МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Деревья

Студентка гр. 9382 _____ Круглова В.Д.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить нелинейную структуру — дерево, способы ее реализации и рекурсивной обработки. Получить навыки решения задач по обработке деревьев, как с использованием рекурсивных функций, так и с использованием функций не имеющих рекурсивной природы.

Основные теоретические положения.

Дерево – конечное множество T, состоящее из одного или более узлов, таких, что:

а) имеется один специально обозначенный узел, называемый корнем данного дерева;

б) остальные узлы (исключая корень) содержатся в $m \geq 0$ попарно не пересекающихся множествах T_1, T_2, ..., T_m, каждое из которых, в свою очередь, является деревом.

Лист (концевой узел) — узел множество поддеревьев которого пусто.

Упорядоченное дерево — дерево в котором важен порядок перечисления его поддеревьев.

Лес – множество (обычно упорядоченное), состоящее из некоторого (быть может, равного нулю) числа непересекающихся деревьев.

Бинарное дерево - конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом.

Задание.

2д. Для заданного бинарного дерева b типа BT с произвольным типом элементов:

- определить максимальную глубину дерева b, т. е. число ветвей в самом длинном из путей от корня дерева до листьев;

- вычислить длину внутреннего пути дерева b, т. е. сумму по всем узлам длин путей от корня до узла.

Описание структуры данных для реализации бинарного дерева.

Для реализации бинарного дерева через динамическую память был создан класс Btree. В нем определено 3 поля: *m_l_tree* — для хранения указателя на левое поддерево, либо нулевого указателя в случае отсутствия этого поддерева; *m_r_tree* — для хранения указателя на правое поддерево, либо нулевого указателя в случае его отсутствия; *m_BTroot* — для хранения значения корня бинарного дерева.

Также определено 3 метода для работы с этим классом: get_left() - для получения значения поля *m_l_tree*; get_right() - для получения значения поля *m_l_right*; get_Btroot() - для получения значения поля *m_Btroot.*

Описание алгоритма.

Чтобы хранить бинарное дерево создается класс BTree рекурсивной природы с использованием динамической памяти.

Дерево заполняется с помощью рекурсивной функции readBT(). В ней идет посимвольное считывание из потока ввода: при нахождении символа пустого дерева „\“ - будет возвращено nullptr, при любом другом символе — функция вызовет себя еще 2 раза для считывания правого и левого поддеревьев и объединит возвращенные значения с корнем этого уровня в одно бинарное дерево с помощью функции ConsBT().

Удаляется дерево с помощью рекурсивной функции destroy(). В ней дважды вызывается сама эта функция для правого и левого поддеревьев, после чего удаляется память выделенная под сам объект.

Проверка на пустоту указателя на бинарное дерево выполняется с помощью функции is_nullptr().

Чтобы посчитать максимальную глубину, рекурсивная функция findMaxDepth проходится по дереву в порядке КЛП и фиксирует изменения в глубине с помощью разыменованного указателя на максимальный элемент.

Чтобы найти ДВП используется рекурсивная функция findInnerWay, которая также в порядке КЛП проходит по дереву и меняет поля объекта, созданного для корня, чтобы после рекурсивного обхода по дереву по формуле вычислить ДВП и вернуть объект класса InnerWayAndCount, а потом извлечь с помощью метода getInnerWay искомый путь. Каждая функция сопровождается соответствующей назначению отладочной информацией.

Описание функций.

template<typename T>
bool isNullptr(BTree<T>* a)
1) Принимает на вход указатель на бинарное дерево произвольного типа данных
2) Возвращает логическое значение bool.
3) Проверяет указатель на то, равен ли он nullptr.

template<typename T>
BTree<T>* ConsBT(const T x, BTree<T>* lst, BTree<T>* rst)
1) Принимает на вход 2 указателя на бинарные деревья произвольного типа и значение произвольного типа данных
2) Возвращает указатель на бинарное дерево произвольного типа.
3) Объединяет корень и 2 бинарных дерева в 1 бинарное дерево.

template<typename T>
void destroy (BTree<T>* b)
1) Принимает на вход указатель на бинарное дерево, хранящее элементы произвольного типа данных.
2) Рекурсивно очищает дерево.

template<typename T>
BTree<T>* readBT()
1) Возвращает указатель на бинарное дерево произвольного типа данных.
2) Считывает из потока ввода бинарное дерево.

template<typename T>
void writeBT(BTree<T> *b)
1) Принимает на вход указатель на бинарное дерево произвольного типа данных.
2) Выводит бинарное дерево.

void printEmptyDepth(int depth)
1) Принимает на вход глубину.
2) Выводит табуляции указанное количество раз.

