# Dvojno povezan seznam

## 1. Splošna predstavitev problema

Dvojno povezan seznam je podatkovna struktura, ki se pogosto uporablja v zahtevnejših aplikacijah, ko je nemogoče vnaprej predvideti število podatkov, ki jih bomo hranili, potrebujemo pa hiter dostop do njih. V tem primeru uporabimo dvojno povezan seznam, ki nam za razliko od klasičnega seznama omogoča pregledovanje tudi od vrha proti dnu oziroma repa proti glavi.

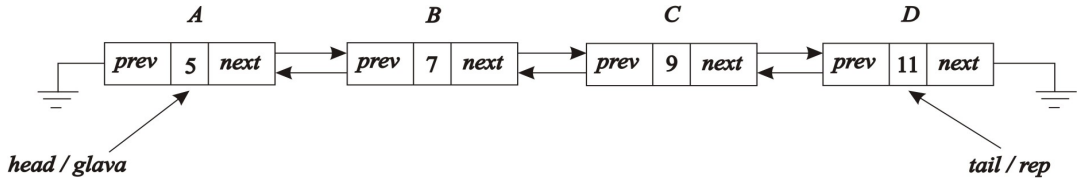
V našem primeru bo dvojno povezan seznam vseboval tri podatke:

- vrednost elementa oz. ključ (*key*),
- kazalec na predhodni element (*prev*) in
- kazalec na naslednji element (*next*).

Zapis elementa v C++ bi bil naslednji (predpostavimo, da je ključ celo število):

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

Prvi element seznama se imenuje *glava* (*head*) in nima predhodnika, kar označujemo z NIL (tj. *x.prev=NIL*). Zadnji element seznama je element brez naslednika (tj. *x.next=NIL*), ki ga imenujemo *rep* (*tail*).




**Slika 1:** Primer dvojno povezanega seznama

Slika 1 prikazuje strukturo dvojno povezanega seznama s celoštevilskimi ključi. Na sliki vidimo štiri elemente *A*, *B*, *C* in *D*, ki hranijo ključe 5, 7, 9, in 11. Do seznama vedno dostopamo preko kazalca na *glavo* (oziroma *rep*, če se premikamo v obratni smeri), med elementi seznama se lahko seveda pomikamo v obeh smereh. Seznam je prazen, če sta tako *glava* kot *rep* enaka NIL (*head=tail=NIL*). V našem primeru kazalec *glava* kaže na element *A*, kazalec *rep* pa na element *D*.

## 2. Pomoč pri implementaciji

Med pomembnejše operacije nad dvojno povezanim seznamom spadajo naslednje:

- iskanje elementa z znanim ključem v seznamu,
- vstavljanje novega elementa v seznam,
- brisanje elementa iz seznama.

### Iskanje elementa

Iskanje elementa v dvojno povezanem seznamu poteka enako kot pri enojno povezanem, tako da pričnemo z iskanjem v glavi seznama in se pomikamo proti repu dokler ne najdemo iskanega ključa ali pa naletimo na *NIL*. V izpisu 1 je prikazan psevdokod funkcije *NAJDI*(*head, key*), ki v seznamu z glavo z imenom *head* poišče in vrne kazalec na element z vrednostjo ključa *key*. Če iskanega elementa ni v seznamu, funkcije vrne *NIL*, glej izpis 1.

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

**Izpis 1:** Psevdokod funkcije *NAJDI*

### Vstavljanje elementa v seznam

Vstavljanje novega elementa v seznam izvedemo tako, da ga vrinemo za nek podan element seznama (izpis 2), ali pa ga vstavimo v glavo (tj. na prvo mesto – izpis 3). Psevdokoda vstavljanja novega elementa *new_el* za nek element *elem*, ki že obstaja v seznamu, je procedura *VSTAVI_ZA*(*elem*, *new_el*).

```
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
```

