МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания № 6, часть 2

Тема:

«Быстрый доступ к данным с помощью хеш-таблиц»

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Лисовский И.В

Группа: ИКБО-21-23

Москва – 2024

СОДЕРЖАНИЕ

1 ЦЕЛЬ

Освоить и закрепить навыки реализации алгоритмов поиска образца в тексте, таких как алгоритмы Бойера-Мура и Рабина-Карпа. Исследовать их эффективность и определить количество сравнений для успешного и безуспешного поиска. Оценить практическую сложность алгоритмов и научиться формировать и анализировать тестовые данные.

2 ЗАДАНИЕ

2.1 Формулировка задачи

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).

В отчёте сделайте вывод о проделанной работе, основанный на полученных результатах.

Вариант №19:

4	1. Дано предложение, состоящее из слов, разделенных знаками препинания. Определить, сколько раз в предложение входит первое слово.
	2. Проверка на плагиат. Используя алгоритм Рабина-Карпа, проверить, входит ли подстрока проверяемого текста в другой текст.

2.2 Математическая модель решения задачи 1

1. **Определение первого слова:**

Выделение первого слова из строки текста. Для этого необходимо итерироваться по строке и собирать символы, пока не встретится первый разделитель (например, пробел или запятая).

```
string firstWord(string Text)   //для нахождения первого слова
{
    string output;
    for (char iter : Text)
    {
        if (iter == ' ' or iter == ',')
            return output;
        output += iter;
    }
}
```

Листинг 1.1 — Функция для нахождения первого слова

2. **Построение таблицы смещений для алгоритма Бойера-Мура:**

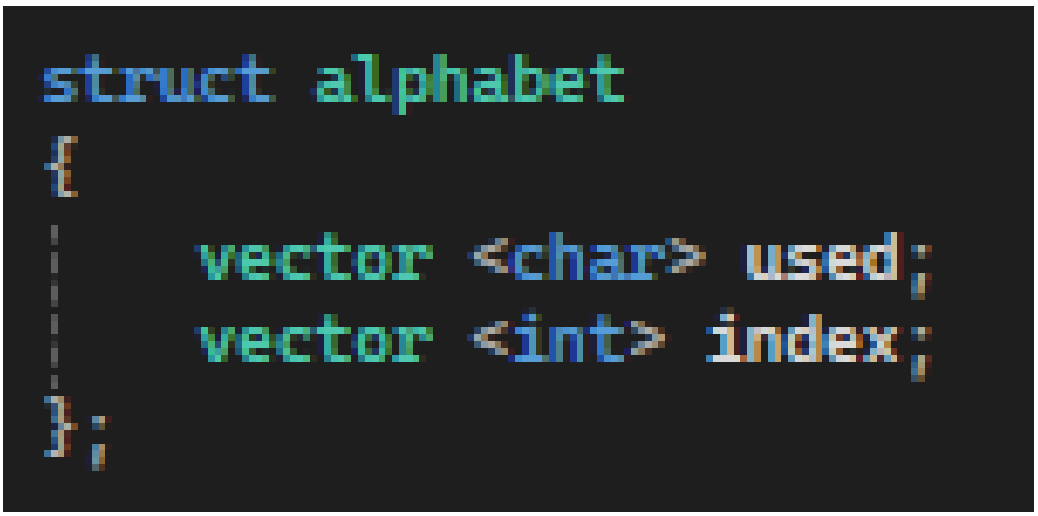
Построение таблицы смещений для символов из первого слова. Она помогает определить, на сколько символов можно сдвинуться вперед при несоответствии символов при сравнении текстовых строк.

Таблица строится следующим образом:

- Для каждого уникального символа, начиная с конца слова, вычисляется максимально возможное смещение, так чтобы пропущенное в сравнении символов не может быть началом новой строки.
- Если символ уникален, его смещение равно длине слова.

