

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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Рандомизированная пирамида поиска»

Студент гр. 8381

Сосновский Д.Н.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2019

Цель работы

Ознакомиться со структурой данных рандомизированной пирамиды поиска. Реализовать программу с этой структурой данных.

Ход работы

Разработка программы производилась на базе ОС Windows 10 в среде разработки QtCreator с использованием фреймворка Qt.

Для выполнения задания был разработан GUI с использованием элемента QGraphicsScene для отображения получаемой пирамиды. Программа была оснащена синтаксическими проверками на правильность ввода. В ходе работы были реализованы абстрактные классы для работа с View и для работы с Controller – некоторое подобие технологии MVC. Вся работа с деревом происходит в самом классе пираамиды. Программа была проверена на различных тестах и успешно справилась со своей задачей.

Так же в GUI есть ещё одно поле, в которое выводятся элементы этой пирамиды в порядке возрастания.

Оценка сложности работы программы

Операция вставки элемента в рандомизированную пирамиду осуществляется за время $O(\log_2 n)$. Тогда сложность всего алгоритма работы программы составляет $O(n \log_2 n)$. Графики анализа времени работы программы приведены в приложении.

Вывод

В ходе выполнения данной лабораторной работы была разработана программа для обращения с рандомизированной пирамидой поиска.

