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

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Случайные БДП – вставка и исключение. Вставка в корень БДП.

Студент гр. 9381 Матвеев А. Н.

Преподаватель _____ Фирсов М. А.

_____

Санкт-Петербург
2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Матвеев А. Н.

Группа 9381

Тема работы:
Вариант 9. Случайные БДП – вставка и исключение. Вставка в корень БДП. Текущий контроль.

Исходные данные:
Размер БДП, сложность задания, тип задания, путь выходного файла, количество заданий.

Содержание пояснительной записки:

«Содержание», «Введение», «Формальная постановка задачи», «Описание алгоритма», «Описание структур данных и функций», «Описание интерфейса пользователя», «Заключение», «Список использованных источников».

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 21.12.2020

Дата защиты реферата: 22.12.2020

Студент		Матвеев А. Н.
Преподаватель		Фирсов М. А.

АННОТАЦИЯ

Задача курсовой работы состоит в разработке программы для для генерации заданий с ответами к ним для проведения текущего контроля среди студентов по теме случайные бинарные деревья поиска (вставка/исключение/вставка в корень). В качестве интерфейса для пользователя было решено реализовать интерфейс командной строки.

Курсовая работа состоит из пояснительной записки и исходного кода разработанной программы. В ходе работы была разработана программа с интерфейсом командной строки. Для написания программы использовался язык программирования C++.

SUMMARY

The task of the course work is to develop a program for generating tasks with answers to them for monitoring among students on the topic of random binary search trees (insert / exclude / insert into the root). It was decided to implement a command line interface as a user interface.

Course work consists of an explanatory note and the source code of the developed program. In the course of work, a program with a command line interface was developed. To write the program, the programming language C ++ was used.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы.

Разработка программы для для генерации заданий с ответами к ним для проведения текущего контроля среди студентов по теме “случайные бинарные деревья поиска” (вставка/исключение/вставка в корень).

Задачи.

- Изучение языка программирования C++;
- Изучение структуры данных “случайное бинарное дерево поиска”;
- Изучение алгоритмов обработки случайного БДП;
- Написание исходного кода программы;
- Программирование генерации заданий для студентов по данной теме;
- Сборка программы;
- Тестирование программы.

Основные теоретические положения.

Случайное бинарное дерево поиска (далее случайное БДП) - динамическая структура данных, представляющая собой бинарное дерево, в котором для каждого узла в левом поддереве находятся элементы, меньшие корня данного узла, а в правом - большие, причём в каждом узле есть счётчик попыток вставки данного элемента в узел. Структура случайного БДП полностью зависит от того («случайного») порядка, в котором элементы расположены во входной последовательности (во входном файле). В данной работе генерируются задания, предполагающие обработку этой структуры данных, а именно вставка некоторого элемента в БДП, исключение некоторого элемента из БДП, а также вставка некоторого элемента в корень БДП. В качестве типа элементов выбран тип int. БДП в данной программе реализовано не на базе указателей, а на базе массива.

ФОРМАЛЬНАЯ ПОСТАНОВКА ЗАДАЧИ

Пользователь вводит размер случайного БДП, сложность и тип задания, а также количество этих самых заданий. Программа генерирует их и выводит в файл или в консоль. Задания могут быть на вставку, вставку в корень и исключение.

1. ОПИСАНИЕ АЛГОРИТМА

Замечание.

Поскольку в данной реализации программы случайное БДП реализовано на базе массива, то вместо указателей на левые и правые поддеревья у каждого узла используются ссылки, выраженные индексами на элементы массива, где находятся правые и левые поддеревья. Также стоит отметить, что при удалении элемента он не удаляется, а просто обнуляется его поле, отвечающее за наличие элемента в векторе. Корень всегда находится в 0-м элементе вектора. Если правое или левое поддерево пусто, то соответствующая ссылка равна 0.

1.1. Вставка в случайное БДП.

При попытке вставки элемента в БДП обход начинается с корня. Сначала узел дерева проверяется на пустоту. Если он пуст, то туда записывается текущий элемент из входной строки. Если же узел не пуст, то поданный элемент сравнивается с весом данного узла.

Если первый меньше, то функция проверяет левого сына данного узла на пустоту. Если левое поддерево пусто, то ему присваивается ссылка на некоторое место в массиве, задаваемое формулой 2*index + 1 (index - индекс текущего узла в массиве). Такая формула вставки нужна для избежания конфликтов при расстановке ссылок. Если не пусто, то сразу рекурсивно вызывается данная функция для левого поддерева (она вызывается в любом случае).

Если же поданный элемент больше веса узла, то функция проверяет правого сына данного узла на пустоту. Если правое поддерево пусто, то ему присваивается ссылка на некоторое место в массиве, задаваемое формулой 2*index + 2 (index - индекс текущего узла в массиве). Если не пусто, то сразу рекурсивно вызывается данная функция для правого поддерева. (она вызывается в любом случае).

Если элемент и вес узла совпадают, то просто увеличивается на 1 счетчик попыток вставки данного элемента в данный узел.

1.2. Исключение элемента из случайного БДП.

Удалить элемент из случайного БДП проще всего, если этот элемент находится в листе дерева. Тогда данный лист непосредственно удаляется (обнуляется поле count). Если же удаляемый элемент находится во внутреннем узле b, то в ситуации b.rightTree != 0 следует найти минимальный элемент правого поддерева, рекурсивно удалить его и заменить им содержимое узла b.

В ситуации b.rightTree == 0 (поскольку узел b − не лист, то, следовательно, b.leftTree != 0) находим максимальный элемент левого поддерева, рекурсивно удаляем его и заменяем им содержимое узла b.

1.3. Левый/правый поворот случайного БДП.

Правый поворот (правое вращение) - нерекурсивная процедура, позволяющая заменить текущий корень дерева его левым сыном с сохранением свойств БДП. Бывший корень при этом становится правым сыном нового корня. Новым левым сыном бывшего корня становится бывший правый сын нового корня (при его наличии).

Левый поворот (правое вращение) - нерекурсивная процедура, позволяющая заменить текущий корень дерева его правым сыном с сохранением свойств БДП. Бывший корень при этом становится левым сыном нового корня. Новым правым сыном бывшего корня становится бывший левый сын нового корня (при его наличии).

1.4. Вставка в корень случайного БДП.

1) Если вставляемый элемент больше корня БДП, то его место в правом поддереве. Поэтому сначала рекурсивно вставим этот элемент в качестве

корня правого поддерева,а затем с помощью левого поворота поднимем его в корень дерева.

2) Если вставляемый элемент меньше корня БДП, то его место в левом поддереве. Поэтому сначала рекурсивно вставим этот элемент в качестве корня левого поддерева, а затем с помощью правого поворота поднимем его в корень дерева.

2. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ФУНКЦИЙ

2.1. Класс BinSTree.

Представляет собой случайное бинарное дерево поиска. Имеет приватные поля **int count** (количество попыток вставки), **Elem info** (вес узла - целое число) , **int leftTree** (ссылка на левое поддерево, представленная индексом массива), **int rightTree** (ссылка на правое поддерево, представленная индексом массива). В качестве публичных методов помимо **конструктора** и **деструктора по умолчанию** реализованы статические методы, которые и представляют собой набор необходимых функций, нужных для работы программы. Среди них:

- **static void searchAndInsert(vector<BinSTree> & array, Elem x, int index)** - функция, позволяющая по записи построить случайное БДП;
- **static void printBST(vector<BinSTree> & array, int index, int move, std::ostream &out)** - функция, изображающая бинарное дерево поиска в наглядном виде;
- **static int getMinRight(vector <BinSTree> & array, int index)** - получение индекса минимального узла правого поддерева;
- **static int getMaxLeft(vector <BinSTree> & array ,int index) -** - получение индекса максимального узла левого поддерева;
- **static int deleteElem(vector <BinSTree> & array , int index, Elem x, ostream &out, int n)** - исключение элемента из БДП;
- **static int deleteElemTest(vector <BinSTree> & array, int index, Elem x, int & changeCount) -** исключение элемента из БДП без отладочных выводов, с накоплением счётчика количества замен для генерации заданий определённой сложности;
- **static void rotateR(vector <BinSTree> & array, int index)** - правый поворот БДП;
- **static void rotateL(vector <BinSTree> & array ,int index)** - левый поворот БДП;

- **static void insInRoot(vector <BinSTree> & array, int index, Elem &x, ostream &out, int n)** - вставка элемента в корень БДП;
- **static void insInRootTest(vector<BinSTree> & array, int index, Elem & x, int & rotateCount) -** вставка элемента в корень БДП без отладочных выводов, с накоплением счётчика количества поворотов для генерации заданий определённой сложности;
- **static void printIndent(int n, std::ostream & out)** - функция, печатающая необходимое количество отступов (нужна в отладочных выводах).

