

# Lecture 14

## Minimum Spanning Trees

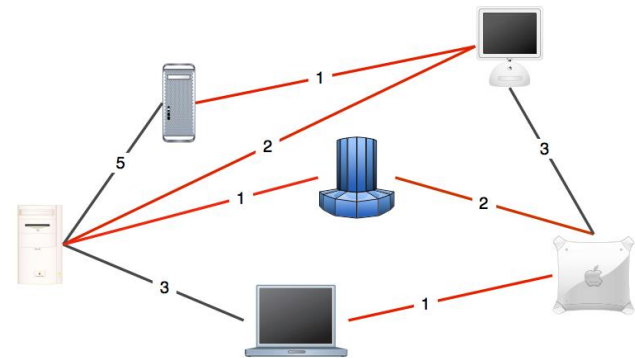
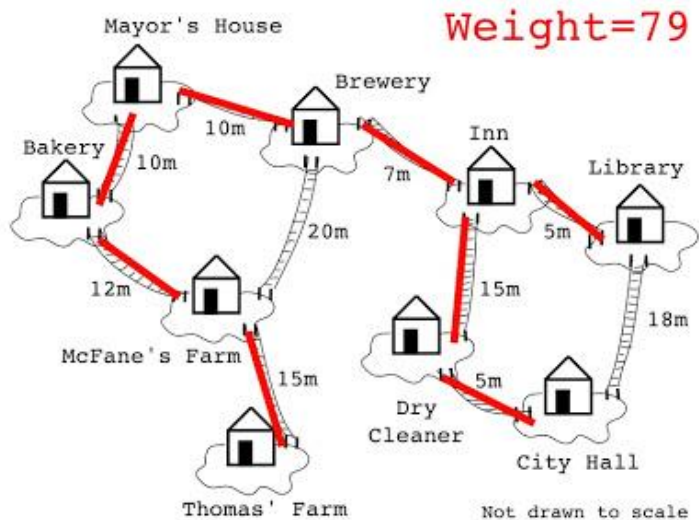
Department of Computer Science  
Hofstra University

# Lecture Goals

- In this lecture we study the minimum spanning tree problem.
- We consider two classic algorithm for the problem—Kruskal's algorithm and Prim's algorithm.
  - Both are greedy algorithms that are also optimal.

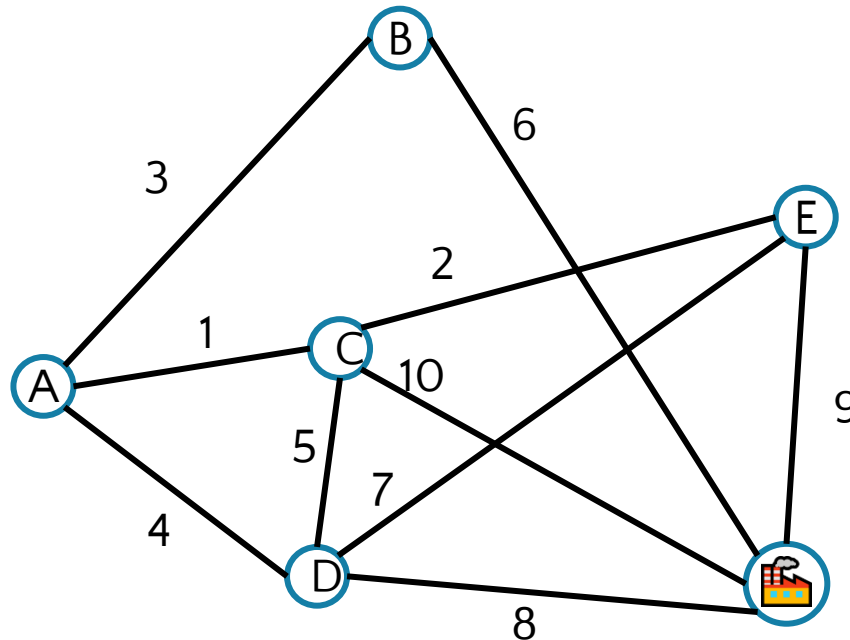
# Minimum Spanning Tree Problem

- A telecommunications company tries to lay cable in a new neighborhood. If it is constrained to bury the cable only along certain paths (e.g. roads), then there would be a graph containing the points (e.g. houses) connected by those paths.
- Some of the paths might be more expensive, because they are longer, or require the cable to be buried deeper; these paths would be represented by edges with larger weights.
- A *Minimum Spanning Tree (MST)* is the one with the lowest total cost for laying the cable.
- Other applications: Network design, Cluster analysis....



# Power Plant Example

- It's the 1920's. The electric company needs to choose where to build wires to connect all these cities (A,B, C, D, E) to the power plant
- They know the cost of to lay electric wires between any pair of cities, and wants the minimize the total cost of these wires

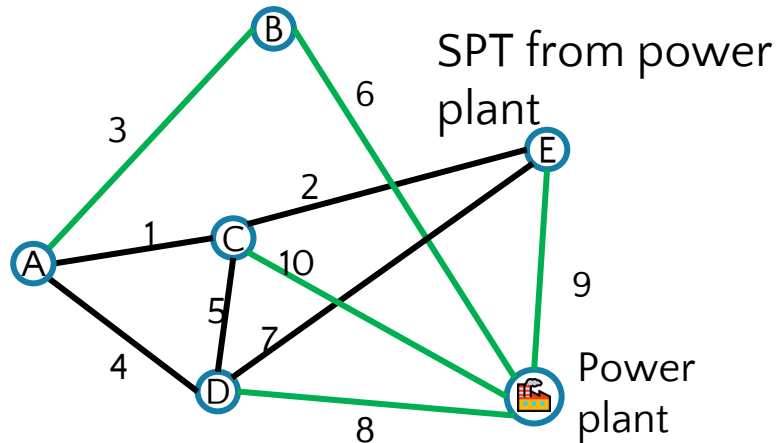


# Shortest Path vs Minimum Spanning Tree

## Shortest Path Problem

**Given:** a directed or undirected graph  $G$  and vertices  $s, t$

**Find:** the shortest path from  $s$  to  $t$ .



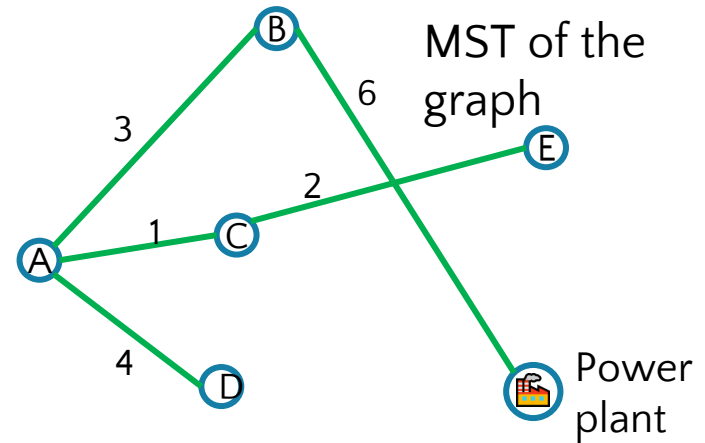
## Shortest Path Tree

- Goal: minimize distance from a specific start node (if you have a different start node, that changes the whole SPT)
- Keeps track of total path length

## Minimum Spanning Tree Problem

**Given:** an undirected graph  $G$

**Find:** A set of edges with minimum total sum of weights such that you can get from any vertex of  $G$  to any other on only those edges.



