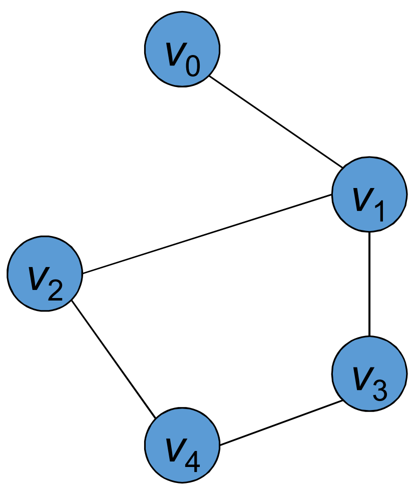
Graph algorithms

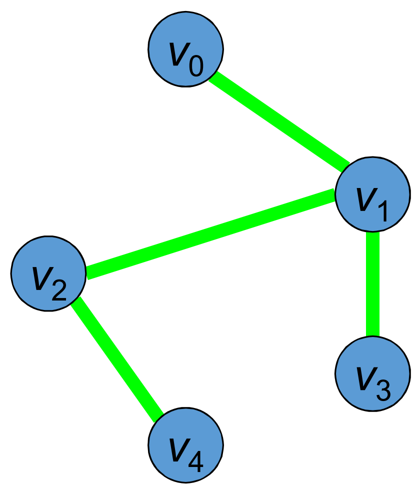
- **Graph Traversal (Graph Searching)**
  - Breadth-first search
  - Depth-first search
- **Shortest-Path Algorithm**
  - Dijkstra’s algorithm
- **Minimum Spanning Tree**
  - Prim’s Algortihm
  - Kruskal’s Algorithm

Spanning Tree

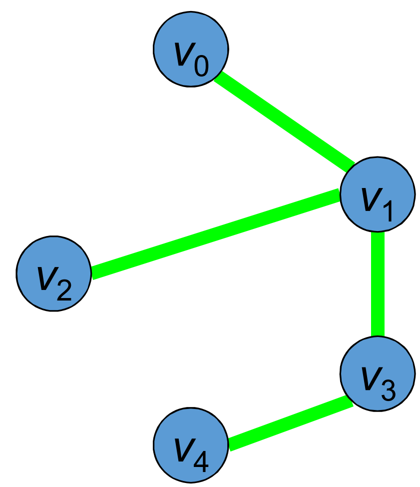
- A ***spanning tree*** (ST) of an *undirected* graph is a *tree* which contains *all vertices* and *some edges* of the graph.



Graph G



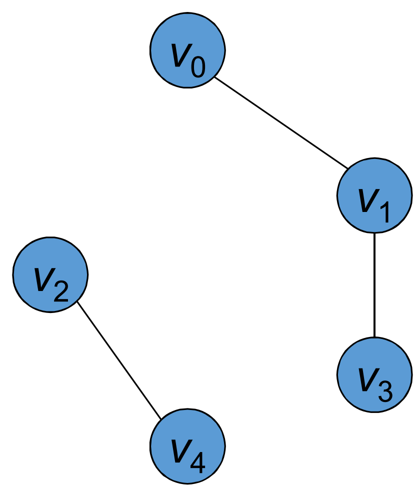
A spanning tree of G



Another spanning tree of G

Spanning Tree and Connectivity

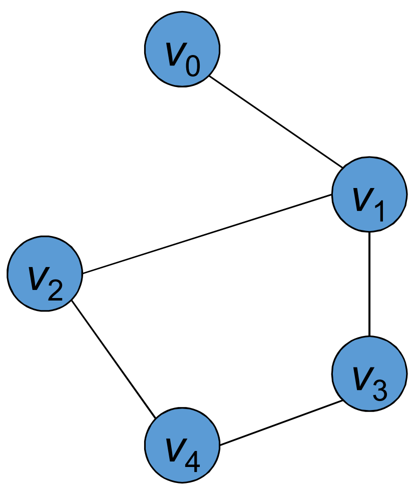
- An *undirected* graph has a spanning tree if and only if the graph is *connected*.



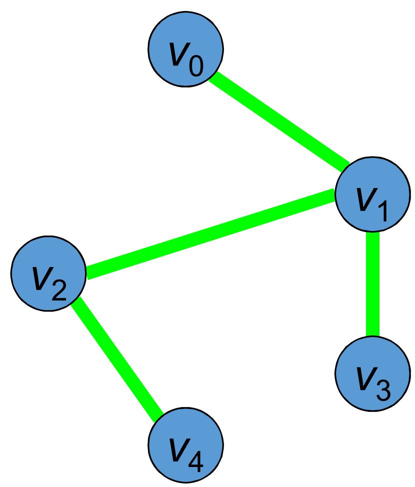
A graph that has ***no*** spanning trees

Spanning Tree

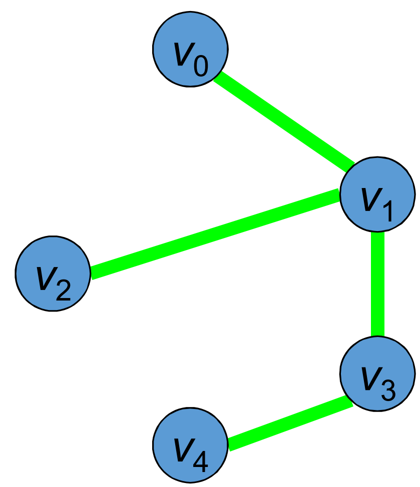
- For a graph with *n* vertices, its spanning tree always has exactly *n* - 1 edges.



Graph *G*



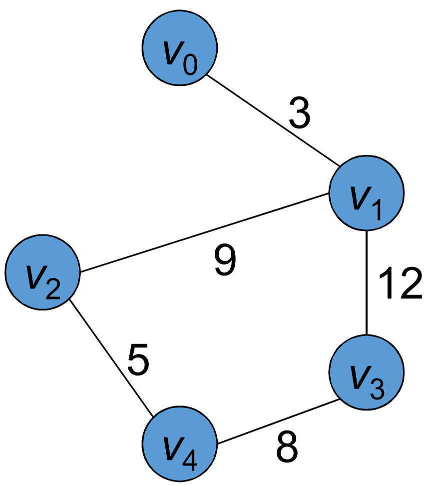
A spanning tree of *G*



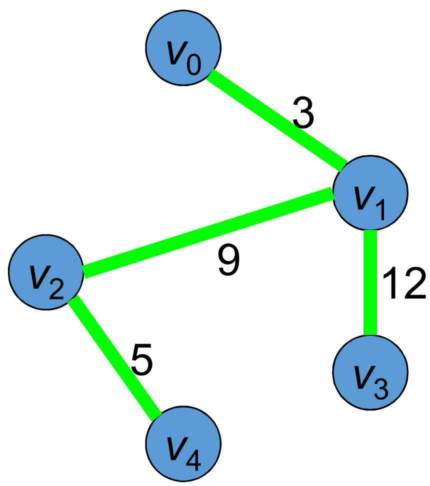
Another spanning tree of *G*

Minimum Spanning Tree

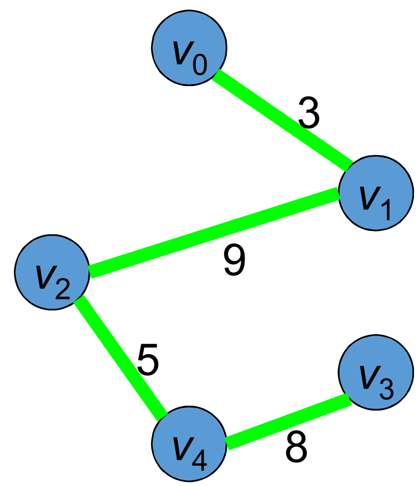
- A ***minimum spanning tree*** (MST) of an *undirected weighted* graph is a spanning tree whose sum of all weights is minimum.



Graph *G*



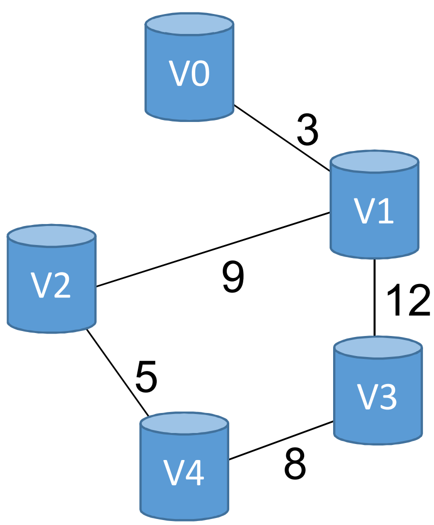
A spanning tree of *G*
(total weight = 3+9+12+5 = 29)

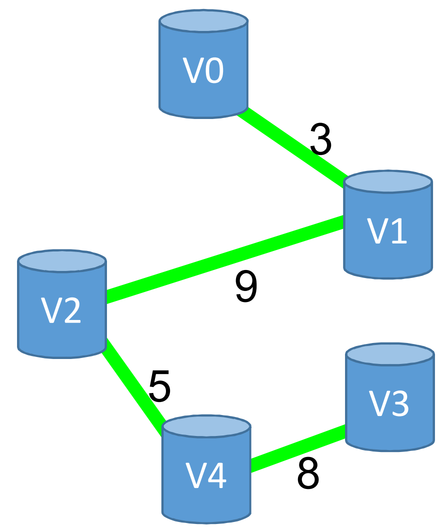


A MST of *G*
(total weight = 3+9+5+8 = 25)

Application : Minimum Spanning Tree

- Find the cheapest route to connect all computers/devices.

