

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ
по практической работе №4
по дисциплине «Построение и Анализ Алгоритмов»
Тема: Алгоритм Кнута-Морриса-Пратта

Студент гр. 8382

Вербин К.М.

Преподаватель

Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с алгоритмом Кнута-Морриса-Пратта для поиска подстроки в строке и реализовать его.

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 .

Входные данные

Первая строка - P .

Вторая строка - T .

Выходные данные

индексы начал вхождений P в T , разделенных запятой, если P не входит в T , то вывести -1.

Пример входных данных

ab

abab

Соответствующие выходные данные

0,2

Задание 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.

Входные данные

Первая строка - A .

Вторая строка - B .

Выходные данные

Если A является циклическим сдвигом B , индекс начала строки B в A , иначе вывести -1. Если возможно несколько сдвигов, вывести первый индекс

Пример входных данных

defabc

abcdef

Соответствующие выходные данные

3

Вариант 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

Префикс функция

Префикс-функция для i -го символа возвращает значение, равное максимальной длине совпадающих префикса и суффикса подстроки (префикс длинны n для строки a – это подстрока $a[0, n-1]$, суффикс длинны n для строки a – это подстрока $a[\text{len}(a) - n, \text{len}(a) - 1]$), которая заканчивается i -м символом (максимальная длина равных префикса и суффикса для строки $[0, i]$).

Алгоритм Кнута-Морриса-Пратта (КМП) позволяет находить префикс-функцию от строки за линейное время, и имеет достаточно лаконичную реализацию, по длине не превышающую наивный алгоритм.

Для начала заметим важное свойство: $\pi[i] \leq \pi[i-1] + 1$ и $\pi[i] \leq \pi[i-1] + 1$. То есть префикс-функция от следующего элемента не более чем на 1 превосходит префикс-функцию от текущего. Случай $\pi[i] = \pi[i-1] + 1$ легко изобразить:

$$s[6] = s[2]$$

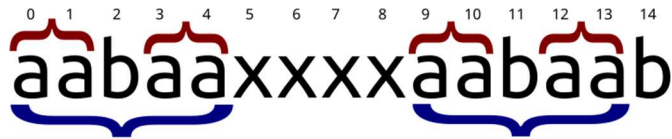
$$\pi[5] = 2$$

$$\pi[6] = \pi[5] + 1 = 3$$

То есть верно следующее утверждение (в 0-индексации):

$$s[i] = s[\pi[i-1]] \Rightarrow \pi[i] = \pi[i-1] + 1$$

Этот случай достаточно тривиален. Но что если $s[i] \neq s[\pi[i-1]]$? Хотелось бы найти такую длину jj , что $s[0..j-1] = s[i-j..i-1]$, но при этом $j < \pi[i-1]$. Если $s[i] = s[j]$, то $\pi[i] = j + 1$. На самом деле, длина jj уже была найдена в процессе нахождения префикс-функции. А именно, $j = \pi[\pi[i-1] - 1]$. Графически это выглядит так:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14


$$s[14] \neq s[\pi[13]]$$

$$j = \pi[\pi[13] - 1] = 2$$

$$s[14] = s[2]$$

$$\pi[14] = j + 1 = 3$$

Если же длина jj также не подходит ($s[i] \neq s[j]$), просто ещё раз уменьшим её по такой же формуле: $j = \pi[j-1]$. Таким образом будем пытаться продолжить префикс длины jj , пока jj не станет равно 0. В таком случае просто сравним $s[i]$ с $s[0]$, и в зависимости от результата присвоим $\pi[i] = 0$ или 1.

Поиск образа в строке

Алгоритм считывает две строки: образец и текст для поиска. Для полученной строки рассчитывается префикс-функция. После получения 2 строк вызывается префикс функция для строки образца. На выходе мы получаем массив чисел равный длине образца.

После вызывается функция поиска подстроки в строке:

- 1) i – индекс символа текста, который обрабатывается на шаге i , k – индекс символа образца с которым мы сравниваем i -тый символ текста.
- 2) Если k равен длине образца, значит мы нашли вхождение образца в текста и выводим символ начала вхождения.

- 3) Проходим по всей строке, проверяя каждый символ: если i -тый и k -тый символы совпадают, то $i = i + 1$, $k = k + 1$. Если символы не совпадают, то $k = \text{prefix}(k-1)$. Это позволяет нам проверять строку за время равное сумме длин двух строк.
- 4) Если ни одного вхождения не было найдено, то выводится -1.

Сложность алгоритма по памяти

Сложность алгоритма по памяти будет $O(P + T)$, т.к. программа хранит 2 строки.

Временная сложность алгоритма

Сложность по времени алгоритма будет $O(P + T)$, т.к. вычисление префикс-функции для образца будет $O(P)$, а поиск подстроки в строке будет выполняться за $O(T)$ тогда весь алгоритм выполняется за $O(P+T)$.

