Strukture podataka
Letnji semestar

Polje
(Array)

Prof. Dr Leonid Stoimenov
Katedra za računarstvo
Elektronski fakultet u Nišu

Polje - Pregled

- Definicija polja
- Memorijska reprezentacija polja
- Operacije za rad sa poljima

Uvod

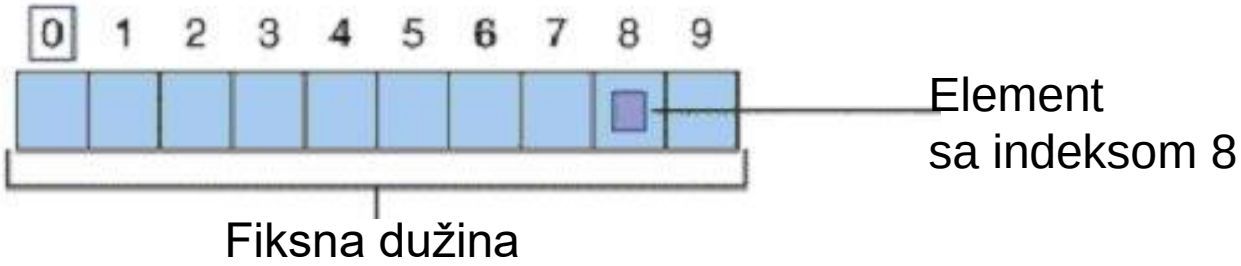
Polje (Array)

- Najjednostavnija struktura podataka
- Mnogi programski jezici imaju ugrađeni ***array*** tip podatka

- Upotreba
  - Za memorisanje **fiksne količine podataka** kojima se slučajno pristupa
  - Za **formiranje složenijih struktura** – niza, magacina, redova, dekova, tablica, gomila, ...

OSNOVNI POJMOVI

- **Polje** je kolekcija homogenih elemenata podataka kojima se pristupa pomoću **numeričkog indeksa**
- Broj elemenata polja naziva se **veličina, rang** ili **dužina** polja
- Broj indeksa koji su dodeljeni svakom elementu polja naziva se **dimenzionalnost** polja
- Najčešće se koriste:
  - **Jednodimenzionalna (1D)** polja ili **vektori**
  - **Dvodimenzionalna (2D)** polja ili **matrice**
- Polje sa više od 2 indeksa se naziva **višedimenzionalno** ili **multidimenzionalno (MD)** polje



INDEKSI POLJA

- Indeksi polja veličine ***n*** mogu uzimati vrednosti:
  - zero-based array 0,1,...,n-1
  - one-based array 1,2,...,n
  - n-based array d,d+1,d+2,...,g
  - d – donja granica indeksa
  - g – gornja granica indeksa
    - $d \leq g$
    - d,g bilo koji ceo broj
  - broj elemenata polja

$$n=|d|-|g|+1$$

- C, C++, C#, Java
  - 0,1,2,...
- Basic, Fortran
  - 1,2,...
- Ada, PL/1, Visual Basic, Visual Basic.NET
  - d,d+1,d+2,...,g
  - ako d nije zadato podrazumeva se 1

Primer *int a[100]*
Polje *a* je ranga:
zero-based array
n = 99-0+1=100
one-based array
n = 100-1+1=100

VRSTE POLJA

- ***Statičko polje*** – polje fiksne veličine, ne menja se ni donja ni gornja granica, pa samim tim ni rang polja
  - npr. U prog. jeziku C *d* je fiksirano na 0, a *g* u vreme pisanja programa
- Dinamičko polje – polje čija se veličina može menjati u vremenu
  - veličina polja se automatski podešava pri dodavanju i brisanju elemenata polja
  - C-jezik *malloc( )*
  - Visual Basic *REDIM*
  - U nekim prog. jezicima sva polja su dinamička *(npr. Perl)*

