

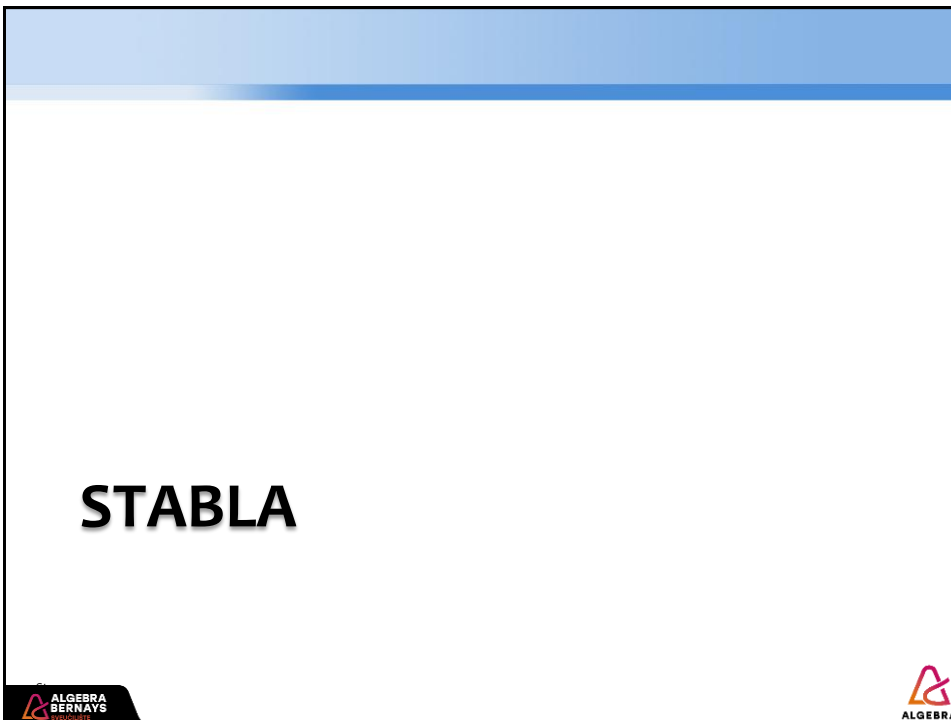


STRUKTURE PODATAKA I ALGORITMI



Predavanje 09

Ishod 3

1



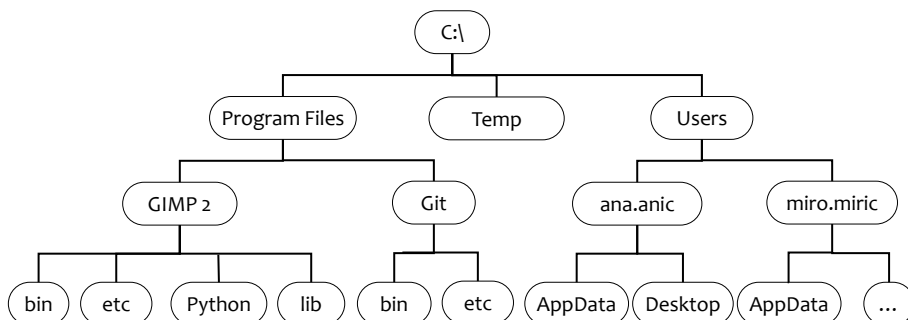
STABLA



2

Uvod

- Sve dosadašnje strukture podataka su bile linearno uređene
 - Možemo li u vektor/listu/stog/red spremiti sljedeće podatke:



- Ne možemo jer su podaci hijerarhijske prirode
 - Potrebna nam je nova struktura – stablo

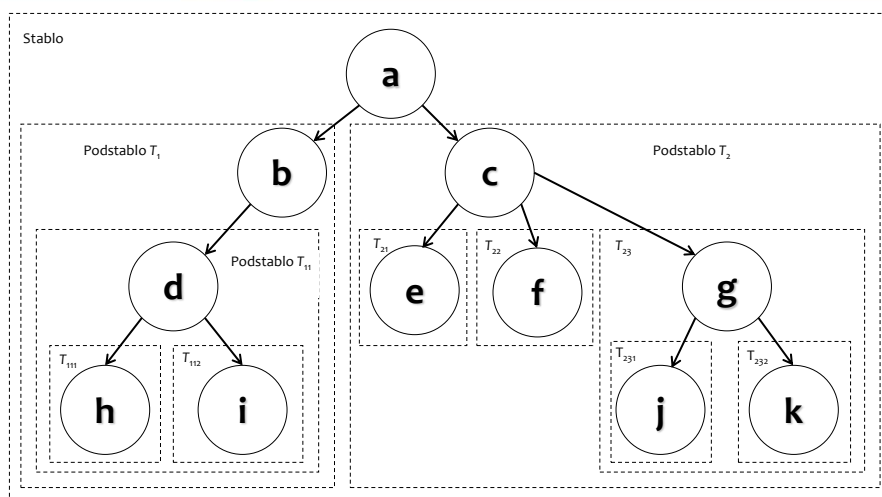
Primjena stabala

- Općenito, stabla su prikladna za:
 - Čuvanje hijerarhijskih podataka
 - Obiteljsko stablo
 - Sportska natjecanja
 - Datotečni sustav
 - Organizacijska shema firme
 - Organizacijska shema vojske
 - Čuvanje podataka u obliku pogodnom za pretraživanje
 - Rječnici
 - Indeksi u bazama podataka

Pojednostavljena definicija stabla

- **Stablo** (engl. *tree*) je skupina povezanih čvorova sa svojstvima:
 - Svaki čvor (engl. *node*) sadrži jednu ili više vrijednosti
 - Čvorovi su hijerarhijski organizirani (roditelj – djeca)
 - Postoji točno jedan čvor koji nema roditelja i koji se naziva **korijen** ili **ishodište stabla** (engl. *tree root*)
 - Svaki čvor je ujedno i **korijen podstabla** (engl. *subtree root*), a to podstablo može biti složeno (sastavljeno od više čvorova) ili trivijalno (sastavljeno samo od 1 čvora)

Primjer stabla



Osnovni pojmovi (1/4)

- Čvorove koji se nalaze direktno ispod nekog čvora nazivamo njegovom **djecom** (engl. *children*)
 - Primjerice, čvorovi **e**, **f** i **g** su djeca čvora **c**
- Osim korijena stabla, svaki čvor ima točno jednog **roditelja** (engl. *parent*), a to je čvor direktno iznad njega
 - Primjerice, roditelj čvora **h** je čvor **d**
 - Svaki čvor može imati više djece, ali najviše jednog roditelja
- Čvorove s istim roditeljem nazivamo **braćom** (engl. *siblings*)
 - Primjerice, čvorovi **e**, **f** i **g** su braća



7

Osnovni pojmovi (2/4)

- **Put** (engl. *path*) od čvora **x** do čvora **y** čini niz čvorova kojima se može direktno doći od **x** do **y** (pri čemu je svaki čvor na putu roditelj sljedećem čvoru na tom putu)
 - Primjerice, put od **a** do **k** je: **a**, **c**, **g**, **k**; put od **e** do **g** ne postoji
- Ako se neki put sastoji od n čvorova, onda je **duljina tog puta** jednaka $n - 1$
 - Primjerice, duljina puta **a**, **c**, **g**, **k** je 3
- Ako gledamo neki čvor **x**:
 - **Potomci** (engl. *descendants*) čvora **x** su svi čvorovi u stablu do kojih postoji put od **x**
 - **Preci** (engl. *ancestors*) čvora **x** su svi čvorovi u stablu od kojih postoji put do **x**



8

Osnovni pojmovi (3/4)

- **List** (engl. *leaf*) je čvor koji nema djece
 - Primjerice, čvorovi **h, i, e, f, j, k** su listovi
- **Unutrašnji** čvor (engl. *internal*) je čvor koji ima djece
 - Primjerice, čvorovi **a, b, c, d, g** su unutarjni
- **Razina** ili **nivo** ili **dubina čvora** (engl. *node level, node depth*) predstavlja njegovu udaljenost (duljinu puta) od korijena
 - Korijen ima razinu 0, njegova djeca imaju razinu 1, njihova djeca razinu 2, itd.
- **Dubina stabla** (engl. *tree depth*) je jednaka maksimalnoj razini nekog čvora u stablu
 - Primjerice, dubina našeg stabla je 3

Osnovni pojmovi (4/4)

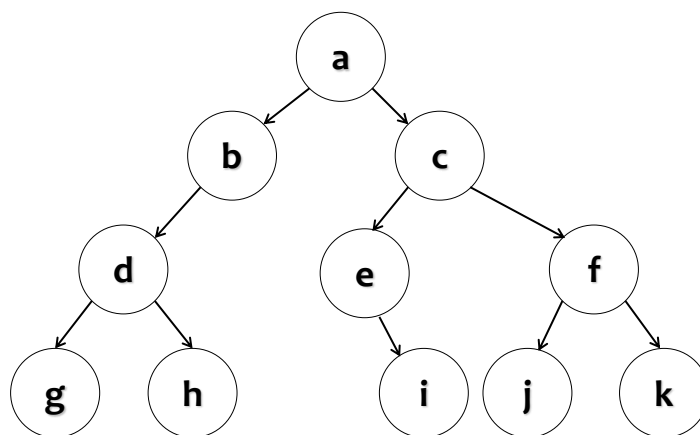
- **Stupanj** čvora je jednak broju njegove djece
 - Primjerice, stupanj od **a** je 2, stupanj od **c** je 3
- **Stupanj stabla** je jednak stupnju čvora s najviše djece
 - Primjerice, stupanj našeg stabla je 3

