DFS & Topological Sort

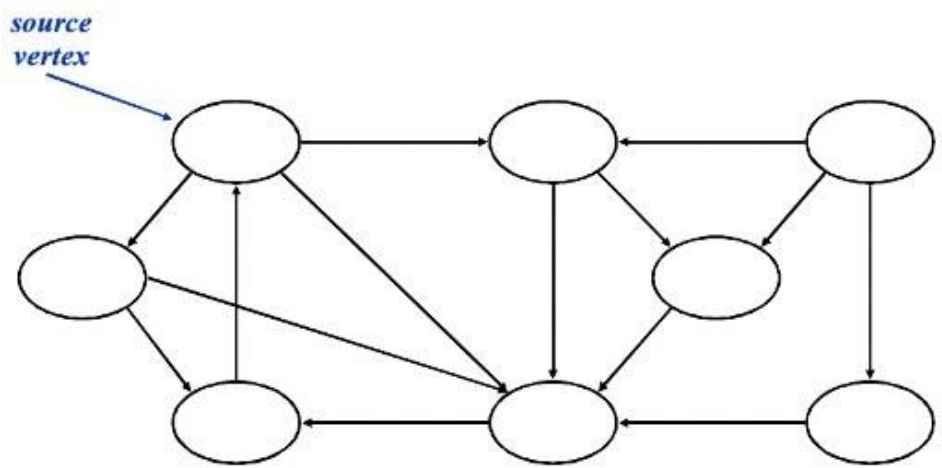
Charles Aunkan Gomes
Lecturer, Dept. of CSE
United International University
charles@cse.uiu.ac.bd

Depth-First Search

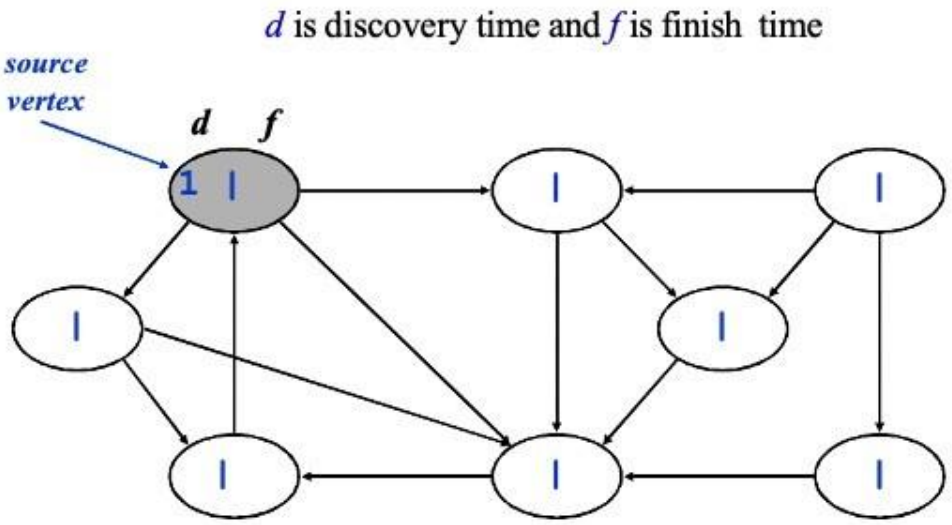
- *Depth-first search* is another strategy for exploring a graph
  - Explore "deeper" in the graph whenever possible
  - Edges are explored out of the most recently discovered vertex *v* that still has unexplored edges
  - When all of *v*'s edges have been explored, backtrack to the vertex from which *v* was discovered

- Vertices initially colored white
- Then colored gray when discovered
- Then black when finished

