Graphs

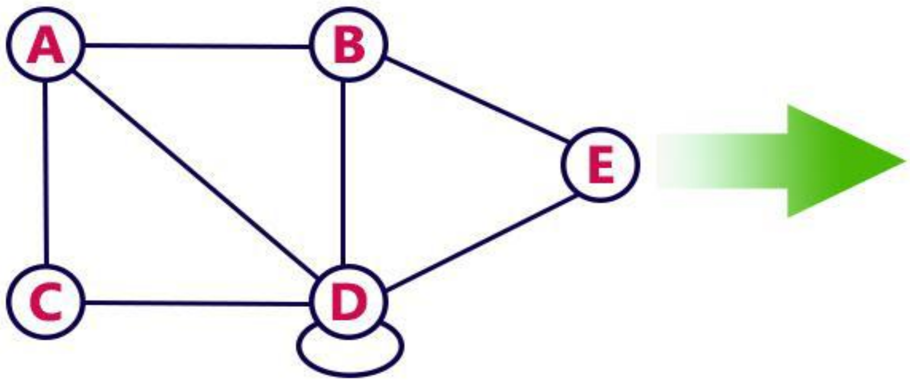
- A graph **G = (V , E)**

  V → Set of vertices

  E → Set of edges.

- Representations of graph
  - Adjacency Matrix
  - Adjacency List

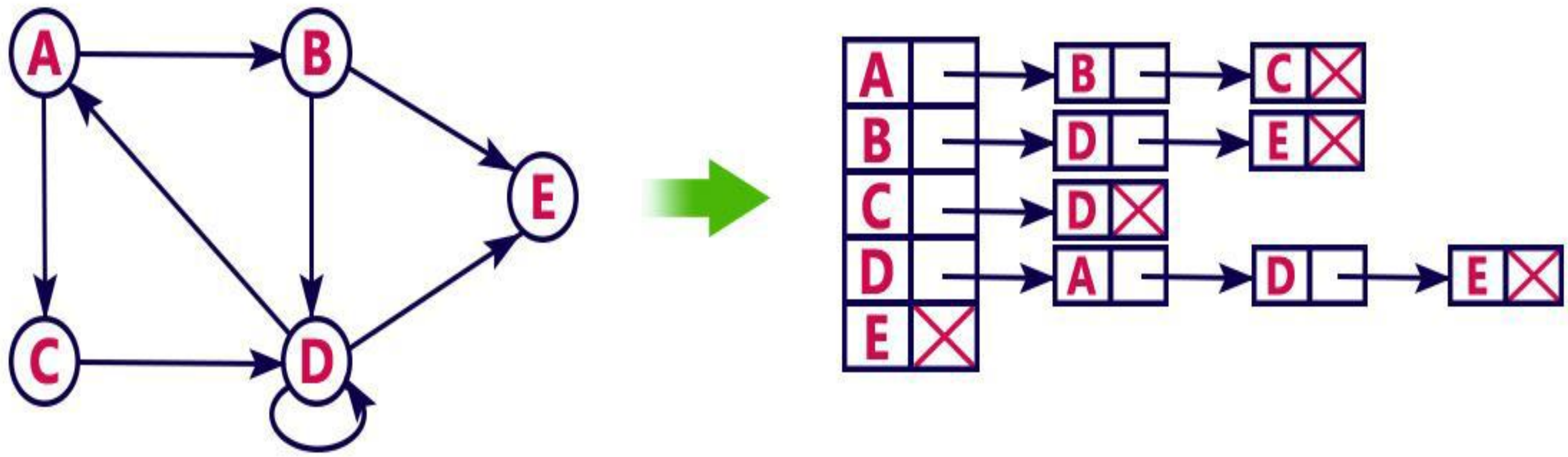
Adjacency Matrix



$$
\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array}
\begin{array}{c}
\begin{array}{ccccc} A & B & C & D & E \end{array} \\
\begin{bmatrix}
0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 1 & 0
\end{bmatrix}
\end{array}
$$

- It is a 2D array(say adj[][]) of size |V|x|V| where |V| is the number of vertices in a graph.
  - If adj[i][j] = 1, then there is an edge from vertex i to vertex j.
  - For a weighted graph if adj[i][j] = w, then there is an edge from vertex i to vertex j with weight w
- Adjacency matrix for undirected graph is always symmetric.

Adjacency List



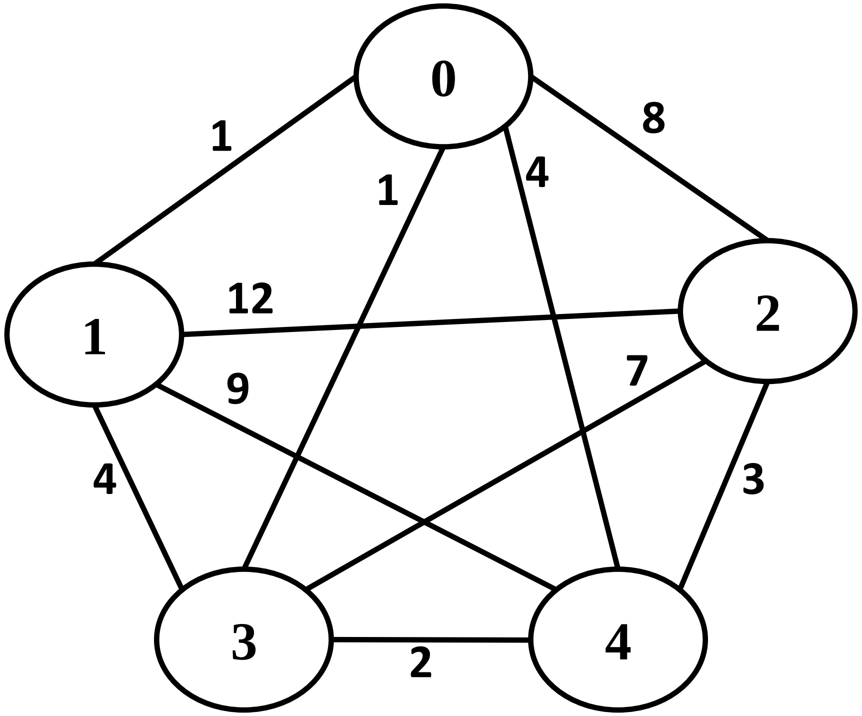
- An array of linked lists is used.
- Size of the array is equal to number of vertices.
- An entry array[i] represents the linked list of vertices adjacent to the **i^th^** vertex.
- This representation can also be used to represent a weighted graph. The weights of edges can be stored in nodes of linked lists.

Type of Graphs

- **Undirected Graph:** A graph with only undirected edges.
- **Directed Graph:** A graph with only directed edges.
- **Directed Acyclic Graphs(DAG):** A directed graph with no cycles.
- **Cyclic Graph:** A directed graph with at least one cycle.
- **Weighted Graph:** It is a graph in which each edge is given a numerical weight.
- **Disconnected Graphs:** An undirected graph that is not connected.

Qtn) Consider a complete undirected graph with vertex set {0, 1, 2, 3, 4}. Entry Wij in the matrix W below is the weight of the edge {i, j}. Construct the graph.

$$W = \begin{pmatrix} 0 & 1 & 8 & 1 & 4 \\ 1 & 0 & 12 & 4 & 9 \\ 8 & 12 & 0 & 7 & 3 \\ 1 & 4 & 7 & 0 & 2 \\ 4 & 9 & 3 & 2 & 0 \end{pmatrix}$$



Graph Traversal Algorithms

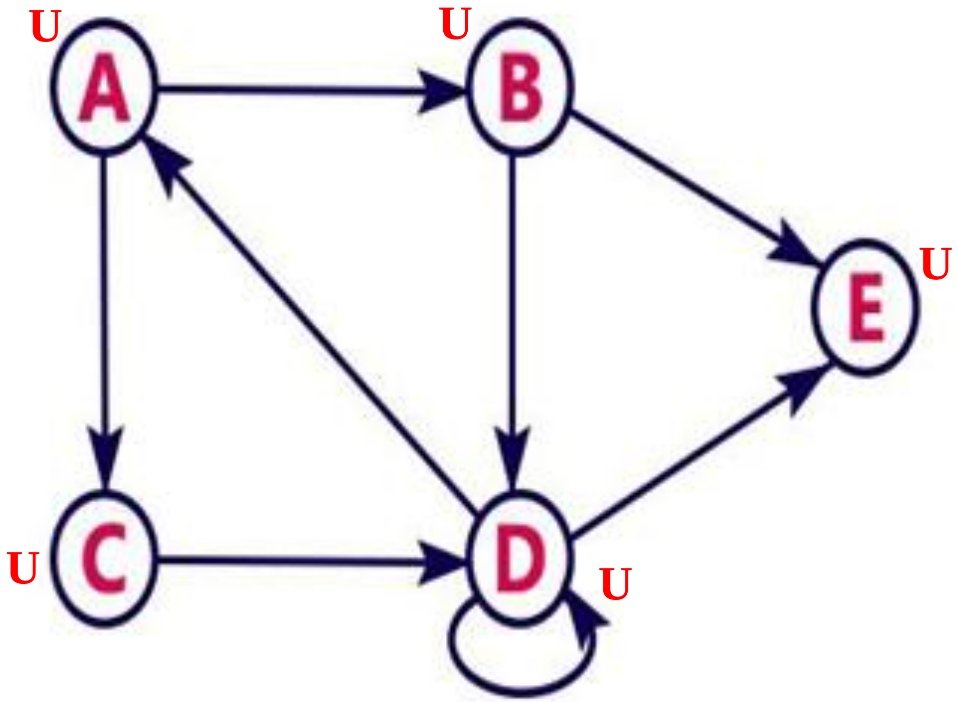
- Breadth First Search(BFS)
- Depth First Search(DFS)

Breadth First Search(BFS) Algorithm

Algorithm BFS(G, u)

1. Set all nodes are unvisited
2. Mark the starting vertex u as visited and put it into an empty Queue Q
3. While Q is not empty
    1. Dequeue v from Q
    2. While v has an unvisited neighbor w
        1. Mark w as visited
        2. Enqueue w into Q
4. If there is any unvisited node x
    1. Visit x and Insert it into Q. Goto step 3

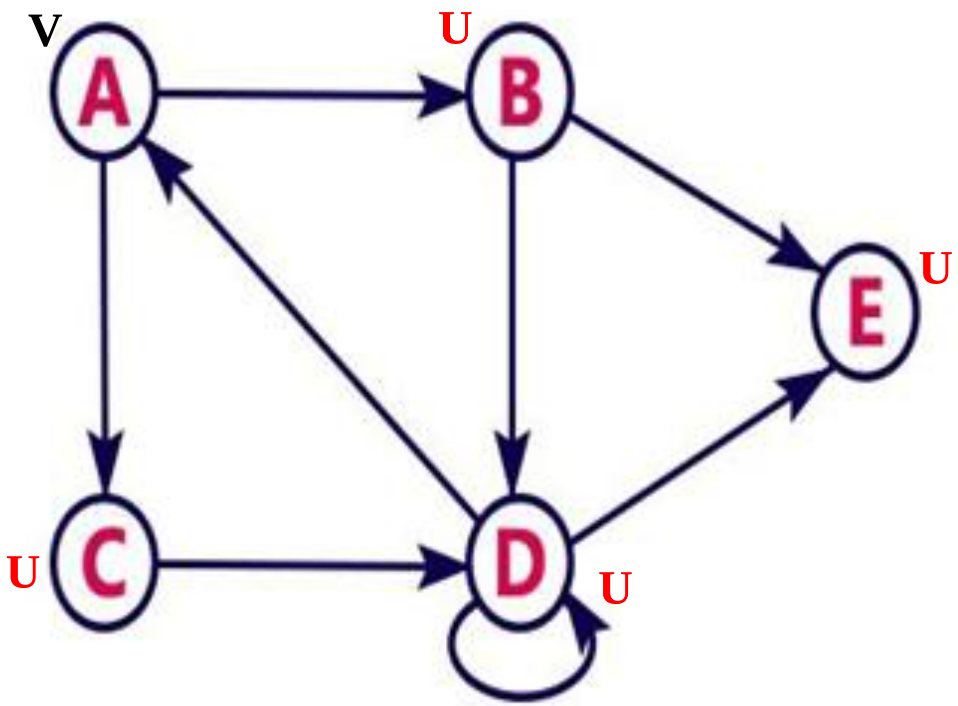
Breadth First Search(BFS) Example



BFS Traversal:

Q:[]

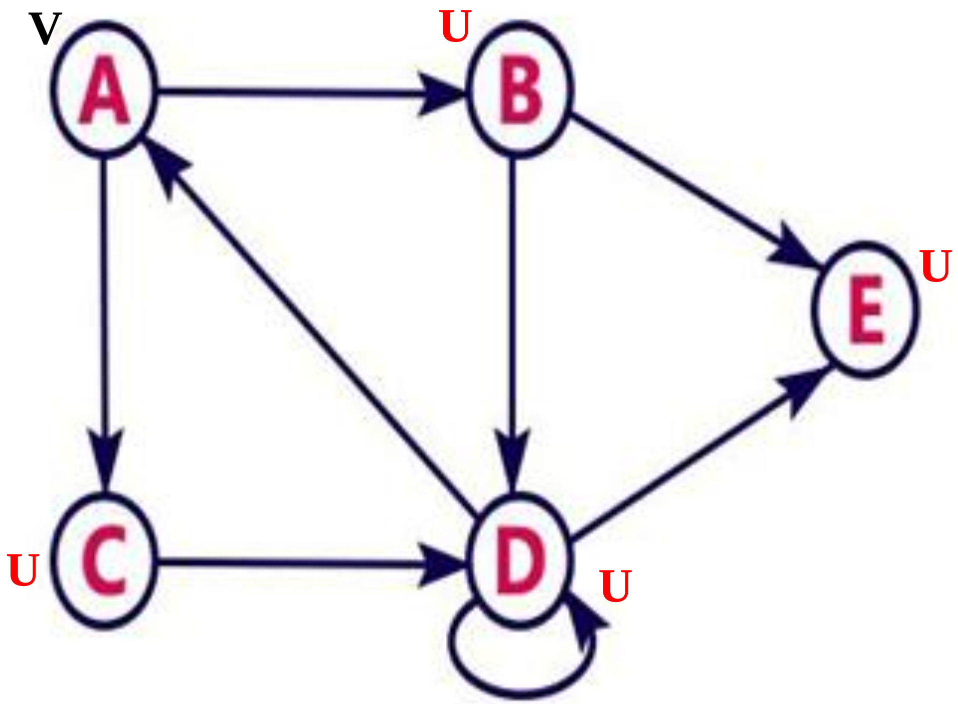
Breadth First Search(BFS) Example



BFS Traversal: A

Q:[A]

