Tree Algorithms

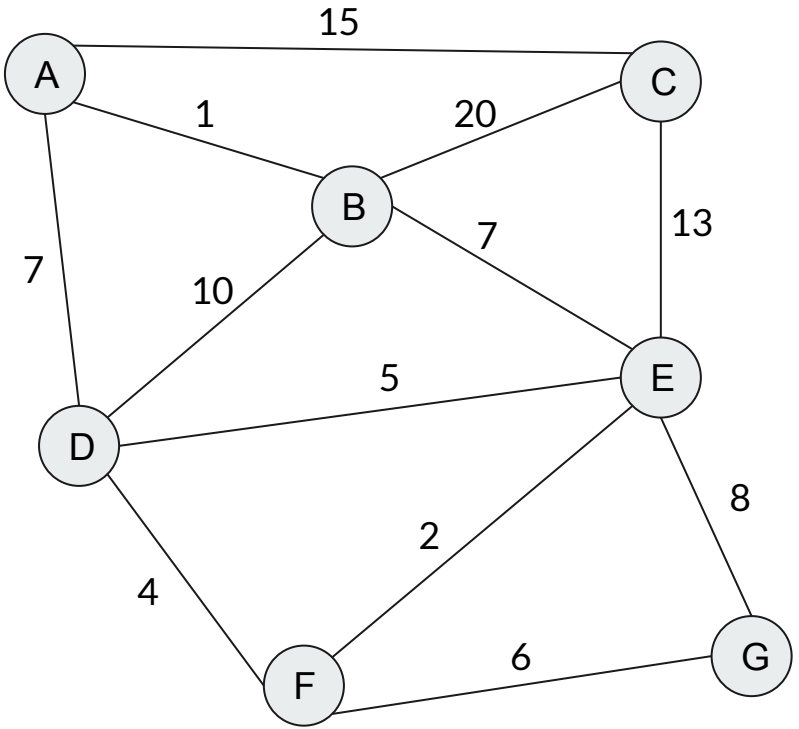
Minimum Spanning Trees

- Spanning trees for undirected, weighted graphs
- Connects all vertices in lowest total weight
- Very important practical problem
- Assuming non-negative weight

Kruskal's Algorithm

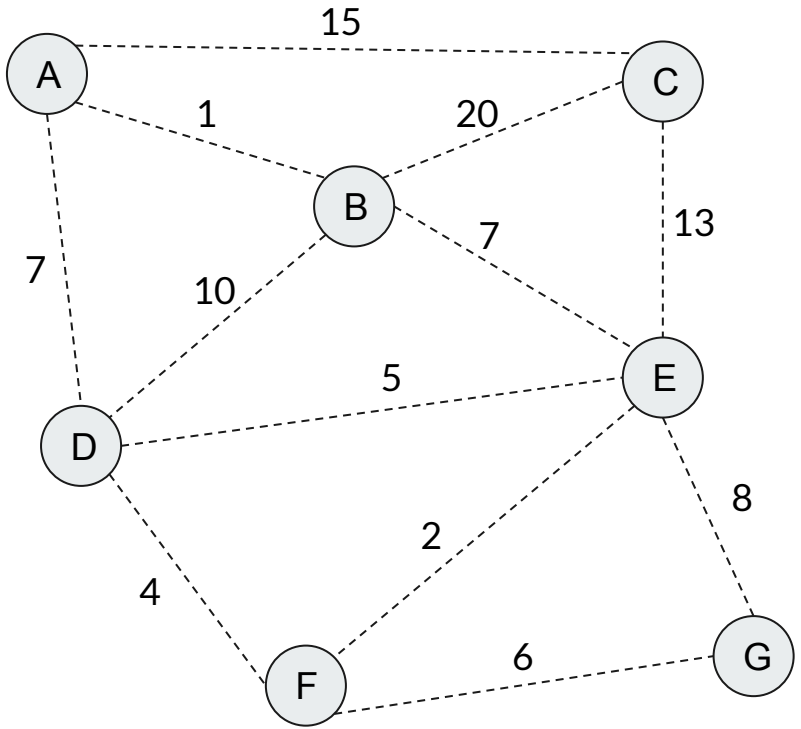
procedure KRUSHKAL(Graph $G = \{V(G), E(G), W(G)\}$)
▷ Note: $W(G)$ is a numeric *weight* for each edge in $E(G)$
$V(T) \leftarrow V(G)$ ▷ Spanning tree T will have all vertices of G
$E(T) \leftarrow \emptyset$ ▷ Edge set of T initially null
sort $W(G)$ and correspondingly $E(G)$ by nondecreasing values in $W(G)$
for all $w \in W(G), e \in E(G)$ **do**
    if numComponents$(T + e) <$ numComponents(T) **then**
        $E(T) \leftarrow E(T) + e$
    if numComponents$(T) = 1$ **then**
        break
return T

