Osnovni algoritamski postupci

Često korišteni algoritmi u programiranju

Primjer 1

Algoritam učitava n brojeva i ispisuje njihov zbroj.

Broj n korisnik odabire na početku.

Rješenje:

```
{
ulaz(n);
zbroj:=0;
za broj_ponavljanja:=1 do n činiti
{
ulaz(broj);
zbroj:=zbroj+broj;
}
izlaz(zbroj);
}
```

*Zbrajanje se svodi na dodavanje novoučitanoga broja prethodno izračunatoj vrijednosti varijable **zbroj** u svakom ponavljanju petlje.*
Očito je da ćemo ovaj algoritam koristiti kad zbrajamo veći broj podataka ili kad je broj podataka unaprijed nepoznat.
U ovom slučaju odabrali smo petlju sa zadanim brojem ponavljanja jer na početku programa korisnik određuje koliko će podatka unositi.

Primjer 2

Algoritam učitava ocjene iz informatike za n učenika.

Ispisuje srednju ocjenu pozitivno ocijenjenih učenika.

Rješenje:

```
{
ulaz(n);
zbroj:=0;
br:=0;
za broj_ucenika:=1 do n činiti
{
ulaz(ocjena);
ako je ocjena >1 onda
{
zbroj:=zbroj+ocjena;
br:=br+1;
}
}
izlaz(zbroj/br);
}
```

U ovom smo primjeru morali provjeriti je li učitana ocjena pozitivna i tek smo je tada zbrajali.

Za izračunavanje srednje ocjene morali prebrojati i koliko je pozitivnih ocjena (varijabla br se za svaku pozitivnu ocjenu poveća za 1).

I zbroj i brojač prije petlje postavljamo na početne vrijednosti nula.

Primjer 3

Algoritam provjerava je li učitani broj prost broj (broj koji je djeljiv samo s 1 i sa samim sobom).

Rješenje:

```
{
ulaz(broj);
br:=0;
za i:=2 do round(sqrt(broj)) činiti
ako je broj mod i=0 onda br:=br+1;
ako je br>0 onda izlaz('Nije prost broj.')
inače izlaz('Broj je prost broj.');
}
```

Poboljšanje algoritma bilo bi u korištenju petlje s uvjetom. Primjerice, u slučaju parnoga broja već bismo u prvom ponavljanju ustanovili da postoji djelitelj te da broj nije prost.

Ovo poboljšanje bolje se uočava kod velikih brojeva (može se bitno smanjiti broj ponavljanja petlje).

```
{
ulaz(broj);
i:=1;
br:=0;
ponavljati
i:=i+1;
ako je broj mod i=0 onda br:=br+1;
do i=round(sqrt(broj)) ILI br>0;
ako je br>0 onda izlaz('Nije prost broj.')
inače izlaz('Broj je prost broj.');
}
```

Primjer 4

Algoritam ispisuje najmanji broj u nizu učitanih n brojeva.

Rješenje:

```
{
ulaz(n);
ulaz(min);
za i:=2 do n činiti
{
ulaz(broj);
   ako je broj <min onda
   min:=broj;
}
izlaz(min);
}
```

učitavamo koliko će biti brojeva

prvu učitanu vrijednost spremimo u varijablu u kojoj pamtimo najmanju

učitavamo sljedeći broj

Ako je manji od zapamćenog minimuma, on postaje najmanji.

Primjer 5

Algoritam provjerava je li učitani broj n savršen broj

(jednak je zbroju svojih djelitelja koji su manji od njega, npr. 6=1+2+3 ili 28=1+2+4+7+14).

Rješenje:

```
{
ulaz(n);
z:=0;
za i:=1 do n-1 činiti
ako je n mod i=0 onda
   z:=z+i;
ako je z=n onda
   izlaz('Savršeni broj.')
inače izlaz('Nije
   savršeni broj.');
}
```

budući zbroj znamenaka postavljamo na početnu vrijednost

Ako pronađemo broj kojim je n djeljiv, dodajemo ga u zbroj djelitelja

Primjer 6

Algoritam ispisuje n članova Fibonaccijevog niza.

Fibonaccijev niz je niz u kojemu je svaki sljedeći član zbroj prethodnih dvaju članova (1, 1, 2, 3, 5, 8, 13, …).

Svaki **n-ti** član niza izračunava se po formuli Fn=Fn-1 +Fn-2, tj. izračunava se kao zbroj prethodnih dvaju članova.

Prvi i drugi član niza jednaki su 1.

Rješenje:

```
{
ulaz(n);
f1:=1;
f2:=1;
izlaz(f1,f2);
za i:=3 do n činiti
{
f:=f1+f2;
izlaz(f);
f1:=f2;
f2:=f;
}
}
```

zanemarili smo mogućnost da je n=1 ili n=2

Primjer 7

Treba primijeniti Euklidov algoritam u traženju najvećega zajedničkog djelitelja dvaju zadanih brojeva.

Euklidov algoritam bazira se na pretpostavci da je najveći zajednički djelitelj dvaju brojeva **u** i **v** (u > v) jednak najvećem zajedničkom djelitelju brojeva **u–v** i **v**.

Npr. NZD(15,12) = NZD(12,3) = NZD(9,3) = NZD(6,3) = NZD(3,3) = NZD(0,3) =3.

Jednostavnim smanjivanjem većega broja u paru brojeva dolazimo do sve jednostavnijih parova. Algoritam se prekida kada prvi broj postane nula. U tome trenutku drugi broj predstavlja najveću zajedničku mjeru početnih brojeva. Važno je primijetiti da se oduzima uvijek manji broj od većega, tako da je u algoritmu bitno osigurati provjeru koji je od brojeva veći te im zamijeniti mjesta ako nisu u dobrom poretku.

Rješenje:

```
{
ulaz(a,b);
dok je a>0 činiti
{
ako je a<b onda
{
t:=a;
a:=b;
b:=t;
}
a:=a-b;
}
izlaz(b);
}
```

Zaključak:

Prilikom rješavanja nekog problema uvijek moramo razmišljati u smjeru poboljšanja i ubrzanja algoritma.