1 Algoritmi i strukture podataka - 1. dvočas

Postoje 3 mere na osnovu kojih se porede efikasnosti algoritama:

1. najgori mogući slučaj
2. prosečan slučaj
3. brzina izvršavanja na unapred određenom skupu instanci (eng. benchmarks)

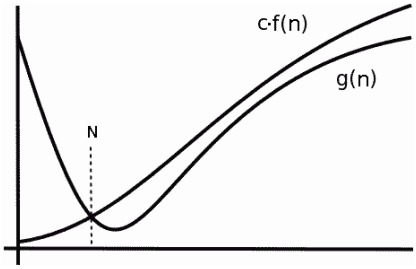
Za merenje u druga 2 slučaja potrebno je imati više informacija o problemu: koji je to prosečan ulaz i koliko se često javlja, kakva je priroda problema i kakve instance problema se najčešće susreću. U prvom slučaju prednost je što se pomoću matematičkog proračuna često lako odredi efikasnost nekog algoritma. Prvi način se pokazao dobar i zato je opšte prihvaćen.

1.1 Oznake O, Ω, Θ

Veliko O notacija koristi se za procenu efikasnosti algoritama.

Definicija 1: Neka su f:N→N i g:N→N proizvoljne funkcije argumenta n. Kažemo da je ***g(n)=O(f(n))*** ako postoje pozitivne konstante c i N tako da $\forall\ n \geq N$ važi: $g(n) \leq c \cdot f(n)$.

Napomena: U profesorovoj knjizi prethodna definicija važi za $\forall\ n > N$.



Slika pokazuje odnos funkcija $g(n)$ i $f(n)$: $g(n)$ **ne raste brže od** $c{\cdot}f(n)$

Oznaka $O(f(n))$ u stvari se odnosi na klasu funkcija, tako da je $g(n) = O(f(n))$ drugi zapis za $g(n){\in}O(f(n))$.

Uočimo i da je $O(1)$ oznaka za klasu ograničenih funkcija.

Takođe važi:
$O(f(n)) + O(g(n)) = O(f(n) + g(n))$
$O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$

Primer: $5n^2 + 8 = O(n^2)$ jer je $5n^2 + 8 \leq 6n^2$ za $n \geq 3$, tj. $c = 6, N = 3$.

Dakle, kada je potrebno dokazati da važi jednakost prethodnog tipa, traže se konstante c i N da bude zadovoljena nejednakost. Prvo se traži c, a potom dovoljno veliko N. Obično postoji veliki (neograničen) broj kombinacija c i N pomoću kojih se može dokazati tvrđenje.

Funkcije se smeštaju u različite klase u zavisnosti od njihove brzine rasta. Prethodni primer govori da funkcija $5n^2 + 8$ ne raste brže od $6 \cdot n^2$. Vidimo da nam multimplikativne konstante ne igraju bitnu ulogu, pa ćemo uvek umesto $O(a \cdot n + b)$ koristiti $O(n)$, gde su a i b proizvoljne konstante (isto važi za $O(n^2), O(\log n)$ itd). Osnova logartima nam nije bitna, što sledi iz jednakosti $\log_a n = log_a(b^{log_b n}) = log_b n \cdot log_a b = log_b n \cdot c$, gde $log_a b$ menjamo oznakom c za konstantu.

Definicija 2: Neka su f:N→N i g:N→N proizvoljne funkcije argumenta n. Kažemo da je $f(n)$ asimptotska donja granica funkcije $g(n)$ i pišemo $g(n) = \Omega(f(n))$ ako postoje pozitivne konstante c i N tako da $\forall n \geq N$ važi: $g(n) \geq c{\cdot}f(n)$.

Definicija 3: Ako za funkcije $f(n)$ i $g(n)$ istovremeno važi $g(n) = O(f(n))$ i $g(n) = \Omega(f(n))$ onda pišemo $g(n) = \Theta(f(n))$.

1. Dokazati da važi: $T(n) = 2n^2 + n - 1 = \Theta(n^2)$
DOKAZ: Potrebno je dokazati da postoje konsante c_1, c_2, N tako da $\forall\ n \geq N$ važi:

$$c_1 n^2 \leq 2n^2 + n - 1 \leq c_2 n^2, \text{ to jest}$$

$$c_1 \leq 2 + \frac{1}{n} - \frac{1}{n^2} \leq c_2$$

Pronalazimo prvo konstante c_1 i N_1 za koje je zadovoljena leva nejednakost, pa konstante c_2 i N_2 za koje je zadovoljena desna nejednakost. Odatle dobijamo c_1, c_2 i N = max{N1,N2}. Jedno rešenje je $c_1 = 2, c_2 = 3, N = 1$.

