Strukture Podataka i Algoritmi

Lekcija 4

leto 2019/2020

Prof. dr Branimir M. Trenkić

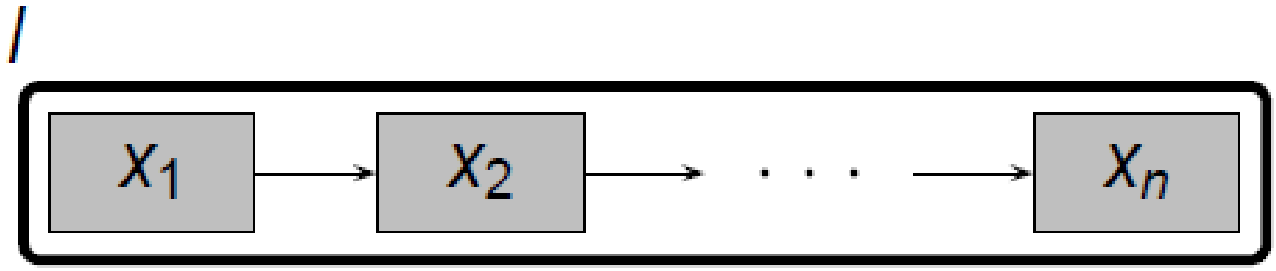
Liste

- ***Lista*** (povezana ili ***ulančana***) je ***linearna struktura*** podataka koja je ***slična nizu***
- Predstavlja ***sekvencu*** uređenih ***elemenata istog tipa***
- Sekvencijalna (***linearna***) ***uređenost*** se dobija:
- ***Kod niza*** - ***indeksiranjem elemenata*** niza
- ***Kod liste*** - ***pokazivačem*** u svakom elementu ***na sledeći element*** u listi

Liste

- Lista elemenata - ***svaki element*** (osim poslednjeg) ima ***tačno jednog sledbenika*** u nizu

- Lista $l = (x_1, x_2, ..., x_n)$ se može grafički predstaviti na sledeći način:



Liste

- ***Terminologija*** - slična onoj za nizove
- ***Broj elemenata*** liste se naziva ***veličina*** ili ***dužina*** liste
- Lista ***bez elemenata*** se naziva ***prazna lista***
- Za dva susedna elementa liste se kaže da je drugi element ***sledbenik*** prvom ili da je prvi ***prethodnik*** drugom
- ***Prvi element*** liste se naziva ***glava liste*** – nema prethodnika
- ***Poslednji element*** liste se naziva ***rep liste*** – nema sledbenika

Liste

- ***Glavna razlika listi*** u odnosu na ***nizove*** je u tome što ***dužina*** liste ***nije unapred određena***

Svaka lista je na početku prazna!

- Njena dužina povećava ili smanjuje ***dodavanjem*** novih ili ***uklanjanjem*** postojećih elemenata

Liste

- ***Prednosti liste*** nad nizom kao apstraktnog tip podataka:
- U primenama kod kojih ***ukupan broj*** nekih ***elemenata*** istog tpa – **nije poznat**
- U slučajevima kada ***operacije uklanjanja i dodavanja*** elemenata ne slede ***unapred poznat obrazac***

Liste

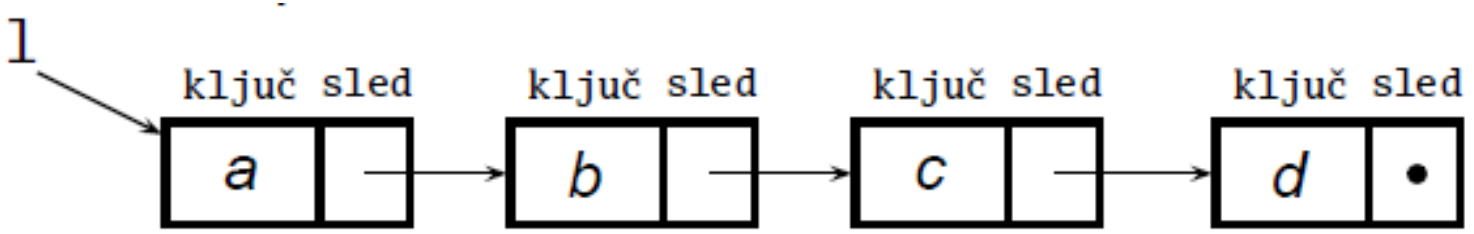
- ***Prednosti niza*** nad listom kao apstraktnog tip podataka:
- Kada je potreban ***efikasan pristup*** proizvoljnom elementu
  - Kod liste, pristup proizvoljnom elementu uvek mora ***početi od prvog*** .....
  - Na osnovu toga zaključujemo da operacija pristupa proizvoljnom elementu liste ***nije operacija sa konstantnim vremenom*** kao što je slučaj kod nizova

Liste

- ***Osnovne operacije*** nad listom su:
  - ***Konstruisanje*** prazne liste;
  - ***Dodavanje*** novog elementa u listu;
  - ***Uklanjanje*** datog elementa iz liste;
  - ***Pretraga*** liste

