Osnovi računarstva II

Algoritamski koraci - ciklus

Složenost algoritma

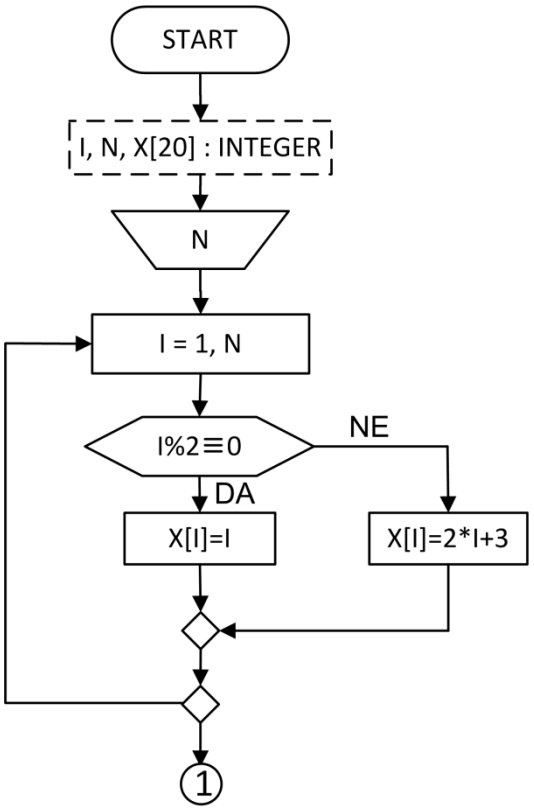
Nizovi - Primjer 2

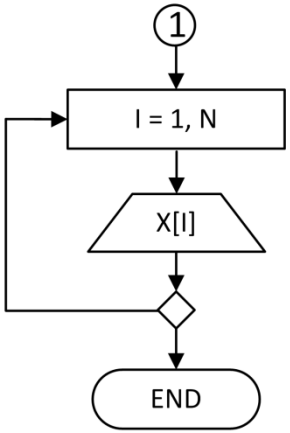
- Nacrtati algoritam kojim se unosi dužina niza N i formira i štampa niz X od N elemenata zadatih sljedećom relacijom:

$$X[I]=\begin{cases} I, & \text{za parno I} \\ 2I+3, & \text{za neparno I.} \end{cases}$$

