

# DATA STRUCTURE AND ALGORITHM

## CLASS 9

---

Seongjin Lee

Updated: 2017-03-06  
DSA\_2017\_09

insight@gnu.ac.kr  
<http://resourceful.github.io>  
Systems Research Lab.  
GNU



# Table of contents

- 1. Graph Operations
  - 1.1 Path Finding
  - 1.2 Connected Graph
  - 1.3 Spanning Tree
  - 1.4 Greedy Strategy
  - 1.5 Kruskal's Algorithm

# GRAPH OPERATIONS

---

# Some of the Graph Problems are

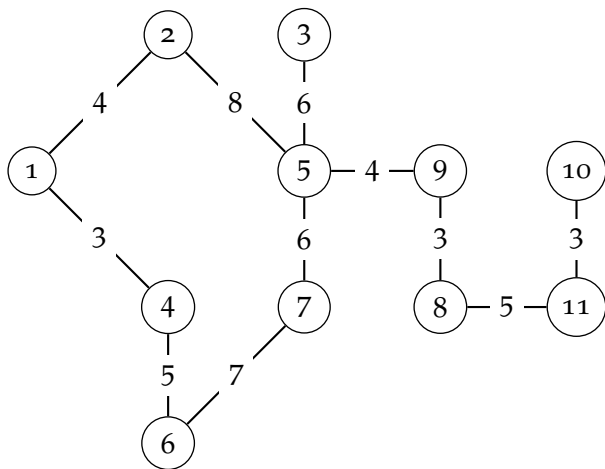
- Path Finding
- Connectedness
- Spanning tree

