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: Бинарные деревья поиска

Студент гр. 9382 _____ Герасев Г.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучить алгоритмы работы с бинарными деревьями поиска. Научиться создавать бинарные деревья поиска по введенным ключам, рисовать деревья, удалять требуемые узлы бинарного дерева поиска.

Задание.

БДП: случайное* БДП; действие: 1+2в

Основные теоретические положения.

Бинарное дерево поиска – дерево, где у каждого узла есть 2 поддерева и в каждом узле хранится по одному значению. Все узлы левее корневого «меньше или равны» корневого, а все узлы правее корневого «больше или равны него».

Функции и структуры данных.

Создан класс бинарного дерева поиска (class BinarySearchTree), каждый из узлов которого содержит хранимое значение (int data), указатель на левое и правое поддерево (BinarySearchTree* left,* right) и число узлов в данном дереве (unsigned int quantityOfNodes). Последнее нужно для реализации случайной вставки, где случайность пропорциональна количеству узлов в дереве. При инициализации (BinarySearchTree(int data = 0)) создается пустое дерево с переданным значением (без поддеревьев).

Для добавления значений в дереве создается метод (BinarySearchTree* insert(int inputData)), который добавляет значение в дерево методом случайной вставки (случайность влияет на то, будет ли вставлено значение в корень дерева, или в поддерево). Данный метод описан в разделе «Описание алгоритма». Соответственно для него реализуется дополнительный метод вставки значения в корень (BinarySearchTree* insertInRoot(int data)), чей метод работы описан там же. Создаются методы локального поворота дерева, требуемые для его работы (BinarySearchTree* rotateLeft(), BinarySearchTree* rotateRight()).

Также реализуется метод удаления первого встретившегося переданного элемента (BinarySearchTree* deleteFirst(int data)). Данный метод работает на слиянии деревьев, если одно из них меньше другого, данная операция реализуется в виде функции join (BinarySearchTree* join(BinarySearchTree* smallerTree, BinarySearchTree* biggerTree)).

Для удобства чтения создается метод, рисующий дерево на экране (void draw(string buffer = "", bool isLast = true)).

Из не перечисленных выше методов, в виду очевидности их работы или их низкой значимости:

int getQuantityOfNodes() – метод, возвращающий количество узлов в дереве (соответствующее поле).

void updateQuantityOfNodes() – метод, проходящий по дереву, и выставляющий правильные значения поля quantityOfNodes. Используется после преобразований дерева.

BinarySearchTree(const BinarySearchTree & binarySearchTree)

~BinarySearchTree() – оператор копирования и деструктор.

void greetingMessage() – функция, выводящая сообщение-приветствие

BinarySearchTree* stdInputTree() – функция, создающая дерево из стандартного потока.

void stdInputCase() – обработчик случая стандартного ввода.

void fileInputCase(string path) – обработчик случая файлового ввода.

int main(int argc, char *argv[]) – функция, запускающаяся при запуске программы.

Описание алгоритма.

Для реализации вставки в корень требуются левые и правые повороты вокруг узла. Данные операции реализуются в виде соответствующих методов. Так как переопределение корневого узла невозможно в методе/функции, то все методы или функции возвращают новый корневой узел.

Через эти методы реализуется вставка в корень – в зависимости от значения оно вставляется в левое или правое поддерево, после чего это поддерево поднимается соответствующим поворотом (после рекурсии получается, что переданное значение оказывается в корневом узле).

В конце концов через этот метод реализуется случайная вставка значения в дерево, которая с шансом равным 1/(значение узлов в дереве + 1) вставляет в корень дерева, а в остальных случаях вставляет в левое или правое поддерево (в зависимости от переданного значения).

Функция слияния деревьев случайно выбирает какой из корневых узлов станет новым корневым (пропорционально количеству узлов в деревьях) после чего вызывается рекурсивно с левым/правым поддеревом и оставшимся деревом для слияния.

Метод удаления просто находит нужный элемент сливает у его узла левое и правое поддерево, и возвращает новый корневой узел.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	1 2 3 0 1	Your tree -- └─1 ├─2 │├─3 │├─ empty │└─ empty └─ empty Number to delete -- 1 The result -- └─2 ├─3 ├─ empty └─ empty	
2.	1 2 3 0 2	Your tree -- └─2 ├─3 ├─ empty └─ empty └─1 ├─ empty	

		└─ empty Number to delete -- 2 The result -- └─1 |─3 | ├─ empty | └─ empty └─ empty	
3.	1 2 3 0 3	Your tree -- └─1 |─2 ||─3 || ├─ empty || └─ empty | └─ empty └─ empty Number to delete -- 3 The result -- └─1 |─2 | ├─ empty | └─ empty └─ empty	
4.	1 2 3 0 4	Your tree -- └─1 |─2	

		||─3 || ├─ empty || └─ empty | └─ empty └─ empty Number to delete -- 4 The result -- └─1 |─2 ||─3 || ├─ empty || └─ empty | └─ empty └─ empty	
5.	1 2 3 4 5 6 7 8 9 10 0 3	Your tree -- └─4 |─9 ||─10 || ├─ empty || └─ empty | └─7 | |─8 | | ├─ empty | | └─ empty | └─6 | ├─ empty | └─5 | ├─ empty	

