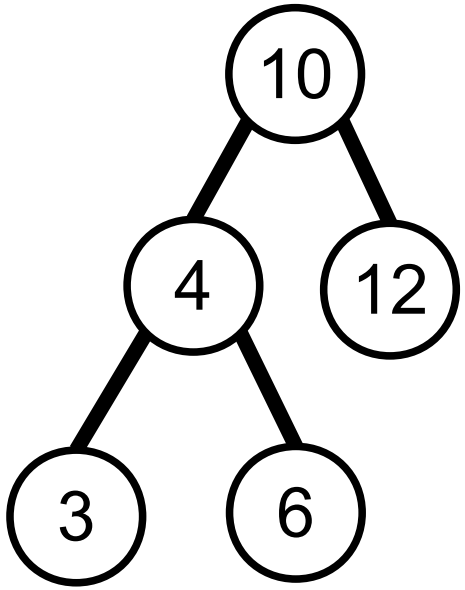
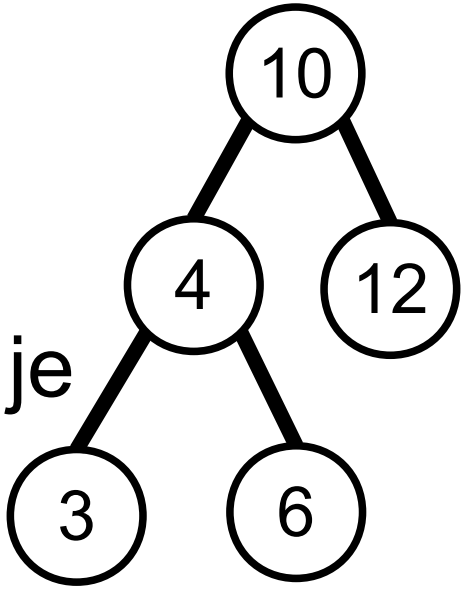
Binarno iskalno drevo

- Drevo: hierarhična podatkovna struktura
- Binarno: vsako vozlišče ima največ dva sinova
- Iskalno: primerno za iskanje podatkov → do podatkov želimo priti v najkrajšem možnem času
- Vsako vozlišče vsebuje podatke

