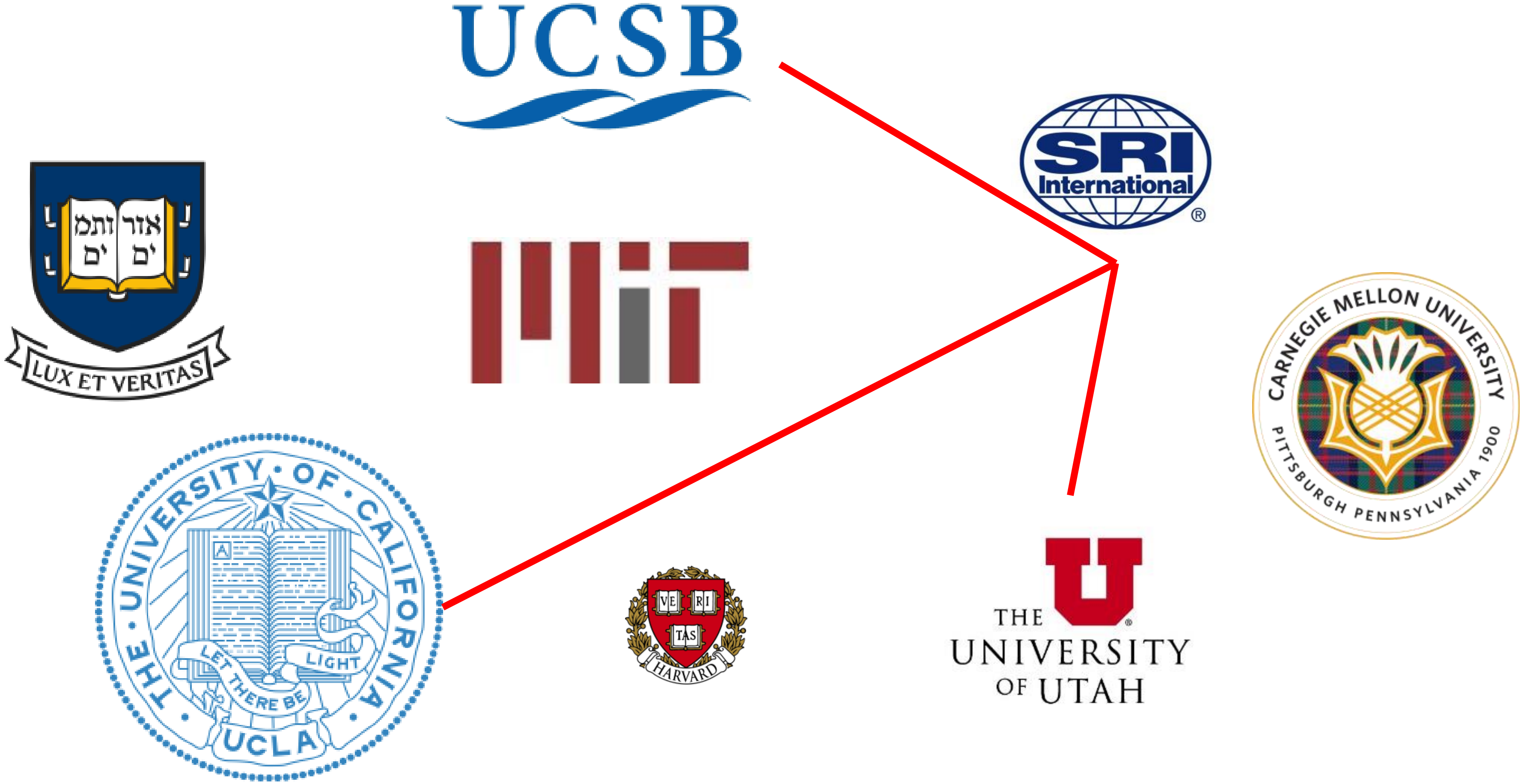
Minimum Spanning Trees (MST)

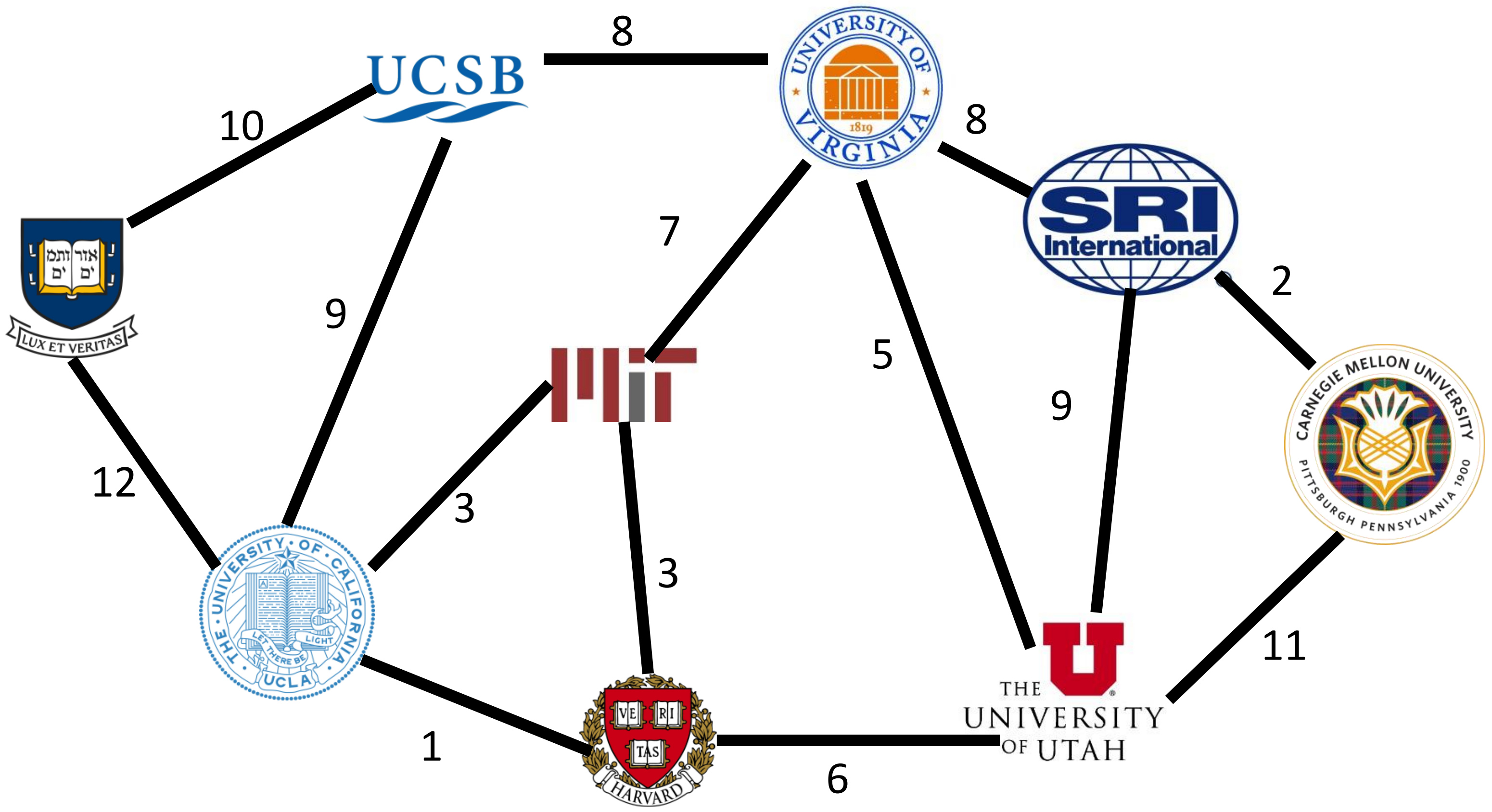
ETHAN BLASER (GRAD TA)

MODIFIED SLIDES FROM PROF. BRUNELLE

ARPANET


UCSB
SRI International
LUX ET VERITAS
CARNEGIE MELLON UNIVERSITY
PITTSBURGH PENNSYLVANIA 1900
THE UNIVERSITY OF CALIFORNIA
LET THERE BE LIGHT
UCLA
VE RI TAS
HARVARD
THE UNIVERSITY OF UTAH

Problem



Find a
Minimum
Spanning Tree

We need to connect together all these places into a network
We have feasible wires to run, plus the cost of each wire
Find the cheapest set of wires to run to connect all places

