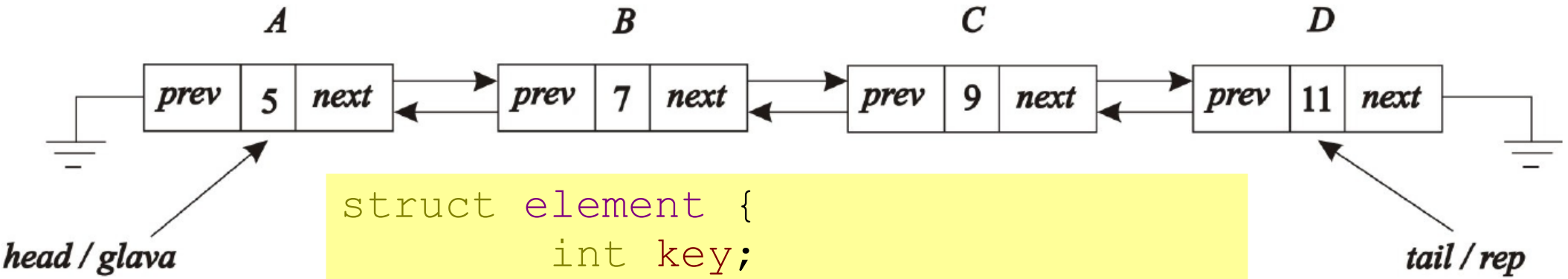
Dvojno povezan seznam

- Sklad oz. vrsta zahteva vnaprej določeno velikost polja
- Seznam nam omogoča dinamično velikost (toliko, kolikor rabimo)
- Primeren, če nimamo ocene o okvirni količini podatkov
  - Večino časa potrebujemo prostora za 10 podatkov, včasih pa za 10^6
  - Pri statičnih podatkovnih strukturah je večina prostora neizkoriščenega
- Primeren, če nove podatke vstavljamo na poljubno mesto

Dvojno povezan seznam

- Vsak element vsebuje:
  - vrednost (key)
  - kazalca na prejšnji in naslednji element
- Elementi so med seboj dvosmerno povezani
- Hitrejše iskanje v obe smeri
- *Obstaja tudi enojno povezan seznam (brez kazalca na prejšnji element)*
- Element predstavimo s strukturo (C/C++)



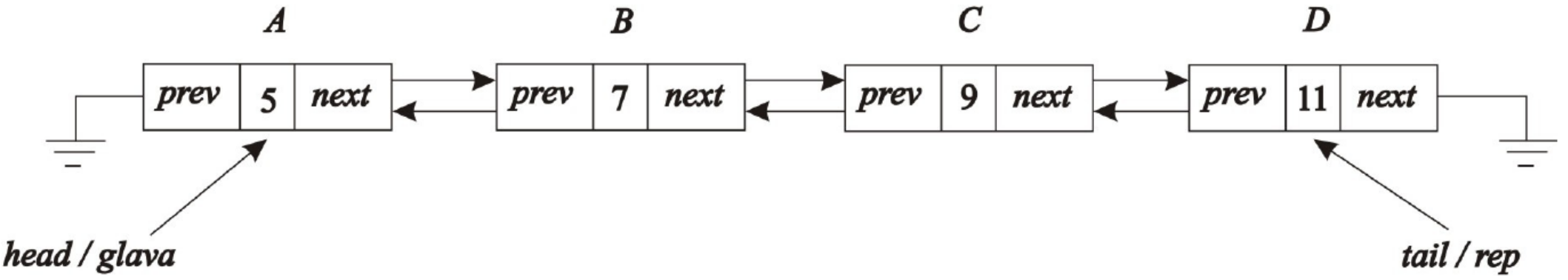

```
struct element {
        int key;
        element *prev, *next;
};
```

```
struct element {
   int key;
   element *prev, *next;
};
```

```
element *A=new element();
A->key=5;
A->prev=NULL;

element *B=new element();
B->key=7;
A->next=B;
B->prev=A;

element *C=new element();
C->key=9;
B->next=C;
C->prev=B;
```

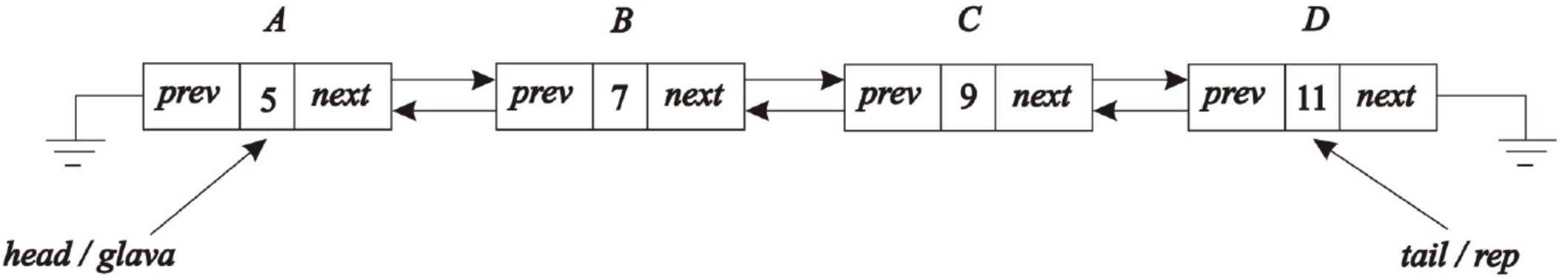
```
element *D=new element();
D->key=11;
C->next=D;
D->prev=C;
D->next=NULL;

element *glava=A;
element *rep=D;

delete A;
delete B;
delete C;
delete D;
```

Preveri kodo v razhroščevalniku!



