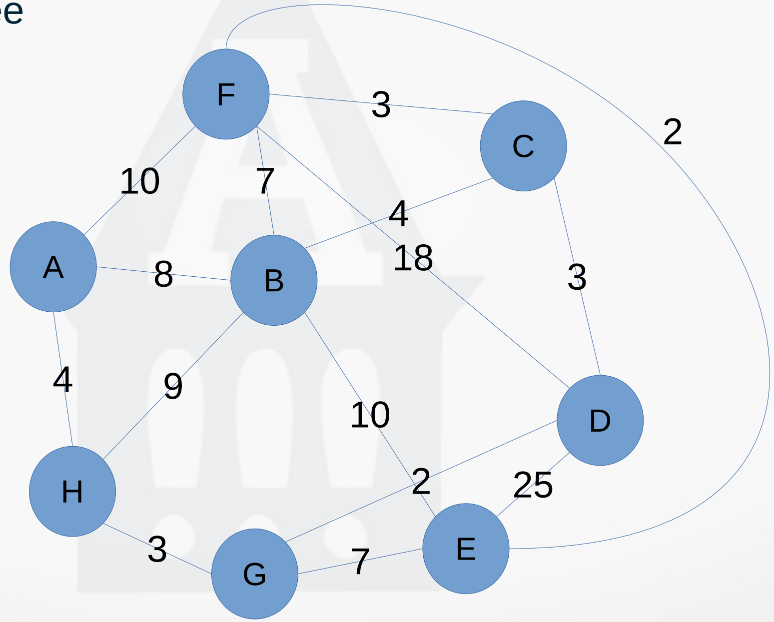
Minimum Spanning Trees

Minimum Spanning Tree

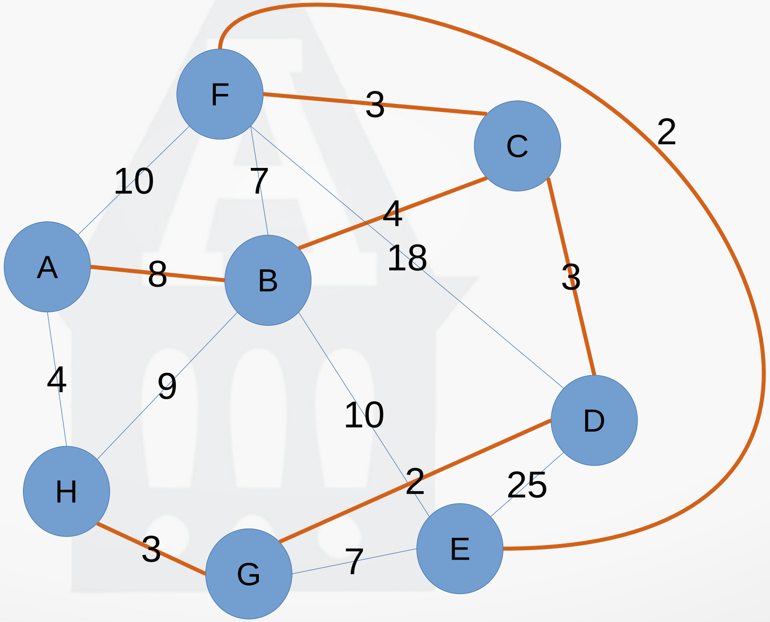
- A Minimum Spanning Tree (MST) is a subgraph of an undirected graph such that the subgraph spans (includes) all nodes, is connected, is acyclic, and has minimum total edge weight
  - Not necessarily a binary tree



Minimum Spanning Tree

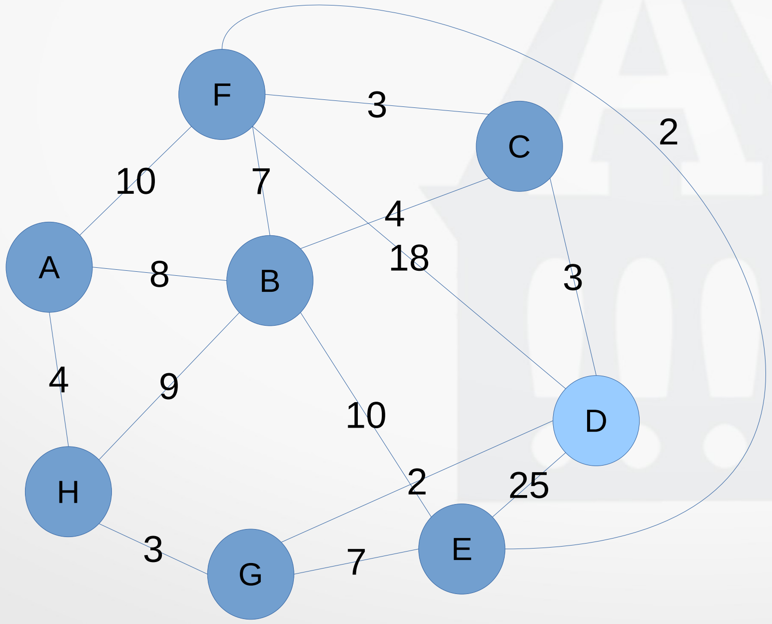
- Let's try to make one
  - It is a subgraph
  - It includes all nodes
  - It is connected
  - It is acyclic
  - It is ***not*** min edge weight

