RI301
Strukture podataka

dr.sc. Edin Pjanić

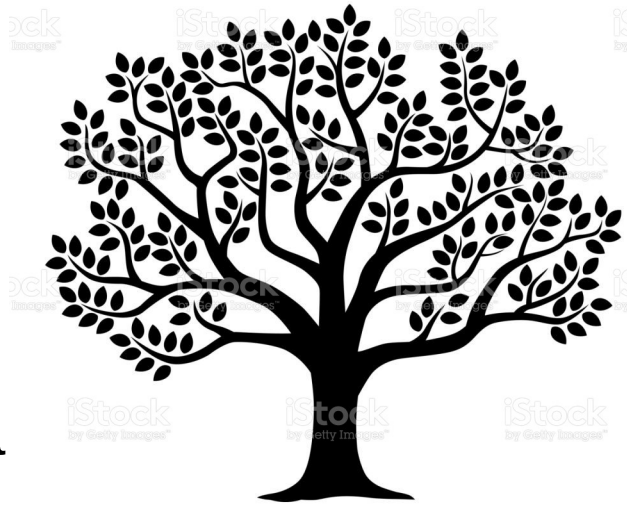
Pregled predavanja

- Stabla
    - definicija i terminologija
- Binarna stabla
- Stablo binarne pretrage (uređeno binarno stablo)

  – *Binary Search Tree (BST)*
- Osnovne operacije

Stabla

- **Stablo**: usmjerena aciklična struktura povezanih čvorova.
  - usmjerena: Ima vezu samo u jednom smjeru
  - aciklična: ako krenemo od jednog čvora, nema puteva koji vode do tog istog čvora odakle smo krenuli. Osim toga, do svakog čvora se može doći samo na jedan način, jednim putem.
- Primjeri stabala:
  - porodično stablo
  - organizaciona struktura firme i sl.
  - umjetna inteligencija: stablo odluka
  - kompajleri: stablo parsiranja...

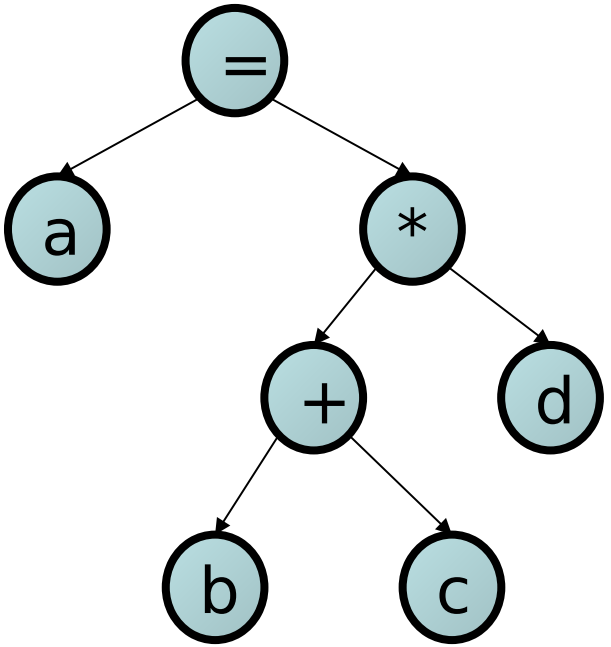


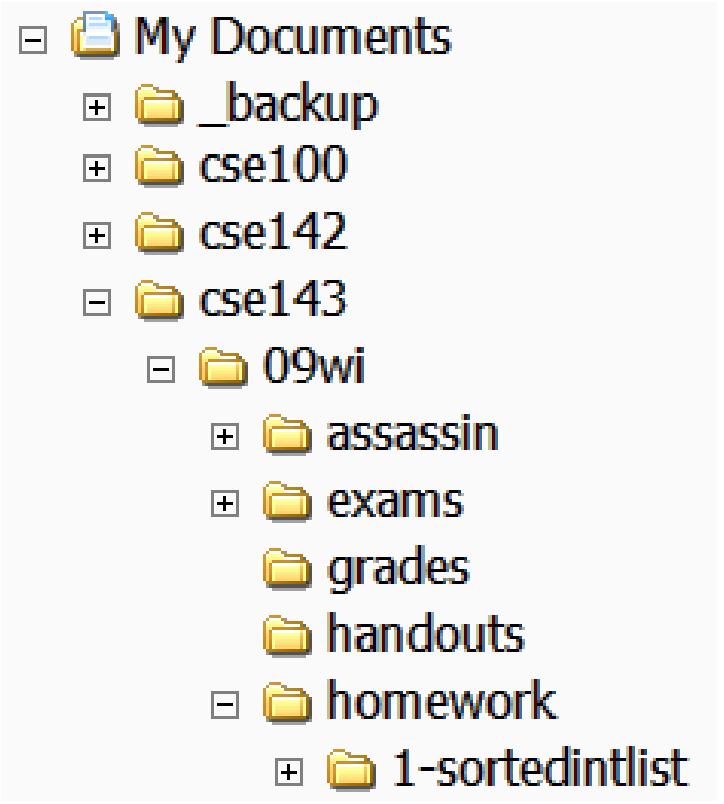
- Stabla imaju dobre performanse pri dodavanju, uklanjanju i pretraživanju elemenata.

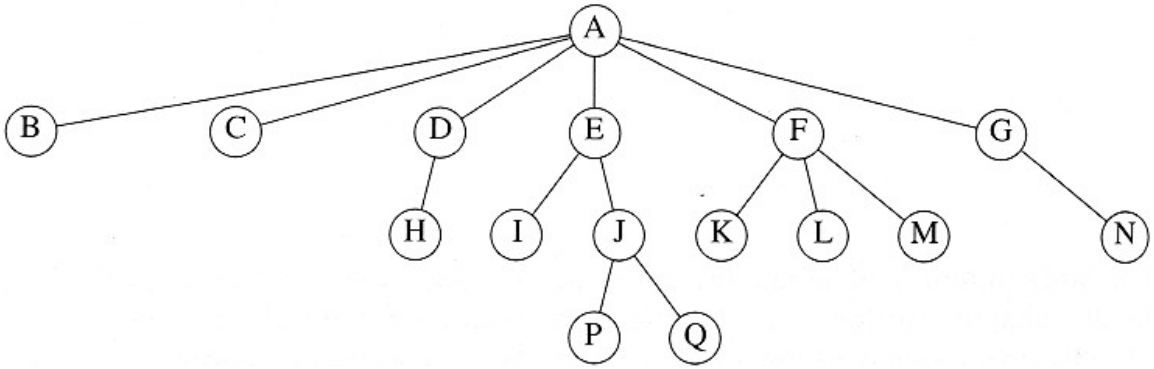
Stabla - primjeri

- File system

- Računanje izraza:
  a= (b+c)*d;

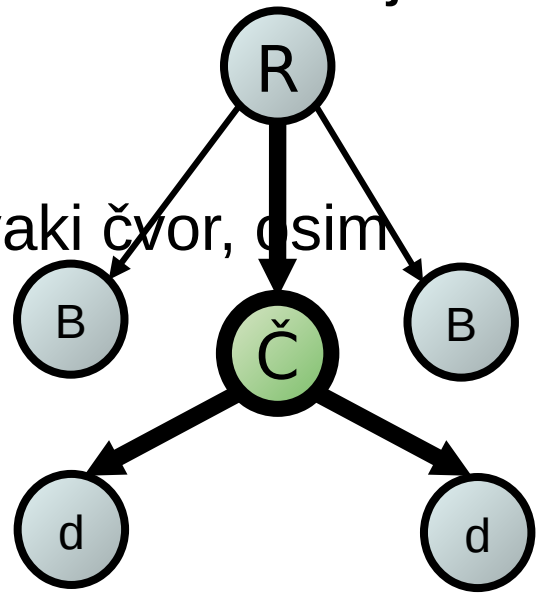






Terminologija

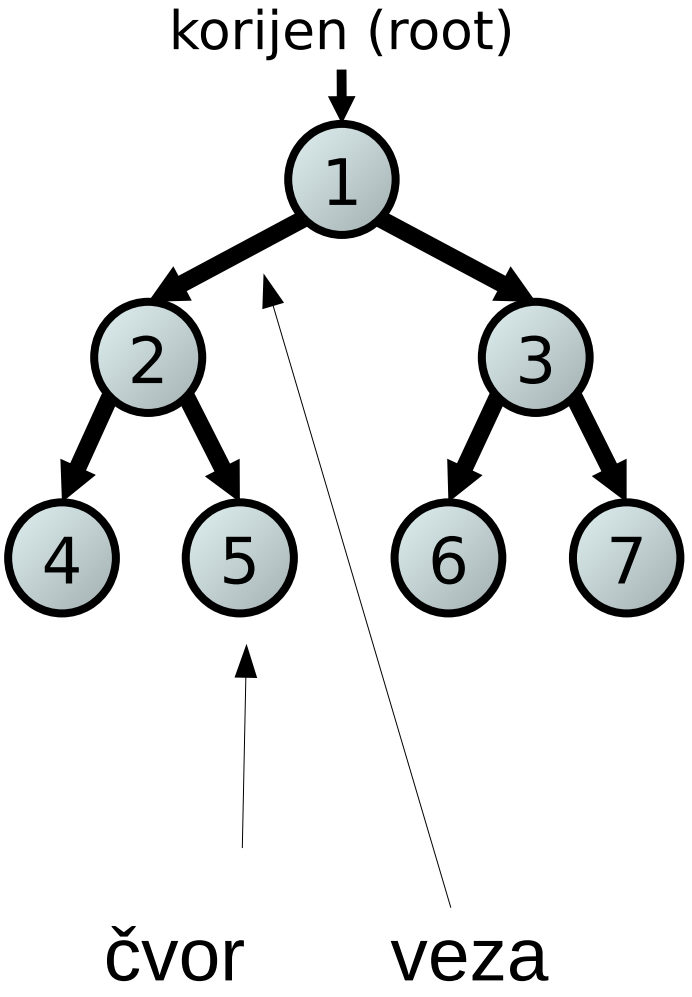
- **čvor (node)**:
  - objekat koji sadrži podatak (element) i veze do svoje djece (čvorova)
- **roditelj (parent)**:
  - čvor koji ima vezu do trenutnog čvora. Svaki čvor, osim korijena, ima tačno jednog roditelja.
- **dijete (child)**:
  - čvor na koji trenutni čvor ima vezu
- **brat (sibling)**:
  - čvor koji ima zajedničkog roditelja sa trenutnim čvorom
- **korijen (root)**: čvor na vrhu (dnu) stabla. Nema roditelja.
- **list (leaf)**: čvor koji nema djece.
- **grana (branch)**:
  - svaki unutrašnji čvor koji nije niti korijen niti list.



