# Graph Theory

Dr. Irfan Yousuf
Department of Computer Science (New Campus)
UET, Lahore

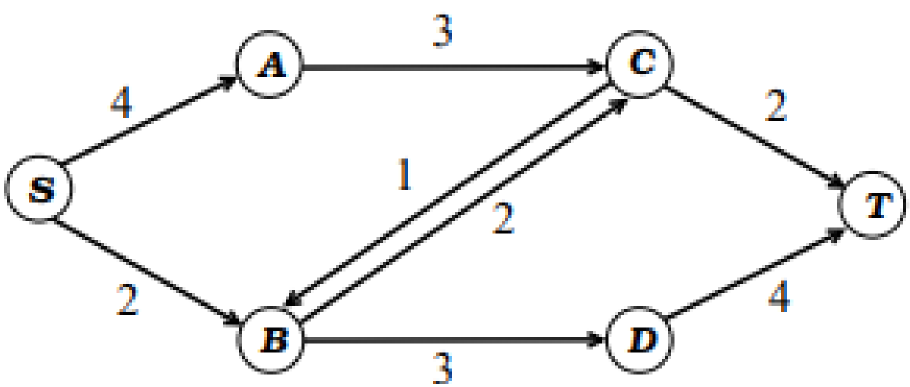
(Lecture # 19; March 29, 2023)

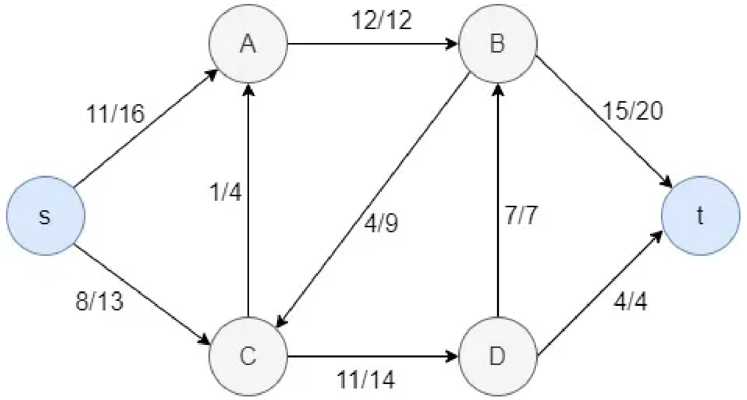
# Outline

- Flow Networks

# Flow Networks

- In graph theory, a flow network (also known as a transportation network) is a directed graph where each edge has a capacity, and each edge receives a flow.
- The amount of flow on an edge cannot exceed the capacity of the edge.



# Flow Networks

- Flow Network is a directed graph that is used for modeling material Flow.
- There are two different vertices; one is a <u>source</u> which produces material at some steady rate, and another one is <u>sink</u> which consumes the content at the same constant speed.
- The flow of the material at any mark in the system is the rate at which the element moves.

- Some real-life problems like the flow of liquids through pipes, the current through wires and delivery of goods can be modeled using flow networks.

# Flow Networks

**Definition:** A Flow Network is a directed graph $G = (V, E)$ such that

1. 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 $c(u, v) = 0$.
2. There are two distinguishing points, the source s, and the sink t;
3. For every vertex $v \in V$, there is a path from s to t containing v.

# Flow Networks

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) ≤ 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

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

1. **Capacity Constraint** makes sure that the flow through each edge is not greater than the capacity.
2. **Skew Symmetry** means that the flow from u to v is the negative of the flow from v to u.
3. The flow-conservation property says that the total net flow out of a vertex other than the source or sink is 0. In other words, the amount of flow into a v is the same as the amount of flow out of v for every vertex v ∈ V - {s, t}

# Network Flow Problem: Definition

- We are given a directed graph G, a start node s, and a sink node t.
- Each edge e in G has an associated non-negative capacity c(e), where for all non-edges it is implicitly assumed that the capacity is 0.
- Our goal is to push as much flow as possible from s to t in the graph.
- The rules are that
  - no edge can have flow exceeding its capacity,
  - and for any vertex except for s and t, the flow into the vertex must equal the flow out from the vertex.

# Network Flow Problems: Types

- In combinatorial optimization, network flow problems are a class of computational problems in which the input is a flow network (a graph with numerical capacities on its edges), and the goal is to construct a flow, numerical values on each edge that respect the capacity constraints and that have incoming flow equal to outgoing flow at all vertices except for source and sink.
  - The **maximum flow problem**, in which the goal is to maximize the total amount of flow out of the source terminals and into the sink terminals.

# Maximum Flow Problem

- In the max flow problem, we have 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 s to t, while respecting the capacity constraints on the edges.
- One common approach to solving the max flow problem is the Ford-Fulkerson algorithm, which is based on the idea of augmenting paths.

# Residual Graphs

- It is a graph, that has the same vertices as the original network with **one or two edges** for each edge in the original network.
- And it indicates the additional possible flow through the network. For each edge, we’ll calculate the additional flow as follows.

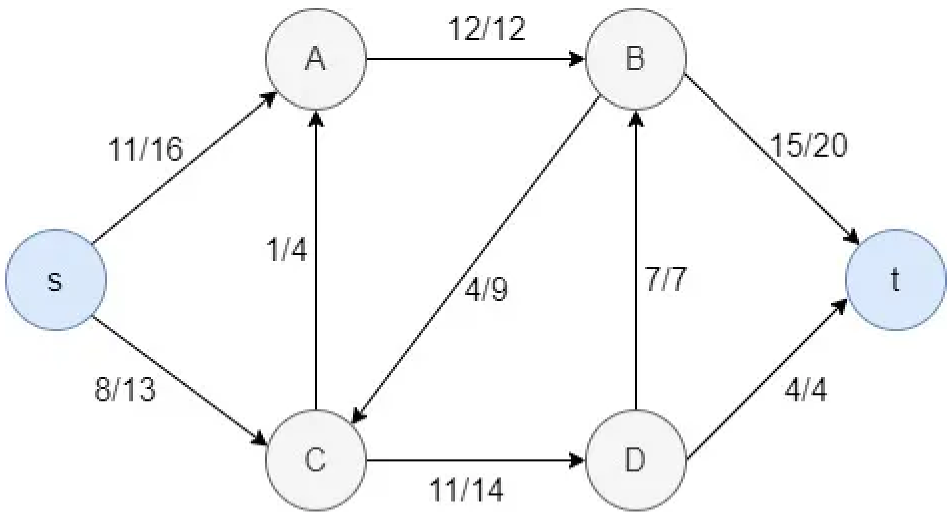
Additional flow = edge’s capacity — the flow of the edge

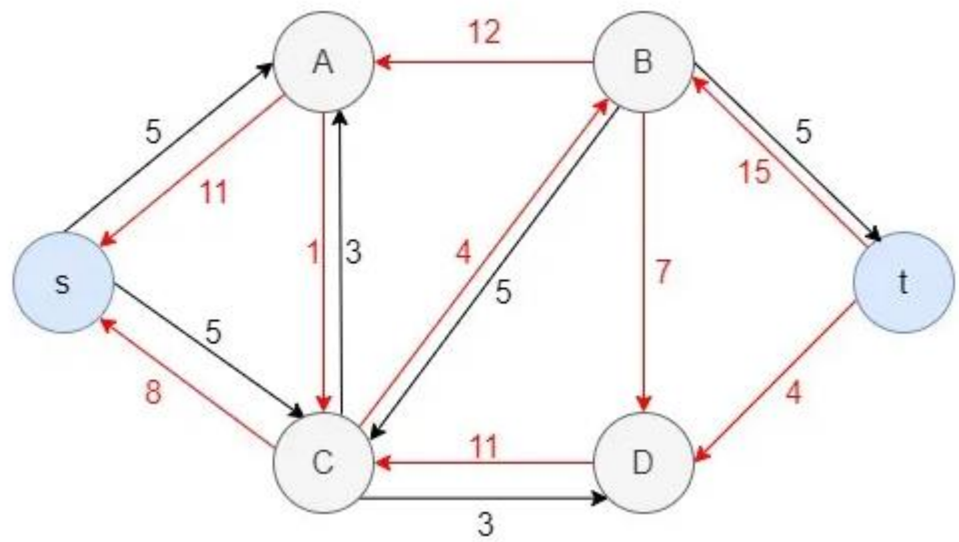
# Residual Graphs

A residual graph $R$ of a network $G$ has the same set of vertices as $G$ and includes, for each edge $e = (u, v) \in G$:

- A forward edge $e' = (u, v)$ with capacity $c_e - f_e$, if $c_e - f_e > 0$.
- A backward edge $e'' = (v, u)$ with capacity $f_e$, if $f_e > 0$.

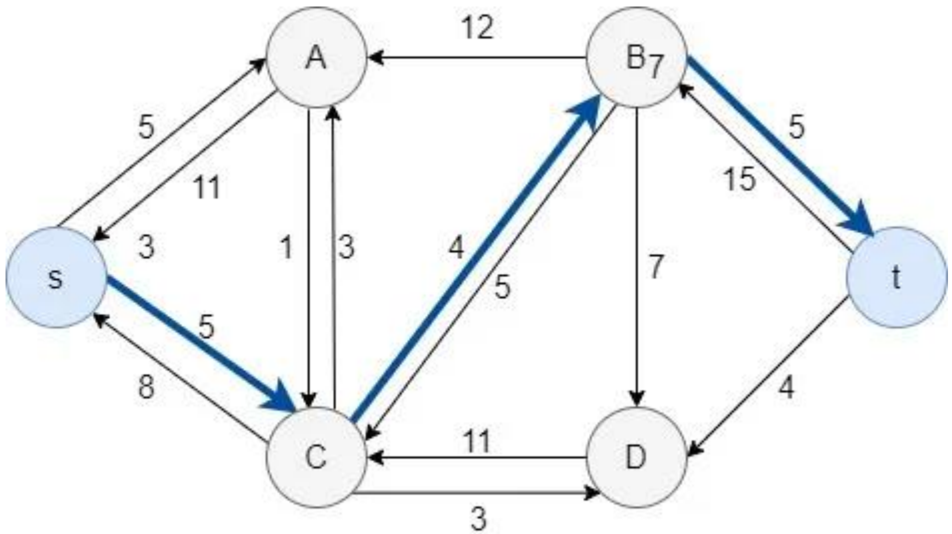
# Residual Graphs



# Augmenting Paths

- Given a flow network G=(V, E), the augmenting path is a simple path from s to t in the corresponding residual graph of the flow network.



# Ford-Fulkerson Algorithm

- Start with $f(e) = 0$ for each edge $e \in E$.
- Find an $s \rightsquigarrow t$ path $P$ in the residual network $G_f$.
- Augment flow along path $P$.
- Repeat until you get stuck.

FORD–FULKERSON($G$)

FOREACH edge $e \in E$ : $f(e) \leftarrow 0$.

$G_f \leftarrow$ residual network of $G$ with respect to flow $f$.

