МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ИНСТИТУТ НЕПРЕРЫВНОГО И ДИСТАНЦИОННОГО ОБРАЗОВАНИЯ

|  |
| --- |
| КАФЕДРА компьютерных технологий и программной инженерии |

ОЦЕНКА

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| «ХЕШИРОВАНИЕ ДАННЫХ» |
| по дисциплине: Структуры и алгоритмы обработки данных |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | Z9431 |  |  |  | Андреев Д.И. |
|  | номер группы |  | подпись, дата |  | инициалы, фамилия |
| Студенческий билет № | 2019/3781 | |  |  |  |

Санкт-Петербург 2020

Оглавление

[1. Цель работы 4](#_Toc55577294)

[2. Вариант задания 4](#_Toc55577295)

[3. Описание хеш-функции 4](#_Toc55577296)

[4. Результат анализа хеш-функции 5](#_Toc55577297)

[5. Листинг программы 7](#_Toc55577305)

[6. Выводы 20](#_Toc55577306)

1. Цель работы

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

1. Вариант задания

Вариант 7.

Составить хеш-функцию и проанализировать ее. 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 – квадратичное опробование.

Формат ключа «БцццББ».

Количество сегментов – 3000.

1. Описание хеш-функции

В ходе анализа задания была выбрана хеш-функция, основанная на сложении кодов символов ключа. Данная функция дает хорошее распределение при относительно небольшом количестве элементов хеш-таблицы и использовании символьного ключа:

unsigned int seg = 0;

for (int i = 0; i < keyLength; ++i)

seg += static\_cast<unsigned int>(key[i));

Область определения данной функции [339; 441]. Так как область определения хеш-функции должна совпадать с количеством сегментов, необходимо привести хеш-функцию к виду, когда ее максимальное значение давало бы максимальный номер сегмента таблицы, равный 3000, а минимальное значение равнялось 0, т.е. включить в функцию следующий код:

seg = ((seg - 339) \* m\_multiCoef);

где m\_multiCoef – коэффициент равный количеству сегментов таблицы разделенному на максимальное значение хеш-функции после вычитания (т.е. 3000 / (441 – 339)).

После выполнения данного преобразования хеш-функция выглядит следующим образом:

unsigned int seg = 0;

for (int i = 0; i < keyLength; ++i)

seg += static\_cast<unsigned int>(key[i));

seg = ((seg - 339) \* m\_multiCoef);

1. Результат анализа хеш-функции

В результате анализа был написан генератор случайных ключей заданного формата, формирующий 9000 случайных ключей. Анализ полученной функции после вставки 9000 ключей в хеш-таблицу показал, что распределение коллизий по хеш-таблице неравномерно (рас. 1).

Рис. 1 – распределение коллизий по таблице до модификации хеш-функции.

В связи с этим функция была модифицирована до следующего вида: искомый сегмент получается, как сумма квадратов кодов символов ключа. Так как при использовании данной функции можно получить номер сегмента превышающий размер таблицы, номер сегмента берется как остаток от деления полученного значения сегмента на общее количество сегментов:

unsigned int seg = 0;

//для каждого символа ключа

for (int i = 0; key[i] != '\0'; ++i)

//прибавить код символа в 3 степени к общей сумме

seg += static\_cast<unsigned int>(key[i] \* key[i]);

seg = ((seg - 339) \* m\_multiCoef) % (m\_capacity);

где m\_capacity – количество сегментов хеш-таблицы

На диаграмме ниже (рис. 2) приведено распределение коллизий по хеш-таблице при использовании модифицированной функции. Из диаграммы видно, что данная функция имеет более равномерное распределение.

Рис. 2 – распределение коллизий по таблице после модификации хеш-функции.

После создания хеш-таблицы при разрешении переполнения данная функция показывали большой расход памяти. При вставке 9000 элементов в таблицу из 3000 сегментов таблица расширялась до 34171 элемента. Было принято решение модифицировать хеш-функцию до сумы кубов кодов элементов ключа:

unsigned int seg = 0;

//для каждого символа ключа

for (int i = 0; key[i] != '\0'; ++i)

//прибавить код символа в 3 степени к общей сумме

seg += static\_cast<unsigned int>(std::pow(key[i], 3));

seg = ((seg - 339) \* m\_multiCoef) % (m\_capacity);

Данная хеш-функция показала такое же распределение коллизий по сегментам таблицы (рис. 3), но при этом при тех же тестах таблица расширялась до 22781 элементов.

