

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический университет
Петра Великого»

Институт компьютерных наук и кибербезопасности
Направление: 02.03.01 Математика и компьютерные науки

Теория графов
ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

Реализация словарей на основе хеш-таблицы и красно-чёрного дерева.
Кодирование информации

Вариант 23

Студент,
группы 5130201/30002

_____ Михайлова А. А.

Преподаватель

_____ Востров А. В.

«_____» _____ 2025 г.

Санкт-Петербург, 2025

Содержание

Введение	4
1 Математическое описание	5
1.1 Красно-чёрное дерево	5
1.2 Хеш-таблица	6
1.3 Алгоритм LZW	8
2 Особенности реализации	10
2.1 Класс RBT	10
2.1.1 Insert	11
2.1.2 Remove	14
2.1.3 Search	17
2.1.4 RotateLeft и RotateRight	18
2.1.5 Transplant	19
2.1.6 GetMinPodNode и GetMaxPodNode	19
2.1.7 Clear и ClearRecursiveFunction	20
2.1.8 Print и PrintPodTree	20
2.2 Класс HashTable	21
2.2.1 RehashTable и StringsEqual	22
2.2.2 Insert	23
2.2.3 Search	24
2.2.4 Remove	25
2.2.5 Clear	26
2.2.6 Print	26
2.3 Класс Dictionary	27
2.4 Класс LZW	29
2.4.1 compress	29
2.4.2 decompress	30
2.4.3 writeCompressedToFile	31
2.4.4 readCompressedFromFile	32
2.4.5 doubleCompress и doubleDecompress	33
2.5 compareStrings	34
2.6 toLowerRussian	35
2.7 isLetter	35
2.8 splitTextIntoWords	36
2.9 safeInputInt	36
2.10 dictionaryMenu	37
2.11 dataEncodingMenu	41
2.12 main	46
3 Результаты работы	48

Заключение	61
Список использованной литературы	62

Введение

В лабораторной работе требуется для выбранного текста на русском языке реализовать структуру данных - хеш-таблицу, для которой характерны операции: добавления, удаления и поиска. На ее основе реализовать словарь. Хеш-функцию выбрать произвольную. Добавить функцию полной очистки словаря и загрузки/дополнения словаря из текстового файла.

Для этого же текста на русском языке построить словарь на основе красно-чёрных деревьев. Реализовать функции добавления, удаления и поиска слова. Добавить функцию полной очистки словаря и загрузки или дополнения словаря из текстового файла.

Случайно сгенерировать файл в 10 тысяч символов в соответствии с распределением Эрланга, используя русский алфавит, цифры и спецсимволы. Закодировать текстовую информацию, используя алгоритм LZW. Декодировать информацию, определить коэффициент сжатия. Программно проверить, что декодирование произошло верно.

Закодировать информацию, применив двухступенчатое кодирование и декодирование. Показать, какой из способов более эффективный.

Применить алгоритм сжатия текстового файла и декодирование к исходному текстовому файлу со словарем.

Лабораторная работа выполнена на языке C++ в среде разработки Visual Studio 2022.

1 Математическое описание

1.1 Красно-чёрное дерево

Бинарное дерево - структура данных в форме иерархии, где каждый узел содержит определённое значение, которое служит ключом, а также ссылки на левого и правого ребенка. Вершина, расположенная на самом верхнем уровне и не являющаяся потомком других узлов, называется корнем. Узлы, у которых нет потомков (оба ребенка NULL) – листья.

Красно-черное дерево – разновидность бинарного дерева, баланс которого поддерживается за счёт окрашивания узлов в два цвета по определённым правилам:

- Каждый узел окрашен либо в красный, либо в чёрный цвет;
- Листьями считаются NULL-узлы («виртуальные» узлы, наследники обычных листьев, на которые «указывают» NULL-указатели), которые всегда чёрные;
- Если узел красный, то его оба потомка должны быть чёрными;
- На всех путях от корня дерева до его листьев количество чёрных узлов должно быть одинаковым.

Для данной работы реализованы операции вставки и удаления узла, поиска значения, считывания текста из файла, вывода дерева и очистки словаря.

Поиск в красно-черном дереве совпадает с поиском в обычном двоичном дереве. Так как каждый узел хранит информацию о своих поддеревьях, поиск выполняется рекурсивно, сравнивая искомый ключ с ключом текущего узла.

При вставке и удалении узлов происходит балансировка.

Основные операции балансировки:

- Изменение цвета узлов для восстановления свойств красно-чёрного дерева;
- Левое и правое вращения для изменения структуры дерева, с сохранением порядка элементов.

Удаление:

- Если удаляем красный узел, то балансировка не нужна;
- Если удаляем черный узел, то балансировка нужна для восстановления одинакового количества чёрных узлов на всех путях от корня до листьев.

Вставка:

- Новый узел – всегда красный;
- Проверяется нарушение свойств красно-черного дерева;
- Если свойства нарушены выполняем операции балансировки.

Пример красно-чёрного дерева представлен на рис. 1.

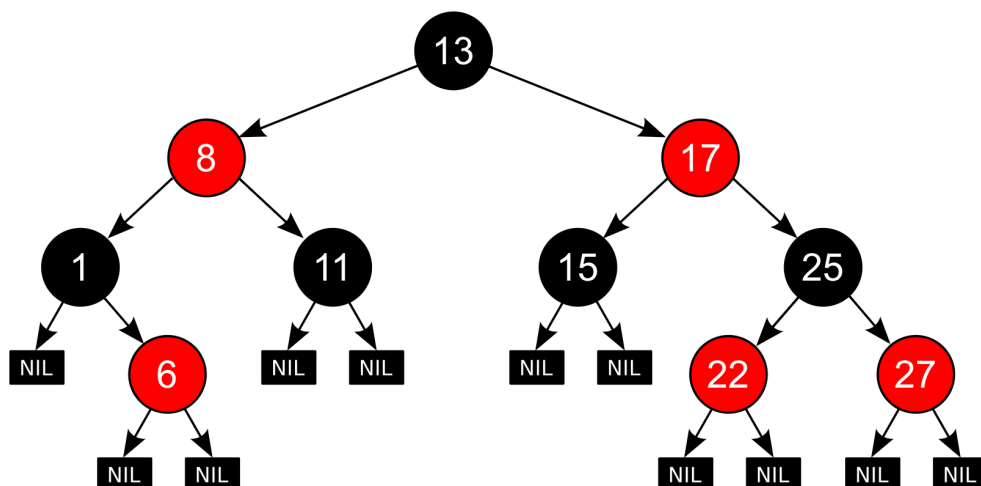


Рис. 1: Красно-чёрное дерево

1.2 Хеш-таблица

Хеш-таблица – это структура данных, используемая для хранения пар ключ-значение с быстрым доступом к элементам по их ключам. Она достигает высокой скорости благодаря применению хеш-функций, которые преобразуют ключи в индексы массива. Позволяет выполнять операции: добавление новой пары, удаление, поиск.

Данная реализация представляет собой хеш-таблицу с цепочками (метод разрешения коллизий через связанные списки).

- Каждая ячейка хеш-таблицы содержит указатель на связный список (цепочку) элементов;
- Если несколько ключей дают одинаковый хеш, они добавляются в один и тот же список.

В данной работе используется полиномиальная хеш-функция

$$H(x) = (\sum x_i * k^i) \bmod N$$

где $k = 37$ – число, которое является простым и обеспечивает наименьшее количество коллизий, большее, чем размер алфавита,

$N = 1000$ – изначальный размер таблицы.

Функция учитывает все символы строки, поэтому похожие ключи дают разные хеши. Она достаточно устойчива к коллизиям. Каждый символ влияет на конечный хеш, гарантирует попадание в диапазон таблицы.

Вставка элемента происходит так:

- Вычисляется хеш ключа для определения индекса;
- Проверяется, существует ли уже такой ключ в цепочке;
- Если ключ найден, значение обновляется;
- Если ключа нет, проверяется необходимость рехеширования (таблица заполнена больше, чем на 50%), создаётся новый узел и добавляется в начало цепочки;
- Фиксируется коллизия (если она была).

Поиск элемента происходит так:

- Вычисляется хеш ключа;
- Проходится цепочка по индексу;
- При нахождении совпадения ключей возвращается значение;
- Если ключ не найден, возвращается сообщение об этом.

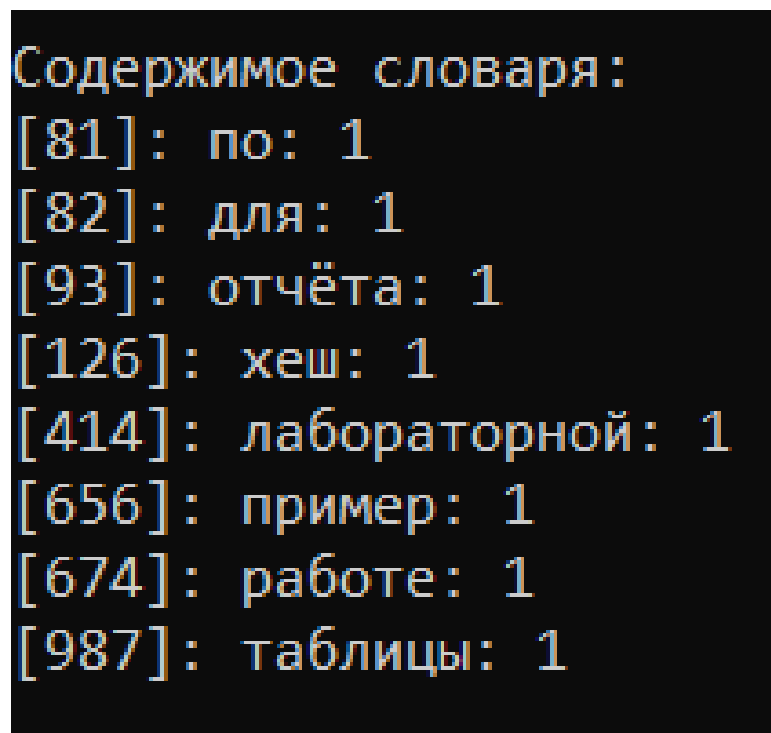
Удаление элемента происходит так:

- Вычисляется хеш ключа;
- Проходится цепочка с сохранением предыдущего узла;
- При нахождении ключа перестраиваются связи и узел удаляется.
- Если ключ не найден, возвращается сообщение об этом.

