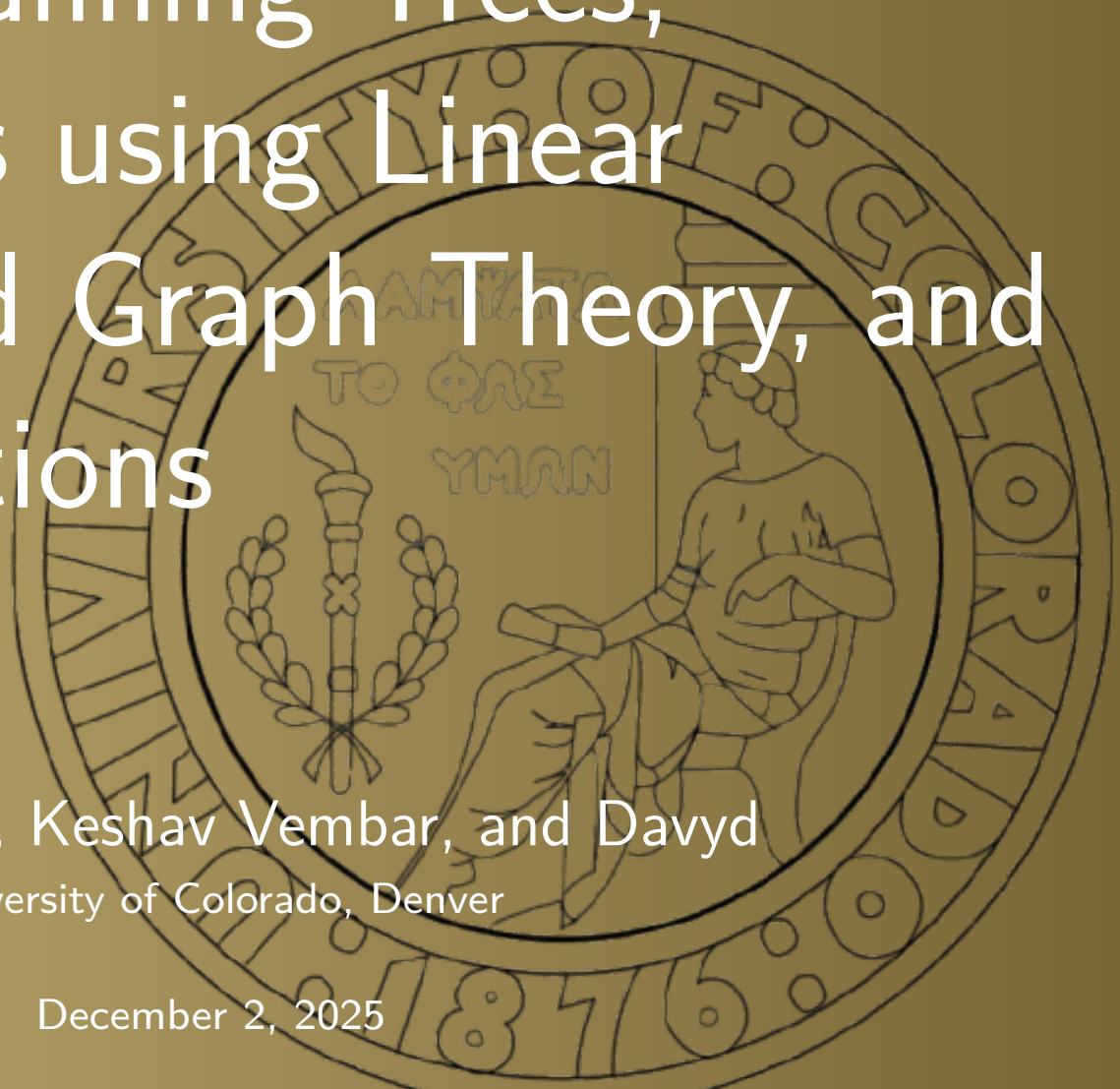
Computational Bonsai: On Minimum Spanning Trees, Constructions using Linear Programs and Graph Theory, and their Applications



Matt Eimers, Keshav Vembar, and Davyd
University of Colorado, Denver

December 2, 2025

Presentation Outline

Topics that will be covered

Matt's Section

- Minimum Spanning Tree (MST) as a integer program (IP),
- MST as a LP vs IP,
- MST example using IP methods,
- MST application in Christofides Algorithm for TSP.

Keshav's Section

- Prim's Algorithm,
- Kruskal's Algorithm,
- Analysis of Algorithms.

What is a Minimum Spanning Tree?

A Minimum Spanning Tree (MST) is defined as the following:

- A subset of the edges in a graph that connects all the nodes with no cycles and with the minimum possible total edge weight

It connects every node as cheaply as possible.

Integer Program (IP) for MST

Given a graph $G = (V, E)$ with costs c_{ij} on edges. Our goal is to connect all vertices with the minimum cost set of edges.

$$\begin{aligned}
\min \quad & \sum_{ij \in E} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{ij \in E} x_{ij} = n - 1, \\
& \sum_{ij \in E : i,j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subseteq V,\ S \neq \emptyset, \\
& x_{ij} \in \{0, 1\} \quad \forall ij \in E.
\end{aligned}$$

Integer Program (IP) for MST

Given a graph $G = (V, E)$ with costs c_{ij} on edges. Our goal is to connect all vertices with the minimum cost set of edges.

$$\begin{aligned}
\min \quad & \sum_{ij \in E} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{ij \in E} x_{ij} = n - 1, \\
& \sum_{ij \in E : i,j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subseteq V,\ S \neq \emptyset, \\
& x_{ij} \in \{0, 1\} \quad \forall ij \in E.
\end{aligned}$$

Resulting structure contains the following properties

- connected,
- cycle-free,
- containing exactly $n - 1$ edges.

