

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра МО ЭВМ**

**ОТЧЕТ**  
**по лабораторной работе №5**  
**по дисциплине «Алгоритмы и структуры данных»**  
**Тема: Хеш - таблицы**

Студентка гр. 9383

\_\_\_\_\_

Лапина А.А.

Преподаватель

\_\_\_\_\_

Попова Е.В.

Санкт-Петербург

2020

**Цель работы.**

Ознакомиться с хеш-таблицами. Реализовать хеш-таблицу с цепочками.

**Задание.**

Вариант №23.

Хеш-таблица: с цепочками; действие: 1+2a

- 1) По заданной последовательности элементов Elem построить структуру данных определённого типа – хеш-таблицу;
- 2) Выполнить:
  - а) Для построенной структуры данных проверить, входит ли в неё элемент e типа Elem, и если входит, то в скольких экземплярах. Добавить элемент e в структуру данных. Предусмотреть возможность повторного выполнения с другим элементом.

## **Теория.**

**Хеш-таблица** — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

**Хеш-функцией** - это математическая или иная функция, которая для строки произвольной длины вычисляет некоторое целое значение или некоторую другую строку фиксированной длины.

**Метод цепочек** — метод, где каждая ячейка массива  $H$  является указателем на цепочку пар ключ-значение, соответствующих одному и тому же хеш-значению ключа. Коллизии просто приводят к тому, что появляются цепочки длиной более одного элемента.

**Коллизия (конфликт)** – событие, когда для разных ключей  $K_i \neq K_j$  функция расстановки  $H$  даёт один и тот же адрес  $H(K_i) = H(K_j)$ .

### **Разрешение коллизий с помощью цепочек.**

В зависимости от того нужна ли нам уникальность значений операции вставки у нас будет работать за разное время. Если не важна, то мы используем список, время вставки в который будет в худшем случае равна  $O(1)$ . Иначе мы проверяем есть ли в списке данный элемент, а потом в случае его отсутствия мы его добавляем. В таком случае вставка элемента в худшем случае будет выполнена за  $O(n)$ .

Операции поиска или удаления элемента требуют просмотра всех элементов соответствующей ему цепочки, чтобы найти в ней элемент с заданным ключом. Для добавления элемента нужно добавить элемент в конец или начало соответствующей цепочки и в случае, если коэффициент заполнения станет слишком велик, увеличить размер массива  $H$  и перестроить таблицу. При предположении, что каждый элемент может попасть в любую позицию таблицы  $H$  с равной вероятностью и независимо от того, куда попал любой

другой элемент, среднее время работы операции поиска элемента составляет  $\Theta(1 + \alpha)$ , где  $\alpha$  — коэффициент заполнения таблицы.

Существуют следующие подходы к реализации:

- 1) Прямой метод цепочек, когда элементы списка находятся в самой хеш-таблице
- 2) Метод, когда элементы списка размещаются в области переполнения (в таблице символов), а хеш-таблица представляет собой таблицу индексов.
- 3) Метод цепочек (использующий таблицу индексов с некоторыми изменениями).

В данной работе использован 3 метод реализации хэш-таблиц.

**Метод цепочек (использующий таблицу индексов с некоторыми изменениями).**

Таблица символов реализована в виде отдельных цепочек, «присоединённая» каждая к своему индексу. Каждая ячейка  $i$  массива  $H$  содержит указатель на начало списка всех элементов, хеш-код которых равен  $i$ , либо указывает на их отсутствие. Коллизии приводят к тому, что появляются списки размером больше одного элемента.

**Преимущества:**

- 1) Не требует перемещения элементов, т.к. каждый элемент таблицы индексов всегда указывает на начало цепочки.
- 2) Легко обрабатывать переполнение таблицы символов, т.к. ограничена таблица индексов, а элементы можно добавлять в список.

**Недостатки:**

При неудачном выборе  $N$  — числа, отвечающего за размер таблицы индексов (то есть количество индексов), будут использованы не все индексы. Например, если использовать хэш-функцию как на рисунке 1

$h(x) = \left( \sum_{i=0}^{len s} ASCII(s[i]) \right) \bmod N$  для  $N=10$  используются только 4 индекса 1, 2, 3, 5, но получается 3 цепочки.

