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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Демонстрация случайного БДП

Студент гр. 9381 _____ Игнашов В.М.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Игнашов В.М.

Группа 9381

Тема работы: Демонстрация случайного БДП

Исходные данные: Требуется реализовать программу в виде оконного изображения, выполняющего функционал задачи, удобный интерфейс для работы со структурой данных с использованием фреймворка Qt на языке C++.
Содержание пояснительной записки:
«Содержание», «Введение», «Задание», «Ход выполнения работы», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 41 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи курсовой работы: 10.12.2020

Дата защиты курсовой работы: 17.12.2020

Студент _____ Игнашов В.М.

Преподаватель _____ Фирсов М.А.

АННОТАЦИЯ

Данная курсовая работа направлена на демонстрацию использования случайных бинарных деревьев поиска с целью использования в обучении для объяснения данной структуры данных и выполняемых с нею действий.

Работа выполнена с применением графического интерфейса для более понятного ознакомления в интегрируемой среде разработки QtCreator.

Реализована возможность построения дерева с последующими действиями в виде добавления, удаления выбранных элементов.

SUMMARY

This coursework is aimed at demonstrating the use of random binary search trees for use in training to explain this data structure and the actions performed with it.

The work was done using a graphical interface for a clearer familiarization in the QtCreator integrated development environment.

Implemented the ability to build a tree with subsequent actions in the form of adding, deleting selected elements.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы – создать графическое приложение с понятным интерфейсом, способное помочь в изучении работы структуры данных, именуемой случайное бинарное дерево поиска.

Необходимо реализовать возможность добавления, удаления элементов и обработки дерева.

Также, необходимо учесть возможные ошибки и исключить варианты падения программы, на каждую проблемную ситуацию выдавать предупреждение о неверном использовании ПО.

1. ЗАДАНИЕ

Задания по **курсовой работе** есть 3 типов:

- "Демонстрация" - визуализация структур данных, алгоритмов, действий. Демонстрация должна быть подробной и понятной (в том числе сопровождаться пояснениями), чтобы программу можно было использовать в обучении для объяснения используемой структуры данных и выполняемых с нею действий.

- "Текущий контроль" - создание программы для генерации заданий с ответами к ним для проведения текущего контроля среди студентов. Задания и ответы должны выводиться в файл в удобной форме: тексты заданий должны быть готовы для передачи студентам, проходящим ТК; все задания должны касаться конкретных экземпляров структуры данных (т.е. не должны быть вопросами по теории); ответы должны позволять удобную проверку правильности выполнения заданий.

- "Исследование" - генерация входных данных, использование их для измерения количественных характеристик структур данных, алгоритмов, действий, сравнение экспериментальных результатов с теоретическими.

Вариант 7

7	Случайные БДП - вставка и исключение. **Демонстрация** с использованием графики

2. СТРУКТУРА ПРОГРАММЫ И ФАЙЛЫ

Для разработки программы была придумана и реализована следующая структура файлов и их объектов:

- ProgWindow – класс, связывающий все другие объекты и демонстрирущий ее интерфейс
- LineOfAmounts – также класс отображения, представляет собой набор изменяемых строк для заполнения его элементами массива элементов
- VisualizeTree – класс, подразумевающий собой сцену, в которой отрисовывается построенное ранее дерево, отображаемое в дальнейшем в ProgWindow
- Node – класс, непосредственно работающий с самой структурой данных, является элементом БДП с надлежащими ему дочерними элементами.
- pt – структура, используемая для работы с отрисовкой дерева в VisualizeTree

Для классов ProgWindow и LineOfAmounts были созданы файлы:

- progwindow.h – заголовочный файл, в котором объявляются классы LineOfAmounts и ProgWindow, а также подключаются необходимые для отображения всего окна заголовочные файлы среды разработки QCreator, а также заголовочный файл visualizetree.h, описанного далее.
- progwindow.cpp – файл, описывающий классы ProgWindow, LineOfAmouns, основные их методы.

Для класса VisualizeTree и структуры pt:

- visualizetree.h – заголовочный файл, в котором объявляется класс VisualizeTree, а также заголовочные файлы среды разработки QCreator, необходимые для отрисовки дерева.
- visualizetree.cpp – файл, описывающий класс VisualizeTree и основные его методы, структуру pt.

Для класса Node:

- node.h – заголовочный файл, в котором объявляется класс Node
- node.cpp – файл, описывающий класс Node и основные его методы обработки БДП.

Также, главный файл – main.cpp – создается и отображается объект класс ProgWindow.

3. ИНТЕРФЕЙС, ОСНОВНОЕ ОКНО ПРОГРАММЫ

Главный элемент интерфейса – объект класса QGraphicsScene, в котором отображается объект класса VisualizeTree, наследованный от QGraphicsView.

Слева от него виджеты, непосредственно каким-либо образом меняющие дерево: сверху – количество элементов исходного массива, меняя которое – меняется количество элементов в окне ниже, преобразовывающегося в последствии в дерево, с кнопкой генерации самого дерева, ниже – поле, содержащее сами элементы, представляет собой объект класса QFrame, содержащего объект класса LineOfAmounts, наследованного от QScrollArea (при большом количестве элементов появляется ползунок, позволяющий добавить элементы с бОльшими номерами) и содержащего изменяемые строки, в которые вписываются элементы.

Ниже от элементов – две строки с двумя кнопками: добавить элемент (слева записывается элемент, который необходимо добавить) и, используемый аналогично, удалить элемент.

Также, ниже вышеописанных элементов добавлены кнопки Clear (очищающая сцену и все дерево) и Instructions (выдающая пользователю QMessageBox с информацией, как использовать программу)

4. ОПИСАНИЕ АЛГОРИТМА

В программе должны быть реализованы 3 алгоритма – создания дерева, добавление нового элемента и удаление элемента.

В первую очередь опишем алгоритм добавления нового элемента, т.к. с помощью него мы будем добавлять в дерево новые элементы при изначальном построении дерева. Для начала рассмотрим вариант, когда первый элемент дерева отсутствует, в таком случае мы указываем, что этот добавляемый элемент – голова дерева. Иначе нам необходимо найти место в дереве, в которое будет добавлен элемент. Происходит это исходя из следующих указаний:

Если элемент, на месте которого мы исследуем, имеет значение больше, чем значение добавляемого элемента, смотрим, если левый потомок отсутствует – устанавливаем добавляемый элемент в качестве левого потомка рассматриваемого сейчас элемента. Если левый потомок имеется - переходим к левому потомку и повторяем алгоритм.

Если же элемент, на месте которого мы исследуем, имеет значение не большее, чем значение добавляемого элемента – выполняем аналогичные действия только для правого потомка и также повторяем алгоритм.

Таким образом по итогу движения по узлам мы приходим к месту, куда должен быть установлен элемент.

Алгоритм создания дерева предельно прост, зная алгоритм добавления элемента, т.к. в таком случае нам достаточно выполнить алгоритм добавления элемента в дерево для всех элементов, заданных заранее пользователем.

Алгоритм удаления элемента имеет несколько различных вариаций. Однако в любом из вариантов первым делом нам необходимо найти сам элемент в дереве. Для этого простым рекурсивным методом движения – сначала исследование левого поддерева, после чего правого, обходим дерево и

наткнувшись на необходим элемент – выходим из функции поиска. Далее, имея необходимый для удаления элемент, выбираем подходящую нам ситуацию:

- Если данный элемент – лист дерева (не имеет потомков), в таком случае – никаких сложных операций – просто отвязываем его от своего предка и удаляем.
- Если данный элемент – узел с одним поддеревом, в таком случае – нам необходимо «перевесить» потомка на предка. Привязав предка элемента к потомку элемента удаляем.
- Если данный элемент – узел с двумя поддеревьями, в таком случае мы не можем просто удалить элемент, так как его поддеревья окажутся разделены. Поэтому, нам необходимо найти элемент, который может заменить наш элемент, при этом он должен быть строго больше чем все элементы левого поддерева и не больше элементов правого поддерева. Таковым элементом будет являться самый левый элемент правого поддерева. Для того, чтобы найти его – движемся у правого дерева постоянно влево, пока можно. Его мы перемещаем на место удаляемого элемента. А с самим этим элементом выполняем действия исходя их первых двух ситуаций. Третьей ситуации быть у него не может, т.к. тогда это не самый левый элемент.

5. ОПИСАНИЕ ФУНКЦИЙ И СТРУКТУР ДАННЫХ

5.1 Описание класса Node

int a;

Поле с содержанием узла/листа конкретного элемента в дереве

int elemNum

Поле, содержащее в себе номер узла/листа в дереве.

Node* left;

Поле с ссылкой на левый элемент узла. В случае отсутствия - nullptr

Node* right;

Поле с ссылкой на правый элемент узла. В случае отсутствия - nullptr

int depth;

Поле, содержащее глубину конкретного узла

Конструктор Node(int, int, int);

Конструктор класса, с передаваемыми в него аргументами значения, глубины и номера конкретного узла, которыми инициализируется новосозданный элемент

Метод addNew(int, int);

Метод, принимающий два целочисленных аргумента – значение и номер, используемый для добавления в дерево нового элемента с соответствующим содержанием и номером.

Создав новую временную переменную tmp типа Node*, инициализируем ее ссылкой на голову дерева и передвигаемся в соответствии с правилами БДП (Движемся в левую сторону если новое значение больше рассматриваемого в данный момент, иначе в правую до места, в которое должен быть добавлен элемент) по дереву, после чего выделяем память под новый элемент, создаем его, и выходим из цикла, а соответственно из метода.

Метод delElem(int);

Метод, удаляющий элемент по значению, переданному в качестве аргумента и возвращающий информацию о том, как прошло удаление в виде целочисленной переменной.

Создав новую логическую переменную onLeft, изучаем если справа или слева присутствует необходимый нам элемент – узнаем сторону и в соответствии с этим инициализируем переменную onLeft – true если элемент слева, иначе false. В противном случае, если искомого нами элемента нет ни слева, ни справа продолжаем движение по дереву. Если двигаться далее по дереву не представляется возможным – выходим со значением 0. Стоит отметить, что голова дерева не рассматривается, и в случае, если она – единственный подобный искомый элемент – также выходим со значением 0.