Terminologija

- **čvor (node)**: svi kružići na slici desno
- **korijen (root)**: 1
- **listovi (leaves)**: 4, 5, 6, 7
- **grane (branches)**: 2, 3

- Posmatrajmo čvor 2
  - **roditelj (parent): 1**
  - **djeca (children): 4, 5**
  - **brat (sibling): 3**

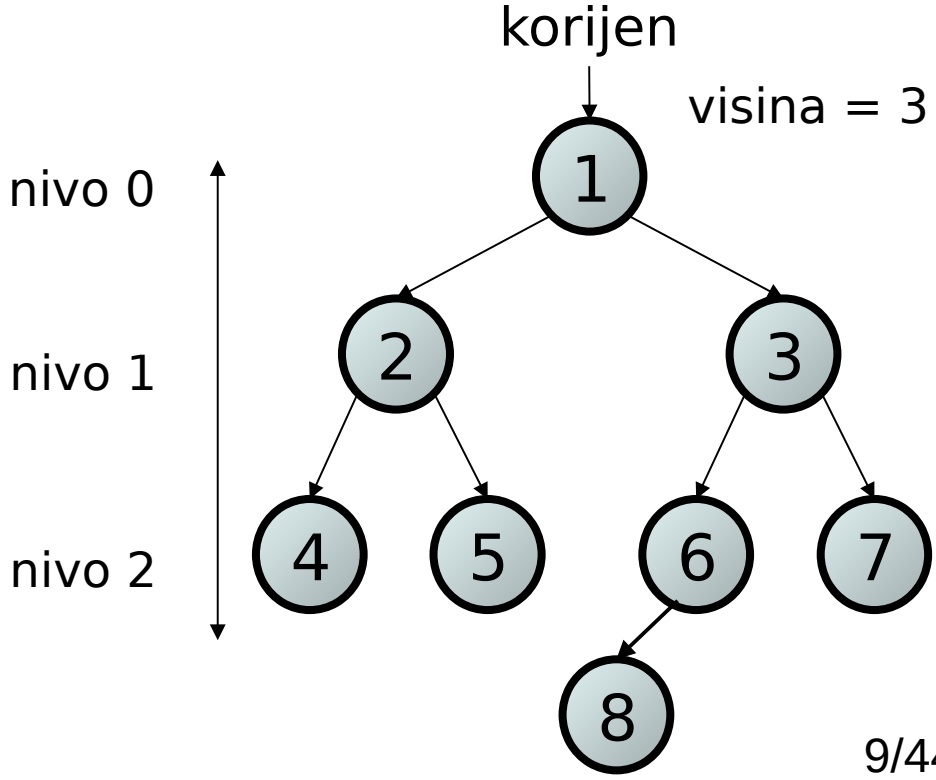


Terminologija

- **podstablo (subtree)**: stablo sa svim čvorovima do kojih se dolazi preko jedne veze od trenutnog čvora. Svaki čvor se može smatrati korijenom podstabla kome pripadaju njegova djeca, unuci itd.

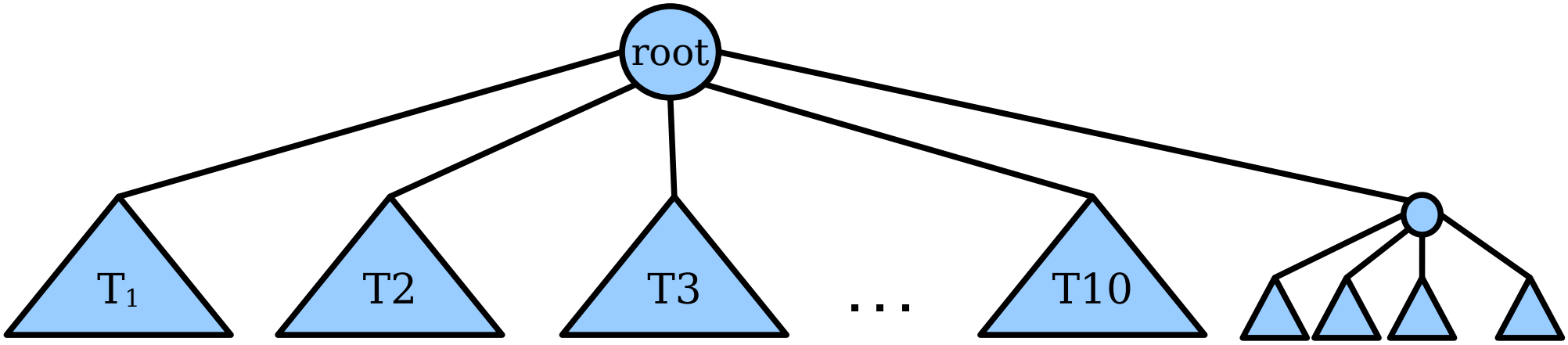
- **visina stabla (height)**: broj čvorova na najdužem putu od korijena do najudaljenijeg lista (ne računajući korijen)

- **nivo ili dubina čvora (level)**: broj grana na putu od datog čvora do korijena

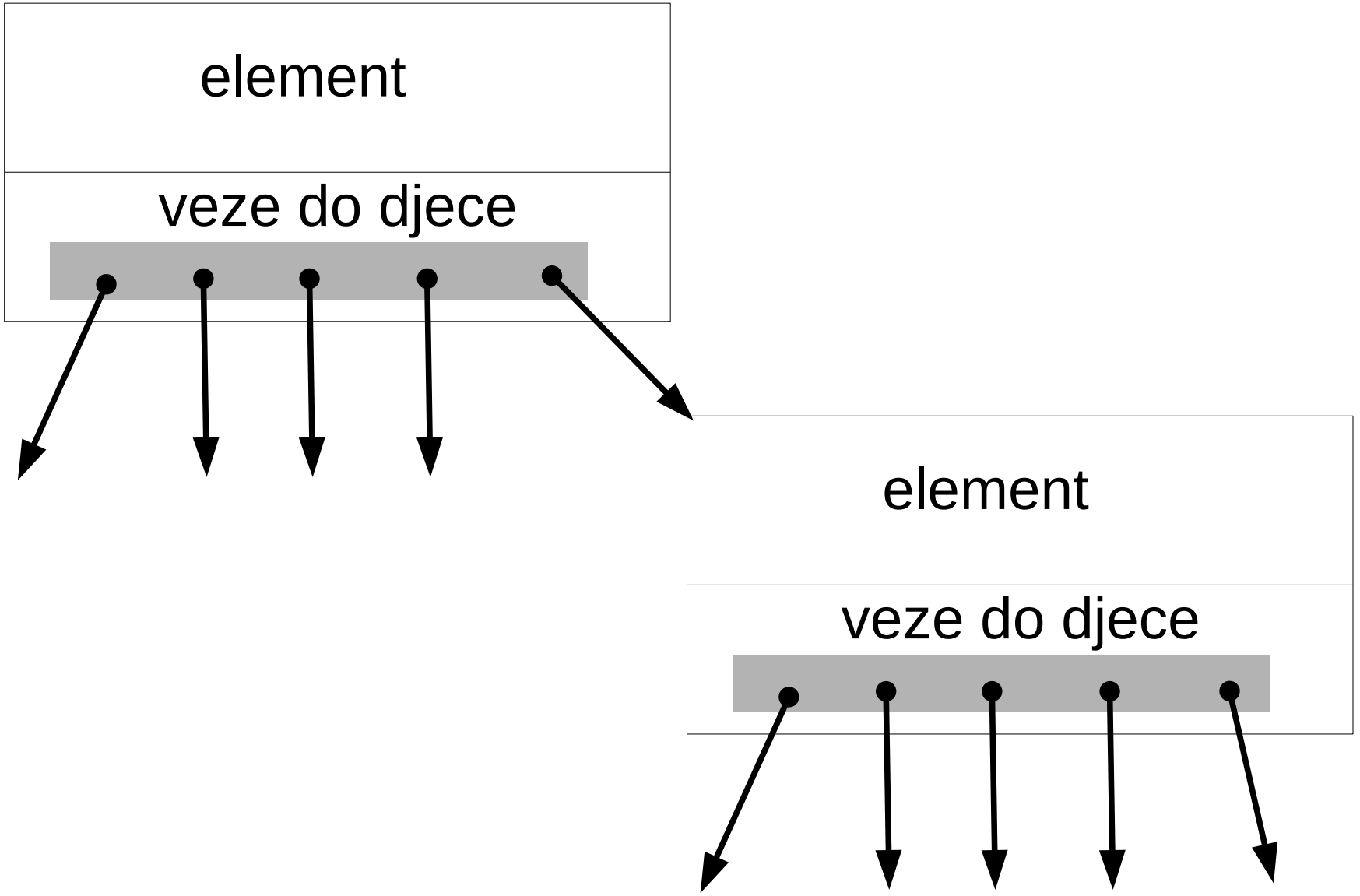


Još jedna definicija stabla (rekurzivna)

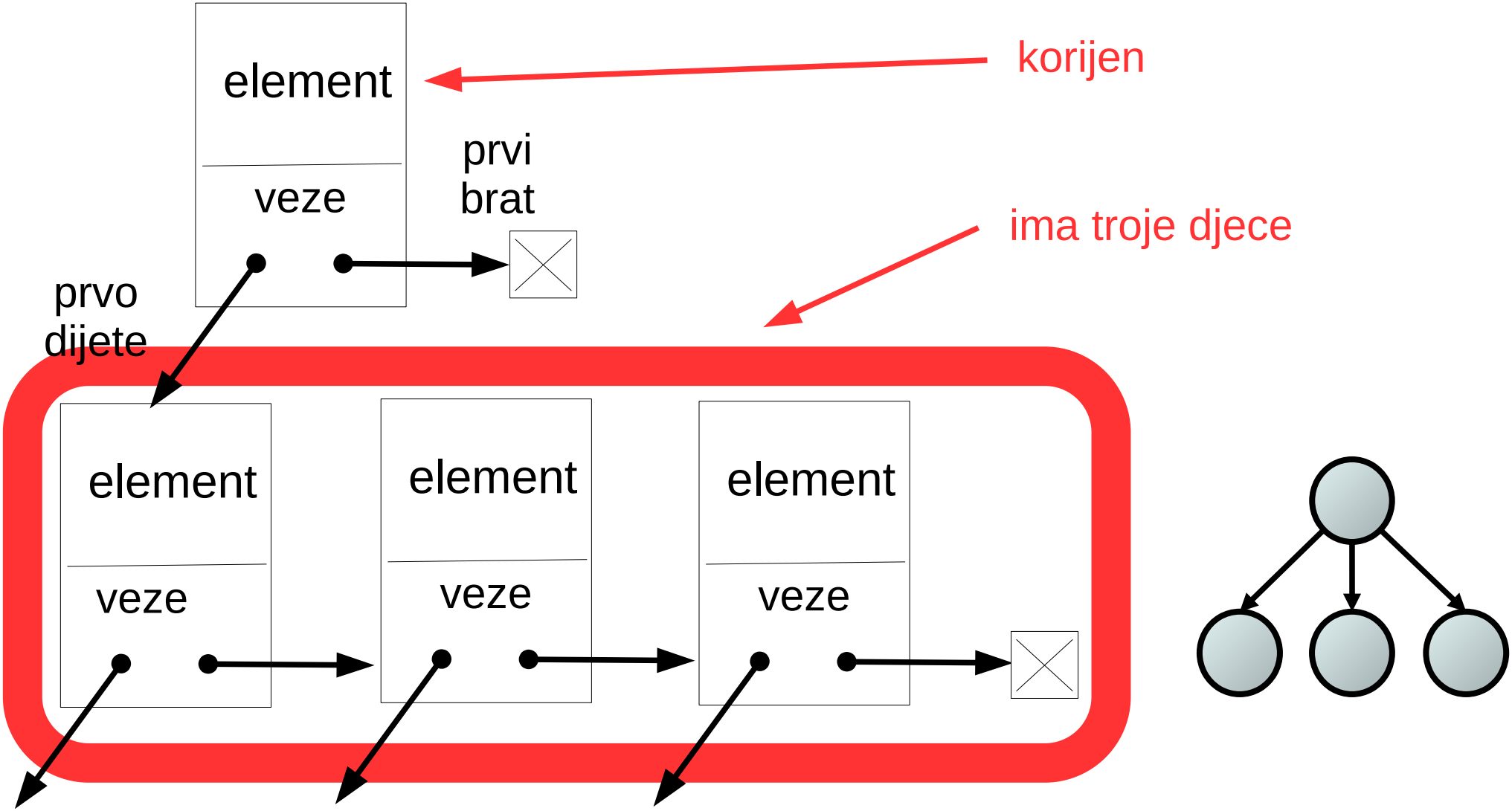
1. Stablo je kolekcija čvorova
2. Stablo može biti prazno
3. Ako nije prazno, stablo se sastoji od jednog čvora (korijen, root) i nula ili više podstabala na čije korijene su spojene veze od korijena trenutnog stabla.