Why the IP Model is Exponentially Large

Denver

- The subtour constraints must be written for *every* subset $S \subseteq V$:

$$\sum_{ij \in E : i,j \in S} x_{ij} \leq |S| - 1$$

- A set with $|V| = n$ vertices then it has 2^n subsets

Why the IP Model is Exponentially Large

- The subtour constraints must be written for *every* subset $S \subseteq V$:

$$\sum_{ij \in E : i,j \in S} x_{ij} \leq |S| - 1$$

- A set with $|V| = n$ vertices then it has 2^n subsets

- 10 nodes $\rightarrow 1024$ constraints

Why the IP Model is Exponentially Large

Denver

- The subtour constraints must be written for *every* subset $S \subseteq V$:

$$\sum_{ij \in E : i,j \in S} x_{ij} \leq |S| - 1$$

- A set with $|V| = n$ vertices then it has 2^n subsets
- 10 nodes $\rightarrow 1024$ constraints
- 20 nodes $\rightarrow$ over 1 million constraints

Why the IP Model is Exponentially Large

Denver

- The subtour constraints must be written for *every* subset $S \subseteq V$:

$$\sum_{ij \in E : i,j \in S} x_{ij} \leq |S| - 1$$

- A set with $|V| = n$ vertices then it has 2^n subsets
- 10 nodes $\rightarrow 1024$ constraints
- 20 nodes $\rightarrow$ over 1 million constraints
- Impossible to write out explicitly for large graphs.

IP vs. LP Program for MST

Denver

Integer Program (IP)

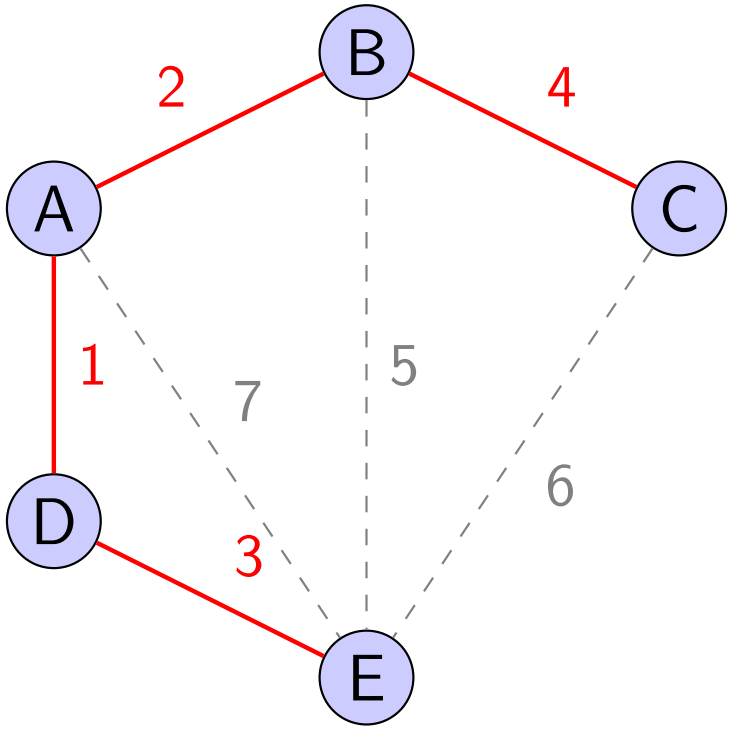
$$\begin{aligned}
\min \quad & \sum_{ij \in E} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{ij \in E} x_{ij} = n - 1, \\
& \sum_{ij \in E : i,j \in S} x_{ij} \leq |S| - 1 \quad \forall S, \\
& x_{ij} \in \{0, 1\}.
\end{aligned}$$

Linear Program (LP)

$$\begin{aligned}
\min \quad & \sum_{ij \in E} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{ij \in E} x_{ij} = n - 1, \\
& \sum_{ij \in E : i,j \in S} x_{ij} \leq |S| - 1 \quad \forall S, \\
& x_{ij} \geq 0.
\end{aligned}$$

- LP yields an *integral* optimal solution.
- Removing binary $x_{ij} \in \{0, 1\}$ does not change the optimum.
- LP solution will still yield x_{ij} to be binary.