Узнав место нахождения элемента, начинаем работать с ним. Выполняемые действия объяснены для случая, если элемент – слева, т.к. для ситуации, если он справа – действия аналогичны. Возможны 4 варианта событий:

- В случае, если элемент – лист (не имеет потомков) – удаляем его и инициализируем nullptr для дальнейшей работы.
- В случае, если элемент имеет одного потомка – справа – инициализируем данный элемент этим потомком, тем самым удаляя его и «перевешивая» на него этого потомка.
- В случае, если элемент имеет одного потомка – слева – инициализируем его этим потомком, тем самым удаляя его и «перевешивая» на него этого потомка.
- В случае если у данного элемента два потомка – нам необходимо найти наименьший элемент справа, а тогда имеем три возможных варианта:
    - Если справа элемент без потомков – его необходимо «перевесить» на искомый элемент. Присваиваем искомому элементу значение этого элемента и удаляем этот элемент.
    - Если справа элемент с одним потомком справа – его необходимо «перевесить» на искомый элемент, после чего на этот же элемент «перевесить его потомка». Присваиваем

искомому элементу значение этого элемента, а сам его инициализируем потомком.

- Слева у правого элемента есть значения, а значит – он не минимальный, движемся влево с помощью переприсваивания временной переменной tmp левым объектом, приходя к элементу, потомок которого – элемент без левого потомка. Значение этого потомка присваиваем искомому элементу и удаляем в случае отсутствия у него других потомков, иначе «перевешиваем» объясненным ранее методом на него потомка.

В каждом из вариантов по завершении изменения – выходим со значением 1, что означает, что дерево было изменено.

Метод reNumAll();

Метод, обновляющий нумерацию у узлов дерева. Принимает в качестве аргумента ссылку на значение, которое необходимо присвоить номеру узла. Соответственно присваиваем номеру элемента это значение и переходим к следующим узлам, заранее увеличив значение по этой ссылке на 1. Таким образом обходим все дерево, меняя номера у элементов.

Метод reDepthAll();

Метод, обновляющий глубину у узлов дерева. Действуем аналогично 2.9, за исключением того, что меняем значение не в ссылке, чтоб не иметь разные значения, а увеличиваем по мере вхождения в нового потомка.

Метод int maxdepth();

Метод, находящий наибольшую глубину в дереве.

Обходя дерево уже описанным ранее методом, считаем максимальную глубину в левом и правом поддереве (в конечных элементах дерева возвращаем глубину, в иных сравниваем глубину левого и правого). Тем самым в конце рекурсии для главного элемента будем знать наибольшую глубину в дереве.

Метод void deleteTree();

Метод используется для освобождения памяти, выделенной под дерево, обходя дерево описанным ранее методом, с конечных элементов удаляем узлы, очищая память и добравшись до главного элемента, также его удаляем.

Методы-геттеры

Методы Node* getLeft(), Node* getRight(), int getA(), int getElemNum(), int getDepth(), описанные в объявлении класса предназначены для получения значения элемента в приватных полях узла. Их реализация предельно понятна – возвращаем значение в соответствующем поле узла.

5.2 Описание класса VisualizeTree

int maxn;

Поле, содержащую информацию о максимальном количестве элементов, возможных для обработки.

QGraphicsSimpleTextItem items;**

Поле – массив элементов-наследников от QGraphicsItem’ов, подразумевающее собой объекты, отрисовывающие значения в узлах дерева в качестве простого текста.

QGraphicsLineItem lineItems;**

Поле – массив элементов-наследников от QGraphicsItem’ов, подразумевающее собой объекты, отрисовывающие линии между узлами дерева.

Конструктор VisualizeTree(int);

Конструктор класса, которому в качестве аргумента передается максимальное количество элементов.

Соответствующее поле инициализируется и выделяется память под массивы items и lineItems в соответствии с максимальным количеством элементов. После чего в цикле под сами элементы в массиве выделяется память и они добавляются в эту сцену (класс наследуется от QGraphicsScene), выключая свое отображение (оно будет включаться по мере отрисовки дерева).

Метод update(Node*, int);

Данный метод класса предназначен для обновления данной сцены деревом. Поэтому в качестве аргументов передается главный узел дерева со всеми поддеревьями (дерево) и количество элементов в дереве.

Выключаем отображение всех item’ов в сцене с целью включать их по мере добавления новых элементов дерева. В цикле проходимся по всем номерам элементов дерева, отрисовывая их, используя функцию нахождения информации об узле findPlace(Node*, int, int, int, int, bool) (Для этого ранее была объявлена структура pt, содержащая для каждого элемента в себе значение нынешних координат, координат предыдущего узла(для отрисовки линии),

значение в данном узле и логическую переменную, отвечающую за то, была ли сформирована структура):

findPlace(Node*, int, int, int, int, bool):

В качестве аргументов в нее передается дерево для поиска места элемента, номер искомого элемента, информация о том, насколько надо сдвинуться влево и вправо для получения координаты по X, также, значение изменяемое в геометрической прогрессии с шагом ½, чтобы построить более точное изображение дерева и информация, с какой стороны пришел элемент.

В самой функции создается возвращаемый элемент retpt, который всю информацию будет совмещать. Рекурсивно движемся по дереву, пока не уткнемся в элемент, номер которого будет совпадать с искомым. Тогда инициализируем все необходимые поля. И возвращаемся.

Тем самым, по итогу мы получаем координаты, по которым надо отрисовать линии и значение, которое соответствует элементу. Отображаем на соответствующем элементе в массиве текстовых item’ов значение узла и между соответствующими координатами рисуем линии (кроме нулевого элемента, потому что у него нет предыдущих узлов) (Умножаем все координаты на постоянную, для увеличения масштаба дерева). Включаем отображение у них.

Деструктор ~VisualizeTree();

Проходя в цикле по массивам, очищаем их элементы, после чего очищаем память, выделенную под сами массивы.

5.3 Описание класса LineOfAmounts

const int maxn=50;

int n=0;

Поле, содержащее информацию о настоящем количестве элементов в дереве. Изначально инициализируется нулем.

QFrame* fr;

Поле для графического отображения, содержащее все строки из arr.

QVBoxLayout* l;

Поле компоновки всех строк из arr, для последующего задания в fr

QLineEdit* arr[maxn];

Массив строк, содержащих значения элементов дерева, вводимых пользователем.

Конструктор LineOfAmounts();

Конструктор класса, выделяющий память под все поля класса и задающий расположение всех объектов в графическом отображении.

Выделяется память под fr и l, после чего в цикле выделяется память под объекты строк, задаем их невидимыми, с фиксированной высотой и добавляем их в компоновку l. После чего задаем объект данного класса, наследуемого от QScrollArea, изменяемым в размерах. Компоновке задаем выравнивание вверх и задаем fr эту компоновку. Помещаем fr в объект этого класса.

Метод void resize(int k);

Данный метод отображает строки, которые должны быть отображены.

Становятся видимыми пользователю новые. Полю n присваивается переданное значение – новое количество элементов.

Деструктор ~LineOfAmounts();

В цикле очищается память выделенная под объекты в массиве линий.

Сеттер void setN(int x);

Метод-сеттер для установления значения приватного поля n из объектов других классов.

5.4 Описание класса ProgWindow

const int maxn=50;

Node* head=nullptr;

Поле, содержащее ссылку на главный элемент дерева

bool created=false;

Поле, содержащее информацию о том, создано ли дерево.

int numOfLei=0;

Поле, содержащее количество элементов в данный момент в виде числа

int amounts[maxn];

Поле массива, содержащего сами элементы

QLineEdit* numOfLe;

Поле, содержащее виджет количества элементов в данный момент в виде изменяемой строки.

LineOfAmounts* loa = nullptr;

Поле, содержащее экземпляр класса LineOfAmounts, описанного выше

QPushButton* calc;

Поле виджета кнокпи, при нажатии на которую будет происходить вычисление дерева и построение в treeScene, отображение в view

QLineEdit* whichElemAdd;

Поле виджета строки, в которую записывается элемент, который необходимо добавить в дерево.

QPushButton* findElemAddButton;

Поле виджета кнопки, при нажатии на которую элемент, содержащийся в whichElemAdd будет добавлен в дерево.

QLineEdit* whichElemDel;

Поле виджета строки, в которую записывается элемент, который необходимо удалить из дерева.

QPushButton* findElemDelButton;

Поле виджета кнопки, при нажатии на которую элемент, содержащийся в whichElemDel будет удален из дерева.

QPushButton* instructions;

Поле виджета кнопки, при нажатии на которую будет выдаваться QMessageBox с информацией о том, как использовать программу

QPushButton* clear;

Поле виджета кнопки, при нажатии на которую сформированное дерево будет удаляться и возвращать все к положению, когда зашли в программу первый раз.

VisualizeTree* treeScene=nullptr;

Поле, содержащее экземпляр класса VisualizeTree, который будет отображен в view. Изначально инициализирован nullptr.

QGraphicsView* view;

Поле виджета QGraphicsView, для отображения сцены, находящейся в treeScene и последующего отображения в приложении дерева.

Конструктор ProgWindow();

Конструктор, выделяющий память под все поля в объекте класса, задающий отображение виджетов и добавление связи между сигналами виджетов и слотами класса.

Выделив память под все поля и создав элементы, в том числе вызван конструктор и построен экземпляр VisualizeTree treeScene, и занят в качестве сцены для view.

Для удобного пользователю интерфейса были созданы компоновки: горизонтальная для левой верхней части со строкой, изменяемой строкой и кнопкой, две горизонтальные компоновки для нижних двух строк с кнопкой, одна вертикальная – для совмещения вышеописанных компоновок, экземпляра LineOfAmounts, кнопки очистки и инструкций. И еще одна горизонтальная для совмещения компоновки левой части и view. Тем самым собирая конечный интерфейс.

А также, были связаны сигналы и слоты следующим образом:

- Изменение строки количества элементов – addArray()
- Нажатие на кнопку генерации дерева – createTree()

- Нажатие на кнопку добавления элемента – addElem()
- Нажатие на кнопку удаления элемента – delElem()
- Нажатие на кнопку инструкций – inst()
- Нажатие на кнопку очистки сцены и дерева – clear()

Слот void addElem();

Слот, вызываемый нажатием на кнопку добавления элемента.

В случае отсутствия чего-либо в строке с элементом, который необходимо добавить – просто выходим из функции.

