CS 4102: Algorithms

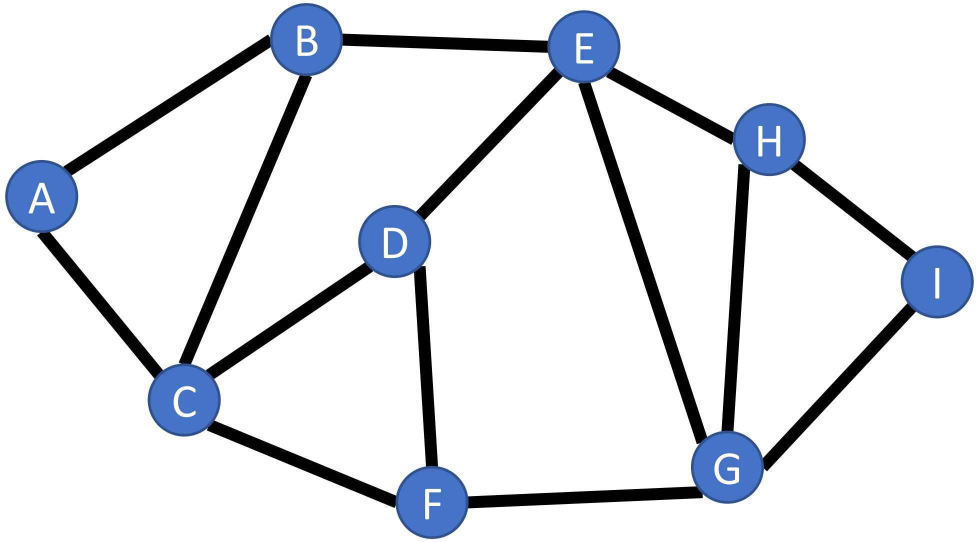
Lecture 19: Graph Algorithms (MST)

David Wu

Fall 2019

Warm-Up

Show that for any graph $G = (V, E)$,
$\sum_{v \in V} \deg(v)$ is even



Recall: degree of a node is number of edges incident upon that node

$\deg(A) = 2$ and $\deg(E) = 4$

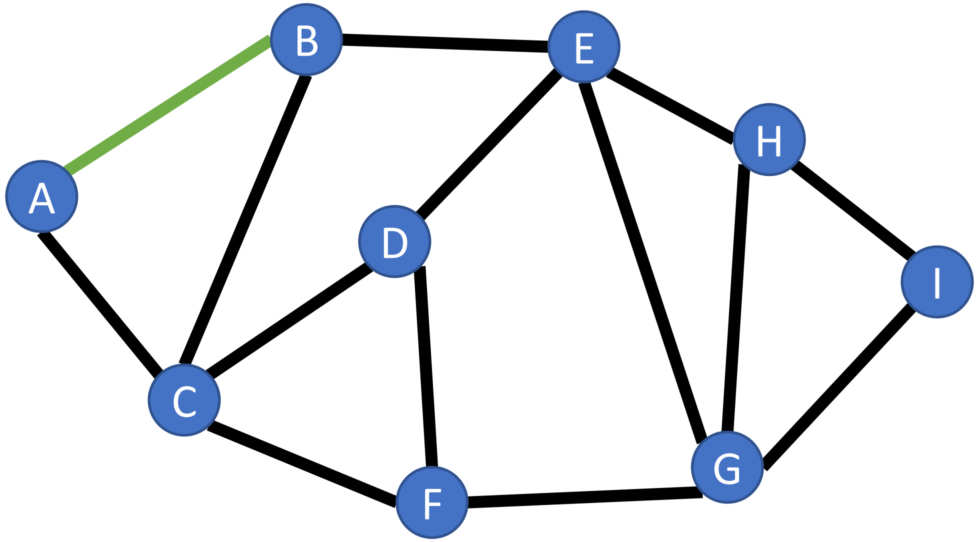
Warm-Up

Consider any edge $e \in E$

This edge is incident on 2 vertices (on each end)

This means $\sum_{v \in V} \deg(v) = 2 \cdot |E|$

Therefore $\sum_{v \in V} \deg(v)$ is even



Today’s Keywords

Greedy Algorithms

Choice Function

Graphs

Minimum Spanning Tree

Kruskal’s Algorithm

Prim’s Algorithm

Cut and Cycle Properties

CLRS Readings: Chapter 22, 23

Homework

tomorrow (Wednesday), 11pm

HW6 due ~~**today (Tuesday, November 5)**~~**, 11pm**

- Dynamic programming and greedy algorithms
- Written (use LaTeX!) – Submit both **zip** and **pdf** (two separate attachments)!

HW10A also due **today, 11pm**

- No late submissions allowed

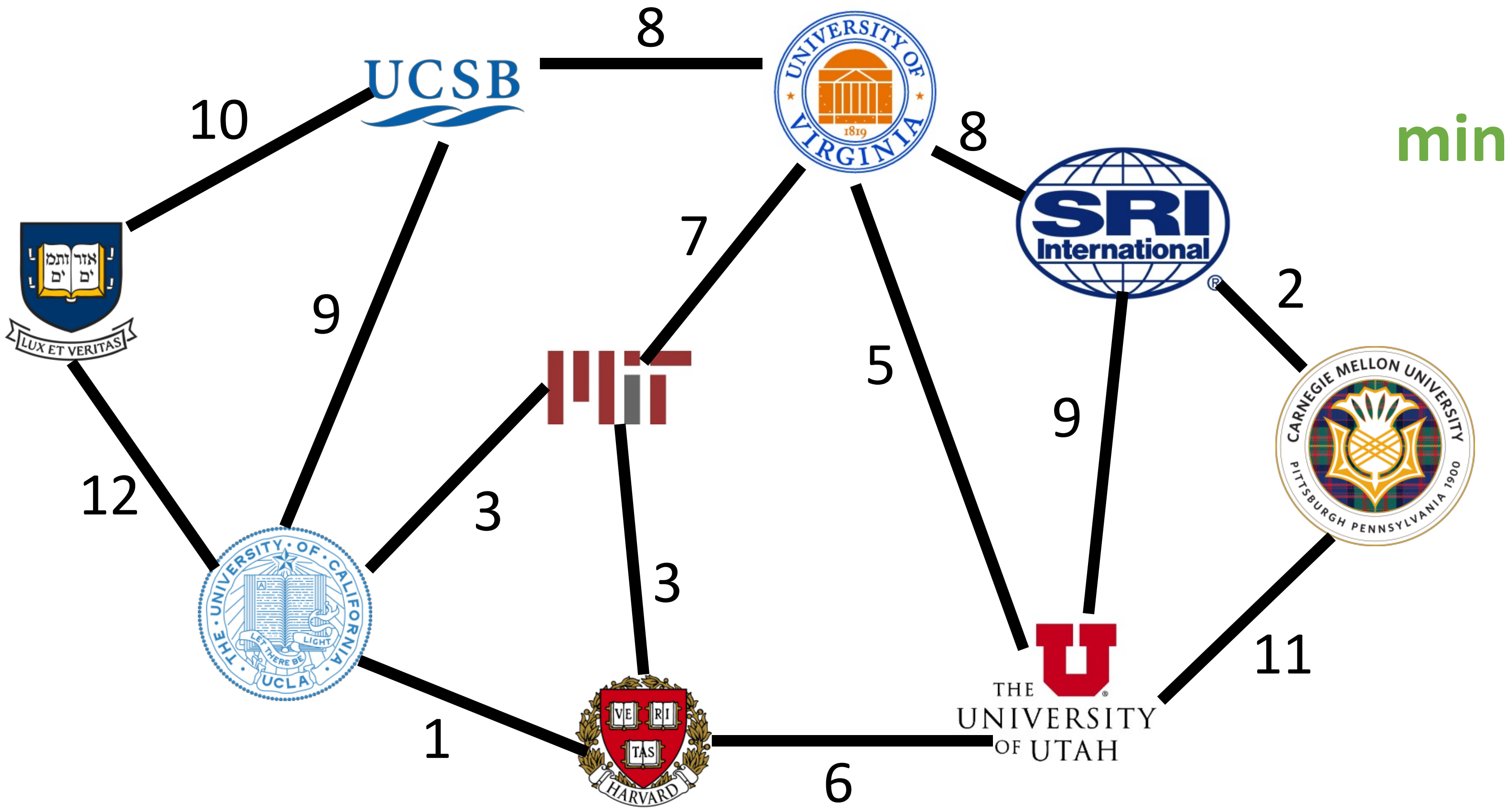
HW7 out today, due **Thursday, November 14, 11pm**

- Graph algorithms
- Written (use LaTeX!) – Submit both **zip** and **pdf** (two separate attachments)!

HW10B also out today, due **Thursday, November 14, 11pm**

- No late submissions allowed

The ARPANET Problem



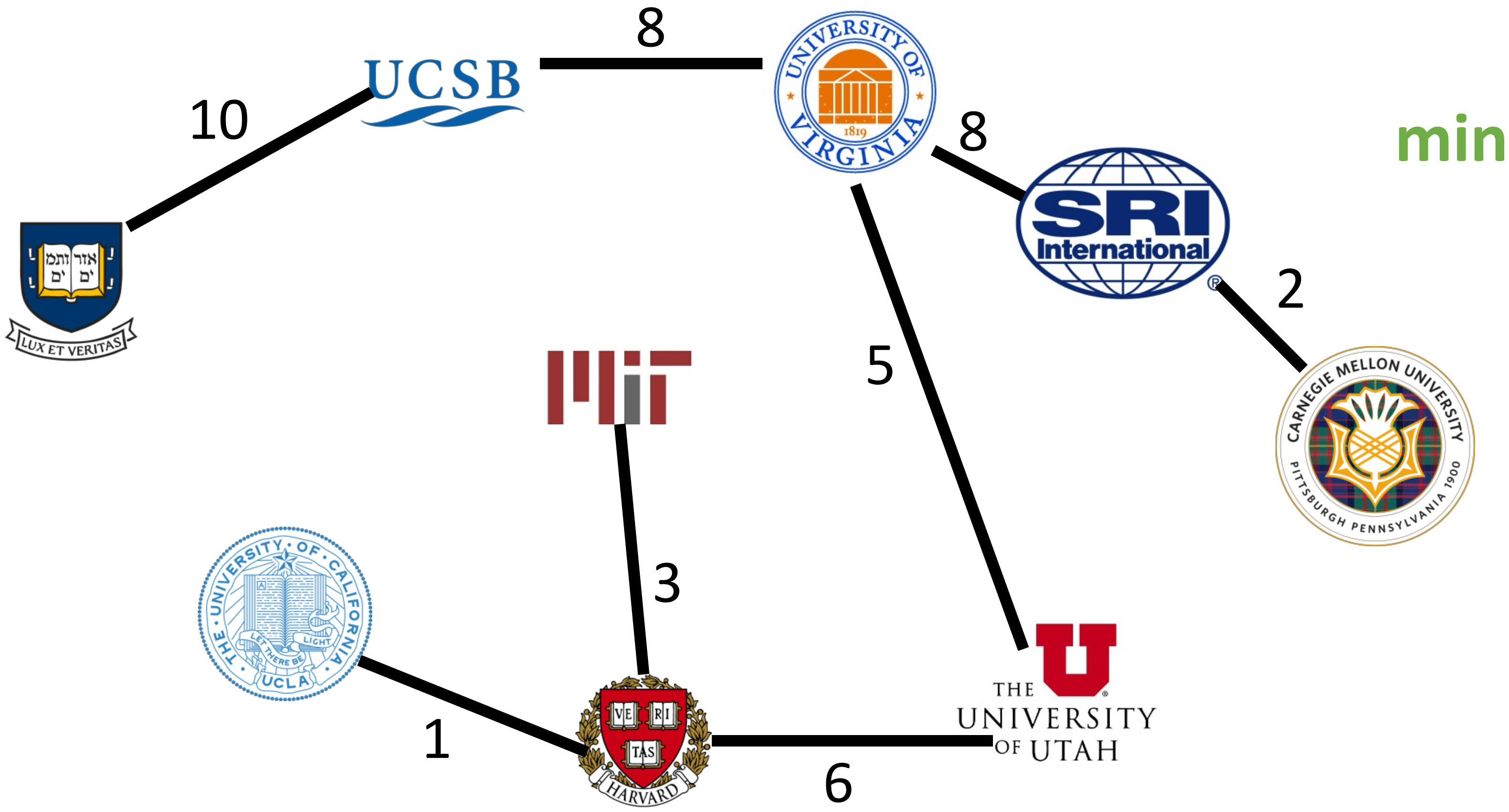
Find a **minimum spanning tree (MST)**

Problem: need to connect all of these places into a network

We have a list of possible wires to use, along with the cost of each wire

Goal: Find the cheapest set of wires to run to connect all places

The ARPANET Problem



Find a **minimum spanning tree (MST)**

Problem: need to connect all of these places into a network

We have a list of possible wires to use, along with the cost of each wire

Goal: Find the cheapest set of wires to run to connect all places

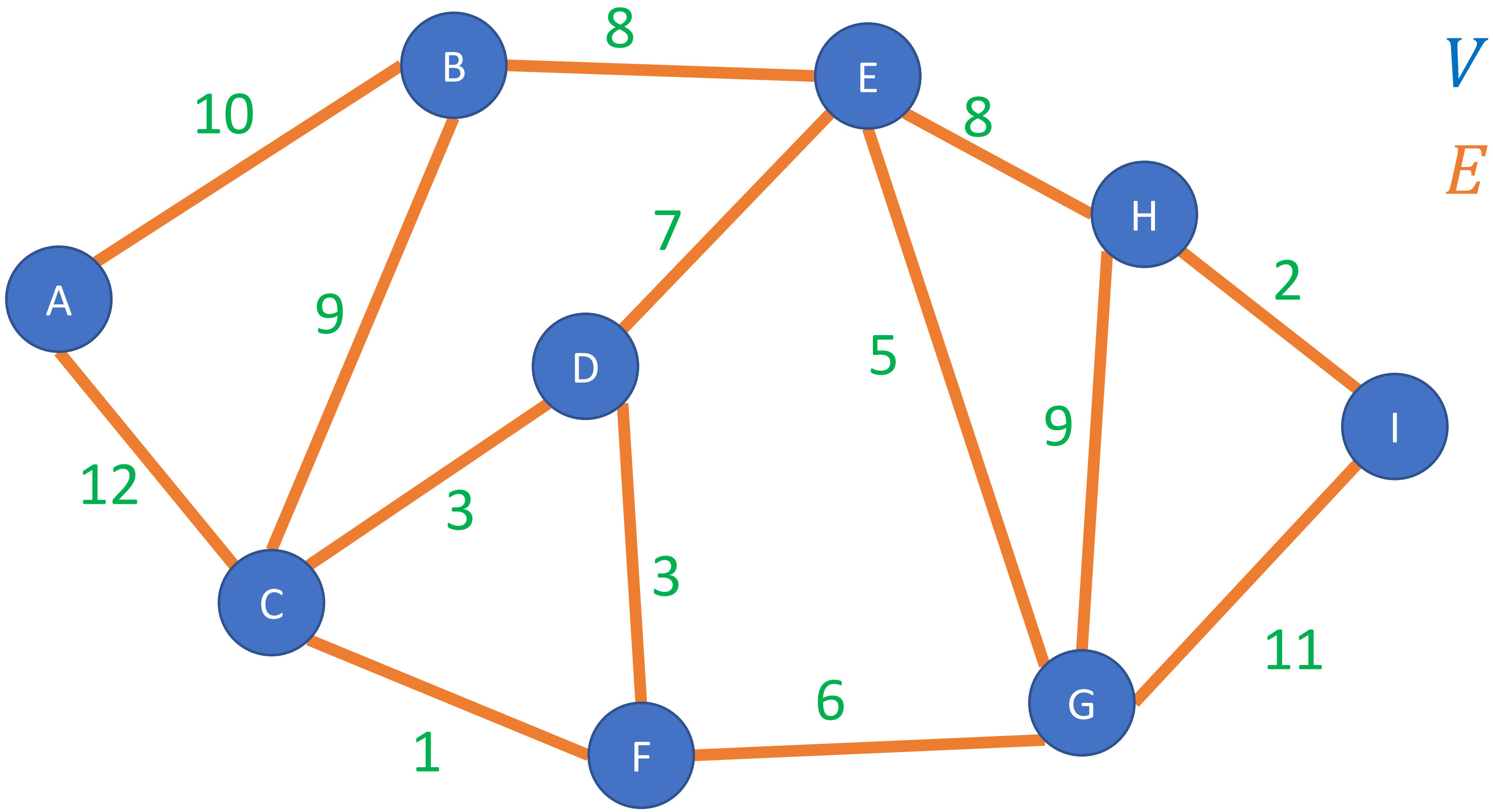
Graphs

Definition: $G = (V, E)$

V: Vertices/Nodes

E: Edges

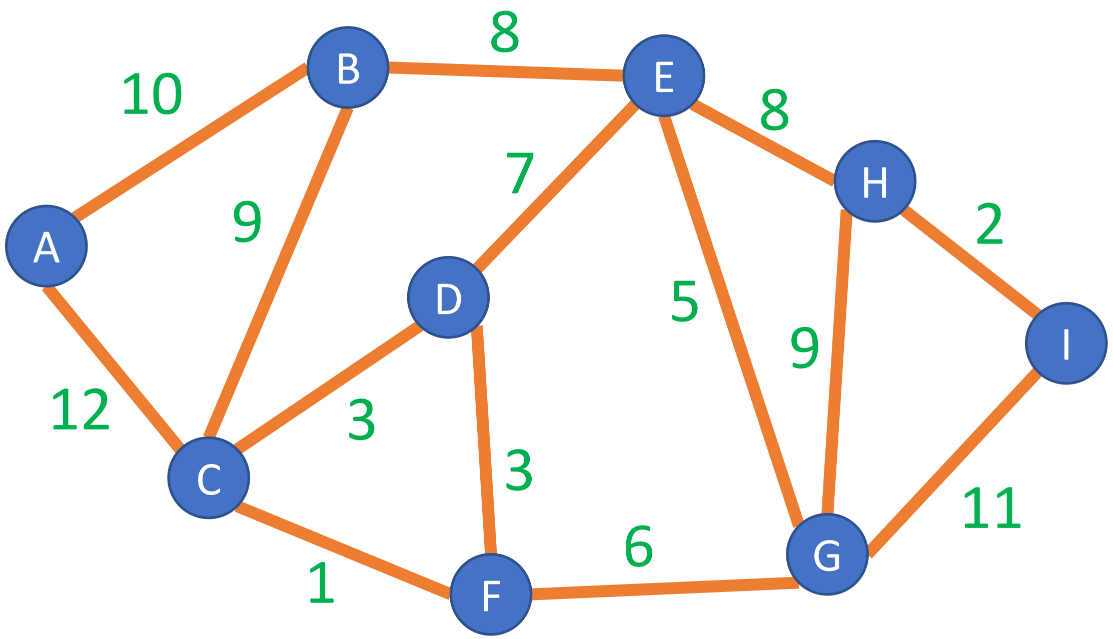
$w(e)$ = weight of edge e



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

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

Adjacency List Representation



A	B	C		
B	A	C	E	
C	A	B	D	F
D	C	E	F	
E	B	D	G	H
F	C	D	G	
G	E	F	H	I
H	E	G	I	
I	G	H		

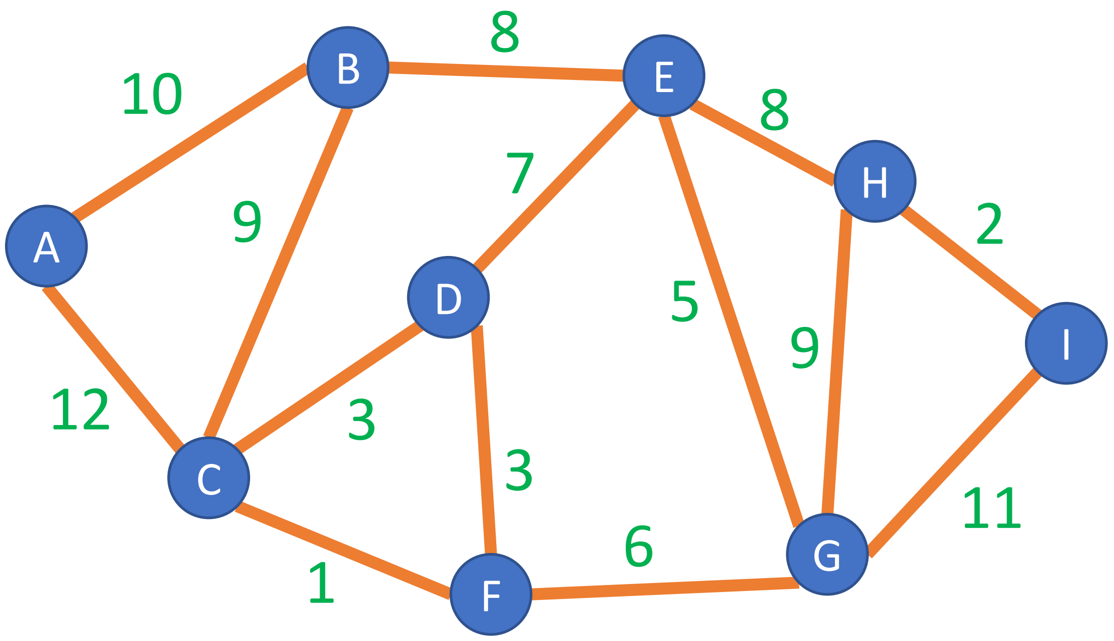
Tradeoffs

Space: $|V| + |E|$

Time to list neighbors: $\deg(A)$

Time to check edge (A, B): $\deg(A)$

Adjacency Matrix Representation



	A	B	C	D	E	F	G	H	I
A		10	12						
B	10		9		8				
C	12	9		3		1			
D			3		7	3			
E		8		7			5	8	
F			1	3			6		
G					5	6		9	11
H					8		9		8
I							11	8	

