САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. ПЕТРА ВЕЛИКОГО
Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

Отчет по лабораторной работе:

Реализация словаря на основе красно-черного дерева и хэш таблицы.

Дисциплина: «Теория графов».

Работу выполнил:
Рябинин А. Д.
Студент группы: 3530201/10001

Проверил:
Востров А.В.

Санкт-Петербург - 2023 г.

Содержание

Введние

В лабораторной работе требуется реализовать приложение с функционалом словаря. Словарь реализовать на основе красно-черного дерева и хэш таблицы.

Необходимо реализовать функции добавления, удаления и поиска ключа, функцию полной очистки словаря и загрузки/дополнения словаря из текстового файла.

Глава 1
Математическое описание

1.1. Алгебраическая структура.

Красно-черное дерево (Red-Black Tree) является сбалансированным двоичным поисковым деревом, где каждый узел содержит ключ и цвет.

Пусть *RBT* будет представлять красно-черное дерево.

Для повышения эффективности операций, используют различные приемы перестройки деревьев, так чтобы высота дерева была величиной (logn). Такие приемы называются балансировкой деревьев. При этом используются разные критерии качества балансировки. Одним из видов сбалансированных деревьев поиска являются так называемые красночерные деревья, для которых предусмотрены операции балансировки, гарантирующие оценку высоты величиной (logn)

1.1.1. Комбинаторные свойства красно-черных деревьев:

Для произвольного узла x определим черную высоту bh(x) как количество черных узлов на пути из x в некоторый лист, не считая сам узел x. По свойству 4 эта сумма не зависит от выбранного листа. Черной высотой дерева будем считать черную высоту его корня. Пусть size[x] – количество внутренних узлов в поддереве с корнем x (nil-узлы не считаются).

1.1.2. Вращения

это манипуляции с красно-черными деревьями с целью восстановления RB-свойств в случае их нарушения. Их используют при реализации операций Insert и Delete. Вращение представляет собой локальную операцию, при которой меняется несколько указателей, но свойство упорядоченности сохраняется.

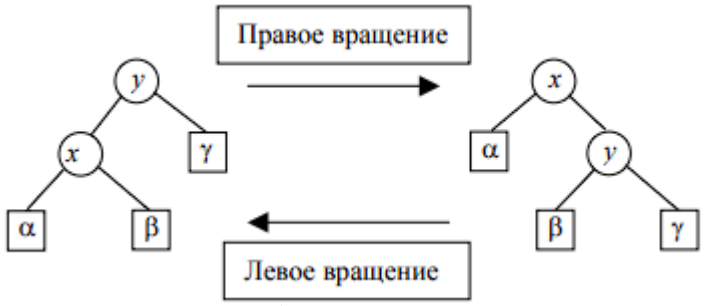

Рис. 1.1: взаимно обратные вращения: левое и правое.

1.1.3. Левое вращения

возможно в любом узле x, правый ребенок которой (назовем его y) не является листом (nil). После вращения y оказывается корнем поддерева, x – левым ребенком узла y, а бывший левый ребенок y – правым ребенком узла x.

1.1.4. Свойства красно-черного дерева:

1. Каждый узел имеет цвет, который может быть красным (R) или черным (B).
2. Корень дерева всегда черный.

3. Каждый лист (NIL) является черным.
4. Если узел красный, то оба его потомка должны быть черными.
5. Все пути от данного узла до его листьев должны содержать одинаковое количество черных узлов.

1.1.5. Структура узла:

1. Каждый узел содержит ключ (значение) и цвет (R или B).
2. Каждый узел имеет ссылки на его левого и правого потомков, а также ссылку на его родителя.

1.1.6. Операции на красно-черном дереве:

1. Вставка: Добавление нового узла в красно-черное дерево, после чего выполняется балансировка дерева, чтобы сохранить его свойства.
2. Удаление: Удаление узла из красно-черного дерева, после чего выполняется балансировка дерева для сохранения его свойств.
3. Поиск: Поиск заданного ключа в красно-черном дереве.

1.2. Хэш таблица

Хэш таблица— это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, данные) и выполнять три операции: добавление новой пары, поиск и удаление пары по ключу. Число хранимых элементов, делённое на размер массива (число возможных значений хэш-функции), называется коэффициентом заполнения хэш-таблицы.
Выполнение любой операции (поиск, добавление и удаление) в хэш-таблице начинается с вычисления хэш-функции от ключа. Получающееся значение является индексом в массиве. Ситуация, когда для различных ключей получается одно и то же значение хэш-функции, называется коллизией.
Ниже рассмотрено несколько вариантов реализаций, которые устраняют данную проблему

1.2.1. Прямая адресация:

В массиве хранятся указатели на связанные списки пар. Также имеются реализации, где в ячейке таблицы хранится сама пара и указатель на следующий элемент. Коллизии просто приводят к тому, что появляются списки длиной более одного элемента. Пример такой таблицы приведён ниже. Данная реализация может быть усовершенствованная путём использования вместо связанных списков других динамических структур данных.

1.2.2. Открытая адресация:

В массиве хранятся сами пары (ключ, данные). В случае возникновения коллизии, алгоритм поиска (удаления, добавления) объекта просто перемещается на ячейку вправо до момента разрешения коллизии. Разрешение коллизии происходит при достижении пустой ячейки или ячейки, в которой хранится пара с заданным ключом. Размер шага смещения вправо может зависеть от значения ключа и вычисляться с помощью второй хэшфункции. Данная техника называется двойным хэшированием с открытой адресацией.

Важное свойство хэш-таблицы состоит в том, что все три операции в среднем выполняются за время O(1). Но при этом не гарантируется, что время выполнения отдельной операции

мало. Это связано с тем, что при достижении некоторого значения коэффициента заполнения необходимо осуществлять перестройку индекса хэш-таблицы: увеличить значение размера массива и заново добавить в пустую хэш-таблицу все пары. Для данной структуры требуется память, равная размеру таблицы, следовательно, пространственная сложность алгоритма составляет O(1).

Хэш-таблицы являются очень удобными для реализации ассоциативных массивов, но у них имеется несколько недостатков:

1. Хэш-таблица занимает много памяти даже при нахождении в ней малого количества элементов..
2. Время любого поиска одинаково (всегда нужно рассчитывать хэш-функцию),то есть при поиске ключа, которого нет в таблице, будет расходоваться столько же времени, сколько и на поиск элемента, который там есть, в отличие от поиска в бинарном дереве когда ещё на ранних стадиях можно определить, что искомого элемента нет.

Глава 2
Особенности реализации.

Лабораторная реализована на языке javascript.

2.1. Реализация структуры красно-черного дерева.

Листинг 2.1: main.js // Реализация класса Node

```
    // Узел красночерного— дерева
    class Node {
     // кончтруктор принимает значения ключа, значения узлацвет, узла
     //  возвращает ключи с данными параметрами
      constructor(key, value, color) {
        this.key = key; // Ключ узла
        this.value = value; // Значение узла
        this.color = color; // Цвет узла может( быть красным"" или черным"")
        this.left = null; // Левый дочерний узел
        this.right = null; // Правый дочерний узел
        this.parent = null; // Родительский узел
      }
    }
```

