

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №4
по дисциплине «Построение и анализ алгоритмов»
Тема: Алгоритм Кнута-Морриса-Пратта

Студентка гр. 7383

Ханова Ю.А.

Преподаватель

Жангиров Т. Р.

Санкт-Петербург

2019

Цель работы

Исследовать и реализовать задачу поиска подстроки в строке при помощи алгоритма Кнута-Морриса-Пратта(КМП).

Формулировка задачи:

1)Реализуйте алгоритм КМП и с его помощью для заданных шаблона Р ($|P| \leq 15000$) и текста Т ($|T| \leq 5000000$) найдите все вхождения Р в Т.

2) Заданы две строки А ($|A| \leq 5000000$) и В ($|B| \leq 5000000$). Определить, является ли А циклическим сдвигом В (это значит, что А и В имеют одинаковую длину и А состоит из суффикса В, склеенного с префиксом В). Например, defabc является циклическим сдвигом abcdef.

Реализация задачи

В данной работе была реализована функция `main()`, в которой производился ввод данных и вызывались функции алгоритма.

`void Pi(string P, vector<int>&pi);` - заполняет массив значениями префикс-функции (значение, равное максимальной длине совпадающих префикса и суффикса подстроки в образе).

`void KMP(string P, string T, vector<int> &pi, vector<int>&result);` - реализация алгоритма КМП строки посимвольно сравниваются, затем в случае несовпадения подстрока смещается на значение префикс-функции предыдущего элемента.

`void Shift(string A, string B, vector<int> &pi, int &index);` - проверка на циклическое смещение(используется алгоритм КМП).

Тестирование

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

Так же было проведено исследование алгоритма. В начале работы вычисляются значения префикс функций каждого символа подстроки. Пусть m – длина первой строки, n – длина основной строки, тогда сложность алгоритма по времени будет составлять $O(m+n)$, что значительно лучше чем алгоритм прямого поиска, чья сложность будет в таком случае $O(m*n)$.

Сложность по памяти составляет $O(m)$.

Выводы

В ходе выполнения лабораторной работы была решена задача поиска подстроки в строке на языке C++, и исследован алгоритм Кнута-Морриса-Пракса. Полученный алгоритм имеет линейную сложность.

Код программ представлен в Приложении А.

ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ

lr4_1.cpp

```
#include <iostream>
```

```
#include <string>
```

```
#include <vector>
```

```
using namespace std;
```

```
void Pi(string P, vector<int>&pi){
```

```
    pi[0]=0;
```

```
    int i = 1, j = 0;
```

```
    while(i<P.length()){
```

```
        if(P[i]==P[j]) {
```

```
            pi[i] = j+1;
```

```
            i++;
```

```
            j++;
```

```
        }
```

```
    else
```

```
        if(j==0){
```

```
            pi[i] = 0;
```

```
            i++;
```

```
        }
```

```
        else j = pi[j- 1];
```

```
    }
```

```
}
```

```
void KMP(string P, string T, vector<int> &pi, vector<int>&result){
```

```
    int k = 0, l = 0;
```

```
    while(k < T.length()){
```

```
        if(T[k] == P[l]){
```

```
            k++;
```

```
            l++;
```

```

        if(l == P.length())
            result.push_back(k - 1);
    } else if(l == 0){
        k++;
    } else l = pi[l - 1];
}
}

int main() {
    string P, T;
    getline(cin, P);
    getline(cin, T);
    vector<int> pi(P.size()), result;
    Pi(P, pi);
    KMP(P, T, pi, result);
    if(result.size() == 0) cout << -1 << endl;
    else {
        for(int i = 0; i < (result.size() - 1); i++){
            cout << result[i] << ",";
        }
        cout << result[result.size() - 1] << endl;
    }
    pi.clear();
    result.clear();
    P.clear();
    T.clear();
    // put your code here
    return 0;
}

```

lr4_2.cpp

```

#include <iostream>
#include <string>
#include <vector>

using namespace std;

void Pi(string P, vector<int>&pi){
    pi[0]=0;

```

```

    int i = 1, j = 0;
    while(i<P.length()){
        if(P[i]==P[j]) {
            pi[i] = j+1;
            i++;
            j++;
        }
        else
            if(j==0){
                pi[i] = 0;
                i++;
            }
            else j = pi[j- 1];
    }
}

void Shift(string A, string B, vector<int> &pi, int &index){
    int k = 0, l = 0;
    while(true){
        if(k == B.length())
            k = 0;
        if(B[k] == A[l]){
            k++, l++;
            if(l == A.length()){
                index = (l - k);
                break;
            }
        } else if(l == 0){
            k++;
            if(k == B.length())
                break;
        } else l = pi[l -1];
    }
}

int main() {
    string A, B;
    getline(cin, A);
    getline(cin, B);
    vector <int> pi(B.size());
    int index = -1;
    Pi(B, pi);
    if(A.length()==B.length()) Shift(A, B, pi, index);
    else index = -1;
    cout << index;
    pi.clear();
    A.clear();
    B.clear();
    // put your code here
    return 0;
}

```

ПРИЛОЖЕНИЕ Б.

ТЕСТОВЫЕ СЛУЧАИ

Результаты тестов представлены в табл. 1-2.

Таблица 1 – запуск lr4_1.cpp.

Входные данные	Выходные данные
msn sljmsnifls	3
ghds lskdamskaowgh	-1
ab sdabslnab	2,7

Таблица 2 – запуск lr4_2.cpp.

Входные данные	Выходные данные
abcdef efabcd	4
abcdefdsa efabcd	-1
slakma slkjmoa	-1