Strukture podataka

Predmet: Uvod u Algoritme 17 - ESI053
Studijski program: Primenjeno softversko inženjerstvo

DEPARTMAN ZA RAČUNARSTVO I AUTOMATIKU
ccd
DEPARTMAN ZA ENERGETIKU, ELEKTRONIKU I KOMUNIKACIJE
DEET

Skupovi

- Skup (*set*) predstavlja kolekciju neuređenih elemenata
- U računarstvu skupovi su promenljivi u vremenu – dodaju im se i brišu elementi – **dinamički** su.
- Algoritmi zahtevaju nekoliko tipova operacija sa skupovima
  - Dodavanje elemenata
  - Brisanje elemenata
  - Test pripadnosti
  - ...
- Elementi dinamičkih skupova mogu imati pridružene ključeve, dodatne (satelitske) podatke, a takođe im se tipično pristupa preko pokazivača (elementi su često predstavljeni objektima)

Operacije sa skupovima

- Razlikujemo dva skupa operacija:
  1. Upiti nad skupom
     - pretraga skupa S po ključu k – PRETRAŽI(S,k) - *Search*
     - vraća elemenat sa najmanjim ključem – MINIMUM(S) - *Minimum*
     - ... sa najvećim ... – MAKSIMUM(S) - *Maximum*
     - vraća sledeći elemenat od x – NAREDNI(S,x) - *Successor*
     - ... prethodni elemenat ... – PRETHODNI(S,x) - *Predecessor*
  2. Operacije izmena skupa
     - dodaje elemenat x u skup S – DODAJ(S,x) – *Insert*
     - briše elemenat x – OBRIŠI(S,x) - *Delete*

Elementarne strukture podataka

- Elementarne strukture:
  - Stek (*Stack*)
  - Red (*Queue*)
  - Povezane liste (*List*)
  - Stabla (*Tree*)

- Stek i red su dinamički skupovi gde se elementi uklanjaju unapred određenim redosledom
  - Kod steka se briše poslednji (najmlađi) dodat elemenat.
    Naziva se LIFO procesiranje (*last-in, first-out*)
  - Kod reda se briše prvi (najstariji) dodat elemenat.
    Naziva se FIFO procesiranje (*first-in, first-out*)

Stek

- Operacije:
  - dodavanje elementa, “guranje” u stek – GURNI – (***Push***)
  - uklanjanje i preuzimanje (brisanje) elementa, povlačenje – POVUCI – (***Pop***)
  - Provera: da li ima elemenata?
- Podaci: može se predstaviti nizom S[1..S.*vrh*] koji sadrži elemente
  - S[1] je elemenat na dnu, a S[S.*vrh*] je na vrhu steka (poslednji dodat).
  - S.*vrh* + 1 je indeks prvog praznog mesta na steku. Ako je S.*vrh* == 0, onda je stek prazan
  - Greška je kada se pozove POVUCI na praznom steku (greška tipa *underflow*)
  - Greška je kada se pozove GURNI na „punom“ steku gde nema dodatnog mesta jer je popunjen sav planiran prostor (*overflow*)

Stek primer

Primer par operacija sa stekom:

(početno stanje a))

GURNI(S, 17)

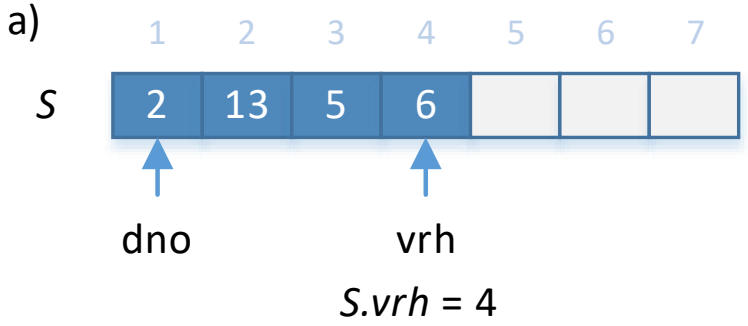
GURNI(S, 3)

(nastaje stanje b))

a = POVUCI(S)

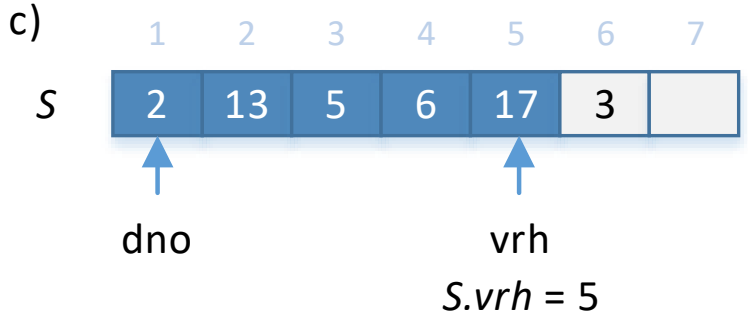
(nastaje stanje c))

a)



b)

c)



Stek operacije

- Sve operacije su brze.
- Svaka traje $O(1)$

- Nedostaje provera „prepunjenosti steka“

```
STEK-PRAZAN(S)
1  if S.vrh == 0
2    return True
3  else return False

GURNI(S,x)
1  S.vrh = S.vrh + 1
2  S[S.vrh]= x

POVUCI(S)
1  if STEK-PRAZAN(S)
2    error “underflow”
3  else S.vrh = S.vrh – 1
4    return S[S.vrh + 1]
```