Na primjer, za **N=5**, niz je

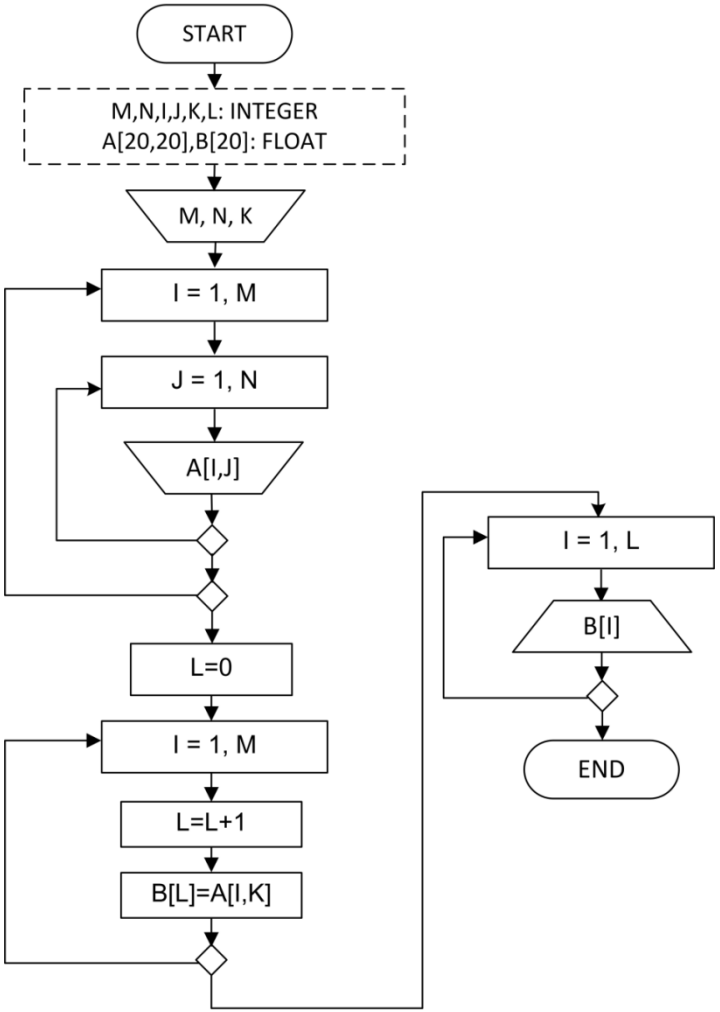
X = 5, 2, 9, 4, 13

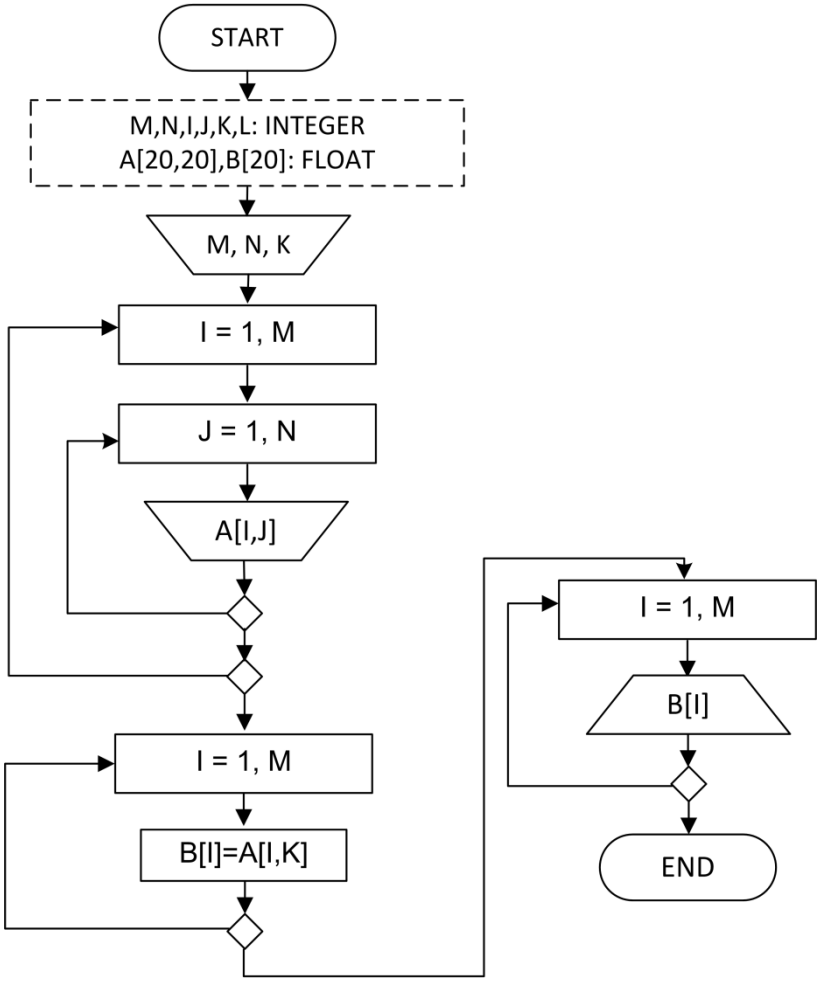




Matrice – Primjer 1

Za datu matricu A, formirati niz B sastavljan od elemenata kolone koju zadaje korisnik.





Složenost algoritama

- Nijesu svi algoritmi koji rješavaju neki problem jednako kvalitetni.
- Mjera kvaliteta algoritama je složenost.
- Tipovi složenosti su:
  - Vremenska složenost
    - Koje vrijeme je 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.

