ALGORITMI I NJIHOVA KOMPLEKSNOST

Strukture podataka i algoritmi 1

Algoritmi i kompleksnost

- Generalni principi
- Efikasnost
- Detaljan model računara
- Pojednostavljen model računara
- Primeri

Generalni principi

- **Algoritam** je postupak (koji se sastoji od koraka) za rešavanje određenog **problema**.
- Algoritam će se izvršavati na nekom procesoru, što može biti: čovek, mehanički ili elektronski procesor.
- Algoritam se mora sastojati od **koraka** koje je procesor u stanju da izvrši.
- Algoritam mora biti **konačan**.

Još principa

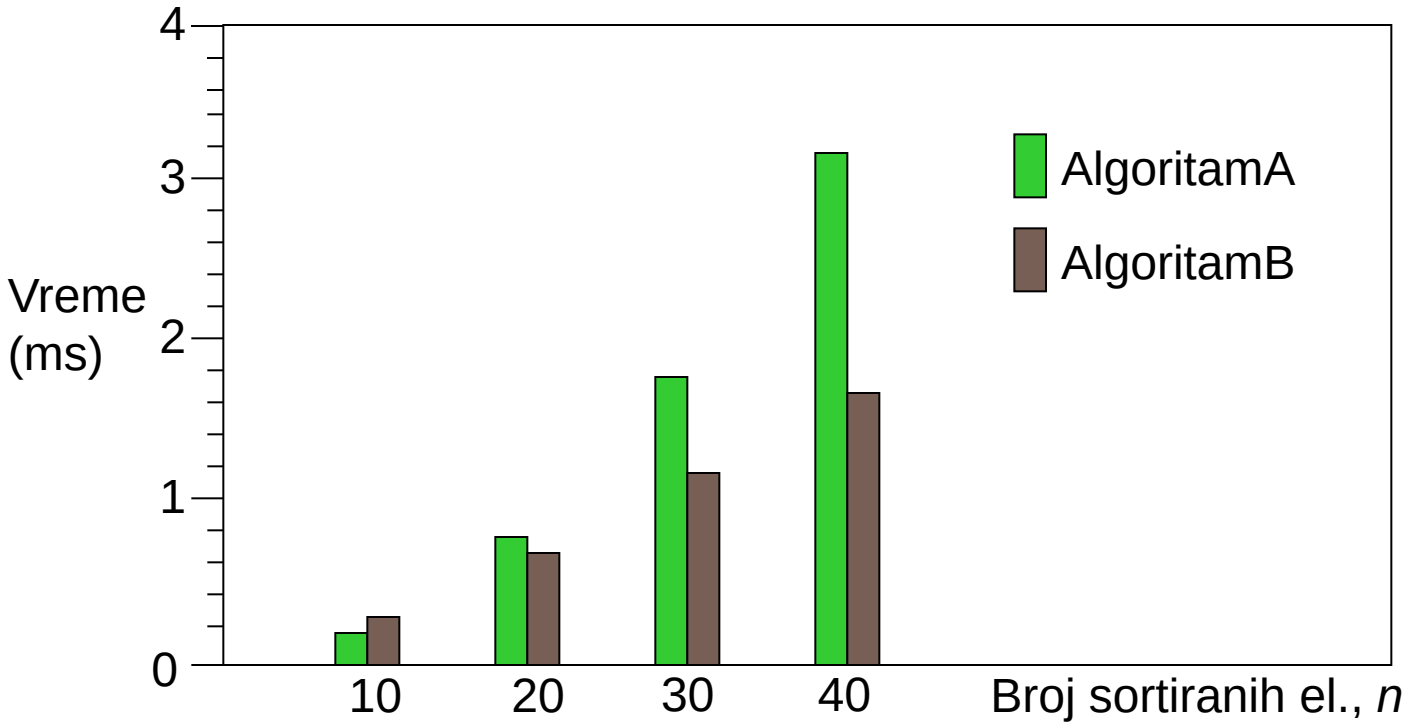
- Algoritam mora biti zapisan na nekom **jeziku** koji procesor “razume” (mada je sama procedura na kojoj se zasniva algoritam potpuno nezavisna od izabranog jezika).
- Problem koji razmatramo mora biti **rešiv,** što znači da je moguće rešiti problem u konačnom nizu koraka.

Efikasnost

- Za nekoliko algoritama koji rešavaju isti problem, kako možemo reći koji je "najbolji":
  - Da li je **upotrebljiv**? Da li je dovoljno efikasan da se koristi u praksi?
  - Koliko vremena mu treba da se izvrši?
  - Koliko prostora, u smislu zauzeća memorije" mu je potrebno?
- U najvećem broju slučajeva, i prostorna i vremenska složenost zavise od veličine ulaznih podataka.