Breadth First Search(BFS) Example

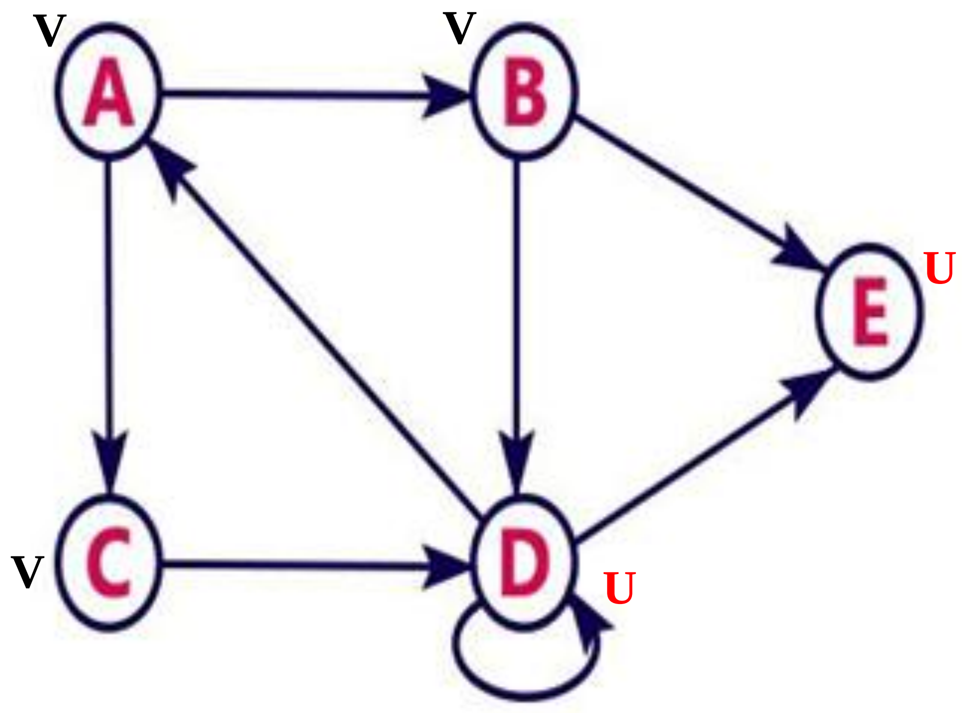


BFS Traversal: A

Q:[]

v = A

Breadth First Search(BFS) Example

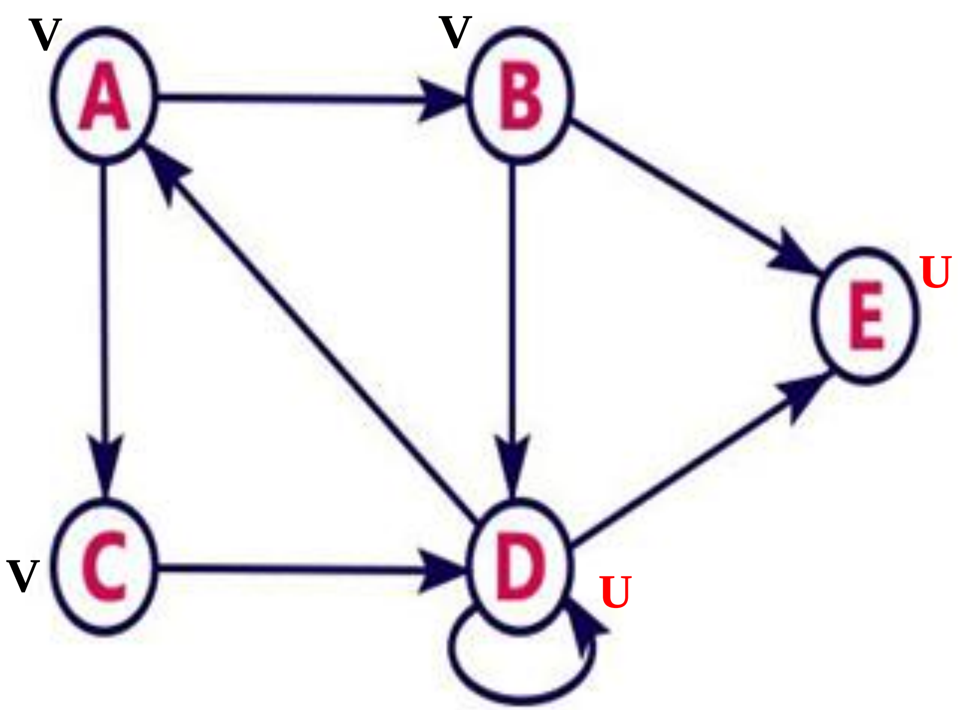


BFS Traversal: A B C

Q:[B,C]

v = A

Breadth First Search(BFS) Example

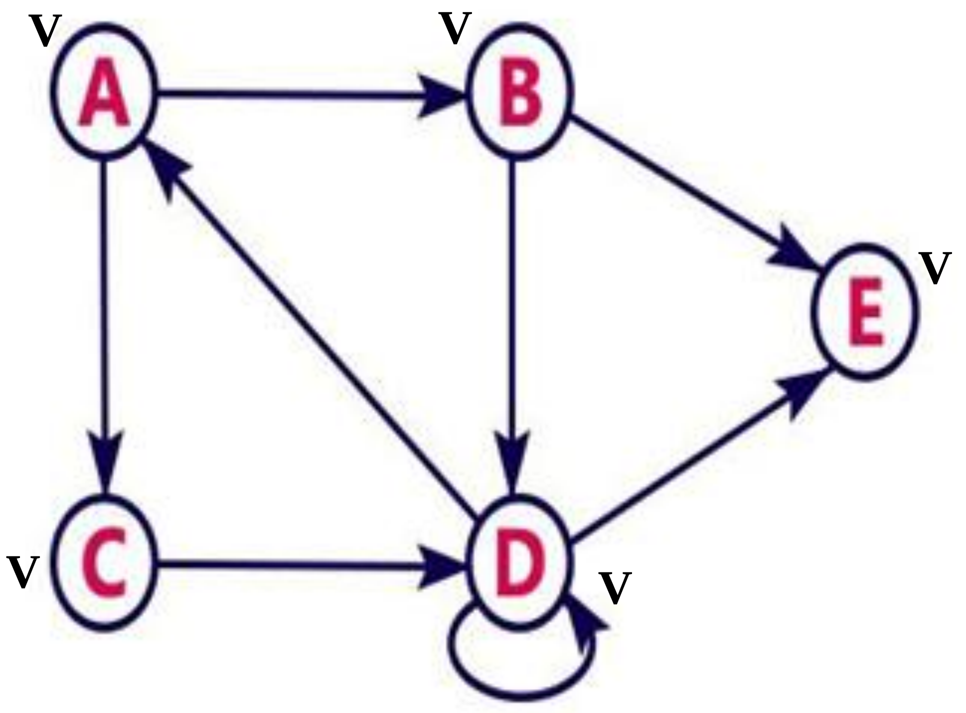


BFS Traversal: A B C

Q:[C]

v = B

Breadth First Search(BFS) Example

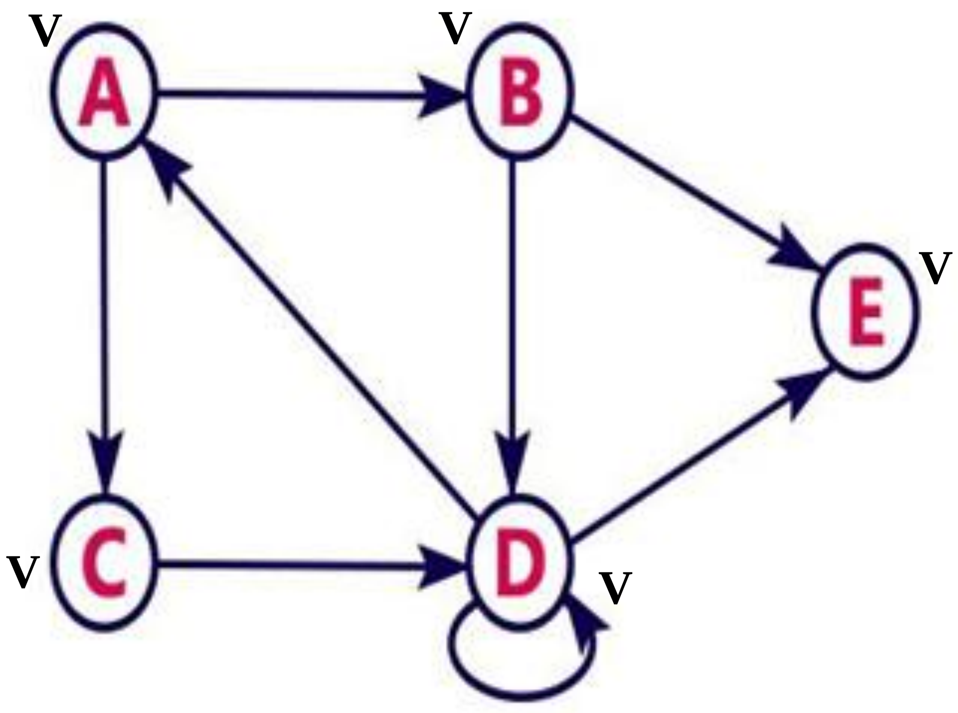


BFS Traversal: A B C D E

Q:[C,D,E]

v = B

Breadth First Search(BFS) Example

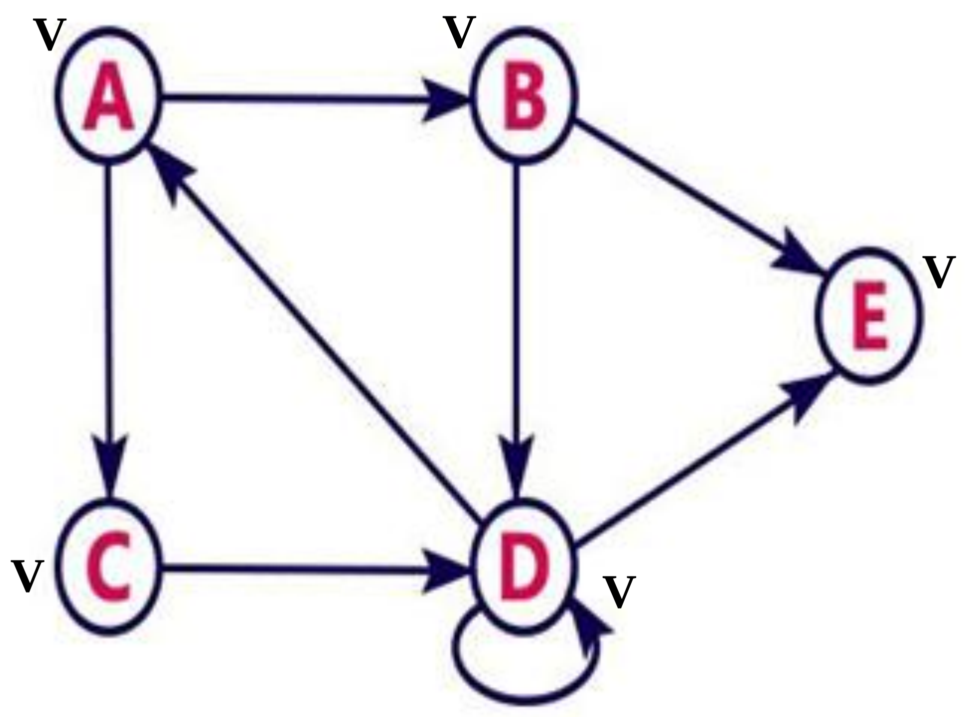


BFS Traversal: A B C D E

Q:[D,E]

