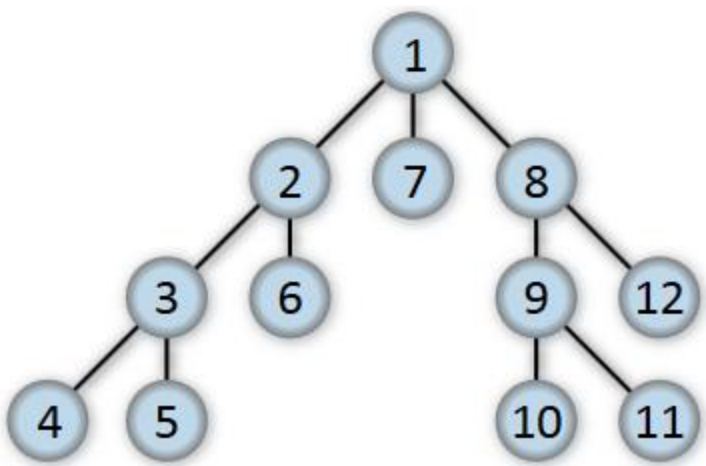
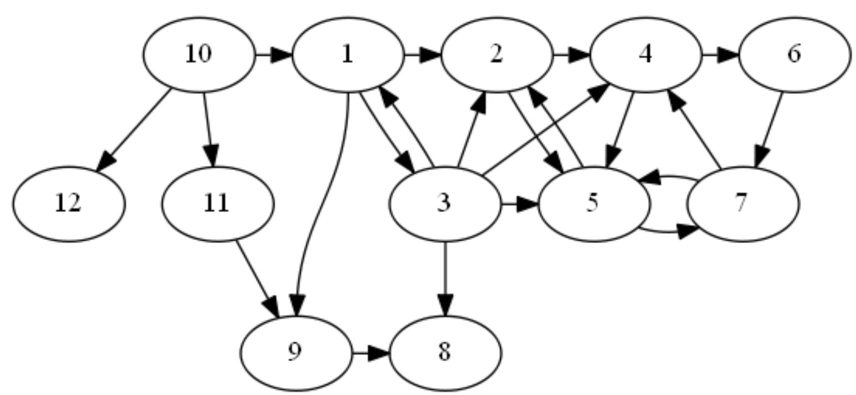
Depth-first algorithm

- Part of your assignment 3
- We’ve learned depth-first search on trees
  - pre-order traversal
- Extend it to graph
  - Instead of “child” in tree, use "neighbor“ (adjacent)
  - For each vertex, need to decide in what order to visit the adjacent vertex – using vertex number to break ties





Depth-First Search code

- Goes as far as possible from a vertex before backing up. Do this until visit all vertices.
- Recursive algorithm

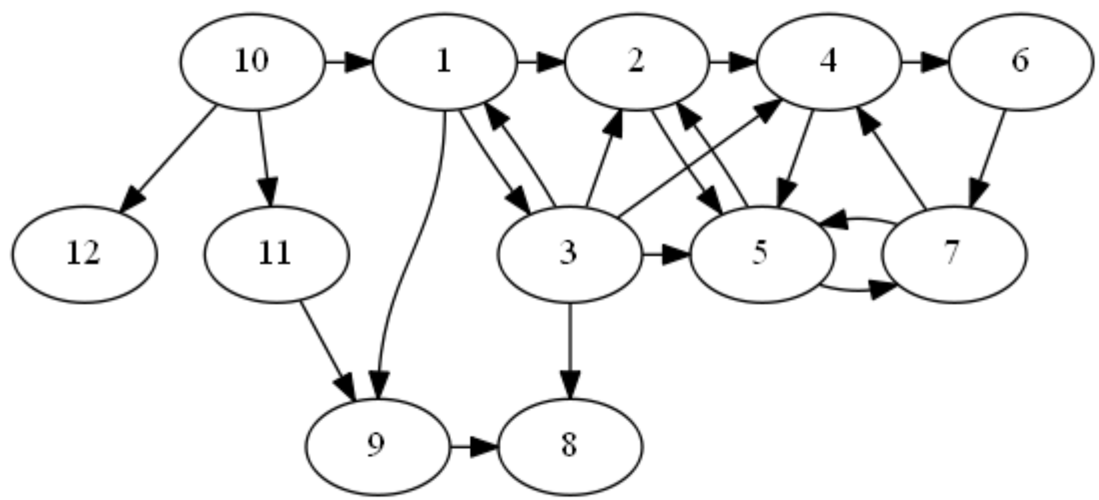
```
// traverses a graph beginning at v by using a DFS
dfs ( v: vertex)
      Mark v as visited
      for (each unvisited vertex u adjacent to v)
            dfs(u)
```

