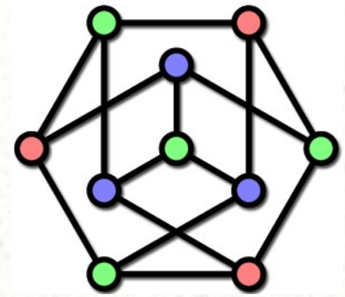


# *APPLIED GRAPH THEORY AND ALGORITHMS (CSC4066)*

## *Flows in Networks*

*Dr. Hasin A Ahmed*
*Assistant Professor*
*Department of Computer Science*
*Gauhati University*

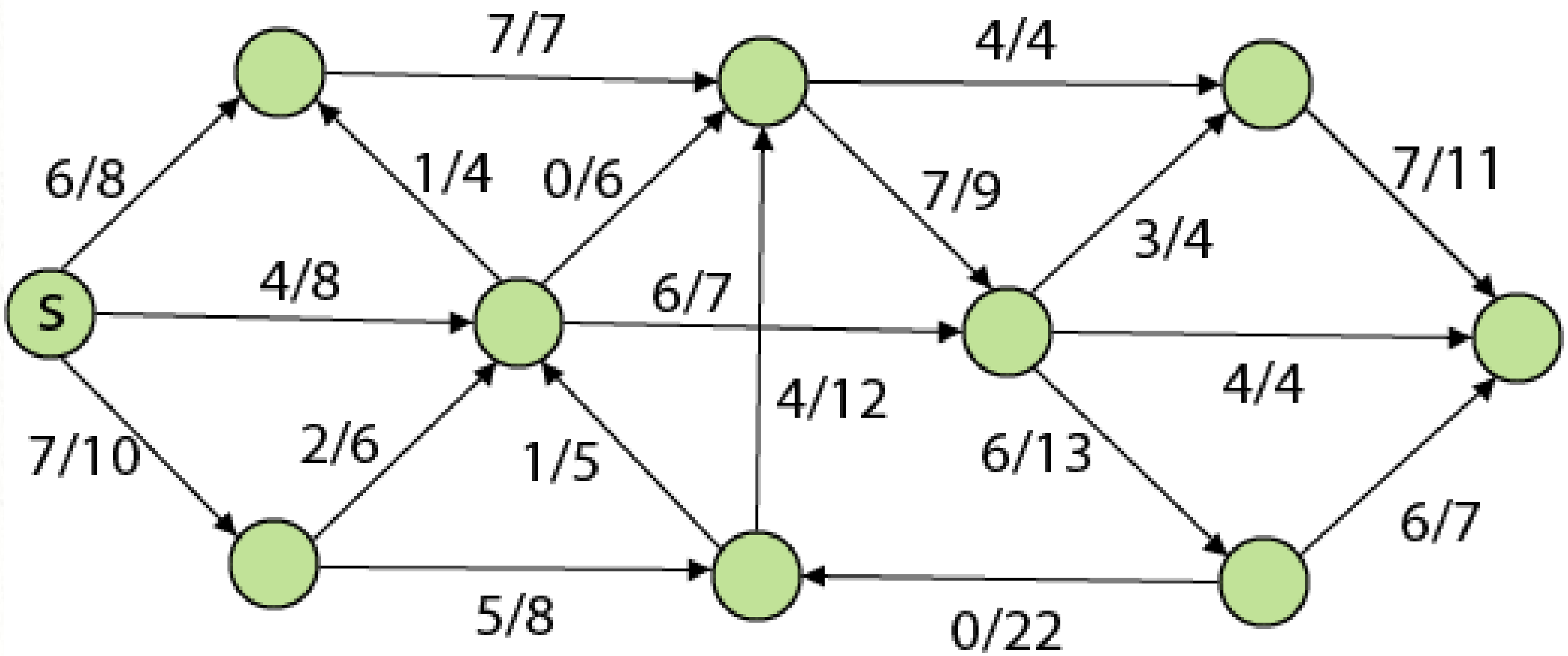
# Flow Network

- Flow Network is a directed graph that is used for modeling material Flow
- There are two different vertices; one is a source *s* which produces material at some steady rate, and another one is sink *t* which consumes the content at the same constant speed
- There will be intermediate nodes between *s* and *t*

# Flow Network

- Each edge in the network will have *capacity* and *flow* associated with it
- *Flow* is the net flow of units between the pair of connected nodes
- An arc’s *flow* cannot exceed it’s *capacity* value
- Some real-life problems like the flow of liquids through pipes, current through wires and delivery of goods can be modeled using flow networks

# Flow Network



# Flow Network

- Flow Network is a directed graph $G = (V, E)$ such that
- For each edge $(u, v) \in E$, we associate a nonnegative weight capacity $c(u, v) \geq 0$.If $(u, v) \notin E$, we assume that $c(u, v) = 0$.
- There are two distinguishing points, the source s, and the sink t;
- For every vertex $v \in V$, there is a path from s to t containing v.

# Flow Network

- Let G = (V, E) be a flow network. Let s be the source of the network, and let t be the sink. A flow in G is a real-valued function f: V x V→R such that the following properties hold:
- Capacity Constraint: For all u, v ∈ V, we need
- $f(u, v) \leq c(u, v)$
- Skew Symmetry: For all u, v ∈ V, we need
- $f(u, v) = - f(u, v)$.
- Flow Conservation: For all u ∈ V-{s, t}, we need

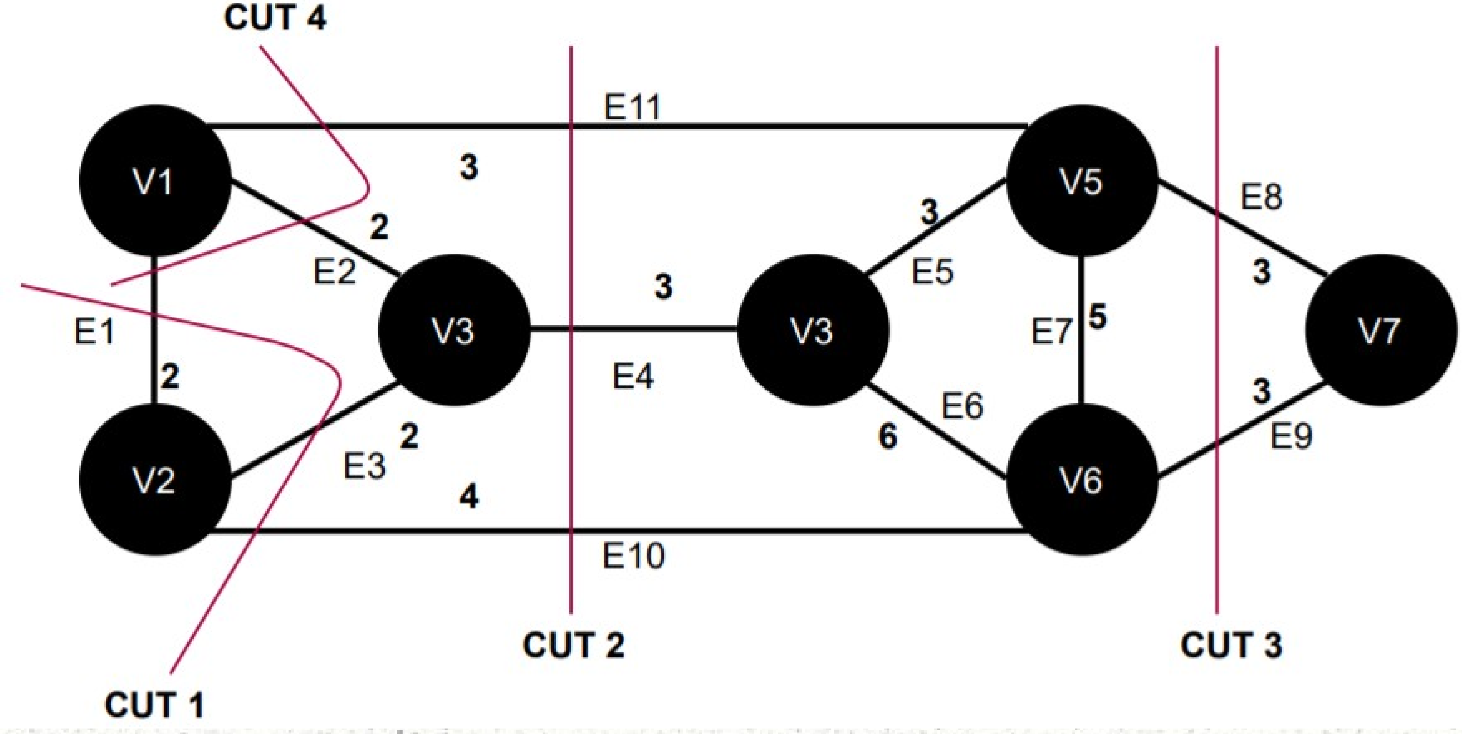
$$\boxed{\sum_{v \in V} f(u,v) = \sum_{u \in V} f(u,v) = 0}$$

