Algorithms

Graph Searching Techniques:

Depth-First Search (DFS)

Topological Sorting

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

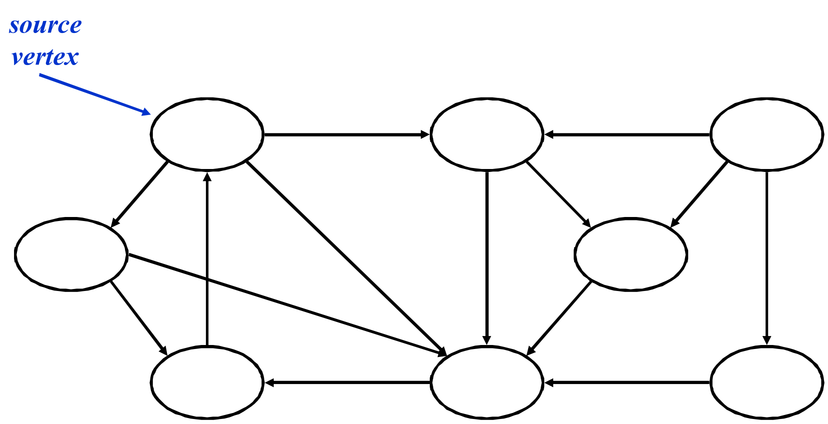
Depth-First Search

- *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 grey when discovered
- Then black when finished

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

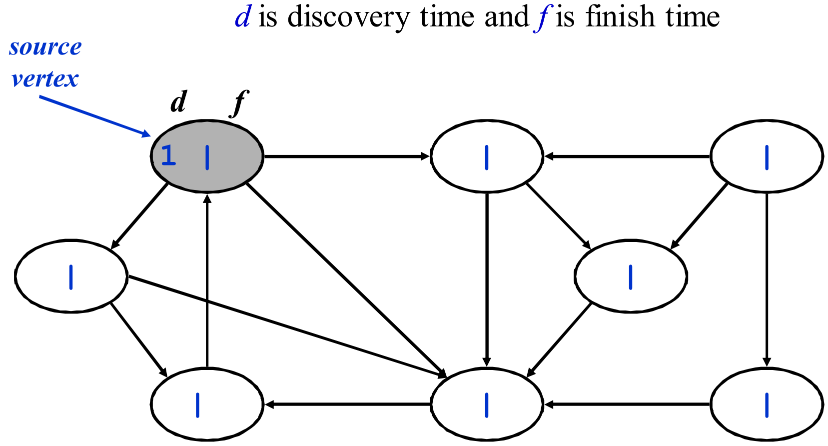
DFS Example



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

DFS Example

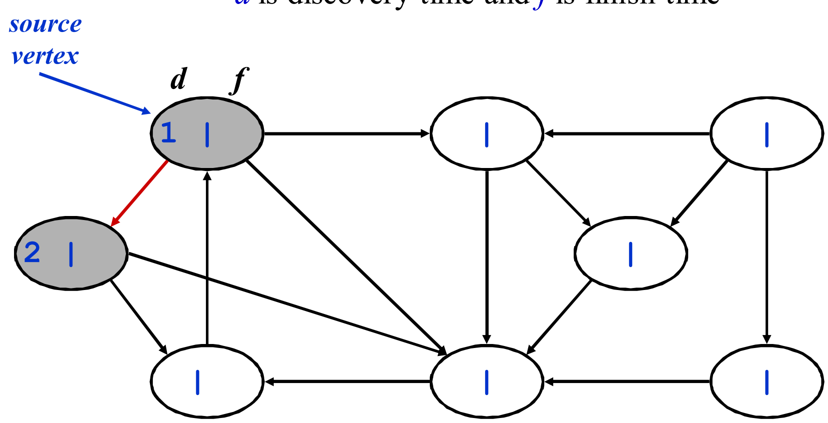
d is discovery time and *f* is finish time



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

DFS Example

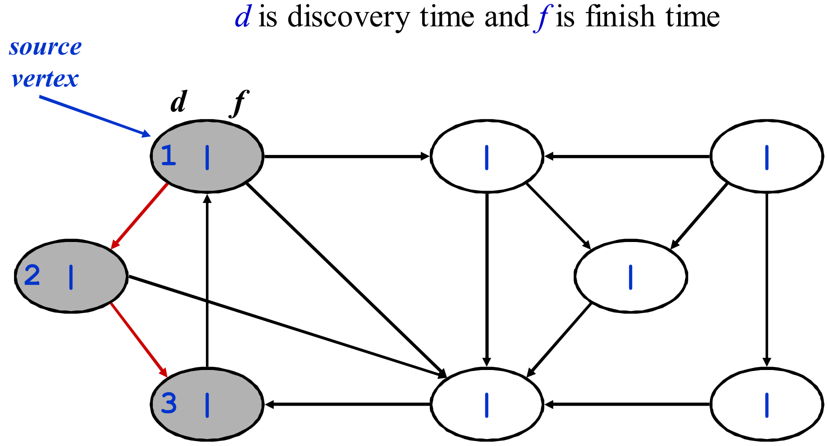
d is discovery time and *f* is finish time



