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: Хеш-таблицы с открытой адресацией — вставка и исключение

Студент гр. 9381 _____ Чухарев И.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Чухарев И.А.

Группа 9381

Тема работы:

Вариант 25. Хеш-таблицы с открытой адресацией – вставка и исключение. Демонстрация.

Исходные данные:

строка, слова из которой необходимо добавить в хеш-таблицу

Содержание пояснительной записки:

«Содержание», «Введение», «Ход выполнения работы», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 15 страниц.

Дата выдачи задания: 31.10.2020

Дата сдачи реферата: 28.12.2020

Дата защиты реферата: 29.12.2020

Студент _____ Чухарев И.А.

Преподаватель _____ Фирсов М.А.

АННОТАЦИЯ

Была реализована хеш-таблица методом открытой адресации на языке программирования C++. Вставка и исключение элементов продемонстрированы. Для работы с программой был написан консольный интерфейс.

SUMMARY

The hash table was implemented using the open addressing method in the C++ programming language. Inserting and excluding elements are demonstrated. To work with the program, a console interface was written.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является написание программы, реализующей хеш-таблицу методом открытой адресации. Для этого необходимо изучить соответствующую структуру данных, операции вставки, удаления и поиска элементов в ней, а также синтаксис языка программирования C++.

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 поиска и операцию удаления пары по ключу.

Метод открытой адресации состоит в том, чтобы, пользуясь каким-либо алгоритмом, обеспечивающим перебор элементов таблицы, просматривать их в поисках свободного места для новой записи.

Линейное опробование сводится к последовательному перебору элементов таблицы с некоторым фиксированным шагом.

Квадратичное опробование отличается от линейного тем, что шаг перебора элементов не линейно зависит от номера попытки найти свободный элемент.

Еще одна разновидность метода открытой адресации, которая называется двойным хешированием, основана на нелинейной адресации, достигаемой за счет суммирования значений основной и дополнительной хеш-функций.

1. ЗАДАНИЕ

Требуется на языке программирования C++ реализовать хеш-таблицу, разрешив коллизию методом открытой адресации. Кроме того, необходимо спроектировать интерфейс для работы с программой, продемонстрировав основные операции с хеш-таблицей.

Программа должна предоставлять основной функционал для работы с хеш-таблицей (вставка/исключение), а также выдавать справочную информацию пользователю, меню для работы и промежуточные выводы.

2. ОПИСАНИЕ АЛГОРИТМА

2.1.Хеш-таблица

Согласно условию задачи, требуется реализовать хеш-таблицу. Для этого на языке программирования C++ был написан соответствующий класс, содержащий API для основных операций (вставка/исключение). Для хранения данных использован контейнер STL (std::vector).

2.2.Коллизии

Поскольку для вставки элемента в хеш-таблицу используется хеш-функция, которая может вернуть для разных элементов одинаковые значения, необходимо решать эту проблему, называемую коллизией. Для этого был использован метод двойного хеширования, работающий следующим образом: используем две хеш-функции, первая из которых возвращает нам натуральное число h1, а вторая — шаг h2. Изначально пытаемся установить элемент на место h1, если место занято, то с помощью шага переходим на следующий индекс (h1 + h2, h1 + 2 * h2 и т.д) до тех пор, пока не наткнёмся на пустое место.

2.3.Основные операции

Вставка реализована следующим образом: высчитываем значения h1, h2. Пытаемся установить элемент на место h1, если не удаётся — делаем шаг h2, пока не найдём пустое место.

Удаление повторяет вставку, но вместо добавления элемента мы удаляем его из таблицы.

3. ОПИСАНИЕ СТРУКТУР ДАННЫХ И ФУНКЦИЙ

3.1.Класс хеш-таблицы HashTable

Для работы с хеш-таблицей был реализован шаблонный класс HashTable, приватными полями которого являются: istd::vector < std::pair<T, bool>* > table - хеш-таблица, int size — текущий уровень заполненности таблицы

Рассмотрим публичные методы класса HashTable:

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 (75%). 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.

3.2.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.

4. ОПИСАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Для консоли был реализован интерфейс пользователя для работы с программой. Он состоит из 7 пунктов. Рассмотрим их:

1. Добавить элемент
2. Удалить элемент
3. Найти элемент (добавляет при отсутствии)
4. Вывод таблицы
5. Открыть файл
6. Закрыть файл и считывать с клавиатуры
7. Завершение программы

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы была реализована хеш-таблица методом открытой адресации на языке программирования C++. Вставка и исключение элементов продемонстрированы. Для работы с программой был написан консольный интерфейс.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Керниган Б. И Ритчи Д. Язык программирования Си М.: Вильямс, 1978 288 с.

ПРИЛОЖЕНИЕ А

ТЕСТИРОВАНИЕ

Номер	Входные данные	Выходные данные	Комментарий
1	1 test	Ваша строка: test Добавляем элемент test Таблица заполнена более чем на 75%, удваиваем размер и пересобираем Считаем h1(test) h1(test) = 0 Считаем h2(test) h2(test) = 1 Хеш-функция на шаге i = 0: (0 + 0 * 1) % 2 = 0 Элемент успешно добавлен [0] = test [1] = nullptr	Элемент был успешно добавлен в таблицу
2	2 test	Ваша строка: test Удаляем элемент test Считаем h1(test) h1(test) = 0 Считаем h2(test) h2(test) = 1 Хеш-функция на шаге i = 0: (0 + 0 * 1) % 2 = 0 [0] = test [1] = nullptr	Элемент был успешно удалён из таблицы
3	1 test	Ваша строка: test Добавляем элемент test Считаем h1(test) h1(test) = 0 Считаем h2(test) h2(test) = 1 Хеш-функция на шаге i = 0: (0 + 0 * 1) % 2 = 0 Элемент успешно добавлен [0] = test	Элемент после удаления был добавлен на то же место

