CSE373: Data Structures & Algorithms

Lecture 18: Minimum Spanning Trees

Lauren Milne

Summer 2015

Announcements

- Homework 3 graded and comments out
- Homework 5 is out
  - Due next Wednesday
  - Can be done with partners
    - List partner on files

So far

- We've figured out how to
  - Find the shortest paths between a vertex and all other vertices
    - Breadth First Search (unweighted graph)
    - Dijsktra (weighted graph)
  - Find a spanning tree on an unweighted graph
    - Graph Traversal (we did DFS)
    - Pick random edges and see if it connects the graph (use Union Find)
- Next up
  - Find a minimum spanning tree on a weighted graph
    - Prim's algorithm
    - Kruskal's algorithm

Minimum Spanning Trees

The minimum-spanning-tree problem

– Given a weighted undirected graph, compute a spanning tree of minimum weight

Given an undirected graph **G**=(**V**,**E**), find a graph **G'=(V, E')** such that:

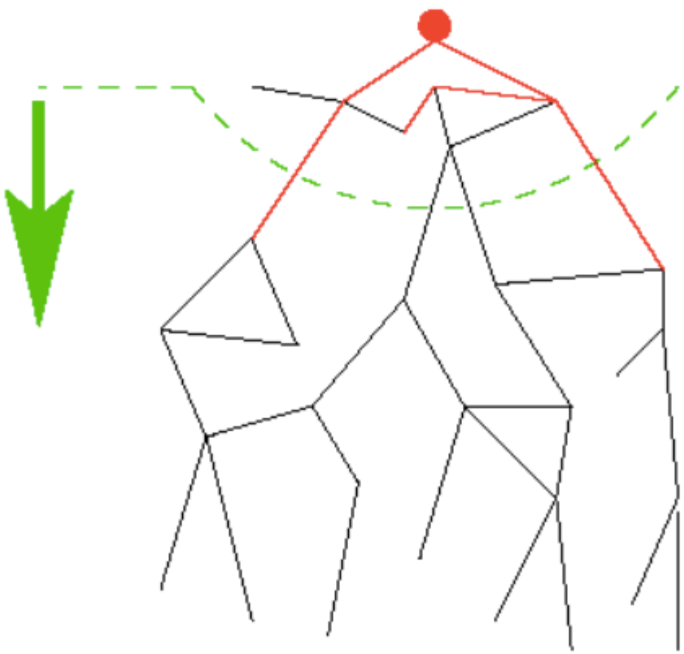
– E' is a subset of E

– |E'| = |V| - 1

– G' is connected

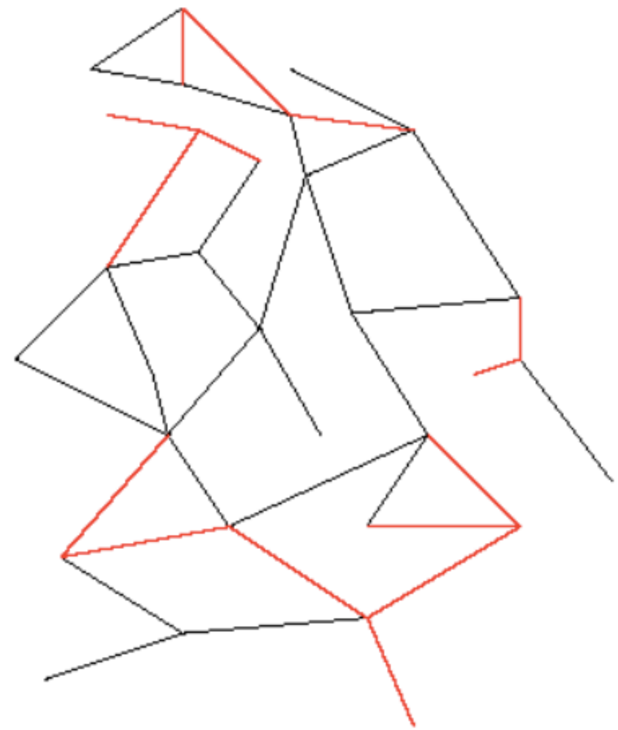
G' is a minimum spanning tree.

Two different approaches



Prim's Algorithm

Almost identical to Dijkstra's



Kruskals's Algorithm

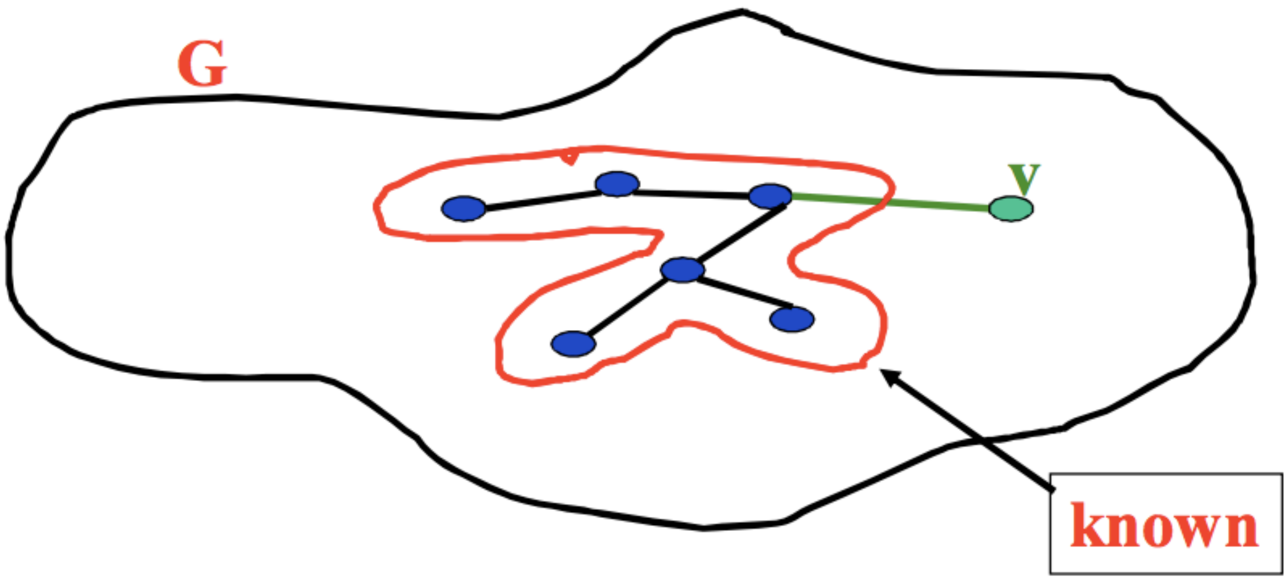
Completely different!

Prim's Algorithm Idea

Idea: Grow a tree by picking a vertex from the unknown set that has the smallest cost. Here cost = cost of the edge that connects that vertex to the known set. *Pick the vertex with the smallest cost that connects "known" to "unknown."*

A *node-based* greedy algorithm

Builds MST by greedily adding nodes



Prim’s vs. Dijkstra’s

Recall:

Dijkstra picked the unknown vertex with smallest cost where
cost = distance to the source.

Prim’s pick the unknown vertex with smallest cost where
cost = distance from this vertex to the known set
(in other words, the cost of the smallest edge connecting this vertex to the known set)

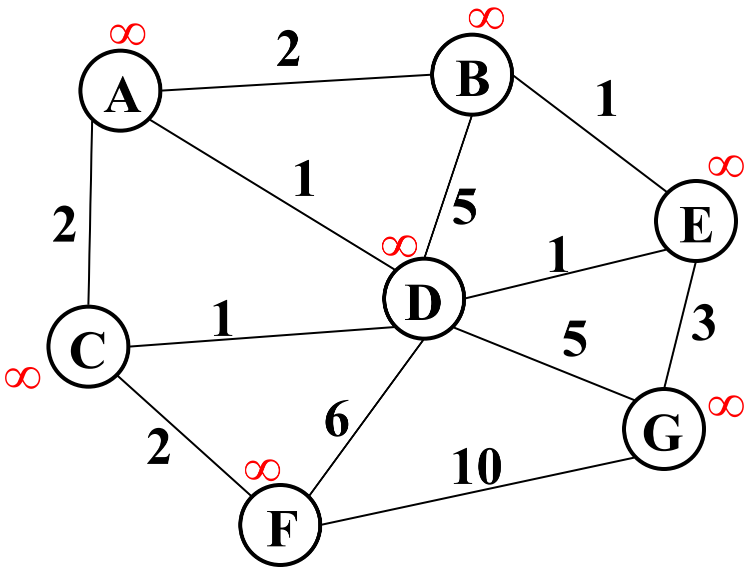
Otherwise identical ☺

Prim's Algorithm

1. For each node **`v`**, set **`v.cost =`** ∞ and **`v.known = false`**
2. Choose any node **`v`**
   a) Mark **`v`** as known
   b) For each edge **`(v,u)`** with weight **`w`**, set **`u.cost=w`** and **`u.prev=v`**
3. While there are unknown nodes in the graph
   a) Select the unknown node **`v`** with lowest cost
   b) Mark **`v`** as known and add **`(v, v.prev)`** to output
   c) For each edge **`(v,u)`** with weight **`w`**,

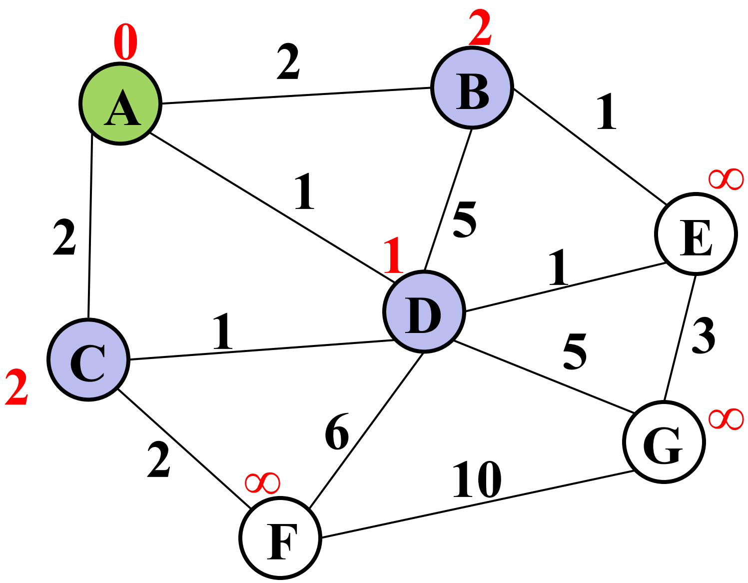
```
if(w < u.cost) {
  u.cost = w;
  u.prev = v;
}
```

Prim's Example



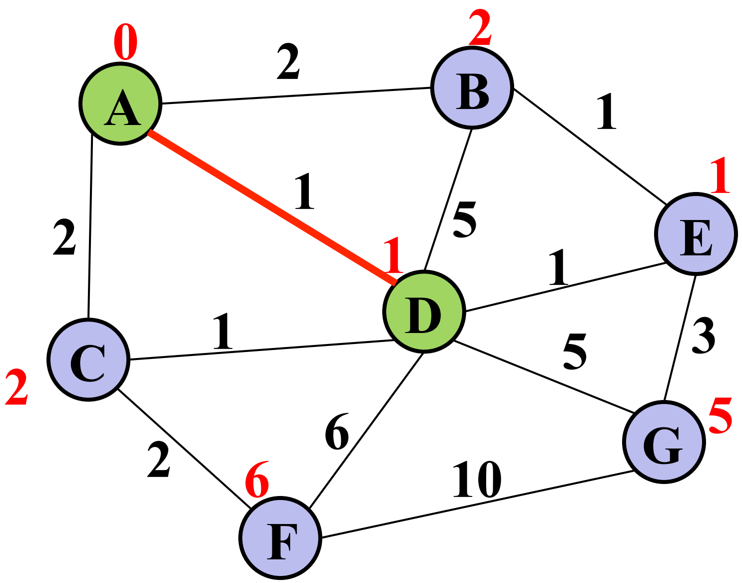
vertex	known?	cost	prev
A		??	
B		??	
C		??	
D		??	
E		??	
F		??	
G		??	

