Algoritmi i strukture podataka - 1.cas

Aleksandar Veljković

October 2016

Materijali su zasnovani na materijalima Mirka Stojadinovića

1 Složenost algoritama

- Približna procena vremena ili prostora potrebnog za izvršavanje algoritma
- Konstantni faktori se zanemaruju
- Procena složenosti se vrši u zavisnosti od veličine ulaznih podataka - n

1.1 Odabir reprezentativnog ulaza veličine n za procenu složenosti algoritma

- Najgori slučaj
- Prosečan slučaj
- Benchmarks

1.2 Notacija

O: Neka su $f(n)$ i $g(n)$ pozitivne funkcije, $f(n)$ pripada $O(g(n))$ ako i samo ako postoje konstante c i N > 0 takve da važi:

$$f(n) \leq c \cdot g(n) \text{ za svako } n>\text{N}$$

Pripadnost $f(n)$ klasi O(g(n)) označavamo sa $f(n) = O(g(n))$ ili $f(n) \in O(g(n))$.

Ω: Neka su $f(n)$ i $g(n)$ pozitivne funkcije, $f(n)$ pripada $\Omega(g(n))$ ako i samo ako postoje konstante c i N > 0 takve da važi:

$$f(n) \geq c \cdot g(n) \text{ za svako } n>\text{N}$$

Pripadnost $f(n)$ klasi Ω(g(n)) označavamo sa $f(n) = \Omega(g(n))$ ili $f(n) \in \Omega(g(n))$.

Θ: Neka su $f(n)$ i $g(n)$ pozitivne funkcije, $f(n)$ pripada $\Theta(g(n))$ ako i samo ako postoje konstante c, N_1 i $N_2 > 0$ takve da važi:

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \text{ za svako } n{>}max\{N_1,\, N_2\}$$

Drugim rečima, $f(n)$ pripada $\Theta(g(n))$ ako i samo ako istovremeno $f(n)$ pripada $O(g(n))$ i $\Omega(g(n))$

Pripadnost $f(n)$ klasi Θ(g(n)) označavamo sa $f(n) = \Theta(g(n))$ ili $f(n) \in \Omega(g(n))$.

o: Neka su $f(n)$ i $g(n)$ pozitivne funkcije, $f(n)$ pripada $o(g(n))$ ako i samo ako važi:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

ω: Neka su $f(n)$ i $g(n)$ pozitivne funkcije, $f(n)$ pripada $\omega(g(n))$ ako i samo ako važi:

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

1.3 Klase složenosti

- Konstantna - $O(1)$
- Sublinearna - $O(logn), O(sqrt(n))$...
- Linearna - $O(n)$
- Superlinearna - $O(nlogn), O(n^2)$...
- Eksponencijalna - $O(2^n), O(3^n)$...

Primer 1. Odrediti vreme izvršavanja sledećeg programskog fragmenta. Rešenje prikazati polinomijalnim izrazom i u O notaciji:

```
for(i = 1; i <= n; i++)
    suma = suma + i;
```

Rešenje

```
        1     n + 1   n
for(i = 1; i <= n; i++)
          1      1        x n
    suma = suma + i;
```

- Inicijalizacija $i = 1$ se izvršava samo jednom, pri ulasku u petlju
- Provera ispunjenosti uslova $i <= n$ se izvršava n puta kada je uslov ispunjen i jednom kada uslov nije ispunjen, pri čemu se tada ne ulazi u telo petlje
- Inkrementacija $i++$ se izvršava n puta
- Sabiranje $+$ se izvršava jednom
- Dodela $=$ se izvršava jednom
- Sabiranje i dodela se izvršavaju u okviru svake iteracije petlje (n puta)

Ukupan broj koraka: $1 + n + 1 + n + n \cdot (1 + 1) = 4n + 2$
Dokažimo da je $4n + 2 = O(n)$
Da bi važilo $4n+2 = O(n)$ potrebno je da pronadjemo konstante c i N >0 takve da je:

$$4n + 2 \leq c \cdot n \text{ za svako n } >\text{N}$$

Za $c = 6$ i $N = 1$ nejednakost je ispunjena i time je dokazano da je složenost navedenog programskog fragmenta zaista $O(n)$

Primer 2. Odrediti vreme izvršavanja sledećeg programskog fragmenta. Rešenje prikazati u O notaciji:

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

Rešenje

```
i = 0; 1
k = 1; 1
while(k <= n)          x log2n + 2
{                      x log2n + 1
```

```
        1   1
k = k * 2;
        1   1
i = i + 1;
}
```

Objašnjenje: Inicijalno promenljiva k predstavlja 2^0, uzastopnim množenjem k sa 2, dok se ne dostigne n, potrebno je $\lfloor \log_2 n \rfloor + 1$ množenja, što je ujedno i broj iteracija petlje. Ipak, uslov se proverava još jednom kada nije ispunjen i otud broj provera $\lfloor \log_2 n \rfloor + 2$. Primer za $n = 5$, vrednosti promenljive k pri svakoj proveri uslova su 1, 2, 4 i 8, uslov je ispunjen za vrednosti 1, 2 i 4, pa je broj provera uslova 4, broj iteracija petlje je 3, dok je $\lfloor \log_2 5 \rfloor = 2$.

Ukupan broj koraka: $1 + 1 + \lfloor \log_2 n \rfloor + 2 + 4 \cdot (\lfloor \log_2 n \rfloor + 1) = 5\lfloor \log_2 n \rfloor + 8 = O(logn)$

Osnovu logaritma možemo izostaviti jer se logaritmi istog broja sa razlicitim osnovama razlikuju za konstantni faktor:

$$\log_a n = \frac{\log_b n}{\log_b a} = \frac{1}{\log_b a} \cdot \log_b n \text{ pri čemu je } \frac{1}{\log_b a} \text{ konstantna vrednost.}$$

1.4 Diferencne jednačine

Složenost rekurzivnih algoritama računa se rešavanjem odgovarajućih diferencnih jednačina.

