

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. 8382

Щемель Д.А.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Задание

Вхождение образца в строку

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

Sample Input:

ab

abab

Sample Output:

0,2

Нахождение циклического сдвига

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.

Sample Input:

defabc

abcdef

Sample Output:

3

Номер варианта: 1. Подготовка к распараллеливанию: работа по поиску разделяется на k равных частей, пригодных для обработки k потоками (при

этом длина образца гораздо меньше длины строки поиска).

Описание алгоритма

Вхождение образца в строку

Считываются две строки: образец и строка для поиска. После эти строки склеиваются между собой через уникальный символ, отсутствующий в обеих строках. Для получившейся строки считается префикс-функция. Это функция, которая для каждого символа строки сохраняет максимальную длину собственного суффикса, равного префиксу этой же строки. Поэтому и используется уникальный символ, чтобы префиксом считался только образец. По тем значениям префикс функции, которые совпадают с длиной образца можно найти индексы вхождения образца в строке. (Если вычесть длину образца + 1 на уникальный символ). Данные значения ищутся в несколько потоков, на каждый поток выделяется свой диапазон значений префикс-функции.

Нахождение циклического сдвига

Считываются две строки. После чего высчитывается префикс-функция для условной “склейки” второй строки, уникального символа и двух идущих подряд первых строк. (В реализации вместо создания новой строки используется функция-обёртка, позволяющая по индексу получить символ, который в этой склейке бы был). В результате в префикс-функции последнее значение, равное длине первой строки и будет требуемым результатом (индекс начала второй строки в первой). Это значение так же ищется в несколько потоков.

Сложности алгоритма:

Вхождение образца в строку

Сложность по количеству операций: $O(p + s)$, где p - длина строки образца, а s - длина строки для поиска. Префикс-функция вычисляется за линейное время, несмотря на наличие двух вложенных циклов в реализации. Это происходит потому, что итоговое количество определяется количеством итераций внутреннего цикла. Которе, в свою очередь определяется максимально возможным уменьшением счётчика, которое не превосходит $O(p + s)$.

Сложность по количеству памяти: $O(p + s)$, где p - длина строки образца, а s - длина строки для поиска. Нам необходимо хранить сами строки, их “склейку”

и значение префикс-функцию для неё.

Нахождение циклического сдвига

Сложность по количеству операций: $O(p + s)$, где p - длина строки образца, а s - длина строки для поиска. Аналогично предыдущей задаче. Сложность по количеству памяти: $O(p + s)$, где p - длина строки образца, а s - длина строки для поиска. Аналогично предыдущей задаче.

Описание функций и структур данных

Вхождение образца в строку

Структуры данных Пользовательские классы, использующиеся в реализации алгоритма не использовались.

Используемые функции (методы)

- *static int[] PrefixFunction(string str)* - используется для вычитывания префикс-функции для строки
- *static List FindPatternsOccurrences(string str, string pattern)* - используется для как точка входа для алгоритма и возвращает список индексов начала образца *pattern* в строке *str*
- *static List FindPatternsOccurrencesInPrefix(string str, int patternLength, IReadOnlyList prefix, int start, int end)* - используется для параллельного нахождения необходимых значений(индексов вхождений) в префикс-функции на заданном промежутке

Нахождение циклического сдвига

Структуры данных Пользовательские классы, использующиеся в реализации алгоритма не использовались.

Используемые функции (методы)

- *char charFromPatternOrStr(int i, const std::string& pattern, const std::string& str)* - принимает индекс и по этому индексу возвращает символ, который стоял бы на месте “склейки” двух строк
- *std::vector prefixFunction(const std::string& pattern, const std::string& str)* - используется для вычисления префикс-функции для двух строк
- *int findPatternsOccurenciesInPrifix(int patternLength, const std::vector& prefix, int start, int end)* - используется для параллельного нахождения необходимого значения(индекса начала вхождения одной строки в другую)
- *int findIndexOfCyclicOffset(const std::string& str, const std::string& pattern)* - точка входа для алгоритма, которая принимает две строки и ищет начало вхождения одной строки в другую

Тестирование

Вхождение образца в строку

Тест 1:

Input:

ab

abab

Output : K value => 0 K value => 0 K value => 0 K index was increased to => 1 K value
=> 1 K index was increased to => 2 K value => 2 K index was decreased to => 0 K index
was increased to => 1 K value => 1 K index was increased to => 2 Threads count => 2
Count of indexes per thread => 2 0,2