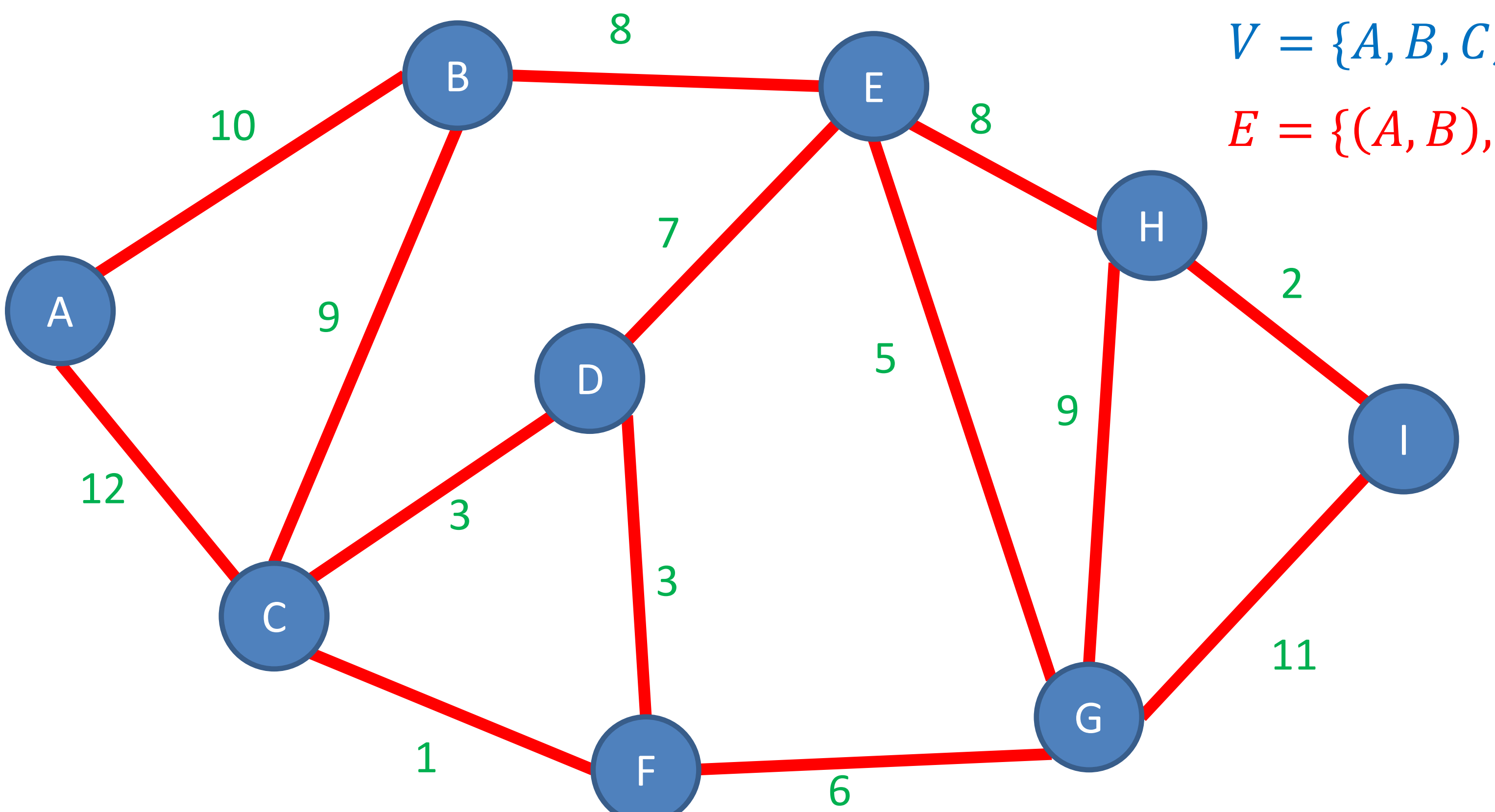
Undirected Graphs

Vertices/Nodes

Definition: $G = (V, E)$

Edges

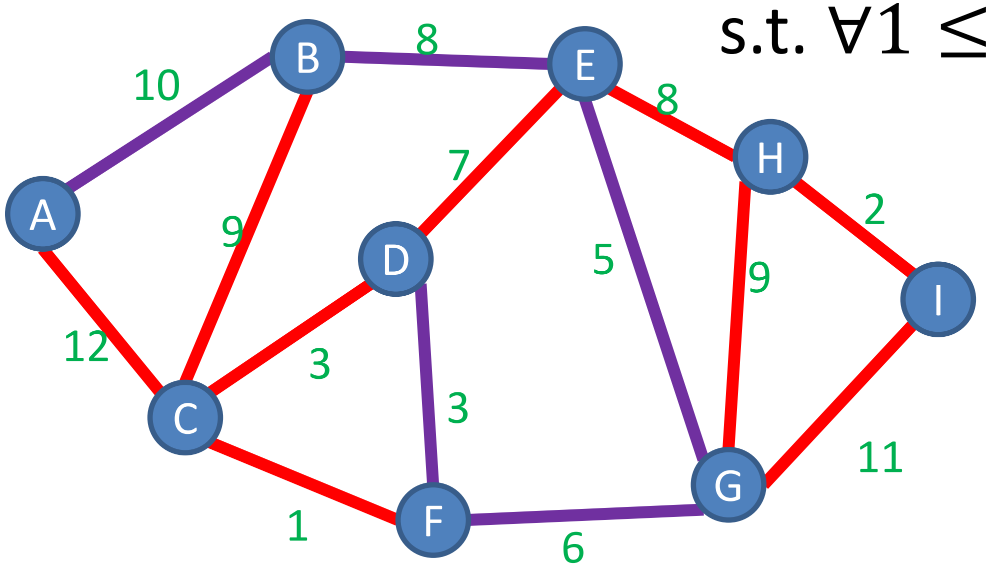
$w(e)$ = weight of edge e



$V = \{A, B, C, D, E, F, G, H, I\}$

$E = \{(A, B), (A, C), (B, C), \dots\}$

Definition: Path



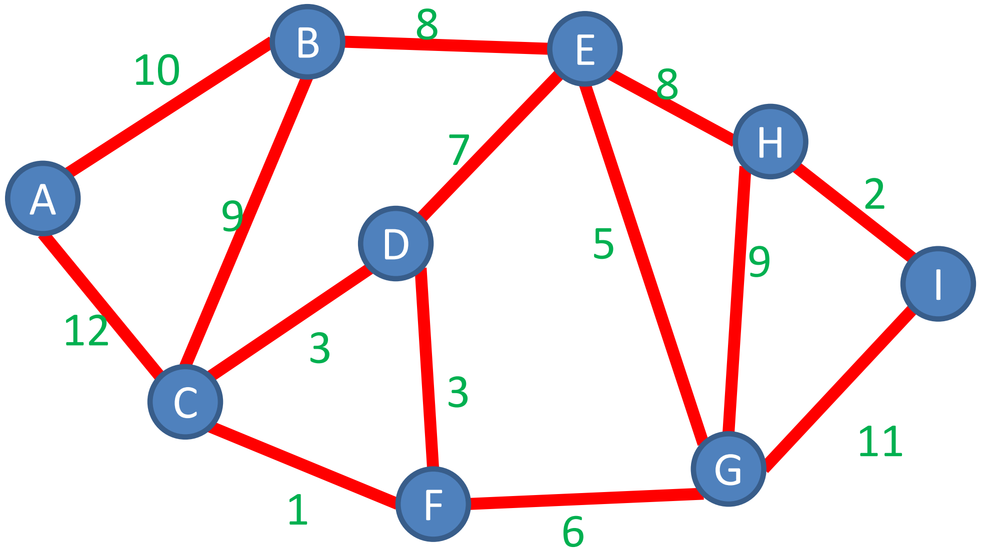
A sequence of nodes $(v_1, v_2, \dots, v_k)$
s.t. $\forall 1 \le i \le k-1, (v_i, v_{i+1}) \in E$

<u>Simple Path:</u>
A path in which each node appears at most once

<u>Cycle:</u>
A path of > 2 nodes in which $v_1 = v_k$

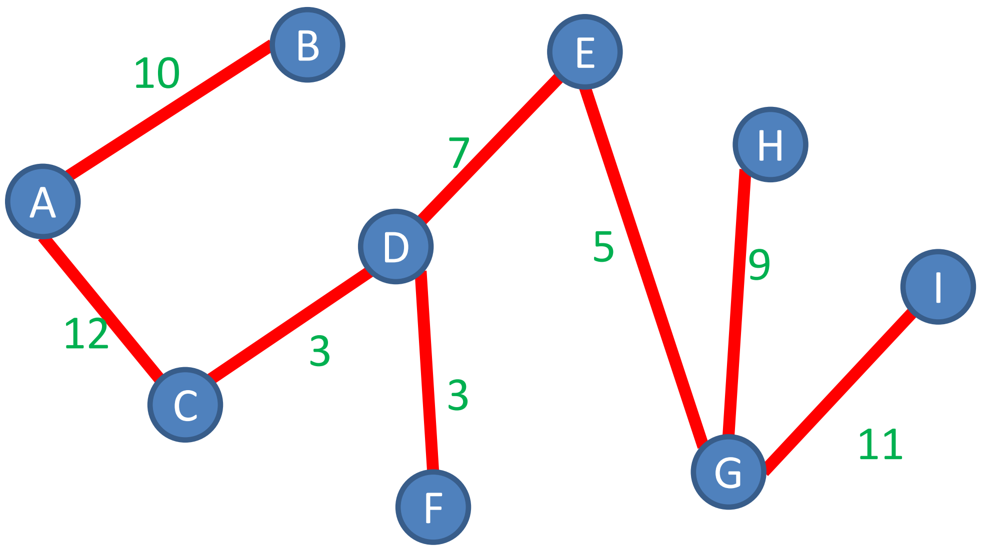
Definition: Connected Graph

A Graph $G = (V, E)$ s.t. for any pair of nodes $v_1, v_2 \in V$ there is a path from v_1 to v_2



Definition: Tree

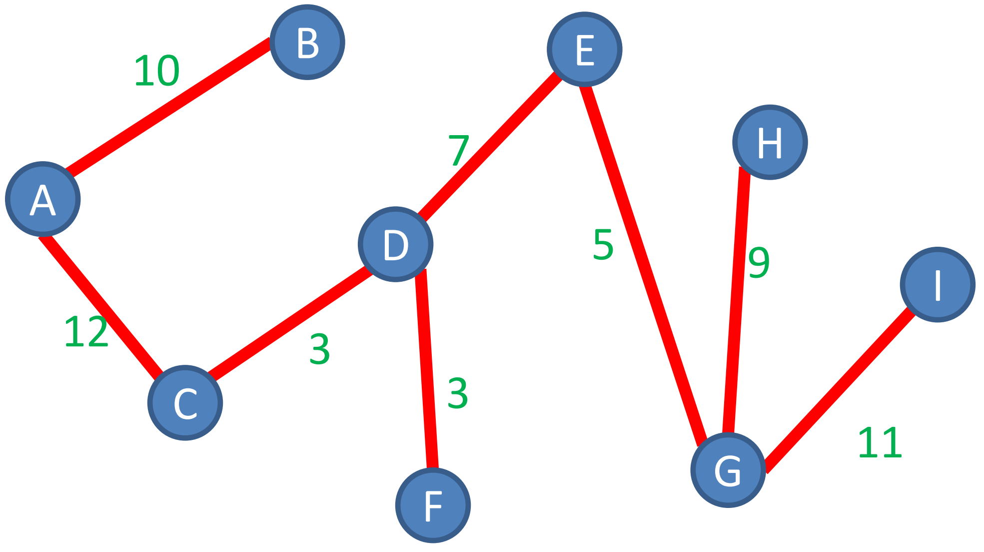
A connected graph with no cycles



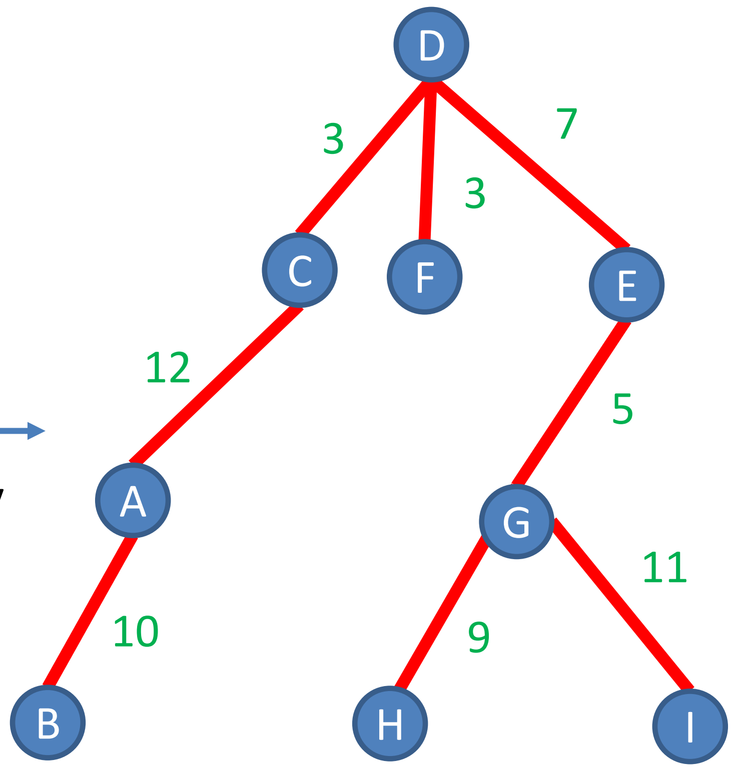
Note: A tree does not need a root, but they often do!

Definition: Tree

A connected graph with no cycles

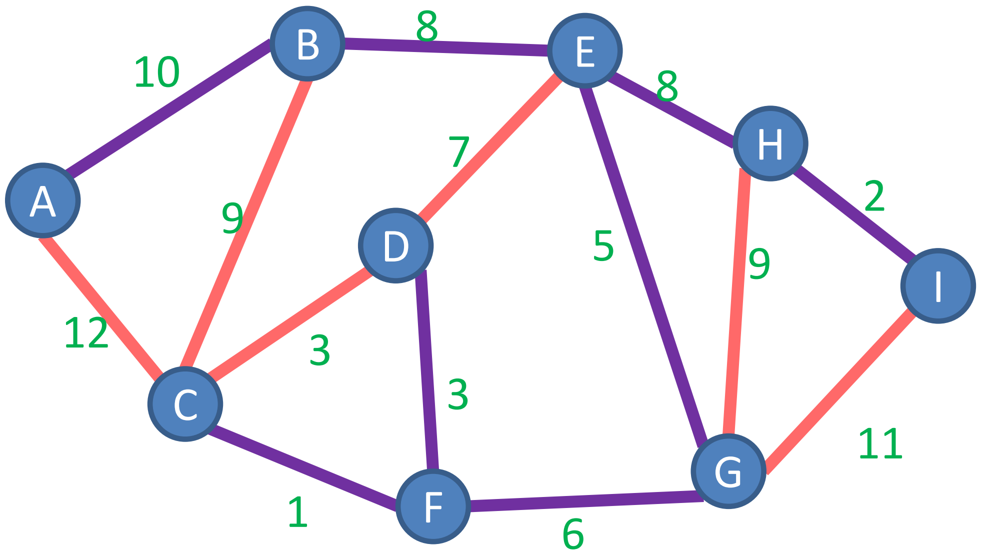


Pick some arbitrary root node and rearrange tree



Definition: Spanning Tree

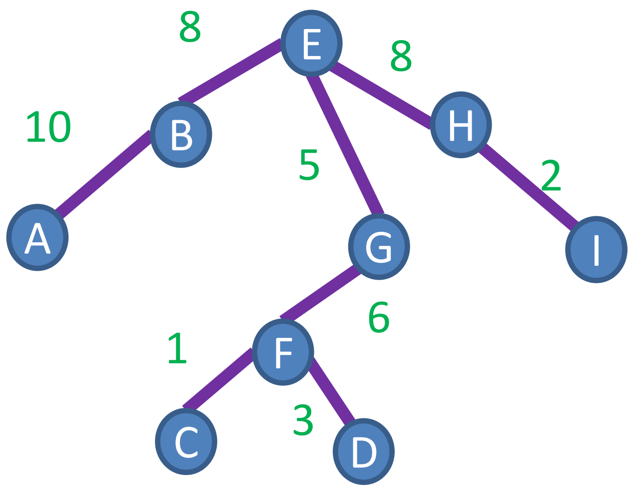
A Tree $\boldsymbol{T} = (V_T, E_T)$ which connects ("spans") all the nodes in a graph $G = (V, E)$



How many edges does T have?