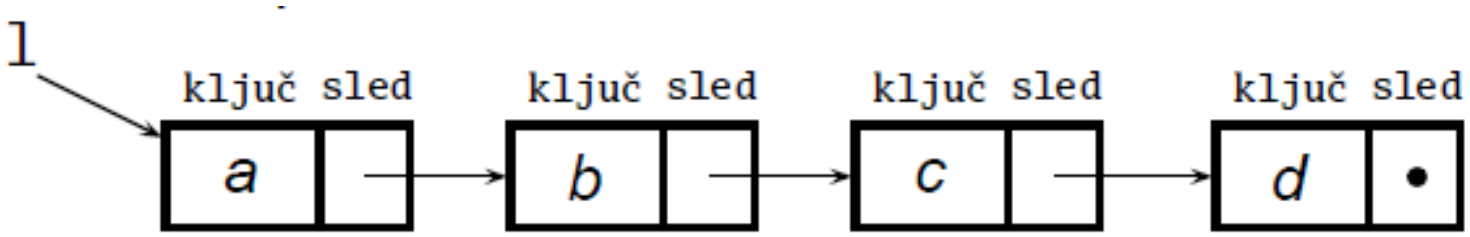
Implementacija liste pokazivačima

- Svaki ***element liste*** predstavlja se ***objektom*** (strukturom) ***sa dva polja***
- Ovaj objekat se obično naziva ***čvor liste***
- ***Prvo polje*** čvora (***ključ***)- ***sadrži*** relevantan ***podatak*** odgovarajućeg ***elementa liste***
- Tip podataka ***zavisi od*** konkretne ***primene*** liste
- Poistovećuje se sa ***pravim sadržajem*** odgovarajućeg ***elementa liste***



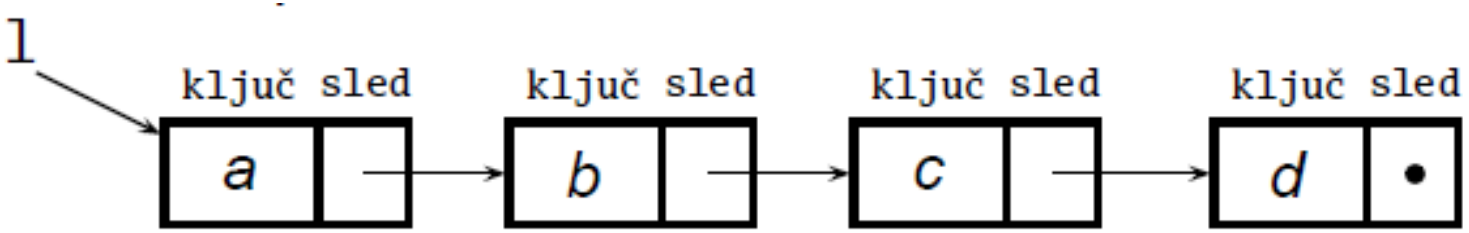
Implementacija liste pokazivačima

- ***Drugo polje*** čvora - sadrži ***pokazivač na sledeći*** čvor liste
- Pokazivačko polje (oznaka ***sled***) ukazuje na sledbenika elementa liste.
- ***Implementacija liste (a, b, c, d)****:*



Implementacija liste pokazivačima

- ***Dodatni element*** - ***pristupni pokazivač***
- Listi ***l*** se grupno pristupa preko posebnog ***spoljašnjeg pokazivača*** (oznaka ***l***)
  - Nije deo liste i
  - Ukazuje na prvi čvor liste



Implementacija liste pokazivačima

- ***Posebna vrednost*** pokazivača ***null***
- Pokazivač sa ovom vrednošću – ***ne ukazuje ni na šta***
- Na taj način je omogućeno ***uniformno tretiranje svih čvorova*** liste
- ***Poslednji čvor liste*** u pokazivačkom polju ***sadrži tu specijalnu vrednost*** pokazivača
- Ako pokazivač ***l*** ima ***vrednost null***, onda smatramo da je ***lista l prazna***

Osnovne operacije

Konstruisanje prazne liste

- Konstruisanje prazne liste se sastoji od ***inicijalizacije*** njegovog ***pristupnog pokazivača*** vrednošću ***null***:

```
// Ulaz: lista l

// Izlaz: prazna lista l

algorithm list-make(l)

        l = null;

        return l;
```

Osnovne operacije

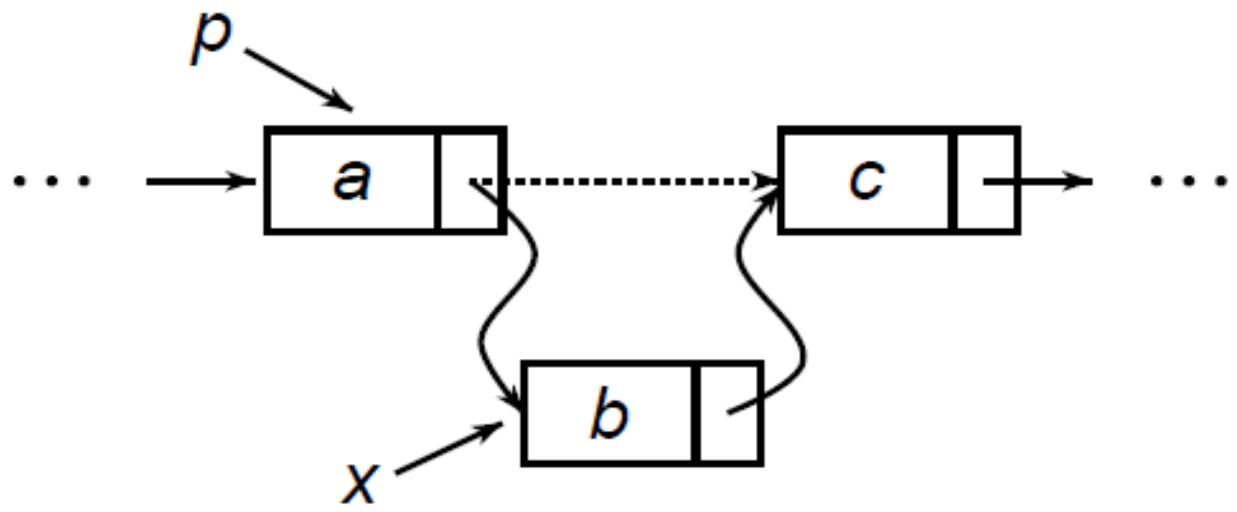
Dodavanje novog čvora u listu

- Mora se ***znati čvor*** liste ***iza koga*** se dodaje novi čvor
- ***Dat je*** (pokazivač na) ***novi čvor x*** i (pokazivač na) ***čvor p*** liste ***iza koga*** se dodaje novi čvor
- Ako je ***vrednost pokazivača p*** jednak ***null*** – to znači da se novi element (čvor) dodaje na početku liste (***kao glava liste***!)

