МИНОБРНАУКИ РОСИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Восточно-Сибирский государственный университет технологий и управления»

(ФГБОУ ВО ВСГУТУ)

Факультет компьютерных наук и технологий

Кафедра программной инженерии и искусственного интеллекта

**ОТЧЕТ**по лабораторной работе №5  
на тему «Методы хэширования»

по дисциплине «Структуры и алгоритмы обработки данных»

Выполнил: студент гр. Б763-2 Угнивенко Д.А.

Проверил: преподаватель Николаева Т. В.

Улан-Удэ

2024

1. **Листинг программы:**

#include <iostream>

#include <vector>

#include <list>

#include <random>

#include <ctime>

using namespace std;

class HashTable {

private:

int tableSize;

vector<list<int>> table;

public:

HashTable(int size) : tableSize(size) {

table.resize(tableSize);

}

int hashFunction(int key) {

return key % tableSize;

}

void insert(int key) {

if (key >= 1000 && key <= 9999) {

int index = hashFunction(key);

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

cout << "Коллизия найдена: " << key << " индекс: " << index << endl;

}

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

}

else {

cout << "Ошибка число не 4х разрядное!" << endl;

}

}

void display() {

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

cout << i << ": ";

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

cout << num << " ";

}

cout << endl;

}

}

};

int main() {

setlocale(LC\_ALL, "Russian");

HashTable ht(10);

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(1000, 9999);

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

int randomNumber = distrib(gen);

int index = ht.hashFunction(randomNumber);

cout << "Ключ: " << randomNumber << ", Хэш значение: " << index << endl;

ht.insert(randomNumber);

}

cout << endl << "Хэш-таблица:" << endl;

ht.display();

return 0;

}

Первая программа: Хэш-таблица с прямой адресацией и обработкой коллизий   
#include <iostream>

#include <vector>

#include <list>

#include <random>

#include <ctime>

using namespace std;

// Класс для реализации хэш-таблицы

class HashTable {

private:

int tableSize; // Размер хэш-таблицы

vector<list<int>> table; // Вектор из списков для хранения значений в каждом хэш-индексе (списки помогают обрабатывать коллизии)

public:

// Конструктор, инициализирующий размер таблицы и создающий пустые списки

HashTable(int size) : tableSize(size) {

table.resize(tableSize); // Устанавливаем размер таблицы

}

// Хэш-функция для вычисления индекса (остаток от деления ключа на размер таблицы)

int hashFunction(int key) {

return key % tableSize;

}

// Метод для вставки ключа в хэш-таблицу

void insert(int key) {

if (key >= 1000 && key <= 9999) { // Проверка, является ли число 4-значным

int index = hashFunction(key); // Вычисляем хэш-индекс

if (!table[index].empty()) { // Проверка на наличие коллизии

cout << "Коллизия найдена: " << key << " индекс: " << index << endl;

}

table[index].push\_back(key); // Добавляем ключ в соответствующий список

}

else {

cout << "Ошибка число не 4х разрядное!" << endl;

}

}

// Метод для отображения содержимого хэш-таблицы

void display() {

for (int i = 0; i < tableSize; i++) { // Перебираем все индексы таблицы

cout << i << ": ";

for (const auto& num : table[i]) { // Перебираем все элементы в списке текущего индекса

cout << num << " ";

}

cout << endl;

}

}

};

int main() {

setlocale(LC\_ALL, "Russian"); // Включаем поддержку русского языка в консоли

HashTable ht(10); // Создаем хэш-таблицу с 10 индексами

// Настройка генератора случайных чисел

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(1000, 9999); // Генерируем случайные числа от 1000 до 9999

// Генерация и вставка 10 случайных чисел в хэш-таблицу

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

int randomNumber = distrib(gen); // Генерируем случайное число

int index = ht.hashFunction(randomNumber); // Вычисляем его хэш-индекс

cout << "Ключ: " << randomNumber << ", Хэш значение: " << index << endl;

ht.insert(randomNumber); // Вставляем число в хэш-таблицу

}

cout << endl << "Хэш-таблица:" << endl;

ht.display(); // Отображаем содержимое хэш-таблицы

return 0;

