Algoritmi i strukture podataka

Studijski programi:
Softversko inženjerstvo
Računarska tehnika
Matematika-informatika

Ulfeta Marovac

O strukturama podataka

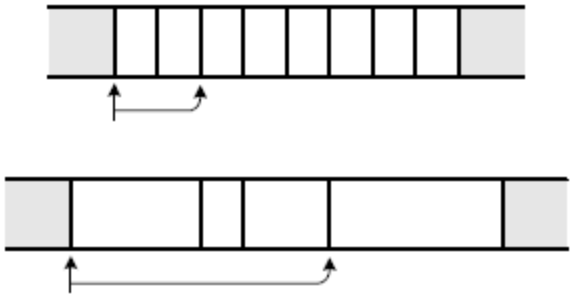
- Osnovna svrha algoritma je da definiše kako se upravlja nekim podacima i vrše njihove transformacije.
- Tako algoritmi i strukture predstavljaju dva neodvojiva dela koje nema smisla posmatrati odvojeno.
- Osnovna podela struktura podataka je na
  - **Linearne**-ukoliko je element strukture u relaciji samo sa dva elementa strukture prethodnikom i sledbenikom
  - **Nelinearna**-ako su medjusobni odnosi elemenata strukture složeni pa jedan element strukture moze biti u vezi sa više drugih elementa strukture.

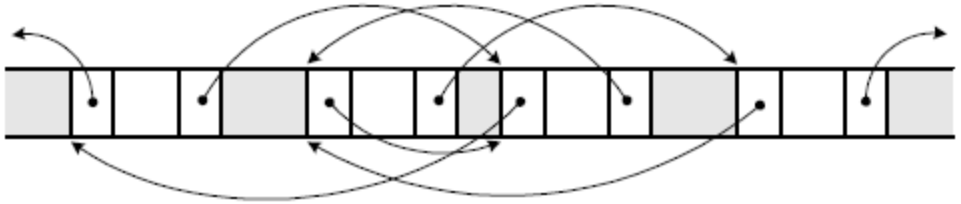
O strukturama podataka

- Strukture možemo deliti i na:
  - Statičke i
  - Dinamičke
- Statičke strukture imaju fiksnu veličinu koja se određuje pri prevođenju
- Dinamičke strukture mogu da menjaju svoju veličinu tokom izvršavanja programa

Memorijska reprezentacija

- Projektovanje strukture zahteva i da se definiše i fizička implementacija strukture podataka u memoriji.
- Postoje dva osnovna načina predstavljanja:
  - Sekvencijalna reprezentacija
  - Ulančana reprezentacija.





Memorijska reprezentacija

- Sekvencijalne reprezentacije elementi strukture se smeštaju jedan za drugim u kontinualnom prostoru, tako da su fizički i logički poredak elemenata isti. Za alokaciju ovakve strukture potrebno je znati ukupan broj elemenata i njihove veličine. Pristup svakom elementu je direktan i zahteva poznavanje početne adrese .
- Ulančane reprezentacije elementi su raspoređeni u nekontinualnom prostoru, na proizvoljnim mestima u memoriji a njihov fizički poredak nema nikakve veze sa logičkim poretkom. Pošto ovde logičko sredstvo ne moze biti izraženo pozicijom elementa mora da postoji drugi način da se izraze veze elemenata u strukturi i to su *pokazivači*.

Pokazivači

- Omogućuju da program u toku rada zahteva dodatnu memoriju od sistema i oslobađa tu naknadno zahtevanu memoriju.
- Vrednost jednog pokazivača je adresa.
- Adresa je zajednička karakteristika svih tipova podataka i kategorija u programu.
- Pokazivači nam omogućavaju dinamičku dodelu memorije i pristup toj dinamički dodeljenoj memoriji preko njene adrese.
- Deklaracija pokazivača:

```
var
 pok: ^integer;
```

```
type
 pokazivac = ^integer;
var
 pok: pokazivac;
```

Pokazivači

- Povezivanje pokazivača i lokacije čiju adresu sadrži pokazivač omogućava dinamičku dodelu memorije.
- Dinamička memorija se zove i heap.
- Pokazivač nam omogućava da u određenom trenutku zahtevamo od sistema da nam dodeli lokaciju iz ostatka memorije, ali pri tome se mora znati koliko memorije i za šta.
- Pascal program se u tom slučaju obraća tzv. upravljaču heapa koji pronalazi gde postoji slobodna memorija koja se traži, označava tu lokaciju kao zauzetu i upisuje njenu adresu u pointer.
- Sa upravljačem heapa se komunicira uz pomoć procedura.
- Procedura new(pok), gde je pok tipizirani pokazivač.
- Posle izvršavanja procedure new, možemo da koristimo promenljivu pok^.

Pokazivači

- Kad nam ova lokacija više nije potrebna, javljamo da se označi kao slobodna procedurom dispose(pok). Posle njenog izvršavanja nam pok^ više nije na raspolaganju, tj. sadržaj pokazivača je nil.
- Pokazivač bilo koje vrste može da uzme vrednost nil. U tom slučaju pokazivač ne pokazuje ni na šta.
- Posle izvršavanja procedure new(pok), ne smemo promenljivoj pok dodeliti vrednost nil mi sami. Desiće se da će se prekinuti veza između pokazivača i lokacije, ali da će lokacija ostati označena kao zauzeta. Zbog toga moramo izvršiti i proceduru dispose, da bi se lokacija korektno oslobodila.

Primer 1.