Dvojno povezan seznam

- Prvi element nima predhodnika: glava (prev=NULL)
- Zadnji element nima naslednika: rep (next=NULL)
- Seznam je prazen, če sta glava in rep NULL
- Ponavadi sta glava in rep vstopna točko v seznam



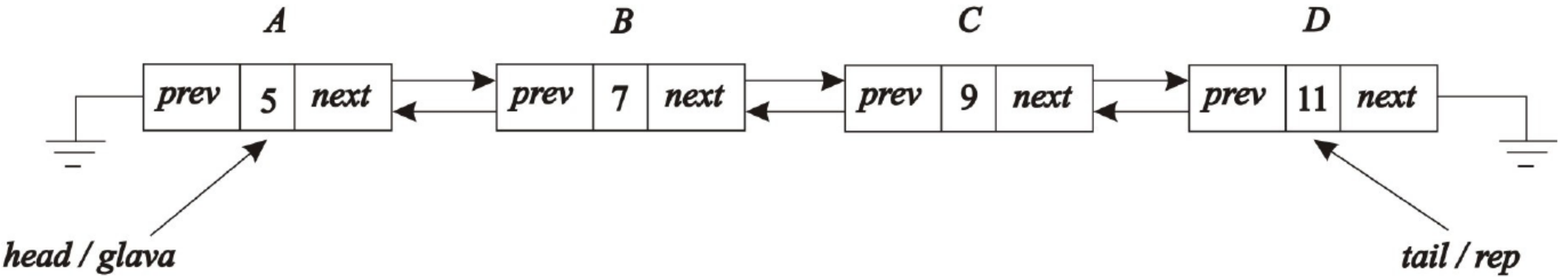
Operacije

- Iskanje
- Vstavljanje
  - v glavo
  - za določenim elementom
- Brisanje
- Izpis
  - od glave proti repu
  - od repa proti glavi

Iskanje

- Pomikanje od glave proti repu
  - Uporaba začasnega kazalca current

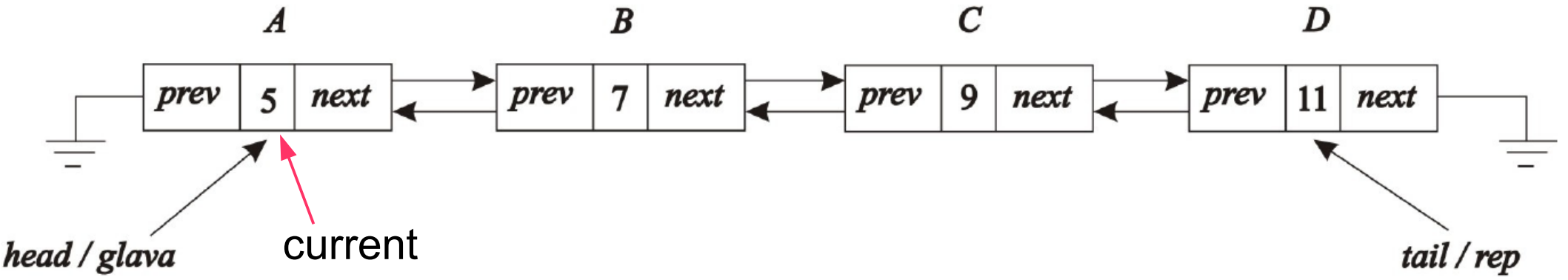
```
function NAJDI(head, key)
 begin
   current := head;
   while current<>NIL and current.key<>key do
      current := current.next;
   end
   return current;
 end
```



Iskanje

- Pomikanje od glave proti repu
  - Uporaba začasnega kazalca current

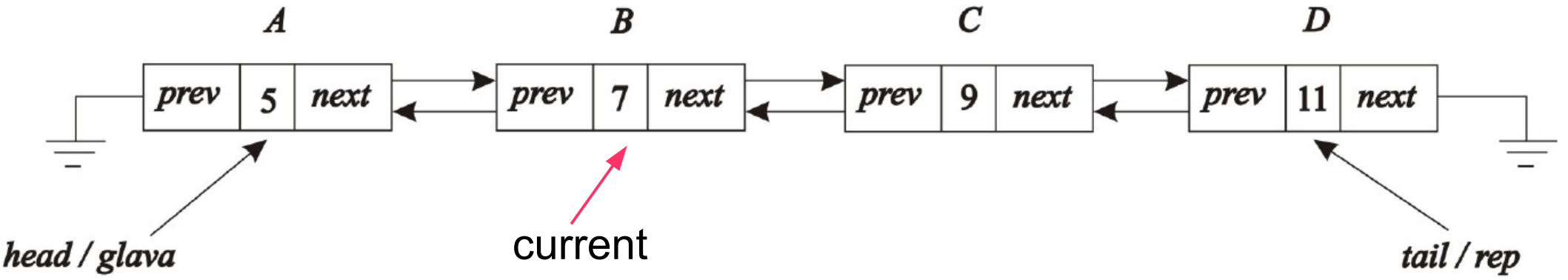
```
function NAJDI(head, key)
 begin
   current := head;
   while current<>NIL and current.key<>key do
      current := current.next;
   end
   return current;
 end
```



Iskanje

- Pomikanje od glave proti repu
  - Uporaba začasnega kazalca current

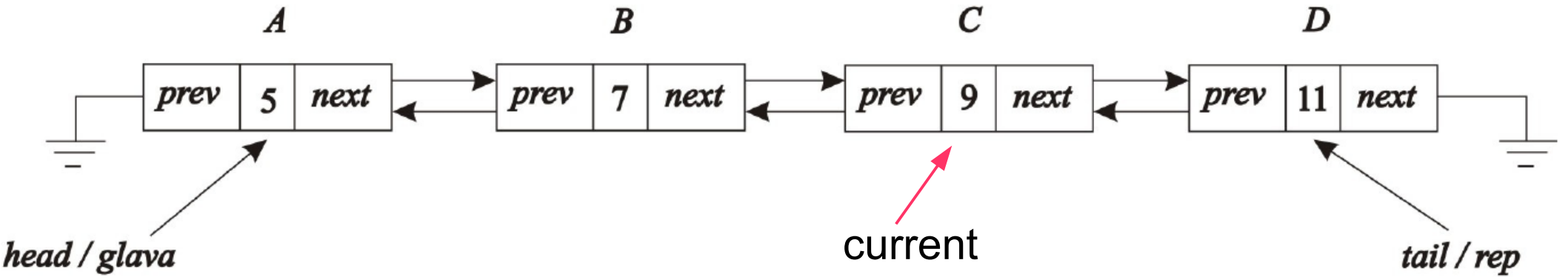
```
function NAJDI(head, key)
 begin
   current := head;
   while current<>NIL and current.key<>key do
      current := current.next;
   end
   return current;
 end
```



