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

Студент гр. 9382 _____ Балаева М.О.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2021

Цель работы.

Разработать программу, реализующую поиск подстроки в строке с помощью алгоритма Кнута-Морриса-Пратта. Также разработать программу, определяющую, является ли первая строка циклическим сдвигом второй строки.

Задание 1

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

Вход:

Первая строка - P

Вторая строка - T

Выход:

индексы начал вхождений P в T, разделенных запятой, если P не входит в T, то вывести -1

Sample Input:

ab

abab

Sample Output:

0,2

Задание 2

Заданы две строки A ($|A| \le 5000000$) и B ($|B| \le 5000000$).

Определить, является ли A циклическим сдвигом B (это значит, что A и B имеют одинаковую длину и A состоит из суффикса B, склеенного с префиксом B). Например, defabc является циклическим сдвигом abcdef.

Вход:

Первая строка - A

Вторая строка - B

Выход:

Если A является циклическим сдвигом B, индекс начала строки B в A, иначе вывести −1. Если возможно несколько сдвигов вывести первый индекс.

Sample Input:

defabc

abcdef

Sample Output:

3

Описание алгоритма.

Изначально создается список для префикс-функций, длина которого равна длине строки, для которой будет считаться префикс-функция. Префикс-функция для рассматриваемого элемента строки — функция, показывающая максимальную длину совпадающих префикса и суффикса подстроки в строке, которая заканчивается рассматриваемым элементом строки, где префикс - подстрока, начинающиеся с начала строки, а суффикс – это подстрока, заканчивающиеся в конце строки, для алгоритма необходимо, чтобы длины префикса и суффикса совпадали. Для первого элемента префикс-функция будет равна 0.

Алгоритм для первого задания.

Считаем префикс-функцию для шаблона, который нужно найти в строке. Далее сравниваем каждый символ строки с каждым символом шаблона, в случае совпадения мы рассматриваем следующие символы. Затем проверяем совпадает ли текущий индекс элемента шаблона с длиной строки, если совпадает, значит, шаблон найден , в ответ будет записан индекс начала вхождения подстроки в строку, а дальше мы будем сравнивать символ подстроки под индексом префикс-функции предыдущего символа, так как

подстрока может сливаться с подстрокой при поиске в строке , если нет , то продолжается работа алгоритма.

Если текущий символ строки не совпадает с текущим символом шаблона, если символ является первым в подстроке, берем для сравнения следующий символ строки. В случае неравенства символов, если рассматриваемый символ не является первым в подстроке, следующим рассматриваемым символом в подстроке станет символ под индексом префикс-функции предыдущего символа, чтобы мы учли то, что в рассмотренных символах могла быть часть подстроки.

Если шаблона нет вообще в строке, будет возвращено -1.

Алгоритм для второго задания.

Изначально проверяются длины строк, если они отличны, следовательно первая строка не будет являться циклическим сдвигом второй.

Далее для выполнения алгоритма необходимо удвоить первую строку, т.к в таком случае первая строка будет содержать в себе вторую строку, если первая строка является циклическим сдвигом второй. Теперь задача свелась к заданию 1, поэтому дальнейший алгоритм совпадает.

Оценка сложности по памяти.

Для первого задания в худшем случае подстрока не будет содержаться в строке, тогда O(A + B), где A-длина первой строки, а B- длина второй строки . Для второго задания в худшем случае нам придется пройти по удвоенной строке, следовательно O(2*A+B), где A-длина первой строки, а B-длина второй строки.

Оценка сложности по времени.

Для первого задания значение префикс-функции вычисляется за O(N) сравнений, где N – длина подстроки, так как необходимо обойти всю строку, чтобы определить префикс-функцию. Поиск подстроки в строке будет O(A)-длина строки, т.к строка будет пройдена 1 раз. В таком случае O(N + A).

Для второго задания сложность по времени вычисляется также и совпадает со сложностью для первого задания и составляет O(N + A).

Тестирование для первого задания.

№	Входные данные	Выходные данные
1	ab abab	0,1
2	abcasda sadfasf	-1
3	aba abababbabab	0,2,7
4	qwe qwerty	0
5	abcasda sadfasf	-1
6	abbaabbab abbaabbaabbaabbababbaabbab	8,17

Тестирование для второго задания.

№	Входные данные	Выходные данные
1	oposoow ooposoow	-1
2	asd adff	-1
3	abbaabbab abbababba	4
4	aaoaa	2

	oaaaa	
5	asd adff	-1
6	defabc abcdef	3

Выводы.