Stablo – moguća implementacija

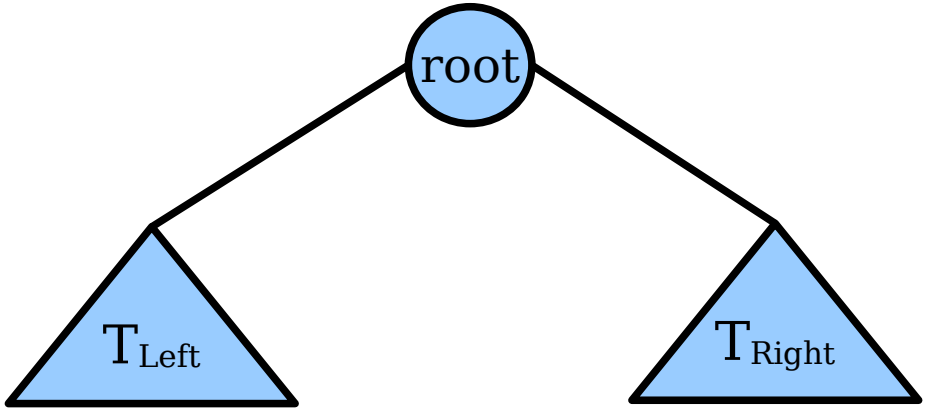


Stablo – moguća implementacija 2

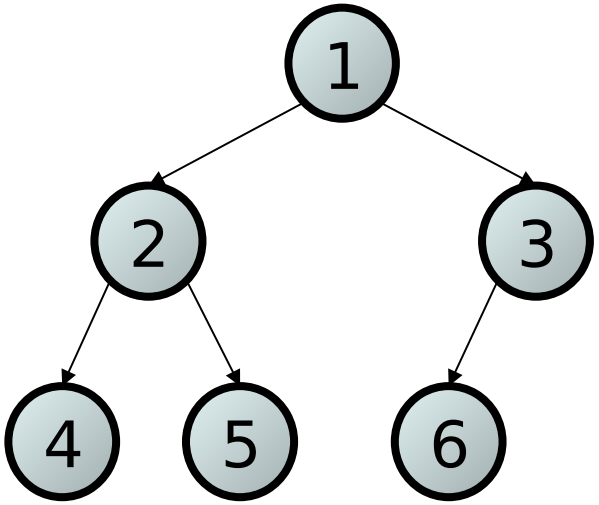


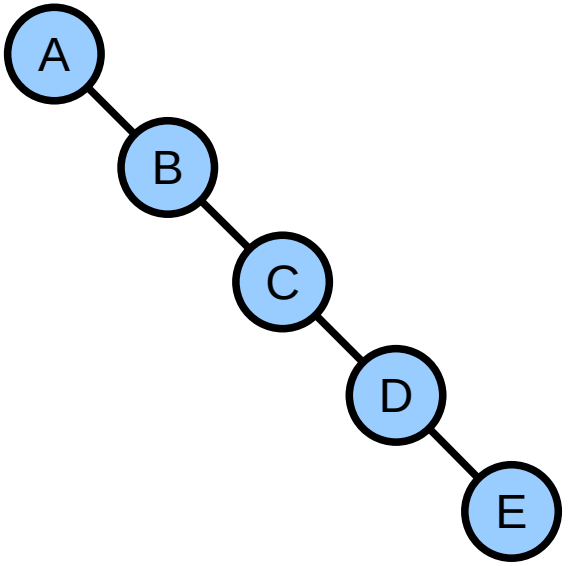
Binarno stablo (Binary tree)

- Stablo kod kojeg čvor može imati najviše 2 djece



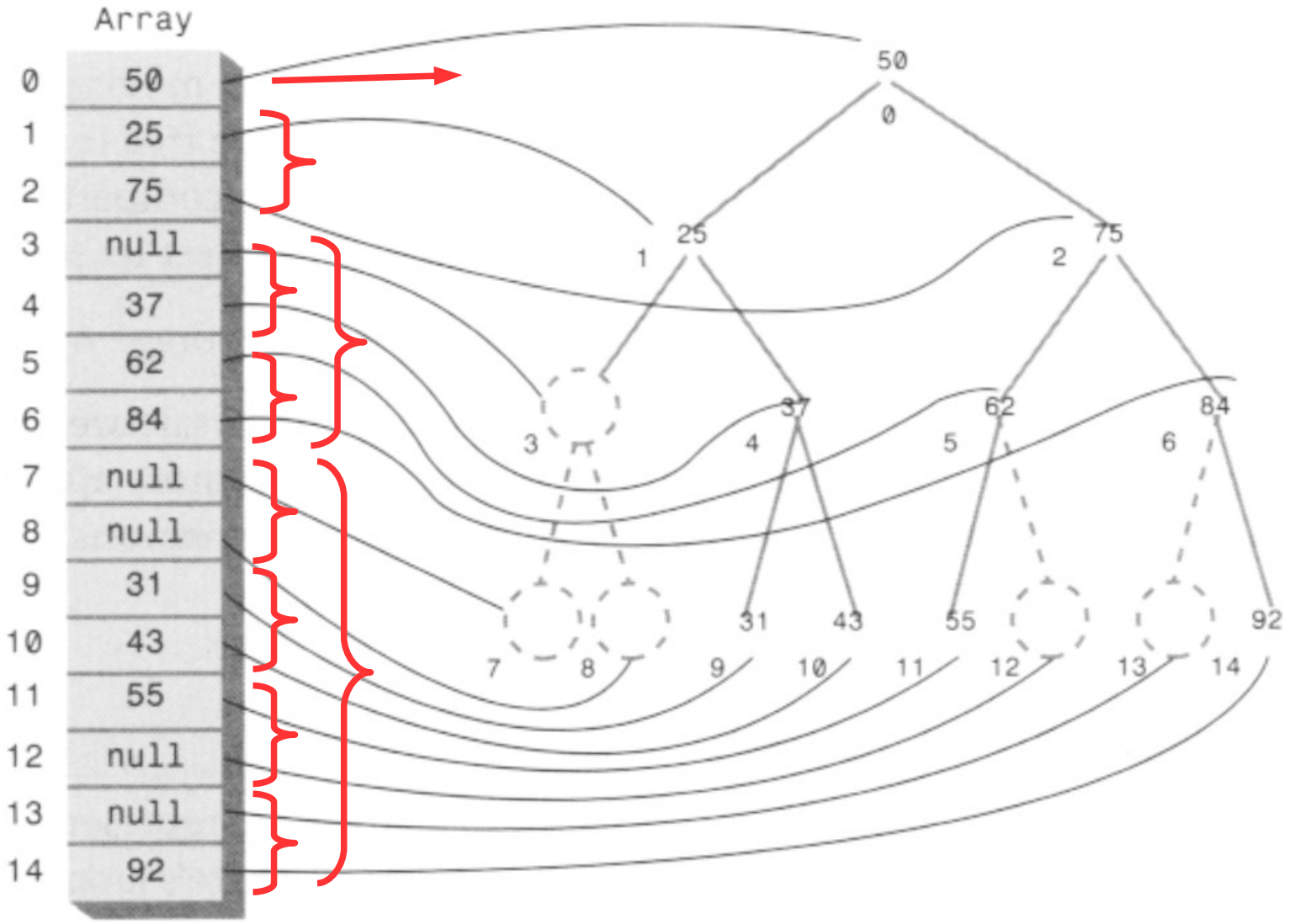
- Visina prosječnog binarnog stabla je mnogo manja od N (broja čvorova), mada u najgorem slučaju može iznositi N-1





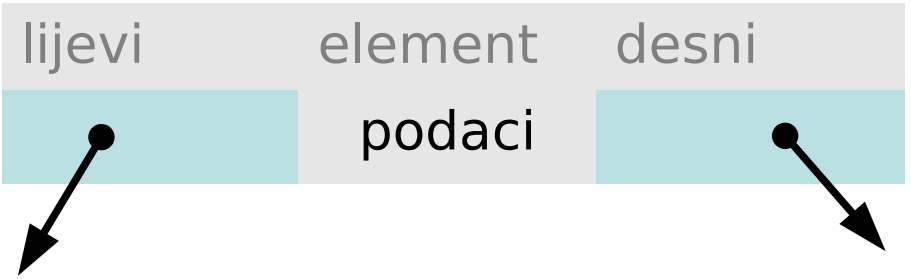
Binarno stablo - implementacija

- Binarno stablo je moguće implementirati pomoću niza:



Binarno stablo - implementacija

- Binarna stabla se uglavnom implementiraju pomoću povezanih čvorova. Takav čvor ima element u koji se smješta podatak i dvije veze (pokazivači) na dvoje svoje djece (lijevo, desno).

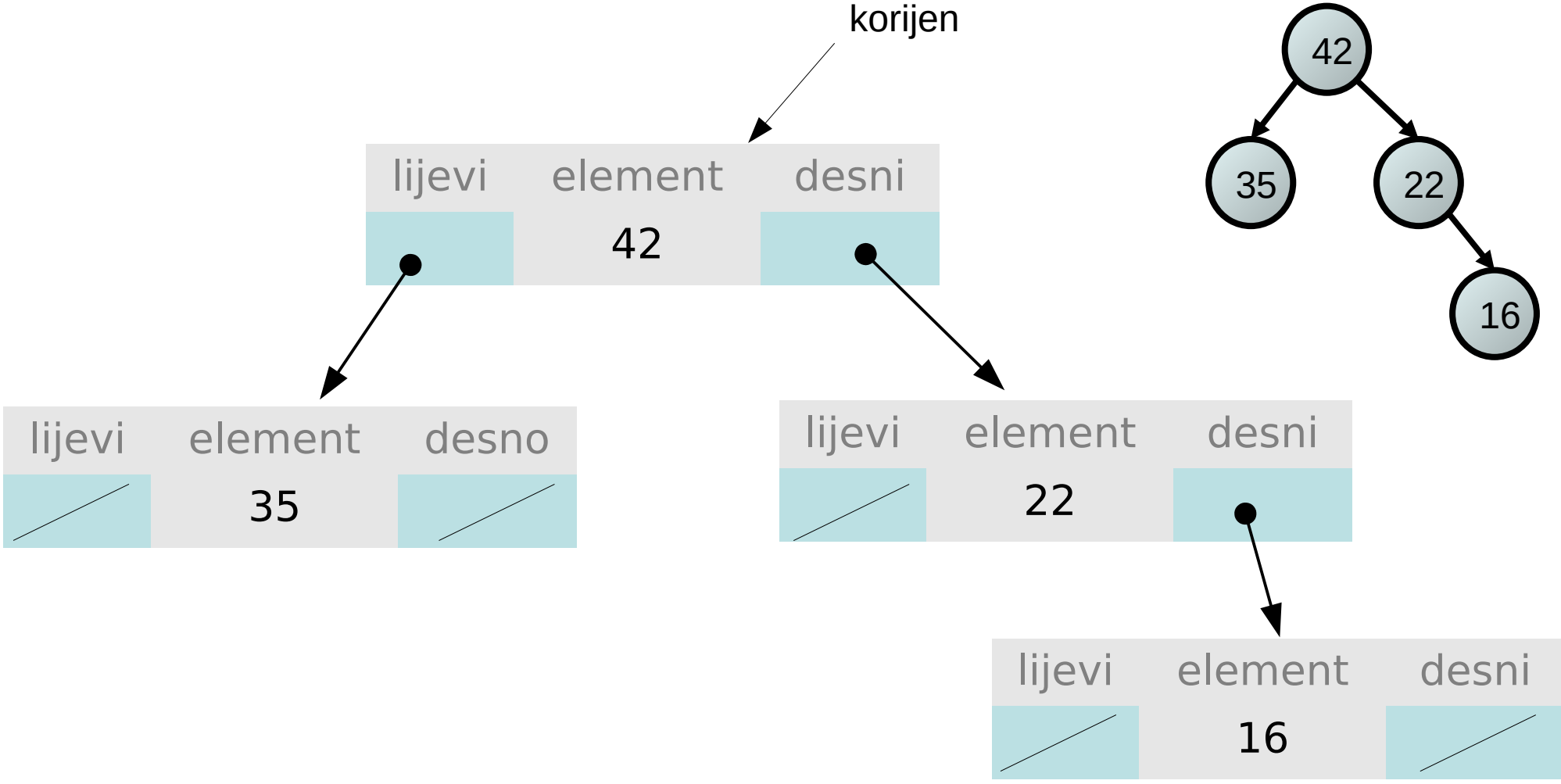



```
struct CvorBinStabla
{
  ElemTip element;
  CvorBinStabla * lijevi;
  CvorBinStabla * desni;
};
```

```
struct BinarnoStablo
{
  CvorBinStabla * korijen;
};
```

