Nelinearne strukture podataka

Vrste apstraktnih tipova podataka

- **lista** (engl. *list*)
- **stek** (engl. *stack*)
- **red** (engl. *queue*)
- **red sa prioritetom** (engl. *priority queue*)
- **dek** (engl. *double-ended queue – deque*)
- **dek sa prioritetom** (engl. *priority deque*)
- **stablo** (engl. *tree*)
- **graf** (engl. *graph*)
- **mapa**, multimapa (engl. *map*, *multimap*)
- skup, multiskup (engl. *set*, *multiset*)

Strukture podataka

Linearne strukture podataka

- polje (niz)
- spregnuta (povezana) lista
- strukture sa posebnim pravilima pristupa:
  - stek
  - red
  - dek

Nelinearne strukture podataka

- stablo
- graf

Nelinearne strukture podataka

Nelinearne strukture podataka su strukture u kojima **elementi nisu organizovani sekvencijalno (mogu imati više od jednog prethodnika i jednog sledbenika)**

Elementi u nelinearnim strukturama podataka mogu biti **povezani sa većim brojem drugih elemenata** čime se odražava poseban odnos između tih elemenata

U opštem slučaju, **svi elementi** u nekoj nelinearnoj strukturi podataka **ne mogu se obići u jednom prolazu**

Dva najznačnija primera nelinearnih struktura podataka su **stablo** i **graf**

Stablo

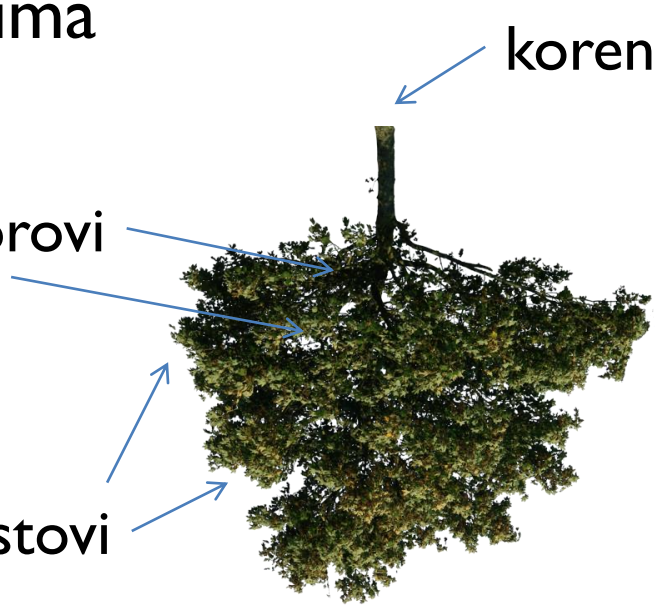
Stablo

Stablo (engl. *tree*) je **apstraktni tip podataka** i **struktura podataka** koja **implementira istoimeni ATP**, u vidu koje se predstavlja **hijerarhijski odnos između elemenata** koji su u **relaciji roditelj-dete** (nadređeni-podređeni), elementi se obično pamte u **čvorovima** (engl. *node*)

Svaki element u stablu, **osim vršnog**, ima **jednog roditelja** i **nula ili više dece**

Vršni element je **koren stabla** (engl. *root*)

Razlikujemo **interne (neterminalne) i eksterne (terminalne) elemente**



Stablo - primene

Primene stabla:

Organizacione šeme

Fajl sistemi

Grafički korisnički interfejsi

(engl. *Graphical User Interface - GUI*)

Baze podataka

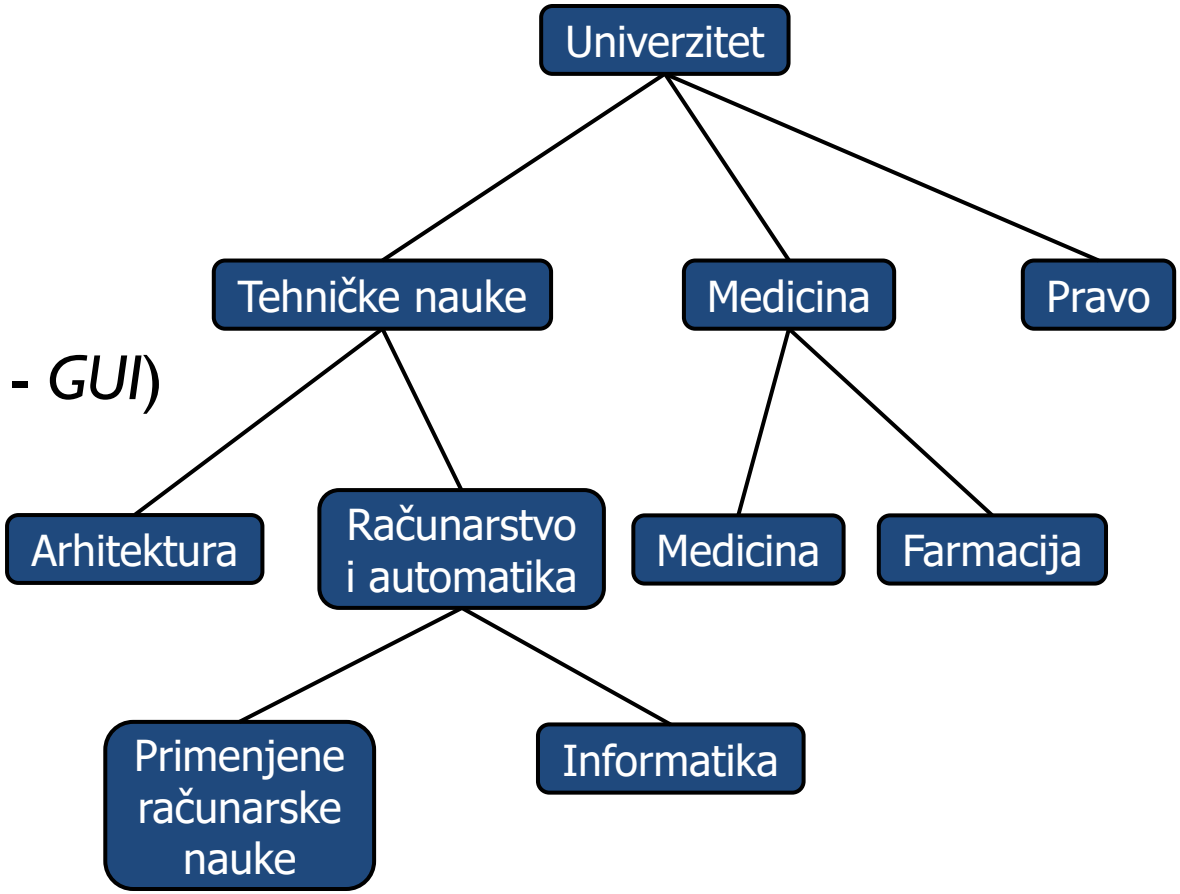
Programski prevodioci

Web sajtovi

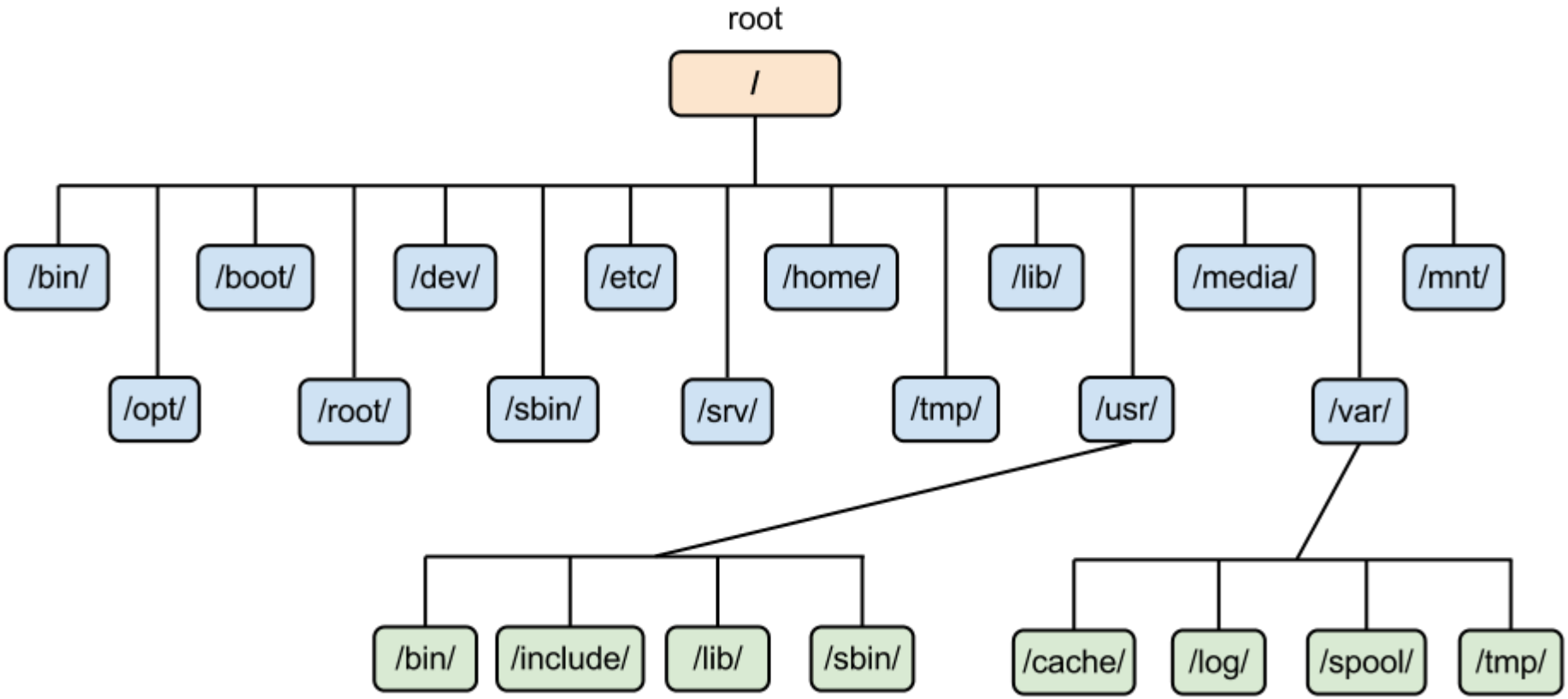
Okruženja za programiranje

(engl. *Integrated Development Environment – IDE*)

...



Primer stabla – Linux fajl sistem



Izvor: https://freedompenguin.com/articles/how-to/learning-the-linux-file-system/

Stablo – matematička definicija

Stablo S je **konačan neprazan skup elemenata**

$$S = \{k\} \cup S_1 \cup S_2 \cup \ldots \cup S_n$$

Sa sledećim svojstvima:

1. Elemenat k je **koren** stabla
2. Ostali elementi su podeljeni u $n \geq 0$ podskupova $S_1, S_2, \ldots, S_n$ od kojih je **svaki stablo**

$S_1, S_2, \ldots, S_n$ su **podstabla** stabla S

Rekurzivna definicija!

Specijalni slučaj $n = 0$ – svako stablo ima barem jedan element – **koren stabla**

Stablo – apstraktni tip i struktura

Posmatrano kao **apstraktni tip podataka**, **stablo** ima **vrednost (podatak)** i **decu**, gde su **deca opet stabla – rekurzivni pojam**

Vrednost stabla se interpretira kao podatak zapisan u korenu, a **deca stabla** kao podstabla koja su deca korena

Posmatrano kao **struktura podataka**, **stablo** je **grupa čvorova** u kojoj **svaki čvor ima vrednost i listu pokazivača na druge čvorove** (njegovu decu)

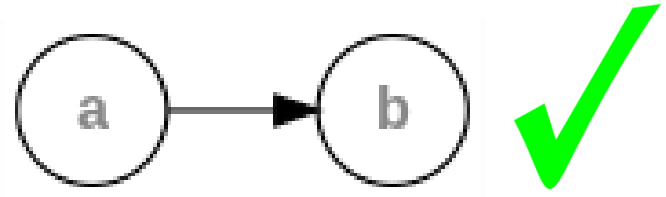
Ograničenja: **dva pokazivača "na dole" ne smeju** pokazivati **na isti čvor,** može biti **samo jedan koren**, **ciklusi ne smeju postojati**

Čvorovi u stablu mogu da čuvaju i pokazivače na sledeći/prethodni element kao i na svog roditelja

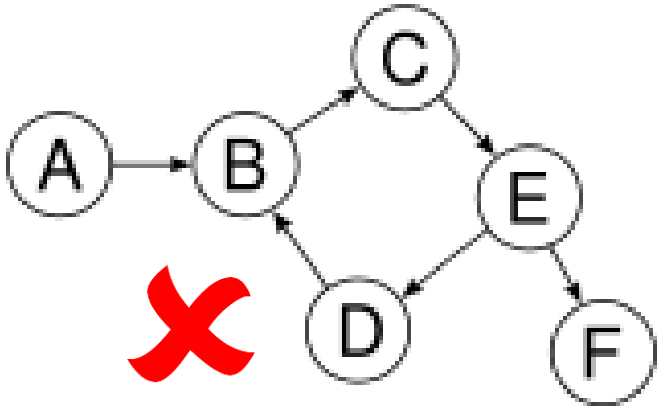
Stabla se najčešće **realizuju** u vidu **pokazivača na koren**

