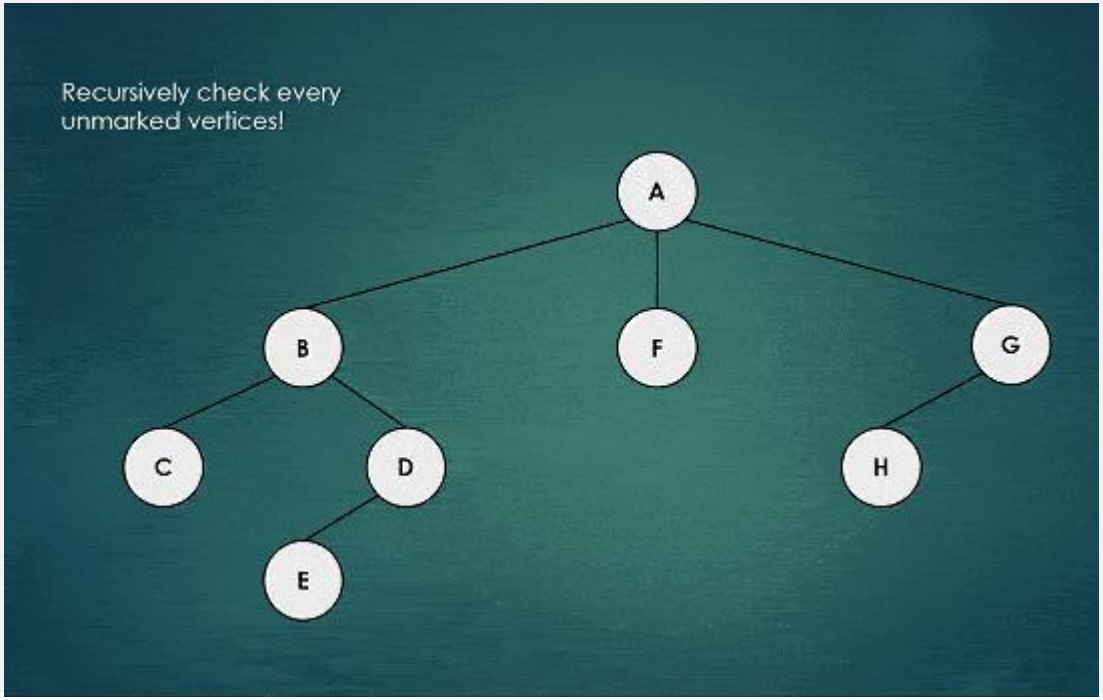
4.4 DFS คืออะไร

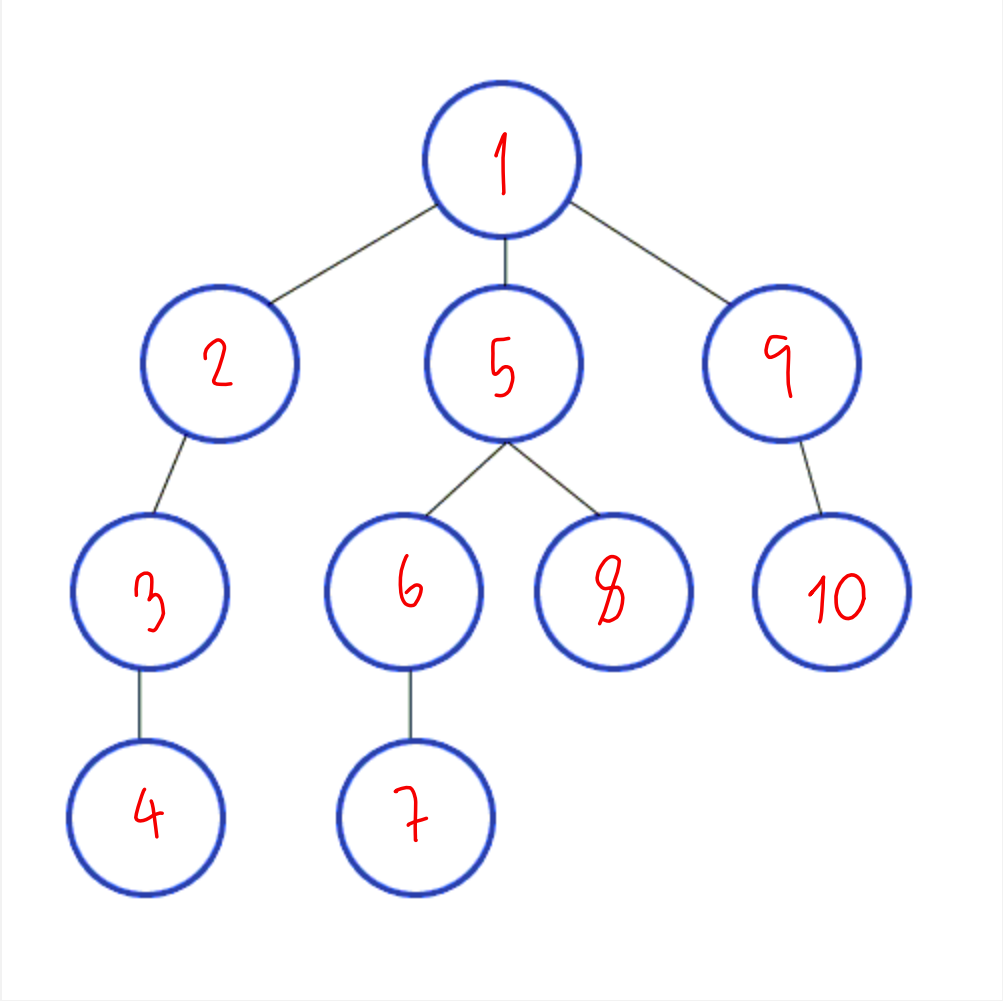
Search ตามความลึก , ใช้ vector ในการออกสอบ, เขียน

- The Depth First Search is fundamental search algorithm used to explore node and edges of a graph.
- It runs with a time O(V+E) Big O
- Often used as a building block in other algorithms.



