Algoritmi i strukture podataka

Studijski programi:
Softversko inženjerstvo
Računarska tehnika
Matematika-informatika

Ulfeta Marovac 2020

Ulančane liste

Ulfeta Marovac 2020

Memorijska reprezentacija

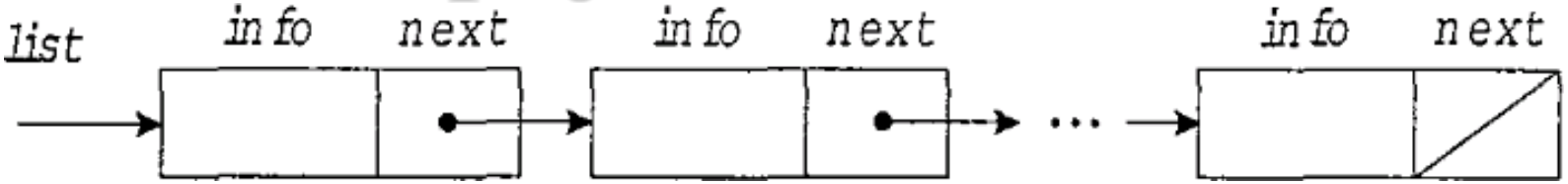
- Postoje dva osnovna načina fizičke implementacije struktura podataka:
  - Sekvencijalna reprezentacija
  - Ulančana reprezentacija.

Reprezentacija	Prednosti	Mane
Sekvencijalna	Efinasan pristup proizvoljnom elementu. Isti logički i fizički poredak.	Nefikasne operacije umetanja i brisanja elemenata. Statički alociran fiksan memorijski prostor
Ulančana	Dinamička alokacija meomorije. Lako ubacivanje i brisanje elemenata liste.	Pristup proizvoljnom elementu indirektan. Fizički poredak se ne može izvesti iz logičkog

Osnovni pojmovi i osobine

- Ulančana implementacije linearne liste se naziva ulančanom listom **(linked list)**
- U ulančanoje reprezentaciji, uzastopni elementi mogu biti bilo gde u memoriji.
- Kako se logički poredak elemenata ne može izvesti iz fizičkog poretka, elementi liste sadrže eksplicitnu adresu narednog elementa – pokazivač na njega.
- Dakle, svaki element liste (**čvor**), pored informacionog sadržaja elementa(**info**) sadrži i pokazivač(**next**) na sledbenika.

Osnovni pojmovi i osobine



Jednostruko ulančana lista

- Listi se pristupa preko spoljašnjeg pokazivača (**list**) koji pokazuje na prvi čvor, a koji nije deo liste
- Prvi čvor se naziva glavom (**head**) liste
- Poslednji čvor se naziva repom (**tail**) liste
- Rep(tail) nema svog sledbenika pa sadrži prazan pokazivač, to je kraj liste
- Ukoliko je lista prazna, list ima vrednost praznog pokazivača(list = **nil**)

Vrste listi

- **Po načinu povezanosti**
  - **Jednostruko ulančana lista** – svaki čvor ukazuje samo na svog sledbenika
  - **Dvostruko ulančana lista** – u kojoj svaki čvor ima dva pokazivača, na predgodnika (prev) i sledbenika(next)
  - **Kružna** – ukoliko je lista povezana tako da poslednji čvor ukazije na prvi
  - **Nekružna** – poslednji element ne ukazuje ni na jedan element
- **Po organizaciji**
  - **Uređena** – ukoliko su čvorovi poređani po rastućim(ili opadajućim) vrednostima, ili se po sadržaju *info* može definisati f-ja poređenja
  - **Neuređena** – ukoliko poredak ne zavisi od vrednosti sadržaja (info)
- Lista u kojoj su čvorovi istog tipa je **homogena**.

Notacija pri operacijama sa listama

- ***p*** pokazivač na neki čvor liste
- ***info(p)*** korisni sadržaj ovog čvora
- **next(p)** je pokazivač na sledbenika
- **prev(p)** je pokazivač na prethodnika

Dinamička alokacija

- Ulančana liste je ***dinamička struktura*** čiji broj čvorova varira saglasno operacijama umetanja i brisanja.
- Prostor za neki čvor se alocira tek pri operaciji umetanja a pri operaciji brisanja se zauzeti prostor oslobađa.
- Znači, potreban je mehanizam koji vrši **dinamičku alokaciju u dealokaciju prostora.**

Dinamička alokacija

- Neka je raspoloživi slobodni prostor *(**storage pool**)* podeljen u jedinice koje imaju strukturu čvora predpostavljene liste
- Pri formiranju novog čvora liste poziva se posebna funkcija **GETNODE** koja uzima jedan slobodan čvor i stavlja ga na raspolaganje algoritmu vraćajući njegovu adresu u memoriji (***pokazivač na njega***)
- Procedura **FREENODE(p)** oslobađa čvor sa zadatom ***adresom p***
- Slobodni prostor je pogodno organizovati u vidu ulančane liste slobodnih čvorova

