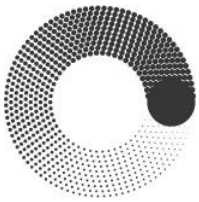
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ



МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет Информационных технологий
Кафедра Информатики и информационных технологий

направление подготовки
09.03.02 «Информационные системы и технологии»

ЛАБОРАТОРНАЯ РАБОТА № 4-5

Дисциплина: «Основы современных алгоритмов»

Тема: «Создание и обход бинарного дерева»

Выполнил: студент группы 211-723

Сергеев Станислав Олегович

Дата, подпись_____ _____
(Дата) (Подпись)

Проверил:_____ _____
(Фамилия И.О., степень, звание) **(Оценка)**

Дата, подпись_____ _____
(Дата) (Подпись)

Замечания: _____

_____

_____

Москва

2022

Создание и обход бинарного дерева

Цель:

Получить знания и практические навыки в создании и манипуляции с данными представляемыми древовидными бинарными структурами.

Постановка задачи:

Разработать программу синтеза и обхода бинарного дерева. Для заданной последовательности данных с помощью созданной программы построить и сохранить в файле бинарное дерево. Используя функцию обхода в разработанной программе произвести чтение структуры бинарного дерева и выполнить его обход, сохранив последовательность прохождения вершин в текстовом файле (.txt).

