# ECAP770

## Advance Data Structures


**Ashwani Kumar**

Assistant Professor

# Learning Outcomes

After this lecture, you will be able to

- Understand network flow problems

# Network Flow Problems

- In graph theory, a flow network is a directed graph involving a source(s) and a sink(t) and several other nodes connected with edges.
- Each edge has an individual capacity which is the maximum limit of flow that edge could allow.

# Network Flow Problems

- Problem1: Given a flow network G = (V, E), the maximum flow problem is to find a flow with maximum value.
- Problem 2: The multiple source and sink maximum flow problem is similar to the maximum flow problem, except there is a set {s1,s2,s3…….sn} of sources and a set {t1,t2,t3……….tn} of sinks.

# Conditions for flow

- For any non-source and non-sink node, the input flow is equal to output flow.
- For any edge($E_i$) in the network, $0 \leq flow(E_i) \leq capacity(E_i)$.
- Total flow out of the source node is equal total to flow in to the sink node.
- Net flow in the edges follows skew symmetry

# Network Flow Problems

- Problem: Maximize the total amount of flow from s to t
- subject to two constraints
  - Flow on edge e doesn't exceed c(e)
  - For every node $v \neq s, t$, incoming flow is equal to outgoing flow

# Types of network flow problems

- Minimum-cost flow problem: in which the edges have costs as well as capacities and the goal is to achieve a given amount of flow (or a maximum flow) that has the minimum possible cost.

# Types of network flow problems

- Multi-commodity flow problem:  in which one must construct multiple flows for different commodities whose total flow amounts together respect the capacities.

# Types of network flow problems

- Nowhere-zero flow: a type of flow studied in combinatorics in which the flow amounts are restricted to a finite set of nonzero values.
- Maximum flow problem.

# Algorithms

- The Ford–Fulkerson algorithm, a greedy algorithm for maximum flow that is not in general strongly polynomial
- Dinic's algorithm, a strongly polynomial algorithm for maximum flow
- The Edmonds–Karp algorithm, a faster strongly polynomial algorithm for maximum flow

# Algorithms

- The network simplex algorithm, a method based on linear programming but specialized for network flow
- The out-of-kilter algorithm for minimum-cost flow
- The push–relabel maximum flow algorithm, one of the most efficient known techniques for maximum flow

# Ford-Fulkerson Algorithm

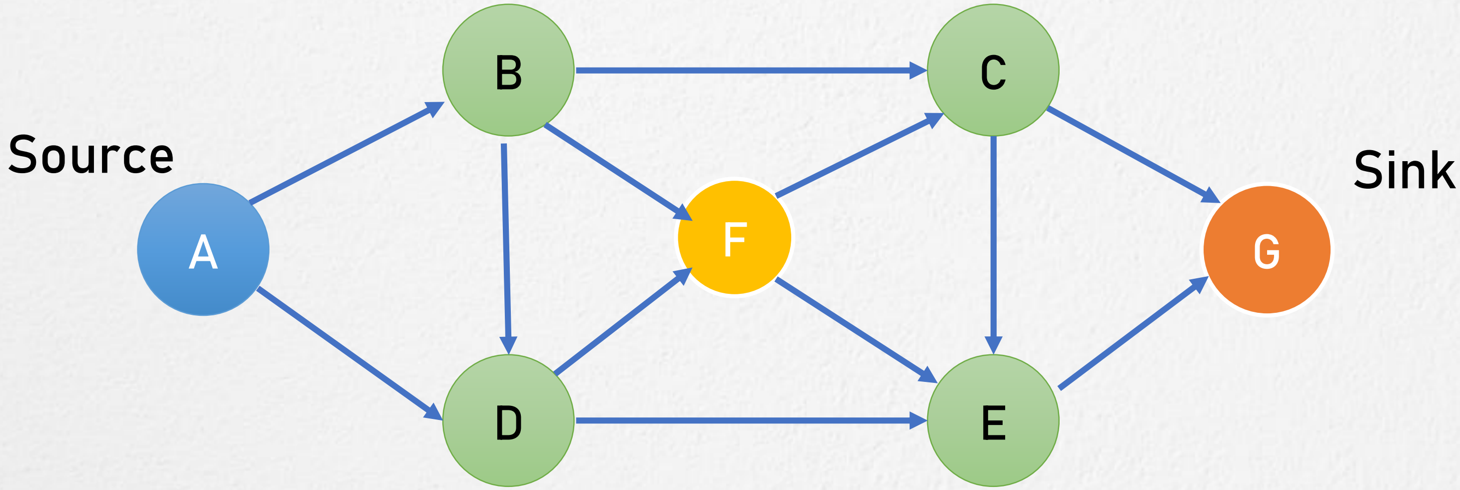
- It was developed by L. R. Ford, Jr. and D. R. Fulkerson in 1956
- A simple and practical max-flow algorithm
- Objective: find valid flow paths until there is none left, and add them up.

