Programski jezici i strukture podataka

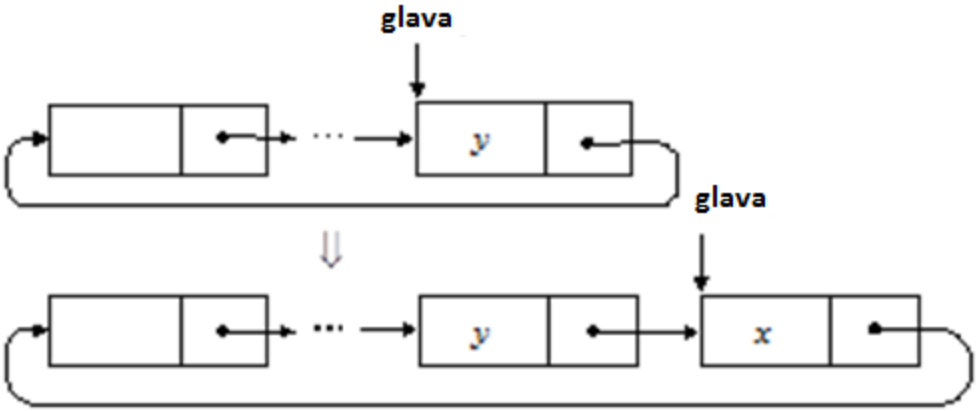
9

Kružne jednostruko spregnute liste

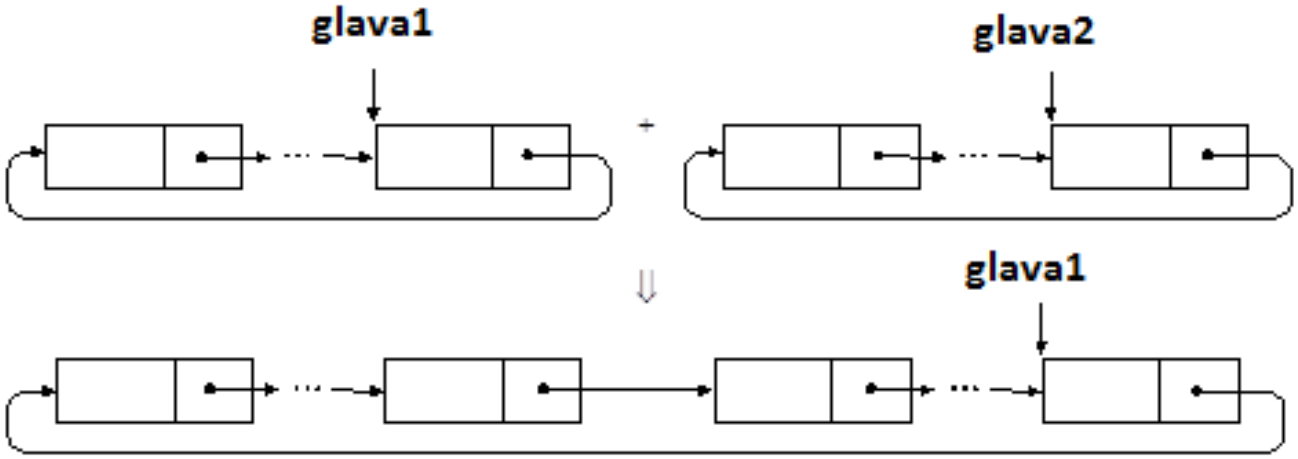
- Vrednost pokazivača na sledeći čvor u čvoru na kraju jednostruko spregnute liste ima vrednost NULL, pošto on ne pokazuje na sledeći čvor.
- U kružnoj listi svaki čvor pokazuje na sledeći u listi. Poslednji čvor pokazuje nazad na početak liste formirajuđi na taj način krug.
- U slučaju kružne liste jednostavnije se izvode operacije kao što su pretraživanje od nekog unutrašnjeg čvora ka bilo kom delu liste.

