

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

Юрьев С.Ю.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2021

Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта для поиска подстроки в тексте. Реализовать соответствующую программу.

Задание.

1:

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

2:

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

Описание алгоритма.**Вычисление префикс функции:**

1) Создается вектор длины n , заполняемый нулями. В нем будут храниться максимальные префикс-суффиксы для различных префиксов строки.

2) Далее следует проверка первой подстроки – ей будет являться подстрока длины 2, тогда сравниваются ее первый и последний символ и если они совпадают, то значение максимального префикс-суффикса для данной подстроки становится равным 1.

3) Далее идет проверка следующей подстроки. Ее максимальный префикс-суффикс может получиться из дополнения уже имеющегося префикс-суффикса. Проверяется следующий за предыдущим максимальным префиксом-суффиксом символ и если он совпадает с последним символом текущей подстроки, то макс. префикс можно увеличить еще на один. Если не совпадает, то рассматриваемый префикс-суффикс для дополнения меняется. Им становится максимальный префикс-суффикс рассмотренного префикс-суффикса. Повторяются аналогичные действия пока не будет найдено совпадение символов (следующего за рассматриваемым префиксом и последним в текущей подстроке), тогда в вектор максимальных префикс-суффиксов кладется длина рассматриваемого префикса, увеличенного на 1. Или когда рассматриваемый префикс не будет пустым. Если пуст, то при

несовпадении первого символа с последним символом данной подстроки длине максимального префикса-суффикса текущей подстроки присваивается 0.

4) В результате получится вектор, состоящий из максимальных префиксов-суффиксов для префиксов исходной строки.

Поиск подстроки:

1) На первом шаге вычисляется префикс-функция подстроки. Заводится переменная, показывающая количество совпавших символов на текущем шаге.

2) Далее идет по циклу проверка символов (следующий за совпавшей частью и текущий символ в цикле). Каждый совпавший символ увеличивает длину совпадающей части на 1.

3) В случае несовпадения символов длина совпадающей части становится равной максимальному префикс-суффиксу этой части.

4) Идет повтор 2 и 3го пункта пока длина совпадающей части не станет равна длине подстроки (это значит подстрока нашлась) или когда не закончится цикл, рассматривающий символы(это значит строка не нашлась).

Определение циклического сдвига:

1) Строка 1 будет являться циклическим сдвигом строки 2, если строка 2 будет содержаться в удвоенной строке 1.

2) Запускается алгоритм КМП поиска строки 2 в строке, состоящей из двух строк 1 и если она находится, значит строка 1 – циклический сдвиг строки 2.

Описание функций и структур данных.

Для хранения строк использовался класс `std::string`.

`std::vector<int> countPrefixFunction(std::string str)` — вычисляет префикс-функцию.

Принимает на вход строку, для которой будет вычисляться префикс функция.

Возвращает вектор из длин максимальных префиксов, являющихся также суффиксами для подстрок полученной строки. Значения индексов в векторе равны длинам подстрок - 1.

`std::vector<int> findPatternWithKMP(std::string text, std::string pattern)` - находит индексы вхождений подстроки в текст при помощи алгоритма Кнута-Морриса-Практа.

Принимает на вход:

`std::string text` — текст, в котором ищется искомый шаблон.

`std::string pattern` — искомый шаблон.

Возвращает массив индексов вхождений подстроки в тексте. Если подстрока ни разу не входит в текст, то возвращается -1.

`void printIndexes(std::vector<int> indexes)` - выводит индексы вхождений подстроки.

Принимает на вход массив индексов.

`int circularShift(std::string shifted, std::string original)` - проверяет, является

ли строка циклическим сдвигом другой.

Принимает на вход:

std::string shifted — сдвинутая строка

std::string original — исходная строка

Возвращает индекс вхождения префикса исходной строки в сдвинутую или -1, если строка не является сдвигом.

void doCircularShiftCheck() - функция-менеджер, которая вызывает другие функции в нужном порядке для выполнения 2-го задания.

void doKMPSearch() - функция-менеджер, которая вызывает другие функции в нужном порядке для выполнения 1-го задания.

Оценка сложности.

