МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

"МИРЭА - Российский технологический университет"

РТУ МИРЭА

Отчет по выполнению практического задания №4

По дисциплине «Структуры и алгоритмы обработки данных»

Тема:

Сбалансированные деревья поиска (СДП) и их применение для поиска данных в файле

Выполнил студент Сидоров С.Д.

группа ИКБО-20-21

Отчёт по заданию 1

1. Постановка задачи:

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

Вариант 21: Рейтинг студентов: ФИО, средний балл, факт присутствия в списках на отчисление.

Дано:

Файл двоичной с записями фиксированной длины.

Структура записи файла:

string double bool

Результат:

Приложение выполняющее операции: удаления из БДП, добавления в БДП, чтение элементов из файла в БДП, поиск в файле с помощью БДП.

2. Подход к решению:

1) БДП - класс

2) Структура элемента БДП: ключ, позиция в файле, правое подерево, левое поддерево.

3) Методы класса БДП: добавление записи, поиск записи, удаление записи, вывод дерева.

3. Операции по управлению БДП:

1) Добавление элемента:

void addNode(long long int iKey, int iRowNumber);

2) Поиск элемента:

int findNode(long long int iKey);

3) Удаление записи:

void deleteNode(long long int iKey);

4) Вывод дерева:

void printExecute(Node* root, int level);

4. Алгоритмы операций на псевдокоде:

1) Вставка в БДП элемента :

```
метод  addNode(целочисленный ключ, целочисленная позиция{
                    Если (дерево пустое) {
                           head = новый элемент(ключ, позиция);
                           выход
                    }
                    элемент* root = head;
                    элемент* rootParent = head;
                    пока (root не нулевой) {
                           rootParent = root;
                           (ключ < root->ключ) ? root = root->left : root = root->right;
                    }
                    rootParent->ветка = new элемент(ключ, позиция);
             }
```

2) Поиск ключа:

```
Целочсиленный метод findNode(целочисленный ключ) {
                    элемент* root = head;

                    Пока (не найден элемент с таким ключом или не кончилось дерево)
 Переход root в root->поддерево в зависимости от (ключ < root->ключ);

                    Возврат root->позиция;
             }
```

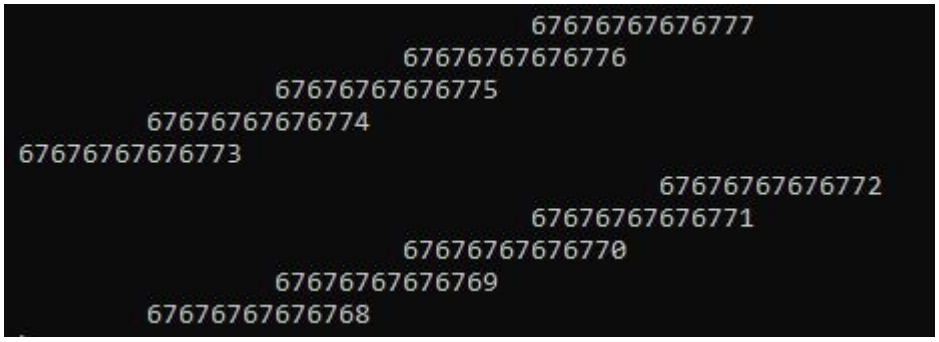
3) Удаление элемента из БДП:

```
метод deleteNode(целочисленный iKey) {
                    элемент* root = head;

                    Пока (root != нулевой и iKey != root->iKey)
(iKey < root->iKey) ? root = root->left : root = root->right;

                    Если (root == нулевой) возврат;

                    если (root->left == нулевой и root->right == нулевой) root =
нулевой;

                    иначе если (root->left != нулевой и root->right == нулевой) {
                          root->swap(root->left);
                    }
                    иначе если (root->right != нулевой и root->left == нулевой) {
                          root->swap(root->right);

                    }
                    иначе {
                          элемент* tmp = root->right;
                          элемент* tmpParent = root;
                          Пока (tmp != нулевой и tmp->left != нулевой)
                          {
                                tmpParent = tmp;
                                tmp = tmp->left;
                          }
                          root->ValueSwap(tmp);

                          (tmpParent->right == tmp) ? tmpParent->right = нулевой :
tmpParent->left = нулевой;
                    }
             }
```

5. Тестирование:

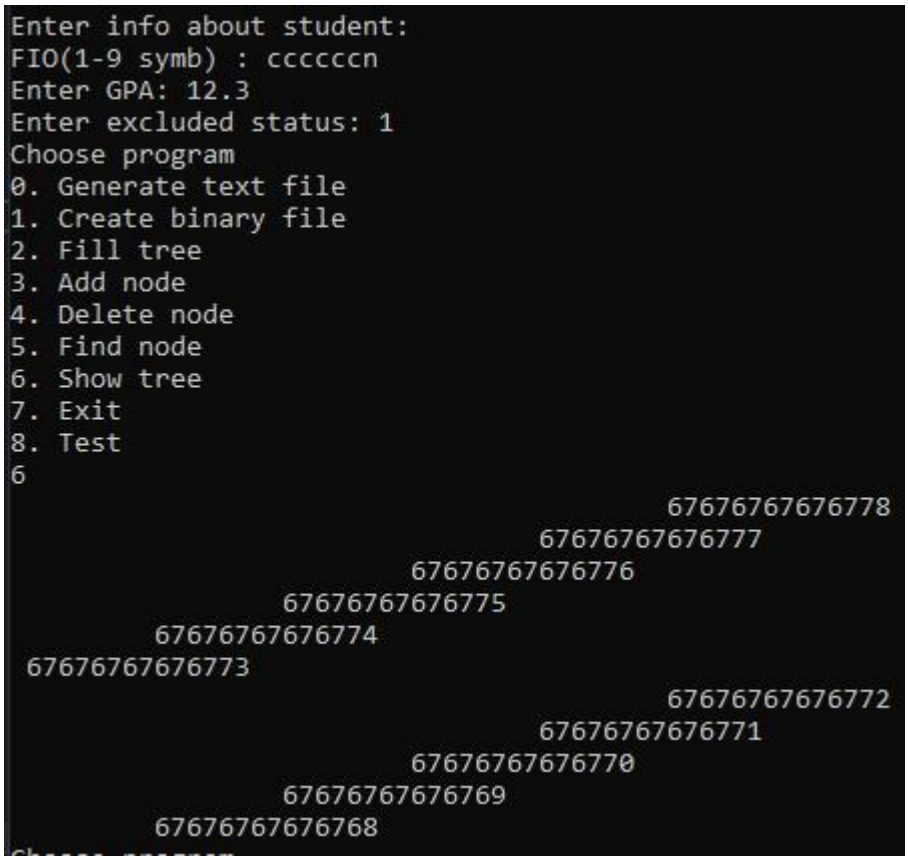
Выполняемые операции:

1) Создание дерева из файла:

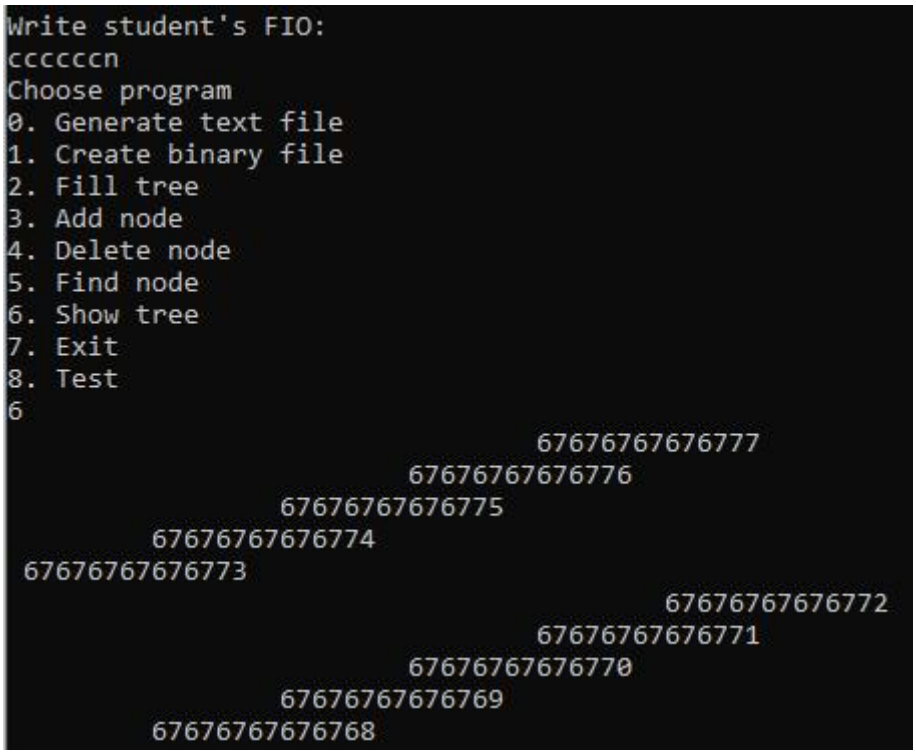


2) Добавление записи:

Ключ (ccccccn - 67676767676778)



3) Удаление записи:



4) Поиск записи:

(ccccccd - 67676767676768)

Содержимое файла:

```
cccccci 289.315 1
ccccccd 191.567 1
cccccce 272.7 0
ccccccf 163.862 1
cccccg 301.566 1
cccccch 252.136 0
ccccccci 289.315 1
cccccj 175.003 0
ccccccck 10.0091 0
ccccccl 125.664 0
ccccccm 59.9536 1
```

Отчёт по заданию 2

1. Постановка задачи:

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

Вариант 21: Красно - чёрное дерево

Дано:

Файл двоичной с записями фиксированной длины.

Структура записи файла:

string double bool

Результат:

Приложение выполняющее операции: удаления из СДП, добавления в СДП, чтение элементов из файла в СДП, поиск в файле с помощью СДП.

2. Подход к решению:

1) СДП - класс

2) Структура элемента СДП: ключ, позиция в файле, правое поддерево, левое поддерево.

3) Методы класса СДП: добавление записи, поиск записи, удаление записи, вывод дерева, левый поворот, правый поворот.

3. Операции по управлению БДП:

1) Добавление элемента:

void addNode(long long int iKey, int iRowNumber);

2) Поиск элемента:

int findNode(long long int iKey);

3) Удаление записи:

void deleteNode(long long int iKey);

4) Вывод дерева:

void printExecute(Node* root, int level);

5) Левый поворот:

void rotateRight(элемент* node);

6) Правый поворот:

void rotateLeft(элемент* node);

4. Алгоритмы операций на псевдокоде:

1) Вставка элемента:

```
метод addNode(целочисленный iKey, целочисленныйiRowNumber) {
                    если (head == нулевой) {
                           head = новый элемент(iKey, iRowNumber, 0, nullptr);
                           возврат;
                    }
                    элемент* node = добавитьЭлементАлгоритмомБСД(iKey, iRowNumber);
                    элемент* parent = node->parent;



                    пока (node != head и parent->iColor == 1) {
                           элемент* grandparent = parent->parent;
                           если (grandparent->left == parent) {
                                  элемент* uncle = grandparent->right;
                                  если (uncle->iRowNumber != -1 && uncle->iColor == 1) {
                                         parent->iColor = 0;
                                         uncle->iColor = 0;
                                         grandparent->iColor = 1;
                                         node = grandparent;
                                         parent = node->parent;
                                  }
                                  иначе {
                                         если (parent->right == node) {
                                                Левый поворот вокруг parent;
                                                Поменять значения(parent, node);
                                         }
                                         Правый поворот вокруг grandparent;
                                         parent->iColor = 0;
                                         grandparent->iColor = 1;
                                         прервать;
                                  }
                           }
                           иначе {
                                  элемент* uncle = grandparent->left;
```

```
                        если (uncle->iRowNumber != -1 && uncle->iColor == 1) {
                                grandparent->iColor = 1;
                                parent->iColor = 0;
                                uncle->iColor = 0;

                                node = grandparent;
                                parent = node->parent;
                        }
                        иначе {
                                если (parent->left == node) {
                                        Правый поворот вокруг parent;
                                        Поменять значения(parent, node);
                                }
                                Левый поворот вокруг grandparent;
                                parent->iColor = 0;
                                grandparent->iColor = 1;
                                прервать;
                        }
                }
        }
        head->iColor = 0;
}
```

2) Поиск ключа - алгоритм совпадает с алгоритмом из задания 1.

3) Удаление элемента из БДП:

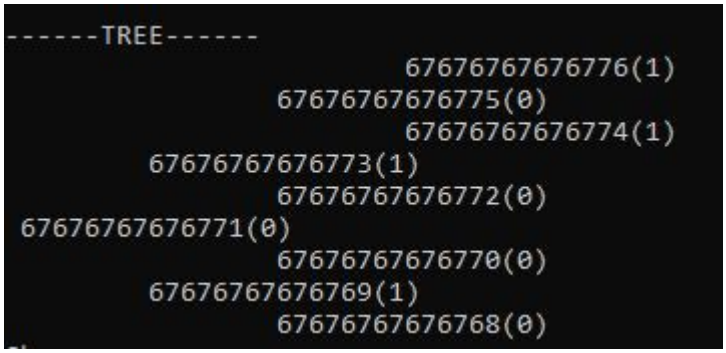
```
метод deleteNode(целочисленный iKey) {
                    элемент* node = удалениеАлгоритмомБДП(iKey);



                    пока (node!= нулевой и node != head и node->iColor == 0) {
                          если (node == node->parent->left) {
                                элемент* sibling = node->parent->right;
                                если (sibling->iColor == 1) {
                                      sibling->iColor = 0;
                                      node->parent->iColor = 1;
                                      левый поворот вокруг(node->parent);
                                      sibling = node->parent->right;
                                }
                                иначе
                                {
                                      если (sibling->right->iRowNumber != -1 и
sibling->right->iColor == 0) {
                                            sibling->left->iColor = 0;
                                            sibling->iColor = 1;
                                            правый поворот вокруг(sibling);
                                            sibling = node->parent->right;
                                      }
                                      sibling->iColor = node->parent->iColor;
                                      node->parent->iColor = 0;
                                      sibling->right->iColor = 0;
                                      левый поворот вокруг(node->parent);
                                      node = head;
                                      прервать;
                                }
                          }
                          иначе {
                                элемент* sibling = node->parent->left;
                                если (sibling->iColor == 1) {
                                      sibling->iColor = 0;
                                      node->parent->iColor = 1;
```

```
                                    левый поворот вокруг(node->parent);
                                    sibling = node->parent->left;
                             }
                             если (sibling->left->iColor == 0 и sibling->right-
>iColor == 0) {
                                    sibling->iColor = 1;
                                    node = node->parent;
                             }
                             иначе
                             {
                                    если (sibling->left->iColor == 0) {
                                           sibling->right->iColor = 0;
                                           sibling->iColor = 1;
                                           левый поворот вокруг(sibling);
                                           sibling = node->parent->left;
                                    }
                                    sibling->iColor = node->parent->iColor;
                                    node->parent->iColor = 0;
                                    sibling->left->iColor = 0;
                                    правый поворот вокруг(node->parent);
                                    node = head;
                                    прервать;
                             }
                      }
                }
          }
```

5. Тестирование:

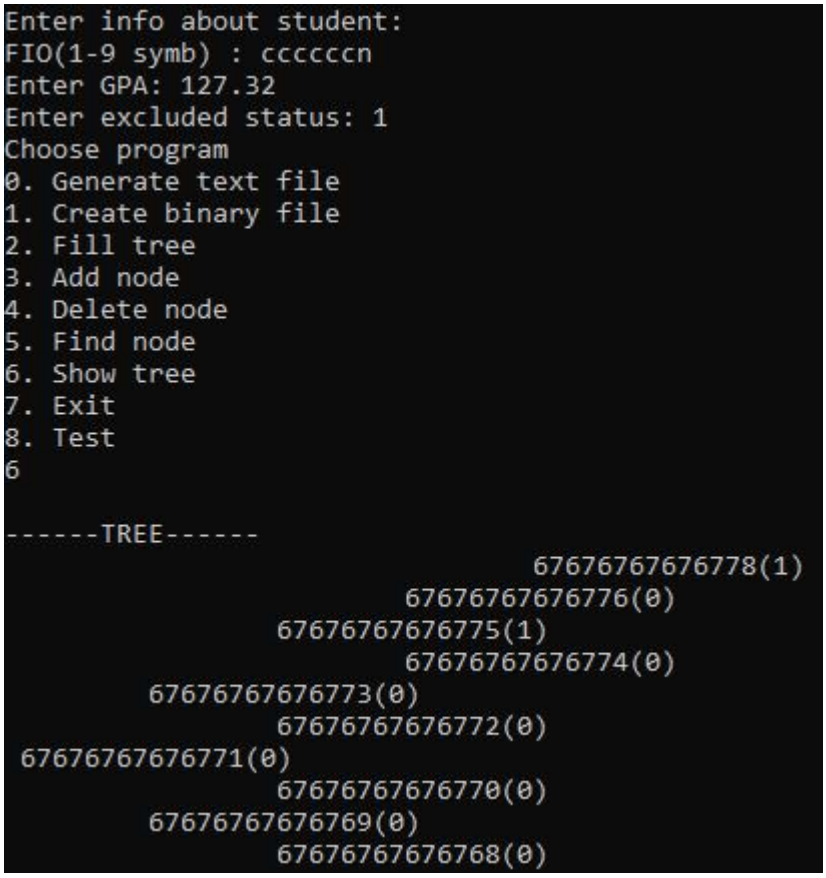
Выполняемые операции:

1) Создание дерева из файла:



2) Добавление записи:

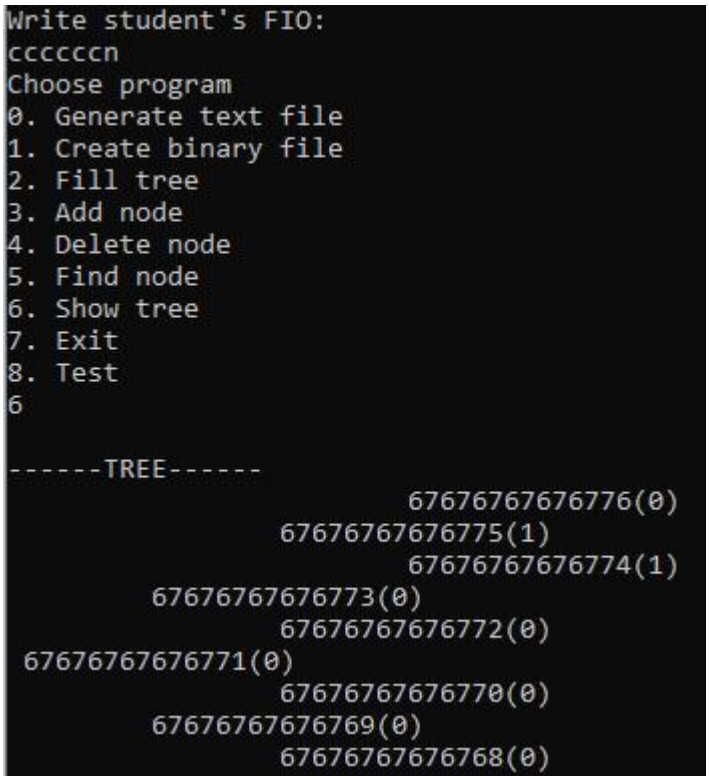
Ключ (ccccccn - 67676767676778)



3) Поиск записи:

(ccccccd - 67676767676768)

4) Удаление записи:



Содержимое файла:

```
cccccci 289.315 1
ccccccd 191.567 1
cccccce 272.7 0
ccccccf 163.862 1
ccccccg 301.566 1
cccccch 252.136 0
cccccci 289.315 1
ccccccj 175.003 0
ccccccк 10.0091 0
ccccccl 125.664 0
ccccccm 59.9536 1
```

6. Количество поворотов:

При увлечении количество поворотов на один добавленный элемент увеличивается. При 10 элементах кол-во поворотов достигло 6, при 100 - 85, при 1000 940, при 10000 9636.

Отчёт по заданию 3.

Вид поисковой структуры	Количество элементов, загруженных в структуру в момент выполнения поиска.	Ёмкостная сложность: объем памяти для структуры.	Количество выполненных сравнений, время на поиск ключа в структуре. (микросекунд)
Хэш-таблица	100	4096	1, 304
Хэш-таблица	10000	262048	2, 522
Хэш-таблица	1000000	2096384	3, 442
БДП	100	3200	47, 317
БДП	10000	320000	4638, 363
БДП	1000000	3200000	9831, 1005
СДП	100	4800	1, 332
СДП	10000	480000	14, 563
СДП	1000000	48000000	29, 704

Доп информация:

1. Приложение для управления файлом:

   Реализация взята из работы 2. Исходный код находится в файле fileController.pdf
   Приложение реализует генерацию файла, преобразование его в бинарный файл, а также полностью работу со всеми тремя видами хранения данных. Для этого все три вида были наследованы от абстрактного класса tree (код в файле tree.pdf) для того, чтобы приложения могло обращаться к экземпляру класса tree, который передаётся в приложение и его реализация может варьироваться в зависимости от целей (выбор различных видов структур).

2. Исходный код :

Бинарное дерево поиска - binaryTreeSearch.h - **binaryTree.pdf**

Красно-черное дерево поиска - red_black_tree.h - **RedBlackTree.pdf**

Приложение - fileController.h - **fileController.pdf**

Дополнительные файлы для реализации приложения - **additional.pdf**

Выводы:

В ходе выполнения данной практической работы были получены навыки работы с различными видами структур хранения. Также на практике было выявлено, что деревья уступают хеш-таблице по скорости, но не значительно. Однако из-за большого объёма структуры записи в деревьях хеш-таблица при том же количестве элементов занимает меньше места, при этом выигрывая по скорости поиска, что делает её в данном случае самым эффективным вариантом хранения данных.

Список литературы:

- Лекции по структурам и алгоритмам обработки данных Рысин М.Л.
- Методическое пособие по выполнению задания 1(битовые операции)