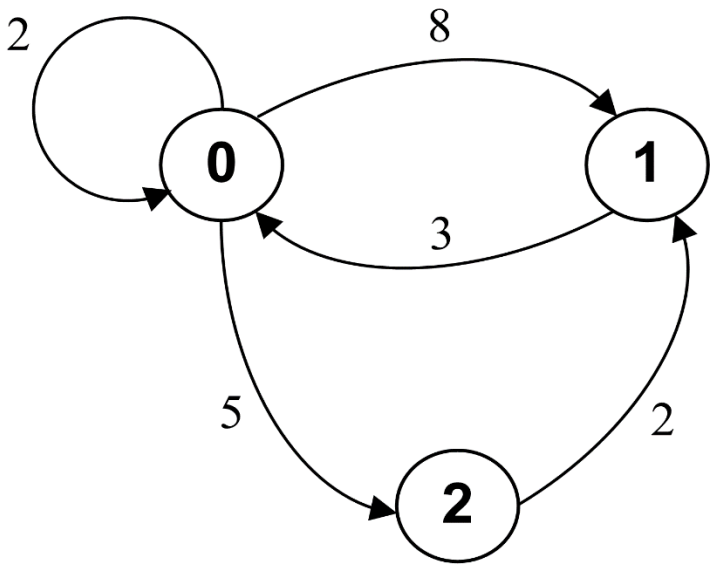
Programiranje I

Grafovi – Problem najkraćeg puta
Stabla

Ilustracija Dijkstra algoritma

- Na primjeru jednostavnog grafa ilustrujmo korake u Dijkstra algoritmu:



Graf

l[I][J]	0	1	2
0	2	8	5
1	3	∞	∞
2	∞	2	∞

Matrica sa upisanim cijenama ivica.

Dijkstra algoritam - Primjer

- Postavimo nule na glavnu dijagonalu.

$C_0[I][J]$	0	1	2
0	0	8	5
1	3	0	∞
2	∞	2	0

- Najkraće putanje preko čvora 0.

$C_1[I][J]$	0	1	2
0	0	8	5
1	3	0	8
2	∞	2	0

Jedina pozicija gdje se dogodila promjena je označena crvenom bojom

Dijkstra algoritam - Primjer

- Najkraće putanje preko čvorova 0 i 1.

$C_2[I][J]$

	0	1	2
0	0	8	5
1	3	0	8
2	5	2	0

- Najkraće putanje preko čvorova 0, 1 i 2, odnosno najkraće (najjeftinije) moguće putanje.

$C_3[I][J]$

	0	1	2
0	0	7	5
1	3	0	8
2	5	2	0

Dijkstra algoritam - Realizacija

```
#include <stdio.h>
#define DIM 3
int min(int, int);
int main()
{
        int ms[][DIM] = {{2,8,5},{3,1000,1000},{1000,2,1000}};
        int N = DIM, i, j, k;
        int cs[DIM][DIM], cn[DIM][DIM];
        for (i=0; i<N; i++)
                for(j=0; j<N; j++)
                        cs[i][j] = ms[i][j];
        for(i=0; i<N; i++)
                cs[i][i] = 0;
```

Matrica sa cijenama putanja; na pozicijama gdje ne postoji veza postavljen je veliki cijeli broj kao zamjena za beskonačno.

Najjeftinije je ne kretati se iz čvora i u čvor i.

Dijkstra algoritam - Realizacija

```
for(k=0;k<N;k++) {
    for(i=0;i<N;i++)
        for(j=0;j<N;j++)
            cn[i][j] = min(cs[i][j], cs[i][k] + cs[k][j]);
        for(i=0;i<N;i++)
            for(j=0;j<N;j++)
                cs[i][j] = cn[i][j];
}
for(i=0;i<N;i++)
    for(j=0;j<N;j++)
printf("\nNajjeftiniji put izmedju %d i %d ima cijenu %d", i, j, cs[i][j]);
}
int min(int a, int b){ return a>b ? b : a;}
```

petlja po k (po čvorovima posrednicima)

ključni korak u algoritmu u dvostrukoj petlji koja provjerava da li se jeftinija konekcija između i i j može obaviti posredstvom čvora k.