Бинарное дерево

```
#include <iostream>
#include <vector>
using namespace std;

struct NodeTree
{
    int data;
    NodeTree* left;
    NodeTree* right;
    NodeTree() : left(nullptr), right(nullptr), data(0) {};
};

class Tree
{
private:
    NodeTree* root;
    int countOfRoots;
    vector <int> forOutPut;
public:
    Tree() : root(nullptr), countOfRoots(0) {};

    bool isEmpty()
    {
        if (root == nullptr)
            return true;
        else
            return false;
    }

    void Insert(int value)
    {
        NodeTree* node = new NodeTree; //выделяем в буфер "node" память для будущего присвоения этой памяти в нужное место
        node->data = value;
        if (isEmpty()) //дерево пусто или вершина отсутствует
        {
            root = node;
            countOfRoots++;
        }
        else
        {
```

```
            NodeTree* fakeRoot = root; //тот указатель по которому мы будем двигаться, чтобы не менять основное поле класса
            while (true) //бесконечное движение
            {
                if (value == fakeRoot->data) return; //если нашли совпадение элемента - выход
                if (value > fakeRoot->data) //если наш вставляемый элемент больше текущего, на котором мы "стоим", то узнаем равен ли справа от него элемент "нулптр"
                {
                    if (fakeRoot->right == nullptr) //если равен, то вставляем на это место нашу выделенную память с новым значением и ретюрнем
                    {
                        fakeRoot->right = node;
                        countOfRoots++;
                        return;
                    }
                    fakeRoot = fakeRoot->right; //если не равен, то цикл не ретюрнется и тогда наш "буфер" по которому мы двигаемся сместится просто на шаг вправо
                }
                if (value < fakeRoot->data) //то же самое, что и с "правом", только с "левом"
                {
                    if (fakeRoot->left == nullptr)
                    {
                        fakeRoot->left = node;
                        countOfRoots++;
                        return;
                    }
                    fakeRoot = fakeRoot->left;
                }
            }

        }

    }

    bool IsFound(int _value)
    {
        NodeTree* fakeRoot = root;
        while (true)
        {
            if (fakeRoot == nullptr) return false;
            if (fakeRoot->data == _value) return true;
            if (fakeRoot->data > _value)
            {
                if (fakeRoot->left == nullptr)
                    return false;
                fakeRoot = fakeRoot->left;
            }
            if (fakeRoot->data < _value)
            {
                if (fakeRoot->right == nullptr)
                    return false;
                fakeRoot = fakeRoot->right;
            }
        }
    }

    void toPushAllElementsIntoVector(NodeTree * p)
    {
        if (p != nullptr)
        {
            toPushAllElementsIntoVector(p->left);
            forOutPut.push_back(p->data);
            toPushAllElementsIntoVector(p->right);
        }
    }

    vector<int> GetFullVector()
    {
        toPushAllElementsIntoVector(root);
```

```
        return forOutPut;
    }

    void ClearVector()
    {
        forOutPut.clear();
    }

    int returnDelete(int item)
    {
        NodeTree** q, * z;

        q = &root;
        z = root;
        //Поиск элемента удаления
        for (;;)
        {
            if (z == nullptr)
                return NULL;
            else if (item == z->data)
                break;
            else if (item > z->data)
            {
                q = &z->right;
                z = z->right;
            }
            else
            {
                q = &z->left;
                z = z->left;
            }
        }
        //Первый случай (удаляемый узел (на который указывает z) не имеет дочернего правого
узла)
        if (z->right == nullptr)
            *q = z->left;
        else
        {
            NodeTree* y = z->right;
            /*Второй случай: удаляемый узел (на который указывает z) имеет необязательный левый
и
            обязательный правый дочерний узел, но тот не имеет левого дочернего узла. */
            if (y->left == nullptr)
            {
                y->left = z->left;
                *q = y;
            }
           /*Третий случай: удаляемый узел имеет левый и правый дочерние узлы и те тоже имеют
левый и
           правый дочерние узлы*/
            else
            {
                NodeTree* x = y->left;
                while (x->left != nullptr)
                {
                    y = x;
                    x = y->left;
                }
                y->left = x->right;
                x->left = z->left;
                x->right = z->right;
                *q = x;
            }
        }
        countOfRoots--;
        free(z); //Удаление элемента
        return 1;
    }
```

```
};

void OutPut(vector<int> _vector)
{
    for (unsigned i = 0; i < _vector.size(); i++)
    {
        cout << _vector[i] << " ";
    }
}

void Find(bool _bool)
{
    if (_bool)
        cout << "Совпадение найдено!";
    else
        cout << "Совпадений не найдено!";
}

void Delete(int a)
{
    if (a)
        cout << "Элемент найден и удален!";
    else
        cout << "Элемент не был найден!";
}

int main()
{
    setlocale(LC_ALL, "Russian");
    int countOfElements = 0;
    Tree tr;
    int choice = 0, element = 0;
    cout << "1. Добавить элементы в дерево\n2. Поиск элемента\n3. Вывод дерева\n4. Удалить
элемент" << endl;
    while (true)
    {
        cout << "\nВыберите действие : ";
            cin >> choice;
        switch (choice)
        {
        case 1:
        {
            cout << "Сколько элементов вы хотите добавить в дерево? : ";
            cin >> countOfElements;
            int* arr = new int[countOfElements];
            cout << "Введите элементы через пробел: ";
            for (int i = 0; i < countOfElements; i++)
            {
                cin >> arr[i];
                tr.Insert(arr[i]);
            }
            delete[] arr;
            break;
        }
        case 2:
        {
            cout << "Введите значение элемента: ";
            cin >> element;
            Find(tr.IsFound(element));
            break;
        }
        case 3:
        {
            OutPut(tr.GetFullVector());
            tr.ClearVector();
            break;
        }
        case 4:
        {
```

```
            cout << "Введите удаляемый элемент: ";
            cin >> element;
            Delete(tr.returnDelete(element));
        }
        }
    }
    return 0;
}
```

```
D:\МПУ\VisualStudio\бинарное дерево\Debug\бинарное дерево.exe
1. Добавить элементы в дерево
2. Поиск элемента
3. Вывод дерева
4. Удалить элемент

Выберите действие : 1
Сколько элементов вы хотите добавить в дерево? : 7
Введите элементы через пробел: 5 3 6 4 8 1 2

Выберите действие : 3
1 2 3 4 5 6 8
Выберите действие : 2
Введите значение элемента: 1
Совпадение найдено!
Выберите действие : 2
Введите значение элемента: 8
Совпадение найдено!
Выберите действие : 2
Введите значение элемента: 7
Совпадений не найдено!
Выберите действие : 4
Введите удаляемый элемент: 4
Элемент найден и удален!
Выберите действие : 4
Введите удаляемый элемент: 1
Элемент найден и удален!
Выберите действие : 4
Введите удаляемый элемент: 8
Элемент найден и удален!
Выберите действие : 4
Введите удаляемый элемент: 7
Элемент не был найден!
Выберите действие : 3
2 3 5 6
Выберите действие :
```