Формально для символа c в слове Word:

$$\text{shift}[c] = \begin{cases} \text{len}(\text{Word}) & \text{, если } c \text{ уникален} \\ \text{len}(\text{Word}) - 1 - \text{pos}(c) & \text{, иначе} \end{cases}$$



```
struct alphabet
{
    vector <char> used;
    vector <int> index;
};
```

Листинг 1.2 — Структура алфавита слова

```
alphabet table_index(string Word)
{
    alphabet ex;
    if (found_count(Word[Word.length() - 1], Word) == 1)    /*если последняя буква уникальна,
                                                             то присвоение индекса длины - 1*/
    {
        ex.used.push_back(Word[Word.length() - 1]);
        ex.index.push_back(Word.length() - 1);
    }
    for (int i = Word.length() - 2; i >= 1; i--)
    {
        if (!found(Word[i], ex.used))
        {
            ex.used.push_back(Word[i]);
            ex.index.push_back(Word.length() - 1 - i);
        }
    }

    ex.used.push_back(Word[0]);     //добавляем первую букву слова
    ex.index.push_back(Word.length() - 1);
    return ex;
}
```

Листинг 1.3 — Функция для создания алфавита

3. **Поиск первого слова в тексте:**

Использование алгоритма Бойера-Мура для поиска всех вхождений первого слова в текст. Сравнивание начинается с конца слова и идет назад. При несоответствии производится сдвиг, основанный на таблице смещений. Для текста и слова Word повторять до конца текста: Если текст [i+j]=Word[j], для всех j от len(Word) − 1 до 0, то найдено вхождение, иначе сдвинуться на текст[i+len(Word)−1].

```
int find_index(string word, string text)
{
    int index = 0;
    int count = 0;
    bool flag = true;
    alphabet ex = table_index(word);
    while (index + word.length() < text.length())
    {

        int offset = 0;
        while (offset < word.length())
        {
            if (text[index + word.length() - 1 - offset] == word[word.length() - 1 - offset])
            {
                offset++;
                if (offset == word.length())
                {
                    count++;
                    index += word.length();
                    cout << print_space(index) << word << "\n";
                    break;
                }
            }
            else if (text[index + word.length() - 1 - offset] != word[word.length() - 1 - offset])
            {
                index += index_from_alphabet(text[index + word.length() - 1 - offset],ex);
                cout << print_space(index) << word << "\n";
                break;
            }
        }
    }
    return count;
}
```

Листинг 1.4 — Функция для нахождения слова с помощью алгоритма БМ

```
int index_from_alphabet(char ex, alphabet alph)
{
    for (int i = 0; i < alph.used.size(); i++)
    {
        if (ex == alph.used[i])
        {
            return alph.index[i];
        }
    }
    return alph.index[alph.index.size() - 1];
}
```

Листинг 1.5 — Вспомогательная функция для возврата значения смещения из алфавита

2.3 Код программы 1 с комментариями

```cpp
#include <iostream>
#include <string>
#include <vector>
using namespace std;

struct alphabet
{
    vector <char> used;
    vector <int> index;
};

string firstWord(string Text)   //для нахождения первого слова
{
    string output;
    for (char iter : Text)
    {
        if (iter == ' ' or iter == ',')
            return output;
        output += iter;
    }
}

string print_space(int count)   //для вывода пробелов (посмотреть на смещения
{
    string spaces = "";
    for (int i = 0; i < count; i++)
        spaces += " ";
    return spaces;
}

int index_from_alphabet(char ex, alphabet alph) //если найдется буква из алфавита,
                                                //возвращается она, иначе последний элемент (смещение длиной слова)
{
    for (int i = 0; i < alph.used.size(); i++)
    {
        if (ex == alph.used[i])
        {
            return alph.index[i];
        }
    }
    return alph.index[alph.index.size() - 1];
}

//БЛОК ДЛЯ СОСТАВЛЕНИЯ ТАБЛИЦЫ ИНДЕКСОВ

int found_count(char example, string ex)    //подсчет количества букв в слове (для уникальности)
{
    int count = 0;
    for (char iter : ex)
    {
        if (example == iter)
            count++;
    }
    return count;
}

bool found(char example, vector <char> ex)  //для проверки, нашлась ли заданная буква в векторе букв алфавита
{
    for (char iter : ex)
    {
        if (example == iter)
            return true;
    }
    return false;
}
```