Dinamička alokacija

- Na početku se svi slobodni čvorovi uvežu u jednostruko ulančanu listu na koju ukazuje globalni pokazivač ***avail***
- Funkcija GETNODE izvlači prvi slobodan čvor iz liste, postavlja polje pokazivača na nultu vrednost(nil), ažurira pokazivač *avail* i vraća adresu pokazivača

```
GETNODE
if (avail = nil) then
    ERROR(Nema slobodnog čvora)
end_if
q = avail
avail = next(avail)
next(q) = nil
return q
```

Dinamička alokacija

- Ako se desi da nema slobodnih čvorova, pretpostavlja se postojanje procedure ERROR koja vrši obradu greške

```
GETNODE
if (avail = nil) then
    ERROR(Nema slobodnog čvora)
end_if
q = avail
avail = next(avail)
next(q) = nil
return q
```

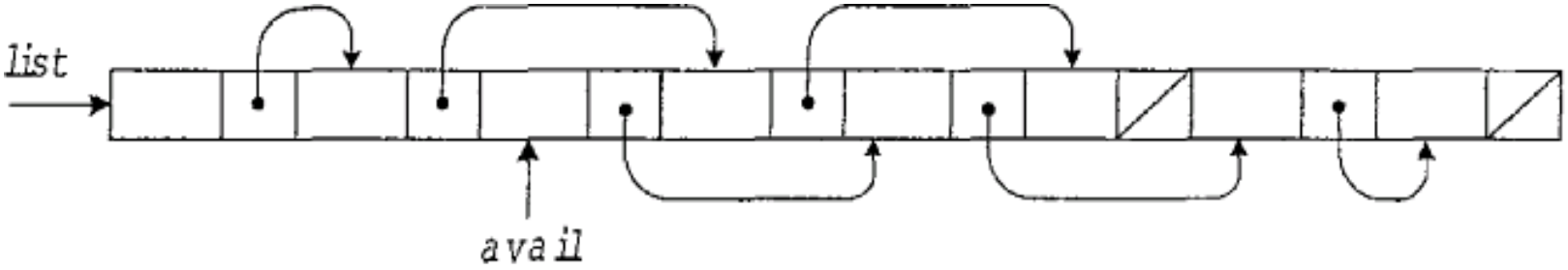
Dinamička alokacija

- Procedura FREENODE(p) vraća oslobođeni prostor na početak liste
- Pokazivač avail se postavlja na njegovu adresu

$$\underline{\text{FREENODE}(p)}$$
$$next(p) = avail$$
$$avail = p$$

Dinamička alokacija

- Opisani način odgovara ulančanoj reprezentaciji steka
- Ovakva struktura se obično naziva stek slobodnog prostora **(availability stack)**



Korisnička lista na koju ukazuje pokazivač *list* i lista slobodnog prostora na koju ukazuje pokazivač *avail*

Operacije sa jednostruko ulančanim listama

- Tipične operacije: pretraživanje, umetanje i brisanje
- Oparacije sa listom manipulišu sa pokazivačima
- Sa pokazivačima su dozvoljene aktivnosti:
  - Testiranja pokazivača na nil
  - Provera na jednakost pokazivača sa drugim pokazivačem
  - Postavljanje pokazivača na nil
  - Dodela vrednosti drugog pokazivača ili adrese datom pokazivaču

Umetanje u listu

- Umetanje čvora u neuređanu listu može da bude izvršeno na bilo kojem mestu(početak, sredina, kraj)
- Operacija INSERT- AFTER(p, x) umeće u listu novi čvor sa sadržajem x **iza** čvora čija je adresa p

INSERT-AFTER(p, x)

$q = \text{GETNODE}$

$info(q) = x$

$next(q) = next(p)$

$next(p) = q$

Umetanje u listu

- Kod operacije INSERT- BEFORE mora prvo da se pristupi prethodnom čvoru, jer njegov pokazivač treba povezati na novi čvor
- Ovo se može izvršiti tako sto se novi čvor prvo ubaci iza tekućeg elementa a onda oni zamene informacione sadržaje
- Postiže se efekat kao da je novi čvor ubačen **ispred** ukazanog čvora

INSERT-BEFORE(p, x)
q = GETNODE
$next(q) = next(p)$
$info(q) = info(p)$
$info(p) = x$
$next(p) = q$