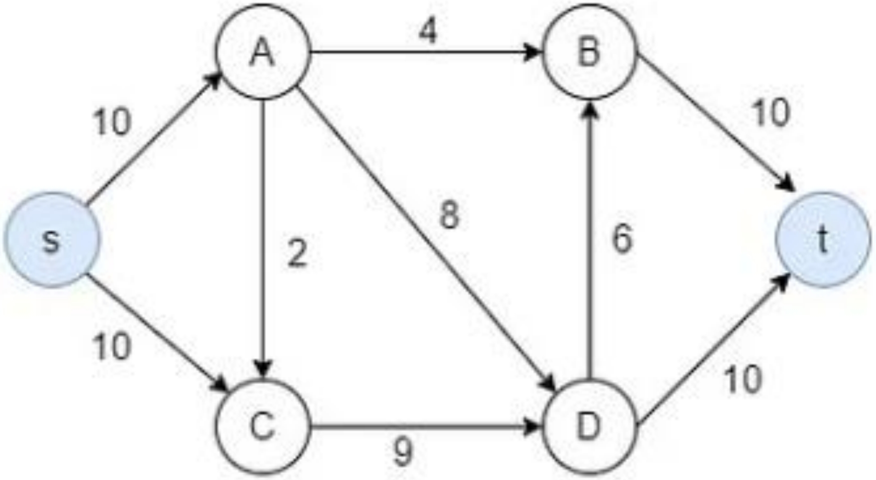
WHILE (there exists an s⇝t path $P$ in $G_f$)

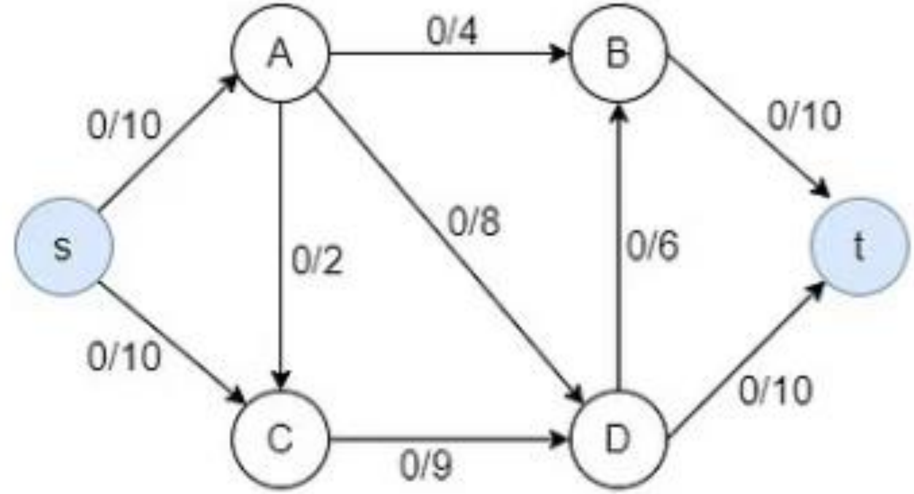
  $f \leftarrow$ AUGMENT($f, c, P$).

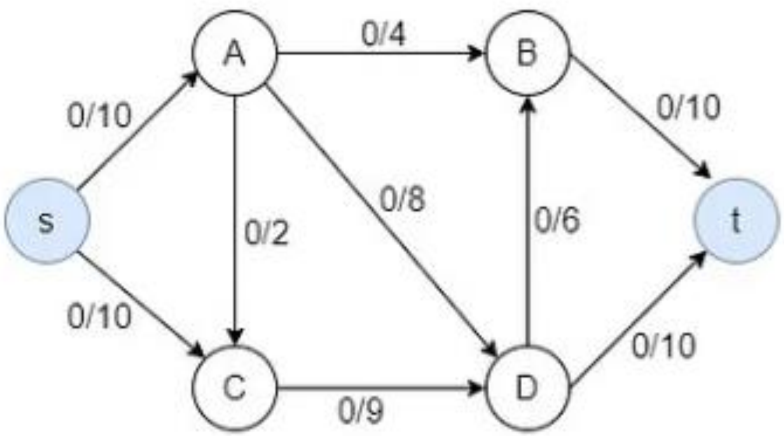
  Update $G_f$.

RETURN $f$.

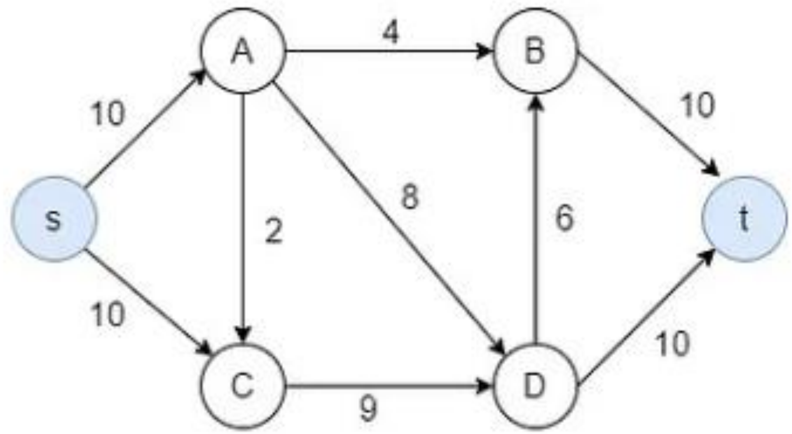
augmenting path



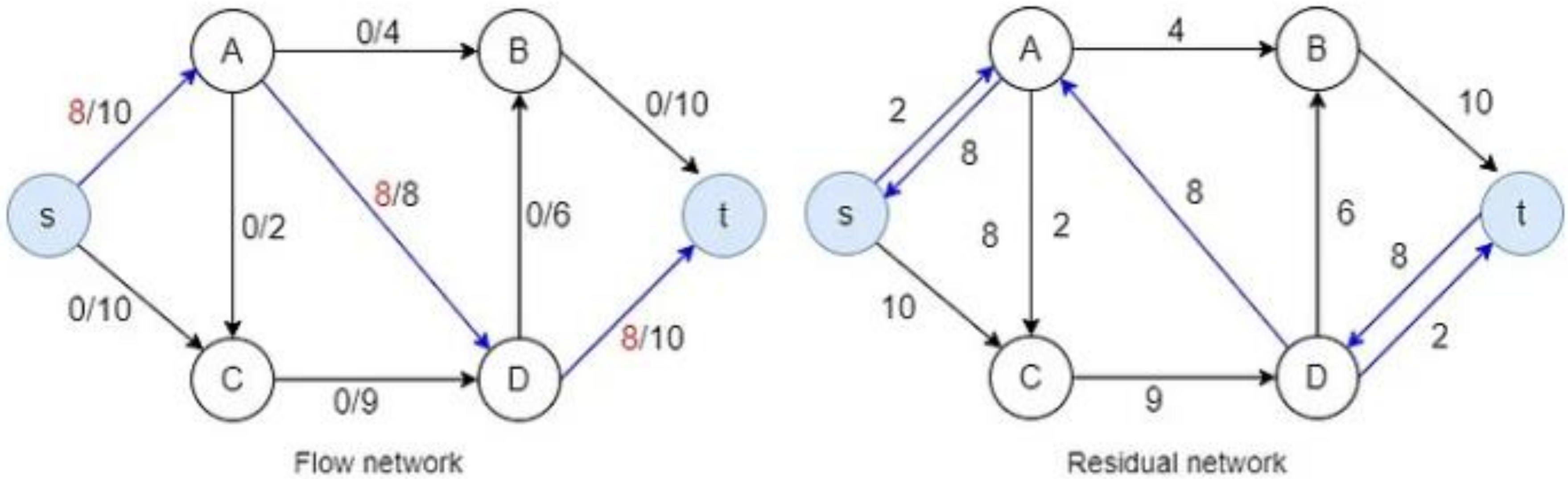



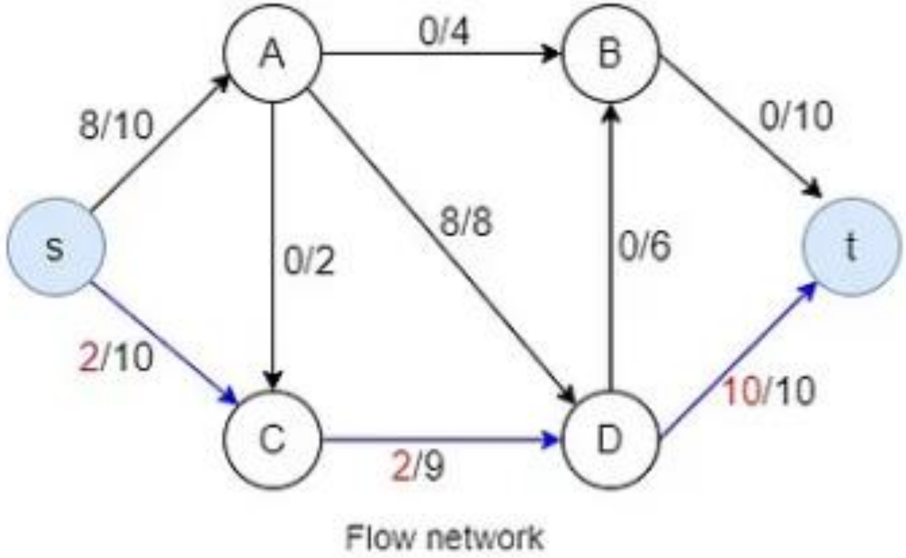

Flow network

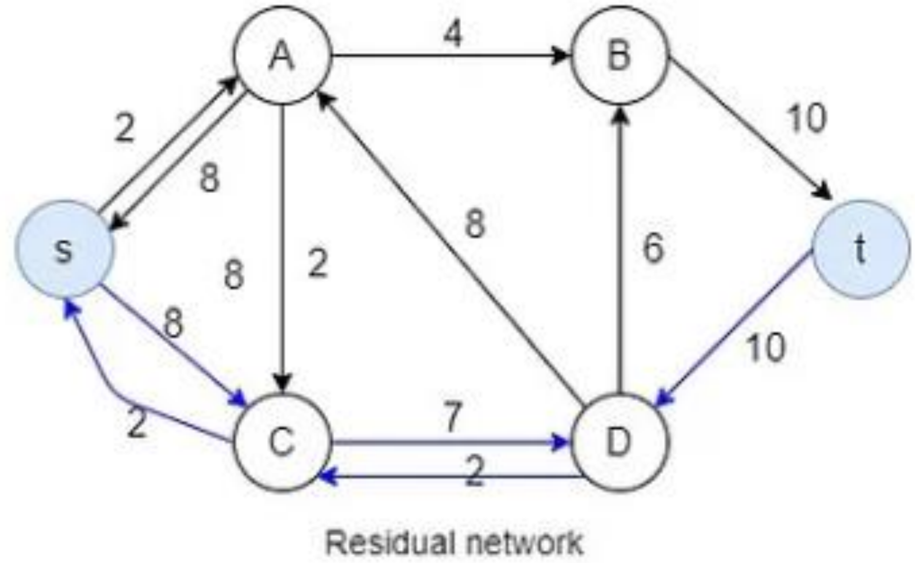



Residual network

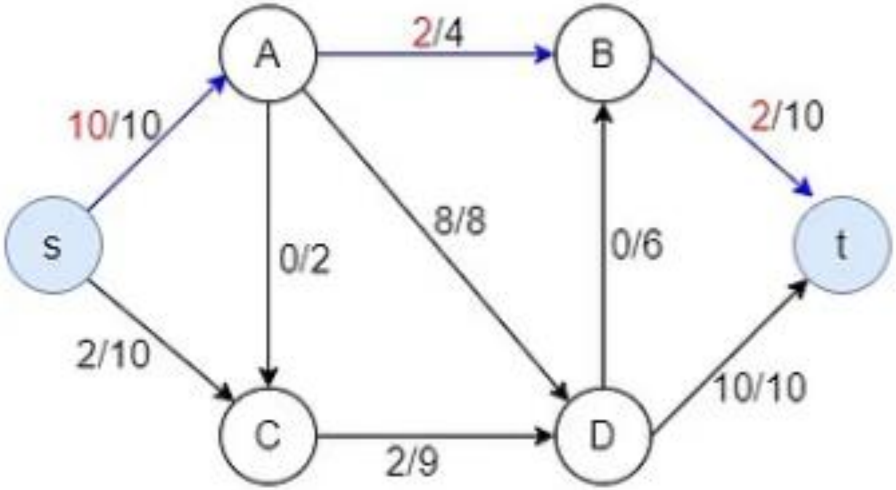



- Find an augmenting path in the residual network.
- s -> A -> D -> t.
- The bottleneck capacity is 8.

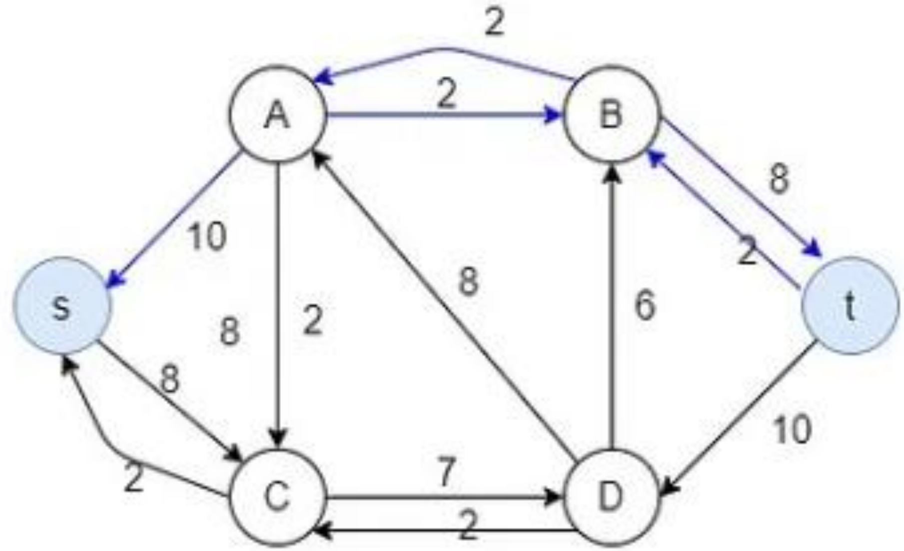



- Find an augmenting path in the residual network.
- s -> C -> D -> t.
- The bottleneck capacity is 2.

