ГУАП

# КАФЕДРА №43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| старший преподаватель |  | 02.11.2023 |  | С.А. Рогачев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 4 |
| ХЕШИРОВАНИЕ ДАННЫХ |
| по курсу: АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4232 |  | 02.11.2023 |  | М.В.Куриш |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

1. **Цель работы**

Целью работы является изучение методов хеширования данных и получение практических навыков реализации хеш-таблиц.

1. **Задание на лабораторную работу**

Составить хеш-функцию в соответствии с заданным вариантом и проанализировать ее. При необходимости доработать хеш-функцию. Используя полученную хеш-функцию разработать на языке программирования высокого уровня программу, которая должна выполнять следующие функции:

* создавать хеш-таблицу;
* добавлять элементы в хеш-таблицу;
* просматривать хеш-таблицу;
* искать элементы в хеш-таблице по номеру сегмента/по ключу;
* выгружать содержимое хеш-таблицы в файл для построения гистограммы в MS Excel, или в аналогичном подходящем ПО;
* удалять элементы из хеш-таблицы;
* в программе должна быть реализована проверка формата вводимого ключа;
* при удалении элементов из хэш-таблицы, в программе должен быть реализован алгоритм, позволяющий искать элементы, вызвавшие коллизию с удаленным;
* в программе должен быть реализован алгоритм, обрабатывающий ситуации с переполнением хэш-таблицы.

Метод разрешения коллизий выбрать в соответствии с заданным вариантом.

Вариант 22.

|  |  |  |  |
| --- | --- | --- | --- |
| **№ вар.** | **Формат ключа** | **Количество сегментов** | **Метод хеширования (разрешения коллизий)** |
| 22 | цББББц | 2000 | Линейное опробование |

Где «ц» – это цифра 0…9; «Б» – это большая буква латиницы A…Z.

1. **Описание хеш-функции**

Разработанная хеш-функция вычисляет хеш ключа, имеющего формат “цББББц” по следующей формуле:



Где hash – значение хеша, key – ключ, массив символов, j – индекс символа в массиве key.

Изначально hash имеет значение 0. Приведенная выше формула выполняется для каждого символа ключа (т.е. j принимает значения от 0 до 5 в соответствии с форматом ключа). Итоговое значение может оказаться больше количества сегментов. К примеру, максимально возможное значение







Количество же сегментов по заданию равно 2000. Следовательно, надо свести итоговое значение хеш-функции к допустимому, взяв остаток от деления значения на количество сегментов:

*hash = hash % numOfSegments*

1. **Результаты анализа хеш-функции**

Чтобы проверить разработанную хеш-функцию на корректность работы, используем её на 6000 ключей (в три раза больше, чем сегментов). Случайно сгенерированные ключи получат тот или иной хеш. В массиве, где каждый индекс подразумевает под собой хеш-значение, увеличим хранимое значение (изначально 0) при совпадении хеш-значения ключа с данным индексом. Результат работы (т.е. массив с частотностью получения хеш-значений) запишем в файл и экспортируем в MS Excel для построения диаграммы и проведения анализа.

Листинг программы, выполняющей проверку хеш-функции

|  |
| --- |
| #include <random>  #include <vector>  #include <string>  #include <fstream>  using namespace std;  int NumOfSegments = 2000;  void hashCode(vector<string>& generatedKeys, vector<int>& segmentsArray) {  int hashNum = 0;  for (int i = 0; i < generatedKeys.size(); i++) {  for (int j = 0; j < 6; j++)  hashNum += generatedKeys[i][j] \* ((j + 1) \* 3) + generatedKeys[i][j] \* j \* j \* j;  hashNum = hashNum % NumOfSegments;  segmentsArray[hashNum]++;  hashNum = 0;  }  }  void generateKeys(vector<string>& keysArray, int count) {  random\_device rd;  mt19937 mt(rd());  uniform\_int\_distribution<int> uniLetters(65, 90);  uniform\_int\_distribution<int> uniNumbers(0, 9);  while (count != 0) {  keysArray.push\_back(to\_string(uniNumbers(mt)) +  (char)uniLetters(mt) +  (char)uniLetters(mt) +  (char)uniLetters(mt) +  (char)uniLetters(mt) +  to\_string(uniNumbers(mt)));  count--;  }  }  int main()  {  vector<string> keys;  generateKeys(keys, NumOfSegments \* 3);  vector<int> segments(NumOfSegments, 0);  hashCode(keys, segments);  ofstream f;  f.open("results.txt");  for (int i = 0; i < segments.size(); i++) {  f << segments[i] << " ";  }  f.close();  } |

