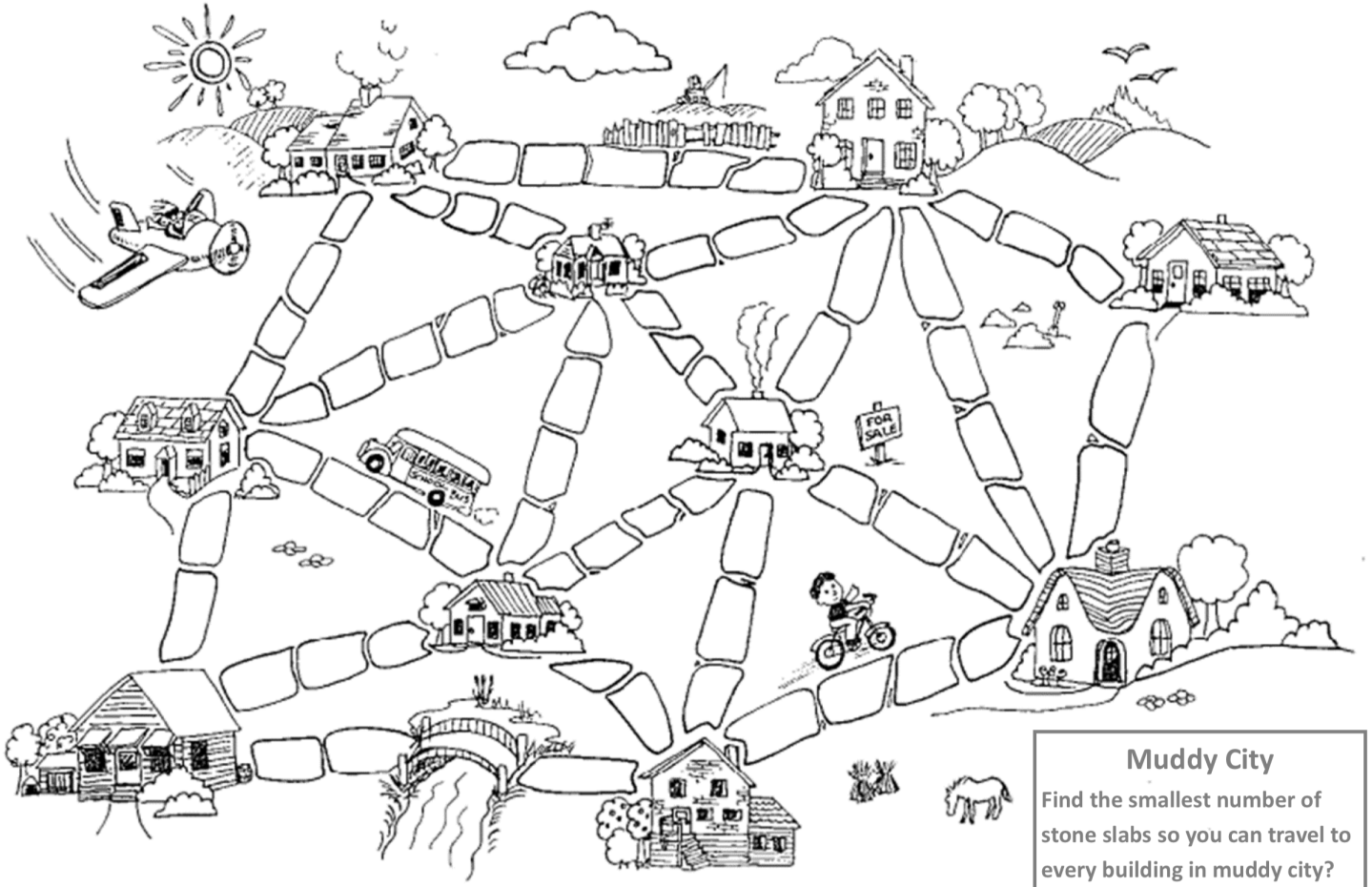
# CSCI 3104: Algorithms

# Lecture 15: Minimum Spanning Trees – continued

Rachel Cox

Department of Computer Science




**Muddy City**

Find the smallest number of stone slabs so you can travel to every building in muddy city?

Image Source

What will we learn today?