# Flow Network

- One interpretation of the Flow-Conservation Property is that the positive net flow entering a vertex other than the source or sink must equal the positive net flow leaving the vertex

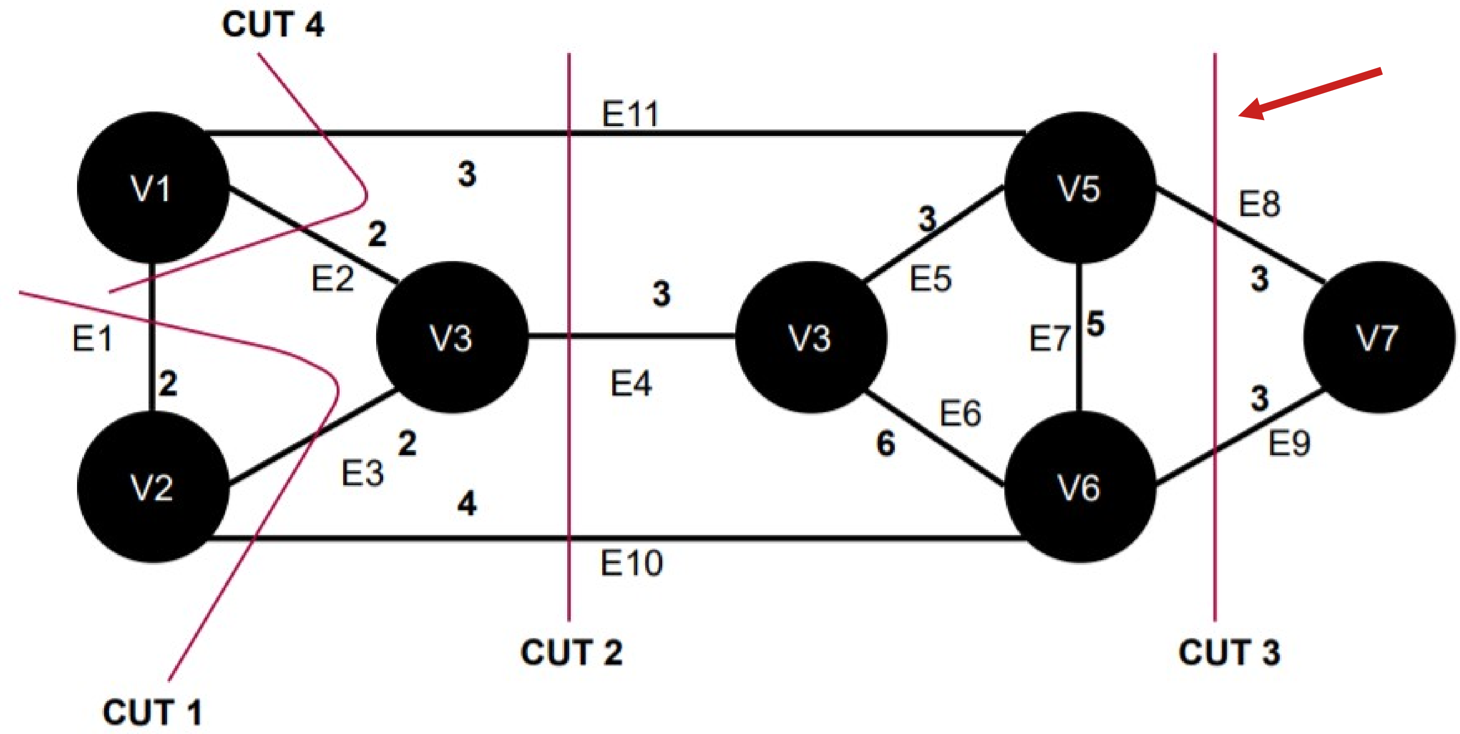
# Max flow min cut theorem

- Minimum Cut: The minimum cut of a weighted graph is defined as the minimum sum of weights of edges that, when removed from the graph, divide the graph into two sets.



# Max flow min cut theorem

- Minimum Cut: The minimum cut of a weighted graph is defined as the minimum sum of weights of edges that, when removed from the graph, divide the graph into two sets.



# Max flow min cut theorem

- Max flow: Maximum flow is defined as the maximum amount of flow that the graph or network would allow to flow from the source node to its sink node.

- ***The max-flow min-cut theorem states that the maximum flow through any network from a given source to a given sink is exactly equal to the minimum sum of a cut***

# Residual Network

- The residual capacity of an arc with respect to a flow *f*, denoted $c_f$, is the difference between the arc's capacity and its flow. That is, $c_f(e) = c(e)-f(e)$
- From this we can construct a residual network, denoted $G_f(V,E_f)$, which models the amount of available capacity on the set of arcs in $G = (V, E)$
- More formally, given a flow network $G$, the residual network $G_f$ has the node set $V$, arc set $E_f = \{e \in V \times V : c_f(e) > 0\}$ and capacity function $c_f$.

# Augmenting Path

- An augmenting path is a path $(u_1, u_2, ..., u_k)$ in the residual network, where $u_1 = s$, $u_k = t$, and $c_f(u_i, u_{i+1}) > 0$
- A network is at maximum flow if and only if there is no augmenting path in the residual network $G_f$

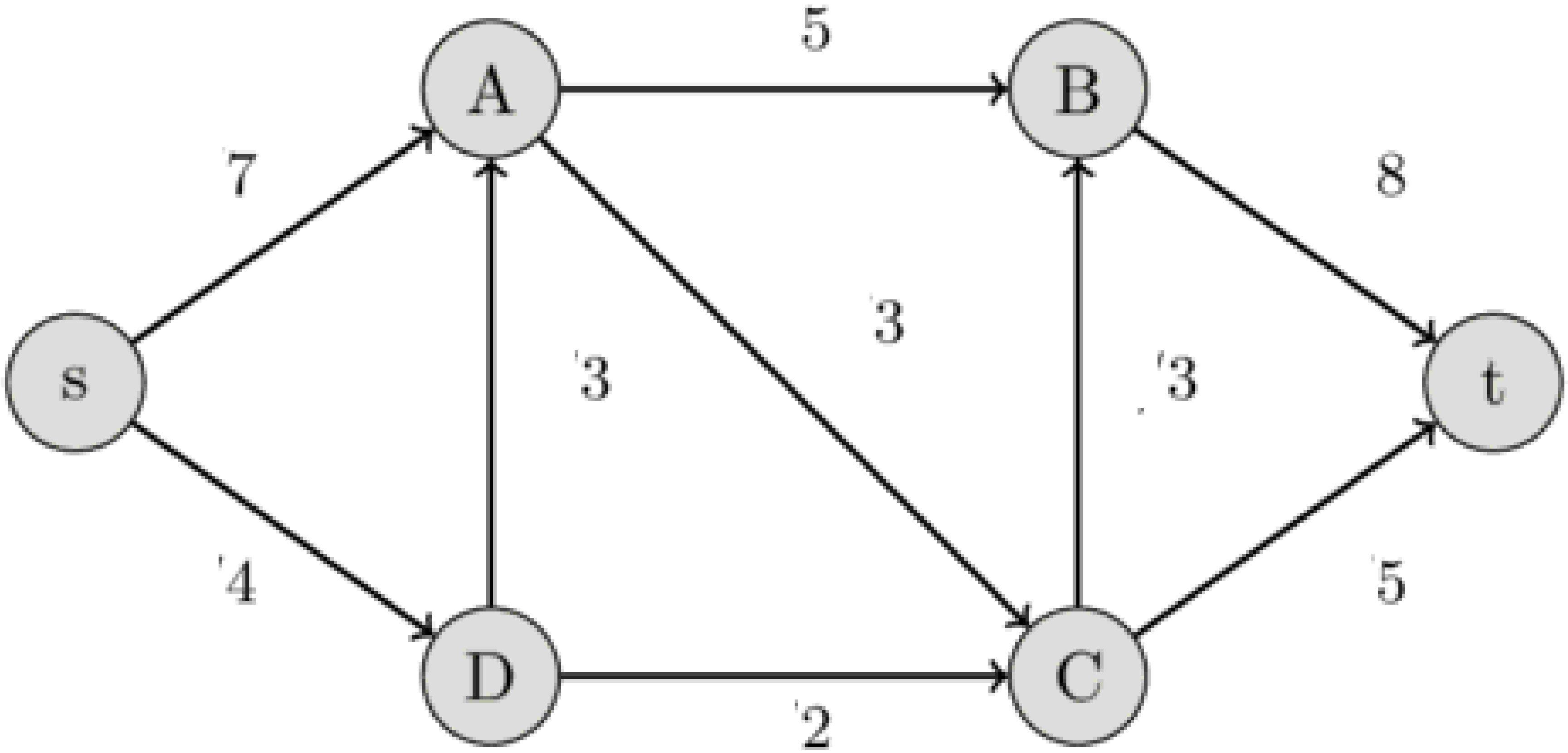
# Ford Fulkerson Algorithm

- It finds the maximum flow of a network or graph
- Initially, the algorithm starts by setting the flow value between the source and sink node to 0.
- At each iteration, we find an augmented path and increase the flow value.
- We’ll terminate the algorithm and return the flow value when no more augmented paths can be found.

