CSC508 Data Structures

Topic 14 : Graph 2

Graph Algorithms

Compiled & edited by: Zahid Zainal

Recap

- Kőnigsberg Bridge Problem
- Graph Definition
- Graph Representations

Topic Structure

- Graph Traversal
- Shortest Path Algorithm
- Minimal Spanning Tree

Learning Outcomes

- At the end of this lesson, students should be able to:
  - Describe graph traversal algorithms
  - Implement shortest path in graph
  - Explain minimal spanning tree

Operation on Graphs

- Basic operations performed on a graph:
  - Creating the graph
    - Nodes are usually fixed after creation, but not always
    - Adding or removing edges
  - Determine whether edge (i,j) exists, and knowing its related data (weight, label, etc.)
  - Know all vertices adjacent to a given vertex
  - Printing graph data

Other Graph Operations

- Traversal: Given G=(V,E) and a vertex v, find all w ∈ V, such that w is reachable from v
- Finding a path between two vertices that is optimum in some sense (shortest, least cost, etc.)
- Building minimum spanning trees
  - A tree with optimum cost that spans all vertices
- Topological Sorting of a directed graph
  - Linear ordering of vertices which resolves dependency

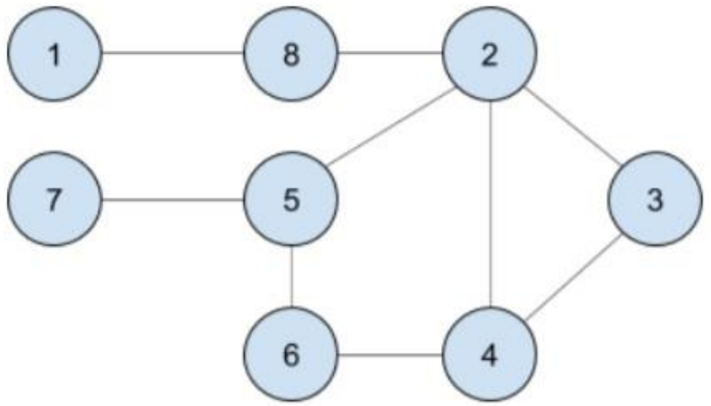
Graph Traversal

- Traversing a graph is similar to traversing a binary tree, except that:
  - A graph might have cycles
  - Might not be able to traverse the entire graph starting from a single vertex
- Each vertex is visited exactly once.
- Most common graph traversal algorithms:
  - Depth First Traversal (DFT)
    - Similar to preorder tree-traversal
  - Breadth First Traversal (BFT)
    - Similar to level order tree-traversal

Depth First Traversal (DFT)

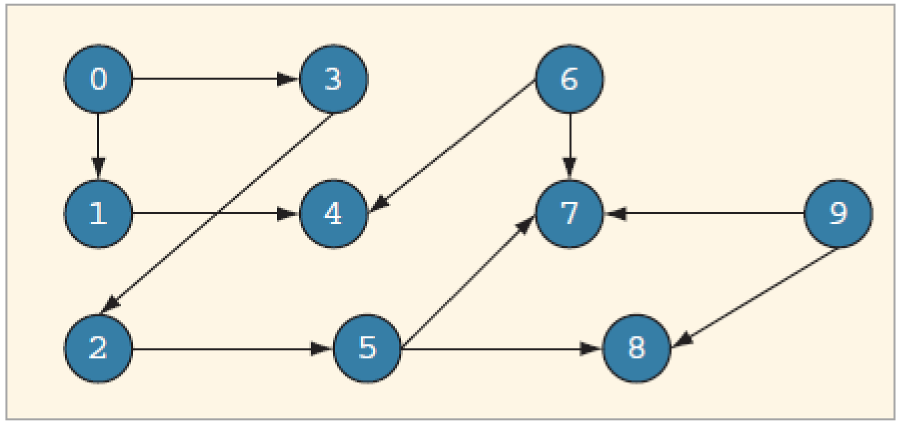
- Similar to binary tree preorder traversal
- Depth first traversal at a given node, v
  - Mark node v as visited
  - Visit the node
  - for each vertex u adjacent to v
    - if u is not visited
    - start the depth first traversal at u

DFT for Undirected Graph



- Depth first visit ordering of the vertices of graph G, starting at vertex **3**: 3, 2, 4, 6, 5, 7, 8, 1
- If there are two or more adjacent vertices, select the vertex with the lowest value to visit next (unless other requirement m

DFT for Directed Graph



- Depth first visit ordering of the vertices of graph G, starting at vertex **0**: 0, 1, 4, 3, 2, 5, 7, 8.
- Vertices 6 and 9 are not accessible due to the edge direction.

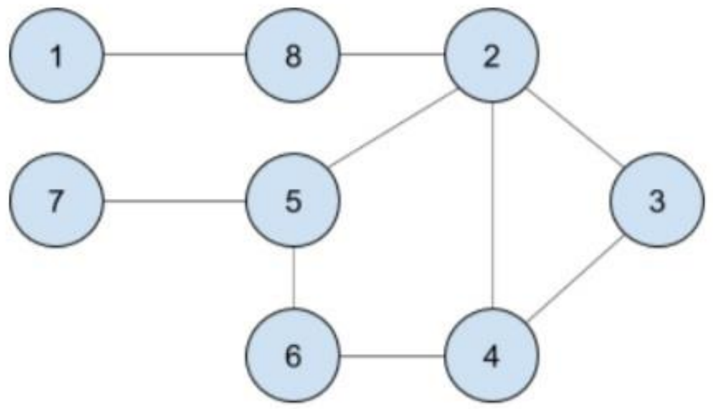
Breadth First Traversal (BFT)

