

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет Информатика и вычислительная техника

Кафедра Кибербезопасность информационных систем

**ЛАБОРАТОРНАЯ РАБОТА № 1**

на тему «Алгоритмы нахождения кратчайших путей в графе»

Выполнил обучающийся гр. ВКБ31

Гришин Данил Васильевич

Проверил

Доцент, Савельев Василий Александрович

Ростов-на-Дону

2022Лабораторная работа №1

«Алгоритмы нахождения кратчайших путей в графе»

Реализовать нахождение кратчайшего пути в графе между двумя заданными вершинами.

Внутренняя структура графа не задается, но рекомендуется ориентироваться на решения

эффективные и по памяти, и по времение реализации алгоритма. Также в задание входит

создание вспомогательных библиотек, обеспечивающих ввод/вывод графа.

Задание 1

Создать библиотеку обеспечивающую ввод и вывод графа в текстовой форме. Граф в

текстовом файле задается списком ребер — одно ребро на строке, ребро задается номером

начальной вершины, номером конечной вершины и весом ребра. Все три значения целые,

разделяются пробельными символами.

Пример:

0 1 7

0 2 9

1 2 10

1 3 15

2 3 11

Написать тесты и отладить эту библиотеку.

Измерить скорость ввода-вывода

Задание 2

Создать библиотеку, обеспечивающую ввод и вывод графа в бинарнойй форме. Использовать

для сохранения значений 32-разрабные целые поля в сетевом порядке байтов. Написать

тесты и отладить библиотеку. Измерить скорость ввода/вывода. Оценить ускорение

ввода/вывода при переходе к бинарному формату.

Задание 3.

Написать программу, способную прочесть граф из указанного файла (как в текстовой, так и

бинарной форме, выбирается ключом коммандной строки), получающую также в командной

строке номера начальной и конечной вершин. Выводится кратчайший путь между этими

вершинами. Если пути нет, выводится одна строка с номерами начальной и конечной вершин

и длиной пути -1. Если конечная вершина не указана, выводится длина кратчайшего пути из

начальной вершины во все остальные вершины.

Написать тесты и отладить программу

Код:

import numpy as np  
import math  
import networkx as nx  
import matplotlib.pyplot as plt  
  
  
class Graph(object):  
 def draw(self):  
 elist = list()  
 for i in range(self.n):  
 for j in range(self.n):  
 if self.adjacency[i, j] == 1:  
 elist.append((i, j, int(self.weights[i, j])))  
  
 vertexLabels = dict()  
  
 for i in range(self.n):  
 vertexLabels[i] = str(i)  
  
  
 G = nx.Graph()  
 G.add\_nodes\_from(np.arange(self.n))  
 pos = nx.spring\_layout(G)  
 G.add\_weighted\_edges\_from(elist)  
 labels\_dictionary = dict()  
 for i in range(len(self.weights)):  
 for j in range(len(self.weights)):  
 if(self.weights[i, j] != float("Inf")):  
 labels\_dictionary[i, j] = int(self.weights[i, j])  
 nx.draw\_networkx\_edge\_labels(G,pos, edge\_labels=labels\_dictionary)  
  
  
  
  
 nx.draw\_circular(G, labels = vertexLabels)  
 #print(nx.cycle\_basis(G, 0))  
 #nx.draw(G, labels=labels\_dictionary)  
  
  
 plt.show()  
  
 def \_\_init\_\_(self, a, b, c, d):  
  
 def generate\_peaks():  
 for i in range(self.n):  
 yield i  
  
 self.\_\_a = a  
 self.\_\_b = b  
 self.\_\_c = c  
 self.\_\_d = d  
  
 if a % 5 <= 1:  
 self.n = 10  
 elif a % 5 <= 3:  
 self.n = 11  
 else:  
 self.n = 12  
  
 self.peaks = np.array(list(generate\_peaks()))  
 self.adjacency = np.zeros((self.n, self.n), dtype=int)  
 self.weights = np.zeros((self.n, self.n))  
  
 h = 1  
  
 for i in range(self.n):  
 for j in range(self.n):  
 calculated\_value = math.sin((i \* j \* c + a) / d) + 1  
  
  
 if 1 <= calculated\_value <= 2:  
 self.adjacency[i, j] = 1  
 self.weights[i, j] = calculated\_value \* 10  
  
 else:  
 self.weights[i, j] = float("Inf")  
  
 if (h % 2 == 0 and h % 3 == 0 and h % 4 == 0):  
 self.weights[i, j] = -100  
  
 if(i == j):  
 self.weights[i, j] = 0  
 self.adjacency[i, j] = 0  
 h = h + 1  
  
 self.ribs = []  
  
 for i in range(self.n):  
 self.ribs.append([])  
 for j in range(self.n):  
 if self.adjacency[i, j] == 1:  
 self.ribs[i].append(j)  
  
 #print(self.ribs)  
  
  
  
 #print(self.weights)  
 #for i in range(self.n - 1):  
 # self.ribs[2][i] = 0  
 # self.ribs[i][2] = 0  
  
