

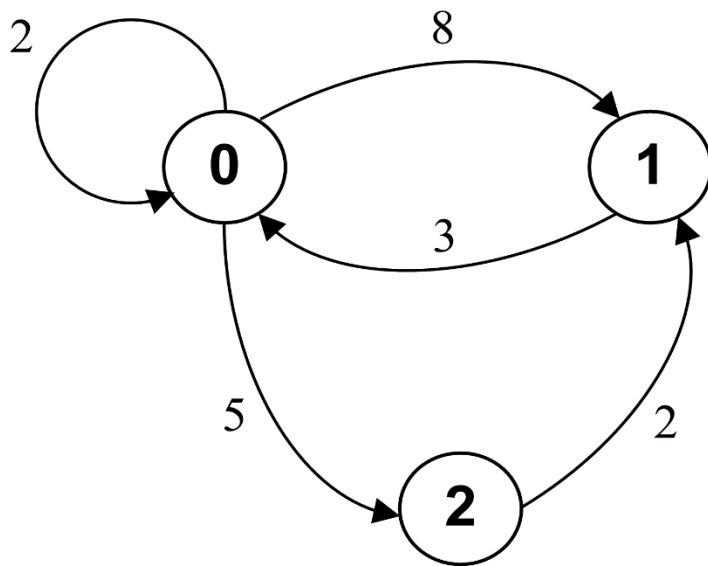


Programiranje I

Grafovi – Problem najkraćeg puta
Drveta

Ilustracija Dijkstra algoritma

- Na primjeru jednostavnog grafa ilustrirajmo korake u Dijkstra algoritmu:



Graf

I[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);
void 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 funkcija

Modifikacija Dijkstra algoritma

- Pretpostavimo da je zadana 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)

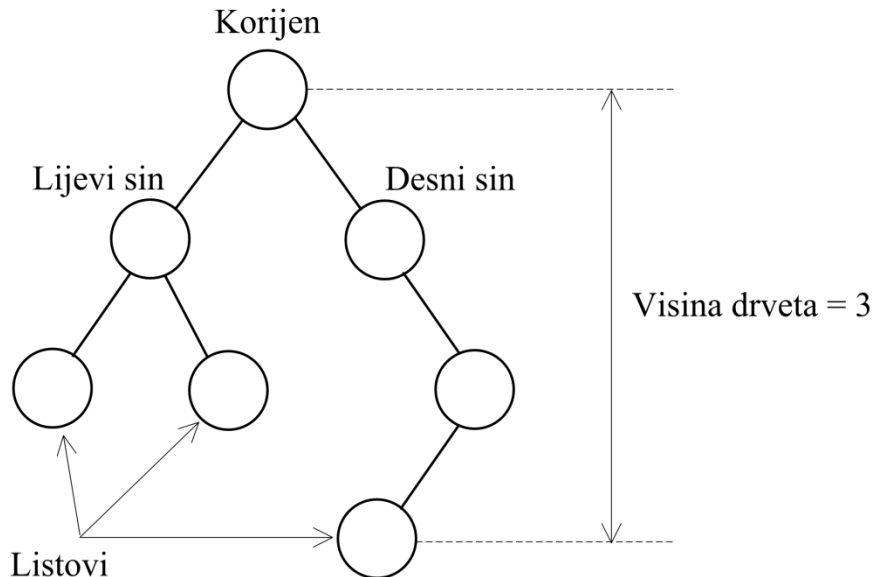
- Još jedan, moramo priznati, čudan naziv u programiranju.
- **Stablo** ili **drvo** je jedan od veoma naprednih i veoma korišćenih tipova podataka.
- **Stablo** je **usmjereni aciklični graf** (graf u kome ne postoji ciklus) koji zadovoljava sljedeća svojstva:
 - Jedan čvor, koji se naziva **korijen**, 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

- Neka je stablo prikazano kao uređeni par čvorova i ivica: $T=(V,E)$. Ako $(v,w) \in E$, gdje su $v,w \in V$, 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 koji nemaju potomaka nazivaju se listovima. Čvor **v** i njegovi potomci čine podstablo čiji je korijen **v**.
- Visina stabla je dužina najdužeg puta od korijena ka listovima.

Binarno stablo – Definicija i primjer

- 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 eventualno postojanje lijevog i desnog podstabla).