Kruskal's: Example



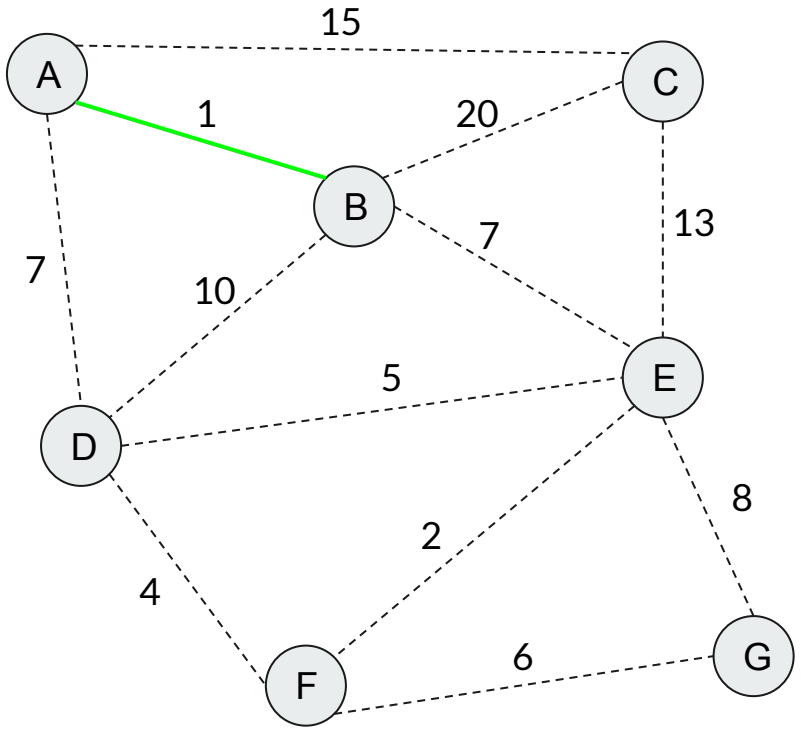
Kruskal's: Example

- Start by removing all edges



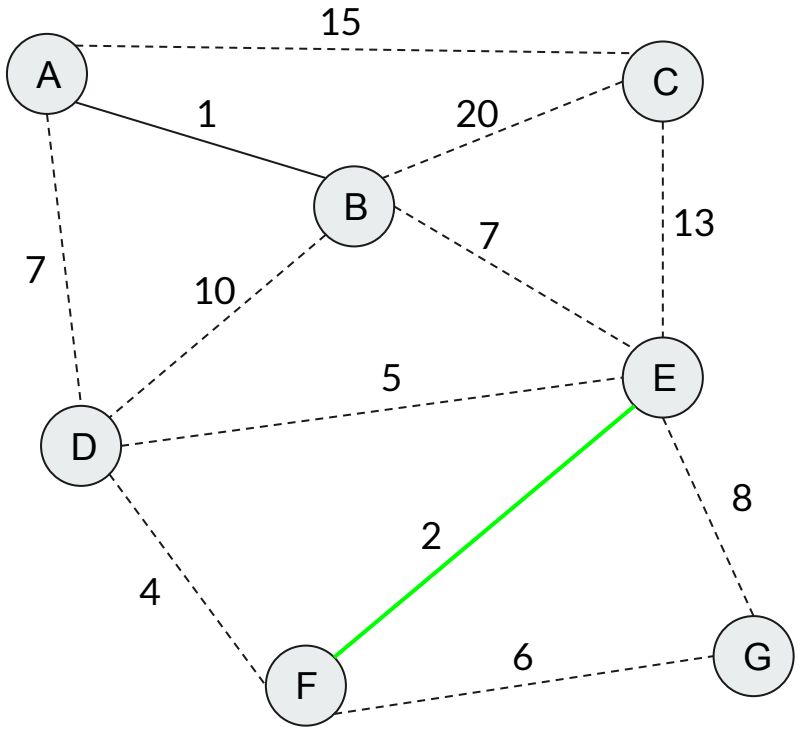
Kruskal's: Example

- Sort edges by weight, add from lowest to highest



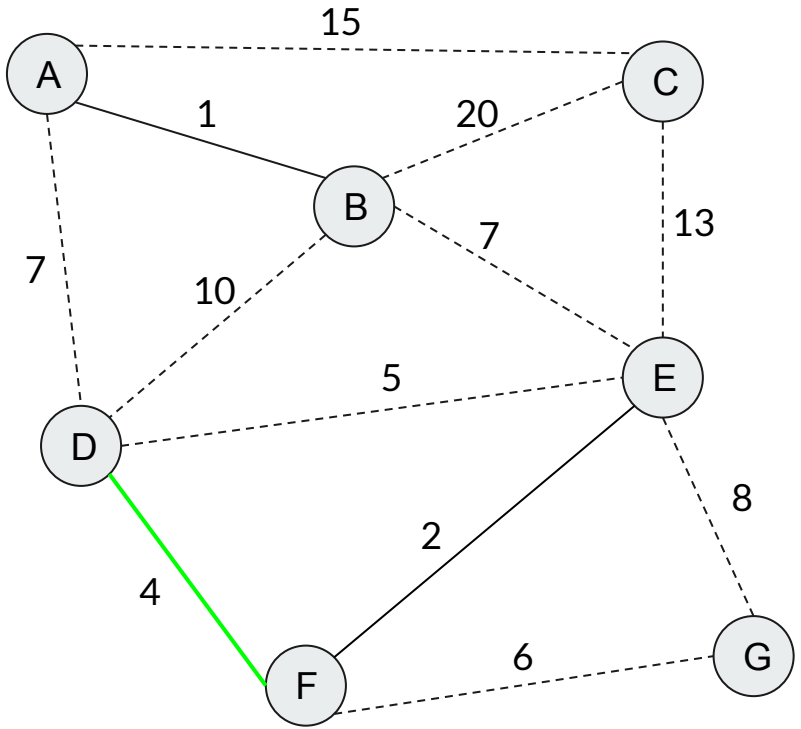
Kruskal's: Example

- Continue adding edges until spanning tree



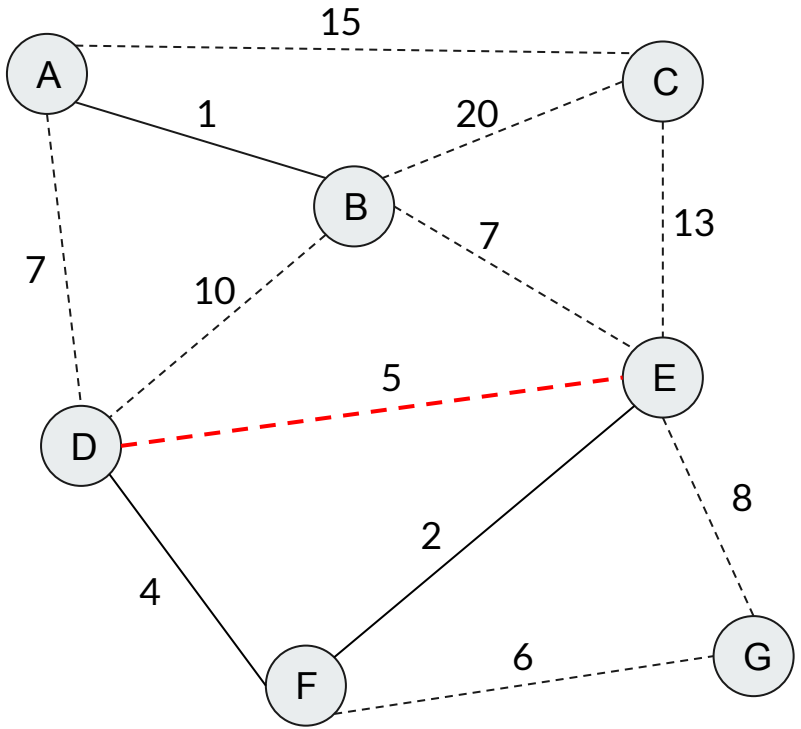
Kruskal's: Example

- Continue adding edges until spanning tree



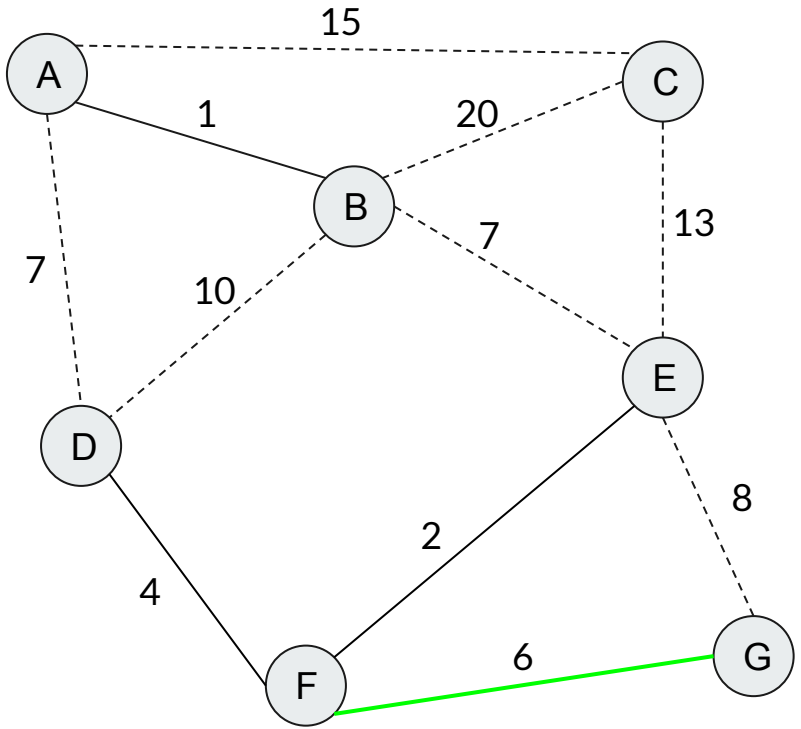
Kruskal's: Example

- Discard an edge if it doesn't decrease number of components



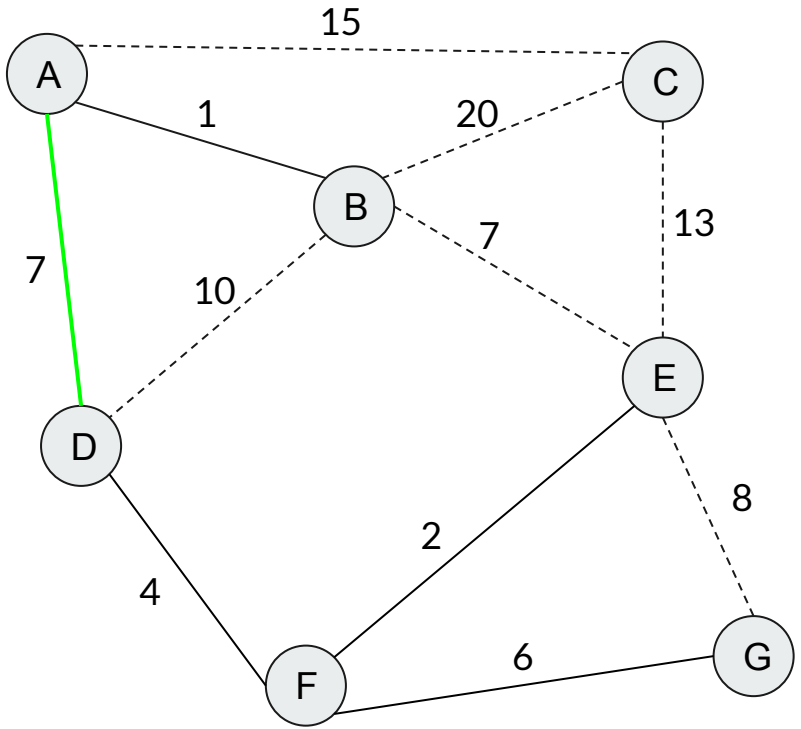
Kruskal's: Example

- Continue adding edges until spanning tree



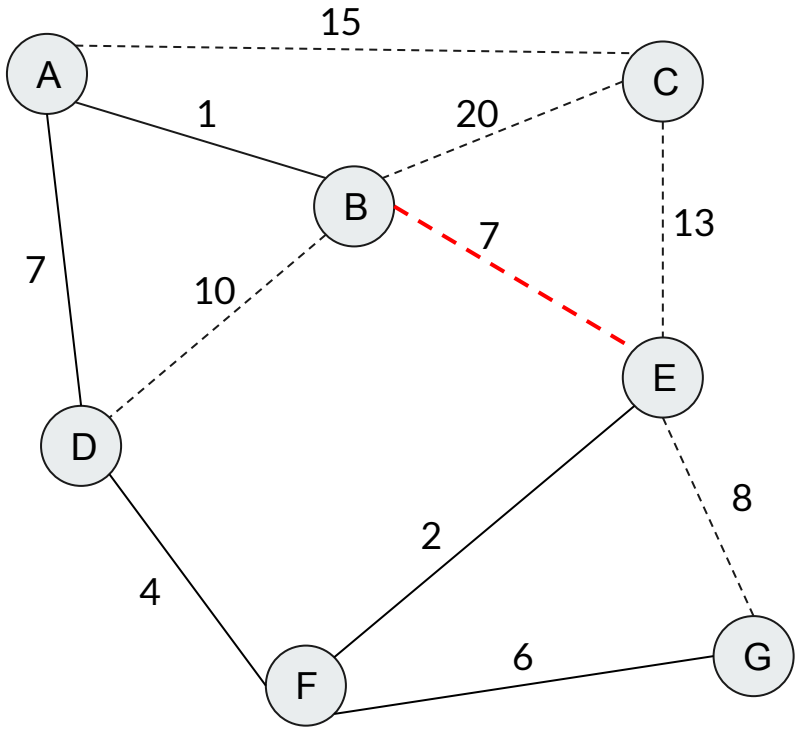
Example

- Break ties arbitrarily



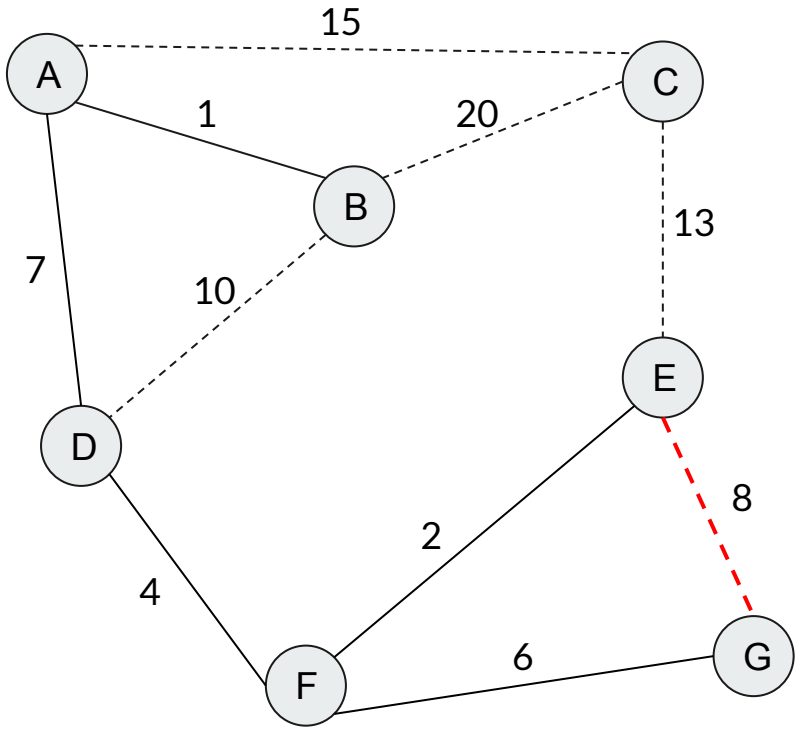
Kruskal's: Example

- Discard an edge if it doesn't decrease number of components



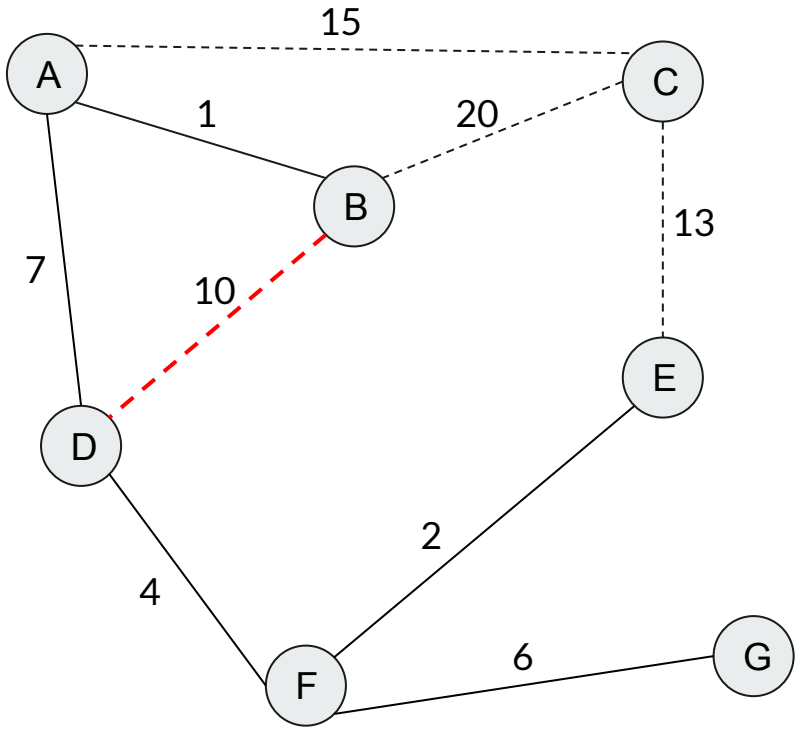
Kruskal's: Example

- Discard an edge if it doesn't decrease number of components



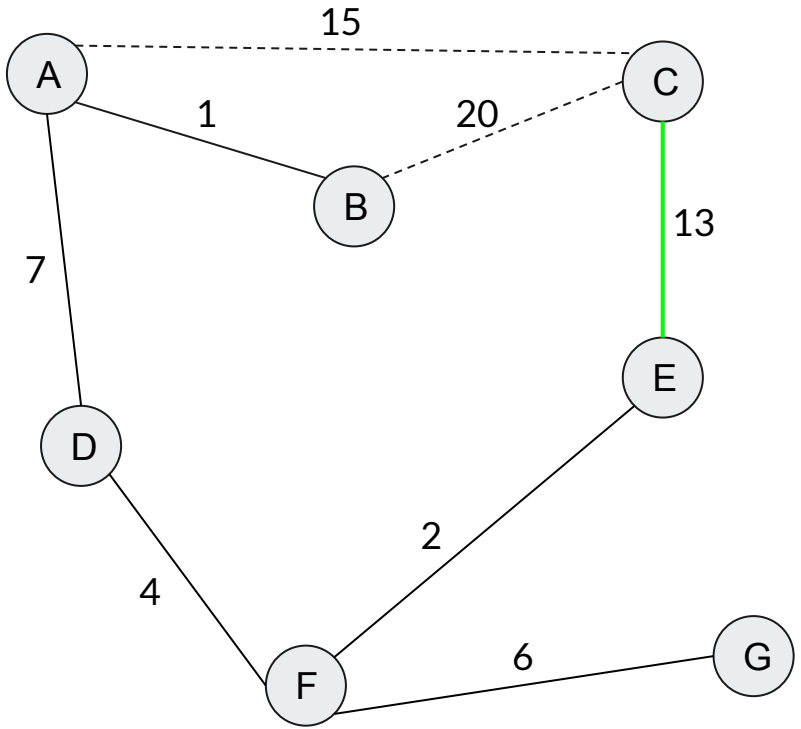
Kruskal's: Example

- Discard an edge if it doesn't decrease number of components



