|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** ***ИУК «Информатика и управление»***

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №5**

**«Реализация алгоритма поиска»**

**ДИСЦИПЛИНА: «Логика и теория алгоритмов»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-42Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Амеличев Г.Э. )  (Подпись) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2022

**Цель:** формирование практических навыков выбора и разработки эффективного алгоритма поиска с учетом особенностей конкретной задачи и структур обрабатываемых данных.

**Задачи:** реализовать алгоритмы вставки и поиска для хэш-таблицы заданного вида и исследовать вычислительную сложность реализованных алгоритмов. Вид хэш-таблицы выбирается в соответствии с вариантом задания, полученным от преподавателя.

**Вариант №3**

Тип таблицы: закрытая

Разрешение коллизий: линейное пробирование

Хэш-функция: деление

Расширение: да

**Листинг:**

***HashTable.h***

#pragma once

#include <string>

namespace LITA\_LW\_5

{

class HashTable

{

public:

HashTable(size\_t size);

~HashTable();

void Add(int key, std::string data);

bool Find(int key, std::string& data);

void Print();

private:

float GetOccupancyRate();

void Expand();

struct Record

{

int key{};

std::string data{};

};

size\_t \_size{};

Record\*\* \_table = nullptr;

size\_t \_elementsNumber = 0;

float \_MaxLoadFactor = 0.7;

uint16\_t \_ExpandFactor = 2;

};

}

***HashTable.cpp***

#include "HashTable.h"

#include <iostream>

namespace LITA\_LW\_5

{

HashTable::HashTable(size\_t size) : \_size(size), \_elementsNumber(0),

\_table(new Record\* [size] {}) {}

HashTable::~HashTable()

{

for (size\_t i{}; i < \_size; ++i)

delete \_table[i];

delete[] \_table;

}

void HashTable::Add(int key, std::string data)

{

size\_t i = 0;

size\_t hash{};

do

{

hash = (key + i++) % \_size;

} while (\_table[hash] != nullptr);

\_table[hash] = new Record{ key, data };

++\_elementsNumber;

if (GetOccupancyRate() > \_MaxLoadFactor)

Expand();

}

bool HashTable::Find(int key, std::string& data)

{

size\_t i = 0;

size\_t hash = (key + i++) % \_size;

while (\_table[hash] != nullptr)

{

if (\_table[hash]->key == key)

{

data = \_table[hash]->data;

return true;

}

hash = (key + i++) % \_size;

}

return false;

}

void HashTable::Print()

{

for (size\_t i{}; i < \_size; ++i)

if (\_table[i] != nullptr)

std::cout << i << " " << \_table[i]->key << " "

<< \_table[i]->data << "\n";

else

std::cout << i << "\n";

std::cout << "\n";

}

float HashTable::GetOccupancyRate() { return (float)\_elementsNumber / \_size; }

void HashTable::Expand()

{

Record\*\* expanded = new Record \* [\_size \* \_ExpandFactor]{};

for (size\_t i{}; i < \_size; ++i)

{

if (\_table[i] != nullptr)

{

size\_t j = 0;

size\_t hash{};

do

{

hash = (\_table[i]->key + j++) % (\_size \* \_ExpandFactor);

} while (expanded[hash] != nullptr);

expanded[hash] = new Record{ \_table[i]->key, \_table[i]->data };

delete \_table[i];

}

}

delete[] \_table;

\_table = expanded;

\_size \*= \_ExpandFactor;

}

}

***Main.cpp***

#include "HashTable.h"

#include <iostream>

#include <vector>

#include <time.h>

#include <iomanip>

#include <chrono>

int main()

{

LITA\_LW\_5::HashTable table(5);

uint16\_t command = 1;

while (command != 0)

{

system("cls");

table.Print();

std::cout << "1. Add\n";

std::cout << "2. Find\n";

std::cout << "0. Exit\n";

std::cout << ">>> ";

std::cin >> command;

std::cout << "\n";

if (command == 1)

{

std::cout << "Key: ";

int key;

std::cin >> key;

std::cin.ignore();

std::cout << "Data: ";

std::string data;

std::getline(std::cin, data);

table.Add(key, data);

}

else if (command == 2)

{

std::cout << "Key: ";

int key;

std::cin >> key;

std::string data;

if (table.Find(key, data))

std::cout << "Value: " << data << "\n";

else

std::cout << "Value wasn't found\n";

system("pause");

}

}

srand(time(NULL));

size\_t elementsNumber = 15;

size\_t tableSize = 20;

std::vector<float>\* addTimes = new std::vector<float>[elementsNumber]{};

std::vector<float>\* findTimes = new std::vector<float>[elementsNumber]{};

std::vector<float> resultAddTimes{};

std::vector<float> resultFindTimes{};

for (int f{}; f < 5; ++f)

{

LITA\_LW\_5::HashTable hashTable(tableSize);

std::vector<int> keys{};

keys.push\_back(rand() % tableSize);

for (int j{}; j < elementsNumber - 1; ++j)

keys.push\_back(keys[j] + rand() % tableSize);

std::vector<int> existingKeys{};

for (int j{}; j < rand() % (elementsNumber + 1); ++j)

existingKeys.push\_back(keys[j]);

for (int j = existingKeys.size(); j < elementsNumber; ++j)

if (j == 0)

existingKeys.push\_back(rand() % tableSize);

else

existingKeys.push\_back(existingKeys[j - 1] + rand() % tableSize);

for (int j{}; j < elementsNumber; ++j)