Osnovne operacije

Dodavanje novog čvora u listu

- Veza između čvora ***p*** i njegovog sledbenika u listi mora se raskinuti i između njih umetnuti čvor x



Osnovne operacije

Dodavanje novog čvora u listu – ***Algoritam***

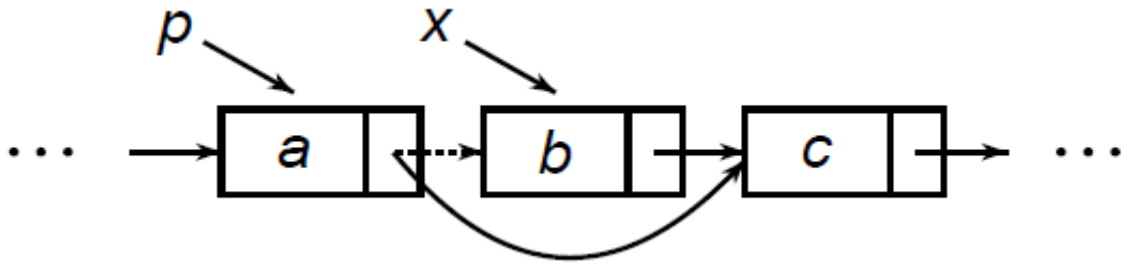
štamparska greška u knjizi! (strana 87)

```
// Ulaz:  novi čvor x, čvor p u listi l
// Izlaz: lista l sa čvorom x iza čvora p
algorithm list-insert(x, p, l)

   if (p == null) then  // novi čvor dodati na početak
      x.sled = l;
      l = x;
   else                 // novi čvor dodati iza p
      x.sled = p.sled;
      p.sled = x;

   return l;
```

Osnovne operacije

Uklanjanje čvora iz liste

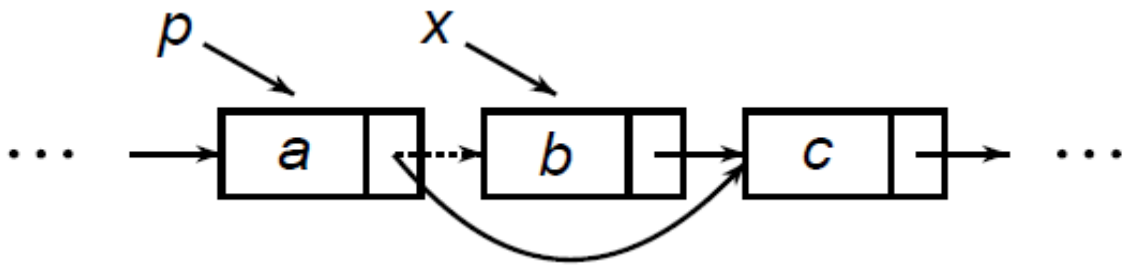
- Mora se ***znati čvor*** liste ***iza koga*** se uklanja postojeći čvor
- Ako je ***dat čvor p*** liste ***čijeg sledbenika treba ukloniti*** - ***veza*** između tog čvora i njegovog ***sledbenika x*** u listi ***se raskida***
- Uspostavlja se nova veza između čvora p i sledbenika starog sledbenika x



Osnovne operacije

Uklanjanje čvora iz liste

- Ako je ***vrednost pokazivača p*** jednak ***null*** – to znači da se uklanja element (čvor) sa početka liste (***glava liste***!)



Osnovne operacije

Uklanjanje čvora iz liste – ***Algoritam***

```
// Ulaz:  čvor p u listi l iza koga treba ukloniti čvor
// Izlaz: lista l bez uklonjenog čvora
algorithm list-delete(p, l)

   if (p == null) then  // ukloniti čvor na početku
      l = l.sled;
   else                 // ukloniti čvor iza p
      p.sled = p.sled.sled;

   return l;
```

Osnovne operacije

Pretraga liste

- Operacijom pretrage liste se određuje da li se u ***datoj listi l*** nalazi čvor sa ***datim ključem k***
- ***Ako je to slučaj***, algoritam za ovu operaciju u nastavku ***kao rezultat vraća*** (pokazivač na) ***prvi pronađeni čvor*** sa datim ključem
- ***U suprotnom slučaju***, ako čvor sa datim ključem nije pronađen u listi, ***kao rezultat se vraća pokazivač null***

Osnovne operacije

Pretraga liste – ***Algoritam***

```
// Ulaz:  lista l, ključ k
// Izlaz: čvor x u listi sa ključem k ili vrednost null
algorithm list-search(l, k)

   x = l;
   while ((x != null) && (x.ključ != k)) do
      x = x.sled;

   return x;
```

Osnovne operacije

Pretraga liste

- ***Operacije dodavanja i uklanjanja*** čvora – ***dva oblika***

1. Tačno ***znamo mesto u listi*** na kojem se dešava odgovarajuća promena
    - ***Iza datog čvora***
2. ***Ne znamo unapred mesto u listi*** na kojem će se desiti odgovarajuća promena
    - ***Iza čvora sa datim ključem***
    - ***Ne možemo*** neposredno ***iskoristiti date algoritme*** za dodavanje i uklanjenje čvora

Osnovne operacije

Pretraga liste

- ***Rešenje***:

a) Najpre iskoristimo algoritam ***list-search*** radi ***nalaženja čvora*** sa datim ključem

b) Prethodno opisanim algoritmima (***list-insert*** ili ***list-delete***) ***dodati ili ukloniti čvor*** iza pronađenog čvora