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

DFS Example

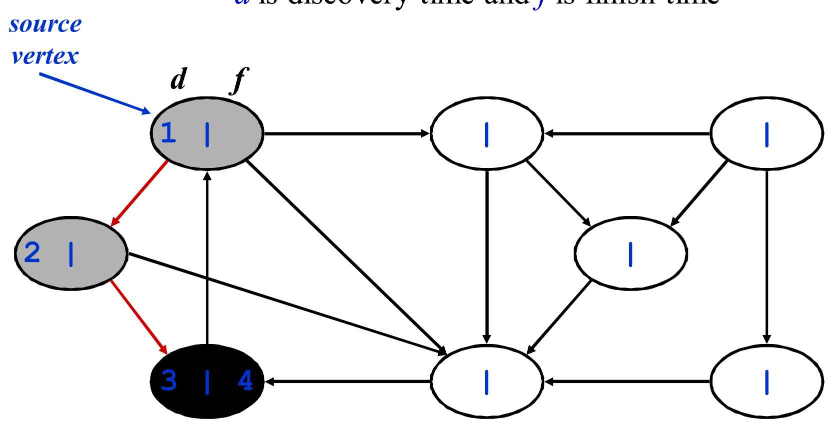
d is discovery time and *f* is finish time



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

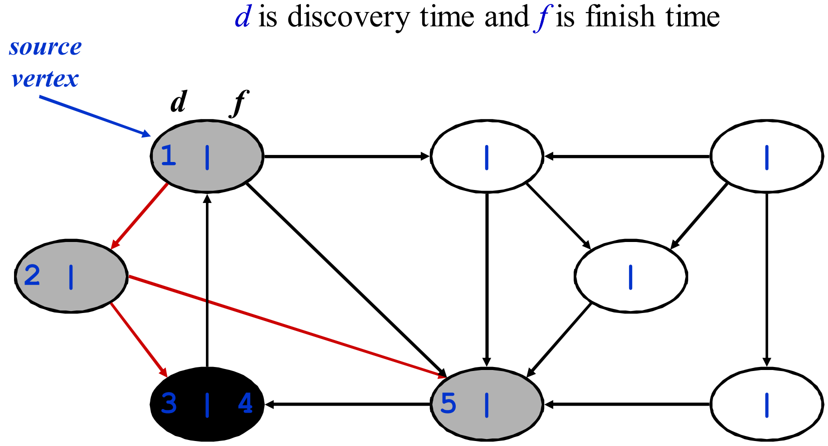
DFS Example

d is discovery time and *f* is finish time



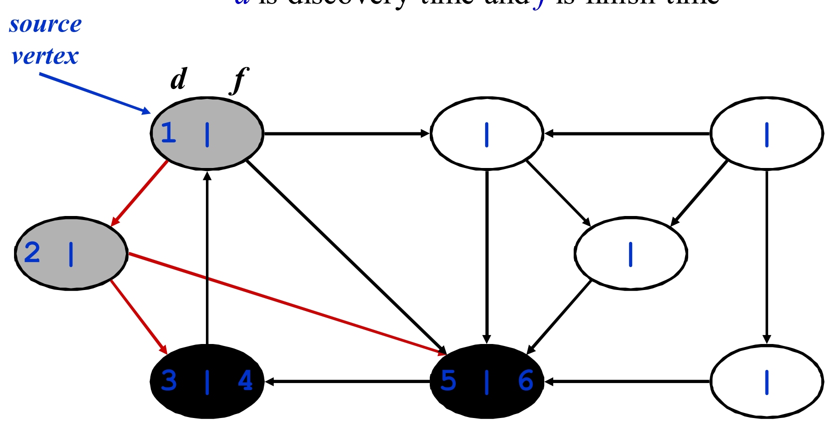
DFS Example

d is discovery time and *f* is finish time



DFS Example

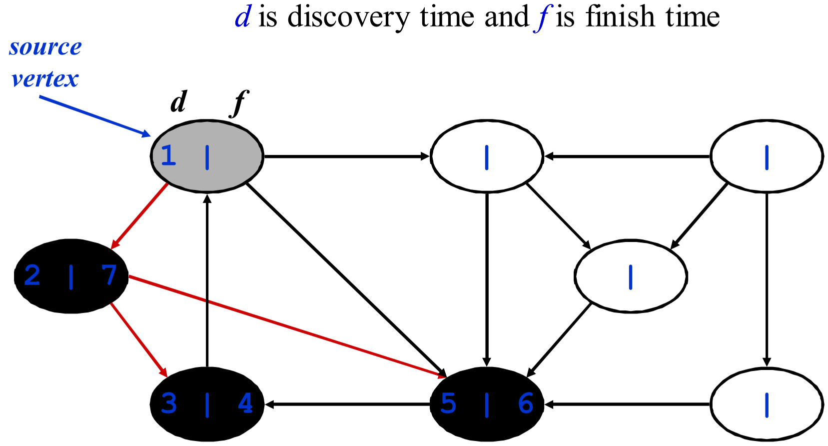
d is discovery time and *f* is finish time