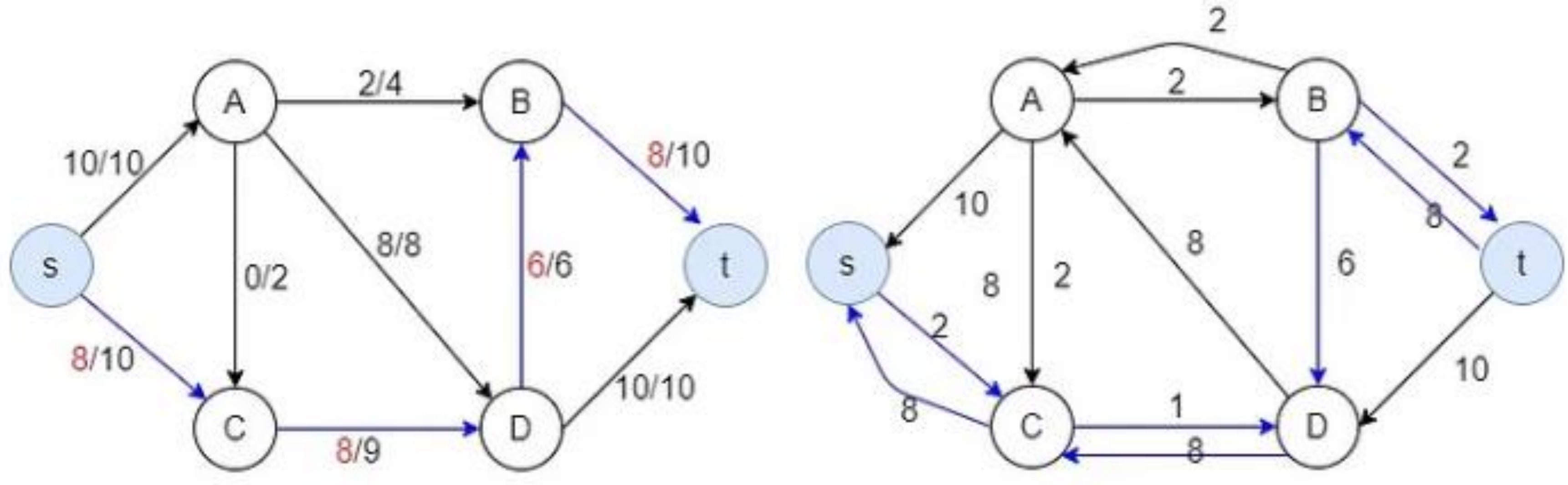



Flow network



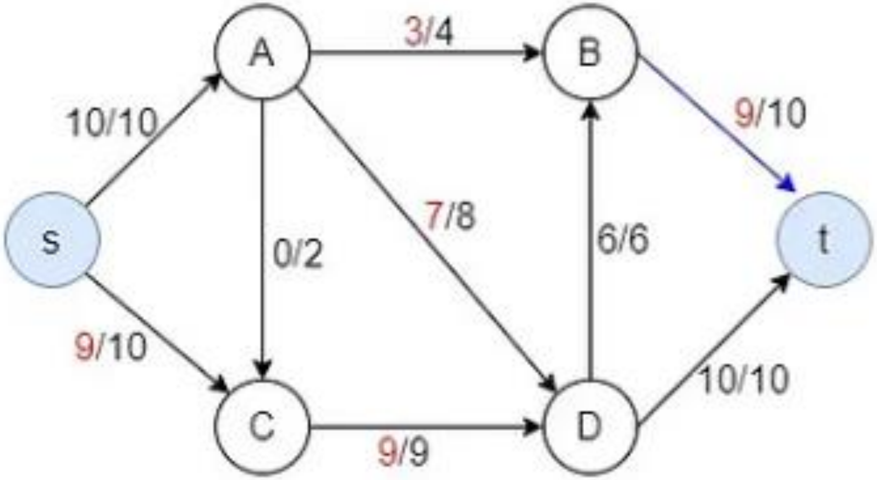

Residual network

- Find an augmenting path in the residual network.
- s -> A -> B->t.
- The bottleneck capacity is 2.

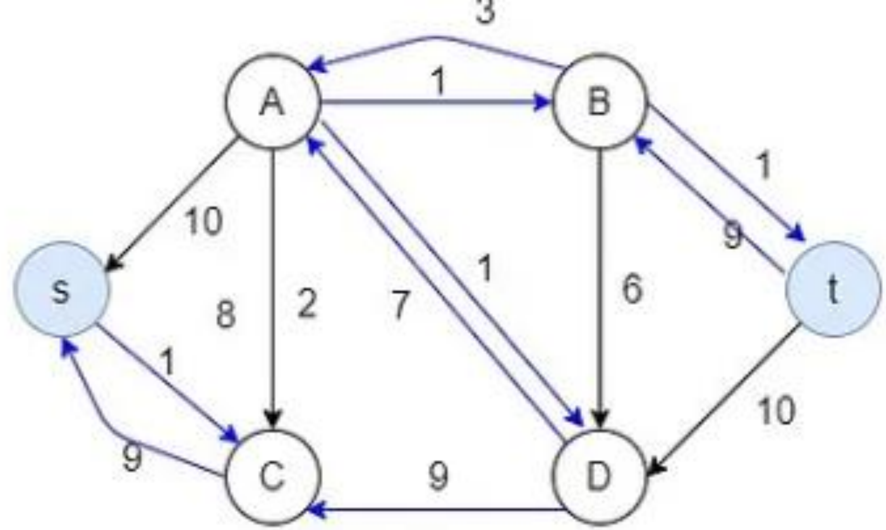



Flow network Residual network

- Find an augmenting path in the residual network.
- s -> C -> D -> B->t.
- The bottleneck capacity is 6.

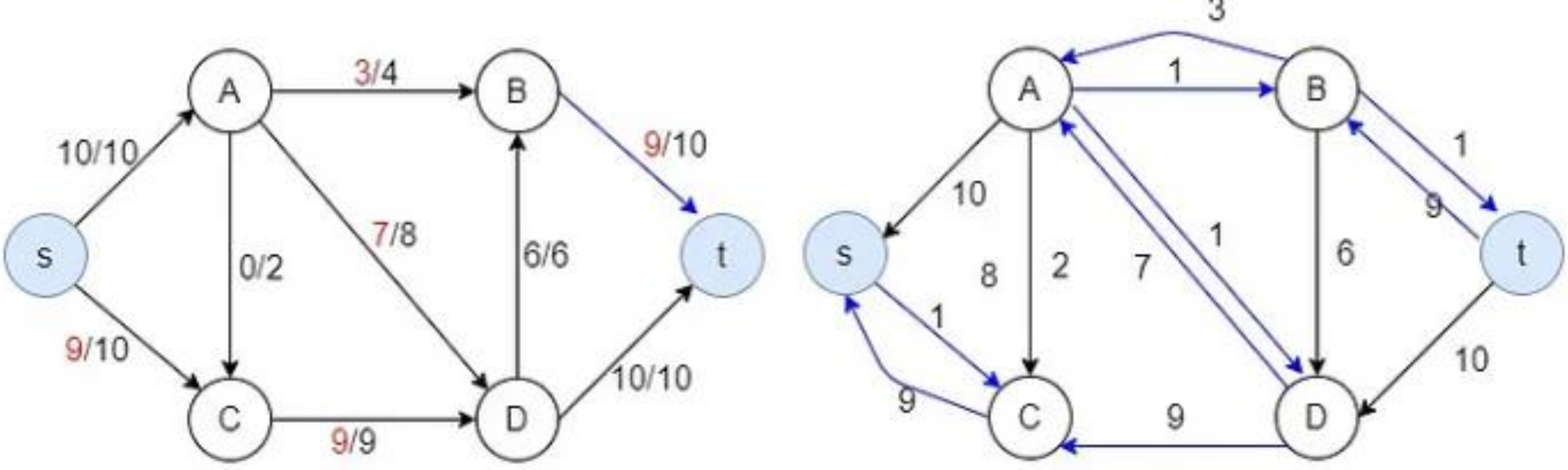


Flow network



Residual network

- Find an augmenting path in the residual network.
- s -> C -> D -> A-> B->t.
- The bottleneck capacity is 1.

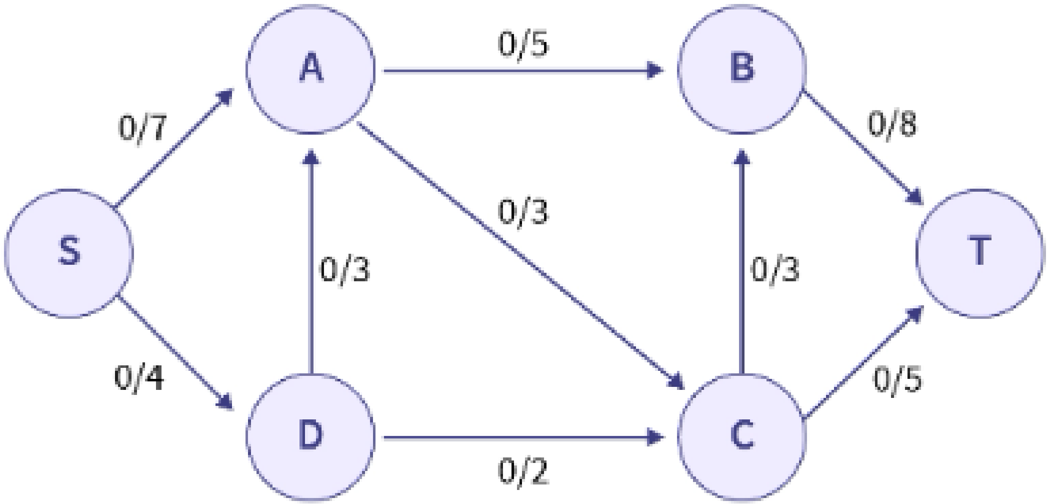


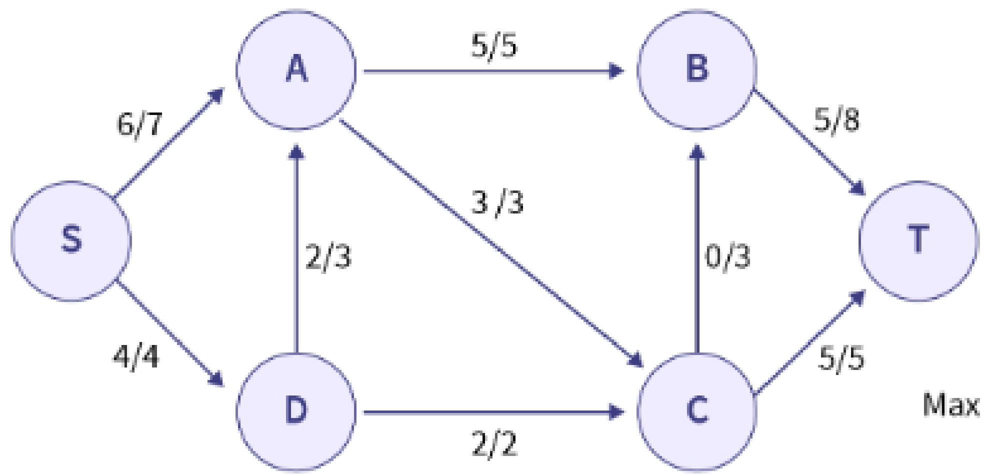

Flow network

Residual network

- Now there are no paths left from the s to t in the residual graph. So, there is no possibility to add flow.
- Since the maximum flow is equal to the flow coming out of the source, in this example, the maximum flow is 10+9 = 19.