Описание функций класса.

static void searchAndInsert(vector<BinSTree> & array, Elem x, int index) - функция, позволяющая по записи построить случайное БДП. Принимает на вход ссылку на вектор, в котором хранятся узлы БДП, текущий элемент x(который предполагается вставить), индекс массива index (ссылку на узел, начиная с которой происходит обработка дерева) Этот статический метод ничего не возвращает.

Стоит заметить, что вектор на протяжении всего хода программы содержит некоторое количество нулевых узлов (все поля узла = 0 изначально), и если текущий индекс при заполнении больше вместимости вектора, то к вектору применяется метод resize() и вместимость увеличивается на 20 элементов.

Если поле count текущего узла нулевое (изначально все поля узла нулевые), то оно устанавливается в 1 и полю info присваивается поданный элемент (далее - x).

Иначе если x < info, следует зайти в левое поддерево и проверить его на возможность вставки (эта функция вызывается рекурсивно для левого поддерева). Причём если поле leftTree не инициализировано (= 0), его надо инициализировать индексом массива по формуле 2*index + 1, чтобы не возникало конфликтов перезаписи ссылок на узлы. (index - ссылка, представленная индексом, на текущий узел).

Если x > info, указанные выше действия выполняются для правого поддерева и они совершенно аналогичны за тем исключением, что инициализация rightTree происходит по формуле 2*index + 2.

В случае если значение x совпало с значением узла, то поле узла count увеличивается на 1.

static int getMinRight(vector <BinSTree> & array, int index) - принимает на вход ссылку на вектор элементов БДП array и индекс index, начиная с которого нужно производить поиск; Возвращает индекс минимального узла правого поддерева текущего корня (он самый левый в правом поддереве). Если ссылка на левое поддерева нулевая, то возвращается индекс, иначе рекурсивно возвращается значение этой функции для левого поддерева данного узла.

static int getMaxLeft(vector <BinSTree> & array ,int index) - принимает на вход ссылку на вектор элементов БДП array и индекс index, начиная с которого нужно производить поиск; Возвращает индекс максимального узла левого поддерева текущего корня. Если ссылка на правое поддерево нулевая, то возвращается индекс, иначе рекурсивно возвращается значение этой функции для правого поддерева данного узла.

static int deleteElem(vector <BinSTree> & array , int index, Elem x, ostream &out, int n) - принимает на вход ссылку на вектор элементов БДП array, индекс index, начиная с которого производится поиск удаляемого элемента, элемент x, который требуется удалить, ссылку на поток вывода out и целое число n, отвечающее за количество отступов в отладочных выводах. Возвращает индекс текущего узла. Если ключ текущего узла > x, то эта функция рекурсивно запускается для левого поддерева данного узла, возвращаемое ей значение присваивается ссылке на левое поддерево текущего узла. Если ключ текущего узла < x, то эта функция рекурсивно запускается для

правого поддерева данного узла, возвращаемое ей значение присваивается ссылке на правое поддерево текущего узла.

Иначе, если ключ узла совпал с удаляемым элементом (нужный узел найден), то проверяется на пустоту правое поддерево. Если оно не пусто, то значение текущего узла заменяется на значение минимального узла из правого поддерева (доступ к нему осуществляется при помощи функции getMinRight()). Ссылке на правое поддерево присваивается значение, возвращаемое функцией deleteElem(), где в качестве индекса выступает правое поддерево текущего узла.

Иначе, если ключ совпал, но правое поддерево пусто, то проверяется на пустоту левое поддерево. Если оно не пусто, то значение текущего узла заменяется на значение максимального о узла из левого поддерева (доступ к нему осуществляется при помощи функции getMaxLeft()). Ссылке на левое поддерево присваивается значение, возвращаемое функцией deleteElem(), где в качестве индекса выступает левое поддерево текущего узла.

Иначе, если ключ совпал, но оба поддерева пусты, то данные элемент удаляется путём обнуления его поля count.

static int deleteElemTest(vector <BinSTree> & array, int index, Elem x, int & changeCount) - принимает на вход ссылку на вектор элементов БДП array, индекс index, начиная с которого производится поиск удаляемого элемента, элемент x, который требуется удалить, и ссылку на целое число changeCount, отвечающее за количество замен элемента. Возвращает индекс текущего узла. Работает совершенно аналогично предыдущей функции за тем исключением, что каждый раз, когда следует совершить замену элемента, changeCount увеличивается. Эта модификация используется при подборе последовательности чисел и удаляемого элемента при генерации заданий.

static void rotateR(vector <BinSTree> & array, int index) - функция, осуществляющая правый поворот БДП. Принимает на вход ссылку на вектор

array, хранящий узлы, и индекс узла, который выступает корнем поворачиваемого поддерева. Ничего не возвращает.

Если текущий узел пуст, происходит возврат из функции. Иначе, если узел не пуст и у него имеется непустое левое поддерево, создаются 2 копии: newRoot - левый сын старого корня и старый корень oldRoot. Затем в текущий корень переписываются все данные из NewRoot кроме ссылки на правое поддерево. Затем на место левого сына старого корня в массиве записывается копия старого корня, после чего ссылке на правого сына текущего корня присваивается ссылка на левое поддерево старого корня (значение по ней уже обновлено). Наконец, ссылке на левое поддерево правого поддерева текущего корня присваивается ссылка на правое поддерево из newRoot.

Таким образом, происходит поворот БДП вправо.

static void rotateL(vector <BinSTree> & array, int index) - функция, осуществляющая левый поворот БДП. Принимает на вход ссылку на вектор array, хранящий узлы, и индекс узла, который выступает корнем поворачиваемого поддерева. Ничего не возвращает.

Если текущий узел пуст, происходит возврат из функции. Иначе, если узел не пуст и у него имеется непустое правое поддерево, создаются 2 копии: newRoot - правый сын старого корня и старый корень oldRoot. Затем в текущий корень переписываются все данные из NewRoot кроме ссылки на левое поддерево. Затем на место правого сына старого корня в массиве записывается копия старого корня, после чего ссылке на левого сына текущего корня присваивается ссылка на правое поддерево старого корня (значение по ней уже обновлено). Наконец, ссылке на правое поддерево левого поддерева текущего корня присваивается ссылка на левое поддерево из newRoot.

Таким образом, происходит поворот БДП влево.

static void insInRoot(vector <BinSTree> & array, int index, Elem &x, ostream &out, int n) - функция вставки в корень БДП. Принимает на вход ссылку на вектор элементов БДП array, индекс index, являющийся корнем текущего поддерева, элемент x, который требуется вставить, ссылку на поток вывода out и целое число n, отвечающее за количество отступов в отладочных выводах. Ничего не возвращает.

Если текущий корень пуст, то элемент x вставляется непосредственно. Иначе, если x меньше значения узла, то функция рекурсивно запускается для левого поддерева данного дерева (причем если ссылка на левое поддерево нулевая, в конец массива добавляется ещё один элемент и ссылка инициализируется). После этого запускается функция rotateR().