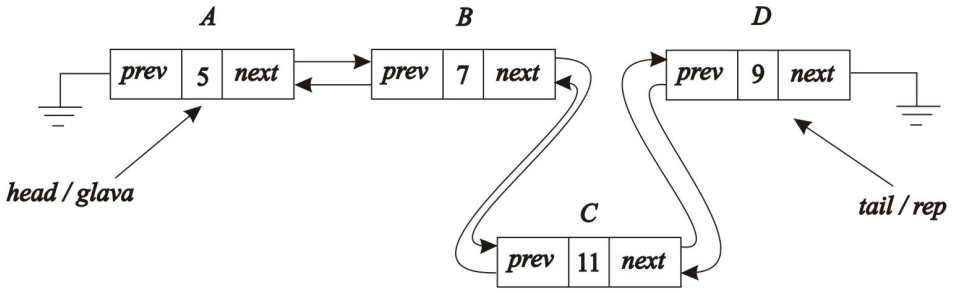
**Izpis 2:** Psevdokod funkcije *VSTAVI_ZA*

Vstavljanja novega elementa *new_el* na prvo mesto seznama z glavo *head* je procedura *VSTAVI_V_GLAVO*(*head*, *new_el*), prikazana v izpisu 3.

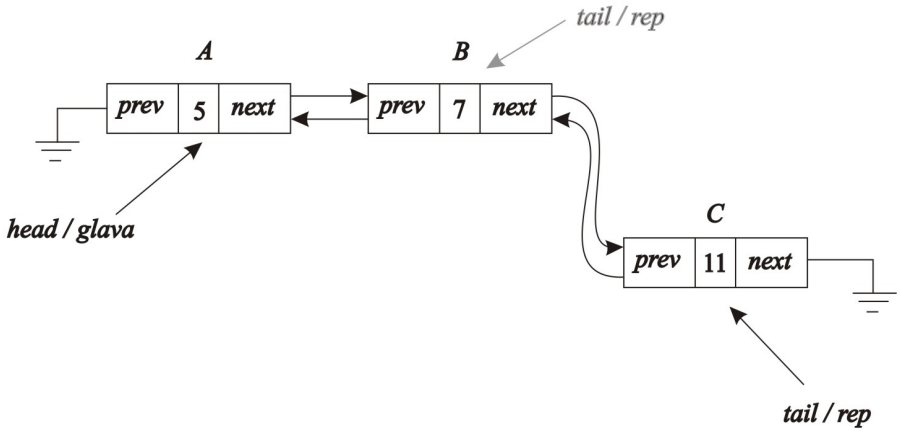
```
procedure VSTAVI_V_GLAVO(head, new_el)
 begin
      new_el.next := head;
      new_el.prev := NIL;
      if head<>NIL then
            head.prev := new_el;
      else
            tail := new_el;

      head := new_el
 end
```

**Izpis 3:** Psevdokod funkcije *VSTAVI_V_GLAVO*

Vstavljanje novega elementa za poljubnim elementom v seznam in vstavljanje v glavo seznama lahko vidimo na slikah 2 in 3.




a)




b)

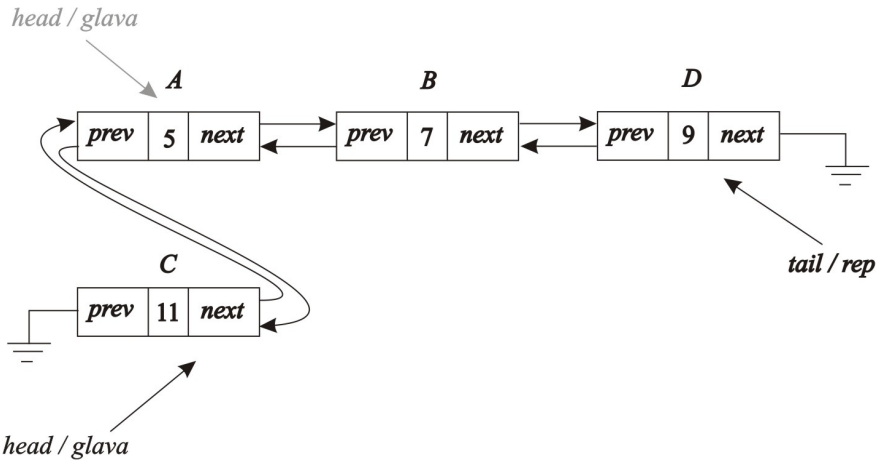
**Slika 2:** a) Vpis novega elementa med dva obstoječa
b) Vpis novega elementa v rep

Na sliki 2a je prikazan primer vstavljanja elementa, ki ga označimo s črko *C* v dvojno povezan seznam med elementa *B* in *D*. Vidimo, da je potrebno najprej postaviti

kazalec *C.next* na vrednost *B.next*, kazalec *B.next* pa postavimo na *C*. Tako bo sedaj element *B* kazal na *C*, ta pa dalje na *D*.