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

DFS Example

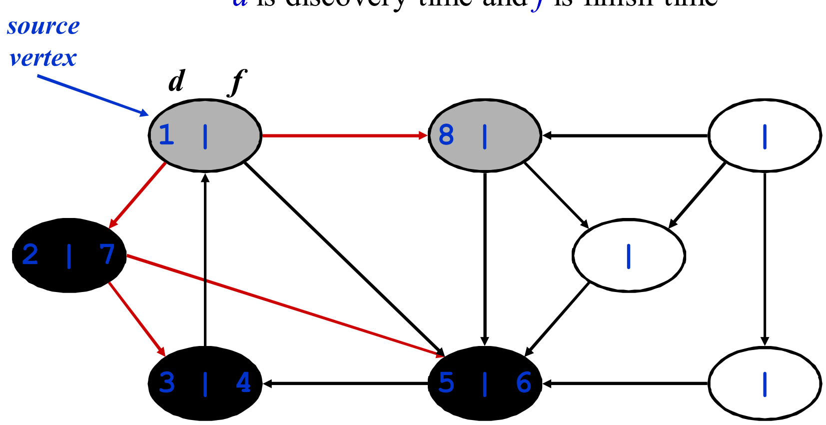
d is discovery time and *f* is finish time



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

DFS Example

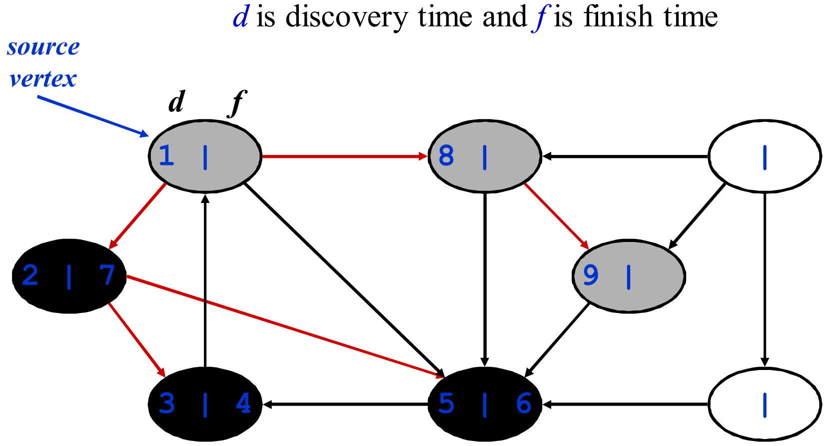
d is discovery time and *f* is finish time



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

DFS Example

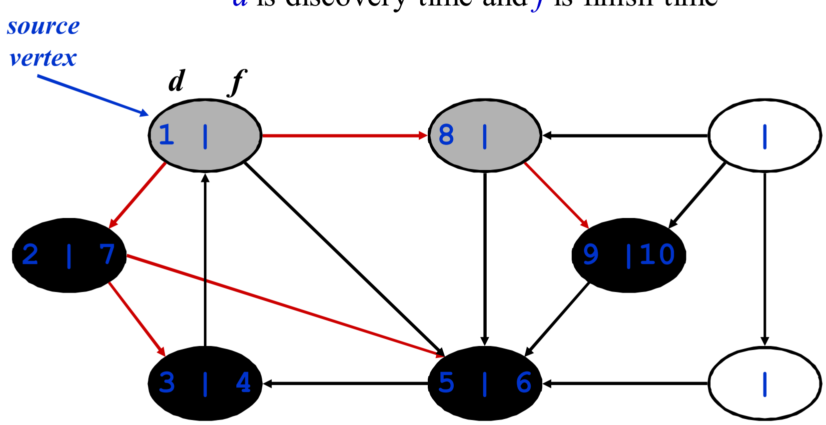
d is discovery time and *f* is finish time



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

DFS Example

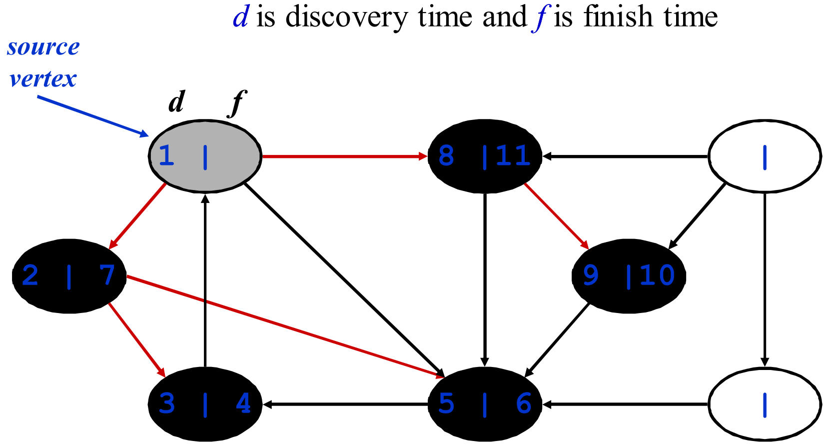
d is discovery time and *f* is finish time