Определение циклического сдвига

В начале алгоритма считываются две строки одинаковой длины A и B , если длины не совпадают, то алгоритм сразу завершает работу и выводит -1, т.к. в таком случае A не является циклическим сдвигом. Затем с помощью той же функции вычисляется префикс-функция для строки B . После этого строка A удваивается, чтобы найти индекс строки B в строке A (если является циклическим сдвигом). После алгоритм заводит два индекса, которые указывают на символы в удвоенной строке A и строке B , после чего алгоритм проходит и сравнивает символы. Если произошло очередное совпадение, то индексы увеличиваются, если происходит несовпадение, то с помощью вектора префикс-функций для B осуществляется "сдвиг" образа (не на 1, как в "наивном" алгоритме, т.к. по префикс-функции можно определить, что часть символов уже совпала). Когда индекс указывающий на символ в образе B равен длине этого образа, это значит, что вхождение найдено. Т.к. A - это

удвоенная строка, то мы определили что исходная строка A является циклическим сдвигом B и алгоритм завершает работу. В противном случае выводится -1.

Сложность алгоритма по памяти

Сложность по памяти будет $O(B + A)$, т.к. в этом случае алгоритм строит префикс-функцию для строки B и на ее хранение требуется B памяти, а удвоение строки требует дополнительно A памяти

Временная сложность алгоритма

Сложность по времени будет $O(2|A| + |B|) = O(3|A|) = O(|A| + |B|)$ т.к. для заполнения префикс-функции нужно $|B|$ операций, а при поиске цикл проходит за линейное время по удвоенной строке A .

Распараллеливание алгоритма

Разделим строку s , в которой осуществляется поиск на n равных (или почти равных) участков. Эти участки пересекаются, причём ширина пересекающихся частей равна $|t| - 1$ (t – образец). Тогда мы получим n участков, на которых можно независимо запустить алгоритм, и получить все вхождения, причём каждое вхождение будет найдено ровно 1 раз. Поскольку алгоритм находит вхождение, когда доходит до его конца, ему нужно пройти $|t|-1$ символов прежде, чем он найдёт хотя бы одно вхождение. В нашем разбиении, вхождения, заканчивающиеся на одном из этих $|t| - 1$ символов, будут найдены экземпляром, обрабатывающим предыдущий участок, поэтому вхождения не теряются. Разумеется, в первых $|t| - 1$ символах исходной строки s вхождения t заканчиваться не могут.

Естественно, что каждый участок не может быть короче, чем образец, поэтому наибольшее число участков, на которые можно разбить строку, будет $|s| - (|t| - 1)$. В этом случае длина каждого участка будет равна $|t|$.

Описание структур данных

`void prefixFunction(std::string& sample, std::vector<int>& prefix)`- вычисляет префикс-функцию для каждого символа переданной строки и сохраняет значения в векторе `prefix`.

- `std::string& sample` – строка, которая обрабатывается префикс функцией.
- `std::vector<int>& prefix` – массив, хранящий результат работы префикс функции.

`bool findEntry(std::string& text, std::string& sample, std::vector<int>& prefix)`- получает на вход текст, образец, вектор префикс-функции, число переданных потоков и вычисляет индексы вхождений образца.

- `std::string& text` – текст, в котором ищется образец.
- `std::string& sample` – образец, искомый в тексте.
- `std::vector<int>& prefix` – массив, хранящий результат работы префикс функции.
- Функция возвращает `true` если образец (сдвиг) был найден хотя бы раз или `false` если не был найден образец (сдвиг)

`void printInterInfo (std::string& splice, std::vector<int> vect, int n)` - выводит промежуточные данные - состояние префикс-функции для текущей части строки (часть строки передается через `n`).

- `std::string& splice` – строка, которая обрабатывается префикс функцией.
- `std::vector<int>& prefix` – массив, хранящий результат работы префикс функции.
- `int n` – индекс, до которого обработала префикс функция.

`std::vector< std::pair<std::string, int >> toFlows(std::string text, const int uFlow, int maxFlow)` – функция разбивает строку `text` на `uFlow` потоков.

- `std::string text` – текст, который разделяется на потоки.

