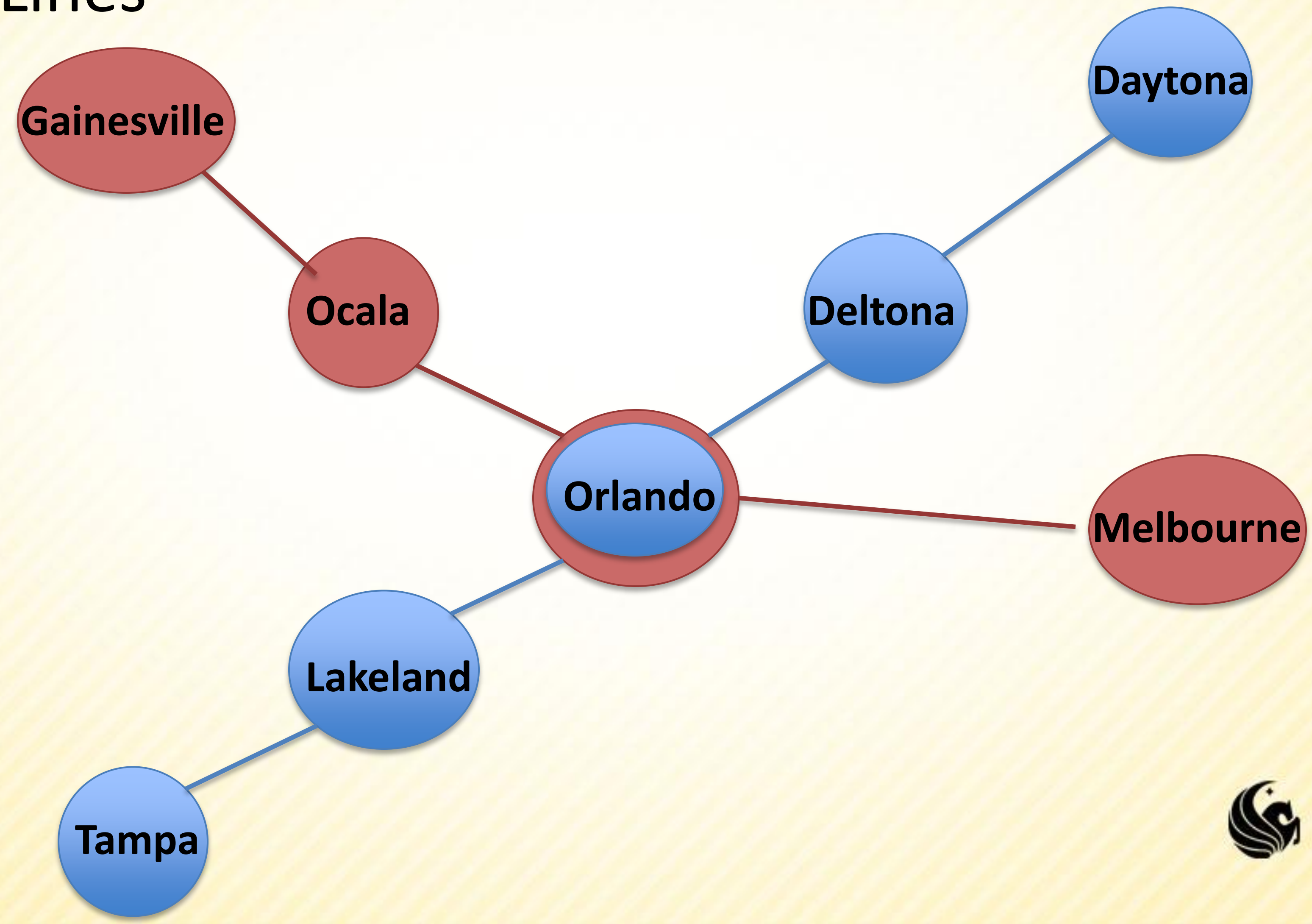
# GRAPHS

COP 3502

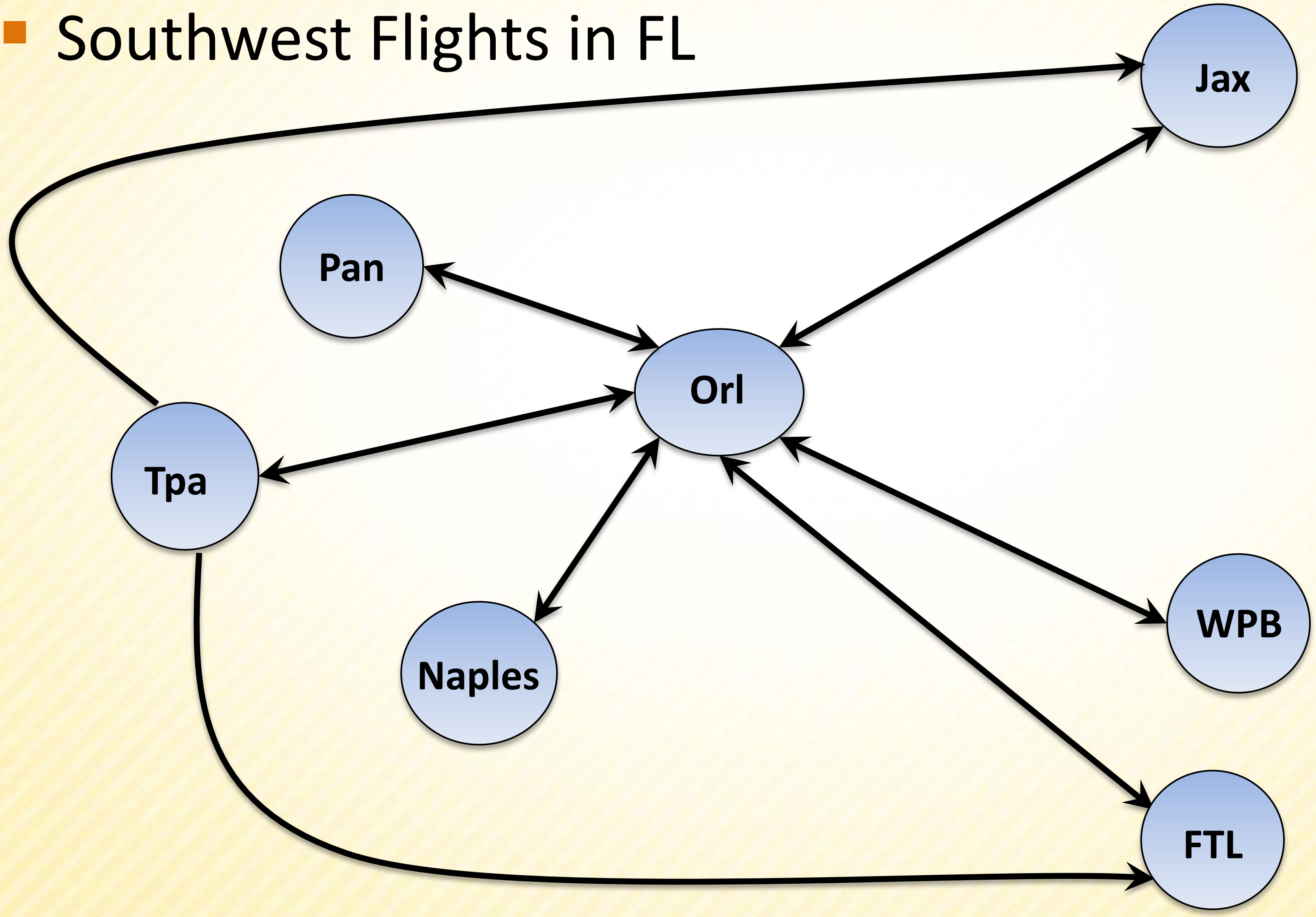
## Graphs

- Train Lines



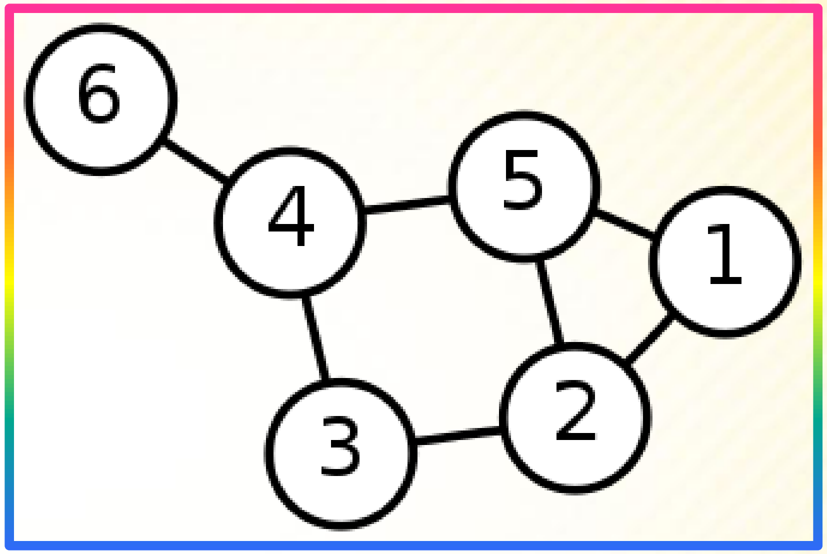
# Graphs

- Southwest Flights in FL



# Graphs

- Definition
  - A Graph G, is a
    - **a set of vertices, V**
    - AND
    - **a set of edges, E**
    - where each edge is associated with a pair of vertices.