Рехеширование происходит, если таблица заполнена больше, чем на 50%. Создаётся новая таблица в 2 раза больше. Все элементы из старой таблицы пересчитываются и переносятся в новую. Старая таблица удаляется. Обновляются счётчики коллизий.

Эта реализация обеспечивает доступ к данным в среднем за $O(1)$ для основных операций при хорошем распределении хеш-функции.

Пример хеш-таблицы представлен на рис. 2.



Содержимое словаря:		
[81]:	по:	1
[82]:	для:	1
[93]:	отчёта:	1
[126]:	хеш:	1
[414]:	лабораторной:	1
[656]:	пример:	1
[674]:	работе:	1
[987]:	таблицы:	1

Рис. 2: Хеш-таблица

1.3 Алгоритм LZW

Этот метод позволяет достичь одну из наилучших степеней сжатия среди других существующих методов сжатия графических данных, при полном отсутствии потерь или искажений в исходных файлах.

Процесс сжатия выглядит следующим образом: последовательно считываются символы входного потока и происходит проверка, существует ли в созданной таблице строк такая строка. Если такая строка существует, считывается следующий символ, а если строка не существует, в поток заносится код для предыдущей найденной строки, строка заносится в таблицу, а поиск начинается снова.

Например, если сжимают байтовые данные (текст), то строк в таблице окажется 256 (от "0" до "255"). Новые строки формируют таблицу последовательно, т.е. можно считать индекс строки ее кодом.

Для декодирования на вход подается только закодированный текст, поскольку алгоритм LZW может воссоздать соответствующую таблицу преобразования непосредственно по закодированному тексту. Алгоритм генерирует однозначно декодируемый код за счет того, что каждый раз, когда генерируется новый код, новая строка добавляется в таблицу строк. LZW постоянно проверяет, является ли строка уже известной, и, если так, выводит существующий код без генерации нового. Таким образом, каждая строка будет храниться в единственном экземпляре и иметь свой уникальный номер. Следовательно, при декодировании во время получения нового кода генерируется

новая строка, а при получении уже известного, строка извлекается из словаря.

Кодирование:

1. Все возможные символы заносятся в словарь. Во входную фразу X заносится первый символ сообщения;
2. Считать очередной символ Y из сообщения.
3. Если Y – это символ конца сообщения, то выдать код для X , иначе:
 - Если фраза XY уже имеется в словаре, то присвоить входной фразе значение XY и перейти к шагу 2,
 - Иначе выдать код для входной фразы X , добавить XY в словарь и присвоить входной фразе значение Y . Перейти к шагу 2.

2 Особенности реализации

2.1 Класс RBT

Класс RBT реализует красно-чёрное дерево. Каждый узел хранит строковый ключ (key), целочисленное значение (value), цвет (red_color), а также указатели на родителя и потомков.

```
1 class RBT {
2 private:
3     struct TreeNode {
4         char* key;
5         int value;
6         bool red_color;
7
8         TreeNode* left_node;
9         TreeNode* right_node;
10        TreeNode* parent_node;
11
12        TreeNode(const char* k, int v);
13        ~TreeNode();
14    };
15
16    TreeNode* main_node;
17    long long items_count;
18
19    void RotateRight(TreeNode* node);
20    void RotateLeft(TreeNode* node);
21
22    TreeNode* SearchNode(const char* key);
23
24    void Transplant(TreeNode* first, TreeNode* second);
25    TreeNode* GetMinPodNode(TreeNode* node);
26    TreeNode* GetMaxPodNode(TreeNode* node);
27
28    void ClearRecursiveFunction(TreeNode* node);
29
30    void PrintPodTree(const TreeNode* node, int level,
31        char branch);
32
33    void ForEachRecursive(TreeNode* node, void (*
34        callback)(const char*, int, void*), void* userData);
35
36 public:
```

```

35     RBT();
36     ~RBT();
37
38     void Insert(const char* key, int value);
39     int Search(const char* key);
40     bool Remove(const char* key);
41     void Clear();
42     void Print();
43
44     void ForEach(void (*callback)(const char*, int,
45 void*), void* userData);
};

```

2.1.1 Insert

Метод Insert добавляет новый узел с заданным ключом и значением в дерево. Сначала он ищет место для вставки, следуя правилам бинарного дерева поиска. После вставки дерево перебалансируется за счёт перекрашивания узлов и выполнения поворотов (RotateLeft/RotateRight), чтобы сохранить свойства красно-чёрного дерева.

Вход: ключ, значение

Выход: вставленный узел

```

1     void RBT::Insert(const char* key, int value)
2 {
3     TreeNode* node = new TreeNode(key, value);
4     TreeNode* first_node = main_node;
5     TreeNode* second_node = nullptr;
6
7     while (first_node != nullptr) {
8         second_node = first_node;
9
10        if (strcmp(node->key, first_node->key) > 0)
first_node = first_node->right_node;
11        else if (strcmp(node->key, first_node->key) <
0) first_node = first_node->left_node;
12        else {
13            first_node->value = value;
14            delete node;
15            return;
16        }
17    }
18 }

```

```

19     node->parent_node = second_node;
20     items_count++;
21
22     if (second_node == nullptr) main_node = node;
23     else if (strcmp(node->key, second_node->key) < 0)
second_node->left_node = node;
24     else second_node->right_node = node;
25
26     if (node->parent_node == nullptr) {
27         node->red_color = false;
28         return;
29     }
30
31     if (!(node->parent_node->parent_node == nullptr)) {
32         TreeNode* temp_node;
33
34         while (node->parent_node != nullptr && node->
parent_node->red_color) {
35             if (node->parent_node == node->parent_node
->parent_node->right_node) {
36                 temp_node = node->parent_node->
parent_node->left_node;
37
38                 if (temp_node != nullptr && temp_node->
red_color) {
39                     temp_node->red_color = false;
40                     node->parent_node->red_color =
false;
41                     node->parent_node->parent_node->
red_color = true;
42                     node = node->parent_node->
parent_node;
43                 }
44                 else {
45                     if (node == node->parent_node->
left_node) {
46                         node = node->parent_node;
47                         RotateRight(node);
48                     }
49
50                     node->parent_node->red_color =
false;
51                     node->parent_node->parent_node->

```

```

52     red_color = true;
53         RotateLeft(node->parent_node->
54     parent_node);
55     }
56     else {
57         temp_node = node->parent_node->
58     parent_node->right_node;
59
60         if (temp_node != nullptr && temp_node->
61     red_color) {
62             temp_node->red_color = false;
63             node->parent_node->red_color =
64     false;
65             node->parent_node->parent_node->
66     red_color = true;
67             node = node->parent_node->
68     parent_node;
69         }
70         else {
71             if (node == node->parent_node->
72     right_node) {
73                 node = node->parent_node;
74                 RotateLeft(node);
75             }
76             node->parent_node->red_color =
77     false;;
78             node->parent_node->parent_node->
79     red_color = true;
80             RotateRight(node->parent_node->
81     parent_node);
82         }
83     }
84     if (node == main_node) break;
85 }
86     main_node->red_color = false;
87 }
88 }

```

2.1.2 Remove

Метод Remove удаляет узел с заданным ключом. Сначала он находит узел с помощью SearchNode, затем в зависимости от количества потомков выполняет удаление (используя вспомогательную функцию Transplant).

Вход: ключ

Выход: удалённый узел

```
1  bool RBT::Remove(const char* key)
2  {
3      TreeNode* node = SearchNode(key);
4      if (node == nullptr) return false;
5
6      TreeNode* first_node;
7      TreeNode* second_node = node;
8
9      bool second_original_red_color = second_node->
red_color;
10
11     if (node->left_node == nullptr) {
12         first_node = node->right_node;
13         Transplant(node, node->right_node);
14     }
15     else if (node->right_node == nullptr) {
16         first_node = node->left_node;
17         Transplant(node, node->left_node);
18     }
19     else {
20         second_node = GetMinPodNode(node->right_node);
21         second_original_red_color = second_node->
red_color;
22         first_node = second_node->right_node;
23
24         if (second_node->parent_node == node) {
25             if (first_node != nullptr) first_node->
parent_node = second_node;
26         }
27         else {
28             Transplant(second_node, second_node->
right_node);
29             second_node->right_node = node->right_node;
30             second_node->right_node->parent_node =
second_node;
31         }
    }
```

```

32         Transplant(node, second_node);
33         second_node->left_node = node->left_node;
34         second_node->left_node->parent_node =
35 second_node;
36         second_node->red_color = node->red_color;
37     }
38
39     delete node;
40     items_count--;
41
42     if (!second_original_red_color && first_node !=
43 nullptr) {
44         TreeNode* s;
45
46         while (first_node != main_node && (first_node
47 == nullptr || !first_node->red_color)) {
48             if (first_node == first_node->parent_node->
49 left_node) {
50                 s = first_node->parent_node->right_node
51 ;
52
53                 if (s->red_color) {
54                     s->red_color = false;
55                     first_node->parent_node->red_color
56 = true;
57                     RotateLeft(first_node->parent_node)
58 ;
59                     s = first_node->parent_node->
60 right_node;
61                 }
62
63                 if ((s->left_node == nullptr || !s->
64 left_node->red_color) &&
65                     (s->right_node == nullptr || !s->
66 right_node->red_color)) {
67                     s->red_color = true;
68                     first_node = first_node->
69 parent_node;
70                 }
71                 else {
72                     if (s->right_node == nullptr || !s
73 ->right_node->red_color) {

```

```

63         if (s->left_node != nullptr) {
64             s->left_node->red_color =
false;
65         }
66         s->red_color = true;
67         RotateRight(s);
68         s = first_node->parent_node->
right_node;
69     }
70
71     s->red_color = first_node->
parent_node->red_color;
72     first_node->parent_node->red_color
= false;
73     if (s->right_node != nullptr) {
74         s->right_node->red_color =
false;
75     }
76     RotateLeft(first_node->parent_node)
;
77     first_node = main_node;
78 }
79 }
80 else {
81     s = first_node->parent_node->left_node;
82
83     if (s->red_color) {
84         s->red_color = false;
85         first_node->parent_node->red_color
= true;
86         RotateRight(first_node->parent_node
);
87         s = first_node->parent_node->
left_node;
88     }
89
90     if ((s->right_node == nullptr || !s->
right_node->red_color) &&
91         (s->left_node == nullptr || !s->
left_node->red_color)) {
92         s->red_color = true;
93         first_node = first_node->
parent_node;

```



```

94         }
95         else {
96             if (s->left_node == nullptr || !s->
left_node->red_color) {
97                 if (s->right_node != nullptr) {
98                     s->right_node->red_color =
false;
99                 }
100                 s->red_color = true;
101                 RotateLeft(s);
102                 s = first_node->parent_node->
left_node;
103             }
104
105             s->red_color = first_node->
parent_node->red_color;
106             first_node->parent_node->red_color
= false;
107             if (s->left_node != nullptr) {
108                 s->left_node->red_color = false
;
109             }
110             RotateRight(first_node->parent_node
);
111             first_node = main_node;
112         }
113     }
114 }
115
116 if (first_node != nullptr) {
117     first_node->red_color = false;
118 }
119 }
120
121 return true;
122 }

```

2.1.3 Search

Метод Search ищет узел по ключу и возвращает его значение. Внутри он использует SearchNode, который обходит дерево, сравнивая ключи. Если узел не найден, возвращается ноль.

