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. 8383 _____ Мирсков А.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы

Изучить принцип работы алгоритма Кнута-Мориса-Прата и реализовать с его помощью программы для поиска вхождений строки и проверки на циклический сдвиг.

Постановка задачи

Вар. 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).

Реализуйте алгоритм KMP и с его помощью для заданных шаблона P (|P|⩽1500) и текста T (|T|⩽5000000) и найдите все вхождения P в T.

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:

ababab

Sample Output:

0,2

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.

Sample Input:

```
defabc
abcdef
```

Sample Output:

```
3
```

Реализация

Префикс функция. Значения вычисляются по очереди. Значение pi[0] считается равным нулю равным нулю. Изначально длина текущего рассматриваемого образца равно предыдущему значению функции. Далее пока текущий символ строки и символ, стоящий в позиции с индексом равным длине текущего рассматриваемого образца различаются, уменьшаем длину образца, полагая её равной уже вычисленному значению функции с индексом, равным длине образца. Если длина стала равна 0, присваиваем значению функции 0, иначе длину образца + 1.

Поиск вхождений образца в текст. Создаем новую строку образец + `#` + текст. Для этой строки считаем префикс функцию. Если какое-то значение функции равно длине образца, значит это вхождение.

Циклический сдвиг. Проверяем длины строк на равенство. Удваиваем первую строку и запускаем поиск вхождений второй строки в первую.

Подготовка к распараллеливанию. Разрезаем текст на k частей, так чтобы каждая следующая часть содержала длина образца — 1 предыдущей. Для каждой части запускаем поиск отдельно.

Сложность алгоритма

Сложности считаются при k = 1.

Сложность по времени — O(n + m), т. к. увеличение текущего значения образца на 1 может произойти максимум n+m раз, а уменьшение не может произойти больше раз, чем увеличение.

Сложность по памяти — O(n + m), т. к. программа хранит массив с префикс функцией строки длиной n + m.

Сложности для циклического сдвига и поиска вхождений одинаковые, т. к. при поиске циклического сдвига просто вызывается поиск вхождений.

Описание функций и структур данных

vector<int> prefixFunction(string s)

Считает префикс функцию для строки s и возвращает её в виде вектора.

vector<string> cutString(string& s, int k, int sampleLen)

Разрезает строку s на k частей для распараллеливания алгоритма.

set<int> findString(string& p, string& t)

Возвращает индексы вхождения образца p в текст t. Если вхождений нет возвращает -1.

int cycle(string& a, string& b)

Если строка b является циклическим сдвигом строки a, возвращает индекс начала циклического сдвига, иначе возвращает -1.

Тестирование

Тесты находятся в папке Tests.

Для тестирования был написан скрипт на python3, который запускает все тесты из папки вместе.

Поиск вхождений.

№	Ввод	Вывод
1	ab abab	0,2
2	def abcdefdefgdef	3,6,10
3	aaaa aaaaaaaaaaaaaa	0,1,2,3,4,5,6,7,8,9,10,11
4	de abcd	-1

Полный вывод программы для теста 2 (при k = 4).

```
def
abcdefdefgdef
strings for parallel algorithm:
abcde
defde
defgd
gdef

work with def#abcde
calculate pi[1]
curLength = 0
compare symbols 0 1
not equal
calculate pi[2]
curLength = 0
compare symbols 0 2
not equal
calculate pi[3]
curLength = 0
compare symbols 0 3
not equal
calculate pi[4]
curLength = 0
compare symbols 0 4
not equal
calculate pi[5]
curLength = 0
compare symbols 0 5
not equal
calculate pi[6]
curLength = 0
compare symbols 0 6
not equal
calculate pi[7]
curLength = 0
compare symbols 0 7
```

```
equal
calculate pi[8]
compare symbols 1 8
equal
compare symbols 1 8
equal
pi: 0 0 0 0 0 0 0 1 2

work with def#defde
calculate pi[1]
curLength = 0
compare symbols 0 1
not equal
calculate pi[2]
curLength = 0
compare symbols 0 2
not equal
calculate pi[3]
curLength = 0
compare symbols 0 3
not equal
calculate pi[4]
curLength = 0
compare symbols 0 4
equal
calculate pi[5]
compare symbols 1 5
equal
compare symbols 1 5
equal
calculate pi[6]
compare symbols 2 6
equal
compare symbols 2 6
equal
calculate pi[7]
compare symbols 3 7
```

```
not equal
curLength = 0
compare symbols 0 7
equal
calculate pi[8]
compare symbols 1 8
equal
compare symbols 1 8
equal
pi: 0 0 0 0 1 2 3 1 2

work with def#defgd
calculate pi[1]
curLength = 0
compare symbols 0 1
not equal
calculate pi[2]
curLength = 0
compare symbols 0 2
not equal
calculate pi[3]
curLength = 0
compare symbols 0 3
not equal
calculate pi[4]
curLength = 0
compare symbols 0 4
equal
calculate pi[5]
compare symbols 1 5
equal
compare symbols 1 5
equal
calculate pi[6]
compare symbols 2 6
equal
compare symbols 2 6
```

```
equal
calculate pi[7]
compare symbols 3 7
not equal
curLength = 0
compare symbols 0 7
not equal
calculate pi[8]
curLength = 0
compare symbols 0 8
equal
pi: 0 0 0 0 1 2 3 0 1

work with def#gdef
calculate pi[1]
curLength = 0
compare symbols 0 1
not equal
calculate pi[2]
curLength = 0
compare symbols 0 2
not equal
calculate pi[3]
curLength = 0
compare symbols 0 3
not equal
calculate pi[4]
curLength = 0
compare symbols 0 4
not equal
calculate pi[5]
curLength = 0
compare symbols 0 5
equal
calculate pi[6]
compare symbols 1 6
equal
```

```
calculate pi[6]
compare symbols 1 6
equal
compare symbols 1 6
equal
calculate pi[7]
compare symbols 2 7
equal
compare symbols 2 7
equal
pi: 0 0 0 0 0 1 2 3

3,6,10
```