Тест 2: Input:

a

aaaa

Output: 0,1,2,3

Тест 3: Input:

ab

ba

Output: -1

Нахождение циклического сдвига

Тест 1:

Input:

1234

2341

Output : K value => 0 K value => 0 K value => 0 K value => 0 K value => 0 K value =>
0 K value => 0 K value => 0 K index was increased to => 1 K value => 1 K index was
increased to => 2 K value => 2 K index was increased to => 3 K value => 3 K index was
increased to => 4 K value => 4 K index was decreased to => 0 K index was increased to
=> 1 Threads count => 2 Count of indexes per thread => 4 1

Тест 2: Input:

123

323

Output: -1

Вывод

В ходе выполнения лабораторной работы были получены навыки применения алгоритма Кнута-Морриса-Пратта на примере создания программ для решения следующих задач: нахождение индексов вхождения образца в строке и индекс циклического смещения одной строки в другой.

Приложение А. Исходный код

Вхождение образца в строку

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;

namespace lab4
{
    /// <summary>
    /// Class wrapper for printing
    /// </summary>
    public static class Logger
    {
        public enum LogLevel
        {
            Debug,
            Info
        }

        /// <summary>
        /// Log message with specify log level
        /// </summary>
        /// <param name="text"></param>
        /// <param name="level"></param>
        public static void Log(object text, LogLevel level = LogLevel.Info)
        {
            int logLevel;
            var hasVariable = int.TryParse(Environment.GetEnvironmentVariable("LAB4
            logLevel = hasVariable ? logLevel : (int)LogLevel.Info;

            if ((int)level >= logLevel)
            {
                Console.WriteLine(text.ToString());
            }
        }
    }
}
```

```

    }
}

}

class Program
{
    /// <summary>
    /// Calculate prefix-function for string
    /// </summary>
    /// <param name="str">String</param>
    /// <returns>Value of prefix-function as array</returns>
    static int[] PrefixFunction(string str)
    {
        var retArray = new int[str.Length];
        retArray[0] = 0;

        for (var i = 1; i < str.Length; ++i)
        {
            var k = retArray[i - 1];
            Logger.Log($"K value => {k}", Logger.LogLevel.Debug);
            while (k > 0 && str[i] != str[k])
            {
                k = retArray[k - 1];
                Logger.Log($"K index was decreased to => {k}", Logger.LogLevel.
            }

            if (str[i] == str[k])
            {
                ++k;
                Logger.Log($"K index was increased to => {k}", Logger.LogLevel.
            }

            retArray[i] = k;
        }
    }
}

```

```

        return retArray;
    }

    /// <summary>
    /// Find indexes of pattern occurrences in string
    /// </summary>
    /// <param name="str">String</param>
    /// <param name="pattern">Pattern for search</param>
    /// <returns>List of indexes</returns>
    static List<int> FindPatternsOccurrences(string str, string pattern)
    {
        var retArray = new List<int>();
        var prefix = PrefixFunction($"{pattern}#{str}");

        int maxThreadsCount;
        int maxAsyncIOThreadsCount;
        var threadsCount = (int)(Math.Floor((double)str.Length / pattern.Length));
        Logger.Log($"Threads count => {threadsCount}", Logger.LogLevel.Debug);
        ThreadPool.GetMaxThreads(out maxThreadsCount, out maxAsyncIOThreadsCount);
        threadsCount = threadsCount < maxThreadsCount ? threadsCount : maxThreadsCount;
        var countsPerThread = (int)(Math.Floor((double)str.Length / threadsCount));
        Logger.Log($"Count of indexes per thread => {countsPerThread}", Logger.LogLevel.Debug);

        for (var i = 0; i < threadsCount; ++i)
        {
            retArray.AddRange(FindPatternsOccurrencesInPrefix(str, pattern.Length, i * countsPerThread,
                str.Length));
        }

        var upperBound = countsPerThread * threadsCount;
        retArray.AddRange(FindPatternsOccurrencesInPrefix(str, pattern.Length, upperBound,
            str.Length));

        return retArray;
    }

```

```

    /// <summary>
    /// Find indexes of patterns occurrences in prefix-function
    /// On specified range
    /// </summary>
    /// <param name="str">String</param>
    /// <param name="patternLength">Length of pattern string</param>
    /// <param name="prefix">Prefix-function</param>
    /// <param name="start">Start of range for search</param>
    /// <param name="end">End of range for search</param>
    /// <returns></returns>
    static List<int> FindPatternsOccurrencesInPrefix(string str, int patternLength, int start, int end)
    {
        var retArray = new List<int>();
        for (var i = start; i < end; ++i)
        {
            if (prefix[patternLength + i + 1] == patternLength)
            {
                retArray.Add(i - patternLength + 1);
            }
        }