Вход: ключ

Выход: найденный узел или 0

```
1  int RBT::Search(const char* key)
2  {
3      TreeNode* node = SearchNode(key);
4      return node ? node->value : 0;
5  }
```

2.1.4 RotateLeft и RotateRight

Методы RotateLeft и RotateRight выполняют левый и правый повороты поддерева вокруг заданного узла. Они используются при балансировке дерева после вставки или удаления.

Вход: узел

Выход: левый или правый повороты поддерева

```
1  void RBT::RotateRight(TreeNode* node)
2  {
3      TreeNode* x = node->left_node;
4      node->left_node = x->right_node;
5
6      if (x->right_node != nullptr) x->right_node->
parent_node = node;
7      x->parent_node = node->parent_node;
8
9      if (node->parent_node == nullptr) main_node = x;
10     else if (node == node->parent_node->right_node)
node->parent_node->right_node = x;
11     else node->parent_node->left_node = x;
12
13     x->right_node = node;
14     node->parent_node = x;
15 }
16
17 void RBT::RotateLeft(TreeNode* node)
18 {
19     TreeNode* y = node->right_node;
20     node->right_node = y->left_node;
21
22     if (y->left_node != nullptr) y->left_node->
parent_node = node;
23     y->parent_node = node->parent_node;
24 }
```

```

25     if (node->parent_node == nullptr) main_node = y;
26     else if (node == node->parent_node->left_node) node
->parent_node->left_node = y;
27     else node->parent_node->right_node = y;
28
29     y->left_node = node;
30     node->parent_node = y;
31 }

```

2.1.5 Transplant

Метод Transplant заменяет одно поддерево другим, обновляя связи между узлами. Он используется при удалении.

Вход: первое и второе поддерево

Выход: замененное поддерево

```

1     void RBT::Transplant(TreeNode* first, TreeNode*
second)
2 {
3     if (first->parent_node == nullptr) main_node =
second;
4     else if (first == first->parent_node->left_node)
first->parent_node->left_node = second;
5     else first->parent_node->right_node = second;
6
7     if (second != nullptr) second->parent_node = first
->parent_node;
8 }

```

2.1.6 GetMinPodNode и GetMaxPodNode

Методы GetMinPodNode и GetMaxPodNode находят узел с минимальным или максимальным ключом в поддереве, начиная с заданного узла.

Вход: узел

Выход: узел с минимальным или максимальным ключом в поддереве

```

1 RBT::TreeNode* RBT::GetMinPodNode(TreeNode* node) {
2     while (node->left_node != nullptr) {
3         node = node->left_node;
4     }
5     return node;
6 }
7
8 RBT::TreeNode* RBT::GetMaxPodNode(TreeNode* node) {

```

```

9      while (node->right_node != nullptr) {
10          node = node->right_node;
11      }
12      return node;
13  }

```

2.1.7 Clear и ClearRecursiveFunction

Метод Clear рекурсивно удаляет все узлы дерева, используя ClearRecursiveFunction, и обнуляет корневой узел и счётчик элементов.

Вход: узел

Выход: очищенное дерево

```

1  void RBT::Clear()
2  {
3      items_count = 0;
4      ClearRecursiveFunction(main_node);
5      main_node = nullptr;
6  }
7  void RBT::ClearRecursiveFunction(TreeNode* node) {
8      if (node == nullptr) return;
9      ClearRecursiveFunction(node->left_node);
10     ClearRecursiveFunction(node->right_node);
11     delete node;
12 }

```

2.1.8 Print и PrintPodTree

Метод Print выводит дерево в консоль в виде ASCII-графики, используя PrintPodTree. Узлы отображаются с отступами в зависимости от уровня, при этом красные узлы выделяются цветом.

Вход: узел, уровень, ветка

Выход: выведенное в консоль дерево

```

1  void RBT::PrintPodTree(const RBT::TreeNode* node, int
2      level, char branch) {
3      if (!node) return;
4
5      PrintPodTree(node->right_node, level + 1, '/');
6
7      for (int i = 0; i < level; i++) {
8          std::cout << "░░░░░";
9      }
10     std::cout << branch << "--░";

```

```

10
11     if (node->red_color) {
12         std::cout << "\033[1;31m";
13     }
14     else {
15         std::cout << "\033[1;30m";
16     }
17
18     std::cout << node->key << ":\u" << node->value << "
\033[0m" << std::endl;
19
20     PrintPodTree(node->left_node, level + 1, '\\');
21 }
22
23 void RBT::Print()
24 {
25     if (main_node) PrintPodTree(main_node, 0, '\u');
26 }

```

2.2 Класс HashTable

Класс HashTable реализует хеш-таблицу. Структура ListNode реализует одну ячейку цепочки.

```

1 class HashTable {
2 private:
3     struct ListNode {
4         char* key;
5         int value;
6         ListNode* node;
7
8         ListNode(const char* k, int v);
9         ~ListNode();
10    };
11
12    ListNode** hash_table;
13    long long table_size;
14    long long items_count;
15
16    long long collisions_count;
17    std::vector<long long> collision_indices;
18 }

```

```

19         unsigned long long HashFunction(const char* key
, unsigned long long p = 37);
20         void RehashTable();
21
22         bool StringsEqual(const char* str1, const char*
str2);
23
24 public:
25         HashTable(long long size);
26         ~HashTable();
27
28         void Insert(const char* key, int value);
29         int Search(const char* key);
30         bool Remove(const char* key);
31         void Clear();
32         void Print();
33
34         void ForEach(void (*callback)(const char*, int,
void*), void* userData);
35 };

```

2.2.1 RehashTable и StringsEqual

Метод RehashTable увеличивает размер таблицы в два раза при достижении коэффициента заполнения 0.5, создает новый массив и пересчитывает индексы всех элементов, сохраняя информацию о новых коллизиях. Вспомогательная функция StringsEqual сравнивает две строки с помощью strcmp.

Вход: заполнение таблицы на 50%

Выход: увеличенная в 2 раза таблица.

Вход: строка 1, строка 2

Выход: результат сравнения строк

```

1 void HashTable::RehashTable() {
2     items_count = 0;
3     collisions_count = 0;
4     collision_indices.clear();
5
6     long long old_size = table_size;
7     ListNode** old_table = hash_table;
8
9     this->table_size *= 2;
10    hash_table = new ListNode * [table_size];
11

```

```

12     for (long long i = 0; i < table_size; i++) {
13         hash_table[i] = nullptr;
14     }
15
16     for (long long i = 0; i < old_size; i++) {
17         ListNode* bucket = old_table[i];
18         while (bucket) {
19             ListNode* node = bucket->node;
20             bucket->node = nullptr;
21
22             long long newIndex = HashFunction(bucket->
key);
23             if (hash_table[newIndex] != nullptr) {
24                 collisions_count++;
25                 collision_indices.push_back(newIndex);
26             }
27
28             bucket->node = hash_table[newIndex];
29             hash_table[newIndex] = bucket;
30             items_count++;
31
32             bucket = node;
33         }
34         old_table[i] = nullptr;
35     }
36
37     delete[] old_table;
38 }
39
40 bool HashTable::StringsEqual(const char* str1, const
char* str2) {
41     if (!str1 || !str2) return false;
42     return strcmp(str1, str2) == 0;
43 }

```

2.2.2 Insert

Метод Insert добавляет новый элемент или обновляет существующий, при необходимости вызывая RehashTable.

Вход: ключ, значение

Выход: вставленный новый элемент

```

1  void HashTable::Insert(const char* key, int value)
2  {
3      long long index = HashFunction(key);
4      ListNode* bucket = hash_table[index];
5
6      while (bucket) {
7          if (StringsEqual(bucket->key, key)) {
8              bucket->value = value;
9              return;
10             }
11             bucket = bucket->node;
12         }
13
14         if (static_cast<float>(items_count + 1) /
15         table_size > 0.5) {
16             RehashTable();
17             index = HashFunction(key);
18         }
19
20         ListNode* new_bucket = new ListNode(key, value);
21
22         if (hash_table[index] != nullptr) {
23             collisions_count++;
24             collision_indices.push_back(index);
25         }
26
27         new_bucket->node = hash_table[index];
28         hash_table[index] = new_bucket;
29         items_count++;
30     }

```

2.2.3 Search

Поиск Search находит элемент по ключу и возвращает его значение или 0, если элемент не найден.

Вход: ключ

Выход: найденный элемент

```

1  int HashTable::Search(const char* key) {
2      long long index = HashFunction(key);
3      ListNode* bucket = hash_table[index];
4
5      while (bucket) {

```



```

6         if (StringsEqual(bucket->key, key)) return
bucket->value;
7         bucket = bucket->node;
8     }
9
10    return 0;
11}

```

2.2.4 Remove

Удаление Remove ищет и удаляет элемент по ключу, корректируя связи в цепочке.

Вход: ключ

Выход: удаленный элемент

```

1    bool HashTable::Remove(const char* key) {
2        long long index = HashFunction(key);
3        ListNode* bucket = hash_table[index];
4        ListNode* prev = nullptr;
5
6        while (bucket) {
7            if (StringsEqual(bucket->key, key)) {
8                if (prev) {
9                    prev->node = bucket->node;
10               }
11               else {
12                   hash_table[index] = bucket->node;
13               }
14
15               bucket->node = nullptr;
16               delete bucket;
17               items_count--;
18               return true;
19           }
20
21           prev = bucket;
22           bucket = bucket->node;
23       }
24       return false;
25   }

```

2.2.5 Clear

Метод Clear полностью очищает таблицу, удаляя все узлы и сбрасывая счетчики.

