

**К У Р С О В А Я Р А Б О Т А**

**Алгоритмы работы со словарями**

по дисциплине **«Алгоритмы и структуры данных»**

Выполнил студент гр. 3530904/20001 Костин Андрей Константинович

Руководитель Павлов Евгений Алексеевич

«24» мая 2023

**СОДЕРЖАНИЕ**

[**Введение. Общая постановка задачи:** 3](#_Toc135797455)

[**Основная часть работы.** 4](#_Toc135797456)

[1. Описание алгоритма решения и используемых структур данных 4](#_Toc135797457)

[2. Анализ алгоритма 8](#_Toc135797458)

[3. Описание спецификации программы 9](#_Toc135797459)

[4. Описание программы 10](#_Toc135797460)

[**Заключение** 14](#_Toc135797461)

[**Список использованных источников** 15](#_Toc135797462)

[**Приложение 1. Текст программы** 16](#_Toc135797463)

[**Приложение 2. Протоколы отладки** 37](#_Toc135797464)

# **Введение. Общая постановка задачи:**

**Тема: Алгоритмы работы со словарями**

* 1. Для разрабатываемого словаря реализовать основные операции:
     + INSERT (ключ, значение) – добавить запись с указанным ключом и значением
     + SEARCH (ключ)- найти запись с указанным ключом
     + DELETE (ключ)- удалить запись с указанным ключом
  2. Предусмотреть обработку и инициализацию исключительных ситуаций, связанных, например, с проверкой значения полей перед инициализацией и присваиванием.
  3. Программа должна быть написана в соответствии со стилем программирования: C++ Programming Style Guidelines (<http://geosoft.no/development/cppstyle.html>)
  4. Тесты должны учитывать как допустимые, так и не допустимые последовательности входных данных.

**Вариант 1.3.3.**

**Частотный словарь. Красно-черное дерево.**

Разработать и реализовать алгоритм формирования **частотного словаря**:

* определить понятие слово;
* прочитать текст и сформировать набор слов вместе с информацией о частоте их встречаемости;
* определить три чаще всего встречающихся слова;

Для реализации задания использовать **красно-черное дерево**. Узел дерева должен содержать:

* Ключ – слово;
* Цвет узла;
* Информационная часть – количество слов.

# **Основная часть работы.**

## Описание алгоритма решения и используемых структур данных

**Красно-черное дерево** представляет собой бинарное дерево поиска с одним дополнительным ***битом цвета*** в каждом узле. Цвет узла может быть либо красным, либо черным. В соответствии с накладываемыми на узлы дерева ограничениями, ни один путь в красно-черном дереве не отличается от другого по длине более чем в два раза, так что красно-черные деревья являются ***приближенно сбалансированными***. [1]

Каждый узел дерева содержит поля **color**, **key**, **left**, **right** и **p**. Если не существует дочернего или родительского узла по отношению к данному, соответствующий указатель принимает значение **NIL**. Мы будем рассматривать эти значения **NIL** как указатели на внешние узлы (листья) бинарного дерева поиска. При этом все “нормальные” узлы, содержащие поле ключа, становятся внутренними узлами дерева.

Бинарное дерево поиска является **красно-черным деревом**, если оно удовлетворяет следующим красно-черным свойствам.

1. Каждый узел является красным или черным.
2. Корень дерева является черным.
3. Каждый лист дерева (NIL) является черным.
4. Если узел — красный, то оба его дочерних узла — черные.
5. Для каждого узла все пути от него до листьев, являющихся потомками

данного узла, содержат одно и то же количество черных узлов.

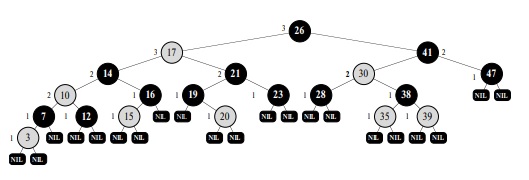


Рисунок . Красно-чёрное дерево

На ***рис.1*** показан пример красно-черного дерева. На рисунке черные узлы показаны темным цветом, красные — светлым. Возле каждого узла показана его “черная” высота. У всех листьев она равна 0.

Количество черных узлов на пути от узла x (не считая сам узел) к листу

будем называть черной высотой узла (**black-height**) и обозначать как bh (x). В соответствии со свойством 5 красно-черных деревьев, черная высота узла — точно определяемое значение. Черной высотой дерева будем считать черную высоту его корня. [1]

Операции над деревом поиска **INSERT** и **DELETE**, будучи применены к красно-черному дереву с n ключами, выполняются за время **O(lg n)**. Поскольку они изменяют дерево, в результате их работы могут нарушаться красно-черные свойства, перечисленные выше. Для восстановления этих свойств мы должны изменить цвета некоторых узлов дерева, а также структуру его указателей.

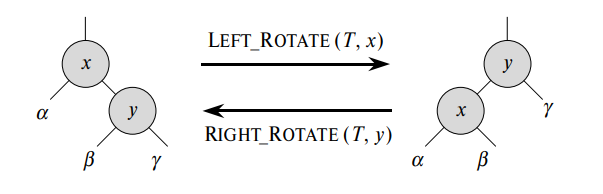


Рисунок . Правый и левый повороты

Изменения в структуре указателей будут выполняться при помощи поворотов (rotations), которые представляют собой локальные операции в дереве поиска, сохраняющие свойство бинарного дерева поиска. На ***рис.2*** показаны два типа поворотов — левый и правый (здесь α, β и γ — произвольные поддеревья). При выполнении левого поворота в узле x предполагается, что его правый дочерний узел y не является листом nil [T]. Левый поворот выполняется “вокруг” связи между x и y, делая y новым корнем поддерева, левым дочерним узлом которого становится x, а бывший левый потомок узла y — правым потомком x.

Код процедуры **RIGHT\_ROTATE (rightRotate())** симметричен коду **LEFT\_ROTATE (leftRotate())**. Обе эти процедуры выполняются за время **O(1)**. При повороте изменяются только указатели, все остальные поля сохраняют свое значение. [3]

**Вставка узла (INSERT – insertNode())**

1. Создать новый узел, который будет вставляться в дерево.
2. Определить место вставки нового узла в дерево. Для этого начать с корневого узла и сравнить значение нового узла с текущим узлом.
3. Если значение нового узла меньше текущего, перейти в левое поддерево текущего узла. Если значение нового узла больше или равно текущему, перейти в правое поддерево.
4. Повторять шаг 3 до тех пор, пока не будет найден листовой узел (узел без потомков), куда можно вставить новый узел.
5. Вставить новый узел как листовой узел в найденное место.
6. Пометить новый узел как красный.
7. Проверить свойства красно-чёрного дерева и выполнить необходимые повороты и перекрашивания узлов, чтобы сохранить эти свойства.
   1. Если родитель нового узла также красный, то выполнить перекрашивание родителя и его соседа в чёрный цвет, а дедушку нового узла - в красный цвет. Затем продолжить проверку свойств с дедушкой нового узла.
   2. Если родитель нового узла является чёрным, то свойства красно-чёрного дерева не нарушены и вставка нового узла завершена.
   3. Если родитель нового узла красный, а дядя нового узла также красный, то выполнить перекрашивание родителя и дяди в чёрный цвет, а дедушку нового узла - в красный цвет. Затем продолжить проверку свойств с дедушкой нового узла.
   4. Если родитель нового узла красный, а дядя нового узла чёрный, то выполнить один из поворотов: левый поворот, если новый узел находится в правом поддереве родителя, или правый поворот, если новый узел находится в левом поддереве родителя. Затем продолжить проверку свойств с новым родителем и дедушкой нового узла.
   5. Если родитель нового узла красный, а дядя нового узла чёрный и новый узел находится в правом поддереве родителя, то выполнить двойной левый поворот: сначала левый поворот относительно родителя нового узла, а затем правый поворот относительно дедушки нового узла. Затем продолжить проверку свойств с новым родителем и дедушкой нового узла.
   6. Если родитель нового узла красный, а дядя нового узла чёрный и новый узел находится в левом поддереве родителя, то выполнить двойной правый поворот: сначала правый поворот относительно родителя нового узла, а затем левый поворот относительно дедушки нового узла. Затем продолжить проверку свойств с новым родителем и дедушкой нового узла. [2]

8. Вернуть корневой узел дерева.