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

DFS Example

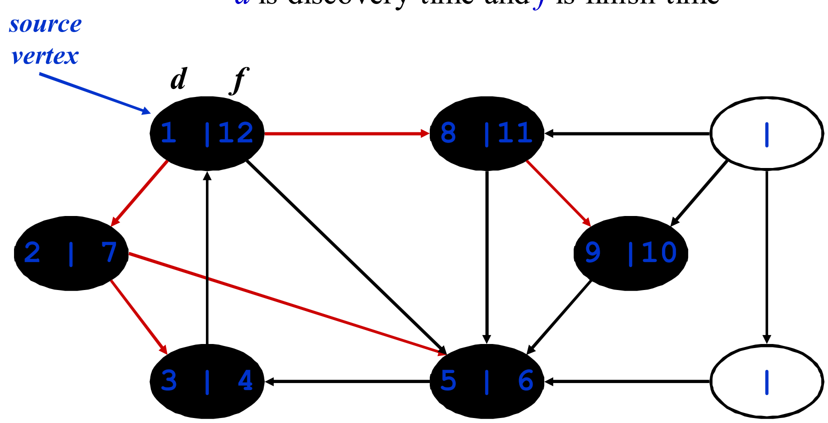
d is discovery time and *f* is finish time



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

DFS Example

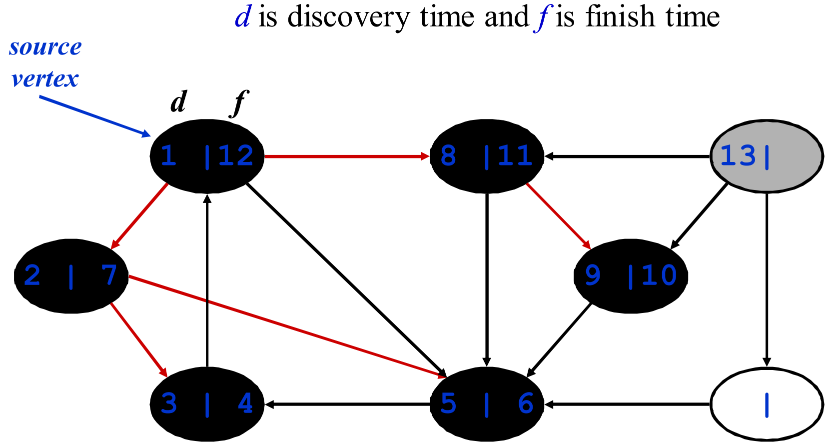
d is discovery time and *f* is finish time



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

DFS Example

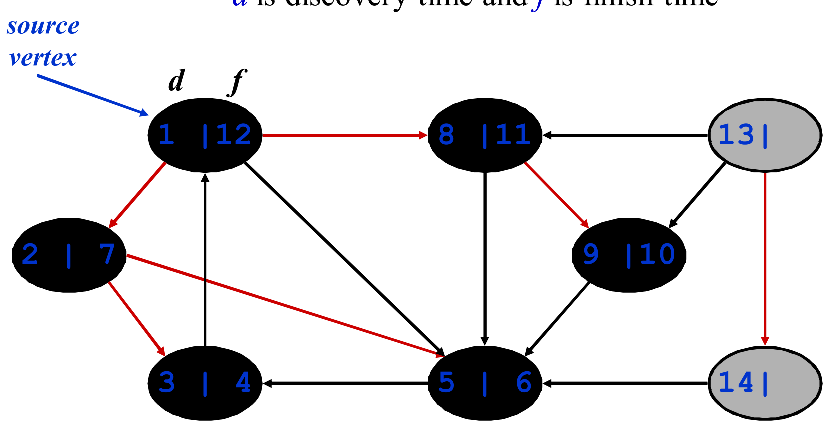
d is discovery time and *f* is finish time



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

DFS Example

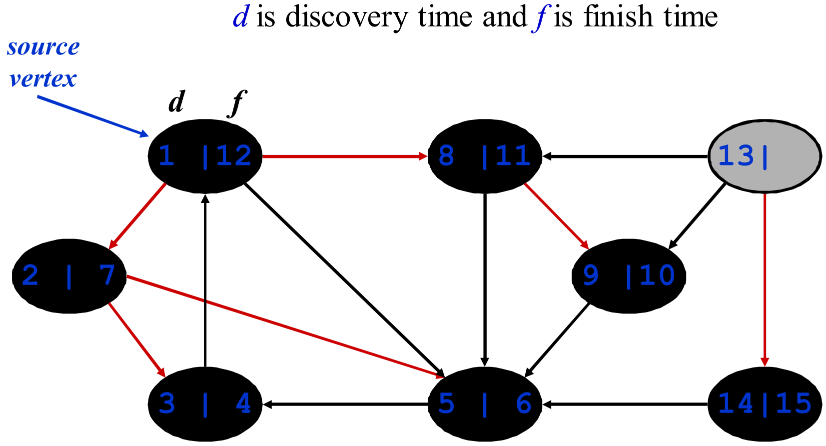
d is discovery time and *f* is finish time