Dvostruko povezane liste

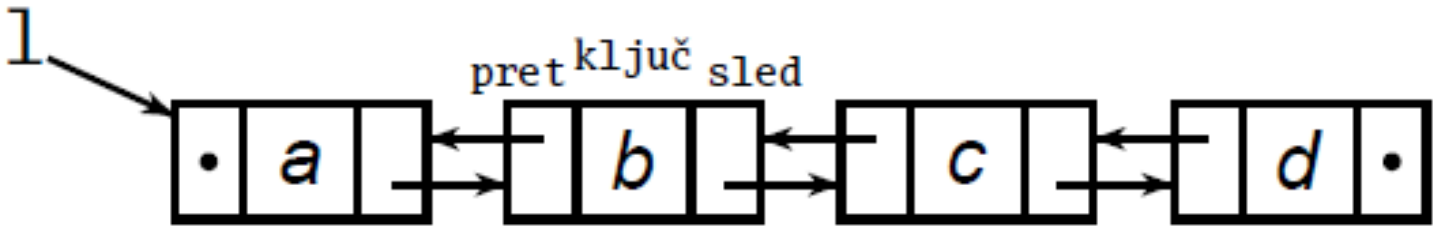
- Liste o kojima smo do sada govorili su ***jednostruko povezane liste***
- ***Svaki čvor*** ima ***jedno pokazivačko polje*** koje ukazuje na ***sledbenika čvora***
- Mogu se ***prelaziti samo u jednom smeru*** sleva na desno (algoritam ***list-search***)
- U nekim primenama je potrebno efikasno ***prelaziti listu u oba smera*** ili
- Za dati čvor je ***potrebno odrediti njegovog prethodnika***, a ne samo njegovog sledbenika

Dvostruko povezane liste

- U takvim slučajevima možemo koristiti ***dvostruko povezanu listu***
- ***Svaki čvor*** ima ***polje ključa*** i ***dva pokazivačka polja*** sa oznakama ***sled*** i ***pret***
  - polje ***sled*** kao i ranije ***ukazuje na sledbenika*** čvora
  - polje ***pret*** ***ukazuje na prethodnika*** čvora
- ***Prvi čvor*** dvostruko povezane liste u polju ***pret*** i ***poslednji čvor*** u polju ***sled*** imaju vrednost ***null***

Dvostruko povezane liste

- Dvostruko povezanoj listi se ***pristupa*** preko posebnog ***spoljašnjeg*** (***pristupnog***) pokazivača koji pokazuje na ***prvi čvor liste***, a koji ***nije deo liste***
- ***Implementacija liste (a, b, c, d)***:



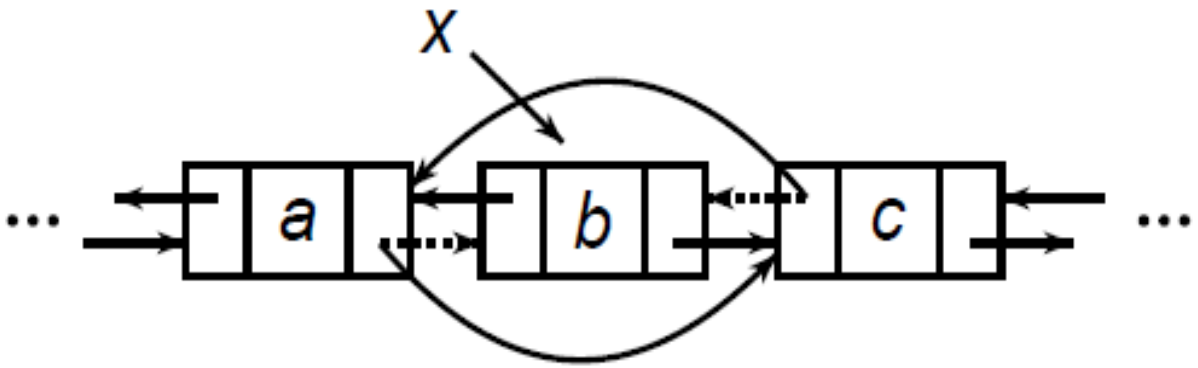
Dvostruko povezane liste

- ***Druga prednost*** dvostruko povezane liste je u tome što se na ***prirodniji način*** može ***locirati mesto*** promene sadržaja liste
- ***Dodavanje čvora*** možemo izvesti ***iza ili ispred datog čvora***
- Za ***uklanjanje čvora*** možemo ***kao parametar direktno koristiti željeni čvor*** (***a ne*** manje prirodno ***njegovog prethodnika)***

Dvostruko povezane liste

Uklanjanje čvora iz liste

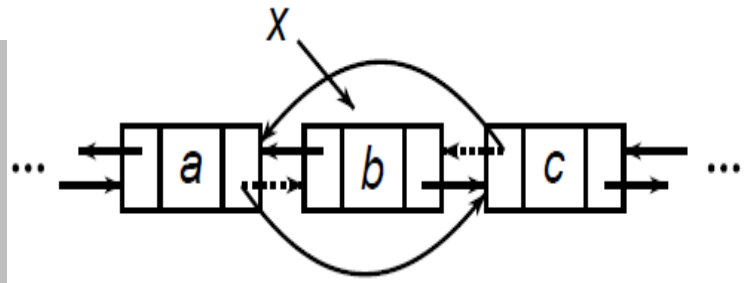
- Za uklanjanje čvora možemo ***kao parametar*** direktno ***koristiti željeni čvor***, a ne manje prirodno njegovog prethodnika



Dvostruko povezane liste

Uklanjanje čvora iz liste – ***Algoritam***

```
// Ulaz:  čvor x koji treba ukloniti iz l
// Izlaz: lista l bez uklonjenog čvora x
algorithm dlist-delete(x, l)

   if (x.pret != null) then   // x nije prvi čvor
      x.pret.sled = x.sled;
   else
      l = x.sled;
   if (x.sled != null) then   // x nije poslednji čvor
      x.sled.pret = x.pret;

   return l;
```