v = C

Breadth First Search(BFS) Example

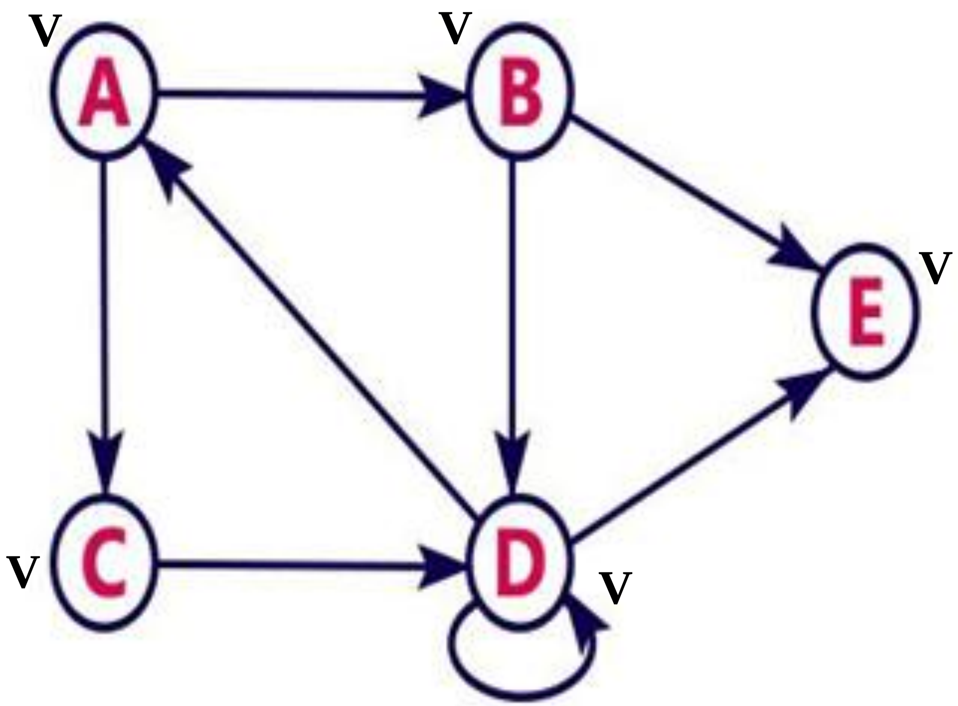


BFS Traversal: A B C D E

Q:[E]

v = D

Breadth First Search(BFS) Example

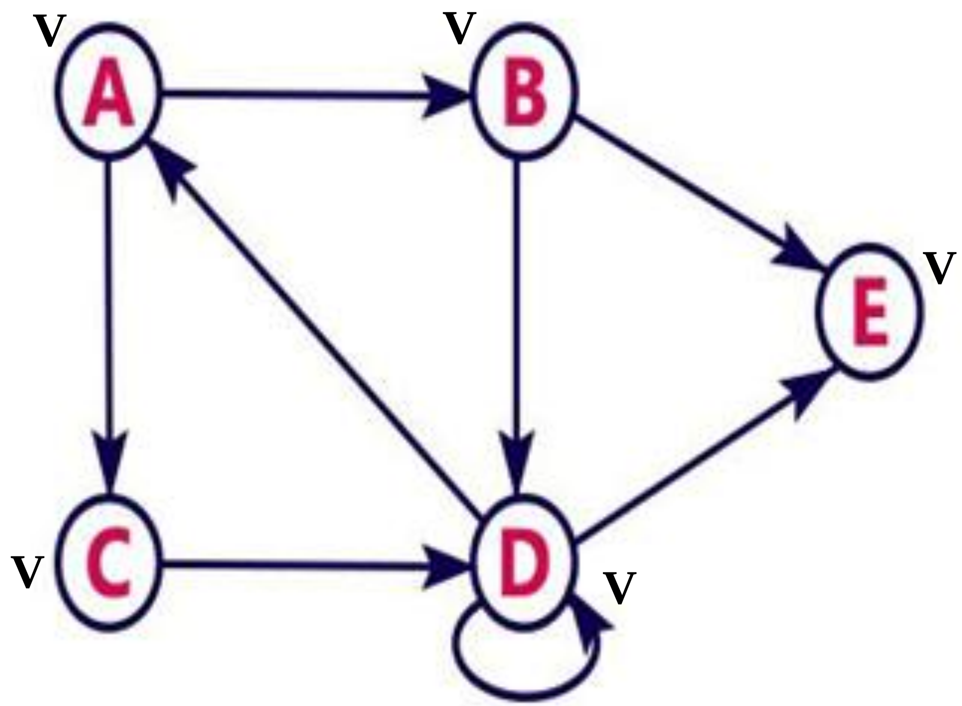


BFS Traversal: A B C D E

Q:[]

v = E

Breadth First Search(BFS) Example



BFS Traversal: A B C D E

Q:[]

BFS Algorithm Complexity

- If the graph is represented as an **adjacency list**
  - Each vertex is enqueued and dequeued atmost once.
  - Each queue operation take O(1) time.
  - So the time devoted to the queue operation is **O(V).**

  - The adjacency list of each vertex is scanned only when the vertex is dequeued.
  - Each adjacency list is scanned atmost once.
  - Sum of the lengths of all adjacency list is |E|.
  - Total time spend in scanning adjacency list is **O(E)**

  - Time complexity of BFS = O(V) + O(E) = **O(V+ E)**

BFS Algorithm Complexity

- If the graph is represented as an **adjacency list**
  - **In a dense graph:**
    - $E=O(V^2)$
    - Time complexity= $O(V) + O(V^2) =$ $\mathbf{O(V^2)}$

- If the graph is represented as an **adjacency matrix**
  - There are $|V|^2$ entries in the adjacency matrix. Each entry is checked once
  - Time complexity of BFS = $\mathbf{O(V^2)}$

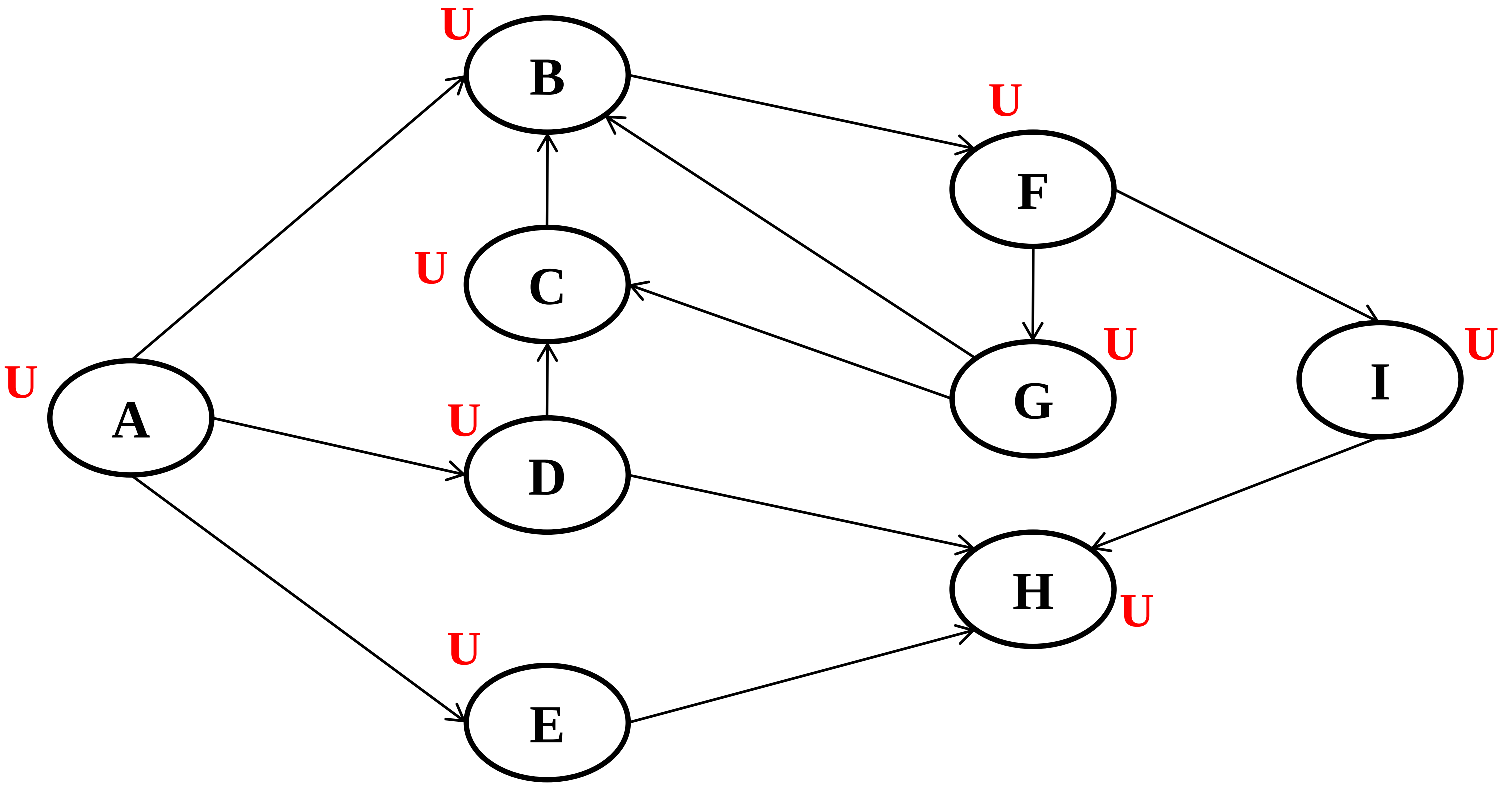
BFS Applications

- Finding shortest path between 2 nodes u and v, with path length measured by number of edges
- Testing graph for bipartiteness
- Minimum spanning tree for unweighted graph
- Finding nodes in any connected component of a graph
- Serialization/deserialization of a binary tree
- Finding nodes in any connected component of a graph

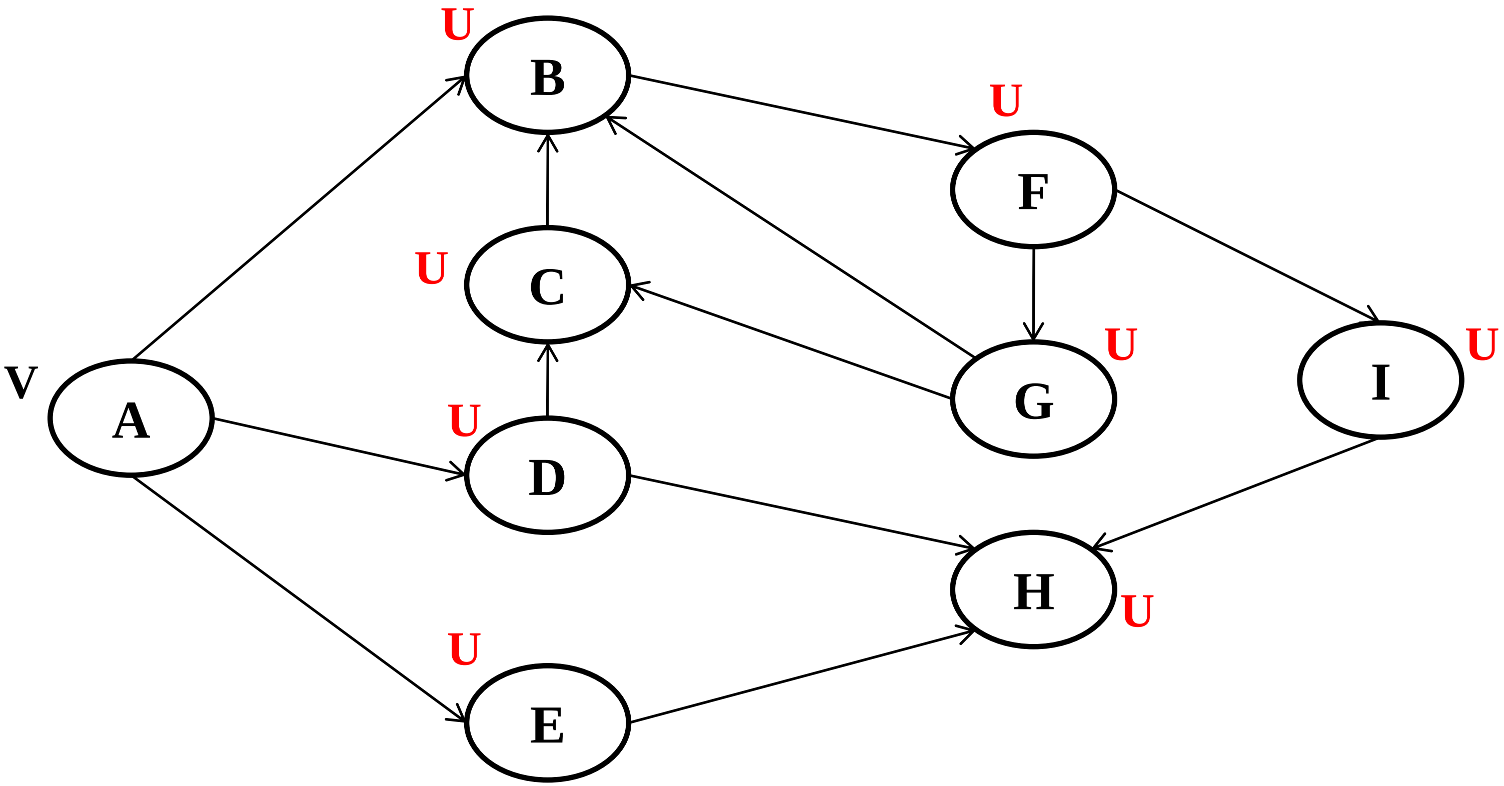
Depth First Search(DFS) Algorithm

```
Algorithm DFS(G, u)
{       Mark vertex u as visited
        For each adjacent vertex v of u
                if v is not visited
                        DFS(G, v)
 }
Algorithm main(G,u)
{       Set all nodes are unvisited.
        DFS(G, u)
        For any node x which is not yet visited
                DFS(G, x)
}
```