 #for i in range(self.n):  
 # self.ribs[2][i] = 0  
 # self.ribs[i][2] = 0  
  
  
  
  
 def fsc(self):  
  
 def Save(i, j, nZ, masQ, ftr):  
 z = i  
 while z != j:  
 if z in masQ[nZ]:  
 break  
 masQ[nZ].append(z)  
 z = ftr[z]  
 masQ[nZ].append(j)  
 masQ[nZ].append(i)  
  
  
 def Cicle(i, k, num, matrix\_size, ftr, nz):  
 k = k + 1  
 num[i] = k  
 for j in self.ribs[i]:  
 if num[j] == 0:  
 ftr[j] = i  
 Cicle(j, k, num, matrix\_size, ftr, nz)  
 elif ftr[i] != j:  
 nz = nz + 1  
 Save(i, j, nz, masQ, ftr)  
  
  
 n = 0  
 m = 0  
  
 num = []  
 ftr = []  
  
 matrix\_size = len(self.ribs)  
  
 for i in range(matrix\_size):  
 num.append(0)  
 ftr.append(0)  
 n = n + 1  
 m = m + len(self.ribs[i])  
  
 m = int(m / 2)  
  
 masQ = []  
 for i in range(m - n + 1):  
 masQ.append([])  
  
  
 k = 0  
 nZ = 0  
  
 Cicle(0, k, num, matrix\_size, ftr, nZ)  
  
  
  
  
  
 print("Р¤СѓРЅРґР°РјРµРЅС‚Р°Р»СЊРЅС‹Рµ С†РёРєР»С‹ РіСЂР°С„Р°: ")  
 ind = 0  
 k = 0  
 while k != len(masQ):  
 if(len(masQ[k]) == 0):  
 k = k + 1  
 continue  
 print("Р¦РёРєР» " + str(ind + 1) + " : ")  
 print(masQ[k])  
 print()  
 ind = ind + 1  
 k = k + 1  
  
 def BellmanFord(self, src):  
  
 dist = [float("Inf")] \* len(self.peaks)  
 predecessor = [src] \* len(self.peaks)  
  
 dist[src] = 0  
  
 for \_ in range(len(self.peaks) - 1):  
  
 for i in range(len(self.weights)):  
 for j in range(len(self.weights)):  
 if self.weights[i][j] != float("Inf") and dist[i] + self.weights[i][j] < dist[j]:  
 predecessor[j] = i  
 dist[j] = dist[i] + self.weights[i][j]  
  
 neg = []  
  
 for i in range(len(self.weights)):  
 for j in range(len(self.weights)):  
 if dist[i] != float("Inf") and dist[i] + self.weights[i][j] < dist[j] :  
 neg.append(i)  
 print("Р’ РіСЂР°С„Рµ СЃСѓС‰РµСЃС‚РІСѓСЋС‚ С†РёРєР»С‹ РѕС‚СЂРёС†Р°С‚РµР»СЊРЅРѕРіРѕ РІРµСЃР°. Р’РµСЃ " + str(i) + " РЅРµРІРµСЂРµРЅ")  
  
 globalPath = []  
  
 for i in range(len(self.weights)):  
 path = []  
 current = i  
 while current != src:  
 if current in neg:  
 path = [-float("inf")]  
 break  
 path.append(current)  
 current = predecessor[current]  
  
 globalPath.append(path[::-1])  
  
 print(globalPath)  
  
  
  
 def prim(self):  
  
 def isValidEdge(u, v, inMST):  
 if u == v:  
 return False  
 if inMST[u] == False and inMST[v] == False:  
 return False  
 elif inMST[u] == True and inMST[v] == True:  
 return False  
 return True  
  
 inMST = [False] \* len(self.weights)  
  
 inMST[0] = True  
  
  
 edge\_count = 0  
 mincost = 0  
 while edge\_count < len(self.weights) - 1:  
  
 minn = float("inf")  
 a = -1  
 b = -1  
 for i in range(len(self.weights)):  
 for j in range(len(self.weights)):  
 if self.weights[i][j] < minn:  
 if isValidEdge(i, j, inMST):  
 minn = self.weights[i][j]  
 a = i  
 b = j  
  
 if a != -1 and b != -1:  
 print("Edge %d: (%d, %d) cost: %d" %  
 (edge\_count, a, b, minn))  
 edge\_count += 1  
 mincost += minn  
 inMST[b] = inMST[a] = True  
  
 print("Minimum cost = %d" % mincost)  
  
 def dfs(self, start, visited=None, nestings = None, currentNesting = 1):  
 if visited is None:  
 visited = list()  
  
 if nestings is None:  
 nestings = list()  
 nestings.append(currentNesting)  
  
 visited.append(start)  
  
 current = self.adjacency[start]  
 for next in np.where(current == 1)[0]:  
 if next in visited:  
 pass  
 else:  
 self.dfs(next, visited, nestings, currentNesting + 1)  
  
 return [visited, nestings]  
  
 def СЃonnectivity\_components(self):  
  
 components = set()  
  
 def modified\_dfs(start, visited=None):  
 if visited is None:  
 visited = list()  
 visited.append(start)  
 current = self.adjacency[start]  
 for next in set(np.where(current == 1)[0]) - set(visited):  
 modified\_dfs(next, visited)  
  
 return frozenset(visited)  
  
 for i in range(self.n):  
 components.add(modified\_dfs(i))  
  
 return components  
  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 G = Graph(14, 15, 16, 17)  
 G.draw()  
 #print(G.dfs(2))  
 #print(G.СЃonnectivity\_components())  
 #G.fsc()  
 G.BellmanFord(6)  
 #G.prim()