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

DFS Example

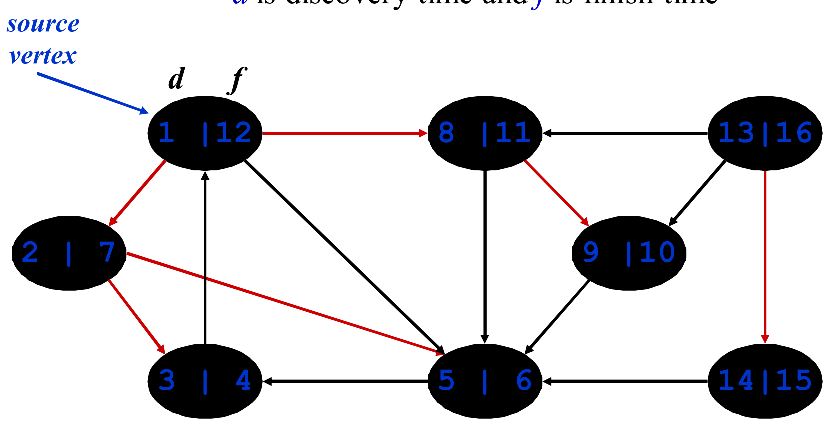
d is discovery time and *f* is finish time



Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

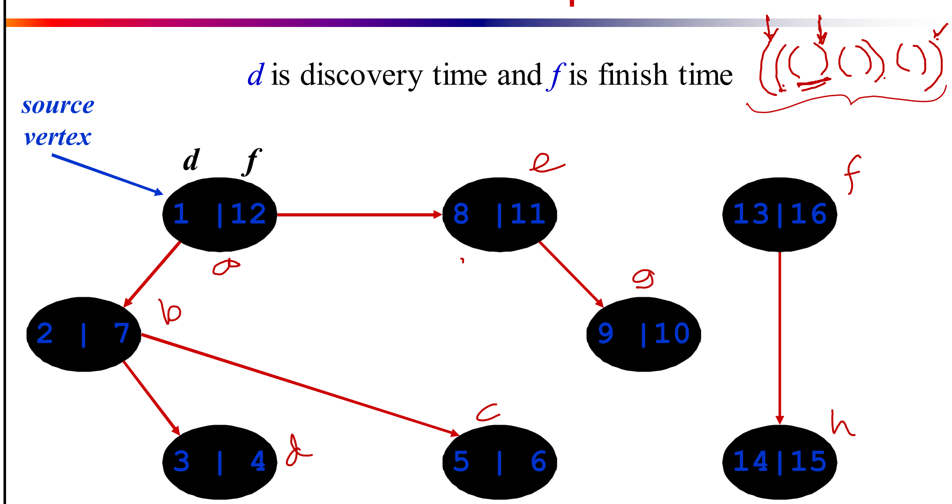
DFS Example

d is discovery time and *f* is finish time



DFS Example

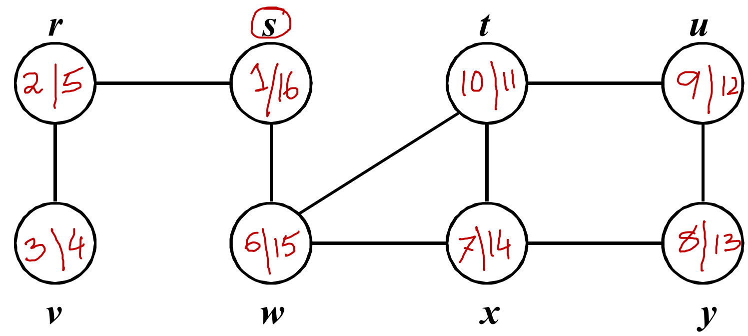
d is discovery time and *f* is finish time



DFS Order:
a b d c e g f h

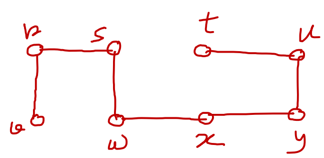
Output: DFS Forest

DFS Example 2



DFS Order: s r v w x y u t

DFS Spanning Tree



Depth-First Search: The 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;
}
```

stack

back track

Depth-First Search: The 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;
}
```

What does `u->d` represent?

Depth-First Search: The 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;
}
```

What does `u->f` represent?

Depth-First Search: The 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;
}
```

Will all vertices eventually be colored black?