Implementacija liste nizovima

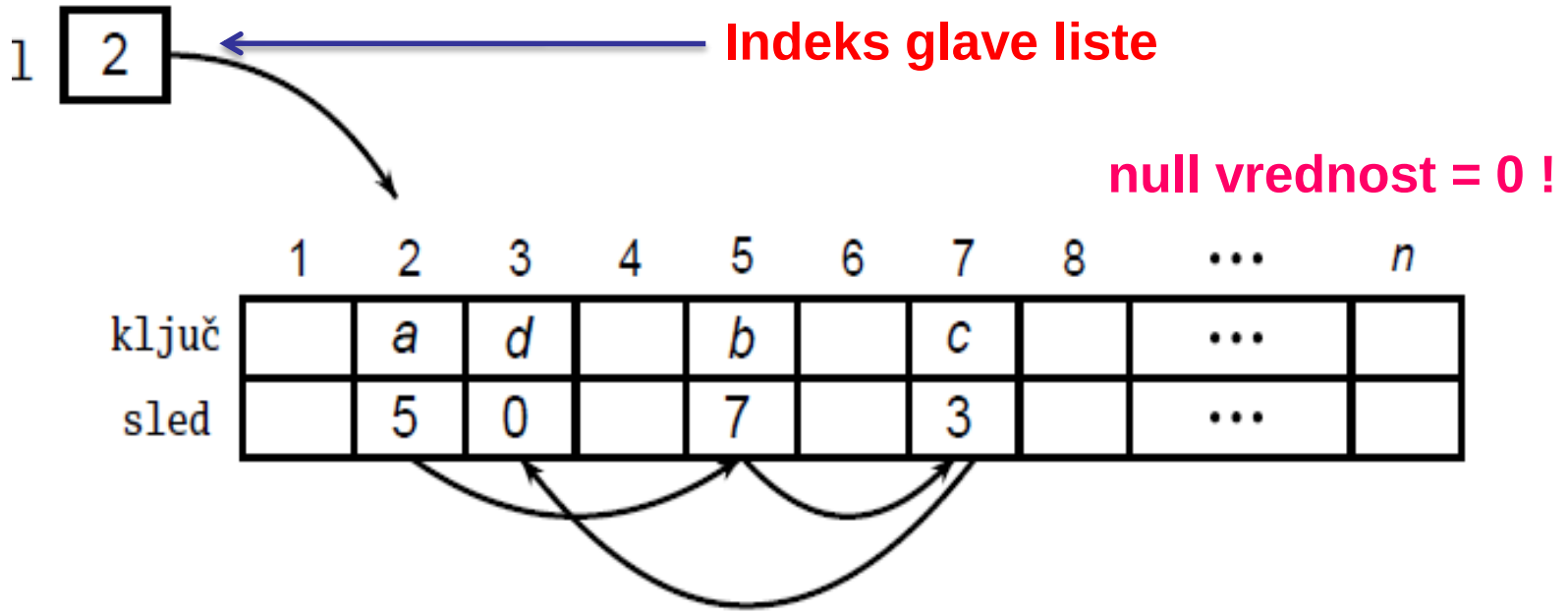
- Implementacija liste ***pomoću pokazivača*** je vrlo ***prirodna i jednostavna***
- Neki programski jezici ***ne omogućavaju*** direktan ***rad sa pokazivačima!!!***
- Zato ćemo kratko opisati:
- Kako se liste mogu predstaviti ***pomoću nizova***
- Kako se ***pokazivači*** mogu ***simulirati*** pomoću ***indeksa*** elemenata nizova

Implementacija liste nizovima

- ***Čvorovi liste*** sa istim poljima mogu se najjednostavnije predstaviti pomoću ***posebnog niza za svako polje***
- ***Jednostruko povezana lista*** se predstavlja pomoću ***dva niza***
  A. ***Niz*** (označen sa ***ključ***) sadrži ***vrednosti polja ključa čvorova liste***
  B. ***Niz*** (označen sa ***sled***) ***simulira pokazivačko polje*** na sledbenika

Implementacija liste nizovima

- ***Jednostruko povezana lista*** se predstavlja pomoću ***dva niza***
- ***Implementacija liste (a, b, c, d)****:*



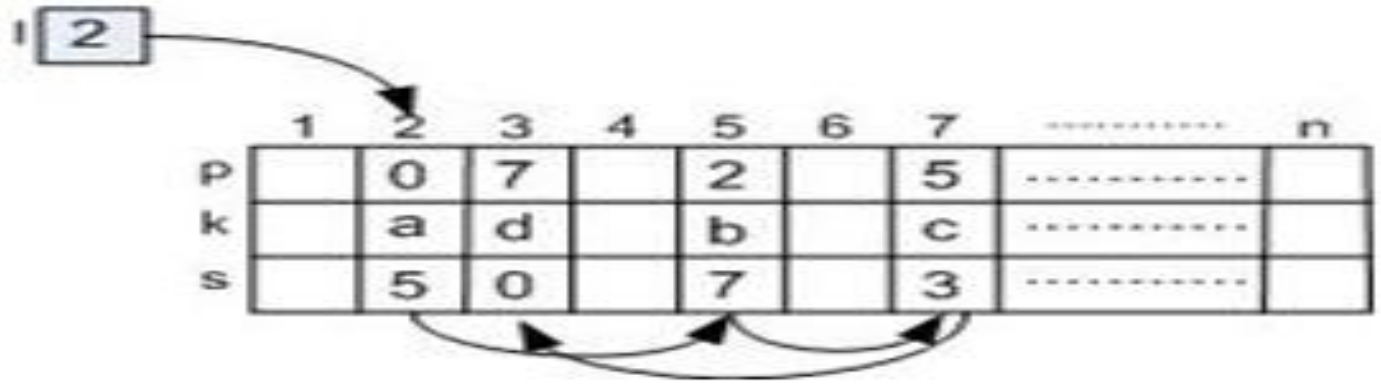
i-ti element ova dva niza – ***ključ(i) i sled(i)*** predstavljaju ***jedan čvor liste l***