BINARNA STABLA

Uvod

- **Binarno stablo** (engl. *binary tree*) je stablo čiji stupanj može biti najviše 2
 - To znači da svaki čvor može imati najviše dva djeteta
- Binarna stabla su podskup općenitih stabala
 - Obično je rad s njima jednostavniji od rada s općenitim stablima
- Pojmovi uvedeni za općenita stabla na isti se način koriste i kod binarnih stabala

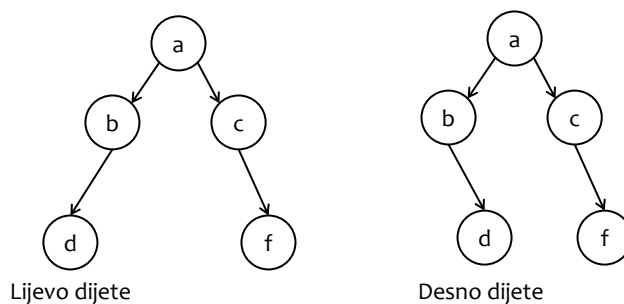
Primjer binarnog stabla



13

Razlika između lijevog i desnog djeteta

- Razlikujemo **lijevo** i **desno** dijete svakog čvora
- Ako neki čvor ima samo jedno dijete, nije svejedno je li ono lijevo ili desno dijete
 - Sljedeća dva binarna stabla nisu jednaka

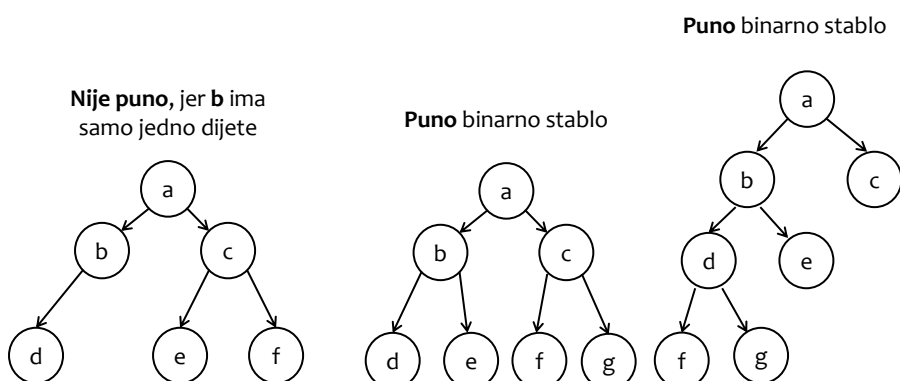


14

Tipovi binarnih stabala

- Zanimaju nas sljedeći tipovi binarnih stabala:
 - **Puno** (engl. *full*) binarno stablo je ono u kojemu svaki čvor koji nije list ima točno 2 djeteta
 - **Savršeno** (engl. *perfect*) binarno stablo je ono koje je **puno** i u kojem su svi listovi u istoj razini
 - **Potpuno** (engl. *complete*) binarno stablo je ono u kojemu su sve razine (osim možda zadnje) **popunjene**, a zadnja razina ima sve čvorove popunjene s lijeva
 - To znači da se čvorovi dodaju u stablo na sljedeći način:
 - Krenemo od korijena i svaku razinu punimo s lijeva na desno dok ima mjesta
 - Kad više nema mjesta, prijedemo na sljedeću razinu

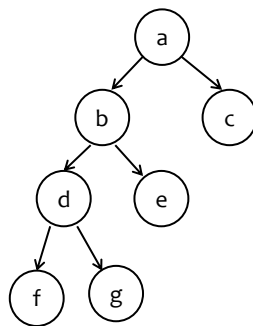
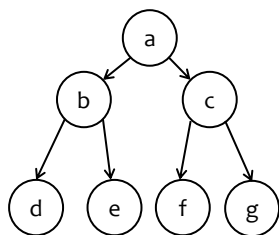
Primjeri punih binarnih stabala



Primjeri savršenih binarnih stabala

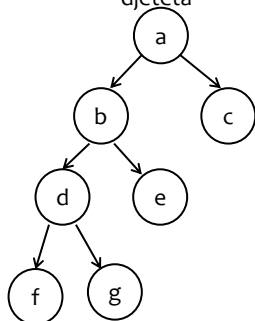
Puno, ali **nije savršeno** jer je razina listova **f** i **g** jednaka 3, lista **e** jednaka 2, a lista **c** jednaka 1

Savršeno binarno stablo

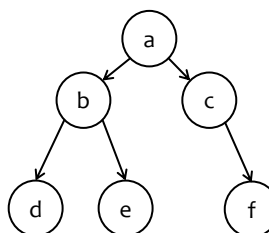


Primjeri potpunih binarnih stabala

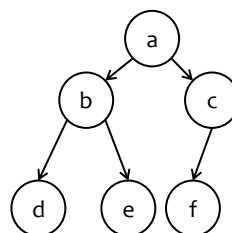
Puno, ali **nepotpuno**, jer **c** i **e** nisu u zadnjem nivou i nemaju oba djeteta



Nepotpuno, jer u zadnjoj razini nisu svi čvorovi popunjeni s lijeva



Potpuno binarno stablo



REKURZIJA

Rekurzija

- Metoda rješavanja problema u kojoj **funkcija poziva samu sebe** s drukčijim vrijednostima parametara
 - Svaki poziv funkcije radi na dijelu problema
 - Mora postojati **uvjet zaustavljanja** (bazni slučaj)
- Svako izvođenje iste funkcije se naziva **iteracija**
 - Svaka iteracija je nezavisna od prethodne
 - U svakoj iteraciji:
 - Riješimo mali dio problema
 - Rekurzivno pozivamo sami sebe kako bismo riješili ostatak problema
 - Pogledajmo

<https://www.cs.usfca.edu/~galles/visualization/RecFact.html>

Primjer rekurzije

- Zadatak svake iteracije jest samo ispisati jedno slovo

```
void ispisi(string ime, int i) {
    if (i == ime.size()) {
        return;
    }
    cout << ime[i] << endl;
    ispisi(ime, i + 1);
}

int main() {
    string ime = "Mirko";
    ispisi(ime, 0);
    return 0;
}
```

Provjera
baznog slučaja

Rješavanje
problema

Rekurzivni
poziv

Inicijalni poziv



21

Zadatak

- Napišimo program koji će ispisati sve podmape unutar zadane mape.
 - Koristit ćemo dirent.h
 - Na linuxu dolazi standardno
 - Za Windowse: github.com/tronkko/dirent
 - Dizajn rekurzivne funkcije:
 - Koji dio problema rješava jedna iteracija?
 - Ispisuje naziv trenutne mape
 - Kakve rekurzivne pozive radimo?
 - Po jedan za svaku podmapu



22

Osnovni kostur rješenja

```

/// <summary>
/// Ispisuje naziv trenutne mape te rekurzivnim pozivima
/// obrađuje podmape.
/// </summary>
/// <param name="roditelj">Putanja do mape roditelja, npr. "C:\\spa\\"</param>
/// <param name="mapa">Naziv trenutne mape, npr. "predavanja"</param>
/// <param name="razina">Koliko treba uvući naziv mape pri ispisu</param>
void process_folder(string roditelj, string mapa, int razina) {

    // Obradi ovaj folder.

    // Napravi punu putanju.

    // Obradi podfoldere.
}

int main() {
    process_folder("C:\\", "Temp", 0);

    return 0;
}

```



23

Detalji rješenja

```

// Obradi ovaj folder.
for (int i = 0; i < razina; i++) {
    cout << " ";
}
cout << mapa << endl;

```



24

Detalji rješenja

```
// Napravi punu putanju.
stringstream sstr;
sstr << roditelj << mapa << "\\";
string mapa_putanja;
sstr >> mapa_putanja;
```



25

Detalji rješenja

```
// Obradi podfoldere.
DIR* dir = opendir(mapa_putanja.c_str());
if (dir == nullptr) { // Možda nemamo prava na ovu mapu.
    return;
}

dirent* ent;
while ((ent = readdir(dir)) != nullptr) {
    if (ent->d_name[0] == '.') {
        continue;
    }

    if (S_ISDIR(ent->d_type) == true) {
        process_folder(mapa_putanja, ent->d_name, razina + 1); //
        // Rekurzivni poziv.
    }
}

closedir(dir);
```



26

OBILASCI STABLA

Uvod

- **Obilazak stabla** (engl. *tree traversal*) jest postupak posjećivanja svih čvorova stabla uz uvjete:
 - Posjetit ćemo svaki element u stablu
 - Niti jedan element nećemo posjetiti dva ili više puta
- Najčešći razlozi posjećivanja su:
 - Čitanje sadržaja elementa
 - Promjena sadržaja elementa

Obilazak linearnih struktura

- Obilazak linearnih struktura (primjerice, liste) je jednostavan: krenemo od prvog elementa i idemo do zadnjeg
- Primjerice, imamo listu od 50 cijelih brojeva
 - Ako želimo izračunati zbroj svih elemenata, proći ćemo od početka do kraja liste i pročitati sadržaj svakog elementa
 - Ako želimo svaki broj pomnožiti s 2, proći ćemo od početka do kraja liste i promijeniti sadržaj svakog elementa



29

Obilazak binarnog stabla

- Obilazak hijerarhijskih struktura je složeniji te postoji više načina njihovog obilaska
- Najpoznatiji algoritmi obilaska binarnog stabla su:
 - DFS algoritmi (engl. *depth-first search*)
 - INORDER
 - PREORDER
 - POSTORDER
 - BFS algoritam (engl. *breadth-first search*)
- Svi algoritmi kreću od korijena, a razlikuju se u redoslijedu posjećivanja čvorova
- Svi algoritmi su rekurzivni



30

DFS algoritmi obilaska

Parent

Left

Right

roditelj

djeca

INORDER	Princip obilaska: Left, Parent, Right Obilazak kreće od lijevog podstabla pa ide na roditelja pa zatim na desno podstablo. Pri tome se svako podstablo obilazi na jednak način (rekurzivnost).
PREORDER	Princip obilaska: Parent, Left, Right Obilazak kreće od roditelja pa ide na lijevo i zatim na desno podstablo. Pri tome se svako podstablo obilazi na jednak način (rekurzivnost).
POSTORDER	Princip obilaska: Left, Right, Parent Obilazak kreće od lijevog podstabla pa ide na desno podstablo i na kraju na roditelja. Pri tome se svako podstablo obilazi na jednak način (rekurzivnost).