priprema za narednu iteraciju

štampanje rezultata

pomoćna fukcija

Modifikacija Dijkstra algoritma

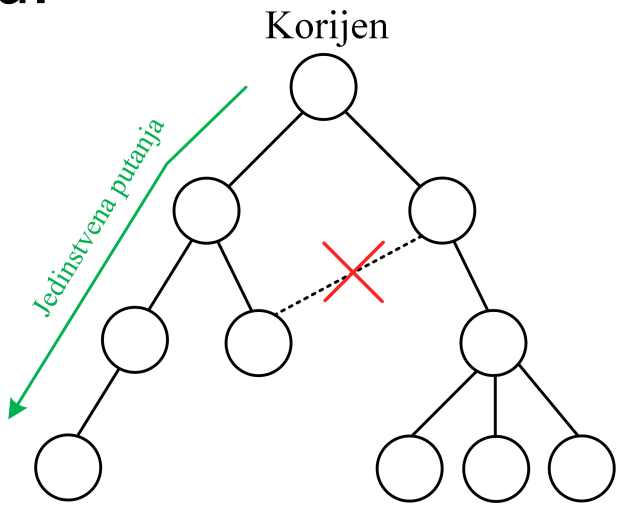
- Pretpostavimo da je zadata matrica susjedstva (a ne matrica cijena ivica).
- Neka je naš zadatak da formiramo matricu koja će imati vrijednost **1** na poziciji **[i][j]** ako između **i** i **j** postoji konekcija preko proizvoljno mnogo ivica i **0** ako ne postoji takva konekcija.
- Predmetni algoritam je modifikacija Dijkstra algoritma u tom smislu što je jedini korak kojeg treba izmijeniti tzv. ključni korak u algoritmu, koji sada može da glasi:

```
cn[i][j] = cs[i][j] || (cs[i][k] && cs[k][j]);
```

- Ovaj korak se sada može tumačiti kao: veza između čvorova postoji ako postoji direktna veza ili veza posredstvom nekog čvora (odnosno, kako k ide od 0 do N-1, preko proizvoljno mnogo čvorova).

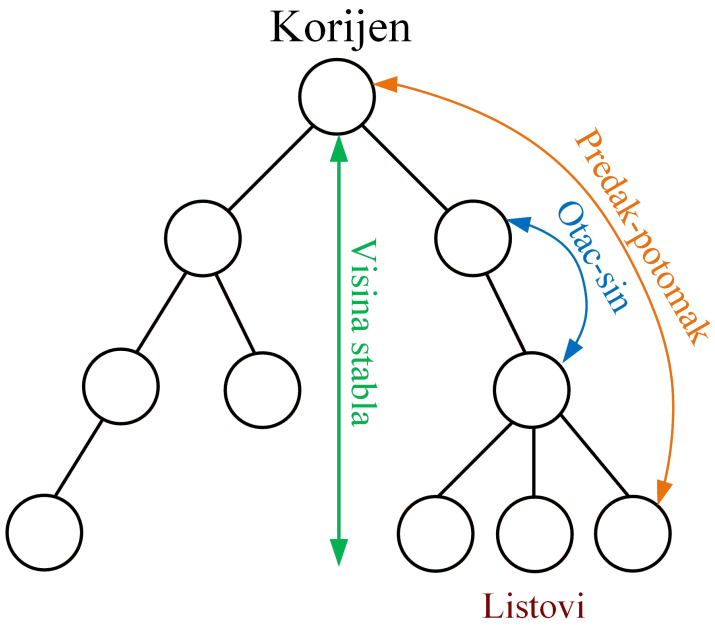
Stablo (drvo)

- Još jedan, moramo priznati, čudan naziv u programiranju.
- **Stablo** ili **drvo** je veoma napredan i veoma korišćen tip podataka.
- **Stablo** je usmjereni aciklični graf (graf u kome ne postoji ciklus) koji zadovoljava sljedeća svojstva:
  - Jedan čvor, koji se naziva korijen, nije kraj nijedne ivice;
  - Svakom čvoru, osim korijena, odgovora tačno jedna ivica čiji je kraj taj čvor;
  - Postoji jedinstvena putanja od korijena ka svakom čvoru drveta.



