Data Structure and Algorithm Design

ME/MSc (Computer) – Pokhara University

Prepared by:

Assoc. Prof. Madan Kadariya (NCIT)

Chapter 2:
Advanced Data Structures and Algorithms (11 hrs)

Outline

1. Advanced Trees
   - B and B+ Trees
2. Advanced Graph
   - Planar Graphs and its applications in Geographic and Circuit Layout Design
3. Network Flow Algorithms
   - Ford-Fulkerson Algorithm
   - Edmonds-Karp Algorithm
4. Advanced Shortest Path Algorithms
   - Bellman-Ford Algorithm
   - Johnson's algorithm
5. Graphical Data Structures
   - Quadtrees, KD Trees, R-Trees
6. Computational Geometry
   - Convex Hull and Voronoi Diagrams
7. Case Studies in Red-Black Tree and Its Applications, Applications of Directed Acyclic Graph, Applications of Minimum Spanning Tree in Network Design

Network Flow Algorithms

1. Ford-Fulkerson Algorithm
2. Edmonds-Karp Algorithm

Network Flow Algorithms

Network Flow

- Network flow is a type of network optimization problem
  - Arise in many different contexts :
    - Networks: routing as many packets as possible on a given network
    - Transportation: sending as many trucks as possible, where roads have limits on the number of trucks per unit time
    - Bridges: destroying some bridges to disconnect source(s) from sink(t), while minimizing the cost of destroying the bridges
- Max flow problem is represented by a directed graph with a source node ***s*** and a sink node ***t***, and each edge has a capacity that represents the maximum amount of flow that can be sent through it.
- The goal is to find the maximum amount of flow that can be sent from source (***s***) to sink(***t***), while respecting the capacity constraints on the edges.

Network Flow Algorithms

Advantages:

1. The max flow problem is a flexible and powerful modeling tool that can be used to represent a wide variety of real-world situations.
2. The max flow problem has many interesting theoretical properties and connections to other areas of mathematics, such as linear programming and combinatorial optimization.

Disadvantages:

1. In some cases, the max flow problem can be difficult to solve efficiently, especially if the graph is very large or has complex capacity constraints.
2. The max flow problem may not always provide a unique or globally optimal solution, depending on the specific problem instance and algorithm used.
3. Maximum flow problems involve finding a feasible flow through a single-source, single-sink flow network that is maximum.

Network Flow Algorithms

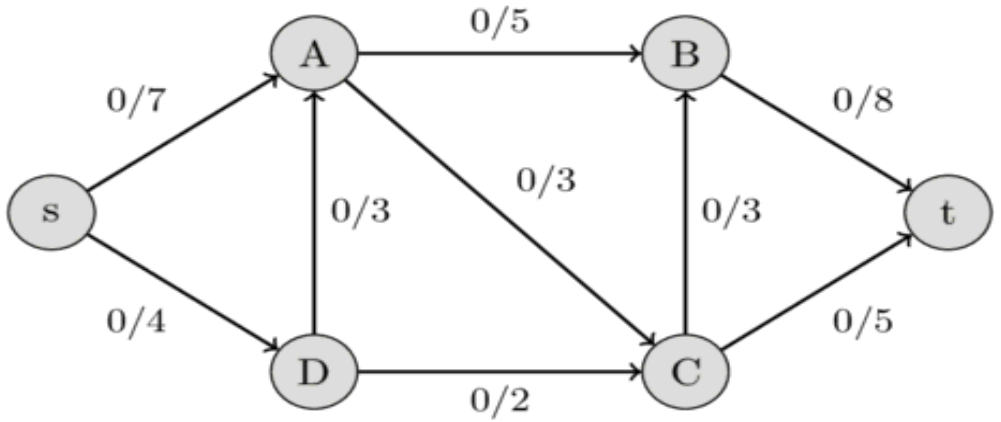
- A **network** is a directed graph G with vertices V and edges E combined with a function c, which assigns each edge $e \in E$ a non-negative integer value, the **capacity** of e. Such a network is called a **flow network**, if we additionally label two vertices, one as **source** and one as **sink**.
- A **flow** in a flow network is function f, that again assigns each edge e a non-negative integer value, namely the flow. The function has to fulfill the following two conditions:
  - The flow of an edge cannot exceed the capacity. $f(e) \leq c(e)$
  - And the sum of the incoming flow of a vertex u has to be equal to the sum of the outgoing flow of u except in the source and sink vertices. $\sum_{(v,u)\in E} f((v,u)) = \sum_{(u,v)\in E} f((u,v))$
- The source vertex s only has an outgoing flow, and the sink vertex t has only incoming flow.

