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

ОТЧЕТ

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

Студент гр. 0382 _____ Корсунов А.А.

Преподаватель _____ Шевская Н.В.

Санкт-Петербург
2022

Цель работы.

Применить на практике знания о построение алгоритма Крута-Морриса-Пратта. Реализовать алгоритм Крута-Морриса-Пратта для поиска всех подстрок по заданному шаблону. Реализовать алгоритм проверки, является ли одна строка циклическим сдвигом другой.

Задание.

1) Реализуйте алгоритм КМП и с его помощью для заданных шаблона P (|P|≤15000) и текста T (|T|≤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) Заданы две строки A (|A|≤5000000) и B (|B|≤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 вляется циклическим сдвигом B, индекс начала строки B в A, иначе вывести −1. Если возможно несколько сдвигов вывести первый индекс.

Ход работы.

1. Был произведен анализ задания.

2. Был реализован алгоритм Кнута-Морриса-Пратта:

1) Алгоритм для первого задания:

Данные, поддающиеся на вход заносятся в две структуры данных типа string, которые передаются в функцию kmp, которая сперва формируется строку, конкатенируя первую и вторую строку (между ними также ставится символ «$»). После чего создает вектор, в который записывает значение функции prefix_function. Функция prefix_function получается на вход одну строку и возвращает значение префикс-функции — массив, элементами которого являются максимальные префикс-суффиксы для строки 0 до k-1, где k изменяется от 0 до <длина первой строки>+<длина второй строки>. Работает эта функция следующим образом: создается вектор типа <int> размером, равным размеру переданной строки и заполняется нулями. Этот вектор по итогу будет равен префикс-функции. Первый элемент префикс функции всегда равен нулю, из этого соображения заводится цикл for от 0 до размера переданной строки. В этом цикле объявляется переменная temp, которая будет хранить в себе максимальную длину префикс-суффикса для строки от 0 до k-1 на предыдущем шаге каждой итерации, причем, если следующий элемент после префикс-суффикса не равен «новому» элементу (элементу, который добавляется на каждой итерации в КМП), то находится префикс-суффикс от префикса-суффикса (идет «откат» назад с помощью ранее заведенного вектора (т. к. фактически элементы этого вектора отражают длину максимального префикс-суффикса для подстроки от 1 до i-1 (из переданной строки, где i – шаг алгоритма)). В начале в эту переменную кладется максимальный префикс-суффикс предыдущей строки, если он не равен нулю и элемент, который стоит после последнего элемента максимального префикс-суффикса предыдущей строки не равен новому элементу, то происходит откат, о котором писалось

выше. Откат происходит до тех пор, пока новый элемент не станет равен элементу после последнего элемента текущего префикс-суффикса или пока не станет равен нулю. В конце как происходит сравнение этого элемента с новым элементом, и в случае, если они равны, temp увеличивается на единицу, отражая тем самым, что для текущей подстроки максимальный префикс-суффикс увеличивается на единицу от temp. Temp записывается в i-ую ячейку вектора. На выходе будет префикс-функция. После чего, в kmp находятся индексы вхождений первого элемента строки 1 в строку 2 и записываются в вектор типа <int> (идет прохождение по строке P+$+T и ищутся в ней элементы, равные длине P (P – первая строка, T – вторая строка));

2) Алгоритм для второго задания:

Алгоритм в точности повторяет первый за исключением следующих изменений:

а) В начале строки сравниваются по длине, если они не равны, выводится -1;

б) В функцию передается не строка P+«$»+T, а строка T+«$»+P+P (в задании P = A, T = B) – прогнав КМП для такой строки можно будет найти циклический сдвиг P в T (если он есть);

в) На выходе у kmp не вектора, а одно число — позиция первого элемента T в P

3) Структура данных, которая представляет строки — string, структура данных, которая представляет префикс-функцию — ветор типа <int>;

4) Сложность алгоритмов по времени — O(|P|+|T|), т. к. нужно построить префикс-функцию и найти в ней потом элементы равные длине искомой строки

Сложность алгоритма по памяти — O(|P|+|T|), т. к. нужно хранить обе строки;

5) Функции и структуры данных:

*vector<int> prefix_function(const std::string& str) – основная часть алгоритма КМП, принимает строку, возвращается префикс-функцию;

*vector<int> kmp(std::string& str_P, std::string& str_T) – часть алгоритма КМП, которая формируется строку для prefix_function и возвращает массив вхождений (для первого задания) и первое вхождение (для второго задания);

