GRAPH SEARCH

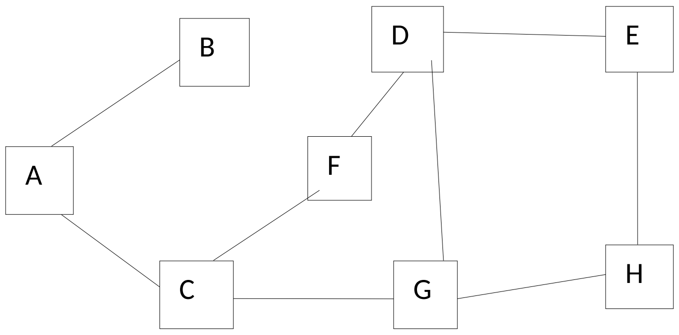
- Breadth first Search
- Depth first Search

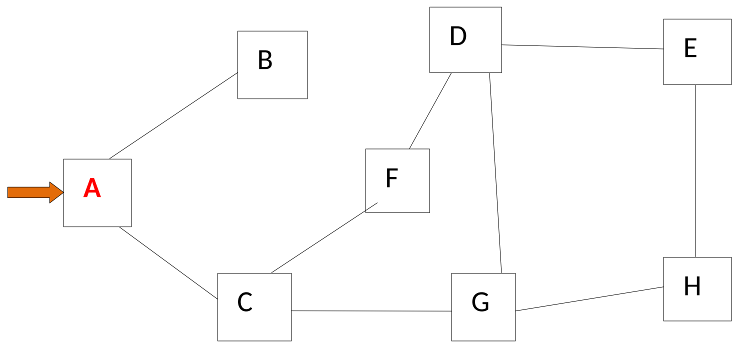
Usage

- Transportation networks (airline carrier, airports as node and direct flights as edges (direct edge).
- Communication networks (a collection of computers as nodes and the physical link between them as edges).
- Information networks (World Wide Web can be viewed as directed graph, the Web pages are nodes and the hyperlink between the pages are directed edges).
- Social Network (People are nodes and friendship is an edge).

Breadth first Search(level order traversal): BFS begins at a root node and inspects all the neighboring nodes. Then for each of those neighbor nodes in turn, it inspects their neighbor nodes which were unvisited, and so on

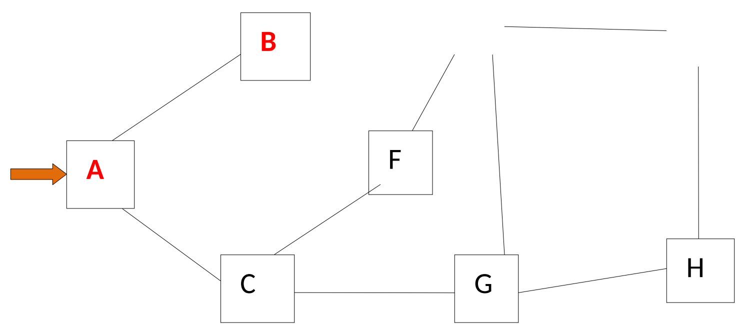
Example:





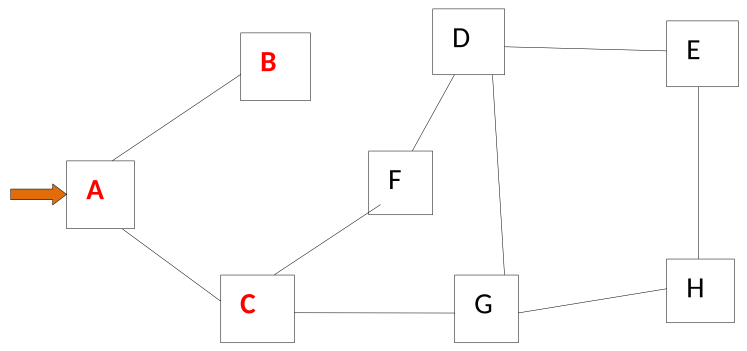
For implementation of BFS we use queue as a data structure

Q {A}
Delete from queue vertex {A} & add their adjacent not visited vertices one bye one to the queue.
Print{}