$V - 1$

Pick some arbitrary root node and rearrange tree

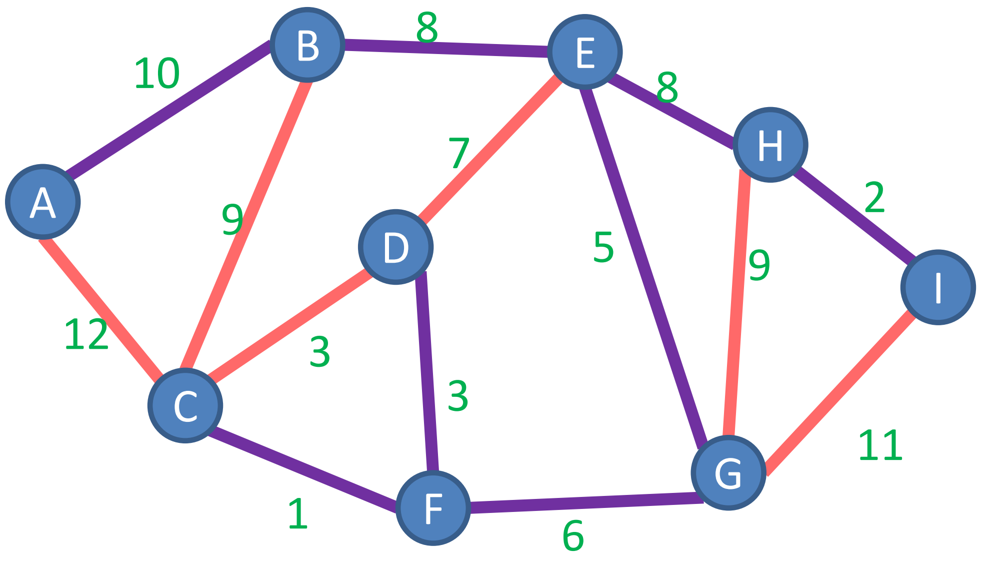


Any set of V-1 edges in the graph that doesn't have any cycles is guaranteed to be a spanning tree!

Any set of V-1 edges that connects all the nodes in the graph is guaranteed to be a spanning tree!

Definition: Minimum Spanning Tree

A Tree $\boldsymbol{T} = (V_T, E_T)$ which connects ("spans") all the nodes in a graph $G = (V, E)$, that has minimal cost



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

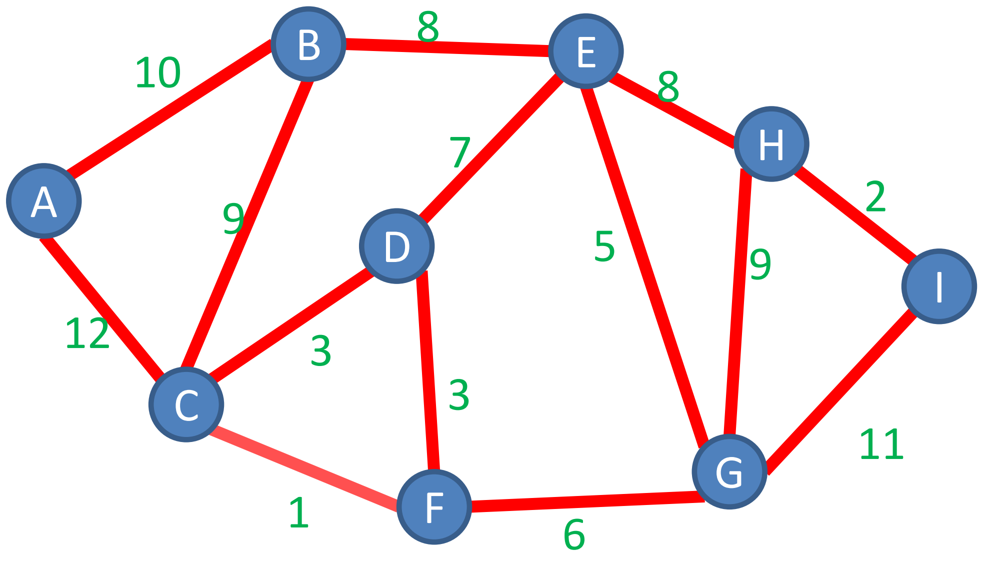
Greedy Algorithms

- Require Optimal Substructure
  - Solution to larger problem contains the solution to a smaller one
  - Only one subproblem to consider!
- Idea:
  1. Identify a greedy choice property
     - How to make a choice guaranteed to be included in some optimal solution
  2. Repeatedly apply the choice property until no subproblems remain

Kruskal’s Algorithm

Start with an empty tree A

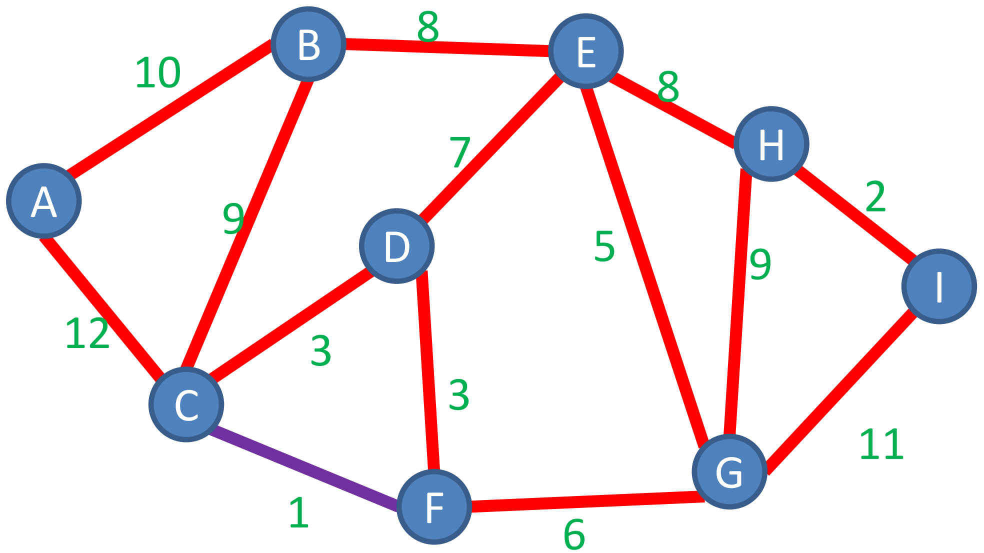
Add to A the lowest-weight edge that does not create a cycle



Kruskal's Algorithm

Start with an empty tree A

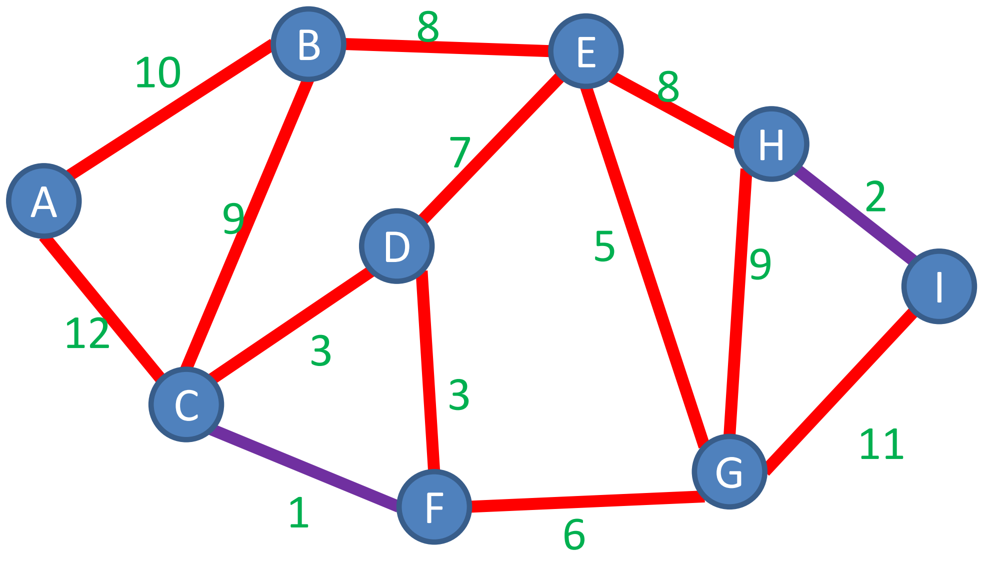
Add to A the lowest-weight edge that does not create a cycle



Kruskal's Algorithm

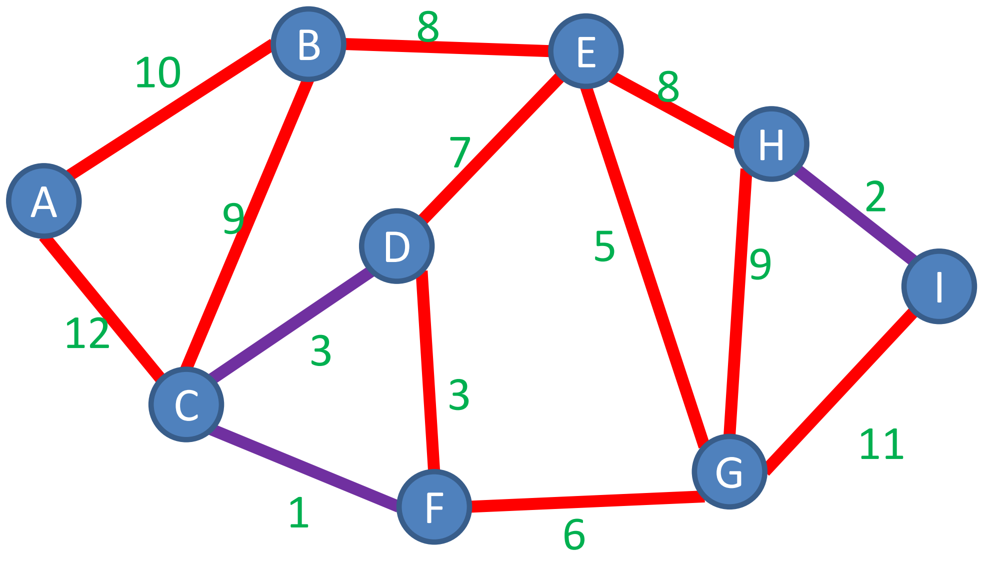
Start with an empty tree A

Add to A the lowest-weight edge that does not create a cycle



Kruskal’s Algorithm

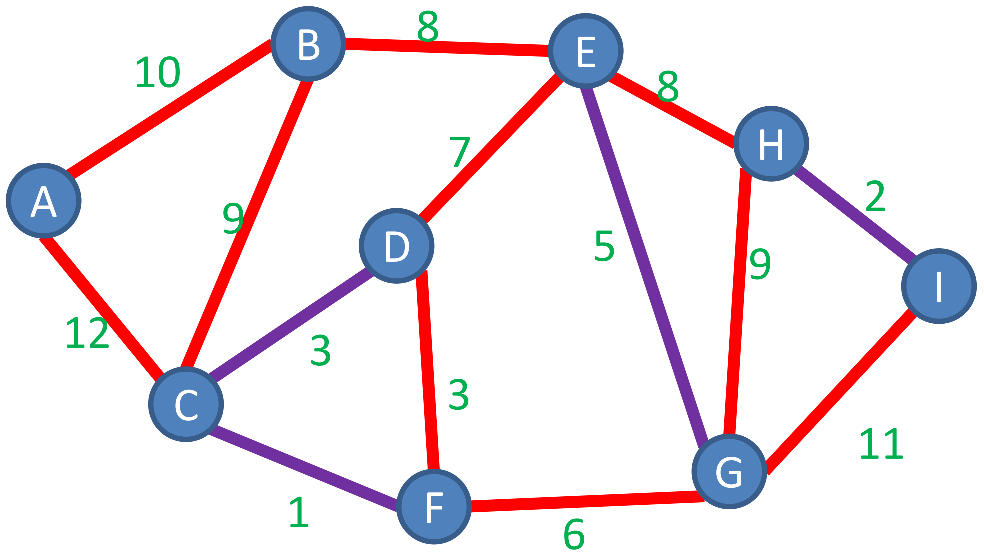
Start with an empty tree A
Add to A the lowest-weight edge that does not create a cycle



Kruskal's Algorithm

Start with an empty tree A

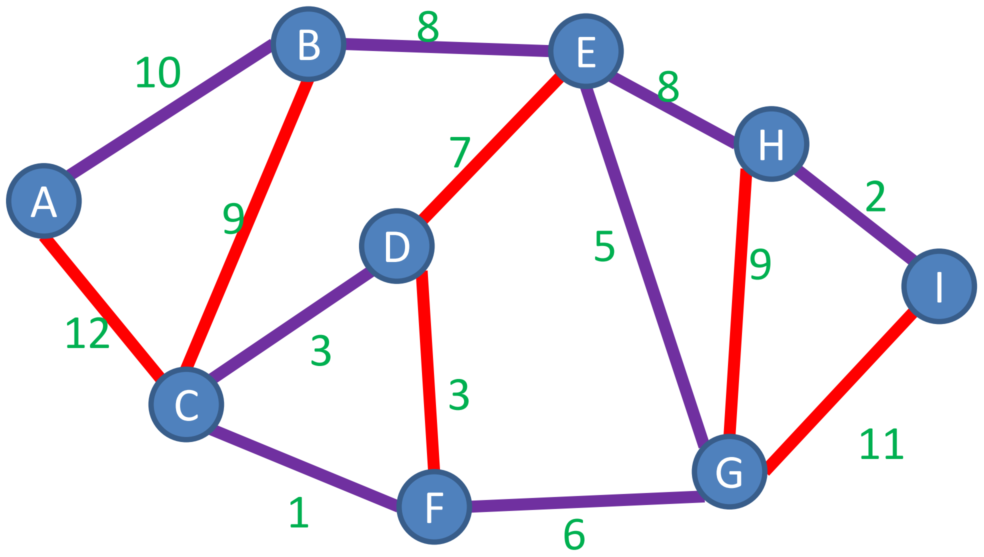
Add to A the lowest-weight edge that does not create a cycle