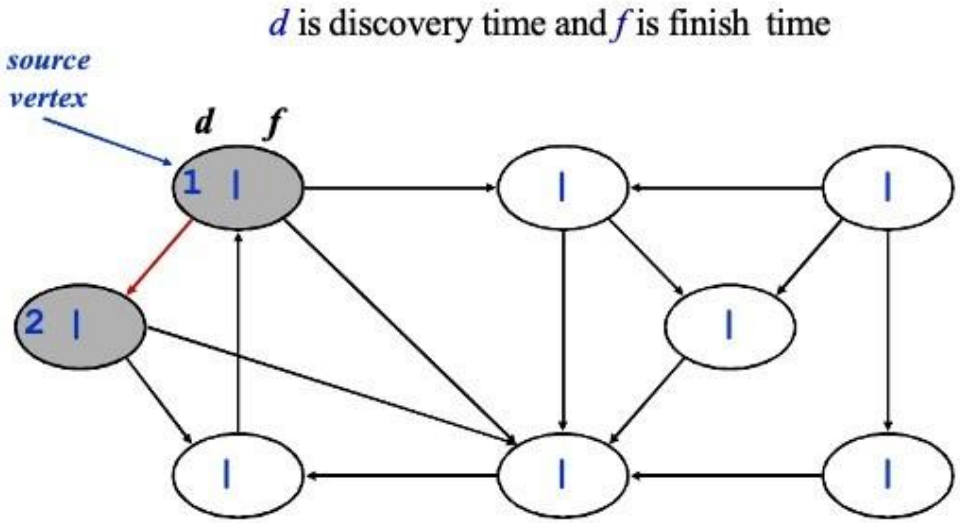
DFS - Example



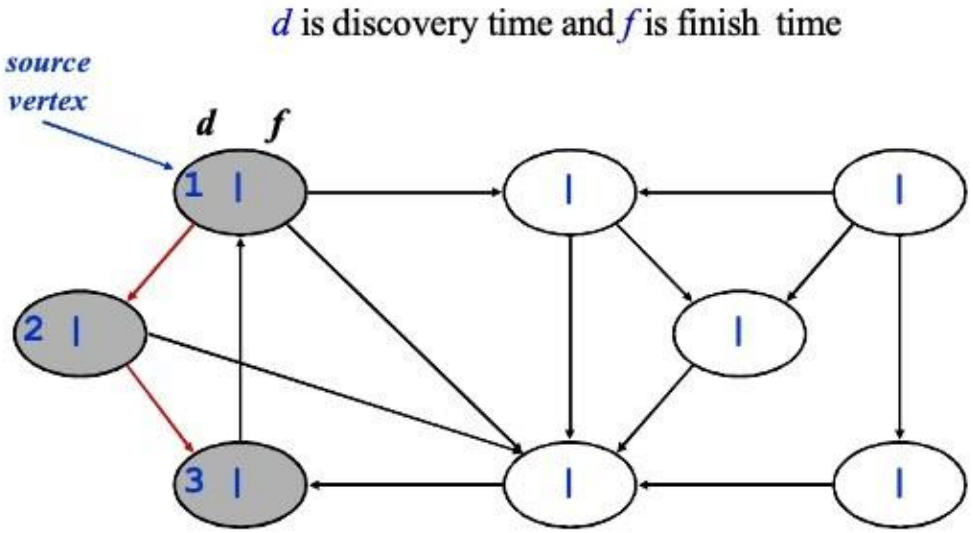
DFS - Example



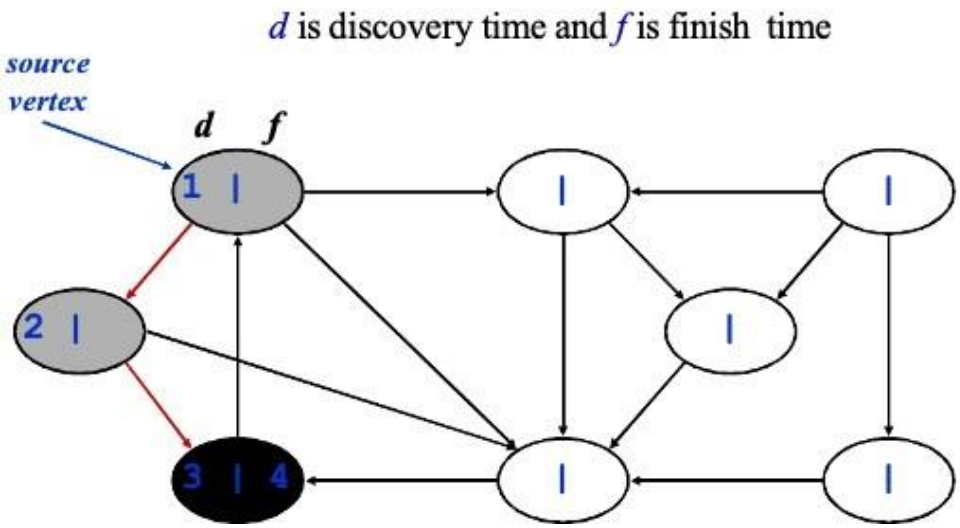
DFS - Example



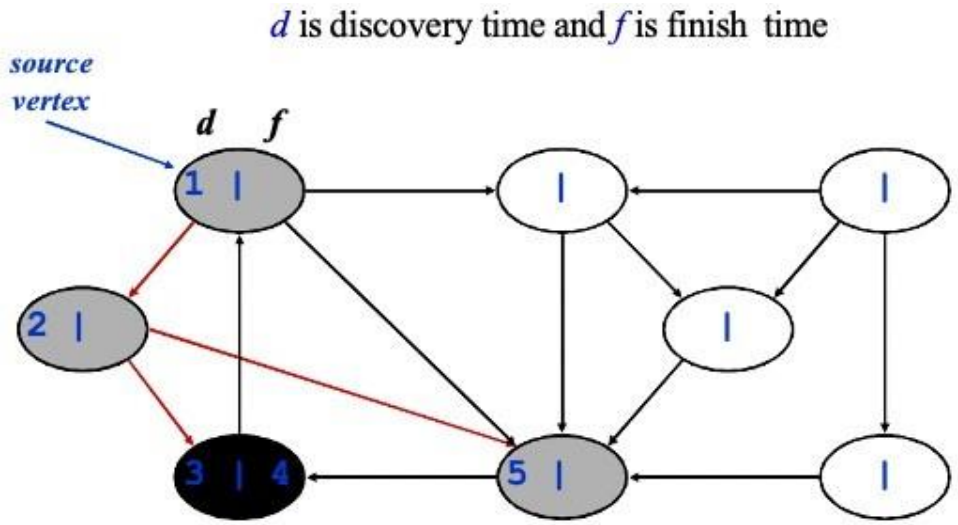
DFS - Example



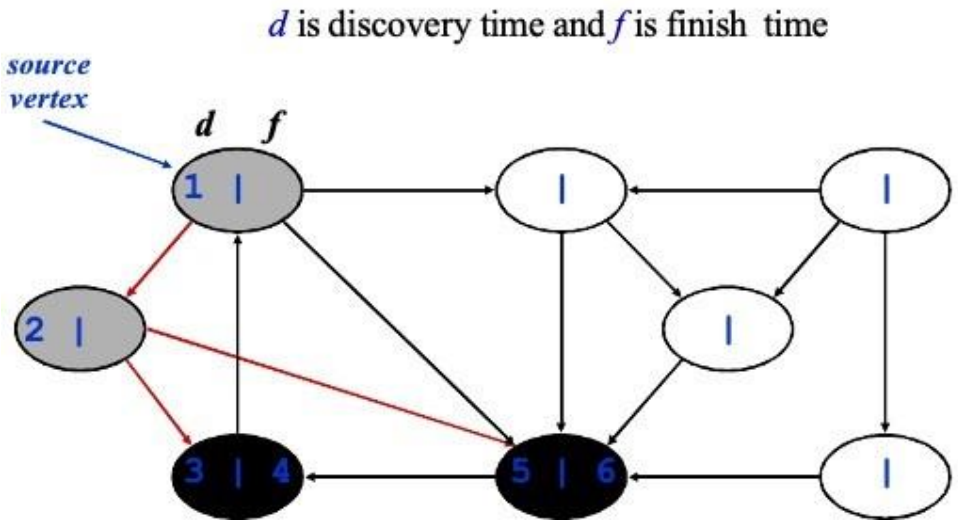
DFS - Example


d is discovery time and f is finish time
source
vertex
d
f
1
2
3
4

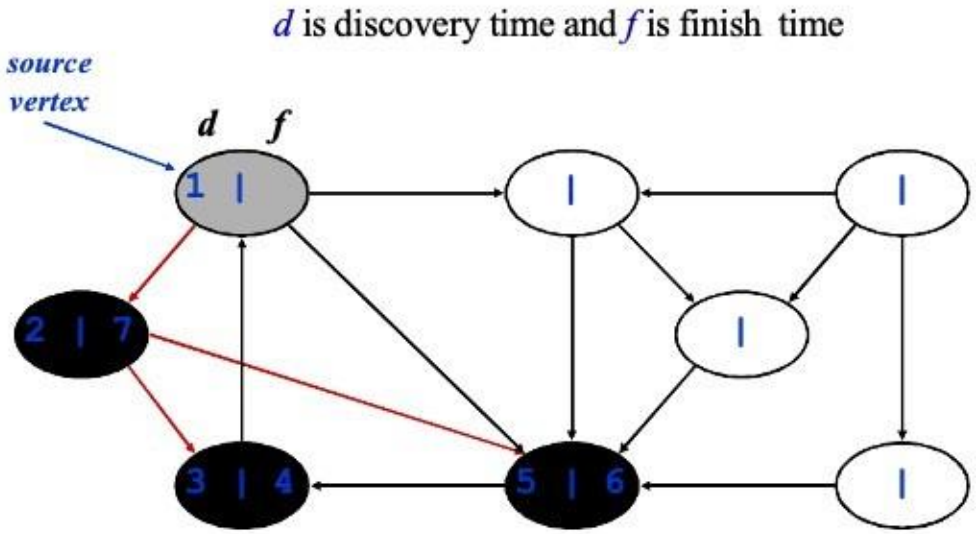
