



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Отчет по выполнению практического задания №6

Тема: «АЛГОРИТМЫ ПОИСКА»

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент
группа

Кузнецов Л. А.
ИКБО-20-23

Москва 2024

СОДЕРЖАНИЕ

ЧАСТЬ 6.1.....	3
Условие.....	3
Метод решения.....	3
ЧАСТЬ 6.2.....	11
Условие.....	11
ЗАДАНИЕ 1.....	12
Формулировка задачи.....	12
Математическая модель решения.....	12
Код программы с комментариями.....	12
Результаты тестирования.....	13
ЗАДАНИЕ 2.....	15
Формулировка задачи.....	15
Математическая модель решения.....	15
Код программы с комментариями.....	15
Результаты тестирования.....	15
ВЫВОД.....	17
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ.....	17

ЧАСТЬ 6.1

Условие

Разработайте приложение, которое использует хеш-таблицу (пары «ключ – хеш») для организации прямого доступа к элементам динамического множества полезных данных. Множество реализуйте на массиве, структура элементов (перечень полей) которого приведена в индивидуальном варианте (п.3).

Приложение должно содержать класс с базовыми операциями: вставки, удаления, поиска по ключу, вывода. Включите в класс массив полезных данных и хеш-таблицу. Хеш-функцию подберите самостоятельно, используя правила выбора функции.

Реализуйте расширение размера таблицы и рехеширование, когда это требуется, в соответствии с типом разрешения коллизий.

Предусмотрите автоматическое заполнение таблицы 5-7 записями.

Реализуйте текстовый командный интерфейс пользователя для возможности вызова методов в любой произвольной последовательности, сопроводите вывод достаточными для понимания происходящего сторонним пользователем подсказками.

Проведите полное тестирование программы (все базовые операции, изменение размера и рехеширование), тест-примеры определите самостоятельно. Результаты тестирования включите в отчет по выполненной работе.

Метод решения

Для начала мы создадим генератор случайных значений для нашей хеш-таблицы (рис. 1). Мы создаём сначала шестизначное число (ключ к данным в обычной таблице), а затем переходим к созданию названия товара и его цены.

После генерации данных мы переводим их в бинарный файл, откуда после считываем значения в нашу хеш-таблицу (рис. 2).

```

//ст генерация txt файла
int n = 17; // кол-во строк в txt файле
for (int i=0; i < n; i++)
{
    tempText = "";
    temp = rand() % 1000000 + 1000000;
    writer << temp << ' ';
    for (int j = 0; j < (rand() % 6 + 3); j++)
    {
        if (j == 0)
            writer << (char)(rand() % 26 + 65);
        else
            writer << (char)(rand() % 26 + 97);
    }
    writer << ' ' << rand() % 101 + 100 << '\n';
}
writer.close();

//end генерация txt файла

```

Рисунок 1 – Генерация данных хеш-таблицы

```

hashClass* h = new hashClass();

//ст считывание чисел из txt и их запись в bin
ifstream reader("t.txt");
ofstream writerBin("t.bin", ios_base::binary);

for (int i = 0; i < n; i++) {
    getline(reader, tempText);
    h->add(tempText);
    tempText = tempText.substr(0, 7);
    temp = 0;
    for (int j = 0; j < tempText.size(); j++)
        temp += ((int)tempText[j] - 48) * pow(10, tempText.size() - j - 1);
    writerBin.write((char*)&temp, sizeof(temp));
}

writerBin.close();
//ст считывание чисел из txt и их запись в bin

ifstream readerBin("t.bin", ios_base::binary);
for (int i = 0; i < n; i++) {
    readerBin.read((char*)&temp, sizeof(int));
    // cout << temp << endl;
}

readerBin.close();
reader.close();

```

Рисунок 2 – Считывание данных в хеш-таблицу

Теперь же следует более подробно описать создание самой хеш-таблицы: как-никак – это и есть вся данная работа. В самом начале следует понять, какой же конструктор нужен для данной работы хеш-таблицы. В итоге на рис. 3 представлен конечный результат размышлений по данному поводу.

```
hashClass(){ // конструктор класса
    for (int i = 0; i < 10; i++)
    {
        arrKey[i] = -1;
    }
}
```

