



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»

РТУ МИРЭА

**ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №3**

Поиск образца в тесте
по дисциплине
«Структуры и алгоритмы обработки данных»

Выполнил студент группы ИКБО-01-21

Луковников Д.Р.

Принял преподаватель

Туманова М.Б.

Практическая

«__»_____2022 г.

работа выполнена

«Зачтено»

«__»_____2022 г.

Москва 2022

СОДЕРЖАНИЕ

СОДЕРЖАНИЕ	2
ЦЕЛЬ РАБОТЫ	3
ХОД РАБОТЫ.....	4
1.1 Задание 1.....	4
1.2 Задание 2.....	5
ВЫВОДЫ	7
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	8
ПРИЛОЖЕНИЯ.....	9

ЦЕЛЬ РАБОТЫ

Получить знания и навыки применения алгоритмов поиска в тексте подстроки (образца).

ХОД РАБОТЫ

1.1 Задание 1

Формулировка задачи: Дано предложение, слова в котором разделены пробелами и запятыми. Распечатать те слова, которые являются обращениями других слов в этом предложении.

Математическая модель решения: для решения задачи, поставим конкретные рамки того, что считать обращением, в рамках задачи обращение – слово, которое с двух сторон обособлено запятыми. С помощью шаблона найдём все такие слова в предложении

Код программы: для решения используем регулярные выражения, если находим вхождение, выводим его.

Листинг 1.1 – Код программы

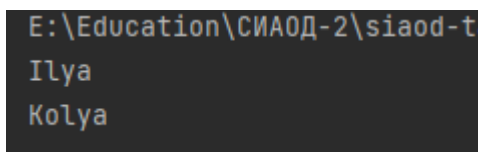
```
#include <iostream>
#include <string>
#include <regex>

using namespace std;

/**
 * This function is used to find the appeals in a string
 * @param str - the string to search
 * @return found words
 */
int main() {
    string sentences = "Would you go, Ilya, with a newspaper, Kolya, QQ.",
    tmp;
    const regex r(R"(, (\w+),)");
    smatch match;
    while (regex_search(sentences, match, r)) {
        // Trim commas from the string
        tmp = match.str(0).substr(2, -1);
        tmp.erase(tmp.size() - 1);
        cout << tmp << endl;

        // Suffix to find the rest of the string
        sentences = match.suffix().str();
    }
    return 0;
}
```

Тестирование: запустим программу с текущими данными, рисунок 1



```
E:\Education\СИАОД-2\siaod-t
Ilya
Kolya
```

Рисунок 1 – Тестирование программы

1.2 Задание 2

Формулировка задачи: Даны две строки *a* и *b*. Требуется найти максимальную длину префикса строки *a*, который входит как подстрока в строку *b*. При этом считать, что пустая строка является подстрокой любой строки. Реализация алгоритмом Кнута-Мориса-Пратта.

Математическая модель решения: данный алгоритм является один из самых эффективных для поиска подстроки в строке, его основная хитрость заключается в префикс функции, которая смотрит на начало и конец подстроки и смотрит сколько символов совпадает, в потом использует это для того, что бы при переборе строки возвращаться назад на минимальное количество символов.

Код программы: для начала напишем префикс функцию

Листинг 2.1 – Префикс функция

```
vector<int> prefixFunction(vector<char> str) {  
    /**  
     * Calculates the prefix function for a string  
     * @param str - string to calculate  
     * @return prefix function  
     */  
    int n = str.size();  
    vector<int> pi(n);  
    for (int i = 1; i < n; i++) {  
        int j = pi[i - 1];  
  
        // Пока не совпадет символ или не дойдем до начала строки  
        while (j > 0 && str[i] != str[j]) {  
            j = pi[j - 1];  
        }  
  
        // Если символы совпали, то увеличиваем значение префикса  
        if (str[i] == str[j]) {  
            j++;  
        }  
        pi[i] = j;  
    }  
    return pi;  
}
```

Далее этот префикс будем использовать для быстрого поиска подстроки в строке.

