**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. 9304 |  | Ковалёв П.Д. |
| Преподаватель |  | Шевская Н.В. |

Санкт-Петербург

2021

**Цель работы.**

Изучить алгоритм Кнута-Морриса-Пратта поиска подстроки в строке. Реализовать программу на C++, которая для заданного текста находит все вхождения заданной строки.

**Задание.**

Реализуйте алгоритм КМП и с его помощью для заданных шаблона *P* () и текста *Т*() найдите все вхождения *P* в *T*.

**Входные данные:**  
 Первая строка — *Р*

Вторая строка — *T*

**Выходные данные:**  
 Индексы начал вхождений *P* в *T*, разделенных запятой если *P* не входит в *T*, то вывести -1.

**Выполнение работы.**

Сначала была написана префикс функция, которая возвращает вектор, элементы которого обозначают длину максимального префикса строки *s[0..i]*, где *i*  - номер элемента массива. Этот вектор используется в функции *kmp()*, которая находит индексы вхождений подстроки в строку. Для этого, нужно всего лишь найти элементы вектора такие, что они равны длине подстроки. В таком случае, получим начало вхождения подстроки *P* в текст *T*.

Алгоритм Кнута-Морриса-Пратта позволяет находить префикс функцию для заданной строки за линейное время и осуществлять поиск подстроки в строке. Алгоритм работает следующим образом: строится строка вида *P#T*, где *P* — образец, который ищется в тексте *T*. После вектор значений префикс функции инициализируется нулями, а его размер равен длине строки. После запускается цикл от 1 до конца длины строки. В цикле запоминаем текущую длину префикса (изначально она равна 0), далее в цикле *while()* мы уеньшаем длину префикса, пока не сможем его продолжить. Если продолжить получается, и *i*-ый (переменная внешнего цикла *for*) символ строки равен *j*-му символу строки, то мы записываем в вектор результатов по индексу *j* значение *j+1*.

После того как мы построили префикс функцию, ищем элементы вектора, равные длине подстроки. Если мы нашли их, то их индекс в векторе — индекс начала подстроки.

**Оценка сложности алгоритма.**

**Тестирование.**

Были написаны юнит-тесты к каждой функции программы, за исключением *main()*. В одном случае тесты проверяли, сможет ли алгоритм корректно найти подстроку в строке, а в другом случае — сможет ли определить, является ли одна строка циклическим сдвигом другой.

Рисунок 1 – Результаты юнит-тестирования

**Выводы.**

Изучили алгоритм Кнута-Морриса-Пратта поиска подстроки в строке. Реализовали программу на C++, которая для заданного текста находит все вхождения заданной строки.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

lr3.py:

import queue

def maximum\_flow(graph, startVertex, endVertex):

f = 0

while (True):

dway = bfs(graph, startVertex, endVertex)

if (dway is None):

break

way = construct\_way(dway, startVertex, endVertex)

cmin = find\_minimal\_capacity(dway, startVertex, endVertex)

f += cmin

for i in range(0, len(way) - 1):

for m in graph[way[i]]:

if m[0] is way[i + 1]:

if way[i+1] != endVertex:

for a in graph[way[i + 1]]:

if a[0] is way[i]:

a[1] += cmin

a[2] -= cmin

if a[3]:

a[3] = True

else:

a[3] = False

break

else:

graph.setdefault(way[i + 1], []).append([way[i], cmin, 0, False])

m[1] -= cmin

m[2] += cmin

res = [f]

for i in graph:

for m in graph[i]:

if(m[2] < 0):

res.append([i, m[0], 0, m[3]])

else:

res.append([i, m[0], m[2], m[3]])

return res

def find\_minimal\_capacity(dway, startVertex, endVertex):

tmpKey = endVertex

cmin = dway[tmpKey][1]

while dway[tmpKey][0] != startVertex:

tmpVal = dway[tmpKey][0]

if (cmin > dway[tmpVal][1]):

cmin = dway[tmpVal][1]

tmpKey = tmpVal

return cmin

def bfs(graph, startV, endV):

q = queue.Queue()

q.put(startV)

paths = {startV: []}

while (not q.empty()):

vertex = q.get()

if vertex in graph:

for v in graph[vertex]:

if (v[1] > 0 and v[0] not in paths):

paths[v[0]] = [vertex, v[1]]

if v[0] == endV:

return paths

q.put(v[0])

return None

def construct\_way(dict\_way, start, end):

way = ""

tmpKey = end

way += tmpKey

while dict\_way[tmpKey][0] != start:

tmpVal = dict\_way[tmpKey][0]

way += tmpVal

tmpKey = tmpVal

way += start

return way[::-1]

def sorter(diction):

key\_list = list(diction.keys())

key\_list.sort()

items\_list = []

for i in range(len(key\_list)):

items\_list.append(diction[key\_list[i]])

for i in items\_list:

i.sort(key=lambda x: x[0])

d = {}

for i in key\_list:

r = []

for m in items\_list:

if m == diction[i]:

r.append(m)

d[i] = r[0]

return d

def main():

n = int(input())

startVertex = input()

endVertex = input()

d = {}

m = []

while (n > 0):

s = input().split()

m.append(s)

d.setdefault(s[0], []).append([s[1], int(s[2]), 0, True])

n -= 1

d = sorter(d)

result = maximum\_flow(d, startVertex, endVertex)

flow = result[0]

l = []

for i in result:

if i != flow:

l.append(i)

l.sort(key=lambda x: x[0])

print(flow)

for i in range(0, len(l)):

if l[i][3]:

s = ""

for m in range(len(l[i])-1):

s += str(l[i][m]) + " "

num = len(s)

print(s[0:num - 1:1])

if \_\_name\_\_ == '\_\_main\_\_':

main()