## GRAPH OPERATIONS : PATH FINDING

---

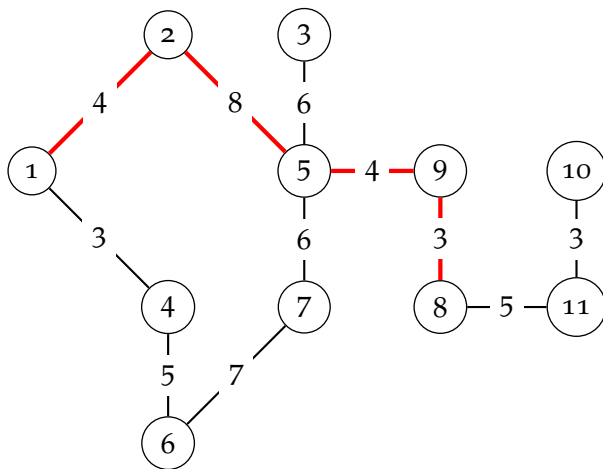
# Path Finding

- Path length between 1 and 8



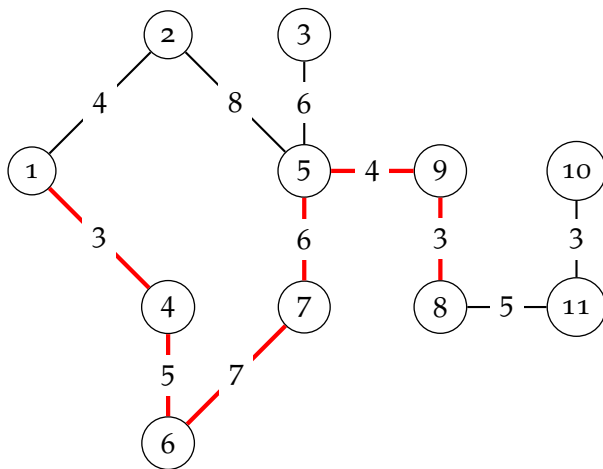
# Path Finding

- Edges (1, 2), (2, 5), (5, 9), and (9, 8) length = 19



# Path Finding

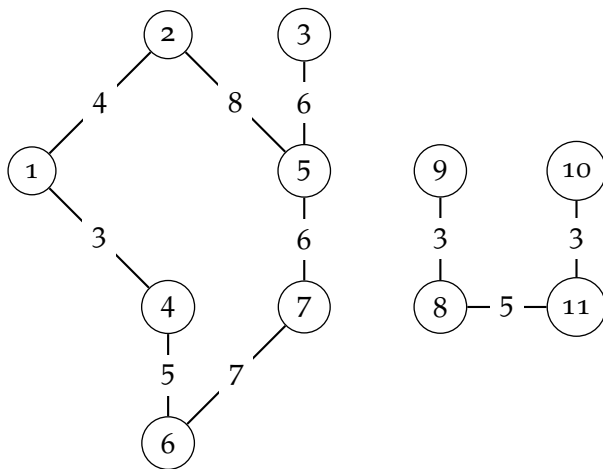
- Edges (1, 4), (4, 6), (6, 7), (5, 9) and (9, 8) length = 28





# Example of No Path

- No path between 4 to 11

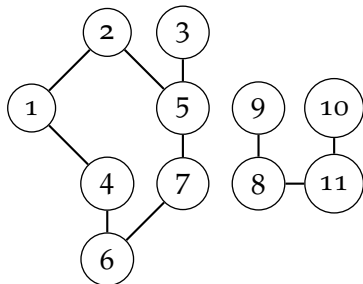


# GRAPH OPERATIONS : CONNECTED GRAPH

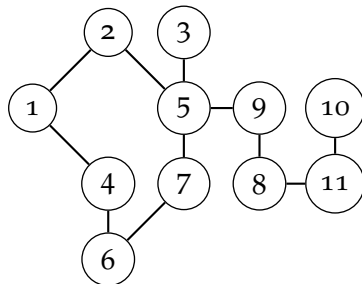
---

# Connected Graph

- Undirected graph
- There is a path between every pair of vertices
- A directed graph  $G = (V, E)$  is **strongly connected** if, for every pair of vertices  $u, v$  in  $V$ , there is a directed path from  $u$  to  $v$  and also from  $v$  to  $u$

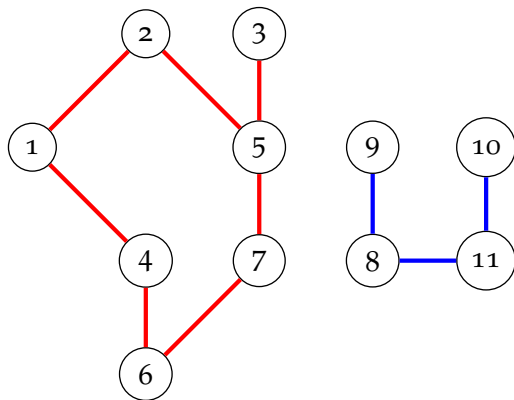


Not connected graph



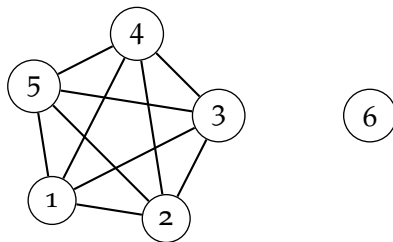
Connected Graph

# Connected Components



# Connected Component

- A connected component is a *maximal subgraph* in which all vertices are reachable from every other vertices.
  - *maximal* means that it is the largest possible subgraph
  - Cannot add vertices and edges from original graph and retain connectedness.
  - A connected graph has exactly 1 component.