Primjene algoritama obilazaka

▪ Primjene algoritama obilazaka:

○ INORDER

- Često se koristi na binarnim stablima traženja (BST) budući da vraća vrijednosti u sortiranom obliku (definiranom samim BST-om)

○ PREORDER

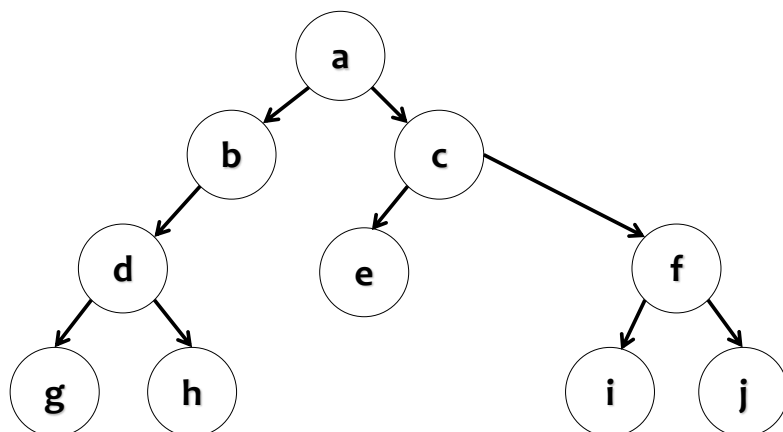
- Često se koristi za dupliciranje stabla jer prvo obradi roditelja, a tek onda djecu

○ POSTORDER

- Često se koristi pri brisanju čvorova i uništenju stabla jer prvo obradi djecu, a tek onda roditelja

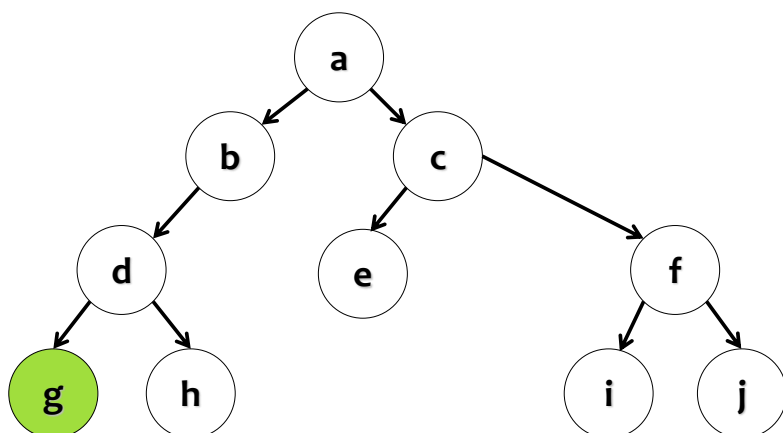
Primjer algoritma INORDER (1/11)

- Primjer stabla kojeg ćemo obići



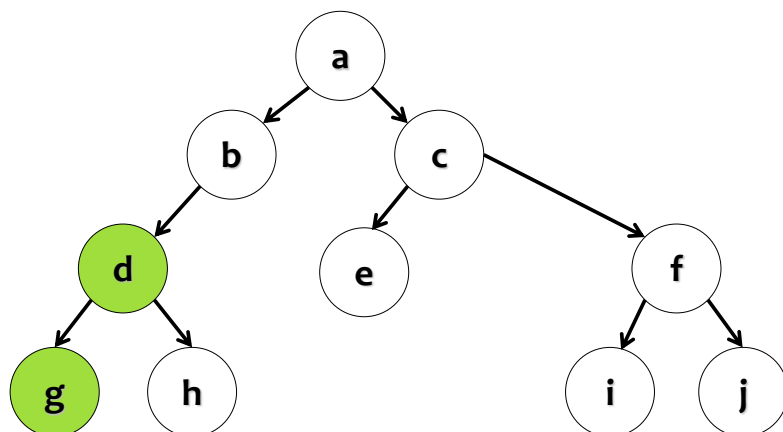
Primjer algoritma INORDER (2/11)

Redoslijed obrade: g



Primjer algoritma INORDER (3/11)

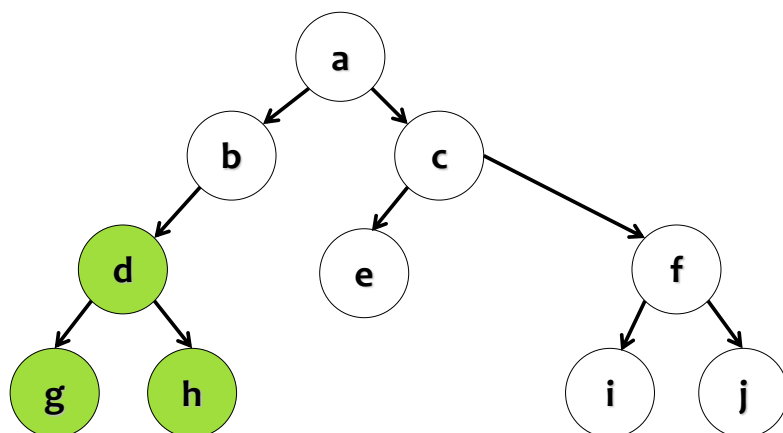
Redoslijed obrade: g, d



35

Primjer algoritma INORDER (4/11)

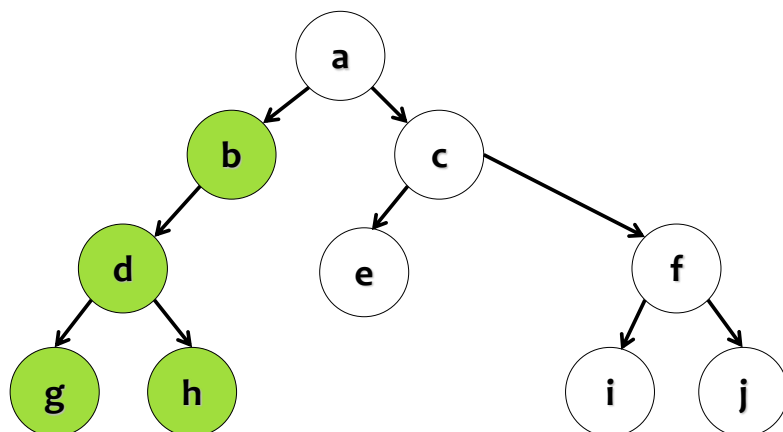
Redoslijed obrade: g, d, h



36

Primjer algoritma INORDER (5/11)

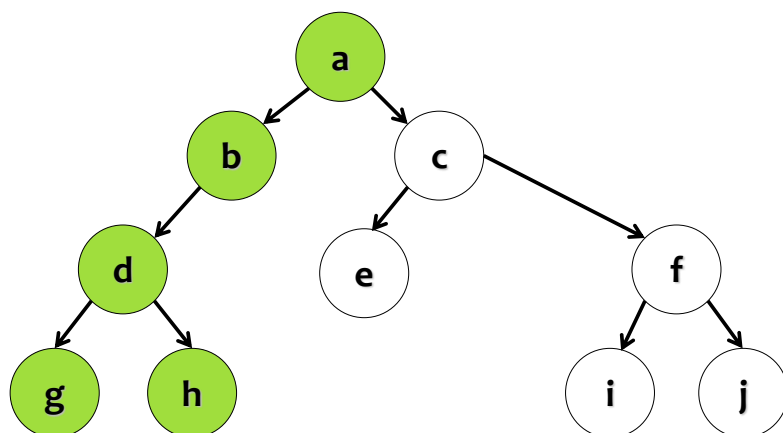
Redoslijed obrade: g, d, h, b



37

Primjer algoritma INORDER (6/11)