- We need an algorithm



Prim's Algorithm – Walkthrough

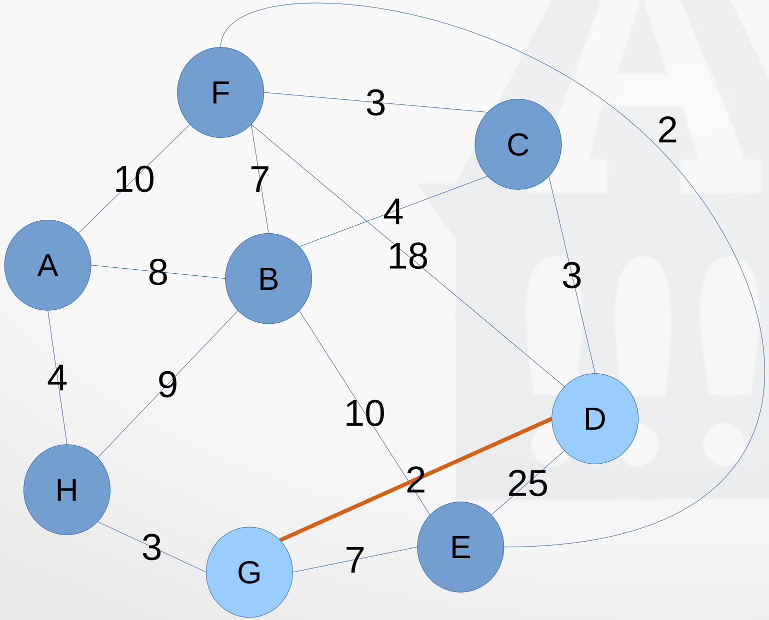
- Similar to Dijkstra's algorithm, but not a shortest path algorithm
  - Pick a starting node, can be any node, doesn't matter
  - Add all nodes reachable from this node to a priority queue



Edge	Weight
DG	2
DC	3
DF	18
DE	25

Prim's Algorithm – Walkthrough

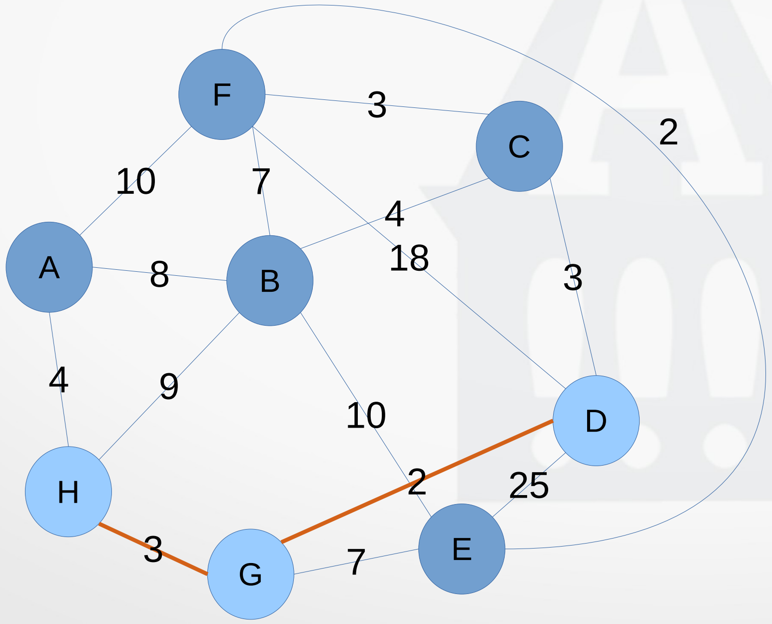
- Similar to Dijkstra's algorithm, but not a shortest path algorithm
  - Pull min distance off: D to G
  - Add additional reachable nodes