DFS - Example


d is discovery time and f is finish time
source
vertex
d
f
1 |
|
|
2 |
|
3 | 4
5 |
|

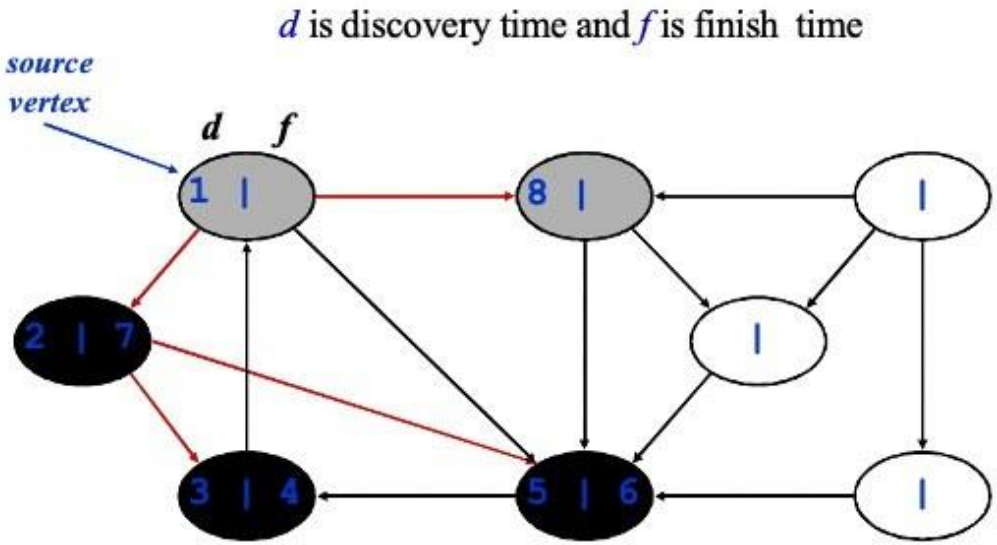
DFS - Example


d is discovery time and f is finish time
source vertex
d
f
1 |
|
|
2 |
|
3 | 4
5 | 6
|

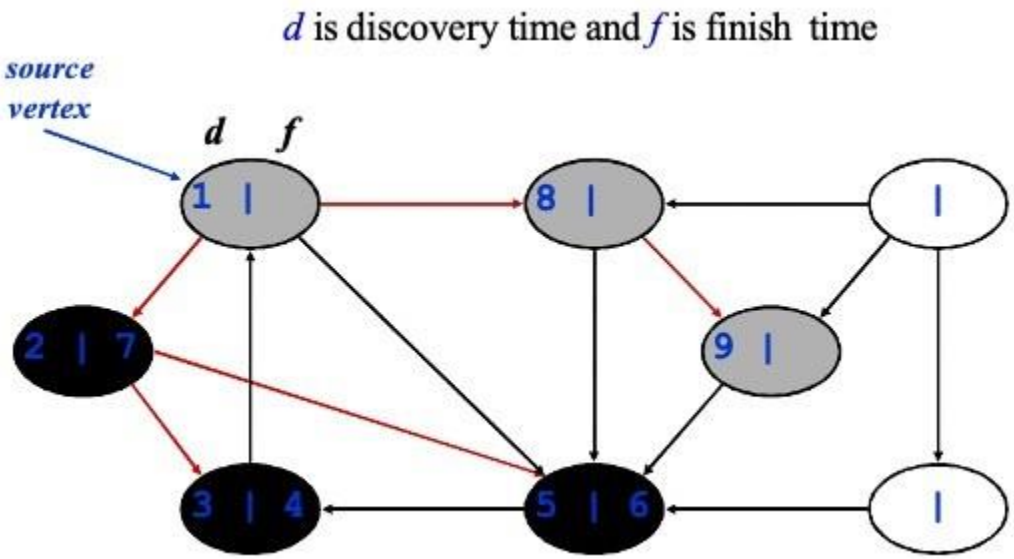
DFS - Example


d is discovery time and f is finish time
source
vertex
d
f
1 | |
| |
| |
2 | 7
| |
3 | 4
5 | 6
| |