Depth First Search(DFS) Example

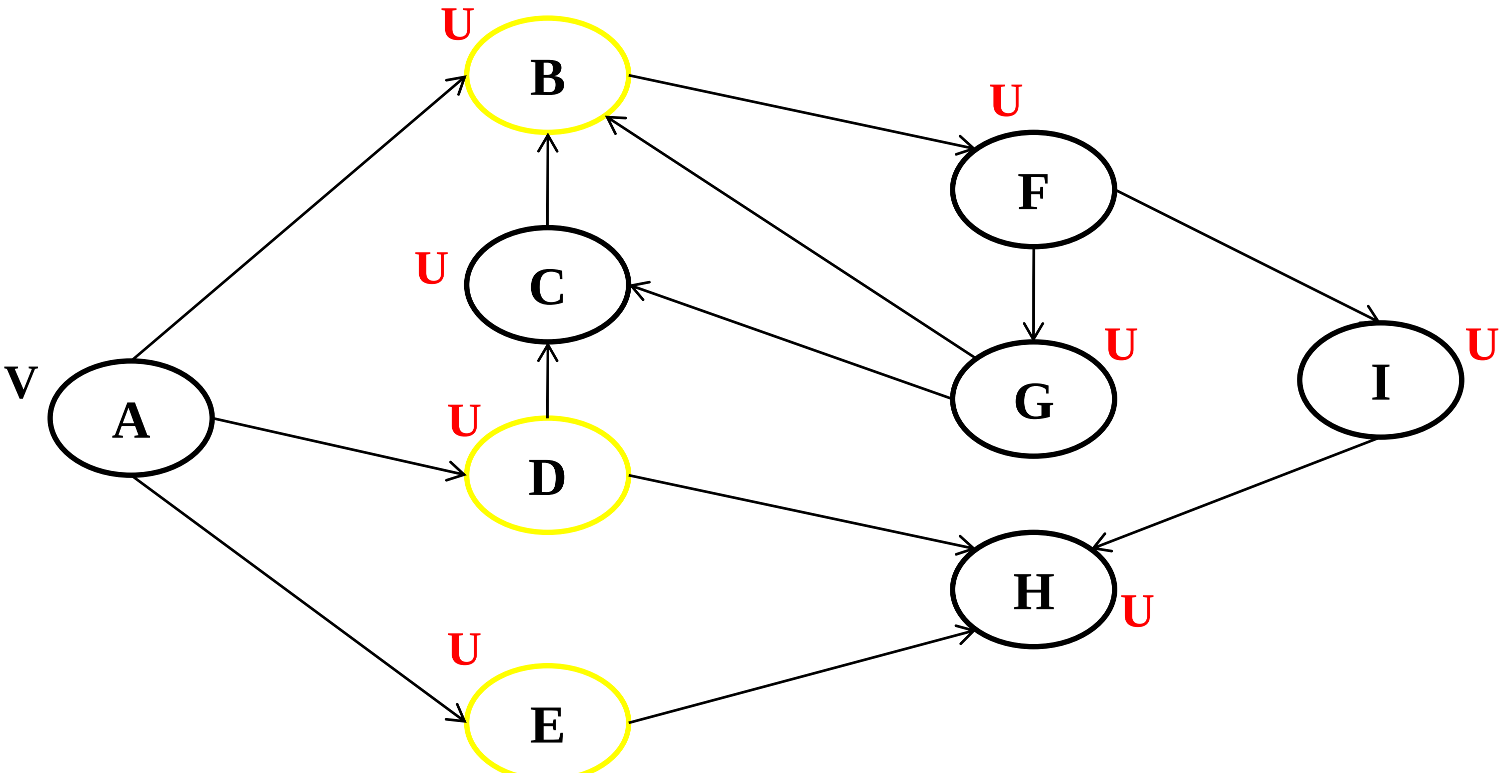


Depth First Search(DFS) Example



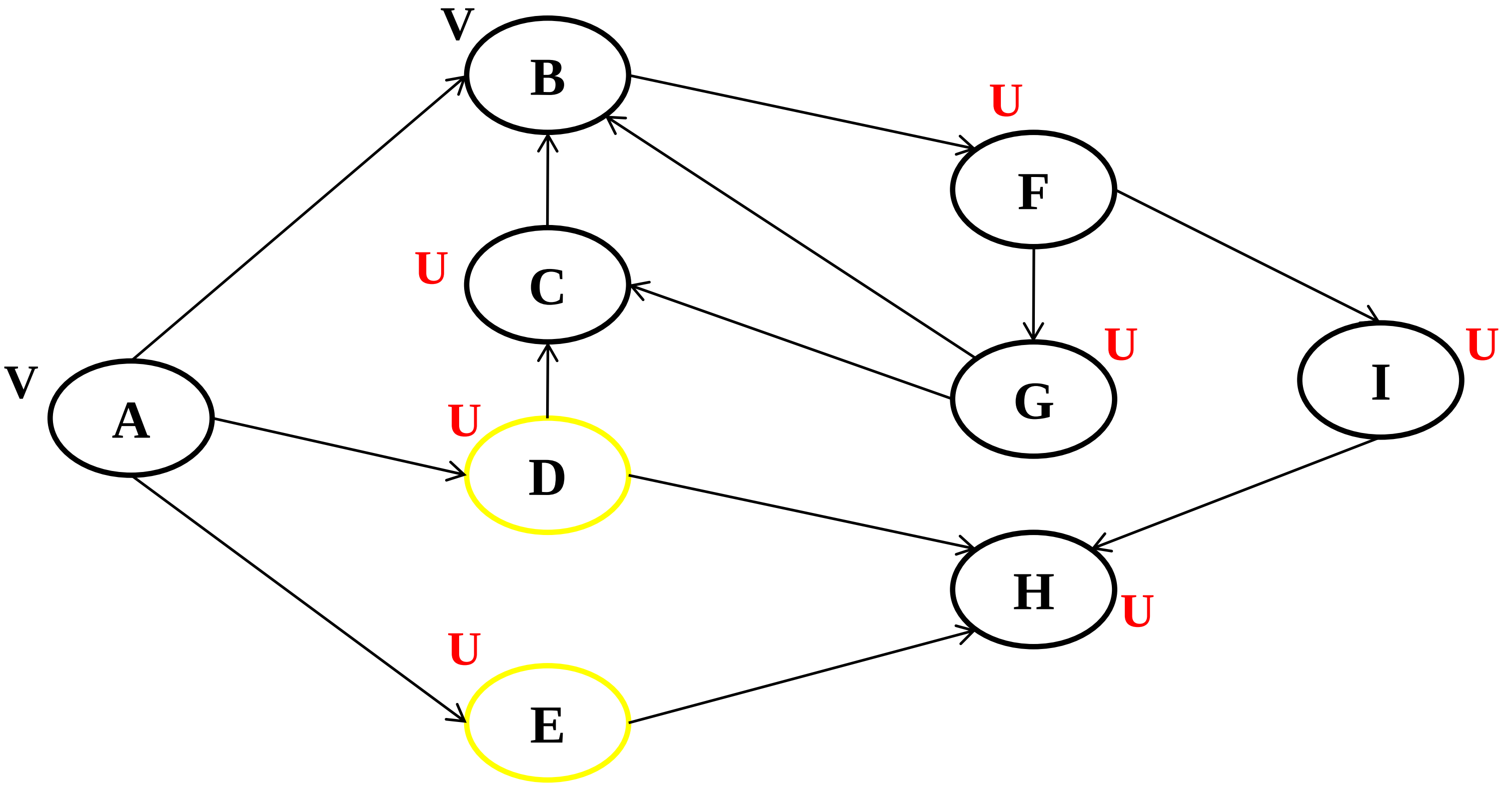
DFS Traversal: A

Depth First Search(DFS) Example



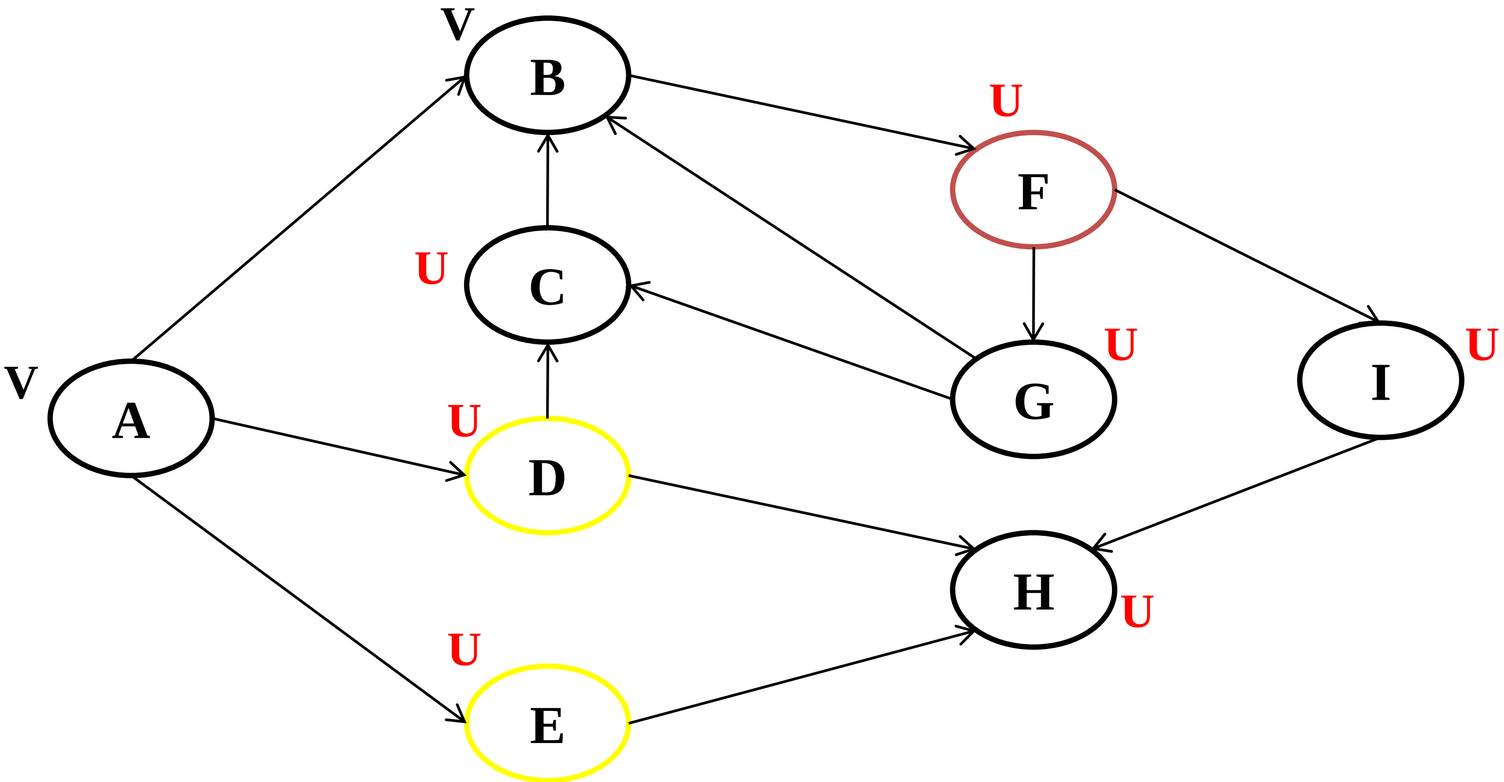
DFS Traversal: A

Depth First Search(DFS) Example



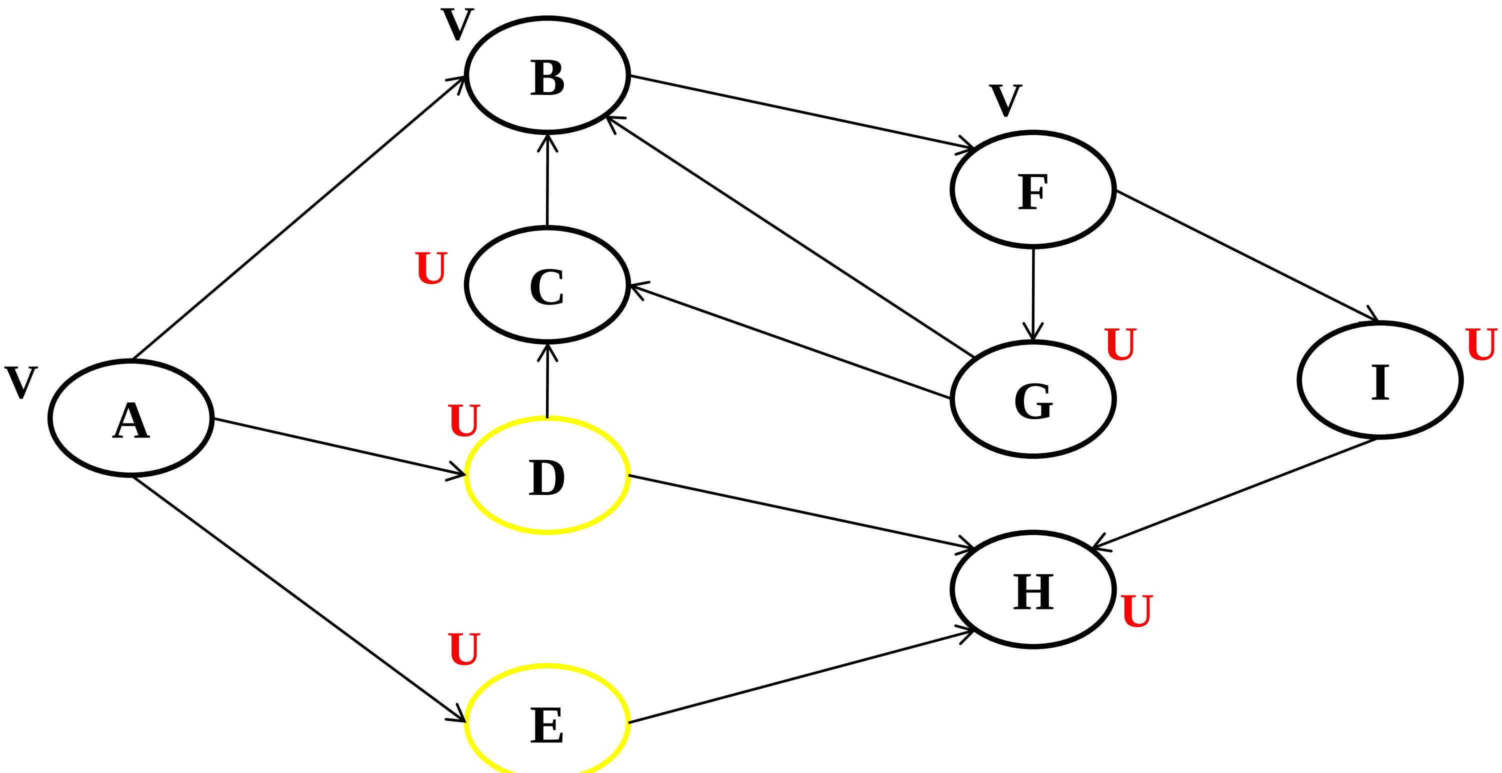
DFS Traversal: A B

Depth First Search(DFS) Example



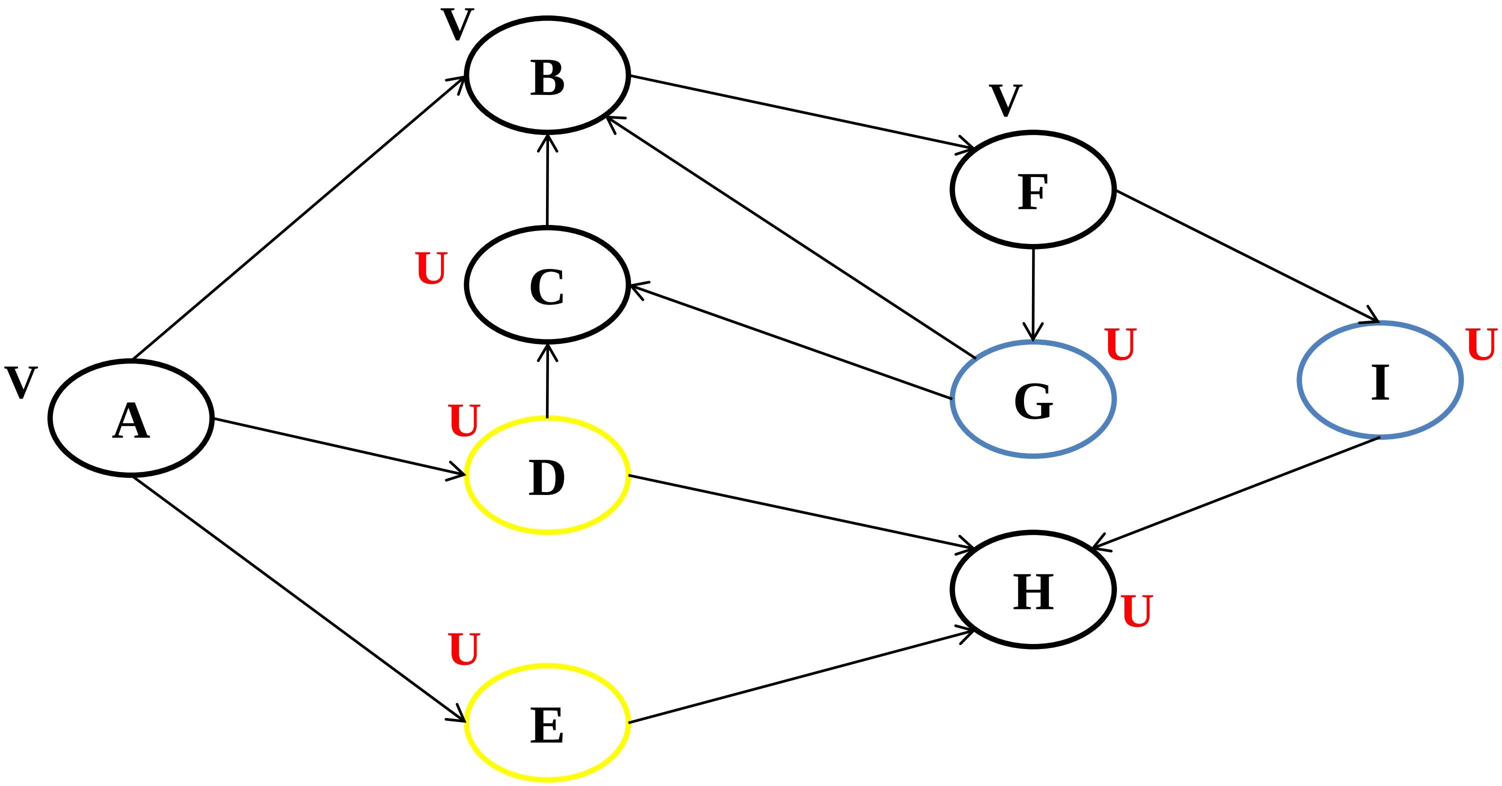
DFS Traversal: A B

Depth First Search(DFS) Example



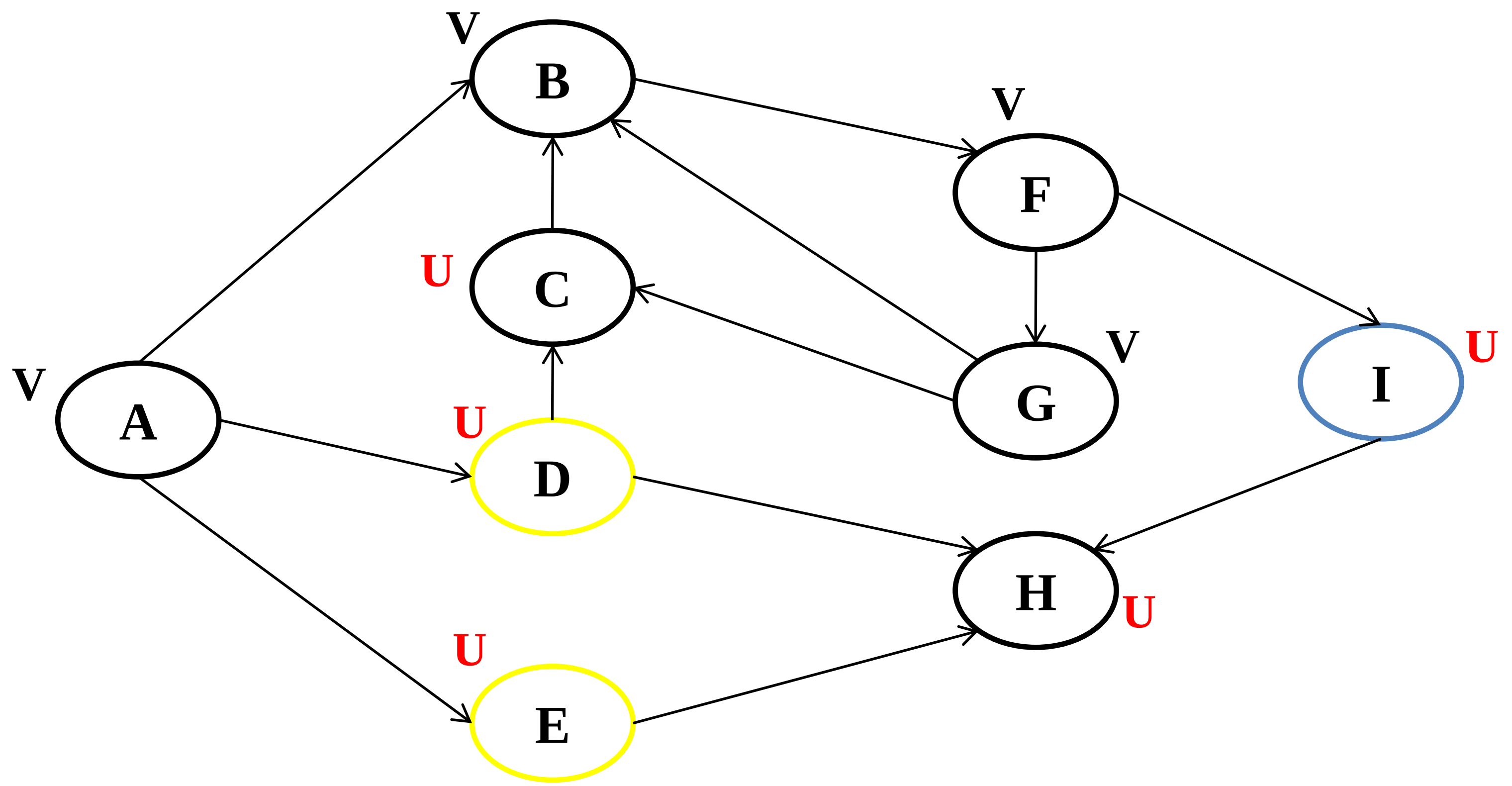
DFS Traversal: A B F

Depth First Search(DFS) Example



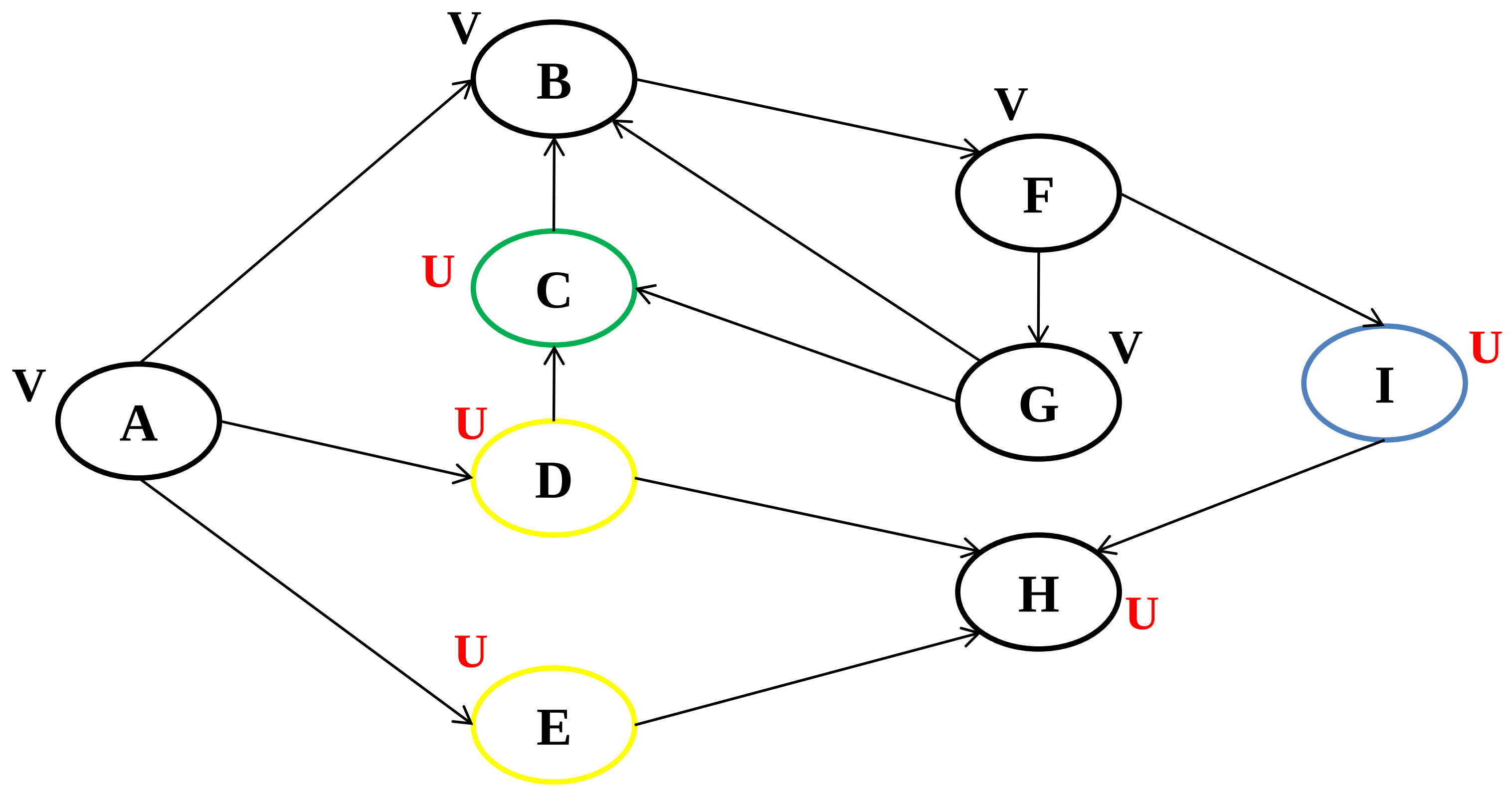
DFS Traversal: A B F

Depth First Search(DFS) Example



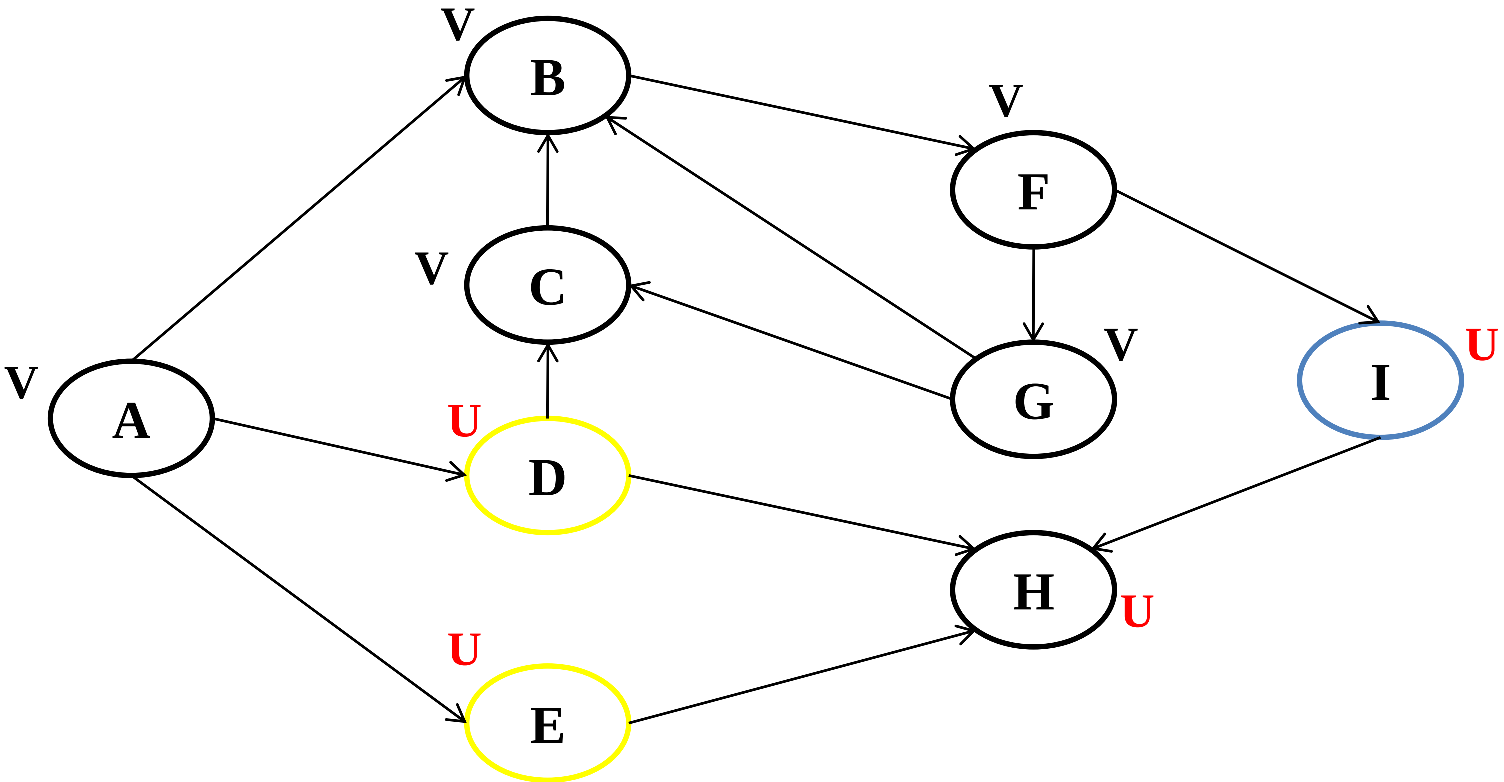
DFS Traversal: A B F G

Depth First Search(DFS) Example



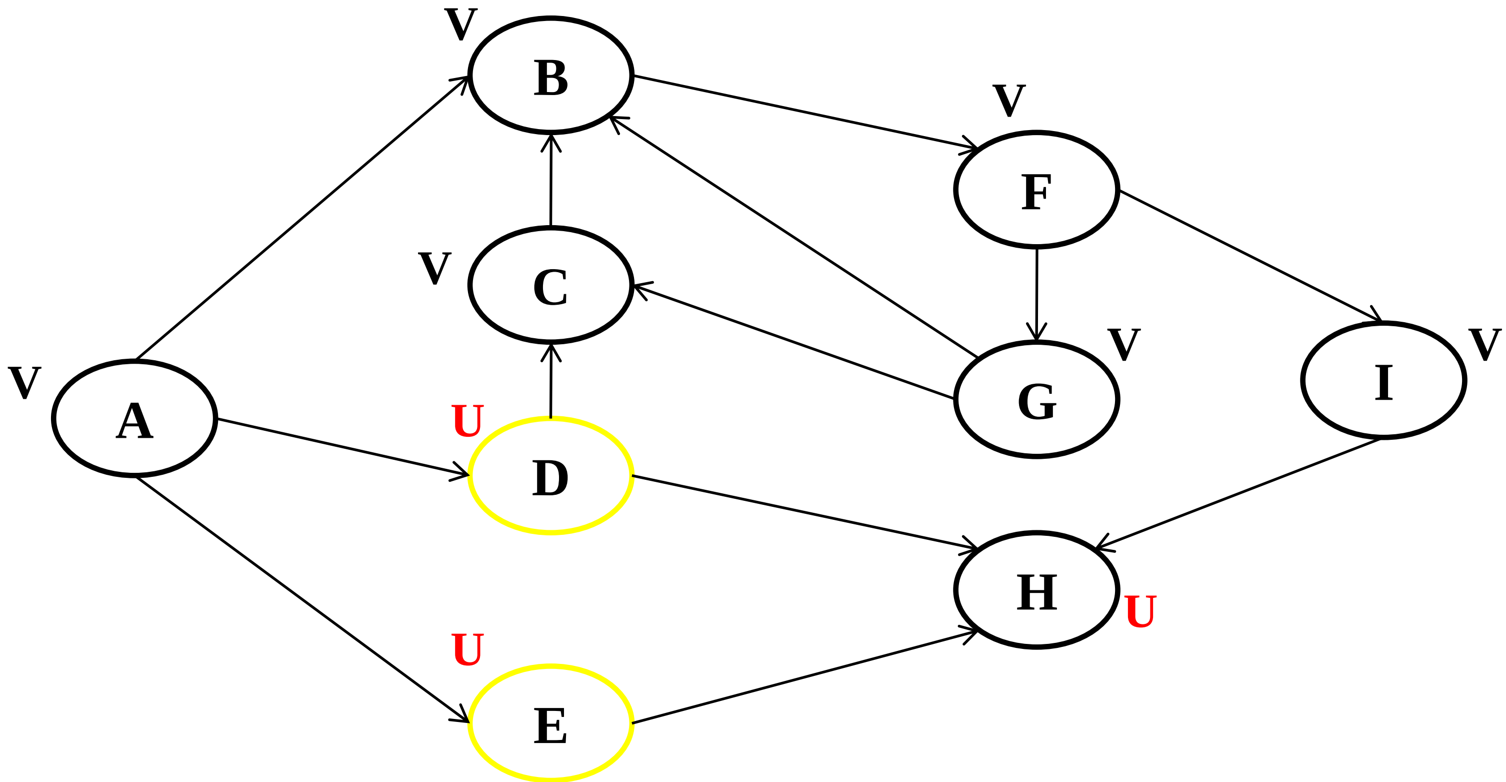
DFS Traversal: A B F G

Depth First Search(DFS) Example



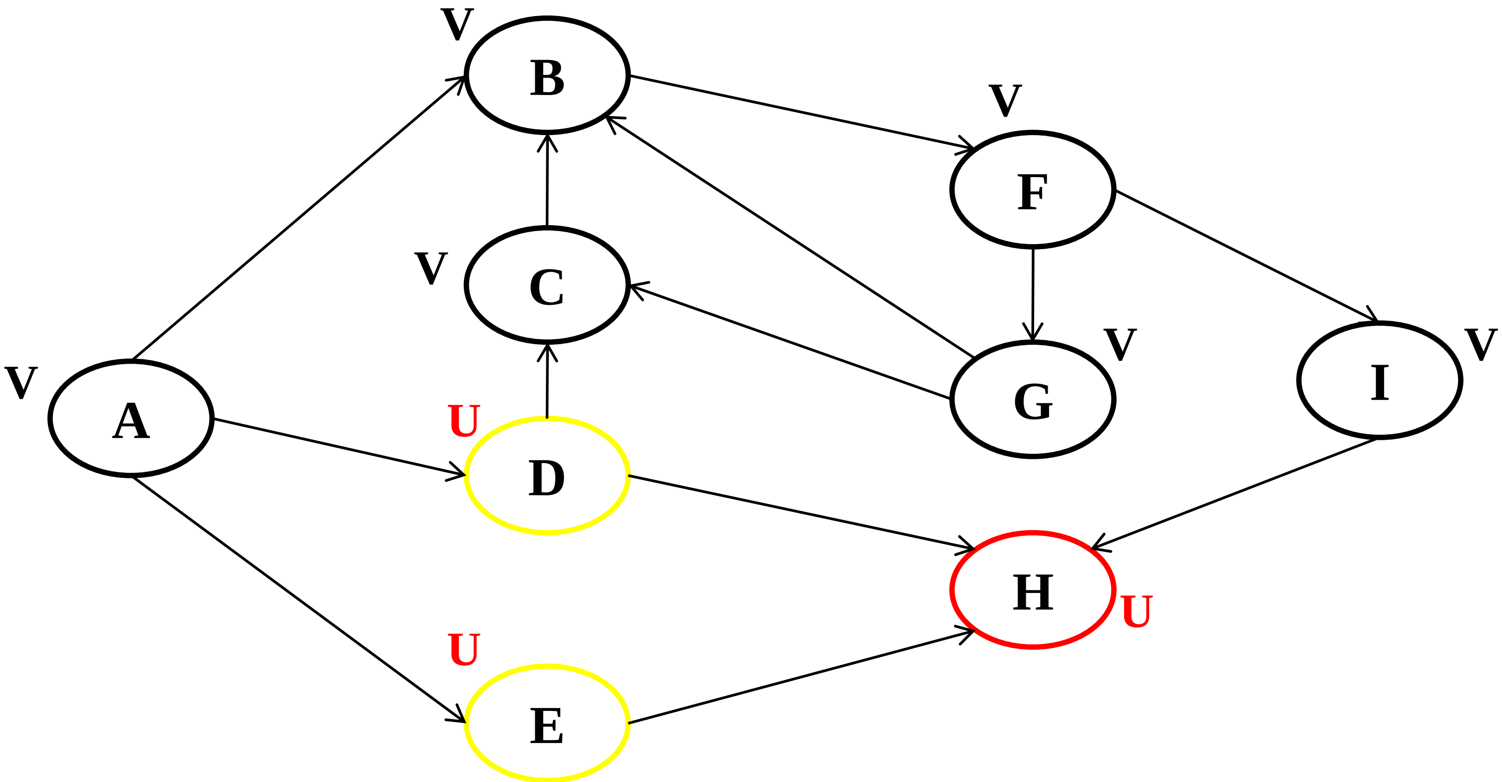
DFS Traversal: A B F G C

Depth First Search(DFS) Example



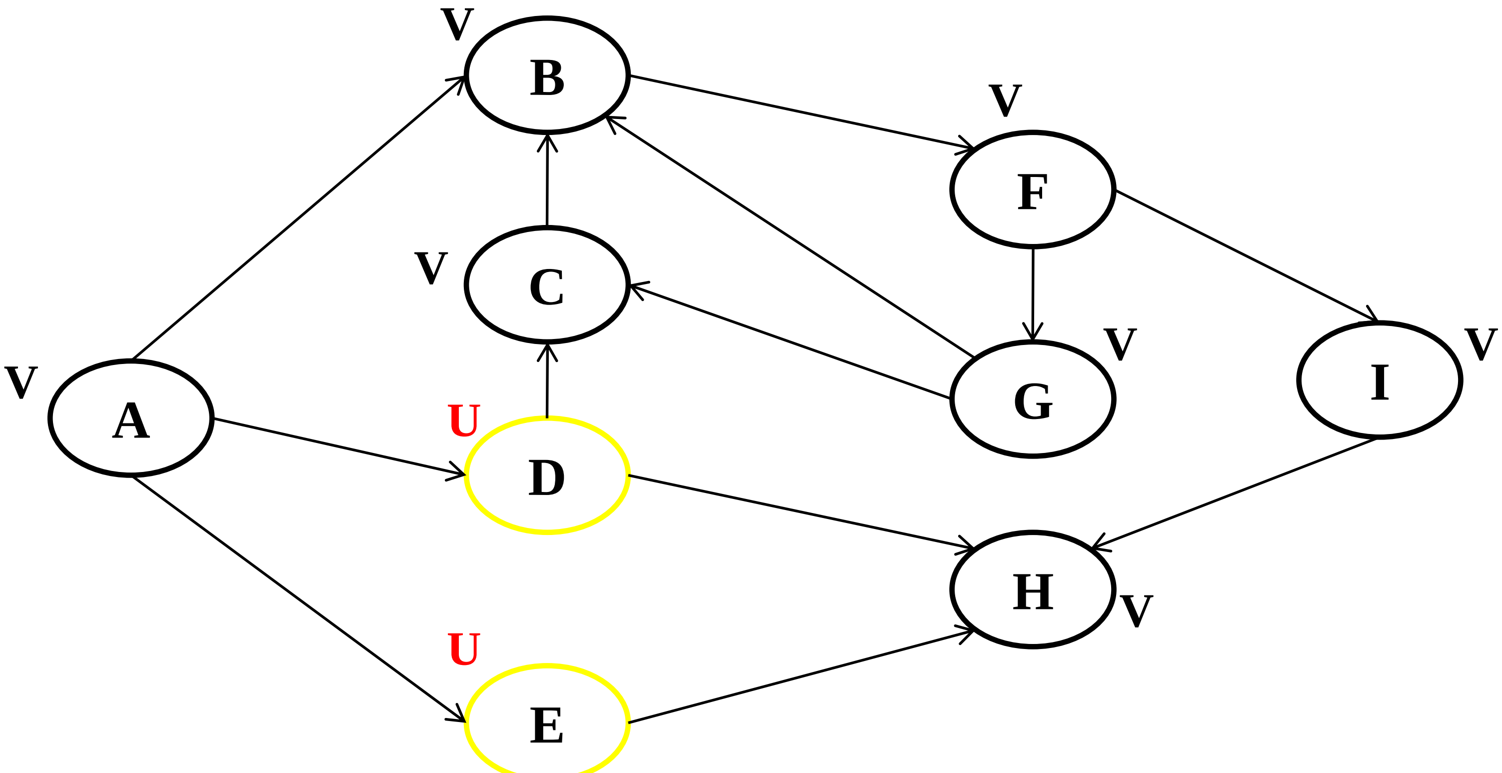
DFS Traversal: A B F G C I

Depth First Search(DFS) Example



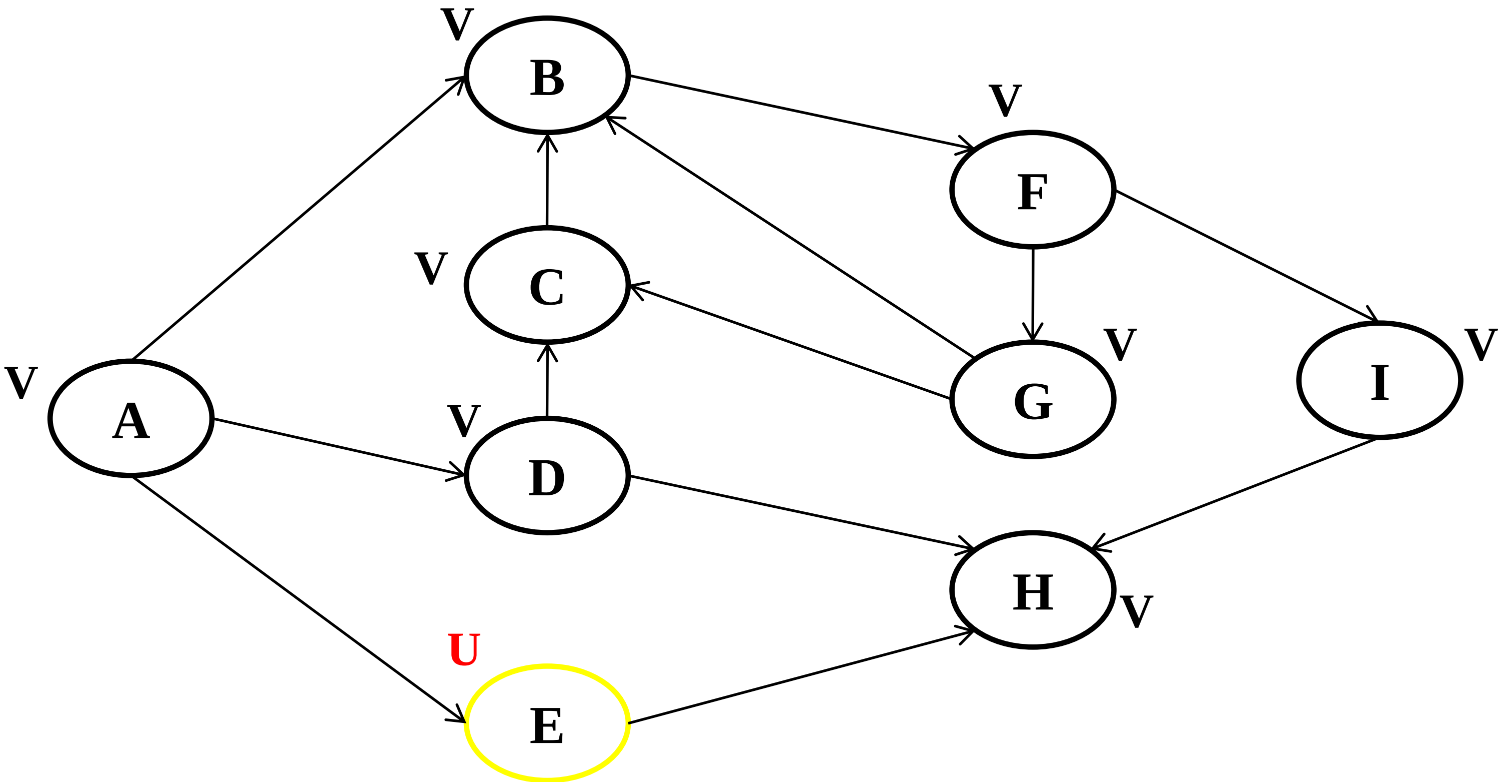
DFS Traversal: A B F G C I

