Algoritmi i strukture podataka

Studijski program softversko inženjerstvo

Računarska tehnika

Matematika i informatika

Ulfeta Marovac

Dizajn i analiza algoritma

- Kod dizajna algoritma posebna pažnja se obraća na:
  - Ispravnost algoritma i
  - Efikasnost algoritma.

Analiza algoritama

- Veoma često postoji više algoritama kojima se može rešiti dati problem.
- Treba postaviti kriterijume za izbor određenog algoritma.
- Jednim od primarnih kriterijuma se smatra **efikasnostili performanse algoritma**
- Efikasnost se obično meri potrošnjom računarskih resursa pri izvršavanju programa (dva osnovna rač. resursa: vreme i prostor)
- Obzirom kako se tehnologija memorija stalno unapređuje, vreme izvršavanja se pojavljuje kao glavni kriterijum performansi algoritma.

Vreme izvršavanja

- Vreme izvršavanja zavisi od sledećih faktora:
  - Skupa mašinskih instrukcija računara na kojem se algoritam izvršava i vremena njihovog trajanja u ciklusima (parametri arhitekture i organizacije) kao i trajanja jednog ciklusa (tehnološki parametar)
  - Kvaliteta mašinskog koda generisanog od strane prevodioca
  - Ulaznih podataka
  - Inherentne vremenske složenosti algoritma

Vremenska složenpost osnovnih algoritamskih konstrukcija

Konstrukcija	Vreme izvršavanja
Naredba serije S; P; Q;	$Ts = T_P + T_Q$
Naredba grananja S; If C then P else Q;	$Ts = T_C + max(T_P, T_Q)$
Naredba petlje S; 1. while C do P; 2. do P while C; 3. for i=j to k do P;	$Ts = n * T_P$ n– najveći broj iteracija petlje

Vremenska složenost osnovnih algoritamskih konstrukcija

- Najvažnije pravilo u ovoj tabeli jeste da je vreme izvršavanja petlje jednako vremenu izvršavanja naredbi u telu petlje pomnoženim sa najvećim mogućim brojem iteracija petlje.
- Primer vremenska složenost ugnježdene petlje je jednaka proizvodu broja iteracija svih petlji sa vremenom izvršavanja naredbe u telu unutrašnje petlje.

  $T(n)=n*n*(1+1)$

```
for i = 1 to n do
   for j = 1 to n do
      if (i < j) then
         swap(a[i,j], a[j,i]);  // jedinična instrukcija
```

Primer najmanji elemant niza

```
// Ulaz:  niz a i njegov broj elemenata n
// Izlaz: indeks najmanjeg elementa niza a
algorithm min(a, n)

   m = a[1];  // najmanji element nađen do sada
   j = 1;     // indeks najmanjeg elementa

   i = 2;
   while (i <= n) do // proveriti sve ostale elemente niza
   {  if (m > a[i]) then  // aktuelni element je manji od
                          // privremeno najmanjeg do sada;
        m = a[i];         // zapamtiti taj manji element,
        j = i;            // kao i njegov indeks

    i = i + 1;     // preći na sledeći element niza
   }
  return j;  // vratiti indeks najmanjeg elementa
```

Primer najmanji elemant niza

- Vreme izvršavanja prethodnog algoritma je
- $T(n)=1+1+1+(n-1)(3+1)+1$
- $T(n)=4+4n-4$
- $T(n)=4n$

Asimptotsko vreme izvršavanja

Prilikom analize algoritama možemo dobiti vrlo komplikovane funkcije za njihova vremena izvršavanja. Na primer, za neki algoritam dobijemo vreme izvršavanja izraženo funkcijom

$$T_1(n) = 10n^3 + n^2 + 30n + 80,$$

a za neki drugi dobijemo

$$T_2(n) = 17n \log n - 23n - 10.$$

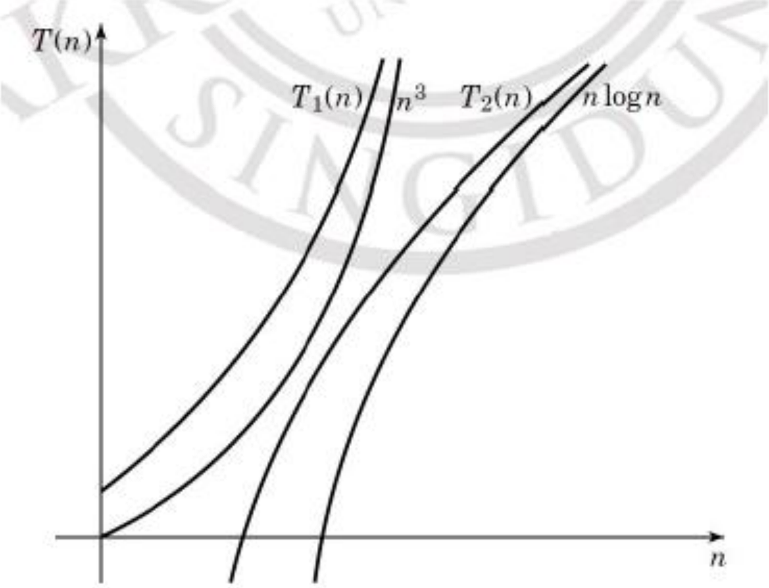
Koji algoritam je brži?