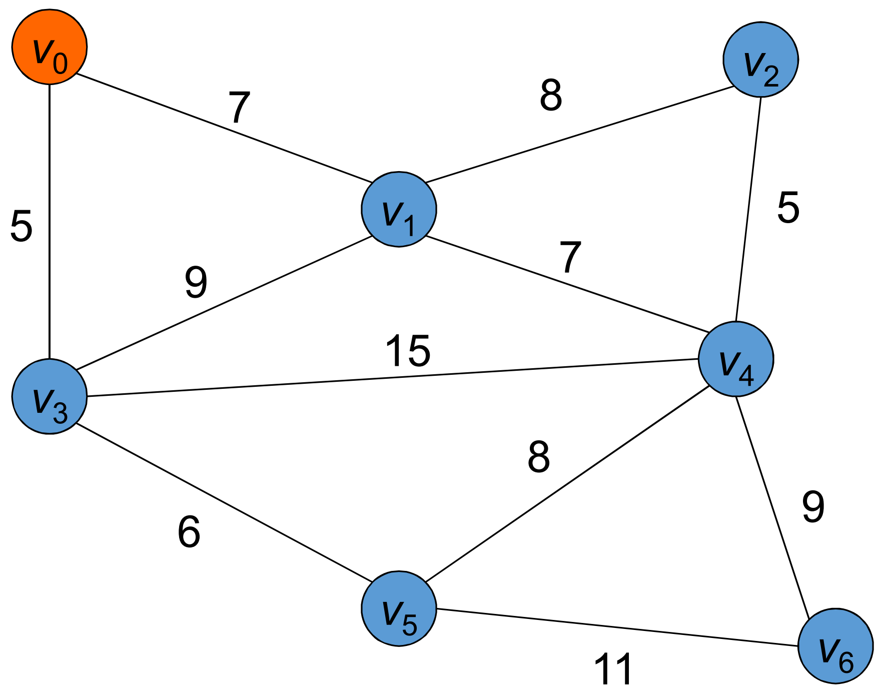




MST Algorithms

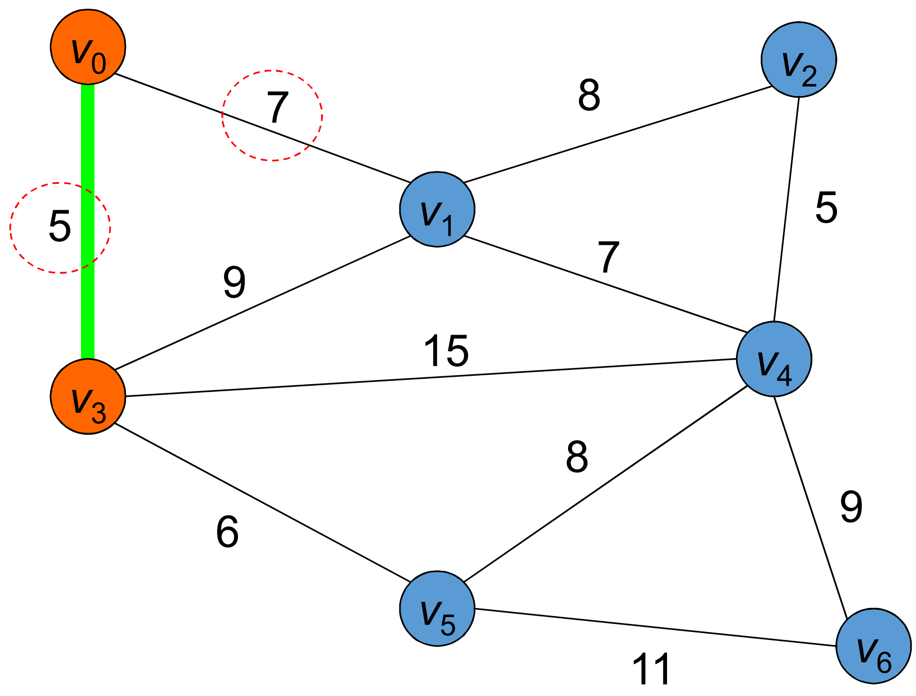
- Two common algorithms for finding MSTs.
  - ***Prim's algorithm***
    - Build tree to span all vertices
  - ***Kruskal's algorithm***
    - From "forest" to tree

Prim's Algorithm



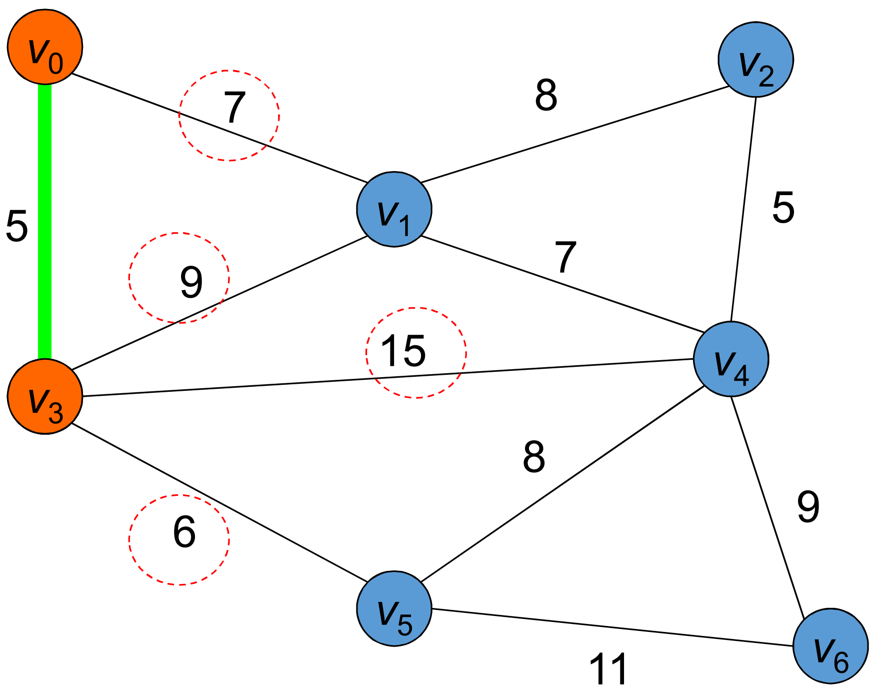
Start from any one vertex, say, v_0.
(Consider v_0 as a tree with one node.)

Prim's Algorithm



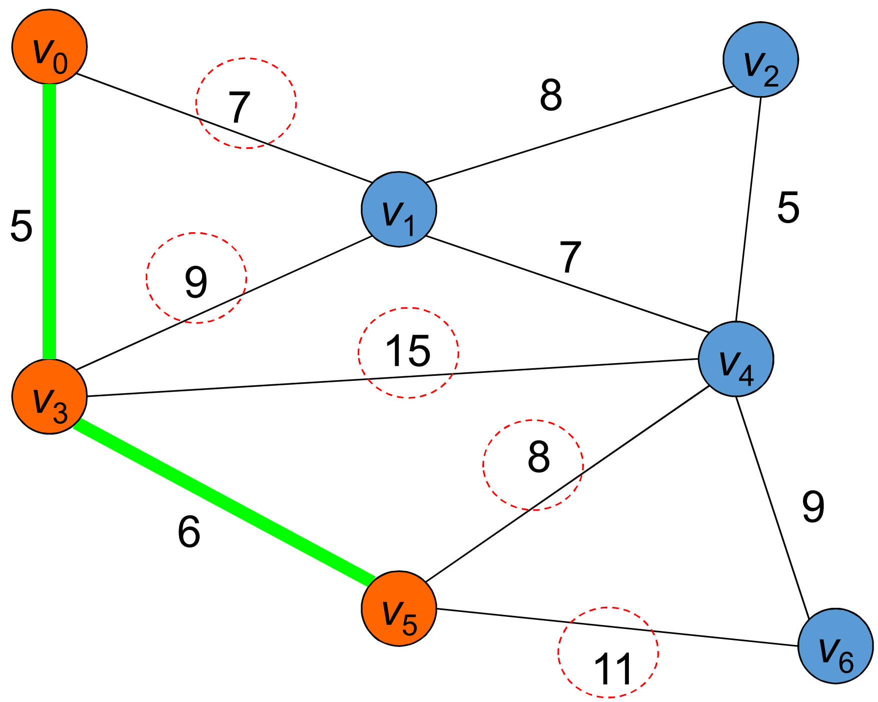
Pick the edge around the current tree whose weight is the smallest and link to an unvisited node.

Prim's Algorithm



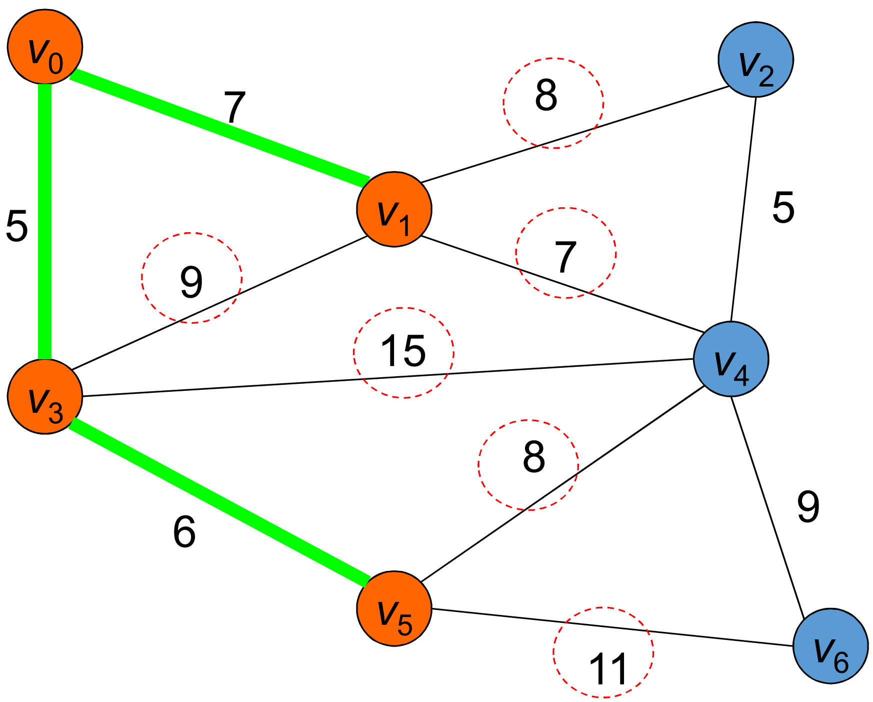
Pick the edge around the current tree whose weight is the smallest and link to an unvisited node.

