Министерство науки и высшего образования РФ  
Федеральное государственное бюджетное образовательное учреждение высшего образования

«Воронежский государственный университет»

**Факультет прикладной математики, информатики и механики  
Кафедра математического обеспечения ЭВМ**

**КУРСОВАЯ РАБОТА**

По дисциплине «Языки и методы программирования»

На тему: «Хеширование»

Выполнили:

студенты 2 курса 61 группы

Гаврилов А.М., Ермаченко И.В.  
Кушнарёв Н.М., Мурзинов П.Д.

Преподаватель:

Горбенко О. Д.

Воронеж, 2019

**Содержание**

[Глава 1. Общее описание HashTable 3](#_Toc9183322)

[Особенности HashTable 3](#_Toc9183323)

[Структура Node 3](#_Toc9183324)

[Глава 2. Описание основных методов 4](#_Toc9183325)

[Добавление элементов 4](#_Toc9183326)

[Поиск данных по ключу 4](#_Toc9183327)

[Удаление элемента по ключу 5](#_Toc9183328)

[Инициализация таблицы 5](#_Toc9183329)

[Получение размера таблицы 5](#_Toc9183330)

[Глава 3. Реализация на C++ 6](#_Toc9183331)

[Файл THash\_Table.h 6](#_Toc9183332)

[Файл THash\_Table.cpp 7](#_Toc9183333)

[Файл Hash\_Table.cpp 9](#_Toc9183334)

**Глава 1. Общее описание HashTable**

Хеш-таблица представлена классом **THash\_table** с массивом **table** типа **Node**. Каждый элемент массива **table** представляет собой структуру **Node**, которая определяет данные пользователя и ключ

**Особенности HashTable**

* Размер таблицы задаётся вручную и равен N, но память под массив table выделяется 2\*N для более эффективного добавления элементов
* Максимальная наполненность HashTable равна N
* При коллизиях используется квадратичное опробование с помощью специально подобранных констант «*c*» и «*d*»
* Индекс для элемента определяется *методом деления.*

**Структура Node**

Поля структуры:

* **TItem –** поле данных
* **TKey** – ключ элемента
* **Status** – состояние ячейки(0 – свободна, 1 – занята, -1 – удалён)

Поле данных **TItem** представлено в виде структуры, содержащей поле string для данных пользователя.

**Глава 2. Описание основных методов**

**Добавление элементов**

Вставки очередного элемента происходит функцией **Insert**:

1. Проверяется наполненность таблицы, если таблица заполнена, то функция выдаёт ошибку и завершается с кодом false, что означает неудачное добавление элемента
2. Потом с помощью хеш-функции **hashkey** генерируется индекс в массиве **table**
3. Далее идёт проверка на коллизию и если она есть происходит разрешение коллизии методом квадратичного опробования
4. Во время разрешения коллизии проверяется ключ на совпадение в соответствующей ячейке массива, и если ключи совпадают, то функция выдаёт ошибку, что элемент с таким ключом уже был добавлен ранее и завершается кодом false
5. Если предыдущие шаги не завершили функцию (в случае переполненности таблицы или повторного добавления одного и того же ключа), то в найденную ячейку массива заносятся данные и ключ добавляемого элемента, а также увеличивается счётчик числа добавленных элементов.

**Поиск данных по ключу**

Шаги:

1. Вычислить по ключу индекс в массиве функцией **hashkey**.
2. Перейти по индексу и сравнить ключ c ключом в ячейке с найденным индексом. Если они равны, то элемент найден и переданному в функцию по ссылке элементу **item** присвоить значение поля **item** структуры **Node** из найденной ячейки
3. Если ключи неравны, то определить следующий индекс возможного расположения нужного элемента с помощью метода, разрешающего коллизию и перейти к п.2
4. Если вдруг во время поиска наткнулись на пустую ячейку, то поиск завершается, и функция возвращает false, что означает, что элемент не был найден

**Удаление элемента по ключу**

Шаги:

1. Вычислить по ключу индекс в массиве функцией **hashkey**.
2. Перейти по индексу и сравнить ключ c ключом в ячейке с найденным индексом. Если они равны и ключ не был удален раннее, то нужная ячейка найдена и меняем её статус (поле **status** структуры **Node**) на удалённую (значение **status** = -1)
3. Если ключи неравны, то определить следующий индекс возможного расположения нужного элемента с помощью метода, разрешающего коллизию и перейти к п.2
4. Если вдруг во время поиска наткнулись на пустую ячейку (**status** = 0), то поиск завершается, и функция возвращает false, что означает, что элемент не был удалён по причине его отсутствия

**Инициализация таблицы**

Инициализация таблицы происходит с помощью файла, имя которого передаётся в функцию.

**Получение размера таблицы**

Вместе с получением размера таблицы пользователю даётся информация о количестве свободных и заполненных ячеек.

**Глава 3. Реализация на C++**

**Файл THash\_Table.h**

#pragma once

#include <string>

#include<fstream>

using namespace std;

struct TItеm //Данные

{

string data;

};

typedef int Tkey;

struct Node

{

TItеm itеm;

Tkey key;

int status;//Состояние 0-свободно,1-занято,-1 - удален

};

class THash\_Table

{

private:

//Кол-во элементов в таблице

int N;

//Указатель на массив таблицы

Node \*table;

//Хеш-Функция

int hashKey(Tkey key);

//Определяем константы для квадратичного пробирования

const int c = 3, d = 9;

//Кол-во занесенных элементов в таблицу

int count\_Items;

public:

//Конструктор

THash\_Table(int N);

//Инициализация таблицы

void initTable(string fileName);

//Сохранение данных в файл

void WriteTable(string fileName);

//Добавление эл-та

bool Insert(Node node);