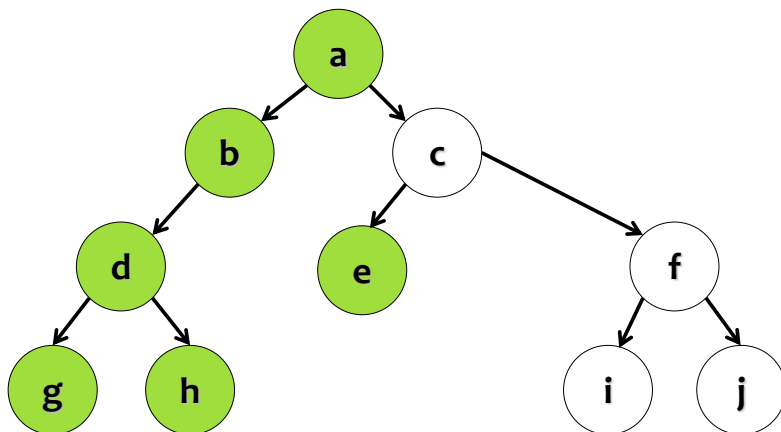
Redoslijed obrade: g, d, h, b, a



38

Primjer algoritma INORDER (7/11)

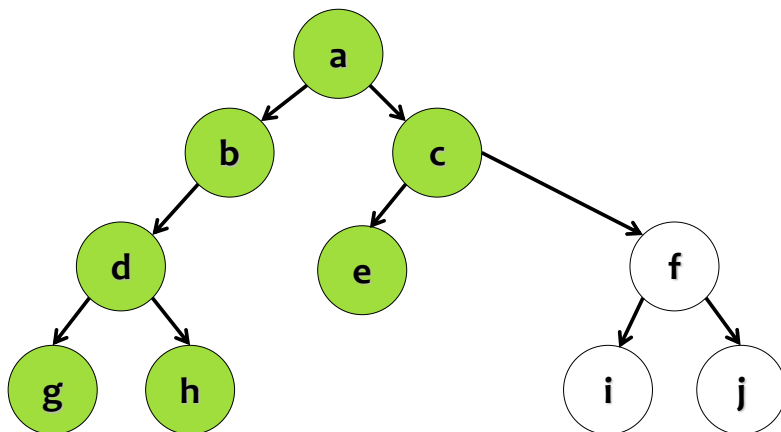
Redoslijed obrade: g, d, h, b, a, e



39

Primjer algoritma INORDER (8/11)

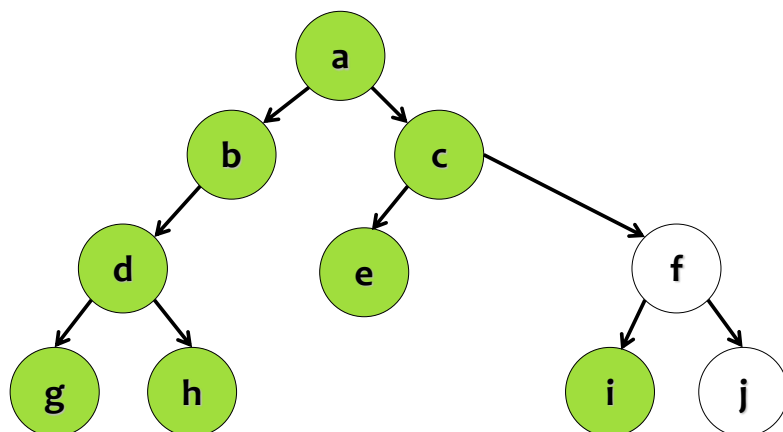
Redoslijed obrade: g, d, h, b, a, e, c



40

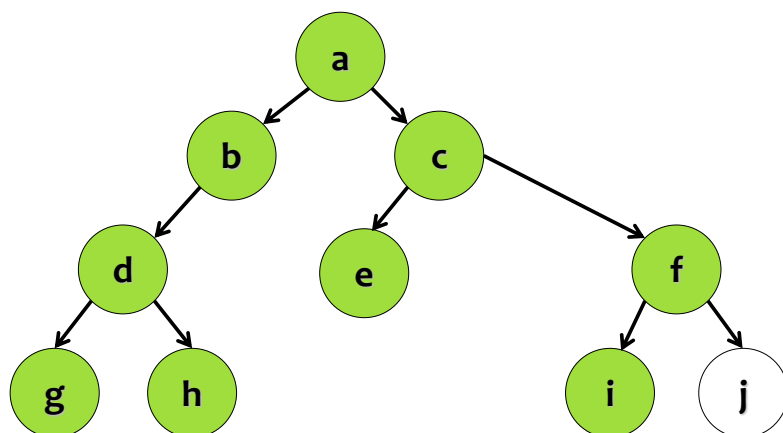
Primjer algoritma INORDER (9/11)

Redoslijed obrade: g, d, h, b, a, e, c, i



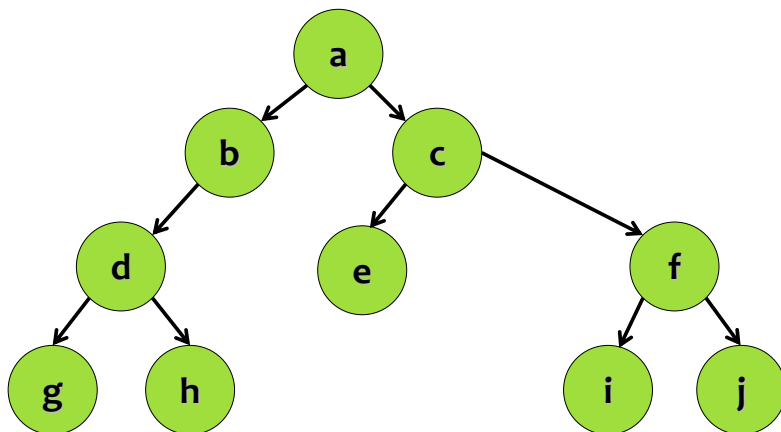
Primjer algoritma INORDER (10/11)

Redoslijed obrade: g, d, h, b, a, e, c, i, f



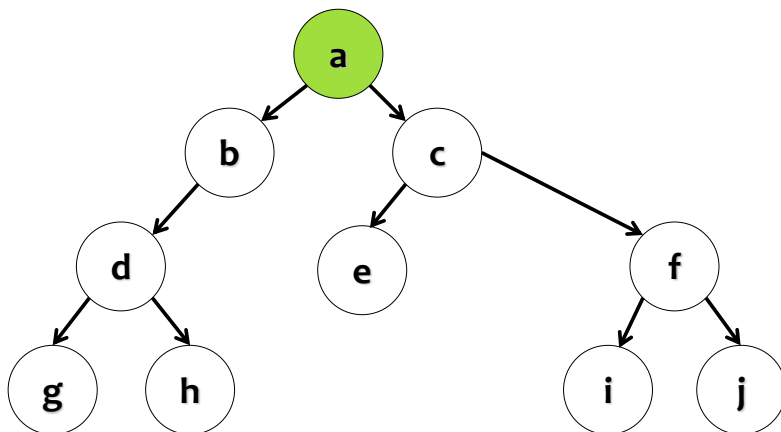
Primjer algoritma INORDER (11/11)

Redoslijed obrade: g, d, h, b, a, e, c, i, f, j



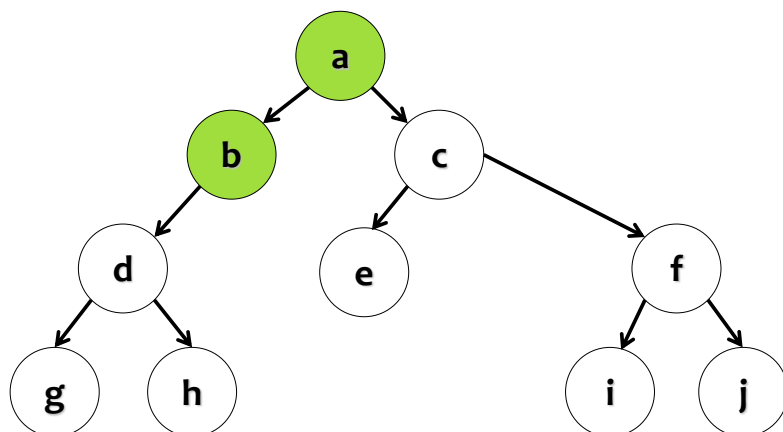
Primjer algoritma PREORDER (1/10)

Redoslijed obrade: a



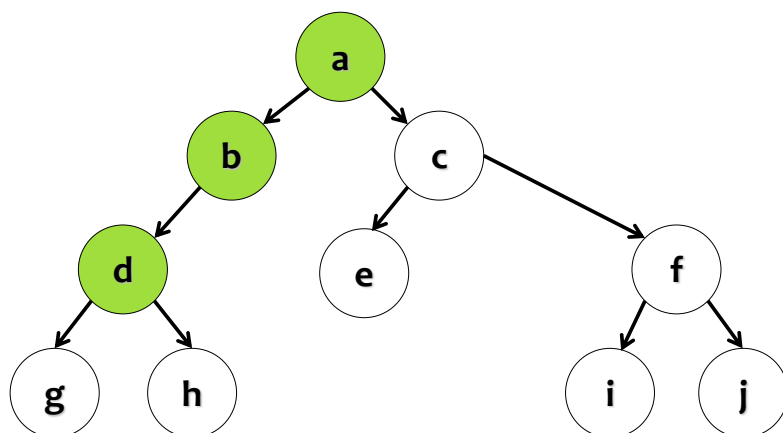
Primjer algoritma PREORDER (2/10)