Kruskal's Algorithm

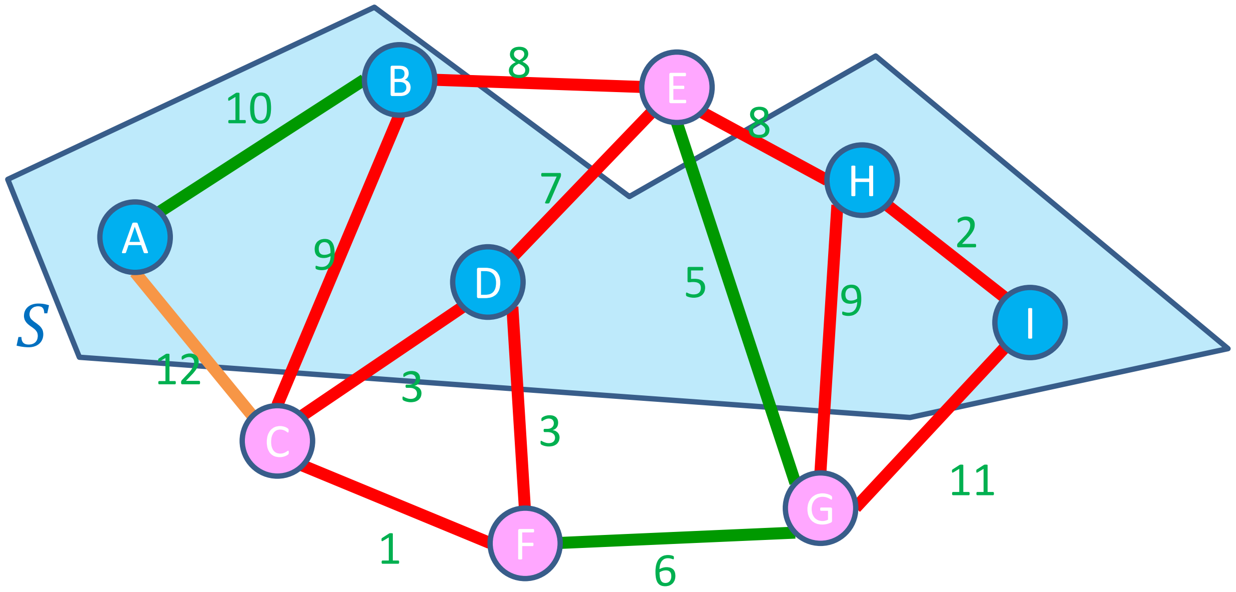
Start with an empty tree A

Add to A the lowest-weight edge that does not create a cycle



Definition: Cut

A Cut of graph $G = (V, E)$ is a partition of the nodes into two sets, S and $V - S$



Edge $(v_1, v_2) \in E$ crosses a cut if $v_1 \in S$ and $v_2 \in V - S$ (or opposite), e.g. (A, C)

A set of edges R Respects a cut if no edges cross the cut e.g. $R = \{(A, B), (E, G), (F, G)\}$

Exchange argument

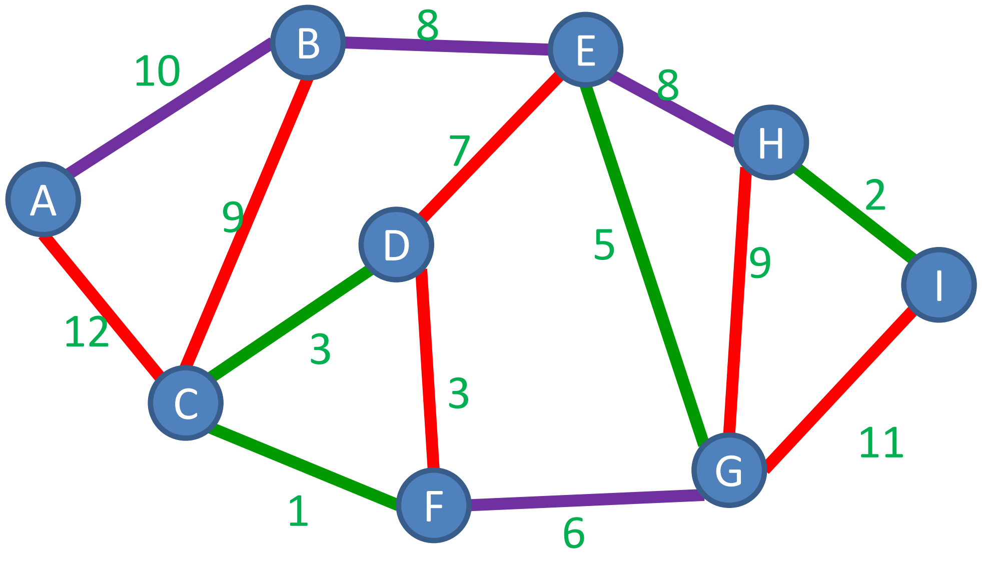
- Shows correctness of a greedy algorithm
- Idea:
  – Show exchanging an item from an arbitrary optimal solution with your greedy choice makes the new solution no worse
  – How to show my sandwich is at least as good as yours:
    - Show: “I can remove any item from your sandwich, and it would be no worse by replacing it with the same item from my sandwich”

Cut Theorem

If a set of edges A is a subset of a minimum spanning tree T, let $(S, V - S)$ be any cut which A respects. Let e be the least-weight edge which crosses $(S, V - S)$. $A \cup \{e\}$ is also a subset of a minimum spanning tree.

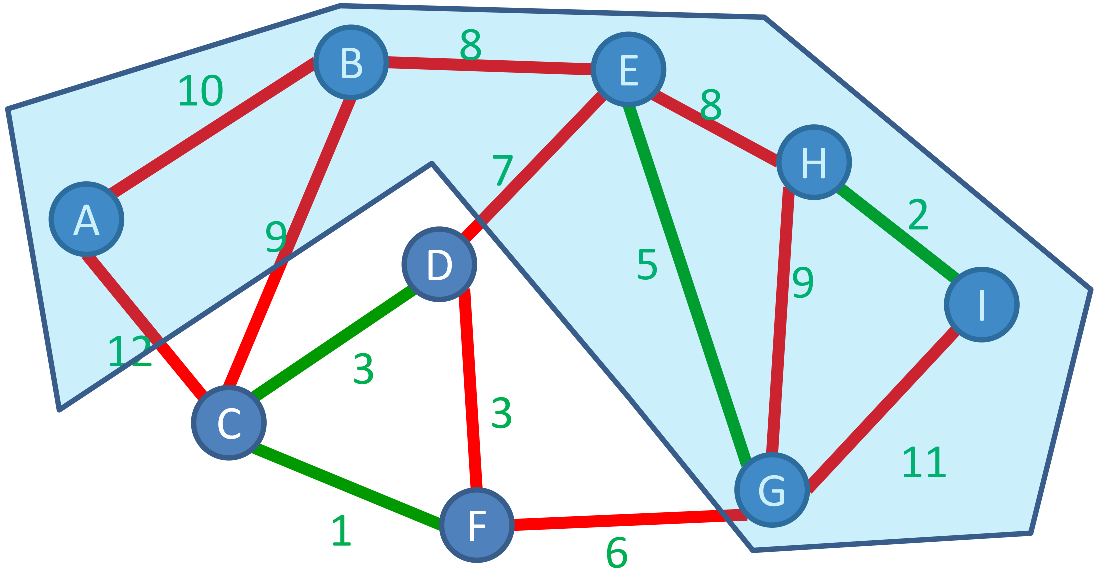
Cut Theorem

If a set of edges A is a subset of a minimum spanning tree T, let $(S, V - S)$ be any cut which A respects. Let e be the least-weight edge which crosses $(S, V - S)$. $A \cup \{e\}$ is also a subset of a minimum spanning tree.



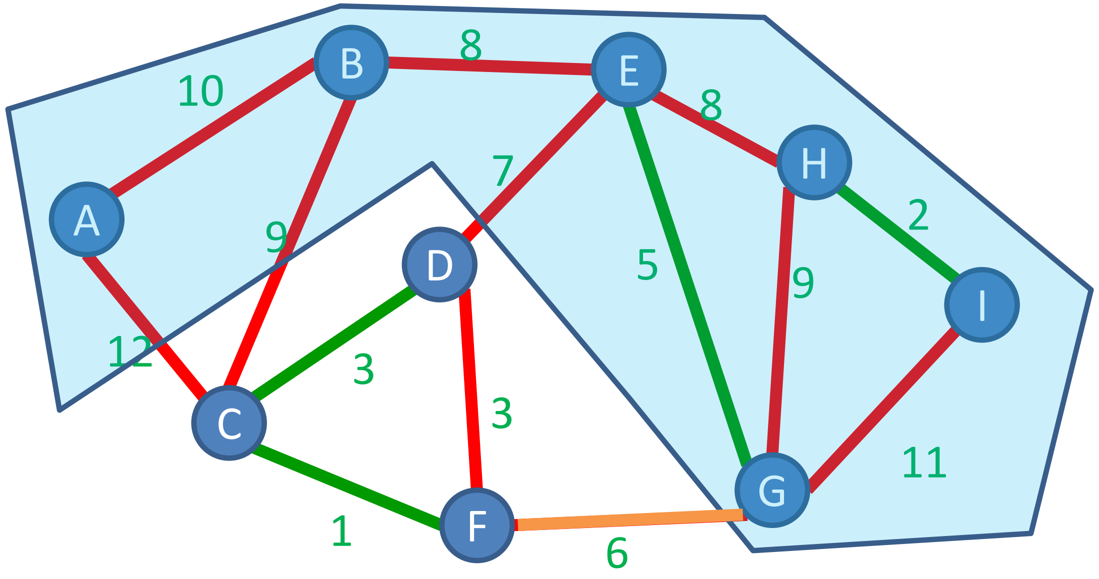
Cut Theorem

If a set of edges A is a subset of a minimum spanning tree T, let $(S, V - S)$ be any cut which A respects. Let e be the least-weight edge which crosses $(S, V - S)$. $A \cup \{e\}$ is also a subset of a minimum spanning tree.



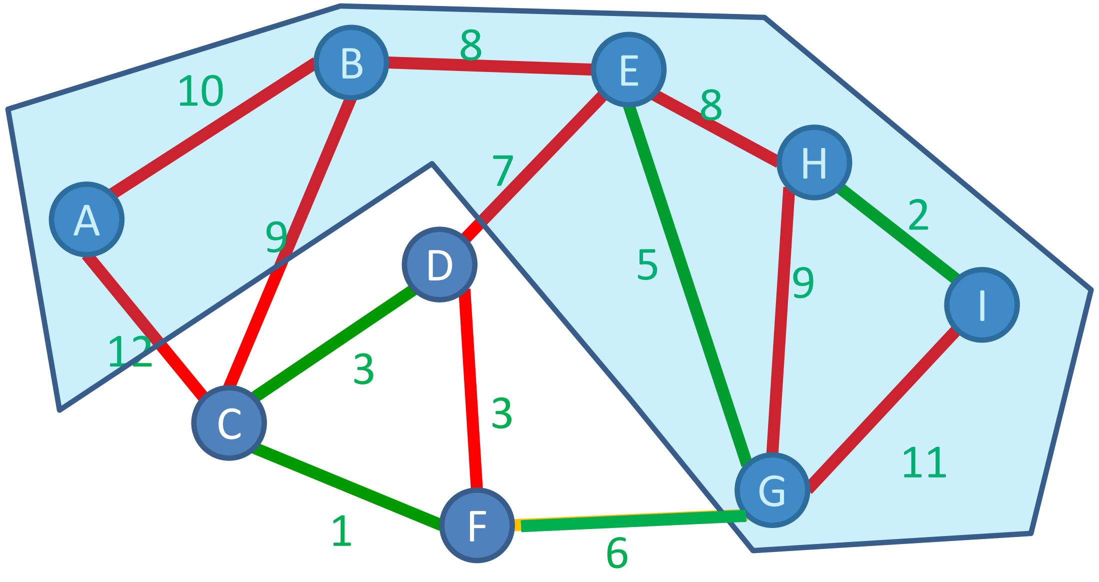
Cut Theorem

If a set of edges A is a subset of a minimum spanning tree T, let $(S, V - S)$ be any cut which A respects. Let e be the least-weight edge which crosses $(S, V - S)$. $A \cup \{e\}$ is also a subset of a minimum spanning tree.