- ❑ Minimum Spanning Trees
- ❑ Generic Algorithm
- ❑ Maximum Flow, Minimum Cut

Intro to Algorithms, CLRS:
Sections 23.2, 26.1

# Acknowledgements & Citations


- Material, examples, and definitions taken from CSCI 3104 Week 9 Notes by Professor Aaron Clauset.

# Minimum Spanning Trees – Recap

A ***spanning tree*** $T$ is a set of edges $E' \subseteq E$, of size $|E'| = |V| - 1$, such that for every pair of vertices $u, v \in V$, there exists a path from $u$ to $v$ in $T$.

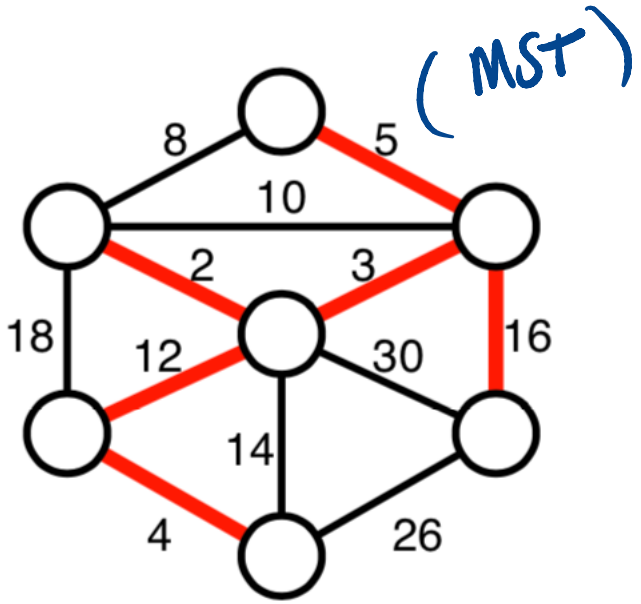
A ***minimum spanning tree*** (MST) is defined as a spanning tree $T$ that minimizes the "cost" function.

$$w(T) = \sum_{e \in T} w(e)$$

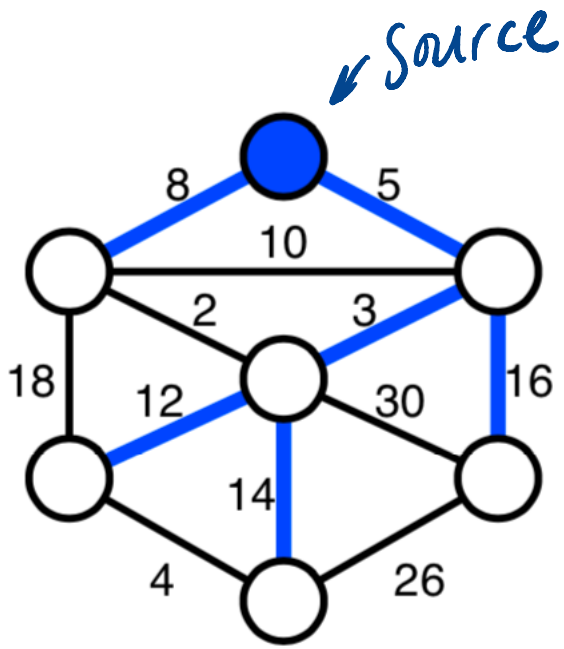
A minimum spanning tree provides a minimum cost to finding a path between any pair of vertices

# Minimum Spanning Trees – Recap

**Example**: The red edges below show the Minimum Spanning Tree, while the blue edges show a Single-Source Shortest Path




(MST)




Single-Source Shortest Path tree (SSSP tree)

SSSP tree ≠ MST

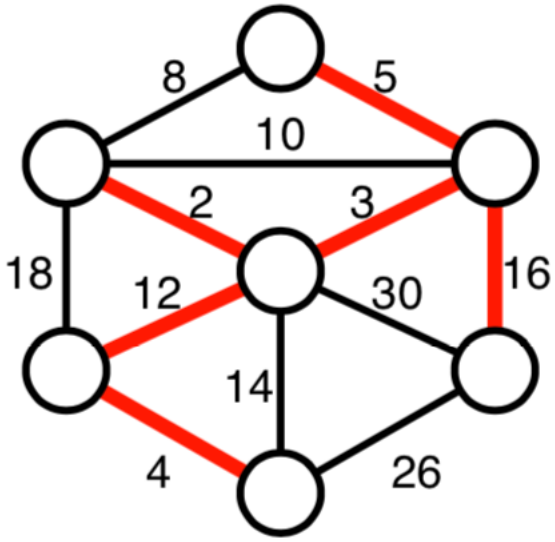
# Minimum Spanning Trees – Generic Algorithm