Prim’s Algorithm



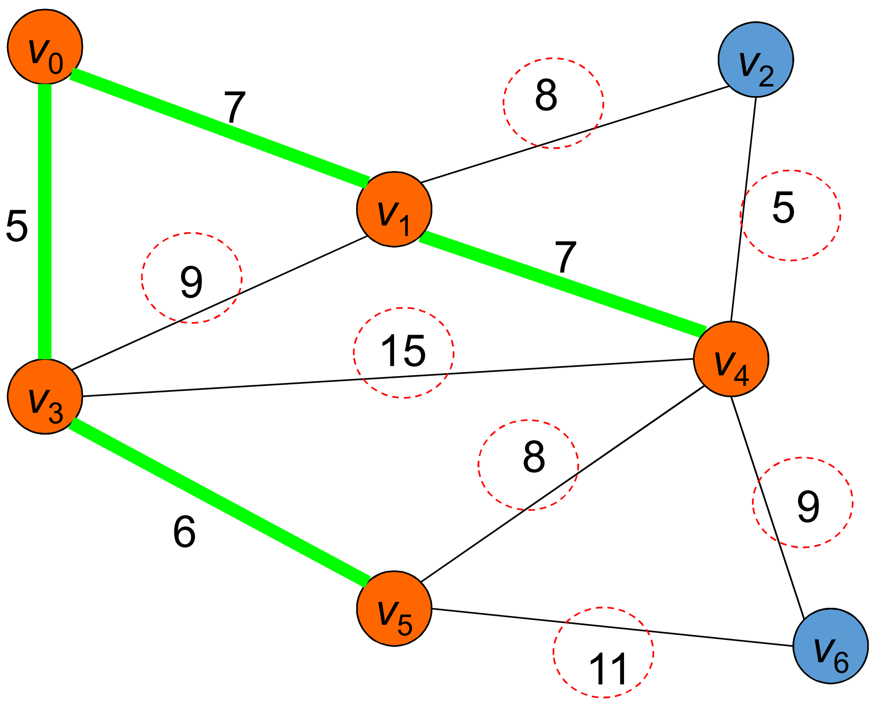
Pick the edge around the current tree whose weight is the smallest and link to an unvisited node.

Prim's Algorithm



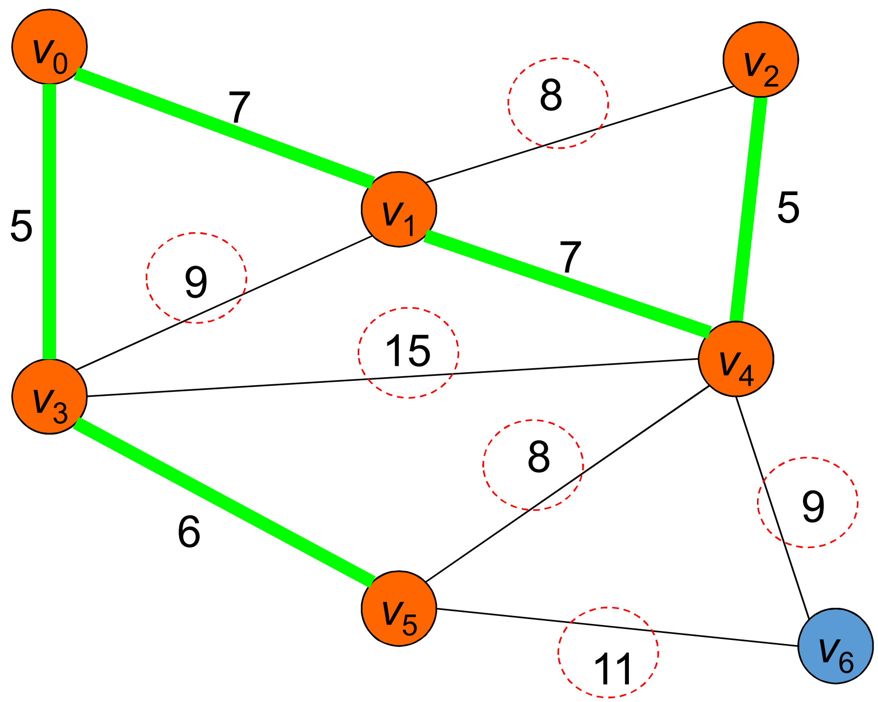
Pick the edge around the current tree whose weight is the smallest and link to an unvisited node.

Prim’s Algorithm



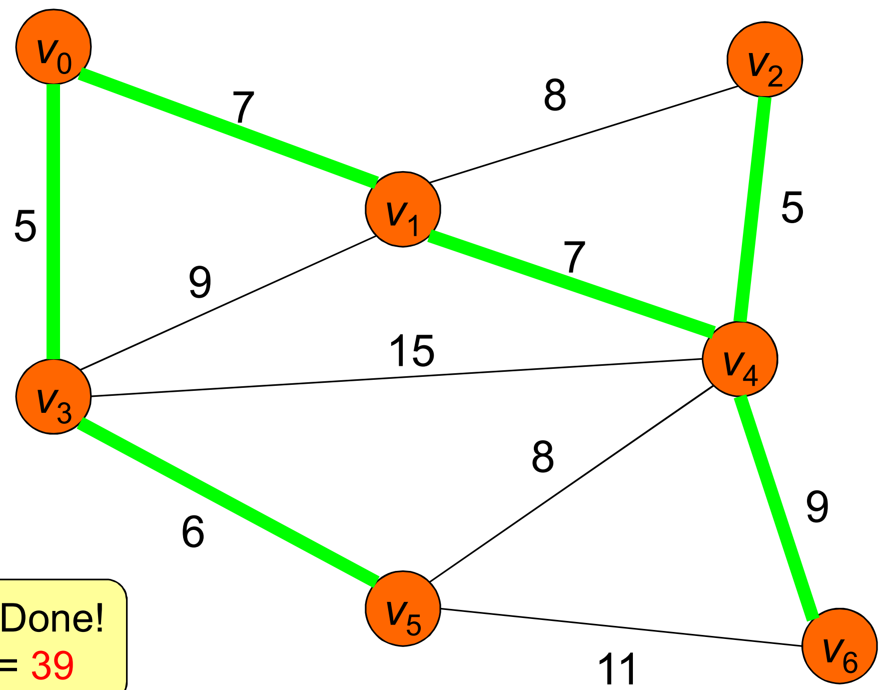
Pick the edge around the current tree whose weight is the smallest and link to an unvisited node.

Prim's Algorithm



Pick the edge around the current tree whose weight is the smallest and link to an unvisited node.

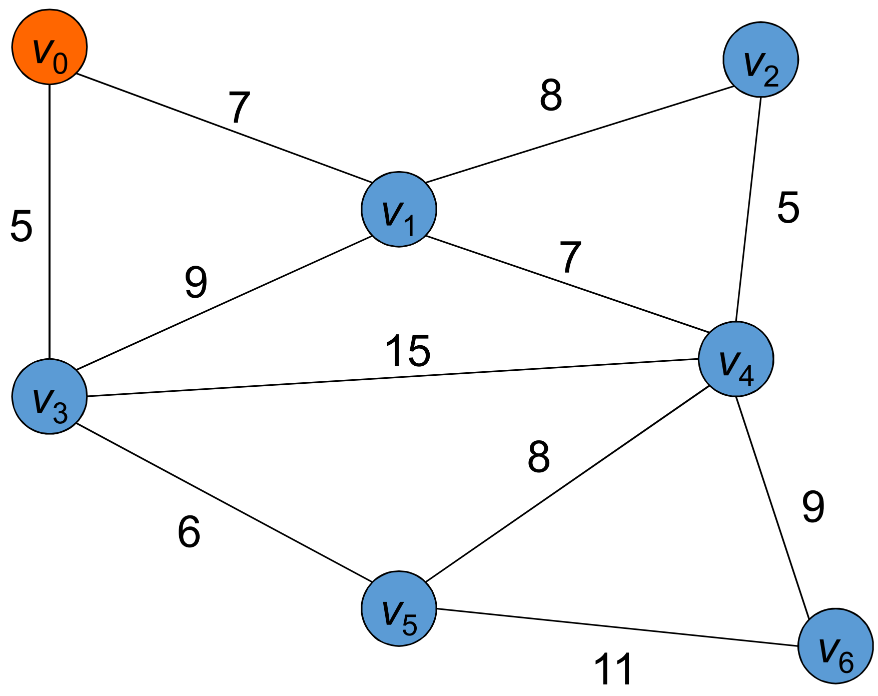
Prim’s Algorithm



Spans all vertices. Done!
Total tree weight = 39

Pick the edge around the current tree whose weight is the smallest and link to an unvisited node.