Edge	Weight
GH	3
DC	3
GE	7
DF	18
DE	25

Prim’s Algorithm – Walkthrough

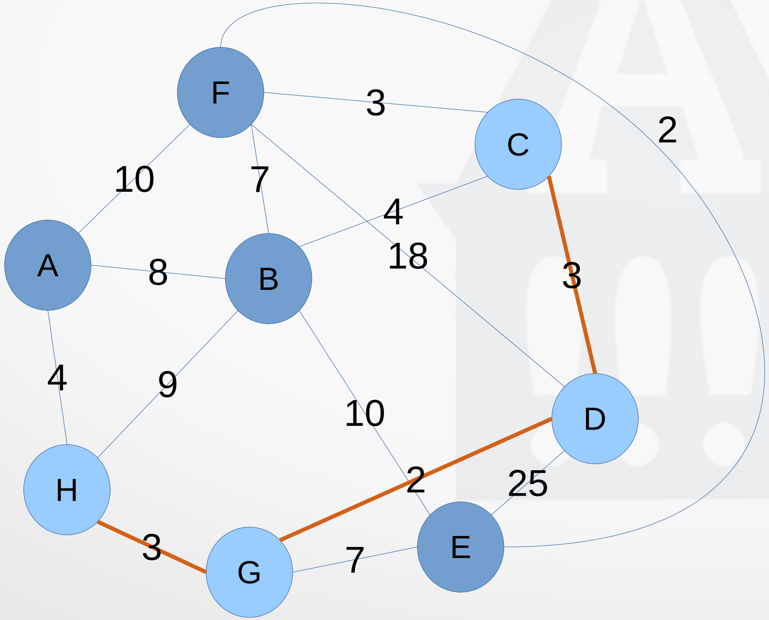
- Similar to Dijkstra’s algorithm, but not a shortest path algorithm
  - Pull min distance off: G to H
  - Add additional reachable nodes



Edge	Weight
DC	3
HA	4
GE	7
HB	9
DF	18
DE	25

Prim's Algorithm – Walkthrough

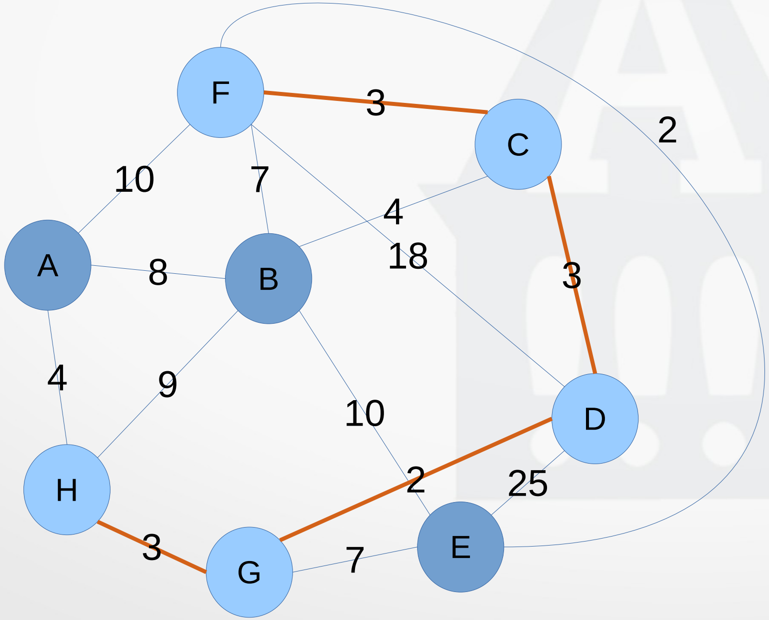
- Similar to Dijkstra's algorithm, but not a shortest path algorithm
  - Pull min distance off: D to C
  - Add additional reachable nodes