## Minimum Spanning Tree

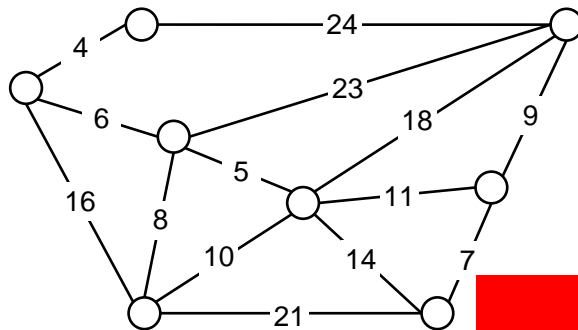
- Goal: minimize total sum of edge weights. Only one possible MST (if no equal edge weights)
- Keeps track of cheapest edges that maintain connectivity

# Minimum Spanning Tree (MST)

**Given.** Undirected graph  $G$  with positive edge weights (connected).

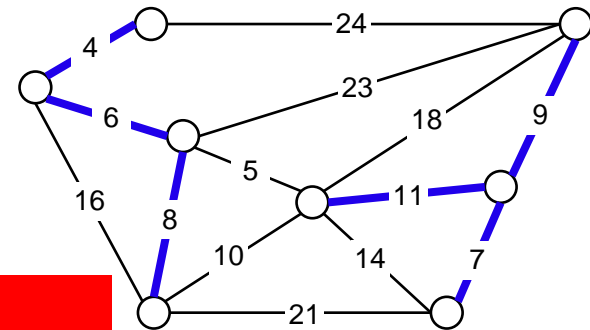
**Def.** A **spanning tree** of  $G$  is a subgraph  $T$  that is both a **tree** (connected and acyclic) and **spanning** (includes all of the vertices).

**Goal.** Find a minimum weight spanning tree.

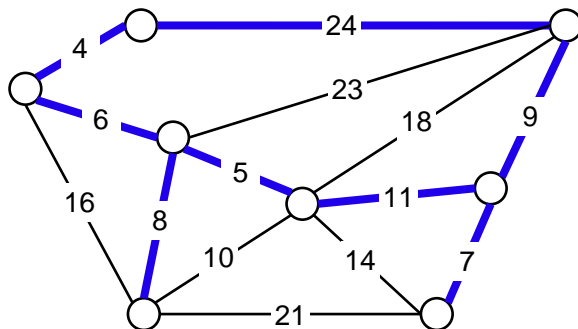


graph  $G$

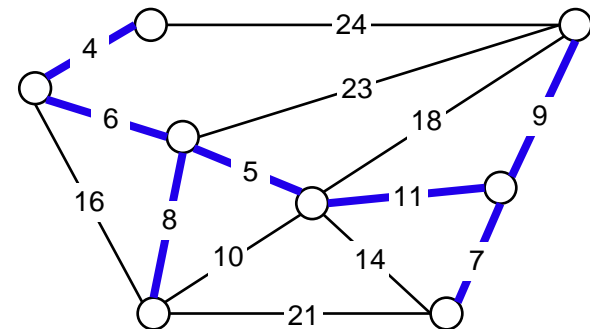
Brute force.  
Try all spanning trees?



not connected



not acyclic



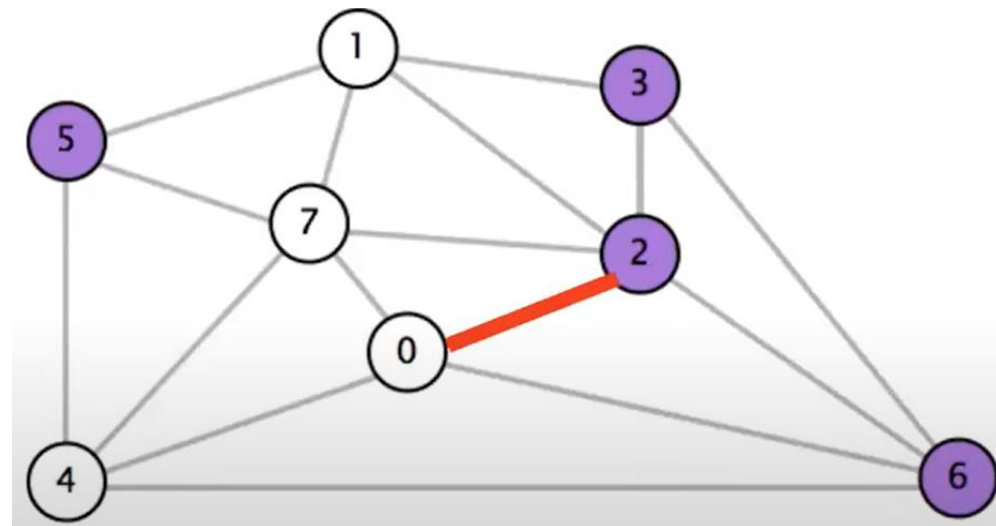
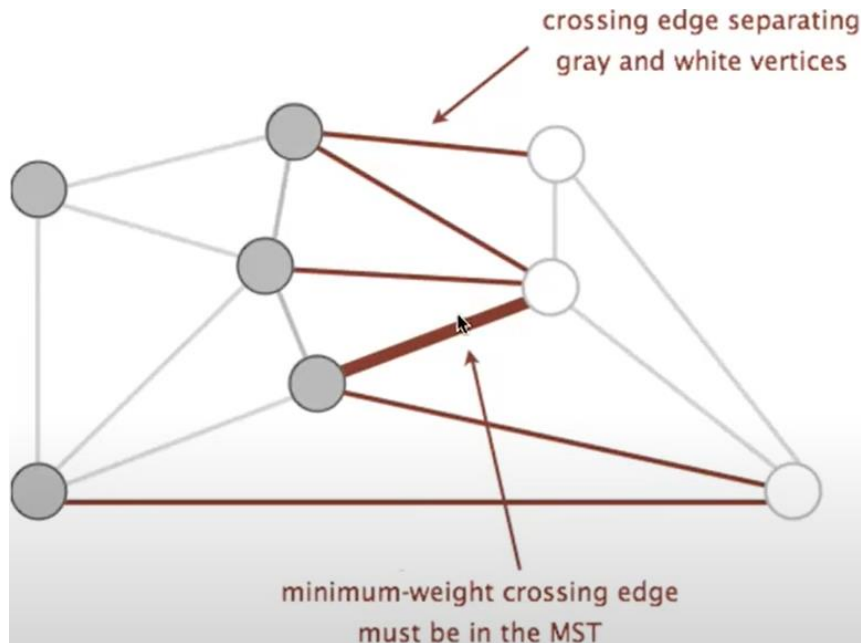
Minimum Spanning Tree  $T$ : cost = 50 = 4 + 6 + 8 + 5 + 11 + 9 + 7

# Greedy Algorithms

- A greedy algorithm makes the locally optimal choice at each step
  - We learned Dijkstra's algorithm for the shortest path problem
  - We will learn Kruskal and Prim's algorithms for the minimum spanning tree problem
- In the lecture and exams, when there are multiple possible orders of visiting the next node (in case of a tie), we generally select the next node in alphabetical or numerical order
  - The solution will always be guaranteed to be optimal, but the MST may be different depending on the order of visits