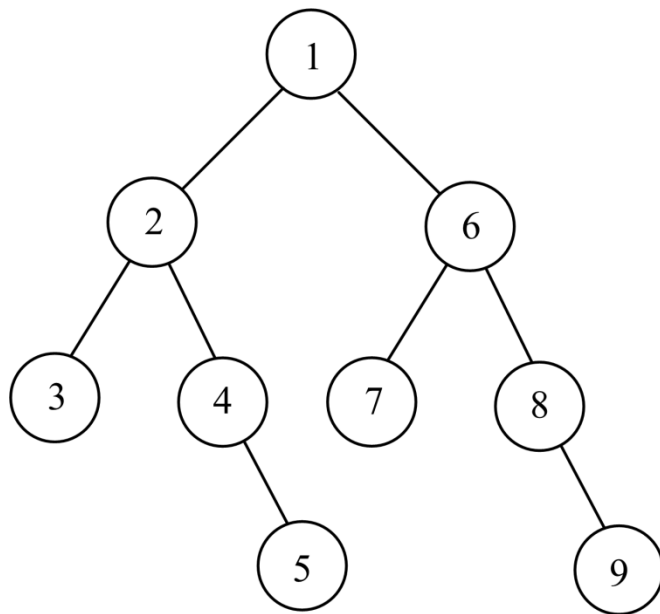


Primjer vizuelizacije jednog binarnog stabla sa ilustracijom nekih od važnih pojmova (korijen, listovi, lijevi i desni sin i visina drveta).

Predstava stabla preko podnizova

- Čvorovi stabla se mogu predstaviti preko struktura. Kako se mogu predstaviti veze između čvorova stabla?
- U upotrebi je i način predstavljanja stabla preko nizova **LEFTSON** i **RIGHTSON**. Elementa ovih nizova ima onoliko koliko ima čvorova u stablu. **LEFTSON[I]** daje koji čvor je lijevi sin čvora **I**, dok **RIGHTSON[I]** daje koji je čvor 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

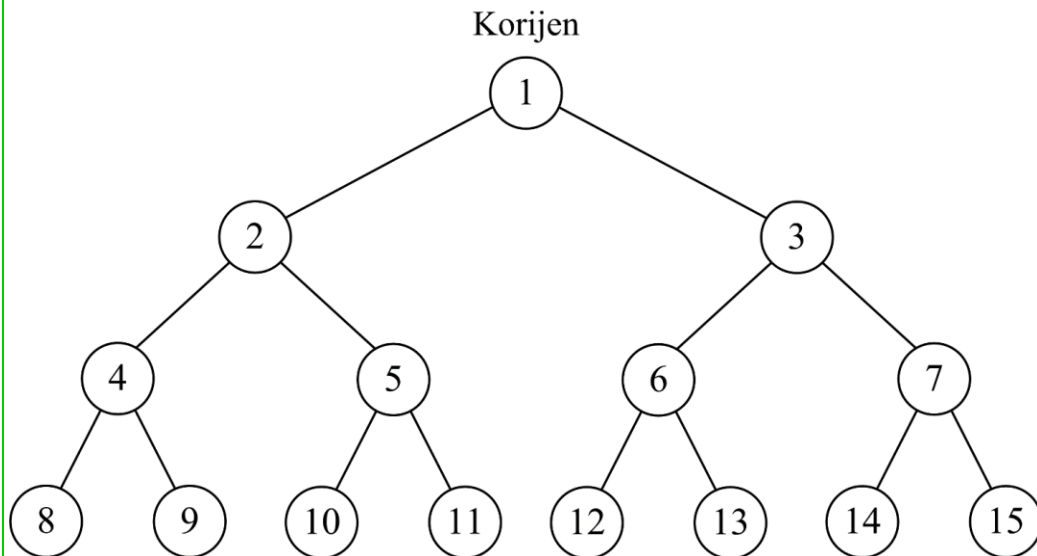


	LEFTSON	RIGHTSON
1	2	6
2	3	4
3	0	0
4	0	5
5	0	0
6	7	8
7	0	0
8	0	9
9	0	0

- 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 pošto nema sinova itd.

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, a koliko visine n ?