# Ford-Fulkerson Algorithm: Exercise




Max Flow = 5+5
= 10

# Max Flow Min Cut Theorem

- In computer science and optimization theory, the max-flow min-cut theorem states that

In a flow network, the maximum amount of flow passing from the source to the sink is equal to the total weight of the edges in a minimum cut, i.e., the smallest total weight of the edges which if removed would disconnect the source from the sink.

# Max Flow Min Cut Theorem

- The second half of the max-flow min-cut theorem refers to the collection of cuts.
- An s-t cut C = (S, T) is a partition of V such that s ∈ S and t ∈ T.
- That is, an s-t cut is a division of the vertices of the network into two parts, with the source in one part and the sink in the other.

# Max Flow Min Cut Theorem

- The cut-set $X_c$ of a cut C is the set of edges that connect the source part of the cut to the sink part:

$$X_C := \{(u,v) \in E \;:\; u \in S, v \in T\} = (S \times T) \cap E.$$

- Thus, if all the edges in the cut-set of C are removed, then no positive flow is possible, because there is no path in the resulting graph from the source to the sink.

# Max Flow Min Cut Theorem

The **capacity** of an *s-t cut* is the sum of the capacities of the edges in its cut-set,

$$c(S,T) = \sum\nolimits_{(u,v)\in X_C} c_{uv} = \sum\nolimits_{(i,j)\in E} c_{ij} d_{ij},$$

where $d_{ij} = 1$ if $i \in S$ and $j \in T$, $0$ otherwise.

There are typically many cuts in a graph, but cuts with smaller weights are often more difficult to find.

> **Minimum s-t Cut Problem.** Minimize $c(S, T)$, that is, determine $S$ and $T$ such that the capacity of the s-t cut is minimal.

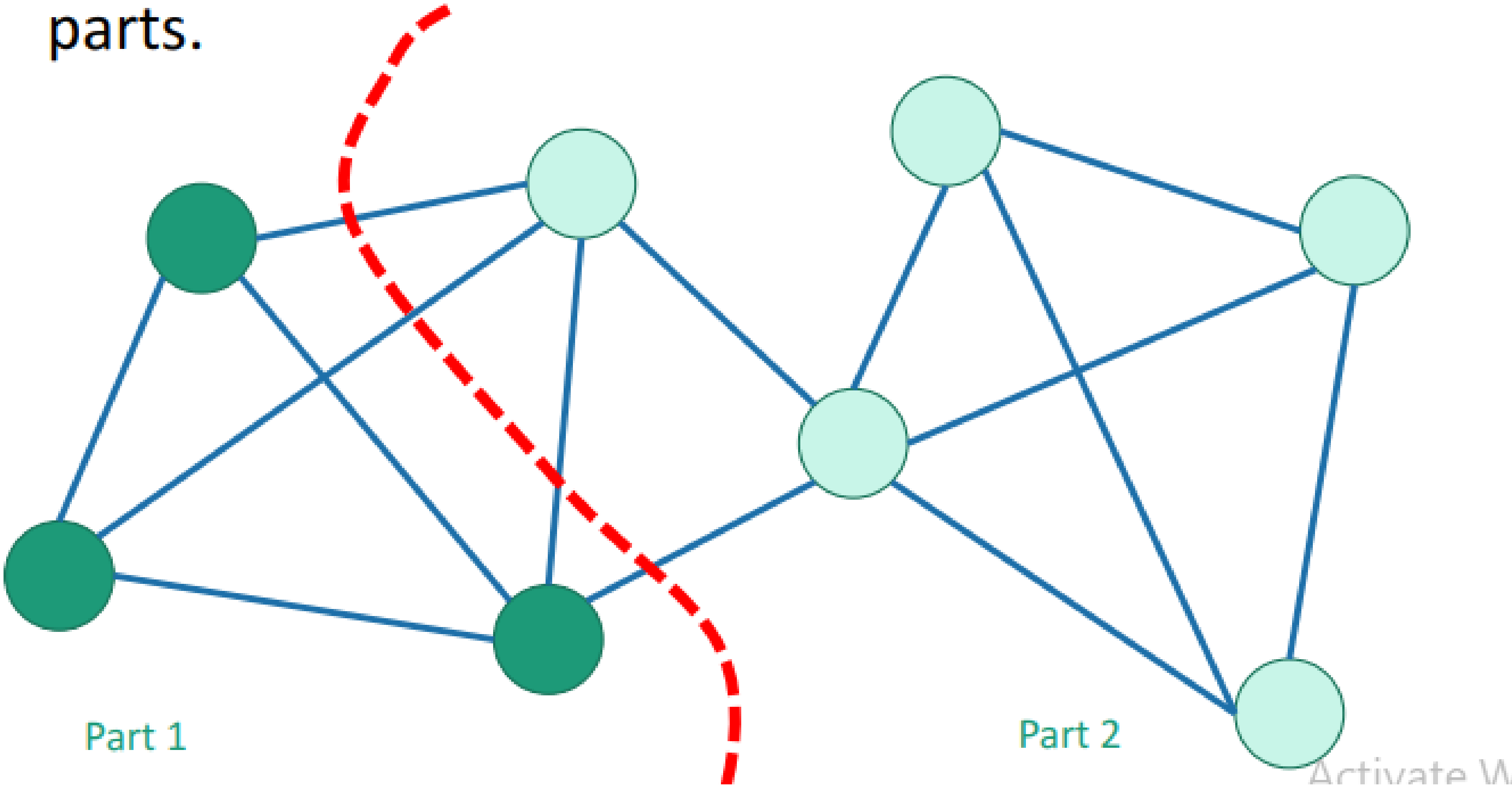
# Max Flow Min Cut Theorem

- The maximum value of an s-t flow is equal to the minimum capacity over all s-t cuts.

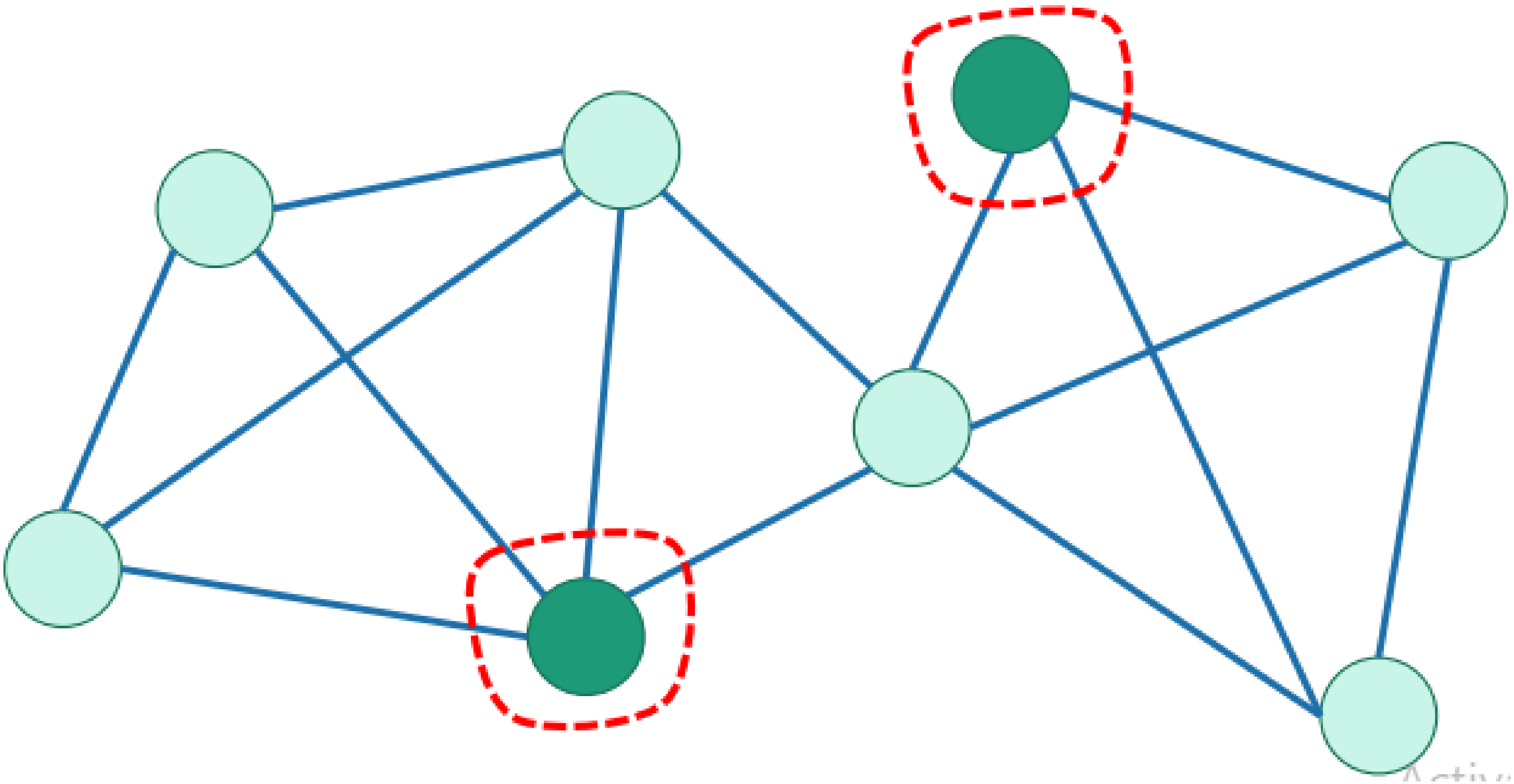
- The max-flow min-cut theorem states that the maximum flow through any network from a given source to a given sink is exactly equal to the minimum sum of a cut.

# Cuts in a Graph

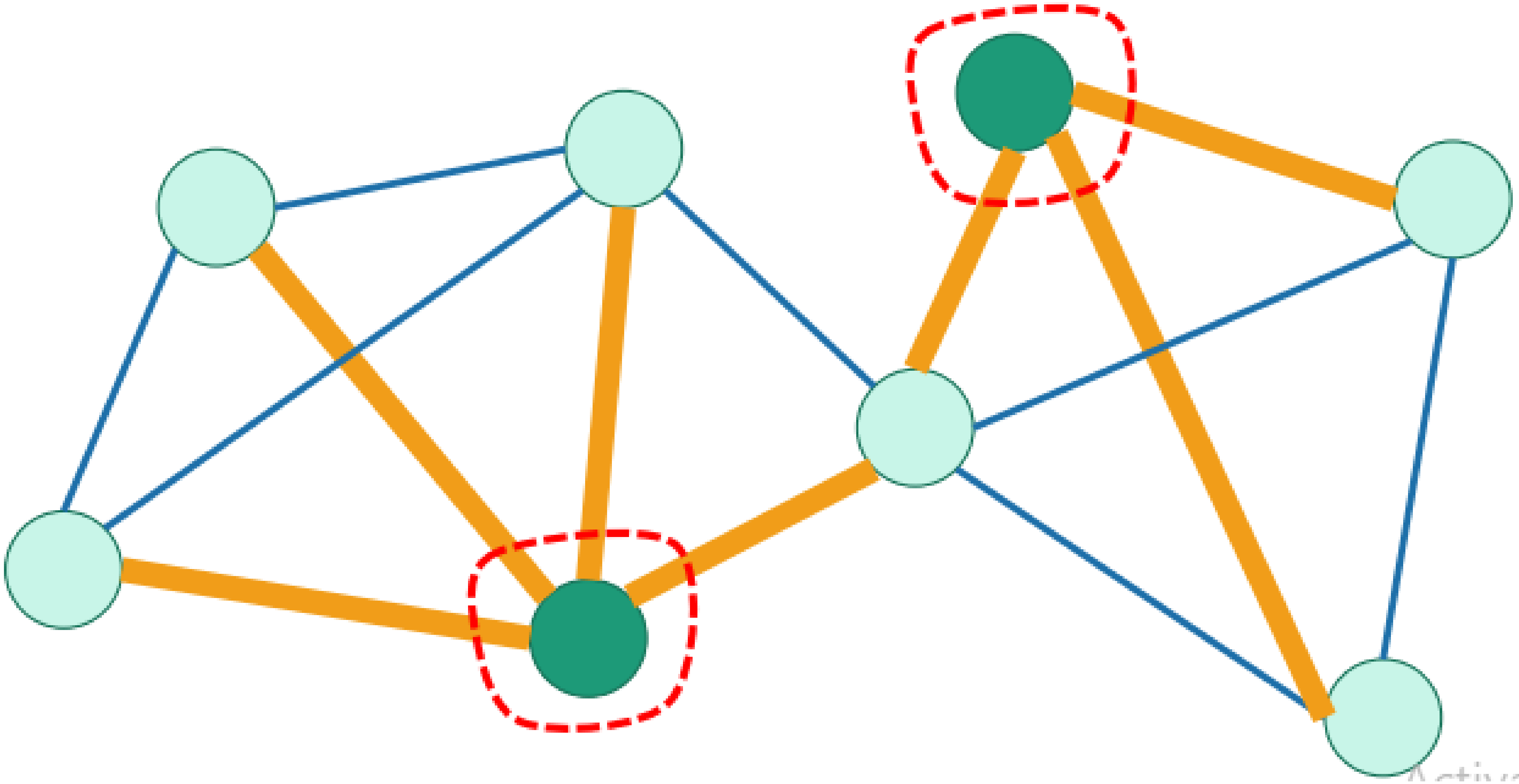
- A cut is a partition of the vertices into two nonempty parts.



