**ЛАБОРАТОРНАЯ РАБОТА №4**

**«СТРУКТУРЫ ДАННЫХ, ОСНОВАННЫЕ НА ХЕШ-ТАБЛИЦАХ»**

**Цель работы**

Исследовать возможности применения нелинейных структур данных – хеш-таблиц для хранения и обработки информации. Приобрести практические навыки использования хеш-таблиц для реализации быстрого доступа к данным. Произвести оценку эффективности использования хеш-таблиц для организации хранения данных.

**Вариант задания**

Вариант 8

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Ключевое поле | N1 | N2 | N3 | N4 | N5 |
| Абонент | 30 | 500 | 1900 | 4000 | 7500 |

Хеш-функция: B) Третья и последняя цифры ключа (например, для ключа 48793 получим значение хеш-функции 73);

* 1. **Ход выполнения работы**

В ходе работы была написана программа на языке C++ содержащая класс, который осуществляет работу с хеш-таблицей.

Для класса были определены методы добавления, поиска, удаления и изменения записей.

В таблице 1 отражено время выполнения операций при разном количестве элементов в хеш-таблице.

Таблица 1 - Результаты проделанной работы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Время N | № | Хеш-таблица | | | |
| Добавление | Удаление | Поиск | Изменение |
| N1 = 30 | 1 | 0,049 | 0,032 | 0,031 | 0,061 |
| 2 | 0,045 | 0,021 | 0,022 | 0,060 |
| 3 | 0,043 | 0,021 | 0,021 | 0,060 |
| 4 | 0,044 | 0,021 | 0,022 | 0,060 |
| 5 | 0,043 | 0,022 | 0,021 | 0,059 |
| Ср. | 0,045 | 0,023 | 0,023 | 0,060 |
| N2 = 500 | 1 | 0,053 | 0,020 | 0,050 | 0,077 |
| 2 | 0,040 | 0,019 | 0,020 | 0,078 |
| 3 | 0,042 | 0,020 | 0,020 | 0,075 |
| 4 | 0,042 | 0,020 | 0,023 | 0,055 |
| 5 | 0,042 | 0,021 | 0,020 | 0,075 |
| Ср. | 0,044 | 0,026 | 0,020 | 0,072 |
| N3 = 1900 | 1 | 0,058 | 0,041 | 0,023 | 0,081 |
| 2 | 0,043 | 0,021 | 0,022 | 0,059 |
| 3 | 0,047 | 0,021 | 0,021 | 0,077 |
| 4 | 0,041 | 0,021 | 0,021 | 0,057 |
| 5 | 0,041 | 0,021 | 0,020 | 0,077 |
| Ср. | 0,046 | 0,022 | 0,025 | 0,070 |
| N4 = 4000 | 1 | 0,046 | 0,046 | 0,028 | 0,090 |
| 2 | 0,039 | 0,027 | 0,030 | 0,070 |
| 3 | 0,040 | 0,026 | 0,030 | 0,065 |
| 4 | 0,039 | 0,046 | 0,047 | 0,062 |
| 5 | 0,045 | 0,030 | 0,048 | 0,094 |
| Ср. | 0,041 | 0,037 | 0,035 | 0,076 |
| N5 = 7500 | 1 | 0,044 | 0,049 | 0,047 | 0,105 |
| 2 | 0,042 | 0,032 | 0,034 | 0,097 |
| 3 | 0,063 | 0,032 | 0,052 | 0,076 |
| 4 | 0,063 | 0,053 | 0,060 | 0,086 |
| 5 | 0,059 | 0,048 | 0,029 | 0,071 |
| Ср. | 0,054 | 0,045 | 0,043 | 0,087 |

Было проведено добавление 10 элементов в хеш-таблицу. Результат изображен на рисунке 1.