В случае, если отсутствует дерево (treeScene==nullptr) выдаем ошибку отсутствия дерева (для этого было создано перечисление возможных ошибок typeOfError и функция с аргументом в виде ошибки, вызывающая предупреждающую ошибку для соответствующего аргумента с помощью стандартного статического метода QMessageBox'a warning(QWidget*, const QString&, const QString&)); если в этой строке 0 – инициализируем ранее объявленную переменную е нулем, иначе пытаемся преобразовать ввод в строке в int, если не удается выводим ошибку неверного ввода. Иначе, добавляем элемент в дерево, методом экземпляра Node* head addnew(int, int), передавая в нее сам элемент(е) и количество и номер элемента – последним.

Увеличиваем количество элементов на 1 и обновляем сцену путем вызова метода treeScene update(Node*, int), передавая в нее главный элемент дерева и количество элементов в дереве.

Перед каждым возвратом из функции очищаем поле добавляемого элемента.

Слот void addArray();

Слот, вызываемый изменением значения в строке количества элементов.

Очищаем дерево и отображение дерева на treeScene, тк начинается генерация нового дерева.

Выключаем видимость всех строк, чтобы не отображать строки, которые нам не нужны. При отсутствии количества строк – очищаем все строки(изменяем их текст на "").

Если возможно число в этой строке преобразовать в int – вызываем метод resize у loa, тем самым отображая необходимые нам строки. Иначе выводим ошибку неверного ввода и очищаем эту строку. В случае, если преобразовать можно, но получившееся число больше максимального числа – выводим ошибку слишком большого количества элементов.

Слот void createTree();

Слот, вызываемый нажатием на кнопку создания дерева(Generate)

Если дерево было создано – вызываем ошибку, что сначала необходимо очистить сцену и дерево, для создания нового.

Сразу проверяем на то, пустая ли строка, или равна ли она нулю – если так, выводим ошибку отсутствия ввода. Очищаем дерево, для того, чтобы выводились ошибки отсутствия дерева при добавлении нового узла или удаления.

Начинаем вводить в массив самих элементов значения из строк элементов. Если вдруг один из элементов не может быть преобразован в int – отображаем ошибку ввода, очищаем дерево, а также отображение.

Очищаем предыдущее дерево и добавляем новое, а также обновляем картинку. Устанавливаем, что дерево было создано.

Слот void delElem();

Слот, вызываемый нажатием на кнопку удаления элемента.

Проверяем на отсутствие удаляемого элемента. Если так – возвращаемся.

Проверяем на наличие дерева. Если отсутствует – выводим соответствующую ошибку. Если введен 0 в качестве удаляемого элемента – инициализируем ранее объявленную переменную е нулем, иначе пытаемся преобразовать в int значение, лежащее в строке, в которой должен лежать удаляемый элемент. Если не получается – выводим ошибку неверного ввода.

Пытаемся удалить элемент путем вызова метода удаления у главного узла дерева. Если возвращаемое значение ноль – выводим ошибку, что такого элемента нет, либо пользователь пытается удалить главный элемент дерева. Возвращаемся из слота.

Перерасчитываем номера элементов и их глубину путем вызова соответствующих методов у главного узла. Вычитаем из количества элементов 1, т.к. получилось удалить и отображаем снова дерево на view с удаленным элементом, вызывая метод обновления у treeScene.

Перед каждым возвратом из функции очищаем поле удаляемого элемента.

Слот void inst();

Слот, вызываемый нажатием на кнопку инструкций.

Создаем объект QMessageBox, предварительно выделив под него память. Устанавливаем название окна и отображаемый текст (инструкцию). Вызываем срабатывание этого QMessageBox’а

Слот void startNew();

Слот, вызываемый нажатием на кнопку очищения сцены и дерева.

Проверяем, было ли построено дерево. Если нет – выходим. Иначе устанавливаем, что дерево не было сформировано, количество элементов делаем равным нулю. Также устанавливаем ноль введенных элементов. Удаляем голову и выключаем все элементы в отображении.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной курсовой работы было создано приложение с графическим интерфейсом для обработки, создания, изучения и действий со структурой данных «случайное бинарное дерево поиска».

Помимо полностью корректной работы программы, была создана графическая оболочка с интуитивно понятным интерфейсом. Учтены все возможные «вылеты» программы и ошибки на протяжении всей ее работы и преобразованы пользователю в виде диалоговых окон ошибок.

Были изучены методы работы с интегрированной средой разработки QtCreator, а также работы со структурами данных бинарных деревьев поиска.

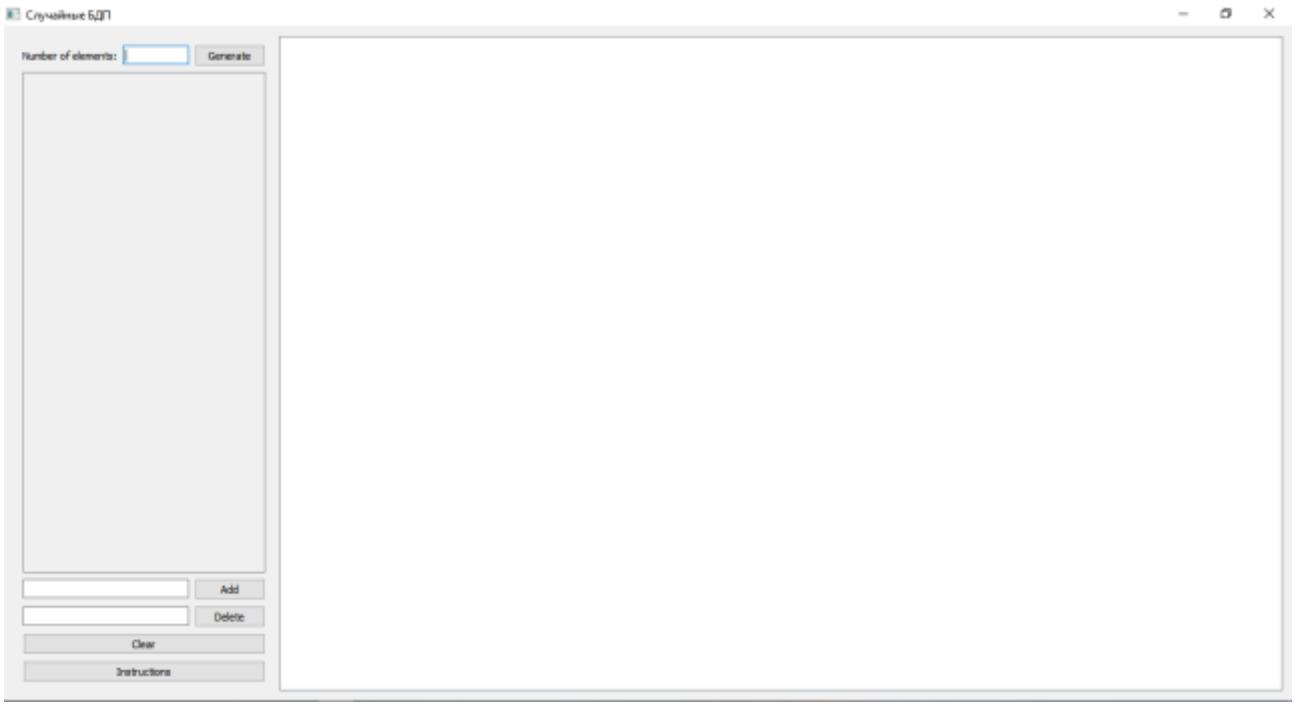
Программа неоднократно протестирована, результат работы программы соответствует заданным условиям. Поставленные задачи решены

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

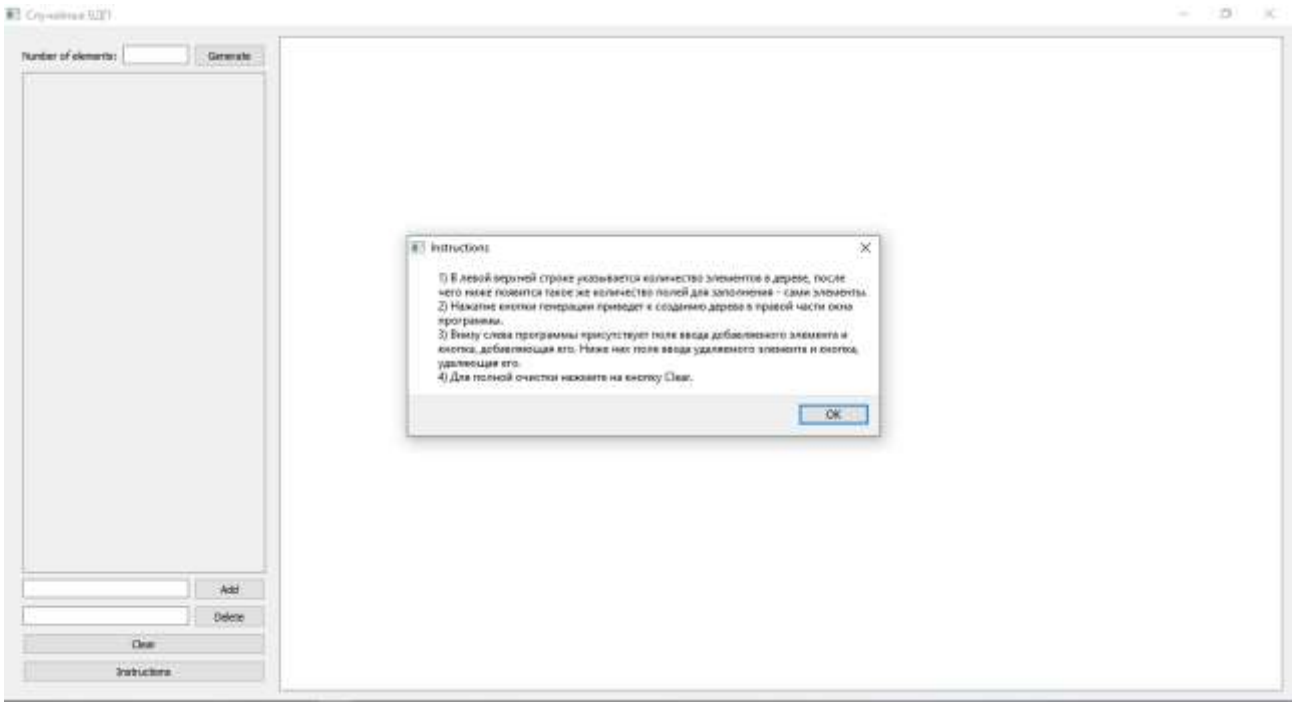
1. Qt Documentation // Qt. URL: https://doc.qt.io/qt-5.15/
2. Перевод и дополнение документации QT // CrossPlatform.RU URL: http://doc.crossplatform.ru/
3. Русскоязычный форум Хабр // Habr URL: https://habr.com/ru/
4. Collection of knowledge and practical advices about programming and information technologies. // EVILEG.COM URL: https://evileg.com/ru/