- We write: G = (V, E)

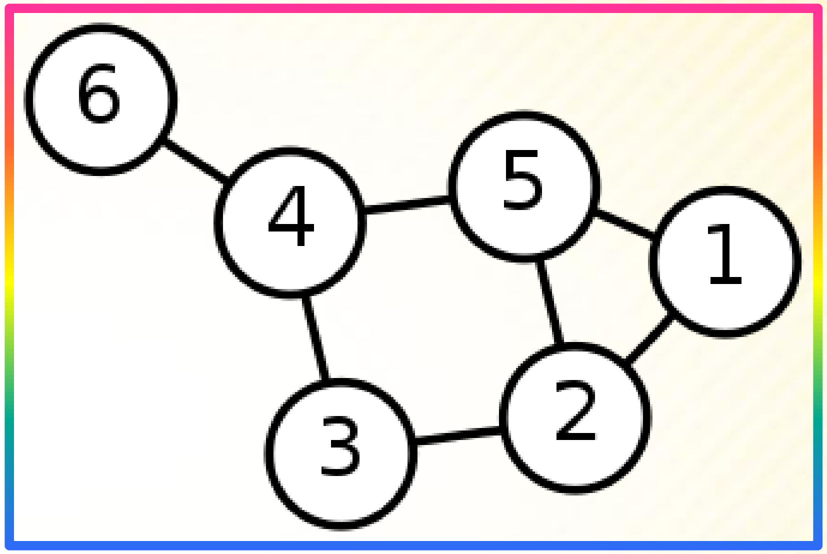


**Vertex set *V*** = {1, 2, 3, 4, 5, 6}

**Edge set *E*** = {{1,2}, {1,5}, {2,3}, {2,5}, {3,4}, {4,5}, {4,6}}

# Graphs

- Undirected vs Directed
  - The following graph is **UNDIRECTED**
  - There is an edge between vertices 4 and 6, but there is no direction.