Иначе, если x больше значения узла, то функция рекурсивно запускается для правого поддерева данного дерева (причем если ссылка на правое поддерево нулевая, в конец массива добавляется ещё один элемент и ссылка инициализируется). После этого запускается функция rotateL().

Если же элемент уже существует, то просто увеличивается на 1 его поле count. (Но в генерации заданий такая ситуация исключена).

Так, при помощи левого или правого поворота вставленный элемент поднимается в корень дерева.

void insInRootTest(vector<BinSTree> & array, int index, Elem & x, int & rotateCount) - функция вставки в корень БДП. Принимает на вход ссылку на вектор элементов БДП array, индекс index, являющийся корнем текущего поддерева, элемент x, который требуется вставить и ссылку на целое число rotateCount, отвечающее за количество поворотов. Ничего не возвращает. Эта функция используется при генерации заданий на вставку в корень, когда необходимо подобрать такую последовательность и такой элемент, чтобы количество поворотов при вставке было заданным. Работает аналогично предыдущей за тем исключением, что перед запуском функции поворота увеличивается значение переменной rotateCount.

static void printBST(vector<BinSTree> & array, int index, int move, std::ostream &out) - функция, изображающая бинарное дерево поиска в наглядном виде. Принимает на вход ссылку на вектор, в котором хранятся узлы БДП, индекс массива index (ссылку на узел, начиная с которого производится обход), целочисленную переменную move, отвечающую за количество отступов при визуализации БДП и ссылку на поток вывода out. Ничего не возвращает. Если индекс больше размера массива, то происходит возврат из функции. Если поле count не равно 0 для конкретного узла, то, при ненулевом левом или правом поддереве выполняется обход сначала правого поддерева (функция вызывает саму себя для поля rightTree), затем выводится корень с соответствующим количеством отступов, а потом производится обход левого поддерева (функция вызывает саму себя для поля leftTree).

static void printIndent(int n, std::ostream & out) - печать отступов в отладочных выводах. Принимает на вход целое число, отвечающее за количество отступов и ссылку на поток вывода. Ничего не возвращает.

2.2. Функции для генерации заданий и вспомогательные функции.

void genBinSTree(vector<BinSTree> & array, vector<Elem> & elemsBDP, int lowerBorder, int upperBorder, int &size) - функция, заполняющая последовательность элементов БДП случайными числами и создающая БДП из этих чисел. Принимает на вход ссылку на вектор узлов будущего БДП array, ссылку на вектор элементов БДП elemsBDP, нижнюю границу диапазона генерируемых чисел lowerBorder, верхнюю границу диапазона генерируемых чисел upperBorder, а также ссылку на размер дерева size. Ничего не возвращает.

void CreateTree(vector<BinSTree> & array,vector<int> &copy) - генерирует дерево по уже заданной последовательности. Принимает на вход ссылку на вектор узлов будущего БДП array, ссылку на вектор элементов БДП copy. Ничего не возвращает.

int genNumberForInsert(vector<int> & elemsBDP , int lowerBorder, int upperBorder) - генерирует число, которое предполагается вставить в БДП. Принимает на вход ссылку на вектор элементов БДП elemsBDP, нижнюю границу диапазона генерируемых чисел lowerBorder, верхнюю границу диапазона генерируемых чисел upperBorder. Возвращает сгенерированное число.

int getNumberToDel(vector<int> & elemsBDP) - генерирует число, которое предполагается удалить из БДП. Принимает ссылку на вектор элементов БДП. Возвращает сгенерированное число.

int getTaskInsInRoot(vector<BinSTree> & array,vector<Elem> & copy,int lowerBorder, int upperBorder, int &size, string & complex) - генерирует такую последовательность элементов и возвращает такое число, при вставке в корень которого будет заданное количество поворотов в зависимости от сложности.

Принимает на вход ссылку на вектор узлов будущего БДП array, ссылку на вектор элементов БДП copy, нижнюю границу диапазона генерируемых чисел lowerBorder, верхнюю границу диапазона генерируемых чисел upperBorder, ссылку на размер дерева size, а также ссылку на строку complex, в которой есть информация о сложности задания (“e” - 1 поворот / “m” - 3 поворота). Возвращает число которое нужно вставить в корень БДП. В результате работы этой функции в copy будет содержаться некоторая последовательность size элементов, а при вставке возвращаемого значения в дерево, построенное по этой последовательности, будет всегда определённое количество поворотов в зависимости от сложности, что позволяет генерировать задания разной сложности.

Генерация таких заданий переборная, в цикле вызываются функции genBinSTree(), genNumberForInsert() и insInRootTest().

int getTaskDel(vector<BinSTree> & array,vector<Elem> & copy,int lowerBorder, int upperBorder, int &size, string & complex) - принимает на вход ссылку на вектор узлов будущего БДП array, ссылку на вектор элементов БДП copy, нижнюю границу диапазона генерируемых чисел lowerBorder, верхнюю границу диапазона генерируемых чисел upperBorder, ссылку на размер дерева size, а также ссылку на строку complex, в которой есть информация о сложности задания (“e” - 1 замена / “m” - 3 замены). Возвращает число, которое нужно удалить из БДП и заполняет copy такой последовательностью чисел, чтобы при удалении возвращаемого числа из БДП было строго определённое количество поворотов в зависимости от сложности.

void manual() - функция справки. Ничего не принимает, ничего не возвращает.

2.3. Функция main().

Используется интерфейс командной строки, реализованный применением встроенной функции getopt_long().

Сначала прописывается вызов функции srand(time(nullptr)) для возможности генерации псевдослучайных чисел, т.к. нужно, чтобы генерировались задания с разными БДП.

Затем в цикле последовательно разбираются аргументы командной строки и инициализируются переменные, отвечающие за сложность, размер, тип и количество заданий и т. д. Далее, устанавливается поток вывода в консоль или файл в зависимости от того, введён ли путь до файла или нет. Наконец, в зависимости от задания, генерируется заданное пользователем количество заданий (при помощи цикла и условного оператора). Предусмотрена обработка ошибок.

3. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Примечание:

1) Генерируемые задания покрывают тему в полной мере, т.к. есть задания на обычную вставку, вставку в корень и удаление элемента.
2) Есть 2 типа сложности: лёгкая и средняя, присутствует её плавная регулировка:

- Задания на обычную вставку вне зависимости от выбора пользователя являются лёгкими в силу простоты самой идеи;
- Задания на удаление элемента из БДП бывают лёгкие и средние:
    - лёгкие - 1 замена при удалении;
    - средние - 3 замены при удалении;
- Задания на вставку в корень БДП тоже бывают лёгкие и средние:
    - лёгкие - 1 поворот при вставке;
    - средние - 3 поворота при вставке.

3) При генерации задания присутствуют отладочные выводы для удобства проверки заданий.

Интерфейс, предоставленный пользователю.

Пользователь запускает программу через терминал linux и вводит следующие параметры:

- --size <arg> / -s <arg> - размер БДП;
- --compl <arg> / -c <arg> - сложность: <e/m>, e - лёгкая, m - средняя;
- --type <arg> / -t <arg> - тип задания: <insert/insInRoot/delete> , где insert - обычная вставка, insInRoot - вставка в корень, delete - удаление элемента;
- --outfile <arg> / -f <arg> - путь до файла в качестве <arg> (при отсутствии этого параметра следует вывод в консоль);
- --help / -h - справка;
- --num <arg> / -n <arg> - количество заданий;

Пример ввода: ./cw --size 3 --compl m --type insInRoot --outfile /home/andrey/result --num 5

ЗАКЛЮЧЕНИЕ

В ходе работы над поставленным заданием была изучена структура данных “случайное бинарное дерево поиска” операции работы с ней, а также была разработана программа с интерфейсом командной строки, которая генерирует задания различной сложности на темы “вставка”, “исключение” и “вставка в корень БДП” для прохождения текущего контроля среди студентов. Программа была успешно протестирована на работоспособность.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б. И Ритчи Д. Язык программирования Си.

