# Programiranje 2

## Predavanje 09 - vezane liste, drugi dio

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

12. svibnja 2023.

# Sortirano ubacivanje elementa u listu

U zadanu, uzlazno sortiranu, vezanu listu cijelih brojeva, zadanu pokazivačem `prvi` treba ubaciti jedan element, zadan pokazivačem `novi` tako da nova lista također bude uzlazno sortirana. Polazna lista smije biti i prazna.

Rješenje problema ima dva ključna koraka: a) pronalaženje odgovarajuće pozicije na koju treba ubaciti novi element (traženje po listi) i b) ubacivanje zadanog elementa na tu poziciju u listu.

Poziciju za ubacivanje novog elementa (pošto je lista sortirana) tražimo tako da preskačemo sve elemente čiji sadržaj je manji od sadržaja novog, a stajemo na prvom elementu čiji sadržaj je veći ili jednak sadržaju novog (ako takav postoji). Time nalazimo element **ispred** kojeg treba ubaciti novi element. Zato koristimo dva pokazivača: `pom` - na element čiji sadržaj testiramo i `preth` - na element ispred tog (ako postoji). Umjesto `pom` možemo koristiti i `preth->sljed`.

# Sortirano ubacivanje elementa u listu

Ubacivanje iza elementa `preth` radimo kao u funkciji `ubaci_iza`.

- Novom elementu postavimo sljedeći.
- Prethodnom elementu (ako postoji) promijenimo sljedeći u `novi`, a ako prethodnog nema, pokazivač `prvi` postavimo na `novi` (u tom slučaju je `novi` prvi element u listi).

```
1  lista sortirano_ubaci(lista prvi, lista novi){
2   /* Insertion sort za jedan element. */
3   /* Ne provjerava novi != NULL. */
4   lista preth, pom;
5   pom = prvi;
6   while (pom != NULL && pom->broj < novi->broj) {
7         preth = pom;
8         pom = preth->sljed; /* pom = pom->sljed;*/
9       }
```

# Funkcija sortirano_ubaci

```
10  /* Ispod je ubaci_iza(prvi, preth, novi) s
11  promjenom uvjeta. To radi i za prvi == NULL.*/
12
13   if (pom == prvi) { /* preth ne treba. */
14       novi->sljed = prvi;
15       prvi = novi;
16   }
17   else { /* Tu je pom == preth->sljed. */
18       novi->sljed = preth->sljed;
19       preth->sljed = novi;
20   }
21
22   return prvi; }
```

# Kraći zapis funkcije sortirano_ubaci

Primijetimo da pom uvijek pokazuje na listu sljedbenika elementa novi. Zato možemo pisati novi->sljed = pom;

```
lista sortirano_ubaci(lista prvi, lista novi){

 lista preth, pom;
 pom = prvi;
 while (pom != NULL && pom->broj < novi->broj) {
     preth = pom;
     pom = preth->sljed; /* pom = pom->sljed;*/ }

 novi->sljed = pom;

 if (pom == prvi) prvi = novi;
 else preth->sljed = novi;

 return prvi; }
```

# Sortiranje ubacivanjem — *Insertion sort*

Sortirano ubacivanje jednog elementa može se koristiti kao dio algoritma za sortiranje niza podataka.

**Sortiranje ubacivanjem ili Insertion sort:**

- krećemo od prazne liste (koja će sadržavati sortirane elemente),
- sortirano ubacujemo jedan po jedan element iz niza kojeg treba sortirati
- postupak ponavljamo dok ne ubacimo sve elemente

Vezana lista je prikladna struktura za realizaciju algoritma (jednostavnost ubacivanja elemenata, nema potrebe za realokacijom)!

# Sortiranje ubacivanjem — *Insertion sort*

U algoritmu nije bitno kako nastaju pojedini elementi sortirane liste. Možemo redom čitati brojeve (sadržaje) i kreirati pripadne elemente (jedan po jedan) ili krenuti od postojeće nesortirane liste elemenata iz koje izbacujemo jedan po jedan element (npr. s početka).

Složenost ovog algoritma je $\mathcal{O}(n^2)$. Zato se u praksi koristi samo za vrlo kratke liste. Postoje puno bolji algoritmi za sortiranje liste (npr. `MergeSort`).

# Izbacivanje traženog elementa iz liste

Iz vezane liste cijelih brojeva, zadane pokazivačem `prvi` na prvi element, želimo obrisati prvi element s parnim brojem (kao sadržajem). Ako takvog elementa nema, lista se ne mijenja. Polazna lista smije biti prazna.

Opet koristimo princip *izbaci* ili *obriši iza nekog* kao u funkciji `obrisi_iza`.

```
1  lista obrisi_prvi_parni(lista prvi){
2
3        lista preth, pom;
4        if (prvi == NULL) return NULL;
5        pom = prvi;
```

# Izbacivanje traženog elementa iz liste

```
6        while (pom != NULL && pom->broj % 2 != 0) {
7                 preth = pom;
8                 pom = preth->sljed;}
9
10       if (pom != NULL) {
11          if (pom == prvi)
12             prvi = prvi->sljed;
13          else
14            preth->sljed = pom->sljed;
15         free(pom);
16         return prvi;
17       }
```

Za ulaz: `2->4->5->6->8->9->10->NULL` i 6 poziva funkcije `obrisi_prvi_parni` dobijemo listu `5->9->NULL`. Zadnji poziv funkcije **ne mijenja** listu jer u tom trenutku lista više nema parnih elemenata.

# Zadaci na temu ubaci/izbaci

Pišemo funkciju `izbaci_iza` sa zaglavljem: `lista izbaci_iza(lista prvi, lista preth, lista *p_izbacen);`

Funkcija treba iz liste zadane pokazivačem `prvi`, koji pokazuje na prvi element, izbaciti element koji se nalazi iza elementa na kojeg pokazuje `preth`. Za razliku od funkcije `obrisi_iza`, izbačeni element se ne briše već treba vratiti pokazivač na njega kroz varijabilni argument `*p_izbacen` (slično kao u funkciji `ubaci_na_kraj`).
Vrijednost funkcije je **pokazivač** na **prvi** element dobivene liste.
Funkcija treba raditi u svim slučajevima:

- `prvi == NULL` - ne radi ništa, vraća `NULL` i `*p_izbacen = NULL`,
- `preth == NULL` - izbacuje **prvi** element liste
- `preth == zadnji`, tj. `preth->sljed == NULL` - ne izbacuje **ništa** i vraća `*p_izbacen = NULL`.

## Zadaci na temu `ubaci/izbaci`

Vezana lista cijelih brojeva zadana je pokazivačem `prvi` na prvi element. Treba napisati funkciju koja rastavlja tu listu u dvije liste, tako da prva lista sadrži samo elemente s parnim sadržajem, a druga lista sadrži samo elemente s neparnim sadržajem iz polazne liste. Funkcija treba vratiti pokazivače na te dvije liste kroz varijabilne argumente (ili vrijednost i varijabilni argument).

Varijacije:

- relativni poredak elemenata u dobivenim listama smije biti bilo koji - obratan od polaznog
- mora biti isti kao u polaznoj listi

**Rastav liste** treba napraviti samo promjenama veza elemenata (pokazivača), **zabranjeno je mijenjati sadržaj elemenata, alocirati i dealocirati dodatnu memoriju, tj. koristiti pomoćne elemente, polja itd.** Navedeno vrijedi u svim zadacima s vezanim listama, uključujući kolokvije (osim ako je u zadatku navedeno drugačije).

## Zadaci na temu - *preuredi listu*

Vezana lista cijelih brojeva zadana je pokazivačem `prvi` na prvi element. Napišite funkciju koja preuređuje tu listu, tako da na početku liste budu svi elementi s parnim sadržajem, a na kraju liste (iza svih parnih) budu svi elementi s neparnim sadržajem iz polazne liste. Funkcija treba vratiti pokazivač na prvi element preuređene liste. Preuređenje liste treba napraviti samo promjenama veza elemenata (pokazivača).

Varijacije: relativni poredak elemenata u parnom i neparnom dijelu dobivene liste

- može biti proizvoljan
- mora biti isti kao u polaznoj listi

Unutar funkcije smijete koristiti rastav liste u dvije liste, a onda spajanje dvije liste - konkatenacija.

Probajte zadatak riješiti bez toga, tj. unutar jedne liste.

# Zadaci na temu - *preuredi listu*

Vezana lista brojeva zadana je pokazivačem `prvi` na prvi element. Napišite funkciju `okreni_listu`: `lista okreni_listu(lista prvi);` koja preuređuje tu listu tako da cijelu listu okreće naopako, tj. invertira poredak elemenata u listi.
Funkcija treba vratiti pokazivač na prvi element okrenute liste (to je zadnji element u polaznoj listi, ako postoji). Invertiranje liste treba napraviti **samo promjenama veza elemenata (pokazivača)**.

# Spajanje (konkatenacija) dvije liste

Zadane su dvije vezane liste cijelih brojeva pokazivačima `prvi_1` i `prvi_2` na prvi element odgovarajuće liste. Te dvije liste treba spojiti u jednu listu, tako da druga lista bude iza prve (poredak elemenata u pojedinoj listi se ne mijenja). Ova operacija se još zove i konkatenacija - po analogiji s konkatenacijom dva stringa (funkcija `strcat`). Obje polazne liste smiju biti prazne. U spojenoj listi, sljedbenik zadnjeg elementa prve liste mora biti prvi element druge liste.