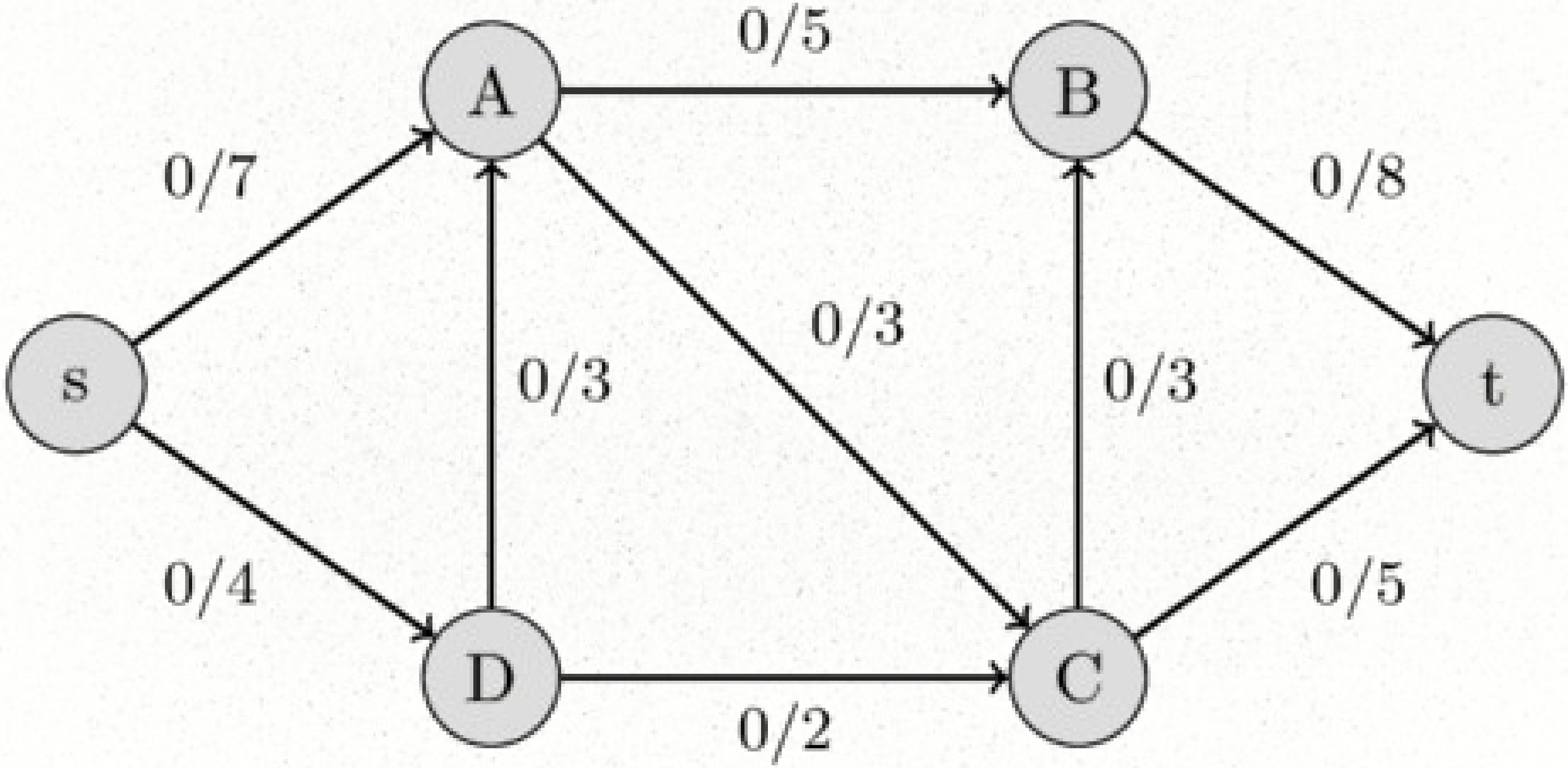
# Ford Fulkerson Algorithm

- The algorithm follows:
- Initialize the flow in all the edges to 0.
- While there is an augmenting path between the source and the sink, add this path to the flow.
- Update the residual graph.