Kružne jednostruko spregnute liste

- Zbog osobine simetričnosti kružne liste termini “prvi” i “poslednji” čvor su upitni.
- Glava liste je korisnije da pokazuje na poslednji čvor liste, jer to operacije umetanja čvora na početak i kraj liste, kao i brisanja čvora sa početka i kraja, čini efikasnijim.



- Operacija spajanja dve liste CONCATENATE (glava1, glava2) ne zahteva prolazak do kraja prve liste pošto pokazivač glava1 ukazuje na poslednji čvor.
- Na kraju operacije pokazivač glava1 ukazuje na poslednji čvor objedinjene liste.



Kružna lista sa zaglavljem

- Ako se vrši pretraživanje kružne liste počevši od prvog čvora, pri neuspešnom pretraživanju dolazi do beskonačne petlje jer nema načina da se detektuje kraj liste (u slučaju jednostruke liste to je bio prazan pokazivač u poslednjem čvoru).
- Prelazak na čelo liste je moguće prepoznati jedino kad se sačuva spoljašnji pokazivač na prvi čvor, sa kojim se tekući pokazivač stalno upoređuje pri pomeranju kroz listu.
- Zbog toga se, obično, na početak liste ubacuje poseban čvor koji se naziva zaglavlje liste.
- Ovaj čvor ukazuje na prvi čvor liste, a razlikuje se od drugih po posebnoj vrednosti u polju info koja ne može biti validni sadržaj nekog čvora.

DVOSTRUKO SPREGNUTA LISTA

Dvostruko spregnuta lista

- Dvostruko spregnuta lista predstavlja **uopštenje jednostruko spregnutih lista**.
- Za svaku vezu (uređeni par) oblika (a,b) uvodi se veza u obrnutom smeru (b,a).
- Drugi naziv za dvostruko spregnute liste je **simetrične liste**.
- Dvostruko spregnute liste mogu biti u **cirkularnom obliku**.

Dvostruko spregnuta lista

- Dozvoljen je pristup svakom elementu.
- Može se dodavati na bilo kom mestu i ukloniti sa bilo kog mesta.
- Može se posmatrati kao unija dve jednostruko spregnute liste.
- Pojedinačne operacije sa dvostruko spregnutom listom su sporije od jednostruke zbog ažuriranja dva pokazivača.
- Razlika u osnovnim operacijama u odnosu na jednostruko spregnutu listu je mala.

Dvostruko spregnuta lista

- Dvostruko spregnuta lista organizuje podatke po jednoj relaciji gde pokazivački deo čvora liste ima dve informacije: ko je sledeći, a ko je prethodni čvor po datoj relaciji.
- Cilj je da se obezbedi brži pristup čvorovima liste a tako i brže operacije dodavanja i uklanjanja.

Definicija

- Dvostruko spregnuta lista se definiše kao uređeni par:

$$DP=(S(DP),r(DP))$$

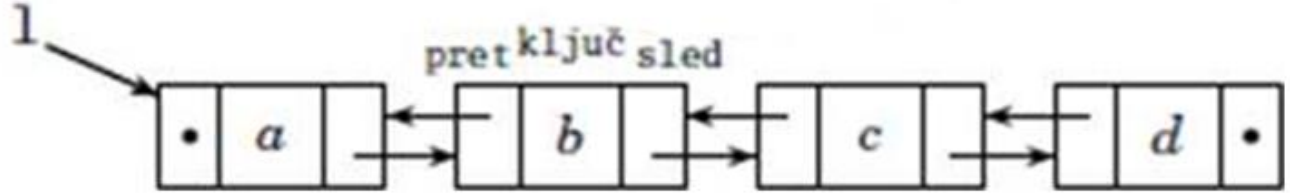
pri čemu se relacija r može razbiti na dve komponente pri čemu treba da je zadovoljeno:

$$r=r1 \cup r2 \qquad r1 \cap r2=\emptyset$$

i da strukture (S(DP),r1(DP)) i (S(DP),r2(DP)) budu jednostruko spregnute liste.