Edge	Weight
CF	3
CB	4
HA	4
GE	7
HB	9
DF	18
DE	25

Prim's Algorithm – Walkthrough

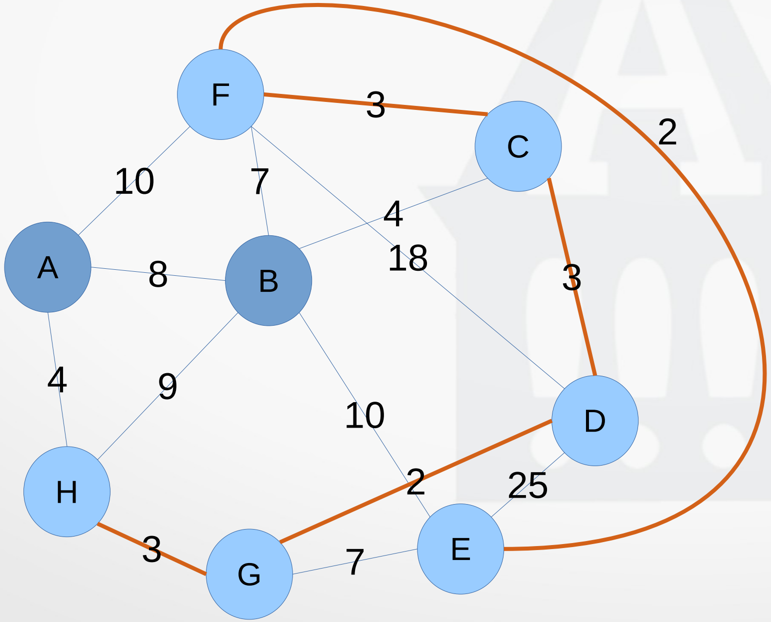
- Similar to Dijkstra's algorithm, but not a shortest path algorithm
  - Pull min distance off: C to F
  - Add additional reachable nodes



Edge	Weight
FE	2
CB	4
HA	4
FB	7
GE	7
HB	9
FA	10
DF	18

Prim's Algorithm – Walkthrough

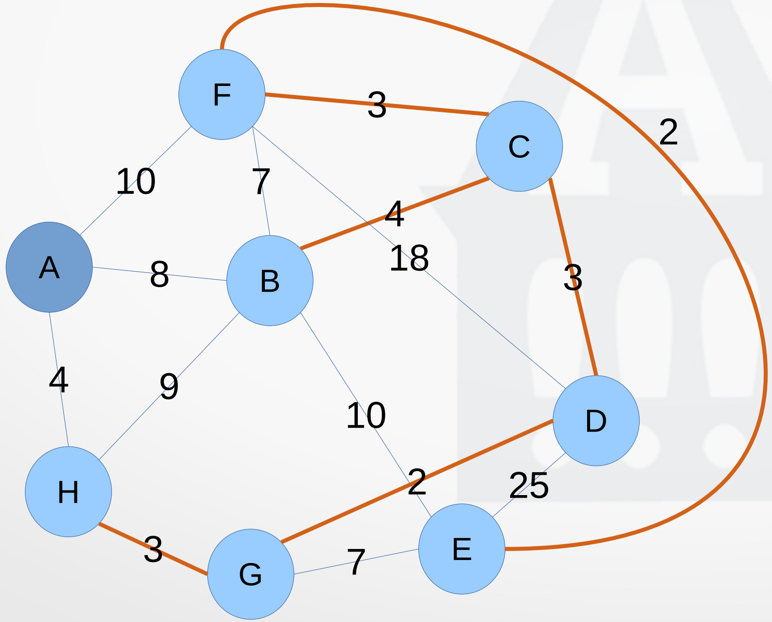
- Similar to Dijkstra's algorithm, but not a shortest path algorithm
  - Pull min distance off: F to E
  - Add additional reachable nodes



Edge	Weight
CB	4
HA	4
FB	7
GE	7
HB	9
EB	10
FA	10
DF	18

Prim’s Algorithm – Walkthrough

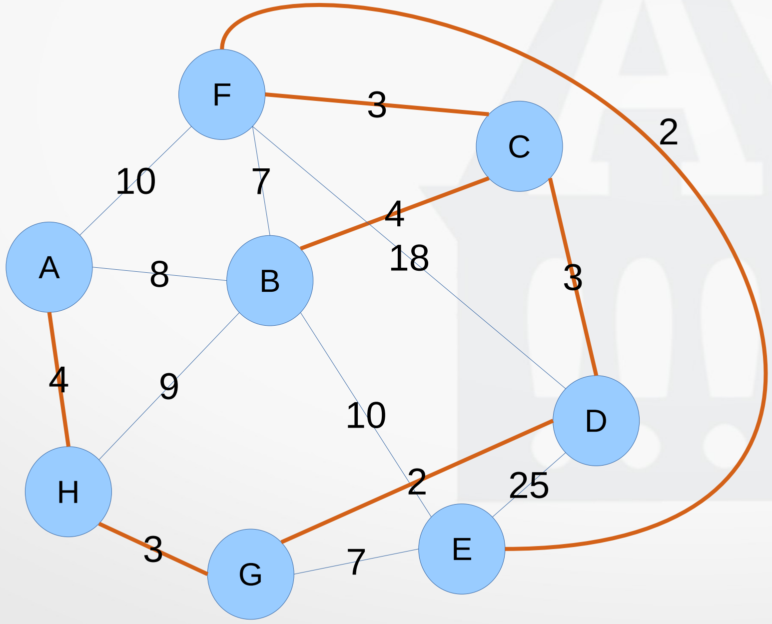
- Similar to Dijkstra’s algorithm, but not a shortest path algorithm
  - Pull min distance off: C to B
  - Add additional reachable nodes