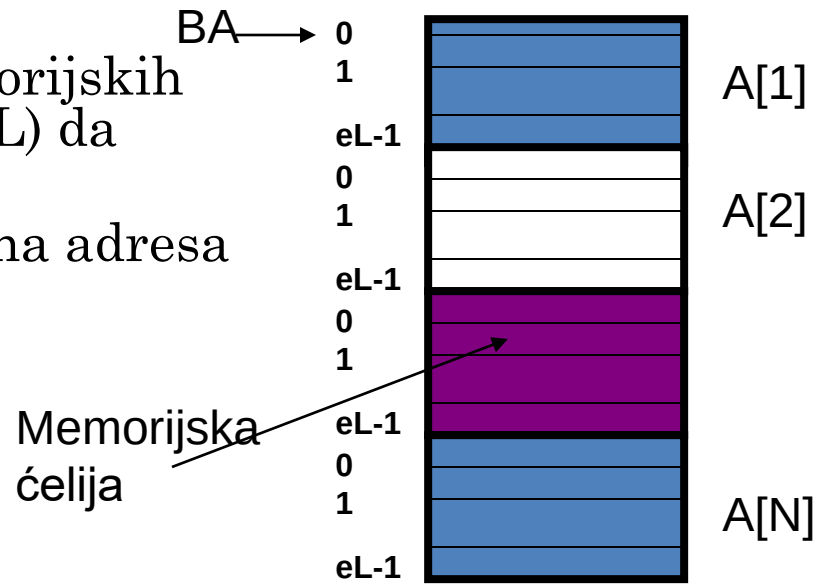
MEMORIJSKA REPREZENTACIJA POLJA

- 1D polje A dužine N tipa T
  - rezerviše se N sukcesivnih memorijskih ćelija, svaka dovoljne veličine (eL) da prihvati podatak zadatog tipa T
  - Adresa prve ćelije se naziva bazna adresa polja A (*BA*)
  - Adresa elementa *A[i] je*

    *BA + i*eL , ako i=0,1,...*

    *BA + (i-1)*eL, ako i=1,2,...*

    *BA + (i-d)*eL, ako i=d,...,g*



- *Deskriptor polja sadrži:*
  - Baznu adresu BA
  - Dužinu polja N ili granice indeksa
  - Tip podatka T ili dužinu elementa polja eL

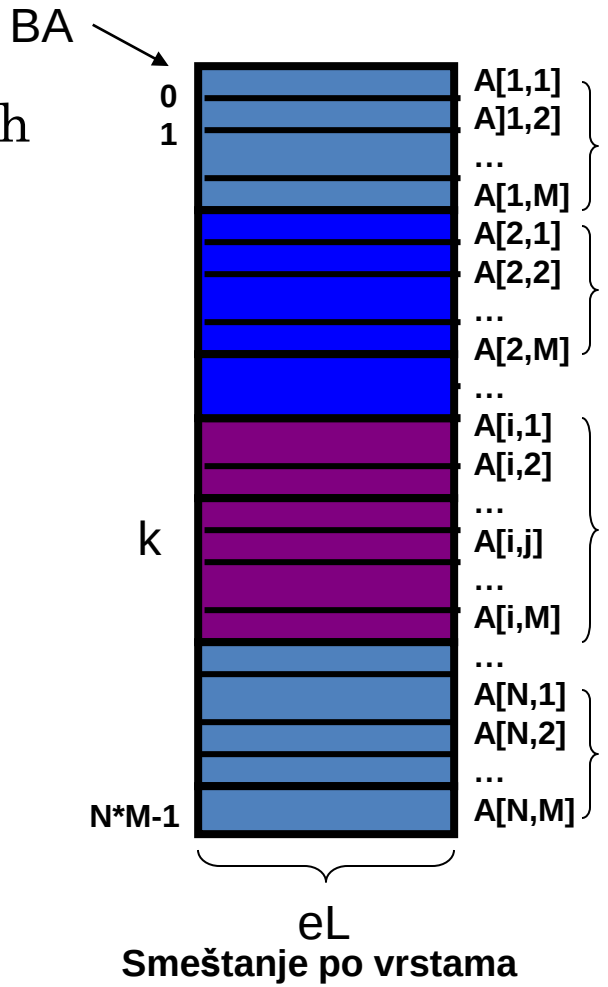
A[1] je memorisan po adresama BA, BA+1,…,BA+eL-1

A[2] je memorisan po adresama BA+eL, BA+eL+1,…, BA+2*eL-1

…….

