# Depth First Search Algorithm

*Presented by Nabanita Das*

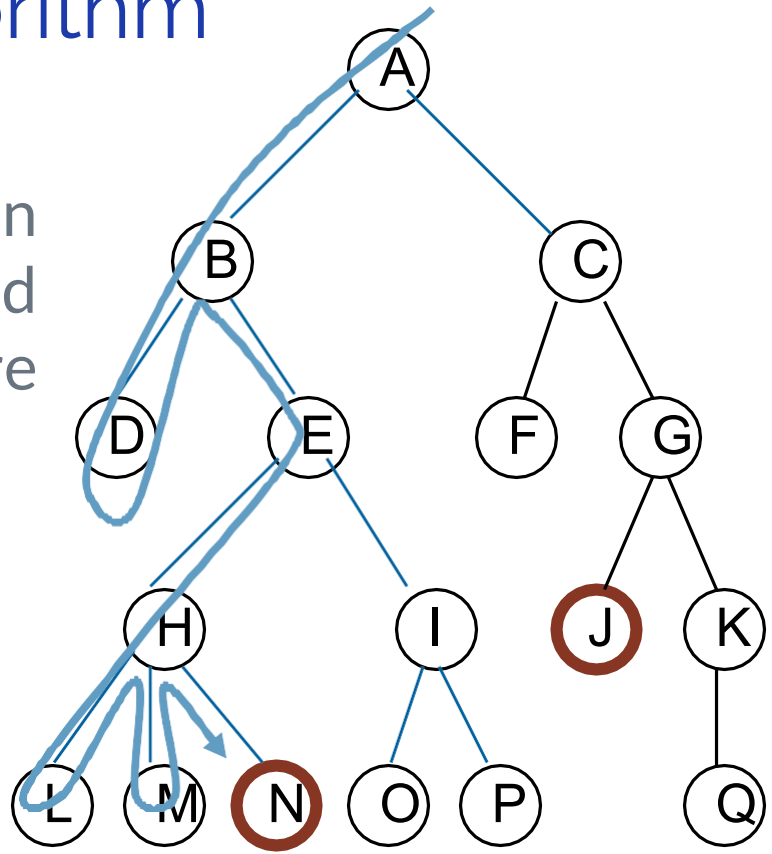
# Depth first search Algorithm

1. It is an algorithm for traversing tree or graph data structures. One starts at the root and explores as deep as possible along each branch before backtracking. It can be implemented using stack.
2. In Depth first search edges are explored out of the most recently discovered vertex. Only edges to unexplored vertices are explored.
3. When all of vertices edges have been explored, the search “backtracks” to explore edges leaving the vertex from which vertex was discovered.
4. The process continues until we have discovered all the vertices that are reachable from the original source vertex.
5. If any undiscovered vertices remain, then one of them is selected as a new source vertex.
6. This process if repeated until all vertices are discovered.