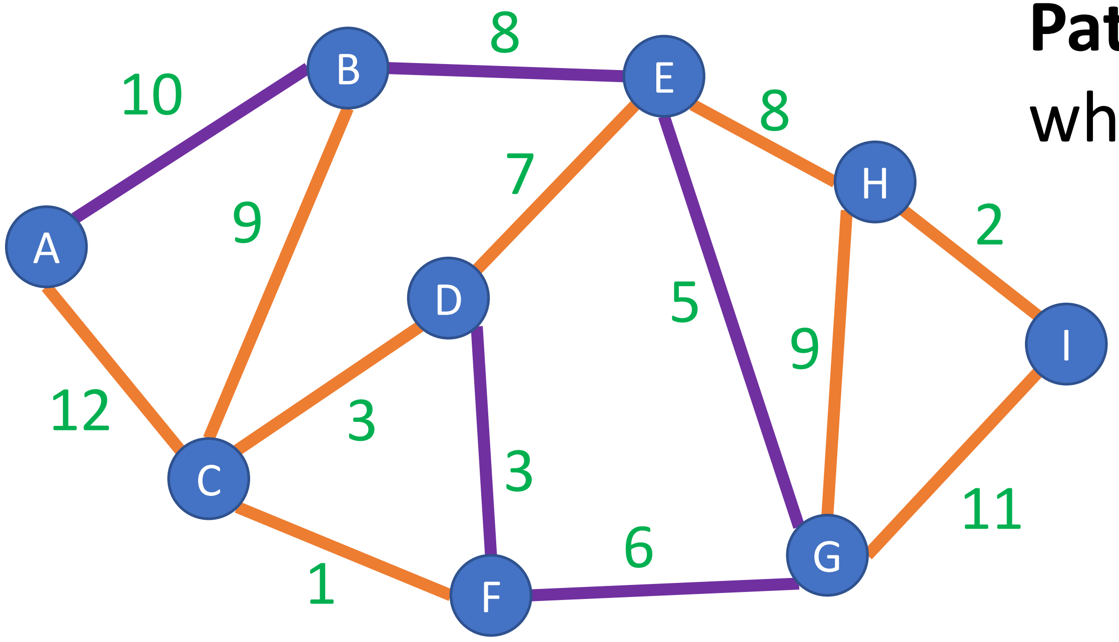
Tradeoffs

Space: $|V|^2$

Time to list neighbors: $|V|$

Time to check edge (A, B): $O(1)$

Paths in Graphs

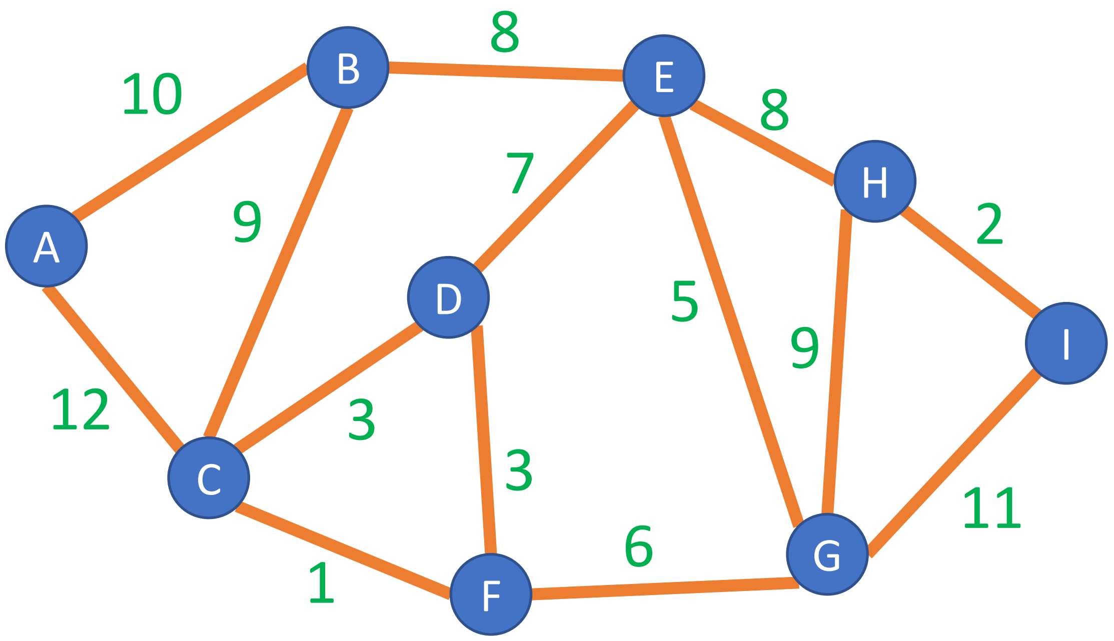


Path: A sequence of nodes $(v_1, v_2, \dots, v_k)$ where $\forall 1 \le i \le k-1$, $(v_i, v_{i+1}) \in E$

Simple Path: A path in which each node appears at most once

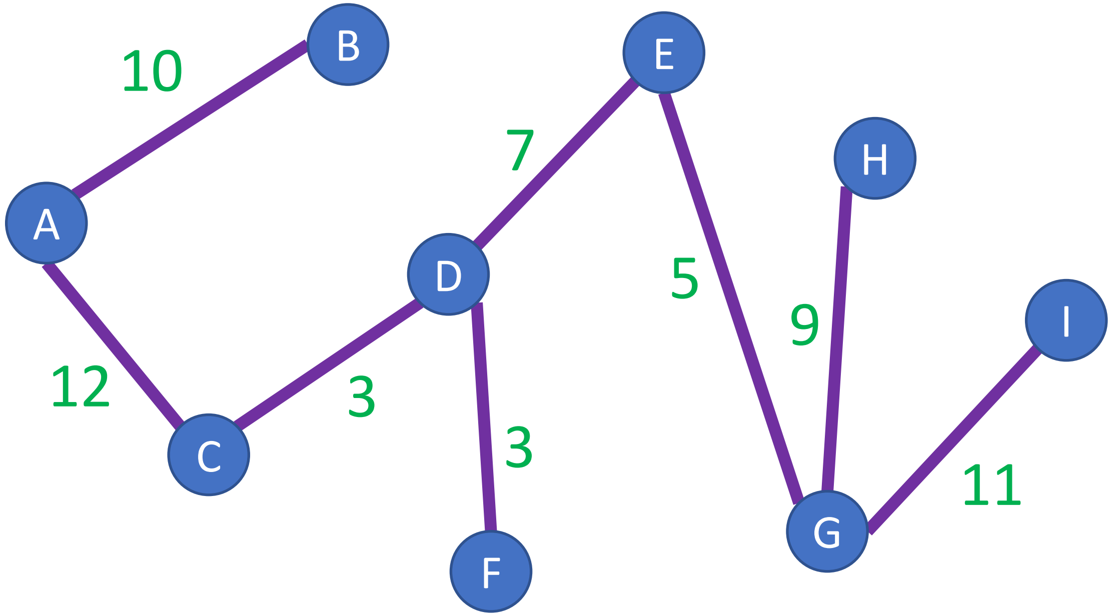
Cycle: A path of length > 2 where $v_1 = v_k$

Connected Graphs



A graph $G = (V, E)$ is **connected** if there is a path from v_1 to v_2 for every pair of distinct nodes $v_1 \neq v_2 \in V$

Trees



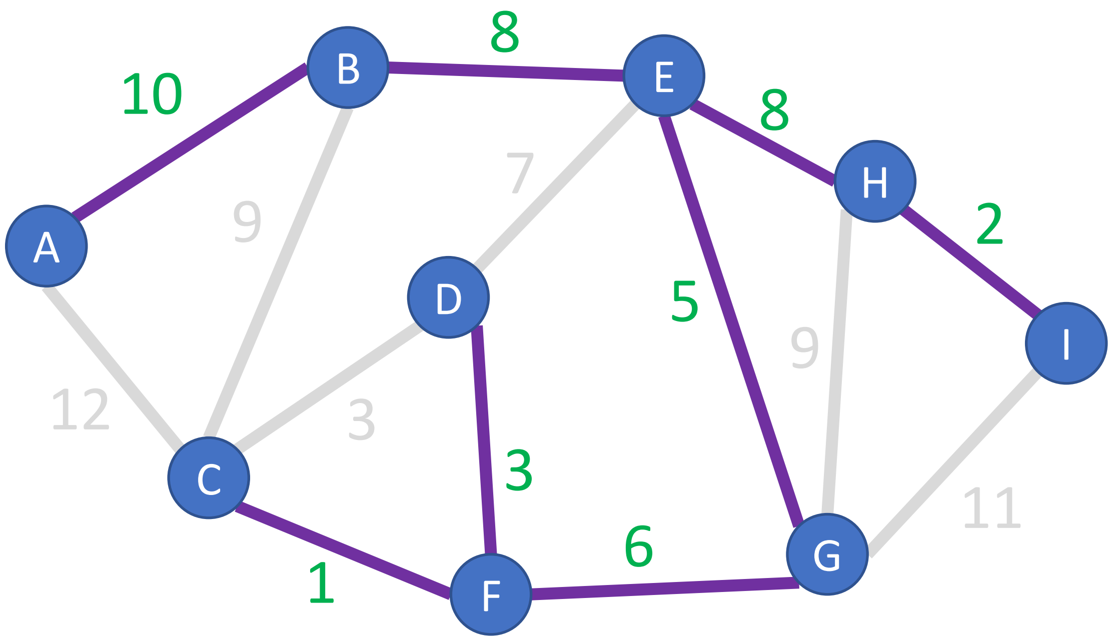
How many edges does T have?

$|V| - 1$

Proof by induction: removing an edge from a tree produces two smaller trees

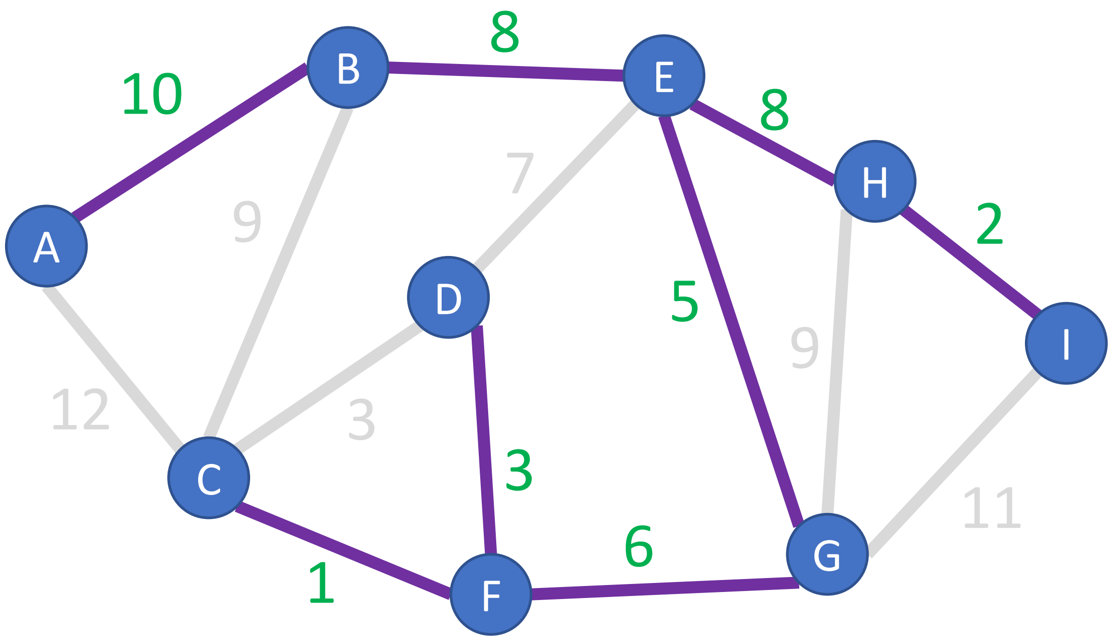
Tree: A connected graph T with no cycles (i.e., there is a unique path from every node to every other node)

Spanning Tree



A tree $T = (V_T, E_T)$ is a **spanning tree** for an undirected graph $G = (V, E)$ if $V_T = V$, $E_T \subseteq E$ (namely, T connects or "spans" all the nodes in G)

Minimum Spanning Tree



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

A tree $T = (V_T, E_T)$ is a **minimum spanning tree** for an undirected graph $G = (V, E)$ if T is a spanning tree of minimal cost

Greedy Algorithms

Requires optimal substructure

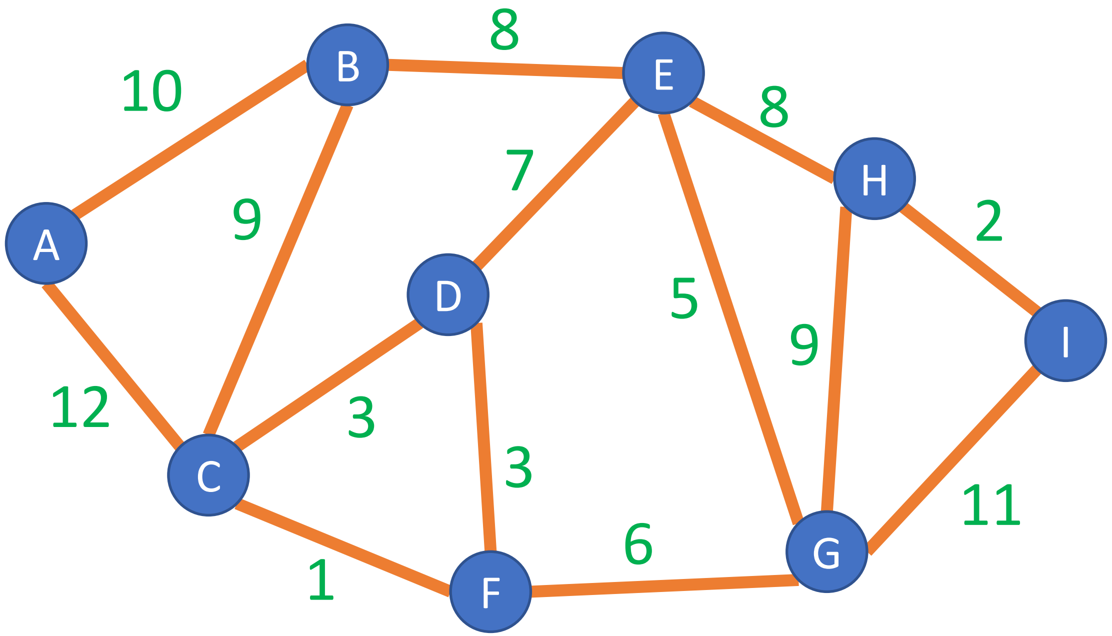
- Solution to larger problem contains the solution to a smaller one
- Only a single subproblem to consider

General Blueprint:

1. Identify a greedy choice property
    - Show that this choice is guaranteed to be included in some optimal solution
2. Repeatedly apply the choice property until no subproblems remain