2. Русскоязычный форум Хабр // Habr URL: https://habr.com/ru/

3. Collection of knowledge and practical advices about programming and information technologies. // EVILEG.COM URL: https://evileg.com/ru/

ПРИЛОЖЕНИЕ А
ТЕСТИРОВАНИЕ

№ п/п	Входные данные	Выходные данные
1.	./cw -s 2 --compl m --type insert --num 1	Замечание: сложность данного типа задания лёгкая вне зависимости от выбора пользователя вследствие простоты идеи. 1. Вставьте элемент 479 в бинарное дерево размера 2: __698 536 ОТВЕТ: __698 536 __479
2.	./cw --size 10 --compl m --type delete --num 3	1. Удалите элемент 851 из бинарного дерева размера 10: 851 __845 _____709 _____688 _____614 ____596 _____501 _____402 _____257 _____87 ОТВЕТ: Заменяем элемент (851) самым правым из левого поддерева (845) Заменяем элемент (845) самым правым из левого поддерева (709) (709) > (596). Заход в правое поддерево (709) > (614). Заход в правое поддерево

		Заменяем элемент (709) самым правым из левого поддерева (688) Теперь левое поддерево обновлённого элемента равно (688) Теперь левое поддерево обновлённого элемента равно (596) Теперь левое поддерево обновлённого элемента равно (709) 845 __709 _____688 _____614 ____596 _____501 _____402 _____257 _____87 2. Удалите элемент 431 из бинарного дерева размера 10: __949 947 ____858 _____806 _____583 _____525 _____474 __431 _____277 ____156 ОТВЕТ: (431) < (947). Заход в левое поддерево Заменяем элемент (431) самым левым из правого поддерева (474)

		(474) < (858). Заход в левое поддерево (474) < (806). Заход в левое поддерево Заменяем элемент (474) самым левым из правого поддерева (525) Заменяем элемент (525) самым левым из правого поддерева (583) Теперь правое поддерево обновлённого элемента равно (583) Теперь правое поддерево обновлённого элемента равно (583) Теперь правое поддерево обновлённого элемента равно (858) __949 947 ____858 ______806 _____583 _____525 __474 _____277 ____156 3. Удалите элемент 991 из бинарного дерева размера 10: __991 ____970 _____755 _____752 588 ____541 _____418 __206 ____163 _____20

		ОТВЕТ: (991) > (588). Заход в правое поддерево Заменяем элемент (991) самым правым из левого поддерева (970) Заменяем элемент (970) самым правым из левого поддерева (755) Заменяем элемент (755) самым правым из левого поддерева (752) Теперь левое поддерево обновлённого элемента равно (752) Теперь левое поддерево обновлённого элемента равно (752) Теперь левое поддерево обновлённого элемента равно (755) __970 ____755 ______752 588 ____541 ______418 __206 ____163 ______20
3.	./cw -s 7 --compl e --type insInRoot --num 2	1. Вставьте элемент 189 в корень бинарного дерева размера 7: ______948 ____746 __675 ________580 ______459 ____426 244 ОТВЕТ:

		Левое поддерево корня (244) пусто. Вставка... (189) < (244) Правое вращение. Старый корень текущего поддерева: (244). Новый корень текущего поддерева: (189) _____948 _____746 ____675 _____580 _____459 _____426 __244 189 2. Вставьте элемент 345 в корень бинарного дерева размера 7: ____632 __621 _____599 ____576 _____521 _____497 430 ОТВЕТ: Левое поддерево корня (430) пусто. Вставка... (345) < (430) Правое вращение. Старый корень текущего поддерева: (430). Новый корень текущего поддерева: (345) _____632 ____621 _____599 _____576 _____521 _____497

		__430 345
4.	./cw -s 7 --compl m --type insInRoot --num 3	1. Вставьте элемент 441 в корень бинарного дерева размера 7: _____908 _____879 _____629 _____536 ____519 __170 122 ОТВЕТ: (441) > (122) (441) > (170) Левое поддерево корня (519) пусто. Вставка... (441) < (519) Правое вращение. Старый корень текущего поддерева: (519). Новый корень текущего поддерева: (441) Левое вращение. Старый корень текущего поддерева: (170). Новый корень текущего поддерева: (441) Левое вращение. Старый корень текущего поддерева: (122). Новый корень текущего поддерева: (441) _____908 _____879 ____629 _____536 __519 441 ____170 __122

		2. Вставьте элемент 248 в корень бинарного дерева размера 7: __995 ____827 329 ____118 ______66 __58 ____53 ОТВЕТ: (248) < (329) (248) > (58) Правое поддерево корня (118) пусто. Вставка... (248) > (118) Левое вращение. Старый корень текущего поддерева: (118). Новый корень текущего поддерева: (248) Левое вращение. Старый корень текущего поддерева: (58). Новый корень текущего поддерева: (248) Правое вращение. Старый корень текущего поддерева: (329). Новый корень текущего поддерева: (248) ____995 ______827 __329 248 ____118 ______66 __58 ____53 3. Вставьте элемент 183 в корень бинарного дерева размера 7:

		_____884 ____790 _____706 __629 _____502 ____258 139 ОТВЕТ: (183) > (139) (183) < (629) Левое поддерево корня (258) пусто. Вставка... (183) < (258) Правое вращение. Старый корень текущего поддерева: (258). Новый корень текущего поддерева: (183) Правое вращение. Старый корень текущего поддерева: (629). Новый корень текущего поддерева: (183) Левое вращение. Старый корень текущего поддерева: (139). Новый корень текущего поддерева: (183) _____884 ____790 _____706 __629 _____502 ____258 183 __139
5.	./cw -s 6 --compl m --type delete --num 5	1. Удалите элемент 401 из бинарного дерева размера 6: __704 665 __401

		____399 _____372 _____223 ОТВЕТ: (401) < (665). Заход в левое поддерево Заменяем элемент (401) самым правым из левого поддерева (399) Заменяем элемент (399) самым правым из левого поддерева (372) Заменяем элемент (372) самым правым из левого поддерева (223) Теперь левое поддерево обновлённого элемента равно (223) Теперь левое поддерево обновлённого элемента равно (223) Теперь левое поддерево обновлённого элемента равно (372) __704 665 __399 ____372 _____223 2. Удалите элемент 886 из бинарного дерева размера 6: __886 ____781 _____658 _____394 179 __18 ОТВЕТ: (886) > (179). Заход в правое поддерево

		Заменяем элемент (886) самым правым из левого поддерева (781) Заменяем элемент (781) самым правым из левого поддерева (658) Заменяем элемент (658) самым правым из левого поддерева (394) Теперь левое поддерево обновлённого элемента равно (394) Теперь левое поддерево обновлённого элемента равно (394) Теперь левое поддерево обновлённого элемента равно (658) __781 ____658 ______394 179 __18 3. Удалите элемент 25 из бинарного дерева размера 6: _____597 _____516 _____401 ____394 __168 25 ОТВЕТ: Заменяем элемент (25) самым левым из правого поддерева (168) Заменяем элемент (168) самым левым из правого поддерева (394) Заменяем элемент (394) самым левым из правого поддерева (401) (401) < (516). Заход в левое поддерево

