Министерство цифрового развития, связи

и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Лабораторная работа №6

по дисциплине «Структуры и алгоритмы обработки данных»

“Сетевые алгоритмы. Динамические алгоритмы поиска путей.”

Выполнила:

студентка группы БСТ2004

Моисеева С.Д.

Проверил:

А.Д. Чайка

Москва 2022

**Цель работы**: изучение основных сетевых алгоритмов/алгоритмов поиска путей и их реализация на Python

**Задание:**

1. Реализовать алгоритм поиска кратчайшего расстояния между двумя вершинами ориентированного взвешенного графа в соответствии с вариантом.
2. Предусмотреть задание графа в виде матрицы смежности/инцидентности, читаемой из файла, либо графически с помощью пользовательского интерфейса.
3. Разработать графический интерфейс пользователя с визуализацией графа и отображением кратчайшего расстояния между задаваемыми пользователем вершинами.
4. По результатам работы проанализировать временную сложность работы заданного алгоритма в зависимости от числа узлов и ребер графа. Данные представить в виде таблицы.

**Ход работы.**

**Задание 1**

Ниже представлена реализация алгоритма Левита с использованием библиотеки collections (для реализации упорядоченной очереди и дэка)

def levit(gr, start, end=None):  
 dist = defaultdict(lambda: inf)  
 dist[start] = 0  
 path = {start: (start, ())}  
 m0 = set()  
 m1, m1\_urg = IndexedQueue.fromkeys([start]), IndexedQueue()  
 m2 = set(chain.from\_iterable((v for v, \_ in from\_u) for from\_u in gr.values())) - {start} # (u,v)∈E => v∈m2  
  
 def relax(u, v, w):  
 if (dist[v] > dist[u] + w):  
 dist[v] = dist[u] + w  
 path[v] = (v, path[u])  
 return True  
 return False  
  
 while m1 or m1\_urg:  
 u = m1\_urg.pop() if m1\_urg else m1.pop()  
 for v, c in gr.get(u, ()):  
 if v in m2:  
 m1.push(v)  
 m2.discard(v)  
 relax(u, v, c)  
 elif v in m1:  
 relax(u, v, c)  
 elif v in m0 and relax(u, v, c):  
 m1\_urg.push(v)  
 m0.discard(v)  
 m0.add(u)  
  
 if end is None:  
 return dist, path  
 elif end in path:  
 return dist[end], restore\_path(path[end])  
 else:  
 return inf, ()

Функции ввода данных в разных форматах:

def from\_matrix():  
 tmp = [s.split() for s in open("matrix2.txt", "r").readlines()]  
 tmp2 = [[s for s in tmp[0]]]  
 for i in tmp[1::]:  
 for j in range(1, len(i)):  
 if int(i[j]) != 100:  
 tmp2.append((tmp2[0][j - 1], i[0], int(i[j])))  
 tmp2.pop(0)  
 return tmp2  
  
  
def from\_classic():  
 tmp = [(s.split()[0], s.split()[1], int(s.split()[2])) for s in open("matrix.txt", "r").readlines()]  
 return tmp

Точка входа программы + графический интерфейс:

flag = input("Хотите задать граф матрицей или набором? y/n")  
edges = []  
if flag == 'y':  
 edges = from\_matrix()  
else:  
 edges = from\_classic()  
print(edges)  
startnode, endnode = input('Введите начальный и конечный узел').split()  
adj\_list = defaultdict(list)  
for u, v, w in edges:  
 adj\_list[u].append((v, w))  
# для симметричного графа  
# adj\_list[v].append((u, w))  
  
# print(dict(adj\_list))  
graph = networkx.Graph()  
edge\_lables\_dict = {}  
for i in list(adj\_list.items()):  
 node1 = i[0]  
 for j in i[1]:  
 graph.add\_edge(node1, j[0], color="grey")  
 edge\_lables\_dict[(node1, j[0])] = j[1]  
pos = networkx.spring\_layout(graph)  
# для таблицы времени  
# t = time.perf\_counter()  
cost, path = levit(adj\_list, startnode, endnode)  
# res = 'Время выполнения для пути длиной ' + str(len(path)) + ': {0:.6f} сек'.format((time.perf\_counter() - t))  
# with open('timetable.txt', 'a') as file:  
# file.write(res + "\n")  
print('Кратчайший путь', path, 'весом:', cost)  
path1 = []  
for i in range(1, len(path)):  
 path1.append((path[i - 1], path[i]))  
graph.remove\_edges\_from(path1)  
graph.add\_edges\_from(path1, color='red')  
colors = networkx.get\_edge\_attributes(graph, 'color').values()  
networkx.draw(graph, pos, edge\_color=colors, with\_labels=True, arrows=True)  
networkx.draw\_networkx\_edge\_labels(  
 graph, pos,  
 edge\_labels=edge\_lables\_dict,  
 font\_color='blue'  
)  
# print(path1)  
plt.show()

