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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

Тема: Идеально сбалансированное дерево поиска

Студент гр. 9304 _____ Цаплин И.В.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с понятием идеально сбалансированного бинарного дерева поиска. Реализовать идеально сбалансированное бинарное дерево поиска на языке программирования C++.

Задание.

Вариант 21.

Реализовать идеально сбалансированное бинарное дерево поиска. По заданной последовательности элементов построить идеально сбалансированное бинарное дерево поиска. Для построенного дерева проверить, входит ли в него элемент e типа Elem, и если входит, то в скольких экземплярах. Добавить элемент e в структуру данных. Предусмотреть возможность повторного выполнения с другим элементом.

Описание алгоритма работы.

Программа принимает на вход строку, в которой записана последовательность чисел, которые входят в идеально сбалансированное бинарное дерево поиска.

Затем строка с помощью функции checkString() проверяется на корректость. Если строка некорректна, выводится сообщение об этом, программа завершается. Если строка корректна, значения чисел записываются в вектор.

Из полученного вектора создаётся идеально сбалансированное бинарное дерево поиска. Затем программа принимает на вход элемент, который необходимо вставить в дерево и элемент, который необходимо найти в дереве. С помощью метода insert() осуществляется вставка в дерево. Затем с помощью метода printLevelOrder() программа выводит полученное дерево по уровням в стандартный поток вывода.

Затем с помощью метода find() осуществляется поиск элемента в дереве. Затем программа выводит количество, найденных элементов, или сообщение о том, что элемента в дереве нет.

Формат входных и выходных данных.

Программа принимает на вход строку, состоящую из целых чисел, разделенных любым количеством пробелов, из этих чисел составляется идеально сбалансированное бинарное дерево. Затем программа принимает число, которое необходимо добавить в дерево и число, число, которое необходимо найти в дереве.

Программа выводит полученное идеально сбалансированное бинарное дерево поиска по уровням. И сообщение о том, сколько раз в дереве встречается требуемый элемент. Если требуемого элемента в дереве нет, об этом выводится сообщение.

Описание основных структур данных и функций.

1. Class BinTreeNode – узел бинарного дерева:
   - std::shared_ptr<BinTreeNode> left — указатель на левое поддерево
   - std::shared_ptr<BinTreeNode> right — указатель на правое поддерево
   - std::weak_ptr<BinTreeNode> parent — указатель на родителя
   - T data — данные типа T
2. Class BinTree – бинарное дерево:
   - std::shared_ptr<BinTreeNode<T>> head – указатель на корень дерева
   - copyBinTree() – метод копирования дерева.

Принимает указатель на узел другого дерева и его родителя, возвращает указатель на копию узла. Метод работает рекурсивно. Сначала создаётся копия текущего узла, затем метод вызывается для правого и левого поддеревьев. Данный метод используется конструктором копирования и оператором копирования.

- vecToBin() – метод, создающий идеально сбалансированное дерево из вектора элементов.

Метод принимает количество элементов в векторе, ссылку на номер текущего элемента и ссылку на вектор. Если число узлов равно нулю возвращает nullptr. Иначе создаёт пустой узел curHead. Вызывает метод

vecToBin(), уменьшив число узлов в два раза, результат работы записывается в поле left узла curHead. Записывает элемент под текущим номером в узел curNode и увеличивает текущий номер на единицу. Вызывает метод vecToBin(), уменьшив число узлов в два раза и отняв единицу, результат работы записывается в поле right узла curHead. Возвращает указатель на узел curHead.

- collect() – метод, собирающий все элементы дерева в вектор.

Записывает данные из текущего узла в вектор, и вызывает метод collect для левого и правого потомков.

- insert() – метод вставки элемента в дерево.

С помощью метода collect() получает вектор, содержащий все элементы дерева. Затем добавляет в вектор элемент, который необходимо вставить и перестраивает дерево с помощью метода vecToBinTree().

- find() – метод поиска элемента по значению.

Если поле data текущего узла равно элементу, который необходимо найти возвращает сумму результатов работы метода find для левого и правого потомков и единицы.

Если поле data текущего узла больше элемента, который необходимо найти возвращает результат работы метода find для левого потомка. Иначе возвращает результат работы метода find для правого потомка.

- printTree() – вывод дерева.

Метод работает рекурсивно. Сначала метод вызывается для левого поддерева. Затем печатается корневой элемент. Затем метод вызывается для правого поддерева.

- printLevelOrder() – вывод дерева по уровням.
- printLevel() – вывод заданного уровня дерева.
- height() – подсчёт высоты дерева.
- getHead() - возвращает указатель на голову списка.

Функция checkString() - проверяет, что строка корректна.

Функция принимает ссылку на строку. Затем функция проверяет, что строка состоит из последовательностей цифр, перечисленных через пробел.