Binarno stablo - primjer

- Više čvorova se veže u stablo na sljedeći način:



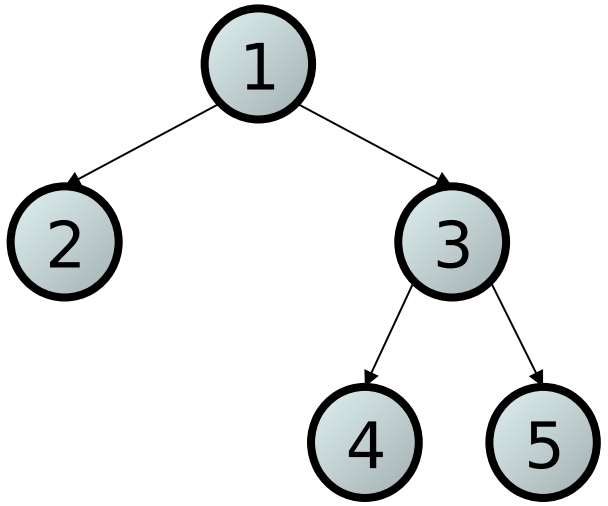
Binarno stablo – generička implementacija: Čvor binarnog stabla

```
template <typename ElemTip>
struct CvorBinStabla
{
    ElemTip element;
    CvorBinStabla * lijevi, * desni;

    CvorBinStabla(const ElemTip & v)
    : element(v), lijevi(nullptr), desni(nullptr)
    {}

    // ostali metodi
};
```

Primjer uvezivanja čvorova u binarno stablo




```
CvorBinStabla<int> a(1);
CvorBinStabla<int> b(2);
a.lijevi = &b;

CvorBinStabla<int> c(3);
a.desni = &c;

CvorBinStabla<int> d(4);
CvorBinStabla<int> e(5);
c.lijevi = &d;
c.desni = &e;
```

```
template <typename ElemTip>
struct CvorBinStabla
{
    ElemTip element;
    CvorBinStabla * lijevi, * desni;

    CvorBinStabla(const ElemTip & v)
      : element(v), lijevi(nullptr), desni(nullptr) {}
};
```

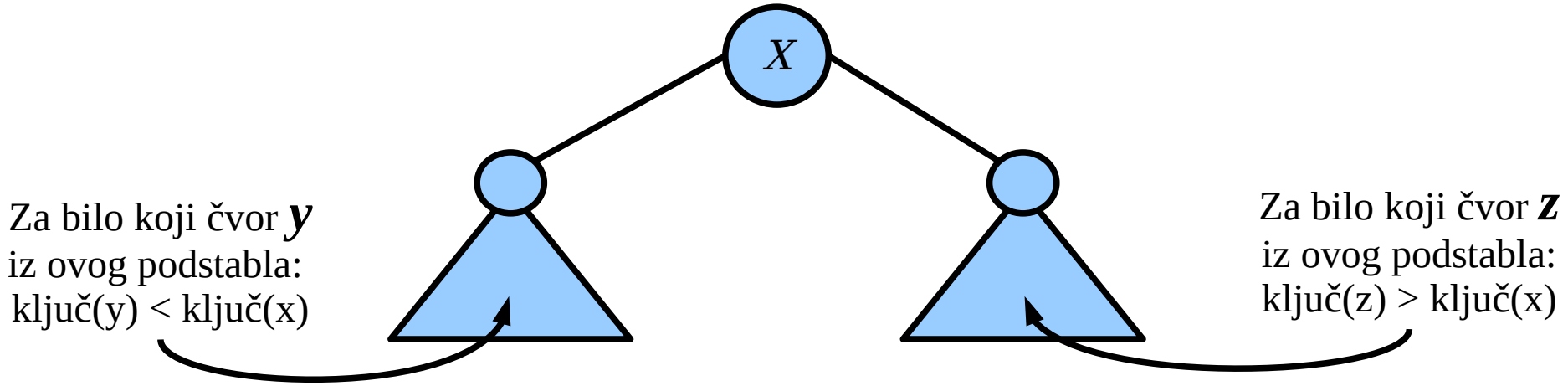
Operacije nad binarnim stablom

Osnovne operacije:

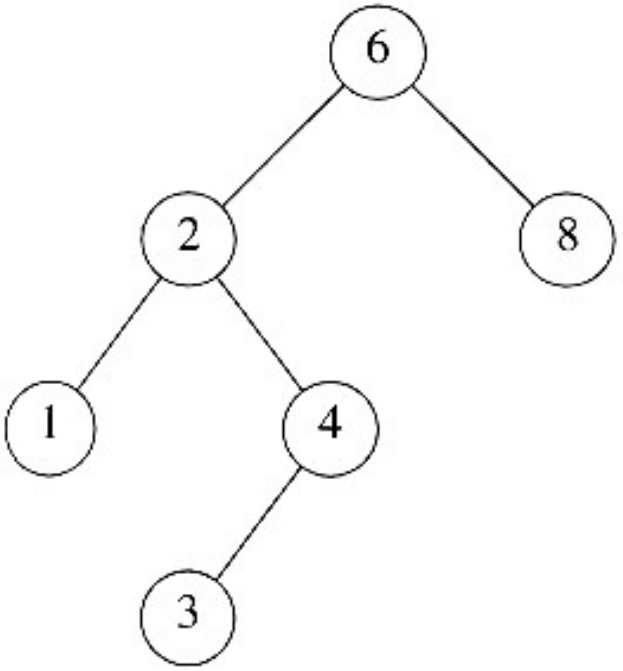
- Ubacivanje novog elementa (čvora)
- Uklanjanje postojećeg elementa (čvora)
- Traženje čvora sa datom vrijednošću elementa
- Obilazak stabla (prolazak kroz stablo, npr. ispis svih elemenata)

Uređena binarna stabla:
Stablo binarne pretrage – (binary search tree - BST)

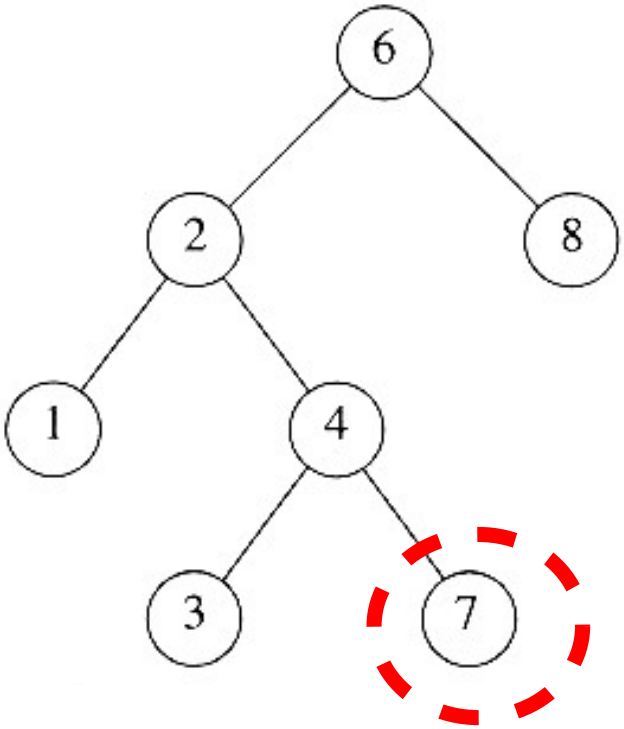
- Kod ovog stabla su vrijednosti elemenata unutar čvorova poredani tako da se operacije ubacivanja, uklanjanja i pretraživanja mogu izvoditi efikasno.
- **Ključ**: dio elementa (ili čvora) koji je kriterij pretraživanja
- Za svaki čvor X, svi ključevi u lijevom podstablu su manji (<) od vrijednosti ključa čvora X, a svi ključevi u desnom podstablu su veći (>) od ključa u



Primjer: koje stablo je binary search tree - BST



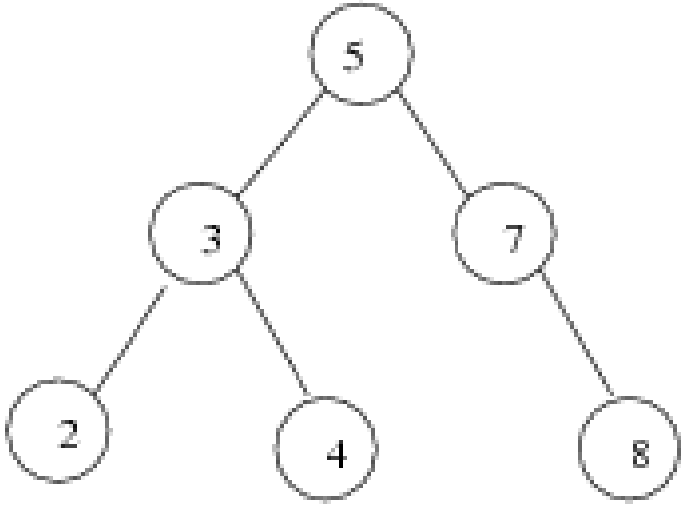
BST STABLO

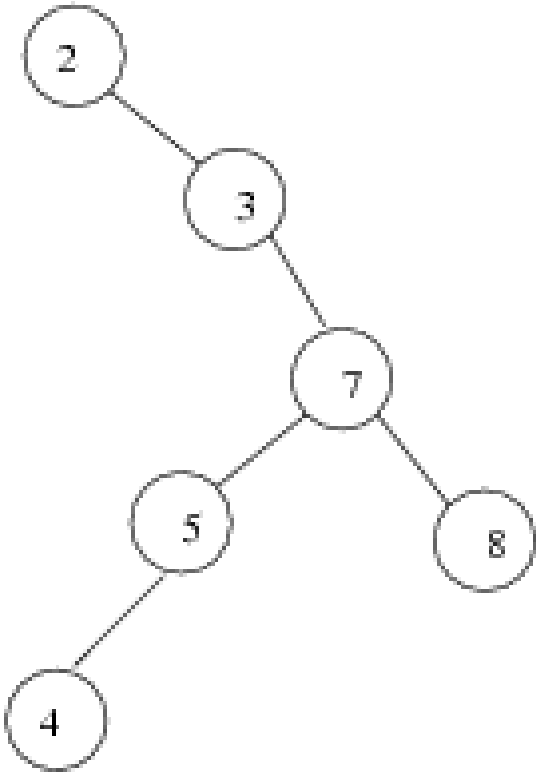


NIJE BST STABLO
jer je 7 > 6 a nalazi se u lijevom podstablu čvora 6

Primjeri: binary search tree - BST

Dva stabla koja sadrže iste podatke

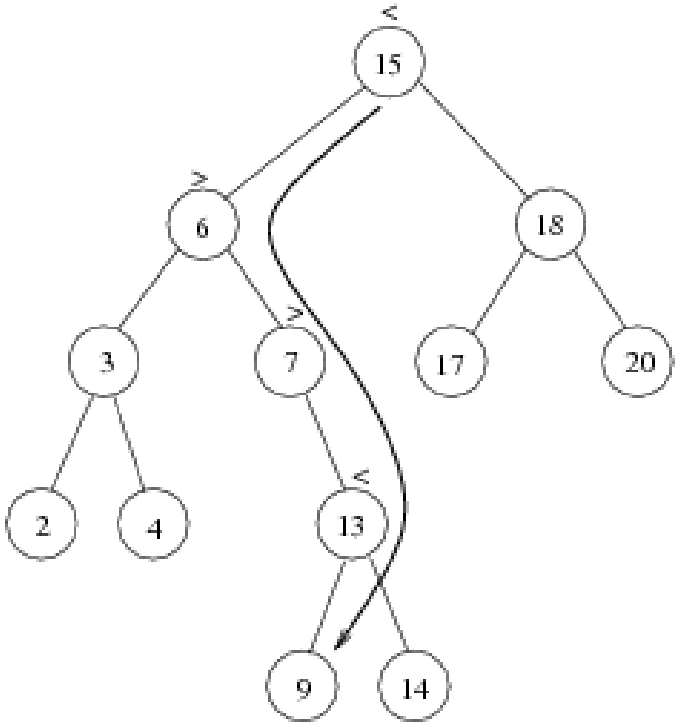




Prosječna dubina čvora je O(logN).
Najveća dubina čvora, u najgorem slučaju, je O(N).