Brisanje

- Operacija brisanja može da ukloni čvor sa bilo kojeg mesta u listi
- Razmatraćemo brisanje sa *sredine* i *kraja* liste
- *Prethodnik* čvora koji se uklanja treba da pokazuje na njegovog *sledbenika*
- Kod nekružne jednostruko spregnute liste nije moguće brisati čvor na koji pokazuje pokazivač jer nema načina da se dođe do njegovog prethodnika da bi mu se ažuriralo polje pokazivača

Brisanje

- Zato je pri brisanju tekućeg čvora u operaciji DELETE-AFTER(p) potebno je dati pokazivač p na njegovog prethodnika

DELETE-AFTER(p)

$q = next(p)$

$next(p) = next(q)$

FREENODE(q)

Brisanje

- Ukoliko čvor nije na kraju liste i ako je lista homogena, sličnom dostkom kao kod umetanja ispred tekućeg čvora, može se rešiti problem brisanja čvora na koji pokazuje pokazivač *p*

$$\underline{\text{DELETE}(p)}$$
$$q = next(p)$$
$$next(p) = next(q)$$
$$info(p) = info(q)$$
$$\text{FREENODE}(q)$$

Pretraživanje

- Potreba lociranja čvora u listi koji ima zadati sadržaj radi pristupa
- Struktura liste uslovljava potpuno sekvencionalno pretraživanje počevši od glave liste i sledeći lanac pokazivača

Pretraživanje

- Operacija SEARCH(list, x) pretražuje neuređenu listu na koju ukazuje pokazivač *list* i ukoliko postoji bar jedan čvor sa zadatim sadržajem *x*, vraća adresu prvog takvog čvora, a ako ne postoji, vraća *nil*.

```
SEARCH(list, x)
p = list
while (p ≠ nil) and (info(p) ≠ x) do
    p = next(p)
end_while
return p
```

Pretraživanje

- Ako je lista uređena (npr. neopadajuće) neuspešno pretraživanje se može proglasiti čim se dođe do prvog čvora sa većim sadržajem, a do tada se ne pronađe zadati čvor

```
SEARCH-ORD(list, x)
p = list
while (p ≠ nil) and (info(p) < x) do
    p = next(p)
end_while
if (p ≠ nil) and (info(p) > x) then
    p = nil
end_if
return p
```

Ostale operacije

- Inverzija poretka čvorova u listi
- Povezivanje dve liste u jednu
- Nalaženje i-tog čvora
- Razbijanje jedne liste u dve liste
- Kopiranje
- Itd.

