Osnovi računarstva II
Čas 2

Miloš Daković

Elektrotehnički fakultet – Podgorica

22. februar 2021.

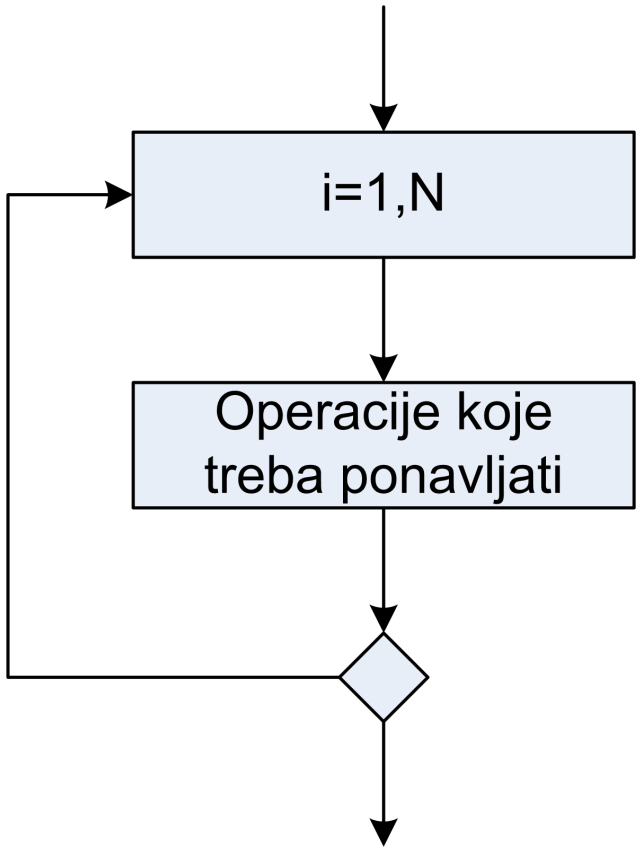
Ciklusi

- Određeni broj algoritamskih koraka koji se ponavlja više puta se naziva ciklusom.
- Postoji više tipova ciklusa:
  - ciklusi koji se ponavljaju tačno određen broj puta
  - ciklusi koji se ponavljaju dok je ispunjen određeni logički uslov. U zavisnosti od mjesta u ciklusu gdje se logički uslov provjerava imamo:
    - cikluse sa izlazom na početku
    - cikluse sa izlazom na kraju
    - cikluse sa izlazom u sredini
- Svi ciklusi se mogu svesti na ciklus sa izlaskom na početku!

Ciklus sa unaprijed zadatim brojem izvršavanja

- Određenu operaciju treba ponoviti N puta gdje je N unaprijed zadati broj
- Koristi se brojačka varijabla
- Najčešće korišćeni tip ciklusa
- Primjeri:
  - Unos N elemenata niza
  - Računanje zbira elemenata niza dužine N
- Pseudokod:

```
FOR i = 1,N
   ...
NEXT
```