At each step of the algorithm:

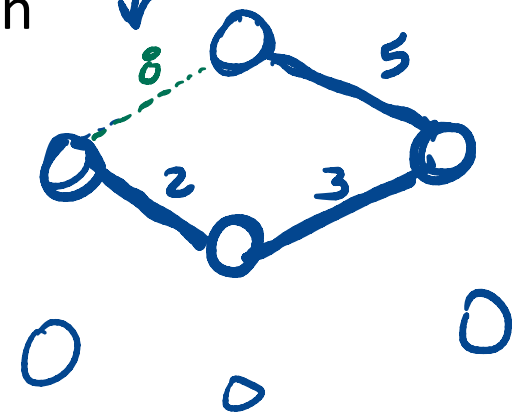
- Maintain and update an acyclic subgraph $F$ on the input graph $G$.
- At each step of the algorithm, we add some edges to $F$

↳ when is an edge "good" or "safe"

An edge is ***useless*** if it is not an edge in $F$ but both of its endpoints are in the same component of $F$.
Adding a useless edge to $F$ would induce a cycle, and can thus never be part of the MST we are building.



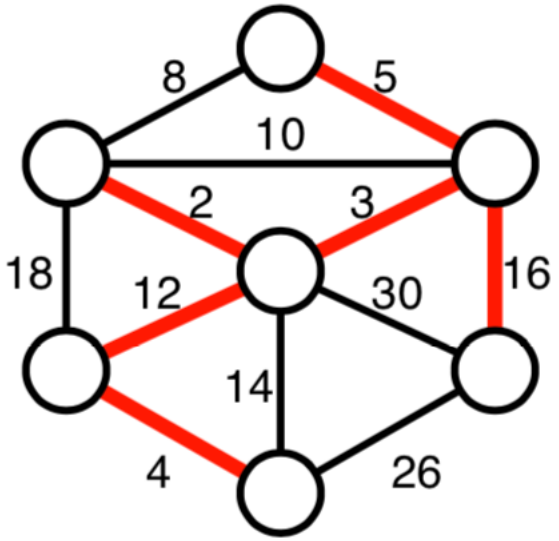

"useless"

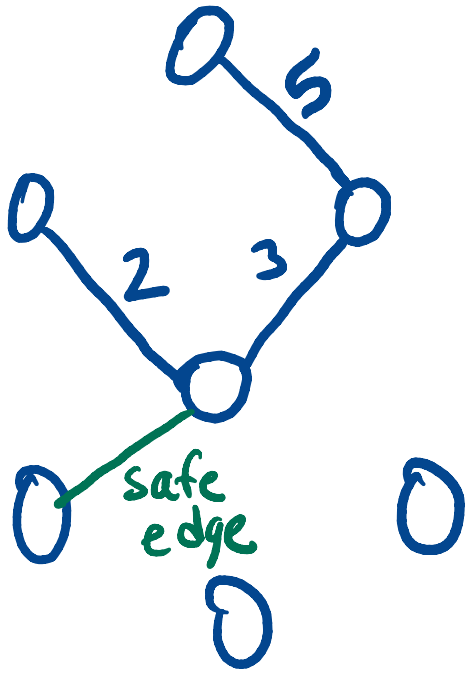


# Minimum Spanning Trees – Generic Algorithm

An edge is called ***safe*** if it is the minimum weight edge among all edges with exactly one endpoint in that component.

- If we assume that edge weights are distinct, a given component can only have one safe edge.




- Edges that are neither safe nor useless are called undecided edges.

