

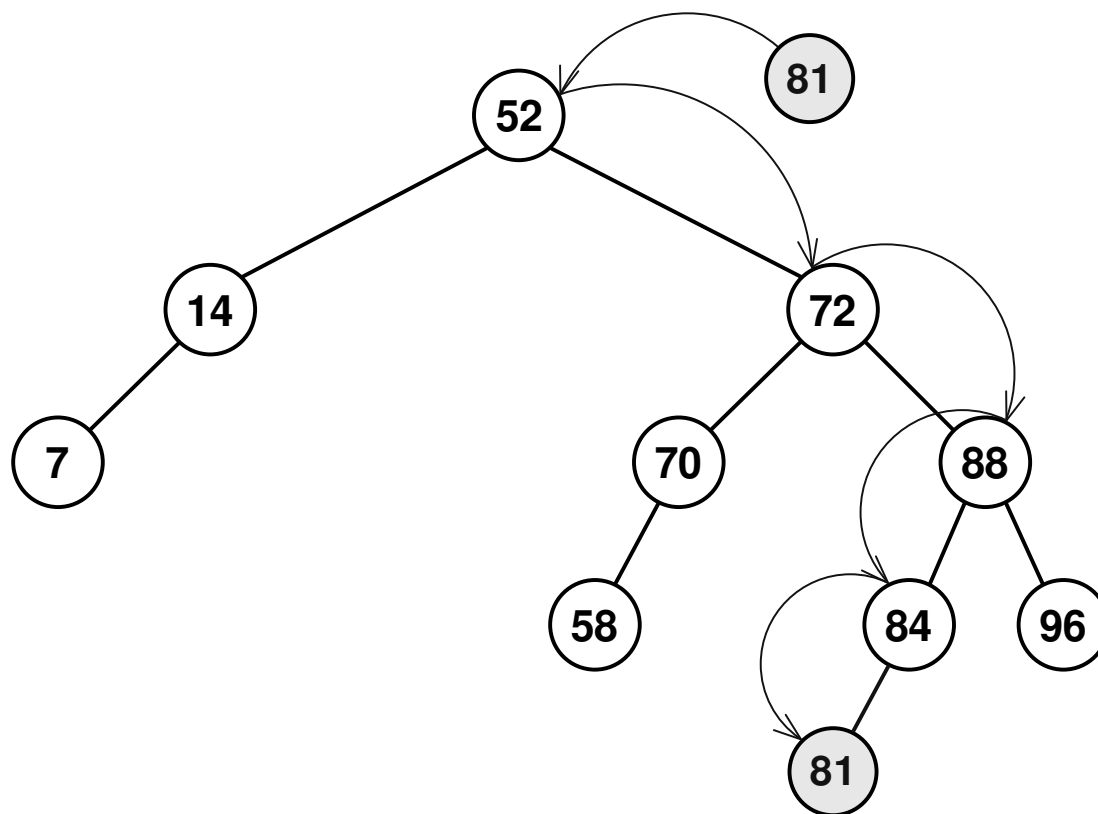
# Zadatak za zadaću

- Strukturu podataka red realizirati uz upotrebu pokazivača. Svaki element reda sadrži naziv (20 znakova) i vrijednost (realni broj). Treba napisati funkciju koja dodaje i funkciju koja briše element iz reda, u kojima treba ispisati element na kojem se obavlja operacija. Također napisati funkciju koja ispiše sve elemente u redu.  
Rok za predaju zadaće: 10. 01. 2007.

# Binarno stablo traženja

- Binarno stablo  $T$  je **binarno stablo traženja** ako su ispunjeni sljedeći uvjeti:
  - čvorovi od  $T$  su označeni podacima nekog tipa na kojem je definiran totalni uređaj  $\leq$ .
  - neka je  $i$  bilo koji čvor od  $T$ . Tada su oznake svih čvorova u lijevom podstablu od  $i$  manje od oznake od  $i$ . Također, oznake svih čvorova u desnom podstablu od  $i$  su veće ili jednake od oznake od  $i$ .