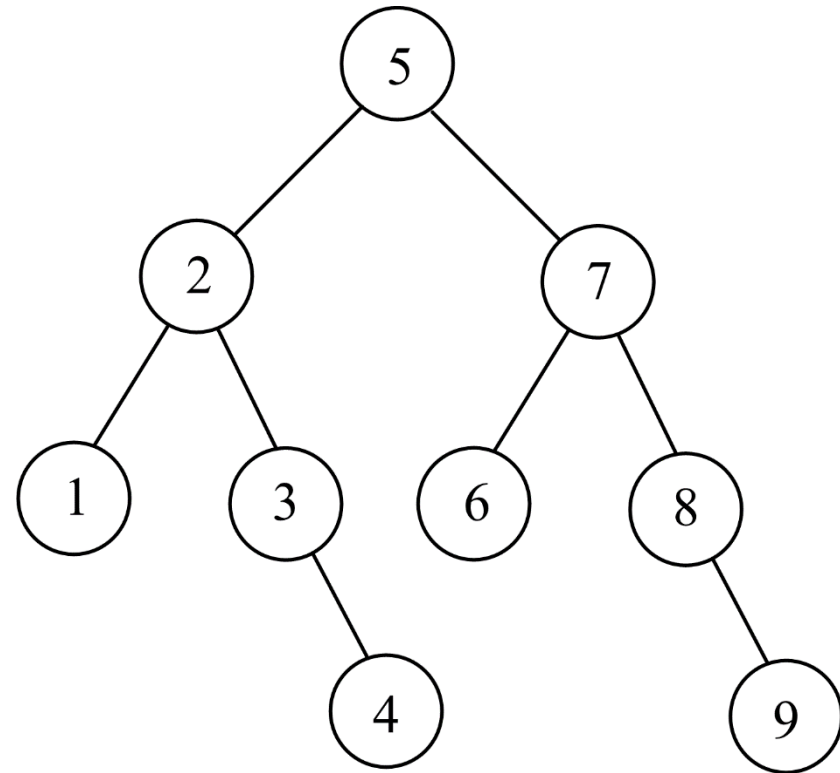
Kod potpunog binarnog stabla usvaja se konvencija da čvor **I** ima lijevog sina **2I** i desnog sina **2I+1**.
Čvoru **J** otac je čvor **[J/2]**, gdje je **[]** operator zaokruživanja nadole.

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 se obavlja ovako:
 - prvo se obiđe lijevo podstablo po srednjem redosljedu,
 - zatim se posjeti korijen i
 - na kraju se posjeti i desno podstablo po srednjem redosljedu.
- Ovakav način obilaska se još naziva **inorder** obilazak.



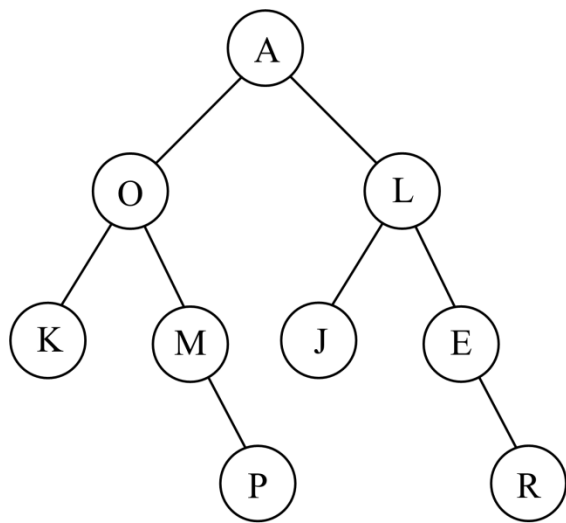
Primjer redosljeda obilaska čvorova po inorder obilasku.

Ostali tipovi obilazaka

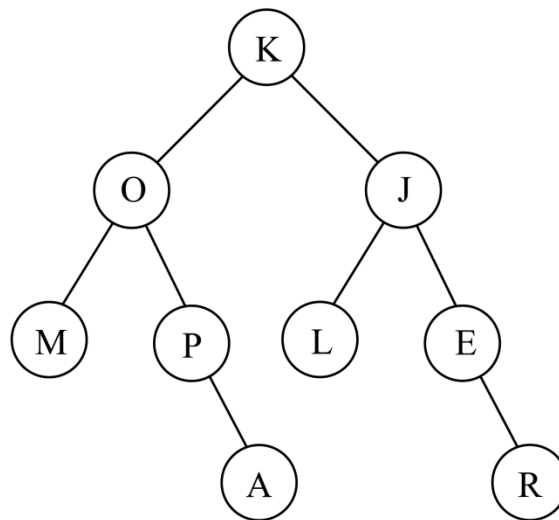
- U prethodnom primjeru treba prvo posjetiti lijevo podstablo po srednjem redosljedu. Lijevi sin korijena ima lijevog sina, a ovaj je list, pa se, dakle, taj list prvi obilazi u cijelom drvetu, zatim njegov roditelj, pa desni dio lijevog podstabla. Zatim se obiđe korijen, pa desno podstablo, koje se ponovo obilazi na isti način, tj. njegovo lijevo podstablo, pa korijen desnog podstabla i na kraju njegovo desno podstablo.
- Pored inorder obilaska postoje i **preorder** (prvo se obiđe korijen, pa lijevo podstablo, pa desno podstablo) i **postorder** (prvo se obiđe lijevo i desno podstablo, pa na kraju korijen).