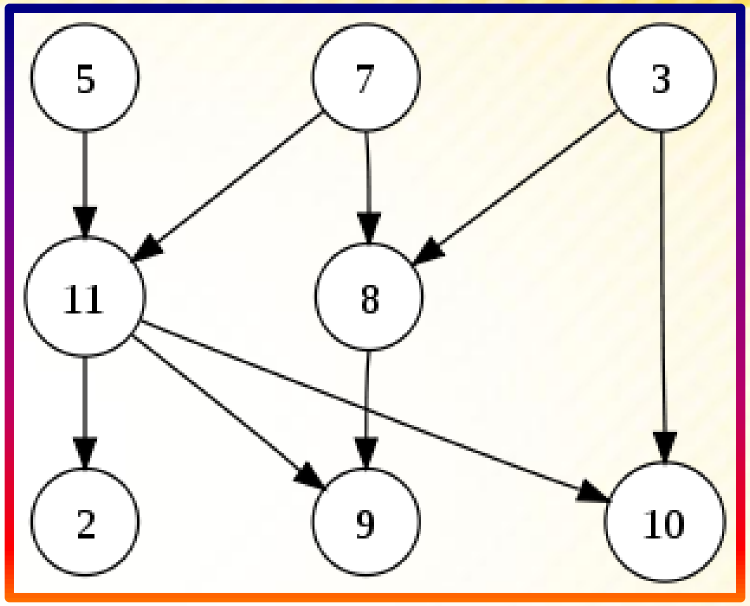


**Vertex set *V*** = {1, 2, 3, 4, 5, 6}

**Edge set *E*** = {{1,2}, {1,5}, {2,3}, {2,5}, {3,4}, {4,5}, {4,6}}

# Graphs

- Directed Graph:
  - Same as before, but where each edge is associated with an *ordered* pair of vertices.
  - Note: This graph does not have the edge (11,5), but it does have (5,11)



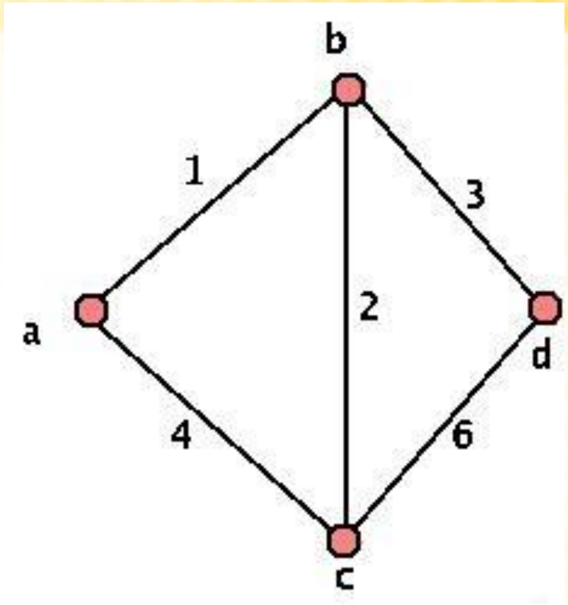
A labeled simple graph:

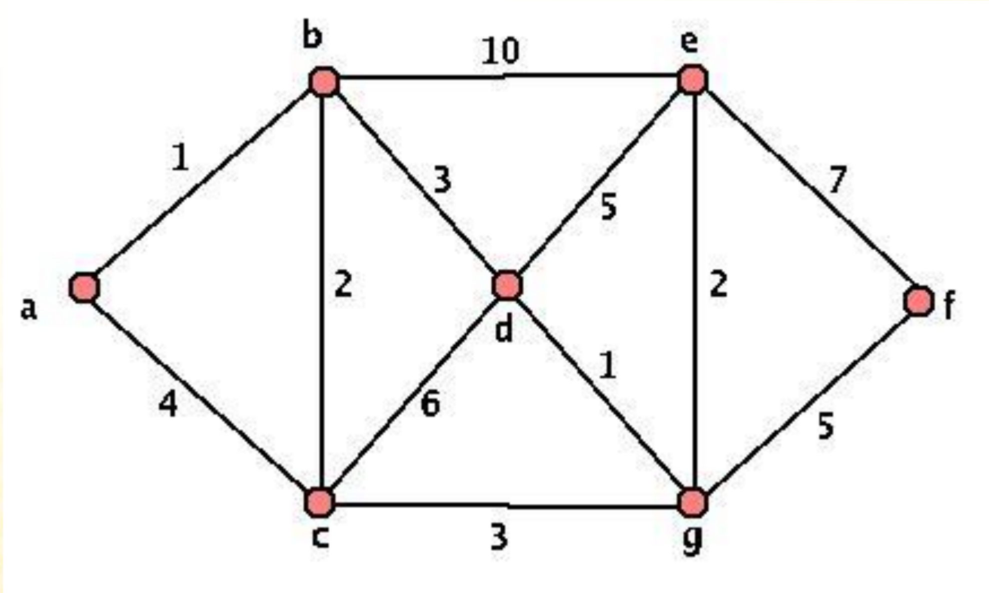
Vertex set *V* = {2,3,5,7,8,9,10,11}

Edge set *E* = {{3,8}, {3,10}, {5,11}, {7,8}, {7,11}, {8,9}, {11,2},{11,9},{11,10}}.

# Graphs

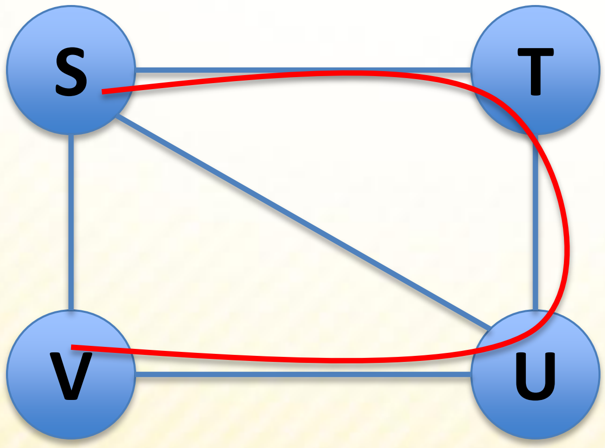
- **Subgraph**
  - Basically, to be a subgraph, all of the edges and vertices must be in the original graph.
  - Just a subset of the vertices and edges





