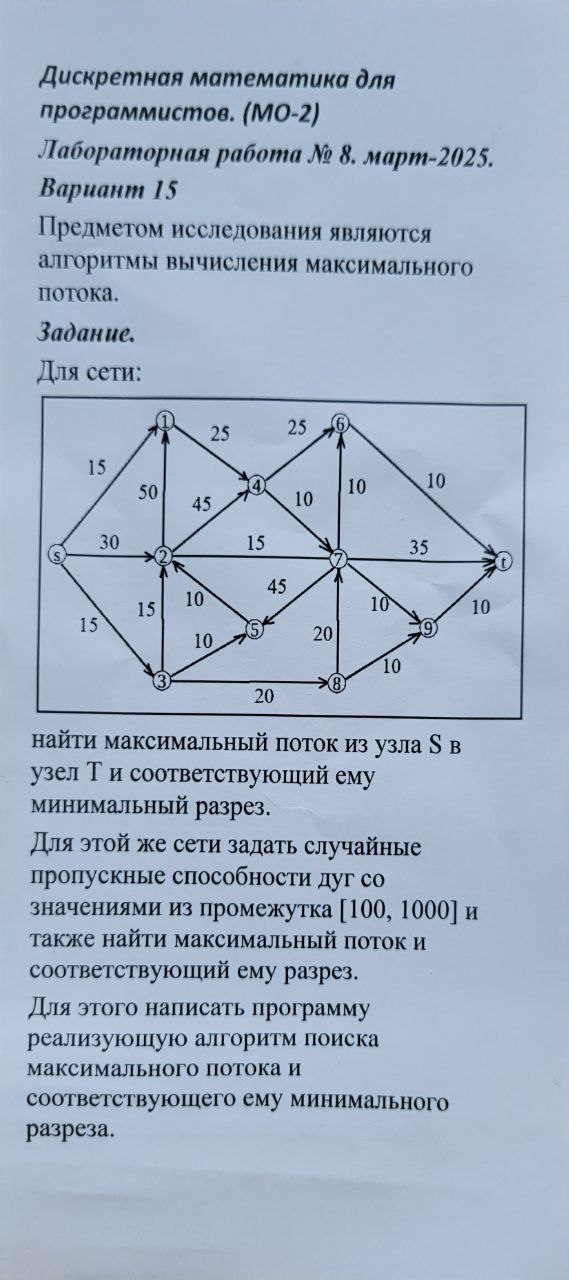
Лабораторная работа № 8

Вариант 15, Ус Владимир, группа 2МО-1



Код:

import random  
import networkx as nx  
  
# 1) Функция для построения сети с заданными (фиксированными) вместимостями:  
def build\_fixed\_network() -> nx.DiGraph:  
 *"""  
 Возвращает ориентированный граф NetworkX (nx.DiGraph)  
 с вершинами {S,A,B,C,D,E,F,G,H,I,T} и фиксированными пропускными способностями.  
 """* G = nx.DiGraph()  
 # Добавим все вершины (хотя добавлять явно необязательно, они появятся  
 # при добавлении дуг, но полезно явно задать порядок):  
 nodes = ["S","A","B","C","D","E","F","G","H","I","T"]#ss a1 b2 c3 d4 e5 f6 g7 h8 i9 tt  
 G.add\_nodes\_from(nodes)  
  
 # Список кортежей (u, v, capacity):  
 edges = [  
 ("S","A", 15), ("S","B", 30), ("S","C", 15),  
 ("A","D", 25),  
 ("B","A", 50), ("B","D", 45), ("B","G", 15),  
 ("C","B", 15), ("C","E", 10), ("C","H", 10),  
 ("D","F", 25), ("D","G", 10),  
 ("E","B", 10),  
 ("F","T", 10),  
 ("G","T", 35), ("G","F", 10), ("G","I", 10), ("G","B", 15),  
 ("H","G", 20), ("H","I", 10),  
 ("I", "T", 10)  
 ]  
  
 for (u, v, cap) in edges:  
 G.add\_edge(u, v, capacity=cap)  
  
 return G  
  
  
# 2) Функция для построения аналога сети, но с "случайными" вместимостями  
# каждая из которых uniformly ∈ [100, 1000]  
def build\_random\_network() -> (nx.DiGraph, list):  
 *"""  
 Строит ту же структуру дуг, что и build\_fixed\_network(),  
 но вместо фиксированных capacities задаёт их случайно из диапазона [100,1000].  
 Возвращает кортеж (G, edges\_with\_caps), где edges\_with\_caps —  
 список (u, v, cap) для всех дуг.  
 """* G = nx.DiGraph()  
 nodes = ["S","A","B","C","D","E","F","G","H","I","T"]  
 G.add\_nodes\_from(nodes)  
  
