from collections import defaultdict, deque

# Create the graph

graph = defaultdict(dict)

# Example edges and capacities (add more as needed based on Table 2 from the PDF)

edges = [

('v56', 'v95', 1), ('v56', 'v94', 6), ('v95', 'v96', 3), ('v96', 'v162', 2),

('v162', 'v117', 2), ('v117', 'v70', 2), ('v70', 'v163', 2), ('v163', 'v164', 8),

('v164', 'v67', 9), ('v67', 'v68', 15), ('v68', 'v165', 7), ('v165', 'v14', 7),

('v14', 'v15', 9), ('v15', 'v26', 3), # Ending at v26

# Add all other edges as needed

]

# Add edges to the graph

for u, v, capacity in edges:

graph[u][v] = capacity

graph[v][u] = 0 # reverse edge for residual capacity

# BFS to find path from source to sink

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

visited = set()

queue = deque([source])

visited.add(source)

while queue:

u = queue.popleft()

for v, capacity in rGraph[u].items():

if v not in visited and capacity > 0:

visited.add(v)

parent[v] = u

if v == sink:

return True

queue.append(v)

return False

# Ford-Fulkerson Algorithm

def ford\_fulkerson(graph, source, sink):

rGraph = defaultdict(dict)

for u in graph:

for v in graph[u]:

rGraph[u][v] = graph[u][v]

parent = {}

max\_flow = 0

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

path\_flow = float('inf')

s = sink

while s != source:

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

s = parent[s]

# update residual capacities

v = sink

while v != source:

u = parent[v]

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

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

v = u

max\_flow += path\_flow

return max\_flow

# Run the algorithm

source = 'v56'

sink = 'v26'

max\_flow = ford\_fulkerson(graph, source, sink)

print(f"Maximum number of public transport units (angkots) that can pass: {max\_flow}")

output –

Maximum number of public transport units (angkots) that can pass: 5