- Breadth first traversal of a graph is similar to traversing a binary tree level by level
  - Nodes at each level
    - Visited from left to right
  - All nodes at any level i
    - Visited before visiting nodes at level i+1
- Use a queue to implement the breadth first traversal algorithm

•Declare a queue and insert the starting vertex.
•Initialize a **visited** array and mark the starting vertex as visited.
•Follow the below process till the queue becomes empty:

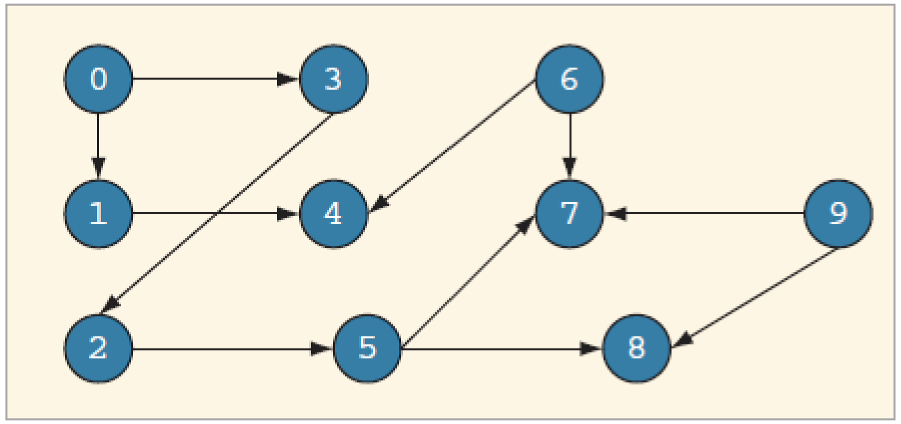
- Remove the first vertex of the queue.
- Mark that vertex as visited.
- Insert all the unvisited neighbors of the vertex into the queue.

BFT for Undirected Graph



- Breadth first visit ordering of the vertices of graph G, starting at vertex **3**: 3, 2, 4, 5, 8, 6, 7, 1
- If there are two or more adjacent vertices, enqueue the vertex with the lowest value into the queue first(unless other requirement m

BFT for Directed Graph



- Depth first visit ordering of the vertices of graph G, starting at vertex **0**: 0, 1, 3, 4, 2, 5, 7, 8.
- Vertices 6 and 9 are not accessible due to the edge direction.

Shortest Path Algorithm

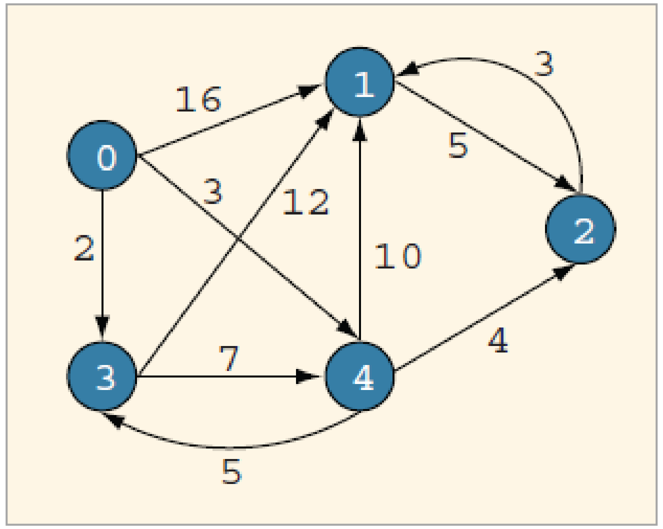
- Finding "optimum" path between two vertices
- Dijkstra's algorithm
  - Between source node and all other nodes
  - When graph weights are positive or zero, with cyclic or acyclic graphs
  - Use Bellman's algorithm with acyclic directed graphs and possibly negative weights
- Floyd-Warshall's algorithm
  - General: shortest path between any two nodes
  - Positive and negative weights, but not negative cycles