- Neka su date deklaracije :

```
var
 p,q:^integer;
 x:integer;
```

- Objasniti i grafički prikazati sledeće naredbe

a)

```
new(p); p^:=1; q:=p;
writeln(p^,q^);
x:=2; q^:=x;
writeln(p^,q^);
new(p); p^:=3;
writeln(p^,q^);
```

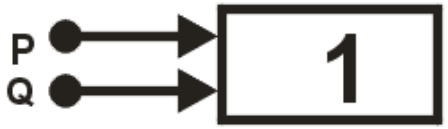
b)

```
new(p); p^:=1:
new(q); q^:=2;
dispose(p);
p:=q
new(q); q^:=3;
writeln(p^,q^);
```

```
new(p);
```
Alocira prostor

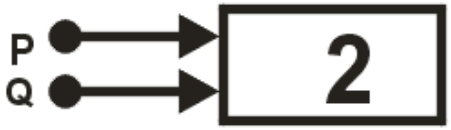
```
p^:=1;
```
p pokazuje na vrednost 1

```
q:=p;
```
q pokazuje na isti prostor kao p




```
writeln(p^,q^);

x:=2;
```

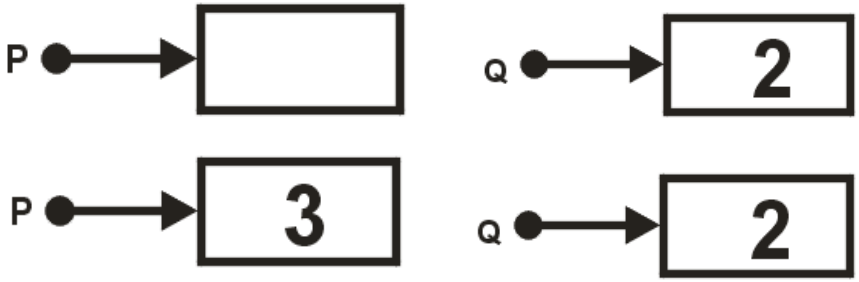



```
q^:=x;
```
p i q pokazuju na vrednost 2

```
writeln(p^,q^);

new(p);
```
Alocira se novi prostor za p

```
p^:=3;
```
p pokazuje na vrednost 3




```
writeln(p^,q^);
```

`new(p);` Alocira prostor za p

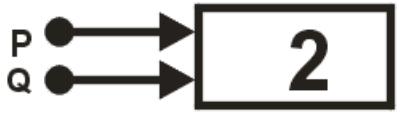
`p^:=1;` p pokazuje na vrednost 1

`new(q)` alocira se prostor q

`q^:=2;` q pokazuje na vrednost 2

`dispose(p);` Oslobađa se prostor od promenljive p

`p:=q` I p i q pokazuju na vrednost 2



`new(q)` alocira se novi prostor za q

`q^:=3;` q pokazuje na vrednost 3


```
writeln(p^,q^);
```

Zadatak 2.

- Odredite koje su naredbe korektne ako su date sledeće deklaracije.

```
type
 intpokazivac=^integer;
 chpokazivac=^char;
var
 pok1,pok2:intpokazivac;
 pok3,pok4:chpokazivac;

a)  new(pok2)
b)  new(pok2^)
c)  pok2:=pok4;
d)  pok2^:=pok2^+pok1^
e)  pok1:=nil
f)  pok3^:=nil
g)  writeln(pok2,pok3)
h)  readln(pok2^)
j)  pok2:=new(pok1)
k)  dispose(pok3)
```

Rešenje

```
type
 intpokazivac=^integer;
 chpokazivac=^char;
var
 pok1,pok2:intpokazivac;
 pok3,pok4:chpokazivac;

a)  new(pok2)
b)  new(pok2^)
c)  pok2:=pok4;
d)  pok2^:=pok2^+pok1^
e)  pok1:=nil
f)  pok3^:=nil
g)  writeln(pok2,pok3)
h)  readln(pok2^)
j)  pok2:=new(pok1)
k)  dispose(pok3)
```

Zadatak 3.

- Odrediti ispis sledećeg programskog segmenta ako su date definicije i opisi:

```
type
 intpokazivac=^integer;
 chpokazivac=^char;
var
 pok1,pok2:intpokazivac;
 pok3,pok4:chpokazivac;
```

```
new(pok1);
new(pok2);
new(pok3);
pok2:=pok1;
pok1^:=1; pok2^:=2; pok3^:='B';
writeln(pok1^,pok2^,pok3^);
```

Rešenje: 22B

Zadatak 4.

- Da li je ispravan sledeći kod

```
var
 pok1:^integer;

new(pok1);
read(pok1^); writeln(pok1^);
dispose(pok1); writeln(pok1^);
```

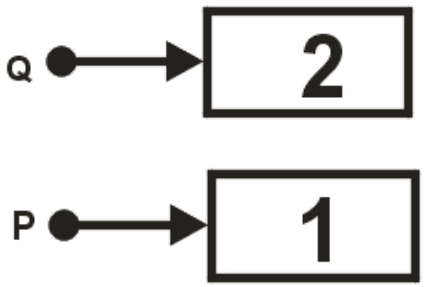
Iza dispose ne postoji promenljiva pok1 ne pokazuje ni na jednu lokaciju

Zadatak 5.

- Šta se dobija kao rezultat?

```
type
 pok=^integer;
var
 p,q:pok;

 p^:=q^;
 if p=q
   then p:=nil
   else if p^=q^
    then q:=p;
 if p=q then q^:=4;
 writeln(p^);
```



Rešenje: 4

Zadatak 6.

```
program Test;
 type
  ucenik=record
   broj:integer;
   odeljenje:char;
  end;
  pokazivac=^ucenik;
 var
  p1,p2:pokazivac;
 begin
  new(p1); p1^.broj:=1; p1^.odeljenje:='a';
  writeln(p1^.broj, p1^.odeljenje);
  new(p2); p2^.broj:=2; p2^.odeljenje:='b';
  writeln(p2^.broj, p2^.odeljenje);
  p1:=p2;
  p2^.broj:=3;
  p2^.odeljenje:='c';
  writeln(p1^.broj,p1^.odeljenje,p2^.broj,p2^.odeljenje);
 end.
```

Zadatak 6.

