МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания 4

Тема: Нелинейные структуры данных. Сбалансированные деревья поиска (СДП) и их применение для поиска данных в файле.

Дисциплина Структуры и алгоритмы обработки данных

Выполнил студент Самойленко М. А.

группа ИНБО-02-20

Москва 2021

Цель:

- получить навыки в разработки и реализации алгоритмов управления бинарным деревом поиска и сбалансированными бинарными деревьями поиска (АВЛ – деревьями);

- получить навыки в применении файловых потоков прямого доступа к данным файла;

- получить навыки в применении сбалансированного дерева поиска для прямого доступа к записям файла.

Задание 1:

Разработать приложение, которое использует бинарное дерево поиска (БДП) для поиска записи с ключом в файле, структура которого представлена в задании 2 вашего варианта (**вариант 9**).

1. Разработать класс «Бинарное дерево поиска». Тип информационной части узла ключ и ссылка на запись в файле (как в практическом задании 2). Методы: включение элемента в дерево, поиск ключа в дереве, удаление ключа из дерева, отображение дерева.

2. Разработать класс управления файлом (если не создали в практическом задании 2). Включить методы: создание двоичного файла записей фиксированной длины из заранее подготовленных данных в текстовом файле; поиск записи в файле с использованием БДП; остальные методы по вашему усмотрению.

3. Разработать и протестировать приложение.

4. Подготовить отчет.

Постановка задачи:

Разработайте приложение, которое использует БДП для организации прямого доступа к записям файла, структура записи которого приведена в варианте.

Структура записи:

- Страховой номер;
- Название компании;

- Фамилия владельца.

Подход к решению:

1) Бинарное дерево поиска – класс BinTreeSearch. В нем определены методы:

- insert(Knot** k, int key, long numInFile) – метод вставки узла в дерево поиска;
- knotDelete(Knot** k, int key)– метод удаления узла бинарного дерева поиска;
- find(Knot* root, int key) – метод поиска узла по его ключу, возвращает указатель на этот узел;
- printTree(Knot** key, int l) – вывод бинарного дерева поиска в консоль;
- createTree(int n, Knot** root) – метод создания бинарного дерева поиска;

Также есть три вспомогательных метода: getRoot (возвращает указатель на корень дерева), searchParent(возвращает указатель на родителя узла), updateRoot(обновление корня дерева, если он будет удален).

2) Узел бинарного дерева – структура Knot, в которой определены поля: указатель на левое поддерево, указатель на правое поддерево, номер полиса (ключ, по которому осуществляется поиск), номер записи в файле.

3) Запись в файле – класс Polis. В нем хранятся поля: Номер полиса, название компании, фамилия владельца, номер записи в файле.

4) Бинарный файл состоит из записей фиксированного размера.

Структура записи файла:

- В бинарный файл поступает объект класса Polis.

Операции по управлению файлом:

- Добавить запись в файл – метод insertInFile();
- Удалить запись из файла – метод deletePolisInBinTxt();
- поиск записи в файле с использованием БДП – метод findInFile();

- создание двоичного файла записей фиксированной длины из заранее подготовленных данных в текстовом файле (данные генерируются случайным образом в файле Polisa.txt) – метод genFile();

Алгоритмы операций:

Алгоритм вставки в БДП элемента:

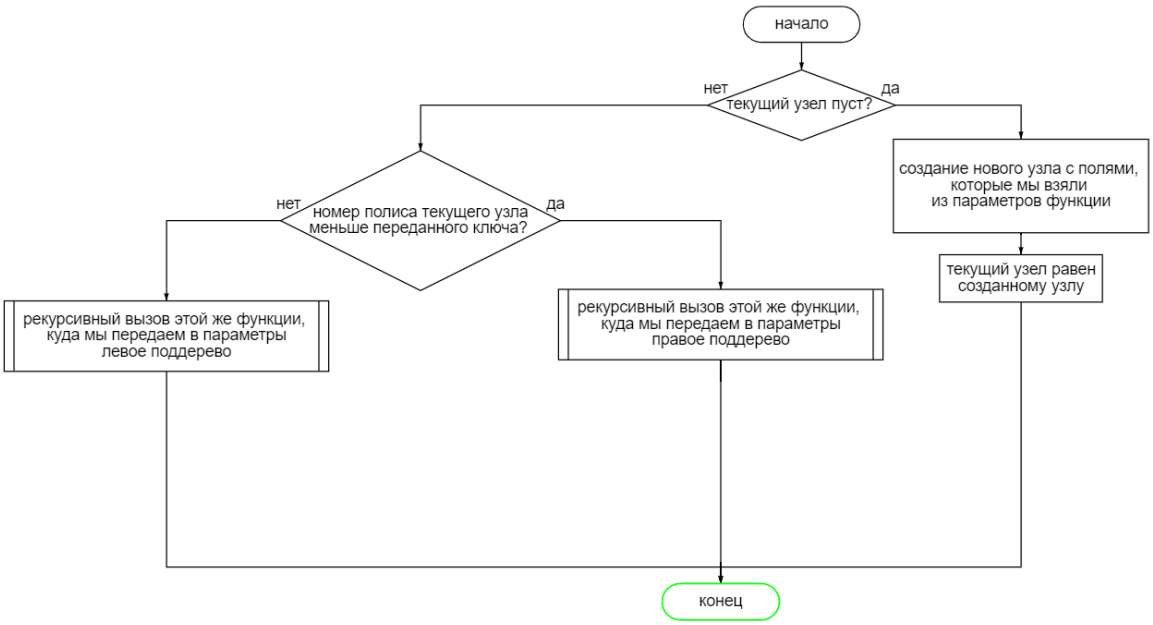


Рисунок 1 – блок-схема алгоритма вставки в БДП

Алгоритм поиска записи по ключу в БДП и возвращение найденной записи:

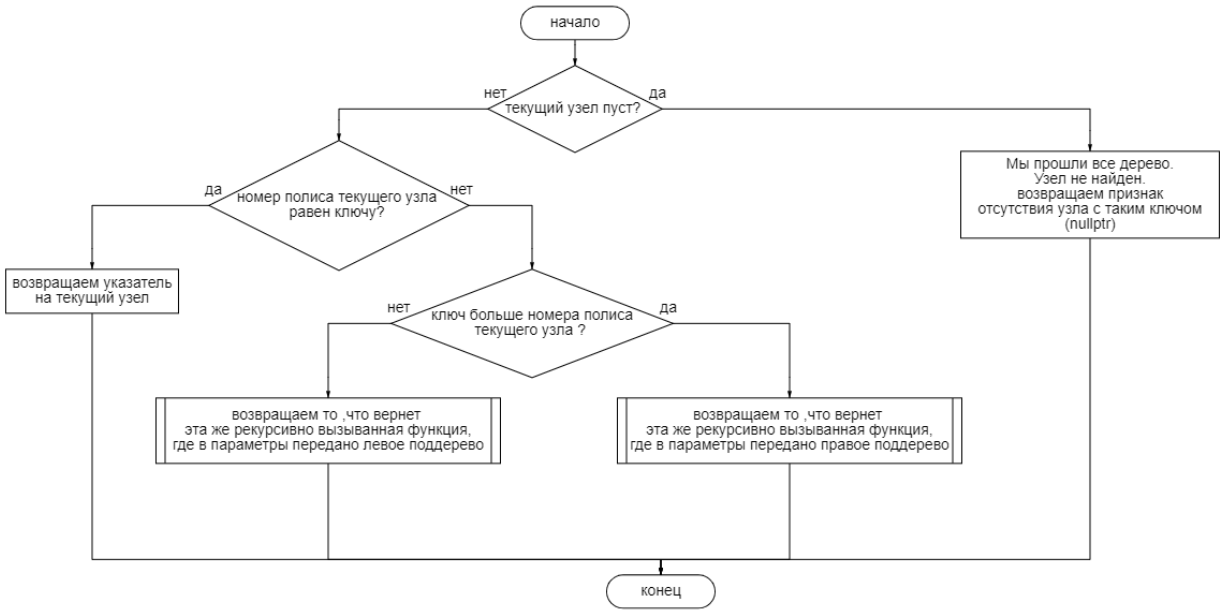


Рисунок 2 – блок-схема алгоритма поиска в БДП

Алгоритм удаления элемента в БДП:

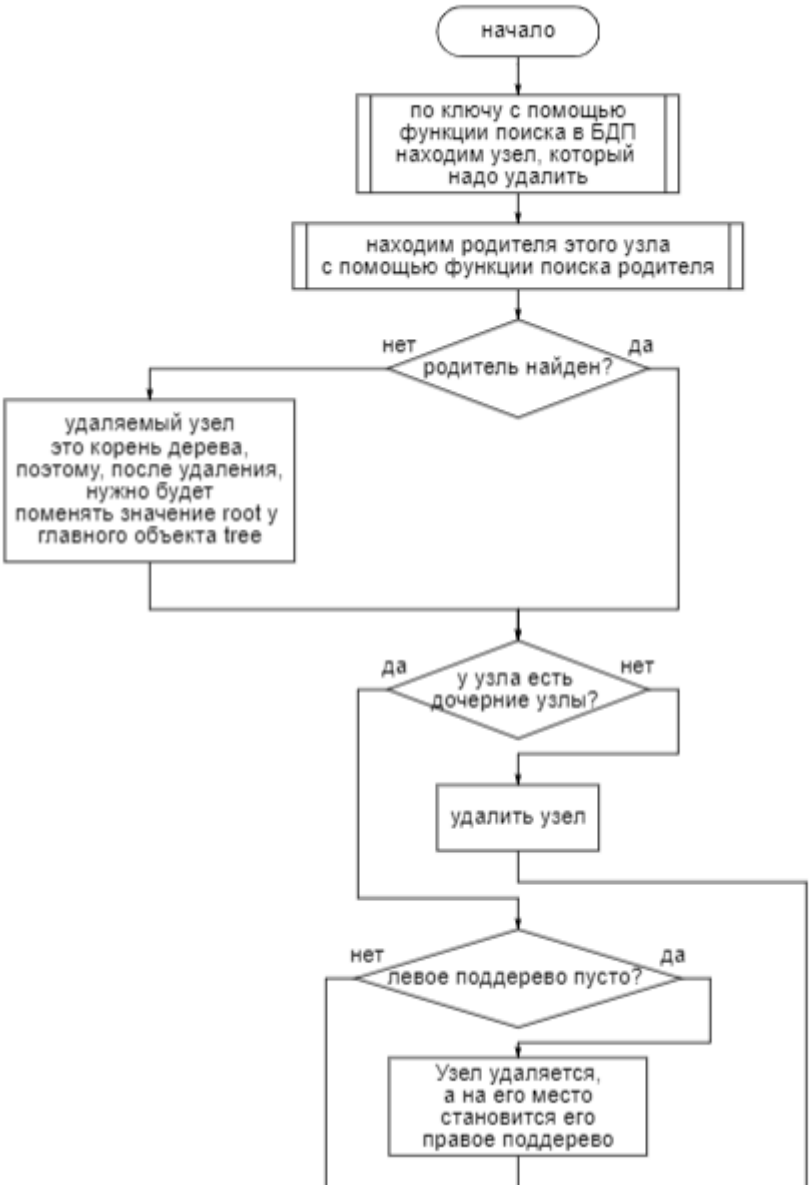


Рисунок 3 – блок-схема алгоритма удаления в БДП. Часть 1

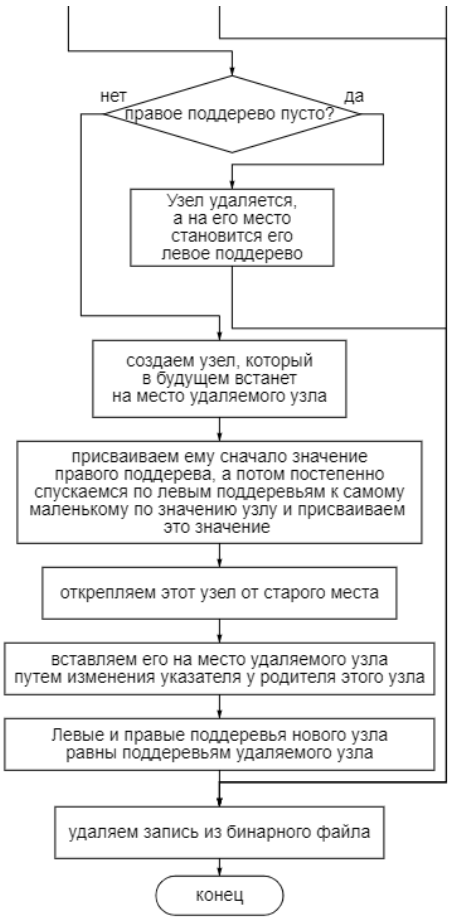


Рисунок 4 – блок-схема алгоритма удаления в БДП. Часть 2

Код приложения: в СДО МИРЭА вместе с отчетом.

Содержание файла:

Текстовый файл генерируется каждый раз случайно, но строки и данные всегда определенной длины.

Строка в текстовом файле:

Номер полиса (3 символа) + «\» + название компании (4 символа) + «\» + Фамилия владельца (6 символов) + «\»

Данные из текстового файла Polisa.txt записываются в файл PolisaBIN.dat, оттуда БДП берет данные для своей работы.

Тестирование:

В БДП будет 20 элементов.