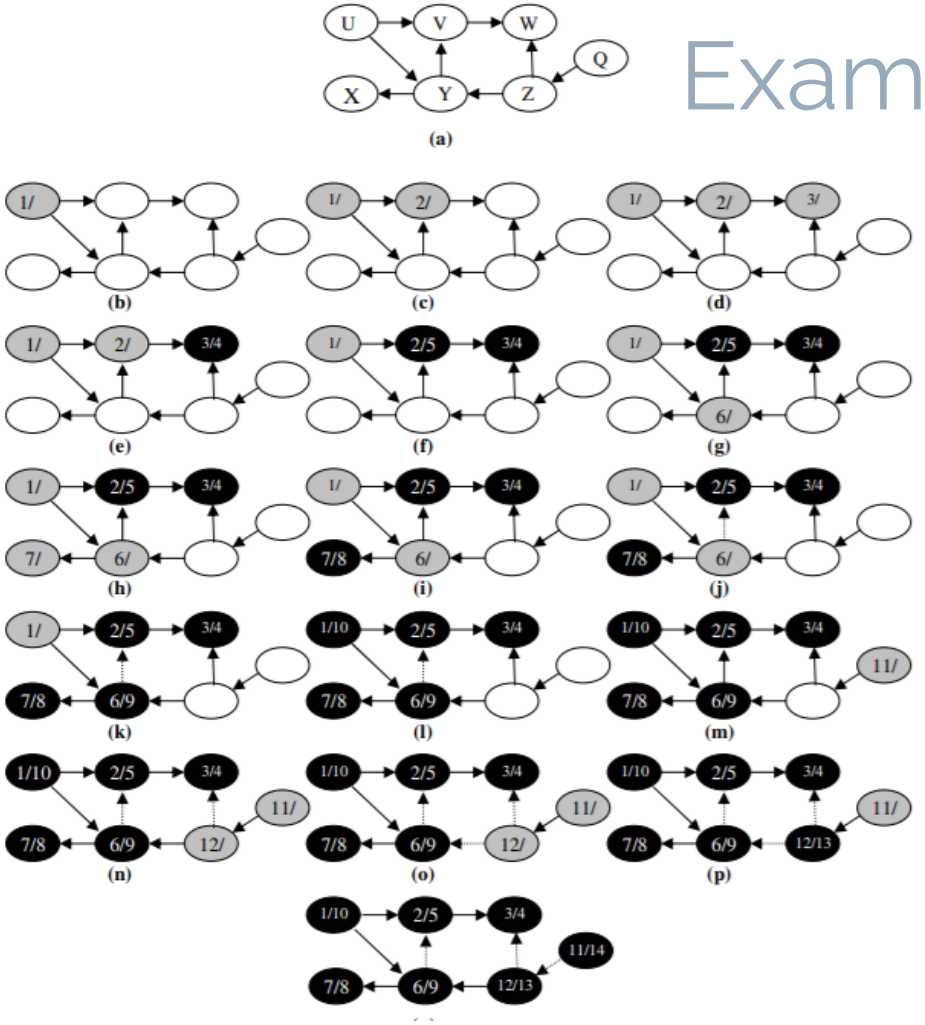
# Depth First Search Algorithm

For example, after searching A, then B, then D, the search backtracks and tries another path from B. Node are explored in the order -

**A B D E H L M N I O P C F G J K Q**



# Example of DFS Algorithm




For implementing DFS algorithm for the given graph we can start from node U and can go to discover node V or Y. Suppose that we discover node V, which has a single outgoing edge to node W. W has no outgoing edges, so this node is finished, and we return to V. From V there is no other choice, so this node is also finished and we return to U.

From node U we can continue to discover Y and its descendants, and the procedure continues similarly. At stage (l) we have discovered and finished nodes U, V, W, X, Y.

Then after selecting node Q as a new starting node, we can discover the remaining nodes (in this case Z)

# DFS Algorithm

```
DFS (V,E)

for each vertex u in V[G]
do color[v] ← WHITE
π[v] ← NIL
time ← 0
for each vertex v in V[G]
do if color[v] ← WHITE
then Depth_First_Search(v)
```

```
Depth_First_Search (v)
color[v] ← GRAY
time ← time + 1
d[v] ← time
for each vertex u adjacent to v
do if color[u] ← WHITE
        π[u] ← v
        Depth_First_Search(u)
color[v] ← BLACK
time ← time + 1
f[v] ← time
```

# Algorithm Explanation Step-01

**Step-01**

Create and maintain 4 variables for each vertex of the graph.

For any vertex ‘v’ of the graph, these 4 variables are-

**1. color[v]-**

This variable represents the color of the vertex ‘v’ at the given point of time.

- The possible values of this variable are- WHITE, GREY and BLACK.
- WHITE color of the vertex signifies that it has not been discovered yet.
- GREY color of the vertex signifies that it has been discovered and it is being processed.
- BLACK color of the vertex signifies that it has been completely processed.

**2. Π[v]-**

This variable represents the predecessor of vertex ‘v’.

**3. d[v]-**

This variable represents a timestamp when a vertex ‘v’ is discovered.

**4. f[v]-**

This variable represents a timestamp when the processing of vertex ‘v’ is completed.