วัตถุประสงค์ & ขั้นตอนการทำงาน

- สำรวจตามความลึกของกราฟ
- โดยหาเส้นทางที่ลึกที่สุดจากการเดินจาก เวอร์เท็กซ์ u ไปยัง เวอร์เท็กซ์ v ซึ่งเป็นเวอร์เท็กซ์ข้างเคียงที่ยังไม่ได้เดินผ่าน ทำเช่นนี้จนกระทั่งไม่มี edge ที่สามารถเดินต่อไปได้อีก จากนั้นจึงทำการเดินกลับ (backtrack) ไปยัง source
- A DFS plunges depth first into a graph without regards for which edge it takes next until it cannot go any further at which point it backtracks and continues.

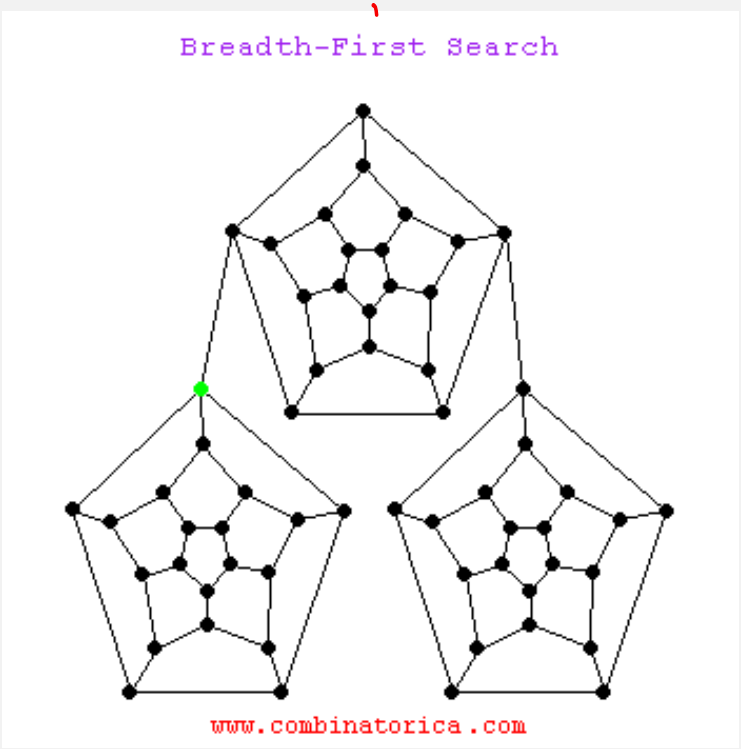


https://commons.wikimedia.org/wiki/File:Depth-First-Search.gif

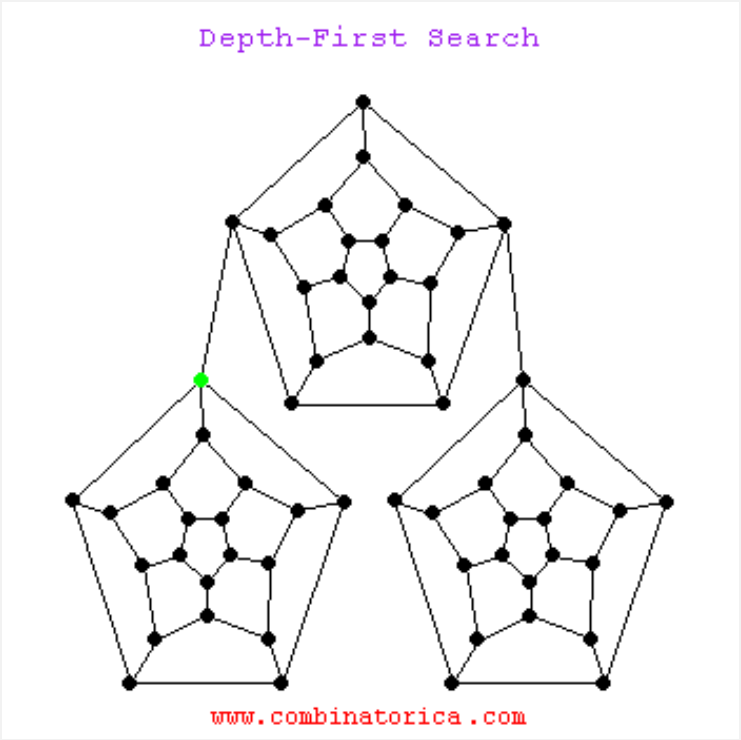
ขั้นตอน(ภาษาอังกฤษ)

1. Edges are explored out of the most recently discovered vertex v that still has unexplored edge leaving it.
2. When all of v's edges have been explored, the search backtracks to explore edges leaving the vertex from which v has discovered. Until we have discovered all the vertices that are reachable from the original source vertex