# Cuts in a Graph

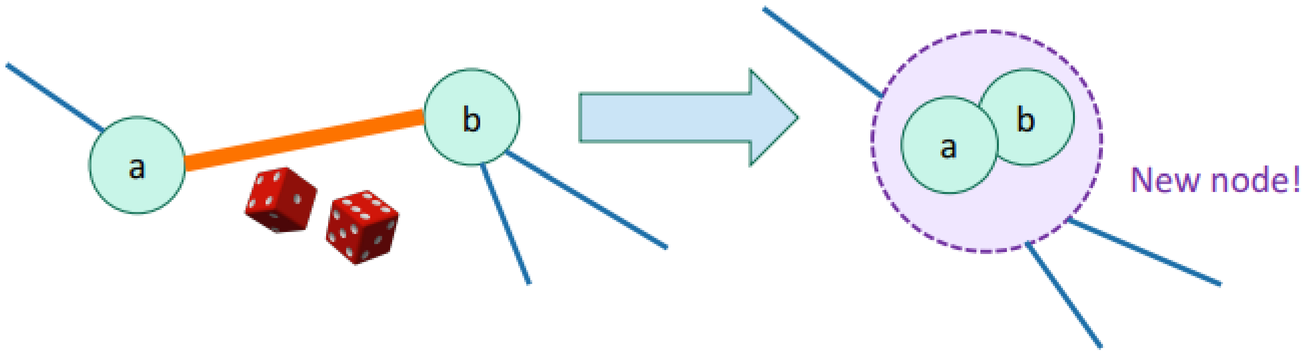


# Cuts in a Graph

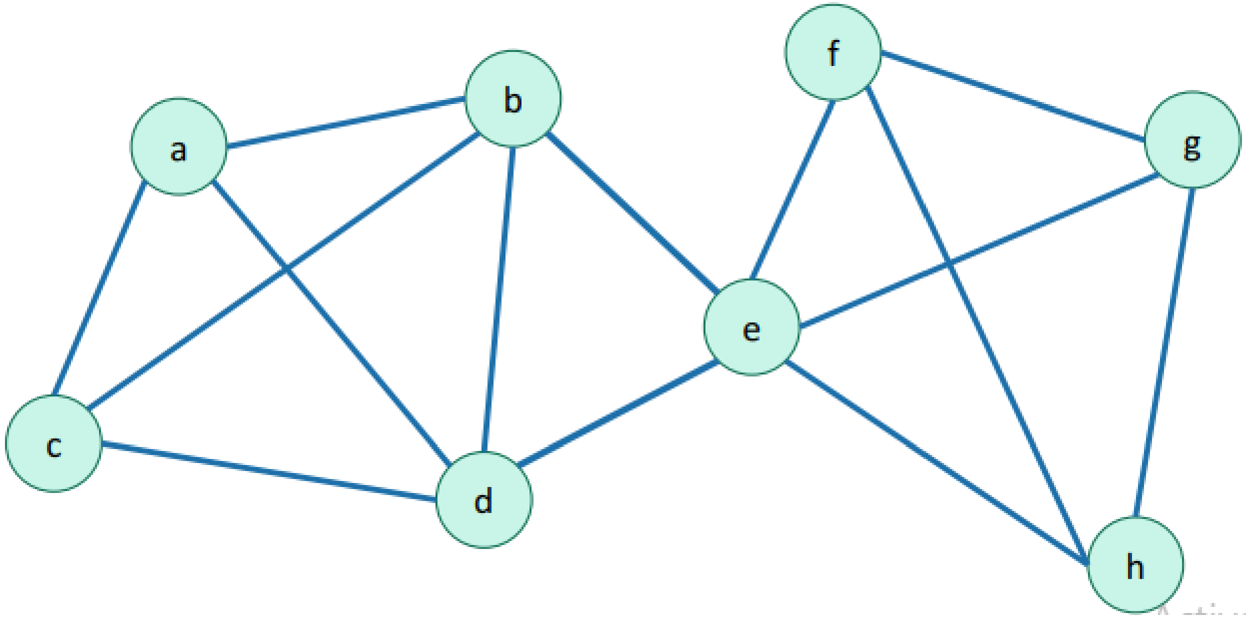


# Karger's Algorithm

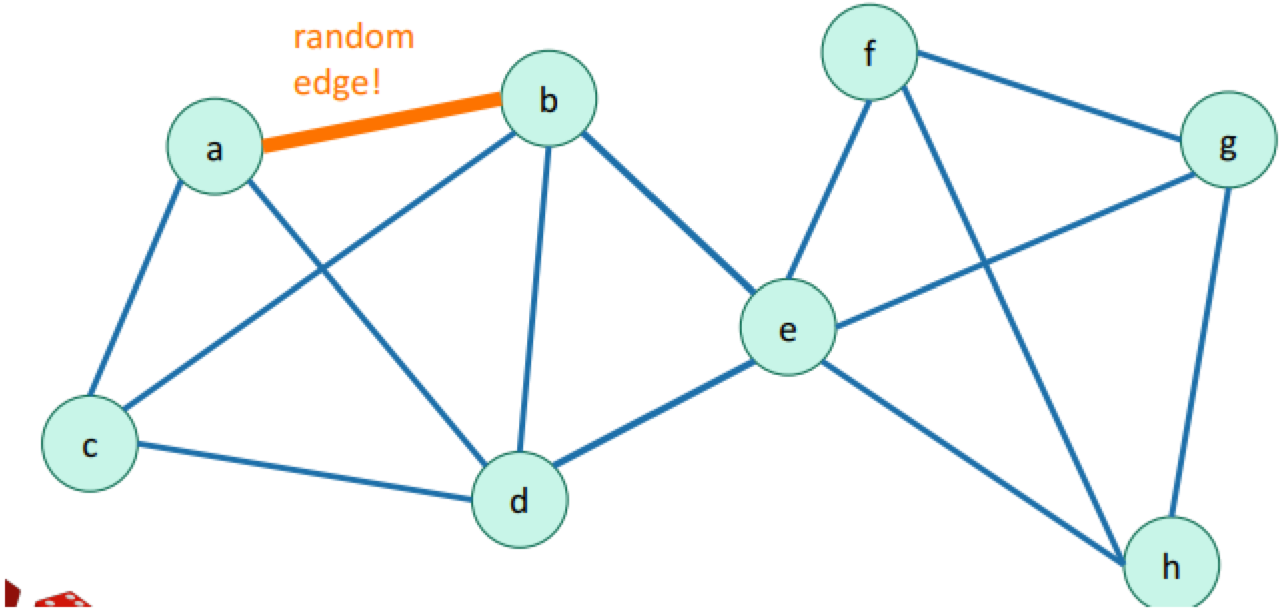
- Pick a random edge.
- **Contract** it.
- Repeat until you only have two vertices left.