Primer

- Na primer, poredimo dva algoritma za sortiranje:



- Vreme algoritma B raste sporije od vremena algoritma A.

Merenja

- Da bi saznali više o algoritmu (i po mogućstvu o implementaciji čitavog programa) možemo analizirati algoritam.
- Šta može da se analizira:
  - Vreme izvršavanja programa kao funkcija veličine ulaza;
  - Ukupni ili maksimalni memorijski zahtevi algoritma;
  - Ukupna veličina programskog koda;
  - Da li program korektno izračunava krajnji rezultat;
  - Kompleksnost programa – koliko lako program može da se čita, razume, modifikuje itd;
  - Robusnost programa – u kojoj meri se program snalazi sa neočekivanim ili pogršnim ulazom?

Više o merenju

- Vreme izvršavanja (ovo nas verovatno najviše zanima).
- Zauzeće memorije tokom rada programa.
- Faktori koji utiču na vreme izvršavanja programa:
  - Sam algoritam,
  - Ulazni podaci,
  - Računar na kome se izvršava program. Hardver:
    - Procesor (tip i brzina),
    - Raspoloživa memorija (keš i RAM),
    - Raspoloživ prostor na disku.
  - Programski jezik na kome je algoritam specificiran,
  - Kompaljer ili interpreter koji je korišćen,
  - Operativni sistem.

Merenje vremena

- Merenje vremena u sekundama?
  - \+ korisno i jednostavno u praksi
  - – zavisi od jezikam kompajlera i procesora.
- Brojanje koraka algoritma?
  - \+ ne zavisi od kompajlera i procesora
  - – zavisi od granularnosti koraka.
- Brojanje **karakterističnih** operacija? (npr. aritmetičkih operacija ili operacija poređenja)
  - \+ zavisi samo od algoritma (što i želimo)
  - \+ meri suštinsku efikasnost algoritma.
- Potrebno je definisati model računara!

Detaljan model računara

- Model nezavisan od aktuelnog hardvera i operativnog sistema.
- Potrebno je analizirati Java kod na Java VM.
- Gubimo ekspresivnu moć programskih jezika.
- I dalje očuvavamo dovoljno detalja.
- Baziran na skupu osnovnih aksioma.

Aksioma1

- Vreme potrebno da se pribavi operand iz memorije je konstantno, τ_{fetch}
- Vreme potrebno da se sačuva neka vrednost u memoriju je konstantno, τ_{store}
- Primeri:

`y = x`	$\tau_{fetch} + \tau_{store}$
`y = 1`	$\tau_{fetch} + \tau_{store}$

Aksioma 2

- Vreme potrebno da se izvrši elementarna matematička operacija (sabiranje, oduzimanje, množenje, deljenje, poredjenje) *je konstantno.*
- *Ova vremena označavamo sa* τ_+, τ_-, τ_x, $\tau_/$ *i* $\tau_<$, *respektivno.*
- Primeri:

`y = y + 1;`	$2\tau_{fetch} + \tau_+ + \tau_{store}$
`y++;`	$2\tau_{fetch} + \tau_+ + \tau_{store}$

Aksioma 3

- Vreme potrebno da se pozove metod je konstantno, τ_{call}
- Vreme potrebno za vraćanje iz metoda je konstantno, τ_{return}

Aksioma 4

- Vreme potrebno za prosleđivanje argumenta metodi je isto kao i vreme potrebno da se sačuva neka vrednost u memoriji, τ_{store}
- Primer

`y = f(x);`	$\tau_{fetch} + 2\tau_{store} + \tau_{call} + T_{f(x)}$
	gde je $T_{f(x)}$ vreme izvršavanja metoda f za ulaz x

Primer

$$\sum_{i=1}^{n} i$$

```
1  public class Example
2  {
3      public static int sum (int n)
4      {
5          int result = 0;
6          for (int i = 1; i <= n; ++i)
7              result += i;
8          return result;
9      }
10 }
```

statement	time	code
5	$\tau_{\text{fetch}} + \tau_{\text{store}}$	`result = 0`
6a	$\tau_{\text{fetch}} + \tau_{\text{store}}$	`i = 1`
6b	$(2\tau_{\text{fetch}} + \tau_{<}) \times (n+1)$	`i <= n`
6c	$(2\tau_{\text{fetch}} + \tau_{+} + \tau_{\text{store}}) \times n$	`++i`
7	$(2\tau_{\text{fetch}} + \tau_{+} + \tau_{\text{store}}) \times n$	`result += i`
8	$\tau_{\text{fetch}} + \tau_{\texttt{return}}$	`return result`
TOTAL	$(6\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{<} + 2\tau_{+}) \times n$ $+ (5\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{<} + \tau_{\texttt{return}})$	