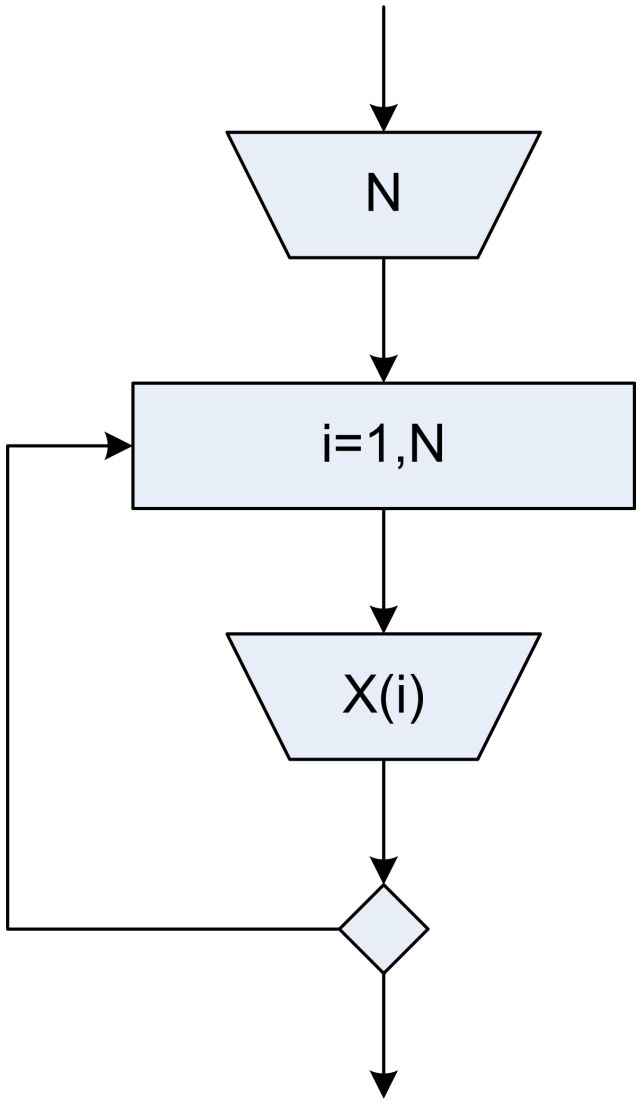


Primjer: Unos elemenata niza

- Broj elemenata niza mora biti unaprijed poznat
- U jednom ulaznom koraku može se učitati samo jedan element niza

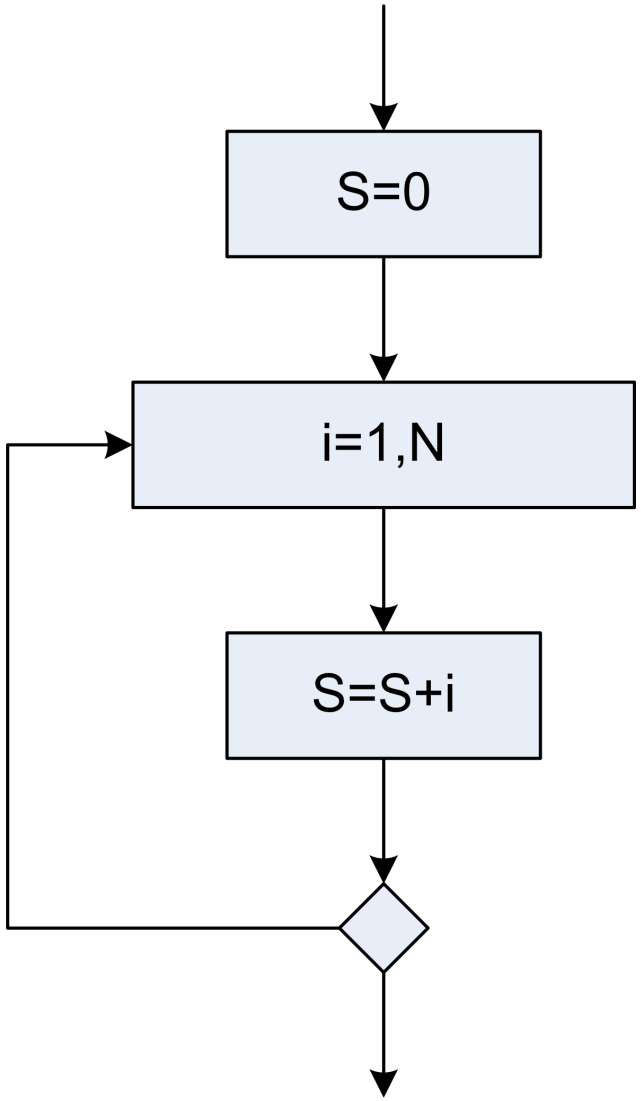
Pseudokod:

```
...
INPUT N
FOR  i = 1,N
   INPUT X[i]
NEXT
...
```



Primjer: Računanje zbira prvih N prirodnih brojeva

- Rezultat je u varijabli S
- Inicijalizacija sume ($S = 0$)
- Podrazumijeva se da je N poznato prije izvršavanja navedenih koraka i da je suma S iskorišćena u nastavku algoritma.

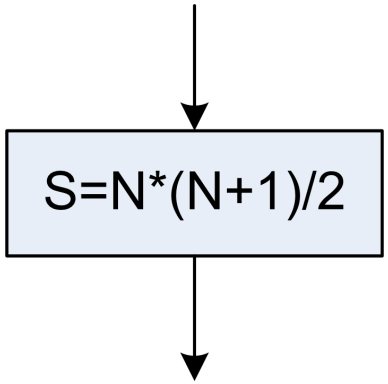


Pseudokod:

```
...
S = 0
FOR  i = 1,N
    S = S+i
NEXT
...
```

Primjer: Računanje zbira prvih N prirodnih brojeva

- Alternativno rješenje bez upotrebe ciklusa
- Smanjen je broj računskih operacija
- Zahtijeva detaljniju analizu posmatranog problema

