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: Кнут-Моррис-Пратт

Студентка гр. 9383 _____ Карпекина А.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2021

Цель работы.

Изучить и реализовать алгоритм Кнута-Морриса-Пратта для поиска всех подстрок по шаблону. Реализовать алгоритм проверки на циклический сдвиг.

Постановка задачи.

1 пункт. Реализуйте алгоритм КМП и с его помощью для заданных шаблона P ($|P|\leq 15000$) и текста T($|T|\leq 5000000$) 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

2 пункт. Заданы две строки A($|A|\leq 5000000$) и B($|B|\leq 5000000$). 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.

Описание алгоритма.

Математически определение префикс-функции можно записать следующим образом: $\pi[i] = max_{n=0\ldots i}\{n : s[0 \ldots n-1] = s[i-n+1 \ldots i]\}$, где s[0 … n-1] - данная строка.

Алгоритм Кнута-Морриса-Пратта, который находит позиции всех вхождений строки P в текст T, работает следующим образом.

Построим строку S=P#T, где # - любой символ, не входящий в алфавит P и T. Посчитаем на ней значение префикс-функции p. Если в какой-то позиции i выполняется условие p[i]=|P|, то в этой позиции начинается очередное вхождение образца в цепочку.

Алгоритм поиска циклического сдвига отличается только тем, что ведется в удвоенной первой строке, так как при сложении строк первая будет содержать в себе вторую строку, если она является циклическим сдвигом.

Алгоритм Кнута-Морриса-Пратта имеет сложность O(P+T) по времени и памяти.

Описание структур и функций.

- vector < int > CalculatingPrefixFunction (string Line) - формирует вектор значений префикс функции символов входной строки и возвращает его;
- void KnuthMorrisPratt (string FirstLine, string SecondLine, vector <int>&Result) - функция реализует алгоритм Кнута-Морриса-Пратта;
- int IsCyclicShift (string FirstLine, string SecondLine) - функция проверяет является ли строка FirstLine, циклическим сдвигом строки SecondLine;

Код разработанных программ см. в приложении А.

Тестирование.

Таблица 1 - результаты тестирования lb4_1.cpp

Тест	Входные данные	результат работы алгоритма
№1	ab abab	0,2
№2	nnfjfkc	-1

	kgykg	
№3	asasas asasas	0

Таблица 2 - результаты тестирования lb4_2.cpp

Тест	Входные данные	результат работы алгоритма
№1	qwerty tyqwer	4
№2	htj uky	-1
№3	qweryt tyqwer	-1

Вывод.

В ходе выполнения работы были изучены алгоритм Кнута - Морриса - Пратта для поиска всех подстрок и префикс функция. Также алгоритм Кнута - Морриса - Пратта был оптимизирован для решения задачи поиска циклического сдвига. Данные алгоритмы были реализованы на языке программирования C++.

ПРИЛОЖЕНИЕ А
Исходный код программы

Название файла: lb4_1.cpp

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

using namespace std;



vector < int > CalculatingPrefixFunction (string Line)
{
  int LineLength = Line.length ();
  vector < int >prefixes (LineLength);
  prefixes[0] = 0;
  for (int i = 1; i < LineLength; i++)
    {
      int ActualLineLength = prefixes[i - 1];
      while (ActualLineLength > 0
             && (Line[ActualLineLength] != Line[i]))
        ActualLineLength = prefixes[ActualLineLength - 1];
      if (Line[ActualLineLength] == Line[i])
        ActualLineLength++;
      prefixes[i] = ActualLineLength;
    }
  return prefixes;
}

void KnuthMorrisPratt (string FirstLine, string SecondLine, vector < int >&Result)
{
```

```
    vector<int> p = CalculatingPrefixFunction(FirstLine + "#");
    int FirstLineStep = 0;
    for(int SecondLineStep = 0; SecondLineStep < SecondLine.size();
++SecondLineStep)
    {
        while(FirstLineStep > 0 && FirstLine[FirstLineStep] !=
SecondLine[SecondLineStep])
            FirstLineStep = p[FirstLineStep-1];
        if(FirstLine[FirstLineStep] == SecondLine[SecondLineStep])
            FirstLineStep++;
        if(FirstLineStep == FirstLine.size())
            Result.push_back(SecondLineStep - FirstLine.size() + 1);
    }

}

int main ()
{
  vector < int >Result;
  string FirstLine, SecondLine;
  cin >> FirstLine;
  cin >> SecondLine;
  KnuthMorrisPratt (FirstLine, SecondLine, Result);
  if (!Result.size ())
    cout << -1;
  else
    {
      string separator;
      for (auto entry : Result)
```

```
    {
      cout << separator << entry;
      separator = ",";
    }
   }
  return 0;
}
```

Название файла: lb4_2.cpp

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

using namespace std;



vector < int > CalculatingPrefixFunction (string Line)
{
  int LineLength = Line.length ();
  vector < int >prefixes (LineLength);
  prefixes[0] = 0;
  for (int i = 1; i < LineLength; i++)
    {
      int ActualLineLength = prefixes[i - 1];
      while (ActualLineLength > 0
             && (Line[ActualLineLength] != Line[i]))
        ActualLineLength = prefixes[ActualLineLength - 1];
      if (Line[ActualLineLength] == Line[i])
        ActualLineLength++;
      prefixes[i] = ActualLineLength;
```

```
    }
  return prefixes;
}

void IsCyclicShift (string FirstLine, string SecondLine, vector < int >&Result)
{

    FirstLine = FirstLine + FirstLine;
    vector<int> Prefixes = CalculatingPrefixFunction(SecondLine + "#");
    int FirstLineStep = 0;
    for(int SecondLineStep = 0; SecondLineStep < FirstLine.size();
++SecondLineStep)
    {
        while(FirstLineStep > 0 && SecondLine[FirstLineStep] !=
FirstLine[SecondLineStep])
            FirstLineStep = Prefixes[FirstLineStep-1];
        if(SecondLine[FirstLineStep] == FirstLine[SecondLineStep])
            FirstLineStep++;
        if(FirstLineStep == SecondLine.size())
        {
            Result.push_back(SecondLineStep - SecondLine.size() + 1);
            if(true) break;
        }
    }
}

int main ()
{
  vector < int >Result;
```

```
  string FirstLine, SecondLine;
  cin >> FirstLine;
  cin >> SecondLine;
  if(SecondLine.size() == FirstLine.size()) IsCyclicShift (FirstLine, SecondLine,
Result);
  if (!Result.size ())
    cout << -1;
  else
    {
      string separator;
      for (auto entry : Result)
      {
        cout << separator << entry;
        separator = ",";
      }
    }
  return 0;
}
```