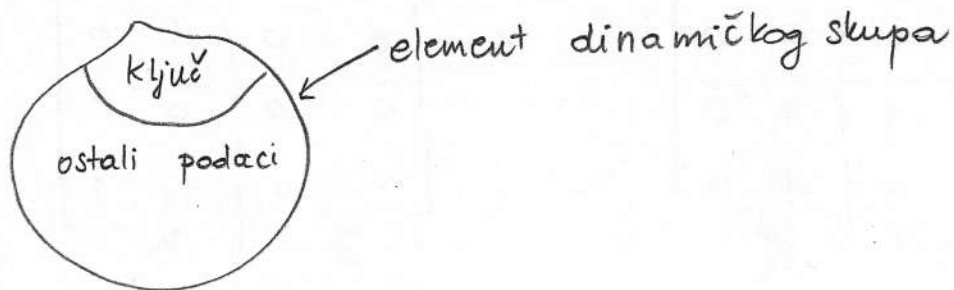


STRUKTURE PODATAKA

dinamički skupovi - skupovi koji se tijekom izvođenja algoritma mijenjaju



ključ - uglavnom se bira iz nekog potpuno uređenog skupa
(primjer: polja)

operacije:

- upiti
- promjene

pitati:

* SEARCH (S, k)

- (dinamički) skup S , ključ k
- vraća pokazivač na element u u S t.d. je $ključ[u] = k$
- ukoliko takav element ne postoji, vraća NIL

* MINIMUM (S)

- skup S
- vraća pokazivač na element skupa S koji ima najmanju vrijednost ključa k s obzirom na potpuni uređaj

* MAXIMUM (S)

- skup S
- vraća pokazivač na element skupa S koji ima najveću vrijednost ključa k

* SUCCESSOR (S, x)

- skup S , element x

- vraća pokazivač na prvi veći element od x u S s obzirom na p.u., ukoliko takav element postoji,
- inače, vraća NULL

* PREDECESSOR (S, x)

- skup S , element x
 - vraća pokazivač na prvi manji element od x u S s obzirom na p.u., ukoliko takav element postoji,
 - inače, vraća NULL
- kako ocijeniti vremensku složenost operacija?

Kao i do sada, tj. kao funkciju veličine dinamičkog skupa

Primjerice: $O(\lg n)$

POVEŽANE LISTE

- struktura podataka
 - linearna
 - samoreferentna
- linearna: elementi su organizirani u linearnom poretku
- samoreferentna: svaki element liste zna gdje je njegov susjed unutar te liste.

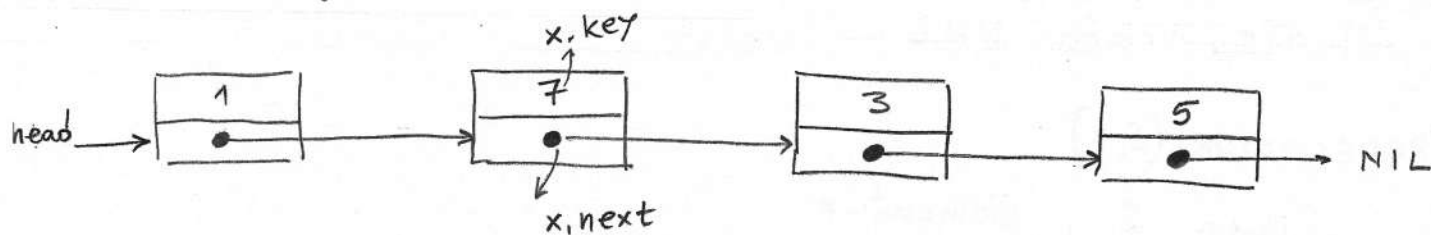
lijevo, desno
↗
potpuni uređaj

Svako ih možemo promatrati kao dinamičke skupove

- razlikujemo:
- * jednostruko - povezane liste
 - * dvostruko - povezane liste

POVEZANE LISTE (eng. LINKED LISTS)

prisjetimo se jednostruko-povezanih listi:



karakteristike:

- svaki element ima ključ $x.key$
- svaki element ima pokazivač na sljedeći (posljednji ima NIL)
- pokazivač na početak liste, tj: na prvi element: $L.head$