Описание основных функций.

template<typename T>
void findMaxDepth(BTree<T>* a, int *max, int depth)
1) Принимает на вход бинарное дерево произвольного типа, указатель на максимальную глубину, глубину рекурсии.
2) Если переданный указатель на бинарное дерево не пуст и если глубина рассматриваемого дерева больше максимальной, то максимум возрастает на 1. После этого функция рекурсивно вызывается для левого поддерева с большей на 1 глубиной рекурсии. Так же для правого поддерева. Все это сопровождается отладочной информацией.

template<typename T>
InnerWayAndCount findInnerWay(BTree<T>* a, int depth)
1) Принимает на вход указатель на бинарное дерево произвольного типа, глубину рекурсии.
2) Ищет длину внутреннего пути по формуле ДВП = ВП правого + ВП левого + кол-во узлов в дереве 1. Создается объект с данными пути и кол-вом узлов для корня, если начального дерева не существует, возвращается объект. Если есть левое поддерево, то вызываем для него эту же функцию с большей на 1 глубиной, результат работы которой будет записан в subObject. Далее прибавляем к количеству узлов данные из левого поддерева, аналогично для пути. Аналогично для правого поддерева. После добавляем к ВП кол-во узлов - 1 и возвращаем объект.
3) Возвращает объект класса InnerWayAndCount.

Пример работы программы

Таблица 1 — Результаты работы

Входные данные	Выходные данные

ab//bc//cd///	Starting finding max depth.
	
	Checking left subtree.
	  Max depth < current depth.
	  Checking left subtree.
	  Binary tree is empty.
	  Checking right subtree.
	  Binary tree is empty.
	Checking right subtree.
	  Checking left subtree.
	  Max depth < current depth.
	  Checking left subtree.
	  Binary tree is empty.
	  Checking right subtree.

	Binary tree is empty. Checking right subtree. Checking left subtree. Max depth < current depth. Checking left subtree. Binary tree is empty. Checking right subtree. Binary tree is empty. Checking right subtree. Binary tree is empty. Ending finding max depth. Max depth = 3 Starting finding inner way of bt. Checking left subtree. Left subtree is empty. Right subtree is empty. Count of nodes in this bt = 1 Inner way in this bt = 0 Checking right subtree. Checking left subtree. Left subtree is empty. Right subtree is empty. Count of nodes in this bt = 1 Inner way in this bt = 0 Checking right subtree. Checking left subtree. Left subtree is empty. Right subtree is empty. Count of nodes in this bt = 1 Inner way in this bt = 0 Right subtree is empty. Count of nodes in this bt = 2 Inner way in this bt = 1 Count of nodes in this bt = 4 Inner way in this bt = 4 Count of nodes in this bt = 6 Inner way in this bt = 9 Ending finding inner way of bt. Inner Way of BT = 9

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 — Результаты тестирования

No	Входные данные	Выходные данные	Комментарии
1	abcd//d//cd//d//bcd//d//cd//d//	Max depth = 3 Inner Way of BT = 34	Полное 4-уровневое бинарное дерево
2	abc//c//bcd///cde///d/e/fgh////	Max depth = 7 Inner Way of BT = 45	большое кол-во уровней по "случайным" сторонам
3	a/b/c/d/e/f/g/h/i/j/k/l/m/n/o/p//	Max depth = 15 Inner Way of BT = 120	большое кол-во уровней по одной стороне дерева
4	a//	Max depth = 0 Inner Way of BT = 0	дерево состоит из одного корня
5	/	Binary tree is empty. Inner Way of BT = 0	Ввод пустого дерева
6	ab//////////////////	Max depth = 1 Inner Way of BT = 1	чрезмерное количество символов "пустоты"
7	a b / / b / /	Max depth = 1 Inner Way of BT = 2	большое количество символов табуляции и пробела между значащими символами
8	a	Enter Binary Tree with '/'	Некорректный ввод
9	a / b / /	Max depth = 1 Inner Way of BT = 1	ввод каждого значащего символа после символа \n

Выводы.