Redoslijed obrade: a, b



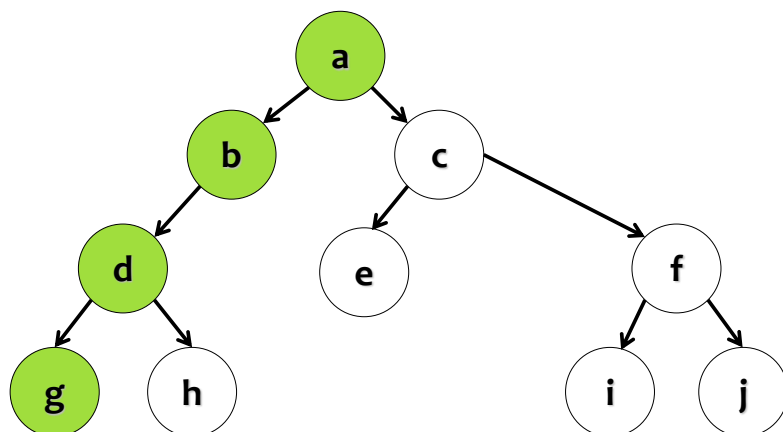
Primjer algoritma PREORDER (3/10)

Redoslijed obrade: a, b, d



Primjer algoritma PREORDER (4/10)

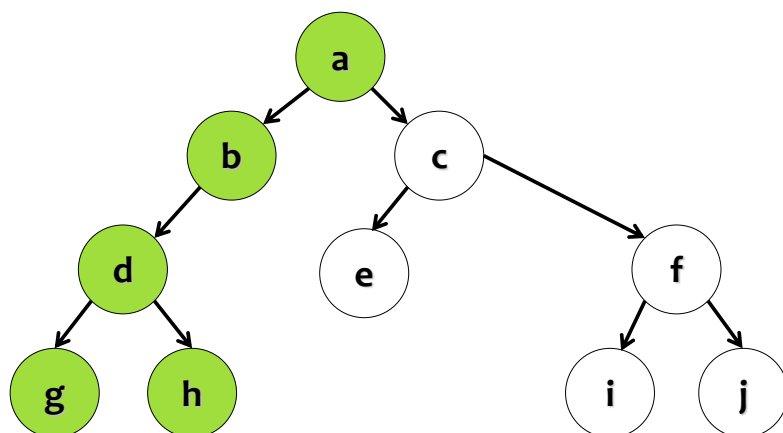
Redoslijed obrade: a, b, d, g



47

Primjer algoritma PREORDER (5/10)

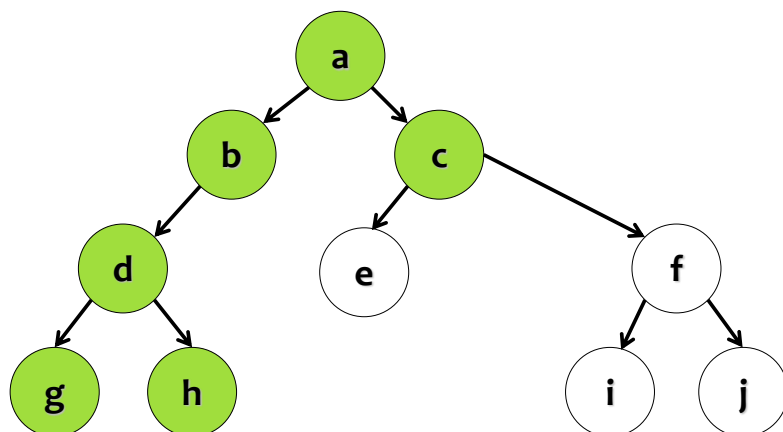
Redoslijed obrade: a, b, d, g, h



48

Primjer algoritma PREORDER (6/10)

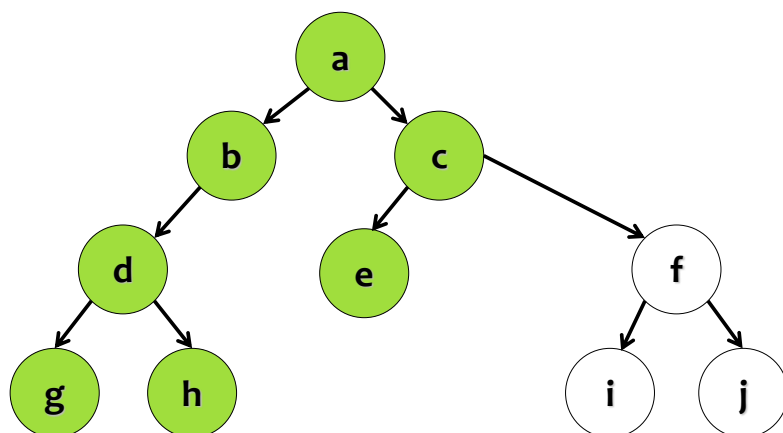
Redoslijed obrade: a, b, d, g, h, c



49

Primjer algoritma PREORDER (7/10)

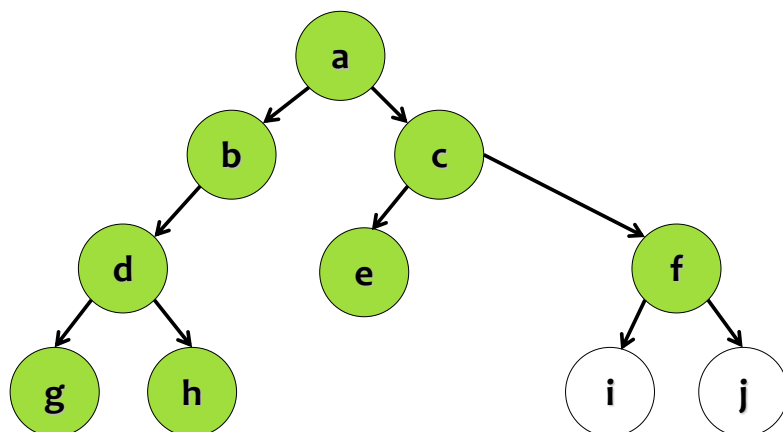
Redoslijed obrade: a, b, d, g, h, c, e



50

Primjer algoritma PREORDER (8/10)

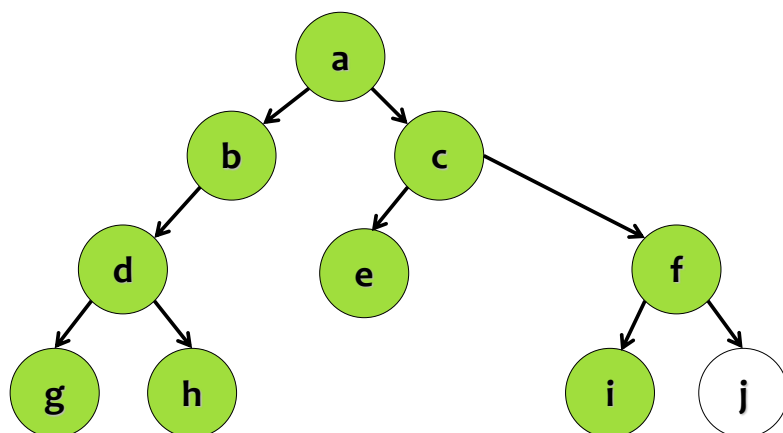
Redoslijed obrade: a, b, d, g, h, c, e, f



51

Primjer algoritma PREORDER (9/10)

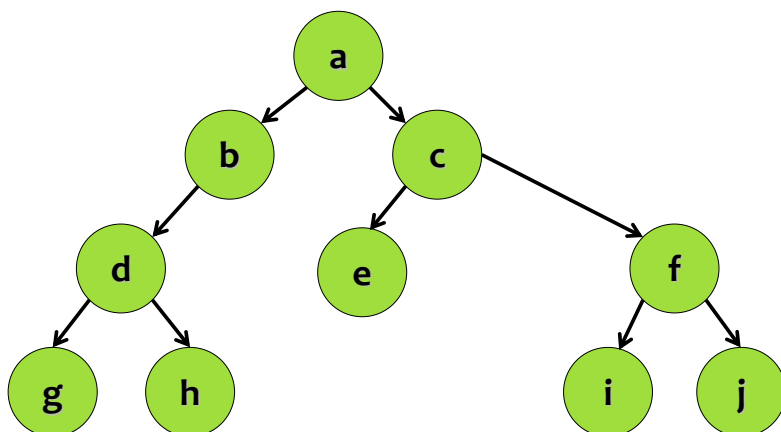
Redoslijed obrade: a, b, d, g, h, c, e, f, i



52

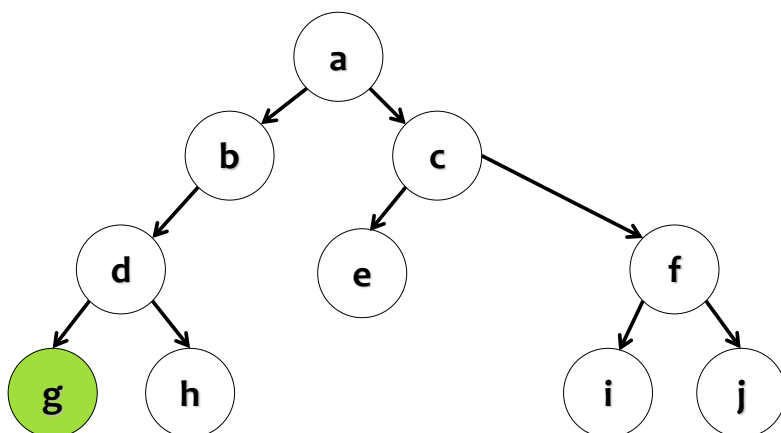
Primjer algoritma PREORDER (10/10)