* เส้นทางสั้นสุด

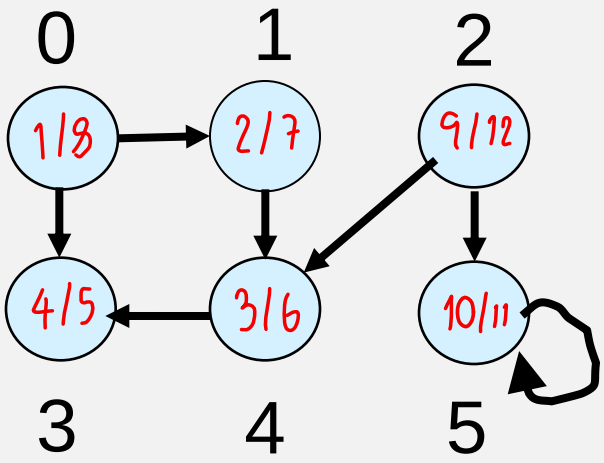


ไม่ก็วนซ้ำ



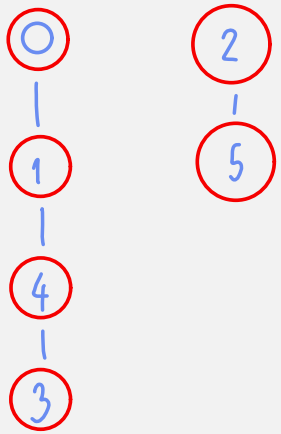
เดินหน้า / ถอยกลับ

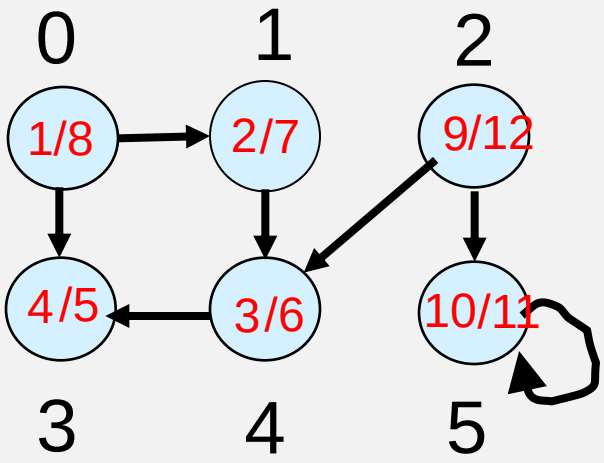
u / v



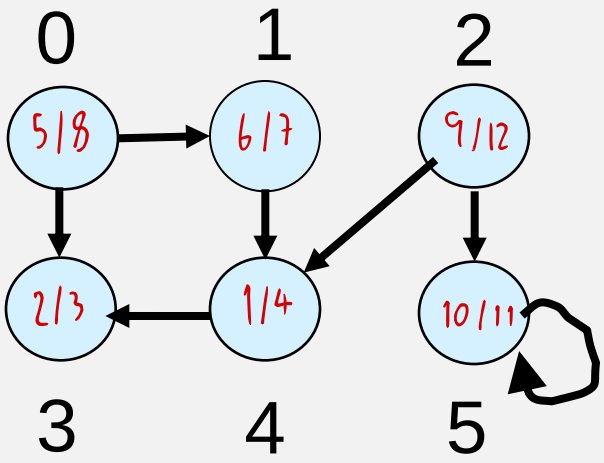
S= 0 * ต้นทาง

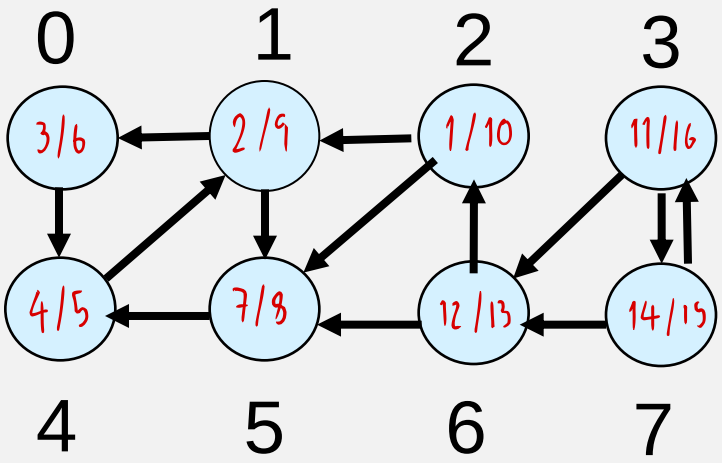
tree



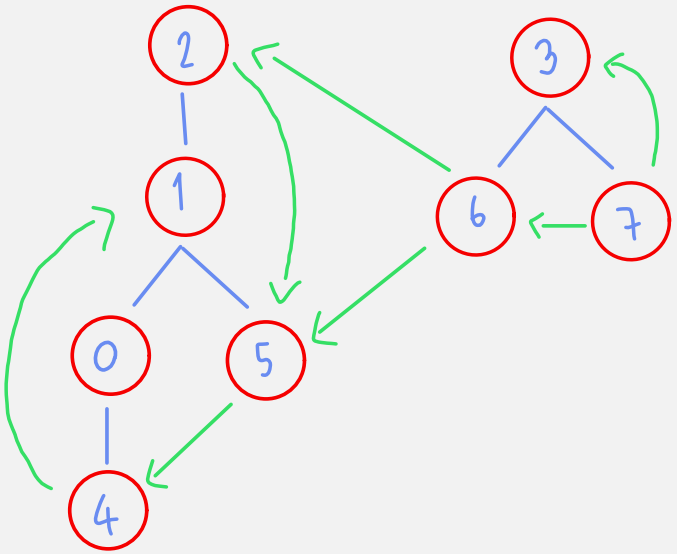

0
1
2
1/8
2/7
9/12
4/5
3/6
10/11
3
4
5

S= 4

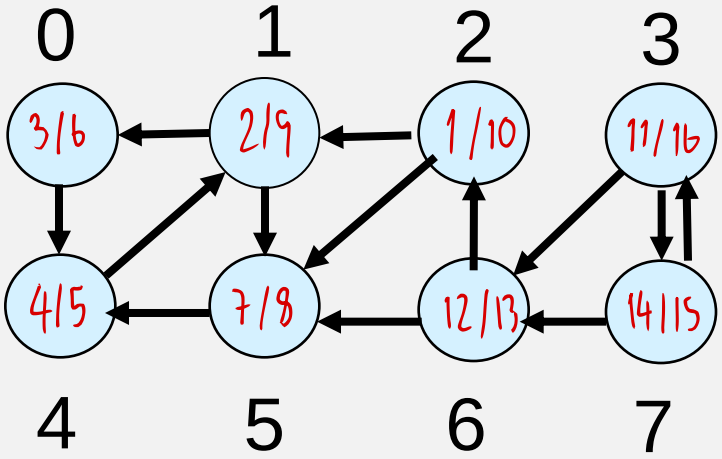




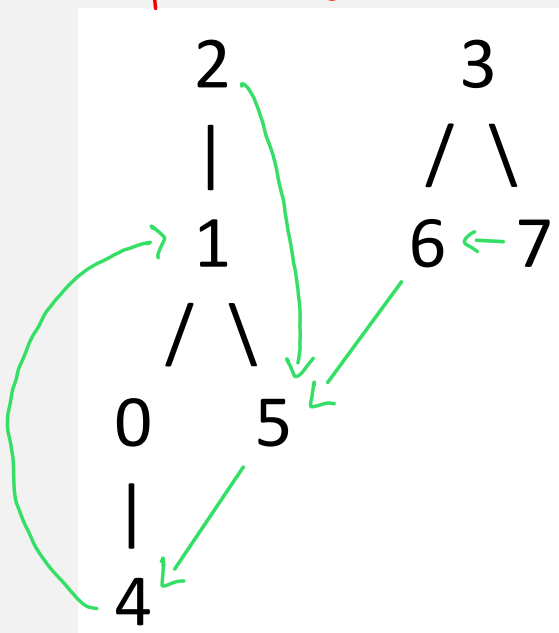
S= 2



DFS trees (u, v)