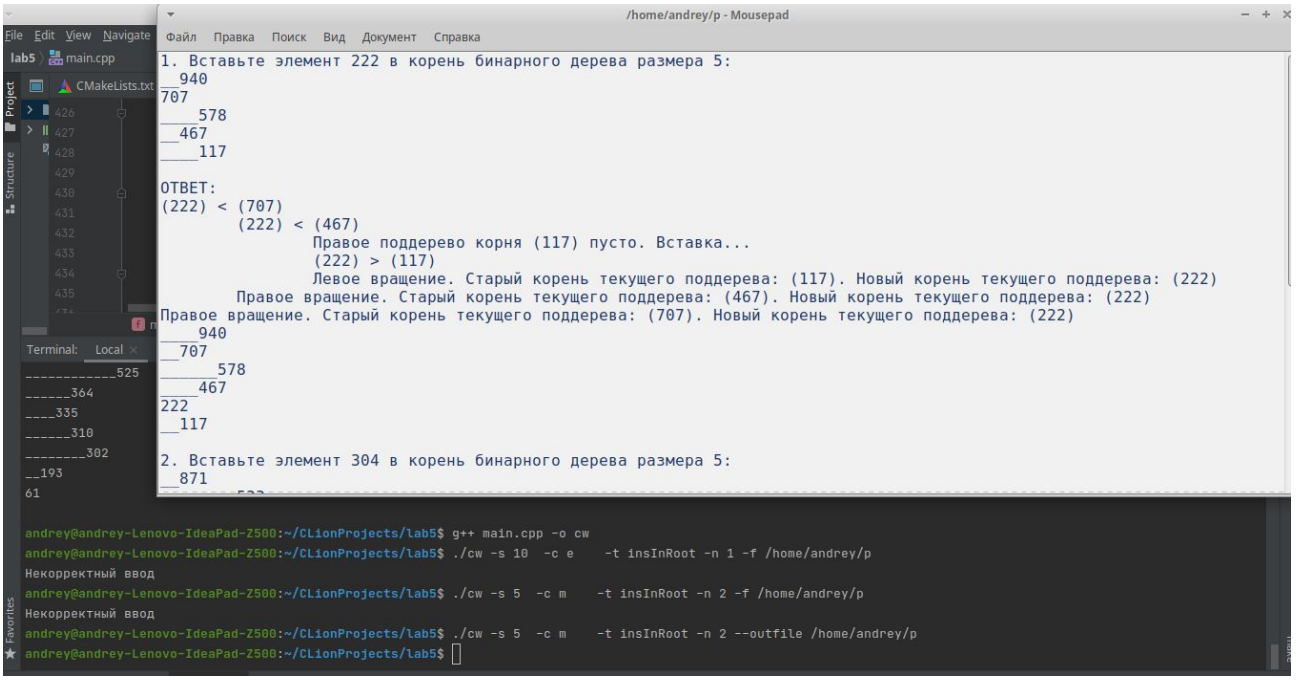
		Теперь правое поддерево обновлённого элемента равно (516) Теперь правое поддерево обновлённого элемента равно (401) Теперь правое поддерево обновлённого элемента равно (394) _____597 _____516 ____401 __394 168 4. Удалите элемент 936 из бинарного дерева размера 6: __936 ____853 _____845 _____130 76 __8 ОТВЕТ: (936) > (76). Заход в правое поддерево Заменяем элемент (936) самым правым из левого поддерева (853) Заменяем элемент (853) самым правым из левого поддерева (845) Заменяем элемент (845) самым правым из левого поддерева (130) Теперь левое поддерево обновлённого элемента равно (130) Теперь левое поддерево обновлённого элемента равно (130) Теперь левое поддерево обновлённого элемента равно (845)

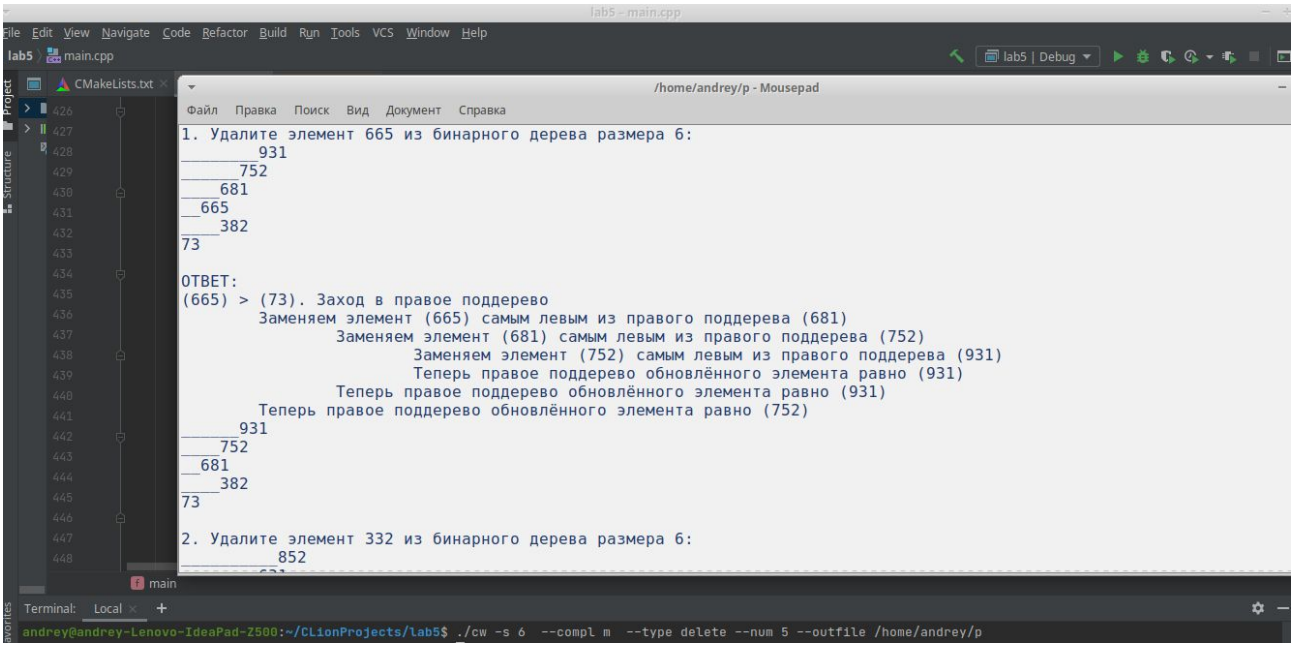
		__853 ____845 ______130 76 __8 5. Удалите элемент 728 из бинарного дерева размера 6: ______940 ____889 __773 728 ____447 __196 ОТВЕТ: Заменяем элемент (728) самым левым из правого поддерева (773) Заменяем элемент (773) самым левым из правого поддерева (889) Заменяем элемент (889) самым левым из правого поддерева (940) Теперь правое поддерево обновлённого элемента равно (940) Теперь правое поддерево обновлённого элемента равно (940) Теперь правое поддерево обновлённого элемента равно (889) ____940 __889 773 ____447 __196

Таблица 2. Тестирование на некорректных данных

№ п/п	Входные данные	Выходные данные
1.	./cw --size 10 --compl m --type dele --num 3	Некорректный тип задания
2.	./cw --size 1 --compl m --type delete --num 3	Слишком маленький размер БДП для данного задания с данной сложностью
3.	./cw --size 10 --compl g --type insInRoot --num 5	Некорректное значение сложности