Redoslijed obrade: a, b, d, g, h, c, e, f, i, j



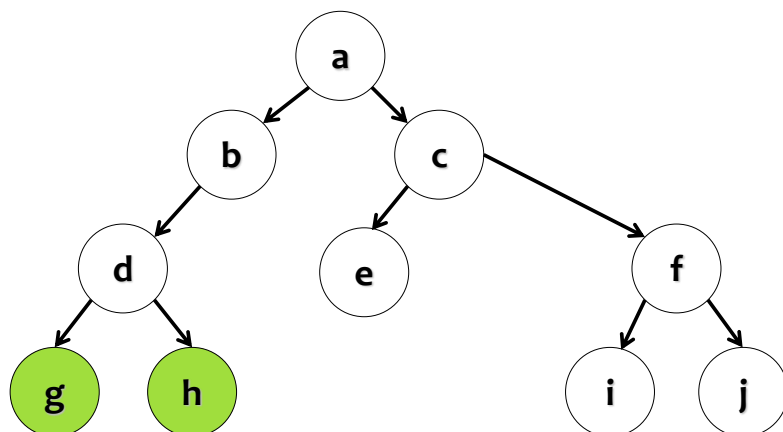
Primjer algoritma POSTORDER (1/10)

Redoslijed obrade: g



Primjer algoritma POSTORDER (2/10)

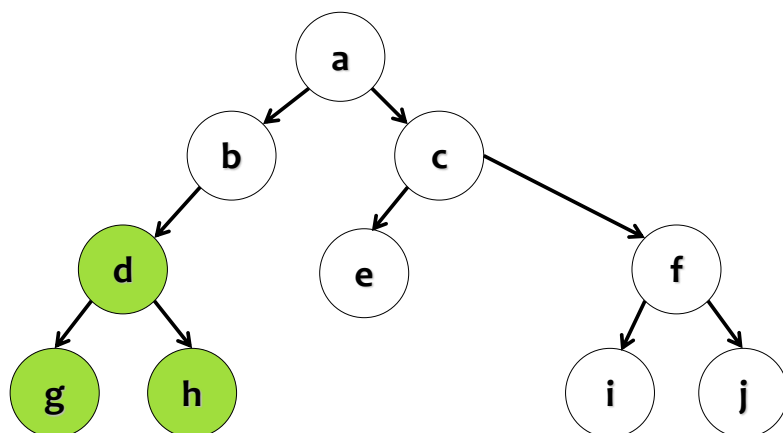
Redoslijed obrade: g, h



55

Primjer algoritma POSTORDER (3/10)

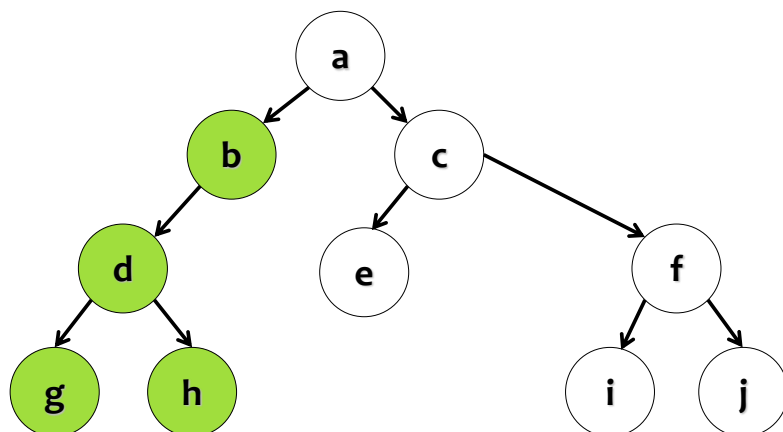
Redoslijed obrade: g, h, d



56

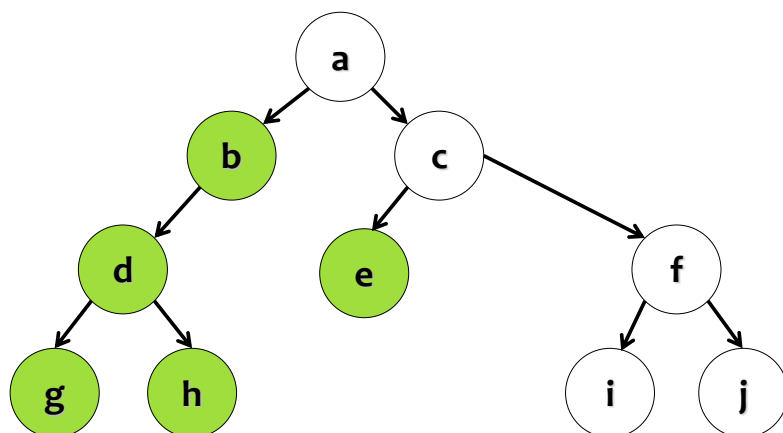
Primjer algoritma POSTORDER (4/10)

Redoslijed obrade: g, h, d, b



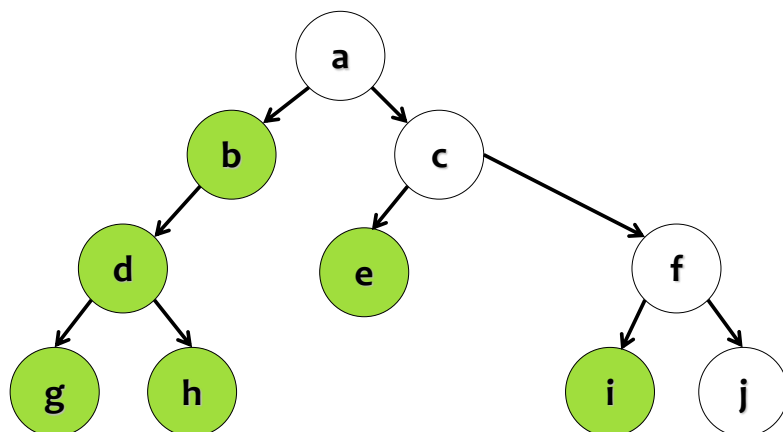
Primjer algoritma POSTORDER (5/10)

Redoslijed obrade: g, h, d, b, e



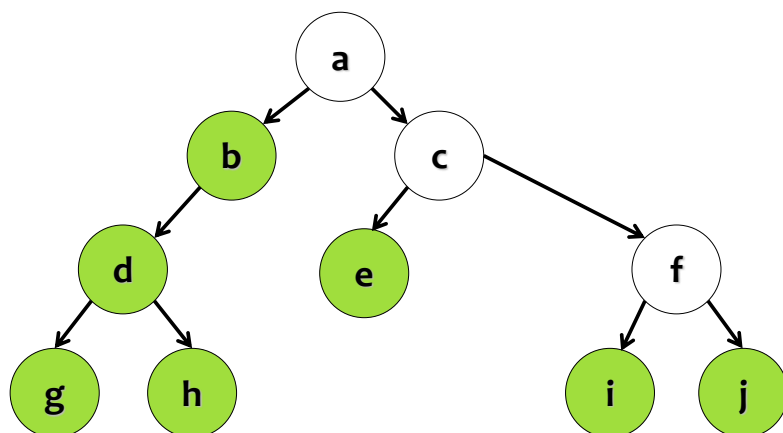
Primjer algoritma POSTORDER (6/10)

Redoslijed obrade: g, h, d, b, e, i



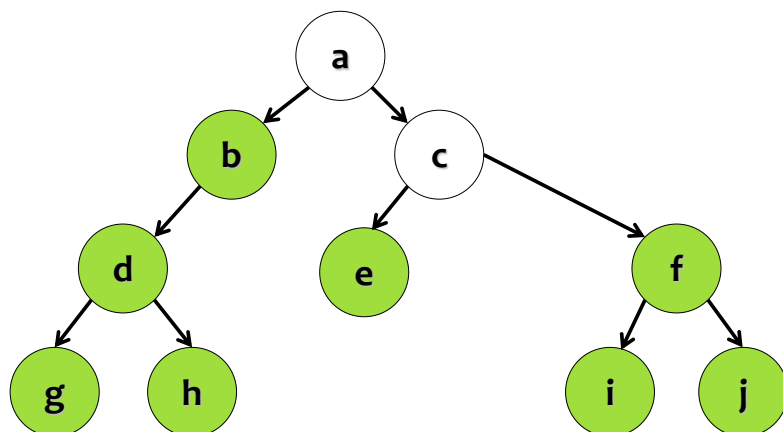
Primjer algoritma POSTORDER (7/10)

Redoslijed obrade: g, h, d, b, e, i, j



Primjer algoritma POSTORDER (8/10)

