Depth first search

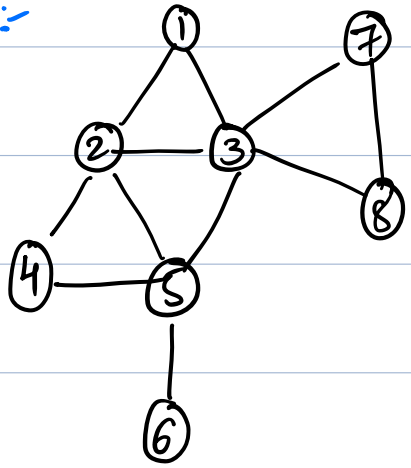
Idea:

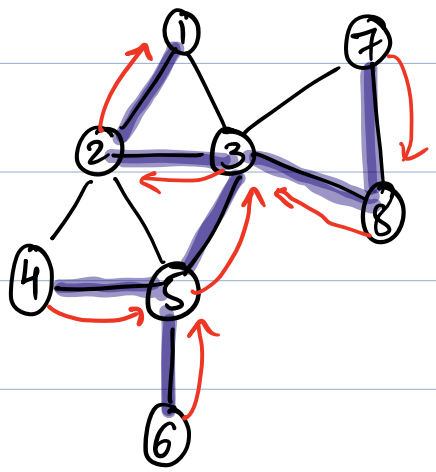
→ To search deeper in the graph whenever possible.

The Algorithm starts from a node s and try the first edge leading out of it to a node v. Then it follow the first edge leading out of v and so on until it reaches a "dead end" (a node for which all neighbors are explored).

Then it backtracks until it finds a node with unexplored neighbor and resumes from there.

Example:-





DFS(u)

mark u as "explored"

For each edge uv incident to u

if v is not marked "explored" then

Recursively invoke DFS(v)

End if

End for

For each node u we have

$u.color$ – color of u

$u.\pi$ – predecessor of u

$u.d$ – discovered time

$u.f$ – finishing time

DFS timestamps each vertex. Each vertex v has two timestamps.

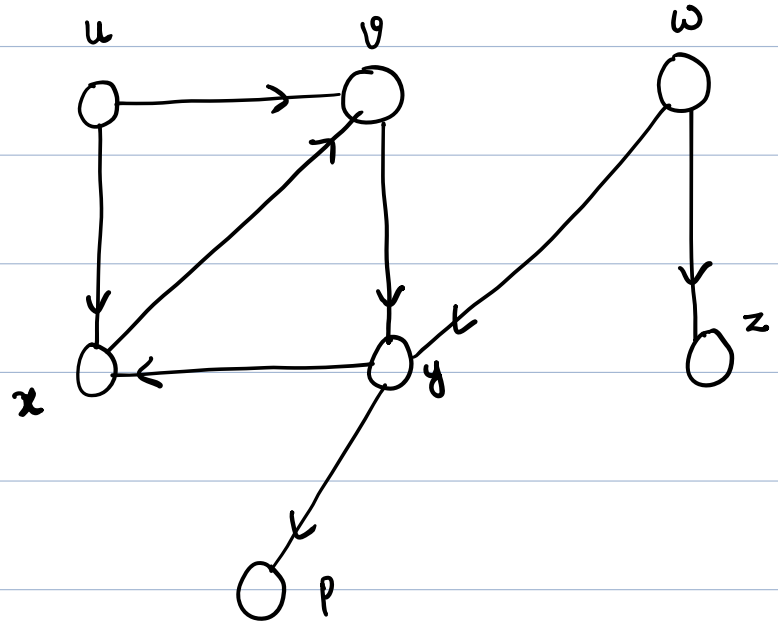
① The first timestamp $v.d$ records when v is first discovered (grayed)