Iskanje

- Pomikanje od glave proti repu
  - Uporaba začasnega kazalca current

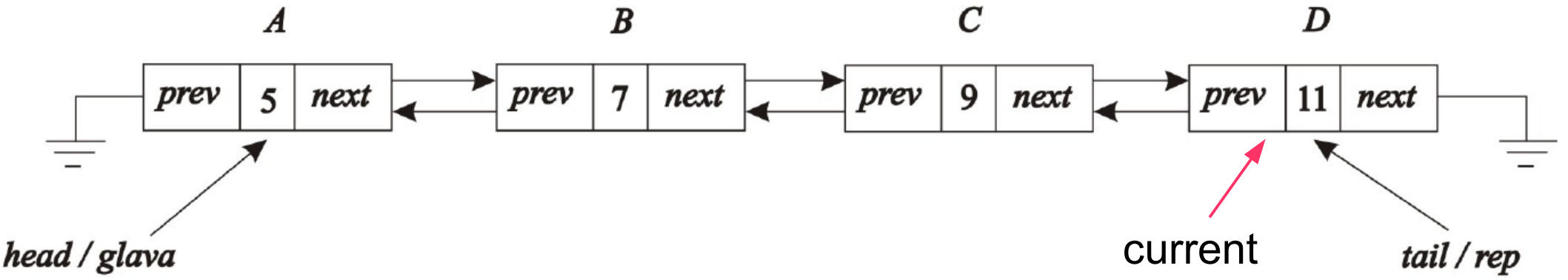
```
function NAJDI(head, key)
 begin
   current := head;
   while current<>NIL and current.key<>key do
      current := current.next;
   end
   return current;
 end
```



Iskanje

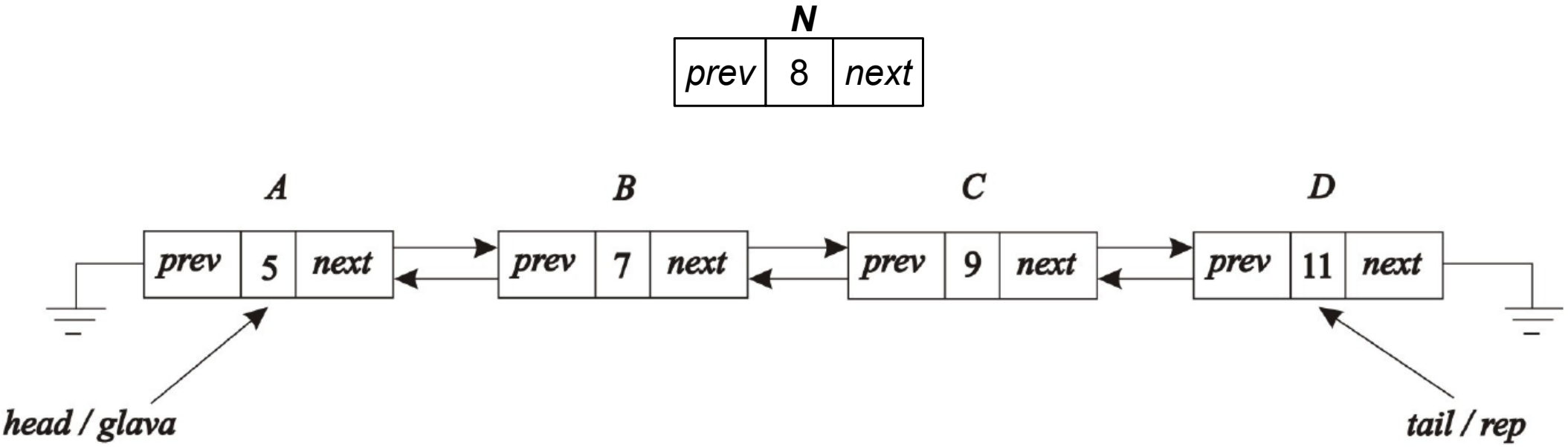
- Pomikanje od glave proti repu
  - Uporaba začasnega kazalca current

```
function NAJDI(head, key)
 begin
   current := head;
   while current<>NIL and current.key<>key do
      current := current.next;
   end
   return current;
 end
```