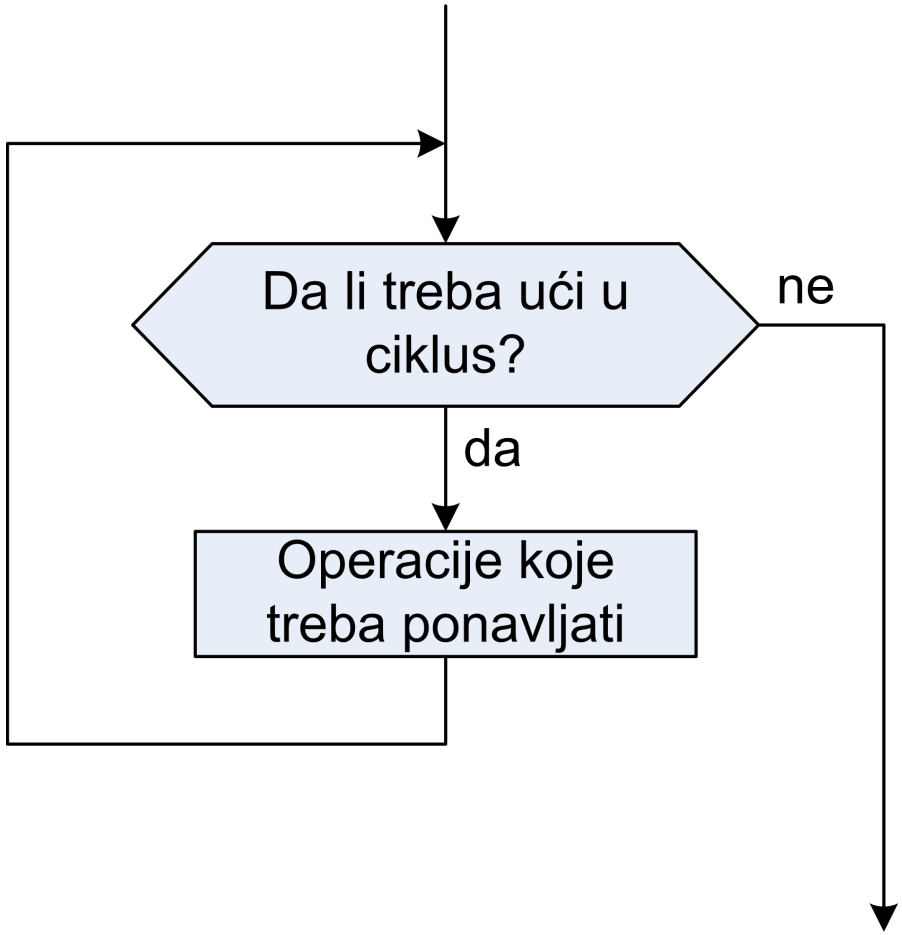


Pseudokod:

```
...
S = N*(N+1)/2
...
```

Ciklus sa izlazom na početku

- Broj ponavljanja ciklusa može biti nula (na samom početku uslov nije zadovoljen)
- Primjena:
  Polazimo od mogućeg rješenja problema i popravljamo ga u svakoj iteraciji (ciklusu) sve dok ne budemo zadovoljni