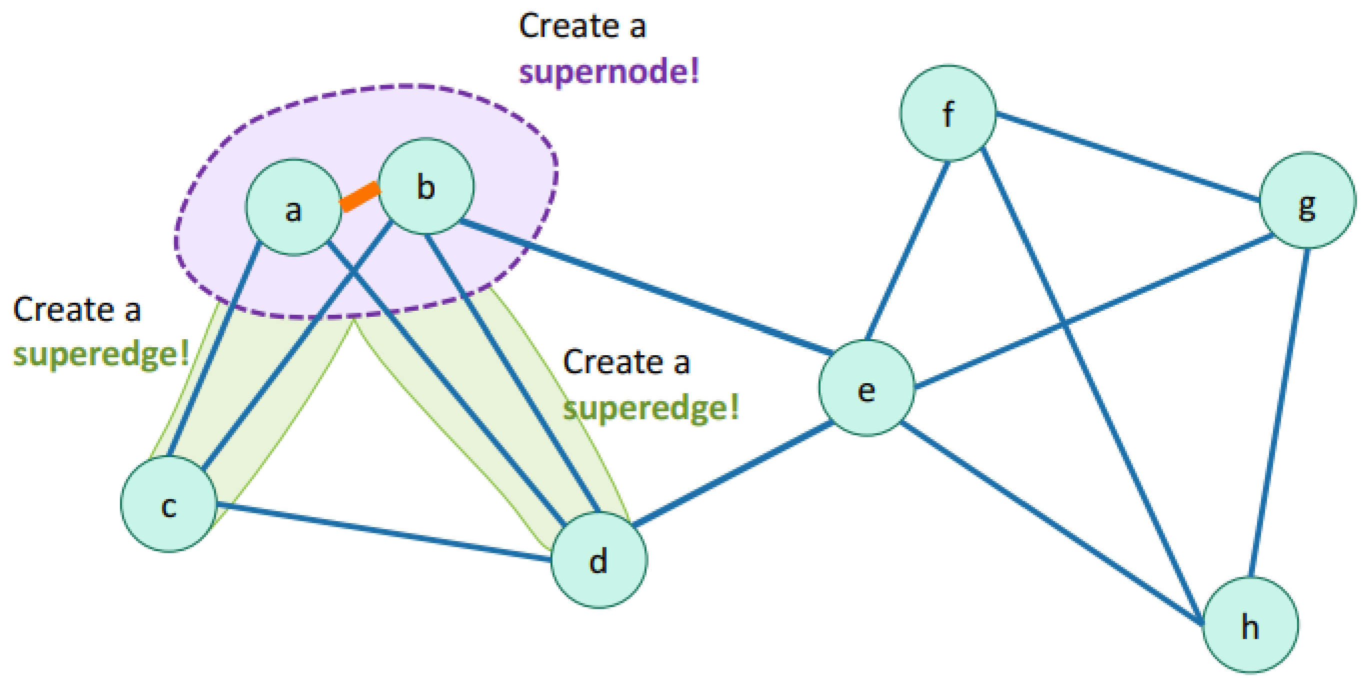
# Karger's Algorithm



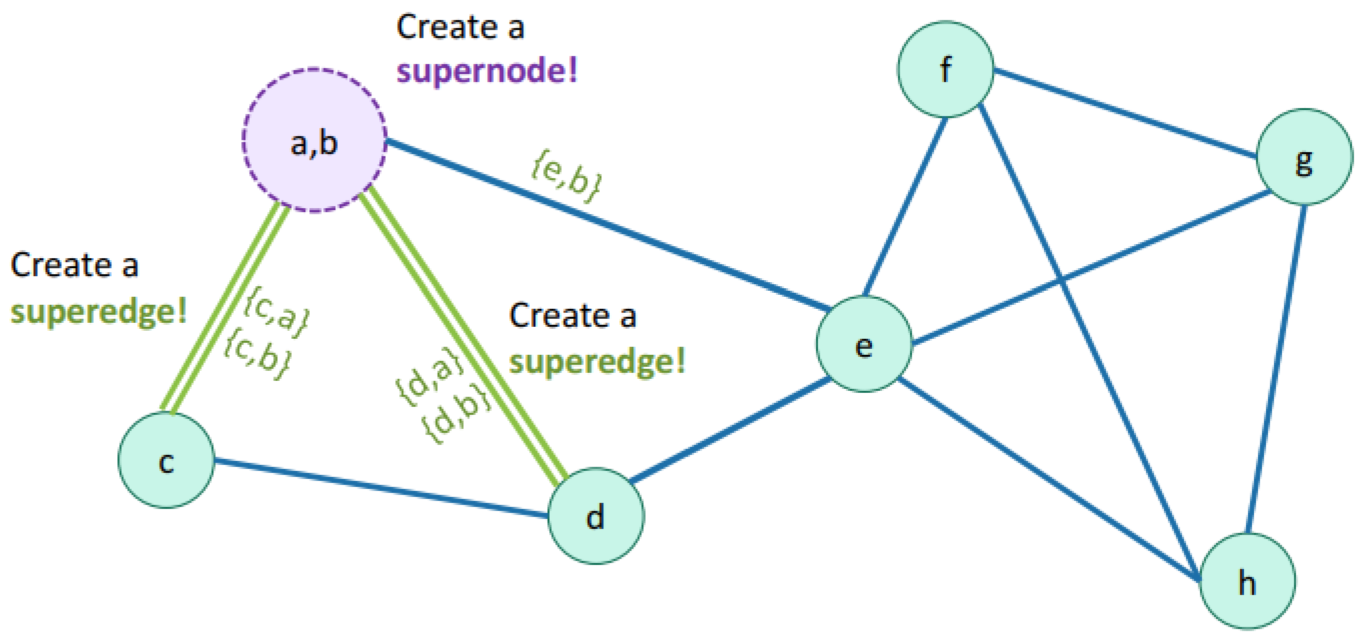
# Karger's Algorithm



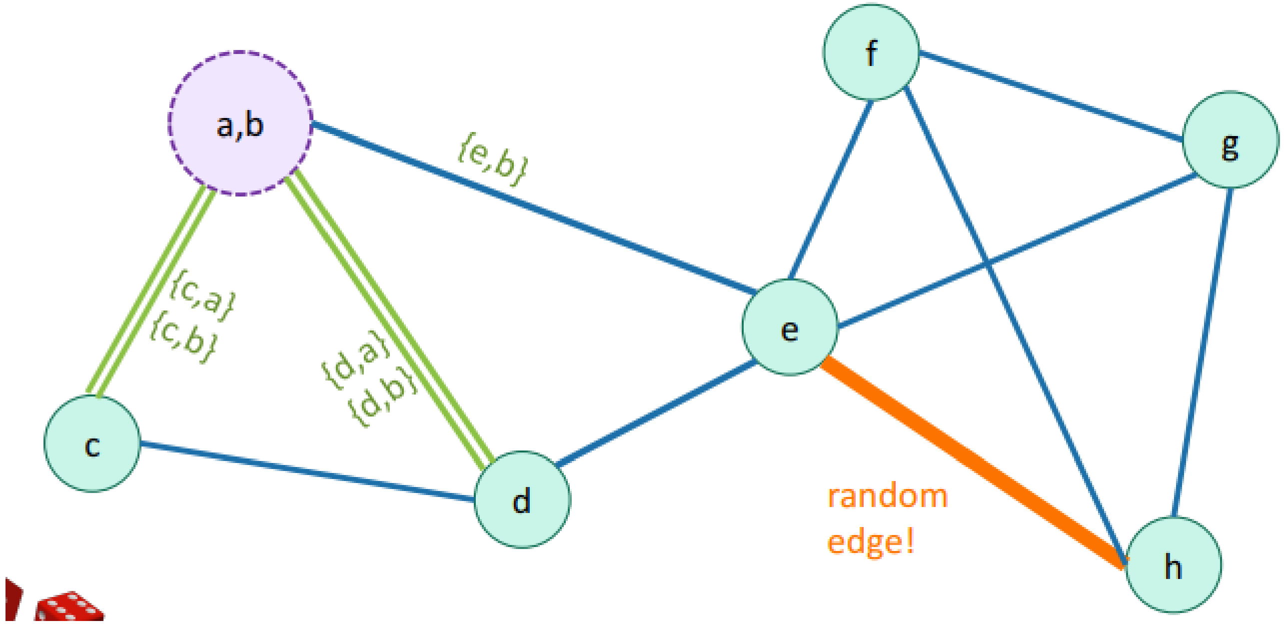
# Karger's Algorithm



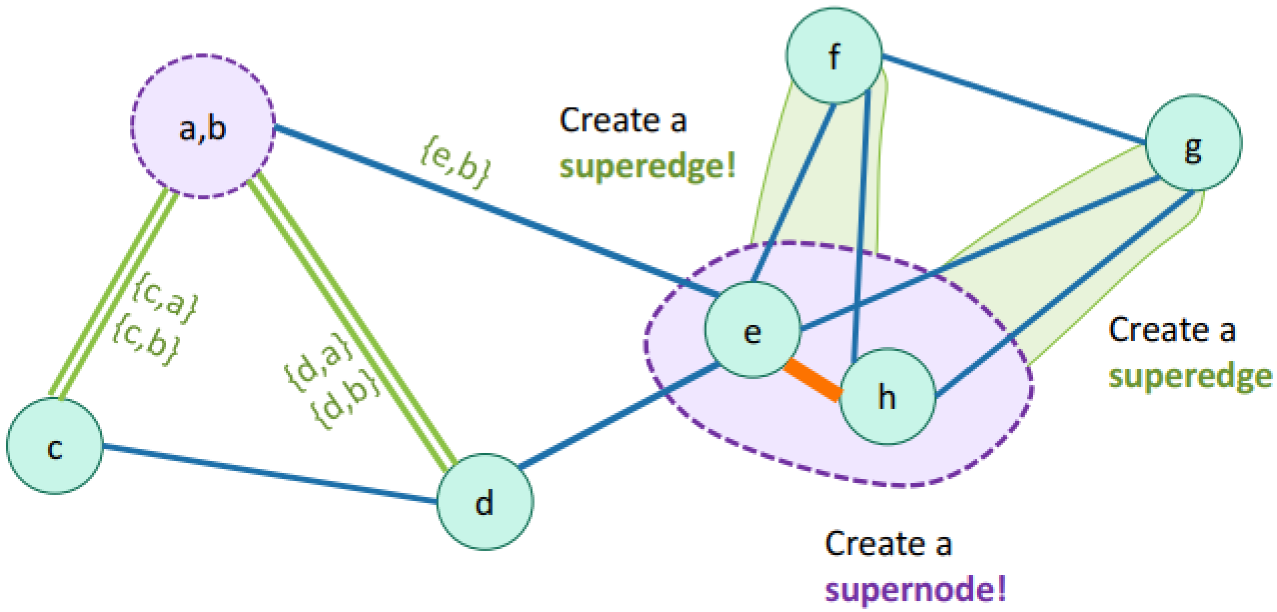
# Karger’s Algorithm



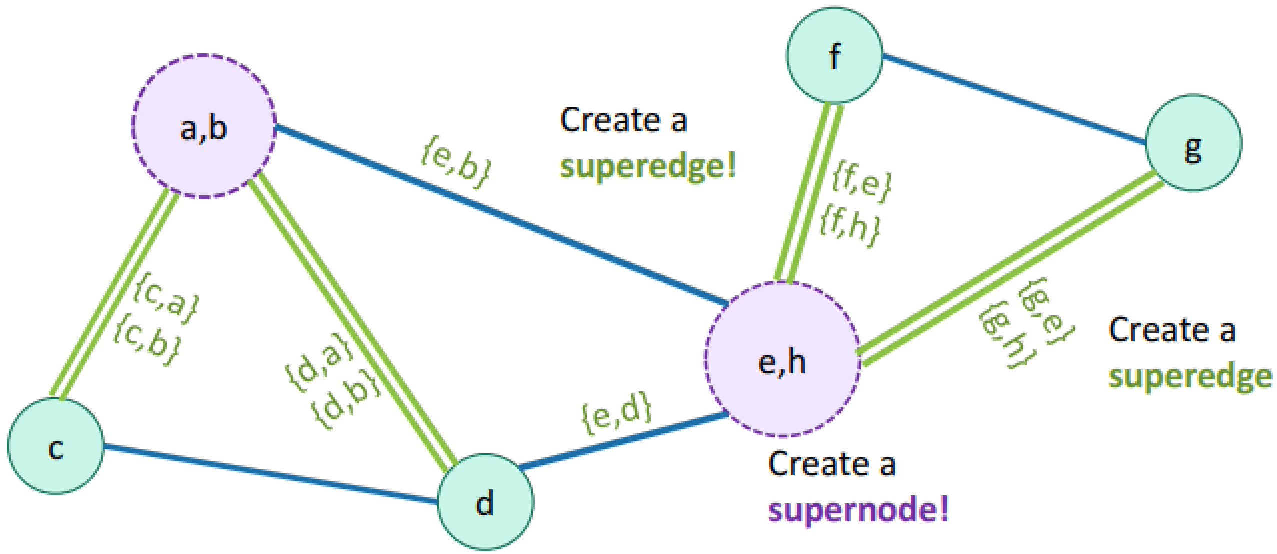
# Karger's Algorithm



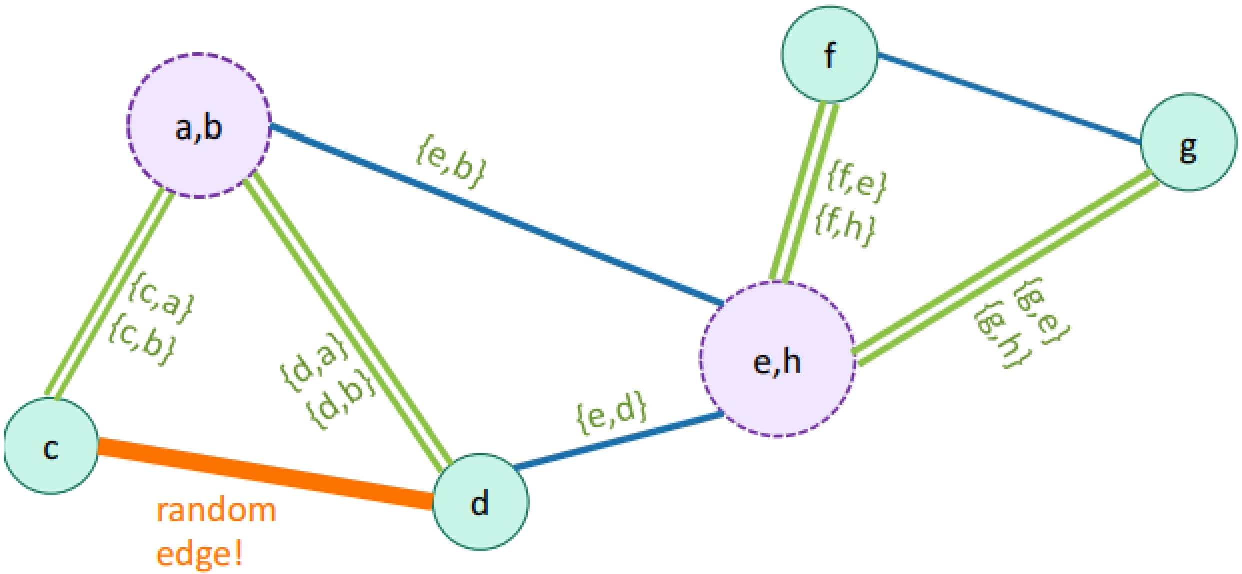
# Karger's Algorithm



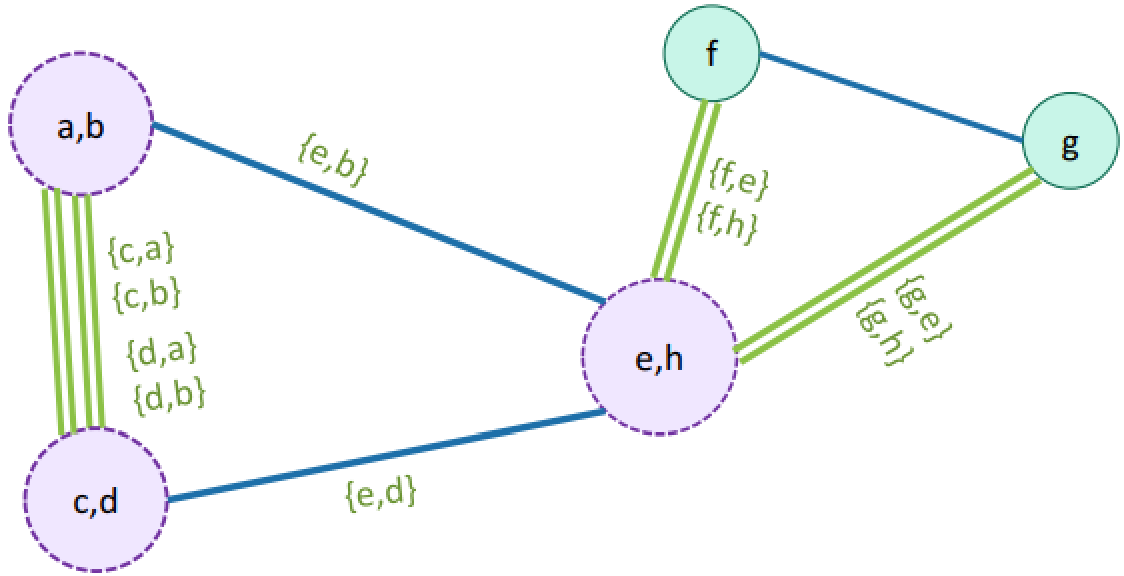
# Karger's Algorithm



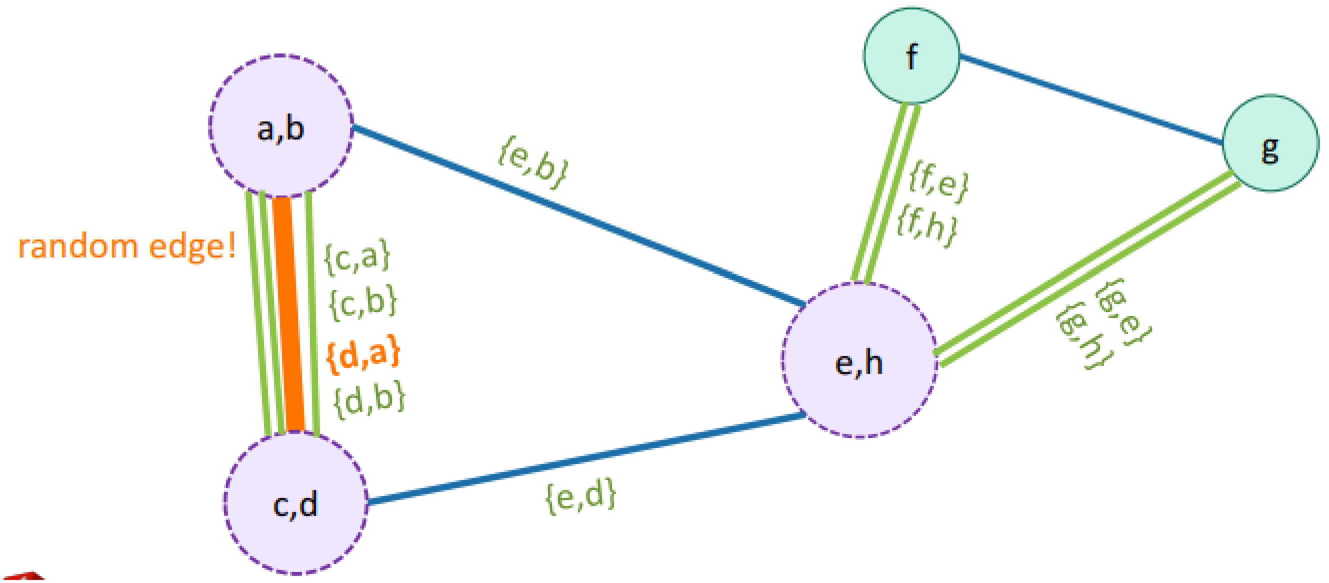
# Karger's Algorithm



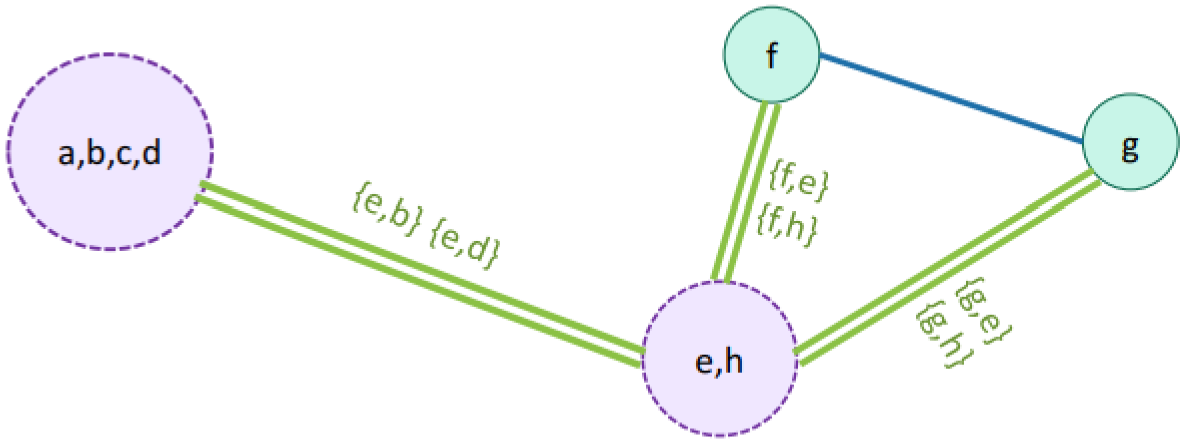
