Федеральное государственное бюджетное образовательное учреждение
высшего образования «Сибирский государственный университет телекоммуникаций
и информатики»

Факультет ИВТ

Кафедра вычислительных систем

Курсовая работа

на тему «Полнотекстовый поиск по шаблону»
Вариант 3.3 «Алгоритм Кнута-Морриса-Пратта»

Выполнил:
студент гр. ИВ-221
Анциферов Я.Г.

Проверил:
Старший преподаватель Кафедры ВС
Фульман В.О.

Новосибирск, 2023

Тема курсовой работы: Алгоритм Кнута-Морриса-Пратта
Тема выбранного раздела: Полнотекстовый поиск по шаблону

Задание на курсовую работу
Реализовать программу **kmpmatcher** (Knuth–Morris–Pratt string MATCHER) полнотекстового поиска по шаблону. Шаблон и имя файла (директории), в которой осуществляется поиск, передаются через аргументы командной строки в следующем порядке:

$ **kmpmatcher** "g*.le" ~/mydir	#Анализ всех файлов, расположенных в ~/mydir.
$ **kmpmatcher -r** "g*.le" ~/mydir	#Рекурсивный поиск во всех директориях, расположенных ниже ~/mydir.

Критерии оценки
- **Оценка «отлично»:** разработанная программа обеспечивает поиск текста по шаблону рекурсивно в заданной директории. Под рекурсивным поиском понимается анализ всех текстовых файлов в текущей директории, а также во всех вложенных директориях.
- **Оценка «хорошо»:** разработанная программа не предусматривает поиск по шаблону ИЛИ не способна выполнять рекурсивный поиск в дереве каталогов (поиск только в одном файле).
- **Оценка «удовлетворительно»:** реализован только алгоритм Кнута-Морриса-Пратта.

Указание к выполнению задания
Алгоритм Кнута-Морриса-Пратта (КМП) основан на применении префикс-функции π_P, подробно описанной в общей информации к данному разделу. В листинге 5 приведен псевдокод алгоритма КМП.

Листинг 5. Псевдокод алгоритма КМП

```
KMP_MATCHER(T, P)
        m ← len(P)
        n ← len(T)
        πP ← COMPUTE_PREFIX_F(P)
        q ← 0
        for i ← 1 to n do
                while q > 0 и P[q + 1] ≠ T[i] do
                        q ← πP[q]
                if P[q + 1] = T[i] then
                        q ← q+1
                if q = m then
                        print "Образец обнаружен при сдвиге" i – m
                q ← πP[q]
```

Анализ задачи

1. Для реализации алгоритма Кнута-Морриса-Пратта необходимо реализовать префикс-функцию, которая является основой этого алгоритма. Ее предназначение заключается в вычислении наибольшей длины собственного суффикса строки (не совпадает со всей строкой), который совпадает с префиксом строки для каждого символа в ней.

   Пример:

   Дана строка “abaabab”. После прохода префикс-функции результат будет таким: 0 0 1 1 2 3 2
   Принцип работы:

   Для i = 0
   “**a**baabab”
   Для нулевого элемента результат всегда будет равен 0, так как он не является собственным суффиксом.

   Для i = 1
   “**ab**aabab”
   Результат 0, так как “a” != “b”

   Для i = 2
   “**aba**abab”
   Результат 1, так как “a” == “a”, но “ab” != “ba”

   Для i = 3
   “**abaa**bab”
   Результат 1, так как “a” == “a”, но “ab” != “aa” и “aba” != “baa”

   Для i = 4
   “**abaab**ab”
   Результат 2, так как “ab” == ab”, но “aba” != “aab” и “abaa” != “baab”

   Для i = 5
   “**abaaba**b”
   Результат 3, так как “aba” == “aba”, но “abaa” != “aaba” и т.д.

   Для i = 6
   “**abaabab**”
   Результат 2, так как “ab” == “ab”, далее суффиксы с префиксами не совпадают.

2. После реализации префикс-функции нужно составить алгоритм Кнута-Морриса-Пратта. Для его работы необходимы: шаблон и строка, где будет происходить поиск этого шаблона, также нужно создать массив для хранения данных от префикс-функции. Сам алгоритм выводит на экран индекс вхождения в строке. Преимущество этого алгоритма перед обычным поиском состоит в том, что он перемещается по строке намного быстрее (обычный поиск начнет работу с прошлой позиции плюс один, в то время как алгоритм может переместиться в случае неудачи максимум на длину шаблона и минимум на 1 от предыдущей позиции).