{

std::string temp{};

auto begin = std::chrono::steady\_clock::now();

hashTable.Find(existingKeys[j], temp);

auto end = std::chrono::steady\_clock::now();

findTimes[j].push\_back(std::chrono::duration\_cast<std::chrono::nanoseconds>(end - begin).count());

begin = std::chrono::steady\_clock::now();

hashTable.Add(keys[j], "Data");

end = std::chrono::steady\_clock::now();

addTimes[j].push\_back(std::chrono::duration\_cast<std::chrono::nanoseconds>(end - begin).count());

}

}

for (int i{}; i < elementsNumber; ++i)

{

float max = 0;

float sum = 0;

for (int j{}; j < 5; ++j)

{

sum += addTimes[i][j];

if (addTimes[i][j] > max)

max = addTimes[i][j];

}

resultAddTimes.push\_back((sum - max) / 4);

max = 0;

sum = 0;

for (int j{}; j < 5; ++j)

{

sum += findTimes[i][j];

if (findTimes[i][j] > max)

max = findTimes[i][j];

}

resultFindTimes.push\_back((sum - max) / 4);

}

for (int i{}; i < elementsNumber; ++i)

{

std::cout << std::setw(6) << (float)i / tableSize << " "

<< std::setw(6) << resultAddTimes[i] << " "

<< std::setw(6) << resultFindTimes[i] << "\n";

}

return 0;

}

**Результат:**

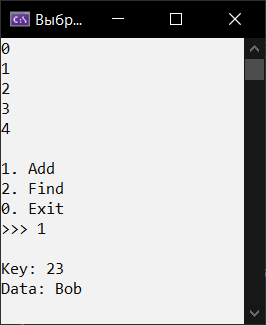
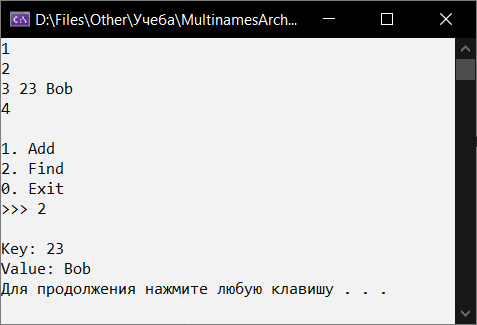
 

Рис. 1. Результат

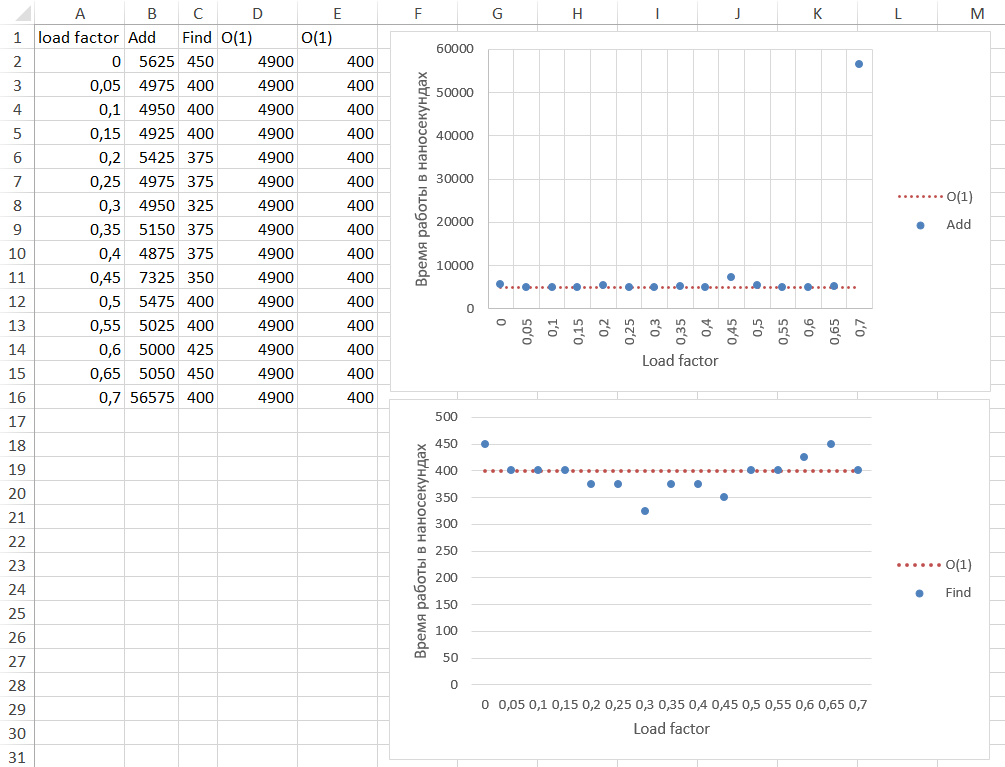


Рис. 2. Производительность

**Вывод:** в ходе выполнения лабораторной работы были получены навыки реализации закрытой хэш-таблицы с линейным пробированием.