На рисунке 1 изображено представление хэш-таблицы, реализованной методом цепочек для функции  $h(x) = \left( \sum_{i=0}^{len s} ASCII(s[i]) \right) \bmod 11$  (Остаток от деления на 11 суммы ASCII-кодов символов, входящих в строку). Где  $i$  — это счетчик для прохода по строке от нулевого элемента до последнего элемента строки ( $len s$  — длина строки, то есть с его помощью узнаем номер последнего символа). Данная хэш-функция была рассмотрена на лекционном занятии. В данной работе реализуем ее с помощью программного кода.

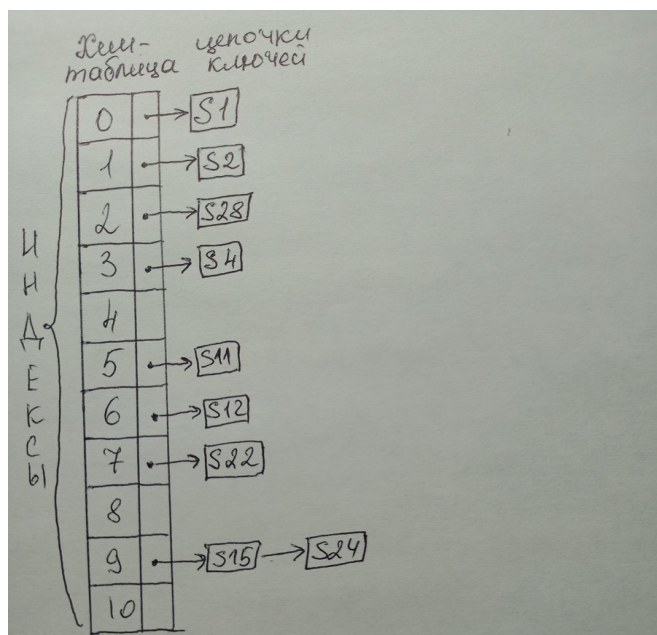


Рисунок 1 — Хэш-таблица, реализованная методом цепочек

### Алгоритм заполнения хэш-таблицы:

На вход подается элемент. Находим индекс для него, с помощью хэш-функции. Затем обращаемся по этому индексу к таблице ключей и если там по этому индексу ключей нет, то вставляем этот ключ в качестве первого. А если там уже лежит ключ (ключи), то вставляем новый к предыдущему в цепочку.

### **Ход работы:**

1. Выбрать метод цепочек.
2. Разработать алгоритм поиска и вставки.
4. Протестировать программу.

### **Выполнение работы:**

В данной задаче необходимо построить структуру данных — хэш-таблицу. Для этого создаем массив `hash` из  $N = 11$  элементов (таблица индексов). Также для реализации данного метода создана таблица символов (ключей) с помощью класса `Elem`.

В программе использованы следующие функции:

- 1) `start()` - заполняет ключи значением «NULL», чтобы понимать лежит ли ключ по данному индексу или нет;
- 2) `print()` - печатает хэш-таблицу;
- 3) `func()` - это хэш — функция, с помощью которой будем вычислять индекс ключа;
- 4) `find()` - функция поиска элемента по ключу. (Вычисляем индекс элемента, с помощью `func()` и обращаемся по полученному из нее индексу. Смотрим лежит ли такой ключ там или нет). После поиска первого элемента спрашиваем у пользователя нужно ли искать еще элемент. Если он вводит «0», то функция завершает работу, если «1», то ищем новый элемент, пока пользователь не выберет «0».
- 4) `main()` — используем вышеперечисленные функции. Создаем таблицу индексов и символов. Заполняем хэш-таблицу, с помощью ключей, который вводит пользователь (для данной программы «!» - является концом ввода ключей), печатает хэш-таблицу, выполняем поиск элементов(действие 2а из задания), пока пользователь не введет «0».

### **Пример работы программы:**

Вводим элементы до «!»

**1) Входные данные:**

S1

S2

S4

S11

S12

S15

S22

S24

S28

! //для конца ввода

//пара для поиска

S1

// для продолжения ввода

1

//пара для поиска

W1

//для выхода из программы

0

**Выходные данные:**

//Получившаяся хеш-таблица с цепочками:

0 - S1

1 - S2

2 - S28

3 - S4

5 - S11

6 - S12

7 - S22

9 - S15 S24

Введите элемент для поиска? //вводим S1

Да, такой элемент есть! В количестве: 1

0 - S1 S1

1 - S2

2 - S28

3 - S4

5 - S11

6 - S12

7 - S22

9 - S15 S24

Найти еще элемент? Если да - нажмите 1, если нет - 0

//вводим 1, чтобы найти еще элемент

Введите элемент для поиска? //вводим W1

Такого элемента нет!