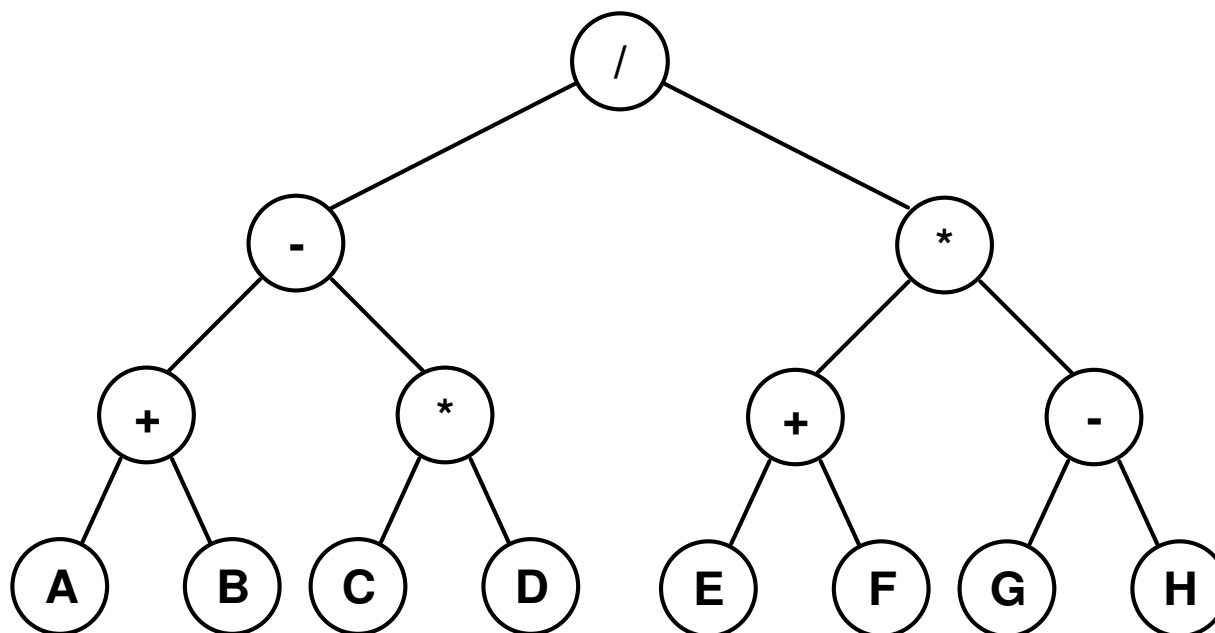
- Zadatak: Nacrtati izgled binarnog sortiranog stabla (binarnog stabla traženja) nakon dodavanja sljedećih podataka: 52, 72, 14, 88, 70, 84, 7, 58 i 96. Nacrtati postupak dodavanja podatka 81 u tako već formirano binarno stablo.



- Zadatak: Ukoliko se stablo prikazuje jednodimenzionalnim poljem, s koliko se članova mora dimenzionirati u najgorem i najboljem slučaju ako je potrebno pohraniti 52 različita elementa?
- Općenito u stablo dubine  $k$  stane  $2^k-1$  članova.
- Najbolji slučaj – stablo će biti **potpuno**, pa je za pohranu potrebna dubina koja je jednaka najmanjoj potenciji broja 2 koja je veća od 52, a to je  $64=2^6$ . Dakle, u stablo dubine 6 stane 63 elementa te je to dovoljno za pohranu 52 elementa. U najboljem slučaju potrebno je polje dimenzionirati na 63.
- Najgori slučaj – stablo će biti potpuno **koso** i dubina stabla će biti 52. Za pohranu je potrebno polje dimenzionirati s  $2^{52}-1= 4,503,599,627,370,495 (\approx 4.5*10^{15})$

- Zadatak: Pretražuje se stablo u kojem se nalazi 27 različitih elemenata. Potrebno je odrediti koliko se operacija uspoređivanja obavi u najboljem i najgorem slučaju za potpuno i koso stablo.
  
- Potpuno stablo:
  - najbolji slučaj – pri prvoj usporedbi nađe se element  $\Rightarrow$  obavljena je jedna usporedba
  - najgori slučaj – podatak se nalazi u "najdubljem" *listu*. Najmanja potencija broja 2 koja je veća od 27 je  $32=2^5$ . Dakle stablo ima dubinu 5 te je potrebno obaviti najviše pet operacija uspoređivanja.
  
- Koso stablo:
  - najbolji slučaj – pri prvoj usporedbi nađe se element  $\Rightarrow$  obavljena je jedna usporedba
  - najgori slučaj – podatak se nalazi u "najdubljem" *listu*. Budući da je u najgorem slučaju stablo u potpunosti koso, razina najdubljeg čvora je 27. Dakle, u najgorem slučaju potrebno je obaviti 27 operacija uspoređivanja.

