

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФГБОУ ВО «Брянский государственный технический университет»

Факультет информационных технологий.

Кафедра «Информатика и программное обеспечение»

Отчет по лабораторной работе

«Хеширование»

Выполнил:

студент гр. О-18-ПРИ-РПС-Б

Подгорняк А.А.

Проверил:

к.т.н., проф. В. К. Гулаков

Брянск 2020

**СОДЕРЖАНИЕ**

[1. ЗАДАНИЕ 3](#_Toc43848296)

[2. Описание программы 4](#_Toc43848297)

[3. Демонстрация программы 7](#_Toc43848298)

[4. ПРИЛОЖЕНИЕ 8](#_Toc43848299)

## ЗАДАНИЕ

Реализовать метод открытого хеширования. Исходные ключи – любые слова (например – фамилии). Размер хеш-таблицы должен задаваться в программе с помощью константы m. Хеш-функция – такая же, что и в задании 15, но делить надо на константу m.

Программа должна выполнять следующие действия:

* добавление нового ключа в таблицу с подсчетом сделанных при этом сравнений
* поиск заданного ключа в таблице с подсчетом сделанных при этом сравнений
* вывод текущего состояния таблицы на экран
* удаление заданного ключа из таблицы

## Описание программы

Программа представляет собой консольное приложение, которое решает поставленную задачу

Для преобразования текстовых ключей в числовые значения использовать суммирование кодов символов текстового ключа. Преобразование числового кода ключа в значение индекса выполнить с помощью простейшей хеш-функции, которая берет остаток от целочисленного деления кода на размер хеш-таблицы

|  |
| --- |
| int Hash(string key,int m)  {  int temp = 0;  for (int i = 0; i < key.size(); i++)  {  temp += key[i];  }  temp %= m;  return temp;  } |

Листинг 1

При запуске программы пользователю предлагается выбор:

* Добавить элемент (Листинг 2)
* Удалить элемент (Листинг 3)
* Вывести таблицу
* Поиск по ключу (Листинг 4)

|  |
| --- |
| int Add(vector <list <string>> &table,string key,int hash)  {  int compares = 0;  for (int i = 0; i < table.size(); i++)  {  compares++;  if(hash==i)  {  table[i].push\_back(key);  cout << "Было выполнено " << compares << " сравнений(ия)" << endl<<endl;  return 1;  }    }  return 0;  } |

Листинг 2

|  |
| --- |
| int del(vector <list <string>>& table, string key)  {    for (int i = 0; i < table.size(); i++)  {    auto begin = table[i].begin();  auto end = table[i].end();  for (begin; begin != end; begin++) {  if (\*begin == key)  {  table[i].erase(begin);  return 1;  }  }  }  return 0;  } |

Листинг 3

|  |
| --- |
| int find(vector <list <string>>& table, string key)  {  int compares = 0;  bool flag = false, find = false;  for (int i = 0; i < table.size(); i++)  {    auto begin = table[i].begin();  auto end = table[i].end();  for (begin; begin != end; begin++) {  compares++;  if (\*begin == key)  {  cout << "Хеш: " << i << " " << "Ключ: " <<\*begin<< endl;  cout << "Было выполнено " << compares << " сравнений(ия)" << endl<<endl;    }  }    if (find)  flag = true;  if(flag)  return 1;  }  return 0;  } |

Листинг 4

## Демонстрация программы

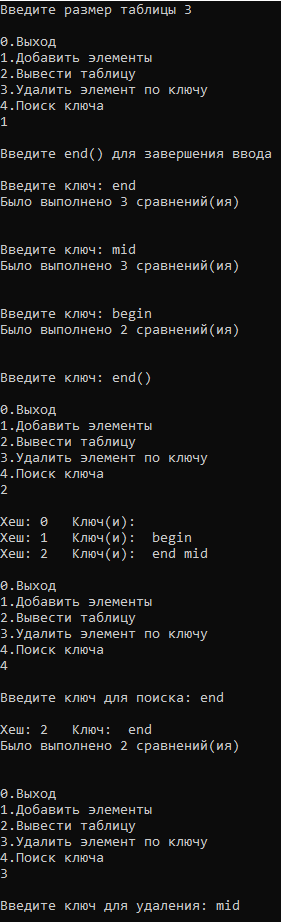


Рис. 1 Пример работы программы

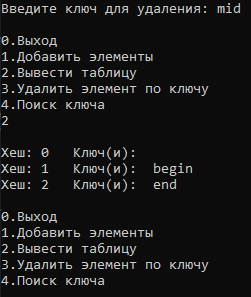


Рис. 2 Пример работы программы

В данном примере пользователь создал таблицу размера три. Добавил три ключа, нашел один из них, а затем удалил один.

