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 хеш-таблицы с открытой адресацией

Студент гр. 9303 _____ Скворчевский Б.С.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Скворчевский Б.С.

Группа 9303

Тема работы: Исследование хеш-таблицы с открытой адресацией

Исходные данные: Реализовать программу для создания и работы хеш-таблицой. Сгенерировать множество входных данных для среднего и худшего случая. Выполнить алгоритм вставки в хеш-таблицу полученных значений, зафиксировав время и количество произведённых базовых операций. Зафиксировать результаты испытаний для различного количества элементов хеш-таблицы. Сопоставить полученные данные с теоретическими оценками.

Содержание пояснительной записки:

- Содержание
- Аннотация
- Введение
- Описание алгоритма
- Описание структур данных и функций
- Тестирование
- Исследование
- Выводы
- Список использованных источников
- Исходный код

Дата выдачи задания: 06.11.2020

Дата сдачи реферата: 25.12.2020

Дата защиты реферата: 25.12.2020

Студент _____ Скворчевский Б.С.

Преподаватель _____ Филатов А.Ю.

АННОТАЦИЯ

Была реализована программа на языке программирования C++ для работы с хеш-таблицей с открытой адресацией, а именно: генерации входных данных, обработки и вывода результатов. Результаты представлены и проанализированы в данном отчёте.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель работы: реализация и экспериментальное машинное исследование хеш-таблицы с открытой адресацией.

Для достижения данной поставленной цели требуется решить следующие задачи

- Изучить алгоритм работы хеш-таблицы с открытой адресацией
- Реализовать методы для работы с хеш-таблицей
- Сгенерировать данные для тестирования
- Изучить полученные результаты

1. ОПИСАНИЕ АЛГОРИТМА

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

Хеш-таблица содержит некоторый массив, элементы которого есть пары ключ – значение.

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

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

Пример хеш-таблицы с открытой адресацией см. на рис. 1.

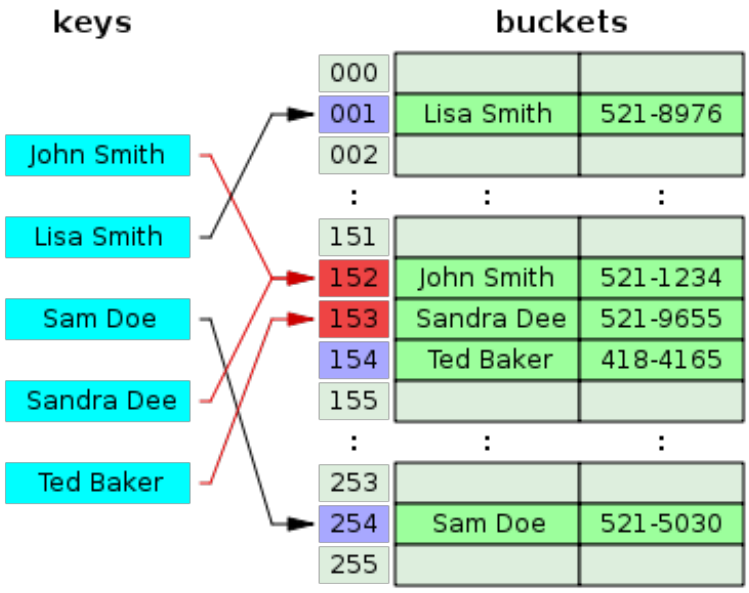

Рисунок 1 – Хеш-таблица с открытой адресацией

Для вставки элемента в хеш-таблицу вычисляется хеш-значение для ключа и элемент вставляется в соответствующий хешу элемент массива, хранящего хеш-таблицу. В случае возникновения коллизии элемент вставляется в ближайшую свободную ячейку массива.

Для удаления элемента из хеш-таблицы сначала вычисляется хеш-значение для ключа, после чего проверяется является ли найденный ключ искомым. В случае, если это так в ячейку массива в качестве ключа и значения заносится число -1, что означает пустую ячейку. Если же найденный элемент имеет иной ключ – производится дальнейший поиск нужного элемента массива.

2. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ФУНКЦИЙ

`class HashNode` – класс элемента хеш-таблицы с полями: V `value` – значение и K `key` – ключ.

`class HashMap` – класс хеш-таблицы с полями `HashNode<K,V> **arr` – массив, содержащий элементы хеш-таблицы, `capacity` – вместимость хеш-таблицы, `int size` – размер (количество элементов) хеш-таблицы, `HashNode<K,V> *undef` – неопределённый элемент хеш-таблицы и методами:

- `int hashCode(K key)` – Метод для получения хэша элемента таблицы по ключу
- `int insertNode(K key, V value)` – Метод вставки элемента в хеш-таблицу
- `V deleteNode(int key)` – Метод удаления элемента из хеш-таблицы по ключу
- `void display()` – Метод для вывода хеш-таблицы
- `void resize(int new_capacity)` – Метод для изменения размера хеш-таблицы и её очистки
- `int sizeofMap()` – Метод для вывода размера хеш-таблицы

3. ТЕСТИРОВАНИЕ

Результат запуска программы:

```
Number of elements is 512
Average insertion time and operations count in the average case is
0.000600 ms and 6 pieces
Average insertion time and operations count in the bad case is
0.003258 ms and 255 pieces
-------------------------------------------------------
Number of elements is 1024
Average insertion time and operations count in the average case is
0.000750 ms and 19 pieces
Average insertion time and operations count in the bad case is
0.005488 ms and 511 pieces
-------------------------------------------------------
Number of elements is 2048
Average insertion time and operations count in the average case is
0.000830 ms and 27 pieces
Average insertion time and operations count in the bad case is
0.010844 ms and 1023 pieces
-------------------------------------------------------
Number of elements is 4096
Average insertion time and operations count in the average case is
0.001010 ms and 25 pieces
Average insertion time and operations count in the bad case is
0.022280 ms and 2047 pieces
-------------------------------------------------------
Number of elements is 8192
Average insertion time and operations count in the average case is
0.000948 ms and 39 pieces
Average insertion time and operations count in the bad case is
0.042365 ms and 4095 pieces
-------------------------------------------------------
Number of elements is 16384
Average insertion time and operations count in the average case is
0.000890 ms and 32 pieces
Average insertion time and operations count in the bad case is
0.083744 ms and 8191 pieces
-------------------------------------------------------
Number of elements is 32768
Average insertion time and operations count in the average case is
0.002238 ms and 143 pieces
Average insertion time and operations count in the bad case is
0.170120 ms and 16383 pieces
-------------------------------------------------------
```

4. ИССЛЕДОВАНИЕ

Полученные результаты представлены в табл. 1.

Таблица 1 – Результаты эксперимента

Количество вставок	Среднее время одной вставки (в миллисекундах)		Среднее количество базовых операций	
	В среднем случае	В худшем случае	В среднем случае	В худшем случае
512	0.000600	0.003258	6	255
1024	0.000750	0.005488	19	511
2048	0.000830	0.010844	27	1023
4096	0.001010	0.022280	25	2047
8192	0.000948	0.042365	39	4095
16384	0.000890	0.083744	32	8191
32768	0.002238	0.170120	143	16383

В среднем случае, согласно теории, время одной вставки и количество базовых операций не зависят от количества вставок. В проведённом эксперименте на случайном наборе данных так же не было выявлено такой зависимости.

В худшем случае, согласно теории, время одной вставки и количество базовых операций линейно зависят от количества вставок. По результатам проведённого эксперимента можно сделать вывод, что при увеличении количества вставок время на одну вставку растёт, в среднем, в 1.937 раза, а количество базовых операций растёт, в среднем, в 2,001 раза, что соответствует теории.

Графики зависимостей см. на рис. 2, 3, 4, 5, 6.

