Assignment 10

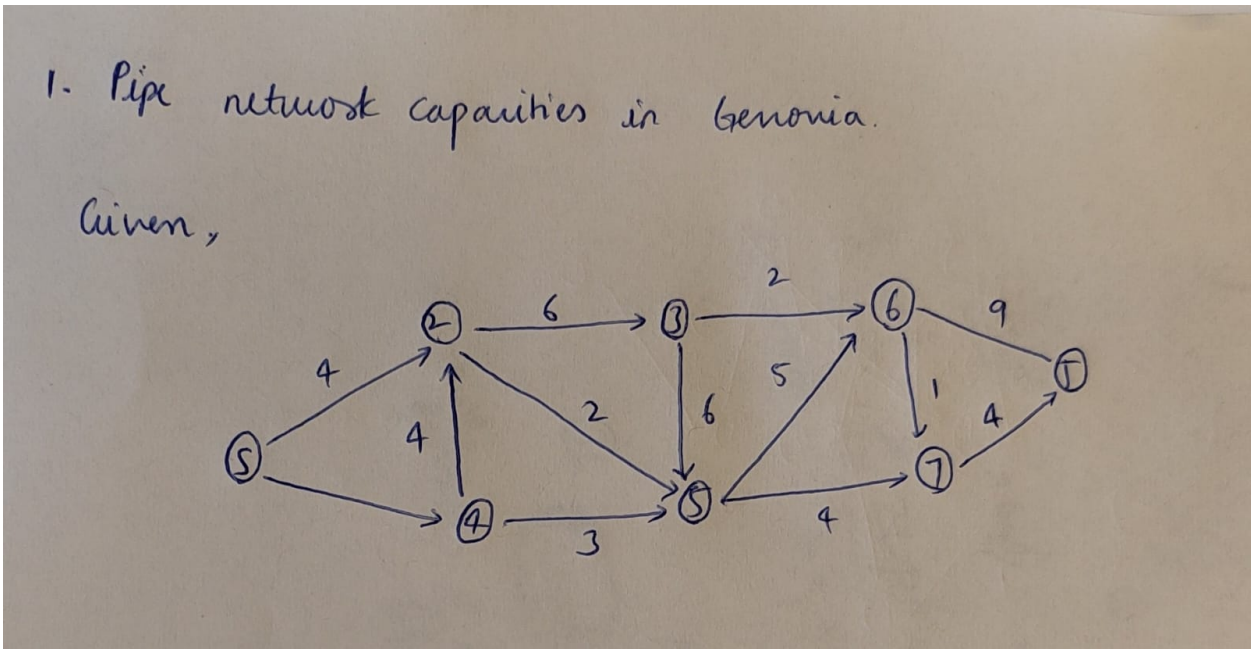
1) Pipe network capacities in Genovia (20 pts total)

After finding out you are secretly part of the royal family of Genovia, you inherit a 16th-century Genovian castle with an elaborate plumbing system that has accumulated pipes, junctions, and clogs over four centuries. Instead of a diagram, you are given a list of pipes and their capacities leading from the water source to your bathroom

Pipes	**S,2**	**S,4**	**2,3**	**2,5**	**4,2**	**4,5**	**3,5**
Capacity	4	7	6	2	4	3	6

(10 points total) Draw the flow graph of your new castle and list:

i. The shortest augmenting path by number of edges (and it’s bottleneck)



If we examine the pipes and capacity according to the provided data, we get a graph of the pipe connections as shown in the following picture, where S is the source of water entering the castle and T is the bathroom sink. The links given here are pipes, and the nodes given here are the corresponding valves.

Using path-finding methods like breadth-first search or the Dijkstra's algorithm, we determine the shortest augmented path. After completing the necessary steps, we obtain the output shown below
S → 2 → 5 → 6 → T.
The corresponding bottleneck capacity is 2.

ii. The highest capacity augmenting path (and it’s bottleneck)

Answer:

Using path-finding methods like depth-first search or the Ford-Fulkerson algorithm, we determine the highest capacity augmented path, after persforming the operations, we obtain the output as shown below:

S → 4 → 2 → 3 → 5 → 6 → T
The corresponding bottleneck capacity is 4

iii. You'd like to know if it is safe to install a modern shower, or if this will eventually overflow the historic bathtub. Use the Ford Fulkerson algorithm to determine the max flow of this flow network/graph. Please draw your final residual graph and write the calculated max flow.