- pronalazimo zadnji element u prvoj listi (kao u funkciji `trazi_zadnji`)
- postavljamo njegovog sljedbenika na prvi element druge liste

Ako je prva lista prazna, rezultat je druga lista (koja također može biti prazna).

# Funkcija spoji_dvije

```
lista spoji_dvije(lista prvi_1, lista prvi_2){
    lista pom;

    if (prvi_1 == NULL) return prvi_2;
       /* Nadji zadnjeg u prvoj i spoji drugu. */
    for (pom = prvi_1; pom->sljed != NULL;
                        pom = pom->sljed);
    pom->sljed = prvi_2;

    return prvi_1;
}
```

# Funkcija spoji_dvije_rek

Konkatenaciju možemo realizirati i rekurzivnom funkcijom:

```
lista spoji_dvije_r(lista prvi_1, lista prvi_2){

      if (prvi_1 == NULL) return prvi_2;
      if (prvi_1->sljed == NULL) /* = zadnji_1. */
              prvi_1->sljed = prvi_2;
      else /* Skrati prvu listu (bez prvog).
            Ne trebamo vrijednost funkcije! */
          spoji_dvije_r(prvi_1->sljed, prvi_2);

      return prvi_1;
}
```

Za dvije ulazne liste: `1 -> 3 -> 5 -> 7 -> NULL` i `2 -> 4 -> 6 -> 8 -> NULL`, lista nakon konkatenacije je: `1 -> 3 -> 5 -> 7 -> 2 -> 4 -> 6 -> 8 -> NULL`.

# Sortirano spajanje (merge) dvije sortirane liste

Ukoliko imamo dvije sortirane liste, spajanje ima smisla raditi tako da rezultantna lista ostane sortirana.

Rješavamo problem u kojem imamo dvije sortirane (uzlazno od početka prema kraju) vezane liste cijelih brojeva, zadane pokazivačima `prvi_1` i `prvi_2` na prvi element odgovarajuće liste. Te dvije liste treba spojiti u jednu listu, tako da spojena lista također bude uzlazno sortirana. Ova operacija se zove **sortirano spajanje** (eng. *merge*). Obje polazne liste smiju biti prazne.

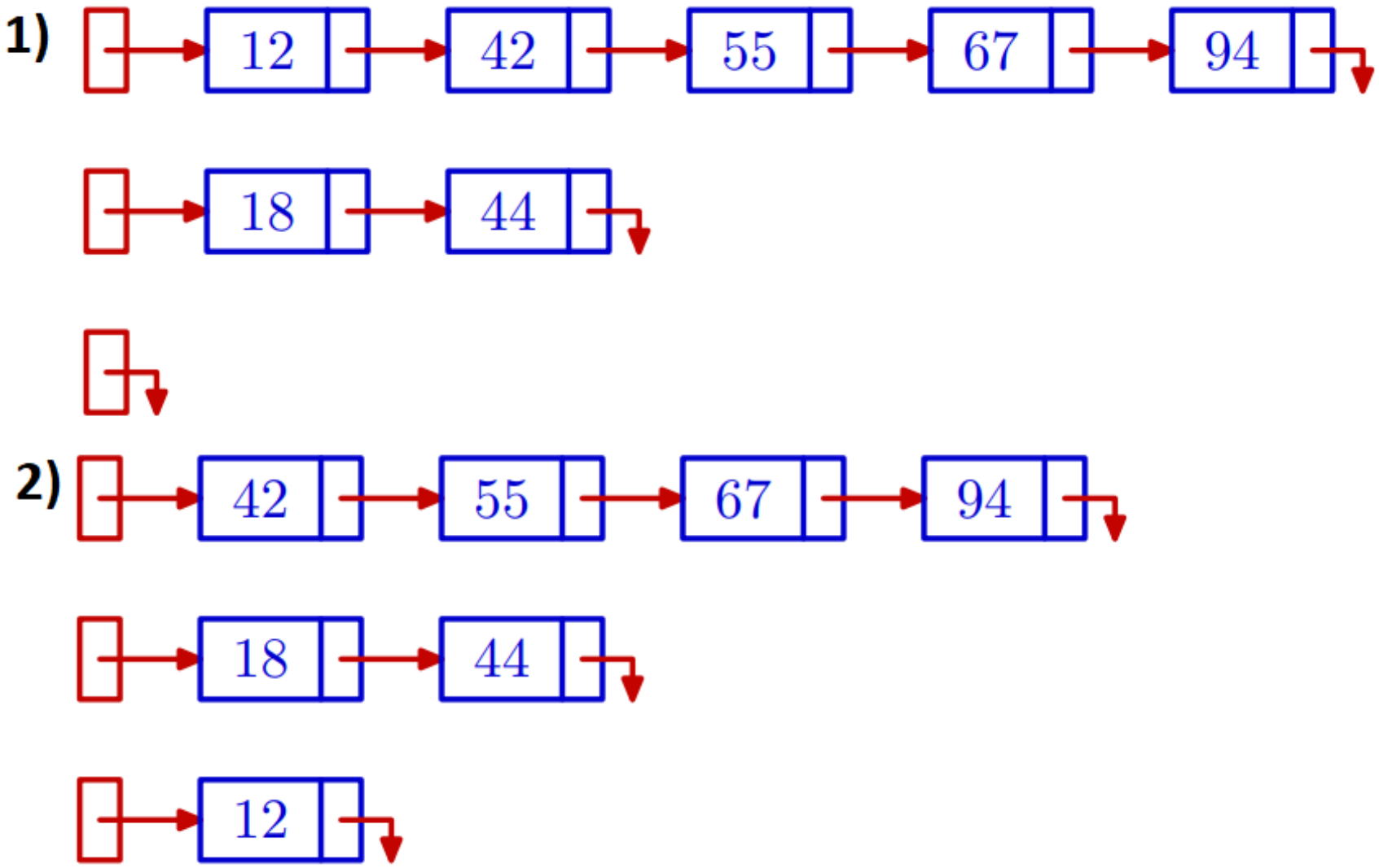
- Ako je barem jedna od dvije liste prazna, rezultat je ona druga lista (može također biti prazna).
- U protivnom su obje liste neprazne te koristimo *pametni Insertion sort*. Na početku je spojena lista prazna. Zatim gradimo spojenu listu (element po element, tako da stalno bude uzlazno sortirana).

# Sortirano spajanje (merge) dvije sortirane liste

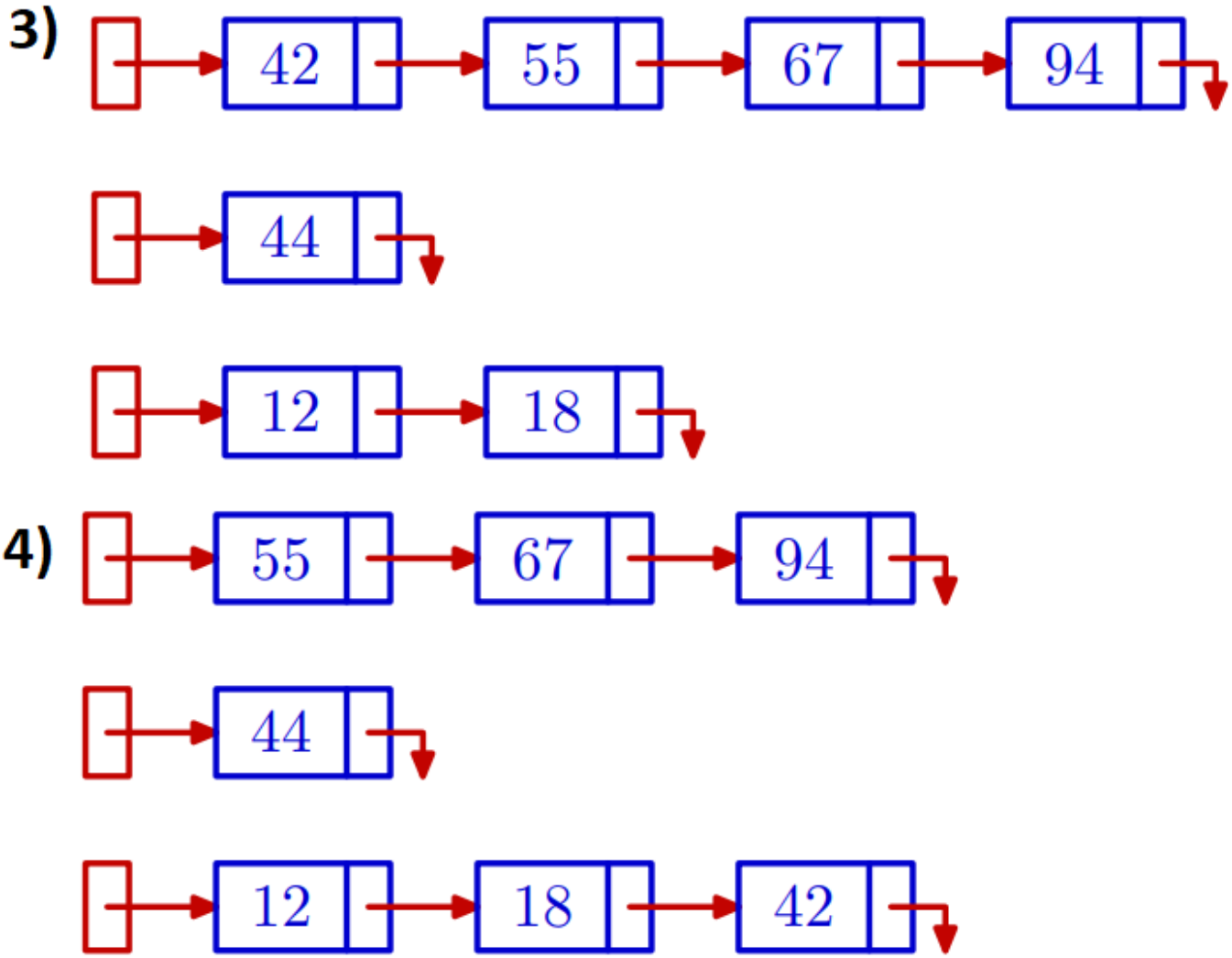
Pošto su obje ulazne liste već sortirane, ne trebamo koristiti funkciju `sortirano_ubaci` već sortiranu listu možemo kreirati učinkovitije.