Memorijska reprezentacija polja (2)

- 2D polje A dimenzije N x M tipa T
  - rezerviše se N*M sukcesivnih memorijskih ćelija, svaka dovoljne veličine (eL) da prihvati podatak zadatog tipa T
  - Adresa prve ćelije se naziva bazna adresa polja A (*BA*)
  - Adresa elementa *A[i,j] je BA+k*eL, k=0,1,..*
    - *k = N*(j-1) + i, smeštanje po kolonama*
    - *k = M*(i-1) + j, smeštanje po vrstama*
    - *i,j=1,2,…,N*
    - *k = N*j + i, smeštanje po kolonama*
    - *k = M*i + j, smeštanje po vrstama*
    - *i,j=0,1,…,N*
- *Deskriptor polja sadrži:*
  - *Baznu adresu BA*
  - *Dužinu polja N, M ili granice oba indeksa*
  - *Tip podatka T ili dužinu elementa polja eL*



Memorijska reprezentacija specijalnih polja

- Matrice
  - retke (manje od 10% elemenata je ≠0)
  - jedinične (elementi matrice su 0 ili 1)
  - dijagonalne
  - trougaone (donja i gornja)
- Zahtevaju poseban način memorijske reprezentacije radi efikasnije obrade
- U ove reprezentacije se ugrađuju inherentna svojstva ovih struktura

Operacije nad poljima

Proste operacije

- Memorisanje elementa polja (***store***)
  A[i] ← E
- Referenciranje, izdvajanje elementa polja (***extract***)
  E ← A**[i]**

Kompozitne operacije

- Obilazak polja *arrayTraversal*
- Traženje
  - Linearno *arrayLinearSeach*
  - Binarno arrayBinarySeach
- Sortiranje
  arrayBubleSort
  QuickSort
  MergeSort
  ...
- Umetanje *arrayInsertAt*
- Brisanje *arrayDeleteAt*

OBILAZAK POLJA

- Podrazumeva obilazak i obradu svih elemenata strukture, u ovom slučaju elemenata polja

Algoritam P.1. Obilazak polja

arrayTraversal(A,d,g)

// 1D polje A[d,g]

// Ovaj algoritam obilazi polje A i nad svakim

// elementom polja primenjuje operaciju OBRADA

```
1      k ←d                    // indeks prvog elementa polja
2      while (k ≤ g)           // ili for petlja!
3         OBRADA(A[k])
4         k ← k+1 // ažuriranje indeksa polja
5      return
```

```
for k=d,g
  OBRADA(A[k])
```

Broj primitivnih operacija: 2+5n
Veličina polja: n=g-d+1
Asimptotska složenost: **O(n), linearna**

Linearno (sekvencijalno) traženje

- Traženje je osnovna operacija za sve strukture podataka, uključujući i polje
- Podrazumeva i obilazak polja, pri čemu ne moraju da se obiđu svi elementi

Algoritam P.2. Linearno traženje

arrayLinearSeach(A,n,E)

// A je 1D polje od n elemenata
// E je element koji se traži
// LOC je lokacija nađenog elementa
// ili je LOC=NULL
// Ovaj algoritam traži element E u polju
// A i vraća njegovu lokaciju,
// ako je traženje uspešno,
// ili LOC=NULL ako je traženje neuspešno

```
1     for LOC=1,n
2           if(A[LOC]=E) then return// traženje uspešno
4      LOC=NULL // traženje neuspešno
5      return LOC
```

Najgori slučaj: E se ne nalazi u A

broj poređenja: n

Najbolji slučaj: E je prvi element polja

broj poređenja: 1

Prosečan slučaj: Ako je ista verovatnoća da se element nađe na bilo kojoj poziciji, tada je prosečna verovatnoća:

1*1/n+2*1/n+…+n*1/n

=(1+2+..+n)/n

=n(n+1)/2n

=(n+1)/2

Asimptotska složenost:

O(n), linearna

Binarno traženje

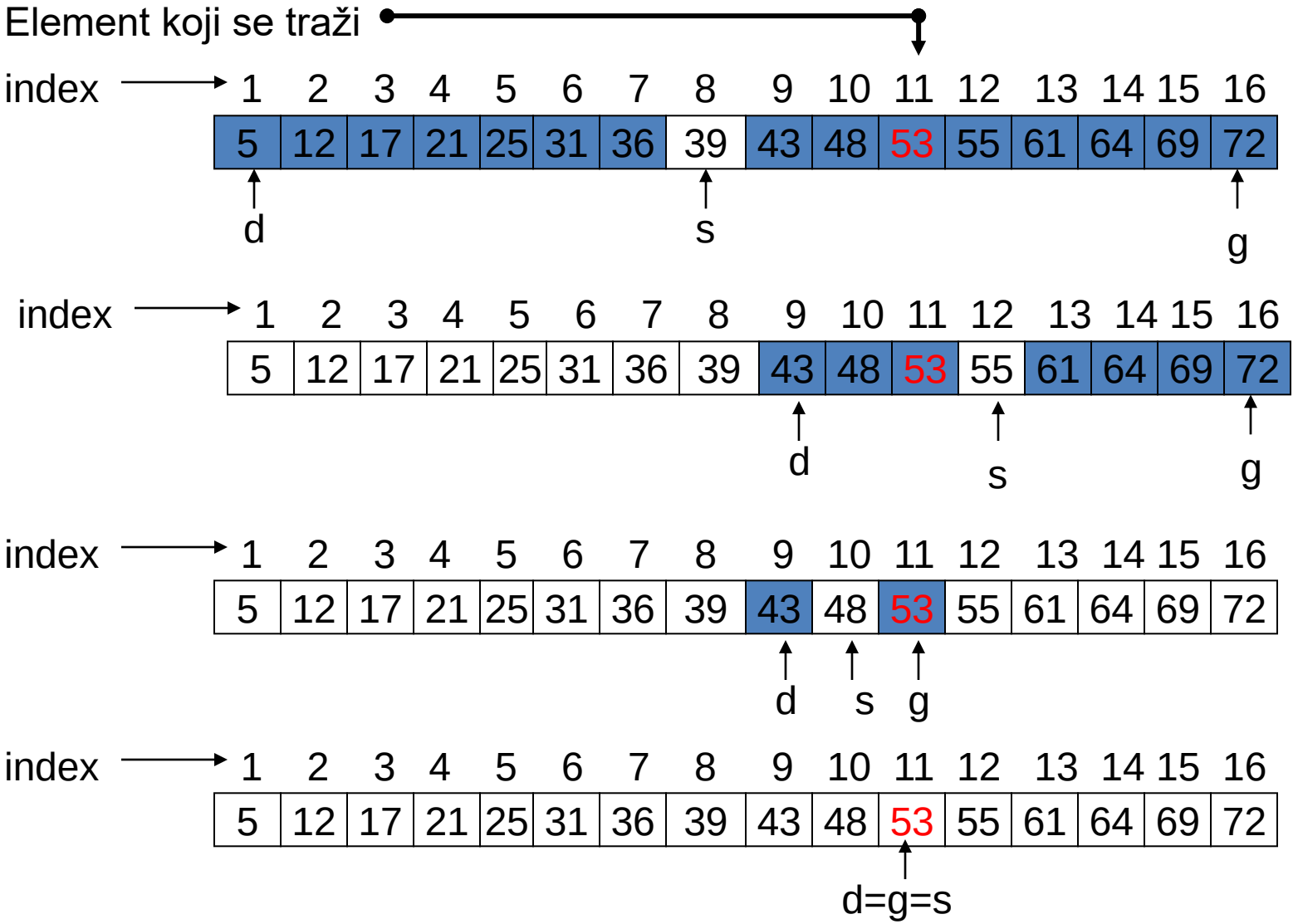
- Omogućava **brzu pretragu polja**
- Preduslov: **sortirano polje**

Algoritam P.3. Binarno traženje
arrayBinarySeach(A,dg,gg,EC)

```
// A je 1D sortirano polje zadato donjom dg i gornjom gg granicom indeksa
// E je element koji se traži
// LOC je lokacija nađenog elementa ili je LOC=NULL
// Ovaj algoritam traži element E u polju A i vraća njegovu lokaciju kao argument
// LOC  ako je traženje uspešno, ili LOC=NULL, ako je traženje neuspešno
1     d ← dg, g ← gg, s ←(d+g)/2 //inicijalizacija segmenta koji se pretražuje
2     while (d≤g and A[s]≠E)
3        if(E<A[s])
4           then g ← s-1
5           else d ← s+1
6        s ← (d+g)/2 //nalaženje sredine segmenta
7     if(A[s]=E) then
          LOC ← s //uspešno traženje
8     else
          LOC ← NULL //neuspešno traženje
9     return LOC
```

