AISP-cas2

Božica Kalinić

October 2020

Složenost algoritama

1 Teorijski uvod

Definicija 1.1. Neka su $f, g : \mathbb{N} \to \mathbb{R}^+$. Kaže se da je $g(n) = O(f(n))$, ako $(\exists c, N \in \mathbb{N})(\forall n \in \mathbb{N}) n > N \Rightarrow g(n) < c\dot{f}(n)$.

Teorema 1. Važe tvrdjenja:

1. $O(f(n)) + O(g(n)) = O(f(n) + g(n))$
2. $O(f(n))O(g(n)) = O(f(n)g(n))$

Definicija 1.2. Neka su $f, g : \mathbb{N} \to \mathbb{R}^+$. Kaže se da je $g(n) = \Omega(f(n))$, ako $(\exists c, N \in \mathbb{N})(\forall n \in \mathbb{N}) n > N \Rightarrow g(n) > c\dot{f}(n)$.

Definicija 1.3. Neka su $f, g : \mathbb{N} \to \mathbb{R}^+$. Kaže se da je $g(n) = \Theta(f(n))$, ako važi $g(n) = \Omega(f(n))$ i $g(n) = O(f(n))$.

Teorema 2. Ako za dati algoritam A funkcija T(n) opisuje broj operacija za ulaz veličine n, pri čemu važi da je zadata vrednost $T(1)$ i

$$T(n) = a \cdot T(\frac{n}{b}) + cn^k, a, b, c, k \geq 0, b \neq 0,$$

tada važi:

$$T(n) = \begin{cases} O(n^{\log_b a}), \ a > b^k \\ O(n^k \log n), \ a = b^k \\ O(n^k), \ a < b^k \end{cases}$$

Definicija 1.4. Analiza ukupne dužine trajanja većeg broja operacija se naziva *amortizovana analiza složenosti*. Amortzovana cena izvršavanja n operacija je prosek vremena izvršavanja tih operacija.

2 Zadaci

1. Proceniti složenost Euklidovog algoritma za nalaženje NZD-a dva prirodna broja.

2. Proceniti složenost ubacivanja n elemenata u dinamički niz.

3. Iz skupa $\{0, 1, ..., n\}$ je izbačen jedan element. Proceniti složenost algoritma koji odredjuje koji je element izbačen.

4. Iz skupa $\{0, 1, ..., n\}$ su izbačena dva elementa. Proceniti složenost algoritma koji pronalazi koja dva elementa fale.

5. Za dati segment prirodnih brojeva $[a, b]$ i prirodan broj k, napraviti algoritam koji pronalazi koliko brojeva iz segmenta je deljivo brojem k.

6. Konstruisati algoritam koji izračunava n-ti Fibonačijev broj.

7. Tačna perioda niza dužine n, je prirodan broj p tako da $p \mid n$, i da niz predstavlja periodično ponavljanje njegovih privih p elemenata.
   Za dati niz dužine $2^k, k \geq 0$, konstruisati algoritam koji pronalazi najmanju tačnu periodu datog niza i proceniti složenost takvog algoritma.

3 Rešenja

1. Proceniti složenost Euklidovog algoritma za nalaženje NZD-a dva prirodna broja.
   Pretpostavimo da je složenost deljenja dva broja O(1).
   Algoritam NZD_1**:**
   Ulaz:
   a, b (prirodni brojevi)

   Izlaz:
   c (Najveći zajednički delilac brojeva a i b)
   begin
   if $a > b$ do
   return $NZD_1(a - b, b)$;
   else if $a < b$ do
   return $NZD_1(a, b - a)$;
   else do
   return a;

   end
   Da li se algoritam zaustavlja?
   Na početku su a i b prirodni brojevi. U svakom slučaju rekurzivno pozivamo funkciju sa argumentima koji ostaju u skupu prirodnih brojeva, jer uvek od većeg broja oduzimamo manji.
   Pri svakom rekurzivno pozivu umanjujemo jedan argument.
   Time smo ograničili broj rekurzivnih poziva, tj. algoritam se zaustavlja.