   Пример:

   Дана строка S = “abbcabbcbba”, шаблон P = “bbcbb”, ожидаемый результат 5.

Принцип работы:

Сначала находим префикс-функцию от P = 0 1 0 1 2
Затем начинаем поиск шаблона в строке

abbcabbcbba
bbcbb
Символы не совпадают, ищем дальше. Индекс = 0

abbcabbcbba
 bbcbb
Символы совпадают, идем дальше по шаблону и строке. Индекс = 1

abbcabbcbba
 bbcbb
Символы совпадают, идем дальше по шаблону и строке. Индекс = 2
abbcabbcbba
 bbcbb
Символы совпадают, идем дальше по шаблону и строке. Индекс = 3

abbcabbcbba
 bbcbb
Символы не совпадают, поэтому мы берем значение индекса и уменьшаем его на один, так как в массиве префикс-функции в элементе [Индекс – 1] хранится 0, следовательно начинаем поиск заново с индекса 4.

abbcabbcbba
 bbcbb
Символы не совпадают, перемещаемся вперед на единицу. Индекс = 4

abbcabbcbba
 bbcbb
Символы совпадают, идем дальше по шаблону и строке. Индекс = 5

abbcabbcbba
 bbcbb
Символы совпадают, идем дальше по шаблону и строке. Индекс = 6

abbcabbcbba
 bbcbb
Символы совпадают, идем дальше по шаблону и строке. Индекс = 7

abbcabbcbba
 bbcbb
Символы совпадают, идем дальше по шаблону и строке. Индекс = 8

abbcabbcbba
 bbcbb
Шаблон найден, возвращаем индекс вхождения, он равен
9 (текущая позиция в строке) – 5 (длина шаблона) + 1 = 5

Тестовые данные
Необходимо провести тестирование приложения. Предполагается, что при корректных данных будет проведен поиск шаблона в тексте и при некорректных выведена ошибка и информация о ней.

Тест на вхождение подстроки “ABC” в строку “ABCADBABCDACBABCAA”:

```
Entry test for 'ABCADBABCDACBABCAA':
'ABC' in position 0
'ABC' in position 6
'ABC' in position 13
No more matches found
```

Тест на вхождение подстроки “WASHINGTON” в строку “LONDONISTHECAPITALOFGREATBRITAIN”:

```
Entry test for 'LONDONISTHECAPITALOFGREATBRITAIN':
No more matches found
```

Тест на вхождение подстроки “MOSCOW” в строку “MOSCOWISTHETHIRDROME”:

```
Entry test for 'MOSCOWISTHETHIRDROME':
'MOSCOW' in position 0
No more matches found
```

Тест на вхождение подстроки “NUMBERS” в строку “ 323353243411422131231”

```
Entry test for '323353243411422131231':
No more matches found
```

Листинг программы

main.c

```c
#include <stdio.h>

void algorithm_KMP (char* massive, char* pod_massive, int* array, int size_m, int size_p_m) {
        printf("Entry test for '%s':\n", massive);

        int i = 0;
        int j = 0;

        while ((i < size_m - 1) || (j < size_p_m - 1)) {
                if (massive[i] == pod_massive[j]) {
                        i++;
                        j++;
                }
                else {
                        if (j == 0)
                                i++;
                        else {
                                j--;
                                i -= array[j];
                                j = 0;
                        }
                }
                if (j == size_p_m - 1)
                        printf("'%s' in position %d\n", pod_massive, i - (size_p_m - 1));
                if (i == size_m - 1) {
                        printf("No more matches found\n");
                        break;
                }
        }

}

void prefix_find (char* pod_massive, int* array, int size) {
        *array = 0;
        for (int i = 1; i < size; i++) {
                int j = array[i-1];
                while ((j > 0) && (pod_massive[i] != pod_massive[j]))
                        j = array[j-1];
                if (pod_massive[i] == pod_massive[j])
                        j++;
                array[i] = j;
        }
}

int main () {
        char massive[] = "323353243411422131231";
        char pod_massive[] = "NUMBERS";
        int array[sizeof(pod_massive)-1];

        prefix_find(pod_massive, array, sizeof(pod_massive)-1);
        algorithm_KMP(massive, pod_massive, array, sizeof(massive)-1, sizeof(pod_massive)-1);

        return 0;
}
```