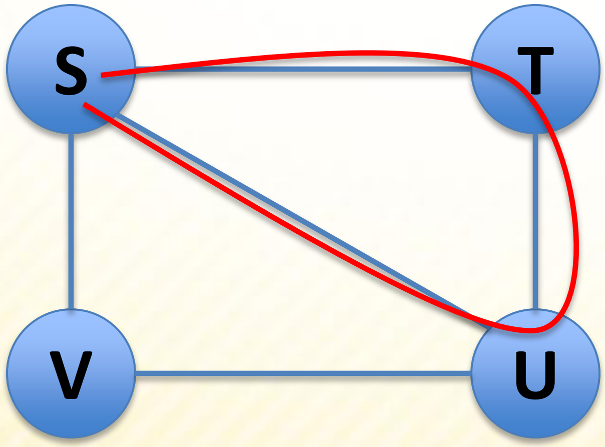
# Graphs

- Simple Path
  - A path such that all vertices are distinct, except that the first and the last could be the same.
    - Simple Path – **S T U V**



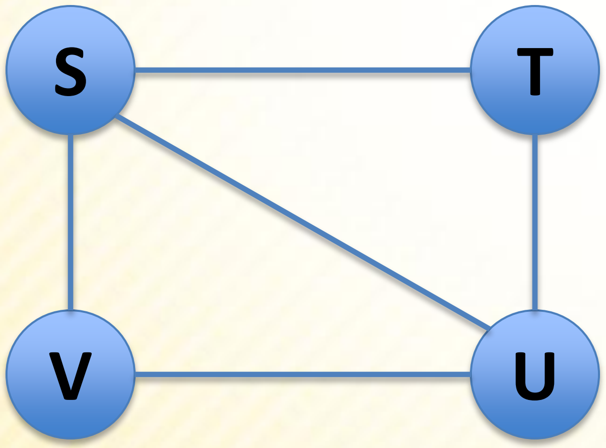
# Graphs

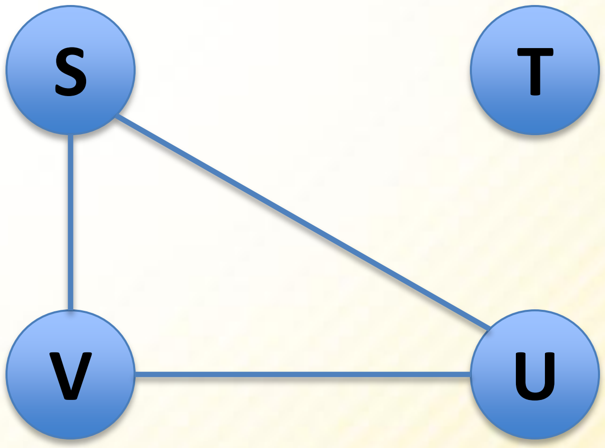
- Cycle
  - A cycle is a path that starts and ends at the same point.  For an undirected graph (such as below) the edges are distinct.
    - **S T U S** is a cycle



## Graphs

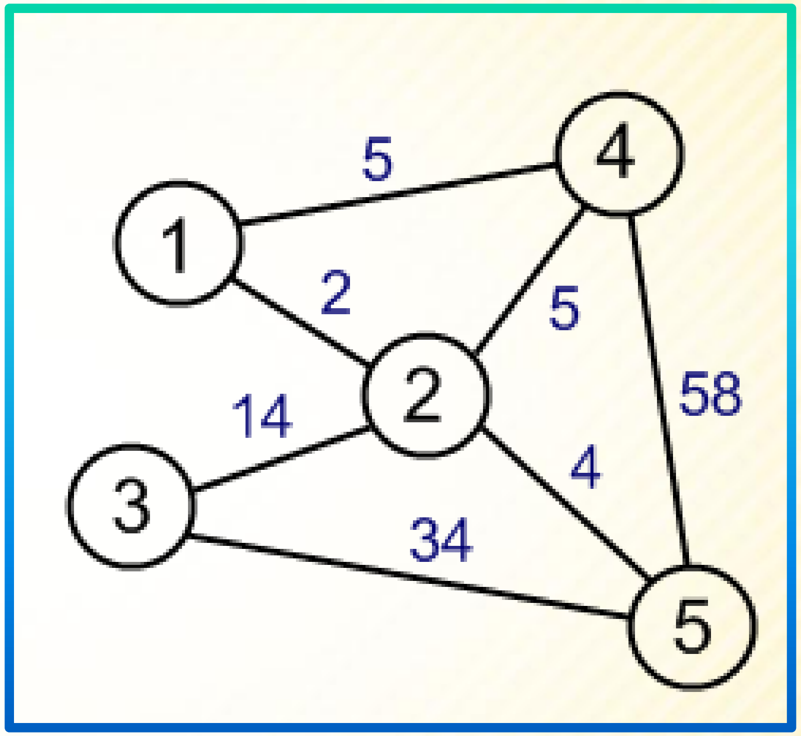
- Connected vs Unconnected
  - A connected graph is one where any pair of vertices in the graph is connected by at least one path.





## Graphs

- Weighted Graph:
  - Same as above, but where each edge *also* has an associated real number with it, known as the edge weight.
    - Can be directed or undirected.



