Računske vježbe 10

Programiranje I

1. Svakom čvoru drveta pridružen je jedan cijeli broj. Potrebno je napisati sljedeće funkcije:

   a) **weight** koja određuje broj čvorova drveta (težinu drveta),

   b) **leaf_count** koja određuje broj listova drveta,

   c) **max_path** koja vrši obilazak drveta od korijena ka listovima, sabira cijele brojeve pridružene čvorovima i pronalazi putanju sa najvećom sumom brojeva.

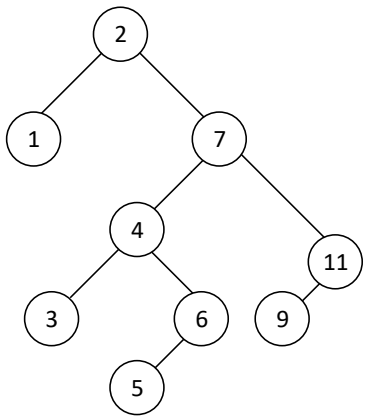
Glavni program kreira stablo i poziva funkcije napisane pod a, b i c.

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

struct node
{
    int value;
    struct node* left;
    struct node* right;
};

struct node* insert(struct node*, int);
void print_inorder(struct node*);

int weight(struct node*);
int leaf_count(struct node*);
int max_path(struct node*);
int max(int, int);

int main()
{
    struct node* root = NULL; // Obavezno zbog prvog uslova insert funkcije!!
    root = insert(root, 2);
    insert(root, 7);
    insert(root, 4);
    insert(root, 11);
    insert(root, 9);
    insert(root, 6);
    insert(root, 5);
    insert(root, 3);
    insert(root, 1);
    printf("Inorder redosljedom stampanja stabla dobijamo: ");
    print_inorder(root);
    printf("\nTezina drveta je: %d", weight(root));
    printf("\nBroj listova je: %d", leaf_count(root));
    printf("\nPutanja sa najvecom sumom brojeva je: %d", max_path(root));
}

```

```c
struct node* insert(struct node* node, int value)
{
    // u slucaju da cvor ne postoji, pravimo ga
    if (node == NULL)
    {
        struct node* temp;
        temp = (struct node*)malloc(sizeof(struct node));
        temp->value = value;
        temp->left = NULL;
        temp->right = NULL;
        return temp;
    }

    // trazimo poziciju novog cvora
    if (value < node->value)
        node->left = insert(node->left, value);
    else if (value > node->value)
        node->right = insert(node->right, value);

    // kada je value == node->value, a cvor vec postoji
    return node;
}

void print_inorder(struct node* root)
{
    if (root != NULL) {
        print_inorder(root->left);
        printf("%d ", root->value);
        print_inorder(root->right);
    }
}

int weight(struct node* root)
{
    if(root->left != NULL && root->right != NULL)
        return 1 + weight(root->left) + weight(root->right);
    else if(root->left != NULL && root->right == NULL)
        return 1 + weight(root->left);
    else if(root->left == NULL && root->right != NULL)
        return 1 + weight(root->right);
    else
    return 1;
}

int leaf_count(struct node* root)
{
    if(root->left != NULL && root->right != NULL)
        return leaf_count(root->left) + leaf_count(root->right);
    else if(root->left != NULL)
        return leaf_count(root->left);
    else if(root->right != NULL)
        return leaf_count(root->right);
    else
    return 1;
}

int max_path(struct node *root)
{
    if(root->left != NULL && root->right != NULL)
```

