ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

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

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

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134K |  |  |  | Опарин С.Н |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2022

**Лабораторная работа №4:** **«ХЕШИРОВАНИЕ ДАННЫХ»**

**Цель работы:**

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

**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 задания:**



**Описание хеш-функции:**

Используя заданный формат ключа и количество сегментов в таблице, составляем хеш- функцию. Ключ представляет собой строку, содержащую буквы и цифры в определенных позициях. Попробуем подход, основанный на сложении остатков от количества сегментов, полученных суммой кодов символов из таблицы ASCII и целочисленного деления количества сегментов на порядок символа в ключе. Далее полученный результат умножается на два, к нему прибавляется единица и находится остаток от деления на количество сегментов в хэш-таблице.

**Листинг программы:**

#include <iostream>

#include <ctime>

#include <fstream>

using namespace std;

class HashTable {

private:

constexpr static const double rehash\_size = 0.75;

struct Node{

bool is\_deleted;

string data;

int colCount = 0;

Node(const string &data) {

this -> data = data;

is\_deleted = false;

}

};

Node \*\*arr;

int size, size\_non\_nullptr, buffSize;

int hash(string str) {

int hashResult = 0;

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

hashResult += ((buffSize / (i + 10)) \* hashResult + int(str[i])) % buffSize;

return (hashResult \* 2) % buffSize;

}

public:

HashTable(int size) {

buffSize = size;

size\_non\_nullptr = this -> size = 0;

arr = new Node \* [buffSize];

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

arr[i] = nullptr;

}

}

void Print() {

cout << "Хеш-таблица:" << endl;

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

if (arr[i] != nullptr && !arr[i] -> is\_deleted) {

cout << i + 1 << ": " << arr[i] -> data << endl;

}

}

}

void Create(const int length) {

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

delete arr[i];

}

delete[] arr;

buffSize = length;

size\_non\_nullptr = this -> size = 0;

arr = new Node \* [buffSize];

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

arr[i] = nullptr;

}

}

bool Find(const string data) {

int index = hash(data);

int i = 0;

while (arr[index] != nullptr && i < buffSize) {

if (arr[index] -> data == data && !arr[index] -> is\_deleted) {

cout << "Введенный ключ имеет индекс " << index + 1 << " в хеш-таблице." << endl;

return true;

}

/\*

++i;

index = (index + (i + 3 \* i \* i)) % buffSize;

\*/

index = (index + (2 \* i + 3 \* i \* i)) % buffSize;

++i;

}

return false;

}

void findByIndex(){

int indx;

cout << "Введите индекс, который вы хотите найти: ";

cin >> indx;

if(indx > buffSize){

cout << "Введите индекс, меньше 6000: ";

cin >> indx;

}

if(arr[indx - 1] != nullptr && !arr[indx - 1] -> is\_deleted){

cout << "Ключ с индексом " << indx << " равен " << arr[indx - 1] -> data << endl;

}

else{

cout << "Ячейка пустая" << endl;

}

}

bool Remove(const string data) {

int index = hash(data);

int i = 0;

while (arr[index] != nullptr && i < buffSize) {

if (arr[index] -> data == data && !arr[index] -> is\_deleted) {

arr[index]->is\_deleted = true; // Cделать вывод элементов, вызвавших коллизию с удаляемым. Сделано!

--size;

i = 1;

index = (index + (2 \* i + 3 \* i \* i)) % buffSize;

// while (arr[index] != nullptr) {

// cout << "Элемент " << index << ": " << arr[index]->data << " вызывал коллизию с удаленным элементом" << endl;

// ++i;

// index = (index + (2 \* i + 3 \* i \* i)) % buffSize;

// }

return true;

}

index = (index + (2 \* i + 3 \* i \* i)) % buffSize;

++i;

}

return false;

}

void Resize() {

int old\_buffSize = buffSize;

size\_non\_nullptr = size = 0;

buffSize \*= 2;

Node\*\* arr2 = new Node \* [buffSize];

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

arr2[i] = nullptr;

swap(arr, arr2);

for (int i = 0; i < old\_buffSize; ++i) {

if (arr2[i] && !arr2[i]->is\_deleted)

Add(arr2[i]->data); // добавляем элементы в новый массив

}

for (int i = 0; i < old\_buffSize; i++) {

delete arr2[i];

}

delete[] arr2;

}

void Rehash()

{

size\_non\_nullptr = 0;

size = 0;

Node\*\* arr2 = new Node \* [buffSize];

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

arr2[i] = nullptr;

std::swap(arr, arr2);

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

if (arr2[i] && !arr2[i]->is\_deleted)

Add(arr2[i]->data);

}

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

if (arr2[i])

delete arr2[i];

delete[] arr2;

}

bool Add(const string data) {

//if (size + 1 > int(rehash\_size \* buffSize))

// Resize();