Исключением является знак минуса, который стоит после пробела, и после которого стоит хотя бы одна цифра.

Разработанный код см. в приложении А.

Тестирование.

Для проведения тестирования был написан bash-скрипт tests_script. Скрипт запускает программу с определёнными входными данными и сравнивает полученные результаты с готовыми ответами. Для каждого теста выводится сообщение TestX <входные данные> passed или TestX <входные данные> failed. Для каждого теста выводится ожидаемый результат и полученный. Полученные в ходе работы файлы с выходными данными удаляются.

Результаты тестирования см. в приложении Б.

Выводы.

Было изучено понятие идеально сбалансированного бинарного дерева поиска. Реализовано идеально сбалансированное бинарное дерево поиска на языке программирования C++.

Реализована программа, котороя создаёт идеально сбалансированное бинарное дерево поиска из набора элементов, добавляет заданный элемент в дерево и находит заданные эелемент в дереве. Проведено тестирование работы программы.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: lab5.cpp

```
#include <memory>
#include <iostream>
#include <string>
#include <algorithm>
#include <queue>
#include <sstream>

template<typename T>
class BinTree;

template<typename T>
class BinTreeNode{
public:
    std::shared_ptr<BinTreeNode> left;
    std::shared_ptr<BinTreeNode> right;
    std::weak_ptr<BinTreeNode> parent;
    T data;
    friend class BinTree<T>;
    BinTreeNode(std::shared_ptr<BinTreeNode> left,
std::shared_ptr<BinTreeNode> right, const
std::shared_ptr<BinTreeNode>& parent , T data):
left(std::move(left)), right(std::move(right)), parent(parent),
data(data)
    {
    }
};

template<typename T>
class BinTree {
public:
    BinTree(std::vector<T> vec){
        std::sort(vec.begin(), vec.end());
        int counter = 0;
        head = vecToBinTree(vec.size(), counter, vec, nullptr);
    }

    ~BinTree() = default;

    BinTree(const BinTree& other) {
        head = copyBinTree(other.head, nullptr);
    }

    BinTree& operator= (const BinTree& other){
        head = copyBinTree(other.head, nullptr);
        return *this;
    }

    BinTree(BinTree &&other) {
```

```
        head = std::move(other.head);
    }

    BinTree& operator= (BinTree &&other) {
        head = std::move(other.head);
        return *this;
    }

    std::shared_ptr<BinTreeNode<T>> copyBinTree(const
std::shared_ptr<BinTreeNode<T>>& otherHead, const
std::shared_ptr<BinTreeNode<T>>& headParent) {
        if (otherHead == nullptr) {
            return nullptr;
        }

        if (otherHead == head) {
            return head;
        }

        std::shared_ptr<BinTreeNode<T>> curHead =
std::make_shared<BinTreeNode>(nullptr, nullptr, headParent,
otherHead->data);
        if (otherHead->left != nullptr) {
            curHead->left = copyBinTree(otherHead->left, curHead);
        }

        if (otherHead->right != nullptr) {
            curHead->right = copyBinTree(otherHead->right,
curHead);
        }

        return curHead;
    }

    void insert(T data){
        std::vector<T> vec{data};
        collect(getHead(), vec);
        std::sort(vec.begin(), vec.end());
        int counter = 0;
        head = vecToBinTree(vec.size(), counter, vec, nullptr);
    }

    int find(const std::shared_ptr<BinTreeNode<T>>& curNode, T
dataToFind){
        if(curNode == nullptr){
            return 0;
        }
        if(dataToFind == curNode->data){
            return find(curNode->left,dataToFind) + find(curNode-
>right,dataToFind) + 1;
        }
        if(dataToFind < curNode->data)
            return find(curNode->left,dataToFind);
```

```
        return find(curNode->right,dataToFind);
    }


    void printTree(const std::shared_ptr<BinTreeNode<T>>& curNode)
{
        if (!curNode) {
            return;
        }
        printTree(curNode->left);
        std::cout << curNode->data << ' ';
        printTree(curNode->right);
    }

    void printLevelOrder(const std::shared_ptr<BinTreeNode<T>>&
curNode){
        for(int d = 0; d < this->height(this->getHead()); d++){
            printLevel(curNode, d);
            std::cout << "\n";
        }
    }

    void printLevel(const std::shared_ptr<BinTreeNode<T>>&
curNode, int level){
        if (curNode == nullptr) {
            std::cout << "_ ";
            return;
        }
        if (level == 0)
            std::cout << curNode->data << ' ';
        else {
            if(level > 0) {
                printLevel(curNode->left, level - 1);
                printLevel(curNode->right, level - 1);
            }
        }
    }


    std::shared_ptr<BinTreeNode<T>> getHead(){
        return head;
    }

    int height(const std::shared_ptr<BinTreeNode<T>>& curNode){
        if(curNode == nullptr){
            return 0;
        }
        if(height(curNode->left) > (height(curNode->right))){
            return height(curNode->left)+1;
        }
        return height(curNode->right)+1;
    }
```

```
private:
    std::shared_ptr<BinTreeNode<T>> head;

    std::shared_ptr<BinTreeNode<T>> vecToBinTree(int numOfNodes,
int& curNode, const std::vector<T>& vec, const
std::shared_ptr<BinTreeNode<T>>& parent){
        if(numOfNodes == 0) {
            return nullptr;
        }
        std::shared_ptr<BinTreeNode<T>> curHead =
std::make_shared<BinTreeNode<T>>(nullptr, nullptr, parent, 0);
        curHead->left = vecToBinTree(numOfNodes/2, curNode, vec,
curHead);
        curHead->data = vec[curNode];
        curNode++;
        curHead->right = vecToBinTree(numOfNodes - numOfNodes/2 -
1, curNode, vec, curHead);
        return curHead;
    }

    void collect(const std::shared_ptr<BinTreeNode<T>>&
curNode,std::vector<T>& vec){
        if(curNode == nullptr){
            return;
        }
        vec.push_back(curNode->data);
        this->collect(curNode->left, vec);
        this->collect(curNode->right, vec);
    }
};

bool checkString(std::string& str){
    auto iterator = str.cbegin();
    while(iterator != str.cend()){
        if(*iterator == '-'){
            iterator++;
        }
        if(!isdigit(*iterator)){
            return false;
        }

        while(isdigit(*iterator)){
            iterator++;
        }

        if((*iterator != ' ') && (iterator != str.cend())){
            return false;
        }

        while(*iterator == ' '){
            iterator++;
        }
```

```
    }
    return true;
}

int main() {
    std::string inString{};
    std::getline(std::cin, inString);
    std::vector<int> vecInt;
    if (!checkString(inString)) {
        std::cout << "Incorrect string\n";
        return 0;
    }

    if (!inString.empty()) {
        std::stringstream iss(inString);
        int number;
        while (iss >> number)
            vecInt.push_back(number);
    }
    std::sort(vecInt.begin(), vecInt.end());
    BinTree tree(vecInt);
    int elemToInsert;
    std::cin >> elemToInsert;
    int elemToFind;
    std::cin >> elemToFind;
    tree.insert(elemToInsert);
    tree.printLevelOrder(tree.getHead());
    if(int counter = tree.find(tree.getHead(), elemToFind)){
        std::cout << "Elem " << elemToFind << " found: "<< counter
<< "\n";
    }else{
        std::cout << "Elem not found" << "\n";
    }
}
```