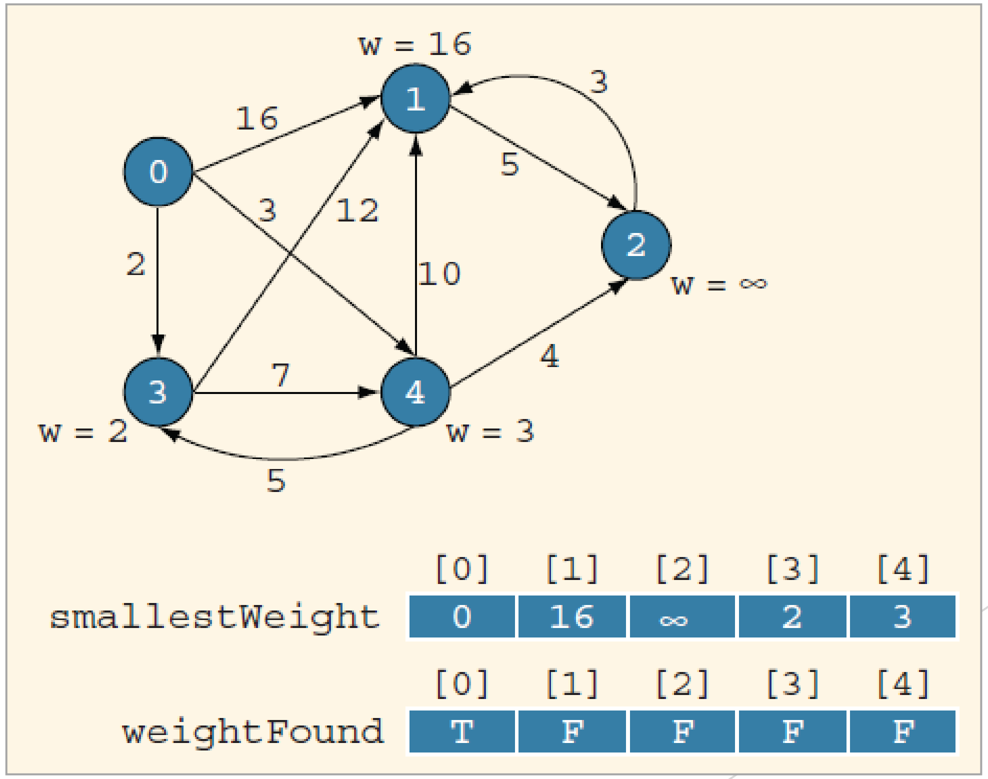
Djikstra's Algorithm

```
1. Initialize the array smallestWeight so that
    smallestWeight[u] = weights[vertex, u]
2. Set smallestWeight[vertex] = 0
3. Find the vertex, v, that is closest to vertex for
   which the shortest path has not been determined
4. Mark v as the (next) vertex for which the smallest
   weight is found
5. For each vertex w in G, such that the shortest path
   from vertex to w has not been determined and an edge
   (v, w) exists, if the weight of the path to w via v is
   smaller than its current weight, update the weight of
   w to the weight of v + the weight of the edge (v, w)
```

Djikstra's Algorithm Simulation

- In this graph, we want to find shortest path from vertex 0, i.e. variable value of `vertex` is 0:



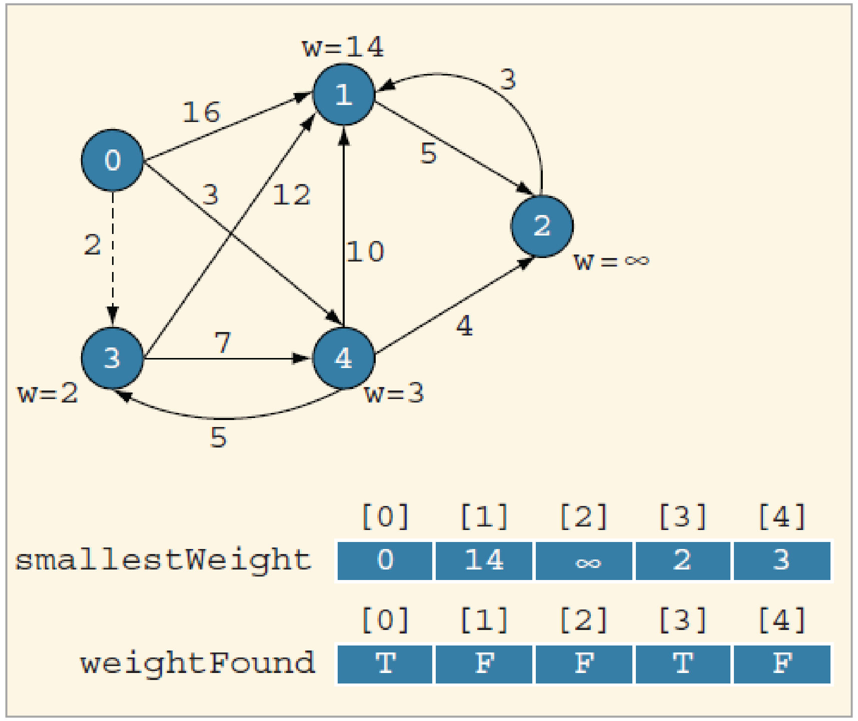


	[0]	[1]	[2]	[3]	[4]
smallestWeight	0	16	∞	2	3

	[0]	[1]	[2]	[3]	[4]
weightFound	T	F	F	F	F

After initialization, **3** is closest adjacent to `vertex`

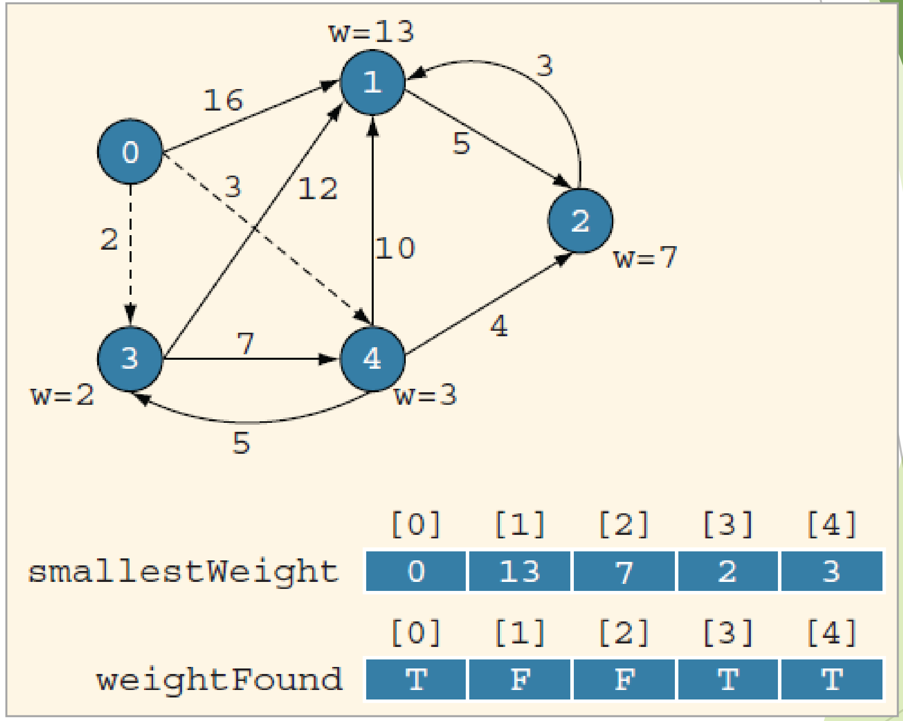
Djikstra’s Algorithm Simulation (cont.)