Stablo

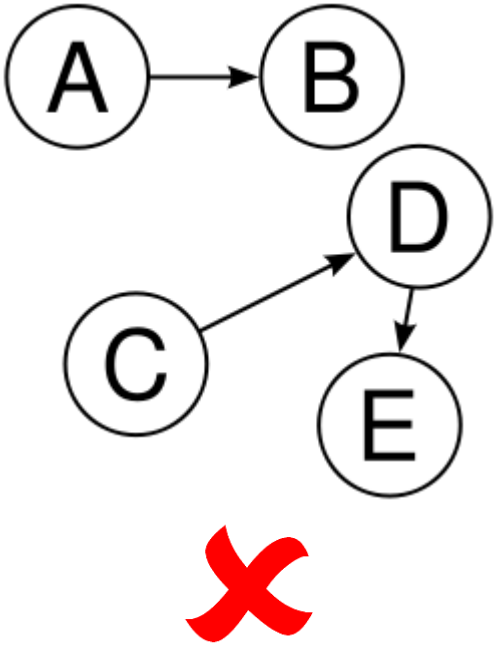
Stablo je **digraf** (engl. *digraph – directed graph*) tj. **usmereni graf** koji se sastoji od **čvorova** i **potega** i **ne sme sadržati cikluse**



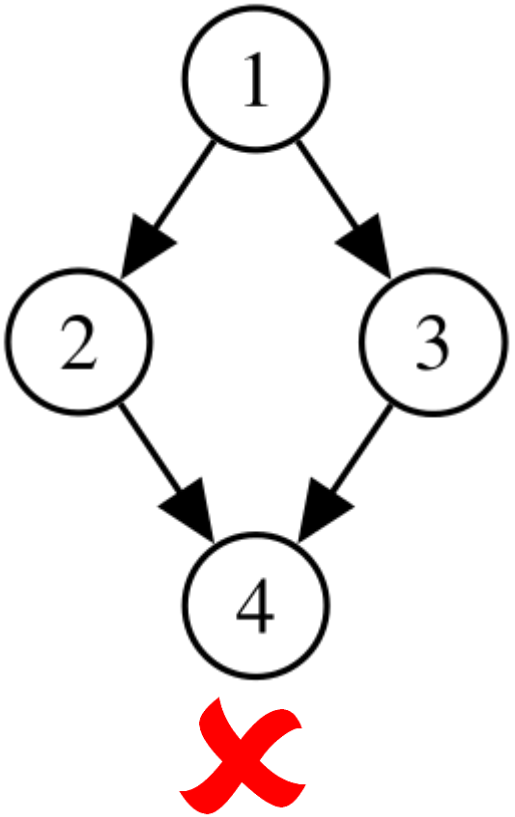
Spregnuta lista je trivijalan slučaj stabla!



Ciklusi su zabranjeni!



Može postojati samo jedan koren!



Svaki čvor ima samo jednog roditelja!

Stablo – terminologija

Koren (engl. *root*) – čvor bez roditelja (A)

Braća/sestre (engl. *siblings*) – čvorovi koji imaju zajedničkog roditelja

Interni (neterminalni) čvor (engl. *internal node*) – čvor sa najmanje jednim detetom (A, B, C, F)

Eksterni (terminalni) čvor (engl. *external node*) – **list** (engl. *leaf*) – čvor bez dece (E, I, J, K, G, H, D)

Preci (engl. *ancestors*) **čvora** – roditelj, pra-roditelj, pra-pra-roditelj itd.

Potomci (engl. *descendants*) **čvora** – dete, unuk, praunuk, itd.

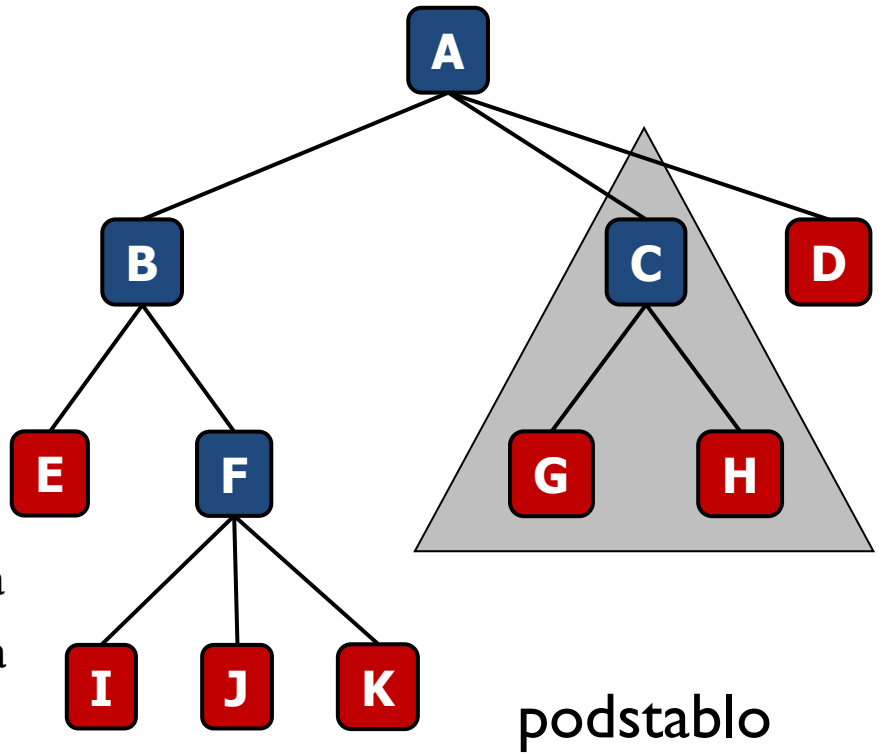
Dubina (engl. *depth*) **čvora** – broj potega do korena

Visina (engl. *height*) **stabla** – maksimalni broj potega od korena do listova u stablu (3)

Stepen (engl. *degree*) **čvora** – broj njegove dece

Stepen stabla – maksimalni step nekog njegovog čvora

Podstablo (engl. *subtree*) – stablo sastavljeno od nekog čvora i njegovih potomaka



Stablo – terminologija

Stablo može biti:

- **neuređeno** (engl. *unordered*)
- **uređeno** (engl. *ordered*)

Stablo je uređeno ako **postoji linearno uređenje definisano za decu svakog čvora**

- bitno je relativno uređenje podstabala u svakom čvoru
- deca se identifikuju kao prvo, drugo, treće, itd.

Stablo – operacije

Opšte (engl. *generic*) operacije:

- **size (veličina)** – vraća veličinu stabla
- **isEmpty (jePrazno)** – provera da li je stablo prazno

Pristupne (engl. *accessor*) operacije:

- **root (koren)** – vraća poziciju korena
- **parent (roditelj)** – vraća poziciju roditelja
- **children (deca)** – vraća iterator kojim se mogu obići potomci čvora

Stablo – operacije

Operacije upita (engl. *query*):

- **isInternal (jeInterni)** – provera da li je čvor interni
- **isExternal (jeEksterni)** – provera da li je čvor eksterni
- **isRoot (jeKoren)** – provera da li je čvor koren

Operacije ažuriranja (engl. *update*):

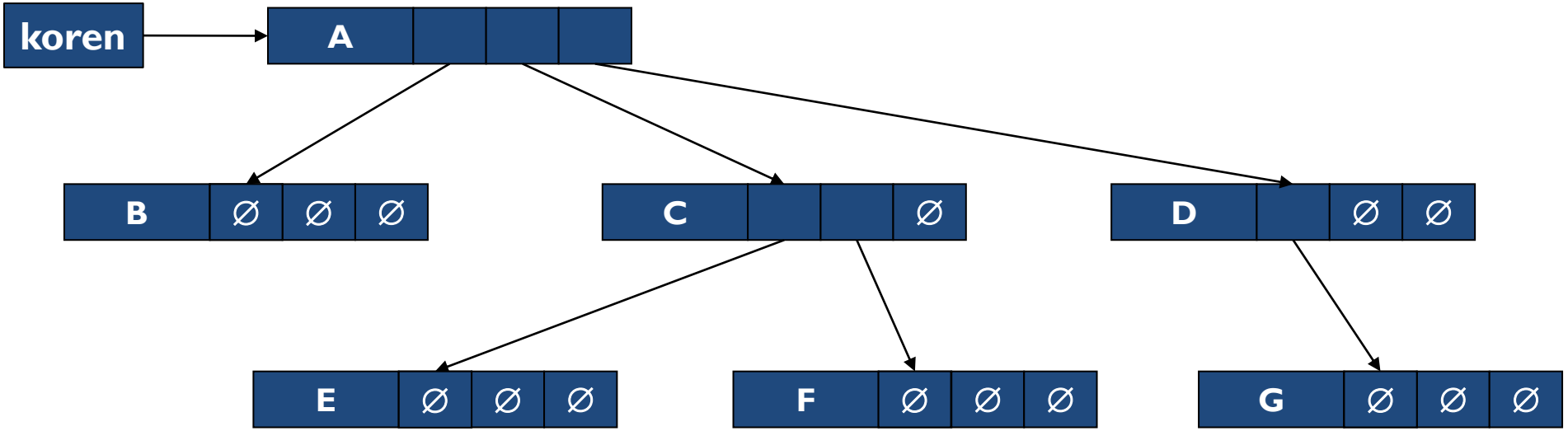
- **swapElements (zameniMestaPodacima)** – zamena vrednosti podataka između dva čvora
- **replaceElement (zameniVrednostPodatka)** – zamena vrednosti podatka u nekom čvoru

Dodatne operacija ažuriranja definišu se u skladu sa strukturom podataka kojom se implementira ATP stablo

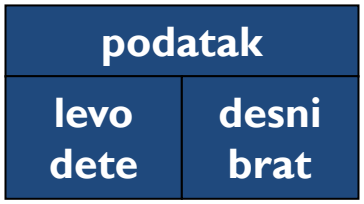
Stablo – primer spregnute realizacije

Svaki čvor u stablu sadrži:

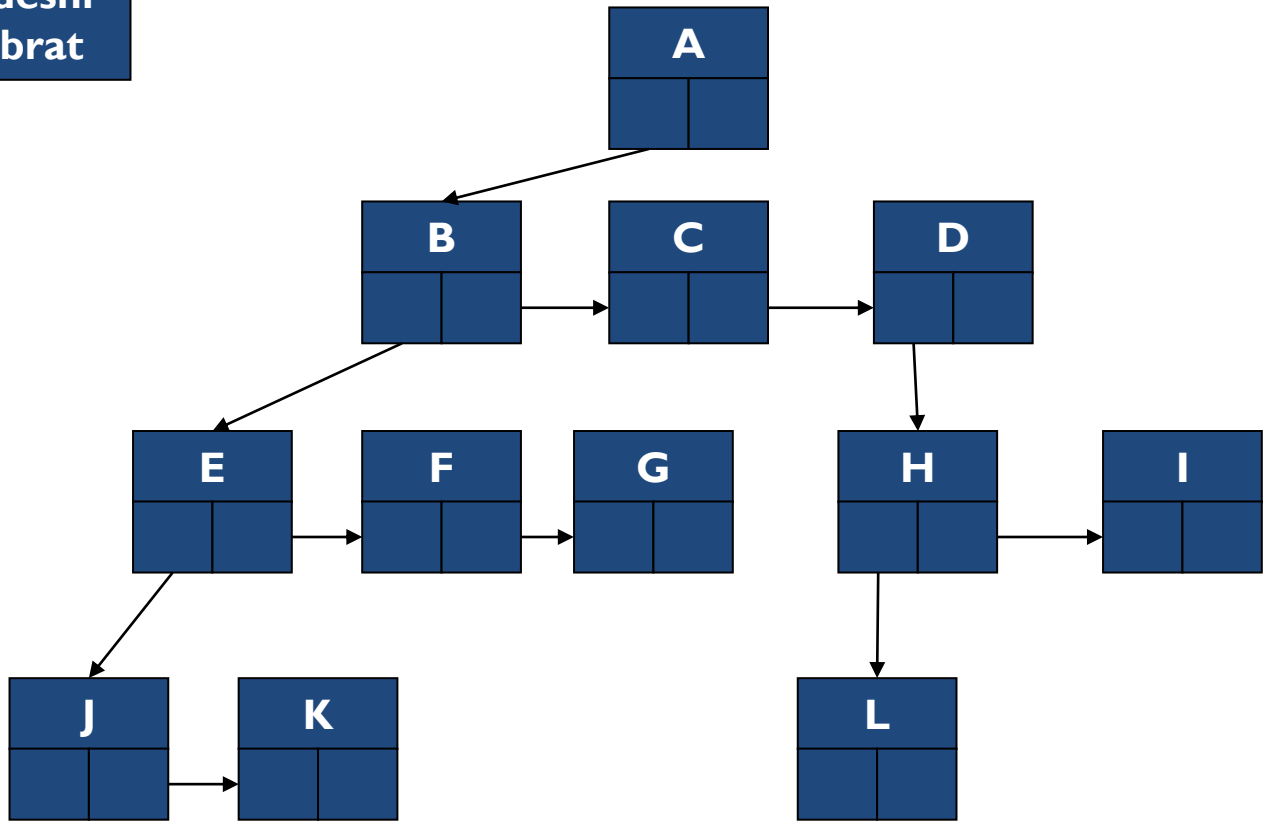
- polje **podatak** – korisne inforamcije
- polja **deca** – pokazivači na njegovu decu (∅ oznaka za NULL)