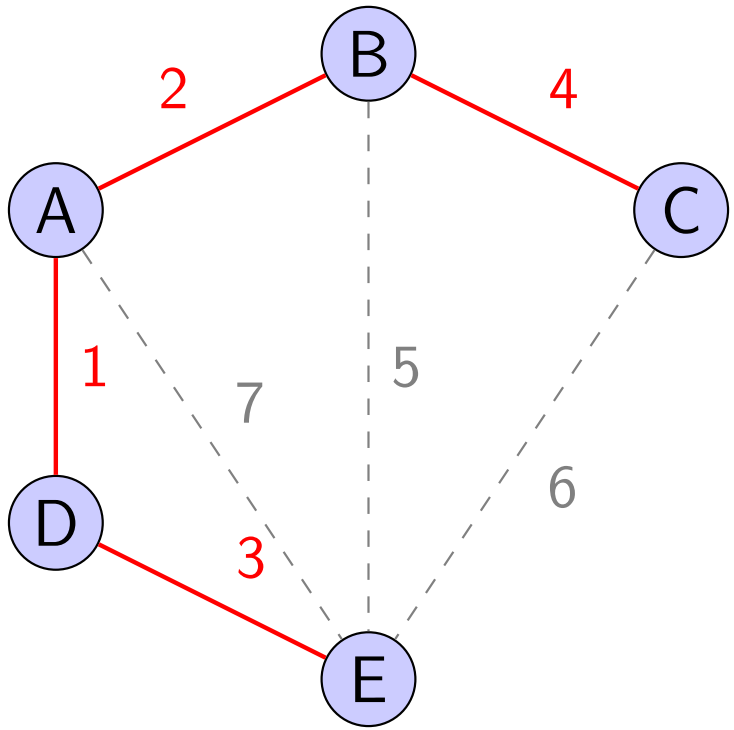
TikZ Model of MST (5 Nodes)



- Each edge ij corresponds to a binary variable x_{ij} in the IP model.
- The IP chooses edges that minimize the total cost.
- Constraint: exactly $n - 1 = 4$ edges must be selected.
- Subtour constraints ensure no subset of vertices forms a cycle.
- Solving the IP is shown with the red edges which form the MST.

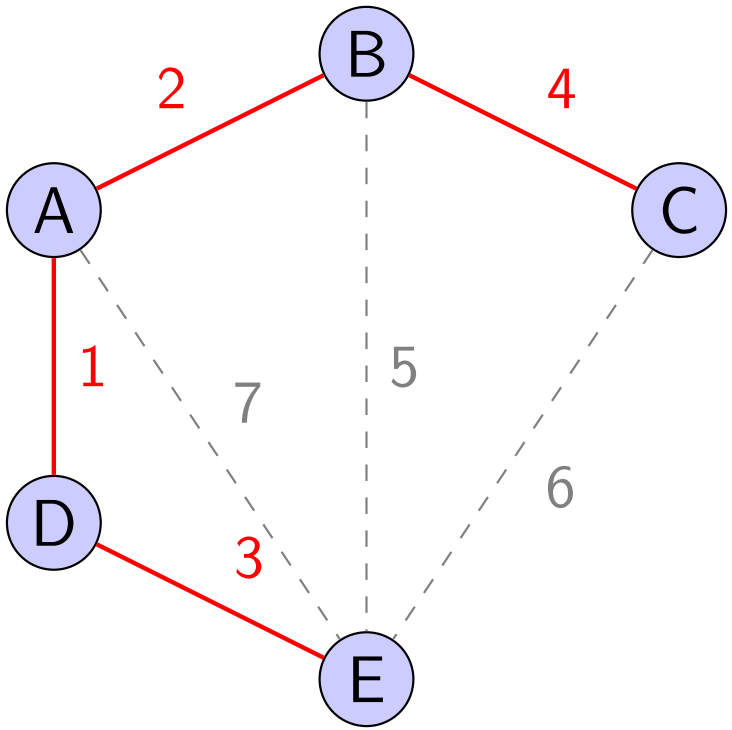
TikZ Model of MST (5 Nodes)

To verify our solution we now encode the same 5 node graph in Python and use the optimization library `PuLP` to compute its minimum spanning tree.



TikZ Model of MST (5 Nodes)

To verify our solution we now encode the same 5 node graph in Python and use the optimization library PuLP to compute its minimum spanning tree.




```
[1]  ✓ 0.0s

...  Optimal MST edges:
       A - B   cost = 2
       A - D   cost = 1
       D - E   cost = 3
       B - C   cost = 4

     Total MST cost: 10
```

Figure: PuLP Verification

PuLP selects the same edges and confirms the total MST cost is

$$1 + 2 + 3 + 4 = 10.$$

Automated Graph and MST Solver

- Generates a random graph with n nodes.
- Ensures the graph is sparse (not all nodes are connected).
- Assigns random edge weights.
- Solves the MST using an IP formulation.
- Designed to scale efficiently for large graphs.

Automated Graph and MST Solver

- Generates a random graph with n nodes.
- Ensures the graph is sparse (not all nodes are connected).
- Assigns random edge weights.
- Solves the MST using an IP formulation.
- Designed to scale efficiently for large graphs.

```
Optimal MST edges:
  X_{1} - X_{4}   cost = 6
  X_{3} - X_{6}   cost = 3
  X_{10} - X_{5}   cost = 5
  X_{11} - X_{5}   cost = 9
  X_{13} - X_{3}   cost = 5
  X_{11} - X_{15}   cost = 1
  X_{13} - X_{2}   cost = 5
  X_{6} - X_{9}   cost = 5
  X_{11} - X_{4}   cost = 1
  X_{12} - X_{15}   cost = 4
  X_{14} - X_{7}   cost = 2
  X_{4} - X_{6}   cost = 6
  X_{14} - X_{4}   cost = 3
  X_{15} - X_{8}   cost = 3

Total MST cost: 58
```