A labeled weighted graph:

Vertex set *V* = {1,2,3,4,5}

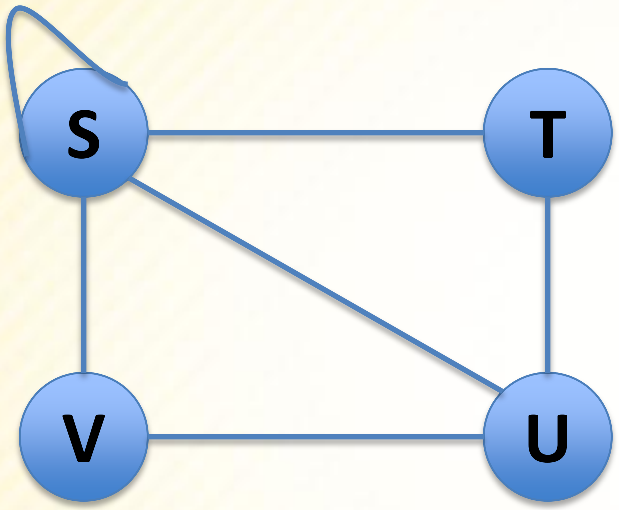
## Graphs

- Graph Storage
  - Adjacency Matrix
  - Adjacency List

# Graph – Adjacency Matrix

- Assume there are N vertices in a graph
  - Then you need an NxN matrix
  - A[0…N-1][0…N-1]
  - If vertex i and vertex j have an edge between them, A[i][j] = 1;
  - Otherwise, A[i][j] = 0
    - ➢Note a vertex can have an edge back to itself (shown as a loop) where A[i][i] = 1, otherwise A[i][i] = 0

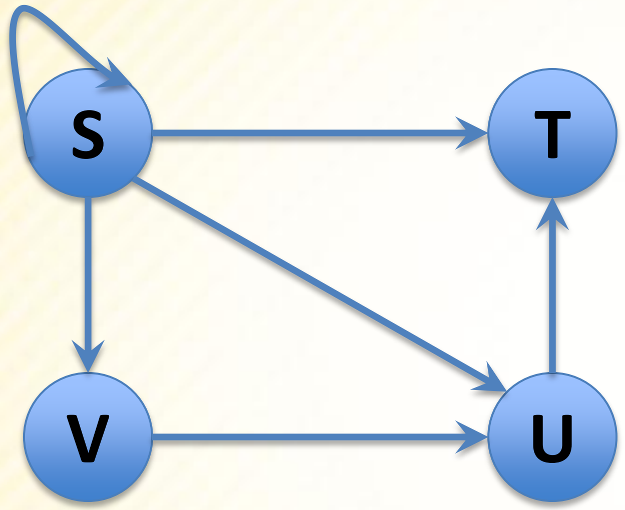
# Adjacency Matrix - Undirected



| | S | T | U | V |
|---|---|---|---|---|
| S | | | | |
| T | | | | |
| U | | | | |
| V | | | | |

- Since this is an undirected graph, the adjacency matrix is symmetric
- A directed graph may not have a symmetric adjacency matrix.

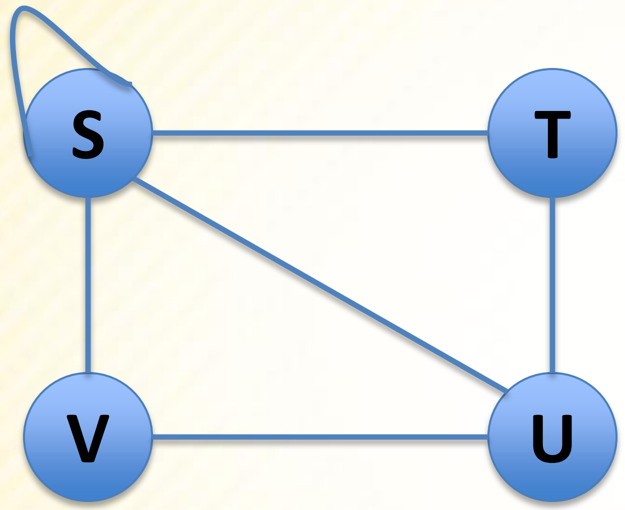
# Adjacency Matrix - Directed



| | S | T | U | V |
|---|---|---|---|---|
| S | | | | |
| T | | | | |
| U | | | | |
| V | | | | |

- A directed graph may not have a symmetric adjacency matrix.
  - Note in this matrix the start is the row, and the end is the column.