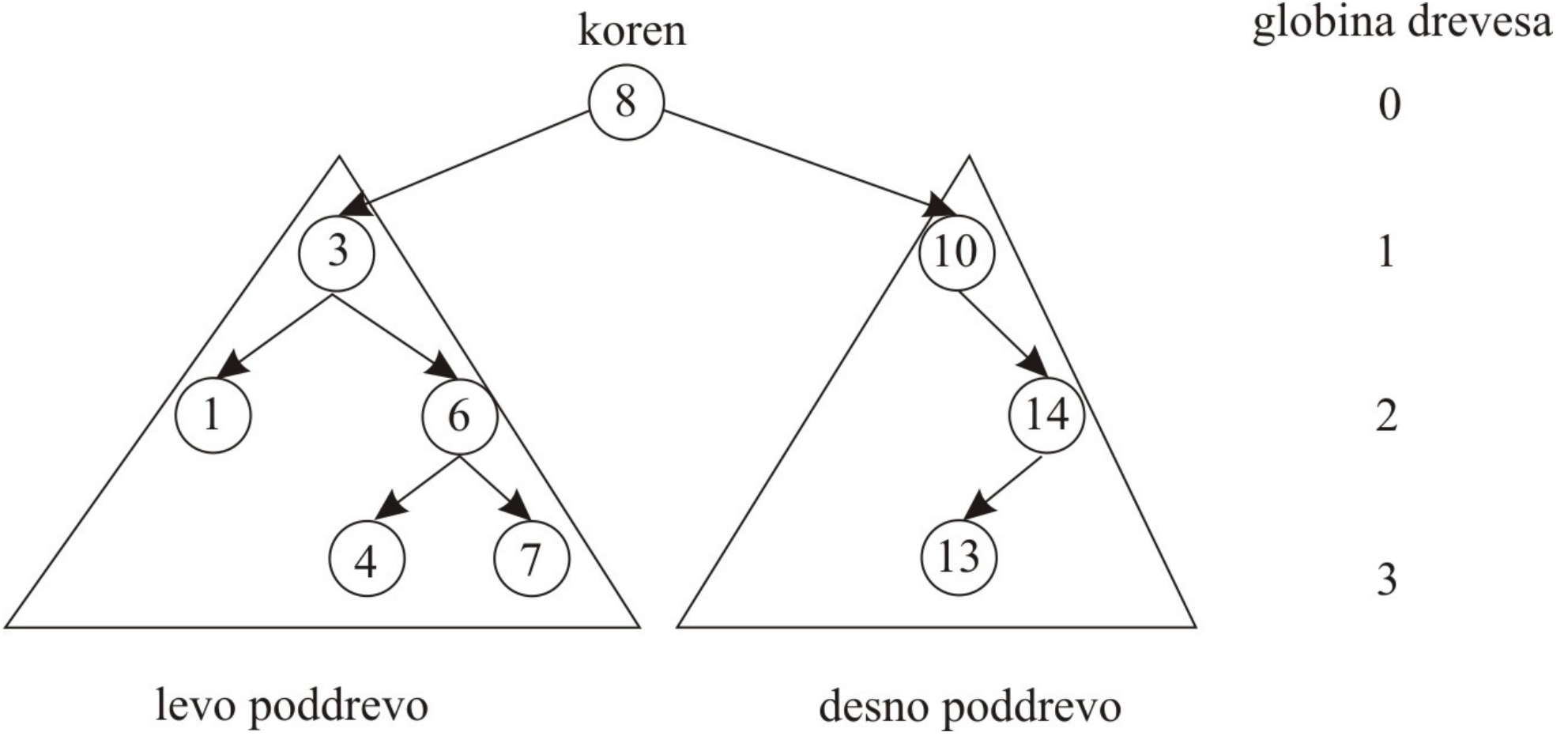


Terminologija

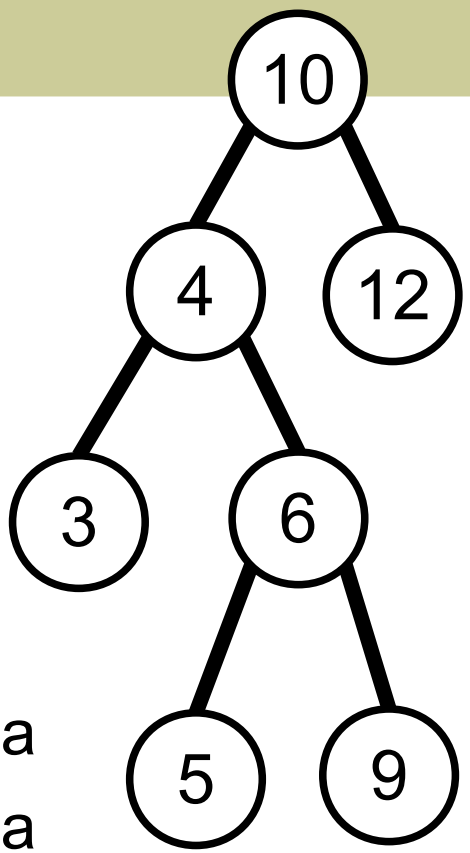
- Koren – vozlišče brez očeta, vrhnje vozlišče
- Vsako vozlišče razen korena ima točno enega očeta
- Vozlišče ima lahko največ 2 sinova (levi in desni)
- Vsako vozlišče vsebuje ključ (število)
- List drevesa je vozlišče brez sinov
- Notranje vozlišče ima vsaj enega sina
- Vsako vozliče z vsemi svojimi potomci je **poddrevo** → rekurzija



Primer binarnega iskalnega drevesa



Organizacija podatkov



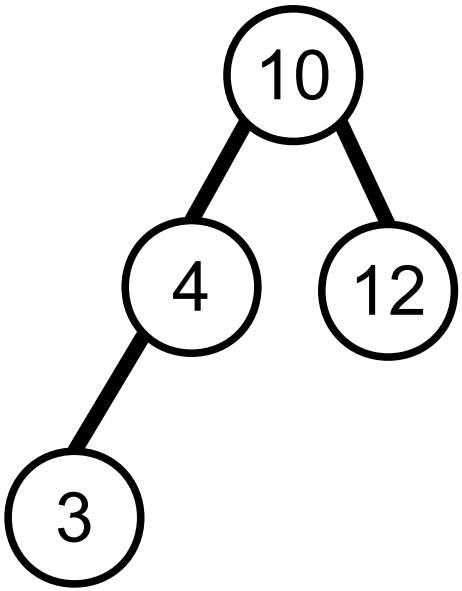
- Iskalno drevo je organizirano tako, da podatke najdemo v zelo kratkem času
  - Vozlišča so urejena po velikosti
    - Levi sin manjši od vrednosti v vozlišču
    - Desni sin večji od vrednosti v vozlišču
    - Vozlišča, ki so v levem poddrevesu so manjša
    - Vozlišča, ki so v desnem poddrevesu so večja
    - V drevesu so shranjena vozlišča z unikatnimi ključi
  - Logaritemski čas iskanja

Primer v C++

```
struct Vozlisce {
    int key;
    Vozlisce* oce;
    Vozlisce* leviSin;
    Vozlisce* desniSin;
};
```

```
Vozlisce *v10=new Vozlisce();
v10->key=10;

Vozlisce *v4=new Vozlisce();
v4->key=4;

Vozlisce *v3=new Vozlisce();
v3->key=3;

Vozlisce *v12=new Vozlisce();
v12->key=12;
```

```
v10->oce=NULL;
v10->leviSin=v4;
v10->desniSin=v12;

v4->oce=v10;
v4->leviSin=v3;
v4->desniSin=NULL;

v3->oce=v4;
v3->leviSin=NULL;
v3->desniSin=NULL;

v12->oce=v10;
v12->leviSin=NULL;
v12->desniSin=NULL;

Vozlisce *koren=v10;
```