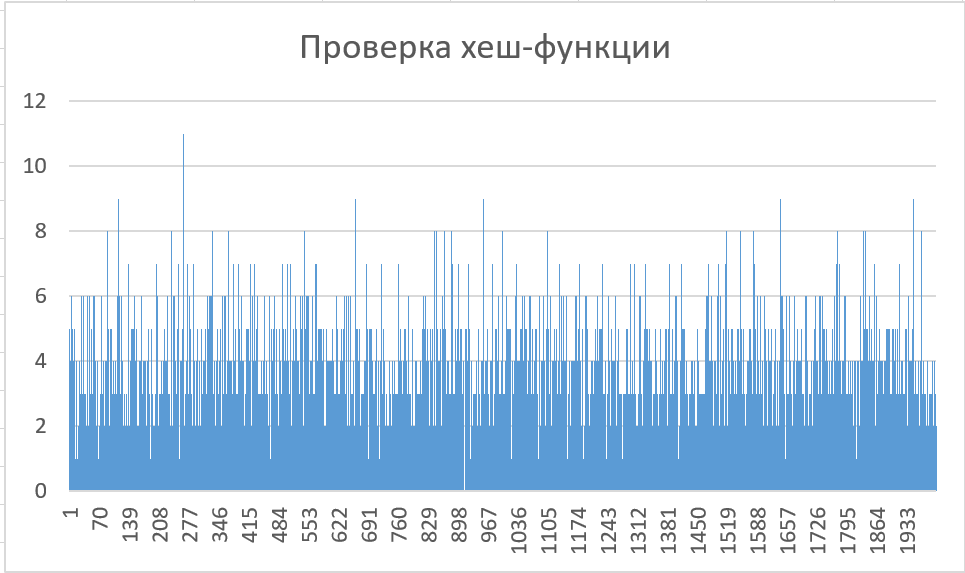


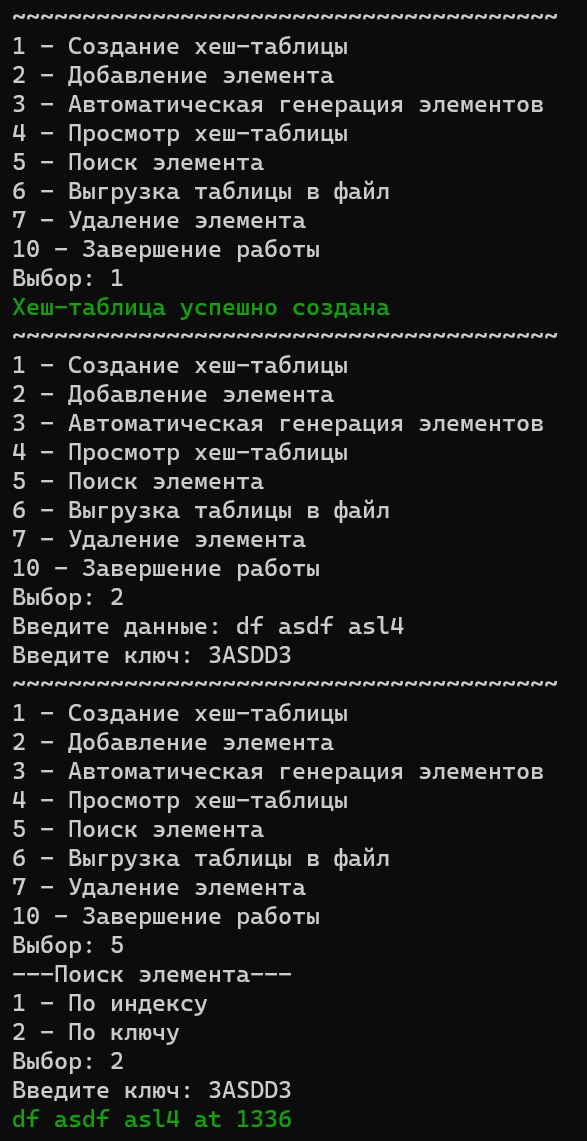
Рисунок 1 – Диаграмма распределения хеш-значений случаных ключей

