ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | М. А. Мурашова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| «ХЕШИРОВАНИЕ ДАННЫХ» |
| по курсу: СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134к |  |  |  | Костяков Н.А. |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

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

**1.2 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.

**Вариант 17**



**Листинг**

#include <iostream>

#include <fstream>

#include <string>

int hash(std::string key) //пример лкюча ццББцц

{

int value = (key[0] \* key[0] + key[1] \* key[1] + key[2] \* key[2] + key[3] \* key[3] + key[4] \* key[4] + key[5] \* key[5]) % 1500;

return value;

}

bool check\_key(std::string key)

{

if ((key[0]>=48 and key[0]<=57) and (key[1] >= 48 and key[1] )and (key[2] >= 65 and key[2]<=90) and (key[3] >= 65 and key[3] <= 90) and (key[4] >= 48 and key[4]) and (key[5] >= 48 and key[5]))

{

return 1;

}

else

{

return 0;

}

}

std::string enter\_key()

{

std::string key= "11цц11";

while (check\_key(key) != 1)

{

std::cout << "\nEnter a key: ";

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

{

std::cin >> key[i];

}

}

return key;

}

class table

{

public:

table(std::string path = "table.txt");

~table();

void create\_table();

void add(std::string key, std::string val = 0);

void show();

std::string find\_id(int id);

int find\_data(std::string data);

void del(int id);

void gen(int count);

void exprt(std::string destination);

void edit\_path(std::string path) { this->path = path; };

class str

{

public:

str(int id, std::string data, std::string val=0) { this->id = id; this->data = data; this->val = val; };

~str();

std::string data;

std::string val;

int id;

str\* next = nullptr;

void edit(std::string val ) { this->val = val; };

};

private:

std::string path;

int size=0;

str\* top = nullptr;

};

table::table(std::string path)

{

this->path = path;

}

table::~table()

{

}

void table::create\_table()

{

std::string respath, adj;

adj = ".txt";

respath = path + adj;

std::ofstream table(respath);

}

void table::add(std::string key, std::string val)

{

if (top==nullptr)

{

top = new str(size+1, key, val);

size++;

}

else

{

str\* current = top;

int id = find\_data(key);

if (id!=0)

{

for (int i = 0; i < id-1; i++)

{

current = current->next;

}

std::string value = current->val + " " + val;

current->edit(value);

}

else

{

while (current->next != nullptr)

{

current = current->next;

}

current->next = new str(size + 1, key, val);

size++;

}

}

}

void table::show()

{

if (size==0)

{

return;

}

for(str\* current = top; current!=nullptr; current=current->next)

{

std::cout << current->id << " " << current->data<<" "<<current->val << "\n";

}

std::cout << "\n";

}

std::string table::find\_id(int id)

{

if (id>size)

{

return std::string("Not found");

}

else

{

str\* current = top;

for (int i = 0; i < id-1; i++)

{

current = current->next;

}

std::string res = current->data;

return res;

}

}

int table::find\_data(std::string data)

{

for (str\* current = top; current != nullptr; current = current->next)

{

if (current->data.size()<data.size())

{

continue;

}

int match = 0;

for (int i =0; i< current->data.size() - data.size()+1; i++)

{

for (int q = 0; q < data.size(); q++)

{

if (current->data[i+q]!=data[q])

{

break;

}

else

{

match++;

}

}

if (match == data.size())

{

return current->id;

}

}

}

return 0;

}

void table::del(int id)

{

if (id>size)

{

return;

}

if (size==1 and id==1)

{

top = nullptr;

size--;

return;

}

str\* current = top;

if (id<=1)

{

top = top->next;

}

for (int i = 0; i < id - 2; i++)

{

current = current->next;

}

current->next = current->next->next;

size--;

}

void table::gen(int count)

{

std::string key= "11цц11";

srand(time(0));

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

{

try

{

key[0] = 48 + rand() % 10;

key[1] = 48 + rand() % 10;

key[2] = 65 + rand() % 26;

key[3] = 65 + rand() % 26;

key[4] = 48 + rand() % 10;

key[5] = 48 + rand() % 10;

add(std::to\_string(hash(key)), key);

}

catch (const std::exception&)

{

continue;

}

}

}

void table::exprt(std::string destination)

