Minimum Spanning Tree

Slides by Shrirang (Shri) Mare
shri@cs.washington.edu

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Robbie Weber, Whitaker Brand, Zora Fung, Stuart Reges, Justin Hsia, Ruth Anderson, and many others for sample slides and materials ...

Four classes of graph problem

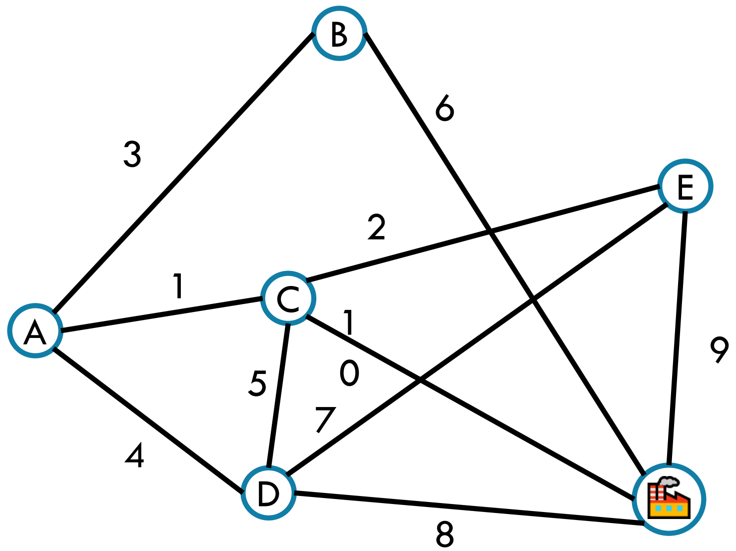
.. that can be solved efficiently (in polynomial time)

1. Shortest path – find a shortest path between two vertices in a graph
2. Minimum spanning tree – find subset of edges with minimum total weights
3. Matching – find set of edges without common vertices
4. Maximum flow – find the maximum flow from a source vertex to a sink vertex

A wide array of graph problems that can be solved in polynomial time are variants of these above problems.

Minimum Spanning Trees

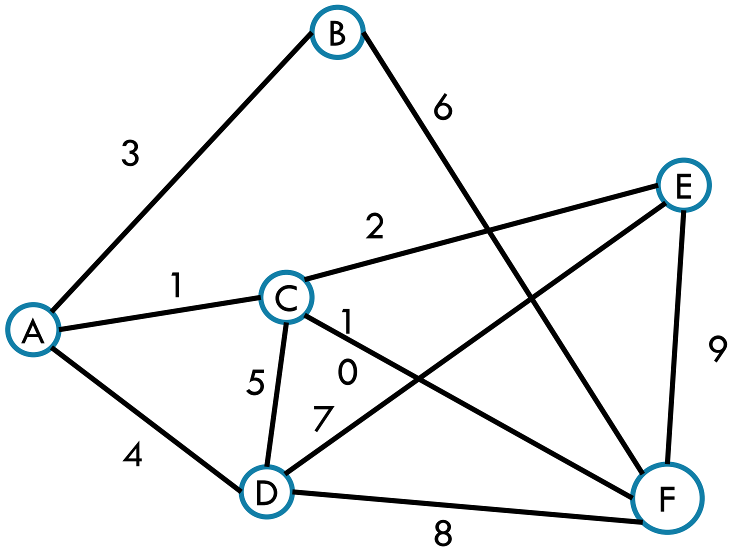
It's the 1920's. Your friend at the electric company needs to choose where to build wires to connect all these cities to the plant.



She knows how much it would cost to lay electric wires between any pair of locations, and wants the cheapest way to make sure electricity from the plant to every city.

Minimum Spanning Trees

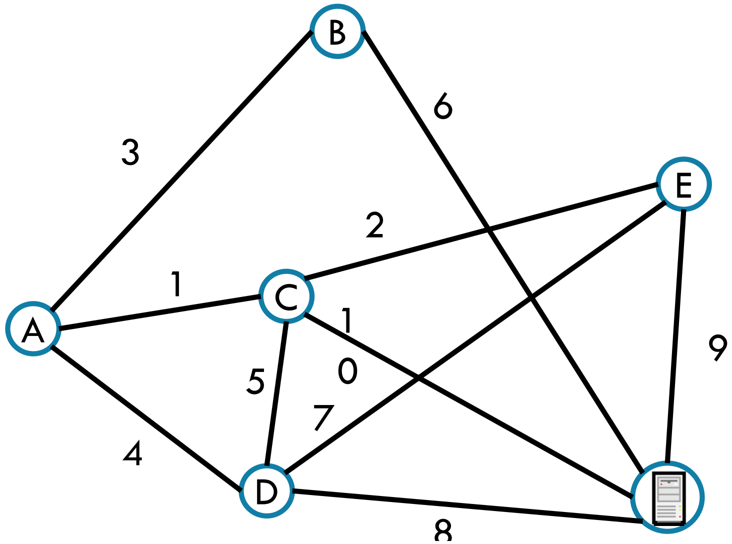
It's the 1950's Your boss at the phone company needs to choose where to build wires to connect all these cities to each other.