- Zadatak: Za već formirano binarno stablo potrebno je napisati što se dobije *inorder* i *postorder* obilaskom. Stablo izgleda na sljedeći način:



- Inorder obilazak stabla daje matematički izraz:
- $A + B - C * D / E + F * G - H$
- Postorder obilazak također daje matematički izraz ali u tzv. RPN notaciji:
- $A B + C D * - E F + G H - * /$

# Implementacija rječnika pomoću binarnog stabla traženja

- Pomoću binarnog stabla traženja možemo na efikasan način implementirati operacije umetanja, traženja i brisanja elemenata u skupu podataka - obradit ćemo primjer rječnika s predavanja
- Operacije koje ćemo implementirati su sljedeće:
  - $\text{Insert}(x, T)$  – operacija koja ubacuje novi element  $x$  u binarno stablo traženja  $T$ .
  - $\text{Member}(x, T)$  – operacija koja vraća vrijednost 1 ako se element  $x$  nalazi u stablu traženja  $T$ , a u suprotnom vraća 0
  - $\text{Delete}(x, T)$  – operacija koja uklanja element  $x$  iz binarnog stabla traženja  $T$ .
  - $\text{Inorder}(T)$  – operacija koja obilazi binarno stablo traženja  $T$  čvor po čvor algoritmom "inorder" i ispisuje sve njegove elemente.
- Operacije  $\text{Insert}$  i  $\text{Member}$  rade slično: traže mjesto u stablu na kojem bi morao biti (novi) element, te ubacuje čvor na to mjesto ili javlja da čvor (ne)postoji

- Nešto je složenija operacija Delete( $x$ ,  $T$ ). Imamo tri slučaja:
  - $x$  je u listu; tada jednostavno izbacimo list iz stabla.
  - $x$  je u čvoru koji ima samo jedno dijete. Tada nadomjestimo čvor od  $x$  njegovim djetetom.
  - $x$  je u čvoru koji ima oba djeteta. Tada nađemo najmanji element  $y$  u desnom podstablu čvora  $x$ . Izbacimo čvor od  $y$  (jedan od dva prethodna slučaja). U čvor  $x$  spremimo  $y$  umjesto  $x$ .
- Sve se ovo spretno može zapisati ako uvedemo pomoćnu funkciju DeleteMin( $T$ ). Ta funkcija iz nepraznog stabla  $T$  izbacuje čvor s najmanjim elementom, te vraća taj najmanji element.
- Svaka od funkcija Member, Insert, Delete prolazi jednim putem, od korijena binarnog stabla do nekog čvora. Zato je vrijeme izvršavanja svih operacija ograničeno visinom stabla. Pretpostavimo da u stablu imamo  $n$  elemenata. Visina stabla tada varira između  $(\log_2(n+1))-1$  i  $n-1$ . Ekstremni slučajevi su potpuno stablo i "ispruženo" stablo – lanac.



```

#include <stdio.h>
#include <stdlib.h>
typedef int elementtype;
struct node{
    elementtype el;
    struct node * left;
    struct node * right; };
typedef struct node * BSTree;
int Member(elementtype x, BSTree t){
    if (t == NULL) return 0;
    if (x == t->el) return 1;
    if (x < t->el) return Member(x, t->left);
    else return Member(x, t->right);
}
int Insert(elementtype x, BSTree * t){
    BSTree tree = *t;
    if (tree == NULL){
        tree = malloc(sizeof(struct node));
        tree->el = x;
        tree->left = tree->right = NULL;
        *t = tree;
        return 1; }
    if (x == tree->el) return 0; // x je već u rječniku
    if (x < tree->el) return Insert(x, &tree->left);
    else return Insert(x, &tree->right);
}

```

```

elementtype DeleteMin(BSTree * t)
{
    BSTree temp;
    BSTree tree = *t;
    elementtype minval;

    if (tree->left == NULL)
    {
        // tree pokazuje na cvor s najmanjim elementom. Nadomjestimo taj cvor njegovim
        // desnim djetetom
        minval = tree->el;
        temp = tree;
        tree = tree->right;
        free(temp);
    }
    else
        //cvor na kojeg pokazuje tree ima lijevo dijete
        minval = DeleteMin(&tree->left);

    *t = tree;
    return minval;
}

```

```

void Delete(elementtype x, BSTree * t) {
    BSTree temp;
    BSTree tree = *t;
    if (tree == NULL) return;
    if (x < tree->el)
        Delete(x, &tree->left);
    else if (x > tree->el)
        Delete(x, &tree->right);
    //ako dođemo ovamo, tada je x u čvoru na kojeg pokazuje t
    else if (tree->left == NULL && tree->right == NULL) {
        free(tree);
        tree = NULL;
    } else {
        if (tree->left == NULL)
        { //nadomjestimo čvor od x njegovim desnim djetetom
            temp = tree;
            tree = tree->right;
            free(temp);
        } else if (tree->right == NULL) { // lijevo dijete u čvor od x
            temp = tree;
            tree = tree->left;
            free(temp);
        } else //postoje oba djeteta
            tree->el = DeleteMin(&tree->right);
    }
    *t = tree;
}

