**Лабораторная работа №5**

**Хеширование данных. Поиск данных в хеш-таблице.**

Разработать программу хеширования данных из файла методом открытого хеширования. Хеш-таблицу закодировать как класс, а операции с ней - как функции-члены класса. В программе предусмотреть следующий действия:

а) создание хеш-таблицы с небольшим количеством классов (B<=30) и небольшим набором исходных данных, вывод всей таблицы на экран, осуществление поиска по таблице, в качестве хеш-функции выбрать h(x)=x%B;

б) создание хеш-таблицы, для которой количество классов B задается пользователем (до 20000 - 50000), осуществление поиска по таблице, подсчет общего числа коллизий в таблице, нахождение самой длинной цепочки коллизий, процента заполняемости таблицы. В качестве хеш-функции выбрать h(x)=(ax+c)%B, проанализировать результаты заполняемости таблицы для различных a и c.

В качестве исходных данных выбрать: Файл целых чисел (варианты 1,4,7,10).

Описание алгоритма.

Класс HashTable

В private:

Описаны переменные B(количество классов), коэффициенты a и c, longest\_chain (максимальная длина цепочки коллизий), collisions (число коллизий), массив data для хранения хеш-таблицы. Так же содержится хеш-функция.

В public:

Присутствует конструктор с инициализацией количества классов(B), параметров a и c, обнуление количества коллизий(collisions) и максимальной цепочки(longest\_chain). Заполнение в массива data линейными списками.

Деструктор для удаления таблицы.

Функция добавления элемента(insert). Элемент хешируется с помощью хеш-функции и добавляется в таблицу. В данной функции так же производится подсчет коллизий и максимальной их цепочки.

Функция поиска(search). Искомый элемент хешируется, производится поиск по всему массиву. Если присутствует, то выводим позицию.

Функция вывода таблицы(print\_table). Выводит таблицу.

Функция вывода максимальной цепочки коллизии(col). Выводит максимальную цепочку коллизий.

Функция вывода характеристик(get\_stats). Проходит по таблице и считает количество заполненных классов. После чего считает процент заполнения. Выводит количество коллизий, их максимальную длину и процент заполнения.

Функция считывания с файла(read).

Создает элемента класса HashTable и считывает в него элементы с файла.

Функция(main).

Запрашиваем у пользователя количество классов.

Если их меньше 30, то создаём элемент класса HashTable и задаем значения коэффициентов a = 1, c = 0. После чего считываем с помощью функции read значения с файла.

После выводим таблицу.

Производим поиск элемента введенного пользователем.

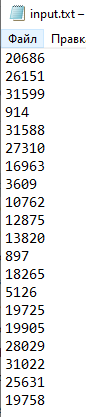
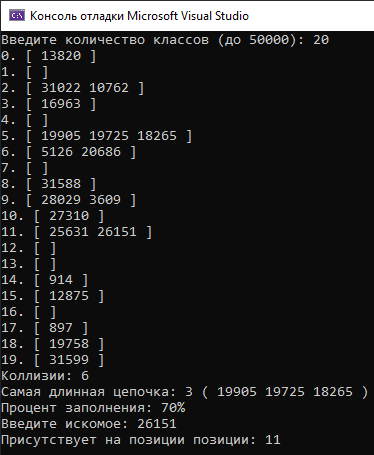
Для количества классов более 30.

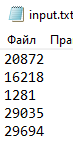
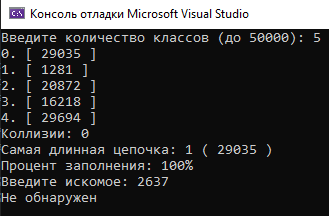
Запрашиваем у пользователя коэффициенты. Далее создаём элемент класса HashTable и задаем значения коэффициентов. После чего считываем с помощью функции read значения с файла.