**Удаление узла (DELETE – deleteNode())**

При удалении элементов из красно-черного дерева необходимо выполнить несколько шагов для поддержания его свойств. Рассмотрим подробно данный алгоритм:

1. Сначала мы определяем, есть ли узел с таким ключом в дереве. Если узел не найден, выводится сообщение: *«’word’: no such word in the dictionary»*, иначе алгоритм продолжается.
2. Если у узла, который нужно удалить, есть два потомка, мы находим замещающий узел. Замещающий узел — это самый левый узел в правом поддереве удаляемого узла. Обновляем значения удаляемого узла значениями замещающего узла, а затем продолжаем удаление замещающего узла вместо исходного узла.
3. Запоминаем потомка удаляемого узла (если он существует), чтобы обновить связи его родителя с потомком.
4. Обновляем связи родителя удаляемого узла с потомком. Если у удаляемого узла нет родителя, то это означает, что удаляемый узел является корнем дерева. В этом случае обновляем корень дерева.
5. Если удаляемый узел был черного цвета, возможно нарушение свойств красно-черного дерева. Для исправления нарушений вызывается вспомогательная функция ***deleteFixUp(child, parent)***. Эта функция принимает указатель на потомка удаляемого узла и его родителя.
6. В функции ***deleteFixUp*** входим в цикл, который выполняется, пока узел *node* не станет корнем дерева или пока *node* или его цвет не равны *nullptr* и черному цвету.
7. Далее рассмотрим несколько **случаев**:
8. ***Случай 1***. Внутри цикла проверяем, является ли брат узла *node* красным. Если это так, мы выполняем повороты и изменяем цвета узлов для восстановления свойств дерева.
9. ***Случай 2.*** Если оба потомка брата узла *node* являются черными или нулевыми, изменяем цвета и переходим к родителю *node*, чтобы продолжить проверку.
10. ***Случай 3***. Если правый потомок брата черный или нулевой, выполняем повороты и изменяем цвета узлов для восстановления свойств дерева.
11. ***Случай 4.*** Если левый потомок брата красный, выполняем повороты и изменяем цвета узлов для восстановления свойств дерева. [4]
12. После выхода из цикла устанавливаем цвет корня дерева в черный, чтобы удовлетворить свойствам красно-черного дерева.
13. Алгоритм завершается удалением узла, освобождением памяти и возвратом из функции.

## Анализ алгоритма

Операции *вставки*, *удаления* и *поиска* требуют в худшем случае времени, пропорционального высоте дерева, что позволяет красно-чёрным деревьям быть более эффективными в худшем случае, чем обычные двоичные деревья поиска.

Сложность реализации красно-черного дерева в O-символике в лучших и худших случаях может быть описана следующим образом:

1. **Вставка (insertNode):**

* *Лучший случай*: O(log n), где n - количество узлов в дереве. В лучшем случае новый элемент будет вставлен в листовой узел дерева, и нам нужно будет пройти только по высоте дерева для выполнения вращений и исправления цветов. Таким образом, время выполнения будет пропорционально высоте дерева, что равно O(log n).
* *Худший случай*: O(log n). В худшем случае дерево будет полностью сбалансировано, и вставка нового элемента потребует выполнения вращений и изменений цветов по всей высоте дерева. Поэтому время выполнения будет также пропорционально высоте дерева, что равно O(log n).

1. **Удаление (deleteNode):**

* *Лучший случай*: O(log n). Лучший случай аналогичен лучшему случаю вставки, когда удаление элемента происходит в листовом узле дерева, и исправление цветов и вращения выполняются только на пути от удаленного узла до корня. Время выполнения будет пропорционально высоте дерева, что равно O(log n).
* *Худший случай*: O(log n). Худший случай аналогичен худшему случаю вставки, когда удаление элемента потребует вращений и изменений цветов по всей высоте дерева. Время выполнения будет пропорционально высоте дерева, что равно O(log n).

1. **Поиск (containsNode, searchNode):**

* *Лучший случай*: O(log n). В лучшем случае поиск элемента будет осуществляться по пути от корня до листового узла дерева, что займет время, пропорциональное высоте дерева, равной O(log n).
* *Худший случай*: O(log n). В худшем случае поиск потребует просмотра всех узлов дерева по пути от корня до искомого элемента. Время выполнения будет пропорционально высоте дерева, что равно O(log n).

В целом, вставка, удаление, поиск и обращение по индексу в красно-черном дереве имеют сложность O(log n) в лучшем и худшем случаях. Это означает, что время выполнения операций масштабируется логарифмически с ростом числа элементов в дереве. [2]

## Описание спецификации программы

Программа представляет консольное приложение, которое сначала выводит тесты формирования таблицы, в которой все слова будут расположены в алфавитном порядке и для каждого слова будет указано количество вхождений.

Также предусмотрена система команд. Пользователь может ввести команду help и получить список всех доступных команд. Пользователь может получить результат выполнения команд прямо в окне консольного приложения, либо указать название файла, в который будет произведена запись. После вывода результата формирования таблицы пользователю будет предложена опция удаления слов из словаря. Результат выполнения удаления слов из словаря запишется в то же место, которое пользователь указал для вывода результата команды.

При формировании частотного словаря должны учитываться все слова, записанные буквами английского, русского алфавитов или цифрами, а также их комбинациями. Любые знаки препинания не учитываются. Например, при анализе строки *«.,»”hey?geek.\!»* мы получим только слова hey и geek. Регистр букв также не учитывается. Например, строки *«GEEk»* и *«geeK»* конвертируются в одно и то же слово geek. Если одно и то же слово встречается в тексте несколько раз и в нескольких строках, то номера строк должны быть перечислены вместе в порядке возрастания через пробел. Вывод самих слов осуществляется в алфавитном порядке.

Кроме того, предусмотрена команда анализа текста, в результате которой формируется словарь, отсортированный по алфавитному порядку ключей и предоставляется возможность удаления и вставки слов. Предусмотрены команды вывод трёх самых частых слов текста и словаря с частотами, отсортированный по невозрастанию информационной части вхождений.

В случае невозможности работы с указанными файлами (например, при отсутствии файла, невозможности его открыть и отсутствии требуемого расширения «.txt», а также при совпадении входного и выходного файлов для нумерации строк) пользователю будет выведено предупреждение. Пользователь сможет заново повторить команду, если введёт команды в соответствии с указанными требованиями.Если исходный файл окажется пустым, то программа отработает, но будет выведено предупреждение о том, что словарь пустой.

## Описание программы

* *Файлы RBTree.h и RBTree.cpp*

В данных файлах представлено объявление и имплементация шаблонного класса RBTree, который представляет собой словарь на основе красно-черного дерева. Шаблон имеет аргументы: K – тип данных ключа (в нашем случае string, так как этот тип данных хранит слова), V – тип данных хранимого информационного значения (в нашем случае это целочисленный тип int количества вхождений). Объект дерева хранит только указатель на корень дерева.

Интерфейс класса имеет следующие публичные методы:

* Метод **insertNode** – добавляет новые пары ключ-значение в красно-черное дерево;
* Метод **deleteNode** – удаляет узел с указанным ключом;
* Метод **containsNode** – возвращает булевое значение true, если переданный в него ключ находится в данном красно-черном дереве, иначе – false;
* Метод **searchNode** - возвращает указатель на узел дерева, содержащий заданный ключ.
* Оператор индексации **[]** – принимает значение ключа. Возвращает хранимое значение в узле, с таким ключом.