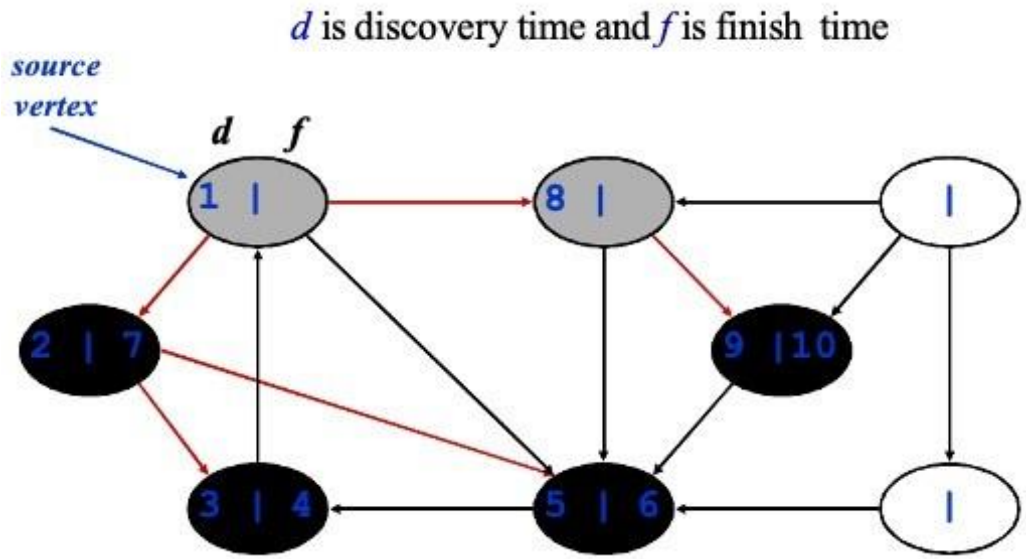
DFS - Example


d is discovery time and f is finish time
source
vertex
d
f
1 | |
8 | |
| |
2 | 7
| |
3 | 4
5 | 6
| |

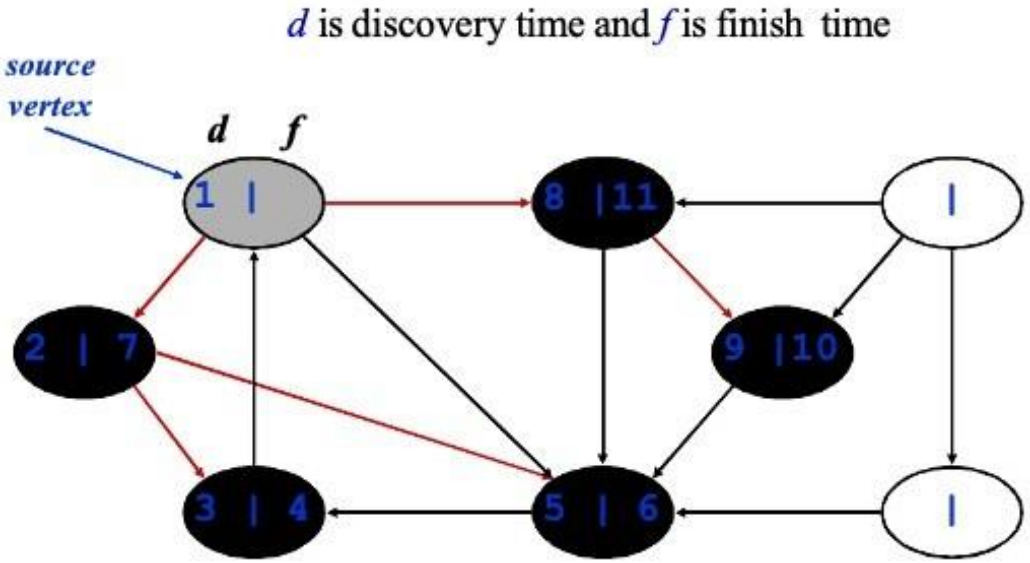
DFS - Example


d is discovery time and f is finish time
source vertex
d f
1 | 8
2 | 7
9 |
3 | 4
5 | 6
|
|
|

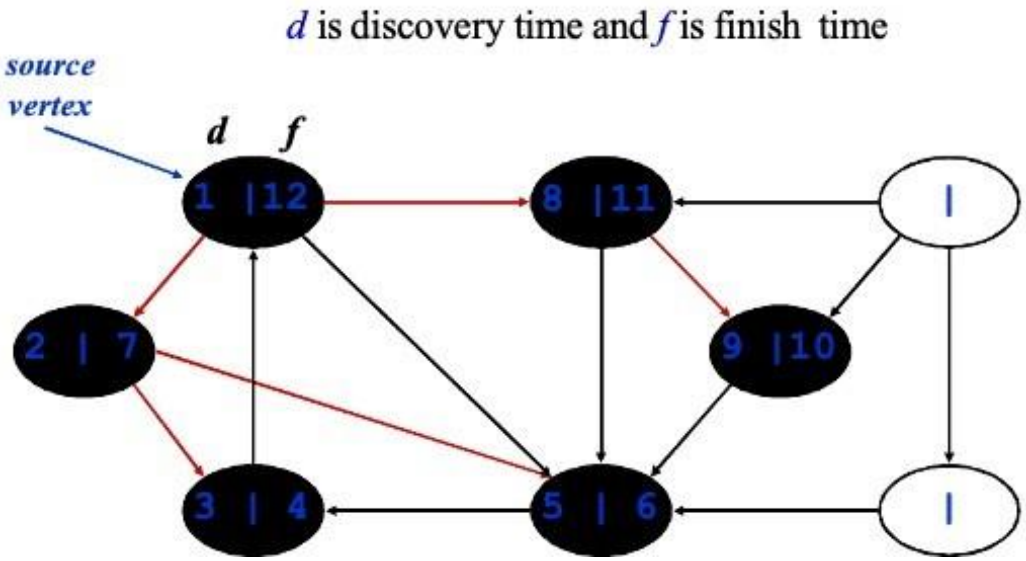
DFS - Example


d is discovery time and f is finish time
source
vertex
d
f
1 |
8 |
|
2 | 7
9 |10
3 | 4
5 | 6
|