- Šta se ispisuje izvršavanjem sledećeg programa. Slogovi sadrže: redni broj učenika u odeljenju, i oznaku odeljenja (a,b,c)

Rešenje:

1a

2b

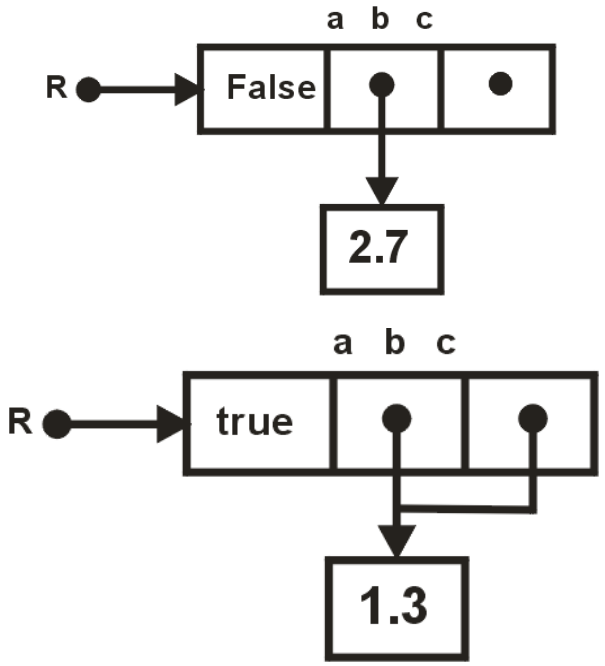
3c3c

Zadatak 7.

- Ako promenljiva r ima vrednost kao na slici, nacrtati strukturu vrednosti promenljive r posle izvršenja sledećih naredbi:

```
type
 pok=record
  a:boolean;
  b,c:^real;
 end;
var
 r:^pok;

 if r^.b<>nil
  then r^.c:=r^.b;
 r^.b^:=r^.c^-1.4;
 r^.a:=r^.b=r^.c;
```



Nizovi

- Niz je osnovni struktuirani tip podataka koji postoji u praktično svim višim programskim jezicima
- Najviše korišćeni složeni tip sa širokim spektrom primena
- Ovaj tip se nekad koristi i za simulaciju i implementaciju nekih drugih struktura, pogotovu linearnih ali i nelinearnih

Vrste nizova

- Niz je linearno uređena struktura koja se sastoji od konačnog broja homogenih elemenata
- Osobina homogenosti znači da su svi elementi niza istog skalarnog ili struktuiranog tipa
- Osobina uređenosti znači da se tačno zna ko je prvi, drugi, i tako do poslednjeg elementa niza
- Po strukturi nizovi mogu biti **jednodimenzionalni** i **višedimenzionalni** nizovi

Jednodimenzionalni nizovi

- Po svojoj stukturi najjednostavniji su **jednodimenzionalni nizovi – vektori**
- Za definisanje mesta pojedinog elementa u ovakvom nizu se koristi **indeks** kao pozicija elementa u okviru niza
- Niz predstavlja preslikavanje iz skupa vrednosti indeksa na skup elemenata niza, tako da svakom indeksu odgovara jedan element i obratno

Jednodimenzionalni nizovi

- Indeks može biti svakog onog tipa koji jednoznačno mapira na skup celih brojeva (logički, znakovni, nabrojivi,..) ali se najčešće koristi ceo broj
- Ako opseg indeksa niza X pokriva celobrojne vrednosti između donje granice *l* i gornje granice *u.* Tada se on može predstaviti:

$$X[l{:}u] = \{X[i]\},\ i = l, l+1, \ldots, u-1, u\ .$$

- Broj elemenata u ovako definisanom nizu je *u*-*l*+1. Donja granica *l* je obično 0 ili 1.

Višedimenzionalni nizovi

- **Višedimenzionalni nizovi** predstavljaju generalizaciju jednodimenzionalnih nizova
- Npr. dvodimenzionalni niz zvani **matrica** se može smatrati jednodimenzionalnim nizom čiji su elementi jednodimenzionalni nizovi
- U opštem slučaju, niz sa *n* dimenzija ima *n* opsega indeksa, po jedan za svaku dimenziju, pa se može predstaviti kao:

$$X[l_1:u_1,\ l_2:u_2,\ldots,\ l_n:u_n]\ .$$

Višedimenzionalni nizovi

- Broj elemenata u ovom nizu se može izračunati kao proizvod veličina opsega po pojedinim dimenzijama

$$\prod_{i=1}^{n} (u_i - l_i + 1) .$$

- Za selekciju elemenata niza potrebno je naznačiti n indeksa od kojih svaki označava relativnu poziciju u ogdovarajućoj dimenziji

$$X[i_1, i_2, \ldots, i_n]$$

gde je $l_k \leq i_k \leq u_k$ za $1 \leq k \leq n$.

Operacije sa nizovima

- Prilikom deklarisanja niza u višem programskom jeziku se obavezno navodi ime i broj dimenzija niza, kao i tip elemenata niza.
- Ako je niz statički, za svaku dimenziju niza se navodi i opseg indeksa, jer se prema tome rezereviše prostor pri prevođenju
- Kod dinamičkih nizova broj elemenata varira u vremenu izvršavanja programa

Operacije sa nizovima

- Osnovna operacija sa nizom je **selekcija** ili **ekstrakcija** elemenata, koja se vrši navođenjem imena niza i po jedne vrednosti indeksa za svaku dimenziju.
- Prilikom selekcije elemenata, indeksi se mogu zadavati kao konstante, promenljive ili kao izrazi
- Pritom se pretpostavlja da se izračunate vrednosti nalaze u dozvoljenom, deklarisanom opsegu po odgovarajućoj dimenziji *(primer)*

Operacije sa nizovima

- **Selekcija** se vrši u cilju manipulacije sa individualnim elementima niza (čitanje ili upisivanje) a dozvoljene opracije su određene tipom samog elementa.
- Da bi se obradilo više elemenata na isti način obično se koriste petlje čija se kontolna promenljiva uzima direktno kao indeks ili služi za njegovo izračunavanje
- Npr. Rotacija vektora X[1:n] oko srednjeg elementa gde zamenjuju mesta prvi i poslednji element, drugi i pretposlednji, itd., vrši se sledećom petljom:

for $i = 1$ **to** $n/2$ **do**

$X[i] \leftrightarrow X[n - i + 1]$

end_for

Operacije sa nizovima

- U nekim programskim jezicima postoje nizovski operatori koji se primenjuju na sve elemente i tako olakšavaju obradu čitavog niza
- Npr. Operacija X=X+10 dodaje 10 na svaki element niza X.