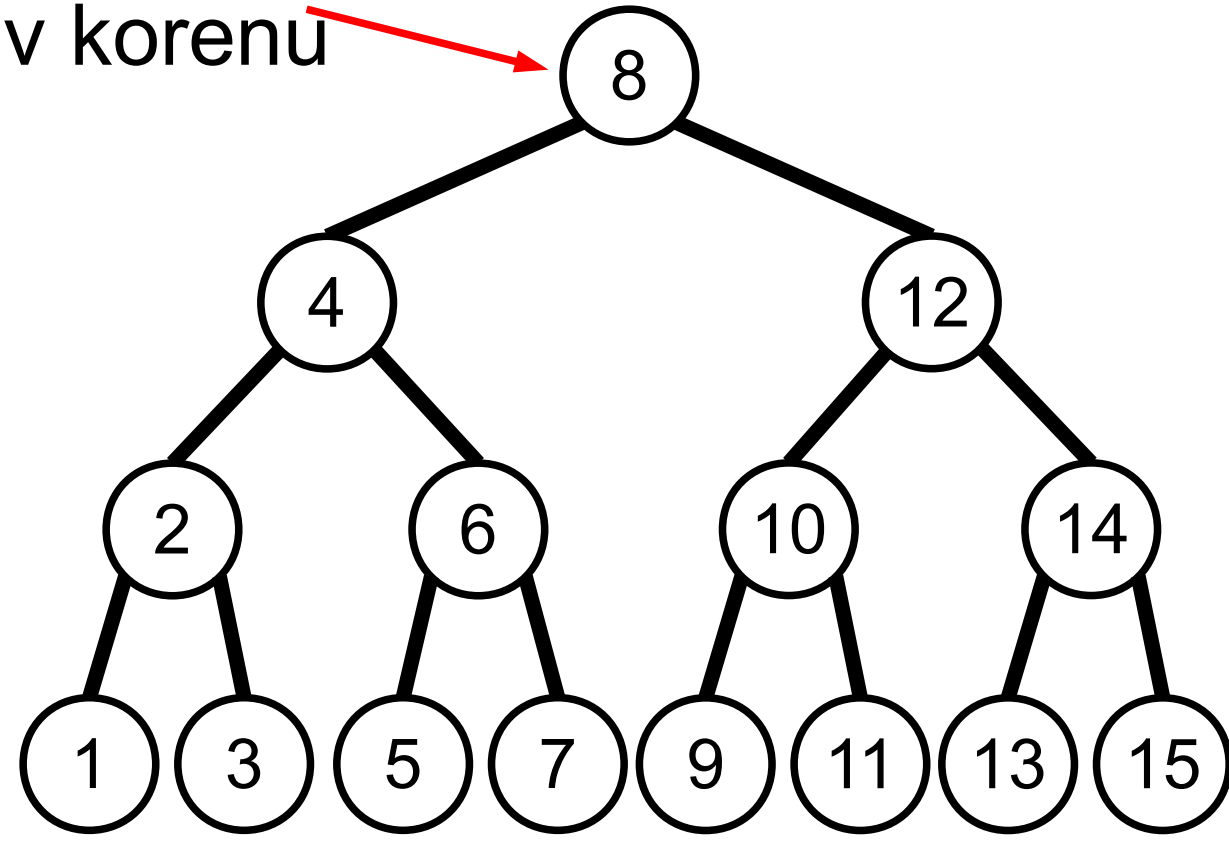
Operacije

- Iskanje podatkov po ključu
- Vstavljanje podatkov
- Brisanje podatkov
- Iskanje najmanjšega in največjega elementa
- Iskanje predhodnika in naslednika nekega elementa
- Operacije so odvisne od globine drevesa O(h), kar je približno $\log_2$ od števila elementov

Iskanje

- Zanima nas ali se nek ključ nahaja v drevesu
- Kako ugotovimo ali se vrednost 5 nahaja v drevesu?

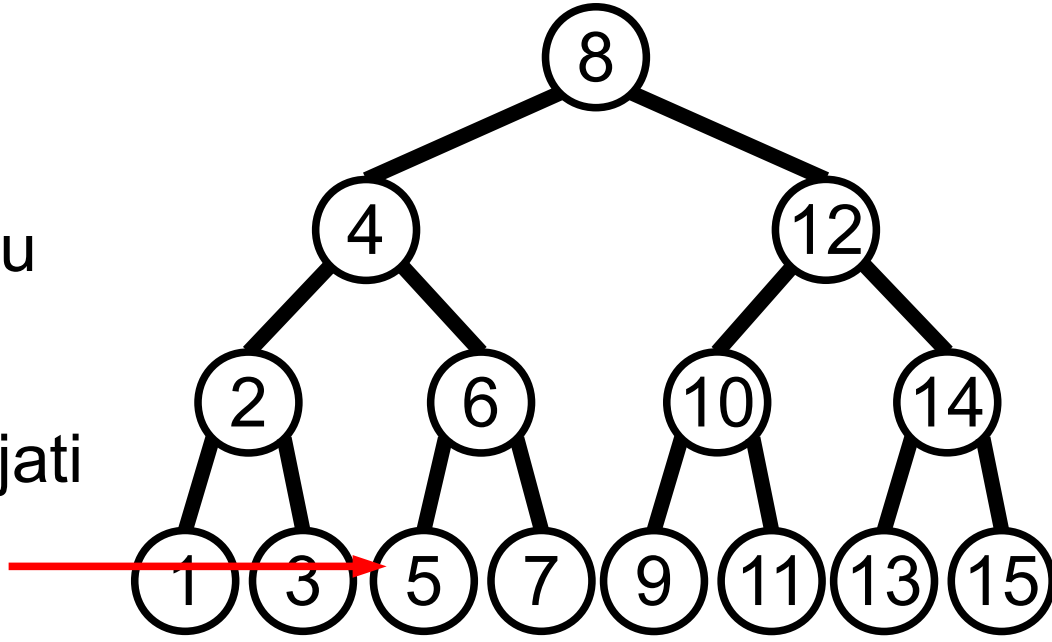
1. Iskanje pričnemo v korenu



Iskanje

- Zanima nas ali se nek ključ nahaja v drevesu
- Kako ugotovimo ali se vrednost 5 nahaja v drevesu?