# Ford Fulkerson Algorithm

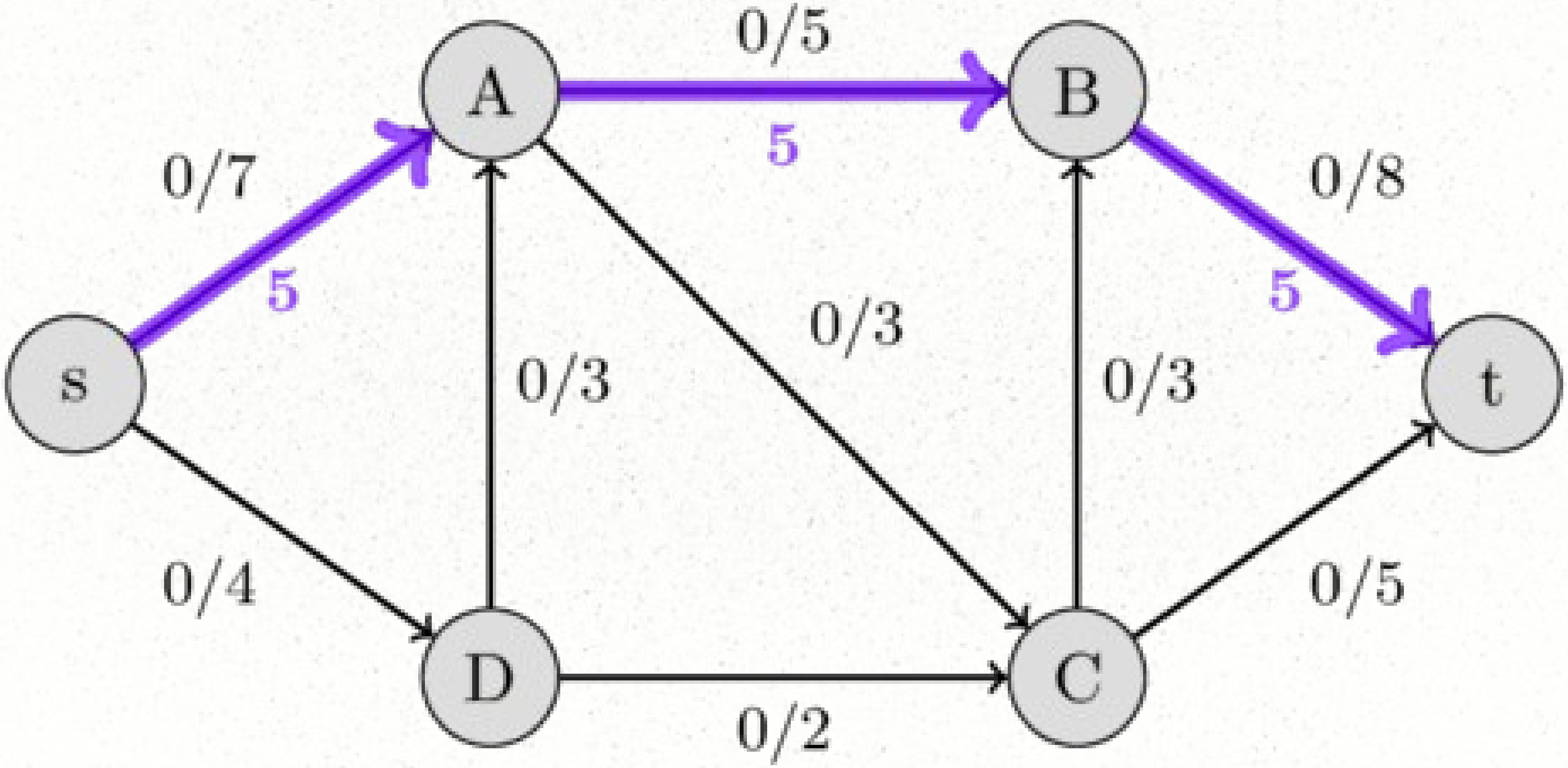


# Ford Fulkerson Algorithm

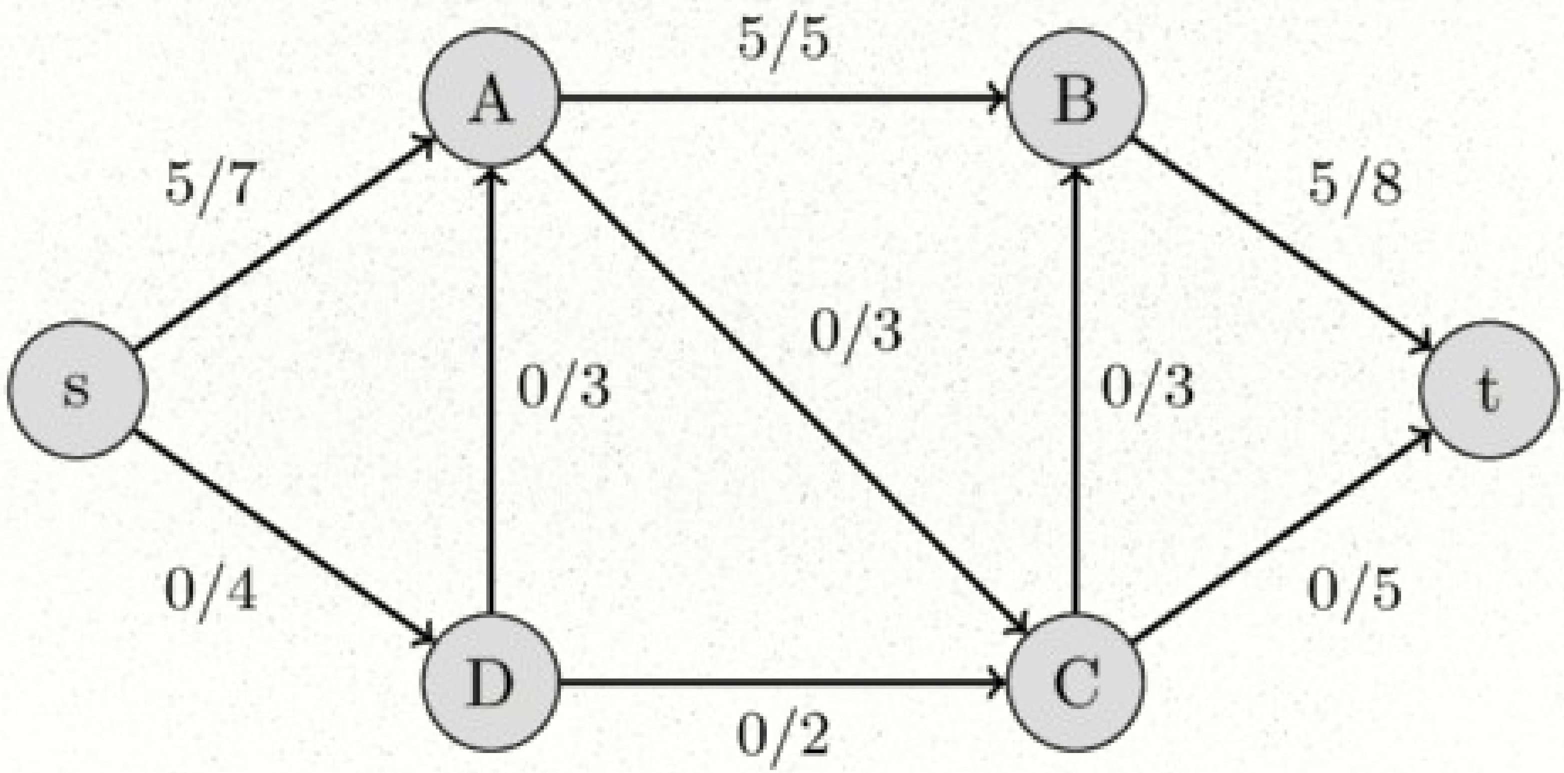


# Ford Fulkerson Algorithm

Residual Capacity=5

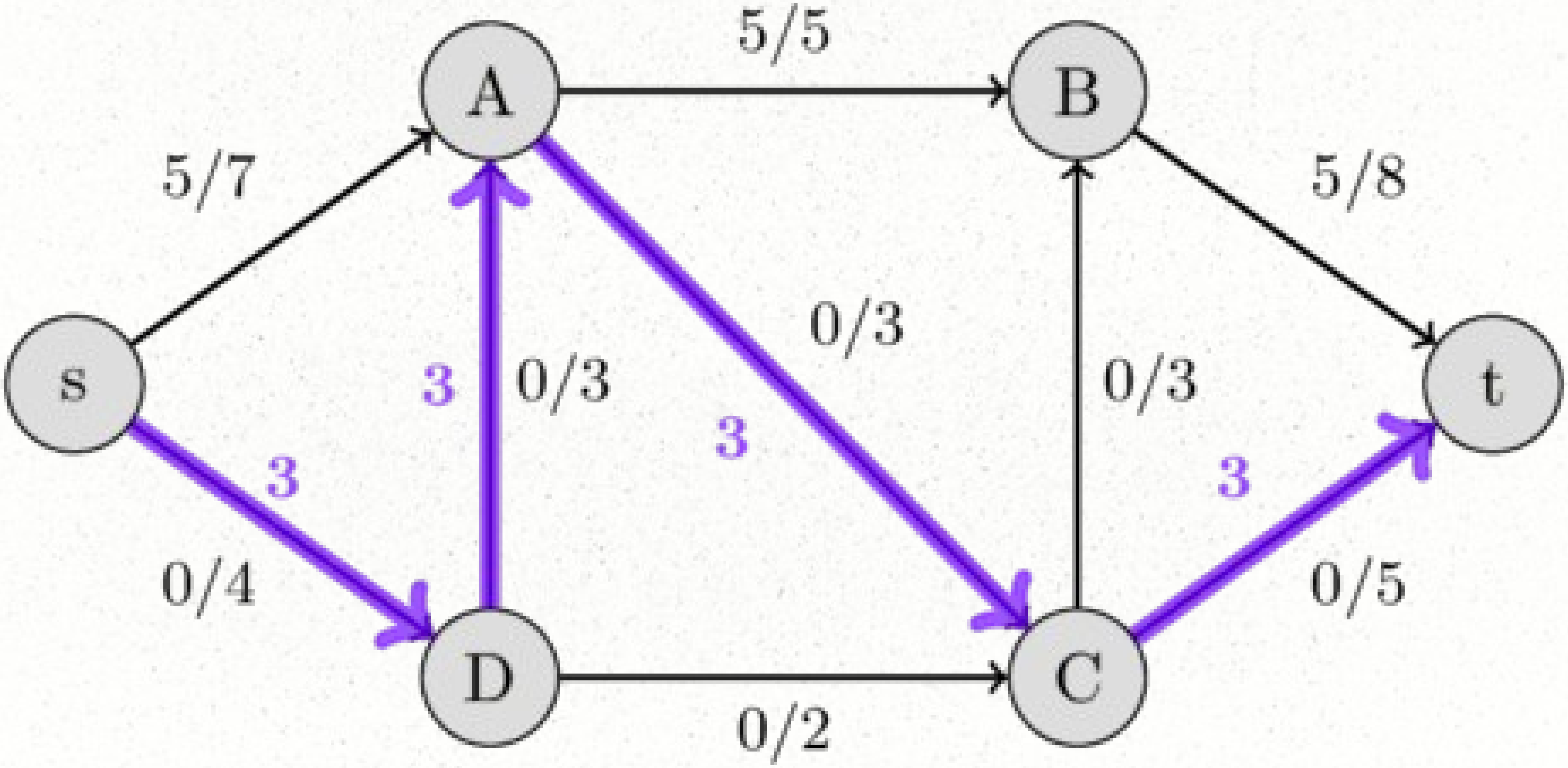


# Ford Fulkerson Algorithm

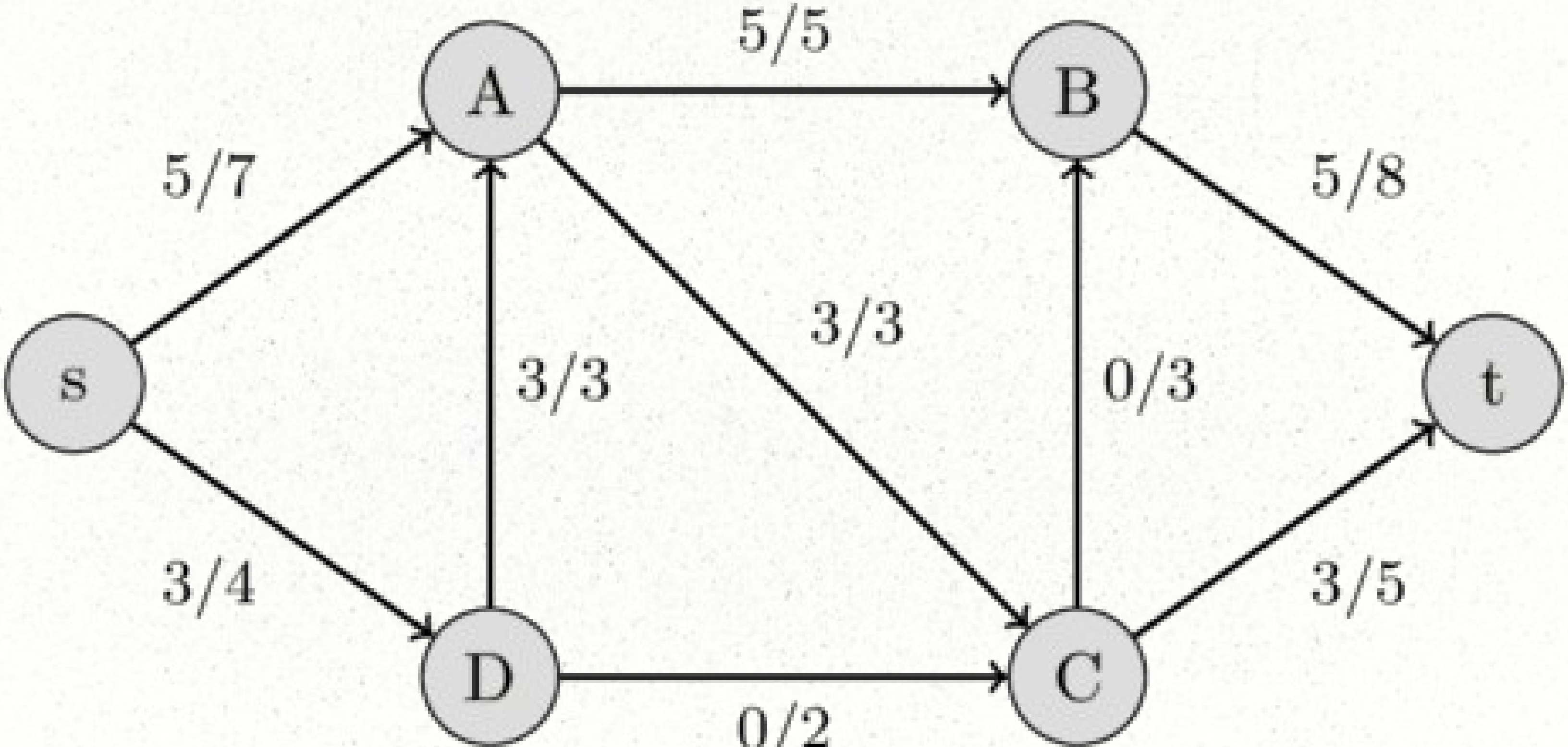


# Ford Fulkerson Algorithm

Residual Capacity=3

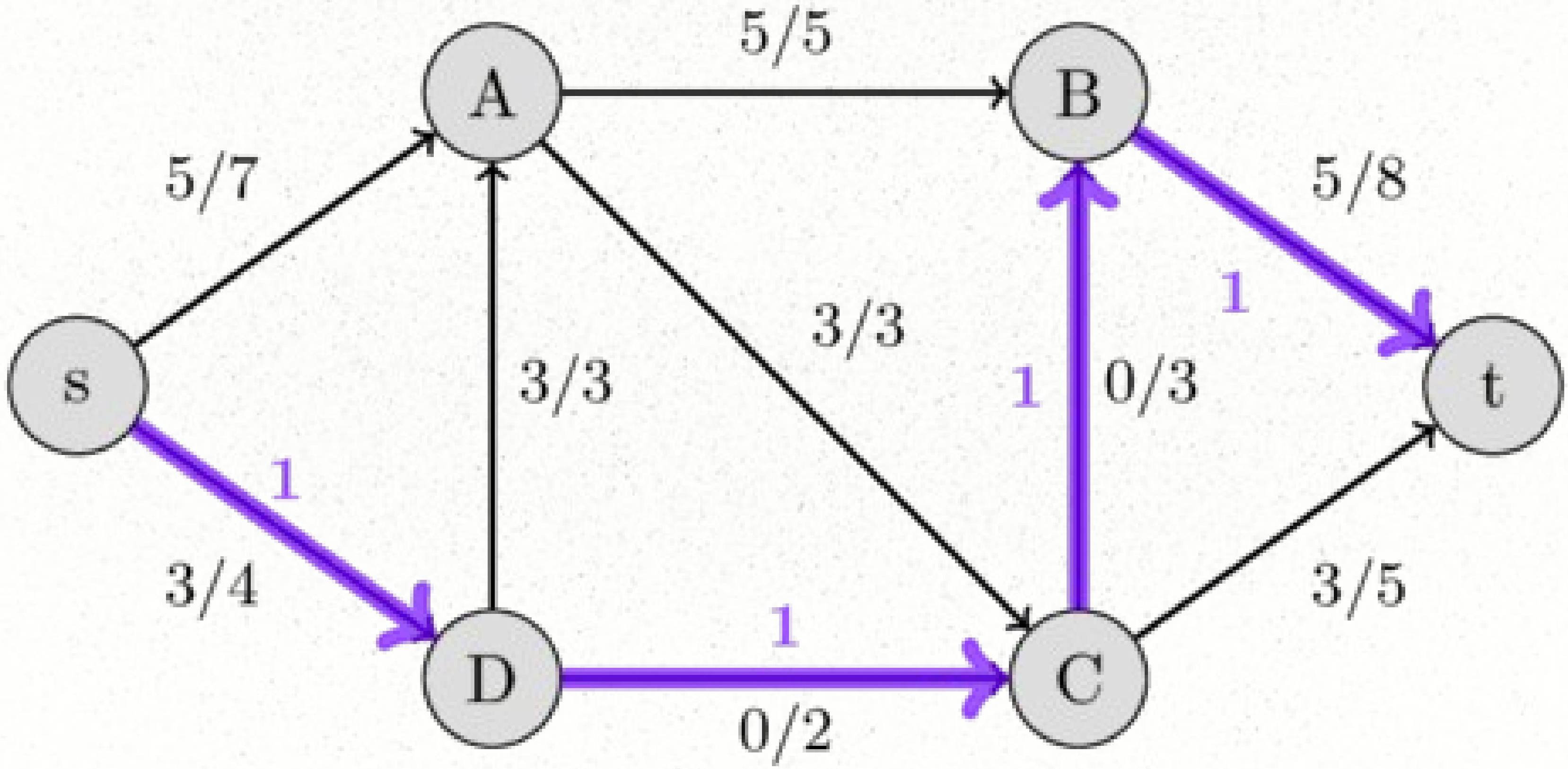


# Ford Fulkerson Algorithm

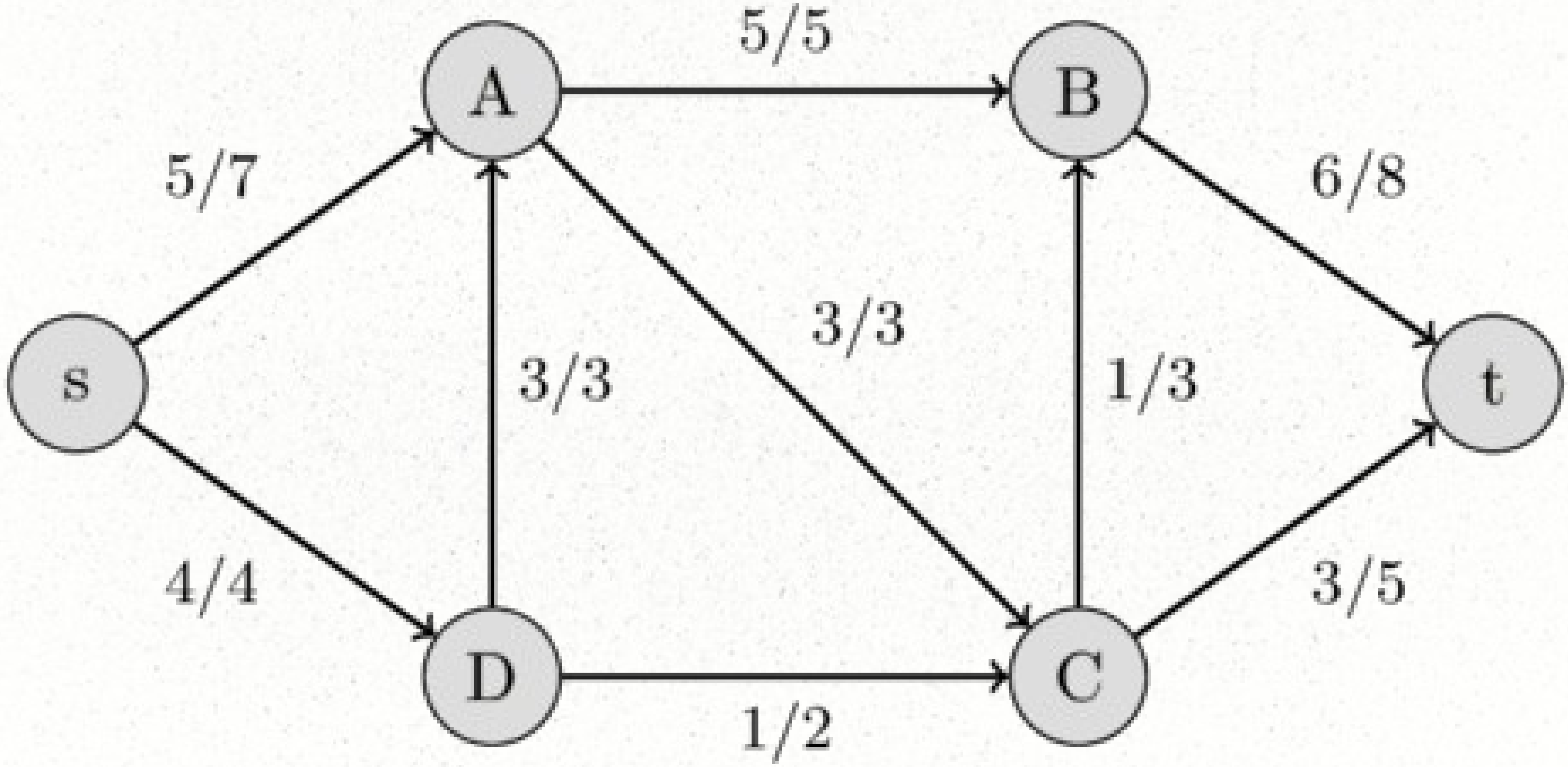


# Ford Fulkerson Algorithm

Residual Capacity=1

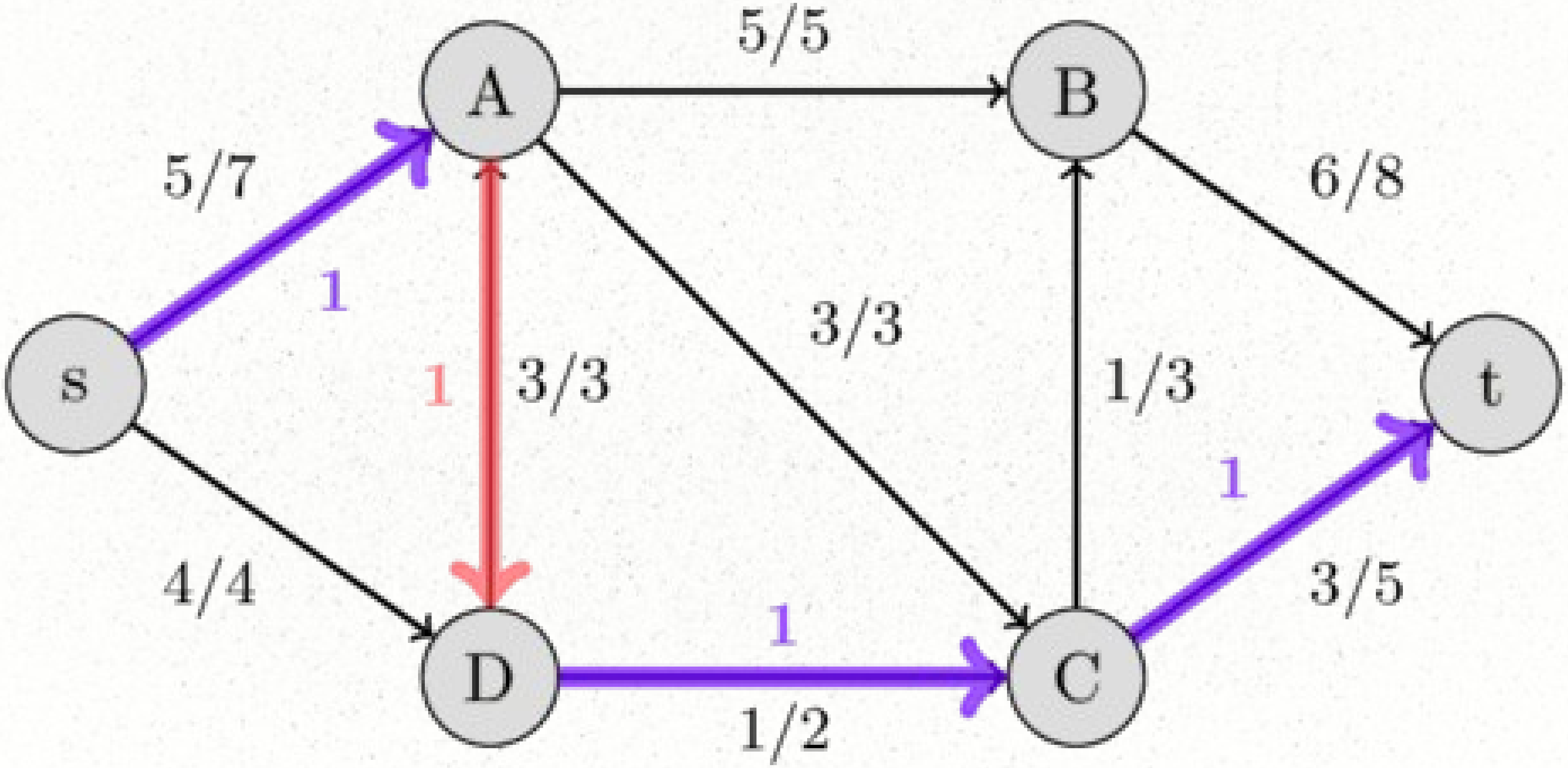


# Ford Fulkerson Algorithm



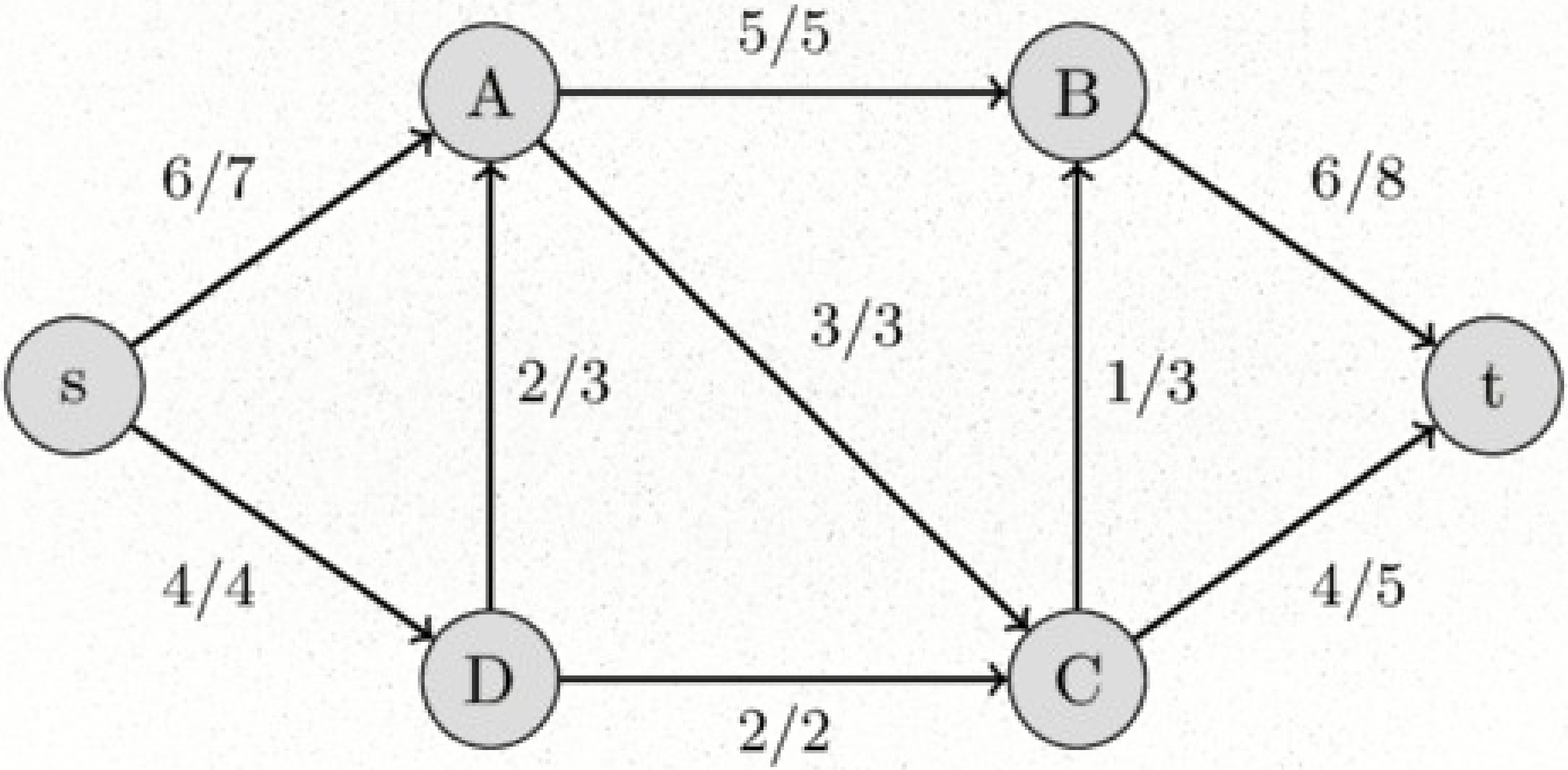
# Ford Fulkerson Algorithm

Residual Capacity=1