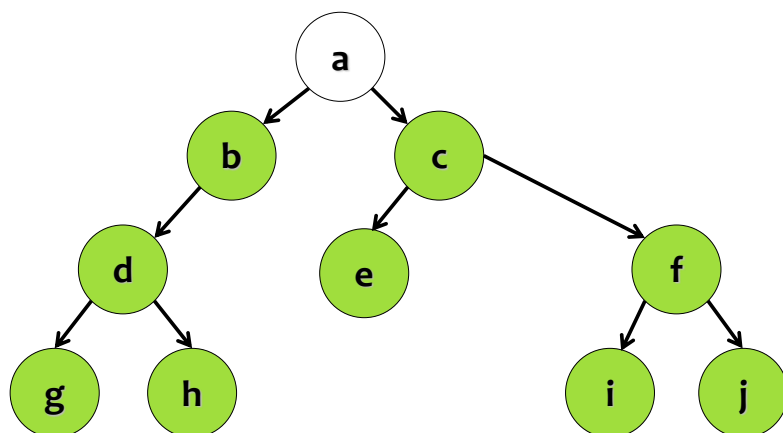
Redoslijed obrade: g, h, d, b, e, i, j, f



61

Primjer algoritma POSTORDER (9/10)

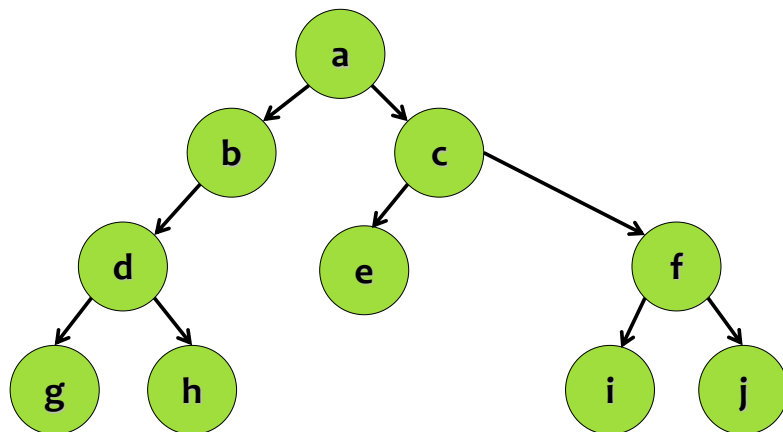
Redoslijed obrade: g, h, d, b, e, i, j, f, c



62

Primjer algoritma POSTORDER (10/10)

Redoslijed obrade: g, h, d, b, e, i, j, f, c, a



63

Redoslijed obrade sva tri algoritma

- Redoslijed obrade čvorova INORDER:
 - g, d, h, b, a, e, c, i, f, j
- Redoslijed obrade čvorova PREORDER:
 - a, b, d, g, h, c, e, f, i, j
- Redoslijed obrade čvorova POSTORDER:
 - g, h, d, b, e, i, j, f, c, a

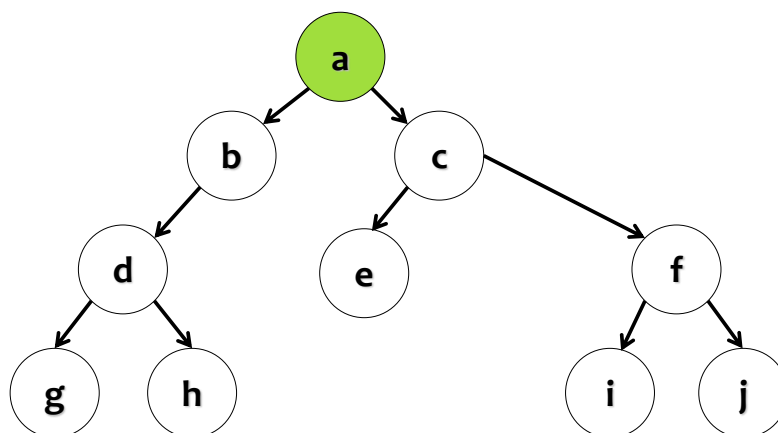
64

BFS algoritam

- BFS algoritam posjećuje čvorove razinu po razinu, s lijeva na desno:
 1. Dodaj korijen u red
 2. Uzmi sljedeći element A iz reda i ispiši vrijednost*
 3. Dodaj djecu čvora A u red
 4. Ako red nije prazan, idi na korak 2
- Primjerice, ako stablo prikazuje hijerarhiju, onda ovaj način obilaska prvo ispisuje one pri vrhu hijerarhije
- Primjena ovog algoritma nije tako česta u praksi

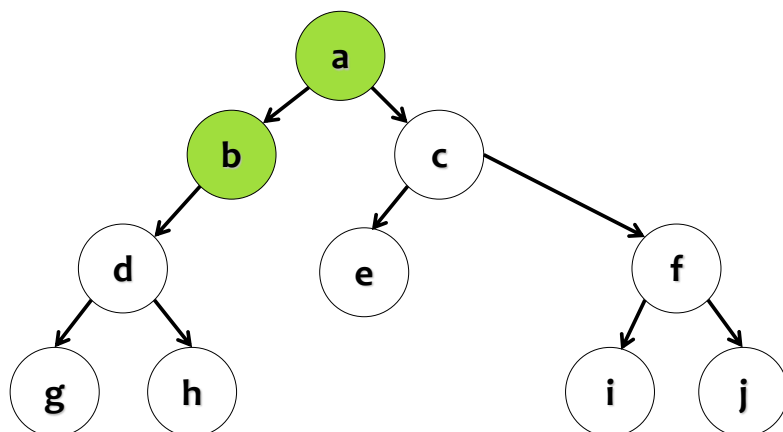
Primjer algoritma BFS (1/10)

Redoslijed obrade: a



Primjer algoritma BFS (2/10)

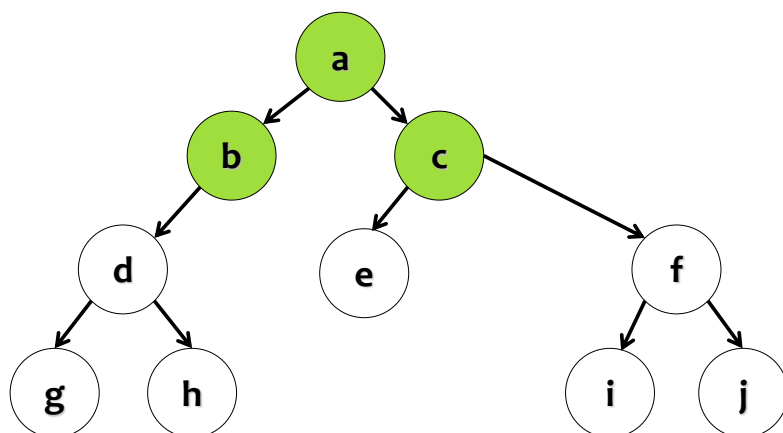
Redoslijed obrade: a, b



67

Primjer algoritma BFS (3/10)

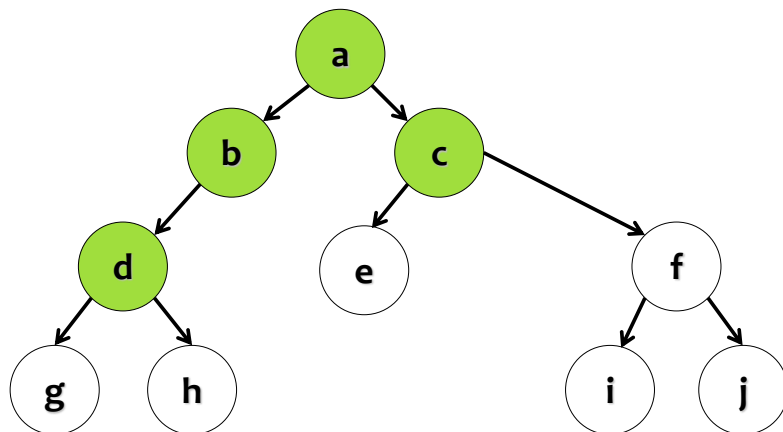
Redoslijed obrade: a, b, c



68

Primjer algoritma BFS (4/10)

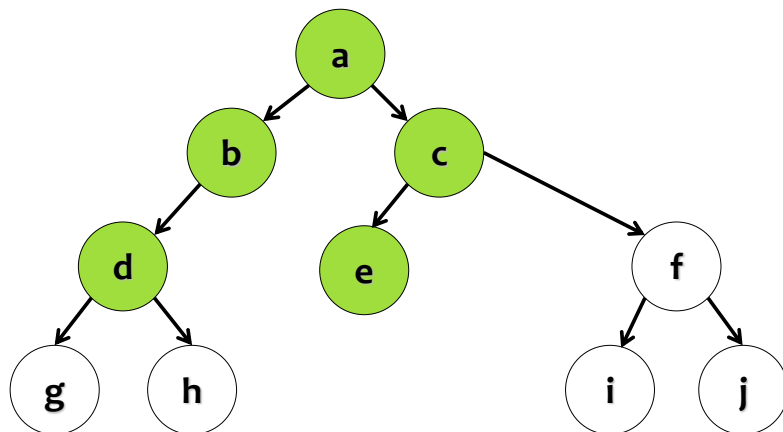
Redoslijed obrade: a, b, c, d



69

Primjer algoritma BFS (5/10)