Рисунок 3 – Конструктор хеш-таблицы

Для работы с хеш-таблицей был выделен отдельный класс, так как в данной работе это более рациональный подход к решению поставленной задачи: более простая реализация различных методов и возможность создавать множество независимых друг от друга экземпляров данного класса

В первую очередь задаётся длина хеш-таблицы и заполнение заранее пустых ячеек определёнными значениями для отслеживания возможности вставки элементов.

Теперь же распишем все методы нашего класса, которые потребуются в дальнейшем выполнении задания (рис. 4-6).

```

public:
    int arrSize = 10;
    int* arrKey = new int [arrSize]; // хранит захешированные ключи
    string* arrUsefulInf = new string[arrSize]; // хранит полезную информацию

    void del(int key, int collInd=-1) { // удаляет элемент по ключу
        int tempInd = search(key, collInd);

        cout << "Вы удалили " << this->arrUsefulInf[tempInd] << endl;
        arrKey[tempInd] = -1;
        arrUsefulInf[tempInd] = "";
    }

    int search(int key, int collInd=-1) // ищет индекс полезной информации по ключу
    {
        if (collInd == -1)
        {
            for (int i = 0; i < arrSize; i++)
            {
                if (arrKey[i] == key)
                {
                    return i;
                    cout << "Вы нашли " << arrUsefulInf[i] << endl;
                }
            }
        }
        else
        {
            int stInd=0;
            for (int i = 0; i < arrSize; i++)
            {
                if (arrKey[i] == key)
                {
                    stInd = i;
                    break;
                }
            }

            cout << "Вы нашли " << arrUsefulInf[stInd + 7 * collInd] << endl;
            return stInd + 7*collInd;
        }
    }
}

```

Рисунок 4 – 1-ая часть всех методов класса хеш-таблицы

На данной иллюстрации (рис4) изображены все необходимые для правильной работы класса поля, а также методы удаления элементов по ключу (del) и поиска по ключу (search). del и search устроены таким образом, что они будут искать и удалять элементы в зависимости от их коллизии, которую приходится, либо где-то хранить, либо постоянно находить, либо попросту знать.

```

void add(string fullInformation) { // добавляем элемент в список
    int tempKey = hashFunc(convertToInt(fullInformation.substr(0, 6)));

    int collisionInd = collisionManager(tempKey);

    if (collisionInd == -1) {
        for (int i = 0; i < arrSize; i++)
        {
            if (arrKey[i] == -1)
            {
                if (i >= 3 * arrSize / 4)
                    this->arrReHash();
                arrKey[i] = tempKey;
                arrUsefulInf[i] = fullInformation;
                return;
            }
        }

        while (arrKey[collisionInd] != -1)
        {
            collisionInd += 7;
            if (3 * (this->arrSize) / 4 <= collisionInd)
                this->arrReHash();
        }

        arrKey[collisionInd] = tempKey;
        arrUsefulInf[collisionInd] = fullInformation;
    }
}