# A Useful Tool for Finding the MST: Cut Property

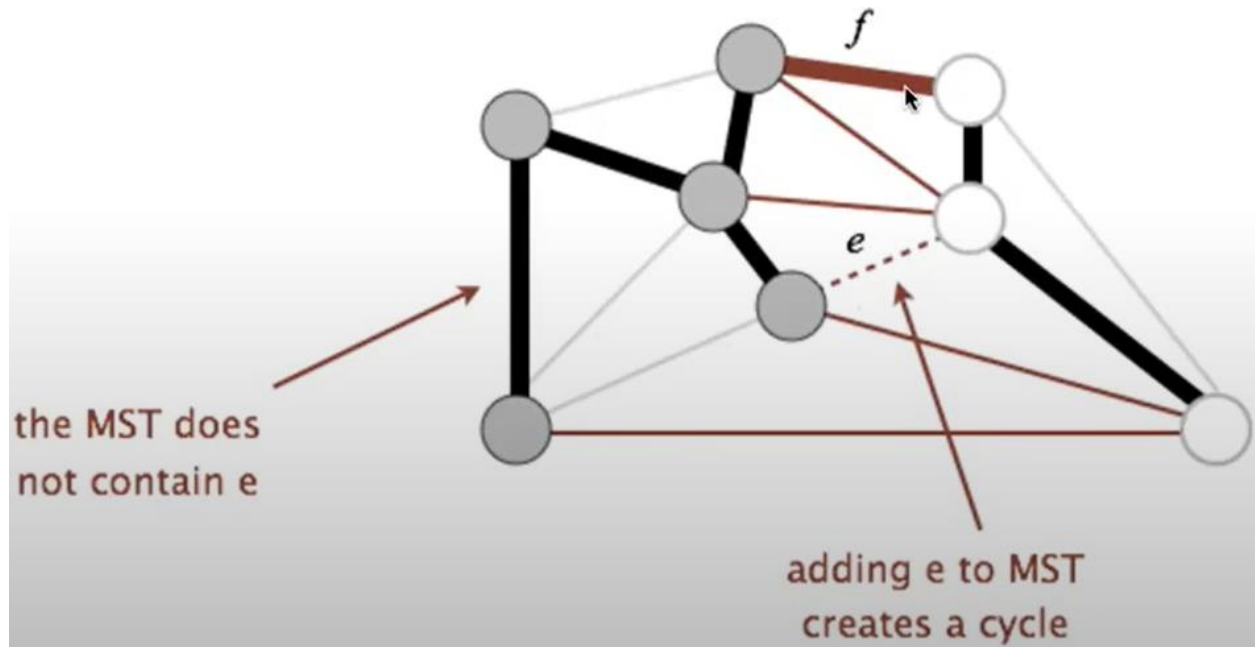
- A cut is an assignment of a graph's nodes to two non-empty sets.
- A **crossing edge** is an edge which connects a node from one set to a node from the other set.
- **Cut property**: Given any cut, minimum weight crossing edge is in the MST.
  - In case of a tie, pick any edge with equal weight. Different choices will result in different MSTs.





# Proof of the Cut Property

- Suppose that the minimum crossing edge  $e$  were not in the MST.
- • Adding  $e$  to the MST creates a cycle.
- • Some other edge  $f$  must also be a crossing edge.
- Replacing  $f$  with  $e$  results in a lower weight spanning tree.  
Contradiction!



# Generic MST Finding Algorithm

- Start with no edges in the MST.
  - Find a cut that has no crossing edges in the MST.
  - Add smallest crossing edge to the MST.
  - Repeat until  $V-1$  edges are added.
- This should work, but we need an efficient way of finding a cut with no crossing edges
  - Prim's Algorithm
  - Kruskal's Algorithm

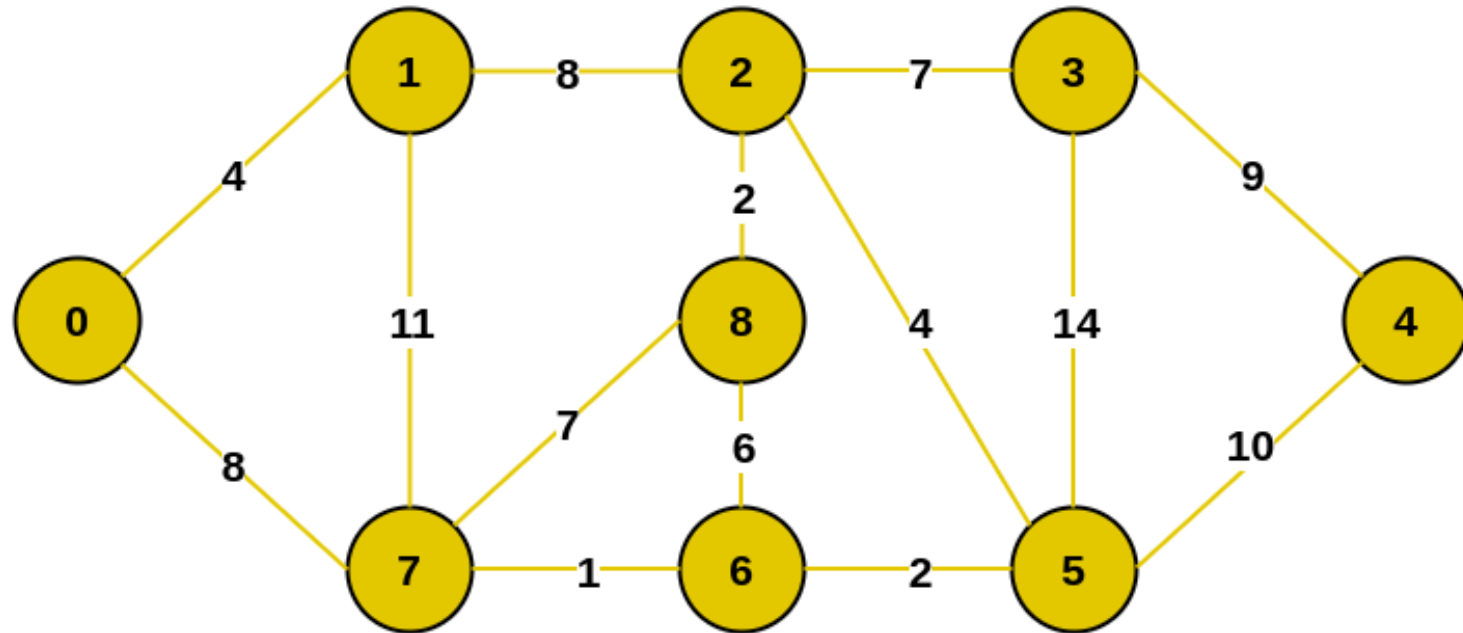
# Prim's Algorithm

- The algorithm starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, and the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.
- Step 1: Determine an arbitrary vertex as the starting vertex of the MST.
- Step 2: Follow steps 3 to 5 till there are vertices that are not included in the MST (known as fringe vertex).
- Step 3: Find edges connecting any tree vertex with the fringe vertices.
- Step 4: Find the minimum among these edges.
- Step 5: Add the chosen edge to the MST if it does not form any cycle.
- Step 6: Return the MST and exit

# Time Complexity of Prim's Algorithm

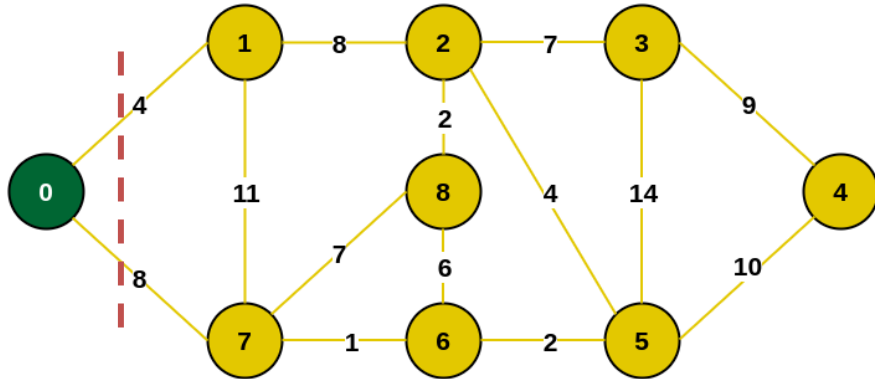
- Time complexity of Prim's algorithm depends on the implementation of the priority queue for finding the minimum weight edge.
- 1. Using an Adjacency Matrix
  - Priority Queue: Simple linear search for the minimum edge.
  - Time Complexity:  $O(V^2)$ . This is because finding the minimum edge in each iteration takes  $O(V)$ , and this is done  $V$  times.
- 2. Using an Adjacency List with a Binary Heap
  - Priority Queue: Binary heap for efficient minimum edge extraction and key updates.
  - Time Complexity:  $O((V+E) \log V)$ , where  $E$  is the number of edges.
- 3. Using an Adjacency List with a Fibonacci Heap
  - Priority Queue: Fibonacci heap for faster decrease-key operations.
  - Time Complexity:  $O(E+V \log V)$ .