We use the Ford-Fulkerson algorithm, which is a widely used method for finding the maximum flow in a flow network, which is a directed graph with capacities assigned to its edges. The concept of "flow" represents the movement of a resource (e.g., water, data, or traffic) from a source node to a sink node through the network, subject to capacity constraints on each edge.

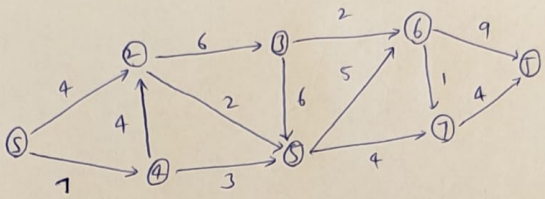
The goal of the Ford-Fulkerson algorithm is to find the maximum flow that can be sent from the source node to the sink node while respecting the capacity of the edges. It was formulated by Lester R. Ford Sr. and Delbert Fulkerson in 1962.

Here's a high-level overview of the algorithm:

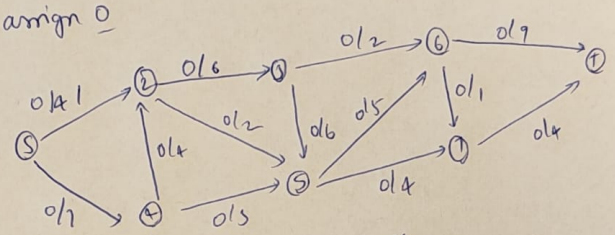
1. Initialize the flow on each edge to zero.
2. While there exists an augmenting path in the residual graph (a graph that represents available capacity on each edge), do:
   a. Find an augmenting path from the source to the sink using a search algorithm like Breadth-First Search (BFS) or Depth-First Search (DFS).
   b. Determine the maximum flow that can be sent along this path. This value is limited by the minimum capacity of the edges in the augmenting path.
   c. Update the flow values along the augmenting path: increase the flow on forward edges and decrease the flow on backward edges.
3. When no augmenting path can be found, the algorithm terminates, and the flow is at its maximum.
4. by using the steps we found the max flow of the graph given

1. Pipe network capacities in Genovia

Given,



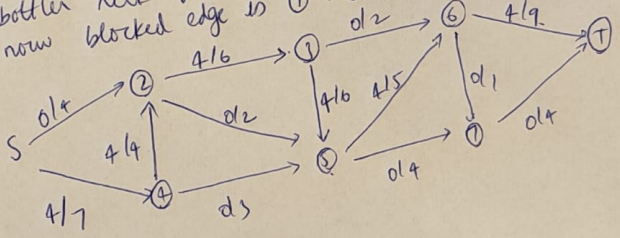
initially assign 0



next we select an augumenting path.

S → 4 → 2 → 3 → 5 → 6 → T

this has bottle neck of 4
now blocked edge is 4 → 2.

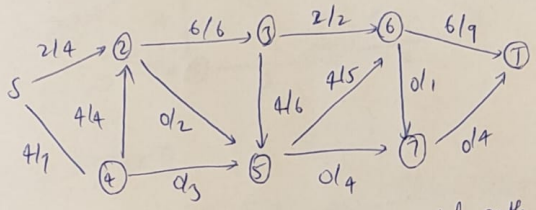


next we choose an other augmented path

$S \longrightarrow 2 \longrightarrow 3 \longrightarrow 6 \longrightarrow T$

here blocked edge is 3→6 and 2→3

bottlec neck is 2

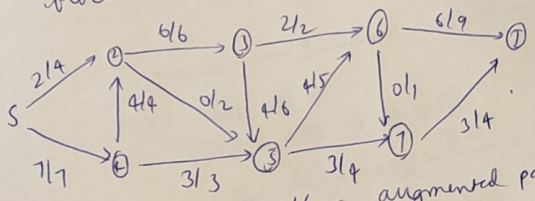


next we choose an other augmented path.