1. Добавление элемента в дерево:

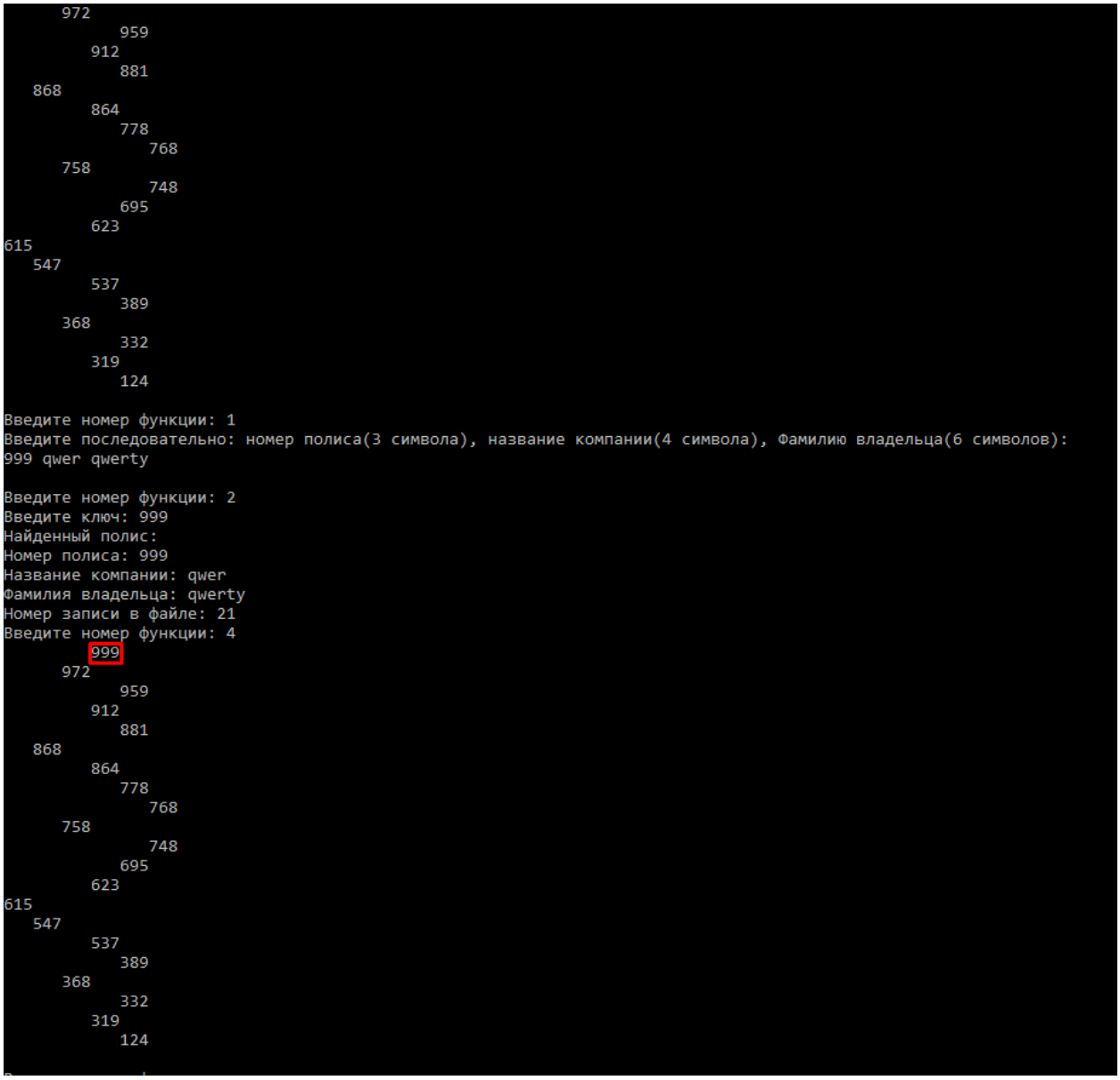


Рисунок 5 – добавление элемента в дерево

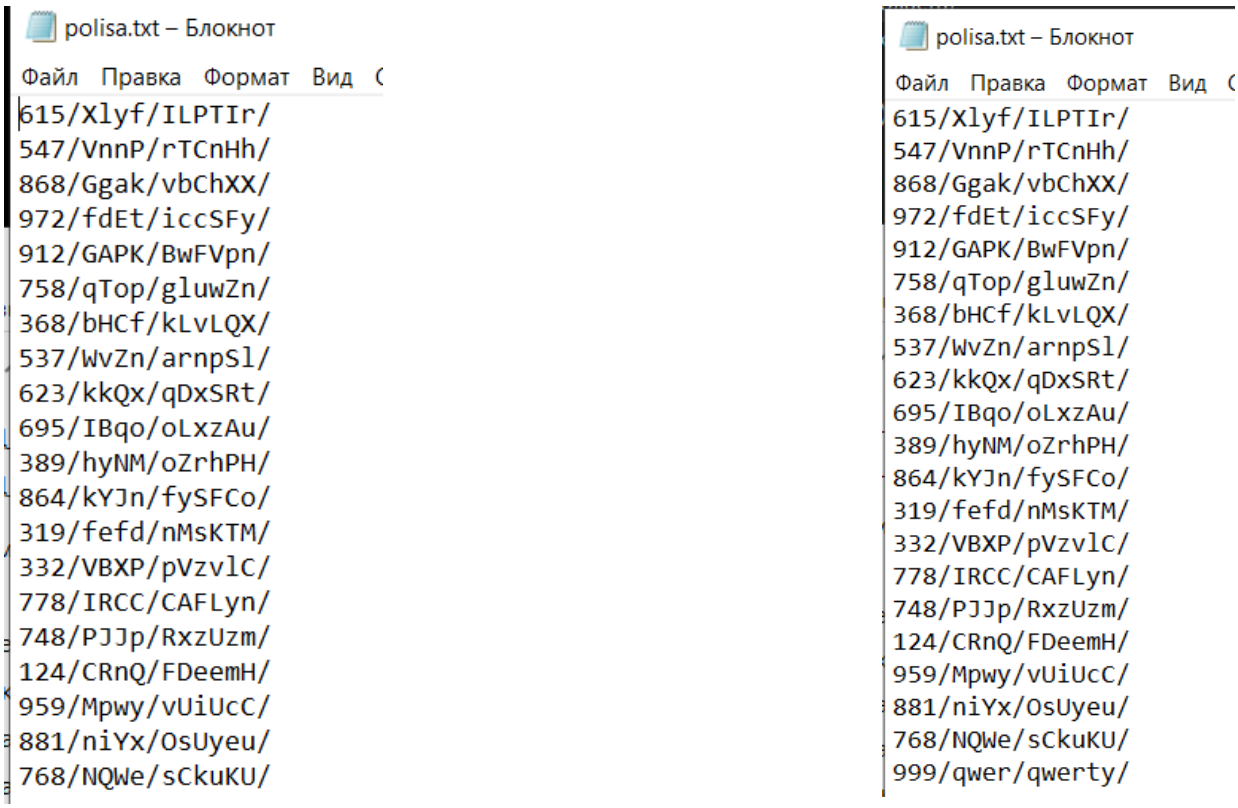


Рисунок 6 – Файл до и после добавления

2. Поиск ключа в дереве:

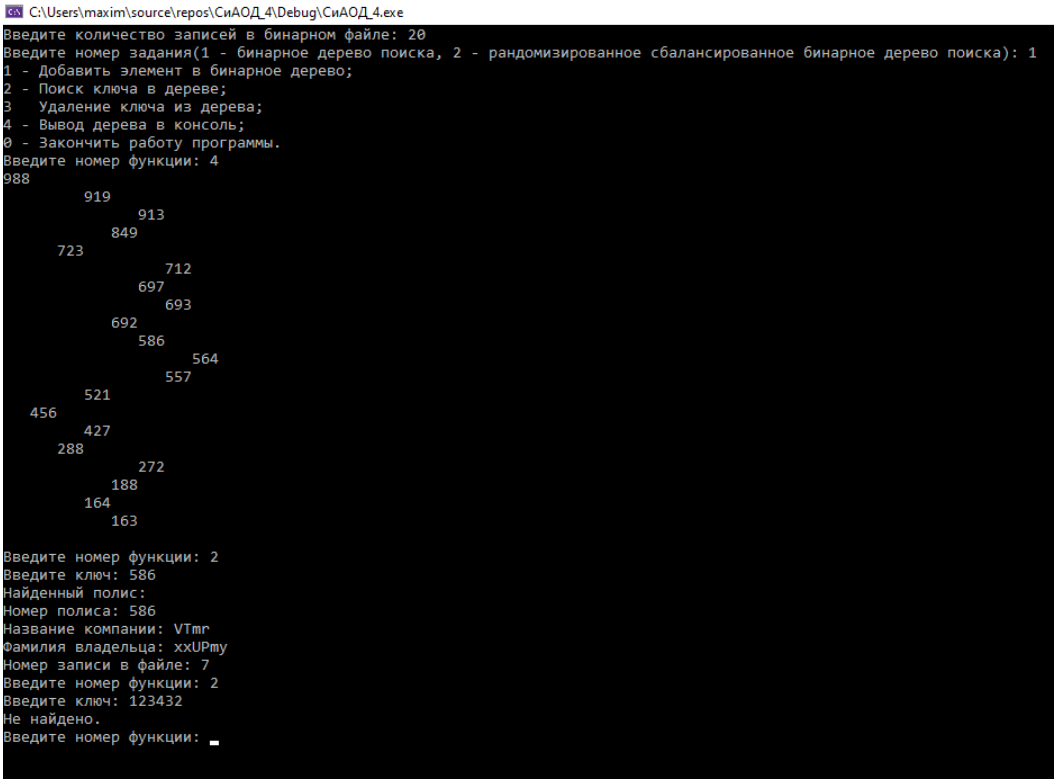


Рисунок 7-8 – поиск в БДП

3. Удаление ключа из дерева: будут удалены 4 ключа – 4 разные случая расположения узлов.

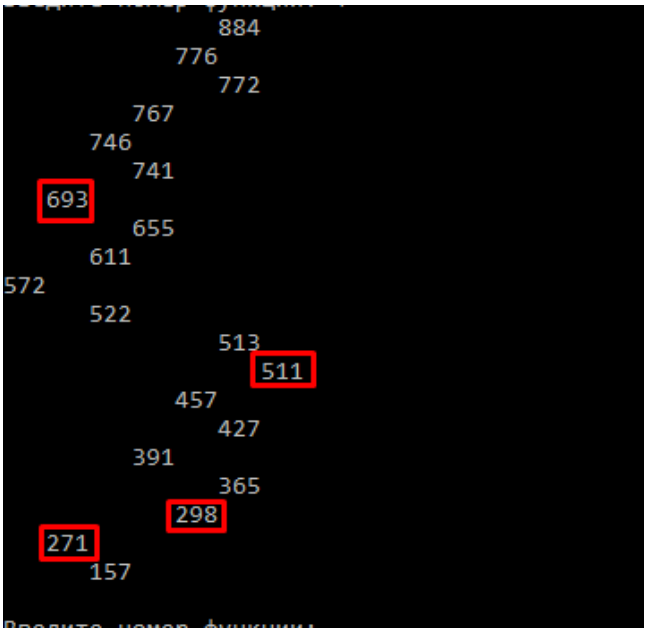


Рисунок 10 – Начальное дерево

Удалим узлы 511, 298, 271, 693.

```
Введите номер функции: 3
Введите ключ: 511
Удаление прошло успешно.
Введите номер функции: 3
Введите ключ: 298
Удаление прошло успешно.
Введите номер функции: 3
Введите ключ: 271
Удаление прошло успешно.
Введите номер функции: 3
Введите ключ: 693
Удаление прошло успешно.
Введите номер функции: 3
Введите ключ: 1342141
Такого узла нет.
Введите номер функции: 4
                884
           776
                772
         767
      746
   741
         655
      611
572
      522
                513
           457
                427
         391
   365
      157
```

Рисунок 11 – дерево после удаления

Рисунок 12 – файл после удаления

Задание 2:

Разработать приложение, которое использует сбалансированное дерево поиска, предложенное в варианте, для доступа к записям файла.

1. Разработать класс СДП с учетом дерева варианта. Структура информационной части узла дерева включает ключ и ссылку на запись в файле (адрес места размещения). Основные методы: включение элемента в дерево; поиск ключа в дереве с возвратом ссылки; удаление ключа из дерева; вывод дерева в форме дерева (с отображением структуры дерева).

2. Разработать приложение, которое создает и управляет СДП в соответствии с заданием.

3. Выполнить тестирование.

4. Определить среднее число выполненных поворотов (число поворотов на общее число вставленных ключей) при включении ключей в дерево при формировании дерева из двоичного файла.

5. Оформить отчет

Постановка задачи:

Разработайте приложение, которое использует СДП для организации прямого доступа к записям файла, структура записи которого приведена в варианте (**вариант 9**).

Структура записи:

- Страховой номер;
- Название компании;
- Фамилия владельца.

Вариант 9:

№	Сбалансированное дерево поиска (СДП)	Структура элемента множества(ключ-подчеркнутое поле) остальные поля представляют данные элемента
9	Рандомизированное	Страховой полис: номер, компания, фамилия владельца

Подход к решению:

Рандомизированное сбалансированное дерево поиска – отличается от обычного дерева поиска, тем, что с вероятность 1/n узел при вставке в дерево может быть корнем. Поэтому кроме обычной вставки, нам понадобятся методы поворотов дерева вокруг определенного узла и так же метод вставки узла в корень. Также каждый узел будет иметь дополнительное поле size – размер дерева с корнем в данном узле.