$$\sum_{(s,u)\in E} f((s,u)) = \sum_{(u,t)\in E} f((u,t))$$

Network Flow Algorithms

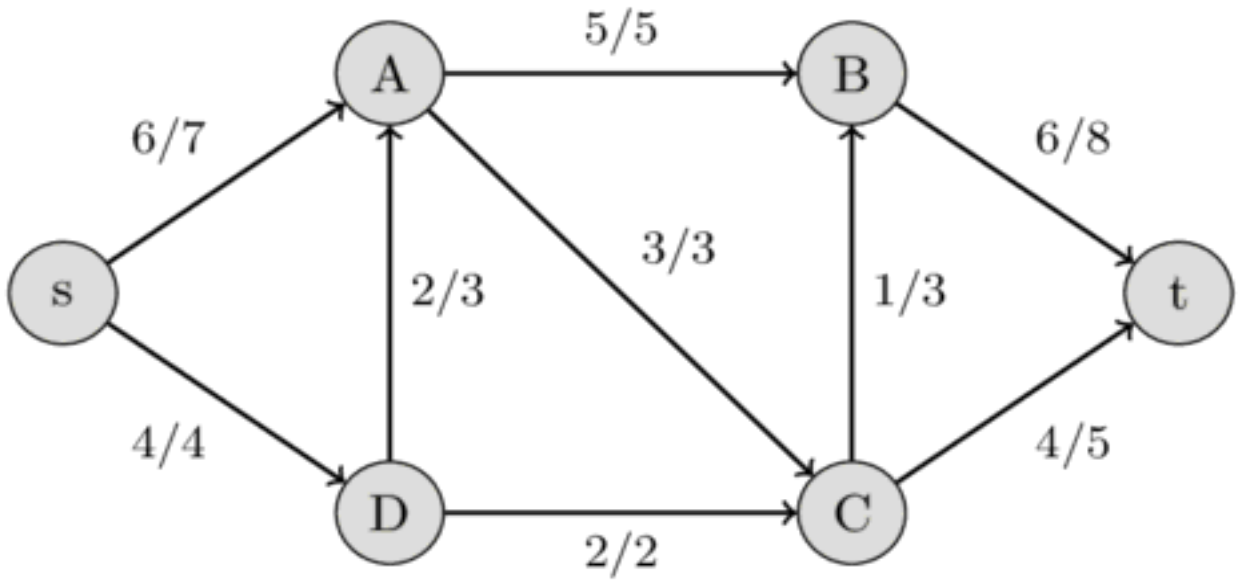
Analogy

- We represent edges as water pipes, the capacity of an edge is the maximal amount of water that can flow through the pipe per second, and the flow of an edge is the amount of water that currently flows through the pipe per second. This motivates the first flow condition. There cannot flow more water through a pipe than its capacity. The vertices act as junctions, where water comes out of some pipes, and then, these vertices distribute the water in some way to other pipes. This also motivates the second flow condition. All the incoming water has to be distributed to the other pipes in each junction. It cannot magically disappear or appear. The source s is origin of all the water, and the water can only drain in the sink t.
- Example of *flow network*. The first value of each edge represents the flow, which is initially 0, and the second value represents the capacity.



Network Flow Algorithms

- The value of the flow of a network is the sum of all the flows that get produced in the source s, or equivalently to the sum of all the flows that are consumed by the sink t.
- A **maximal flow** is a flow with the maximal possible value. Finding this maximal flow of a flow network is the problem that we want to solve.
- In the visualization with water pipes, the problem can be formulated in the following way: how much water can we push through the pipes from the source to the sink?
- The following image shows the maximal flow in the flow network.



Network Flow Algorithms

Ford-Fulkerson Algorithm

- The Ford-Fulkerson algorithm works by looking for a path with available capacity from the source to the sink (called an *augmented path*), and then sends as much flow as possible through that path.
- The Ford-Fulkerson algorithm continues to find new paths to send more flow through until the maximum flow is reached.
- **Residual graph**: A graph that represents the remaining capacity on each edge after the current flow has been sent through the network. [*original capacity – flow sent*]
- **Augmenting path:** A simple path from the source node to the sink node in the residual graph that has available capacity on all its edges. Augmenting path can be done in two ways:
  i. Non- Full forward edges
  ii. Non-Empty backward edges
- **Bottleneck capacity:** The minimum capacity of all the edges in an augmenting path.