$S \longrightarrow 4 \longrightarrow 5 \longrightarrow 7 \longrightarrow T$

here bottleneck is 3

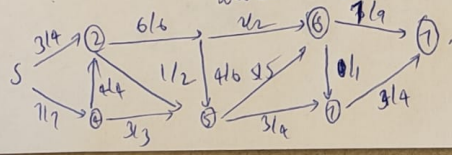
blocked edges are S→4 and 4→5.

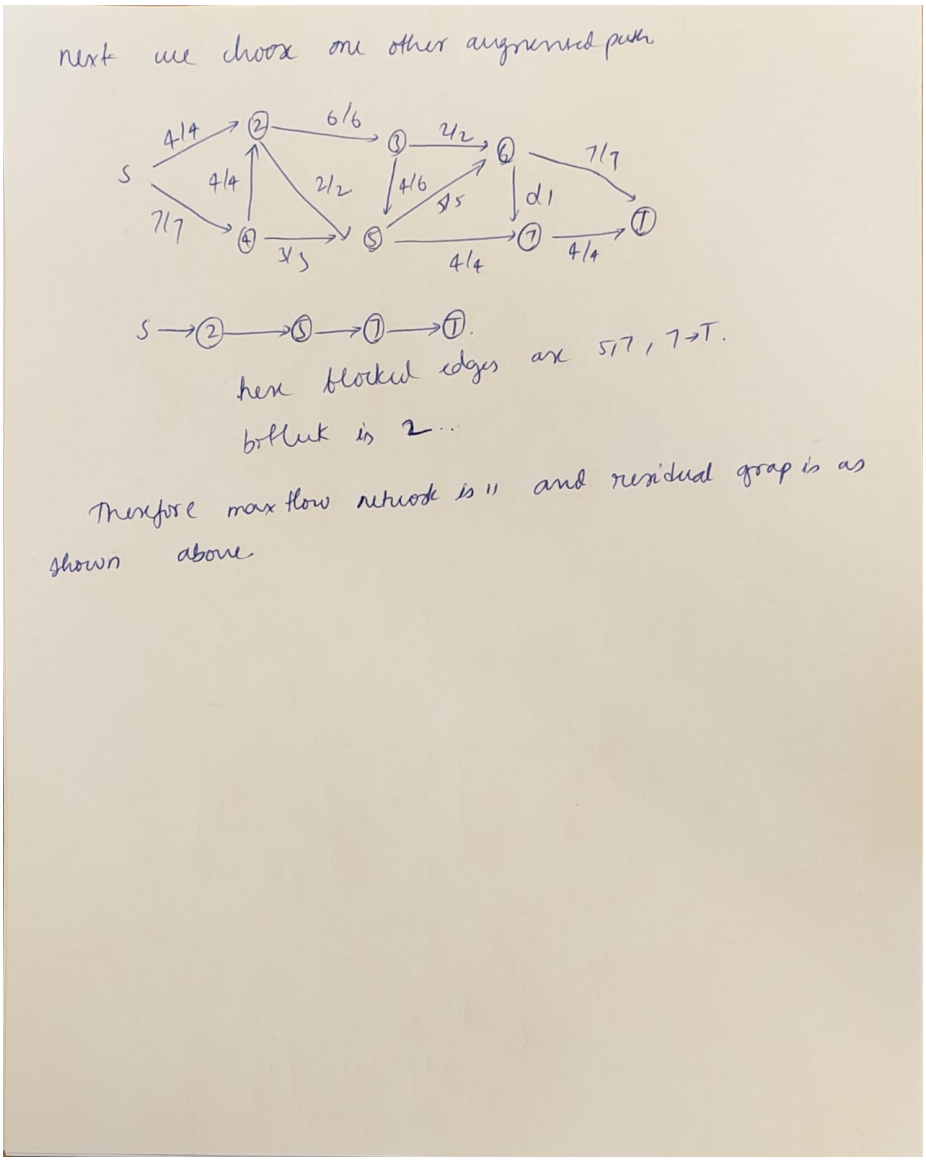


next we choose one other augmented path: S→2→6→T

here the bottle neck is 1.

blocked edges are 2→3, 5→6





Thus, the maximum flow network is 11 and the corresponding residual graph is displayed above.