Для получения всех узлов дерева применяются публичные классы итератора iterator и const iterator с перегруженными операторами и методами:

* Оператор **!=** сравнивает два итератора и возвращает true, если они указывают на разные узлы дерева;
* Оператор **++** перемещает итератор на следующий узел дерева;
* Оператор **+ val** перемещает итератор на определённое количество узлов далее (val);
* Методы **key()** и **value()** возвращают ключ и значение текущего узла;
* Метод **move\_next()** используется для перемещения итератора на следующий узел дерева – инфиксный обход.

Имплементация класса достигается с помощью структуры Node, которая хранит пару ключ-значение, а также указателей на двух потомков (левый и правый) и собственного предка (родителя). Также используются private методы для восстановления свойств красно-черного дерева после добавления или удаления элементов, а также методы для поворота дерева.

* *Файлы TextProcessing.h и TextProcessing.cpp*

В данных файлах представлено объявление и имплементация класса ***TextHandler*** (обработчик текста), который читает файл и формирует таблицу, в которой все слова будут расположены в алфавитном порядке и для каждого слова будет указан количество вхождений. Конструктор класса инициализирует пустой словарь.

Интерфейс класса имеет следующие публичные методы:

* Метод **analyse** – получает на вход название файла или входной поток в зависимости от переданного параметра. Формирует частотный словарь;
* Метод **deleteWord** – получает на вход слово, которое следует удалить. Удаляет это слово из словаря;
* Метод **insertWord** – получает на вход слово и количество вхождений, которое следует вставить определенное количество раз. Вставляет это слово в словарь;
* Метод **printStatistics** (std::ostream& os) – принимает на вход выходной поток, в который будет совершаться запись. Выводит содержимое словаря в выходной поток в виде таблицы, где каждому слову соответствует количество вхождений.
* Метод **printStatistics** (const std::string& filename) - принимает на вход название файла, в который будет совершаться запись. Записывает содержимое словаря в файл в виде таблицы, аналогичной методу printStatistics.
* Метод **print3MFW** (std::ostream& os) – принимает на вход выходной поток, в который будет совершаться запись. Выводит содержимое словаря из трёх самых встречаемых слов в выходной поток в виде таблицы, где каждому слову соответствует количество вхождений, отсортированный по невозрастанию вхождений.
* Метод **print3MFW** (const std::string& filename) - принимает на вход название файла, в который будет совершаться запись. Записывает содержимое словаря в файл в виде таблицы, аналогичной методу print3MFW.
* Метод **printFreqStat** (std::ostream& os) – принимает на вход выходной поток, в который будет совершаться запись. Выводит содержимое словаря в выходной поток в виде таблицы, где каждому слову соответствует количество вхождений, отсортированный по невозрастанию вхождений.
* Метод **printFreqStat** (const std::string& filename) - принимает на вход название файла, в который будет совершаться запись. Записывает содержимое словаря в файл в виде таблицы, аналогичной методу printFreqStat.
* *Файлы Commands.h и Commands.cpp*

В данных файлах представлено объявление и имплементация функций для реализации системы команд в окне консольного приложения:

* **readCommand** - функция, которая считывает команды пользователя и вызывает соответствующую функцию для их выполнения. Входной параметр - строка *«command»*, содержащая команду пользователя;
* **printHelp** - функция, которая выводит на экран список доступных команд для пользователя;
* **analyseText** - функция, которая анализирует текст из файла и выводит результаты анализа по ключам в терминал или записывает их в файл. Входной параметр - строка «filename», содержащая имя файла с текстом;
* **ThreeMFWText** – функция, которая анализирует текст из файла и выводит результаты анализа трех самых частых слов в терминал или записывает их в файл. Входной параметр - строка «filename», содержащая имя файла с текстом;
* **analyseFreq** - функция, которая анализирует текст из файла и выводит результаты анализа частотности в терминал или записывает их в файл. Входной параметр - строка «filename», содержащая имя файла с текстом.
* *Файл main.cpp*

В данном файле представлены функции tableTest, которые используются для тестирования функций analyseText соответственно.

Функция tableTest проверяет работу класса TextWorker, который используется для анализа текстового файла и формирования таблицы, в которой все слова будут расположены в алфавитном порядке и для каждого слова будет указан список вхождений в алфавитном порядке ключей. Сначала она создает объект класса TextHandler, вызывает метод analyse для анализа файла, затем выводит результаты в консоль и записывает в другой файл. Далее происходят проверки на удаление слов (advantages, networks) из словаря и обработку исключений, такие как передача несуществующего файла, запись в файл с неправильным расширением и невозможность создать выходной файл.

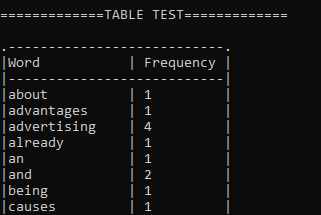


Рисунок . Тест форматирования таблицы

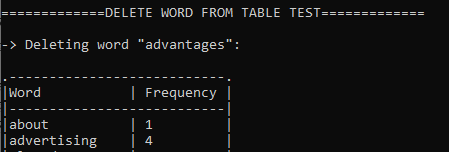


Рисунок . Тест удаления слов

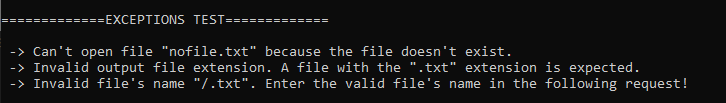


Рисунок . Тест исключений

В функции main вызываются функции tableTest для тестирования функционала программы. Затем происходит считывание команд пользователя с помощью функции std::getline, которая передает команду в функцию readCommand. Эта функция анализирует команду и вызывает соответствующую функцию для ее выполнения.

# **Заключение**

В результате выполнения курсовой работы мной была изучена структура данных – красно-черное дерево, а также алгоритмы и методы для работы с ним. В процессе работы были разработаны классы, представляющие собой частотный словарь на основе красно-черного дерева и анализ полученного словаря относительно алфавитного и частотного порядков. Также реализована система команд для взаимодействия пользователя с программой в окне консольного приложения. В результате, программа соответствует всем поставленным целям и задачам.

# **Список использованных источников**

1. Алгоритмы: построение и анализ: / Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. - М.: Вильямс, 2011.- 1296 с.
2. Introduction to Red-Black Tree/ URL: <https://www.geeksforgeeks.org/introduction-to-red-black-tree/> – (дата обращения 20.05.2023)
3. Википедия – Красно-Чёрное дерево / URL: <https://ru.wikipedia.org/wiki/Красно-чёрное_дерево> – (дата обращения 20.05.2023)
4. Red-Black Tree/ URL: <https://www.programiz.com/dsa/red-black-tree> – (дата обращения 10.05.2023)
5. Визуализатор красно-черного дерева / URL: <https://www.cs.usfca.edu/~galles/visualization/RedBlack.html> - (дата обращения 13.05.2023).
6. Понимаем красно-черное дерево. Часть 1. Введение / URL: <https://habr.com/ru/articles/555404/> - (дата обращения 08.05.2023).
7. Понимаем красно-черное дерево. Часть 2. Балансировка и вставка / URL: <https://habr.com/ru/articles/557328/> - (дата обращения 08.05.2023).