- `const int uFlow` – количество потоков.
- `int maxflow` – максимально возможное количество потоков.
- Функция возвращает вектор, хранящий в себе потоки

Тестирование

Input	Output
Поиск подстроки	
ab abab 1	Индексы: 0,2
a ababfaabfaaabaafhaaa 5	Индексы: 0,2,5,6,9,10,11,13,14,17,18,19
aaa aaaaa 2	Индексы: 0, 1, 2
abbabba abbabbabba 3	Индексы: 0, 4
Определение сдвига	
qwerty ertyqw 2	Найден циклический сдвиг! позиция: 2
qweqweqwe weqweqweq 4	Найден циклический сдвиг! позиция: 1 Найден циклический сдвиг! позиция: 4 Найден циклический сдвиг! позиция: 7

adadadad dadadada 1	Найден циклический сдвиг! позиция: 1
qwertyuiop rtyuiopqwr 3	Не является циклическим сдвигом: -1

Вывод

В ходе работы был разобран и реализован алгоритм Кнута-Морриса-Пратта для поиска вхождений подстроки в строку

ПРИЛОЖЕНИЕ

КОД ПРОГРАММЫ

Поиск подстроки

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <cmath>
#include <climits>

bool INTER = true;
//перегружены оператор вывода вектора в поток
std::ostream& operator<<(std::ostream& stream, std::vector<int> vect) {
    stream << vect[0];
    for (int i = 1; i < vect.size(); i++) {
        stream << "," << vect[i];
    }
    return stream;
}

void printInterInfo(std::string& splice, std::vector<int> vect, int n) {
    std::cout << "\nСостояние префикс-функции для строки: " <<
std::endl;
    for (int i = 0; i <= n; i++)
        std::cout << splice[i] << " ";
    std::cout << std::endl;
    for (int i = 0; i <= n; i++)
        std::cout << vect[i] << " ";
    std::cout << std::endl;
}

void prefixFunction(std::string& sample, std::vector<int>& prefix) {
    prefix[0] = 0;
    std::cout << "Начало расчета префикс-функции" << std::endl;
    int n = 1;
    for (int i = 1; i < sample.length(); i++)
    { // поиск какой префикс-суффикс можно расширить
        int k = prefix[i - 1]; // длина предыдущего префикса
        while (k > 0 && sample[i] != sample[k]) // этот нельзя расширить,
            k = prefix[k - 1]; // берем ранее рассчитанное значение
    }
```

```

        if (sample[k] == sample[i])
            k = k + 1; // расширяем найденный префикс-
суффикс

```

```

        prefix[i] = k;

```

```

        if (INTER)
            printInterInfo(sample, prefix, n++);
    }
}

```

```

std::vector< std::pair<std::string, int >> toFlows(std::string text, const int
uFlow, int maxFlow) {

```

```

    std::vector< std::pair<std::string, int >> tmp;
    int step = maxFlow / uFlow;
    int ind = 0;
    int freeSize = text.length() - ( step * uFlow + text.length() - maxFlow );

```

```

    for (int i = 0; i < uFlow; i++) {
        int len;
        if (uFlow - i == 1) {
            len = text.length() - ind;
        }
        else {
            len = step + text.length() - maxFlow;
            if (freeSize > 0) {
                len++;
            }
        }
        tmp.push_back(std::make_pair(text.substr(ind, len), ind));
        if (freeSize > 0) {
            ind++; freeSize--;
        }
        ind += step;
        std::cout << "Добавлен поток: " << tmp[tmp.size() - 1].first << "
Начальный индекс: " << tmp[tmp.size() - 1].second << std::endl;
    }

```

```

    return tmp;
}
//функция определения сдвига

```

```

    bool findEntry(std::string& text, std::string& sample, std::vector<int>&
prefix) {
        int k = 0; //индекс сравниваемого символа в sample
        bool flag = false;
        if (text.length() <= sample.length()) {
            std::cout << "\nНе может быть образов: ";
            std::cout << "-1";
            return false;
        }
        int maxFlow = text.length() - (sample.length() - 1);
        std::cout << "Максимальное количество потоков: " << maxFlow << "
|Введите количество потоков: ";
        int uFlow;
        std::cin >> uFlow;
        if (uFlow > maxFlow || uFlow < 0) {
            std::cout << "Ошибка в вводе! | Введите количество потоков: ";
            std::cin >> uFlow;
        }
        else {
            std::vector< std::pair<std::string, int >> flows = toFlows(text, uFlow,
maxFlow);
            for (int j = 0; j < flows.size(); j++) {
                std::cout << "\nПоиск образа в потоке: " << flows[j].first << ";";
                for (int i = 0; i < flows[j].first.length(); i++) {
                    while (k > 0 && sample[k] != flows[j].first[i])
                        k = prefix[k - 1];

                    if (sample[k] == flows[j].first[i])
                        k = k + 1;
                    else if (k != 0) k = prefix[k - 1];
                    if (k == sample.length()) {
                        std::cout << "\n\n\tНайден образ! позиция: " <<
flows[j].second + i - sample.length() + 1 << std::endl;
                        flag = true;
                    }

                }
            }
            if (!flag) std::cout << "\nНет образов: " << -1;
            return flag;
        }
    }
}

```