2) Augmenting Paths (10 points)

Not all augmenting paths are equal, and starting with different paths leads to different residual graphs, although all selections produce the same max-flow result. Determine a process for selecting your augmenting paths. Justify your answer. **Hint:** most implementations of Ford-Fulkerson take a greedy approach.

Answer:

A common approach used in many implementations is the "greedy" method, which involves selecting augmenting paths that have available capacity (positive residual capacity) and are reachable from the source node.

Here's a process for selecting augmenting paths using the greedy approach:

1. Start with an initial flow of zero on all edges.
2. While there exists a path from the source to the sink in the residual graph (where residual capacity > 0), do:
   a. Perform a graph traversal (e.g., BFS or DFS) from the source node to the sink, searching only for edges with positive residual capacity.
   b. If a path is found, select the augmenting path with the minimum capacity among the edges in the path. This choice ensures

that the maximum possible flow increment is made on the selected path.
c. Update the flow on each edge in the augmenting path by adding the minimum capacity found in step 2b.
d. Update the residual capacities of the edges in the graph based on the new flow values. For each forward edge (u, v) in the augmenting path, reduce the residual capacity by the flow that was just added. For each backward edge (v, u) in the augmenting path, increase the residual capacity by the flow that was just added.
e. Repeat steps 2a to 2d until no path from the source to the sink with positive residual capacity can be found.

Justification for the greedy approach:

The greedy approach is chosen because it guarantees convergence to the maximum flow when the capacities are integral. In each iteration, the algorithm increases the flow along the selected augmenting path by the minimum capacity in the path. This ensures that the algorithm is always making progress towards the optimal solution, increasing the flow while respecting the capacities of the edges.

Moreover, using the greedy approach with BFS or DFS for finding augmenting paths ensures that the algorithm finds the shortest augmenting path (in terms of the number of edges) in each iteration. This choice can lead to faster convergence compared to other approaches, as it minimizes the number of iterations required to reach the maximum flow.

However, it's important to note that the Ford-Fulkerson algorithm itself does not guarantee polynomial time complexity for all possible scenarios. In certain cases, such as when the capacities are real numbers or when the algorithm uses a bad selection strategy for augmenting paths, it may not converge or may take an impractically long time to do so. This is why more efficient algorithms like the Edmonds-Karp algorithm (using BFS for path selection) and the Push-Relabel algorithm

Edmonds-Karp algorithm and the Push-Relabel algorithm are more efficient than the Ford-Fulkerson algorithm. The Edmonds-Karp algorithm has a time complexity of $O(E^2)$, while the Push-Relabel algorithm has a time complexity of $O(V^2 * E)$. This means that they can find a maximum flow in a network much faster than the Ford-Fulkerson algorithm.

The Edmonds-Karp algorithm uses a breadth-first search to find an augmenting path. This is a more efficient way to find an augmenting path than the depth-first search used by the Ford-Fulkerson algorithm.

The Push-Relabel algorithm uses a different approach to finding an augmenting path. It maintains a label for each vertex in the network. The label of a vertex represents the minimum amount of flow that can be pushed from the vertex to its neighbors. The Push-Relabel algorithm then iteratively pushes flow from vertices with high labels to vertices with low labels. This process continues until no more flow can be pushed.

Both the Edmonds-Karp algorithm and the Push-Relabel algorithm are more efficient than the Ford-Fulkerson algorithm. They are also more complex, so they may not be suitable for all applications. However, they are a good choice for applications where speed is important.