Листинг 2.2 – Основная функция программы

```
int main() {  
    string as, bs;  
    int maxLen = 0, index = 0;  
    // as = "aabaa";  
    // bs = "aaaaaabalaaa";  
    cin >> as >> bs;  
  
    // Переводим строки в векторы  
    vector<char> a = stringToVector(as), b = stringToVector(bs);
```

```

// Генерируем префикс-функцию для строки a
vector<int> pi = prefixFunction(a);
for (int i = 0; i < b.size(); i++) {

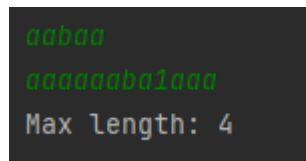
    while (index > 0 && b[i] != a[index]) {
        index = pi[index - 1];
    }

    // Если символы совпали, то увеличиваем значение префикса
    if (b[i] == a[index]) {
        index++;
        maxLen = max(maxLen, index);
    }

    // Если префикс равен длине строки, то выходим, строка входит целиком
    if (index == a.size()) {
        index = pi[index - 1];
    }
}
cout << "Max length: " << maxLen << endl;
return 0;
}

```

Тестирование: запустим программу для небольшой строки и подстроки, рисунок 2.



```

aaba
aaaaababaaa
Max length: 4

```

Рисунок 2 – Тестирование программы

ВЫВОДЫ

При выполнении работы были получены навыки реализации хэш таблиц, в частности хэш таблицы с открытой адресацией и квадратичным пробированием.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Страуструп Б. Программирование. Принципы и практика с использованием С++. 2-е изд., 2016.
2. Документация по языку С++ [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).

ПРИЛОЖЕНИЯ

Приложение А – Исходный код программы алгоритма Кнута-Мориса-Пратта.

Приложение А

Исходный код программы алгоритма Кнута-Мориса-Пратта.

Листинг 3.1 – main.cpp

```
#include <iostream>
#include <string>
#include <regex>

using namespace std;

/**
 * Даны две строки a и b. Требуется найти максимальную длину
 * префикса строки a, который входит как подстрока в строку b. При
 * этом считать, что пустая строка является подстрокой любой
 * строки. Реализация алгоритмом Кнута-Мориса-Пратта.
 * @return
 */

int max(int a, int b) {
    /**
     * Returns the maximum of two numbers
     * @param a - first number
     * @param b - second number
     * @return maximum of two numbers
     */
    return a > b ? a : b;
}

vector<char> stringToVector(const string& str) {
    /**
     * Converts a string to a vector
     * @param str - string to convert
     * @return vector of chars
     */
    vector<char> vec;
    for (char c: str) {
        vec.push_back(c);
    }
    return vec;
}

vector<int> prefixFunction(vector<char> str) {
    /**
     * Calculates the prefix function for a string
     * @param str - string to calculate
     * @return prefix function
     */
    int n = str.size();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i - 1];

        // Пока не совпадет символ или не дойдем до начала строки
        while (j > 0 && str[i] != str[j]) {
            j = pi[j - 1];
        }

        // Если символы совпали, то увеличиваем значение префикса
        if (str[i] == str[j]) {
            j++;
        }
    }
}
```

Продолжение Листинг 3.1

```
        pi[i] = j;
    }
    return pi;
}

void showPrefix(const vector<int> &pi) {
    /**
     * Shows the prefix function
     * @param pi - prefix function
     */
    for (int i: pi) {
        cout << i << " ";
    }
    cout << endl;
}

int main() {
    string as, bs;
    int maxLen = 0, index = 0;
    //   as = "aabaa";
    //   bs = "aaaaaabalaaa";
    cin >> as >> bs;

    // Переводим строки в векторы
    vector<char> a = stringToVector(as), b = stringToVector(bs);

    // Генерируем префикс-функцию для строки a
    vector<int> pi = prefixFunction(a);
    for (int i = 0; i < b.size(); i++) {

        while (index > 0 && b[i] != a[index]) {
            index = pi[index - 1];
        }

        // Если символы совпали, то увеличиваем значение префикса
        if (b[i] == a[index]) {
            index++;
            maxLen = max(maxLen, index);
        }

        // Если префикс равен длине строки, то выходим, строка входит целиком
        if (index == a.size()) {
            index = pi[index - 1];
        }
    }
    cout << "Max length: " << maxLen << endl;
    return 0;
}
```