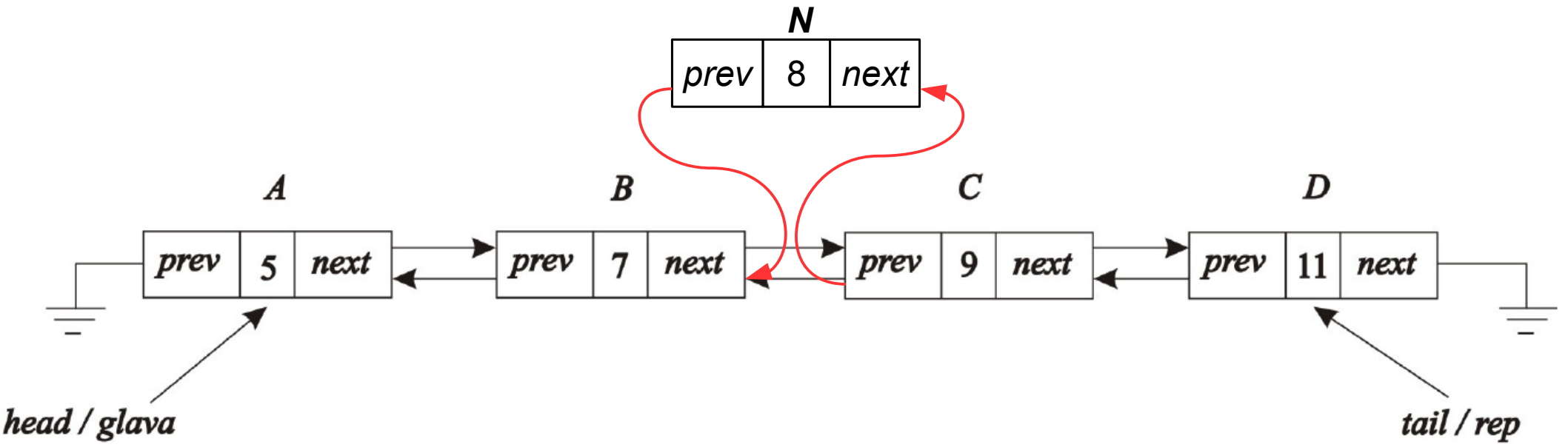
Vstavljanje za elementom

- Želimo vstaviti element ***N*** za elementom ***B***



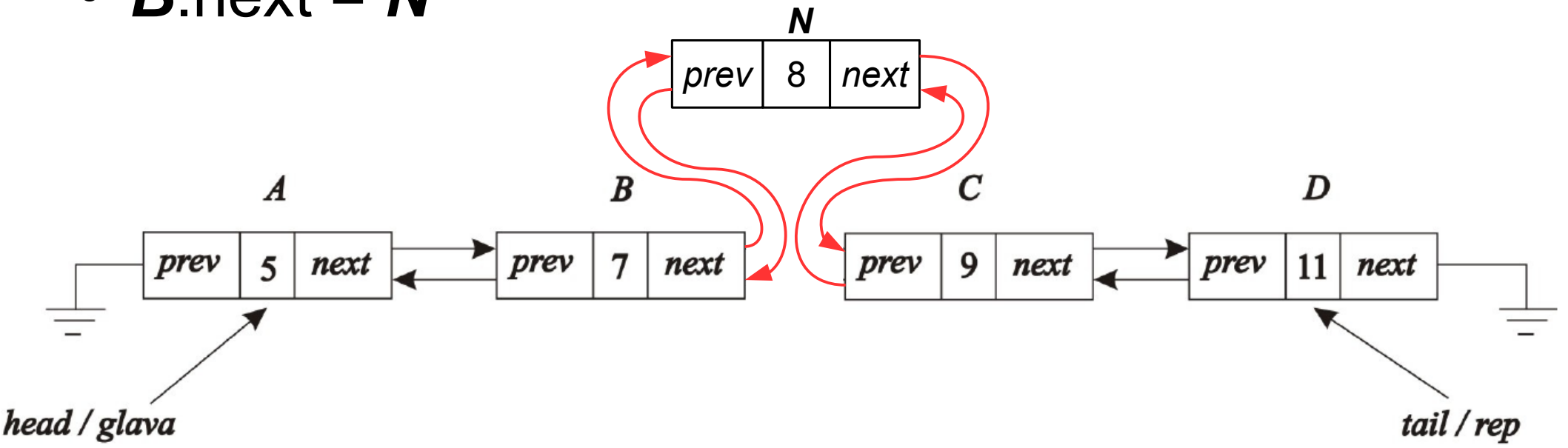
Vstavljanje za elementom

- Želimo vstaviti element ***N*** za elementom ***B***
- Povežemo kazalce:
- ***N***.prev na ***B***
- ***N***.next = ***C*** (***B***.next)



Vstavljanje za elementom

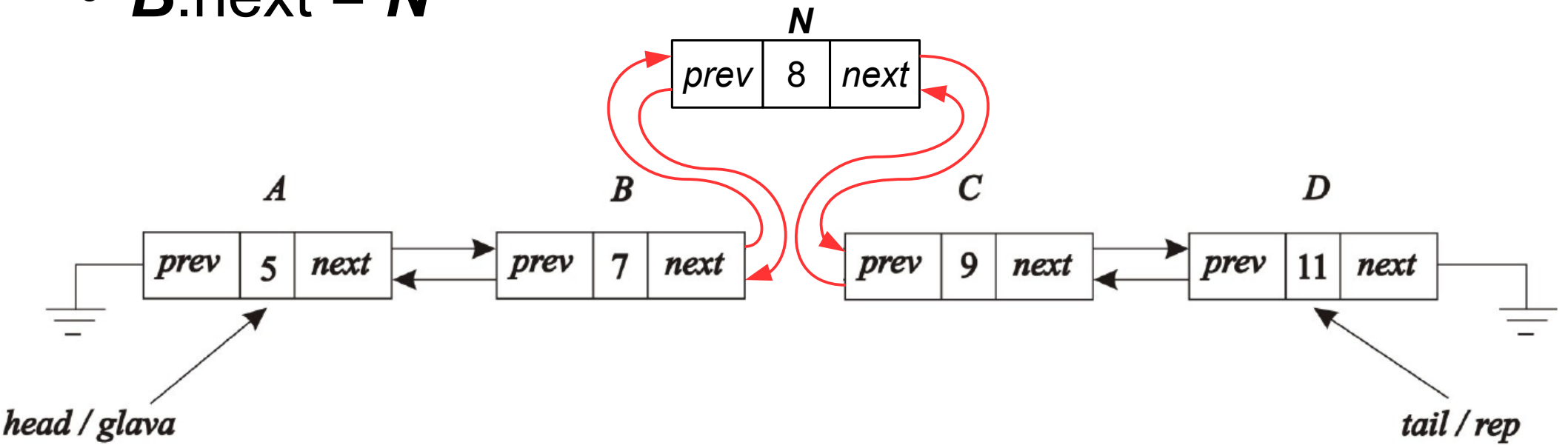
- Želimo vstaviti element ***N*** za elementom ***B***
- Povežemo kazalce:
- ***N***.prev na ***B***
- ***N***.next = ***C*** (***B***.next)
- ***C***.prev (**B**.next.prev) = ***N***
- ***B***.next = ***N***



Vstavljanje za elementom

- Želimo vstaviti element ***N*** za elementom ***B***
- Povežemo kazalce:
- ***N***.prev na ***B***
- ***N***.next = ***C*** (***B***.next)
- ***C***.prev (**B**.next.prev) = ***N***
- ***B***.next = ***N***

Kaj če vstavljamo za ***D?***
rep = ***N***

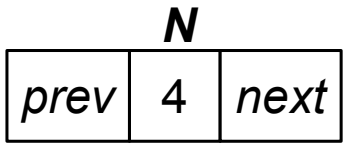


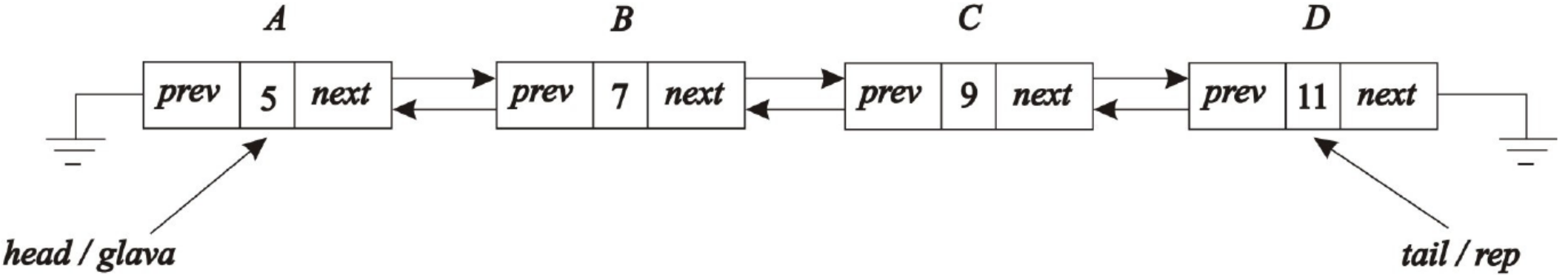
Vstavljanje za elementom

```
procedure VSTAVI_ZA(elem, new_el)
 begin
    new_el.prev := elem;
    new_el.next := elem.next;
    elem.next := new_el;
    if new_el.next <> NIL then
       new_el.next.prev := new_el;
    else
       tail := new_el;
 end
```