Figure: Optimal MST Solution using PuLP

Christofides' Algorithm Application

I generated a complete graph with my Python code and used its MST as the first step in Christofides' Algorithm to approximate a TSP tour.

```
Christofides Algorithm TSP tour:
X_{1} -> X_{58} -> X_{61} -> X_{81} -> X_{87} -> X_{17} -> X_{5} -> X_{10} -> X_{47} -> X_{74}
X_{28} -> X_{9} -> X_{16} -> X_{89} -> X_{35} -> X_{75} -> X_{99} -> X_{68} -> X_{29} -> X_{6}
X_{19} -> X_{77} -> X_{98} -> X_{42} -> X_{53} -> X_{11} -> X_{44} -> X_{27} -> X_{83} -> X_{93}
X_{2} -> X_{96} -> X_{60} -> X_{94} -> X_{56} -> X_{46} -> X_{48} -> X_{52} -> X_{43} -> X_{31}
X_{79} -> X_{73} -> X_{76} -> X_{24} -> X_{18} -> X_{59} -> X_{51} -> X_{49} -> X_{38} -> X_{45}
X_{67} -> X_{80} -> X_{7} -> X_{97} -> X_{3} -> X_{12} -> X_{50} -> X_{15} -> X_{92} -> X_{91}
X_{85} -> X_{86} -> X_{55} -> X_{69} -> X_{32} -> X_{57} -> X_{23} -> X_{66} -> X_{21} -> X_{100}
X_{78} -> X_{20} -> X_{37} -> X_{13} -> X_{8} -> X_{25} -> X_{33} -> X_{39} -> X_{40} -> X_{82}
X_{84} -> X_{90} -> X_{54} -> X_{88} -> X_{65} -> X_{62} -> X_{30} -> X_{63} -> X_{22} -> X_{14}
X_{36} -> X_{71} -> X_{95} -> X_{70} -> X_{64} -> X_{4} -> X_{34} -> X_{26} -> X_{41} -> X_{72}
X_{1}

Christofides tour cost: 1536
```

Figure: TSP using Christofides Algorithm on 100 Nodes

Christofides' Algorithm Application

I generated a complete graph with my Python code and used its MST as the first step in Christofides' Algorithm to approximate a TSP tour.

```
Christofides Algorithm TSP tour:
X_{1} -> X_{58} -> X_{61} -> X_{81} -> X_{87} -> X_{17} -> X_{5} -> X_{10} -> X_{47} -> X_{74}
X_{28} -> X_{9} -> X_{16} -> X_{89} -> X_{35} -> X_{75} -> X_{99} -> X_{68} -> X_{29} -> X_{6}
X_{19} -> X_{77} -> X_{98} -> X_{42} -> X_{53} -> X_{11} -> X_{44} -> X_{27} -> X_{83} -> X_{93}
X_{2} -> X_{96} -> X_{60} -> X_{94} -> X_{56} -> X_{46} -> X_{48} -> X_{52} -> X_{43} -> X_{31}
X_{79} -> X_{73} -> X_{76} -> X_{24} -> X_{18} -> X_{59} -> X_{51} -> X_{49} -> X_{38} -> X_{45}
X_{67} -> X_{80} -> X_{7} -> X_{97} -> X_{3} -> X_{12} -> X_{50} -> X_{15} -> X_{92} -> X_{91}
X_{85} -> X_{86} -> X_{55} -> X_{69} -> X_{32} -> X_{57} -> X_{23} -> X_{66} -> X_{21} -> X_{100}
X_{78} -> X_{20} -> X_{37} -> X_{13} -> X_{8} -> X_{25} -> X_{33} -> X_{39} -> X_{40} -> X_{82}
X_{84} -> X_{90} -> X_{54} -> X_{88} -> X_{65} -> X_{62} -> X_{30} -> X_{63} -> X_{22} -> X_{14}
X_{36} -> X_{71} -> X_{95} -> X_{70} -> X_{64} -> X_{4} -> X_{34} -> X_{26} -> X_{41} -> X_{72}
X_{1}

Christofides tour cost: 1536
```

Figure: TSP using Christofides Algorithm on 100 Nodes

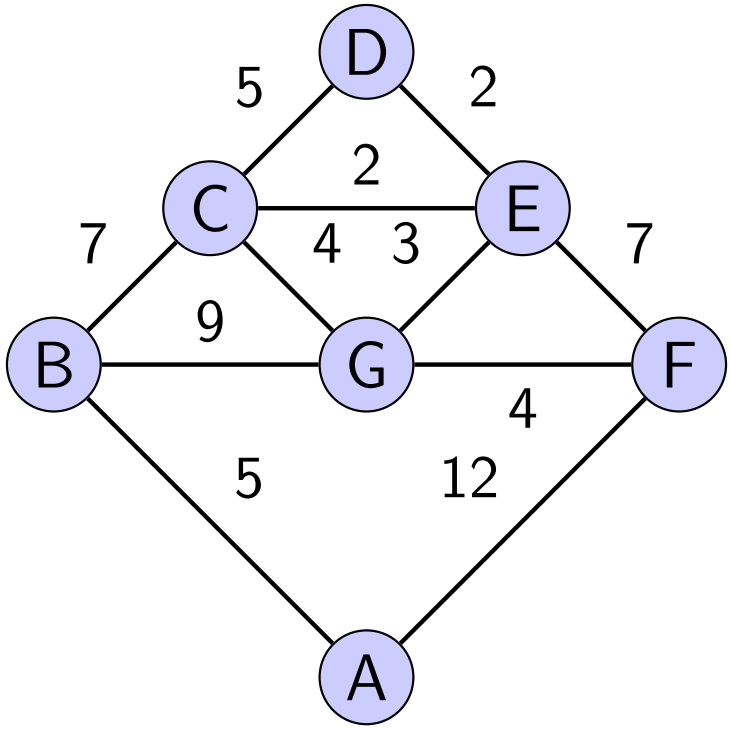
Christofides' Algorithm always gives a solution within