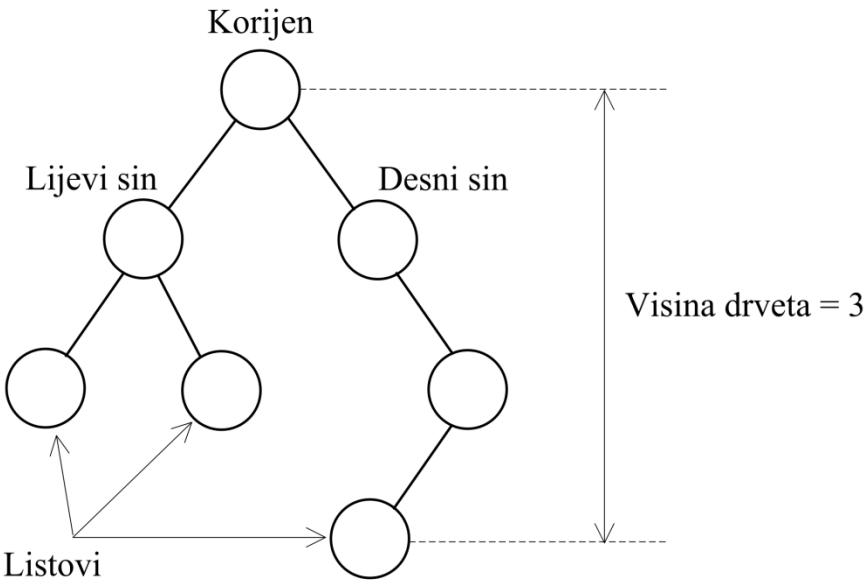
Stablo – Pojmovi

- Stablo je predstavljeno kao uređeni par čvorova i ivica: **T=(V,E)**. Ako (v,w)∈E, tada se kaže da je v **otac** čvoru w, odnosno da je w **sin** čvora v. Ako postoji putanja od v ka w preko proizvoljno mnogo ivica, kaže se da je v **predak** čvora w, odnosno da je w **potomak** čvora v. Čvorovi bez potomaka se nazivaju **listovima**. Čvor v i njegovi potomci čine podstablo čiji je korijen v.
- **Visina stabla** je dužina najdužeg puta od korijena ka listovima.



Binarno stablo

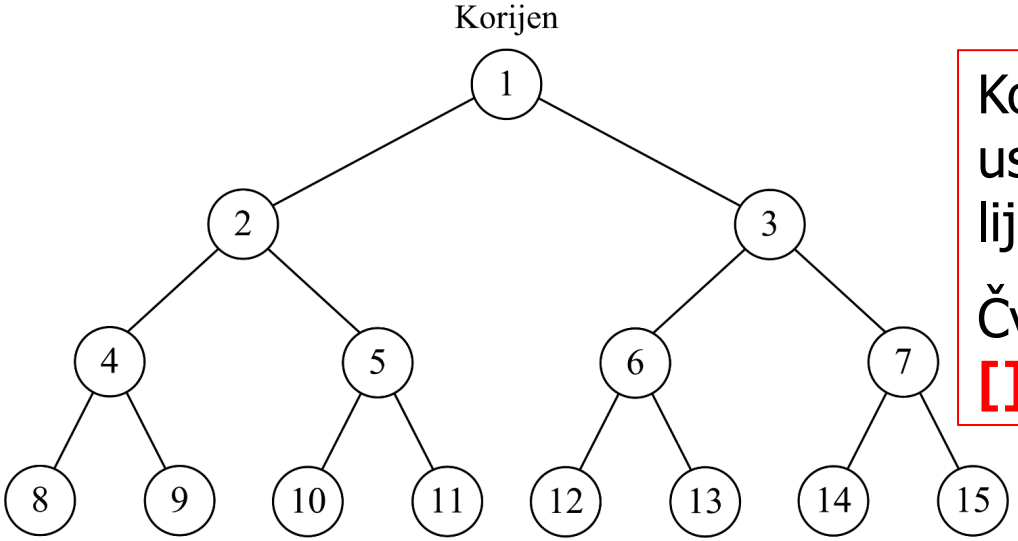
- Binarno stablo je specijalni i najčešće korišćeni tip stabla. U njemu svaki čvor ne može da ima više od dva sina od kojih se jedan naziva lijevi sin, a drugi desni sin (ovo implicira moguće postojanje lijevog i desnog podstabla).



Vizuelizacija jednog binarnog stabla sa ilustracijom nekih od važnih pojmova (korijen, listovi, lijevi i desni sin, i visina stabla).