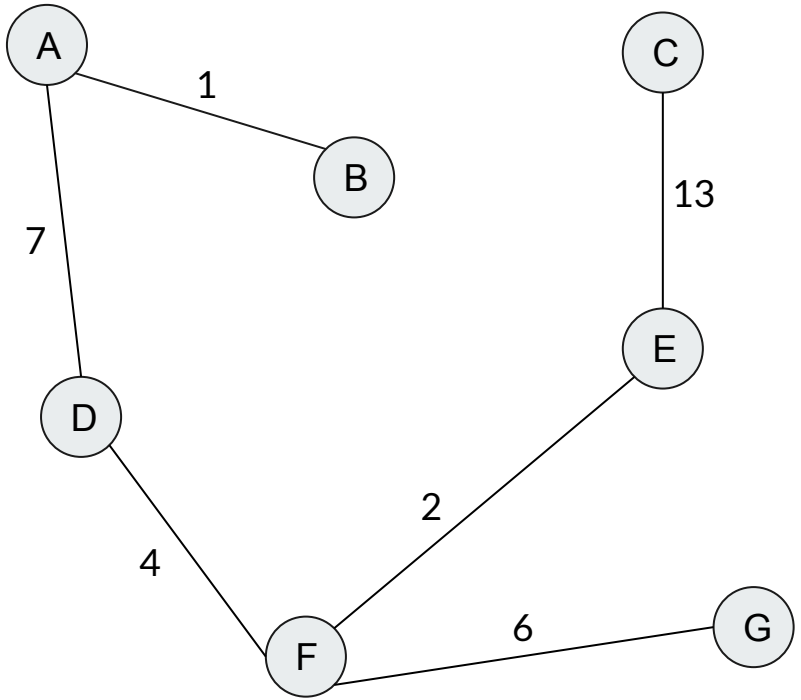
Kruskal's: Example

- Continue adding edges until spanning tree



Kruskal's: Example

- Return spanning tree



Kruskal's Algorithm Proof

Prove: Kruskal's algorithm generates a minimum spanning tree (Assume G is connected)

First Proof: Kruskal's algorithm produces a spanning tree

Let H be the subgraph of G returned by Kruskal's algorithm. H has no cycles, because any edge that would create a cycle will not be added. H cannot be disconnected, because any edge that would connect multiple components of H would be added before returning. Therefore, H is a spanning tree.

Kruskal's Algorithm Proof

Second Proof: If F is the set of edges chosen at any stage of the algorithm, there exists a minimum spanning tree that contains F.

Basis step: At the first step there are no edges, so all minimum spanning trees contain F.

Kruskal's Algorithm Proof

Inductive Step:

Let T be the tree that contains F(n), and let e be the new edge added. If e is in T, there exists a minimum spanning tree containing F(n+1). If not, T + e must have a cycle. In this cycle, there must exist edge f not in F, because e does not