ПРИЛОЖЕНИЕ

Графики оценки сложности

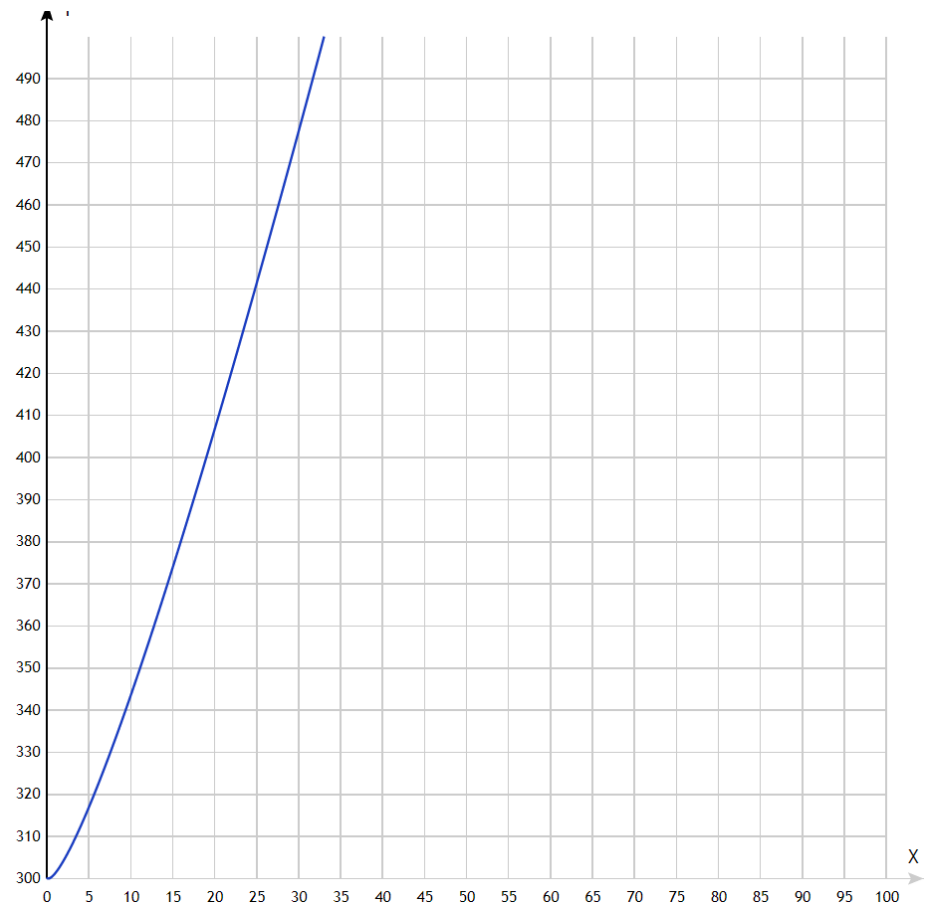


Рисунок 1 - первый график

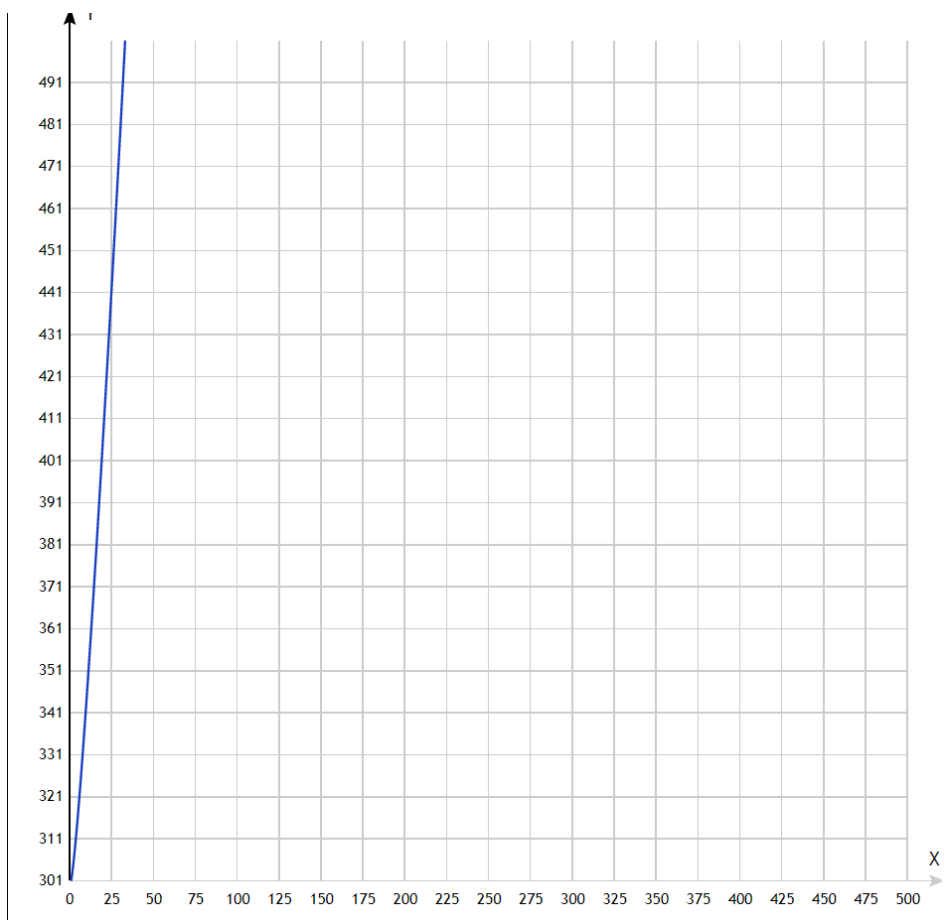


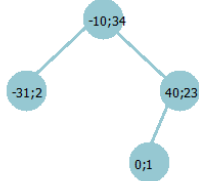


Рисунок 2 - второй график

Примеры работы программы

Входные данные	Выходные данные
$(32+2)1$	Неопознанный символ + на позиции 3
$(32;1),(35;5),(4;3)$	<div>4 32 35</div> 

(30;10),(50;1),(60;43),(10;23),(40;6)	10 30 40 50 60 
(0;1),(-10;34),(40;23),(-31;2)	-31 -10 0 40 

Исходный код программы

```

#ifndef BINARYRANDOMTREAP_H
#define BINARYRANDOMTREAP_H
#include <qrandom.h>
#include <QString>
#include <stdlib.h>
#include <QGraphicsScene>
#include <QGraphicsTextItem>
#include "queue.h"
#include <math.h>
#include "validator.h"
#include "randomgenerator.h"

class BinaryRandomTreap
{
private:
    class Node{
    public:
        QString value;
        int priority;
        Node* left;
        Node* right;
        Node(QString value, int priority);
    };
};