0 - S1 S1

1 - S2

2 - S28

3 - S4

4 - W1

5 - S11

6 - S12

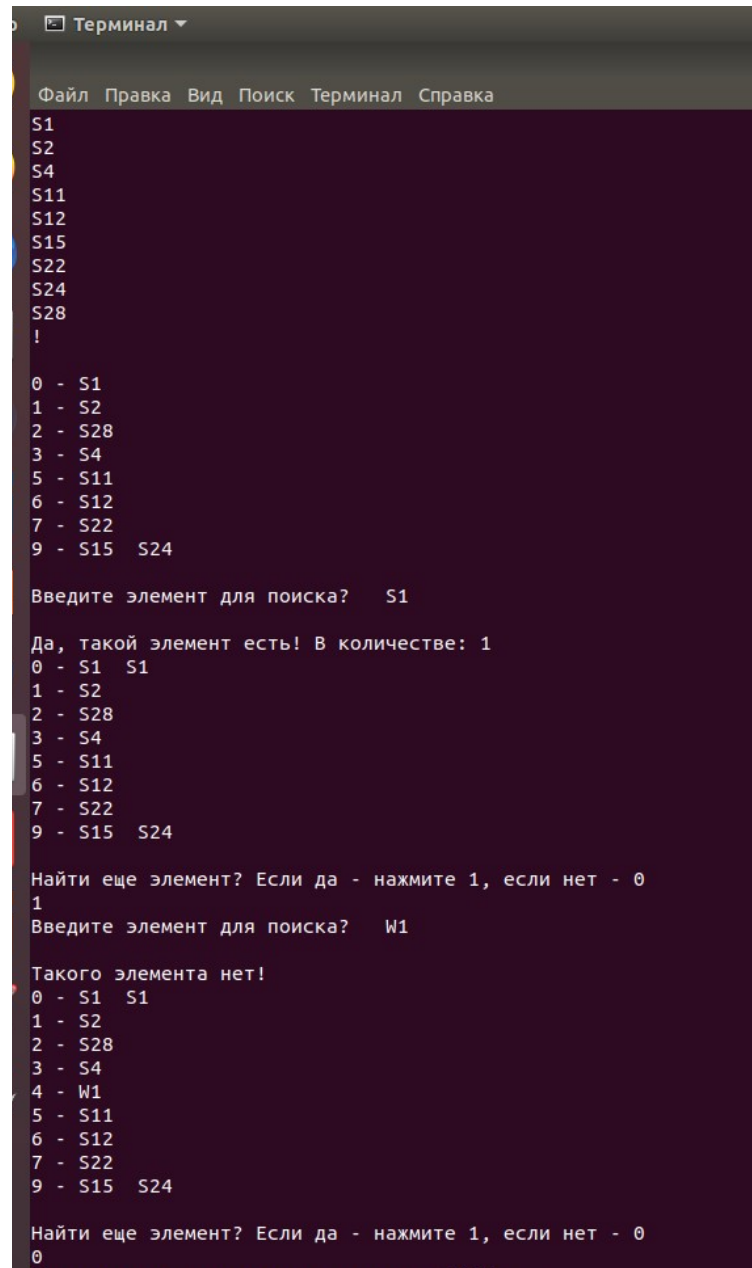
7 - S22



9 - S15 S24

Найти еще элемент? Если да - нажмите 1, если нет - 0

//вводим 0 для выхода из программы



```
Терминал
Файл Правка Вид Поиск Терминал Справка
S1
S2
S4
S11
S12
S15
S22
S24
S28
!
0 - S1
1 - S2
2 - S28
3 - S4
5 - S11
6 - S12
7 - S22
9 - S15 S24

Введите элемент для поиска? S1

Да, такой элемент есть! В количестве: 1
0 - S1 S1
1 - S2
2 - S28
3 - S4
5 - S11
6 - S12
7 - S22
9 - S15 S24

Найти еще элемент? Если да - нажмите 1, если нет - 0
1
Введите элемент для поиска? W1

Такого элемента нет!
0 - S1 S1
1 - S2
2 - S28
3 - S4
4 - W1
5 - S11
6 - S12
7 - S22
9 - S15 S24

Найти еще элемент? Если да - нажмите 1, если нет - 0
0
```

Рисунок 2 — Демонстрация работы программы с входными данными №1

## 2) Входные данные:

//вводим элементы

vgrvsv

we

gfd

vf

we

!