Predstavljanje nizova u memoriji

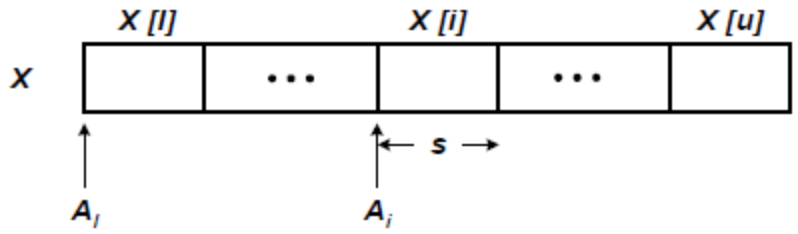
- Za predstavljanje nizova u memoriji koristi se skoro isključivo sekvencijalna reprezentacija, jer ona predstavlja prirodan način implementacije linearne liste
- Zato se i ne razdvaja logićki koncept niza od njegove implementacije
- Niz se, dakle, u memoriji implementira kao kontinualni skup susednih memorijskih lokacija

Predstavljanje nizova u memoriji

- Zbog ove sekvencijalne alokacije je izuzetno nepogodno umetati novi element na proizvoljnu poziciji i brisati ga sa proizvoljne pozicije
- I u slučaju umetanja i brisanja, menjaju se indeksi svih elemenata od pozicije na kojoj je izvršeno umetanje ili brisanje.

Smeštanje vektora

- Najjednostavnije se implementira jednodimenzionalni niz ili vektor.
- Ako je za smeštaj jednog elementa potrebno **s** memorijskih reči, onda je za smeštaj čitavog vektora X[l:u], potrebno (u-l+1)*s reči



Predstavljanje vektora u memoriji

- Elementi su fizički poređani po redosledu njihovih indeksa, što omogućava jednostavan i ravnopravan pristup svakom elementu

Smeštanje vektora

- Ako je početna adresa niza $\mathbf{A_l}$, onda se za pristup elementu X[i] koristi linearna fuknkcija koja zavisi od početne adrese, indeksa i veličine elementa

$$\mathbf{A_i = A_l + (i-l)s}$$

- Iskorišćenje prostora kod sekvencijalne reprezentacije je optimalmo a pristup efikasan ako je veličina elementa s jednaka jednoj memorijskoj reči ili njenom celom multiplu
- Problemi nastaju ako **s** nije ceo broj memorijskih reči. Tada se postupa na dva načina koji imaju implikacije na iskorišćenje prostora i efikasnost operacija