1.5 × the optimal TSP cost

Graph Theoretic Algorithms Overview

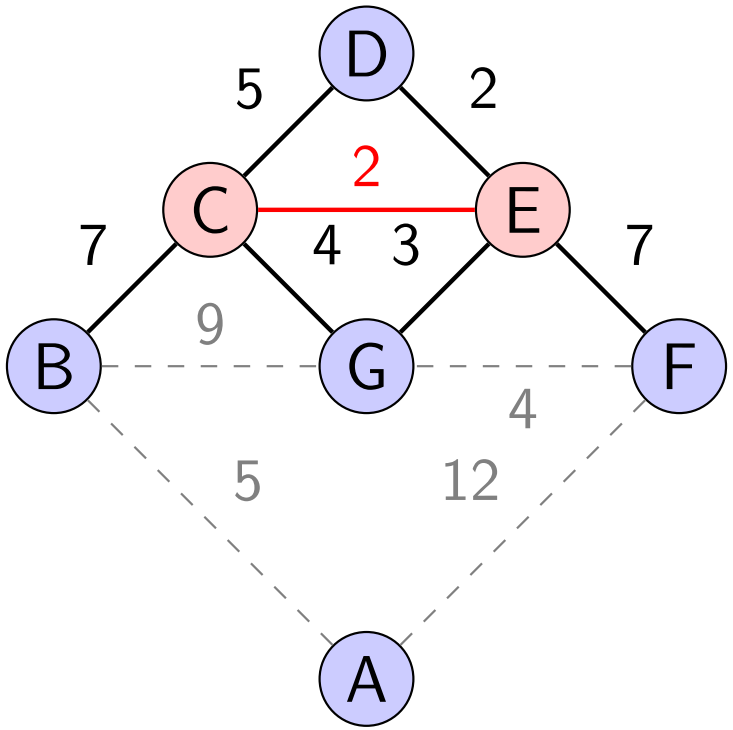
- There are many graph-theory-based algorithms used to compute the MST, separate from the IP formulation.
- Two of the most popular greedy algorithms: Prim's (1956) and Kruskal's (1957) Algorithm.
- Prim's Algorithm
  - Can run on graph implicitly
  - 'Growing' a tree
- Kruskal's Algorithm
  - Needs explicit knowledge of graph
  - 'Merging' many trees into one larger tree