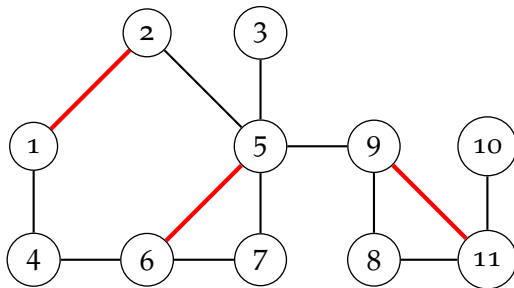
# Connectedness

There are two types of connected components in digraphs

- Strong Components
  - maximal subgraph in which there is a path from every vertex to every vertex following all the edges in the direction they are pointing
- Weak Components
  - maximal subgraph which would be connected if we ignore the direction of the edges

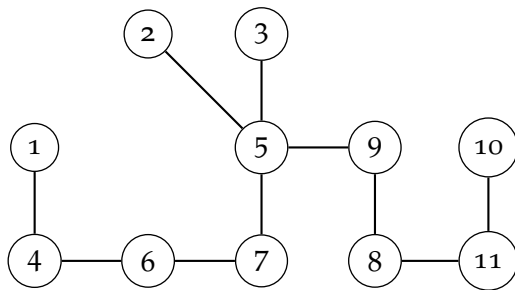
# Cycles and Connectedness

Removal of an edge that is on a cycle does not affect connectedness



# Cycles and Connectedness

Connected subgraph with all vertices and minimum number of edges has no cycles



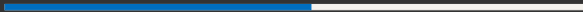


# Tree

A tree can be thought of as connected graph that has no cycles

- $n$  vertex connected graph with  $n - 1$  edges

# GRAPH OPERATIONS : SPANNING TREE

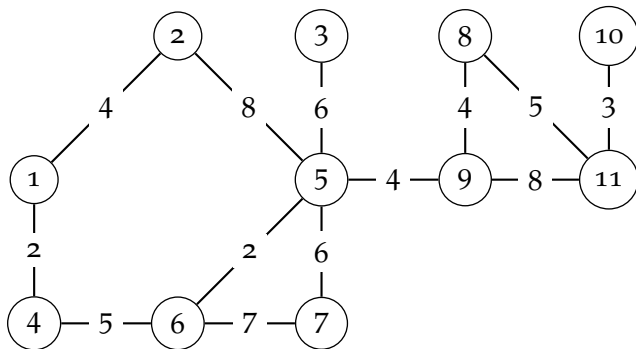


# Spanning Tree

- Subgraph that includes all vertices of the original graph.
- Subgraph is a tree.
  - If original graph has  $n$  vertices, the spanning tree has  $n$  vertices and  $n - 1$  edges.

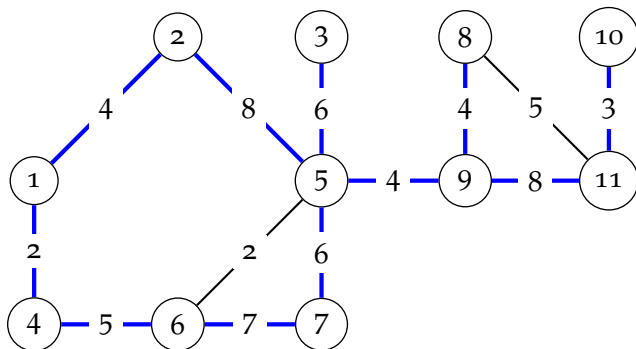
# Minimum Cost Spanning Tree

- Tree cost is sum of edge weights/costs



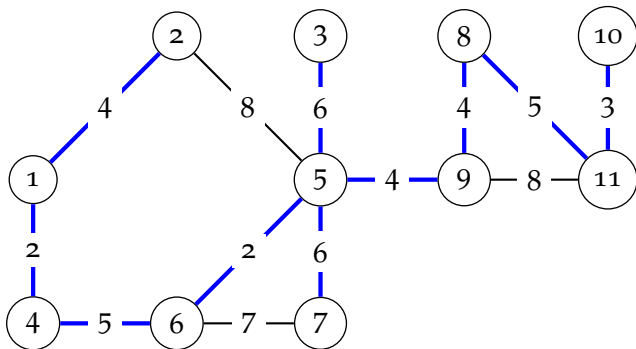
# A Spanning Tree

- Spanning Tree cost is 51



# A Spanning Tree

- Spanning Tree cost is 41

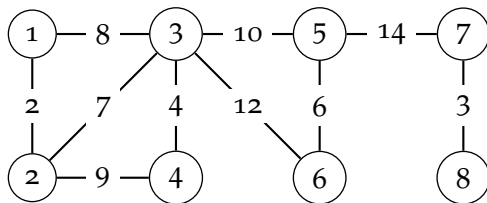


# Minimum-Cost Spanning Tree

- weighted connected undirected graph
- spanning tree
- cost of spanning tree is sum of edge costs
- find spanning tree that has minimum cost