# Adjacency Matrix - Weighted

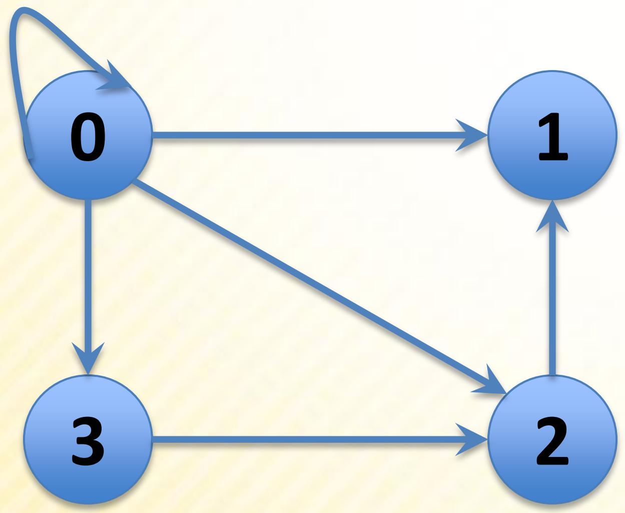


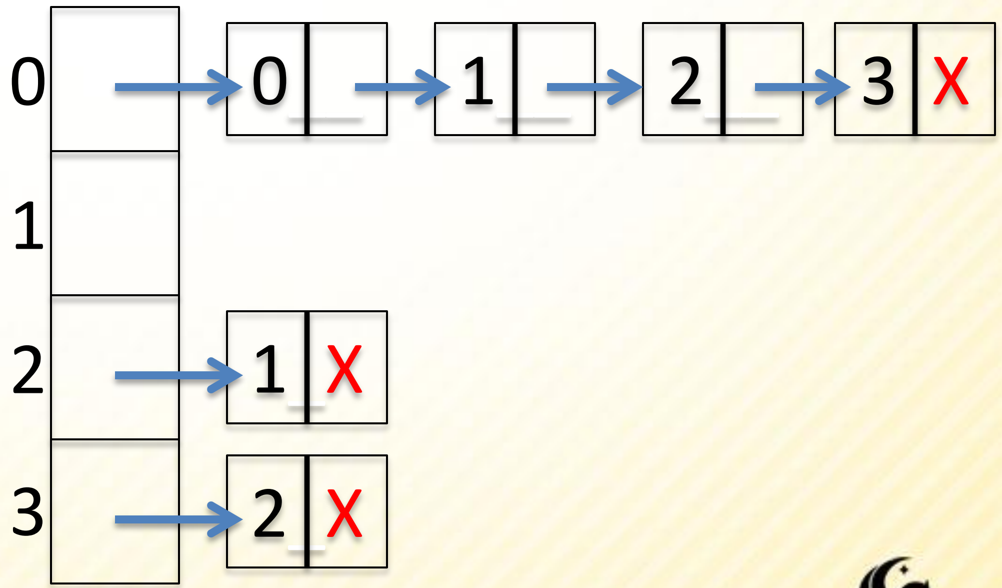
| | S | T | U | V |
|---|---|---|---|---|
| S | | | | |
| T | | | | |
| U | | | | |
| V | | | | |

- Since this is an undirected graph, the adjacency matrix is symmetric
- A directed graph may not have a symmetric adjacency matrix.

# Graph – Adjacency List

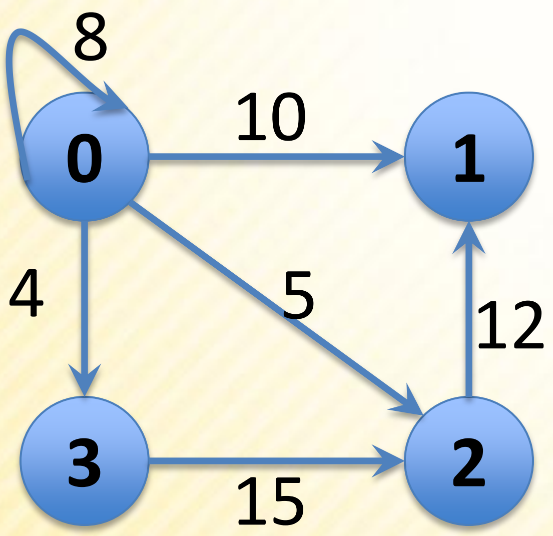
- An array of linked lists
  - Where Array[i] holds the list of vertices that are adjacent to vertex i

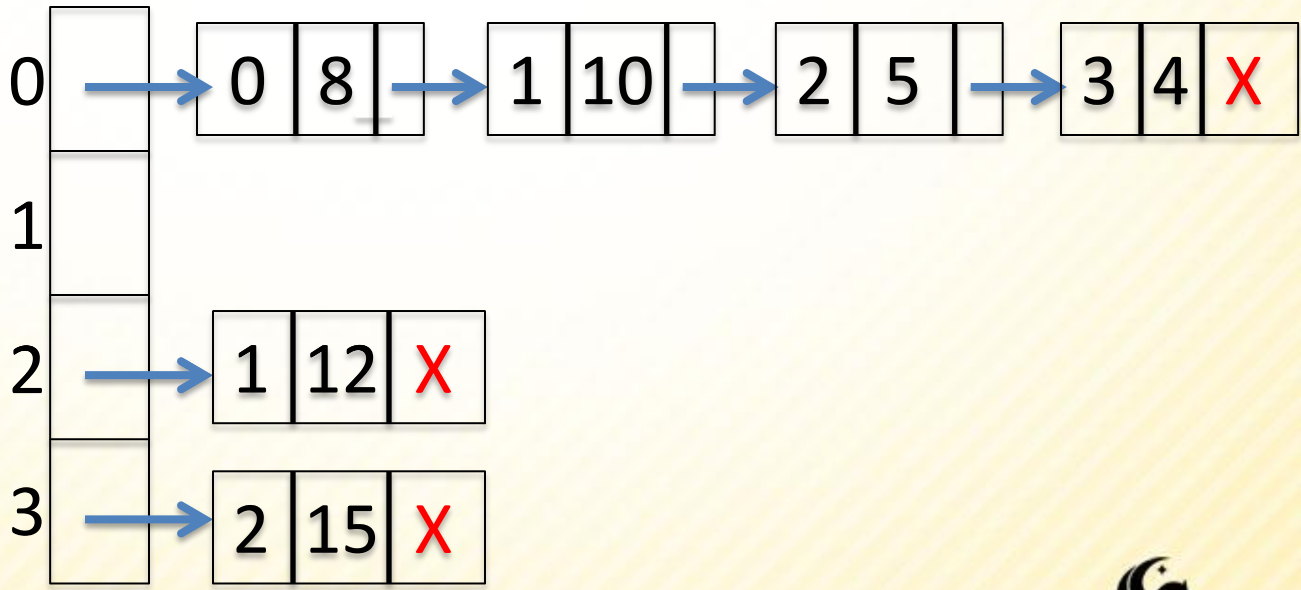




# Graph – Adjacency List

- Weighted Graph
  - Adjacency List - array of linked lists
    - Where Array[i] holds the list of vertices that are adjacent to vertex i