Asimptotsko vreme izvršavanja

- Na osnovu grafika vidljivo je da je funkcija T_1 reda veličine n^3 dok je T_2 reda veličine nlog(n).
- Stoga je drugi algoritam brži.

$$T_1(n) = 10n^3 + n^2 + 30n + 80,$$

$$T_2(n) = 17n \log n - 23n - 10.$$



Asimptotsko vreme izvršavanja

- Za vreme izvršavanja nekog algoritma uzimamo jednostavnu funkciju koja za velike vrednosti njenog argumenta najbolje aproksimira tačnu funkciju vremena izvrsavanja tog algoritma.
- Približno vreme izvršavanja se naziva **asimpotosko** vreme izvršavanja..
- Jednostavna funkcija za vreme izvršavanja se uzima bez konstanti.
- Algoritam A_1 je najsporiji, A_2 srednji, A_3 najbrži.

Primer.

Neka su A_1, A_2, A_3 tri algoritma čija su vremena izvršavanja data funkcijama $2^n, 5n^2$ i $100n$, tim redom, gde je n veličina ulaza.

n	2^n	$5n^2$	$100n$
1	2	5	100
10	1024	500	1000
100	2^{100}	50000	10000
1000	2^{1000}	$5 \cdot 10^6$	100000

n	$A1$	$A2$	$A3$
1	$1\,\mu s$	$5\,\mu s$	$100\,\mu s$
10	$1\,ms$	$0{,}5\,ms$	$1\,ms$
100	$2^{70}\,g$	$0{,}05\,s$	$0{,}01\,s$
1000	$2^{970}\,g$	$5\,s$	$0{,}1\,s$

Vreme izvršavanja

Algoritamski fragment koji matricu dimenzije $n \times n$ inicijalizuje da bude jedinična matrica

```
1 for i=1 to n do
2    for j=1 to n do
3        a[i,j]=0;
4 for i=1 to n do
5    a[i,j]=1;
```

- Vreme izvršavanja T(n) ovog fragmenta jednako je zbiru vremena izvršavanja dve petlje, prve for petlje u prvom redu i druge for petlje u cetvroom redu.
- Pošto se telo unutrašnje petlje u trećem redu izvršava za konstantno vreme, a broj iteracija spoljašnje i unutrašnje petlje jednak je n, ukupno vreme izvrsavanja for petlje u prvom redu je neka konstanta (koju možemo zanemariti) pomnožena sa n^2.
- Vreme izvrsavanja druge for petlje u četvrom redu je reda n pa je ukupno vreme izvršavanja T(n) reda $n^2 + n$.
- Ukupno vreme izvrsavanja T(n) je zapravo n^2 jer n^2 dominira funkcijom n za veliko n, tj. $T(n) = O(n^2)$.

Primer: binarna pretraga sortiranog niza

- Opšti problem pretrage niza je da, za dati niz od n neuređenih brojeva 1; 2; ...; n i dati broj x, treba odrediti da li se broj x nalazi u nizu . Ukoliko je to slučaj, rezultat treba da bude indeks niza i takav da je x = i ; u suprotnom slučaju, rezultat treba da bude 0.
- Dati niz *a* je niz brojeva a_1, a_2, ... ,a_n.
- Telefonski imenik.

```
// Ulaz: sortiran niz a, broj elementa n niza a, trazeni broj x
// Izlaz: k takvo da je x=a_k ili 0 ako se x ne nalazi u nizu a
algorithm bin-search(a, n, x)
  i=1; j=n;   //oblast pretrage je ogranicen indeksima i i j
  while (i<=j) do
       k=(i+j)/2;   //indeks srednjeg elementa
       if (x<a[k]) then
          j=k-1;     //x se nalazi u prvoj polovini oblasti pretrage
       else if (x>a[k]) then
           i=k+1;  //x se nalazi u drugoj polovini oblasti pretrage
       else
           return k;  //x je nadjen
   return 0;         // x nije nadjen
```

Primer: binarna pretraga sortiranog niza

- Da bismo odredili vreme izvršavanja T(n) bin-search, primetimo da je dovoljno odrediti vreme izvršavanja njegove while petlje. Ostali delovi tog algoritma izvršavaju se za konstantno vreme.

- Vreme izvršavanja whlie petlje je proporcionalno broju iteracije te petlje, jer se telo te petlje izvršava za konstantno vreme.

- Najveći broj iteracija while petlje izvršava se kada se broj x ne nalazi u nizu .