ПРИЛОЖЕНИЕ А
ТЕСТИРОВАНИЕ

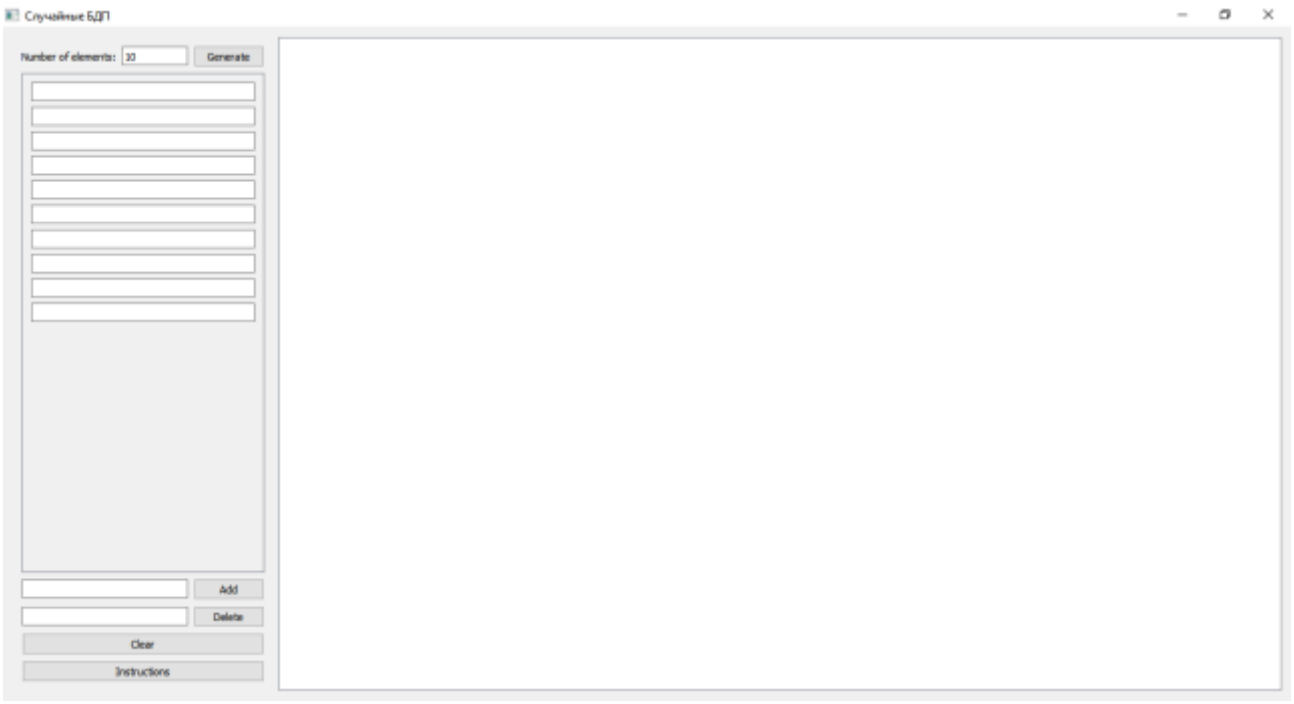
1. Вид программы после запуска



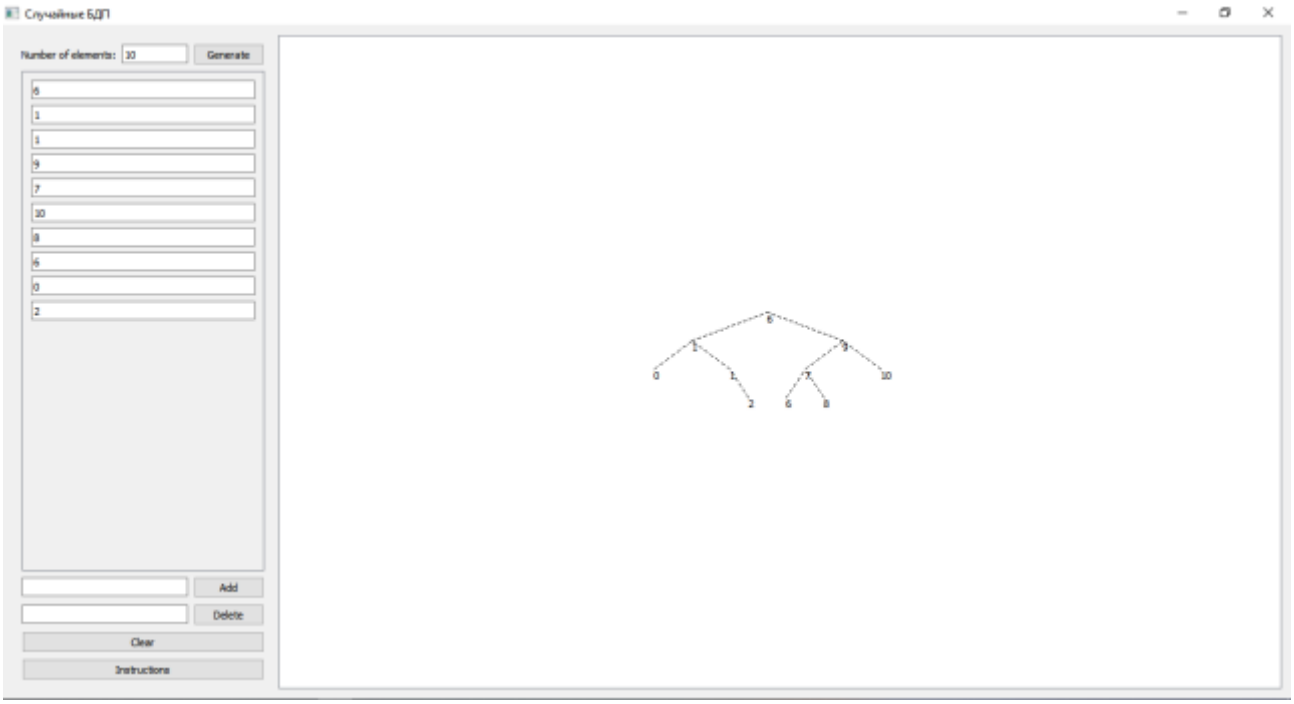
2. Просмотрим инструкции по использованию программы (кнопка Instructions).



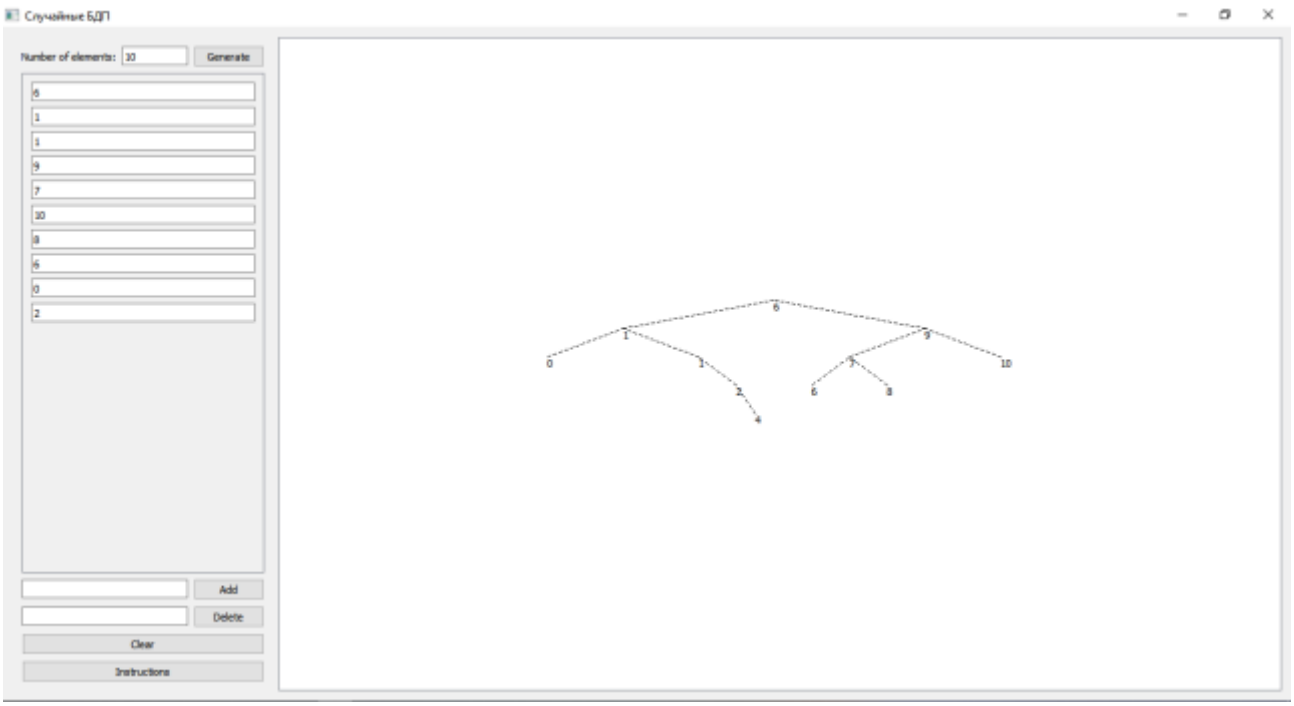
3. Введение количества элементов вызвало срабатывание слота addArray()

