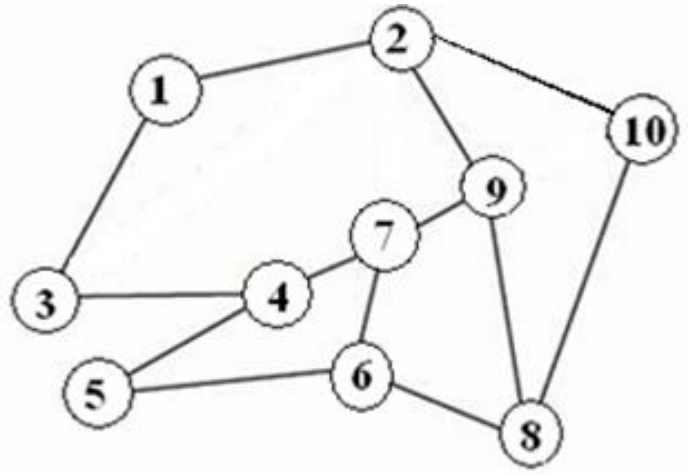
PROBLEM SESSION 3 (GRAPHS)

Graph traversal -1:

Traverse the following graph, starting from Node#1, by the use of depth_first_search and breadth _first search approaches. Make sure that when there are multiple nodes to be considered, the smallest one will be selected. Draw, seperately, the result of each approach.



- Draw it (BFS, DFS)

Graph traversal -2:

What are the main data structures employed when implementing DFS and BFS on a graph? Why?

Answer:

- BFS -> Queue, DFS -> Stack
- In BFS, we trace the graph layer by layer by considering all children of a node before starting to trace nodes further away. This can be done by a FIFO queue, ordering the nodes by layer number.
- In DFS, algorithm considers the immediate unexplored children before considering the other children of a node while moving from parent node to child node. It goes deeper through the branches of the graph before tracing other branches. «Last in first out» mechanism in the stack fits for this algorithm.

Testing Bipartiteness:

What is a bipartite graph? Is the graph (for the graph above) bipartite? Why?

Answer:

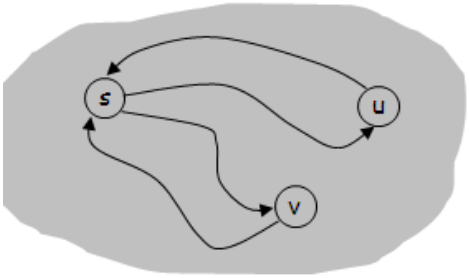
- A graph G (V, E) is *bipartite* if its node set V can be partitioned into sets X and Y in such a way that every edge has one end in X and the other end in Y.
- This is a bipartite graph. If we re-draw the graph, it can be observed.
- (Re-draw it)

Connectivity in Directed Graphs -1:

What is the strong connectivity for a graph? For a given graph G, how could we test if it is strongly connected or not? Explain step by step. You may refer to some algorithms discussed in the class without writing them.

Answer:

- A directed graph is strongly connected if, for every two nodes u and v, there is a path from u to v and a path from v to u. Ex:



- Yes we can test if it is strongly connected or not. The algorithm:
    - pick any node s
    - run BFS in G starting from s
    - define a new directed graph, G^{rev}, that is obtained from G simply by reversing the direction of every edge.
    - run BFS starting from s in G^{rev}
    - check if both algorithm have reached every node or not
    - if yes, G is strongly connected

Connectivity in Directed Graphs – 2:

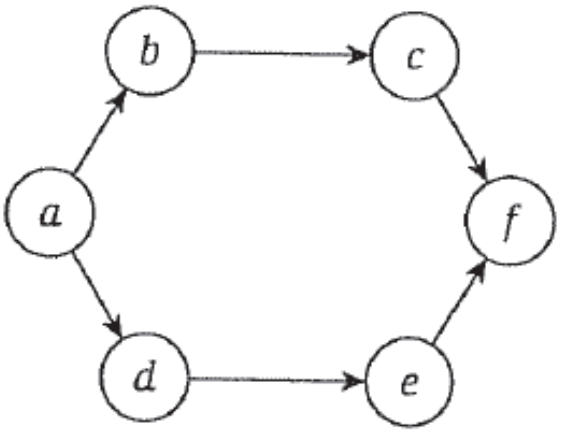
Write an algorithm that finds **all** two-edge long paths with distinct nodes in a directed graph.

Answer:

```
Function ( M[1…n][1…n] )  //we hold directed graph in matrix M

    List L; // L  holds two-edge long paths.

    for i  = 1 to n

        for j = 1 to n

            for k = 1 to n

                if  M[i][j] + M[j][k] == 2 && i != k && (i, j, k) pair is not in L

                    Add (i, j, k) pair to L
```

Dags and Topological Order:

Consider the directed acyclic graph below. How many topological orderings does it have?



Answer:

- A topological order of a directed graph G = (V, E) is an ordering of its nodes as v_1, v_2, …, v_n so that for every edge (v_i, v_j) we have i < j.
- It is clear that node a comes first, node f comes last.
- b must precede c, and d must precede e.
- But b comes either before or after d, and so on.
- Consider the probabilities of these four nodes:
    - b, c, d, e / b, d, e, c / b, d, c, e
    - d, e, b, c / d, b, c, e / d, b, e, c
- Thus the number of topological orderings is 6.