        return retArray;
    }

    static void Main(string[] args)
    {
        var pattern = Console.ReadLine();
        var str = Console.ReadLine();
        var result = FindPatternsOccurrences(str, pattern);
        Logger.Log(result.Any() ? string.Join(",", result) : "-1");
    }
}

```

Нахождение циклического сдвига

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

/// Return char that appearance at {pattern#strstr}
/// \param i index
/// \param pattern first string
/// \param str second str
/// \return char
char charFromPatternOrStr(int i, const std::string& pattern, const std::string& str)
{
    if (i < pattern.length())
    {
        return pattern[i];
    }

    if (i == pattern.length())
    {
        return '#';
    }

    if (i < pattern.length() + str.length() + 1)
    {
        return str[i - pattern.length() - 1];
    }

    return str[i - pattern.length() - str.length() - 1];
}

/// Calculate prefix-function for {pattern#strstr}
/// \param pattern first string
```

```

/// \param str second str
/// \return prefix-function as vector of int
std::vector<int> prefixFunction(const std::string& pattern, const std::string& str)
{
    int arraySize = pattern.length() + str.length() + str.length() + 1;
    std::vector<int> retVec(arraySize);
    retVec[0] = 0;

    for (int i = 1; i < arraySize; ++i)
    {
        int k = retVec[i - 1];
        std::cout << "K value => " << k << std::endl;
        char iChar = charFromPatternOrStr(i, pattern, str);
        char kChar = charFromPatternOrStr(k, pattern, str);
        while (k > 0 && iChar != kChar)
        {
            k = retVec[k - 1];
            kChar = charFromPatternOrStr(k, pattern, str);
#ifdef NDEBUG
            std::cout << "K index was decreased to => " << k << std::endl;
#endif
        }

        if (iChar == kChar)
        {
            ++k;
#ifdef NDEBUG
            std::cout << "K index was increased to => " << k << std::endl;
#endif
        }

        retVec[i] = k;
    }

    return retVec;
}

```

```
}
```

```
/// Find first pattern length occurrence in specified range in prefix-function  
/// \param patternLength length of pattern  
/// \param prefix prefix-function  
/// \param start start of range  
/// \param end end of range  
/// \return first pattern occurrence
```

```
int findPatternsOccurenciesInPrifix(int patternLength, const std::vector<int>& pref  
{  
    for (int i = end - 1; i >= start; --i)  
    {  
        if (prefix[patternLength + i + 1] == patternLength)  
        {  
            return i - patternLength + 1;  
        }  
    }  
  
    return -1;  
}
```

```
/// Find index of cyclic offset second string in first one  
/// \param str first string  
/// \param pattern second string  
/// \return index of offset
```

```
int findIndexOfCyclicOffset(const std::string& str, const std::string& pattern)  
{  
    int result = -1;  
    std::vector<int> prefix = prefixFunction(pattern, str);  
  
    int maxThreadsCount = std::thread::hardware_concurrency();  
    int threadsCount = (int)(floor((double)2 * str.length() / pattern.length()));  
#ifndef NDEBUG  
    std::cout << "Threads count => " << threadsCount << std::endl;  
#endif
```



```

    int countsPerThread = (int)(floor((double)2 * str.length() / threadsCount));
    threadsCount = threadsCount < maxThreadsCount ? threadsCount : maxThreadsCount;
#ifdef NDEBUG
    std::cout << "Count of indexes per thread => " << countsPerThread << std::endl;
#endif

    int upperBound = countsPerThread * threadsCount;
    result = findPatternsOccurenciesInPrifix(pattern.length(), prefix, upperBound,

    if (result == -1)
    {
        for (int i = threadsCount; i > 0; --i)
        {
            result = findPatternsOccurenciesInPrifix(pattern.length(), prefix, coun
            if (result == -1)
            {
                continue;
            }

            break;
        }
    }

    return result != -1 ? str.length() - result : result;
}

int main()
{
    std::string stringA;
    std::cin >> stringA;
    std::string stringB;
    std::cin >> stringB;

    std::cout << findIndexOfCyclicOffset(stringB, stringA) << std::endl;
}

```