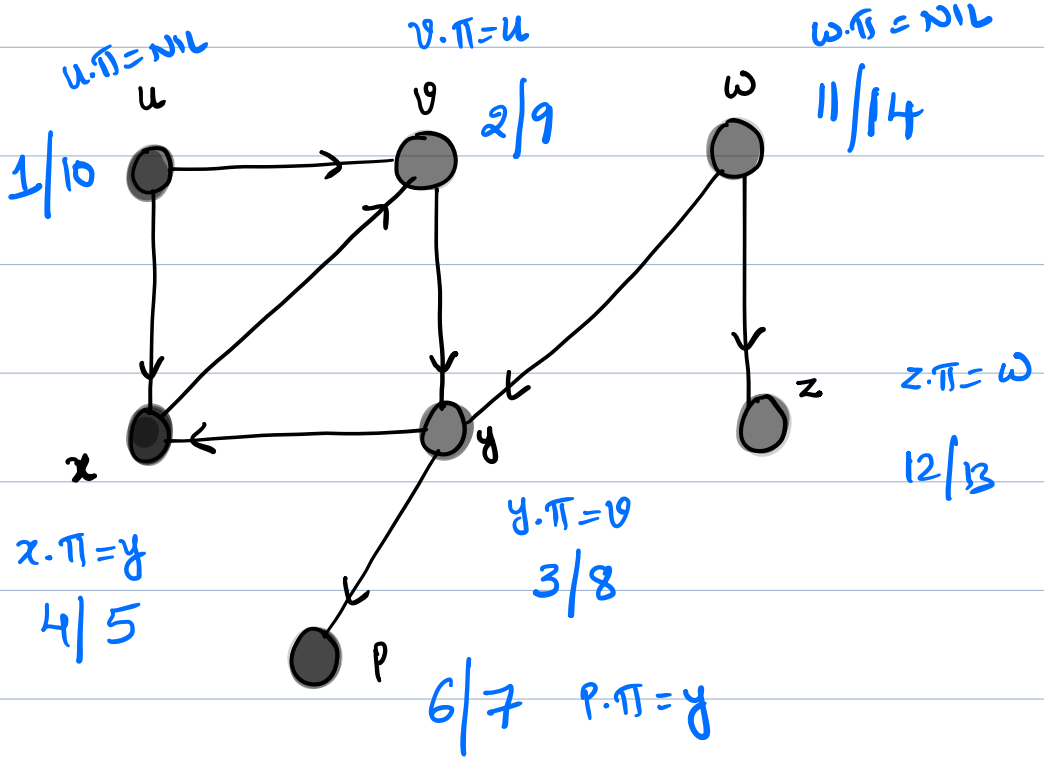
② The second timestamp $v.f$ record when the search finishes examining v's adjacency list.

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1          // white vertex u has just been discovered
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]    // explore edge (u, v)
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK          // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```

Example:


u
v
w
x
y
z
p


u.π = NIL
u
1/10
v.π = u
v
2/9
w.π = NIL
w
11/14
x
x.π = y
4/5
y
y.π = v
3/8
z
z.π = w
12/13
p
6/7
p.π = y

Depth first search tree / forest

$$G_\pi = (V, E_\pi)$$

where $E_\pi = \{ (v.\pi, v) \mid v \in V \text{ and } v.\pi \neq NIL \}$

Classification of edges

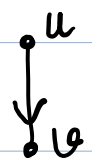
DFS can be used to classify the edges of the input graph.

The type of each edge gives some information about the structure of the graph.

Based on DFS forest G_{π} we can define four edge types.

1. ***Tree edges*** are edges in the depth-first forest G_π. Edge (u, v) is a tree edge if v was first discovered by exploring edge (u, v).
2. ***Back edges*** are those edges (u, v) connecting a vertex u to an ancestor v in a depth-first tree. We consider self-loops, which may occur in directed graphs, to be back edges.
3. ***Forward edges*** are those nontree edges (u, v) connecting a vertex u to a descendant v in a depth-first tree.
4. ***Cross edges*** are all other edges. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first trees.

When we first explore an edge uv the color of vertex v tells us something about the edge.