Dvostruko spregnuta lista

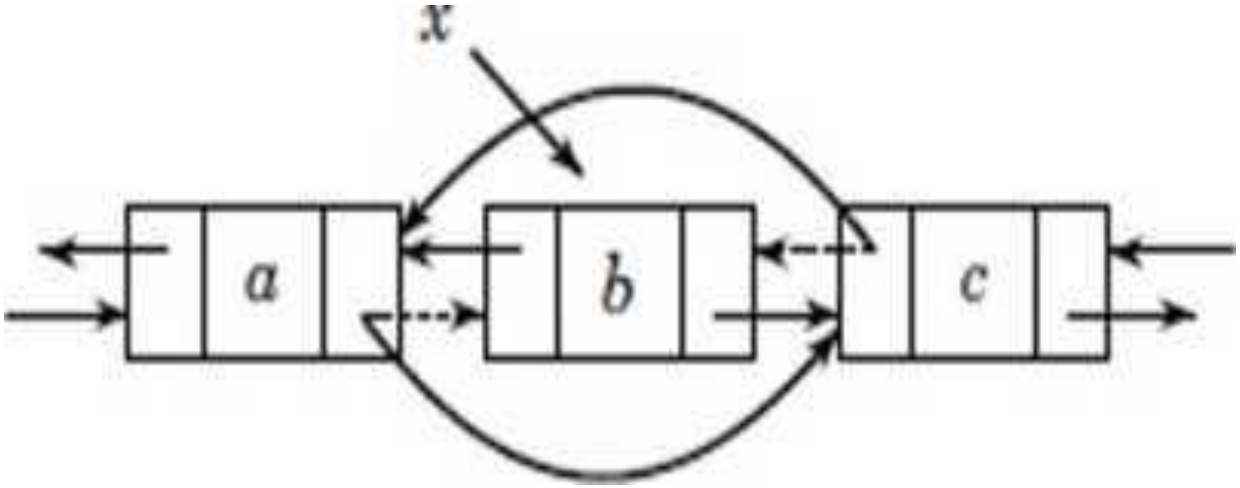
- Svaki čvor ima polje **ključ** i **dva pokazivačka polja** sa oznakama sled i pret.
- Polje **sled** ukazuje na sledbenika čvora.
- Polje **pret** ukazuje na prethodnika čvora.
- Prvi čvor dvostruko povezane liste u polju pret i poslednji čvor u polju sled imaju vrednost **null**.
- Kao kod jednostruko povezane liste, dvostruko povezanoj listi se pristupa preko spoljašnjeg pokazivača **glava** koji pokazuje na prvi čvor liste.



Dvostruko spregnuta lista

- Jedna od prednosti dvostruko povezane liste je u tome što dodavanje čvora možemo izvoditi iza ili ispred datog čvora.
- Za uklanjanje čvora možemo kao parametar direktno koristiti željeni čvor, a ne njegovog prethodnika kako je to kod jednostruko spregnute liste.

Uklanjanje čvora iz dvostruko spregnute liste



Element liste i lista (strukture)

```
typedef struct elem {
    int broj;
    struct elem *sled, *pret;
} Elem;

typedef struct {
    Elem *prvi, *posl
} Lista;
```

Prazna lista

```
Lista lst={NULL,NULL};
```

Obilazak liste u napred

```
for (tek=lst.prvi;tek;tek=tek->sled);
```

Obilazak liste u nazad

```
for (tek=lst.posl;tek;tek=tek->pret);
```

Ispisivanje liste u napred

```
void pisi_unapred (Lista lst)
{
      Elem *tek;

      for (tek=lst.prvi;tek;tek=tek->sled)
            printf("%d\n",tek->broj);
}
```

Ispisivanje liste u nazad

```
void pisi_unazad (Lista lst)
{
      Elem *tek;

      for (tek=lst.posl;tek;tek=tek->pret)
            printf("%d\n",tek->broj);
}
```