1. Začnemo v korenu
2. Preverimo če je 5 v korenu
3. Vrednost 5 bi se morala nahajati v levem poddrevesu
4. 5<>4
5. Vrednost 5 bi se morala nahajati v desnem poddrevesu
6. 5<>6
7. Vrednost 5 bi se morala nahajati v levem poddrevesu



Iskanje

```
function POISCI(T, k)
 begin
   x := T
   while x<>NIL do
     if k = x.key
        return x;
     if k < x.key then
           x := x.leviSin;
        else
           x := x.desniSin;
   return x;
end
```

Iterativna varianta

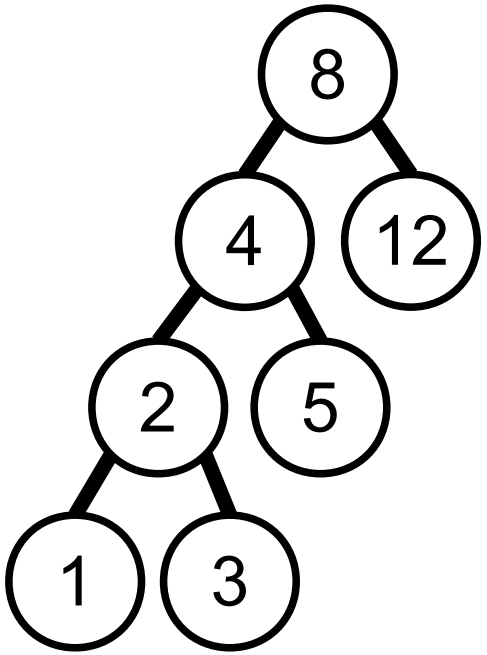
```
function POISCI(x, k)
 begin
   if x=NIL or x.key = k then
     return x;
   else
     if k < x.key then
      return POISCI(x.leviSin, k)
     else
      return POISCI(x.desniSin, k)
end
```

Rekurzivna varianta

- Iskanje začnemo v korenu in gremo do listov

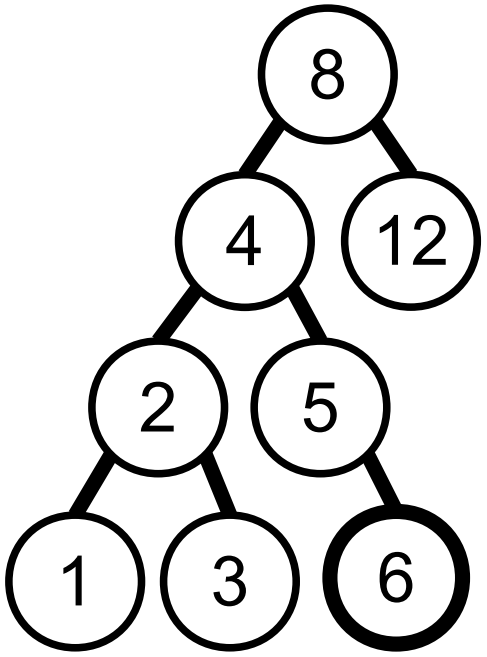
Vstavljanje

- Podobno kot pri iskanju: gremo do dna
- Ustvarimo levo ali desno vozlišče od lista
- Kako bi vstavili vrednost 6?



Vstavljanje

- Podobno kot pri iskanju: gremo do dna
- Ustvarimo levo ali desno vozlišče od lista
- Kako bi vstavili vrednost 6?
  - Gremo do dna, kot da bi iskali 6
  - Ker je 6>5, 6 vstavimo kot desni sin



Vstavljanje

- Kaj če vstavljamo v prazno drevo?
  - Kazalec na drevo mora kazati na novo vozlišče
- Kaj če vozlišče že obstaja?
  - Javimo napako ali implementiramo podporo za več enakih vrednosti

```
function VSTAVI(T, k)
 begin
   y := NIL;
   x := T;

   while x<>NIL do
      y := x;
      if k < x.key then
        x := x.leviSin;
      elseif k > x.key then
        x := x.desniSin;
      else
        Javi napako in prekini vstavljanje
   end
   z := USTVARI_VOZLISCE();
   z.key := k;
   z.oce := y;

   if y = NIL then
      T := z;
   else
      if z.key < y.key then
         y.leviSin := z;
      else
         y.desniSin := z;
 end
```

pomikamo se do dna, oz. najdemo prostor za vstavljanje

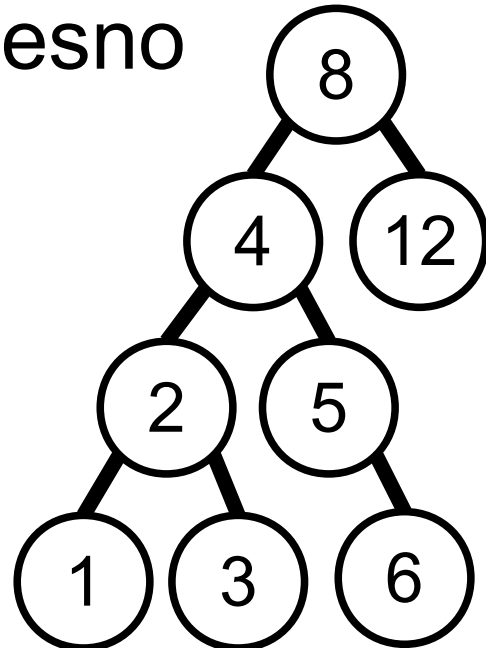
povezovanje kazalcev

vstavljamo v prazno drevo

prevezovanje kazalcev

Urejen izpis

- Omogoča izpis vseh vrednosti v drevesu po naraščajočem vrstnem redu
- Vemo, da je drevo urejeno
  - Levo poddrevo vsebuje manjše vrednost, desno pa večje
- Najprej izpišimo levo poddrevo in nato desno