* ตัวอย่าง



DFS

* เท่านั้น

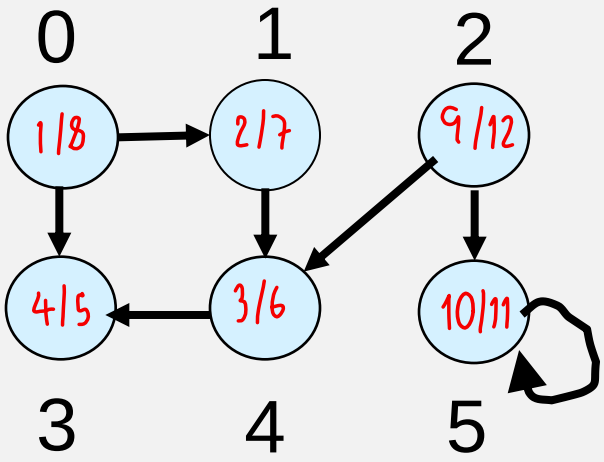
```
vector <int> adj(MAX);
int degree[MAX];

void readInput()
    int n, m;
    cout << "Enter v & e : ";
    cin >> n >> m;
    for(int i=0; i< m; i++){
       int u, v;
       cin >> u >> v;
       adj[u].push_back(v);
       degree[u]++; // หรือใช้ getCount();
    }
}
```

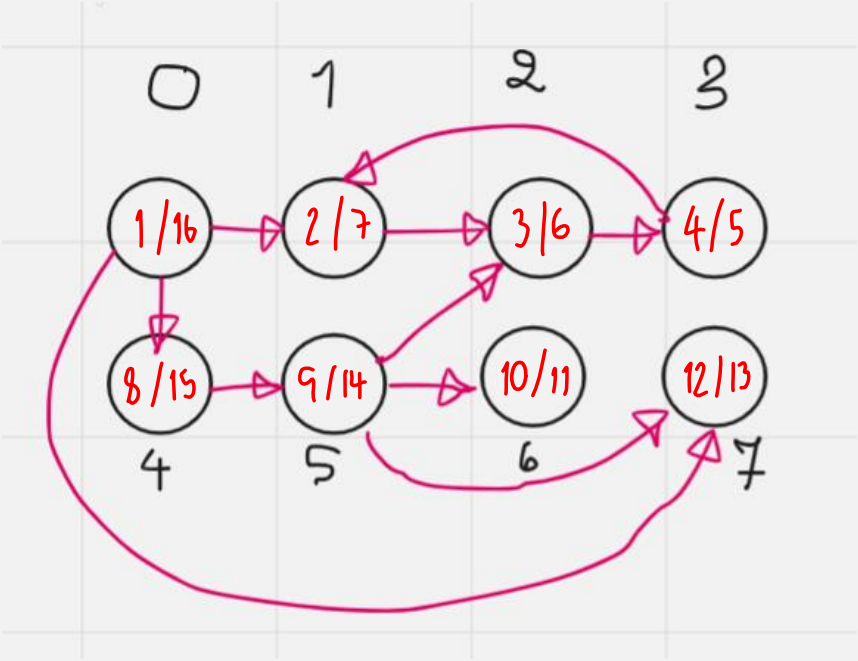
```
for(int i = 0; i < 3; i++){
   for(int j=0; j < degree[i]; j++){
      cout << adj[i][j] << " ";
```