Скриншоты работы программы






```
andrey@andrey-Lenovo-IdeaPad-Z500:~/CLionProjects/lab5$  ./cw --size 10  --compl m  --type dele --num 3
Некорректный тип задания
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: main.cpp

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>
#include <getopt.h>
#include <ctime>
#include <algorithm>
#include <ctime>
#include <cstring>
#define SHORTOPTS "s:c:t:o:n:f:h"

using namespace std;
typedef int Elem;
class BinSTree;

int getTaskInsInRoot(vector<BinSTree> & array, vector<Elem> & copy,int
lowerBorder, int upperBorder, int &size, string & complex);
int  getTaskDel(vector<BinSTree>  &  array,  vector<Elem>  &  copy,int
lowerBorder, int upperBorder, int &size, string & complex);
void genBinSTree(vector<BinSTree> & array, vector<Elem> & elemsBDP, int
lowerBorder, int upperBorder, int &size);
int getNumberToDel(vector<int> & elemsBDP );
int  genNumberForInsert(vector<int>  &  elemsBDP  ,  int  lowerBorder,  int
upperBorder );
void CreateTree(vector<BinSTree> & array,vector<int> &copy);
void manual();

class BinSTree {// случайное бинарное дерево поиска

    int count = 0; // количество попыток вставки
    Elem info = 0;// данные узла
```

```
    int leftTree = 0; // левое поддерево
    int rightTree = 0; // правое поддерево
public:
    BinSTree() = default;// конструктор класса БД
    ~BinSTree() = default; // деструктор класса БД
     static void searchAndInsert(vector<BinSTree> &array, Elem x, int
index) { // функция, позволяющая по записи построить случайное БДП

        if (index >= array.capacity() || index >= array.size()) {
            array.resize(index + 20); // если требуется, размер массива
увеличивается
        }
        BinSTree &tmp = array.at(index);
           if (tmp.count == 0) { // если поле count текущего узла = 0
(узел свободен), то инициализируем его поданным на вход элементом
            tmp.count = 1;
            tmp.info = x;
         } else if (x < tmp.info) { // если поданное значение < значения
узла, то переходим к его левому сыну
              if (tmp.leftTree == 0) { /* если левого сына ещё нет, то
даём узлу ссылку в виде индекса на место в векторе, где будет
находиться
                  его левое поддерево , в массиве эта ссылка задаётся
формулой 2*(индекс узла) + 1 */
                tmp.leftTree = 2 * index + 1;
            }
               searchAndInsert(array, x, tmp.leftTree); // рекурсивный
вызов для левого сына текущего узла
        } else if (x > tmp.info) { // если поданное значение > значения
узла, то переходим к его правому сыну
             if (tmp.rightTree == 0) {/* если правого сына ещё нет, то
даём узлу ссылку в виде индекса на место в векторе, где будет
находиться
                 его правое поддерево , в массиве эта ссылка задаётся
формулой 2*(индекс узла) + 2 */
                tmp.rightTree = 2 * index + 2;
            }
```

```
                searchAndInsert(array, x, tmp.rightTree); // рекурсивный
вызов для правого сына текущего узла
        } else { // поданное значение уже есть в дереве
                array.at(index).count++; // увеличиваем счётчик попыток
вставки на 1
        }
    } //+

    static void printBST(vector<BinSTree> & array, int index, int move,
std::ostream &out) // функция, изображающая бинарное дерево поиска
(повернутое) в наглядном виде
    { // БДП выводится справа налево с необходимым количеством отступов
        // это позволяет представить БДП в привычном виде
         // эта функция также изображает обход дерева в порядке правый
-> корень -> левый
        if (index >= array.size())
            return;
        BinSTree &tmp = array.at(index);
        if (tmp.count != 0) { // если текущий узел непуст
              if (tmp.rightTree != 0) // если правый сын узла непуст,
производится рекурсивный обход правого поддерева
                printBST(array,tmp.rightTree, move + 2, out);
            for (int i = 0; i < move; i++)
                out << "_";
                 out << tmp.info << std::endl; // когда правое дерево
полностью просмотрено, выводится корень
                if (tmp.leftTree != 0) // если левый сын узла непуст,
производится рекурсивный обход левого поддерева
                printBST(array,tmp.leftTree, move + 2, out);
        }
    }


    static int getMinRight(vector <BinSTree> & array, int index) {  //
получение индекса минимального узла в правом поддереве
        BinSTree &tmp = array.at(index);
        if (tmp.leftTree == 0) {
            return index;
```

```
        } else {
            return getMinRight(array,tmp.leftTree);
        }
    }

     static int getMaxLeft(vector <BinSTree> & array ,int index){ //
получение индекса максимального узла в левом поддереве
        BinSTree & tmp = array.at(index);
        if(tmp.rightTree == 0){
            return index;
        }
        else{
            return getMaxLeft(array, tmp.rightTree);
        }
    }


    static int deleteElem(vector <BinSTree> & array , int index, Elem
x, ostream &out, int n) { //удаление элемента по ключу
        BinSTree & tmp = array.at(index);
        if(tmp.count == 0)
            return index;
        if(x < tmp.info) {
            printIndent(n, out);
             out << "(" << x << ")" << " < (" <<  array.at(index).info
<< "). Заход в левое поддерево\n";
             tmp.leftTree = deleteElem(array,tmp.leftTree, x, out, n +
1);
        }
        else if(x > tmp.info) {
            printIndent(n, out);
             out << "(" << x << ")" << " > (" <<  array.at(index).info
<< "). Заход в правое поддерево\n";
            tmp.rightTree = deleteElem(array,tmp.rightTree, x, out, n +
1);
        }
        else if(tmp.rightTree != 0){
            printIndent(n, out);
```

```
            out << "Заменяем элемент ("<< x <<") самым левым из правого поддерева (" << array.at(getMinRight(array,tmp.rightTree)).info << ")\n";
                                tmp.info = array.at(getMinRight(array, tmp.rightTree)).info;
              tmp.rightTree = deleteElem(array,tmp.rightTree, tmp.info, out, n+1);
            printIndent(n, out);
            out << "Теперь правое поддерево обновлённого элемента равно (" << array.at(tmp.rightTree).info << ")\n";
        }
        else{
            if(tmp.leftTree != 0){
                printIndent(n, out);
                  out << "Заменяем элемент ("<< x <<") самым правым из левого поддерева (" << array.at(getMaxLeft(array,tmp.leftTree)).info << ")\n";
                                                        tmp.info = array.at(getMaxLeft(array,tmp.leftTree)).info;
                tmp.leftTree = deleteElem(array,tmp.leftTree, tmp.info, out, n+1);
                printIndent(n, out);
                  out << "Теперь левое поддерево обновлённого элемента равно (" << array.at(tmp.leftTree).info << ")\n";
            }
            else {
                tmp.count = 0;
            }
        }
        return index;
    }

     static int deleteElemTest(vector <BinSTree> & array, int index, Elem x, int & changeCount) { //удаление элемента по ключу
        BinSTree & tmp = array.at(index);
        if(tmp.count == 0)
            return index;
        if(x < tmp.info) {
```

```
                tmp.leftTree = deleteElemTest(array,tmp.leftTree, x,
changeCount );
        }
        else if(x > tmp.info) {
                tmp.rightTree = deleteElemTest(array,tmp.rightTree, x,
changeCount);
        }
        else if(tmp.rightTree != 0){
            changeCount++;
            tmp.info = array.at(getMinRight(array,tmp.rightTree)).info;
                tmp.rightTree = deleteElemTest(array, tmp.rightTree,
tmp.info, changeCount);
        }
        else{
            if(tmp.leftTree != 0){
                changeCount++;
                                tmp.info = array.at(getMaxLeft(array
,tmp.leftTree)).info;
                    tmp.leftTree = deleteElemTest(array ,tmp.leftTree,
tmp.info, changeCount);
            }
            else {
                tmp.count = 0;
            }
        }
        return index;
    }

     //функция удаления
      static void rotateR(vector <BinSTree> & array, int index){ //
правое вращение
        if(array.at(index).count == 0){
            return;
        }
        else{
            if(!array.at(index).leftTree)
                return;
```

```
            BinSTree newRoot = array.at(array.at(index).leftTree); // новый корень - левый сын старого корня (копия)