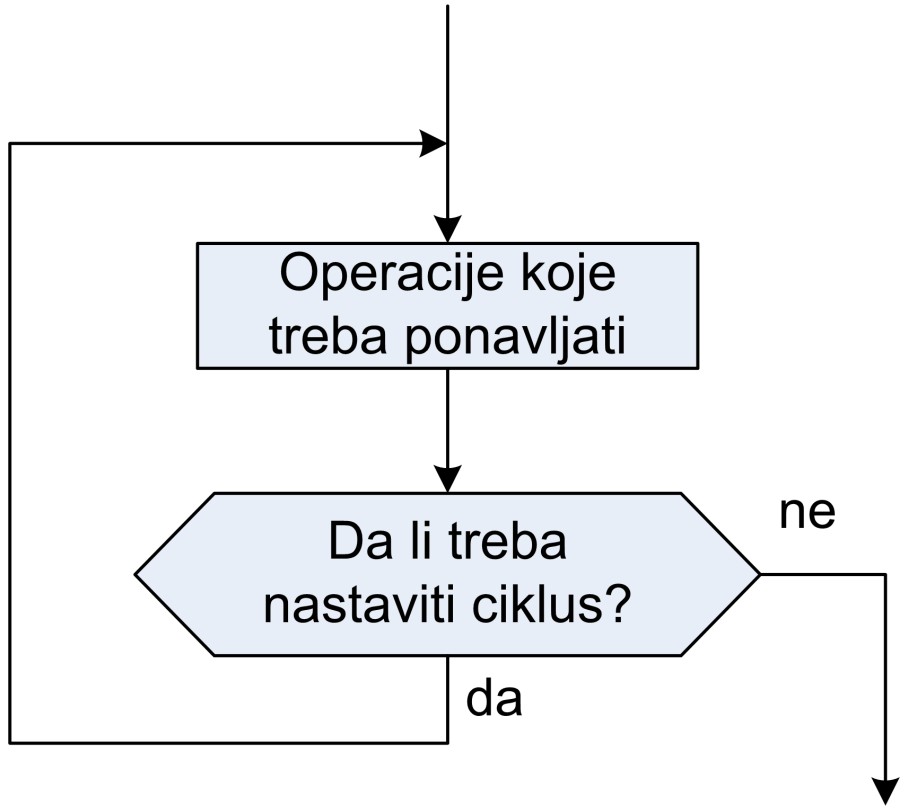


Pseudokod:

```
WHILE uslov
    ...
ENDWHILE
```

Ciklus sa izlazom na kraju

- Broj ponavljanja ciklusa je najmanje 1
- Primjena:
  Unos podataka koji moraju zadovoljiti postavljene kriterijume
  Primjer: od korisnika tražimo unos pozitivnog cijelog broja



Pseudokod:

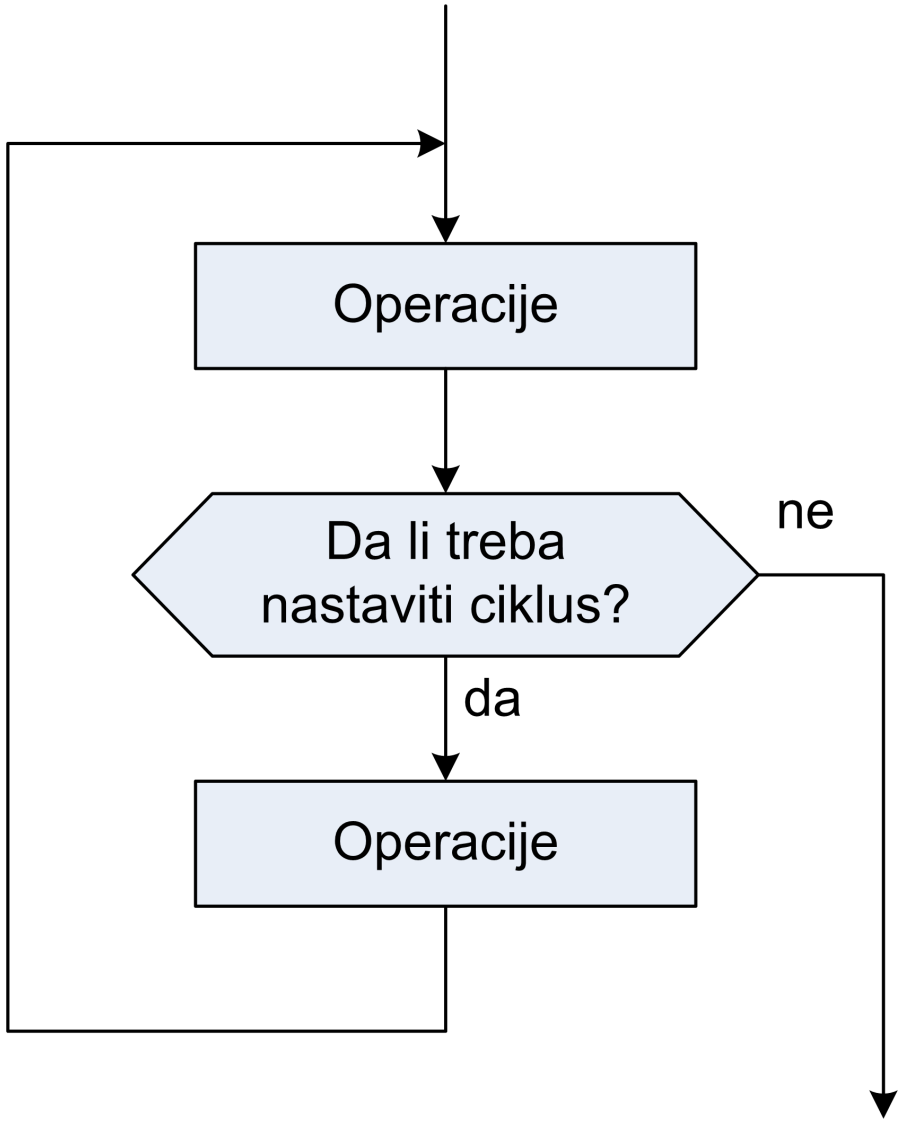
```
DO
    ...
WHILE uslov
```

Ciklus sa izlazom u sredini

- Dio ciklusa mora biti izvršen a nakon toga se donosi odluka da li nastaviti sa izvršavanjem drugog dijela.
- Svaki ciklus se može svesti na ciklus sa izlazom na početku.