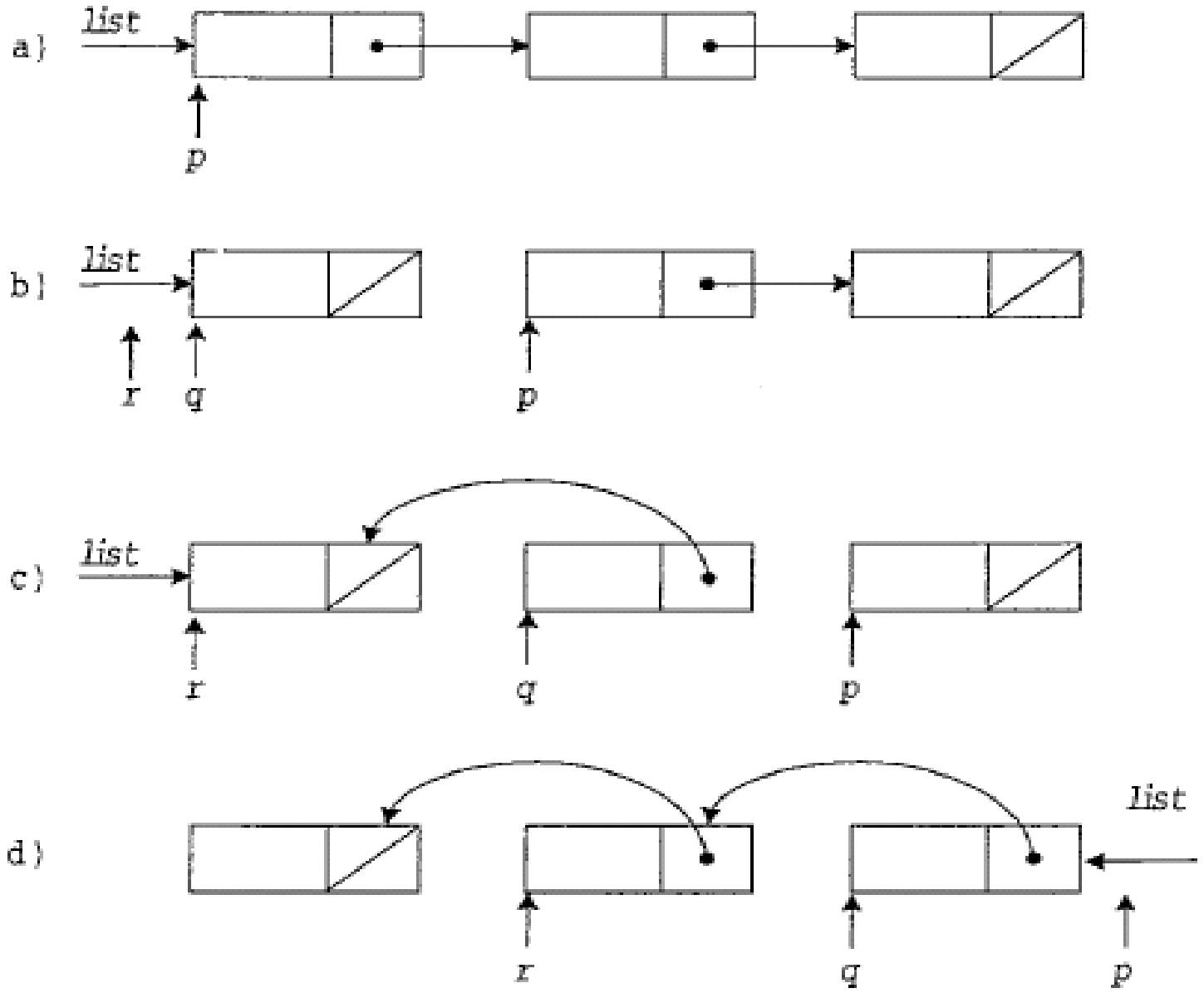
Ostale operacije

- INVERT(list) operacija obrće poredak čvorova u listi na čiji prvi čvor ukazije pokazivač *list* tako da prvi čvor postaje poslednji, drugi pretposlednji, itd.
- Ova operacija koristi tri pokazivača koji se sukcesivno pomeraju od početka do kraja liste tako da **p** pokazuje na sledbenika čvora ukazanog sa **q** a **r** na prethodnika čvora ukazanog sa **q**.

Ostale operacije

- U svakoj iteraciji se polje pokazivača čvora ukazanog sa **q** prebaci sa sledbenika **p** na prethodnika **r** i sva tri tekuća pokazivača se pomere za po jedan čvor.

```
INVERT(list)
p = list
q = nil
while (p ≠ nil) do
    r = q
    q = p
    p = next(p)
    next(q) = r
end_while
list = q
```



Ilustracija operacije inverzije poretka čvorova liste - izgled liste: a) pre početka operacije, b) posle prve, c)posle druge i d) posle treće iteracije

Ostale operacije (povezivanje 2 liste)

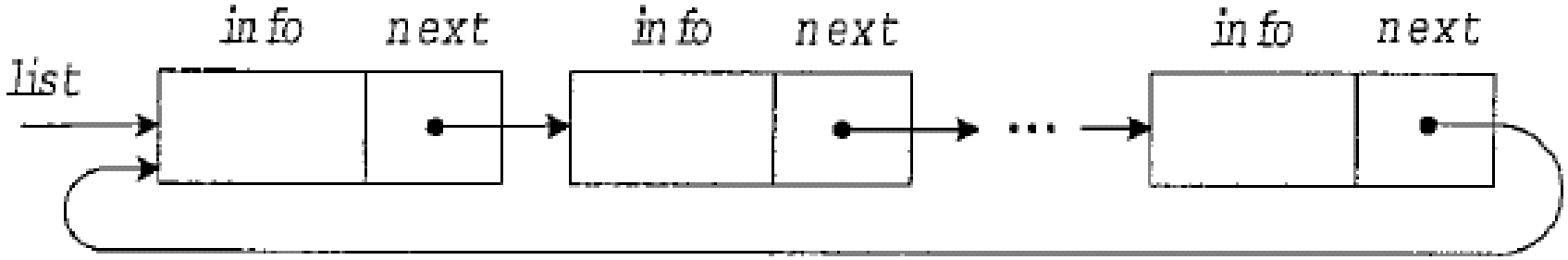
- Operacija CONCATENATE(list1, list2) nadovezije listu na koju pokazuje list2 na listu ukazanu sa list1.
- Prvo se ispitaju specijalni slučajevi kad je jedna ili druga lista prazna
- Ako to nije slučaj, onda se dođe do kraja prve liste pa se napravi da poslednji čvor prve liste pokazuje na prvi čvor druge liste

```
CONCATENATE(list1, list2)
if list1 = nil then
    list1 = list2
    return
else if list2 = nil then
    return
end_if
p = list1
while (next(p) ≠ nil) do
    p = next(p)
end_while
next(p) = list2
```

Operacije sa kružnim ulančanim listama

- Nedostatak jednostruke liste je mogućnost kretanja kroz listu samo u jednom smeru
- Od jednog čvora se ne može doći do bilo kojeg čvora u listi ispred njega
- Jedan način da se ovaj nedostatak otkloni a da se ne menja struktura čvora je da se lista pretvori u kružnu
- Umesto da poslednji čvor sadrži prazan pokazivač, on ukazuje na prvi čvor