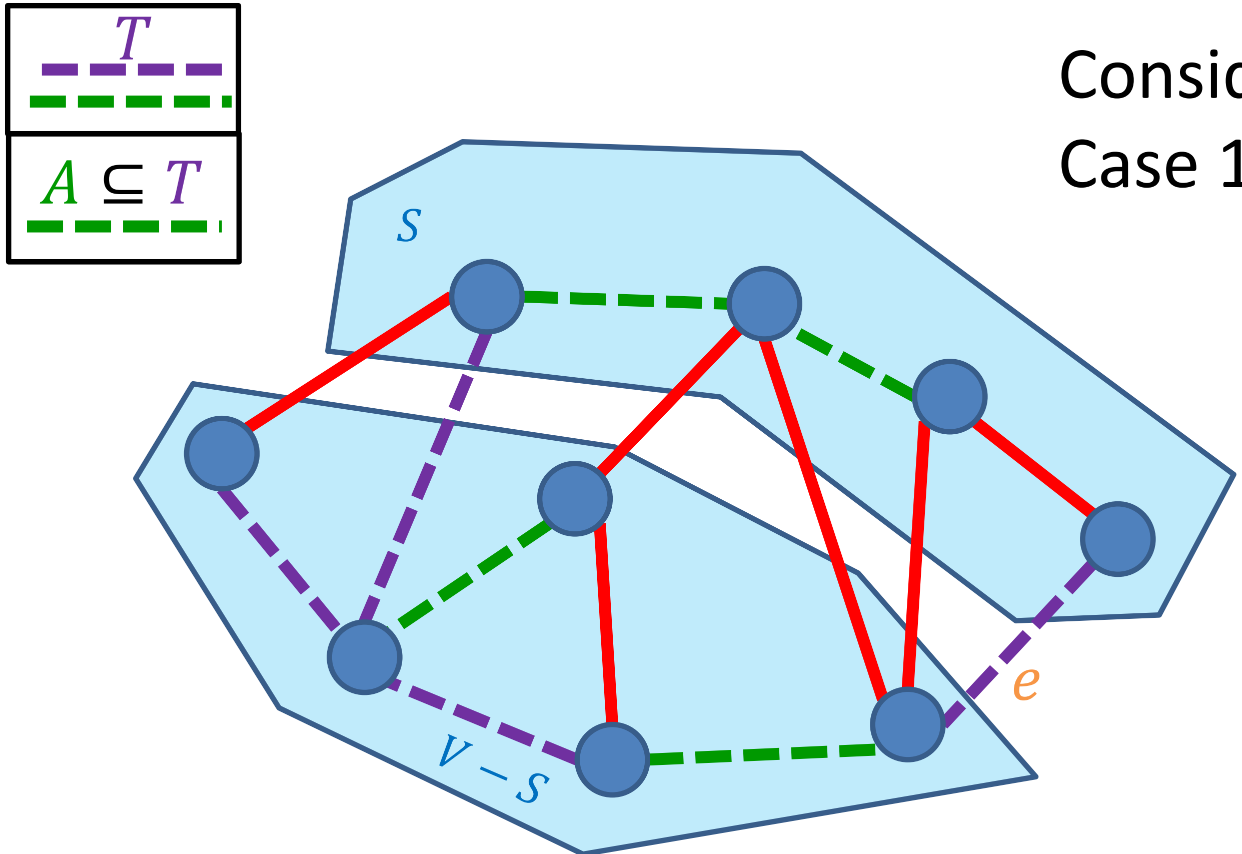
Cut Theorem

If a set of edges A is a subset of a minimum spanning tree T, let $(S, V - S)$ be any cut which A respects. Let e be the least-weight edge which crosses $(S, V - S)$. $A \cup \{e\}$ is also a subset of a minimum spanning tree.



Proof of Cut Theorem

Claim: If A is a subset of a MST T, and e is the least-weight edge which crosses cut $(S, V - S)$ (which A respects) then $A \cup \{e\}$ is also a subset of a MST.



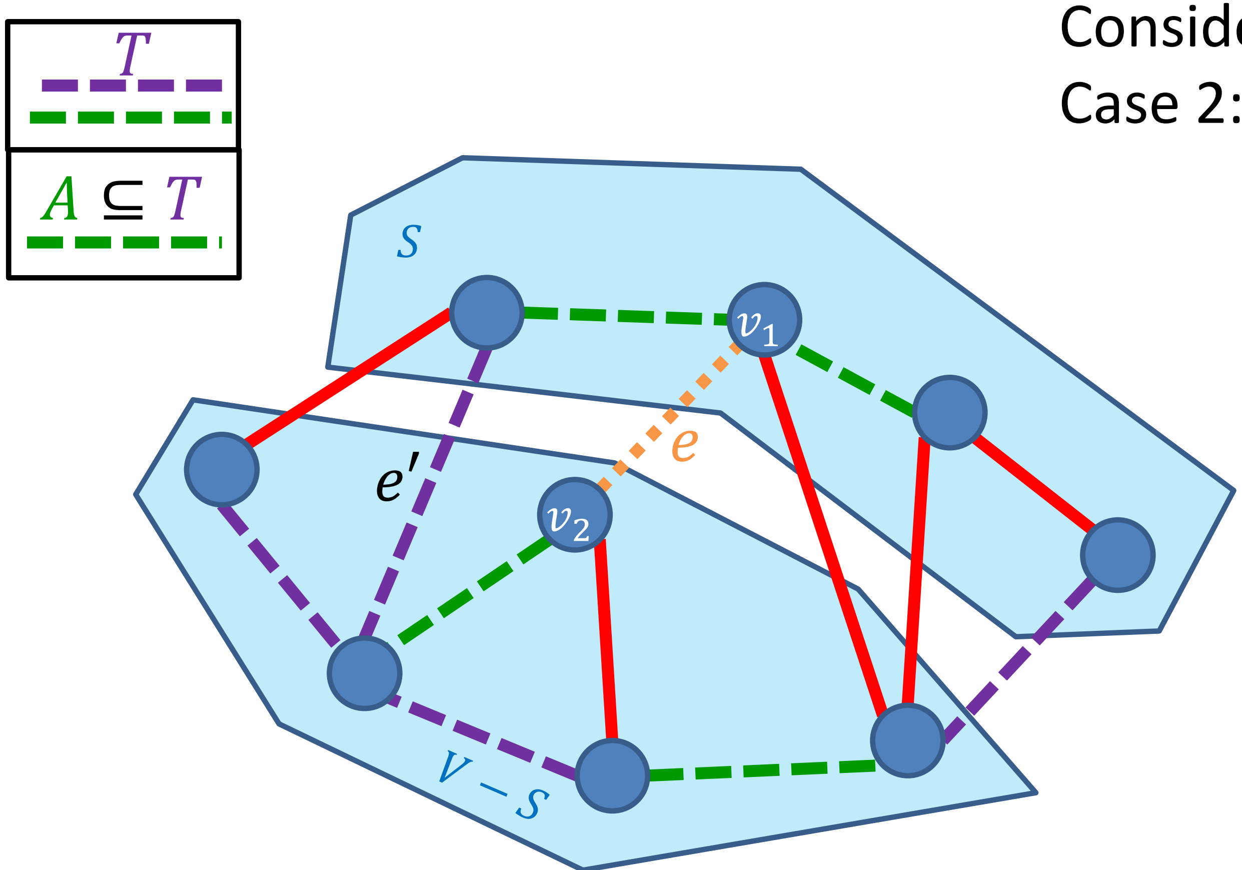
Consider some MST T,

Case 1: (the easy case)

If $e \in T$ Then claim holds

Proof of Cut Theorem

Claim: If A is a subset of a MST T, and e is the least-weight edge which crosses cut $(S, V - S)$ (which A respects) then $A \cup \{e\}$ is also a subset of a MST.



Consider some MST T,

Case 2:

Consider if $e = (v_1, v_2) \notin T$

Since T is a MST, there is some path from v_1 to v_2.

Let e' be the first edge on this path which crosses the cut

Build tree T' by exchanging e' for e

Proof of Cut Theorem

- We must show the following 2 things after building a new tree T':

1. The sum of the weights did not go up
   - $w(T') \leq w(T)$
2. T' is still a spanning tree

Proof of Cut Theorem

Claim: If A is a subset of a MST T, and e is the least-weight edge which crosses cut $(S, V - S)$ (which A respects) then $A \cup \{e\}$ is also a subset of a MST.

Consider some MST T,

Case 2:

Consider if $e = (v_1, v_2) \notin T$

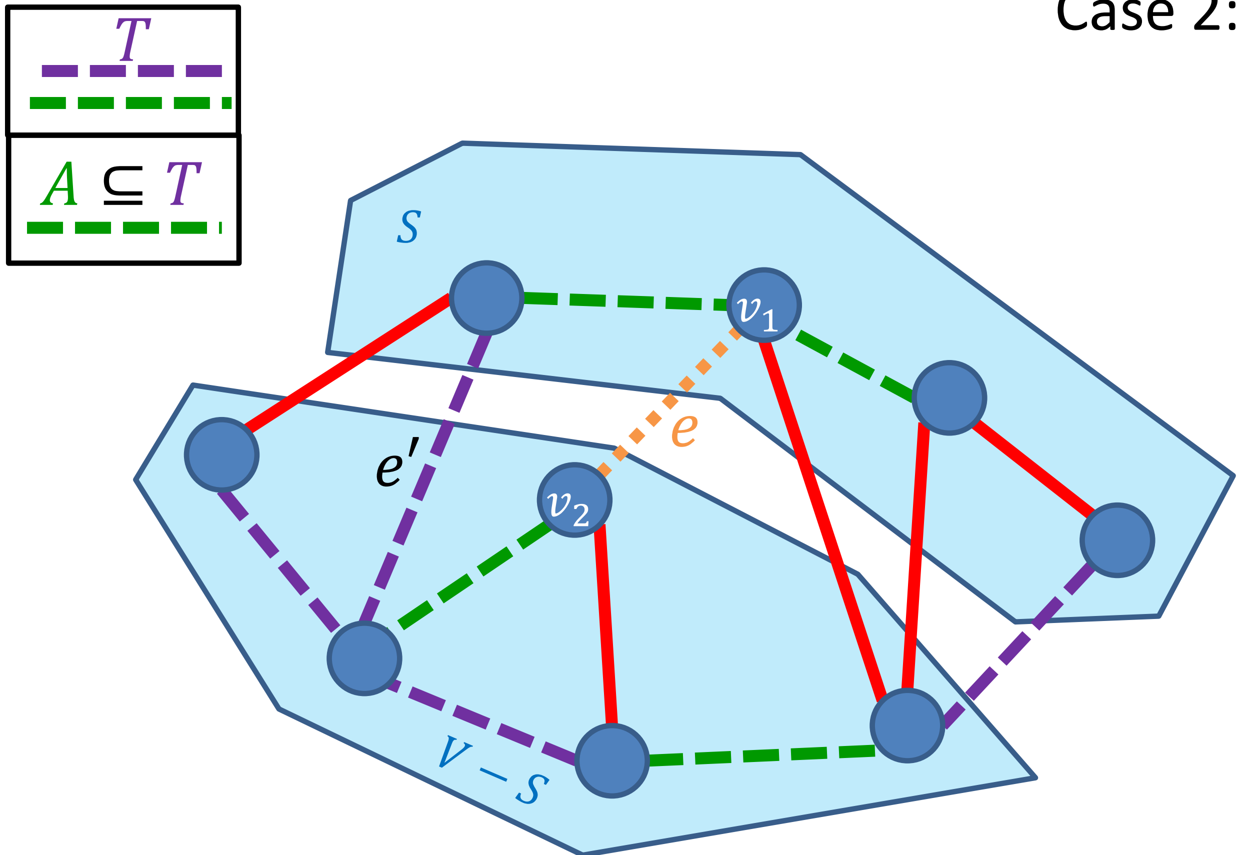
$T' = T$ with edge e instead of e'

We assumed $w(e) \leq w(e')$

$w(T') = w(T) - w(e') + w(e)$

$w(T') \leq w(T)$

We have proven the cost of T' is less than or equal to T



Proof of Cut Theorem

- We must show the following 2 things after building a new tree T':

1. The sum of the weights did not go up ✓
   - We proved that $w(T') \leq w(T)$
2. T' is still a spanning tree. How do we prove this?

Any set of V-1 edges that connects all the nodes in the graph is guaranteed to be a spanning tree!

Any set of V-1 edges in the graph that doesn't have any cycles is guaranteed to be a spanning tree!

Proof of Cut Theorem

- We must show the following 2 things after building a new tree T':

1. The sum of the weights did not go up
   – We proved that $w(T') \leq w(T)$
2. T' is still a spanning tree
   – T' has V-1 edges
   – T' connects all the nodes in the graph

Proof of Cut Theorem

Claim: If A is a subset of a MST T, and e is the least-weight edge which crosses cut $(S, V - S)$ (which A respects) then $A \cup \{e\}$ is also a subset of a MST.