Вход: запрос на очищение таблицы

Выход: очищенная таблица

```
1 void HashTable::Clear() {
2   for (long long i = 0; i < table_size; i++) {
3       if (hash_table[i]) {
4           delete hash_table[i];
5           hash_table[i] = nullptr;
6       }
7   }
8   items_count = 0;
9   collisions_count = 0;
10  collision_indices.clear();
11 }
```

2.2.6 Print

Print выводит содержимое таблицы и статистику, включая количество элементов, коллизий и индексы коллизий.

Вход: запрос на вывод словаря

Выход: словарь в консоли

```
1 void HashTable::Print() {
2   for (long long i = 0; i < table_size; i++) {
3       if (hash_table[i] != nullptr) {
4           cout << "[" << i << "]:_";
5           ListNode* bucket = hash_table[i];
6           while (bucket) {
7               cout << bucket->key << ":_ " << bucket->
value;
8               if (bucket->node) cout << "_-->_";
9               bucket = bucket->node;
10          }
11          cout << endl;
12      }
13  }
14
15  cout << "\Статистика\n";
16  cout << "Количество_элементов:_ " << items_count <<
endl;
```

```

17     cout << "Количество_коллизий:_" << collisions_count <<
endl;
18
19     std::sort(collision_indices.begin(),
collision_indices.end());
20     collision_indices.erase(std::unique(
collision_indices.begin(), collision_indices.end()),
collision_indices.end());
21
22     if (!collision_indices.empty()) {
23         vector<long long> sorted_collisions =
collision_indices;
24         sort(sorted_collisions.begin(),
sorted_collisions.end());
25
26         cout << "Коллизии_произошли_в_ячейках:_" ;
27         for (size_t i = 0; i < sorted_collisions.size()
; i++) {
28             cout << sorted_collisions[i];
29             if (i != sorted_collisions.size() - 1) cout
<< ",_";
30         }
31         cout << endl;
32     }
33 }

```

2.3 Класс Dictionary

Класс реализует обобщённый словарь – представляет интерфейс для работы с HashTable и RBT. Также в нем реализованы дополнительные функции для работы с файлами.

ExportToFile – сохраняет словарь в файл.

Вход: имя файла

Выход: сохраненный в файле словарь

ImportFromFile – загружает словарь из файла.

Вход: имя файла

Выход: импортированный из файла словарь

```

1 class Dictionary {
2 private:
3     DataStruct data;
4
5 public:

```

```

6      Dictionary() : data() {}
7      Dictionary(long long size) : data(size) {}
8
9      void Insert(const char* key, int value) {
10         data.Insert(key, value);
11     }
12
13     int Search(const char* key) {
14         return data.Search(key);
15     }
16
17     bool Remove(const char* key) {
18         return data.Remove(key);
19     }
20
21     void Clear() {
22         data.Clear();
23     }
24
25     void Print() {
26         data.Print();
27     }
28
29     bool ExportToFile(const char* filename) {
30         std::ofstream file(filename);
31         if (!file.is_open()) return false;
32
33         struct ExportData {
34             std::ofstream& file;
35         } exportData{ file };
36
37         auto callback = [](const char* key, int value,
38 void* userData) {
39             ExportData* data = static_cast<ExportData
*>(userData);
40             data->file << key << ":" << value << "\n";
41         };
42
43         data.ForEach(callback, &exportData);
44
45         file.close();
46         return true;
47     }

```

```

47
48     bool ImportFromFile(const char* filename) {
49         std::ifstream file(filename);
50         if (!file.is_open()) return false;
51
52         Clear();
53
54         std::string line;
55         while (std::getline(file, line)) {
56             size_t colon_pos = line.find(':');
57             if (colon_pos != std::string::npos) {
58                 std::string key = line.substr(0,
colon_pos);
59                 int value = std::stoi(line.substr(
colon_pos + 1));
60                 Insert(key.c_str(), value);
61             }
62         }
63
64         file.close();
65         return true;
66     }
67 };

```

2.4 Класс LZW

Класс реализует алгоритм сжатия LZW.

2.4.1 compress

Метод compress выполняет сжатие входной строки.

Вход: входная строка

Выход: строка после сжатия

```

1         static std::vector<int> compress(const std::
string& input) {
2             std::map<std::string, int> dictionary;
3             for (int i = 0; i < chars.size(); i++) {
4                 dictionary[std::string(1, chars[i])] = i;
5             }
6
7             int dict_size = chars.size();
8             std::vector<int> result;

```

```

9      std::string current;
10     for (int i = 0; i < input.size(); i++) {
11         char c = input[i];
12         std::string current_c = current + c;
13         if (dictionary.count(current_c)) {
14             current = current_c;
15         }
16         else {
17             result.push_back(dictionary[current]);
18             dictionary[current_c] = dict_size++;
19             current = std::string(1, c);
20         }
21     }
22
23     if (!current.empty()) {
24         result.push_back(dictionary[current]);
25     }
26     return result;
27 }

```

2.4.2 decompress

Метод decompress выполняет обратное преобразование, восстанавливая исходную строку из сжатых данных.

Вход: сжатая строка

Выход: исходная строка

```

1      static std::string decompress(const std::vector<int
>& compressed) {
2      std::map<int, std::string> dictionary;
3      for (int i = 0; i < chars.size(); i++) {
4          dictionary[i] = std::string(1, chars[i]);
5      }
6
7      std::string current = dictionary[compressed[0]];
8      std::string result = current;
9      for (size_t i = 1; i < compressed.size(); i++) {
10         int code = compressed[i];
11         std::string entry;
12         if (dictionary.count(code)) {
13             entry = dictionary[code];
14         }
15         else if (code == dictionary.size()) {

```

```

16         entry = current + current[0];
17     }
18     else {
19         throw std::runtime_error("Bad_compressed_
code");
20     }
21
22     result += entry;
23     dictionary[dictionary.size()] = current + entry
[0];
24     current = entry;
25 }
26 return result;
27 }

```

2.4.3 writeCompressedToFile

Для записи сжатых данных в файл используется метод writeCompressedToFile.

Вход: сжатые данные

Выход: имя файла

```

1     static void writeCompressedToFile(const std::vector
<int>& compressed, const std::string& filename) {
2         int max_code = *std::max_element(compressed.begin()
, compressed.end());
3         int bits_per_code = std::ceil(std::log2(max_code +
1));
4
5         std::ofstream file(filename, std::ios::binary);
6         if (!file) {
7             throw std::runtime_error("Ошибка_открытия_файла");
8         }
9
10        file.write(reinterpret_cast<const char*>(&
bits_per_code), sizeof(bits_per_code));
11
12        unsigned char buffer = 0;
13        int bit_pos = 0;
14
15        for (int code : compressed) {
16            for (int i = 0; i < bits_per_code; i++) {
17                int bit = (code >> (bits_per_code - 1 - i))
& 1;

```

```

18         buffer |= (bit << (7 - bit_pos));
19         bit_pos++;
20
21         if (bit_pos == 8) {
22             file.write(reinterpret_cast<const char
*>(&buffer), sizeof(buffer));
23             buffer = 0;
24             bit_pos = 0;
25         }
26     }
27 }
28
29 if (bit_pos > 0) {
30     file.write(reinterpret_cast<const char*>(&
buffer), sizeof(buffer));
31 }
32 }

```

2.4.4 readCompressedFromFile

Метод readCompressedFromFile реализует чтение сжатых данных из файла.

Вход: имя файла

Выход: прочитанные данные из файла

```

1     static std::vector<int> readCompressedFromFile(
const std::string& filename) {
2         std::ifstream file(filename, std::ios::binary);
3         if (!file) {
4             throw std::runtime_error("Невозможно открыть файл"
);
5         }
6
7         int bits_per_code;
8         file.read(reinterpret_cast<char*>(&bits_per_code),
sizeof(bits_per_code));
9
10        std::vector<int> result;
11        unsigned char buffer;
12        int bits_available = 0;
13        int current_code = 0;
14        int bits_read = 0;
15

```



```

16     while (file.read(reinterpret_cast<char*>(&buffer),
sizeof(buffer))) {
17         for (int i = 7; i >= 0; i--) {
18             int bit = (buffer >> i) & 1;
19             current_code = (current_code << 1) | bit;
20             bits_read++;
21
22             if (bits_read == bits_per_code) {
23                 result.push_back(current_code);
24                 current_code = 0;
25                 bits_read = 0;
26             }
27         }
28     }
29
30     return result;
31 }

```

2.4.5 doubleCompress и doubleDecompress

Методы doubleCompress и doubleDecompress предоставляют возможность двухступенчатого кодирования и декодирования.

Вход: строка

Выход: строка после двухступенчатого кодирования

Вход: закодированная строка

Выход: декодированная строка

```

1     static std::vector<int> doubleCompress(const std::
string& input) {
2         std::vector<int> firstStage = compress(input);
3
4         std::string secondInput;
5         for (int code : firstStage) {
6
7             std::string codeStr = std::to_string(code);
8             secondInput += codeStr + " ";
9         }
10
11
12         return compress(secondInput);
13     }
14 static std::string doubleDecompress(const std::vector<
int>& compressed) {

```

```

15
16     std::string secondOutput = decompress(compressed);
17
18
19     std::vector<int> firstStage;
20     std::istringstream iss(secondOutput);
21     std::string token;
22     while (iss >> token) {
23         firstStage.push_back(std::stoi(token));
24     }
25
26
27     return decompress(firstStage);
28 }

```

2.5 compareStrings

Функция compareStrings сравнивает две строки поэлементно и возвращает true, если они идентичны, и false в противном случае.

Вход: строка 1, строка 2

Выход: true или false

```

1     bool compareStrings(const std::string& str1, const
std::string& str2) {
2         if (str1.length() != str2.length()) {
3             return false;
4         }
5
6         for (size_t i = 0; i < str1.length(); ++i) {
7             if (str1[i] != str2[i]) {
8                 std::cout << "Difference at position " << i
<< " : "
9                 << str1[i] << " vs " << str2[i] << "
'\n";
10                return false;
11            }
12        }
13
14        return true;
15    }

```

2.6 toLowerRussian

Функция toLowerRussian предназначена для преобразования русских и английских букв в строке в нижний регистр с учетом особенностей кириллицы.