Depth-First Search

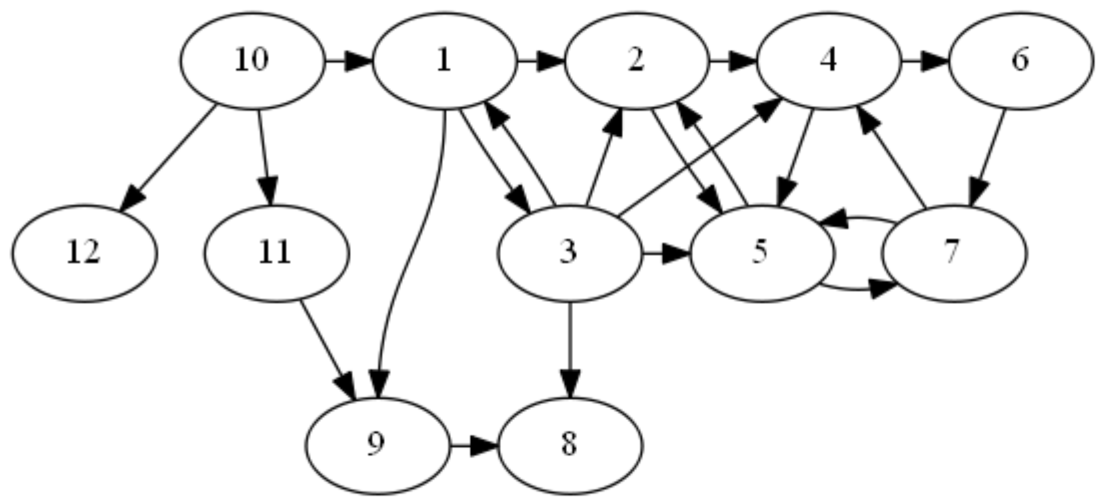
- Iterative algorithm, using a stack

```
// traverses a graph beginning at v by using a DFS
dfs ( v: vertex)
       s = a new empty stack
       // push v onto the stack and mark it
       s.push(v)
       Mark v as visited
       while (!s.isEmpty()){
         if (!unvisited vertices adjacent to top of stack)
              s.pop()
         else{
              select an unvisited u adjacent to top of stack)
              s.push(u)
              Mark u as visited
              }
       }
```