```
97          return root->value + max(max_path(root->left), max_path(root->right));
98      else if(root->left != NULL)
99          return root->value + max_path(root->left);
100     else if(root->right != NULL)
101         return root->value + max_path(root->right);
102     else
103         return root->value;
104 }
105
106 int max(int num1, int num2)
107 {
108     return (num1 > num2 ) ? num1 : num2;
109 }
```

Kako bismo demonstrirali rad zadatih funkcija, napisali smo nekoliko dodatnih za formiranje stabla i štampanje inorder obilaskom. U prilogu ovih vježbi može se naći i pristup formiranja stabla upotrebom četiri pokazivačke promjenljive. Funkcija *insert* formira sortirano binarno stablo u kojem lijevo podstablo čvora sadrži samo one čvorove koji imaju manju vrijednost od njega dok desno sadrži samo one čvorove koji imaju veću vrijednost od njega. Čvor sa istom vrijednošću se ne može ponoviti. Sortirano binarno stablo će se češće sresti pod nazivom binarno stablo pretrage (engl. *binary search tree - BST*) i predstavlja važan pojam u računarstvu zbog efikasnosti koju pruža pri upotrebi algoritama sortiranja i pretrage. Dakle, ako je proslijeđena vrijednost manja od čvora koji se posmatra, prosljeđujemo je lijevo, u suprotnom prosljeđujemo je desno. Kada dođemo u situaciju da čvor ne postoji tada ga treba napraviti. Interesantno je da se inorder obilaskom ovakvo stablo uvijek štampa u rastućem poretku. Stablo koje smo formirali se može vidjeti na slici 1. Imati na umu da se parametar ovih funkcija naziva **root**, ali to nije nužno korijen početnog stabla. Svaki čvor u stablu koji ima makar jedno podstablo jeste iz perspektive tog podstabla njegov korijen, zar ne? Funkcija **weight** je veoma jednostavna, za svaki posjećeni čvor daje 1 i nastavlja dalje. Lijevo, desno ili u oba smjera, zavisno od toga koliko dati čvor ima sinova. List je čvor koji nema potomaka te je i problem koji funkcija **leaf_count** rješava također jednostavan. Treba vratiti 1 samo za one čvorove koji nemaju potomaka, a ukoliko imaju barem lijevog ili desnog sina nastaviti pretragu u tim pravcima. Funkcija **max_path** djeluje značajno složenije, ali zapravo nije. Šta ona radi? Pa za posmatrani čvor ona posmatra njegovo lijevo i desno podstablo. Ako ima oba, pomoćnom funkcijom **max** biramo ono koje daje duži put. U suprotnom, ide preostalim putem sve dok ne stigne do listova. Drugim riječima, funkcija će se spuštati sve do listova i birati lijevi ili desni list zavisno od njegove vrijednosti. Putanje se preko očeva akumuliraju, a očevi očeva listova sada biraju jednog od sinova koji daje veću putanju i tako sve do korijena cijelog stabla.



Slika 1: Sortirano binarno stablo kreirano u glavnom programu pomoću funkcije *insert*. Lijevo podstablo čvora sadrži samo one čvorove koji imaju manju vrijednost od njega dok desno sadrži samo one čvorove koji imaju veću vrijednost od njega.

2. Svakom čvoru drveta pridružena je jedna riječ, dužine manje od 12 karaktera. Napisati funkciju **word_frequency** koja određuje koliko puta se pojavljuje tražena riječ.

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

struct node
{
    char word[12];
    struct node* left;
    struct node* right;
};

void print_inorder(struct node*);
struct node* insert(struct node*, char*);

int word_frequency(struct node*, char*);

int main()
{
    char word[12];
    struct node* root = NULL;
    root = insert(root, "korijen");
    puts("Unosimo rijeci. Unesi 'kraj' za kraj!");
    while(1)
    {
        scanf("%s", word);
        if(strcmp(word, "kraj") == 0) break;
        insert(root, word);
    }
    printf("Rijeci u stablu su: ");
    print_inorder(root);
    printf("\nUnesite rijec od interesa: ");
    scanf("%s", word);
    printf("Odredjujemo koliko puta se pojavljuje trazena rijec...");
    printf("\nTrazena rijec se pojavljuje %d puta!", word_frequency(root, word));
    return 0;
}

int word_frequency(struct node *root, char *word)
{
    int ind = 0;
    if(strcmp(root->word, word) == 0) ind = 1;
    if(root->left != NULL && root->right != NULL)
        return ind + word_frequency(root->right, word) + word_frequency(root->left,
   word);
    else if(root->left != NULL)
        return ind + word_frequency(root->left, word);
    else if(root->right != NULL)
        return ind + word_frequency(root->right, word);
    else
        return ind;
}

void print_inorder(struct node* root)
{
    if (root != NULL) {
        print_inorder(root->left);
        printf("%s ", root->word);
```

```
        print_inorder(root->right);
    }
}

struct node* insert(struct node* node, char* word)
{
    // u slucaju da cvor ne postoji, pravimo ga
    if (node == NULL)
    {
        struct node* temp;
        temp = (struct node*)malloc(sizeof(struct node));
        strcpy(temp->word, word);
        temp->left = NULL;
        temp->right = NULL;
        return temp;
    }

    // trazimo poziciju novog cvora
    if (strcmp(word, node->word) <= 0)
        node->left = insert(node->left, word);
    else
        node->right = insert(node->right, word);
}
```