Edge	Weight
HA	4
FB	7
GE	7
BA	8
HB	9
EB	10
FA	10
DF	18

Prim’s Algorithm – Walkthrough

- Similar to Dijkstra’s algorithm, but not a shortest path algorithm
  - Pull min distance off: H to A
  - Add additional reachable nodes



Edge	Weight
FB	7
GE	7
BA	8
HB	9
EB	10
FA	10
DF	18

Prim’s Algorithm – Characteristics

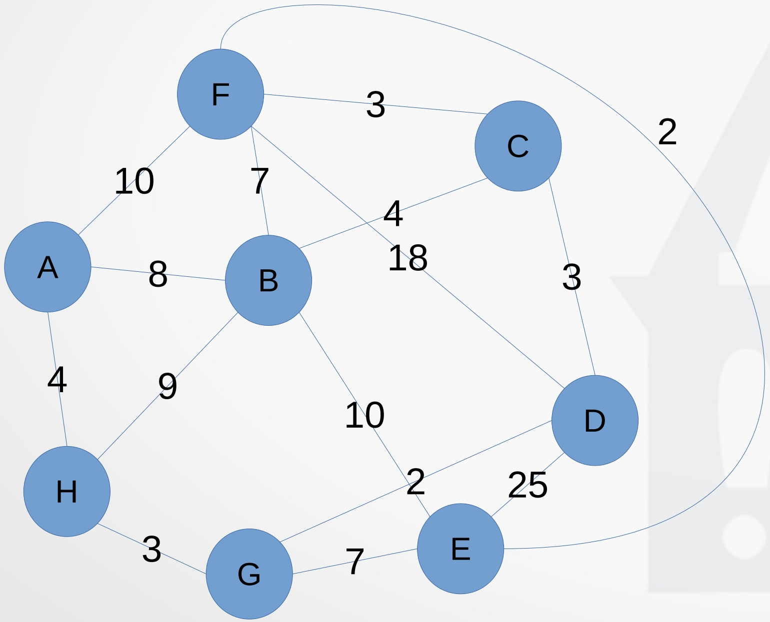
- Works with undirected graphs
- Works with weighted and unweighted graphs
  - But more interesting when edges are weighted
- Greedy algorithm, but produces optimal solutions
- Complexity
  - Worst case: all edges go on the queue, all edges come off the queue
    - e inserts: e (on average insert is O(1), but O(log(n)) worst case)
    - e removals: log(e)
  - O(n log(n))

Kruskal’s Algorithm

- Similar to Prim’s
- Doesn’t have concept of *in the tree* and *not in the tree* (like Prim’s)
- Tree grows in different places and are then joined (it’s actually a union-find algorithm)
- Two steps
  - Sort edges by increasing edge weight
  - Select the first |V| - 1 edges that do not generate a cycle

