I Konstrukcija algoritama

Algoritam koji zadovoljava rešenje formulisanog problema opisuje se

- koracima
- skupom ulaznih parametara
- svojstvima koje moraju posedovati vrednosti na izlazu
- skupom izlaznih vrednosti

Primer : Sortiranje

Input: a, N (**a** je niz od N celih brojeva) $a_1 \ldots a_n$

Output: niz A sortiran u nerastućem poretku $a_1 <= a_2 <= \ldots <= a_n$

kôdiranje:

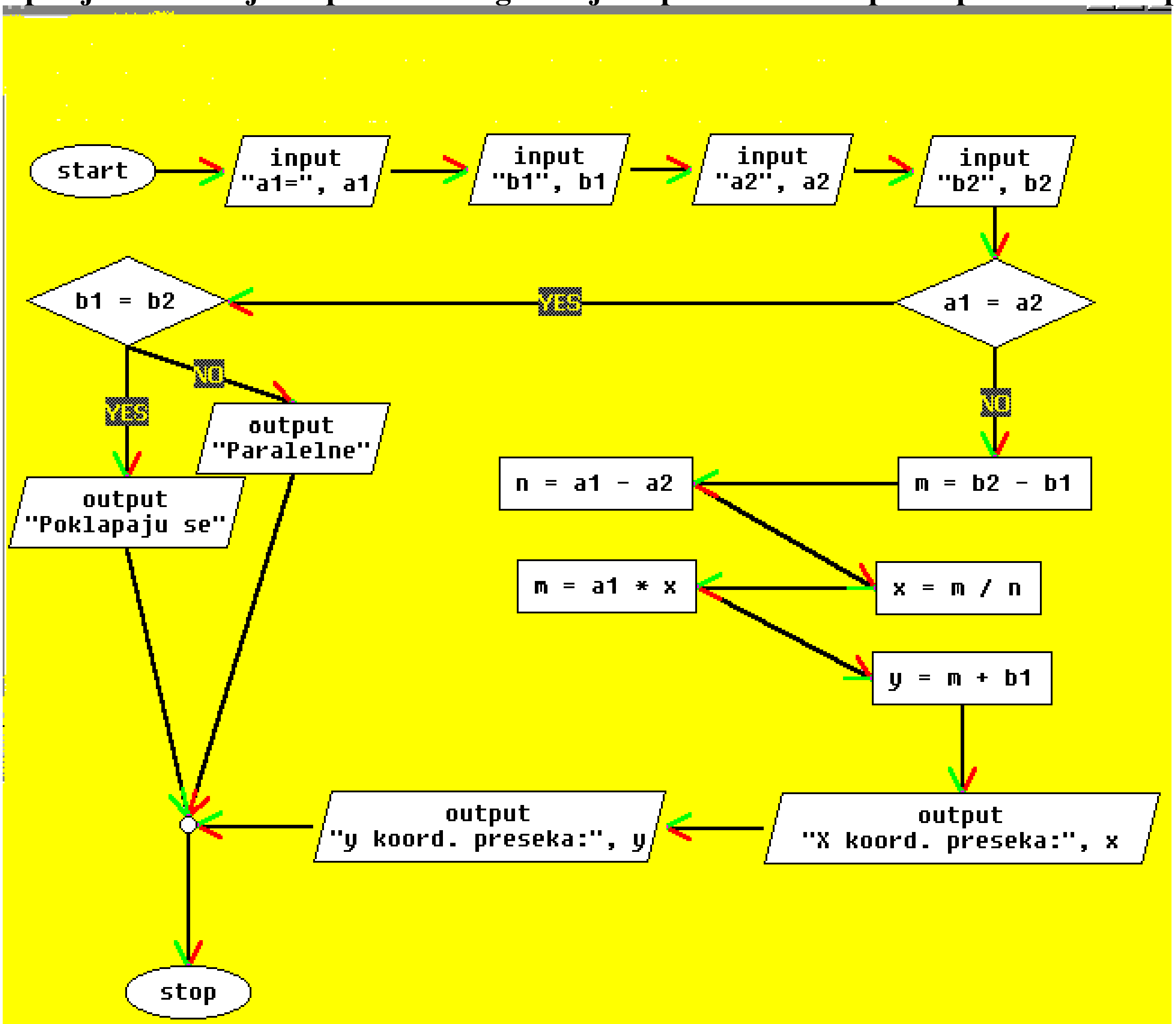
- shemom
- pseudo-kôdom
- programskim jezikom
- prirodnim jezikom

kôdiranje programskim jezikom:

```
for(j=0; j<N-1; j++)
  for(k=j+1; k<N; k++)
  if( A[j]>A[k] )
   {
     int rezerva;
     rezerva=A[j];
     A[j]=A[k];
     A[k]=rezerva;
    }
```

kôdiranje shemom:

Formirati/opisati algoritam koji za učitane koeficijente jednacina dve prave (oblika: y = ax + b) ispisuje na izlaz njihov presek ili odgovarajucu poruku ako su prave paralelne ili se poklapaju.



pseudo-jezik:

Algoritam Max2Broja(x, y)
input: x, y /*broj sa ulaza*/
output: max
{

if (y>x) max=y
else max=x

output: max

}

Ciljevi tokom ovog kursa: konstruisati algoritam koji je *KOREKTAN* i *EFIKASAN*.