Pseudokod:

```
DO
    ...
IF uslov THEN EXIT
    ...
LOOP
```



Ciklus – primjer

Kao primjer ciklusa (onog koji se izvršava nepoznat broj puta, odnosno, dok je zadovoljen određeni logički uslov) navodimo Njutnov algoritam za približno određivanje kvadratnog korijena realnog broja.

Korak 1. Neka tražimo kvadratni korijen broja X. Kao početno pogađanje njegovog kvadratnog korijena uzmimo $G = X/2$. Izračunajmo broj H kao $H = X/G$.

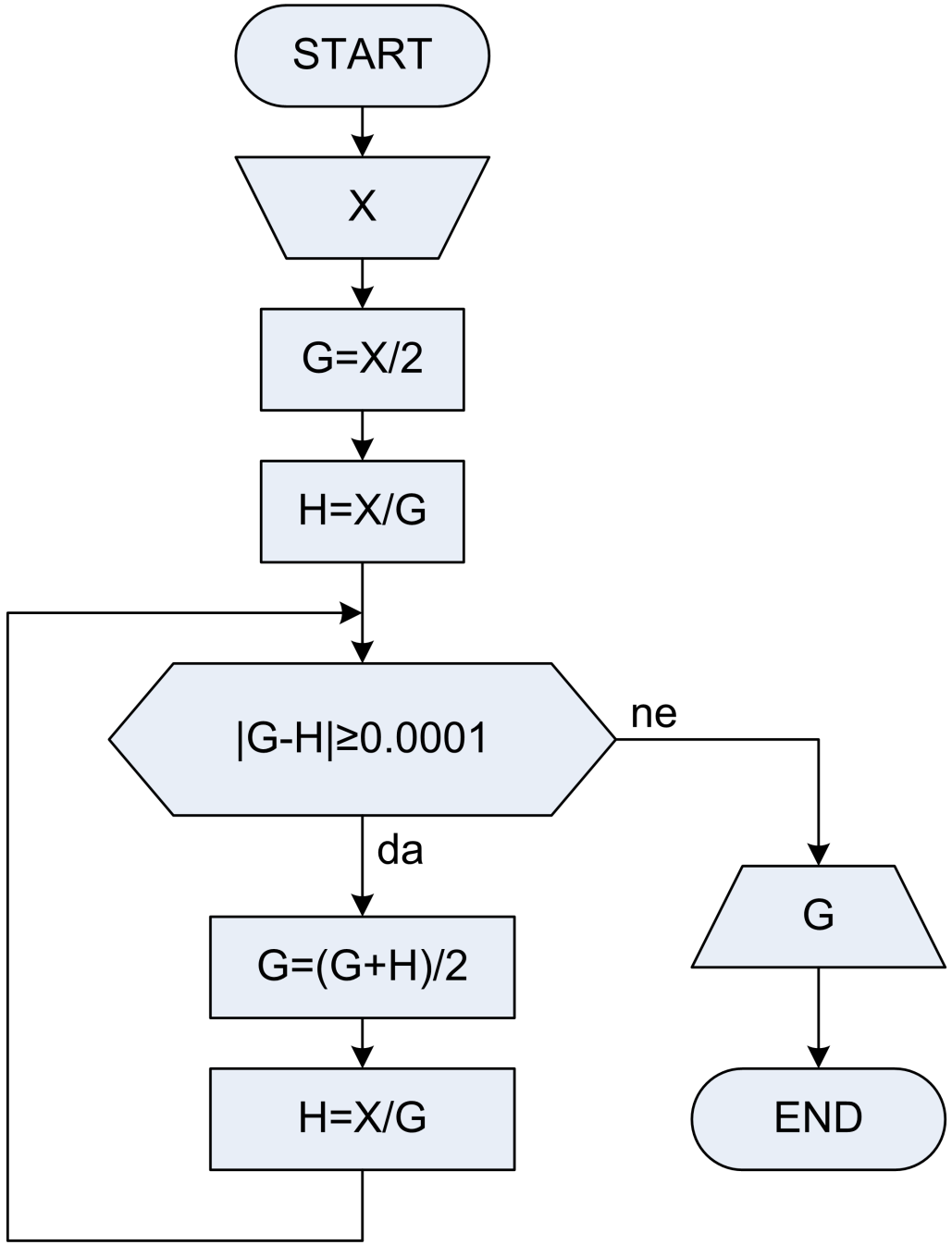
Korak 2. Ako su G i H *približno jednaki* s obzirom da je $G * H$ jednako X, zaključujemo da su G i H približno jednaki kvadratnom korijenu broja X.

