МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

Тема: Хэш-таблицы

Студент гр. 8304 _____ Алтухов А.Д.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Цель работы.

Ознакомиться с такой структурой данных, как хэш-таблица, способами ее представления и реализации.

Задание.

Вариант 23.

Хеш-таблица с цепочками; действие: 1+2а.

1) По заданному файлу F (типа file of Elem), все элементы которого различны, построить структуру данных определённого типа – БДП или хеш-таблицу;

2) Выполнить одно из следующих действий:

а) Для построенной структуры данных проверить, входит ли в неё элемент e типа Elem, и если не входит, то добавить элемент e в структуру данных. Предусмотреть возможность повторного выполнения с другим элементом.

Описание алгоритма.

Хэш-таблица реализована на массиве однонаправленных списков. Применена полиномиальная хэш-функция вида

$$(s_1a^1 + s_2a^2 + \ldots + s_na^n) \; mod \; len,$$

где s_i — элемент строки, а a — некоторый коэффициент. Файл со входными данными открывается, считанные строки до разделительного символа (0) записываются в хэш-таблицу. Далее считываются строки, которые ищутся в таблице путем хеширования и перебора списка в данной ячейке таблицы. Если элемент не найден, он добавляется в таблицу.

Основные функции и структуры.

`class HashTable` — реализация хэш-таблицы.

`std::forward_list<std::string>* arr` — массив с информацией.

`size_t len` — количество выделенной памяти.

`size_t realLen` — количество элементов в массиве.

`unsigned long long* coeffs` — массив коэффициентов.

`unsigned long long hash(std::string elem)` — функция хэширования, описанная в разделе описания алгоритма.

`void expandTable()` — функция расширения и перестройки таблицы.

`void insert(std::string elem)` — функция вставки нового элемента в таблицу.

`bool find(std::string elem, unsigned long long myHash)` — функция проверки наличия элемента в таблице. В случае наличия высчитанного заранее хэша элемента, он передается вторым аргументом.

`void checkCollisions() const` — проверка записанных элементов на наличие коллизий.

`size_t getLen() const` — получение количества выделенной памяти под таблицу.

Тестирование.

№	Ввод	Полученный ответ
1	строка1 другая строка 0 строка2	Введена строка: строка2 Поиск в таблице... Не найдено. Элемент добавлен в таблицу.
2	stroka6 ss да нет строки 0 строка есть строка есть нет строки	Введена строка: строка есть Поиск в таблице... Не найдено. Элемент добавлен в таблицу. Введена строка: строка есть Поиск в таблице... Найдено. Введена строка: нет строки Поиск в таблице... Найдено.
3	Вышел заяц на крыльцо 12345 0 вышел заяц на крыльцо 12346 оцьлырк ан цяаз лешыв	Введена строка: вышел заяц на крыльцо Поиск в таблице... Не найдено. Элемент добавлен в таблицу. Введена строка: 12346 Поиск в таблице... Не найдено.

		Элемент добавлен в таблицу. Введена строка: оцьлырк ан цяаз лешыв Поиск в таблице... Не найдено. Элемент добавлен в таблицу.
4	А ну спать иди Завтра тест как сделать economiks а б в 0 Завтра тест? а г д е	Введена строка: Завтра тест? Поиск в таблице... Не найдено. Элемент добавлен в таблицу. Введена строка: а Поиск в таблице... Найдено. Введена строка: г Поиск в таблице... Не найдено. Элемент добавлен в таблицу. Введена строка: д Поиск в таблице... Не найдено. Элемент добавлен в таблицу. Введена строка: е Поиск в таблице... Не найдено. Элемент добавлен в таблицу.
5	Каждый день — новый. Каждый день — ваш! Обновлён сегодня Алиса собирает и комментирует треки специально для вас. Скоро Треки из вашей коллекции, которые вы могли забыть Вы ещё не слушали эти треки, но, похоже, вам они понравятся 0 Обновлён сегодня Обновлён 8 декабря Мотивирующие цитаты каждый день	Введена строка: Обновлён сегодня Поиск в таблице... Найдено. Введена строка: Обновлён 8 декабря Поиск в таблице... Не найдено. Элемент добавлен в таблицу. Введена строка: Мотивирующие цитаты каждый день Поиск в таблице... Не найдено. Элемент добавлен в таблицу.

Выводы.

В ходе выполнения лабораторной работы была исследована такая структура данных, как хэш-таблица, изучены методы работы с ней.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД

Файл HashTable.h

```
#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <forward_list>

constexpr size_t DEFAULT_LEN = 103;//стартовая длина
constexpr size_t COEFF = 31; //стартовый множитель для хэша
constexpr size_t COEFFS_COUNT = 100; //количество степеней множителя,
высчитываемых заранее
constexpr size_t PRIMES_COUNT = 24;//количество простых чисел для
расширения хэш-таблицы
constexpr size_t PRIMES[] = { 103 , 211, 331 , 449 , 587 , 709 , 853 ,
991 , 1117 , 1279 , 1433 , 1567 , 1709 ,   1873 , 2027 , 2179 , 2341 ,
2477 , 2671 , 2797 , 2963 ,     3163, 3319, 3469 };


class HashTable {
private:
      std::forward_list<std::string>* arr; //массив с информацией
      size_t len; //количество выделенной памяти
      size_t realLen; //количество элементов в массиве
      unsigned long long* coeffs;//массив коэффициентов
      unsigned long long hash(std::string elem);
public:
      void expandTable();
public:
      HashTable();
      ~HashTable();

      void insert(std::string elem);
      bool find(std::string elem, unsigned long long myHash);
      void checkCollisions() const;
      size_t getLen() const;

};
```