```

int main()
{
    setlocale(LC_ALL, "Russian");
    std::string text;
    std::string sample;
    std::cin >> sample; std::cin >> text;
    std::vector<int> prefix(sample.length());
    prefixFunction(sample, prefix);

    findEntry(text, sample, prefix);
    return 0;
}

```

Циклический сдвиг

```

#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <cmath>
#include <climits>

```

```

bool INTER = true;
//перегружены оператор вывода вектора в поток
std::ostream& operator<<(std::ostream& stream, std::vector<int> vect) {
    stream << vect[0];
    for (int i = 1; i < vect.size(); i++) {
        stream << "," << vect[i];
    }
    return stream;
}

```

```

void printInterInfo(std::string& splice, std::vector<int> vect, int n) {
    std::cout << "\nСостояние префикс-функции для строки: " <<
std::endl;
    for (int i = 0; i <= n; i++)
        std::cout << splice[i] << " ";
    std::cout << std::endl;
    for (int i = 0; i <= n; i++)
        std::cout << vect[i] << " ";
    std::cout << std::endl;
}

```

```

void prefixFunction(std::string& sample, std::vector<int>& prefix) {
    prefix[0] = 0;
    std::cout << "Начало расчета префикс-функции" << std::endl;
    int n = 1;
    for (int i = 1; i < sample.length(); i++)
    { // поиск какой префикс-суффикс можно расширить
        int k = prefix[i - 1]; // длина предыдущего префикса
        while (k > 0 && sample[i] != sample[k]) // этот нельзя расширить,
            k = prefix[k - 1]; // берем ранее рассчитанное значение

        if (sample[k] == sample[i])
            k = k + 1; // расширяем найденный префикс-
суффикс

        prefix[i] = k;

        if (INTER)
            printInterInfo(sample, prefix, n++);
    }
}

```

```

std::vector< std::pair<std::string, int >> toFlows(std::string text, const int
uFlow, int maxFlow) {

    std::vector< std::pair<std::string, int >> tmp;
    int step = maxFlow / uFlow;
    int ind = 0;
    int freeSize = text.length() - ( step * uFlow + text.length() - maxFlow );

    for (int i = 0; i < uFlow; i++) {
        int len;
        if (uFlow - i == 1) {
            len = text.length() - ind;
        }
        else {
            len = step + text.length() - maxFlow;
            if (freeSize > 0) {
                len++;
            }
        }
        tmp.push_back(std::make_pair(text.substr(ind, len), ind));
        if (freeSize > 0) {

```

```

        ind++; freeSize--;
    }
    ind += step;
    std::cout << "Добавлен поток: " << tmp[tmp.size() - 1].first << "
Начальный индекс: " << tmp[tmp.size() - 1].second << std::endl;
}

return tmp;
}

//функция определения сдвига
bool findEntry(std::string& text, std::string& sample, std::vector<int>&
prefix) {
    int k = 0; //индекс сравниваемого символа в sample
    bool flag = false;
    if (text.length() != sample.length()) {
        std::cout << "\nНе является циклическим сдвигом: ";
        std::cout << "-1";
        return false;
    }
    text = text + text;
    int maxFlow = text.length() - (sample.length() - 1);
    std::cout << "Максимальное количество потоков: " << maxFlow << " |
Введите количество потоков: ";
    int uFlow;
    std::cin >> uFlow;
    if (uFlow > maxFlow || uFlow < 0) {
        std::cout << "Ошибка в вводе! | Введите количество потоков: ";
        std::cin >> uFlow;
    }
    else {
        std::vector< std::pair<std::string, int >> flows = toFlows(text, uFlow,
maxFlow);
        std::cout << std::endl;
        for (int j = 0; j < flows.size(); j++) {
            std::cout << "Поиск циклического сдвига в потоке: " <<
flows[j].first << ";" << std::endl;
            for (int i = 0; i < flows[j].first.length(); i++) {
                while (k > 0 && sample[k] != flows[j].first[i])
                    k = prefix[k - 1];

                if (sample[k] == flows[j].first[i])
                    k = k + 1;
                else if (k != 0) k = prefix[k - 1];
            }
        }
    }
}

```

```

        if (k == sample.length()) {
            std::cout << "\n\tНайден циклический сдвиг! позиция: " <<
flows[j].second + i - sample.length() + 1 << std::endl;
            flag = true;
        }

    }
}
if (!flag) {
    std::cout << "\nНе является циклическим сдвигом: ";
    std::cout << "-1";
}

}
return flag;
}

```

```

int main()
{
    setlocale(LC_ALL, "Russian");
    std::string text;
    std::string sample;
    std::cin >> sample; std::cin >> text;
    std::vector<int> prefix(sample.length());
    prefixFunction(sample, prefix);

    findEntry(text, sample, prefix);
    return 0;
}

```