1 2 3 4 5 6 8 12

Urejen izpis vrednosti

```
function UREJEN_IZPIS(x)
 begin
       if x <> NIL then
        begin
            UREJEN_IZPIS(x.leviSin);
            Izpiši x.key;
            UREJEN_IZPIS(x.desniSin);
        end
 end
```

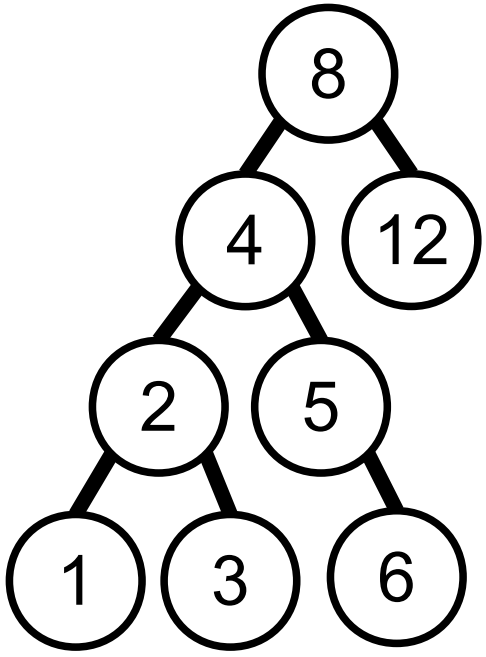
Izpis vseh povezav

```
function IZPIS_POVEZAV(x)
 begin
   if x.leviSin<>NIL then
    begin
     Print (x.key -> x.leviSin.key)
     IZPIS_POVEZAV(x.leviSin);
    end

   if x.desniSin<>NIL then
    begin
     Print (x.key -> x.desniSin.key)
     IZPIS_POVEZAV(x.desniSin);
    end
end
```

Minimalna in maksimalna vrednost

- Pomikamo se po levih/desnih sinovih od korena do lista

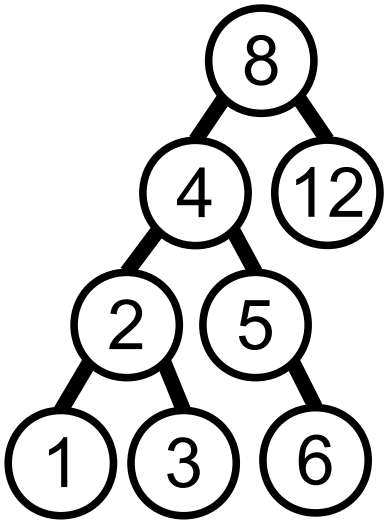
Min = 1
Max = 12



Minimalna in maksimalna vrednost

```
function MINIMUM(T)
 begin
    x := T
    while x.leviSin<>NIL do
       x := x.leviSin;
    return x;
end
```

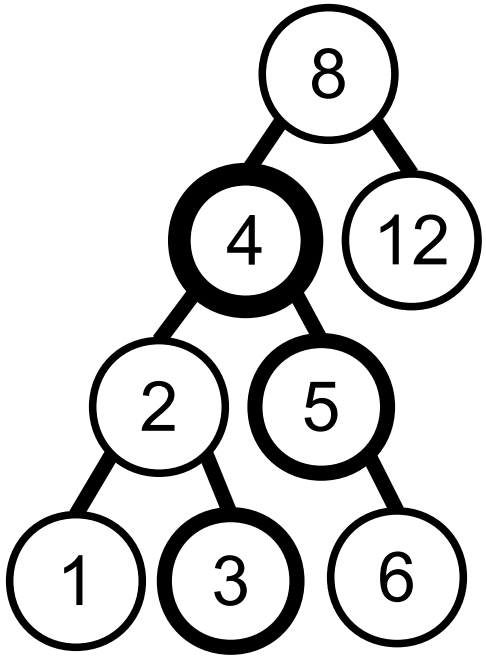
Min = 1
Max = 12



Naslednik/predhodnik

- Naslednik od 4?
- Predhodnik od 4?

- Postopek:
  - Minimalna vrednost v desnem poddrevesu
  - Maksimalna vrednost v levem poddrevesu

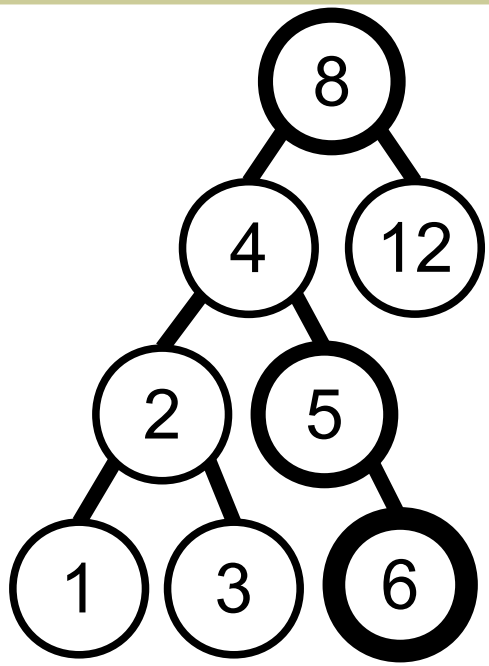


1 2 **3** **4** **5** 6 8 12

Naslednik/predhodnik

- Naslednik od 6?
- Predhodnik od 6?

