Лабораторная работа №3

Черепенников Роман 3 курс 8 группа

Задача 16. Кратчайший маршрут с дополнительным временем преодоления перекрёстка

План города представляет собой множество перекрёстков, соединённых дорогами (по дороге движение разрешено в обоих направлениях). Каждая дорога задаётся номерами перекрёстков, которые она соединяет, и временем движения по ней. Между двумя различными перекрёстками может быть не более одной дороги. Дорога не может соединять перекрёсток сам с собой. Кроме того, время преодоления перекрёстка I равно qk_i, где q — заданная константа, а k_i — число дорог, подходящих к перекрёстку i. Необходимо найти кратчайший по времени маршрут от перекрёстка s до перекрёстка f. Конечный перекрёсток f преодолевать не надо, а стартовая вершина s маршрута преодолевается как перекрёсток.

Формат входных данных

В первой строке находятся числа N и M — число перекрёстков ($1 \le N \le 11000$) и число дорогна плане города ($0 \le M \le 100000$) соответственно. В каждой из следующих M строк файла сначала расположены номера перекрёстков, которые связывает очередная дорога, а затем время движения по ней (от 0 до 1000). В последней строке находятся номера перекрёстков s и f ($1 \le s, f \le N$), а также константа q ($1 \le q \le 100$).

Формат выходных данных

Если проехать от перекрёстка s до перекрёстка f можно, в первой строке выведите сообщение Yes, а во второй строке — минимальное время движения. Если проехать нельзя, то в единственной строке выведите сообщение No.

Пример

входной файл	выходной файл
6 9 1 2 2 1 3 1 2 4 1 2 5 1 3 4 3 3 5 1 4 5 2 4 6 1 5 6 1 1 6 1	Yes 12

Алгоритм решения:

Зададим граф матрицей смежности, $A(a_{ij}) \in R^{N \times N}$, где a_{ij} время движения по дороге связывающей перекрестки i и j.

Преобразуем эту матрицу следующим образом:

$$a_{ij} = a_{ij} + q * k_j, j \neq f$$

$$a_{if} = a_{ij}$$

Теперь воспользуемся алгоритмом Дейкстры, чтобы найти кратчайший путь из s в f. Здесь, не учтено время, которое необходимо провести на перекрестке s перед началом движения, поэтому к результату алгоритма прибавим $q * k_s$ и получим итоговый результат.

Листинг программы:

```
from typing import Tuple

import numpy as np


def read_from_file(path: str) -> Tuple[np.ndarray, int, int, int]:
    with open(path) as f:
        line = f.readline().split()
        n = int(line[0])
        m = int(line[1])
        graph_matrix = np.ones((n, n)) * np.inf

        for _ in range(m):
            line = f.readline().split()
            graph_matrix[int(line[0]) - 1, int(line[1]) - 1] = int(line[2])
            graph_matrix[int(line[1]) - 1, int(line[0]) - 1] = int(line[2])

        line = f.readline().split()
        start, finish, q = int(line[0]) - 1, int(line[1]) - 1, int(line[2])

        return graph_matrix, start, finish, q


def degree(graph_matrix: np.ndarray, v: int) -> int:
    """Find degree of graph node"""
    return np.count_nonzero(~np.isinf(graph_matrix[v]))


def degrees(graph_matrix: np.ndarray) -> np.ndarray:
    """Find degrees of every node in graph"""
    n = graph_matrix.shape[0]
    res = np.zeros(n)
    for i in range(n):
        res[i] = degree(graph_matrix, i)
    return res


def transform_graph_weights(graph_matrix: np.ndarray, finish: int, q: int) ->
np.ndarray:
    degs = degrees(graph_matrix)
    degs[finish] = 0
    return graph_matrix + degs * q


def select_node(distances: np.ndarray, visited: np.ndarray) -> int:
    min_ = np.min(distances[~visited])
    for i in range(distances.shape[0]):
        if not visited[i] and distances[i] == min_:
            return i
    return -1


def dijkstra(graph_matrix: np.ndarray, start: int) -> np.ndarray:
    """Find shortest path from start node to every other"""
    n = graph_matrix.shape[0]
    distances = np.ones(n) * np.inf
    distances[start] = 0
```

```
    visited = np.zeros(n, dtype=bool)

    for i in range(n):
        v = select_node(distances, visited)
        visited[v] = True
        if distances[v] == np.inf:
            break

        for j in range(n):
            distances[j] = min(distances[j], distances[v] +
graph_matrix[v][j])

    return distances


if __name__ == '__main__':
    graph_matrix, s, f, q = read_from_file('input.txt')
    graph_matrix = transform_graph_weights(graph_matrix, f, q)
    result = int(dijkstra(graph_matrix, s)[f]) + degree(graph_matrix, s)

    if result == np.inf:
        print('No')
    else:
        print('Yes')
        print(result)
```