# **Приложение 1. Текст программы**

*Файл RedBlackTree.h*

#ifndef CROSS\_REFERENCES\_REDBLACKTREE\_H

#define CROSS\_REFERENCES\_REDBLACKTREE\_H

enum color\_t //тип цвета узлов

{

RED = false,

BLACK = true

};

template <typename K, typename V>

struct Node //структура узла

{

K key; //ключ - слово

V value; //информационная часть - частота встречаемости

color\_t color; //цвет узла (для балансировки КЧД)

Node<K, V>\* parent;

Node<K, V>\* left;

Node<K, V>\* right;

Node(K key, V value, color\_t color, Node\* parent, Node\* left, Node\* right) :

key(key), value(value), color(color), parent(parent),

left(left), right(right)

{}

};

template <typename K, typename V>

class RBTree

{

public:

RBTree() :

root\_(nullptr)

{

root\_ = nullptr;

}

~RBTree();

class iterator;

void insertNode(const K& key, const V& value); //вставка узла

void deleteNode(const K& key); //удаление узла

bool containsNode(const K& key) const; //проверка содержания узла

Node<K, V>\* searchNode(const K& key) const; //поиск узла

V& operator[](const K& key);

const V& operator[](const K& key) const;

iterator begin();

iterator end();

iterator preend();

bool isEmpty(); //проверка на пустоту КЧД

private:

void deleteNode(Node<K, V>\* node); //удаление узла

void deleteFixUp(Node<K, V>\* p, Node<K, V>\* parent); //балансировка после удаления

void leftRotate(Node<K, V>\* x); //левый поворот

void rightRotate(Node<K, V>\* x); //правый поворот

void insertFixUp(Node<K, V>\* z); //балансировка после вставки

void destroy(Node<K, V>\*& node); //удаление дерева

Node<K, V>\* root\_; //корень КЧД

};

template <typename K, typename V>

class RBTree<K, V>::iterator

{

public:

explicit iterator(Node<K, V>\* node) :

node\_{ node }

{}

bool operator!=(const iterator& rhs) const

{

return node\_ != rhs.node\_;

}

iterator operator++()

{

move\_next();

return \*this;

}

iterator operator+(int val)

{

for (int i = 0; i < val; i++)

{

move\_next();

}

return \*this;

}

K& key()

{

return node\_->key;

}

V& value()

{

return node\_->value;

}

private:

void move\_next()

{

if (node\_->right)

{

node\_ = node\_->right;

while (node\_->left)

{

node\_ = node\_->left;

}

}

else if (node\_->parent && (node\_->parent->left == node\_))

{

node\_ = node\_->parent;

}

else {

while (node\_->parent && (node\_->parent->right == node\_))

{

node\_ = node\_->parent;

}

node\_ = node\_->parent;

}

}

Node<K, V>\* node\_;

};

#endif

*Файл RBTree.cpp*

#include "RedBlackTree.h"

#include <iostream>

template<typename K, typename V>

RBTree<K, V>::~RBTree()

{

destroy(root\_);

}

template<typename K, typename V>

void RBTree<K, V>::insertNode(const K& key, const V& value) //вставка узла

{

Node<K, V>\* z = new Node<K, V>(key, value, RED, nullptr, nullptr, nullptr);

Node<K, V>\* y = nullptr; // указатель на родителя

Node<K, V>\* x = root\_; // указатель на текущий узел

while (x != nullptr)

{

y = x; // запоминаем родителя

if (z->key < x->key) // если ключ z меньше ключа x, то идем влево

{

x = x->left;

}

else // если ключ z больше или равен ключу x, то идем вправо

{

x = x->right;

}

}

z->parent = y;

if (y == nullptr)

{

root\_ = z;

}

else if (z->key < y->key) // если ключ z меньше ключа y, то z становится левым потомком y

{

y->left = z;

}

else // если ключ z больше или равен ключу y, то z становится правым потомком y

{

y->right = z;

}

z->color = RED;

insertFixUp(z);

}

template<typename K, typename V>

void RBTree<K, V>::deleteNode(const K& key) //удаление узла

{

Node<K, V>\* dNode = searchNode(key);

if (dNode != nullptr)

{

deleteNode(dNode);

}

}

template<typename K, typename V>

bool RBTree<K, V>::containsNode(const K& key) const //содержит ли КЧД искомый узел

{

Node<K, V>\* current = root\_;

while (current)

{

if (current->key == key)

{

return true;

}

else if (current->key > key)

{

current = current->left;

}

else

{

current = current->right;

}

}

return false;

}

template<typename K, typename V>

Node<K, V>\* RBTree<K, V>::searchNode(const K& key) const //поиск узла

{

Node<K, V>\* current = root\_;

while (current)

{

if (current->key == key)

{

return current;

}

else if (current->key > key)

{

current = current->left;

}

else

{

current = current->right;

}

}

}

template<typename K, typename V>

V& RBTree<K, V>::operator[](const K& key)

{

Node<K, V>\* node = searchNode(key);

if (node)

{

return node->value;

}

throw std::invalid\_argument{ "No such key in the RBTree!" };

}

template<typename K, typename V>

const V& RBTree<K, V>::operator[](const K& key) const

{

Node<K, V>\* node = searchNode(key);

if (node)

{

return node->value;

}

throw std::invalid\_argument{ "No such key in the RBTree!" };

}

template<typename K, typename V>

typename RBTree<K, V>::iterator RBTree<K, V>::begin()

{

Node<K, V>\* c = root\_;

if (root\_ == nullptr)

{

return iterator(c);

}

while (c->left)

{

c = c->left;

}

return iterator(c);

}

template<typename K, typename V>

typename RBTree<K, V>::iterator RBTree<K, V>::end()

{

return iterator(nullptr);

}

template<typename K, typename V>

typename RBTree<K, V>::iterator RBTree<K, V>::preend()

{

Node<K, V>\* c = root\_;

if (root\_ == nullptr)

{

return iterator(c);

}

while (c->right)

{

c = c->right;

}

return iterator(c);

}

template<typename K, typename V>

bool RBTree<K, V>::isEmpty() //проверка на пустоту КЧД

{

if (root\_ == nullptr)

{

return true;

}

else

return false;

}

template<typename K, typename V>

void RBTree<K, V>::deleteNode(Node<K, V>\* node) //удаление узла

{

Node<K, V>\* child;

Node<K, V>\* parent;

color\_t color;

if (node->left != nullptr && node->right != nullptr) {

// Два потомка

// Найти замещающий узел (самый левый узел в правом поддереве)

Node<K, V>\* replace = node->right;

while (replace->left != nullptr) {

replace = replace->left;

}

// Заменить значение удаляемого узла значением замещающего узла

node->key = replace->key;

node->value = replace->value;

node = replace;

}

// Запомнить потомка удаляемого узла

if (node->left != nullptr) {

child = node->left;

}

else {

child = node->right;

}

parent = node->parent;

color = node->color;

// Удалить узел, обновив связи его родителя с потомком

if (child != nullptr) {

child->parent = parent;

}

if (parent == nullptr) {

root\_ = child; // Удаляемый узел - корень дерева

}

else if (node == parent->left) {

parent->left = child;

}

else {

parent->right = child;

}

if (color == BLACK) {

deleteFixUp(child, parent); //балансировка КЧД после удаления узла

}

delete node;

}

template<typename K, typename V>

void RBTree<K, V>::deleteFixUp(Node<K, V>\* node, Node<K, V>\* parent) { //балансировка после удаления узла

Node<K, V>\* sibling;

while (node != root\_ && (node == nullptr || node->color == BLACK)) {

if (node == parent->left) {

sibling = parent->right;

if (sibling->color == RED) {

// #1: Брат узла node - красный

sibling->color = BLACK;

parent->color = RED;

leftRotate(parent);

sibling = parent->right;

}

if ((sibling->left == nullptr || sibling->left->color == BLACK) &&

(sibling->right == nullptr || sibling->right->color == BLACK)) {

// #2: Оба потомка брата черные

sibling->color = RED;

node = parent;

parent = node->parent;