Example



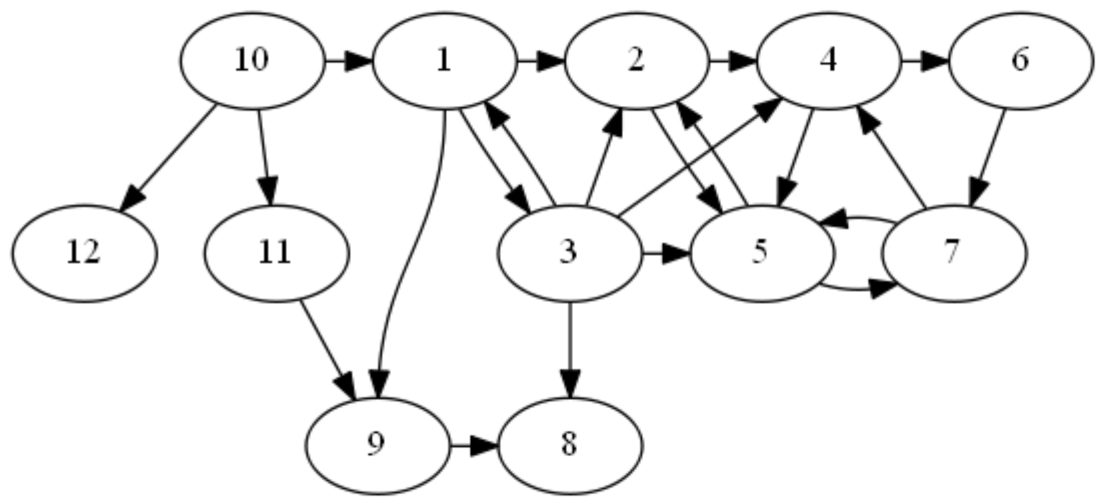
Example



Adjacency list

1→2→3→9
2→4→5
3→1→2→4→5→8
4→5→6
5→2→7
6→7
7→4→5
8
9→8
10→1→11→12
11→9
12

Example



Adjacency list

1→2→3→9
2→4→5
3→1→2→4→5→8
4→5→6
5→2→7
6→7
7→4→5
8
9→8
10→1→11→12
11→9
12

s(stack): 1 2 4 5 7
DFS : 1 2 4 5 7

s(stack): 1 2 4 5 ~~7~~
DFS : 1 2 4 5 7

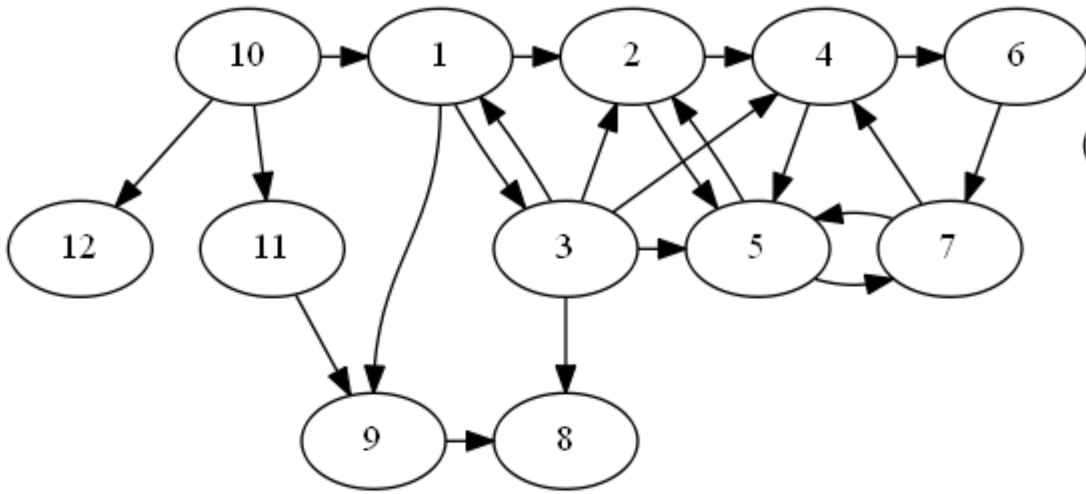
s(stack): 1 2 4 ~~5~~
DFS : 1 2 4 5 7

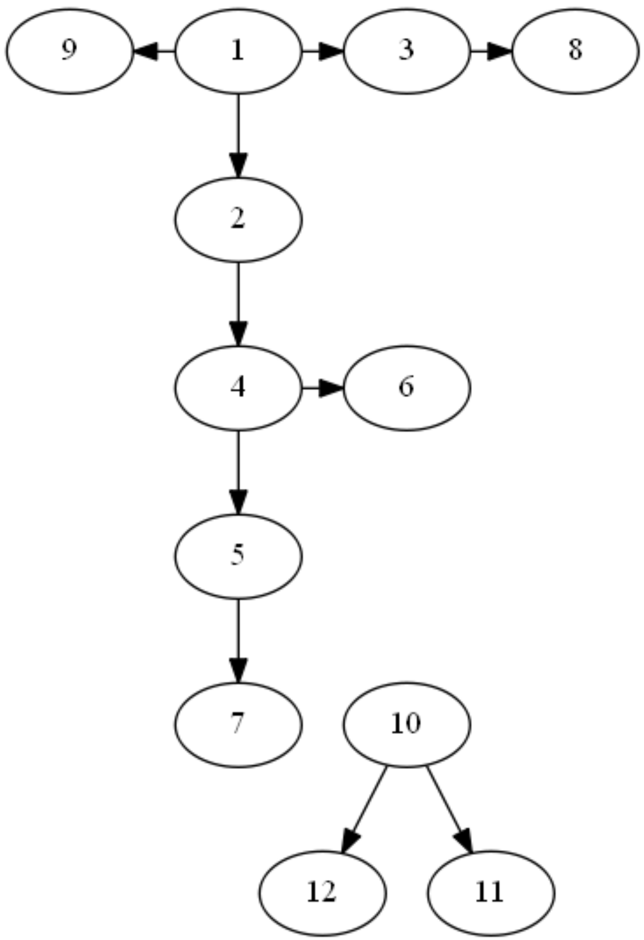
s(stack): 1 2 4 6
DFS : 1 2 4 5 7 6