Вход: строка

Выход: преобразованная строка

```
1 string toLowerRussian(const string& str) {
2     string result;
3     for (char ch : str) {
4         if (ch >= 'А' && ch <= 'Я') {
5             result += ch + ('a' - 'A');
6         }
7         else if (ch == 'Ё') {
8             result += 'ё';
9         }
10        else if (isalpha(ch)) {
11            result += tolower(ch);
12        }
13        else {
14            result += ch;
15        }
16    }
17    return result;
18 }
```

2.7 isLetter

Функция isLetter проверяет, является ли символ ch буквой русского или английского алфавита в любом регистре.

Вход: буква

Выход: буква в нужном регистре

```
1 bool isLetter(char ch) {
2     return (ch >= 'А' && ch <= 'Z') || (ch >= 'a' && ch
3         <= 'z') ||
4         (ch >= 'А' && ch <= 'я') || ch == 'ё' || ch ==
5         'Ё';
6 }
```

2.8 splitTextIntoWords

Функция splitTextIntoWords разбивает входной текст на слова, используя критерии символов и применяя нормализацию регистра.

Вход: текст

Выход: слова

```
1 vector<string> splitTextIntoWords(const string& text) {
2     vector<string> words;
3     string word;
4     for (char ch : text) {
5         if (isLetter(ch)) {
6             word += ch;
7         }
8         else if (!word.empty()) {
9             words.push_back(toLowerRussian(word));
10            word.clear();
11        }
12    }
13    if (!word.empty()) {
14        words.push_back(toLowerRussian(word));
15    }
16    return words;
17 }
```

2.9 safeInputInt

Функция safeInputInt реализует безопасный ввод целого числа с обработкой ошибок.

Вход: ввод целого числа

Выход: целое число

```
1 int safeInputInt() {
2     int value;
3     while (true) {
4         cin >> value;
5         if (cin.fail()) {
6             cin.clear();
7             cin.ignore(numeric_limits<streamsize>::max(), '\n');
8             cout << "Неверный ввод. Пожалуйста, введите число: ";
9         }
10        else {
```

```

11         cin.ignore(numeric_limits<streamsize>::max(), '\n');
12         return value;
13     }
14 }
15 }

```

2.10 dictionaryMenu

Функция dictionaryMenu реализует меню для структур данных хеш-таблица и красно-черное дерево.

Вход: имя структуры, словарь

Выход: результаты работы алгоритмов

```

1     void dictionaryMenu(Dictionary<T>& dict, const
string& structName) {
2         while (true) {
3             cout << "\Текущаян_структура_данных:_" <<
structName << endl;
4             cout << "-----\n";
5             ";
6             cout << "1. _Очистить_словарь\n";
7             cout << "2. _Добавить_текст_в_словарь\n";
8             cout << "3. _Вывести_словарь_в_консоль\n";
9             cout << "4. _Найти_слово\n";
10            cout << "5. _Удалить_слово\n";
11            cout << "6. _Добавить_текст_из_файла\n";
12            cout << "-----\n";
13            ";
14            cout << "7. _Экспорт_словаря_в_файл\n";
15            cout << "8. _Импорт_словаря_из_файла\n";
16            cout << "-----\n";
17            ";
18            cout << "0. _Выход_в_главное_меню\n\n";
19            cout << "Выберите_действие: _";
20
21            int choice;
22            while (!(cin >> choice) || choice < 0 || choice
> 8) {
23                cin.clear();
24                cin.ignore(numeric_limits<streamsize>::max
(), '\n');

```

```

22         cout << "Неверный ввод. Пожалуйста, введите
число от 0 до 8: ";
23     }
24
25     switch (choice) {
26     case 1: {
27         dict.Clear();
28         cout << "Словарь очищен.\n";
29         break;
30     }
31     case 2: {
32         cout << "\Введите текст для завершения ввода
дважды нажмите Enter):\n";
33         string text, line;
34
35         cin.ignore(numeric_limits<streamsize>::max
(), '\n');
36         while (getline(cin, line) && !line.empty())
37         {
38             text += line + "\n";
39         }
40         vector<string> words = splitTextIntoWords(
text);
41
42         for (const string& word : words) {
43             int count = dict.Search(word.c_str());
44             dict.Insert(word.c_str(), count + 1);
45         }
46
47         cout << "\Текст добавлен в словарь.\n";
48         break;
49     }
50     case 3: {
51         cout << "\Содержимое словаря:\n";
52         dict.Print();
53         break;
54     }
55     case 4: {
56         cout << "\Введите слово для поиска: ";
57         string word;
58         cin >> word;
59

```

```

60     word = toLowerRussian(word);
61     int count = dict.Search(word.c_str());
62
63     if (count > 0) {
64         cout << "\Словон" << word << " ' "
встречается" << count << " раза() .\n";
65     }
66     else {
67         cout << "\Словон" << word << " ' "не
найден.\n";
68     }
69     break;
70 }
71 case 5: {
72     cout << "Введите слово для удаления: ";
73     string word;
74     cin >> word;
75
76     word = toLowerRussian(word);
77     if (dict.Remove(word.c_str())) {
78         cout << "\Словон" << word << " ' "
удалено.\n";
79     }
80     else {
81         cout << "\Словон" << word << " ' "не
найден.\n";
82     }
83     break;
84 }
85 case 6: {
86     cout << "\Введите имя файла по (умолчанию
input.txt): ";
87     string filename;
88     cin.ignore(numeric_limits<streamsize>::max
(), '\n');
89     getline(cin, filename);
90
91     if (filename.empty()) {
92         filename = "input.txt";
93     }
94     else if (filename.size() < 4 || filename.
substr(filename.size() - 4) != ".txt") {
95         filename += ".txt";

```

```

96         }
97
98         ifstream file(filename);
99         if (!file) {
100             cout << "\Ошибка\nоткрытия_файла_" <<
filename << " '\n";
101             break;
102         }
103
104         string fileText((istreambuf_iterator<char>(
file)), istreambuf_iterator<char>());
105         file.close();
106
107         vector<string> words = splitTextIntoWords(
fileText);
108         for (const string& word : words) {
109             int count = dict.Search(word.c_str());
110             dict.Insert(word.c_str(), count + 1);
111         }
112
113         cout << "Текст_из_файла_" << filename << " '_
добавлен_в_словарь.\n";
114         break;
115     }
116     case 7: {
117         cout << "\Введите_имя_файла_для_экспорта: ";
118         string filename;
119         cin.ignore(numeric_limits<streamsize>::max
(), '\n');
120         getline(cin, filename);
121
122         if (dict.ExportToFile(filename.c_str())) {
123             cout << "Словарь_успешно_экспортирован_в_
файл_" << filename << " '\n";
124         }
125         else {
126             cout << "Ошибка_при_экспорте_словаря\n";
127         }
128         break;
129     }
130     case 8: {
131         cout << "\Введите_имя_файла_для_импорта: ";
132         string filename;

```



```

133         cin.ignore(numeric_limits<streamsize>::max
134         (), '\n');
135         getline(cin, filename);
136         if (dict.ImportFromFile(filename.c_str()))
137     {
138             cout << "Словарь успешно импортирован из
139     файла '" << filename << "'\n";
140         }
141         else {
142             cout << "Ошибка при импорте словаря\n";
143         }
144         break;
145     }
146     case 0: {
147         return;
148     }
149     default: {
150         cout << "Неверный выбор. Попробуйте снова.\n";
151     }
152 }

```

2.11 dataEncodingMenu

Функция реализует меню для кодирования данных.

Вход: запрос на кодирование данных

Выход: результаты кодирования

```

1  void dataEncodingMenu() {
2  while (true) {
3      std::cout << "\Меню кодирования данных:\n";
4      std::cout << "1. Случайно сгенерировать файл в 10
тысяч символов\n";
5      std::cout << "2. Закодировать файл с помощью
алгоритма LZW\n";
6      std::cout << "3. Двуступенчатое кодирование LZW\n";
7      std::cout << "0. Возврат в главное меню\n\n";
8      std::cout << "Выберите действие: ";
9
10     int choice = safeInputInt();
11

```

```

12         switch (choice) {
13         case 1: {
14             std::cout << "\Введите имя файла для
сохранения по (умолчанию random_text.txt): ";
15             std::string filename;
16             std::getline(std::cin, filename);
17
18             if (filename.empty()) {
19                 filename = "random_text.txt";
20             }
21
22             generateRandomFile(filename);
23             break;
24         }
25         case 2: {
26             std::cout << "\Введите имя файла для
кодирования по (умолчанию random_text.txt): ";
27             std::string inputFilename;
28             std::getline(std::cin, inputFilename);
29
30             if (inputFilename.empty()) {
31                 inputFilename = "random_text.txt";
32             }
33
34             std::ifstream inputFile(inputFilename);
35             if (!inputFile) {
36                 std::cerr << "Ошибка открытия файла: " <<
inputFilename << std::endl;
37                 break;
38             }
39
40             std::string originalText((std::
istreambuf_iterator<char>(inputFile)),
41                                     std::istreambuf_iterator<char>());
42             inputFile.close();
43
44             std::vector<int> compressed = LZW::compress
(originalText);
45
46             std::cout << "\Закодированные данные:\n";
47             for (size_t i = 0; i < compressed.size(); i
48             ++){
49                 std::cout << compressed[i] << " ";

```

```

49         }
50
51         std::string compressedFilename =
inputFilename + ".lzw";
52         LZW::writeCompressedToFile(compressed,
compressedFilename);
53
54         std::vector<int> readCompressed = LZW::
readCompressedFromFile(compressedFilename);
55         std::string decompressedText = LZW::
decompress(readCompressed);
56
57         std::cout << "\n\Декодированные данные:\n";
58         std::cout << decompressedText << endl;
59
60         if (compareStrings(originalText,
decompressedText)) {
61             std::cout << "\nПроверка: оригинальный и
декодированный тексты совпадают!\n";
62         }
63         else {
64             std::cerr << "\nОшибка: оригинальный и
декодированный тексты не совпадают!\n";
65             break;
66         }
67
68         size_t originalSize = originalText.size();
69         size_t compressedSizeBits = 0;
70
71         int max_code = *std::max_element(compressed
.begin(), compressed.end());
72         int bits_per_code = std::ceil(std::log2(
max_code + 1));
73         compressedSizeBits = compressed.size() *
bits_per_code * 0.8;
74
75         std::ifstream file(compressedFilename, std
::ios::binary | std::ios::ate);
76         size_t fileSizeBytes = file.tellg();
77         file.close();
78
79         std::cout << "\nИспользовано бит:" <<
compressedSizeBits << "\n";