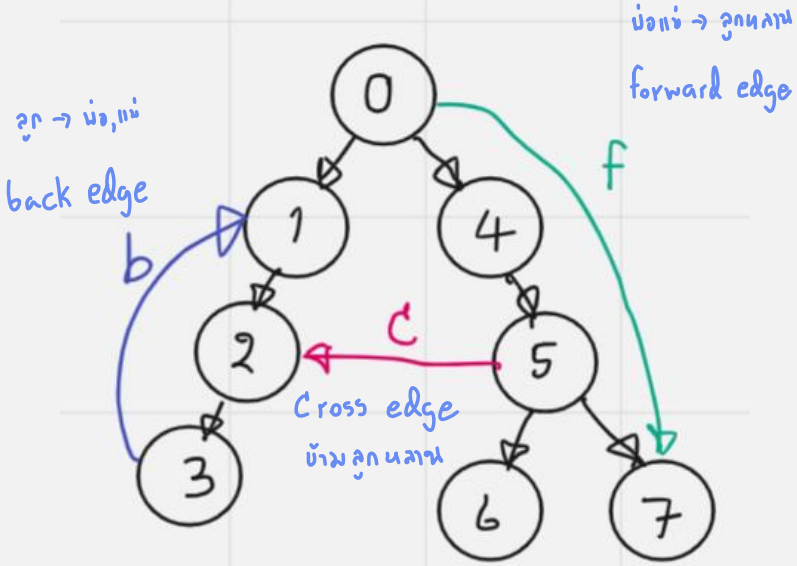
Edge Classification

S= 0



Edge Classification

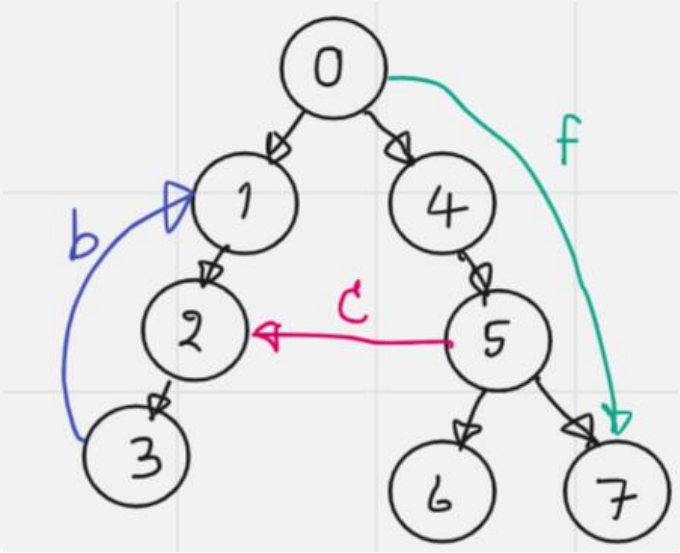




Edge Classification

1. **Tree edges** : are edges in the dfs forest G''. Edge (u,v) is a tree edge if v was discovered by exploring edge(u,v).

2. **Back edge** : are those edges (u,v) connecting a vertex u to an ancestor v in a dfs tree. Self-loop are considered to be back edges.
   which point from a node to one of its ancestors

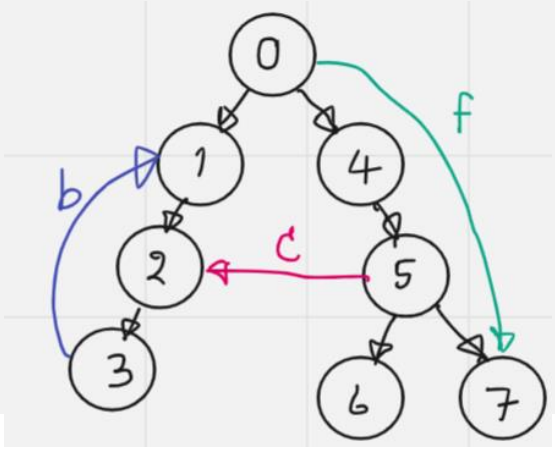


Classification of Edge

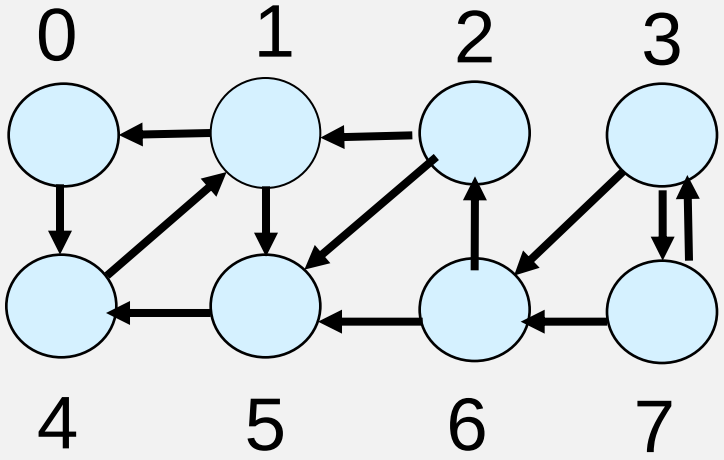
3. **Forward edges** : are those non tree edge (u,v) connecting a vertex u to a descendent v in a dfs tree.