Kruskal’s Algorithm – Walkthrough

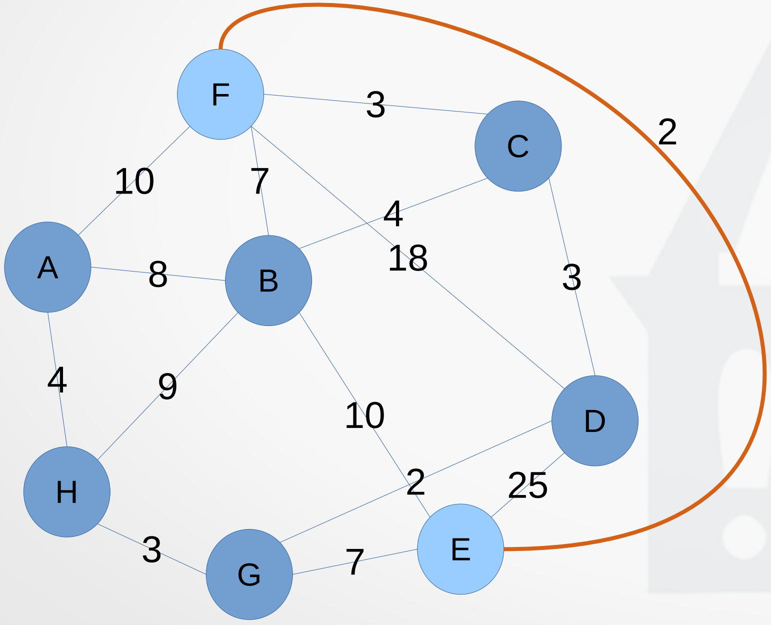
- Start by sorting edges by weight



edge	d_v	group
FE	2	
DG	2	
FC	3	
CD	3	
GH	3	
AH	4	
BC	4	
FB	7	
GE	7	
AB	8	
BH	9	
FA	10	
BE	10	
FD	18	
DE	25	

Kruskal’s Algorithm – Walkthrough

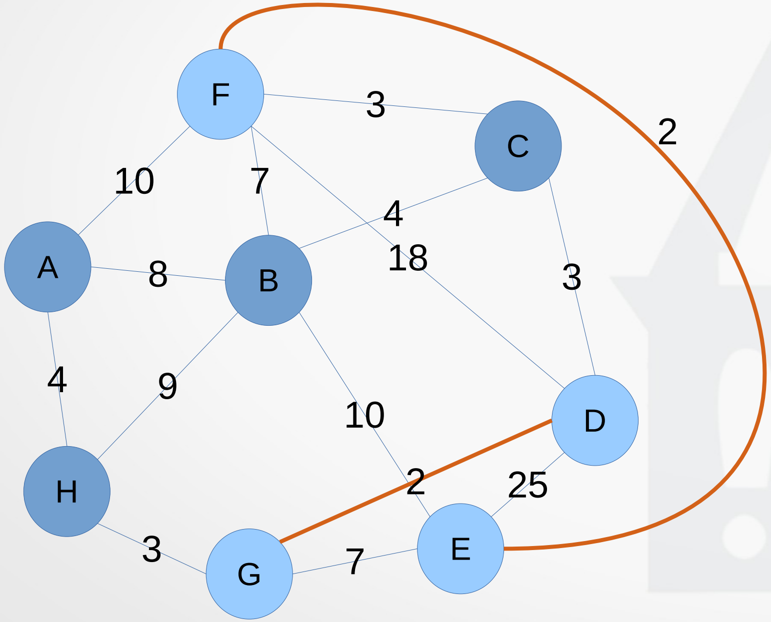
- Then pick edge with smallest weight



edge	d_v	group
FE	2	t1
DG	2	
FC	3	
CD	3	
GH	3	
AH	4	
BC	4	
FB	7	
GE	7	
AB	8	
BH	9	
FA	10	
BE	10	
FD	18	
DE	25	

Kruskal's Algorithm – Walkthrough

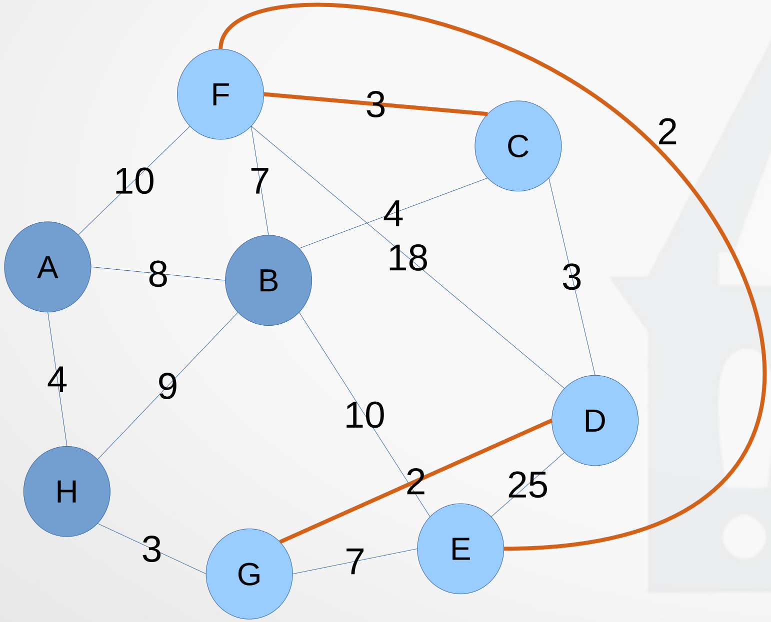
- Then pick edge with smallest weight



edge	d_v	group
FE	2	t1
DG	2	t2
FC	3	
CD	3	
GH	3	
AH	4	
BC	4	
FB	7	
GE	7	
AB	8	
BH	9	
FA	10	
BE	10	
FD	18	
DE	25	