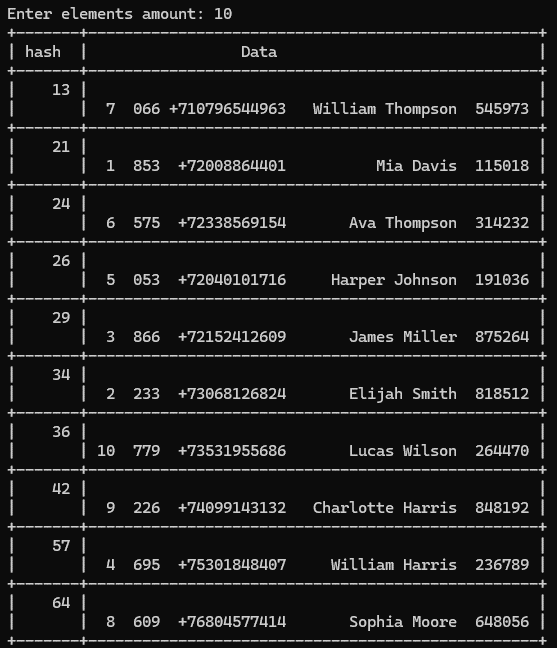


Рисунок 1 – Добавление 10 записей в хеш-таблицу

Как видно, при таком количестве элементов хэш-таблица работает максимально эффективно, так как в ней отсутствуют коллизии.

Добавим 20 элементов в хеш-таблицу (рисунок 2).



Рисунок 2 – Первые записи хэш-таблицы с 20 элементами

В этот раз можно наблюдать появление коллизий из-за недостаточно эффективной хеш-функции. Можно сделать вывод, что на больших количествах данных коллизий будет больше. Это очень сильно влияет на производительность хеш-таблицы.

Для того, чтобы избежать коллизий используется метод цепочек. В каждой ячейке таблице храниться список элементов, для которых хеш-функция возвращает одинаковые значения.

**Вывод**

В ходе выполнения лабораторной работы была написана программа, реализующая создание хеш-таблицы и выводящая ее содержимое со временем выполнения операций. Был применен один из методов разрешения коллизий.

**ПРИЛОЖЕНИЕ А**

Листинг 1 - Текст программы с реализацией класса для работы с хеш-таблицей.

#include <iostream>

#include <windows.h>

#include <fstream>

#include <random>

#include <ctime>

#include <vector>

#include <chrono>

#include <iomanip>

#include <list>

#include <string>

using namespace std;

struct Data {

int ip;

string subscriber;

string phoneNumber;

string name;

int amount;

};

string generateRandomPhoneNumber() {

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> dis(0, 9);

string phoneNumber = "";

for (int i = 0; i < 10; i++) {

if (i == 0) {

phoneNumber += to\_string(dis(gen) + 1);

}

else {

phoneNumber += to\_string(dis(gen));

}

}

return "+7" + phoneNumber;

};

vector<string> firstNames = {

"Liam", "Emma", "Noah", "Olivia", "William", "Ava", "James", "Isabella",

"Oliver", "Sophia", "Benjamin", "Mia", "Elijah", "Charlotte", "Lucas", "Harper"

};

vector<string> lastNames = {

"Smith", "Johnson", "Williams", "Brown", "Davis", "Miller", "Wilson", "Moore",

"Taylor", "Anderson", "Thomas", "Jackson", "White", "Harris", "Martin", "Thompson"

};

string generateRandomName() {

static random\_device rd;

static mt19937 gen(rd());

static uniform\_int\_distribution<> firstNameDist(0, firstNames.size() - 1);

static uniform\_int\_distribution<> lastNameDist(0, lastNames.size() - 1);

string firstName = firstNames[firstNameDist(gen)];

string lastName = lastNames[lastNameDist(gen)];

return firstName + " " + lastName;

}

int generateRandomNumber() {

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> dis(100000, 999999);

int randomNumber = dis(gen);

return randomNumber;

}

string generateRandomString() {

static random\_device rd;

static mt19937 gen(rd());

static uniform\_int\_distribution<> charDist(48, 57);

string randomString;

for (int i = 0; i < 3; i++) {

randomString += static\_cast<char>(charDist(gen));

}

return randomString;

}

Data generateRandomData(int i) {

Data data;

data.ip = i + 1;

data.subscriber = generateRandomString();

data.phoneNumber = generateRandomPhoneNumber();

data.name = generateRandomName();

data.amount = generateRandomNumber();

return data;

}

class HashTable {

private:

const int SIZE = 100;

vector<list<Data>> table;

public:

HashTable() : table(SIZE) {}

int hash(const string& key) {

int third\_digit = key[2] - '0';

int last\_digit = key[key.length() - 1] - '0';

return third\_digit \* 10 + last\_digit;