Stablo – primer spregnute realizacije



Reprezentacija krajnje levo dete – desni brat



Stablo – obilazak

Obilazak (engl. *traversal*) stabla predstavlja postupak **pristupanja** njegovim **čvorovima na sistematski način**

Dva glavna načina obilaska su:

- **preorder (sa vrha ka dnu)**
- **postorder (sa dna ka vrhu)**

Priroda ovih postupaka je **rekurzivna**

Preorder (sa vrha ka dnu):

- poseti koren
- obiđi u preorder redosledu decu (podstabla)

Postorder (sa dna ka vrhu):

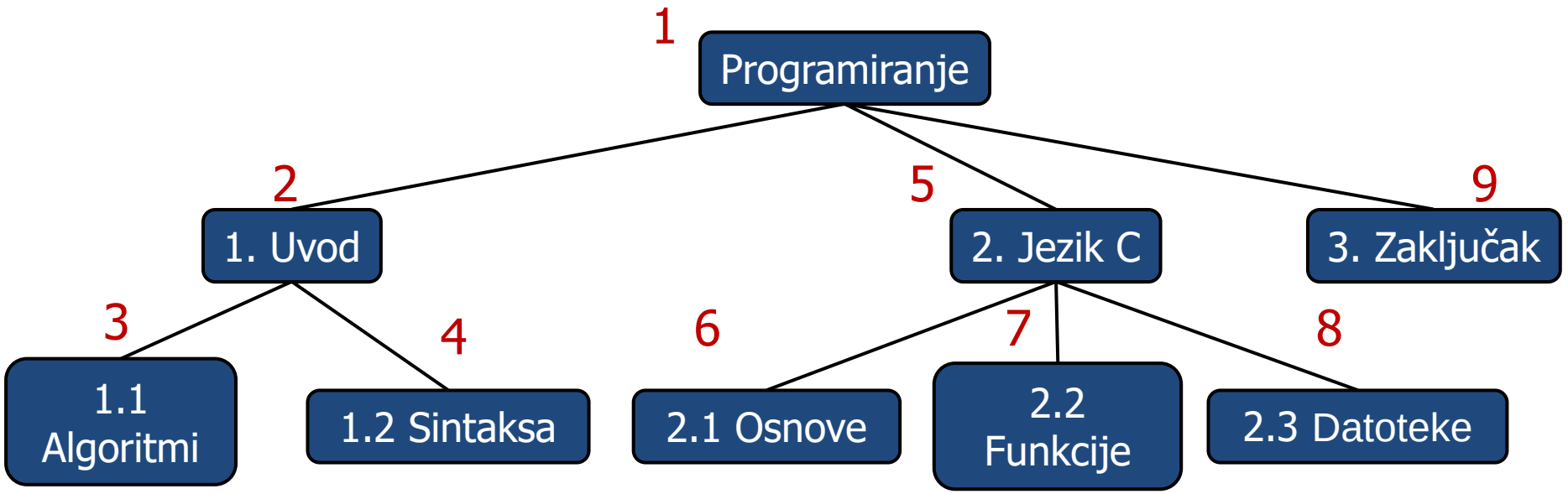
- obiđi u postorder redosledu decu (podstabla)
- poseti koren

Stablo – preorder obilazak

Kod obilaska sa vrha ka dnu, čvor se posećuje pre posete njegovim potomcima

```
algoritam preorder(v)
    poseti(v)
    za svako dete w čvora v
        preorder (w)
```

Primer: prikaz strukture dokumenta

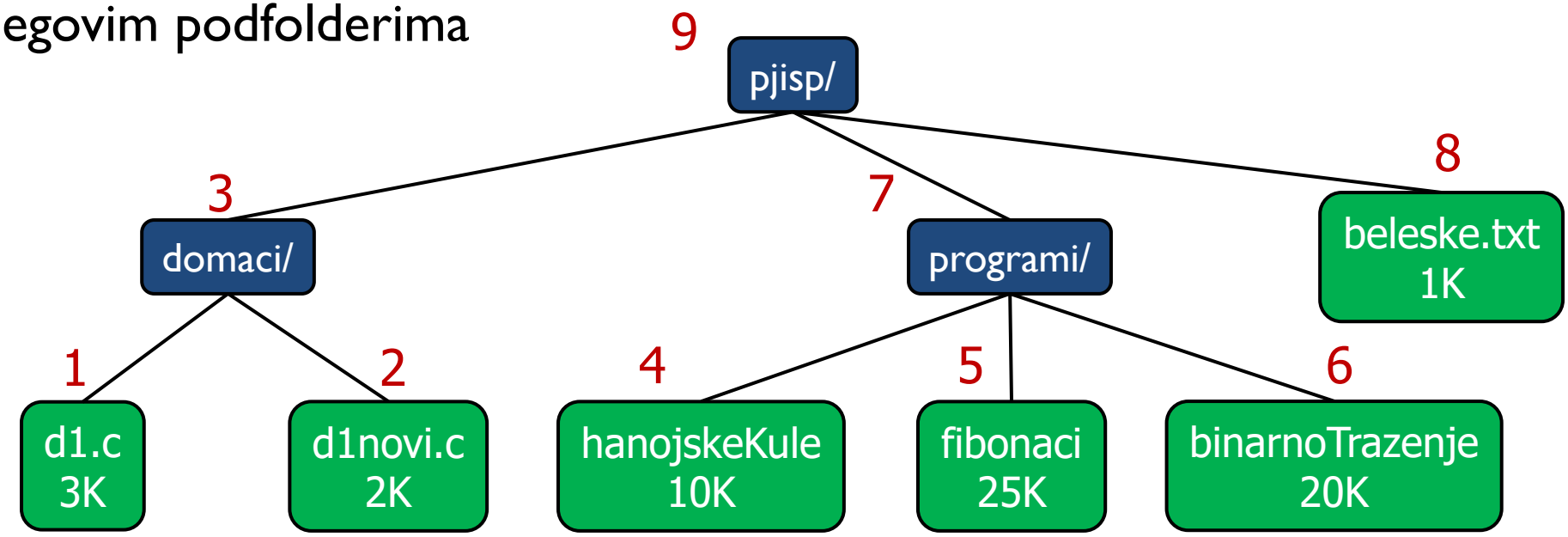


Stablo – postorder obilazak

Kod obilaska sa dna ka vrhu, čvor se posećuje nakon posete potomaka

```
algoritam postorder(v)
    za svako dete w čvora v
        postorder (w)
    poseti(v)
```

Primer: izračunavanje zauzeća memorije od strane fajlova u folderu i njegovim podfolderima



Stablo – primer sintaksno stablo

Apstraktno sintaksno stablo (engl. *abstract syntax tree – AST*) ili, prostije, **sintaksno stablo**, je reprezentacija apstraktne sintaksne strukture izvornog koda napisanog u nekom programskom jeziku

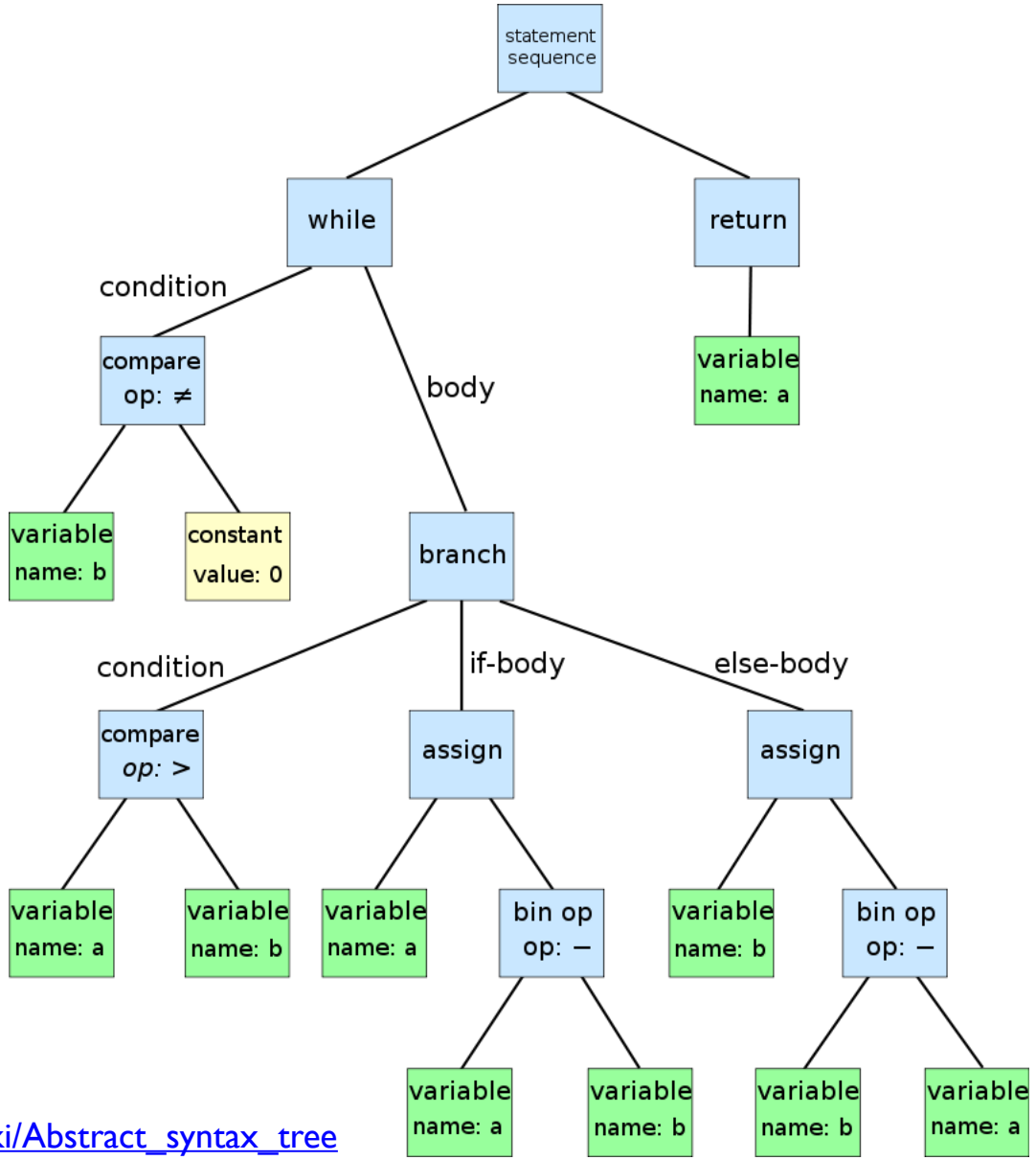
Svaki **čvor u sintaksnom stablu** odgovara nekom **elementu** koji se javlju u **konstrukciji izvornog koda**

Sintaksa je apstraktna u smislu da **ne predstavlja svaki detalj** koji se pojavljuje u "pravoj" sintaksi – npr. vitičaste zagrade { } koje ograničavaju blokove su implicitne, kao i karakteri ; kojima se označava kraj naredbe

Stablo – primer sintaksno stablo

Euklidov algoritam:

```
while (b != 0){
        if (a > b)
                a = a − b;
        else
                b = b − a;
}
return a;
```



Izvor: https://en.wikipedia.org/wiki/Abstract_syntax_tree

Binarno stablo

Binarno stablo

Binarno stablo (engl. *binary tree*) je uređeno stablo kod kojeg svaki čvor ima najviše dvoje dece

Deca čvora u binarnom stablu nazivaju se **levo i desno dete** (engl. *left and right child*)

Podstabla S_1 i S_2 čvora S nazivaju se **levo i desno podstablo**

Binarno stablo koje u svakom čvoru ima ili nijedno ili oba podstabla naziva se **striktno binarno stablo (pravo binarno stablo)**

Realizacija binarnog stabla može biti **sekvencijalna** i **spregnuta**

Binarno stablo – vrste

Potpuno binarno stablo visine *h* ima $2^{h+1}-1$ čvorova

Potpuno binarno stablo (visine *h*) je **striktno binarno stablo** čiji su **svi listovi na nivou *h***