Obilasci - Primjer

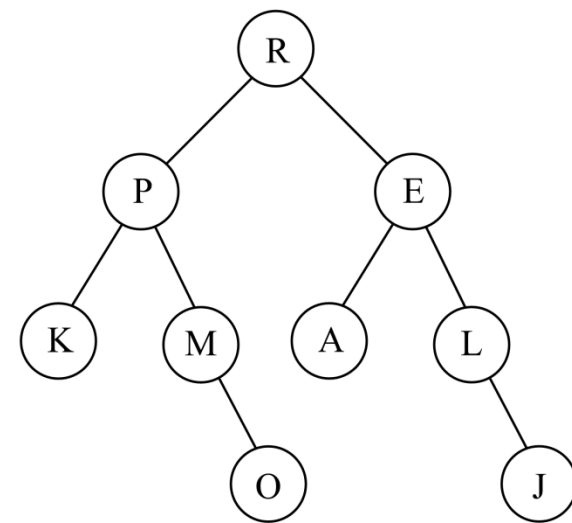
- Slova riječi KOMPAJLER su postavljena u stablu po odgovarajućim obilascima. **Provjerite!**



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 korjena, pa je, stoga, pokazivač na glavu drveta uobičajeni podatak koji se prosljeđuje funkcijama za rad sa drvetom.

Formiranje drveta preko struktura

- U programu koji formira drvo (jedna primitivna aplikacija, data je u zbirci zadataka, primjer 2.13.1) 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 zaboravili provjeru alokacije. 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

- Uradićemo nekoliko primjera koji ilustruju upotrebu drveta. Neki od njih će biti opisani za slučaj drveta koje je prikazano preko dva niza.
- Prvi problem kojega ćemo analizirati je određivanje visine drveta. Ovdje možemo razdvojiti 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 datog 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 istumačite 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 koda 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 *root){
    if(root->left==NULL && root->right==NULL)
        return 0;
    else if(root->left==NULL)    return 1+visina(root->right);
    else if(root->right==NULL)  return 1+visina(root->left);
    else    return 1+max(visina(root->left),visina(root->right));
}
```


Inorder štampanje čvorova drveta

- Da li se prethodni program mogao uraditi jednostavnije ako bismo ispitali da li je aktuelni čvor **NULL** pokazivač?
- 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 *korijen){  
    if(korijen->left!=NULL)    print_drvo(korijen->left);  
    printf("%d\n",korijen->i);  
    if(korijen->right!=NULL)    print_drvo(korijen->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.

Drvo – Za vježbu

- Napisati funkciju koja određuje širinu drveta. To je najveći broj čvorova koji se nalazi na određenoj distanci od korijena.
- Sve funkcije napisati za drvo reprezentovano preko nizova i preko samoreferentnih struktura.

KURS – ZAKLJUČAK

- Studenti nakon ovog kursa treba da su ovladali sljedećim pojmovima:
 - Elementarni tipovi podataka i operacije (80%)
 - Nizovi i pokazivači (60%)
 - Stringovi (80%)
 - Matrice (30%)
 - Funkcije (poziv po vrijednosti i referenci) (70%)
 - Funkcije - napredne opcije (30%)
 - Fajlovi (50%)
 - Strukture (70%)
 - Napredni linkovani tipovi podataka (40%)

procenti predstavljaju procijenjeni značaj za programersko opismenjavanje

KURS – ZAKLJUČAK

- Studenti koji su savladali ovaj kurs se mogu smatrati programerski pismenim, a to dalje znači da mogu da nastave sa usavršavanjem u oblasti programiranja.

■ **Sa srećom!**