Листинг 2.2: Order.h // Реализация класса RedBlackTree

```
    class RedBlackTree {

        /*
        Конструктор инициализирует корень дерева,
        а также переменные для хранения всех данных и структуры дерева.
        */

        // конструктор не принимает аргументов,
        //  возвращает корень КЧ дерева
        constructor() {
        this.root = null; // Корень дерева
        this.allData = ""; // Все данные дерева в виде строки
        this.struct = ""; // Структура дерева в виде строки
        }

        // Вызываем метод "traverseTree" для обхода дерева, начиная с корневого узла
        // Выводит структуру дерева в консоль

        // Вход:
        // Выход: структура дерева
        printStructure() {
            // Создаем переменную "struct" для хранения
            // структуры дерева в виде строки
            this.struct = "";

            // Вызываем метод "traverseTree" для обхода дерева,
            // начиная с корневого узла
            // Передаем корневой узел, пустую строку
            // и значение true для отступа перед узлом
            this.traverseTree(this.root, "", true);

            // Выводим структуру дерева в консоль
            console.log(this.struct);

            // выводим структуру дерева на экран
            document.getElementById("story").value = this.struct;
            }
```

```
39
40
41
42              // Рекурсивно проходит по дереву и собирает информацию о каждом узле
43
44              // Вход:    узел, строка буфер, флаг последеного элемента
45              // Выход: структура дерева в строке буфере
46          traverseTree(node, indent, isLast) {
47
48              // Проверяем, является ли узел пустым (null)
49              if (node === null) {
50                  return;
51              }
52
53              // Определяем маркер для текущего узла в зависимости от того,
54              //  является ли он последним в своем уровне
55              const marker = isLast ? "" : "";
56
57              // Определяем цвет для текущего узла красный( или черный)
58
59              const color = node.color === "red" ? "[R]" : "[B]";
60
61              // Добавляем информацию о текущем узле в структуру дерева
62              this.struct += ${indent}${marker} ${color}${node.key + ": " +
    node.value}\n;
63
64              // Определяем отступ для дочерних узлов
65              const childIndent = indent + (isLast ? " " : " ");
66
67              // Рекурсивно вызываем метод "traverseTree" для левого поддерева
68              this.traverseTree(node.left, childIndent, false);
69
70              // Рекурсивно вызываем метод "traverseTree" для правого поддерева
71              this.traverseTree(node.right, childIndent, true);
72          }
73
74          // Вспомогательный метод для вставки узла
75          insertNode(node) {
76              // Инициализируем переменную "current" значением корневого узла
77              let current = this.root;
78
79               // Инициализируем переменную "parent" значением null
80              let parent = null;
81
82              // Итеративно ищем место для вставки узла в дерево
83              while (current !== null) {
84
85                  // Запоминаем текущий узел в переменной "parent"
86                  parent = current;
87                  if (node.key < current.key) {
88                      // Если значение ключа вставляемого узла
89                      // меньше значения ключа текущего узла,
90                      // переходим к левому потомку
91                      current = current.left;
92                  } else {
93                      // Иначе переходим к правому потомку
94                      current = current.right;
95                  }
96              }
97
98              // Присваиваем вставляемому узлу найденного родителя
99              node.parent = parent;
100
101             if (parent === null) {
102
103                 // Если дерево пустое, делаем вставляемый узел корневым узлом
104                 this.root = node;
105
106             } else if (node.key < parent.key) {
107
108                 // Если значение ключа вставляемого узла меньше
109                 // значения ключа родительского узла,
110                 // делаем вставляемый узел левым потомком
111                 parent.left = node;
112
113             } else if (node.key > parent.key) {
114
115                 // Если значение ключа вставляемого узла
116                 // больше значения ключа родительского узла,
117                 // делаем вставляемый узел правым потомком
118                 parent.right = node;
119
120             } else {
```

```
121                 // Если значения ключей равны,
122                 // обновляем значение родительского узла
123                 parent.value = node.value;
124             }
125
126             // Устанавливаем цвет вставленного узла в красный
127             node.color = "red";
128
129             // Вызываем метод для восстановления свойств красночерного— дерева
130             this.insertFixup(node);
131         }
132
133         // Вспомогательный метод
134         // для исправления свойств красночерного— дерева после вставки
135
136             // Вход:    корень КЧ дерева
137             // Выход: исправленное КЧ дерево
138
139         insertFixup(node) {
140             // Пока у узла есть родитель и цвет родителя равен красный""
141             while (node.parent !== null && node.parent.color === "red") {
142                 // Если родитель узла является левым потомком своего родителя
143                 if (node.parent === node.parent.parent.left) {
144                     // Находим дядю узла
145                     let uncle = node.parent.parent.right;
146                     // Если дядя существует и его цвет равен красный""
147                   if (uncle !== null && uncle.color === "red") {
148                     // Изменяем цвет родителя узла и дяди на черный"",
149                     // а цвет родителя родителя на красный""
150                     node.parent.color = "black";
151                     uncle.color = "black";
152                     node.parent.parent.color = "red";
153                     node = node.parent.parent; // Переходим к родителю
    родителя узла
154                   } else {
155                     // Если узел является правым потомком своего родителя
156                     if (node === node.parent.right) {
157                       node = node.parent; // Переходим к родителю узла
158                       this.leftRotate(node); // Выполняем левое вращение
159                     }
160
161                     // Изменяем цвет родителя узла на черный"",
162                     // а цвет родителя родителя на красный""
163                     node.parent.color = "black";
164                     node.parent.parent.color = "red";
165                     // Выполняем правое вращение относительно родителя родителя
166                     this.rightRotate(node.parent.parent);
167                   }
168                 } else {
169                   // Если родитель узла является правым потомком своего родителя
170                   let uncle = node.parent.parent.left;
171
172                   // Если дядя существует и его цвет равен " красный"
173                   if (uncle !== null && uncle.color === "red") {
174                     // Изменяем цвет родителя узла и дяди на " черный",
175                     // а цвет родителя родителя на " красный"
176                     node.parent.color = "black";
177                     uncle.color = "black";
178                     node.parent.parent.color = "red";
179
180                      // Переходим к родителю родителя узла
181                     node = node.parent.parent;
182                   } else {
183                     // Если узел является левым потомком своего родителя
184                     if (node === node.parent.left) {
185                       node = node.parent; // Переходим к родителю узла
186
187                       // правое вращение относительно родителя
188                       this.rightRotate(node);
189                     }
190
191                     // Изменяем цвет родителя узла на черный"",
192                     // а цвет родителя родителя на красный""
193                     node.parent.color = "black";
```

```
                node.parent.parent.color = "red";

                // левое вращение относительно родителя родителя
                this.leftRotate(node.parent.parent);
                }
              }
            }
        this.root.color = "black";
    }

    // Вспомогательный метод для левого поворота

    // Вход:   корень КЧ дерева
    // Выход:  КЧ дерево после левого поворота

    leftRotate(node) {
        // Сохраняем правого потомка узла в переменную "rightChild"
        let rightChild = node.right;

        // Переназначаем правого потомка узла
        // на левого потомка правого потомка
        node.right = rightChild.left;

        // Проверяем, существует ли левый потомок правого потомка
        // Если существуетто, устанавливаем его родителя на текущий узел "node"
        if (rightChild.left !== null) {
            rightChild.left.parent = node;
        }

        // Переназначаем родителя правого потомка
        // на родителя текущего узла "node"
        rightChild.parent = node.parent;

        // Проверяем, является ли текущий узел "node" корневым узлом
        if (node.parent === null) {
            // Если является, то переназначаем корневой узел на правого потомка
            this.root = rightChild;
        } else if (node === node.parent.left) {
            // Если текущий узел "node" — левым потомок своего родителя,
            // переназначаем левого потомка родителя на правого потомка
            node.parent.left = rightChild;
        } else {
            // В противном случае, текущий узел "node"
            // является правым потомком своего родителя,
            // переназначаем правого потомка родителя на правого потомка
            node.parent.right = rightChild;
        }

        // Устанавливаем левого потомка правого потомка на узел "node"
        rightChild.left = node;

        // Устанавливаем родителя текущего узла "node" на правого потомка
        node.parent = rightChild;
    }

    // Вход:   корень КЧ дерева
    // Выход:  КЧ дерево после правого поворота
     rightRotate(node) {// аналогично leftRotate }


     // Метод для вставки ключа и значения в словарь

     // Вход:   ключ и значение
    // Выход:  дерево, в которое включен новый узел
    insert(key, value) {
        // Создаем новый узел с переданным ключом,
        // значением и цветом "red"
        const newNode = new Node(key, value, "red");

        // Вызываем метод "insertNode" для вставки нового узла в дерево
        this.insertNode(newNode);
    }

    // Метод для получения значения по ключу
    // Вход:   ключ
    // Выход:  значение, найденное по ключу в КЧ дереве
```

```
274         get(key) {
275             // Создаем переменную "current"
276             // и присваиваем ей значение корневого узла
277             let current = this.root;
278 
279             // Пока "current" не равен null
280             while (current !== null) {
281                 // Проверяем, равен ли ключ "key" ключу текущего узла "current"
282                 if (key === current.key) {
283                     // Если равны, возвращаем значение текущего узла
284                     return current.value;
285                 }
286                 // Если ключ "key" меньше ключа текущего узла "current"
287                 else if (key < current.key) {
288                     // Переходим к левому потомку текущего узла
289                     current = current.left;
290                 }
291                 // Если ключ "key" больше ключа текущего узла "current"
292                 else {
293                     // Переходим к правому потомку текущего узла
294                     current = current.right;
295                 }
296             }
297 
298         // Если ключ не найден, возвращаем null
299         return null;
300         }
301 
302 
303 
304         // Вспомогательный метод для удаления узла
305 
306         // Вход:    узел
307         // Выход:   дерево без данного узла
308 
309         removeNode(node) {
310             let y = node;
311             let yOriginalColor = y.color;
312             let x; // Создаем переменную "x" для хранения узла
313 
314             if (node.left === null) {
315                 // Проверяем, есть ли у узла левый дочерний узел
316 
317                  // Присваиваем переменной "x" правый дочерний узел
318                 x = node.right;
319 
320                 // Вызываем метод "transplant" для замены узла
321                 // на его правого дочернего узла
322                 this.transplant(node, node.right);
323             } else if (node.right === null) {
324                 // Проверяем, есть ли у узла правый дочерний узел
325 
326                 // Присваиваем переменной "x" левый дочерний узел
327                 x = node.left;
328 
329                 // Вызываем метод "transplant" для замены узла
330                 // на его левого дочернего узла
331                 this.transplant(node, node.left);
332             } else { // Если у узла есть и левый, и правый дочерний узлы
333 
334                 // Находим минимальный узел в правом поддереве и
335                 // присваиваем его переменной "y"
336                 y = this.minimum(node.right);
337 
338                 // Сохраняем цвет минимального узла в
339                 // переменную "yOriginalColor"
340                 yOriginalColor = y.color;
341 
342                 // Присваиваем переменной "x" правого
343                 // дочернего узла минимального узла "y"
344                 x = y.right;
345 
346                 if (y.parent === node) {
347                 // Проверяем, является ли родитель
348                 // минимального узла "y" узлом "node"
349 
350 
351 
352                 // Присваиваем родителю "x"
353                 // минимального узла "y" значение "y"
```

```
354                 x.parent = y;
355
356             } else {
357                 // замены минимального узла "y"
358                 // на его правого дочернего узла
359                 this.transplant(y, y.right);
360
361                 y.right = node.right;
362
363                 // Присваиваем родителю правого дочернего узла
364                 // минимального узла "y" значение "y"
365                 y.right.parent = y;
366             }
367
368             // Вызываем метод "transplant" для замены узла
369             // на минимальный узел "y"
370             this.transplant(node, y);
