Section 12

CSCI E-22

Will Begin Shortly

Graphs: Minimum Spanning Trees

Recall:

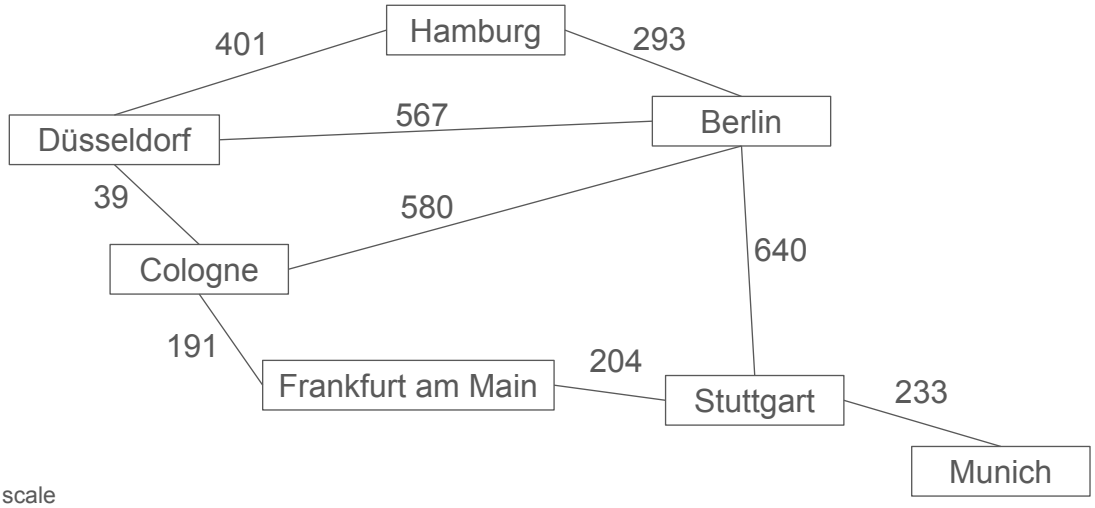
- A *minimum spanning tree* (MST) has the smallest total cost among all possible spanning trees.
  - If all edges have unique costs, there is only one MST. However, if some edges have the same cost, there may be more than one.

Graphs: Minimum Spanning Trees

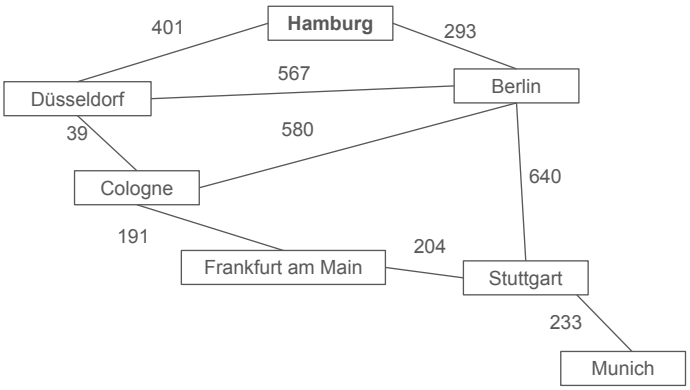
Recall:

- A *minimum spanning tree* (MST) has the smallest total cost among all possible spanning trees.
  - If all edges have unique costs, there is only one MST. However, if some edges have the same cost, there may be more than one.
- One way we can find a MST is ***Prim's MST Algorithm***
  - Use sets A and B, staring with one vertex in set A and all others in set B
  - Select the lowest cost edge that connects set A to set B, and add it to the spanning tree
  - Move the newly connected vertex from set B to set A
  - Repeat

Consider the following weighted, undirected graph representing the distances from city to city by way of the Autobahn in Germany:



*not to scale



Edge Added	Set A	Set B

Remember:

- The MST is the spanning tree with the minimal *total* edge cost.
- It does not necessarily include the minimal cost path between a pair of vertices.

Remember:

- The MST is the spanning tree with the minimal *total* edge cost.
- It does not necessarily include the minimal cost path between a pair of vertices.

It can be helpful to know the shortest path from one vertex to another; how can we figure this out?

