$, gdje je t neparan. Tada bar za jednu bazu b $(2 \leq b \leq n-1)$ i jedan broj r $(0 \leq r \leq s-1)$ važi $b^{(n-1)/2^{r+1}} \not\equiv \pm 1 \,(\mathrm{mod}\ n)$ i $b^{(n-1)/2^r} \equiv 1 \,(\mathrm{mod}\ n)$.

Razmotrimo zadatak o podjeli: da li dati prirodni brojevi $x_1, \ldots, x_n$ mogu da se podijele u dvije grupe tako da se poklapa zbir članova jedne i druge grupe. Jedna moguća procedura za provjeru za taj zadatak predviđa da, u slučaju n–torke brojeva za koju takva podjela postoji, kao sertifikat posluži n–torka $(c_1, \ldots, c_n)$, gdje $c_i \in \{0, 1\}$, takva da će se dva zbira poklopiti ako se broj x_i upiše u prvu grupu kada je $c_i = 0$, odnosno treba ga staviti u drugu grupu ako je $c_i = 1$, za svako $i = 1, \ldots, k$.

Prelazimo na $\mathcal{NP}$. Razmotrimo neki jezik L, jednu njegovu proceduru za provjeru $\mathcal{A}$ i početnu poziciju na traci $]\star c \star w \underset{\uparrow}{\star} \ldots$. Dogovorimo se o sljedeće tri važne okolnosti. Uzmimo da se veličina ulaza (dimenzioni broj instance) definiše kao broj slova riječi w, odnosno da pomoćna riječ c i njena dužina uopšte ne utiču (na definiciju veličine ulaza). Zatim, uzmimo da se prilikom računanja vremenske složenosti algoritma $\mathcal{A}$ uzimaju u obzir samo slučajevi kada $w \in L$, odnosno da dešavanja tokom rada algoritma u slučajevima $w \notin L$ ne utiču na bilo kakav način na definiciju složenosti. I još, neka se prilikom definicije složenosti, za dato $w \in L$, uzima u obzir samo slučaj najpovoljnijeg od svih sertifikata te riječi w. Najpovoljniji je onaj sertifikat koji potroši najmanje taktova dok se ne dostigne završna pozicija (znamo da će završna pozicija biti $]\sim a_0\, a_1 \underset{\uparrow}{a_0}$). Naime, jasno je da jednoj riječi w odgovara više pomoćnih riječi c (za koje će se saopštiti rezultat "da"), uopšte uzev. Da izrazimo pomoću formule:

$$T(n) = \max_{w \in L,\, |w|=n} \; \min_{c:\; Y(c)} |\mathrm{niz}(c, w)|,$$

gdje je $\mathrm{niz}(c, w)$ – niz pozicija kroz koje mašina prođe od početne do završne pozicije, a $Y(c)$ znači da će se u slučaju pomoćne riječi c saopštiti "da" kao rezultat (pomoćna riječ c je sertifikat). Još, stavlja se da je $T(n) = 1$ ako u jeziku L nema nijedne riječi dužine n.

Nakon usvojenog dogovora, možemo da krenemo dalje. Za jezik L kaže se da pripada klasi $\mathcal{NP}$ (klasa nedeterministički polinomskih jezika) ako postoji njemu odgovarajuća procedura za provjeru $\mathcal{A}$ čija je vremenska složenost polinomska, gdje prilikom računanja složenosti svakako imamo u vidu tri navedene okolnosti. Pomoću formule: za svako n važi $T(n) \leq p(n)$, gdje je $p = p(n)$ neki polinom. Kratko: $\mathcal{NP}$ – polinomska provjera.

Ovo je jedna od dvije mogućnosti koje postoje u literaturi da se definiše klasa jezika $\mathcal{NP}$ (klasa zadataka $\mathcal{NP}$). Druga mogućnost: preko pojma nedeterminističke Tjuringove mašine. Razlike između jedne i druge definicije su samo tehničke prirode. O tome i sličnom govori se u sljedećem poglavlju: Teorija NP–kompletnih zadataka.

Vratimo se zadatku o primalnosti. Teorema iz teorije brojeva koja je navedena – poznata je odavno. Vidimo da nema puno razlike između iskaza teoreme i procedure za provjeru $\mathcal{A}$. U tom smislu, može se reći da je i ta procedura poznata odavno. Pratt je dao procjenu za visinu drveta koje se pojavljuje i za druga svojstva tog drveta. Zatim je još i izračunao vremensku složenost procedure za provjeru, njenu složenost u maločas definisanom smislu. Pokazalo se da je ta složenost polinomska. U zaključku, za proceduru se kaže da je Prattova procedura, a za sertifikat koji se u njoj pojavljuje kaže se da je Prattov sertifikat primalnosti. Može se reći da je Pratt dokazao da zadatak "prime" pripada klasi $\mathcal{NP}$.

7. TEORIJA NP–KOMPLETNIH ZADATAKA

Već smo rekli da se zadatak naziva lakim ako za njega postoji algoritam čija je složenost polinomska, a u suprotnom slučaju se naziva teškim. Vidjećemo da je za mnoge zadatke otvoreno pitanje – postoji li takav algoritam, tj. pripada li zadatak klasi $\mathcal{P}$, tj. da li je taj zadatak lak ili težak. Biće uvedeni i nedeterministički modeli za računanje. U okviru toga, pojavljuje se jedna važna klasa zadataka – klasa NP–kompletnih zadataka. Vidjećemo da je i za tu klasu vezano jedno otvoreno pitanje. Prvo otvoreno pitanje svodi se ustvari na drugo otvoreno pitanje i obrnuto.

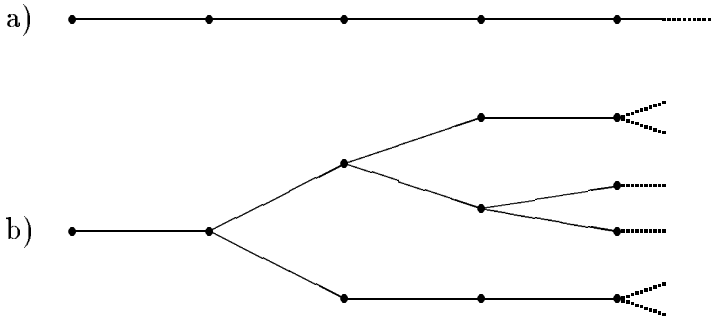
7.1. NEDETERMINISTIČKA TJURINGOVA MAŠINA

Kod nedeterminističke Tjuringove mašine (NDTM) se prilikom svakog takta umjesto pridruživanja jednog objekta iz $V^k \times Q$ može pridružiti nekoliko takvih objekata, čime nastaje nekoliko grana procesa obrade. Drukčije se može reći da je funkcija φ višeznačna, a kod obične ili prave ili determinističke Tjuringove mašine (TM) funkcija φ je kao što znamo jednoznačna, φ – tablica mašine. Obične oznake: $V = A \cup \{r, l, s\} = \{a_0\} \cup A_t \cup \{r, l, s\} = \{a_0\} \cup \{a_1, \ldots, a_t\} \cup \{r, l, s\}$ i $Q = \{q_0, q_1, \ldots, q_s\}$. Zajedno ćemo tretirati slučajeve $k = 1$ i $k > 1$, k – broj traka.

Definicija 1. NDTM sa k traka T definiše se svojom tablicom φ koja je jedno preslikavanje iz $Q \times A^k$ u $P(V^k \times Q)$. Ostali elementi definicije su isti kao u slučaju obične TM.

Znamo da $P(X)$ označava partitivni skup skupa X. Kod obične TM je bilo: iz $Q \times A^k$ u $V^k \times Q$. Moglo bi se uvesti ograničenje na broj izbora d, recimo $d \leq 2$, da se jednom objektu iz $Q \times A^k$ pridružuju najviše dva objekta iz $V^k \times Q$. Da je funkcija φ najviše dvoznačna.

Uzmimo da je početna pozicija data i da mašina radi. U slučaju TM, stvara se jedan potpuno određeni niz ili sekvenca tekućih pozicija. U slučaju NDTM, nastaju razne varijante procesa obrade, kako vrijeme protiče tako se broj varijanti povećava. Drugim riječima, imamo nekoliko sekvenci tekućih pozicija, kako vrijeme protiče tako se vrše nova grananja. Dakle, procesu računanja odgovara jedno drvo tekućih pozicija. Ove okolnosti odnosno proces računanja prikazane su na slici i to dio a) prikazuje rad TM, a dio b) prikazuje rad NDTM.