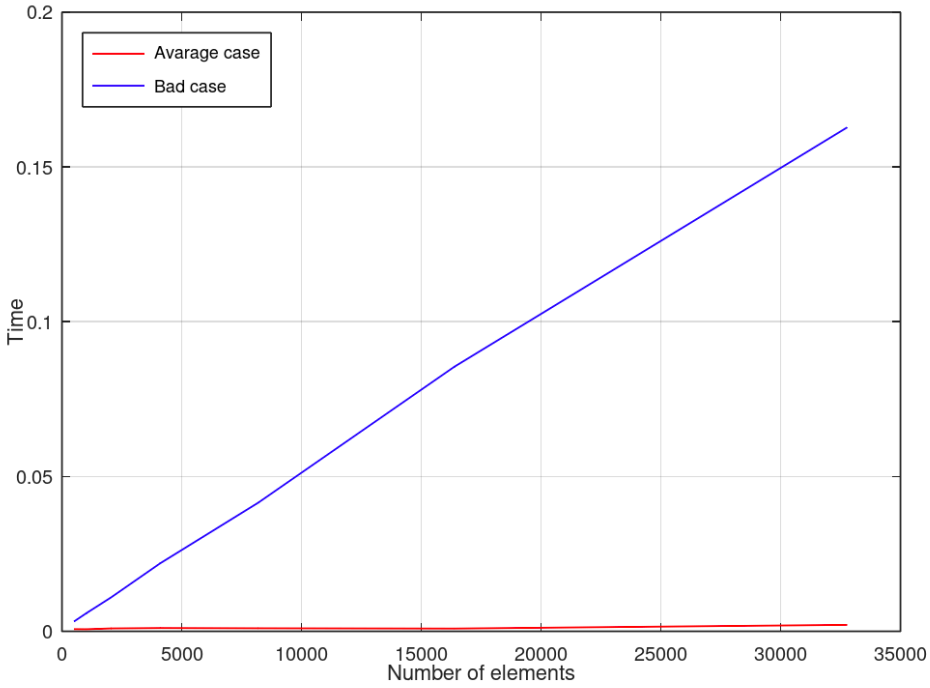

Рисунок 2 – График зависимости времени от количества вставок в среднем и худшем случае (на основе полученных данных)

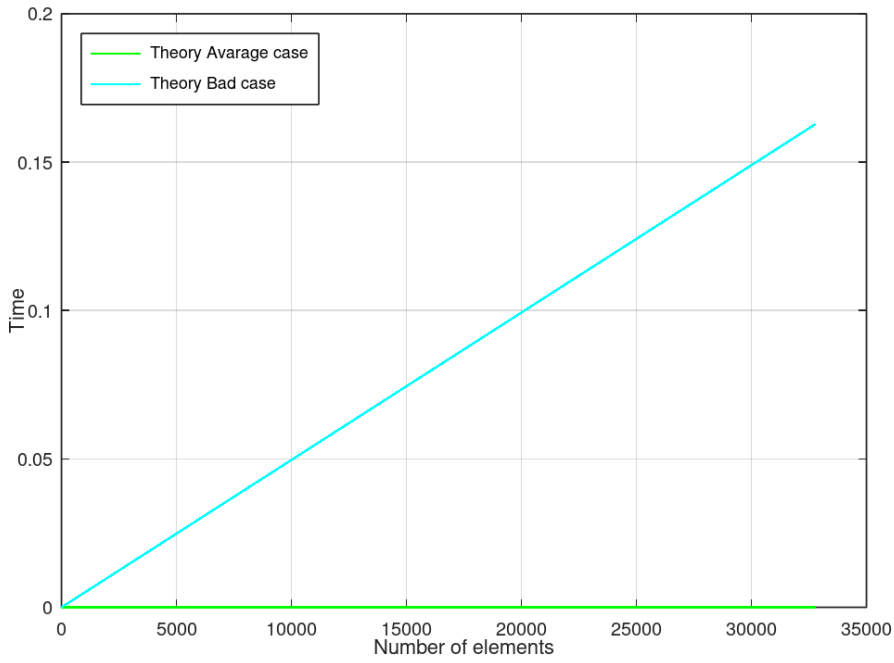

Рисунок 3 – График зависимости времени от количества вставок в среднем и худшем случае (на основе теории)