s(stack): ~~1~~
DFS : 1 2 4 5 7 6 3 8 9

s(stack): ~~10~~ ~~12~~
DFS : 1 2 4 5 7 6 3 8 9 10 11 12

Example



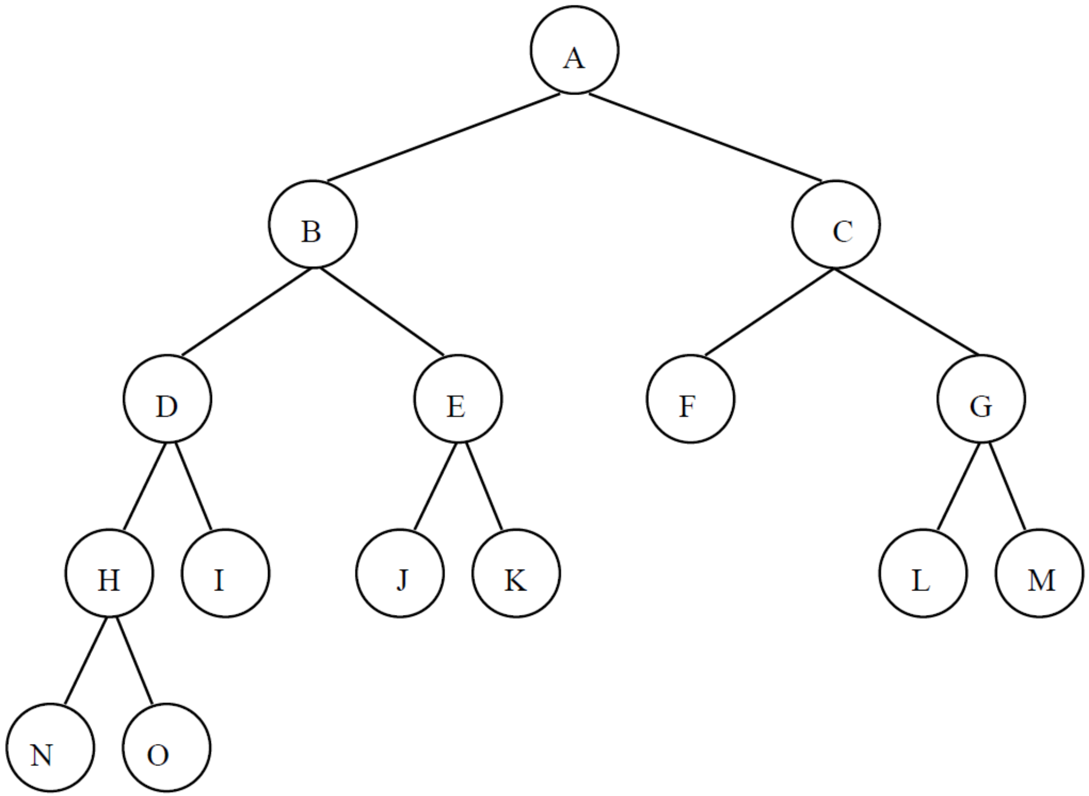


DFS : 1 2 4 5 7 6 3 8 9 10 11 12

- Draw DFS spanning tree

1, 2, 4, 5, 7, 6, 3, 8, 9, 10, 11, 12

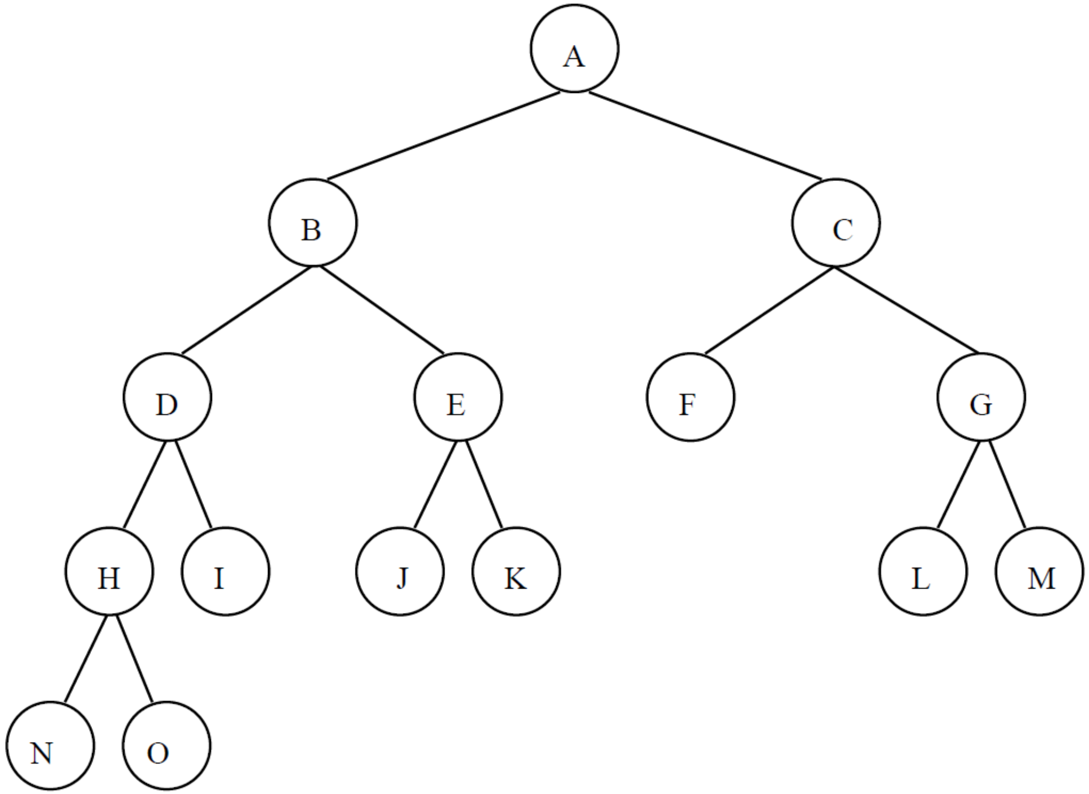
DFS in a tree



DFS:

Preorder:

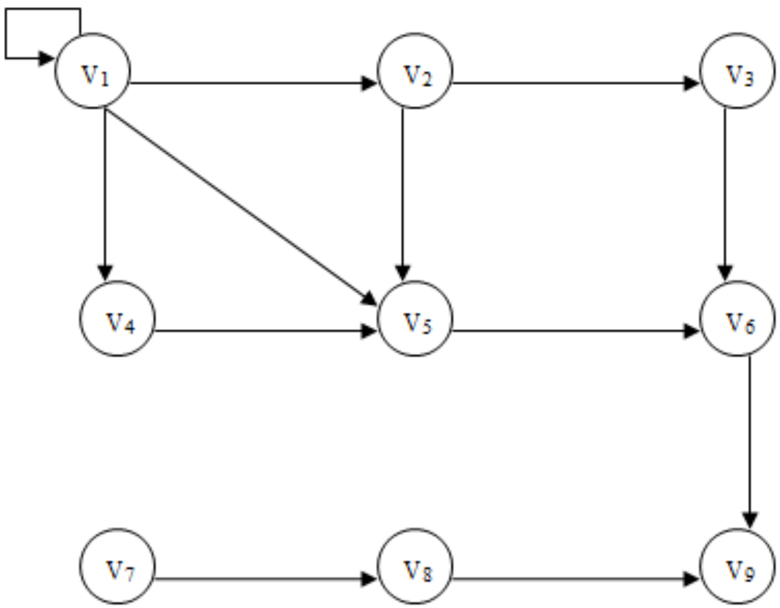
DFS in a tree



DFS: A B D H N O I E J K C F G L M

Preorder: A B D H N O I E J K C F G L M

Exercise– depth first



Is it connected?