Листинг 1.6 — Код программы

```
alphabet table_index(string Word)
{
    alphabet ex;
    if (found_count(Word[Word.length() - 1], Word) == 1)    /*если последняя буква уникальна,
                                                            то присвоение индекса длины - 1*/
    {
        ex.used.push_back(Word[Word.length() - 1]);
        ex.index.push_back(Word.length() - 1);
    }
    for (int i = Word.length() - 2; i >= 1; i--)    //добавление букв слова в алфавит с проверкой на уникальность
    {
        if (!found(Word[i], ex.used))
        {
            ex.used.push_back(Word[i]);
            ex.index.push_back(Word.length() - 1 - i);
        }
    }

    ex.used.push_back(Word[0]);     //добавляем первую букву слова
    ex.index.push_back(Word.length() - 1);  //добавляем для нее смещение длиной всего слова
    return ex;
}

int find_index(string word, string text)    //нахождение индекса вхождения (в итоге находит количества вхождений слова)
{                                           //отображение смещений включено
    int index = 0;
    int count = 0;
    bool flag = true;
    alphabet ex = table_index(word);
    while (index + word.length() < text.length())
    {

        int offset = 0;     //переменная показывающая итерации по слову
                            //(для проверки, если к примеру сошлись последние буквы, то идет смещение)
        while (offset < word.length())      //пока смещение меньше длины всего слова, то будет идти проверка по всему слову с текстом
        {
            if (text[index + word.length() - 1 - offset] == word[word.length() - 1 - offset])   //если последние буквы сошлись, то идет
                                                                                                //по слову полностью
            {
                offset++;
                if (offset == word.length())        //если полностью совпало слово, то увеличиваем количество вхождений слова в текст на один
                                                    //и перешагиваем через все слово, чтобы не делать лишних проверок
                {
                    count++;
                    index += word.length();
                    cout << print_space(index) << word << "\n";
                    break;
                }
            }
            else if (text[index + word.length() - 1 - offset] != word[word.length() - 1 - offset])  //если буква не совпала, то перешагиваем
                                                                                                    //на значение из функции возвращающей размер смещения
            {
                index += index_from_alphabet(text[index + word.length() - 1 - offset],ex);
                cout << print_space(index) << word << "\n";
                break;
            }
        }
    }
    return count;
}
```

Листинг 1.7 — Код программы

```
int main()
{
    setlocale(0, "");
    //пример - string text = "buffalo, named Buffalo, is from Buffalo, buffalo, other buffaloes from Buffalo.";
    string text;
    getline(cin, text);
    string first_word = firstWord(text);
    cout << text << "\n";
    cout << first_word << "\n";
    cout << find_index(first_word, text);
}
```

