Министерство цифрового развития, связи и массовых коммуникаций

Ордена Трудового Красного Знамени федеральное государственное бюджетное

образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и информационных технологий

Отчёт по лабораторной работе № 3

Методы поиска подстроки в строке

по дисциплине «Сиаод»

Выполнил: студент группы

БВТ1902

Сорокин Никита Андреевич

Москва

2021

Цель работы

Реализовать методы поиска подстроки в строке. Добавить возможность ввода строки и подстроки с клавиатуры. Предусмотреть возможность существования пробела. Реализовать возможность выбора опции чувствительности или нечувствительности к регистру. Оценить время работы каждого алгоритма поиска и сравнить его со временем работы стандартной функции поиска, используемой в выбранном языке программирования. Алгоритмы: 1.Кнута-Морриса-Пратта 2.Упрощенный Бойера-Мура.

Написать программу, определяющую, является ли данное расположение решением для игры “пятнашки”, то есть можно ли из него за конечное число шагов перейти к правильному. Если это возможно, то необходимо найти хотя бы одно решение - последовательность движений, после которой числа будут расположены в правильном порядке.

Код программы

```
def suf_pref_inxd(stroka_poisk):
    global p
    p = [0] * len(stroka_poisk)
    j = 0
    i = 1

    while i < len(stroka_poisk):
        if stroka_poisk[j] == stroka_poisk[i]:
            p[i] = j + 1
            i += 1
            j += 1
        else:
            if j == 0:
                p[i] = 0;
                i += 1
            else:
                j = p[j - 1]


def alg_kmp(stroka_poisk, p, stroka):
    m = len(stroka_poisk)
    n = len(stroka)

    i = 0
    j = 0
    while i < n:
        if stroka[i] == stroka_poisk[j]:
            i += 1
            j += 1
            if j == m:
                print("Подстрока найдена: ")
                print("Индекс вхождения(начало): ", i - m)
                print("Индекс вхождения(конец): ", i)
                break
        else:
            if j > 0:
                j = p[j - 1]
            else:
                i += 1

    if i == n:
        print("образ не найден")


stroka_poisk = "ddddd"
```

```
stroka = "лидддддлилилась"
suf_pref_inxd(stroka_poisk)
alg_kmp(stroka_poisk, p, stroka)
def table_smesh(t):
    global d
    global M
    S = set()
    M = len(t)
    d = {}

    for i in range(M - 2, -1, -1):
        if t[i] not in S
            d[t[i]] = M - i - 1
            S.add(t[i])

    if t[M - 1] not in S:
        d[t[M - 1]] = M

    d['*'] = M


def poisk_v_stroke(a,t):
    N = len(a)

    if N >= M:
        i = M - 1
        while (i < N):
            k = 0
            j = 0
            for j in range(M - 1, -1, -1):
                if a[i - k] != t[j]:
                    if j == M - 1:
                        off = d[a[i]] if d.get(a[i], False) else d['*']
                    else:
                        off = d[t[j]]

                    i += off
                    break

                k += 1

            if j == 0:
                print("Подстрока найдена: ")
                print("Индекс вхождния(начало): ", i - k + 1)
                print("Индекс вхождния(конец): ",i)
                break
        else:
            print("образ не найден")
    else:
        print("образ не найден")
```

```
stroka_poisk = "данные"
stroka = "метеоданные"
table_smesh(stroka_poisk)
poisk_v_stroke(stroka, stroka_poisk)


import copy

def perestanovka(board):
    global l, r, u, d
    l = copy.deepcopy(board)
    r = copy.deepcopy(board)
    u = copy.deepcopy(board)
    d = copy.deepcopy(board)

    for i in range(len(board)):
        for j in range(len(board)):
            if board[i][j] == 0:
                k1 = i
                k2 = j
    try:
        if k2 != 0:
            l[k1][k2 - 1], l[k1][k2] = l[k1][k2], l[k1][k2 - 1]
        else:
            l = 0
    except IndexError:
        l = 0

    try:
        r[k1][k2 + 1], r[k1][k2] = r[k1][k2], r[k1][k2 + 1]
    except IndexError:
        r = 0

    try:
        if k1 != 0:
            u[k1 - 1][k2], u[k1][k2] = u[k1][k2], u[k1 - 1][k2]
        else:
            u = 0
    except IndexError:
        u = 0

    try:
        d[k1 + 1][k2], d[k1][k2] = d[k1][k2], d[k1 + 1][k2]
    except IndexError:
        d = 0


def calculateManhattan(l, r, u, d,l_number = 0):
```

```
f = 999
global board,k,m
if l != 0 and k != 1:
    distance = 0
    for i in range(4):
        for j in range(4):
            if l[i][j] != 0:
                x, y = divmod(l[i][j] - 1, 4)
                distance += abs(x - i) + abs(y - j)
    if distance < f:
        f = distance
        k = 0
        m = -1
        f_name = 'l'
        board = list(l)

if r != 0 and k != 0:
    distance = 0
    for i in range(4):
        for j in range(4):
            if r[i][j] != 0:
                x, y = divmod(r[i][j] - 1, 4)
                distance += abs(x - i) + abs(y - j)
    if distance < f:
        f = distance
        k = 1
        m = -1
        f_name = 'r'
        board = list(r)

if u != 0 and m != 3:
    distance = 0
    for i in range(4):
        for j in range(4):
            if u[i][j] != 0:
                x, y = divmod(u[i][j] - 1, 4)
                distance += abs(x - i) + abs(y - j)
    if distance < f:
        f = distance
        m = 2
        k = -1
        f_name = 'u'
        board = list(u)

if d != 0 and m != 2:
    distance = 0
    for i in range(4):
        for j in range(4):
            if d[i][j] != 0:
```

```
                x, y = divmod(d[i][j] - 1, 4)
                distance += abs(x - i) + abs(y - j)
        if distance < f:
            f = distance
            m = 3
            k = -1
            f_name = 'd'
            board = list(d)

    print("Лучший вес:", f, " Меняем:", f_name)


board = [[1, 2, 3, 4],[5, 0, 7, 8],[13, 6, 11, 12],[10, 9, 14, 15]]

count = 0
m = -1
k = -1
go = 0
print("Начальная позиция")
while go == 0:
    perestanovka(board)
    for i in range(4):
        print(board[i])
    calculateManhattan(l,r,u,d)
    count += 1
    if board == [[1, 2, 3, 4],[5, 6, 7, 8],[9, 10, 11, 12],[13, 14, 15, 0]]:
        go += 1
        for i in range(4):
            print(board[i])
        count += 1

print("Шагов: ", count)
```