# Minimum Spanning Trees – Generic Algorithm

```
Generic-MST(G,w) {
    F = {}                                          // initially F contains no edges
    while (F not a spanning tree) {                 // are we done yet?
      • find edge (u,v) that is safe for F          //    make a safe choice
      • add (u,v) to F                              //    and grow F
    }
    return F                                        // we are done
}
```

- in each step we add a safe edge
- stop when we have no more safe edges

- Prim's
  - grow the MST from some initial vertex until we span the graph
  - use a min-heap with the edge weights as keys
  - check edge to see if its useless or safe

- Krusksl's
  - Repeatedly adding the smallest weight edge
  - no cycles

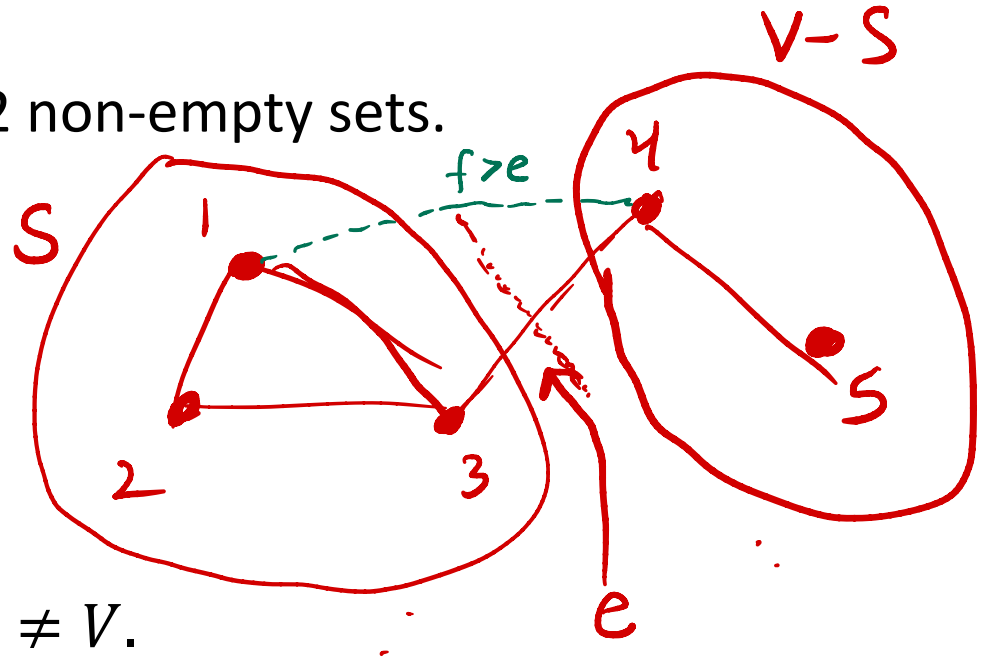
# Theory Behind MST Algorithms

- When is it safe to add an edge to an MST?

- How do we know that the edge selection produces a valid MST?

# Theory Behind MST Algorithms

A ***cut*** of a graph $G = (V, E)$ is a partition of $V$ into 2 non-empty sets.




**Cut Property:**

- Assume distinct edge costs.
- Let $S$ be a set of vertices that is not empty and $S \neq V$.
- Let edge $e = (v, w)$ be a minimum cost edge with $v \in S$ and $w \in V - S$

➢ Every MST contains edge $e$.

# Theory Behind MST Algorithms

**Example:** Prove the **Cut Property.**

Idea: Exchange Argument

- T - spanning tree
- Identify an edge $e'$ in T that is more expensive than e and exchange e and $e'$

# Theory Behind MST Algorithms

**Example:** Prove the **Cut Property.**

Proof: Let $e = (v, w)$ be the minimum cost edge with $v \in S$ and $w \in V - S$

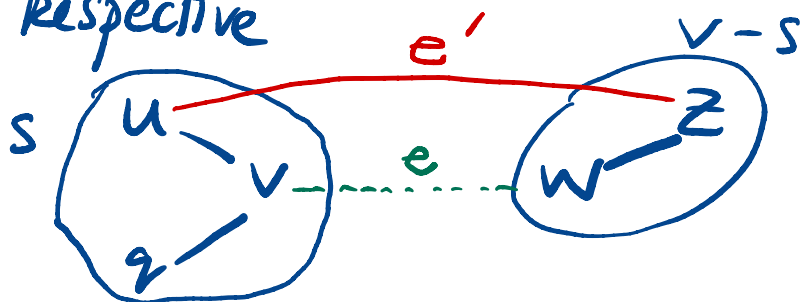
Let $T$ be a spanning tree that doesn't contain $e$.

Note, as a spanning tree $(V, T)$ is connected and acyclic.