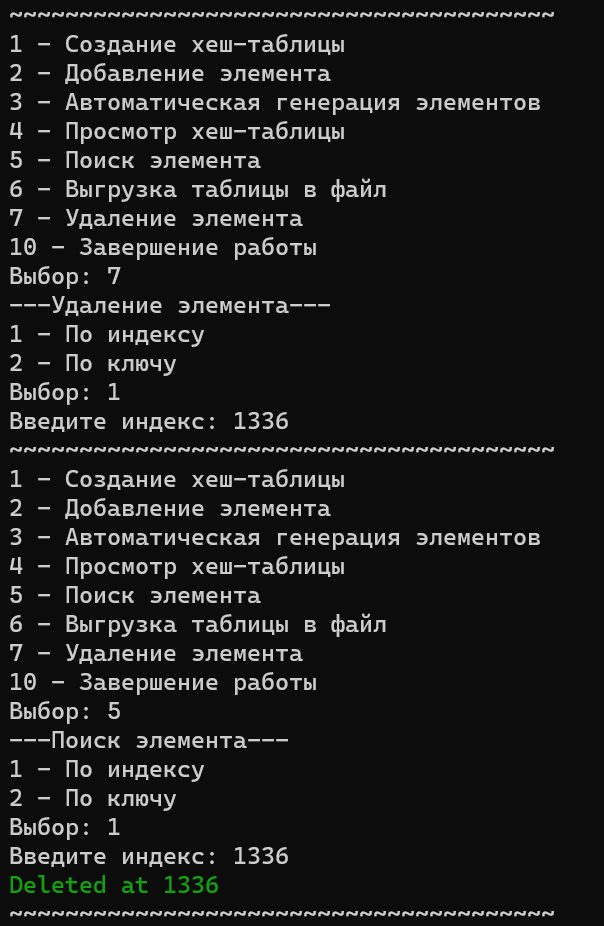
По рисунку 1 можно сделать вывод о том, что хеш-функция достаточно равномерно генерирует значения, а значит, хеш-функция разработана удачно и её можно использовать для реализации хеш-таблицы.

