Uvod u računarstvo
15. predavanje

Saša Singer

singer@math.hr
web.math.hr/~singer

PMF – Matematički odjel, Zagreb

Sadržaj predavanja

- Pretraživanje i sortiranje nizova:
  - Razne varijante Selection sorta.
  - Složenost sortiranja — općenito.
  - Složenost Selection sorta.
  - Sortiranje zamjenama susjednih elemenata — Bubble sort.
  - Poboljšana varijanta Bubble sorta.
  - Složenost Bubble sorta.
- Završni primjeri (ponavljanje za kolokvij):
  - Zadatak 1.
  - Zadatak 2.

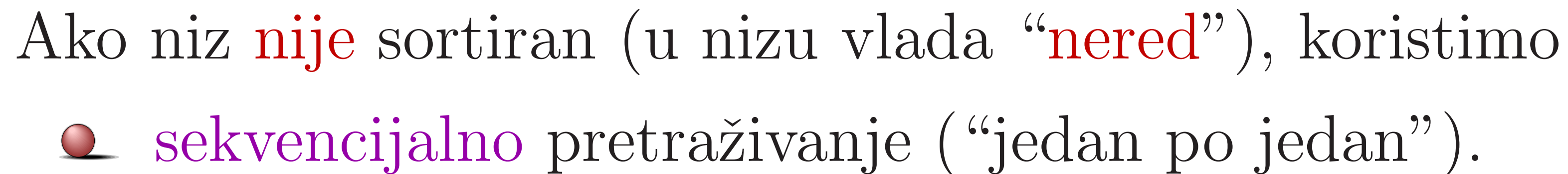
Pretraživanje i sortiranje (nastavak)

Sadržaj

- Pretraživanje i sortiranje nizova:
  - Pretraživanje — sekvencijalno i binarno (ponavljanje).
  - Sortiranje izborom ekstrema — Selection sort (ponavljanje).
  - Razne varijante Selection sorta.
  - Složenost sortiranja — općenito.
  - Složenost Selection sorta.
  - Sortiranje zamjenama susjednih elemenata — Bubble sort.
  - Poboljšana varijanta Bubble sorta.
  - Složenost Bubble sorta.

Pretraživanje nizova (ponavljanje)

Problem pretraživanja: provjeriti nalazi li se zadani element `elt` među članovima zadanog niza



$$x_0, x_1, \ldots, x_{n-1}.$$

Ako niz nije sortiran (u nizu vlada “nered”), koristimo

- sekvencijalno pretraživanje (“jedan po jedan”).

Sekvencijalno pretraživanje

```
int seq_search(int x[], int n, int elt)
{
    int i;

    for (i = 0; i < n; ++i)
        if (x[i] == elt)
            return 1;

    return 0;
}
```

Sekvencijalno pretraživanje (nastavak)

Složenost mjerimo brojem usporedbi

- "jednak", odnosno, "različit".

U najgorem slučaju, moramo provjeriti sve članove niza, tj.

$$\text{broj usporedbi} \leq n.$$

Ova mjera složenosti je dobra procjena za trajanje izvršavanja algoritma sekvencijalnog pretraživanja.

Zapis:

$$T(n) = \mathcal{O}(n).$$

Značenje: trajanje u najgorem slučaju linearno ovisi o n.

Točno značenje zapisa složenosti

Točno matematičko značenje zapisa

$$T(n) = \mathcal{O}(f(n))$$

za neke funkcije T i f (sa $\mathbb{N}$ u $\mathbb{R}$):

Postoji konstanta $c \in \mathbb{R}$ i postoji $n_0 \in \mathbb{N}$ takvi da za svaki $n \in \mathbb{N}$ vrijedi implikacija

$$n \geq n_0 \implies T(n) \leq c \cdot f(n).$$

Prijevod: T raste sporije od neka konstanta puta f, za sve dovoljno velike n.

Binarno pretraživanje

Ako je niz uzlazno ili silazno sortiran,

$$x_0 \le x_1 \le \cdots \le x_{n-1} \quad \text{ili} \quad x_0 \ge x_1 \ge \cdots \ge x_{n-1},$$

koristimo

- binarno pretraživanje ("raspolavljanje").

Binarno pretraživanje (nastavak)

```
int binary_search(int x[], int n, int elt)
{
    int l = 0, d = n - 1, i;
    while (l <= d) {
        i = (l + d) / 2;
        if (elt < x[i])
            d = i - 1;
        else if (elt > x[i])
            l = i + 1;
        else
            return 1;
    }
    return 0; }
```

Binarno pretraživanje (nastavak)

Složenost opet mjerimo brojem usporedbi, ali sada koristimo

- "manji (ili jednak)", odnosno, "veći (ili jednak)".