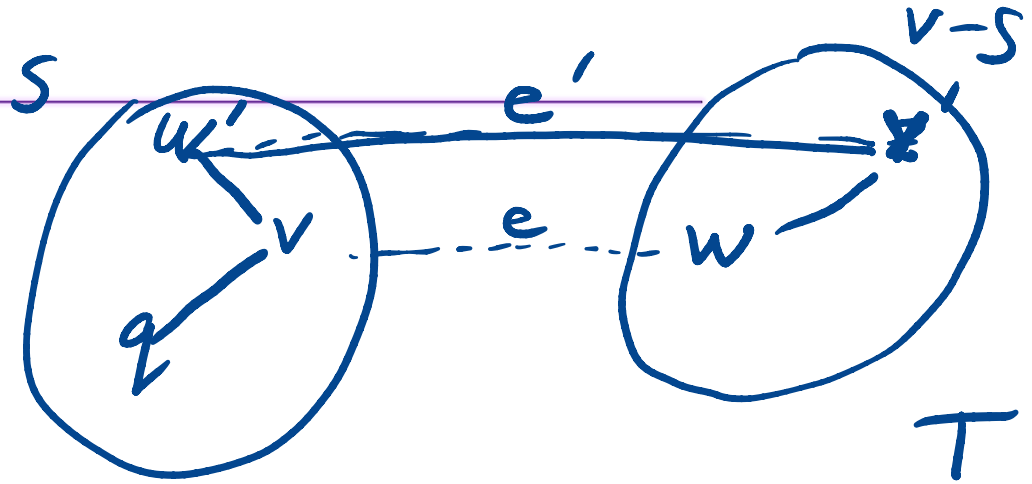
We'll show that $T$ is not the minimum possible cost.

The ends of $e$ are in $S$ and $V - S$ respectively.

We select an $e'$ that also has its respective edges in $S$ and $V - S$



# Theory Behind MST Algorithms




**Example:** Prove the **Cut Property.**

Let $u' \in S$ and $v' \in V-S$ such that $e' = (v', w')$ is in the spanning tree $T$

Exchange edge $e$ for edge $e'$ to get edge set $T'$ where $T' = T - \{e'\} \cup \{e\}$

Thus $T'$ contains edge $e$.

We know $(V, T')$ is connected because $(V, T)$ was connected.

# Theory Behind MST Algorithms

**Example:** Prove the **Cut Property.**

Path edge $e' = (v', w')$ in $T$ is replaced by $e = (v, w)$ in $T'$ where $v \in S$ and $w \in V - S$

Since the only edges connecting $S$ and $V-S$ were $e$ and $e'$ and $e'$ was removed, we know that $T'$ is acyclic.

Both $e$ and $e'$ have an end in $S$ and the other end in $V-S$. $e$ is cheaper $\Rightarrow$ $c_e < c_{e'}$

$\Rightarrow$ total cost of $T'$ < total cost of $T$

# Theory Behind MST Algorithms

**Cycle Property:**

- Let $c$ be any cycle in $G$ and let edge $e = (v, w)$ be the most expensive edge in the cycle $c$. Then, $e$ does not belong to any MST of $G$.

no proof

By contradiction proof of cut property

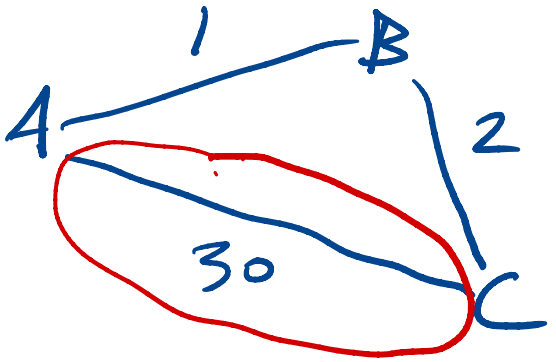
Pf: Suppose e belongs to some MST $T^*$

If we delete edge e from $T^*$, then $T^*$ is now disconnected

Suppose there is some other edge, f

$T = T^* \cup \{f\} - \{e\}$ is also a spanning tree. Since $c_f < c_e$

$cost(T) < cost(T^*)$. ✱ that $T^*$ is MST.