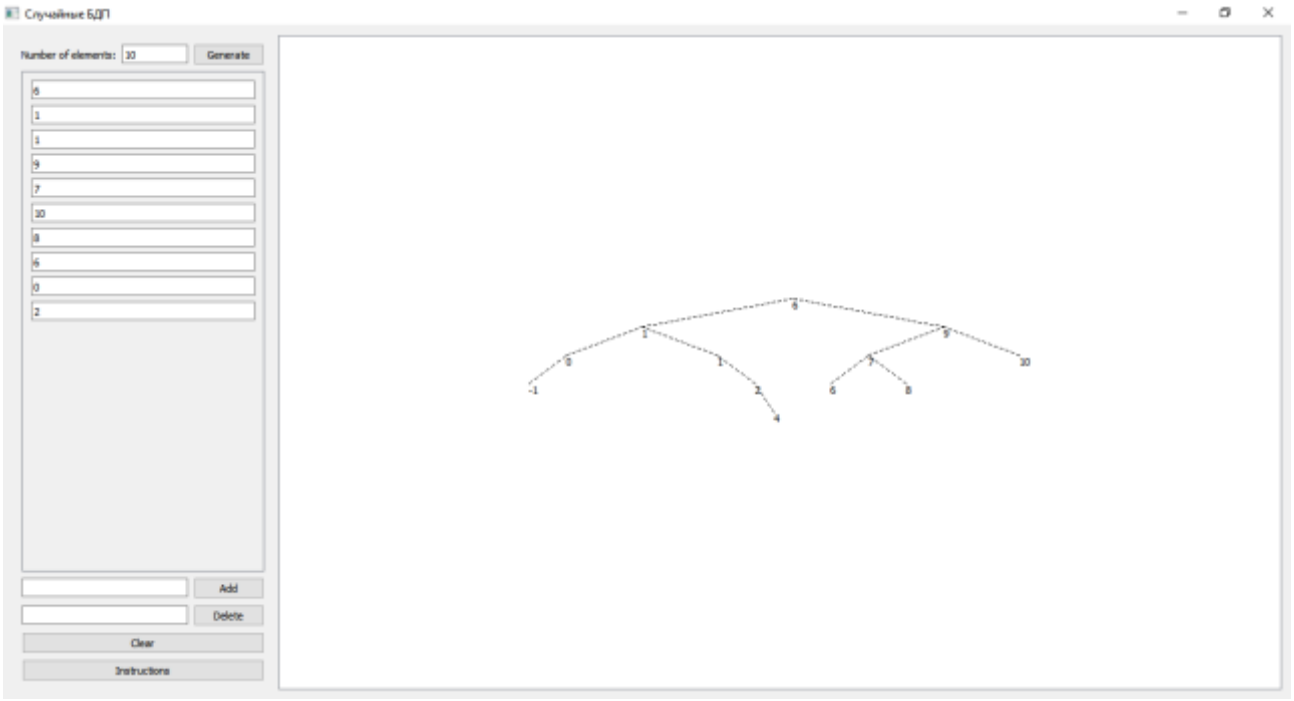


4. Ввели элементы и нажали на кнопку «Generate», тем самым построив дерево

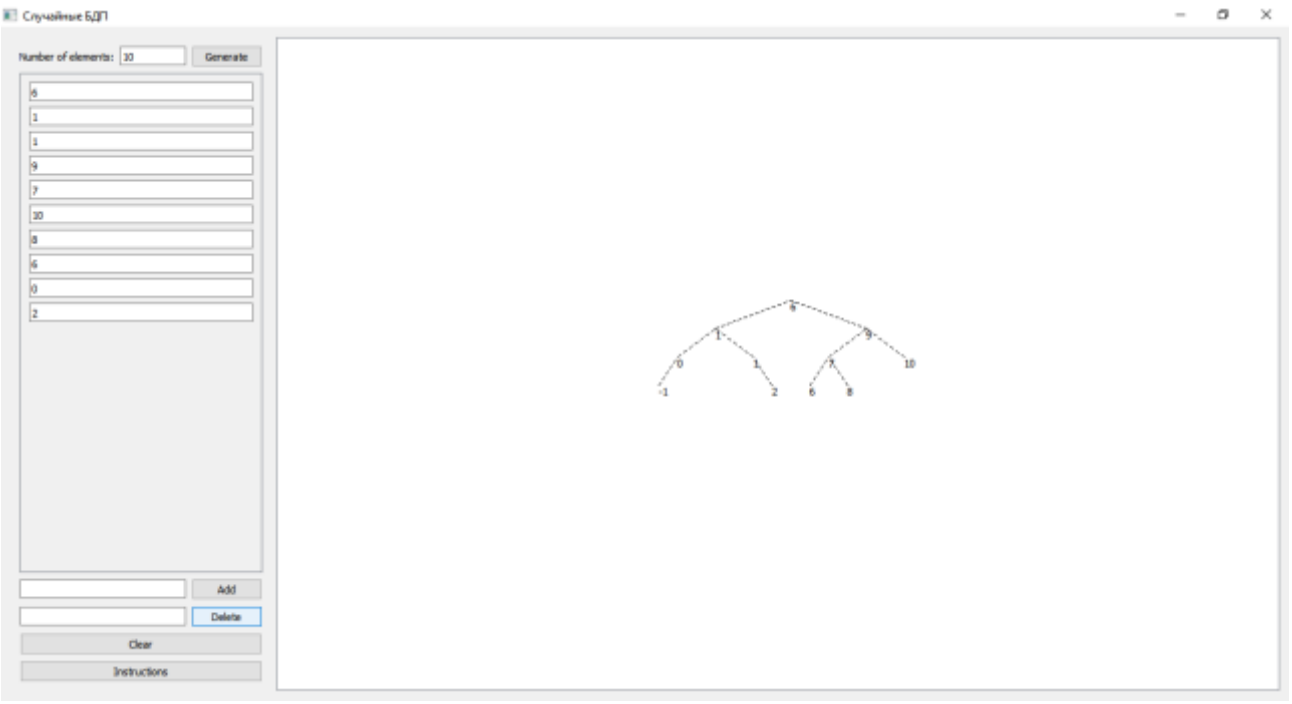


5. Добавим элемент в сформированное дерево введением элемента в строку добавления элемента и нажатием кнопки добавления