Листинг 1.8 — Код программы

2.4 Тестирование программы 1

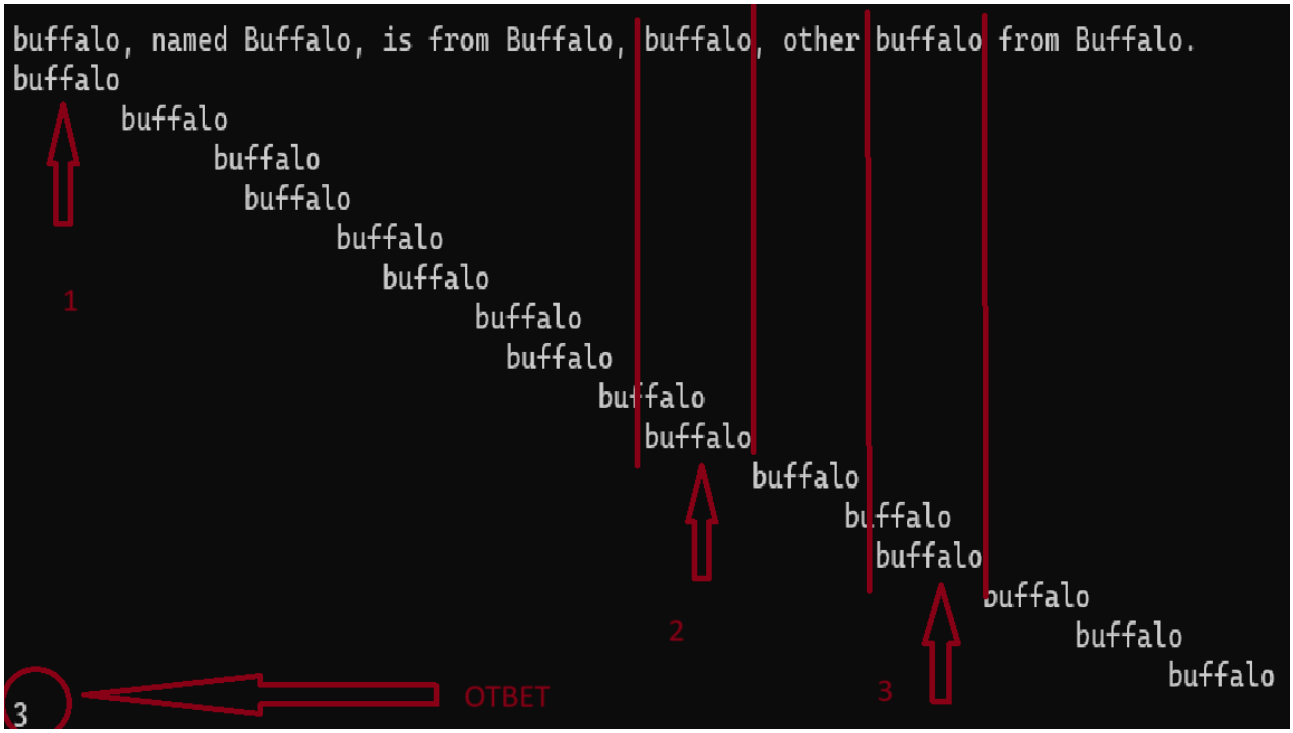


Рисунок 1 — Тестирование кода

2.5 Математическая модель решения задачи 2

1. **Хеш-функция:**

Для каждой подстроки вычисляется хеш-код, который преобразует строку в числовое значение для быстрого сравнения. Используется полиномиальная хеш-функция.

```
int hashfunc(string word)
{
    int sum = 0;
    for (int i = 0; i < word.length(); i++ )
    {
        sum += word[i] * pow(11, word.length() - i - 1);
    }
    return sum;
}
```

Листинг 2.1 — Функция для вычислений хеша

2. **Алгоритм Рабина-Карпа:**

Вычисляется хеш-код искомой подстроки. Сравниваются хеш-коды подстроки и каждого окна в исходном тексте. Если хеши совпадают, выполняется дополнительное посимвольное сравнение для подтверждения. Для каждой позиции в исходном тексте вычисляется хеш-код подстроки той же длины, что и искомая. Если хеши совпадают, производится посимвольное сравнение подстроки и образца. Если совпадение найдено, возвращается индекс начала вхождения. Если совпадение не найдено, продолжается поиск. Если подстрока не найдена, то возвращается -1. Алгоритм Рабина-Карпа по сложности в худшем O(n*m) случае имеет сложность , по времени работы ничем сильно не выделяется, но при этом делает больше сравнений. Ведь перед тем как сверять строку мы сначала сверяем хеш значения, а если и находится, то целую