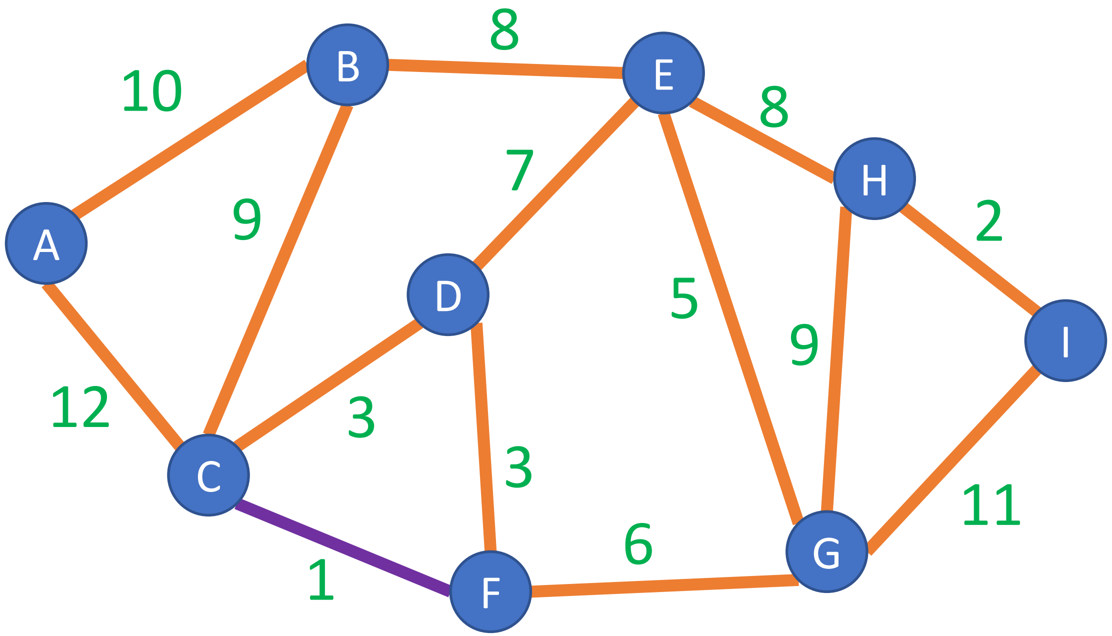
Kruskal's Algorithm

1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle



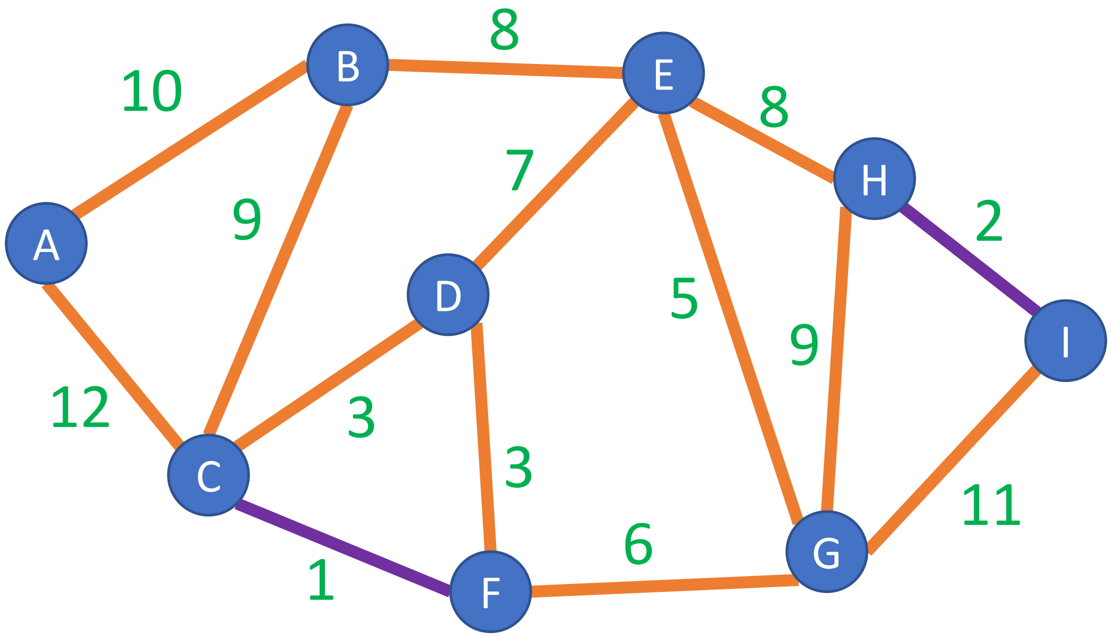
Kruskal’s Algorithm

1. Start with an empty tree T
2. Repeatedly add to T the <u>lowest-weight</u> edge that does not create a cycle



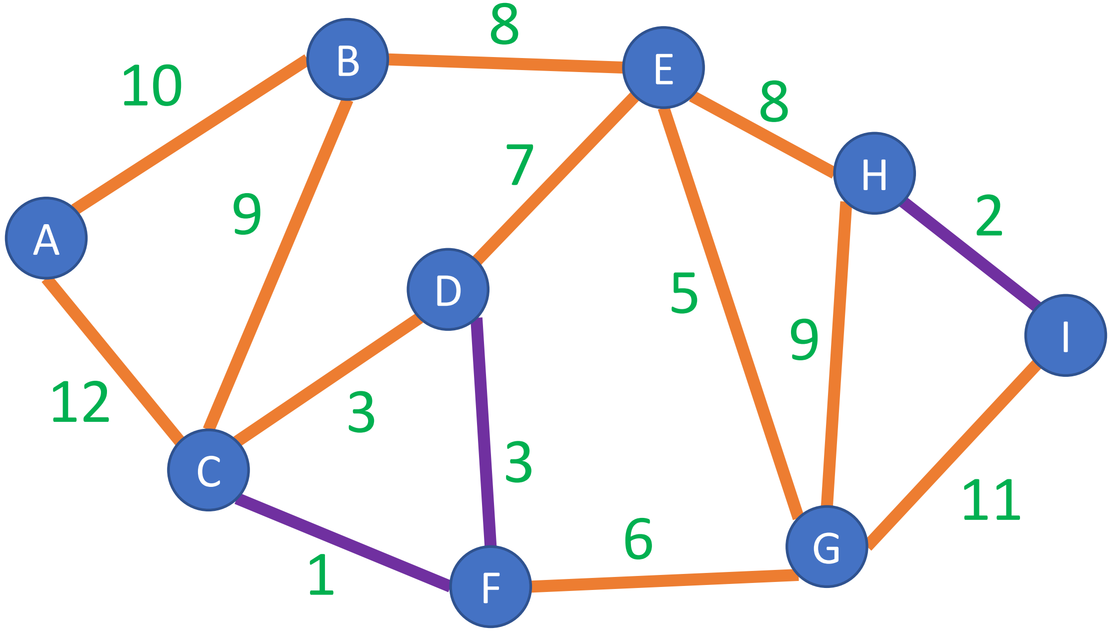
Kruskal’s Algorithm

1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle



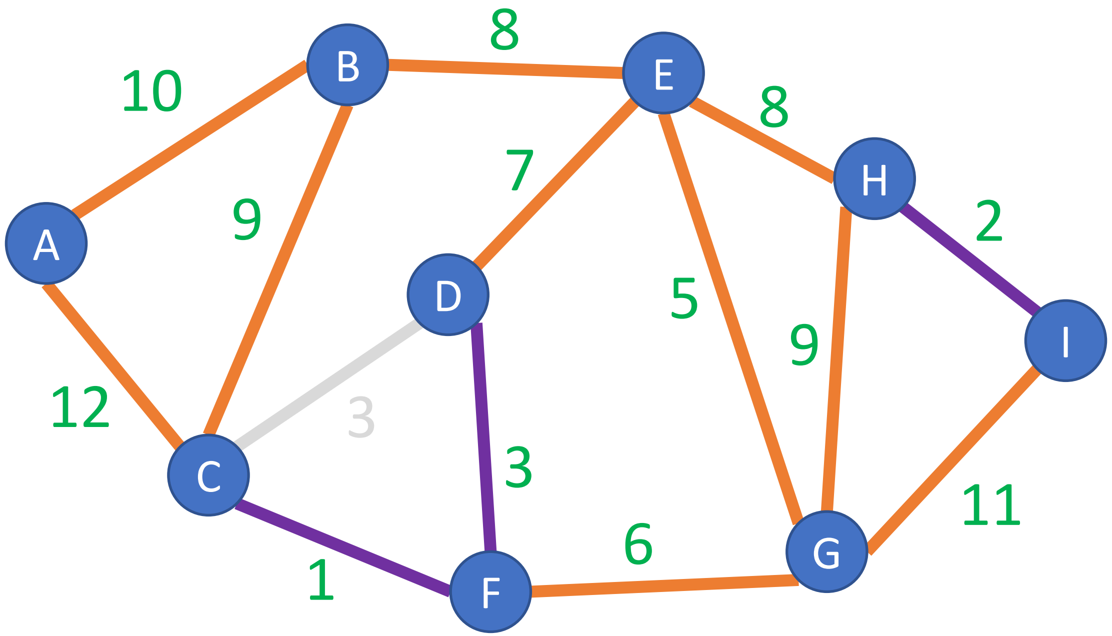
Kruskal’s Algorithm

1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle



Kruskal's Algorithm

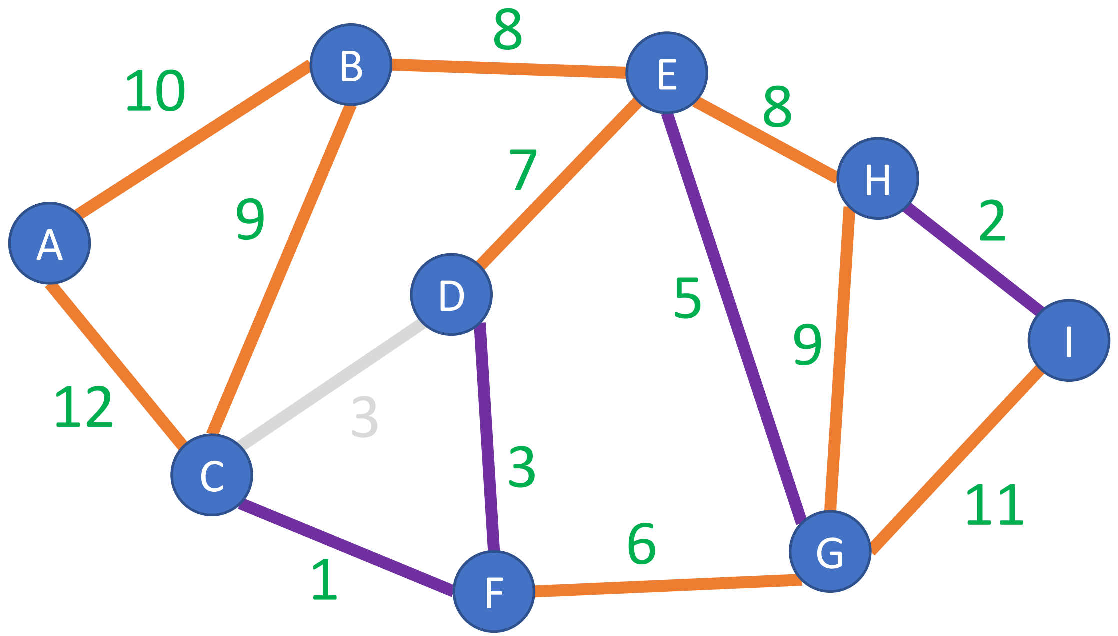
1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle



Edge forms a cycle, so discard

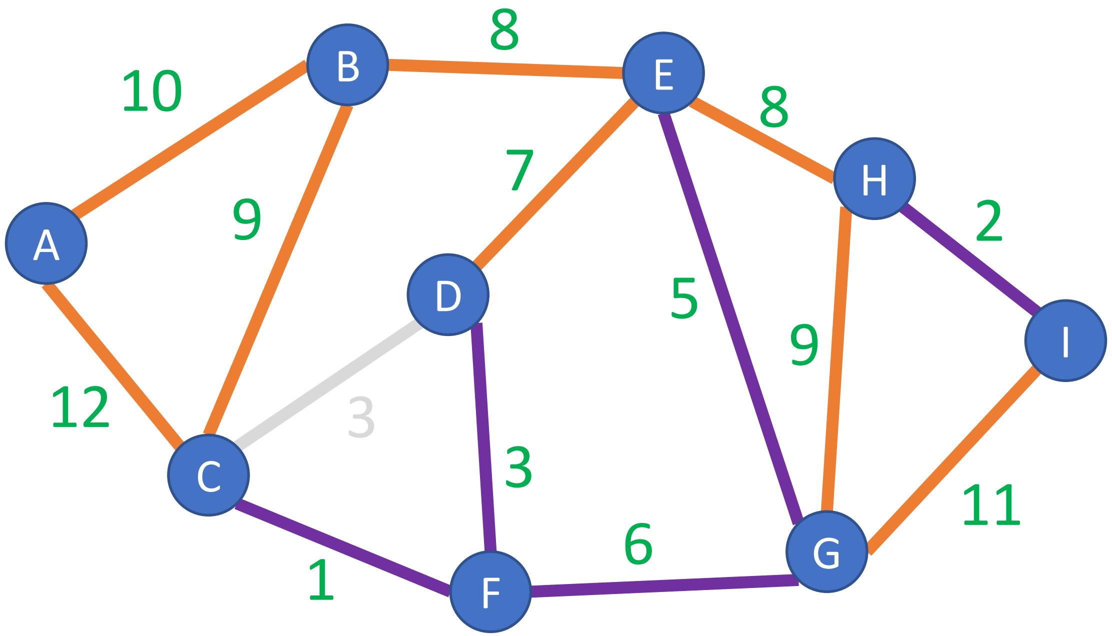
Kruskal’s Algorithm

1. Start with an empty tree T
2. Add to T the lowest-weight edge that does not create a cycle



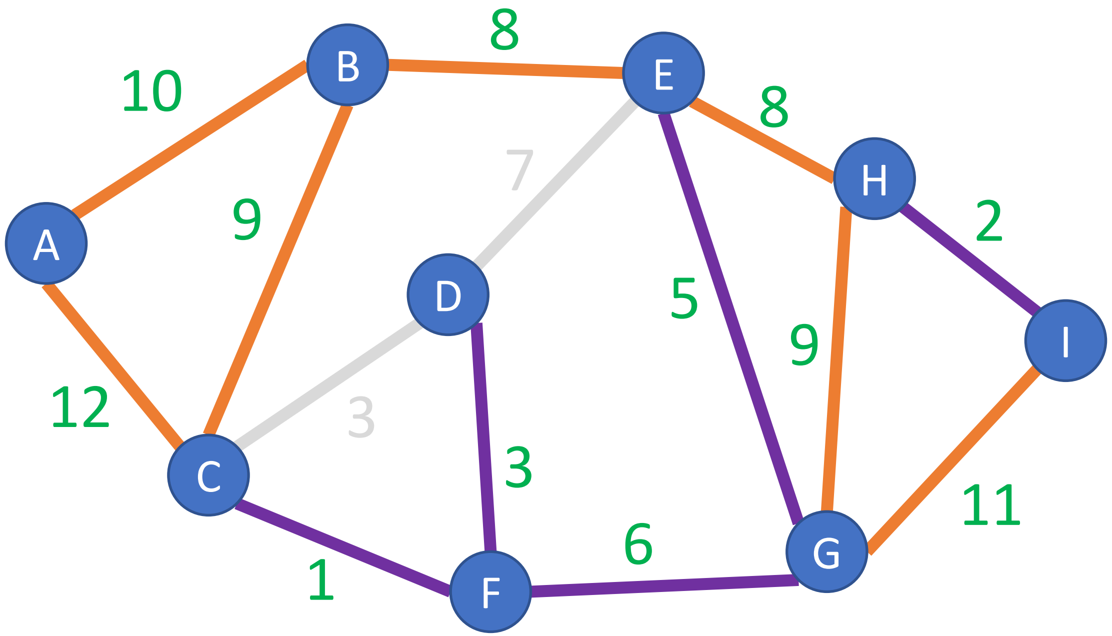
Kruskal’s Algorithm

1. Start with an empty tree T
2. Add to T the lowest-weight edge that does not create a cycle



Kruskal’s Algorithm

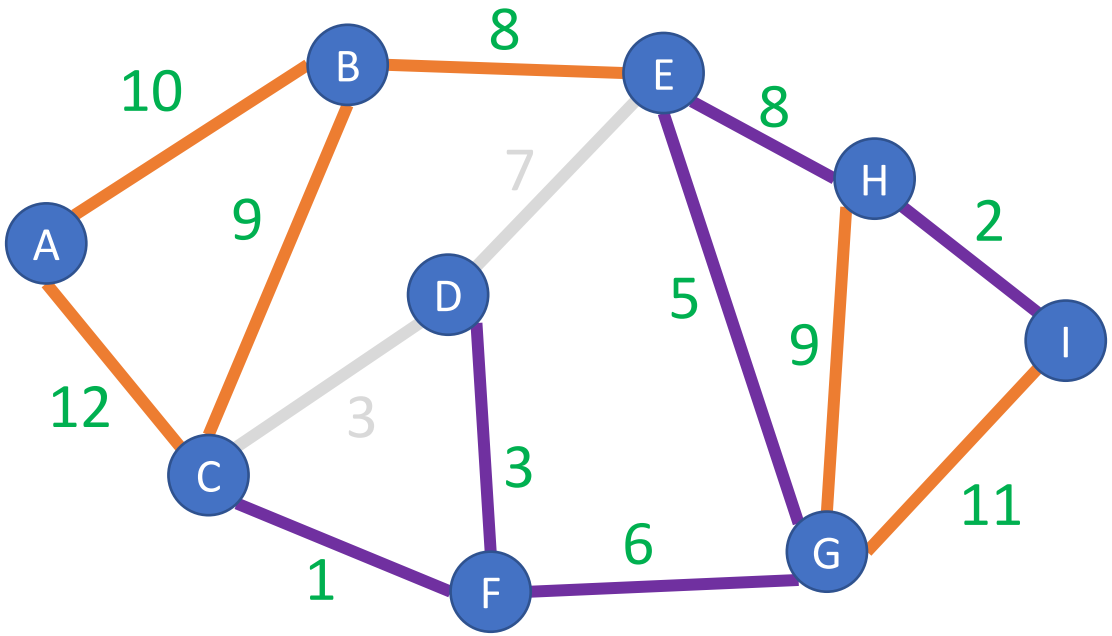
1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle



Edge forms a cycle, so discard

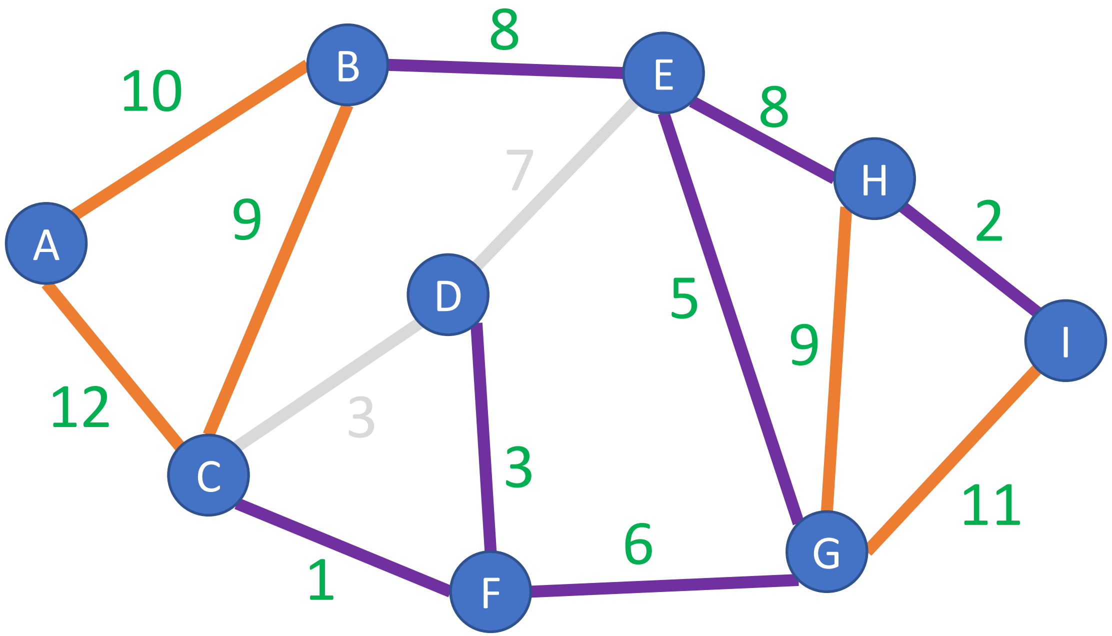
Kruskal's Algorithm

1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle



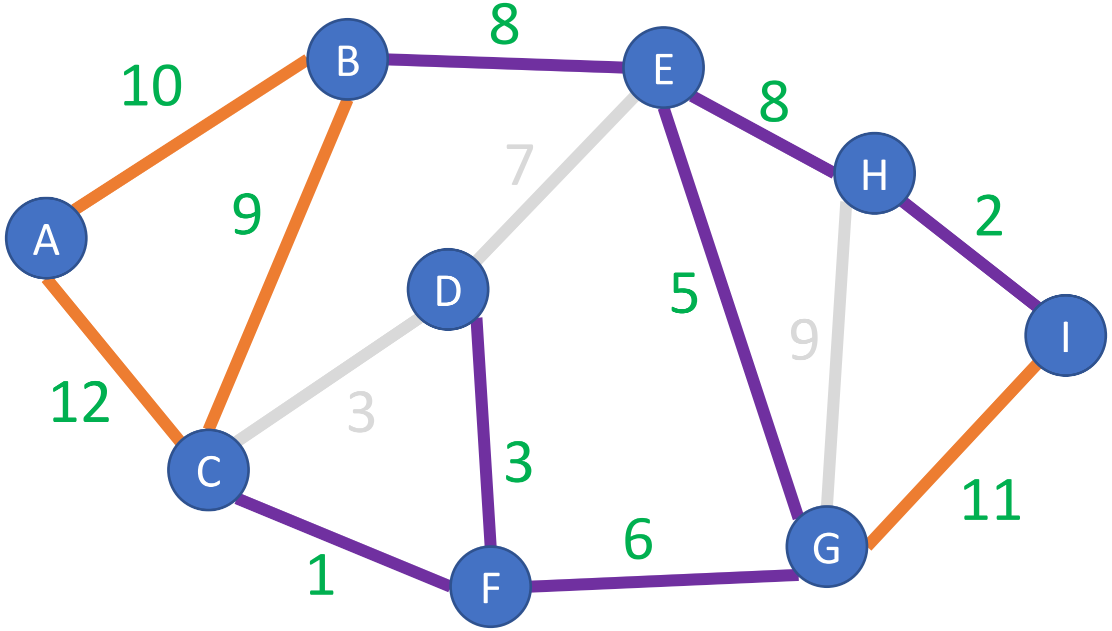
Kruskal's Algorithm

1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle



Kruskal’s Algorithm

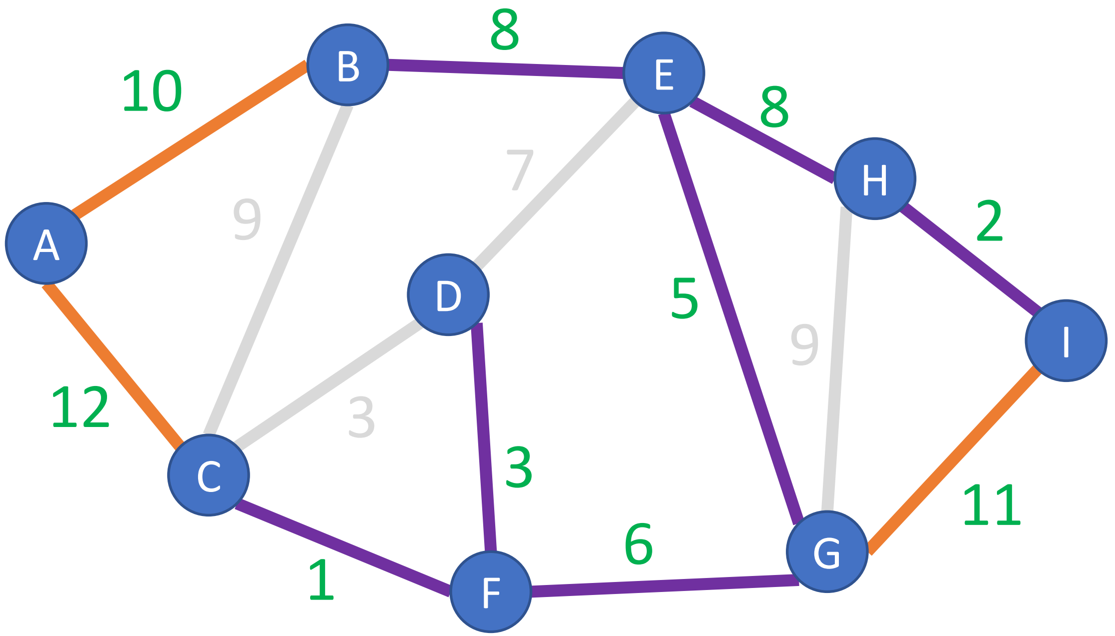
1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle



Edge forms a cycle, so discard

Kruskal's Algorithm

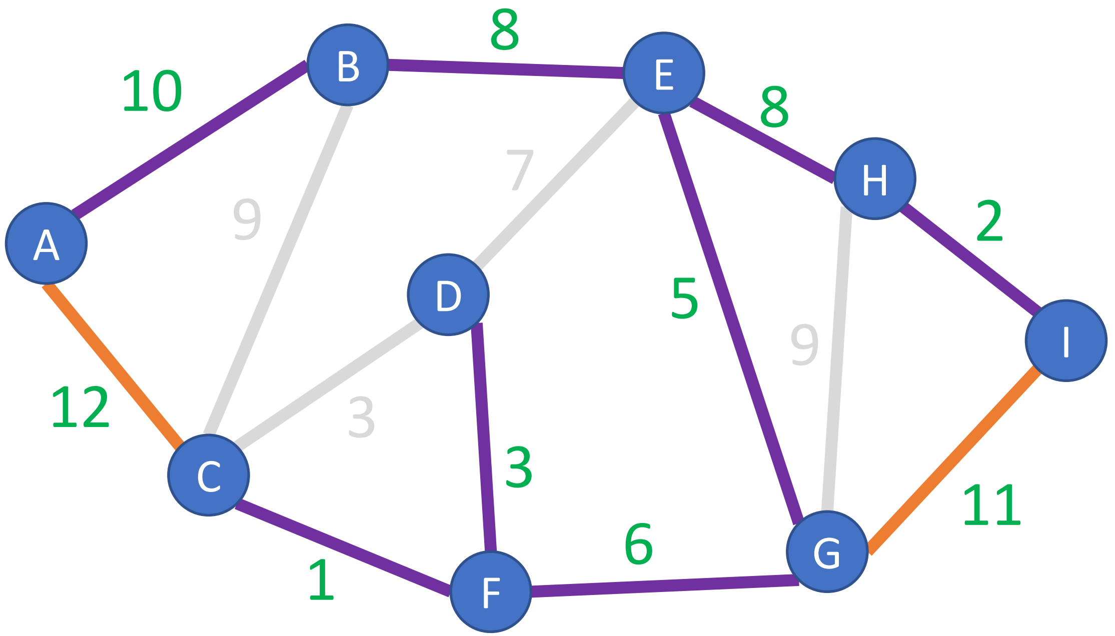
1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle



Edge forms a cycle, so discard

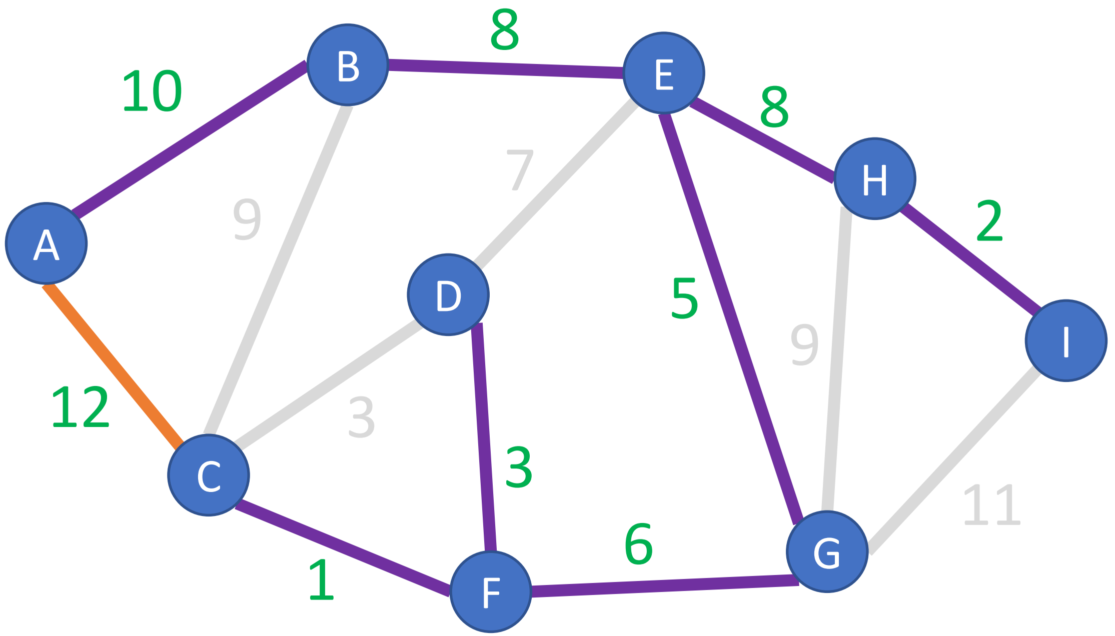
Kruskal’s Algorithm

1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle



Kruskal's Algorithm

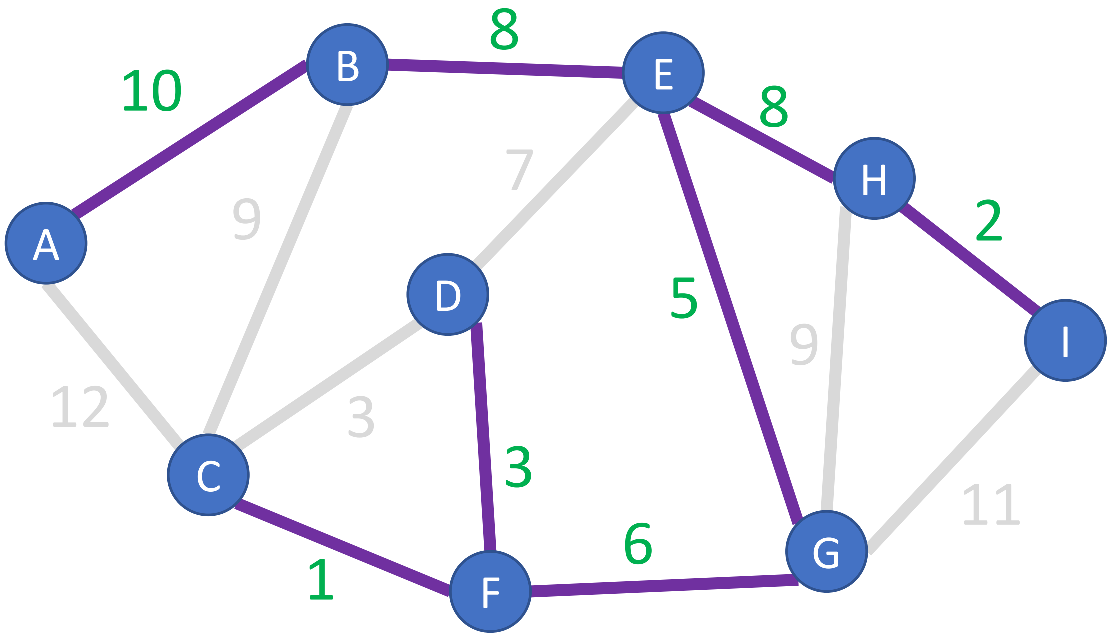
1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle



Edge forms a cycle, so discard

Kruskal's Algorithm

1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle



Edge forms a cycle, so discard

Proof of Correctness: Exchange Argument

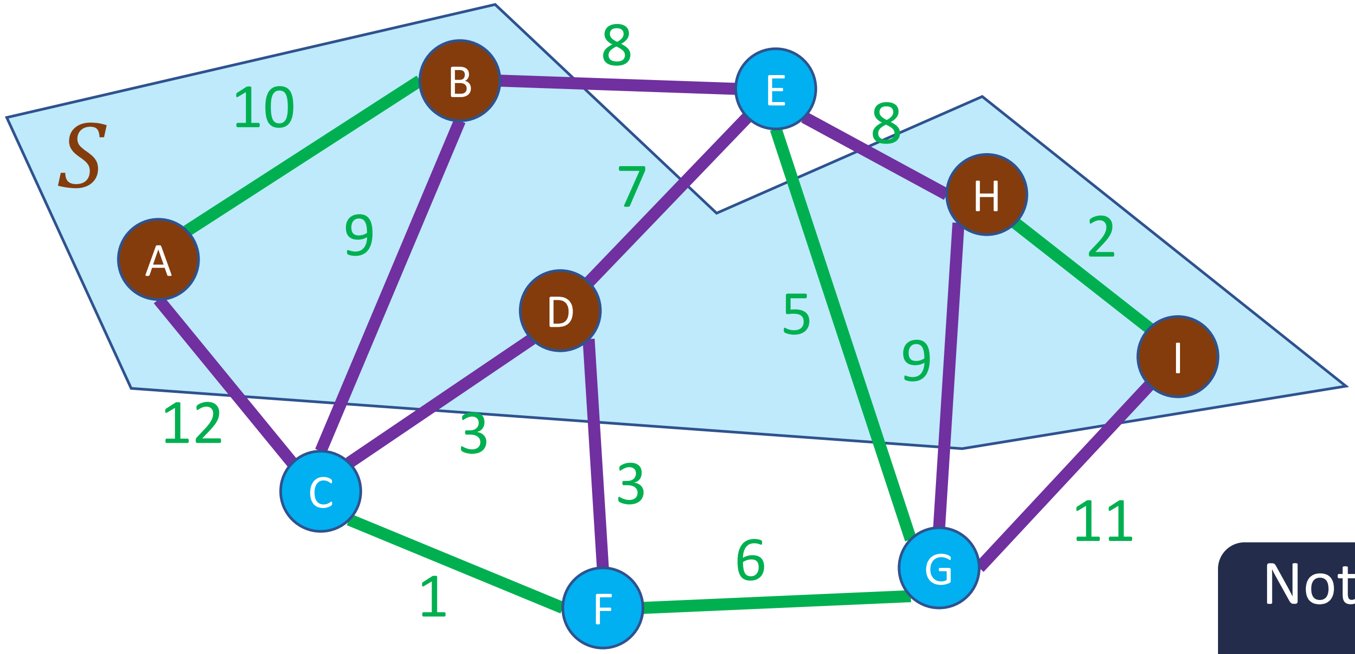
Common technique to show correctness of a greedy algorithm

General idea: argue that at every step, the greedy choice is part of some optimal solution

Approach: Start with an arbitrary optimal solution and show that exchanging an item from the optimal solution with your greedy choice makes the new solution no worse (i.e., the greedy choice is as good as the optimal choice)

Graph Cuts

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



Notion extends naturally to a set of edges

An edge $(v_1, v_2) \in E$ crosses a cut if $v_1 \in S$ and $v_2 \in V - S$

An edge $(v_1, v_2) \in E$ respects a cut if $v_1, v_2 \in S$ or if $v_1, v_2 \in V - S$