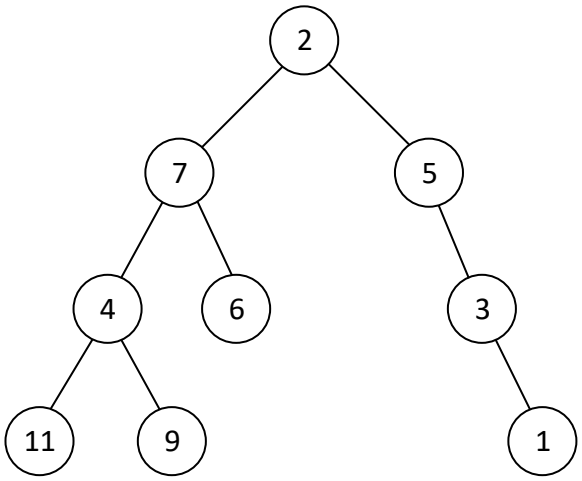
Kako je potrebno izračunati koliko se puta tražena riječ pojavljuje, izmijenili smo funkciju iz prethodnog zadatka da dozvoli upis istih vrijednosti. U suprotnom, zadatak ne bi imao smisla. Obratite pažnju da smo za unos riječi u stablo koristili beskonačnu petlju. Beskonačne petlje, dakle, nisu uvijek negativne, sve dok postoji jasan mehanizam njihovog zaustavljanja. Kao i prethodne funkcije, **word_frequency** također stablo dijeli na lijevo i desno podstablo, a proces ponavlja za svaki od posjećenih čvorova. Prvo se uslov postavlja za posjećeni čvor, a onda za njegova podstabla. To radimo pomoću promjenljive ind. **Pažnja!** Prilikom alokacije memorije nismo vršili provjeru uspješnosti, a nismo ni dealocirali memoriju zauzetu za stablo. Izmijeniti funkciju *insert* i napisati funkciju **free_tree** kojom se dealocira memorija koju zauzima stablo.

3. Dato je binarno drvo u čijim su čvorovima upisani cijeli brojevi. Odrediti redosljed štampanja tih brojeva ukoliko koristimo:

   a) inorder obilazak (lijevo poddrvo, korijen, desno poddrvo),

   b) preorder obilazak (korijen, lijevo poddrvo, desno poddrvo),

   c) postorder obilazak (lijevo poddrvo, desno poddrvo, korijen).



Slika 2: Zadato drvo.