```

```

80         std::cout << "Размер_до_сжатия:_" <<
originalSize << "_Байт\n";
81         std::cout << "Размер_после_сжатия:_" <<
compressedSizeBits / 8 << "_Байт\n";
82
83
84         double compressionRatio = static_cast<
double>(originalSize * 8) / compressedSizeBits;
85         std::cout << "\Коэффициент_сжатия:_" << std
::fixed << std::setprecision(4)
86         << compressionRatio << "\n";
87
88         break;
89     }
90     case 3: {
91         std::cout << "\Введите_имя_файла_для_
кодирования_по_(умолчанию_random_text.txt):_";
92         std::string inputFilename;
93         std::getline(std::cin, inputFilename);
94
95         if (inputFilename.empty()) {
96             inputFilename = "random_text.txt";
97         }
98
99         std::ifstream inputFile(inputFilename);
100         if (!inputFile) {
101             std::cerr << "Ошибка_открытия_файла:_" <<
inputFilename << std::endl;
102             break;
103         }
104
105         std::string originalText((std::
istreambuf_iterator<char>(inputFile)),
106             std::istreambuf_iterator<char>());
107         inputFile.close();
108
109         std::vector<int> compressed = LZW::
doubleCompress(originalText);
110
111         std::cout << "\Закодированные_данные:\n";
112         for (size_t i = 0; i < compressed.size(); i
113         ++){
            std::cout << compressed[i] << "_";

```

```

114         }
115
116         std::string compressedFilename =
inputFilename + ".lzw2";
117         LZW::writeCompressedToFile(compressed,
compressedFilename);
118
119         std::vector<int> readCompressed = LZW::
readCompressedFromFile(compressedFilename);
120         std::string decompressedText = LZW::
doubleDecompress(readCompressed);
121
122         std::cout << "\n\Декодированные данные:\n";
123         std::cout << decompressedText;
124
125
126         if (compareStrings(originalText,
decompressedText)) {
127             std::cout << "\Проверка: оригинальный и
декодированный тексты совпадают!\n";
128         }
129         else {
130             std::cerr << "\Ошибка: оригинальный и
декодированный тексты не совпадают!\n";
131             break;
132         }
133
134         size_t originalSize = originalText.size();
135         size_t compressedSizeBits = 0;
136
137         int max_code = *std::max_element(compressed
.begin(), compressed.end());
138         int bits_per_code = std::ceil(std::log2(
max_code + 1));
139         compressedSizeBits = compressed.size() *
bits_per_code;
140
141         std::ifstream file(compressedFilename, std
::ios::binary | std::ios::ate);
142         size_t fileSizeBytes = file.tellg();
143         file.close();
144
145         std::cout << "\Размер до сжатия: " <<

```

```

146     originalSize << "Байт\n";
        std::cout << "Размер после сжатия: " <<
fileSizeBytes << "Байт\n";
147         std::cout << "Использовано бит: " <<
compressedSizeBits << "\n";

148
149         double compressionRatio = static_cast<
double>(originalSize * 8) / compressedSizeBits;
150         std::cout << "Коэффициент сжатия: " << std::
fixed << std::setprecision(4)
151             << compressionRatio << "\n";
152
153         break;
154     }
155     case 0: {
156         return;
157     }
158     default: {
159         std::cout << "Неверный выбор. Попробуйте
снова.\n";
160     }
161 }
162 }
163 }

```

2.12 main

Функция main реализует главное меню программы.

Вход: инициализация параметров программы

Выход: результат обработки пользовательского ввода

```

1  int main() {
2      SetConsoleCP(1251);
3      SetConsoleOutputCP(1251);
4      setlocale(LC_ALL, "Russian");
5      srand(time(NULL));
6
7      while (true) {
8          cout << "\Главное меню:\n";
9          cout << "1. Создать словарь\n";
10         cout << "2. Кодирование данных\n";
11         cout << "0. Выход\n";
12         cout << "Выберите действие: ";

```

```

13
14     int choice = safeInputInt();
15
16     switch (choice) {
17     case 1: {
18         cout << "\Выберите n_структуру_данных:\n";
19         cout << "1. _Хештаблица-\n";
20         cout << "2. _Красночерное- _дерево\n";
21         cout << "Выберите: _";
22
23         int structChoice = safeInputInt();
24
25         if (structChoice == 1) {
26             Dictionary<HashTable> dict(1000);
27             dictionaryMenu(dict, "Хештаблица-");
28         }
29         else if (structChoice == 2) {
30             Dictionary<RBT> dict;
31             dictionaryMenu(dict, "Красночерное- _
дерево");
32         }
33         else {
34             cout << "Неверный_выбор.\n";
35         }
36         break;
37     }
38     case 2: {
39         dataEncodingMenu();
40         break;
41     }
42     case 0: {
43         return 0;
44     }
45     default: {
46         cout << "Неверный_выбор. _Попробуйте_ снова.\n";
47     }
48 }
49 }
50 }

```

3 Результаты работы

При запуске программы пользователь видит главное меню и запрос на ввод номера действия – рис. 3.

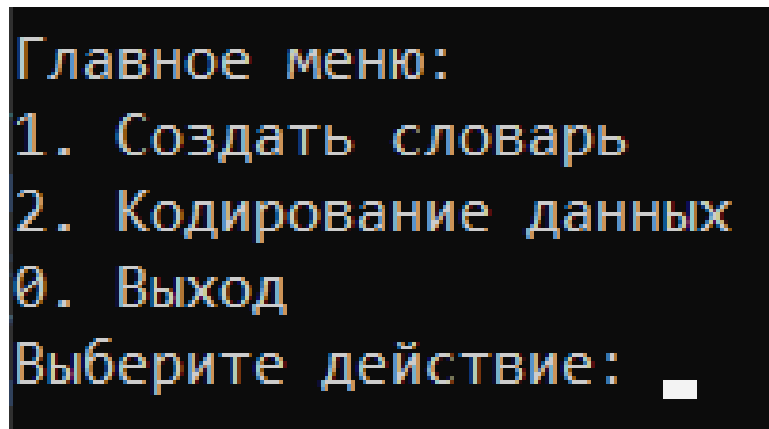


Рис. 3: Главное меню

При выборе 1 в главном меню пользователя просят выбрать структуру данных для словаря – рис. 4.

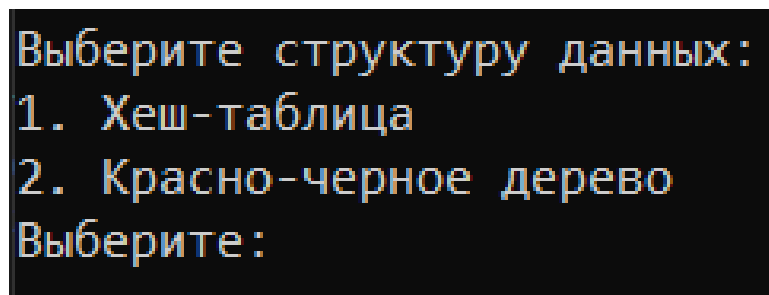


Рис. 4: Выбор структуры данных

При выборе 1 выводится меню для хеш-таблицы – рис. 5.

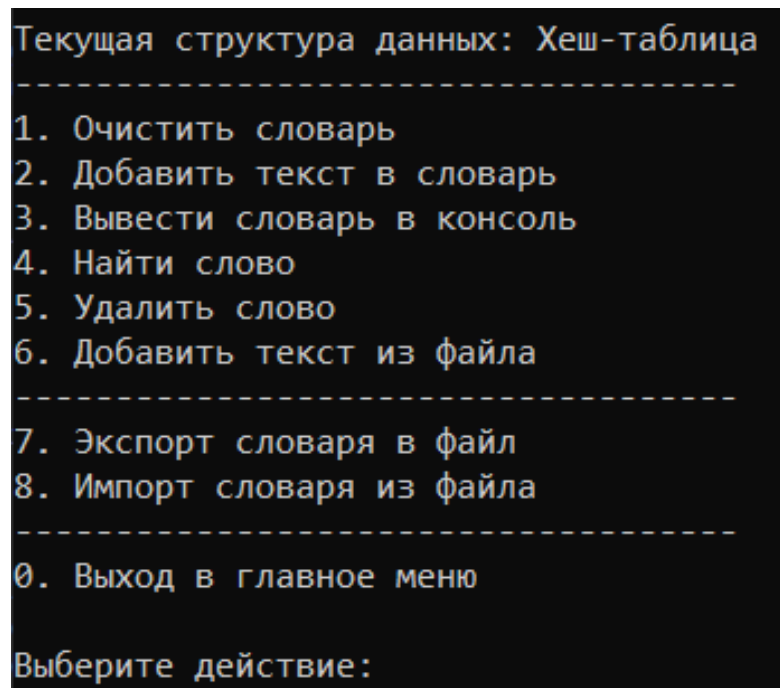


Рис. 5: Меню для хеш-таблицы

Очищение словаря – рис. 6.

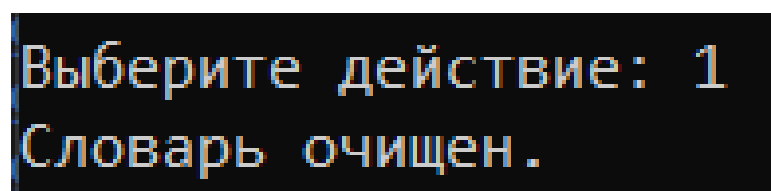


Рис. 6: Очищение словаря для хеш-таблицы

Добавление текста в словарь – рис. 7.