Koristimo činjenicu da je najmanji element u svakoj od ulaznih sortiranih lista upravo prvi element. Uspoređujemo prve elemente i manji od njih je najmanji u obje liste. Izbacujemo ga iz odgovarajuće liste (s početka) i ubacimo na kraj spojene liste (on je $\geq$) od svih ranijih. Postupak ponavljamo dok su obje liste neprazne. Na kraju imamo jednu nepraznu listu koju spojimo na kraj spojene liste.

# *Merge* dvije sortirane liste

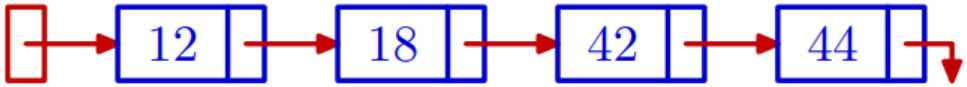


# *Merge* dvije sortirane liste



# *Merge* dvije sortirane liste

5)




6)

# Funkcija merge

```
1  lista merge(lista prvi_1, lista prvi_2){
2  /* Sortirano spaja dvije liste (merge). */
3        lista prvi = NULL, zadnji, pom;
4  /* Ako je jedna lista prazna, rezultat je
5  ona druga lista. */
6        if (prvi_1 == NULL) return prvi_2;
7        if (prvi_2 == NULL) return prvi_1;
8  /* U nastavku obrade, obje liste sigurno
9  NISU prazne. */
10 /* Prolaz po obje liste, sve dok su obje
11 neprazne. */
12
13       while (prvi_1 != NULL && prvi_2 != NULL) {
14 /* Nadji manjeg od prvih iz obje liste.
15 Izbaci ga s pocetka njegove liste. */
```

# Funkcija merge

```
16          if (prvi_1->broj <= prvi_2->broj) {
17              pom = prvi_1;
18              prvi_1 = prvi_1->sljed;
19           }
20          else {
21             pom = prvi_2;
22             prvi_2 = prvi_2->sljed;
23           }
24           /* Ubaci ga na kraj sortirane liste. */
25           if (prvi == NULL)
26               prvi = pom;
27           else
28               zadnji->sljed = pom;
29           zadnji = pom;
30    }
```

# Funkcija merge

```
31 /* Spoji ostatak na kraj sortirane liste. */
32     if (prvi_1 == NULL) zadnji->sljed = prvi_2;
33     if (prvi_2 == NULL) zadnji->sljed = prvi_1;
34     return prvi;
35 }
```

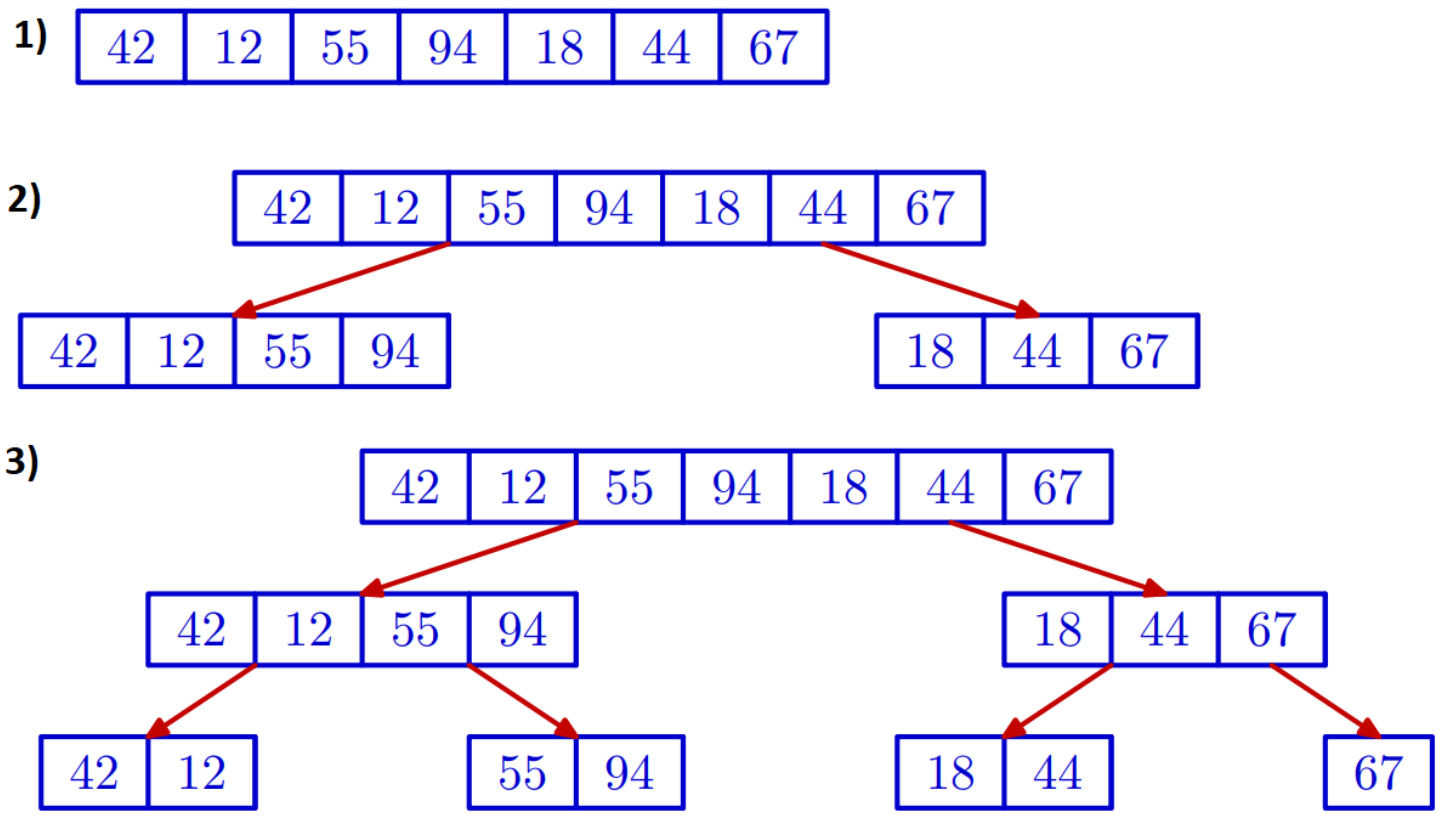
Za ulazne liste `1 -> 3 -> 5 -> 7 -> NULL` i `2 -> 4 -> 6 -> 8 -> NULL`, nakon sortiranog spajanja dobivamo listu `1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> NULL`.