Red

- Operacije:
  - dodavanje elementa – DODAJ – (***Enqueue***)
  - preuzimanje i uklanjanje elementa – PREUZMI – (***Deqeueu***)
  - Provera: da li ima elemenata?
- Podaci:
  - „glava“ (*head*) pokazuje na prvi elemenat reda.
    Preuzima se (*dequeue*) element sa početka – glave reda.
  - „rep“ (*tail*) pokazuje na poslednji elemenat reda.
    Dodavanje elementa ga smešta na kraj – rep reda.
- Implementacija (jedan način):
  - niz Q[1..*n*] je prostor za najviše *n*-1 elemenata reda
  - Q se koristi kao kružni bafer

Operacije sa redom

- U gornjim operacijama nedostaju provere
  - Greška je kada se pozove PREUZMI na praznom redu (greška tipa *underflow*)
  - Greška je kada se pozove DODAJ na punom redu (*overflow*)

```
DODAJ(Q,x)
1  Q[Q.rep] = x
2  if Q.rep == Q.length
3    Q.rep = 1
4  else Q.rep = Q.rep + 1
```

```
PREUZMI(Q,x)
1  x = Q[Q.glava]
2  if Q.glava == Q.length
3    Q.glava = 1
4  else Q.glava = Q.glava + 1
5  return x
```

Primer reda

- Primer par operacija sa redom:

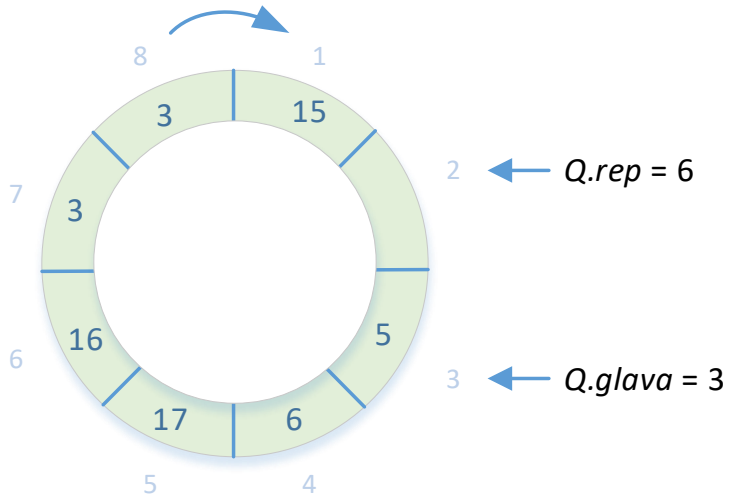
  DODAJ(Q,16)

  DODAJ(Q,3)

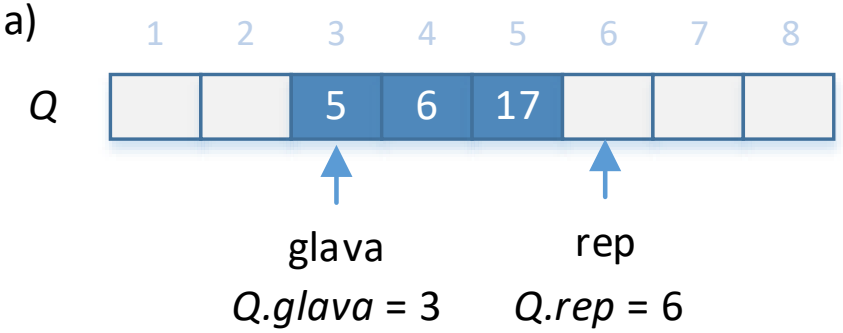
  DODAJ(Q,3)

  DODAJ(Q,15)

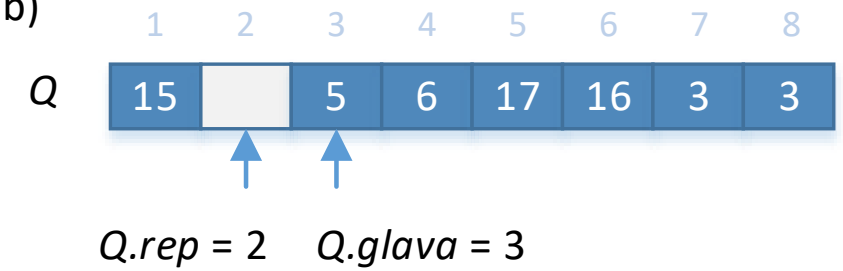
  a = PREUZMI(Q) // a = 5



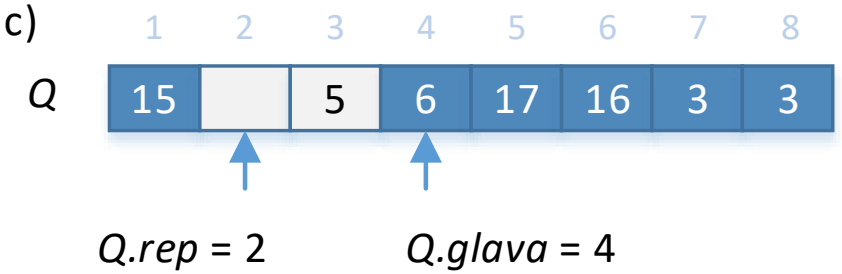
a)



b)