Potpuno binarno stablo

- Kod **potpunog binarnog stabla**, svi čvorovi osim listova imaju oba sina i svi listovi su na istom rastojanju od korijena.
- Potpuno binarno stablo visine 3 ima 15 čvorova. Koliko čvorova ima potpuno binarno stablo visine 4? Koliko stablo visine n?

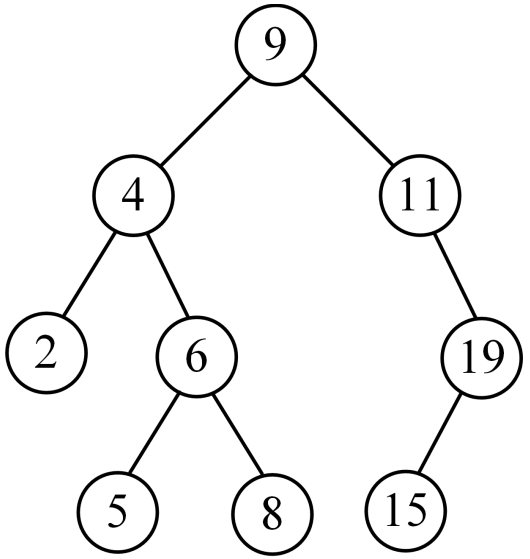


Kod potpunog binarnog stabla usvaja se konvencija da čvor **I** ima lijevog sina **2I** i desnog sina **2I+1**.

Čvoru **J** otac je čvor **[J/2]**, gdje je **[]** operator zaokruživanja nadole.

Binarno stablo pretrage

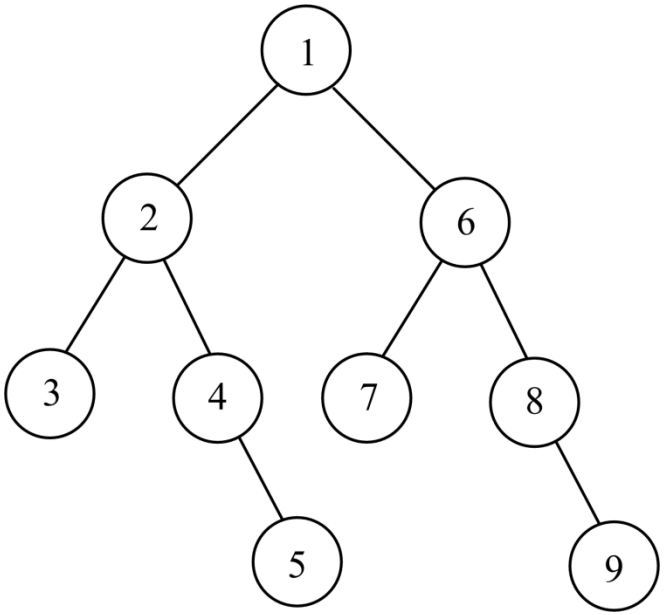
- Kod binarnog stabla pretraga (BSP), poznatog i kao sortirano binarno stablo, vrijednost svakog čvora je veća od vrijednosti svakog čvora u njegovom lijevom podstablu i manja od vrijednosti svakog čvora u njegovom desnom podstablu.
- Postoje realizacije BSP-a sa i bez ponavljanja elemenata.
- Novi čvorovi se dodaju kao listovi.
- Prilikom brisanja, može se desiti nekoliko različitih situacija. Istražite sami.
- BSP ostaje sortirano prilikom dodavanja i brisanja čvorova, što omogućava bržu pretragu nego većina drugih struktura.
- Složenost pretrage je O(V), gdje je V visina stabla.



Predstava stabla preko podnizova

- Čvorovi stabla se mogu predstaviti preko struktura. Kako se mogu predstaviti veze između čvorova stabla?
- Jedan od načina predstavljanja stabla je preko nizova LEFTSON i RIGHTSON. Ovi nizovi imaju elemenata koliko je čvorova u stablu.
- **LEFTSON[I]** predstavlja lijevi sin čvora **I**, dok **RIGHTSON[I]** predstavlja desni sin čvora **I**. U slučaju da čvor **I** nema nekog od sinova u odgovorajućem nizu se može upisati **0** (u C-u je pogodniji upis **–1**, pošto numerisanje čvorova može da se obavi kao kod nizova, od **0** pa na dalje).