U najgorem slučaju, za broj raspolavljanja k vrijedi

$$2^{k-1} \leq n < 2^k,$$

ili

$$k \leq 1 + \lfloor \log_2 n \rfloor.$$

Svako raspolavljanje ima najviše 2 usporedbe, pa je

$$\text{broj usporedbi} \leq 2 \cdot (1 + \lfloor \log_2 n \rfloor).$$

Binarno pretraživanje (nastavak)

Mož se napraviti i varijanta sa samo jednom usporedbom u svakom raspolavljanju (probajte sami).

Zapis za trajanje:

$$T(n) = \mathcal{O}(\log n).$$

Značenje: trajanje u najgorem slučaju logaritamski ovisi o n.

Zaključak: Sortiramo zato da bismo brže tražili!

Sortiranje nizova izborom ekstrema

Ideja: Koristimo usporedbe i zamjene elemenata u nizu.

- Dovedi najmanji element niza $x_0, x_1, \ldots, x_{n-1}$ na njegovo mjesto.
- To mjesto je prvo u cijelom nizu, pa je (nakon zamjene), nova vrijednost elementa x_0 upravo najmanji element niza.
- Postupak ponovi na skraćenom (nesređenom) nizu $x_1, \ldots, x_{n-1}$ (duljine za jedan manje, tj. $n - 1$).
- Niz se "skraćuje" sprijeda.
- To ponavljamo sve dok ne "stignemo" na niz sa samo jednim elementom (x_{n-1}) — taj je sigurno sortiran!

Sortiranje nizova izborom ekstrema (nastavak)

Prva varijanta (prošli puta):

- kod traženja ekstrema pamtimo:
  - vrijednost ekstrema,
  - indeks elementa na kojem se ekstrem dostiže.

Skraćena varijanta (po duljini kôda, ali ne mora biti i brža):

- očito je dovoljno pamtiti samo
  - indeks elementa na kojem se ekstrem dostiže,

  i to iskoristiti kod usporedbi.

Sortiranje nizova izborom ekstrema (nastavak)

```
void selection_sort(int x[], int n) {
    int i, j, ind_min, temp;
    for (i = 0; i < n - 1; ++i) {
        ind_min = i;
        for (j = i + 1; j < n; ++j)
            if (x[j] < x[ind_min])
                ind_min = j;
        if (i != ind_min) {
            temp = x[i];
            x[i] = x[ind_min];
            x[ind_min] = temp; }
    }
    return; }
```

Složenost sortiranja nizova

Kako ćemo uspoređivati koliko je brzo sortiranje (raznim algoritmima)?

- Možemo mjeriti vrijeme.
- Možemo uspoređivati broj operacija koje program obavlja. Taj broj operacija jedna je od mjera složenosti algoritma.

Primijetite da kod sortiranja imamo dvije bitno različite operacije (koje ne moraju jednako trajati):

- uspoređivanje elemenata,
- zamjene elemenata.

Složenost sortiranja izborom ekstrema

Kod sortiranja izborom ekstrema vrijedi:

- broj usporedbi u svakom koraku jednak je duljini trenutnog niza umanjenoj za 1, jer se svaki element uspoređuje s trenutno najmanjim.

  Za sve korake zajedno, broj usporedbi je zbroj:

$$(n-1)+(n-2)+\cdots+2+1=\frac{(n-1)\cdot n}{2}\approx\frac{n^2}{2}.$$

  Dakle, broj usporedbi (sigurno) kvadratno ovisi o n.

Složenost sortiranja izborom ekstrema

- u svakom koraku vrši se najviše jedna zamjena nekog para elemenata (može je i ne biti, ako je najmanji na pravom mjestu).

  Ukupno imamo najviše

  $$n - 1 \quad \text{zamjena.}$$

  Dakle, broj zamjena (najviše) linearno ovisi o n.

Zaključak: za trajanje vrijedi

$$T(n) = \mathcal{O}(n^2).$$

Sortiranje nizova izborom ekstrema (nastavak)

Dosad smo uvijek sortirali dovođenjem najmanjeg elementa na početak. Isti efekt imat će i dovođenje najvećeg na kraj. Ideja:

- Dovedi najveći element niza $x_0, x_1, \ldots, x_{n-1}$ na njegovo mjesto (to je zadnje u cijelom nizu).
- Postupak ponovi na skraćenom (nesređenom) nizu $x_0, \ldots, x_{n-2}$ (duljine za jedan manje, tj. $n - 1$).
- Niz se "skraćuje" straga.