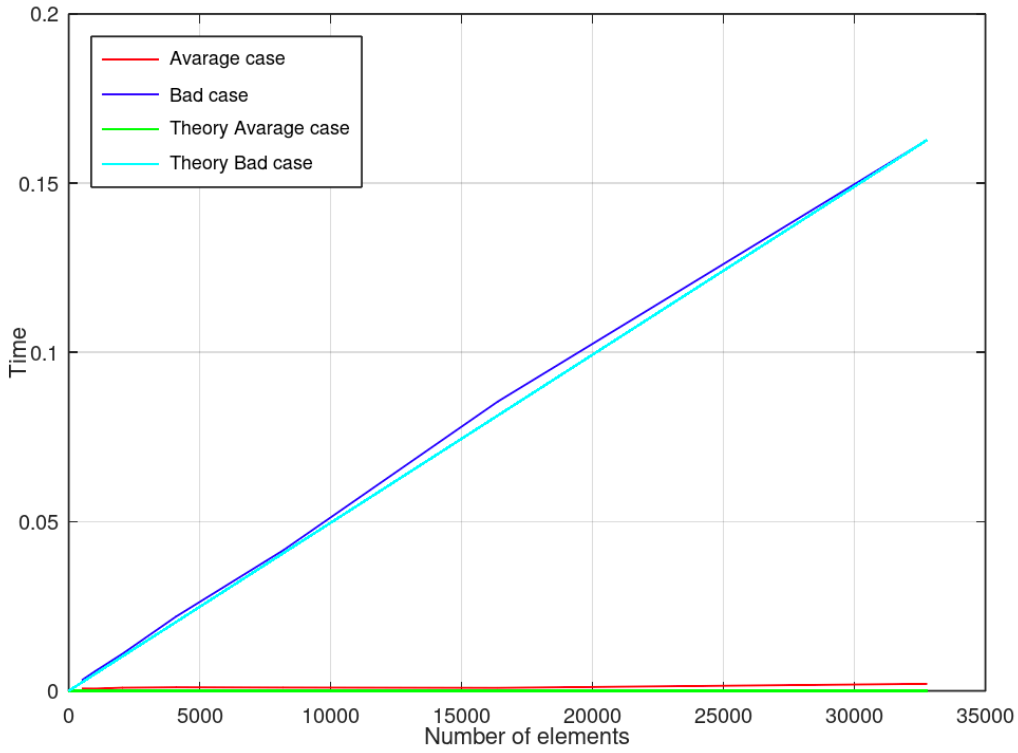

Рисунок 4 – Сравнение зависимостей на основе теоретических и практических данных

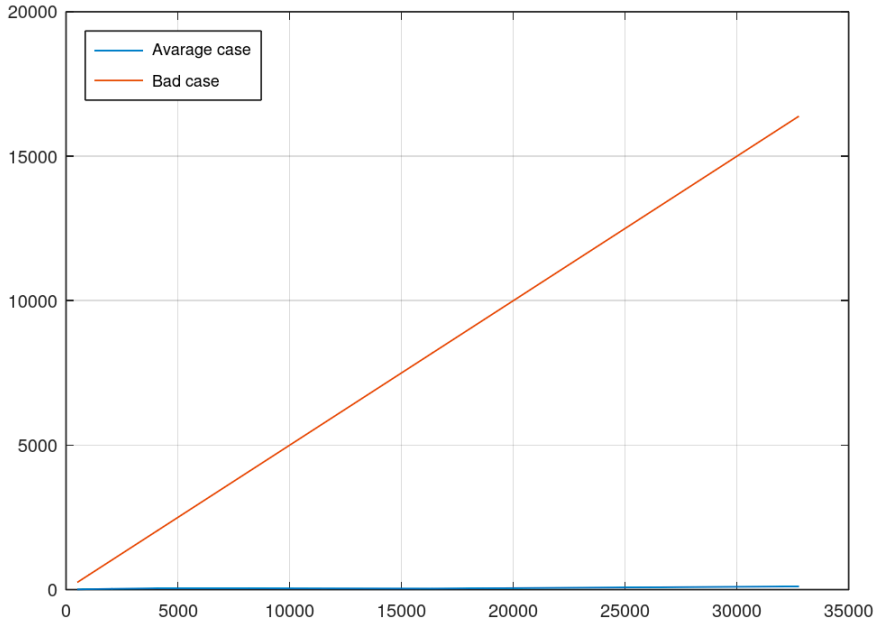

Рисунок 5 – График зависимости количества базовых операций от количества вставок в среднем и худшем случае (на основе полученных данных)