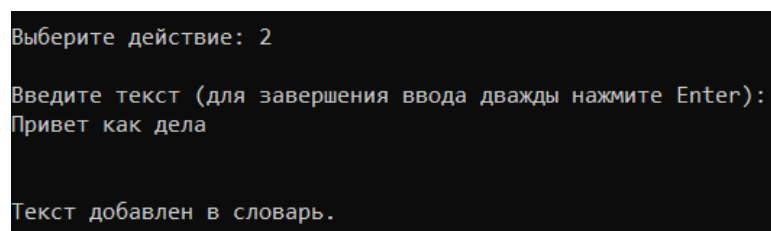
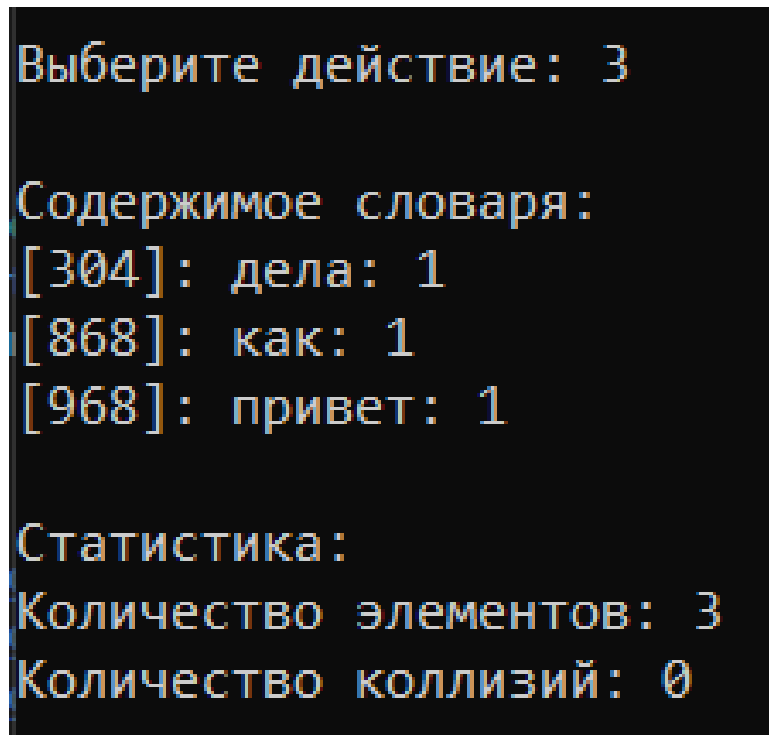


Рис. 7: Добавление текста в словарь для хеш-таблицы

Вывод словаря в консоль – рис. 8.



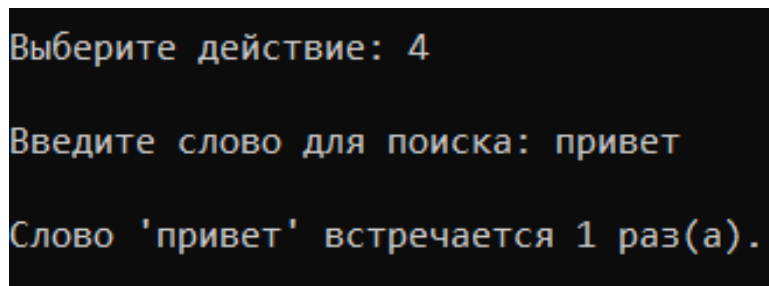
```
Выберите действие: 3

Содержимое словаря:
[304]: дела: 1
[868]: как: 1
[968]: привет: 1

Статистика:
Количество элементов: 3
Количество коллизий: 0
```

Рис. 8: Вывод словаря в консоль для хеш-таблицы

Поиск слова – рис. 9.



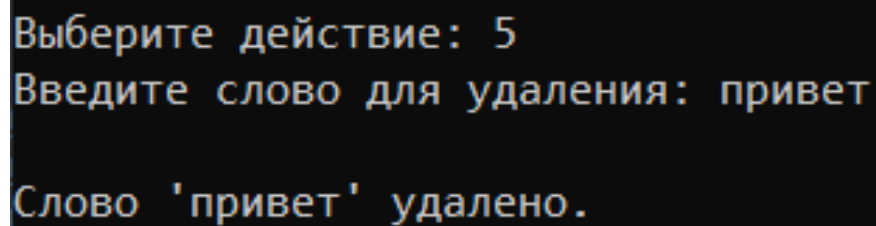
```
Выберите действие: 4

Введите слово для поиска: привет

Слово 'привет' встречается 1 раз(а).
```

Рис. 9: Поиск слова для хеш-таблицы

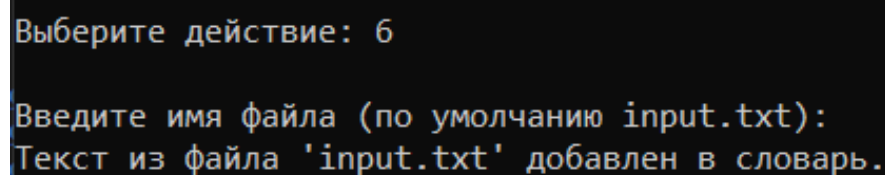
Удаление слова – рис. 10.



```
Выберите действие: 5
Введите слово для удаления: привет
Слово 'привет' удалено.
```

Рис. 10: Удаление слова для хеш-таблицы

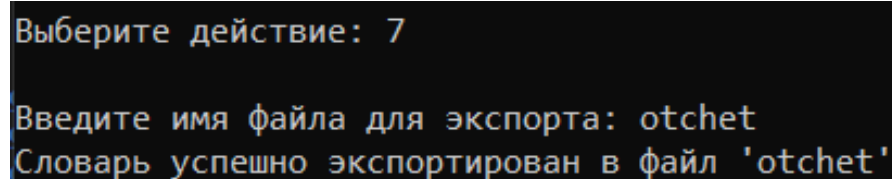
Добавление текста из файла – рис. 11.



```
Выберите действие: 6
Введите имя файла (по умолчанию input.txt):
Текст из файла 'input.txt' добавлен в словарь.
```

Рис. 11: Добавление текста из файла для хеш-таблицы

Экспорт словаря в файл – рис. 12.



```
Выберите действие: 7
Введите имя файла для экспорта: otchet
Словарь успешно экспортирован в файл 'otchet'
```

Рис. 12: Экспорт словаря в файл для хеш-таблицы

Импорт словаря из файла – рис. 13.

```
Выберите действие: 8  
  
Введите имя файла для импорта: file  
Словарь успешно импортирован из файла 'file'
```

Рис. 13: Импорт словаря из файла для хеш-таблицы

При выборе 2 выводится меню для красно-чёрного дерева – рис. 14.

```
Текущая структура данных: Красно-черное дерево  
-----  
1. Очистить словарь  
2. Добавить текст в словарь  
3. Вывести словарь в консоль  
4. Найти слово  
5. Удалить слово  
6. Добавить текст из файла  
-----  
7. Экспорт словаря в файл  
8. Импорт словаря из файла  
-----  
0. Выход в главное меню  
  
Выберите действие: █
```

Рис. 14: Меню для красно-чёрного дерева

Очищение словаря – рис. 15.

```
Выберите действие: 1  
Словарь очищен.
```

Рис. 15: Очищение словаря для красно-чёрного дерева

Добавление текста в словарь – рис. 16.

```
Выберите действие: 2
Введите текст (для завершения ввода дважды нажмите Enter):
Привет как дела хорошо нормально отлично супер класс словарь красно черное дерево
Текст добавлен в словарь.
```

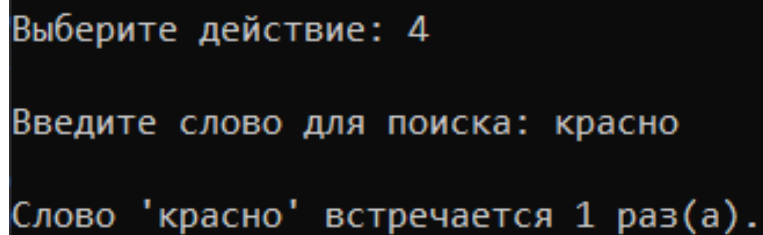
Рис. 16: Добавление текста в словарь для красно-чёрного дерева

Вывод словаря в консоль – рис. 17.

```
Выберите действие: 3
Содержимое словаря:
                /-- черное: 1
                /-- хорошо: 1
                /-- супер: 1
                \-- словарь: 1
        /-- привет: 1
        \-- отлично: 1
-- нормально: 1
                /-- красно: 1
                /-- класс: 1
        \-- как: 1
                /-- дерево: 1
                \-- дела: 1
```

Рис. 17: Вывод словаря в консоль для красно-чёрного дерева

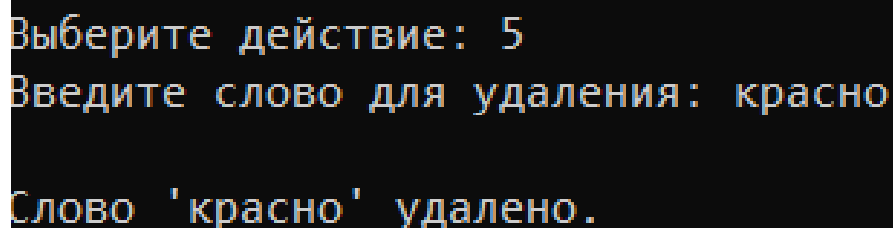
Поиск слова – рис. 18.



```
Выберите действие: 4
Введите слово для поиска: красно
Слово 'красно' встречается 1 раз(а).
```

Рис. 18: Поиск слова для красно-чёрного дерева

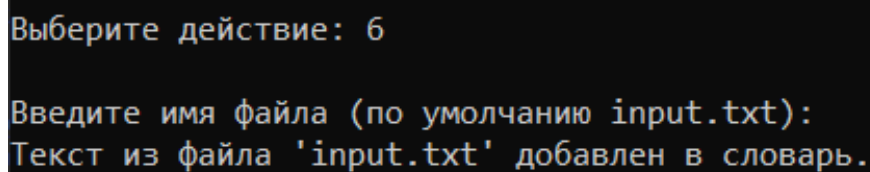
Удаление слова – рис. 19.



```
Выберите действие: 5
Введите слово для удаления: красно
Слово 'красно' удалено.
```

Рис. 19: Удаление слова для красно-чёрного дерева

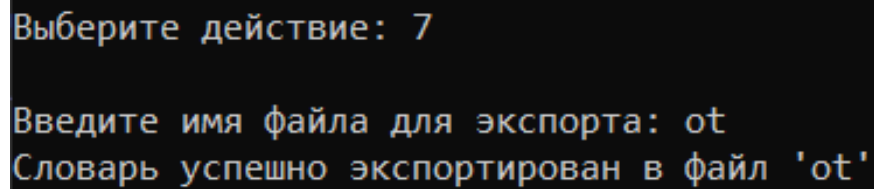
Добавление текста из файла – рис. 20.



```
Выберите действие: 6
Введите имя файла (по умолчанию input.txt):
Текст из файла 'input.txt' добавлен в словарь.
```