Predstavljanje preko podnizova



	LEFTSON	RIGHTSON
1	2	6
2	3	4
3	0	0
4	0	5
5	0	0
6	7	8
7	0	0
8	0	9
9	0	0

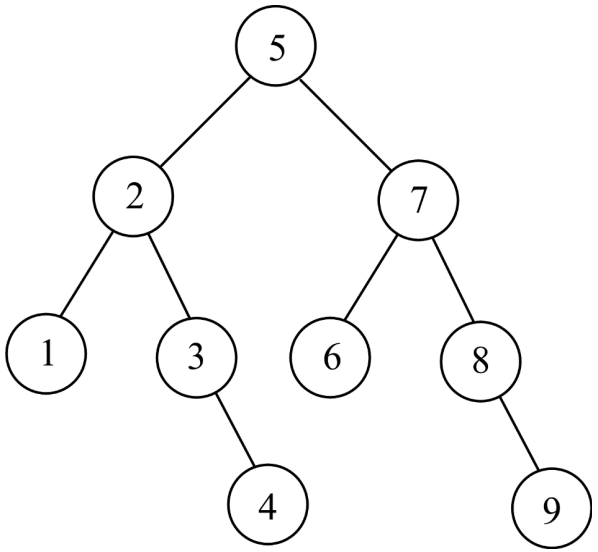
- **Tumačenje**: Čvor 1 ima dva sina od kojih je 2 lijevi, a 6 desni; čvor 2 ima oba sina i to čvor 3 lijevi i čvor 4 desni; čvor 3 je list (nema sinova) itd.

Obilazak stabla

- Algoritmi nad stablom često podrazumijevaju da se svaki čvor stabla mora obići jednom. Stoga se mora definisati jedinstvena metodologija za obilazak stabla (eng. tree traversal).
- Postoje tri metodologije obilaska:
  - inorder (srednji redosljed)
  - preorder
  - postorder

Obilazak po srednjem redosljedu

- Obilazak stabla po srednjem redosljedu (inorder) se obavlja:
  - prvo se obiđe lijevo podstablo,
  - zatim se obiđe korijen podstabla i
  - na kraju se obiđe desno podstablo.



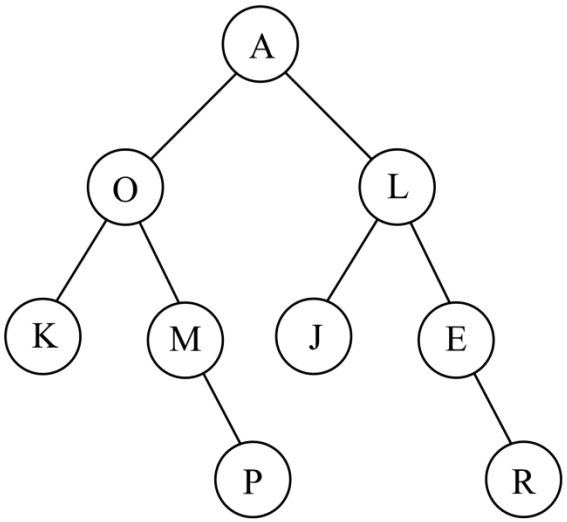
Prvo se obiđe lijevo podstablo korijena (5). Pošto lijevi sin korijena, tj. čvor (2), ima svoje lijevo podstablo, idemo dalje, tj. obilazimo njegovo lijevo podstablo. U njegovom lijevom podstablu se nalazi list (1), i pošto (1) nema svoje lijevo podstablo, obilazimo prvo njega. Dakle, prvi čvor koji smo obišli je (1). Sa njim završavamo obilazak lijevog podstabla čvora (2), pa obiđemo čvor (2), i prelazimo na njegovo desno podstablo. Pošto čvor (3) nema lijevo podstablo, obilazimo čvor (3), pa prelazimo na njegovo desno podstablo. Čvor (4) je list – obiđemo i njega. Sami nastavite dalje.

