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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

Тема: БДП и Хеш-таблицы

Студент гр. 9381 _____ Чухарев И.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы

Познакомиться с хеш-таблицей методом открытой адресации, представить реализацию на языке программирования C++.

Постановка задачи

Вариант 25.

Хеш-таблица: с открытой

адресацией; действие: 1+2a

1) По заданной последовательности элементов Elem построить структуру данных определённого типа – БДП или хеш-таблицу;

2) Выполнить одно из следующих действий:

a) Для построенной структуры данных проверить, входит ли в неё элемент e типа Elem, и если входит, то в скольких экземплярах. Добавить элемент e в структуру данных. Предусмотреть возможность повторного выполнения с другим элементом.

Описание алгоритма

Был написан класс HashTable, в котором реализована хеш-таблица методом открытой адресации двойным хешированием. Для добавления элемента реализован следующий алгоритм: написаны две хеш функции, первая из которых возвращает натуральное число h1, по индексу которого пытаемся добавить элемент. В случае неудачи высчитываем второй хеш-функцией натуральное число h2, которое является шагом, на который мы будем сдвигаться для поиска пустой ячейки.

Описание классов:

HashTable — шаблонный класс хеш-таблицы. std::vector < std::pair<T, bool>* > table - хеш-таблица, int size — текущий уровень заполненности таблицы.

Методы класса:

1.void add(const T &key) — добавление элемента в хеш-таблицу. T key — элемент, который необходимо добавить. Высчитываем хеш, при необходимости делаем шаг, пока не найдём пустое место и добавляем элемент.

2.int count(const T &key) — поиск элемента в хеш-таблице. T key — элемент, который необходимо посчитать. Аналогично добавлению считаем хеш, пока не наткнёмся на элемент. Добавляем индекс в массив, если нашли элемент.

4.bool remove(const T &key) — удаление элемента из хеш-таблицы. T key — элемент, который необходимо удалить. Возвращает true, если элемент удалён, иначе — false.

5.void reHash() — изменение размера таблицы, при достижении уровня заполненности (80%). Удваивает объем таблицы, пересчитывает хеш и добавляем элементы.

6.int h1Calculate(const T &key, int m) — высчитывание хеш-функции h1 для элементе T key, корректирует размер относительно m — размера таблицы. Возвращает хеш.

7.int h2Calculate(const T &key, int m) — высчитывание хеш-функции h2 для элемента T key, корректирует размер относительно m — размера таблицы. Возвращает хеш.

Описание функций:

1.std::vector<std::string> split(const std::string &str, char delimiter) — разбиение строки str по разделителю delimiter, возвращает вектор строк.

2.std::string getString(std::istream &iStream) — считывает строку из iStream, после чего возвращает её.

Тестирование

Номер	Входные данные	Исходные данные
1	1 simple test	Ваша строка: simple test Добавляем элемент simple Таблица заполнена более чем на 80%, удваиваем размер и пересобираем Считаем h1(simple) h1(simple) = 0 Считаем h2(simple) h2(simple) = 1 Хеш-функция на шаге i = 0: (0 + 0 * 1) % 2 = 0 Элемент успешно добавлен Добавляем элемент test Таблица заполнена более чем на 80%, удваиваем размер и пересобираем Считаем h1(simple) h1(simple) = 2 Считаем h2(simple) h2(simple) = 1 Хеш-функция на шаге i = 0: (2 + 0 * 1) % 4 = 2 Считаем h1(test) h1(test) = 2 Считаем h2(test) h2(test) = 1 Хеш-функция на шаге i = 0: (2 + 0 * 1) % 4 = 2 Хеш-функция на шаге i = 1: (2 + 1 * 1) % 4 = 3 Элемент успешно добавлен [0] = nullptr [1] = nullptr [2] = simple [3] = test
2	3	Таблица: [0] = nullptr [1] = nullptr [2] = simple [3] = test

3	2 simple	Ваша строка: simple Ищем элемент simple Считаем h1(simple) h1(simple) = 2 Считаем h2(simple) h2(simple) = 1 Хеш-функция на шаге i = 1: (2 + 1 * 1) % 4 = 3 Хеш-функция на шаге i = 2: (2 + 2 * 1) % 4 = 0 [0] = nullptr [1] = nullptr [2] = simple [3] = test Количество элемента simple в таблице: 1
4	2 add	Ваша строка: add Ищем элемент add Считаем h1(add) h1(add) = 1 Считаем h2(add) h2(add) = 3 Считаем h1(add) h1(add) = 1 Считаем h2(add) h2(add) = 3 Хеш-функция на шаге i = 0: (1 + 0 * 3) % 4 = 1 Количество элемента add в таблице: 0
5	3	Ваша строка: add Ищем элемент add Считаем h1(add) h1(add) = 1 Считаем h2(add) h2(add) = 3 Считаем h1(add) h1(add) = 1 Считаем h2(add) h2(add) = 3 Хеш-функция на шаге i = 0: (1 + 0 * 3) % 4 = 1 Количество элемента add в таблице: 0