# Prim's Algorithm Example 1

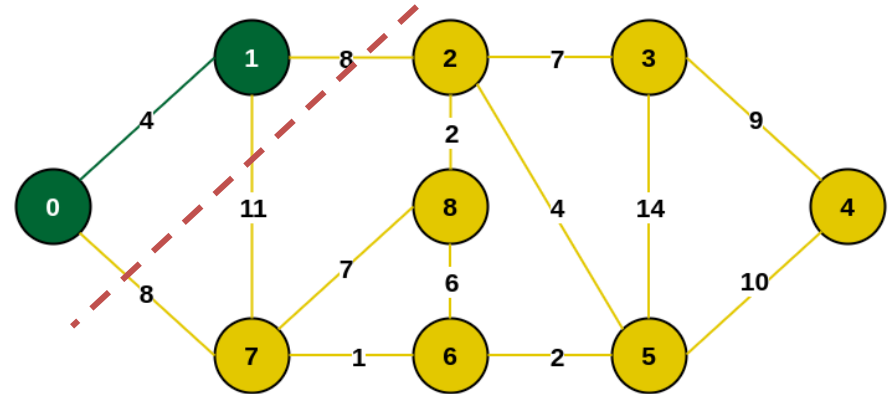


Example of a Graph

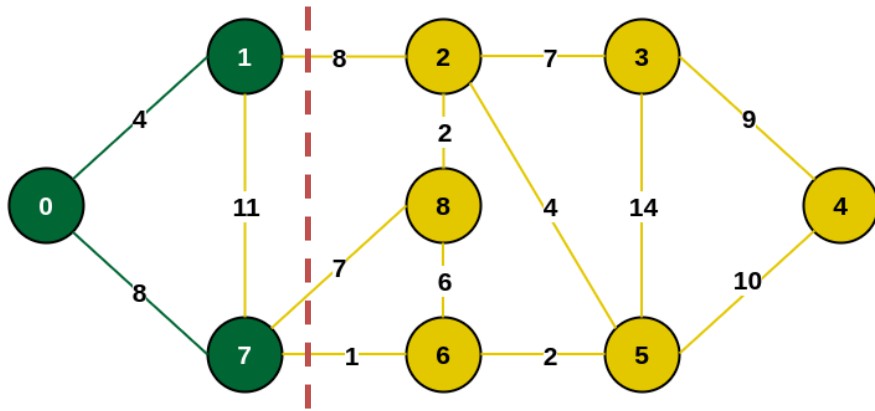
# Steps 1-4



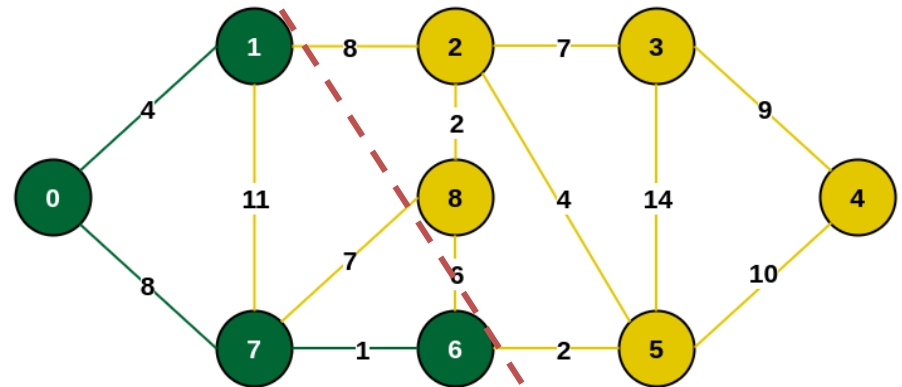
Select an arbitrary starting vertex. Here we have selected 0



Minimum weighted edge from MST to other vertices is 0-1 with weight 4



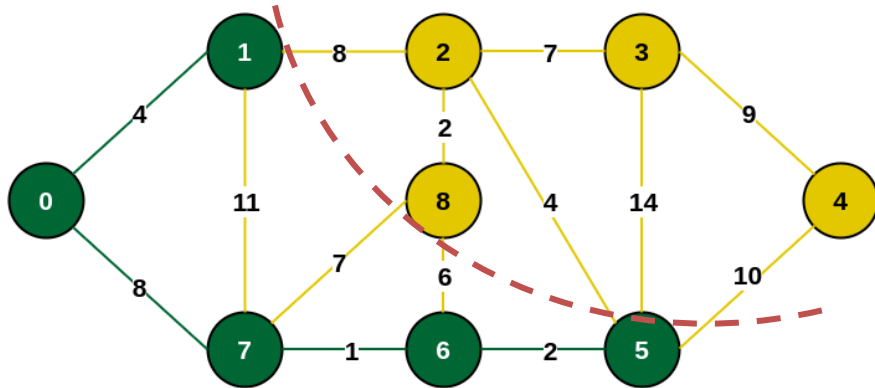
Minimum weighted edge from MST to other vertices is 0-7 with weight 8



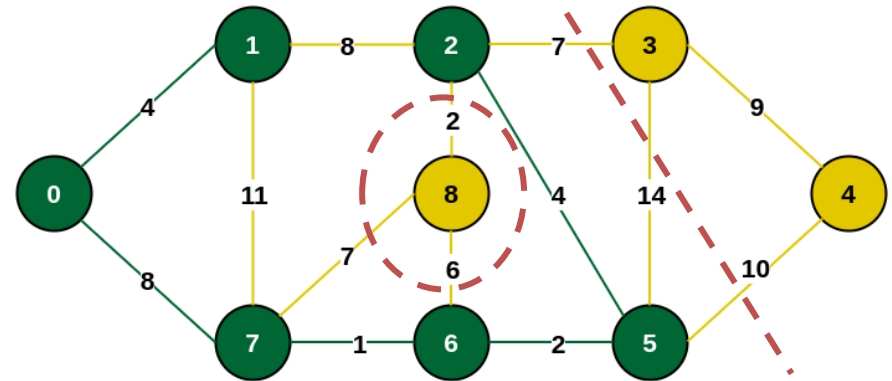
Minimum weighted edge from MST to other vertices is 7-6 with weight 1

Red dotted line denotes the cut between current MST and remaining vertices

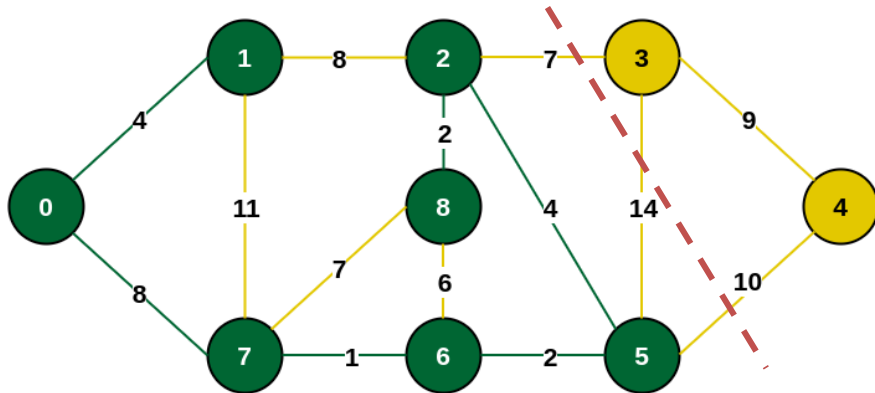
# Steps 5-8



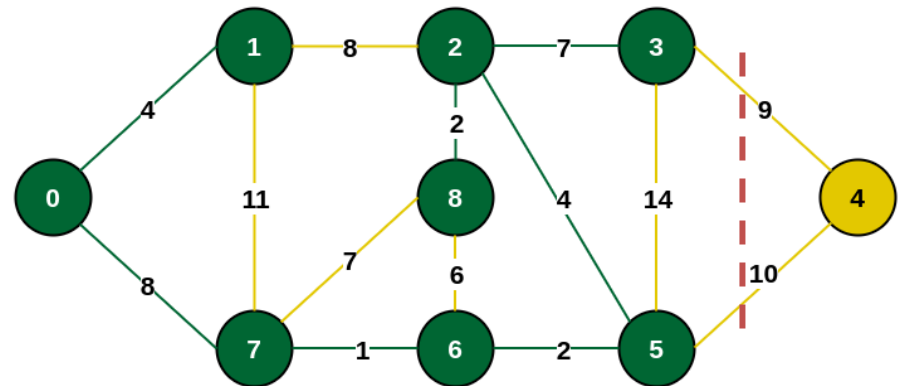
Minimum weighted edge from MST to other vertices is 6-5 with weight 2