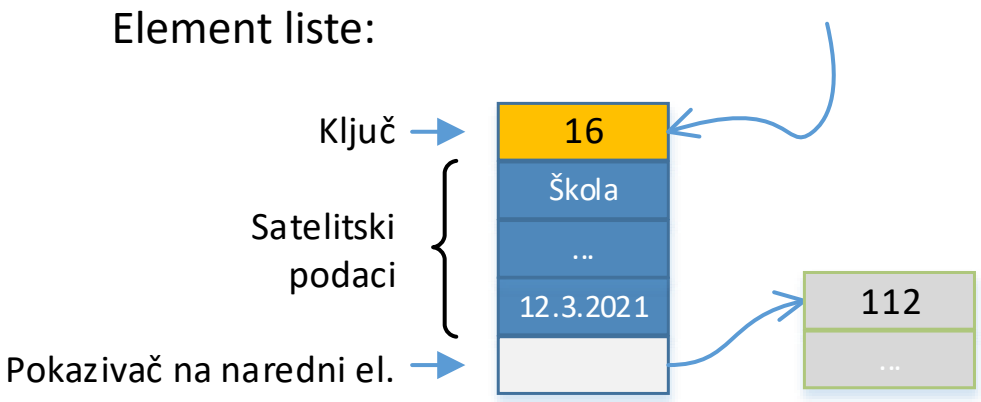


c)



Povezane liste

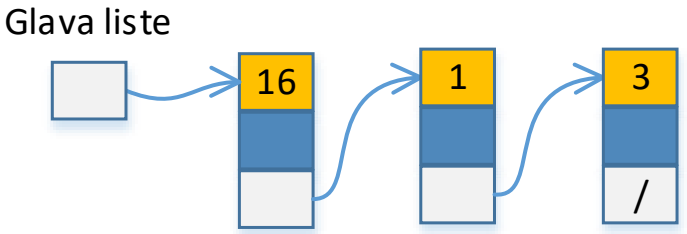
- Liste su strukture podataka gde su elementi uređeni u linearnom poretku.
- Za razliku od nizova gde je poredak uređen preko indeksa, kod lista imamo pokazivače na susedne elemente.
- Pogodna su za:
  - Česta dodavanja i brisanja elemenata
  - Česta posećivanja elemenata niza – iteriranje kroz listu.
- Nisu pogodne za pretrage



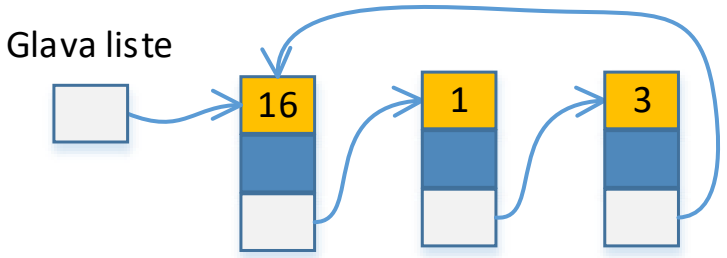
Tipovi lista

- Razlikuju se:
  - **Jednostruko spregnute liste** gde element pokazuje na naredni element. Omogućeno je kretanje samo u jednu stranu.
  - **Dvostruko spregnute liste** gde element pokazuje i na naredni i na prethodni elemenat. Omogućeno je kretanje na obe strane.
  - Cirkularne liste – elementi su u „prstenu“ (poslednji el. pokazuje na prvi, ...)
- Dodatno, lista može biti sortirana na osnovu ključa
  - Npr. prvi elemenat je sa najmanjim ključem, a poslednji sa najvećim

Primer jednostruko spregnute (uvezane) liste



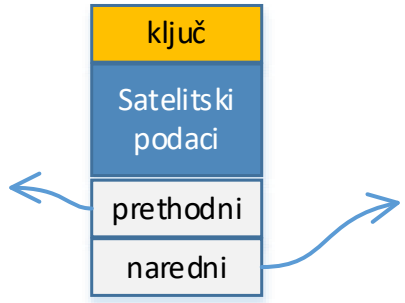
Primer cirkularne (kružne) liste

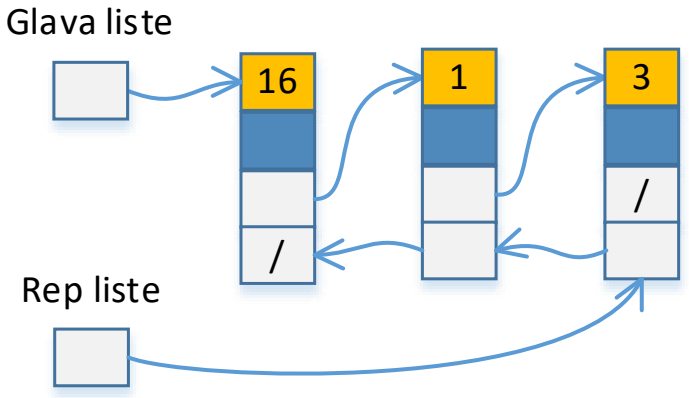


Dvostruko spregnuta lista

- Svaki elemenat sadrži
  - *naredni* – pokazivač na naredni elemenat (*next*) (== Nil kada je poslednji)
  - *prethodni* – pokazivač na prethodni elemenat (*prev*) (== Nil kada je prvi)
  - *ključ*, prisutan kod sortiranih lista (*key*)
  - drugi – satelitski podaci
- Pokazivač na prvi elemenat – L.*glava* (.*head*)
  - Ako je L. *glava* == NIL lista je prazna
- Pokazivač na poslednji elemenat – L.*rep* (.*tail*) – nije uvek prisutan

Element liste:





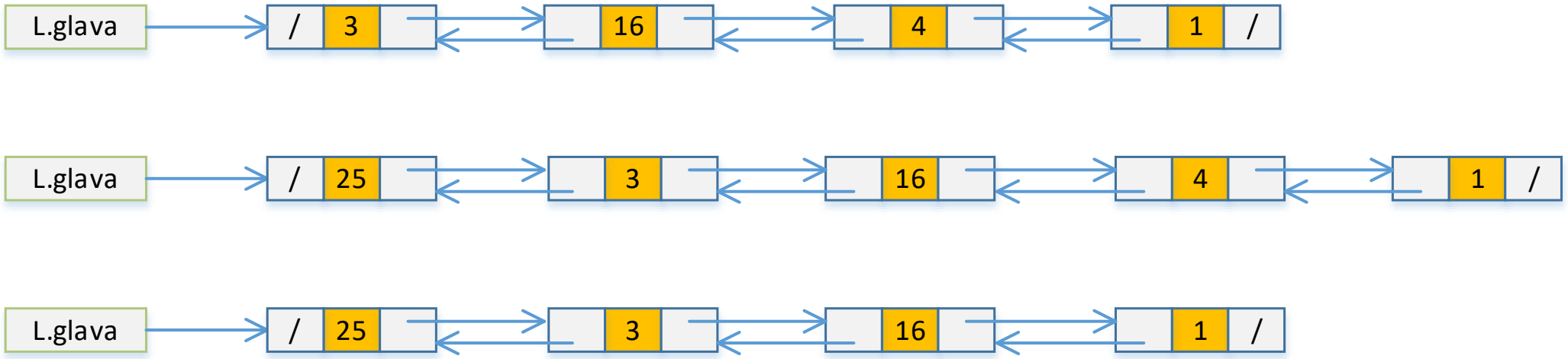
Napomena: Strelice treba da pokazuju na početak strukture elementa liste, ali je „nezgodno“ za crtanje.

Primer rada sa listom

a) Lista sadrži 4 elementa sa ključevima {3, 16, 4, 1}

b) Dodat je novi elemenat sa ključem 25
Primetiti: dodan je na početak liste

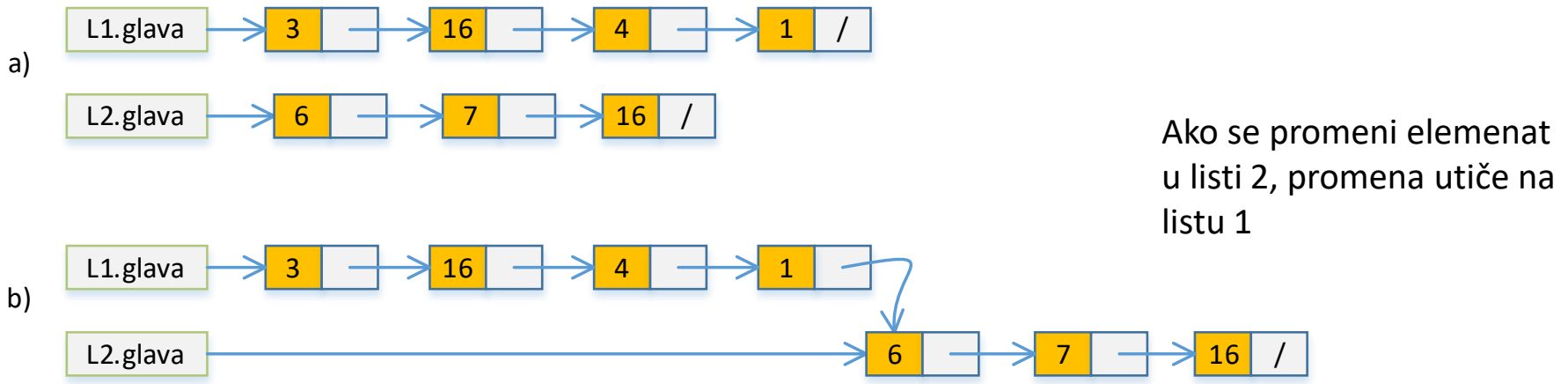
c) Pretraživanjem liste je dobijen pokazivač na elemenat sa ključem 4 i zatim je obrisan.



Jednostruko spregnuta lista

- Jednostavnija je od dvostruko spregnute liste
  - Elementi nemaju polje *prethodni* koje pokazuje na prethodni elemenat
  - Nema L.*rep*
- Omogućava kretanje (samo) od početka (L.*glava*) ka narednim elementima.
- Nema mogućnost kretanja ka prethodnom elementu.
  - Jedini način da se dođe do prethodnog elementa je ponovno kretanje od L.*glava*.