	[0]	[1]	[2]	[3]	[4]
smallestWeight	0	14	∞	2	3
weightFound	T	F	F	T	F

Mark **3** as final and update smallest weights for vertices reachable from vertex **3**:

1 becomes 14, and **4** remains as is

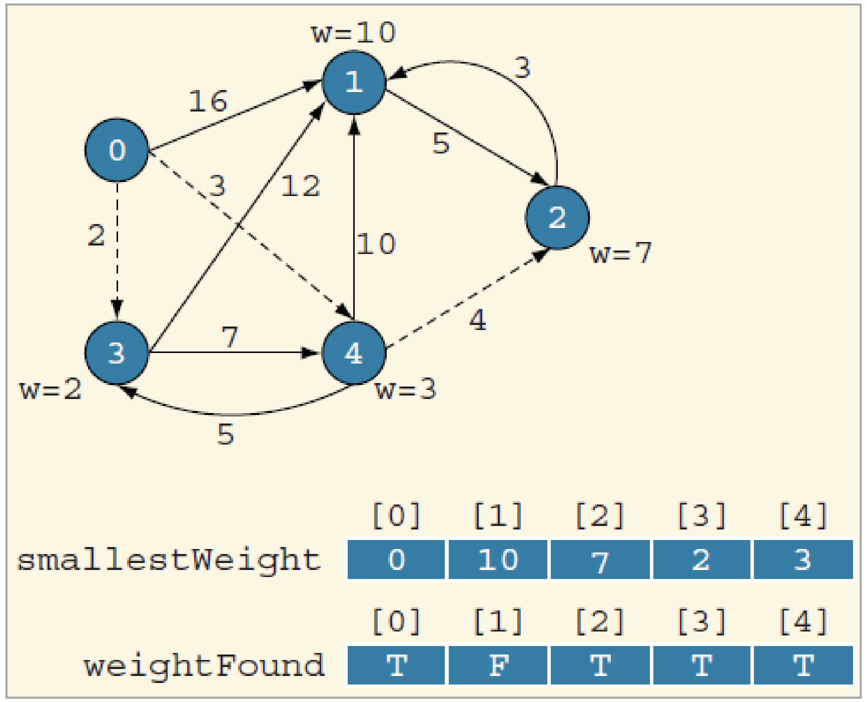


	[0]	[1]	[2]	[3]	[4]
smallestWeight	0	13	7	2	3
weightFound	T	F	F	T	T

Next vertex with smallest non-final weight is **4**. Mark it as final and update smallest weights:

1 becomes 13, and **2** becomes 7

Djikstra's Algorithm Simulation (cont.)

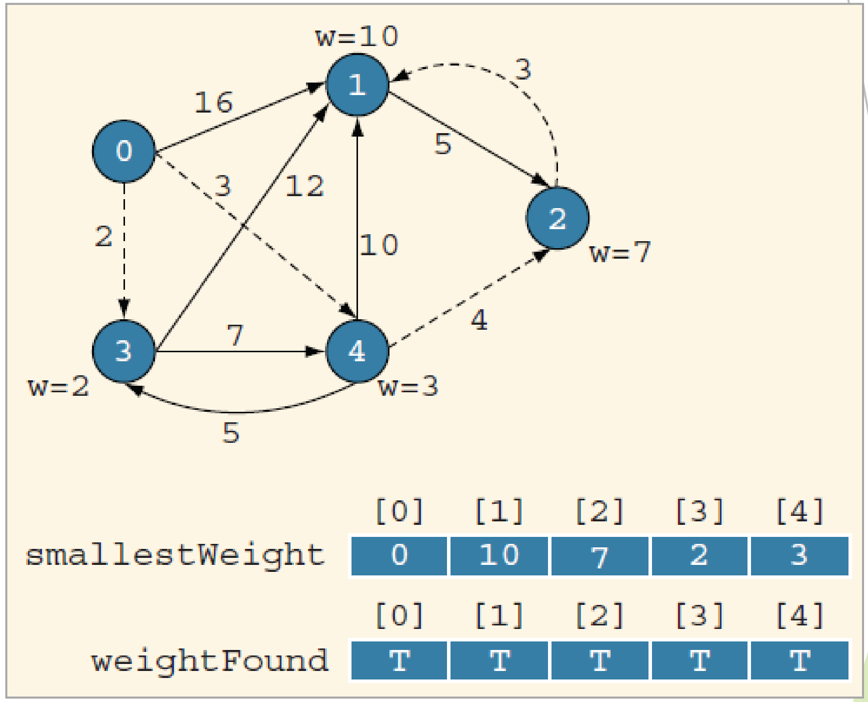


	[0]	[1]	[2]	[3]	[4]
smallestWeight	0	10	7	2	3

	[0]	[1]	[2]	[3]	[4]
weightFound	T	F	T	T	T

Next vertex with smallest non-final weight is 1. Mark it as final and update smallest weights:

No updates in this case



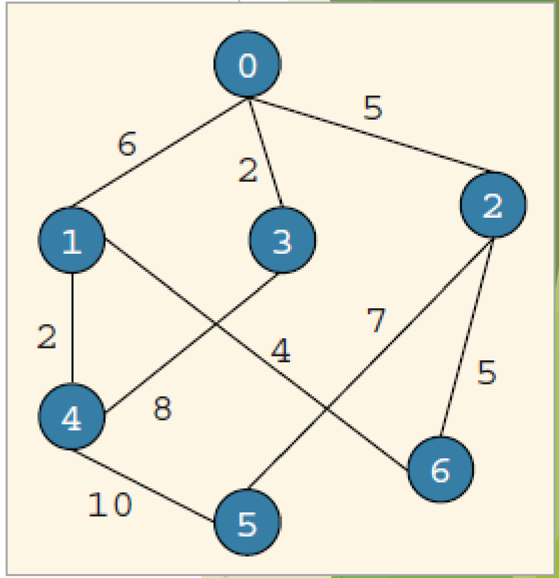
	[0]	[1]	[2]	[3]	[4]
smallestWeight	0	10	7	2	3

	[0]	[1]	[2]	[3]	[4]
weightFound	T	T	T	T	T

Algorithm stops since smallest path cost has been determined for all vertices

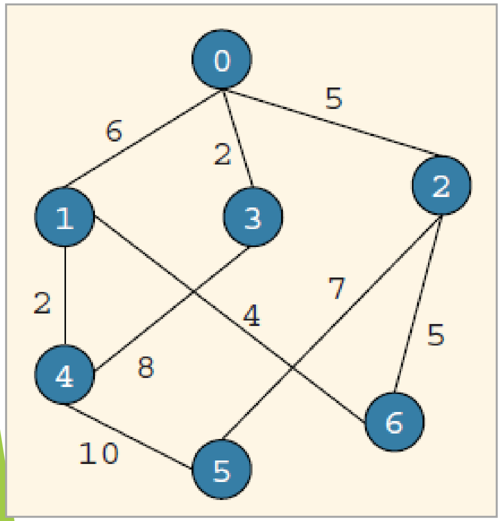
Minimal Spanning Tree

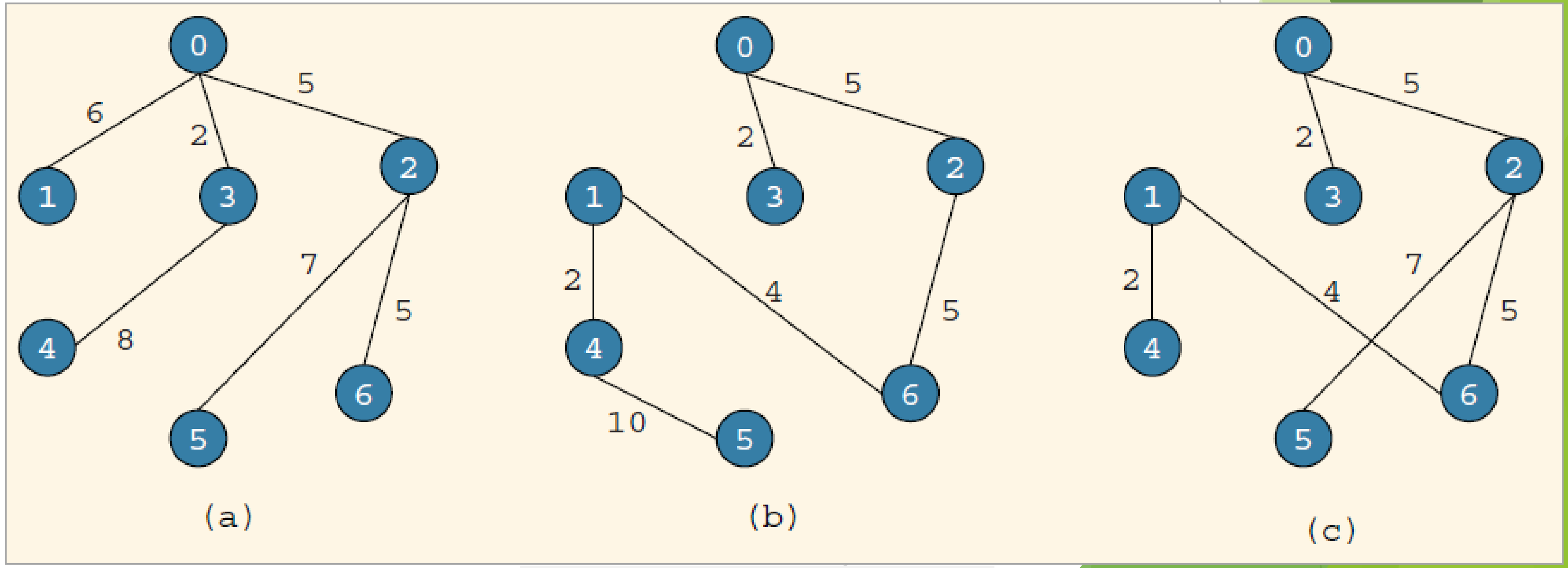
- Minimal Spanning Tree (MST) is a subset of edges of a connected weighted undirected graph that connects all the vertices together with the minimum possible total edge weight.
- Example: Graph, G, is a route map for an airline company. Due to financial hardship, company needs to shut down the maximum number of connections and still be able to fly from one city to another



Minimal Spanning Tree (cont.)