# Step-02 and 03

**Step-02**

For each vertex of the graph, initialize the variables as-

- color[v] = WHITE
- π[v] = NIL
- time = 0 (Global Variable acting as a timer)

**Step-03**

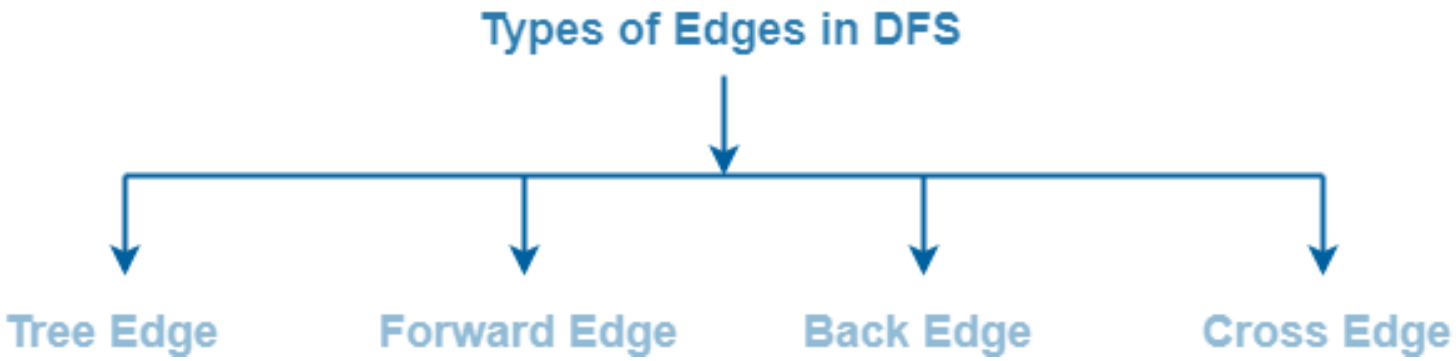
Repeat the following procedure until all the vertices of the graph become BLACK-

Consider any white vertex ‘v’ and call the following Depth_First_Search function on it.

Depth_First_Search (G,v)

1. color[v] = GRAY
2. time = time + 1
3. d[v] = time
4. For each adjacent WHITE vertex ‘u’ of ‘v’, set π[u] = v and call Depth_First_Search (G,u)
5. color[v] = BLACK
6. time = time + 1
7. f[v] = time

# Types of Edges in DFS-



### 1. Tree Edge-

- A tree edge is an edge that is included in the DFS tree.

### 2. Back Edge-

▷ An edge from a vertex ‘u’ to one of its ancestors ‘v’ is called as a back edge.

▷ A self-loop is considered as a back edge.

A back edge is discovered when-
- DFS tries to extend the visit from a vertex ‘u’ to vertex ‘v’
- And vertex ‘v’ is found to be an ancestor of vertex ‘u’ and grey at that time.

### 3. Forward Edge-

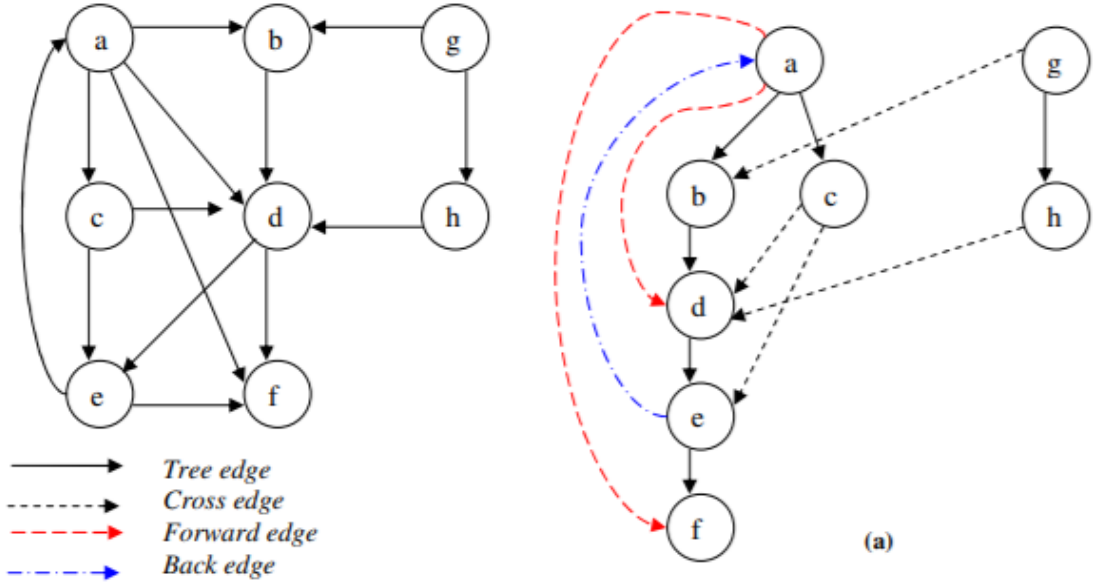
▷ An edge from a vertex ‘u’ to one of its descendants ‘v’ is called as a forward edge.