Depth First Search(DFS) Example



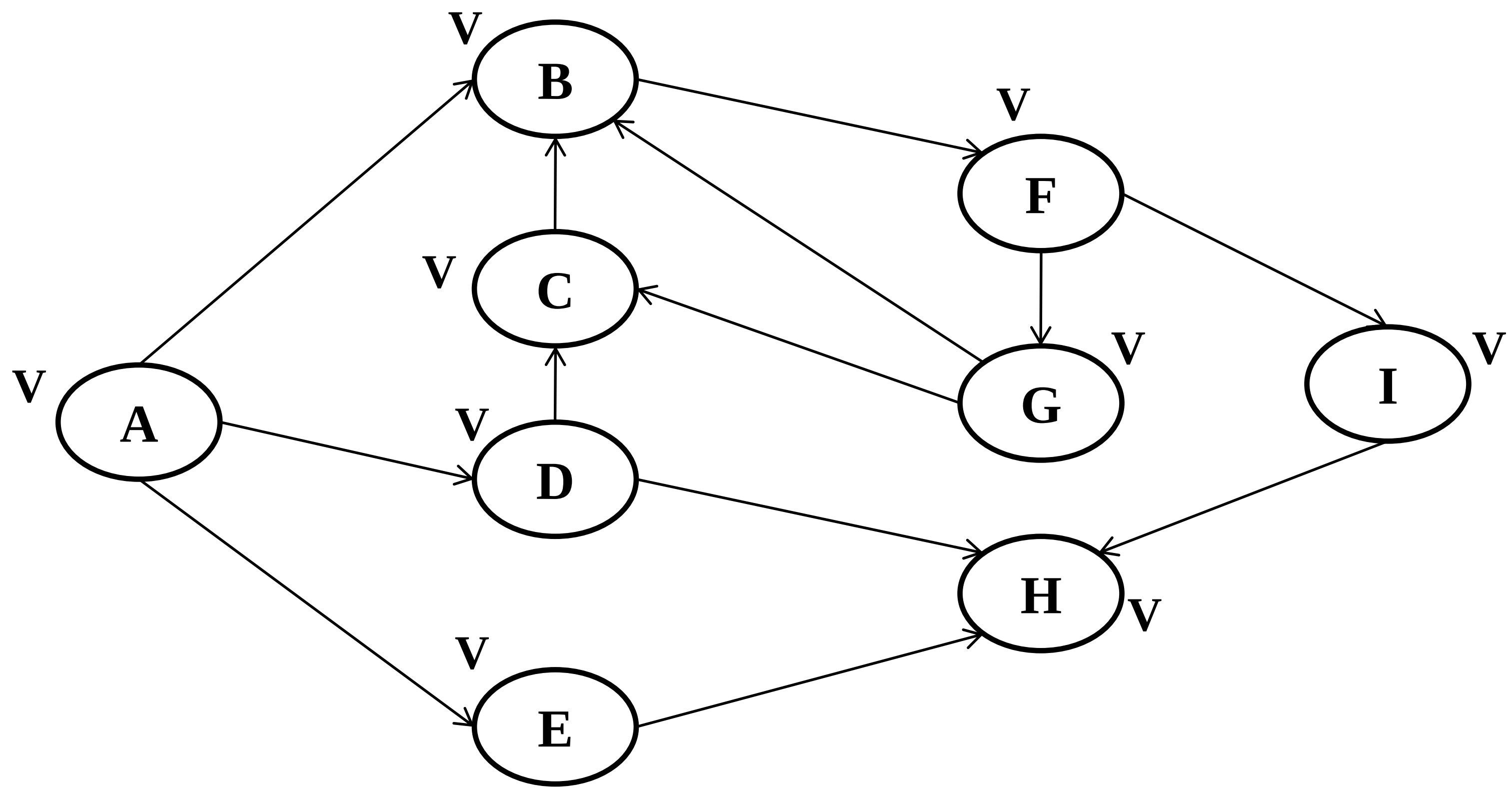
DFS Traversal: A B F G C I H

Depth First Search(DFS) Example



DFS Traversal: A B F G C I H D

Depth First Search(DFS) Example



DFS Traversal: A B F G C I H D E

DFS Algorithm Complexity

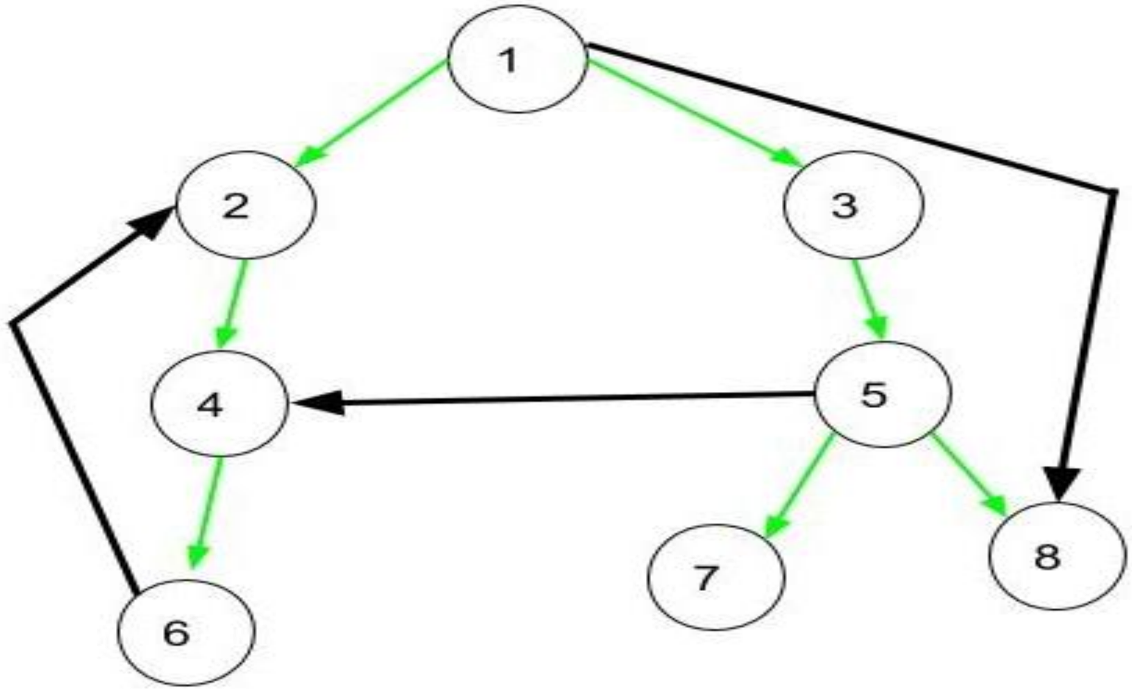
- If the graph is represented as an **adjacency list**
  - Each vertex is visited atmost once. So the time devoted is **O(V)**
  - Each adjacency list is scanned atmost once. So the time devoted is **O(E)**
  - Time complexity = **O(V+ E).**

- If the graph is represented as an **adjacency matrix**