Minimum weighted edge from MST to other vertices is 5-2 with weight 4



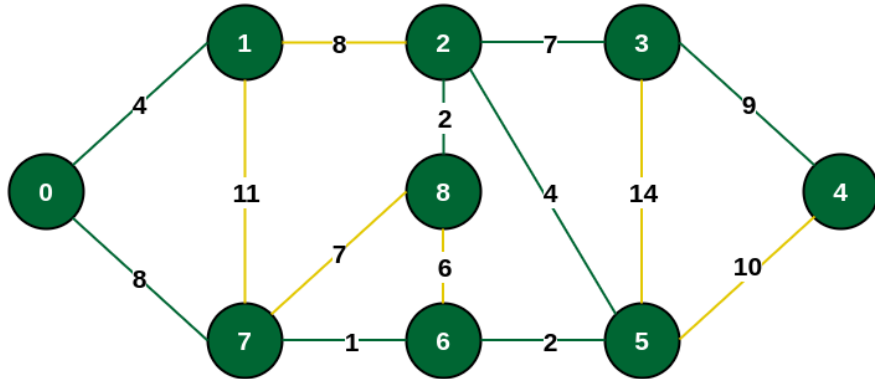
Minimum weighted edge from MST to other vertices is 2-8 with weight 2



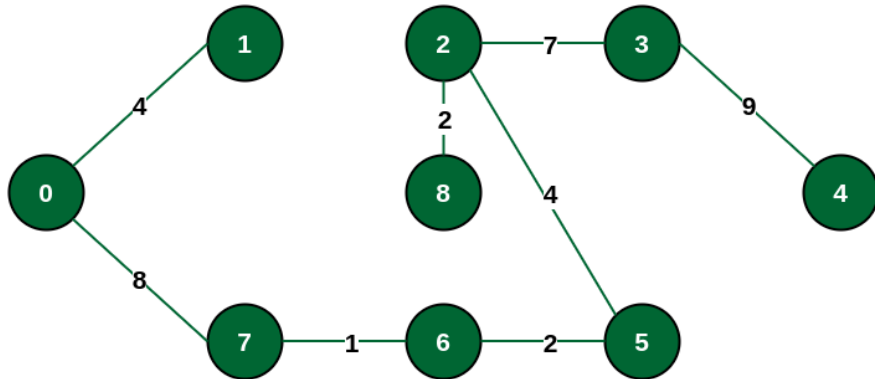
Minimum weighted edge from MST to other vertices is 2-3 with weight 7

Red dotted line denotes the cut between current MST and remaining vertices

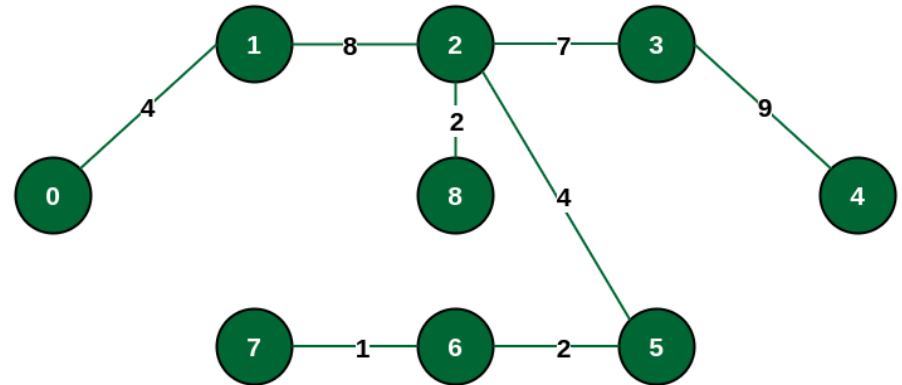
# Step 9



Minimum weighted edge from MST to other vertices is 3-4 with weight 9



The final structure of MST



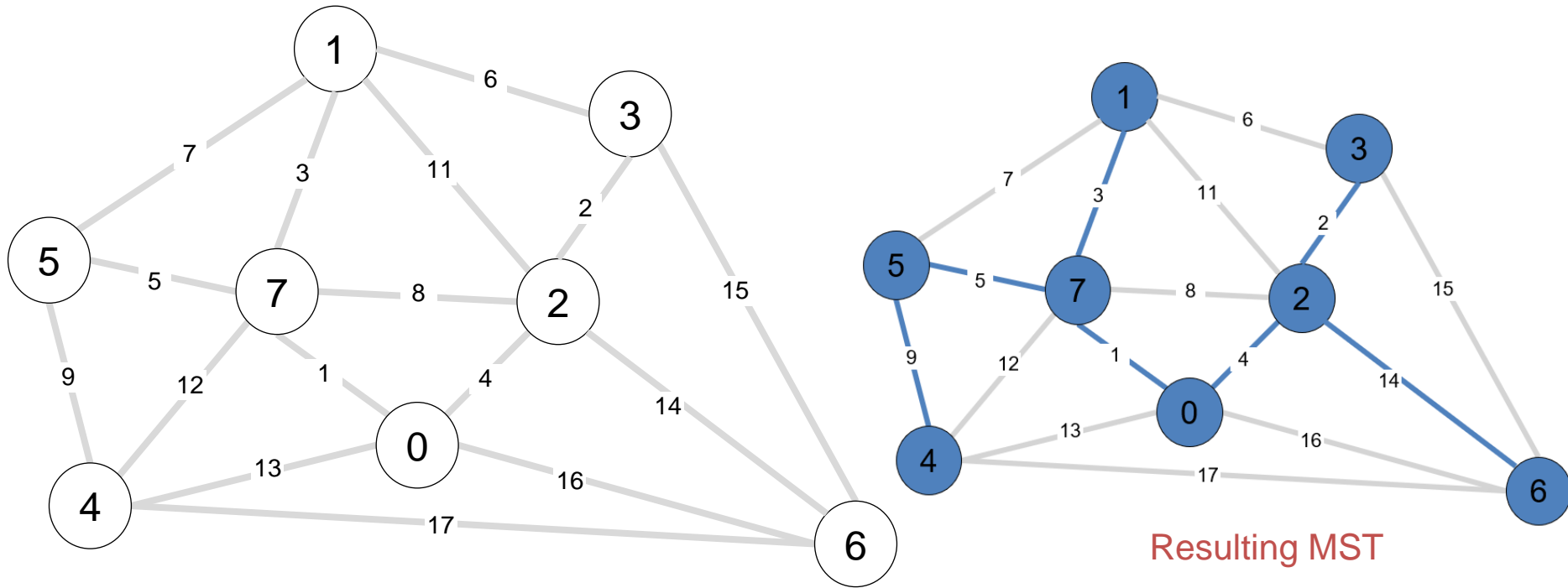
Alternative MST structure

If we had selected the edge {1, 2} in the third step then the MST would look like this one



# Prim's Algorithm Example 2

- Start with vertex 0 and greedily grow tree T.
- Add to T the min weight edge with exactly one endpoint in T.
- Repeat until  $V - 1$  edges.



Prim's algorithm in 2 minutes

<https://www.youtube.com/watch?v=cplfcGZmX7I>

# Proof of Correctness for Prim's Algorithm

- **Loop Invariant**

At each step, the edges selected by Prim's Algorithm form a subset of some MST.