```

Рисунок 5 – 2-ая часть всех методов класса хеш-таблицы

На текущем же рисунке можно увидеть полноценный метод добавления элементов в нашу хеш-таблицу (add). Метод add получает на вход добавляемую строку и совершает проверку на наличие коллизий, и в самом конце, удостоверившись, что текущая ячейка свободна, добавляет в неё элемент в зависимости от образовавшейся коллизии.

```

int collisionManager(int key) { // метод нахождения свободного места в случае коллизии
    int c = 7;
    int i = 0;
    int stInd = -1;
    for (int j = 0; j < this->arrSize; j++)
    {
        if (this->arrKey[j] == key){
            stInd = j;
            break;
        }
    }
    if (stInd == -1)
        return -1;
    while (true) {
        if (3 * (this->arrSize) / 4 <= (c * i + stInd))
            this->arrReHash();
        if (this->arrKey[c * i + stInd] == key && arrKey[c * i + stInd] != -1)
            i++;
        else
            return c * i + stInd;
    }
}

void arrReHash() { // рехеширование таблицы и массива
    this->arrSize *= 2;
    //int* tempMiniArr = new int[2];
    string* tempArrUsefulInf = new string [this->arrSize];
    int* tempArrKey = new int[this->arrSize];
    for (int i = 0; i < this->arrSize / 2; i++)
    {
        tempArrUsefulInf[i] = this->arrUsefulInf[i];
        tempArrKey[i] = this->arrKey[i];
    }
    for (int i = this->arrSize / 2; i < this->arrSize; i++)
    {
        tempArrKey[i] = -1;
    }
    delete[] this->arrKey;
    delete[] this->arrUsefulInf;
    this->arrKey = tempArrKey;
    this->arrUsefulInf = tempArrUsefulInf;
}

void outArr() { // вывод на экран содержимого хеш-таблицы и массива с полезной информацией
    for (int i = 0; i < this->arrSize; i++)
    {
        cout << this->arrKey[i] << " | " << this->arrUsefulInf[i] << endl;
    }
}

int hashFunc(int num) { // хеш-функция (делает магию)
    return (num % 100 + (num / 100) % 100) % 100;
}

```

Рисунок 6 – 3-я часть всех методов класса хеш-таблицы

И последней частью всех методов является рис. 6, где изображены метод управления коллизией (collisionManager), метод рехеширования таблицы (arrReHash), метод вывода содержимого таблицы на экран (outArr) и метод хеш-функции.

collisionManager лишь вызывается в прочих методах в качестве помощника, ищущего образование различных коллизий.

arrReHash также вызывается исключительно в прочих методах для расширения границ таблицы в зависимости от сложившейся ситуации.

C outArr всё гораздо проще – он выводит на экран всё содержимое имеющейся хеш-таблицы (выводятся даже пустые ячейки).

A hashFunc – это всего-навсего фантазия автора данного кода (то есть меня), которая позволяет создавать уникальные ключи с желательной периодичностью.

Единственное, что осталось добавить, так это UI и на рис. 7 как раз-таки представлен код реализующий тот самый UI.

```
cout << "----- Добро пожаловать в нашу программу! -----" << endl;
while (true) {

    string tempText;

    cout << "1 - Вывести на экран список\t2 - Удалить элемент по ключу\t3 - Добавить элемент\t4 - Найти элемент по ключу\t5 - Очистить консоль\t6 - Завершить работу программы" << endl;

    getline(cin, tempText);
    if (tempText == "1") {
        h->outArr();
    }
    else if (tempText == "2") {
        cout << "Удаляйте по шаблону - \"%00(ключ) 000...(индекс коллизии)\"< endl;
        getline(cin, tempText);
        int tempKey = convertToInt(tempText.substr(0, 2));
        int tempColl = convertToInt(tempText.substr(3, h->arrSize-3));
        h->del(tempKey, tempColl);
    }
    else if (tempText == "3") {
        cout << "Добавляйте элемент по шаблону:" << endl
            << "000000(шестизначное число) аааа...(слово n-ой длины) 000(трёхзначное число)" << endl;
        getline(cin, tempText);
        h->add(tempText);
    }
    else if (tempText == "4"){
        cout << "Ищите по шаблону - \"%00(ключ) 000...(индекс коллизии)\"< endl;
        getline(cin, tempText);
        int tempKey = convertToInt(tempText.substr(0, 2));
        int tempColl = convertToInt(tempText.substr(3, h->arrSize - 3));
        h->search(tempKey, tempColl);
    }
    else if (tempText == "5") {
        system("CLS");
        cout << "----- Добро пожаловать в нашу программу! -----" << endl;
    }
    else if (tempText == "6") {
        cout << "Завершение работы программы" << endl;
        break;
    }
    cout << "-----" << endl;
}
```

Рисунок 7 – Реализация в коде UI

В данном интерфейсе предусмотрены 6 различных ситуаций и, конечно же, если условия в них не будут соблюдены то программа даст сбой, поэтому мы предполагаем, что работаем с заранее обработанными данными.