Vstavljanje v glavo

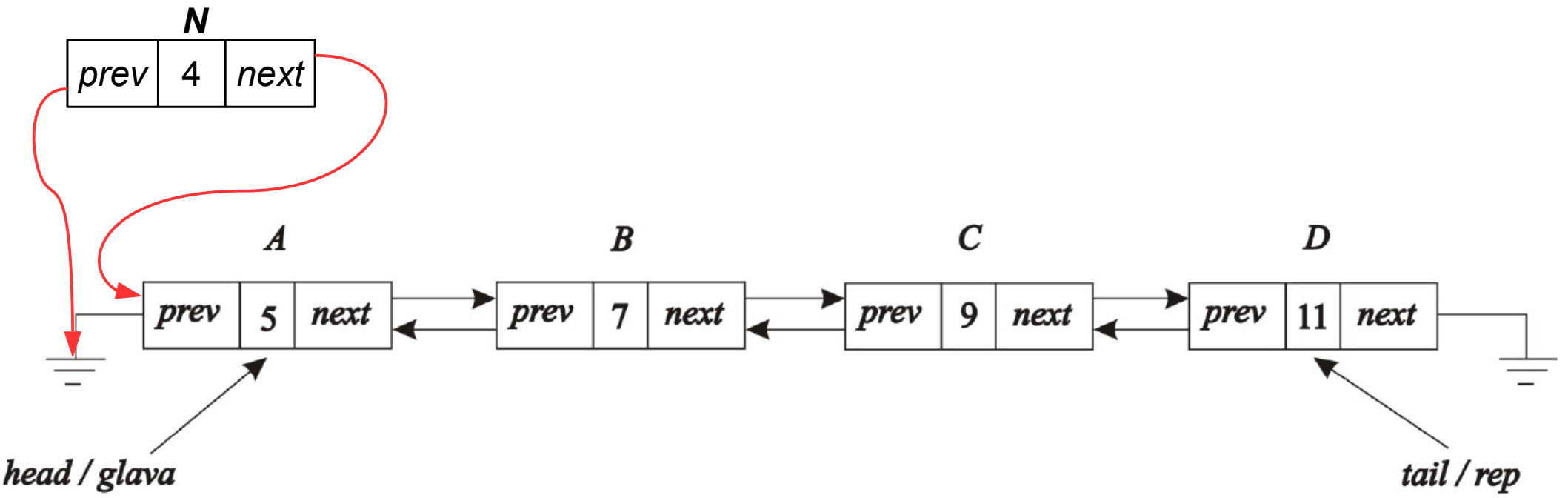
- Želimo vstaviti element ***N*** v glavo (pred elementom ***A***)





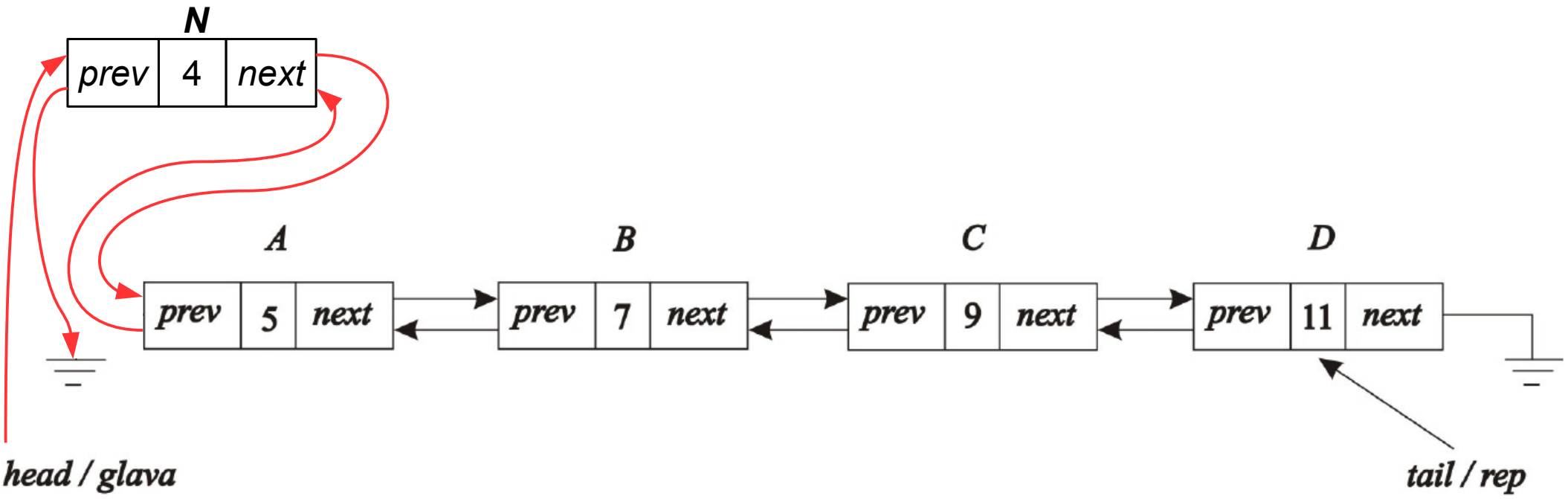
Vstavljanje v glavo

- Želimo vstaviti element ***N*** v glavo (pred elementom ***A***)
- Povežemo kazalce:
  - ***N***.next = ***A***, ***N***.prev = NIL