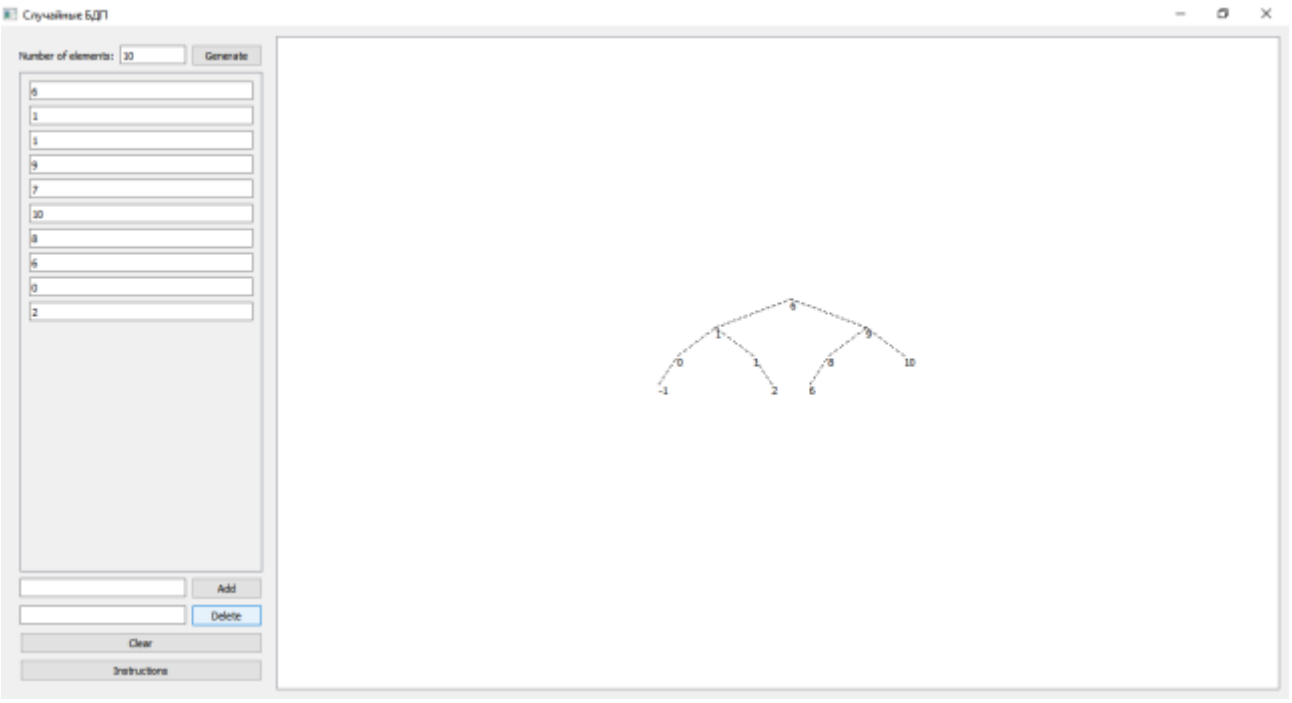




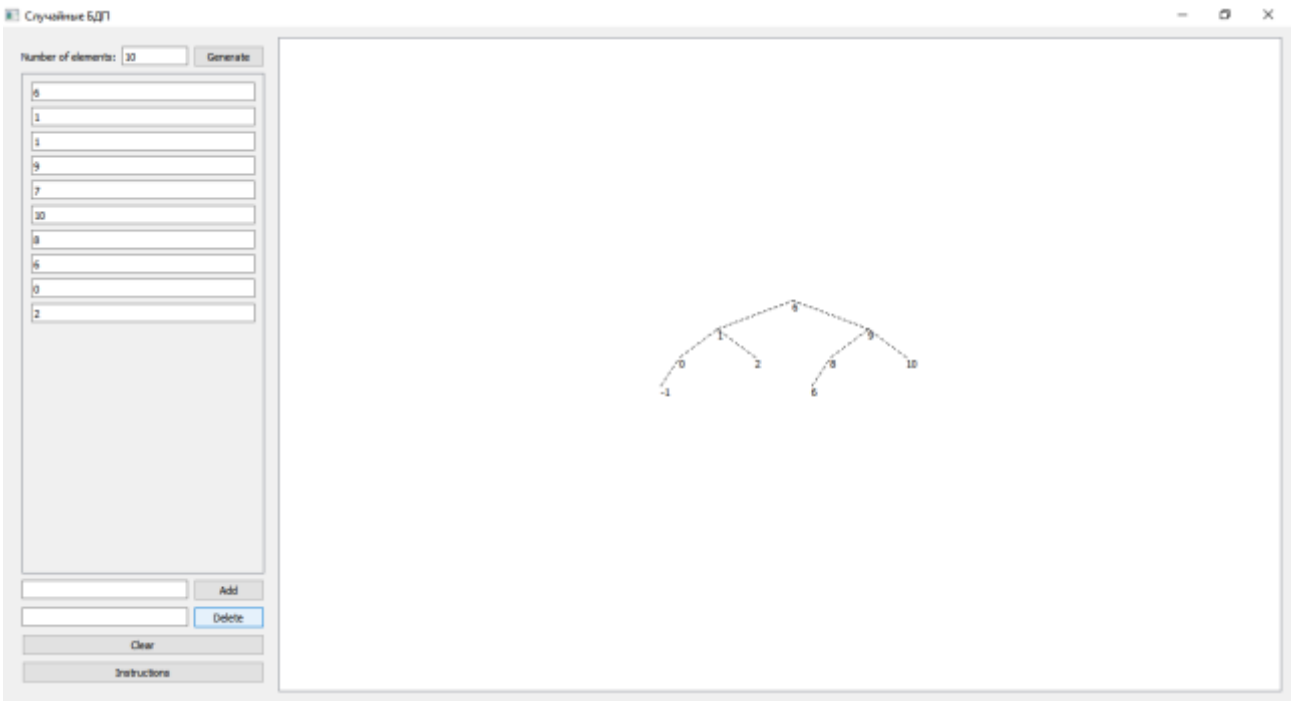
6. Поудаляем элементы с различными случаями, чтобы убедиться, что удаление работает верно.



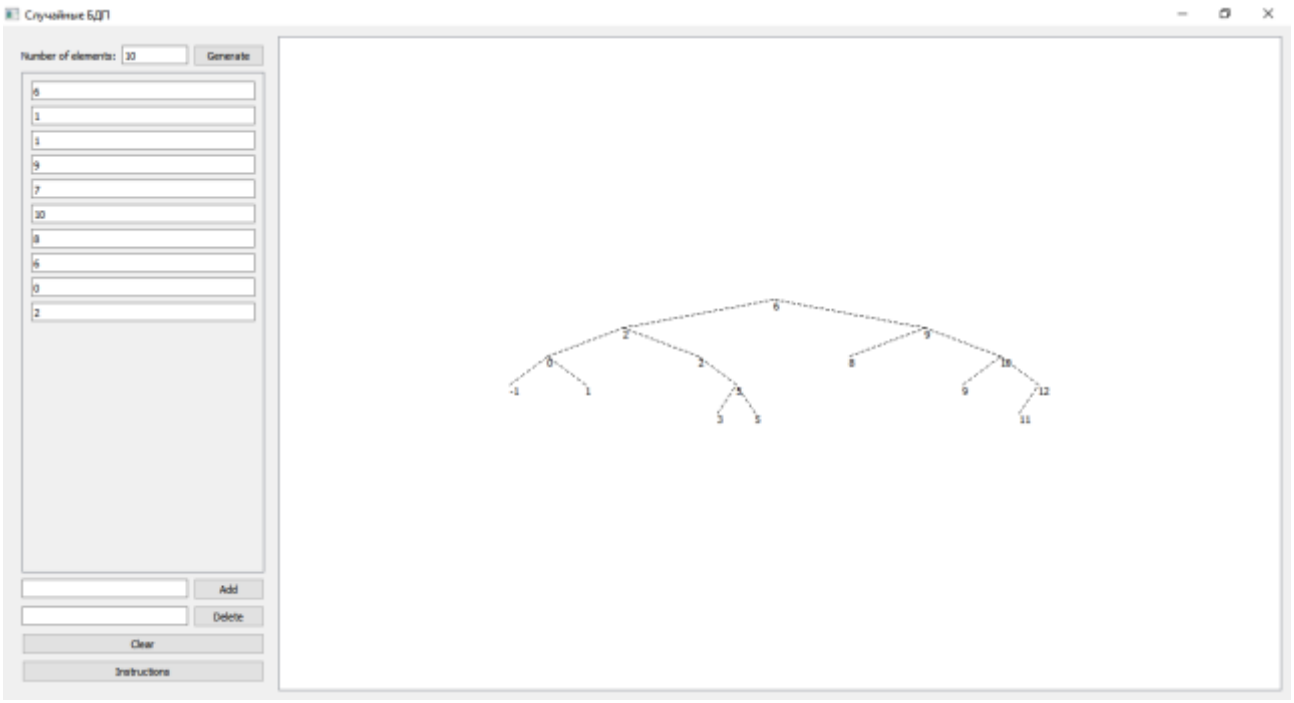
7. Одиноко весящий лист удаляется. Проверим с узлом с двумя потомками.



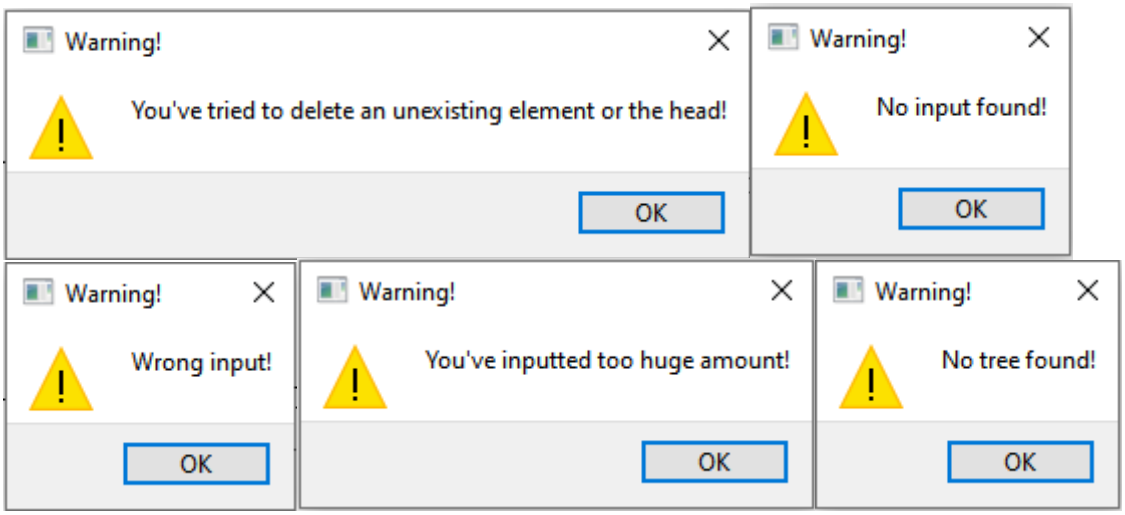
8. Все работает, попробуем удалить элемент с одним потомком справа, у которого свои потомки



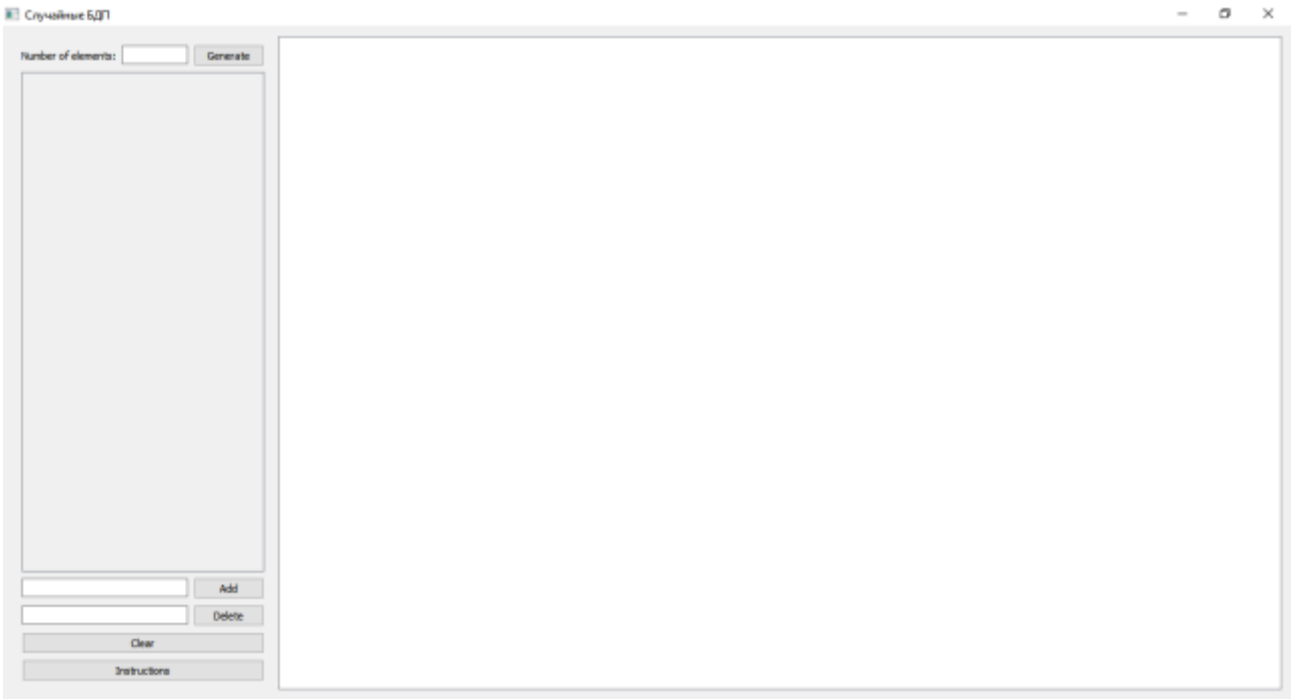
9. Все верно. Пример неоднократного добавления и удаления элементов:



10. Убедившись в работоспособности программы, попробуем протестировать на некорректных данных. Будем натыкаться на следующие ошибки:



11. Нажмем на кнопку Clear для очистки сцены и введения новых значений



12. Введем новое дерево


Случайные БДП
Number of elements: 4
Generate
2
1
4
3
Add
Delete
Clear
Instructions
2
1
4
3

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл CurseAiSD.pro

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

CONFIG += c++11 +=static +=release += staticlib

# You can make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the
APIs deprecated before Qt 6.0.0

SOURCES += \
    main.cpp \
    node.cpp \
    progwindow.cpp \
    visualizetree.cpp