# Terminology: Ford-Fulkerson Algorithm

- Source
- Sink
- Bottleneck capacity
- Flow
- Augmenting path
- Residual capacity

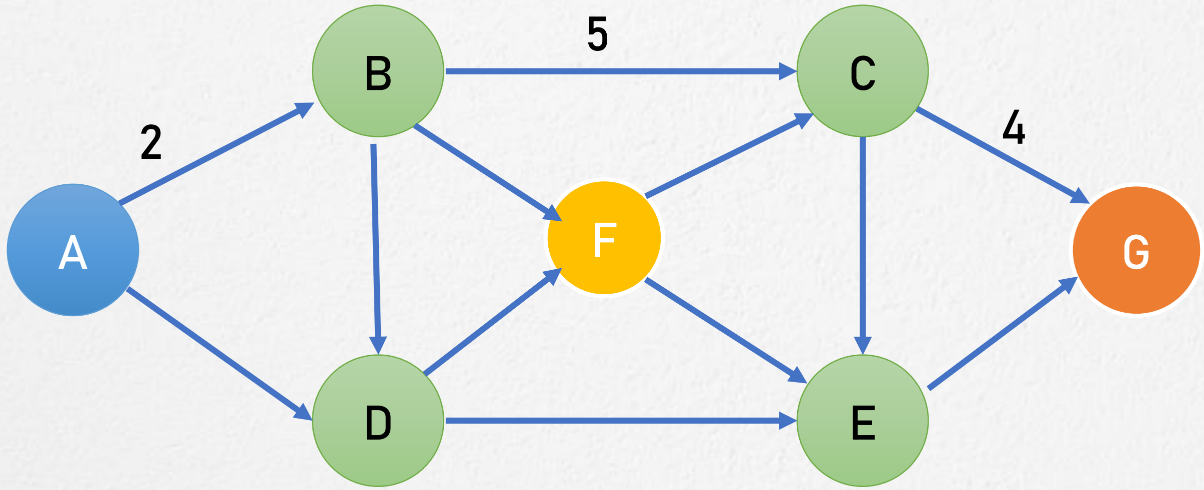
# Terminology: Ford-Fulkerson Algorithm

- The source vertex has all outward edges, no inward edges.
- Sink have all inward edges, no outward edges.