Рис. 3 – распределение коллизий по таблице после повторной модификации хеш-функции.

1. Листинг программы

Файл main.cpp.

Содержит код пользовательского интерфейса программы, а также реализацию проверки ключей на соответствие и реализацию генератора случайных ключей

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <ctime>

#include "Hash.h"

using namespace std;

const int quantOfKeys = 9000;

short getCommand();

void saveKeysGenToFile();

void saveKeysGenToTable(Hash& h);

void \_\_keysGen(char(\*keysContainer)[7], const int size);

void createTableFromFile(Hash& h);

void addElementToTable(Hash& h);

void deleteElementByKey(Hash& h);

void deleteElementBySegment(Hash& h);

void clearTable(Hash& h);

void showTable(const Hash& h);

void findElementByKey(const Hash& h);

void findElementBySegment(const Hash& h);

void getTableAsFile(const Hash& h);

void showHint();

bool \_\_getAndCheckKey(char\* key);

void getTestDataFile(const Hash& h);

int main()

{

setlocale(LC\_ALL, "Russian");

cout << "Лабораторная №4.\nХеширование данных.\nВариант " << (81 % 25) + 1 << "\n\n";

short command = -1;

Hash hashTable;

do {

command = getCommand();

switch (command)

{

case 0:

createTableFromFile(hashTable);

break;

case 1:

saveKeysGenToFile();

break;

case 2:

saveKeysGenToTable(hashTable);

break;

case 3:

addElementToTable(hashTable);

break;

case 4:

deleteElementByKey(hashTable);

break;

case 5:

deleteElementBySegment(hashTable);

break;

case 6:

clearTable(hashTable);

break;

case 7:

findElementByKey(hashTable);

break;

case 8:

findElementBySegment(hashTable);

break;

case 9:

getTableAsFile(hashTable);

break;

case 10:

showTable(hashTable);

break;

case 11:

showHint();

break;

case 12:

break;

case 13:

getTestDataFile(hashTable);

break;

case 14: {

char(\*keys)[7] = new char[quantOfKeys][7];

hashTable.keys(keys);

for (int i = 0; i < quantOfKeys; ++i) {

cout << "ключ: " << keys[i] << " значение: " << hashTable[keys[i]] << '\n' ;

}

break;

}

default:

cout << "Неизвестная команда\n";

break;

}

} while (command != 12);

return 0;

}

short getCommand()

{

short com;

cout << "Введите команду для продолжения (11 - показать все команды): ";

cin >> com;

while (!cin) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Введите коректной значение (12 для выхода):";

cin >> com;

}

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return com;

}

void saveKeysGenToFile()

{

const int fileLength = 100;

cout << "Введите полный путь к файлу (до " << fileLength << " символов): ";

char filePath[fileLength];

int i = 0;

for (i; i < fileLength && cin.peek() != '\n'; ++i) {

cin >> filePath[i];

}

filePath[i] = '\0';

ofstream ofs;

ofs.open(filePath);

if (!ofs.is\_open()) {

cout << "Невозможно открыть файл " << filePath << '\n';

ofs.clear();

return;

}

char(\*keys)[7] = new char[quantOfKeys][7];

\_\_keysGen(keys, quantOfKeys);

for (int i = 0; i < quantOfKeys; ++i) {

ofs << keys[i] << '\n';

}

ofs.close();

delete[] keys;

}

void saveKeysGenToTable(Hash& h)

{

char(\*keys)[7] = new char[quantOfKeys][7];

\_\_keysGen(keys, quantOfKeys);

srand(clock());

int badKeyOnGen = 0;

for (int i = 0; i < quantOfKeys; ++i) {

if (h.hasKey(keys[i])) {

++badKeyOnGen;

continue;

}

h.insert(keys[i], rand() % 1000);

}

cout << "Сгенерированно " << quantOfKeys - badKeyOnGen << " уникальных ключей\n";

delete[] keys;

}

void \_\_keysGen(char(\*keysContainer)[7], const int size)

{

srand(clock());

for (int i = 0; i < size; ++i) {

for (int j = 0; j < 7; ++j) {

switch (j)