подстроку.

```
int RK_func(string text, string sub)
{
    string iter = "";
    int hsub = hashfunc(sub), htext;
    int start = 0, stop = sub.length();
    while (start != text.length() - sub.length())
    {
        for (int i = start; i < stop; i++)
        {
            iter += text[i];
        }
        htext = hashfunc(iter);
        cout << iter << "  " << htext << "\n";
        if (hsub == htext)
        {
            if (iter == sub)
            {
                return start;
            }
        }
        start++;
        stop++;
        iter = "";
    }
    return -1;

}
```

Листинг 2.2 — Функция, реализовывающая алгоритм Рабина-Карпа

2.6 Код программы 2 с комментариями

```cpp
#include <iostream>
#include <string>
using namespace std;

int hashfunc(string word)       //полиномиальная хеш-функция
{
    int sum = 0;
    for (int i = 0; i < word.length(); i++ )
    {
        sum += word[i] * pow(11, word.length() - i - 1);
    }
    return sum;
}

int RK_func(string text, string sub)
{
    string iter = "";       //итерируемое слово(хеш), длиной заданной подстроки
    int hsub = hashfunc(sub), htext;
    int start = 0, stop = sub.length();     //начало с 0 индекса, по индекс - длина подстроки
    while (start != text.length() - sub.length())
    {
        for (int i = start; i < stop; i++)
        {
            iter += text[i];            //в итерируемую строку добавляем символы, в результате - строка длиной заданной подстроки
        }
        htext = hashfunc(iter);     //вычисляется хеш для этой пробной строки
        cout << iter << "  " << htext << "\n";  //вывод временной подстроки и ее хеша для наглядности алгоритма
        if (hsub == htext)  //если хеш заданной подстроки равен хеш временной-итерируемой подстроки
        {
            if (iter == sub)        //проверка, равны ли тогда эти слова? если да, то возврат индекса начала этой строки в тексте
            {
                return start;
            }
        }
        start++;            //иначе, смещение на 1 в тексте, обнуление итерируемой строки
        stop++;
        iter = "";
    }
    return -1;      //если не найдено, возврат -1 - код ошибки

}
int main()
{
    setlocale(0, "");
    //для проверки - string text = "There is something incredibly special about the arrival of summer. The air becomes warmer, and the days grow longer";
    //cout << text << "\n";
    string sub;
    string text;
    cout << "Введите текст: ";
    getline(cin, text);
    cout << "Введите образец для нахождения: ";
    getline(cin,sub);
    if (RK_func(text, sub) == -1)
    {
        cout << "Строка не найдена";
    }
    else
    {
        cout << "Строка найдена. Индекс первой буквы: " << RK_func(text, sub);
    }
}
```

Листинг 2.3 — Код программы

2.7 Тестирование программы 2

```
There is something incredibly special about the arrival of summer. The air becomes warmer, and the days grow longer
Введите образец для нахождения: special about the arrival of
Хеш образца: 124973857
There is something incredibly  1587408
here is something incredibly   270768781
ere is something incredibly s  16380752
re is something incredibly sp  4777883
e is something incredibly spe  91505194
 is something incredibly spec  -26288792
is something incredibly speci  178489535
s something incredibly specia  287462314
 something incredibly special  -1300502559
something incredibly special   309824171
omething incredibly special a  231631143
mething incredibly special ab  14584479
ething incredibly special abo  -14981121
thing incredibly special abou  88515047
hing incredibly special about  -59180392
ing incredibly special about   31040743
ng incredibly special about t  -1334474379
g incredibly special about th  -63865777
 incredibly special about the  -20498423
incredibly special about the   242183521
ncredibly special about the a  345019517
credibly special about the ar  190062204
redibly special about the arr  -1300109358
edibly special about the arri  314149455
dibly special about the arriv  64850407
ibly special about the arriva  -105132570
bly special about the arrival  -1331868663
ly special about the arrival   -35202973
y special about the arrival o  294792431
 special about the arrival of  280640889
special about the arrival of   124973857
There is something incredibly  1587408
here is something incredibly   270768781
ere is something incredibly s  16380752
re is something incredibly sp  4777883
e is something incredibly spe  91505194
 is something incredibly spec  -26288792
is something incredibly speci  178489535
s something incredibly specia  287462314
 something incredibly special  -1300502559
something incredibly special   309824171
omething incredibly special a  231631143
mething incredibly special ab  14584479
ething incredibly special abo  -14981121
thing incredibly special abou  88515047
hing incredibly special about  -59180392
ing incredibly special about   31040743
ng incredibly special about t  -1334474379
g incredibly special about th  -63865777
 incredibly special about the  -20498423
incredibly special about the   242183521
ncredibly special about the a  345019517
credibly special about the ar  190062204
redibly special about the arr  -1300109358
edibly special about the arri  314149455
dibly special about the arriv  64850407
ibly special about the arriva  -105132570
bly special about the arrival  -1331868663
ly special about the arrival   -35202973
y special about the arrival o  294792431
 special about the arrival of  280640889
special about the arrival of   124973857
Строка найдена. Индекс первой буквы: 30
```