```

```

    bool isContentIntegers = true;

    Node* rotateRight(Node* root);

    Node* rotateLeft(Node* root);

    Node* insert(Node* root, QString insertingValue, int priority);

    void remove(Node* root, Node* parent, bool isLeft, QString& task, int
step);

    Node* find(Node* root, QString value);

    Node* findParent(Node* root, Node* kid);

    Node* root;

    QString getPrefix(Node* root);

    void treePainter(QGraphicsScene *&scene, Node *node, int w, int h,
int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont &font, int
depth);

    int getTreeHeight(Node* node);

    QString getStrTree(Node* node);

    void plotTreeOfIntegers(QString string);

    void plotTreeOfChars(QString string, int* priorities);

public:
    BinaryRandomTreap(QString inputString);

    void removeNode(QString value, QString& task);

    BinaryRandomTreap(int* input, int* priorities, int amount, QString &
task);

    QString getFromLowestToGreatest();

    QGraphicsScene* drawTree();
};

#endif // BINARYRANDOMTREAP_H

#include "binaryrandomtreap.h"

```

```

BinaryRandomTreap::BinaryRandomTreap(QString inputString)
{
    Validator validator;

    validator.checkCorrectness(inputString);

    root = nullptr;

    plotTreeOfIntegers(inputString);
}

BinaryRandomTreap::BinaryRandomTreap(int* input, int* priorities, int
amount, QString& task)
{
    root = nullptr;

    for(int i = 0; i < amount; i++)
    {
        root = insert(root, QString::number(input[i]), priorities[i]);

        task += "War ";
        task += QString::number(i+1);
        task += ": ";
        task += getStrTree(root);
        task += "\n";
    }
}

QString BinaryRandomTreap::getStrTree(Node *node)
{
    QString str = nullptr;

    Queue<Node*> queue;

    Node* tmp;

    queue.enqueue(node);
    while (queue.checkNotNull() != false) {

        tmp = queue.dequeue();

        if(tmp == nullptr)
        {
            str += "#";
            queue.enqueue(nullptr);
            queue.enqueue(nullptr);
        }
    }
}

```

```

        else
        {
            str += tmp->value;
            queue.enqueue(tmp->left);
            queue.enqueue(tmp->right);
        }
    }

    return str;
}

void BinaryRandomTreap::plotTreeOfIntegers(QString string)
{
    int currentPriorityNumber = 0;
    bool lessZero = false;
    bool isPriority = false;
    int number;
    int priority;
    for(int i = 0; i < string.size(); i++)
    {
        QChar currentSymbol = string[i];
        if(currentSymbol == '-')
        {
            lessZero = true;
        }
        else if(currentSymbol == ' ')
        {
            continue;
        }
        else if(currentSymbol == '(')
        {
            isPriority = false;
            lessZero = false;
        }
        else if(currentSymbol == ';')
        {
            isPriority = true;
            lessZero = false;
        }
        else if(currentSymbol == ')')
        {
            root = insert(root, QString::number(number), priority);
        }
        else if(currentSymbol.isNumber())
        {
            QString currentNumber = nullptr;
            while(string[i].isNumber() && i < string.size())
            {
                currentSymbol = string[i];

```



```

        currentNumber += currentSymbol;
        i++;
    }
    i--;

    int current = currentNumber.toInt();

    if(lessZero) current *= -1;

    if(isPriority == false) number = current;
    else priority = current;
    }
}

void BinaryRandomTreap::plotTreeOfChars(QString string, int* priorities)
{
    int currentPriorityNumber = 0;

    for(int i = 0; i < string.size(); i++)
    {
        QChar currentSymbol = string[i];

        root = insert(root, QString(currentSymbol),
priorities[currentPriorityNumber]);

        currentPriorityNumber++;
    }
}

BinaryRandomTreap::Node::Node(QString value, int priority)
{
    this->value = value;
    this->left = nullptr;
    this->right = nullptr;
    this->priority = priority;
}

BinaryRandomTreap::Node* BinaryRandomTreap::rotateRight(Node *root)
{
    Node* leftTree = root->left;
    if(leftTree == nullptr) return root;
    root->left = leftTree->right;
    leftTree->right = root;
    return leftTree;
}