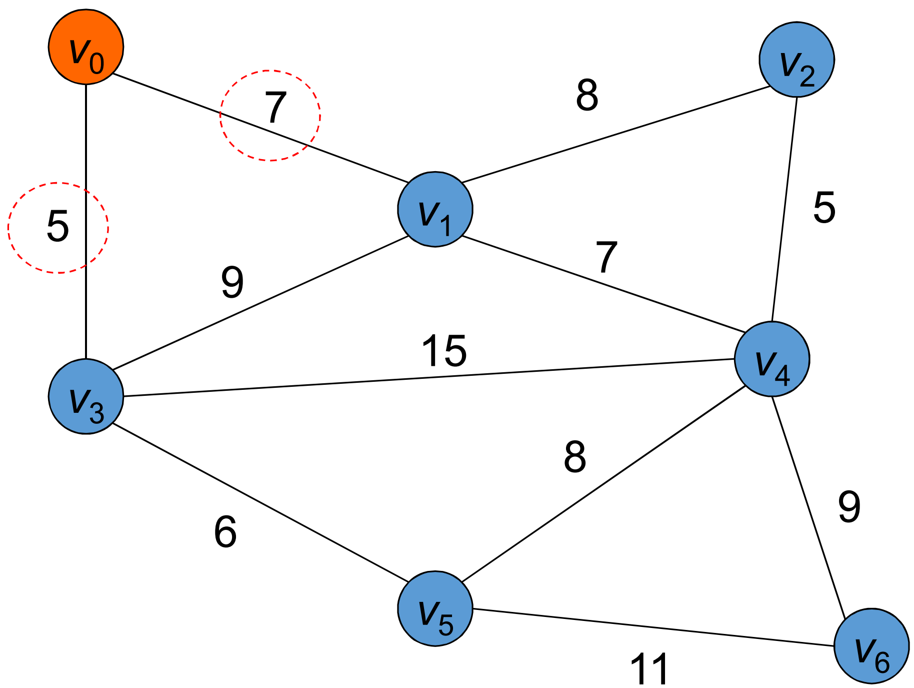
Prim’s Algorithm



vertex	distance	previous
v_0	0	0
v_1	∞	0
v_2	∞	0
v_3	∞	0
v_4	∞	0
v_5	∞	0
v_6	∞	0

Start from any one vertex, say, v_0.
(Consider v_0 as a tree with one node.)

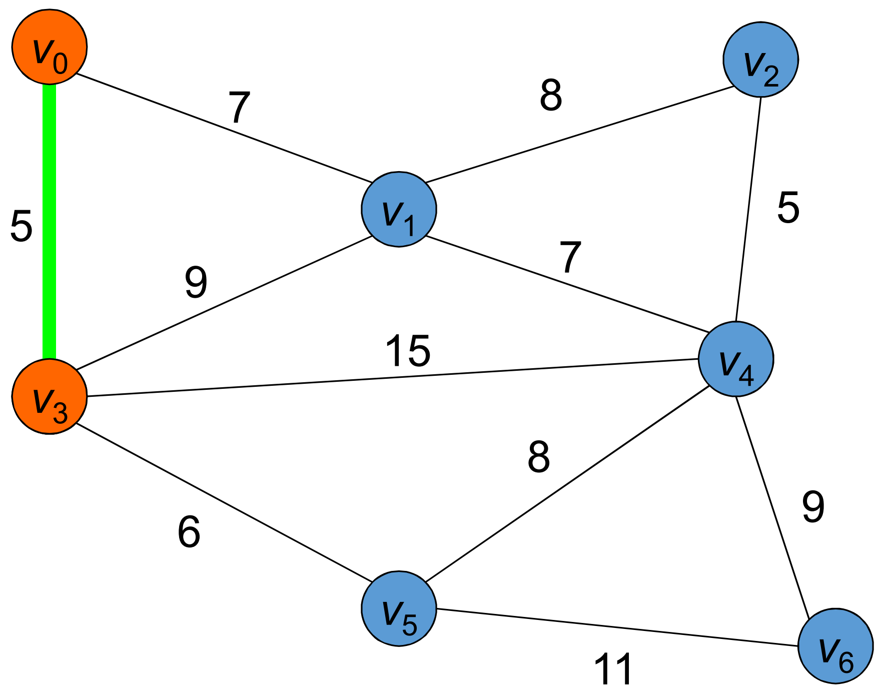
Prim’s Algorithm



vertex	distance	previous
v_0	0	0
v_1	7	v_0
v_2	∞	0
v_3	5	v_0
v_4	∞	0
v_5	∞	0
V_6	∞	0

Update the distance to all nodes adjacent to v_0

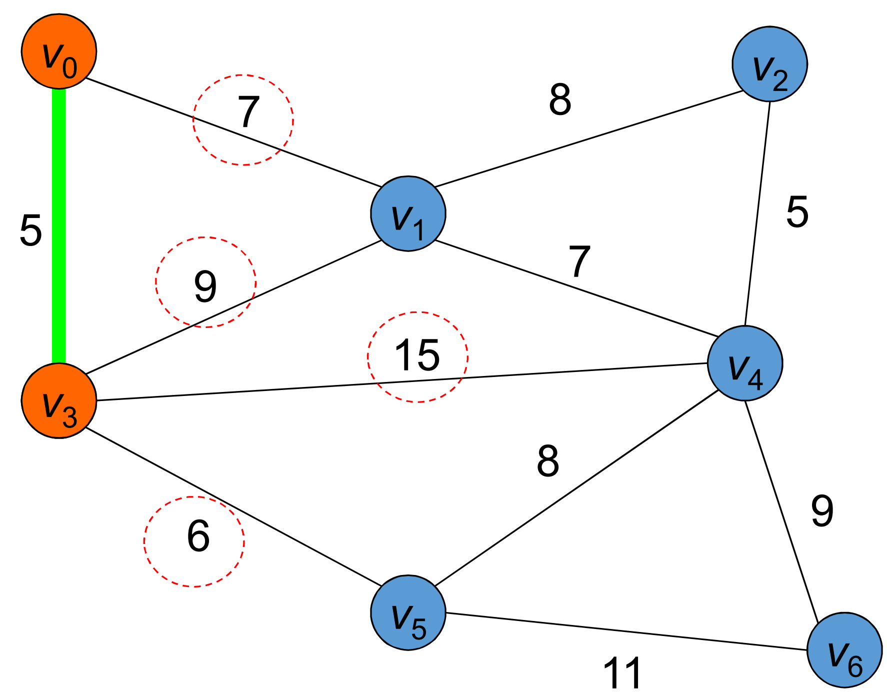
Prim's Algorithm



vertex	distance	previous
v_0	0	0
v_1	7	v_0
v_2	∞	0
v_3	5	v_0
v_4	∞	0
v_5	∞	0
v_6	∞	0

Pick the edge around the current tree whose weight is the smallest and link to an unvisited node.

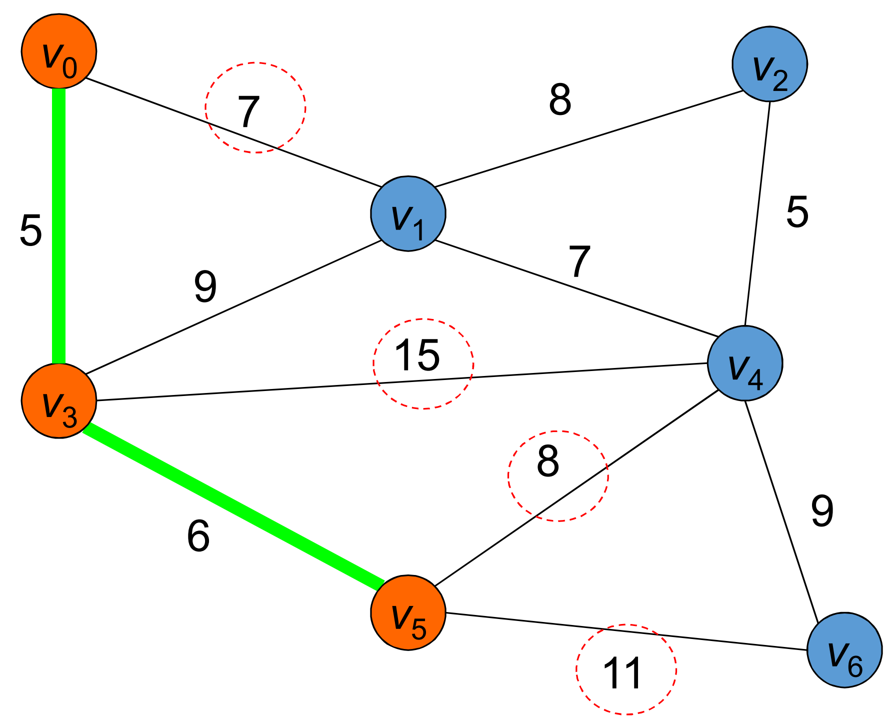
Prim’s Algorithm



vertex	distance	previous
v_0	0	0
v_1	Min(7,9)	v_0
v_2	∞	0
v_3	5	v_0
v_4	15	v_3
v_5	6	v_3
v_6	∞	0

