



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 6-2

**Тема: «Поиск образца в тексте»**

Дисциплина: Структуры и алгоритмы обработки данных

Выполнил студент Фамилия И.О.  
Группа AAAA-00-00

**Москва 2024**

## СОДЕРЖАНИЕ

1 ВВЕДЕНИЕ.....	3
1.1 Цель работы.....	3
1.2 Индивидуальный вариант.....	3
2 ХОД РАБОТЫ.....	4
2.1 Задание 1.....	4
2.2 Задание 2.....	5
5 ВЫВОД.....	8
6 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ.....	9

## 1 ВВЕДЕНИЕ

### 1.1 Цель работы

Освоить приёмы реализации алгоритмов поиска образца в тексте.

### 1.2 Индивидуальный вариант

Таблица 1 — Индивидуальный вариант

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

## 2 ХОД РАБОТЫ

### 2.1 Задание 1

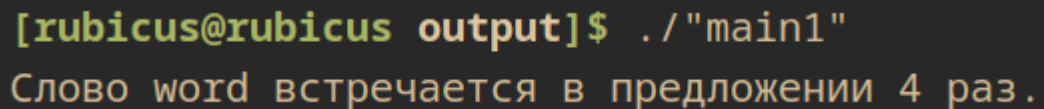
Первое задание представляет собой обычный линейный поиск. Получим первое слово из предложения, и будем сравнивать слова после каждой встречающейся запятой. Если слова посимвольно равны, то слово найдено.

Количество сравнений при безуспешном поиске в лучшем случае будет равно длине  $n$  - исходной строки, в худшем  $n * m$ , где  $m$  размер поисковой строки.

Далее будем вести подсчёт количества совпадений. Код программы представлен в листинге 1. Результат тестирования на рис. 1.

Листинг 1 - Решение задачи

```
#include <iostream>
#include <sstream>
using namespace std;
int main()
{
    string sentence =
"word,wmeo,vowmbvme,vwokef,word,wrvimwe,word,foejgo,rog,word";
    istringstream ss(sentence);
    string first_word, word;
    int count = 1;
    getline(ss, first_word, ',');
    while (getline(ss, word, ','))
    {
        if(word == first_word) count += 1;
    }
    cout << "Слово " << first_word << " встречается в
предложении " << count << " раз." << endl;
    return 0;
}
```



```
[rubicus@rubicus output]$ ./"main1"
Слово word встречается в предложении 4 раз.
```

Рисунок 1 - Результаты тестирования

## 2.2 Задание 2

Во втором задании просят реализовать алгоритм Рабина-Карпа для поиска плагиата в тексте. Для реализации алгоритма необходима хеш-функция для строки. Воспользуемся полиномиальной нарастающей хеш-функцией. Описание функции представлено на рис. 2.

$$\begin{aligned}\text{hash}(s) &= s[0] + s[1] \cdot p + s[2] \cdot p^2 + \dots + s[n-1] \cdot p^{n-1} \mod m \\ &= \sum_{i=0}^{n-1} s[i] \cdot p^i \mod m,\end{aligned}$$

Рисунок 2 - Полиномиальная хеш функция

где  $s$  — исходная строка,  $n$  — длина строки,  $p$  и  $m$  — случайные положительные числа. Реализуем данную хеш-функцию. Код функции представлен в листинге 2.

Листинг 2 - Полиномиальная хеш функция

```
long long s_hash(string const& s) {
    const int p = 31;
    const int m = 1e9 + 9;
    long long hash_value = 0;
    long long p_pow = 1;
    for (char c : s) {
        hash_value = (hash_value + (c - 'a' + 1) * p_pow) % m;
        p_pow = (p_pow * p) % m;
    }
    return hash_value;
}
```

Далее реализуем алгоритм поиска Рабина-Карпа. Линейно проходим по исходной строке и берём хеши от подстрок нужной длины. Если хеши исходной подстроки и поисковой совпали, то производим посимвольное сравнения во избежание ложного совпадения. Если сравнение удалось, то подстрока найдена и возвращаем найденный индекс. Реализация функции поиска представлена в листинге 3.

Алгоритм Рабина-Карпа хорош настолько, насколько хороша его хеш функция. В худшем случае даёт количество сравнение  $n * m$  как и линейный поиск. Но в зависимости от хеш в функции, в лучшем и среднем  $n$ .

### Листинг 3 - Алгоритм поиска Рабина-Карпа

```
int RK_search(string str, string substr)
{
    int n = str.length(), m = substr.length();

    long long hsub = s_hash(substr);
    long long hs = s_hash(str.substr(0, m));

    for (int i = 1; i < n - m + 1; i++)
    {
        if (hs == hsub)
        {
            if (str.substr(i, m) == substr)
                return i;
        }
        hs = s_hash(str.substr(i + 1, m));
    }
    return -1;
}
```

В главной функции программы используем данный поиск для нахождения подстроки в заданном тексте. Код программы представлен в листинге 4. Результаты тестирования на рис. 3. Таблица сравнений при разных случаях поиска представлена в таблице 2. Стоит сказать, что разница между измерениями в разных случаях получилась настолько мала, что её можно счесть за погрешность, и вывод из таких данных сделать затруднительно.

### Листинг 4 - Код программы

```
int main()
{
    string s = "Добро пожаловать в с++: самый удобный язык  
для работы со строками!";
    string sub = "удобный";

    int found = RK_search(s, sub);
    string res = s.substr(found, sub.length());

    cout << "Слово \"" << res << "\" найдено на " << found  
<< " позиции." << endl;
}
```

```
[rubicus@rubicus output]$ ./"main2"  
Слово "удобный" найдено на 51 позиции.
```

Рисунок 3 - Результаты тестирования

Таблица 1 - Сравнительные результаты тестирования

Текст/Поиск	Удачный	Неудачный
Малый	36 нс.	33 нс.
Большой	33 нс.	35 нс.

## **5 ВЫВОД**

Были освоены приёмы реализации алгоритмов поиска образца в тексте на примере алгоритмов линейного поиска и алгоритма поиска Рабина-Карпа.



## **6 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ**

1. Структуры и алгоритмы обработки данных (часть 2): Лекционные материалы / Рысин М. Л. МИРЭА — Российский технологический университет, 2022/23. – 82 с.