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 Кнутта-Морриса-Пратта

Студент гр. 8383 ___ Бабенко Н.С.

Преподаватель ____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы

Изучение алгоритма Кнутта-Морриса-Пратта для поиска подстроки в строке.

Задание 1

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P (|P|≤1500) 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

Sample Input:

ab

abab

Sample Output:

0,2

Задание 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 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

Индивидуализация

Вариант 2

Оптимизация по памяти: программа должна требовать O(m) памяти, где m - длина образца

Описание префикс-функции

Префикс функция($\pi(s, i)$) – функция, которая, по заданному индексу строки, возвращает, по срезу строки до переданного индекса - длину максимального суффикса, который равен префиксу.

Из работы алгоритма подсчёта префикс-функции, стоит отметить, что работает он на 3-х простых свойствах. Опишем работу вычисления $\pi(i)$. Из обозначений и условия, пусть дана строка s размера n и значение $k = \pi(\ i - 1)$ (срезом строки s будем называть строку $s[x\ ...\ y]$, начинающуюся с индекса x размера $y - x$) необходимо вычислить $\pi(i)$:

1. Может оказаться достаточным сравнить два символа $s[i]$ и $s[k]$, действительно, если эти символы оказались равны, то символ $s[i]$ дополняет суффикс строки $s[0\ ...\ i]$ размера k так, что он равен префиксу размера $k + 1$, таким образом, можно сказать, что $\pi(i) = k + 1$;
2. Если же символы оказались различны, тогда будем брать значение префикс-функции префикс-функции(то есть производить присваивание $k = \pi(k)$), действительно, $s[0\ ...\ k]$ является префиксом и суффиксом строки $s[0\ ...\ i]$, переходим к шагу 1.

Таким образом, итерации происходят, пока k не стало равным 0, после этого выполняем проверку на то, являются ли символы $s[0]$ и $s[i]$ равными, в случае равенства $-\ \pi(i) = 1$.

Описание алгоритма задания 1

В программе используется алгоритм Кнутта-Морриса-Пратта с оптимизацией по памяти. Он заключается в том, что для строки вида **<паттерн>** и в дальнейшем для каждого символа строки **<текст>** выполняется вычисление префикс-функции. Особенность индивидуализации в том, что префикс-функция для строки **<текст>** не хранится в массиве, а для каждого элемента вычисляется и используется in-place. Тем самым достигается сложность по памяти O(m), где m – длина паттерна. Для каждого полученного значения выполняется поиск значений равных длине паттерна. Индекс такого значения, за вычетом длины паттерна, и есть индекс начала паттерна в тексте.

Шаги алгоритма Кнутта-Морриса-Пратта:

1. Выполняется вычисление префикс-функции для строки **<паттерн>**;
2. Проходимся по строке **<текст>** и вычисляем значение префикс функции для каждой подстроки вида **<паттерн>+text[0…i]**, где **text - <текст>.**
3. Так как используется оптимизация по памяти, то мы храним только значения префикс-функции для паттерна и каждого очередного элемента текста.
4. Для каждого посчитанного элемента выполняем проверку на равенство длине паттерна, в случае равенства, выполняется запись в результирующий вектор индекса начала вхождения паттерна в текст (продолжая работу алгоритма над примером выше, получаем, что для индексов 3, 5 в массиве значений префикс-функции склейки, выполняется равенство длине паттерна, записываем индекс начала вхождения паттерна ($idx - 2 \times m + 1$, где idx – индекс, на котором было обнаружено равенство значений префикс функции и длины паттерна, а m – длина паттерна)).

Описание алгоритма задания 2

Сдвиг строки, относительно исходной, можно найти путём удвоения циклически сдвинутой строки, если она таковой является, то в такой строке содержится исходная. Таким образом, используя алгоритм Кнутта-Морриса-Пратта, можно найти индекс циклического сдвига, приняв за паттерн - исходную строку, а за текст – удвоенную циклически сдвинутую.