Preorder i postorder obilasci

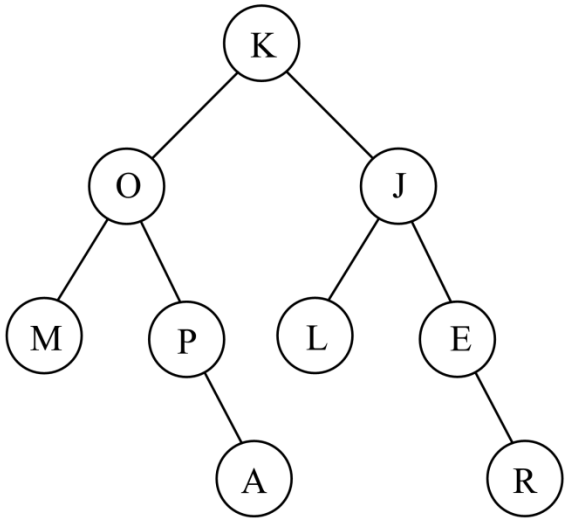
- Preorder obilazak:
  - prvo se obiđe korijen podstabla,
  - zatim se obiđe lijevo podstablo i
  - na kraju se obiđe desno podstablo.
- Postorder obilazak:
  - prvo se obiđe lijevo podstablo,
  - zatim se obiđe desno podstablo i
  - na kraju se obiđe sam korijen podstabla.

Obilasci – Primjer

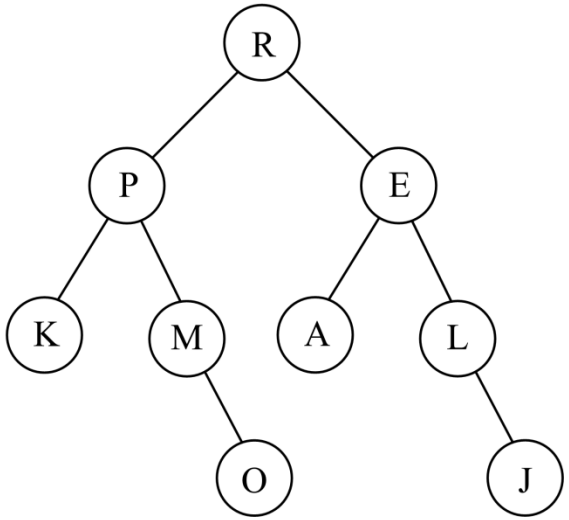
- Slova riječi KOMPAJLER su postavljena u stablu po odgovarajućim obilascima. Provjerite!



a) Inorder obilazak



b) Preorder obilazak



c) Postorder obilazak

Drvo preko samoreferentne strukture

- Drugi način za realizaciju veza između čvorova stabla je preko samoreferentne strukture. Ova struktura sada mora imati dva pokazivača koji pokazuju na lijevog i desnog sina (kod binarnog drveta). Primjer strukture **drvo**:

```
struct drvo {
        int i;                      // i ostali podaci članovi
        struct drvo *left;          // pokazivač na lijevog sina
        struct drvo *right;         // pokazivač na desnog sina
}
```

- Ukoliko drvo nema nekog od sinova odgovarajući pokazivač je NULL. Svi algoritmi koji rade sa drvetom polaze od korijena; stoga se pokazivač na korijen prosljeđuje funkcijama za rad sa drvetom.

Formiranje drveta preko struktura

- U programu koji formira drvo potrebno je imati nekoliko pokazivača na strukture tipa **drvo**. Na primjer:
  struct drvo *p, *r, *q, *root;
- Alocira se memorija za korijen (preskačemo provjeru):
  p = (struct drvo *) malloc(sizeof(struct drvo));
- Upišu se podaci u ovaj element:
  p->i = 4; p->left = NULL; p->right = NULL;
- Ovaj čvor možemo proglasiti korijenom:
  root = p;
- Svi algoritmi koji rade sa drvetom polaze od korijena. Stoga korijen treba trajno memorisati.

Formiranje drveta preko struktura

- Alocirajmo memoriju za dva sina korijena, ako ih korijen ima:
  q = (struct drvo *) malloc(sizeof(struct drvo));
  t = (struct drvo *) malloc(sizeof(struct drvo));
- Ponovo smo izostavili provjeru alokacije (vi nemojte!). Korijen (root, odnosno p) sada pokazuje na sinove:
  p->left = q; p->right = t;
- U sinove su upišu podaci i oni sada pokažu na NULL:
  q->i = 7; q->left = NULL; q->right = NULL;
  t->i = 1; t->left = NULL; t->right = NULL;
- Nastaviti sa formiranjem drveta, počevši od korijena poddrveta q i t.