Depth-First Search: The Code

```
DFS(G)
{
O(n)  for each vertex u ∈ G->V
      {
          u->color = WHITE;
      }
O(1) ← time = 0;
O(n)  for each vertex u ∈ G->V
      {
          if (u->color == WHITE)
              DFS_Visit(u);
      }
}
```

$O(n)$

```
DFS_Visit(u) ← is called n times
{
O(n)   u->color = GREY;    each call O(1), total : O(n)
       time = time+1;
       u->d = time;
O(m) → for each v ∈ u->Adj[] ←  deg(u)  ← each
       {
  each O(1)  if (v->color == WHITE)
                 DFS_Visit(v);
       }
total O(n)  u->color = BLACK;
            time = time+1;
            u->f = time;
}
```

Total time: $\sum deg(u) = 2m = O(m)$

each → $O(1)$

Total time: $O(n) + O(n) + O(m) + O(n) = O(n+m)$

Sparse: $m = O(n)$

Dense: $m = O(n^2)$

Sparse: $O(n)$ ←

Dense: $O(n^2)$ ←

What will be the running time?

Depth-First Search: The 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;
}
```

Running time: O(n^2) because call DFS_Visit on each vertex, and the loop over Adj[] can run as many as |V| times

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Depth-First Search: The 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;
}
```

BUT, there is actually a tighter bound.

How many times will DFS_Visit() actually be called?

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Depth-First Search: The Code

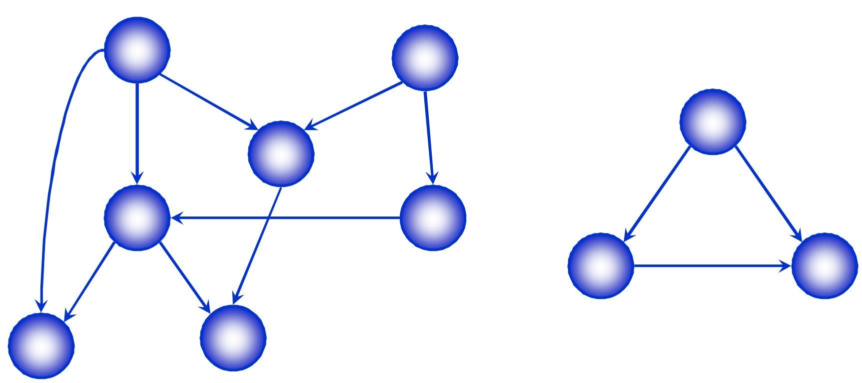
```
DFS(G)
{
    for each vertex u ∈ G->V
    {
        u->color = WHITE;
    }
    time = 0;
    for each vertex u ∈ G->V
    {
        if (u->color == WHITE)
            DFS_Visit(u);
    }
}
```

```
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;
}
```

So, running time of DFS = O(V+E)

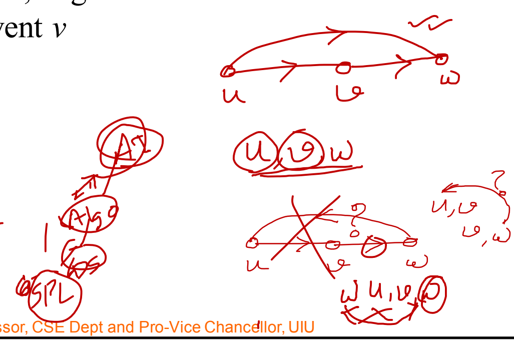
Directed Acyclic Graphs

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



Topological Sort

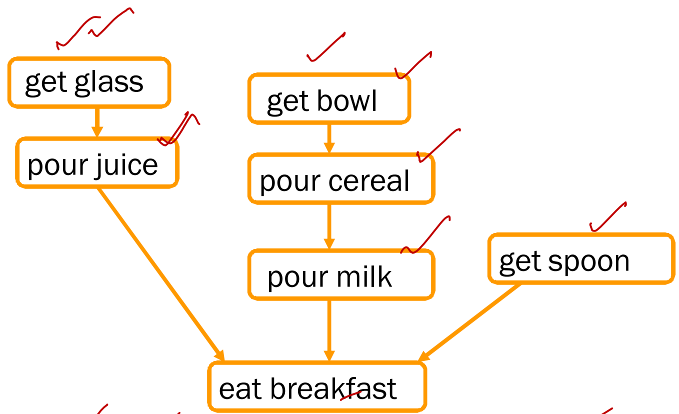
- A *topological sort* of a DAG is
  - a linear ordering of all vertices of the graph G such that vertex u comes before vertex v if (u, v) is an edge in G.
- DAG indicates precedence among events:
  - events are graph vertices, edge from u to v means event u has precedence over event v
- Real-world example:
  - 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.

