БІНАРНІ ДЕРЕВА

Алгоритми

Лабораторна робота №8

Варіант 7

Виконав: Конча Вадим

Завдання:

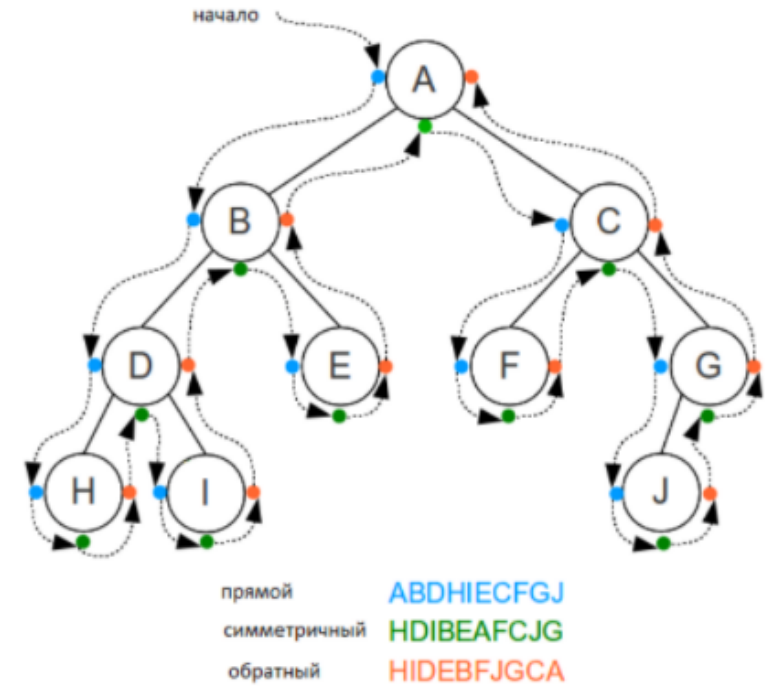
Вимоги до виконання роботи

1. Складіть програму, яка містить процедури та функції для обробки бінарних дерев: побудови бінарного дерева пошуку із масиву даних, обходу дерева, пошуку по дереву, вставки елемента в дерево, видалення елемента з дерева, видалення всього дерева. При цьому передбачте можливість введення вхідного масиву з клавіатури або файлу.

2. Після побудови дерева та після кожної операції над ним виведіть на екран значення усіх вершин дерева у порядку прямого, зворотного та симетричного обходів.

3. Доповніть програму процедурами та функціями, необхідними для виконання вашого індивідуального завдання. Результат виконання виведіть на екран.

7. Написати функцію, яка визначає число входжень вершини із заданим елементом E в дерево T.



Судячи з цього зображення:
прямий - prefix
симетричний - infix
зворотній - postfix