Primer nadovezivanja listi:



Ako se promeni elemenat u listi 2, promena utiče na listu 1

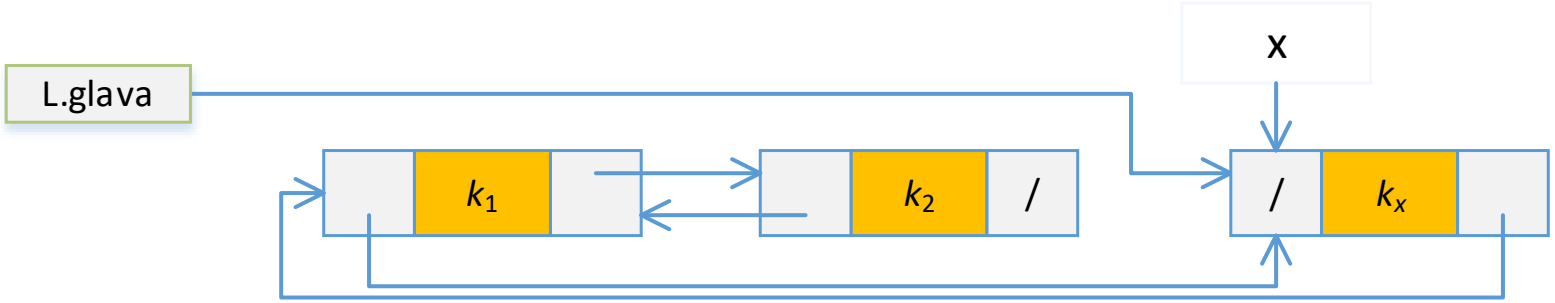
Pretraga u listi

```
PRETRAŽI-LISTU(L,k)
1  x = L.glava
2  while x ≠ Nil and x.ključ ≠ k
3     x = x.naredni
4  return x
```

- U najgorem slučaju traženi elemenat je na kraju liste ili ga nema.
- Složenost $\mathrm{O}(n)$, gde je n broj elemenata u listi
- Pretraga je primenljiva na jednostruko i dvostruko spregnute liste.

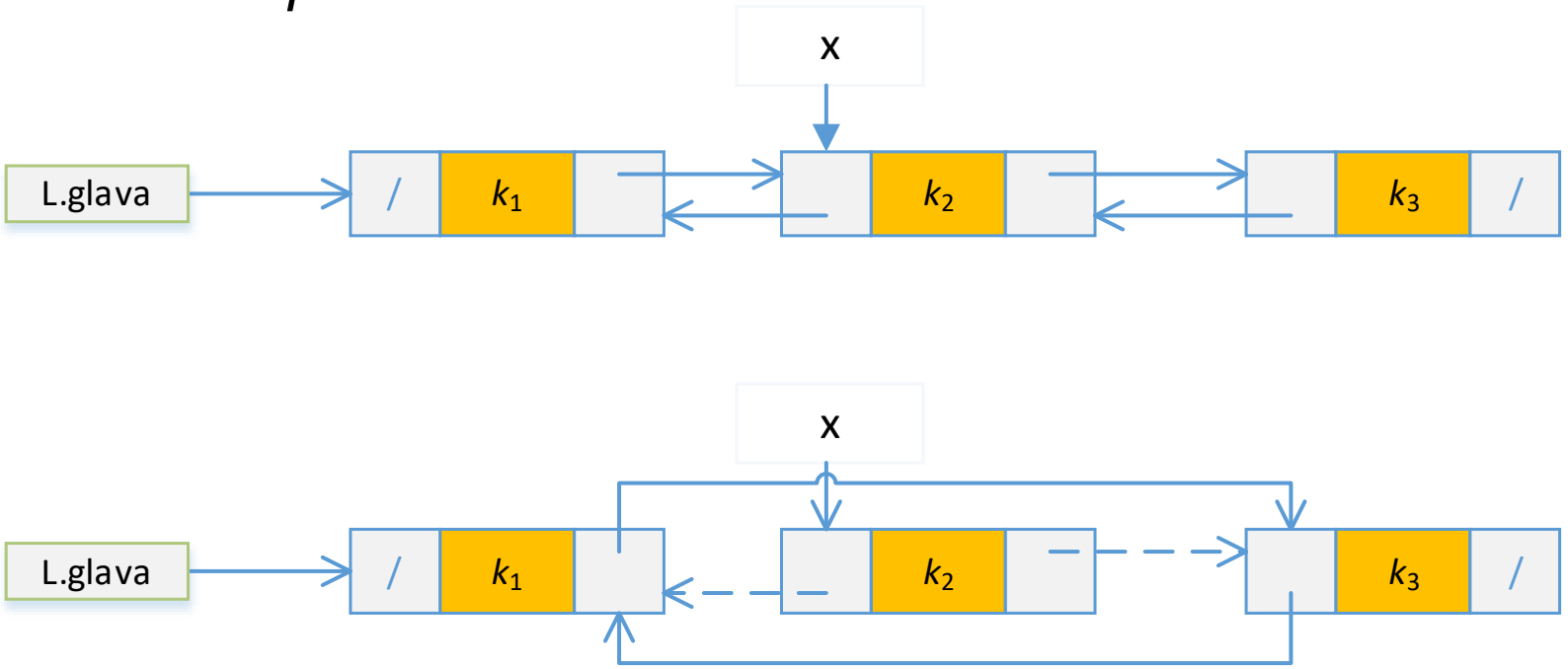
Dodavanje elementa u listu

```
DODAJ-U-LISTU(L,x)
1  x.naredni = L.glava
2  if L.glava ≠ Nil
3     L.glava.prethodni = x
4  L.glava = x
5  x.pretodni = Nil
```