Složenost algoritama

- Prostorna složenost
  - Podrazumjeva koliko prostora u memoriji zauzimaju podaci 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

- 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
  vrijeme = broj operacija x neki konstantni interval.

Vremenska složenost

- 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.
- Sada ćemo videti kako se broje aritmetičke operacije.

Vremenska složenost

- Kod sekvence se broje operacije na koje se naiđe. Na primjer, ako je dominantna operacija u algoritmu sabiranje:

```
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)

Vremenska složenost

- 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 24 operacije (12 ponavljanja po 2 op.)

- Drugi primjer:

```
I=1
WHILE I≤N
   A=A+B
   I=I+1
ENDWHILE
```

→ ukupno 2N operacija

Vremenska složenost

- Ugniježdeni ciklusi:

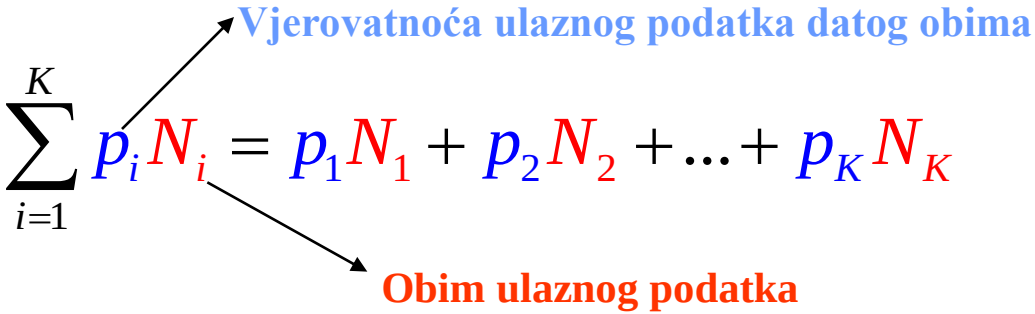
```
WHILE I≤N
  WHILE J≤N
    ....
    J=J+1
  ENDWHILE
  ...
  I=I+1
ENDWHILE
```

Za svaku iteraciju spoljašnjeg ciklusa, izvršava se kompletan unutrašnji ciklus, koji ima N iteracija.

Broj operacija proporcionalan sa N^2

Vremenska složenost

- Što 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 (ovo je češća strategija).
  - Analiza prosječnog slučaja (average case analiza)
    - Za svaki mogući obim ulaznih podataka treba znati koliko operacija je potrebno. Također, treba znati sa kojom se vjerovatnoćom taj podatak pojavljuje. Prosječan broj operacija je:



Vremenska složenost

- Za vjerovatnoće važi:

$$\sum_{i=1}^{K} p_i = p_1 + p_2 + \ldots + p_K = 1$$

- **Primjer:** Parni i neparni brojevi se pojavljuju u po polovini slučajeva. Obim operacija kod parnih brojeva je reda veličine **N**, dok je kod neparnih **2N**. Što je najgori slučaj, a što je prosječni slučaj?