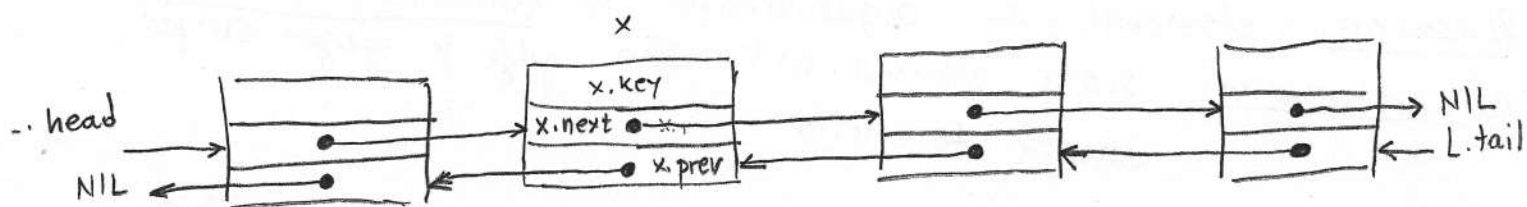
Zaključak:

- elementi formiraju niz
- elementi pokazuju na druge elemente (u ovom slučaju na svog sljedbenika)

Jednostruko povezana lista je linearna samoreferentna struktura podataka

povezane liste promatramo kao dinamičke skupove

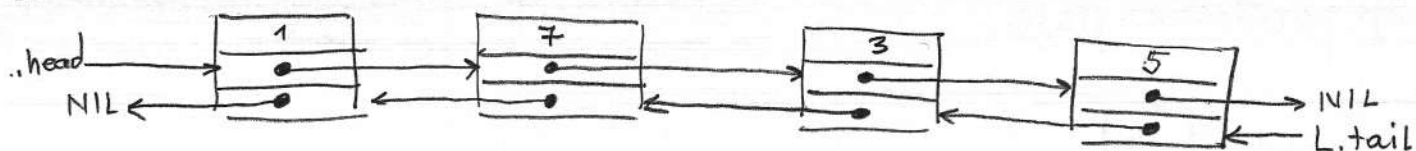
DVOSTRUKO-POVEZANE LISTE



znake:

- * $x.key$ - ključ elementa x
- * $x.prev$ - pokazivač na element koji prethodi elementu x
- * $x.next$ - pokazivač na element koji slijedi nakon elementa x

Primjer:



Pitanje: Koja je razlika između jednostruko i dvostruko povezane liste?

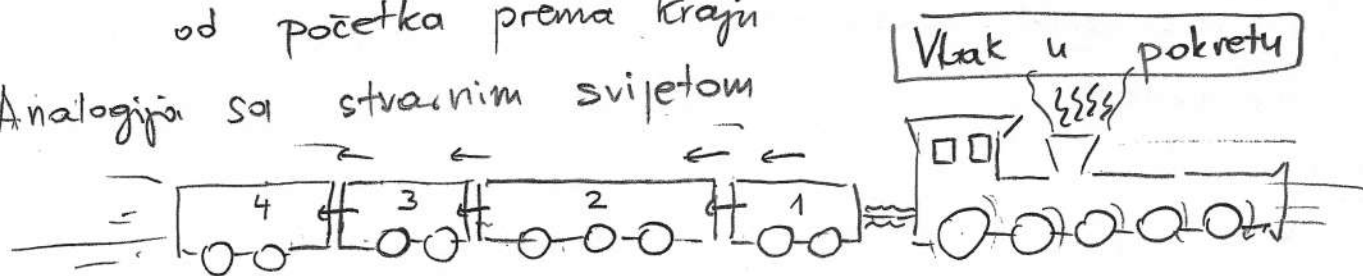
* dvostruku listu možemo obilaziti dvosmjerno:

od početka prema kraju; od kraja prema početku

* jednostruku listu možemo obilaziti jednosmjerno:

od početka prema kraju

Analogija sa stvarnim svijetom



- ne mogu preskakati vagona

- jednostruko povezana lista:

* vrata na svakom vagonu su jednosmjerna
(u smjeru od lokomotive do zadnjeg vagona)

* ako želim doći iz 1. vagona u 4., moram ići redom

* ne mogu se vratiti (vrata su jednosmjerna)

dvostruko povezana lista:

* vrata na svakom vagonu su dvosmjerna

* mogu se vraćati ali ne mogu preskakati vagona

* realna situacija

Vlak u kolodvoru

- ukoliko vlak stoji u kolodvoru, u bilo koji vagon možemo otići direktno → polje

obilazak vlaka u pokretu odgovara pretraživanju
dvostruko-povezane liste.