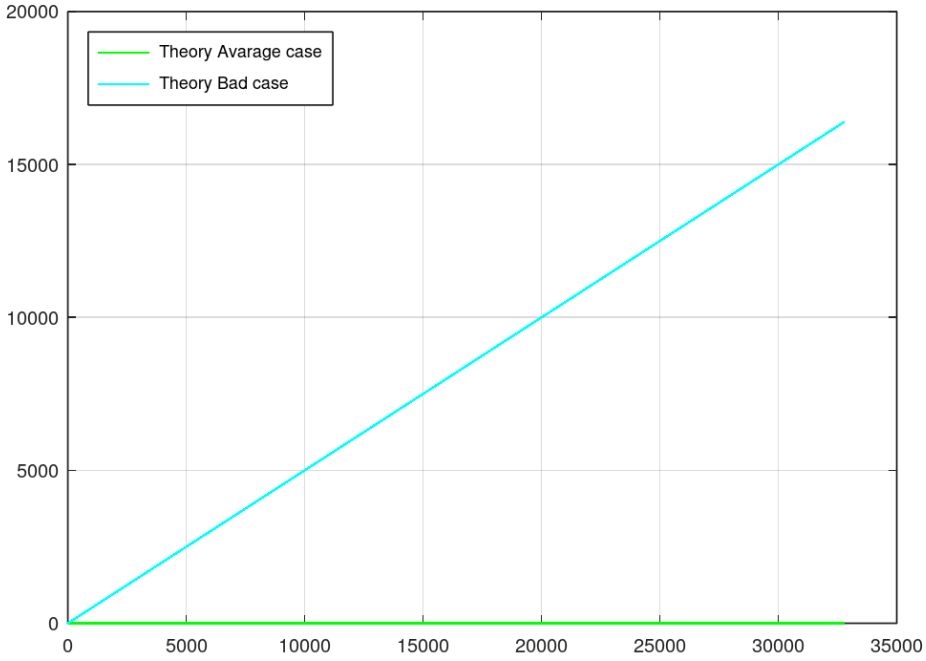

Рисунок 6 – График зависимости количества базовых операций от количества вставок в среднем и худшем случае (на основе теории)

ВЫВОДЫ

В ходе работы была реализована программа на языке программирования C++ для работы с хеш-таблицей с открытой адресацией, а именно: генерации входных данных, обработки и вывода результатов. Были найдены и изучены такие характеристики хеш-таблицы, как среднее время и количество базовых операций, необходимых для вставки элемента в хеш-таблицу. Так же с помощью полученных данных были подтверждены теоретические данные.

Исходный код программы см. в приложении А.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