create a cycle in F. Since f is not in F, it must not have yet been considered in the algorithm. This means it has a weight greater than or equal to e. Therefore, T-f+e is also a minimum spanning tree, so there still exists a spanning tree containing F(n+1).

If at all steps, F is part of a minimum spanning tree, and F becomes a spanning tree, F must become a minimum spanning tree.

Prim’s Algorithm

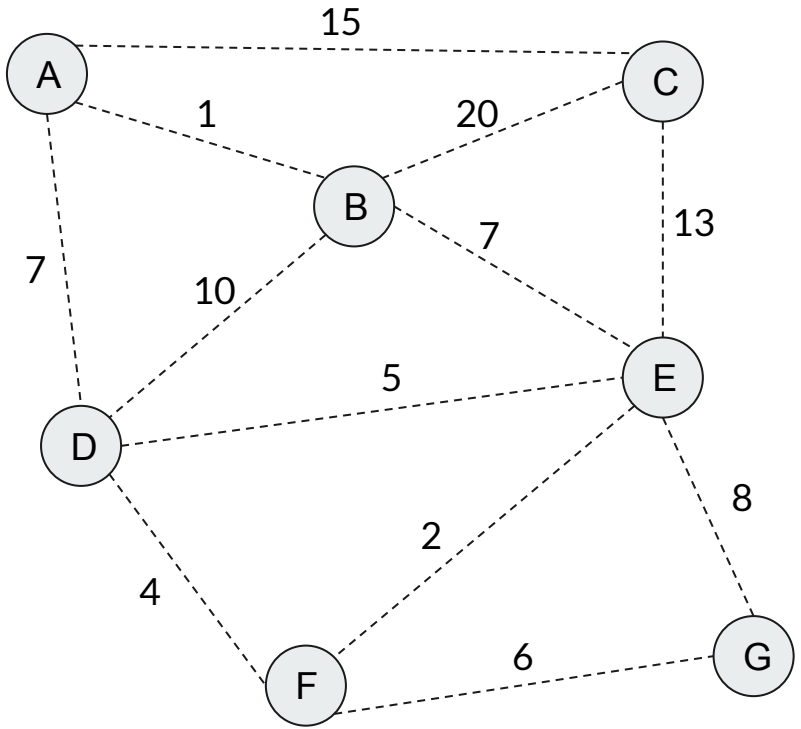
- Also generates minimum spanning trees
- Same order notation as Kruskal’s algorithm
- Works by continuously building the tree as opposed to picking edges

Prim's Algorithm

procedure PRIM(Graph $G = \{V(G), E(G), W(G)\}$)
▷ Note: $W(G)$ is a numeric *weight* for each edge in $E(G)$
$V(T) \leftarrow \emptyset$ ▷ Spanning tree's vertices initially null
$E(T) \leftarrow \emptyset$ ▷ Edge set of T initially null
$V(T) \leftarrow \text{randomSelect}(V(G))$ ▷ Randomly select one vertex from G
while $V(T) \neq V(G)$ **do**
$e \leftarrow \min(W(u, v)) \in E(G) : u \in V(T), v \notin V(T)$
▷ Minimum weight edge in G with only one vertex in T
$E(T) \leftarrow E(T) + e$
$V(T) \leftarrow V(T) + v$
return T

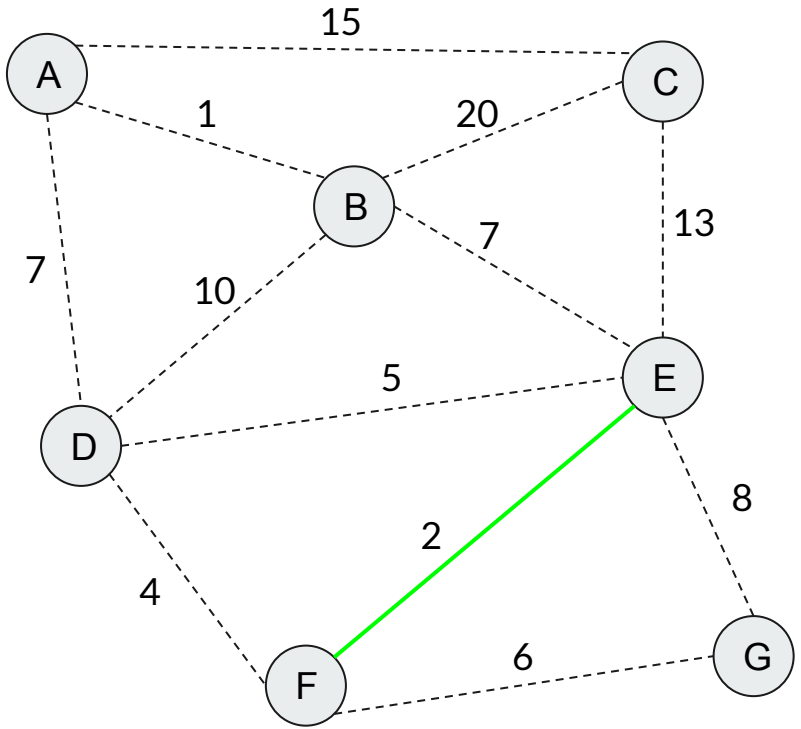
Prim's: Example

- Randomly start with vertex F



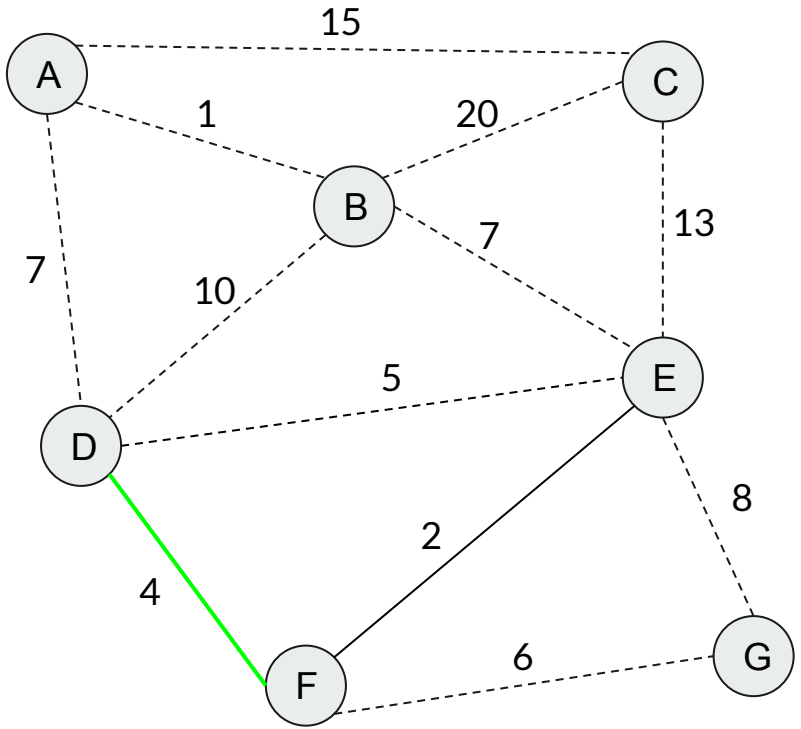
Prim's: Example

- Find the minimum weight edge that adds a new vertex to the tree



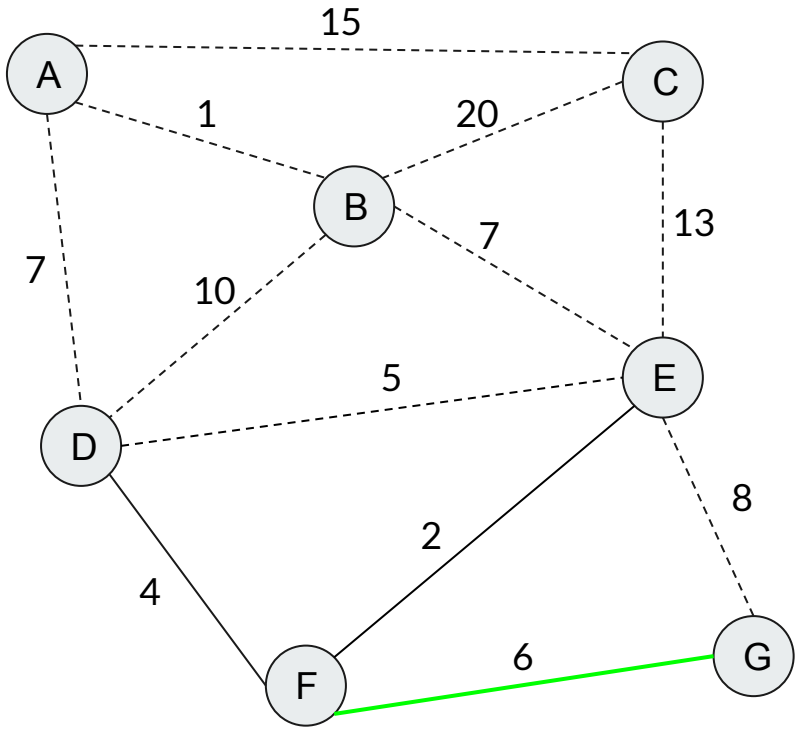
Prim's: Example

- Find the minimum weight edge that adds a new vertex to the tree



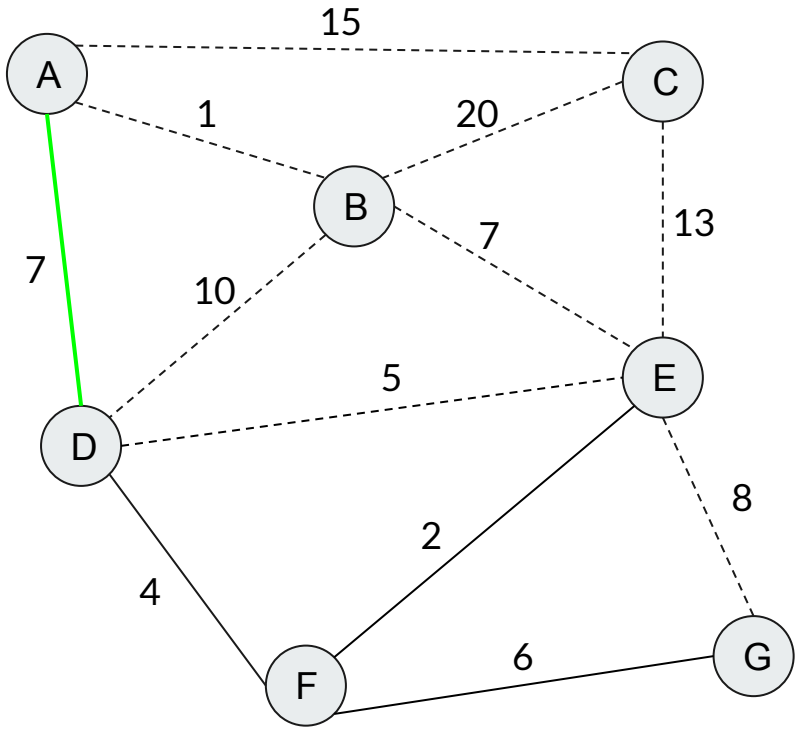
Prim's: Example

- Find the minimum weight edge that adds a new vertex to the tree



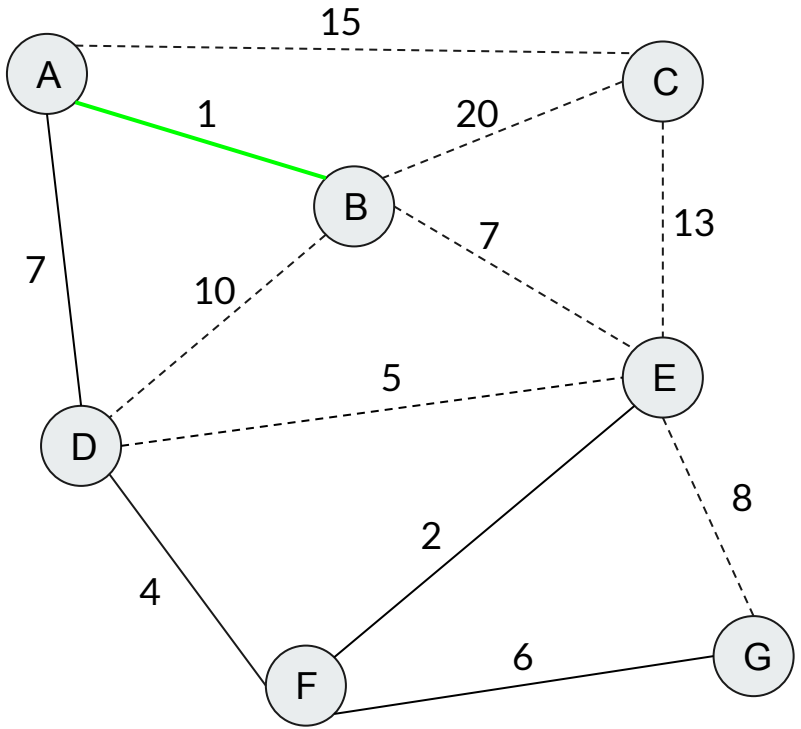
Prim's: Example

- Again, break ties arbitrarily



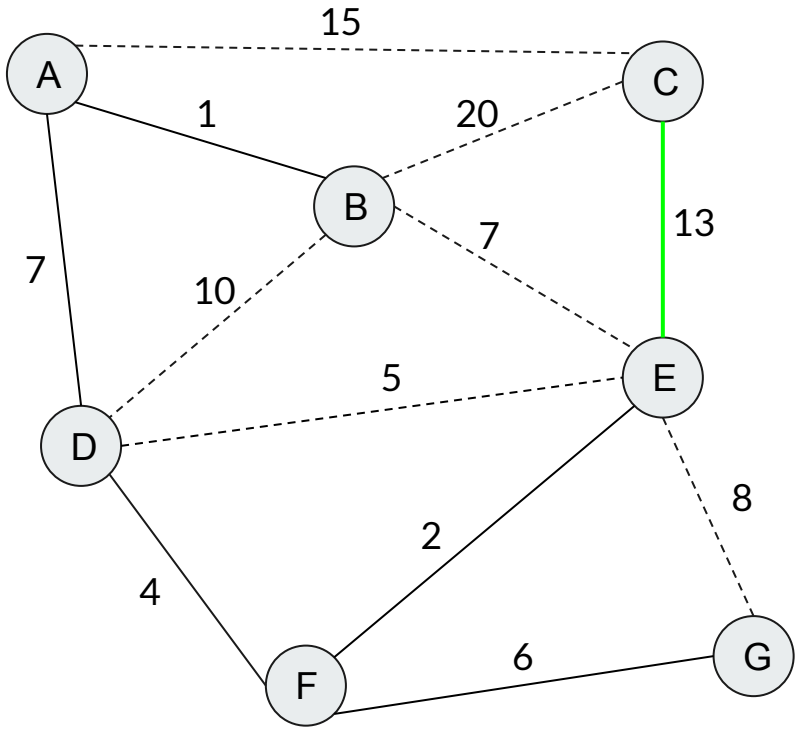
Prim's: Example

- Find the minimum weight edge that adds a new vertex to the tree



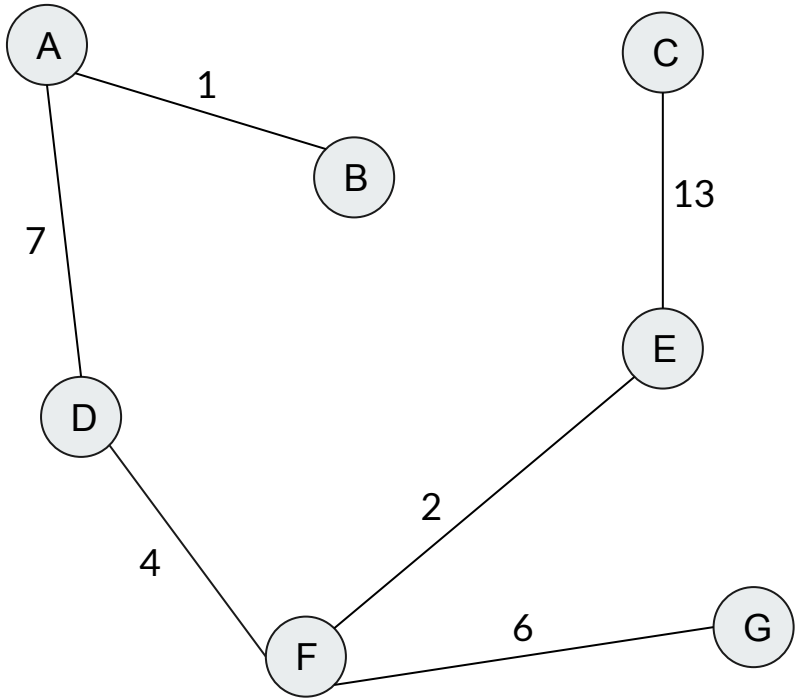
Prim's: Example

- Find the minimum weight edge that adds a new vertex to the tree



Prim's: Example

- Return spanning tree



Shortest-Paths