Сложность алгоритма Кнута-Морриса-Пратта по времени = $O(P+S)$, где P — длина шаблона, а S — длина заданного текста.

Сложность алгоритма по памяти = $O(P)$, где P — длина шаблона.

Тестирование.

Поиск подстроки:

№ п/п	Входные данные	Выходные данные	Комментарий
1	ab ababab	0,2,4	Простейший пример с 3 вхождениями
2	ab abcdefghijklmnoab	0,16	Пример с большим размером текста

3	aa aaaaaaaa	0,1,2,3,4,5,6,7	Наслоение подстроки
4	aa ababa	-1	Отсутствие искомой подстроки
5	aaa aaa	0	Полное совпадение строк

Определение циклического сдвига:

№ п/п	Входные данные	Выходные данные	Комментарий
1	defabc abcdef	3	Простейший пример со сдвигом на 3 символа
2	aaaaabaaaaaaaaaaaaa abaaaaaaaaaaaaaaaaa	4	Пример с большим размером текста
3	aaa aaa	0	Полное совпадение строк
4	aab ba	-1	Разная длина строк
5	abcdefghij abcdefghijh	-1	Не циклический сдвиг

Выводы.

В ходе выполнения работы была изучен алгоритм Кнута-Морриса-Пратта. Также была написана программа, реализующая этот алгоритм.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

lab4_1.cpp :

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

// вычисляет префикс-функцию
std::vector<int> countPrefixFunction(std::string str)
{
#ifdef ADDINFO
    std::cout << "Считаем префикс-функцию для строки \"" << str << "\".\n" << std::endl;
    std::cout << "Значение функции для строки из одного первого символа = 0" << std::endl;
#endif

    int stringLen = str.size();

    std::vector<int> prefixes(stringLen); // вектор из значений максимальных префикс-
    суффиксов
    prefixes[0] = 0; // значение префикс-функции для строки длины 1
    всегда = 0

    for (int i = 1; i < stringLen; i++) // рассмотрим подстроки всех длин
    {
        int borderPrefixLen = prefixes[i - 1]; // это значение граничное (оно может увеличиться
        лишь на 1)
        int prefixLen = borderPrefixLen;

#ifdef ADDINFO
        std::cout << "\nРассматриваем подстроку = \"" << str.substr(0, i + 1) << "\"." << std::endl;
        std::cout << "Длина \"граничного\" префикса предыдущей подстроки: " << str.substr(0,
        prefixLen) << std::endl;
#endif

        while (prefixLen > 0 && str[prefixLen] != str[i]) // если последние символы не
        совпали - уменьшаем длину
        {
            prefixLen = prefixes[prefixLen - 1]; // полагая ее равной p(k-1)

#ifdef ADDINFO
            std::cout << "Крайние символы не совпали -> переходим к длине префикса = " << pre-
            fixLen << "." << std::endl;
#endif
        }
    }
```



```

        if (str[prefixLen] == str[i]) // если последний добавленный символ равен
последнему символу в только что увеличенном префиксе
        {
            prefixLen += 1;          // увеличим длину префикса и переходим к
следующему индексу

#ifdef ADDINFO
            std::cout << "Крайние символы совпали -> увеличиваем длину префикса на 1." <<
std::endl;
#endif
        }

        else if (prefixLen != 0)      // если же мы дошли до длины 0, но так и
не нашли совпадения
        {
            prefixLen = prefixes[prefixLen - 1]; // то останавливаем перебор,
полагаем значение равным 0, и переходим к следующему индексу

#ifdef ADDINFO
            std::cout << "Мы дошли до минимальной длины, но так и не получили совпадение." <<
std::endl;
#endif
        }

        prefixes[i] = prefixLen;

#ifdef ADDINFO
        std::cout << "На этом шаге в итоге получен префикс длины = " << prefixLen << "." <<
std::endl;
#endif
    }

#ifdef ADDINFO
    std::cout << "\nИтоговая полученная префикс-функция: ";
    for (auto& it : prefixes)
        std::cout << it << " ";
    std::cout << "\n";
#endif
    return prefixes;
}