{

str\* current = top;

std::ofstream file(destination);

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

{

file << current->id << " " << current->data << " " << current->val << "\n";

current = current->next;

}

file.close();

}

int main()

{

table table;

std::cout << "1) to add random key\t2) to remove index\n3) to show the table\t4) to find id\n5) to find hashed key\t6) to export\n7)Enter your key\n\n88)to quit\n";

int v=1;

std::string data;

int ind, key;

while (v!=88)

{

std::cin >> v;

switch (v)

{

case 1:

std::cout << "How many: ";

std::cin >> ind;

table.gen(ind);

std::cout <<ind << " Random key`s added\n";

break;

case 2:

std::cout << "Enter index: ";

std::cin >> ind;

table.del(ind);

std::cout << "Removed\n";

break;

case 3:

table.show();

break;

case 4:

std::cout << "Enter id: ";

std::cin >> ind;

std::cout << table.find\_id(ind);

case 5:

std::cout << "Enter data: ";

std::cin >> data;

std::cout << table.find\_data(data);

case 6:

std::cout << "Enter Destination point (file.txt): ";

std::cin >> data;

table.exprt(data);

std::cout << "Exported to " << data<<"\n";

break;

case 7:

std::cout << "Enter key in format 11WW11: ";

data = enter\_key();

key = hash(data);

table.add(std::to\_string(key), data);

break;

default:

break;

}

}

}

**Проверка эффективности хеш функции**

Функция хеширования:

int hash(std::string key) //пример лкюча ццББцц

{

int value = (key[0] \* key[0] + key[1] \* key[1] + key[2] \* key[2] + key[3] \* key[3] + key[4] \* key[4] + key[5] \* key[5]) % 1500;

return value;

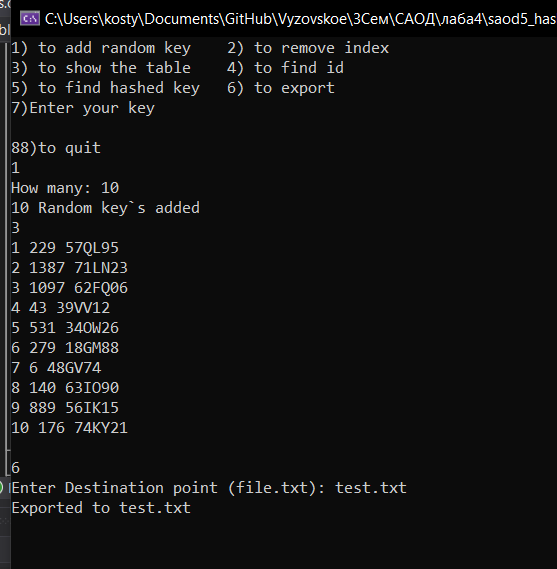
}

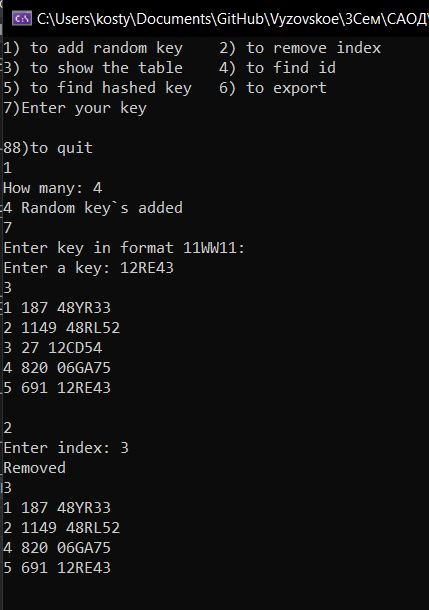
Программа берет код каждого символа в строке и складывает их квадраты, деля на количество сегментов (В моем случае 1500)

График идет на всей области определения равномерно. Скачок в начале диаграммы вызван особенностью функции rand(); из-за того, что оно является генератором псевдослучайных чисел. Так на старте программы код выполняется слишком быстро и seed для функции srand(time(0)); не успевает измениться. Этим и обусловлен скачек колизий на старте программы.  
диаграмма основана на 3000 ключей



**Результат работы**





**Вывод**

Я освоил методы ведения хештаблиц и хеширования.