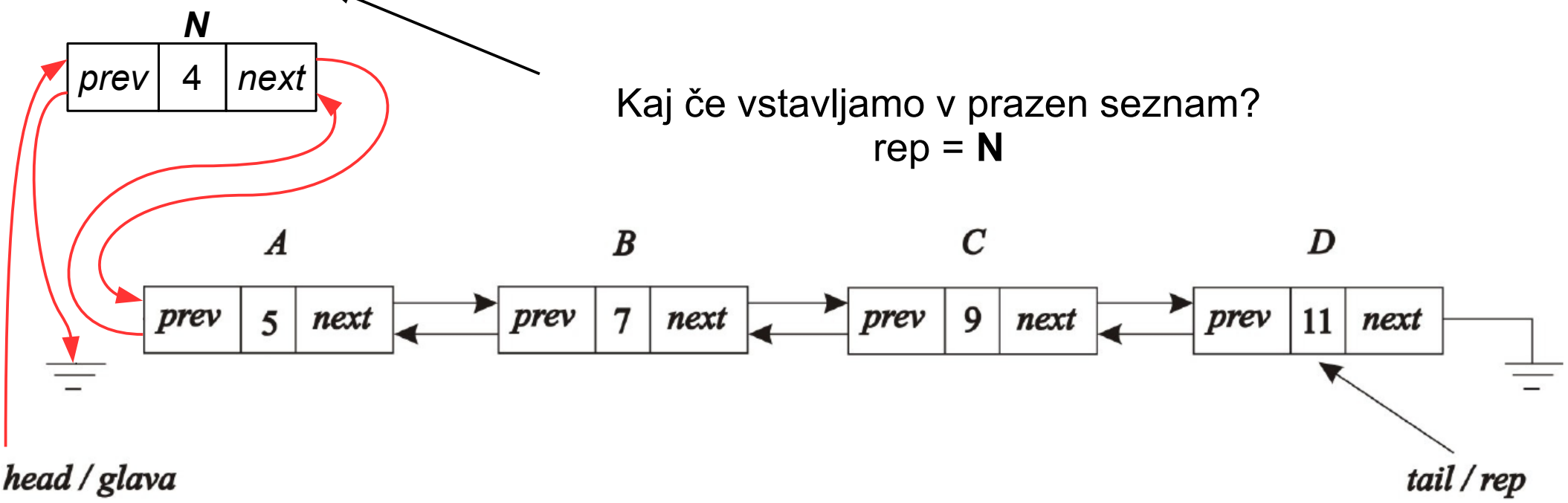
Vstavljanje v glavo

- Želimo vstaviti element ***N*** v glavo (pred elementom ***A***)
- Povežemo kazalce:
  - ***N***.next = ***A*** (glava), ***N***.prev = NIL
  - ***A*** (glava).prev = ***N***, glava = ***N***



Vstavljanje v glavo

- Želimo vstaviti element ***N*** v glavo (pred elementom ***A***)
- Povežemo kazalce:
  - ***N***.next = ***A*** (glava), ***N***.prev = NIL
  - ***A*** (glava).prev = ***N,*** glava = ***N***

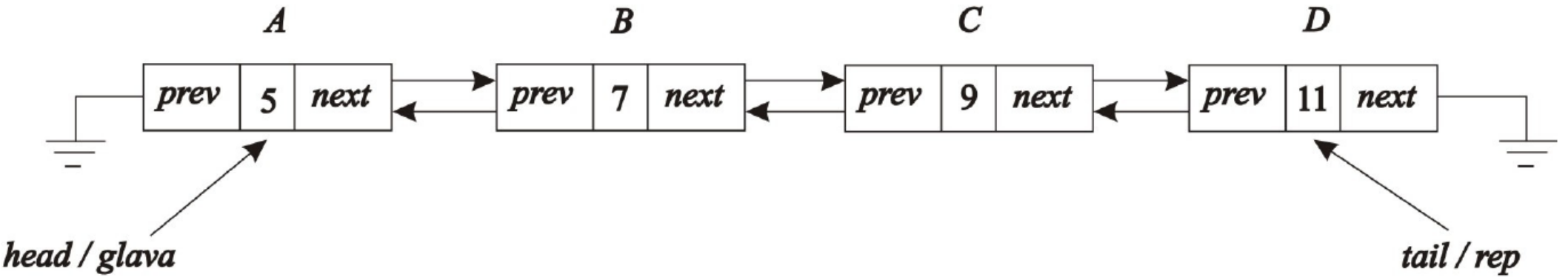


Vstavljanje v glavo

```
procedure VSTAVI_V_GLAVO(head, new_el)
 begin
    new_el.next := head;
    new_el.prev := NIL;
    if head<>NIL then
          head.prev := new_el;
    else
          tail := new_el;

    head := new_el
 end
```

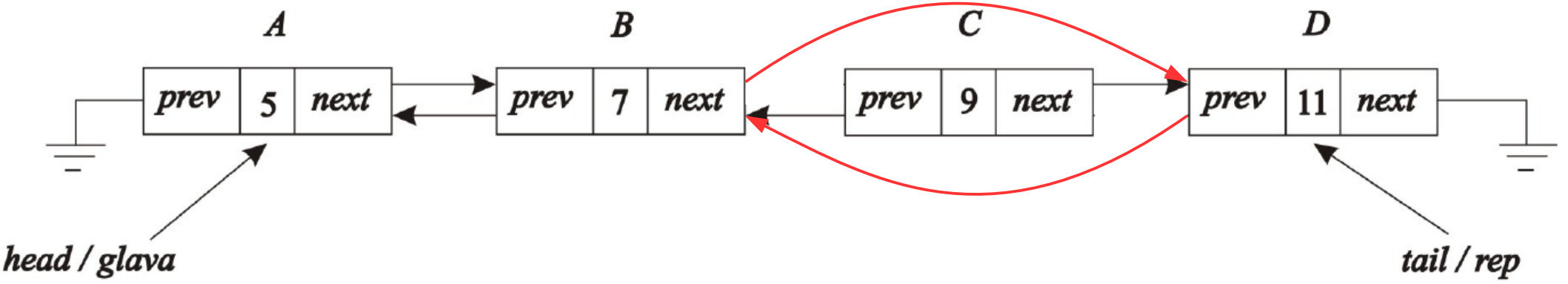
Brisanje elementa

- Želimo izbrisati ***C***



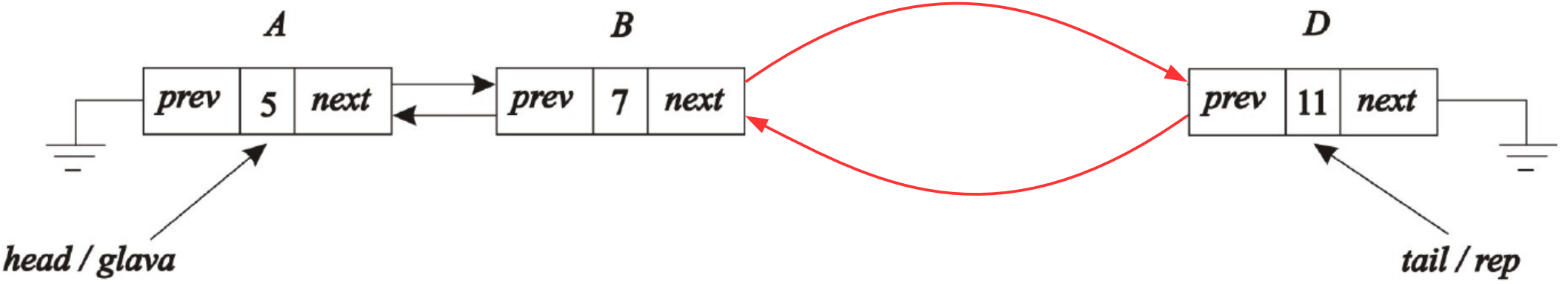
Brisanje elementa

- Želimo izbrisati ***C***
- Prevežemo kazalce mimo ***C***
  - ***B***.next (***C***.prev.next) = ***D***
  - ***D***.prev (***C***.next.prev) = ***B***