```

```

void Inorder(BSTree t) {
    if (t == NULL) return;
    if (t->left != NULL)
        Inorder(t->left);
    printf("%i ", t->el);
    if (t->right != NULL)
        Inorder(t->right);
}

```

```

int main() {
    BSTree myTree = NULL; // kreiramo prazno stablo i ubacimo nekoliko elemenata
    int broj, ind=0, fg;
    elementtype clan;

    printf ("Koliko clanova zelite upisati u listu\n");
    scanf ("%d", &broj);
    printf ("\n");
    srand(time(NULL));
    do {
        clan = (elementtype) 100 * ((float)rand() / (RAND_MAX + 1));
        fg=Insert(clan, &myTree);
        ind+=fg,
    } while(ind<broj)

    //prosetamo po stablu algoritmom inorder i ispisemo elemente
    Inorder(myTree);
    printf ("\n");
}

```

```

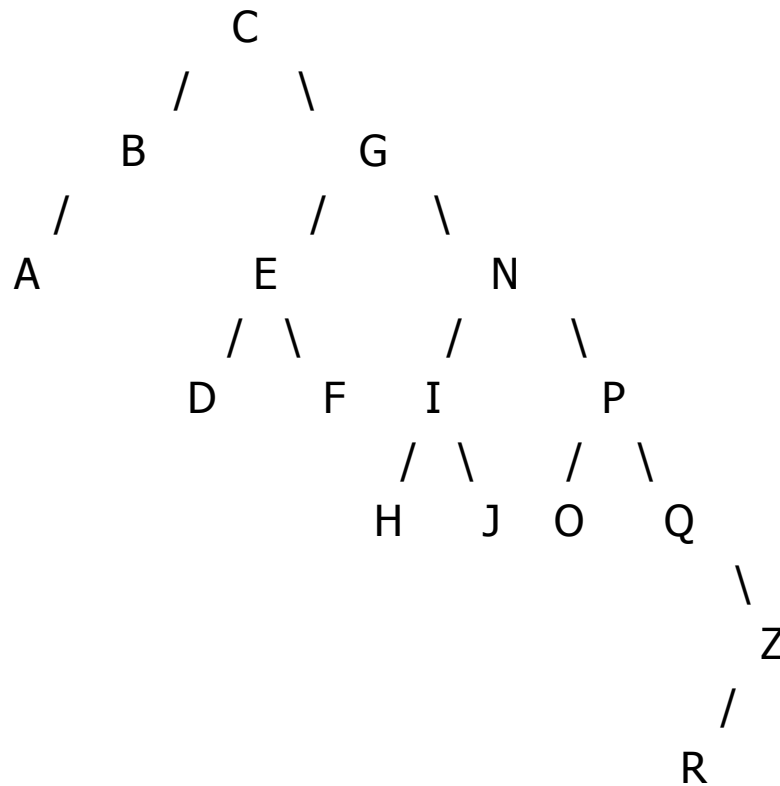
//izbrisemo element
printf("Upisite element koji zelite izbrisati, -1 za kraj\n");
do {
    scanf("%d",&clan);
    Delete(clan,&myTree);
    Inorder(myTree);
    printf("\n");
}
    while (clan != -1);
// provjera da li je element u rjecniku

printf("Upisite element koji trazite, -1 za kraj\n");
do {
    scanf("%d",&clan);
    if (Member(clan, myTree))
        printf("Broj %d se nalazi u stablu",clan);
    else
        printf("Broj %d se ne nalazi u stablu",clan);
    printf("\n");
} while (clan != -1);
system("PAUSE");
return 0;
}

```

## Drugačija izvedba binarnog stabla traženja

- Primjer: napraviti binarno stablo traženja u koje se unosi do 14 znakova (char) po uređaju abecede. Ispisati stablo po inorder, preorder i postorder algoritmu. Također ispisati strukturu stabla. Napisati funkciju koja pretražuje stablo i javlja da li se traženi element nalazi u stablu.
- Neka su ulazni podaci {C, B, G, A, E, F, N, D, P, C, Q, Z, I, R, H, J, O}. Stablo će onda biti:



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

```
struct cv {
    char element[15];
    struct cv *lijevo;
    struct cv *desno;
};
typedef struct cv cvor;
```

```
// upisuje u stablo podatke: lijevo manji, desno veci
```

```
cvor *upis (cvor *glava, char element[]) {
    int smjer; // odluka o podstablu
```

```
    if (glava == NULL) { // prazno (pod)stablo
```

```
        glava = (cvor *) malloc (sizeof (cvor));
```

```
        if (glava) {
```

```
            strcpy (glava->element, element);
```

```
            glava->lijevo = glava->desno = NULL;
```

```
        } else {
```

```
            printf ("U memoriji nema mjesta za upisati '%s'\n", element);
```

```
        }
```

```
    }
```

```

else if ((smjer = strcmp (element, glava->element)) < 0) {
    glava->lijevo = upis (glava->lijevo, element);
} else if (smjer > 0) {
    glava->desno = upis (glava->desno, element);
} else {
    printf ("Podatak '%s' vec postoji!\n", element);
}
return glava; // pokazivac na zadnji element
}

```

// obilazak inorder lijevo-desno:obilazi najlijevije podstablo, pa korijen, pa onda ostala

```

void ispisinld (cvor *glava) {
    if (glava != NULL) {
        ispisinld (glava->lijevo);
        printf ("%s \n", glava->element);
        ispisinld (glava->desno);
    }
}

```

// obilazak inorder desno-lijevo

```

void ispisindl (cvor *glava) {
    if (glava != NULL) {
        ispisindl (glava->desno);
        printf ("%s \n", glava->element);
        ispisindl (glava->lijevo);
    }
}

```



```
// obilazak preorder: prvo korijen, pa obilazi podstabla od najlijevijeg
```

```
void ispispre (cvor *glava) {  
    if (glava != NULL) {  
        printf ("%s \n", glava->element);  
        ispispre (glava->lijevo);  
        ispispre (glava->desno);  
    }  
}
```

```
// obilazak postorder: obilazi podstabla od najlijevijeg, korijen zadnji
```

```
void ispispost (cvor *glava) {  
    if (glava != NULL) {  
        ispispost (glava->lijevo);  
        ispispost (glava->desno);  
        printf ("%s \n", glava->element);  
    }  
}
```

```
// ispis stabla
```

```
void ispissta (cvor *glava, int nivo) {  
    int i;  
    if (glava != NULL) {  
        ispissta (glava->desno, nivo+1);  
        for (i = 0; i < nivo; i++) printf(" ");  
        printf ("%s \n", glava->element);  
        ispissta (glava->lijevo, nivo+1);  
    }  
}
```

```

// trazenje cvora u binarnom stablu
cvor *trazi (cvor *glava, char element[]) {
    int smjer;
    if (glava) {
        if ((smjer = strcmp (element, glava->element)) < 0) {
            return trazi (glava->lijevo, element);
        } else if (smjer > 0) {
            return trazi (glava->desno, element);
        }
    }
    return glava; // ili je pronadjen ili NULL; }

void main(void) {
    FILE *fi; // ulazna datoteka
    int j; // brojac podataka
    cvor *glava, *p; // pokazivac na korijen, pomocni pokazivac
    char ime[15]; //

    fi = fopen ("UlazZaSortiranoStablo.txt", "r");
    if (fi) { // inicijalizacija i citanje podataka
        j = 1;
        glava = NULL;
        while (fscanf (fi, "%s", &ime) != EOF) {
            printf ("%d. ulazni podatak je %s \n", j++, ime);
            glava = upis (glava, ime);
        } fclose (fi);
    }
}

```

```

getchar (); // obilazak i ispis stabla
printf ("Ispis inorder lijevo-desno\n");
ispisind (glava); getchar ();
printf ("Ispis inorder desno-lijevo\n");
ispisindl (glava); getchar ();
printf ("Ispis preorder\n");
ispispre (glava); getchar ();
printf ("Ispis postorder\n");
ispispost (glava); getchar ();
printf ("Ispis stabla\n");
ispissta (glava, 0);
    while (1) { // trazenje elementa
        printf ("Unesite element koji trazite, ili KRAJ >");
        scanf ("%s", ime);
        if (strcmp (ime, "KRAJ") == 0) break;
        p = trazi (glava, ime);
        if (p) {
            printf ("Pronadjen je element: %s\n", p->element);
        } else {
            printf ("Nije pronadjen element: %s\n", ime);
        } }
else {
    printf ("Nema ulaznih podataka\n");
    exit (1); }
system("PAUSE");
exit (0); }

```