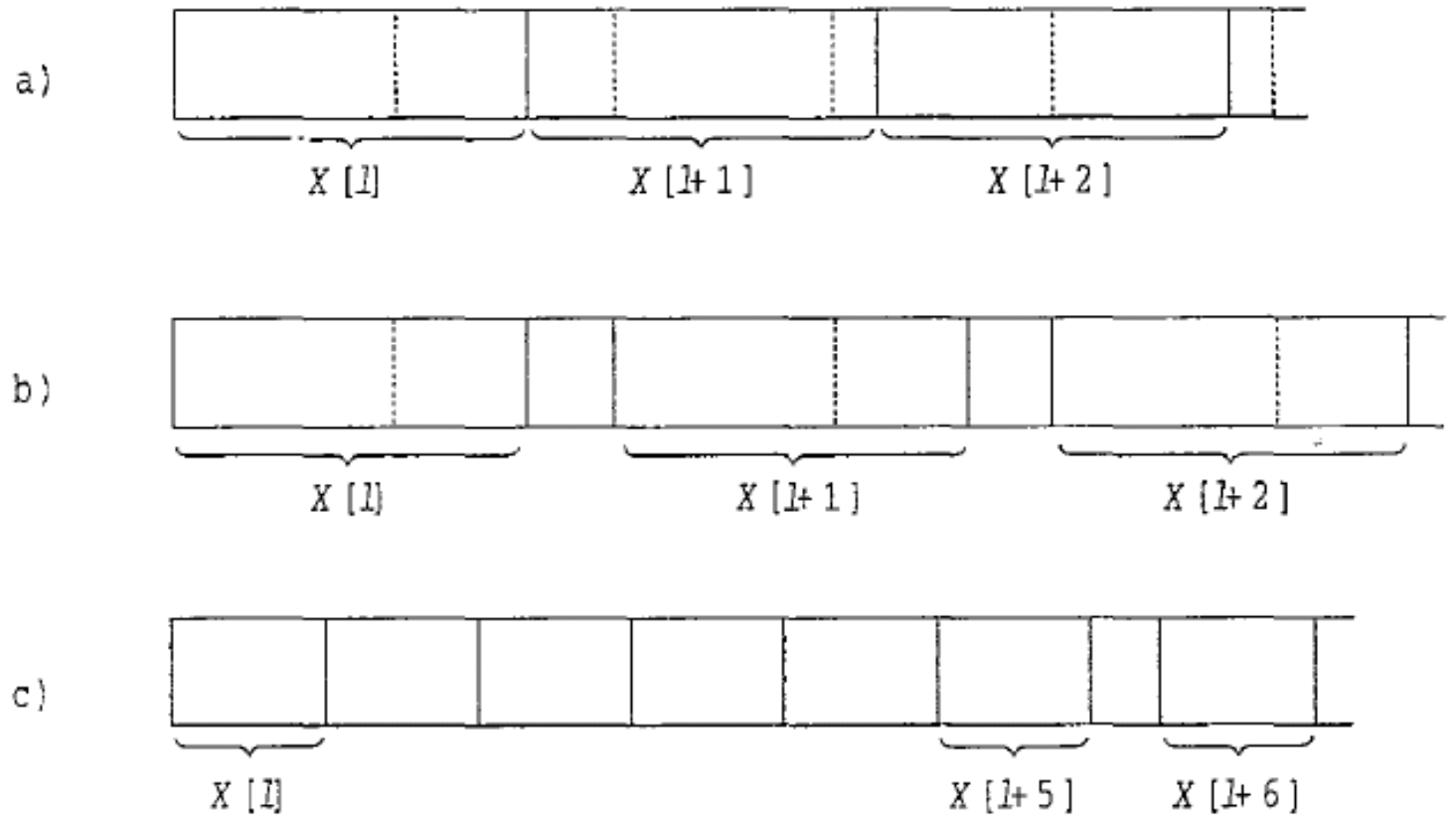
Kontinualno smeštanje vektora

- Ako je iskorišćenje prostora prioritetni cilj, onda se elementi **kontinualno** smeštaju jedan za drugim.
- Ovde se potpuno iskorišćenje plaća neefikasnošću pristupa.
- Slika a), s=1.6 memorijskih reči
- Element X[l+1] se proteže preko tri memorijske reči
- Da bi se elemenat dohvatio, treba pročitati sve tri reči i izvesti potrebne ekstrakcije, pomeranja, spajanja, itd.

Kontinualno smeštanje vektora

- Pored toga, ovaj mehanizam ne mora biti uniforman za svaku komponentu
- Npr. X[I+2] je smešten u dve reči i to na različitim mestima u odnosu na predhodni slučaj
- Znači, pristup elementima kontinualno alociranog niza zahteva:
  - dodatne instrukcije i
  - povećava izvršni kod
- Time se umanjuju prednosti dobro iskorišćenog prostora, a svakako se gubi i na vremenu.

Smeštanje vektora – tri načina



Različiti načini sekvencijalne alokacije niza: a) bez dopunjavanja, $s = 1.6$, a) sa dopunjavanjem, $s = 1.6$, i c) pakovanje, $s = 0.15$

Smeštanje vektora dopunjavanjem

- Smeštanje **dopunjavanjem (padding)** je prikazano na slici b)
- Kod ovog načina se za smeštanje elementa uzima [s] memorijskih reči
- Preostali prostor u poslednjoj memorijskoj reči koju elemenat zauzima do početka naredne reči ostavlja neiskorišćenim
- Sledeći elemenat uvek počinje od početka memorijske reči
- Operacije za pristup elementu su *uniformne* i *efikasnije*, sa nešto *slabijim iskorišćenjem prostora*

Smeštanje vektora dopunjavanjem

- Iskorišćenje se može izračunati kao odnos stvarne veličine elementa i veličine prostora koji je za njega rezervisan $S_u = s / \lceil s \rceil$.
- Kad je Su>0.5, obično se u ovakvim slučajevima usvaja način smeštanja sa dopunjavanjem

Smeštanje vektora tehnikom pakovanja

- Ako je s<0.5, faktor iskorišćenja prostora se može popraviti stavljanjem više od jednog elementa u jednu memorijsku reč primenom tehnike **pakovanja**
- Slika c) za s=0.15 memorijskih reči
- U jednu reč stane 6 elemenata
- Ako se u jednu reč stavlja k elemenata. iskorišćenje se popravlja na $S_u = ks / \lceil ks \rceil$ gde je $k = \lfloor 1/s \rfloor$.

Smeštanje matrica

- Zahvaljujući tome što je memorija jednodimenzionalni niz reči sa linearno rastućim adresama, vektor se u memoriju smešta na prirodan način, gde su logički i fizički redosled elemenata isti
- Pošto je matrica dvodimenzionalna struktura, za njeno smeštanje u memoriju se mora uraditi **linearizacija**
- Dvodimenzionalni logički poredak elemenata treba prevesti u jednodimenzionalni fizički poredak
- Za linearizaciju postoje dva osnovna metoda:
  - Smeštanje po vrstama i
  - Smeštanje po kolonama

Smeštanje po vrstama

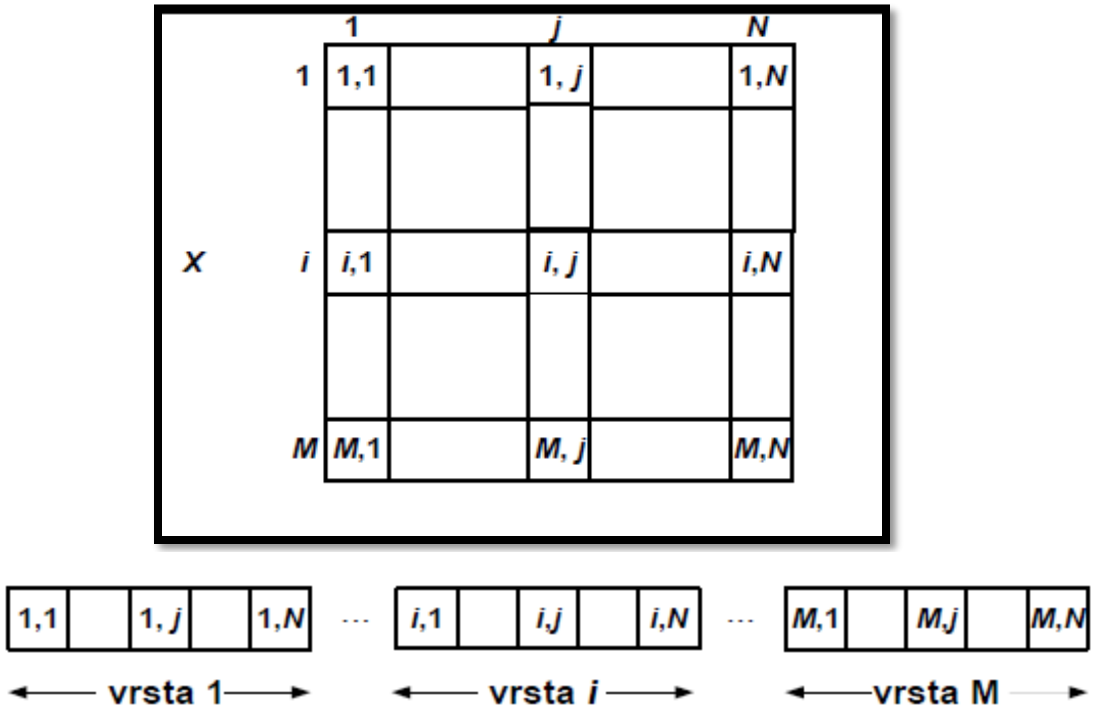
- **Smeštanje po vrstama** (row-major) u alociranom prostoru od početne adrese smešta prvu vrstu, pa drugu, i tako do poslednje.
- Vrsta(red) se tretira kao zapis koji ima onoliko elemenata koliko ima kolona
- Za matricu X[1:M,1:N] adresna funkcija za pristup proizvoljnom elementu X[i,j] svodi na