  - There are $|V|^2$ entries in the adjacency matrix. Each entry is checked once.
  - Time complexity = **O(V^2)**

Applications of DFS

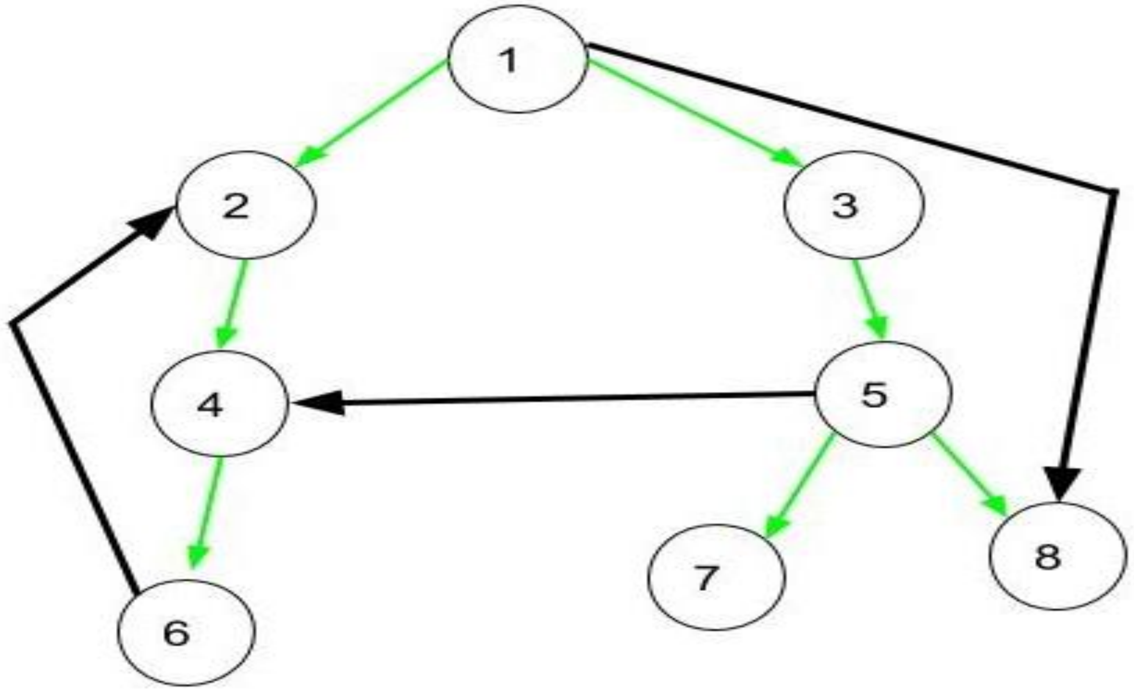
- Finding connected components in a graph
- Topological sorting in a DAG
- Scheduling problems
- Cycle detection in graphs
- Finding 2-(edge or vertex)-connected components
- Finding 3-(edge or vertex)-connected components
- Finding the bridges of a graph
- Finding strongly connected components
- Solving puzzles with only one solution, such as mazes
- Finding biconnectivity in graphs