Operacije sa kružnim ulančanim listama



Jednostruko ulančana kružna lista

- Analizom operacija se dolazi do zaključka da je kod kružne liste korisnije da *list* pokazuje na *poslednji čvor liste*
- Operacija umetanja iza čvora pokazanog pokazivačem p je slična kao i u nekružnoj listi

Operacije sa kružnim ulančanim listama

- Jedina razlika se javlja kada se ubacuje novi čvor iza poslednjeg čvora u listi na kojeg ukazuje spoljašnji pokazivač
- Tada novi čvor postaje poslednji, a spoljašnji pokazivač list treba da pokazuje na njega (list=q)

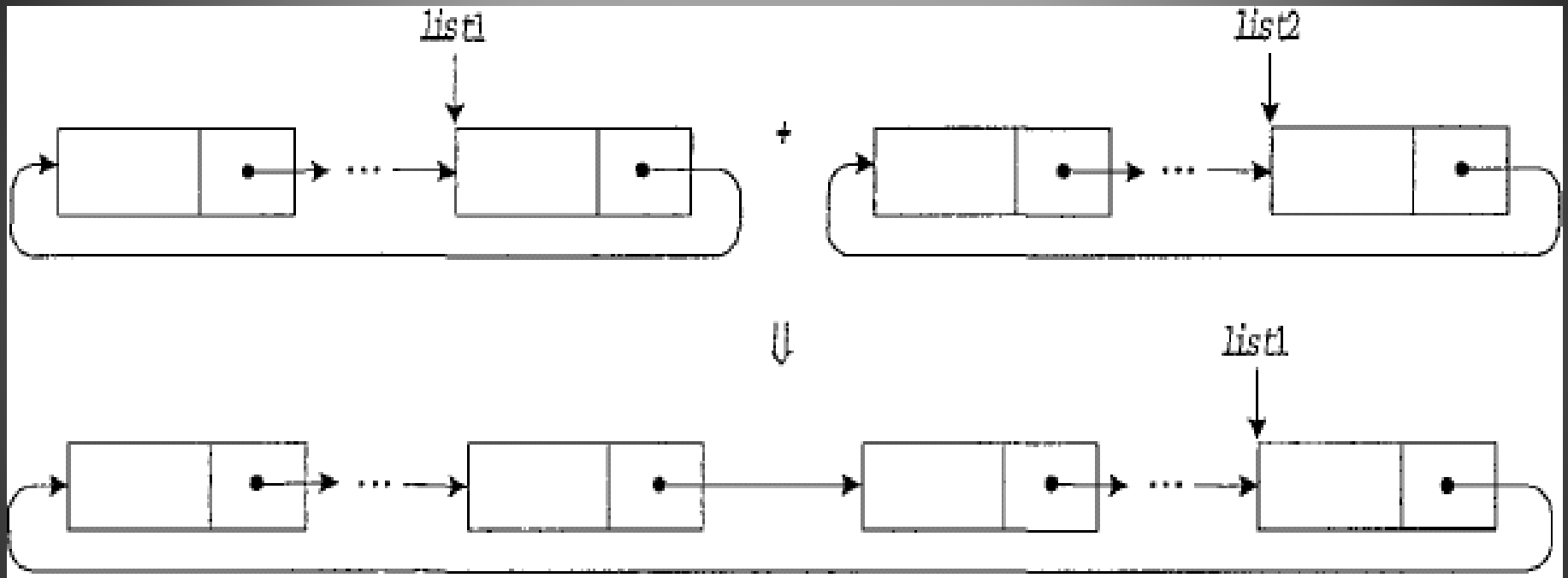
```
INSERT-AFTER-C(p, x)
q = GETNODE
info(q) = x
next(q) = next(p)
next(p) = q
if (list = p) then
    list = q
end_if
```

Operacije sa kružnim ulančanim listama

- Operacija spajanje dve liste CONCATENATE(*list1*, *list2*) sada, takođe ne zahteva prolazak do kraja prve liste pošto pokazivač *list1* ukazuje na poslednji čvor
- Na kraju pokazivač *list1* ukazuje na poslednji čvor objedinjene liste

```
CONCATENATE-C(list1, list2)
if (list1 = nil) then
    list1 = list2
    return
else if (list 2 = nil) then
    return
end_if
p = next(list1)
next(list1) = next(list2)
next(list2) = p
list1 = list2
```