## ПРИЛОЖЕНИЕ

Исходный код всей программы(Листинг 4)

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include <fstream>  #include <string>  #include <ctime>  #include <queue>  #include <stack>  using namespace std;  struct City  {  int Country;  bool Tax;  int way;  int coast;  };  struct Edge {  int begin;  int end;  };  class RailwayGraph  {  public:  RailwayGraph()  {  CountNodes = 0;  MatrAdj = NULL;  nodes = NULL;  }  bool InputFromFile()  {  system("cls");  string namefile = "Data.txt";  ifstream fin(namefile);  if (!fin.is\_open())  {  cout << "Ошибка при открытии файла для чтения\n";  fin.close();  return false;  }  else  {  fin >> CountNodes;  nodes = new City[CountNodes];  for (int i = 0; i < CountNodes; i++)  {  nodes[i].way = -1;  nodes[i].coast = 0;  }  MatrAdj = new int\* [CountNodes];  for (int i = 0; i < CountNodes; i++)  {  MatrAdj[i] = new int[CountNodes];  }  int flag;  for (int i = 0; i < CountNodes; i++)  {  for (int j = 0; j < CountNodes; j++)  {  fin >> MatrAdj[i][j];  }  fin >> nodes[i].Country >> flag;  if (flag > 0)nodes[i].Tax = true;  else nodes[i].Tax = false;  }  }  fin.close();  return true;  }  void HandInput()  {  system("cls");  do  {  cout << endl << "Введите кол-во вершин: ";  cin >> CountNodes;  } while (CountNodes < 1);  nodes = new City[CountNodes];  for (int i = 0; i < CountNodes; i++)  {  nodes[i].way = -1;  nodes[i].coast = 0;  }  MatrAdj = new int\* [CountNodes];  for (int i = 0; i < CountNodes; i++)  {  MatrAdj[i] = new int[CountNodes];  }  int flag;  cout << endl << endl << "Введите матрицу смежности и в конце строк страну города(число) и есть(1) или нет(0) доп. налог" << endl;  for (int i = 0; i < CountNodes; i++)  {  for (int j = 0; j < CountNodes; j++)  {  cin >> MatrAdj[i][j];  }  cin >> nodes[i].Country >> flag;  if (flag > 0)nodes[i].Tax = true;  else nodes[i].Tax = false;  }  }  void print()  {  cout << setw(10) << "Вершины";  for (int i = 0; i < CountNodes; i++)  {  cout << setw(8) << "Node\_" << i;  }  cout << setw(9) << "Country" << setw(9) << "Tax";  cout << endl;  for (int i = 0; i < CountNodes; i++)  {  cout << setw(9) << "Node\_" << i;  for (int j = 0; j < CountNodes; j++)  {  cout << setw(9) << MatrAdj[i][j];  }  cout << setw(9) << nodes[i].Country;  if (nodes[i].Tax)cout << setw(9) << "true";  else cout << setw(9) << "false";  cout << endl;  }  cout << endl << endl;  }  void find()  {  do  {  cout << endl << "Введите номер начального города A: ";  cin >> A;  } while (0 > A || A >= CountNodes);  do  {  cout << endl << "Введите номер конечного города B: ";  cin >> B;  } while (0 > B || B >= CountNodes);  Edge e;  Queue.push(A);  nodes[A].way = 0;  nodes[A].coast = 0;  while (!Queue.empty())  {  int node = Queue.front();  Queue.pop();    for (int j = 0; j < CountNodes; j++)  {  if (MatrAdj[node][j] != 0 && !nodes[j].Tax)  {  if (nodes[j].way == -1)  {  Queue.push(j);  nodes[j].way = nodes[node].way + 1;  if (nodes[j].Country != nodes[node].Country)nodes[j].coast = nodes[node].coast + 1;  e.begin = node; e.end = j;  Edges.push(e);  if (node == B) break;  }  else if (nodes[node].coast + 1 < nodes[j].coast)  {  Queue.push(j);  nodes[j].way = nodes[node].way + 1;  if (nodes[j].Country != nodes[node].Country)nodes[j].coast = nodes[node].coast + 1;  e.begin = node; e.end = j;  Edges.push(e);  if (node == B) break;  }  else if (nodes[node].way + 1 < nodes[j].way && nodes[node].coast + 1 == nodes[j].coast)  {  Queue.push(j);  nodes[j].way = nodes[node].way + 1;  if (nodes[j].Country != nodes[node].Country)nodes[j].coast = nodes[node].coast + 1;  e.begin = node; e.end = j;  Edges.push(e);  if (node == B) break;  }  }  }  }  cout << "Путь до вершины " << B << endl;  cout << B;  while (!Edges.empty())  {  e = Edges.top();  Edges.pop();  if (e.end == B)  {  B = e.begin;  cout << " <- " << B;  }  }  }  private:  int CountNodes; //Кол-во вершин  int\*\* MatrAdj; //Матрица смежности  City\* nodes; //вершины  queue<int>Queue; //Очередь для обхода в ширину  stack<Edge>Edges; //Стек для сохранения пути  int A; //Начальная вершина  int B; //Конечная вершина  };  void main()  {  setlocale(LC\_ALL, "Russian");  RailwayGraph railway;  int flag;  do  {  system("cls");  do  {  cout << endl << "Введите:" << endl << "1 - Ручной ввод" << endl << "2 - Ввод из файла" << endl;  cin >> flag;  } while (flag != 1 && flag != 2);  if (flag == 1)railway.HandInput();  else railway.InputFromFile();  railway.print();  do  {  railway.find();  do  {  cout << endl << "Введите:" << endl << "0 - чтобы выйти" << endl << "1 - что бы продолжить" << endl << "2 - что бы задать новый граф" << endl;  cin >> flag;  } while (flag != 1 && flag != 0 && flag != 2);  } while (flag != 2 && flag != 0);  } while (flag != 0);  } |

Листинг 4