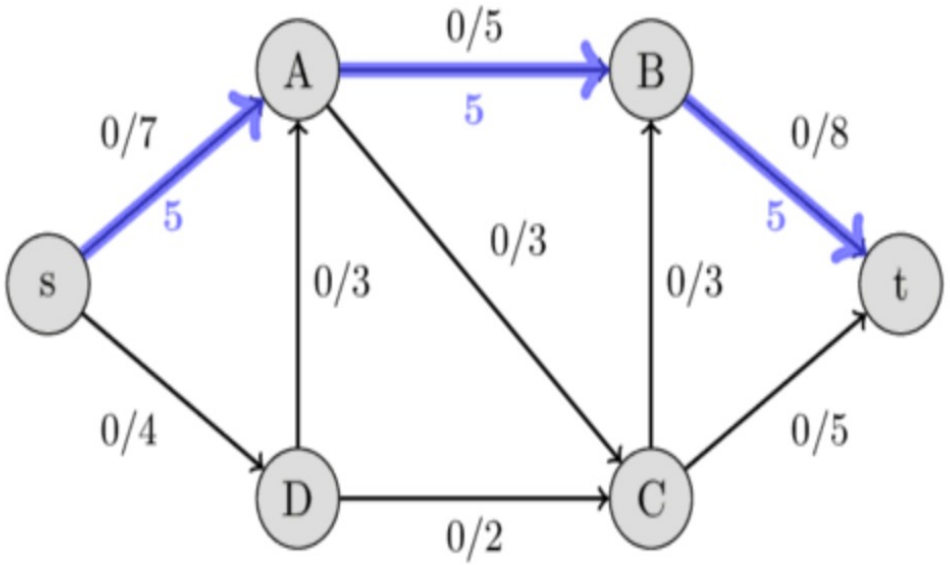
Network Flow Algorithms

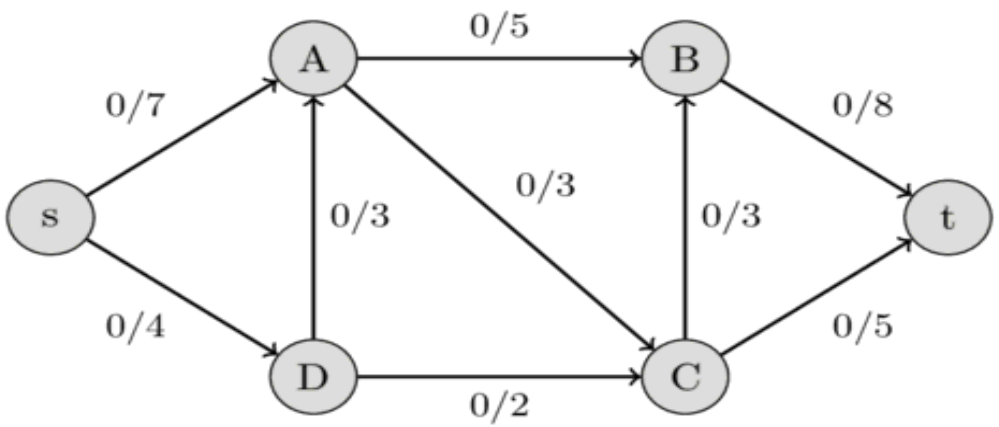
Ford-Fulkerson Algorithm

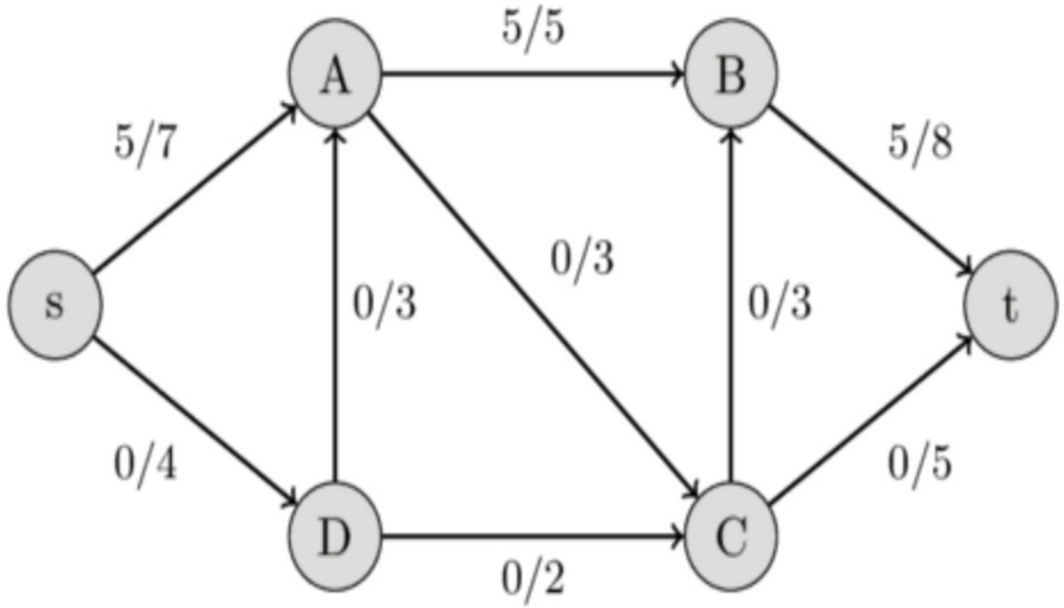
- **Formal Definition**: A **residual capacity** of a directed edge is the capacity minus the flow. It should be noted that if there is a flow along some directed edge (u, v), then the reversed edge has capacity 0 and we can define the flow of it as $f((v, u)) = -f((u, v))$. This also defines the residual capacity for all the reversed edges.
- We can create a **residual network** from all these edges, which is just a network with the same vertices and edges, but we use the residual capacities as capacities.
- Working Principle of Ford-Fulkerson Method:
  - First, we set the flow of each edge to zero. Then we look for an **augmenting path** from s to t.
  - An augmenting path is a simple path in the residual graph where residual capacity is positive for all the edges along that path. If such a path is found, then we can increase the flow along these edges. We keep on searching for augmenting paths and increasing the flow. Once an augmenting path doesn't exist anymore, the flow is maximal.
  - Let C be the smallest residual capacity of the edges in the path. Then we increase the flow in the following way: we update $f((u, v)) \mathrel{+}= C$ and $f((v, u)) \mathrel{-}= C$ for every edge (u, v) in the path.