Riječ w se smatra prihvaćenom od strane NDTM ako bar jedna od nastalih varijanti prihvata w. Dakle, dovoljno je da postoji bar jedna sekvenca tekućih pozicija koja vodi do prihvatanja. Ako je prihvatanje riječi signalizovano onda nas ostale sekvence više ne zanimaju.

Definicija 2. Kažemo da NDTM sa k traka T prihvata riječ $w \in \Omega(A_t)$ ako $]\,0\,w\,\underset{\uparrow}{0}\ldots$ (na prvoj traci), $]\,\underset{\uparrow}{0}\ldots$ (na ostalim trakama) $\Rightarrow$ $]\sim 0\,a_1\,\underset{\uparrow}{0}$ (na prvoj traci), $]\sim$ (na ostalim trakama) bar za jednu od nastalih grana procesa obrade, tj. bar za jednu od nastalih varijanti. Skup svih takvih riječi čini jezik $L(T)$ koji mašina prihvata.

Ako nekoliko sekvenci signalizuje prihvatanje jedne riječi onda ćemo mi o prihvatanju te riječi saznati po najkraćoj od tih sekvenci. Zato se kod definisanja složenosti uzima u obzir dužina (odgovarajući broj taktova) te najkraće sekvence. Ako nema sekvence koja bi signalizovala prihvatanje onda složenost ostaje nedefinisana. Zato se u nedeterminističkom slučaju kod definisanja složenosti računanja uzimaju u obzir samo ulazi veličine n koji će biti prihvaćeni, a ne svi mogući ulazi veličine n (kako važi za deterministički slučaj).

Definicija 3. Kažemo da NDTM sa k traka T ima vremensku složenost $T(n)$ ako za svaku prihvaćenu riječ dužine n postoji bar jedna sekvenca tekućih pozicija čija je dužina $\leq T(n)$ a koja vodi do prihvatanja te riječi. Ako za neko n nijedna riječ nije prihvaćena onda za to n stavljamo da je $T(n) = 1$.

Izložimo formulom definiciju složenosti u D (determinističkom) odnosno u ND (nedeterminističkom) slučaju. Neka je T TM (neka je T DTM). Označimo sa niz(w) niz konfiguracija mašine u slučaju da je u početnoj poziciji bila upisana riječ w na kanonski način ili na uobičajen način, taj niz je jednoznačno određen. Označimo sa $\ell(\text{niz}(w)) \in N \cup \{+\infty\}$ dužinu niza, tj. broj taktova. Funkcija složenosti $T = T(n)$, gdje $n \in N$, $T(n) \in N \cup \{+\infty\}$, definiše se relacijom:

$$T(n) = \max_{w \in \Omega(A_t),\ |w|=n}\ \ell(\text{niz}(w)).$$

Neka je sada T NDTM. Veličina niz(w) nije više jedna jedina, budući da tokom procesa računanja dolazi do grananja. Ako određeni niz niz(w) ima svojstvo da signalizuje prihvatanje riječi w onda to zapisujemo pomoću $\Upsilon(\text{niz}(w))$. Funkcija složenosti definiše se relacijom:

$$T(n) = \max_{w \in L(T),\ |w|=n}\ \min_{\text{niz}(w),\ \Upsilon(\text{niz}(w))}\ \ell(\text{niz}(w))$$

i $T(n) = 1$ ako u jeziku $L(T)$ pridruženom mašini T nema nijedne riječi dužine n, sada je $T(n) \in N$.

Primjer. Formulišimo jedan zadatak. Zadatak o podjeli, engl. partition problem. Dato je nekoliko prirodnih brojeva. Odgovoriti na sljedeće pitanje: da li dati brojevi mogu da se podijele u dvije grupe (svaki broj je dopao u jednu od grupa) tako da zbir svih članova prve grupe bude jednak zbiru svih članova druge grupe. Recimo, u slučaju primjerka zadatka: dati brojevi su 4, 14 i 10, odgovor na postavljeno pitanje je očito "da". Ne zna se da li zadatak o podjeli $\in \mathcal{P}$ ili $\notin \mathcal{P}$. Sastaviti program za NDTM za rješavanje zadatka o podjeli. Neka su ulazni podaci prikazani na prvoj traci na sljedeći način:

$$]\,2\ \underbrace{1\ldots1}_{x_1 \text{ puta}}\ 0\ \underbrace{1\ldots1}_{x_2 \text{ puta}}\ 0\ \ldots\ 0\ \underbrace{1\ldots1}_{x_n \text{ puta}}\ 2\underset{\uparrow}{0}\ldots,$$

što znači da je dato n prirodnih brojeva i da su ti prirodni brojevi upravo $x_1, x_2, \ldots, x_n$. Završna pozicija na prvoj traci treba da bude oblika $]\sim 0\,1\underset{\uparrow}{0}$ ako i samo ako je odgovor na postavljeno pitanje, za dati primjerak, jednak "da".

Ideja rješenja ili ideja programa za NDTM. Koristi se mašina sa tri trake. Vidimo da na prvoj traci imamo n nizova jedinica. Jedan po jedan niz jedinica redom biva nedeterministički (slučajno) upisan na drugu ili treću traku, tj. biva dopisan već postojećim jedinicama druge trake ili biva dopisan već postojećim jedinicama treće trake. Kada se svih n nizova ovako prepiše onda se još samo uporedi broj jedinica na drugoj traci sa onim na trećoj traci. Ako se brojevi poklapaju onda se na prvu traku upiše $0\,1\underset{\uparrow}{0}$ i zaustavi se. Ako se ne poklapaju, ne mora ništa da se radi. U nastavku – skica rješenja:

1: Pređi na korak 2 ili pređi na korak 3

2: Jedinice koje su lijevo od tekućeg radnog polja prve trake (sve do znaka koji je $\neq 1$) prebaci na drugu traku, tj. dopiši ih desno od već postojećih tamo, pređi na 4

3: ... prebaci na treću traku ...

4: Ako je u tekućem radnom polju prve trake upisan znak 0 onda pređi na korak 1, a inače pređi na korak 5

5: Ako se količine znakova 1 na drugoj traci i na trećoj traci poklapaju onda pređi na 6, a inače se zaustavi

6: Odštampaj na prvoj traci 010 i zaustavi se.

Vidimo da se mogućnost nedeterminizma koristi u naredbi 1, "ili". Iz očitih tehničkih razloga izostavljamo dalje detaljisanje izloženog rješenja (sastavljanje tablice mašine). Završen primjer.

Naglasimo da svojstvo nedeterminizma samo liči na svojstvo paralelizma. Nedeterministički uređaj za računanje: tokom izvršavanja programa stvara se više grana procesa računanja, onda te grane rade paralelno, kako vrijeme protiče tako tih grana ima sve više i više uopšte uzev, nema gornjeg ograničenja na broj grana. Paralelni računar: ima recimo 2^{16} procesora koji rade paralelno i usaglašeno, broj procesora je fiksiran.

Oblast djelovanja (klasa izračunljivih funkcija) za dva modela TM i NDTM je jedna te ista.

Vremenska složenost programa za NDTM sa jednom trakom i vremenska složenost odgovarajućeg ili ekvivalentnog programa za NDTM sa k traka su dvije funkcije koje su međusobno polinomski povezane.

Na kraju, možda je model RAM jednostavniji za razumijevanje od TM. Polazeći od nekog (determinističkog) modela za računanje može se definisati njemu odgovarajući nedeterministički model. Tako može da bude definisana nedeterministička mašina RAM.

NDRAM nastaje kada se postojećem repertoaru naredbi mašine RAM doda naredba CHOICE čiji opšti oblik glasi CHOICE($L_1, L_2, \ldots, L_k$). Ovdje su $L_1, L_2, \ldots, L_k$ labele (obilježja naredbi). Ova naredba zahtijeva da se slučajno (nedeterministički) izabere jedna od k naredbi koje imaju obilježja redom $L_1, L_2, \ldots, L_k$. Drugim riječima, izvršavanjem ove naredbe stvara se k grana procesa računanja, grane se odsad izvršavaju istovremeno, nezavisno jedna od druge. Tako da proces računanja na mašini NDRAM može da bude prikazan pomoću jednog drveta tekućih pozicija mašine. Efekat nedeterminizma bi bio u potpunosti postignut i kada bi grananje u pojedinom vrhu tog zamišljenog drveta bilo ograničeno. Na primjer, da broj izbora na pojedinom mjestu bude sveden na $d = 2$, tj. da naredba CHOICE ima opšti oblik CHOICE(L_1, L_2).