1) Рандомизированное сбалансированное дерево поиска – класс RandomTreeSearch. В нем определены методы:

- insert(Knot** k, int key, long numInFile) – метод вставки узла в дерево;
- knotDelete(Knot** k, int key)– метод удаления узла;
- find(Knot* root, int key) – метод поиска узла по его ключу, возвращает указатель на этот узел;
- printTree(Knot** key, int l) – вывод сбалансированного дерева поиска в консоль;
- createTree(int n, Knot** root) – метод создания сбалансированного дерева поиска;
- rotateLeft(Knot* k) – метод поворота налево относительно узла k;
- rotateRight(Knot* k) – метод поворота направо относительно узла k;
- insertRoot(Knot** k, int key, long numInFile) – метод вставки в корень дерева;
- getSize(Knot* k) – метод, возвращающий размер дерева с корнем k;

- fixSize(Knot* k) – метод, переопределяющий размер дерева с корнем k;
- join(Knot* small, Knot* big) – метод, соединяющей деревья с корнями small и big, при этом каждая вершина дерева с корнем small меньше, чем любая вершина дерева с корнем big. Каждый из указателей может стать корнем нового дерева с вероятностью: размер этого дерева/(размер этого дерева + размер другого дерева).

2) Узел сбалансированного дерева – структура Knot, в которой определены поля: указатель на левое поддерево, указатель на правое поддерево, номер полиса (ключ, по которому осуществляется поиск), номер записи в файле, размер дерева с корнем в данном узле.

3) Запись в файле – класс Polis. В нем хранятся поля: Номер полиса, название компании, фамилия владельца, номер записи в файле.

4) Бинарный файл состоит из записей фиксированного размера.

Структура записи файла:

- В бинарный файл поступает объект класса Polis.

Операции по управлению файлом:

- Добавить запись в файл – метод insertInFile();
- Удалить запись из файла – метод deletePolisInBinTxt();
- поиск записи в файле с использованием СДП – метод findInFile();
- создание двоичного файла записей фиксированной длины из заранее подготовленных данных в текстовом файле (данные генерируются случайным образом в файле Polisa.txt) – метод genFile();

Алгоритмы операций:

Алгоритм вставки в СДП элемента осуществляется работой 4-х важных функция. Рассмотрим их поочередно:

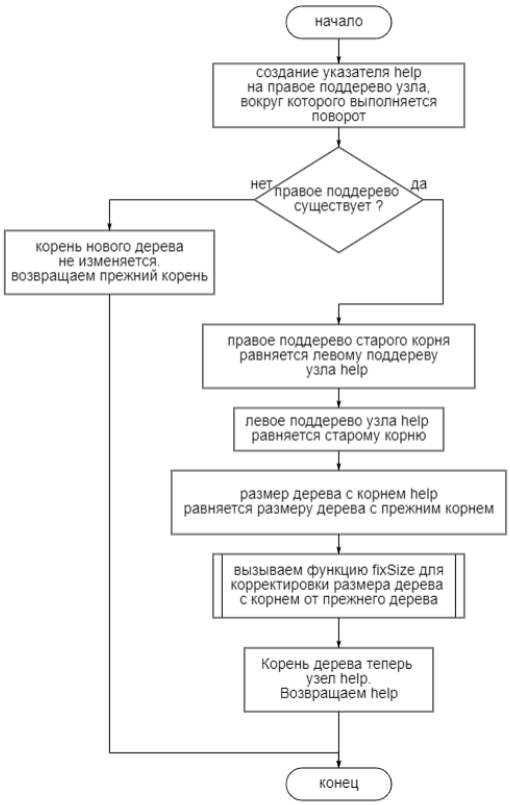


Рисунок 13 – Блок-схема алгоритма поворота налево

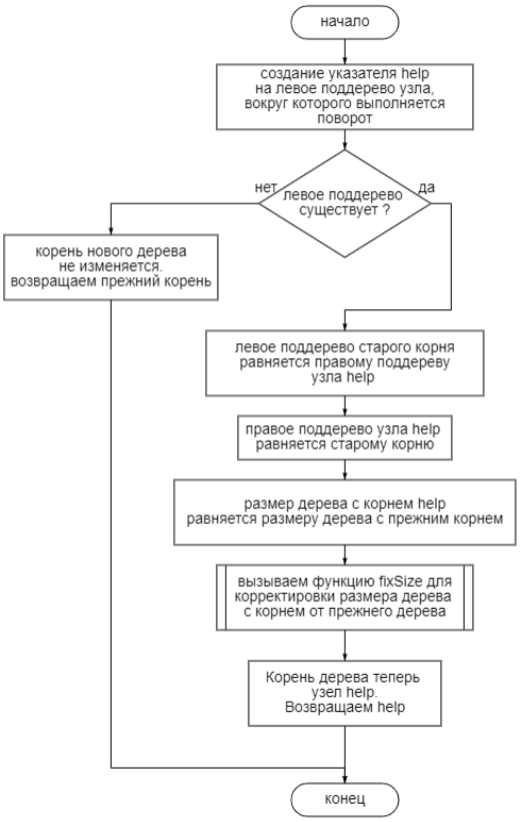


Рисунок 14 – Блок-схема алгоритма поворота направо

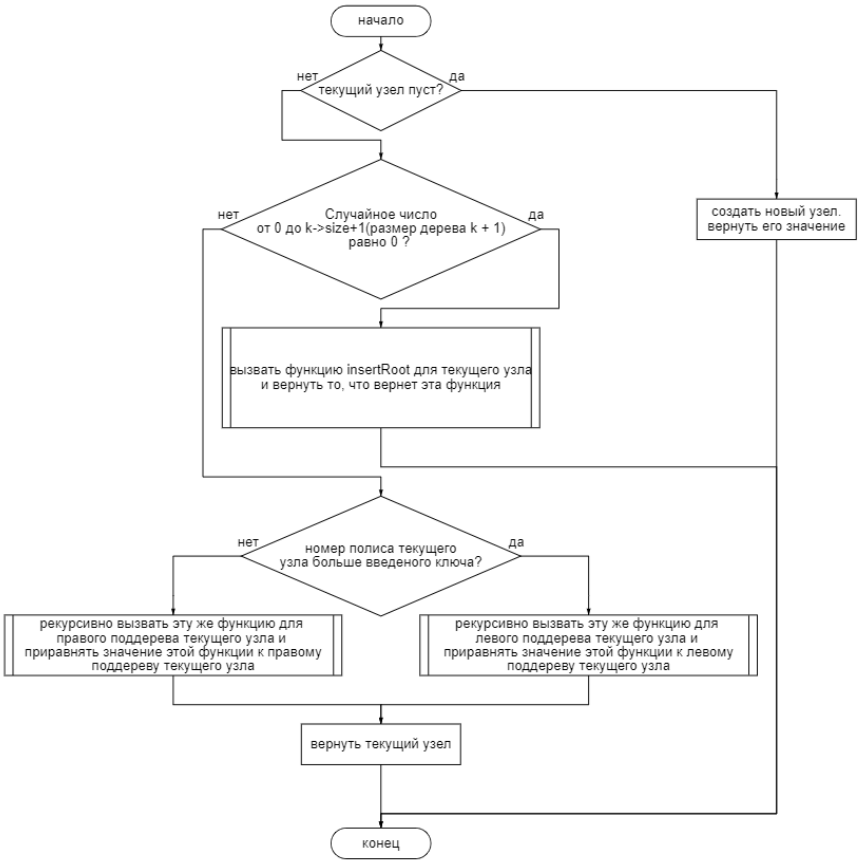


Рисунок 15 – Блок-схема алгоритма вставки

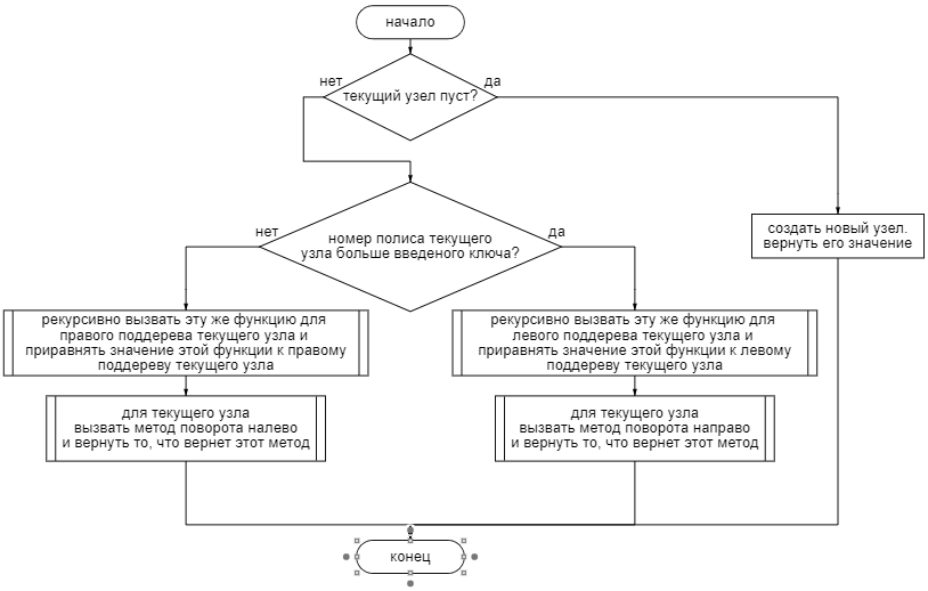


Рисунок 16 – Блок-схема алгоритма вставки в корень

Алгоритм поиска записи по ключу в СДП и возвращение найденной записи:

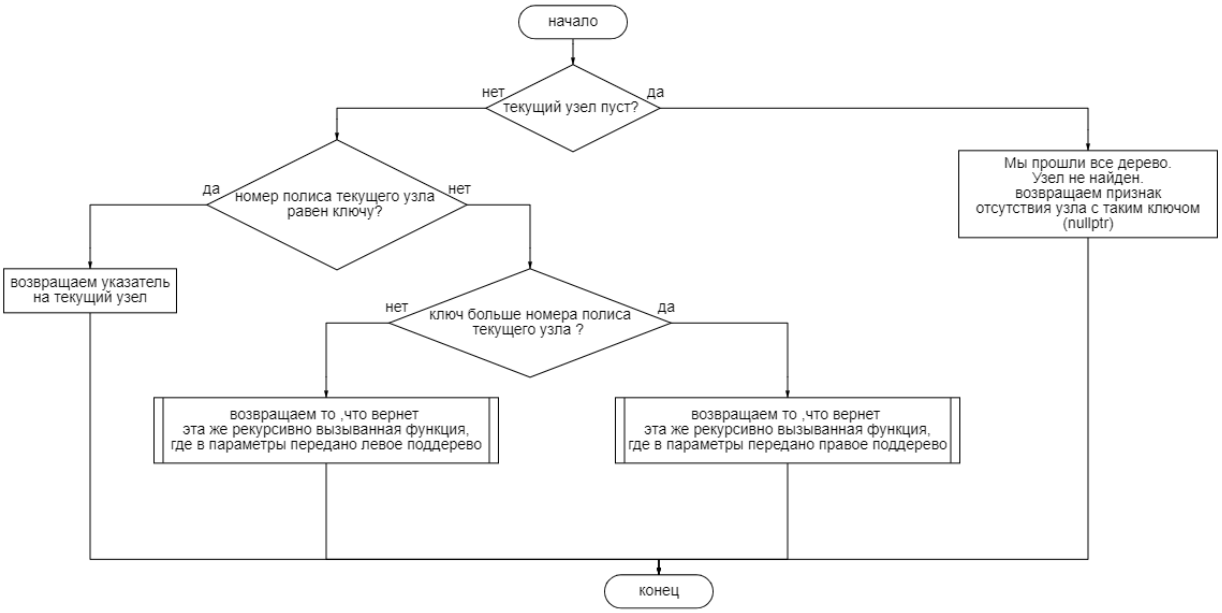


Рисунок 17 – Блок-схема алгоритма поиска в СДП

Алгоритм удаления элемента в СДП:

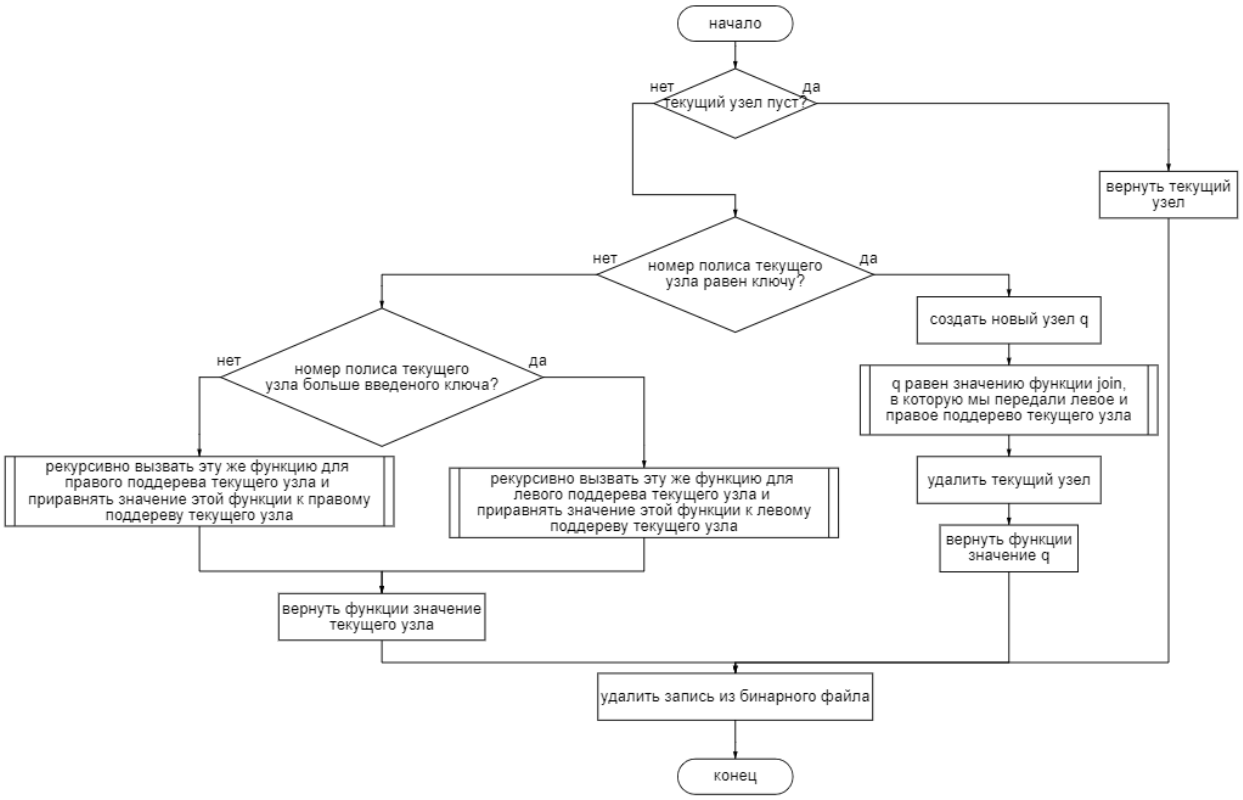


Рисунок 18 – Блок-схема алгоритма удаления элемента

Код приложения: в СДО МИРЭА вместе с отчетом.

Содержание файла:

Текстовый файл генерируется каждый раз случайно, но строки и данные всегда определенной длины.

Строка в текстовом файле:

Номер полиса (3 символа) + «\» + название компании (4 символа) + «\» + Фамилия владельца (6 символов) + «\»

Данные из текстового файла Polisa.txt записываются в файл PolisaBIN.dat, оттуда СДП берет данные для своей работы.

Среднее число выполненных поворотов:

Определить среднее число выполненных поворотов (число поворотов на общее число вставленных ключей) при включении ключей в дерево при формировании дерева из двоичного файла:

Сделаем 3 теста для 100,10 000,100 000 элементов:

```
Введите количество записей в бинарном файле: 100
Введите номер задания(1 - бинарное дерево поиска, 2 - рандомизированное сбалансированное бинарное дерево поиска): 2
Число вставленных ключей: 100
Количество поворотов: 206
Среднее число выполненных поворотов: 2.06
```

Рисунок 19 – среднее число поворотов для 100 узлов

```
Введите количество записей в бинарном файле: 10000
Введите номер задания(1 - бинарное дерево поиска, 2 - рандомизированное сбалансированное бинарное дерево поиска): 2
Число вставленных ключей: 10000
Количество поворотов: 28848
Среднее число выполненных поворотов: 2.8848
```

Рисунок 20 – среднее число поворотов для 10 000 узлов

```
Введите количество записей в бинарном файле: 100000
Введите номер задания(1 - бинарное дерево поиска, 2 - рандомизированное сбалансированное бинарное дерево поиска): 2
Число вставленных ключей: 100000
Количество поворотов: 314076
Среднее число выполненных поворотов: 3.14076
```

Рисунок 21 – среднее число поворотов для 100 000 узлов

Из тестов видно, что, при количестве элементов от 100-100000, на один вставленный узел приходится в среднем от 2 до 3 поворотов.

Тестирование:

В СДП будет 20 элементов.

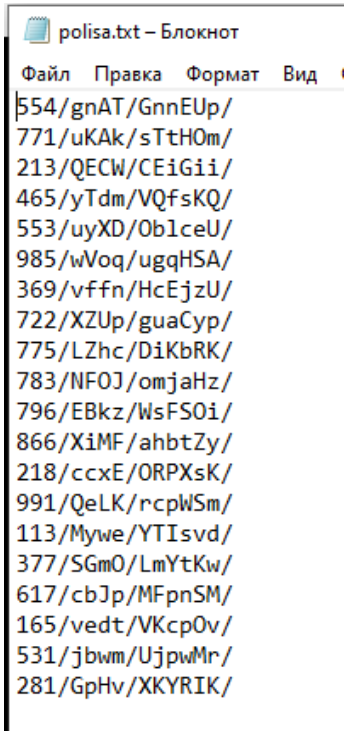
1. Добавление элемента в дерево:

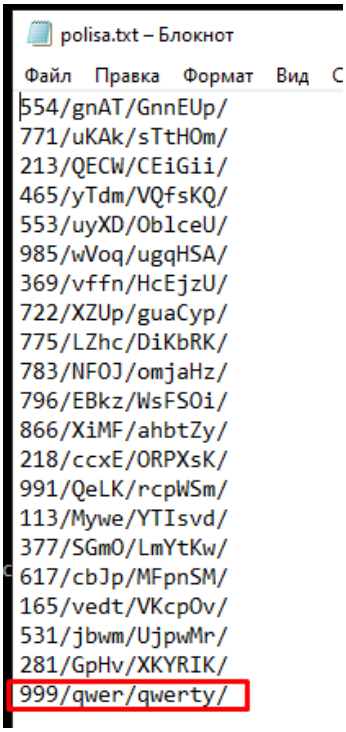
```
  991
    985
      866
        796
783
          775
        771
    722
      617
          554
        553
  531
      465
        377
              369
            281
          218
            213
    165
      113

Введите номер функции: 1
Введите последовательно: номер полиса(3 символа), название компании(4 символа), Фамилию владельца(6 символов):
999 qwer qwerty

Введите номер функции: 2
Введите ключ: 999
Найденный полис:
Номер полиса: 999
Название компании: qwer
Фамилия владельца: qwerty
Номер записи в файле: 21
Введите номер функции: 4
  999
    991
      985
        866
          796
783
          775
        771
    722
      617
          554
        553
  531
      465
        377
              369
            281
          218
            213
    165
      113
```

Рисунок 22 – добавление элемента в дерево





Рисунки 22-23 – Файл до и после добавления

2. Поиск ключа в дереве:

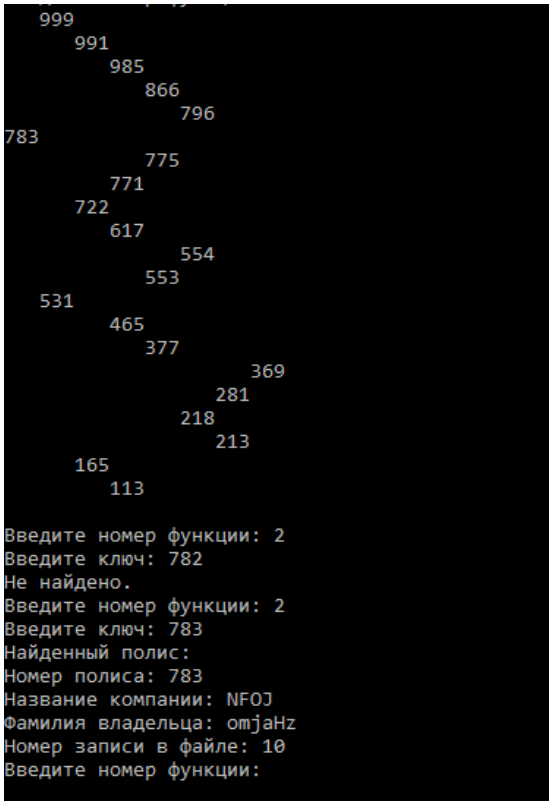


Рисунок 24 – Поиск элемента в дереве

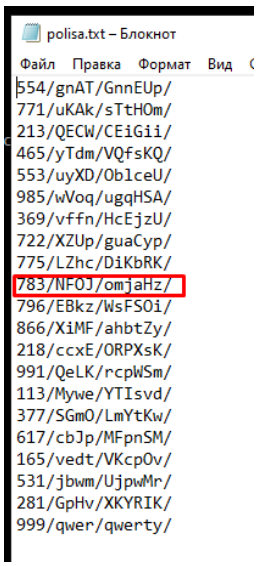


Рисунок 25 – поиск в БДП

3. Удаление ключа из дерева: будут удалены 4 ключа – 4 разные случая расположения узлов.

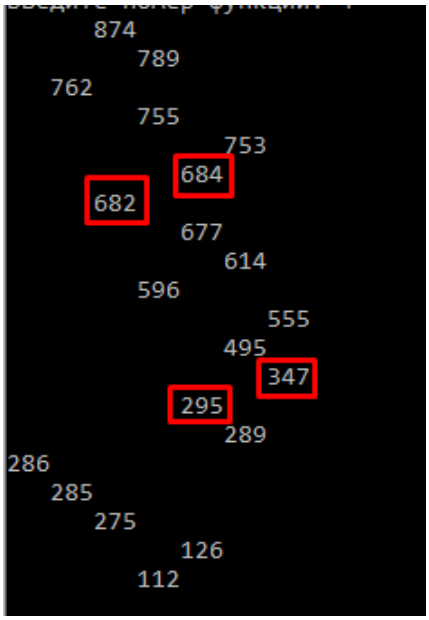


Рисунок 26 – Начальное дерево

Удалим узлы 684,682,295,347.

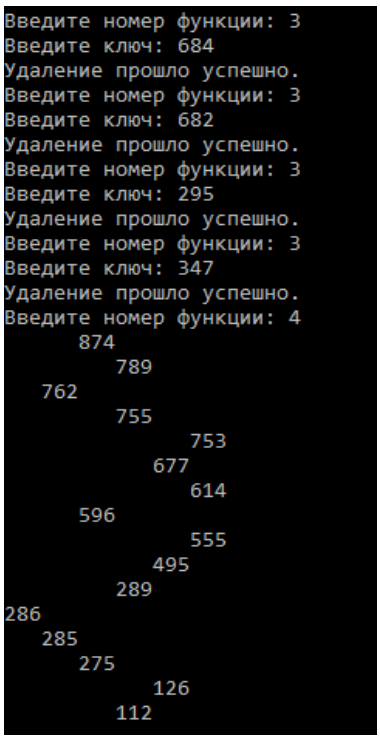


Рисунок 27 – дерево после удаления

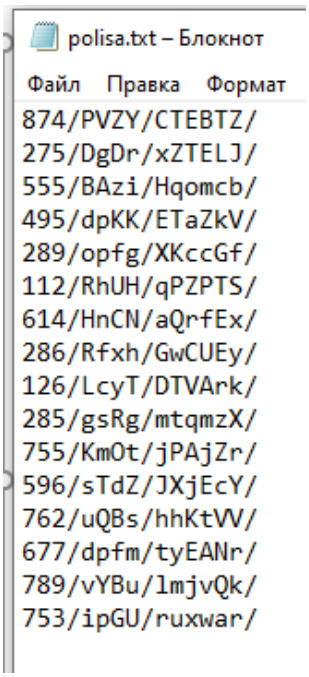


Рисунок 28– файл после удаления

Задание 3:

Осуществить поиск введенного целого числа в двоичном дереве поиска, в сбалансированном дереве и в хеш-таблице. Оформить таблицу результатов.

Подход к решению:

Тесты будем делать на 1000, 100 000, 1 000 000 элементов. Поиск осуществляется разными методами, но с одинаковыми файлами, чтобы точно сравнить работы поисковых структур.

На 1000 элементов ищем полис с номером 7678266, 936 место в файле;

На 100 000 элементов ищем полис с номером 239, 60822 место в файле;

На 1 000 000 элементов ищем полис с номером 9199618, 945205 место в файле.

Вид поисковой структуры	Кол-во элементов, загруженных в структуру в момент выполнения поиска	Емкостная сложность: объем памяти для структуры	Кол-во выполненных сравнений, время на поиск ключа в структуре
Хеш-таблица(цепное хеширование)	1000	22 килобайта	Сравнений: 1 Время: 0.004
Хеш-таблица(цепное хеширование)	100 000	2147 килобайта	Сравнений: 1 Время: 0.004
Хеш-таблица(цепное хеширование)	1 000 000	21485 килобайта	Сравнений: 1 Время: 0.005
БДП	1000	22 килобайта	Сравнений: 29 Время: 0.007
БДП	100 000	2147 килобайта	Сравнений: 43 Время: 0.005
БДП	1 000 000	21485 килобайта	Сравнений: 57 Время: 0.007
СДП(Рандомизированное)	1000	22 килобайта	Сравнений: 23 Время: 0.005
СДП(Рандомизированное)	100 000	2147 килобайта	Сравнений: 33 Время: 0.005

СДП(Рандомизированное)	1 000 000	21485 килобайта	Сравнений: 53 Время: 0.005

Выводы по таблице:

У хеш-таблицы с цепным хешированием наименьшее время поиска и наименьшее количество сравнений, вне зависимости от объема данных. Такая быстрота обусловлена хорошей работой хеш-функции, которая позволила равномерно распределить записи по хеш-таблице, что дало возможность находить каждый элемент за O(1). Сравнивая БДП и СДП видно, что БДП работало чуть медленнее и имела больше сравнений, т.к. у СДП высота дерева меньше, чем у БДП, что позволяло быстрее добираться до нужного элемента.

Выводы

В ходе практической работы были получены навыки по использованию и построению сбалансированного бинарного дерева поиск и обычного бинарного дерева поиска. Были реализованы функции: создания бинарного дерева поиска, вывод бинарного дерева, удаления элементов бинарного дерева, поиск элементов в дереве.

Список информационных источников:

1. Лекции по дисциплине «Структуры и алгоритмы обработки данных» / Л. А. Скворцова, МИРЭА – Российский технологический университет, 2021.