МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структуры данных»

Тема: Исследование хэш-таблицы с цепочками

Студент гр. 9303 _____ Павлов Д.Р.

Преподаватель _____ Филатов А.Ю

Санкт-Петербург

2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Павлов Д.Р.

Группа 9303

Тема работы: исследование хеш-таблицы с цепочками

Исходные данные:
Написать программу для создания структур данных, обработки и генерации входных данных, использовать их для измерения количественных характеристик хеш-таблицы с цепочками, сравнить экспериментальные результаты с теоретическими.

Содержание пояснительной записки: «Содержание», «Введение», «Описание структур данных», «Описание алгоритма», «Тестирование», «Исследование», «Исходный код», «Заключение», «Использованные источники»

Предполагаемый объем пояснительной записки:
Не менее 10 страниц.

Дата выдачи задания: 6.11.2020

Дата сдачи реферата: 25.12.2020

Дата защиты реферата: 25.12.2020

Студент _____ Павлов Д.Р.

Преподаватель _____ Филатов А.Ю.

АННОТАЦИЯ

В данной работе была создана программа на языке программирования C++ для генерации, обработки данных и вывода результатов: экспериментальных характеристик хеш-таблицы с цепочками. Результаты были проанализированы и приведены в данном отчёте.

SUMMARY

In this work, a program was created in the C ++ programming language for generating, processing data, and outputting results: experimental characteristics of a hash table with chains. The results were analyzed and presented in this report.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В данной курсовой работе реализована структура данных “словарь” на основе хеш-таблицы с цепочечным методом разрешения коллизий, ключом является строка и для получения его хеш-значения используется метод хеширования путём взятия остатка.

ЦЕЛЬ РАБОТЫ

Реализация и экспериментальное машинное исследование алгоритма вставки в хеш-таблицу с цепочками.

1. ОПИСАНИЕ СТРУКТУР ДАННЫХ

Хеш-табли́ца - это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

Хеш-таблица содержит некоторый массив, элементы которого списки пар (хеш-таблица с цепочками).

Пример хеш-таблицы с цепочками приведён на рис. 1.

Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Получающееся хеш-значение играет роль индекса в массиве. Затем выполняемая операция (добавление, удаление или поиск) перенаправляется объекту, который хранится в соответствующей ячейке.

Ситуация, когда для различных ключей получается одно и то же хеш-значение, называется коллизией. Механизм разрешения коллизий — важная составляющая любой хеш-таблицы.

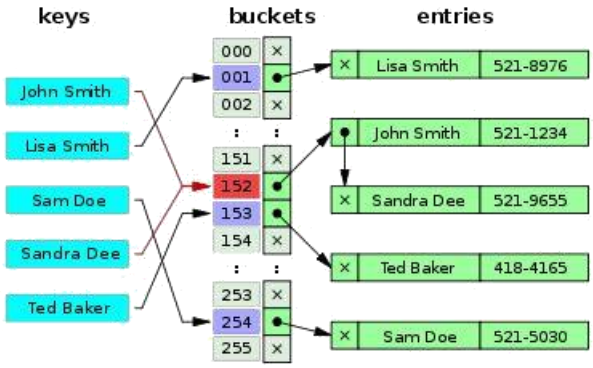

Рисунок 1. Хеш-таблица с цепочками.

2. ОПИСАНИЕ АЛГОРИТМОВ

2.1. Операции удаления, добавления элемента:

Вставка элемента в хеш-таблицу происходит путём вычисления хеш-значения ключа и последующего добавления элемента в конец списка, находящегося в ячейке массива с номером равным хеш-значению.

При удалении элемента сначала осуществляется поиск элемента в хеш-таблице как описано выше, и, в случае, если элемент найден, происходит его удаление из списка, находящегося в ячейке массива с номером равным хеш-значению ключа.

2.2. Генерация входных и получение выходных данных

Данные генерируются с увеличением числа элементов на степень двойки начиная с 512 семь раз. С помощью стандартной 32 разрядной версии вихря Мерсенна – средние случаи и по формуле (k + 1) * 2 ^ i, при этом 2 ^ i должно быть больше генерируемого количества элементов, - худшие случаи. Худшим является случай, когда все элементы добавляются в список, находящийся в одной и той же ячейке массива. Для анализа используется среднее время вставки элемента. Среднее время выводится в секундах.

3. ОПИСАНИЕ КЛАССОВ

3.1 HashTable

```
intgetHash(int key) const;
```

Вычисляет хеш ключа методом взятия остатка от деления.

```
void insertElement(int key);
```

Вставляет элемент в хеш-таблицу.

```
void removeElement(const long long int & key);
```

Удаляет элемент по ключу.