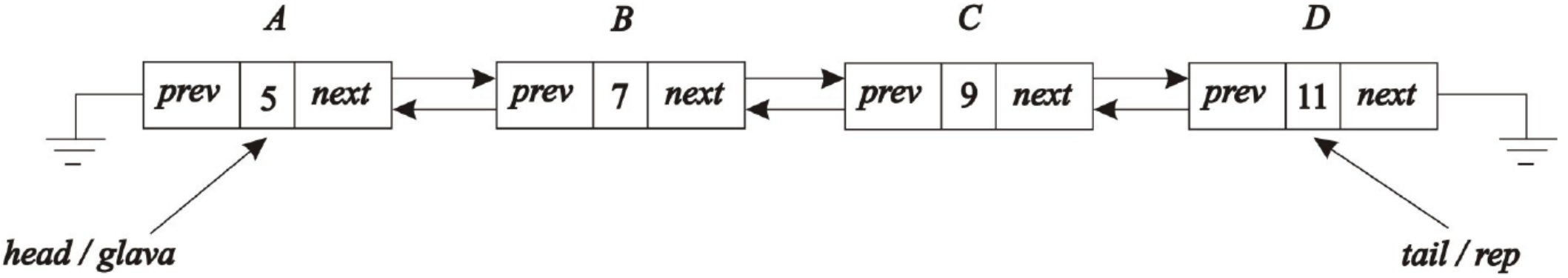
Brisanje elementa

- Želimo izbrisati ***C***
- Prevežemo kazalce mimo ***C***
  - ***B***.next (***C***.prev.next) = ***D***
  - ***D***.prev (***C***.next.prev) = ***B***
- Odstranimo ***C*** (sprostimo pomnilnik)



Brisanje elementa

- Kaj če:
  - Brišemo edini element?
    - glava = rep = NIL
  - Brišemo glavo (***A***)
    - glava = ***A***.next
    - ***A***.prev.next ne spreminjamo
  - Brišemo rep (***D***)
    - rep = ***D***.prev
    - ***D***.next.prev ne spreminjamo



Brisanje elementa

```
procedure BRIŠI(head, elem)
 begin
   if elem.prev=NIL and elem.next=NIL then
     begin
       head:=NIL;
       tail:=NIL;
     end
   else
     begin
        if elem.prev<>NIL then
            elem.prev.next := elem.next;
        else
         begin
            head := elem.next;
            head.prev := NIL;
         end

        if elem.next<>NIL then
            elem.next.prev := elem.prev
        else
         begin
            tail := elem.prev;
            tail.next := NIL;
         end
     end
    delete elem;
end
```

brišemo edini element

brišemo glavo

brišemo rep

Izpis seznama

- Od glave proti repu
- Od repa proti glavi
- Podobno iskanju

```
function IZPISI_GLAVA_DO_REP(head)
 begin
    current := head;
    while current<>NIL do
        izpisi current.key;
        current := current.next;
    end
end
```

- Alternativa: rekurziven izpis (tudi iskanje)

Zahteve naloge

- **Implementirajte dvojno-povezan seznam**, vse funkcije in aplikacijo za delo z njim
- Implementirajte funkcije za testiranje hitrosti seznama in polja
  - Testirajte hitrost, primerjajte s poljem in zapišite v poročilo -> več na naslednjih prosojnicah
- Po vsaki akciji se naj ponovno prikaže glavni meni (razen izhod)
- Obravnavajte robne primere. Aplikacija se ne sme sesuti, ne sme podajati lažnih informacij. Obravnava in javljanje napak: iskanje v praznem seznamu, brisanje neobstoječega podatka…
- Izpis ne sme ničesar spreminjati

```
Dvojno povezan seznam – izbira:

1) Iskanje podatka
2) Vnos podatka v glavo
3) Vnos podatka za elementom
4) Vnos podatka za repom
5) Brisanje podatka
6) Izpis seznama od glave proti repu
7) Izpis seznama od repa proti glavi
8) Testiraj hitrost
9) Konec

Izbira:
```

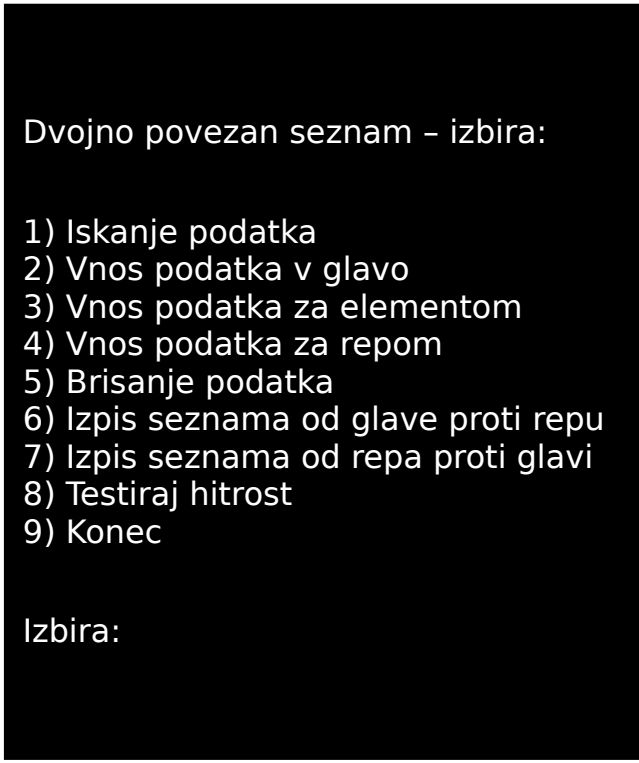
Zahteve naloge

1) Uporabnik vnese vrednost, ki jo želi poiskati. Program izpiše, ali vrednost že obstaja.
2) Uporabnik vnese vrednost za vnos
3) Uporabnik vnese dve števili:
    1) Vrednost za vnos
    2) Vrednost (ključ) za katerim elementom želi vnesti nov element
        1) Izvede se procedura ISKANJE
4) Uporabnik vnese vrednost za vnos
5) Uporabnik vnese vrednost elementa za brisanje (ključ)
    1) Izvede se procedura ISKANJE
    2) Izbriše se prva najdena vrednost
6) in 7) Izpišite ključe (vsebino) seznama

```
Dvojno povezan seznam – izbira:

1) Iskanje podatka
2) Vnos podatka v glavo
3) Vnos podatka za elementom
4) Vnos podatka za repom
5) Brisanje podatka
6) Izpis seznama od glave proti repu
7) Izpis seznama od repa proti glavi
8) Testiraj hitrost
9) Konec

Izbira:
```

Implementirati je potrebno funkcije *NAJDI*, *VSTAVI_ZA*, *VSTAVI_V_GLAVO,* *BRIŠI,* *IZPISI_GLAVA_DO_REP, IZPISI_REP_DO_GLAVA*. Uporabnik bo lahko klical funkcije *NAJDI, VSTAVI_V_GLAVO*, *VSTAVI_ZA* in *BRIŠI* neposredno preko menija, medtem, ko bo funkcijo *NAJDI* klical posredno tudi pri vstavljanju novega elementa v seznam in pri brisanju. Ko se program zažene, se mora izpisati meni na sliki, nato pa mora program čakati na vpis posamezne izbire.