}

else {

if (sibling->right == nullptr || sibling->right->color == BLACK) {

// #3: Правый потомок брата черный

sibling->left->color = BLACK;

sibling->color = RED;

rightRotate(sibling);

sibling = parent->right;

}

// #4: Правый потомок брата красный

sibling->color = parent->color;

parent->color = BLACK;

sibling->right->color = BLACK;

leftRotate(parent);

node = root\_;

break;

}

}

else {

sibling = parent->left;

if (sibling->color == RED) {

// #1: Брат узла node - красный

sibling->color = BLACK;

parent->color = RED;

rightRotate(parent);

sibling = parent->left;

}

if ((sibling->left == nullptr || sibling->left->color == BLACK) &&

(sibling->right == nullptr || sibling->right->color == BLACK)) {

// #2: Оба потомка брата черные

sibling->color = RED;

node = parent;

parent = node->parent;

}

else {

if (sibling->left == nullptr || sibling->left->color == BLACK) {

// #3: Левый потомок брата черный

sibling->right->color = BLACK;

sibling->color = RED;

leftRotate(sibling);

sibling = parent->left;

}

// #4: Левый потомок брата красный

sibling->color = parent->color;

parent->color = BLACK;

sibling->left->color = BLACK;

rightRotate(parent);

node = root\_;

break;

}

}

}

if (node != nullptr) {

node->color = BLACK;

}

}

template<typename K, typename V>

void RBTree<K, V>::leftRotate(Node<K, V>\* x) //левый поворот

{

Node<K, V>\* y = x->right;

x->right = y->left;

if (y->left != nullptr)

{

y->left->parent = x;

}

y->parent = x->parent;

if (x->parent == nullptr)

{

root\_ = y;

}

else if (x == x->parent->left)

{

x->parent->left = y;

}

else

{

x->parent->right = y;

}

y->left = x;

x->parent = y;

}

template<typename K, typename V>

void RBTree<K, V>::rightRotate(Node<K, V>\* x) //правый поворот

{

Node<K, V>\* y = x->left;

x->left = y->right;

if (y->right != nullptr)

{

y->right->parent = x;

}

y->parent = x->parent;

if (x->parent == nullptr)

{

root\_ = y;

}

else if (x == x->parent->right)

{

x->parent->right = y;

}

else

{

x->parent->left = y;

}

y->right = x;

x->parent = y;

}

template<typename K, typename V>

void RBTree<K, V>::insertFixUp(Node<K, V>\* z) //балансировка КЧД после удаления узла

{

Node<K, V>\* parent = z->parent;

while (z != root\_ && parent->color == RED)

{

Node<K, V>\* grandparent = parent->parent;

if (grandparent->left == parent)

{

Node<K, V>\* uncle = grandparent->right;

if (uncle != nullptr && uncle->color == RED)

{

parent->color = BLACK;

uncle->color = BLACK;

grandparent->color = RED;

z = grandparent;

parent = z->parent;

}

else

{

if (parent->right == z)

{

leftRotate(parent);

std::swap(z, parent);

}

rightRotate(grandparent);

grandparent->color = RED;

parent->color = BLACK;

break;

}

}

else

{

Node<K, V>\* uncle = grandparent->left;

if (uncle != nullptr && uncle->color == RED)

{

parent->color = BLACK;

uncle->color = BLACK;

grandparent->color = RED;

z = grandparent;

parent = z->parent;

}

else

{

if (parent->left == z)

{

rightRotate(parent);

std::swap(parent, z);

}

leftRotate(grandparent);

grandparent->color = RED;

parent->color = BLACK;

break;

}

}

}

root\_->color = BLACK;

}

template<typename K, typename V>

void RBTree<K, V>::destroy(Node<K, V>\*& node) //удаление дерева

{

if (node == nullptr)

return;

destroy(node->left);

destroy(node->right);

delete node;

node = nullptr;

}

*Файл TextProcessing.h*

#ifndef CROSS\_REFERENCES\_TEXT\_PROCESSING

#define CROSS\_REFERENCES\_TEXT\_PROCESSING

#include "RedBlackTree.h"

#include <list>

#include <string>

#include <ios>

class TextHandler

{

public:

TextHandler();

~TextHandler() = default;

void analyse(const std::string& filename);

void analyse(std::istream& is);

void deleteWord(std::string& word);

void insertWord(std::string& word, int val);

void print3MFW(std::ostream& os);

void print3MFW(const std::string& filename);

void printFreqStat(std::ostream& os);

void printFreqStat(const std::string& filename);

void printStatistics(std::ostream& os);

void printStatistics(const std::string& filename);

bool isDictEmpty();

private:

RBTree< std::string, int > dictionary;

};

#endif

*Файл TextProcessing.cpp*

#include "TextProcessing.h"

#include "RedBlackTree.h"

#include "RedBlackTree.cpp"

#include <iostream>

#include <fstream>

#include <regex>

#include <string>

std::string generateSpaces(size\_t length);

std::string generateLine(size\_t length);

TextHandler::TextHandler() :

dictionary{ }

{}

void TextHandler::analyse(const std::string& filename) //анализ файла

{

dictionary = RBTree< std::string, int >{ }; //создаём пустой словарь

std::ifstream is(filename); //открываем поток для чтения файла

if (!is)

{

throw std::invalid\_argument{ " -> Can't open file \"" + filename + "\" because the file doesn't exist." };

}

analyse(is);

is.close();

}

void TextHandler::analyse(std::istream& is) //анализ потока

{

dictionary = RBTree< std::string, int >{ }; //создаём пустой словарь

auto word\_regex = std::regex{ "[a-zA-Zа-яА-Я0-9]+" };

std::string currline;

while (!is.eof())

{

std::getline(is, currline);

auto words\_begin = std::sregex\_iterator{ currline.begin(), currline.end(), word\_regex };

auto words\_end = std::sregex\_iterator{ };

for (auto itr = words\_begin; itr != words\_end; ++itr)

{

auto word = itr->str(); //перевод в строковый тип

for (char& c : word)

{

c = std::tolower(c); //понижение регистра

}

if (dictionary.containsNode(word))

{

dictionary[word]++;

}

else

{

dictionary.insertNode(word, 1);

}

}

}

}

void TextHandler::deleteWord(std::string& word)

{

if (dictionary.containsNode(word))

{

dictionary.deleteNode(word);

}

else

{

std::cout << word << ": invalid word (there is no such word in the dictionary).\n";

}

}

void TextHandler::insertWord(std::string& word, int val)

{

if (dictionary.containsNode(word))

{

dictionary.searchNode(word)->value += val;

}

else

{

dictionary.insertNode(word, val);

}

}

void TextHandler::print3MFW(std::ostream& os)

{

std::string word = "Word";

auto coloumnswidth = word.length();

auto max1 = dictionary.preend();

int valm = max1.value();

max1.value() = 0;

for (auto itr = dictionary.begin(); itr != dictionary.end(); ++itr)

{

if (itr.value() > max1.value()) max1 = itr;

}

for (auto itr = dictionary.begin(); itr != dictionary.end(); ++itr)

{

coloumnswidth = std::max(itr.key().length(), coloumnswidth); //слово максимальной длины

}

const auto margin = size\_t{ 2 };

const auto margin2 = size\_t{ 11 };

const auto margin3 = size\_t{ 9 };

const auto margin4 = size\_t{ 1 };

size\_t margin5;

if (max1.value() <= 9) margin5 = size\_t{ 0 };

else if (max1.value() <= 99) margin5 = size\_t{ 1 };

else if (max1.value() <= 999) margin5 = size\_t{ 2 };

else if (max1.value() <= 9999) margin5 = size\_t{ 3 };

else if (max1.value() <= 99999) margin5 = size\_t{ 4 };

os << "." << generateLine(margin2 + coloumnswidth + margin + margin4) << ".\n";

os << "|Word" << generateSpaces(coloumnswidth - word.length() + margin) << "| Frequency |\n";

os << "|" << generateLine(margin2 + coloumnswidth + margin + margin4) << "|\n";

os << "|" << max1.key() << generateSpaces(coloumnswidth - max1.key().length() + margin) << "| " << max1.value() << generateSpaces(margin3 - margin5) << "|\n";

for (int i = 0; (max1.value() > 0) && (i < 2); max1.value()--)