- **Odgovor:** Najgori slučaj je **2N**, dok je prosječni

$$\frac{1}{2} N + \frac{1}{2} 2N = \frac{3}{2} N$$

Vremenska složenost

- Najčešće je veoma teško utvrditi kolika je **tačna** vremenska složenost, ali se može procijeniti red veličine broja potrebnih operacija.
- Neka je potreban broj operacija za računanje algoritma $g(N)$. Kaže se da je broj operacija reda $\Omega(f(N))$ ako važi:

$$\lim_{N\to\infty} \frac{g(N)}{f(N)} = c \in \mathbf{R}^{+}$$

Skup pozitivnih realnih brojeva (ne uključuje nulu i beskonačno)

Primjer: $g(N)=6N^2+3N+1$ je reda $\Omega(N^2)$

- Posmatrajmo sljedeći 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, jer postoji mogućnost da i sa tim brojem kao početnim imamo niz uzastopnih brojeva, koji daju sumu 21. 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 narednom iteracijom (broj 3).

Zadatak sigurno ima jedno rješenje (niz od samo jednog prirodnog broja N)

Analiza prethodnog primjera

- Koji su podaci i koja je struktura ovakvog programa?
- Učitavamo N.
- U jednoj petlji idemo od K=1 do K=N.
- Svaki put u petlji po K 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 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≡N štampamo (prikazujemo) niz brojeva od K do P, a ako je S>N ne štampamo.
- Zatim prelazimo na sljedeće K i ponovimo postupak.
- Ovdje ćemo dati i pseudokod ovog algoritma.

Primjer - Nizovi koji daju sumu N

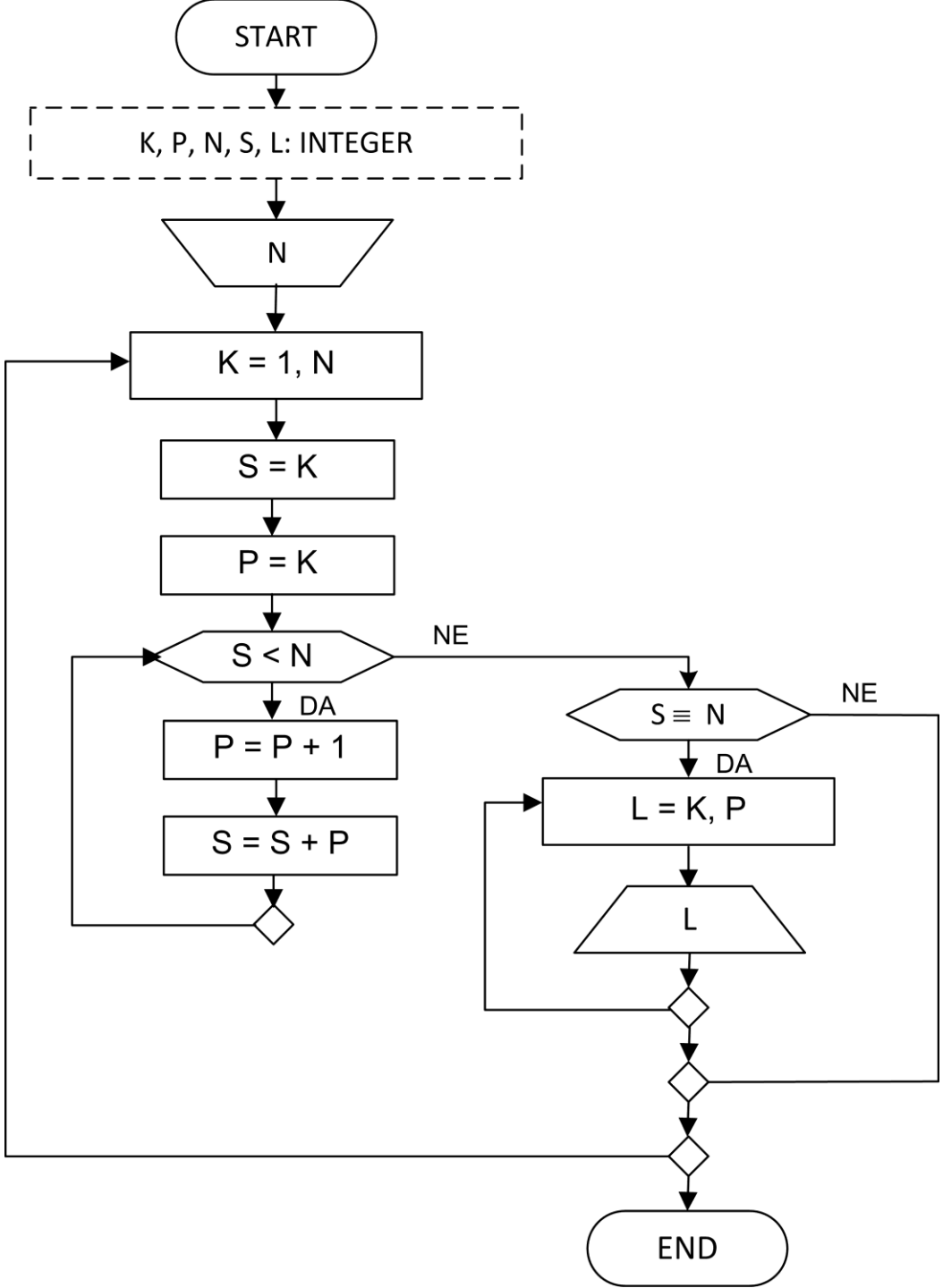
```
START
K,N,P,S,L:INTEGER
INPUT N
K = 1
WHILE K ≤ N
  S = K
  P = K
  WHILE S < N
    P = P + 1
    S = S + P
  ENDWHILE
  IF S ≡ N
    M = K
    WHILE L ≤ P
      OUTPUT L
      L = L + 1
    ENDWHILE
  ENDIF
  K = K + 1
ENDWHILE
END
```

Primjer - Nizovi koji daju sumu N

Isti problem se može riješiti korišćenjem FOR petlje