- **Base Case**

- Starting from any vertex, the empty set of edges is trivially part of some MST
- The initial vertex forms our first component

- **Inductive Step**

Let  $A$  be our current tree and  $e$  be the next edge selected by Prim's:

1.  $e$  is the minimum weight edge connecting  $A$  to  $V-A$  (where  $V$  is the set of all vertices)
2.  $e$  crosses a cut between visited and unvisited vertices
3. By the cut property,  $e$  must be in some MST

- **Safety of Edge Selection**

- For any edge  $e$  selected by Prim's:

- It creates a cut  $(S, V-S)$  where  $S$  is the set of visited vertices
- $e$  is the lightest edge crossing this cut
- No lighter edge could exist (or it would have been chosen)
- Therefore,  $e$  is a safe addition to our growing MST

- **Conclusion**

- The algorithm is correct because:

- It maintains a single growing tree
- Each edge selection is optimal (by the cut property)
- It terminates with  $n-1$  edges (where  $n$  is the number of vertices)
- The result is connected (by construction)

- Therefore, Prim's Algorithm produces an MST

# Dijkstra's Algorithm vs. Prim's algorithm

- Similarities:
  - Both use a greedy approach.
  - They employ similar data structures, often using a priority queue.
  - The basic structures of both algorithms are very similar, with the main difference being in how they update vertex values
- Dijkstra's finds shortest paths, while Prim's constructs a minimum spanning tree (MST).
  - This is reflected in how they calculate and update vertex values:
  - In Dijkstra's algorithm, choose **the closest vertex to the source node (via a directed path)**, the distance to a vertex is relaxed to sum of the edge weight plus the distance to the previous vertex if it is smaller.
  - In Prim's algorithm, choose **the closest vertex to the tree (via an undirected edge)**, i.e., the vertex with the minimum weight of the edge connecting it to the MST.

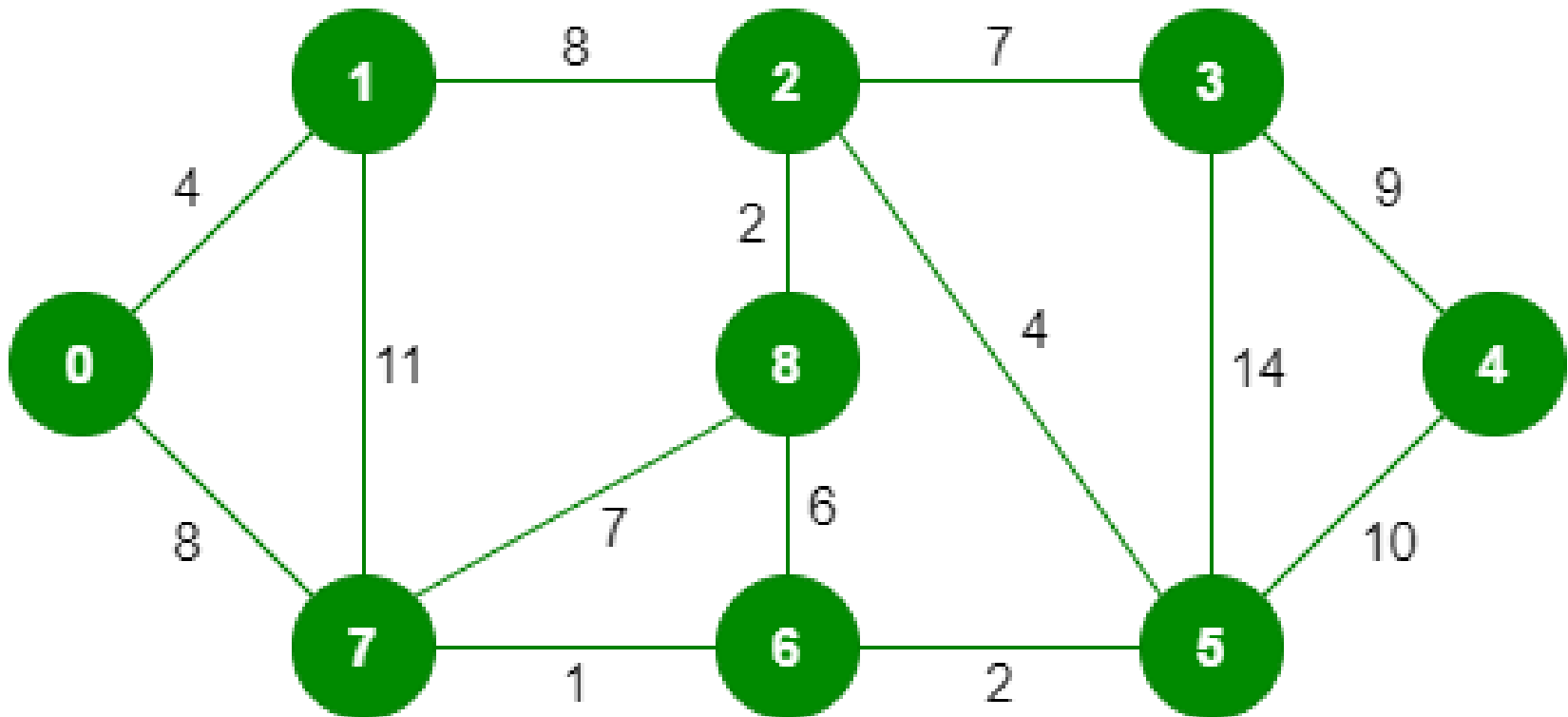
# Kruskal's Algorithm

- Sort all edges of the given graph in increasing order. Then it keeps on adding new edges and nodes in the MST if the newly added edge does not form a cycle. It picks the minimum weighted edge at first and the maximum weighted edge at last. Thus we can say that it makes a locally optimal choice in each step in order to find the optimal solution. Hence this is a Greedy Algorithm.
- 1. Sort all the edges in non-decreasing order of their weight.
- 2. Pick the lowest-cost edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it.
- 3. Repeat Step 2 until there are  $(V-1)$  edges in the spanning tree.

# Time Complexity of Kruskal's Algorithm

1. The time complexity of Kruskal's algorithm is determined by two main operations:
  1. Sorting the edges: Sorting all  $E$  edges by weight takes  $O(E \log E)$  using a comparison-based sorting algorithm.
  2. Union-Find operations: For each edge, the algorithm performs union and find operations to check for cycles, with time complexity  $O(\alpha(V))$ , where  $\alpha$  is the inverse Ackermann function, which grows slower than logarithmic function  $O(\log V)$ .
- Thus, the overall time complexity is dominated by the sorting step, resulting in time complexity of  $O(E \log E)$  or equivalently  $O(E \log V)$ , since  $E \leq V^2$  for a connected graph.

# Kruskal's Algorithm Example 1



e	W
7-6	1
8-2	2
6-5	2
0-1	4
2-5	4
8-6	6
2-3	7
7-8	7
0-7	8
1-2	8
3-4	9
5-4	10
1-7	11
14	14