{

for (auto itr = dictionary.begin(); (itr != dictionary.end()) && (i < 2); ++itr)

{

if ((itr.value() == max1.value()) && (itr.key() != max1.key()))

{

if (itr.value() <= 9) margin5 = size\_t{ 0 };

else if (itr.value() <= 99) margin5 = size\_t{ 1 };

else if (itr.value() <= 999) margin5 = size\_t{ 2 };

os << "|" << itr.key() << generateSpaces(coloumnswidth - itr.key().length() + margin) << "| " << itr.value() << generateSpaces(margin3 - margin5) << "|\n";

i++;

}

}

}

os << "'" << generateLine(margin2 + coloumnswidth + margin + margin4) << "\'\n";

max1.value() = valm;

if (dictionary.isEmpty())

{

os << "Dictionary is empty!\n";

}

}

void TextHandler::print3MFW(const std::string& filename)

{

if (filename.length() <= 4 || filename.substr(filename.size() - 4) != ".txt") //проверка на расширение и название файла

{

throw std::invalid\_argument(" -> Invalid output file extension. A file with the \".txt\" extension is expected.");

}

std::ofstream os(filename);

if (!os)

{

throw std::invalid\_argument{ " -> Invalid file's name \"" + filename + "\". Enter the valid file's name in the following request!" };

}

print3MFW(os);

os.close();

}

void TextHandler::printFreqStat(std::ostream& os)

{

std::string word = "Word";

auto coloumnswidth = word.length();

auto max1 = dictionary.preend();

int valm = max1.value();

max1.value() = 0;

for (auto itr = dictionary.begin(); itr != dictionary.end(); ++itr)

{

if (itr.value() > max1.value()) max1 = itr;

}

for (auto itr = dictionary.begin(); itr != dictionary.end(); ++itr)

{

coloumnswidth = std::max(itr.key().length(), coloumnswidth); //слово максимальной длины

}

const auto margin = size\_t{ 2 };

const auto margin2 = size\_t{ 11 };

const auto margin3 = size\_t{ 9 };

const auto margin4 = size\_t{ 1 };

size\_t margin5;

os << "." << generateLine(margin2 + coloumnswidth + margin + margin4) << ".\n";

os << "|Word" << generateSpaces(coloumnswidth - word.length() + margin) << "| Frequency |\n";

os << "|" << generateLine(margin2 + coloumnswidth + margin + margin4) << "|\n";

if (max1.value() <= 9) margin5 = size\_t{ 0 };

else if (max1.value() <= 99) margin5 = size\_t{ 1 };

else if (max1.value() <= 999) margin5 = size\_t{ 2 };

else if (max1.value() <= 9999) margin5 = size\_t{ 3 };

else if (max1.value() <= 99999) margin5 = size\_t{ 4 };

os << "|" << max1.key() << generateSpaces(coloumnswidth - max1.key().length() + margin) << "| " << max1.value() << generateSpaces(margin3 - margin5) << "|\n";

for (max1; max1.value() > 0; max1.value()--)

{

for (auto itr = dictionary.begin(); itr != dictionary.end(); ++itr)

{

if ((itr.value() == max1.value()) && (itr.key() != max1.key()))

{

if (itr.value() <= 9) margin5 = size\_t{ 0 };

else if (itr.value() <= 99) margin5 = size\_t{ 1 };

else if (itr.value() <= 999) margin5 = size\_t{ 2 };

os << "|" << itr.key() << generateSpaces(coloumnswidth - itr.key().length() + margin) << "| " << itr.value() << generateSpaces(margin3 - margin5) << "|\n";

}

}

}

os << "'" << generateLine(margin2 + coloumnswidth + margin + margin4) << "\'\n";

max1.value() = valm;

if (dictionary.isEmpty())

{

os << "Dictionary is empty!\n";

}

}

void TextHandler::printFreqStat(const std::string& filename)

{

if (filename.length() <= 4 || filename.substr(filename.size() - 4) != ".txt") //проверка на расширение и название файла

{

throw std::invalid\_argument(" -> Invalid output file extension. A file with the \".txt\" extension is expected.");

}

std::ofstream os(filename);

if (!os)

{

throw std::invalid\_argument{ " -> Invalid file's name \"" + filename + "\". Enter the valid file's name in the following request!" };

}

printFreqStat(os);

os.close();

}

void TextHandler::printStatistics(std::ostream& os)

{

std::string word = "Word";

auto coloumnswidth = word.length();

for (auto itr = dictionary.begin(); itr != dictionary.end(); ++itr)

{

coloumnswidth = std::max(itr.key().length(), coloumnswidth); //слово максимальной длины

}

const auto margin = size\_t{ 2 };

const auto margin2 = size\_t{ 11 };

const auto margin3 = size\_t{ 9 };

const auto margin4 = size\_t{ 1 };

size\_t margin5;

os << "." << generateLine(margin2 + coloumnswidth + margin + margin4) << ".\n";

os << "|Word" << generateSpaces(coloumnswidth - word.length() + margin) << "| Frequency |\n";

os << "|" << generateLine(margin2 + coloumnswidth + margin + margin4) << "|\n";

for (auto itr = dictionary.begin(); itr != dictionary.end(); ++itr)

{

if (itr.value() <= 9) margin5 = size\_t{ 0 };

else if (itr.value() <= 99) margin5 = size\_t{ 1 };

else if (itr.value() <= 999) margin5 = size\_t{ 2 };

os << "|" << itr.key() << generateSpaces(coloumnswidth - itr.key().length() + margin) << "| " << itr.value() << generateSpaces(margin3 - margin5) << "|\n";

}

os << "'" << generateLine(margin2 + coloumnswidth + margin + margin4) << "\'\n";

os << "\n";

if (dictionary.isEmpty())

{

os << "Dictionary is empty!\n";

}

}

void TextHandler::printStatistics(const std::string& filename)

{

if (filename.length() <= 4 || filename.substr(filename.size() - 4) != ".txt") //проверка на расширение и название файла

{

throw std::invalid\_argument(" -> Invalid output file extension. A file with the \".txt\" extension is expected.");

}

std::ofstream os(filename);

if (!os)

{

throw std::invalid\_argument{ " -> Invalid file's name \"" + filename + "\". Enter the valid file's name in the following request!" };

}

printStatistics(os);

os.close();

}

bool TextHandler::isDictEmpty()

{

if (dictionary.isEmpty())

{

return true;

}

return false;

}

std::string generateSpaces(size\_t length)

{

std::string spaces;

for (size\_t i = 0; i < length; ++i) {

spaces += " ";

}

return spaces;

}

std::string generateLine(size\_t length)

{

std::string lines;

for (size\_t i = 0; i < length; ++i) {

lines += "-";

}

return lines;

}

*Файл Commands.h*

#ifndef CROSS\_REFERENCES\_COMMANDS\_H

#define CROSS\_REFERENCES\_COMMANDS\_H

#include "TextProcessing.h"

#include <string>

void analyseText(const std::string& filename);

void ThreeMFWText(const std::string& filename);

void analyseFreq(const std::string& filename);

void printHelp();

void readCommand(std::string command);

#endif

*Файл Commands.cpp*

#include "Commands.h"

#include "TextProcessing.h"

#include <iostream>

#include <string>

#include <sstream>

#include <fstream>

void readCommand(std::string command) //чтение команды

{

std::istringstream iss(command);

std::string cmd;

std::string filename;

iss >> cmd;

if (cmd == "help")