Korak 3. Ako G i H nijesu približno jednaki onda treba uzeti da je novo G jednako $G = (G + H)/2$ i ponovo sračunati H kao $H = X/G$.

Korak 4. Koraci 2 i 3 se ponavljaju dok se ne dođe do rješenja.

Kvadratni korijen
Pseudokod i algoritam

```
START
INPUT X
G = X/2
H = X/G
WHILE abs(G-H)>=0.0001
    G=(G+H)/2
    H=H/G
ENDWHILE
OUTPUT G
END
```



Ciklusi i selekcija

- Unutar jednog ciklusa se može nalaziti proizvoljna sekvenca naredbi.
- Nekoliko ciklusa mogu jedan za drugim da čine sekvencu.
- Unutar ciklusa se može nalaziti selekcija.
- Unutar selekcije se može nalaziti ciklus.
- Unutar selekcije se može nalaziti druga selekcija
- Unutar ciklusa se može nalaziti drugi ciklus (to se naziva ugnježdavanje ciklusa).

Važno pravilo je da se prvo završava unutrašnji dio (unutrašnji ciklus ili selekcija), pa tek onda spoljašnji, tj. ciklusi i selekcije ne mogu se sjeći.

Primjeri za vježbu – rad sa matricama

1. Sabiranje, oduzimanje i množenje dvije matrice, sa provjerom da li im dimenzije omogućavaju ove operacije
2. Odrediti sumu (ili proizvod) elemenata matrice
3. Odrediti maksimalni (ili minimalni) element matrice
4. Elemente date matrice upisati u niz (vrstu po vrstu)
5. Odrediti sumu (ili proizvod) svih kolona (ili vrsta) matrice
6. Kreirati vektor koji se sastoji od elemenata matrice sa glavne (ili sporedne) dijagonale
7. Provjeriti da li su učitane kvadratne matrice inverzne
8. Sračunati trag matrice (trag matrice je suma elemenata glavne dijagonale matrice)
9. Provjeriti da li je matrica simetrična
10. Odrediti transponovanu matricu date matrice

Složenost algoritama

- Broj operacija potreban da se algoritam izvrši od početka do kraja
- Memorijski zahtjevi algoritma
- U najjednostavnijim primjerima broj operacija i memorijski zahtjevi algoritma su konstantni.
- Primjer: Učitati tri broja A, B i C i izračunati izraz $2A + B/C$
  - broj računskih operacija je 3
  - 4 memorijske lokacije su iskorišćene (3 za ulazne podatke i jedna za rezultat)
  - pod memorijskom lokacijom se smatra memorija dovoljna za pamćenje jednog „broja“.
- Nijesu svi algoritmi koji rješavaju neki problem jednako kvalitetni.
- Mjera kvaliteta algoritama je složenost.

Složenost algoritama – tipovi složenosti

- Vremenska složenost
  - Vrijeme potrebno za izvršavanje algoritma.
  - Proporcionalno je broju operacija (aritmetičkih, logičkih ili nekih drugih), koje algoritam izvršava.
  - Dakle, procjena vremenske složenosti se svodi na brojanje operacija u algoritmu.
- Prostorna složenost
  - Memorisjki prostor koji zauzimaju svi podaci korišćeni u algoritmu.
  - Određuje se sabiranjem veličine memorijskih lokacija koje zauzimaju promjenljive upotrijebljene u algoritmu.
- Komunikaciona složenost
  - Određuje koliko je potrebno komunikacije procesora sa periferijama, diskovima i memorijom prilikom izvršavanja programa.
  - Posebno je značajna kod paralelnih kompjutera, kada razni procesori međusobno komuniciraju.

Vremenska složenost algoritama

- Mi ćemo se detaljnije zadržati na vremenskoj složenosti i donekle na prostornoj, dok komunikacionu nećemo razmatrati.
- Pod vremenskom složenošću se podrazumjeva vrijeme izvršavanja algoritma koje je proporcionalno broju operacija u algoritmu

$$\text{vrijeme} = \text{broj operacija} \times \text{neki konstantni interval}$$

- Teško je odrediti svaku operaciju u algoritmu, a teško je generalno i porediti operacije kao što su sabiranje i npr. operacije poređenja.
- Uočeno je da, u pojedinim algoritmima, neka operacija dominira nad svim ostalim i da se češće upotrebljava.
- Na primjerima ćemo videti kako se broje aritmetičke operacije.