# Example

- Network has 10 edges
- Spanning tree has only  $n - 1 = 7$  edges
- Need to either select 7 edges or discard 3





## GRAPH OPERATIONS : GREEDY STRATEGY



# Edge Selection Greedy Strategies

- Start with an  $n - vertex, 0 - edge$  forest. Consider edges in ascending order of cost. Select edge if it does not form a cycle together with already selected edges.
  - Kruskal's algorithm
- Start with a  $1 - vertex$  tree and grow it into an  $n - vertex$  tree by repeatedly adding a vertex and an edge. When there is a choice, add a least cost edge.
  - Prim's algorithm
- Start with an  $n - vertex$  forest. Each component/tree selects a least cost edge to connect to another component/tree. Eliminate duplicate selections and possible cycles. Repeat until only 1 component/tree is left.
  - Sollin's algorithm

# Edge Rejection Greedy Strategies

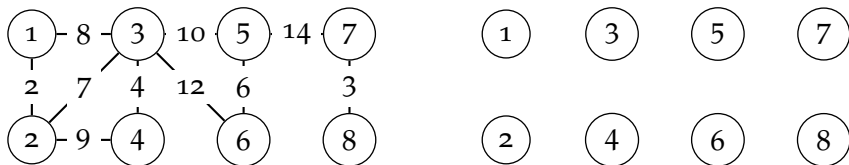
- Start with the connected graph. Repeatedly find a cycle and eliminate the highest cost edge on this cycle. Stop when no cycles remain.
- Consider edges in descending order of cost. Eliminate an edge provided this leaves behind a connected graph.

## GRAPH OPERATIONS : KRUSKAL'S ALGORITHM

---

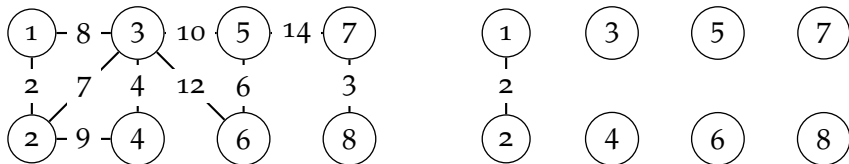
# Kruskal's Algorithm

- Start with a forest that has no edges
- Consider edges in ascending order of cost.



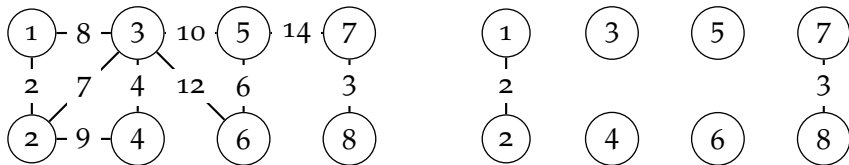
# Kruskal's Algorithm

- Start with a forest that has no edges
- Consider edges in ascending order of cost.
- Edge (1,2) is considered first and added to the forest.