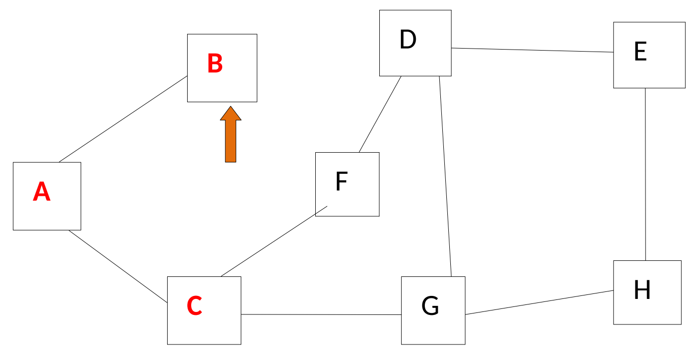
Q {B}

Print{A}

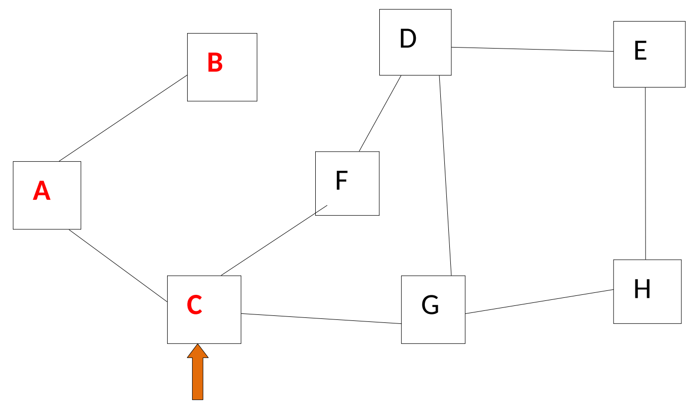


Q {B,C}

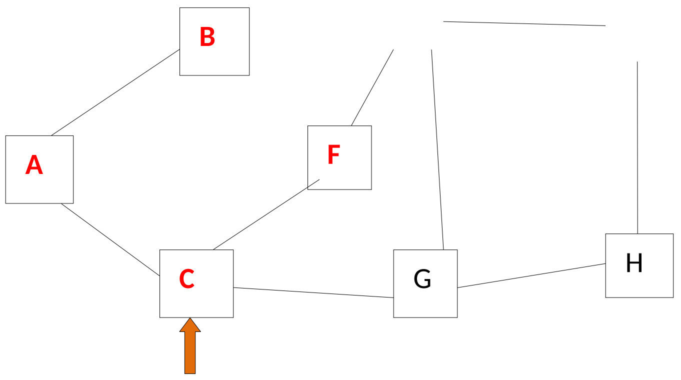
Print{A}



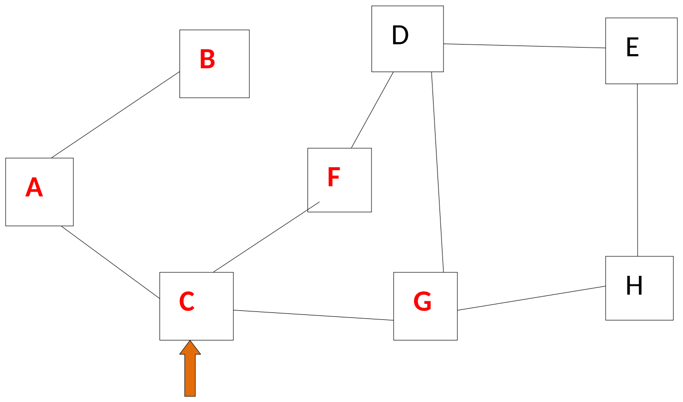
Q {B,C}
Delete vertex {B} & add not visited adjacent vertices of B to queue
Print{A}