Računanje vremenske složenosti algoritma

- Kod sekvence se broje operacije na koje se naiđe. Na primjer, ako su dominantne operacije u algoritmu sabiranje i množenje:

```
A = B+C+D
E = A+F
G = A*B
```

3 sabiranja i 1 množenje (kod prve naredbe postoje dva sabiranja).

- Kod selekcije se uzima varijanta sa više operacija:

```
IF(A>3)
   B = A+B+3
ELSE
   C = A+1
ENDIF
```

2 sabiranja (uzeta je gora varijanta).

Računanje vremenske složenosti algoritma

- Kod ciklusa se broj operacija tokom jednog izvršavanja ciklusa množi sa brojem ponavljanja ciklusa:

```
I=1
WHILE I<12
    A=A+B
    I=I+1
ENDWHILE
```

ukupno 22 operacije sabiranja (11 ponavljanja po 2 operacije)

dodatno: 12 operacija poređenja i 23 operacije dodjele vrijednosti

```
FOR I=1,N
   S=S+I;
   P=P*I+1;
NEXT
```

Ukupno $3N$ operacija. Dva sabiranja i jedno množenje po svakom prolazu kroz ciklus.

„Skrivena“ operacija uvećavanja brojača za 1.

Računanje vremenske složenosti algoritma

- Primjer kada se unutar jednog ciklusa nalazi drugi ciklus:

```
FOR I=1,N
   J=0
   WHILE J<I
      J=J+1
      IF P<100
         P=P+I*J
      ENDIF
      S=S+I*J
   ENDWHILE
NEXT
```

Unutrašnji ciklus se ponavlja I puta. U unutrašnjem ciklusu imamo tri sabiranja i dva množenja, dakle 5 operacija.

Ukupan broj operacija je

$$1 \times 5 + 2 \times 5 + \cdots + N \times 5 = \frac{5N(N+1)}{2}$$

odnosno proporcionalan sa N^2. Uočite da je u naredbi selekcije posmatran najgori slučaj.

Računanje vremenske složenosti algoritma

- Šta raditi kada ne postoji jedinstvena veza ulaznih podataka i složenosti? Postoje dvije strategije:
  - Analiza najgoreg slučaja (worst case analiza)
    Uzme se najgori mogući slučaj i kaže se da algoritam ima toliku vremensku složenost u najgorem slučaju.
  - Analiza prosječnog slučaja (average case analiza)
    Ovdje analiziramo sve moguće slučajeve, dijelimo ih u klase (obilježimo broj klasa sa K), pri čemu smo za svaku klasu u stanju da procijenimo potreban broj operacija $N_1, N_2, \ldots N_K$. Neophodno je znati i vjerovatnoće sa kojima se posmatrana klasa javlja na ulazu algoritma $p_1, p_2, \ldots p_K$. Zbir ovih vjerovatnoća mora biti jednak 1. Prosječan broj operacija se računa kao

$$N_{op} = \sum_{i=1}^{K} p_i N_i = p_1 N_1 + p_2 N_2 + \cdots + p_K N_K$$

Računanje vremenske složenosti algoritma

- Primjer: Parni i neparni prirodni brojevi se pojavljuju kao ulazni podaci algoritma sa jednakim vjerovatnoćama. Broj operacija kod parnih brojeva je $N+5$, dok je kod neparnih $2N-1$. Šta je najgori slučaj, a šta je prosječni slučaj?
  - Najgori slučaj je kada se na ulazu pojavi neparan broj (veći od 6, zašto?). Tada je broj operacija jednak $2N-1$
  - Imamo $K=2$ klase ulaznih podataka (parne i neparne brojeve), vjerovatnoće pojavlivanja klasa su $p_1=\frac{1}{2}$ i $p_2=\frac{1}{2}$. Odgovarajući brojevi operacija su $N_1=N+5$ i $N_2=2N-1$, a prosječan broj operacija

$$N_{op}=\frac{1}{2}(N+5)+\frac{1}{2}(2N-1)=\frac{3}{2}N+2$$

O – notacija

- Često je teško, ako ne i nemoguće, doći do tačnog broja potrebnih računskih operacija. U takvim slučajevima se koristimo aproksimacijama i dolazimo do približnih rezultata.
- Neka je sa $g(N)$ obilježen tačan broj operacija potreban za izvršenje algoritma, pri čemu je ulazni podatak N red problema.
- Kažemo da je potreban broj operacija reda ne većeg od $f(N)$, odnosno $g(N) = O(f(N))$ ukoliko je

$$\lim_{N \to \infty} \frac{g(N)}{f(N)} = const. < \infty$$

Na primjer, ako je tačan broj operacija $g(N) = 4N^3 - 32N^2 + 125$ možemo reći da je $g(N) = O(N^3)$.