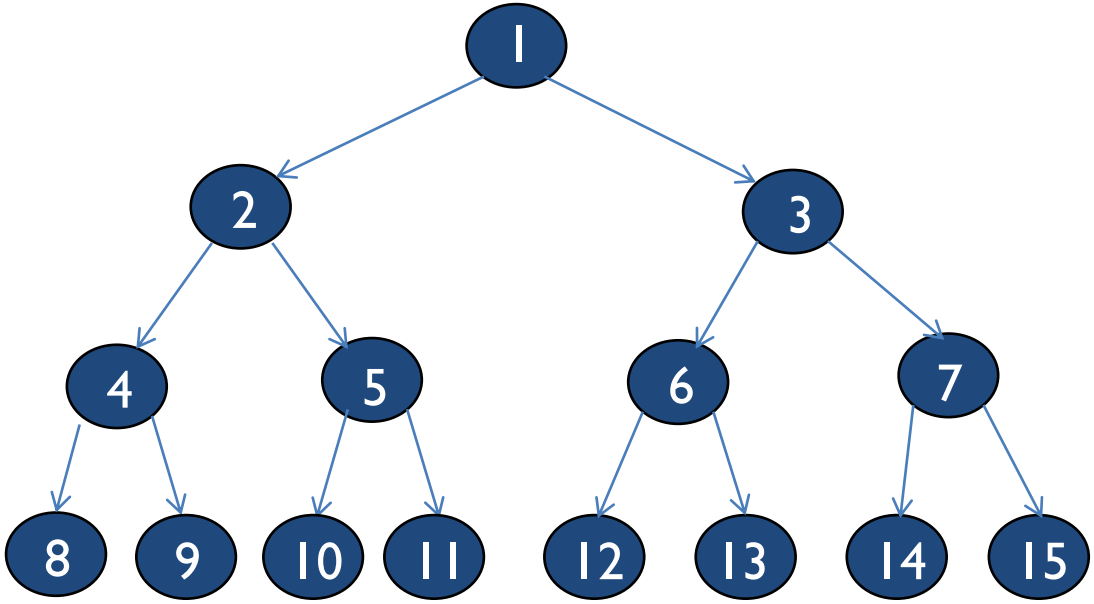
Gotovo potuno binarno stablo (visine *h*) je **striktno binarno stablo** čiji su **svi listovi na nivou *h* ili *h*-1** i kod koga **bilo koji čvor koji ima desnog potomka na nivou *h*** ima i **sve leve potomke na nivou *h***

Kompletno binarno stablo je binarno stablo kod koga su **svi nivoi, osim možda poslednjeg, kompletno popunjeni** i **svi čvorovi su što je moguće više "levo"**

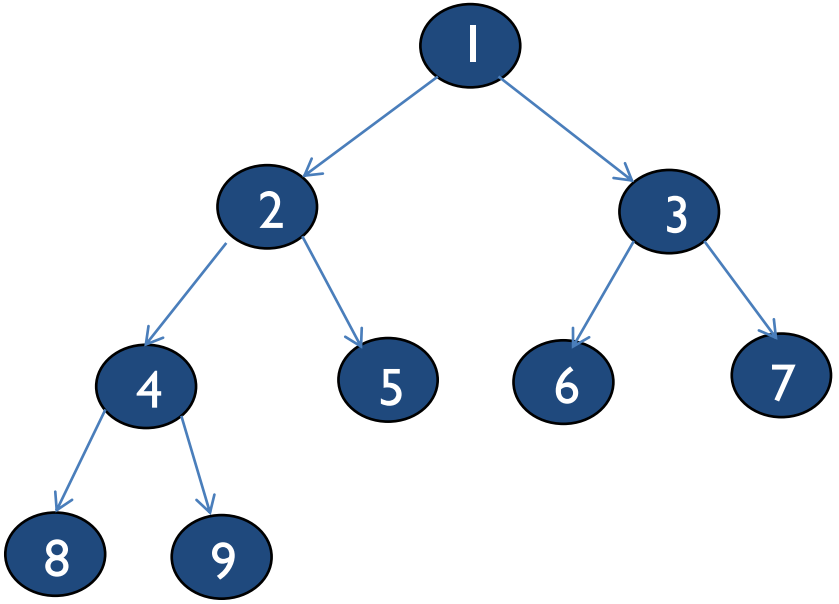
Balansirano binarno stablo je binarno stablo kod koga se **broj elemenata podstabla na istom hijerarhijskom nivou razlikuje najviše za 1**

Binarno stablo – vrste

Potpuno binarno stablo



Kompletno binarno stablo



Binarno stablo – primer i primene

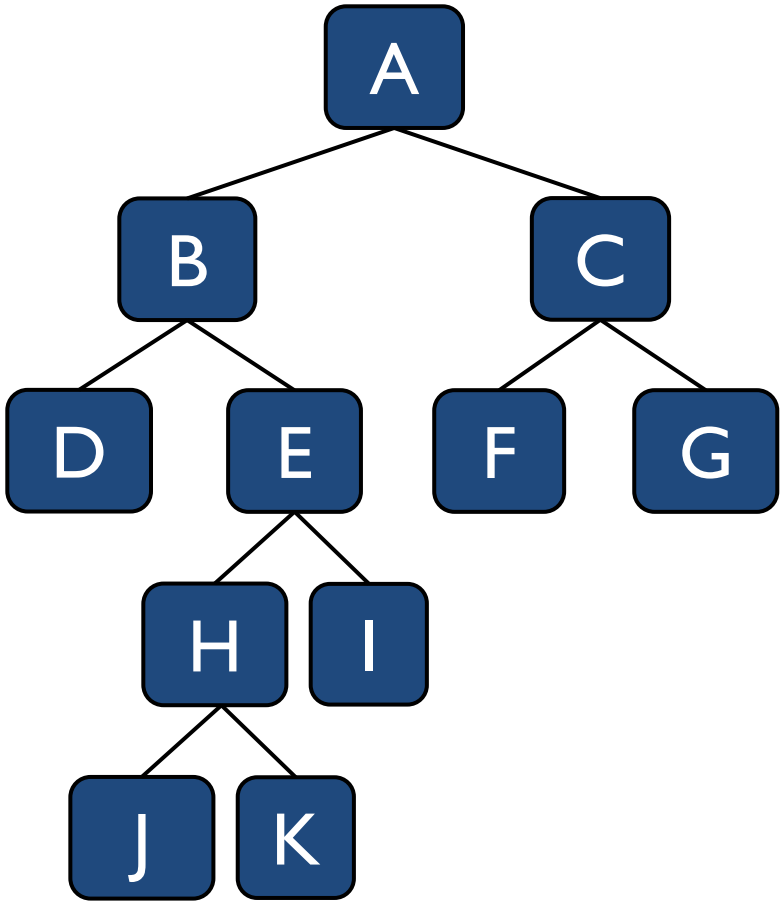
Primene binarnih stabala:

- Evaluacija aritmetičkih izraza
- Procesi odlučivanja
- Traženje –

  binarna stabla traženja
- Sortiranje –

  gomila (engl. *heap*)

  heapsort algoritam



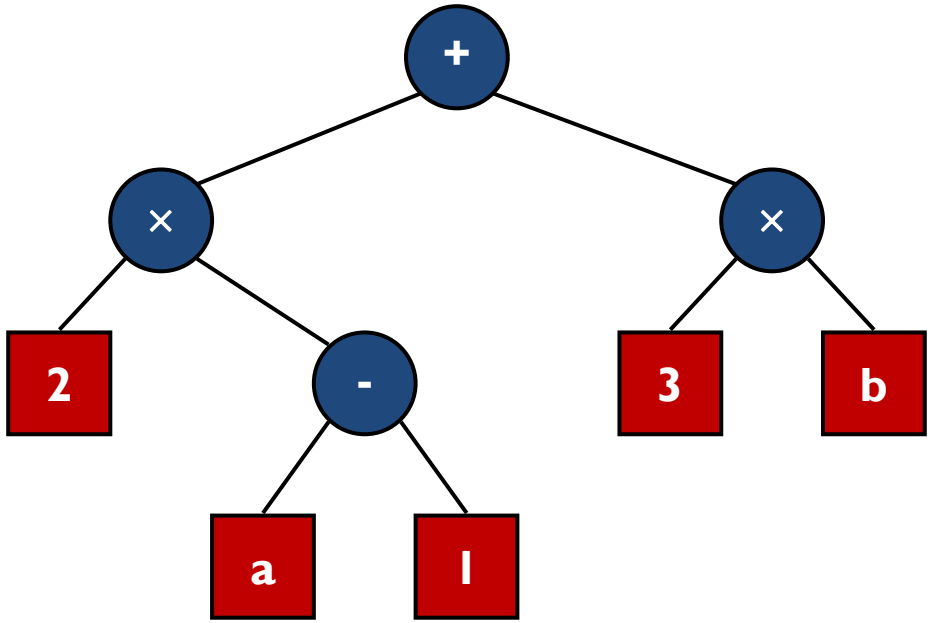
Binarno stablo – primena

Binarno **stablo aritmetičkog izraza**

interni čvorovi – **operatori**

eksterni čvorovi – **operandi**

Primer: binarno stablo za aritmetički izraz (2×(a–1)+(3×b))



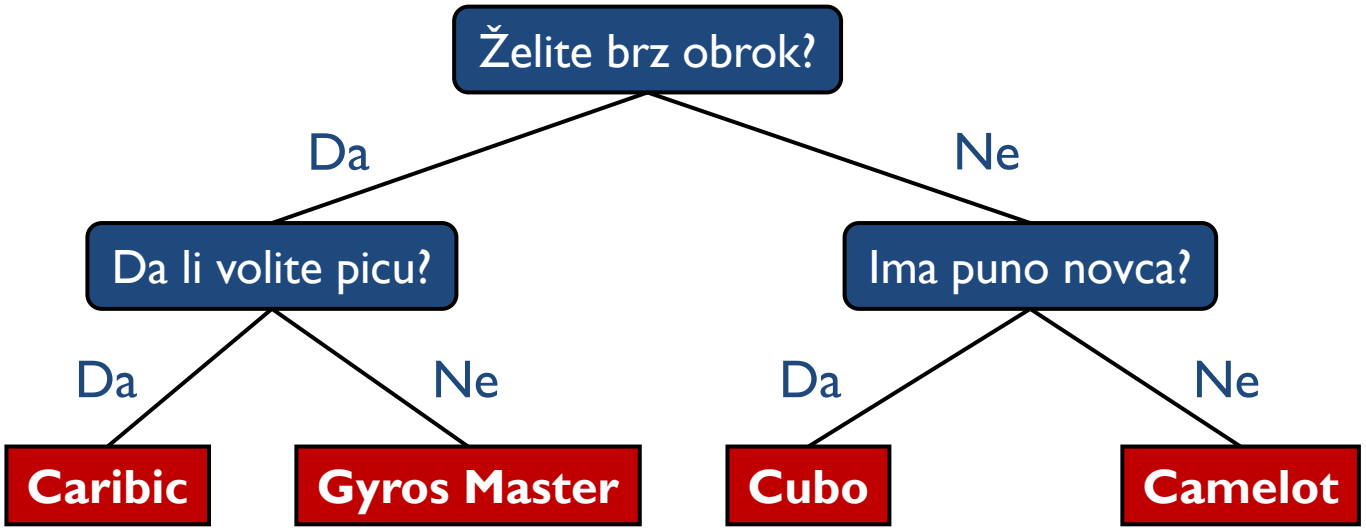
Binarno stablo – primena

Binarno **stablo odlučivanja** (engl. *binary decision tree*)

interni čvorovi – **pitanja sa da/ne odgovorima**

eksterni čvorovi – **odluke**

Primer: gde za večeru?