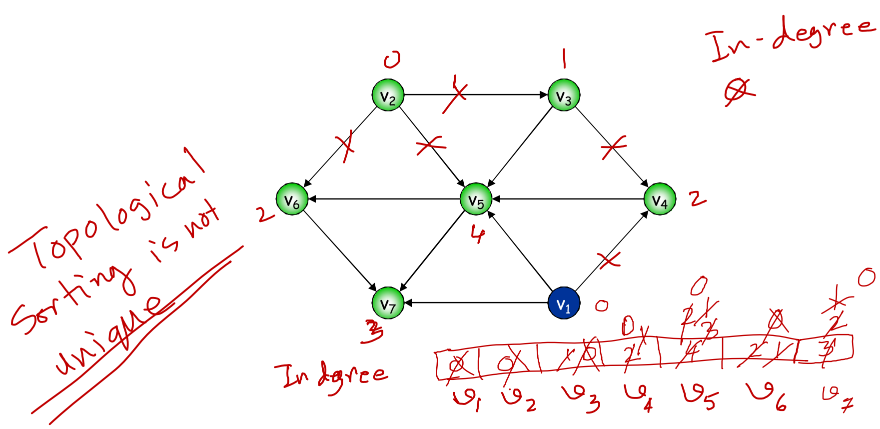
Topological Sort Algorithm

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: $O(V + E)$
- Correctness: Want to prove that
  $(u, v) \in E(G) \Rightarrow f(u) > f(v)$

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU 35

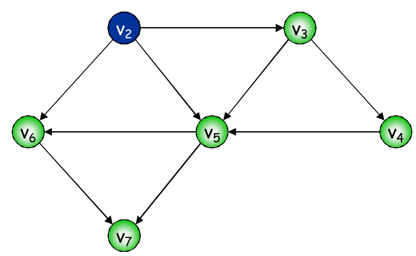
Topological Sort: Using In-degree



Topological order: v_2 v_3 v_1 v_4 v_5 v_6 v_7

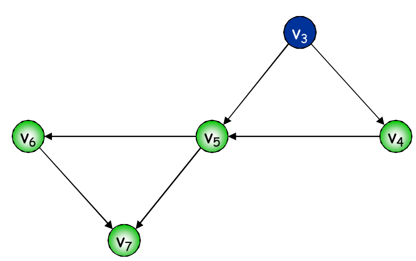
Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Topological Sort: Using In-degree



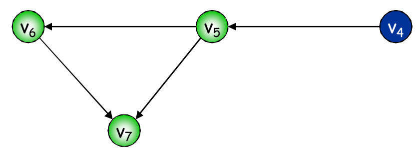
Topological order: v_1

Topological Sort: Using In-degree



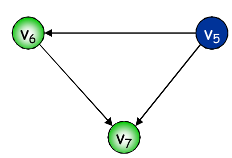
Topological order: v_1, v_2

Topological Sort: Using In-degree



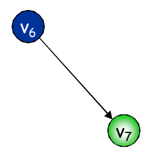
Topological order: v_1, v_2, v_3

Topological Sort: Using In-degree



Topological order: v_1, v_2, v_3, v_4

Topological Sort: Using In-degree

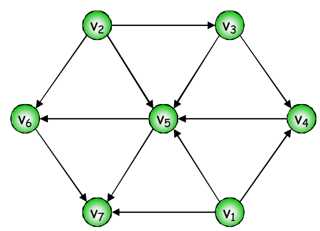


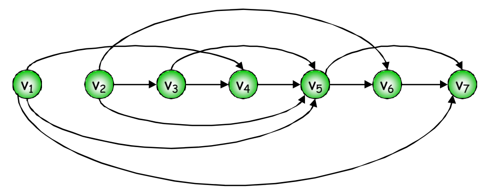
Topological order: v_1, v_2, v_3, v_4, v_5

Topological Sort: Using In-degree

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

Topological Sort: Using In-degree





Topological order: $v_1, v_2, v_3, v_4, v_5, v_6, v_7$.

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Topological Sort: Using In-degree

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

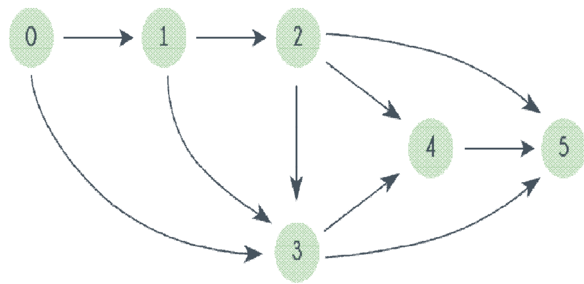
- Increment count of visited nodes by 1;
- Decrease in-degree by 1 for all its neighboring nodes;
- If in-degree of a neighboring node is reduced to zero, then add it to the queue;

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

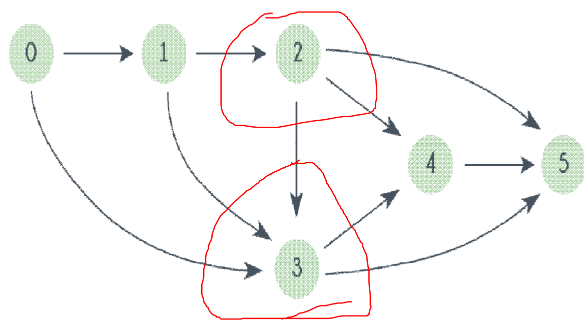
Step 5: If count of visited nodes is **not** equal to the number of nodes in the graph then the topological sort is not possible for the given graph.