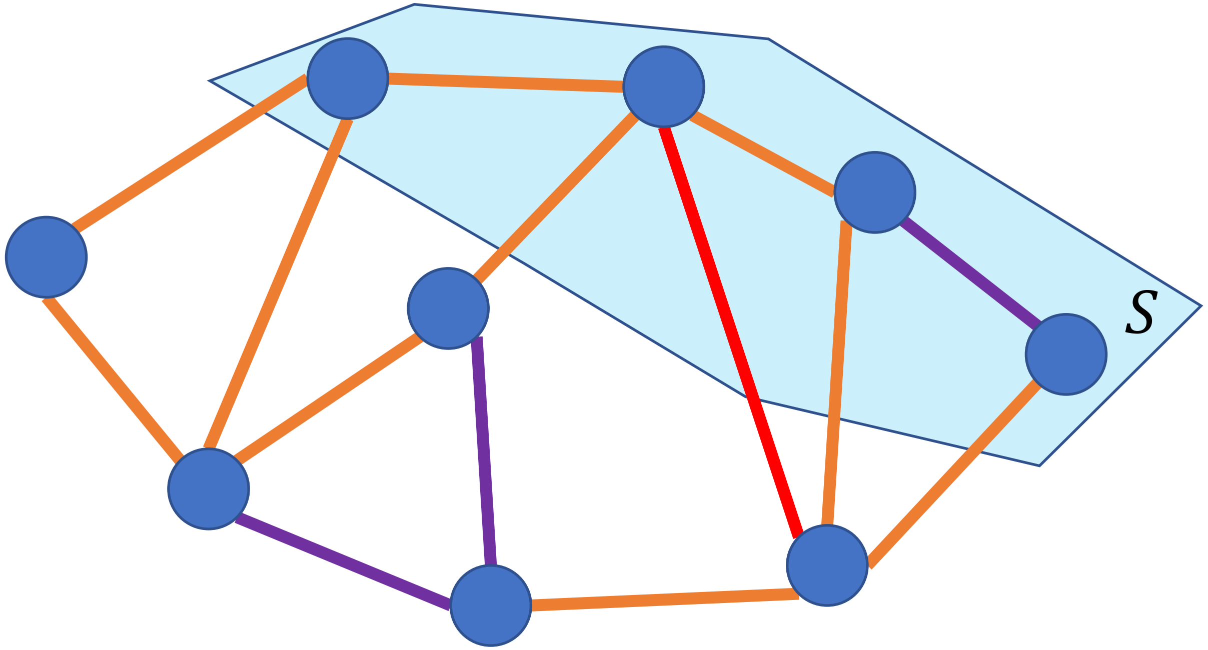
Cut Property of MSTs

Suppose A is a subset of edges of some minimum spanning tree T

Let $(S, V - S)$ be any cut which A respects

Let e be the minimum-weight edge which crosses $(S, V - S)$

Claim: $A \cup \{e\}$ is also a subset of some minimum spanning tree



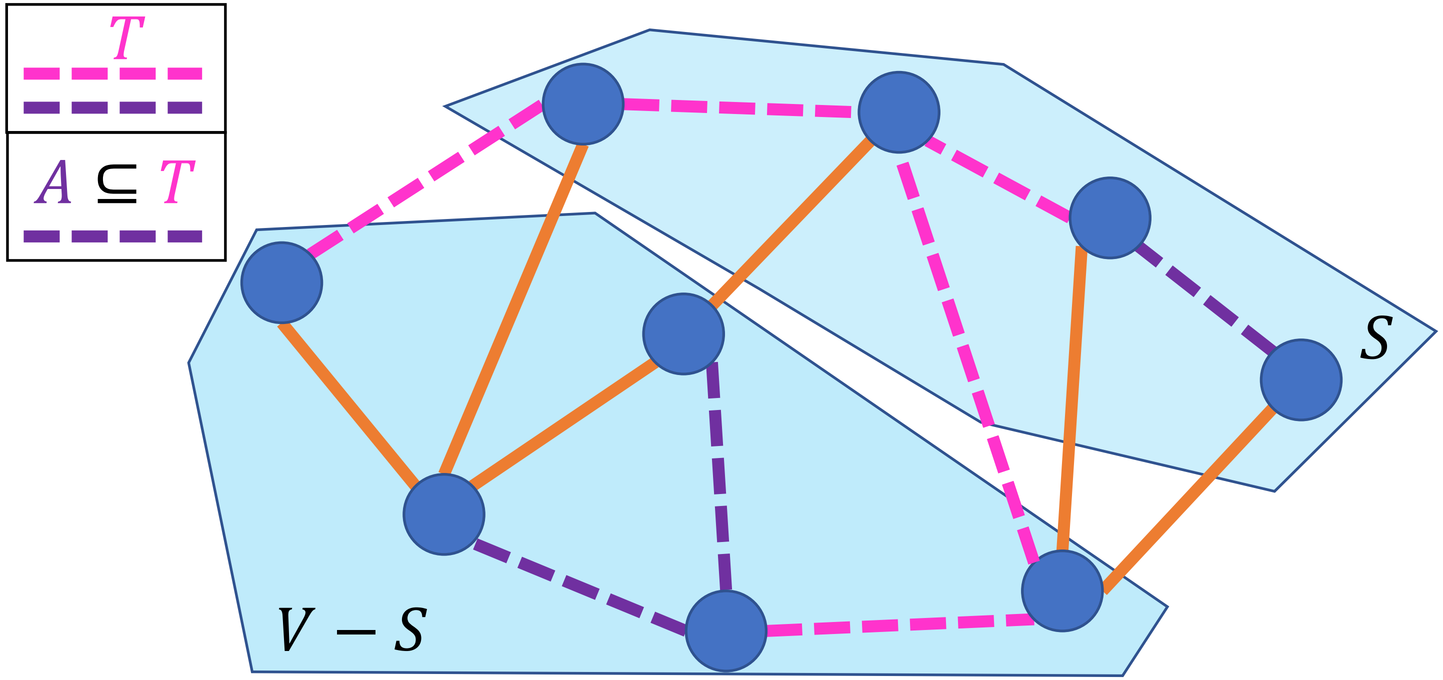
Proof of Cut Property

Suppose A is a subset of edges of some minimum spanning tree T

Let $(S, V - S)$ be any cut which A respects

Let e be the minimum-weight edge which crosses $(S, V - S)$

Claim: $A \cup \{e\}$ is also a subset of some minimum spanning tree



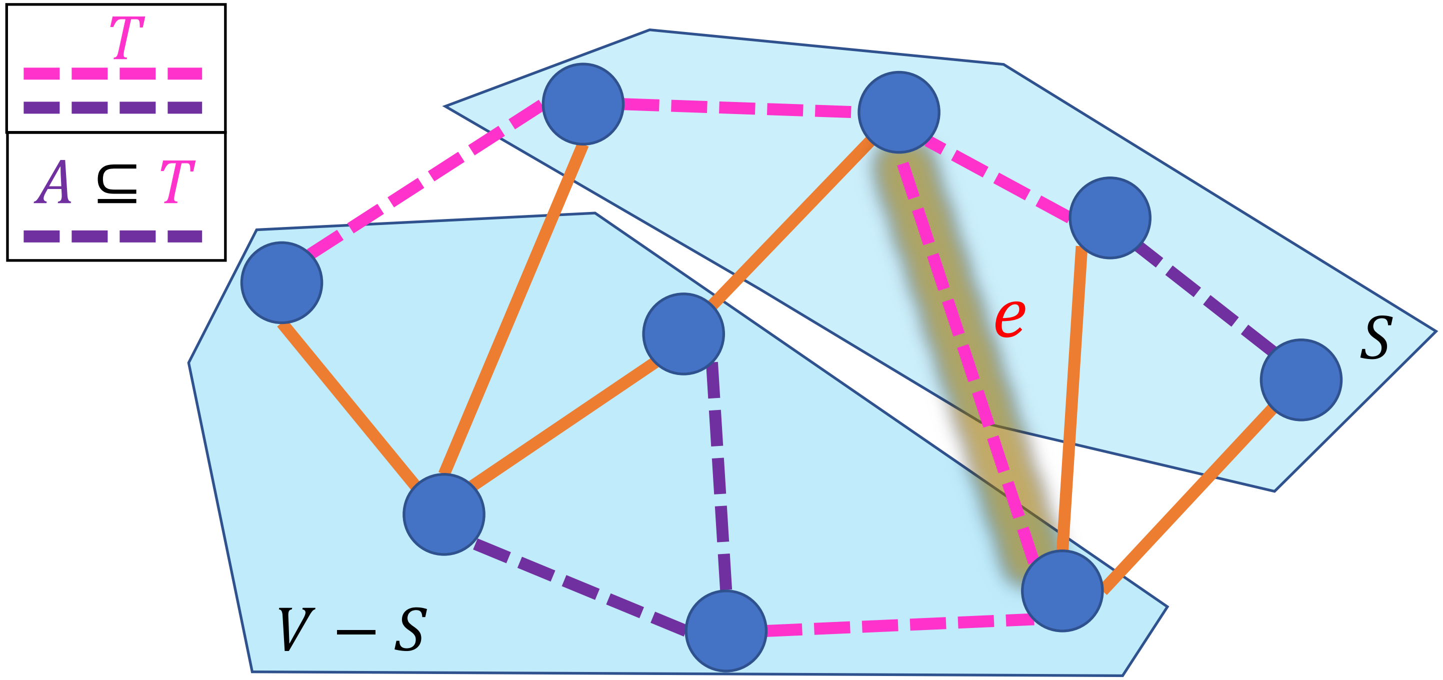
Proof of Cut Property

Suppose A is a subset of edges of some minimum spanning tree T

Let $(S, V - S)$ be any cut which A respects

Let e be the minimum-weight edge which crosses $(S, V - S)$

Claim: $A \cup \{e\}$ is also a subset of <u>some</u> minimum spanning tree



Case 1: $e \in T$

Claim holds

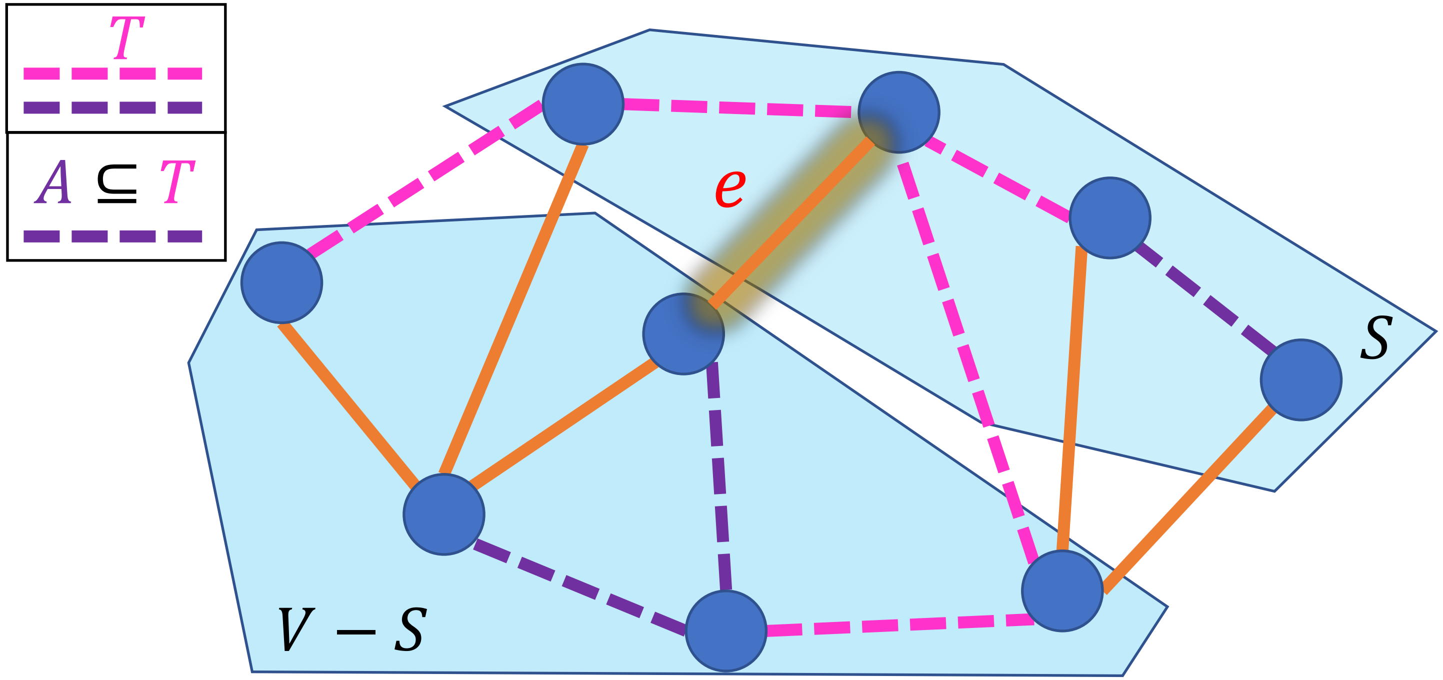
Proof of Cut Property

Suppose A is a subset of edges of some minimum spanning tree T

Let $(S, V-S)$ be any cut which A respects

Let e be the minimum-weight edge which crosses $(S, V-S)$

Claim: $A \cup \{e\}$ is also a subset of some minimum spanning tree



Case 2: $e \notin T$

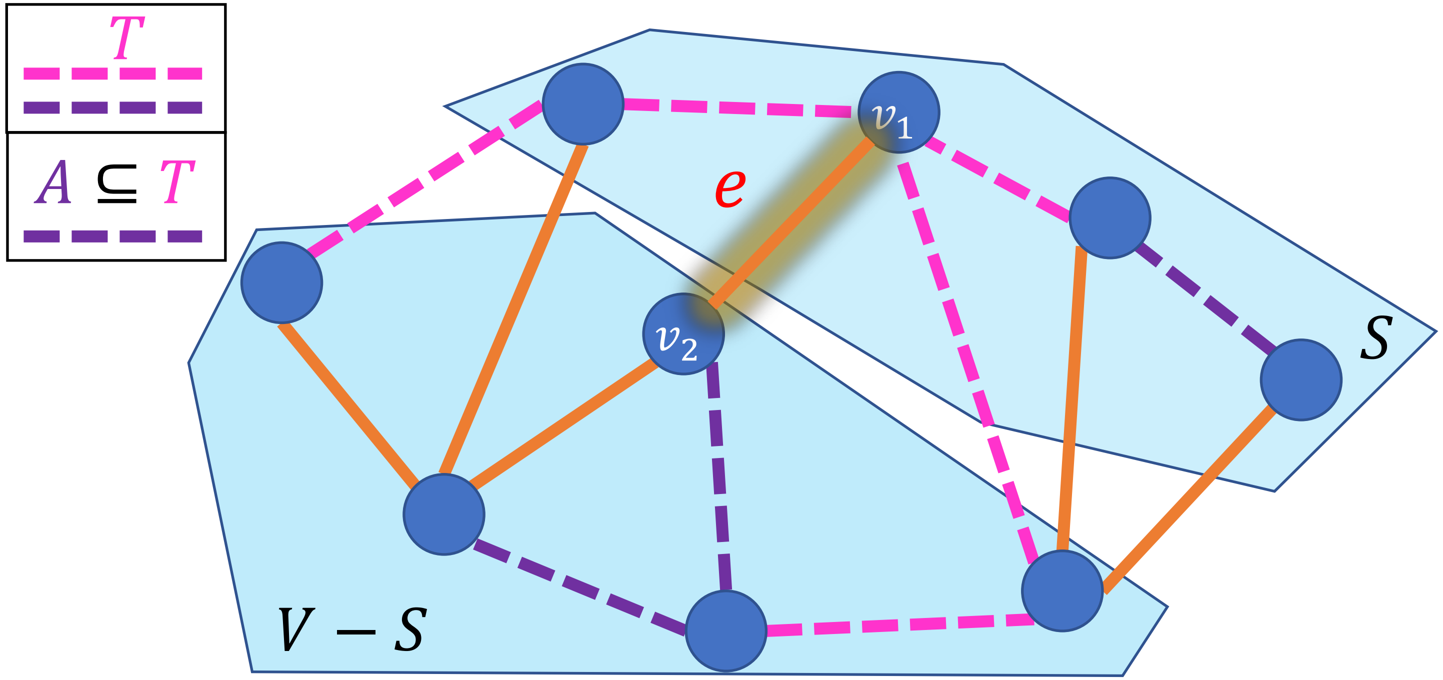
Proof of Cut Property

Suppose A is a subset of edges of some minimum spanning tree T

Let $(S, V - S)$ be any cut which A respects

Let e be the minimum-weight edge which crosses $(S, V - S)$

Claim: $A \cup \{e\}$ is also a subset of some minimum spanning tree



Case 2: $e \notin T$

Let $e = (v_1, v_2)$

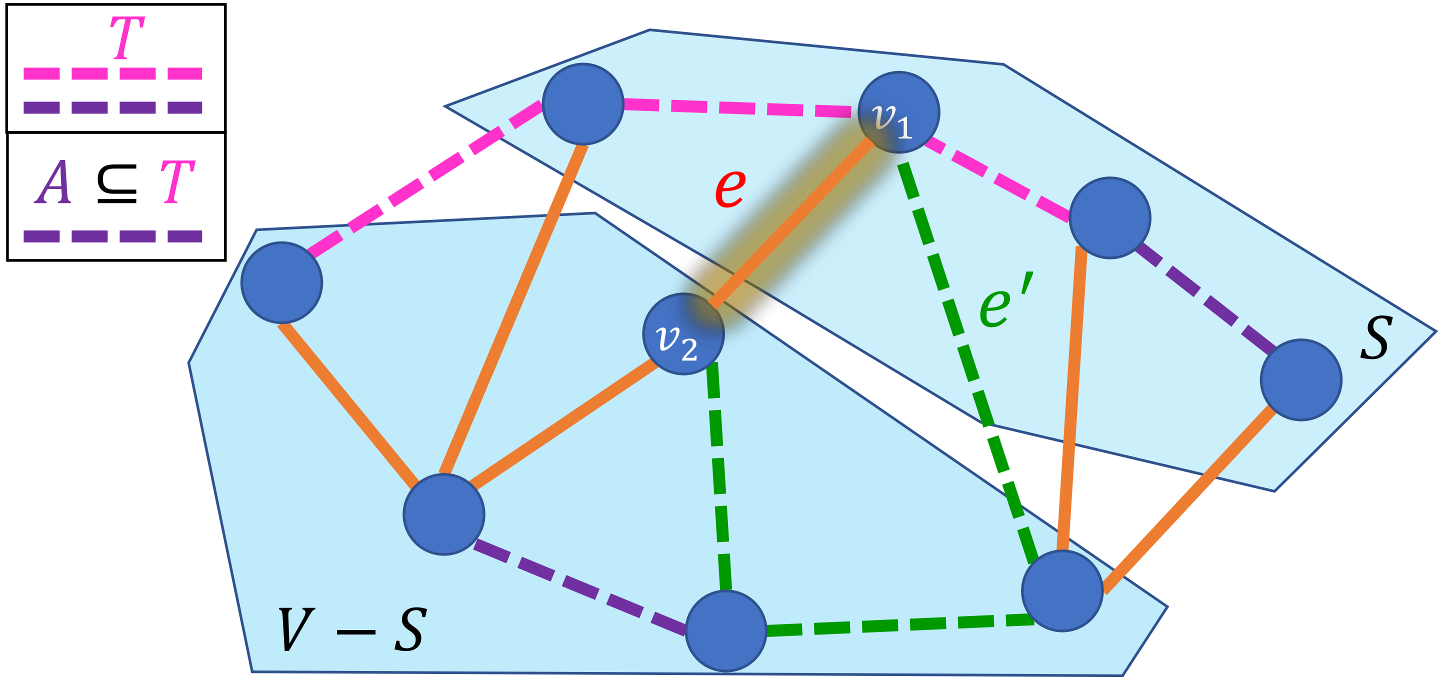
Proof of Cut Property

Suppose A is a subset of edges of some minimum spanning tree T

Let $(S, V - S)$ be any cut which A respects

Let e be the minimum-weight edge which crosses $(S, V - S)$

Claim: $A \cup \{e\}$ is also a subset of <u>some</u> minimum spanning tree



Case 2: $e \notin T$

Let $e = (v_1, v_2)$

Since T is a spanning tree, there is a path from v_1 to v_2 in T

Let e' be an edge that crosses the cut

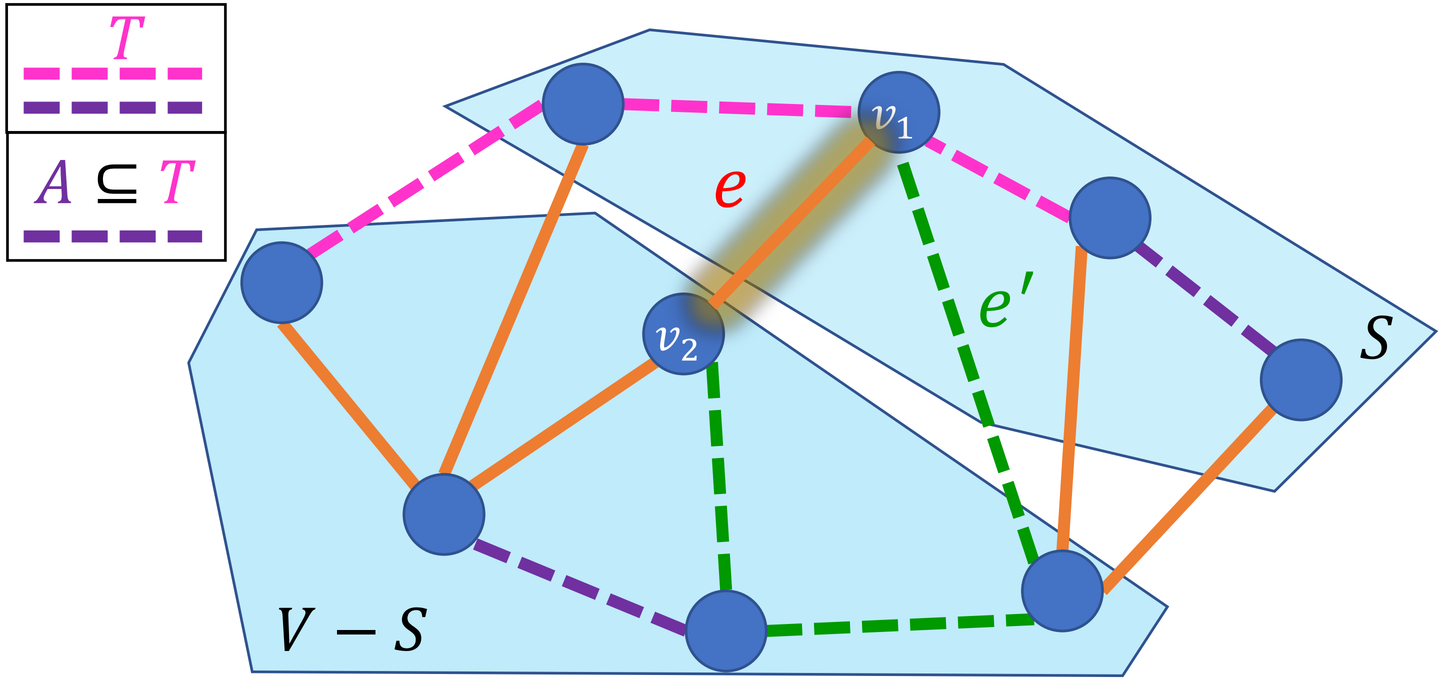
Replace e' with e in T

Proof of Cut Property

Let T' be the tree obtained by replacing e' with e in T

- T' is still a spanning tree (all nodes in S and $V-S$ are connected, and there is an edge between S and $V-S$)
- $\text{Cost}(T') = \text{Cost}(T) - w(e') + w(e) \leq \text{Cost}(T)$ since $w(e') \geq w(e)$

Conclusion: if T is a MST, then so is T'



Case 2: $e \notin T$

Let $e = (v_1, v_2)$

Since T is a spanning tree, there is a path from v_1 to v_2 in T

Let e' be an edge that crosses the cut

Replace e' with e in T

Correctness of Kruskal's Algorithm

1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle

Let T_0 be the initial (empty) tree, and T_i be the tree after adding i edges (using the greedy strategy above).