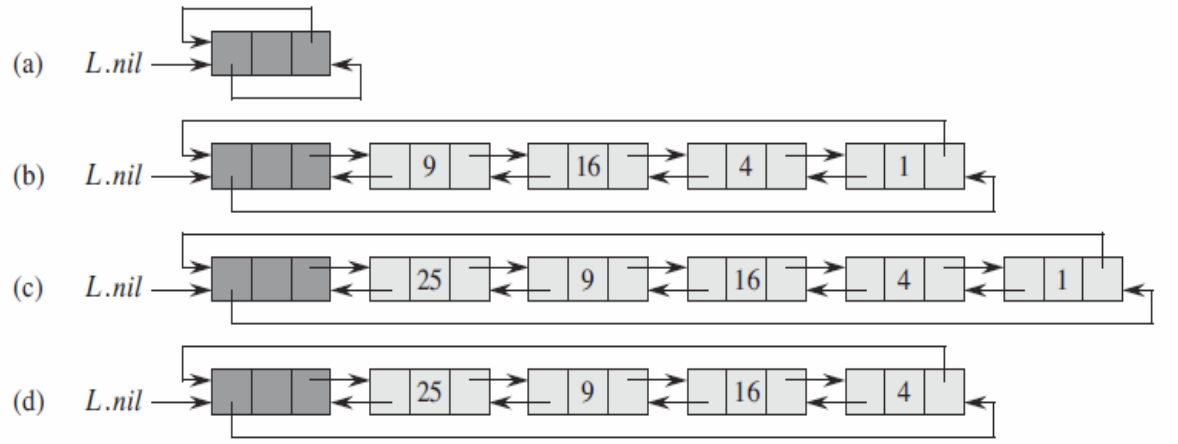
Brisanje elementa iz liste

```
BRIŠI-IZ-LISTE(L,x)
1  if x.prethodni ≠ Nil
2     x.prethodni.naredni = x.naredni
3  else L.glava = x.naredni
4  if x.naredni ≠ Nil
5     x.naredni.prethodni = x.prethodni
```



Upotreba „čuvara“

- Implementacije operacija se pojednostavljuju ako se ignorišu granični uslovi.
  - Podaci se organizuju u cirkularnu listu
  - Dodaje se čuvar (*sentinel*) kao fiktivan (prazan) objekat
  - Dodaje se pokazivač na njega L.nil
  - Poređenje sa NIL se svodi na poređenje sa L.nil




```
PRETRAŽI-LISTU(L,k)
1  x = L.nil.naredni
2  while x ≠ L.nil and x.ključ ≠ k
3     x = x.naredni
4  return x
```

```
BRIŠI-IZ-LISTE(L,x)
1  x.prethodni.naredni = x.naredni
2  x.naredni.prethodni = x.prethodni
```

```
DODAJ-U-LISTU(L,x)
1  x.naredni = L.nil.naredni
2  L.nil.naredni.prethodni = x
3  L.nil.naredni = x
4  x.prethodni = L.nil
```

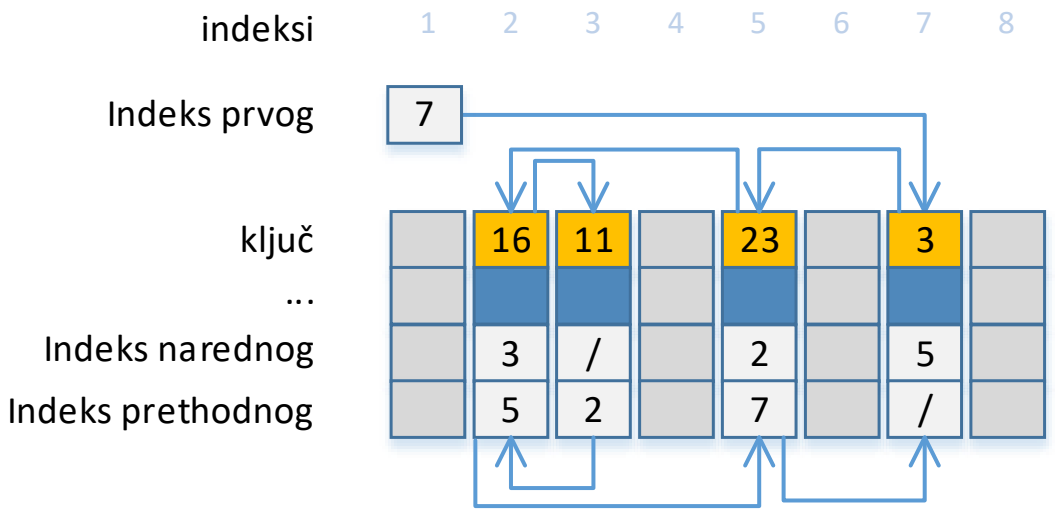
Implementacija pokazivača i objekata preko nizova

- Kako implementirati pokazivače u programskim jezicima koji ih ne podržavaju?
  Rešenje: Upotreba nizova i indeksiranje elemenata.

- Upotrebljava se nekoliko rešenja, preko:
  1. Niza struktura
  2. Nekoliko usklađenih nizova (gde svaki sadrži neki atribut elemenata)
  3. Jednog niza (i preračunavanja veličine elementa)

Implementacija liste pomoću nizova

- Npr. implementacija liste sadrži 4 niza:
  - vrednosti ključeva
  - satelitske podatke
  - indekse narednih elemenata
  - indekse prethodnih elemenata
- Nizovi su „usklađeni“ – na indeksu i u svim nizovima nalaze se polja istog elementa.
- Nil vrednost se može predstaviti sa 0 ili -1.