            BinSTree oldRoot = array.at(index); // делаем копию старого корня
              array.at(index).info = newRoot.info; // корень - начало массива, поэтому копируем туда данные из newRoot, которые не изменятся
            array.at(index).count = newRoot.count;
            array.at(index).leftTree = newRoot.leftTree;
               array.at(oldRoot.leftTree) = oldRoot; // старый корень будет правым сыном нового корня, в массиве он должен занять бывшее место нового корня
             array.at(index).rightTree = oldRoot.leftTree; // обновляем ссылку у нового корня о нахождении правого сына
                          array.at(array.at(index).rightTree).leftTree = newRoot.rightTree; // правый сын нового корня  теперь левый сын старого корня. Обновляем ссылку
        }
    }

      static void rotateL(vector <BinSTree> & array ,int index) { // левое вращение
        if(array.at(index).count == 0){
            return;
        }
        else{
            if(!array.at(index).rightTree)
                return;
             BinSTree newRoot = array.at(array.at(index).rightTree); // создаём копию будущего корня
                BinSTree oldRoot = array.at(index); // создаём копию старого корня
             array.at(index).info = newRoot.info; // мы условились, что начало массива - это корень - поэтому в него надо скопировать новые данные
              array.at(index).count = newRoot.count; // втч флаг того, есть там что-нибудь или нет
```

```
                array.at(index).rightTree = newRoot.rightTree; // новый корень имеет другого правого сына, поэтому ссылку на него тоже надо обновить
            array.at(oldRoot.rightTree) = oldRoot; // в качестве нового корня взяли правого сына старого корня, поэтому надо старый корень засунуть в массиве на бывшее место нового корня
              array.at(index).leftTree = oldRoot.rightTree; // и старый корень станет левым сыном нового корня, поэтому не забываем обновить ссылку
                        array.at(array.at(index).leftTree).rightTree  = newRoot.leftTree; // правый сын старого корня теперь бывший левый сын нового корня
        }
    }

     static void insInRoot(vector <BinSTree> & array, int index, Elem &x, ostream &out, int n) {

            if (array.at(index).count == 0 ) {
                array.at(index).info = x;
                array.at(index).count = 1;
            }
            else {
                if (x <  array.at(index).info ) {
                    if(array.at(index).leftTree == 0) {
                        printIndent(n, out);
                                out << "Левое поддерево корня (" << array.at(index).info <<  ") пусто. Вставка...\n";
                        BinSTree newRoot;
                        array.push_back(newRoot);
                                        array.at(index).leftTree  = (int)(array.size()-1);
                    }
                        printIndent(n, out);
                                out << "(" << x << ")" << " < (" << array.at(index).info << ")\n";
                          insInRoot(array, array.at(index).leftTree, x, out, n + 1);
```

```
                        printIndent(n, out);
                        out << "Правое вращение. Старый корень текущего
поддерева: (" << array.at(index).info << "). ";
                        rotateR(array, index);
                         out << "Новый корень текущего поддерева: (" <<
array.at(index).info << ")\n";
                }
                else if (x > array.at(index).info) {
                    if(array.at(index).rightTree == 0) {
                        printIndent(n, out);
                                 out << "Правое поддерево корня (" <<
array.at(index).info <<  ") пусто. Вставка...\n";
                        BinSTree newRoot;
                        array.push_back(newRoot);
                                            array.at(index).rightTree  =
(int)array.size()-1;
                    }
                        printIndent(n, out);
                                out << "(" << x << ")" << " > (" <<
array.at(index).info << ")\n";
                         insInRoot(array, array.at(index).rightTree, x,
out, n + 1);
                        printIndent(n, out);
                         out << "Левое вращение. Старый корень текущего
поддерева: (" << array.at(index).info << "). ";
                        rotateL(array, index);
                         out << "Новый корень текущего поддерева: (" <<
array.at(index).info << ")\n";
                }
                else {
                    array.at(index).count++;
                    printIndent(n, out);
                      out << "(" << array.at(0).info << ") уже является
корнем дерева\n";
                }
            }
    }
```

```
    static void insInRootTest(vector<BinSTree> & array, int index, Elem
& x, int & rotateCount) {

        if (array.at(index).count == 0 ) {
            array.at(index).info = x;
            array.at(index).count = 1;
        }
        else {
            if (x <  array.at(index).info ) {
                if(array.at(index).leftTree == 0) {
                    BinSTree newRoot;
                    array.push_back(newRoot);
                    array.at(index).leftTree = (int)(array.size()-1);
                }
                     insInRootTest(array, array.at(index).leftTree, x,
rotateCount );
                rotateR(array, index);
                rotateCount++;
            }
            else if (x > array.at(index).info) {
                if(array.at(index).rightTree == 0) {
                    BinSTree newRoot;
                    array.push_back(newRoot);
                    array.at(index).rightTree = (int)array.size()-1;
                }
                     insInRootTest(array, array.at(index).rightTree, x,
rotateCount);
                rotateL(array, index);
                rotateCount++;
            }
            else {
                array.at(index).count++;
            }
        }
    }

      static void printIndent(int n, std::ostream & out){ // печать
отступов (нужно в отладочных выводах)
```

```
        for(int i = 0; i < n; i++ )
            out << "\t";
    }

};

void manual(){
   cout << "ВНИМАНИЕ!!!: Для корректной работы программы дерево должно быть определённого размера. Далее (сложность -> размер):\n"
           "Для удаления: (лёгкая -> >= 2), (средняя -> >= 4)\n"
           "Для вставки: (>= 1 при любой сложности)\n"
           "Для вставки в корень: (лёгкая -> >= 1), (средняя -> >= 3)\n"
             "\nРегулировка сложности состоит в количестве производимых операций (поворот/замена)\n"
           "Для удаления: (лёгкая -> 1 замена), (средняя -> 3 замены)\n"
            "Для вставки в корень: (лёгкая -> 1 поворот), (средняя -> 3 поворота)\n\n"
              "Интерфейс, предоставляемый пользователю: (все команды и параметры обязательны (кроме --outfile <arg>, при его отсутствии вывод в консоль))\n"
           "--size <arg> / -s <arg> - размер БДП\n"
           "--compl <arg> / -c <arg> - сложность: <e/m>, e - лёгкая, m - средняя\n"
                        "--type <arg> / -t <arg> - тип задания: <insert/insInRoot/delete>, <вставка/вставка в корень/удаление> соответственно\n"
           "--outfile <arg> - путь до файла в качестве <arg>\n"
           "--help / -h - справка\n"
           "--num <arg> / -n <arg> - количество заданий\n\n"
             "Пример ввода: ./cw --size 3  --compl m  --type insInRoot --outfile /home/andrey/result --num 5\n\n";
}

void genBinSTree(vector<BinSTree> & array, vector<Elem> & elemsBDP, int lowerBorder, int upperBorder, int &size) { // генерация нового бдп и освобождение старого
        int elem;
```

```
            for (int i = 0; i < size; i++) { // заполнение случайной
последовательности случайными неодинаковыми числами
                    elem = rand() % (upperBorder - lowerBorder + 1) +
lowerBorder;
                   while(find(elemsBDP.begin(), elemsBDP.end(), elem) !=
elemsBDP.end())
                       elem = rand() % (upperBorder - lowerBorder + 1) +
lowerBorder;
            elemsBDP.push_back(elem);
        }

        for(int i = 0; i < size; i++){
            BinSTree::searchAndInsert(array, elemsBDP.at(i), 0);
        }
}

void CreateTree(vector<BinSTree> & array,vector<int> &copy) {
    for(int i : copy)
        BinSTree::searchAndInsert(array, i, 0);
}

int genNumberForInsert(vector<int> & elemsBDP , int lowerBorder, int
upperBorder ){ // генерирует число для вставки в корень с учётом