Были изучены и опробованы различные методы работы с бинарными деревьями на языке C++. Была создана программа для реализации и работы с бинарными деревьями, использующая, как рекурсивные функции, так и функции не рекурсивной природы.

ПРИЛОЖЕНИЕ С КОДОМ

main.cpp :

```
#include "headers.h"

int main()
{
    BTree<char> *a; // работаем через указатели

    setlocale(LC_ALL, "rus"); // подключен русский язык

    cout << "Enter Binary Tree with \'/\': " << endl;
    a = readBT<char>(); // явно указываем тип, т.к. нет аргументов
    cout << endl;

    cout << "Entered Binary Tree:\n";
    writeBT(a); // a здесь компилятор сам видит тип
    cout << "\n\n";

    {
    int max_depth = 0;

    cout << "Starting finding max depth.\n" << endl;
    findMaxDepth(a, &max_depth, 0);
    cout << "\nEnding finding max depth.\n" << endl;

    cout << "Max depth = " << max_depth << '\n' << endl;
    }

    {
    cout << "\n\nStarting finding inner way of bt.\n" << endl;
    InnerWayAndCount obj = findInnerWay(a, 0);
    cout << "\nEnding finding inner way of bt.\n" << endl;

    cout << "Inner Way of BT = " << obj.getInnerWay() << endl;
    } // obj будет уничтожен здесь

    return 0;
}

void printEmptyDepth(int depth) // просто печатает табуляции в количестве
глубины
{
    for (int j = 0; j < depth; j++)
        cout << "   ";
    return;
}
```