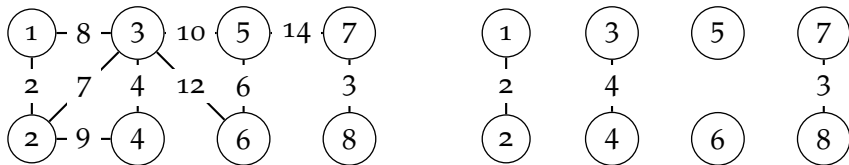
# Kruskal's Algorithm

- Start with a forest that has no edges
- Consider edges in ascending order of cost.
- Edge (1,2) is considered first and added to the forest.
- Edge (7,8)



# Kruskal's Algorithm

- Start with a forest that has no edges
- Consider edges in ascending order of cost.
- Edge (1,2) is considered first and added to the forest.
- Edge (7,8)    ○ Edge (3,4)



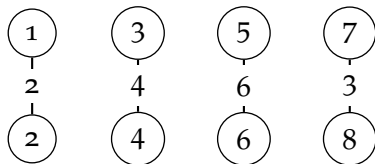
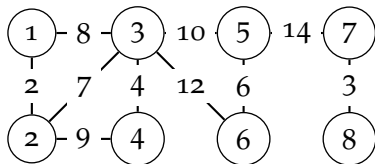


# Kruskal's Algorithm

- Start with a forest that has no edges
- Consider edges in ascending order of cost.
- Edge (1,2) is considered first and added to the forest.

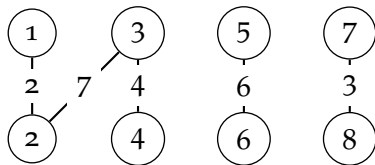
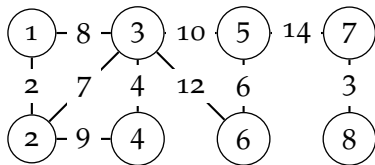
○ Edge (7,8)    ○ Edge (3,4)

○ Edge (5,6)



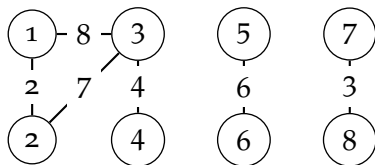
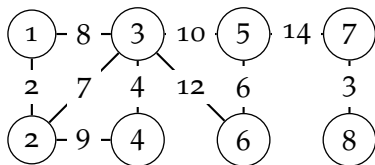
# Kruskal's Algorithm

- Start with a forest that has no edges
- Consider edges in ascending order of cost.
- Edge (1,2) is considered first and added to the forest.
- Edge (7,8)
- Edge (3,4)
- Edge (5,6)
- Edge (2,3)



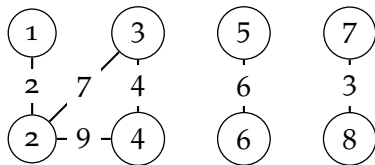
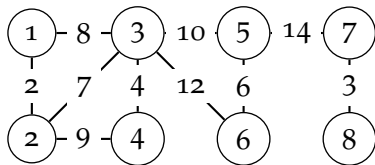
# Kruskal's Algorithm

- Start with a forest that has no edges
- Consider edges in ascending order of cost.
- Edge (1,2) is considered first and added to the forest.
- Edge (7,8)
- Edge (3,4)
- Edge (5,6)
- Edge (2,3)
- Edge (1,3) creates cycle (rejected)



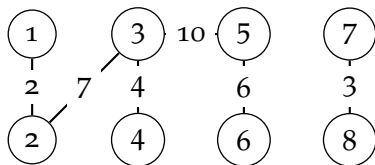
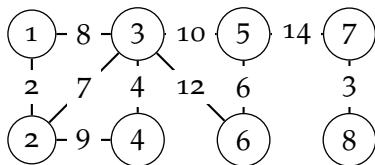
# Kruskal's Algorithm

- Start with a forest that has no edges
- Consider edges in ascending order of cost.
- Edge (1,2) is considered first and added to the forest.
- Edge (7,8)
- Edge (3,4)
- Edge (5,6)
- Edge (2,3)
- Edge (1,3) creates cycle (rejected)
- Edge (2,4) creates cycle