Prim's Example



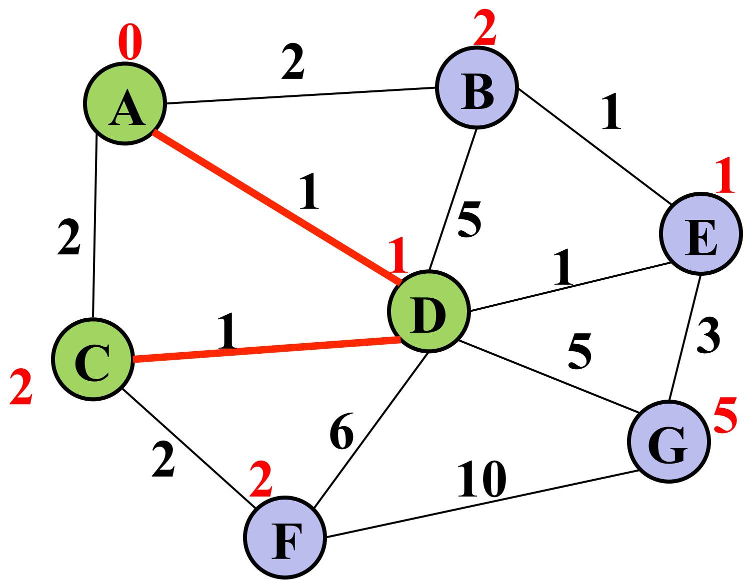
vertex	known?	cost	prev
A	Y	0	
B		2	A
C		2	A
D		1	A
E		??	
F		??	
G		??	

Prim's Example



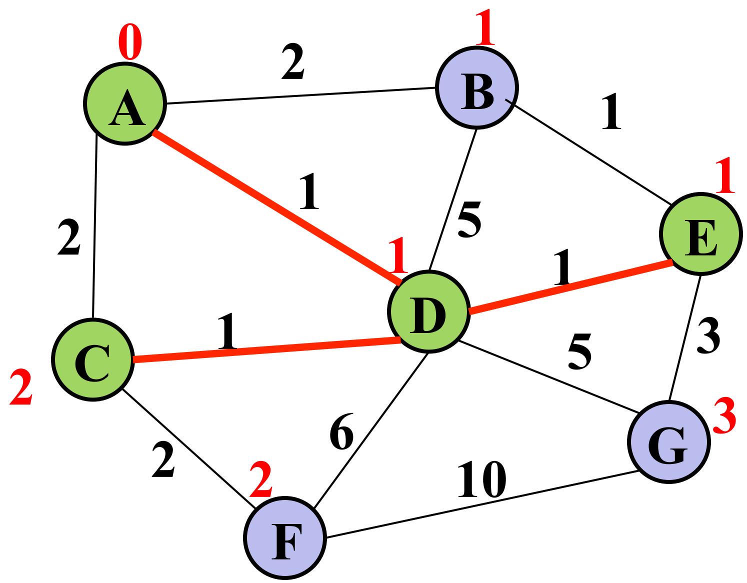
vertex	known?	cost	prev
A	Y	0	
B		2	A
C		1	D
D	Y	1	A
E		1	D
F		6	D
G		5	D

Prim's Example



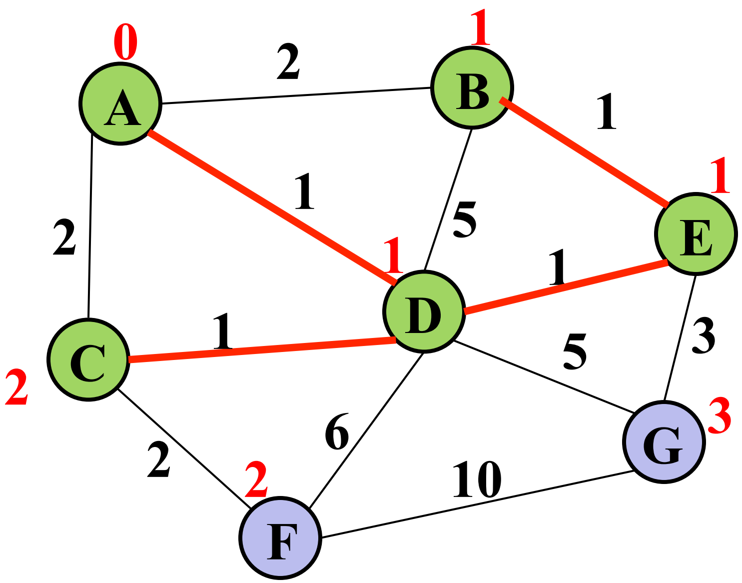
vertex	known?	cost	prev
A	Y	0	
B		2	A
C	Y	1	D
D	Y	1	A
E		1	D
F		2	C
G		5	D

Prim's Example



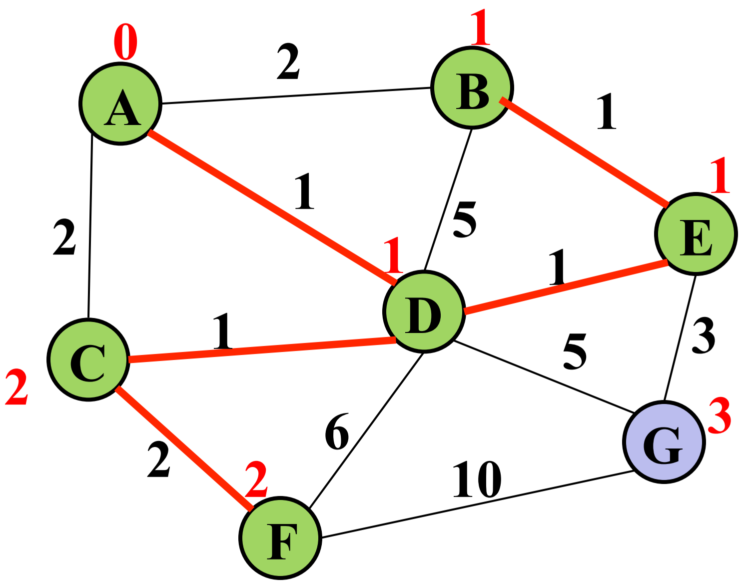
vertex	known?	cost	prev
A	Y	0	
B		1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F		2	C
G		3	E

Prim's Example



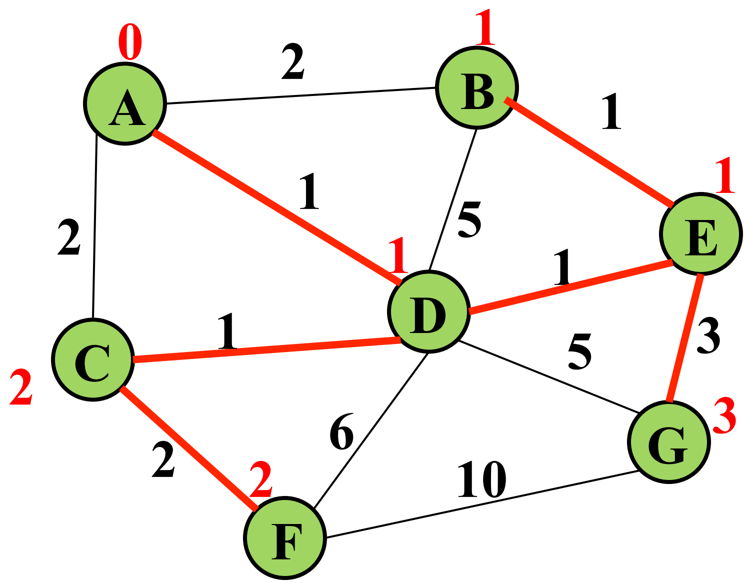
vertex	known?	cost	prev
A	Y	0	
B	Y	1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F		2	C
G		3	E

Prim's Example



vertex	known?	cost	prev
A	Y	0	
B	Y	1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F	Y	2	C
G		3	E

Prim's Example



vertex	known?	cost	prev
A	Y	0	
B	Y	1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F	Y	2	C
G	Y	3	E