Сложность алгоритма

Сложность алгоритма вычисления префикс функции – $O(n)$, где n – длина строки. Обусловлена тем, что k (значение префикс-функции) на каждой итерации цикла for не может быть увеличено больше, чем на единицу, либо k сохраняет нулевое значение. На итерациях цикла $while$, k может только уменьшаться (т.к. берётся значение префикс-функции от k), таким образом, максимальное кол-во итераций цикла $while$ за один цикл for не превышает $O(n)$ (для строки вида “$s_0 s_0 \ldots s_0 s_1$”), таким образом сложность вычисления префикс-функции по времени составляет $O(n + n) = O(n)$.

Сложность алгоритма 1 по времени составляет - $O(m + n)$, где m-длина паттерна, а n – длина текста, эта сложность обусловлена вычислением префикс-функции для склейки $< \text{паттерн} > + < \text{текст} >$.

Сложность алгоритма 2 по времени составляет – $O(n)$, где n – длина паттерна (исходной строки), обусловлена тем, что алгоритм начинает работу

только со строками одинаковой длины, таким образом для склейки вида $<$ паттерн $>+<$ сдвиг $>+<$ сдвиг $>$ сложность вычисления префикс-функции будет составлять $O(n+n+n)$, что эквивалентно $O(n)$.

Сложность по памяти для обоих алгоритмов составляет – $O(m)$, где m – длина паттерна. Эта сложность обусловлена тем, что для вычисления i-го значения префикс-функции, ей необходимо значение $\pi(i-1)$ и первые m элементов префикс-функции. Эта идея основана на том факте, что нет необходимости сохранять значение префикс функции, которое превышает длину паттерна. Таким образом, можно посчитать значения префикс-функции только для первых m элементов в склейке паттерна и строки, в которой его необходимо найти.

Описание функций и структур данных

Class ConcatStrings – структура, для хранения склейки строк без использования доп. памяти (использование ссылок на них).

Поля ConcatStrings:

- `const string &p` – паттерн;
- `const string &t` – текст;

Методы ConcatStrings:

- `ConcatStrings(const string &p, const string &t)` – конструктор для заполнения паттерна и строки, в которойведётся поиск;
- `char operator[](size_t)` – возвращает по заданному индексу значение строки $p+t$;

Class PrefixFunction – структура, для вычисления префикс функции от строки.

Поля PrefixFunction:

- `ConcatStrings str` – строка, склейка паттерна и текста;
- `size_t patternSize` – длина паттерна;
- `size_t currIndex` – текущий индекс, для вычисления $\pi(i)$;
- `vector <size_t> prefixValues` – вектор, в котором хранятся значения префикс-функции паттерна;

Методы PrefixFunction:

- `PrefixFunction(const string &p, const string &t)` – конструктор, в котором вычисляется значение префикс-функции

паттерна;

- `size_t nextValue()` – метод, возвращающая следующее значение префикс-функции;

Функции KMP1.cpp:

- `vector <size_t> getSubstringIndices(const string &p, const string &t)` – функция, возвращающая вектор индексов, на которых p встречается в t.

Функции KMP2.cpp:

- `int getShiftingIndex(const string &p, const string &t)` – функция, возвращающая индекс, на котором p встречается в t + t.

Тестирование

Задание 1:

Тест 1:

```
ab
aaaaaaaaaab
```

Вывод:

```
9
```

Тест 2:

```
aba
abaababbaabababababa
```

Вывод:

```
0,3,9,11,13,15
```

Тест 3:

```
baba
abbabbabba
```

Вывод:

```
-1
```

Тест 4:

```
aaa
aabaaaaaaabaa
```

Вывод:

```
3,4,5,6,7
```

Тест 5:

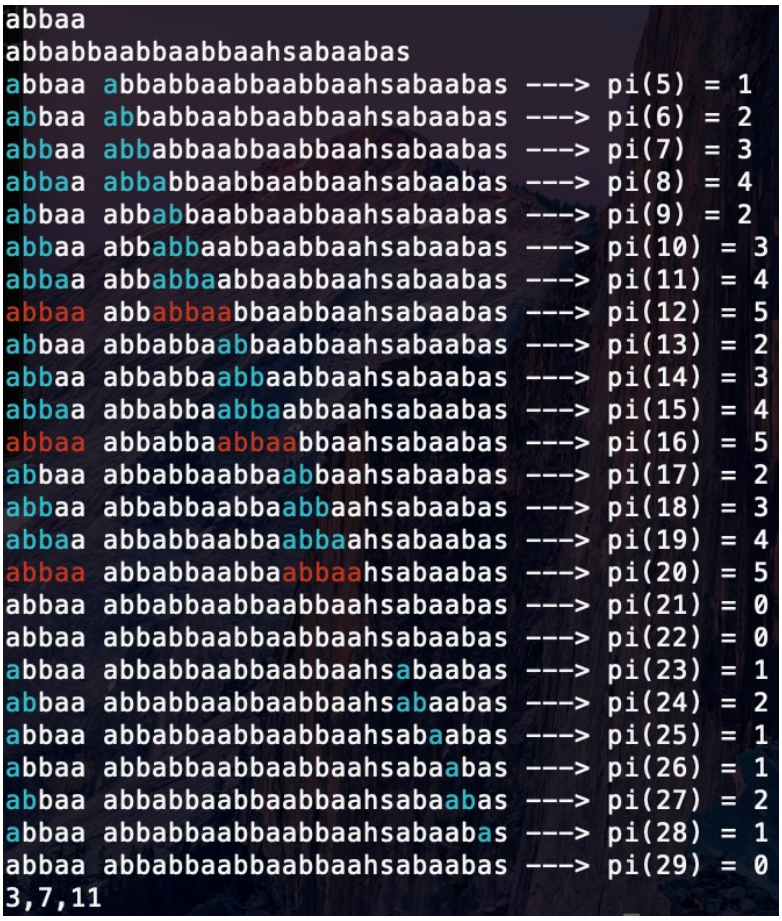
```
aaaaaaa

aaaaaaa
```

Вывод:

```
0
```

Тест с подробным промежуточным выводом:



Задание 2:

Тест 1:

```
defabc
abcdef
```

Вывод:

```
3
```

Тест 2:

```
iercour
courier
```

Вывод:

```
3
```

Тест 4:

```
hello
hello
```

Вывод:

```
0
```

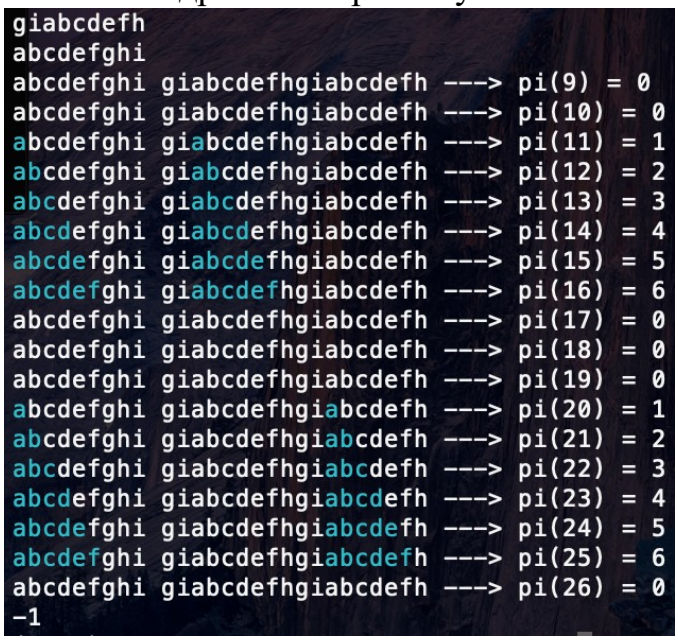
Тест 5:

```
image
qwert
```

Вывод:

```
-1
```

Тест 4 с подробным промежуточным выводом:



Вывод

В ходе выполнения лабораторной работы были изучены алгоритм Кнутта-Морриса-Пратта для нахождения подстроки в строке, а также префикс-функция для нахождения наибольшего суффикса равного префиксу в строке.