Non-full forward edge
Non-zero reverse edge

# Ford Fulkerson Algorithm



# Ford Fulkerson Algorithm

- Maximal Flow is the sum of residual capacities of the augmenting paths=5+3+1+1=10

# Ford Fulkerson Algorithm

- Maximal Flow is the sum of residual capacities of the augmenting paths=5+3+1+1=10
- Ford-Fulkerson method doesn't specify a method of finding the augmenting path
- Possible approaches are using DFS or BFS
- Time complexity of Ford-Fulkerson is O(EF), where F is the maximal flow of the network

# Edmonds Karp Algorithm

- Edmonds-Karp algorithm is just an implementation of the Ford-Fulkerson method that uses BFS for finding augmenting paths
- The complexity can be given independently of the maximal flow.

The algorithm runs in time $O(ve^2)$

# Dinic's Algorithm

## Definitions

A **residual network** $G^R$ of network $G$ is a network which contains two edges for each edge $(v, u) \in G$:

- $(v, u)$ with capacity $c_{vu}^R = c_{vu} - f_{vu}$
- $(u, v)$ with capacity $c_{uv}^R = f_{vu}$

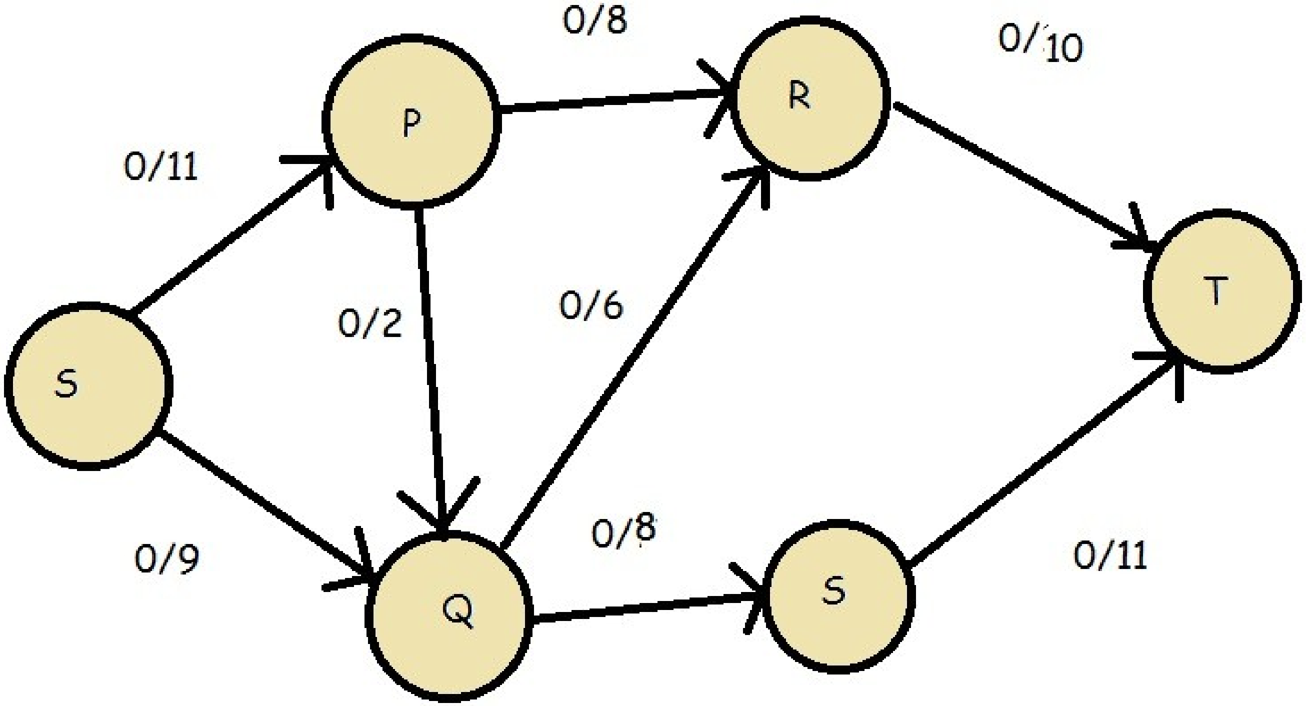
A **blocking flow** of some network is such a flow that every path from $s$ to $t$ contains at least one edge which is saturated by this flow. Note that a blocking flow is not necessarily maximal.

A **layered network** of a network $G$ is a network built in the following way. Firstly, for each vertex $v$ we calculate $level[v]$ - the shortest path (unweighted) from $s$ to this vertex using only edges with positive capacity. Then we keep only those edges $(v, u)$ for which $level[v] + 1 = level[u]$. Obviously, this network is acyclic.

# Dinic's Algorithm

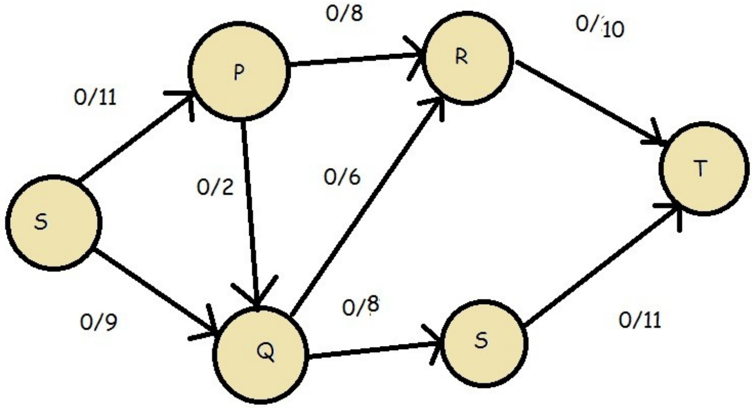
1) Set *flow*=0
2) Prepare a residual augmented graph from the initial graph
3) Prepare level graph from the residual augmented graph
4) Find blocking paths in the residual graph using the edges in level graph. Add bottleneck value/ blocking value of each blocking path to *flow*. If no path is detected, return *flow* and exit.
5) Augment the residual graph using blocking paths
6)