Теперь стоит перейти к тестированию нашей программы. Результаты тестов будут приведены на рис. 8 – 11.

```

----- Добро пожаловать в нашу программу! -----
1 - Вывести на экран список      2 - Удалить элемент по ключу      3 - Добавить элемент
4 - Найти элемент по ключу      5 - Очистить консоль      6 - Завершить работу программы
1
6 | 117927 Rlxmet 176
89 | 123257 Njsr 179
22 | 111309 Bcfprsl 164
47 | 100245 Ucbh 132
66 | 100660 Mqmv 136
1 | 120794 Qgpz 107
50 | 125199 Skcfq 191
78 | 131959 Lswct 196
11 | 121992 Gqxfyekk 107
72 | 127200 Ney 127
59 | 128970 Bqk 127
39 | 123108 Gxuzaq 135
86 | 118204 Cylb 113
17 | 103879 Ecgww 158
80 | 122951 Zsjd 172
31 | 120625 Ccho 168
58 | 113820 Nri 190
-1 |
-1 |
-1 |
-1 |
-1 |
-1 |

```

Рисунок 8 – Вывод содержимого хеш-таблицы на экран

```

1 - Вывести на экран список      2 - Удалить элемент по ключу      3 - Добавить элемент
4 - Найти элемент по ключу      5 - Очистить консоль      6 - Завершить работу программы
3
Добавляйте элемент по шаблону:
000000(шестизначное число) ааааа...(слово n-ой длины) 000(трёхзначное число)
112236 Collision 188
- - - - -
1 - Вывести на экран список      2 - Удалить элемент по ключу      3 - Добавить элемент
4 - Найти элемент по ключу      5 - Очистить консоль      6 - Завершить работу программы
1
6 | 117927 Rlxmet 176
89 | 123257 Njsr 179
22 | 111309 Bcfprsl 164
47 | 100245 Ucbh 132
66 | 100660 Mqmv 136
1 | 120794 Qgpz 107
50 | 125199 Skcfq 191
78 | 131959 Lswct 196
11 | 121992 Gqxfyekk 107
72 | 127200 Ney 127
59 | 128970 Bqk 127
39 | 123108 Gxuzaq 135
86 | 118204 Cylb 113
17 | 103879 Ecgww 158
80 | 122951 Zsjd 172
31 | 120625 Ccho 168
58 | 113820 Nri 190
-1 |
-1 |
-1 |
-1 |
-1 |
-1 |
58 | 112236 Collision 188
-1 |

```

Рисунок 9 – Добавление элемента в таблицу с учётом коллизии

```

1 - Вывести на экран список      2 - Удалить элемент по ключу      3 - Добавить элемент
4 - Найти элемент по ключу      5 - Очистить консоль      6 - Завершить работу программы
2
Удаляйте по шаблону - "00(ключ) 000....(индекс коллизии)"
58 0
Вы нашли 113820 Nri 190
Вы удалили 113820 Nri 190
-----
1 - Вывести на экран список      2 - Удалить элемент по ключу      3 - Добавить элемент
4 - Найти элемент по ключу      5 - Очистить консоль      6 - Завершить работу программы
1
6 | 117927 Rlxmet 176
89 | 123257 Njsr 179
22 | 111309 Bcfprsl 164
47 | 100245 Ucbh 132
66 | 100660 Mqmv 136
1 | 120794 Qgpz 107
50 | 125199 Skcfq 191
78 | 131959 Lswct 196
11 | 121992 Gqxfyekk 107
72 | 127200 Ney 127
59 | 128970 Bqk 127
39 | 123108 Gxuzaq 135
86 | 118204 Cylb 113
17 | 103879 Ecggw 158
80 | 122951 Zsjd 172
31 | 120625 Ccho 168
-1 |
-1 |
-1 |
-1 |
-1 |
-1 | endl;
-1 |
-1 |
58 | 112236 Collision 188
-1 |
-1 |