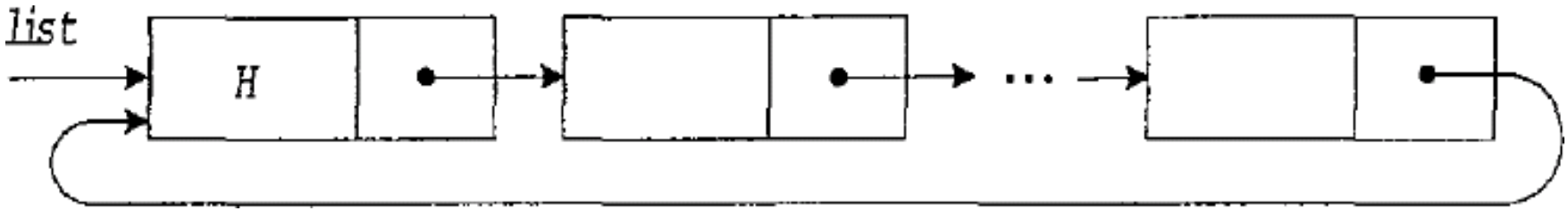
Spajanje dve kružne liste u jednu



Spajanje dve kružne liste u jednu

Kružna lista sa zaglavljem

- Kod pretraživanja kružne liste počevši od prvog čvora, pri neuspešnom pretražiavnju dolazi do beskonačne petlje jer nema načina da se detektuje kraj liste.
- Zbog toga se, obično, na početak liste ubacuje poseban čvor koji se naziva **zaglavlje liste**



Kružna lista sa zaglavljem

- 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
- Drugi način je postavljanje posebne oznake koja govori da je čvor zaglavlje
- U tom slučaju se u polje info mogu smestiti i neke informacije o samoj listi, kao što je broj elemenata, itd.

Kružna lista sa zaglavljem

- Spoljašnji pokazivač *list* sad ukazuje na zaglavlje, pa je ubacivanje novog čvora na početak liste u operaciji INSERT-CH(list, x) malo drugačije

INSERT-CH($list$, x)

p = GETNODE

$info(p) = x$

$next(p) = next(list)$

$next(list) = p$

- Kružna lista sa zaglavljem nikada ne može da bude prazna što pojednostavljuje određene operacije, jer nije potrebno detektovati poseban slučaj prazne liste.
- Kod liste bez ostalih čvorova je next(list)=list

Operacije sa dvostruko ulančanim listama

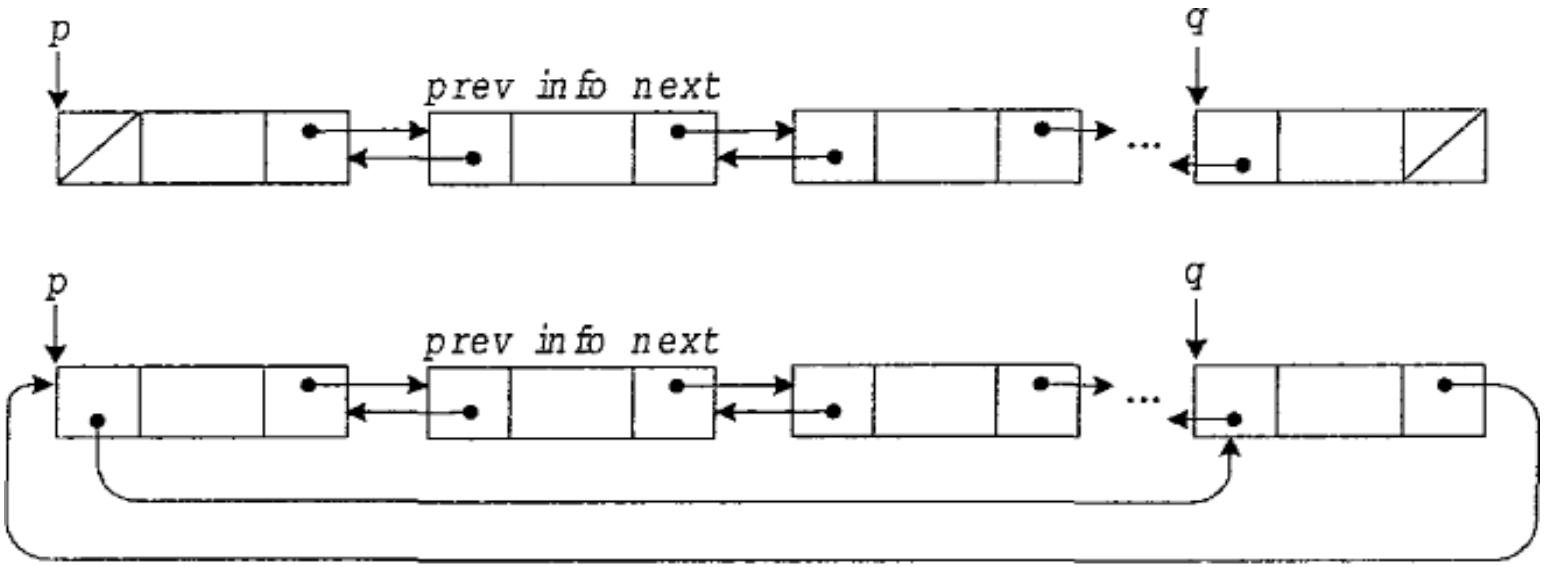
- Iako se u kružnoj listi od svakog čvora može doći do svakog drugog, prelazak može biti vrlo neefikasan
- Npr, ako treba od jednog čvora doći do njegovog prethodnika, mora se proći čitava lista
- Zato je korisno da svaki čvor, pored pokazivača na sledeći čvor sadrži i pokazivač na prethodni čvor.
- Ovakva lista je **dvostruko ulančana**