# Karger's Algorithm



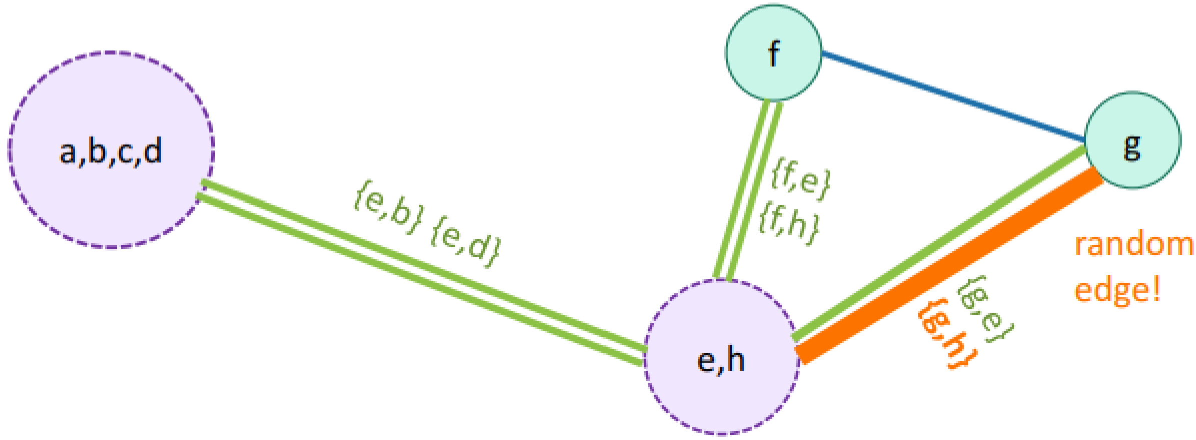
# Karger's Algorithm



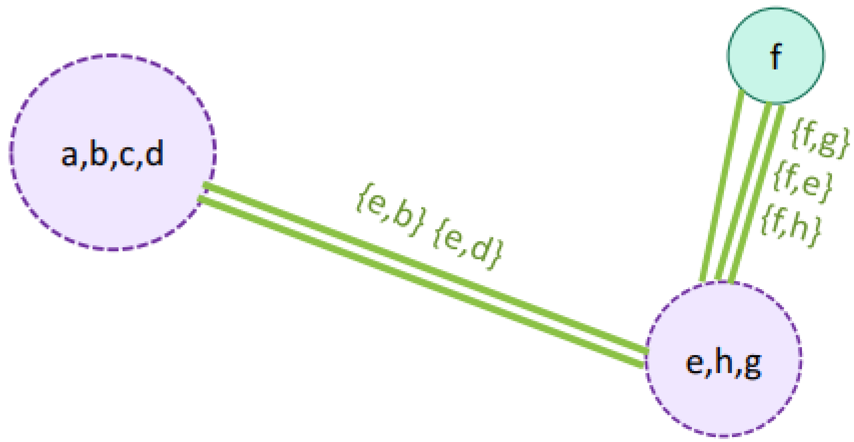
# Karger's Algorithm



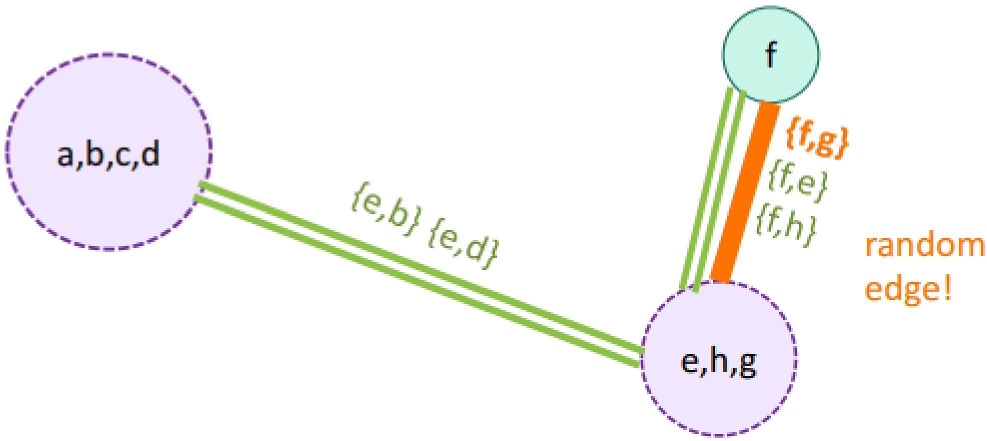
# Karger's Algorithm



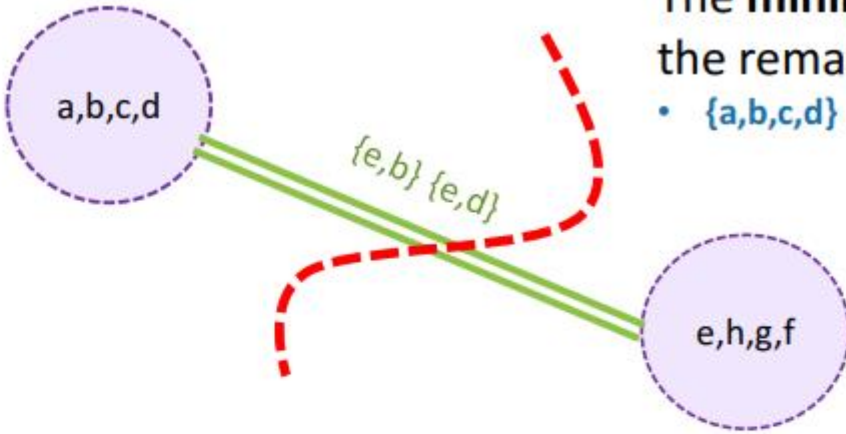
# Karger's Algorithm



# Karger's Algorithm



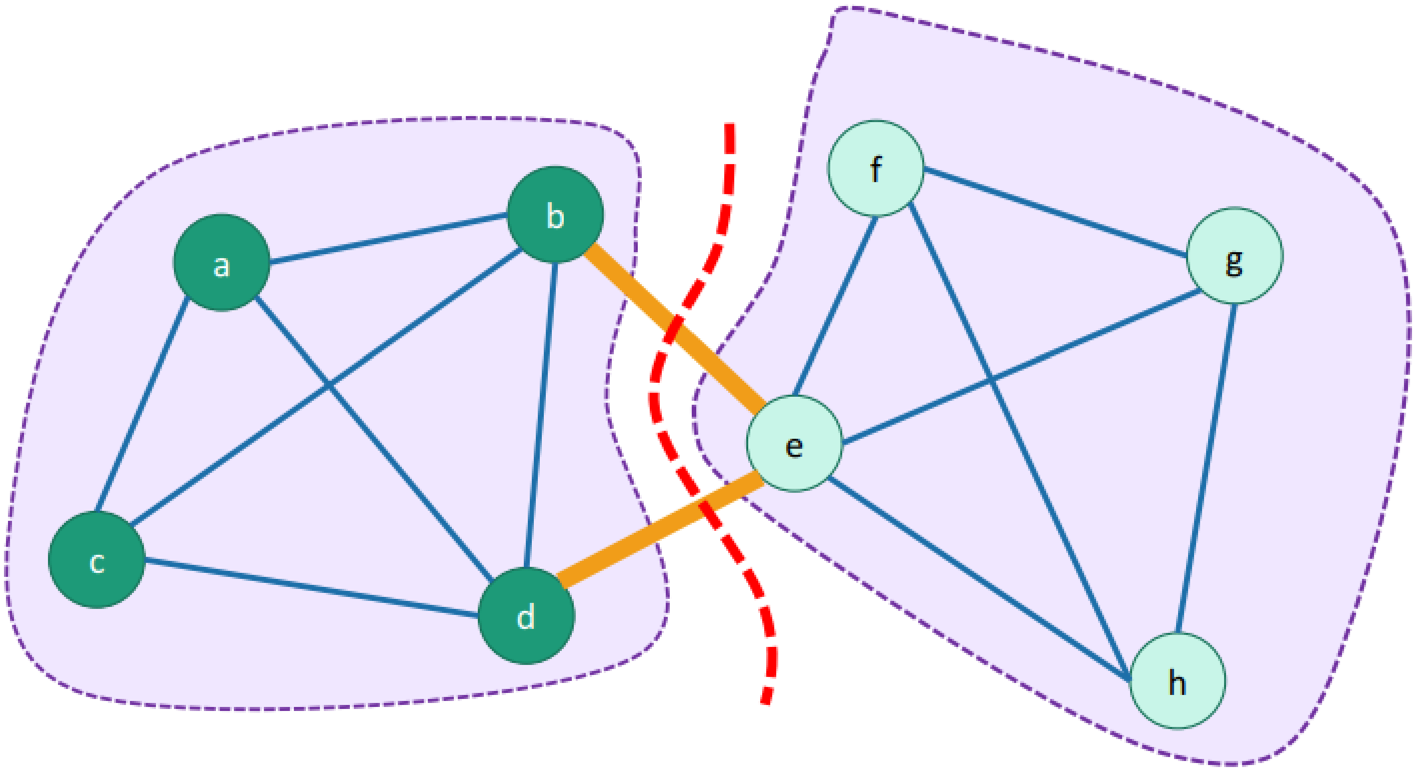
# Karger’s Algorithm



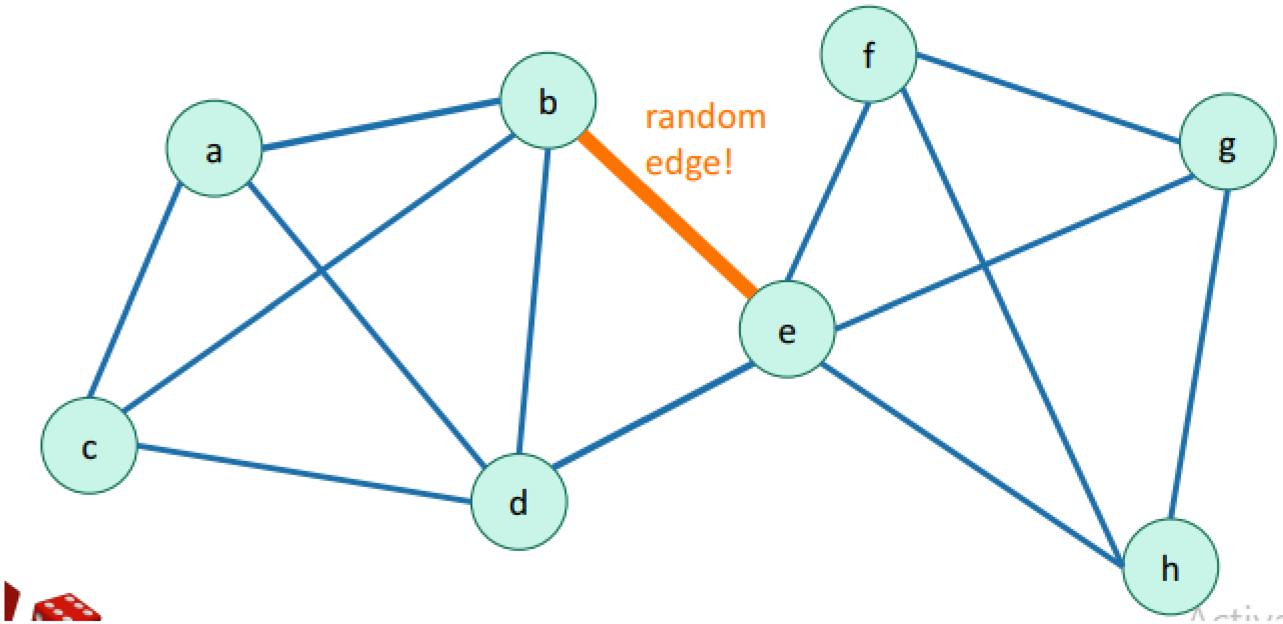

The **minimum cut** is given by the remaining super-nodes:

- {a,b,c,d} and {e,h,f,g}

# Karger's Algorithm

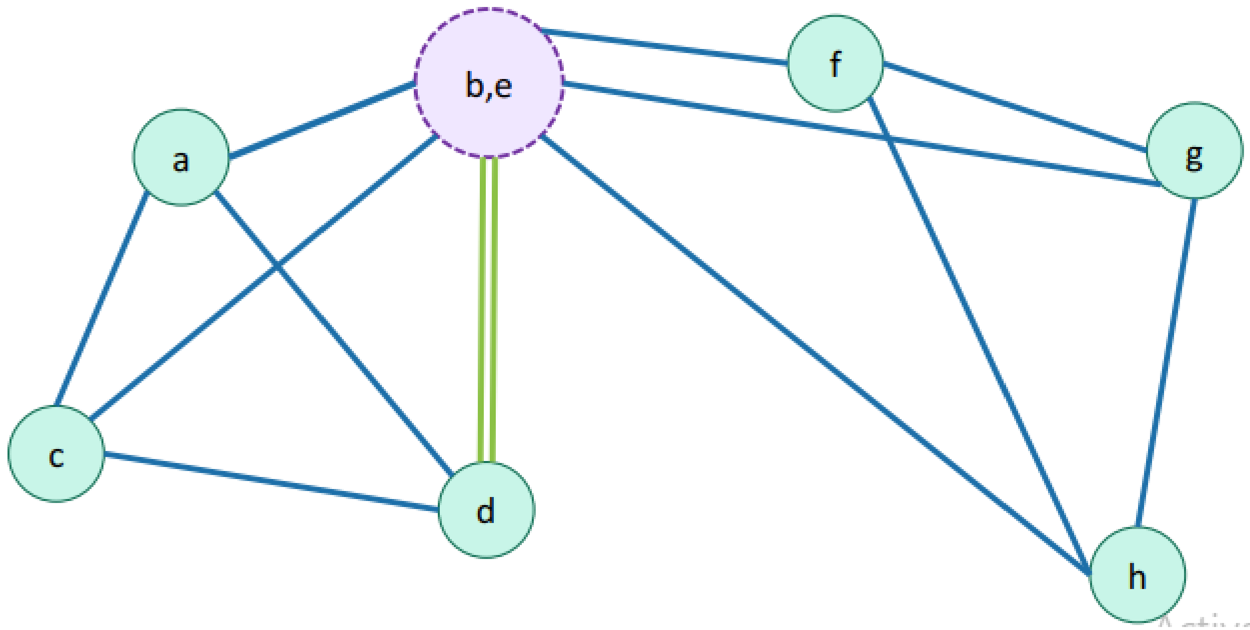