Binarno stablo – operacije

Binarno stablo kao apstraktni tip podataka je proširenje stabla, tj. **nasleđuje sve operacije** definisane za ATP stablo

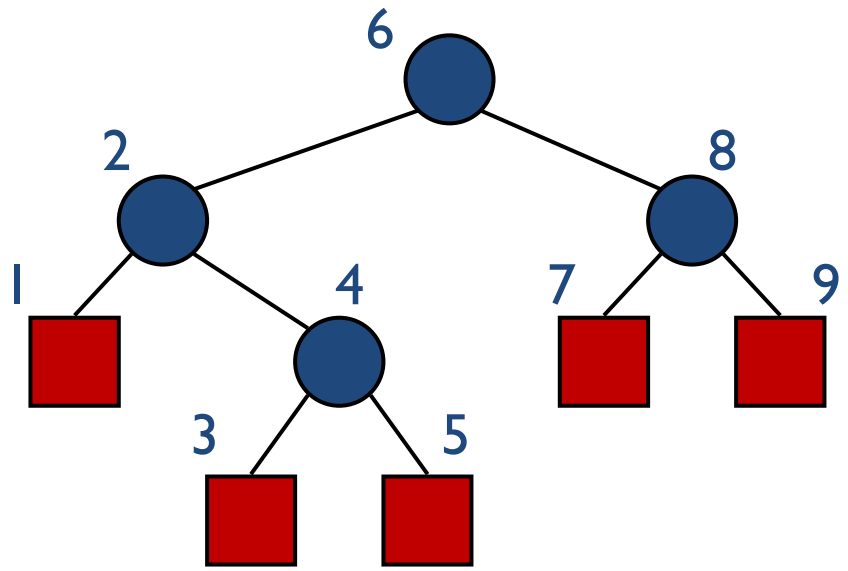
Dodatne operacije:

- **leftChild (levoDete)** – vraća poziciju levog deteta
- **rightChild (desnoDete)** – vraća poziciju desnog deteta
- **sibling (brat)** – vraća poziciju brata

Prilikom implementacije obično se definišu i dodatne operacije za ažuriranje binarnog stabla

Binarno stablo – obilazak

Kod binarnih stabala, pored preorder i postorder obilazaka, definišemo i **inorder** obilazak (s leva na desno) – **čvor se posećuje nakon obilaska** njegovog **levog podstabla**, a **pre obilaska desnog podstabla**

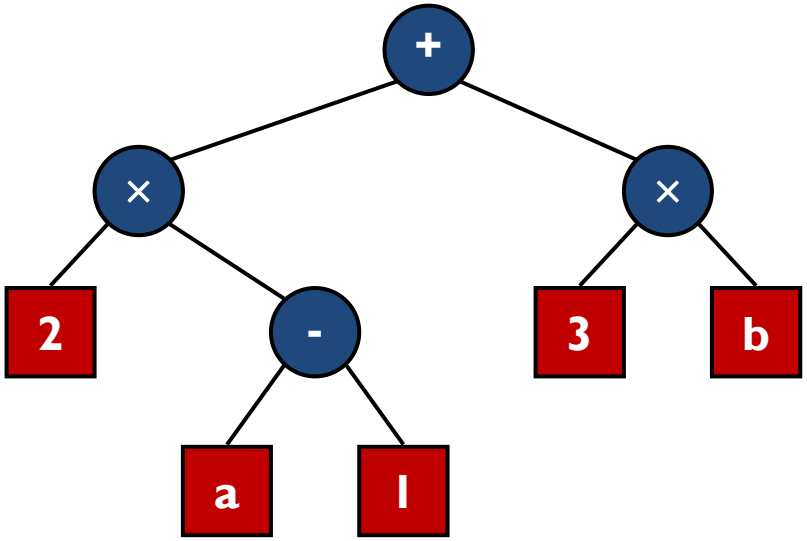



```
algoritam inorder(v)
    ako je jeInterni (v)
        inorder (levoDete (v))
    poseti(v)
    ako je jeInterni (v)
        inorder (desnoDete (v))
```

Prikaz aritmetičkih izraza

Specijalizacija inorder obilaska:

- odštampaj operand ili operator prilikom posete čvoru
- odštampaj '(' pre obilaska levog podstabla
- odštampaj ')' posle obilaska desnog podstabla



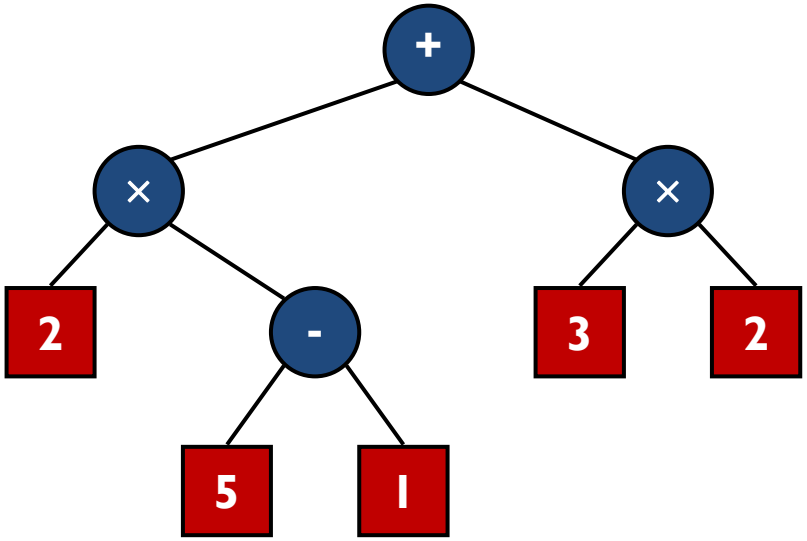
((2 × (a - 1)) + (3 × b))

```
algoritam inorder (v)
    ako jeInterni (v){
        štampaj('(')
        inorder (levoDete (v))}
    štampaj(v.podatak ())
    ako jeInterni (v){
        inorder (desnoDete (v))
        štampaj (')')}
```

Evaluacija aritmetičkih izraza

Rekurzivna funkcija koja **vraća vrednost podstabla**

Prilikom **posete internom čvoru, kombinuju se vrednosti podstabala**



14

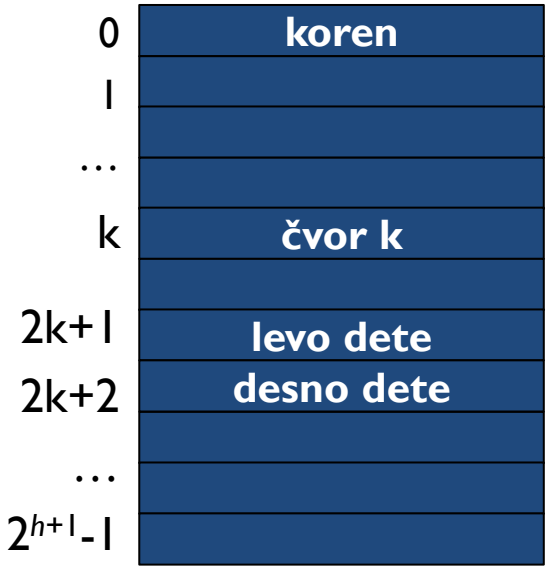
```
algoritam evalIzraz(v)
    ako jeEksterni (v)
        vrati v.podatak ()
    u suprotnom
        x ← evalIzrar(levoDete (v))
        y ← evalIzraz(desnoDete (v))
        ◊ ← operator smešten u v
        vrati x ◊ y
```

Binarno stablo – sekvencijalna realizacija

Reprezentacija pogodna za **potpuno** ili **gotovo potpuno binarno stablo**

Koristi se polje **stablo** dužine $2^{h+1}-1$ (*h* - visina stabla)

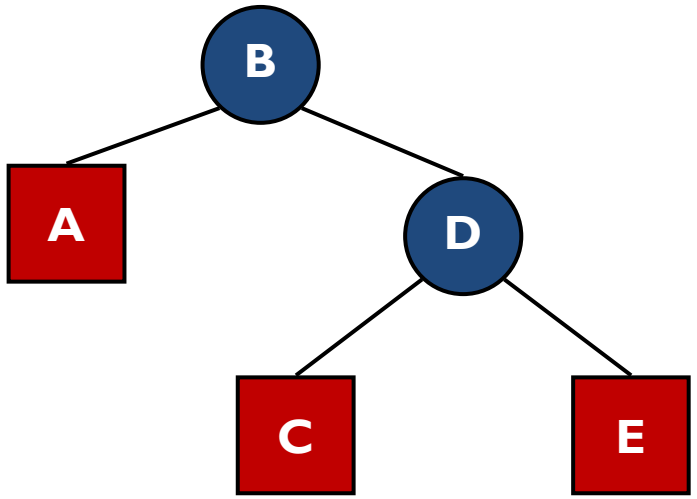
stablo[0] pamti **koren**, ako se u **stablo[k]** pamti vrednost **čvora k,** njegovo **levo dete** se pamti u **stablo[2k+1],** a **desno dete** u **stablo[2k+2], roditelj** se dobija kao **stablo[(k-1)/2]** (celobr. deljenje)

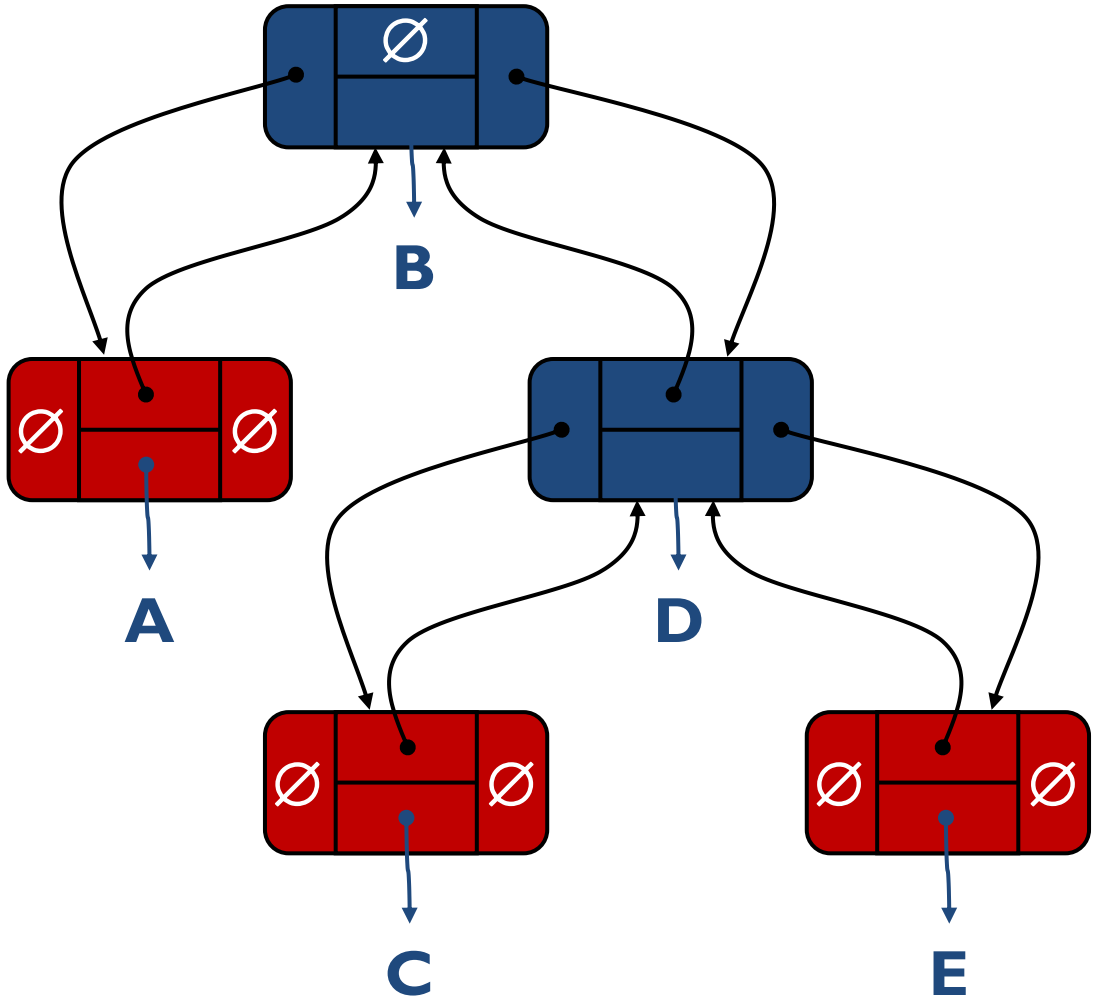


Binarno stablo – spregnuta realizacija

Čvor se sastoji od sledećih polja:

- **podatak**
- **pokazivač na roditelja**
- **pokazivač** na **levo dete**
- **pokazivač** na **desno dete**





Implementacija binarnog stabla

Zadatak 1:

Napisati u jeziku C program kojim se **binarno stablo** realizuje **spregnuto**. Implementirati operacije za dodavanje i traženje elemenata u stablu, brisanje stabla, preorder, inorder i postorder obilazak stabla.

Vežba 1:

Izmeniti prethodni program tako da korisnik može da bira željenu operaciju za rad sa binarnim stablom. Omogućiti ponavljanje izabranih operacija sve dok korisnik ne odluči da izađe iz programa.