Step 1

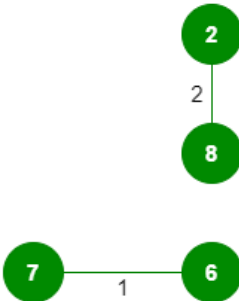
Add edge 7-6 in the MST



MST using Kruskal's algorithm

Step 2

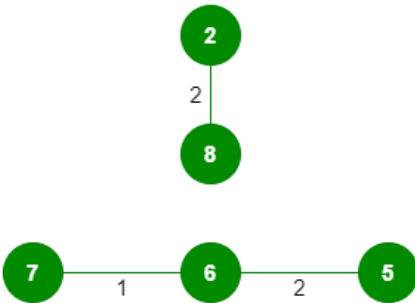
Add edge 8-2 in the MST



MST using Kruskal's algorithm

Step 3

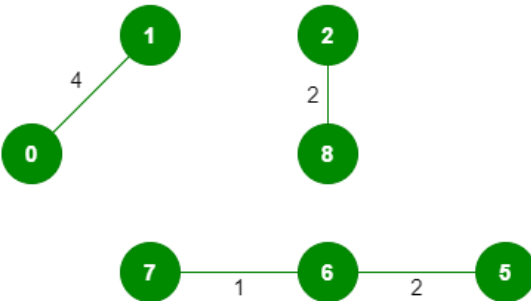
Add edge 6-5 in the MST



MST using Kruskal's algorithm

Step 4

Add edge 0-1 in the MST

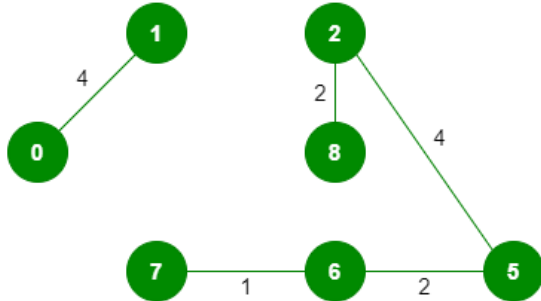


MST using Kruskal's algorithm

e	W
7-6	1
8-2	2
6-5	2
0-1	4
2-5	4
8-6	6
2-3	7
7-8	7
0-7	8
1-2	8
3-4	9
5-4	10
1-7	11
14	14

**Step  
5**

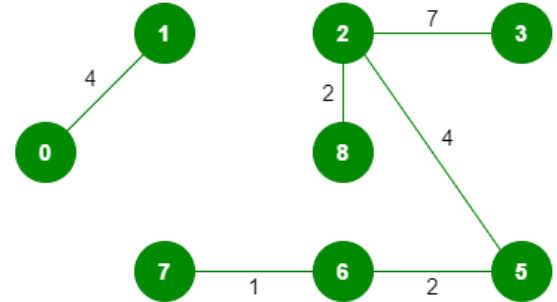
Add edge 2-5 in the MST



MST using Kruskal's algorithm

**Step  
6**

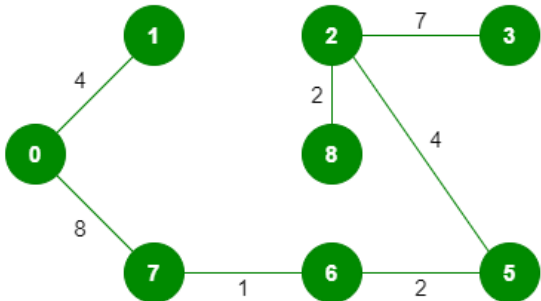
Add edge 2-3 in the MST as 8-6 can't be added



MST using Kruskal's algorithm

**Step  
7**

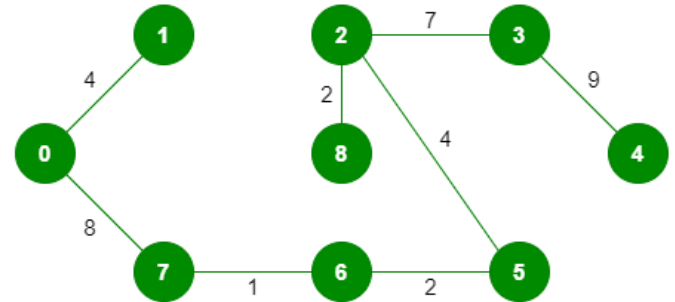
Add edge 0-7 in the MST as 7-8 can't be added



MST using Kruskal's algorithm

**Step  
8**

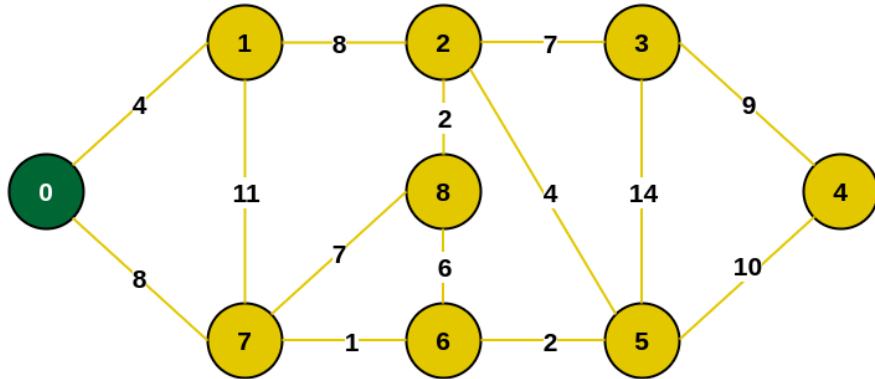
Add edge 3-4 in the MST. It completes the MST



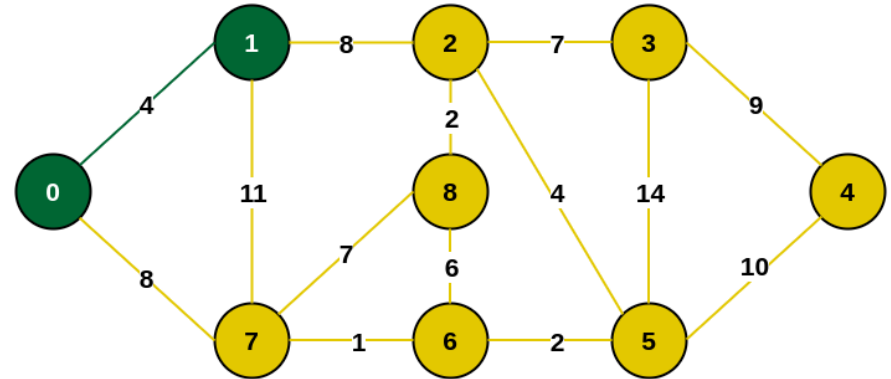
MST using Kruskal's algorithm



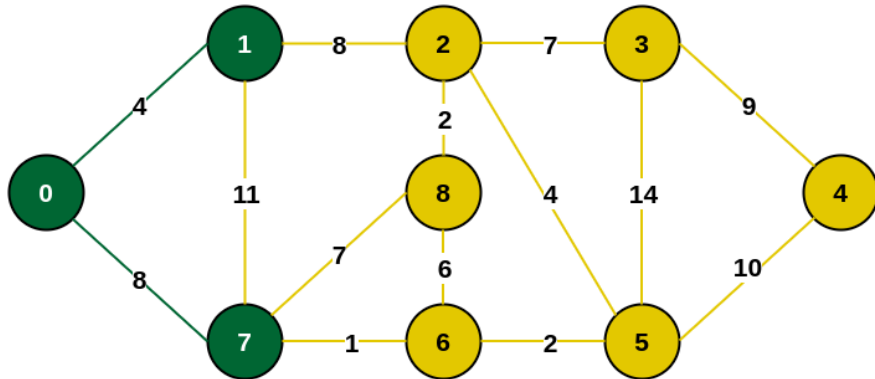
# Steps 1-4



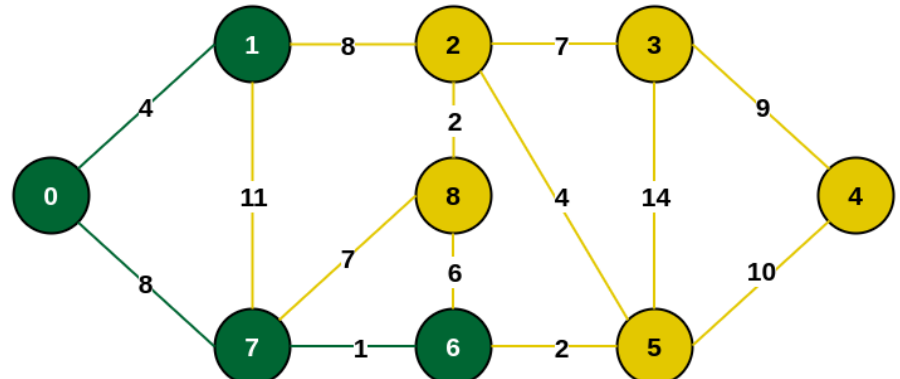
Select an arbitrary starting vertex. Here we have selected 0