{

case 0:

case 4:

case 5:

keysContainer[i][j] = static\_cast<char>(rand() % 25 + 65);

break;

case 1:

case 2:

case 3:

keysContainer[i][j] = static\_cast<char>(rand() % 10 + 48);

break;

case 6:

keysContainer[i][j] = '\0';

default:

break;

}

}

}

}

void createTableFromFile(Hash& h)

{

const int fileLength = 100;

cout << "Введите полный путь к файлу (до " << fileLength << " символов): ";

char filePath[fileLength];

int i = 0;

for (i; i < fileLength && cin.peek() != '\n'; ++i) {

cin >> filePath[i];

}

filePath[i] = '\0';

ifstream ifs;

ifs.open(filePath);

if (!ifs.is\_open()) {

cout << "Невозможно открыть файл " << filePath << '\n';

ifs.clear();

return;

}

char key[7]{};

srand(clock());

while (!ifs.eof())

{

ifs >> key;

if (key[0] == '\0')

continue;

h.insert(key, rand()%1000);

}

ifs.close();

}

void addElementToTable(Hash& h)

{

cout << "Введите пару ключ - значение через пробел: ";

char key[7];

int val = 0;

if (!\_\_getAndCheckKey(key)) {

cout << "Неверный формат ключа\n";

std::cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return;

}

if (cin.peek() != ' ') {

cout << "Неверный формат ключа\n";

std::cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return;

}

std::cin >> val;

if (!std::cin) {

cout << "Неверный формат ключа\n";

std::cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return;

}

h.insert(key, val);

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

}

void deleteElementByKey(Hash& h)

{

cout << "Введите ключ: ";

char key[7]{};

if (!\_\_getAndCheckKey(key)) {

cout << "Неверный формат ключа\n";

std::cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return;

}

if(!h.erase(key)) cout << "Не удалось удалить элемент\n";

}

void deleteElementBySegment(Hash& h)

{

cout << "Введите номер сегмента: ";

int seg = 0;

cin >> seg;

if (!std::cin) {

cout << "Ошибка ввода\n";

std::cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return;

}

if (!h.erase(seg)) cout << "Не удалось удалить элемент\n";

}

void clearTable(Hash& h)

{

h.clear();

}

void showTable(const Hash& h)

{

if (h.size() == 0)

cout << "Таблица пуста\n";

char(\*keys)[7] = new char[h.size()][7]{};

h.keys(keys);

for (int i = 0; i < h.size(); ++i)

cout << "Ключ: " << keys[i] << " значение: " << h[keys[i]] << '\n';

delete[] keys;

}

void findElementByKey(const Hash& h)

{

cout << "Введите ключ: ";

char key[7]{};

if (!\_\_getAndCheckKey(key)) {

cout << "Неверный формат ключа\n";

std::cin.clear();

std::cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return;

}

cout << "Введенный ключ: " << key << " значение: " << h[key] << '\n';

}

void findElementBySegment(const Hash& h)

{

cout << "Введите номер сегмента: ";

int seg = 0;

cin >> seg;

if (!std::cin) {

cout << "Ошибка ввода\n";

std::cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

return;

}

cout << "Ключ: " << h.key(seg) << " значение: " << h[seg] << '\n';

}

void getTableAsFile(const Hash& h)

{

const int fileLength = 100;

cout << "Введите полный путь к файлу (до " << fileLength << " символов): ";

char filePath[fileLength];

int i = 0;

for (i; i < fileLength && cin.peek() != '\n'; ++i) {

cin >> filePath[i];

}

filePath[i] = '\0';

ofstream ofs;

ofs.open(filePath);

if (!ofs.is\_open()) {

cout << "Невозможно открыть файл " << filePath << '\n';

ofs.clear();

return;

}

for (int i = 0; i < h.capacity(); ++i)

ofs << h.key(i) << " " << h[i] << '\n';

ofs.close();

}

void showHint()

{

cout << "0 - Создать хеш-таблицу из файла ключей со случайными числами\n";

cout << "1 - Сгенерировать случайные ключи в файл\n";

cout << "2 - Сгенерировать случайные ключи в хеш-таблицу со случайными числами\n";

cout << "3 - Добавить новый элемент в хеш-таблицу\n";

cout << "4 - Удалить элемент по ключу\n";

cout << "5 - Удалить элемент по сегменту\n";

cout << "6 - Очистить хеш-таблицу\n";

cout << "7 - Найти элемент в хеш-таблице по ключу\n";

cout << "8 - Найти элемент в хеш-таблицу по сегменту\n";

cout << "9 - Выгрузить хеш-таблицу в файл\n";

cout << "10 - Вывести хеш-таблицу на экран\n";

cout << "11 - Показать все команды\n";

cout << "12 - Выход\n";

cout << "13 - Выгрузить тестовые данные в файл\n";