Primer: binarna pretraga sortiranog niza

- Sustina binarne pretrage ogleda se u tome da je dužina oblasti pretrage početno n i u svakom saledećem koraku je duplo manja.
- Ako m označava ukupan broj iteracija whlie petlje kada se broj x ne nalazi u nizu , onda:
  - na početku prve iteracije, dužina oblasti pretrage iznosi n;
  - na početku druge iteracije, dužina oblasti pretrage iznosi otprilike $n/2 = n/2^1$;
  - na početku treće iteracije, dužina oblasti pretrage iznosi otprilike $(n/2)/2 = n/4=n/2^2$;
  - na početku četvrte iteracije, dužina oblasti pretrage iznosi otprilike $(n/4)/2 = n/8 = n/2^3$;

  i tako dalje...
  - na pocetku poslednje m-te iteracije, duzina oblasti pretrage iznosi otprilike $n/2^{(m-1)}$:

Primer: binarna pretraga sortiranog niza

- Oblast pretrage na početku poslednje iteracije je 1 što znači da je

$$\frac{n}{2^{(m-1)}} = 1$$

$$n = 2^{(m-1)}$$

$$m - 1 = \log_2 n$$

$$m = 1 + \log_2 n$$

Tipične funkcije za vremena izvršavanja algoritama

Funkcija	Neformalno ime
1	konstantna
log n	logaritamska
n	linearna funkcija
n log n	linearno-logaritamska
n^2	kvadratna funkcija
n^3	kubna funkcija
2^n	eksponencijalna funkcija

O-zapis

- O-zapis se koristi za precizno definisanje pojma da je neka funkcija manja od druge.

Definicija 1. (**O-zapis**) Za dve nenegativne funkcije $f, g : \mathrm{N} \rightarrow \Re^{+}$ kažemo da je $f(n) = O(g(n))$ ako postoje pozitivne konstante c i n_0 takve da za svako $n \geq n_0$ važi nejednakost

$$f(n) \leq c * g(n).$$

Složenost algoritama

- Aloritmi se mogu svrstati u sledeće grupe po rastućoj složenosti:

O(1)-konstanti algoritmi čija složenost ne zavisi od ulaznih podataka su najpoželjniji algoritmi, ali su najređi. Umetanje na početak ulančane liste se izvodi u konstantnom vremenu bez obzira na broj elemenata liste

O(log n)-logaritamski algoritmi su takođe veoma poželjni zbog vrlo sporog porasta logaritamske funkcije. Osnova algoritma menja složenost samo za konstantan faktor. Primer algoritma logaritamske složenosti je binarno pretrazivanje uređenog niza

O(n)-linearni algoritmi imaju opštu formu ciklusa koji se izvršava n puta i karakteristični su za probleme koji obrađuju sve ulazne podatke. Npr. Sekvencionalno pretrazivanje neuređenog niza

Složenost algoritama

O(n log n)-linearno logaritamski algoritmisu česta klasa algoritama koji su zasnovani na binarnom odlučivanju, polovljenju problema, gde se ipak obrađuju svi podaci

O(n^2)-kvadratni algoritmi obično imaju formu dve ugnježdene petlje dimenzije n. Npr. Direktni metodi sortiranja imaju kvadratnu složenost

O(n^k)-polinomijalni

O(k^n)-eskponencijalni algoritmi, k>1

Implementacija algoritma

- Pri projektovanju ili izboru algoritma treba voditi računa o njegovoj implementaciji jer ona može dosta da utiče na efikasnost.
- Implementacija algoritma je zavisna od programskog jezika i mašine na kojoj se algoritam izvršava
- Detalji implementacije pogotovu utiču na konstantni faktor složenosti i članove nižeg reda koji ne određuju dominantno složenost ali mnogu mnogo da utiču na vreme izvršavanja
- Zato je važno težiti optimizovanoj implementaciji

Implementacija algoritama

- Optimizacija je posebno važna ako se algoritam izvršava veliki broj puta ili ima izuzetno veliku dimenziju.
- Tada se složeni programerski napor svakako vraća kroz smanjenu cenu izvršavanja
- Za algoritme koji se retko izvršavaju ne vredi ulagati veliki napor na perfektnu optimizaciju jer se ne isplati –u tim slučajevima je važnije lako razumevanje, kodiranje i testiranje.

Implementacija algoritama

- Pri izboru algoritma za sortiranje ne treba voditi računa samo o njegovoj opštoj složenosti nego i o tipičnim uslovima u kojima de se on izvršavati
- Ni algoritam sa najboljom složenošdu obično nije superioran u svim uslovima pa može u nekom specifičnom slučaju imati slabije performanse od generalno lošijeg algoritma

Implementacija algoritama

- Na kraju treba obratiti pažnju i na prostornu složenost algoritma
- Prostorna složenost se ogleda u memorijskom prostoru koji algoritam zahteva pri izvršavanju.
- Iako je aspekt prostorne složenosti obično manje važan nego vremenska složenost, poželjnijji su algoritmi koji troše manje prostora
- Čest je slučaj da je zahtevi za manjom vremenskom i prostornom složenošću kontradiktorni i da se uštede u vremenu postižu na račun povedanog korišćenja prostora, i obrnuto.