Название файла: tests_script

```
#! /bin/bash


printf "\nRunning tests...\n\n"

for n in {1..8}

do

    ./lab5 < "./Tests/tests/test$n.txt" > "./Tests/out/out$n.txt"

    printf "\e[1;34mTest$n:\e[0m\n"

    cat "./Tests/tests/test$n.txt"

    if cmp "./Tests/out/out$n.txt" "./Tests/true/true_out$n.txt" >

/dev/null; then

            printf "\e[1;32m - Passed\e[0m\n"

    else
```

```
            printf "\e[1;31m - Failed\e[0m\n"
    fi
    printf "\nDesired result:\n"
    cat "./Tests/true/true_out$n.txt"
    printf "\nActual result:\n"
    cat "./Tests/out/out$n.txt"
    printf "\n"
done
rm ./Tests/out/out*
```

Название файла: Makefile

```
lab5: Source/lab5.cpp
        g++ -Wall -std=c++17 Source/lab5.cpp -o lab5

run_tests: lab5
        ./tests_script
```

ПРИЛОЖЕНИЕ Б
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

№	Входные данные	Выходные данные
1	1 2 3 4 5 6 6	4 2 6 1 3 5 _ Elem 6 found: 1
2	7 6 5 4 3 2 1 8 8	5 3 7 2 4 6 8 1 _ _ _ _ _ _ _ Elem 8 found: 1
3	5 5 5 5 5 1 5	5 5 5 1 5 5 _ Elem 5 found: 5
4	1 2 0 1	1 0 2 Elem 1 found: 1
5	2 3 4 5 5	4 3 5 2 _ _ _ Elem 5 found: 1
6	10 3 2 8 3 0 6 3 3	3 3 8 2 3 6 10 0 _ _ _ _ _ _ _ Elem 3 found: 3
7	68 30 -4 2 1 -4 -50 3 30	2 -4 30

	-4	-4 1 30 68 -50 _ _ _ 3 _ _ _ Elem -4 found: 2
8	1 2 3 4 5a 5 5	Incorrect string