7.2. TRI KLASE JEZIKA

Mi sada govorimo samo o zadacima koji su algoritamski rješivi, tj. o zadacima za koje postoji odgovarajući program. Znamo da se zadatak u opštem slučaju odnosi na računanje vrijednosti funkcije. Ulazni podatak programa je argument funkcije ili argumenti funkcije (kada se radi o funkciji od više promjenljivih), a rezultat treba da bude – vrijednost funkcije. Ako postoje samo dvije moguće vrijednosti funkcije (0 i 1 ili svejedno "da" i "ne" ili svejedno pripada i ne pripada) onda je u pitanju tzv. da–ne zadatak ili zadatak o prihvatanju (prepoznavanju, raspoznavanju) jezika. U tom slučaju, ulazni podatak programa jeste riječ, a program saopšti – pripada li ta riječ jeziku ili ne pripada. Sada će biti definisane tri važne klase. Mogu se definisati kao klase zadataka ili kao klase jezika. Mi ćemo ih definisati kao klase jezika, jer je tako uobičajeno i jer je tako jednostavnije, sva svojstva teorije se lijepo vide. Ukratko o tri

važne klase. Klasa $\mathcal{P}$, od riječi polinomski, tu pripadaju jezici koji su laki, koji su laki u pravom smislu riječi, u deterministočkom smislu. Klasa $\mathcal{NP}$, od riječi nedeterministički polinomski, tu pripadaju jezici koji su "laki", koji su laki u nedeterminističkom smislu. Klasa NPC, od riječi – nedeterministički polinomski kompletan, ovoj kritičnoj klasi jezika pripadaju jezici iz klase $\mathcal{NP}$ koji se nalaze na vrhu same te klase $\mathcal{NP}$.

Definicija 4. Označimo sa $\mathcal{P}$ skup svih jezika koji mogu da budu prihvaćeni od neke TM čija je složenost polinomska. Dakle, $\mathcal{P} = \{L \mid \exists$ TM T i $\exists$ polinom $p = p(n)$ takvi da je $L = L(T)$ i $T(n) \leq p(n)\}$. Ovdje $L(T)$ označava jezik koji mašina T prihvata. Ovdje je $T(n)$ vremenska složenost mašine T.

Kao što smo radili, ako $w \in L$ onda $]\,0\,w\,\underset{\uparrow}{0}\ldots \Rightarrow\,]\sim 0\,a_1\,\underset{\uparrow}{0}$, ako $w \notin L$ onda $]\,0\,w\,\underset{\uparrow}{0}\ldots \Rightarrow\,]\sim 0\,\underset{\uparrow}{0}$, uzeli smo konvenciju da riječ a_1 znači "da" a da riječ $\Box$ znači "ne"; $L \subset \Omega(A_t)$.

Definicija 5. Označimo sa $\mathcal{NP}$ skup svih jezika koji mogu da budu prihvaćeni od neke NDTM čija je složenost polinomska.

Oznake: ako je $A = \{1\}$, $B = \{1, 2\}$ onda pišemo $A \subset B$, $A \subset_{\neq} B$, $B \subset B$. Jasno je da važi $\mathcal{P} \subset \mathcal{NP}$. Zaista, NDTM nije obavezna da koristi mogućnost grananja, a dodatno se kod NDTM prilikom računanja složenosti uzimaju u obzir samo prihvaćene riječi dužine n (a kod obične TM – sve riječi dužine n). S druge strane, nije nađen primjer jezika koji bi pripadao $\mathcal{NP}$ a ne bi pripadao $\mathcal{P}$. Dakle, otvoreno pitanje: da li je $\mathcal{P}$ pravi podskup od $\mathcal{NP}$ ili se pak suprotno $\mathcal{P}$ i $\mathcal{NP}$ poklapaju. Formulom: da li je $\mathcal{P} \subset_{\neq} \mathcal{NP}$ ili je $\mathcal{P} = \mathcal{NP}$. Ili: da li je $\mathcal{P} \neq \mathcal{NP}$ ili je $\mathcal{P} = \mathcal{NP}$.

Dakle, NP–hipoteza ili hipoteza o NP–kompletnosti glasi: $\mathcal{P}$ je pravi podskup od $\mathcal{NP}$. Ovo je otvoreno pitanje, tj. ova hipoteza predstavlja najvažniji otvoreni problem (neriješeni problem) teorijskog programiranja. Među stručnjacima preovladava uvjerenje da je hipoteza istinita.

Prilikom raznih izučavanja vezanih za ovo otvoreno pitanje, na prirodan način se pojavila jedna važna klasa jezika, tzv. klasa NP–kompletnih jezika, skraćeno klasa NPC. Naime, ako neko usmjeri svoja ispitivanja ka tome da dokaže da je $\mathcal{P}$ pravi podskup od $\mathcal{NP}$ onda se on prirodno prvo zapita: koji su zadaci najteži u $\mathcal{NP}$. Pa će za izdvojene zadatke nastojati da dokaže da su van $\mathcal{P}$, da je njihova (deterministička) složenost iznad polinomske.

Definicija 6. Za jezik $L_0 \in \mathcal{NP}$ kažemo da je NP–kompletan ako je ispunjen sljedeći uslov. Neka imamo deterministički, tj. običan algoritam vremenske složenosti $T(n) \geq n$ za prihvatanje jezika L_0. Tada za svaki jezik $L \in \mathcal{NP}$ možemo da nađemo deterministički, tj. običan algoritam za njegovo prihvatanje vremenske složenosti do $T(p_L(n))$, gdje je p_L polinom koji zavisi od L.

Riječima, složenost jezika L manja je ili jednaka od na izvjestan polinomski način uvećane složenosti jezika L_0. Vidimo sljedeće: ako je $T(n)$ polinom onda je i $T(p_L(n))$ polinom. Prema tome, ako za L_0 postoji polinomski algoritam onda za svako $L \in \mathcal{NP}$ postoji polinomski algoritam. Drugim riječima, ako je bar jedan NP–kompletan zadatak lak (može da bude riješen za polinomsko vrijeme, pripada $\mathcal{P}$) onda su i svi zadaci iz $\mathcal{NP}$ takođe laki (pripadaju $\mathcal{P}$). Hipoteza o NP–kompletnosti prelama se na jednom bilo kom zadatku iz skupa NPC. Dalje, ako je bar jedan zadatak iz $\mathcal{NP}$ težak (ne pripada $\mathcal{P}$) onda su i svi NP–kompletni zadaci teški (ne pripadaju $\mathcal{P}$).

Zanimljivo je na jednom crtežu prikazati ove tri klase, v. crtež. Ako važi hipoteza o NP–kompletnosti, tj. ako je $\mathcal{P} \subset_{\neq} \mathcal{NP}$ onda je istinito ono što pokazuje slika a). A ako je $\mathcal{P} = \mathcal{NP}$ onda je istinito ono što pokazuje slika b).

Prilikom proučavanja najtežih (ili pretpostavljeno najtežih) zadataka iz $\mathcal{NP}$, različiti autori

su na različite načine definisali predmet proučavanja. Tako je prisutna i sljedeća, druga definicija za NPC.

Definicija 7. Za jezik $L_0 \in \mathcal{NP}$ kaže se da je NP–kompletan ako je ispunjen sljedeći uslov. Za svaki jezik $L \in \mathcal{NP}$ postoji deterministički (običan) algoritam, tj. TM čija je složenost polinomska a koja vrši sljedeću obradu: svaku riječ w u azbuci jezika L ona pretvara u riječ w_0 u azbuci jezika L_0 tako da važi $w \in L$ ako i samo ako $w_0 \in L_0$. Kaže se da se pomoću ove mašine jezik L transformiše ili svodi na jezik L_0.

Jasno je da je w ulazni podatak ove mašine i da je w_0 njen izlazni podatak, tj. rezultat njenog rada. Vidimo da ova mašina M za polinomsko vrijeme svodi pitanje koje se tiče jezika L na pitanje koje se tiče jezika L_0.

Prema tome, za prepoznavanje jezika L predlaže se sljedeći algoritam koji se sastoji iz dva koraka. Korak A: w se pomoću M prevede na w_0. Korak B: na w_0 se primijeni neki algoritam za L_0. Njegov odgovor će biti preuzet, tj. predstavljaće i naš odgovor.