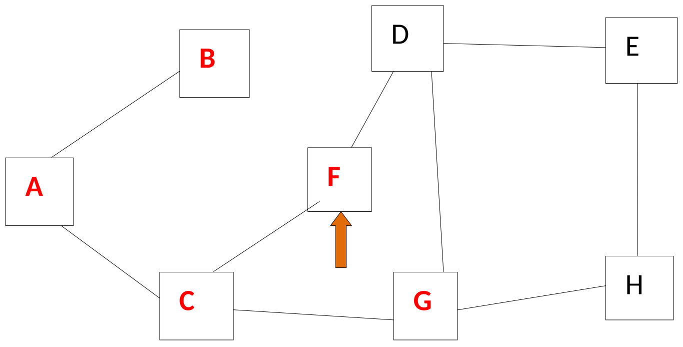
Q {C}
Delete vertex {C} & add not visited adjacent vertices of C to queue one bye one
Print{A, B}



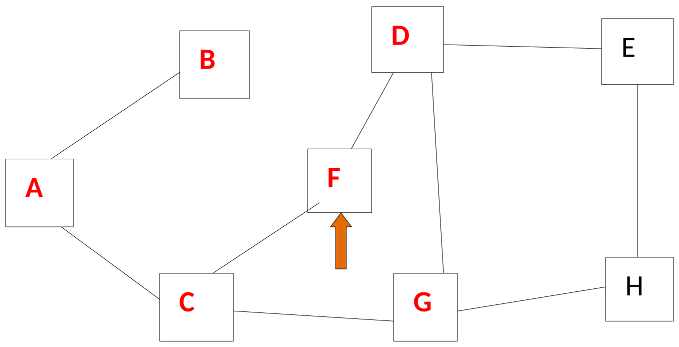
Q {F}
Print{A, B,C}



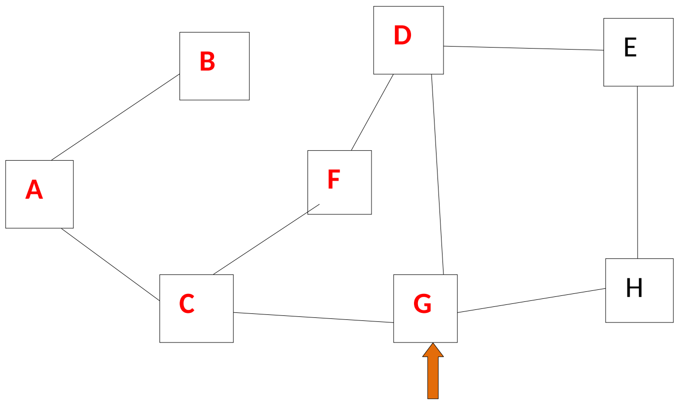
Q {F,G}
Print{A, B,C}



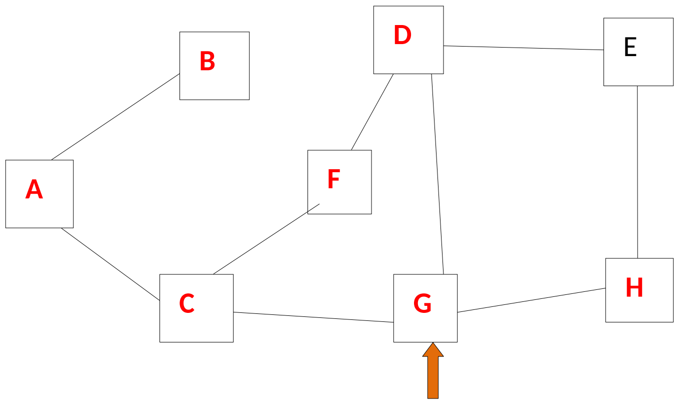
Q {F,G}
Delete vertex {F} & add not visited adjacent vertices of F to queue one bye one
Print{A, B,C}



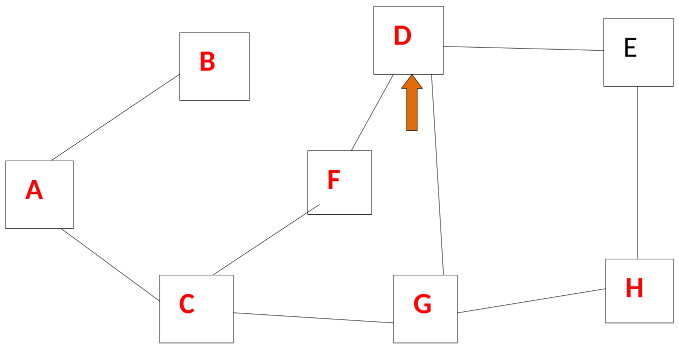
Q {G,D}
Print{A, B,C,F}



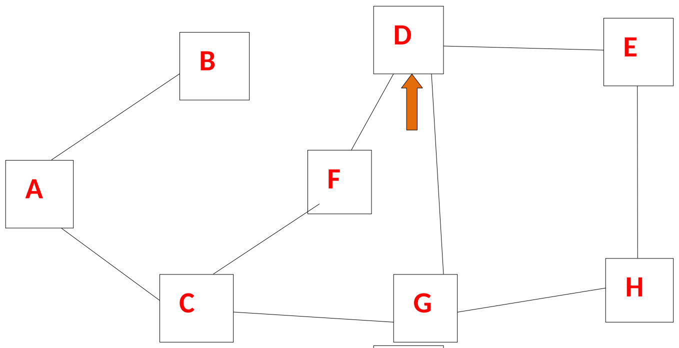
Q {G,D}
Delete vertex {G} & add not visited adjacent vertices of G to queue one bye one
Print{A, B,C,F}



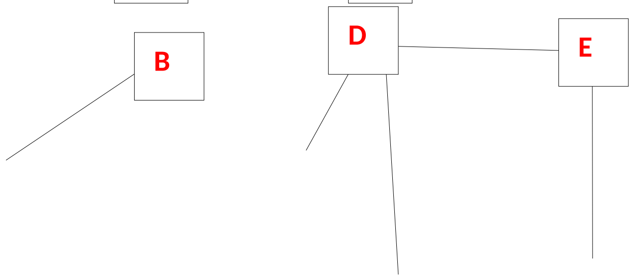
Q {D,H}
Print{A, B,C,F,G}



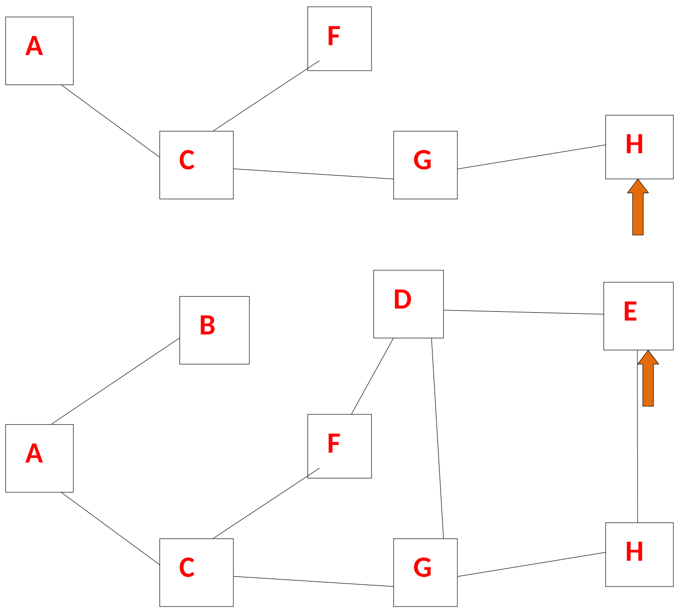
Q {D,H}
Delete vertex {D} & add not visited adjacent vertices of D to queue one bye one
Print{A, B,C,F,G}



Q {H,E}
Print{A, B,C,F,G,D}

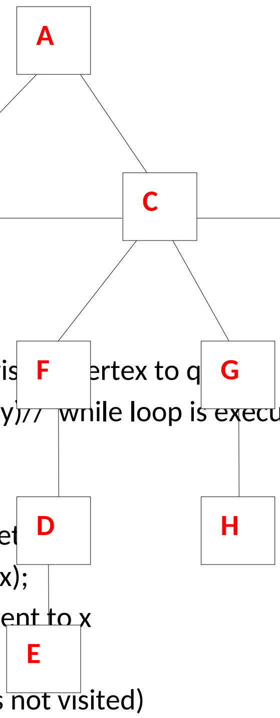


Q {E}
Delete vertex {H} & add not visited adjacent vertices of H to queue one bye one
Print{A, B,C,F,G,D,H}