# Terminology: Ford-Fulkerson Algorithm

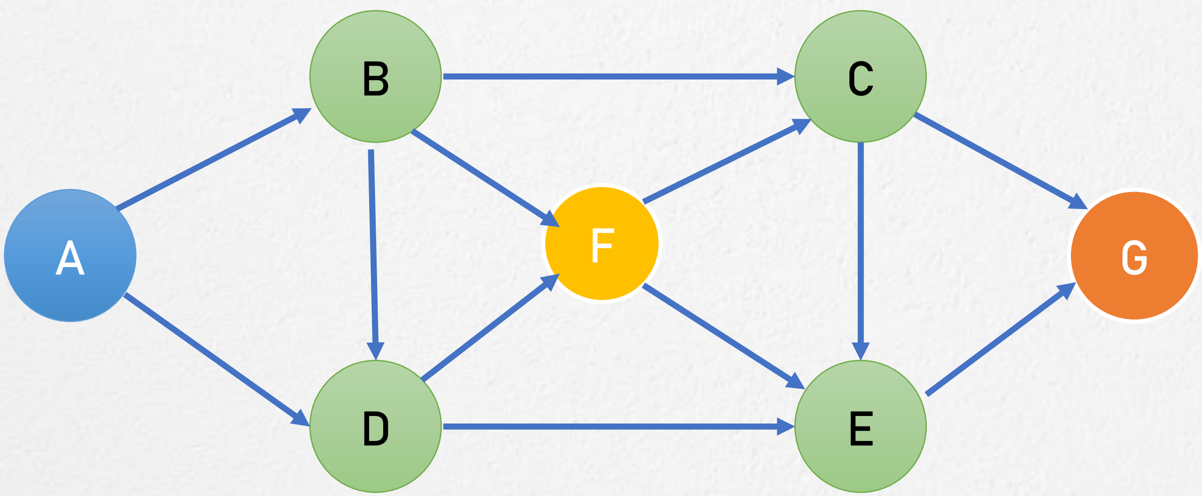
- Bottleneck capacity of a path is the minimum capacity of any edge on the path.




A-B-C-G = 2

# Terminology: Ford-Fulkerson Algorithm

- An augmenting path is a simple path from source to sink which do not include any cycles and that pass only through positive weighted edges.