Graph Example



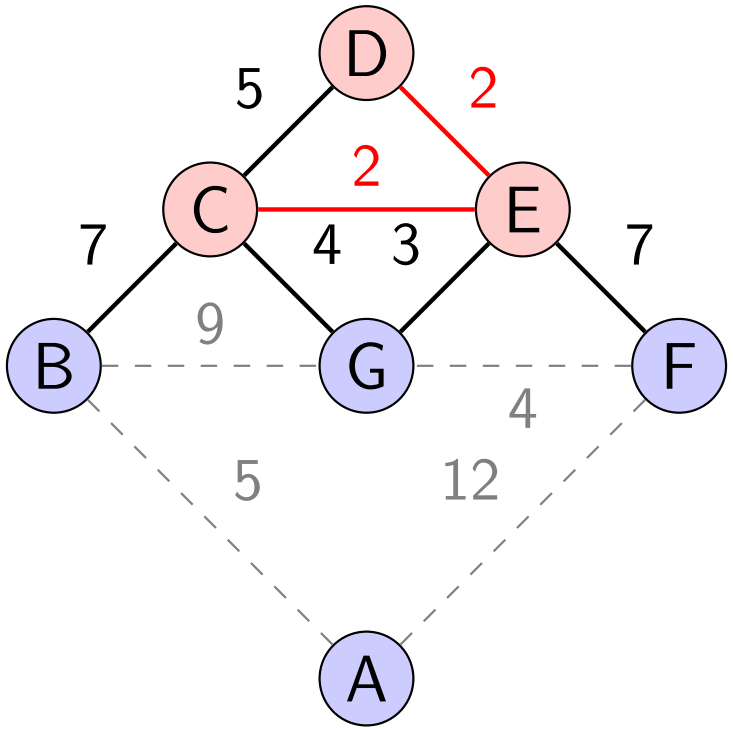
Prim's Algorithm Example

- Add the cheapest edge from the tree

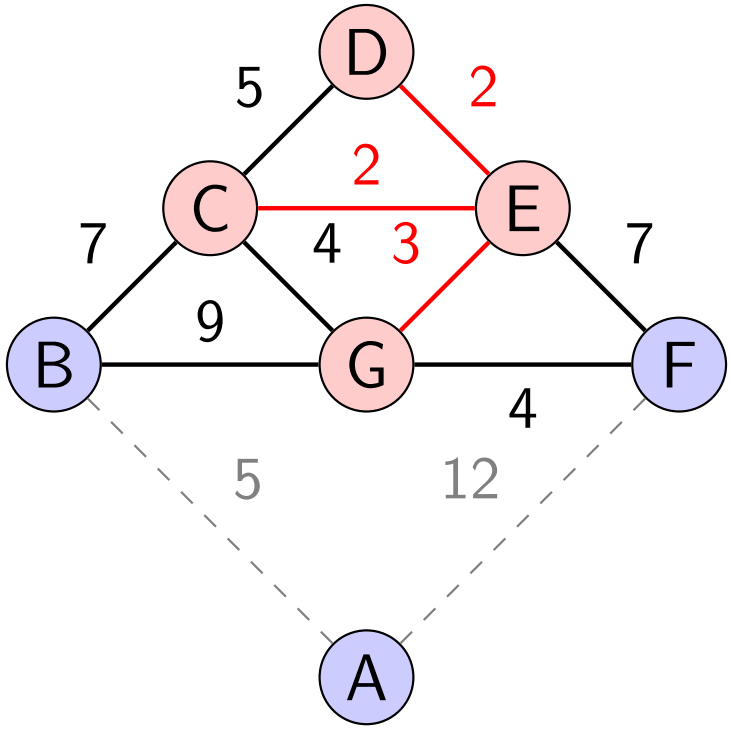


Prim's Algorithm Example

- For (V-1) iterations total, add the cheapest edge from the black edges.
- Edge must go from a connected node to a non-connected node.

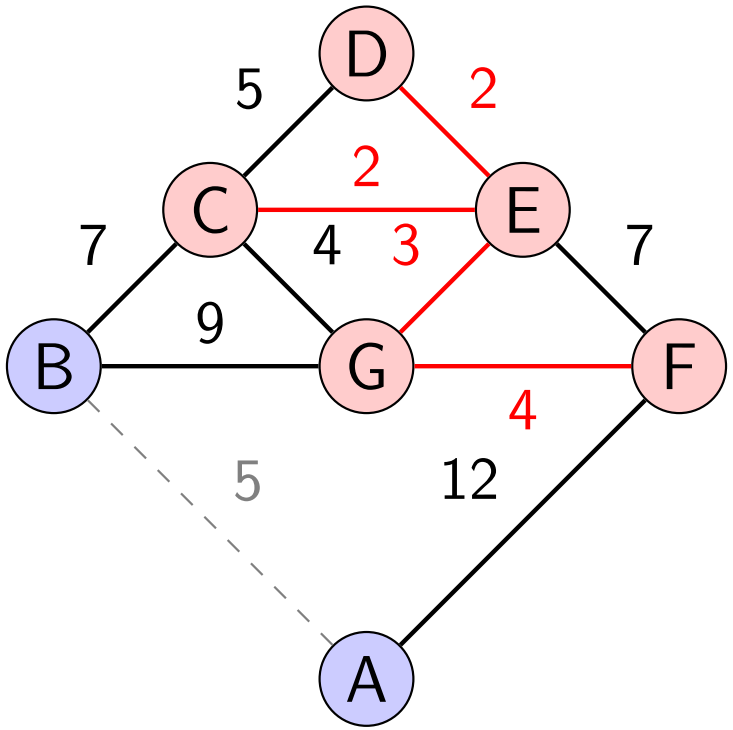


Prim's Algorithm Example



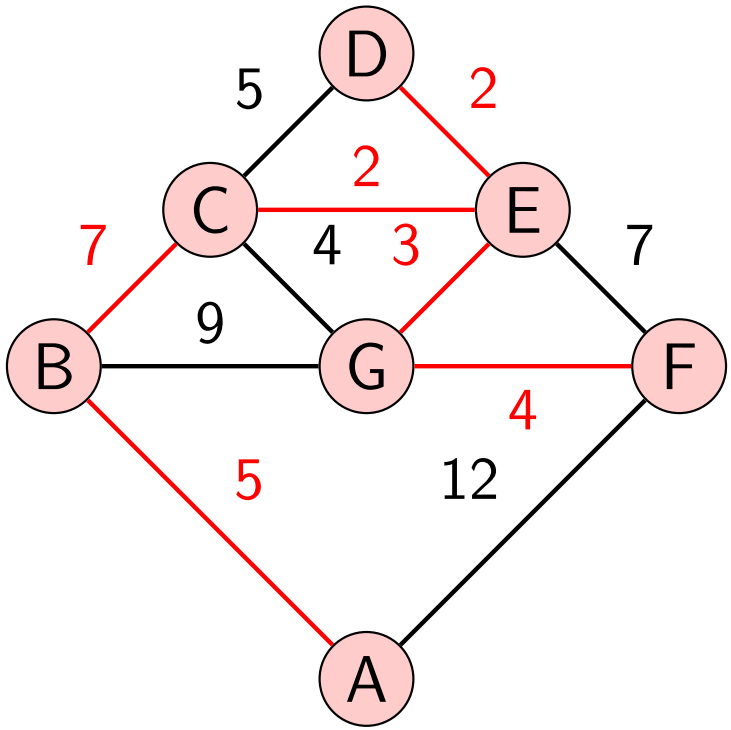
Prim's Algorithm Example

- Edge from (G) to (C) not added to prevent a cycle.