Claim: If T_i is consistent with some MST, then T_{i+1} is also consistent with some MST

Proof of Kruskal's Theorem: Follows by induction on the number of nodes in G:

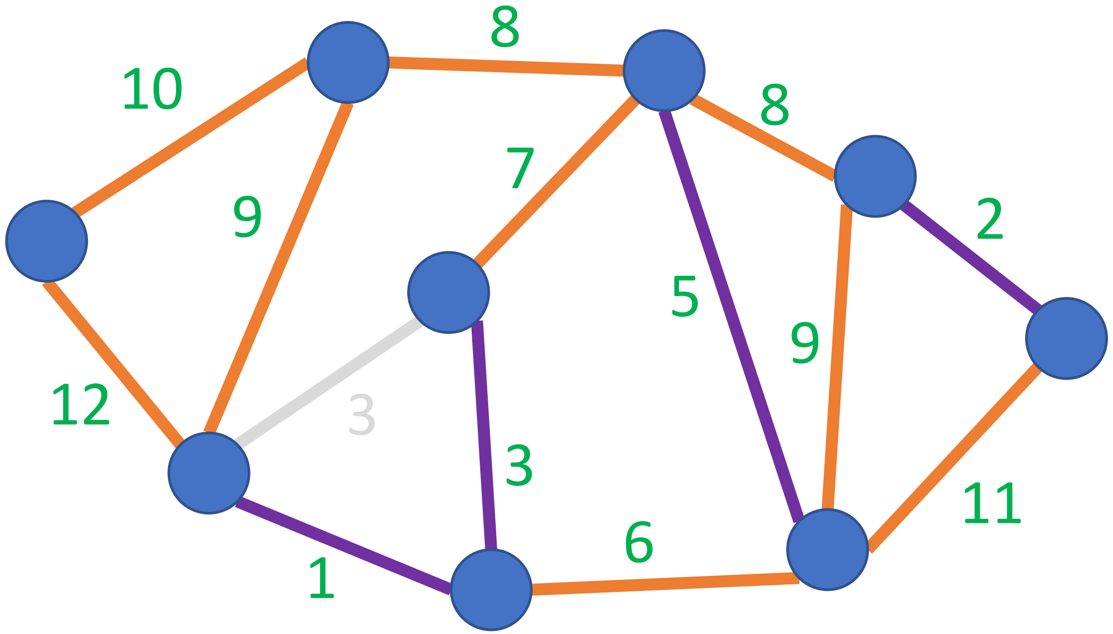
- T_0: an empty tree is (trivially) consistent with an MST
- By the above claim, if T_i is consistent with some MST, so is T_{i+1}

Conclusion: $T_{|V|-1}$ is consistent with some MST, which is the output of the algorithm

Correctness of Kruskal’s Algorithm

Let T_0 be the initial (empty) tree, and T_i be the tree after adding i edges (according to the specification of Kruskal’s algorithm)

Claim: If T_i is consistent with some MST, then T_{i+1} is also consistent with some MST

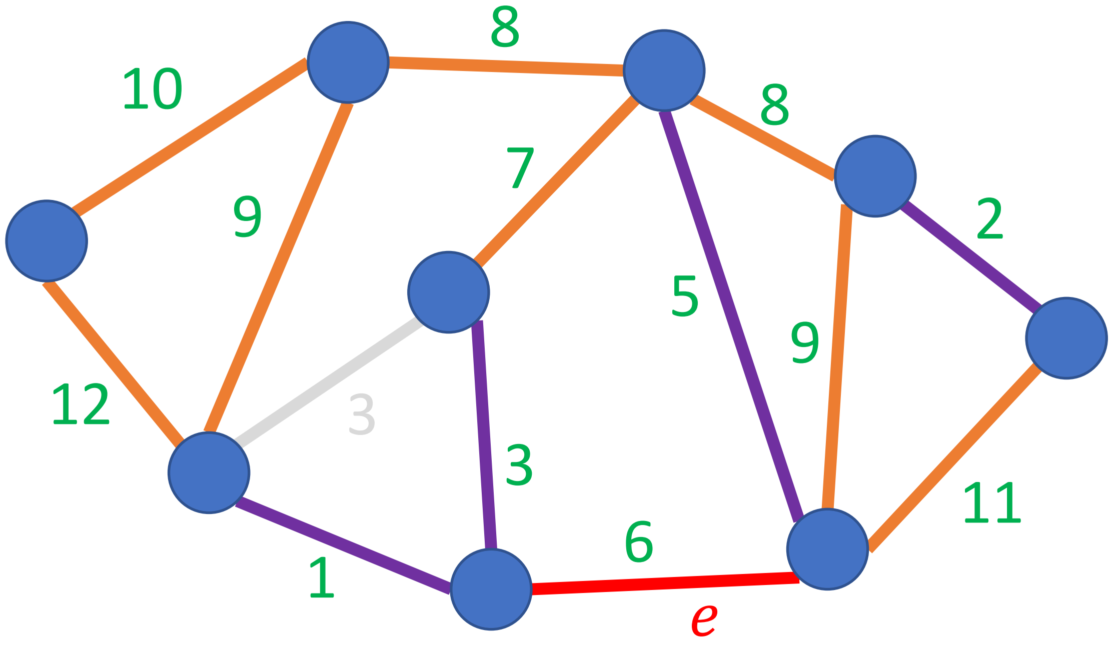


Tree T_i after adding i nodes

Correctness of Kruskal’s Algorithm

Let T_0 be the initial (empty) tree, and T_i be the tree after adding i edges (according to the specification of Kruskal’s algorithm)

Claim: If T_i is consistent with some MST, then T_{i+1} is also consistent with some MST



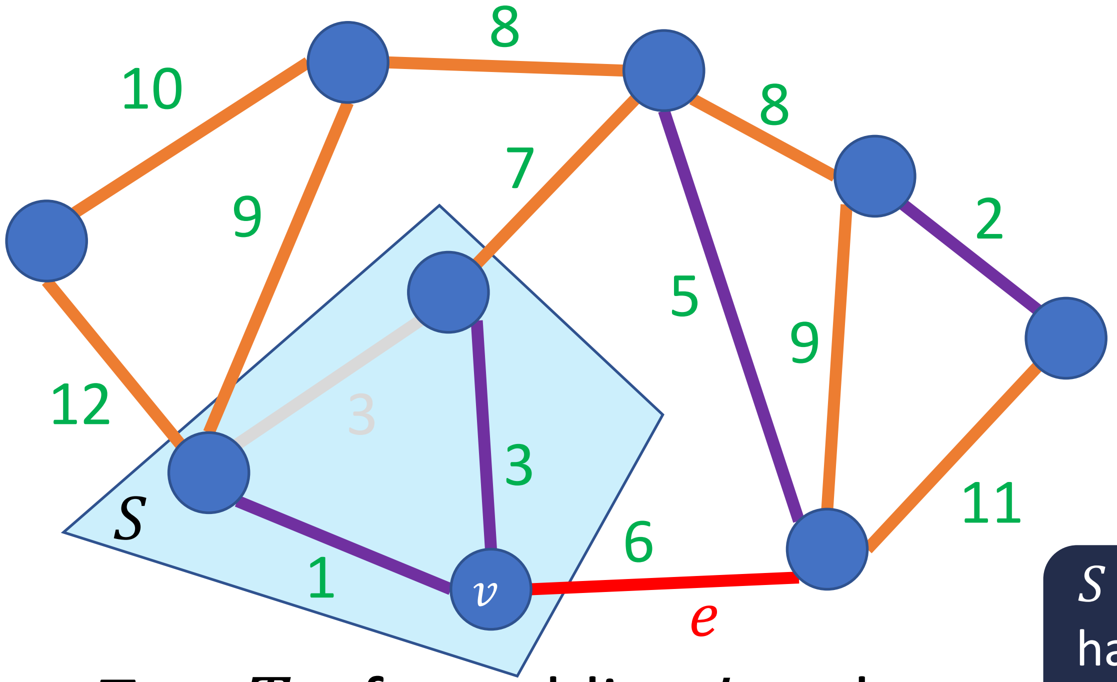
Tree T_i after adding i nodes

Consider edge e chosen by Kruskal’s algorithm

Correctness of Kruskal’s Algorithm

Let T_0 be the initial (empty) tree, and T_i be the tree after adding i edges (according to the specification of Kruskal’s algorithm)

Claim: If T_i is consistent with some MST, then T_{i+1} is also consistent with some MST



Tree T_i after adding i nodes

Consider edge e chosen by Kruskal’s algorithm

Choose one of the endpoints v of e arbitrarily and let S be the set of nodes reachable from v in T_i

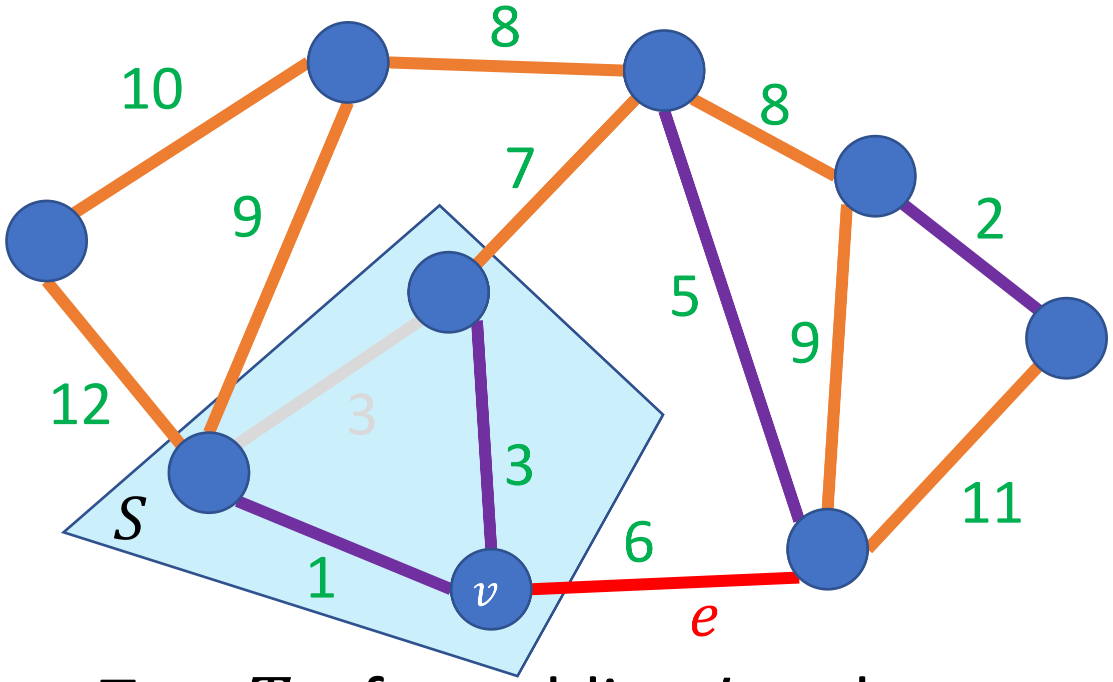
By assumption, T_i is consistent with some MST and respects the cut $(S, V - S)$

S is the set of nodes reachable from v: cannot have an edge between node reachable from V and one not reachable from V

Correctness of Kruskal's Algorithm

Let T_0 be the initial (empty) tree, and T_i be the tree after adding i edges (according to the specification of Kruskal's algorithm)

Claim: If T_i is consistent with some MST, then T_{i+1} is also consistent with some MST



Tree T_i after adding i nodes

Consider edge e chosen by Kruskal's algorithm

Choose one of the endpoints v of e arbitrarily and let S be the set of nodes reachable from v in T_i

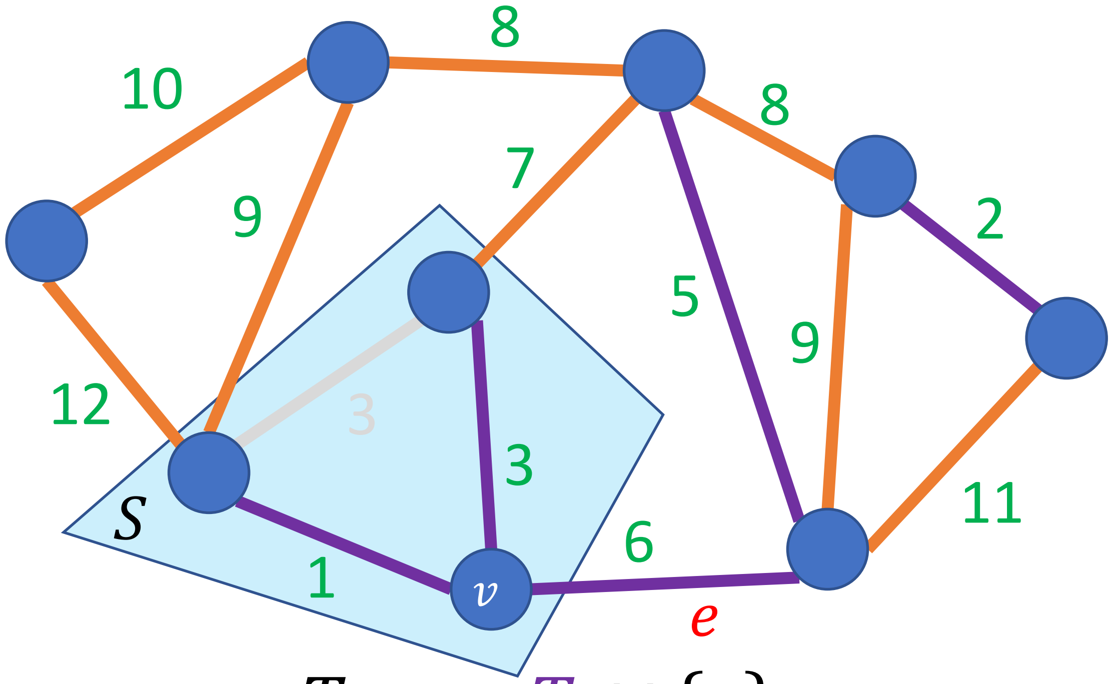
By assumption, T_i is consistent with some MST and respects the cut $(S, V - S)$

Cut property: $T_i \cup \{e\} = T_{i+1}$ is also consistent with some MST

Correctness of Kruskal's Algorithm

Let T_0 be the initial (empty) tree, and T_i be the tree after adding i edges (according to the specification of Kruskal's algorithm)

Claim: If T_i is consistent with some MST, then T_{i+1} is also consistent with some MST



$T_{i+1} = T_i \cup \{e\}$

Consider edge e chosen by Kruskal's algorithm

Choose one of the endpoints v of e arbitrarily and let S be the set of nodes reachable from v in T_i

By assumption, T_i is consistent with some MST and respects the cut $(S, V - S)$

Cut property: $T_i \cup \{e\} = T_{i+1}$ is also consistent with some MST

Kruskal’s Algorithm

1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle

Implementation: iterate over each of the edges in the graph (sorted by weight), and maintain nodes in a union-find (also called disjoint-set) data structure:

- Data structure that tracks elements partitioned into different sets
- **Union:** Merges two sets into one
- **Find:** Given an element, return the index of the set it belongs to
- Both “union” and “find” operations are very fast

Time complexity: $O\big(\alpha(n)\big)$,
where α is the “inverse Ackermann function” (extremely slow-growing function)
for all “practical” n, $\alpha(n) < 5$ (e.g., for all $n < 2^{2^{2^{65536}}} - 3$)

Kruskal’s Algorithm

1. Start with an empty tree T
2. Repeatedly add to T the lowest-weight edge that does not create a cycle

Implementation: iterate over each of the edges in the graph (sorted by weight), and maintain nodes in a union-find (also called disjoint-set) data structure:

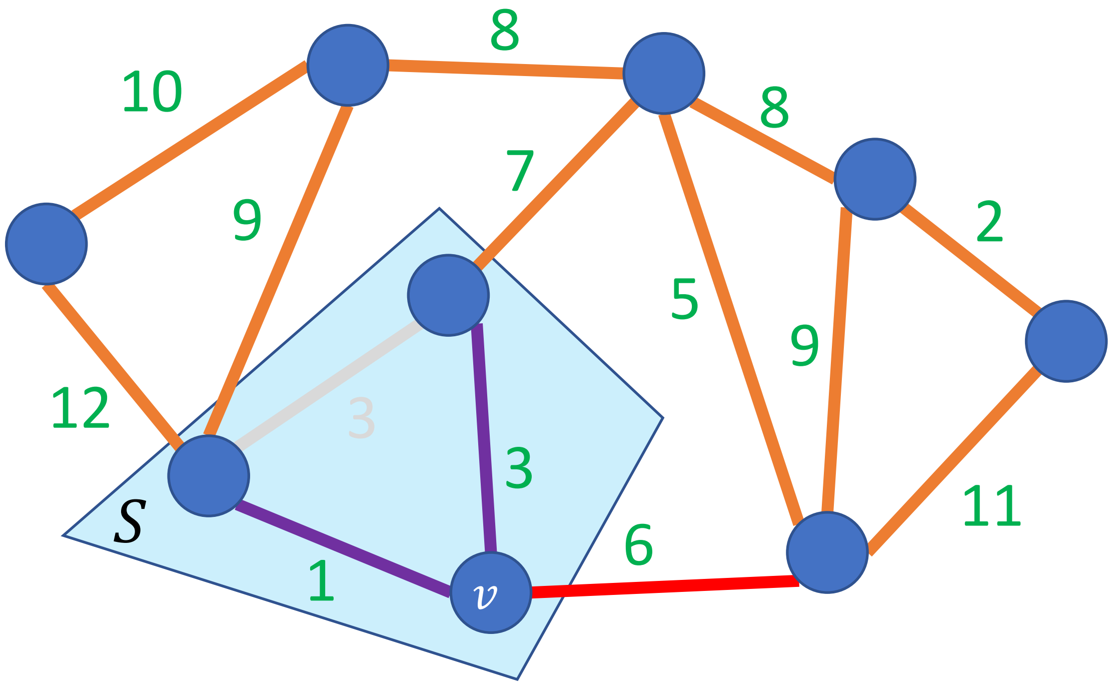
- Data structure that tracks elements partitioned into different sets
- **Union:** Merges two sets into one
- **Find:** Given an element, return the index of the set it belongs to
- Both “union” and “find” operations are very fast