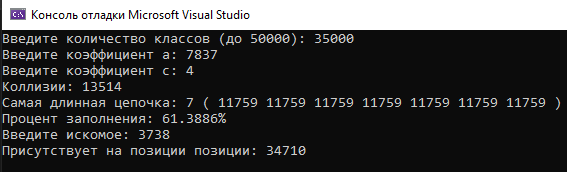
Выводим количество коллизий, их максимальную длину и процент заполняемости.

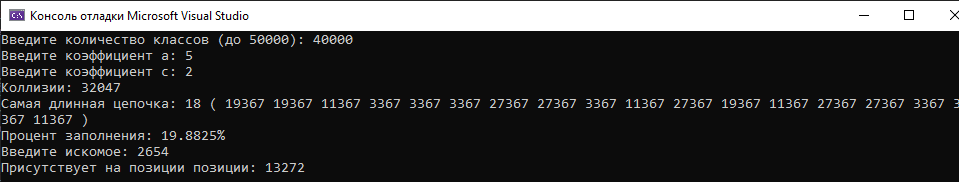
Производим поиск элемента введенного пользователем.

Тесты





Анализ:

1. «c» не влияет на заполняемость
2. «a» существенно влияет на заполняемость – если a и B взаимно простые числа/

Код программы

#include <iostream>

#include <fstream>

#include <list>

#include <random>

#include <ctime>

using namespace std;

// Класс хеш-таблицы

class HashTable {

private:

int B, a, c;

int collisions;

int longest\_chain;

list<int>\* table;

int hash\_function(int key) {

return (a \* key + c) % B;

}

public:

HashTable(int B, int a, int c) : B(B), a(a), c(c), collisions(0), longest\_chain(0) {

table = new list<int>[B];

}

~HashTable() {

delete[] table;

}

void insert(int key) {

int index = hash\_function(key);

table[index].push\_front(key);

collisions += (table[index].size() > 1);

longest\_chain = max(longest\_chain, (int)table[index].size());

}

int search(int key) {

int index = hash\_function(key);

for (int k : table[index]) {

if (k == key) {

return index;

}

}

return 0;

}

void col(int l\_c) {

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

if (table[i].size() == l\_c) {

for (int k : table[i]) {

cout << k << " ";

}

return;

}

}

}

void print\_table() {

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

cout << i << ". [ ";

for (int k : table[i]) {

cout << k << " ";

}

cout << "]" << endl;

}

}

void get\_stats() {

int filled\_classes = 0;

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

if (!table[i].empty()) {

filled\_classes++;

}

}

cout << "Коллизии: " << collisions << endl;

cout << "Самая длинная цепочка: " << longest\_chain << " ( ";

col(longest\_chain);

cout << ")" << endl;

cout << "Процент заполнения: " << (double)filled\_classes / B \* 100 << "%" << endl;

}

};

HashTable read(const string& filename, int B, int a, int c) {

HashTable table(B, a, c);

ifstream file(filename);

int num;

while (file >> num) {

table.insert(num);

}

file.close();

return table;

}

//void create\_random\_file(const string& filename, int count) {

// ofstream file(filename);

// srand(time(0));

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

// file << rand() % 500000 << endl;

// }

// file.close();

//}

int main() {

setlocale(LC\_ALL, "RUS");

//create\_random\_file("input.txt", 40000);

int B;

cout << "Введите количество классов (до 50000): "; cin >> B;

if (B <= 30) {

HashTable table = read("input.txt", B, 1, 0);

table.print\_table();

table.get\_stats();

int tmp;

cout << "Введите искомое: "; cin >> tmp;

tmp = table.search(tmp);

if (tmp) {

cout << "Присутствует на позиции позиции: " << tmp << endl;

}

else

cout << "Не обнаружен" << endl;

}

else {

int a, c;

cout << "Введите коэффициент a: "; cin >> a;

cout << "Введите коэффициент c: "; cin >> c;

HashTable table = read("input.txt", B, a, c);

table.get\_stats();

int tmp;

cout << "Введите искомое: "; cin >> tmp;

tmp = table.search(tmp);

if (tmp) {

cout << "Присутствует на позиции позиции: " << tmp << endl;

}

else

cout << "Не обнаружен" << endl;

}

return 0;

}