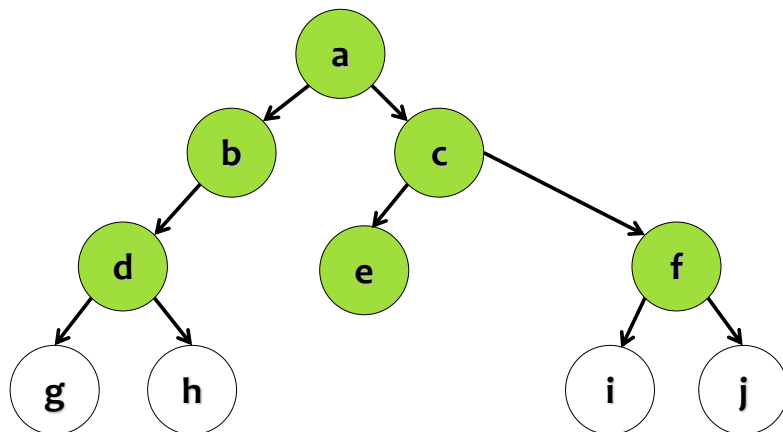
Redoslijed obrade: a, b, c, d, e



70

Primjer algoritma BFS (6/10)

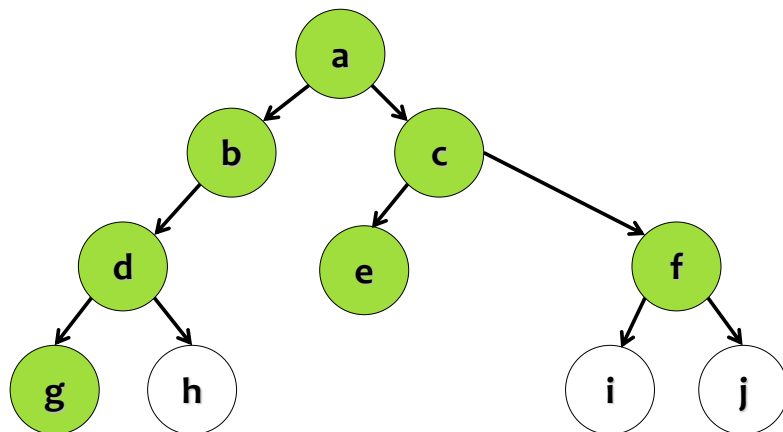
Redoslijed obrade: a, b, c, d, e, f



71

Primjer algoritma BFS (7/10)

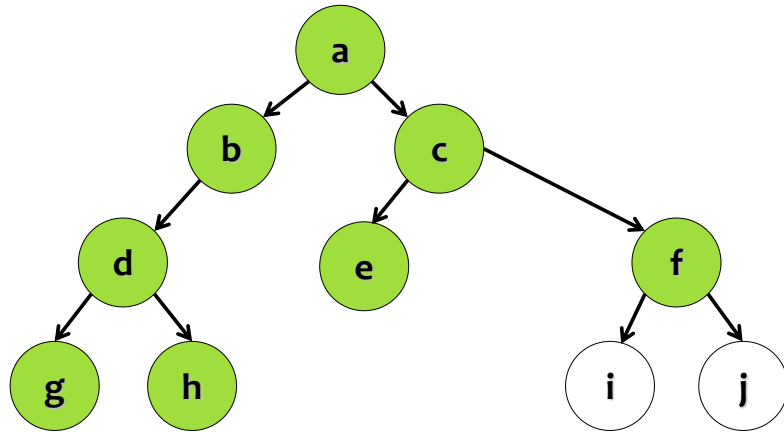
Redoslijed obrade: a, b, c, d, e, f, g



72

Primjer algoritma BFS (8/10)

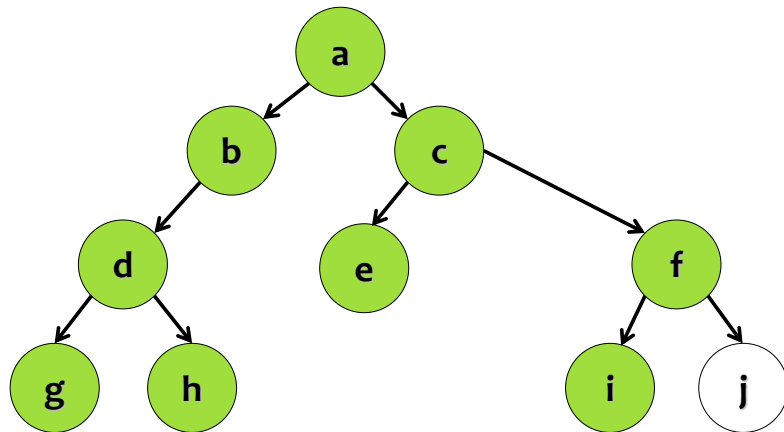
Redoslijed obrade: a, b, c, d, e, f, g, h



73

Primjer algoritma BFS (9/10)

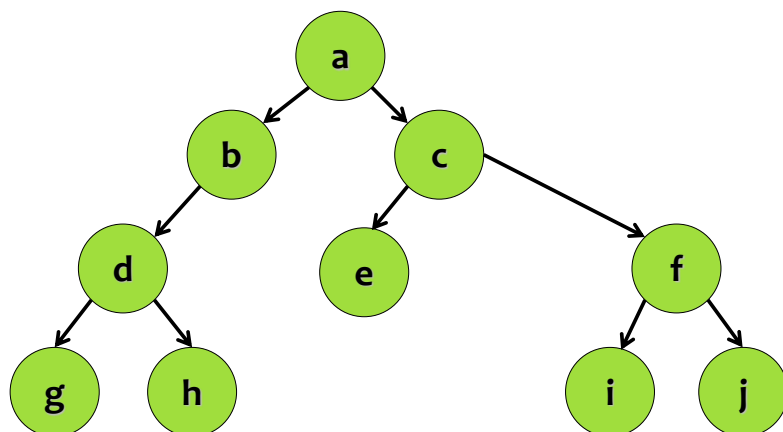
Redoslijed obrade: a, b, c, d, e, f, g, h, i



74

Primjer algoritma BFS (10/10)

Redoslijed obrade: a, b, c, d, e, f, g, h, i, j



IMPLEMENTACIJA STABLA

Uvod

- STL ne sadrži „čistu” implementaciju stabla
 - Stablo se interno koristi kao podrška za neke kontejnere
- Dobra implementacija je na <https://github.com/kpeeters/tree.hh>
- Mi ćemo koristiti našu jednostavnu implementaciju
 - Nedostaje destruktork zbog jednostavnosti



77

Source.cpp

```

btree stablo("a");

node* korijen = stablo.root();
stablo.insert_left(korijen, "b");
stablo.insert_right(korijen, "c");

node* cvor_b = stablo.get_left_child(korijen);
stablo.insert_left(cvor_b, "d");

node* cvor_c = stablo.get_right_child(korijen);
stablo.insert_left(cvor_c, "e");
stablo.insert_right(cvor_c, "f");

node* cvor_d = stablo.get_left_child(cvor_b);
stablo.insert_left(cvor_d, "g");
stablo.insert_right(cvor_d, "h");

node* cvor_f = stablo.get_right_child(cvor_c);
stablo.insert_left(cvor_f, "i");
stablo.insert_right(cvor_f, "j");

stablo.bfs(korijen);

```



78

btree.h

```
#pragma once
#include <iostream>
#include <string>
#include <queue>
using namespace std;

struct node {
    string element;
    node* left_child;
    node* right_child;
};

class btree {
private:
    node* root_node;
    node* create_new_node(string element);
public:
    btree(string element);
    void insert_left(node* parent, string element);
    void insert_right(node* parent, string element);
    node* root();
    node* get_left_child(node* parent);
    node* get_right_child(node* parent);
    void inorder(node* parent);
    void preorder(node* parent);
    void postorder(node* parent);
    void bfs(node* parent);
```



79

btree.cpp

```
#include "btree.h"

node* btree::create_new_node(string element) {
    node* novi = new node;
    novi->element = element;
    novi->left_child = nullptr;
    novi->right_child = nullptr;
    return novi;
}

btree::btree(string element) {
    root_node = create_new_node(element);
}

void btree::insert_left(node* parent, string element) {
    parent->left_child = create_new_node(element);
}

void btree::insert_right(node* parent, string element) {
    parent->right_child = create_new_node(element);
}
```



80

btree.cpp

```
node* btree::root() {
    return root_node;
}

node* btree::get_left_child(node* parent) {
    return parent->left_child;
}

node* btree::get_right_child(node* parent) {
    return parent->right_child;
}
```



81

btree.cpp

```
void btree::inorder(node* parent) {
    if (parent == nullptr) {
        return;
    }

    inorder(parent->left_child);
    cout << parent->element;
    inorder(parent->right_child);
}

void btree::preorder(node* parent) {
    if (parent == nullptr) {
        return;
    }

    cout << parent->element;
    preorder(parent->left_child);
    preorder(parent->right_child);
}
```



82

btree.cpp

```
void btree::postorder(node* parent) {
    if (parent == nullptr) {
        return;
    }

    postorder(parent->left_child);
    postorder(parent->right_child);
    cout << parent->element;
}

void btree::bfs(node* parent) {
    queue<node*> q;
    q.push(root_node);

    while (!q.empty()) {
        node* current = q.front();
        cout << current->element;
        if (current->left_child != nullptr) {
            q.push(current->left_child);
        }
        if (current->right_child != nullptr) {
            q.push(current->right_child);
        }
        q.pop();
    }
}
```



83

Dodatni materijali

- Dodatni materijali su dostupni na:
 - Basic terms
 - <https://youtu.be/7DENJi3UoZw>
 - Binary trees
 - <https://youtu.be/iXOJ561Roaw>
 - Tree traversals
 - <https://youtu.be/e9kiaQzFrE>
 - Implementing your own tree
 - <https://youtu.be/lrOk-TR4W5E>



84