1 2 3 4 **5** **6** **8** 12



- Iskanje lahko poteka tudi navzgor
- Naslednik (6 je predhodnik od 8)
  - Inverz iskanja predhodnika od 8
  - Potujemo gor preko desnih sinov + en skok preko levega sina
- Predhodnik (6 je naslednik od 5)
  - Inverz iskanja naslednika od 5
  - Potujemo gor preko levih sinov + en skok preko desnega sina

Naslednik/predhodnik

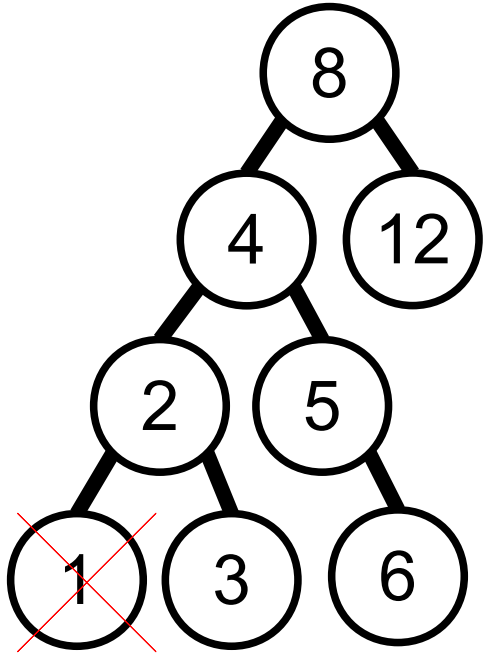
```
function NASLEDNIK(x)
 begin
   if x.desniSin<>NIL then
      return MINIMUM(x.desniSin);

   y := x.oce;
   while y <> NIL and x = y.desniSin do
      begin
         x := y;
         y := y.oce;
      end
   return y;
end
```

pomikamo se navzgor

Predhodnik: zamenjajte levi/desni in minimum/maksimum

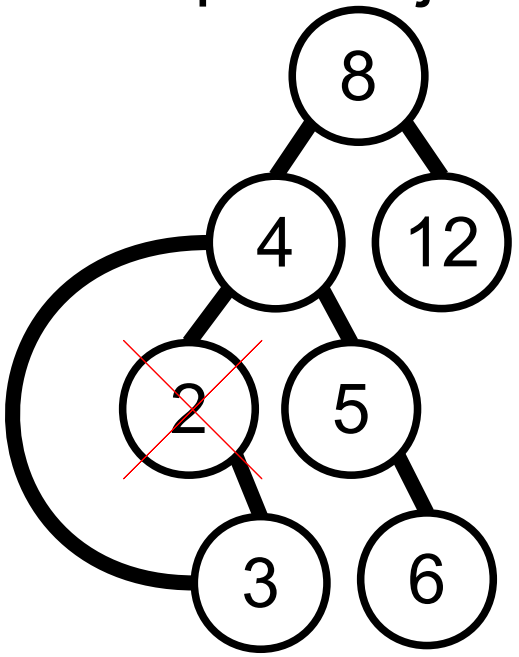
Brisanje vozlišča

- Brisanje mora ohranjati urejenost drevesa
- Želimo izbrisati 1 (list)



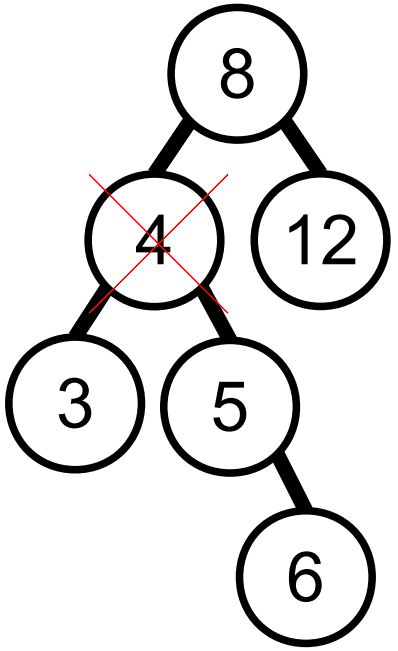
Brisanje vozlišča

- Brisanje mora ohranjati urejenost drevesa
- Želimo izbrisati 1 (list)
- Želimo izbrisati 2 (vozlišče z enim potomcem)
  - Prevežemo kazalce mimo 2 (podobno kot pri dvojno povezanemu seznamu)



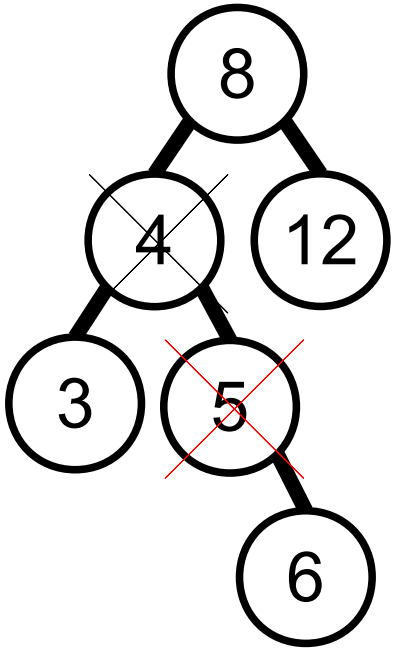
Brisanje vozlišča

- Želimo izbrisati 4
- Trik: Uporabi obstoječo rešitev
  - Najdi primerno vozlišče, ki ima največ enega potomca
  - Izbriši najdeno vozlišče
  - Vrednost premakni v vozlišče 4