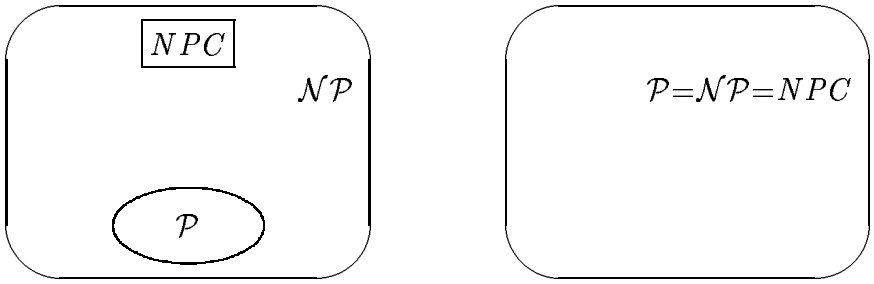
Da li $w_0 \in L_0$: algoritam PROG na w_0. Da li $w \in L$: algoritam M na riječ w, a onda još na izlazni podatak ovog algoritma primijeniti algoritam PROG. Zapaziti da je algoritam M efikasan (ima polinomsku složenost), tj. da izvršavanje koraka A troši malo vremena.

Dakle, bilo koji zadatak $L \in \mathcal{NP}$ je eventualno samo malo teži od $L_0 \in NPC$. Niko ne brani da se L riješi pomoću nekog drugog algoritma, bez svođenja na L_0, ne dovodeći ga u bilo kakvu vezu sa L_0, moguće je da taj drugi algoritam bude efikasniji od algoritma u kome se primjenjuje svođenje.

Vidimo sljedeće: ako je algoritam PROG efikasan, tj. ako zadatak L_0 ima polinomsku složenost (pripada $\mathcal{P}$) onda i zadatak L ima polinomsku složenost (pripada $\mathcal{P}$), budući da efikasan algoritam M + PROG služi za njega.

Može se dokazati da je $NPC_7 \subset NPC_6$. U dokazu, treba da pođemo od jednog jezika L_0 koji zadovoljava uslove definicije 7 i da pokažemo da taj jezik zadovoljava i uslove definicije 6.

Zanimljivo je da nije poznato da li važi $NPC_7 \subset_{\neq} NPC_6$ ili pak važi $NPC_7 = NPC_6$. Mi usvajamo definiciju 6 kao definiciju klase NPC. Prilikom dokazivanja da neki zadatak pripada klasi NPC imamo pravo da se oslonimo na definiciju 7 zato što $L_0 \in NPC_7 \Rightarrow L_0 \in NPC_6$.



a) b)

Jedna slika je istinita a jedna je lažna

7.3. JEZICI I ZADACI

Biće navedeno nekoliko primjera zadataka (sastaviti program za računanje funkcije) i specijalno nekoliko primjera jezika (sastaviti program za prepoznavanje jezika). Biće uvedeno nekoliko jednostavnih pravila za zapisivanje ili kodiranje ulaznih podataka programa, odnosno za prikazivanje primjerka zadatka.

Pođimo od drugog, od tih pravila. Ta pravila definišu uslove koje treba da ispunjava prikazivanje, tj. koje treba da ispunjava šema za kodiranje. Ako šema ispunjava uslove onda se za šemu kaže da je razumna.

Zadaci koji se rješavaju odnose se obično na brojeve, grafove, drveta, Bulove izraze i slično. Šema za kodiranje ne treba da vještački preuveličava obim ulaza n. Zato se brojevi ne prikazuju u unarnom obliku nego se prikazuju u binarnom ili dekadnom obliku. Niti da vještački umanjuje. Šema za kodiranje ne smije da pomaže programu. Recimo, ako se ulazni podaci sastoje od nekoliko brojeva onda šema za kodiranje ne smije da traži da se ti brojevi unose uređeni po veličini.

Ako šema za kodiranje nije razumna onda ona daje nepravilnu sliku o veličini ulaznih podataka i o složenosti algoritma (programa) koji se predlaže za rješavanje postavljenog zadatka. Naime, znamo da složenost predstavlja jednu funkciju ili funkcionalnu zavisnost potrebnog broja koraka $T(n)$, izraženog u zavisnosti od obima ulaza n. Vidimo da način definisanja obima ulaza iz osnova utiče na tu zavisnost i da čak može da utiče i na oblik zavisnosti (da li je zavisnost polinomska ili nije).

Nasuprot dosad rečenom, detalji u pravilima o kodiranju koji ne utiču puno na ukupnu dužinu koda – nisu značajni.

Za kodiranje se kaže da je razumno ako su ispunjena sljedeća pravila. Ova pravila definišu standard za prikazivanje argumenta zadatka:

1 Prirodni ili cijeli brojevi predstavljaju se u dekadnom ili binarnom brojnom sistemu, tj. u bilo kom brojnom sistemu čija je osnova veća od jedan. Tako da broj N ima otprilike $\log N$ ili $\log_2 N$ cifara.

2 Vrhovi grafa koji ima n vrhova označavaju se brojevima od 1 do n, a ivica grafa koja vodi od vrha i_1 ka vrhu i_2 označava se kao par (i_1, i_2).

3 Ako Bulov izraz ima n promjenljivih onda se promjenljive označavaju brojevima od 1 do n. Bulove operacije "i", "ili" i "ne" označavaju se redom znacima recimo $*$, $+$ i $\neg$. U zapisu Bulovog izraza učestvuju i zagrade.

Slijede primjeri zadataka (postavke zadataka). Dati su i primjeri šema za kodiranje (šema koje su pravilne).

1. Zadatak o podjeli, v. ranije u 7.1. Standardno kodiranje u slučaju ranije navedenog primjerka može da bude X4␣14␣10X. Obim ulaza $= |w| = n = 9$.

2. Zadatak o ispunljivosti ili zadatak SAT, od engleske riječi satisfiability. Dat je Bulov izraz u konjunktivnoj normalnoj formi (KNF). Neka izraz E zavisi od n promjenljivih $u_1, u_2, \ldots, u_n$. Da li je izraz E bar jednom istinit (ispunjen), tj. da li je bar za jednu kombinaciju vrijednosti promjenljivih $u_1, u_2, \ldots, u_n$ taj izraz $E = 1$ (= TRUE). Slijedi postavka nešto potpunije opisana. Nazovimo disjunkcijom izraz oblika $\vee_j v_j$ (ili $\sum_j v_j$), gdje je v_j jednako u_i za neko i ili je v_j jednako $\overline{u_i}$ za neko i. Bulovim izrazom $E = E(u_1, \ldots, u_n)$ nazivamo izraz oblika $E = \&_k\, d_k$ (ili $E = \prod_k d_k$), gdje je d_k disjunkcija za svako k. Zadatak SAT: da li se mogu izabrati vrijednosti za promjenljive $u_1, \ldots, u_n$ tako da za izabrane vrijednosti bude $E(u_1, \ldots, u_n) = 1$.

Razmotrimo primjerak Bulovog izraza od tri promjenljive $E = (u_1 \vee u_2 \vee \overline{u_3}) \,\&\, (u_1 \vee u_2) \,\&\, (u_1 \vee \overline{u_2})$. Kodira se kao $(1{+}2{+}\neg(3)){*}(1{+}2){*}(1{+}\neg(2))$. Dužina ovog koda ili obim ulaza iznosi kada se izbroji $|w| = n = 25$.

Mi kažemo programeru da zadatak SAT glasi: sastaviti program koji učitava riječ w (riječ w definiše jednu KNF) i onda štampa odgovor "postoji kombinacija" ili "ne postoji kombinacija". Bilo bi dobro da sastavljeni program ima polinomsku složenost.

Vraćajući se primjerku od maločas, za taj primjerak odgovor očito glasi "kombinacija pos-

toji" (izraz je ispunljiv), tj. riječ w pripada jeziku ("da"). Zaista, dovoljno je izabrati $u_1 = 1$. Kada je $u_1 = 1$, bez obzira na vrijednosti u_2 i u_3 (možemo reći $u_2 = 0$ i $u_3 = 0$), tada važi $E = E(u_1, u_2, u_3) = 1$.