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

#include <iostream>

#include <cmath>

#include <string>

#include <random>

#include <vector>

#include <unordered\_map>

using namespace std;

int hashfunc(int key) {

long long squared = static\_cast<long long>(key) \* key;

string squaredStr = to\_string(squared);

int len = squaredStr.length();

int start = (len - 2) / 2;

string middlePart = squaredStr.substr(start, 2);

return std::stoi(middlePart);

}

int main() {

setlocale(LC\_ALL, "Russian");

const int num = 10;

vector<int> randomKeys(num);

vector<int> hashValues(num);

unordered\_map<int, int> hashsave;

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(1000, 9999);

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

randomKeys[i] = distrib(gen);

hashValues[i] = hashfunc(randomKeys[i]);

if (hashsave.find(hashValues[i]) != hashsave.end()){

cout << "Коллизия найдена: Ключ: " << randomKeys[i] << " - Хэш: " << hashValues[i] << endl;

}

else {

hashsave[hashValues[i]] = randomKeys[i];

}

}

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

cout << "Ключ: " << randomKeys[i] << " -> Хэш: " << hashValues[i] << endl;

}

return 0;

}  
Вторая программа: Метод середины квадрата для хэширования   
#include <iostream>

#include <cmath>

#include <string>

#include <random>

#include <vector>

#include <unordered\_map>

using namespace std;

// Хэш-функция, использующая метод середины квадрата

int hashfunc(int key) {

long long squared = static\_cast<long long>(key) \* key; // Возводим ключ в квадрат

string squaredStr = to\_string(squared); // Преобразуем результат в строку

int len = squaredStr.length(); // Определяем длину строки

int start = (len - 2) / 2; // Находим позицию для выделения двух центральных символов

string middlePart = squaredStr.substr(start, 2); // Извлекаем 2 центральные цифры

return std::stoi(middlePart); // Преобразуем строку обратно в число

}

int main() {

setlocale(LC\_ALL, "Russian"); // Включаем поддержку русского языка

const int num = 10; // Количество случайных чисел

vector<int> randomKeys(num); // Вектор для хранения случайных чисел

vector<int> hashValues(num); // Вектор для хранения хэш-значений

unordered\_map<int, int> hashsave; // Словарь для отслеживания коллизий (хэш -> ключ)

// Генератор случайных чисел

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(1000, 9999); // Генерация 4-значных чисел

// Генерация ключей и их хэширование

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

randomKeys[i] = distrib(gen); // Генерация случайного ключа

hashValues[i] = hashfunc(randomKeys[i]); // Получаем хэш-значение

// Проверка на коллизию

if (hashsave.find(hashValues[i]) != hashsave.end()) {

cout << "Коллизия найдена: Ключ: " << randomKeys[i] << " - Хэш: " << hashValues[i] << endl;

}

else {

hashsave[hashValues[i]] = randomKeys[i]; // Сохраняем пару хэш -> ключ

}

}

// Вывод результатов

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

cout << "Ключ: " << randomKeys[i] << " -> Хэш: " << hashValues[i] << endl;

}

return 0;

}

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

#include <iostream>

#include <vector>

#include <unordered\_map>

#include <random>

using namespace std;

unsigned int hashfunc(unsigned int num) {

unsigned int part1 = num / 100;

unsigned int part2 = num % 100;

unsigned int hashValue = part1 + part2;

const unsigned int MAX\_HASH\_VALUE = 100;

return hashValue % MAX\_HASH\_VALUE;

}

int main() {

setlocale(LC\_ALL, "Russian");

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(1000, 9999);

vector<unsigned int> nums;

unordered\_map<unsigned int, vector<unsigned int>> hashsave;

for (int i = 0; i <50; i++) {//генерация и заполнение

unsigned int randNum = distrib(gen);

nums.push\_back(randNum);

unsigned int hashValue = hashfunc(randNum);

hashsave[hashValue].push\_back(randNum);

}

cout << "Число\tХэш-значение" << endl;

for (unsigned int num : nums) {

cout << num << "\t" << hashfunc(num) << endl;

}

cout << "\nПроверка коллизий: " << endl;

for (const auto& entry : hashsave) { //entry - это пара хэш-ключ и вектор чисел

if (entry.second.size() > 1) { //entry.second - вектор чисел, которые имеют одно и тоже значение

cout << "Коллизия в хэш-значении " << entry.first << " с числами: ";

for (unsigned int num : entry.second) {

cout << num << " ";