6) Тестирование (1 программа):

а) Входные данные:

ab

abab

Выходные данные:

0, 2

б) Входные данные:

ab

abbaabbab

Выходные данные:

0, 4, 7

в) Входные данные:

efef

efefeftef

Выходные данные:

0, 2

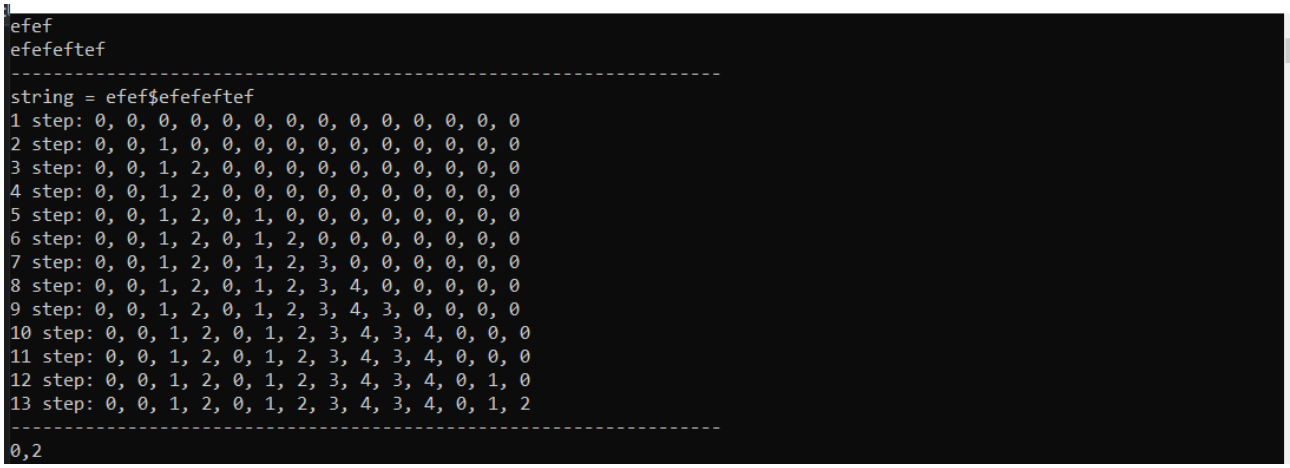

Рисунок 1 - Пример работы первой программы

7) Тестирование (2 программа):

а) Входные данные:

defabc

abcdef

Выходные данные:

3

б) Входные данные:

abac

acab

Выходные данные:

2

в) Входные данные:

yqwert

qwerty

Выходные данные:

1

```
defabc
abcdef
------------------------------------------------------------------
string = abcdef$defabcdefabc
1 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
2 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
3 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
4 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
5 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
6 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
7 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
8 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
9 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
10 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0
11 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0
12 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 0, 0, 0, 0, 0
13 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 0, 0, 0, 0
14 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 0, 0, 0
15 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 0, 0
16 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 1, 0
17 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 1, 2
18 step: 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 1, 2, 3
------------------------------------------------------------------
3
```

Рисунок 2 - Пример работы второй программы

8) Были разработаны Юнит-тесты для обеих программ:

Для Юнит-тестов использовался встроенный модуль Microsoft CppUnitTest

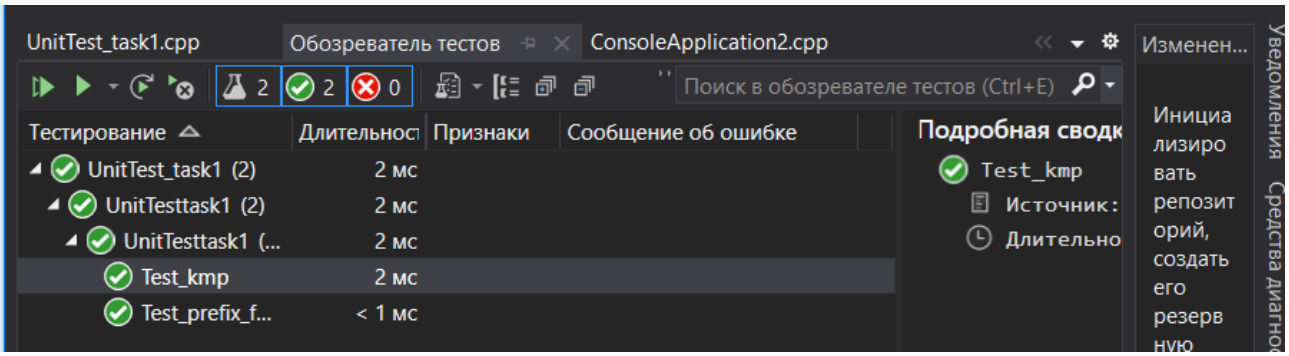

Рисунок 3 — Результат тестирования для первой программы

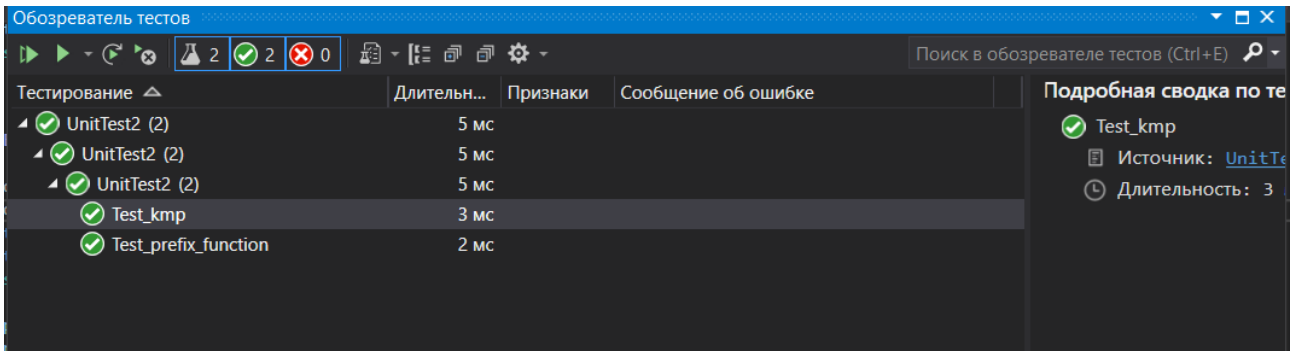

Рисунок 4 — Результат тестирования для второй программы

Ответы к тестам программы 1:

Таблица 1 (Test_prefix_function)

№ теста	Данные на вход (строка)	Ожидаемые данные на выход	Результат теста
1	“ab$abab”	{0, 0, 0, 1, 2, 1, 2}	Пройден успешно
2	“”	{}	Пройден успешно
3	“ab$abbaabbab”	{0, 0, 0, 1, 2, 0, 1, 1, 2, 0, 1, 2}	Пройден успешно
4	“efef$efefeftef”	{0, 0, 1, 2, 0, 1, 2, 3, 4, 3, 4, 0, 1, 2}	Пройден успешно

Таблица 2 (Test_kmp)

№ теста	Данные на вход (2 строки)	Ожидаемые данные на выход	Результат теста
1	“ab” и “abab”	{0, 2}	Пройден успешно
2	“” и “”	{}	Пройден успешно
3	“ab” и “abbaabbab”	{0, 4, 7}	Пройден успешно
4	“efef” и “efefeftef”	{0, 2}	Пройден успешно
5	“aaaaaaaaaaaa” и “efefeftef”	{}	Пройден успешно

Ответы к тестам программы 2:

Таблица 3 (Test_prefix_function)

№ теста	Данные на вход (строка)	Ожидаемые данные на выход	Результат теста
1	“abcdef$defabcdefabc”	{0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 1, 2, 3}	Пройден успешно
2	“acab$abacabac”	{0, 0, 1, 0, 0, 1, 0, 1, 2, 3, 4, 1, 2}	Пройден успешно
3	“”	{}	Пройден успешно
4	“qwerty$yqwertyqwert”	{0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5}	Пройден успешно

Таблица 4 (Test_kmp)

№ теста	Данные на вход (2 строки)	Ожидаемые данные на выход	Результат теста
1	“defabc” и “abcdef”	3	Пройден успешно
2	“abac” и “acab”	2	Пройден успешно
3	“” и “”	-13	Пройден успешно
4	“yqwert” и “qwerty”	1	Пройден успешно
5	“qwe” и “w”	-13	Пройден успешно

Комментарий к тесту: «-13» в ожидаемом значении - флаг, который возвращается функцией в main, main в таком случае выводит «-1».

Выводы.