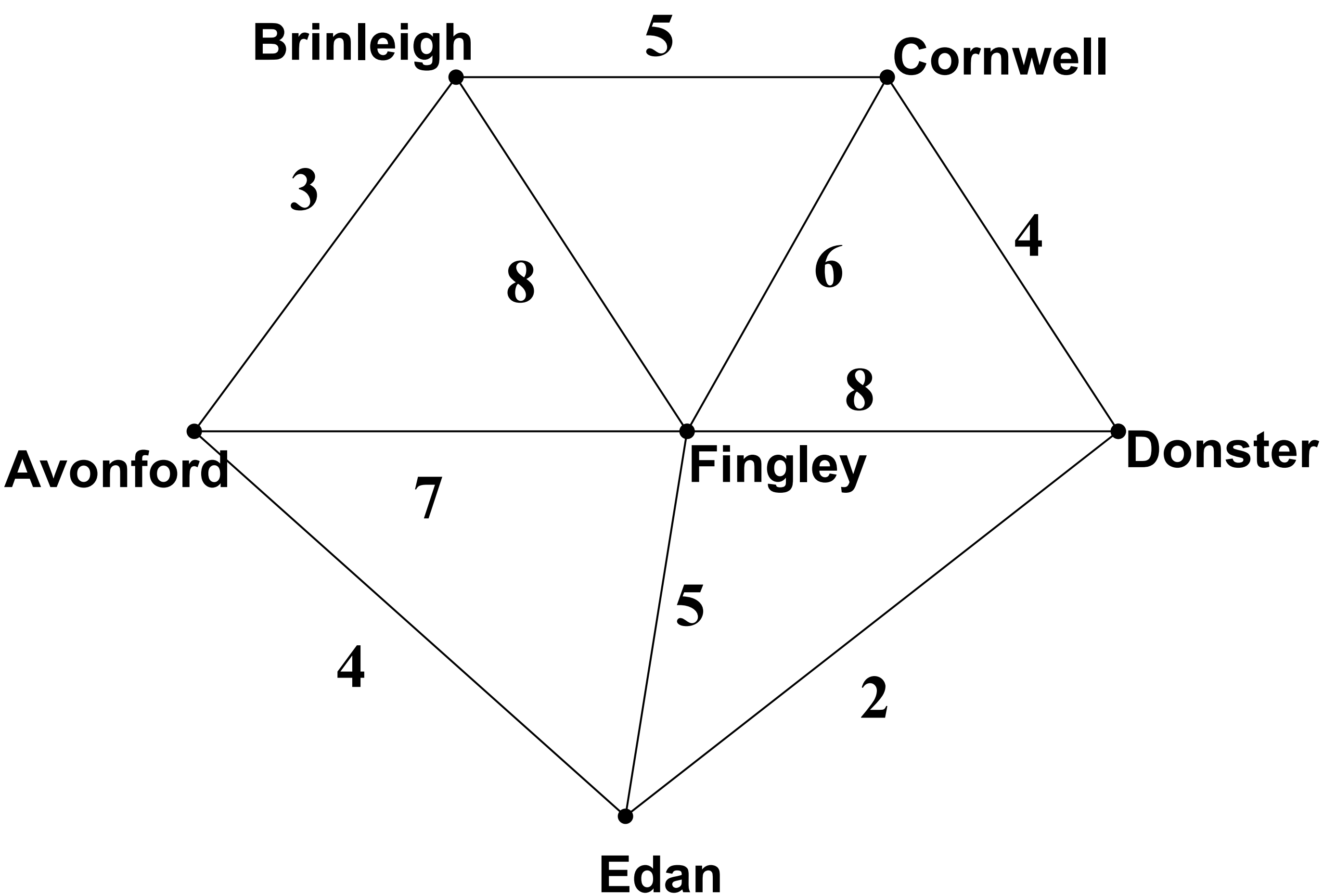
Analysis

- Correctness
  - A bit tricky
  - Intuitively similar to Dijkstra

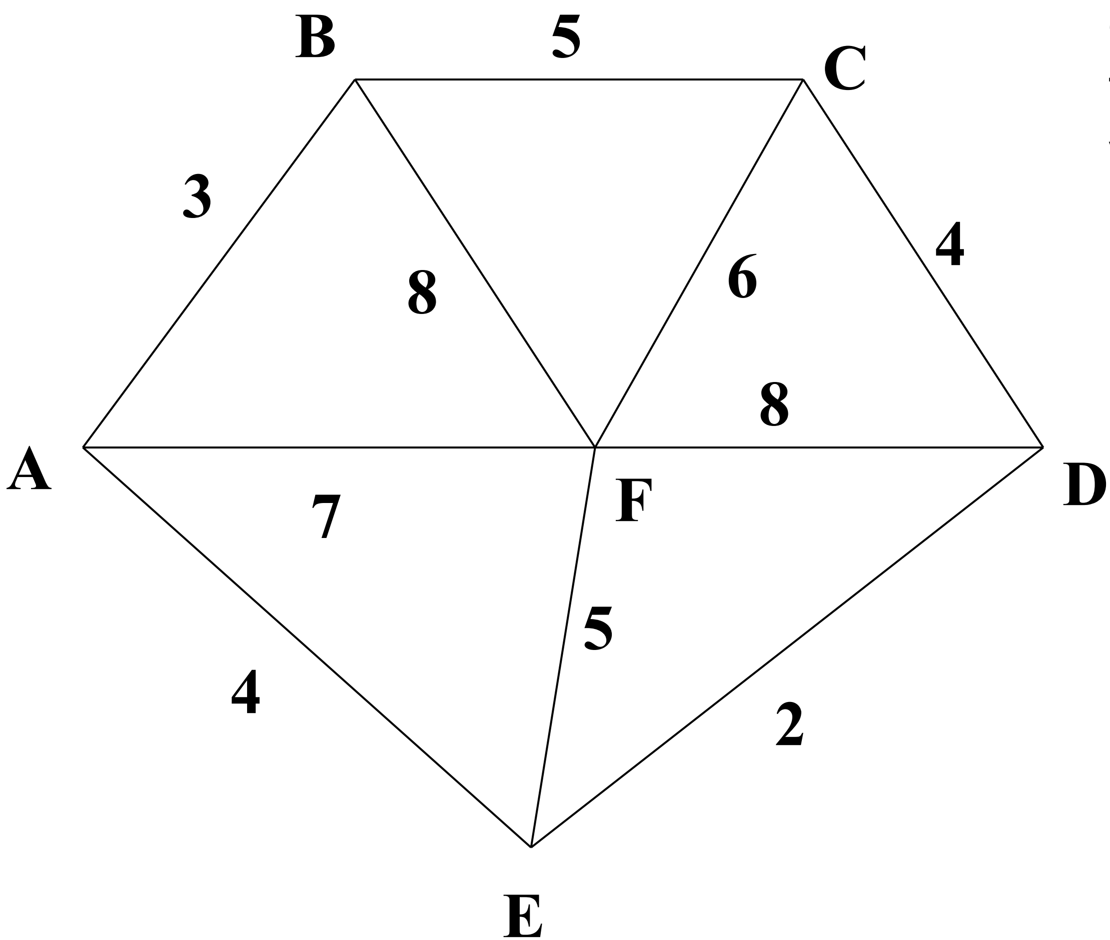
- Run-time
  - Same as Dijkstra
  - $O(|E| \texttt{log} |V|)$ using a priority queue
    - Costs/priorities are just edge-costs, not path-costs

Another Example

A cable company wants to connect five villages to their network which currently extends to the town of Avonford. What is the minimum length of cable needed?

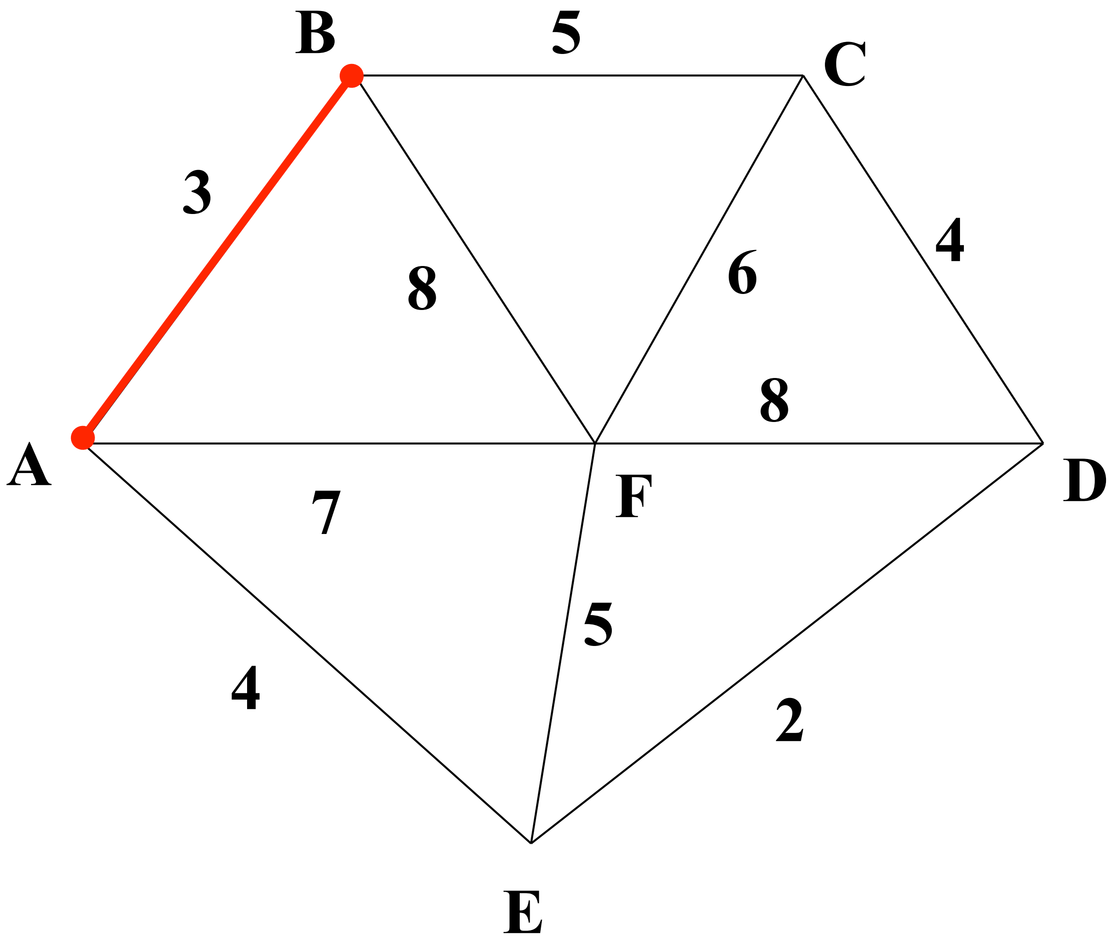


Prim's Algorithm



Model the situation as a graph and find the MST that connects all the villages (nodes).

Prim's Algorithm



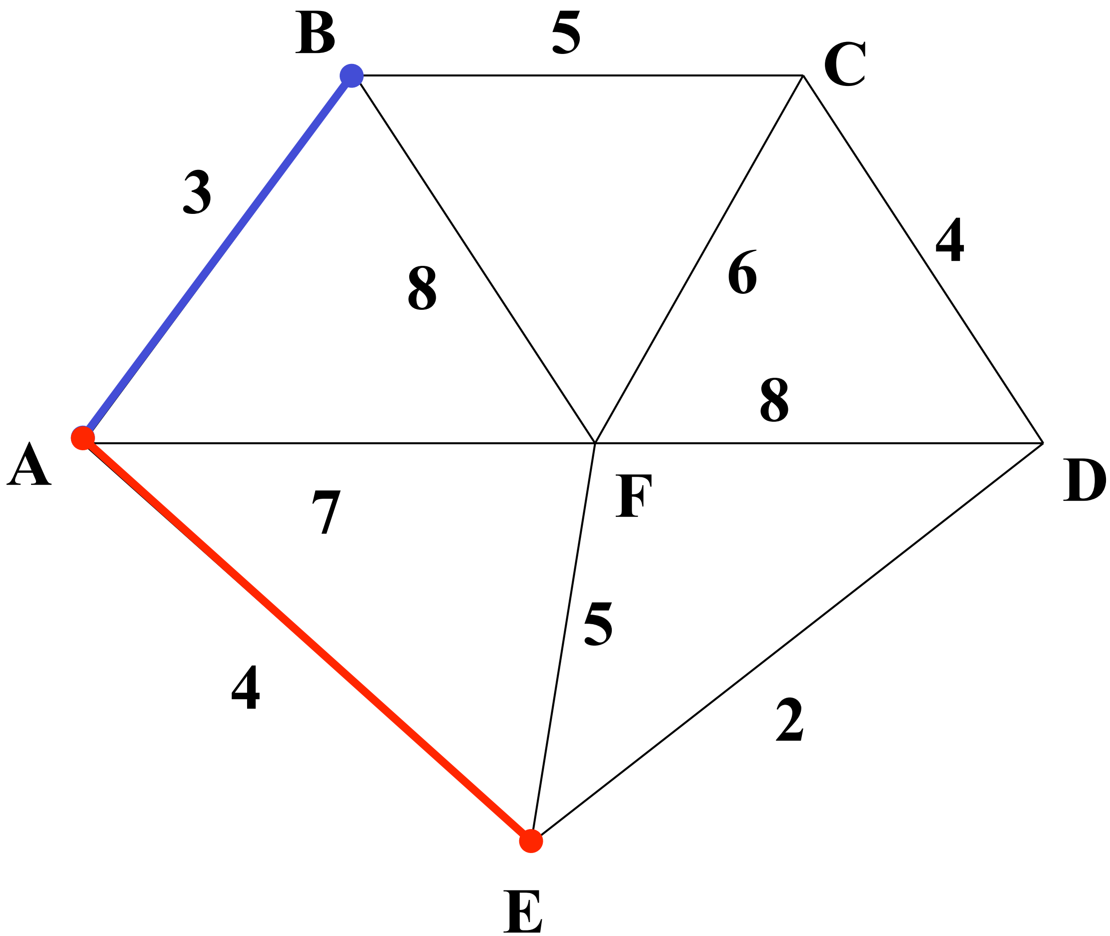
Select any vertex

A

Select the shortest edge connected to that vertex

AB 3

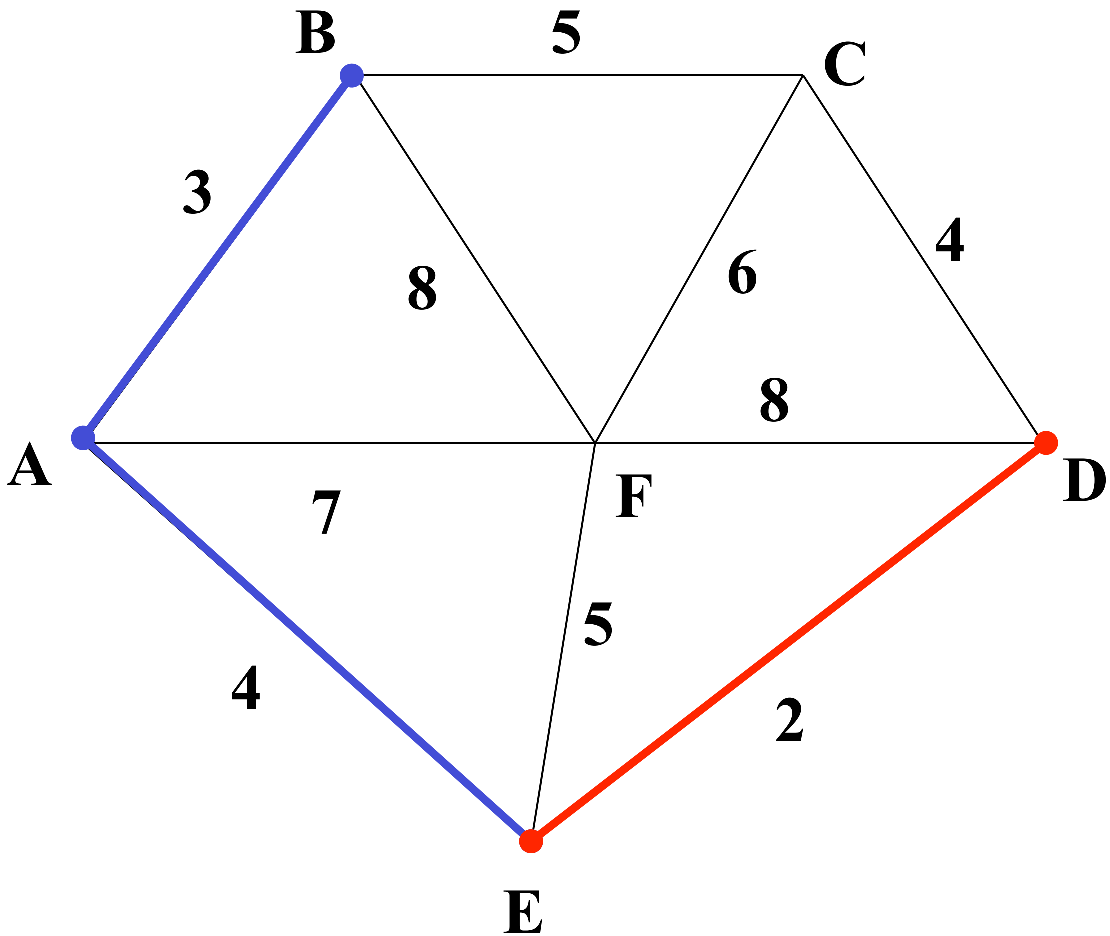
Prim's Algorithm



Select the shortest edge that connects an unknown vertex to any known vertex.

AE 4

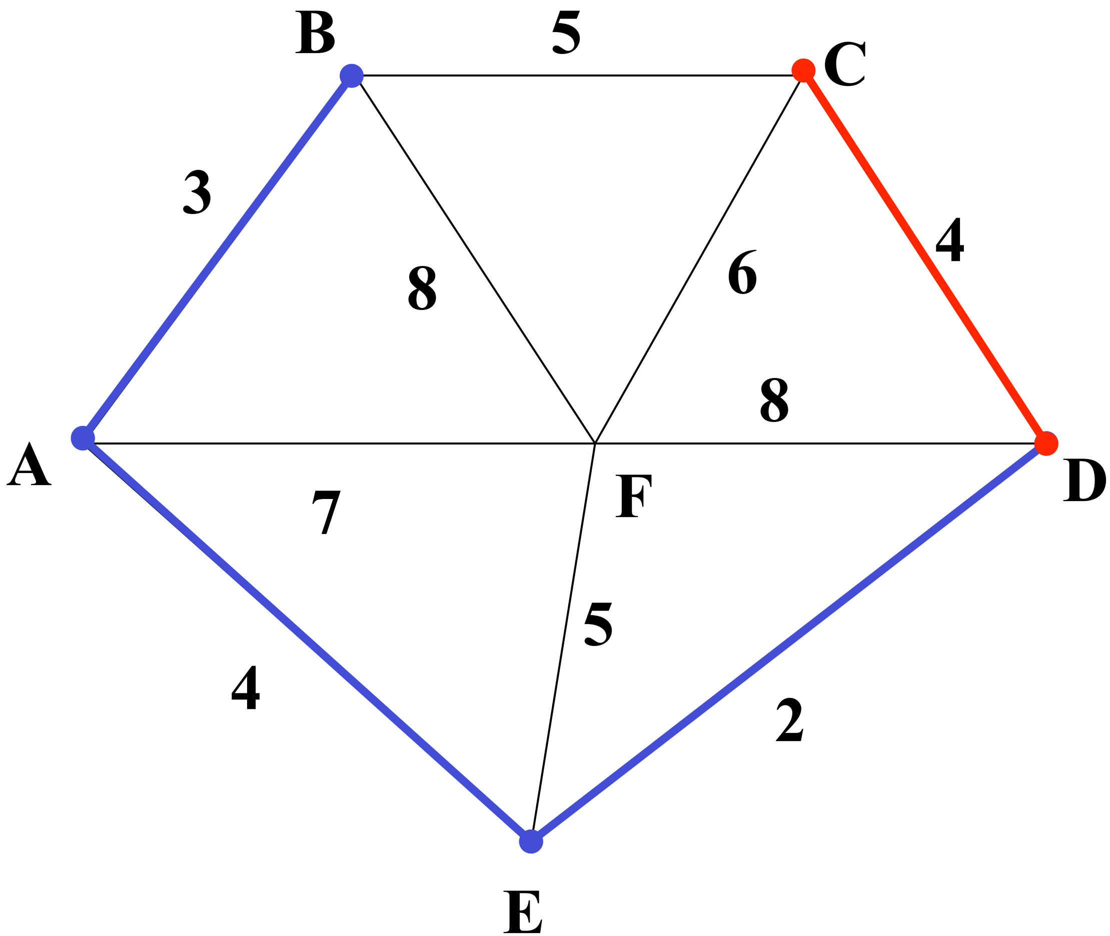
Prim's Algorithm



Select the shortest edge that connects an unknown vertex to any known vertex.

ED 2

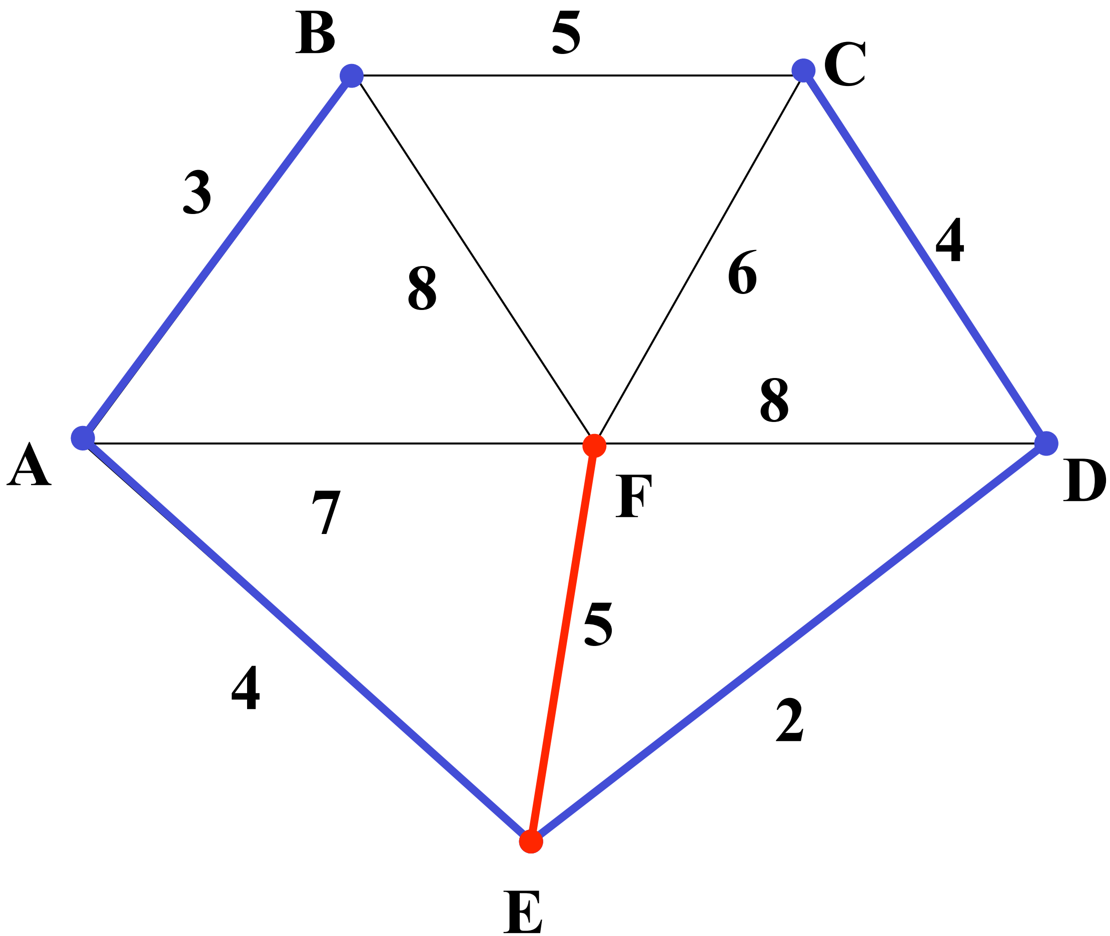
Prim's Algorithm



Select the shortest edge that connects an unknown vertex to any known vertex.