A forward edge is discovered when-
- DFS tries to extend the visit from a vertex ‘u’ to a vertex ‘v’
- And finds that color(v) = BLACK and d(v) > d(u).

### 4. Cross Edge-

▷ An edge from a vertex ‘u’ to a vertex ‘v’ that is neither its ancestor nor its descendant is called as a cross edge.

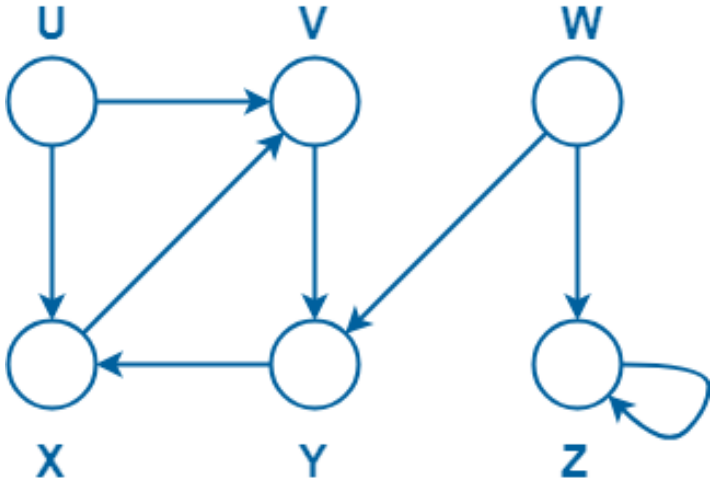
A cross edge is discovered when-
- DFS tries to extend the visit from a vertex ‘u’ to a vertex ‘v’
- And finds that color(v) = BLACK and d(v) < d(u).




Edge classification with DFS. In the case of scenario (a), the discovery sequence is a, b, d, e, f, c, g, h. When we reach node d from a, we characterize edge (a.d) as a forward edge because d is already discovered (from node b). The same applies for node f. When examining the outgoing edges of e we characterize edge (e,a) as a back edge, since a is already discovered and is an ancestor of e. Edges (c,d), (c,e) are cross edges because d and e are already discovered and have no ancestor-descendant relation with c.

# PROBLEM BASED ON DEPTH FIRST SEARCH

- Compute the DFS tree for the graph given below-




- Also, show the discovery and finishing time for each vertex and classify the edges.

# Step-01

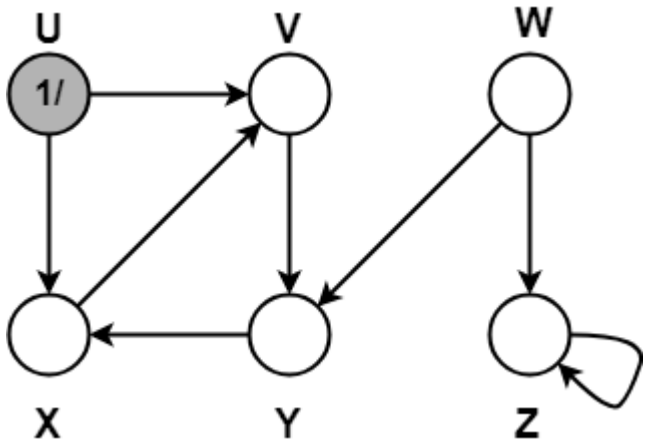
Initially for all the vertices of the graph, we set the variables as-

- color[v] = WHITE
- π[v] = NIL
- time = 0 (Global)

Let us start processing the graph from vertex U.