# Kruskal's Algorithm

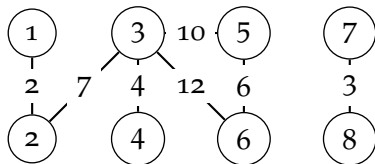
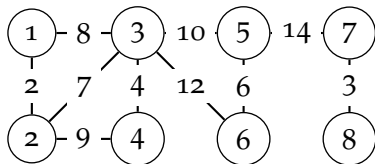
- Start with a forest that has no edges
- Consider edges in ascending order of cost.
- Edge (1,2) is considered first and added to the forest.
- Edge (7,8)
- Edge (3,4)
- Edge (5,6)
- Edge (2,3)
- Edge (1,3) creates cycle (rejected)
- Edge (2,4) creates cycle
- Edge (3,5)



# Kruskal's Algorithm

- Start with a forest that has no edges
- Consider edges in ascending order of cost.
- Edge (1,2) is considered first and added to the forest.

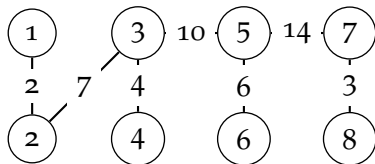
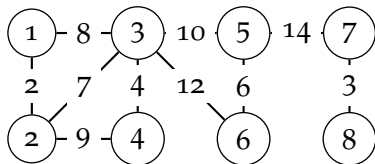
- Edge (7,8)
- Edge (3,4)
- Edge (5,6)
- Edge (2,3)
- Edge (1,3) creates cycle (rejected)
- Edge (2,4) creates cycle
- Edge (3,5)
- Edge (3,6) creates cycle



# Kruskal's Algorithm

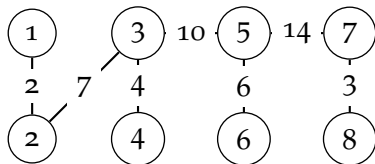
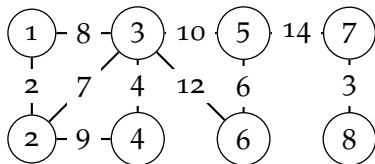
- Start with a forest that has no edges
- Consider edges in ascending order of cost.
- Edge (1,2) is considered first and added to the forest.

- Edge (7,8)
- Edge (3,4)
- Edge (5,6)
- Edge (2,3)
- Edge (1,3) creates cycle (rejected)
- Edge (2,4) creates cycle
- Edge (3,5)
- Edge (3,6) creates cycle
- Edge (5,7)



# Kruskal's Algorithm

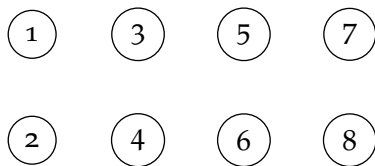
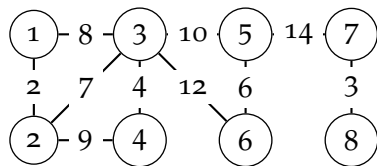
- $n - 1$  edges have been selected and no cycle formed, so we must have a spanning tree
  - The cost is 46
- The minimum cost spanning tree is unique when all edge costs are different





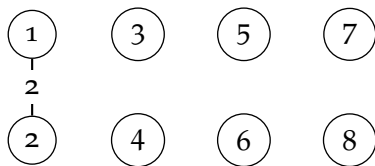
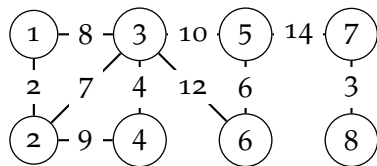
# Prim's Algorithm

- Start with any single vertex tree
- Get a 2-vertex tree by adding a cheapest edge
- Get a 3-vertex tree by adding a cheapest edge
- Grow the tree one edge at a time until the tree has  $n - 1$  edges (and hence has all  $n$  vertices)



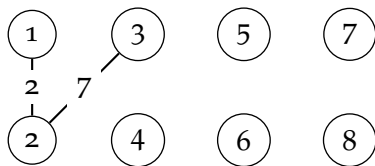
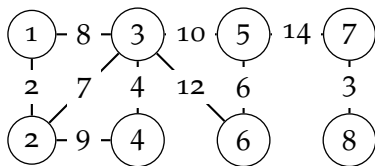
# Prim's Algorithm