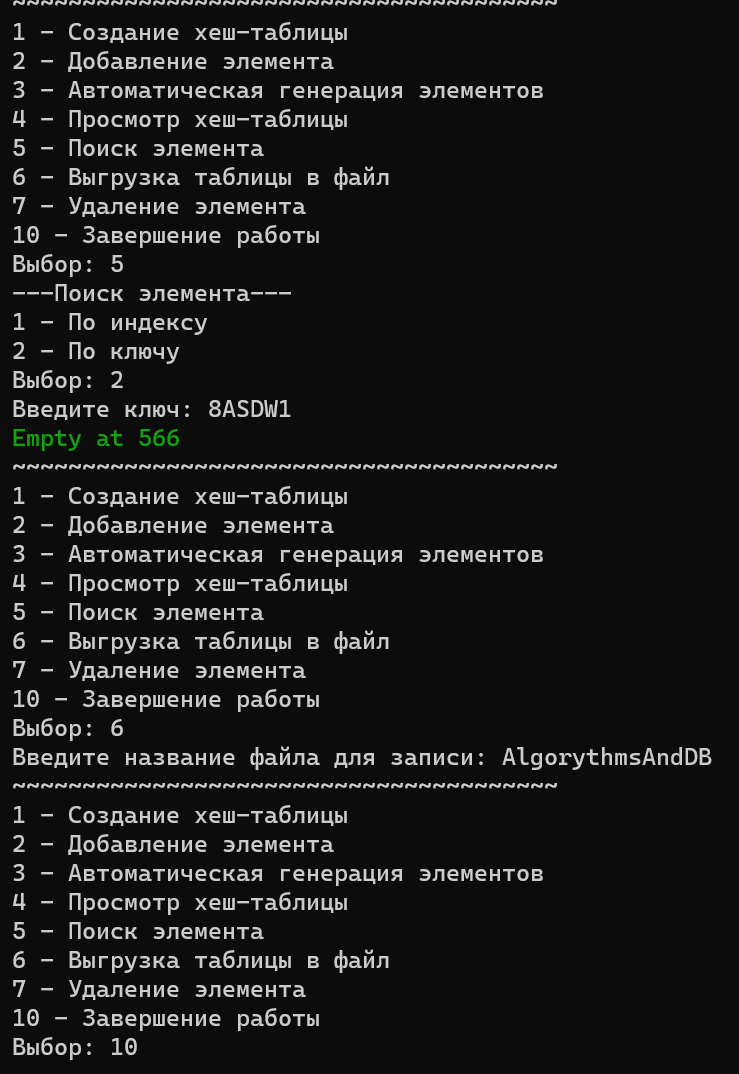
1. **Листинг программы**

|  |
| --- |
| #include <iostream>  #include <random>  #include <vector>  #include <string>  #include <fstream>  using namespace std;  int NumOfSegments = 2000;  int intInput(string welcome) {  int a;  cout << welcome;  cin >> a;  while (!cin) {  cout << "\x1B[31mНеверный ввод. (Ожидается число)\033[0m\n";  cin.clear();  cin.ignore(999, '\n');  cout << "Введите заново: ";  cin >> a;  }  return a;  }  int intInputInBounds(string welcome, int left, int right) {  int a = intInput(welcome);  while (a < left || a > right) {  cout << "\x1B[31mНеверный ввод.\033[0m Значение должно быть на отрезке [" << left << ", " << right << "]\n";  cout << "Повторите попытку: ";  a = intInput(welcome);  }  return a;  }  bool checkKey(string key) {  if (key.size() != 6)  return false;  if (key[0] < 48 || key[0] > 57)  return false;  if (key[1] < 65 || key[1] > 90)  return false;  if (key[2] < 65 || key[2] > 90)  return false;  if (key[3] < 65 || key[3] > 90)  return false;  if (key[4] < 65 || key[4] > 90)  return false;  if (key[5] < 48 || key[5] > 57)  return false;  return true;  }  string enterKey() {  string inp;  cout << "Введите ключ: ";  cin >> inp;  while (!checkKey(inp)) {  cout << "\x1B[31mНеверно введен ключ.\033[0m Формат цББББц.\nПовторите попытку ввода: ";  cin >> inp;  }  return inp;  }  string enterString(string welcome) {  string inp = "";  char letter;  cout << welcome;  cin.ignore(999, '\n');  cin.clear();  getline(cin, inp);  while (inp.size() == 0) {  cout << "\x1B[31mНеверный ввод\033[0m\nПовторите попытку: ";  getline(cin, inp);  }  return inp;  }  class HashTable {  private:  class Node {  public:  string value;  bool state;  Node() {  value = "";  state = false;  }  Node(string value) {  this->value = value;  state = true;  }  };  Node\* nodeArr = new Node[NumOfSegments];  ~HashTable() {  delete nodeArr;  }  int hashCode(string& key) {  int hashNum = 0;  for (int j = 0; j < 6; j++)  hashNum += key[j] \* ((j + 1) \* 3) + key[j] \* j \* j \* j;  hashNum = hashNum % NumOfSegments;  return hashNum;  }  public:  void put(string key, string value) {  int hash = hashCode(key);  int firstVal = hash;  bool firstTime = true;  while (nodeArr[hash].state) {  if (hash == firstVal && !firstTime) {  cout << "Значение по ключу '" << key << "' не удалось вставить: таблица заполнена\n";  return;  }  hash = (hash + 1) % NumOfSegments;  firstTime = false;  }  nodeArr[hash].value = value;  nodeArr[hash].state = true;  }  void show() {  for (int i = 0; i < NumOfSegments; i++)  if (nodeArr[i].state)  cout << nodeArr[i].value << endl;  else  if (nodeArr[i].value == "")  cout << "Empty\n";  else  cout << "Deleted\n";  }  string get(int index) {  if (nodeArr[index].state)  return nodeArr[index].value + " at " + to\_string(index);  if (nodeArr[index].value == "")  return "Empty at " + to\_string(index);  return "Deleted at " + to\_string(index);  }  string get(string key) {  int hash = hashCode(key);  if (nodeArr[hash].state)  return nodeArr[hash].value + " at " + to\_string(hash);  if (nodeArr[hash].value == "")  return "Empty at " + to\_string(hash);  return "Deleted at " + to\_string(hash);  }  void writeToFile(string fileName) {  ofstream ofs;  fileName += ".txt";  ofs.open(fileName);  for (int i = 0; i < NumOfSegments; i++) {  if (nodeArr[i].state)  ofs << nodeArr[i].value << " ";  else  if (nodeArr[i].value == "")  ofs << "Empty ";  else  ofs << "Deleted ";  }  ofs.close();  }  void remove(string key) {  int hash = hashCode(key);  if (!nodeArr[hash].state) {  cout << "По ключу '" << key << "' в таблице не было.