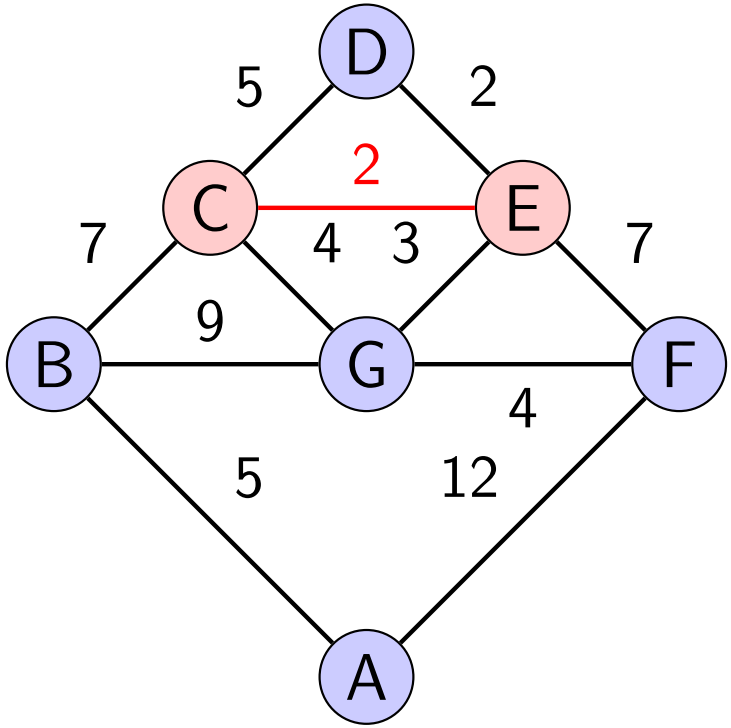
Prim's Algorithm Example

- End result of iterations.



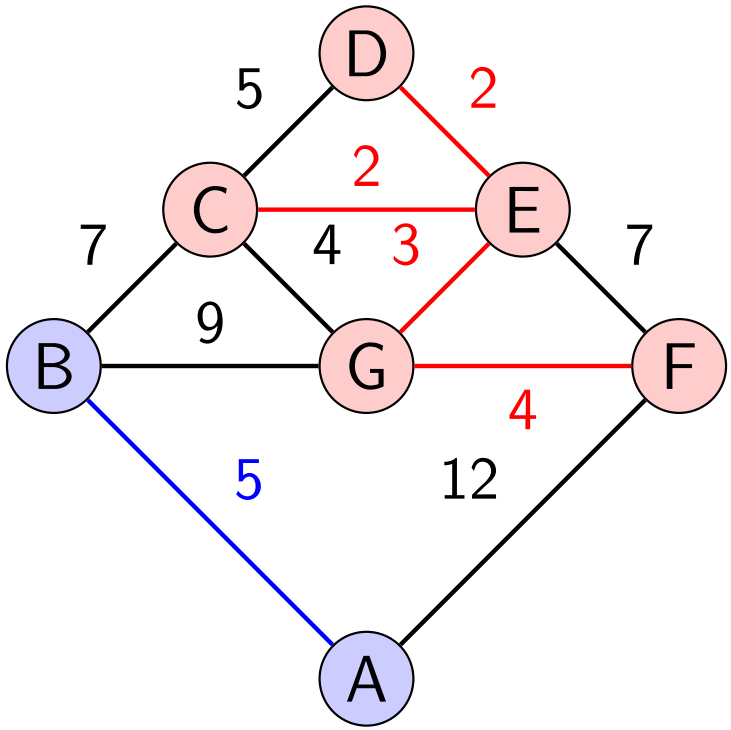
Kruskal's Algorithm Example

- Much like Prim's, you add the cheapest edge from the tree...



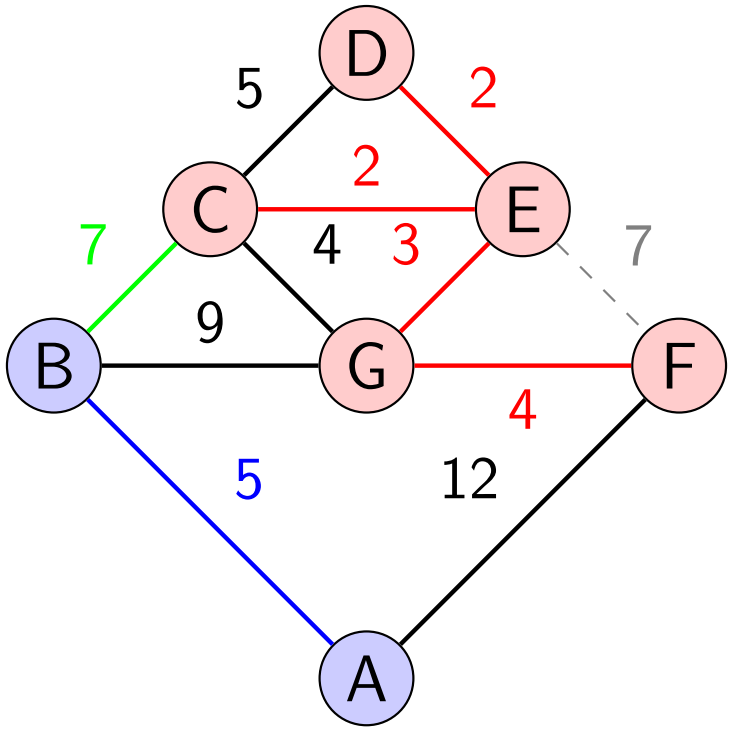
Kruskal's Algorithm Example

- ...but connectedness to the tree does not matter.
- Red and blue nodes are two 'connected components.'