//Поиск по ключу

bool Find(Tkey key, TItеm &item);

//Удаление по ключу

bool Delete(Tkey key);

// Полученuе размера хеш-таблицы

int GetSize();

};

**Файл THash\_Table.cpp**

#include "pch.h"

#include"THash\_Table.h"

#include<fstream>

#include <iostream>

int THash\_Table::hashKey(Tkey key)

{

return key % N;

}

THash\_Table::THash\_Table(int n)

{

N = n; // установить размер таблицы

table = new Node[2 \* N];

// инициализировать данные

for (int i = 0; i < 2 \* N; ++i)

table[i].status = 0;

count\_Items = 0;

}

bool THash\_Table::Insert(Node node) // функция вставки элемента

{

if (count\_Items >= N) // если таблица заполнена

{

std::cerr << "Таблица переполнена!" << endl;

return false;

}

int hkey = hashKey(node.key); // получить индекс

int i = 0; //шаг коллизии

while (table[hkey].status == 1)//Пока ячейка занята - двигаемся дальше

{

if (table[hkey].key == node.key) // есди элемент с таким ключом уже есть

{

std::cerr << "Элемент с ключом " << node.key << " уже существет!" << endl;

return false;

}

hkey = (hkey + c + d \* d\*i\*i) % (2 \* N); // если снова коллизия - изменить индекс

++i;

}

//Заполняем ячейку массива

node.status = 1;

table[hkey] = node;

count\_Items++; // увеличить счетчки элементов

return true;

}

bool THash\_Table::Find(Tkey key, TItеm &item) // функция поиска

{

int hkey = hashKey(key); // получить инедкс из хеш-функции

//шаг коллизии

int i = 0;

while (table[hkey].status != 0) // пока не дойдём до пустого поля

{

if (table[hkey].key == key && table[hkey].status == 1) { // если нашли ключ и он не удалён

item = table[hkey].itеm;

return true;

}

hkey = (hkey + c + d \* i\*i) % (2 \* N); // если коллизия - перейти к след. элементу

++i;

}

return false;

}

bool THash\_Table::Delete(Tkey key) // функция удаления

{

int hkey = hashKey(key);

//шаг коллизии

int i = 0;

while (table[hkey].status != 0) // пока не дойдём до пустого поля

{

if (table[hkey].key == key && table[hkey].status == 1) // если нашли элемент

{

table[hkey].status = -1;

count\_Items--;

return true;

}

hkey = (hkey + c + d \* i\*i) % (2 \* N); // индекс следующего поля

++i;

}

return false;

}

void THash\_Table::initTable(string fileName) // Инициализация таблицы из файла

{

ifstream ifile(fileName);

Node node;

while (!ifile.eof() && count\_Items < N)

{

ifile >> node.itеm.data;

ifile >> node.key;

Insert(node);

}

}

void THash\_Table::WriteTable(string fileName) // Запись таблицы в файл

{

ofstream ofile(fileName);

for (int i = 0; i < 2 \* N; ++i)

{

if (table[i].status)

{

ofile << table[i].itеm.data << '\t';

ofile << table[i].key << '\t';

ofile << '\n';

}

}

}

int THash\_Table::GetSize() // получение размера таблицы

{// заодно посчитаем кол-во свободных и занятых ячеек

int count = 0; //

for (int i = 0; i < 2 \* N; ++i)

{

if (table[i].status == 1)

count++;

}

cout << "Занято: " << count << endl;

cout << "Свободно: " << N - count << endl;

return N;

}

**Файл Hash\_Table.cpp**

#include "pch.h"

#include <iostream>

#include "THash\_Table.h"

#include <string>

using namespace std;

void testTable(THash\_Table &table);

int main()

{

setlocale(LC\_ALL, "Russian");

int n;

cout << "Введите размер таблицы: ";

cin >> n;

if (n <= 0)

{

cerr << "Некорректные данные";

return -1;

}

THash\_Table table = THash\_Table(n);

table.initTable("input.txt");

testTable(table); // функция для тестирования таблицы

}

void testTable(THash\_Table &table)

{

cout << "Добавить/Удалить/Найти/Получть размер/Записать в файл/выйти? A/D/F/S/W/E" << endl; // add,delete,find,exit

char answer;

cin >> answer;

answer = (char)toupper(answer); // преобразовать в старший регистр

switch (answer)

{

case 'A': // добавление элемента

{

Node node;

int key;

cout << "Веддите: данные ключ" << endl;

cin >> node.itеm.data;

cin >> node.key;

if (table.Insert(node)) {

cout << "Элемент " << node.itеm.data << " успешно добавлен" << endl;

}

else

{

cout << "Ошибка добавления!" << endl;

}

break;

}

case 'D': // удаление элемента

{

Tkey key;

cout << "Веддите: ключ" << endl;

cin >> key;

if (table.Delete(key))

{

cout << "Элемент с ключом " << key << " успешно удален" << endl;

}

else

{

cout << "Невозможно удалить элемент с ключом " << key << endl;

}

break;

}

case 'F': // посик жлемента по ключу

{

Tkey key;

cout << "Веддите: ключ" << endl;

cin >> key;

TItеm item;

if (table.Find(key, item)) {

cout <<"Элемент: " << item.data << endl;

}

else

{

cerr << "Ключ не найден!" << endl;

}

break;

}

case 'W': // запись таблицы в текствый файл

{

string fileName;

cout << "Введите имя файла(с расширением):" << endl;

cin >> fileName;

table.WriteTable(fileName);

break;

}

case 'S': // получение размера таблицы

{

cout << "Размер: " << table.GetSize() << endl;

break;

}

case 'E': // выход из тестируемой функции

{

return;

break;

}

default:

cout << endl << endl;

testTable(table);

break;

}

cout << endl << endl;

testTable(table);

}