Time for DFS

```
// traverses a graph beginning at v by using a DFS
dfs ( v: vertex)
        s = a new empty stack
        // push v onto the stack and mark it
        s.push(v)
        Mark v as visited
        while (!s.isEmpty())      {
          if (!unvisited vertices adjacent to top of stack)
                s.pop()
          else{
                select an unvisited u adjacent to top of stack)
                s.push(u)
                Mark u as visited
                }
        }
```

Each vertex is visited once, and each neighbor is visited at most once in directed graph, twice in undirected graph.

Therefore, O(V+E)

Analysis

- DFS: O(V+E)

Question) in a binary tree, what is the complexity of preorder? O(V+V-1) = O(V), since E = V-1 in a binary tree

- Finding articulation points:
  1. DFS ordering and find seq# : O(V+E)
  2. Compute low#: O(V)
     1) Get postorder in the DFS : O(V)
     2) For each vertex v,

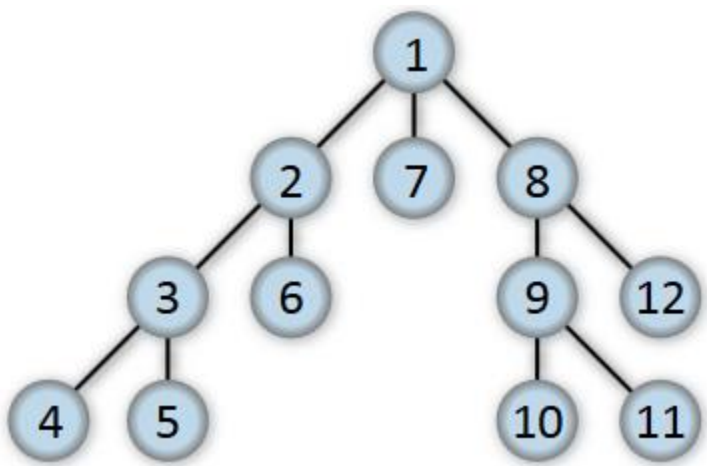
$$low(v) = \min \left\{ \begin{array}{l} seq(v), \\ seq(w) \text{ where } v \rightarrow w \text{ is backedge} \\ low(w) \text{ where } v \rightarrow w \text{ is tree edge (w is child of w)} \end{array} \right\}$$

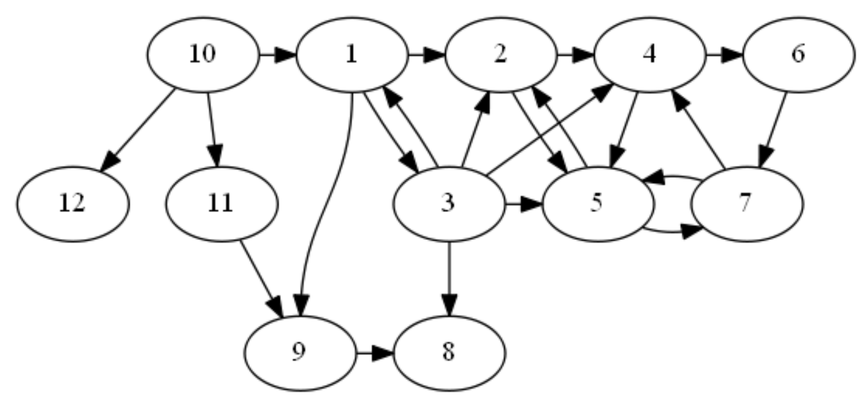
Applications of Depth first search

- Minimum spanning trees in unweighted graphs
- Cycle detection (using back edges)
- Path finding
- Topological Sorting
- Puzzles with only one solution (e.g., mazes)

Breadth first algorithm

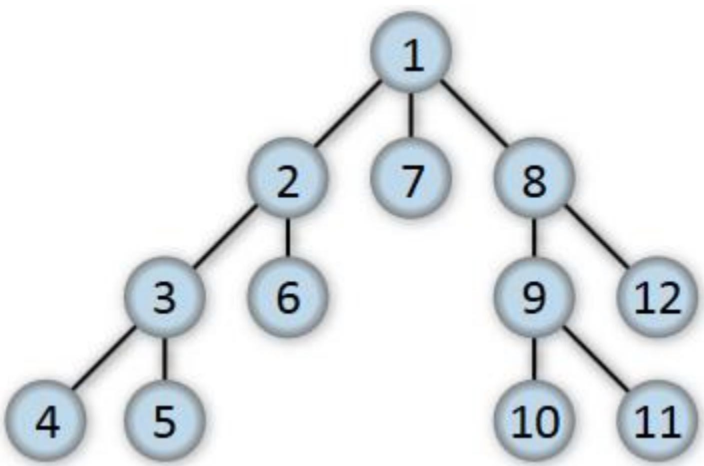
- Breadth-first search on trees
  - Level-order traversal
- Extend it to graph
  - Instead of “child” in tree, use "neighbor“ (adjacent)
  - Keep track of paths
  - Find BFS ordering in those graphs