Kruskal’s Algorithm – Walkthrough

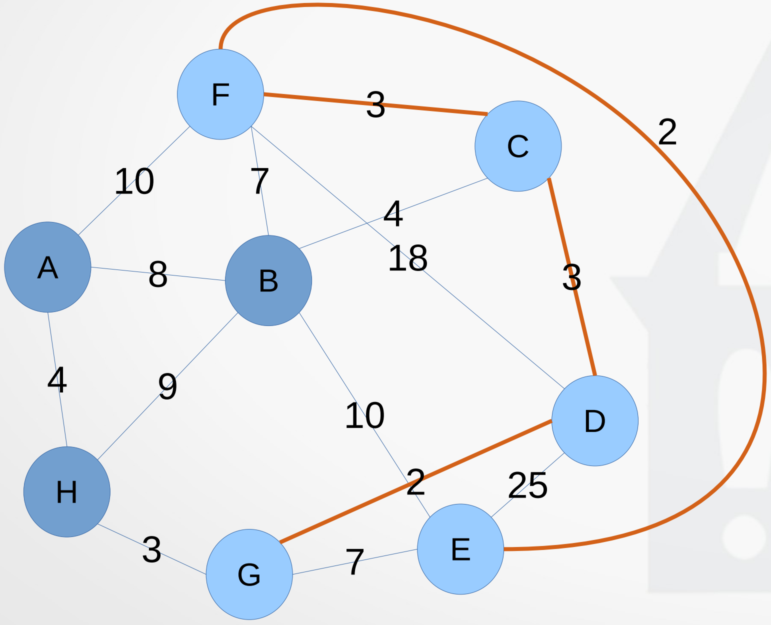
- Then pick edge with smallest weight



edge	d_v	group
FE	2	t1
DG	2	t2
FC	3	t1
CD	3	
GH	3	
AH	4	
BC	4	
FB	7	
GE	7	
AB	8	
BH	9	
FA	10	
BE	10	
FD	18	
DE	25	

Kruskal's Algorithm – Walkthrough

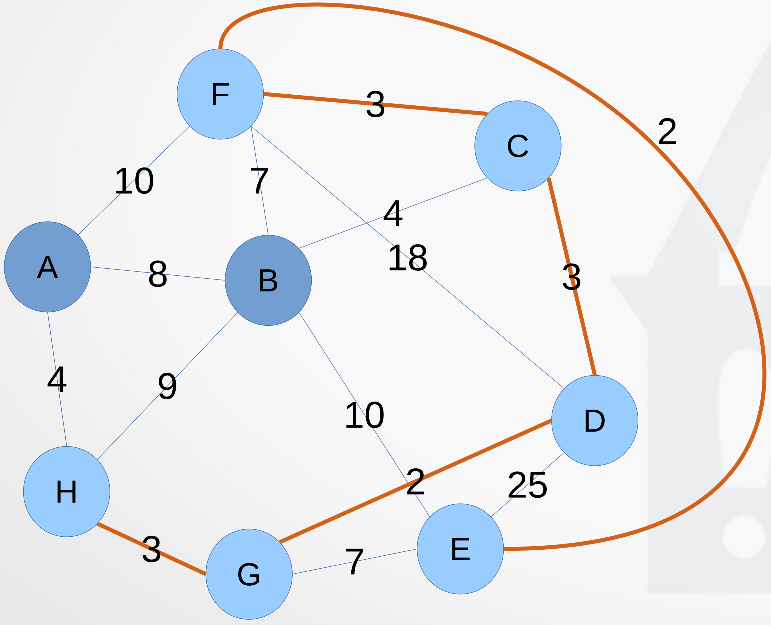
- Then pick edge with smallest weight



edge	d_v	group
FE	2	t1
DG	2	t1
FC	3	t1
CD	3	t1
GH	3	
AH	4	
BC	4	
FB	7	
GE	7	
AB	8	
BH	9	
FA	10	
BE	10	
FD	18	
DE	25	