DC 4

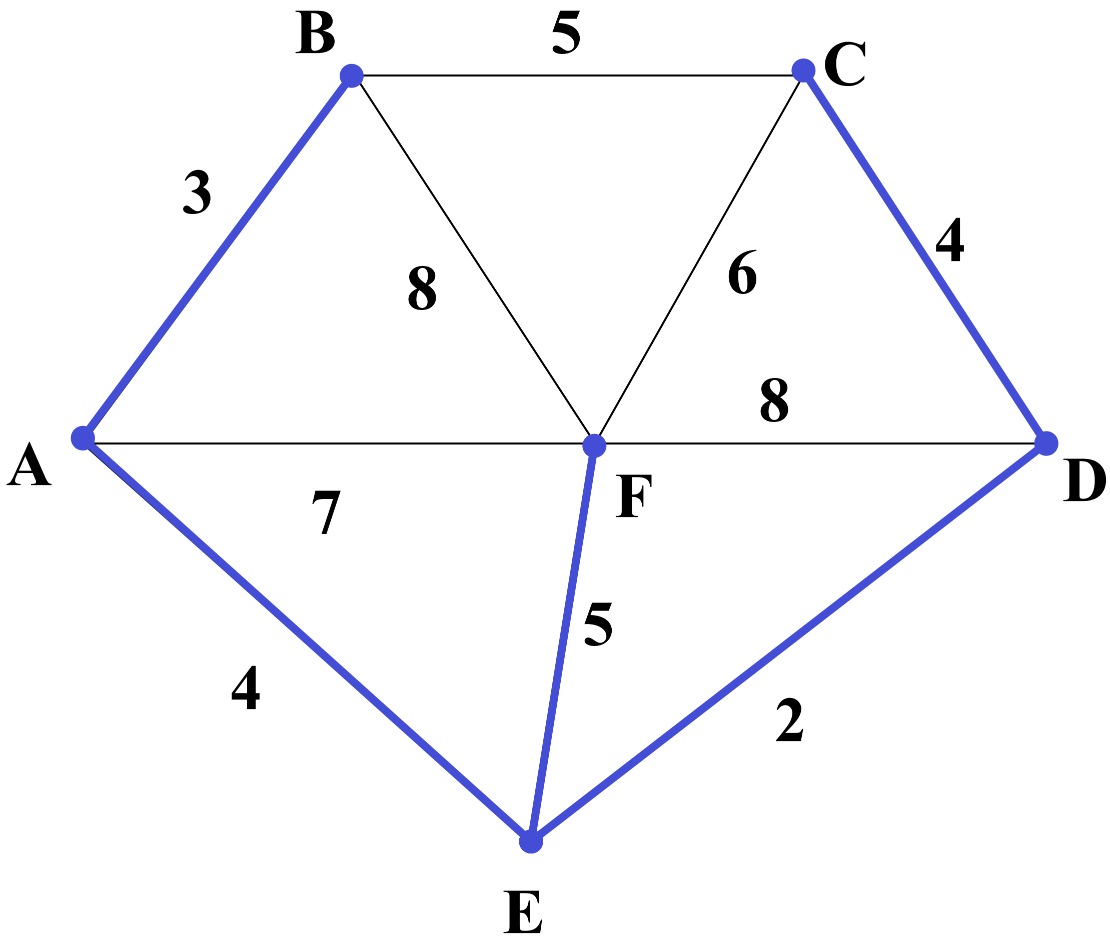
Prim's Algorithm



Select the shortest edge that connects an unknown vertex to any known vertex.

EF 5

Prim's Algorithm



All vertices have been connected.

The solution is

AB 3
AE 4
ED 2
DC 4
EF 5

Total weight of tree: 18

Minimum Spanning Tree Algorithms

- Prim's Algorithm for Minimum Spanning Tree
  - Similar idea to Dijkstra's Algorithm but for MSTs.
  - Both based on expanding cloud of known vertices

- Kruskal's Algorithm for Minimum Spanning Tree
  - Another, but different, greedy MST algorithm.
  - Uses the Union-Find data structure.

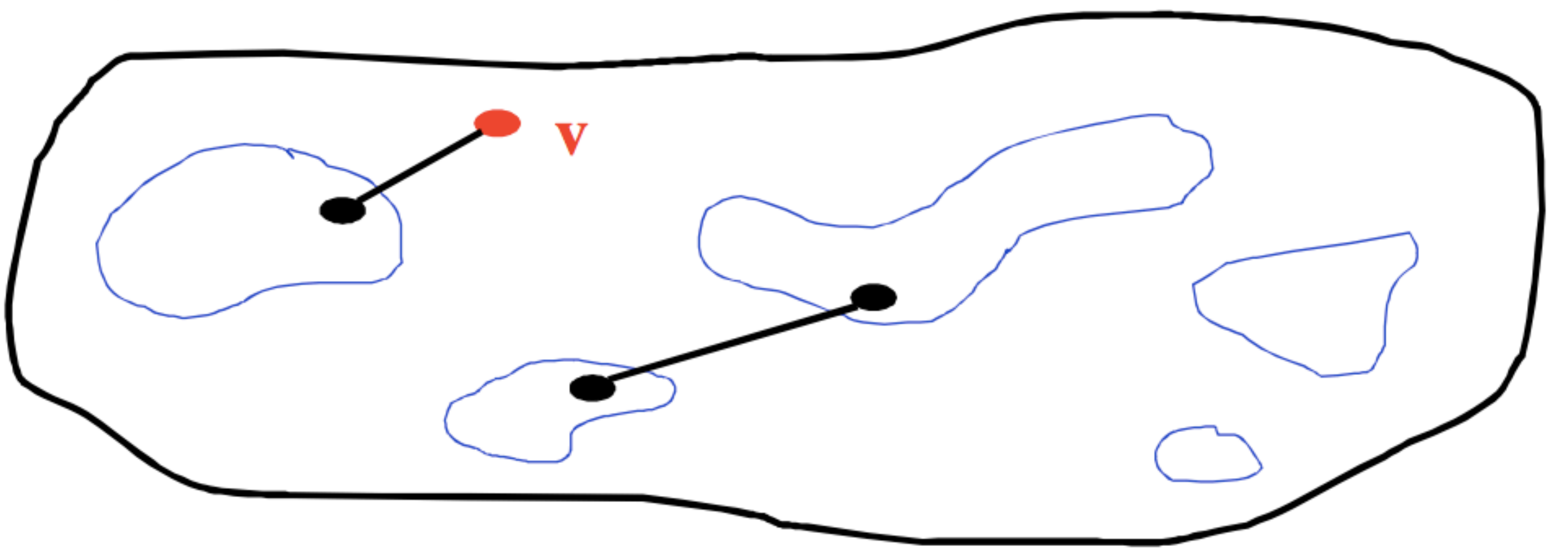
Kruskal's Algorithm

Idea: Grow a forest out of edges that do not create a cycle. Pick an edge with the smallest weight.

An *edge-based* greedy algorithm

Builds MST by greedily adding edges

G=(V,E)



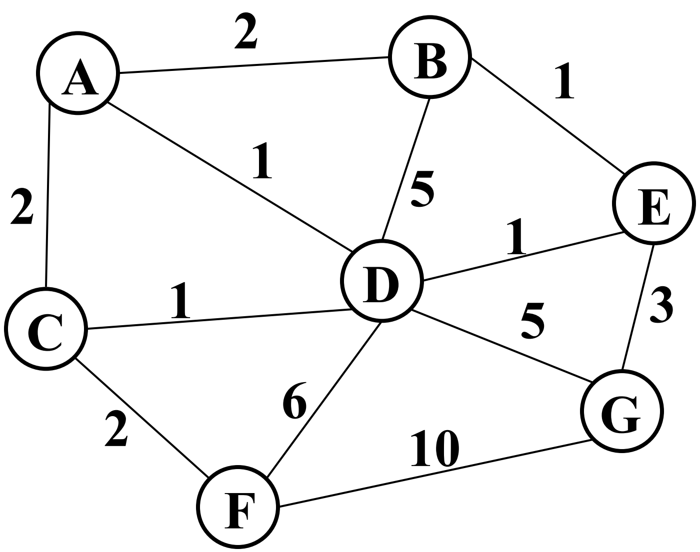
Kruskal's Algorithm Pseudocode

1. Sort edges by weight (better: put in min-heap)
2. Each node in its own set
3. While output size **< |V|-1**
   - Consider next smallest edge **`(u,v)`**
   - if **`find(u)`** and **`find(v)`** indicate **`u`** and **`v`** are in different sets
     - output **`(u,v)`**
     - **`union(find(u),find(v))`**

invariant:

`u` and **`v`** in same set if and only if connected in output-so-far

Kruskal's Example



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

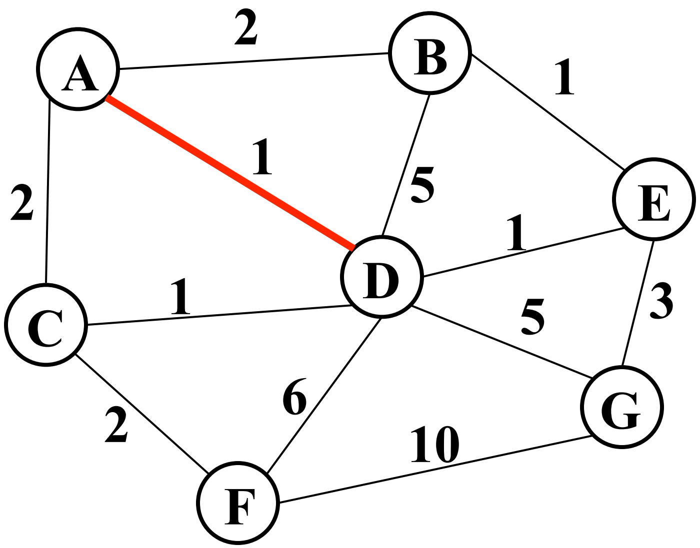
6: (D,F)

10: (F,G)

Output:

Note: At each step, the union/find sets are the trees in the forest

Kruskal's Example



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

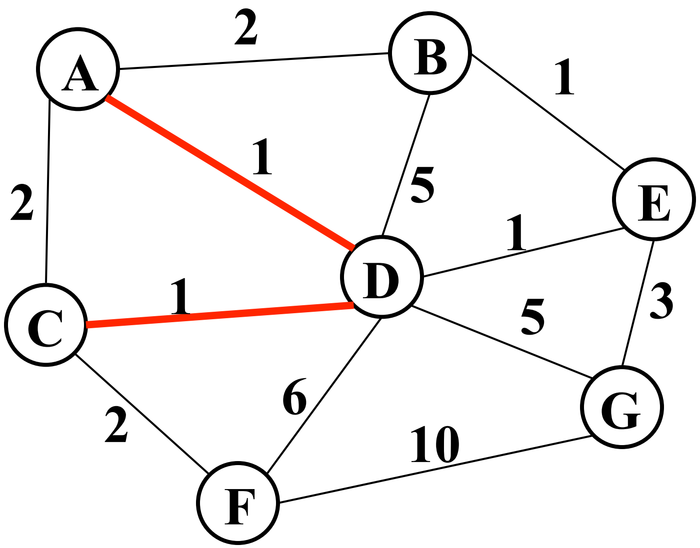
6: (D,F)

10: (F,G)

Output: (A,D)