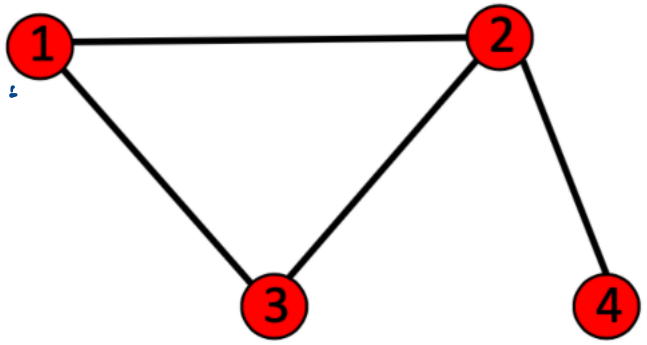

$\bar{AC}$ is not part of any MST

D E F

# Review? – Adjacency Matrices

- An ***adjacency matrix*** is one way to represent a graph.

**Example**: Write the adjacency matrix for the following unweighted, undirected graph.



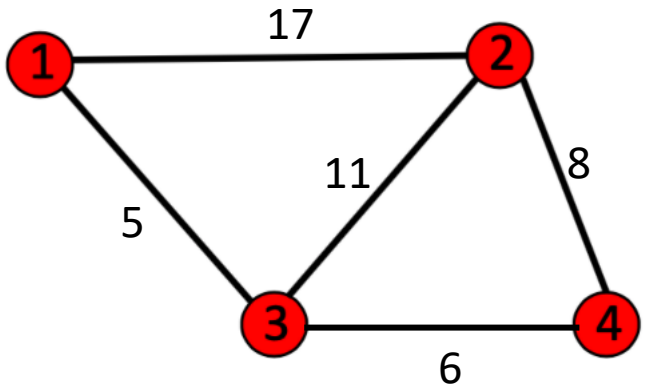

Let $a_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in E \\ 0 & \text{otherwise} \end{cases}$

$$A = \begin{array}{c} \\ \text{Row 1} \\ \text{Row 2} \\ \text{Row 3} \\ \text{Row 4} \end{array} \begin{array}{c} \begin{array}{cccc} \text{Column 1} & \text{Column 2} & \text{Column 3} & \text{Column 4} \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

# Review? – Adjacency Matrices

- An ***adjacency matrix*** is one way to represent a graph.

**Example**: Write the adjacency matrix for the following weighted, undirected graph.



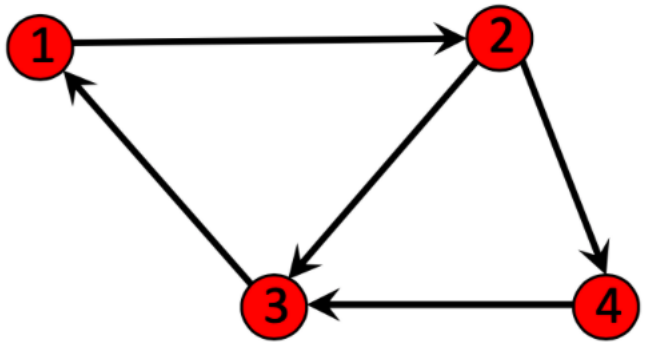

$$a_{ij} = \begin{cases} w_{ij} & \text{if } e_{ij} \in E \\ 0 & \text{otherwise} \end{cases}$$

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{bmatrix} 0 & 17 & 5 & 0 \\ 17 & 0 & 11 & 8 \\ 5 & 11 & 0 & 6 \\ 0 & 8 & 6 & 0 \end{bmatrix}$$

# Review? – Adjacency Matrices

- An ***adjacency matrix*** is one way to represent a graph.