- Find shortest paths in weighted graph G from vertex u to all other vertices
- Again assume connected undirected weighted graph, no negative weights
- Also important practical problem

Dijkstra's Algorithm

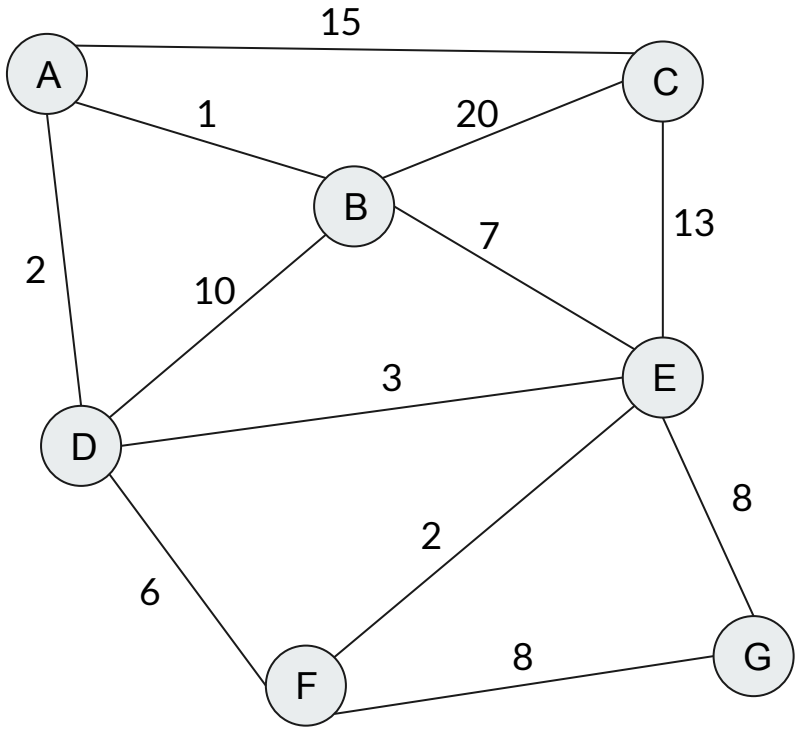
procedure DIJKSTRA(Graph $G = \{V(G), E(G), W(G)\}$, vertex u)
▷ Finding all distances from u
  for all $v \in V(G)$ **do**
    $D(v) \leftarrow \infty$ ▷ Distances from u
  $D(u) \leftarrow 0$
  $S \leftarrow V(G)$ ▷ Unvisited set
  while $S \neq \emptyset$ **do**
    $w \leftarrow \min(D(v), v \in S)$
      ▷ Current vertex considered has minimum distance in unvisited set
    for all $x \in N(w)$, $x \in S$ **do**
      $t \leftarrow W(w, x)$ ▷ Weight of edge between w and x
      if $D(w) + t < D(x)$ **then**
        $D(x) \leftarrow D(w) + t$
    $S \leftarrow S - w$ ▷ Remove w from unvisited set
  return D