U najgorem slučaju, ili a ili b su jedan. Tada nastaje barem a-1 ili b-1 oduzimanja, tj. $max(a,b)-1$ oduzimanja, pa dobijamo procenu složenosti $O(a+b)$.

Algoritam NZD_2**:**
Ulaz:
a, b (prirodni brojevi t.d. $a > b$)

Izlaz:
c (Najveći zajednički delilac brojeva a i b)
begin
if $b = 0$ do
    return a;
else do
    return $NZD_2(b, a \mod b)$;
end
Primetimo da je prvi argument veći od drugog, jer u rekurzivnom pozivu prosledjujemo b, $a \mod b \in \{0, 1, ..., b-1\}$.
Dokažimo da je $a \mod b < \frac{a}{2}$.

- Ako je $b \leq \frac{a}{2}$, tada je $a \mod b < b \leq \frac{a}{2}$,
- Ako je $b > \frac{a}{2}$, tada je a div $b = 1$, tj. $a \mod b = a - b < a - \frac{a}{2} = \frac{a}{2}$

Time se drugi argument barem dvostuko umanji na svaka dva koraka. Zbog toga je složenost algoritma barem ograničena sa brojem deljenja sa 2 veća od dva argumenata, tj. $O(\log_2(a+b)) = O(\log(a+b))$

2. Razmatramo prvo dinamički niz početnog kapaciteta 0, kod koga se tokom realokacije kapacitet povećava za k. Realokacija se dešava samo ako je popunjen kapacitet niza, tj. ako je njegova dužina jednaka njegovom kapacitetu. Prvo vršimo realokaciju, gde je kapacitet niza k, a dužina 0.
Zatim upisujemo prvih k elemenata.
Nakon toga se vrši realokacija gde tražimo prostor za niz kapaciteta 2k i gde kopiramo prvih k elemenata.
Nakon toga unosimo još k elemenata.
Nakon toga, realociramo niz na prostor u memoriji gde će imati kapacitet od 3k, pri čemu kopiramo prvih 2k elemenata i unosimo još k novih elemenata itd. Operacije prate sledeći niz:
$((k+k)+(2k+k+k)+(3k+2k+k)+(4k+3k+k)+...)$
Realokaciju vršimo oko $\frac{n}{k}$ puta, jer ćemo svaki put povećavati dužinu niz za k. Za svaku realokaciju kopiramo prethodan sadržaj niza i unosimo novih k elemenata, tj. za svaku realokaciju broj operacija jednak je kapacitetu, time će broj operacija biti:
$((k+k)+(2k+k+k)+(3k+2k+k)+(4k+3k+k)+...+(k\frac{n}{k}+k\frac{n}{k})) = \frac{n}{k}(2k+2k\frac{n}{k})/2 = n(1+\frac{n}{k}) = n+\frac{n^2}{k}$. S obzirom da je k konstanta koja ne zavisi od n, dobijamo procenu da je složenost dodavanja n elemenata

u dinamički niz $O(n^2)$, tj. da je amortizovana cena dodavanja jednog elementa u dinamički niz $O(n)$.
Razmotrimo sada geometrijsku varijantu uvećanja kapaciteta dinamičkog niza, tj. dinamičkog niza gde se kapacitet tokom realokacije množi sa faktorom $q > 1$. Neka na početku imamo m elemenata u nizu i želimo da ubacimo n elemenata. Prvo ćemo uneti m elemenata, pa ćemo realocirati niz tako da mu je kapacitet qm i unećemo $mq - m$ elemenata, kopirajući stari sadržaj niza tj. prvih m elemenata. Zatim ćemo alocirati niz kapaciteta q^2m i prekopirati stare elemente kojih ima qm i unećemo preostalih $q^2m - qm$, itd.
Time broj operacija prati sledeći šablon:

$$m + (mq + m + (mq - m)) + (mq^2 + mq + (mq^2 - mq)) + ...$$

Posle k realokacija broj operacija je jednak

$$m+2mq+2mq^2+...+2mq^k = 2m+2mq+2mq^2+...+2mq^k-m = 2m\frac{1-q^{k+1}}{1-q}-m.$$

