Федеральное агентство связи

Ордена Трудового Красного Знамени

федеральное государственное бюджетное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Математическая кибернетика и информационные технологии»

Лабораторные работы по дисциплине

«Структуры и алгоритмы обработки данных»

Лабораторная работа №6

«Поиск кратчайшего пути»

Выполнила студентка группы БСТ2002

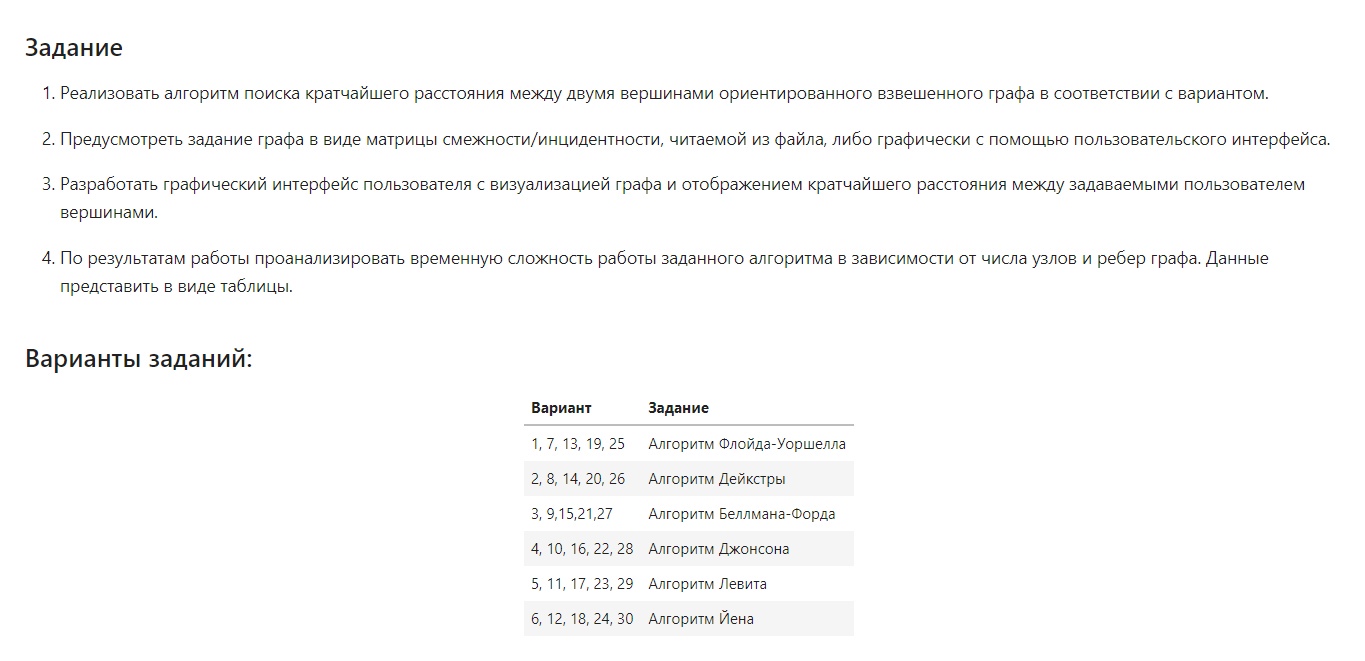
Сергеева А.А.