Код:

```
#include <iostream>
#include <vector>
#include "razdel.cpp"
#include <string>
#include <fstream>

using namespace std;
int maxit = 0; //максимальная дальность в ветках
int iqq = 0; //итер для вывода (к-во элементов)
int counter = 0; //считалка элементов (для функции 7)

struct vetka
{
    int value;
    vetka* a = 0;
    vetka* b = 0; //sons
    // a < value < b
    int iterat; //номер ветки
    int prarodidetel;
    bool status1 = true; //для сортировки
    bool status2 = true; //для сортировки
    bool status3 = true; //для сортировки
};
int iden(vetka*& t, int n)
{//возвращает итератор элемента в дереве
    if (!t) return -1;
    if (t->value == n) return t->iterat;
    if (iden(t->a, n) != -1) return iden(t->a, n);
    if (iden(t->b, n) != -1) return iden(t->b, n);
}
void Add(vetka*& t, int n, int prarodid)
{
    //Если ветка не существует
    if (!t)
    { //то её надо создать
        t = new vetka; //создание
        t->value = n; //закидываем туда значение
        t->a = 0; //обозначение а и б как нулей
        t->b = 0; //(их адреса нули, что говорит нам о их несуществовании)
        if (prarodid == 0)
        {
            t->iterat = 0;
        }
        else
        {
        t->prarodidetel = prarodid; //обозночает прародителя
        }
    }
    else //если вдруг такая ветка есть
        if (t->value > n) //новое значение меньше того что в ветке?
        {
```

```
            Add(t->a, n,t->value); //если да, то пошлём его по новой ветке (левой)
        }
        else
        {
            Add(t->b, n,t->value); //правой
        };
}
void indexFORALL(vetka*& t)
{
    if (!t) return;
    else {
        if (!t->a);
        else  t->a->iterat = t->iterat + 1; //обозначает индекс элемента зависимо от прародителя
        if (!t->b);
        else  t->b->iterat = t->iterat + 1; //обозначает индекс элемента зависимо от прародителя
        indexFORALL(t->a); //левая ветка
        indexFORALL(t->b); //правая
    }
}
void printALL(vetka*& t)
{
    if (!t) return;
    else
    {
        iqq++;
        cout << iqq <<") "<< t->value << " and index = " << t->iterat << " and praroditel = " <<
t->prarodidetel << endl;
        printALL(t->a); //левая ветка
        printALL(t->b); //правая
    }
}
int maxiden(vetka*& t)
{
    if (!t) return maxit;
    if (maxit < iden(t, t->value)) maxit = iden(t, t->value);
    maxiden(t->a);
    maxiden(t->b);
}
void makelist(vetka*& t, int n, vector <int> &valuelist)
{
    if (!t) return;
    else {
        if (!t->a);
        else   if (t->a->iterat == n) valuelist.push_back(t->a->value);
        if (!t->b);
        else  if (t->b->iterat == n) valuelist.push_back(t->b->value);
        makelist(t->a, n,valuelist);
        makelist(t->b, n,valuelist);
    }
}
/*
*     if (!t) return;
```

```
    ////////ОСНОВНОЙ КОД//////

    maxiden(t); //максимальный итератор
    for (int i = 0; i<=maxit; i++)
        if (t->iterat == maxit - i)
        {
            cout << string(t->iterat, ' ') << t->value << endl;
        }
    printVetka(t->a); //Выведем ветку и ее подветки
    printVetka(t->b);//И ветки, что справа
*/
void printVetka(vetka*& t, int startpoint)
{
    if (!t) return;
    maxiden(t); //максимальный итератор
    vector <int> valuelist;
    cout << endl << "  "  << startpoint;
    for (int i = 0; i <= maxit; i++)
    {
        makelist(t, i, valuelist);
        for (int k = 0; k < valuelist.size(); k++)
        {
            if (valuelist[k] < startpoint) cout << valuelist[k] << " ";
        }
        cout << string(i*i, ' ');
        for (int k = 0; k < valuelist.size(); k++)
        {
            if (valuelist[k] > startpoint) cout << valuelist[k] << " ";
        }
        valuelist.clear();
        cout << endl;
    }

}
void deleteALL(vetka* t)
{
    if (!t) return;
    deleteALL(t->a);
    deleteALL(t->b);
    delete t;
}
void deleteElement(vetka*& t, int n)
{
    if (!t) return;
    if (t->value == n)
    {
        deleteALL(t);
        t = 0;
        return;
    }
    else
    {
```

```
        deleteElement(t->a, n);
        deleteElement(t->b, n);
    }
}
void TheEnd_counting(vetka*& t, int n)
{
    if (!t) return;
    else {
        if (!t->a);
        else   if (t->a->value == n) counter++;
        if (!t->b);
        else  if (t->b->value == n) counter++;
        TheEnd_counting(t->a, n);
        TheEnd_counting(t->b, n);
    }
}
void firstGO(vetka*& t, vector <int> &sortfg)
{
    if (!t) return;
    else
    {
        if (t->status1 == true)
        {
            sortfg.push_back(t->value);
            t->status1 = false;
        }
        firstGO(t->a, sortfg);
        firstGO(t->b, sortfg);
    }
}
void secondGO(vetka*& t, vector <int>& sorttg)
{
    if (!t) return;
    else
    {
        if (t->status2 == true)
        {
            secondGO(t->a, sorttg);
            sorttg.push_back(t->value);
            t->status2 = false;
            secondGO(t->b, sorttg);
        }
    }
}
void thirdGO(vetka*& t, vector <int>& sorttg)
{
    if (!t) return;
    else
    {
        if (t->status3 == true)
        {
            thirdGO(t->a, sorttg);
```

```
            thirdGO(t->b, sorttg);
            sorttg.push_back(t->value);
            t->status3 = false;
        }
    }
}

int main()
{
    string message;

    ifstream file("data.txt");
    if (!file)
    {
        cout << "File is not open\n\n";
    }
    else
    {
        cout << "File is open!\n\n";
    }
    getline(file, message);

    //getline(cin, message);
    vector <int> f=
      razdel(message); //ввод через пробел
   // { 56, 38, 73, 25, 64, 15, 87, 47, 93, 101, 120, 110, 105 };
   //56 38 73 25 64 15 87 47 93 101 120 110 105


    vetka* t = 0;
    for (int i = 0; i < f.size(); i++) Add(t, f[i], 0);
    indexFORALL(t); //дадим же всем их индексы
    printALL(t); //да напечатаем же мы все с их данными
    printVetka(t,f[0]); //ОКАЗЫВАЕТСЯ ЭТО НЕ НАДО, АХХАХАХХАХАХАХХАА, треш.
    /*
    //удаление элемента
        deleteElement(t, 87); //если крч, не надо, чтобы всю ветку отрезало, а типо смещалось, то
ахах, просто убери из массива командой кой нить этот элемент
        iqq = 0; //итер для вывода
        printALL(t);
    */

    //first - префиксный обход
    cout << endl << endl << "prefix: ";
    vector <int> sortfg;
    firstGO(t, sortfg);
    for (int i = 0; i < sortfg.size(); i++)
    {
        cout << sortfg[i] << " ";
    }

    //second - инфиксный обход (по сути сортировка)
```

```
    cout << endl << endl << "infix: ";
    vector <int> sortsg;
    secondGO(t, sortsg);
    for (int i = 0; i < sortsg.size(); i++)
    {
        cout << sortsg[i] << " ";
    }

    //third постфиксный
    cout << endl << endl << "postfix: ";
    vector <int> sorttg;
    thirdGO(t, sorttg);
    for (int i = 0; i < sorttg.size(); i++)
    {
        cout << sorttg[i] << " ";
    }
    cout << endl;

    TheEnd_counting(t, 25);
    cout << "\n" << counter << " is how often element can be in this tree... Like this)\n\n";
}
```