 # Базовая структура дуг без capacity:  
 base\_edges = [  
 ("S", "A"), ("S", "B"), ("S", "C"),  
 ("A", "D"),  
 ("B", "A"), ("B", "D"), ("B", "G"),  
 ("C", "B"), ("C", "E"), ("C", "H"),  
 ("D", "F"), ("D", "G"),  
 ("E", "B"),  
 ("F", "T"),  
 ("G", "T"), ("G", "F"), ("G", "I"), ("G", "B"),  
 ("H", "G"), ("H", "I"),  
 ("I", "T")  
 ]  
  
 edges\_with\_caps = []  
 for (u, v) in base\_edges:  
 cap = random.randint(100, 1000)  
 edges\_with\_caps.append((u, v, cap))  
 G.add\_edge(u, v, capacity=cap)  
  
 return G, edges\_with\_caps  
  
  
# 3) Функция, которая на вход принимает ориентированный граф G, исток s, сток t,  
# и возвращает кортеж (max\_flow\_value, S\_side, T\_side), где (S\_side, T\_side) —  
# разбиение вершин, соответствующее минимальному разрезу (S\_side ∪ T\_side = V(G)).  
def compute\_maxflow\_and\_mincut(G: nx.DiGraph, s: str, t: str):  
 *"""  
 Использует встроенные в networkx алгоритмы:  
 - maximum\_flow(...) — для поиска max flow и flow dict  
 - minimum\_cut(...) — для поиска минимального разреза  
 Возвращает:  
 max\_flow\_value, (set\_of\_nodes\_in\_S\_side, set\_of\_nodes\_in\_T\_side)  
 """* # 3.1. Вычисляем максимальный поток:  
 max\_flow\_value, flow\_dict = nx.maximum\_flow(G, s, t, capacity="capacity")  
  
 # 3.2. Найдём минимальный S–T разрез (value и сами множества вершин):  
 cut\_value, (S\_side, T\_side) = nx.minimum\_cut(G, s, t, capacity="capacity")  
  
 # для проверки должно быть cut\_value == max\_flow\_value  
 return max\_flow\_value, cut\_value, (S\_side, T\_side), flow\_dict  
  
  
# === Главное тело программы ===  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("\n=== 1) Запись исходных (фиксированных) пропускных способностей ===")  
 fixed\_G = build\_fixed\_network()  
  
 # 4.1. Решаем задачу максимального потока / минимального разреза на фиксированной сети  
 print("\n--- Вычисляем max-flow и min-cut для фиксированной сети ---")  
 max\_flow\_val, cut\_val, (S\_part, T\_part), flow\_dict = compute\_maxflow\_and\_mincut(fixed\_G, "S", "T")  
  
 print(f"Результат (фиксированная сеть):\n"  
 f" • Максимальный поток S→T = {max\_flow\_val}\n"  
 f" • Минимальный разрез = {cut\_val}\n"  
 f" • S-часть разреза = {sorted(S\_part)}\n"  
 f" • T-часть разреза = {sorted(T\_part)}\n")  
  
 # Явно выведем границу разреза — дуги, идущие из S\_part в T\_part:  
 cut\_arcs = []  
 for u, v, data in fixed\_G.edges(data=True):  
 if (u in S\_part) and (v in T\_part):  
 cut\_arcs.append((u, v, data["capacity"]))  
 print(" • Дуги, пересекающие минимальный разрез (u→v : capacity):")  
 for (u, v, cap) in cut\_arcs:  
 print(f" {u} → {v} : {cap}")  
  
 # 4.2. Пример «случайной» сети (вместимости ∈ [100,1000]) и снова max-flow/min-cut  
 print("\n--- Генерируем сеть со случайными пропускными способностями ∈ [100, 1000] ---")  
 random\_G, random\_edges = build\_random\_network()  
  
 print(" → Случайные вместимости дуг (u→v : capacity):")  
 for (u, v, cap) in random\_edges:  
 print(f" {u} → {v} : {cap}")  
  
 print("\n--- Вычисляем max-flow и min-cut для «случайной» сети ---")  
 max\_flow\_rand, cut\_val\_rand, (Srand, Trand), flow\_rand\_dict = \  
 compute\_maxflow\_and\_mincut(random\_G, "S", "T")  
  
 print(f"Результат (случайная сеть):\n"  
 f" • Максимальный поток S→T = {max\_flow\_rand}\n"  
 f" • Минимальный разрез = {cut\_val\_rand}\n"  
 f" • S-часть разреза = {sorted(Srand)}\n"  
 f" • T-часть разреза = {sorted(Trand)}\n")  
 print(" • Дуги, пересекающие минимальный разрез (u→v : capacity):")  
 for (u, v, data) in random\_G.edges(data=True):  
 if (u in Srand) and (v in Trand):  
 print(f" {u} → {v} : {data['capacity']}")  
  
 print("\n=== Работа программы завершена ===\n")

Вывод:

