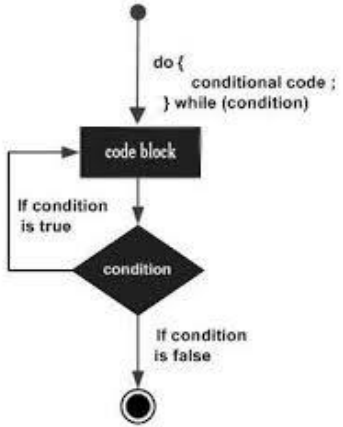
Lecture 13

Greedy Algorithms IV

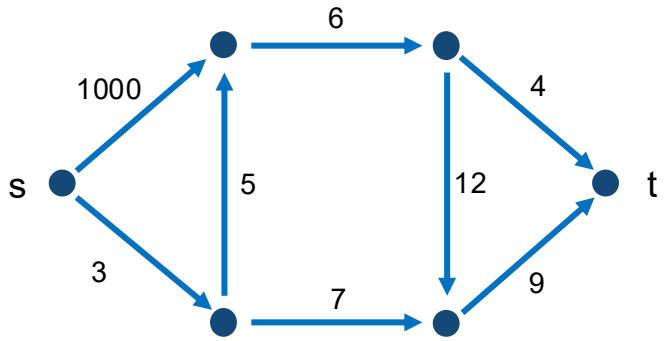


Announcements

- Homework 5 due Sunday night
- Group Meetings continue
  - New groups next week
- Individual Project 2 due next Sunday night

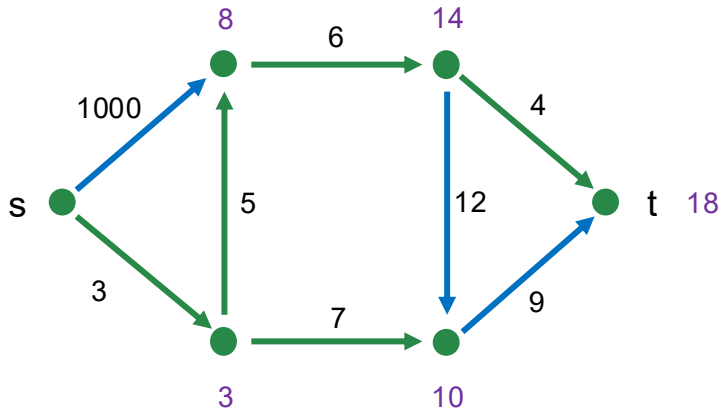
Shortest path problem

- Find shortest paths in a directed graph with edge weights (lengths)
- **Problem**: can we efficiently find $d(v)$ for all vertices $v \in V$?



Shortest path problem

- Find shortest paths in a directed graph with edge weights (lengths)
- **Problem**: can we efficiently find $d(v)$ for all vertices $v \in V$?

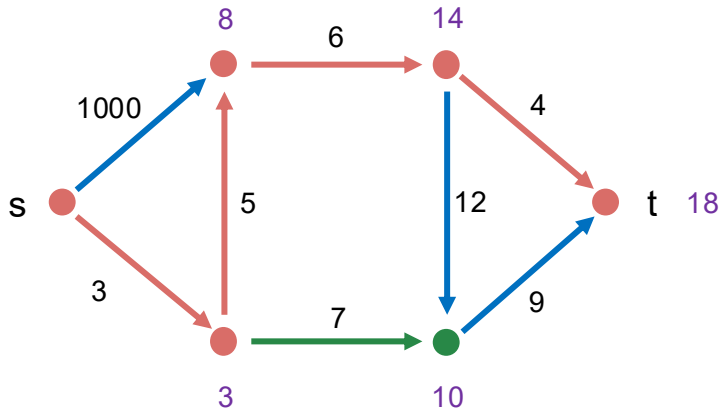


Dijkstra's Algorithm

- Idea: explore the "nearest" vertex from s
- Weighted version of BFS
- The shortest path from s to t has length 18
- Running time: $O\big((m+n)\log n\big)$

Shortest path problem

- Find shortest paths in a directed graph with edge weights (lengths)
- **Problem**: can we efficiently find $d(v)$ for all vertices $v \in V$?



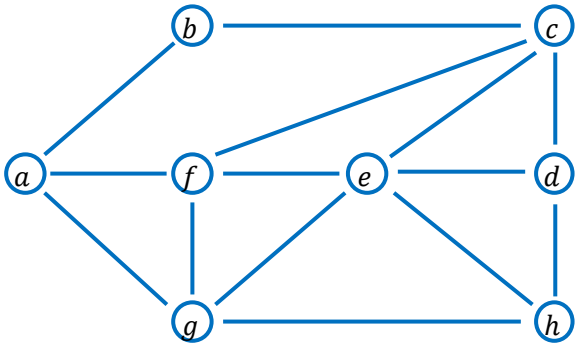
Dijkstra's Algorithm

- Idea: explore the "nearest" vertex from s
- Weighted version of BFS
- The shortest path from s to t has length 18
- Running time: $O\big((m+n)\log n\big)$

Cycles and Cuts

Recall some graph properties:

- $abcdef$ is a **path** from a to f
- $abcdefa$ is a **cycle**



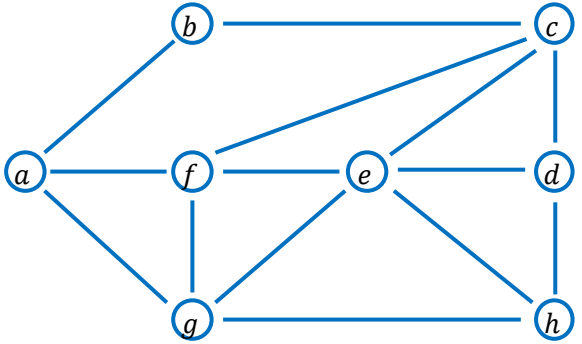
Cycles and Cuts

Recall some graph properties:

- $abcdef$ is a **path** from a to f
- $abcdefa$ is a **cycle**

Here are some new properties:

- A **cut** is a partition of the vertices into two non-empty subsets S and $V \backslash S$



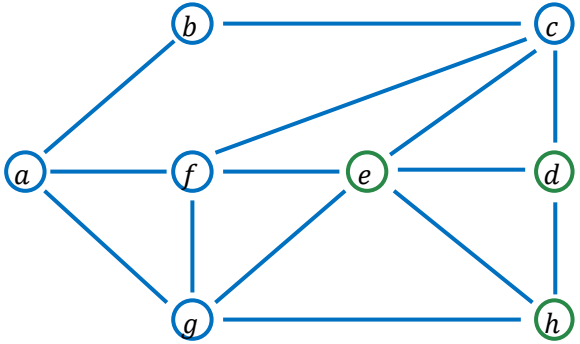
Cycles and Cuts

Recall some graph properties:

- $abcdef$ is a **path** from a to f
- $abcdefa$ is a **cycle**

Here are some new properties:

- A **cut** is a partition of the vertices into two non-empty subsets S and $V \backslash S$



$S = \{e, d, h\}$

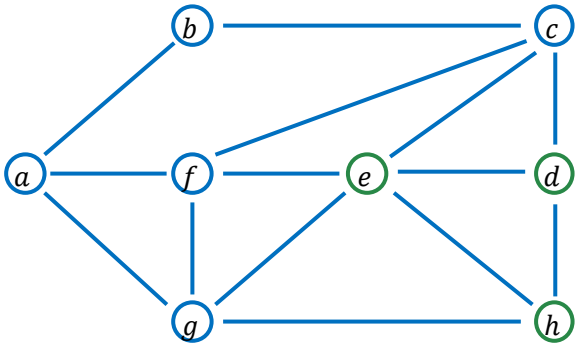
Cycles and Cuts

Recall some graph properties:

- $abcdef$ is a **path** from a to f
- $abcdefa$ is a **cycle**

Here are some new properties:

- A **cut** is a partition of the vertices into two non-empty subsets S and $V \backslash S$
- A **cutset** of a cut S is the set of edges with exactly one end in S



$S = \{e, d, h\}$

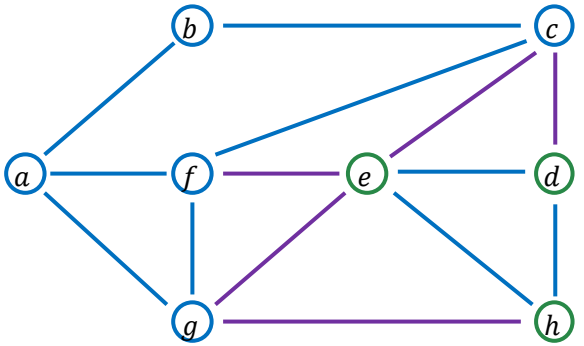
Cycles and Cuts

Recall some graph properties:

- $abcdef$ is a **path** from a to f
- $abcdefa$ is a **cycle**

Here are some new properties:

- A **cut** is a partition of the vertices into two non-empty subsets S and $V \backslash S$
- A **cutset** of a cut S is the set of edges with exactly one end in S
- Deleting a cutset D of edges disconnects the subsets of a cut



$$S = \{e, d, h\}$$

$$D = \{(e, g), (e, f), (c, e), (c, d), (g, h)\}$$

Exercise I

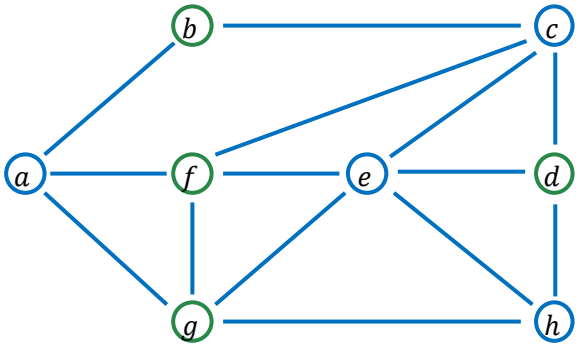
Consider the cut $S = \{b, f, g, d\}$. Which edge is in the cutset of S?

a. (b, f)

b. (g, f)

c. (f, e)

d. (c, e)



Exercise I

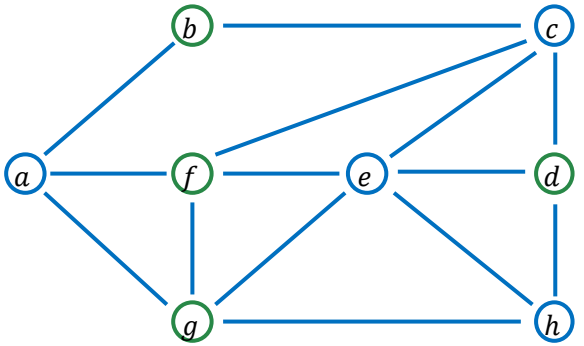
Consider the cut $S = \{b, f, g, d\}$. Which edge is in the cutset of S?

a. (b, f)

b. (g, f)

c. (f, e)

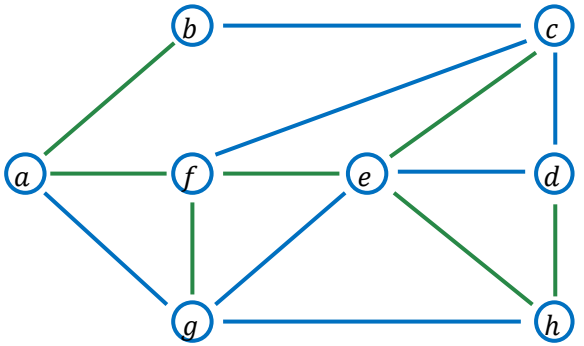
d. (c, e)



Spanning Trees

Let $H = (V, T)$ be a subgraph of an undirected graph $G = (V, E)$. Then H is a **spanning tree** of G if H is both acyclic and connected.

- In other words: a spanning tree is a subgraph on all the vertices that is a tree
- Where have we seen spanning trees before?



Exercise II

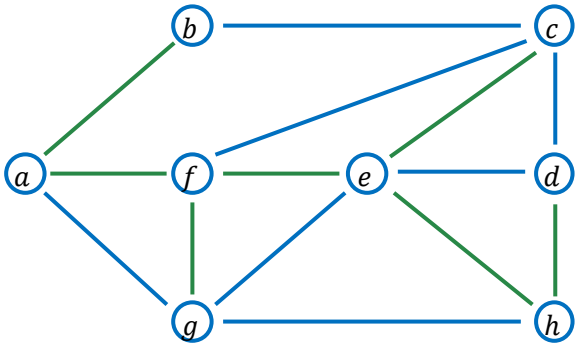
Which of the following properties are true for all spanning trees H?

a. H contain exactly $|V| - 1$ edges

b. Removing any edge from H disconnects it

c. Adding any edge to H creates a cycle

d. All of the above



Exercise II

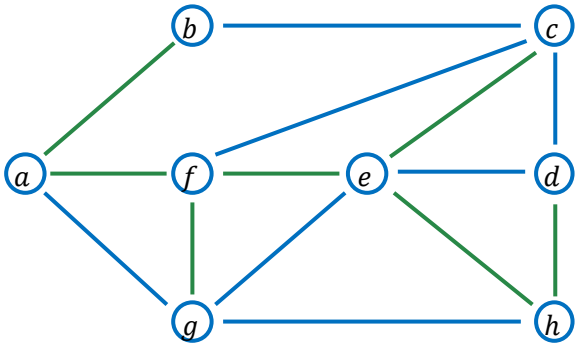
Which of the following properties are true for all spanning trees H?

a. H contain exactly $|V| - 1$ edges

b. Removing any edge from H disconnects it

c. Adding any edge to H creates a cycle

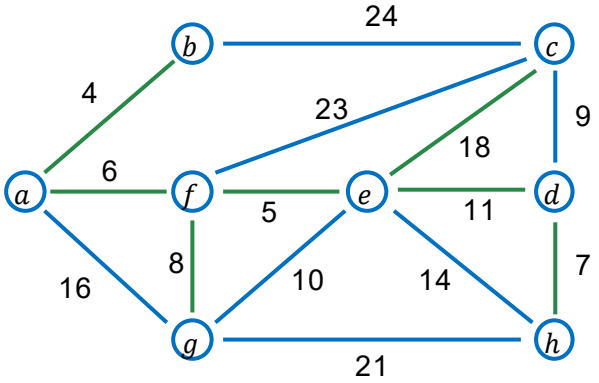
d. All of the above



Minimum spanning tree (MST)

Given a connected, undirected graph $G = (V, E)$ with edge weights c_e, a **minimum spanning tree** (V, T) is a spanning tree of G such that the sum of the edge costs in T is minimized

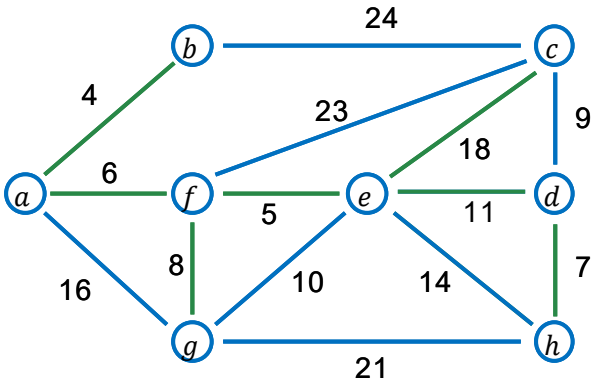
- In other words: an MST is a spanning tree with the lowest edge costs



Minimum spanning tree (MST)

Given a connected, undirected graph $G = (V, E)$ with edge weights c_e, a **minimum spanning tree** (V, T) is a spanning tree of G such that the sum of the edge costs in T is minimized

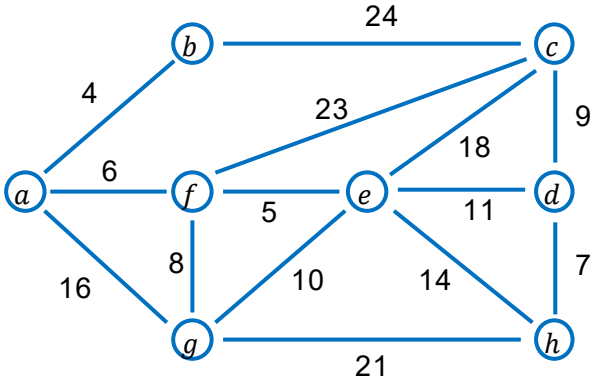
- In other words: an MST is a spanning tree with the lowest edge costs
- Finding an MST is a fundamental problem in CS with diverse applications



Cut Property

Th: Let D be the cutset of cut S in graph G. Then, the minimum weight edge e in D belongs to every MST of G.

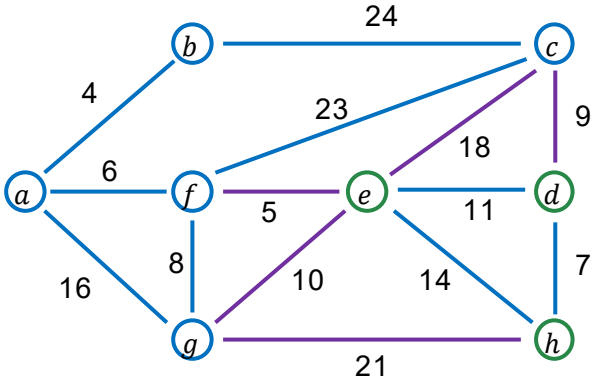
- e is the *cheapest* or *lightest* edge in the cut
- It is *safe* to add e to an MST



Cut Property

Th: Let D be the cutset of cut S in graph G. Then, the minimum weight edge e in D belongs to every MST of G.

- e is the *cheapest* or *lightest* edge in the cut
- It is *safe* to add e to an MST
- (f, e) belongs to every MST of G



$$S = \{e, d, h\}$$

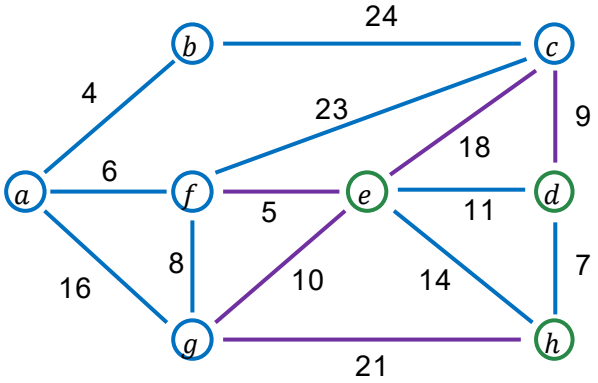
$$D = \{(e, g), (e, f), (c, e), (c, d), (g, h)\}$$

Cut Property

Th: Let D be the cutset of cut S in graph G. Then, the minimum weight edge e in D belongs to every MST of G.

- e is the *cheapest* or *lightest* edge in the cut
- It is *safe* to add e to an MST
- (f, e) belongs to every MST of G

- Prove using contradiction by supposing e is the minimum weight edge in the cutset for cut S but e is not in an MST.
- Good idea for Project II

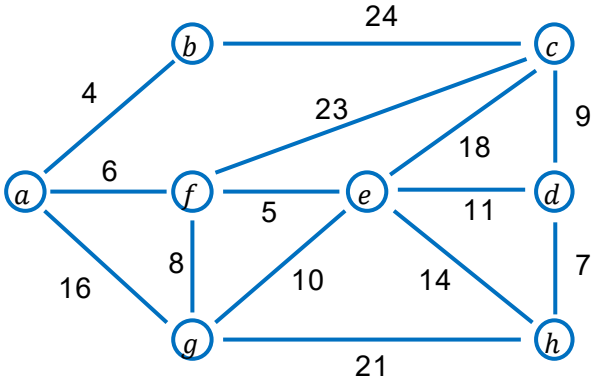


$$S = \{e, d, h\}$$

$$D = \{(e, g), (e, f), (c, e), (c, d), (g, h)\}$$

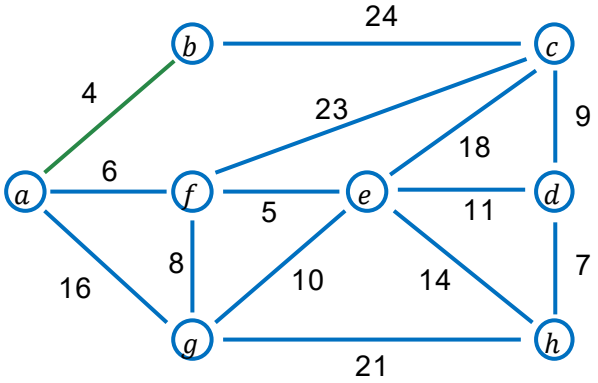
Kruskal's Algorithm

Idea: add edges in order of weight so long as it does not create a cycle



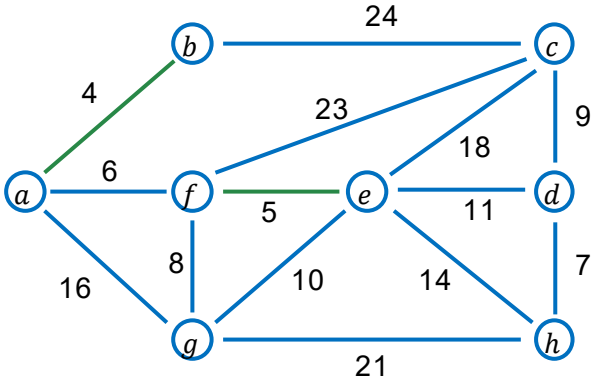
Kruskal's Algorithm

Idea: add edges in order of weight so long as it does not create a cycle



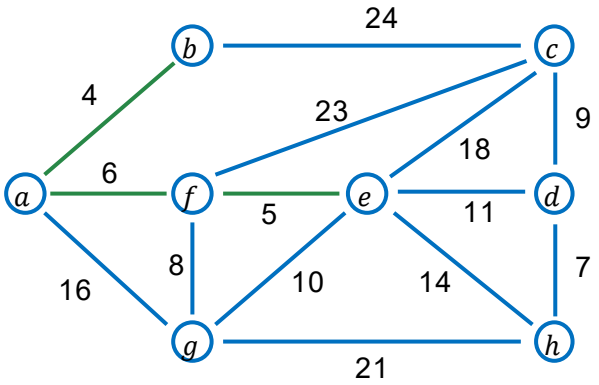
Kruskal's Algorithm

Idea: add edges in order of weight so long as it does not create a cycle



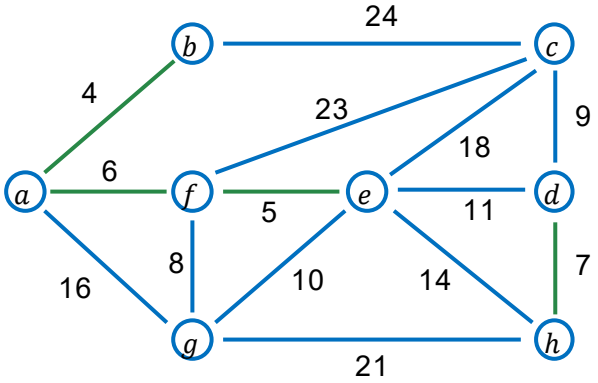
Kruskal's Algorithm

Idea: add edges in order of weight so long as it does not create a cycle



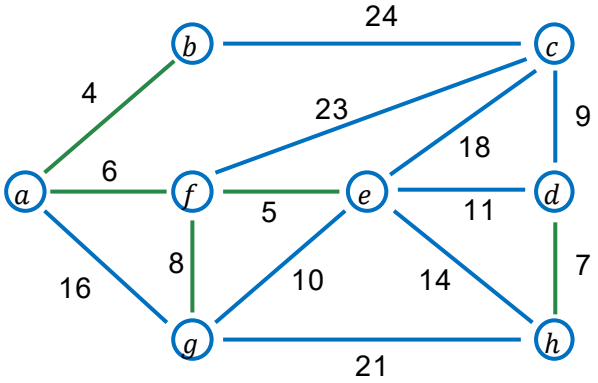
Kruskal's Algorithm

Idea: add edges in order of weight so long as it does not create a cycle



Kruskal's Algorithm

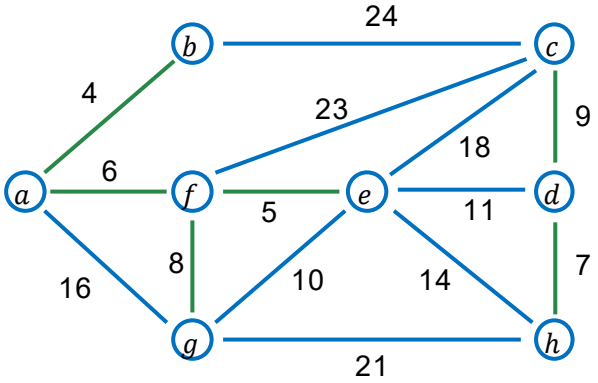
Idea: add edges in order of weight so long as it does not create a cycle



Kruskal's Algorithm

Idea: add edges in order of weight so long as it does not create a cycle

How does this use the cut property?



Kruskal's Algorithm

Idea: add edges in order of weight so long as it does not create a cycle

Assume edges are numbered $e = 1, \ldots, m$

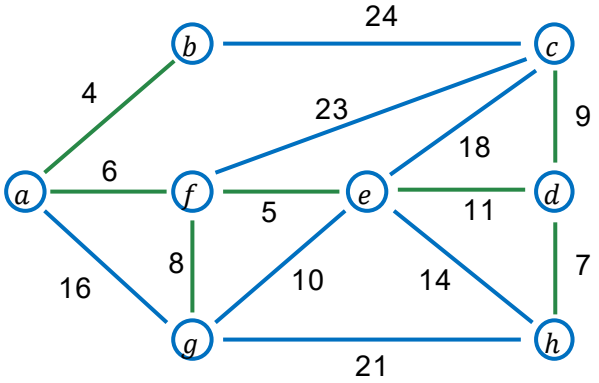
Sort edges by weight so $c_1 \leq \cdots \leq c_m$

Initialize $T = \{\ \}$

for $e = 1$ to m **do**

if adding e to T does not form a cycle **then**

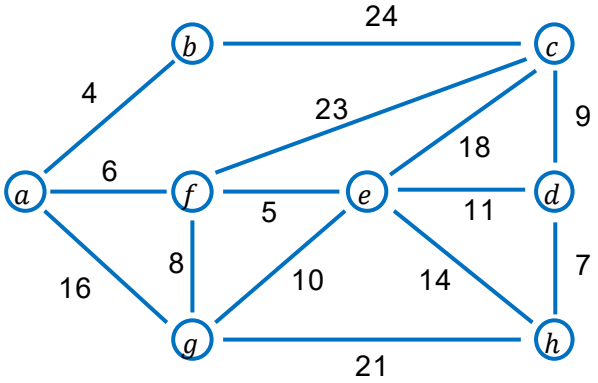
$T = T \cup \{e\}$



Prim's Algorithm

Idea: grow a tree as a single connected component from a vertex s

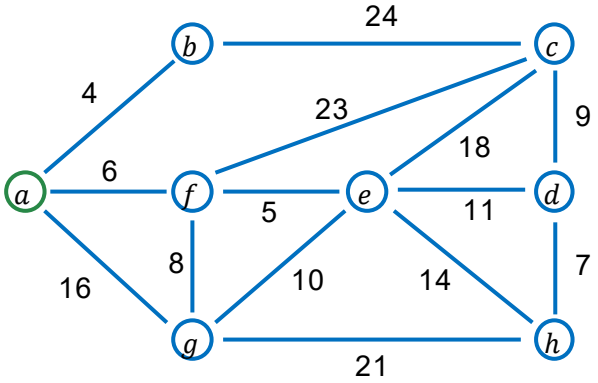
- Let S be the component containing s
- Add the cheapest edge from S to $V \backslash S$



Prim's Algorithm

Idea: grow a tree as a single connected component from a vertex s

- Let S be the component containing s
- Add the cheapest edge from S to $V \backslash S$

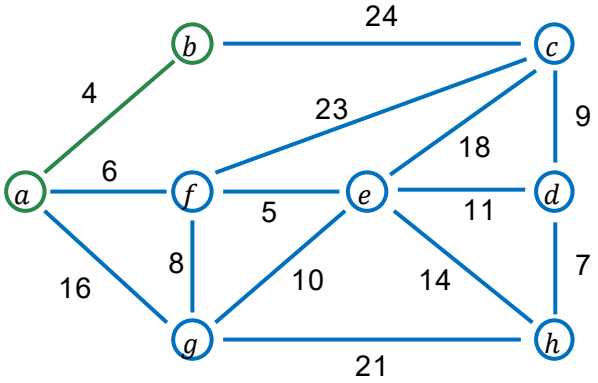


$S = \{a\}$

Prim's Algorithm

Idea: grow a tree as a single connected component from a vertex s

- Let S be the component containing s
- Add the cheapest edge from S to $V \backslash S$

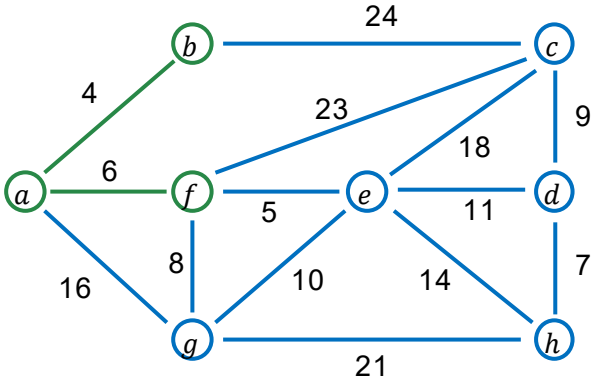


$S = \{a, b\}$

Prim's Algorithm

Idea: grow a tree as a single connected component from a vertex s

- Let S be the component containing s
- Add the cheapest edge from S to $V \backslash S$

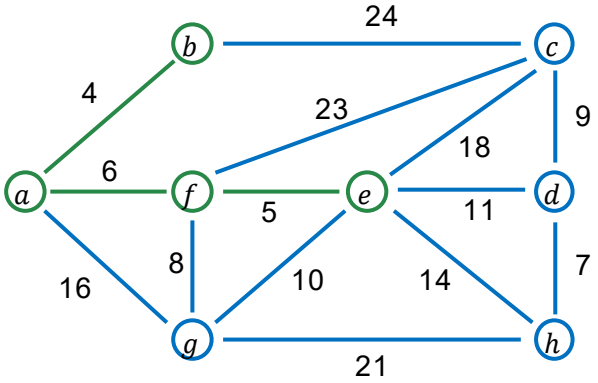


$S = \{a, b, f\}$

Prim's Algorithm

Idea: grow a tree as a single connected component from a vertex s

- Let S be the component containing s
- Add the cheapest edge from S to $V \backslash S$

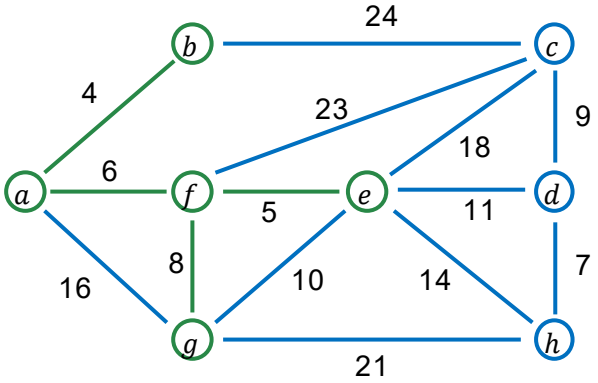


$S = \{a, b, f, e\}$

Prim's Algorithm

Idea: grow a tree as a single connected component from a vertex s

- Let S be the component containing s
- Add the cheapest edge from S to $V \backslash S$

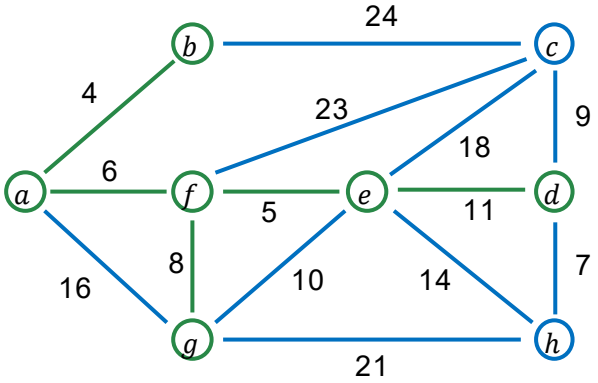


$S = \{a, b, f, e, g\}$

Prim's Algorithm

Idea: grow a tree as a single connected component from a vertex s

- Let S be the component containing s
- Add the cheapest edge from S to $V \backslash S$

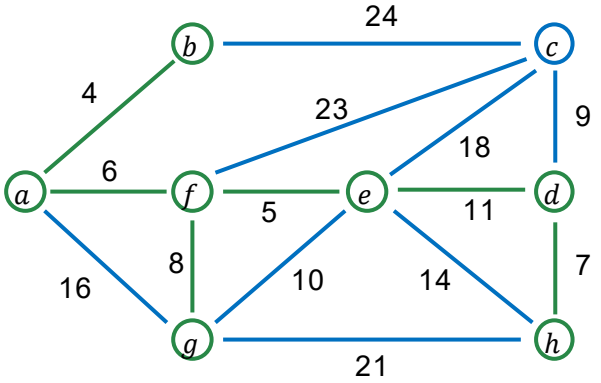


$S = \{a, b, f, e, g, d\}$

Prim's Algorithm

Idea: grow a tree as a single connected component from a vertex s

- Let S be the component containing s
- Add the cheapest edge from S to $V \backslash S$

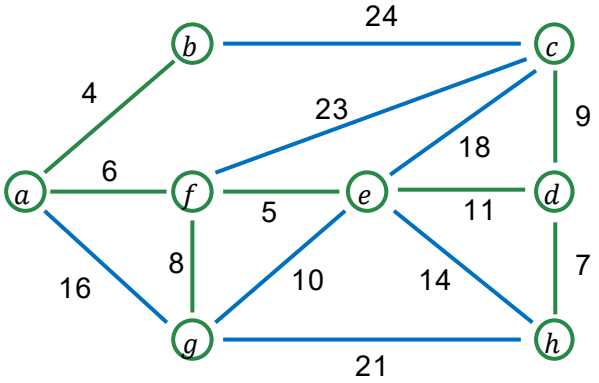


$S = \{a, b, f, e, g, d, h\}$

Prim's Algorithm

Idea: grow a tree as a single connected component from a vertex s

- Let S be the component containing s
- Add the cheapest edge from S to $V \backslash S$



$S = \{a, b, f, e, g, d, h, c\}$

Next Time

- Dynamic Programming