Example: Ford-Fulkerson

Augmenting path s - A - B - t with the residual capacities 7, 5, and 8. Their minimum is 5, therefore we can increase the flow along this path by 5. This gives a flow of 5 for the network.



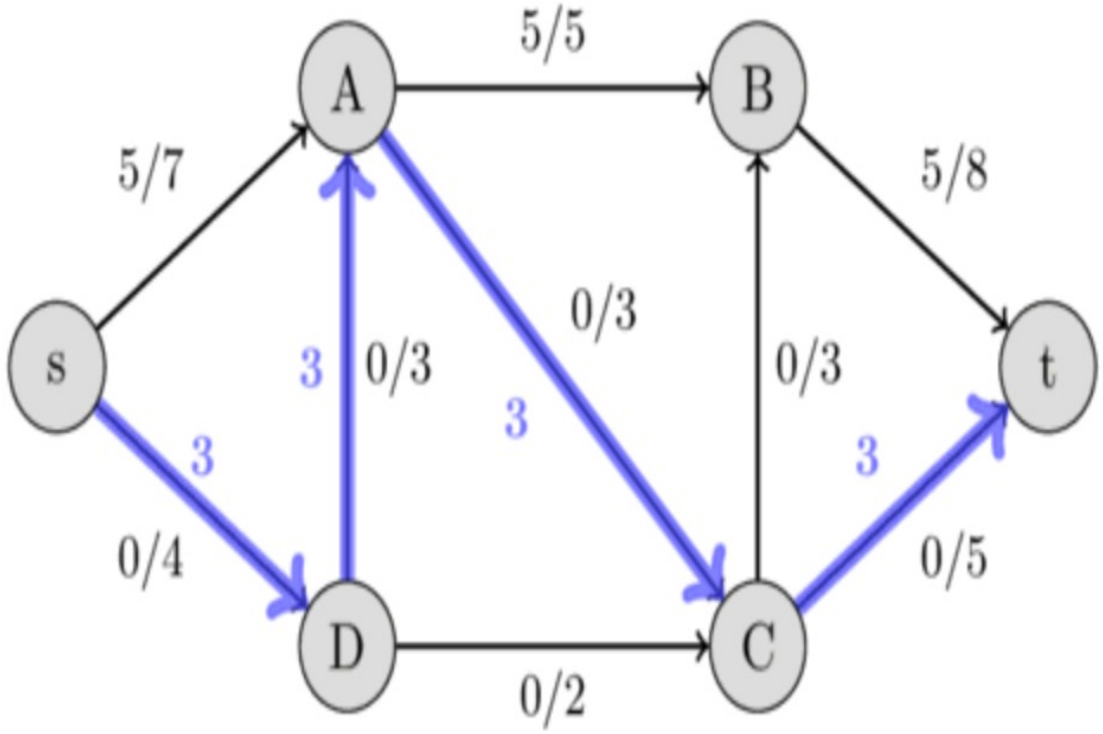


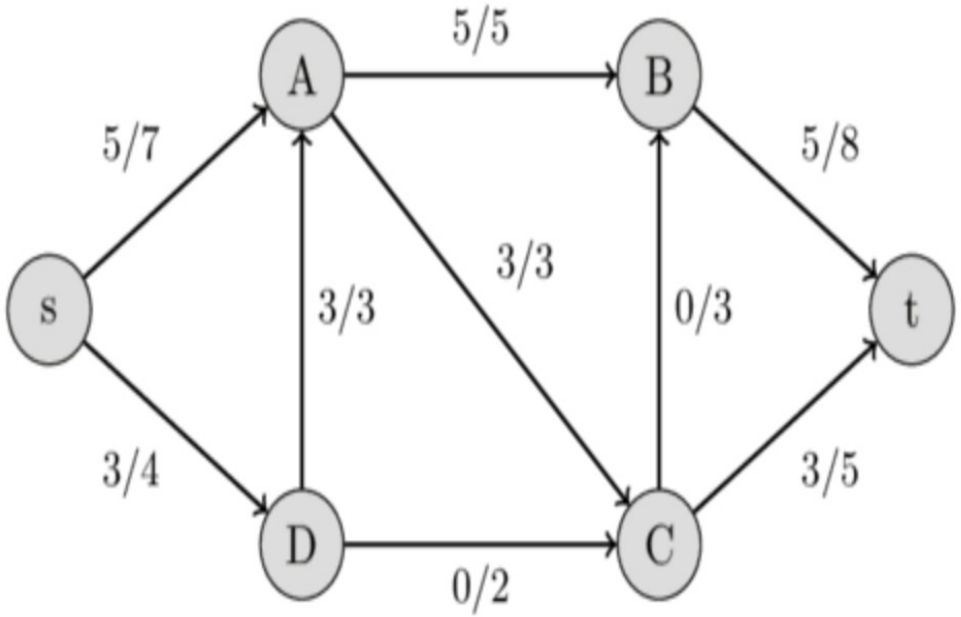


Network Flow Algorithms

Example: Ford-Fulkerson

Another augmenting path: *s-D-A-C-t* with the residual capacities 4, 3, 3, and 5.
Therefore we can increase the flow by 3 and we get a flow of 8 for the network.