Implementacija liste nizovima

- ***Dvostruko povezana lista*** se predstavlja pomoću ***tri niza***
- Jedan ***niz***, recimo ***k***, sadrži ***vrednosti polja ključa*** čvorova liste
- Preostala ***dva niza***, recimo ***s*** i ***p***, ***simuliraju pokazivačka polja*** na sledbenika i prethodnika

Implementacija liste nizovima

- ***Dvostruko povezana lista*** se predstavlja pomoću ***tri niza***
- Za dati ***indeks x*** ***k(x)***, ***s(x)*** i ***p(x)*** čine ***jedan*** element (***čvor***) ***liste***
- Celi brojevi koji ukazuju na pozicije u tim nizovima - ***kursori***



Implementacija liste nizovima

- ***Povezane liste*** možemo ***predstaviti*** i pomoću ***samo jednog niza***
- ***Slično*** kako se ***objekti smeštaju u memoriji*** računara - ***jedan objekat zauzima susedne lokacije*** u memoriji
- ***Pokazivač na*** neki ***objekat*** predstavlja ***adresu prve memorijske lokacije*** tog objekta
- ***Polja unutar tog objekta*** mogu se ***relativno indeksirati*** u odnosu na početni pokazivač

Implementacija liste nizovima

- Stanje ***jednog niza m*** koji se koristi za predstavljanje ***dvostruko povezane liste***

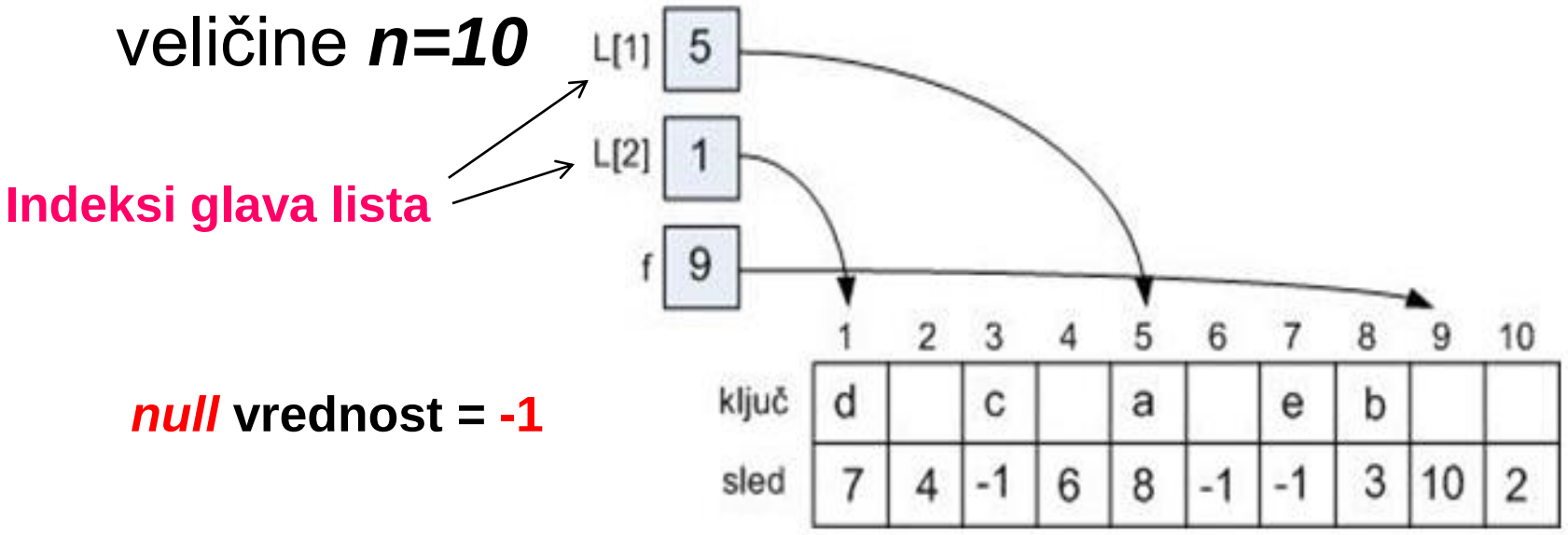
- ***Jedan čvor liste*** zauzima mesto neprekidnog podniza ***m[i], m[i+1], m [i+2]*** niza ***m***
- ***Pokazivač na ovaj čvor*** je određen ***početnim indeksom i*** podniza
- ***Relativne pozicije*** polja ***ključ***, ***sled*** i ***prev*** u čvoru su ***0, 1 i 2***

Implementacija liste nizovima

- Generalno,
- ***Prednost upotrebe jednog niza*** za predstavljanje struktura podataka je u tome što se ***na taj način mogu predstaviti*** i ***kolekcije heterogenih objekata*** različite veličine
- Sa druge strane, rad sa takvom reprezentacijom je ***komplikovaniji***