Operacije sa dvostruko ulančanim listama

- Kod dvostruko ulančane liste je jednako efikasno kretati se po listi u oba smera, što pokazuje sledeća jednakost (p pokazivač na neki čvor) :
  - ***next(prev(p))=p=prev(next(p))***
- Ovakva lista, takođe, može da bude nekružna ili kružna, sa zaglavljem ili bez njega

Operacije sa dvostruko ulančanim listama

- Ako p ukazuje na prvi čvor a q na poslednji:
  - u nekružnoj listi je tada *prev(p)=next(q)=nil*
  - a u kružnoj *prev(p)=q i next(q)=p*



Dvostruko ulančana lista: a) nekružna i b) kružna

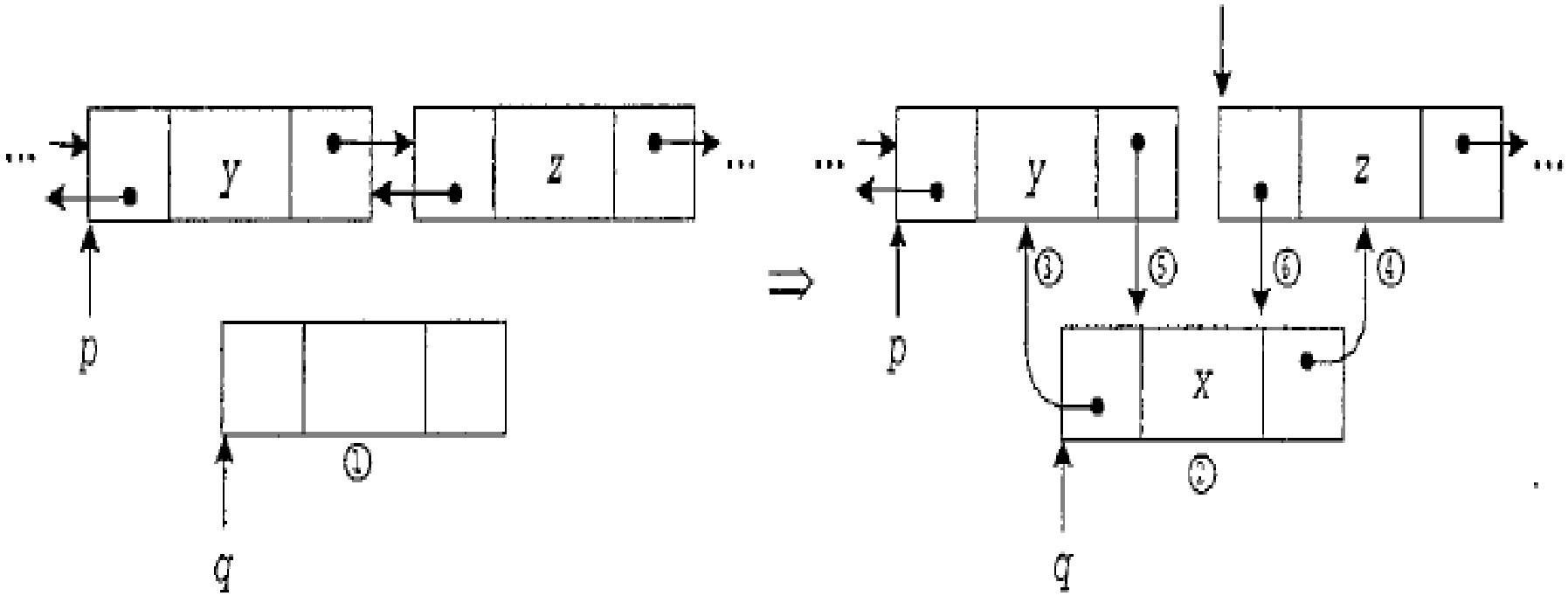
Operacije sa dvostruko ulančanim listama