371
372             Присваиваем// левому дочернему узлу минимального узла "y"
373             // значение левого дочернего узла узла "node"
374             y.left = node.left;
375
376             // Присваиваем родителю левого дочернего узла
377             // минимального узла "y" значение "y"
378             y.left.parent = y;
379
380             // Присваиваем цвету минимального узла "y"
381             // значение цвета узла "node"
382             y.color = node.color;
383         }
384
385         if (yOriginalColor === "black") {
386         // Проверяем, был ли цвет минимального узла "y" черным
387         // Вызываем метод "deleteFixup" для исправления
388         // нарушений свойств красночерного— дерева
389         this.deleteFixup(x);
390         }
391     }
392
393
394
395
396     // Вход:    2 узла
397     // Выход:  дерево, в котором узлы поменялись местами
398
399     // Метод для перемещения узла v вместо узла u в дереве
400     transplant(u, v) {
401         // Проверяем, является ли узел u корневым узлом не( имеет родителя)
402         if (u.parent === null) {
403             // Если узел u — корневой узел,
404             // то заменяем корневой узел на узел v
405             this.root = v;
406         }
407         // Проверяем, является ли узел u левым потомком своего родителя
408         else if (u === u.parent.left) {
409             // Если узел u — левый потомок своего родителя,
410             // то заменяем левого потомка родителя на узел v
411             u.parent.left = v;
412         }
413         // Узел u является правым потомком своего родителя
414         else {
415             // Заменяем правого потомка родителя на узел v
416             u.parent.right = v;
417         }
418
419         // Проверяем, является ли узел v нулевым не( имеет родителя)
420         if (v !== null) {
421             // Если узел v не является нулевым,
422             то// устанавливаем его родителем родителя узла u
423             v.parent = u.parent;
424         }
425     }
426
427
428     // Метод для удаления узла по ключу
429     // Вход:    ключ
430     // Выход:  КЧ дерево без узла с заданным ключом
431     remove(key) {
```

```
432             // Проверяем, если ключ пустой, то возвращаемся из метода
433             if (key == "") {
434                 return;
435             }
436
437             // Ищем узел по ключу в дереве
438             const node = this.search(key);
439
440             // Если найденный узел не равен null,
441             // то вызываем метод "removeNode" для удаления узла
442             if (node !== null) {
443                 this.removeNode(node);
444             }
445         }
446
447
448         // Метод для исправления нарушений свойств
449         // красночерного– дерева при удалении узла
450         // Вход:    Корень КЧ дерева
451         // Выход:  КЧ дерево с исправленными свойствами
452         deleteFixup(node) {
453             // Пока текущий узел не является корневым
454             // и его цвет равен черный"" или узел равен null
455             while (node !== this.root && (node === null || node.color === "
    black")) {
456                 // Если текущий узел является левым потомком своего родителя
457                 if (node === node.parent.left) {
458                 // Получаем сиблинга правого( потомка родителя)
459                 let sibling = node.parent.right;
460                 // Если сиблинг красного цвета
461                   if (sibling.color === "red") {
462
463                     // Меняем цвет сиблинга на черный""
464                     sibling.color = "black";
465
466                      // Меняем цвет родителя на красный""
467                     node.parent.color = "red";
468
469                     // Производим левый поворот вокруг родителя
470                     this.leftRotate(node.parent);
471
472                     // Обновляем сиблинга
473                     sibling = node.parent.right;
474                   }
475
476                   // Если оба потомка сиблинга
477                   // левый( и правый) являются черными"" или null
478                   if (
479                     (sibling.left === null || sibling.left.color === "black
    ") &&
480                     (sibling.right === null || sibling.right.color === "
    black")
481                   ) {
482                   // Меняем цвет сиблинга на красный""
483                     sibling.color = "red";
484
485                     // Переходим к родителю
486                     node = node.parent;
487                   } else {
488                     // Если правый потомок сиблинга черный"" или null
489                     if (sibling.right === null || sibling.right.color === "
    black") {
490                       // Меняем цвет левого потомка сиблинга на черный""
491                       sibling.left.color = "black";
492
493                       // Меняем цвет сиблинга на красный""
494                       sibling.color = "red";
495
496                       // Производим правый поворот вокруг сиблинга
497                       this.rightRotate(sibling);
498
499                       // Обновляем сиблинга
500                       sibling = node.parent.right;
501                     }
502
503                     // Присваиваем сиблингу цвет родителя
```

```
504                     sibling.color = node.parent.color;
505
506                     // Меняем цвет родителя на черный""
507                     node.parent.color = "black";
508
509                     // Меняем цвет правого потомка сиблинга на черный""
510                     sibling.right.color = "black";
511
512                     // Производим левый поворот вокруг родителя
513                     this.leftRotate(node.parent);
514
515                      // Переходим к корневому узлу
516                     node = this.root;
517                   }
518                 } else {
519                   // Если текущий узел является правым потомком своего родителя
520                    // Получаем сиблинга левого( потомка родителя)
521                   let sibling = node.parent.left;
522
523                   // Если сиблинг красного цвета
524                   if (sibling.color === "red") {
525                   // Меняем цвет сиблинга на черный""
526                     sibling.color = "black";
527
528                     // Меняем цвет родителя на красный""
529                     node.parent.color = "red";
530
531                      // Производим правый поворот вокруг родителя
532                     this.rightRotate(node.parent);
533
534                      // Обновляем сиблинга
535                     sibling = node.parent.left;
536                   }
537
538                   // Если оба потомка сиблинга правый( и левый)
539                   // являются черными"" или null
540                   if (
541                     (sibling.right === null || sibling.right.color === "
    black") &&
542                     (sibling.left === null || sibling.left.color === "black
    ")
543                   ) {
544                   // Меняем цвет сиблинга на красный""
545                     sibling.color = "red";
546
547                      // Переходим к родителю
548                     node = node.parent;
549                   } else {
550                     // Если левый потомок сиблинга черный"" или null
551                     if (sibling.left === null || sibling.left.color === "
    black") {
552                     // Меняем цвет правого потомка сиблинга на черный""
553                       sibling.right.color = "black";
554
555                       // Меняем цвет сиблинга на красный""
556                       sibling.color = "red";
557
558                       // Производим левый поворот вокруг сиблинга
559                       this.leftRotate(sibling);
560
561                       // Обновляем сиблинга
562                       sibling = node.parent.left;
563                     }
564
565                      // Присваиваем сиблингу цвет родителя
566                     sibling.color = node.parent.color;
567
568                     // Меняем цвет родителя на черный""
569                     node.parent.color = "black";
570
571                      // Меняем цвет левого потомка сиблинга на черный""
572                     sibling.left.color = "black";
573
574                      // Производим правый поворот вокруг родителя
575                     this.rightRotate(node.parent);
576
577                      // Переходим к корневому узлу
578                     node = this.root;
579                   }
```

```
580                 }
581             }
582
583             // Если узел не равен null, устанавливаем его цвет в черный""
584             if (node !== null) {
585                 node.color = "black";
586             }
587         }
588
589
590         printValues() {
591             this.inorderTraversal(this.root);
592         }
593
594         // Вход:    корень
595         // Выход:  список всех значений в дереве
596         // Метод для выполнения обхода дерева в порядке inorder
597         // левый( поддерево -> корень -> правый поддерево)
598         inorderTraversal(node) {
599             // Проверяем, что узел не является пустым
600             if (node !== null) {
601                 // Рекурсивно вызываем метод inorderTraversal для левого поддерева
602                 this.inorderTraversal(node.left);
603                 // Выводим значение узла в формате
604                 // ключ": значение" в элемент с идентификатором "story"
605                 document.getElementById("story").value += node.key + ": " +
    node.value + "\n";
606
607                 // Рекурсивно вызываем метод
608                 // inorderTraversal для правого поддерева
609                 this.inorderTraversal(node.right);
610             }
611         }
612
613         // Объявление функции "getValues",
614         // которая будет возвращать все значения дерева в виде строки
615         // Вход:
616         // Выход:  список всех значений в дереве в виде строки
617         getValues() {
618             // Внутри функции объявляем вспомогательную функцию "removeNewLine"
619             // Функция проверяет, заканчивается ли строка на \n
620             // и удаляет его, если да
621             const removeNewLine = (str) => {
622                 if (str.endsWith("\n")) {
623                     return str.slice(0, -1);
624                 }
625                 return str;
626             };
627
628             // Инициализируем переменную "allData"
629             // для хранения всех данных дерева в виде строки
630             this.allData = "";
631
632             // Вызываем метод "inorderTraversalforGet" для обхода дерева
633             // В результате обхода, значения узлов добавляются в переменную "allData"
634             this.inorderTraversalforGet(this.root);
635
636             // Удаляем символ перевода строки, если он есть в конце строки "allData"
637             this.allData = removeNewLine(this.allData);
638
639             // Возвращаем значение "allData"
640             return this.allData;
641         }
642
643
644         // Метод для обхода дерева в порядке возрастания значений (in-order
    traversal)
645         // Вход:    корень
646         // Выход:  список всех значений в дереве  в порядке возрастания значений
647         inorderTraversalforGet(node) {
648             // Проверяем, что узел не является пустым (null)
649             if (node !== null) {
650                 // Добавляем значение ключа и значения узла к переменной "allData"
651                 this.allData += node.key + ": " + node.value + "\n";
652
```

```
653                 // Рекурсивно вызываем метод  для левого поддерева
654                 this.inorderTraversalforGet(node.left);
655
656                 // Рекурсивно вызываем метод  для правого поддерева
657                 this.inorderTraversalforGet(node.right);
658             }
659         }
660
661         // Метод "search" для поиска узла в дереве по заданному ключу
662         // Вход:    ключ
663         // Выход:  значение, найленное по заданному ключу
664
665         search(key) {
666             // Вызываем метод "searchNode" для поиска узла, начиная с корневого
    узла
667             // Передаем корневой узел и заданный ключ в качестве аргументов
668             return this.searchNode(this.root, key);
669         }
670
671
672         // Вход:    корень  дерева
673         // Выход:  значение, найленное по заданному ключу
674         // Метод для поиска узла в дереве по заданному ключу
675         searchNode(node, key) {
676             // Проверяем, является ли текущий узел пустым или
677             // ключ совпадает с заданным ключом
678             if (node === null || key === node.key) {
679                 return node; // Возвращаем текущий узел
680             }
681
682             // Если заданный ключ меньше ключа текущего узла,
683             // вызываем рекурсивно метод "searchNode" для левого поддерева
684             if (key < node.key) {
685                 return this.searchNode(node.left, key);
686             }
687
688             // Если заданный ключ больше ключа текущего узла,
689             // вызываем рекурсивно метод "searchNode" для правого поддерева
690             return this.searchNode(node.right, key);
691         }
692
693         // Метод "removeValues" вызывается для удаления значений из дерева.
694         removeValues() {
695             // Вызываем метод "removeTraversal" и
696             // передаем ему корневой узел в качестве аргумента.
697             this.removeTraversal(this.root);
698         }
699
700
701         // Метод для удаления узлов дерева с помощью обхода в глубину
702         // Вход:    Узел
703         // Выход:  КЧ дерево без данного узала
704         removeTraversal(node) {
705             // Проверяем, что узел не является пустым (null)
706             if (node !== null) {
707                 // Рекурсивно вызываем метод removeTraversal для левого поддерева
708                 this.removeTraversal(node.left);
709
710
711                 // Удаляем ключ узла из словаря
712
713                 dictionary.remove(node.key);
714
715                 // Рекурсивно вызываем метод removeTraversal для правого поддерева
716                 this.removeTraversal(node.right);
717             }
718         }
719
720         // Метод для поиска узла по ключу в дереве
721         search(key) {
722             // Вызываем метод "searchNode" и
723             // передаем ему корневой узел и ключ для поиска
724             return this.searchNode(this.root, key);
725         }
726
727
728     }
729
730
```