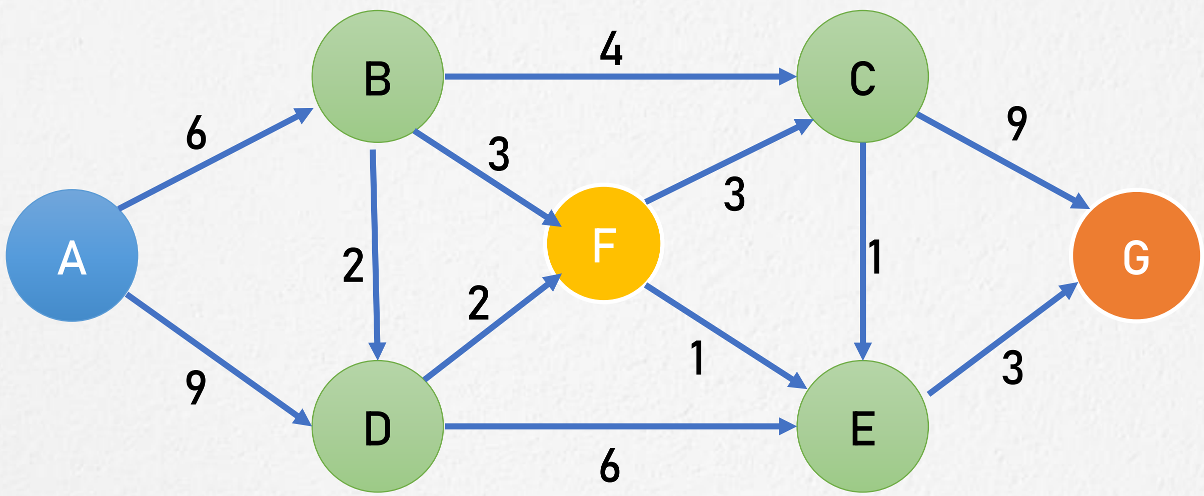



A-B-C-G and A-D-E-C-G

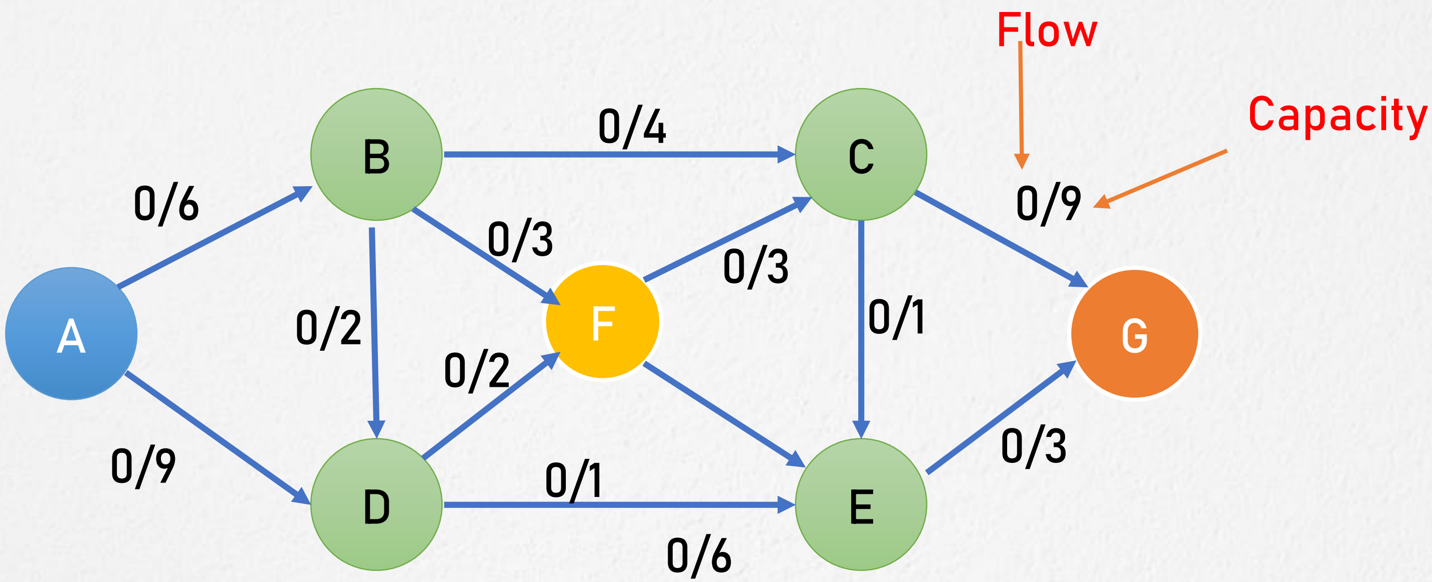
# Terminology: Ford-Fulkerson Algorithm

- Residual capacity: which is equal to original capacity of the edge minus current flow. Residual capacity is basically the current capacity of the edge.