Результат выполнения задания представлен на рисунке 1:

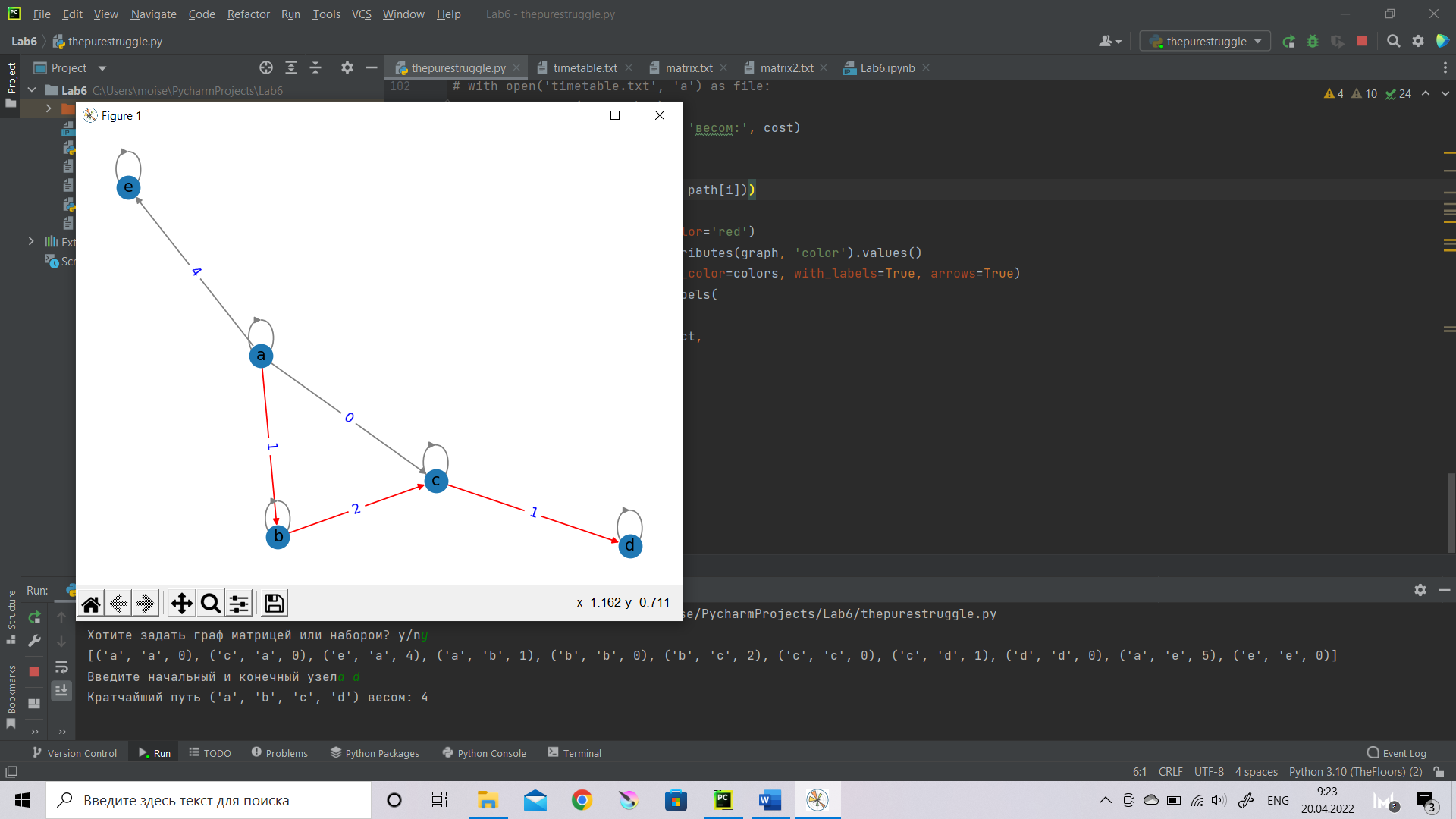


Рисунок 1 – Результат выполнения с вводом из матрицы на параметрах a d

Оценка времени в зависимости от веса:

Время выполнения для пути длиной 4: 0.000108 сек  
Время выполнения для пути длиной 3: 0.000094 сек  
Время выполнения для пути длиной 2: 0.000068 сек  
Время выполнения для пути длиной 1: 0.000066 сек

Вывод: в ходе работы были изучены основные сетевые алгоритмы/алгоритмы поиска путей и реализован алгоритм Левита с графическим представлением и возможностью разных способов задания начального графа

Ссылка на github: https://github.com/mmaider/SAOD6

**Список литературы**

1. Камаев В.А., Костерин В.В. Технологии программирования. М.: Высшая школа,

2006.

2. Жоголев Е.А.Технология программирования. – М.: Научный мир, 2004.

Git:

3. Scott Chacon, Ben Straub «Pro Git»

4. git-scm.com