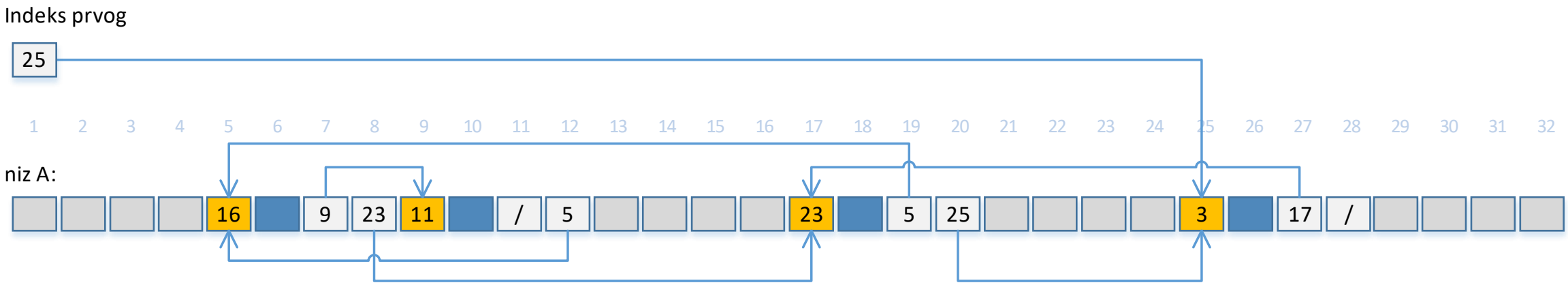


Napomena: prikazane strelice nisu pokazivači, nego su prikazane radi jednostavnijeg praćenja veza.

Slična je implementacija upotrebom niza struktura sa poljima: prethodni, kluč, naredni

Implementacija liste preko jednog niza

- Jedan niz sadrži sve elemente sa indeksima gde se nalaze prethodni i naredni elementi.
- Ograničenje: ključ i drugi korisni podaci elementa se moraju „uklopiti“ sa tipovima podataka upotrebljenim za indekse (tipično celobrojne).



Zauzimanje i oslobađanje elemenata liste

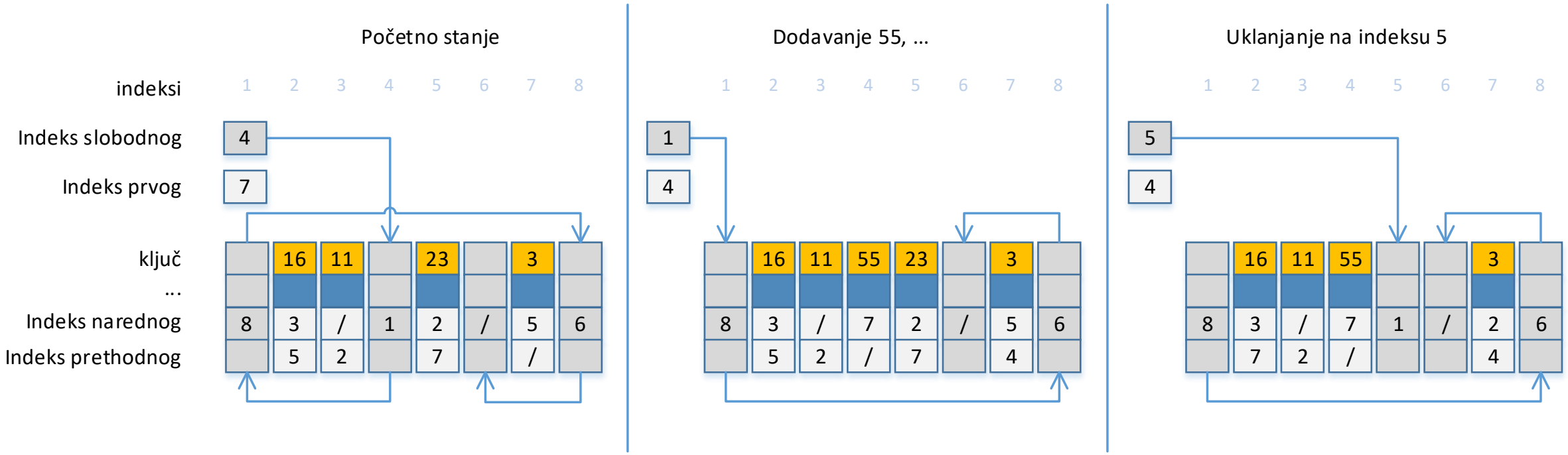
- Prethodne implementacije zahtevaju označavanje neupotrebljenih „slotova“ (potencijalnih mesta) za memorisanje elemenata.
- Pogodno je neupotrebljene „slotove“ povezati u dodatnu listu čiji je početak određen pokazivačem *slobodni*.
- U implementaciji preko više nizova njihova dužina je n, a tekući broj zauzetih elemenata je m ($m \leq n$).

Primeri koji ažuriraju samo slobodne „slotove“:

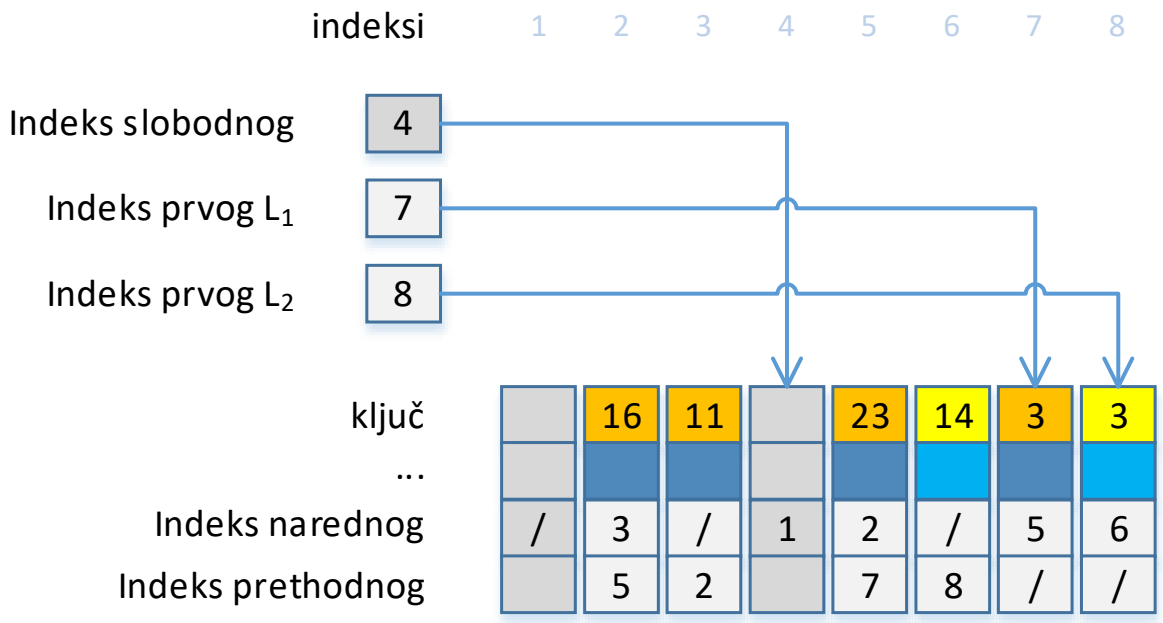
```
DODAJ-ELEMENT()
1  if slobodan == Nil
2     error „nema prostora“
3  else x = slobodan
4     slobodan = x.naredni
5     return x
```

```
UKLONI-ELEMENT(x)
1  x.naredni = slobodan
2  slobodan = x
```

Primer zauzimanja i oslobađanja elemenata

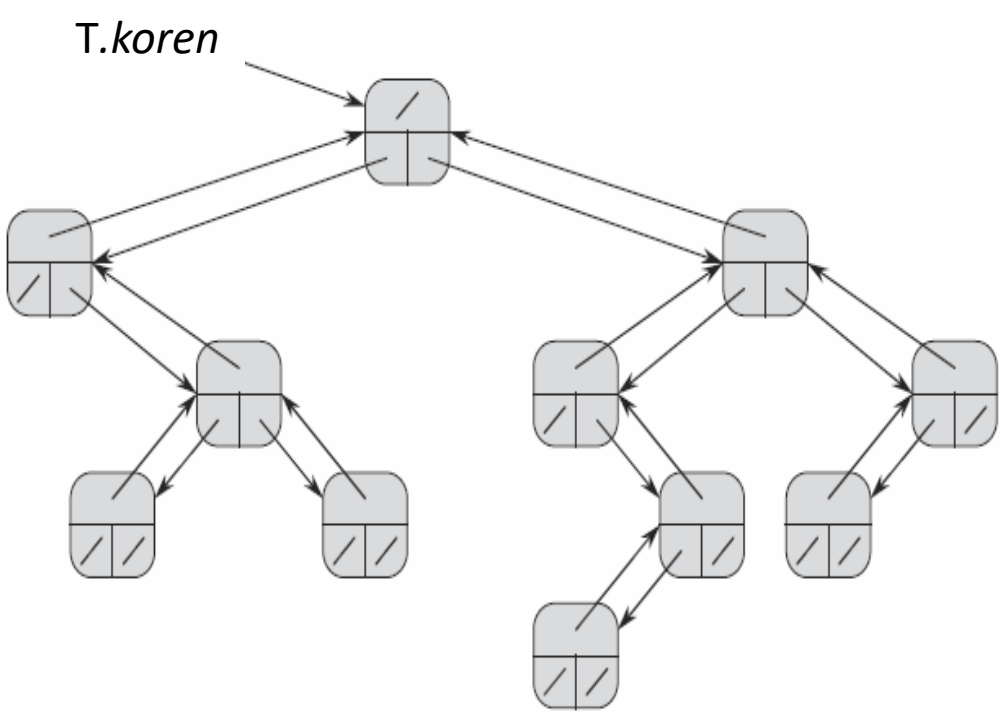


Primer sa tri liste u nizovima



Binarno stablo (uopšteno)

- Svaki elemenat stabla sadrži pokazivače na
  - Roditelja *r,*
  - Levo dete *levo,* i
  - Desno dete *desno.*
- Element u korenu stabla *x* ima polje *x.r* = NIL
- Element koji nema dece (ili ima samo jedno) polja *levo* i/ili *desno* postavlja na NIL.
- Struktura binarnog stabla sadrži pokazivač na koren stabla T.*koren*
  - Stablo bez elemenata ima T.*koren* = NIL



Stablo (uopšteno)

- Svaki roditelj može imati više dece što se može predstaviti na razne načine.

- Ukoliko uglavnom svi roditelji imaju jednak broj dece onda se elemenat stabla može proširiti poljima: $dete_1$, $dete_2$, ..., $dete_k$ (umesto polja *levo* i *desno*)

- Ukoliko broj dece varira od elementa do elementa – tada se mogu upotrebiti samo dva pokazivača:
  1. na levo (prvo) dete - *levo*
  2. na brata/sestru

- Pravi se kompromis memorijskog zauzeća i povezanosti elemenata.

Primer jedne implementacije stabla