4. ТЕСТИРОВАНИЕ

Результат запуска программы:

```
Elements - 512 Average case insertion - 0.000001 sec
Operation Counts - 1
Elements - 512 Bad case insertion - 0.000006 sec Average
operation counts - 256

Elements - 1024 Average case insertion - 0.000001 sec Average
Operation Counts - 1
Elements - 1024 Bad case insertion - 0.000010 sec Average
operation counts - 512

Elements - 2048 Average case insertion - 0.000001 sec Average
Operation Counts - 1
Elements - 2048 Bad case insertion - 0.000019 sec Average
operation counts - 1024

Elements - 4096 Average case insertion - 0.000001 sec Average
Operation Counts - 1
Elements - 4096 Bad case insertion - 0.000040 sec Average
operation counts - 2048

Elements - 8192 Average case insertion - 0.000001 sec Average
Operation Counts - 1
Elements - 8192 Bad case insertion - 0.000075 sec Average
operation counts - 4096

Elements - 16384 Average case insertion - 0.000001 sec
Average Operation Counts - 1
Elements - 16384 Bad case insertion - 0.000147 sec Average
operation counts - 8192

Elements - 32768 Average case insertion - 0.000001 sec
Average Operation Counts - 1
Elements - 32768 Bad case insertion - 0.000281 sec Average
operation counts - 16384
```

5. ИССЛЕДОВАНИЕ

Составим таблицу результатов эксперимента и построим графики зависимости среднего времени вставки от их количества(табл. 1):

Количество вставок	Среднее время одной вставки(сек)		Отношение текущего и предыдущего времени вставки	Среднее количество базовых операций	
	Средний случай	Худший случай		Средний случай	Худший случай
512	0.000001	0.000006	-	1	256
1024	0.000001	0.000010	1.67	1	512
2048	0.000001	0.000019	1.9	1	1024
4096	0.000001	0.000040	2.10	1	2048
8192	0.000001	0.000075	1.875	1	4096
16384	0.000001	0.000147	1.96	1	8192
32768	0.000001	0.000281	1.91	1	16384

Таблица 1. Результаты эксперимента.

В теории в среднем случае время затраченное на вставку не зависит напрямую от количества элементов. В проведённом эксперименте на случайном наборе данных так же отсутствует какая-либо корелляция между количеством вставок и средним временем вставки, что можно увидеть в табл. 1.

В худшем же случае, согласно теории, время затраченное на вставку линейно растёт при увеличении количества вставляемых элементов. Обращаясь к табл. 1, при увеличении количества вставок в 2 раза, среднее

время требуемое на 1 вставку так же увеличивается в среднем в 2.0014 раза, что соответствует теории. Графики зависимостей смотри на рис. 2