		│ └─ empty └─3 ├─ empty └─1 ├─2 │ ├─ empty │ └─ empty └─ empty Number to delete -- 3 The result -- └─4 ├─9 │├─10 │ │ ├─ empty │ │ └─ empty │ └─7 │ ├─8 │ │ ├─ empty │ │ └─ empty │ └─6 │ ├─ empty │ └─5 │ ├─ empty │ └─ empty └─1 ├─2 │ ├─ empty │ └─ empty	

		└─ empty	
6.	1 10 100 1000 0 1	Your tree -- └─100 |─1000 | ├─ empty | └─ empty └─1 |─10 | ├─ empty | └─ empty └─ empty Number to delete -- 1 The result -- └─100 |─1000 | ├─ empty | └─ empty └─10 ├─ empty └─ empty	

Выводы.

Были изучены алгоритмы работы с бинарными деревьями поиска. Создана программа, создающая бинарные деревья поиска по введенным ключам, рисующая деревья а также удаляющая требуемые узлы бинарного дерева поиска.

Метод случайной вставки дает крайне сбалансированные деревья, особенно учитывая то, что метод не требует полной балансировки дерева после

каждой вставки. То же самое можно сказать и про данный метод удаления, который сохраняет сбалансированность дерева после своей работы, и тем более сохраняющий его свойства как дерева поиска.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: BinarySearchTree.cpp

```
#include "BinarySearchTree.h"

BinarySearchTree::BinarySearchTree(int inputData)
{
   data = inputData;
   quantityOfNodes = 1;
   pointers.left = nullptr;
   pointers.right = nullptr;
}

BinarySearchTree::BinarySearchTree(const BinarySearchTree & binarySearchTree) // Copy operator
{
   data = binarySearchTree.data;

   if (binarySearchTree.pointers.left != nullptr)
      pointers.left = new BinarySearchTree(*binarySearchTree.pointers.left);

   if (binarySearchTree.pointers.right != nullptr)
      pointers.right = new BinarySearchTree(*binarySearchTree.pointers.right);
}

BinarySearchTree::~BinarySearchTree()
{
   if (pointers.left != nullptr)
      delete pointers.left;
   if (pointers.right != nullptr)
      delete pointers.right;
}

int BinarySearchTree::getQuantityOfNodes()
{
   return quantityOfNodes;
}

void BinarySearchTree::updateQuantityOfNodes()
{
   unsigned int quantityOfNodesLeft = 0;
   unsigned int quantityOfNodesRight = 0;
   if (pointers.left != nullptr)
      quantityOfNodesLeft = pointers.left->getQuantityOfNodes();

   if (pointers.right != nullptr)
      quantityOfNodesRight = pointers.right->getQuantityOfNodes();

   quantityOfNodes = 1 + quantityOfNodesLeft + quantityOfNodesRight;
}

BinarySearchTree* BinarySearchTree::rotateLeft() // (A, (B, C)) -> ((A, B), C))
{
   BinarySearchTree* right = pointers.right;
   if (right == nullptr)
      return this;
   pointers.right = right->pointers.left;
```

```
    right->pointers.left = this;

    right->quantityOfNodes = quantityOfNodes;
    updateQuantityOfNodes();
    return right;
}

BinarySearchTree* BinarySearchTree::rotateRight() //((A, B), C)) -> (A, (B, C))
{
    BinarySearchTree* left = pointers.left;
    if (left == nullptr)
        return this;
    pointers.left = left->pointers.right;
    left->pointers.right = this;

    left->quantityOfNodes = quantityOfNodes;
    updateQuantityOfNodes();
    return left;
}

BinarySearchTree* BinarySearchTree::insertInRoot(int inputData)
{
    if (inputData < data)
    {
        if (pointers.left == nullptr)
        {
            pointers.left = new BinarySearchTree(inputData);
            return this;
        }
        else
        {
            pointers.left = pointers.left->insertInRoot(inputData);
            return rotateRight();
        }
    }

    if (pointers.right == nullptr)
    {
        pointers.right = new BinarySearchTree(inputData);
        return this;
    }
    else
    {
        pointers.right = pointers.right->insertInRoot(inputData);
        return rotateLeft();
    }
}

BinarySearchTree* BinarySearchTree::insert(int inputData)
{
    int randNumber = rand();
    srand(randNumber);

    bool stopHere = false;
    if (randNumber%(quantityOfNodes + 1) == 0)
        stopHere = true;

    if (stopHere)
    {
```

```
        BinarySearchTree* res = insertInRoot(inputData);
        updateQuantityOfNodes();
        return res;
    }

    if (inputData < data)
    {
        if (pointers.left == nullptr)
            pointers.left = new BinarySearchTree(inputData);
        else
            pointers.left = pointers.left->insert(inputData);
    }
    else
    {
        if (pointers.right == nullptr)
            pointers.right = new BinarySearchTree(inputData);
        else
            pointers.right = pointers.right->insert(inputData);
    }

    updateQuantityOfNodes();
    return this;
}