Были применены на практике знания о построение алгоритма Крута-Морриса-Пратта. Реализован алгоритм Крута-Морриса-Пратта для поиска всех подстрок по заданному шаблону. Реализован алгоритм проверки, является ли одна строка циклическим сдвигом другой.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл kmp_1.cpp:

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

std::vector<int> prefix_function(const std::string& str)
{
   std::vector<int> pi(str.size()); //массив префикс-функции (изначально заполняется нулями)
   std::cout << "----------------------------------------------------------------\n";
   std::cout << "string = " << str << "\n";

   for (int i = 1; i < str.size(); i++) //первый элемент всегда будет равен 0 в pi
   {
      int temp = pi[i - 1]; //кладется предыдущее значение массива pi, которое равно максимально длины префикс-суффикса для i - 1 строки

      while ((temp > 0) && (str[i] != str[temp])) //"откат до последнего элемента текущего префикс-суффикса, причем, элемент, который стоит после последнего элемента префикса должен быть равен
      {
         temp = pi[temp - 1]; //новому элементу, который добавляется на i-ом (в СИ нумерация с нулевого элемента, поэтому temp уже будет указывать на элемент после последнего элемента префикса
      }

      if (str[i] == str[temp]) //temp-1 - индекс последнего элемента текущего префикс-суффикса, temp - следующего (т.к. в СИ нумерация с нулевого элемента)
      {
         temp++; //если этот предыдущий элемент равен новому элементу (который добавляется на i-ом шаге), то префикс функция для i-ой подстроки равна temp+1
      }
      pi[i] = temp;
      std::cout << i << " step: " << pi[0];
      for (int i = 1; i < pi.size(); i++)
      {
         std::cout << ", " << pi[i];
      }
```

```
        std::cout << "\n";
    }
    std::cout << "-----------------------------------------------------------------\n";
    return pi;
}

std::vector<int> kmp(std::string& str_P, std::string& str_T)
{
    std::vector<int> temp = prefix_function(str_P + '$' + str_T);
    std::vector<int> result;

    for (int i = 0; i < temp.size(); i++)
    {
        if (temp[i] == str_P.size())
        {
            if (str_P.size() == 0)
            {
                break;
            }
            result.push_back(i - 2 * str_P.size()); // i(индекс последнего элемента P в P+'$'+T) - str_P.size()-1(P+'$') - str_P.size()+1(индекс первого элемента P в P+'$'+T)
        }
    }

    return result;
}

int main()
{
    std::string P, T;
    std::cin >> P >> T;

    std::vector<int> out = kmp(P, T);

    if (out.empty())
    {
        std::cout << -1;
        return 0;
    }

    std::cout << out[0];

    if (out.size() > 1)
```

```
    {
        for (int i = 1; i < out.size(); i++)
        {
            std::cout << ',' << out[i];
        }
    }

    return 0;
}
```

файл UnitTest_task1.cpp:

```
#include "pch.h"
#include "CppUnitTest.h"
#include <vector>
#include "../ConsoleApplication2/ConsoleApplication2.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTesttask1
{
        TEST_CLASS(UnitTesttask1)
        {
        public:

                TEST_METHOD(Test_prefix_function)
                {
                        std::string input = "ab$abab";
                        std::vector<int> expected = { 0, 0, 0, 1, 2, 1, 2 };
                        std::vector<int> actual = prefix_function(input);
                        Assert::IsTrue(expected ==  actual);

                        input = "";
                        expected = {};
                        actual = prefix_function(input);
                        Assert::IsTrue(expected == actual);

                        input = "ab$abbaabbab";
                        expected = { 0, 0, 0, 1, 2, 0, 1, 1, 2, 0, 1, 2 };
                        actual = prefix_function(input);
                        Assert::IsTrue(expected == actual);

                        input = "efef$efefeftef";
                        expected = { 0, 0, 1, 2, 0, 1, 2, 3, 4, 3, 4, 0, 1, 2 };
                        actual = prefix_function(input);
```

```
            Assert::IsTrue(expected == actual);
        }

        TEST_METHOD(Test_kmp)
        {
            std::string input_P = "ab";
            std::string input_T = "abab";
            std::vector<int> expected = { 0, 2 };
            std::vector<int> actual = kmp(input_P, input_T);
            Assert::IsTrue(expected == actual);

            input_P = "";
            input_T = "";
            expected = {};
            actual = kmp(input_P, input_T);
            Assert::IsTrue(expected == actual);

            input_P = "ab";
            input_T = "abbaabbab";
            expected = { 0,4,7 };
            actual = kmp(input_P, input_T);
            Assert::IsTrue(expected == actual);