Određivanje visine drveta

- Uradimo nekoliko primjera koji ilustruju rad sa drvetom.
- Prvi problem je određivanje visine drveta. Ovdje imamo četiri slučaja:
  - Korijen nema sinova. Visina je 0.
  - Korijen ima samo lijevog sina. Visina je jednaka 1 plus visina lijevog podstabla.
  - Korijen ima samo desnog sina. Visina je jednaka 1 plus visina desnog podstabla.
  - Korijen ima oba sina. Visina je jednaka 1 plus visina višeg od dva podstabla.
- Mnoštvo obrada drveta ima sličnu strukturu sa modifikacijama koje se primjenjuju u prethodna četiri slučaja.

Visina drveta zadatog preko nizova

```
#include <stdio.h>
int visina(int i)
int max(int, int);
int n, le[100], ri[100];

int main() {
   int j;
   printf("Unijeti broj cvorova");
   scanf("%d", &n);
   printf("Unijeti LEFTSON i RIGHTSON nizove ");
   for(j=1; j<=n; j++)
       scanf("%d%d", le+j, ri+j);
   printf("%d", visina(1));
}
```

```
int visina(int i) {
   int l, r;
   l = le[i];
   r = ri[i];
   if(l + r == 0)
       return 0;
   else if(l == 0)
       return 1 + visina(r);
   else if(r == 0)
       return 1 + visina(l);
   else
       return 1 + max(visina(l), visina(r));
}

int max (int p, int q) {
    if(p>q)  return p;
    else  return q;
}
```

Visina drveta – Komentar

- Iako ćemo vam prepustiti da sami protumačite veći dio ovog programa, dajemo vam nekoliko komentara.
- Funkcija int visina(int i) daje visinu podstabla čiji je korijen čvor i.
- Kako glavni program poziva funkciju za čvor **1** (korijen) to funkcija vraća rezultat koji je jednak visini kompletnog drveta.
- Pored prethodno opisanog pravila vezanog za određivanje visine drveta, naš program počiva i na činjenici da smo niz lijevih sinova i niz desnih sinova memorisali kao globalne promjenljive.
- Tumačite ostatak kôda sami, a pokušajte i da prepravite realizaciju da radi u slučaju kada ne želimo da koristimo globalne promjenljive.

Visina drveta – Preko struktura

- Pretpostavljamo da je drvo već formirano i to preko struktura. Napišimo funkciju koja određuje visinu drveta. Funkciji se prosljeđuje pokazivač na korijen:

```
int visina(struct drvo *cvor) {
    if(cvor->left == NULL && cvor->right == NULL)
        return 0;
    else if(cvor->left == NULL)
        return 1 + visina(cvor->right);
    else if(cvor->right == NULL)
        return 1 + visina(cvor->left);
    else
        return 1 + max(visina(cvor->left), visina(cvor->right));
}
```

Inorder štampanje čvorova drveta

- Funkcije koje rade sa drvetom preko strukture intenzivno koriste rekurziju.
- Primjer efikasne rekurzije može biti funkcija koja po inorder obilasku obilazi i štampa sadržaj svih čvorova drveta:

```
void print_drvo(struct drvo *cvor) {
        if(cvor->left!=NULL)    print_drvo(cvor->left);
        printf("%d\n", cvor->i);
        if(cvor->right!=NULL)  print_drvo(cvor->right);
}
```

Drvo – Za vježbu

- Za vježbu odraditi sljedeće probleme kod drveta:
  - Napisati funkciju koja formira potpuno binarno drvo.
  - Napisati funkciju koja dealocira drvo polazeći od korijena.
  - Napisati funkciju koja mjeri težinu drveta (broj čvorova drveta).
  - Napisati funkciju koja određuje da li je drvo potpuno binarno.
  - Napisati funkciju koja određuje koliko drvo ima listova.

Drvo – Za vježbu

- Napisati funkciju koja određuje najtežu putanju od korijena do lista. Najteža putanja je ona koja daje najveću vrijednost sume brojeva upisanih u čvorovima na svom putu.
- Napisati funkciju koja određuje najtežu prosječnu putanju od korijena ka listovima. To je ona putanja koja daje najveću vrijednost sume brojeva upisanih u čvorovima na tom putu podijeljenu sa brojem čvorova na tom putu.
- Drvo je sortirano inorder. Napisati funkciju koja vraća pokazivač na čvor u kome je upisan traženi broj ili NULL ako traženi broj ne postoji u drvetu.
- Napisati funkciju koja određuje širinu drveta. To je najveći broj čvorova koji se nalazi na određenoj distanci od korijena.