Update the distance to all nodes adjacent to v_0

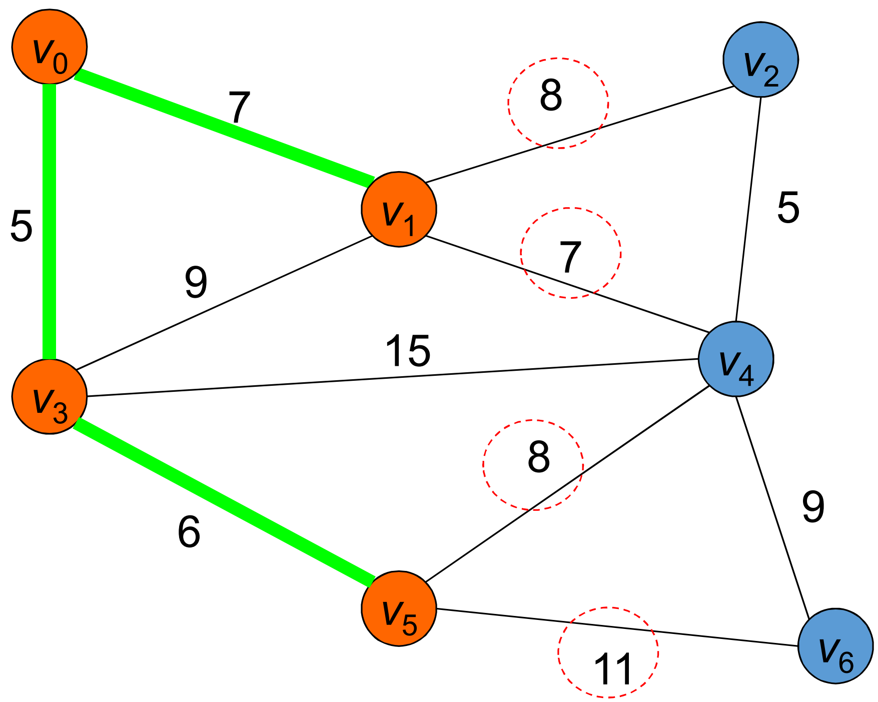
Prim's Algorithm



vertex	distance	previous
v_0	0	0
v_1	7	v_0
v_2	∞	0
v_3	5	v_0
v_4	Min(15,8)	v_5
v_5	6	v_3
v_6	11	v_5

Pick the edge around the current tree whose weight is the smallest and link to an unvisited node.

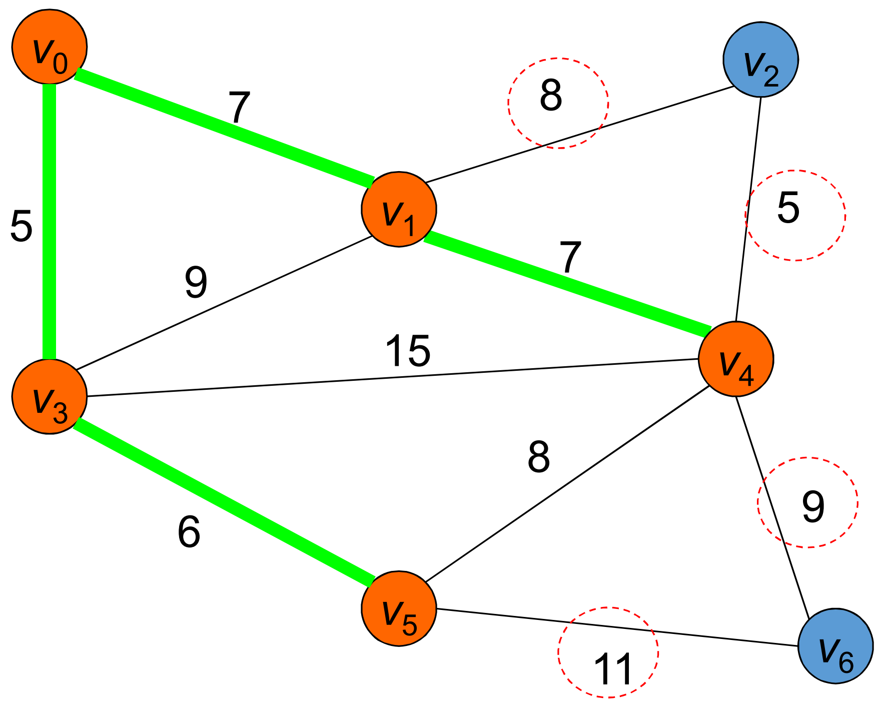
Prim's Algorithm



vertex	distance	previous
v_0	0	0
v_1	7	v_0
v_2	8	v_1
v_3	5	v_0
v_4	Min(8,7)	v_1
v_5	6	v_3
v_6	11	v_5

Pick the edge around the current tree whose weight is the smallest and link to an unvisited node.

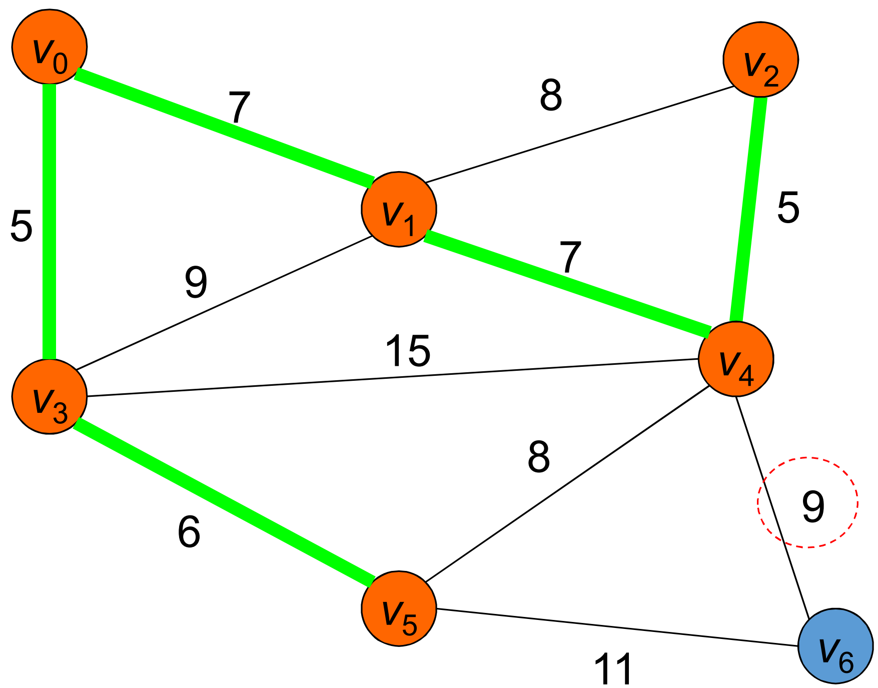
Prim's Algorithm



vertex	distance	previous
v_0	0	0
v_1	7	v_0
v_2	Min(8,5)	v_4
v_3	5	v_0
v_4	7	v_1
v_5	6	v_3
v_6	Min(11,9)	v_4

Pick the edge around the current tree whose weight is the smallest and link to an unvisited node.

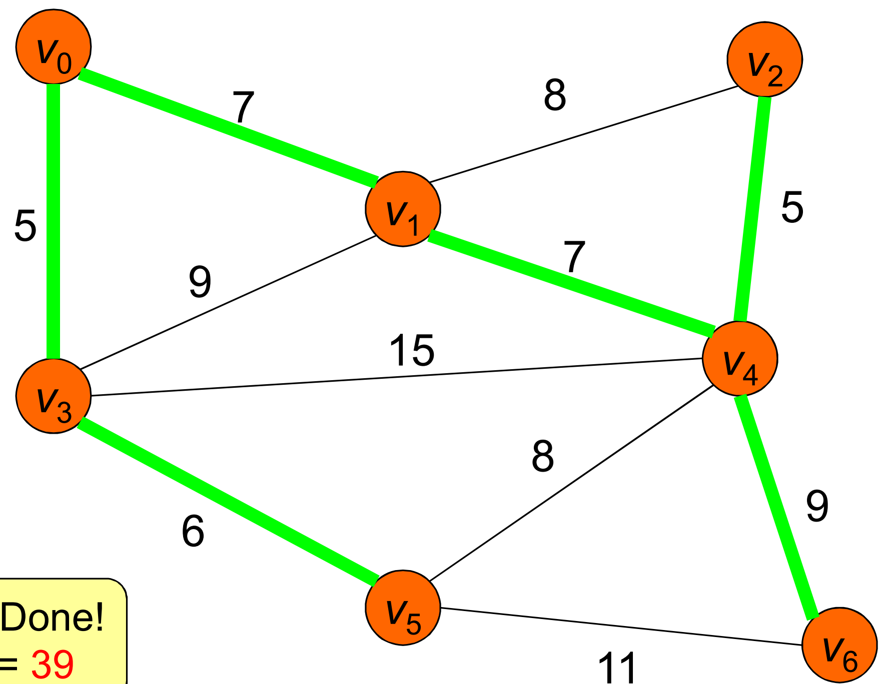
Prim's Algorithm



vertex	distance	previous
v_0	0	0
v_1	7	v_0
v_2	5	v_4
v_3	5	v_0
v_4	7	v_1
v_5	6	v_3
v_6	9	v_4

Pick the edge around the current tree whose weight is the smallest and link to an unvisited node.

Prim’s Algorithm



vertex	distance	previous
v_0	0	0
v_1	7	v_0
v_2	5	v_4
v_3	5	v_0
v_4	7	v_1
v_5	6	v_3
v_6	9	v_4

Spans all vertices. Done!
Total tree weight = 39

Pick the edge around the current tree whose weight is the smallest and link to an unvisited node.

Prim's Algorithm

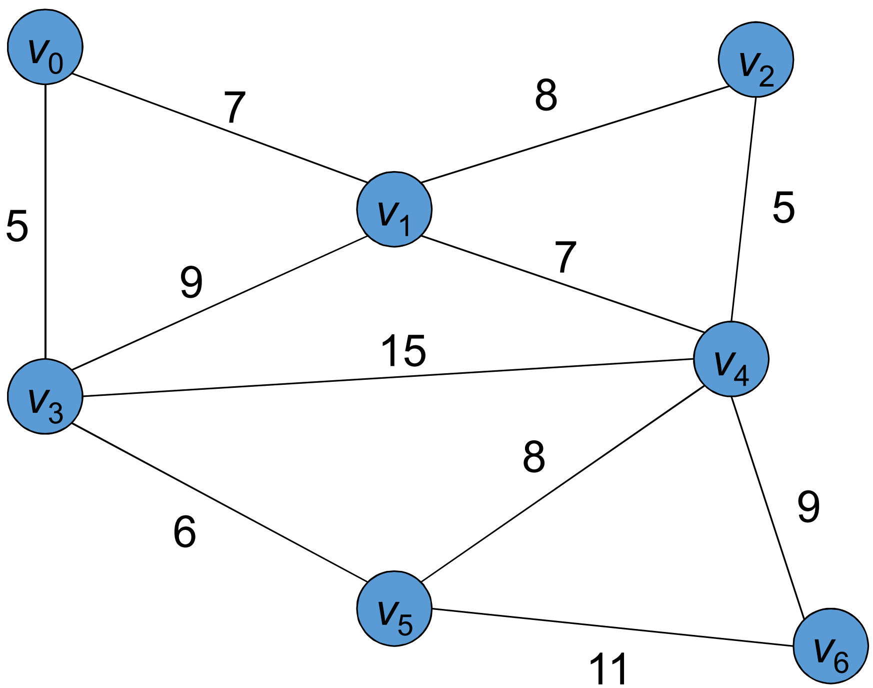
- Initialize the current tree T to have any one vertex.
- While T has fewer than $n - 1$ edges
  - Pick the edge around T whose weight is the smallest and link to an unvisited node.
  - (When no such edge is found, graph is disconnected and no MST exists.)

Kruskal’s Algorithm

Increasingly sorted by weights

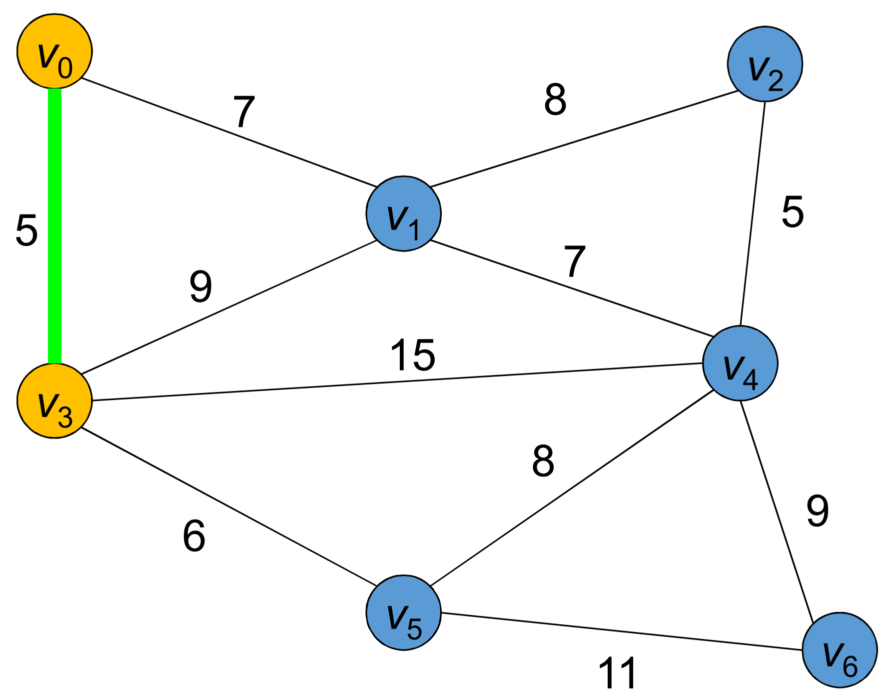
Edge	Weight
(v_0, v_3)	5
(v_2, v_4)	5
(v_3, v_5)	6
(v_0, v_1)	7
(v_1, v_4)	7
(v_1, v_2)	8
(v_4, v_5)	8
(v_1, v_3)	9
(v_4, v_6)	9
(v_5, v_6)	11
(v_3, v_4)	15

Pick edge one by one



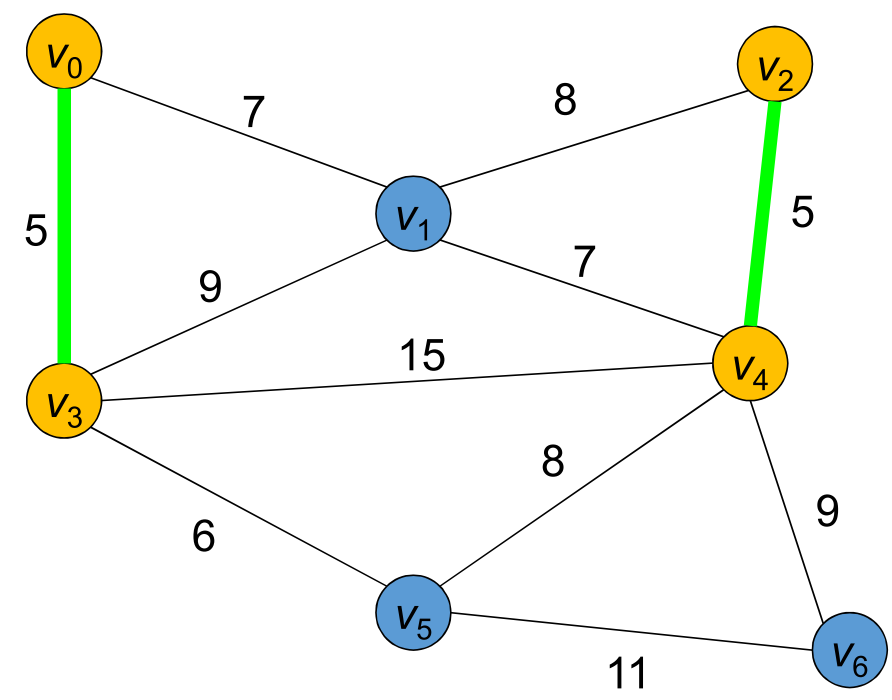
Kruskal's Algorithm

Edge	Weight	
(v_0, v_3)	5	✓
(v_2, v_4)	5	
(v_3, v_5)	6	
(v_0, v_1)	7	
(v_1, v_4)	7	
(v_1, v_2)	8	
(v_4, v_5)	8	
(v_1, v_3)	9	
(v_4, v_6)	9	
(v_5, v_6)	11	
(v_3, v_4)	15	



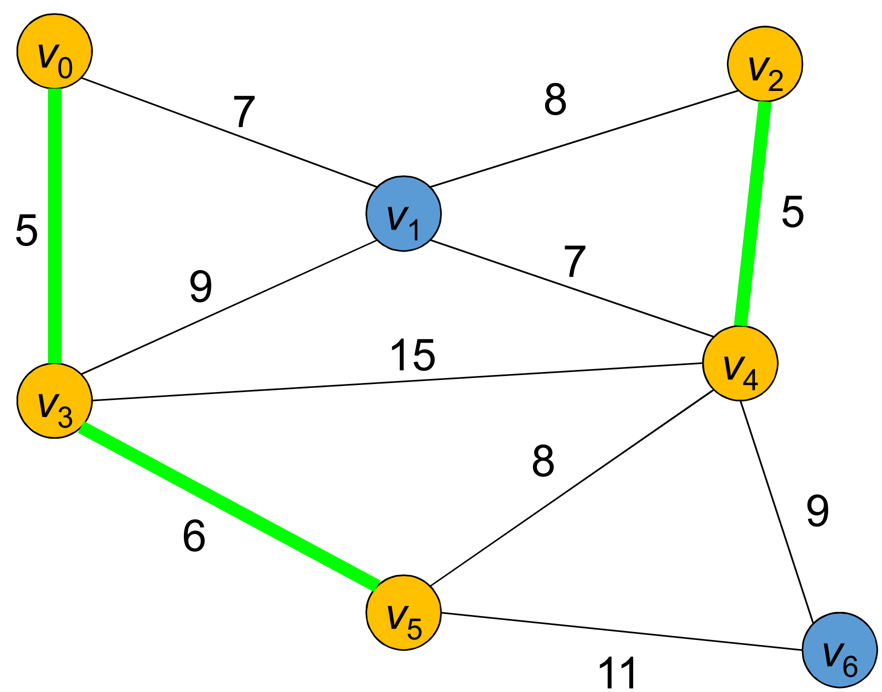
Kruskal's Algorithm

Edge	Weight	
(v_0, v_3)	5	✓
(v_2, v_4)	5	✓
(v_3, v_5)	6	
(v_0, v_1)	7	
(v_1, v_4)	7	
(v_1, v_2)	8	
(v_4, v_5)	8	
(v_1, v_3)	9	
(v_4, v_6)	9	
(v_5, v_6)	11	
(v_3, v_4)	15	