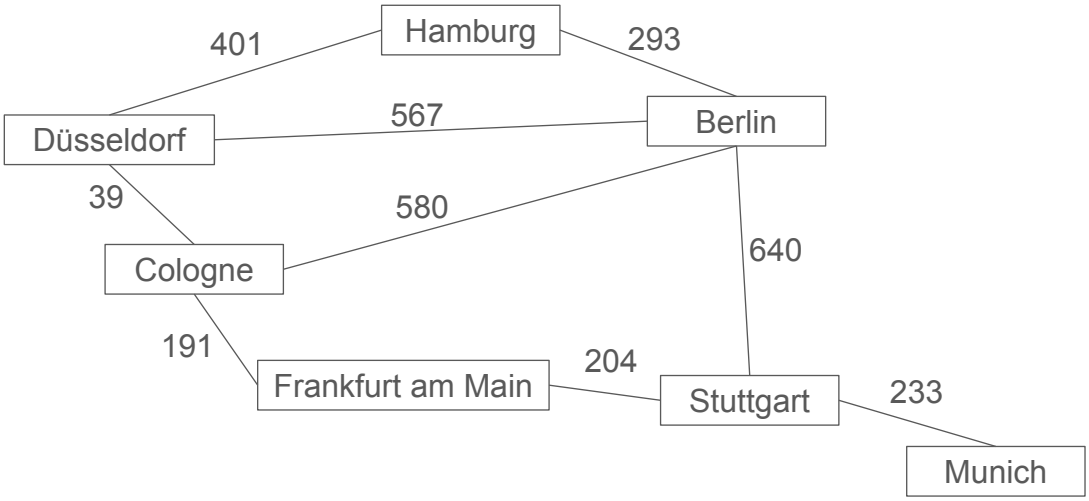
Dijkstra's Algorithm

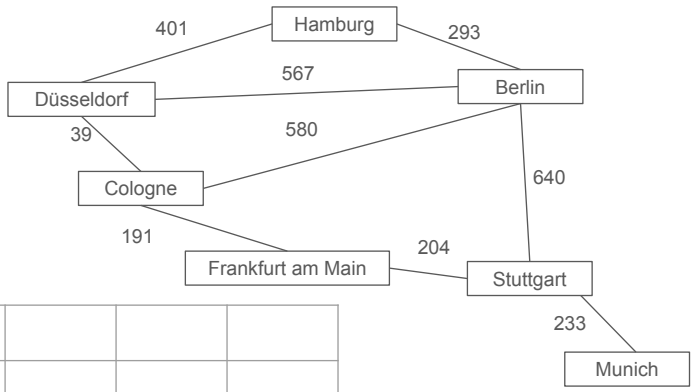
- Allows us to find the shortest path from a vertex v (the origin) to all other vertices that can be reached from v.

How?

- Maintain estimates of the shortest paths from the origin to every vertex (along with their costs)
- Gradually refine these estimates as we traverse the graph

Using our same graph as before, use Dijkstra's shortest path algorithm to obtain the shortest paths starting from Cologne.



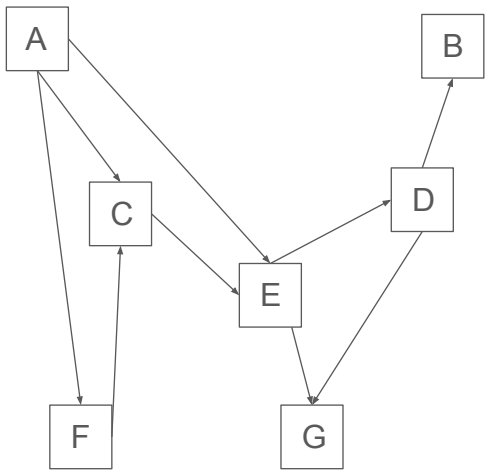


B	∞						
C	**0**						
D	∞						
F	∞						
H	∞						
M	∞						
S	∞						

Topological Sort

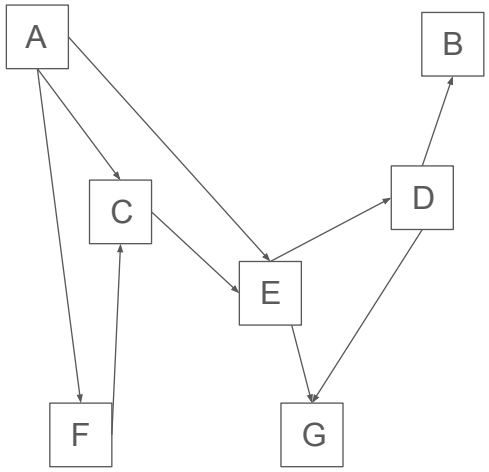
Recall:

- Used to order the vertices in a directed acyclic graph (a DAG).
- It is an ordering of the vertices such that, if there is directed edge from a to b, a comes before b.



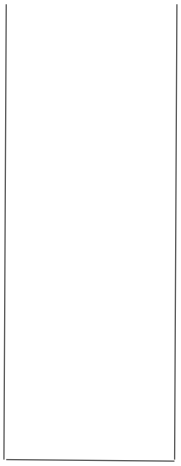
Topological Sort

- S = a stack to hold the vertices
- while there are still unvisited vertices:
  - find a vertex v with no unvisited successors mark v as visited
  - S.push(v)
- return S



Push:

Stack:



*For the purpose of this section, tie-breaker will be what comes first alphabetically