# Dinic's Algorithm

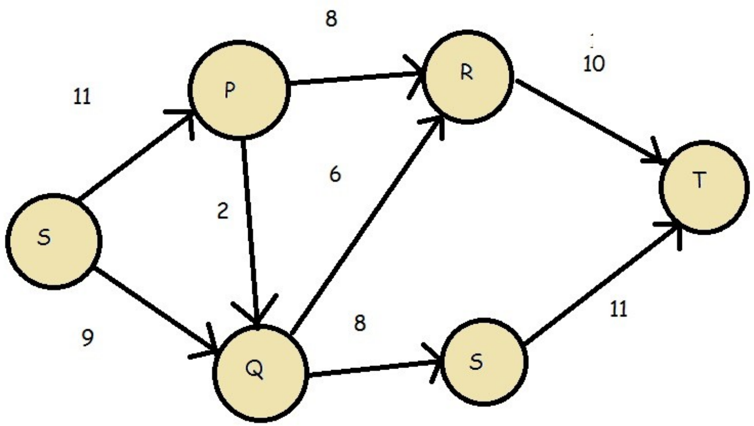



LEVEL GRAPH

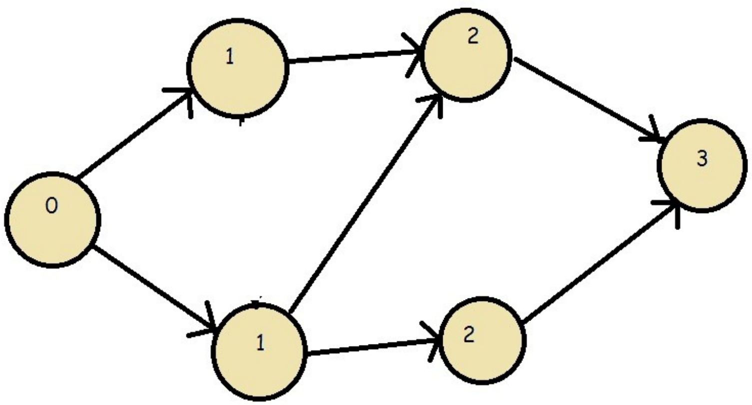
NETWORK GRAPH




RESIDUAL GRAPH



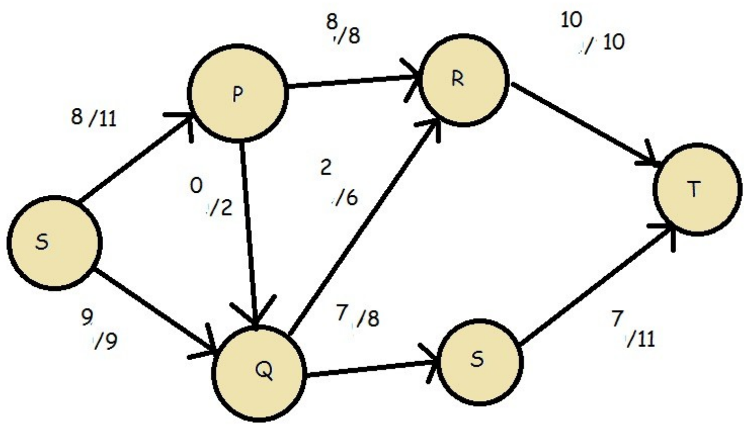

LEVEL GRAPH



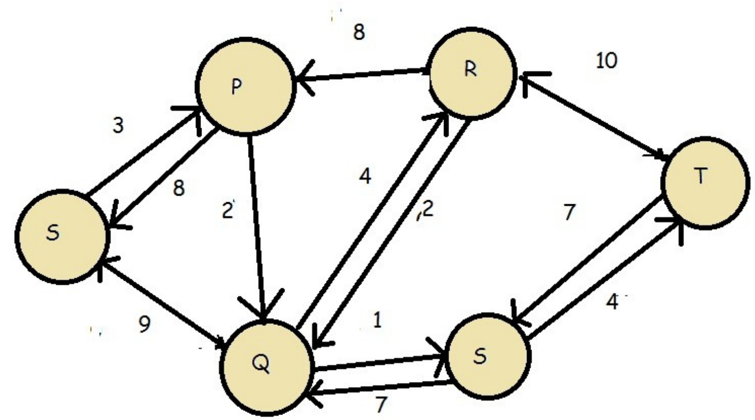

BLOCKING PATHS

| | |
|---|---|
| S - P - R - T | FLOW 8 |
| S - Q - R - T | FLOW 2 |
| S - Q - S - T | FLOW 7 |

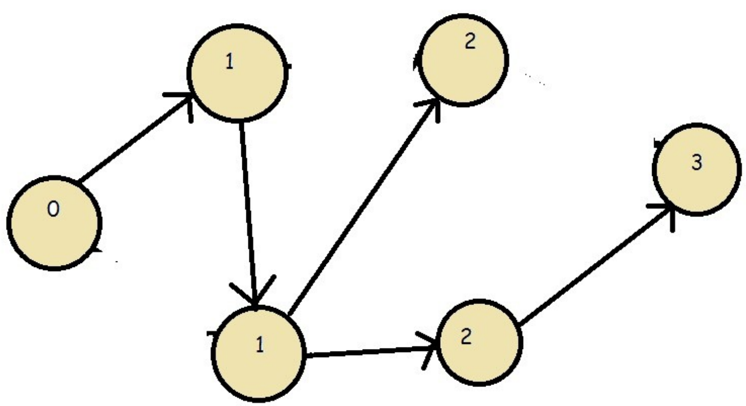
NETWORK GRAPH




RESIDUAL GRAPH



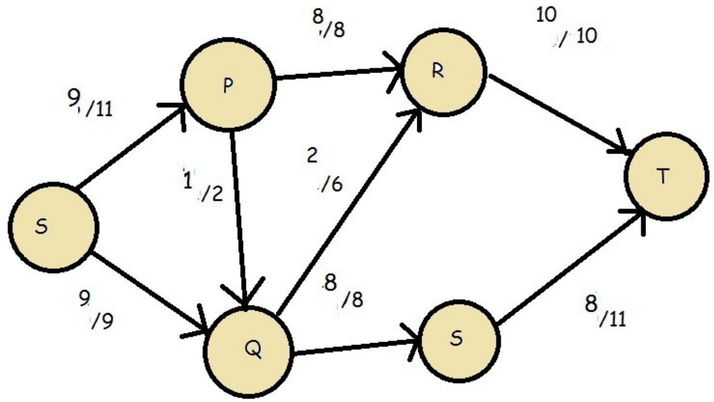

LEVEL GRAPH



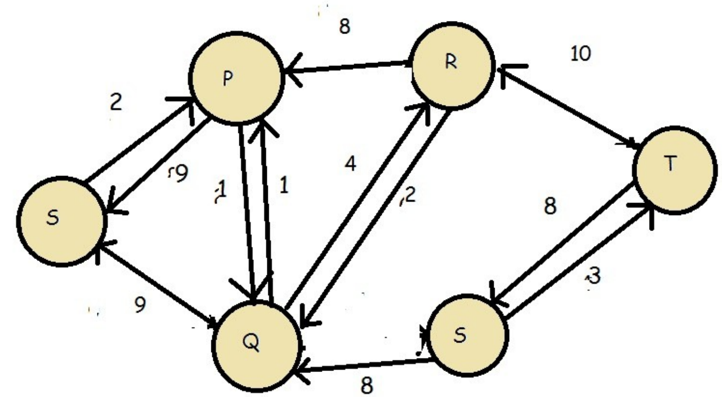

BLOCKING PATHS

S - P - Q - S - T FLOW 1

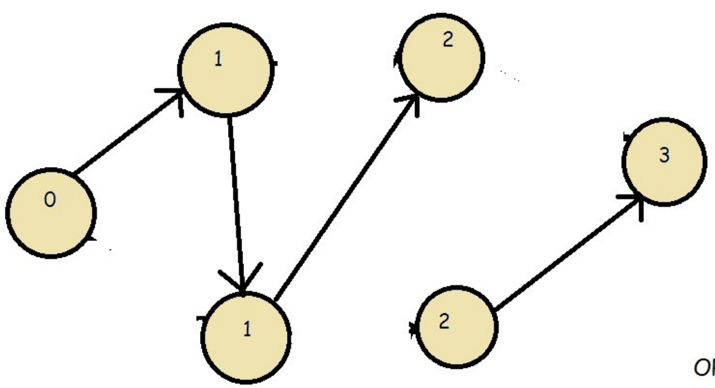
NETWORK GRAPH




RESIDUAL GRAPH




LEVEL GRAPH




BLOCKING PATHS

NO PATHS LEFT

FLOW: 18

OPENGENUS FOUNDATION

**Thank you**

**Any Question ??????????**