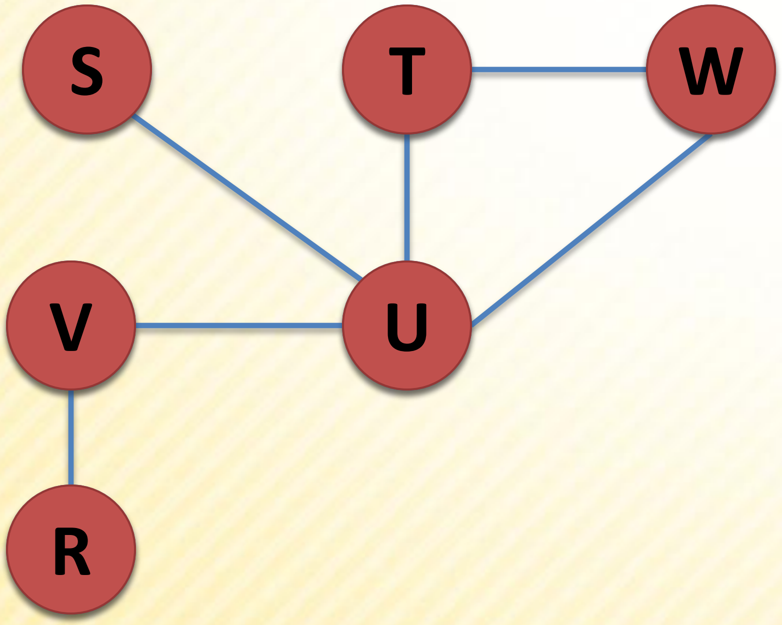


## Graph Traversals

- Depth First Search
- Breadth First Search

# Graph Traversals

- **<u>Depth First Search</u>**
- The general "rule" used in searching a graph using a depth first search is to search down a path from a particular source vertex as far as you can go.
  - When you can go to farther, "backtrack" to the last vertex from which a different path could have been taken.
  - Continue in this fashion, attempting to go as deep as possible down each path until each node has been visited.



**What is the DFS Traversal of the following Graph starting from S? When you have a choice between which vertex to search next choose alphabetically.**

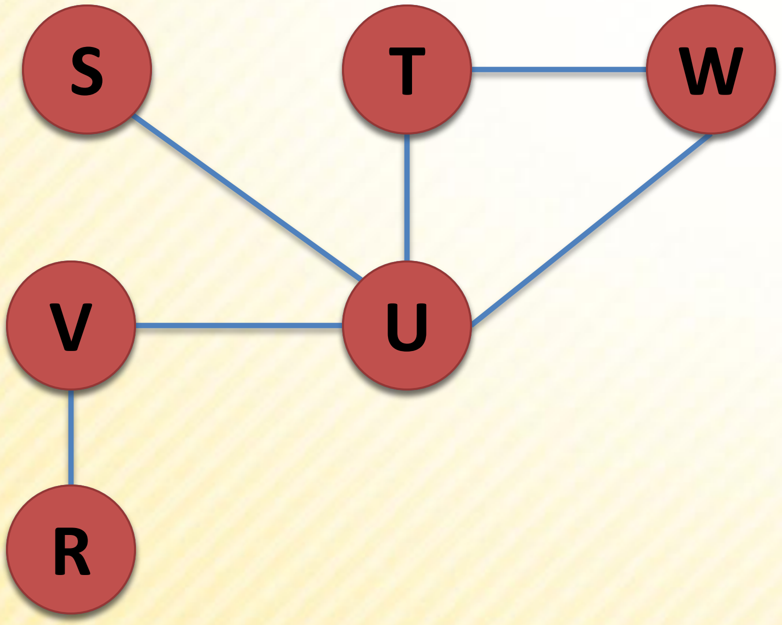
S U T W V R

## Depth First Search

- The general "rule" used in searching a graph using a depth first search is to search down a path from a particular source vertex as far as you can go.
  - When you can go to farther, "backtrack" to the last vertex from which a different path could have been taken.
  - Continue in this fashion, attempting to go as deep as possible down each path until each node has been visited.
- The most difficult part of this algorithm is keeping track of what nodes have already been visited, so that the algorithm does not run ad infinitum. We can do this by labeling each visited node.

# Graph Traversals

- <u>Breadth First Search</u>
- The idea in a breadth first search is opposite to a depth first search. Instead of searching down a single path until you can go no longer, you search all paths at an uniform depth from the source before moving onto deeper paths.
  - Once again, we'll need to mark both edges and vertices based on what has been visited.



What is the BFS Traversal of the following Graph starting from S? When you have a choice between which vertex to search next choose alphabetically.

S U T V W R

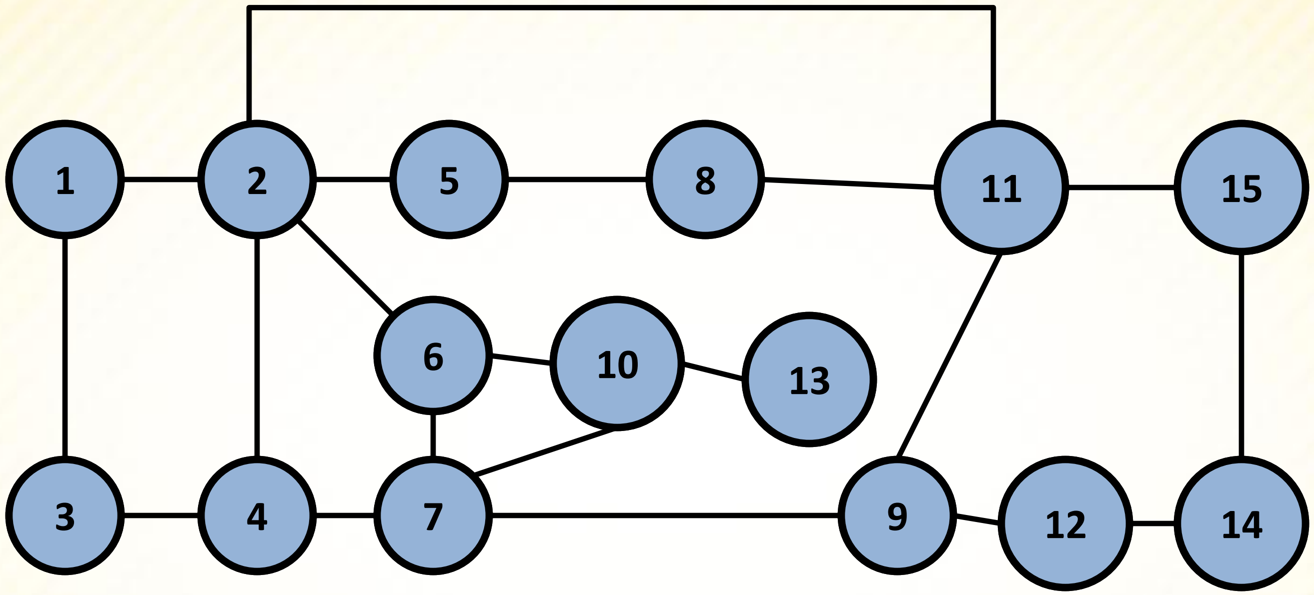
## Breadth First Search

- Instead of searching down a single path until you can go no longer, you search all paths at an uniform depth from the source before moving onto deeper paths. Once again, we'll need to mark both edges and vertices based on what has been visited.