## Sortiranje spajanjem - *Merge sort*

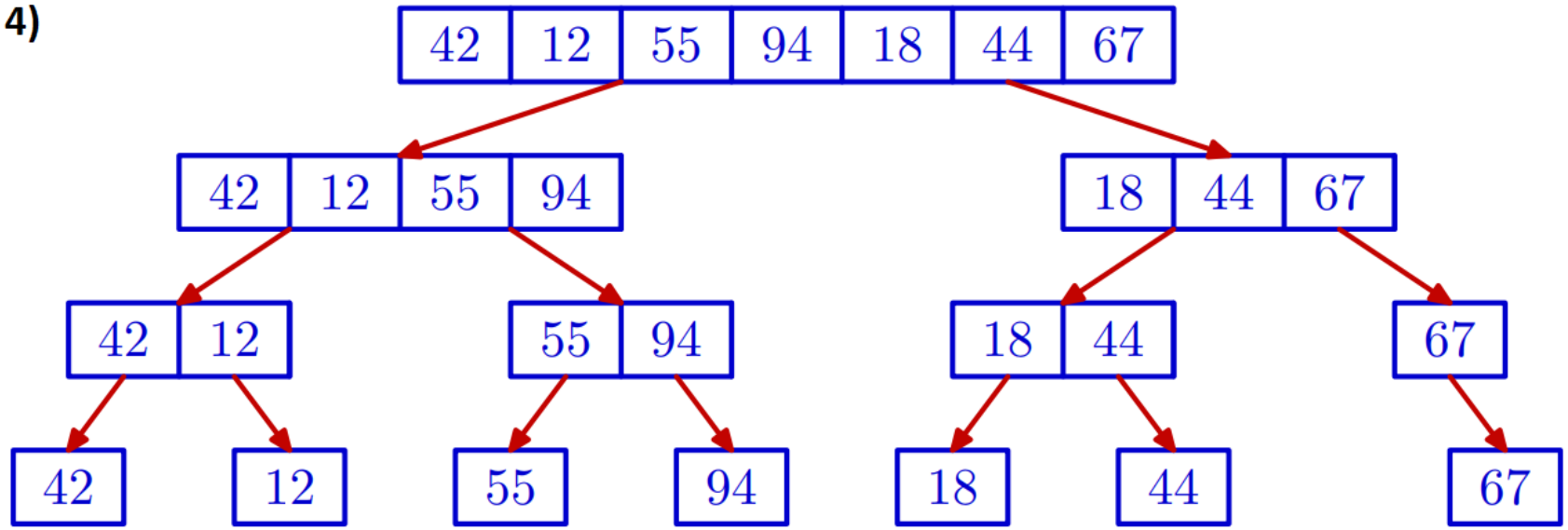
Zadana je jedna vezana lista cijelih brojeva pokazivačem `prvi` na prvi element liste. Listu treba uzlazno sortirati po sadržaju elemenata (od početka prema kraju liste) i vratiti pokazivač na početak sortirane liste. Polazna lista smije biti prazna. Sortiranje liste treba napraviti samo promjenama veza elemenata (pokazivača).

Za sortiranje liste koristimo `MergeSort` algoritam. To je rekurzivni algoritam za sortiranje niza podataka, baziran na operaciji `merge` (sortiranom spajanju već sortiranih nizova podataka). Osnovna ideja `MergeSort` algoritma je: a) podijeliti nesortirani niz (listu) podataka na dva dijela (podniza) podjednake duljine, b) rekurzivno sortirati svaki od nastalih podnizova, c) sortirano spojimo (`merge`) ta dva već sortirana podniza u jedan sortirani niz. Rekurzivno raspolavljanje i sortiranje radimo sve dok ne dobijemo trivijalni niz (prazan ili jednočlan). Takvi nizovi su već sortirano stoga samo vraćamo pokazivač na tu listu.

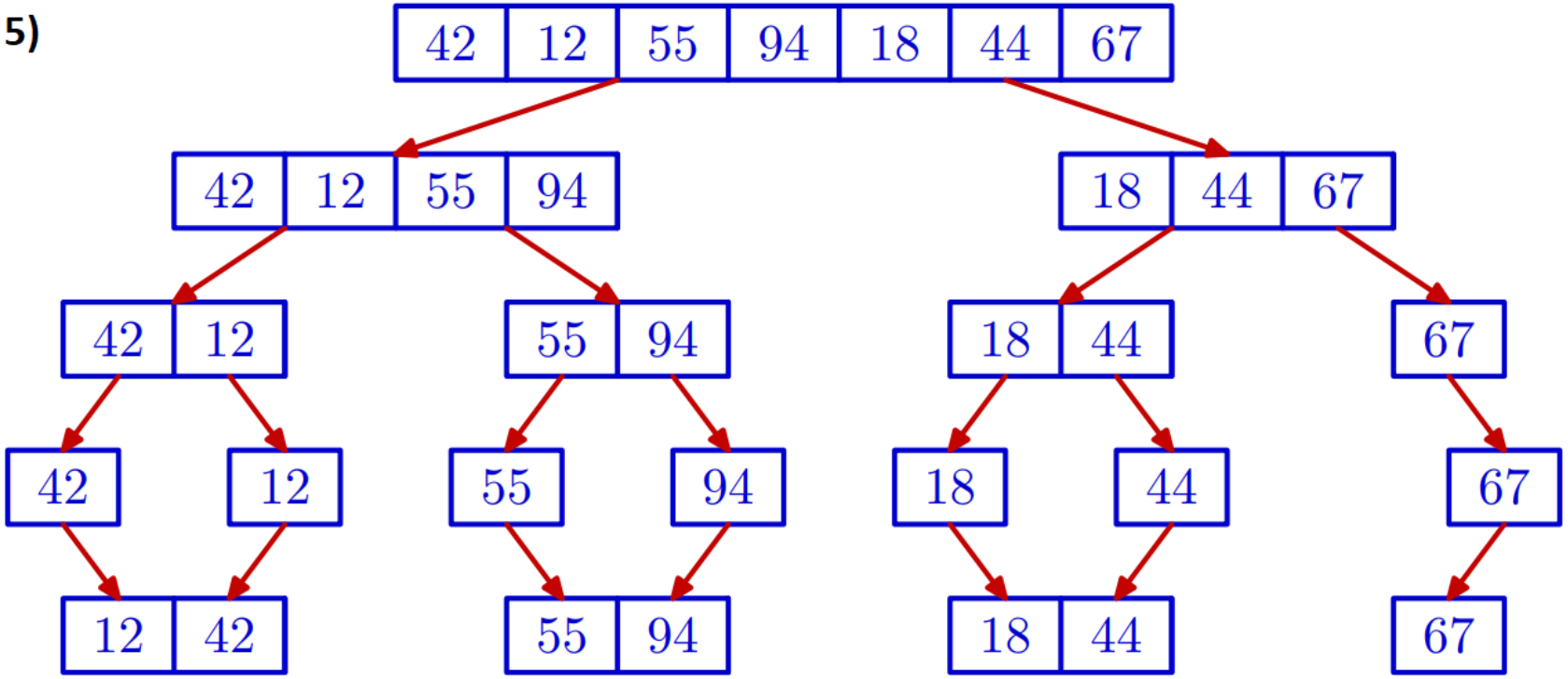
# Sortiranje spajanjem - *Merge sort* - primjer