**Example**: Write the adjacency matrix for the following undirected graph.




$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

not specified

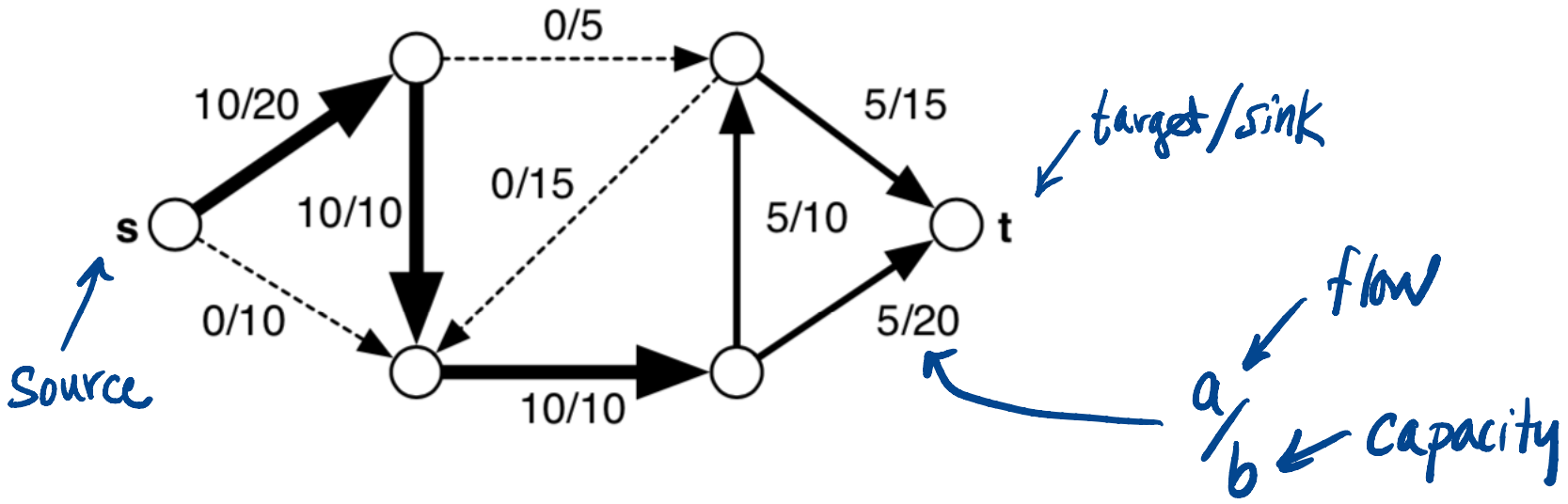
# Maximum Flow

In max-flow:

❖the input is:

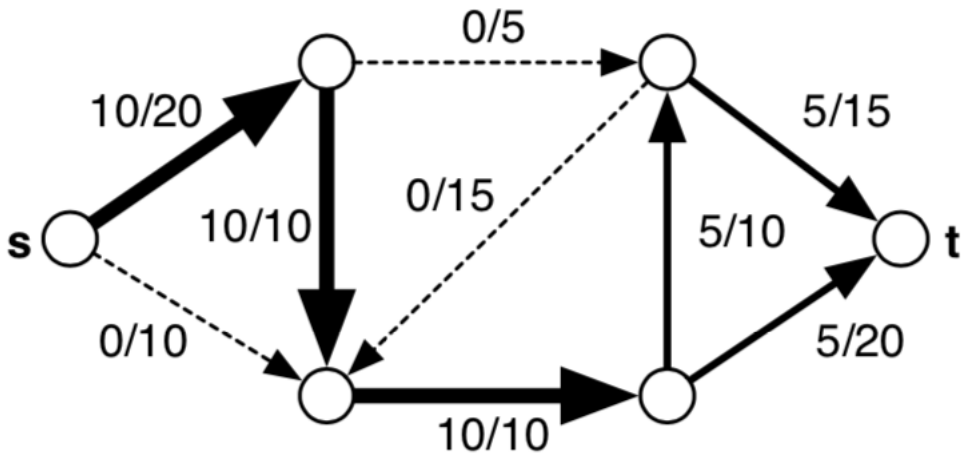
- a directed graph $G = (V, E)$,
- an edge weight function that represents the capacities of the edges,
- and a pair of vertices that represent the source and target /sink

❖The output is a flow function.



# Maximum Flow

***The maximum flow problem***: Given a graph $G$, capacity function $c$, and choices $s, t$, compute a feasible $(s,t) - flow$ on $G$ whose value $|f|$ is maximized.