Kruskal’s Algorithm – Walkthrough

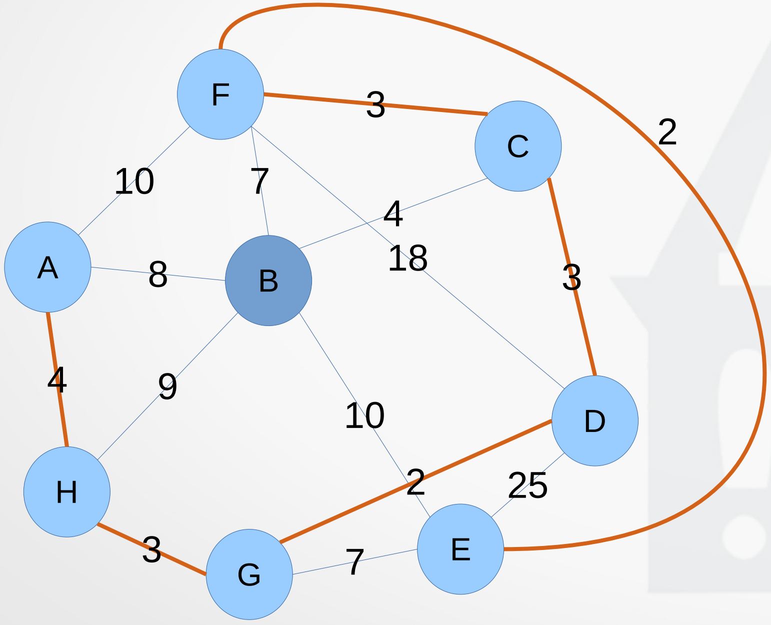
- Then pick edge with smallest weight



edge	d_v	group
FE	2	t1
DG	2	t1
FC	3	t1
CD	3	t1
GH	3	t1
AH	4	
BC	4	
FB	7	
GE	7	
AB	8	
BH	9	
FA	10	
BE	10	
FD	18	
DE	25	

Kruskal's Algorithm – Walkthrough

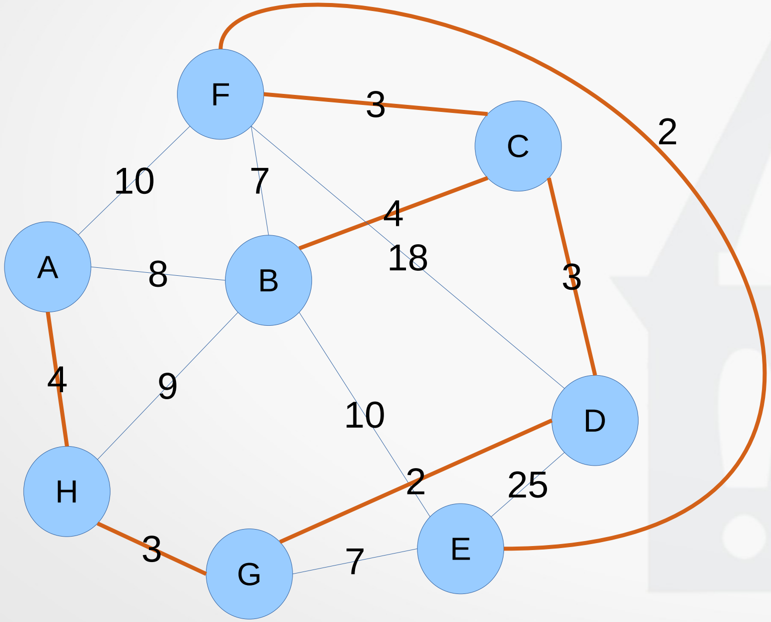
- Then pick edge with smallest weight



edge	d_v	group
FE	2	t1
DG	2	t1
FC	3	t1
CD	3	t1
GH	3	t1
AH	4	t1
BC	4	
FB	7	
GE	7	
AB	8	
BH	9	
FA	10	
BE	10	
FD	18	
DE	25	

Kruskal's Algorithm – Walkthrough

- Then pick edge with smallest weight...**all done!**



edge	d_v	group
FE	2	t1
DG	2	t1
FC	3	t1
CD	3	t1
GH	3	t1
AH	4	t1
BC	4	t1
FB	7	
GE	7	
AB	8	
BH	9	
FA	10	
BE	10	
FD	18	
DE	25	

Kruskal's Algorithm – Complexity

- Sort the edges: e log(e)
- Pick edges, worst case find(e), find(tree), union: these are constant operations
  - find(e) : to find the group of the edge we just removed
  - find(tree) : to find the group of the minimum spanning tree
  - union : to join the edge and tree
- O(n log(n))

Minimum Spanning Tree – Applications

- Cluster analysis – Build MST, cut most expensive edges
- Construction of computer networks, or trees for broadcast model over a computer network
  - Design of any kind of network that needs to connect things: roads, utilities, computer
- Image registration and segmentation
- Feature extraction in computer vision
- Handwriting recognition
- Approximation of the TSP
- Many others