Файл HashTable.cpp

```
#include "HashTable.h"
```

```
HashTable::HashTable() :arr(new
std::forward_list<std::string>[DEFAULT_LEN]), len(DEFAULT_LEN),
realLen(0), coeffs(new unsigned long long[COEFFS_COUNT]) {
//высчитываем коэффициенты для хэша
	for (int i = 0; i < COEFFS_COUNT; i++) {
		coeffs[i] = (i > 0) ? coeffs[i - 1] * COEFF : COEFF;
	}
}

HashTable::~HashTable() {
	delete[] coeffs;
	delete[] arr;
}

unsigned long long HashTable::hash(std::string elem) {//полиномиальная
функция хэширования (s1*a^1 + s2*a^2 + ... + sn*a^n) mod len

	unsigned long long myHash = 0;
	for (int i = 0; i < elem.length(); i++) {
		myHash += elem[i] * coeffs[i%COEFFS_COUNT];//если нужно
больше коэффициентов, то и так сойдет
	}
	myHash %= len;
	return myHash;

}

bool HashTable::find(std::string elem, unsigned long long myHash) {
	if (myHash == len)//если хеш еще не посчитан то передается
несуществующий
		myHash = hash(elem);
	if (!arr[myHash].empty()) {
		auto current = arr[myHash].begin();
		auto end = arr[myHash].end();
		while (current != end) {
			if (*current == elem)
				return true;
			current++;
		}
		return false;
	}
	else return false;
}


void HashTable::insert(std::string elem) {

	if (realLen > 0.9 * len)
		expandTable();
```

```
	unsigned long long myHash = hash(elem);
	if (find(elem, myHash)) { //а вот тут уже высчитан хэш, передадим
его чтобы не пересчитывать
		return;
	}


	arr[myHash].push_front(elem);
	realLen++;

}

void HashTable::checkCollisions() const {

	std::cout << "Проверка коллизий...\n";
	for (int i = 0; i < len; i++) {
		if (!arr[i].empty()) {
			std::cout << "Занятая ячейка " << i <<"\n";
			auto current = arr[i].begin();
			auto end = arr[i].end();
			size_t counter = 0;
			std::string collisions;//содержимое ячейки
			while (current != end) {
				if (!counter) {
					collisions.clear();
				}
				collisions += *current + "\n";
				counter++;
				current++;
			}
			if (counter > 1) {
				std::cout << "Коллизия!\n" << collisions; //<<
"\n";
			}
		}
	}
}
void HashTable::expandTable() {

	std::cout << "Расширение таблицы!\n";
	size_t oldLen = len;
	for (int i = 0; i < PRIMES_COUNT; i++) {
		if ((PRIMES[i] == oldLen) && (i + 1) < PRIMES_COUNT) {
			len = PRIMES[i + 1]; //за новую длину берем следующее
простое число из заданных
			break;
		}
		else if ((i + 1) >= PRIMES_COUNT)
			len += DEFAULT_LEN; //если кончились, что поделать
	}
```

```
	realLen = 0; //само пересчитается в insert
	auto oldArr = arr;
	arr = new std::forward_list<std::string>[len];

	for (int i = 0; i < oldLen; i++) {
		if (!oldArr[i].empty()) {
			auto current = oldArr[i].begin();
			auto end = oldArr[i].end();
			while (current != end) {
				insert(*current);
				current++;
			}
		}
	}

	delete[] oldArr;

}

size_t HashTable::getLen() const {
	return len;
}
```

Файл main.cpp

```
#include <iostream>
#include <locale>
#include <fstream>

#include "HashTable.h"

int main(int argc, char* argv[]) {
	setlocale(LC_ALL, "Russian");

	std::cout << "\n\n=====Запуск программы=====\n\n\n";

	std::ifstream inputF;

	if (argc > 1)
		inputF.open(argv[1]);
	else
		inputF.open("input.txt");

	if (!inputF.is_open()) {
		std::cerr << "Невозможно открыть файл со входными данными";
		if (argc == 1)
			std::cerr << " input.txt";
		else
			std::cerr << " " << argv[1];
		std::cerr << "\n";
		return 0;
```

```
	}


	std::ofstream outputF("output.txt");
	if (!outputF.is_open()) {
		std::cerr << "Невозможно открыть файл вывода output.txt\n";
		return 0;
	}


	HashTable hashTable;

	std::string nextLine;

	while (std::getline(inputF, nextLine) && (nextLine != "0")) {//0
- разделитель между стартовыми данными и данными, которые надо
проверить
		hashTable.insert(nextLine);
	}

	hashTable.checkCollisions();
	std::cout << "\n\n";

	while (std::getline(inputF, nextLine)) {
		std::cout << "Введена строка " << nextLine << "\nПоиск в
таблице...";
		outputF << "Введена строка " << nextLine << "\nПоиск в
таблице...";
		if (hashTable.find(nextLine, hashTable.getLen())) {
			std::cout << " Найдено.\n";
			outputF << " Найдено.\n";
		}
		else {
			std::cout << " Не найдено. Элемент добавлен в
таблицу.\n";
			outputF << " Не найдено. Элемент добавлен в
таблицу.\n";
			hashTable.insert(nextLine);
		}
	}
	hashTable.checkCollisions();

	inputF.close();
	outputF.close();

	std::cout << "\n\n=====Завершение программы=====\n\n";

	return 0;
}
```