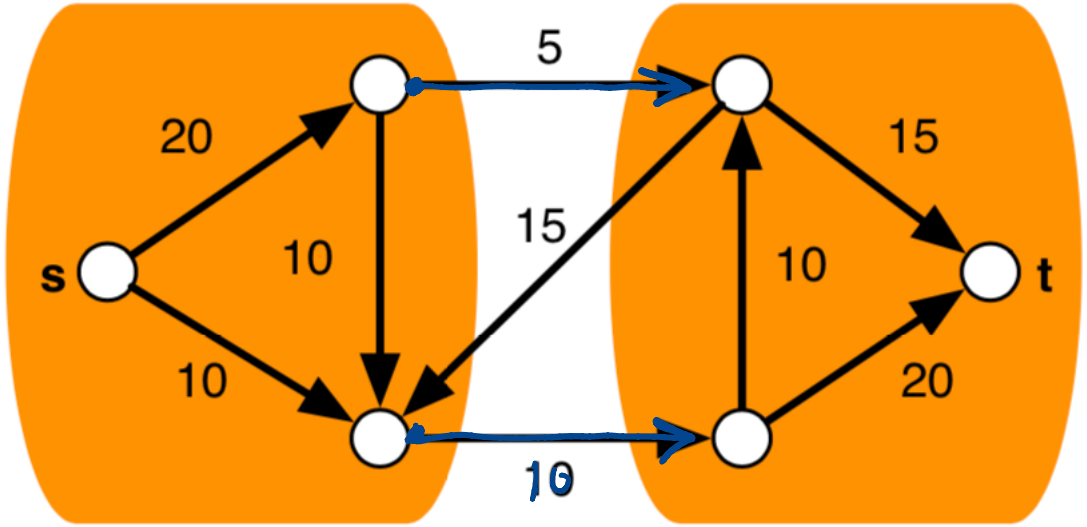

- Flow can only be created at the source $s$
- Flow can only be consumed at the target $t$

A flow is ***feasible*** with respect to the edge capacities $c$ if for every edge $e \in E$, the capacity of $e$ is not exceeded by the flow on $e$; $f(e) \leq c(e)$

# Minimum Cut

***The minimum cut problem***: Given a graph $G$, a capacity function $c$, and choices $s, t$, find an $(s, t) - cut$ whose capacity $\|S, T\|$ is minimized.

A $(\boldsymbol{s},\ \boldsymbol{t}) - \boldsymbol{cut}$ on $G$ is a bipartitioning of the vertices $V$ into subsets $S$ and $T$, with $s \in S$ and $t \in T$

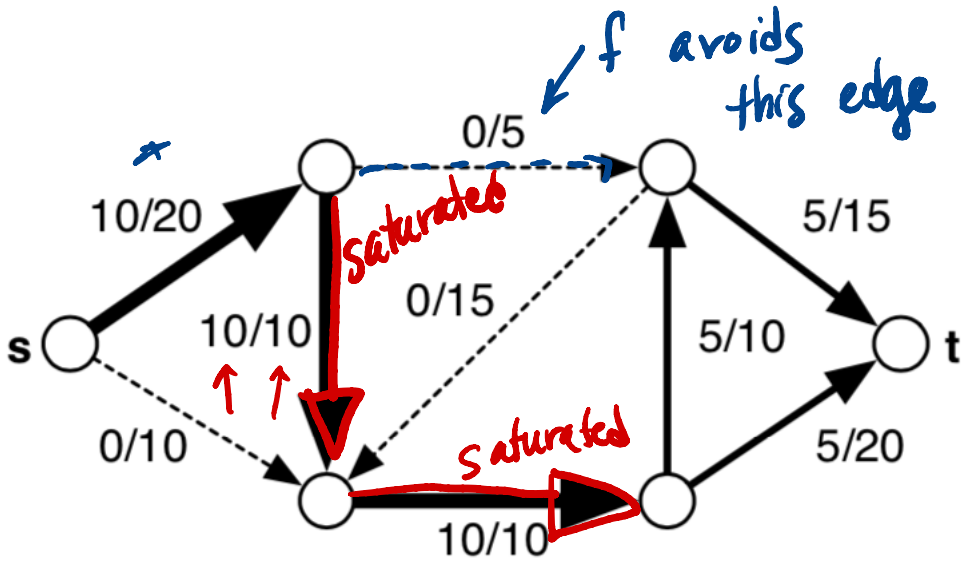



The ***capacity*** of the cut is the total capacity of the edges "in" the cut, which is defined as edges that start in $S$ and end in $T$.

$\|S, T\| = 15$

# Max–Flow Min–Cut

- If $f(e) = c(e)$, then we say that $f$ ***saturates*** the edge $e$.
- If $f(e) = 0$, then we say that $f$ ***avoids*** the edge $e$



# Max_Flow Min_Cut

***Max-Flow Min-Cut Theorem***: The value of the max flow equals the capacity of the min cut.

A simple greedy approach:

- Saturate the largest capacity path from $s$ to $t$, recurse until no more flow can be pushed through.