Note: At each step, the union/find sets are the trees in the forest

Kruskal's Example



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

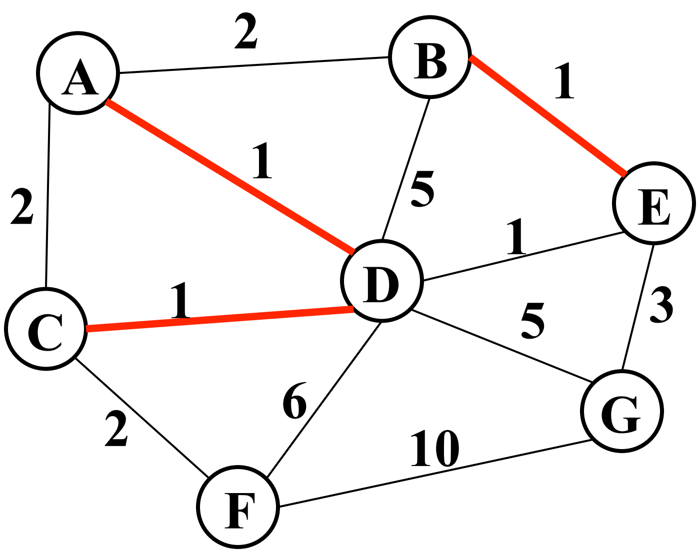
6: (D,F)

10: (F,G)

Output: (A,D), (C,D)

Note: At each step, the union/find sets are the trees in the forest

Kruskal's Example



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

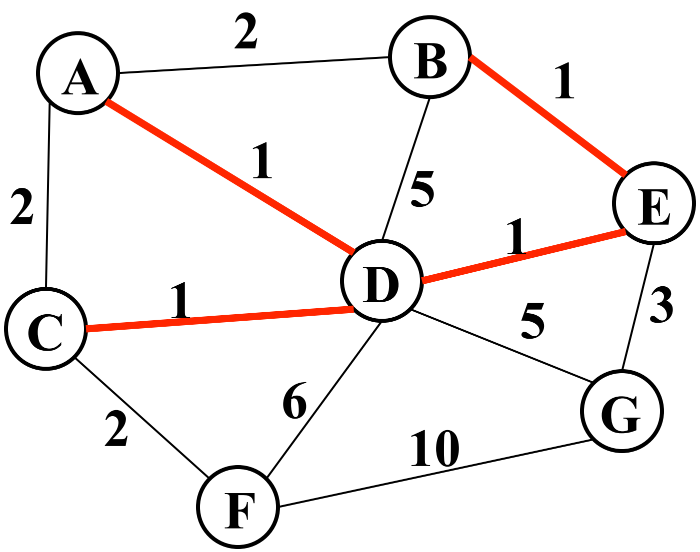
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E)

Note: At each step, the union/find sets are the trees in the forest

Kruskal's Example



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

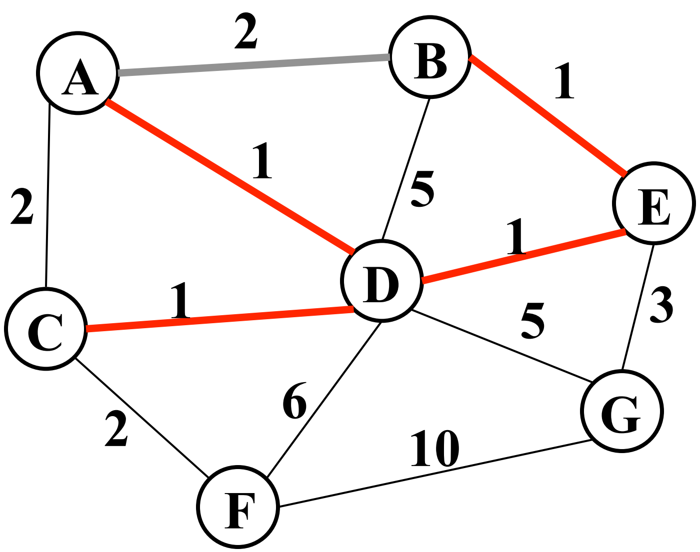
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E)

Note: At each step, the union/find sets are the trees in the forest

Kruskal's Example



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

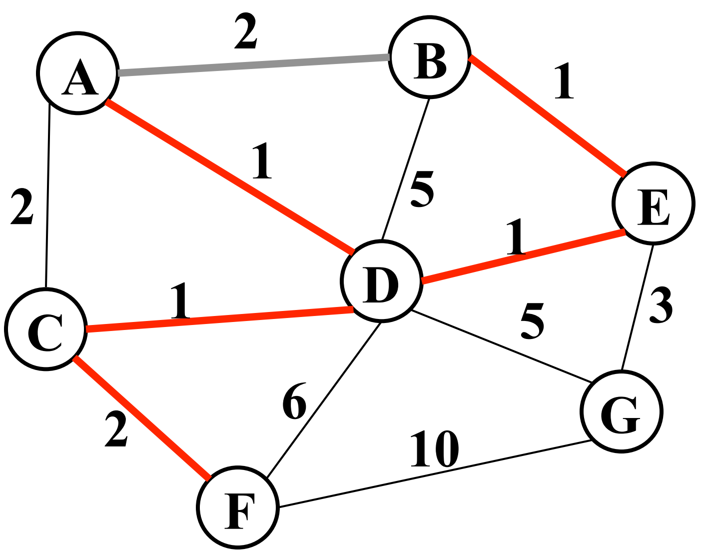
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E)

Note: At each step, the union/find sets are the trees in the forest

Kruskal's Example



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

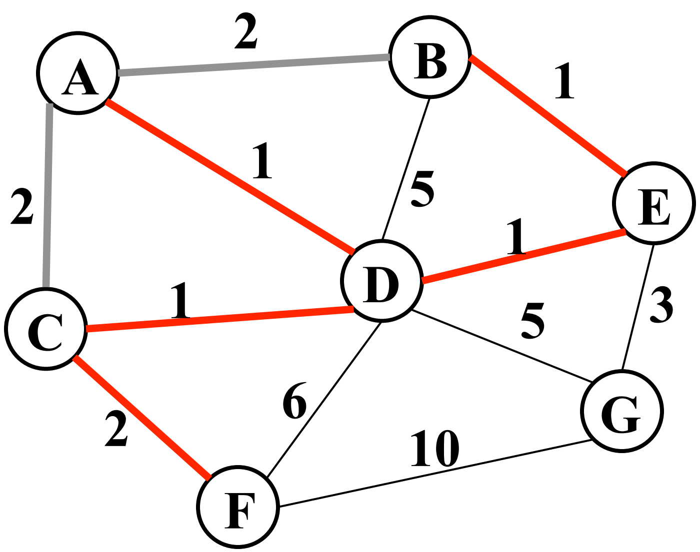
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E), (C,F)

Note: At each step, the union/find sets are the trees in the forest

Kruskal's Example



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

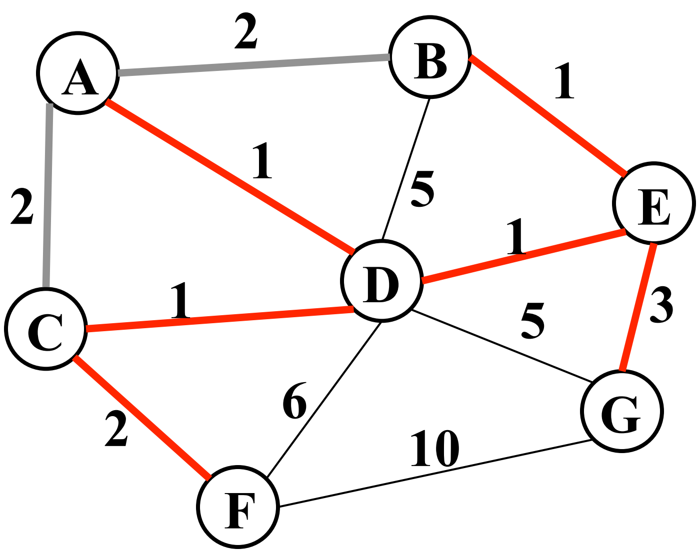
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E), (C,F)

Note: At each step, the union/find sets are the trees in the forest

Kruskal's Example



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E), (C,F), (E,G)

Note: At each step, the union/find sets are the trees in the forest

Kruskal's Algorithm Analysis

Idea: Grow a forest out of edges that do not grow a cycle

But now consider the edges in order by weight

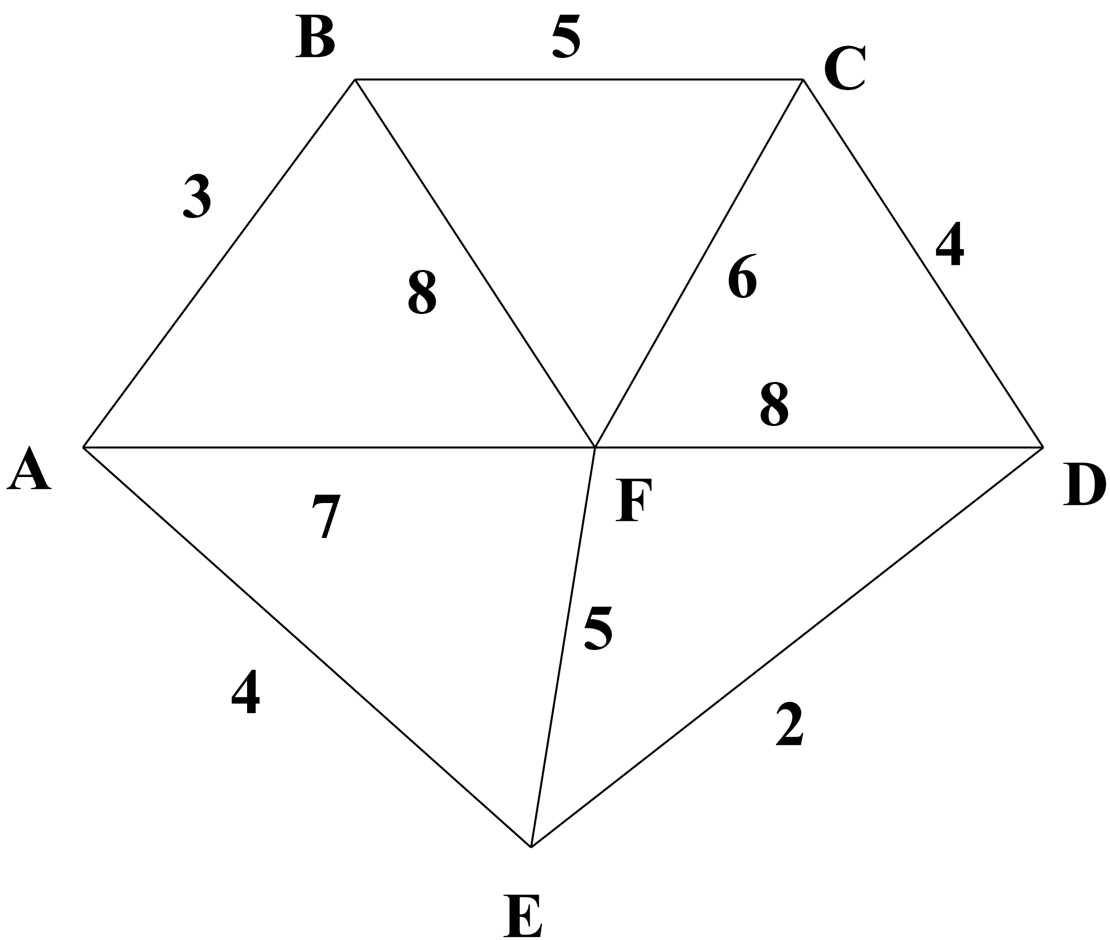
So:

- Sort edges: *O*(**|E|**`log` **|E|**) (next lecture)
- Iterate through edges using union-find for cycle detection almost *O*(**|E|**)

Somewhat better:

- Build min-heap with edges *O*(**|E|**) (Floyd's algorithm)
- Iterate through edges using union-find for cycle detection and **`deleteMin`** to get next edge *O*(**|E|**`log`**|E|**)
- Not better *worst-case* asymptotically, but often stop long before considering all edges.