# Ford-Fulkerson Algorithm



# Ford-Fulkerson Algorithm



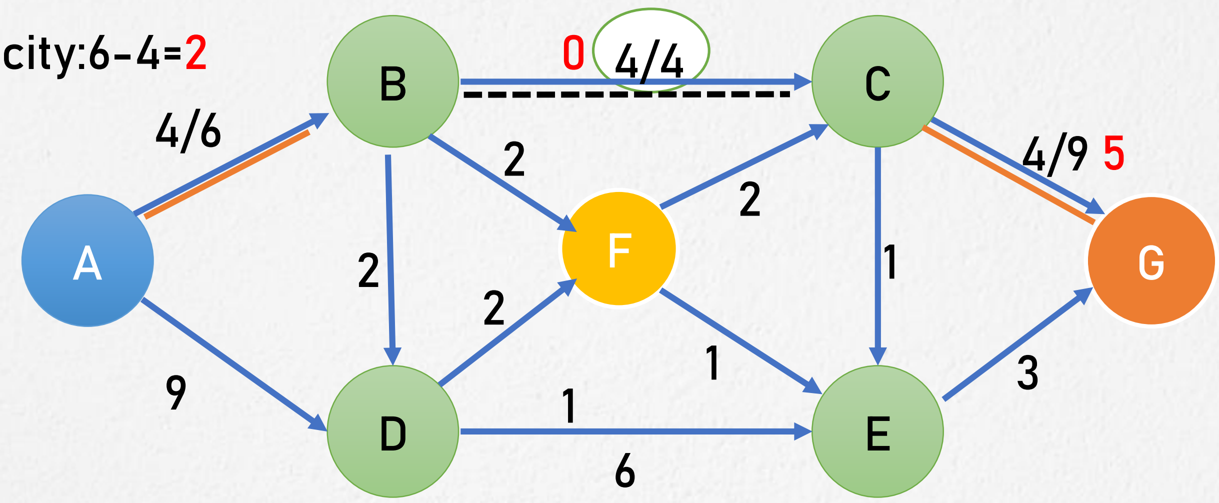

Flow = 0

# Ford-Fulkerson Algorithm

Path: A-B-C-G

Flow = 4

Residual capacity:6-4=2

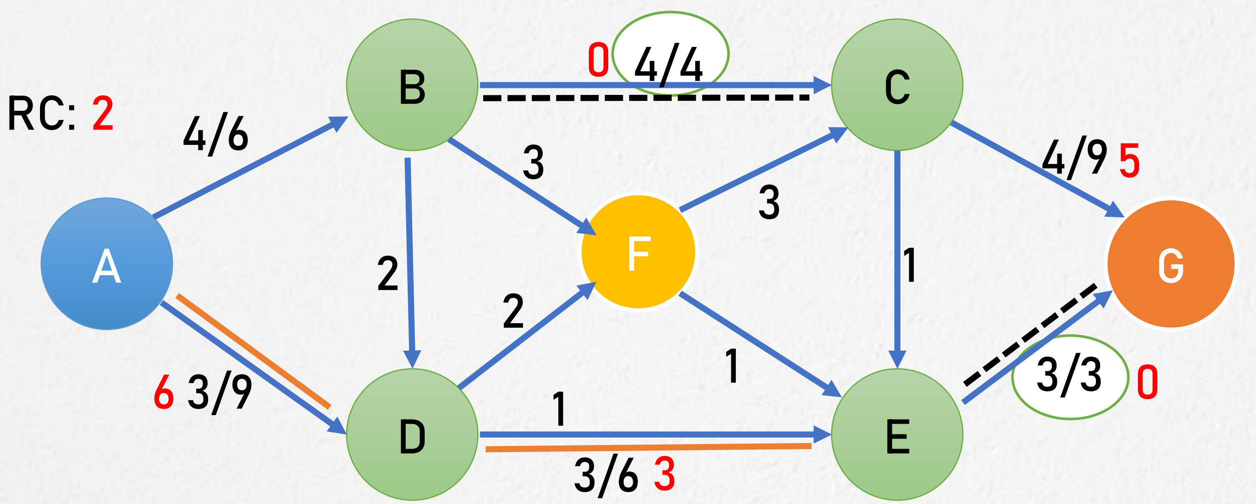



| Augmenting path | Bottleneck capacity |
|---|---|
| A-B-C-G | 4 |
| | |

# Ford-Fulkerson Algorithm

Path: A-D-E-G

Flow = 4+3

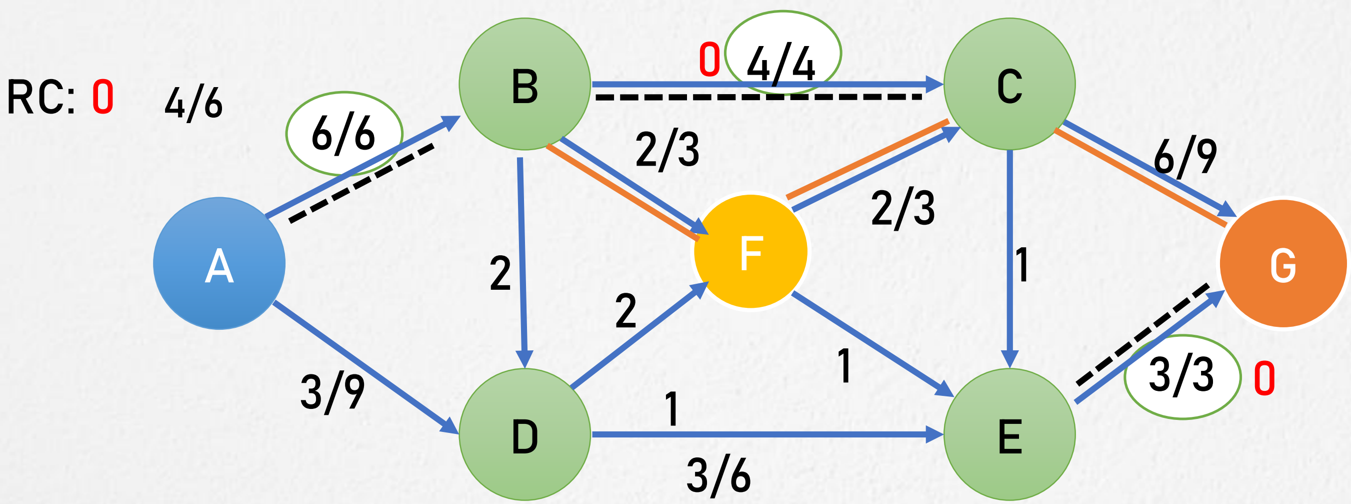



| Augmenting path | Bottleneck capacity |
|---|---|
| A-B-C-G | 4 |
| A-D-E-G | 3 |

# Ford-Fulkerson Algorithm

Path: A-B-F-C-G

Flow = 4+3+2