Algoritam – primjer

- Za zadati prirodan broj N treba odrediti sve nizove uzastopnih prirodnih brojeva, čija je suma jednaka N.
- Na primjer, ako je korisnik zadao $N = 21$, treba da budu prikazani sljedeći nizovi:
  - 1, 2, 3, 4, 5, 6
  - 6, 7, 8
  - 10, 11
  - 21
- Krećemo od 1 i sabiramo sa 2, 3, 4, 5, 6 i dobijamo 21, dakle jedno rješenje je niz 1, 2, 3, 4, 5 i 6.
- U narednoj iteraciji počinjemo od 2. Formiramo zbir 2+3+4+5+6+7 i taj je zbir veći od 21, dakle pošto mu je suma veća od 21 ne postoji niz uzastopnih prirodnih brojeva koji počinje sa 2 i daje zbir 21.
- Nastavljamo sa narednim iteracijama (brojevi 3, 4,..., N).
- Zadatak sigurno ima jedno rješenje (niz od jednog prirodnog broja N)

Algoritam – analiza

- Koji su ulazni podaci, šta je izlaz iz algoritma i koja je struktura ovakvog programa?
- Učitavamo ulazni podatak N.
- U jednom ciklusu (petlji) idemo od $K = 1$ do $K = N$.
- Unutar ciklusa inicijalizujemo sumu S na $S = K$, jer za svako K pretpostavljamo da može da bude prvi element niza uzastopnih prirodnih brojeva čija je suma jednaka N.
- U ugnježdenoj petlji (ciklusu) na sumu S dodajemo uzastopne brojeve počevši od $P = K + 1$ i ostajemo u petlji sve dok je ta suma S manja od N.
- Ako je $S \equiv N$ štampamo (prikazujemo) niz brojeva od K do P.
- Zatim prelazimo na narednu vrijednost K.

Algoritam – pseudokod

Algoritam je realizovan pomoću brojačke petlje po varijabli K. Unutrašnja petlja je ciklus sa izlaskom na početku. Za ispis rezultata korišćena je brojačka petlja.

```
START
N,K,S,P,M :INTEGER
INPUT N
FOR K = 1,N
    S = K
    P = K
    WHILE S<N
        P = P+1
        S = S+P
    ENDWHILE
    IF S == N
        FOR M = K,P
            OUTPUT M
        NEXT
    ENDIF
NEXT
END
```

Algoritam – efikasnije rješenje

- Naš cilj je da nađemo nizove cijelih brojeva $K, K+1, \ldots, K+M$ čija je suma jednaka N. Sumu ovog niza možemo izračunati:

$$K + (K+1) + \cdots + (K+M) = K(M+1) + \frac{M(M+1)}{2}$$

Problem se svodi na rješavanje jednačine

$$K(M+1) + \frac{M(M+1)}{2} = N \qquad K = \frac{N - \frac{M(M+1)}{2}}{M+1}$$

- Ne zaboravimo da su K i M prirodni brojevi, pri čemu M može biti jednako nuli.

Algoritam – efikasnije rješenje

- Formirajmo petlju po M, koja kreće od 0 (jer se niz može sastojati i od samo jednog člana K). Zatim, za posmatrano M, provjerimo da li postoji prirodan broj K koji je rješenje posmatrane jednačine.
- Ukoliko postoji rješenje po K štampamo niz brojeva $K, K+1, \ldots, K+M$ kao jedno od rješenja našeg problema.
- Problem koji trebamo pomenuti, prije nego pređemo na pisanje koda, je gornja granica za M. Očigledno je, da M mora zadovoljiti uslov

$$\frac{M(M+1)}{2} < N$$

Na primjer, za $N = 10$ dovoljno je da petlja po M ide do 3, jer već za $M = 4$ nije moguće ispunjenje prethodnog uslova.

Algoritam – efikasnije rješenje – pseudokod

```
START
N,M,P,K,R :INTEGER
INPUT N
M=0
WHILE M*(M+1)/2<N
    P = N-M*(M+1)/2
    IF P-[P/(M+1)]*(M+1) == 0
        K = P/(M+1)
        FOR R = K,K+M
            OUTPUT R
        NEXT
    ENDIF
    M = M+1
ENDWHILE
END
```

IF naredbom provjeravamo da li je P djeljivo bez ostatka sa $M+1$.

Uočite da se WHILE petlja izvršava približno $\sqrt{N}$ puta.

Vremenska složenost ovog algoritma je $O\left(\sqrt{N}\right)$.

U prethodnom slučaju složenost je $O\left(\sqrt{N^3}\right)$.