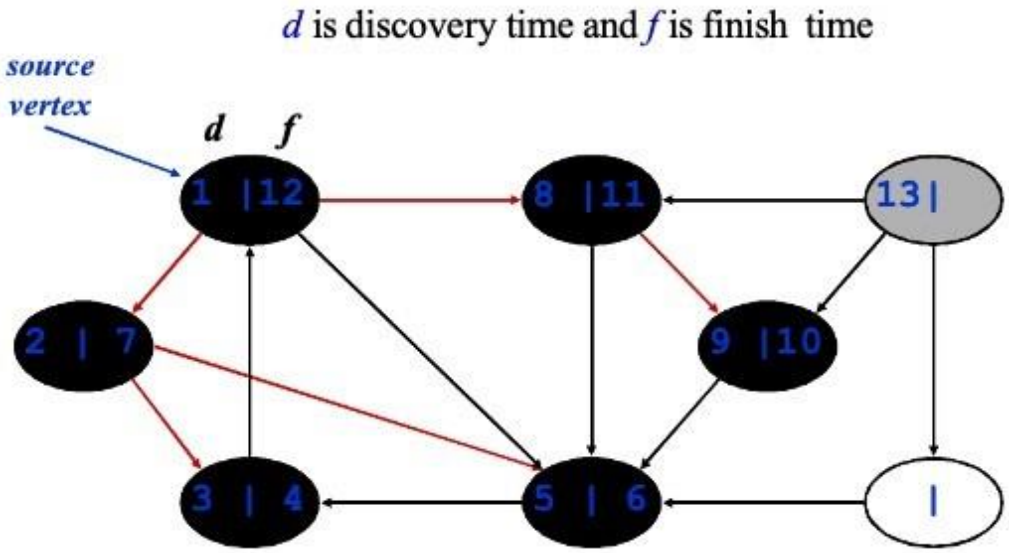
DFS - Example


d is discovery time and f is finish time
source vertex
d f
1 | 8 |11 |
2 | 7 9 |10
3 | 4 5 | 6 |

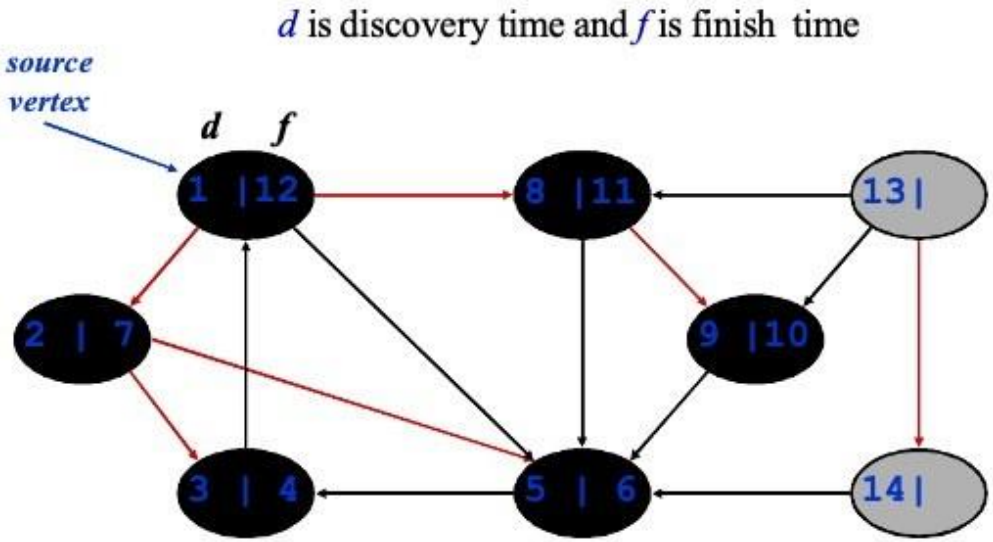
DFS - Example


d is discovery time and f is finish time
source
vertex
d
f
1 | 12
8 | 11
|
2 | 7
9 | 10
3 | 4
5 | 6
|

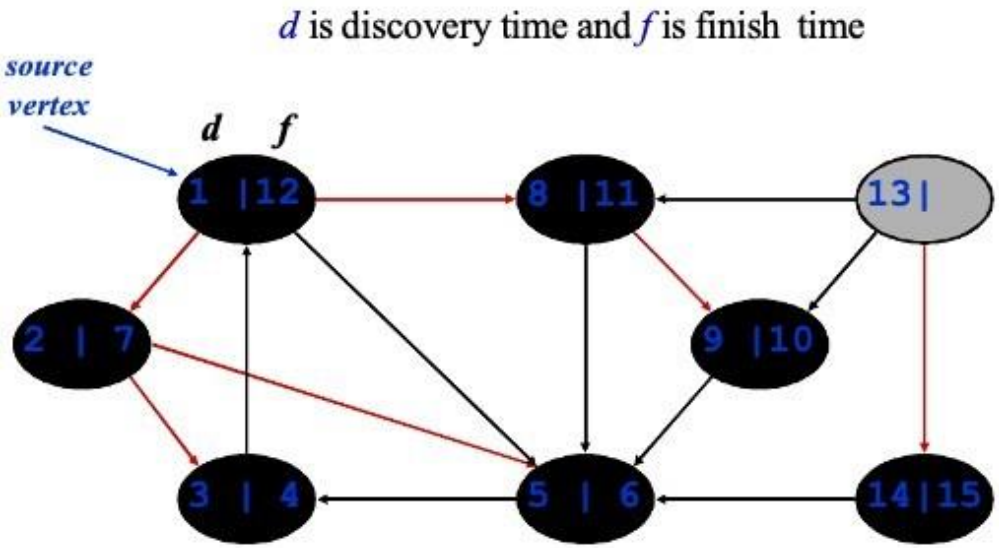
DFS - Example


d is discovery time and f is finish time
source
vertex
d
f
1 | 12
8 | 11
13 |
2 | 7
9 | 10
3 | 4
5 | 6
|