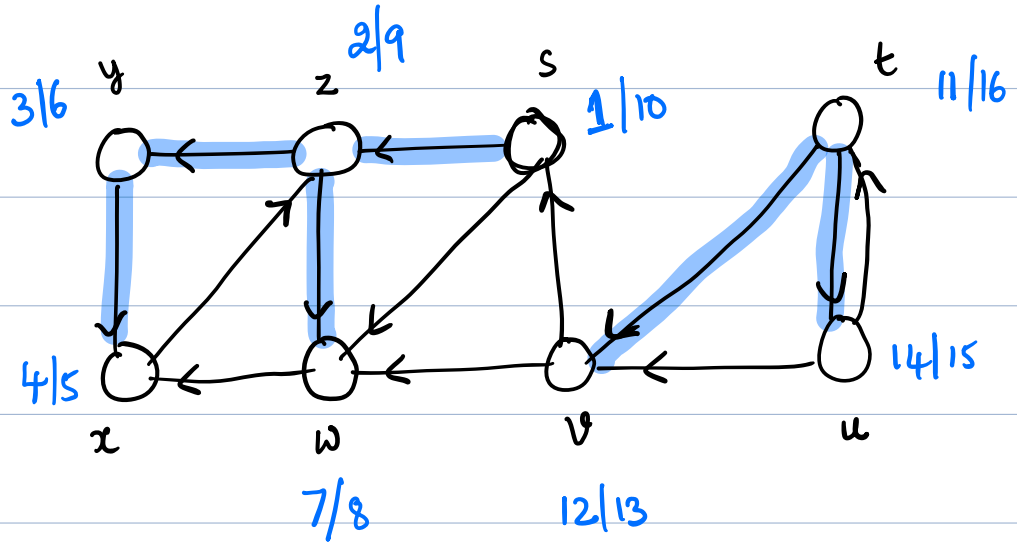


White indicates tree edge

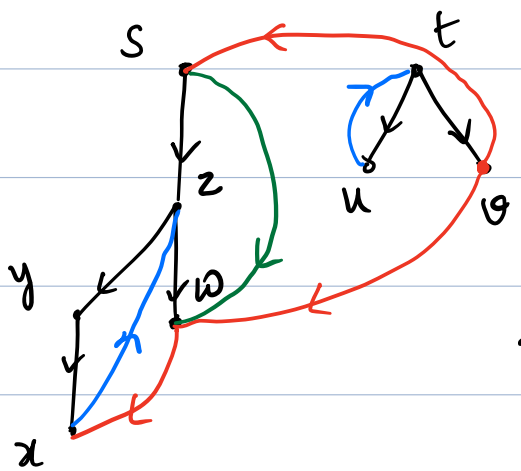
Gray " back edge

Black " forward / Cross edge.

Example:



DFS forest:



Tree edges - black colored

Back edges - blue "

Cross edges - red "

forward edges - green "

$(u,v) \in E(G)$

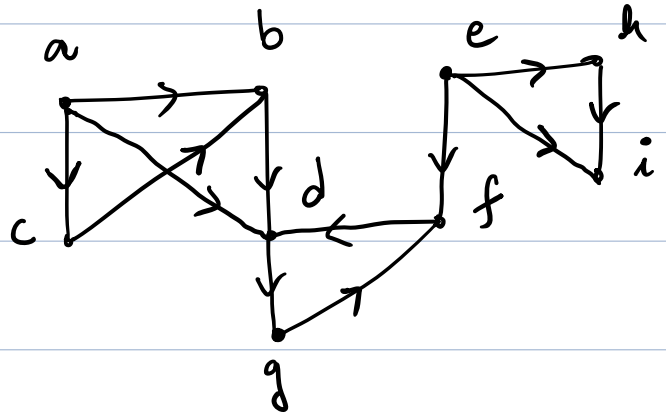
FORWARD EDGE – $u.d < v.d$ & $u.f > v.f$

Back EDGE – $u.d > v.d$ and $u.f < v.f$

Tree edge – $u.d < v.d$ & $u.f > v.f$

Cross edge – $v.d < v.f < u.d < u.f$

Exercise:



(A) Perform a depth first search starting from node 'a' with Preference for visiting lower-character vertices before higher-character vertices.

(B) Write down the discover and finish times for each node

(C) Also classify the edges.