}

void insert(const Data& data) {

int index = hash(data.phoneNumber);

table[index].push\_back(data);

}

bool remove(const string& phoneNumber) {

int index = hash(phoneNumber);

auto it = table[index].begin();

while (it != table[index].end()) {

if (it->phoneNumber == phoneNumber) {

table[index].erase(it);

return true;

}

++it;

}

return false;

}

bool update(const string& phoneNumber, const Data& newData) {

int index = hash(phoneNumber);

auto it = table[index].begin();

while (it != table[index].end()) {

if (it->phoneNumber == phoneNumber) {

\*it = newData;

return true;

}

++it;

}

return false;

}

Data\* find(const string& phoneNumber) {

int index = hash(phoneNumber);

auto it = table[index].begin();

while (it != table[index].end()) {

if (it->phoneNumber == phoneNumber) {

return &(\*it);

}

++it;

}

return nullptr;

}

void printTable() {

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

cout << "| hash | Data |" << endl;

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

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

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

cout << "|" << right << setw(6) << i << " | |" << endl;

for (const auto& data : table[i]) {

cout << "| |" << setw(3) << data.ip << " " << setw(4) << data.subscriber

<< " " << setw(13) << data.phoneNumber << " " << setw(18) << data.name

<< " " << setw(7) << data.amount << " |" << endl;

}

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

}

}

}

void printTimeTable() {

double addTime = 0.0, deleteTime = 0.0, searchTime = 0.0, editTime = 0.0;

auto start = chrono::high\_resolution\_clock::now();

auto end = chrono::high\_resolution\_clock::now();

ofstream out("result.txt", ios::app);

for (int i = 0; i < 5; i++) {

auto start = chrono::high\_resolution\_clock::now();

insert(generateRandomData(i));

auto end = chrono::high\_resolution\_clock::now();

double oneElementAddTime = chrono::duration<double, milli>(end - start).count();

cout << "Addition #" << i << ": " << oneElementAddTime << " ms" << endl;

addTime += oneElementAddTime;

out << oneElementAddTime << " ";

start = chrono::high\_resolution\_clock::now();

remove(generateRandomPhoneNumber());

end = chrono::high\_resolution\_clock::now();

double oneElementDeleteTime = chrono::duration<double, milli>(end - start).count();

cout << "Deletion #" << i << ": " << oneElementDeleteTime << " ms" << endl;

deleteTime += oneElementDeleteTime;

out << oneElementDeleteTime << " ";

start = chrono::high\_resolution\_clock::now();

find(generateRandomPhoneNumber());

end = chrono::high\_resolution\_clock::now();

double oneElementSearchTime = chrono::duration<double, milli>(end - start).count();

cout << "Search #" << i << ": " << oneElementSearchTime << " ms" << endl;

searchTime += oneElementSearchTime;

out << oneElementSearchTime << " ";

start = chrono::high\_resolution\_clock::now();

update(generateRandomPhoneNumber(), generateRandomData(0));

end = chrono::high\_resolution\_clock::now();

double oneElementEditTime = chrono::duration<double, milli>(end - start).count();

cout << "Update #" << i << ": " << oneElementEditTime << " ms" << endl;

editTime += oneElementEditTime;

out << oneElementEditTime << " ";

out << endl;

}

out << addTime / 5 << " " << searchTime / 5 << " " << deleteTime / 5 << " " << editTime / 5 << endl;

cout << "Avg. operations time:" << endl;

cout << "| Add | " << fixed << setprecision(3) << setw(12) << addTime / 5 << " |" << endl;

cout << "| Search | " << fixed << setprecision(3) << setw(12) << searchTime / 5 << " |" << endl;

cout << "| Delete | " << fixed << setprecision(3) << setw(12) << deleteTime / 5 << " |" << endl;

cout << "| Update | " << fixed << setprecision(3) << setw(12) << editTime / 5 << " |" << endl;

}

};

void createHashTable(HashTable& hashTable) {

int elementsAmount;

cout << "Enter elements amount: ";

cin >> elementsAmount;

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

hashTable.insert(generateRandomData(i));

}

}

int main() {

SetConsoleOutputCP(65001);

HashTable hashTable;

createHashTable(hashTable);

hashTable.printTable();

hashTable.printTimeTable();

system("pause");

return 0;

}