Minimum weighted edge from MST to other vertices is 0-1 with weight 4



Minimum weighted edge from MST to other vertices is 0-7 with weight 8

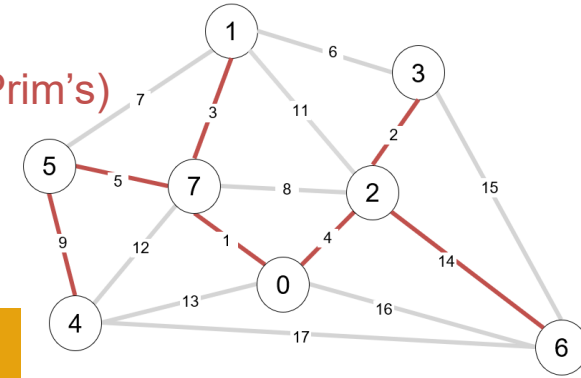


Minimum weighted edge from MST to other vertices is 7-6 with weight 1

Resulting MST  
(same result as Prim's)

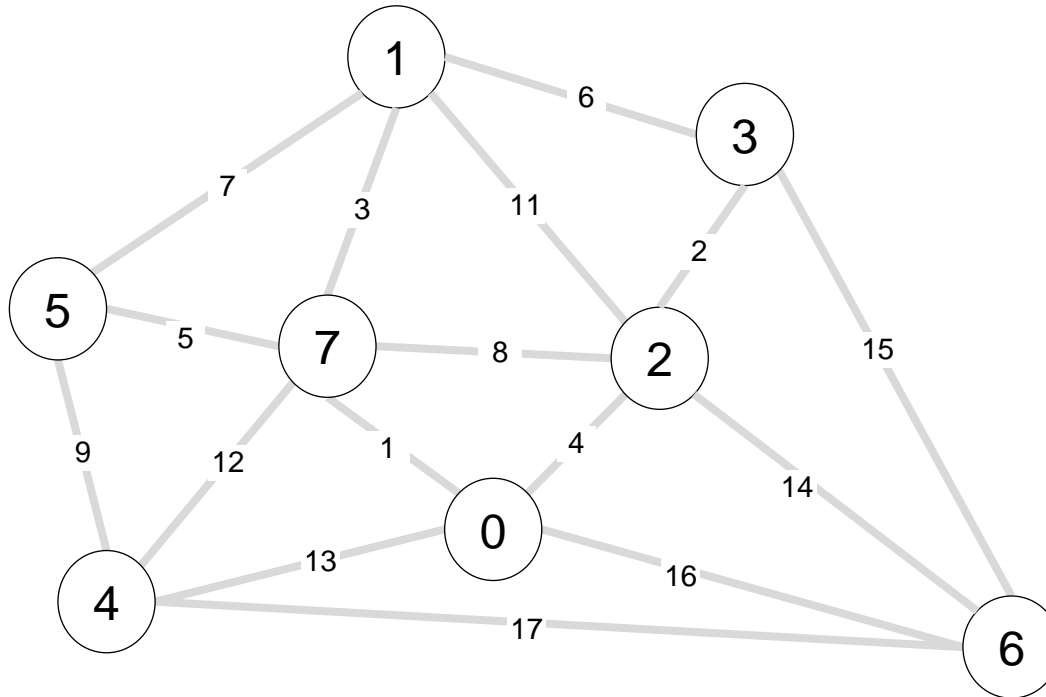
# Kruskal's Algorithm Example 2

- Consider edges in ascending order of weight.
- Add next edge to tree T unless doing so would create a cycle.



graph edges sorted by weight

does not create a cycle



creates a cycle

0 - 7	1	←
2 - 3	2	←
1 - 7	3	←
0 - 2	4	←
5 - 7	5	←
1 - 3	6	←
1 - 5	7	←
2 - 7	8	←
4 - 5	9	←
1 - 2	11	←
4 - 7	12	←
0 - 4	13	←
2 - 6	14	←
3 - 6	15	←
0 - 6	16	←
4 - 6	17	←

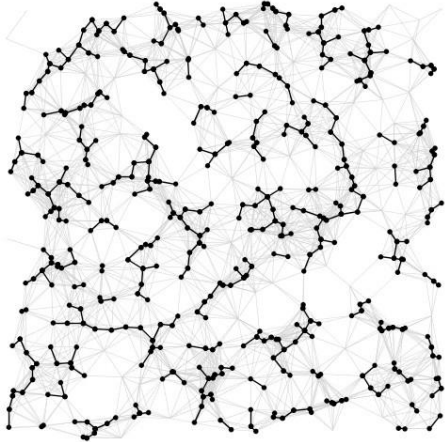
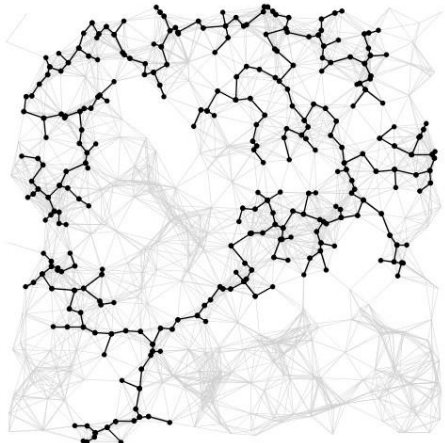
Kruskal's algorithm in 2 minutes

<https://www.youtube.com/watch?v=71UQH7Pr9kU>

# Proof of Correctness for Kruskal's Algorithm

- **Loop Invariant**  
At each step, the set of edges chosen by Kruskal's Algorithm is a subset of some MST.
- **Base Case**
  - Initially, the empty set of edges is trivially a subset of any MST
- **Inductive Step**
- Let  $e$  be the next edge selected by Kruskal's Algorithm. Two key facts:
  1.  $e$  is the minimum weight edge that doesn't create a cycle with previously selected edges
  2. The edges selected so far form a forest  $F$
- **Safety of Edge Selection**
- Consider the components  $C1$  and  $C2$  that edge  $e$  connects:
  - $e$  is the minimum weight edge between these components
  - Any path between  $C1$  and  $C2$  in the MST must use at least one edge
  - That edge cannot be lighter than  $e$  (or Kruskal would have chosen it)
  - Therefore, we can safely include  $e$  in our solution
- **Conclusion**
- The algorithm terminates when:
  - It has selected exactly  $n-1$  edges (where  $n$  is the number of vertices)
  - The result is connected (due to the selection process)
  - Each edge is chosen optimally (by the cut property)
- Therefore, Kruskal's Algorithm produces an MST

# Summary

algorithm	visualization	bottleneck	running time
<b>Kruskal</b>		sorting union-find	$E \log V$
<b>Prim</b>		priority queue	$E \log V$

# References

- Prim's Algorithm for Minimum Spanning Tree (MST) | GeeksforGeeks
  - <https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/>
  - <https://www.youtube.com/watch?v=eB61LXLZVqs>
- Kruskal's Algorithm for Minimum Spanning Tree | GeeksforGeeks
  - <https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/>
  - [https://www.youtube.com/watch?v=3rrNH\\_AizMA](https://www.youtube.com/watch?v=3rrNH_AizMA)
- Prim's algorithm in 2 minutes
  - <https://www.youtube.com/watch?v=cplfcGZmX7I>
- Kruskal's Algorithm in 2 minutes
  - <https://www.youtube.com/watch?v=71UQH7Pr9kU&t=1s>
- vid10 kruskals vs prims
  - <https://www.youtube.com/watch?v=vmWSnkBVvQ0>

# Full-Length Lectures

- [CSE 373WI24] Lecture 16: Minimum Spanning Trees
  - [https://www.youtube.com/watch?v=9dGPQqE0dUw&list=PLEcoVsAaONjd5n69K84sSmAuvTrTQT\\_Nl&index=15](https://www.youtube.com/watch?v=9dGPQqE0dUw&list=PLEcoVsAaONjd5n69K84sSmAuvTrTQT_Nl&index=15)