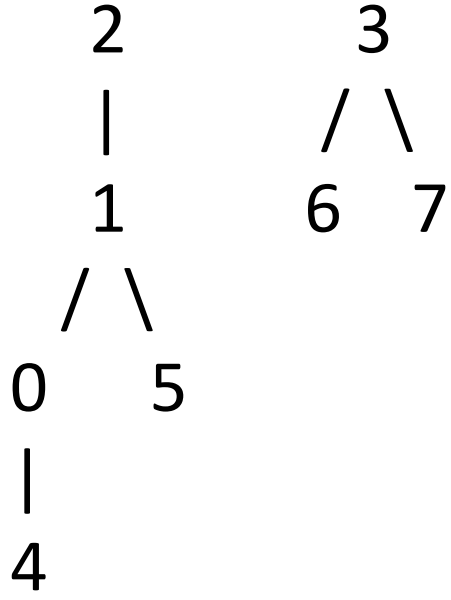
which point from a node of the tree to one of its descendants

4. **Cross edge :** are all other edges. They can go between vertices in the same depth-first search tree, as long as one vertex is not ancestor of the other, or they can go between vertices in difference dfs tree.



DFS trees

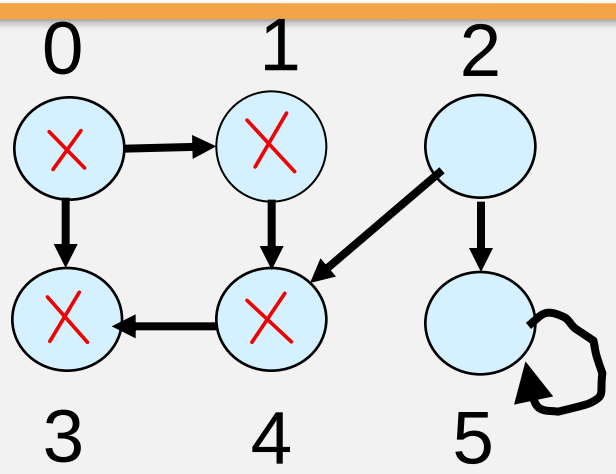


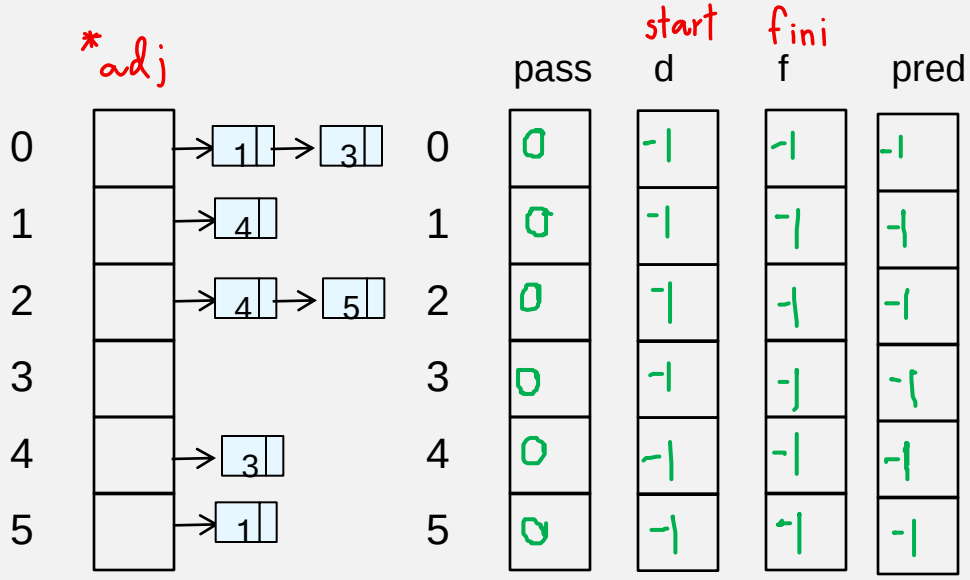


1. Tree edges
2. Back edges
3. Forward edges
4. Cross edges

4.4.2 Algorithm DFS (G)

```
DFS(G)
time=0;
for ( u=0;u< 6; u++)
{       pass[u]=0
        pred[u]=d[u]=f[u]=-1;
}
u=S  //สมมุติ s คือ โหนด 0
DFS_Visit(u)  กลับ
for ( u=0;u<6;u++)
{       if (pass[u]=0)
        {   DFS_Visit(u) 2
        }
}
```





```
// time=0 is global
 DFS_Visit(u)
1 pass[u]=1
2 d[u]=++time;
3 for each v ∈ adj[u]
4 {       if (pass[v] == 0)
5         {       pred[v] = u
6                 DFS_Visit(v)
7         }
8 }
9 pass[u]=1
10 f[u]=++time;
```

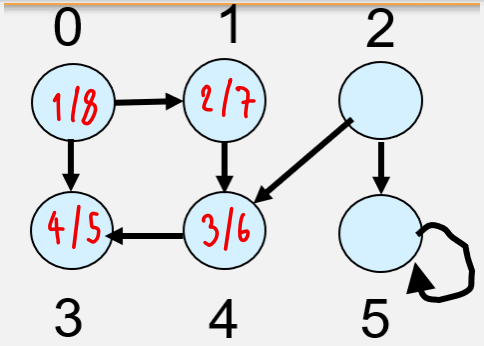
0,1,4,3

u(0) < V(1), V(3)