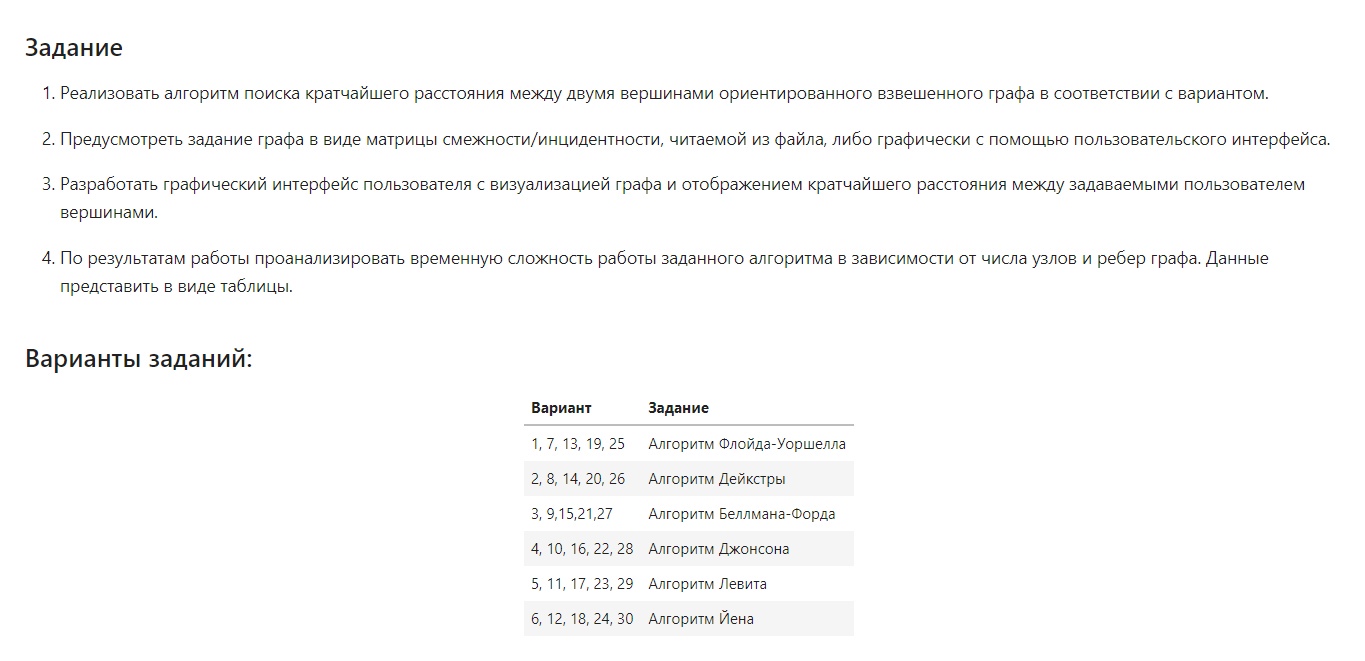
Вариант №16

Проверил: Аршинов Е.А.

Москва 2022

**Задание**





**Ход работы**

Буду использовать алгоритм Джонсона. Это алгоритм, который совмещает в себе алгоритмы Беллмана-Форда и Дейкстры. Этапы:

1. Строится новая вершина, соединяющаяся со всеми остальными. Новые дуги имеют вес 0.
2. Применяется алгоритм Беллмана-Форда, где вычисляются расстояния от новой вершины до всех остальных.
3. Изменяем веса дуг по формуле.
4. Удаляем новую вершину.
5. Применяем алгоритм Дейкстры ко всем вершинам.

**Файл jonson.py:**

Создаем класс Graph, где есть конструктор с количеством вершин и список для хранения графа. Функция addEdge добавляет ребра к графу. В функции BellmanFord инициализируем расстояние от новой вершины до всех остальных и избавляемся от отрицательных весов.

Далее формируем матрицу смежности.

После переходим к осуществлению алгоритма Дейкстры.

Сначала опишем функцию arg\_min, использующую ниже. Она возвращает вершину с минимальным весом. Там мы перебираем строку в матрице и находим минимальное значение, не входящее в множество просмотренных вершин.

Инициализируем число вершин в графе, последнюю строку таблицы, стартовую вершину, множество просмотренных вершин, вес для стартовой вершины, оптимальные связи между вершинами.

В цикл while заходим пока не просмотрим все вершины. Там мы перебираем все связанные вершины со стартовой. И если вершина еще не просмотрена, то формируем ее вес по формуле. Далее делаем проверку: если текущий вес меньше уже записанного, то перезаписываем этот новый вес. Потом в цикле while выбираем следующий узел с наименьшим весов с помощью функции arg\_min. А дальше добавляем новую вершину в просмотренные.

Выходя из цикла, формируем оптимальный маршрут.

Функция jon, где передаем количество ребер, список кортежей, начальную и конечную вершину. Внутри инициализируем граф с количеством ребер, а потом их все добавляем.

**Файл main.py:**

В этом файле мы представляем графически наш граф.

Читаем файл с вершинами и весами (по условию задания). В кортежах все значения в формате строк, переделываем из в числа.

Создаем направленный граф, принимающий список кортежей. Делаем вывод графа в планарном виде.

Считаем количество вершин в графе. Задаем множество, куда добавляем первые 2 элемента.

Устанавливаем размер множества (количество вершин графа).

Вызываем функция jon для нахождения минимального пути.

Перезаписываем граф так, чтобы красным цветом выделялся наш путь.

**Код программы:**

**Файл jonson.py:**

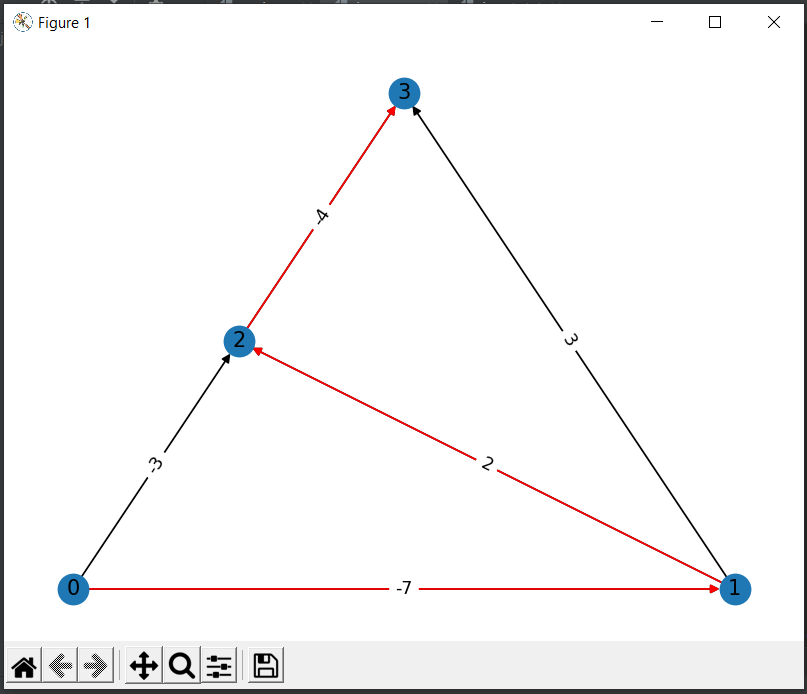
import math  
  
  
class Graph:  
  