3. Zadatak o kliki. Dati su podaci o neusmjerenom grafu G i dat je prirodan broj k. Sastaviti program koji odgovara na sljedeće pitanje: da li postoji potpun podgtaf grafa G čija je veličina jednaka k (broj vrhova). Znamo da klika ili potpun graf znači graf u kome su bilo koja dva vrha povezani ivicom.

Razmotrimo primjerak kada je $k = 3$ i kada je graf G v. sliku. Navedeni ulazni podaci mogu da se kodiraju kao 3//1,2/1,4/2,3/2,4/3,4/3,5/4,5.

Što se tiče slike, klika veličine 3 očito postoji, npr. $\{v_1, v_2, v_4\}$, odgovor je "da".

Kodiranje od maločas izvedeno je na osnovu sljedećeg pravila ili šeme koja se može uvesti za zadatak o kliki: $k//i_1, j_1/i_2, j_2/ \ldots /i_m, j_m$, što znači da se pitanje odnosi na potpun podgraf veličine k u grafu koji ima m ivica (i_1, j_1), ..., (i_m, j_m).

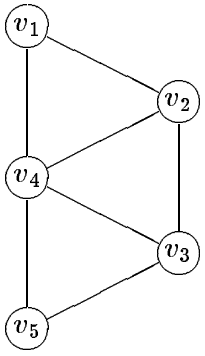
4. Zadatak o trgovačkom putniku. Dat je potpun neusmjeren graf, vrhove grafa smatramo gradovima. Svakoj ivici grafa pridružen je prirodan broj koji predstavlja dužinu ili cijenu ivice, predstavlja rastojanje između dva grada koji su krajevi ivice (predstavlja dužinu puta između ta dva grada). Trgovački putnik treba tačno jednom da prođe kroz svaki grad i da se na kraju nađe u gradu iz koga je krenuo. Odrediti optimalnu maršrutu, tj. onu za koju je ukupna cijena najmanja moguća.

Sljedeća slika služi da ilustruje zadatak o trgovačkom putniku. Na slici je prikazan jedan primjerak zadatka. Da li je zadatak o trgovačkom putniku lak ili težak?

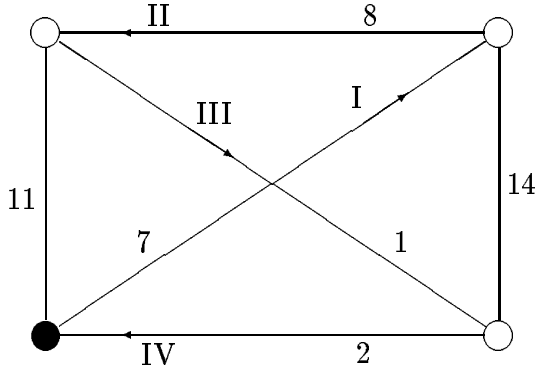
Svaki od zadataka od 1 do 4 je NP–kompletan.

Zadatku 4 odgovara nešto jednostavniji zadatak oblika da–ne, kako slijedi. Dati su podaci o težinskom grafu kao maločas i još je dat prirodan broj B. Postoji li maršruta za trgovačkog putnika čija je ukupna cijena $\leq B$?

Zadatak o optimizaciji ponekad može da se svede na odgovarajući (jednostavniji) da–ne zadatak. Umjesto "da–ne zadatak" može se kazati problem odlučivanja.



Klika



Trgovački putnik

7.4. DOKAZ DA JE ZADATAK SAT NP–KOMPLETAN

U ovom naslovu biće dokazana teorema koja jednostavno tvrdi da zadatak SAT pripada klasi NPC. Ovu teoremu dokazao je S. A. Cook 1971. godine, to je tzv. Kukova teorema. Sa tom teoremom i počinje razvoj teorije NP–kompletnih zadataka. Dakle, gledano po vremenskoj osi, zadatak SAT je prvi zadatak za koji je dokazano da je NP–kompletan. U tom smislu, dokaz se

izvodi oslanjanjem na definiciju klase NPC. Postavku zadatka SAT v. u prethodnom naslovu 7.3. Da li je ikad $E = 1$, gdje je $E = E(u_1, \ldots, u_n) = (u_{11} \vee u_{12} \vee \ldots \vee u_{1n_1}) \,\&\, \ldots \,\&\, (u_{k1} \vee u_{k2} \vee \ldots \vee u_{kn_k})$, s tim da su pojedini u_{ij} jednaki u_1 ili $\overline{u_1}$ ili itd. ili u_n ili $\overline{u_n}$.

Teorema. Zadatak SAT pripada klasi NPC.

Dokaz. Dokaz ima dva dijela. U prvom dijelu biće dokazano da zadatak SAT pripada klasi $\mathcal{NP}$ time što će biti sastavljen nedeterministički polinomski algoritam za njegovo rješavanje. U drugom dijelu biće pokazano, grubo govoreći, da u klasi $\mathcal{NP}$ nema težeg zadatka od SAT. Govoreći preciznije, da se svaki član L skupa $\mathcal{NP}$ može za polinomsko vrijeme redukovati na zadatak SAT. Dakle, ovdje će biti upotrebljena definicija 7.

Prvi dio dokaza. Vidjećemo da NDTM stvara, što je prije moguće, 2^n grana procesa računanja, svaka grana odgovara jednom potencjalnom rješenju, tj. odgovara jednoj dodjeli vrijednosti n–torki promjenljivih $(u_1, u_2, \ldots, u_n)$, moguće vrijednosti koje se dodjeljuju su 0 i 1 (netačno i tačno). Zatim se za svako od 2^n potencijalnih rješenja ispita da li ono stvarno pokazuje da je bar jednom $E(u_1, u_2, \ldots, u_n) = 1$, da li pokazuje da je odgovor "da". Koliko vremena treba da se izračuna vrijednost Bulovog izraza E kada su dodijeljene vrijednosti promjenljivima tog izraza? Označimo dužinu zapisa izraza E na Tjuringovoj traci kao $|E| = N$. Lako se vidi da postoji algoritam čija je složenost $O(N^2)$.

Tako da se prvi dio dokaza sastoji od sljedećih glavnih elemenata. Opišimo odgovarajuću NDTM. Neka smo se opredijelili za određenu šemu za kodiranje i neka su ulazni podaci (prikaz tj. kod Bulovog izraza) zapisani na početku prve trake. Program ove NDTM predviđa sljedeće. Za svako i od 1 do n ponovi: upiši na prvo slobodno mjesto na drugoj traci 0 ili upiši na prvo slobodno mjesto na drugoj traci 1, "ili" – nedeterminizam. Sada očito ima 2^n grana procesa računanja. Zatim izračunaj vrijednost izraza koji je prikazan na prvoj traci u slučaju da brojevi upisani na drugoj traci predstavljaju redom vrijednosti za $u_1, u_2, \ldots, u_n$. Ako je izračunata vrijednost $E = 0$ onda se zaustavi. A ako je $E = 1$ onda treba signalisati pozitivan odgovor, tj. odštampaj na prvoj traci 010 i zaustavi se. Jasno je da se tokom računanja vrijednosti Bulovog izraza mogu koristiti još neke trake.

Treba pokazati da je izloženi nedeterministički algoritam – polinomske složenosti. Neka je $|E| = |w|$ dužina zapisa na prvoj traci. Neka je $f(|w|)$ broj taktova u slučaju da je došlo do prihvatanja riječi w upisane na prvoj traci, misli se – broj taktova izvršenih u onoj sekvenci koja je dovela do prihvatanja riječi, odnosno u najkraćoj od takvih sekvenci. Detaljnija analiza bi pokazala da je $f(|w|) \leq p(|w|)$ za neki polinom $p = p(x)$.

Drugi dio dokaza. Treba dokazati da svaki jezik $L \in \mathcal{NP}$ može biti polinomski redukovan na zadatak SAT. Neka je M NDTM čija je složenost polinomska (u nedeterminističkom smislu) koja raspoznaje L i neka je w ulaz za M. Polazeći od M i w, mi ćemo konstruisati Bulov izraz w_0 takav da M prihvata w ako i samo ako je w_0 ispunljiv. Konstrukciju Bulovog izraza w_0 (po datim M i w) obavlja naravno jedna detetministička TM u oznaci recimo M_1 čija je složenost polinomska (čija je deterministička složenost polinomska). Drugi dio dokaza teoreme čini glavninu dokaza teoreme i veoma je obiman, pa se zato izostavlja.

Dokaz je završen.