Consider some MST T,

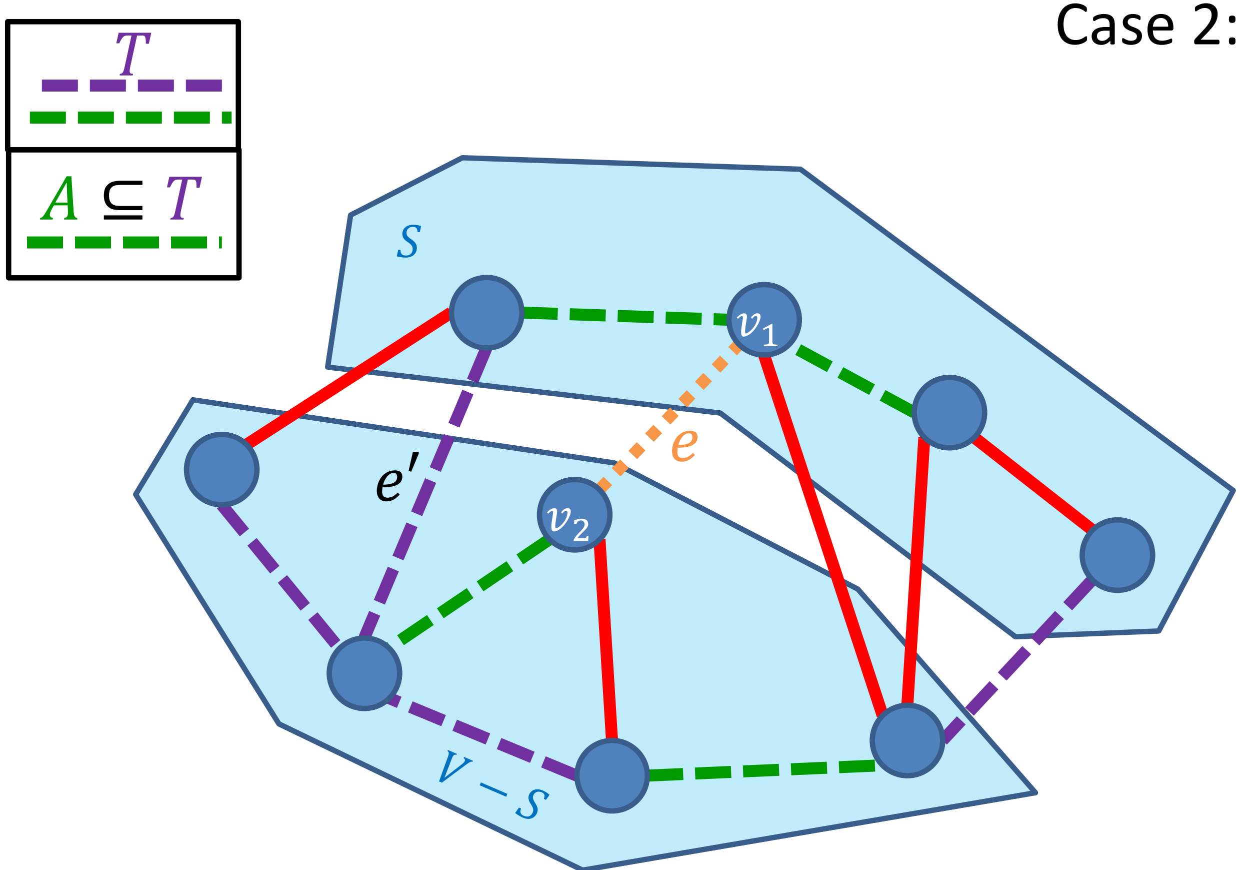
Case 2:

Consider if $e = (v_1, v_2) \notin T$

$T' = T$ with edge e instead of e'

We know T has $V - 1$ edges

We did a 1 for 1 edge swap so it must be the case that T' also has $V - 1$ edges



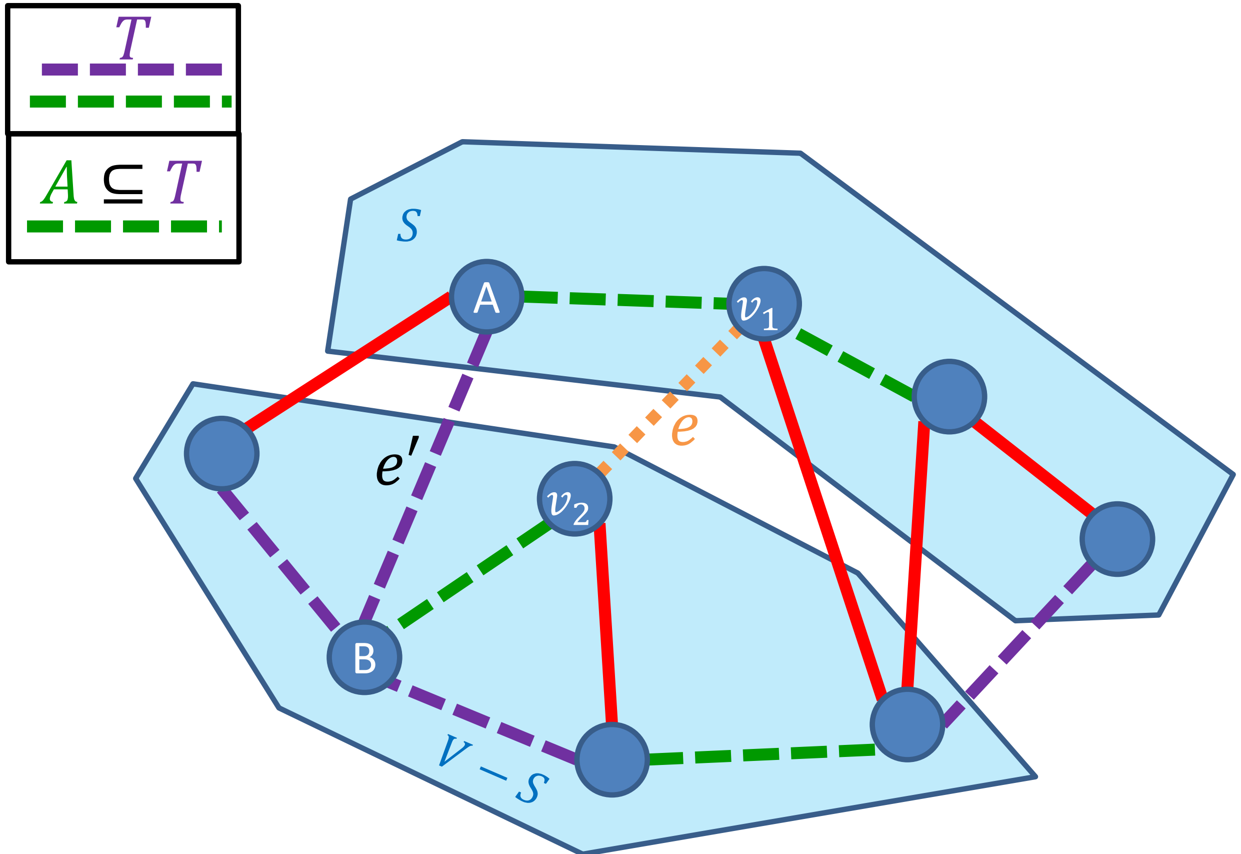
Proof of Cut Theorem

- We must show the following 2 things after building a new tree T':

1. The sum of the weights did not go up ✓
   - We proved that $w(T') \leq w(T)$
2. T' is still a spanning tree
   - T' has V-1 edges ✓
   - T' connects all the nodes in the graph

Proof of Cut Theorem

Claim: If A is a subset of a MST T, and e is the least-weight edge which crosses cut $(S, V - S)$ (which A respects) then $A \cup \{e\}$ is also a subset of a MST.



Consider some MST T,

Consider if $e = (v_1, v_2) \notin T$

$T' = T$ with edge e instead of e'

Show that we didn't disconnect the end points of e' which we will call (A, B):

There was some path in T from $v_1$to v_2 that uses e'. So, there must be a path from v_1 to A and B to v_2.

We have connected v_1 and v_2 with e, so we're good!

Proof of Cut Theorem

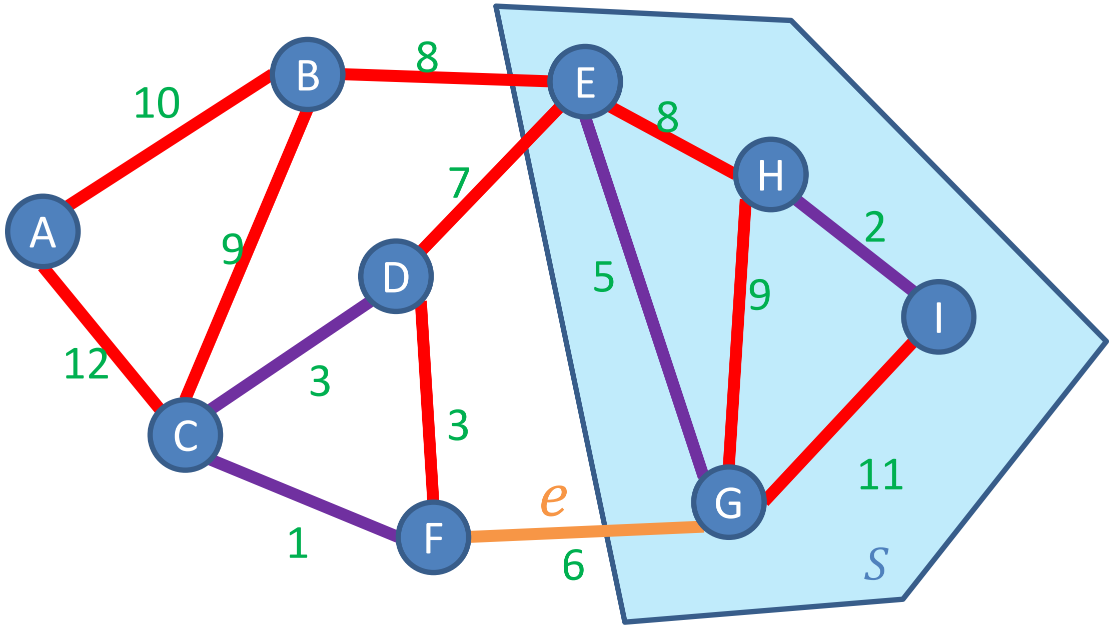
- We must show the following 2 things after building a new tree T’:

1. The sum of the weights did not go up ✓
   - We proved that $w(T') \leq w(T)$
2. T' is still a spanning tree ✓
   - T’ has V-1 edges
   - T’ connects all the nodes in the graph

Therefore T’ is also a MST on G

Proof of Kruskal’s Algorithm

Start with an empty tree A
Repeat $V - 1$ times:
Add the min-weight edge that doesn’t cause a cycle



Proof: Suppose we have some arbitrary set of edges A that Kruskal’s has already selected to include in the MST. $e = (F, G)$ is the edge Kruskal’s selects to add next

We know that there cannot exist a path from F to G using only edges in A because e does not cause a cycle

We can cut the graph therefore into 2 disjoint sets:

- nodes reachable from G using edges in A
- nodes reachable from F using edges in A

e is the minimum cost edge that crosses this cut, so by the Cut Theorem, Kruskal’s is optimal!