```

```

BinaryRandomTreap::Node* BinaryRandomTreap::rotateLeft(Node *root)
{
    Node* rightTree = root->right;
    if(rightTree == nullptr) return root;
    root->right = rightTree->left;
    rightTree->left = root;
    return rightTree;
}

void BinaryRandomTreap::removeNode(QString value, QString& task)
{
    Node* finded = find(root, value);

    Node* findedParent = findParent(root, finded);

    int step = 1;
    finded->priority = 0;

    if(findedParent == nullptr)
    {
        if(this->root->left == nullptr && this->root->right == nullptr)
        {
            delete this->root;
            this->root = nullptr;
            task += "War ";
            task += QString::number(step);
            task += ": ";
            task += "\n";
            task += getStrTree(this->root);
            task += "\n";
            step++;
            return;
        }
        int n1, n2;
        if(root->left != nullptr) n1 = root->left->priority;
        else n1 = 0;
        if(root->right != nullptr) n2 = root->right->priority;
        else n2 = 0;

        if(n1 >= n2)
        {
            this->root = rotateRight(this->root);
            finded = this->root->right;
            findedParent = this->root;
        }
        else {
            this->root = rotateLeft(this->root);
            finded = this->root->left;
        }
    }
}

```

```

        findedParent = this->root;
    }
    task += "War 1: ";
    task += getStrTree(this->root);
    task += "\n";
    step++;
}

bool isLeft;
if(findedParent->left == finded) isLeft = true;
else isLeft = false;

remove(finded, findedParent, isLeft, task, step);
}

void BinaryRandomTreap::remove(Node *root, Node* parent, bool isLeft,
QString & task, int step)
{
    if(root->left == nullptr && root->right == nullptr)
    {
        delete root;
        if(isLeft) parent->left = nullptr;
        else parent->right = nullptr;

        task += "War ";
        task += QString::number(step);
        task += ": ";
        task += getStrTree(this->root);
        task += "\n";
        step++;
        return;
    }

    int n1, n2;
    if(root->left != nullptr) n1 = root->left->priority;
    else n1 = 0;
    if(root->right != nullptr) n2 = root->right->priority;
    else n2 = 0;

    if(n1 >= n2)
    {
        if(isLeft){
            parent->left = rotateRight(parent->left);
            task += "War ";
            task += QString::number(step);
            task += ": ";

```

```

        task += getStrTree(this->root);
        task += "\n";
        step++;
        remove(parent->left->right, parent->left, false, task, step);
    }
    else
    {
        parent->right = rotateRight(parent->right);
        task += "War ";
        task += QString::number(step);
        task += ": ";
        task += getStrTree(this->root);
        task += "\n";
        step++;
        remove(parent->right->right, parent->right, false, task,
step);
    }
}
else {
    if(isLeft)
    {
        parent->left = rotateLeft(parent->left);
        task += "War ";
        task += QString::number(step);
        task += ": ";
        task += getStrTree(this->root);
        task += "\n";
        step++;
        remove(parent->left->left, parent->left, true, task, step);
    }
    else
    {
        parent->right = rotateLeft(parent->right);
        task += "War ";
        task += QString::number(step);
        task += ": ";
        task += getStrTree(this->root);
        task += "\n";
        step++;
        remove(parent->right->left, parent->right, true, task, step);
    }
}
}

BinaryRandomTreap::Node* BinaryRandomTreap::find(Node* root, QString
value)
{
    if(root == nullptr) return nullptr;

```

```

    if(root->value == value) return root;
    else
    {
        Node* node1 = find(root->left, value);
        Node* node2 = find(root->right, value);

        if(node1 != nullptr) return node1;
        if(node2 != nullptr) return node2;
        else return nullptr;
    }
}

BinaryRandomTreap::Node* BinaryRandomTreap::findParent(Node* root, Node*
kid)
{
    if(root == nullptr) return nullptr;
    if(this->root == kid) return nullptr;
    if(root->right == kid || root->left == kid) return root;
    else {
        Node* n1 = findParent(root->left, kid);
        Node* n2 = findParent(root->right, kid);

        if(n1 != nullptr) return n1;
        if(n2 != nullptr) return n2;
        else return nullptr;
    }
}