Step-01:

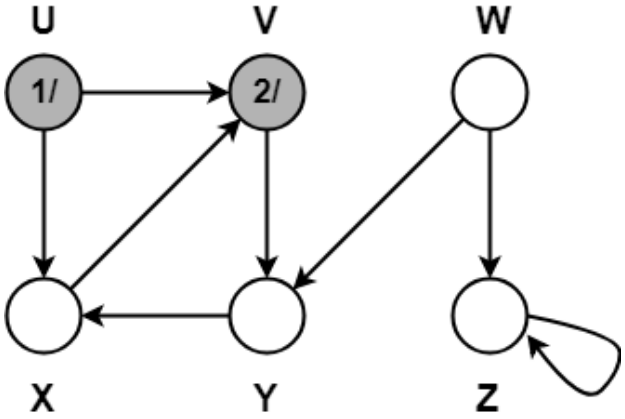
- color[U] = GREY
- time = 0 + 1 = 1
- d[U] = 1



# Step-02 and 03

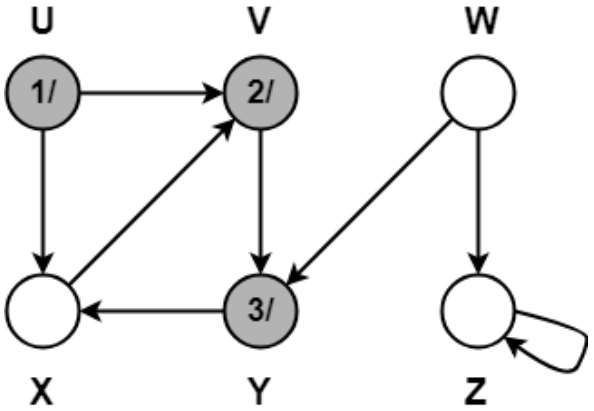
Step-03:

- π[V] = U
- color[V] = GREY
- time = 1 + 1 = 2
- d[V] = 2




Step-03:

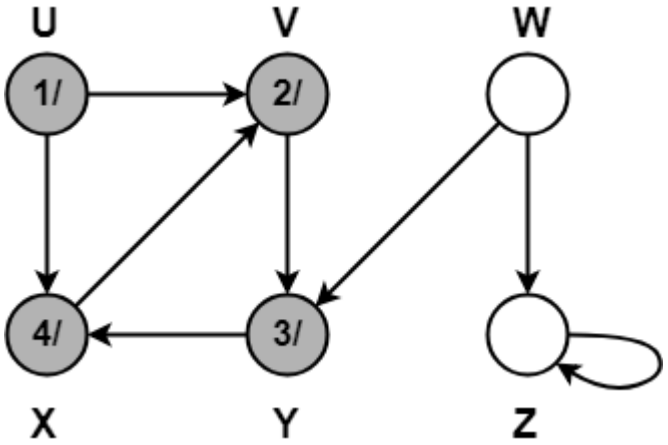
- π[Y] = V
- color[Y] = GREY
- time = 2 + 1 = 3
- d[Y] = 3



# Step-04

Step-04:

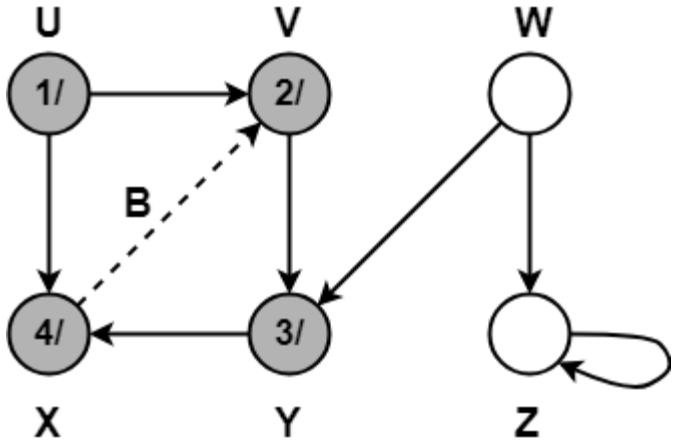
- π[X] = Y
- color[X] = GREY
- time = 3 + 1 = 4
- d[X] = 4



# Step-05

When DFS tries to extend the visit from vertex X to vertex V, it finds-

- Vertex V is an ancestor of vertex X since it has already been discovered
- Vertex V is GREY in color.
- ▷ Thus, edge XV is a back edge.