# Sortiranje spajanjem - *Merge sort* - primjer

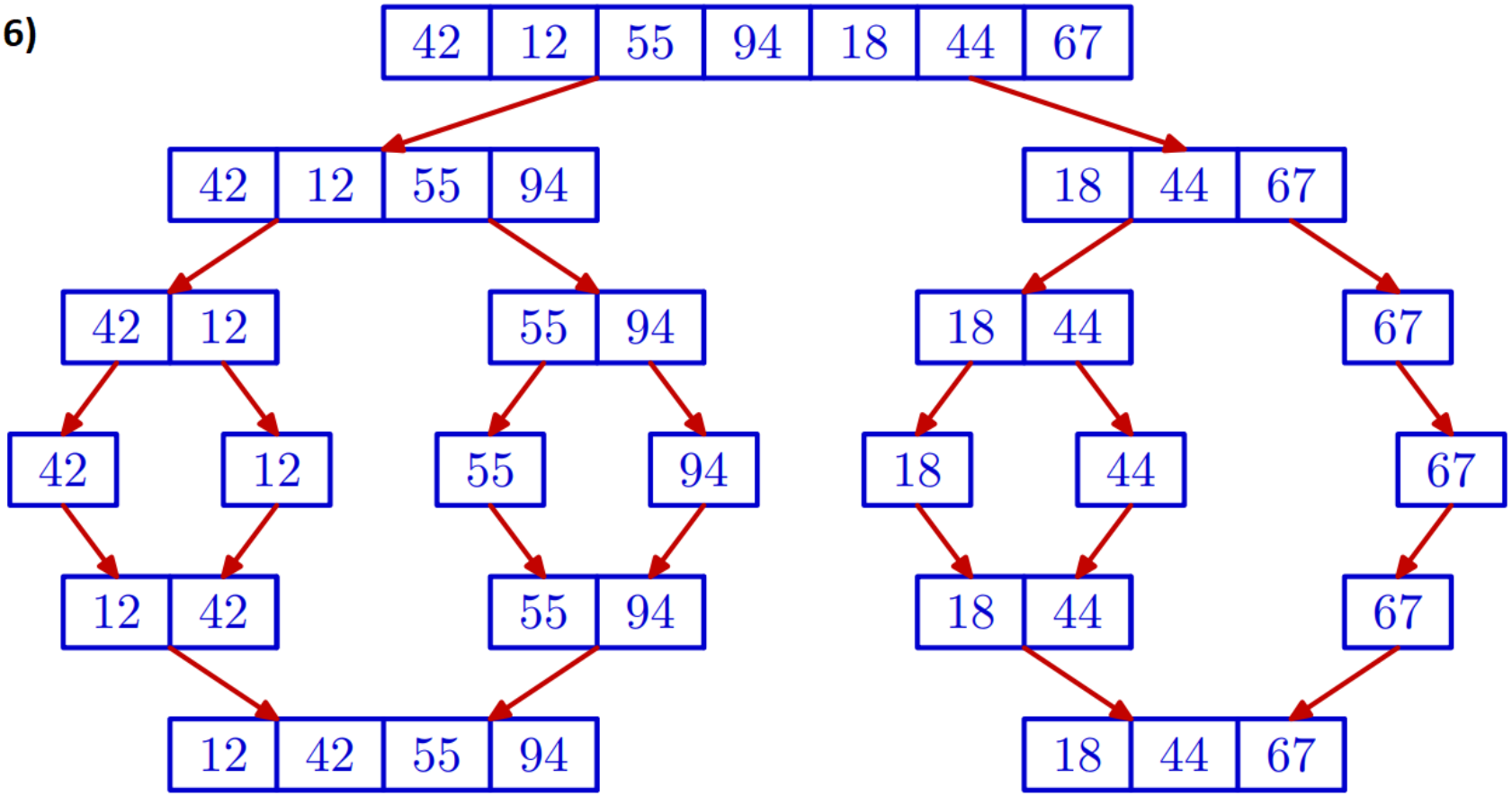


# Sortiranje spajanjem - *Merge sort* - primjer

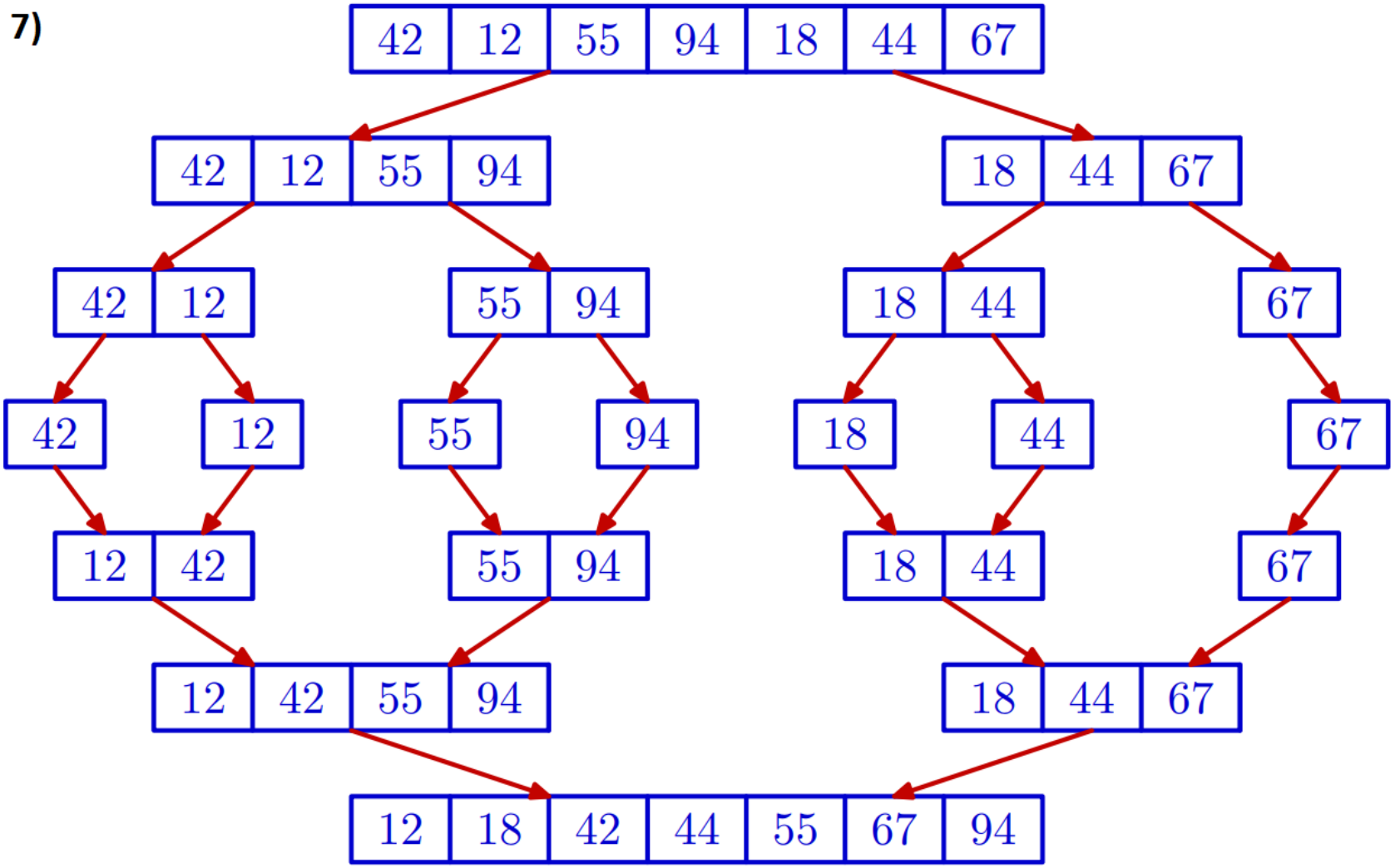


# Sortiranje spajanjem - *Merge sort* - primjer

6)



# Sortiranje spajanjem - *Merge sort* - primjer



# Sortiranje spajanjem - *Merge sort*

Raspolavljanje liste možemo realizirati na dva načina: a) preko pokazivača, pažljivim iteriranjem po listi (elegantnije ali sporije), b) brojanjem elemenata u listi (brže, potreban dodatan ulazni argument).

Složenost `MergeSort` algoritma u najgorem slučaju je $\mathcal{O}(n\ log(n))$.

**Dokaz** slijedi iz prethodne slike. Neka je $n$ broj elemenata u nizu i neka je $2^k < n \leq 2^k$ ($2^k$ je najmanja potencija broja 2 veća ili jednaka $n$), $k = \lceil log_2 n \rceil$.

Na slici je $n = 7$ i $k = 3$.

# Sortiranje spajanjem - *Merge sort*

Uzmimo da polazni, nesortirani niz, ima razinu (nivo) 0. Nakon toga, imamo točno $k$ horizontalnih razina raspolavljanja podnizova i sortiranog spajanja podnizova. Broj razina je $2k = 2log_2 n$.

Svako raspolavljanje i svaki `merge` traju linearno u duljini većeg odgovarajućeg niza (svaki element u većem niz prolazimo najviše jednom). Zbrajanjem operacija na istoj razini (imamo ih maksimalno $n$) zaključujemo da na svakoj razini imamo $\mathcal{O}(n)$ operacija. Množenjem broja operacija na razini i broja razin, dobivamo složenost algoritma $\mathcal{O}(n\ log(n))$.

Autor `MergeSort` algoritma je **John von Neumann**, 1945. godine. To je bio **prvi** program napisan za računalo (`EDVAC`) koje sprema i podatke i programe (von Neumann-ov model računala).

# Funkcija merge_sort

```
1  lista merge_sort(lista prvi){
2  /* Sortira listu Merge_Sort algoritmom. */
3       lista zadnji, prvi_2;
4  /* Test na praznu ili jednoclanu listu. */
5       if ((prvi == NULL) || (prvi->sljed == NULL))
6            return prvi;
7  /* U nastavku obrade, lista ima bar dva
8  elementa. */
9  /* Raspolovi listu. Pokazivac zadnji pokazuje
10 na zadnjeg u prvom dijelu. Pokazivac prvi_2 je
11 pomocni i sluzi za raspolavljanje liste. */
12      zadnji = prvi;
13      prvi_2 = prvi->sljed;
```

# Funkcija merge_sort

```
14 /* Pomicemo zadnjeg za JEDNO mjesto, prvi_2 za
15 DVA mjesta, dok prvi_2 ne bude na kraju liste. */
16      while ((prvi_2 != NULL) &&
17             (prvi_2->sljed != NULL)) {
18               zadnji = zadnji->sljed;
19               prvi_2 = prvi_2->sljed->sljed;
20         }
21 /* Pokazivac zadnji sad korektno pokazuje na
22 zadnjeg u prvom dijelu. Pokazivac prvi_2 zadamo
23 kao prvog u drugom dijelu (prvi iza zadnjeg) i
24 korektno zavrsavamo prvi dio. */
25     prvi_2 = zadnji->sljed;
26     zadnji->sljed = NULL;
27 /* Rekurzivno sortiranje i merge. */
28     prvi = merge(merge_sort(prvi),
29     merge_sort(prvi_2));
30    return prvi; }
```

# MergeSort

Za zadanu ulaznu listu brojeva `42 -> 12 -> 55 -> 94 -> 18 -> 44 -> 67 -> NULL`, nakon poziva funkcije `MergeSort` dobivamo listu `12 -> 18 -> 42 -> 44 -> 55 -> 67 -> 94 -> NULL`.

**Zadatak:** Napišite funkciju za `MergeSort` liste koja kao argument prima broj elemenata u listi.

**Zadatak:** Napišite funkciju za `MergeSort` na polju. Za `merge` (spajanje) smijete koristiti jedno dodatno pomoćno polje! Probajte koristiti što manje kopiranja iz jednog polja u drugo.

# Okretanje liste - rješenje

Zadana je vezana lista pokazivačem `prvi` na prvi element. Napišite funkciju koja preuređuje tu listu tako da cijelu listu okreće naopako, tj. invertira poredak elemenata u listi. Funkcija treba vratiti pokazivač na prvi element okrenute liste (zadnji element u polaznoj listi, ako postoji). Invertiranje liste treba napraviti samo promjenama veza elemenata (pokazivača).