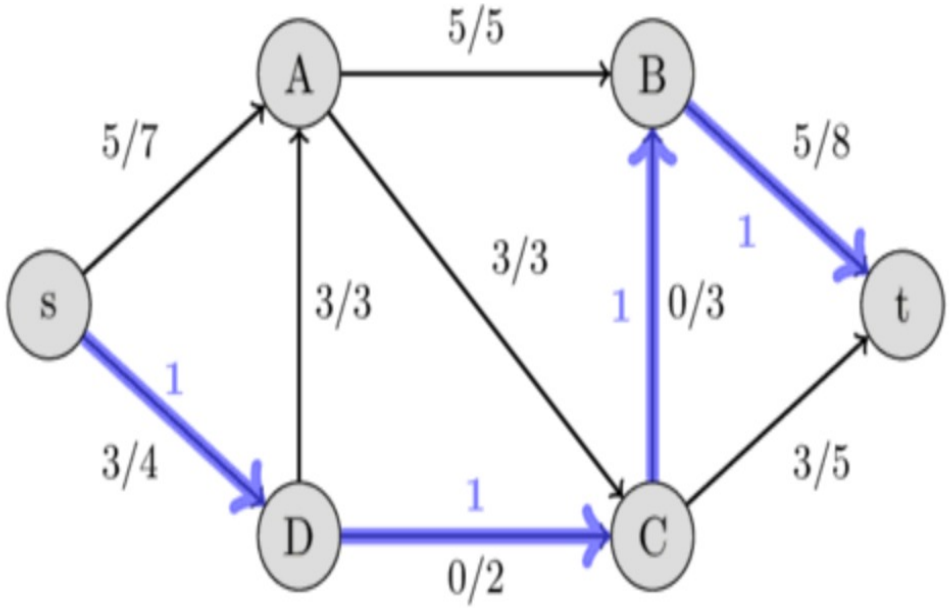


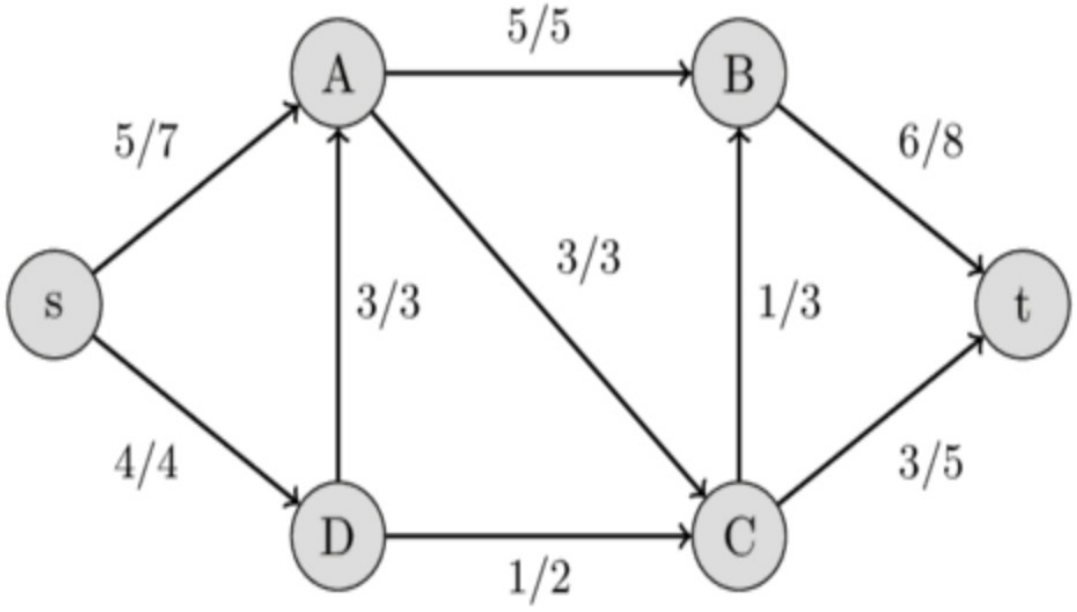


Network Flow Algorithms

Example: Ford-Fulkerson

Another Augmenting path *s-D-C-B-t* with the residual capacities 1, 2, 3, and 3, and hence, we increase the flow by 1. Flow = 5+3+1 = 9