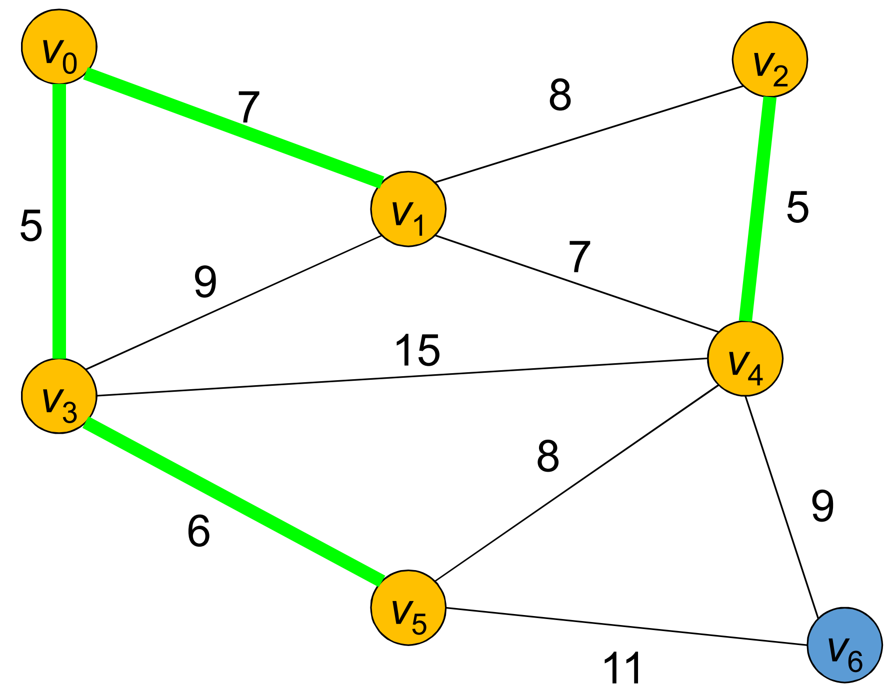
Kruskal's Algorithm

Edge	Weight	
(v_0, v_3)	5	✓
(v_2, v_4)	5	✓
(v_3, v_5)	6	✓
(v_0, v_1)	7	
(v_1, v_4)	7	
(v_1, v_2)	8	
(v_4, v_5)	8	
(v_1, v_3)	9	
(v_4, v_6)	9	
(v_5, v_6)	11	
(v_3, v_4)	15	



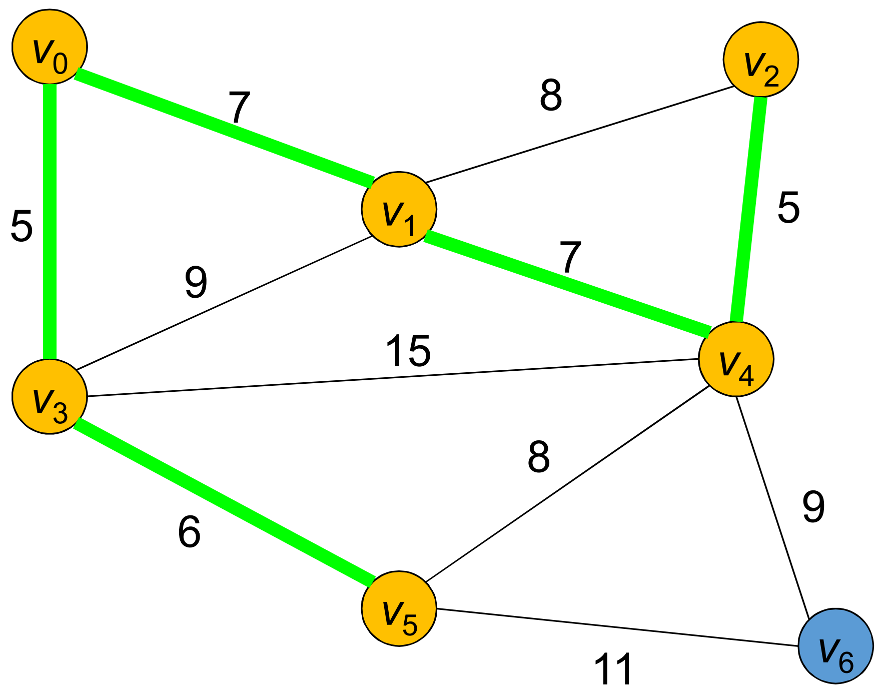
Kruskal's Algorithm

Edge	Weight	
(v_0, v_3)	5	✓
(v_2, v_4)	5	✓
(v_3, v_5)	6	✓
(v_0, v_1)	7	✓
(v_1, v_4)	7	
(v_1, v_2)	8	
(v_4, v_5)	8	
(v_1, v_3)	9	
(v_4, v_6)	9	
(v_5, v_6)	11	
(v_3, v_4)	15	



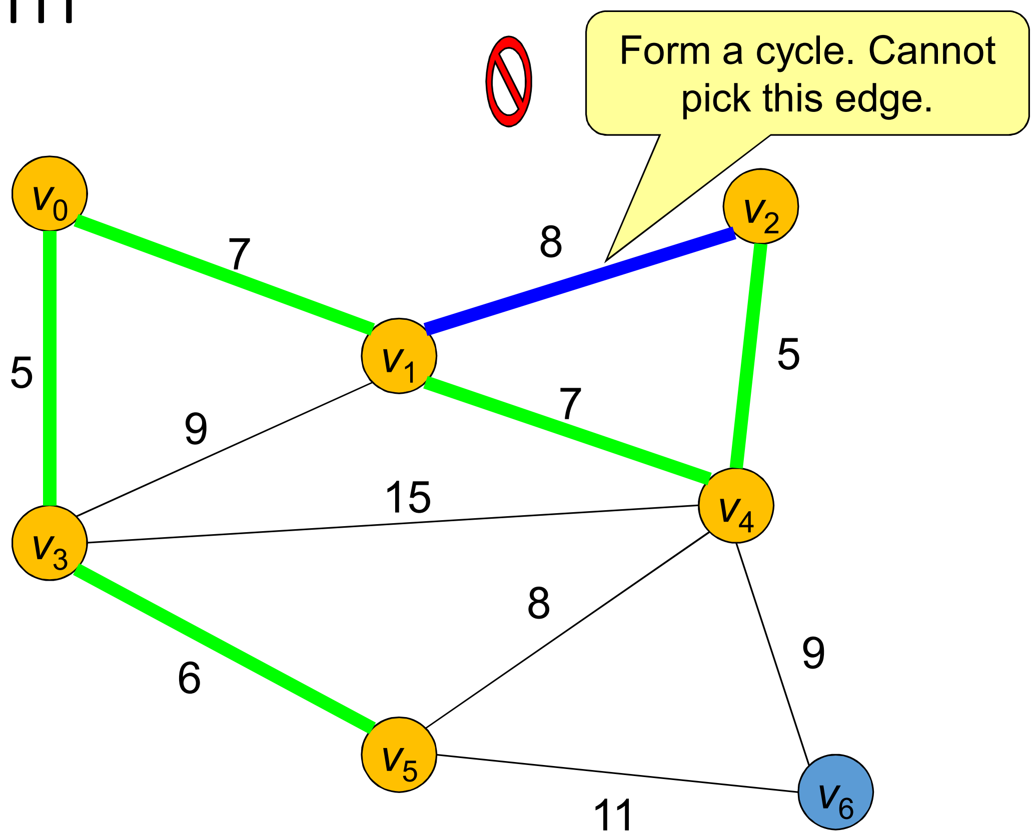
Kruskal’s Algorithm

Edge	Weight	
(v_0, v_3)	5	✓
(v_2, v_4)	5	✓
(v_3, v_5)	6	✓
(v_0, v_1)	7	✓
(v_1, v_4)	7	✓
(v_1, v_2)	8	
(v_4, v_5)	8	
(v_1, v_3)	9	
(v_4, v_6)	9	
(v_5, v_6)	11	
(v_3, v_4)	15	



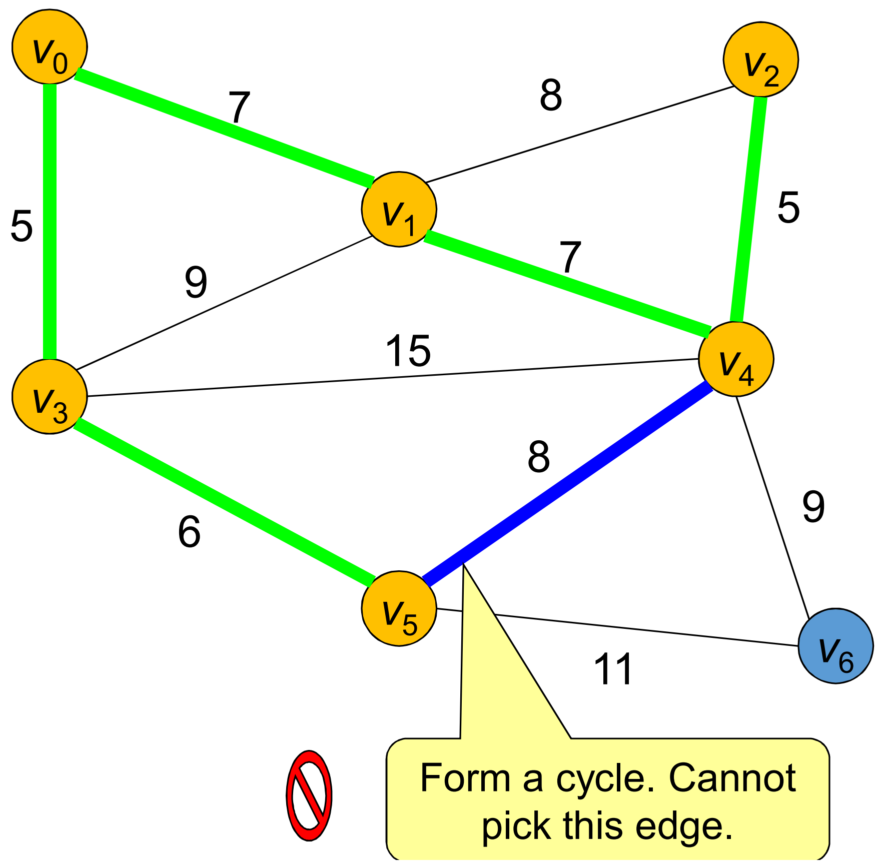
Kruskal's Algorithm

Edge	Weight	
(v_0, v_3)	5	✓
(v_2, v_4)	5	✓
(v_3, v_5)	6	✓
(v_0, v_1)	7	✓
(v_1, v_4)	7	✓
(v_1, v_2)	8	×
(v_4, v_5)	8	
(v_1, v_3)	9	
(v_4, v_6)	9	
(v_5, v_6)	11	
(v_3, v_4)	15	