$$A_{i,j} = A_{1,1} + ((i - 1)N + j - 1)s .$$

- Za matricu X[1:M,1:N] smeštanje po vrstama je ilustrovano na slici b)

Smeštanje po vrstama

- Za matricu X[1:M,1:N] smeštanje po vrstama je ilustrovano na slici:



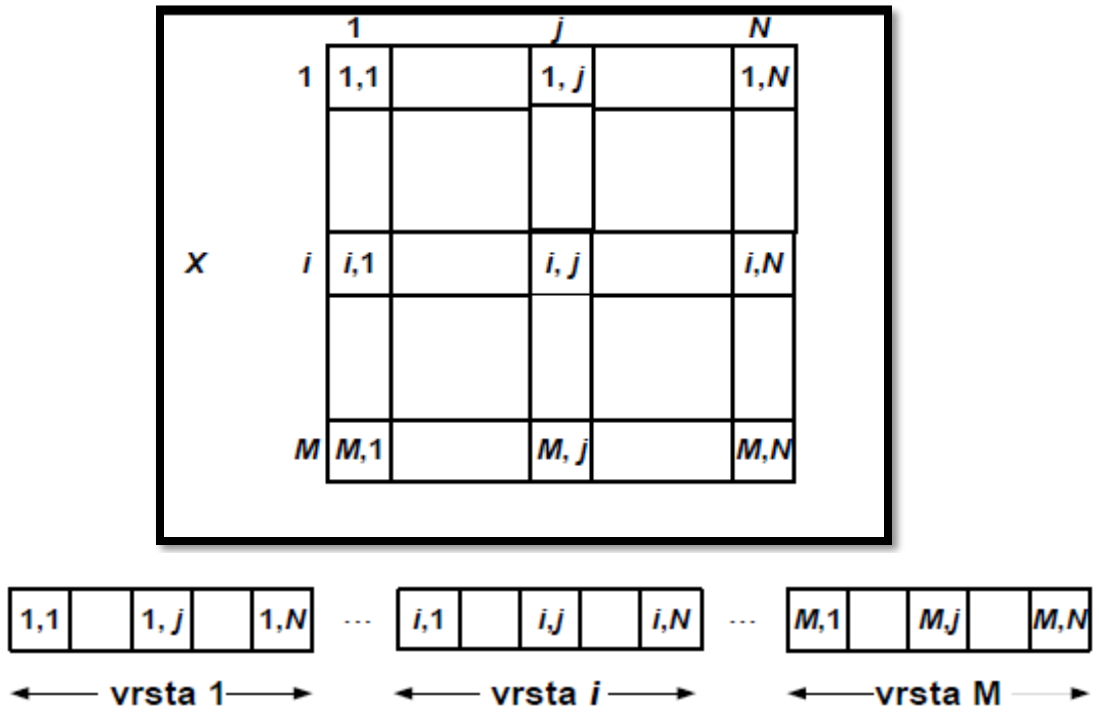
Smeštanje po kolonama

- **Smeštanje po kolonama** (column-major), koristi suprotan način linearizacije
- Od početne adrese se smešta prva kolona, zatim druga i tako redom do poslednje
- Za matricu [1:M,1:N] adresna funkcija je
- Pri smeštanju po kolonama, prvo varira levi indeks(broj vrste) elementa a zatim desni

$$A_{i,j} = A_{1,1} + ((j - 1)M + i - 1)s .$$

Smeštanje po kolonama

- Smeštanje po kolonama je ilustrovano na slici :



Smeštanje višedimenzionalnih nizova

- Načini smeštanja po vrstama i kolonama mogu da se uopšete i na slučaj višedimenzionalnih nizova.
- Ovde je pojam vrste i kolone nešto generalniji
- Geometrijska analogija i grafičko predstavljanje ovakvih nizova sa više od tri dimenzije su veoma otežani
- Ako je n-dimenzionalni niz $X[1{:}u_1, 1{:}u_2, .., 1{:}u_n]$ smešten po vrstama

Smeštanje višedimenzionalnih nizova

- Tada se elementiređaju jedan za drugim u sledećem poretku

$$X[1, \ldots, 1, 1] \quad X[1, \ldots, 1, 2] \quad \ldots \quad X[1, \ldots, 1, u_n]$$

$$X[1, \ldots, 2, 1] \quad X[1, \ldots, 2, 2] \quad \ldots \quad X[1, \ldots, 2, u_n]$$

$$\ldots$$

$$X[u_1, \ldots, u_{n-1}, 1] \; X[u_1, \ldots, u_{n-1}, 2] \quad \ldots \quad X[u_1, \ldots, u_{n-1}, u_n] \; .$$

Smeštanje višedimenzionalnih nizova

- Poslednji indeks najbrže varira a prvi najsporije.
- Da bi došli do proizvoljnog $X[i_1,i_2,..,i_n]$ n-dimenzionalnog niza smeštenog u memoriji po vrstama treba prvo proći i_1-1 hiper - ravni sa po $u_2u_{3...}u_n$ elemenata da bi dosli do $X[i_1,1,..,1]$.
- Zatim je neophodno proći i_2-1 hiper -ravni sa po $u_3u_{4...}u_n$ elemenata da bi dosli do $X[i_1, i_2,..,1]$.
- Ovakav postupak se ponavlja kroz n-1 dimenziju sve dok se ne dođe do elementa $X[i_1, i_2,...,i_{n-1},1]$ odakle treba preći i_n-1 elemenata da bi se došlo do traženog elementa.