}

bool \_\_getAndCheckKey(char\* key)

{

bool fineFlag = true;

for (int i = 0; i < 6; ++i) {

std::cin >> key[i];

switch (i)

{

case 0:

case 4:

case 5:

if (static\_cast<int>(key[i]) < 65 ||

static\_cast<int>(key[i]) > 90) {

fineFlag = false;

}

break;

case 1:

case 2:

case 3:

if (static\_cast<int>(key[i]) < 48 ||

static\_cast<int>(key[i]) > 57) {

fineFlag = false;

}

break;

default:

break;

}

}

key[6] = '\0';

return fineFlag;

}

void getTestDataFile(const Hash& h)

{

const int fileLength = 100;

cout << "Введите полный путь к файлу (до " << fileLength << " символов): ";

char filePath[fileLength];

int i = 0;

for (i; i < fileLength && cin.peek() != '\n'; ++i) {

cin >> filePath[i];

}

filePath[i] = '\0';

if (!h.testDataFile(filePath)) {

cout << "Невозможно открыть файл " << filePath << '\n';

return;

}

}

Файл Hash.h.

Содержит определение класса хеш-таблицы.

#ifndef HASH\_H

#define HASH\_H

#include <cstdlib>

#include <iostream>

#include <fstream>

const char deletedSegment[7] = "\*\*\*\*\*\0";

const char emptySegment[7] = " \0";

class Hash

{

public:

Hash()

: m\_elements(new Data[m\_capacity])

, m\_size(0)

, m\_capacity(3000)

, m\_multiCoef(m\_capacity / 102) {}

~Hash() { delete[] m\_elements; }

//Вставка значение value по ключу key

void insert(const char\* key, const int& value);

//Удаление значения по ключу key

bool erase(const char\* key);

//Перегруженный метод. Удаление по номеру сегмента

bool erase(const unsigned int seg);

//Полная очистка табилцы

void clear();

//Получения значения по ключу

int operator[](const char\* key) const;

//Получение значения по сегменту

int operator[](const unsigned int seg) const;

//Выгружает все ключи в переданный массив

void keys(char(\*keysStorage)[7]) const;

//Получение ключа по номеру сегмента

char\* key(const unsigned int seg) const { return m\_elements[seg].m\_key; }

//Содержит ли таблица переданный ключ

bool hasKey(const char\* key) const;

//Значащий размер таблицы

int size() const { return m\_size; }

//Фактичекй размер таблицы

int capacity() const { return m\_capacity; }

bool testDataFile(const char \*fileName) const;

private:

//Увеличение фактического размера таблицы

void resize(unsigned int size = 0);

//Хеш-функция

unsigned int hashFunction(const char\* key) const;

//Функция квадратичного опробования

void quadTesting(int tryNum, unsigned int& seg) const { seg += 1 \* tryNum + 1 \* tryNum \* tryNum; }

//Структура для хранения данных элемента хеш-таблицы

struct Data {

//стандартный конструктор

Data() : m\_val(0), m\_key(new char[7]), m\_segCalcCount(0) { std::strcpy(m\_key, emptySegment); }

//Чтобы небыло утечек при удалении

~Data() { delete[] m\_key; }

//Поля значения и ключа

int m\_val;

char\* m\_key;

unsigned int m\_segCalcCount;

};

//Значащий размер таблицы

unsigned int m\_size;

//Фактический размер таблицы

unsigned int m\_capacity;

//Множитель размера таблицы

unsigned int m\_multiCoef;

//Массив элементов

Data\* m\_elements;

};

#endif // HASH\_H

Файл Hash.cpp

Содержит реализацию методов хеш-таблицы, в т.ч. реализацию хеш-функции, алгоритмов вставки, удаления и поиска элементов в таблице и методов разрешения коллизий и переполнения таблицы.