```
START
K,N,P,S,L : 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 L = K, P
      OUTPUT L
    NEXT
  ENDIF
NEXT
END
```

Primijetite da je algoritam čitljiviji ukoliko koristimo petlju sa poznatim brojem ponavljanja, kada je taj broj poznat.



Primjer - Nizovi koji daju sumu N

Primjer – Efikasnije rješenje

- Naš cilj je da nađemo niz cijelih brojeva (K, K+1, ..., P) čija je suma jednaka N. Zapišimo ovaj niz kao: (K, K+1, ..., K+M). Njegova suma je jednaka:

$$\sum_{R=K}^{K+M} R = K(M+1) + \frac{M(M+1)}{2}$$

- Dakle, naš problem se svodi na rješavanje jednačine:

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

sa parom rješenja (K,M) iz skupa prirodnih brojeva.

Primjer – Efikasnije rješenje

- Postavljeni problem se sada može riješiti efikasnije.
- Formiramo petlju po **M**, koja ide od 0 (jer se niz može sastojati i od samo jednog člana **K**) ka nekom cijelom pozitivnom broju (vidjećemo koji je to broj).
- Zatim se za posmatrano **M** provjerava da li postoji cijeli broj **K** koji je rješenje jednačine:

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

Prethodna jednačina je riješena po K, koji predstavlja prvi u nizu brojeva koji se sumiraju.

- Ako postoji, onda se odštampa odgovarajući niz od **K** do **K+M**, a ako ne postoji posmatra se sljedeće **M**.

Primjer – Efikasnije rješenje

- Problem koji trebamo pomenuti, prije nego pređemo na pisanje koda, su granice za M. Očigledno je, da je maksimalno M posljednje M za koje je zadovoljen uslov M(M+1)/2<N. Npr. za N=10 dovoljno je da petlja po M ide do 3, jer već za M=4 nije moguće ispunjenje prethodnog uslova.
- Pseudokod realizacije:

```
START
K,P,M,N,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
```

Primjer - *Komentari*

- Druga realizacija je, očigledno, znatno efikasnija. Isključujući dio za štampanje niza, čitav kod se odvija u jednoj petlji i npr. za N=10 samo se četiri puta ulazi u ciklus za M=0, 1, 2 i 3. Prva realizacija je zahtjevala dvostruki ciklus i 15 ulazaka u unutrašnju petlju.
- Postoje mogućnosti da se još malo popravi efikasnost programa, uz neznatno više analize programa.
- Dakle, nijesu sva rješenja jednog problema istog kvaliteta.
- Bolja su ona rješenja koja daju manji broj potrebnih operacija.
- Posebno je važno da analiza, koja vodi do razvoja “jednostavnijeg” algoritma, nije posebno složena.