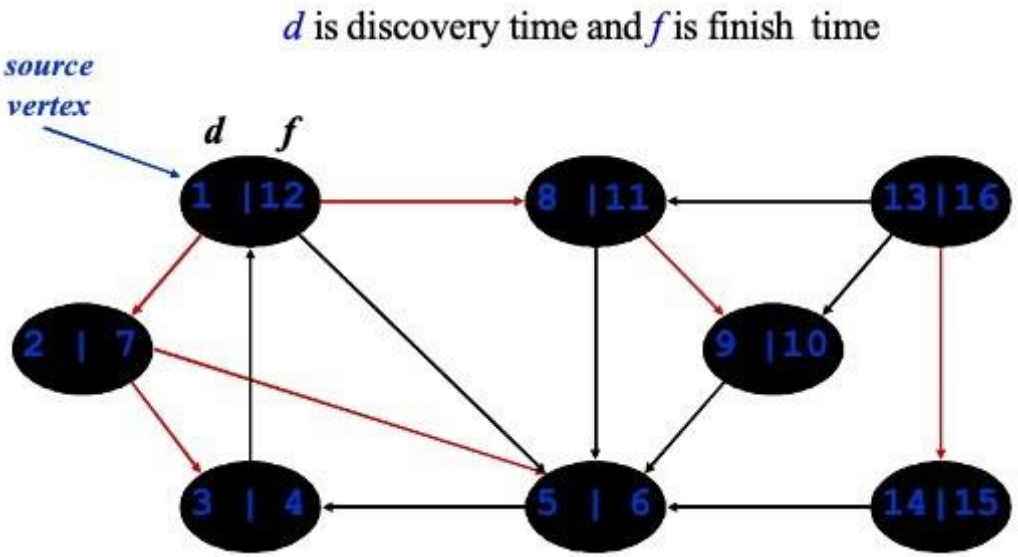
DFS - Example


d is discovery time and f is finish time
source
vertex
d
f
1 |12
8 |11
13|
2 | 7
9 |10
3 | 4
5 | 6
14|

DFS - Example

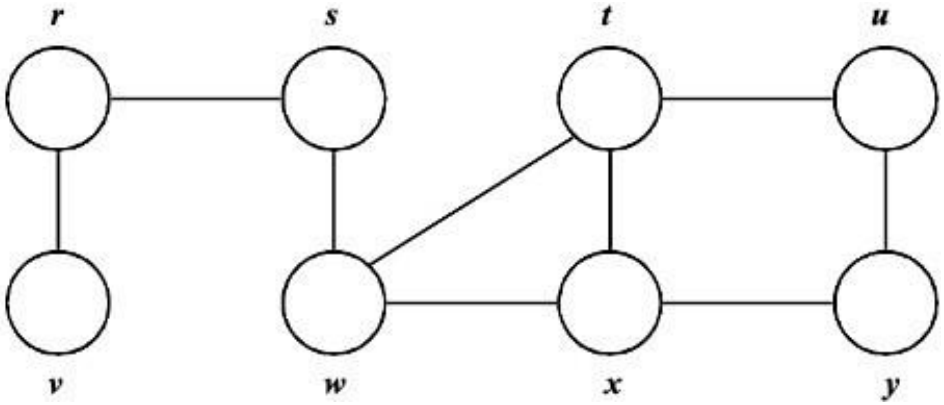

d is discovery time and f is finish time
source
vertex
d
f
1 | 12
8 | 11
13 |
2 | 7
9 | 10
3 | 4
5 | 6
14 | 15

DFS - Example


d is discovery time and f is finish time
source
vertex
d
f
1 | 12
8 | 11
13 | 16
2 | 7
9 | 10
3 | 4
5 | 6
14 | 15

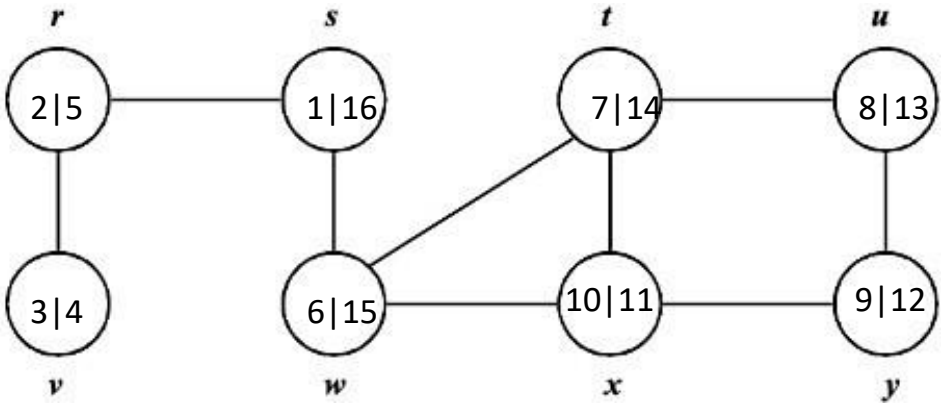
DFS – Example 2

Class activity – Try this one yourselves



DFS – Example 2

Sample Solution(Not only solution)



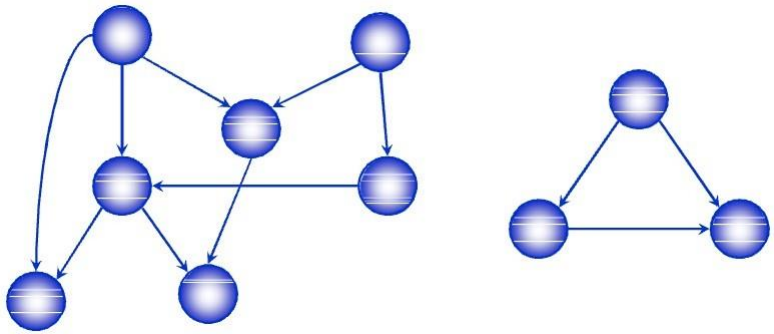
DFS – Code

```
DFS(G)
{
   for each vertex u∈G->V
   {
      u->color = WHITE;
   }
   time = 0;
   for each vertex u∈G->V
   {
      if (u->color == WHITE)
         DFS_Visit(u);
   }
}
```

```
DFS_Visit(u)
{
  u->color = GREY;
  time = time+1;
  u->d = time;
  for each v ∈ u->Adj[]
  {
     if (v->color == WHITE)
        DFS_Visit(v);
  }
  u->color = BLACK;
  time = time+1;
  u->f = time;
  }
```

Directed Acyclic Graph

- A *directed acyclic graph* or *DAG* is a directed graph with no directed cycles



Topological Sort

- A *topological sort* of a DAG is
  - a linear ordering of all vertices of the graph G such that vertex u comes before vertex v if (u, v) is an edge in G.