II Korektnost

Potrebno je za svaki algoritam pokazati da *uvek* kao rezultat vraća očekivani izlaz za sve dozvoljene ulazne parametre formulisanog problema.

Na primer:
kod sortiranja izlaz mora biti neopadajući poredak niza A, čak i ako niz A
(1) već jeste sortiran, ili
(2) sadrži elemente koji se ponavljaju..

Šta mislite, nad koliko test primera je potrebno propustiti program da bi proverili da je on korektan? Da li je dovoljno 2, 10, 1000 test primera?

Primeri konstrukcije sa algoritama dokazom korektnosti

1. Napisati algoritam za određivanje najvećeg zajedničkog delioca dva prirodna broja i dokazati korektnost napisanog algoritma. (napomena a mod b = ostatak pri celobrojnom deljenju a sa b)

REŠENJE:
Lema. NZD(a, b) = NZD(b, r), r = a mod b

Dokaz:
$a = q \cdot b + r$, NZD(b, r) | b, r => NZD(b, r) | a
Dalje, NZD(b, r) | a, NZD(b, r) | b => NZD(b, r) | NZD(a, b)

Slicno,
$r = a - q \cdot b$, NZD(a, b) | a, b => NZD(a, b) | r
Dalje, NZD(a, b) | b , NZD(a, b) | r => NZD(a, b) | NZD(b, r)

dakle,
NZD(b, r) | NZD(a, b) , NZD(a, b) | NZD(b, r) => NZD(a, b) = NZD(b, r)
sto je i trebalo dokazati.

```
Algoritam NZD(m,n);  /* napisan u PSEUDO jeziku*/
input: m,n; output: nzd;
 begin
 a:=max(m,n);
 b:=min(m,n);
 r:=b;

while r>0 do
begin
 r:=a mod b;
 a:=b;
 b:=r;
end;

nzd:=a;
end.
```

Dokaz korektnosti algoritma izvodimo pomoću invarijante petlje principom matematičke indukcije.
Invarijanta petlje je relacija izmedju promenljivih koja važi nakon svakog izvršenja bloka naredbi u okviru petlje.

Invarijanta petlje u ovom algoritmu je nzd(m, n) = nzd(a, b).
Baza: Pre ulaska u petlju tvrđenje važi, jer je nzd(a, b) = nzd(max(m, n), min(m, n)) = nzd(m, n).
IH:
Pretpostavimo da tvrdjenje važi pre nekog izvršenja bloka naredbi u okviru petlje i dokažimo da važi i

nakon tog izvršenja. Dakle, pretpostavimo da pre nekog izvršenja bloka naredbi u okviru petlje važi nzd(m, n) = nzd(a, b).
Nakon izvršenja bloka naredbi u okviru petlje, promenljiva a dobija vrednost a' = b, a promenljiva b vrednost b' = a mod b.
Kako vazi nzd(a', b') = nzd(b, a mod b) = nzd(a, b) i kako je, na osnovu induktivne pretpostavke nzd(m, n) = nzd(a, b), sledi nzd(m, n) = nzd(a', b'), sto je i trebalo dokazati.
Nakon svakog izvrsenja bloka naredbi u okviru petlje promenljiva r ima strogo manju vrednost nego pre njega (jer je r' = a mod b = a mod r < r).
Dakle niz vrednosti promenljive r je strogo opadajuci niz nenegativnih celih brojeva, pa nakon konacnog broja koraka vrednost promenljive r dostici ce vrednost 0 i tada algoritam zavrsava sa radom.
Na osnovu svojstava invarijante, tada vazi nzd(m, n) = nzd(a, b) = nzd(a, 0) = a.
Dakle, nakon izvrsenja petlje promenljiva a ima vrednost nzd(m, n), pa tu vrednost nakon komande nzd:=a ima i promenljiva nzd, sto je i trebalo dokazati.

2. (za domaći) Neka je f funkcija koja prirodni broj n preslikava u prirodni broj sa istim ciframa, ali u obrnutom poretku (npr: f(7893) = 3987). Konstruisati algoritam koji za ulaznu vrednost - prirodan broj n, izračunava vrednost f(n). Dokazati korektnost napisanog algoritma.

III Efikasnost

Da li je dovoljno da naš program radi korektno? Da li je potrebno da bude i efikasan?