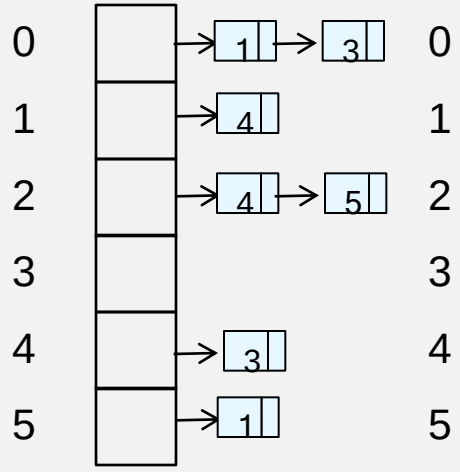
u(1) — V(4) — return กลับ

u(4) — V(3) — return กลับ

u(3) — return กลับ



*adj

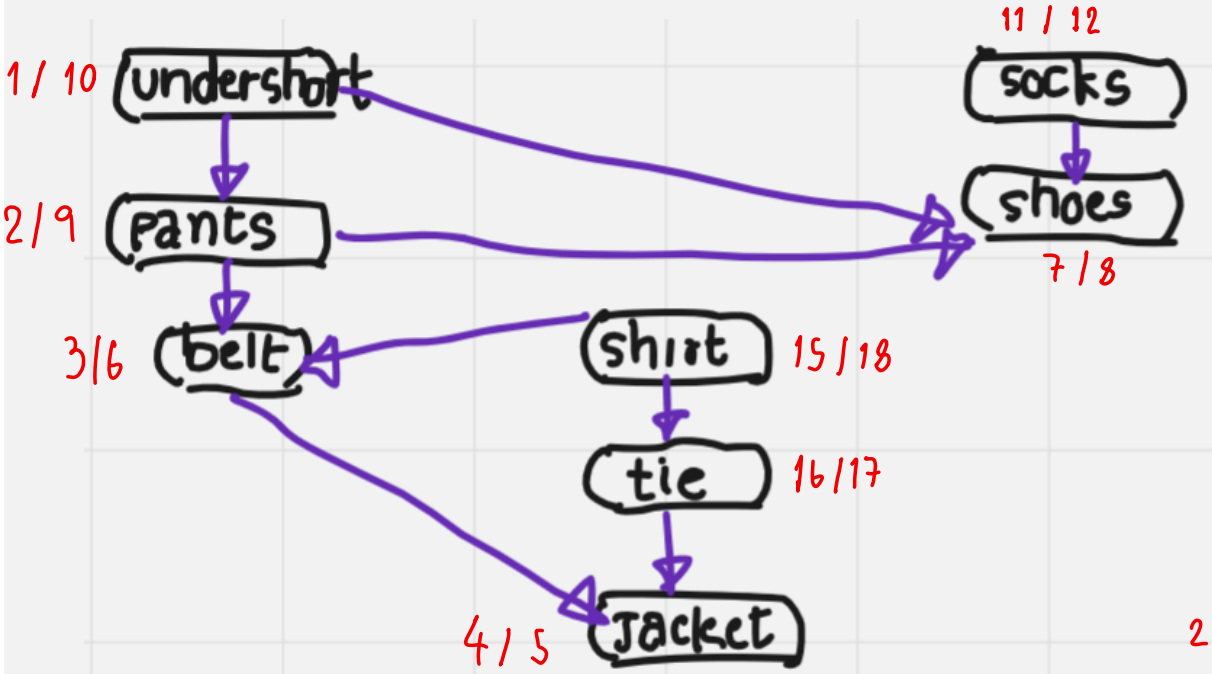


	pass	d	f	pred
0	0 1	-1 1	-1 8	-1
1	0 1	-1 2	-1 7	-1 0
2	0 1	-1 9	-1 12	-1
3	0 1	-1 4	-1 5	-1 4
4	0 1	-1 3	-1 6	-1 1
5	0 1	-1 10	-1 11	-1 2

4.4.3 Topological sort

Topological sort of a DAG G=(V,E) is a linear ordering of all its vertices such that if G contains an edge(u,v) , then u appears before v in the ordering. (If the graph is not acyclic, then no linear ordering is possible.)

Topological sort คือการนำเวอร์ทิซของกราฟกราฟหนึ่งมา
- เรียงเป็นเส้นตรงแบบมีลำดับ
- มีข้อกำหนดว่ากราฟนั้นจะต้องมีคุณสมบัติเป็น directed acyclic graph หรือ DAG (คือกราฟที่ไม่มีไซเคิล)



1. ทำ DFS โดยเริ่มต้นจาก node ที่ไม่มี incident to (ไม่ถูกชี้)
2. เลือก node ที่ f สูงสุด

4.4.4 Algorithm Topological_sort(G)

Topological_sort(G)

1) call DFS(G) to computer finishing times f[v] for each vertex v.

- ทำ DFS กราฟ

เลือก โหนด ที่ไม่ถูกชี้

2) as each vertex is finished, insert it onto the front of a linked list

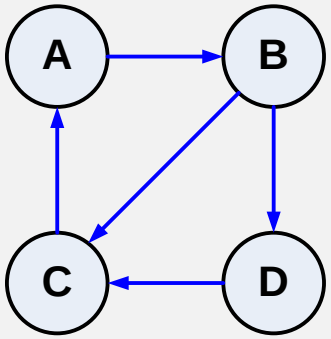
- เมื่อ DFS แล้ว ให้นำ vertex ที่มี finish time สูงสุดใส่ไว้ตำแหน่งแรกของ linklist

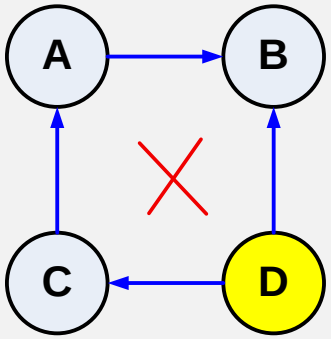
- finish time ที่มีค่ารองลงมาให้นำไปใส่ไว้ตำแหน่งที่ 2 ของ linklist และทำเช่นนี้โดยเรียงจากมากไปน้อย

3) return the link list of vertices.