{

printHelp();

std::cout << "\n> ";

}

else if (cmd == "analyse")

{

if (iss >> filename)

{

try

{

analyseText(filename);

std::cout << "\n> ";

}

catch (const std::invalid\_argument& e)

{

std::cerr << e.what() << '\n';

std::cout << "\n> ";

}

}

}

else if (cmd == "exit")

{

exit(0);

}

else if (cmd == "3mfw")

{

if (iss >> filename)

{

try

{

ThreeMFWText(filename);

std::cout << "\n> ";

}

catch (const std::invalid\_argument& e)

{

std::cerr << e.what() << '\n';

std::cout << "\n> ";

}

}

}

else if (cmd == "frequency")

{

if (iss >> filename)

{

try

{

analyseFreq(filename);

std::cout << "\n> ";

}

catch (const std::invalid\_argument& e)

{

std::cerr << e.what() << '\n';

std::cout << "\n> ";

}

}

}

}

void printHelp()

{

std::cout << "\n.-----------------------------------------------------------------------------------------------.\n";

std::cout << "| AVAILABLE COMMANDS | DESCRIPTION |\n";

std::cout << "|-----------------------------------------------------------------------------------------------|\n";

std::cout << "| help | display this help message |\n";

std::cout << "| analyse <filename> | print table of analysed text from <filename> (keys sorting) |\n";

std::cout << "| 3mfw <filename> | print 3 most frequent words |\n";

std::cout << "| frequency <filename> | print table of analysed frequency from <filename> (values sorting) |\n";

std::cout << "| exit | exit the program |\n";

std::cout << "'-----------------------------------------------------------------------------------------------'\n";

}

void ThreeMFWText(const std::string& filename)

{

TextHandler dict{ };

dict.analyse(filename);

std::cout << "\nChoose output options: \n";

std::cout << " -> terminal analyse 3 most frequent words to terminal\n";

std::cout << " -> <filename> analyse 3 most frequent words to file (warning: filename must include extension .txt)\n";

std::cout << "\n>> ";

std::string outfile;

std::cin >> outfile;

if (!std::cin)

{

throw std::invalid\_argument{ "Reading input error" };

}

if (outfile == "terminal")

{

std::cout << "\n";

dict.print3MFW(std::cout);

}

else

{

dict.print3MFW(outfile);

}

}

void analyseText(const std::string& filename)

{

TextHandler dict{ };

dict.analyse(filename);

std::cout << "\nChoose output options: \n";

std::cout << " -> terminal analyse text to terminal\n";

std::cout << " -> <filename> analyse text to file (warning: filename must include extension .txt)\n";

std::cout << "\n>> ";

std::string outfile;

std::cin >> outfile;

if (!std::cin)

{

throw std::invalid\_argument{ "Reading input error" };

}

if (outfile == "terminal")

{

std::cout << "\n";

dict.printStatistics(std::cout);

}

else

{

dict.printStatistics(outfile);

}

while (!dict.isDictEmpty())

{

std::cout << "Do you want to delete or insert word or nothing? (D/I/N)\n";

auto answer1 = std::string{ };

std::cin >> answer1;

if (!std::cin)

{

throw std::invalid\_argument{ "Reading input error" };

}

if (answer1 == "D")

{

std::cout << "\nEnter the word to delete: ";

auto word = std::string{};

std::cin >> word;

dict.deleteWord(word);

if (outfile == "terminal")

{

std::cout << "\n";

dict.printStatistics(std::cout);

}

else

{

dict.printStatistics(outfile);

}

}

else if (answer1 == "I")

{

std::cout << "\nEnter the word to insert: ";

auto word1 = std::string{};

std::cin >> word1;

std::cout << "Enter the amount of words: ";

int val;

std::cin >> val;

dict.insertWord(word1, val);

if (outfile == "terminal")

{

std::cout << "\n";

dict.printStatistics(std::cout);

}

else

{

dict.printStatistics(outfile);

}

}

else if (answer1 == "N")

{

break;

}

}

}

void analyseFreq(const std::string& filename)

{

TextHandler dict{ };

dict.analyse(filename);

std::cout << "\nChoose output options: \n";

std::cout << " -> terminal analyse frequency of words to terminal\n";

std::cout << " -> <filename> analyse frequency of words to file (warning: filename must include extension .txt)\n";

std::cout << "\n>> ";

std::string outfile;

std::cin >> outfile;

if (!std::cin)

{

throw std::invalid\_argument{ "Reading input error" };

}

if (outfile == "terminal")

{

std::cout << "\n";

dict.printFreqStat(std::cout);

}

else

{

dict.printFreqStat(outfile);

}

}

*Файл main.cpp*

#include "RedBlackTree.h"

#include "TextProcessing.h"

#include "Commands.h"

#include "RedBlackTree.cpp"

#include <string>

#include <iostream>

#include <fstream>

void tableTest()

{

TextHandler dict;

std::cout << "\n=============TABLE TEST=============\n\n";

try

{

dict.analyse("text0.txt");

dict.printStatistics(std::cout);

dict.printStatistics("out.txt");

}

catch (std::invalid\_argument& e)

{

std::cerr << e.what() << '\n';

}

std::cout << "=============DELETE WORD FROM TABLE TEST=============\n\n";

std::cout << "-> Deleting word \"advantages\":\n\n";

try

{

std::string word = "advantages";

dict.deleteWord(word);

dict.printStatistics(std::cout);

dict.printStatistics("out1.txt");

}

catch (std::invalid\_argument& e)

{

std::cerr << e.what() << '\n';

}

std::cout << "-> Deleting word \"networks\":\n\n";

try

{

std::string word = "networks";

dict.deleteWord(word);

dict.printStatistics(std::cout);

dict.printStatistics("out2.txt");

}

catch (std::invalid\_argument& e)

{

std::cerr << e.what() << '\n';

}

std::cout << "=============EXCEPTIONS TEST=============\n\n";

try

{

dict.analyse("nofile.txt"); //нет такого файла

dict.printStatistics(std::cout);

}

catch (std::invalid\_argument& e)

{

std::cerr << e.what() << '\n';

}

try

{

dict.analyse("text.txt");

dict.printStatistics("file"); //без расширения

}

catch (std::invalid\_argument& e)

{

std::cerr << e.what() << '\n';

}

try

{

dict.analyse("text.txt");

dict.printStatistics("/.txt"); //невозможно создать

}

catch (std::invalid\_argument& e)

{

std::cerr << e.what() << '\n';

}

}

int main()

{

setlocale(LC\_ALL, "ru");

tableTest();

std::cout << "\n=============USER CONSOLE=============\n";

std::cout << "\n> ";

std::string command;

while (std::cin)

{

std::getline(std::cin, command);

readCommand(command);

}

return 0;

}

# **Приложение 2. Протоколы отладки**

**Входные данные**

На вход программе может подаваться любой текстовой файл.

*Файл text.txt*

Advertising is everywhere being introduced into our lives, and is

already an integral part of it, going viral in various social networks.

Thus, in our society, the issue of advertising often causes discussions.

So let's talk about the advantages and disadvantages of advertising. Advertising

touches everything!

**Выходные данные**

**Команда help**

Данная команда выводит на экран список доступных команд для пользователя.

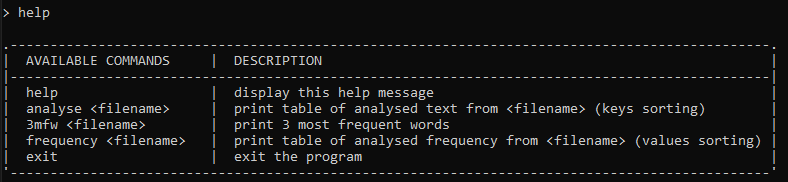


Рисунок . Команда help

**Команда exit**

Завершает программу.

