Институт приборостроения, автоматизации и информационных технологий

Кафедра информационных систем и цифровых технологий

Дисциплина «Алгоритмы и структуры данных»

Отчет к лабораторной работе № 6

«Анализ методов разрешения коллизий при хешировании»

Выполнил:

Василения Иван Валерьевич

Мельниченко Артём Олегович

Принял:

Рыженков Д. В.

Орёл, 2024г

Листинг файла main.cpp

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <cmath> // floor

#include <fstream>   // Для работы с файлами

using *namespace* std;

*int* r[6];

string\* keys;

*int*\* hashMap;

*int* mapSize = 1000;

*int* keysAmount = 400;

*int* iterationLimit = 100;

string getKey() {

*char* symbols[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";

    const *int* size = 62;

    string key = "";

    for (*int* i = 0; i < 6; i++)

    {

        key += symbols[rand() % size];

    }

    return key;

}

*void* clearHashMap(*int* *map*[], *int* *size*) {

    for (*int* i = 0; i < size; i++)

    {

        map[i] = 0;

    }

}

*void* getR(*int* *arr*[], *int* *size*) {

    for (*int* i = 0; i < size; i++)

    {

        arr[i] = rand() % 256;

    }

}

*int* xorMethod(string *key*, *int* *r*[]) {

*int* sum = 0;

    for (*int* i = 0; i < key.length(); i++)

    {

        sum += (*int*)key[i] ^ r[i];

    }

    return sum;

}

*int* divideHashMethod(*int* *M*, *int* *k*) {

    return k % M;

}

*int* multiplyMethod(*int* *M*, *int* *k*) {

*float* kA = 0.6180339887 \* k;

    return M \* (kA - floor(kA));

}

*int* linearMethod(string *key*, *int* (\**hashMethod*)(*int*, *int*), *int* *hashParam*, *int* *c*) {

*int* k = xorMethod(key, r);

*int* iterationCntr = 0;

    while (iterationCntr < iterationLimit)

    {

*int* address = (hashMethod(hashParam, k) + (c \* iterationCntr++)) % mapSize;

        if (hashMap[address] == 0) {

            hashMap[address]++;

            return iterationCntr;

        }

    }

    return -1;

}

*int* squareMethod(string *key*, *int* (\**hashMethod*)(*int*, *int*), *int* *hashParam*, *int* *c*, *int* *d*) {

*int* k = xorMethod(key, r);

*int* iterationCntr = 0;

    while (iterationCntr < iterationLimit)

    {

*int* address = (hashMethod(hashParam, k) + (c \* iterationCntr) + (d \* iterationCntr \* iterationCntr)) % mapSize;

        iterationCntr++;

        if (hashMap[address] == 0) {

            hashMap[address]++;

            return iterationCntr;

        }

    }

    return -1;

}

*double* averageLD() {

*double* sum = 0;

    for (*int* i = 0; i < keysAmount; i++)

    {

        sum += linearMethod(keys[i], divideHashMethod, mapSize / 3, 139);

    }

    clearHashMap(hashMap, mapSize);

    return sum / keysAmount;

}

*double* averageLM() {

*double* sum = 0;

    for (*int* i = 0; i < keysAmount; i++)

    {

        sum += linearMethod(keys[i], multiplyMethod, mapSize, 139);

    }

    clearHashMap(hashMap, mapSize);

    return sum / keysAmount;

}

*double* averageSD() {

*double* sum = 0;

    for (*int* i = 0; i < keysAmount; i++)

    {

        sum += squareMethod(keys[i], divideHashMethod, mapSize / 3, 577, 14);

    }

    clearHashMap(hashMap, mapSize);

    return sum / keysAmount;

}

*double* averageSM() {

*double* sum = 0;

    for (*int* i = 0; i < keysAmount; i++)

    {

        sum += squareMethod(keys[i], multiplyMethod, mapSize, 577, 14);

    }

    clearHashMap(hashMap, mapSize);

    return sum / keysAmount;

}

*void* generateKeys() {

    for (*int* i = 0; i < keysAmount; i++)

    {

        keys[i] = getKey();

    }

}

*void* generate() {

    keys = new string[keysAmount];

    hashMap = new *int*[mapSize];

    getR(r, 6);

    generateKeys();

    clearHashMap(hashMap, mapSize);

    cout << "Средняя длина пути линейного метода + метода деления " << averageLD() << endl;

    cout << "Средняя длина пути линейного метода + метода умножения " << averageLM() << endl;

    cout << "Средняя длина пути квадратичного метода + метода деления " << averageSD() << endl;

    cout << "Средняя длина пути квадратичного метода + метода умножения " << averageSM() << endl;

    delete[] keys;

    delete[] hashMap;

}

*void* createAvg(string *fileName*, *double*(\**averageFunc*)()) {

    keys = new string[mapSize];

    hashMap = new *int*[mapSize];

    clearHashMap(hashMap, mapSize);

    ofstream outputFile(fileName);

    if (!outputFile) {

        std::cerr << "Не удалось открыть файл!" << std::endl;

        return;

    }

    for (keysAmount = 10; keysAmount < mapSize - 10; keysAmount += 10)

    {

        for (*int* i = 0; i < keysAmount; i++)

        {

            keys[i] = getKey();

        }

*double* avg = averageFunc();

        if (avg == -1) { continue; }

        cout << "AVG: " << avg << " keysAmount: " << keysAmount << endl;

        outputFile << keysAmount << ";" << avg << endl;

    }

    outputFile.close();

    delete[] keys;

    delete[] hashMap;

}

*int* getValidNumber() {

*int* number;

    while (true) {

        cin >> number;

        if (cin.fail()) {

            cin.clear();

            cin.ignore(numeric\_limits<streamsize>::max(), '\n');

            cout << "Ошибка: введите корректное целое число!" << endl;

        }

        else {

            cin.ignore(numeric\_limits<streamsize>::max(), '\n');

            break;

        }

    }

    return number;

}

*int* main() {

    srand(static\_cast<*unsigned* *int*>(std::time(0))); // Инициализация генератора случайных чисел

*int* temp;

    // Устанавливаем локаль для вывода с запятой вместо точки

    std::locale::global(std::locale("ru\_RU.UTF-8"));

    std::cout.imbue(std::locale());

    setlocale(LC\_ALL, "RUS");

    while (true)

    {

        cout << "Размер таблицы: " << mapSize;

        cout << " Количество ключей: " << keysAmount << endl;

        cout << "1. Ввести размер хеш таблицы" << endl;

        cout << "2. Ввести количество ключей" << endl;

        cout << "3. Смоделировать хеширование" << endl;

        cout << "0. Выход" << endl;

*char* c;

        cin >> c;

        switch (c)

        {

        case '1':

            cout << "Введите размер" << endl;

            temp = getValidNumber();

            if (keysAmount > temp) {

                cout << "Невалидный размер хеш таблицы" << endl;

                break;

            }

            mapSize = temp;

            break;

        case '2':

            cout << "Введите число ключей" << endl;

            temp = getValidNumber();

            if (mapSize < temp) {

                cout << "Невалидное количество ключей" << endl;

                break;

            }

            keysAmount = temp;

            break;

        case '3':

            generate();

            break;

        case '0':

            return 0;

        default:

            cout << "Неизвестная команда\n";

            break;

        }

    }

}