2.2. Реализация структуры хэш таблицы

Все значения храянтся в таблице в виде узлов

Листинг 2.3: Order.h // Реализация класса HSNode

```
    // Определение класса HSNode узел( хэштаблицы—)
    class HSNode {
        // Конструктор класса, принимающий ключ и значение
        constructor(key, value) {
        // Присваиваем ключ текущему узлу
        this.key = key;

        // Присваиваем значение текущему узлу
        this.value = value;

        // Устанавливаем ссылку на следующий узел в цепочке, по умолчанию — null
        this.next = null;
        }
    }


```

Листинг 2.4: Order.h // Реализация класса HashTable

```
    // Объявление класса HashTable
    class HashTable {
        // Конструктор класса, принимает необязательный параметр size
        constructor(size = 10) {
        // Инициализация свойства "have" счетчиком количества элементов в хэштаблице—
        this.have = 0;

        // Инициализация свойства "size" размером хэштаблицы—

        this.size = size;

        // Инициализация свойства "storage" массивом заданного размера для хранения
    элементов
        this.storage = new Array(size);
        }

        // Определяем метод "hash" для хэширования ключа
        // Вход:   ключ
        // Выход:  хэш значение
        hash(key) {
            // Инициализируем переменную "hash" со значением 0, которая будет
    хранить хэш ключа
            let hash = 0;

            // Проходим по каждому символу в ключе
            for (let i = 0; i < key.length; i++) {
                // Добавляем код символа к текущему значению хэша
                hash += key.charCodeAt(i);
            }

            // Возвращаем остаток от деления хэша на размер хэштаблицы—,
            // чтобы получить индекс, где ключ будет храниться
            return hash % this.size;
        }


        / Метод для добавления элемента в хэштаблицу—
        // Вход:   ключ и значение
        // Выход:  хэш таблица с добавленным в нее элементом с данным значением
        add(key, value) {
            // Увеличиваем счетчик элементов в хэштаблице—
            this.have += 1;

            // Проверяем, достигли ли мы максимальной загрузки хэштаблицы—
            if (this.have == this.size) {
                // Если достигли, вызываем метод "resize"
                // для увеличения размера хэштаблицы— в два раза
                this.resize(this.size * 2);
            }

            // Вычисляем индекс для ключа с помощью хэшфункции—
```

```
50              const index = this.hash(key);
51
52              // Проверяем, есть ли уже элемент в ячейке с данным индексом
53              if (!this.storage[index]) {
54                  // Если ячейка пуста, создаем новый узел
55                  c// ключом и значением и сохраняем его в ячейку
56                  this.storage[index] = new HSNode(key, value);
57              } else {
58                  // Если в ячейке уже есть элементы,
59                  проходим// по списку элементов
60                  let currentNode = this.storage[index];
61
62                  // Проверяем ключ каждого узла в списке
63                  while (currentNode.next) {
64                    // Если ключ совпадает с добавляемым ключом,
65                    обновляем// значение и завершаем метод
66                    if (currentNode.key === key) {
67                      currentNode.value = value;
68                      return;
69                    }
70
71                    // Переходим к следующему узлу в списке
72                    currentNode = currentNode.next;
73                  }
74
75                  // Проверяем последний узел списка на соответствие ключу
76                  if (currentNode.key === key) {
77                    // Если ключ совпадает, обновляем значение
78                    currentNode.value = value;
79                  } else {
80                    // Если ключ не совпадает, с
81                    //оздаемс новый узел и добавляем его в конец списка
82                    currentNode.next = new HSNode(key, value);
83                  }
84              }
85          }
86
87          // Вход:    ключ
88          // Выход:  хэш таблица без данного ключа
89          remove(key) {
90              // Вычисляем индекс хеша для ключа
91              const index = this.hash(key);
92
93              // Проверяем, существует ли элемент с данным индексом в хранилище
94              if (!this.storage[index]) {
95                  // Если элемент не существует, возвращаемся из метода
96                  return;
97              }
98
99              // Инициализируем переменные для текущего и предыдущего узлов
100             let currentNode = this.storage[index];
101             let previousNode = null;
102
103             // Пока существует текущий узел
104             while (currentNode) {
105                 // Проверяем, совпадает ли ключ текущего узла с заданным ключом
106                 if (currentNode.key === key) {
107                     // Если ключи совпадают, то выполняем удаление узла из
    хештаблицы—
108
109                   // Проверяем, является ли текущий узел первым узлом в цепочке
110                   if (previousNode === null) {
111                     // Если текущий узел первый, то переустанавливаем ссылку в
    хранилище на следующий узел
112                     this.storage[index] = currentNode.next;
113                   } else {
114                     // Если текущий узел не первый, то переустанавливаем ссылку в
    предыдущем узле на следующий узел
115                     previousNode.next = currentNode.next;
116                   }
117                   // Возвращаемся из метода, после удаления узла
118                   return;
119                 }
120                 // Присваиваем предыдущему узлу значение текущего узла
```

```
121                 previousNode = currentNode;
122                 // Присваиваем текущему узлу значение следующего узла
123                 currentNode = currentNode.next;
124             }
125         }
126
127     // удажение всех элементов хэш таблицы
128     // происходит созданием нового массива
129     // Вход:
130     // Выход:  новая хэш таблица
131     removeAll() {this.storage = new Array(this.size) }
132
133
134     // Вход:  ключ
135     // Выход:  знчение элемента сданным ключом
136     / Метод "get" для получения значения по ключу из хэштаблицы—
137     get(key) {
138         // Вычисляем индекс хэша для ключа
139         const index = this.hash(key);
140
141         // Получаем ссылку на первый узел в цепочке, соответствующей данному индексу
142         let currentNode = this.storage[index];
143
144         // Пока есть текущий узел
145         while (currentNode) {
146             // Если ключ текущего узла совпадает с заданным ключом
147             if (currentNode.key === key) {
148                 // Возвращаем значение текущего узла
149                 return currentNode.value;
150             }
151             // Переходим к следующему узлу в цепочке
152             currentNode = currentNode.next;
153         }
154
155         // Если не найдено соответствующего ключа, возвращаем undefined
156         return undefined;
157     }
158
159     // Метод для изменения размера хранилища
160     // Вход:  новый размер массива  хэш таблицы
161     // Выход:  новая хэш таблица
162     resize(newSize) {
163         // Сохраняем текущее хранилище в переменную "currentStorage"
164         const currentStorage = this.storage;
165
166         // Обновляем размер хранилища
167         this.size = newSize;
168
169         // Создаем новое хранилище с новым размером
170         this.storage = new Array(newSize);
171
172         // Проходимся по каждому узлу в текущем хранилище
173         for (const node of currentStorage) {
174             // Создаем переменную "currentNode" и присваиваем ей текущий узел
175             let currentNode = node;
176
177             // Пока "currentNode" существует
178             while (currentNode) {
179               // Добавляем ключ и значение текущего узла в новое хранилище
180               this.add(currentNode.key, currentNode.value);
181
182               // Переходим к следующему узлу
183               currentNode = currentNode.next;
184             }
185         }
186     }
187
188
189     // Вход:
190     // Выход:  значения в хэш таблице
191     // Метод для вывода значений в консоль или элемент с идентификатором "story"
192     printValues() {
193         // Цикл для прохода по массиву "storage"
194         for (let index = 0; index < this.storage.length; index++) {
195             // Проверяем, существует ли элемент в текущей позиции "index" массива
196             if (this.storage[index]) {
197
```

```
                // Присваиваем текущий узел из "storage" переменной "currentNode"
                let currentNode = this.storage[index];

              // Цикл выполняется, пока "currentNode" существует
              while (currentNode) {
                // Добавляем значение ключа и значения текущего узла к элементу с идентификатором "story"
                document.getElementById("story").value += currentNode.key + ": " + currentNode.value + "\n";

                // Переходим к следующему узлу в связанном списке
                currentNode = currentNode.next;
              }
            }
        }
    }


    // Вход:
    // Выход:  все данные из хэш таблицы в виде строки
    // Определяем функцию removeNewLine, которая удаляет символ новой строки ("\n") в конце строки
    getValues() {
        const removeNewLine = (str) => {
            // Проверяем, заканчивается ли строка символом новой строки
            if (str.endsWith("\n")) {
                // Если да, то возвращаем строку
                // без последнего символа символа( новой строки)
                return str.slice(0, -1);
            }
            // Если строка не заканчивается символом новой строки,
            // возвращаем исходную строку
            return str;
        };

        // Создаем переменную "allData" для хранения всех данных в виде строки
        let allData = "";

        // Проходим по элементам в массиве "storage"
        for (let index = 0; index < this.storage.length; index++) {
            // Проверяем, есть ли элемент в текущем индексе массива
            if (this.storage[index]) {
                // Присваиваем переменной "currentNode" текущий элемент
                let currentNode = this.storage[index];
                // Пока существует текущий узел
                while (currentNode) {
                    // Добавляем ключ и значение текущего узла к строке "allData"
                    allData += currentNode.key + ": " + currentNode.value + "\n";

                        // Переходим к следующему узлу
                        currentNode = currentNode.next;
                }
            }
        }

        // Удаляем конец строки "allData" с помощью функции removeNewLine
        allData = removeNewLine(allData);

        // Возвращаем все данные в виде строки
        return allData;
    }



    }
```

2.3. Реализация интерфейса

Листинг 2.5: Order.h // Реализация класса RedBlackTree

```
1
3
```

В данной лабораторной работе по разработке словаря на основе хэш-таблицы был реализован пользовательский интерфейс с использованием HTML, CSS, JavaScript и фреймворка Bootstrap. В этом разделе документации представлено описание и объяснение реализации интерфейса.

Html код главной страницы представлен в приложении.

На главной странице доступны условия лабораторной работы.

На первой странице слайдера можно выбрать структору данных, которая будет использоваться для работы с записями. На второй странице слейдера можно вывести структуру дерева.

В центре страницы расположены блоки с функционалом словаря: Добавление, удаление, поиск, загрузка, дополнене, сохранение словаря.