HEADERS += \
    node.h \
    progwindow.h \
    visualizetree.h

# Default rules for deployment.
qnx: target.path = /tmp/${TARGET}/bin
else: unix:!android: target.path = /opt/${TARGET}/bin
!isEmpty(target.path): INSTALLS += target
```

Файл node.h

```
#ifndef NODE_H
#define NODE_H


class Node
{
private:
    int a;//Содержание
    int elemNum;//Номер элемента
    Node* left;//Ссылка на левый элемент
    Node* right;//Ссылка на правый элемент
    int depth;//Глубина вхождения в дерево
public:
    Node(int am, int depth, int elemNum);//Конструктор
    void addnew(int x, int num);//Добавление элемента
    int delElem(int x);//Удаление элемента
    void reNumAll(int* n);//Пересчет номеров элементов
    void reDepthAll(int n);//Пересчет глубин элементов
    int maxdepth();//Поиск максимаьлной глубины
    void deleteTree();//Удалить дерево

    Node* getLeft(){return left;}
    Node* getRight(){return right;}
    int getA(){return a;}
    int getElemNum(){return elemNum;}
```

```
    int getDepth(){return depth;}
};

#endif // NODE_H
```

Файл node.cpp

```
#include "node.h"

Node::Node(int am, int depth, int elNum)
{
    elemNum=elNum;
    a=am;
    this->depth=depth;
    left=nullptr;
    right=nullptr;
}


void Node::addnew(int x, int num)
{
    //Последовательный обход в соотвествии с поиском по бдп и добавление
в место, где элемент ддолжен быть
    Node* tmp = this;
    while(true){
        if(tmp->a>x){
            if(tmp->left==nullptr){
                tmp->left=new Node(x,tmp->depth+1,num);
                break;
            }
            tmp=tmp->left;
        }
        else{
            if(tmp->right==nullptr){
                tmp->right=new Node(x,tmp->depth+1,num);
                break;
            }
            tmp=tmp->right;
        }
    }
}

int Node::delElem(int x){
    bool onLeft;
    if((left!=nullptr&&left->a==x)||(right!=nullptr&&right->a==x)){//С
какой стороны элемент
        if(left!=nullptr&&left->a==x)
            onLeft=true;
        else
            onLeft=false;
    }else{//Если не найден здесь - ищем дальше, или возвращаемся если во
всем дереве нет этого элемента(не считая головы)
        if(left!=nullptr&&left->delElem(x)==1)
            return 1;
        if(right!=nullptr&&right->delElem(x)==1)
            return 1;
        return 0;
    }
    if(onLeft==true){//Если слева
```

```
        if(left->left==nullptr&&left->right==nullptr){//Если это лист
            delete left;
            left=nullptr;
            return 1;
        }
        if(left->left==nullptr){//Если только справа
            left=left->right;
            return 1;
        }
        if(left->right==nullptr){//Если только справа
            left=left->left;
            return 1;
        }
        Node* tmp=left->right;//Если два элемента есть - ищем минимальный
справа
        if(tmp->left==nullptr&&tmp->right==nullptr){//Если сам такой
элемент
            left->a=tmp->a;
            left->right->deleteTree();
            left->right=nullptr;
            return 1;
        }
        if(tmp->left==nullptr&&tmp->right!=nullptr){//Если сам такой
элемент и справа висит что-то
            left->a=left->right->a;
            left->right=left->right->right;
            return 1;
        }
        while(tmp->left->left!=nullptr){//Переходим к минимальному
            tmp=tmp->left;
        }
        left->a=tmp->left->a;
        if(tmp->left->right!=nullptr){//Если у минимального ничего не
висит
            tmp->left=tmp->left->right;
        }else{
            tmp->left->deleteTree();//Если у минимального висит справа
            tmp->left=nullptr;
        }
        return 1;

    }else{//Все аналогично вышеописанному, но если искомый элемент справа
        if(right->left==nullptr&&right->right==nullptr){
            delete right;
            right=nullptr;
            return 1;
        }
        if(right->left==nullptr){
            right=right->right;
            return 1;
        }
        if(right->right==nullptr){
            right=right->left;
            return 1;
        }
        Node* tmp=right->right;
        if(tmp->left==nullptr&&tmp->right==nullptr){
```

```
            right->a=tmp->a;
            right->right->deleteTree();
            right->right=nullptr;
            return 1;
        }
        if(tmp->left==nullptr&&tmp->right!=nullptr){
            right->a=right->right->a;
            right->right=right->right->right;
            return 1;
        }
        while(tmp->left->left!=nullptr){
            tmp=tmp->left;
        }
        right->a=tmp->left->a;
        if(tmp->left->right!=nullptr){
            tmp->left=tmp->left->right;
        }else{
            tmp->left->deleteTree();
            tmp->left=nullptr;
        }
        return 1;
    }
    return 0;//Если ничего не подходит - удалять точно нечего
}

void Node::reNumAll(int* n)
{
    this->elemNum=*n;
    *n+=1;
    if(left!=nullptr){
        left->reNumAll(n);
    }
    if(right!=nullptr){
        right->reNumAll(n);
    }
}

void Node::reDepthAll(int n)
{
    this->depth=n;
    if(left!=nullptr){
        left->reDepthAll(n+1);
    }
    if(right!=nullptr){
        right->reDepthAll(n+1);
    }
}

int Node::maxdepth()
{
    //Последовательный обход и сравнение максимальной глубины слева и
справа
    int maxdepth_left=0;
    int maxdepth_right=0;
    if(left!=nullptr)
        maxdepth_left=left->maxdepth();
    if(right!=nullptr)
```

```
        maxdepth_right=right->maxdepth();
    if(left==nullptr&&right==nullptr)
        return depth;
    else
        return maxdepth_left>maxdepth_right ? maxdepth_left :
maxdepth_right;
}

void Node::deleteTree()
{
    //Последовательный обход и удаление
    if(left!=nullptr){
        left->deleteTree();
        left=nullptr;
    }
    if(right!=nullptr){
        right->deleteTree();
        right=nullptr;
    }
    delete this;
}
```

Файл progwindow.h

```
#ifndef PROGWINDOW_H
#define PROGWINDOW_H

#include <QWidget>
#include <QPushButton>
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QLineEdit>
#include <QLabel>
#include <QMessageBox>
#include <QDesktopWidget>
#include "visualizetree.h"

const int maxn=50;

class LineOfAmounts: public QScrollArea{
private:
    int n=0;
public:
    LineOfAmounts();
    QFrame* fr;

    QVBoxLayout* l;
    QLineEdit* arr[maxn];
    void resize(int k);
    ~LineOfAmounts();

    void setN(int x){n=x;}
};

class ProgWindow: public QWidget
{
    Q_OBJECT
private:
    Node* head=nullptr;//Главный элемент дерева
```

```
    bool created=false;
    int numOfLEi=0;//Количество строк элементов
    int amounts[maxn];//массив элементов
public:
    ProgWindow();
    //Для графического отображения
    QLineEdit* numOfLE;
    LineOfAmounts* loa = nullptr;
    QPushButton* calc;

    QLineEdit* whichElemAdd;
    QPushButton* findElemAddButton;

    QLineEdit* whichElemDel;
    QPushButton* findElemDelButton;

    QPushButton* instructions;
    QPushButton* clear;

    VisualizeTree* treeScene=nullptr;
    QGraphicsView* view;

public slots:
    void addElem();//Слот поиска элемента
    void addArray();//Слот добавления массива элементов
    void createTree();//Слот создания дерева
    void delElem();//Слот удаления элемента
    void inst();//Слот инструкций
    void startNew();//Слот инструкций
};

#endif // PROGWINDOW_H
```

Файл progwindow.cpp

```
#include "progwindow.h"

enum typeOfError{//Перечисление с типами ошибки
    WRONGINPUT,
    NOTREE,
    NOINPUT,
    BIGAMOUNT,
    WRONGDELETE,
    TREEISCREATED
};

void error(typeOfError type){//Обработка ошибок
    switch(type){
        case WRONGINPUT:
            QMessageBox::warning(nullptr, "Warning!","Wrong input!");
            break;
        case NOTREE:
            QMessageBox::warning(nullptr, "Warning!","No tree found!");
            break;
        case NOINPUT:
            QMessageBox::warning(nullptr, "Warning!","No input found!");
            break;
        case BIGAMOUNT:
```

```
            QMessageBox::warning(nullptr, "Warning!","You've inputted too
huge amount!");
            break;
        case WRONGDELETE:
            QMessageBox::warning(nullptr, "Warning!","You've tried to
delete an unexisting element or the head!");
            break;
        case TREEISCREATED:
            QMessageBox::warning(nullptr, "Warning!","Tree is already
created, so use the Clear button!");
            break;
    }
}

ProgWindow::ProgWindow()
{
    //Выделение памяти под объекты
    numOfLE=new QLineEdit();
    loa=new LineOfAmounts;
    whichElemAdd=new QLineEdit;
    findElemAddButton=new QPushButton("Add");
    whichElemDel=new QLineEdit;
    findElemDelButton=new QPushButton("Delete");
    instructions=new QPushButton("Instructions");
    clear=new QPushButton("Clear");
    calc=new QPushButton("Generate");
    treeScene = new VisualizeTree(maxn);
    view=new QGraphicsView;
    view->setScene(treeScene);//Установка сцены с деревом в view
    //Компоновка графического отображения
    QHBoxLayout* layh = new QHBoxLayout;
    QLabel *NumberOf = new QLabel("Number of elements:");

    layh->addWidget(NumberOf);
    layh->addWidget(numOfLE);
    layh->addWidget(calc);

    QHBoxLayout* layh2=new QHBoxLayout;
    layh2->addWidget(whichElemAdd);
    layh2->addWidget(findElemAddButton);

    QHBoxLayout* layh3=new QHBoxLayout;
    layh3->addWidget(whichElemDel);
    layh3->addWidget(findElemDelButton);

    QVBoxLayout* layv=new QVBoxLayout;
    layv->addLayout(layh);
    layv->addWidget(loa);
    layv->addLayout(layh2);
    layv->addLayout(layh3);
    layv->addWidget(clear);
    layv->addWidget(instructions);

    QWidget* leftThing=new QWidget;
    leftThing->setLayout(layv);
    leftThing->setFixedWidth(275);
```

```
    QHBoxLayout* layh4=new QHBoxLayout;
    layh4->addWidget(leftThing);
    layh4->addWidget(view);