Binarno stablo – Zadatak 1

```
#include<stdlib.h>
#include<stdio.h>

typedef struct cvor {
        int podatak;
        struct cvor *desnoDete, *levoDete;
} tCvorBinarnogStabla;

void dodajCvor(tCvorBinarnogStabla**, int);
void preorderObilazak(tCvorBinarnogStabla*);
void inorderObilazak(tCvorBinarnogStabla*);
void postorderObilazak(tCvorBinarnogStabla*);
void obrisiStablo(tCvorBinarnogStabla*);
tCvorBinarnogStabla* trazi(tCvorBinarnogStabla**, int);
```

Binarno stablo – Zadatak 1

```
int main()
{
        tCvorBinarnogStabla *koren, *pomocni;
        koren = NULL;

        // Dodavanje cvorova u stablo
        dodajCvor(&koren, 7);
        dodajCvor(&koren, 4);
        dodajCvor(&koren, 13);
        dodajCvor(&koren, 6);
        dodajCvor(&koren, 42);
        dodajCvor(&koren, 17);
        dodajCvor(&koren, 2);

        // Obilazak stabla
        printf("Preorder (sa vrha ka dnu) obilazak:\n");
        preorderObilazak(koren);

        printf("Inorder (sa leva na desno) obilazak:\n");
        inorderObilazak(koren);
        …
```

Binarno stablo – Zadatak 1

```
...

printf("Postorder (sa dna ka vrhu) obilazak:\n");
postorderObilazak(koren);

// Trazenje vrednosti u stablu
pomocni = trazi(&koren, 4);
if (pomocni)
{
        printf("Pronadjen cvor=%d\n", pomocni->podatak);
}
else
{
        printf("Vrednost nije pronadjena u stablu!\n");
}

// Brisanje stabla
obrisiStablo(koren);

return 0;
}
```

Binarno stablo – Zadatak 1

```
void dodajCvor(tCvorBinarnogStabla **stablo, int vrednost)
{
        tCvorBinarnogStabla *noviCvor = NULL;
        if (!(*stablo))          // Dodavanje cvora u prazno stablo
        {
                noviCvor = (tCvorBinarnogStabla *)malloc(sizeof(tCvorBinarnogStabla));
                noviCvor->levoDete = noviCvor->desnoDete = NULL;
                noviCvor->podatak = vrednost;
                *stablo = noviCvor;
                return;
        }

        if (vrednost < (*stablo)->podatak)
        {
                dodajCvor(&(*stablo)->levoDete, vrednost);
        }
        else if (vrednost >(*stablo)->podatak)
        {
                dodajCvor(&(*stablo)->desnoDete, vrednost);
        }
}
```

Binarno stablo – Zadatak 1

```
void preorderObilazak(tCvorBinarnogStabla *stablo)
{
        if (stablo)
        {
                printf("%d\n", stablo->podatak);
                preorderObilazak(stablo->levoDete);
                preorderObilazak(stablo->desnoDete);
        }
}

void inorderObilazak(tCvorBinarnogStabla *stablo){
        if (stablo){
                inorderObilazak(stablo->levoDete);
                printf("%d\n", stablo->podatak);
                inorderObilazak(stablo->desnoDete);
        }
}
```

Binarno stablo – Zadatak 1

```
void postorderObilazak(tCvorBinarnogStabla *stablo)
{
        if (stablo)
        {
                postorderObilazak(stablo->levoDete);
                postorderObilazak(stablo->desnoDete);
                printf("%d\n", stablo->podatak);
        }
}

void obrisiStablo(tCvorBinarnogStabla *stablo)
{
        if (stablo)
        {
                obrisiStablo(stablo->levoDete);
                obrisiStablo(stablo->desnoDete);
                free(stablo);
        }
}
```

Binarno stablo – Zadatak 1

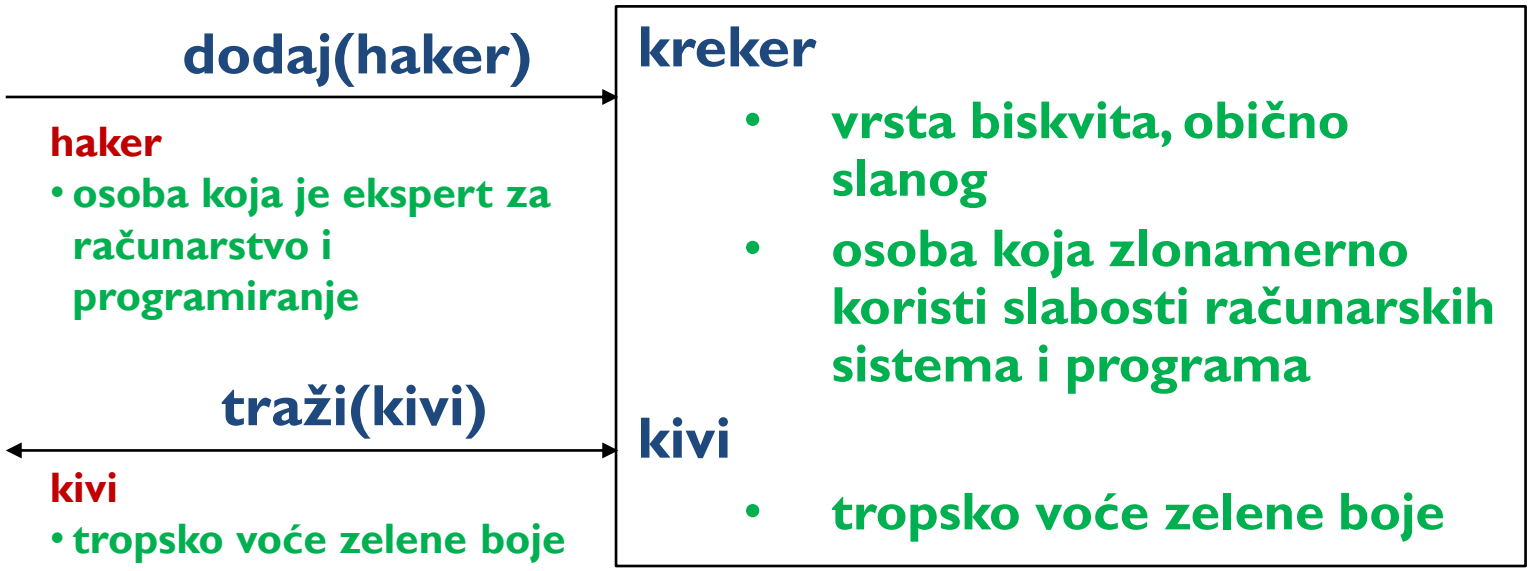
```
tCvorBinarnogStabla* trazi(tCvorBinarnogStabla **stablo, int vrednost)
{
        if (!(*stablo))
        {
                return NULL;
        }
        if (vrednost < (*stablo)->podatak)
        {
                trazi(&((*stablo)->levoDete), vrednost);
        }
        else if (vrednost >(*stablo)->podatak)
        {
                trazi(&((*stablo)->desnoDete), vrednost);
        }
        else if (vrednost == (*stablo)->podatak)
        {
                return *stablo;
        }
}
```

Binarno stablo traženja

Rečnik i ATP za traženje

Operacije:

- **kreiraj**
- **uništi**
- **dodaj**
- **traži**
- **obriši**



Rečnik (engl. *dictionary*) ili **mapa** (engl. *map*) ili **asocijativni niz** (engl. *associative array*) čuva **vrednosti** (engl. *values*) pridružene **ključevima** (engl. *keys*) specificiranim od strane korisnika – **uređeni par (k, v)**

- **ključevi** mogu biti bilo kog homogenog uporedivog tipa
- **vrednosti** mogu biti bilo kog homogenog tipa
- implementacija – **stablo traženja** (engl. *search tree*) ili **heš tabela** (engl. *hash table*), polje za podatke je struktura (**struct**) sa dva dela – ključ i vrednost

ATP za traženje (engl. *search ADT*) – **ključevi = vrednosti**

"Naivne" implementacije rečnika

	neuređeno polje	uređeno polje	spregnuta lista
dodaj (sa/bez duplikata)	traži + O(1)	O(*N*)	traži + O(1)
traži	O(*N*)	O(log *N*)	O(*N*)
briši	traži + O(1)	O(*N*)	traži + O(1)

Cilj: **tražiti brzo** kao u **uređenom polju**, dinamički **brzo dodavati/brisati** čvorove kao kod **spregnute liste**

Binarno stablo traženja

Binarno stablo traženja (engl. *binary search tree – BST*) je binarno stablo koje čuva **ključeve** (ili **parove ključ-vrednost**) u **internim čvorovima** i za koje važi da, ako su *u*, *v* i *w* tri čvora, takva da je *u* u levom podstablu čvora *v*, a *w* je u desnom podstablu *v*, tada važi da je **ključ**(*u*) < **ključ**(*v*) < **ključ**(*w*)

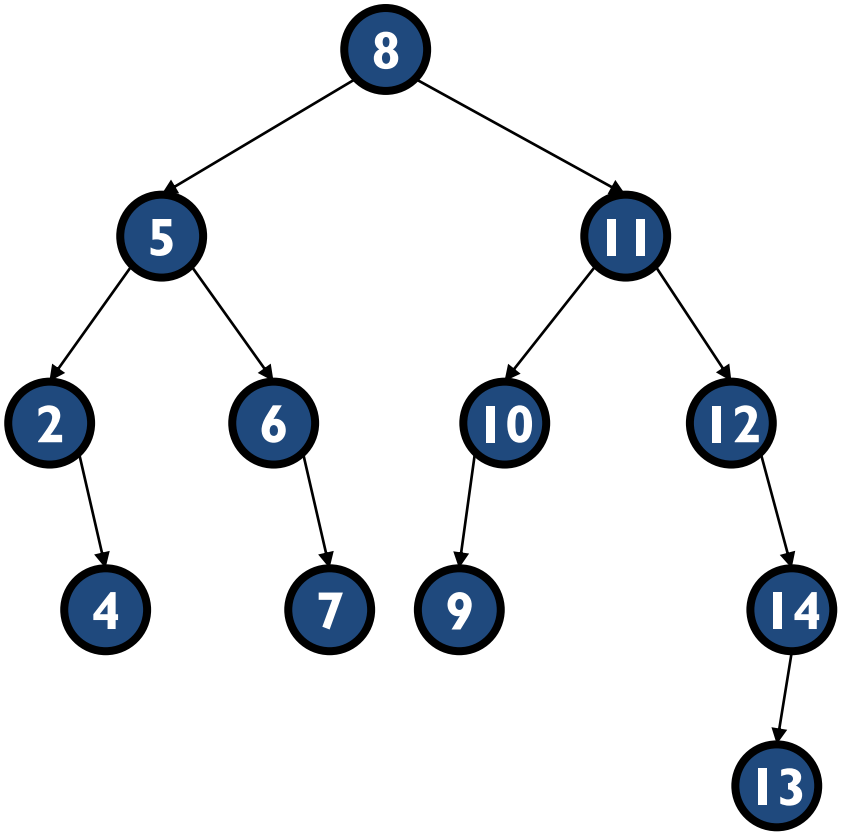
Vrednost čvora je **veća** od **svih vrednosti levog podstabla** i **manja** od svih vrednosti **desnog podstabla**

Eksterni čvorovi se mogu definisati da ne sadrže ključeve/vrednosti

Binarno stablo traženja

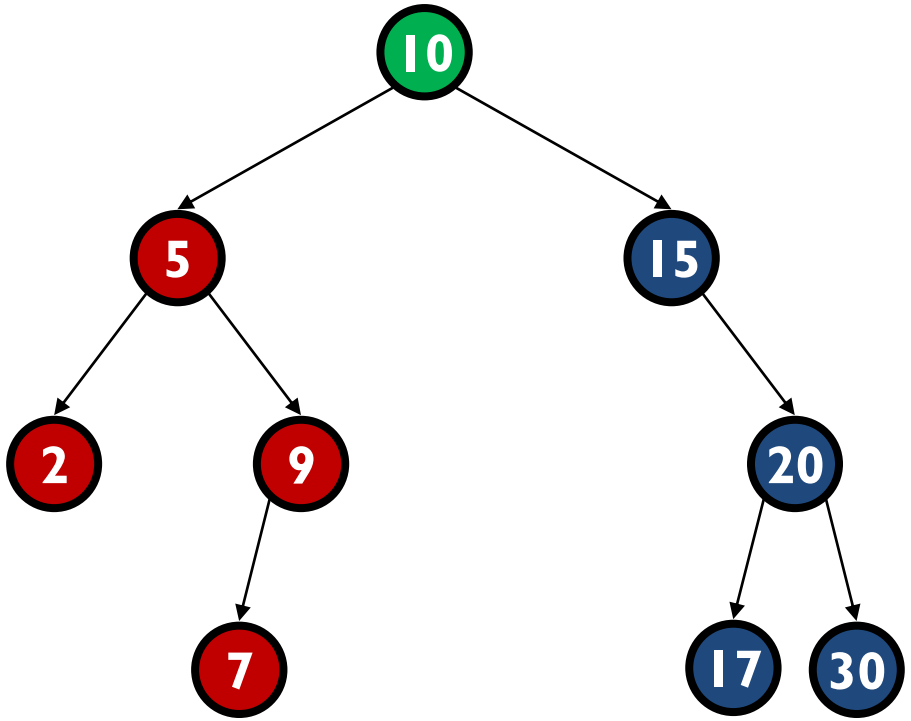
Svojstva BST:

- Svi ključevi u levom podstablu su manji od ključa korena
- Svi ključevi u desnom podstablu su veći od ključa korena
- Posledice:
  - Lako je pronaći bilo koji ključ
  - Dodavanje i brisanje čvorova može se efikasno realizovati izmenom pokazivača