- **Overall running time:** $O(|E| \log |E|) = O(|E| \log |V|)$

$$|E| \leq |V|^2 \Rightarrow \log|E| = O(\log|V|)$$

General MST Algorithm

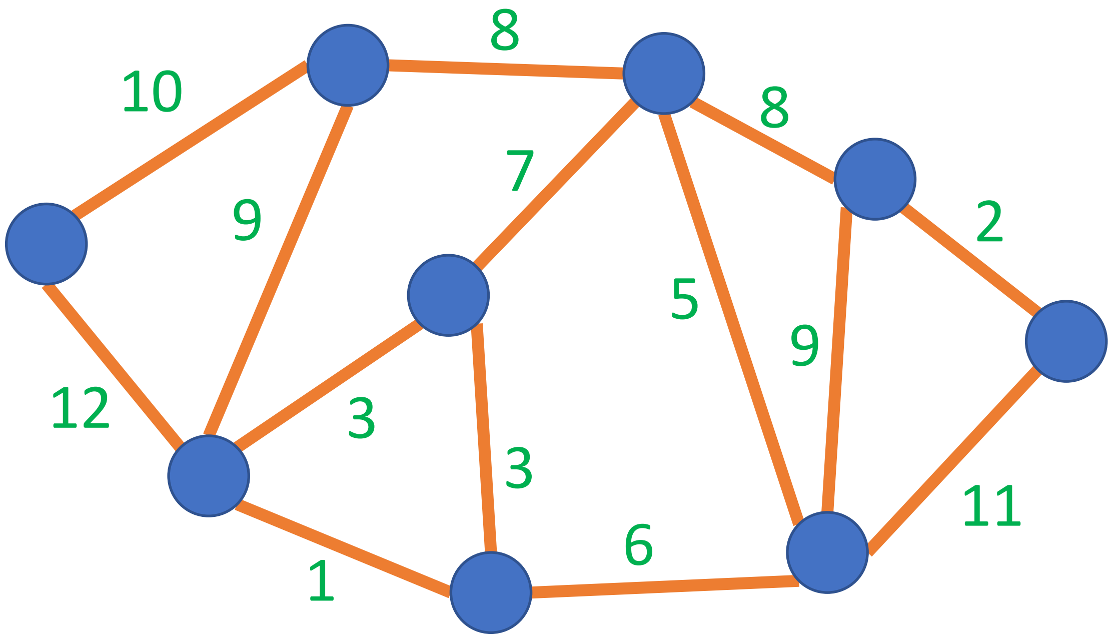
1. Start with an empty tree T
2. Repeat $|V| - 1$ times:
    - Pick a cut $(S, V - S)$ which T respects
    - Add the min-weight edge which crosses $(S, V - S)$



Correctness analysis follows by repeated application of Cut Property

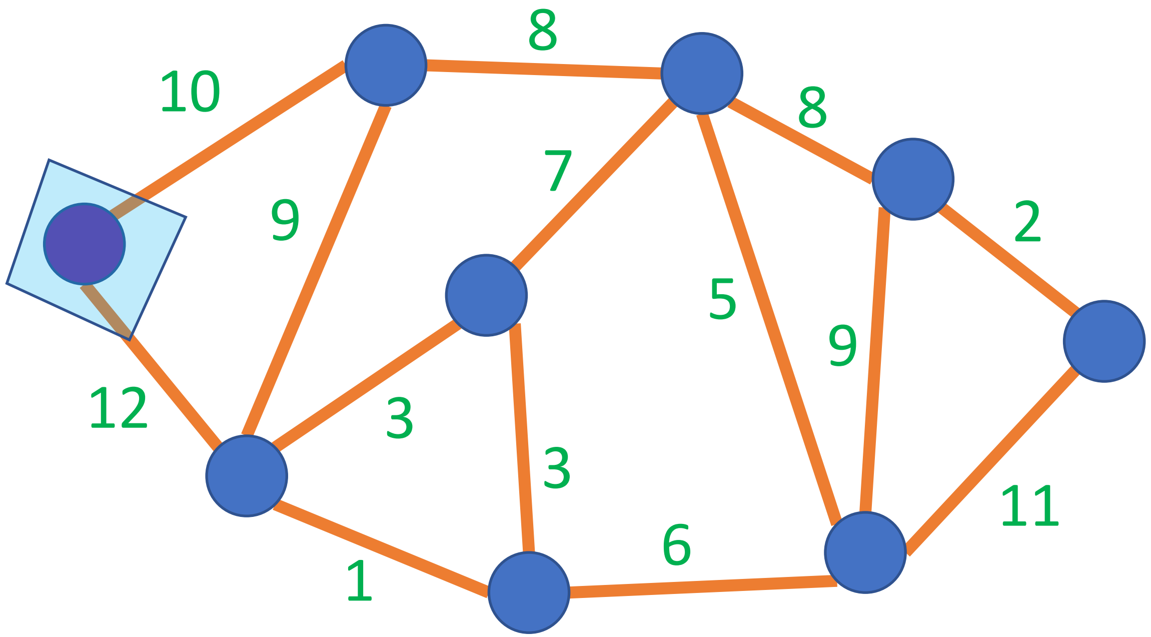
Prim’s Algorithm

1. Start with an empty tree T and pick a start node and add it to T
2. Repeat $|V| - 1$ times:
   - Add the min-weight edge which connects a node in T with a node not in T



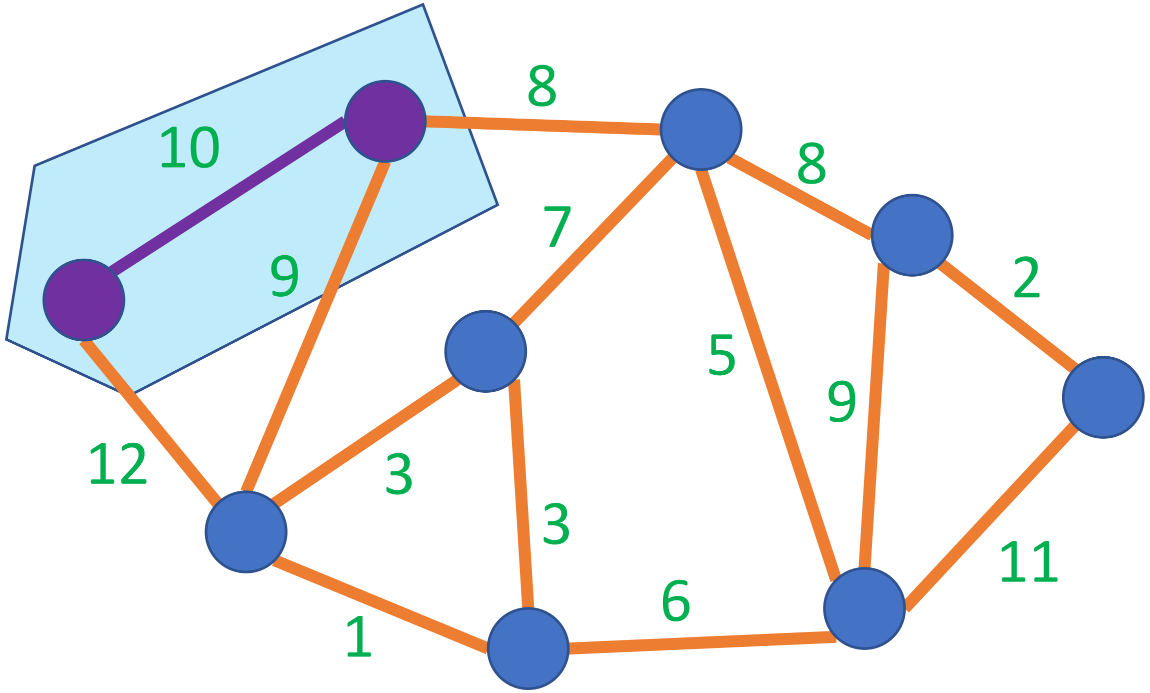
Prim’s Algorithm

1. Start with an empty tree T and pick a start node and add it to T
2. Repeat $|V| - 1$ times:
   - Add the min-weight edge which connects a node in T with a node not in T



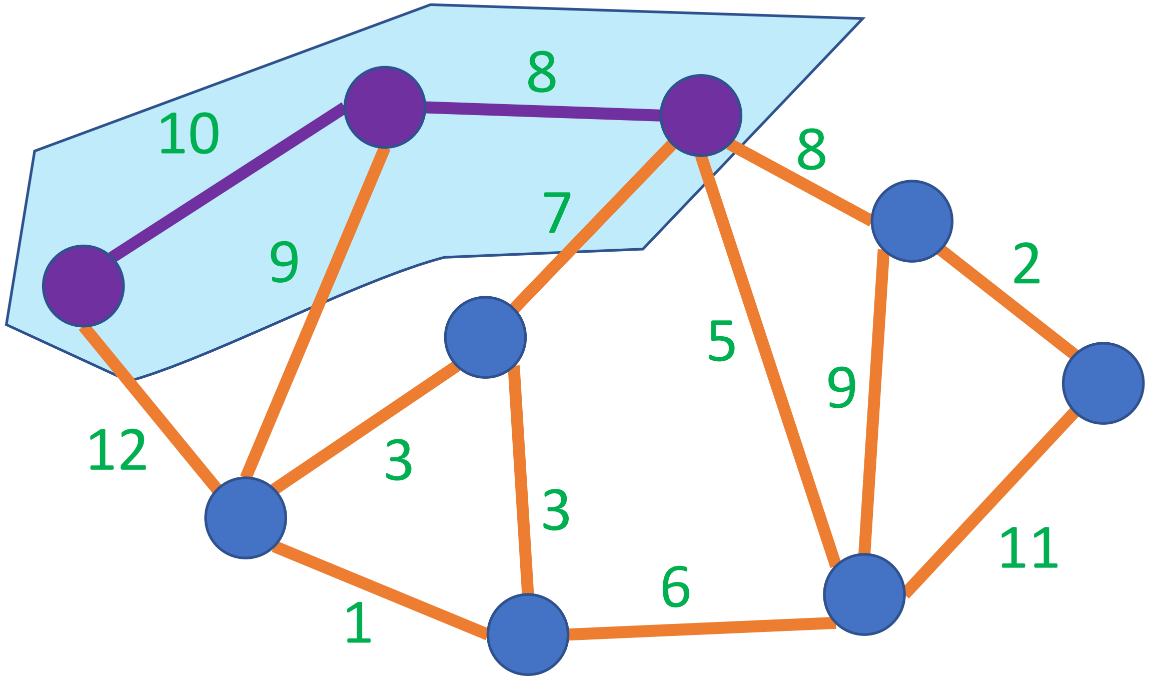
Prim’s Algorithm

1. Start with an empty tree T and pick a start node and add it to T
2. Repeat $|V| - 1$ times:
   - Add the min-weight edge which connects a node in T with a node not in T



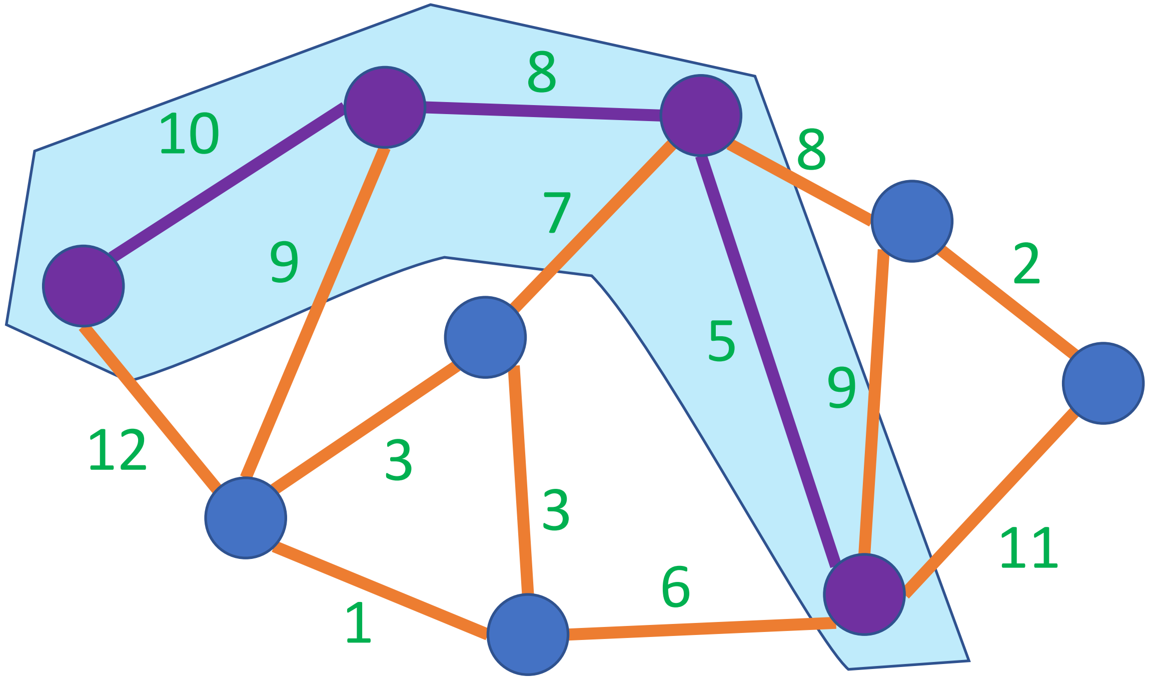
Prim’s Algorithm

1. Start with an empty tree T and pick a start node and add it to T
2. Repeat $|V| - 1$ times:
   - Add the min-weight edge which connects a node in T with a node not in T



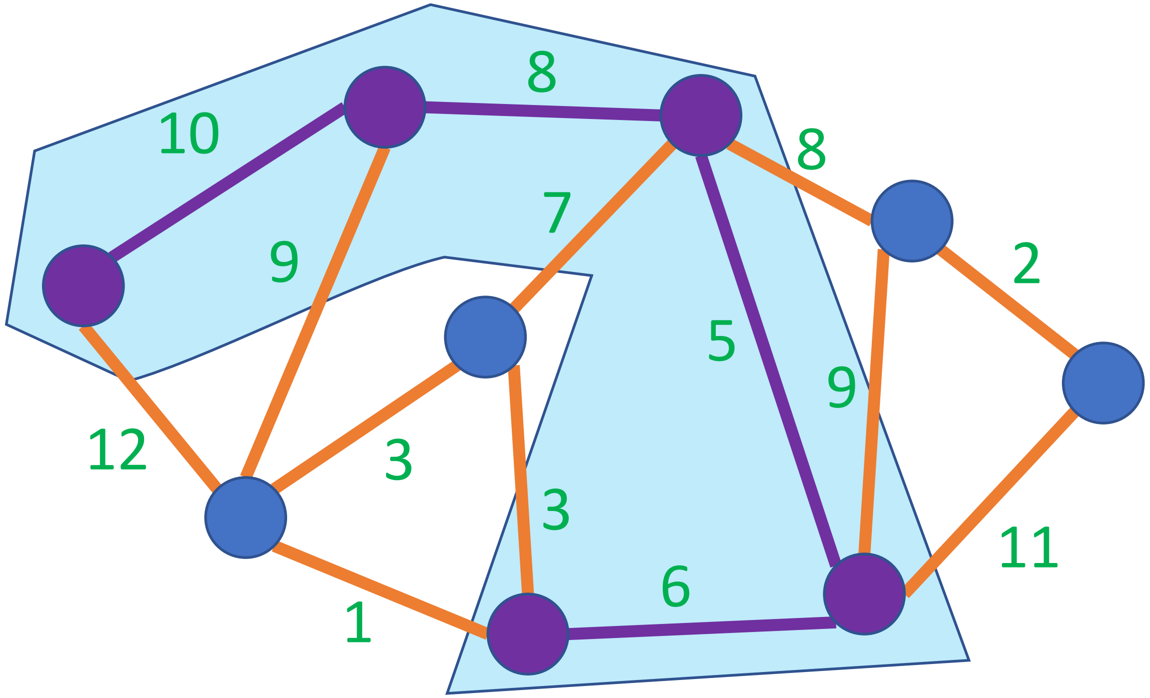
Prim's Algorithm

1. Start with an empty tree T and pick a start node and add it to T
2. Repeat $|V| - 1$ times:
   - Add the min-weight edge which connects a node in T with a node not in T



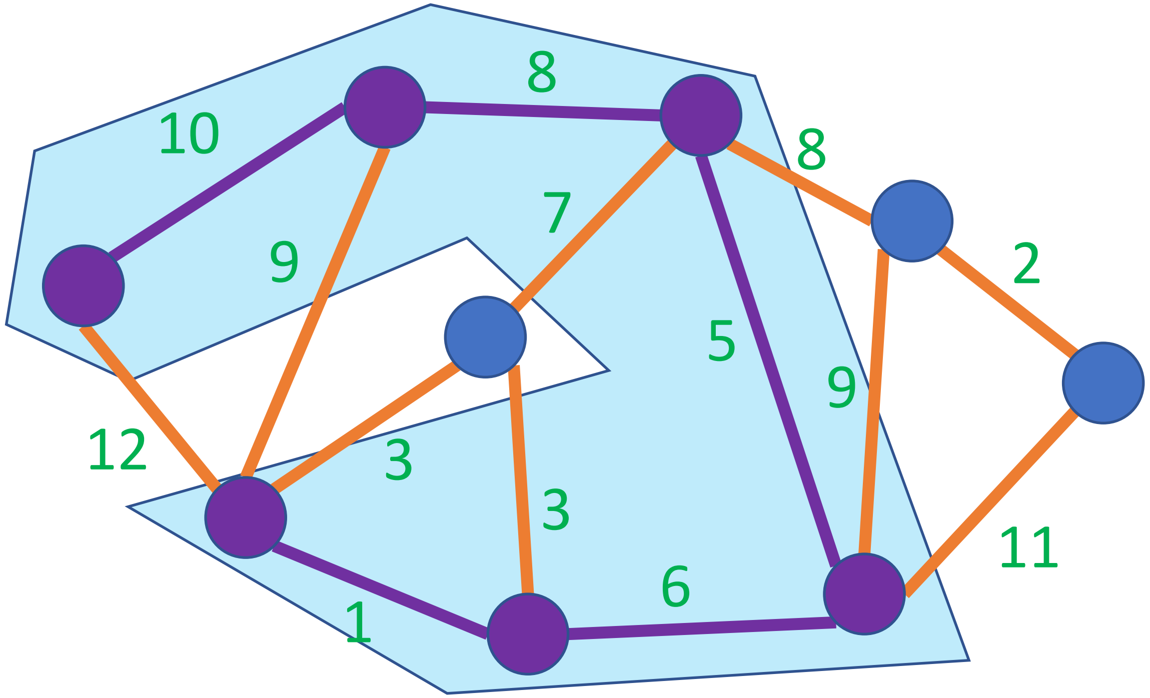
Prim’s Algorithm

1. Start with an empty tree T and pick a start node and add it to T
2. Repeat $|V| - 1$ times:
   - Add the min-weight edge which connects a node in T with a node not in T



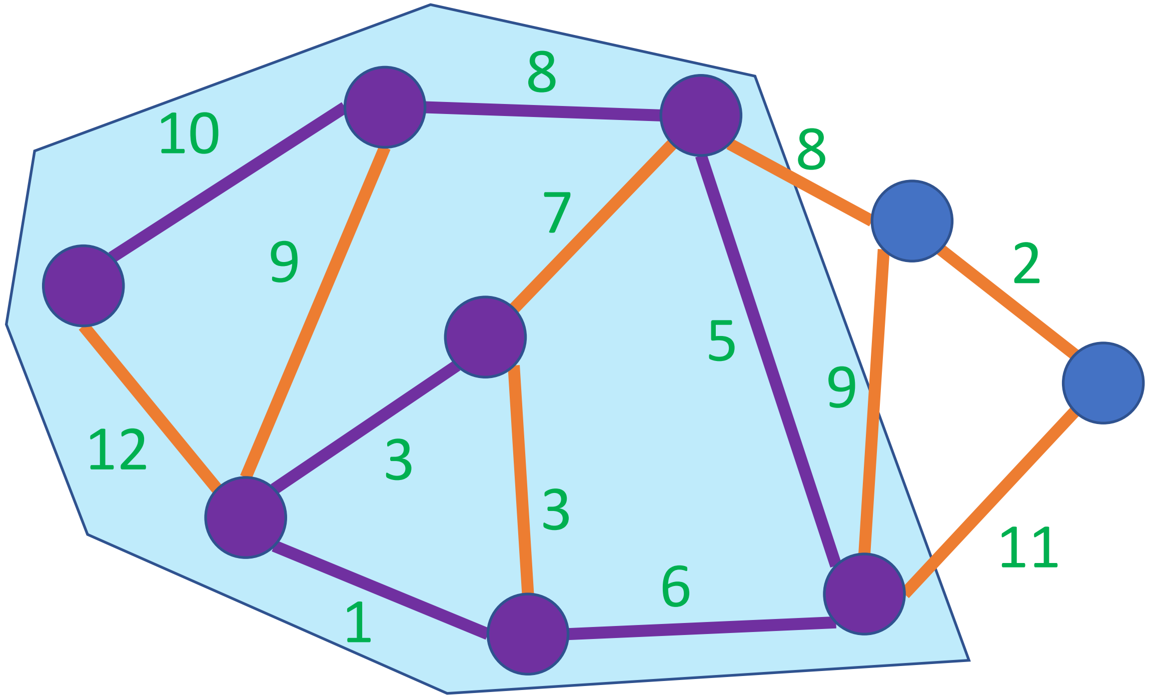
Prim's Algorithm

1. Start with an empty tree T and pick a start node and add it to T
2. Repeat $|V| - 1$ times:
   - Add the min-weight edge which connects a node in T with a node not in T