Листинг 2.6: LinearOrder.cpp // Реализация функции суммы

```
```

на Рис 2.1 представлены условия лабораторной работы

2.4. Интерфейс

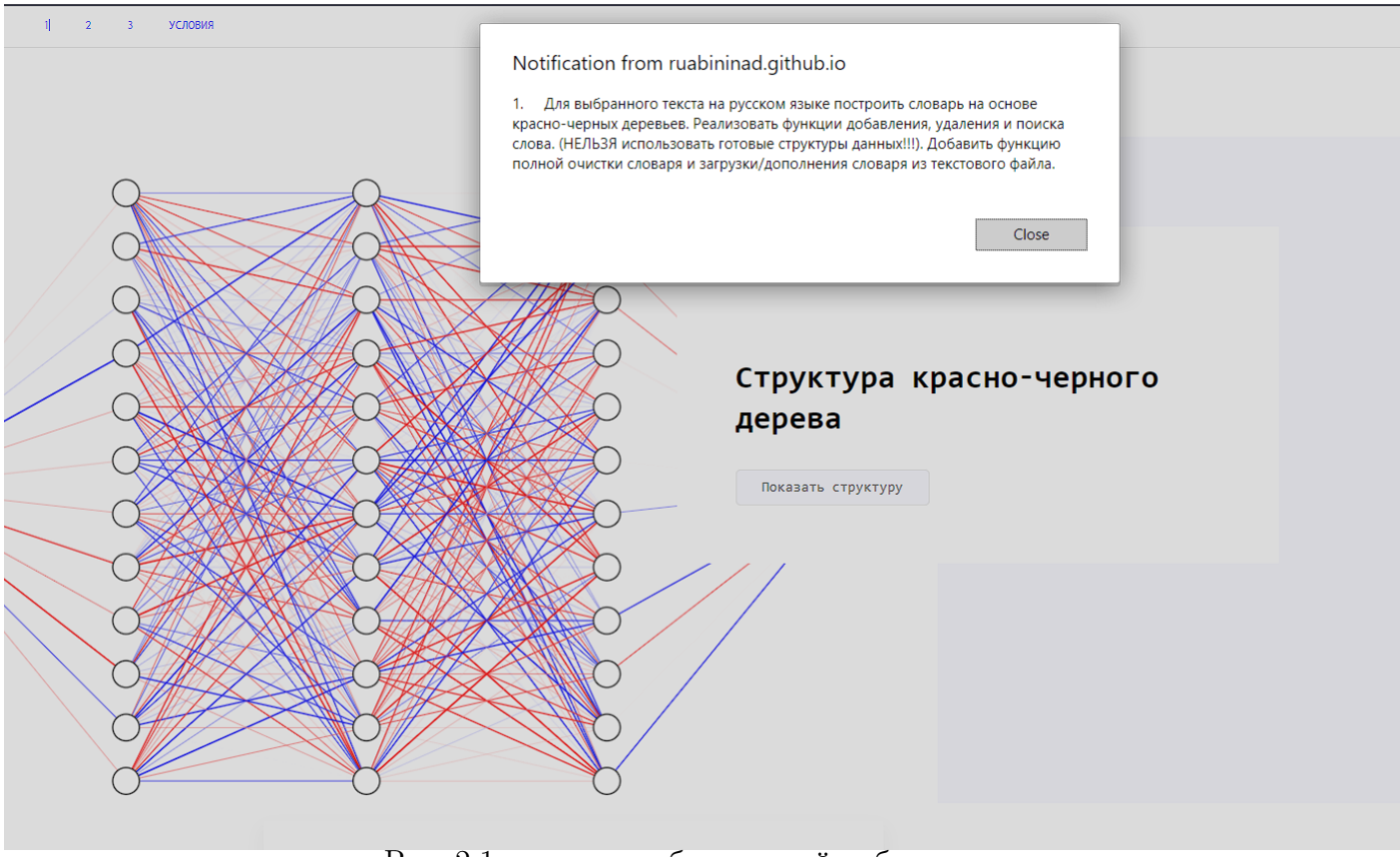

Рис. 2.1: условия лабораторной работы

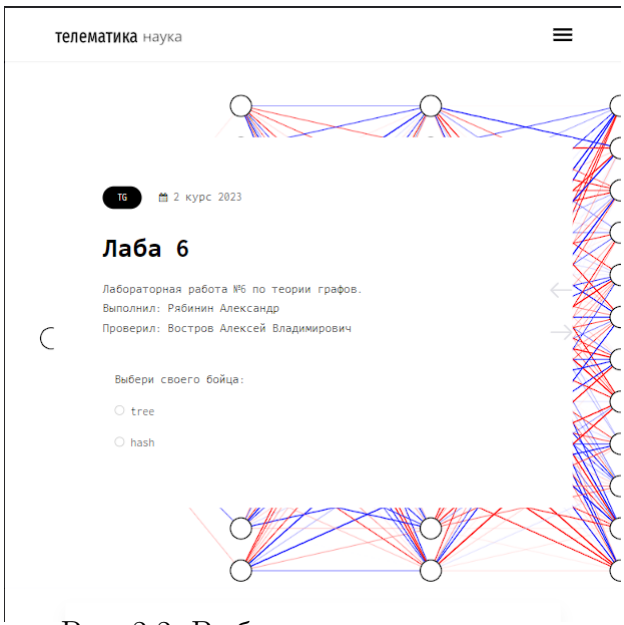

Рис. 2.2: Выбор структуры данных

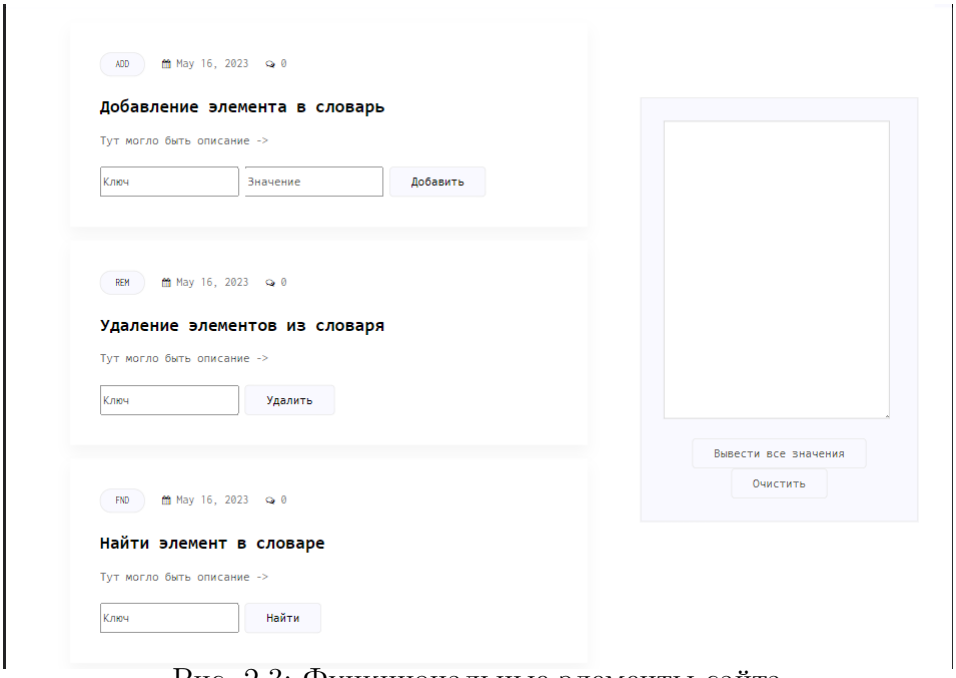

Рис. 2.3: Функциональные элементы сайта

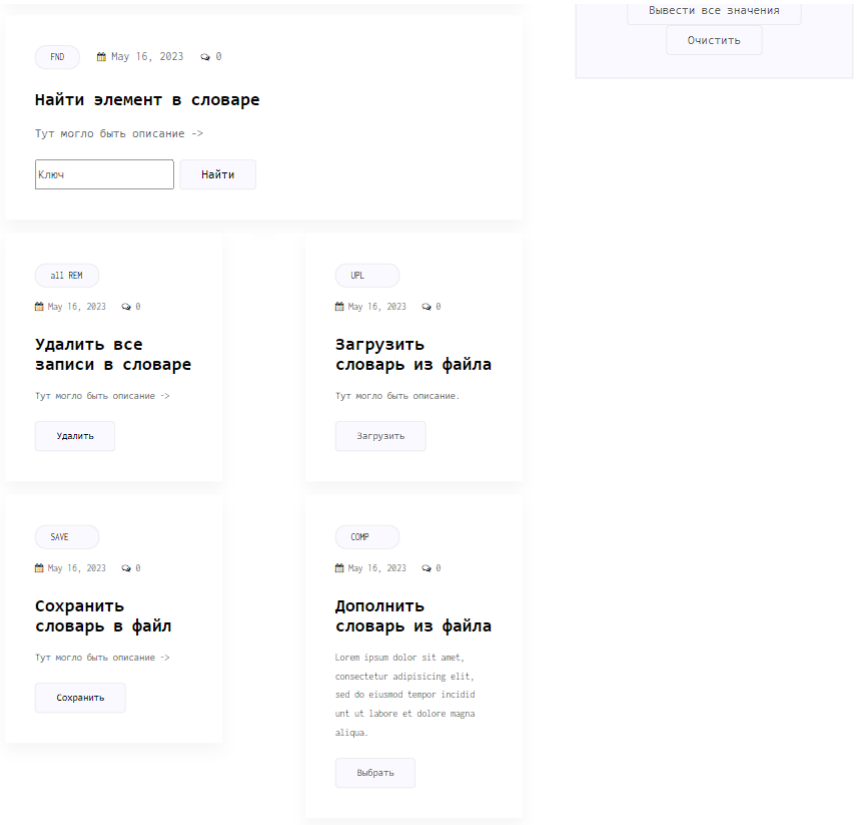

Рис. 2.4: Функциональные элементы сайта

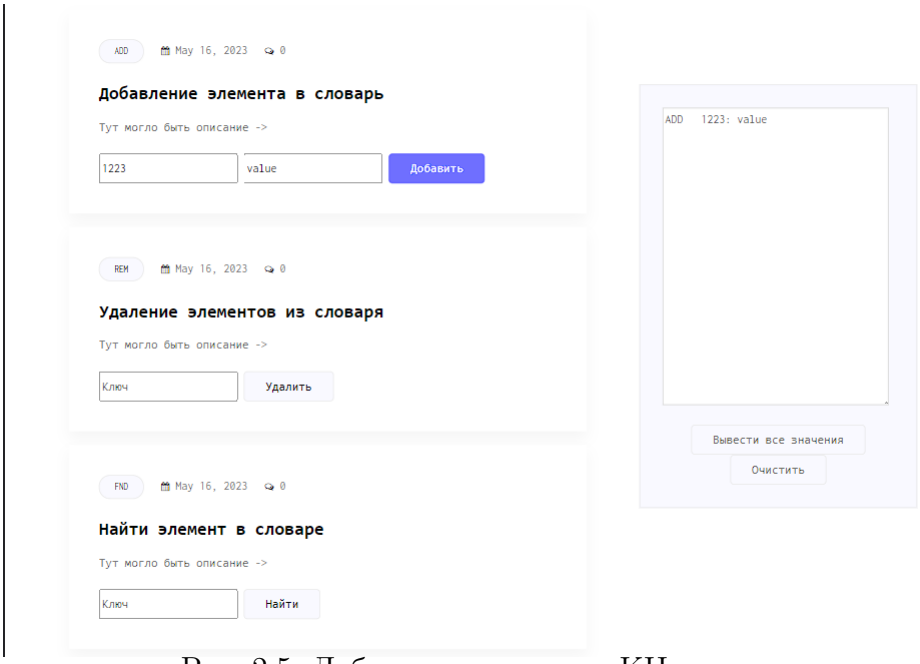

Рис. 2.5: Добавление записи в КЧ дерево

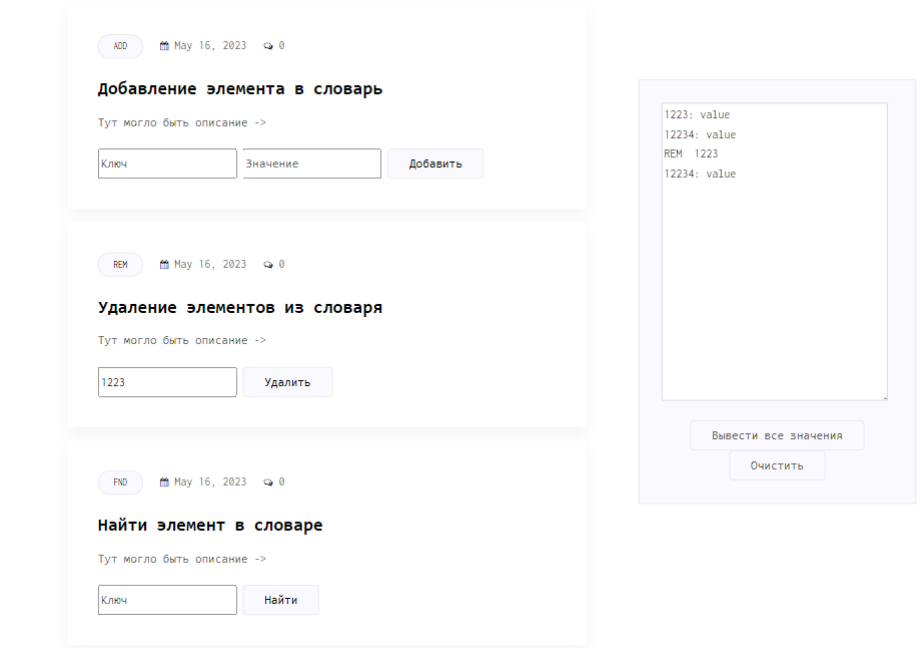

Рис. 2.6: Удаление записи из КЧ дерева

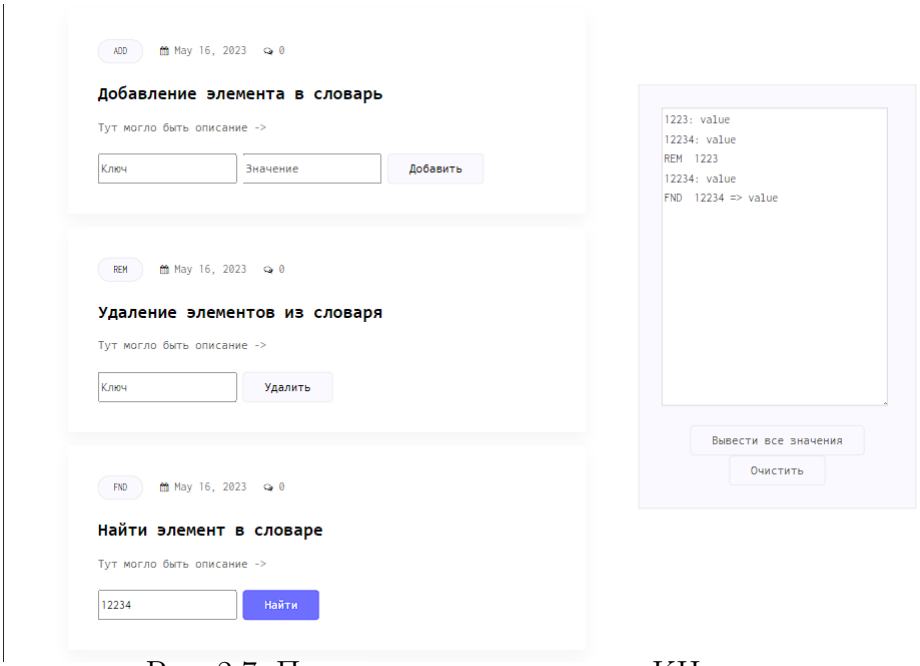

Рис. 2.7: Поиск записи по ключу в КЧ дереве

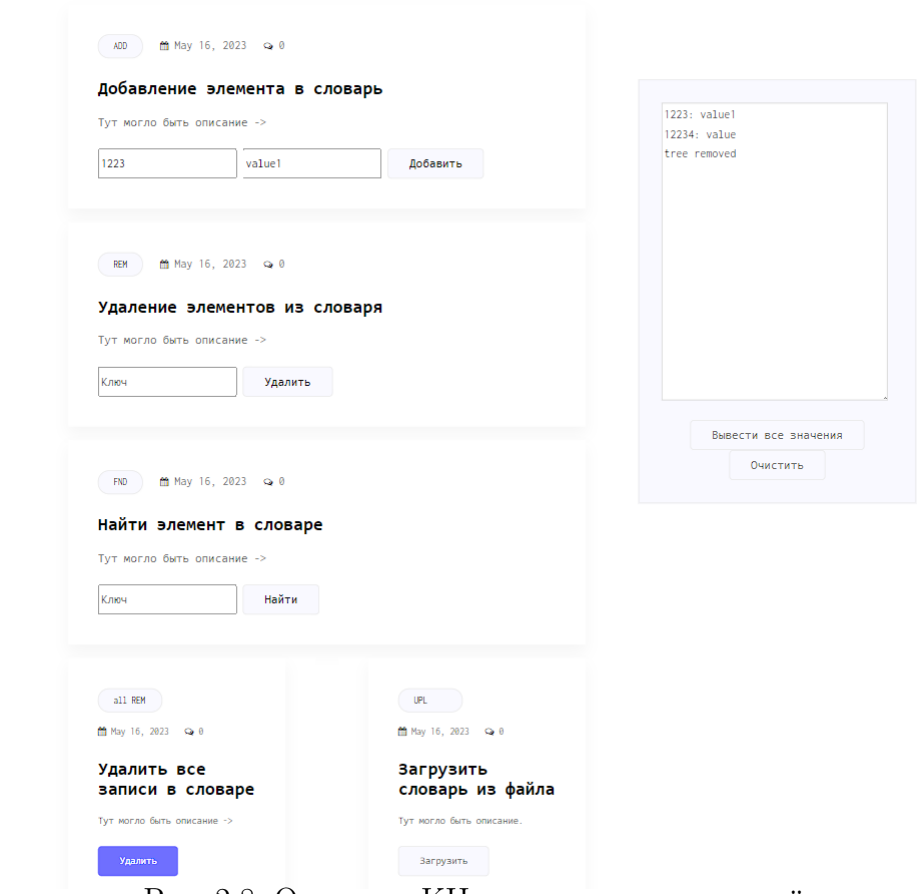

Рис. 2.8: Очистить КЧ дерево от всех записей

Рис. 2.9: Кнопочка показа структуры дерева

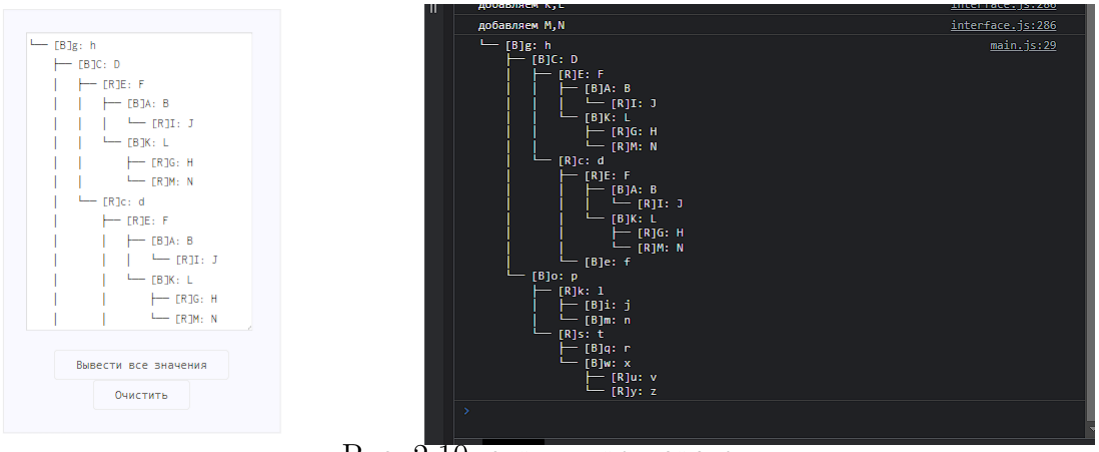

Рис. 2.10: структура дерева

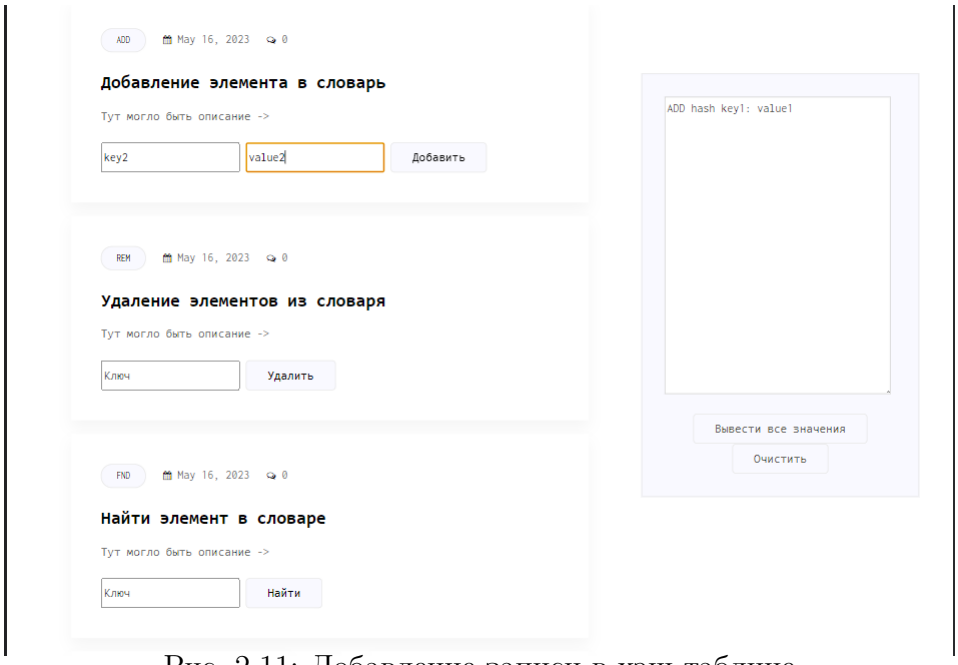

Рис. 2.11: Добавление записи в хэш таблице

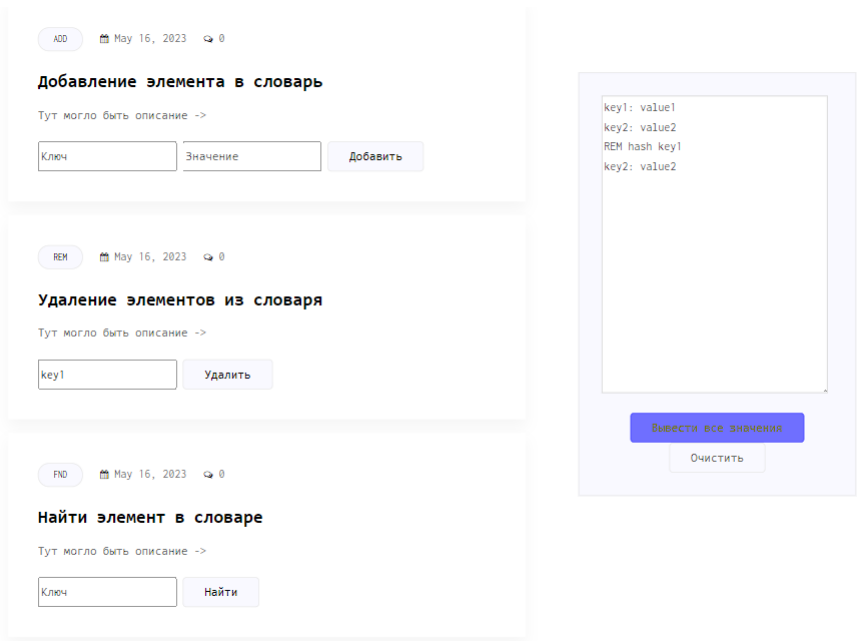