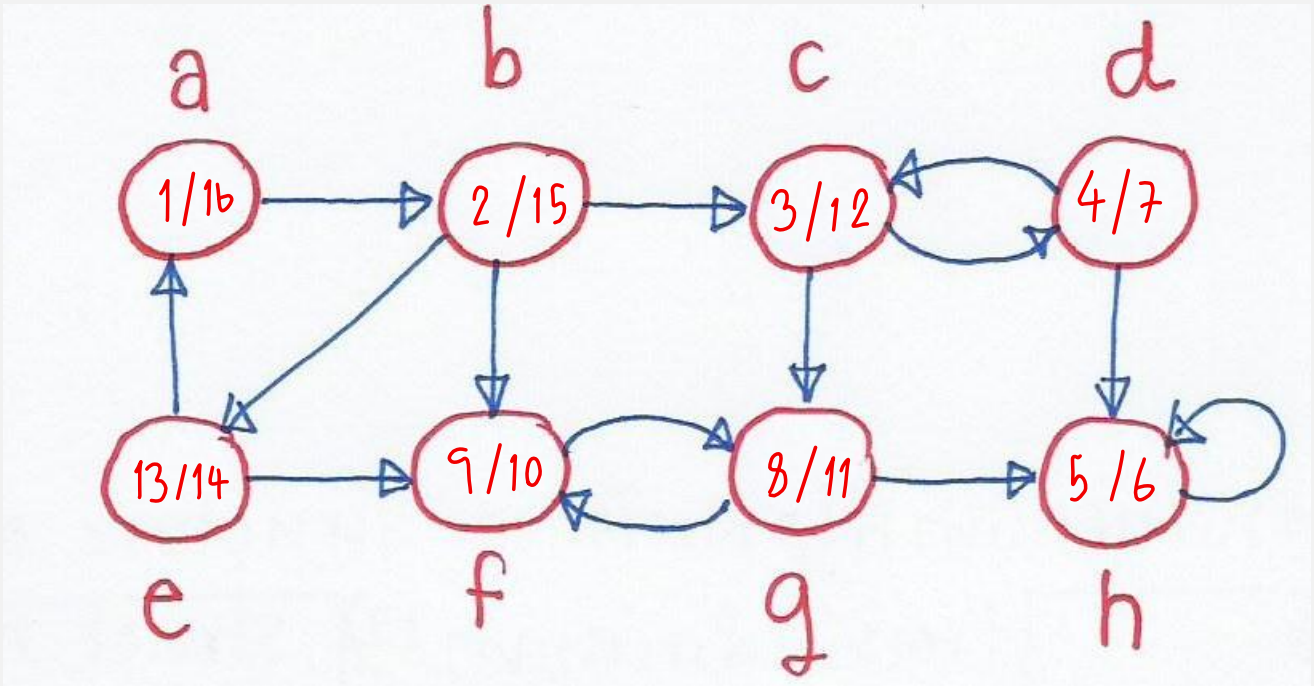
4.4.5 Strongly connected components

A digraph is ***strongly connected*** if every two vertices are reachable from each other. The ***strongly connected components*** of a graph are the equivalence classes of vertices under the "are ***mutually reachable***" relation.

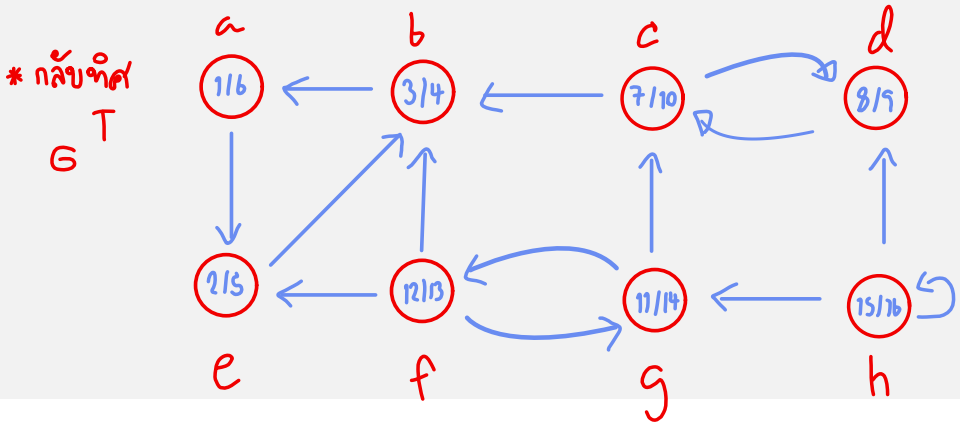


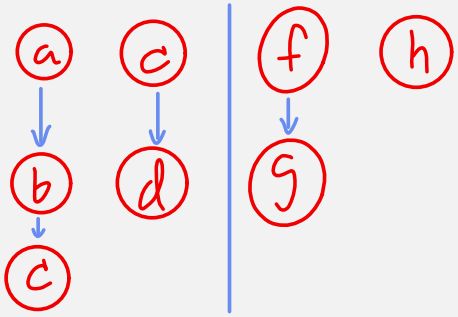


* ทุกโหนดไปหากันได้


a
b
c
d
1/16
2/15
3/12
4/7
13/14
9/10
8/11
5/6
e
f
g
h

DFS


* กลับทิศ
G T
a
b
c
d
1/16
3/4
7/10
8/9
2/15
12/13
11/14
15/16
e
f
g
h


a
b
c
c
d
f
g
h

4.4.6 Strongly-Connected-Component Algorithm

1) call DFS(G) to compute finishing times f[u] for each vertex u

2) compute GT (transpose graph)

3) call DES(GT), but the main loop of DFS, consider the vertices in order of decreasing f[u] (as computed in line 1) Max F[u] ⟶ min F[u]

4) output the vertices of each tree