LIST-SEARCH (L, k)

L - lista koju pretražujemo

k - ključ

▷ tražimo pokazivača x u listi L takav da je $x.key = k$
ukoliko takav x ne postoji, vraćamo NIL .

$x = L.head$

while $x \neq NIL$ and $x.key \neq k$
 $x = x.next$

end while

return x

analogija: 'Nalazimo se u vlaku (koji se giba) i želimo
ronaci vagon br. k .

UBACIVANJE ELEMENATA U LISTU

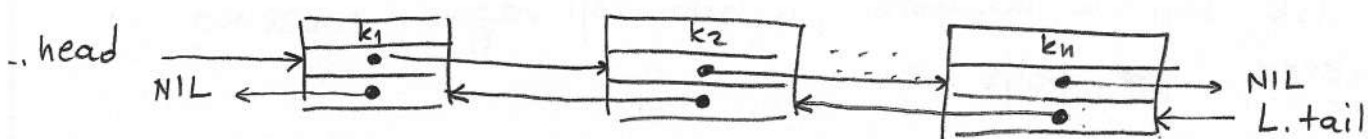
razmotrit ćemo 3 slučaja:

* ubacivanju elementa na početak (dvostruko-)
povezane liste

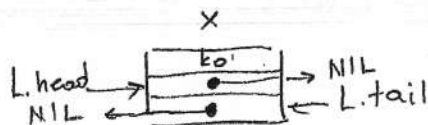
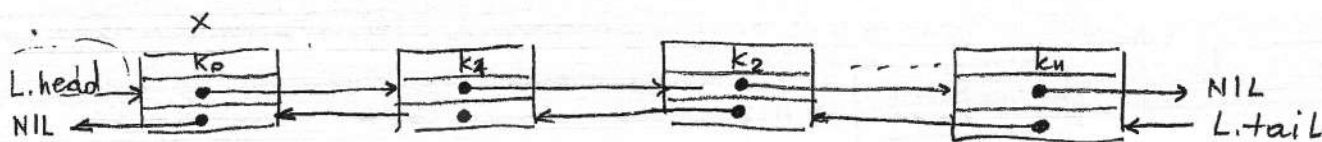
* ubacivanje elementa na kraj (dvostruko-)
povezane liste

* ubacivanje elementa nakon nekog zadanog
elementa

1) Ubacivanje elementa na početak



ubaciti element x s ključem k_0 na početak liste L



L.head = NIL
L.tail = NIL

LIST-INSERT-FRONT (L, x)

```

x.next = L.head
if IS-EMPTY (L)
    L.tail = x
else
    L.head.prev = x
end if
L.head = x
x.prev = NIL
    
```

IS-EMPTY (L)

return L.head = NIL
and
L.tail = NIL

Primer: Lista L je prazna. Dodajte dva elementa liste s kyuievima 5 i 6 na pocetak liste.

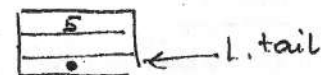
1. L.head = NIL
L.tail = NIL

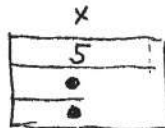
3. Poziv funkcije LIST-INSERT-FRONT (L, x)

