Iterativni i rekurzivni postupci

Uvod

- Sastavni delovi svakog programa su **algoritmi** i **strukture podataka**
- **Algoritam** je diskretan, jednoznačan, čisto mehanički postupak, koji je definisan konačnim brojem pravila i koji dovodi do rešenja svakog konkretnog problema iz nekog skupa problema u konačno mnogo koraka
- Reč “algoritam” potiče od latiniziranog imena arapskog matematičara Muhameda ibn-Muse al-Hvarizmija koji je živeo u IX veku i između ostalog napisao knjigu o “računu Indijaca”, tj. o računanju sa arapskim ciframa
- Prvobitno je reč “algoritam” označavala pravila za sabiranje, oduzimanje, množenje i deljenje, a kasnije se pojam algoritma proširio da obuhvata sve mehaničke postupke

Uvod

- Jedini sačuvani latinski prevod odlomka te knjige počinje rečima “Dixit Algorismi...” (Algorizmi je rekao...), po čemu je i današnji pojam algoritam dobio ime

- Svi programi koji se izvode na računarima su realizacija algoritama - da bi se problem mogao rešiti na računaru mora da bude algoritamski rešiv

- Ukoliko neki problem ima algoritamsko rešenje, tada obično ima više različitih načina za njegovo rešavanje pa se prirodno postavlja pitanje koji je od mogućih algoritama za rešavanje problema bolji

Uvod

Kriterijumi po kojima se algoritmi razlikuju, i po kojima se može vršiti izbor pravog algoritma za dati problem, su sledeći:

- **Korektnost:** algoritam treba da ispravno reši postavljeni problem. Ovaj uslov zahteva da problem bude dovoljno precizno postavljen da bismo uopšte bili u stanju da analiziramo da li je problem rešen korektno
- **Količina memorije** potrebna za rešavanje problema. Ovaj kriterijum vremenom gubi na značaju jer je memorija postala dovoljno jeftina, tako da se često ne isplati gubiti vreme na skraćivanje programa ili optimizaciju potrošene memorije na uštrb jasnoće koda
- **Jasnoća i jednostavnost.** Ovaj kriterijum sve više dobija na značaju, jer se danas potencira zahtev da se problem što pre reši na što jasniji način, te da to rešenje bude što shvatljivije i lako za izmene i prilagođavanja
- **Efikasnost.** Pretpostvalja se da je efikasniji algoritam bolji, a da bi se definisalo šta se pod efikasnim algoritmom podrazumeva, uvodi se pojam složenosti algoritma. Kao mera složenosti uzima se broj koraka koji je potrebno izvesti da bi se do rešenja došlo. Broj koraka za rešavanje jednog problema najčešće zavisi i od veličine početnih podataka pa se broj koraka izražava kao funkcija veličine polaznih podataka

Rekurentni nizovi jedne promenljive

- U mnogim oblastima primene računara javlja se potreba za izračunavanjem elemenata matematičkog niza
- Jedna vrsta matematičkih nizova pogodnih za izračunavanje su rekurentni nizovi, tj. nizovi $\{s_n\}$, gde je n-ti element niza, s_n, definisan preko prethodnih elemenata niza (s_{n-1}, s_{n-2}, ...)
- Posmatraćemo prvo najjednostavniji slučaj kada je element niza definisan preko tačno jednog prethodnog:

$$s_n = \begin{cases} a, & n = 0 \\ f(s_{n-1}), & n \geq 1 \end{cases}$$

  gde funkcija f može da ima još neke parametre koji zavise od n
- Zbog toga što elementi niza zavise od prethodnih kaže se "**rekurentni**", a za gornji slučaj pošto element zavisi od tačno jednog prethodnog kaže se "**jedne promenljive**"

Rekurentni nizovi jedne promenljive

- Indekse (matematičkog) niza predstavićemo Javinim tipom `int`, a elemente tipom `double`
- Funkciju *f* za računanje sledećeg elementa niza predstavićemo metodom:
  `static double f(double stari, int n)`
- Metod `f` će za dati indeks `n` i prethodni element niza s_{n-1} prosleđen u parametru `stari` vratiti izračunatu vrednost novog elementa s_n
- Moguće je da metod `f` osim indeksa `n` koristi i još neke dodatne parametre koji su zbog jednostavnosti izostavljeni

Rekurentni nizovi jedne promenljive

- Uz navedene pretpostavke dobija se sledeći metod za računanje elemenata niza:

```
static double s(int n, double pocetnaVrednost) {
  double tekucaVrednost = pocetnaVrednost;
  for (int i = 1; i <= n; i++) {
    tekucaVrednost = f(tekucaVrednost, i);
  }
  return tekucaVrednost;
}
```

Rekurentni nizovi jedne promenljive

- Sledećim programskim fragmentom ispisuje se vrednost izračunatog elementa niza:

```
System.out.print("Unesite n: ");
int n = Svetovid.in.readInt();
System.out.print("Unesite pocetnu vrednost: ");
double a = Svetovid.in.readDouble();
System.out.println("s(n) = " + s(n, a));
```

Primer: niz parnih prirodnih brojeva

- Niz parnih prirodnih brojeva može se definisati na sledeći način:

$$s_n = \begin{cases} 0, & n = 0 \\ s_{n-1} + 2, & n \geq 1 \end{cases}$$

- Lako se vidi da je $s_0 = 0$, $s_1 = 2$, $s_2 = 4$, ...

- Kad se primeni opisana metodologija dobija se sledeći program za računanje *n*-tog elementa ovog niza

Primer: niz parnih prirodnih brojeva

```
class Parni {
  static double f(double stari, int n) {
    return stari + 2;
  }
  static double s(int n, double pocetnaVrednost) {
    double tekucaVrednost = pocetnaVrednost;
    for (int i = 1; i <= n; i++) {
      tekucaVrednost = f(tekucaVrednost, i);
    }
    return tekucaVrednost;
  }
  public static void main(String[] args) {
    System.out.print("Unesite n: ");
    int n = Svetovid.in.readInt();
    System.out.println("s(n) = " + s(n, 0));
  }
}
```