Classification of Edges Based on DFS



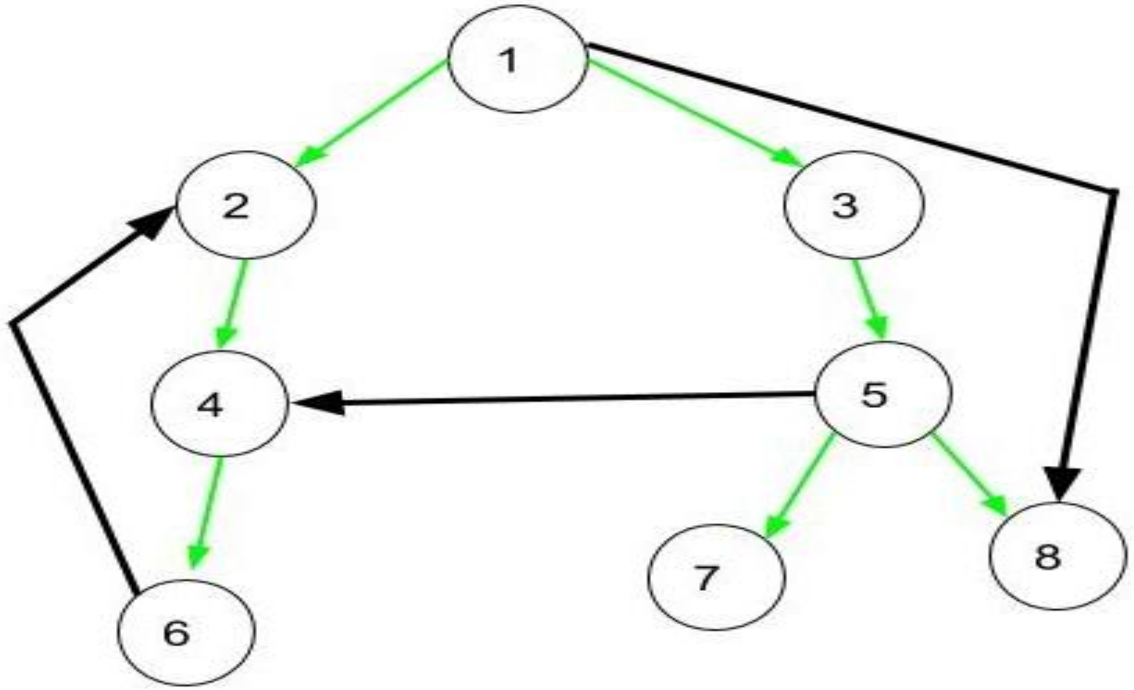
- The DFS traversal of the above graph is 1 2 4 6 3 5 7 8
- **Tree Edge:** It is a edge in tree obtained after applying DFS on the graph
  - Eg: (1,2), (2,4), (4,6), (1,3), (3,5), (5,7) and (5,8)

Classification of Edges Based on DFS



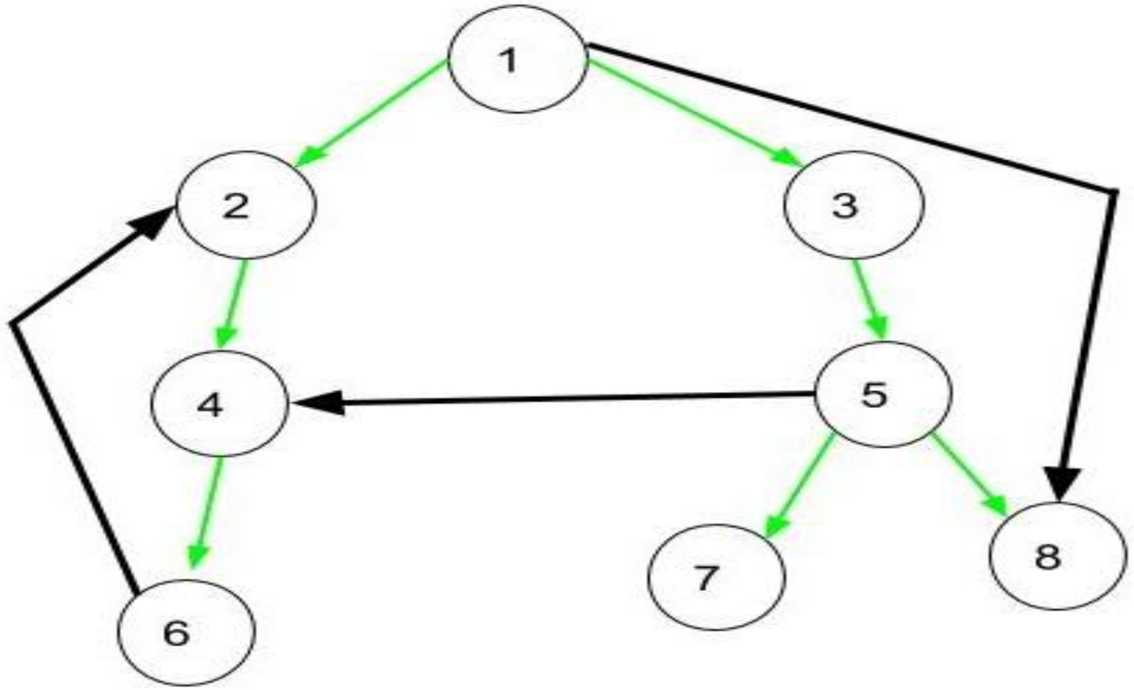
- The DFS traversal of the above graph is 1 2 4 6 3 5 7 8
- **Forward Edge:** It is an edge (u, v) such that v is descendant but not part of the DFS tree
  - Eg: (1,8)

Classification of Edges Based on DFS



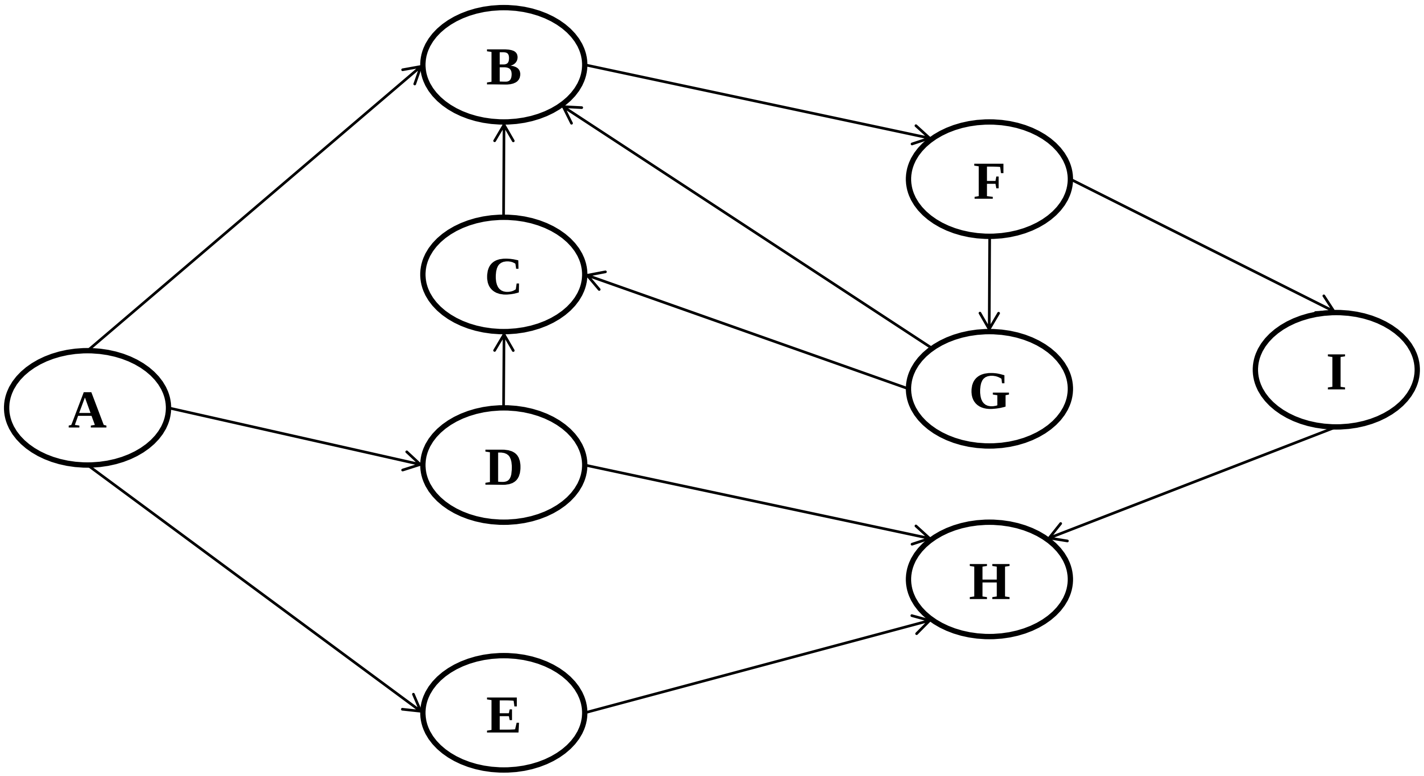
- The DFS traversal of the above graph is 1 2 4 6 3 5 7 8
- **Backward Edge:** It is an edge (u, v) such that v is ancestor of edge u but not part of DFS tree
  - Eg: (6,2)

Classification of Edges Based on DFS

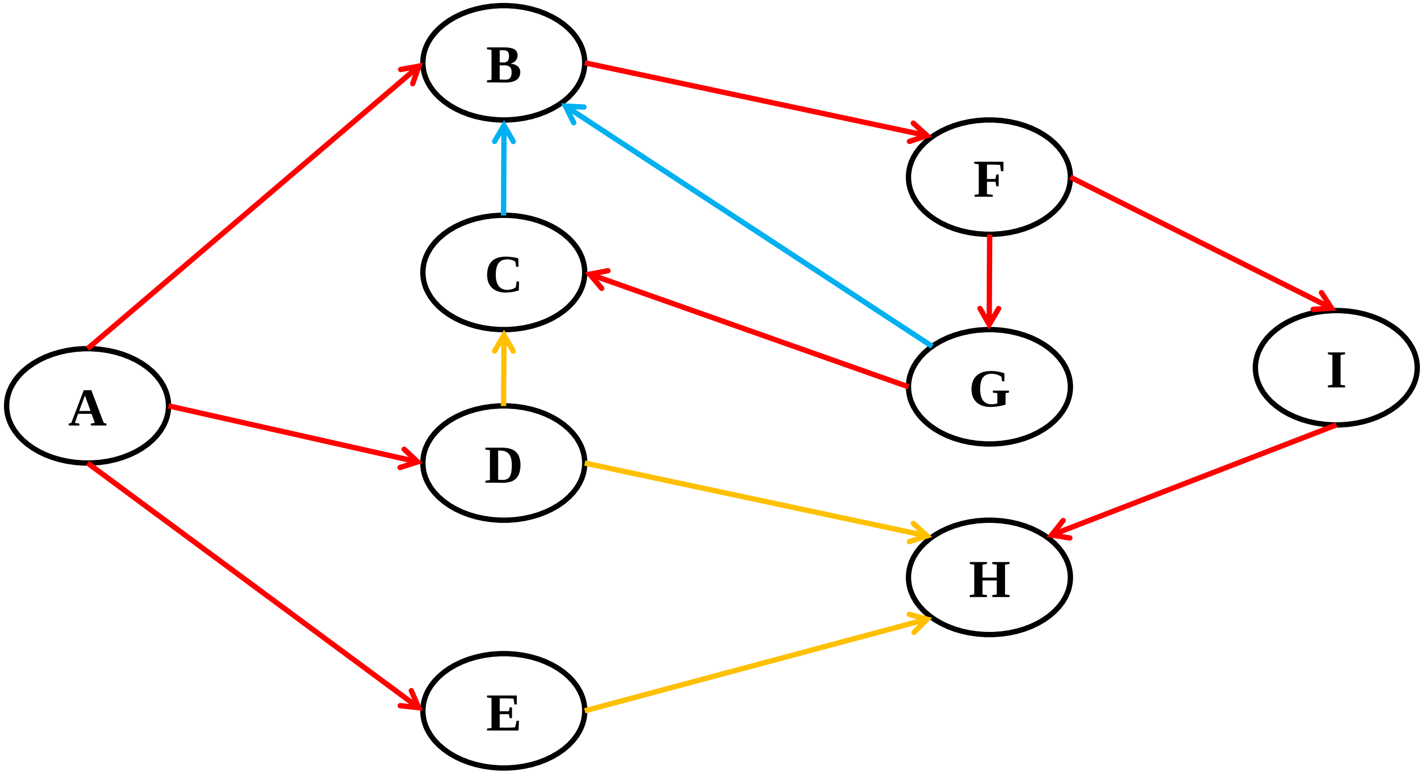


- The DFS traversal of the above graph is 1 2 4 6 3 5 7 8
- **Cross Edge:** It is a edge which connects two node such that they do not have any ancestor and a descendant relationship between them.
  - Eg: (5,4)

Classify the edges of the given graph



Classify the edges of the given graph



DFS Traversal : A B F G C I H D E

Tree Edge :(A,B),(B,F),(F,G),(G,C),(F,I),(I,H),(A,D),(A,E)

Forward Edge : --

Backward Edge :(G,B),(C,B)

Cross Edge :(D,C),(D,H),(E,H)