}

cout << endl;

}

}

return 0;

}  
 **Третья программа: Хэш-функция на основе деления и сложения частей числа   
#include** <iostream>

#include <vector>

#include <unordered\_map>

#include <random>

using namespace std;

// Хэш-функция, которая делит число на две части и суммирует их

unsigned int hashfunc(unsigned int num) {

unsigned int part1 = num / 100; // Первая часть числа (первая половина)

unsigned int part2 = num % 100; // Вторая часть числа (вторая половина)

unsigned int hashValue = part1 + part2; // Сумма двух частей

const unsigned int MAX\_HASH\_VALUE = 100; // Ограничение на хэш-значение

return hashValue % MAX\_HASH\_VALUE; // Возвращаем остаток от деления

}

int main() {

setlocale(LC\_ALL, "Russian");

random\_device rd;

mt19937 gen(rd());

uniform\_int\_distribution<> distrib(1000, 9999); // Генерация 4-значных чисел

vector<unsigned int> nums; // Список чисел

unordered\_map<unsigned int, vector<unsigned int>> hashsave; // Хранилище для проверки коллизий

// Генерация чисел и их хэширование

for (int i = 0; i < 50; i++) {

unsigned int randNum = distrib(gen);

nums.push\_back(randNum);

unsigned int hashValue = hashfunc(randNum);

hashsave[hashValue].push\_back(randNum);

}

// Вывод чисел и их хэш-значений

cout << "Число\tХэш-значение" << endl;

for (unsigned int num : nums) {

cout << num << "\t" << hashfunc(num) << endl;

}

// Проверка на наличие коллизий

cout << "\nПроверка коллизий: " << endl;

for (const auto& entry : hashsave) {

if (entry.second.size() > 1) {

cout << "Коллизия в хэш-значении " << entry.first << " с числами: ";

for (unsigned int num : entry.second) {

cout << num << " ";

}

cout << endl;

}

}

return 0;

}

**2.Тестирование:**

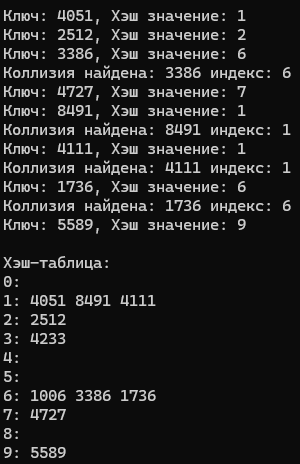


Рисунок 2 – Результаты метода деление по модулю T

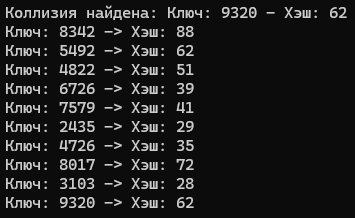


Рисунок 2 – Результаты метода средняя часть квадрата

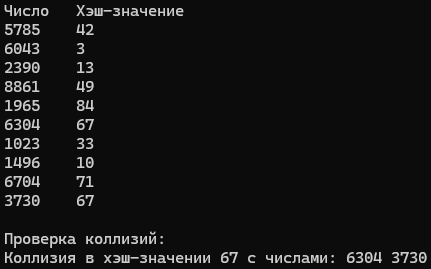


Рисунок 3 – Результаты метода разбиения на части и сложения

**3. Вычислительный эксперимент**

Для определения практической сложности алгоритмов сортировки были проведены вычислительные эксперименты.

Рисунок 4 – График зависимости времени от объема выборки

При малом количестве ключей (N = 10), эффективным методом является метод «Разбиение на части и сложение», так как было в результате хэширования не было коллизий. Но при увеличении количество ключей, он становится эффективнее метода «Деление по модулю» в 2,4 раза и менее эффективнее, чем метод «СЧК» в 1,25 раз.

Метод «Деление по модулю» является не эффективным, так как при малом количестве элементов, количество коллизий больше в 5 раз, чем при методе «Разбиение на части и сложение». При количестве ключей равному 100, метод менее эффективнее, чем метод «Разбиение на части и сложение» в 3 раза.

**Метод «Разбиение на части и сложение» является наиболее эффективным при любом количестве элементов.**

**Заключение**

В ходе лабораторной работы я изучил 3 метода хэширования.