Aksioma 5

- Vreme potrebno za računanje adresa koje je implicirano pristupom elementu niza (a[i]), je konstantno, $\tau_{[\bullet]}$. Ovo vreme ne uključuje vreme potrebno za izračunavanje izraza u uglastim zagradama niti vreme za pristup elementu niza.
- Primer:

`y = a[i];`	$3\tau_{fetch} + \tau_{[\bullet]} + \tau_{store}$

Primer: Hornerova šema

$$\sum_{i=1}^{n} a_i x^i$$

```
1  public class Example
2  {
3      public static int horner (int[] a, int n, int x)
4      {
5          int result = a [n];
6          for (int i = n - 1; i >= 0; --i)
7              result = result * x + a [i];
8          return result;
9      }
10 }
```

statement	time
5	$3\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{\text{store}}$
6a	$2\tau_{\text{fetch}} + \tau_{-} + \tau_{\text{store}}$
6b	$(2\tau_{\text{fetch}} + \tau_{<}) \times (n+1)$
6c	$(2\tau_{\text{fetch}} + \tau_{-} + \tau_{\text{store}}) \times n$
7	$(5\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{+} + \tau_{\times} + \tau_{\text{store}}) \times n$
8	$\tau_{\text{fetch}} + \tau_{\texttt{return}}$
TOTAL	$(9\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{<} + \tau_{[\cdot]} + \tau_{+} + \tau_{\times} + \tau_{-}) \times n$ $+ (8\tau_{\text{fetch}} + 2\tau_{\text{store}} + \tau_{[\cdot]} + \tau_{-} + \tau_{<} + \tau_{\texttt{return}})$

Analiza rekurzivnih funkcija

- Faktorijel prirodnih brojeva
  - Iterativno

$$n! = \begin{cases} 1 & n = 0, \\ \prod_{i=1}^{n} i & n > 0. \end{cases}$$

  - Rekurzivno

$$n! = \begin{cases} 1 & n = 0, \\ n \times (n-1)! & n > 0. \end{cases}$$

Analiza *n*!

```
1  public class Example
2  {
3      public static int factorial (int n)
4      {
5          if (n == 0)
6              return 1;
7          else
8              return n * factorial (n - 1);
9      }
10 }
```

	time	
statement	$n = 0$	$n > 0$
5	$2\tau_{\text{fetch}} + \tau_{<}$	$2\tau_{\text{fetch}} + \tau_{<}$
6	$\tau_{\text{fetch}} + \tau_{\texttt{return}}$	—
8	—	$3\tau_{\text{fetch}} + \tau_{-} + \tau_{\text{store}} + \tau_{\times} + \tau_{\text{call}} + \tau_{\texttt{return}} + T(n-1)$

Analiza *n*! (nastavak)

$$T(n) = \begin{cases} t_1 & n = 0, \\ T(n-1) + t_2 & n > 0, \end{cases}$$

where $t_1 = 3\tau_{\text{fetch}} + \tau_{<} + \tau_{\texttt{return}}$,

and $t_2 = 5\tau_{\text{fetch}} + \tau_{<} + \tau_{-} + \tau_{\text{store}} + \tau_{\times} + \tau_{\text{call}} + \tau_{\texttt{return}}$.

$$\begin{aligned} T(n) &= T(n-1) + t_2 \\ &= (T(n-2) + t_2) + t_2 \\ &= T(n-2) + 2t_2 \\ &= (T(n-3) + t_2) + 2t_2 \\ &= T(n-3) + 3t_2 \\ &\vdots \\ &= T(n-k) + kt_2 \\ &\vdots \\ &= T(0) + nt_2 \\ &= t_1 + nt_2 \end{aligned}$$

Napredniji primer

$$\max_{0 \le i \le n} a_i$$

```
1  public class Example
2  {
3      public static int findMaximum (int[] a)
4      {
5          int result = a [0];
6          for (int i = 1; i < a.length; ++i)
7              if (a [i] > result)
8                  result = a [i];
9          return result;
10     }
11 }
```

statement	time
5	$3\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{\text{store}}$
6a	$\tau_{\text{fetch}} + \tau_{\text{store}}$
6b	$(2\tau_{\text{fetch}} + \tau_{<}) \times n$
6c	$(2\tau_{\text{fetch}} + \tau_{+} + \tau_{\text{store}}) \times (n-1)$
7	$(4\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{<}) \times (n-1)$
8	$(3\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{\text{store}}) \times ?$
9	$\tau_{\text{fetch}} + \tau_{\text{store}}$

$$T(n, a_0, a_1, \ldots, a_{n-1}) = t_1 + t_2 n + \sum_{\substack{i=1 \\ a_i > (\max_{0 \le j < i} a_j)}}^{n-1} t_3$$

$$\begin{aligned} t_1 &= 2\tau_{\text{store}} - \tau_{\text{fetch}} - \tau_{+} - \tau_{<} \\ t_2 &= 8\tau_{\text{fetch}} + 2\tau_{<} + \tau_{[\cdot]} + \tau_{+} + \tau_{\text{store}} \\ t_3 &= 3\tau_{\text{fetch}} + \tau_{[\cdot]} + \tau_{\text{store}}. \end{aligned}$$

$$T_{\text{average}}(n) = t_1 + t_2 n + \sum_{i=1}^{n-1} p_i t_3$$

gde je p_i verovatnoća da je i-ti član niza veći od svih prethodnih a[0]..a[i-1]