Efikasnost se meri potrošnjom računarskih resursa pri izvršavanjem programa, a najznačajniji su vreme i prostor

Vreme izvršavanja

zavisi od sledećih faktora:

1. skupa mašinskih instrukcija računara na kom se algoritam izvršava, vremena njihovog trajanja u ciklusima, kao i trajanja jednog ciklusa

   Na primer: i=i+1; i++; i+=1;

2. kvaliteta mašinskog kôda generisanog od strane prevodioca

   Na primer: a=3; b=a+3; ILI a=3; b=6;

3. ulaznih podataka
4. vremenske složenosti algoritma

Faktori 1 i 2 zavise od konkretnog računara i prevodioca i ne odražavaju suštinu algoritma.

Vremenska složenost algoritama (svojstvo 4) se meri u odnosu na prebrojani broj upotrebljenih koraka.

Zadatak 3: Prebrojavanje koraka
Svaku od osnovnih operacija (sabiranje, oduzimanje, mno¶ zenje, deljenje, povecanje za jedan, dodela, poredenje, siftovanje) cemo radi jednostavnosti smatrati jednim korakom.
Odrediti vreme izvršavanja sledećeg programskog fragmenta. Rešenje prikazati polinomijalnim izrazom i u O notaciji.
for (i=1; i<=n; i++)
suma = suma + i;

Rešenje:

i=1 izvršava se 1 korak
i<=n izvršava se n+1 put (n puta je ispunjen uslov ulaska u petlju, ali n+1. put nije - uslov se ipak proverava)
i++ izvršava se n puta (onoliko puta koliko se uđe u petlju)
suma = suma + i izvršava se n puta, sadrži 2 operacije, sabiranje i dodelu, pa sadrži ukupno 2n koraka
Dakle ukupno: 1 + n+1 + n + 2n = 4n + 2 koraka, što je u O notaciji O(n).

IV RAM model

Analiza performansi algoritama u računarstvu se može odvijati i nezavisno od analize performansi hardvera ili konkretnog programskog jezika, iako kompletna analiza pretenduje da diskutuje i o uticaju tih parametara.

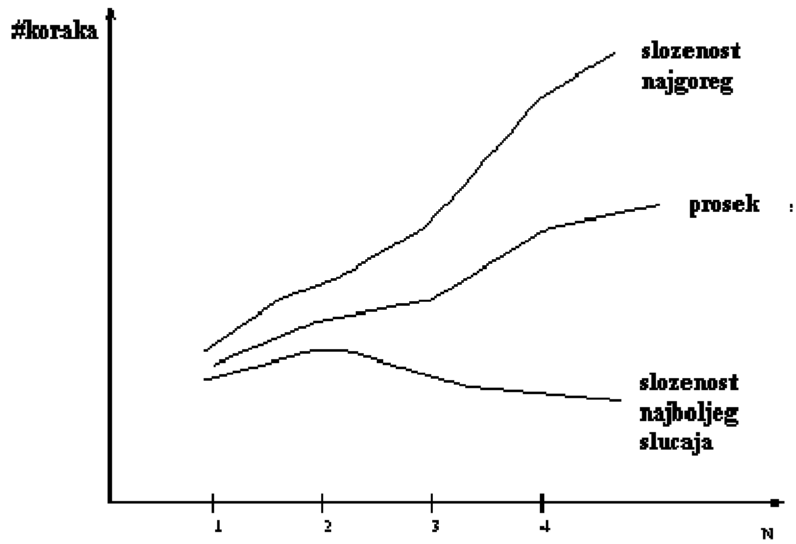
Za potrebe ALGORITMIKE se pri konstrukciji algoritama i diskusiji o njihovim osobinama koristi RAM model:

- svaka "elementarna" operacija (+, -, =, if, višestruka selekcija) traje tačno 1 korak.
- Petlje i pozivi potprograma *nisu* elementarne operacije i zavisne su od veličine podataka sa kojima rade i od sopstvenih podkoraka. Npr. sortiranje nije 1-koračna operacija.
- Pristup memoriji se može uzeti za 1-koračnu operaciju.

Gore navedene pretpostavke važe sve dok se ne kaže drukčije.

V Najbolji, najgori i prosečan slučaj

Složenost najgoreg slučaja algoritma jeste funkcija definisana u odnosu na najveći broj potrebnih koraka za obradu ulaza dimenzije *n*.



Složenost najboljeg slučaja algoritma jeste funkcija definisana u odnosu na najmanji broj potrebnih koraka za obradu ulaza dimenzije *n*.

Složenost prosečnog slučaja algoritma jeste funkcija definisana u odnosu na prosečni broj potrebnih koraka za obradu ulaza dimenzije *n*.

Zadatak 4. Odredite vremensku složenost najboljeg i najgoreg slučaja datog fragmenata programskog kôda u C-u. Broj koraka prikazati u formi polinomijalnog izraza i u O notaciji.

```
for (j=0;j<n;j+=2)
   if (a[j]==b) { printf(“%d ”,j);  break; }
```