Pri izbiri *Iskanje podatka* uporabnik vnese število, ki ga želi poiskati. Program izpiše ali element obstaja. Pri izbiri *Vnos podatka v glavo* se prebere eno samo število, ki vpišemo v glavo seznama. Pri izbiri *Vnos podatka za elementom*, uporabnik vpiše dve števili. Najprej število, za katerim bi rad vpisal novo vrednost, nato pa še vrednost elementa, ki bi ga želel dodati. S funkcijo *Vnos podatka za repom* od uporabnika zahtevamo število, ki dodamo na konec seznama. S funkcijo *NAJDI* je potrebno poiskati element, ki vsebuje to število. Glede na izpis 1 se iskanje ustavi pri prvem najdenem elementu. Funkcija *NAJDI* nam tako vrne kazalec na ta element, nakar kličemo še funkcijo *VSTAVI_ZA*, v katero vstavimo ta kazalec kot parameter. V kolikor število, za katerim bi želeli dodati nov element, v seznamu ne obstaja, mora program javiti napako. Enako mora javiti napako, če želimo pri menijski postavki *Brisanje podatka* izbrisati podatek, ki ga v seznamu ni. Ob izbiri menijskih postavk 6 in 7 je potrebno izpisati vsebino dvojno povezanega seznama. Program se konča, ko uporabnik izbere menijsko postavko *Konec*.



Testiranje hitrosti seznama in polja

- Opcija 8)
- Uporabnika najprej vprašajte za število elementov (*N*):
  1) Izvedite vstavljanje *N* (1->*N*) elementov v glavo seznama
  2) Izračunajte vsoto vseh ključev iz seznama
  3) Izvedite vstavljanje *N* (1->*N*) elementov na prvo mesto v polju (polje za vsako vstavljeno vrednost pomaknite za eno mesto v desno) -> končni rezultat: [N, N-1,...., 1]
  4) Izračunajte vsoto vseh vrednosti iz polja
  5) Izvedite vstavljanje *N* (1->*N*) elementov na konec polja -> končni rezultat [1, 2, …, N]
- Pri vsaki operaciji (1-5) izpišite čas, ki ga je operacija potrebovala
- Pri opciji 8 je dovoljeno uporabiti seznam, vgrajen v C++: http://en.cppreference.com/w/cpp/container/list

1) vstavljanje v glavo

```
POČISTI_SEZNAM();
for i:=1 to N do
      e=USTVARI_ELEMENT();
      e.key=i;
      VSTAVI_V_GLAVO(glava,rep,e);
end
```

2) vsota

```
function VSOTA_GLAVA_DO_REP(head)
 begin
     vsota := 0
     current := head;
     while current<>NIL do
          vsota := vsota + current.key;
          current := current.next;
     end
     izpisi(vsota)
end
```

3) vstavljanje na prvo mesto

```
int* polje=new int[N];
for(int i=1;i<=N;i++){
    PREMAKNI_VSE_VREDNOSTI_DESNO()
    polje[0]=i;
}
```

4) vsota

```
long long sum=IZRACUNAJ_
    VSOTO_VSEH_ELEMENTOV(polje);
```

5) vstavljanje na konec polja

```
int* polje=new int[N];
for(int i=0;i<N;i++)
      polje[i]=i+1;
```

Testiranje hitrosti seznama in polja

- V tekstovno datoteko (.txt), ki jo tudi oddate na eštudij, zapišite:
  - Skupen čas vstavljanja 100000 elementov (1 do 100000) v glavo seznama
  - Skupen čas vstavljanja 100000 elementov (1 do 100000) na prvo mesto v polje
  - Skupen čas vstavljanja 100000 elementov (1 do 100000) na konec polja
  - Skupen čas izračuna vsote 100000 elementov (ključev) iz seznama
  - Skupen čas izračuna vsote 100000 vrednosti iz polja
    - Vrednost vsote bi morala biti enaka kot pri seznamu (5000050000)
    - Uporabite long long: 64-bitni podatkovni tip
  - Ugotovitve
    - Katera podatkovna struktura je primernejša (hitrejša) za posamezen scenarij (vstavljanje, premikanje skozi vse vrednosti). Če kakšna operacija traja premalo/preveč časa, lahko testiranje izvedete z večjim/manjšim številom elementov.
- Program prevedite kot »**release/izdaja**« in ne kot »debug/razhroščevanje«!

Merjenje časa izvajanja

C++

```
#include <time.h>
                                        ...
clock_t start = clock();
dolg_algoritem();
clock_t finish = clock();
double duration = (double)(finish - start)/CLOCKS_PER_SEC;
                                        ...
```

C++11

Boljši pristop

```
#include <chrono>
int main(int argc, char *argv[]){
                                        ...
auto start=std::chrono::steady_clock::now();
dolg_algoritem();
auto end=std::chrono::steady_clock::now();
std::cout << "Čas trajanja: " <<
   std::chrono::duration_cast<std::chrono::microseconds>(end - start).count() <<
   "µs."<<std::endl;
```

lahko uporabite tudi `milliseconds`

C++11

- Pri novejših prevajalnikih je podpora za C++11 privzeto vključena
- V nasprotnem primeru je potrebno ročno vklopiti podporo za C++11 v vašem razvojnem okolju oziroma nastavitvah projekta (CXXFLAGS += -std=c++11):
  - http://stackoverflow.com/questions/16948382/how-to-enable-c11-in-qt-creator
  - http://stackoverflow.com/questions/9131763/eclipse-cdt-c11-c0x-support

Merjenje časa - Java

```
long start = System.currentTimeMillis();
// ...
long finish = System.currentTimeMillis();
long timeElapsed = finish - start;
```

- Vrednost naloge:
  - 4 točke
    - Vstavljanje: 1
    - Iskanje: 0,5
    - Izpis: 0,5
    - Brisanje: 1
    - Testiranje hitrosti (koda + rezultati): 1 točka