```

```

BinaryRandomTreap::Node* BinaryRandomTreap::insert(Node *root, QString
insertingValue, int priority)
{
    if (root == nullptr)
    {
        root = new Node(insertingValue, priority);
        return root;
    }

    int rootValueNumber;
    int insertingValueNumber;

    if(isContentIntegers == true){
        rootValueNumber = root->value.toInt();
        insertingValueNumber = insertingValue.toInt();
    }
    else
    {

```

```

        rootValueNumber = root->value[0].unicode();
        insertingValueNumber = insertingValue[0].unicode();
    }

    if (rootValueNumber > insertingValueNumber)
    {
        root->left = insert(root->left, insertingValue, priority);
        if(root->priority < root->left->priority)
            root = rotateRight(root);
    }
    else if (rootValueNumber <= insertingValueNumber)
    {
        root->right = insert(root->right, insertingValue, priority);
        if(root->priority < root->right->priority)
            root = rotateLeft(root);
    }

    return root;
}

QString BinaryRandomTreap::getFromLowestToGreatest()
{
    return getPrefix(root);
}

QString BinaryRandomTreap::getPrefix(Node *root)
{
    if(root == nullptr) return nullptr;
    QString str = nullptr;
    if(root->left != nullptr)
        str += getPrefix(root->left);
    str += root->value;
    str += " ";
    if(root->right != nullptr)
        str += getPrefix(root->right);
    return str;
}

int BinaryRandomTreap::getTreeHeight(Node* node)
{
    int h1=0;
    int h2=0;

    if(node == nullptr) return 0;

```

```

        if (node->left)
        {
            h1=getTreeHeight(node->left);
        }
        if (node->right)
        {
            h2=getTreeHeight(node->right);
        }
        return(std::max(h1,h2)+1);
    }

QGraphicsScene* BinaryRandomTreap::drawTree()
{
    QGraphicsScene* scene = new QGraphicsScene;

    if (root == nullptr)
        return scene;
    scene->clear();
    QPen pen;
    QColor color;
    color.setRgb(150, 200, 210);
    pen.setColor(color);
    QBrush brush (color);
    QFont font;
    font.setFamily("Helvetica");
    pen.setWidth(3);

    int depth = getTreeHeight(root);

    int wDeep = static_cast<int>(pow(2, depth)+2);
    int hDelta = 70;
    int wDelta = 15;
    font.setPointSize(wDelta);
    int width = (wDelta*wDeep)/2;
    treePainter(scene, root, width/2, hDelta, wDelta, hDelta, pen, brush,
font, wDeep);
    return scene;
}

void BinaryRandomTreap::treePainter(QGraphicsScene *&scene, Node* node,
int w, int h, int wDelta, int hDelta, QPen &pen, QBrush &brush, QFont
&font, int depth)
{
    if (node == nullptr)
        return;
    QString out;
    out += node->value;
    out += ";";
    out += QString::number(node->priority);

```

```

    QGraphicsTextItem *textItem = new QGraphicsTextItem;
    textItem->setPos(w-5, h+10);
    textItem->setScale(0.5);
    textItem->setPlainText(out);
    textItem->setFont(font);
    scene->addEllipse(w-wDelta/2, h, wDelta*5/2, wDelta*5/2, pen, brush);
    if (node->left != nullptr)
        scene->addLine(w+wDelta/2, h+wDelta, w-(depth/2)*wDelta+wDelta/2,
h+hDelta+wDelta, pen);
    if (node->right != nullptr)
        scene->addLine(w+wDelta/2, h+wDelta, w+(depth/2)*wDelta+wDelta/2,
h+hDelta+wDelta, pen);
    scene->addItem(textItem);
    treePainter(scene, node->left, w-(depth/2)*wDelta, h+hDelta, wDelta,
hDelta, pen, brush, font, depth/2);
    treePainter(scene, node->right, w+(depth/2)*wDelta, h+hDelta, wDelta,
hDelta, pen, brush, font, depth/2);
    return;
}

```