Рис. 20: Добавление текста из файла для красно-чёрного дерева

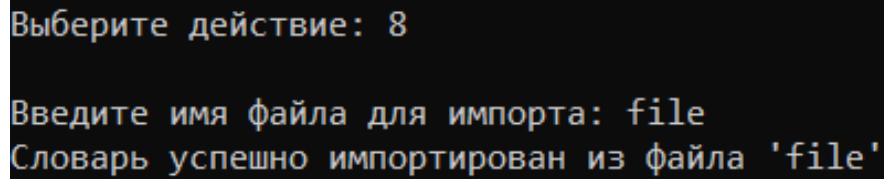
Экспорт словаря в файл – рис. 21.



```
Выберите действие: 7  
Введите имя файла для экспорта: ot  
Словарь успешно экспортирован в файл 'ot'
```

Рис. 21: Экспорт словаря в файл для красно-чёрного дерева

Импорт словаря из файла – рис. 22.



```
Выберите действие: 8  
Введите имя файла для импорта: file  
Словарь успешно импортирован из файла 'file'
```

Рис. 22: Импорт словаря из файла для красно-чёрного дерева

Фрагмент заполненной и большой хеш-таблицы – рис. 23.

```
[600]: загорались: 1
[622]: желания: 1
[654]: место: 1
[682]: городке: 1
[683]: цветок: 1
[689]: жил: 1
[756]: наполнял: 1
[771]: за: 1
[783]: менять: 1
[785]: радость: 1
[803]: способна: 1
[811]: воздух: 1
[819]: знал: 1
[829]: их: 1
[844]: каждый: 3
[849]: люди: 1
[881]: мечты: 1
[890]: цветы: 1
[899]: него: 1
[905]: тихом: 1
[908]: мир: 1
[922]: волшебства: 1
[942]: растений: 1
[961]: дружбы: 1
[962]: имени: 1
[963]: доброта: 1
[968]: счастьем: 1

Статистика:
Количество элементов: 69
Количество коллизий: 1
Коллизии произошли в ячейках: 31
```

Рис. 23: Фрагмент хеш-таблицы

Фрагмент заполненного и большого красно-чёрного дерева – рис. 24.


```

        /-- радость: 1
            /-- приходили: 1
            \-- превращался: 1
            \-- пожелание: 1
    /-- по: 1
        /-- о: 1
        \-- него: 1
            /-- наполнял: 1
            \-- надеждой: 1
\-- могли: 1
    /-- мир: 1
    \-- мечты: 1
    /-- место: 1
    \-- менять: 1
    /-- люди: 1
    \-- людей: 1
    \-- любви: 1
    \-- легко: 1
    \-- красиво: 1
\-- когда: 1
    /-- каждый: 3
    \-- их: 1
    \-- исполнять: 1
    \-- искры: 1
-- имени: 1
    /-- издалека: 1
    /-- и: 6
        /-- знал: 1
        \-- загорались: 1
        \-- за: 1
    /-- жил: 1
    /-- желания: 1
    \-- его: 2
        /-- дружбы: 1
        /-- доброта: 1
        \-- день: 1
        \-- дарил: 1
\-- городке: 1
    /-- глазах: 1
    /-- где: 2
    /-- встречал: 1
    \-- всегда: 1
    \-- волшебства: 1
\-- волшебный: 1
    /-- воздух: 1
    /-- в: 3
    \-- был: 1
        /-- ароматом: 1
        \-- анатолий: 2

```

Рис. 24: Фрагмент красно-чёрного дерева

При выборе 2 в главном меню открывается меню кодирования – рис. 25.

```

Меню кодирования данных:
1. Случайно сгенерировать файл в 10 тысяч символов
2. Закодировать файл с помощью алгоритма LZW
3. Двуступенчатое кодирование LZW
0. Возврат в главное меню

```

Рис. 25: Меню кодирования

Генерация случайного файла в 10 тысяч символов – рис. 26.

```

Выберите действие: 1

Введите имя файла для сохранения (по умолчанию random_text.txt):

Файл 'random_text.txt' успешно создан с 10000 символами.

```

Рис. 26: Генерация случайного файла

Фрагмент закодированного, декодированного файла и результатов кодирования – рис. 27, 28.

```

Выберите действие: 2

Введите имя файла для кодирования (по умолчанию random_text.txt):

Закодированные данные:
68 103 79 29 80 32 23 75 83 81 70 10 30 68 79 103 21 4 13 1 66 77 74 68 83 11 14 29
32 6 8 77 29 14 189 5 76 6 70 20 25 21 69 26 0 7 68 5 5 27 4 69 83 72 75 71 74 9 72
66 141 0 13 22 18 32 75 72 0 16 247 10 81 77 80 11 79 27 71 103 31 23 4 27 240 23 78
18 8 25 70 30 18 30 79 68 17 12 68 10 18 1 16 14 9 83 25 2 12 13 265 11 9 76 82 27 1
2 67 16 5 17 27 72 5 11 29 72 303 22 14 212 26 82 83 1 7 73 26 125 3 1 29 12 26 72 8
32 81 80 236 31 22 402 22 23 2 2 24 358 70 83 12 79 75 228 31 15 78 28 0 408 11 497
83 68 30 21 29 16 158 5 348 2 0 394 374 74 167 19 5 2 70 76 304 419 79 32 15 25 13 1
2 1 602 271 73 68 72 12 370 17 9 17 118 605 13 66 82 68 69 30 22 12 81 381 30 30 5 7
226 67 68 6 30 199 22 13 4 435 350 20 69 25 205 81 7 528 78 78 81 71 79 17 4 14 4 2
479 417 14 361 75 17 80 18 2 151 9 11 75 25 81 25 69 359 11 187 292 522 31 29 5 23
95 26 76 0 25 4 70 67 74 560 185 3 70 4 30 27 103 14 103 10 304 69 13 11 22 355 18 5
7 30 73 1 3 68 66 278 273 656 20 83 21 74 10 26 68 7 66 505 6 119 32 103 19 19 20 2
6 236 67 165 9 618 25 7 19 69 368 6 69 21 27 2 15 282 79 78 852 18 26 77 5 363 20 67

```

Рис. 27: Кодирование

```

1атйё0#нярт%хсе81т?ъ!2ьуэвн*иы&*вф0дёёсч5лмёг64%ну1д!8щ2щп2%!д
5ьзу*щ8н540срзк#ъ#9$?нтщ64эц*?ж#6^л*сашу?цпрёез$бк6!мим^дшиих7
шзбзпж*б0эттндваеьб0уёбнё9$с$гц5нр@юз0ь?ы&чэюхсёнэ*з*кшь1ьав0п
е*фувьёрт?153коэъ%ёя!к55щэп!к12цифэи*щц7г2&ж1ню#!кёоцём5зу4? ?ж
605?ад2йхчрбм1ым26р#мз68флж0мхцсюлцякмпг1амз$ц#$щяйврйвд!8щц@ш
лфис*22кгдмб13!жф0@&2мк13йй$сщцб#иъс9еиъэивцо!жм!жфьбонйъчйа^р
б0ывэз3в3о0цооэ2&юпер9фмЗыя@#ыбю0чрб$16т$2ютл24фёд9ьфа8ржшюэкщ@

Проверка: оригинальный и декодированный тексты совпадают!

Использовано бит: 64282
Размер до сжатия: 10000 Байт
Размер после сжатия: 8035 Байт

Коэффициент сжатия: 1.2445

```

Рис. 28: Декодирование и результаты

Двуступенчатое кодирование – рис. 29.

```

605?ад2йхчрбм1ым26р#мз68флж0мхцсюлцякмпг1амз$ц#$щяйврйвд!8щц
лфис*22кгдмб13!жф0@&2мк13йй$сщцб#иъс9еиъэивцо!жм!жфьбонйъчйа
б0ывэз3в3о0цооэ2&юпер9фмЗыя@#ыбю0чрб$16т$2ютл24фёд9ьфа8ржшюэк
Проверка: оригинальный и декодированный тексты совпадают!

Размер до сжатия: 10000 Байт
Размер после сжатия: 11399 Байт
Использовано бит: 91156
Коэффициент сжатия: 0.8776

```

Рис. 29: Двуступенчатое кодирование

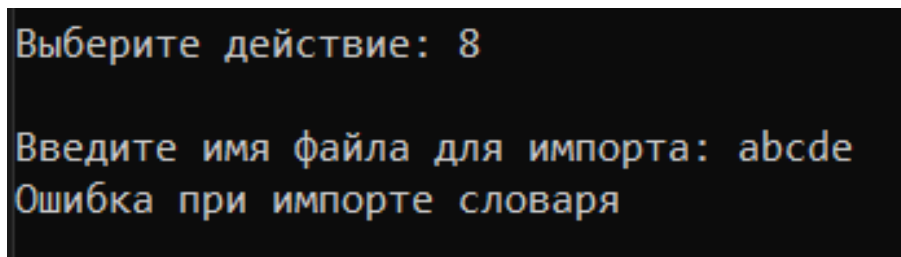
Примеры некорректного ввода – рис. 30, 31.

```

Выберите действие: двдвдв\
Неверный ввод. Пожалуйста, введите число: двжыдв
Неверный ввод. Пожалуйста, введите число: _

```

Рис. 30: Некорректный ввод числа



Выберите действие: 8

Введите имя файла для импорта: abcde

Ошибка при импорте словаря

Рис. 31: Выбор несуществующего файла

Заключение

В результате работы реализованы словари на основе структур данных хеш-таблица и красно-чёрное дерево, а также функции их очистки, добавления, поиска, удаления и загрузки слов из текстового файла.

Реализованы генерация файла длиной 10 тысяч символов на основе распределения Эрланга и функции кодирования с использованием алгоритма LZW.

Структуры данных были реализованы без использования готовых структур данных.

Достоинства:

- Записав словарь на основе хеш-таблицы в файл, его можно будет считать и для красно-чёрного дерева и наоборот;
- Поиск и вставка в хеш-таблице выполняются за константное время $O(1)$.

Недостатки:

- Отсутствие графического интерфейса.

Масштабирование: в программу можно добавить другие алгоритмы кодирования, например Фано или Хаффмена, также можно реализовать графический интерфейс.

Список литературы

- [1] Секция "Телематика" / текст : электронный / — URL: <https://tema.spbstu.ru/dismath/> (Дата обращения 28.05.2025).
- [2] Вадзинский Р. Н. Справочник по вероятностным распределениям - Санкт-Петербург: Наука, 2001 - 295 с.
- [3] Алгоритм LZW // ИТМО URL: https://neerc.ifmo.ru/wiki/index.php?title=Алгоритм_LZW (Дата обращения 28.05.2025)