- Start with any single vertex tree
- Get a 2-vertex tree by adding a cheapest edge
- Get a 3-vertex tree by adding a cheapest edge
- Grow the tree one edge at a time until the tree has  $n - 1$  edges (and hence has all  $n$  vertices)



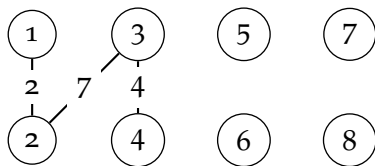
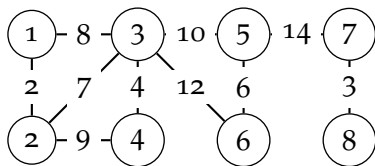
# Prim's Algorithm

- Start with any single vertex tree
- Get a 2-vertex tree by adding a cheapest edge
- Get a 3-vertex tree by adding a cheapest edge
- Grow the tree one edge at a time until the tree has  $n - 1$  edges (and hence has all  $n$  vertices)



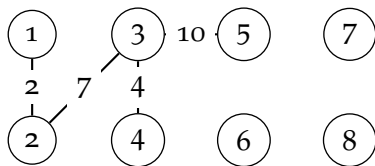
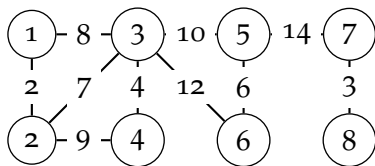
# Prim's Algorithm

- Start with any single vertex tree
- Get a 2-vertex tree by adding a cheapest edge
- Get a 3-vertex tree by adding a cheapest edge
- Grow the tree one edge at a time until the tree has  $n - 1$  edges (and hence has all  $n$  vertices)



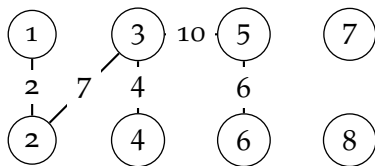
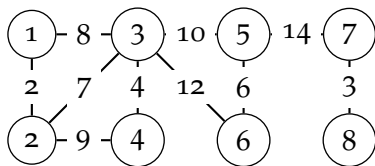
# Prim's Algorithm

- Start with any single vertex tree
- Get a 2-vertex tree by adding a cheapest edge
- Get a 3-vertex tree by adding a cheapest edge
- Grow the tree one edge at a time until the tree has  $n - 1$  edges (and hence has all  $n$  vertices)



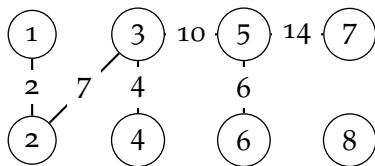
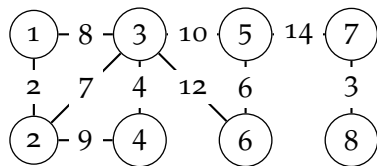
# Prim's Algorithm

- Start with any single vertex tree
- Get a 2-vertex tree by adding a cheapest edge
- Get a 3-vertex tree by adding a cheapest edge
- Grow the tree one edge at a time until the tree has  $n - 1$  edges (and hence has all  $n$  vertices)



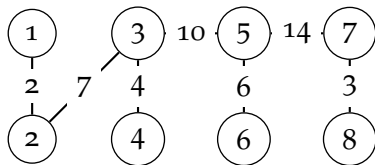
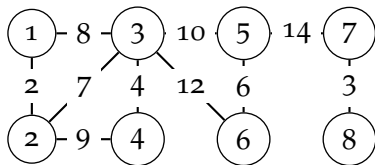
# Prim's Algorithm