Циклический сдвиг

№	Ввод	Вывод
1	abcd bcda	1
2	aaaaa aaaaa	0
3	abab baba	1
4	abc cba	-1

Выводы

В ходе лабораторной работы был изучен принцип работы алгоритма Кнута-Мориса-Прата и было решено две задачи, с пользованием этого алгоритма.

```
#include <iostream>
#include <string>
#include <vector>
#include <set>

using namespace std;

vector<int> prefixFunction(string s) {
    cout << "work with " << s << '\n';
    int n = s.length();
    vector<int> pi(n,0);
    for (int i = 1; i < n; i++) {
        cout << "calculate pi[" << i << "]\n";
        int curLength = pi[i-1];
         if (curLength > 0) cout << "compare symbols " << curLength << ' ' << i
<< '\n';
        while (curLength > 0 && s[i] != s[curLength]) {
            cout << "equal\n";
            curLength = pi[curLength - 1];
            if (curLength > 0) cout << "compare symbols " << curLength << ' ' <<
i << '\n';
        }
        if (curLength == 0) cout << "curLength = 0\n";
        else cout << "not equal\n";
        cout << "compare symbols " << curLength << ' ' << i << '\n';
        if (s[curLength] == s[i]) {
            cout << "equal\n";
            curLength++;
        }
        else cout << "not equal\n";
        pi[i] = curLength;
    }
    cout << "pi: ";
    for (auto i:pi) cout << i << ' ';
    cout << "\n\n";
    return pi;
}

vector<string> cutString(string& s, int k, int sampleLen) {
    vector<string> cutStrings;
    int strLen = s.length();
    sampleLen--;
```

```
    int oneStringLen = (strLen - sampleLen) / k + sampleLen;
    if (sampleLen >= oneStringLen - 1) {
        cout << "k is too big\nno cut\n";
        cutStrings.push_back(s);
        return cutStrings;
    }
    int plusOneCount = (strLen - sampleLen) % k;
    int ind = 0;
    for (int i = 0; i < plusOneCount; i++) {
        cutStrings.push_back(s.substr(ind, oneStringLen + 1));
        ind += oneStringLen + 1 - sampleLen;
    }
    for (int i = 0; i < k - plusOneCount; i++) {
        cutStrings.push_back(s.substr(ind, oneStringLen));
        ind += oneStringLen - sampleLen;
    }

    return cutStrings;
}

set<int> findString(string& p, string& t) {
    vector<string> cutStrings = cutString(t, 4, p.length());
    cout << "strings for parallel algorithm:\n";
    for (auto& i:cutStrings) cout << i << '\n';
    cout << '\n';
    set<int> indexes;
    int curIndex = 0;
    for (string& partOfT:cutStrings) {
        string newString = p + '#' + partOfT;

        vector<int> pi = prefixFunction(newString);
        int pLength = p.length(), tLength = t.length();

        for (int i = pLength * 2; i < pLength + tLength + 1; i++)
            if (pi[i] == pLength)
                indexes.insert(i - 2 * pLength + curIndex);
        curIndex += partOfT.length() - p.length() + 1;
    }
    return indexes;
}

int cycle(string& a, string& b) {
```

```
    if (a.length() != b.length()) return -1;
    a += a;
    set<int> indexes = findString(b, a);
    if (indexes.empty()) return -1;
    return *indexes.begin();
}

int main() {
    string a, b;
    cin >> a >> b;
    set<int> answer = findString(a,b);
    if (answer.empty()) cout << -1;
    else {
        auto answerEnd = answer.end();
        answerEnd--;
        for (auto i = answer.begin(); i != answerEnd; i++) {
            cout << *i << ',';
        }
        cout << *answerEnd;
    }

    //int answer = cycle(a,b);
    //cout << answer;
}
```