She knows how much it would cost to lay elec phone between any pair of locations, and wants the cheapest way to make sure electricit Everyone can call everyone else.

Minimum Spanning Trees

It's · today . Your friend at the ISP needs to choose where to build wires to connect all these cities to the Internet with fiber optic cable



She knows how much it would cost to lay elec cable 'een any pair of locations, and wants the cheapest way to make sure electricit Everyone can reach the server

Minimum Spanning Trees

What do we need? A set of edges such that:

- Every vertex touches at least one of the edges. (the edges **span** the graph)
- The graph on just those edges is **connected**.
- The minimum weight set of edges that meet those conditions.

Assume all edge weights are positive.

Claim: The set of edges we pick never has a cycle. Why?

MST Problem

What do we need? A set of edges such that:

- Every vertex touches at least one of the edges. (the edges **span** the graph)
- The graph on just those edges is **connected**.
- The minimum weight set of edges that meet those conditions.

Our goal is a tree!

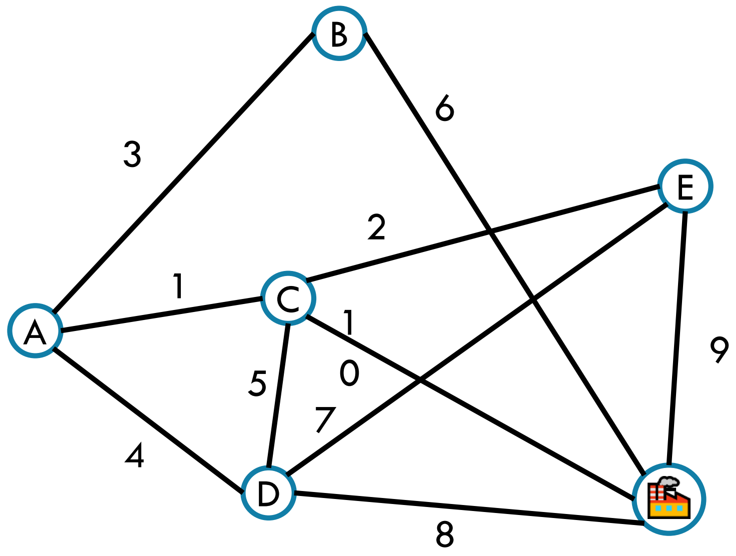
Minimum Spanning Tree Problem

Given: an undirected, weighted graph G
Find: A minimum-weight set of edges such that you can get from any vertex of G to any other on only those edges.

We'll go through two different algorithms for this problem today.

Example

Try to find a MST of this graph:



Prim's Algorithm

Algorithm idea: choose an arbitrary starting point. Add a new edge that:

- Will let you reach more vertices.
- Is as light as possible

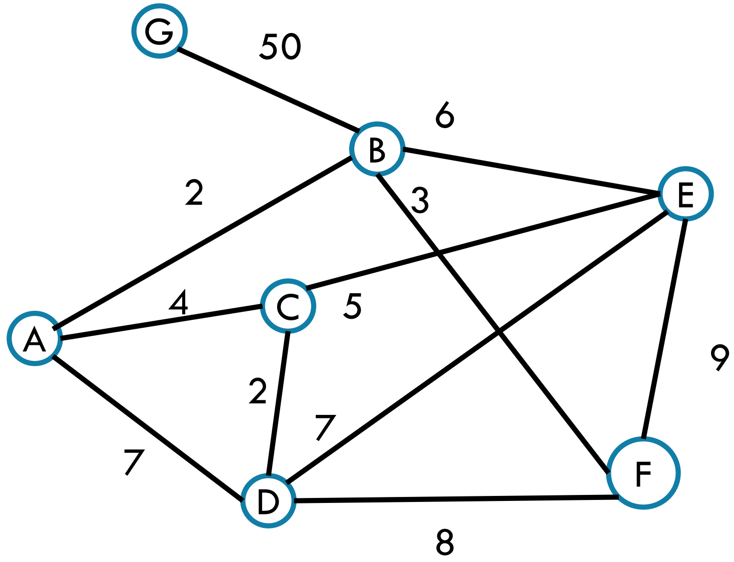
We'd like each not-yet-connected vertex to be able to tell us the lightest edge we could add to connect it.