- In essence, we only want to explore one "unit" away from a searched node before we move to a different node to search from. All in all, we will be adding nodes to the back of a queue to be ones to searched from in the future.

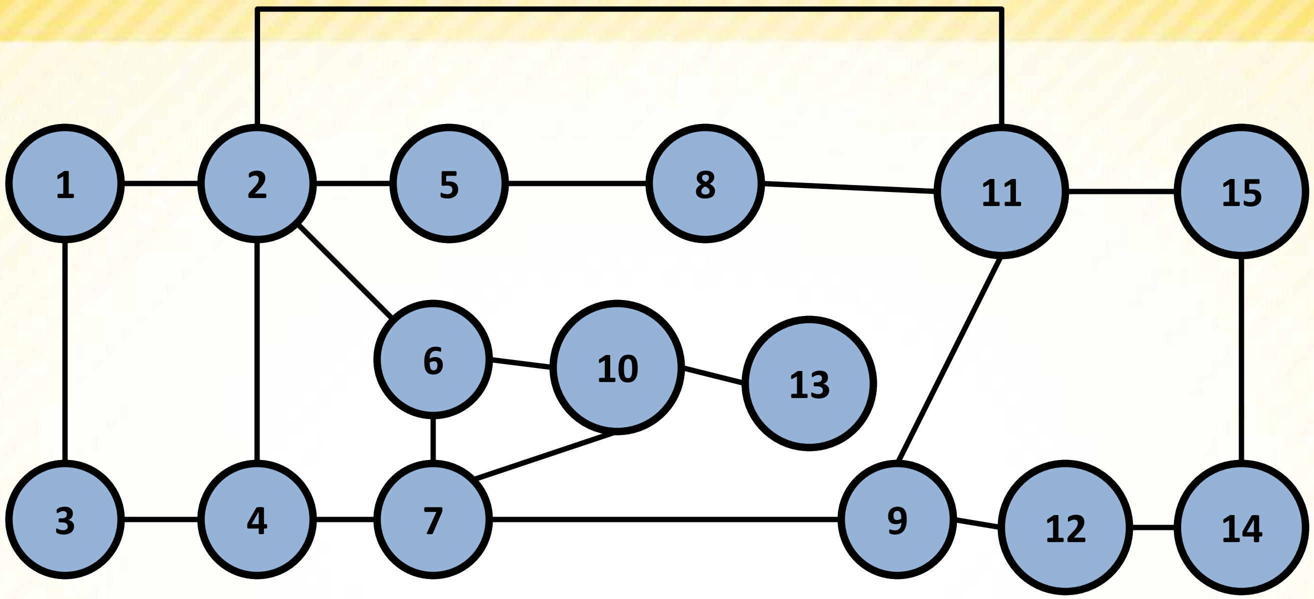
# Depth First Search

**Output List:** 1, 2, 4, 3, 7, 6, 10, 13, 9, 11, 8, 5, 15, 14, 12



- Start with 1
- 2,4,3
  - Stop at 3
- Backtrack to 4, can we search?
- 7,6
  - Stop at 2
- Backtrack to 6, can we search?
- 10,13
  - Stop at 13
- Backtrack to 10,6,7
- 9, 11, 8, 5
  - Stop at 5
- Backtrack to 8,11
- 15,14,12
  - STOP – All nodes are marked!!

# Breadth First Search



- L0: 1
- L1: 2, 3
- L2: 4,5,6,11
- L3: 7,8,10,9,15
- L4: 13,12,14

**Final output:** **1, 2, 3, 4, 5, 6, 11, 7, 8, 10, 9, 15, 13, 12, 14**

## Breadth First Search

- The basic idea
  - we have successive rounds and continue with our rounds until no new vertices are visited on a round.
  - For each round, we look at each vertex connected to the vertex we came from.
  - And from this vertex we look at all possible connected vertices.
- This leaves no vertex unvisited
  - because we continue to look for vertices until no new ones of a particular length are found.
  - If there are no paths of length 10 to a new vertex, surely there can be no paths of length 11 to a new vertex.
- The algorithm also terminates since no path can be longer than the number of vertices in the graph.

## Directed Graphs

- *Traversals*
  - Both of the traversals DFS and BFS are essentially the same on a directed graph.
  - When you run the algorithms, you must simply pay attention to the direction of the edges.
  - Furthermore, you must keep in mind that you will only visit edges that are reachable from the source vertex.

# Graph Traversal – Application

- *Mark and Sweep Algorithm for Garbage Collection*
  - A mark bit is associated with each object created in a Java program.
    - It indicates if the object is live or not.
  - When the JVM notices that the memory heap is low, it suspends all threads, and clears all mark bits.
    - To garbage collect, we go through each live stack of current threads and mark all these objects as live.
    - Then we use a DFS to mark all objects reachable from these initial live objects. (In particular each object is viewed as a vertex and each reference as a directed edge.)
    - This completes marking all live objects.
      - Then we scan through the memory heap freeing all space that has NOT been marked.

## Example Exam Questions

- What is the DFS and BFS of the following Graph?
- What is the adjacency matrix and adjacency list of the following Graph?