поданной последовательности
    int num = rand() % (upperBorder - lowerBorder + 1 ) + lowerBorder ;
             while(find(elemsBDP.begin(),   elemsBDP.end(),   num)   !=
elemsBDP.end()){
         num = rand() % (upperBorder - lowerBorder + 1  ) + lowerBorder
+ 1;
    }
    return num;
}

int getNumberToDel(vector<int> & elemsBDP ){
    int num = elemsBDP.at(rand () % (elemsBDP.size()));
    return num;
}
```

```
int getTaskInsInRoot(vector<BinSTree> & array,vector<Elem> & copy,int lowerBorder, int upperBorder, int &size, string & complex){
     /* функция, генерирующая необходимую последовательность и число, которое надо вставить, чтобы заданная сложность задания
     * вставки в корень была соблюдена */
    int numToIns = 0;
    int rotateCount = 0;
    int rotateCountNeeded = 0;
    if(complex == "e")
        rotateCountNeeded = 1;
    else if(complex == "m")
        rotateCountNeeded = 3;

    rotateCount = 0;
          while(rotateCount != rotateCountNeeded){ // генерирует такое дерево и такой элемент для вставки, чтобы получился ровно 1 или 3поворота
            array.clear();
            copy.clear();
            rotateCount = 0;
            genBinSTree( array,copy,lowerBorder, upperBorder, size);
                    numToIns = genNumberForInsert(copy, lowerBorder, upperBorder);
            BinSTree::insInRootTest(array , 0, numToIns, rotateCount );

        }
    return numToIns;
}
int getTaskDel(vector<BinSTree> & array,vector<Elem> & copy,int lowerBorder, int upperBorder, int &size, string & complex){
    int numToDel = 0;
    int changeCount = 0;
    int changeCountNeeded = 0;
    if(complex == "e")
        changeCountNeeded = 1;
    else if(complex == "m")
        changeCountNeeded = 3;
```

```
    while(changeCount != changeCountNeeded){ // генерирует такое дерево
и такой элемент для удаления, чтобы получилось 1 либо 3 замены
        array.clear();
        copy.clear();
        changeCount = 0;
        genBinSTree( array,copy,lowerBorder, upperBorder, size);
        numToDel = getNumberToDel(copy);
        BinSTree::deleteElemTest(array, 0, numToDel, changeCount);
    }
    return numToDel;

}



int main(int argc, char* argv[]) {
    srand(time(nullptr));
    string path = "$"; // путь до файла
    string elements; // входная строка
    opterr = 0;
    int res;
    int size = -1; // размер БДП
    string complex = "error"; // сложность
    int count = -1; // количество заданий
    string type = "error";
    int lowerBorder = 1; // нижние и верхние границы бдп
    int upperBorder = 1000;
    int fileFlag = 0;
    ofstream outFile;
    static struct option longopt[] = {
            {"size",    required_argument, nullptr, 's'},
            {"compl",   required_argument, nullptr, 'c'},
            {"type",    required_argument, nullptr, 't'},
            {"help",    no_argument,       nullptr, 'h'},
            {"num",     required_argument, nullptr, 'n'},
            {"outfile", required_argument, nullptr, 'f'},
            {nullptr, 0,nullptr, 0}
    };
```

```
    while ((res = getopt_long(argc, argv, SHORTOPTS, longopt, &optind))
!= -1) {

                if (strcmp(argv[optind - 2], "--size") != 0 &&
strcmp(argv[optind - 2], "-s") != 0 &&
                    strcmp(argv[optind - 2], "--compl") != 0 &&
strcmp(argv[optind - 2], "-c") != 0 &&
                       strcmp(argv[optind - 2], "--type") != 0 &&
strcmp(argv[optind - 2], "-t") != 0 &&
                   strcmp(argv[optind - 2], "--outfile") != 0 &&
strcmp(argv[optind - 2], "-f") != 0 &&
                       strcmp(argv[optind - 1], "--help") != 0 &&
strcmp(argv[optind - 1], "-h") != 0 &&
                       strcmp(argv[optind - 2], "--num") != 0 &&
strcmp(argv[optind - 2], "-n") != 0)
        {
            cerr << "Некорректный ввод\n";
            exit(1);
        }
        switch (res) {
            case 's':
                size = atoi(optarg);
                if(size <= 0)
                {
                    cerr << "Некорректный размер\n";
                    exit(1);
                }
                break;
            case 'c':
                complex = optarg;
                if(complex != "e" && complex != "m")
                {
                    cerr << "Некорректное значение сложности\n";
                    exit(1);
                }
                break;
            case 't':
```

```
                type = optarg;
                    if(type != "delete" && type != "insert" && type !=
"insInRoot")
                {
                    cerr << "Некорректный тип задания\n";
                    exit(1);
                }
                break;
            case 'f':
                path = optarg;
                outFile.open(path);
                if(!outFile.is_open())
                          cerr << "Некорректный путь до файла. Вывод в
консоль\n";
                else
                    fileFlag = 1;
                break;
            case 'n':
                count = atoi(optarg);
                if(count <= 0){
                    cerr << "Некорректное количество заданий\n";
                    exit(1);
                }
                break;
            case 'h':
                manual();
                break;
            default:
                break;
        }

    }
         ostream &out = fileFlag ? outFile : cout; // открыть поток на
вывод

        vector<int> listOfElems;
        vector<BinSTree> array;
```

```
        Elem elemToIns;
        Elem elemToDel;


    if(type == "insInRoot"){
          if((complex == "e" && size < 1) || (complex == "m" && size <
3)){
            cerr << "Слишком маленький размер БДП для данного задания с
данной сложностью\n";
            exit(1);
        }
        for(int i = 0; i < count; i++){
                    elemToIns = getTaskInsInRoot(array, listOfElems,
lowerBorder, upperBorder, size, complex);
            out << i + 1 << ". Вставьте элемент " << elemToIns << " в
корень бинарного дерева размера " << listOfElems.size() << ": \n";
                array.clear(); // массив, представляющий собой бдп,
необходимо перезаписать
                CreateTree(array, listOfElems); // создать дерево по
заданной последовательности
           BinSTree::printBST(array, 0, 0, out);
           out << "\nОТВЕТ:\n";
           BinSTree::insInRoot(array, 0, elemToIns,  out, 0);
           BinSTree::printBST(array, 0, 0, out);
           out << endl;
        }
    }
    else if(type == "delete"){
          if((complex == "e" && size < 2) || (complex == "m" && size <
4)){
            cerr << "Слишком маленький размер БДП для данного задания с
данной сложностью\n";
            exit(1);
        }
        for(int i = 0; i < count; i++) {
               elemToDel = getTaskDel(array, listOfElems, lowerBorder,
upperBorder, size, complex);
```

```
            out << i + 1 << ". Удалите элемент " << elemToDel << " из
бинарного дерева размера " << listOfElems.size() << ": \n";
                array.clear(); // массив, представляющий собой бдп,
необходимо перезаписать
                CreateTree(array, listOfElems); // создать дерево по
заданной последовательности
            BinSTree::printBST(array, 0, 0, out);
            out << "\nOTBET:\n";
            BinSTree::deleteElem(array, 0, elemToDel,  out, 0);
            BinSTree::printBST(array, 0, 0, out);
            out << endl;
        }
    }
    else if(type == "insert") { // все задания на обычную вставку (не в
корень) имеют сложность - легкое вследствие простоты самой идеи
         out << "Замечание: сложность данного типа задания лёгкая вне
зависимости от выбора пользователя вследствие простоты идеи.\n";
        for(int i = 0; i < count; i++) {
            array.clear();
                listOfElems.clear(); // список элементов и бдп надо
перезаписывать
             genBinSTree(array, listOfElems, lowerBorder, upperBorder,
size);
              elemToIns = genNumberForInsert(listOfElems, lowerBorder,
upperBorder);
             out << i + 1 << ". Вставьте элемент " << elemToIns << " в
бинарное дерево размера " << listOfElems.size() << ": \n";
            BinSTree::printBST(array, 0, 0, out);
            out << "\nOTBET:\n";
            BinSTree::searchAndInsert(array, elemToIns, 0);
            BinSTree::printBST(array, 0, 0, out);
            out << endl;
        }
    }

    listOfElems.clear();
    array.clear();
    return 0;
```

```
}
```