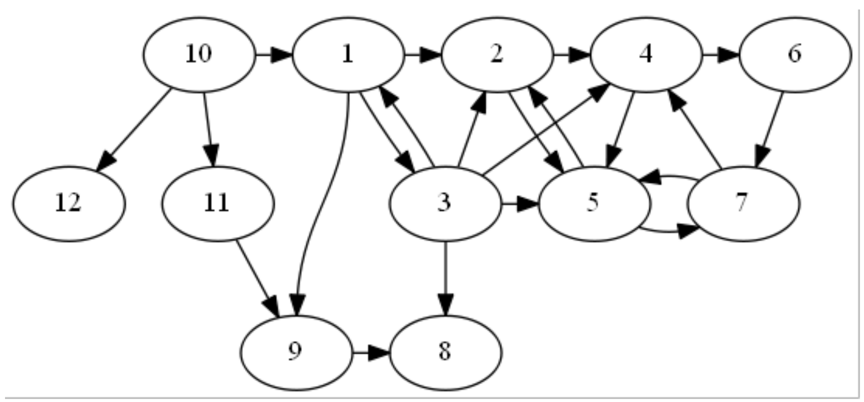




Breadth first algorithm

- Breadth-first search on trees
  - Level-order traversal
- Extend it to graph
  - Instead of “child” in tree, use "neighbor“ (adjacent)
  - Keep track of paths
  - Find BFS ordering in those graphs

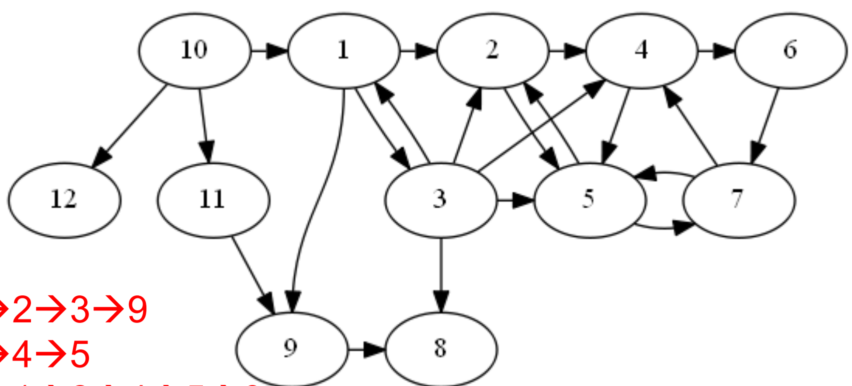




1,2,7,8,3,6,9,12,4,5,10,11

Breadth first

- Level-order traversal: process vertices closest to the start first, and the most distant last
- Keep track of paths
- Use queue

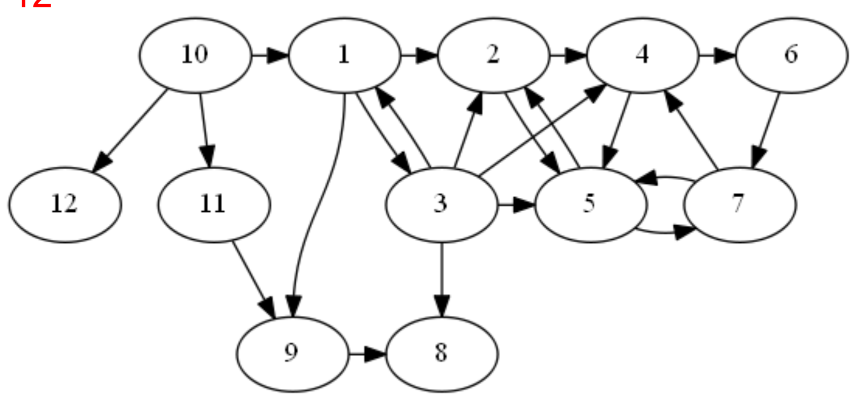


1→2→3→9
2→4→5
3→1→2→4→5→8
4→5→6
5→2→7
6→7
7→4→5
8
9→8
10→1→11→12
11→9
12

```
start_bfs(){
Q← a queue
for v=1 to n
    if (v is not visited)
                bfs(v, Q);
}

bfs(v, Q){
  mark v visited
  Q.enqueue(v);
  while (Q is not empty){
     u=Q.dequeue()
     for each w adjacent to u {
        if (w is not visited){
            mark w visited;
            Q.enqueue(w)
         }
     }
   }
}
```