Najbolji slučaj: E je u sredini polja
Najgori slučaj: E nije u polju
Asimptotska složenost:
$O(\log_2 n)$

Binarno traženje – Primer



Binarno traženje – Primer 2

A=(5,10,15,20,25), E=20

Inicijalizacija: d=1,g=5,s=3

Prva iteracija: d≤g and A[3]=15 ≠ E=20, 20>15, d=4,s=4

Druga iteracija: d≤g and A[4]=20 izlaz iz petlje, LOC=4

Trazenje uspešno!

A=(5,10,15,20,25), E=8

Inicijalizacija: d=1,g=5,s=3

Prva iteracija: d≤g and A[3]=15 ≠ E=8, 8<15, g=2,s=1

Druga iteracija: d≤g and A[1]=5 ≠ E=8, 8>5, d=2,s=2

Treca iteracija: d≤g and A[2]=10 ≠ E=8, 8<10, g=1,s=1

Četvrta iteracija: d>g izlaz iz petlje, LOC=NULL

Trazenje neuspešno!

SORTIRANJE – METODA MEHURA

Algoritam P.4. Sortiranje polja

arrayBuble(A,n)

// Ovaj algoritam sortira elemente polja A od n elemenata
// u rastući redosled korišćenjem metode mehura

```
1     for k=1,n-1
2       p ← 1 //inicijalizacija pokazivača poređenja
3       while (p≤n-k)
4         if(A[p]>A[p+1]) then
                zameni(A[p],A[p+1])
5         p ← p+1
6       endwhile
7     endfor
b     return
```

Složenost algoritma

Meri se brojem poređenja da bi se polje sortiralo (broj prolaza petljom)

$f(n) = (n-1) + (n-2)+\ldots+2+1 = n(n-1)/2$

Asimptotska složenost:

$O(n^2)$, kvadratna

SORTIRANJE – METODA MEHURA PRIMER

10 15 3 8 4 17 12

Prolaz 1: 10 15 3 8 4 17 12
10 15 3 8 4 17 12
10 3 15 8 4 17 12
10 3 8 15 4 17 12
10 3 8 4 15 17 12
10 3 8 4 15 17 12
10 3 8 4 15 12 17
Broj poređenja n-1 = 6

Prolaz 2: 10 3 8 4 15 12 17
3 10 8 4 15 12 17
3 8 10 4 15 12 17
3 8 4 10 15 12 17
3 8 4 10 15 12 17
3 8 4 10 12 15 17
Broj poređenja n-2 = 5

Prolaz 3: 3 8 4 10 12 15 17
3 8 4 10 12 15 17
3 4 8 10 12 15 17
3 4 8 10 12 15 17
3 4 8 10 12 15 17
Broj poređenja n-3 = 4

Prolaz 4: 3 4 8 10 12 15 17
3 4 8 10 12 15 17
3 4 8 10 12 15 17
3 4 8 10 12 15 17
Broj poređenja n-4 = 3

Prolaz 5: 3 4 8 10 12 15 17
3 4 8 10 12 15 17
3 4 8 10 12 15 17
Broj poređenja n-5 = 2

Prolaz 6: 3 4 8 10 12 15 17
3 4 8 10 12 15 17

Kraj sortiranja

Broj poređenja n-6 = 1

Prednosti i mane Polja

- Prednosti
  - Efikasan **direktni pristup** elementima polja, O(1)
  - Pogodna za obradu elemenata u petljama
- Mane
  - Neefikasne operacije umetanja i brisanja, O(n), gde je n veličina polja
  - Polje je fiksne veličine, a operacija promene veličine polja je skupa
- **Upotreba**
  - Za memorisanje fiksne količine podataka kojima se slučajno pristupa
  - Za formiranje složenijih struktura – niza, magacina, redova, dekova, tablica, gomila, ...

Pitanja, ideje, komentari