Kruskal's Algorithm Example

- Adding the green edge merges the two sub-trees into an MST.
- Adding the dashed edge (same cost) creates a cycle.
- Separate connected components are formed (unlike in Prim's) and need to be accounted for.



Time Complexity Analysis

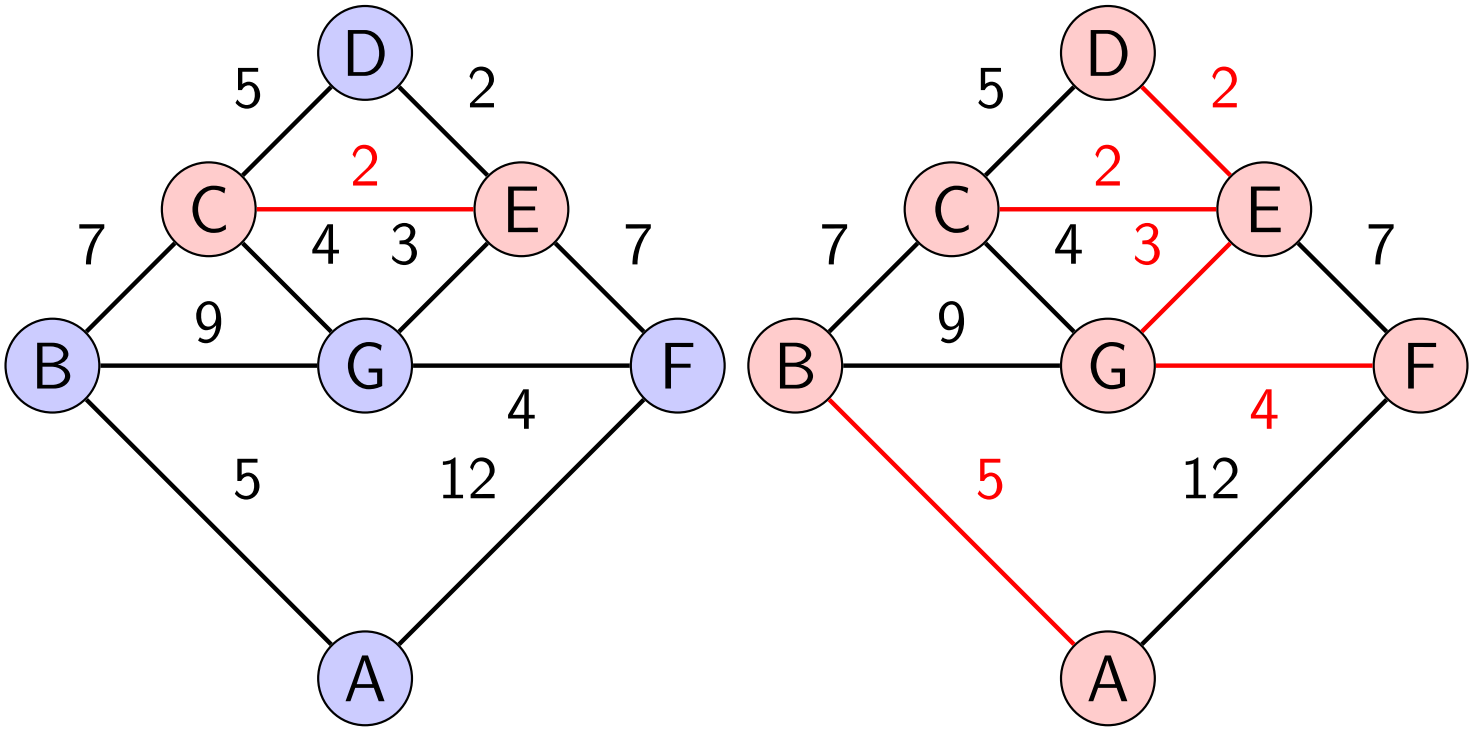
Both algorithms rely on sorting edges from smallest to largest, so sorting edge costs and keeping track of connected components are the two critical factors that limit the speed of the algorithms.

Algorithm	**Time Complexity**
Prim	$O(E + V \log V)$ (using fibonacci heap)
Kruskal	$O(E \log E)$ (sorting time)

Prim's algorithm is best used on dense graphs, while Kruskal's algorithm is best used on sparse graphs.

Iterative Building Example

- IP initializes from ST $\rightarrow$ MST.
- Graph Theoretic algorithms add edge-by-edge.
- Examples below are not MSTs, they the violate 'complete tree' constraint from original IP.



Insights

- MST is an 'easy' IP.
  - Allows for an LP relaxation.
  - Initialization of a vertex for the simplex method is easy.
  - Prim's and Kruskal's demonstrate another level of relaxation on constraints in the LP.
- A case where the IP can be very complicated space-wise despite an LP relaxation.