Ako je $n = mq^k$, tj. $k = \log_q(\frac{n}{m})$, zamenom u formuli dobijamo

$$2m\frac{1-q\frac{n}{m}}{1-q} - m = \frac{2m - 2qn - m + mq}{1-q} = \frac{m - 2qn + mq}{1-q}.$$

Zanemarujući konstante dobijamo da je složenost $O(n)$, čime je amortizovana cena dodavanja elementa $O(1)$.

3. Iz skupa $\{0, 1, ..., n\}$ je izbačen jedan element. Proceniti složenost algoritma koji pronalazi koji element fali.
Razmotrimo nekoliko različitih rešenja koje bi mogle da nam padnu na pamet:
Algoritam $Resenje_1$**:**
Ulaz:
S (skup $\subset \{0, 1, 2, ..., n\}$ veličinice n)
Izlaz:
b (t.d. S $\cup\{b\} = \{0, 1, 2, ..., n\}$)
begin
$b := 0$;
while $b \leq n$ **do**
    $i := 0$;
    $ima := false$;
        while $i < n$ **do**
            if $S[i] = b$ **do**
                $ima := true$;
            $i := i + 1$;
        if $ima = false$ **do**
            return b;

$b := b + 1$;
end
Složenost ovog algoritma je $O(n^2)$.

Dalje, možda bi neko pomislio da možemo prvo da sortiramo skup, što možemo da uradimo algoritmom složenosti $O(n \log n)$, pa da prodjemo redom kroz skup i da vidimo koji element fali, tj.

Algoritam *Resenje*$_2$**:**
Ulaz:
S (skup $\subset \{0, 1, 2, ..., n\}$ veličinice n)

Izlaz:
b (t.d. S $\cup\{b\} = \{0, 1, 2, ..., n\}$)
begin
QuickSort(S, n+1);
$i := 0$
while $i < n$ do
    if $S[i] \neq i$ do
        return i;
    $i := i + 1$;
return n;
end
Ovakav algoritam je složenosti $O(n \log n) + O(n) = O(n \log n)$.

Medjutim ovaj problem možemo rešiti uz Gausovu pomoć ;)
Suma aritmetičkog niza $\{0, 1, .., n\}$ je $\frac{n(n+1)}{2}$. Izračunavanjen razlike prethodne sume i sume datog skupa ($\frac{(n+1)n}{2} - \sum_{i=1}^{n} S[i]$) saznajemo koji element fali.

Algoritam *Resenje*$_3$**:**
Ulaz:
S (skup $\subset \{0, 1, 2, ..., n\}$ veličinice n)

Izlaz:
b (t.d. S $\cup\{b\} = \{0, 1, 2, ..., n\}$)
begin
$i := 0$
$suma := 0$;
while $i < n$ do
    $suma := suma + S[i]$;
    $i := i + 1$;
return $n(n + 1)/2 - suma$;
end
Ovakav algoritam je složenosti O(n).

4. Ideja je slična. Želimo na neka dva različina načina da akumuliramo vrednosti datog skupa i da formiramo sistem od dve jednačine, čijim rešavanjem saznajemo koji brojevi fale. Neka x, y označavaju dva ele-

menta koja nedostaju datom skupu. Tada važe sledeće jednačine:

$$x + y + \sum_{i=1}^{n-1} S_i = \sum_{i=0}^{n} i$$

$$xx + yy + \sum_{i=1}^{n-1} S_i S_i = \sum_{i=0}^{n} ii$$

Daljim izračunavanjem dobijamo sistem
$x + y = \sum_{i=0}^{n} i - \sum_{i=1}^{n-1} S_i$
$x^2 + y^2 = \sum_{i=0}^{n} ii - \sum_{i=1}^{n-1} S_i S_i$
koji se direktno rešava.