Приложение А

Исходный код KMP1.cpp

```
#include <iostream>
#include <vector>
#include <cstdio>
#define DEBUG
#ifdef DEBUG
#define RESET "\033[0m"
#define SET_COLOR "\033[36m"
#define SET_SUBSTR_COLOR "\033[31m"
#endif

using namespace std;

class ConcatStrings {
 const string &p;
 const string &t;
 public:
  ConcatStrings(const string &p, const string &t) : p(p), t(t) {}
  char operator[](size_t i)
  {
      if (i < p.size())
          return p[i];

      return t[(i - p.size()) % t.size()];
  }
};

class PrefixFunction {
  ConcatStrings str;
  size_t patternSize;
  size_t currIndex;
  vector <size_t> prefixValues;
 public:
  PrefixFunction(const string& p, const string& t) : str(p, t),
patternSize(p.size()), currIndex(p.size())
  {
      // выделяем память под размер паттерна
      prefixValues.resize(patternSize);
      prefixValues[0] = 0;

      for (size_t i = 1; i < patternSize; i++)
      {
          // берём значение вычисленное на предыдущем шаге
          auto k = prefixValues[i - 1];

          // пока новый последний символ суффикса не равен последнему символу
префикса
          while (k > 0 && p[i] != p[k])
              // изменяем значение префикс-функции, на значение вычисленное п/ф
для k-1
              k = prefixValues[k - 1];

          //если конец префикса стал равен новому концу суффикса, то увеличиваем k
на один,
```

```
            //иначе не было найдено такого символа на конце префикса, равное концу
суффикса
            prefixValues[i] = (p[i] == p[k]) ? k + 1 : 0;
        }
    }

    size_t nextValue()
    {
        static size_t k = 0;

        // пока новый последний символ суффикса не равен последнему символу префикса
        while (k > 0 && str[currIndex] != str[k])
            // изменяем значение префикс-функции, на значение вычисленное п/ф для k-
1
            k = prefixValues[k-1];

        // если конец префикса стал равен новому концу суффикса, то увеличиваем k на
один
        if (str[currIndex++] == str[k])
            k++;

        // если значение п/ф оказалось равным длине паттерна, тогда возвращаем её,
        // а значение префикс функции на тек. итерации изменяем на значение п/ф от
k-1
        if (k == patternSize)
        {
            k = prefixValues[k-1];
            return patternSize;
        }

        return k;
    }
};

#ifdef DEBUG
void debugPrint(const string &str, size_t currInd, size_t val, size_t patternSize)
{
    currInd += patternSize;
    cout << (val == patternSize ? SET_SUBSTR_COLOR : SET_COLOR);
    for (size_t i = 0; i < val; i++)
        cout << str[i];
    cout << RESET;

    for (size_t i = val; i < currInd - val + 1; i++)
    {
        if (i == patternSize)
            cout << ' ';
        cout << str[i];
    }

    cout << (val == patternSize ? SET_SUBSTR_COLOR : SET_COLOR);
    for (size_t i = currInd - val + 1; i < currInd + 1; i++)
    {
        if (i == patternSize)
            cout << ' ';
        cout << str[i];
```

```
    }
    cout << RESET;

    for (size_t i = currInd + 1; i < str.size(); i++)
        cout << str[i];

    cout << " ---> " << "pi(" << currInd << ") = " << val << endl;
}
#endif

vector <size_t> getSubstringIndices(const string &p, const string &t)
{
    PrefixFunction prefixFunction(p, t);
    vector <size_t> substringIndices;
#ifdef DEBUG
    const string str = p + t;
#endif

    for (size_t i = 0; i < t.size(); i++)
    {
        size_t val = prefixFunction.nextValue();
#ifdef DEBUG
        debugPrint(str, i, val, p.size());
#endif
        //если значение п/ф совпало с длиной паттерна - кладём в вектор индекс
начала вхождения паттерна
        if (val ==  p.size())
            substringIndices.push_back(i - p.size() + 1);
    }

    return substringIndices;
}

int main()
{
    string p, t;
    cin >> p >> t;

    auto result = getSubstringIndices(p, t);
    if (result.empty())
        cout << -1;
    else for (size_t i = 0; i < result.size(); i++)
        {
            cout << result[i];
            if (i + 1 != result.size())
                cout << ',';
        }
    cout << endl;
    return 0;
}
```