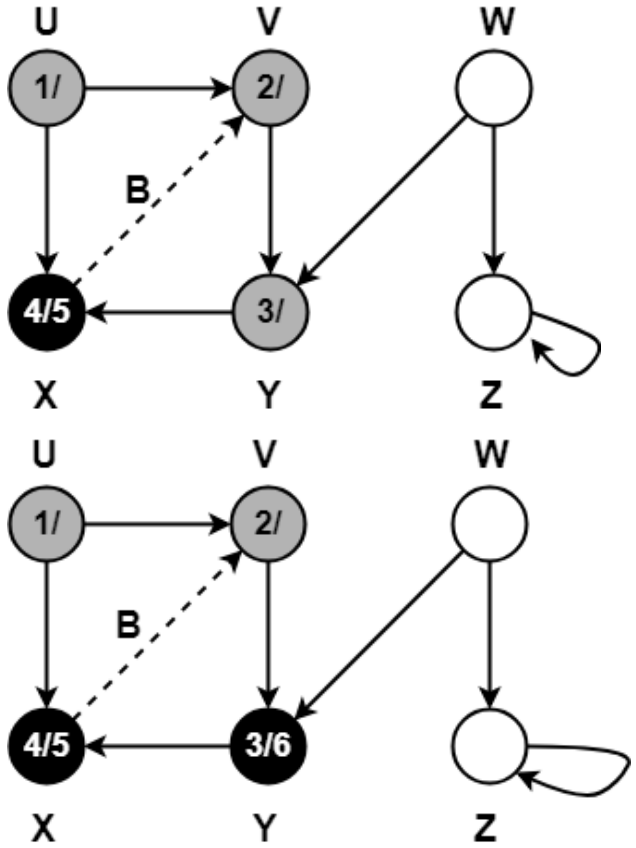
# Step- 06 and 07

Step-06:

- color[X] = BLACK
- time = 4 + 1 = 5
- f[X] = 5

Step-07:

- color[Y] = BLACK
- time = 5 + 1 = 6
- f[Y] = 6

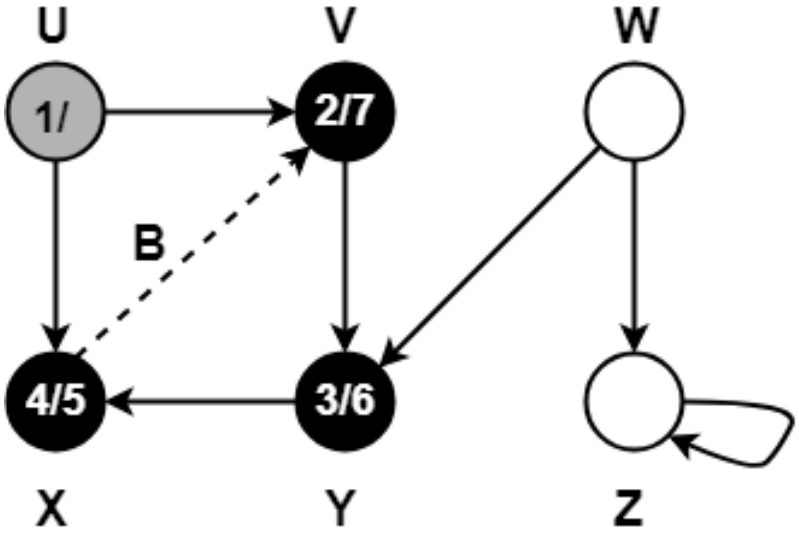


# Step-08

color[V] = BLACK

time = 6 + 1 = 7

f[V] = 7



# Step-09

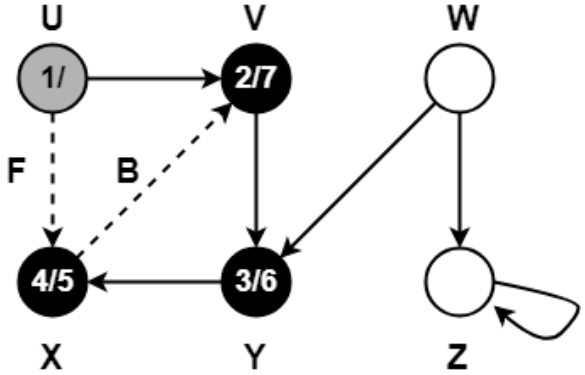
When DFS tries to extend the visit from vertex U to vertex X, it finds-

- Vertex X has already been completely processed i.e. vertex X has finished and is black.
- But vertex U has still not finished.

Alternatively, when DFS tries to extend the visit from vertex U to vertex X, it finds-

- Color(X) = BLACK
- d(X) > d(U)

Thus, edge UX is a forward edge.



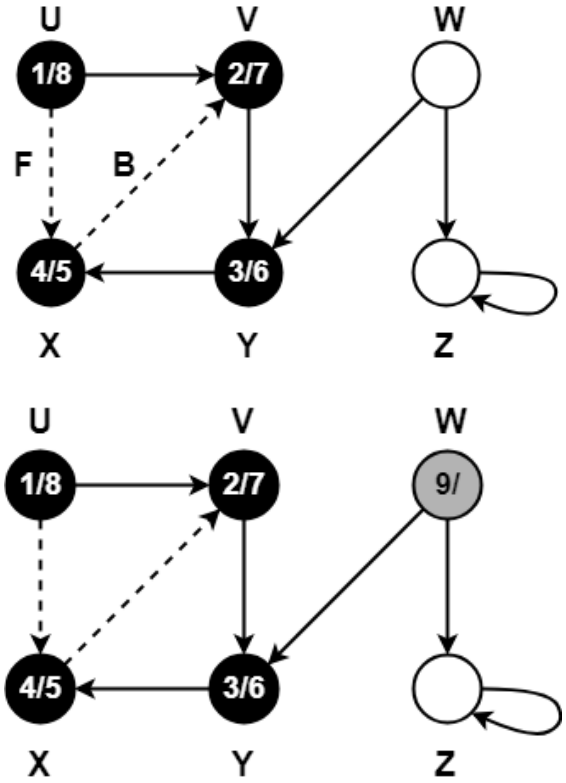
# Step-10 and Step-11

Step-10:

- color[U] = BLACK
- time = 7 + 1 = 8
- f[U] = 8

Step-11:

- color[W] = GREY
- time = 8 + 1 = 9
- d[W] = 9



# Step-12

When DFS tries to extend the visit from vertex W to vertex Y, it finds-

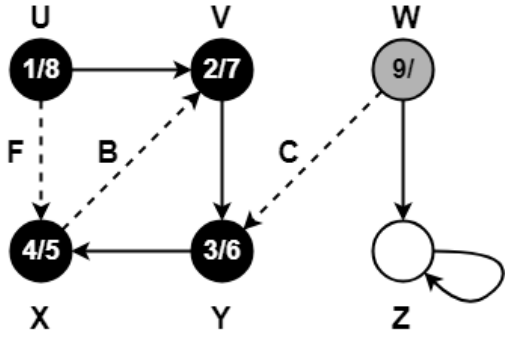
- Vertex Y has already been completely processed i.e. vertex Y has finished.
- Vertex Y is neither a descendant nor an ancestor of vertex W.