Algoritam $Naci\_koji\_elementi\_fale$**:**
Ulaz:
S (skup $\subset \{0, 1, 2, ..., n\}$ veličinice n-1)

Izlaz:
a, b (tako da S $\cup\{a, b\} = \{0, 1, 2, ..., n\}$)
begin
$i := 0$
$suma := 0$;
$kvadrati := 0$
while $i < n - 1$ do
    $suma := suma + S[i]$;
    $kvadrati := kvadrati + S[i]S[i]$;
    $i := i + 1$;
$c_1 = n(n + 1)/2 - suma$;
$c_2 = n(n + 1)(2n + 1)/6 - kvadrati$;
$x = (c_1 + sqrt(2c_2 - c_1c_1))/2$;
$y = c_1 - x$;
return (x, y);
end

5. **Algoritam** $Pokusaj_1$**:**
Ulaz:
a, b, k (prirodni brojevi)

Izlaz:
n (broj prirodnih brojeva iz segmenta [a,b] deljivih sa k)
begin
$n := 0$;
$i := a$;
while $i \leq b$ do
    if $i \mod k = 0$ do
        $n := n + 1$;
    $i := i + 1$;
return n;

end
Ovaj algoritam je složenosti O(b-a).

Algoritam $Pokusaj_2$**:**
Ulaz:
a, b, k (prirodni brojevi)

Izlaz:
n (broj prirodnih brojeva iz segmenta [a,b] deljivih sa k)
begin
$m_1 := a$ div k;
$m_2 := b$ div k;
$r_1 := a$ mod k;
$r_2 := b$ mod k;
if $r_1 = 0$ do
    return $m_2 - m_1 + 1$;
else do
    return $m_2 - m_1$;

end
Ovaj algoritam je složenosti O(1).

6. Pretpostavimo da je sabiranje dva prirodna broja složenosti O(1). **Algoritam** Fib_1**:**
Ulaz:
n (prirodan broj)

Izlaz:
rez (n-ti Fibonačijev broj)
begin
if $n = 0$ do
    return 1;
else if $n = 1$ do
    return 1;
else do
    return $Fib_1(n-1) + Fib_1(n-2)$;
end
Ovakav algoritam je eksponencijalne složenosti.

Primetimo da je ono što algoritam čini računski zahtevnim to što više puta računamo vrednost Fibonačijevog broja za isto n. Primer: $Fib_1(7)$:

- F_6, F_5
- $(F_5, F_4), (F_4, F_3)$
- $((F_4, F_3), (F_3F_2)), ((F_3F_2), (F_2, F_1))$
- $(((F_3, F_2), (F_2, F_1)), ((F_2, F_1)(F_1, F_0))), (((F_2, F_1)(F_1, F_0)), ((F_1, F_0), F_1))$
- $((((F_2, F_1), (F_1, F_0)), ((F_1, F_0), F_1)), (((F_1, F_0), F_1)(F_1, F_0))), ((((F_1, F_0), F_1)$ $(F_1, F_0)), ((F_1, F_0), F_1))$

- $(((((F_1, F_0), F_1), (F_1, F_0)), ((F_1, F_0), F_1)), (((F_1, F_0), F_1)$ $(F_1, F_0))), ((((F_1, F_0), F_1)(F_1, F_0)), ((F_1, F_0), F_1))$

Ideja je da svaku potrebnu vrednost Fibonačijevog niza računamo samo jendom. Zato uvodimo pomoćni niz koji će nam čuvati do sada izračunate vrednosti.

Algoritam Fib_2**:**
Ulaz:
n (prirodan broj)
Pomoćna struktura niz (dinamički niz kapaciteta n)

Izlaz:
rez (n-ti Fibonačijev broj)
begin
$niz[0] = 1;$
$niz[1] = 1;$
$i := 2;$
while $i \leq n$ do
    $niz[i] = niz[i-1] + niz[i-2];$
    $i := i + 1;$
return $niz[n];$
end
Za dinamički niz nam je potrebna alokacija prostora veličine n. Pristupanje i menjanje vrednosti elemenata niza možemo uraditi u složenosti O(1). Time je vredmenska složenost O(n), a prostorna O(n).