#include "Hash.h"

#include <cmath>

void Hash::insert(const char\* key, const int& value)

{

//Вызвать хеш-функцию

unsigned int seg = hashFunction(key);

//Пока полученный сегмент не за пределами таблицы

for (int i = 0; seg < m\_capacity; ++i) {

//Подсчет количества обращений к сегменту

++m\_elements[seg].m\_segCalcCount;

//Если элемент имеет такой же ключ как передаваемый в функцию

if (!std::strcmp(m\_elements[seg].m\_key, key)) {

//Изменить значение по данному адресу

m\_elements[seg].m\_val = value;

return;

}

//Если элемент по полученному адресу пуст или удален

else if (!std::strcmp(m\_elements[seg].m\_key, deletedSegment) ||

!std::strcmp(m\_elements[seg].m\_key, emptySegment)) {

//Вставить новый элемент в таблицу

m\_elements[seg].m\_val = value;

std::strcpy(m\_elements[seg].m\_key, key);

++m\_size;

return;

}

else

//Вызвать функцию квадратичного опробования для получения нового адреса

quadTesting(i, seg);

}

//Полученный сегмент за пределами таблицы

//Рекурсивно изменять фактический размер таблицы, пока ключ не будет вставлен в таблицу

//Это гарантирует, что ключ будет вставлен в таблицу

resize();

insert(key, value);

}

bool Hash::erase(const char\* key)

{

//Вызвать хеш-функцию

unsigned int seg = hashFunction(key);

//Пока полученный сегмент не за пределами таблицы

for (int i = 0; seg < m\_capacity; ++i) {

//Если ключ по полученному сегменту равен ключу переданному в функцию

if (!std::strcmp(key, m\_elements[seg].m\_key)) {

//пометить его как удаленный

std::strcpy(m\_elements[seg].m\_key, deletedSegment);

m\_elements[seg].m\_segCalcCount = 0;

--m\_size;

return true;

}

else

//Вызвать функцию квадратичного опробования

quadTesting(i, seg);

}

//Полученный сегмент за пределами таблицы.

//Данного элемента нет в таблице

return false;

}

bool Hash::erase(const unsigned int seg)

{

//Если сегмент за пределами таблицы - операция имеет неопределенное поведение

if (seg >= m\_capacity) {

std::cout << "Попытка обращения к несуществующему сегменту\n";

return false;

}

//Если элемент по сегменту уже удален - операция не имеет смысла

else if (!std::strcmp(m\_elements[seg].m\_key, deletedSegment) ||

!std::strcmp(m\_elements[seg].m\_key, emptySegment)) {

return false;

}

//Иначе, помечаем элемент по сегменту как удаленный

return erase(m\_elements[seg].m\_key);

}

void Hash::clear()

{

//Как в erase только для всех сегментов таблицы

for (unsigned int i = 0; i < m\_capacity; ++i) {

if (std::strcmp(m\_elements[i].m\_key, deletedSegment) &&

std::strcmp(m\_elements[i].m\_key, emptySegment)) {

std::strcpy(m\_elements[i].m\_key, deletedSegment);

m\_elements[i].m\_segCalcCount = 0;

--m\_size;

}

}

//Вернуть размер таблицы в исходное состояние

//Иначе после каждой очистки insert начнет раздувать размер таблицы

m\_capacity = 3000;

delete[] m\_elements;

m\_elements = new Data[m\_capacity];

m\_multiCoef = static\_cast<unsigned int>(m\_capacity / 102);

}

int Hash::operator[](const char\* key) const

{

//Вызвать хеш-функцию

unsigned int seg = hashFunction(key);

//Пока полученный сегмент не за пределами таблицы

for (int i = 0; seg < m\_capacity; ++i) {

//Если по сегменту лежит тот же ключ, что и передаваемый в функцию

if (!std::strcmp(m\_elements[seg].m\_key, key)) {

//Вернуть значение

return m\_elements[seg].m\_val;

}

//Если элемент по полученному сегменту пуст или удален

else {

//Вызвать функцию квадратичного опробования для получения нового адреса

quadTesting(i, seg);

continue;

}

}

//Полученный сегмент за пределами таблицы.