Исходный код KMP2.cpp

```
#include <iostream>
#include <vector>
#include <cstdio>
#define DEBUG
#ifdef DEBUG
#define RESET "\033[0m"
#define SET_COLOR "\033[36m"
#endif

using namespace std;

class ConcatStrings {
    const string &p;
    const string &t;
public:
    ConcatStrings(const string &p, const string &t) : p(p), t(t) {}
    char operator[](size_t i)
    {
        if (i < p.size())
            return p[i];

        return t[(i - p.size()) % t.size()];
    }
};

class PrefixFunction {
    ConcatStrings str;
    size_t patternSize;
    size_t currIndex;
    vector <size_t> prefixValues;
public:
    PrefixFunction(const string& p, const string& t) : str(p, t),
patternSize(p.size()), currIndex(p.size())
    {
        // выделяем память под размер паттерна
        prefixValues.resize(patternSize);
        prefixValues[0] = 0;

        for (size_t i = 1; i < patternSize; i++)
        {
            // берём значение вычисленное на предыдущем шаге
            auto k = prefixValues[i - 1];

            // пока новый последний символ суффикса не равен последнему символу
префикса
            while (k > 0 && p[i] != p[k])
                // изменяем значение префикс-функции, на значение вычисленное п/ф
для k-1
                k = prefixValues[k - 1];

            //если конец префикса стал равен новому концу суффикса, то увеличиваем
k на один,
            //иначе не было найдено такого символа на конце префикса, равное концу
```

```
суффикса
            prefixValues[i] = (p[i] == p[k]) ? k + 1 : 0;
        }
    }

    size_t nextValue()
    {
        static size_t k = 0;

        // пока новый последний символ суффикса не равен последнему символу
префикса
        while (k > 0 && str[currIndex] != str[k])
            // изменяем значение префикс-функции, на значение вычисленное п/ф для
k-1
            k = prefixValues[k-1];

        // если конец префикса стал равен новому концу суффикса, то увеличиваем k
на один
        if (str[currIndex++] == str[k])
            k++;

        // если значение п/ф оказалось равным длине паттерна, тогда возвращаем её,
        // а значение префикс функции на тек. итерации изменяем на значение п/ф от
k-1
        if (k == patternSize)
        {
            k = prefixValues[k-1];
            return patternSize;
        }

        return k;
    }
};

#ifdef DEBUG
void debugPrint(const string &str, size_t currInd, size_t val)
{
    currInd += str.size() / 3;
    cout << SET_COLOR;
    for (size_t i = 0; i < val; i++)
        cout << str[i];
    cout << RESET;

    for (size_t i = val; i < currInd - val + 1; i++)
    {
        if (i == str.size() / 3)
            cout << ' ';
        cout << str[i];
    }

    cout << SET_COLOR;
    for (size_t i = currInd - val + 1; i < currInd + 1; i++)
    {
        if (i == str.size() / 3)
            cout << ' ';
        cout << str[i];
```

```
    }
    cout << RESET;

    for (size_t i = currInd + 1; i < str.size(); i++)
        cout << str[i];

    cout << " ---> " << "pi(" << currInd << ") = " << val << endl;
}
#endif

int getShiftingIndex(const string &p, const string &t)
{
    if (p.size() != t.size())
        return -1;

    PrefixFunction prefixFunction(p, t);
#ifdef DEBUG
    const string str = p + t + t;
#endif

    for (size_t i = 0; i < 2 * t.size(); i++)
    {
        size_t val = prefixFunction.nextValue();
#ifdef DEBUG
        debugPrint(str, i, val);
#endif
        //если значение п/ф совпало с длиной паттерна - возвращаем индекс начала
сдвига
        if (val == p.size())
            return int(i - p.size() + 1);
    }

    return -1;
}

int main()
{
    string p, t;

    cin >> t >> p;
    cout << getShiftingIndex(p, t) << endl;

    return 0;
}
```