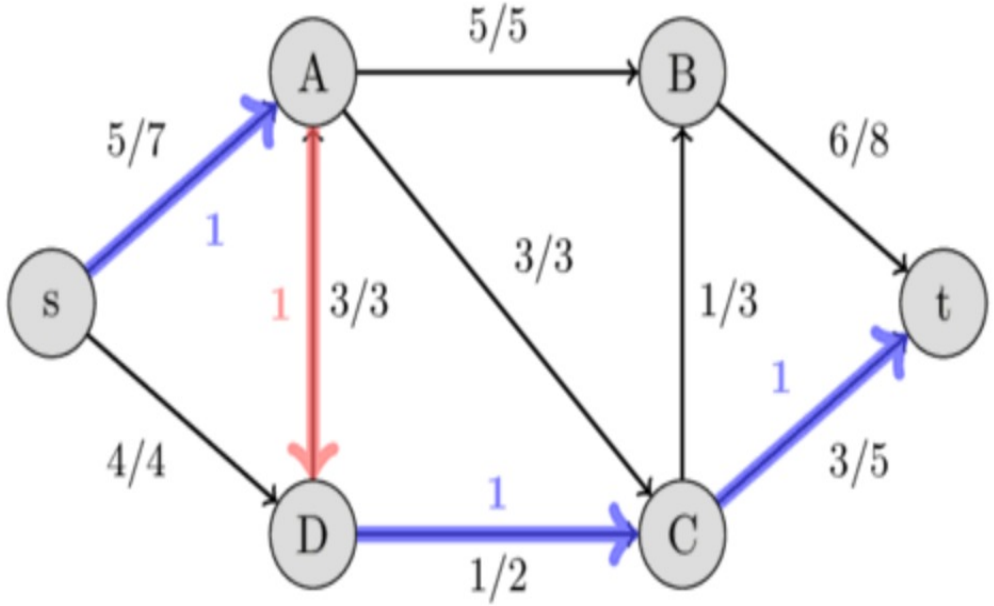


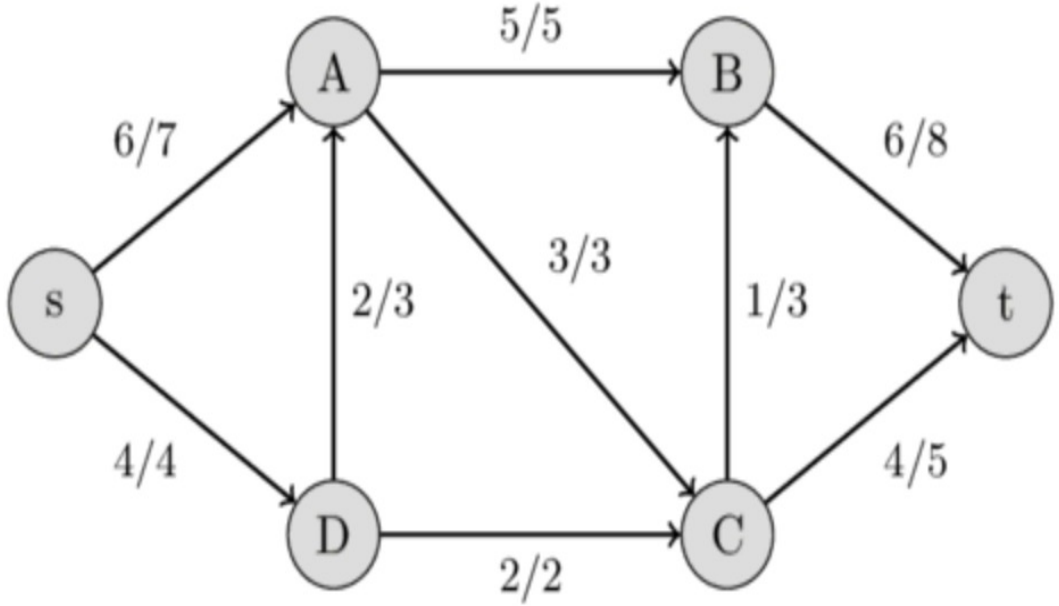


Network Flow Algorithms

Example: Ford-Fulkerson

This time we find the augmenting path *s-A-D-C-t* with the residual capacities 2, 3, 1, and 2. We can increase the flow by 1. But this path is very interesting. It includes the reversed edge *(A, D)*. In the original flow network, we are not allowed to send any flow from *A* to *D*. But because we already have a flow of 3 from *D* to *A*, this is possible. The intuition of it is the following: Instead of sending a flow of 3 from *D* to *A*, we only send 2 and compensate this by sending an additional flow of 1 from *s* to *A*, which allows us to send an additional flow of 1 along the path *D-C-t*. **Flow = 5+3+1+1=10**





Network Flow Algorithms

Example: Ford-Fulkerson (Conclusion)

Now, it is impossible to find an augmenting path between s and t, therefore this flow of 10 is the maximal possible.

It should be noted, that the Ford-Fulkerson method doesn't specify a method of finding the augmenting path. Possible approaches are using DFS or BFS which both work in $O(E)$.
If all the capacities of the network are integers, then for each augmenting path the flow of the network increases by at least 1. So DFS runs **F** times.
Therefore, the complexity of Ford-Fulkerson is $O(EF)$, where **F** is the maximal flow of the network.