// находит индексы вхождений подстроки в текст
std::vector<int> findPatternWithKMP(std::string text, std::string pattern)
{
#ifdef ADDINFO
    std::cout << "Вызов алгоритма Кнута-Морриса-Пратта\n";
    std::cout << "Текст: " << text << "\nШаблон: " << pattern << "\n\n";
#endif
}

```

```

    int n = text.size();
    int m = pattern.size();

    std::vector<int> indexes;
    std::vector<int> pattern_prefixes = countPrefixFunction(pattern);
    int match = 0;

#ifdef ADDINFO
    std::cout << "\nПереход к поиску подстроки:\n" << std::endl;
#endif

    for (int i = 0; i < n; i++)
    {
#ifdef ADDINFO
        std::cout << "Длина текущего совпадения: " << match << "\n";
        std::cout << "Символ текста: " << text[i] << "\nСимвол шаблона: " <<
pattern[match % m] << "\n\n";
#endif

        while (match > 0 && pattern[match] != text[i]) // обновление совпадающей
части
        {
#ifdef ADDINFO
            std::string pre = "_";
            int newMatch = pattern_prefixes[match - 1];
            if (newMatch > 0)
                pre = pattern.substr(0, newMatch);
            std::cout << "Обновление совпадающей части " << pattern.substr(0,
match) << " -> " << pre << "\n";
#endif

            match = pattern_prefixes[match - 1];
        }

        if (pattern[match] == text[i]) // если символы на текущей позиции в строках
совпадают
        {
#ifdef ADDINFO
            std::cout << "Символы совпадают. Увеличение длины совпадающей части\n\n";
#endif

            match++;
            // то длина совпадения увеличивается
        }

        if (match == m) // если длина совпадения равна длине шаблона
        {
#ifdef ADDINFO
            std::cout << "Найдено совпадение\nИндекс начала искомой строки в тексте: " << i -
match + 1 << std::endl;
#endif

```

```

#endif

        indexes.push_back(i - match + 1); // то добавим это совпадение в
список индексов
    }

#ifdef ADDINFO
    if (match == 0)
        std::cout << "Найден несовпадающий символ. Длина совпадения = 0\n" <<
std::endl;
#endif

    }

#ifdef ADDINFO
    std::cout << "Алгоритм завершил работу\n";
#endif
    return indexes;
}

// выводит индексы вхождений подстроки
void printIndexes(std::vector<int> indexes)
{
    if (!indexes.empty())
    {
        int indexesCount = indexes.size() - 1;
        for (int i = 0; i < indexesCount; i++)
        {
            std::cout << indexes[i] << ",";
        }
        std::cout << indexes[indexesCount];

        return;
    }

    std::cout << "-1";
}

// проверка циклического сдвига
int circularShift(std::string shifted, std::string original)
{
#ifdef ADDINFO
    std::cout << "Проверка циклического сдвига. \n\n";
#endif

    if (shifted.size() != original.size())
        return -1;
}

```

```

        std::vector<int> indexes = findPatternWithKMP(shifted + shifted, original);
        if (!indexes.empty())
            return indexes[0];

        return -1;
    }

// выполняет второе задание
void doCircularShiftCheck()
{
    std::string s1;
    std::string s2;

    std::cin >> s2 >> s1;          // считывание ввода пользователя

    int index = circularShift(s2, s1); // вызов самого алгоритма

#ifdef ADDINFO
    if (index != -1)
    {
        std::cout << "Строка " << s2 << " является циклическим сдвигом строки " <<
s1 << "\n";
        std::cout << "Индекс начала вхождения: ";
    }
    else
    {
        std::cout << "Строка " << s2 << " не является является циклическим сдвигом
строки " << s1 << "\n";
        std::cout << "Выведен код ошибки: ";
    }
}
#endif

    std::cout << index;          // вывод полученного ответа
}

// выполняет первое задание
void doKMPSearch()
{
    std::string s1;
    std::string s2;

    std::cin >> s2 >> s1; // считывание ввода пользователя

    std::vector<int> indexes = findPatternWithKMP(s1, s2); // вызов самого алгоритма

#ifdef ADDINFO
    if (!indexes.empty())

```

```

    {
        std::cout << "Индексы вхождений строки " << s2 << " в " << s1 << ": ";
    }
    else
    {
        std::cout << "Строка " << s2 << " не входит в" << s1 << "\n";
        std::cout << "Результат: ";
    }
#endif

    printIndexes(indexes); // выводит готовый ответ
}

```

```

int main()
{
    setlocale(LC_ALL, "rus");
    doKMPSearch();          // 1 задание
    //doCircularShiftCheck(); // 2 задание
    return 0;
}

```

lab4_2.cpp :

