**Problem: Maximum Water Supply in a City Network**

**Problem Statement**

You are an engineer responsible for managing the water supply network of a city. The network is represented as a directed graph where nodes represent junctions and edges represent pipelines with specific capacities. Your task is to determine the maximum amount of water that can be transported from the main reservoir (source) to the city's central water tank (sink).

**Scenario**

The city's water supply network consists of several junctions connected by pipelines. Each pipeline has a maximum capacity for water flow. Due to varying demands and capacities, it is crucial to understand the maximum water flow that the network can handle from the main reservoir to the central tank. This information will help in optimizing the water distribution and planning future infrastructure improvements.

**Input Format**

* The first line contains four integers n, m, s, and t, representing the number of junctions (nodes), the number of pipelines (edges), the source junction, and the sink junction respectively.
* The next m lines contain three integers u, v, and c indicating that there is a directed pipeline from junction u to junction v with capacity c.

**Constraints**

* 2 <= n <= 100
* 0 <= m <= n×(n−1)
* 0 <= s, t < n
* s != t
* 1 <= c <= 1000

**Output Format**

* Print an integer denoting the maximum water flow from the source junction s to the sink junction t.

**Sample Input**

4 5 0 3

0 1 100

0 2 100

1 2 1

1 3 100

2 3 100

**Sample Output**

200

**Explanation**

The maximum flow from junction 0 (main reservoir) to junction 3 (central water tank) is 200 units.

**Additional Test Cases**

**Test Case 1**

6 7 0 5

0 1 16

0 2 13

1 2 10

1 3 12

2 1 4

2 4 14

3 2 9

3 5 20

4 3 7

4 5 4

**Output:**

0

**Test Case 2**

5 7 0 4

0 1 10

0 2 5

1 2 15

1 3 10

2 3 10

2 4 10

3 4 10

**Output:**

15

**Test Case 3**

3 2 0 2

0 1 10

1 2 5

**Output:**

5

**Test Case 4**

5 6 0 4

0 1 10

0 2 5

1 3 10

2 3 15

3 4 10

1 4 5

**Output:**

15

**Test Case 5**

4 3 0 3

0 1 8

1 2 6

2 3 5

**Output:**

5

**Solution**

To solve this problem, we can use the Edmonds-Karp algorithm, which is an implementation of the Ford-Fulkerson method for computing the maximum flow in a flow network. The algorithm uses BFS to find augmenting paths.

Here's the solution in Python:

python

from collections import defaultdict, deque

def bfs(capacity, source, sink, parent):

visited = set()

queue = deque([source])

visited.add(source)

while queue:

current = queue.popleft()

for neighbor, cap in capacity[current].items():

if neighbor not in visited and cap > 0:

parent[neighbor] = current

visited.add(neighbor)

queue.append(neighbor)

if neighbor == sink:

return True

return False

def edmonds\_karp(n, edges, source, sink):

capacity = defaultdict(lambda: defaultdict(int))

for u, v, c in edges:

capacity[u][v] += c # handle multiple edges between same nodes

parent = {}

max\_flow = 0

while bfs(capacity, source, sink, parent):

path\_flow = float('Inf')

s = sink

while s != source:

path\_flow = min(path\_flow, capacity[parent[s]][s])

s = parent[s]

v = sink

while v != source:

u = parent[v]

capacity[u][v] -= path\_flow

capacity[v][u] += path\_flow

v = parent[v]

max\_flow += path\_flow

return max\_flow

# Input

n, m, s, t = map(int, input().strip().split())

edges = [list(map(int, input().strip().split())) for \_ in range(m)]

# Output

print(edmonds\_karp(n, edges, s, t))

4o