Pretraživanje u binarnom BST stablu

Primjer traženja čvora sa vrijednošću 9:



Tražimo 9:

1. Krećemo od korijena
2. Poredimo 9:15 (korijen) => idi u lijevo podstablo
3. Poredimo 9:6 => idi u desno podstablo
4. Poredimo 9:7 => idi u desno podstablo
5. Poredimo 9:13 => idi u lijevo podstablo
6. Poredimo 9:9 => PRONAĐENO!

Ovo je rekurzivan proces.

Složenost: $O(visina\ stabla)$

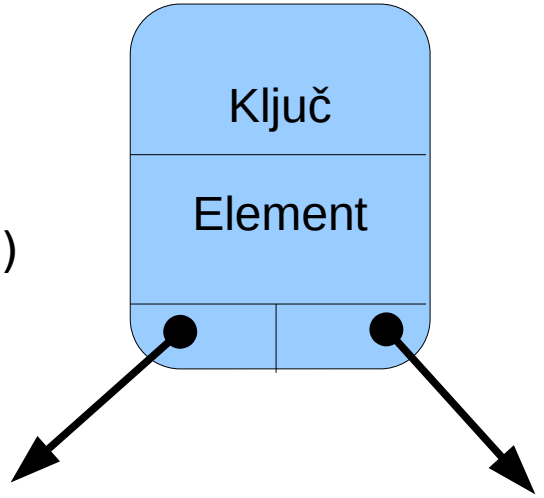
Binarno stablo – generička implementacija: Čvor BST stabla

- Da bismo mogli vršiti pretraživanje (razlikovanje) objekata proizvoljnog tipa po nekom ključu možemo uvesti ključ kao poseban član čvora.
- Tada uvijek tražimo po ključu.

```
template <typename KljucTip, typename ElemTip>
struct CvorBST
{
  KljucTip kljuc;
  ElemTip element;
  CvorBST * lijevi, * desni;

  CvorBST(const KljucTip & k, const ElemTip & v)
   : kljuc(k), element(v),
     lijevi(nullptr), desni(nullptr){}

  // ostali metodi
  bool postoji(const KljucTip &) const;
};
```



Pretraživanje u binarnom BST stablu

```
template <typename KljucTip, typename ElemTip>
bool CvorBST<KljucTip, ElemTip>::postoji(KljucTip const & k) const
{
  if(k == kljuc)
    return true;
  else if(k < kljuc && lijevi != nullptr)
    return lijevi->postoji(k);
  else if(kljuc < k && desni != nullptr)
    return desni->postoji(k);
  else
    return false;
}
```

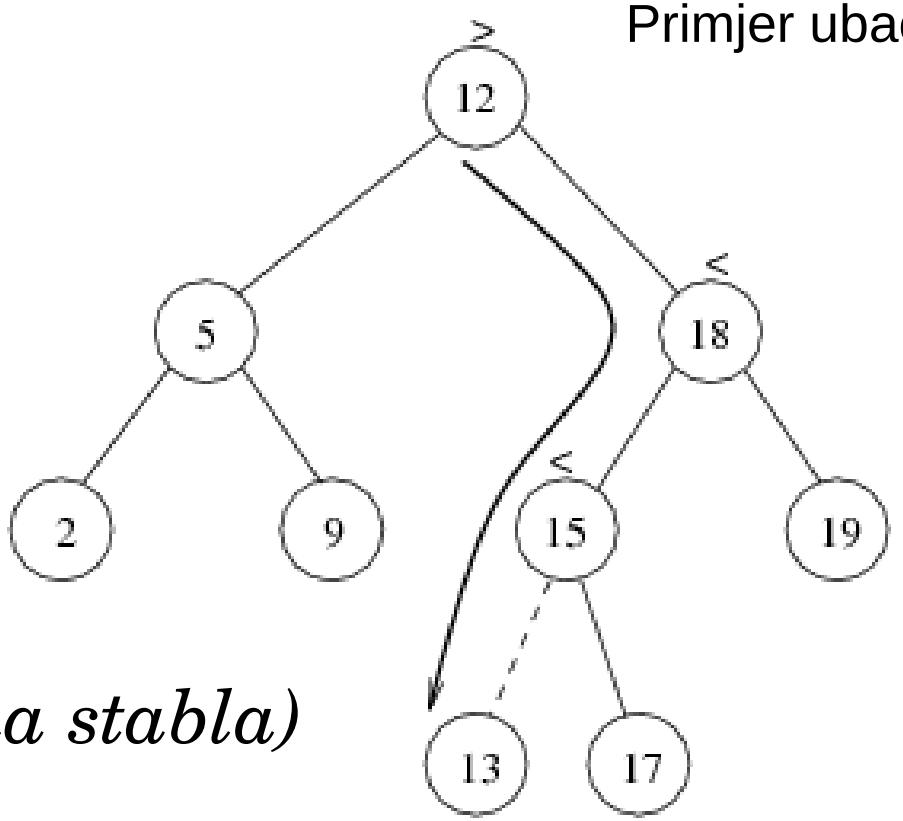
Bez rekurzije

```
{
 CvorBST const * tmp = this;
 while(tmp != nullptr)
 {
   if(tmp->kljuc == k)
     return true;
   if(k < tmp->kljuc)
     tmp = tmp->lijevi;
   else
     tmp = tmp->desni;
 }
 return false;
}
```

Ubacivanje čvora u BST

- Proći kroz stablo kao kod pretraživanja
- Ako je nađen X ne raditi ništa (ili treba izvršiti ažuriranje, ili generisati grešku, zavisno od dizajna)
- Ako X nije nađen, ubaciti X na zadnjem čvoru pretrage (ondje gdje bi X trebao biti).

Primjer ubacivanja broja 13



Složenost: *O(visina stabla)*

BST – uklanjanje čvora

- Uklanjanje čvora je složenije od dodavanja.
- I ovdje najprije moramo pronaći čvor koji uklanjamo krećući od korijena stabla.
- Pri uklanjanju moramo voditi računa kako tretiramo djecu čvora koji se uklanja:
    - BST mora zadržati svoje osobine brzog pretraživanja.

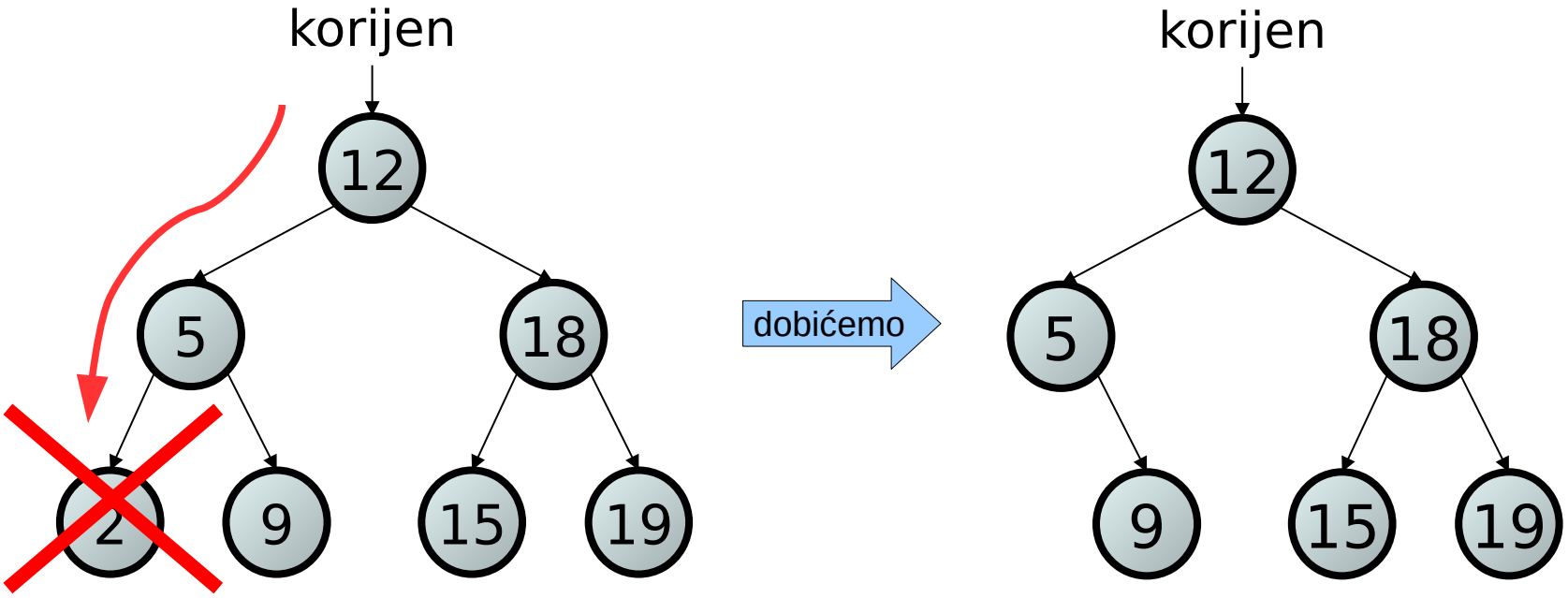
- Pri uklanjanju čvora mogu nastati tri slučaja:

1. čvor koji se uklanja je list (nema djece)
2. čvor koji se uklanja ima jedno dijete
3. čvor koji se uklanja ima dvoje djece

Uklanjanje čvora – 0 djece

- Da bismo uklonili čvor koji je list, jednostavno ćemo ga dealocirati a odgovarajući pokazivač na njegovom roditelju ćemo postaviti na null vrijednost. Npr.:

```
stablo.ukloni(2);
```

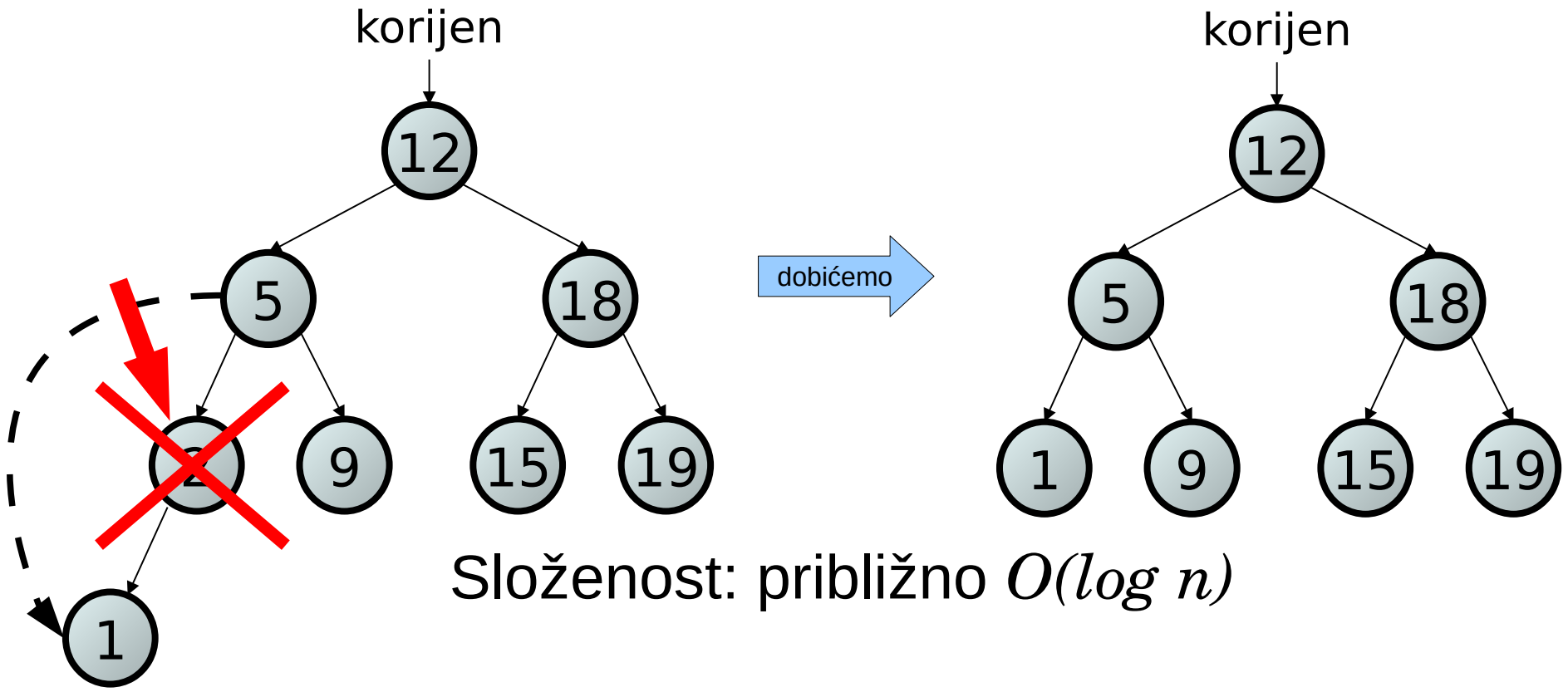


Složenost: *O(visina stabla), približno O(log n)*

Uklanjanje čvora - 1 dijete

- Da bismo uklonili čvor koji ima jedno dijete, pokazivač na njegovom roditelju, koji pokazuje na taj čvor ćemo postaviti tako da pokazuje na jedino dijete čvora kojeg uklanjamo
- Nakon toga jednostavno dealociramo dati čvor.

```
stablo.ukloni(2);
```



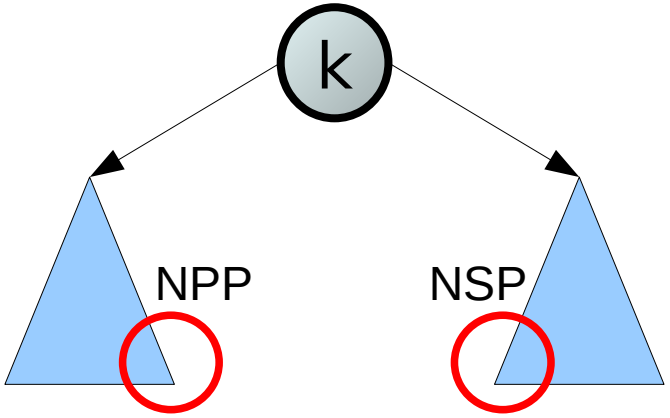
Složenost: približno $O(log\ n)$

Uklanjanje čvora – 2 djece

- Ovo je najsloženiji slučaj kod uklanjanja čvorova iz binarnog stabla jer:
  - Moramo voditi računa o tome da ne razdvojimo stablo na dva podstabla.
  - Moramo voditi računa o osobinama uređenog binarnog stabla (BST).
- Uklanjanje vršimo tako što pronađemo "*odgovarajući*" čvor koji ima 0 ili 1 dijete, zamijenimo vrijednosti elemenata (i ključeva) čvora kojeg želimo ukloniti sa tim čvorom i onda uklonimo taj drugi čvor.
- "*Odgovarajući*" u gornjem kontekstu je onaj koji ne remeti redoslijed BST-a, odnosno prethodnik ili sljedbenik (u poretku) čvora kojeg uklanjamo.

Uklanjanje čvora – 2 djece

- Def.: Neposredni prethodnik u poretku (NPP) nekog čvora je čvor u njegovom lijevom podstablu koji ima najveću vrijednost (*engl. inorder predecessor - IOP*)
  - to je "najdesniji" čvor u tom podstablu i sigurno ima 0 ili 1 dijete jer je skroz desno
- Def.: Neposredni sljedbenik u poretku (NSP) nekog čvora je čvor u njegovom desnom podstablu koji ima najmanju vrijednost (*engl. inorder successor – IOS*)
  - to je čvor skroz lijevo u tom podstablu i sigurno ima 0 ili 1 dijete



Uklanjanje čvora – 2 djece

Na osnovu prethodnog razmatranja algoritam za uklanjanje čvora sa 2 djece je sljedeći:

Prvi način:

- zamijeniti vrijednost (i ključ) datog čvora sa elementima čvora koji ima najmanji ključ u desnom podstablu
- izbrisati taj najmanji čvor u desnom podstablu (prema algoritmu za 0 ili 1 dijete)

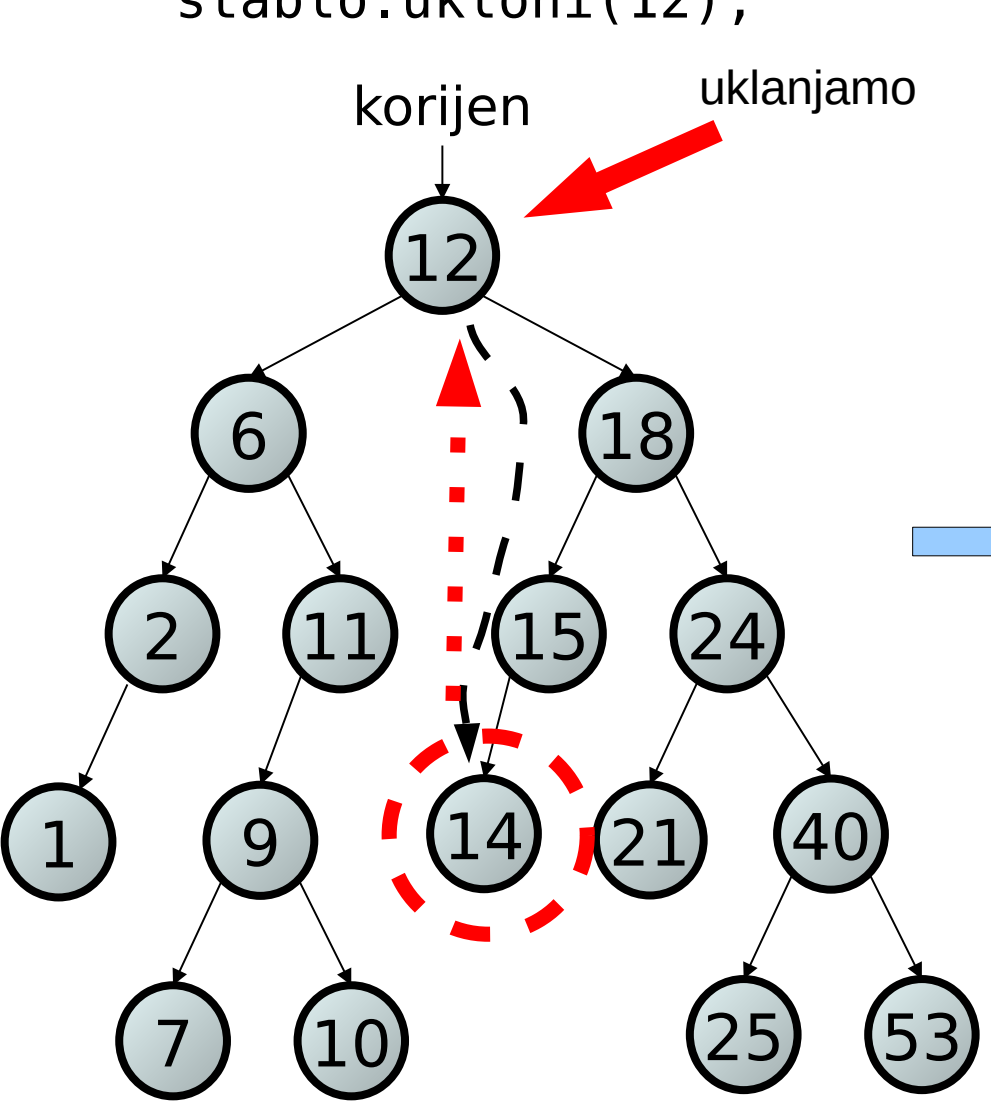
Drugi način:

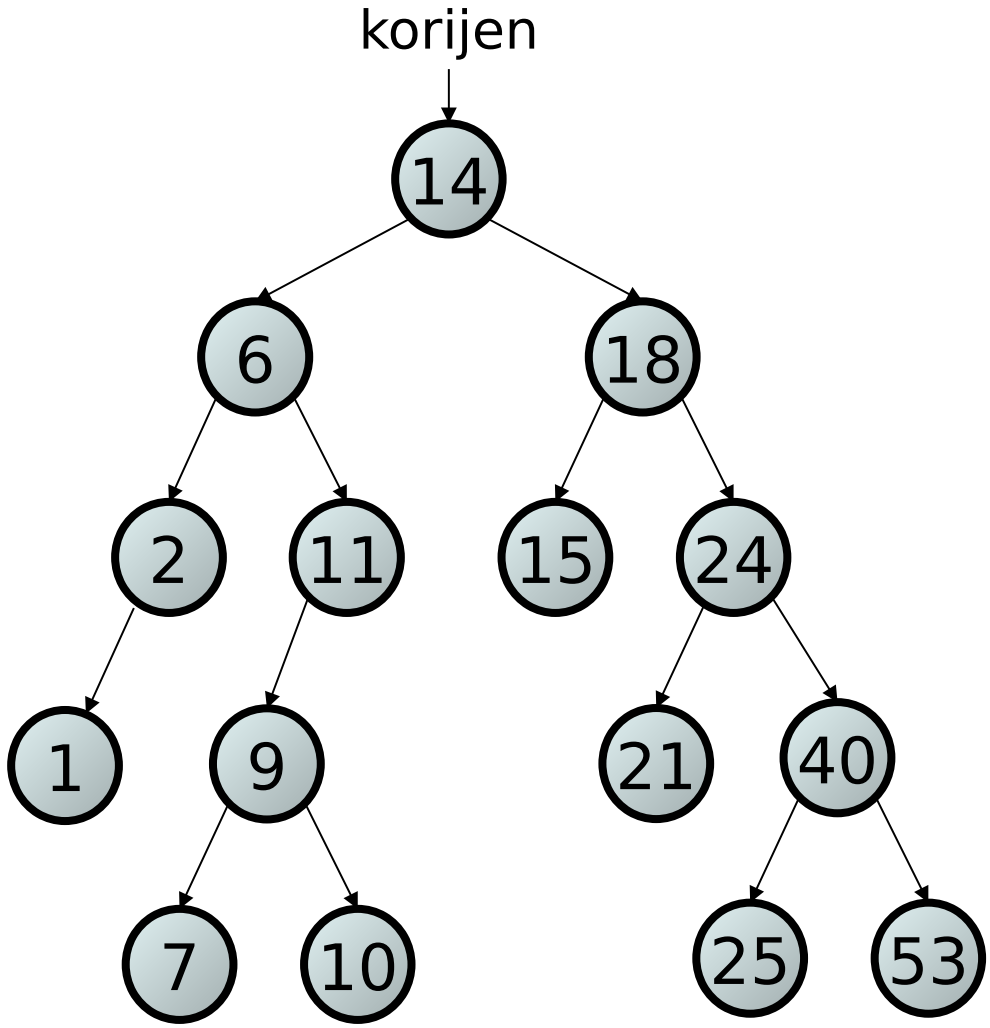
- zamijeniti vrijednost (i ključ) datog čvora sa elementima čvora koji ima najveći ključ u lijevom podstablu
- izbrisati taj najveći čvor u lijevom podstablu (prema algoritmu za 0 ili 1 dijete)

Složenost: $O(visina\ stabla)$

Primjer – uklanjanje pomoću NSP (sljedbenik)

```
stablo.ukloni(12);
```

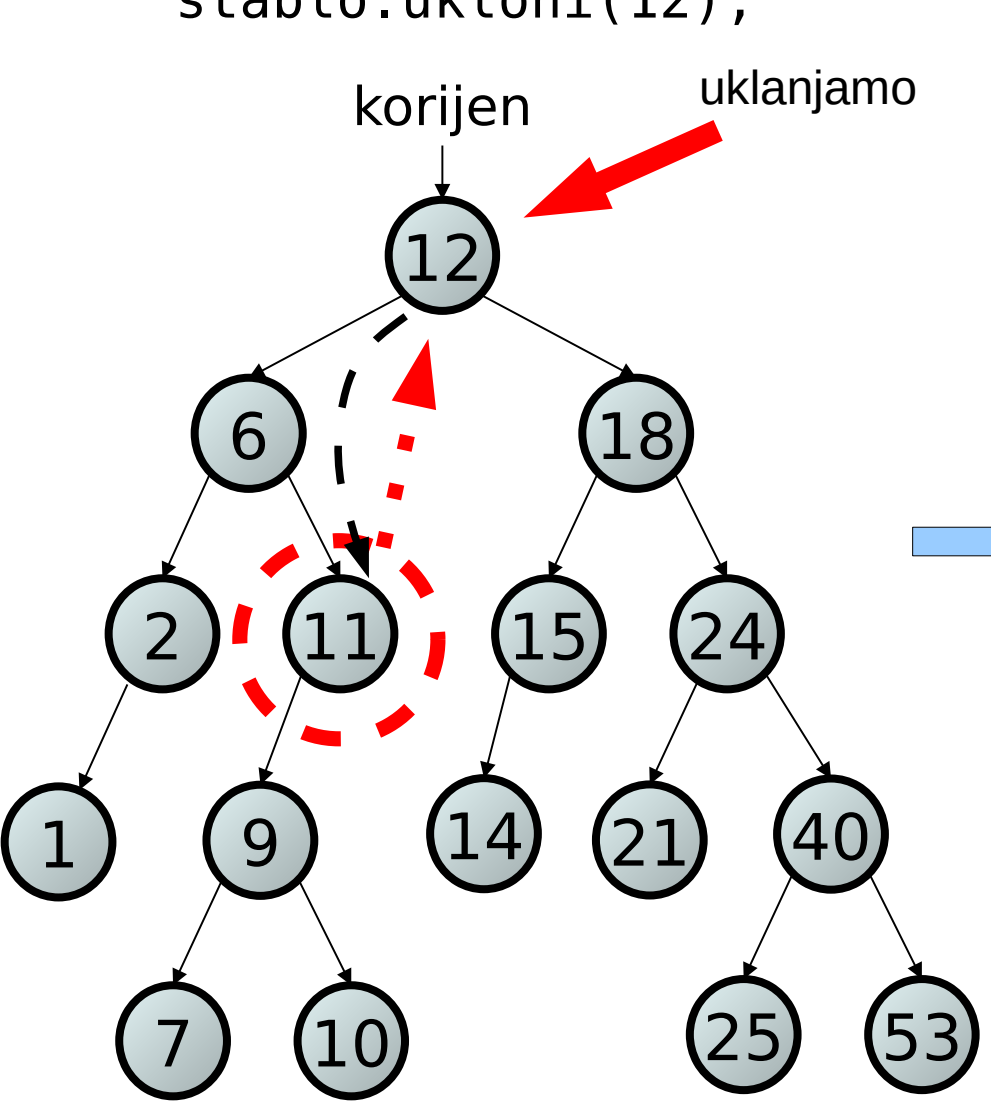


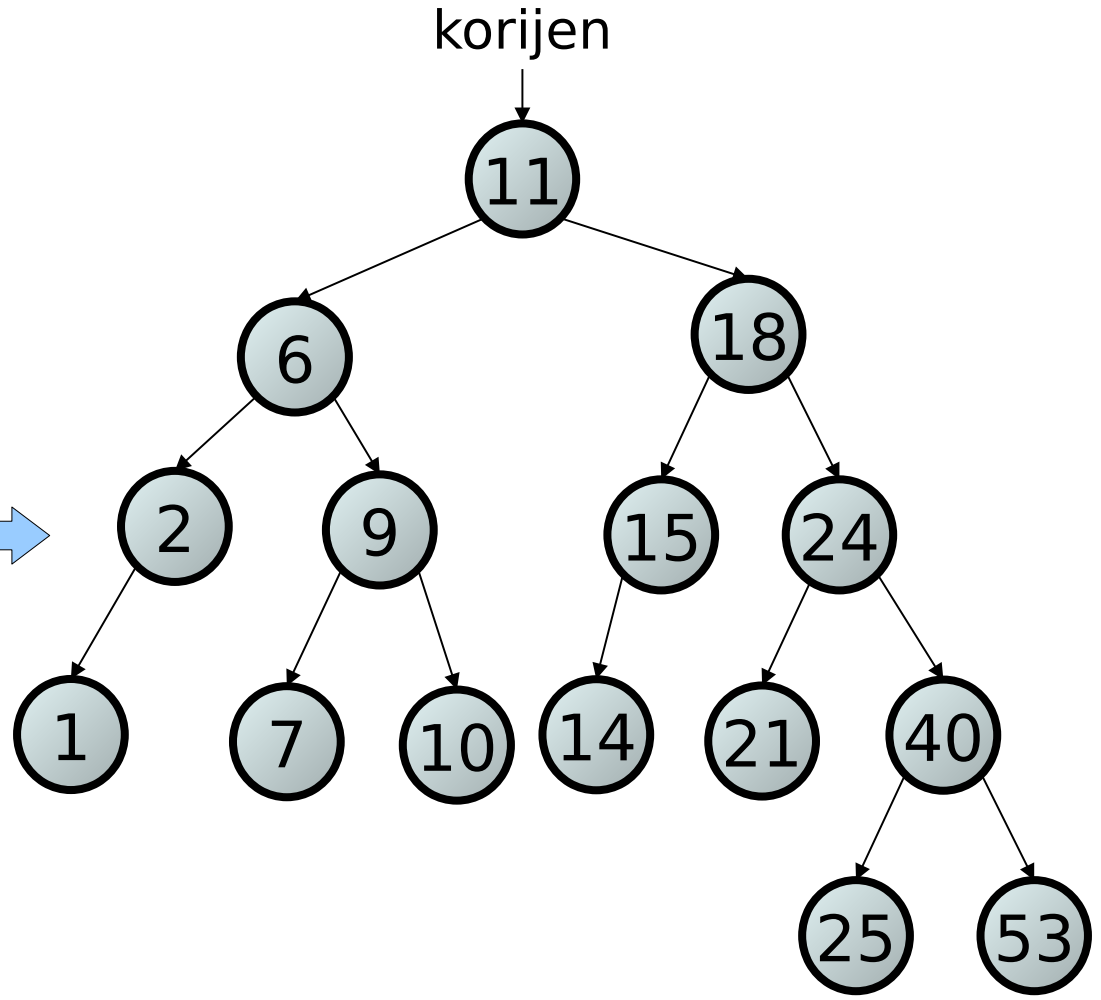


Složenost: $O(visina\ stabla)$

Primjer – uklanjanje pomoću NPP (prethodnik)

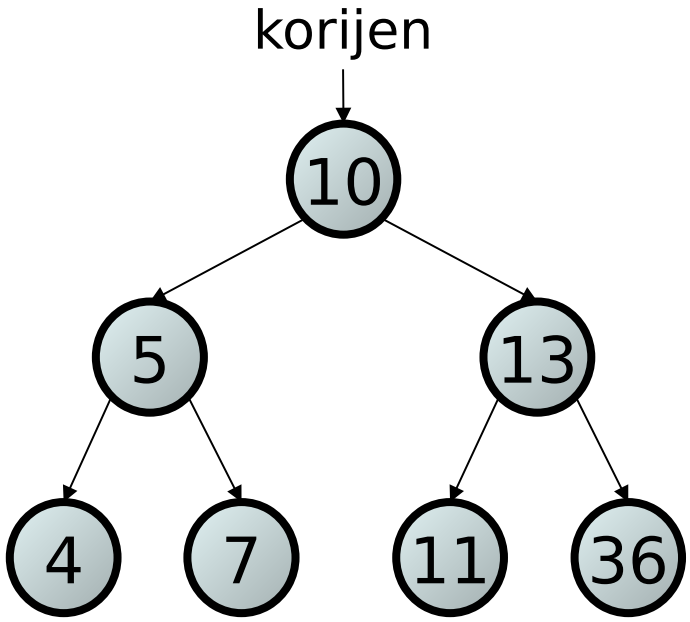
```
stablo.ukloni(12);
```





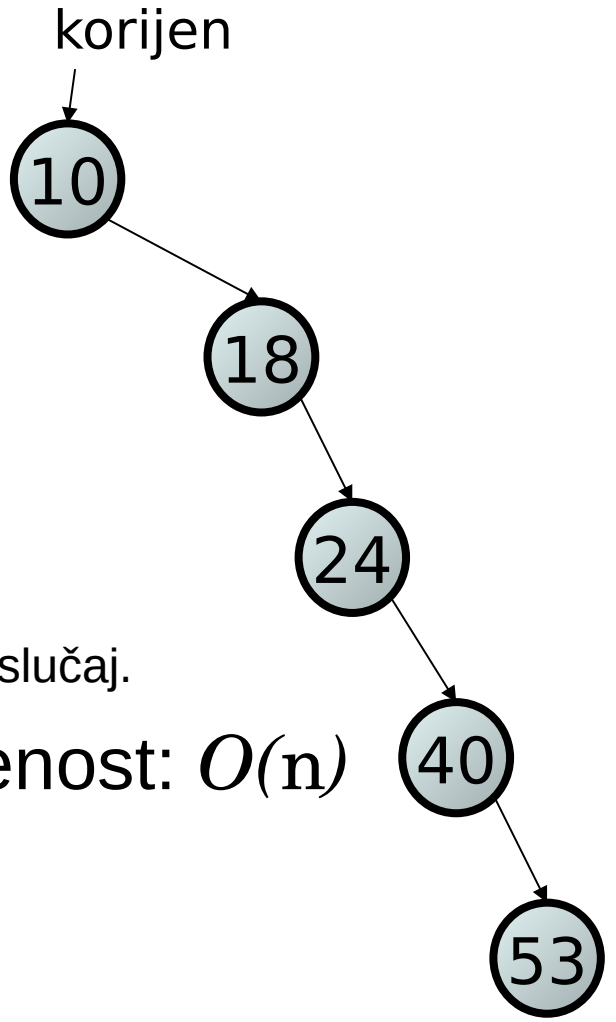
Složenost: $O(visina\ stabla)$

BST – složenost operacija pretraživanja, dodavanja i uklanjanja



Savršeno balansirano stablo

Složenost: $O(log\ n)$



Najgori slučaj.

Složenost: $O(n)$

Sve operacije u prosjeku nemaju veću složenost od $O(log\ n)$.

Ova struktura je korisna kad imamo često pretraživanje ili pristup po ključu.