Aksioma 6

- Vreme potrebno da se kreira novi objekat korišćenjem `new` operatora je konstantno, τ_{new}. Ovo vreme ne obuhvata vreme potrebno da se objekat inicijalizuje.
- Primer

`Integer ref = new Integer (0);`	$\tau_{new} + \tau_{fetch} + 2\tau_{store} + \tau_{call} + T_{(\texttt{Integer}())}$
	where $T_{(\texttt{Integer}())}$ *is the running time of the* `Integer` *constructor*

Pojednostavljeni model računara

- Detaljni model je dovoljno fleksibilan, ali ima mnogo parametara.
- Pojednostavljeni model je manje precizan, ali jednostavniji za korišćenje.
- Vremena predstavljaju umnoške taktova računara
  - $\tau_{return} = k_{return} x T,\ k_{return} \in N$
- Pretpostavke:
  - Svi vremenski parametri su izraženi u jedinicama taktova, T=1
  - Sve konstante iz aksioma su jednake jedan, k=1
- Sada brojimo samo taktove!

Primer 1

$$\sum_{i=0}^{n} x^i$$

```
1  public class Example
2  {
3      public static int geometricSeriesSum (int x, int n)
4      {
5          int sum = 0;
6          for (int i = 0; i <= n; ++i)
7          {
8              int prod = 1;
9              for (int j = 0; j < i; ++j)
10                 prod *= x;
11             sum += prod;
12         }
13         return sum;
14     }
15 }
```

statement	time
5	2
6a	2
6b	$3(n+2)$
6c	$4(n+1)$
8	$2(n+1)$
9a	$2(n+1)$
9b	$2\sum_{i=0}^{n}(i+1)$
9c	$4\sum_{i=0}^{n} i$
10	$4\sum_{i=0}^{n} i$
11	$4(n+1)$
13	2
TOTAL	$\frac{11}{2}n^2 + \frac{47}{2}n + 24$

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

Primer 2

$$\sum_{i=0}^{n} x^i$$

```
1  public class Example
2  {
3      public static int geometricSeriesSum (int x, int n)
4      {
5          int sum = 0;
6          for (int i = 0; i <= n; ++i)
7              sum = sum * x + 1;
8          return sum;
9      }
10 }
```

statement	time
5	2
6a	2
6b	$3(n+2)$
6c	$4(n+1)$
7	$6(n+1)$
8	2
TOTAL	$13n+22$

Primer x^n

```
1  public class Example
2  {
3      public static int power (int x, int n)
4      {
5          if (n == 0)
6              return 1;
7          else if (n % 2 == 0) // n is even
8              return power (x * x, n / 2);
9          else // n is odd
10             return x * power (x * x, n / 2);
11     }
12 }
```

statement	time		
	$n = 0$	$n > 0$ n is even	$n > 0$ n is odd
5	3	3	3
6	2	—	—
7	—	5	5
8	—	$10 + T(\lfloor n/2 \rfloor)$	—
10	—	—	$12 + T(\lfloor n/2 \rfloor)$
TOTAL	5	$18 + T(\lfloor n/2 \rfloor)$	$20 + T(\lfloor n/2 \rfloor)$

$$T(n) = \begin{cases} 5 & n = 0 \\ 18 + T(\lfloor n/2 \rfloor) & n > 0, n \text{ is even}, \\ 20 + T(\lfloor n/2 \rfloor) & n > 0, n \text{ is odd}. \end{cases}$$

$$T(n) = 19(\lfloor \log_2 n \rfloor + 1) + 5$$

Primer 3

$$\sum_{i=0}^{n} x^i$$

```
1  public class Example
2  {
3      public static int geometricSeriesSum (int x, int n)
4      {
5          return (power (x, n + 1) - 1) / (x - 1);
6      }
7  }
```

$$T(n) = 19(\lfloor \log_2(n+1) \rfloor + 1) + 18$$

Poređenje

- Primer 1
- Primer 2
- Primer 3

T(n)
$(\frac{11}{2}n^2 + \frac{47}{2}n + 24)$
$13n + 22$
$19(\lfloor \log_2(n+1) \rfloor + 1) + 18$