# Problem with Karger's Algorithm



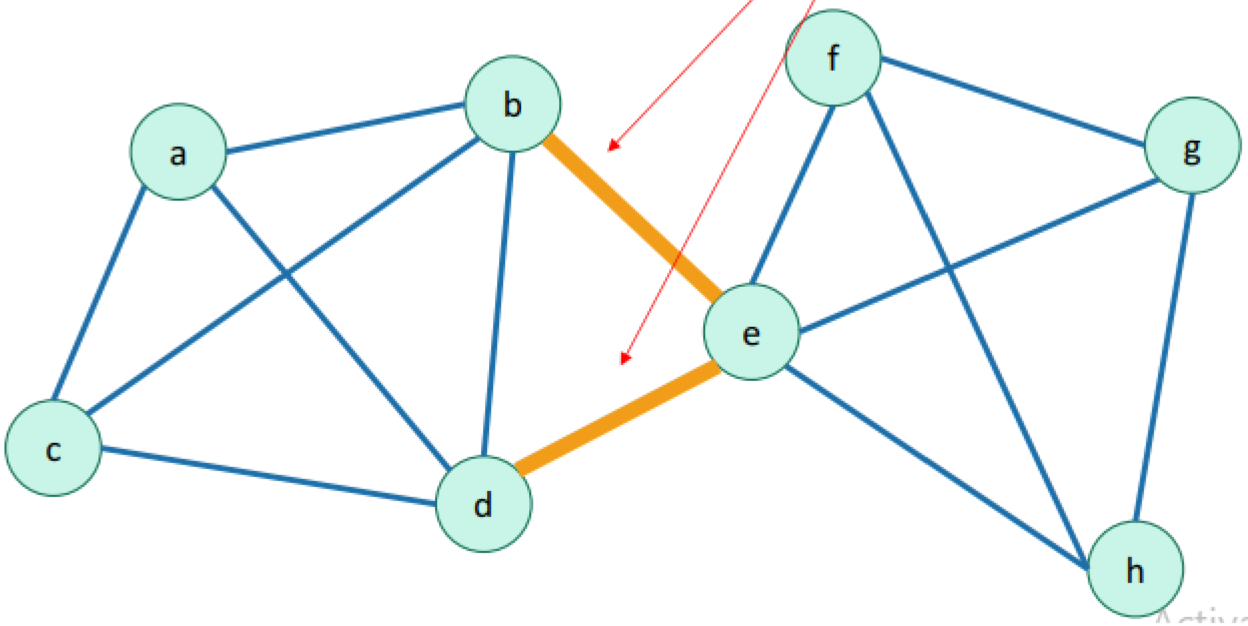
# Problem with Karger's Algorithm

Now there is **no way** we could return a cut that separates b and e.



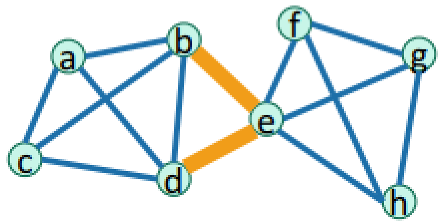
# Problem with Karger’s Algorithm

If the algorithm **EVER** chooses either of **these edges**, **it will be wrong.**



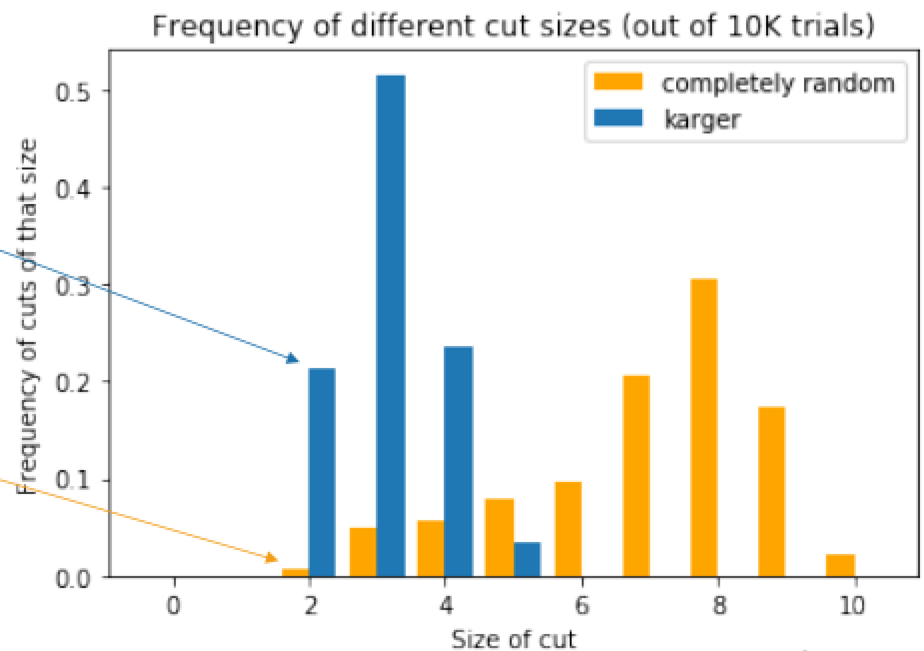
# Problem with Karger's Algorithm

## Karger is better than completely random!




Karger's alg. is correct about 20% of the time

Completely random is correct about 0.8% of the time



# Summary

- Flow Networks