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Topological Sort: Using In-degree



Topological Sort: Using In-degree

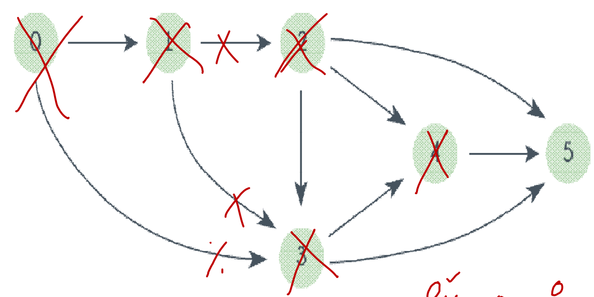


QUEUE : 0

in_degree						
	0	1	~~2~~ 1	~~2~~ 3	2	3
	0	1	~~2~~	3	4	5

T : --

Topological Sort: Using In-degree



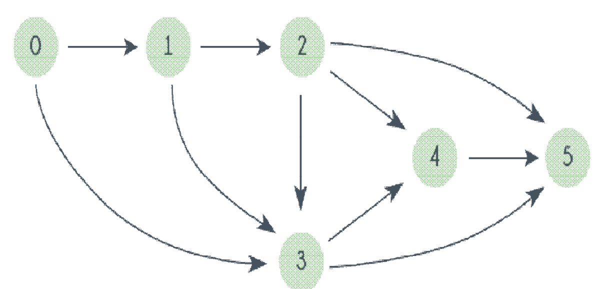
QUEUE : 1

in_degree	0	1	2	3	4	5
	0	~~0~~ 0	~~1~~ 0	~~2~~ 0	~~2~~ 0	~~3~~ 0

T : 0 1 2 3 4 5

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Topological Sort: Using In-degree



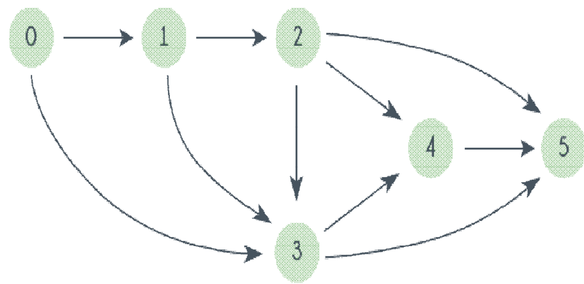
QUEUE : 2

in_degree	0	1	2	3	4	5
	0	0	0	1	1	3

T : 0 1

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Topological Sort: Using In-degree



QUEUE : 3

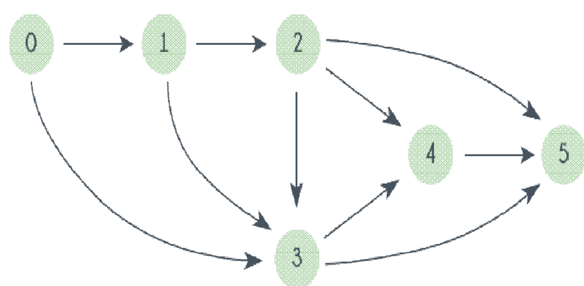
in_degree

0	1	2	3	4	5
0	0	0	0	1	2

T : 0 1 2

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Topological Sort: Using In-degree



QUEUE : 4

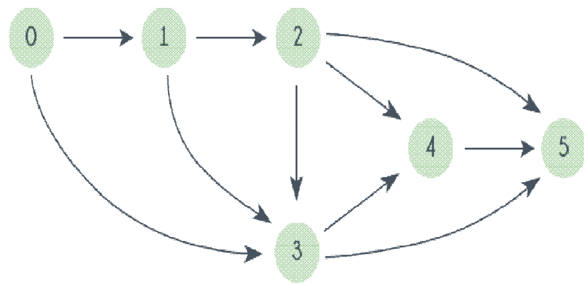
in_degree

0	1	2	3	4	5
0	0	0	0	0	1

T : 0 1 2 3

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Topological Sort: Using In-degree



QUEUE : 5

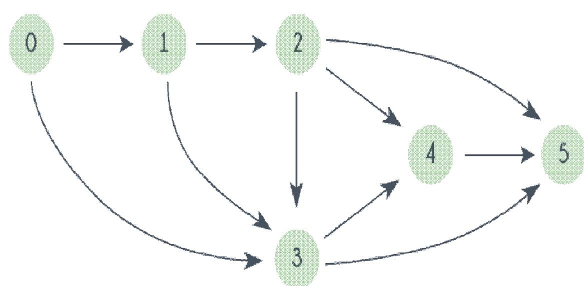
in_degree

0	1	2	3	4	5
0	0	0	0	0	0

T : 0 1 2 3 4

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU

Topological Sort: Using In-degree



QUEUE : --

in_degree

0	1	2	3	4	5
0	0	0	0	0	0

T : 0 1 2 3 4 5

Dr. Md. Abul Kashem Mia, Professor, CSE Dept and Pro-Vice Chancellor, UIU