Выводы

Была изучена и реализована на языке программирования C++ хеш-таблица методом открытой адресации.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.cpp

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

#define NO_COLOR "\033[0m"
#define GREEN_COLOR "\033[1;32m"
#define BLUE_COLOR "\033[0;34m"

#define REHASH_FACTOR 0.8

// шаблонный класс хеш-таблицы
template <typename T>
class HashTable {
    std::vector < std::pair<T, bool>* > table; // вектор пар ключ-значение
    int size; // количество заполненных элементов в таблице

public:
    explicit HashTable(int size) : size(size) {
        table.resize(size); // выделяем память под элементы
    }

    // вывод элементов таблицы
    void output() {
        for (auto i = 0; i < table.size(); i++) {
            if (table[i])
                std::cout << "[" << i << "] = " << table[i]->first << (table[i]->second ? " (deleted)\n" : "\n");
            else
                std::cout << "[" << i << "] = nullptr" << '\n';
        }
    }
```

```
        // хеш-функция h1(x), где x - элемент
        int h1Calculate(const T& key, int m) {
            std::cout << "Считаем h1(" << key << ")\n";
            int hash = 0, a = 11;

            for (const auto symbol : key) {
                hash = (hash * a + symbol) % m;
            }

            std::cout << "h1(" << key << ") = " << hash << '\n';
            return hash;
        }

        // хеш-функция h2(x), где x - элемент
        int h2Calculate(const T& key, int m) {
            std::cout << "Считаем h2(" << key << ")\n";
            int hash = 0, a = 17;

            for (const auto symbol : key) {
                hash = (hash * a + symbol) % m;
            }

            hash = (hash * 2 + 1) % m;
            std::cout << "h2(" << key << ") = " << hash << '\n';
            return hash;
        }

        // хеш-функция
        int hashFunction(int h1, int h2, int i, int m) {
            if (!m) {
                std::cout << "Вспомогательная хеш-функция = 0" << '\n';
                return 0;
            }

            int hash = (h1 + i * h2) % m;
            std::cout << "Хеш-функция на шаге i = " << i << ": ("
            << h1 << " + " << i << " * " << h2 << ") % " << m << " = "
<< hash
            << '\n';
```

```
            return hash;
        }

        void paintElement(const std::vector<std::string> &elements,
const char *color) {
            for (auto i = 0; i < table.size(); i++) {
                if (table[i] && std::find(elements.begin(),
elements.end(), table[i]->first) != elements.end()) {
                    std::cout << color << "[" << i << "] = " <<
table[i]->first << '\n' << NO_COLOR;
                } else if (table[i]) {
                    std::cout << "[" << i << "] = " << table[i]->first
<< (table[i]->second ? " (deleted)\n" : "\n");
                } else {
                    std::cout << "[" << i << "] = nullptr" << '\n';
                }
            }
        }

        void paintElement(const std::vector<int> &index, const char
*color) {
            for (auto i = 0; i < table.size(); i++) {
                if (std::find(index.begin(), index.end(), i) !=
index.end() && table[i]) {
                    std::cout << color << "[" << i << "] = " <<
table[i]->first << '\n' << NO_COLOR;
                } else if (table[i]) {
                    std::cout << "[" << i << "] = " << table[i]->first
<< (table[i]->second ? " (deleted)\n" : "\n");
                } else {
                    std::cout << "[" << i << "] = nullptr" << '\n';
                }
            }
        }

        void paintElement(int index, const char *color) {
            if (!table[index])
                return;
```

```
            for (auto i = 0; i < table.size(); i++) {
                if (i == index) {
                    std::cout << color << "[" << i << "] = " <<
table[i]->first << '\n' << NO_COLOR;
                } else if (table[i]) {
                    std::cout << "[" << i << "] = " << table[i]->first
<< (table[i]->second ? " (deleted)\n" : "\n");
                } else {
                    std::cout << "[" << i << "] = nullptr" << '\n';
                }
            }
        }

        // добавление элемента
        void add(const T &key) {
            if (static_cast<double>(size) /
static_cast<double>(table.size()) >= REHASH_FACTOR) // пересобираем
таблицу при
                reHash();