Рисунок 2 — Тестирование кода

3 ВЫВОД

В ходе выполнения практической работы были реализованы и изучены два алгоритма поиска подстрок: Бойера-Мура и Рабина-Карпа. Эти алгоритмы продемонстрировали свои сильные и слабые стороны в контексте различных задач. Оба алгоритма значительно расширили понимание различных подходов к задаче поиска подстрок.

1. **Алгоритм Бойера-Мура:**

Эффективность: Этот алгоритм показал высокую скорость поиска за счёт использования таблицы смещений, что позволило уменьшить количество сравнений символов.

Сложность: В лучшем случае алгоритм работает за время близкое к линейному, что делает его полезным при анализе больших текстовых массивов.

Применимость: Подходит для задач, где требуется найти частое и регулярное вхождение подстрок, как в случае с поиском первого слова в тексте.

2. **Алгоритм Рабина-Карпа:**

Эффективность: Предоставил удобный способ поиска подстрок за счёт хеширования, что позволяет быстро отсеять несовпадающие окна.

Сложность: Хотя теоретически он работает в среднем за линейное время, на практике может иметь коллизии хешей, что требует дополнительных проверок.

Применимость: Идеально подходит для задач, требующих проверки на плагиат, где необходимо многократно проверять сходства между большими текстами.

Реализация данных методов позволила глубже понять их теоретическую основу и увидеть, как на практике они решают задачи различной специфики и сложности. Полученные результаты подтвердили эффективность каждого из

алгоритмов в соответствующих контекстах, что будет полезно при выборе подходящего метода для различных практических применений.

4 ЛИТЕРАТУРА

1. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона, 2010.
2. Кнут Д. Искусство программирования. Тома 1-4, 1976-2013.
3. Бхаргава А. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих, 2017.
4. Кормен Т.Х. и др. Алгоритмы. Построение и анализ, 2013.
5. Лафоре Р. Структуры данных и алгоритмы в Java. 2-е изд., 2013.
6. Макконнелл Дж. Основы современных алгоритмов. Активный обучающий метод. 3-е доп. изд., 2018.
7. Скиена С. Алгоритмы. Руководство по разработке, 2011.
8. Хайнеман Д. и др. Алгоритмы. Справочник с примерами на C, C++, Java и Python, 2017.
9. Гасфилд Д. Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология, 2003.

По языку C++:

10. Страуструп Б. Программирование. Принципы и практика с использованием C++. 2-е изд., 2016.
11. Павловская Т.А. C/C++. Программирование на языке высокого уровня, 2003.
12. Прата С. Язык программирования C++. Лекции и упражнения. - 6-е изд., 2012.
13. Седжвик Р. Фундаментальные алгоритмы на C++, 2001-2002
14. Хортон А. Visual C++ 2010. Полный курс, 2011.
15. Шилдт Г. Полный справочник по C++. 4-е изд., 2006.