Nalaženje prve pojave broja

```
Elem* nadji_prvi (Lista lst, int b)
{
   Elem *tek;

   for (tek=lst.prvi;tek && tek->broj!=b;tek=tek->sled);

   return tek;
}
```

Nalaženje poslednje pojave broja

```
Elem* nadji_posl (Lista lst, int b)
{
   Elem *tek;

   for (tek=lst.posl;tek && tek->broj!=b;tek=tek->pret);

   return tek;
}
```

Dodavanje novog elementa na početak liste

```
Lista na_pocetak (Lista lst, int b)
{
  Elem *novi=malloc(sizeof(Elem));
  novi->broj=b;
  novi->sled=lst.prvi;
  novi->pret=NULL;
  if (!lst.posl) lst.posl=novi;
  else lst.prvi->pret=novi;
  lst.prvi=novi;
  return lst;
}
```

Dodavanje novog elementa na kraj liste

```
Lista na_kraj (Lista lst, int b)
{
  Elem *novi=malloc(sizeof(Elem));
  novi->broj=b;
  novi->pret=lst.posl;
  novi->sled=NULL;
  if (!lst.prvi) lst.prvi=novi;
  else lst.posl->sled=novi;
  lst.posl=novi;
  return lst;
}
```

Brisanje svih elemenata liste

```
void brisi (Lista *plst)
{
     Elem *tek=plst->prvi, *stari;
     while (tek)
     {
       stari=tek;
       tek=tek->sled;
       free(stari);
     }
     plst->prvi=plst->posl=NULL;
}
```

MULTI LISTA

Multi lista

- Multi lista organizuje podatke po više relacija u više dimenzija, gde broj dimenzija definiše broj klasa podataka uključenih u listi.
- Broj pokazivačkih skalara identifikatora liste, glava, mora odgovarati broju relacija pri čemu jedan pokazivački skalar pokazuje na prvi čvor liste po jednoj relaciji.
- Neki autori ovu klasu multilista nazivaju i liste u listi.

- Za razliku od jednostruko i dvostruko spregnutih lista, višestruke liste ili multiliste višestruko povezuju elemente, ali u skladu sa većim brojem kriterijuma.
- Za primer podataka o studentima, elementi bi mogli biti povezani tako da se, uz osnovnu vezu
  - prema **rastućem** broju indeksa, uvede i veza između elemenata koji odgovaraju
  - studentima **prve** godine, veza koja spaja
  - studente **druge**,
  - zatim **treće**, itd. godine, nakon toga veze koje spajaju po
  - **smerovima**,
  - **polu** i tome slično.
- Svaki od ovih kriterijuma definiše po jednu parcijalno jednostruko spregnutu listu (ređe dvostruku), a njihova unija sačinjava multilistu.

Višestruka lista

- Naglašena je terminološka razlika između **višestruka** u odnosu na **višestruko spregnuta** jer se ovde radi o vezama koje su po interpretaciji različite.
- Interesantno je primetiti da **ne moraju** svi elementi da učestvuju **u svim relacijama**.

Definicija

- Model višestruke listu je uređena entorka sledećeg oblika:

$$VP=(S(VP), r1,..., rn)\ n>2$$

- pri čemu su (Si,ri) takvi da je Si podskup od S(VP), jedostruko spregnute liste za i=1...n.
- Svaka od relacija ri (i=1...n) predstavlja model jednog tipa povezivanja po nekom od kriterijuma.

Višestruka lista

- Osnovne **operacije**, **programska** i **fizička struktura** višestruke liste grade se na istim načelima kao i kod jednostruko spregnute liste i pojavljuju se kao njihova uopštenja.
- Treba još uočiti da svaki element mora imati prostor za sve pokazivače(svake pojedinačne relacije) bez obzira da li je član pojedinačne relacije ili ne.