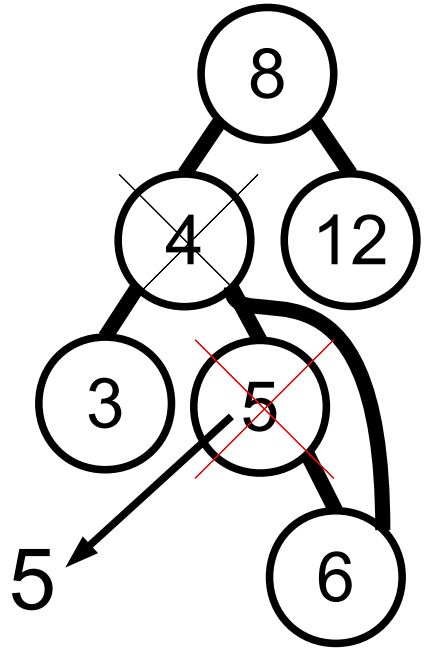
Brisanje vozlišča

- Želimo izbrisati 4
- Trik: Uporabi obstoječo rešitev
  - Najdi primerno vozlišče, ki ima največ enega potomca → naslednik
  - Izbriši naslednika (5) → brisanje vozlišča z največ enim potomcem
    - Naslednik od vozlišča z dvema potomcema ima vedno največ enega potomca (desni sin)
  - Vrednosti iz naslednika vpiši v vozlišče 4



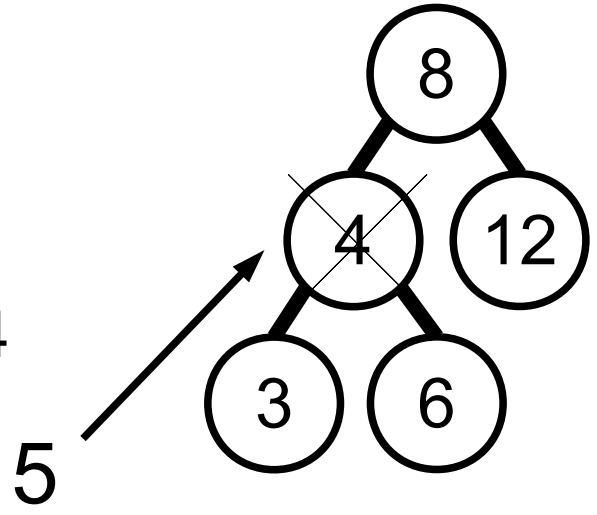
Brisanje vozlišča

- Želimo izbrisati 4
- Trik: Uporabi obstoječo rešitev
  - Najdi primerno vozlišče, ki ima največ enega potomca → naslednik
  - Izbriši naslednika (5) → brisanje vozlišča z največ enim potomcem
    - Naslednik od vozlišča z dvema potomcema ima vedno največ enega potomca (desni sin)
  - Vrednosti iz naslednika vpiši v vozlišče 4



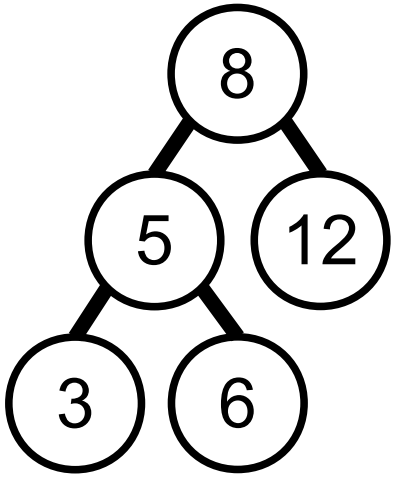
Brisanje vozlišča

- Želimo izbrisati 4
- Trik: Uporabi obstoječo rešitev
  - Najdi primerno vozlišče, ki ima največ enega potomca → naslednik
  - Izbriši naslednika (5) → brisanje vozlišča z največ enim potomcem
    - Naslednik od vozlišča z dvema potomcema ima vedno največ enega potomca (desni sin)
  - Vrednosti iz naslednika vpiši v vozlišče 4



Brisanje vozlišča

- Želimo izbrisati 4
- Trik: Uporabi obstoječo rešitev
  - Najdi primerno vozlišče, ki ima največ enega potomca → naslednik
  - Izbriši naslednika (5) → brisanje vozlišča z največ enim potomcem
    - Naslednik od vozlišča z dvema potomcema ima vedno največ enega potomca (desni sin)
  - Vrednosti iz naslednika vpiši v vozlišče 4



```
procedure BRISI(T, z)
 begin
    if z.leviSin = NIL or z.desniSin = NIL then
        y := z;
    else
        y := NASLEDNIK(z);

    if y.leviSin <> NIL then
        x := y.leviSin;
    else
        x := y.desniSin;

    if x <> NIL then
        x.oce := y.oce;
    if y.oce = NIL then
        T := x;
    else
        if y = y.oce.leviSin then
            y.oce.leviSin := x;
        else
            y.oce.desniSin := x;

    if y <> z then
        z.key := y.key;

    delete y;
 end
```

Brisanje po scenariju 3

Brisali bomo vozlišče y

Scenarij 1 in 2 (prevezovanje kazalcev)

Brisanje po scenariju 3 (premik vrednosti iz izbrisanega naslednika v z)

Zahteve naloge

- Implementirajte binarno iskalno drevo in aplikacijo za delo z njim
- Uporaba obstoječih knjižnic ni dovoljena
- 1) Uporabnik vnese vrednost, ki jo želi vnesti v drevo.
  - Program naj zavrne vstavljanje enakih vrednosti
- 2,3) Program izpiše celotno drevo
  - 2) Vrednosti so urejene
- 4) Uporabnik vnese število, ki ga želi poiskati, program izpiše ali vrednost že obstaja.
- 5) Program izpiše minimalno in maksimalno vrednost v celotnem drevesu
- 6) Uporabnik vnese ključ, od katerega želi poiskati predhodnika in naslednika
- 7) Uporabnik vnese ključ za brisanje
- Po vsaki akciji se glavni meni ponovno prikaže (razen izhod)

```
Binarno iskalno drevo – izbira:

1) Vnos podatka
2) Urejen izpis vrednosti
3) Izpis povezav
4) Iskanje
5) Poisci minimum in maksimum
6) Poisci predhodnika in naslednika
7) Brisi vrednost
8) Konec

Izbira:
```