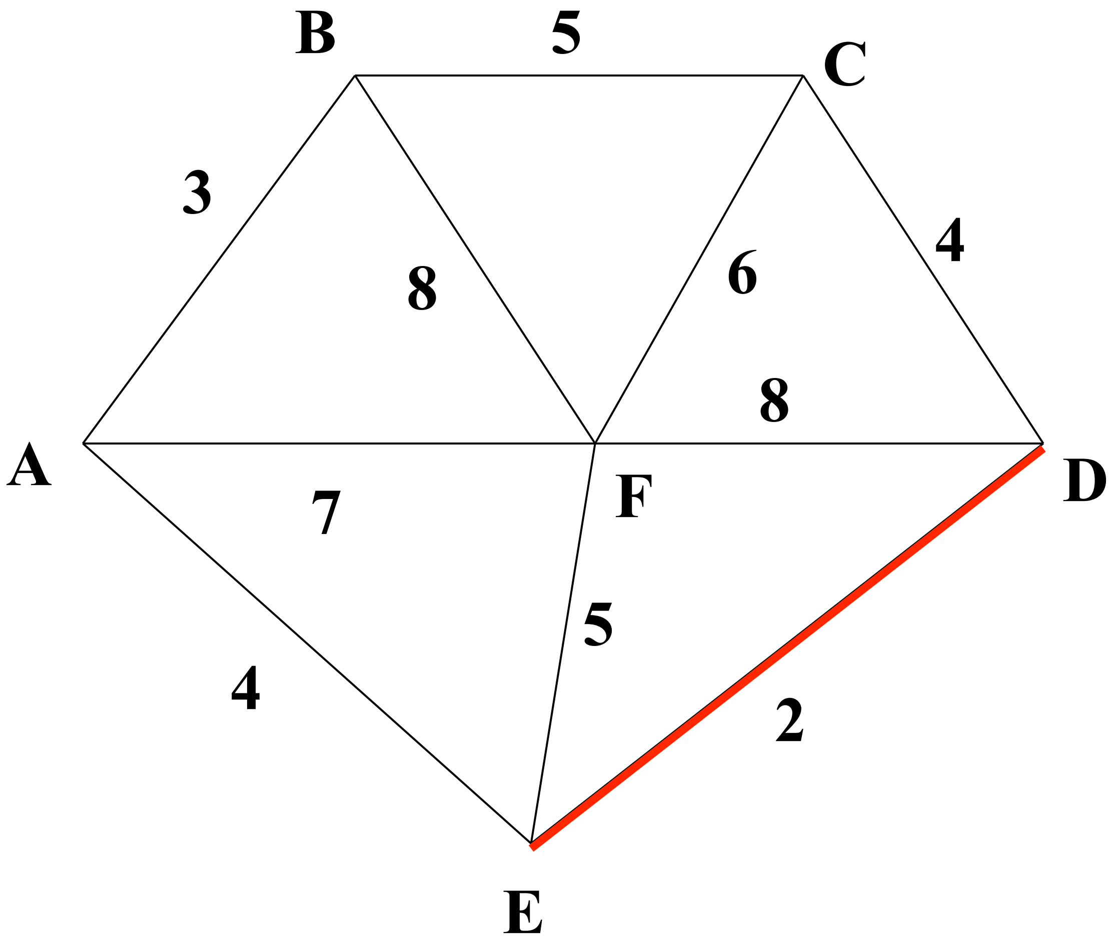
Kruskal's Algorithm



List the edges in order of size:

ED 2
AB 3
AE 4
CD 4
BC 5
EF 5
CF 6
AF 7
BF 8
CF 8

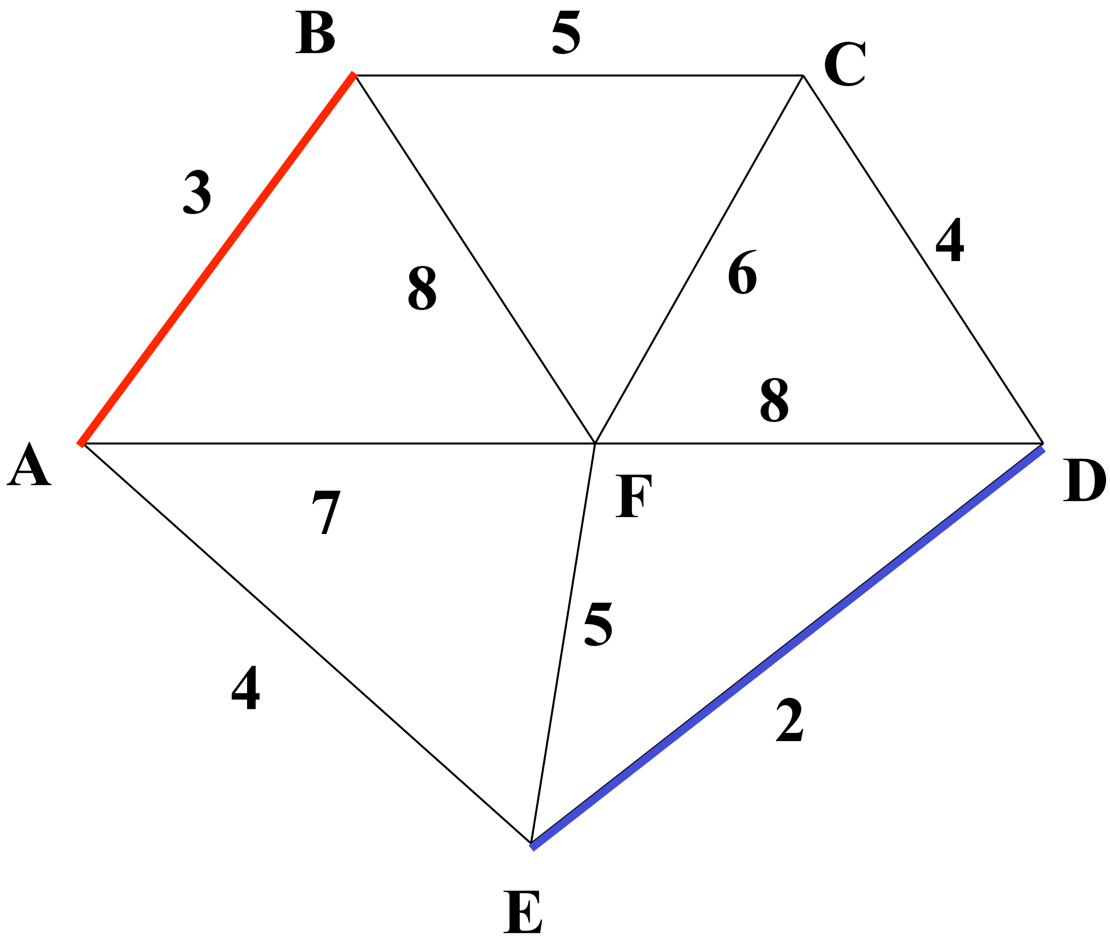
Kruskal's Algorithm



Select the edge with min cost

ED 2

Kruskal’s Algorithm

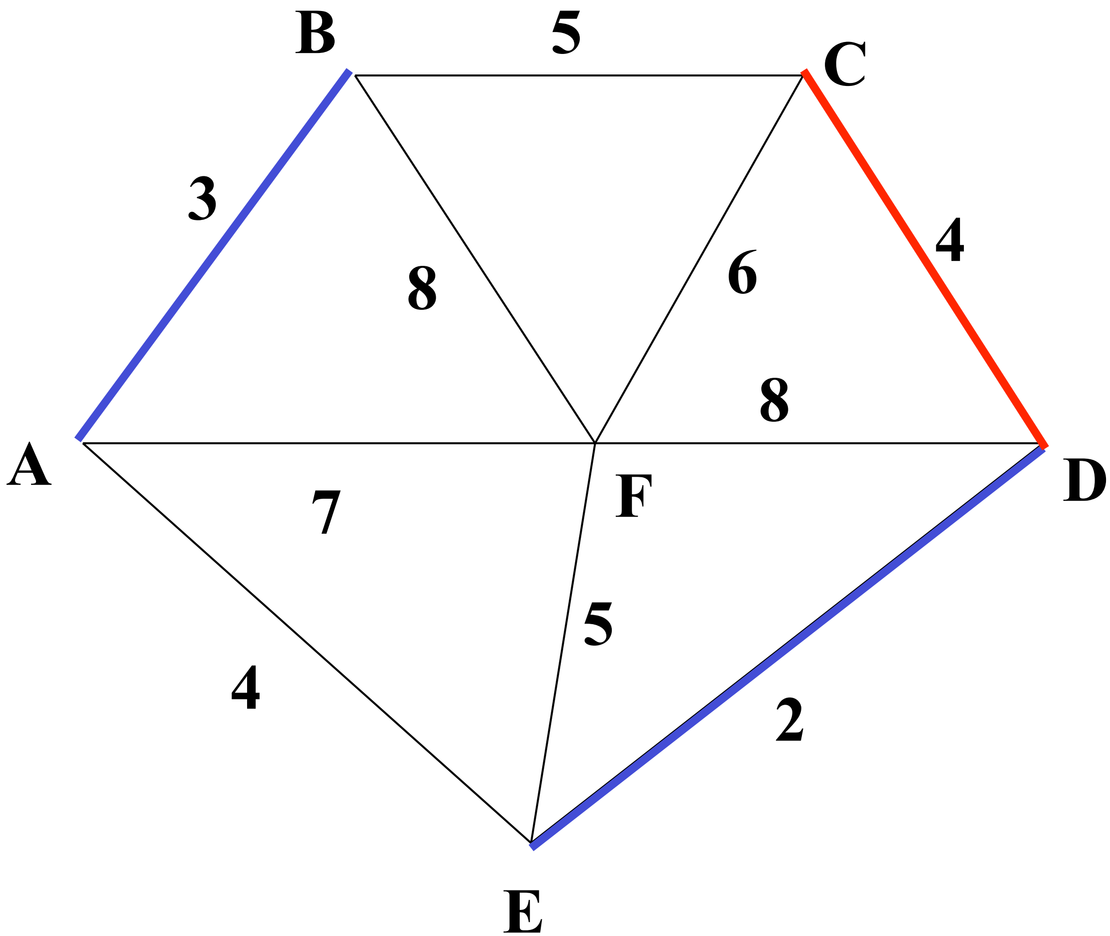


Select the next minimum cost edge that does not create a cycle

ED 2

AB 3

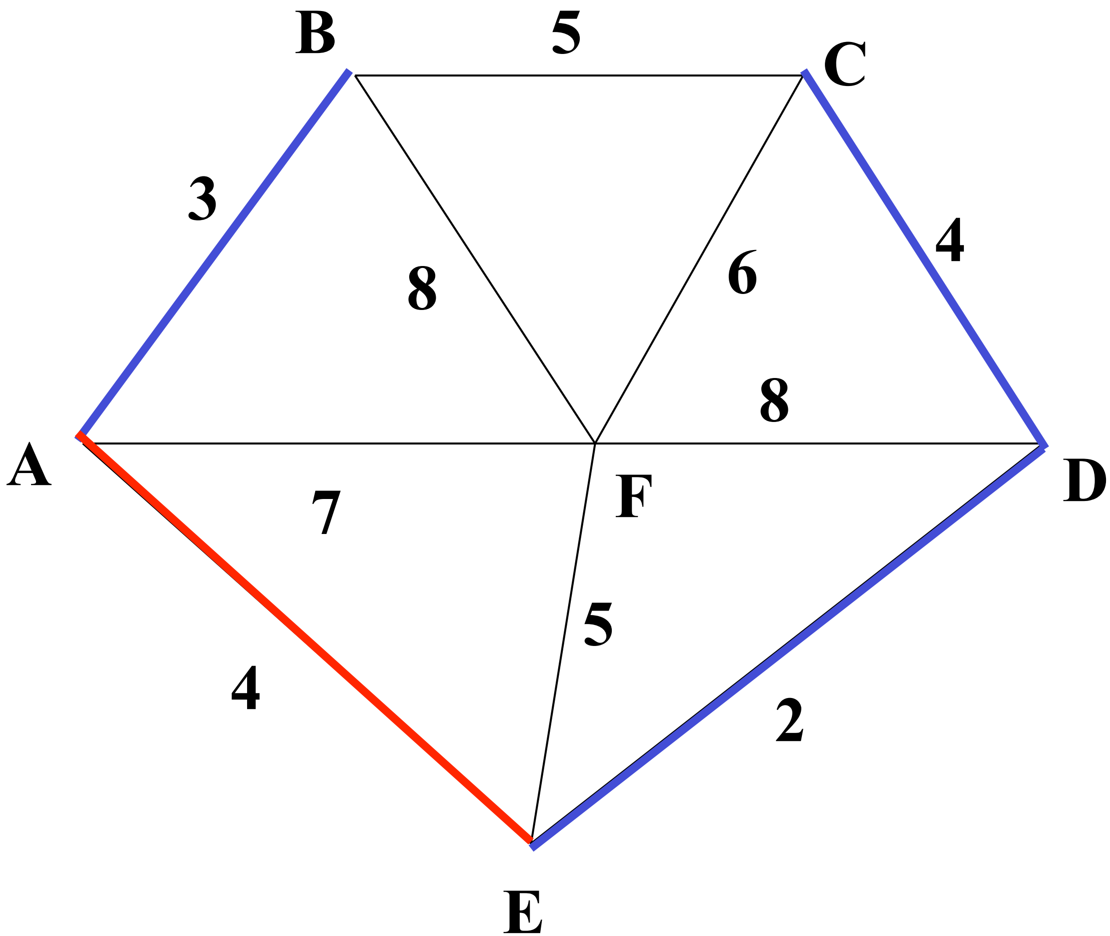
Kruskal's Algorithm



Select the next minimum cost edge that does not create a cycle

ED 2
AB 3
CD 4 (or AE 4)

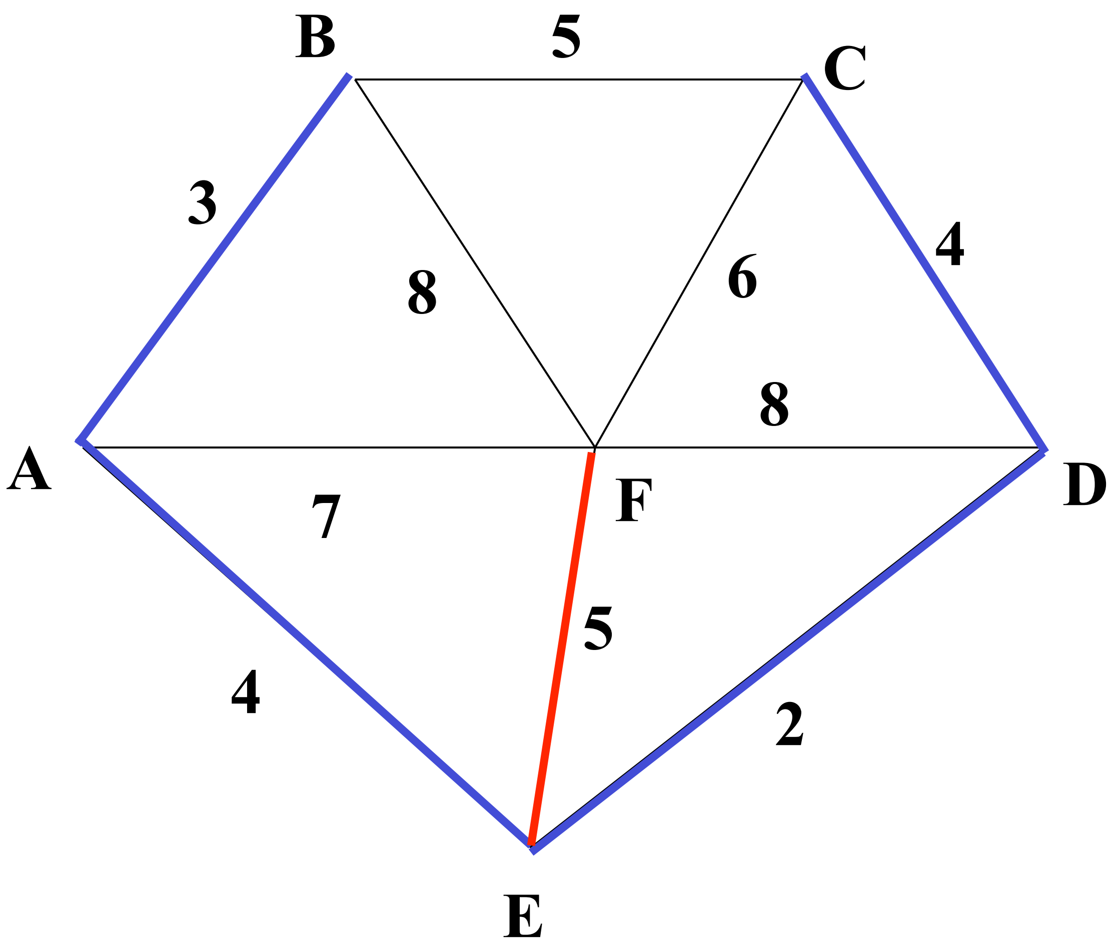
Kruskal’s Algorithm



Select the next minimum cost edge that does not create a cycle

ED 2
AB 3
CD 4
AE 4

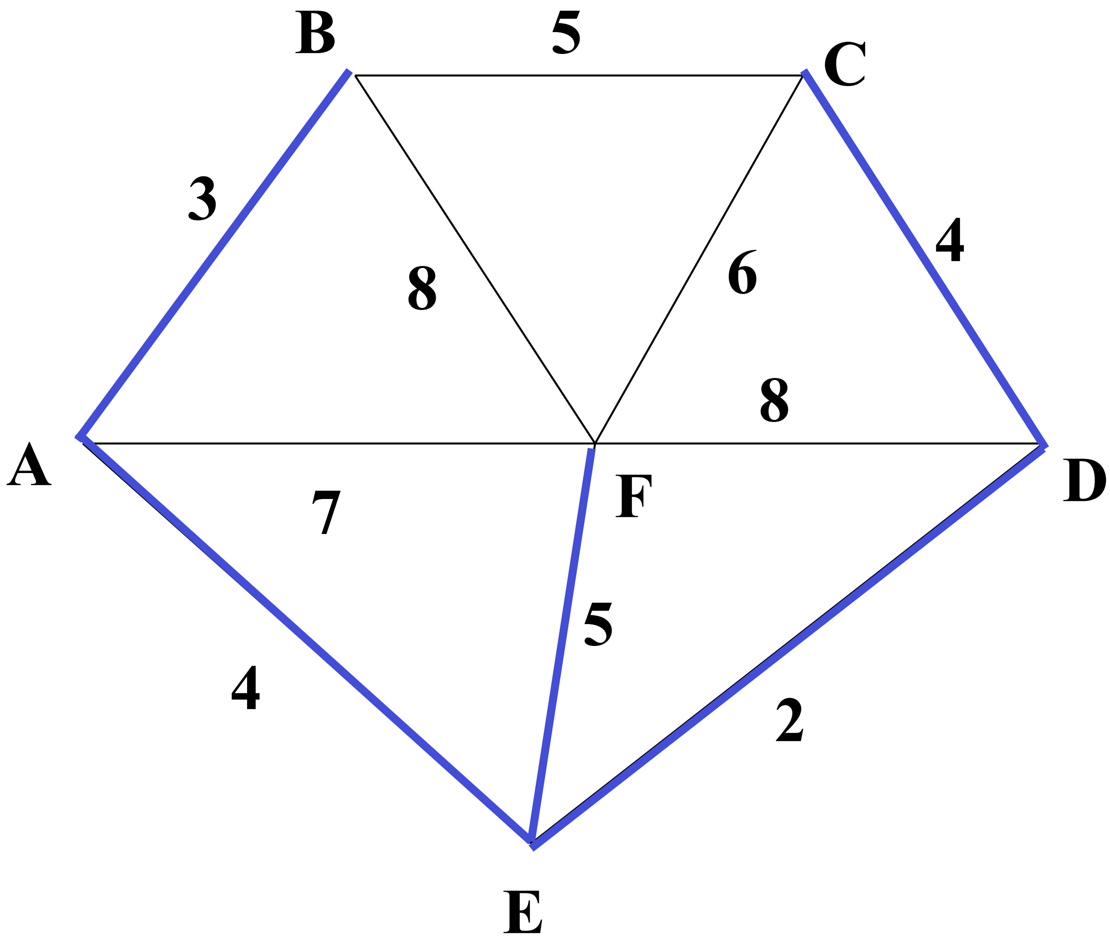
Kruskal's Algorithm



Select the next minimum cost edge that does not create a cycle

ED 2
AB 3
CD 4
AE 4
BC 5 – forms a cycle
EF 5

Kruskal's Algorithm



All vertices have been connected.

The solution is

ED 2
AB 3
CD 4
AE 4
EF 5

Total weight of tree: 18

Done with graph algorithms!

Next lecture…

- Sorting
- More sorting
- Even more sorting

☺

Homework 5

- Due 11pm next Wednesday
- You may work with a partner
- Create graph representation in MyGraph.java
  - adjacency list or adjacency matrix
  - don’t change constructor!
  - deal with edge cases/exceptions as outlined in html
  - probably want to use map to look up info about some vertex
- Compute shortestPath() using Dijkstra’s
  - not required to use priority queue to store un-explored vertices
  - use equals, not == to determine if same vertex, FindPaths() create copies of vertices
  - finish FindPaths.java so it prints correct output
- Test and Readme