Dopuna. Može se pokazati da je i zadatak 3–SAT NP–kompletan. Misli se na izraz E koji je konjunkcija izvjesnih objekata, ti objekti su disjunkcije koje sadrže najviše tri člana, a članovi su sa svoje strane promjenljive ili negacije promjenljivih. Drugim riječima $n_1 \leq 3$, ..., $n_k \leq 3$. Za E se naravno pita da li je ispunljiv. Recimo, može da bude $E = (u_1 \vee \neg u_2 \vee u_7) \,\&\, \ldots$.

Međutim, može se pokazati da je zadatak 2–SAT lak, tj. da pripada klasi $\mathcal{P}$.

7.5. DOKAZ DA JE ZADATAK O KLIKI NP–KOMPLETAN

Zadatak o kliki bio je ranije formulisan u 7.3. Ponovimo ukratko: da li u datom neusmjerenom grafu postoji potpun podgraf date veličine. Biće dokazana teorema koja jednostavno tvrdi da zadatak o kliki pripada klasi NPC. Na ovom primjeru vidjećemo običan način da se dokaže da neki zadatak L_2 (u ovom slučaju, zadatak o kliki) pripada klasi NPC. Naime, polazi se od nekog zadatka L_1 za koji se odranije zna da pripada klasi NPC. Zatim se pokaže da se L_1 svodi na L_2, svodi za polinomsko vrijeme u determinističkom smislu. Ulogu zadatka L_1 u našem slučaju imaće zadatak SAT. Slijedi da je zadatak L_2 NP–kompletan. Naravno ako još L_2 pripada klasi $\mathcal{NP}$. Mi ovdje koristimo definiciju 7 i svojstvo tranzitivnosti relacije polinomskog svođenja.

Ponovimo, šablon dokaza je sljedeći: $L_1 \in$ NPC & L_1 se polinomski svodi na L_2 & $L_2 \in \mathcal{NP}$ $\Rightarrow$ $L_2 \in$ NPC.

Teorema. Zadatak o kliki je NP–kompletan.

Dokaz teoreme. Dokaz teoreme sastoji se iz dva dijela. Prvi dio dokaza teoreme. Da je zadatak o kliki u klasi $\mathcal{NP}$. Zaista, lako se konstruiše polinomski algoritam za NDTM, otprilike kako slijedi. Prvo se (koristeći nedeterminizam) kreiraju sve moguće n–torke čiji su članovi 0 i 1, takvih n–torki ima 2^n, gdje je n broj vrhova grafa. Imamo 2^n grana procesa računanja. Svaka n–torka definiše jedan podgraf: ako je na i–tom mjestu 0 onda to znači da i–ti vrh ne pripada podgrafu. Za podgraf se ispita da li je potpun (da li ima sve moguće ivice) i da li ima tačno k vrhova, ovdje je k očito predviđena veličina klike (potpunog podgrafa). Ako podgraf isounjava oboje onda saopštiti "da". Drugi dio dokaza teoreme. Pokazaćemo da se zadatak SAT može za polinomsko vrijeme (na običnoj Tjuringovoj mašini) svesti na razmatrani zadatak o kliki. Poslije toga je dokaz završen, uzimajući u obzir da je zadatak SAT NP–kompletan. Ostaje da se opiše kako se vrši spomenuto svođenje. U nastavku slova n i k uzimaju drugo značenje.

Razmotrimo Bulov izraz E koji zavisi od n promjenljivih ($n \geq 1$). Označimo promjenljive kao $x_1, \ldots, x_n$. Uzimamo da je izraz $E = E(x_1, \ldots, x_n)$ prikazan u KNF, tj. uzimamo da je $E = S_1 \wedge S_2 \wedge \ldots \wedge S_k$ ($k \geq 1$), gdje je $S_i = x_{i1} \vee x_{i2} \vee \ldots \vee x_{in_i}$ ($n_1 \geq 1$, $n_2 \geq 1$, ..., $n_k \geq 1$). Ovdje je x_{ij} promjenljiva ili negacija promjenljive, tj. važi $x_{ij} = x_1$ ili $x_{ij} = \overline{x_1}$ ili itd. ili $x_{ij} = x_n$ ili $x_{ij} = \overline{x_n}$. Izrazu $E = E(x_1, \ldots, x_n)$ pridružujemo graf G po sljedećem propisu. Kaži koji je skup vrhova grafa i kaži koji je skup ivica grafa. Za skup vrhova grafa G uzimamo skup svih uređenih parova prirodnih brojeva oblika (i, j), gdje je $1 \leq j \leq n_i$, s tim da je $1 \leq i \leq k$. Dva vrha (i, j) i (p, q) ćemo povezati ivicom u grafu G ako je $i \neq p$ i $x_{ij} \neq \overline{x_{pq}}$.

Lako se vidi da broj vrhova grafa G ne prevazilazi dužinu zapisa izraza E (ne prevazilazi $|E|$). Odatle se lako zaključuje da je vrijeme potrebno da se ostvari opisano pridruživanje–svođenje (da se sastavi–zapiše graf G) ograničeno odozgo nekim polinomom od $|E|$. Svođenje zahtijeva samo polinomsko vrijeme.

Tvrdimo da je izraz E ispunljiv ako i samo ako u grafu G postoji potpun podgraf sa k ili više vrhova. Ostaje još samo da se ovo dokaže.

U jednom smjeru. Neka je izraz E ispunljiv. Tada postoji kombinacija vrijednosti promjenljivih $x_1, \ldots, x_n$ za koju su svi izrazi S_i istiniti ($S_1 = 1$, $S_2 = 1$, ..., $S_k = 1$). Zato za svako $i \in \{1, 2, \ldots, k\}$ postoji bar jedna vrijednost $j_i \in \{1, 2, \ldots, n_i\}$ takva da je $x_{ij_i} = 1$. Posmatrajmo skup vrhova $\{(i, j_i) |\ 1 \leq i \leq k\}$ i posmatrajmo sve ivice grafa G čija su oba kraja u tom skupu vrhova. Taj skup vrhova zajedno sa tim ivicama čini očito podgraf koji ima k vrhova. Više od toga, taj podgraf je potpun. Zaista, ako između dva njegova različita vrha (p, j_p) i

(q, j_q) ne bi postojala ivica onda bi važilo (uzimajući u obzir da je $p \neq q$) $x_{pj_p} = x_{qj_q}$, v. definiciju svođenja. Međutim, ovo posljednje je nemoguće, budući da za spomenutu kombinaciju vrijednosti promjenljivih $x_1, \ldots, x_n$ važi $x_{pj_p} = 1$ i $x_{qj_q} = 1$. Klika postoji.

$S_1 \wedge S_2 \wedge S_k = 1 \Rightarrow S_1 = 1$, $S_2 = 1$, ..., $S_k = 1$

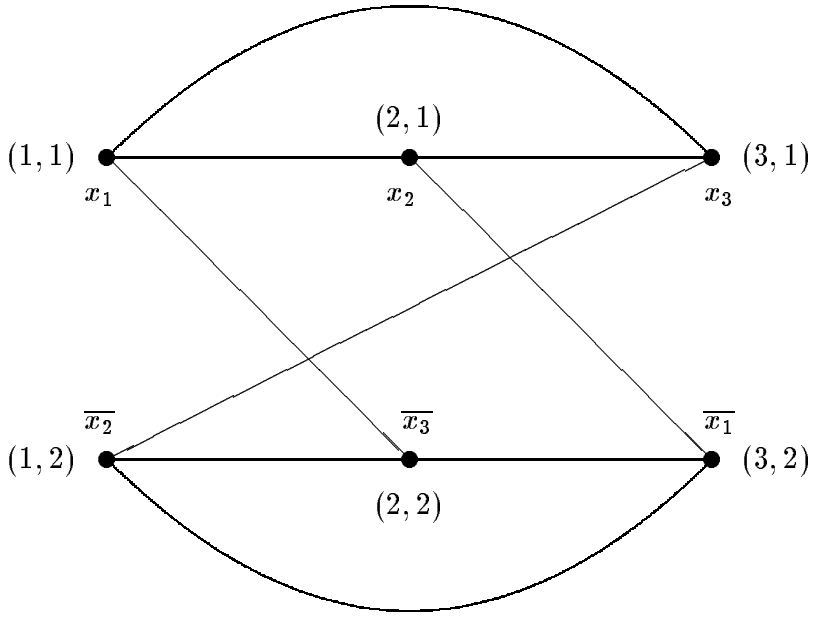
$x_{i1} \vee x_{i2} \vee \ldots \vee x_{in_i} = 1 \Rightarrow x_{i1} = 1$ ili $x_{i2} = 1$ ili ... ili $x_{in_i} = 1$