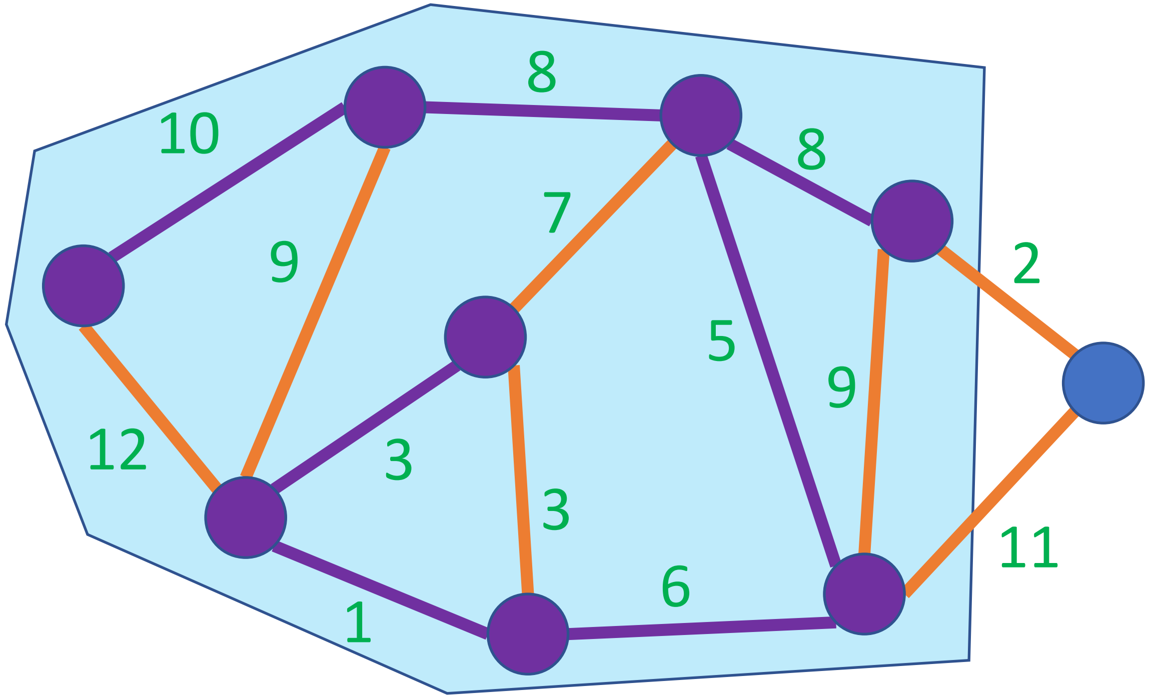
Prim's Algorithm

1. Start with an empty tree T and pick a start node and add it to T
2. Repeat $|V| - 1$ times:
   - Add the min-weight edge which connects a node in T with a node not in T



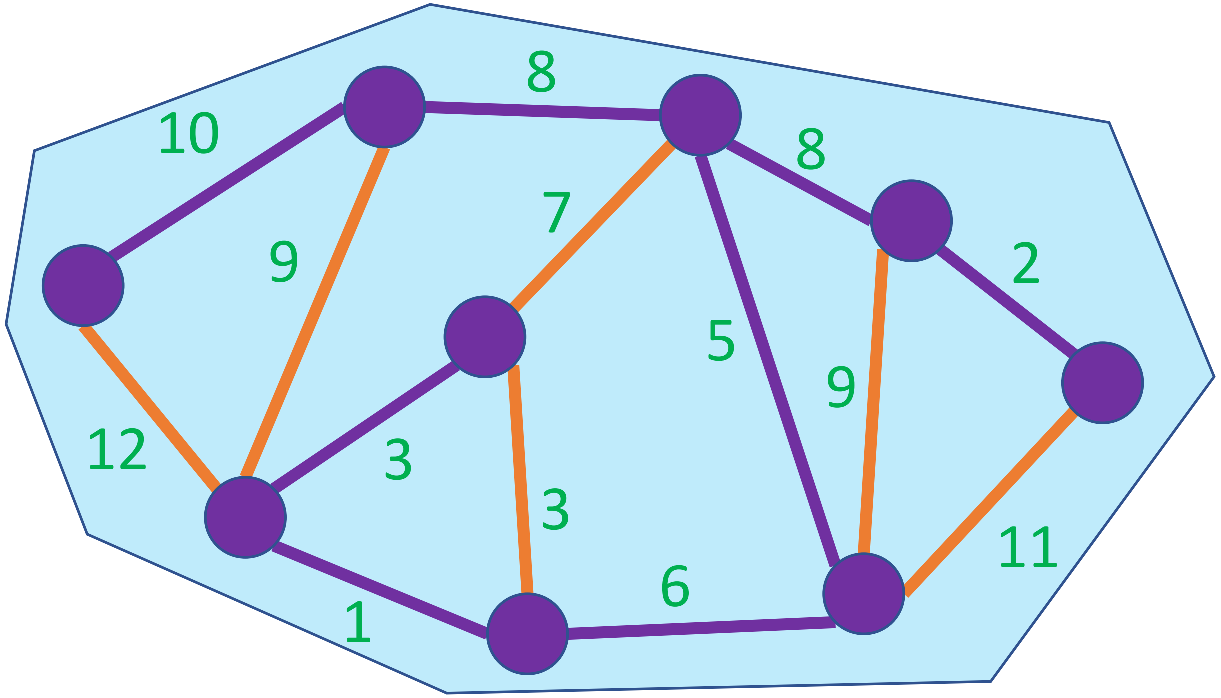
Prim's Algorithm

1. Start with an empty tree T and pick a start node and add it to T
2. Repeat $|V| - 1$ times:
   - Add the min-weight edge which connects a node in T with a node not in T



Prim's Algorithm

1. Start with an empty tree T and pick a start node and add it to T
2. Repeat $|V| - 1$ times:
    - Add the min-weight edge which connects a node in T with a node not in T



Prim's Algorithm

1. Start with an empty tree T and pick a start node and add it to T
2. Repeat $|V| - 1$ times:
    - Add the min-weight edge which connects a node in T with a node not in T

Implementation:

- Maintain edges incident on T in a min-heap (priority queue)
- Maintain a (sorted) list of nodes that have already been added to the tree
- Each time node v is added to the tree, add all edges incident on v to heap
- To find the next edge to add, repeatedly extract from heap until finding an edge incident on node that is not currently contained in the tree

Overall running time: $O(|E| \log |V|)$

- If we use Fibonacci heaps instead of binary heaps: $O(|E| + |V| \log|V|)$

MST Algorithms

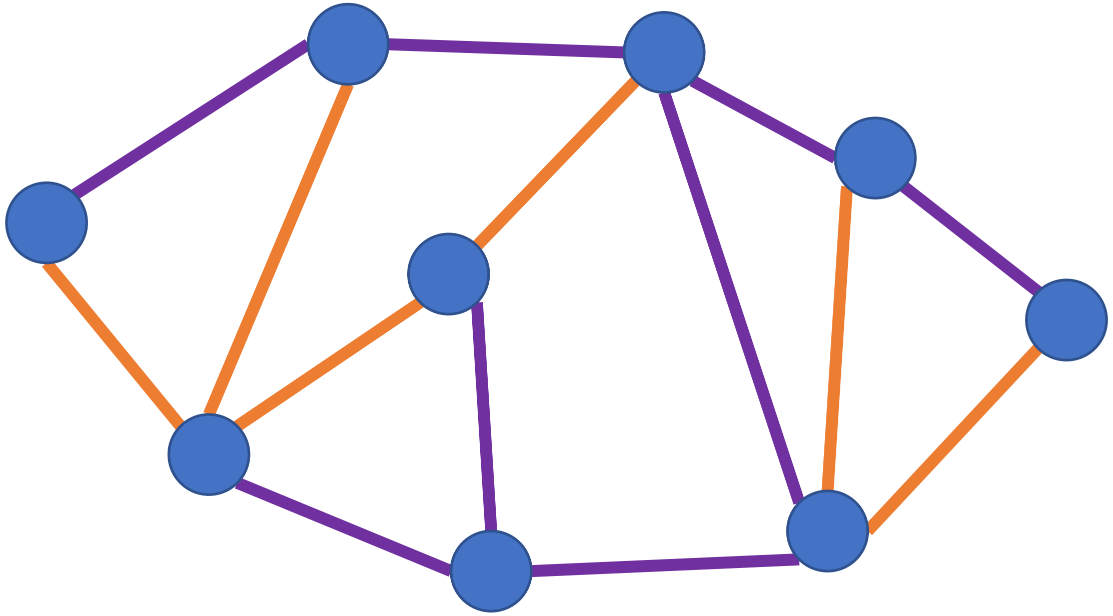
Kruskal ’56; Prim ‘57:	$O(\lvert E\rvert \log\lvert V\rvert)$
Fredman-Tarjan ‘84:	$O(\lvert E\rvert + \lvert V\rvert \log\lvert V\rvert)$
Gabow-Galil-Spencer-Tarjan ‘86:	$O(\lvert E\rvert \log(\log^*\lvert V\rvert))$
Chazelle ‘00:	$O(\lvert E\rvert \cdot \alpha(\lvert V\rvert))$
Pettie-Ramachandran ’02:	$O(?)$ (optimal, but unknown running time)
Karger-Klein-Tarjan ‘95:	$O(\lvert E\rvert)$ (in expectation)

Extra Credit: Read + summarize any of these algorithms (other than Kruskal/Prim)

Cycle Property of MSTs

Take any cycle in a graph $G = (V, E)$

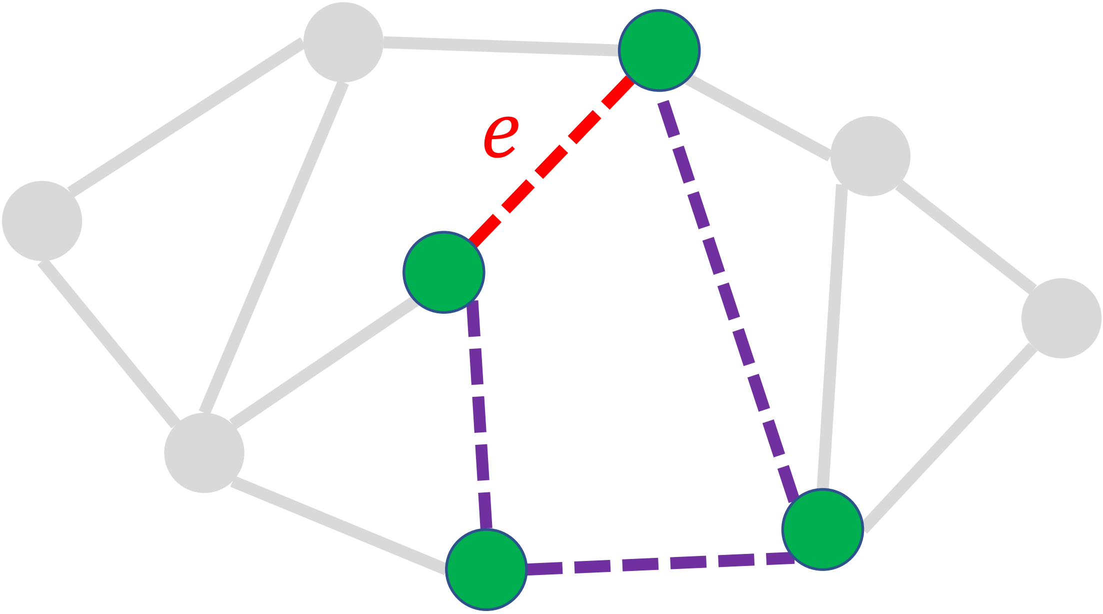
Then, there exists some MST of G that does not contain the maximum-weight edge on that cycle



Cycle Property of MSTs

Take any cycle in a graph $G = (V, E)$

Then, there exists some MST of G that does not contain the maximum-weight edge on that cycle



Proof. Take any cycle $(v_1, v_2, \ldots, v_t, v_1)$ in G and take any MST T of G

Let e be the maximum-weight edge in the cycle

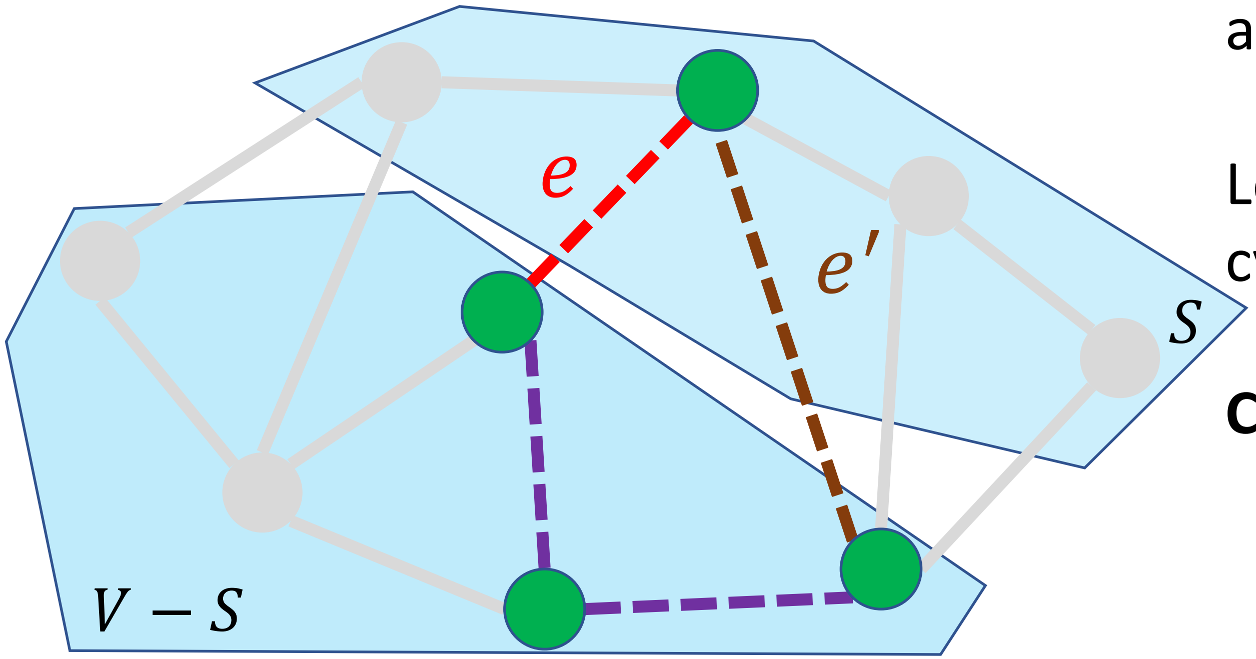
Case 1: $e \notin T$

- Claim follows

Cycle Property of MSTs

Take any cycle in a graph $G = (V, E)$

Then, there exists some MST of G that does not contain the maximum-weight edge on that cycle



Proof. Take any cycle $(v_1, v_2, \ldots, v_t, v_1)$ in G and take any MST T of G

Let e be the maximum-weight edge in the cycle

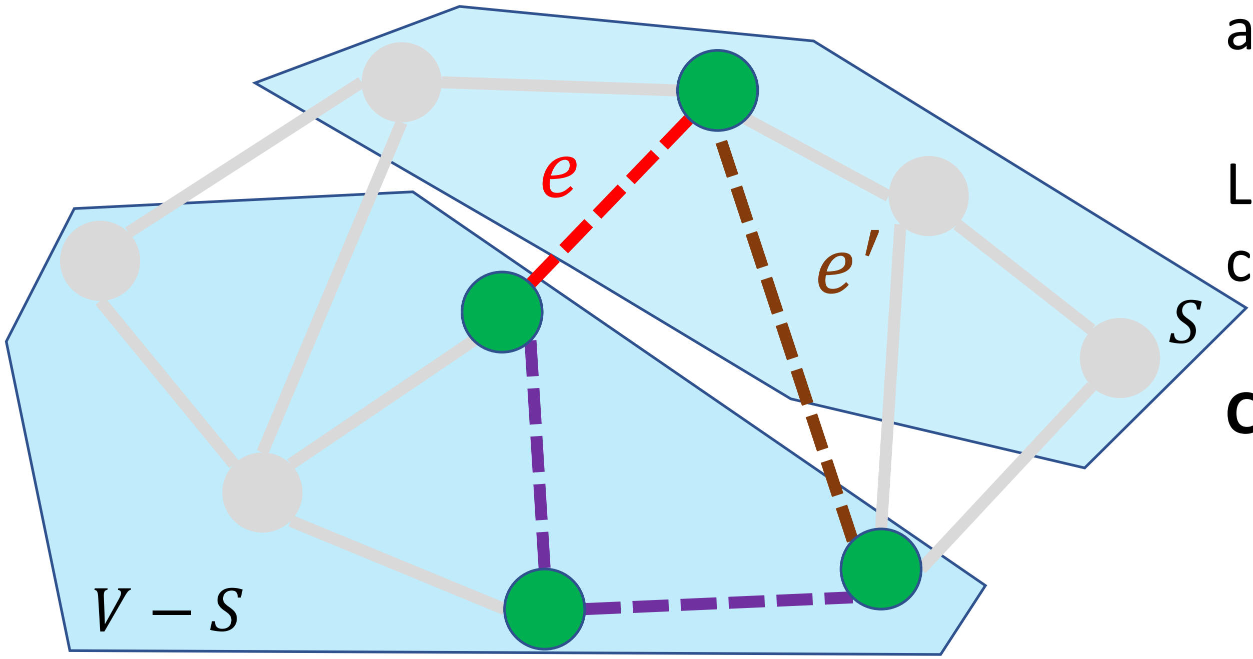
Case 2: $e \in T$

- Take any cut $(S, V - S)$ that e crosses
- There is another edge e' that crosses the cut (since we have a cycle)
- Exchange e with e'

Cycle Property of MSTs

Take any cycle in a graph $G = (V, E)$

Then, there exists some MST of G that does not contain the maximum-weight edge on that cycle



Proof. Take any cycle $(v_1, v_2, \dots, v_t, v_1)$ in G and take any MST T of G

L

c

C

- Resulting tree is still spanning (since S and $V - S$ still spanned and e' connects S with $V - S$)
- Cost of new tree is
  $\text{cost}(T) - w(e) + w(e') \le \text{cost}(T)$
  since $w(e') \le w(e)$
- Resulting tree must also be a MST

the c ce we have a cycle)

- Exchange e with e'