**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: **Кнут-Моррис-Пратт**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9303 |  | Павлов Д.Р. |
| Преподаватель |  | Шевская Н.В. |

Санкт-Петербург

2021

## Цель работы.

Изучить алгоритм Кнута-Морриса-Пратта поиска подстроки в строке.

## Задание.

1. Реализуйте алгоритм КМП и с его помощью для заданных шаблона *P* (∣P∣≤15000∣*P*∣≤15000) и текста *T* (∣T∣≤5000000∣*T*∣≤5000000) найдите все вхождения *P* в *T*.  
   Вход:

Первая строка - *P*   
Вторая строка - *T*  
Выход:  
индексы начал вхождений *P*  в  *T*, разделенных запятой, если *P* не входит в *T*, то вывести −1

2) Заданы две строки *A*(|*A*|*≤*500000) и *B*(|*B*|*≤* 500000).  
 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 является *A* циклическим сдвигом *B*, индекс начала строки *B* в *A*,

иначе вывести -1. Если возможно несколько сдвигов вывести первый индекс.

## Выполнение работы.

lps\_func(string txt, vector<int>&lps) – Префикс.

KMP(string pattern, string text, list<size\_t>&result) – Алгоритм Кнута-Морриса-Пратта.

kmp\_cycle(std::string& pat, std::string& text) – Алгоритм, который находит циклический сдвиг.

**Описание алгоритма.**

Для нахождения всех вхождений шаблона P в текст T, надо посчитать значения префикс-функции для строки P#T. Если функция содержит значения, равные длине P, то P входит в T.

Для определения, является ли строка A циклическим сдвигом строки B, надо посчитать значение префикс-функции для строки B#AA.

**Анализ алгоритма.**

Пусть m — длина строки, для которой вычисляется префикс-функция, n — длина текста. Тогда сложность алгоритма будет равняться *O*(*n*+*m*) .

Разработанный программный код см. в приложении А.

Результаты тестирования см. в приложении Б.

## Выводы.

Был успешно реализован алгоритм Форда-Фалкерсона для нахождения максимального потока в графе.

# Приложение А Исходный код программы

Название файла: main.py

from sys import argv, stdin

from icecream import ic

class Edge(object):

def \_\_init\_\_(self, u, v, w):

self.source = u

self.sink = v

self.capacity = w

def \_\_repr\_\_(self):

return "%s %s:%s" % (self.source, self.sink, self.capacity)

class FlowNetwork(object):

def \_\_init\_\_(self):

self.founded = []

self.adj = {}

self.flow = {}

def add\_vertex(self, vertex):

self.adj[vertex] = []

def get\_edges(self, v):

ic(v)

return self.adj[v]

def add\_edge(self, u, v, w=0):

if u == v:

raise ValueError("u == v")

edge = Edge(u, v, w)

redge = Edge(v, u, 0)

edge.redge = redge

redge.redge = edge

self.adj[u].append(edge)

self.adj[v].append(redge)

self.flow[edge] = 0

self.flow[redge] = 0

def find\_path(self, source, sink, path):

if source == sink:

return path

for edge in self.get\_edges(source):

ic(edge)

residual = edge.capacity - self.flow[edge]

if residual > 0 and edge not in path:

result = self.find\_path(edge.sink, sink, path + [edge])

if result is not None:

return result

def max\_flow(self, source, sink):

path = self.find\_path(source, sink, [])

while path is not None:

residuals = [edge.capacity - self.flow[edge] for edge in path]

flow = min(residuals)

for edge in path:

self.flow[edge] += flow

self.flow[edge.redge] -= flow

path = self.find\_path(source, sink, [])

return sum(self.flow[edge] for edge in self.get\_edges(source))

def get\_result(self):

new\_dict = {}

for i in self.flow:

if self.flow[i] >= 0:

name = i.\_\_repr\_\_().split(":")[0]

value = self.flow[i]

if (name in new\_dict and value > new\_dict[name]) or (name not in new\_dict) and name in self.founded:

new\_dict[name] = value

res = []

for i in new\_dict.keys():

res.append(f'{i} {new\_dict[i]}')

return sorted(res)

def main():

g = FlowNetwork()

vers = []

N = int(input())

v0 = input()

vn = input()

strlist = []

[strlist.append(line) for line in stdin]

s = "".join(strlist).splitlines()

for i in s:

vers.append(i[0])

vers.append(i[2])

g.founded.append("".join(i[0] + ' ' + i[2]))

vers = list(set(vers))

[g.add\_vertex(v) for v in vers]

for i in s:

g.add\_edge(i.split(' ')[0], i.split(' ')[1], int(i.split(' ')[2]))

print(g.max\_flow(v0, vn))

res = g.get\_result()

for i in res:

print(i)

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

main()

# Приложение Б Тестирование

Таблица 1 - Примеры тестовых случаев.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Входные данные | Выходные данные | Комментарии |
|  | 7  a  f  a b 7  a c 6  b d 6  c f 9  d e 3  d f 4  e c 2 | 12  a b 6  a c 6  b d 6  c f 8  d e 2  d f 4  e c 2 | Программа работает корректно |
|  | 8  1  4  1 2 7  1 3 6  2 4 8  2 5 1  3 5 2  3 6 4  6 5 7  5 4 6 | 13  1 2 7  1 3 6  2 4 7  2 5 0  3 5 2  3 6 4  5 4 6  6 5 4 | Программа работает корректно |