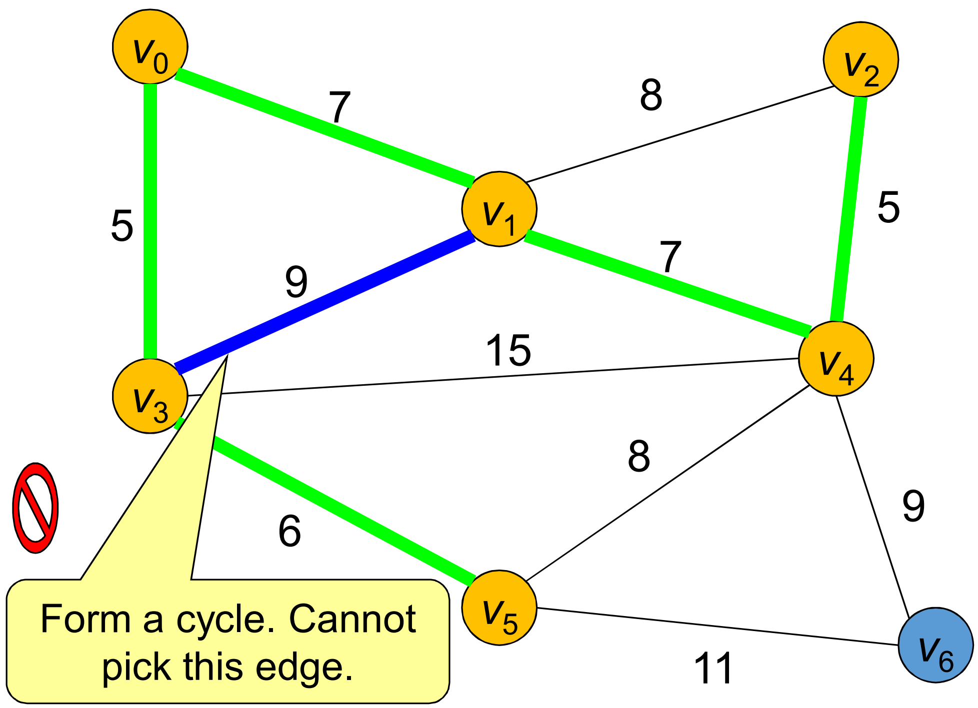
Kruskal's Algorithm

Edge	Weight	
(v_0, v_3)	5	✓
(v_2, v_4)	5	✓
(v_3, v_5)	6	✓
(v_0, v_1)	7	✓
(v_1, v_4)	7	✓
(v_1, v_2)	8	×
(v_4, v_5)	8	×
(v_1, v_3)	9	
(v_4, v_6)	9	
(v_5, v_6)	11	
(v_3, v_4)	15	



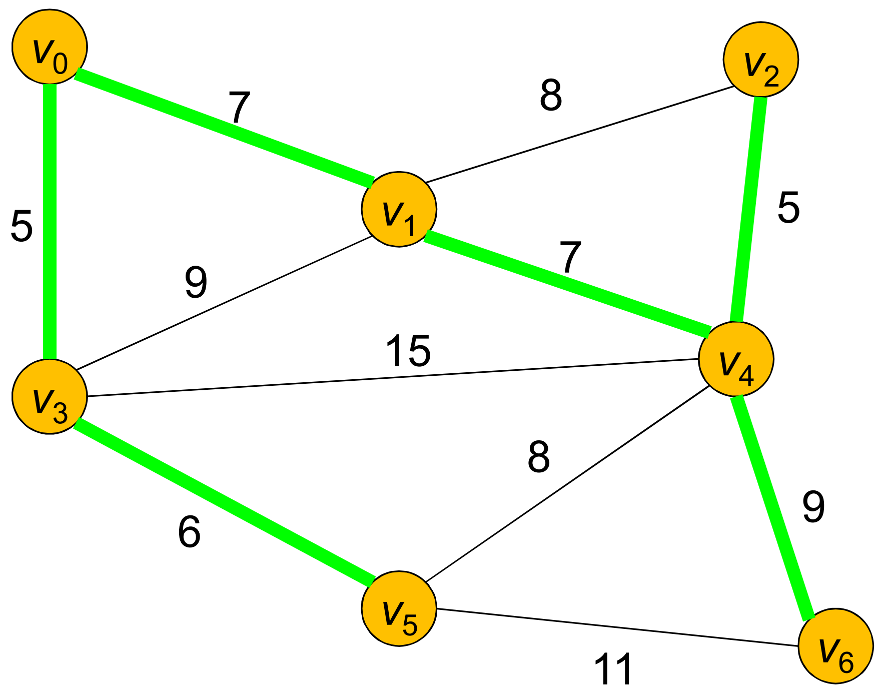
Kruskal's Algorithm

Edge	Weight	
(v_0, v_3)	5	✓
(v_2, v_4)	5	✓
(v_3, v_5)	6	✓
(v_0, v_1)	7	✓
(v_1, v_4)	7	✓
(v_1, v_2)	8	×
(v_4, v_5)	8	×
(v_1, v_3)	9	×
(v_4, v_6)	9	
(v_5, v_6)	11	
(v_3, v_4)	15	



Kruskal's Algorithm

Edge	Weight	
(v_0, v_3)	5	✓
(v_2, v_4)	5	✓
(v_3, v_5)	6	✓
(v_0, v_1)	7	✓
(v_1, v_4)	7	✓
(v_1, v_2)	8	×
(v_4, v_5)	8	×
(v_1, v_3)	9	×
(v_4, v_6)	9	✓
(v_5, v_6)	11	
(v_3, v_4)	15	



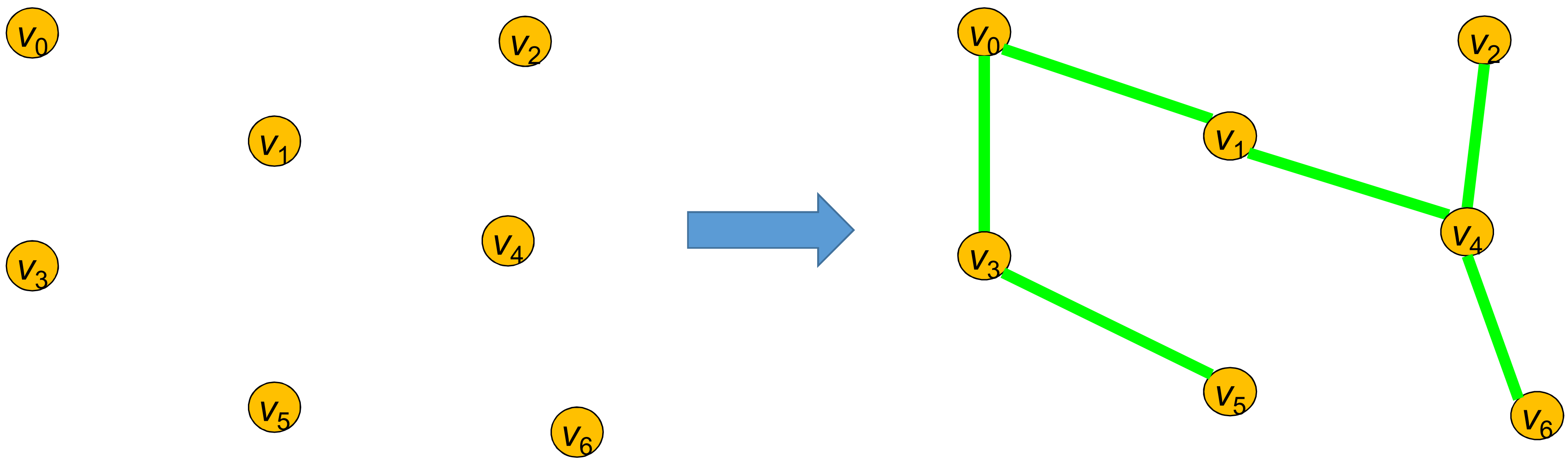
Spans all vertices. Done!
Total tree weight = 39

Kruskal's Algorithm

- Increasingly sort the edges by weights.
- For each edge *e* in sorted order
  - If *e* does not form a cycle with the already picked edges, then
    - Pick *e*. (Done when n - 1 edges are picked.)
  - Else
    - Discard *e*.
- If fewer than n - 1 edges are picked, then
  - Graph is disconnected. No MST exists.

Kruskal's Algorithm

- From "forest" to tree



Kruskal's vs Prim's Algorithms

- Order of edges picked to solution

Kruskal	Prim
(v_0, v_3)	(v_0, v_3)
(v_2, v_4)	(v_3, v_5)
(v_3, v_5)	(v_0, v_1)
(v_0, v_1)	(v_1, v_4)
(v_1, v_4)	(v_2, v_4)
(v_4, v_6)	(v_4, v_6)

- Kruskal's algorithm repeatedly merges multiple trees (forest) until only one tree is left.
- Prim's algorithm repeatedly grows one tree until all vertices are spanned.