Na početku je okrenuta lista prazna (`okr = NULL`). Prolazimo kroz zadanu listu i u svakom koraku:

- izbacimo prvi element iz preostale neokrenute liste
- ubacimo ga na početak okrenute liste (zadane s `okr`).

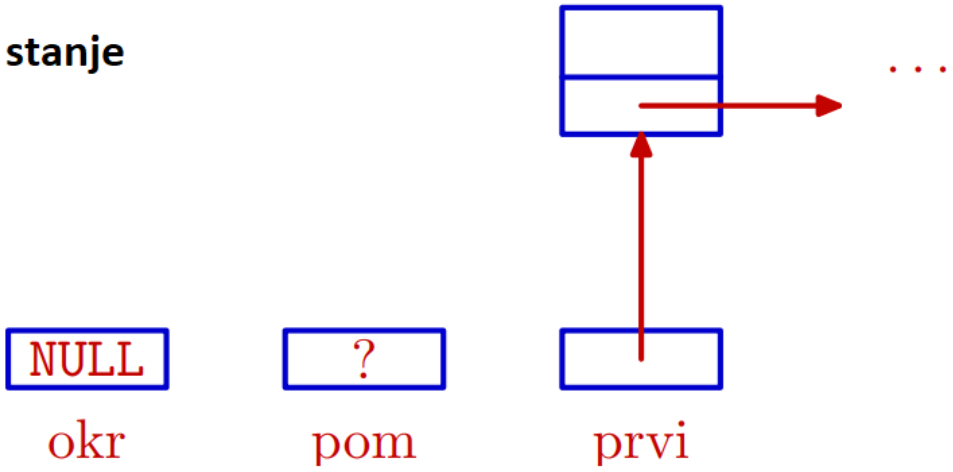
Postupak ponavljamo dok preostala ulazna lista nije prazna.

# Funkcija okreni_listu

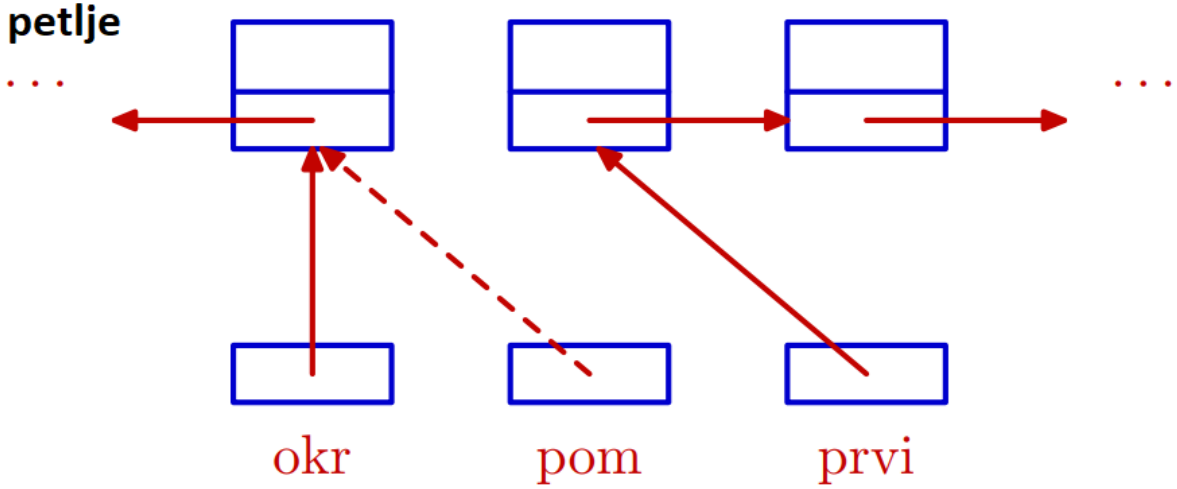
```
1  lista okreni_listu(lista prvi){
2
3        lista okr = NULL, pom;
4
5        while (prvi != NULL) {
6  /* Izbaci s pocetka stare. */
7              pom = prvi;
8              prvi = prvi->sljed;
9  /* Ubaci na pocetak okrenute. */
10             pom->sljed = okr;
11             okr = pom;
12         }
13       return okr;
14   }
```

# Okretanje liste - primjer

1) **Početno stanje**



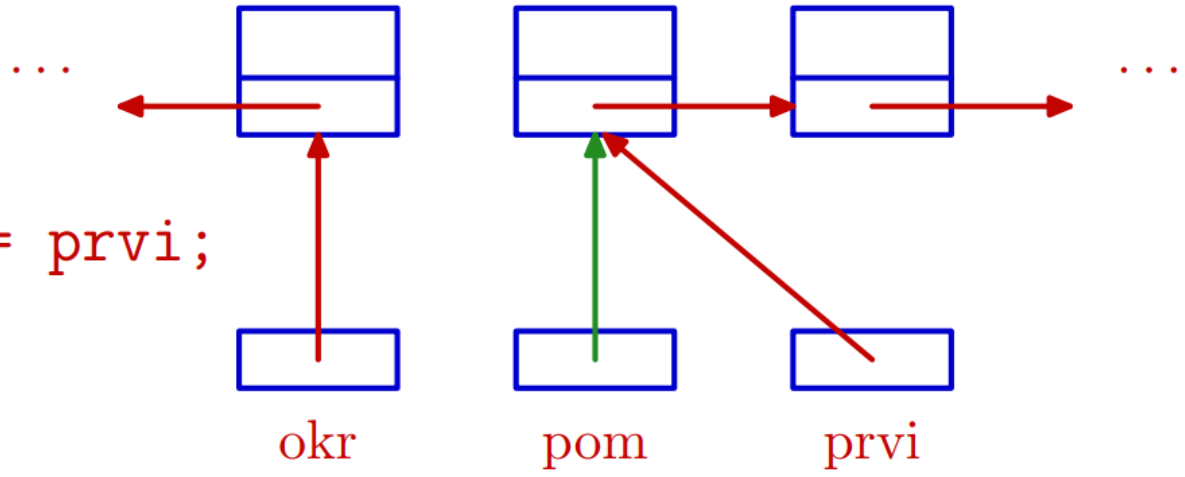

2) **Vrh petlje**




**pretpostavka**
**prvi!=NULL**

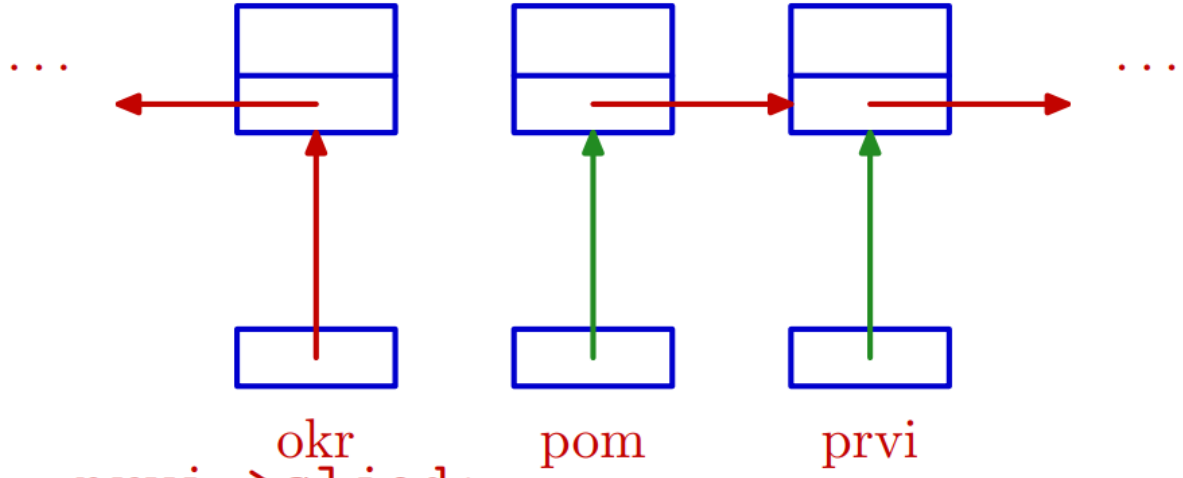
# Okretanje liste - primjer

3)




```
pom = prvi;
```

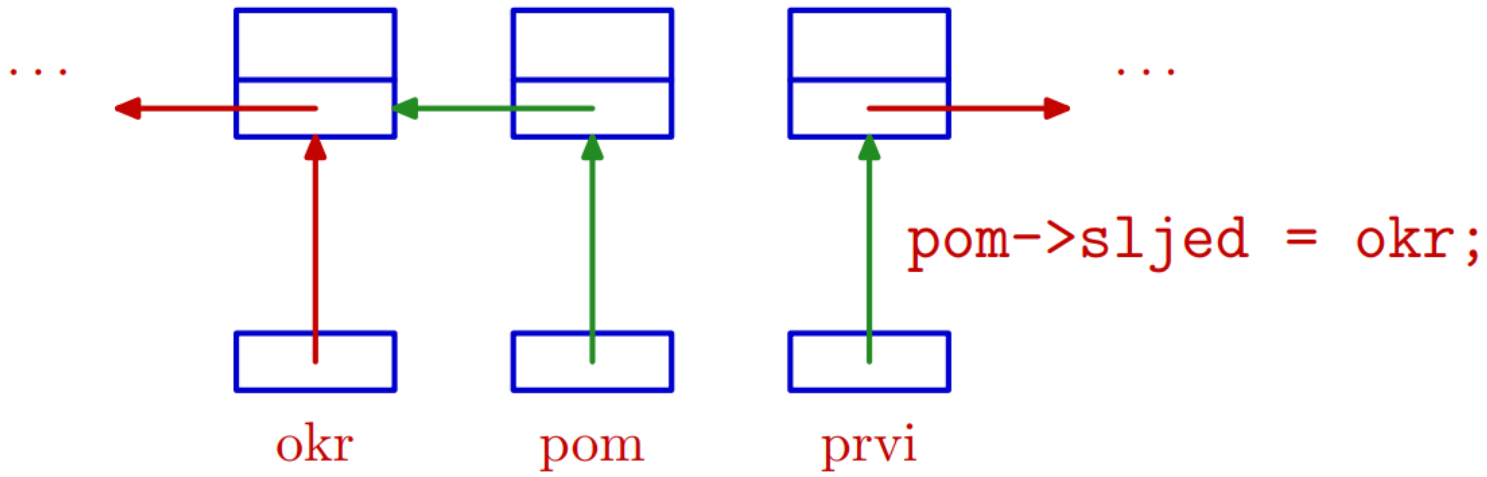
4)