 def \_\_init\_\_(self, vertices):  
 self.V = vertices  
 self.graph = []  
  
 def addEdge(self, u, v, w):  
 self.graph.append([u, v, w])  
  
 def BellmanFord(self, src, end):  
  
 dist = [math.inf] \* self.V  
 dist[src] = 0  
  
 for i in range(self.V - 1):  
 for u, v, w in self.graph:  
 if dist[u] != math.inf and dist[u] + w < dist[v]:  
 dist[v] = dist[u] + w  
  
 matrix = [[math.inf for j in range(len(dist))] for i in range(len(dist))]  
 for i in self.graph:  
 matrix[i[0]][i[1]] = i[2] + dist[i[0]] - dist[i[1]] # новое значение получаем по формуле Форда-Беллмана  
  
 # for i in matrix:  
 # print(i)  
  
 def arg\_min(T, S):  
 amin = -1  
 m = math.inf  
 for i, t in enumerate(T):  
 if t < m and i not in S:  
 m = t  
 amin = i  
  
 return amin  
  
 N = len(matrix)  
 T = [math.inf] \* N  
  
 v = 0  
 S = {v}  
 T[v] = 0  
 M = [0] \* N  
  
 while v != -1:  
 for j, dw in enumerate(matrix[v]):  
 if j not in S:  
 w = T[v] + dw  
 if w < T[j]:  
 T[j] = w  
 M[j] = v  
  
 v = arg\_min(T, S)  
 if v >= 0:  
 S.add(v)  
  
  
 path = [end]  
 while end != src:  
 end = M[path[0]]  
 path = [end] + path  
  
 return path  
  
def jon(size, tuples, start, end):  
 g = Graph(size)  
 for i in tuples:  
 g.addEdge(i[0], i[1], i[2])  
  
 return g.BellmanFord(start, end)

**Файл main.py:**

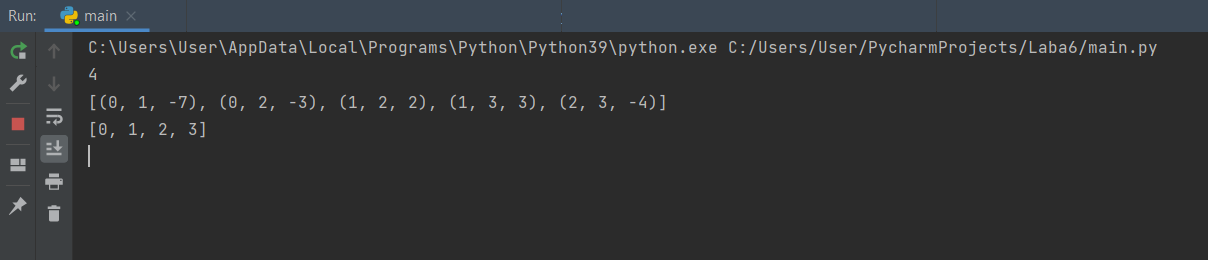
import networkx as nx  
import matplotlib.pyplot as plt  
  
import jonson as jon  
  
with open('inputs.txt', 'r') as inputs:  
 global graph  
 graph = []  
  
 for line in inputs:  
 graph.append(tuple(line[:-1].split(', ')))  
  
  
graph = [(int(i[0]), int(i[1]), int(i[2])) for i in graph]  
  
  
G = nx.DiGraph()  
G.add\_weighted\_edges\_from(graph)  
pos = nx.planar\_layout(G)  
  
  
nx.draw(G, with\_labels='True', pos=pos)  
nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=nx.get\_edge\_attributes(G, 'weight'))  
  
  
vertices = set()  
for i in graph:  
 vertices.add(i[0])  
 vertices.add(i[1])  
size = len(vertices)  
print(size)  
print(graph)  
  
path = jon.jon(size, graph, 0, 3)  
print(path)  
path = [(path[i], path[i+1]) for i in range(len(path)-1)]  
  
nx.draw\_networkx\_edges(G, pos=pos, edgelist=path, edge\_color='red')  
nx.draw\_networkx\_edge\_labels(G, pos, edge\_labels=nx.get\_edge\_attributes(G, 'weight'))  
  
plt.show()

**Результат работы программы:**

Вывод графически

****

Вывод в консоли

****

**Вывод**

В данной лабораторной работе научились применять алгоритм Джонсона для поиска наименьшего пути.