Sortiranje nizova izborom ekstrema (nastavak)

```
void selection_sort(int x[], int n) {
    int i, j, ind_max, temp;
    for (i = n - 1; i > 0; --i) {
        ind_max = i;
        for (j = 0; j < i; ++j)
            if (x[j] > x[ind_max])
                ind_max = j;
        if (i != ind_max) {
            temp = x[i];
            x[i] = x[ind_max];
            x[ind_max] = temp; }
    }
    return; }
```

Sortiranje nizova izborom ekstrema (nastavak)

Složenost — ista kao kod dovođenja najmanjeg na početak:

- broj usporedbi:

$$\frac{(n-1)\cdot n}{2} \approx \frac{n^2}{2} = \mathcal{O}(n^2).$$

- broj zamjena je manji ili jednak:

$$n-1 = \mathcal{O}(n).$$

Sortiranje zamjenama susjeda

Sortiranje zamjenama susjeda (engl. bubble sort, bubble = mjehurić) bazira se na zamjenama susjednih elemenata.

Ideja:

- Ako dva susjedna člana niza x_j i x_{j+1} nisu u dobrom poretku, zamijeni im mjesto (vrijednost).
- Kad stignemo do kraja niza (u prvom prolazu), ponovimo postupak.
- Nije jasno kada ćemo stati (jer se stalno vraćamo na početak!) — to ćemo analizirati nakon primjera.

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

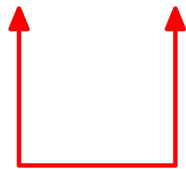
42	12	55	94	18	44	67

```
zamjena = 0
```

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

42	12	55	94	18	44	67



`zamjena = 1`

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	94	18	44	67

```
zamjena = 1
```

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

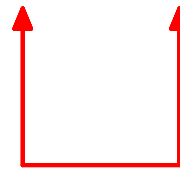
12	42	55	94	18	44	67

`zamjena = 1`

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	94	18	44	67

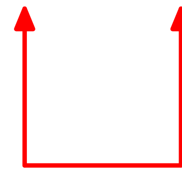


`zamjena = 1`

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	18	94	44	67

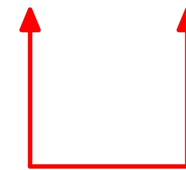


`zamjena = 1`

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	18	44	94	67



zamjena = 1

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	18	44	67	94

```
zamjena = 0
```

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	18	44	67	94

```
zamjena = 0
```

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

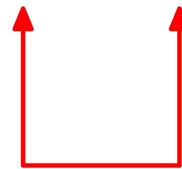
12	42	55	18	44	67	94

`zamjena = 0`

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	55	18	44	67	94

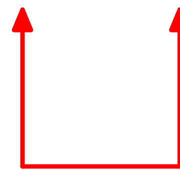


`zamjena = 1`

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	18	55	44	67	94



`zamjena = 1`

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	18	44	55	67	94

```
zamjena = 1
```

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	18	44	55	67	94

```
zamjena = 0
```

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

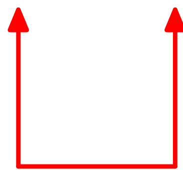
12	42	18	44	55	67	94

```
zamjena = 0
```

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	42	18	44	55	67	94



```
zamjena = 1
```

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

```
zamjena = 1
```

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

`zamjena = 1`

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

```
zamjena = 0
```

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

```
zamjena = 0
```

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

```
zamjena = 0
```

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

`zamjena = 0`

Sortiranje zamjenama susjednih elemenata

Primjer. Zamjenama susjednih elemenata sortirajte zadano polje.

12	18	42	44	55	67	94

Sortiranje zamjenama susjeda (nastavak)

Kada stajemo?

- Primijetimo da smo u prvom prolazu najveći element "odgurali" na kraj niza (njegovo pravo mjesto).
- I drugi, veći elementi počeli su "putovati" prema kraju niza.
- Zaključak: nakon prvog koraka niz možemo skratiti za posljednji element i nastaviti postupak s nizom $x_0, \ldots, x_{n-2}$.