Kruskal's Algorithm Runtime

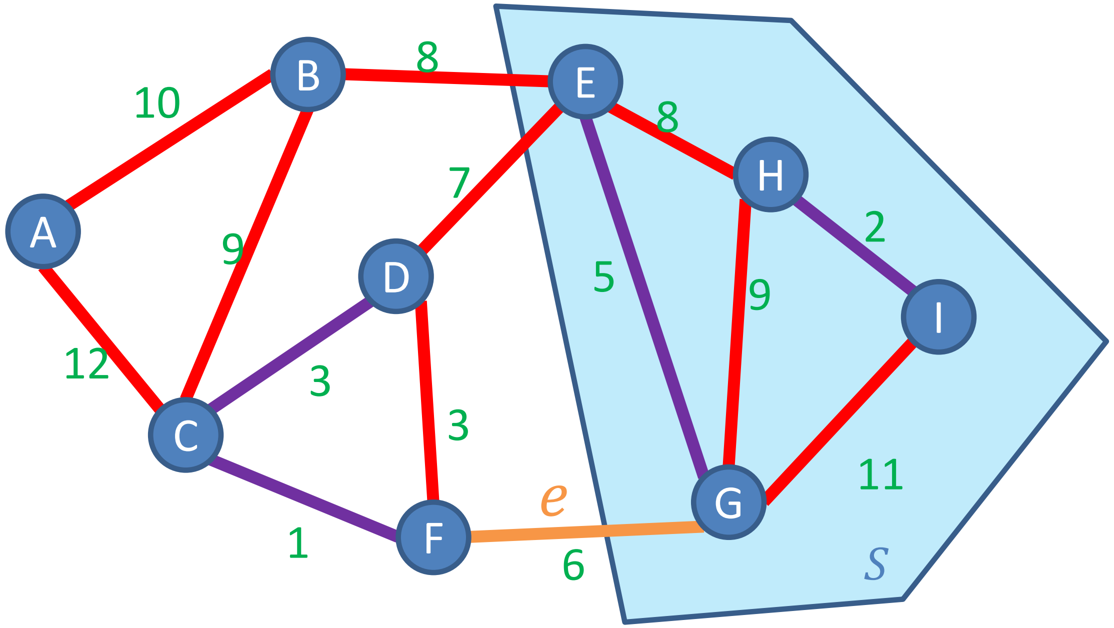
Start with an empty tree A

Repeat $V - 1$ times:

Add the min-weight edge that doesn't cause a cycle

Keep edges in a Disjoint-set data structure (very fancy)

$O(E \log V)$



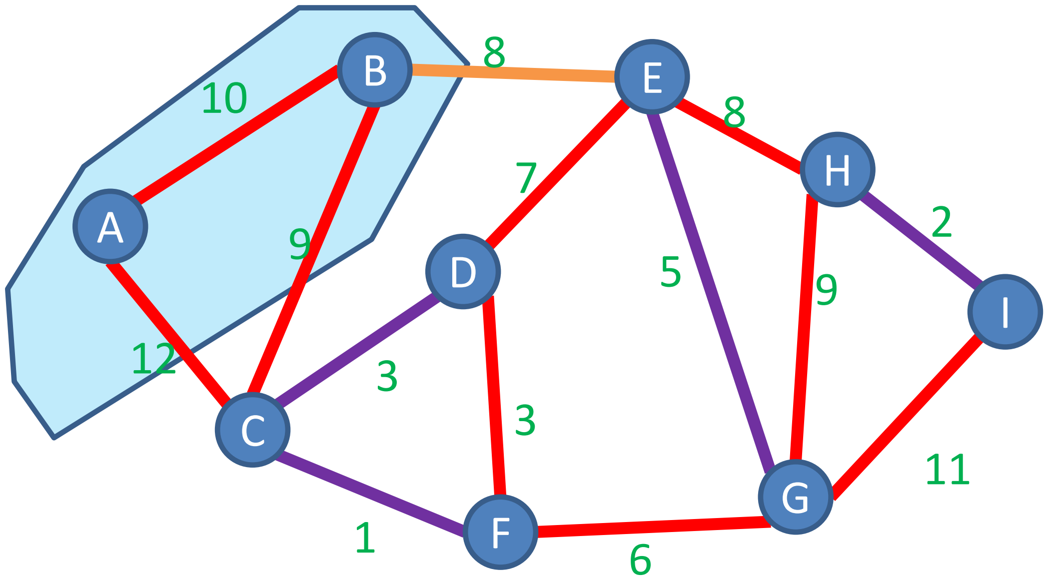
General MST Algorithm

Start with an empty tree A

Repeat $V-1$ times:

Pick a cut $(S, V-S)$ which A respects

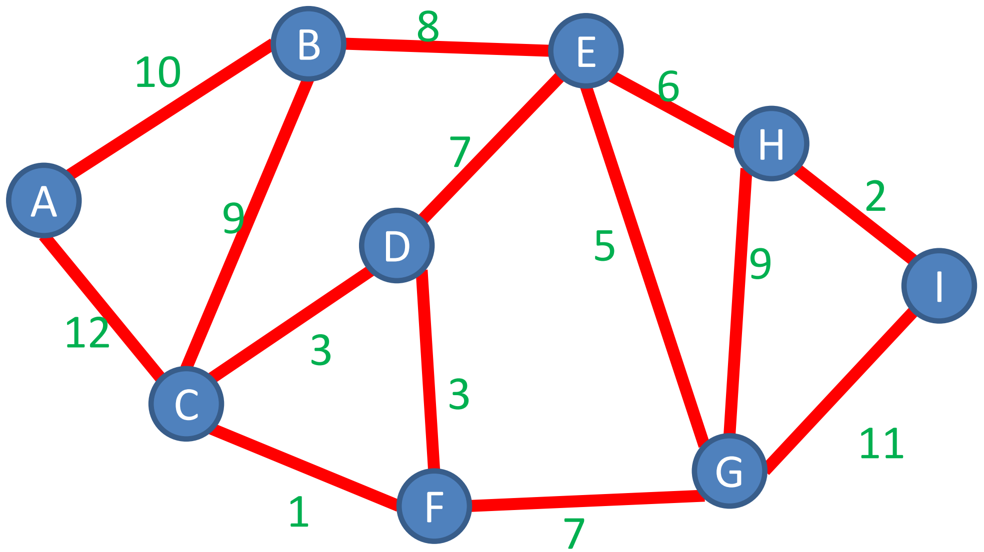
Add the min-weight edge which crosses $(S, V-S)$



Prim's Algorithm

Start with an empty tree A
Repeat $V - 1$ times:
	Pick a cut $(S, V - S)$ which A respects
	Add the min-weight edge which crosses $(S, V - S)$

S is all endpoint of edges in A
e is the min-weight edge that grows the tree



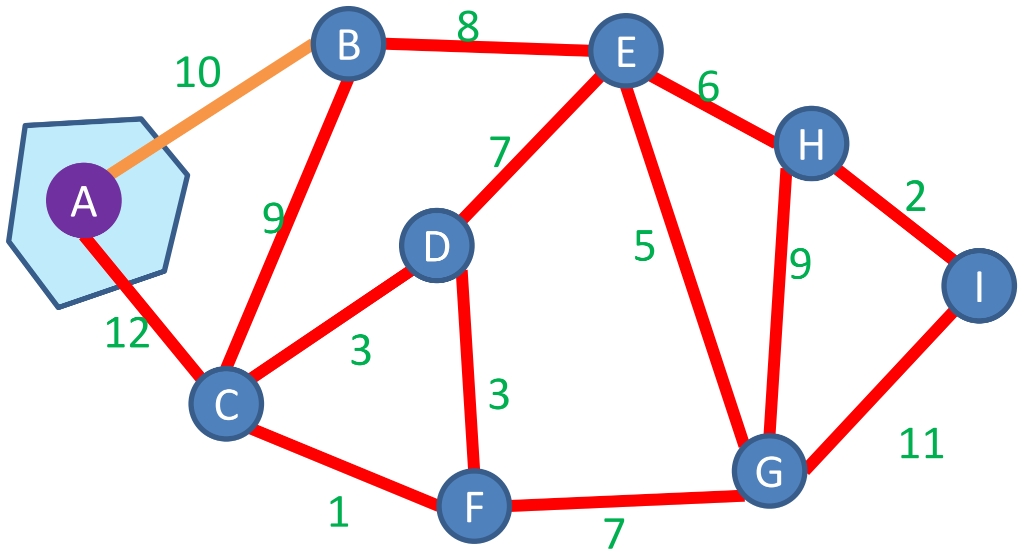
Prim's Algorithm

Start with an empty tree A

Pick a start node

Repeat $V - 1$ times:

Add the min-weight edge which connects to node in A with a node not in A



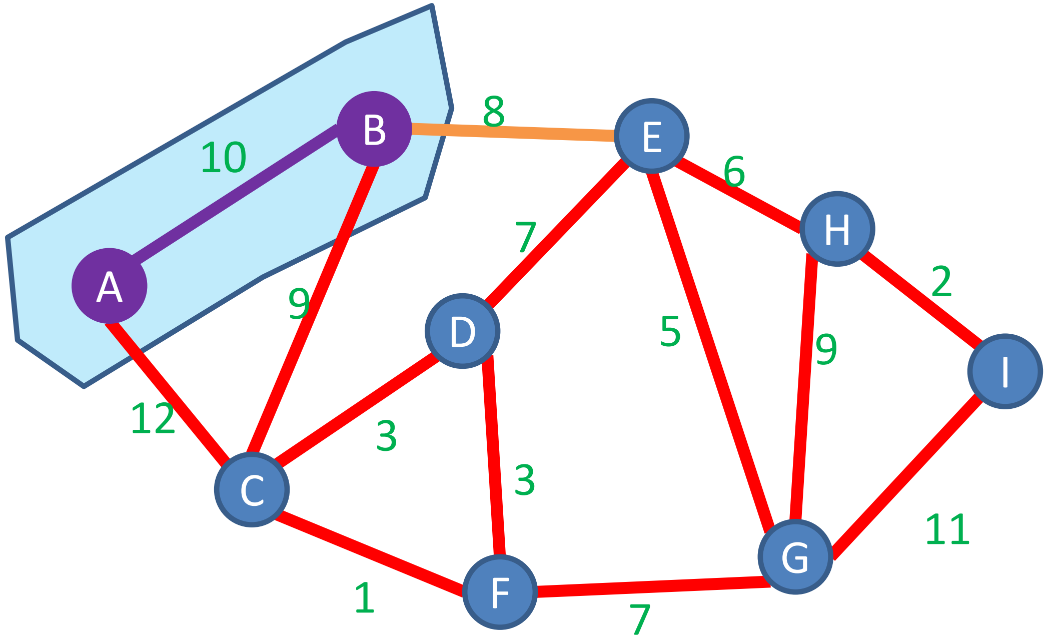
Prim's Algorithm

Start with an empty tree A

Pick a start node

Repeat $V - 1$ times:

Add the min-weight edge which connects to node in A with a node not in A



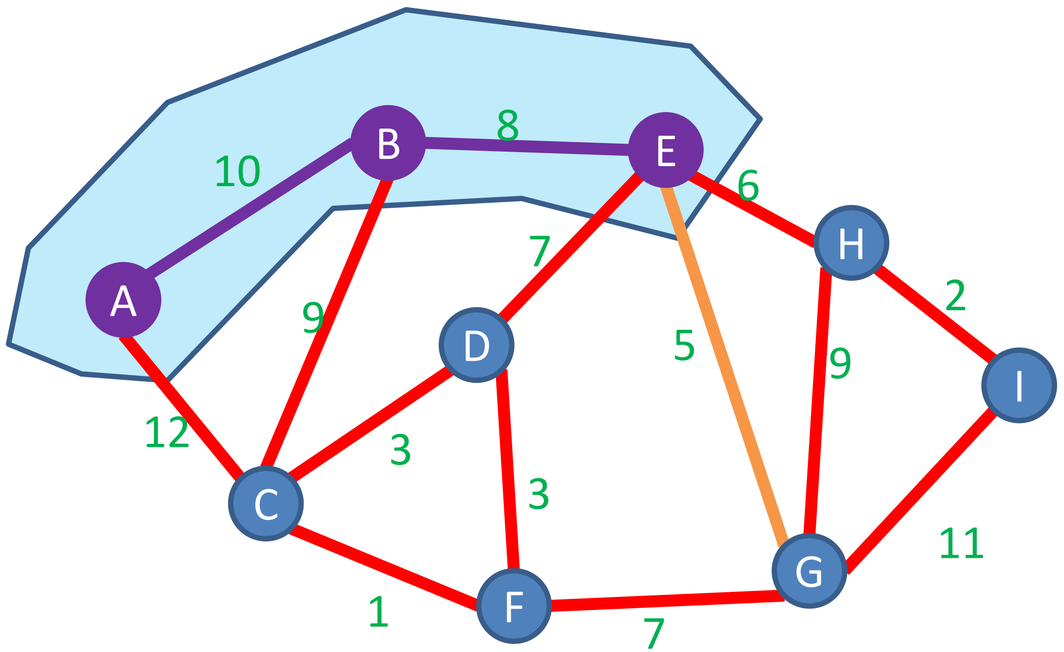
Prim’s Algorithm

Start with an empty tree A

Pick a start node

Repeat $V - 1$ times:

Add the min-weight edge which connects to node in A with a node not in A



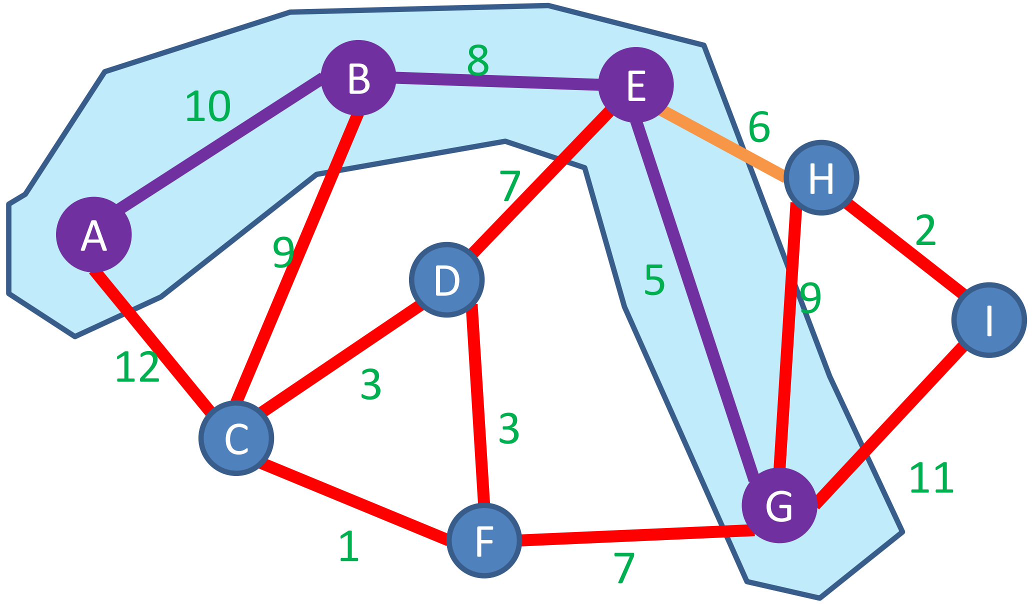
Prim's Algorithm

Start with an empty tree A

Pick a start node

Repeat $V - 1$ times:

Add the min-weight edge which connects to node in A with a node not in A



Prim's Algorithm

Start with an empty tree A

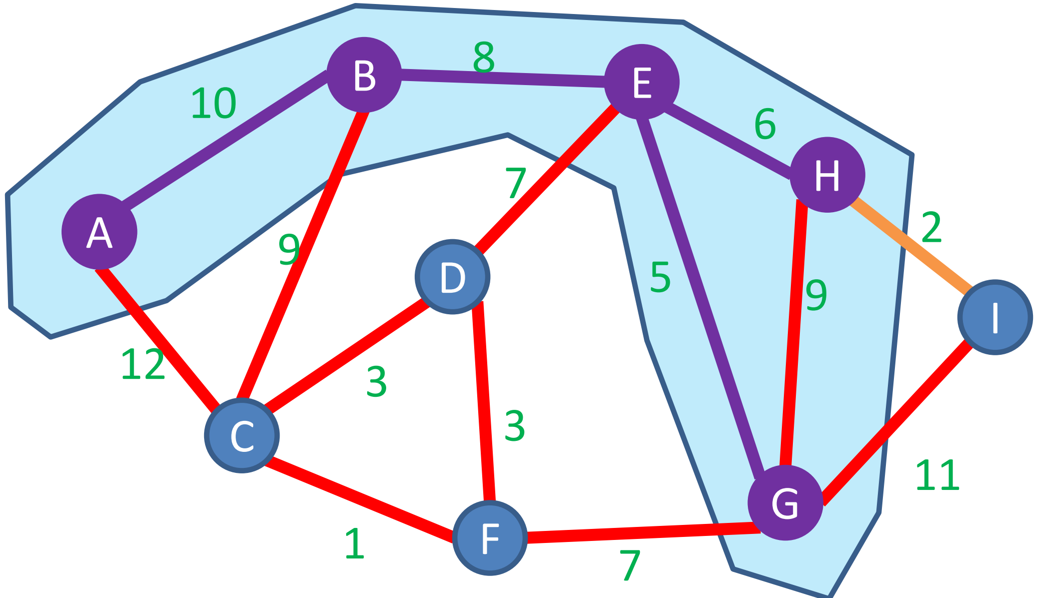
Pick a start node

Repeat $V - 1$ times:

Add the min-weight edge which connects to node in A with a node not in A

Keep edges in a Heap

$O(E \log V)$



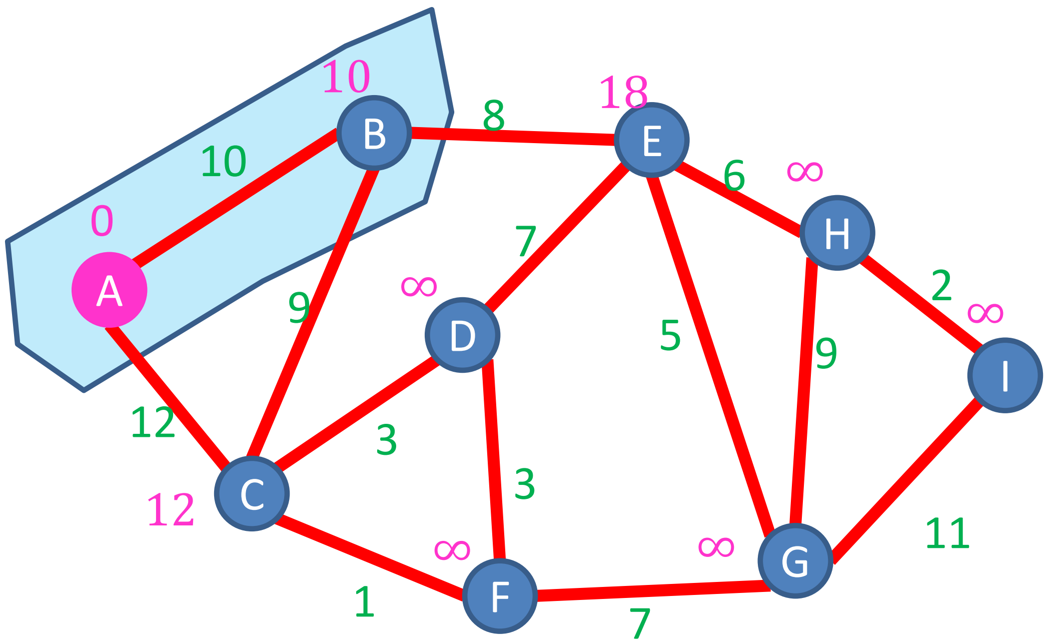
Dijkstra's Algorithm

Given some start node s

Start with an empty tree A

Repeat $V - 1$ times:

  Add the "nearest" node to s not yet in A



Dijkstra's Algorithm

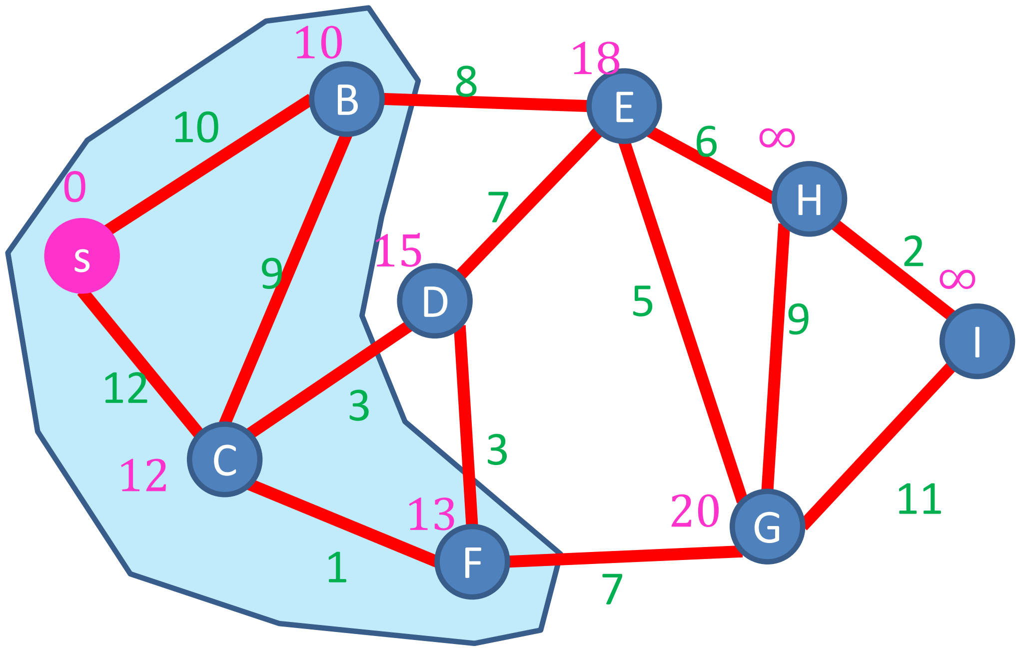
Given some start node s

Start with an empty tree A

Repeat $V - 1$ times:

Add the "nearest" node to s not yet in A

VERY similar to Prim's!



Dijkstra's Algorithm

Initialize $d_v = \infty$ for each node v

Keep a priority queue PQ of nodes, using d_v as key

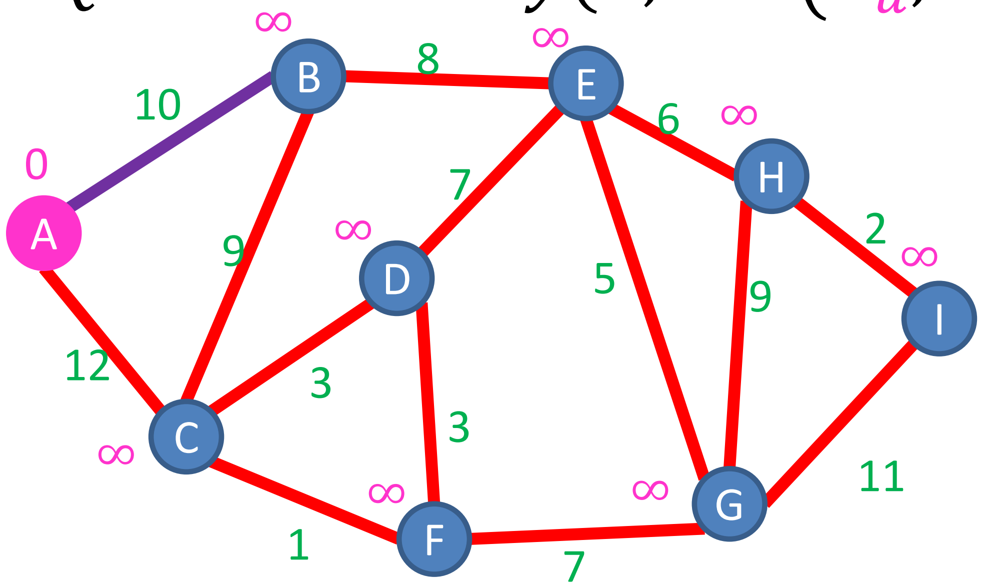
Pick a start node s, set $d_s = 0$

While PQ is not empty:

  $v = PQ.extractmin()$

  for each $u \in V$ s.t. $(v, u) \in E$:

    $PQ.decreaseKey(u, \min(d_u, d_v + w(v, u)))$



Prim's Algorithm

Initialize $d_v = \infty$ for each node v

Keep a priority queue PQ of nodes, using d_v as key

Pick a start node s, set $d_s = 0$

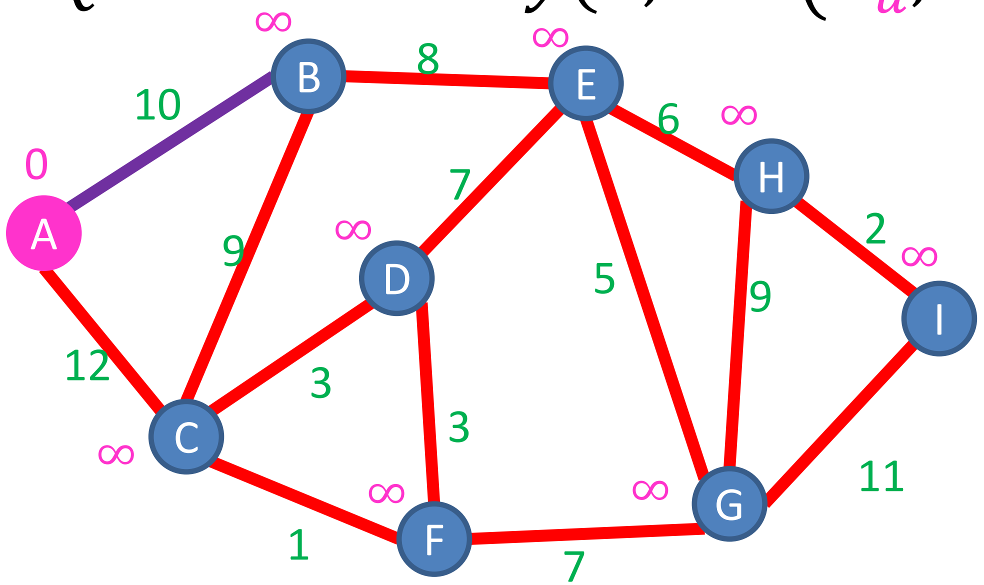
While PQ is not empty:

    $v = PQ.extractmin()$

    for each $u \in V$ s.t. $(v, u) \in E$:

        $PQ.decreaseKey(u, \min(d_u, w(v, u)))$

Can use min heap dataset to make each decreaseKey() operation faster!



Summary of MST results

- Fredman-Tarjan ‘84: $\Theta(E + V \log V)$
- Gabow et al ‘86: $\Theta(E \log \log^* V)$
- Chazelle ‘00: $\Theta(E\alpha(V))$
- Pettie-Ramachandran ’02: $\Theta(?)$ (optimal)
- Karger-Klein-Tarjan ‘95: $\Theta(E)$ (randomized)