headers.h :

```
#ifndef HEADERS_H
#define HEADERS_H

#include "InnerWayAndCount.h" // подключили созданный класс
#include <iostream>
using namespace std;
```

```
// базовые функции

template<class T>
class BTree
{
private:
public:
    BTree* m_l_tree = nullptr;
    BTree* m_r_tree = nullptr;
    T m_BTroot = '\0';

    BTree* get_left() {return m_l_tree;};
    BTree* get_right() {return m_r_tree;};
    T get_BTroot() {return m_BTroot;};
};

template<typename T>
bool isNullptr(BTree<T>* a) // проверка на nullptr
{return a == nullptr;}

template<typename T>
BTree<T>* ConsBT(const T x, BTree<T>* lst,  BTree<T>* rst) // объединение корня
и 2 подддеревьев
{
    BTree<T>* p = nullptr;
      p = new BTree<T>;

      if (!isNullptr(p))
    {
            p->m_BTroot = x;
            p->m_l_tree = lst;
            p->m_r_tree = rst;
            return p;
      }
      else
    {
        cerr << "Memory error!\n";
        exit(1);
    }
}

template<typename T>
void destroy (BTree<T>* b) // рекурсивная очистка дерева
{
    if (!isNullptr(b))
    {
            destroy (b->m_l_tree);  // рекурсивный вызов этой же функции для
обоих поддеревьев
            destroy (b->m_r_tree);
            delete b;               // и очистка памяти для дерева, в котором
находимся
            b = nullptr;                  // обнуление висячего указателя
      }
}

template<typename T>
BTree<T>* readBT() // считывание дерева заданного с пом. '/'
{
      T ch;
      BTree<T> *p, *q;
      cin >> ch;
```

```
      if (ch == '/')
            return nullptr;
      else
      {
            p = readBT<T>(); // рекурсивный вызов этой же функции для создания
левого поддерева
            q = readBT<T>(); // для создания правого поддерева
            return ConsBT(ch, p, q); // объединение поддеревьев и корня в одно
бинарное дерево
      }
}

template<typename T>
void writeBT(BTree<T> *b) // вывод дерева в том же виде, в котором оно и задано
{
      if (!isNullptr(b))
    {
            cout << b->get_BTroot();
            writeBT(b->get_left());
            writeBT(b->get_right());
      }
      else cout << '/';
}

// специфичные для конкретного варианта функции

void printEmptyDepth(int depth);

template<typename T>
void findMaxDepth(BTree<T>* a, int *max, int depth)
{
    if (!isNullptr(a)) // если переданный указатель не пуст (дерево есть)
    {
        if (*max < depth) // если глубина нынешнего дерева вдруг больше
максимальной
        {
            { // отладочная информация
                printEmptyDepth(depth);
                cout << "Max depth < current depth." << endl;
            }
            *max += 1;  // то значение значение между ними различается лишь на 1
        }

        { // отладочная информация
            printEmptyDepth(depth);
            cout << "Checking left subtree." << endl;
        }
        findMaxDepth(a->get_left(), max, depth + 1); // вызвали эту же функцию
для левого поддерева

        { // отладочная информация
            printEmptyDepth(depth);
            cout << "Checking right subtree." << endl;
        }
        findMaxDepth(a->get_right(), max, depth + 1); // вызвали эту же функцию
для правого поддерева
    }
    else // отладочная информация
    {
        printEmptyDepth(depth);
        cout << "Binary tree is empty." << endl;
    }
}
```

```
template<typename T>
InnerWayAndCount findInnerWay(BTree<T>* a, int depth)
{
    // функция основана на формуле: длина внутреннего пути = внутренний путь правого поддерева + внутренний путь левого поддерева + количество узлов в дереве - 1

    InnerWayAndCount object; // создали объект с полями (way = 0, count = 1) (корень тоже узел, отсюда и 1)

    if((depth == 0) && isNullptr(a)) // если начального дерева не существует
    {
        cout << "Binary tree is empty." << endl;
        return object;
    }

    if (!isNullptr(a->get_left())) // если есть левое поддерево
    {
        { // отладочная информация
            printEmptyDepth(depth);
            cout << "Checking left subtree." << endl;
        }
        // то вызовем для него эту же функцию
        InnerWayAndCount subObject = findInnerWay(a->get_left(), depth + 1); // этот объект уничтожится уже при выходе из этого блока if

        object.addCount(subObject.getCount()); // прибавили количество узлов из левого поддерева
        object.addInnerWay(subObject.getInnerWay()); // прибавили внутренний путь из левого поддерева
    }
    else // отладочная информация
    {
        printEmptyDepth(depth);
        cout << "Left subtree is empty." << endl;
    }

    if (!isNullptr(a->get_right())) // если есть правое поддерево
    {
        { // отладочная информация
            printEmptyDepth(depth);
            cout << "Checking right subtree." << endl;
        }
        InnerWayAndCount subObject = findInnerWay(a->get_right(), depth + 1); // то вызовем для него эту же функцию

        object.addCount(subObject.getCount()); // прибавили количество узлов из правого поддерева
        object.addInnerWay(subObject.getInnerWay()); // прибавили внутренний путь из правого поддерева
    }
    else // отладочная информация
    {
        printEmptyDepth(depth);
        cout << "Right subtree is empty." << endl;
    }

    object.addInnerWay(object.getCount() - 1); // добавляем к внутреннему пути кол-во узлов - 1 (для листьев после действия ничего не поменяется)

    { // отладочная информация
        printEmptyDepth(depth);
```

```
        cout << "Count of nodes in this bt = " << object.getCount() << endl;

        printEmptyDepth(depth);
        cout << "Inner way in this bt = " << object.getInnerWay() << endl;
    }

    return object; // объект передается по значению -> там, где он будет
приниматься, вызовется конструктор копирования по-умолчанию
} // здесь будет уничтожен object



#endif
```

InnerWayAndCount.h :

```
class InnerWayAndCount
{ // здесь нет сложных данных, поэтому прописывать иные конструкторы нет нужды
(копирование и так будет работать нормально)
private:
    int m_count; // количество узлов в дереве
    int m_innerWay; // внутренний путь дерева
public:
    InnerWayAndCount(); // конструктор создания со списком инициализации

    int getCount();
    int getInnerWay();
    void addCount(int k);
    void addInnerWay(int k);
};
```

InnerWayAndCount.cpp :

```
#include "headers.h"


int InnerWayAndCount::getCount(){return m_count;};
int InnerWayAndCount::getInnerWay(){return m_innerWay;};
void InnerWayAndCount::addCount(int k){m_count += k;};
void InnerWayAndCount::addInnerWay(int k){m_innerWay += k;};


InnerWayAndCount::InnerWayAndCount(): m_count{1}, m_innerWay{0} // конструктор
создания со списком инициализации
{};
```