Рис. 2.12: Удаление записи из хэш таблицы

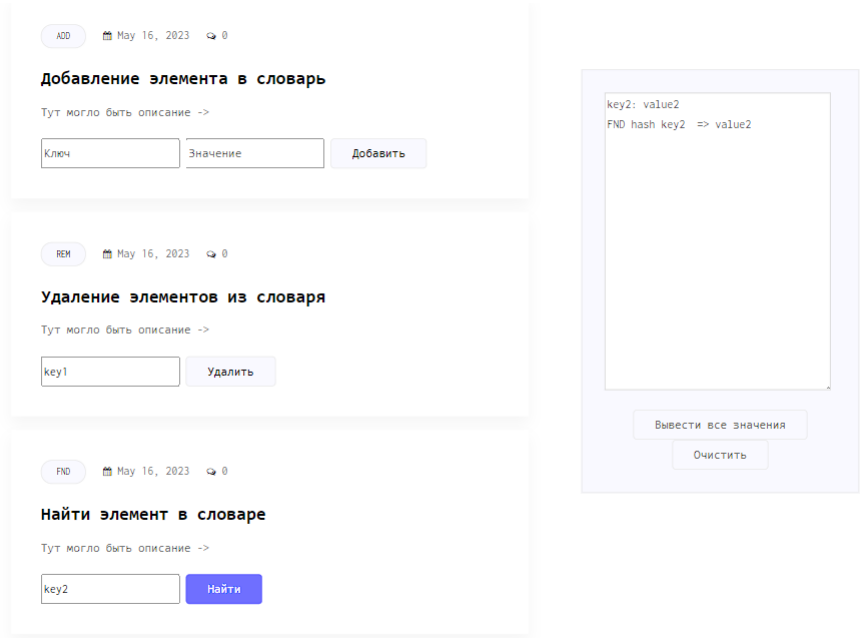

Рис. 2.13: Поиск записи по ключу в хэш таблице

Рис. 2.14: название

Заключение

В ходе данной работы была реализовван веб сервис, реализующую функционал словаря на основе красно-черного дерева и хэш таблицы.

В моей программе реализован следующий функционал:

1. Добавление записи в словарь
2. Удаление записи из словаря
3. Импорт из файла
4. Экспорт в файл
5. Очистка словаря
6. Поиск по ключу
7. Вывод структуры красно-чернорго дерева.

Преимущества

Реализован свободный доступ к веб сервису. при разработке использовался *javascript*, *html*, *css*, *php*

Сервис адаптирован для мобильных устройств.

Недостатки реализации

javascript выполняется на стороне клиента, а не на стороне сервера, что может привести к различному времени работы алгоритмов на разных компьютерах.

Масштабирование

Благодаря модульной системе проекта врозможно добавление иных структур данных, реализация иных кэш функций, Данный серсвис может быть встроен в другой сервис.