Скриншоты работы программы

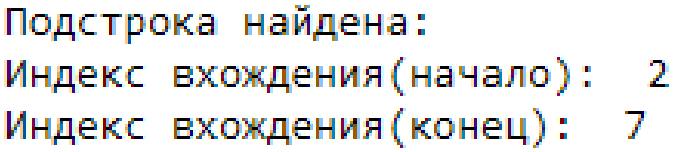

Рис. 1 – Результат алгоритма КМП

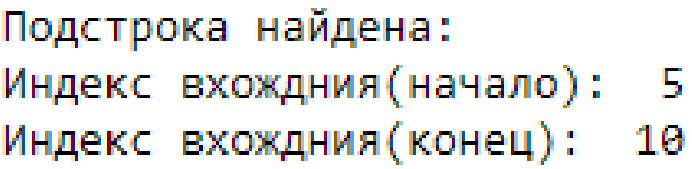

Рис. 2 – Результат алгоритма Бойера-Мура

```
начальная позиция
[1, 2, 3, 4]
[5, 0, 7, 8]
[13, 6, 11, 12]
[10, 9, 14, 15]
Лучший вес: 7  Меняем: d
[1, 2, 3, 4]
[5, 6, 7, 8]
[13, 0, 11, 12]
[10, 9, 14, 15]
Лучший вес: 6  Меняем: d
[1, 2, 3, 4]
[5, 6, 7, 8]
[13, 9, 11, 12]
[10, 0, 14, 15]
Лучший вес: 5  Меняем: l
[1, 2, 3, 4]
[5, 6, 7, 8]
[13, 9, 11, 12]
[0, 10, 14, 15]
Лучший вес: 4  Меняем: u
[1, 2, 3, 4]
[5, 6, 7, 8]
[0, 9, 11, 12]
[13, 10, 14, 15]
Лучший вес: 3  Меняем: r
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 0, 11, 12]
[13, 10, 14, 15]
Лучший вес: 2  Меняем: d
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 0, 14, 15]
Лучший вес: 3  Меняем: l
```

```
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[0, 13, 14, 15]
Лучший вес: 4  Меняем: u
[1, 2, 3, 4]
[5, 6, 7, 8]
[0, 10, 11, 12]
[9, 13, 14, 15]
Лучший вес: 3  Меняем: d
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[0, 13, 14, 15]
Лучший вес: 2  Меняем: r
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 0, 14, 15]
Лучший вес: 1  Меняем: r
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 14, 0, 15]
Лучший вес: 0  Меняем: r
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 14, 15, 0]
Шагов:  13
```

Рис. 3 – Результат алгоритма поиска решения “пятнашек”

Вывод

В ходе выполнения лабораторной работы, я научился реализовывать различные алгоритмы поиска подстроки в строке. В ходе тестов было понятно, что скорость зависит от количества исходный элементов, но самыми быстрым является алгоритм Бойера-Мура.