https://www.youtube.com/watch?v=rVr1y32fDI0

https://ru.wikipedia.org/wiki/Хеш-таблица

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

HashMap.h

```
#include <iostream>
#include <vector>
#include <fstream>
#include <sstream>
#include <ctime>

#define DValue -1

using namespace std;

template<typename K, typename V>
// Класс элемента хеш-таблицы
class HashNode {
public:
    V value;
    K key;

    // Конструктор класса
    HashNode(K key, V value){
        this->value = value;
        this->key = key;
    }
};

template<typename K, typename V>
// Класс хеш-таблицы
class HashMap {
private:
    HashNode<K,V> **arr;
    int capacity;
    int size;
```

```
    HashNode<K,V> *undef;

public:
    // Конструктор класса
    HashMap(int cap){
        this->capacity = cap;
        this->size = 0;
        arr = new HashNode<K,V>*[capacity];

        for(int i = 0; i < capacity; i++)
            arr[i] = NULL;

        undef = new HashNode<K,V>(-1, DValue);
    }

    int hashCode(K key) { // Метод для получения хэша элемента
таблицы по ключу
        return key % capacity;
    }

    int insertNode(K key, V value) { // Метод вставки элемента в
хеш-таблицу
        auto *temp = new HashNode<K,V>(key, value);

        int hashIndex = hashCode(key);
        int counter = 0;

        while(arr[hashIndex] != NULL && arr[hashIndex]->key != -1)
{
            hashIndex++;
            hashIndex %= capacity;
            counter++;
        }

        if(arr[hashIndex] == NULL || arr[hashIndex]->key == -1) {
```

```
            size++;
            arr[hashIndex] = temp;
        }
        return counter;
    }

    V deleteNode(int key) { // Метод удаления элемента из хеш-таблицы по ключу
        int hashIndex = hashCode(key);

        while(arr[hashIndex] != NULL) {
            if(arr[hashIndex]->key == key) {
                HashNode<K,V> *temp = arr[hashIndex];

                arr[hashIndex] = undef;

                size--;
                return temp->value;
            }
            hashIndex++;
            hashIndex %= capacity;
        }

        return DValue;
    }

    void display() { // Метод для вывода хеш-таблицы
        cout << "Hash table is:\n";
        for (int i = 0; i < capacity; i++) {
            if(arr[i] != NULL && arr[i]->key != -1)
                cout << "key = " << arr[i]->key
                     <<" value = "<< arr[i]->value << endl;
        }
    }
```

```
    void resize(int new_capacity) { // Метод для изменения размера
хеш-таблицы и её очистки
        this->capacity = new_capacity;
        delete [] arr;
        arr = new HashNode<K,V>*[capacity];
        for(int i = 0; i < capacity; i++)
            arr[i] = NULL;
        size = 0;
    }

    int sizeofMap() { // Метод для вывода размера хеш-таблицы
        cout << "Size is " << size << endl;
        return size;
    }
};
```

main.cpp

```
#include "HashMap.h"

int main() {
    int capacity = 512;
    auto *h = new HashMap<int, int>(capacity);
    unsigned int start_time;
    double used_time = 0;
    unsigned int bad_key;
    int r_value;
    int r_key;
    int operations_count = 0;
    for (int k = 0; k < 7; k++) {
        printf("Number of elements is %d\n", capacity);
        // Вставка случайных чисел со случайным ключом в хеш-
таблицу (средний случай)
        for (int i = 0; i < capacity; i++) {
            srand((int) clock());
            r_key = rand() % 1000000; // Генерация ключа
```

```
            r_value = rand() % 1000000 + 1; // Генерация значения
            start_time = clock();
            operations_count += h->insertNode(r_key, r_value);
            used_time += ((double) (clock() - start_time)) /
CLOCKS_PER_SEC;
        }
        used_time *= 1000;
        used_time /= capacity;
        operations_count /= capacity;
        printf("Average insertion time and operations count in the
average case is %lf ms and %d pieces\n", used_time,
operations_count);
        used_time = 0;
        operations_count = 0;
        srand((int) clock());
        bad_key = rand() % 10000; // Генерация ключа для худшего
случая
        h->resize(capacity);
        // Вставка случайных чисел с одним и тем же ключом в хеш-
таблицу (худший случай)
        for (int i = 0; i < capacity; i++) {
            srand((int) clock());
            r_value = rand() % 1000000 + 1; // Генерация значения
            start_time = clock();
            operations_count += h->insertNode(bad_key, r_value);
// Вызов функции для вставки в хеш-таблицу
            used_time += ((double) (clock() - start_time)) /
CLOCKS_PER_SEC;
        }
        used_time *= 1000;
        used_time /= capacity;
        operations_count /= capacity;
        printf("Average insertion time and operations count in the
bad case is %lf ms and %d pieces\n", used_time, operations_count);
```

```
        printf("-------------------------------------------------
-------\n");
        capacity *= 2;
        h->resize(capacity);
        used_time = 0;
        operations_count = 0;
    }

    return 0;
}
```