Допоміжний код:

```
// razdel.cpp вроде
#include <iostream>
#include <string>
#include <stdlib.h>
#include <vector>

using namespace std;

vector <int> razdel(string str, char joke = ' ')
{
//          string str = "12 23 456 "; //воображаемая строка
//          char joke = ' '; //воображаемый разделитель

            //!
                        for (int g = 1; g < str.length(); g++) //проверка на кривой ввод
                        {
                                    if (str[g] == str[g - 1] && str[g] == joke)  //Если несколько разделителей
подряд
                                    {
                                                str.erase(g, 1); //уничтожение повторяющегося разделмтеля
                                                g--; //уменьшение индекса для коректной работы
                                    }
                        }
            //!

            //определение size
```

```
	int size = 1;
	for (int i = 0; i < str.length(); i++) if (str[i] == joke) size++; //подсчёт пробелов +1 =
количество элементов
	string* arr = new string[size]; //создание динамического массивчика)))
	string buff; //буфер для значений

	//кастыли, которые закрывают дыры функции
	if (str[str.size() - 1] != joke) str += joke; //если в конце нету разделителя, то добавить его
	else size -= 1; //если он там есть, то уменьшить список на 1 ячейку

	for (int k = 0; k < size; k++) //перебор всех ячеек массива
	{
		for (int i = 0; i < str.size(); i++) //перебор всех символов
		{
			//этот индекс не последний?
			if (i == str.length()) { //сохранение в массив последнего элемента(иначе он
теряется)
				arr[k] += buff;
				buff = ""; //буфер очищается
				k++;
			}

			if (str[i] != joke) //если место в строке это НЕ пробел
			{
				buff += str[i]; //то в буфере появляется новая цифра, это сделано
для больших чисел
				//cout << "\n" << "newbufer : " << buff << endl;
			}
			else { //если там разделитель (пробел)
				arr[k] += buff; //то в массив добавляется новый элемент, который
равен буферу
				buff = ""; //буфер очищается
				k++;
			}
		}
	}
```

РЕЗУЛЬТАТИ:

```
5 element: 64
6 element: 15
7 element: 87
8 element: 47
9 element: 93
1) 56 and index = 0 and praroditel = -842150451
2) 38 and index = 1 and praroditel = 56
3) 25 and index = 2 and praroditel = 38
4) 15 and index = 3 and praroditel = 25
5) 47 and index = 2 and praroditel = 38
6) 73 and index = 1 and praroditel = 56
7) 64 and index = 2 and praroditel = 73
8) 87 and index = 2 and praroditel = 73
9) 93 and index = 3 and praroditel = 87

  56
38  73
25 47     64 87
15          93


prefix: 56 38 25 15 47 73 64 87 93

infix: 15 25 38 47 56 64 73 87 93

postfix: 15 25 47 38 64 93 87 73 56

1 is how often element can be in this tree... Like this)
```

І навіть зробив графічний вигляд дерева. Так, не найкращий, але цього і не було в завданні. Що я зрозумів доволі пізно, але не суть. :)

Джерела:

[Бинарное дерево поиска - YouTube]

[Обход деревьев - YouTube]

Музика, що не давала заснути:

[Валентин Стрыкало - Гори - YouTube]

та інше…