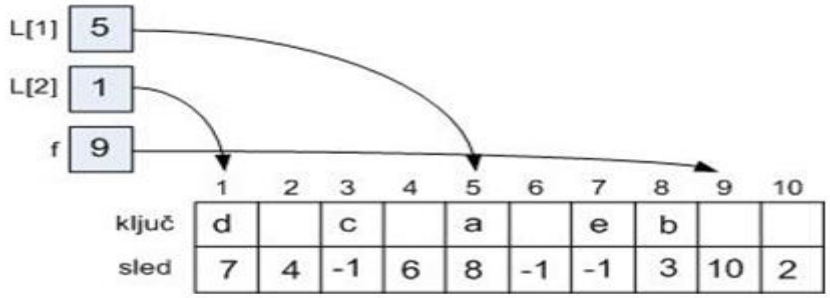
Implementacija liste nizovima

- ***Više jednostruko*** (dvostruko) povezanih ***listi*** možemo kombinovati ***u dva*** (odnosno tri) ***niza***
- Demonstriraćemo to na primeru dve jednostruko povezane liste: ***L[1] = (a, b, c)*** i ***L[2] = (d, e)***
- Neka ove dve liste dele ***dva niza*** – ***ključ*** i ***sled*** veličine ***n=10***



null **vrednost = -1**

Implementacija liste nizovima



- U nekom trenutku ***sve liste*** (2) ***koriste m*** (m ≤ n) ***elemenata*** niza ***ključ***
- Preostalih ***n – m*** elemenata su ***slobodni*** – od njih se formira ***slobodna lista*** (koristi samo elemente niza sled)
  - Globalna promenljiva ***f*** ukazuje na ***glavu slobodne liste***
  - Svi ***elementi*** niza ključ koji ***ne pripadaju*** ni jednoj listi su ***povezani u slobodnu*** listu

Implementacija liste nizovima

- ***Pretpostavke*** za algoritamsku realizaciju:
- U cilju ***pojednostavljenja*** zapisa algoritma:
- ***Nizovi*** (***ključ*** i ***sled***) čiji elementi reprezentuju čvorove list – smatramo ***globalno definisanim strukturama***
- Promenljive ***n***, ***f*** i ***L*** su, takođe, ***globalne promenljive***
- Algoritmi (operacije) koje se realizuju nad globalnim promenljivama – algoritam ***nema*** ulazno izlaznih parametara
  - Ovakve operacije se pozivaju kao ***podprogrami***

Implementacija liste nizovima

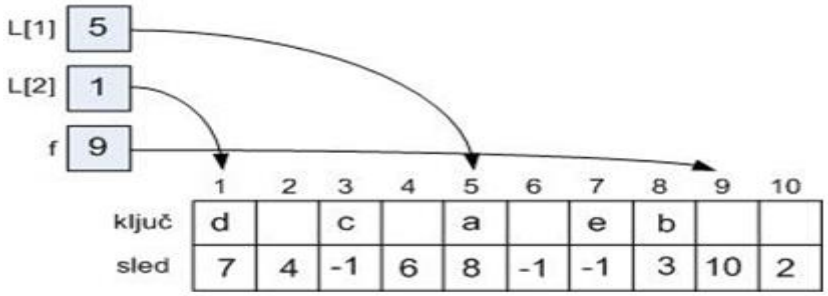
Inicijalizacija globalnih promenljivih

- ***Početno stanje*** nizova – formiranje ***slobodne liste*** kojoj inicijalno pripadaju svi elementi datih nizova
- Operacija (***init-free***) povezuje sve elemente niza u slobodnu listu (svaki element ukazuje na sledeći)
- ***Napomena***: U ***slobodnoj listi nije neophodno tretirati*** odgovarajuće vrednosti ***elemenata niza ključ*** – njih identifikuje činjenica da pripadaju slobodnoj listi

Implementacija liste nizovima

Inicijalizacija globalnih promenljivih - ***Algoritam***

```
// Ulaz: globalne promenljive

// Izlaz:

algorithm init-free()

    f = 1;                  // početak slobodne liste

    for i = 1 to n - 1 do

        sled[i] = i + 1;    // svaki element ukazuje na sledeći

    sled[n] = -1;           // kraj slobodne liste
```



Implementacija liste nizovima

Operacije nad slobodnom listom (dodavanje i uklanjanje elemenata)

- Operacija (***new-elem***) ***uklanja*** element iz slobodne liste ***sa glave***
- Ovaj element se dalje može ***dodeliti*** nekoj drugoj listi u okviru zajedničke reprezentacije

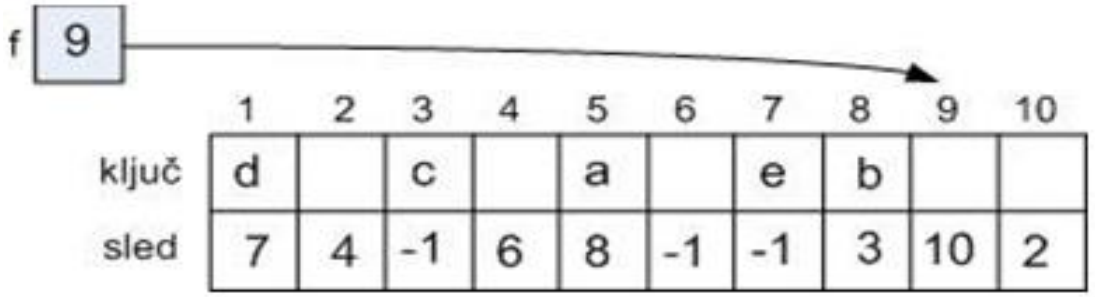
---

- ***Uklonjeni*** element liste se vraća u slobodnu listu
- Operacija (***free-elem***) uklonjeni element ***i*** liste ***dodati na početak*** (glava) ***slobodne liste***

Implementacija liste nizovima

Uklanjanje elementa iz slobodne liste - ***Algoritam***

```
// Ulaz: globalne promenljive

// Izlaz: i - indeks elementa izdvojenog iz slobodne liste sa glave

algorithm new-elem()

    if (f == -1)  then

        return -1;              // nema slobodnih elemenata

    else

        i = f;

        f = sled[i];

        return i;
```