- DAG indicates precedence among events:
  - events are graph vertices, edge from u to v means event u has precedence over event v

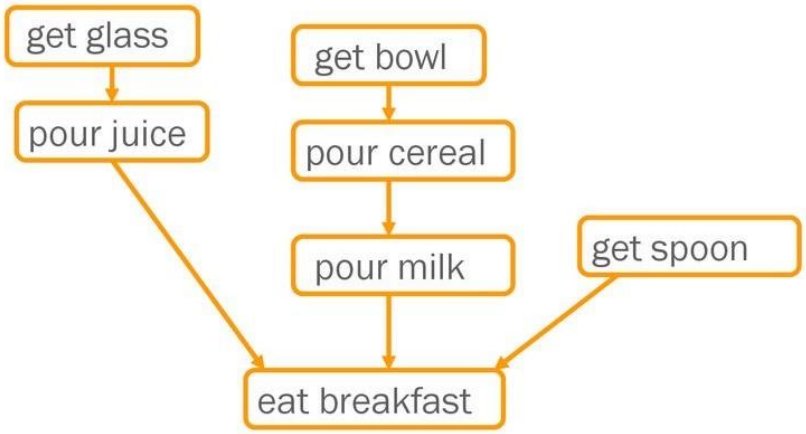
- Real-world example:
  - getting dressed
  - course registration
  - tasks for eating meal

Precedence Example

- Tasks that have to be done to eat breakfast:
  - get glass, pour juice, get bowl, pour cereal, pour milk, get spoon, eat.

- Certain events must happen in a certain order (ex: get bowl before pouring milk)
- For other events, it doesn't matter (ex: get bowl and get spoon)

Precedence Example



Order: glass, juice, bowl, cereal, milk, spoon, eat.

Why Acyclic?

- Why must directed graph by acyclic for the topological sort problem?
- Otherwise, no way to order events linearly without violating a precedence constraint.

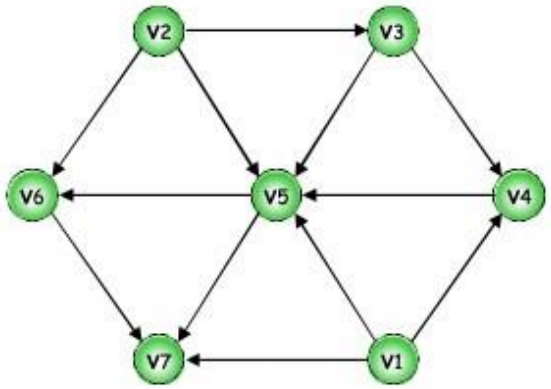
Topological Sort Algorithm

TOPOLOGICAL-SORT(G)

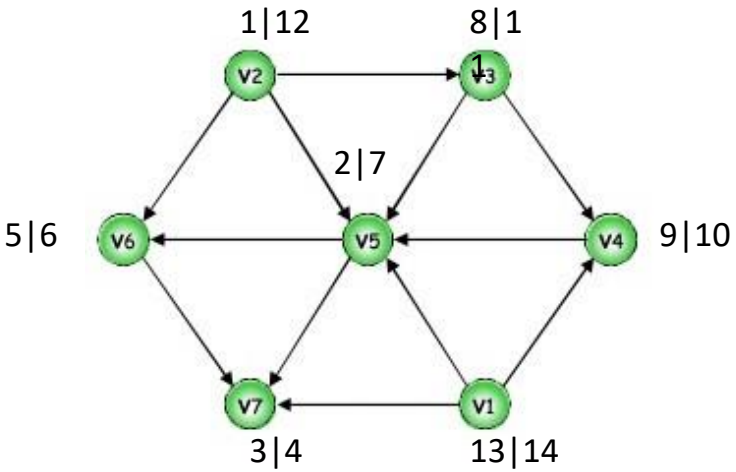
1 call DFS(G) to compute finishing times $f[v]$ for each vertex v
2 as each vertex is finished, insert it onto the front of a linked list
3 **return** the linked list of vertices

- Time: $O(V + E)$

Topological Sort: Example

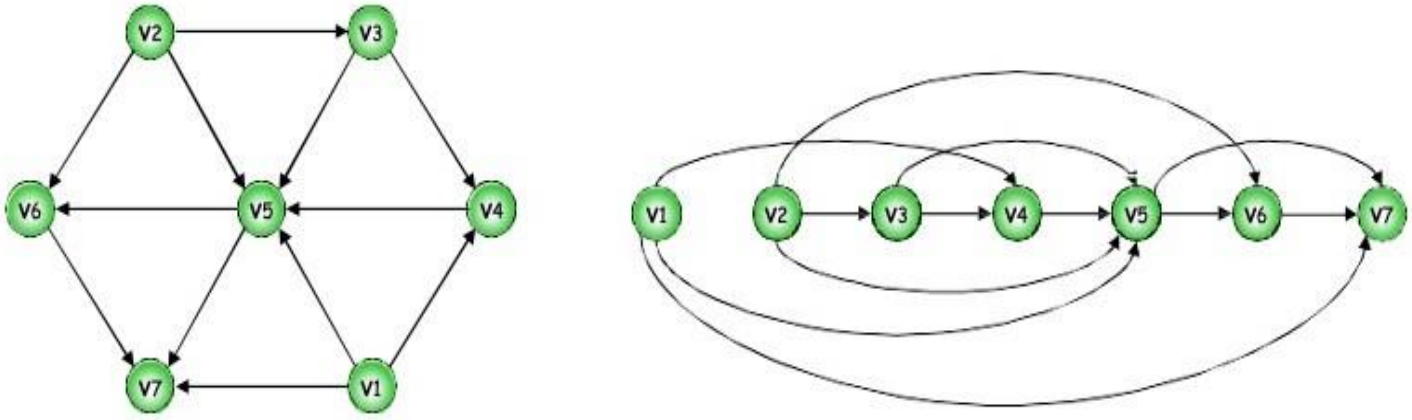


Topological Sort: Example



Topological Order: v1->v2->v3->v4->v5->v6->v7

Topological Sort: Example



Topological Order: v1, v2, v3, v4, v5, v6, v7

Topological Sort: Using In Degree (Algorithm)

- Steps for finding the topological ordering of a DAG:

Step-1: **Compute in-degree** for each of the vertices present in the DAG and initialize the count of visited nodes as 0;