- Several spanning trees, connecting all destinations (vertices) can be extracted.
  - But only one can be the minimal spanning tree (miminum sum of all weights)





Spanning trees of graph G

MST Notation

- (Free) tree T : simple graph such that if u and v are two vertices in T, then there is a unique path from u to v
- Rooted tree: tree in which a particular vertex is designated as a root
- Weighted tree: tree in which weight is assigned to the edges in T
- If T is a weighted tree, the weight of T, denoted by W(T), is the sum of the weights of all the edges in T
- A tree T is called a spanning tree of graph G if T is a subgraph of G such that V(T) = V(G),
- All the vertices of G are in T.

MST Notation (cont.)

- Theorem: A graph G has a spanning tree if and only if G is connected.
- In order to determine a spanning tree of a graph, the graph must be connected.
- Let G be a weighted graph. A minimal spanning tree of G is a spanning tree with the minimum weight.
- Kruskal’s algorithm is one algorithm used to fine an MST.

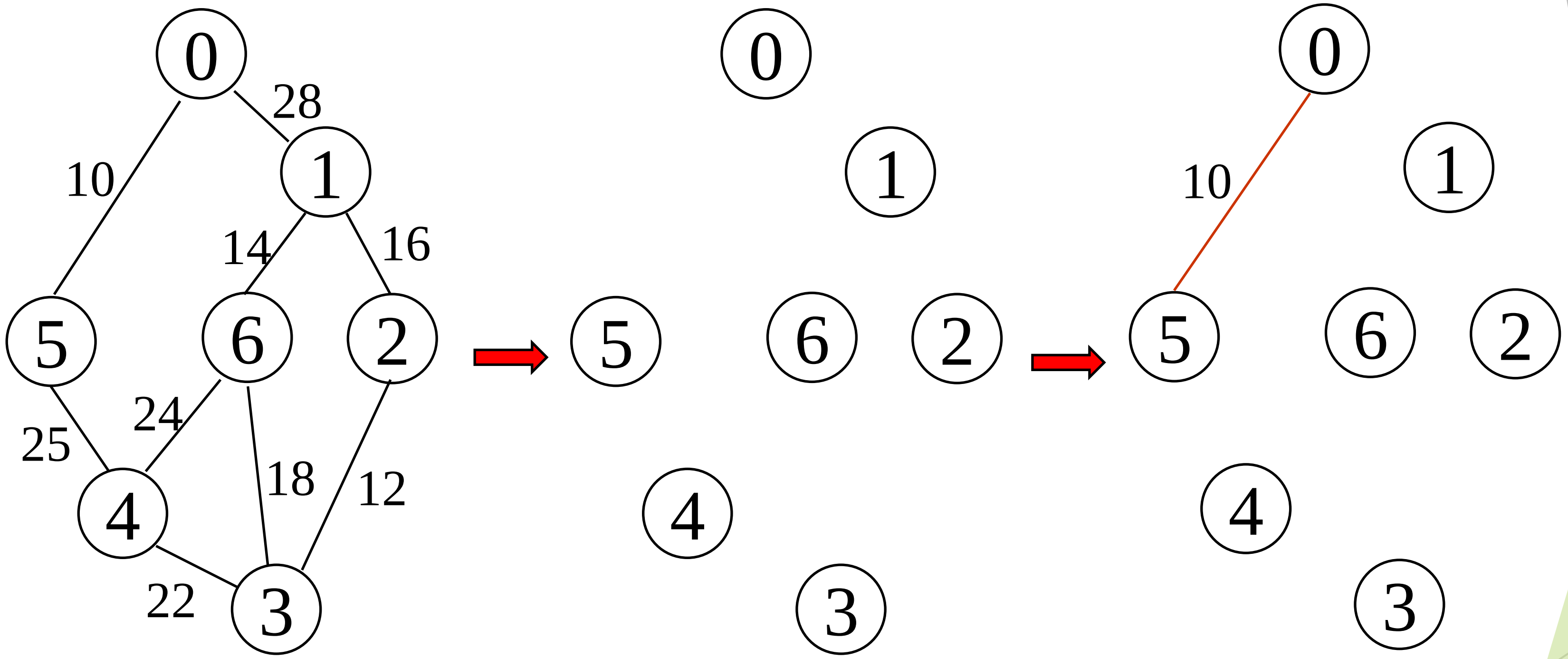
Kruskal’s Algorithm

- Kruskal’s algorithm maintains a forest – a collection of trees.
  - Adding an edge merges two trees into one. When the algorithm terminates, there is only one tree (minimum spanning tree).

```
Step 1: Sort all edges in increasing order of their edge
   weights.
Step 2: Pick the smallest edge.
Step 3: Check if the new edge creates a cycle or loop in a
   spanning tree.
Step 4: If it doesn’t form the cycle, then include that edge
   in MST. Otherwise, discard it.
Step 5: Repeat from step 2 until it includes |V| - 1 edges in
   MST.
```