```

#include <iostream>
#include <string>
#include <vector>
#define ADDINFO

// вычисляет префикс-функцию
std::vector<int> countPrefixFunction(std::string str)
{
    #ifdef ADDINFO
        std::cout << "Считаем префикс-функцию для строки \"" << str << "\".\n" << std::endl;
        std::cout << "Значение функции для строки из одного первого символа = 0" << std::endl;
    #endif

    int stringLen = str.size();

    std::vector<int> prefixes(stringLen); // вектор из значений максимальных префикс-
    суффиксов
    prefixes[0] = 0;                      // значение префикс-функции для строки длины 1
    всегда = 0

    for (int i = 1; i < stringLen; i++) // рассмотрим подстроки всех длин
    {

```

```
int borderPrefixLen = prefixes[i - 1]; // это значение граничное (оно может увеличиться  
лишь на 1)
```

```
int prefixLen = borderPrefixLen;
```

```
#ifdef ADDINFO
```

```
std::cout << "\nРассматриваем подстроку = \"" << str.substr(0, i + 1) << "\"." << std::endl;
```

```
std::cout << "Длина \"граничного\" префикса предыдущей подстроки: " << str.substr(0,  
prefixLen) << std::endl;
```

```
#endif
```

```
while (prefixLen > 0 && str[prefixLen] != str[i]) // если последние символы не  
совпали - уменьшаем длину
```

```
{
```

```
    prefixLen = prefixes[prefixLen - 1]; // полагая ее равной p(k-1)
```

```
#ifdef ADDINFO
```

```
std::cout << "Крайние символы не совпали -> переходим к длине префикса = " << pre-  
fixLen << "." << std::endl;
```

```
#endif
```

```
}
```

```
if (str[prefixLen] == str[i]) // если последний добавленный символ равен  
последнему символу в только что увеличенном префиксе
```

```
{
```

```
    prefixLen += 1; // увеличим длину префикса и переходим к  
следующему индексу
```

```
#ifdef ADDINFO
```

```
std::cout << "Крайние символы совпали -> увеличиваем длину префикса на 1." <<  
std::endl;
```

```
#endif
```

```
}
```

```
else if (prefixLen != 0) // если же мы дошли до длины 0, но так и  
не нашли совпадения
```

```
{
```

```
    prefixLen = prefixes[prefixLen - 1]; // то останавливаем перебор,  
полагаем значение равным 0, и переходим к следующему индексу
```

```
#ifdef ADDINFO
```

```
std::cout << "Мы дошли до минимальной длины, но так и не получили совпадение." <<  
std::endl;
```

```
#endif
```

```
}
```

```
prefixes[i] = prefixLen;
```

```
#ifdef ADDINFO
```

```
std::cout << "На этом шаге в итоге получен префикс длины = " << prefixLen << "." <<  
std::endl;
```

```
#endif
```

```

    }

#ifdef ADDINFO
    std::cout << "\nИтоговая полученная префикс-функция: ";
    for (auto& it : prefixes)
        std::cout << it << " ";
    std::cout << "\n";
#endif
    return prefixes;
}

// находит индексы вхождений подстроки в текст
std::vector<int> findPatternWithKMP(std::string text, std::string pattern)
{
#ifdef ADDINFO
    std::cout << "Вызов алгоритма Кнута-Морриса-Пратта\n";
    std::cout << "Текст: " << text << "\nШаблон: " << pattern << "\n\n";
#endif

    int n = text.size();
    int m = pattern.size();

    std::vector<int> indexes;
    std::vector<int> pattern_prefixes = countPrefixFunction(pattern);
    int match = 0;