Приложение

главная страница

Листинг 2.7: LinearOrder.cpp // Реализация главной страницы сайта

```
<!doctype html>
<html lang="en">
    <head>
        <!-- Required meta tags -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
        <link rel="icon" href="img/favicon.png" type="image/png">
        <title Телематика> в науке</title>
        <!-- Bootstrap CSS -->
        <link rel="stylesheet" href="css/bootstrap.css">
        <link rel="stylesheet" href="vendors/linericon/style.css">
        <link rel="stylesheet" href="css/font-awesome.min.css">
        <link rel="stylesheet" href="vendors/owl-carousel/owl.carousel.min.css">
        <link rel="stylesheet" href="vendors/lightbox/simpleLightbox.css">
        <link rel="stylesheet" href="vendors/nice-select/css/nice-select.css">
        <link rel="stylesheet" href="vendors/animate-css/animate.css">
        <link rel="stylesheet" href="vendors/jquery-ui/jquery-ui.css">
        <!-- main css -->
        <link rel="stylesheet" href="css/style.css">
        <link rel="stylesheet" href="css/responsive.css">


        <script src="main.js"></script>
      <script src="interface.js"></script>
      <script src="md5.js"></script>
      <script src="hash_table.js"></script>

    </head>
    <body>

        <!--================Header Menu Area =================-->
        <header class="header_area">
      <div class="main_menu">
        <nav class="navbar navbar-expand-lg navbar-light">
          <div class="container box_1620">
            <!-- Brand and toggle get grouped for better mobile display -->
            <a class="navbar-brand logo_h" href="index.html"><img src="img/logo.png" alt=""></a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
              <span class="icon-bar"></span>
              <span class="icon-bar"></span>
              <span class="icon-bar"></span>
            </button>
            <!-- Collect the nav links, forms, and other content for toggling -->
            <div class="collapse navbar-collapse offset" id="navbarSupportedContent">
              <ul class="nav navbar-nav menu_nav">
                <li class="nav-item active"> <a class="nav-link" id="b1" >1</a></li>
                <li class="nav-item active"> <a class="nav-link" id="b2" >2</a></li>
                <li class="nav-item active"> <a class="nav-link" id="b3" >3</a></li>
```

```
51                 <li class="nav-item active"> <a class="nav-link" id="b4" Условия></a></li>
52
53                 <script>
54                   document.getElementById('b1').onclick = function (){alert("1.      Для выбранного текста на русском языке построить словарь на основе красночерных- деревьев. Реализовать функции добавления, удаления и поиска слова. НЕЛЬЗЯ( использовать готовые структуры данных!!!). Добавить функцию полной очистки словаря и загрузкидополнения/ словаря из текстового файла. ")}
55                   document.getElementById('b2').onclick = function (){alert("2.      Реализовать структуру данных - хештаблицу-, для которой характерны операции: добавления, удаления и поиска. На ее основе реализовать приложение - словарь. В качестве хешфункции-, можно выбрать произвольную функцию, например, брать первую букву слова. Добавить функцию полной очистки словаря и загрузкидополнения/ словаря из текстового файла. ")}
56                   document.getElementById('b3').onclick = function (){alert("3.      Для выбранного текста на русском языке построить словарь на основе B деревьев+-. Реализовать функции добавления, удаления и поиска слова. НЕЛЬЗЯ( использовать готовые структуры данных!!!). Добавить функцию полной очистки словаря и дополнения словаря из текстового файла.")}
57                   document.getElementById('b4').onclick = function (){alert("4.      Минимальная реализация включает либо пункты 1 и 2, либо пункт 2 и 3. Улучшенная реализация - все три пункта. \Максимальнаяn реализация также включает возможность оптимизации словаря например(, поиск потенциальных однокоренных слов в разных формах; самый простой критерий - 2/3 длины слова). Выбор слов, которые следует оставить в словаре - за пользователем.  Использовать готовые решения из библиотеки STL не разрешается")}
58
59                 </script>
60
61
62
63
64
65
66
67
68
69
70
71
72             </ul>
73             <ul class="nav navbar-nav navbar-right header_social ml-auto">
74               <li class="nav-item"><a href="#"><i class="fa fa-facebook "></i></a></li>
75             </ul>
76           </div>
77         </div>
78       </nav>
79     </div>
80     <div class="logo_part">
81       <div class="container">
82         <a class="logo" href="#"><img src="img/logo.png" alt=""></a>
83       </div>
84     </div>
85       </header>
86       <!--================Header Menu Area =================-->
87
88       <!--================Home Banner Area =================-->
89       <section class="home_banner_area">
90         <div class="container">
91       <div class="row">
92         <div class="col-lg-5"></div>
93         <div class="col-lg-7">
94           <div class="blog_text_slider owl-carousel">
95             <div class="item">
96               <div class="blog_text">
97                 <div class="cat">
98                   <a class="cat_btn" href="#">TG</a>
99                   <a href="#"><i class="fa fa-calendar" aria-hidden="true "></i> 2 курс 2023</a>
100
101                 </div>
102                 <a href="pages/vishnakov.html"><h4Лаба> 6</h4></a>
103                 <pЛабораторная> работа №6 по теории графов.<br>
104                 Выполнил: Рябинин Александр<br>
105                 Проверил: Востров Алексей Владимирович</p>
106
107
108               <a class="nav-link" Выбери> структуру данных:</a>
109
110
```

```
111
112                 <a class="nav-link" >
113                   <input type="radio" name="option" value="treeButton"> tree
114                     </a >
115
116                  <a class="nav-link" >
117                   <input type="radio" name="option" value="hashButton"> hash
118                   </a>
119
120
121
122
123                 </div>
124               </div>
125               <div class="item">
126                 <div class="blog_text">
127                   <div class="cat">
128
129
130                   </div>
131                   <a href="#"><h4Структура> красночерного- дерева</h4></a>
132                   <p> </p>
133                    <h3></h3>
134
135
136
137
138                   <a class="blog_btn" onclick="dictionary.printStructure()"
    Показать> структуру</a>
139                 </div>
140               </div>
141
142             </div>
143           </div>
144         </div>
145           </div>
146         </section>
147         <!--================End Home Banner Area =================-->
148
149         <!--================Blog Area ================-->
150         <section class="blog_area p_120">
151             <div class="container">
152                 <div class="row">
153                     <div class="col-lg-8">
154                         <div class="blog_left_sidebar">
155                           <!-- большой блок -->
156                             <article class="blog_style1">
157                               <div class="blog_img">
158
159                               </div>
160                               <div class="blog_text">
161                   <div class="blog_text_inner">
162                     <div class="cat">
163                       <a class="cat_btn" href="#">ADD</a>
164                        <a href="#"><i class="fa fa-calendar" aria-hidden="
    true"></i>  May 16, 2023</a>
165                          <a href="#"><i class="fa fa-comments-o" aria-hidden="
    true"></i> 0</a>
166                     </div>
167                     <a href="#"><h4Добавление> элемента в словарь </h4></a>
168                     <pТут> могло быть описание ->  </p>
169                     <div class="input-group">
170                                           <input type="text" id="input1"
    autocomplete="off" name="fkey" placeholderКлюч="">  
171                         <input type="text" id="input2" autocomplete="off"
    name="fvalue" placeholder3начение=""> 
172
173                                           <span class="input-group-btn">
174                                             <button id="addButton" class="
    blog_btn" onclick="addForm()" type="buttonДобавить"></button>
175                                           </span>
176                                       </div>
177
178
179
180
181                   </div>
182                 </div>
183                             </article>
184 <br><br><br>
185                              <article class="blog_style1">
186                               <div class="blog_img">
187
```

```
188                                 </div>
189                                 <div class="blog_text">
190                 <div class="blog_text_inner">
191                   <div class="cat">
192                     <a class="cat_btn" href="#">REM</a>
193                     <a href="#"><i class="fa fa-calendar" aria-hidden="
    true"></i>  May 16, 2023</a>
194                     <a href="#"><i class="fa fa-comments-o" aria-hidden="
    true"></i> 0</a>
195                   </div>
196                   <a href="#"><h4Удаление> элементов из словаря </h4></a>
197                   <pTут> могло быть описание ->  </p>
198                   <div class="input-group">
199                                         <input type="text" id="remove1"
    autocomplete="off" name="fkey" placeholderКлюч="">  
200                                         <span class="input-group-btn">
201                                             <button class="blog_btn"
    onclick="removeForm()" type="buttonУдалить"></button>
202                                         </span>
203                                     </div>
204
205
206
207
208                 </div>
209               </div>
210                             </article>
211
212                             <br><br><br>
213                              <article class="blog_style1">
214                                <div class="blog_img">
215
216                                </div>
217                                <div class="blog_text">
218                 <div class="blog_text_inner">
219                   <div class="cat">
220                     <a class="cat_btn" href="#">FND</a>
221                     <a href="#"><i class="fa fa-calendar" aria-hidden="
    true"></i>  May 16, 2023</a>
222                     <a href="#"><i class="fa fa-comments-o" aria-hidden="
    true"></i> 0</a>
223                   </div>
224                   <a href="#"><h4Найти> элемент в словаре </h4></a>
225                   <pTут> могло быть описание -> </p>
226                   <div class="input-group">
227                                         <input type="text" id="find1"
    autocomplete="off"  name="fkey" placeholderКлюч=""> 