В ходе работы был реализован алгоритм Кнута-Морриса-Пратта для поиска подстрок в строке, а также его модификация для проверки циклического сдвига двух строк

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: kmp.cpp

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

std::vector < size_t > prefix (std::string s)
{
   std :: cout << "Building a prefix function for a string" << s << "\n";
   std::vector < size_t > pr(s.size(), 0);
   for (size_t i = 1; i < s.size(); i++)
   {
      size_t j = pr[i - 1];
      while (j > 0 && s[i] != s[j]) j = pr[j - 1];

      std :: cout << "\n Compare" << j + 1 << "th character (" << s[j] << ") and " <<
i + 1 << "th character (" << s[i] << ") lines \n";
      if (s[i] == s[j]) j++;
      pr[i] = j;
      std :: cout << "Characters are equal, comparison continues one position
further \n";
   }
   std::cout<<"The prefix function is: ";
   for(int i = 0; i < pr.size(); i++){
      std::cout<<pr[i];
   }
   std::cout<<std::endl;
   return pr;
}

std::vector < size_t > kmp (std::string s, std::string t)
{

   std::vector < size_t > prefix_func = prefix(t);
   size_t j = 0;
   std::vector < size_t > find;
   std :: cout << "\n Searches for the pattern" << t << "in the string" << s << "\n";
   for (size_t i = 0; i < s.size(); i++)
```

```
    {
        while (j > 0 && t[j] != s[i]) j = prefix_func[j - 1];
        if (t[j] == s[i]){

            std :: cout << j << "th character of pattern (" << t[j] << ") and " << i << "th
character of string (" << s[i] << ") match \n ";
            j++;
        }
        if (j == t.size())
        {
            std::cout<<"Found pattern in string and added to answer"<<std::endl;
            find.push_back(i + 1 - t.size());
        }
    }
    return find;
}

int main()
{
    std::string s, t;
    std::cin >> t;
    std::cin >> s;
    std::vector < size_t > rez = kmp(s, t);
    if (rez.size() > 0)
    {
        for (size_t i = 0; i < rez.size() - 1; i++)
            std::cout << rez[i] << ",";
        std::cout << rez[rez.size() - 1];
    }
    else std::cout << -1;
}
```

Название файла: kmp2.cpp

```
#include <iostream>
#include <vector>
#include <string>
std::vector < size_t > prefix (std::string s)
{
   std :: cout << "Building a prefix function for a string " << s << "\n";
   std::vector < size_t > pr(s.size(), 0);
   for (size_t i = 1; i < s.size(); i++)
   {
      size_t j = pr[i - 1];
      while (j > 0 && s[i] != s[j]) j = pr[j - 1];
      std :: cout << "\n Compare " << j + 1 << "th character (" << s[j] << ") and "
<< i + 1 << "th character (" << s[i] << ") lines \n";
      if (s[i] == s[j]) j++;
      pr[i] = j;
      std :: cout << "Characters are equal, comparison continues one position
further \n";
   }
   std::cout<<"The prefix function is: ";
   for(int i = 0; i < pr.size(); i++){
      std::cout<<pr[i];
   }
   std::cout<<std::endl;
   return pr;
}
int Cycle (const std::string & s, const std::string & t)
{
   if (s.size() != t.size()){
      std::cout<<("The string "+s+" is not a circular shift of the string "+ t+  " ,
because the lengths are different\n");
      return -1;
   }
   std::cout<<("duplicate the line " + s);
   std::string new_s = s + s;
   std::cout<<("New string:" + new_s);
   std::vector < size_t > prefix_ = prefix(t);
   size_t j = 0;
   size_t find = -1;
   std :: cout << "\n Searches for the pattern " << t << " in the string " << s << "\n";
   for (size_t i = 0; i < new_s.size(); i++)
   {
```

```
        while (j > 0 && t[j] != new_s[i]) j = prefix_[j - 1];
        if (t[j] == new_s[i]) j++;
        std :: cout << j << "th character of pattern (" << t[j] << ") and " << i << "th
character of string (" << s[i] << ") match \n ";
        if (j == t.size())
        {
            std::cout<<"Found pattern in string and added to answer"<<std::endl;
            find = (i + 1 - t.length());
            break;
        }
    }
    return find;
}

int main()
{
    std::string s, t;
    std::cin >> t;
    std::cin >> s;
    std::cout << Cycle(t,s);
}
```