U suprotnom smjeru. Neka G ima potpun podgraf od k vrhova. Označimo vrhove kao (i_1, j_1), (i_2, j_2), ..., (i_k, j_k). Tada su sigurno sve vrijednosti $i_1, i_2, \ldots, i_k$ međusobno različite. Usputno zapažamo da potpun podgraf grafa G ovog oblika ne može imati više od k vrhova. Pored toga, među sabircima $x_{i_1j_1}, x_{i_2j_2}, \ldots, x_{i_kj_k}$ nijedan nije negacija nekog drugog. Zato mogu da budu izabrane vrijednosti promjenljivih tako da svi sabirci budu istiniti. Recimo, ako je $x_{i_1j_1} = x_4$ onda stavi $x_4 = 1$ a ako je $x_{i_1j_1} = \overline{x_4}$ onda izaberi $x_4 = 0$, slično postupi za $x_{i_2j_2}, \ldots, x_{i_kj_k}$. Svaki sabirak je u svojoj disjunkciji S_i. Prema tome, za izabranu kombinaciju vrijednosti promjenljivih važi $E = 1$. Dakle, izraz E je ispunljiv.

Mi smo pokazali: izraz je ispunljiv $\Leftrightarrow$ k–klika postoji. Teorema je dokazana.

Pogledajmo primjer. Neka Bulov izraz glasi $E = E(x_1, x_2, x_3) = (x_1 \vee \overline{x_2}) \wedge (x_2 \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_1})$. U pridruženom grafu, dva vrha neće biti povezani ivicom ako je prva komponenta uređenih parova korespondiranih jednom i drugom vrhu jedna te ista. Osim toga, neće se povezati x_1 sa $\overline{x_1}$ i slično. Na slici je prikazan pridruženi graf G.

Sada je $E = S_1 \wedge S_2 \wedge S_3$. Izraz je ispunljiv $\Leftrightarrow$ u grafu postoji 3–klika. Vidimo da je izraz ispunljiv. Recimo, pogodna kombinacija je $x_1 = 1$, $x_2 = 1$, $x_3 = 1$. Vidimo da u grafu postoji potpun podgraf sa k vrhova ($k = 3$). Recimo, takav je podgraf čiji su vrhovi $(1,1)$, $(2,1)$ i $(3,1)$.



A decision problem c is said to be NP–complete if: 1. c is in NP, and 2. Every problem in NP is reducible to c in polynomial time.

The easiest way to prove that some new problem is NP–complete is first to prove that it is in NP, and then to reduce some known NP–complete problem to it.

7.6. NEKI ZADACI ZA KOJE JE DOKAZANO DA SU TEŠKO RJEŠIVI

Kao što je rađeno, za jezik L (za problem odlučivanja L) kaže se da je teško rješiv ako $L \notin \mathcal{P}$. Izraze "teško rješiv" i "težak" i engl. intractable upotrebljavamo kao sinonime. Biće navedeno nekoliko primjera zadataka za koje je dokazano da su van skupa $\mathcal{NP}$. Dakle, svaki od tih zadataka je težak u nedeterminističkom smislu. Samim tim je (tim prije) težak i u determinističkom smislu riječi (težak u pravom smislu riječi). Drugim riječima, ne postoji polinomski algoritam ni deterministički, ni nedeterministički ni za koji od tih zadataka. Uostalom, budući da nije riješena–raspravljena glavna hipoteza (da li je $\mathcal{P} \subset_{\neq} \mathcal{NP}$, da li je $\mathcal{P}$ pravi podskup od $\mathcal{NP}$), to nije jasno ni da li postoji razlika između "težak u deterministčkom smislu" i "težak u nedeterminističkom smislu" (da li postoji razlika između $L \notin \mathcal{P}$ i $L \notin \mathcal{NP}$).

U ovom naslovu biće uveden pojam regularnog izraza i biće navedena dva zadatka za koje je dokazano da su teško rješivi. Predstavljamo ukratko i bez dokaza.

Neka je A fiksirana azbuka. Neka su L_1 i L_2 jezici u toj azbuci, tj. neka je $L_1 \subset \Omega(A)$ i $L_2 \subset \Omega(A)$. Njihova konkatenacija označava se kao L_1L_2 a definiše se relacijom $L_1L_2 = \{xy |\ x \in L_1,\ y \in L_2\}$, kada se poslije riječi x dopiše riječ y onda se tako dobije riječ xy. Definišimo i stepene nekog jezika L. Neka je $L^n = L^{n-1}L$ za $n \geq 2$ i neka je $L^0 = \{\Box\}$, gdje $\Box$ označava tzv. praznu riječ (riječ dužune nula slova). Uvedimo i pojam iteracije jezika L ili Klinijevog zatvorenja jezika L u oznaci L^*, neka je $L^* = \{\Box\} \cup L \cup L^2 \cup \ldots = \bigcup_{n=0}^{\infty} L^n$.

Definicija. Neka je A azbuka. Regularan izraz R u azbuci A i skup riječi ili jezik L koji je označen tim regularnim izrazom (skup riječi koji je pridružen tom regularnom izrazu) definišu se rekurzivno kako slijedi. 1. $\emptyset$ je regularan izraz, označava prazan skup riječi. 2. $\Box$ je regularan izraz, označava skup riječi $\{\Box\}$. 3. Za svako $a \in A$, a je regularan izraz a označava skup $\{a\}$. 4. Neka su p i q regularni izrazi i označimo sa P i Q redom njima pridružene (regularne) skupove riječi. Tada su i $(p+q)$, (pq) i (p^*) regularni izrazi i to oni redom označavaju skupove riječi $P \cup Q$, PQ i P^*. Za jezik koji je definisan (opisan) nekim regularnim izrazom kaže se da je regularan jezik.

Prilikom pisanja regularnih izraza mi možemo da izostavimo veliki broj zagrada ako uzmemo da operacija a^* ima veći prioritet od konkatenacije, a konkatenacija da ima veći prioritet od $+$. Na primjer, $((0(1^*)) + 0)$ može da bude zapisano kao $01^* + 0$.

Primjeri. 1. 01 je regularan izraz koji označava skup $\{01\}$. 2. $(0+1)^*$ označava skup $\Omega(\{0,1\})$. 3. $1(0+1)^*1+1$ označava skup svih riječi koje počinju i završavaju se sa slovom 1. Objašnjenja. 1. $0+1$ znači 0 ili 1. 2. 1^* znači 1 ma koliko puta (uključujući i mogućnost nula puta), tj. znači $\Box$ ili 1 ili 11 ili 111 ili itd. 3. $R = (0+1)2^*3^*$ znači 0 ili 1, a zatim 2 nekoliko puta, a zatim 3 nekoliko puta. Recimo, jeziku L koji je definisan posljednjim regularnim izrazom pripadaju između ostalih i riječi $w = 033$ i $w = 122223$.

Zadatak. Sastaviti regularne izraze za sljedeće skupove riječi u azbuci $A = \{a, b\}$. 1. Sve riječi u kojima se ne pojavljuju dva uzastopna slova a. 2. Sve riječi sa neparnim brojem slova a i parnim brojem slova b. 3. Sve riječi koje ne sadrže podriječ aba. Sami za vježbu.

Definicija. Poluproširenu regularan izraz: dodaje se operacija presjek, u oznaci $\cap$. Detaljnije, ako su p i q poluproširenu regularni izrazi, označimo sa P i Q njima odgovarajuće jezike, onda je i $p \cap q$ poluproširenu regularan izraz, s tim da njemu odgovara skup riječi $P \cap Q$.

Definicija. Proširenu regularan izraz: na mogućnosti koje ima poluproširenu dodata je operacija $\neg$, komplement. Konkretno, ako izrazu p odgovara jezik P onda je i $\neg p$ izraz, s tim da njemu odgovara jezik $\Omega(A) \setminus P$.

S jedne strane, klasa jezika koji mogu da budu prikazani ne izmijeni se (ne poveća se) dodavanjem mogućnosti presjeka i komplementa, to i dalje ostaje klasa svih regularnih jezika, ovo može da bude dokazano. S druge strane, upotreba dvije operacije presjek i komplement

može znatno da smanji dužine izraza potrebnih da se opišu neki skupovi riječi.