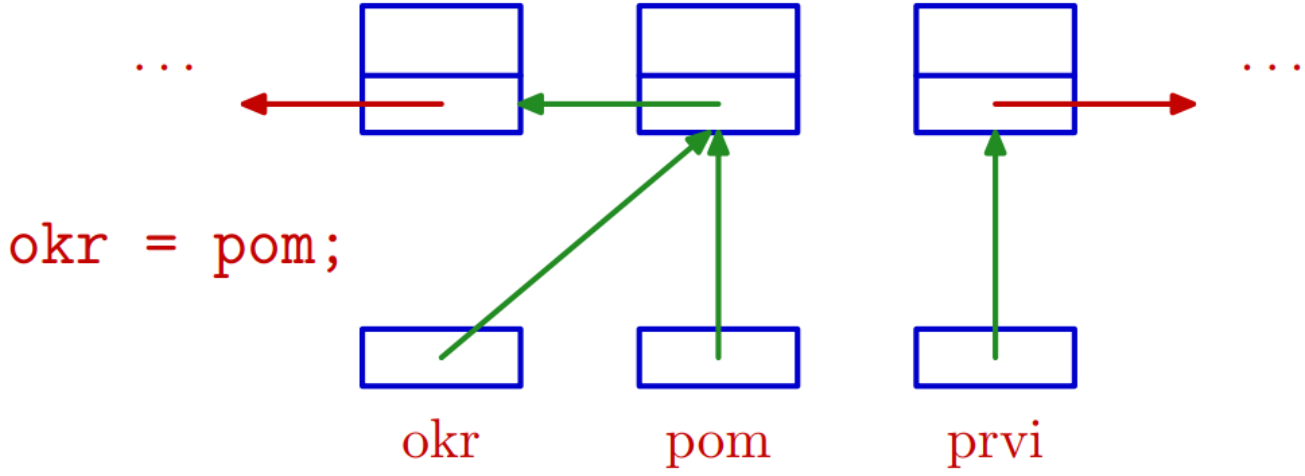

```
prvi = prvi->sljed;
```

# Okretanje liste - primjer

5)




6)



# Okretanje liste - drugo rješenje

Problem možemo riješiti i prozorom od nekoliko susjednih pokazivača koji prolaze kroz listu. U svakom koraku pomaknemo prozor za jedno mjesto unaprijed i okrenemo (usmjerimo pokazivač `sljed` u drugu stranu) jedan element u tom prozoru. Dovoljna su tri pokazivača:

- `preth` - pokazuje na prethodno okrenutu listu, u trenu kad ubacujemo novi element (pomoćni pokazivač)
- `ovaj` - pokazuje na element kojeg prebacujemo, to će biti (novi) početak okrenute liste
- `sljed` - pokazuje na početak preostalih elemenata originalne liste

Okretanje (izbacivanje/ubacivanje) odgovara naredbi: `ovaj->sljed = preth`.

# Funkcija okreni_listu - rješenje 2

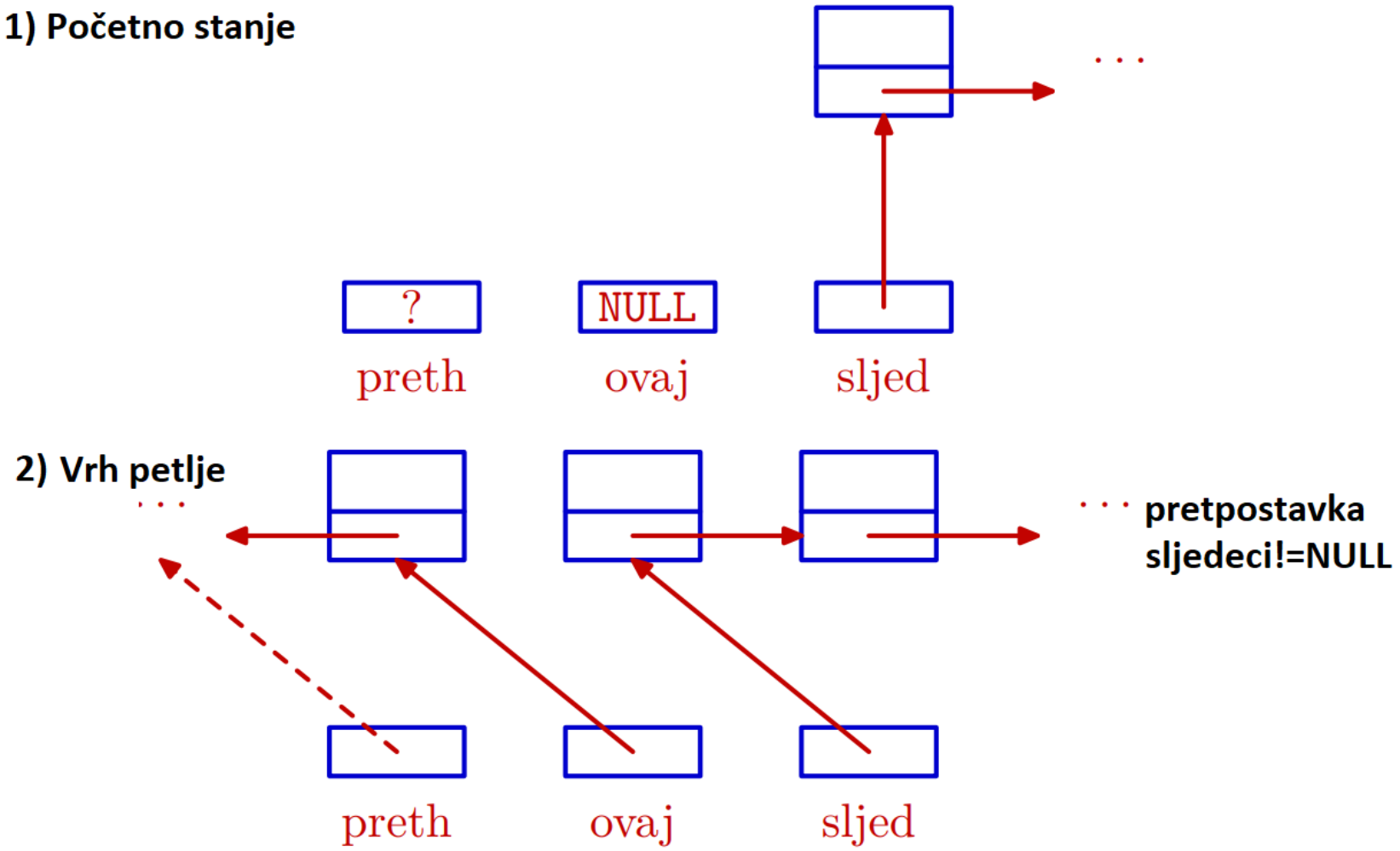
```
lista okreni_listu(lista prvi){
      lista preth, ovaj = NULL, sljed = prvi;

      while (sljed != NULL) {
            preth = ovaj;
            ovaj = sljed;
            sljed = sljed->sljed;
            ovaj->sljed = preth;
      }

      return ovaj;
}
```

# Funkcija okreni_listu - skraćeno rješenje 2

```
lista okreni_listu(lista prvi){

      lista preth, ovaj = NULL;

      while (prvi != NULL) {
          preth = ovaj;
          ovaj = prvi;
          prvi = prvi->sljed;
          ovaj->sljed = preth;
       }

      return ovaj;
}
```