Breadth first

1→2→3→9
2→4→5
3→1→2→4→5→8
4→5→6
5→2→7
6→7
7→4→5
8
9→8
10→1→11→12
11→9
12

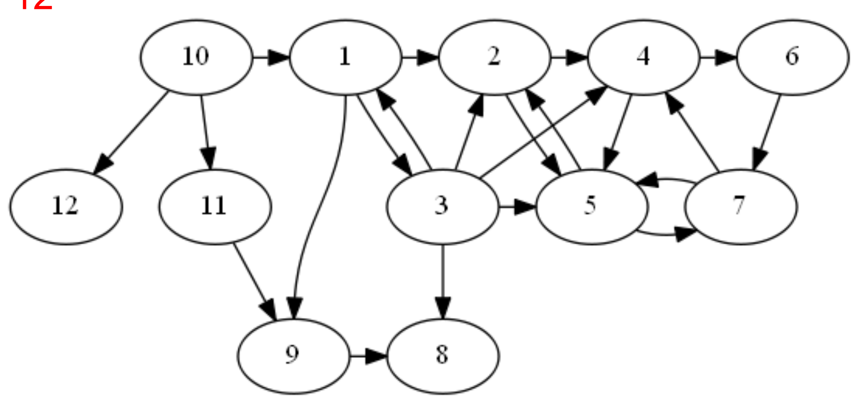


1, 2, 3, 9, 4, 5, 8, 6, 7, 10, 11, 12

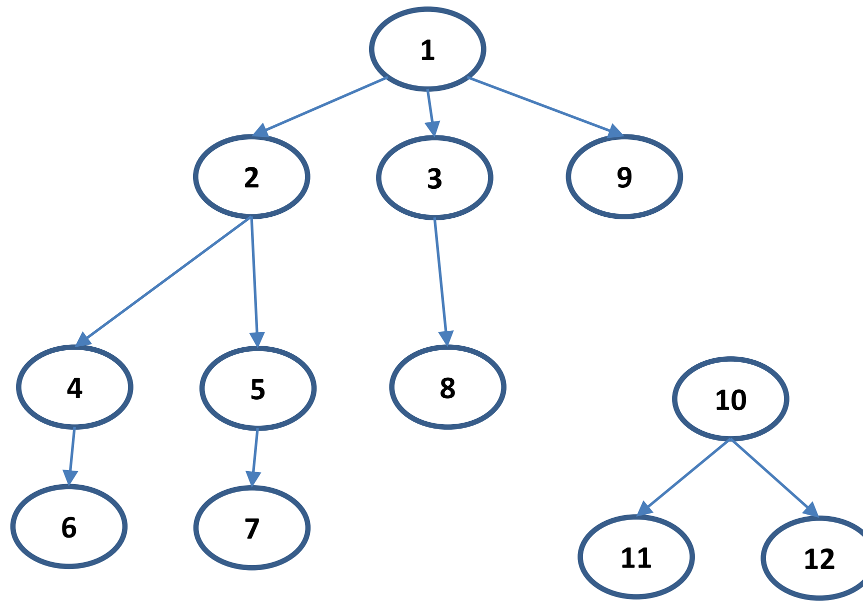
```
start_bfs(){
Q← a queue
for v=1 to n
    if (v is not visited)
            bfs(v, Q);
}

bfs(v, Q){
  mark v visited
  Q.enqueue(v);
  while (Q is not empty){
     u=Q.dequeue()
     for each w adjacent to u {
        if (w is not visited){
           mark w visited;
           Q.enqueue(w)
        }
     }
  }
}
```