Code

```
PrimMST(Graph G)
   initialize distances to ∞
   mark source as distance 0
   mark all vertices unprocessed
   foreach(edge (source, v) )
      v.dist = w(source,v)
   while(there are unprocessed vertices){
      let u be the closest unprocessed vertex
      add u.bestEdge to spanning tree
      foreach(edge (u,v) leaving u){
         if(w(u,v) < v.dist){
            v.dist = w(u,v)
            v.bestEdge = (u,v)
         }
      }
     mark u as processed
   }
```

Try it Out

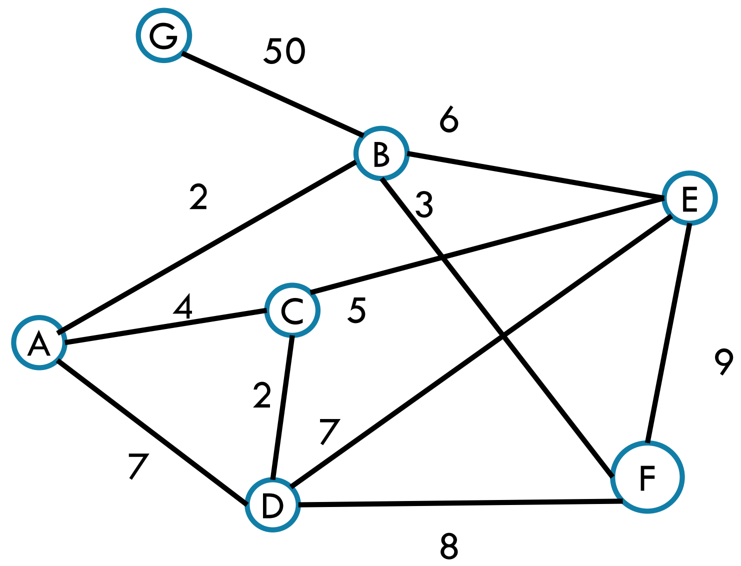
```
PrimMST(Graph G)
   initialize distances to ∞
   mark source as distance 0
   mark all vertices unprocessed
   foreach(edge (source, v) )
      v.dist = w(source,v)
   while(there are unprocessed vertices){
      let u be the closest unprocessed vertex
      add u.bestEdge to spanning tree
      foreach(edge (u,v) leaving u){
         if(w(u,v) < v.dist){
            v.dist = w(u,v)
            v.bestEdge = (u,v)
         }
      }
     mark u as processed
   }
```



Vertex	Distance	Best Edge	Processed
A			
B			
C			
D			
E			
F			
G			

Try it Out

```
PrimMST(Graph G)
   initialize distances to ∞
   mark source as distance 0
   mark all vertices unprocessed
   foreach(edge (source, v) )
      v.dist = w(source,v)
   while(there are unprocessed vertices){
      let u be the closest unprocessed vertex
      add u.bestEdge to spanning tree
      foreach(edge (u,v) leaving u){
         if(w(u,v) < v.dist){
            v.dist = w(u,v)
            v.bestEdge = (u,v)
         }
      }
     mark u as processed
   }
```



Vertex	Distance	Best Edge	Processed
A			
B			
C			
D			
E			
F			
G			

Does This Algorithm Always Work?

Prim's Algorithm is a **greedy** algorithm. Once it decides to include an edge in the MST it never reconsiders its decision.

Greedy algorithms rarely work.

There are special properties of MSTs that allow greedy algorithms to find them.

In fact MSTs are so *magical* that there's more than one greedy algorithm that works.

A different Approach

Prim’s Algorithm started from a single vertex and reached more and more other vertices.

Prim’s thinks vertex by vertex (add the closest vertex to the currently reachable set).

What if you think edge by edge instead?