BST – inorder obilazak

1. poseti levo podstablo
2. obradi čvor
3. poseti desno podstablo



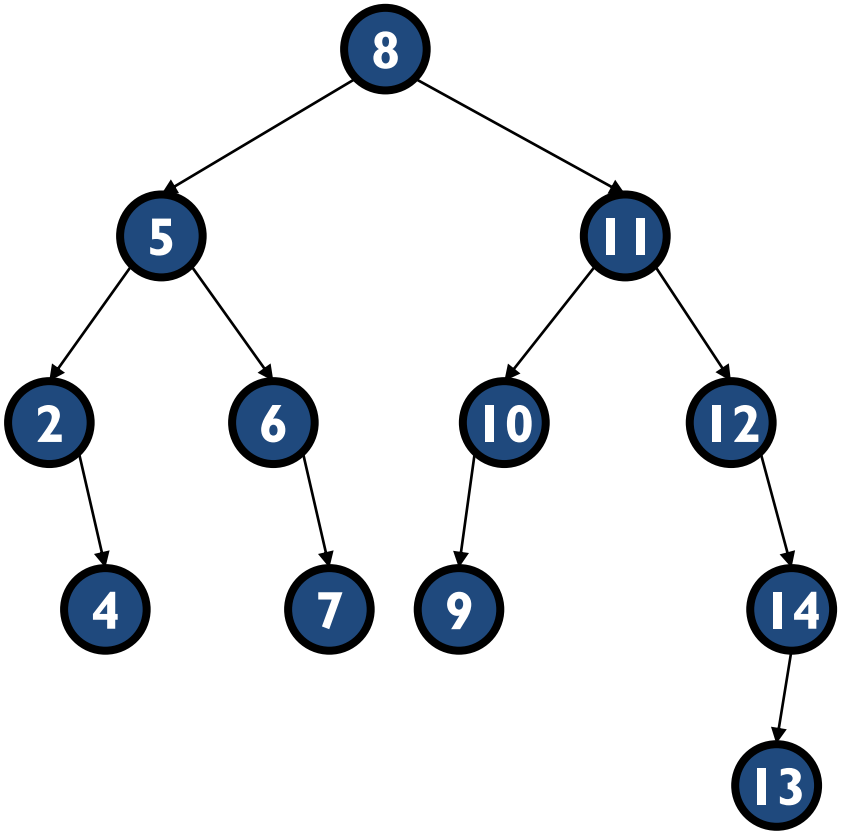
Inorder obilazak:

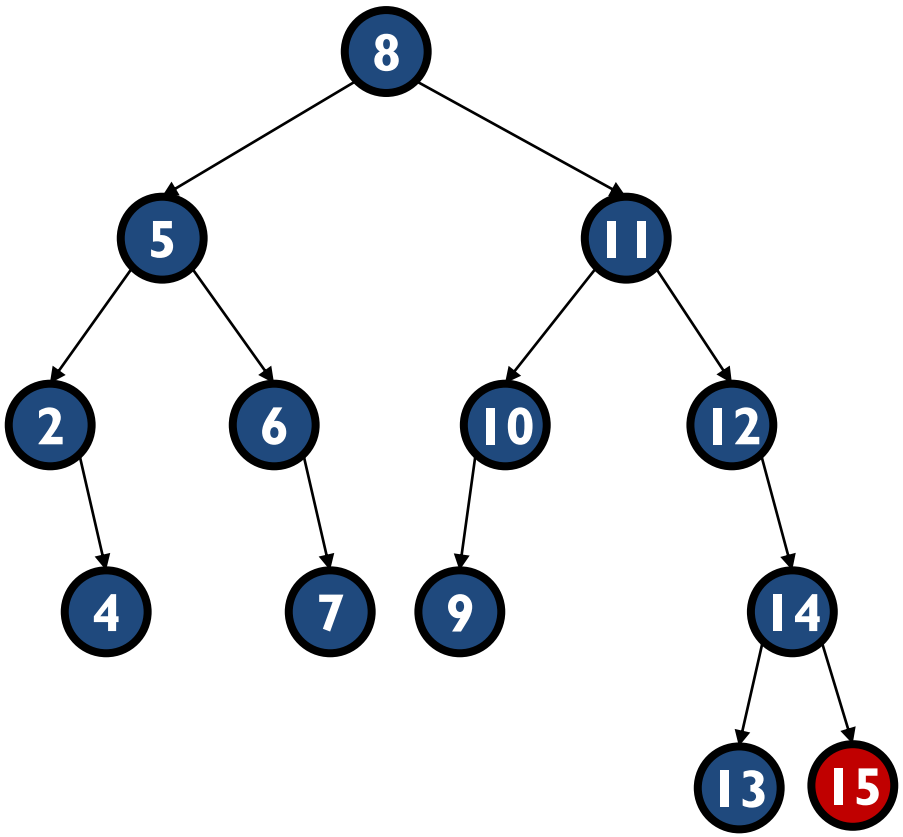
2→5→7→9→10→15→17→20→30

Inorder obilazak BST daje niz uređen u **rastućem** poretku

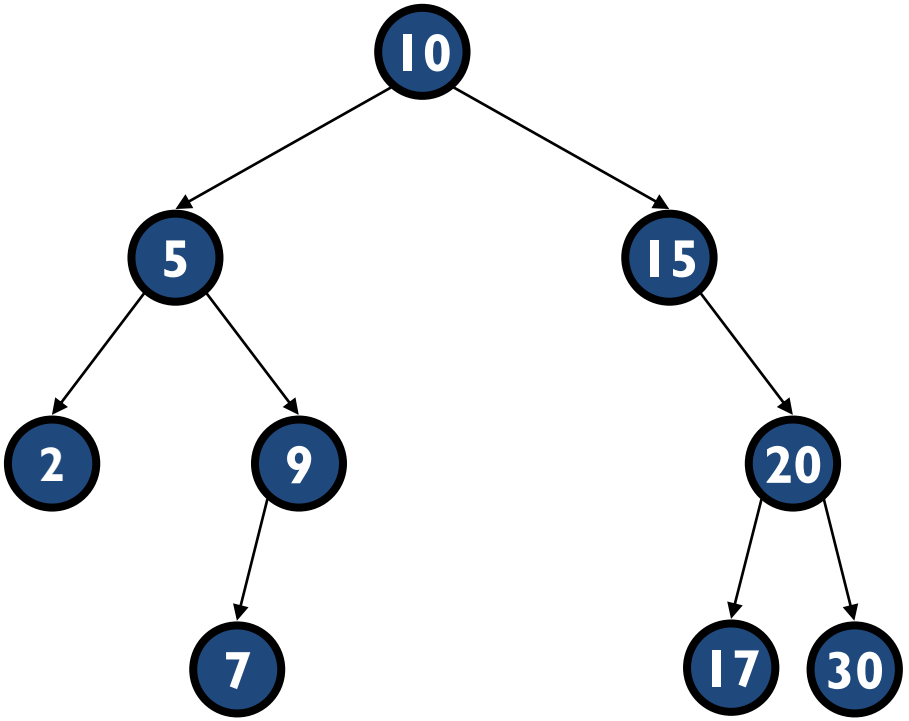
BST – dodavanje čvora

dodaj(15)





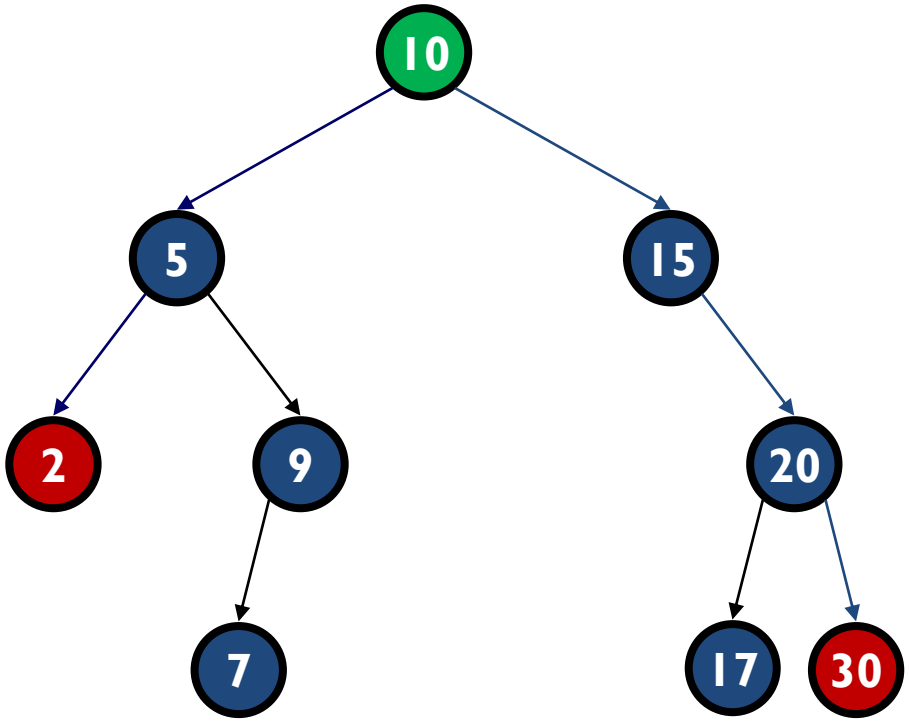
BST – traženje čvora



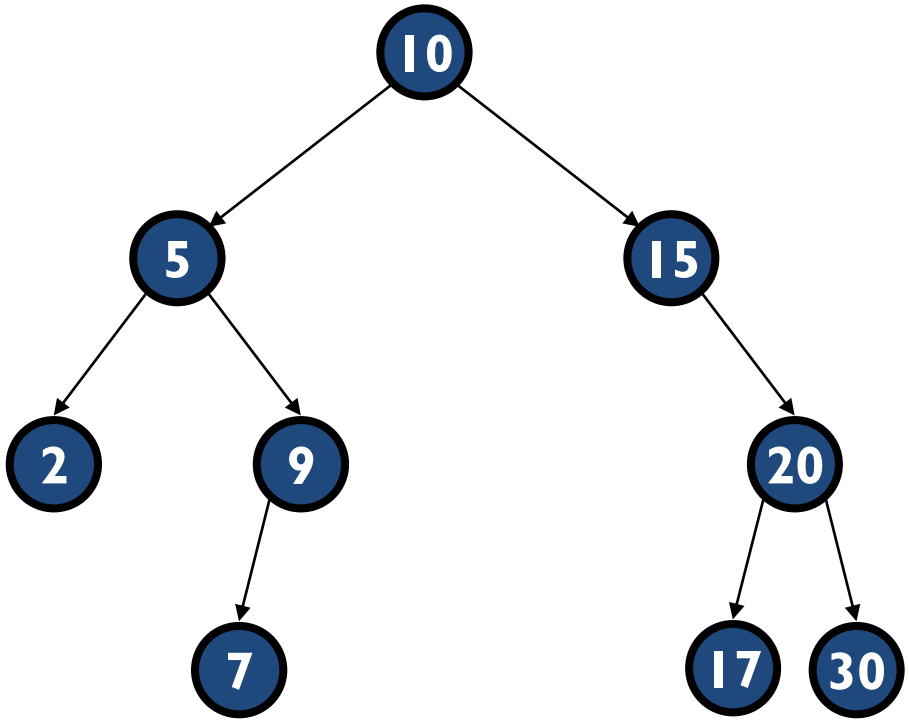

```
tCvor* trazi(tCvor *cvor, int kljuc) {
    if (!cvor)
        return NULL;

    if (cvor->kljuc == kljuc)
            return kljuc;

    if (cvor->kljuc > kljuc)
            return trazi(cvor->levo, kljuc);
    else
            return trazi(cvor->desno, kljuc);
}
```

BST – vratiMin/vratiMax

```
tCvor* min(tCvor* koren)
{
  if (koren.levo == NULL)
    return koren;
  else
    return min(koren.levo);
}
```



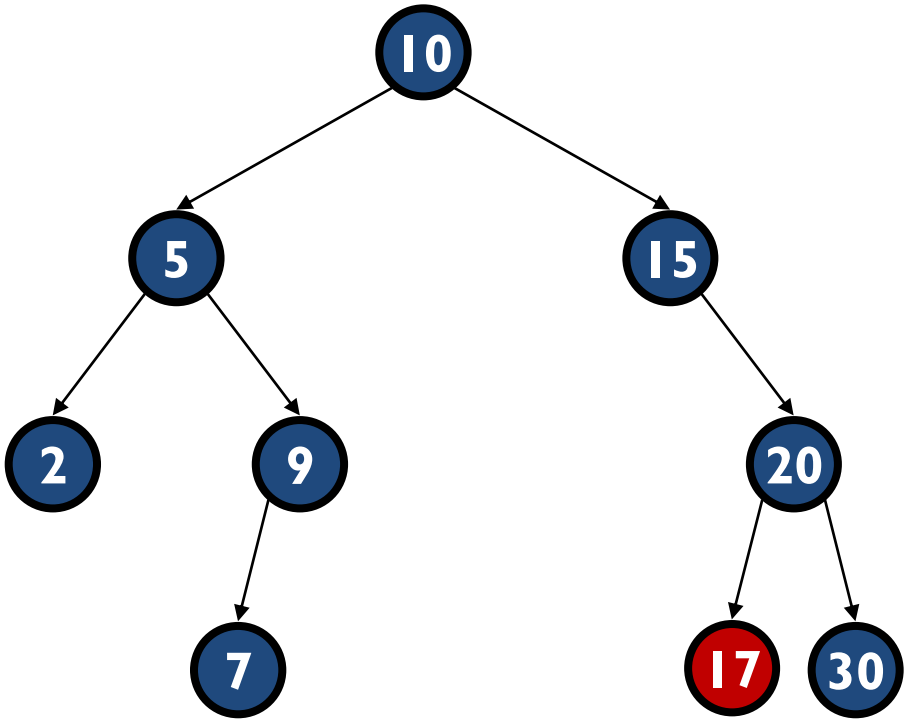
BST – brisanje čvora



Zašto brisanje čvora može biti teže od dodavanja?