Smeštanje višedimenzionalnih nizova

- Adresna funkcija glasi:

$$A_{i1,..,\,in} = A_{1,..,1} + ((i_1 - 1)u_2u_3...u_n + (i_2 - 1)\,u_3u_4...u_n + ... + (i_{n-1} - 1)u_n + i_n - 1)s$$

- Efikasno izračunavanje adresne funkcije:

```
offset = 0
for j = 1 to n do
    offset = U [j ] offset + l [j ] - 1
end_for
A = A1,..,1 + s * offset
```

Optimizacije pri smeštanju nizova

- Način smeštanja nizova po vrstama i kolonama su izabrani tako da se omogući *efikasan pristup* proizvoljnom elementu niza izračunavanjem *adresne funkcije*
- Adresna funkcija predstavlja linearnu funkciju indeksa elementa
- Za svaki element je predviđeno tačno određeno mesto u memoriji koje zavisi samo od njegovog indeksa a ne i od njegove vrednosti

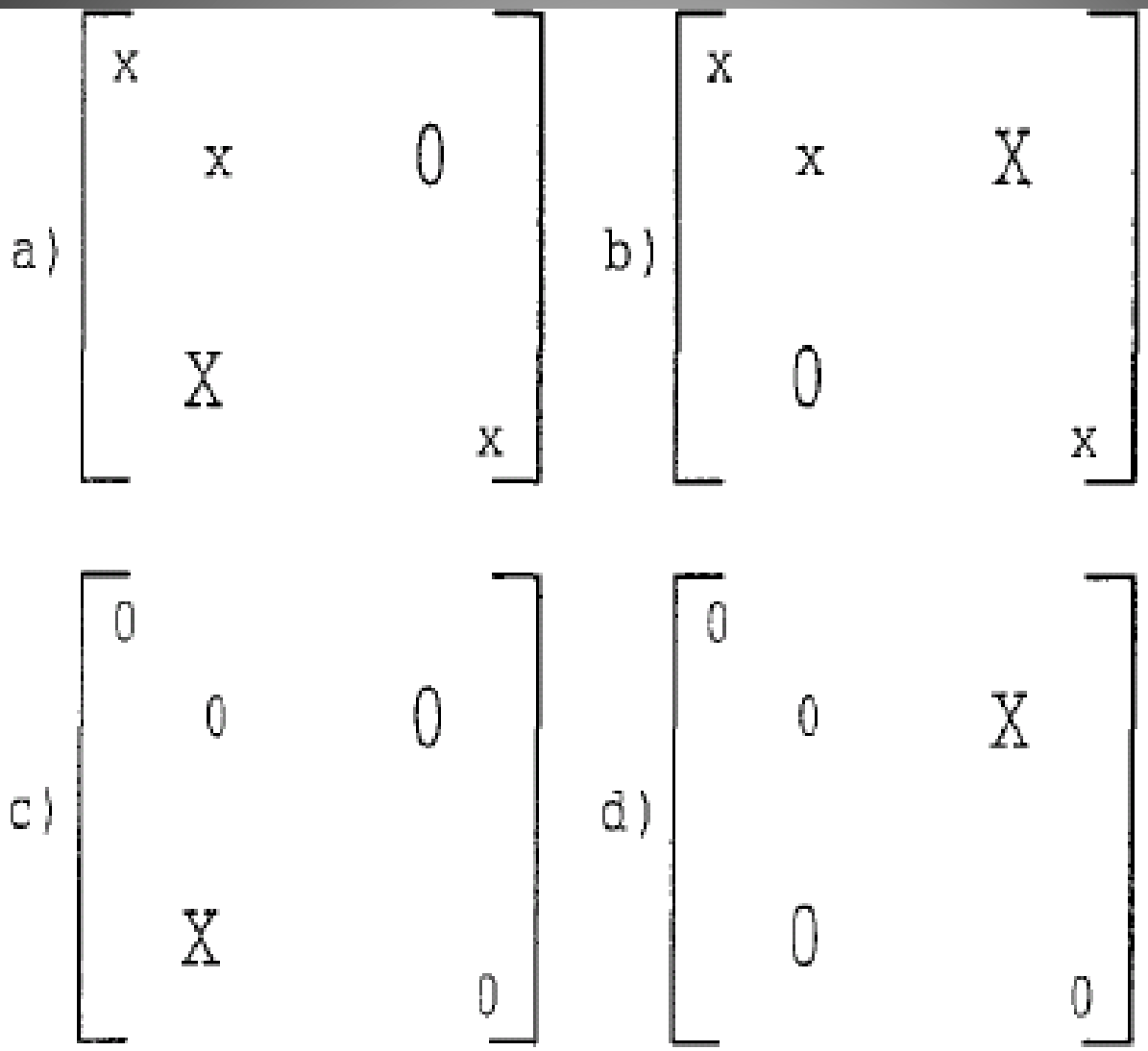
Posebne vrste nizova

- *Posebne vrste nizova* - kod kojih značajan broj elemenata ima nultu vrednost, pa se optimizacijom može ostvariti iskorišćenje prostora tako što bi se pamtile samo vrednosti
- Razmatraćemo dve vrste ovakvih niizova: **trougaone matrice** i **retki nizovi**