            input_P = "efef";
            input_T = "efefeftef";
            expected = { 0,2 };
            actual = kmp(input_P, input_T);
            Assert::IsTrue(expected == actual);
        }
    };
}
```

файл kmp_2.cpp:

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

std::vector<int> prefix_function(const std::string& str)
{
   std::vector<int> pi(str.size()); //массив префикс-функции (изначально
заполняется нулями)
   std::cout << "-----------------------------------------------------------\n";
   std::cout << "string = " << str << "\n";

   for (int i = 1; i < str.size(); i++) //первый элемент всегда будет равен 0 в pi
```

```
    {
        int temp = pi[i - 1]; //кладется предыдущее значение массива pi, которое равно максимально длины префикс-суффикса для i - 1 строки

        while ((temp > 0) && (str[i] != str[temp])) //"откат до последнего элемента текущего префикс-суффикса, причем, элемент, который стоит после последнего элемента префикса должен быть равен
        {
            temp = pi[temp - 1]; //новому элементу, который добавляется на i-ом (в СИ нумерация с нулевого элемента, поэтому temp уже будет указывать на элемент после последнего элемента префикса
        }

        if (str[i] == str[temp]) //temp-1 - индекс последнего элемента текущего префикс-суффикса, temp - следующего (т.к. в СИ нумерация с нулевого элемента)
        {
            temp++; //если этот предыдущий элемент равен новому элементу (который добавляется на i-ом шаге), то префикс функция для i-ой подстроки равна temp+1
        }
        pi[i] = temp;
        std::cout << i << " step: " << pi[0];
        for (int i = 1; i < pi.size(); i++)
        {
            std::cout << ", " << pi[i];
        }
        std::cout << "\n";
    }
    std::cout << "----------------------------------------------------------------\n";
    return pi;
}

int kmp(std::string& str_P, std::string& str_T)
{

    std::vector<int> temp = prefix_function(str_T + '$' + str_P + str_P);
    int result = -13;

    for (int i = 0; i < temp.size(); i++)
    {
        if (temp[i] == str_P.size())
        {
            if (str_P.size() == 0)
```

```
            {
                break;
            }
            result = i - 2 * str_P.size(); // i(индекс последнего элемента P в P+'$'+T) - str_P.size()-1(P+'$') - str_P.size()+1(индекс первого элемента P в P+'$'+T)
            break;
        }
    }

    return result;
}

int main()
{
    std::string P, T;
    std::cin >> P >> T;

    if (P.size() != T.size())
    {
        std::cout << -1;
        return 0;
    }

    int out = kmp(P, T);

    if (out == -13)
    {
        std::cout << -1;
        return 0;
    }

    std::cout << out;

    return 0;
}
```

файл UnitTest2_task2.cpp:

```
#include "pch.h"
#include "CppUnitTest.h"
#include <vector>
#include "../kmp2/kmp2.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;
```

```
namespace UnitTest2
{
	TEST_CLASS(UnitTest2)
	{
	public:

		TEST_METHOD(Test_prefix_function)
		{
			std::string input = "abcdef$defabcdefabc";
			std::vector<int> expected = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4,
5, 6, 1, 2, 3 };
			std::vector<int> actual = prefix_function(input);
			Assert::IsTrue(expected == actual);

			input = "acab$abacabac";
			expected = { 0, 0, 1, 0, 0, 1, 0, 1, 2, 3, 4, 1, 2 };
			actual = prefix_function(input);
			Assert::IsTrue(expected == actual);

			input = "";
			expected = {};
			actual = prefix_function(input);
			Assert::IsTrue(expected == actual);

			input = "qwerty$yqwertyqwert";
			expected = { 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5 };
			actual = prefix_function(input);
			Assert::IsTrue(expected == actual);
		}

		TEST_METHOD(Test_kmp)
		{
			std::string input_A = "defabc";
			std::string input_B = "abcdef";
			int expected = 3;
			int actual = kmp(input_A, input_B);
			Assert::IsTrue(expected == actual);

			input_A = "abac";
			input_B = "acab";
			expected = 2;
			actual = kmp(input_A, input_B);
			Assert::IsTrue(expected == actual);
```

```
            input_A = "";
            input_B = "";
            expected = -13;
            actual = kmp(input_A, input_B);
            Assert::IsTrue(expected == actual);

            input_A = "yqwert";
            input_B = "qwerty";
            expected = 1;
            actual = kmp(input_A, input_B);
            Assert::IsTrue(expected == actual);
        }
    };
}
```