```

Рисунок 10 – Удаление элемента из таблицы с учётом коллизии

```

1 - Вывести на экран список      2 - Удалить элемент по ключу      3 - Добавить элемент
4 - Найти элемент по ключу      5 - Очистить консоль      6 - Завершить работу программы
4
Ищите по шаблону - "00(ключ) 000....(индекс коллизии)"
58 0
Вы нашли 112236 Collision 188

```

Рисунок 11 – Нахождение элемента с учётом коллизии

Тесты проведены и, как можно заметить, успешно – программы работает исправно и выдаёт желаемый результат без сбоев, что и требовалось доказать в качестве работоспособности данной программы.

ЧАСТЬ 6.2

Условие

Разработайте приложения в соответствии с заданиями в индивидуальном варианте (п.2).

В отчёте в разделе «Математическая модель решения (описание алгоритма)» разобрать алгоритм поиска на примере. Подсчитать количество

сравнений для успешного поиска первого вхождения образца в текст и безуспешного поиска.

Определить функцию (или несколько функций) для реализации алгоритма поиска. Определить предусловие и постусловие.

Сформировать таблицу тестов с указанием успешного и неуспешного поиска, используя большие и небольшие по объему текст и образец, провести на её основе этап тестирования.

Оценить практическую сложность алгоритма в зависимости от длины текста и длины образца и отобразить результаты в таблицу (для отчета).

ЗАДАНИЕ 1

Формулировка задачи

Дано предложение, состоящее из слов. Найти самое длинное слово предложения, первая и последняя буквы которого одинаковы.

Математическая модель решения

Мы будем получать на вход строку с разными разделителями, от которых мы впоследствии избавимся при помощи нашего собственного метода, который будет искать разделители между словами.

В конечном итоге вся работа будет заключаться в прохождении по строке слева направо в поиске слов, чей первый и последний символ одинаковы. Мы добьемся распознавания этих символов при помощи отдельных полей, отвечающих за начало и конец слов.

Код программы с комментариями

На рис. 12 представлен код программы для решения задания 1.

Сначала идёт метод `find`, осуществляющий поиск разделителей и возвращает значение индекса этого разделителя.

Метод же `search1` ответственен за поиск самого длинного слова с повторяющимися первым и последним символами. Поля `longest` и `tempStr` нужны для хранения длиннейшего слова и временного слова, которое будет выведено проверку, соответственно. Поля `start` и `end` служат ограничителями начала и конца слова.

Работа метода следующая: до тех пор, пока мы не дошли до конца строки, идёт проверка первого и последнего символа слов, с последующим нахождением их длины в удовлетворительном случае, а также с поиском следующего разделителя. В случае нахождения конца строки идет простая

проверка первого и последнего символа слова. И в конце возвращается самое длинное слово с повторяющимися первым и последним символами.

```
int find(string text, int start, int end, char searchedSymbol) {
    for (int i = start; i < end; i++) {
        if (text[i] == searchedSymbol)
            return i;
    }
    return -1;
}

string search1(string text) {
    string longest = "";
    string tempStr = "";
    int start = 0;
    int end = 0;
    while (true) {
        end = find(text, start, text.size(), ' ');
        if (end == -1)
        {
            if (toupper(text[start]) == toupper(text[text.size()-1]) && (text.size() - start) > longest.size())
            {
                longest = "";
                for (int i = start; i < text.size(); i++)
                {
                    longest += text[i];
                }
            }
            break;
        }
        else
        {
            if (toupper(text[start]) == toupper(text[end - 1]) && ((end - start) > longest.size()))
            {
                longest = "";
                for (int i = start; i < end; i++)
                {
                    longest += text[i];
                }
            }
            start = end + 1;
        }
    }
    return longest;
}
```

Рисунок 12 – Код программы для задания 1

Результаты тестирования

Сделаем таблицу 1 с результатами всех вычислений.