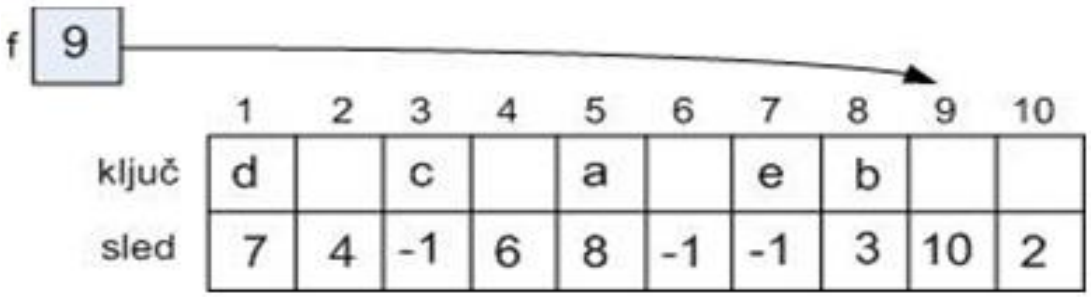
Implementacija liste nizovima

Dodavanje elementa u slobodnu listu - ***Algoritam***

```
// Ulaz: i indeks elementa koji vraćamo u slobodnu listu

// Izlaz: nema

algorithm free-elem(i)

    sled[i] = f;

    f = i;
```

Primer :
i = 5

sled[5] = 9
f = 5



Implementacija liste nizovima

Kreiranje nove (prazne) liste

- ***l*** - ***redni broj liste*** koju kreiramo
- ***L[l]*** – ***pokazivač*** (indeks) na (početak) kreirane liste (inicijalna vrednost pokazivača – ***null (-1)***)

Implementacija liste nizovima

Kreiranje nove (prazne) liste - ***Algoritam***

```
// Ulaz: l redni broj liste koju kreiramo

// Izlaz: nema

algorithm list-make(l)

    L[l] = -1;
```

Implementacija liste nizovima

- *Dodavanje novog elementa u listu*
- Mora se ***znati element*** liste ***iza koga*** se dodaje novi
- Data je ***vrednost ključa novog elementa x*** i ***pozicija elementa p*** liste ***iza koga*** se dodaje novi čvor
- Operacija (***list-insert***) dodaje novi element ***x*** u listu sa rednim brojem ***l*** iza elementa na poziciji ***p***
- Operacija pokriva ***oba slučaja***
  a) Dodavanje ***na početak liste*** (***p = -1***)
  b) Dodavanje ***iza datog elementa*** (***p != -1***)

Implementacija liste nizovima

- *Dodavanje novog elementa u listu* - ***Algoritam***

```
// Ulaz: lista l u koju se dodaje element x iza elementa na poziciji p
// Izlaz:
algorithm list-insert(l, x, p)
    i = new-elem();
    ključ[i] = x;
    if (p == -1)  then              // novi element dodati na početku liste
        sled[i] = sled[L[l]];
        L[l] = i;
    else                            // novi element dodati iza p
        sled[i] = sled[p];
        sled[p] = i;
```

Implementacija liste nizovima

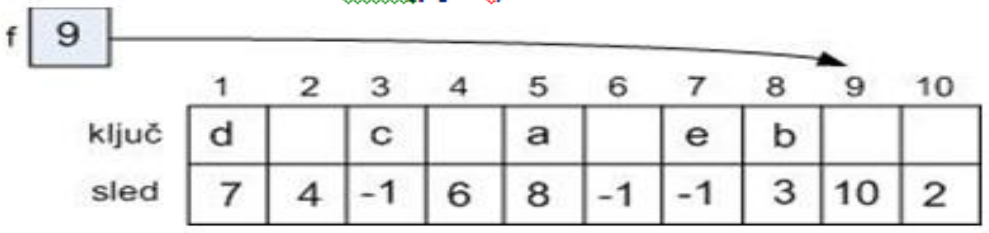
- *Dodavanje novog elementa u listu* - **Primer**

Primer: L[2] = (d, e)

x

- Dodaje se x iza d koje je na poziciji 1
- ***p = 1***
- i = 9 (nakon new-elem())
- ključ[9] = x
- sled[9] = sled[1]
- sled[1] = 9

```
// Ulaz: lista l u koju se dodaje element x iza elementa na poziciji p
// Izlaz:
algorithm list-insert(l, x, p)
    i = new-elem();
    ključ[i] = x;
    if (p == -1)  then          // novi element dodati na početku liste
        sled[i] = sled[L[l]];
        L[l] = i;
    else                        // novi element dodati iza p
        sled[i] = sled[p];
        sled[p] = i;
```



Implementacija liste nizovima

- *Uklanjanje elementa iz liste*
- Mora se ***znati pozicija*** elementa ***koji prethodi*** elementu koji uklanjamo iz liste
- Nakon uklanjanja iz liste – oslobađamo da ***smeštanjem na početak slobodne liste***
- Operacija (***list-delete***) uklanja element iz liste ***l*** i to sa pozicije iza onog koji je ukazan pozicijom ***p***

Implementacija liste nizovima

- *Uklanjanje elementa iz liste* - ***Algoritam***

```
// Ulaz: lista l iz koje se uklanja element iza elementa na poziciji p

// Izlaz:

algorithm list-delete(l, p)

    if (p != -1)  then              // uklanja se element iza elementa p

        i = sled[p];

        sled[p] = sled[i];

    else                            // element se uklanja kao prvi u listi (glava)

        i = L[l];

        L[l] = sled[i];

    free-elem(i);
```

Implementacija liste nizovima

- *Uklanjanje elementa iz liste* - ***Primer***

Primer: Ukloniti element na poziciji ***iza pozicije 5***

- Treba ukloniti ***b*** na poziciji ***8***
- i = sled[5] = 8
- sled[5] = sled[8] = 3
- free-elem(8)

```
// Ulaz: lista l iz koje se uklanja element iza elementa na poziciji p

// Izlaz:

algorithm list-delete(l, p)

    if (p != -1)  then          // uklanja se element iza elementa p

        i = sled[p];

        sled[p] = sled[i];

    else                        // element se uklanja kao prvi u listi (glava)

        i = L[l];

        L[l] = sled[i];

    free-elem(i);
```