Q {}
Delete vertex {E} & add not visited adjacent vertices of E to queue one bye one
Print{A, B,C,F,G,D,H,E}
Q is empty and we have traverse all the vertices in order **{ A, B,C,F,G,D,H,E }**

Tree after BFS run



```
BFS(V)
{
    Visited (v)=1;
    Add(v, Q);   // add visited vertex to queue
   While(Q is not empty)//  while loop is executing V times
   {
                  x= delete(Q);
                  print (x);
          For all w adjacent to x
          {
                  If( w is not visited)
                  {    visited(w) =1;
                       Add (w, Q);
                  }
                  }
    }
}
```

B

1) If we represent the graph G by adjacency matrix then the running time of BFS algorithm is O(V^2) because for find adjacent vertices of a particular vertex in adjacency matrix we require V time and for all the vertices running time will be V^2 , where V is the number of nodes.

2) If we represent the graph G by link lists then the running time of BFS algorithm is O(E + V), where E is the number of edges and V is the number of nodes.

BFS Applications

Breadth-first search can be used to solve many problems in graph theory, for example:

- Finding all nodes within one connected component
- Finding the shortest path between two nodes u and v
- Finding the diameter of a graph.
- To obtain number of connected component we use BFS
- If in the uniform weighted graphs (all edges weight are same) to find out shortest path from source we can use BFT.
- Testing a graph for bipartiteness. If there are two vertices x,y in the same level (layer) L_i that are adjacent then the graph is not bipartite.

Depth-First Search (DFS)

We don’t visit the nodes level by level. As long as there is an unvisited node adjacent to the current visited node we continue. Once we are stuck, trace back and go to a different branch.DFS use stack as data structure.

Depth-First Search Algorithm

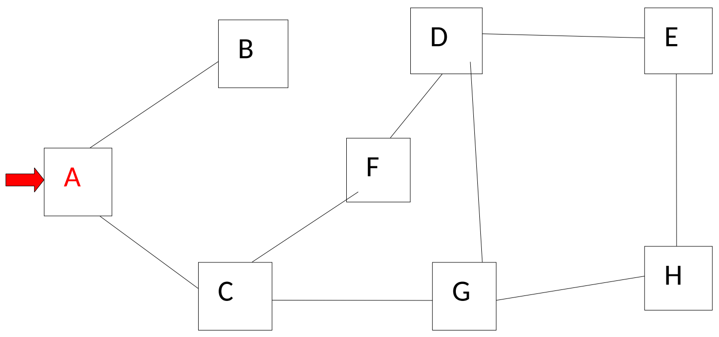
```
DFS(V)
{
    Visited (v)=1;
    Print(v);
    For all w adjacent to v {
            If( w is not visited)
                    {
                     DFS(w);
                    }

    }
}
```

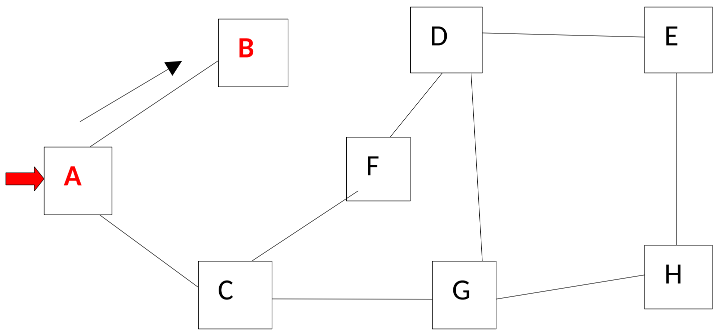
DFS Running time

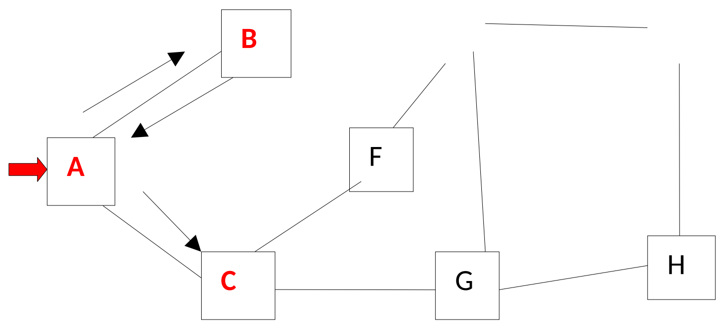
3) If we represent the graph G by adjacency matrix then the running time of DFS algorithm is O(V^2) because for find adjacent vertices of a particular vertex in adjacency matrix we require V time and for all the vertices running time will be V^2 , where V is the number of nodes.