Kruskal’s Algorithm Simulation

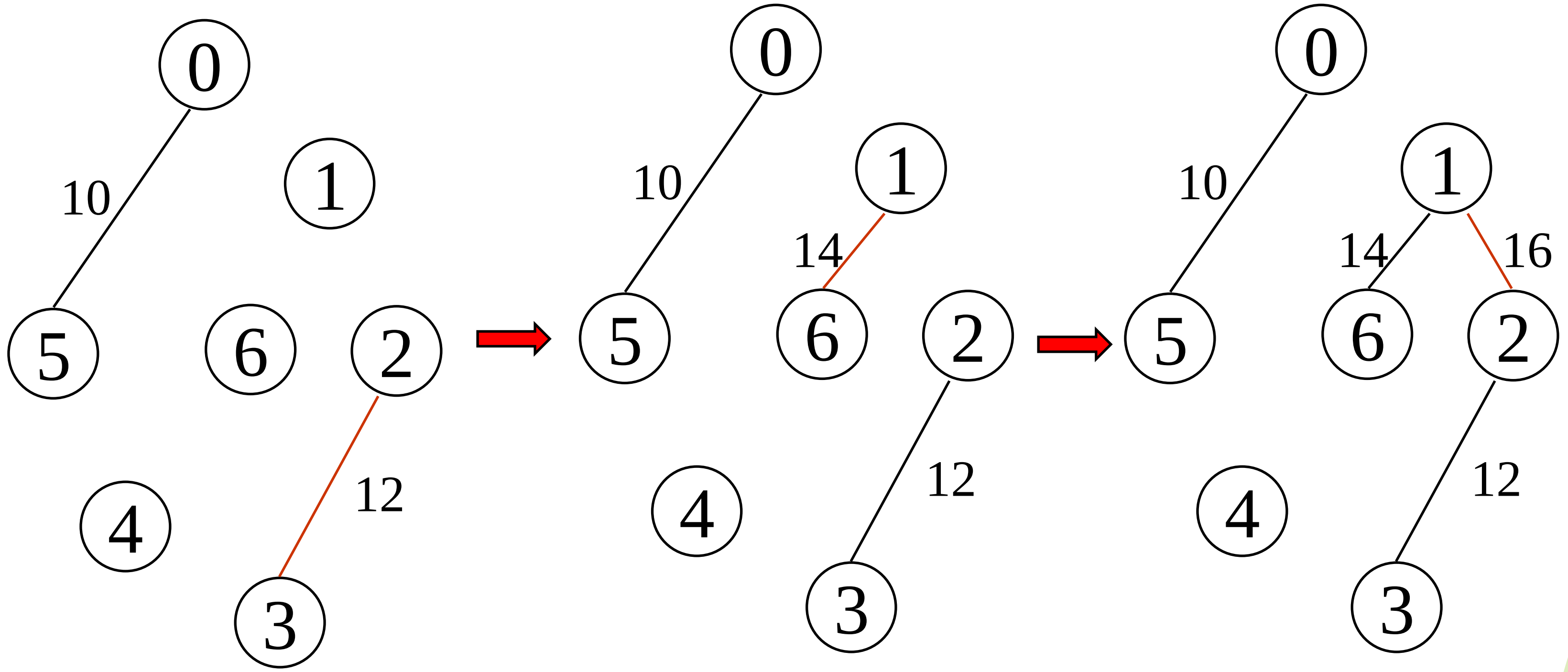
✓	$0 \xrightarrow{10} 5$
	$2 \xrightarrow{12} 3$
	$1 \xrightarrow{14} 6$
	$1 \xrightarrow{16} 2$
	$3 \xrightarrow{18} 6$
	$3 \xrightarrow{22} 4$
	$4 \xrightarrow{24} 6$
	$4 \xrightarrow{25} 5$
	$0 \xrightarrow{28} 1$



Compiled & edited by: Zahid Zainal

Kruskal's Algorithm Simulation (cont.)

- ✓ $0 \overset{10}{\text{———}} 5$
- ✓ $2 \overset{12}{\text{———}} 3$
- ✓ $1 \overset{14}{\text{———}} 6$
- ✓ $1 \overset{16}{\text{———}} 2$
- ✗ $3 \overset{18}{\text{———}} 6$
- $3 \overset{22}{\text{———}} 4$
- $4 \overset{24}{\text{———}} 6$
- $4 \overset{25}{\text{———}} 5$
- $0 \overset{28}{\text{———}} 1$