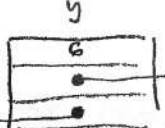
x.next = NIL
IS-EMPTY (L) = TRUE

L.tail = x

L.head = x
x.prev = NIL



2. 
Stvorim element x

4. 
Stvorim element y

5. LIST-INSERT-FRONT (L, y)

y.next = x

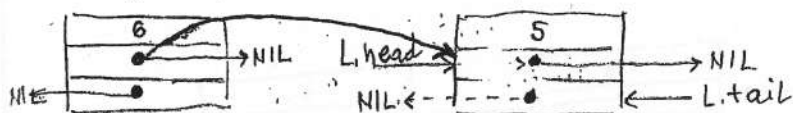
IS-EMPTY (L) = FALSE

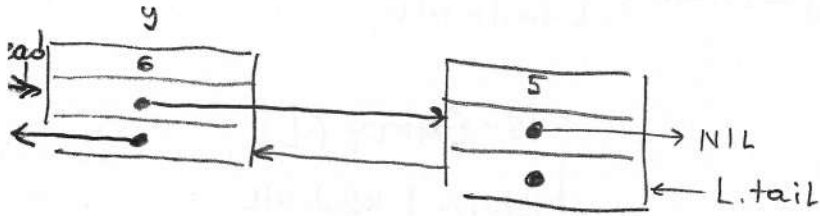
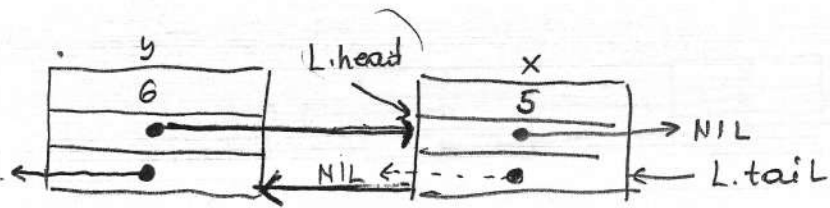
L.head.prev = y

L.head = y

y.prev = NIL

Ilustracija poziva LIST-INSERT-FRONT (L, y)





ubaciti element x s ključem k na kraj liste L

LIST-INSERT-BACK (L, x)

$x.\text{prev} = L.\text{tail}$

if IS-EMPTY (L)

| $L.\text{head} = x$

else

| $L.\text{tail}.\text{next} = x$

end if

$L.\text{tail} = x$

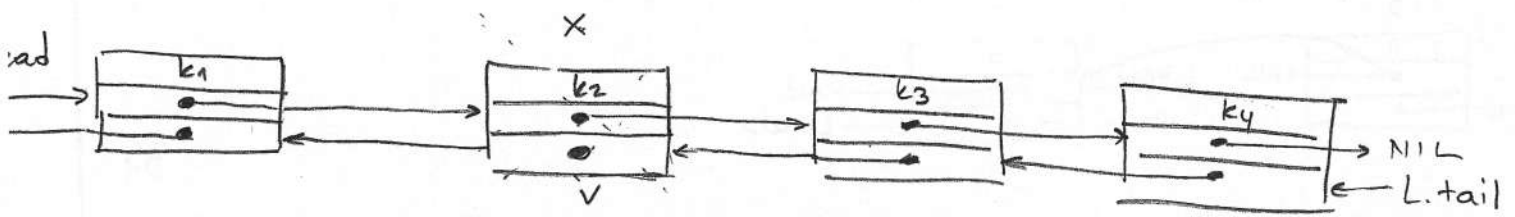
$x.\text{next} = \text{NIL}$

BRISANJE ELEMENTA IZ LISTE

razlikujemo dva slučaja:

- 1.) brisanje elementa x (tj. dan je pokazivač na x)
- 2.) brisanje elementa s ključem k

Brisanje elementa x



LIST-DELETE (L, x)

```
if  $x.\text{prev} \neq \text{NIL}$ 
|
| then  $x.\text{prev}.\text{next} \leftarrow x.\text{next}$ 
|
| else
|
|  $L.\text{head} \leftarrow x.\text{next}$ 
|
end if
```

```
if  $x.\text{next} \neq \text{NIL}$ 
|
| then  $x.\text{next}.\text{prev} \leftarrow x.\text{prev}$ 
|
| else
|
|  $L.\text{tail} \leftarrow x.\text{prev}$ 
```

2.) brisanje elementa s ključem k

LIST-DELETE (L, k)

$x \leftarrow \text{LIST-SEARCH}(L, k)$

```
if  $x \neq \text{NIL}$ 
|
| then LIST-DELETE( $L, x$ )
|
end if
```