BinarySearchTree* join(BinarySearchTree* smallerTree, BinarySearchTree* biggerTree)
{
    if (smallerTree == nullptr)
        return biggerTree;
    if (biggerTree == nullptr)
        return smallerTree;

    int randNumber = rand();
    srand(randNumber);
    bool  goSmaller = false;
    if (randNumber%(smallerTree->getQuantityOfNodes() + biggerTree->getQuantityOfNodes()) < small-
erTree->getQuantityOfNodes())
         goSmaller = true;

    if (goSmaller) {
        smallerTree->pointers.right = join(smallerTree->pointers.right, biggerTree);
        smallerTree->updateQuantityOfNodes();
        return smallerTree;
    } else {
        biggerTree->pointers.left = join(smallerTree, biggerTree->pointers.left);
        biggerTree->updateQuantityOfNodes();
        return biggerTree;
    }
}

BinarySearchTree* BinarySearchTree::deleteFirst(int inputData)
{
    if (inputData == data)
    {
        BinarySearchTree* res = join(pointers.left, pointers.right);
        return res;
    }

    if (inputData < data)
    {
```

```
        if (pointers.left != nullptr)
            pointers.left = pointers.left->deleteFirst(inputData);
    }

    else
    {
        if (pointers.right != nullptr)
            pointers.right = pointers.right->deleteFirst(inputData);
    }

    return this;
}

void BinarySearchTree::draw(string buffer, bool isLast)
{
    string branch = " ├";
    string pipe = "│";
    string end = " └";
    string dash = "─";

    if (isLast)
    {
        cout << buffer << end << dash << data << '\n';
        buffer += "  ";
    }

    else
    {
        cout << buffer << pipe << dash << data << '\n';
        buffer += pipe + " ";
    }

    if (pointers.right != nullptr)
        pointers.right->draw(buffer, false);
    else
        cout << buffer << branch << dash << " empty\n";


    if (pointers.left != nullptr)
        pointers.left->draw(buffer, true);
    else
        cout << buffer << end << dash << " empty\n";
}

void greetingMessage()
{
    cout << "\nFile input example -- ./main -f input.txt\n\n";
    cout << "Hello. Please input null terminated ";
    cout << "sequence of numbers.\nThe programm will make ";
    cout << "binary out of them.\nThen input a number, to ";
    cout << "delete it from the tree.\n";
}

BinarySearchTree* stdInputTree()
{
    int data;
    cin >> data;
    BinarySearchTree* tree = new BinarySearchTree(data);
    cin >> data;
```

```
    while (data != 0)
    {
        tree = tree->insert(data);
        cin >> data;
    }
    return tree;
}

void stdInputCase()
{
    BinarySearchTree* tree = stdInputTree();

    cout << "\nYour tree -- \n";
    tree->draw();

    int data;
    cout << "\nNumber to delete -- ";
    cin >> data;
    tree = tree->deleteFirst(data);

    cout << "\nThe result -- \n";
    tree->draw();
}

void fileInputCase(string path)
{
    ifstream fin;
    fin.open(path);

    int data;
    fin >> data;
    BinarySearchTree* tree = new BinarySearchTree(data);

    while (fin >> data)
    {
        tree = tree->insert(data);
    }

    cout << "\nYour tree -- \n";
    tree->draw();


    cout << "\nNumber to delete -- ";
    cin >> data;

    tree->deleteFirst(data);

    cout << "\nResult -- \n";
    tree->draw();
}

int main(int argc, char *argv[])
{
    srand(time(0));
    if (argc>= 2) // Arguments case
    {
        string flag(argv[1]);
        string path(argv[2]);
```

```
        if (flag.compare("-f") == 0)
            fileInputCase(path); // No obvious way to overload the function
        return 0;
    }
    greetingMessage();
    stdInputCase();
    return 0;
```

Название файла: BinarySearchTree.h

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <cstdlib>

using namespace std;

class BinarySearchTree
{
    struct Pointers
    {
        BinarySearchTree* left = nullptr;
        BinarySearchTree* right = nullptr;
    };

public:
    int data; // in our case the node data == node key, but it's easy to change
    Pointers pointers;
    unsigned int quantityOfNodes; // For random bin search tree only

    BinarySearchTree(int data = 0);
    BinarySearchTree(const BinarySearchTree & binarySearchTree); // Copy operator
    ~BinarySearchTree();

    void draw(string buffer = "", bool isLast = true);

    int getQuantityOfNodes();
    void updateQuantityOfNodes();

    BinarySearchTree* rotateLeft();
    BinarySearchTree* rotateRight();
```

```
    BinarySearchTree* insertInRoot(int data); // Return root pointer
    BinarySearchTree* insert(int data);

    BinarySearchTree* deleteFirst(int data);
};
```