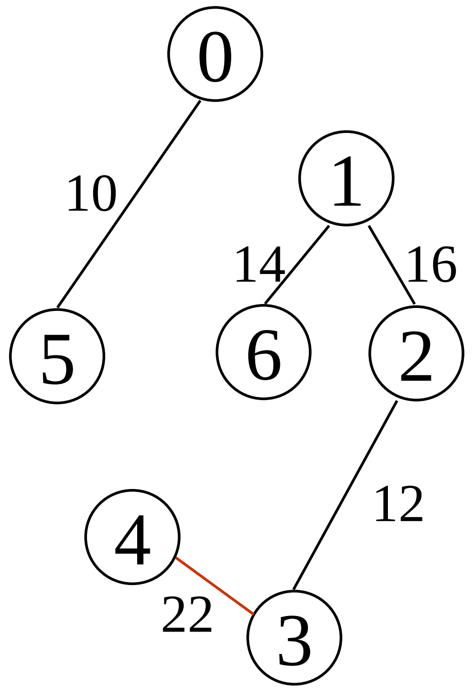


3——6

Creates a cycle, don't add

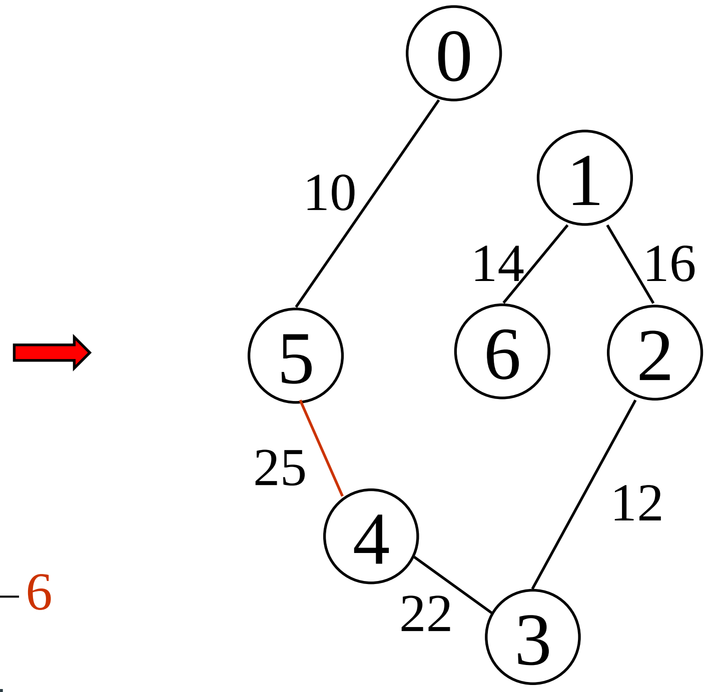
Kruskal’s Algorithm Simulation (cont.)

✓	$0 \overset{10}{—} 5$
✓	$2 \overset{12}{—} 3$
✓	$1 \overset{14}{—} 6$
✓	$1 \overset{16}{—} 2$
✗	$3 \overset{18}{—} 6$
✓	$3 \overset{22}{—} 4$
✗	$4 \overset{24}{—} 6$
✓	$4 \overset{25}{—} 5$
✗	$0 \overset{28}{—} 1$



$4 — 6$

cycle



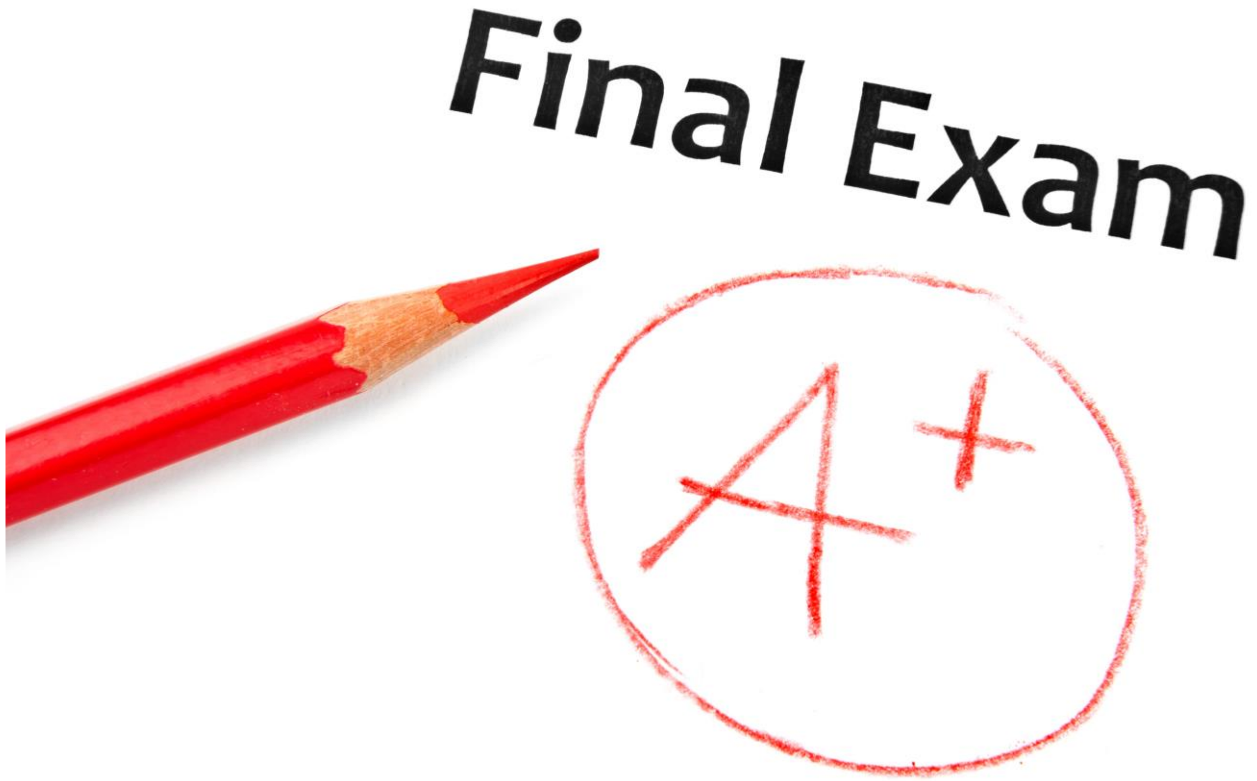
Stop

Tree weight = 10 +25+22+12+16+14

Summary

- Two graph traversal algorithms – Depth first & Breadth first
- Shortest path algorithms finds "optimum" path between two vertices
  - Djikstra's algorithm
- A minimal spanning tree is a spanning tree with the minimum weight
  - Kruskal's algorithm

Next Topic…



References

- Carrano, F. & Savitch, W. 2005. *Data Structures and Abstractions with Java, 2nd ed. Prentice-Hall*.
- Malik D.S, & Nair P.S., Data Structures Using Java, Thomson Course Technology, 2003.
- Rada Mihalcea, CSCE 3110 Data Structures and Algorithm Analysis notes, U of North Texas.