Pseudocode of Fold-Fulkerson Method

```
flow = 0
for each edge (u, v) in G:
        flow(u, v) = 0
while there is a path, p, from s -> t in residual network G_f:
        residual_capacity(p) = min(residual_capacity(u, v) : for (u, v) in p)
        flow = flow + residual_capacity(p)
        for each edge (u, v) in p:
                if (u, v) is a forward edge:
                        flow(u, v) = flow(u, v) + residual_capacity(p)
                else:
                        flow(u, v) = flow(u, v) - residual_capacity(p)
return flow
```

Network Flow Algorithms

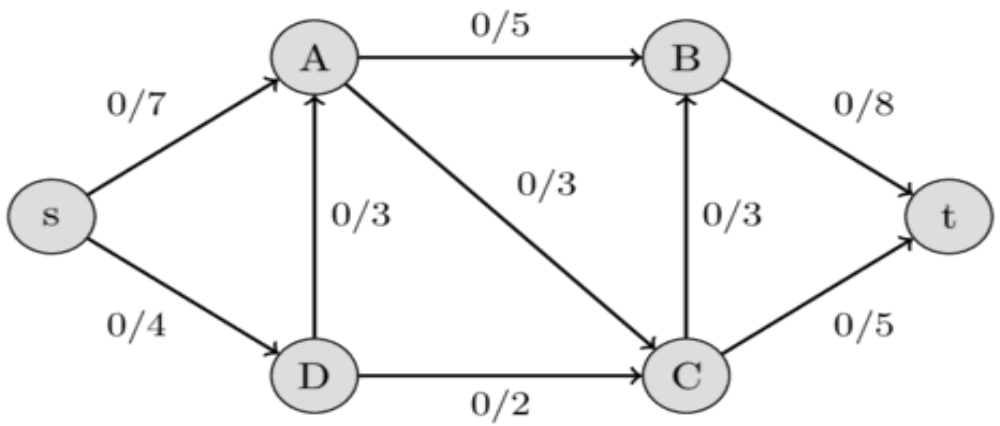
Edmonds-Karp algorithm

- Edmonds-Karp algorithm is just an implementation of the Ford-Fulkerson method that uses BFS for finding augmenting paths.
- The algorithm was first published by Yefim Dinitz in 1970, and later independently published by Jack Edmonds and Richard Karp in 1972.
- The complexity can be given independently of the maximal flow.
- The algorithm runs in $O(VE^2)$ time.
- This means Edmonds-Karp does not depend on the maximum flow, like Ford-Fulkerson does, but on how many vertices and edges we have.
- The intuition is, that every time we find an augmenting path one of the edges becomes saturated, and the distance from the edge to s will be longer if it appears later again in an augmenting path.
- The length of the simple paths is bounded by V.