//Такого ключа в таблице нет

//! Можно сделать out of range, но такой задачи нет

std::cout << "Ключ " << key << " не найден\n";

return m\_elements[0].m\_val;

}

int Hash::operator[](const unsigned int seg) const

{

//Если сегмент за пределами таблицы - операция имеет неопределенное поведение

//! Можно сделать out of range, но такой задачи нет

if (seg > m\_capacity) {

std::cout << "обращение к несуществующему сегменту хеш-таблицы\n";

return m\_elements[0].m\_val;

}

return m\_elements[seg].m\_val;

}

void Hash::keys(char(\*keysStorage)[7]) const

{

//Копирует все ключи, не помеченные как удаленные или пустые в keysStorage

for (unsigned int i = 0, j = 0; i < m\_capacity; ++i) {

if (std::strcmp(m\_elements[i].m\_key, emptySegment) &&

std::strcmp(m\_elements[i].m\_key, deletedSegment)) {

std::strcpy(keysStorage[j], m\_elements[i].m\_key);

++j;

}

}

}

bool Hash::hasKey(const char\* key) const

{

for (unsigned int i = 0; i < m\_capacity; ++i)

if (!std::strcmp(key, m\_elements[i].m\_key))

return true;

return false;

}

bool Hash::testDataFile(const char\* fileName) const

{

//Записывает в файл количество обращений к каждому сегменту

std::ofstream ofs;

ofs.open(fileName);

if (!ofs.is\_open()) {

ofs.clear();

return false;

}

for (unsigned int i = 0; i < m\_capacity; ++i)

ofs << m\_elements[i].m\_segCalcCount << '\n';

ofs.close();

return true;

}

void Hash::resize(unsigned int size)

{

unsigned int newCapacity = static\_cast<unsigned int>(m\_capacity \* 1.5);

//Создается временное хранилище элементов таблицы

//все ее элементы копируются в это хранилище

Data\* temp = new Data[newCapacity];

for (unsigned int i = 0; i < m\_capacity; ++i) {

std::strcpy(temp[i].m\_key, m\_elements[i].m\_key);

temp[i].m\_val = m\_elements[i].m\_val;

}

//Удаляется массив таблицы и создается новый нового размера

delete[] m\_elements;

m\_elements = new Data[newCapacity];

//обычная вставка в получившийся массив

//из временного хранилища. Иначе потеряются ключи

m\_capacity = newCapacity;

m\_size = 0;

for (unsigned int i = 0; i < newCapacity; ++i) {

if (std::strcmp(temp[i].m\_key, deletedSegment) &&

std::strcmp(temp[i].m\_key, emptySegment))

insert(temp[i].m\_key, temp[i].m\_val);

}

delete[] temp;

}

unsigned int Hash::hashFunction(const char\* key) const

{

unsigned int seg = 0;

//для каждого символа ключа

for (int i = 0; key[i] != '\0'; ++i)

//прибавить код символа в 3 степени к общей сумме

seg += static\_cast<unsigned int>(std::pow(key[i], 3));

return ((seg - 339) \* m\_multiCoef) % (m\_capacity);

}

Описание основных алгоритмов:

Алгоритм вставки элемента в хеш-таблицу:

//Вызвать хеш-функцию

unsigned int seg = hashFunction(key);

//Пока полученный сегмент не за пределами таблицы

for (int i = 0; seg < m\_capacity; ++i) {

//Подсчет количества обращений к сегменту

++m\_elements[seg].m\_segCalcCount;

//Если элемент имеет такой же ключ как передаваемый в функцию

if (!std::strcmp(m\_elements[seg].m\_key, key)) {

//Изменить значение по данному адресу

m\_elements[seg].m\_val = value;

return;

}

//Если элемент по полученному адресу пуст или удален

else if (!std::strcmp(m\_elements[seg].m\_key, deletedSegment) ||

!std::strcmp(m\_elements[seg].m\_key, emptySegment)) {

//Вставить новый элемент в таблицу

m\_elements[seg].m\_val = value;

std::strcpy(m\_elements[seg].m\_key, key);

++m\_size;

return;

}

else

//Вызвать функцию квадратичного опробования для получения нового адреса

quadTesting(i, seg);

}

//Полученный сегмент за пределами таблицы

//Рекурсивно изменять фактический размер таблицы, пока ключ не будет вставлен в таблицу

//Это гарантирует, что ключ будет вставлен в таблицу

resize();

insert(key, value);

}

A1 [хеш-функция] Вызвать хеш-функцию для получения сегмента

A2 [сравнение ключей] Если ключ по полученному сегменту равен ключу, переданному в функцию, вставить значение, переданное в функцию, по данному сегменту. Завершить выполнение алгоритма. Иначе перейти к шагу A3