#ifdef ADDINFO
    std::cout << "\nПереход к поиску подстроки:\n" << std::endl;
#endif

    for (int i = 0; i < n; i++)
    {
#ifdef ADDINFO
        std::cout << "Длина текущего совпадения: " << match << "\n";
        std::cout << "Символ текста: " << text[i] << "\nСимвол шаблона: " <<
pattern[match % m] << "\n\n";
#endif

        while (match > 0 && pattern[match] != text[i]) // обновление совпадающей
части
        {
#ifdef ADDINFO
            std::string pre = "_";
            int newMatch = pattern_prefixes[match - 1];
            if (newMatch > 0)
                pre = pattern.substr(0, newMatch);
            std::cout << "Обновление совпадающей части " << pattern.substr(0,
match) << " -> " << pre << "\n";

```

```

#endif

        match = pattern_prefixes[match - 1];
    }

    if (pattern[match] == text[i]) // если символы на текущей позиции в строках
совпадают
    {
#ifdef ADDINFO
        std::cout << "Символы совпадают. Увеличение длины совпадающей части\n\n";
#endif

        match++; // то длина совпадения увеличивается
    }

    if (match == m) // если длина совпадения равна длине шаблона
    {
#ifdef ADDINFO
        std::cout << "Найдено совпадение\nИндекс начала искомой строки в тексте: " << i -
match + 1 << std::endl;
#endif

        indexes.push_back(i - match + 1); // то добавим это совпадение в
список индексов
    }

#ifdef ADDINFO
    if (match == 0)
        std::cout << "Найден несовпадающий символ. Длина совпадения = 0\n" <<
std::endl;
#endif

    }

#ifdef ADDINFO
    std::cout << "Алгоритм завершил работу\n";
#endif
    return indexes;
}

// выводит индексы вхождений подстроки
void printIndexes(std::vector<int> indexes)
{
    if (!indexes.empty())
    {
        int indexesCount = indexes.size() - 1;
        for (int i = 0; i < indexesCount; i++)
        {
            std::cout << indexes[i] << ",";

```



```

    }
    std::cout << indexes[indexesCount];

    return;
}

std::cout << "-1";
}

// проверка циклического сдвига
int circularShift(std::string shifted, std::string original)
{
#ifdef ADDINFO
    std::cout << "Проверка циклического сдвига. \n\n";
#endif

    if (shifted.size() != original.size())
        return -1;

    std::vector<int> indexes = findPatternWithKMP(shifted + shifted, original);
    if (!indexes.empty())
        return indexes[0];

    return -1;
}

// выполняет второе задание
void doCircularShiftCheck()
{
    std::string s1;
    std::string s2;

    std::cin >> s2 >> s1;          // считывание ввода пользователя

    int index = circularShift(s2, s1); // вызов самого алгоритма

#ifdef ADDINFO
    if (index != -1)
    {
        std::cout << "Строка " << s2 << " является циклическим сдвигом строки " <<
s1 << "\n";
        std::cout << "Индекс начала вхождения: ";
    }
    else
    {
        std::cout << "Строка " << s2 << " не является является циклическим сдвигом
строки " << s1 << "\n";
    }
}

```

```

        std::cout << "Выведен код ошибки: ";
    }
#endif

    std::cout << index;          // вывод полученного ответа
}

// выполняет первое задание
void doKMPSearch()
{
    std::string s1;
    std::string s2;

    std::cin >> s2 >> s1; // считывание ввода пользователя

    std::vector<int> indexes = findPatternWithKMP(s1, s2); // вызов самого алгоритма

#ifdef ADDINFO
    if (!indexes.empty())
    {
        std::cout << "Индексы вхождений строки " << s2 << " в " << s1 << ": ";
    }
    else
    {
        std::cout << "Строка " << s2 << " не входит в" << s1 << "\n";
        std::cout << "Результат: ";
    }
}
#endif

    printIndexes(indexes); // выводит готовый ответ
}

int main()
{
    setlocale(LC_ALL, "rus");
    //doKMPSearch();      // 1 задание
    doCircularShiftCheck(); // 2 задание
    return 0;
}

```