Breadth first

1→2→3→9
2→4→5
3→1→2→4→5→8
4→5→6
5→2→7
6→7
7→4→5
8
9→8
10→1→11→12
11→9
12

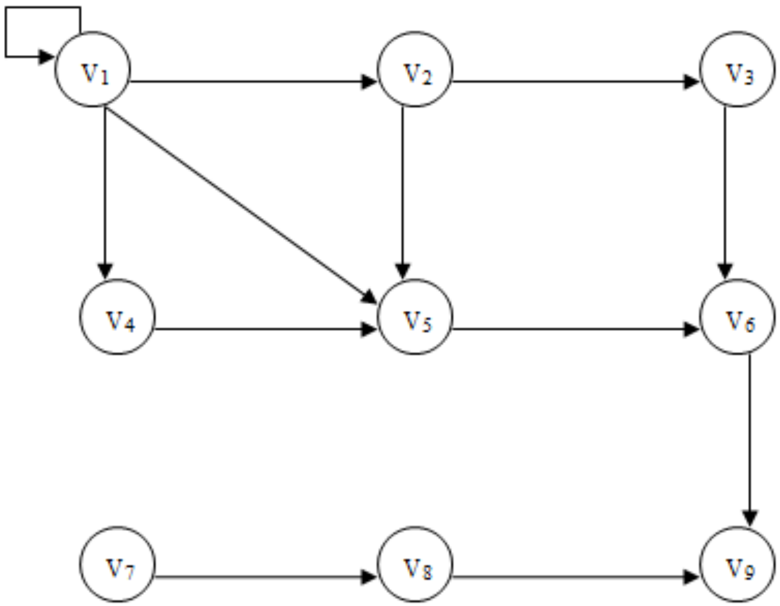


BFS tree:
Whenever dequeue v, the neighbors are added as children



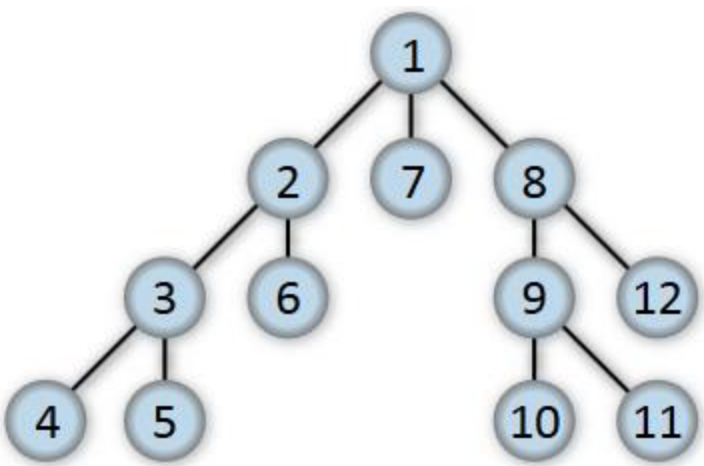
1, 2, 3, 9, 4, 5, 8, 6, 7, 10, 11, 12

Exercise – Breadth first



Shortest path and BFS

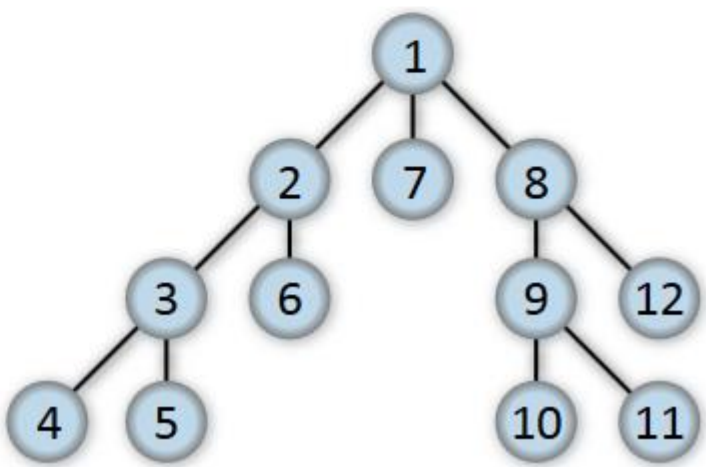
- BFS can only be used to find shortest path in a graph if
  - There are no loops
  - All edges have same weight or no weight
  1. Start from the source and perform a BFS
  2. Along with BFS order, keep the shortest paths for each vertex



Shortest path and BFS

- BFS can only be used to find shortest path in a graph if
  - There are no loops
  - All edges have same weight or no weight

1. Start from the source and perform a BFS
2. Along with BFS order, keep the shortest paths for each vertex

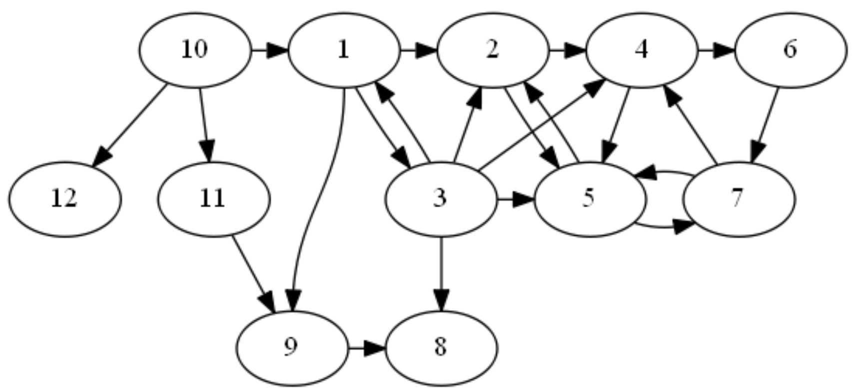


Along with the BFS order, update the neighbor's distance if unvisited

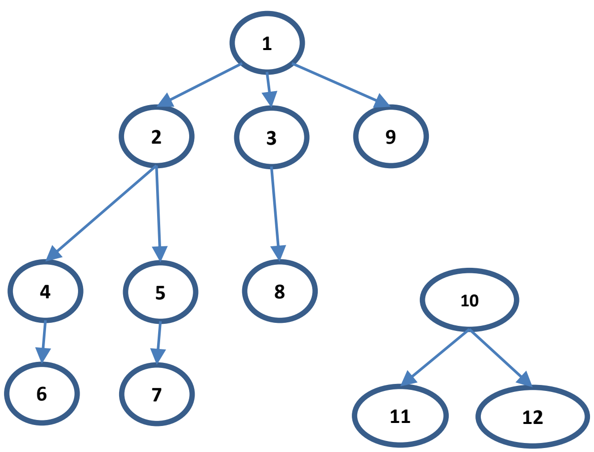
BFS	1	2	7	8	3	6	9	12	4	5	10	11
Prev		1	1	1	2	2	8	8	3	3	9	9
Path	0	1	1	1	2	2	2	2	3	3	3	3

Shortest path and BFS

1. Start from the source and perform a BFS
2. Along with BFS order, keep the shortest paths for each vertex



1→2→3→9
2→4→5
3→1→2→4→5→8
4→5→6
5→2→7
6→7
7→4→5
8
9→8
10→1→11→12
11→9
12



Along with the BFS order, update the neighbor's distance if unvisited

BFS	1	2	3	9	4	5	8	6	7	10	11	12
Prev		1	1	1	2	2	3	4	5		10	10
Path	0	1	1	1	2	2	2	3	3	Inf	Inf	inf

Applications of BFS

- Graphs
  - Shortest path(BFS + Dijkstra's algorithm)
  - Minimum spanning trees
  - Cycle detection
- Search Engines: index building for visited links (crawlers)
- Social Networks: find people with a given distance from someone
- Networking and broadcasting: find neighboring nodes
- Garbage collectors: better locality of reference