//элемент для поиска

we

//для выхода

0

**Выходные данные:**

0 - we vf we

4 - vgrvsv

8 - gfd

Введите элемент для поиска? //вводим we

Да, такой элемент есть! В количестве: 2

0 - we vf we we

4 - vgrvsv

8 - gfd

Найти еще элемент? Если да - нажмите 1, если нет - 0

//0 для выхода

```
Терминал
Файл Правка Вид Поиск Терминал Справка
vgrvsv
we
gfd
vf
we
!

0 - we vf we
4 - vgrvsv
8 - gfd

Введите элемент для поиска? we

Да, такой элемент есть! В количестве: 2
0 - we vf we we
4 - vgrvsv
8 - gfd

Найти еще элемент? Если да - нажмите 1, если нет - 0
0
Для закрытия данного окна нажмите <ВВОД>...
```

Рисунок 3 — Демонстрация работы программы с входными данными №2

### 3) Входные данные:

fgh  
lkm  
ssxcegeqve  
qwf  
!  
//элемент для поиска  
qwf  
//для выхода  
0

### Выходные данные:

1 - fgh  
4 - qwf  
5 - lkm  
8 - ssxcegeqve

Введите элемент для поиска? //вводим qwf

Да, такой элемент есть! В количестве: 1

1 - fgh

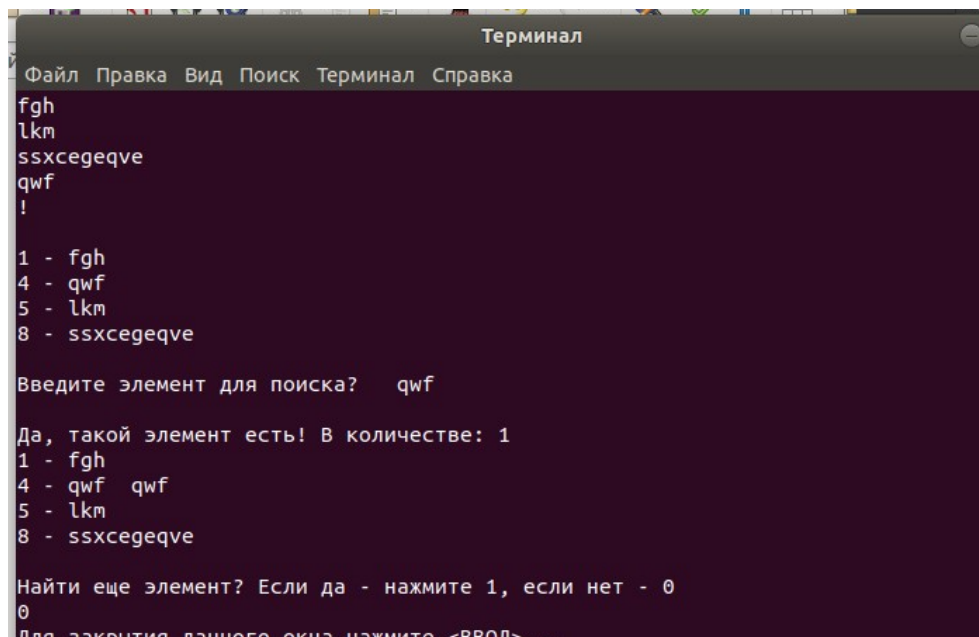
4 - qwf qwf

5 - lkm

8 - ssxcegeqve

Найти еще элемент? Если да - нажмите 1, если нет - 0

//вводим для выхода 0



```
Терминал
Файл Правка Вид Поиск Терминал Справка
fgh
lkm
ssxcegeqve
qwф
!

1 - fgh
4 - qwф qwф
5 - lkm
8 - ssxcegeqve

Введите элемент для поиска?   qwф

Да, такой элемент есть! В количестве: 1
1 - fgh
4 - qwф qwф
5 - lkm
8 - ssxcegeqve

Найти еще элемент? Если да - нажмите 1, если нет - 0
0
Для закрытия данного окна нажмите «ВВОД»
```

Рисунок 4 — Демонстрация работы программы с входными данными №3

#### 4) Входные данные:

we

vf

gu

!