- Ako kružna lista ima i zaglavlje, onda je
  prev(list)=q
  next(list)=p
  prev(p)=next(q)=list
- Kod kružne liste sa zaglavljem koja nema drugih čvorova oba pokazivača u zaglavlju ukazuju na samo zaglavlje
  - prev(list)=list=next(list)
- Sa dvostrukom listom su moguće sve ranije razmoterene operacije
- One su malo složenije jer treba ažurirati oba pokazivača

Operacije sa dvostruko ulančanim listama

- Neka je dvostruko ulančana lista kružna i neka ima zaglavlje.
- Tada se operacija umetanja novog čvora sa informacionim sadržajem x iza čvora ukazanog pokazivačem p, INSERT-AFTER- D(P, X), realizuje

INSERT-AFTER-D(p, x)
q = GETNODE
$info(q) = x$
$r = next(p)$
$prev(q) = p$
$next(q) = r$
$next(p) = q$
$prev(r) = q$

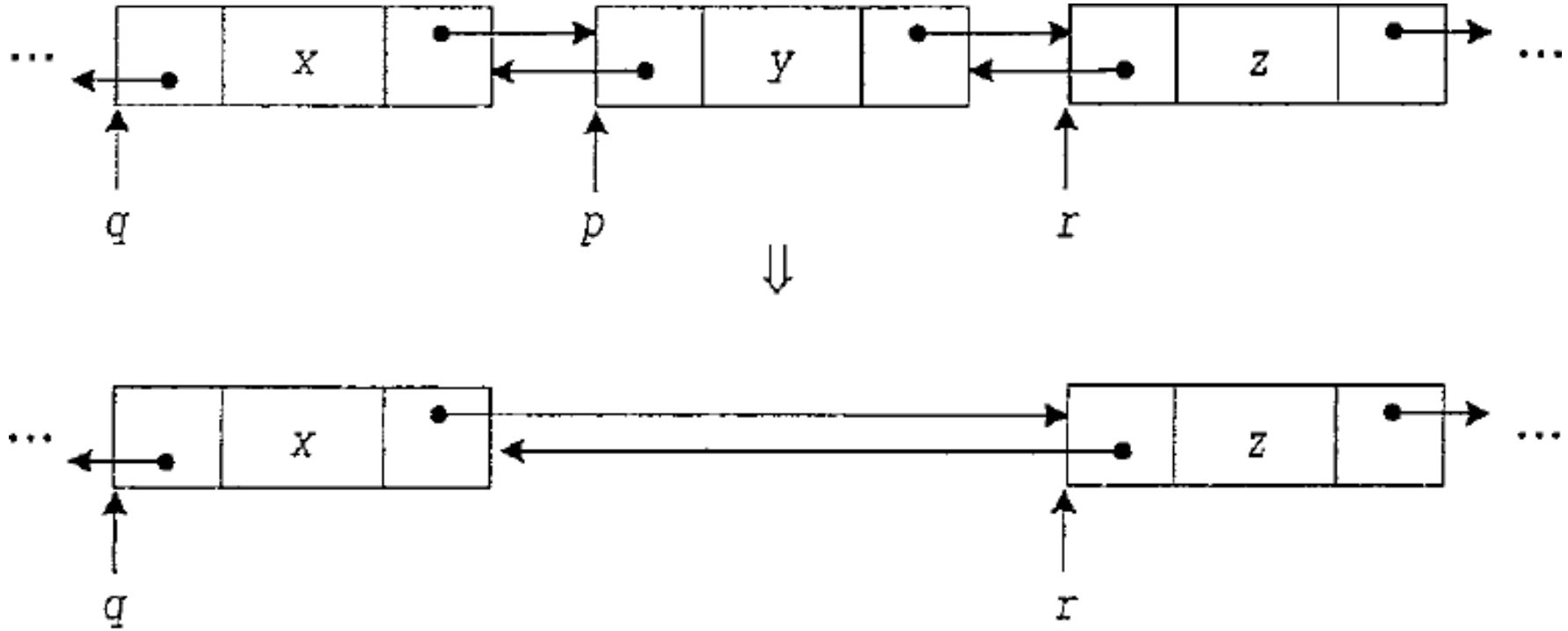


Umetanje čvora u dvostruko ulančanu listu

Operacije sa dvostruko ulančanim listama Brisanje

- Problem brisanja tekućeg čvora se efikasno rešava jer se direktno pristupa i prethodnom i sledećem čvoru pri povezivanju liste
- Pretpostavke o postojanju zaglavlja i kružnosti su bitne zbog pojednostavljenja operacije- ne mora se voditi računa o specijalnim slučajevima brisanja sa početka i kraja liste

DELETE-D(p)
$q = prev(p)$
$r = next(p)$
$next(q) = r$
$prev(r) = q$
FREENODE(p)



Brisanje čvora iz dvostruko ulančane liste