Alternatively,

When DFS tries to extend the visit from vertex W to vertex Y, it finds-

- Color(Y) = BLACK
- d(Y) < d(W)

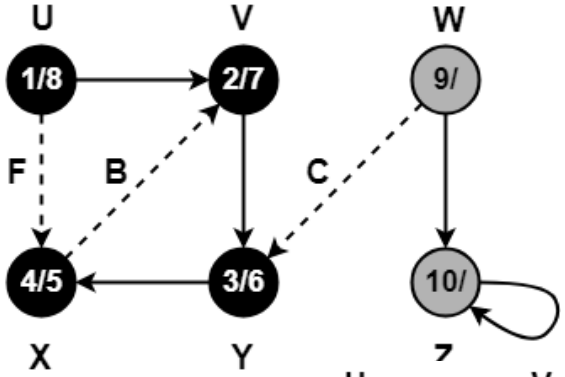
Thus, edge WY is a cross edge.



# Step-13 and 14

Step-13:

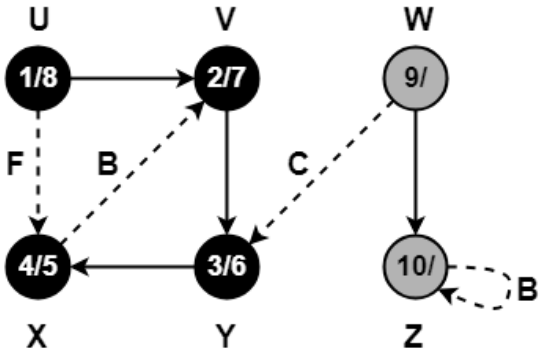
- π[Z] = W
- color[W] = GREY
- time = 9 + 1 = 10
- d[W] = 10



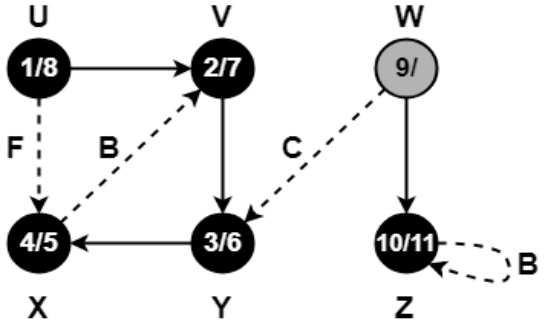

Step-14:

Since, self-loops are considered as back edges.

Therefore, self-loop present on vertex Z is considered as a back edge.

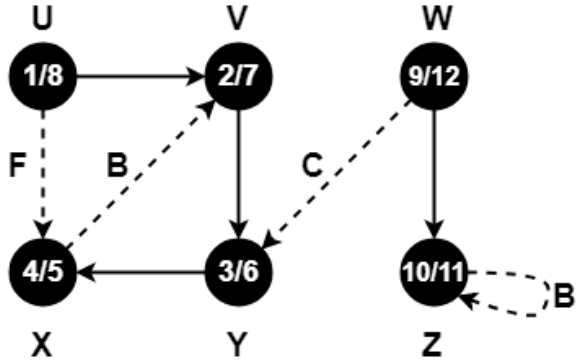


# Step- 15 and 16




Step-15:

- color[Z] = BLACK
- time = 10 + 1 = 11
- f[Z] = 11




Step-16:

- color[W] = BLACK
- time = 11 + 1 = 12
- f[W] = 12

Since all the vertices have turned black, so we stop.

This is how a given graph is traversed using Depth First Search (DFS) technique.

# Complexity Calculation

- The total running time for Depth First Search is θ (V+E).
- The time complexity of DFS if the entire tree is traversed is O(V) where V is the number of nodes.
- If the graph is represented as adjacency list:
  - Here, each node maintains a list of all its adjacent edges. Let's assume that there are V number of nodes and E number of edges in the graph.
  - For each node, we discover all its neighbors by traversing its adjacency list just once in linear time.
  - For a directed graph, the sum of the sizes of the adjacency lists of all the nodes is E. So, the time complexity in this case is O(V) + O(E) = O(V + E).
  - For an undirected graph, each edge appears twice. Once in the adjacency list of either end of the edge. The time complexity for this case will be O(V) + O (2E).
- If the graph is represented as an adjacency matrix (a V x V array):
  - For each node, we will have to traverse an entire row of length V in the matrix to discover all its outgoing edges.
  - Note that each row in an adjacency matrix corresponds to a node in the graph, and that row stores information about edges emerging from the node. Hence, the time complexity of DFS in this case is O(V * V) = O(V^2).