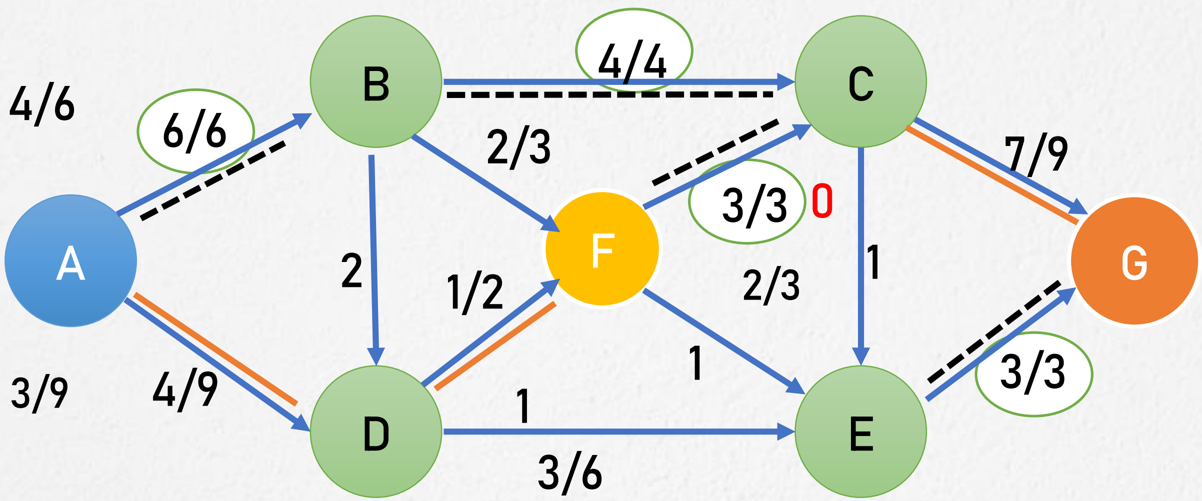



| Augmenting path | Bottleneck capacity |
|---|---|
| A-B-C-G | 4 |
| A-D-E-G | 3 |
| A-B-F-C-G | 2 |

# Ford-Fulkerson Algorithm

Path: A-D-F-C-G

Flow = 4+3+2+1 = 10




| Augmenting path | Bottleneck capacity |
|---|---|
| A-B-C-G | 4 |
| A-D-E-G | 3 |
| A-B-F-C-G | 2 |
| A-D-F-C-G | 1 |

# Ford-Fulkerson Applications

- Circulation with demands
- Water distribution pipeline
- Bipartite matching problem

# Complexity

- Time Complexity: Time complexity of the above algorithm is O(max_flow * E).

That's all for now…