Dijkstra's: Example

Finding shortest paths from F

S = {A,B,C,D,E,F,G}

V	D(v)
A	∞
B	∞
C	∞
D	∞
E	∞
F	0
G	∞

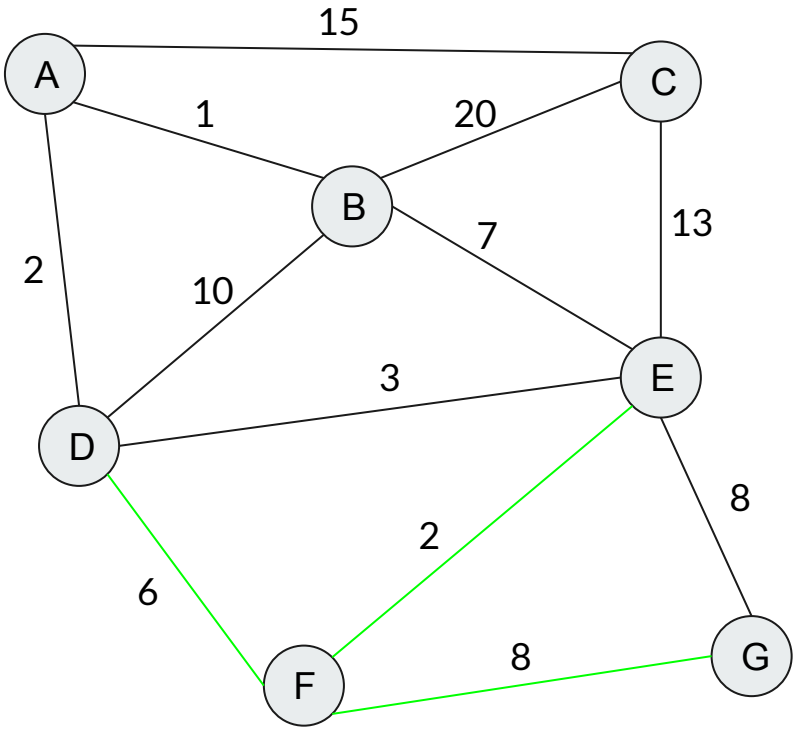


Dijkstra's: Example

Visiting F

S = {A,B,C,D,E,G}

V	D(v)
A	∞
B	∞
C	∞
D	6
E	2
F	0
G	8

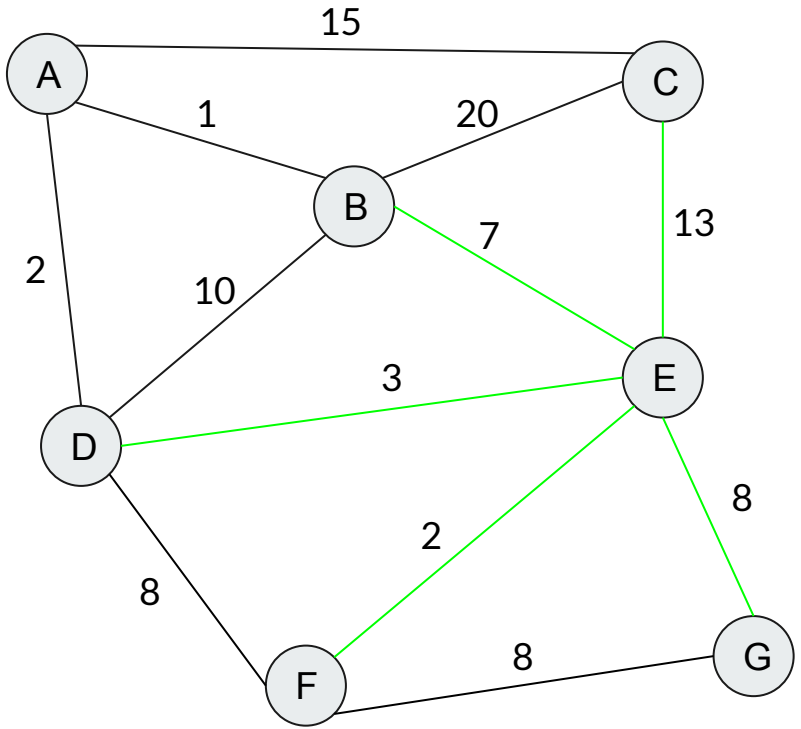


Dijkstra's: Example

Visiting E

S = {A,B,C,D,G}

V	D(v)
A	∞
B	9
C	15
D	5
E	2
F	0
G	8

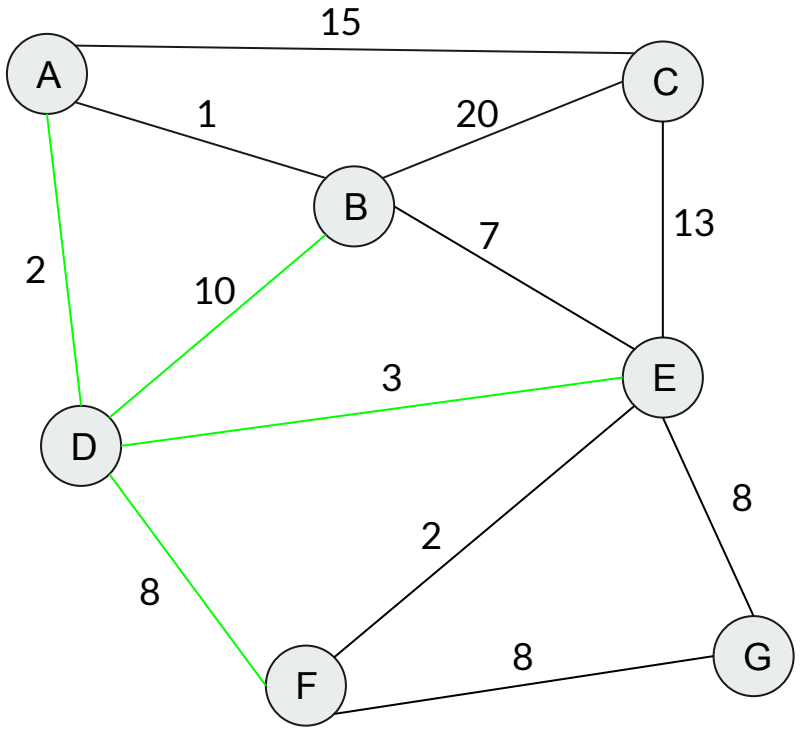


Dijkstra's: Example

Visiting D

S = {A,B,C,G}

V	D(v)
A	7
B	9
C	15
D	5
E	2
F	0
G	8

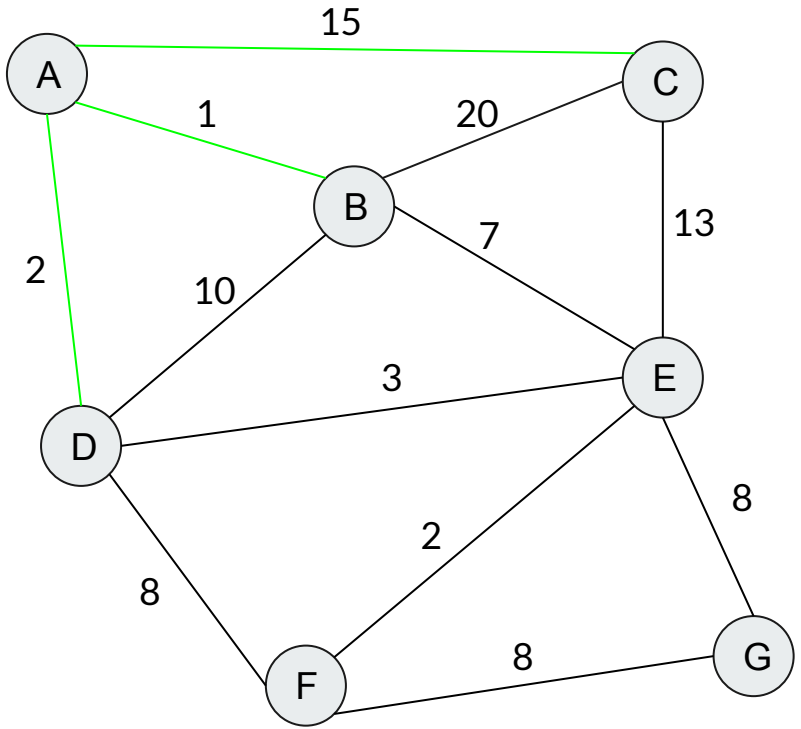


Dijkstra's: Example

Visiting A

S = {B,C,G}

V	D(v)
A	7
B	8
C	15
D	5
E	2
F	0
G	8

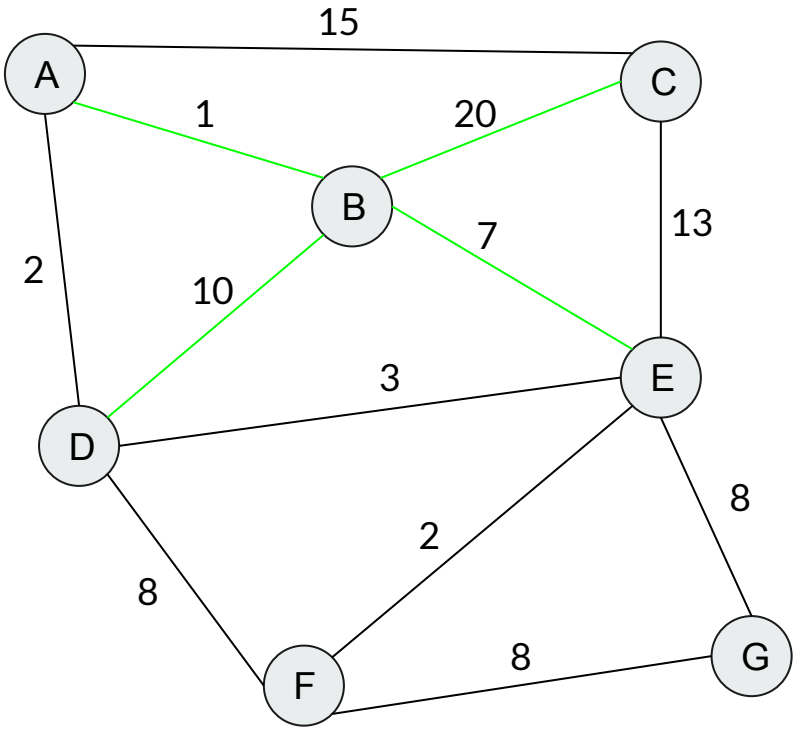


Dijkstra's: Example

Visiting B

S = {C,G}

V	D(v)
A	7
B	8
C	15
D	5
E	2
F	0
G	8

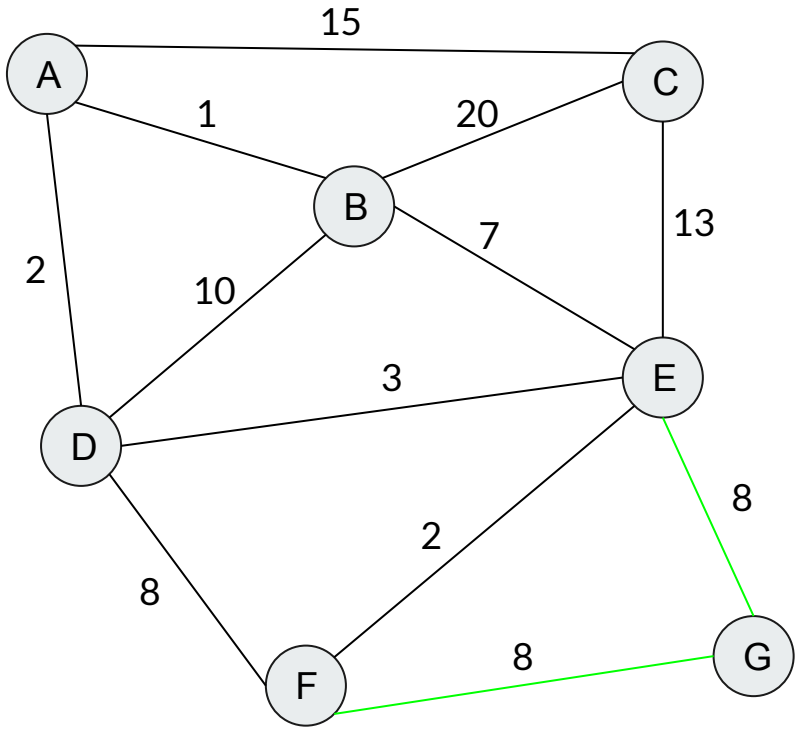


Dijkstra's: Example

Visiting G

S = {C}

V	D(v)
A	7
B	8
C	15
D	5
E	2
F	0
G	8

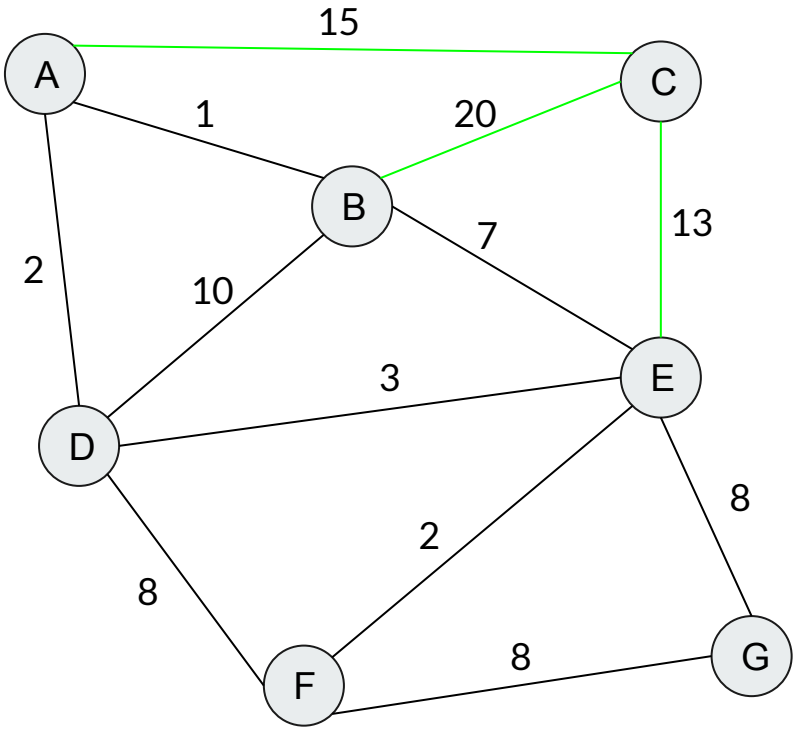


Dijkstra's: Example

Visiting C

S = {}

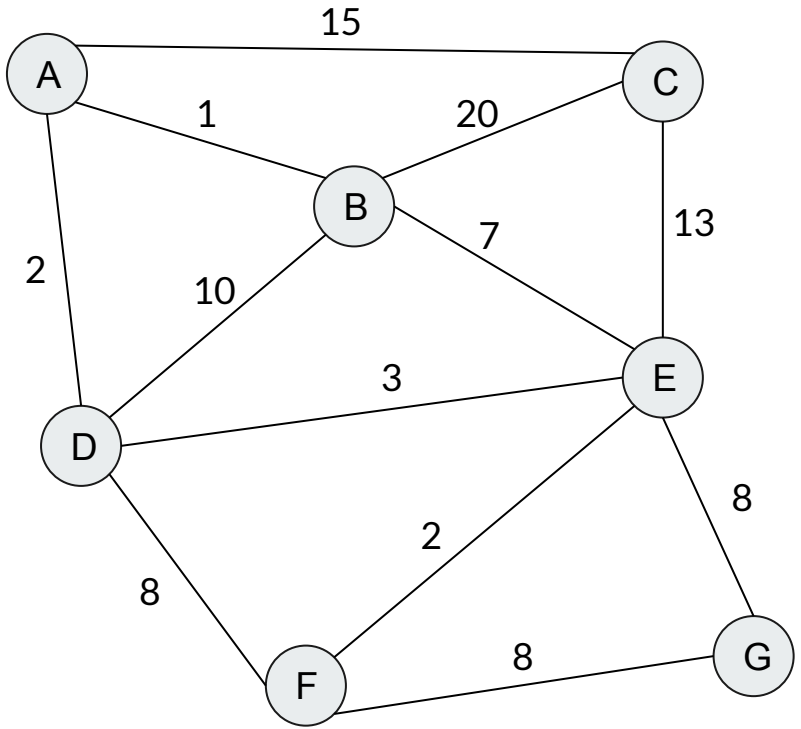
V	D(v)
A	7
B	8
C	15
D	5
E	2
F	0
G	8



Dijkstra's: Example

Return distances to all vertices.

V	D(v)
A	7
B	8
C	15
D	5
E	2
F	0
G	8



Dijkstra's Algorithm Proof

Proof by induction: For all visited nodes, D(v) is the shortest distance to that node. For all unvisited nodes, D(v) is the shortest distance to that node only using visited nodes.

Basis step: When only the base node is visited, its distance is 0. For all other nodes, D(v) is set to the shortest distance to that node using only the source.

Dijkstra's Algorithm Proof

Inductive step:

Assume for n nodes, the proposition holds. We consider adding node n+1, which we denote u. We show that D(u) is the shortest path to u by contradiction, in two cases.

Case 1: There is a shorter path to u, which contains an unvisited node w. Because of the proposition, D(u) must be less than D(w). This leads to a contradiction, because a path that runs through a further point cannot be a shortest path.

Case 2: There is a shorter path to u, which contains only visited nodes. In this case, the last node on the path must have been already visited, since it must have been closer than u. This means that D(u) along this path must already have been calculated, leading to a contradiction.