- Start with any single vertex tree
- Get a 2-vertex tree by adding a cheapest edge
- Get a 3-vertex tree by adding a cheapest edge
- Grow the tree one edge at a time until the tree has  $n - 1$  edges (and hence has all  $n$  vertices)



# Prim's Algorithm

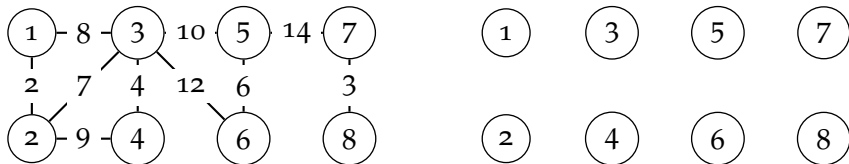
- Start with any single vertex tree
- Get a 2-vertex tree by adding a cheapest edge
- Get a 3-vertex tree by adding a cheapest edge
- Grow the tree one edge at a time until the tree has  $n - 1$  edges (and hence has all  $n$  vertices)





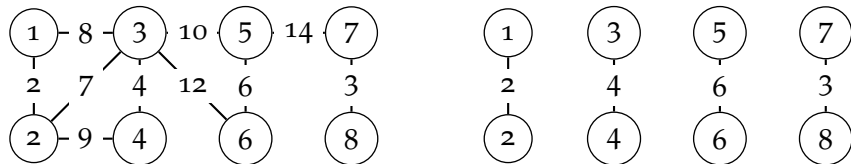
# Sollin's Algorithm

- Start with a forest that has no edges.
- Each component selects a least cost edge with which to connect to another component.
- Duplicate selections are eliminated.
- Cycles are possible when the graph has some edges that have the same cost.
- Each component that remains selects a least cost edge with which to connect to another component.
- Beware of duplicate selections and cycles.



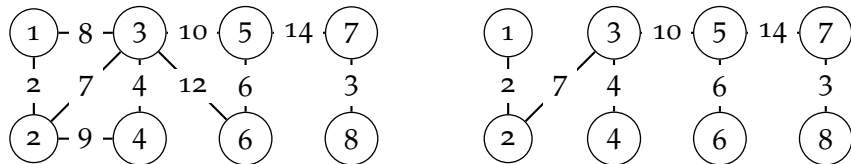
# Sollin's Algorithm

- Start with a forest that has no edges.
- Each component selects a least cost edge with which to connect to another component.
- Duplicate selections are eliminated.
- Cycles are possible when the graph has some edges that have the same cost.
- Each component that remains selects a least cost edge with which to connect to another component.
- Beware of duplicate selections and cycles.



# Sollin's Algorithm

- Start with a forest that has no edges.
- Each component selects a least cost edge with which to connect to another component.
- Duplicate selections are eliminated.
- Cycles are possible when the graph has some edges that have the same cost.
- Each component that remains selects a least cost edge with which to connect to another component.
- Beware of duplicate selections and cycles.



# Greedy Minimum-Cost Spanning Tree Algorithms

- Can prove that all result in a minimum-cost spanning tree.
- Prim's Algorithm is the fastest
  - $O(n^2)$  using an implementation similar to that of Dijkstra's shortest-path algorithm
  - $O(e + n \log n)$  using a Fibonacci heap
- Kruskal's algorithm uses **union-find trees** to run in  $O(n + e \log e)$  time
  - $\text{union}(x, y)$  joins two subsets containing  $x$  and  $y$  into a single subset
  - $\text{find}(x)$  determines the subset with the element  $x$

## Exmple: Union-find

Assume the following set  $S = \{1, 2, 3, 4, 5, 6\}$  and create a six independent sets:  $\{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}$ .

After performing  $\text{union}(1, 4)$  and  $\text{union}(2, 5)$ , then we have  $\{1, 4\}, \{5, 2\}, \{3\}, \{4\}$

After running  $\text{union}(2, 1)$  and  $\text{union}(3, 6)$ , then we have  $\{1, 2, 4, 5\}, \{3, 6\}$