Start from the lightest edge; add it if it connects new things to each other (don’t add it if it would create a cycle)

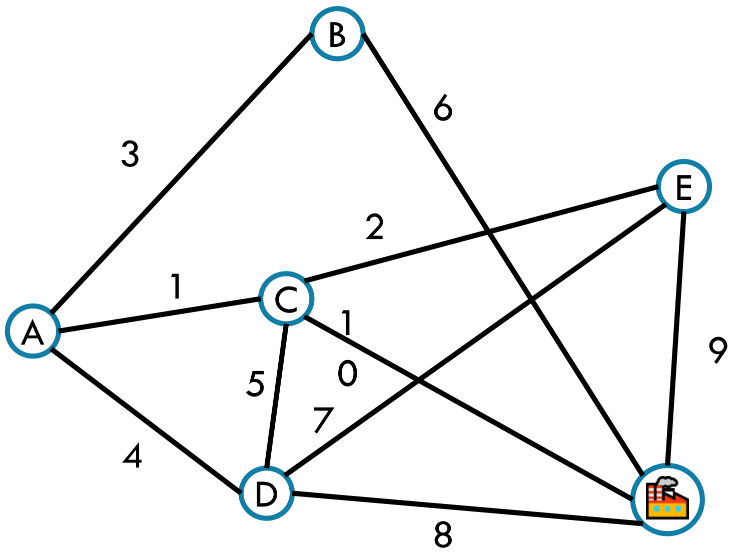
This is Kruskal’s Algorithm.

Kruskal's Algorithm

```
KruskalMST(Graph G)
   initialize each vertex to be a connected component
   sort the edges by weight
   foreach(edge (u, v) in sorted order){
      if(u and v are in different components){
         add (u,v) to the MST
         Update u and v to be in the same component
      }
   }
```

Try It Out

```
KruskalMST(Graph G)
   initialize each vertex to be a connected component
    sort the edges by weight
    foreach(edge (u, v) in sorted order){
        if(u and v are in different components){
            add (u,v) to the MST
            Update u and v to be in the same component
        }
    }
```

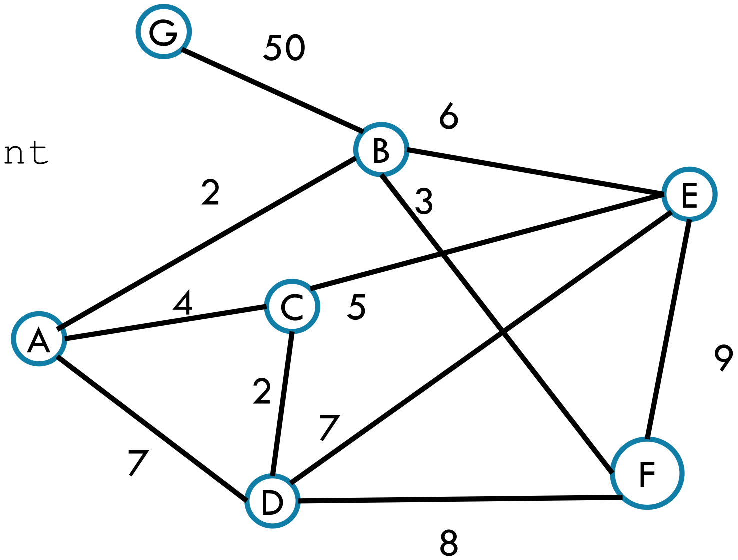


Kruskal's Algorithm: Running Time

```
KruskalMST(Graph G)
   initialize each vertex to be a connected component
   sort the edges by weight
   foreach(edge (u, v) in sorted order){
      if(u and v are in different components){
         add (u,v) to the MST
         Update u and v to be in the same component
      }
   }
```

Try it Out

```
KruskalMST(Graph G)
   initialize each vertex to be a connected component
   sort the edges by weight
   foreach(edge (u, v) in sorted order){
      if(u and v are in different components){
         add (u,v) to the MST
         Update u and v to be in the same component
      }
   }
```



Some Extra Comments

Prim was the employee at Bell Labs in the 1950's

The mathematician in the 1920's was Boruvka

- He had a different *also greedy* algorithm for MSTs.
- Boruvka's algorithm is trickier to implement, but is useful in some cases.

There's at least a fourth greedy algorithm for MSTs…

If all the edge weights are distinct, then the MST is unique.

If some edge weights are equal, there may be multiple spanning trees. Prim's/Dijkstra's are only guaranteed to find you one of them.

Why do all of these MST Algorithms Work?

MSTs satisfy two very useful properties:

Cycle Property: The heaviest edge along a cycle is NEVER part of an MST.

Cut Property: Split the vertices of the graph any way you want into two sets A and B. The lightest edge with one endpoint in A and the other in B is ALWAYS part of an MST.

Whenever you add an edge to a tree you create exactly one cycle, you can then remove any edge from that cycle and get another tree out.

This observation, combined with the cycle and cut properties form the basis of all of the greedy algorithms for MSTs.