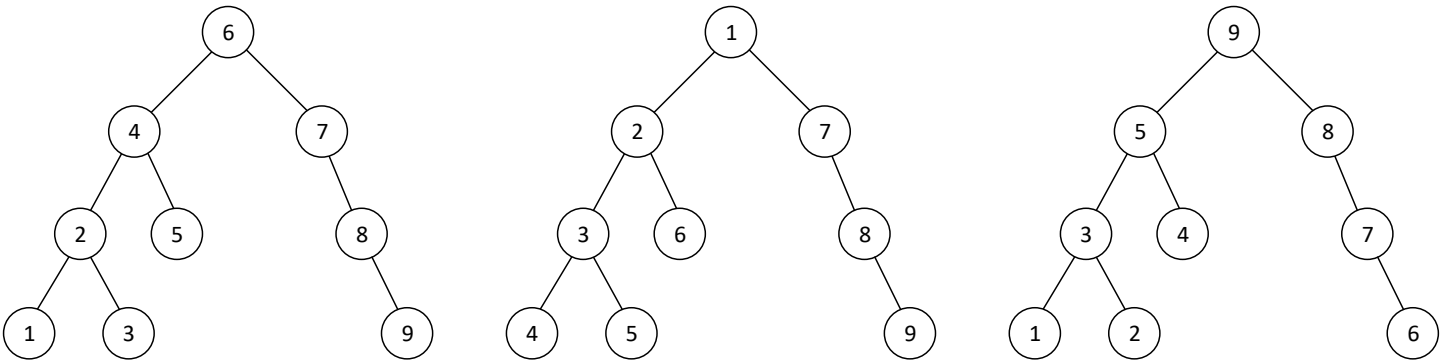
Rješenje zadatka je:

a) 11, 4, 9, 7, 6, 2, 5, 3, 1;

b) 2, 7, 4, 11, 9, 6, 5, 3, 1;

c) 11, 9, 4, 6, 7, 1, 3, 5, 2.

Odnosno, redosljed se može prikazati i grafički što je i urađeno na slici 3.



(a) inorder obilazak (b) preorder obilazak (c) postorder obilazak

Slika 3: Grafički prikaz obilazaka.

Dodatak: Programski kod koji kreira drvo iz prethodnog zadatka.

```
    struct node *p = NULL, *q = NULL, *r = NULL, *t = NULL;
    //2
    p = (struct node *)malloc(sizeof(struct node));
    if(p == NULL) exit(1);
    p->value = 2;
    p->left = NULL;
    p->right = NULL;
    //7
    q = (struct node *)malloc(sizeof(struct node));
    if(q == NULL) exit(1);
    q->value = 7;
    q->left = NULL;
    q->right = NULL;
    p->left = q;
    //5
    r = (struct node *)malloc(sizeof(struct node));
    if(r == NULL) exit(1);
    r->value = 5;
    r->left = NULL;
    r->right = NULL;
    p->right = r;

    t = q;
    //6
    q = (struct node *)malloc(sizeof(struct node));
    if(q == NULL) exit(1);
    q->value = 6;
    q->left = NULL;
    q->right = NULL;
    t->right = q; // prvo upisujemo 6 jer se radi o listu

    //4
    q = (struct node *)malloc(sizeof(struct node));
    if(q == NULL) exit(1);
    q->value = 4;
    q->left = NULL;
    q->right = NULL;
    t->left = q;

    t = q;
    //11
    q = (struct node *)malloc(sizeof(struct node));
    if(q == NULL) exit(1);
    q->value = 11;
    q->left = NULL;
    q->right = NULL;
    t->left = q;
    //9
    q = (struct node *)malloc(sizeof(struct node));
    if(q == NULL) exit(1);
    q->value = 9;
    q->left = NULL;
    q->right = NULL;
    t->right = q;
    //vracamo se na desno podstablo cvora (2) tj. pokazivac r
    //cija je vrijednost 5
    //3
```

```
    q = (struct node *)malloc(sizeof(struct node));
    if(q == NULL) exit(1);
    q->value = 3;
    q->left = NULL;
    q->right = NULL;
    r->right = q;

    r = q;
    //1
    q = (struct node *)malloc(sizeof(struct node));
    if(q == NULL) exit(1);
    q->value = 1;
    q->left = NULL;
    q->right = NULL;
    r->right = q;
```