//else if (size\_non\_nullptr > 2 \* size)

// Rehash();

int index = hash(data);

int i = 0, first\_deleted = -1;

while (arr[index] != nullptr && i < buffSize) {

if (arr[index]->data == data && !arr[index]->is\_deleted)

return false;

if (arr[index]->is\_deleted && first\_deleted == -1) // находим место для нового элемента

first\_deleted = index;

else{

arr[index] -> colCount++;

}

index = (index + (2 \* i + 3 \* i \* i)) % buffSize;

++i;

}

if (first\_deleted == -1) {

arr[index] = new Node(data);

++size\_non\_nullptr;

}

else {

arr[first\_deleted]->data = data;

arr[first\_deleted]->is\_deleted = false;

}

++size;

return true;

}

void write() {

ofstream out\_file("/Users/sergeytremplix/Desktop/Структуры и алгоритмы обработки данных/Лабораторная работа №4/Лабораторная работа №4/hash.txt", ios\_base::out);

if (!out\_file.is\_open()) {

cerr << "Не удалось найти файл или он не открылся" << endl;

return;

}

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

if (arr[i] && !arr[i] -> is\_deleted)

out\_file << arr[i] -> colCount << " " << i << endl;

}

out\_file.close();

cout << "Данные загрузились в файл" << endl;

}

~HashTable() {

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

delete arr[i];

}

delete[] arr;

}

};

bool check\_key(string key) {

bool check = true;

int length = 0;

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

if ((i == 0 || i == 5) && (key[i] < 65 || key[i] > 90) )

check = false;

else if ((i != 0 && i != 5) && (key[i] < 48 || key[i] > 57))

check = false;

length++;

}

if (length != 6)

return false;

if (!check)

return false;

else

return true;

}

string create\_key() {

char possibleNumbers[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' };

char possibleLiterals[] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O','P', 'Q', 'R', 'S', 'T','U', 'V', 'W', 'X', 'Y', 'Z' };

int LENGTH=6;

string key;

string temp;

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

if (i == 0 || i == 5) {

temp = possibleLiterals[rand() % 26];

} else {

temp = possibleNumbers[rand() % 10];

}

key = key + temp;

}

return key;

}

void menu() {

cout << "\n---------------------------------------------" << endl;

cout << "0 - Создать таблицу(будет удалена предыдущая)" << endl;

cout << "1 - Сгенерировать" << endl;

cout << "2 - Добавить элемент" << endl;

cout << "3 - Вывести таблицу" << endl;

cout << "4 - Найти элемент" << endl;

cout << "5 - Удалить элемент" << endl;

cout << "6 - Найти элемент по индексу" << endl;

cout << "7 - Записать в файл" << endl;

cout << "8 - Выход" << endl;

cout << "---------------------------------------------" << endl;

}

void select(HashTable& table) {

while (true) {

menu();

int choice;

cout << "Ваш выбор: ";

cin >> choice;

switch (choice) {

case 0: {

int size;

cout << "Введите размер таблицы: ";

cin >> size;

if (size > 3000)

cout << "Размер таблицы не должен превышать 3000" << endl;

else

table.Create(size);

break;

}

case 1: {

int number;

cout << "Введите количество сгенерируемых ключей: ";

cin >> number;

if (number > 3000)

cout << "Количество ключей не должно превышать 3000" << endl;

else {

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

table.Add(create\_key());

}

}

break;

}

case 2: {

string key;

cout << "Шаблон: БццццБ\nВведите ключ для добавления: ";

cin >> key;

if (check\_key(key))

table.Add(key);

else

cout << "Введенный ключ некорректен." << endl;

break;

}

case 3: {

table.Print();

break;

}

case 4: {

string key;

cout << "Шаблон: БццццБ\nВведите ключ для поиска: ";

cin >> key;

if (check\_key(key)) {

if(!table.Find(key))

cout << "Ключ " << key << " не найден!" << endl;

}

else

cout << "Введенный ключ некорректен." << endl;

break;

}

case 5: {

string key;

cout << "Шаблон: БццццБ\nВведите ключ для удаления: ";

cin >> key;

if (check\_key(key)) {

if (!table.Remove(key)){

cout << "Ключ " << key << " не найден!" << endl;

}

else{

cout << "Элемент успешно удален" << endl;

}

}

else

cout << "Введенный ключ некорректен." << endl;

break;

}

case 6: {

table.findByIndex();

break;

}

case 7: {

table.write();

break;

}

case 8: {

exit(0);

break;

}

}

}

}

int main(){

HashTable Table(100);

select(Table);

return 0;

}

**Вывод:**

В ходе лабораторной работы познакомился с основными методами хеширования данных и получил практические навыки в работе с хеш-таблиц, освоил метод разрешения коллизий «квадратичное опробование.  
Работал с текстовыми файлами и их содержимым.