2. Dokazati da važi:$17n \log_2 n - 23n - 10 = \Theta(n \log_2 n)$
DOKAZ: Potrebno je dokazati da postoje konsante c_1, c_2, N tako da $\forall n \geq N$ važi:

$$c_1 n \log_2 n \leq 17n \log_2 n - 23n - 10 \leq c_2 n \log_2 n, \text{ to jest}$$

$$c_1 \leq 17 - \frac{23}{\log_2 n} - \frac{10}{n \log_2 n} \leq c_2$$

Poslednja nejednakost važi za $c_1 = 4, c_2 = 17, N = 4$. Obrazložiti zašto nejednakost važi.

1.2 Broj koraka

Svaku od osnovnih operacija (sabiranje, oduzimanje, množenje, deljenje, povećanje za jedan, dodela, poređenje, šiftovanje) ćemo radi jednostavnosti smatrati jednim korakom.

Veliko O notacija je nastala za potrebe matematike, ali se danas dosta koristi i u algoritmici da bi se izrazila procena zauzeća memorije ili vreme izvršavanja algoritma. Postavlja se pitanje da li je opravdano koristiti je zbog zanemarivanja multiplikativnih konstanti ali sledeći razlozi govore u prilog njenom korišćenju:

1. jednostavna je za korišćenje
2. konstantni faktori se mogu zanemarivati jer se prilikom konkretne implementacije pseudokoda brojevi koraka menjaju u zavisnosti od toga koji se programski jezik koristi, koji programer piše kod, itd.
3. iako ne oponaša efikasnost na malim dimenzijama, ona sa velikom preciznošću pokazuje efikasnost algoritma kada veličina ulaza (broj n) postane velika. To i jeste interesantno jer se na malim ulazima svi algoritmi brzo izvršavaju.

3. 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)
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 $O(n)$.

4. Odrediti vreme izvršavanja sledećeg programskog fragmenta. Rešenje prikazati polinomijalnim izrazom i u O notaciji.

```
i=0; k=1;
while (k<=n)
{
  k = 2*k;
  i = i+1;
}
```

Rešenje: Broj koraka je $1 + 1 + \lfloor \log_2 n \rfloor + 2 + 4(\lfloor \log_2 n \rfloor + 1) = 8 + 5\lfloor \log_2 n \rfloor$ što je u O notaciji $O(\log n)$ (oznaka $\lfloor$k$\rfloor$ se odnosi na najveći ceo broj manji ili jednak od realnog broja k, npr. $\lfloor 2.12 \rfloor$=2, $\lfloor 3 \rfloor$=3).

1.3 Klase složenosti

Cilj nam je da za svaki problem napišemo što je efikasniji alogritam i procenimo njegovu složenost u O notaciji. Algoritmi se grubo mogu podeliti u sledeće klase složenosti (n je veličina ulaza):

- konstantna složenost ($O(1)$). Ovo znači da algoritam uopšte ne zavisi od dimenzije ulaza n. Idealan slučaj koji se retko javlja u praksi (npr. pristup prvom elementu liste).
- sublinearna složenost (npr. $O(\log n)$, $O(\sqrt{n})$). Ovo je klasa vrlo efikasnih algoritama kod kojih je vreme izvršavanja manje od vremena potrebnog da se pročita ulaz ($O(n)$). Primer je binarna pretraga.
- linearna složenost ($O(n)$). Ovo su efikasni algoritmi. Vreme izvršavanja je u rangu vremena potrebnog da se pročita ulaz. Primer je linearna pretraga.
- superlinearna složenost (npr. $O(n \log n)$, $O(n^2)$). Ovi algoritmi se smatraju dovoljno dobrim za implementaciju na računaru.
- eksponencijalna složenost (npr. $O(2^n)$, $O(n^n)$). Njihova implementacija je nepraktična za veće dimenzije ulaza n.

Pogledati grafike prethodnih funkcija.

1.4 Rekurentne relacije

Detaljno objašnjenje za rešavanje diferencnih jednačina naći u profesorovoj knjizi.