// достижении процента заполненности
            // вычисление хеша
            int h1 = h1Calculate(key, table.size());
            int h2 = h2Calculate(key, table.size());
            int h = hashFunction(h1, h2, 0, table.size());

            for (auto i = 0; i < table.size() && table[h]; i++) {
                if (table[h]->second) { // если такой элемент был
удалён из таблицы
                    table[h]->first = key;
                    table[h]->second = false;
                    size++;
                    return;
                }

                h = hashFunction(h1, h2, i + 1, table.size());
            }

            // добавляем элемент
```

```
            table[h] = new std::pair <T, bool> (key, false);
            size++;
        }

        int count(const T &key) {
            int i = 0;
            int h1 = h1Calculate(key, table.size());
            int h2 = h2Calculate(key, table.size());
            int h = h1;
            std::vector<int> count;

            while (table[h] && i < table.size()) {
                if (table[h]->first == key && !table[h]->second)
                    count.push_back(h);

                h = hashFunction(h1, h2, i + 1, table.size());
                i++;
            }

            if (count.empty())
                add(key);
            else
                paintElement(count, BLUE_COLOR);

            return count.size();
        }

        void reHash() {
            std::cout << "Таблица заполнена более чем на 80%, удваиваем
размер и пересобираем" << '\n';
            auto newBufferSize = table.size() * 2;
            std::vector< std::pair <T, bool>* >
newBuffer(newBufferSize, nullptr);

            for (auto i = 0; i < table.size(); i++) {
                if (table[i] && !table[i]->second) {
                    int h1 = h1Calculate(table[i]->first,
newBufferSize);
```

```
                    int h2 = h2Calculate(table[i]->first,
newBufferSize);
                    int h = hashFunction(h1, h2, 0, newBufferSize);

                    for (auto j = 0; j < newBufferSize; j++) {
                        if (!newBuffer[h])
                            break;

                        h = hashFunction(h1, h2, j, newBufferSize);
                    }

                    newBuffer[h] = table[i];
                }
            }

            table = newBuffer;
        }

        ~HashTable() {
            for (auto &elem : table)
                delete elem;
        }
    };

    // разбиение исходной строки по разделителю, возвращает вектор
строк
    std::vector<std::string> split(const std::string &string, char
delimiter) {
        std::vector<std::string> result;
        std::string token;
        std::istringstream tokenStream(string);

        while (std::getline(tokenStream, token, delimiter))
            result.push_back(token);

        return result;
    }

    // считывание строки из потока ввода
```

```
std::string getString(std::istream &iStream) {
    std::string value;
    std::cout << "Введите вашу строку: ";
    std::getline(iStream, value, '\n');
    std::cout << "Ваша строка: " << value << '\n';
    return value;
}

int main() {
    HashTable<std::string> table(1); // хеш-таблица
    int command = 0; // комманда пользователя
    int count = 0; // количество повторений элемента
    std::string value; // ввод пользователя
    std::vector <std::string> elements; // ввод, разбитый на слова
    std::istream *iStream = &std::cin; // поток ввода
    std::ifstream file;

    while (command != 6) {
        std::cout << "Выберите один из предложенных пунктов: " <<
'\n';
        std::cout << "1. Добавить элемент" << '\n';
        std::cout << "2. Найти элемент (добавляет при отсутствии)"
<< '\n';
        std::cout << "3. Вывод таблицы" << '\n';
        std::cout << "4. Открыть файл" << (file.is_open() ? "
(сейчас открыт)" : " (сейчас закрыт)") << '\n';
        std::cout << "5. Закрыть файл и считывать с клавиатуры" <<
'\n';
        std::cout << "6. Завершение программы" << '\n';
        std::cout << "Ваш выбор: ";
        std::cin >> command;
        std::cin.ignore(); // пропускаем символ переноса строки

        if (std::cin.fail())
            break;

        switch (command) {
            case 1:
                value = getString(*iStream);
```

```
                    elements = split(value, ' ');

                    for (const auto &word : elements) {
                        std::cout << "Добавляем элемент " << word << '\
n';
                        table.add(word);
                        std::cout << "Элемент успешно добавлен" << '\n'
<< '\n';
                    }

                    table.paintElement(elements, GREEN_COLOR);
                    break;
                case 2:
                    value = getString(*iStream);
                    elements = split(value, ' ');

                    for (const auto &word : elements) {
                        std::cout << "Ищем элемент " << word << '\n';
                        count = table.count(word);
                        std::cout << "Количество элемента " << word <<
" в таблице: " << count << "\n\n";
                    }

                    break;
                case 3:
                    std::cout << "Таблица: " << '\n';
                    table.output();
                    break;
                case 4:
                    std::cout << "Введите путь до файла: ";
                    std::cin >> value;

                    file.open(value);

                    if (!file.is_open()) {
                        std::cout << "Файл не удалось открыть" << '\n';
                        continue;
                    }
```

```
                iStream = &file;
                break;
            case 5:
                if (file.is_open())
                    file.close();

                iStream = &std::cin;
                break;
            case 6:
            default:
                return 0;
        }

        std::cout << '\n';
    }

    return 0;
}
```