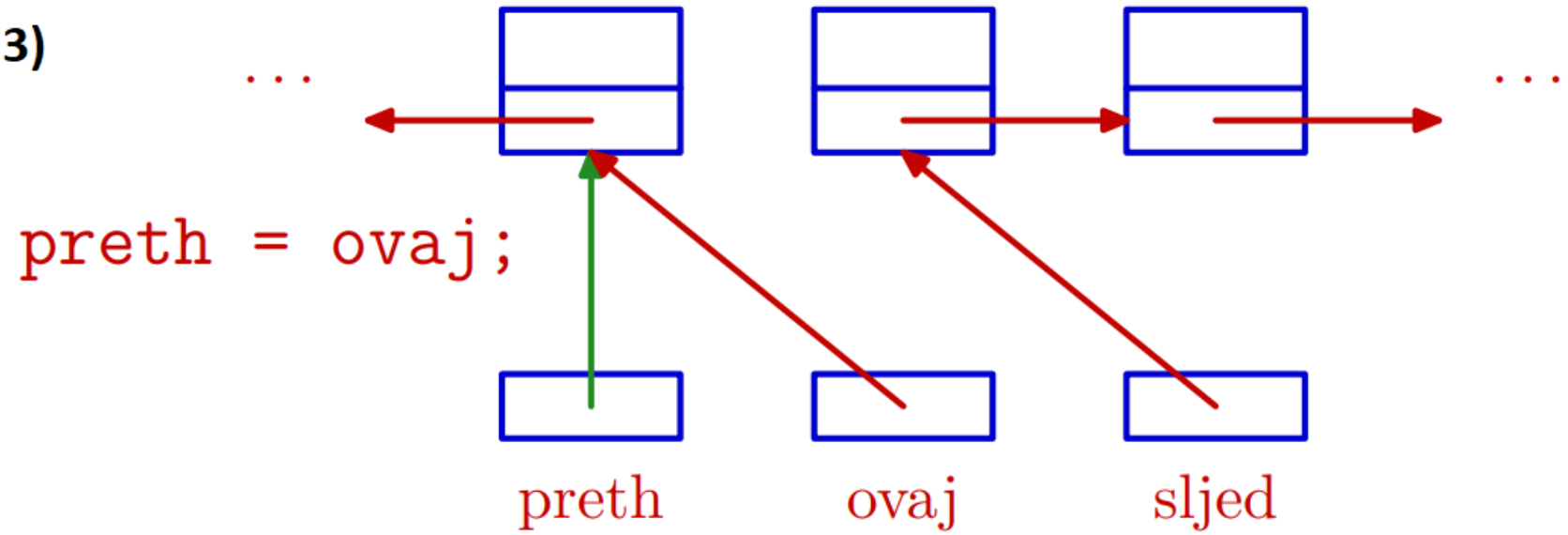
sljed pamtimo u varijabli prvi. Algoritam je sličan algoritmu iz prvog rješenja.

# Okretanje liste - primjer

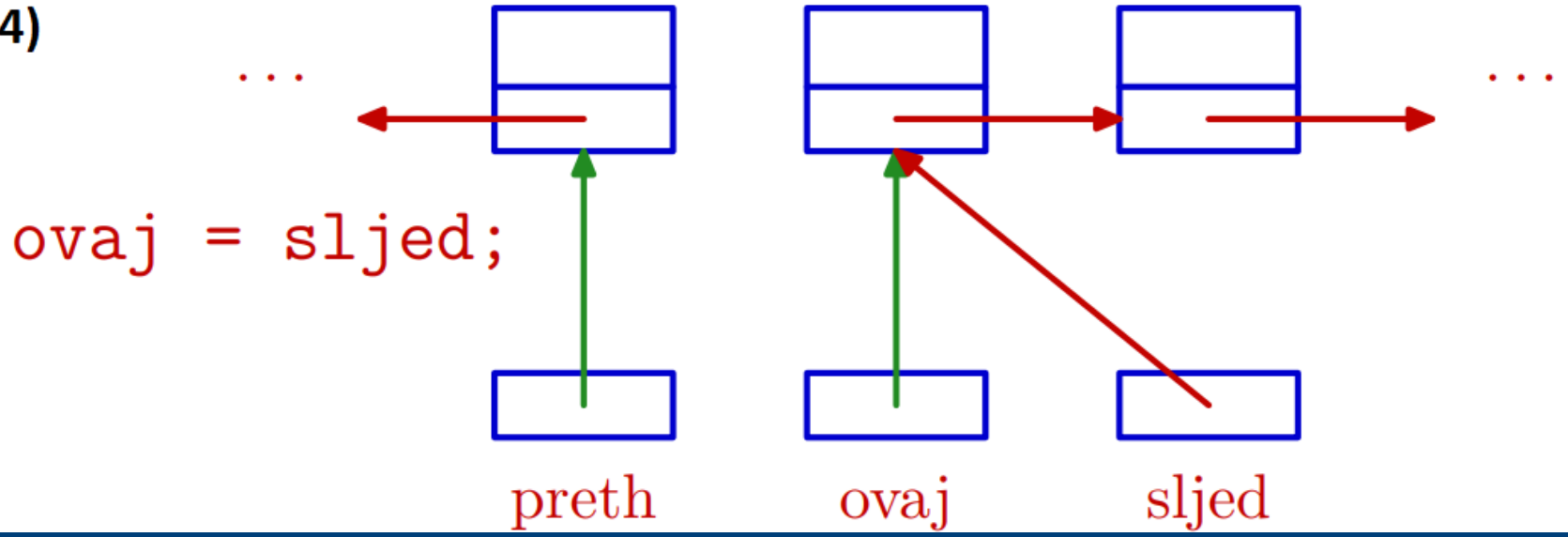


# Okretanje liste - primjer

3)

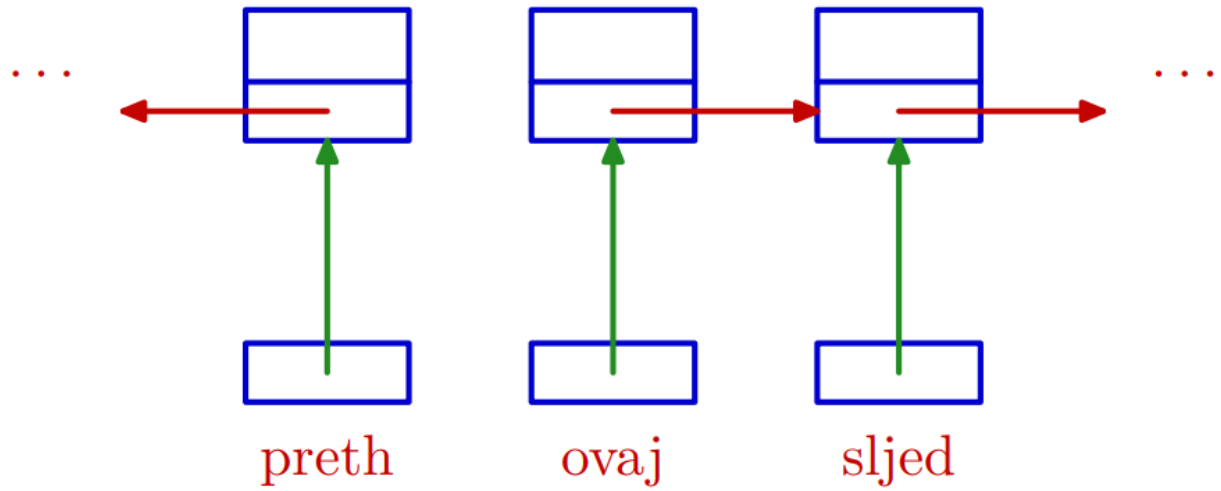



4)



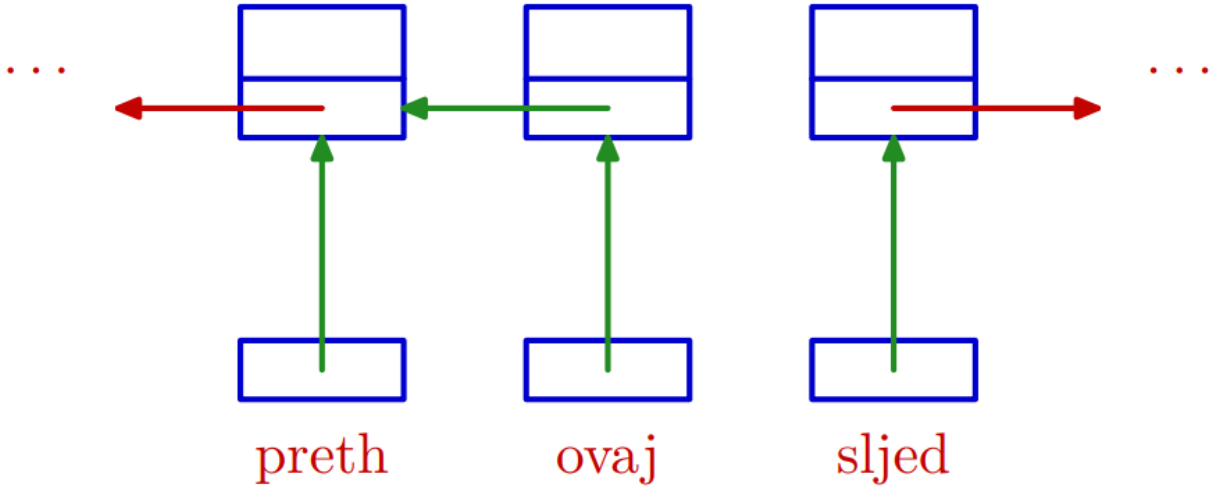
# Okretanje liste - primjer

5)




```
sljed = sljed->sljed;
```

6)




```
ovaj->sljed = preth;
```