Trougaone matrice

- Trougaona matrica je kvadratna matrica kod koje su svi elementi iznad ili ispod glavne dijagonale jednaki nuli
- Data kvadratna matrica X[1:n, 1:n]
- Ako je X[i,j]=0 za i<j, **donja trougaona** matrica
- Ako je X[i,j]=0 za i>j, **gornja trougaona** matrica
- Ako je X[i,j]=0 za i≤j, **strogo donja trougaona** matrica
- Ako je X[i,j]=0 za i≥j, **strogo gornja trougaona** matrica

Trougaone matrice



Trougaone matrice: a) donja, b) gornja, c) strogo donja i d) strogo gornja

Trougaone matrice

Npr. donja trougaona matrica
$X[i,j] = 0$, $i < j$

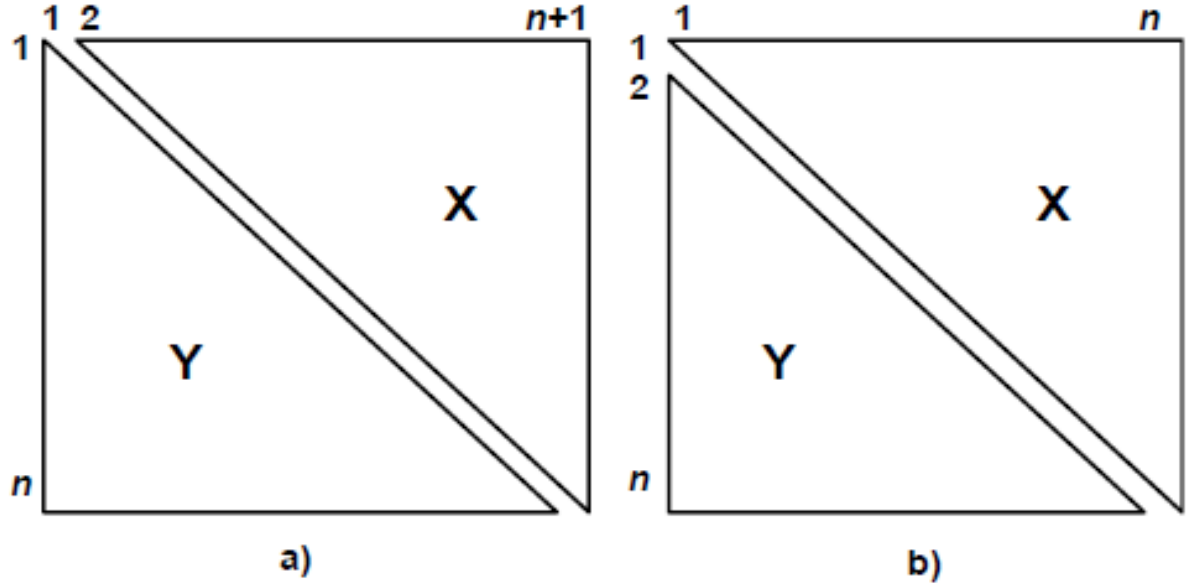
Primena smeštanja po vrstama

$X[1,1]$ $X[2,1]$ $X[2,2]$ $X[3,1]$ $X[3,2]$ $X[3,3]$ ...

Ušteda u prostoru oko 50%

$A_{i,j} = A_{1,1} + (i(i - 1)/2 + j - 1)s$ ako je $i \geq j$

Trougaone matrice



- $X[i,j] = Z[i,j+1]$ ako je $i \leq j$ $\quad X[i,j] = 0$ za $i > j$
- $Y[i,j] = Z[i,j]$ ako je $i \geq j$ $\quad Y[i,j] = 0$ za $i < j$

Retki nizovi

- Dok se kod trougaonih matrica pri smeštanju može uštedeti oko 50% prostora, a postoje nizovi gde ušteda može da bude još veća
- To su retki nizovi kod kojih postoji relativno veliki broj nenultih elemenata
- Definicija nije precizna, nema tačnog kriterijuma za procenat nenultih elemenata
- Niz koji ima preko 80% ili 90% nenultih elemenata se može nazvati retkim

Primer retke matrice

$$X = \begin{bmatrix} 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 11 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 15 \end{bmatrix}$$

Retki nizovi

- Postoji nekoliko načina za optimizaciju pri alokaciji prostora za retke nizove i svi se zasnivaju na smeštanju samo nenultih elemenata
- Ove tehnike se mogu koristiti i kada većina elemenata ima neku drugu nenultu, ali istu vrednost
- Tada se elementi sa ovom vrednošću ne pamte nego podrazumevaju a pamte se elementi sa različitim vrednostima od podrazumevane

Vektorska reprezentacija retkih nizova

- Vektorska reprezentacije jednog n- dimenzionalnog niza X pamti sve nenulte elemente
- Jednom ovakvom elementu odgovara jedan element vektora koji ima sledeći format
- Odnos prostora koji zauzimnaju vektorska reprezentacija i standardni način smeštanja:

$$(n+1)n_{nz}/(u_1u_2...u_n).$$

Vektorska reprezentacija retkih nizova

(sa jednim vektorom zapisa od po tri polja)

Vektorska reprezentacija matrice X sa slike prikazana je na slici desno
Polje R sadrži indeks vrste
Polje C indeks kolone
Polje V vrednost nenultog elementa

$$X = \begin{bmatrix} 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 11 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 15 \end{bmatrix}$$

R	C	V
1	3	4
2	4	5
2	6	11
4	1	9
4	4	8
5	7	15

Vektorska reprezentacija retkih nizova (sa tri posebna vektora)

Za indekse svake dimenzije se koriste posebni vektori različitih dimenzija
Vektor V sadrži vrednosti nenultih elemenata
Vektor C sadrži indekse odgovarajućih kolona
Vektor R sadrži indekse ulaza u vektoru C koji odgovaraju prvim nenultim elementima u odgovarajućim vrstama

$$X = \begin{bmatrix} 0 & 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 11 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 15 \end{bmatrix}$$