A3 [поиск пустых сегментов] Если полученный сегмент помечен как пустой или удаленный, создать новый элемент по данному сегменту хеш-таблицы с переданными в функцию ключом и значением. Завершить выполнение алгоритма. Иначе перейти к шагу A4

A4 [элемент не найден] По данному адресу невозможно вставить данное значение – вызвать функцию линейного опробования. Если полученный сегмент меньше общего количества сегментов перейти к шагу A2. Иначе перейти к шагу A5

A5 [достигнут конец таблицы] Увеличить размер таблицы. Перейти к шагу A1

Алгоритм удаления элемента из хеш-таблицы по ключу:

bool Hash::erase(const char\* key)

{

//Вызвать хеш-функцию

unsigned int seg = hashFunction(key);

//Пока полученный сегмент не за пределами таблицы

for (int i = 0; seg < m\_capacity; ++i) {

//Если ключ по полученному сегменту равен ключу переданному в функцию

if (!std::strcmp(key, m\_elements[seg].m\_key)) {

//пометить его как удаленный

std::strcpy(m\_elements[seg].m\_key, deletedSegment);

m\_elements[seg].m\_segCalcCount = 0;

--m\_size;

return true;

}

else

//Вызвать функцию квадратичного опробования

quadTesting(i, seg);

}

//Полученный сегмент за пределами таблицы.

//Данного элемента нет в таблице

return false;

}

B1 [хеш-функция] Вызвать хеш-функцию для получения сегмента

B2 [сравнение ключей] Если ключ по полученному сегменту равен ключу, переданному в функцию, пометить элемент как удаленный. Завершить выполнение алгоритма. Иначе перейти к шагу B3

B3 [элемент не найден] По данному сегменту элемента с таким ключом нет – вызвать функцию линейного опробования. Если полученный сегмент меньше общего количества сегментов перейти к шагу С2. Иначе перейти к шагу С4

B4 [достигнут конец таблицы] Элемента с таким ключом в таблице нет. Завершить выполнение алгоритма

Алгоритм поиска элемента в хеш-таблице по ключу:

int Hash::operator[](const char\* key) const

{

//Вызвать хеш-функцию

unsigned int seg = hashFunction(key);

//Пока полученный сегмент не за пределами таблицы

for (int i = 0; seg < m\_capacity; ++i) {

//Если по сегменту лежит тот же ключ, что и передаваемый в функцию

if (!std::strcmp(m\_elements[seg].m\_key, key)) {

//Вернуть значение

return m\_elements[seg].m\_val;

}

//Если элемент по полученному сегменту пуст или удален

else {

//Вызвать функцию квадратичного опробования для получения нового адреса

quadTesting(i, seg);

continue;

}

}

//Полученный сегмент за пределами таблицы.

//Такого ключа в таблице нет

//! Можно сделать out of range, но такой задачи нет

std::cout << "Ключ " << key << " не найден\n";

return m\_elements[0].m\_val;

}

С1 [хеш-функция] Вызвать хеш-функцию для получения сегмента

С2 [сравнение ключей] Если ключ по полученному сегменту равен ключу, переданному в функцию, элемент найден. Завершить выполнение алгоритма. Иначе перейти к шагу С3

С3 [элемент не найден] По данному сегменту элемента с таким ключом нет – вызвать функцию линейного опробования. Если полученный сегмент меньше общего количества сегментов перейти к шагу С2. Иначе перейти к шагу С4

С4 [достигнут конец таблицы] Элемента с таким ключом в таблице нет. Завершить выполнение алгоритма

1. Выводы

В рамках лабораторной работы была разработана и проанализирована хеш-функция, на основе которой, в соответствии с вариантом реализована хеш-таблица, реализованы алгоритмы вставки, удаления и поиска в хеш-таблице, а также методы разрешения коллизий и переполнения хеш-таблицы.

Данная хеш-функция дает хорошее распределение коллизий по хеш-таблице. Метод разрешения переполнения хеш-таблицы гарантирует, что любой элемент будет вставлен в хеш-таблицу и при этом при увеличении хеш-таблицы не происходит большой траты ресурсов (при вставке 9000 случайных ключей в хеш-таблицу с 3000 сегментами результирующий размер таблицы на тестах равен 15187).

Также тесты показали, что изменение размера таблицы происходит при заполнении таблицы примерно на 50%.