BST – moguća implementacija

```
template <typename KljucTip, typename ElemTip>
class StabloBST
{
 private:
  CvorBST<KljucTip, ElemTip> * korijen;
 public:
  StabloBST() : korijen (nullptr) {}
  StabloBST& ubaci(const KljucTip & k, const ElemTip & v) {
   if(korijen != nullptr)
    ...
   else
    korijen = new CvorBST<KljucTip, ElemTip>(k, v);
   return *this;
  }
   ...
};
```

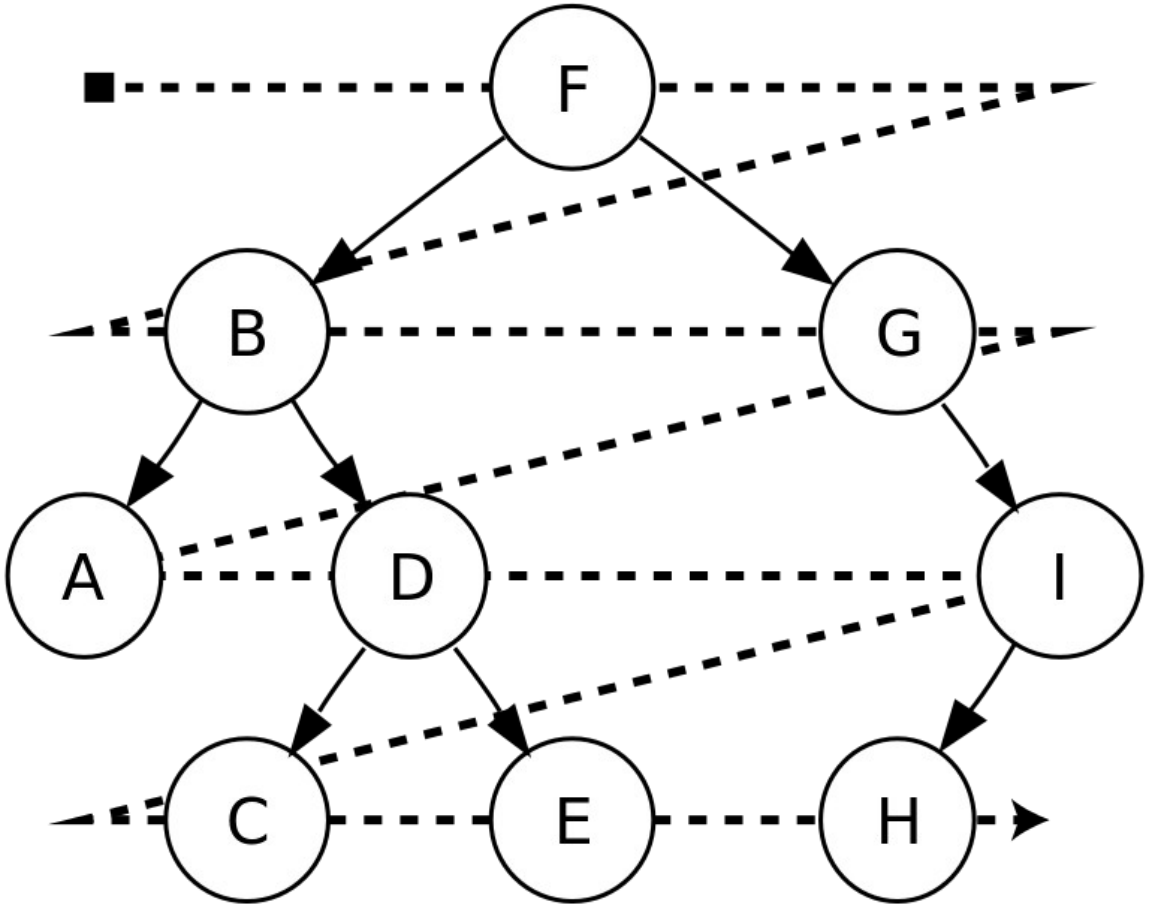
Prolazak kroz binarno stablo – obilazak stabla (engl. traversal)

- Prolazak kroz sve elemente stabla. Ovaj proces se koristi u mnogim algoritmima sa stablima.

- Dvija tipa obilaska stabla:
  - po širini, odnosno nivoima (*breadth first*)
  - po dubini (*depth first*)

Prolazak kroz binarno stablo – po nivoima (*breadth first*)

- Prolazak kroz sve čvorove stabla.
- Procesiramo sve čvorove na jednom nivou pa tek onda idemo na naredni nivo.



Redoslijed procesiranja:

F B G A D I C E H

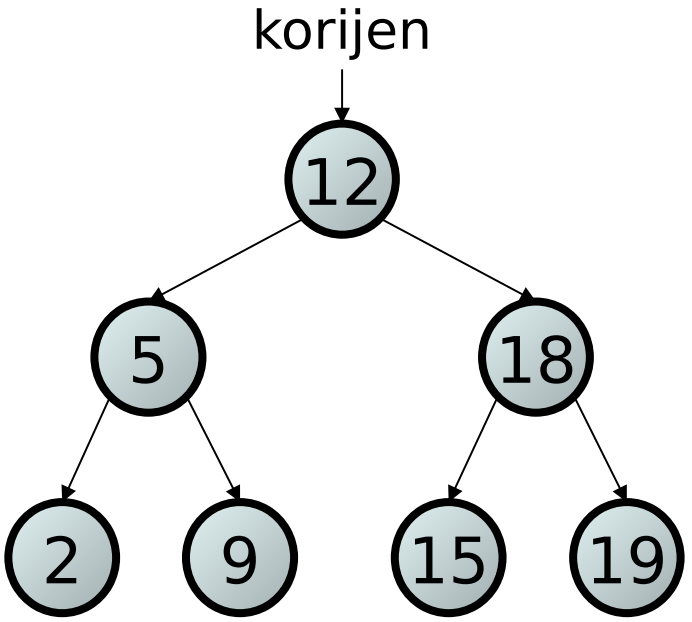
Prolazak kroz binarno stablo

– po dubini (*depth first*)

- Prolazak kroz sve čvorove stabla. Ovaj proces se koristi u mnogim algoritmima sa stablima.

- Najčešći načini prolaska kroz stablo (prema redoslijedu procesiranja podatka u čvoru):
- **pre-order:** (roditelj prije djece)
  - procesira se čvor pa onda lijevo pa desno podstablo tog čvora
- **in-order:** (roditelj između djece)
  - procesira se lijevo podstablo pa onda čvor pa desno podstablo
- **post-order:** (roditelj poslije djece)
  - procesira se lijevo pa desno podstablo pa onda čvor

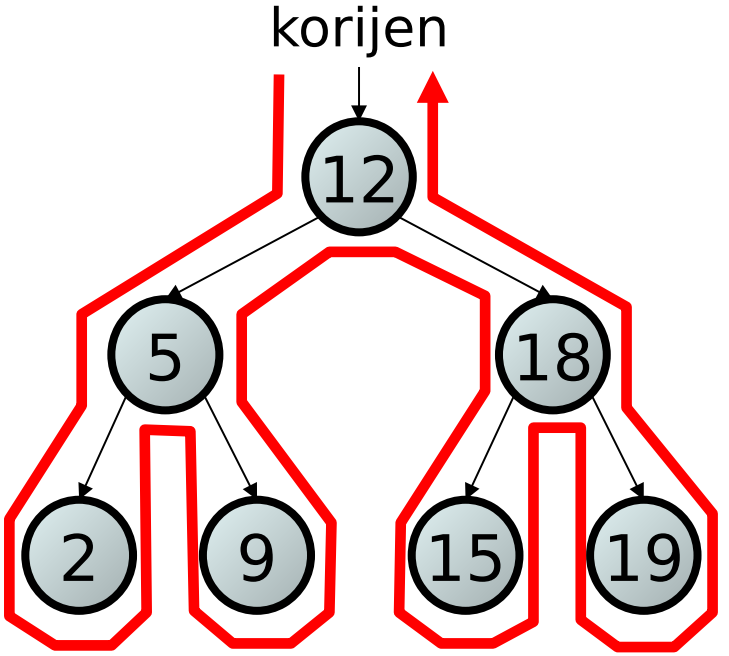
Prolazak kroz binarno stablo – po dubini (primjer)



- **pre-order:** 12 5 2 9 18 15 19
- **in-order:** 2 5 9 12 15 18 19
- **post-order:** 2 9 5 15 19 18 12

Prolazak kroz binarno stablo – "trik"

- Brzo generisanje rezultata prolaska kroz stablo:
  - Nacrtati putanju oko stabla.
  - Kad prođete pored čvora sa “prave strane”, procesirati ga.
    - pre-order: sa lijeve strane
    - in-order: odozdo
    - post-order: sa desne strane



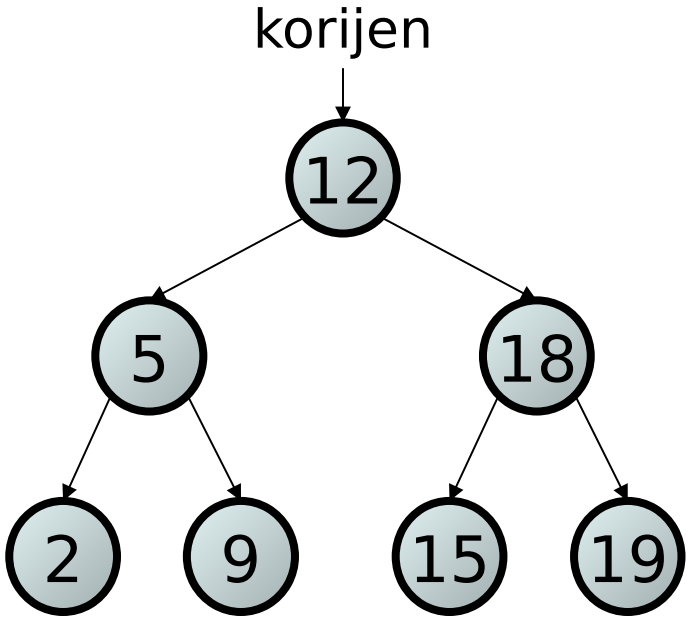
- **Pre-order:** 12 5 2 9 18 15 19
- **in-order:** 2 5 9 12 15 18 19
- **Post-order:** 2 9 5 15 19 18 12

Obilazak po dubini: moguća implementacija (rekurzivno)

```
template <typename KljucTip, typename ElemTip>
template <typename F>
void CvorBST<KljucTip, ElemTip>::inorder(F const & f)
{
  if(lijevi != nullptr)
   lijevi->inorder(f);

  f(kljuc, element);

  if(desni != nullptr)
   desni->inorder(f);
}
```



Složenost: $O(n)$

Binarno stablo: copy konstruktor

Pri realizaciji copy konstruktora za binarno stablo, u novokreiranom objektu čvorovi trebaju biti identično poredani kao u stablu koji se koristi kao argument ovog metoda. Slično važi i za operator = (dealokacija postojećih pa alokacija novih).

- Krenućemo od početka – korijena i kreirati novi čvor sa vrijednostima identičnim originalnom čvoru.
- Nakon toga to isto uradimo za djecu korijena.
- Za svako dijete korijena kopiramo njihovu djecu itd. za sve čvorove originalnog stabla.

Koji je najjednostavniji način da ovo uradimo? Koji prolazak?

- Očigledno, koristićemo **pre-order prolazak** kroz originalno stablo i kopiju čvora do kojeg smo došli dodavaćemo u novokreirano stablo.

Složenost: $O(n)$

Binarno stablo: move konstruktor

```
template <typename KljucTip, typename ElemTip>
StabloBST<KljucTip, ElemTip>::StabloBST(StabloBST && m)
  : korijen (m.korijen)
{
    m.korijen = nullptr;
};
```

Složenost: $O(1)$

Binarno stablo: destruktor

Pri realizaciji destruktora za binarno stablo, moramo dealocirati sve čvorove (baš sve, ne smijemo izgubiti ni jedan). Kako?

Lijeni način (bez mnogo razmišljanja):

- Uklonimo korijen (metod za uklanjanje čvora). Svako uklanjanje će srediti stablo tako da će neki čvor doći na mjesto korijena. Uklanjamo dokle god postoji korijen.
- Složenost: približno $O(n \log n)$

Pametniji način:

- Krenemo otpozadi i uklanjamo listove (nemaju djece).
- Očigledno, u ovom slučaju ćemo koristiti **post-order prolazak** kroz stablo i uklanjati čvor (list) do kojeg smo došli.
- Složenost:
    - $O(n)$

BST – STL implementacija: map kontejner

- Header `<map>`, klasa `std::map`
- Metodi:
  - ubacivanje elementa x sa ključem k
    - `insert(): kont.insert(std:pair<K, V>{k, x})`
    - `operator[]: kont[k] = x;`
  - Uklanjanje elementa sa ključem k:
    - `erase(): kont.erase(k)`
  - `size` – broj elemenata u kontejneru
  - `empty` – vraća `true` ako je prazan, `false` ako nije
- Primjer korištenja:

```
#include<map>
  ...
  std::map<int, Student> studenti;
  Student novokreirani_student = ...
  studenti[19234] = novokreirani_student;
  ...
```