**Команда analyse <filename>**

Данная команда формирует таблицу, в которой все слова будут расположены в алфавитном порядке и для каждого слова будет указано количество вхождений (частота встречаемости).

<filename> - исходный файл с текстом

* При попытке открыть для анализа несуществующий файл или невозможности открыть входной файл, будет выведено предупреждение. Пользователь сможет повторить ввод команды.



Рисунок . Ошибка отсутствия искомого файла

* При попытке указать выходной файл, но без требуемого расширения, будет выведено предупреждение. Пользователь сможет повторить ввод команды.

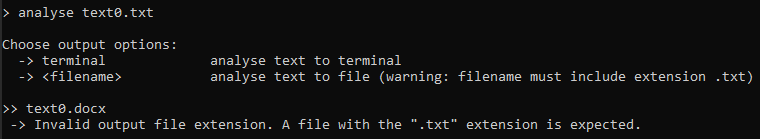


Рисунок . Ошибка расширения

* При попытке указать выходной файл, который невозможно создать, будет выведено предупреждение. Пользователь сможет повторить ввод команды.



Рисунок . Ошибка названия

* Если все команды введены правильно, исходный файл существует, и пользователь захотел вывести таблицу в терминал, то программа отработает корректно и выведет результат в окно консольного приложения.

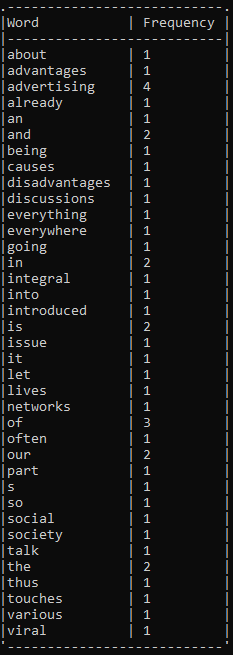


Рисунок . Корректный вывод таблицы

* Если все команды введены правильно, исходный файл существует, и пользователь захотел вывести таблицу в выходной файл (название которого указано в соответствии с требованиями), то программа отработает корректно и выведет результат в выходной файл.

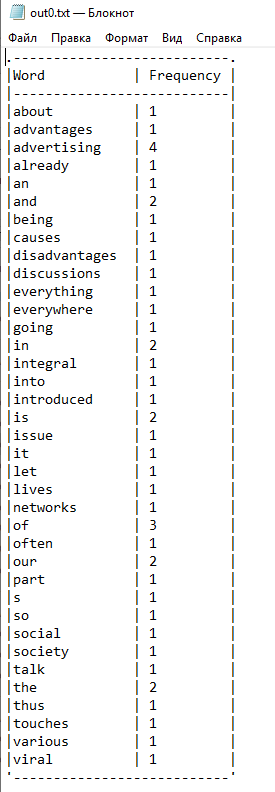


Рисунок . Корректный вывод в файл

* После вывода таблицы пользователю будет предложена опция удаления слов из словаря. Если пользователь хочет удалить слово из словаря, ему нужно написать D, а после написать слово, которое он хочет удалить.
* После вывода таблицы пользователю будет предложена опция вставки слов в словарь. Если пользователь хочет вставить слово в словарь, ему нужно написать I, а после написать слово, которое он хочет вставить и количество вхождений.

Если пользователь не хочет удалять или вставлять слово из словаря, ему нужно написать N, тогда команда завершится. Если пользователь хочет удалить слово, которого нет в словаре, команда предупредит его об этом. Данная опция будет до тех пор, пока словарь не будет пустым. Обновленный словарь запишется в выходной файл или снова выведется в окно консольного приложения.

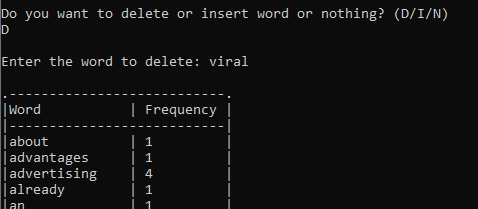


Рисунок . Удаление слова

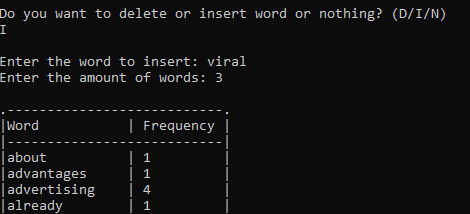


Рисунок . Вставка слова

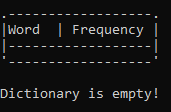


Рисунок . Пустой словарь

**Команда 3mfw <filename>**

Данная команда формирует таблицу, в которой отображаются три самых часто встречаемых слова в порядке невозрастания количества вхождений.

<filename> - исходный файл с текстом

Ошибки выводятся аналогичным образом прошлой команде.

Вывод осуществляется либо в терминал, либо в указываемый файл.

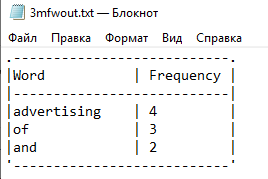
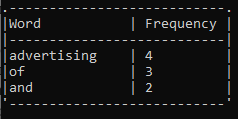


Рисунок . Вывод 3mfw в консоль

Рисунок . Вывод 3mfw в файл

**Команда frequency <filename>**

Данная команда формирует таблицу, в которой отображаются слова в порядке невозрастания количества вхождений.

<filename> - исходный файл с текстом

Ошибки выводятся аналогичным образом прошлой команде.

Вывод осуществляется либо в терминал, либо в указываемый файл.

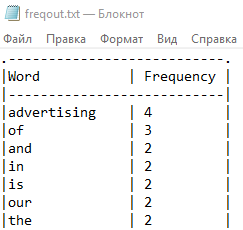
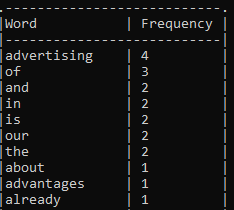


Рисунок . Вывод frequancy в файл

Рисунок . Вывод frequancy в консоль

**Список иллюстраций**

[Рисунок 1. Красно-чёрное дерево 4](#_Toc135797437)

[Рисунок 2. Правый и левый повороты 5](#_Toc135797438)

[Рисунок 3. Тест форматирования таблицы 12](#_Toc135797439)

[Рисунок 4. Тест удаления слов 13](#_Toc135797440)

[Рисунок 5. Тест исключений 13](#_Toc135797441)

[Рисунок 6. Команда help 37](#_Toc135797442)

[Рисунок 7. Ошибка отсутствия искомого файла 37](#_Toc135797443)

[Рисунок 8. Ошибка расширения 38](#_Toc135797444)

[Рисунок 9. Ошибка названия 38](#_Toc135797445)

[Рисунок 10. Корректный вывод таблицы 38](#_Toc135797446)

[Рисунок 11. Корректный вывод в файл 39](#_Toc135797447)

[Рисунок 12. Удаление слова 40](#_Toc135797448)

[Рисунок 13. Вставка слова 40](#_Toc135797449)

[Рисунок 14. Пустой словарь 40](#_Toc135797450)

[Рисунок 15. Вывод 3mfw в консоль 41](file:///D:\Загрузки\Telegram%20Desktop\Курсовая_работа_3530904_20001_КобаАЮ.docx#_Toc135797451)

[Рисунок 16. Вывод 3mfw в файл 41](#_Toc135797452)

[Рисунок 17. Вывод frequancy в файл 41](file:///D:\Загрузки\Telegram%20Desktop\Курсовая_работа_3530904_20001_КобаАЮ.docx#_Toc135797453)

[Рисунок 18. Вывод frequancy в консоль 41](file:///D:\Загрузки\Telegram%20Desktop\Курсовая_работа_3530904_20001_КобаАЮ.docx#_Toc135797454)