228                                         <span class="input-group-btn">
229                                             <button class="blog_btn" onclick
    ="findForm()" type="buttonНайти"></button>
230                                         </span>
231                                     </div>
232
233
234
235
236                 </div>
237               </div>
238                             </article>
239
240                             <br><br><br>
241
242                             <!-- маленькие блоки -->
243                             <div class="row">
244                               <div class="col-md-6">
245                                 <article class="blog_style1 small">
246                 <div class="blog_img">
247
248                 </div>
249                 <div class="blog_text">
250                   <div class="blog_text_inner">
251                     <div class="cat">
252                       <a class="cat_btn" href="#">all REM</a>
253                       <a href="#"><i class="fa fa-calendar" aria-hidden
    ="true"></i> May 16, 2023</a>
254                       <a href="#"><i class="fa fa-comments-o" aria-
    hidden="true"></i> 0</a>
255                     </div>
```

```
256                         <a href="single-blog.html"><h4Удалить> все записи в
    словаре</h4></a>
257                         <pТут> могло быть описание -> </p>
258
259
260                         <span class="input-group-btn">
261                                                 <button class="blog_btn"
    onclick="allRemove()" type="buttonУдалить"></button>
262                                                 </span>
263                       </div>
264                     </div>
265                   </article>
266                               </div>
267                               <div class="col-md-6">
268                                 <article class="blog_style1 small">
269                     <div class="blog_img">
270
271                     </div>
272                     <div class="blog_text">
273                       <div class="blog_text_inner">
274                         <div class="cat">
275                           <a class="cat_btn" href="#">UPL</a>
276                           <a href="#"><i class="fa fa-calendar" aria-hidden
    ="true"></i>  May 16, 2023</a>
277                           <a href="#"><i class="fa fa-comments-o" aria-
    hidden="true"></i> 0</a>
278                         </div>
279                         <a href="single-blog.html"><h4Загрузить> словарь из
    файла</h4></a>
280                         <pТут> могло быть описание.</p>
281                         <a class="blog_btn" onclick="upload()" Загрузить></a>
282
283                       </div>
284                     </div>
285                   </article>
286                               </div>
287                             </div>
288
289                             <br><br><br>
290
291                             <!-- маленькие блоки -->
292                             <div class="row">
293                               <div class="col-md-6">
294                                 <article class="blog_style1 small">
295                     <div class="blog_img">
296
297
298                     </div>
299                     <div class="blog_text">
300                       <div class="blog_text_inner">
301                         <div class="cat">
302                           <a class="cat_btn" href="#">SAVE</a>
303                           <a href="#"><i class="fa fa-calendar" aria-hidden
    ="true"></i> May 16, 2023</a>
304                           <a href="#"><i class="fa fa-comments-o" aria-
    hidden="true"></i> 0</a>
305                         </div>
306                         <a href="single-blog.html"><h4Сохранить> словарь в
    файл</h4></a>
307                         <pТут> могло быть описание -> </p>
308
309
310                         <span class="input-group-btn">
311                                                 <button class="blog_btn"
    onclick="save()" type="buttonСохранить"></button>
312                                                 </span>
313                       </div>
314                     </div>
315                   </article>
316                               </div>
317                               <div class="col-md-6">
318                                 <article class="blog_style1 small">
319                     <div class="blog_img">
320
321                     </div>
322                     <div class="blog_text">
```

```
323                         <div class="blog_text_inner">
324                           <div class="cat">
325                             <a class="cat_btn" href="#">COMP</a>
326                             <a href="#"><i class="fa fa-calendar" aria-hidden
    ="true"></i>  May 16, 2023</a>
327                             <a href="#"><i class="fa fa-comments-o" aria-
    hidden="true"></i> 0</a>
328                           </div>
329                           <a ><h4Дополнить>  словарь из файла</h4></a>
330                           <p>Lorem ipsum dolor sit amet, consectetur
    adipisicing elit, sed do eiusmod tempor incidid unt ut labore et dolore
    magna aliqua.</p>
331                           <a class="blog_btn" onclick="complement()"
    Выбрать></a>
332
333
334 <input id="file-input" type="file" name="name" style="display: none;" />
335
336                         </div>
337                       </div>
338                     </article>
339                               </div>
340                             </div>
341
342
343
344                           </div>
345                         </div>
346
347
348                         <div class="col-lg-4">
349                           <div class="blog_right_sidebar">
350                             <aside class="single_sidebar_widget
    search_widget">
351                               <!--  <div class="input-group">
352                                 <input type="text" class="form-control"
    placeholder="Search Posts">
353                                 <span class="input-group-btn">
354                                   <button class="btn btn-default" type
    ="button"><i class="lnr lnr-magnifier"></i></button>
355                                 </span>
356                               </div> --><!-- /input-group -->
357                               <!-- <div class="br"></div> -->
358                             </aside>
359                             <aside class="single_sidebar_widget
    author_widget">
360                               <!-- <img class="author_img img-fluid" src="
    img/blog/author.jpg" alt=""> -->
361
362                               <!--<h4Михаил> Александрович Курочкин</h4>
363                                <p>Senior blog writer</p>
364                               <pПрофессор> кафедры телематики Института
    прикладной математики и механики СанктПетербургского- государственного
    политехнического университета. Он имеет ученую степень кандидата технических наук и
    ученое звание доцента. Преподает в СПбПУ более 47 лет и специализируется в области
    вычислительных машин, комплексов, систем и сетей и разработки человекомашинного-
    интерфейса.</p>-->
365
366
367
368
369
370                               <div class="input-group">
371                                 <textarea type="text" id="story" disabled
    style="background-color: #ffffff; " name="story" rows="15" cols="36"></
    textarea>
372                               </div>
373                               <br>
374                                 <center>
375                                   <a class="blog_btn" onclick="printTextarea
    ()" id="printВывести"> все значения</a>
376                                   <a class="blog_btn" onclick="document.
    getElementById('story').value= '' " id="printОчистить"></a>
377                               </center>
378
379                             </aside>
380
381                           </div>
382                         </div>
```

```
                </div>
            </div>
        </section>
        <!--================Blog Area =================-->

        <!--================ start footer Area  ================-->
        <footer class="footer-area p_120">
            <div class="container">
                <div class="row">
                    <div class="col-lg-3  col-md-6 col-sm-6">
                        <div class="single-footer-widget">
                            <h6 class="footer_title">О нас</h6>
                            <p><a href="https://tema.spbstu.ru/
    obshie_svedeniyaКафедра/"> телематики СанктПетербургского– Политехнического
    Университета Петра Великого</a></p>
                        </div>
                    </div>

                    <div class="col-lg-3 col-md-6 col-sm-6">
                        <div class="single-footer-widget f_social_wd">
                            <h6 class="footer_titleМы"> в соцсетях</h6>
                            <!-- <pМы> в соцсетях</p> -->
                            <div class="f_social">
                              <li class="nav-item"><a href="https://vk.com/
    telematic"><i class="fa fa-facebook"></i></a></li>

                            </div>
                        </div>
                    </div>
                </div>

            </div>
        </footer>
    <!--================ End footer Area  ================-->


        <!-- Optional JavaScript -->
        <!-- jQuery first, then Popper.js, then Bootstrap JS -->
        <script src="js/jquery-3.2.1.min.js"></script>
        <script src="js/popper.js"></script>
        <script src="js/bootstrap.min.js"></script>
        <script src="js/stellar.js"></script>
        <script src="vendors/lightbox/simpleLightbox.min.js"></script>
        <script src="vendors/nice-select/js/jquery.nice-select.min.js"></
    script>
        <script src="vendors/isotope/imagesloaded.pkgd.min.js"></script>
        <script src="vendors/isotope/isotope-min.js"></script>
        <script src="vendors/owl-carousel/owl.carousel.min.js"></script>
        <script src="vendors/jquery-ui/jquery-ui.js"></script>
        <script src="js/jquery.ajaxchimp.min.js"></script>
        <script src="js/mail-script.js"></script>
        <script src="js/theme.js"></script>
    </body>
</html>
```

Литература

[1] Ф.А Новиков. Дискретная математика для программистов: Учебник для вузов. 3-е изд. — СПб: Питер, 2008. —384 с.

[2] Графы Модели вычислений Структуры данных (2004).pdf / сост: Алексеев В.Е.Таланов В.А. доп. — Москва 2004. —164 с. — URL https://www.hse.ru/data/2012/02/08/1262556187Алексеев В.Е.Таланов В.А.Графы Модели вычислений Структуры данных (2004).pdf (дата обращения 20.05.2023)

[3] Хорстман К. С. Х79 Современный JavaScript для нетерпеливых / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2021. – 288 с.: и