Postoji način da rešimo ovaj problem tako da imamo prostornu složenost O(1).

Algoritam Fib_3**:**
Ulaz:
n (prirodan broj)
Izlaz:
rez (n-ti Fibonačijev broj)
begin
$prepre = 1;$
$pre = 1;$
$i := 2;$
while $i \leq n$ do
    $sad = pre + prepre;$
    $prepre := pre;$
    $pre := sad;$
    $i := i + 1;$
return $pre;$
end
Rešavanjem rekurentne jednačine za Fibonačijeve brojeve dobijamo:
$F_n = \frac{\phi^n - (1-\phi)^n}{2}$, $\phi = \frac{1+\sqrt{5}}{2}$.

Pretpostavimo da je složenost množenja dva prirodna broja O(1). Na osnovu ove pretpostavke i prethodnog rešenja rekurentne jednačine možemo konstruisati algoritam za izračunavanje n-tog Fibonačijevog broja u složenosti $O(\log n)$.

Algoritam *Stepen*:
Ulaz:
x (realan broj)
n (prirodan broj)
Izlaz:
x^n
begin
if $n = 0$ do
    return 1;
if $n \mod 2 = 0$ do
    $y =$ Stepen(x, n div 2);
    return $y \cdot y$;
else do
    $y =$ Stepen(x, n div 2);
    return $x \cdot y \cdot y$;
end
Algoritam Fib_4:
Ulaz:
n (prirodan broj)
Izlaz:
F_n (n-ti Fibonačijev broj)
begin
$phi := \frac{1+\sqrt{5}}{2}$;
return $(Stepen(phi, n) - Stepen(1 - phi, n))/2$;
end
Napomena:
U ovom algoritmu postoji problem u zaokruživanju vrednosti transcedentnog broja ϕ na konačan broj cifara, što utiče na preciznost rezultata.

Pouka ovog primera je da ako krenemo od naše početne ideje, koja možda nije optimalna, i pokušamo da razmislimo da li je svaki korak neophodan i da li možemo nešto optimalnije da odradimo, da često možemo da dobijemo bolje rezultate. Teško je osmisliti od jednom najbolje rešenje. Krenite od prve ideje koja vam padne napet i dalje je razgradite. Ili ćete uspeti ili možete da saznate zašto je skupa i na čemu treba da poradite.

7. **Algoritam** $Proba_1$:
Ulaz:
a (niz brojeva)
n (prirodan broj koji predstavlja dužinu niza)
Izlaz:

p (Tačna perioda datog niza)
begin
$p := 1$;
while $p \leq n$ do
    $i := p$;
    $jeste := true$;
    while $i \leq n$ do
        if $a[i] \neq a[i \mod p]$ do
            $jeste := false$;
        $i := i + 1$;
    if $jeste = true$ do
        return p;
    $p := 2p$;
return n;
end
U najgorem slučaju je tačna perioda niza dužina tog niza. Složenost ovakvog algoritma $\mathrm{O}(n \log n)$.

Da li možemo da poboljšamo ovu ideju? Ključno je uočiti da ako je p tačna perioda niza, da je onda i 2p tačna perioda niza. Ovo možemo iskoristiti da bi umesto linearne pretrage po tačnoj periodi probali binarnu pretragu.
Druga ideja je da krenemo od toga da je perioda n i da vidimo da li možemo da je smanjimo.

Algoritam $Proba_2$**:**
Ulaz:
a (niz brojeva)
n (prirodan broj koji predstavlja dužinu niza)
Izlaz:
p (Tačna perioda datog niza)
begin
if $n = 1$ do
    return 1;
else do
    $i := 0$;
    while $i < n/2$ do
        if $a[i] \neq a[i + n/2]$ do
            return n;
        $i := i + 1$;
    return $Proba_2(a, n/2)$;
end
Složenost algoritma zadovoljava diferencnu jednačinu: $T(n) = T(n/2) + n/2 = O(n)$.