		[1] = nullptr	
4	4	Таблица: [0] = test [1] = nullptr	Таблица была выведена на экран
5	1 simple test one two three	Ваша строка: simple test one two three Добавляем элемент simple Таблица заполнена более чем на 75%, удваиваем размер и пересобираем Считаем h1(test) h1(test) = 2 Считаем h2(test) h2(test) = 1 Хеш-функция на шаге i = 0: (2 + 0 * 1) % 4 = 2 Считаем h1(simple) h1(simple) = 2 Считаем h2(simple) h2(simple) = 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 Добавляем элемент test Таблица заполнена более чем на 75%, удваиваем размер и пересобираем Считаем h1(test) h1(test) = 6 Считаем h2(test) h2(test) = 1 Хеш-функция на шаге i = 0: (6 + 0 * 1) % 8 = 6 Считаем h1(simple) h1(simple) = 2 Считаем h2(simple) h2(simple) = 5	Добавление нескольких элементов в таблицу

		Хеш-функция на шаге i = 0: (2 + 0 * 5) % 8 = 2 Считаем h1(test) h1(test) = 6 Считаем h2(test) h2(test) = 1 Хеш-функция на шаге i = 0: (6 + 0 * 1) % 8 = 6 Хеш-функция на шаге i = 1: (6 + 1 * 1) % 8 = 7 Элемент успешно добавлен Добавляем элемент one Считаем h1(one) h1(one) = 6 Считаем h2(one) h2(one) = 5 Хеш-функция на шаге i = 0: (6 + 0 * 5) % 8 = 6 Хеш-функция на шаге i = 1: (6 + 1 * 5) % 8 = 3 Элемент успешно добавлен Добавляем элемент two Считаем h1(two) h1(two) = 0 Считаем h2(two) h2(two) = 5 Хеш-функция на шаге i = 0: (0 + 0 * 5) % 8 = 0 Элемент успешно добавлен Добавляем элемент three Таблица заполнена более чем на 75%, удваиваем размер и пересобираем Считаем h1(two) h1(two) = 0 Считаем h2(two) h2(two) = 5	

		Хеш-функция на шаге i = 0: (0 + 0 * 5) % 16 = 0 Считаем h1(simple) h1(simple) = 10 Считаем h2(simple) h2(simple) = 5 Хеш-функция на шаге i = 0: (10 + 0 * 5) % 16 = 10 Считаем h1(one) h1(one) = 6 Считаем h2(one) h2(one) = 5 Хеш-функция на шаге i = 0: (6 + 0 * 5) % 16 = 6 Считаем h1(test) h1(test) = 14 Считаем h2(test) h2(test) = 1 Хеш-функция на шаге i = 0: (14 + 0 * 1) % 16 = 14 Считаем h1(test) h1(test) = 14 Считаем h2(test) h2(test) = 1 Хеш-функция на шаге i = 0: (14 + 0 * 1) % 16 = 14 Хеш-функция на шаге i = 0: (14 + 0 * 1) % 16 = 14 Хеш-функция на шаге i = 1: (14 + 1 * 1) % 16 = 15 Считаем h1(three) h1(three) = 10 Считаем h2(three) h2(three) = 1 Хеш-функция на шаге i = 0: (10 + 0 * 1) % 16 = 10 Хеш-функция на шаге i = 1: (10 + 1 * 1) % 16 =	

		11 Элемент успешно добавлен [0] = two [1] = nullptr [2] = nullptr [3] = nullptr [4] = nullptr [5] = nullptr [6] = one [7] = nullptr [8] = nullptr [9] = nullptr [10] = simple [11] = three [12] = nullptr [13] = nullptr [14] = test [15] = test	

ПРИЛОЖЕНИЕ Б

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 RED_COLOR "\033[0;31m"
#define BLUE_COLOR "\033[0;34m"

#define REHASH_FACTOR 0.75

// 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++;
}

// удаление элемента
bool remove(const T &key) {
    // вычисляем хеш
    int h1 = h1Calculate(key, table.size());
    int h2 = h2Calculate(key, table.size());
    int j = hashFunction(h1, h2, 0, table.size());

    for (auto i = 0; i < table.size(); i++) {
        if (!table[j])
            return false;

        // удаляем элемент при наличии
        if (table[j]->first == key && !table[j]->second) {
            table[j]->second = true;
            size--;
            paintElement(j, RED_COLOR);
            return true;
        }

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

    // элемента нет
    return false;
}

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 75%, 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
        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 != 7) {
            std::cout << "Выберите один из предложенных пунктов: " <<
'\n';
            std::cout << "1. Добавить элемент" << '\n';
            std::cout << "2. Удалить элемент" << '\n';
            std::cout << "3. Найти элемент (добавляет при отсутствии)"
<< '\n';
            std::cout << "4. Вывод таблицы" << '\n';
            std::cout << "5. Открыть файл" << (file.is_open() ? "
(сейчас открыт)" : " (сейчас закрыт)") << '\n';
            std::cout << "6. Закрыть файл и считывать с клавиатуры" <<
'\n';
            std::cout << "7. 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 << "Удаляем элемент " << word << '\
n';
                        table.remove(value);
                        std::cout << '\n';
                    }

                    break;
                case 3:
                    value = getString(*iStream);
                    elements = split(value, ' ');

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

                    break;
                case 4:
                    std::cout << "Таблица: " << '\n';
                    table.output();
                    break;
                case 5:
                    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 6:
                if (file.is_open())
                    file.close();

                iStream = &std::cin;
            case 7:
            default:
                return 0;
        }
        std::cout << '\n';
    }

    return 0;
}
```