Primer 3. Odrediti rešenje diferencne jednačine $T(n) = 3T(n-1) + 2$, sa početnim uslovom $T(1) = 1$.

Rešenje
Nehomogenu jednačinu svodimo na homogenu

$$T(n) = 3T(n-1)$$

$$T(n-1) = 3T(n-2) + 2$$

$$\text{———————————} -$$

$$T(n) - T(n-1) = 3T(n-1) - 3T(n-2)$$

$$T(n) - 4T(n-1) + 3T(n-2) = 0$$

Formiramo nehomogenu jednačinu

$$t^n - 4t^{n-1} + 3t^{n-2} = 0 \ / \cdot \frac{t^2}{t^n}$$

$$t^2 - 4t + 3 = 0$$

Nule dobijene kvadratne jednačine su $t_1 = 3$ i $t_2 = 1$

Opšte rešenje jednačine tražimo u obliku:

$$T(n) = c_1 \cdot t_1^n + c_2 \cdot t_2^n = c_1 \cdot 3^n + c_2 \cdot 1^n = c_1 \cdot 3^n + c_2$$

Koeficijente c_1 i c_2 tražimo rešavanjem sistema:

$$\text{za } n = 1, T(1) = c_1 \cdot 3 + c_2 = 1$$

$$\text{za } n = 2, T(2) = c_1 \cdot 9 + c_2 = 3T(1) + 2 = 5$$

Rešenja sistema su $c_1 = \frac{2}{3}$ i $c_2 = -1$ pa je rešenje diferencne jednačine:

$$T(n) = \frac{2}{3} \cdot 3^n - 1 = 2 \cdot 3^{n-1} - 1$$

1.5 Primeri računanja vremenske složenosti algoritama za sortiranje

Algoritam: SelectionSort(A)
Ulaz: Niz A:$[a_1, a_2, ..., a_n]$
Izlaz: Niz A':$[a'_1, a'_2, ..., a'_n]$, $a'_i \in A \forall i = 1...n, a'_1 \leq a_2 \ \leq ... \leq a'_n$

```
begin
    for i := 1 to n do
    min_indeks := i
        for j := i + 1 to n do
            if A[j] <= A[min_indeks] then
                min_indeks := j
        tmp := A[i]
        A[i] := A[min_indeks]
        A[min_indeks] = tmp
end
```

Petlja po i izvršava n iteracija. U okviru svake iteracije izvršava se $n - i$ iteracija petlje po j, pa je ukupan broj iteracija:

$$\sum_{i=1}^{n-1} i = 1 + 2 + ... + n - 1 = \frac{n(n+1)}{2} - n = O(n^2)$$

Algoritam: MergeSort(A)
Ulaz: Niz A:$[a_1, a_2, ..., a_n]$
Izlaz: Niz A':$[a'_1, a'_2, ..., a'_n]$, $a'_i \in$ A $\forall i = 1...n, a'_1 \leq a_2 \ \leq ... \leq a'_n$

```
begin
    if n <= 1
        return A
    else
        L := A[a1, ... , an/2]
        R := A[an/2+1, ... , an]
        L := MergeSort(L)
        R := MergeSort(R)
        A' := Merge(L, R)

        return A'
end
```

Algoritam: Merge(L, R)
Ulaz: L:$[l_1, l_2, ..., l_{\frac{n}{2}}]$, R:$[r_1, r_2, ..., r_{\frac{n}{2}}]$, $l_1 \leq l_2 \leq ... \leq l_{\frac{n}{2}}, r_1 \leq r_2 \leq ... \leq r_{\frac{n}{2}}$
Izlaz: Niz A':$[a'_1, a'_2, ..., a'_n]$, $a'_i \in$ L ili $a'_i \in$ R $\forall i = 1...n, a'_1 \leq a_2 \ \leq ... \leq a'_n$

```
begin
    j1 = 1
    j2 = 1
    for i := 1 to n do
        if (j2 > n/2 or L[j1] <= R[j2]) and j1 < n/2 then
            A'[i] := L[j1]
            j1 := j1 + 1
        else if j2 < n/2 then
            A'[i] = R[j2]
            j2 := j2 + 1
    return A'
end
```

Složenost procedure *Merge* je $O(n)$ Diferencna jednačina koja odgovara broju koraka algoritma MergeSort je $T(n) = 2T(\frac{n}{2}) + cn$, gde je c konstanta, pa je složenost MergeSort algoritma $O(nlogn)$.

Dokažimo da je složenost MergeSort algoritma $O(nlogn)$

Početni rekurzivni poziv poziva dva rekurzivna poziva za ulaz veličine $\frac{n}{2}$ i izvršava proceduru Merge u cn koraka. Drvo rekurzije predstavljeno je sledećom slikom:

```
          cn                - cn
       /     \
    cn/2     cn/2           - 2cn/2 = cn
   /    \   /     \
 cn/4 cn/4 cn/4 cn/4        - 4cn/4 = cn
  / \  / \  / \  / \
  . . . . . . . . . .
 c c c c c c c c ... c      - nc = cn
```

Visina stabla rekurzije je $log_2 n + 1$ dok je suma svakog nivoa stabla jednaka cn, pa je ukupna suma celokupnog stabla jednaka $(log_2 n + 1) \cdot cn = cn \cdot log_2 n + cn = O(nlogn)$