Zahteve naloge

- Robni primeri! Obravnava in javljanje napak! Iskanje v praznem drevesu, brisanje neobstoječega podatka, vstavljanje enakih vrednosti...

```
Binarno iskalno drevo – izbira:

1) Vnos podatka
2) Urejen izpis vrednosti
3) Izpis povezav
4) Iskanje
5) Poisci minimum in maksimum
6) Poisci predhodnika in naslednika
7) Brisi vrednost
8) Konec

Izbira:
```

Zahteve naloge

Najprej je potrebno implementirati funkcije *VSTAVI, UREJEN_IZPIS, POISCI*, *MAKSIMUM, MINIMUM, PREDHODNIK* ter *NASLEDNIK*.

Funkcije za iskanje minimuma, maksimuma predhodnika in naslednika se bodo klicale direktno iz menija, ko bo uporabnik izbral ustrezen ukaz, funkcija za iskanje vozlišča pa se bo klicala posredno ob klicu funkcij *PREDHODNIK* in *NASLEDNIK*.

Nato bo potrebno implementirati še funkcijo *BRISI*, ki bo prav tako predhodno klicala funkcijo *POISCI*, da bo poiskala kazalec na vozlišče, ki ga želimo izbrisati.

Ob zagonu programa se tako mora zagnati meni, ki je prikazan na prejšnji prosojnici.

Ob izbiri ukaza *Vnos podatka* se vnesena vrednost vpiše v drevo. Pri vpisu že shranjene vrednosti vam bo program javil napako. Pri ukazu *Urejen izpis vrednosti* je potrebno izpisati urejeno zaporedje predhodno vnesenih vrednosti. Ukaz *Izpis povezav* izpiše vse povezave od očetov do sinov.

Ob ukazu Iskanje je potrebno uporabnika najprej vprašati za vrednost, ki jo iščemo v drevesu. Zatem izpišemo ali vrednost v drevesu obstaja ali ne. Ob ukazu *Poisci minimum in maksimum* se mora na ekran izpisati najmanjša oziroma največja vrednost, shranjena v drevesu. Pri iskanju predhodnika in naslednika je potrebno uporabnika predhodno vprašati za vrednost, katere predhodnika in naslednika bomo iskali, nato pa ustrezno vrednost tudi izpisati. Prav tako je potrebno uporabnika vprašati tudi za vrednost pri ukazu *Brisanje vrednosti*. Po vsaki operaciji mora ukaz *Urejen izpis* še naprej vračati urejeno zaporedje števil.

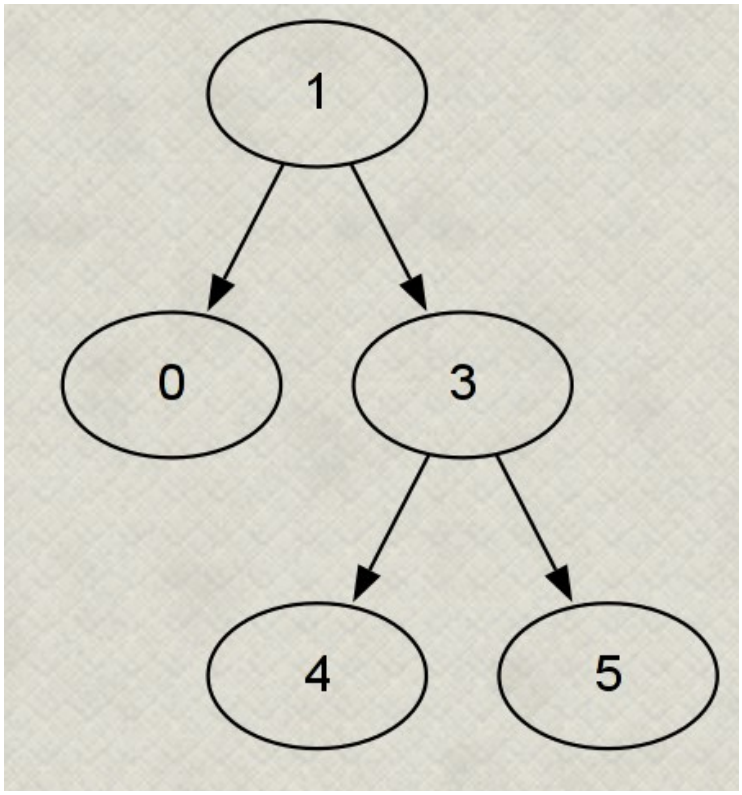
Program se konča, ko uporabnik izbere menijsko postavko *Konec*.

Izris drevesa – za lažje testiranje pravilnosti

```
digraph testgraph{
    1->0;
    1->3;
    3->4;
    3->5;
}
```

Uporabite:
IZPIS_POVEZAV(x)

- http://ushiroad.com/jsviz/



- Vrednost: 5 točk
  - Osnova (vstavljanje in iskanje): 2 točki
  - Izpis povezav in urejen izpis vrednosti: 0,5 točke
  - Maksimum in minimum: 0,5 točke
  - Predhodnik in naslednik: 1 točka
  - Brisanje: 1 točka