Network Flow Algorithms

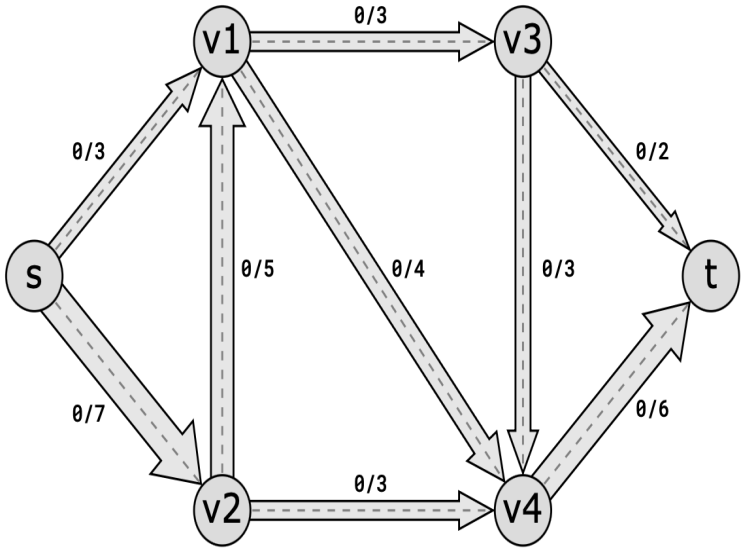
Example: Edmonds-Karp algorithm

Augmenting Path	Bottleneck Capacity
s-A-B-t	5
s-D-C-t	2
s-A-C-t	2
s-D-A-C-t	1
Total Flow	10

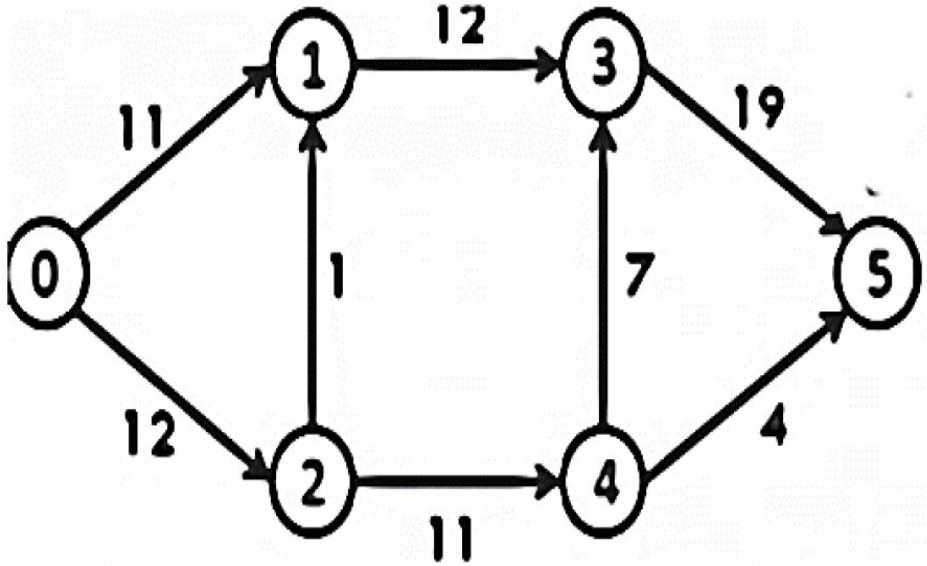


Try Yourself

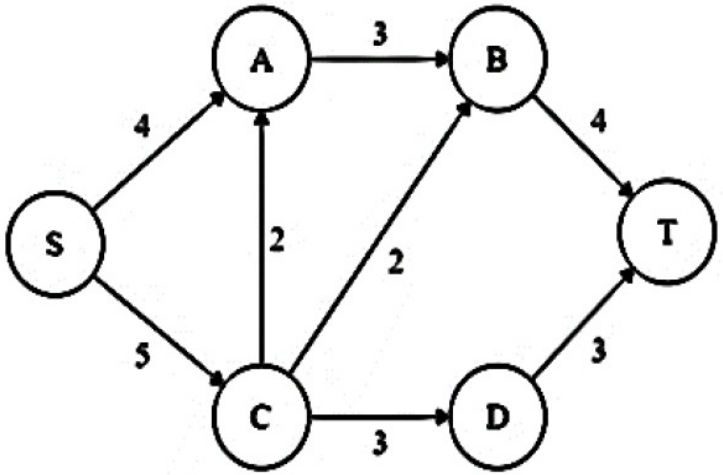
Find the maximum flow in a given network using Ford-Fulkerson and Edmond Karp Algorithm



Ans : 8



Ans : 23



Ans : 7

Try Python Implementation of Edmond Karp Algorithm from :
https://www.w3schools.com/dsa/dsa_algo_graphs_edmondskarp.php