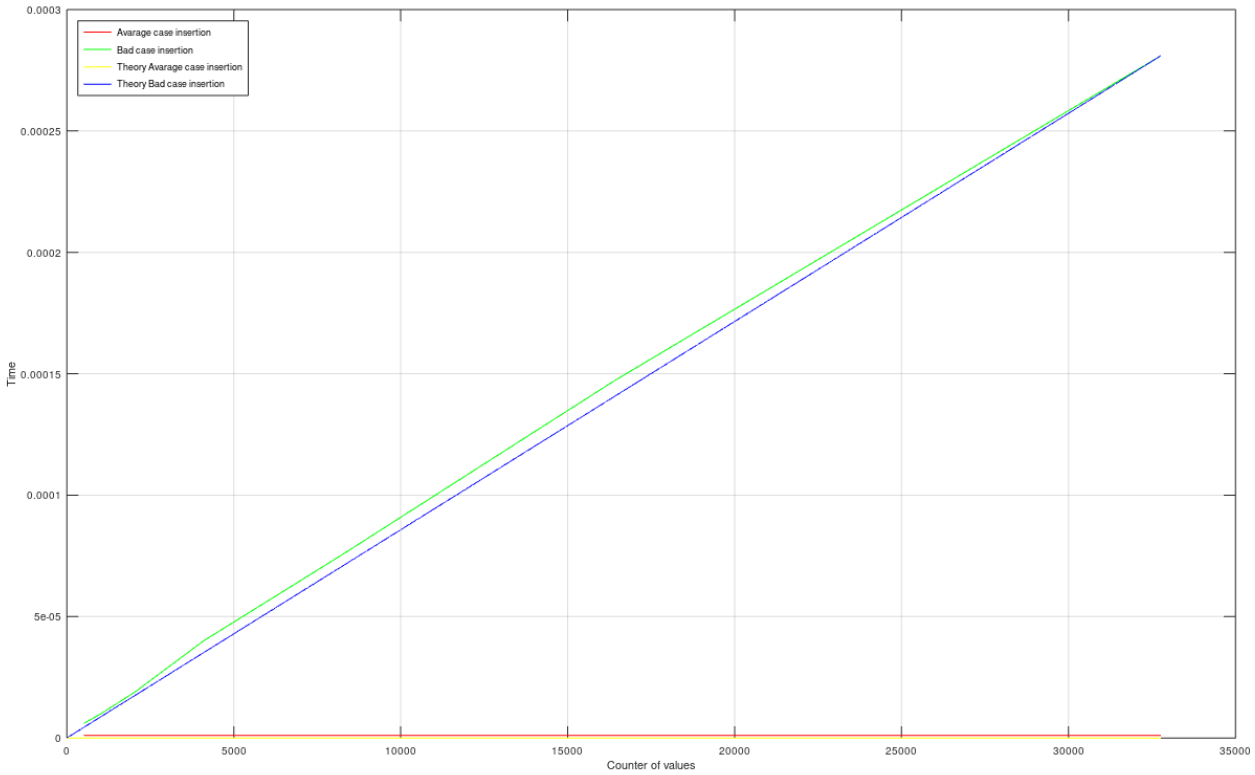

Рисунок 2. Теоретическая и практическая зависимость времени вставки от количества элементов в среднем и в худшем случае.

ЗАКЛЮЧЕНИЕ

В ходе работы была написана программа для генерации данных, создания такой структуры данных как хеш-таблица с цепочками и методы для взаимодействия с ней. С помощью программы были найдены экспериментальные числовые характеристики хеш-таблицы при вставке, при исследовании которых были подтверждены теоретические данные.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. https://ru.wikipedia.org/wiki/Хеш-таблица
2. https://habr.com/ru/post/509220/

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл: HashTable.h

```
#ifndef COURSEWORK_HASHTABLE_H
#define COURSEWORK_HASHTABLE_H
#include <iostream>
#include <list>


using namespace std;

//Creating a hashtable class
class HashTable{
private:
    list<int> *table;
    int total_elements;

    [[nodiscard]] int getHash(int key) const{
        return key % total_elements;
    }

public:
    //Contructor
    explicit HashTable(int n){
        total_elements = n;
        table = new list<int>[total_elements];
    }
    //Insertion method
    int insertElement(int key){
        int operation_counter = 1;
        unsigned hash = getHash(abs(key));
        list<int>& valueList = table[hash];
        for(auto& elem : valueList){
            if (elem == key){
                elem = key;
                return operation_counter++;
            }
            operation_counter++;
        }
        int newElem = key;
        table[hash].push_back(newElem);
        return operation_counter;
    }
    //Removal method
    void removeElement(int key){
        int x = getHash(key);

        list<int>::iterator i;
        for (i = table[x].begin(); i != table[x].end(); i++) {
            if (*i == key)
                break;
        }


        if (i != table[x].end())
            table[x].erase(i);
        else{
            cout << "[WARNING] Key not found!\n";
```

```
        }

    }
    //Method to print all hash table
    void printAll(){
        for(int i = 0; i < total_elements; i++){
            cout << "Index " << i << ": ";
            for(int j : table[i]) {
                cout << j << " => ";
            }

            cout << endl;
        }
    }
    //Method to print list of values by key
    void printSearch(int key){
        for(int i = 0; i < total_elements; i++){
            if (i == key){
                for(int j : table[i]) {
                    cout << j << " => ";
                }
            }

        }
    }
    //Clear hash table values
    void clear(){
        delete [] table;
        table = new list<int>[this->total_elements];
    }
};


#endif //COURSEWORK_HASHTABLE_H
```

Файл: main.cpp

```
#include <iostream>
#include "HashTable.h"
#include <ctime>
#include <random>

using namespace std;


int main() {
    std::mt19937 rndGen(static_cast<unsigned>(time(nullptr))); // Generator
Mersenne twister
    int testCount = 512; // Start test data
    for (int i = 0; i < 7; ++i) {
        int operCounter = 0; // Operation counter
        HashTable avrHT(testCount);

        clock_t clocks;
        float time = 0;
        float avrTime;
        avrHT.clear();
        //
        for (int j = 0; j < testCount; ++j) {
            int value = static_cast<int>(rndGen()); // Generating a value
            clocks = clock();
            operCounter += avrHT.insertElement(value);
```

```
            clocks = clock() - clocks;
            time += static_cast<float>(clocks) / CLOCKS_PER_SEC;
            }

        if(i == 1){
            avrHT.printAll();
        }
        avrTime = time / (float )testCount;
        operCounter /= testCount;
        std::cout << "Elements - " << testCount << " Average case insertion - "
<<fixed<< avrTime << " sec " <<"Operation Counts - "<< operCounter << std::endl;
        time = 0;



        avrHT.clear();
        long long int base = 1;
        while (base <= testCount){
            base <<= 1;
        }
        operCounter = 0;
        for (int j = 0; j < testCount; ++j) {
            long long int key = (j+1)*base;
            clocks = clock();
            operCounter += avrHT.insertElement(key);
            clocks = clock() - clocks;
            time += static_cast<float>(clocks) / CLOCKS_PER_SEC;
        }



        avrTime = time / (float )testCount;
        operCounter /= testCount;
        std::cout << "Elements - " << testCount << " Bad case insertion - "
<<fixed<< avrTime << " sec" <<" Average operation counts - "<< operCounter <<
std::endl << std::endl;
        testCount *= 2;

    }
    return 0;
}
```