5. Rešiti diferencnu jednačinu $T(n) = 3T(n-1) + 2$ za $n \geq 2, T(1) = 1$

Rešenje: Ovo nije homogena linearna diferencna jednačina pa je moramo transformisati u taj oblik. To ćemo postići oduzimanjem date jednačine od jednačine za računanje n-1. člana niza: $T(n-1) = 3T(n-2)+2$. Dobijamo jednačinu:

$$\text{T(n)-4T(n-1)+3T(n-2)=0}$$

Rešavamo odgovarajuću karakterističnu jednačinu $r^2 - 4r + 3 = 0$ i dobijamo da su rešenja: $r_1 = 3$ i $r_2 = 1$. Rešenje tražimo u obliku: $T(n) = \sum_{i=1}^{k} c_i r_i^n$. U našem slučaju je k=2, pa je rešenje oblika

$$T(n) = c_1 3^n + c_2 1^n = c_1 3^n + c_2$$

Da bismo našli c_1 i c_2 potrebne su nam dve jednačine. Iz početne jednačine dobijamo T(2)=5, pa imamo sistem:

$$3c_1 + c_2 = 1$$

$$9c_1 + c_2 = 5$$

Rešavanjem dobijamo $c_1 = \frac{2}{3}, c_2 = -1$, tj. rešenje je $T(n) = 2 \cdot 3^{n-1} - 1$.

6. Rešiti diferencnu jednačinu $T(n) = 5T(n-1) - 8T(n-2) + 4T(n-3)$ za $n \geq 4$, pri čemu je dato da je T(1)=2, T(2)=4, T(3)=12.

Rešenje: Ovo je već homogena jednačina pa rešavamo karakterističnu jednačinu

$$r^3 - 5r^2 + 8r - 4 = 0.$$

Dobijamo $r_1 = 1, r_2 = 2, r_3 = 2$. Rešenje tražimo u obliku

$$T(n) = c_1 r_1^n + (c_2 + c_3 n) r_2^n$$

(videti knjigu za detaljno objašnjenje), tj.

$$T(n) = c_1 + (c_2 + c_3 n) 2^n.$$

Za ovo su nam potrebne 3 jednačine, koje dobijamo iz uslova T(1)=2, T(2)=4, T(3)=12. Na kraju dobijamo rešenje $T(n) = 4 + (-2 + n)2^n$

7. Odrediti broj koraka sledećeg pseudokoda:
Algoritam Fibonaci (n)
Ulaz ceo broj n
Izlaz n-ti član Fibonačijevog niza

```
{
  if (n > 2)
    return Fibonaci (n-1) + Fibonaci (n-2);
  else
    return 1;
}
```

Ideja rešenja: Postaviti diferencnu jednačinu i rešiti je. Zadatak urađen u profesorovoj knjizi.

NAPOMENA: Lako se može proveriti da li je dobijeno dobro rešenje diferencne jednačine tako što se pomoću rekurentne formule i pomoću rešenja izračunaju npr. $T(2), T(3), T(4), \ldots$ i proveri da li se dobijaju iste vrednosti.

1.5 Zadaci za vežbu

8. Sledećih 9 funkcija poređati u niz $f_1, f_2, ..., f_9$ tako da za svako $i = 1..8$ važi $f_i = O(f_{i+1})$:

(a) $\sqrt{n}$ n 2^n $n \log n$ $n - n^3 + 7n^5$ $n^2 + \log n$ n^2 n^3 $\log n$

(b) $n^{1/3} + \log n$ $(\log n)^2$ $n!$ $\ln n$ $\frac{n}{\log n}$ $\log \log n$ $(\frac{1}{3})^n$ $(\frac{3}{2})^n$ 6.

9. Za svaki par funkcija $f(n)$ i $g(n)$ važi ili $f(n) = O(g(n))$ ili $g(n) = O(f(n))$. Odrediti koja od ove 2 relacije važi i pokazati zašto važi.

(a) $f(n) = \frac{n^2-n}{2}$, $g(n) = 6n$.

(b) $f(n) = n + 2\sqrt{n}$, $g(n) = n^2$.

(c) $f(n) = n + \log n$, $g(n) = n\sqrt{n}$.

10. Odrediti vreme izvršavanja sledećeg programskog fragmenta. Rešenje prikazati polinomijalnim izrazom i u O notaciji.

```
for (i=0; i<n; i++)
   for (j=0; j<n-1; j++){
      pom = i + j;
      s = s + pom;
   }
```

11. Rešiti diferencnu jednačinu $T(n) = 3T(n-1) - 15$ za $n \geq 2, T(1) = 8$.

12. Rešiti diferencnu jednačinu $T(n) = T(n-1) + n - 1$ za $n \geq 2, T(1) = 2$ (Upustvo: 2 puta ponoviti postupak iz zadatka 6).

Rešenje: $T(n) = 2 - \frac{1}{2}n + \frac{1}{2}n^2$.

13. Pokazati da važi:

(a) ako je $T(n) = a_k n^k + \ldots + a_1 n + a_0$ onda je $T(n) = O(n^k)$.

(b) ako je $T(n) = O(n^k)$ onda je $T(n) \neq O(n^{k-1})$.

(c) ako je $T(n) = 2^{10n}$ onda je $T(n) \neq O(2^n)$.

(d) $max(f, g) = \theta(f + g)$