Sedaj je potrebno postaviti še povratne povezave. Tako kazalec *C.prev* postavimo na *B*, prav tako pa moramo preusmeriti tudi kazalec *D.prev*, ki mora kazati na element *C*. Do elementa *D* pridemo s kazalcem *C.next*. Pri tem moramo paziti, da element *B* ni bil rep. To preverimo s pogojem `if new_el.next <> NIL` . Če je bil element *B* namreč rep seznama, element *D* ne obstaja, novi *rep* pa mora kazati na kazalec *C*, glej slika 2b.

Na sliki 3 vidimo primer vstavljanja v glavo seznama. Ta funkcija pride vedno v poštev na začetku, ko je seznam prazen, lahko pa po potrebi tudi kadarkoli pozneje.




**Slika 3:** Vstavljanje v glavo seznama

Na sliki 3 vidimo primer, ko kaže *glava* pred vrivanjem novega elementa na element *A*, ta pa dalje na element *B*. Sedaj pa želimo na prvo mesto vstaviti element *C*. Tako rečemo, da je kazalec *C.next* enak *glavi*, *C.prev* pa je enak *NIL*.
Nato je potrebno vzpostaviti tudi ustrezno povratno povezavo. Če vstavljamo v prazen seznam (*head = NIL*), moramo samo preusmeriti kazalec *rep*, da bo kazal na ta novi element, sicer pa postavimo kazalec *head.prev* na novi element *C*.
V vsakem primeru pa je na koncu potrebno prestaviti še kazalec *glava*, ki mora kazati na element *C*, ki je nov začetek seznama.

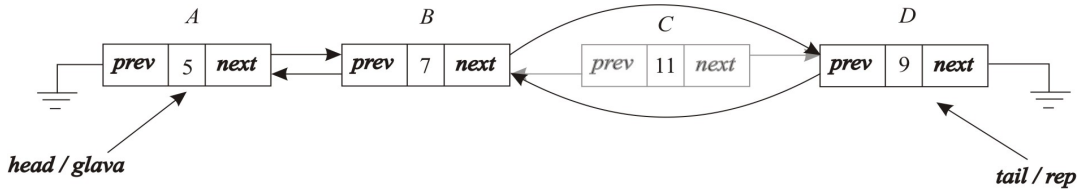
## Brisanje elementa iz seznama

Pri brisanju elementa iz seznama je potrebno paziti, da pravilno povežemo kazalce predhodnika ter naslednika izbranega elementa tako, da ta element preprosto izločimo iz seznama. Psevdokod procedure brisanja elementa *elem* iz seznama z glavo *head*, je procedura *BRIŠI*(*head*, *elem*), ki jo lahko vidimo v izpisu 4.

```
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
```

**Izpis 4:** Psevdokod funkcije *BRIŠI*

Brisanje elementa iz dvojno povezanega seznama je prikazano na sliki 4, kjer iz seznama brišemo element C, ki se nahaja med elementoma B in D.




**Slika 4:** Brisanje elementa iz dvojno povezanega seznama

Kot je iz slike 4 razvidno, postavimo najprej kazalec *B.next* na vrednost kazalca *C.next*, s čimer preskočimo element *C*. Podobno pa naredimo tudi s kazalcem *D.prev*, ki ga postavimo na vrednost *C.prev*, s čimer preskočimo element *C* tudi s te strani. Ker na elemet *C* sedaj ne kaže noben kazalec več, ga lahko zbrišemo.

## Izpis vsebine seznama

V dvojno povezanem seznamu imamo dve možnosti izpisa podatkov in sicer od *glave* proti *repu* in obratno od *repa* proti *glavi*. V obeh primerih je izpisovanje seznama enostavno. Primer za izpis od glave do repa prikazuje psevdokod v izpisu 5.

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

**Izpis 5:** Psevdokod funkcije IZPISI_GLAVA_DO_REP