Prvi zadatak ili zadatak Z_1. Data je azbuka A i dat je regularan izraz R u toj azbuci čime je definisan jedan jezik L. Da li je komplement od L (u odnosu na skup $\Omega(A)$) prazan skup ili svejedno da li je $L = \Omega(A)$, tj. da li R označava sve moguće riječi u azbuci A. Kratko se kaže: zadatak o praznom komplementu.

Drugi zadatak ili zadatak Z_2. Data je azbuka A i dat je poluproširenи regularan izraz R u toj azbuci ...

Treći zadatak ili zadatak Z_3. Data je azbuka A i dat je prošireni regularan izraz R u toj azbuci ...

Sva tri zadatka su algoritamski rješivi. Da li su teški? Zapaziti odmah da je Z_2 teži od Z_1, tj. u najmanju ruku nije lakši od Z_1. Slično i Z_3 ima veću ili jednaku težinu (stepen složenosti) od Z_2. Zato što je složenost (govoreći opisno): broj taktova kroz obim ulaza; "imenilac" tj. obim ulaza kod Z_2 je manji (kraći) od onog kod Z_1.

Neko može da pročita deset strana za dva sata vremena. Efikasnost čitanja će biti bolja ako uspije da poveća broj strana ili da smanji vrijeme na satipo. Bilo koji algoritam za Z_3 može da posluži i za Z_1.

Što se tiče zadatka Z_1, nije poznato da li pripada ili ne pripada skupu $\mathcal{NP}$.

Teorema. Zadatak Z_2 ne pripada skupu $\mathcal{NP}$.

Teorema. Zadatak Z_3 ne pripada skupu $\mathcal{NP}$.

Izostavljaju se dokazi dvije navedene teoreme.

* * * * * * * * * *

Kao obnavljanje, skicirajmo definicije nekih važnih pojmova, radi njihovog pravilnog rasporeda. Kako treba definisati vremensku složenost? Da li se slaže sa realnim kompjuterom?

Znamo da u definiciji vremenske složenosti jednog algoritma učestvuju nezavisno promjenljiva n – obim ulaza (veličina instance) i zavisno promjenljiva t_n – odgovarajući vremenski trošak. U slučaju Tjuringove mašine, n je broj polja na traci upotrebljenih da se saopšte ulazni podaci (prije starta). Dok se t_n izražava preko broja potrebnih radnji, preko broja članova u nizu pozicija (od početne do završne). Na primjer, mašina za kopiranje K vrši obradu $a_0\, w\, a_0 \ldots \Rightarrow a_0\, w\, a_0\, w\, a_0 \ldots$ (gdje riječ $w \in \Omega(A)$) i za nju važi $t_n \sim 2n^2$. U slučaju računara, n predstavlja broj bajta (karaktera) za ulazne veličine. Drugim riječima, koliko puta treba pritisnuti taster da bismo računaru saopštili ulazne podatke. Ili svejedno: ako se podaci preuzimaju iz jednog fajla, kolika je dužina tog fajla u bajtima. Dok t_n znači: koliko se bit–operacija izvrši tokom rada programa. Lako se zapaža da su dvije skicirane tzv. idealne definicije vremenske složenosti, u slučaju Tjuringove mašine i u slučaju računara, u dobroj saglasnosti jedna sa drugom. Takve definicije otvaraju put prema izgradnji pravilne teorije složenosti algoritama, npr. teorije NP–kompletnih problema.

Primjer mašinske instrukcije je ADD EAX, EBX. Znamo da računar, baziran na procesoru Pentium ili nekom sličnom, izvrši 32 ili 64 bit–operacije u okviru jedne mašinske instrukcije. S druge strane, za jednu instrukciju treba u prosjeku 2–4 takta. Dalje, učestanost takta iznosi 1 GHz (to znači jedna nano–sekunda za jedan takt) ili 4 GHz ili tako. Prema tome, postoji direktna veza oblika $y = ax$ između broja izvršenih bit–operacija x i odgovarajućeg potrošenog vremena rada računara y izraženog npr. u sekundama. Naime, $a = 10^{-10}$ ili $a = 10^{-11}$. Dakle, postoji praktično potpuna saglasnost među veličinama: broj bit–operacija, potreban broj mašinskih instrukcija i potrošeno vrijeme po časovniku.

Programiranje II, Sadržaj predavanja:

2.1 Strukture podataka: liste, redovi i stekovi drugaen.tex

2.2 Predstavljanje skupova drugaen.tex

2.3 Grafovi (definicija i memorijsko predstavljanje) drugaeo.tex

2.4 Drveta (razne definicije, memorijsko predstavljanje i tri načina obilaska) drugaeo.tex

2.5 Rekurzija (inorder numeracija sa i bez rekurzije i rekurzija u slučaju RASP) drugaep.tex

2.6 Podijeli pa vladaj (najmanji i najveći član skupa i proizvod dva binarna broja) drugaep.tex

2.7 Balansiranje, algoritam mergesort drugaep.tex

2.8 Dinamičko programiranje (zadatak o množenju nekoliko matrica) drugaep.tex

3.1 Postavka zadatka o uređivanju drugaeq.tex

3.2 Radikalno uređivanje ili bucketsort drugaeq.tex

3.3 Uređivanje pomoću uspoređivanja (pomoću poređenja); donja granica drugaeq.tex

3.4 Heapsort drugaf1.tex

3.5 Quicksort drugaf1.tex

3.6 Order statistics (linearna složenost u najgorem slučaju) drugaf1.tex

3.7 Order statistics, produžetak (linearna složenost u očekivanom slučaju) drugaf1.tex

4.1 Osnovne operacije nad skupovima drugag.tex

4.2 Hešovanje drugag.tex

4.3 Binarno pretraživanje drugag.tex

4.4 Drveta binarnog pretraživanja drugag.tex

4.5 Optimalna drveta binarnog pretraživanja drugag.tex

4.6 Jednostavni algoritam za uniju dva disjunktna skupa drugag.tex

4.7 Šeme (algoritmi) koje koriste balansirana drveta drugag.tex

5.1 Drvo koje povezuje minimalne cijene (Kruskalov algoritam za obuhvatno drvo) drugah.tex

5.2 Obilazak grafa po dubini drugah.tex

5.3 Algoritam za tranzitivno zatvorenje (da li postoji put) drugai.tex

5.4 Algoritam za najkraći put drugai.tex

5.5 Zadatak o jednom izvoru (Dijkstra) drugai.tex

5.6 Zadatak o nezavisnom skupu i backtracking drugaj.tex

5.7 Zadatak o trgovačkom putniku i backtracking drugaj.tex

6.1 Euklidov algoritam i njegova složenost (za NZD) drugak.tex

6.2 Neke teoreme iz teorije brojeva (mala Fermaova i kineska o ostacima) drugak.tex

6.3 Uopšteni Euklidov algoritam drugak.tex

6.4 Kriptografija pomoću javnog ključa: način RSA drugal.tex

6.5 Digitalni potpis drugal.tex

6.6 Kriptografija pomoću javnog ključa: metoda kvadratnog ostatka drugal.tex

6.7 Ispitivanje primalnosti: Millerov postupak drugal1.tex

6.8 Ispitivanje primalnosti: Miller–Rabinov postupak drugal1.tex

6.9 Stream cipher drugal1.tex

6.10 Prattov sertifikat primalnosti drugal2.tex

7.1 Nedeterministička Tjuringova mašina drugal3.tex

7.2 Tri klase jezika drugal3.tex

7.3 Jezici i zadaci (razumna šema za kodiranje i postavke nekih zadataka) drugal3.tex

7.4 Dokaz da je zadatak SAT NP–kompletan drugal3.tex

7.5 Dokaz da je zadatak o kliki NP–kompletan drugam.tex

7.6 Neki zadaci za koje je dokazano da su teško rješivi drugam.tex

Fajlovi (gsview): drugaen.tex 3 pages drugaeo.tex 5 pages drugaep.tex 10 pages drugaeq.tex 4 pages drugaf1.tex 8 pages drugag.tex 15 pages drugah.tex 5 pages drugai.tex 7 pages drugaj.tex 6 pages drugak.tex 6 pages drugal.tex 10 pages drugal1.tex 7 pages drugal2.tex 3 pages drugal3.tex 8 pages drugam.tex 4 pages. $\sum =$ 15 files & 101 pages