Step-2: Add all **vertices with in-degree equals 0** into a queue

Step-3: Remove a vertex from the queue and

then Increment count of visited nodes by

1;

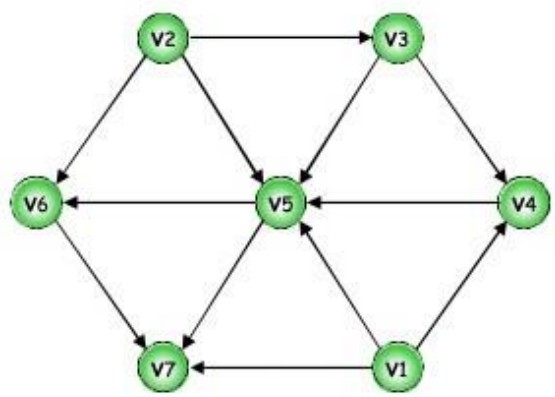
Decrease in-degree by 1 for all its neighboring nodes;

If in-degree of a neighboring node is reduced to zero, then add it to the queue;

Step 4: Repeat Step 3 until **the queue is empty**;

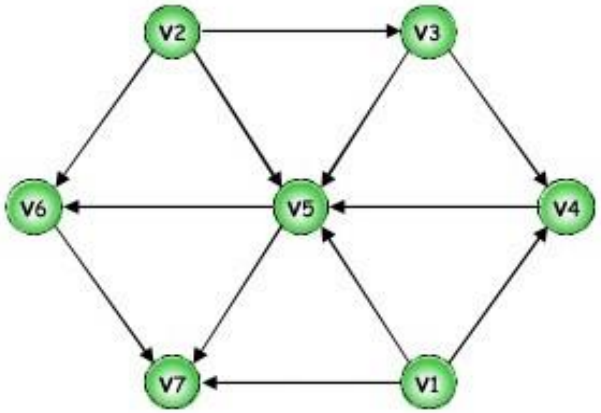
Step 5: If count of visited nodes is **not** equal to the number of nodes in the graph then the topological sort is not possible for the given graph.

Topological Sort: Using In Degree (Algorithm)

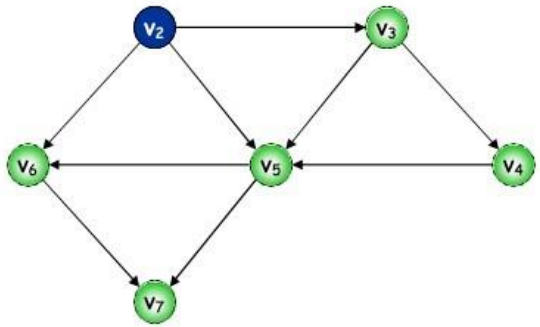


0	0	1	2	4	2	3
v1	v2	v3	v4	v5	v6	v7

Topological Sort: Using In Degree

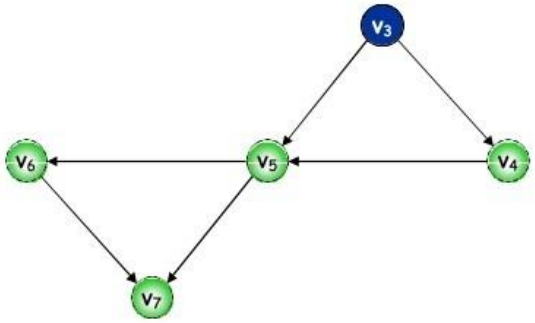


Topological Sort: Using In Degree



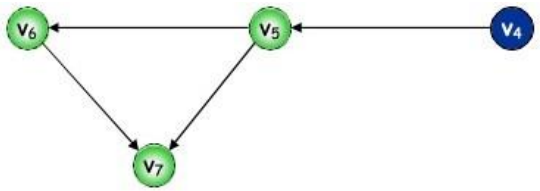
Topological order: v_1

Topological Sort: Using In Degree



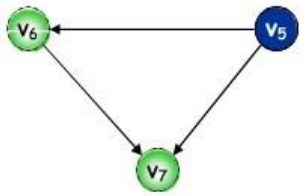
Topological order: v_1, v_2

Topological Sort: Using In Degree



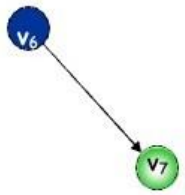
Topological order: v_1, v_2, v_3

Topological Sort: Using In Degree



Topological order: v_1, v_2, v_3, v_4

Topological Sort: Using In Degree

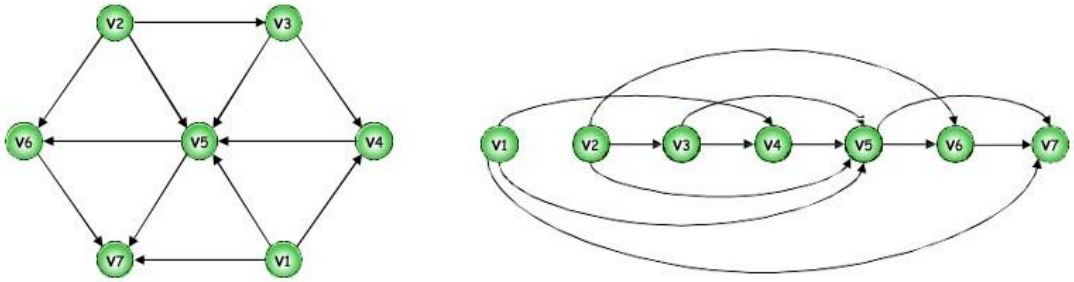


Topological order: v_1, v_2, v_3, v_4, v_5

Topological Sort: Using In Degree

Topological order: $v_1, v_2, v_3, v_4, v_5, v_6$

Topological Sort: Example



Topological Order: v1, v2, v3, v4, v5, v6, v7

When to use: BFS vs DFS

- If we know a solution is not far from the root of the tree, BFS might be better.
- If the tree is very deep and solutions are rare, DFS might take an extremely long time, but BFS could be faster.
- If the tree is very wide, a BFS might need too much memory, so it might be completely impractical.
- If solutions are frequent but located deep in the tree, DFS could be better.
- If the search tree is very deep we will need to restrict the search depth for DFS (for example with iterative deepening).

THANK YOU