Таблица 1 – Результаты тестирования с маленьким и большим текстами

	Кол-во итераций	
	Маленький текст	Большой текст
Худший случай	24	350
Лучший случай	14	50

Всевозможные случаи будут представлены на рис. 13-16.

Для маленького текста возьмём худший случай как «**jij somotos ege sfadds abba dfdffgdd**», где мы будем обрабатывать каждое слово, ведь все они имеют

Всё аналогично предыдущим тестам: лучший случай = кол-во слов + длина самого большого слова, худший = кол-во слов + длина всех слов.

ЗАДАНИЕ 2

Формулировка задачи

Используя алгоритм Кнута-Мориса-Пратта, найти индекс последнего вхождения образца в текст.

Математическая модель решения

Мы идём по каждому элементу нашего текста и поочерёдно сравниваем символы образца и элемента текста, в конечном итоге либо получая, либо нет искомый образец из текста.

Код программы с комментариями

На рис. 17 показан код программы для задания 2.

```
int search2(string example, string text) {
    for (int i = text.size() - example.size(); i > -1 ; i--) {
        for (int j = 0; j < example.size(); j++)
        {
            if (text[i + j] != example[j])
                break;
            if (j == example.size() - 1)
                return i;
        }
    }
    return -1;
}
```

Рисунок 17 – Код алгоритма Кнута-Мориса-Пратта

Как и говорилось в математической модели, мы поочерёдно проходимся по всем словам нашего текста в поиске желанного образца и, если в какой-то момент текущее слово не совпадает с образцом, мы тут же прекращаем проверку этого слова. В случае, если мы ничего не находим, то программа возвращает -1, как индекс несуществующего образца.

Результаты тестирования

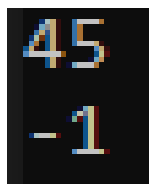
Сделаем таблицу 2 с результатами всех вычислений.

Таблица 2 – Результаты тестирования с маленьким и большим текстами

	Кол-во итераций	
	Маленький текст	Большой текст
Худший случай	45	829
Лучший случай	5	50

Результаты всех тестов представлены на рис. 18-21.

В качестве примера худшего случая для маленького текста примем текст «slov slov slov slov slov slov» с образцом «slovo».



45
-1

Рисунок 18 – Результат работы программы в худшем случае с маленьким текстом

А в качестве лучшего случая примем текст «slov slov slov slov slov slov».



5
25

Рисунок 19 – Результат работы программы в лучшем случае с маленьким текстом

Как можно заметить, кол-во итерация составило всего длину образца, а также мы получили индекс начала искомого слова.

В качестве большого текста возьмём условие практики 6. А в качестве образца укажем слово «рыба».



829
-1

Рисунок 20 – Результат работы программы в худшем случае с большим текстом

Данный случай является приблизительно худшим, так как не состоит исключительно из слов «рыб», что вызвало бы дополнительные итерации.



4
801

Рисунок 21 – Результат работы программы в лучшем случае с большим текстом

Как видно из предыдущего образца кол-во итераций составило всего длину самого образца.

Из выше перечисленных примеров можно сделать вывод, что в худшем случае кол-во итерация = $(\text{длина образца} * 2 - 1) * (\text{кол-во слов} - 1)$. А лучшем случае = длина образца.

ВЫВОД

В практической работе 6.1 изучили методы работы с хеш-таблицей, а также выработали навыки необходимые для создания подобного рода таблиц, решив поставленную задачу.

В практической работе 6.2 разработали методы поиска слов в тексте при помощи различных инструментов, а также осуществили оценку полученных результатов в ходе пробных тестов программ.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
2. Документация по языку C++ [Электронный ресурс]. URL: <https://docs.microsoft.com/ruru/cpp/cpp/> (дата обращения 01.09.2021).
3. Курс: Структуры и алгоритмы обработки данных. Часть 2 [Электронный ресурс]. URL: <https://online-edu.mirea.ru/course/view.php?id=4020> (дата обращения 01.09.2021).