//элемент для поиска

vf

//для выхода

0

### **Выходные данные:**

0 - we vf gu

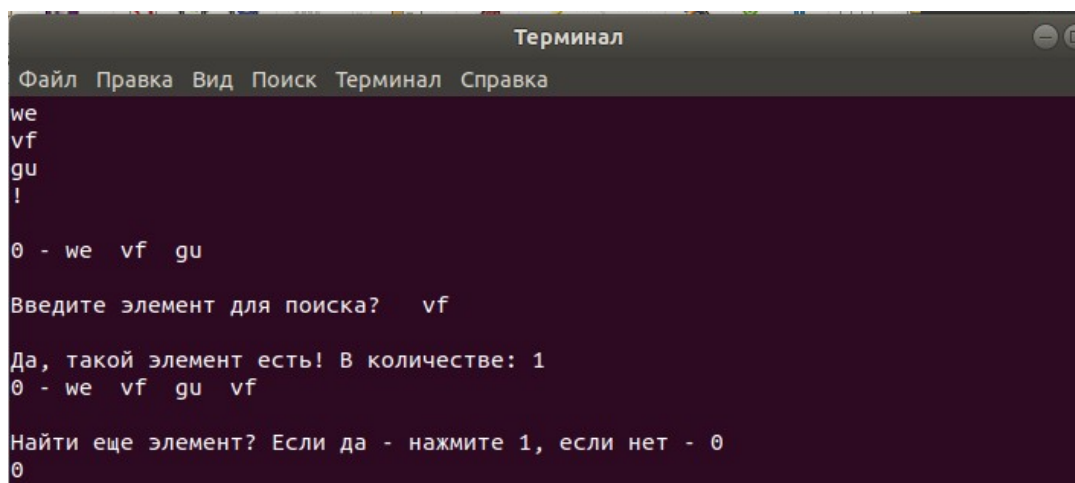
Введите элемент для поиска? //вводим vf

Да, такой элемент есть! В количестве: 1

0 - we vf gu vf

Найти еще элемент? Если да - нажмите 1, если нет - 0

//вводим для выхода 0



```
Терминал
Файл Правка Вид Поиск Терминал Справка
we
vf
gu
!
0 - we vf gu
Введите элемент для поиска? vf
Да, такой элемент есть! В количестве: 1
0 - we vf gu vf
Найти еще элемент? Если да - нажмите 1, если нет - 0
0
```

Рисунок 5 — Демонстрация работы программы с входными данными №4

### **Вывод.**

Изучены хеш-таблицы и реализована хэш-таблица с цепочками.

## Приложение А

### Исходный код программы

Название файла: main.cpp

```
#include <iostream>
#include <cstring>
#define N 11

using namespace std;

class Elem{
public:
    string key[10];
    int value[10];
    Elem* next;

    void start(){
        for (int i = 0; i < 10; i++){
            key[i] = "NULL";
        }
    }
};

int print(Elem hash[N]){
    for (int i = 0; i < N; i++){
        if (hash[i].key[0] != "NULL"){
            cout << i << " - ";
            int j = 0;
            while (hash[i].key[j] != "NULL"){
                cout << hash[i].key[j] << " ";
                j++;
            }
            cout << "\n";
        }
    }
    cout << "\n";
}

int func(string s){
    int i = 0;
    int res = 0;
    while (s[i] != '\0'){
        res = (int)s[i] + res;
        i++;
    }
    res = res % N;
    return res;
}

void find(Elem hash[N]){
    int find = 1;
    while(find){
```

```

    cout << "Введите элемент для поиска? ";
    string find_key;
    cin >> find_key;
    int f = func(find_key);
    int count = 0;
    int j = 0;
    while (hash[f].key[j]!="NULL"){
        if(hash[f].key[j]==find_key)
            count++;
        j++;
    }
    cout << "\n";
    if(j == 0 )
        cout << "Такого элемента нет!\n";
    else
        cout << "Да, такой элемент есть! В количестве: " << count << "\n";
    hash[f].key[j] = find_key;
    print(hash);
    cout << "Найти еще элемент? Если да - нажмите 1, если нет - 0\n";
    cin >> find;
}

int main()
{
    Elem hash[N];
    int i = 0;
    for(i = 0; i<N; i++){
        hash[i].start();
    }
    string s;
    i = 0;
    cin>>s;
    int j = 0;
    while(s != "!"){
        i = func(s);
        j = 0;
        if (hash[i].key[j]=="NULL"){
            hash[i].key[j] = s;
        }
        else{
            while (hash[i].key[j]!="NULL")
                j++;
            hash[i].key[j] = s;
        }
        cin>>s;
    }
    cout << "\n";

    print(hash);
    find(hash);
    return 0;
}

```