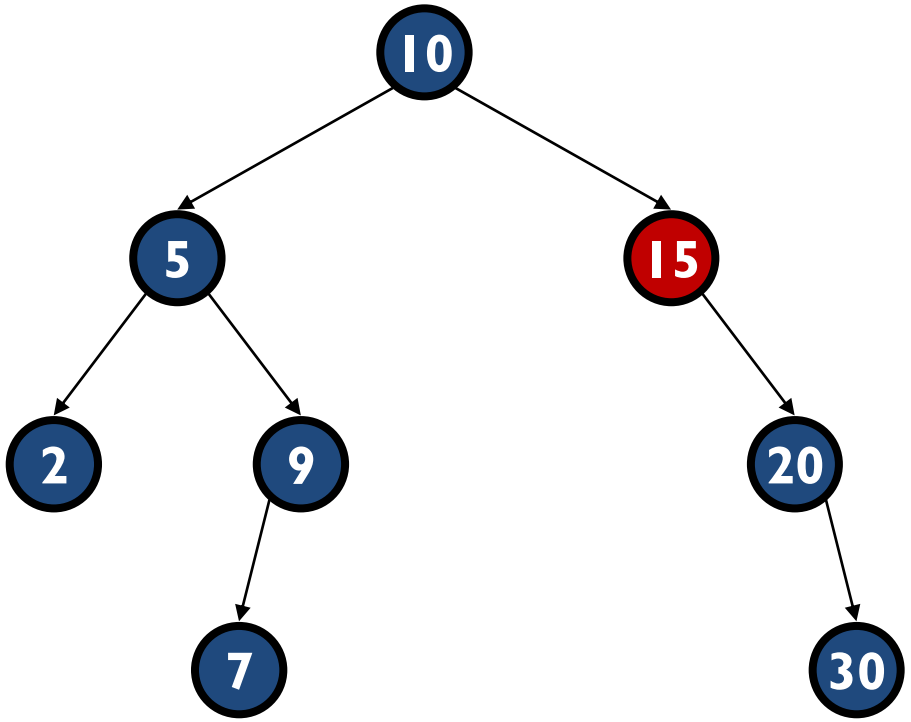
BST – brisanje čvora (slučaj list)

obriši(17)



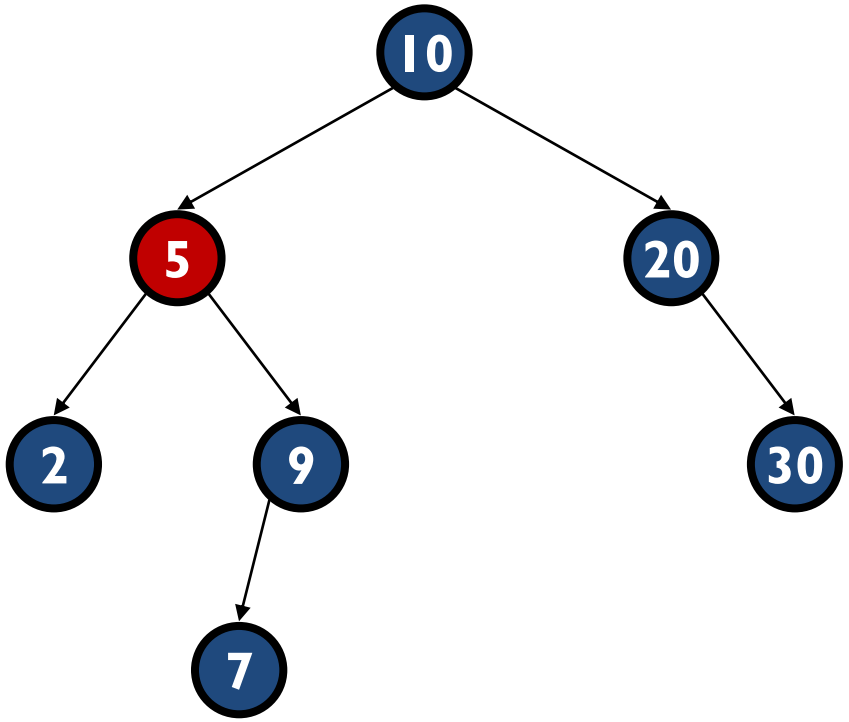
BST – brisanje (slučaj jedno dete)

obriši(15)



BST – brisanje (slučaj dva deteta)

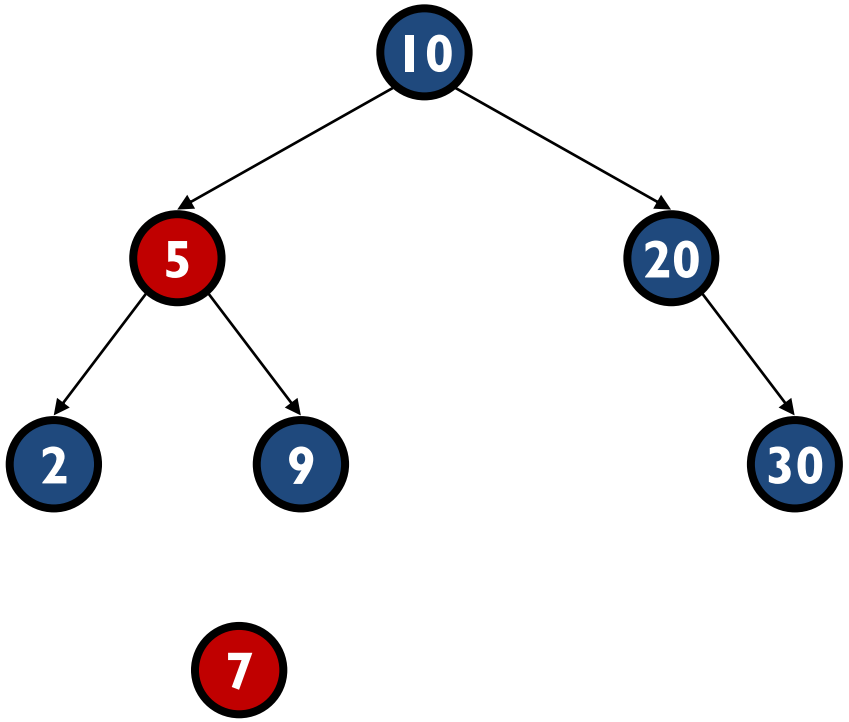
obriši(5)



Zameni čvor sa onim koji se **garantovano** nalazi između levog i desnog podstabla – **sledbenik** u **inorder** obilasku

BST – brisanje (slučaj dva deteta)

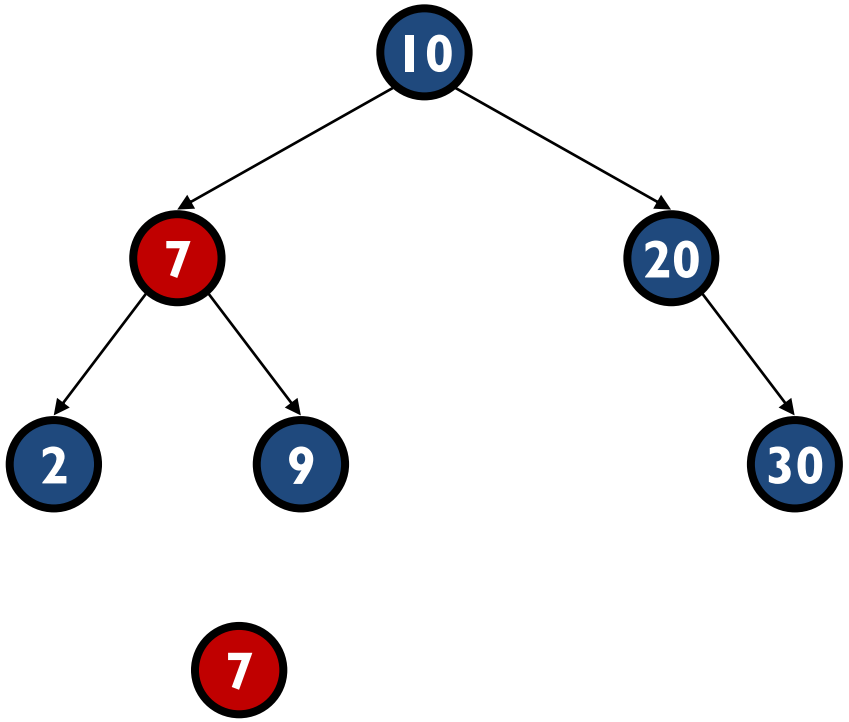
obriši(5)



Jednostavno je obrisati sledbenika – uvek ima nijedno ili jedno dete! Da li se možete iskoristiti i in-order prethodnik?

BST – brisanje (slučaj dva deteta)

obriši(5)



Na kraju je potrebno iskopirati vrednost podatka iz obrisanog naslednika u originalni čvor

BST – odloženo brisanje

Odloženo brisanje (engl. *lazy deletion*) – umesto fizičkog brisanja čvorova, samo ih obeležimo (engl. *flag*) kao obrisane

+ prostija realizacija

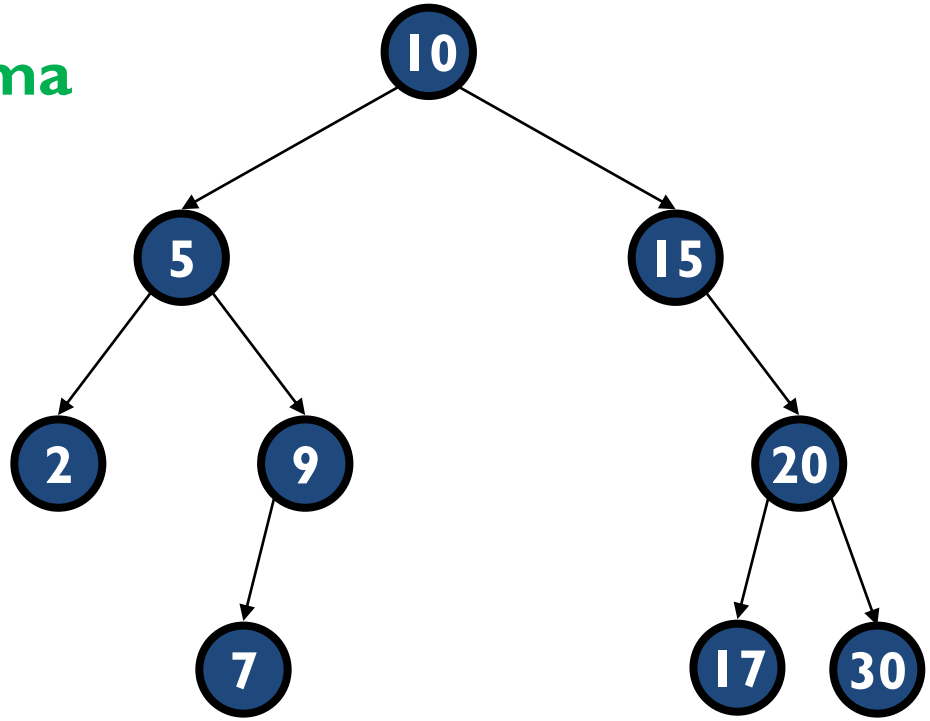
+ fizičko brisanje radi se u grupama

+ neka dodavanja se svode na prosto menjanje obeležja

– dodatne lokacije za obeležje

– mnogo odloženih brisanja usporava traženje

– neke operacije moraju se modifikovati (npr. min i max)



BST – implementacija rečnika

	neuređeno polje	**uređeno polje**	**spregnuta lista**	**binarno stablo traženja**
dodaj (sa/bez duplikata)	traži + O(1)	O(*N*)	traži + O(1)	O(*d*)
traži	O(*N*)	O(log *N*)	O(*N*)	O(*d*)
briši	traži + O(1)	O(*N*)	traži + O(1)	O(*d*)

BST ima **odlične performanse** za **"plitka"** (engl. *shallow*) **stabla**, tj. kada je **dubina *d* mala** (log *N*), **u suprotnom BST** su **jednako loši** kao **spregnute liste!**

BST – balansiranje

Problemi se javljaju kada neka od **grana postane značajno duža**

Degenerisano stablo je stablo kod kojeg **svaki roditelj** ima samo **jedno** dete. Ono je **nebalansirano** i, u najgorem slučaju, **performanse** se spuštaju na **nivo spregnute liste**

Ako funkcija za dodavanje čvora ne podržava **balansiranje**, onda se lako može stvoriti **degenerisano stablo** – ako mu dodajemo podatke koji su već sortirani (tada se, po pitanju performansi, stablo ponaša kao spregnuta lista)

BST – balansiranje

balans = visina(levog podstabla) - visina(desnog podstabla)

Konvencija: **visina praznog podstabla** je **-1**

Svuda nula – **perfektno balansirano**

Svuda mala vrednost – **dovoljno balansirano θ(log*N*)**

- θ je uska asimptotska granica, max dubina je 1.44log*N*

Samo-balansirajuće BST je poznato kao **AVL (Adelson, Velsky, Landis) stablo**