\n";  return;  }  nodeArr[hash].state = false;  }  void remove(int index) {  if (!nodeArr[index].state) {  cout << "По индексу '" << index << "' в таблице не было значения.\n";  return;  }  nodeArr[index].state = false;  }  };  void generateKeys(vector<string>& keysArray, int count) {  random\_device rd;  mt19937 mt(rd());  uniform\_int\_distribution<int> uniLetters(65, 90);  uniform\_int\_distribution<int> uniNumbers(0, 9);  while (count != 0) {  keysArray.push\_back(to\_string(uniNumbers(mt)) +  (char)uniLetters(mt) +  (char)uniLetters(mt) +  (char)uniLetters(mt) +  (char)uniLetters(mt) +  to\_string(uniNumbers(mt)));  count--;  }  }  int main()  {  setlocale(LC\_ALL, "RUS");  HashTable\* ht = nullptr;  bool flag1 = true;  while (flag1) {  cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n";  cout << "1 - Создание хеш-таблицы\n";  cout << "2 - Добавление элемента\n";  cout << "3 - Автоматическая генерация элементов\n";  cout << "4 - Просмотр хеш-таблицы\n";  cout << "5 - Поиск элемента\n";  cout << "6 - Выгрузка таблицы в файл\n";  cout << "7 - Удаление элемента\n";  cout << "10 - Завершение работы\n";  switch (intInput("Выбор: ")) {  case 1:  if (ht != nullptr)  cout << "\x1B[33mХеш-таблица уже создана\033[0m\n";  else {  ht = new HashTable();  cout << "\x1B[32mХеш-таблица успешно создана\033[0m\n";  }  break;  case 2:  if (ht == nullptr)  cout << "\x1B[31mСначала создайте таблицу\033[0m\n";  else {  ht->put(enterKey(), enterString("Введите данные: "));  }  break;  case 3:  if (ht == nullptr)  cout << "\x1B[31mСначала создайте таблицу\033[0m\n";  else {  vector<string> generatedKeys;  generateKeys(generatedKeys, intInput("Введите количество генерируемых элементов: "));  for (int i = 0; i < generatedKeys.size(); i++) {  ht->put(generatedKeys[i], generatedKeys[i] + " was added");  }  }  break;  case 4:  if (ht == nullptr)  cout << "\x1B[31mСначала создайте таблицу\033[0m\n";  else {  ht->show();  }  break;  case 5:  if (ht == nullptr)  cout << "\x1B[31mСначала создайте таблицу\033[0m\n";  else {  bool flag2 = true;  while (flag2) {  cout << "---Поиск элемента---\n";  cout << "1 - По индексу\n";  cout << "2 - По ключу\n";  string foundElement;  switch (intInput("Выбор: ")) {  case 1:  foundElement = ht->get(intInputInBounds("Введите индекс: ", 0, NumOfSegments - 1));  cout << "\x1B[32m" << foundElement << "\033[0m\n";  flag2 = false;  break;  case 2:  foundElement = ht->get(enterKey());  cout << "\x1B[32m" << foundElement << "\033[0m\n";  flag2 = false;  break;  default:  cout << "\x1B[31mНеверный выбор пункта меню\033[0m\nПовторите попытку\n";  break;  }  }  }  break;  case 6:  if (ht == nullptr)  cout << "\x1B[31mСначала создайте таблицу\033[0m\n";  else {  ht->writeToFile(enterString("Введите название файла для записи: "));  }  break;  case 7:  if (ht == nullptr)  cout << "\x1B[31mСначала создайте таблицу\033[0m\n";  else {  bool flag3 = true;  while (flag3) {  cout << "---Удаление элемента---\n";  cout << "1 - По индексу\n";  cout << "2 - По ключу\n";  switch (intInput("Выбор: ")) {  case 1:  ht->remove(intInputInBounds("Введите индекс: ", 0, NumOfSegments - 1));  flag3 = false;  break;  case 2:  ht->remove(enterKey());  flag3 = false;  break;  default:  cout << "\x1B[31mНеверный выбор пункта меню\033[0m\nПовторите попытку\n";  break;  }  }  }  break;  case 10:  flag1 = false;  break;  default:  cout << "\x1B[31mНеверный выбор пункта меню\033[0m\nПовторите попытку\n";  break;  }  }  } |

1. **Результаты работы**







1. **Выводы**

Изучены методы хеширования данных и получены практические навыки реализации хеш-таблиц. На практике реализована хеш-таблица с методом линейного опробования.