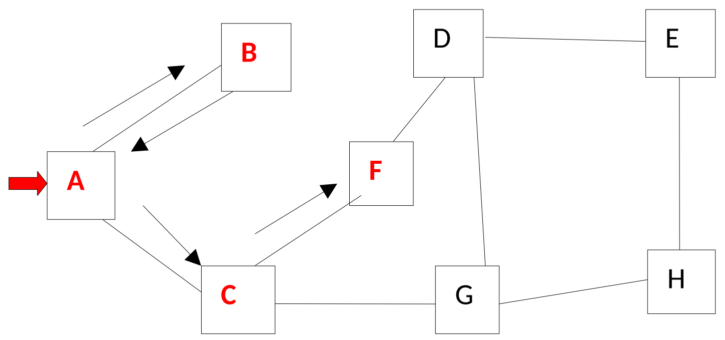
4) If we represent the graph G by link lists then the running time of DFS algorithm is O(E + V), where E is the number of edges and V is the number of nodes.

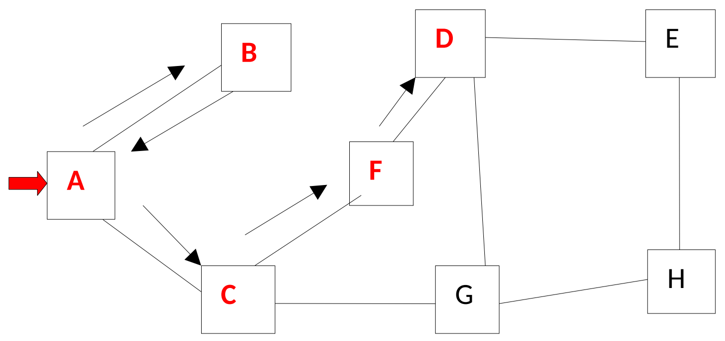
Example:

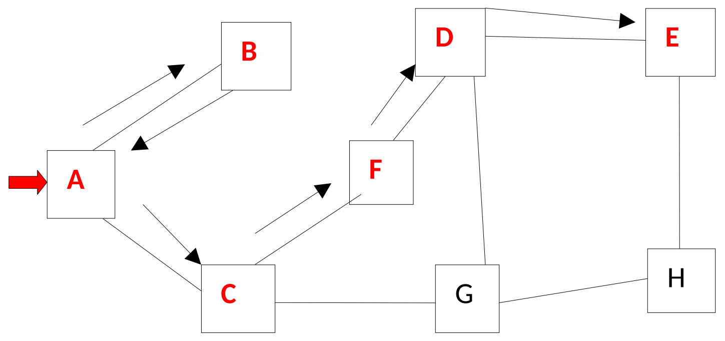


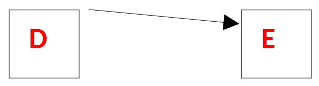


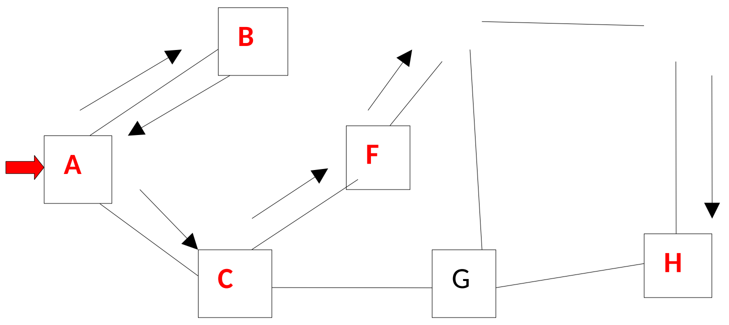

B
A
F
C
G
H

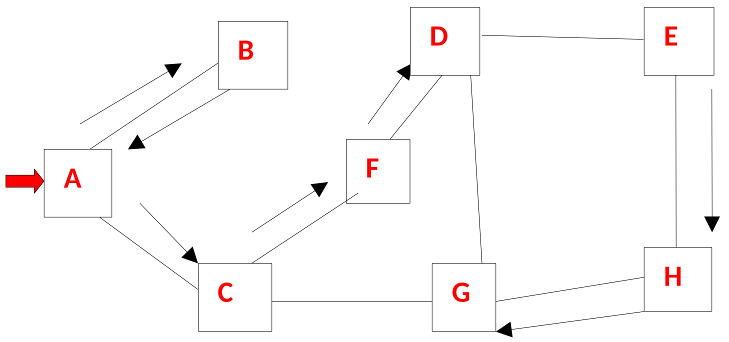

D
E
B
A
F
C
G
H


D
E
B
A
F
C
G
H


D
E
B
A
F
C
G
H


D
E

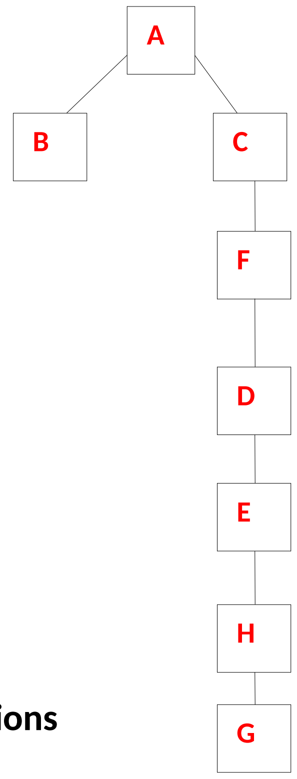




DFS{A,B,C,F,D,E,H,G}

Edges of Graph G that are not in DFS –{C-G, D-G}

Tree after DFS run



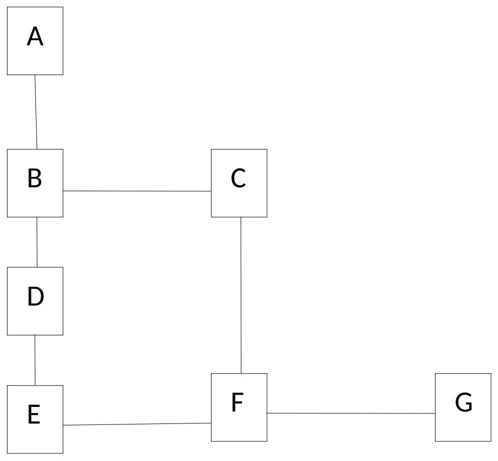
DFS Applications

- We use DFS to find a cycle or shortest cycle.
- We also use DFS to find strong components in digraph.
- Check whether graph is connected or not.
- Finding the number of connected component.
- Find articulation point in a graph. In the given connected graph by deleting any vertex if the graph is disconnected then the deleted vertex is called articulation point.

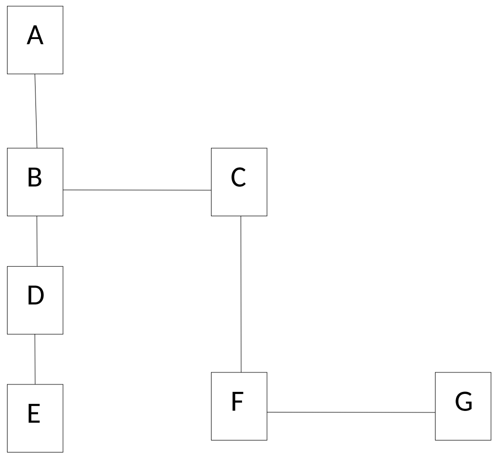
Example of BFS & DFS

V{A,B,C,D,E,F,G}

E{AB,BD,DE,BC,EF,CF,FG}



BFS-{AB,BD,BC,DE,CF,FG}



DFS-{AB,BD,DE,EF,FG,FC}