Sortiranje zamjenama susjeda (nastavak)

```
void bubble_sort(int x[], int n) {
    int i, j, temp;

    for (i = 1; i < n; ++i)
        for (j = 0; j < n - i; ++j)
            if (x[j] > x[j + 1]) {
                temp = x[j];
                x[j] = x[j + 1];
                x[j + 1] = temp; }
    return; }
```

Sortiranje zamjenama susjeda (nastavak)

Analiza složenosti algoritma:

- U prvom prolazu uspoređujemo $n-1$ parova susjeda, u drugom $n-2$, ... , i tako redom. Dakle, ukupan broj usporedbi je

$$(n-1)+(n-2)+\cdots+2+1=\frac{(n-1)\cdot n}{2}\approx\frac{n^2}{2}=\mathcal{O}(n^2).$$

- Broj zamjena može drastično varirati od 0 (ako je niz već sortiran) do najviše

$$(n-1)+(n-2)+\cdots+2+1=\frac{(n-1)\cdot n}{2}\approx\frac{n^2}{2}=\mathcal{O}(n^2).$$

To će se dogoditi ako je niz naopako sortiran.

Sortiranje zamjenama susjeda (nastavak)

Dakle, ovakvo sortiranje zamjenama susjeda ne treba upotrebljavati, jer je loše (ima previše zamjena).

Može li se algoritam poboljšati?

- Možemo pamtiti koliko smo zamjena napravili u trenutnom prolazu nizom. Ako nismo napravili nijednu — možemo stati.
  - Dovoljno je: "ima — nema" zamjena u tom prolazu.
- Recimo, za ispravno sortiran niz, potreban je samo jedan prolaz da se ustanovi da je niz sortiran.
- U slučaju naopako (obratno) sortiranog niza, nismo uštedili ništa (nema spasa).

Sortiranje zamjenama susjeda (nastavak)

```
void bubble_sort(int x[], int n) {
    int i, j, temp, zamjena;
    i = n - 1;
    do {
        zamjena = 0;
        for (j = 0; j < i; ++j)
            if (x[j] > x[j + 1]) {
                temp = x[j];
                x[j] = x[j + 1];
                x[j + 1] = temp;
                zamjena = 1; }
        --i;  /* smanji i za sljedeci prolaz. */
    } while (zamjena);
    return; }
```

Sortiranje zamjenama susjeda (nastavak)

Zaključak:

- sortiranje zamjena susjeda se ne isplati za opće nizove,
- možda se i može isplatiti za "skoro sortirane nizove", ali to nije lako definirati što je.

Još o sortiranju

- U “bubble” sortu uvijek napredujemo od početka niza. Ako jednom krenemo od početka, pa zatim na nazad, pa opet naprijed, dobit ćemo tzv. engl. shaker sort (u doslovnom prijevodu streseni sort).
- Možemo sortirati na “kontrolirane” udaljenosti, recimo susjede, pa malo dalje članove ... Takav sort zove se engl. shell sort (u doslovnom prijevodu školjkasti sort). Analiza složenosti mu je komplicirana, ali algoritam može biti brži od kvadratnog.

Još o sortiranju

- Za sortiranje zamjenama elemenata unutar istog niza korištenjem usporedbi članova, može se pokazati da je

$$\text{broj usporedbi} \geq c \cdot n \log n,$$

(c je pozitivna konstanta) tj. broj usporedbi je reda veličine barem $n \log n$, što može biti bitno brže nego n^2 usporedbi (za velike n).

Još o sortiranju

Algoritmi koji se koriste u praksi:

- Quicksort – prosječna složenost mu je $n \log n$ za slučajne dobro razbacane nizove, ali u najgorem slučaju i dalje mu je složenost n^2. Koristi se zbog dobre prosječne brzine i dio je standardne C-biblioteke.
- Heapsort – ima i prosječnu i najgoru složenost $n \log n$, ali je u prosjeku sporiji od quicksorta (opis SPA).

Grubi opis quicksorta

Quicksort se temelji na principu podijeli pa vladaj.

- Uzmemo jedan element x_k iz niza i dovedemo ga na njegovo pravo mjesto.
- Lijevo od njega ostavimo elemente koji su manji ili jednaki njemu (u bilo kojem poretku).
- Desno od njega ostavimo elemente koji su veći od njega (u bilo kojem poretku).
- Ako smo dobro izabrali, tj. ako je mjesto x_k blizu sredine, onda ćemo morati sortirati dva polja polovične duljine.
- U najgorem slučaju, ako smo izabrali “krivi” x_k, morat ćemo sortirati polje duljine $n - 1$.