    //Добавление связи между сигналами и слотами

    connect(calc,SIGNAL(clicked()),this,SLOT(createTree()));
    connect(findElemAddButton,SIGNAL(clicked()),this,SLOT(addElem()));
    connect(findElemDelButton,SIGNAL(clicked()),this,SLOT(delElem()));
    connect(instructions,SIGNAL(clicked()),this,SLOT(inst()));
    connect(clear,SIGNAL(clicked()),this,SLOT(startNew()));
    connect(numOfLE,SIGNAL(textChanged(const QString
&)),this,SLOT(addArray()));

    this->setLayout(layh4);
}

void ProgWindow::createTree()
{
    if(created){
        error(TREEISCREATED);
        return;
    }
    if(numOfLEi==0){//Проверка на ошибку отсутствия ввода
        error(NOINPUT);
        if(head!=nullptr)
            head->deleteTree();
        head=nullptr;
        treeScene->update(head,0);
        return;
    }
    for(int i=0;i<numOfLEi;i++){//Проверка но ошибку неверного ввода
        if(!(amounts[i]=loa->arr[i]->text().toInt())&&(loa->arr[i]-
>text()!="0")){
            error(WRONGINPUT);
            if(head!=nullptr)
                head->deleteTree();
            head=nullptr;
            treeScene->update(head,0);
            return;
        }
    }
    if(head!=nullptr)//Очистка предыдущего дерева
        head->deleteTree();
    head = new Node(amounts[0],0,0);//Добавление нового
    for(int i=1;i<numOfLEi;i++)
        head->addnew(amounts[i],i);
    treeScene->update(head,numOfLEi);
    created=true;
}

void ProgWindow::delElem()
{
    int e;
    if(whichElemDel->text()=="")
        return;
    if(head==nullptr){//Проврека на ошибку отсутствия дерева
```

```
            error(NOTREE);
            whichElemDel->setText("");
            return;
        }
        if(whichElemDel->text()=="0"){
            e=0;
        }else{
            if(!(e=whichElemDel->text().toInt())){//Проверка на ошибку
неверного ввода
                error(WRONGINPUT);
                whichElemDel->setText("");
                return;
            }
        }
        if(head->delElem(e)==0){//Удаление
            error(WRONGDELETE);//Ошибка несуществующего элемента или удаление
главного элемента
            whichElemDel->setText("");
            return;
        }
        int tmpn=0;
        head->reNumAll(&tmpn);//Перерасчет номеров элементов
        head->reDepthAll(0);//Перерасчет глубины
        numOfLEi--;
        treeScene->update(head,numOfLEi);
        whichElemDel->setText("");
}

void ProgWindow::inst()
{
        QMessageBox* instBox=new QMessageBox;
        instBox->setWindowTitle("Instructions");
        instBox->setText("1) В левой верхней строке указывается количество
элементов в дереве, после чего ниже появится такое же количество полей
для заполнения - сами элементы.\n"
                         "2) Нажатие кнопки генерации приведет к созданию
дерева в правой части окна программы.\n"
                         "3) Внизу слева программы присутствует поле ввода
добавляемого элемента и кнопка, добавляющая его. Ниже них поле ввода
удаляемого элемента и кнопка, удаляющая его.\n"
                         "4) Для полной очистки нажмите на кнопку Clear.");
        instBox->exec();
}

void ProgWindow::startNew()
{
        if(created==false){
            return;
        }
        created=false;
        numOfLEi=0;
        numOfLE->setText("");
        if(head!=nullptr)
            head->deleteTree();
        head=nullptr;
        treeScene->update(head,0);
}
```

```
void ProgWindow::addElem()
{
    int e;
    if(whichElemAdd->text()=="")
        return;
    if(head==nullptr){//Проврека на ошибку отсутствия дерева
        error(NOTREE);
        whichElemAdd->setText("");
        return;
    }
    if(whichElemAdd->text()=="0"){
        e=0;
    }else
        if(!(e=whichElemAdd->text().toInt())){//Проверка на ошибку
неверного ввода
            error(WRONGINPUT);
            whichElemAdd->setText("");
            return;
        }
    head->addnew(e,numOfLEi);
    numOfLEi+=1;
    treeScene->update(head,numOfLEi);
    whichElemAdd->setText("");
}

void ProgWindow::addArray()
{
    if(head!=nullptr)
        head->deleteTree();
    head=nullptr;
    treeScene->update(head,0);
    for(int i=0;i<maxn;i++){//Делаем невидимыми для пользователя все
строки
        loa->arr[i]->setVisible(false);
    }
    if(numOfLE->text()==""){//Если нет количества строк - все пустые
        loa->resize(0);
        for(int i=0;i<maxn;i++)
            loa->arr[i]->setText("");
        numOfLEi=0;
        return;
    }
    if(numOfLEi=numOfLE->text().toInt()){//Проверка на верный ввод
        if(numOfLEi>=maxn){//Проверка на ошибку большого количества
элементов
            error(BIGAMOUNT);
            loa->setN(maxn);
            return;
        }
        loa->resize(numOfLEi);
    }else{
        error(WRONGINPUT);
        numOfLE->setText("");
    }
}
```

```
LineOfAmounts::LineOfAmounts()
{
    //Выделение памяти и создание графического отображения строк ввода
элементов
    fr=new QFrame;
    l = new QVBoxLayout;
    for(int i=0;i<maxn;i++){
        arr[i]=new QLineEdit;
        arr[i]->setVisible(false);
        arr[i]->setFixedHeight(20);
        l->addWidget(arr[i]);
    }
    setWidgetResizable(true);
    l->setAlignment(Qt::AlignTop);
    fr->setLayout(l);

    setWidget(fr);
}

void LineOfAmounts::resize(int k)
{
    //Перерасчет количества отображаемых строк
    for(int i=0;i<maxn;i++){//Делаем невидимыми для пользователя все
строки
        arr[i]->setVisible(false);
    }
    for(int i=0;i<k;i++){
        arr[i]->setVisible(true);
    }

    this->n=k;
}

LineOfAmounts::~LineOfAmounts()
{
    for(int i=0;i<maxn;i++){
        delete arr[i];
    }
}
```

Файл visualizetree.h

```
#ifndef VISUALIZETREE_H
#define VISUALIZETREE_H
#include <QGraphicsScene>
#include <QGraphicsView>
#include <QGraphicsItem>
#include <QtMath>
#include "node.h"

class VisualizeTree: public QGraphicsScene
{
private:
    int maxn;//Максимальное количество элементов
    QGraphicsSimpleTextItem** items;//Массив элементов
    QGraphicsLineItem** lineItems;//Массив линий между ними
public:
    VisualizeTree(int maxn);
    void update(Node* head, int numOfLeaves);//Обновление картины дерева
```

```
    ~VisualizeTree();
};

#endif // VISUALIZETREE_H
```

Файл visualizetree.cpp

```
#include "visualizetree.h"

struct pt{
    int x;//Настоящий x
    int y;//Настоящий y
    int xlast;//Предыдущий x
    int ylast;//Предыдущий y
    int contains;//Содержит
    bool made=false;//Структура собрана/не собрана
};

pt findPlace(Node* tmphead, int elemNum, int leftMove, int rightMove, int
widthch, bool toLeft){
    pt retpt;
    if(tmphead->getElemNum()==elemNum){//Если совпадает - инициализируем
и возвращаем
        retpt.x=-leftMove+rightMove;
        retpt.y=tmphead->getDepth();
        retpt.ylast=retpt.y-1;
        if(toLeft){
            retpt.xlast=retpt.x+widthch*2;
        }else{
            retpt.xlast=retpt.x-widthch*2;
        }
        retpt.contains=tmphead->getA();
        retpt.made=true;
        return retpt;
    }
    //Ищем пока не совпадет
    if(tmphead->getLeft()!=nullptr){
        retpt=findPlace(tmphead-
>getLeft(),elemNum,leftMove+widthch,rightMove,widthch/2,true);
        if(retpt.made){//Чтобы в очередной раз не инициализировать после
входа в right
            return retpt;
        }
    }
    if(tmphead->getRight()!=nullptr){
        retpt=findPlace(tmphead-
>getRight(),elemNum,leftMove,rightMove+widthch,widthch/2,false);
        if(retpt.made){
            return retpt;
        }
    }
    return retpt;
}


VisualizeTree::VisualizeTree(int maxn)
{
    //Выделение памяти всех элементов и выключение их для пользователя
    this->maxn=maxn;
```

```
    items=new QGraphicsSimpleTextItem*[maxn];
    lineItems=new QGraphicsLineItem*[maxn];
    for(int i=0;i<maxn;i++){
        items[i]=new QGraphicsSimpleTextItem;
        lineItems[i]=new QGraphicsLineItem;
        addItem(items[i]);
        addItem(lineItems[i]);
        items[i]->setVisible(false);
        lineItems[i]->setVisible(false);
    }
}

void VisualizeTree::update(Node* head, int numOfLeaves)
{
    for(int i=0;i<maxn;i++){//очистим сцену
        items[i]->setVisible(false);
        lineItems[i]->setVisible(false);
    }

    for(int i=0;i<numOfLeaves;i++){//Для каждого элемента будем искать
его место и значение
        pt pl = findPlace(head,i,0,0,pow(2,(head->maxdepth())),true);
        items[i]->setText(QString::number(pl.contains));
        items[i]->setPos(pl.x*10,pl.y*30);
        if(i!=0){//Если 0 - рисовать линии наверх не надо
            lineItems[i]-
>setLine(pl.x*10,pl.y*30,pl.xlast*10,pl.ylast*30);
            lineItems[i]->setPen(Qt::DashLine);
            lineItems[i]->setVisible(true);
        }
        items[i]->setVisible(true);//Отображаем необходимые
    }
}

VisualizeTree::~VisualizeTree()
{
    //Очищаем выделенную под массивы память
    for(int i=0;i<maxn;i++){
        delete items[i];
        delete lineItems[i];
    }
    delete[] items;
    delete[] lineItems;
}
```

Файл main.cpp

```
#include <QApplication>
#include "progwindow.h"
using namespace std;

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    ProgWindow* pg = new ProgWindow;
    pg->setWindowTitle("Случайные БДП");
    pg->showMaximized();//Отображение главного окна
    return a.exec();
```

```
}
```