Sortiranje liste (*merge-sort*)

- Metod sortiranja neuređene (jednostruko povezane) liste brojeva, koji se popularno naziva *sortiranje objedinjavanjem* (engl. *merge-sort),* zasniva se na tri glavna koraka.
  - U prvom koraku se najpre deli data lista **l** od **n** brojeva u dve manje liste $\mathbf{l_1}$i $\mathbf{l_2}$.
  - U drugom koraku se zatim rekurzivno sortiraju ove dve polovične liste posebno.
  - U trećem koraku se dve sortirane polovične liste dobijene u drugom koraku objedinjuju u završnu sortiranu celu listu.

Da bismo datu listu **l** podelili u dve polovične liste l_1 i l_2 u prvom koraku, redom uklanjamo glavu date liste i dodajemo je naizmenično na početak prve ili druge liste.

```
// Izaz: lista l
// Izlaz: liste l1 i l2 formirane od naizmeničnih elemenata
// liste l

algorithm list-split(l, l1, l2)
        l1 = null; l2 = null;
        pd = true;// indikator dodavanja prvoj ili drugoj listi

        while (l != null) do
                x = l; // uklanjanje glave liste l
                l = list-delete(null,l);
                if (pd) then//dodavanje glave liste l na poč. od l1
                        l1=list-insert(x,null,l1);
                else   // dodavanje glave liste l na poč. Od  l2
                        l2 = list-insert(x,null,l2);
                pd = !pd;
        return l1, l2;
```

- Objediniti dve sortirane liste u trećem koraku sortiranja objedinjavanjem znači proizvesti jedinstvenu sortiranu listu koja sadrži sve elemente datih listi (i nijedan drugi element).
- Na primer, ako su date dve sortirane liste (1,2,3,7,8) i (1,2,6,9).
- Jedinstvena sortirana lista dobijena njihovim objedinjavanjem je (1, 1, 2, 2, 3, 6, 7, 8, 9).
- Kod objedinjavanja dve liste se podrazumeva da su one već sortirane.

- Postupak za objedinjavanje dve sortirane liste jeste da se obe liste paralelno ispituju redom od početka do kraja.
- Pri tome se u svakom koraku upoređuju glave dve liste, bira se manji element od njih (ako su oba elementa jednaka, uzima se bilo koji) za naredni element objedinjene liste.
- Na kraju, uklanja se izabrani element iz njegove liste.

```
// Ulaz: sortirane liste l1 i l2
// Izlaz: sortirana lista l formirana objedinjavanjem l1 i l2
Algorithm list-merge(l1,l2,l)
    l = null; // glava liste l
    r = null; // rep liste l
    while ((l1 != null) && (l2 != null)) do
        if (l1.ključ <= l2.ključ) then
                x = l1;
                l1= list-delete(null,l1);
        else
                x = l2;
                l2= list-delete(null,l2);
        l= list-insert(x,r,l);
        r = x;
    while (l1 != null) do
        x = l1;
        l1= list-delete(null,l1);
        l = list-insert(x,r,l);
        r = x;
    while (l2 != null) do
        x = l2;
        l2= list-delete(null,22);
        l= list-insert(x,r,l);
        r = x;
    return l1, 12;
```

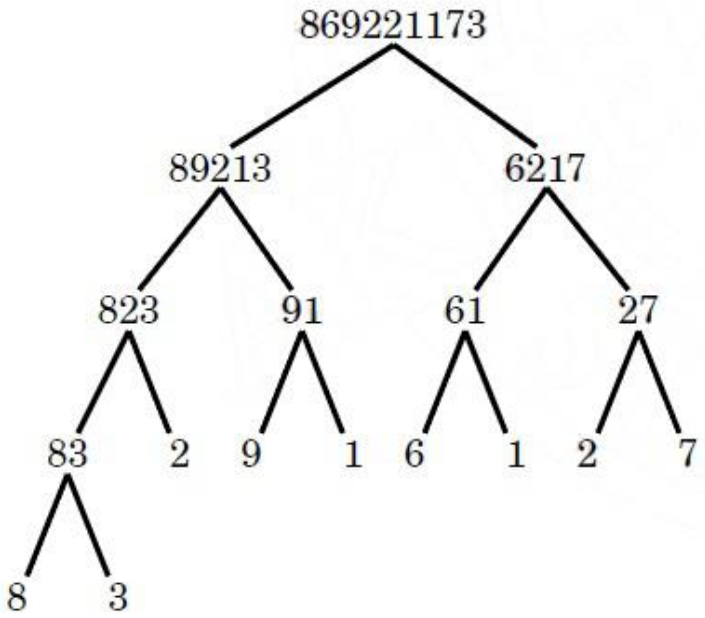
Upotrebom prethodna dva algoritma u prvom i trećem koraku postupka za sortiranje i dva rekurzivna poziva za sortiranje manjih lista brojeva u drugom koraku, kompletan algoritam za sortiranje je:

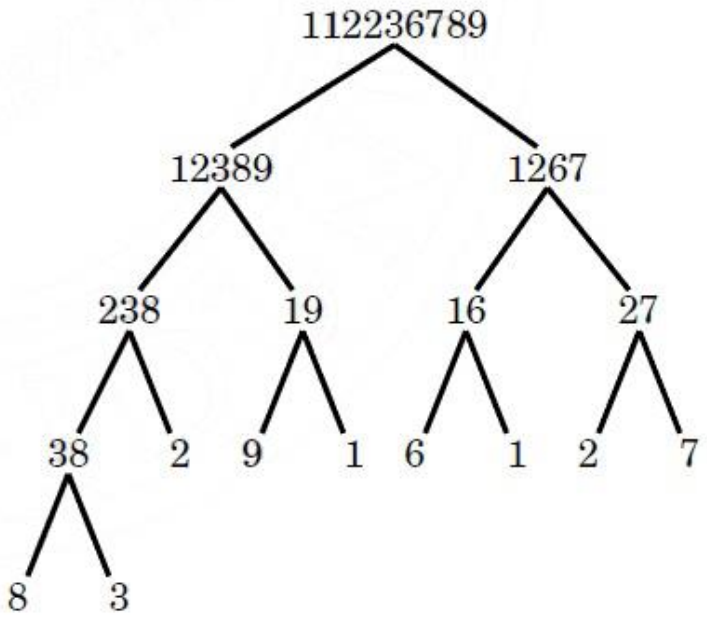
```
// Ulaz:   neuređena lista l
// Izlaz: sortirana lista l
algorithm merge-sort(l)

      if (l.sled!=null)then // lista ima više od
                            //jednog elementa
            l1,l2=list-split(l,l1,l2);
            l1 = merge-sort(l1);
            l2 = merge-sort(l2);
            l = list-merge(l1,l2,l);
      return l;
```

- Merge-sort na primeru liste jednocifrenih brojeva 8,6,9,2,2,1,1,7,3.
- Tako, ulazna lista je l = 869221173.
- Ova lista se najpre algoritmom list-split deli u dve liste **l1** i **l2**.
- Ove liste se sastoje od svakog drugog elementa polazne liste, odnosno prva lista sadrži neparne, a druga parne elemente polazne liste.
- l1= 89213 i l2= 6217.
- Zatim se ove polovične liste rekurzivno sortiraju istim algoritmom.
- Kao rezultat dobijaju seliste l1= 12389 i l2= 1267.
- Na kraju, algoritam list-merge proizvodi objedinjenu sortiranu listu l = 112236789.

- Algoritmom merge-sort se najpre deli ulazna lista na dve manje liste.
- Zatim se nastavlja ovo rekurzivno deljenje manjih lista sve dok se ne dobije to da svaka lista ima po jedan element.
- Zatim se dobijene manje liste objedinjuju u parovima, idući odozdo uz stablo, dok se cela lista ne dobije sortirana.


869221173
89213
6217
823
91
61
27
83
2
9
1
6
1
2
7
8
3


112236789
12389
1267
238
19
16
27
38
2
9
1
6
1
2
7
8
3