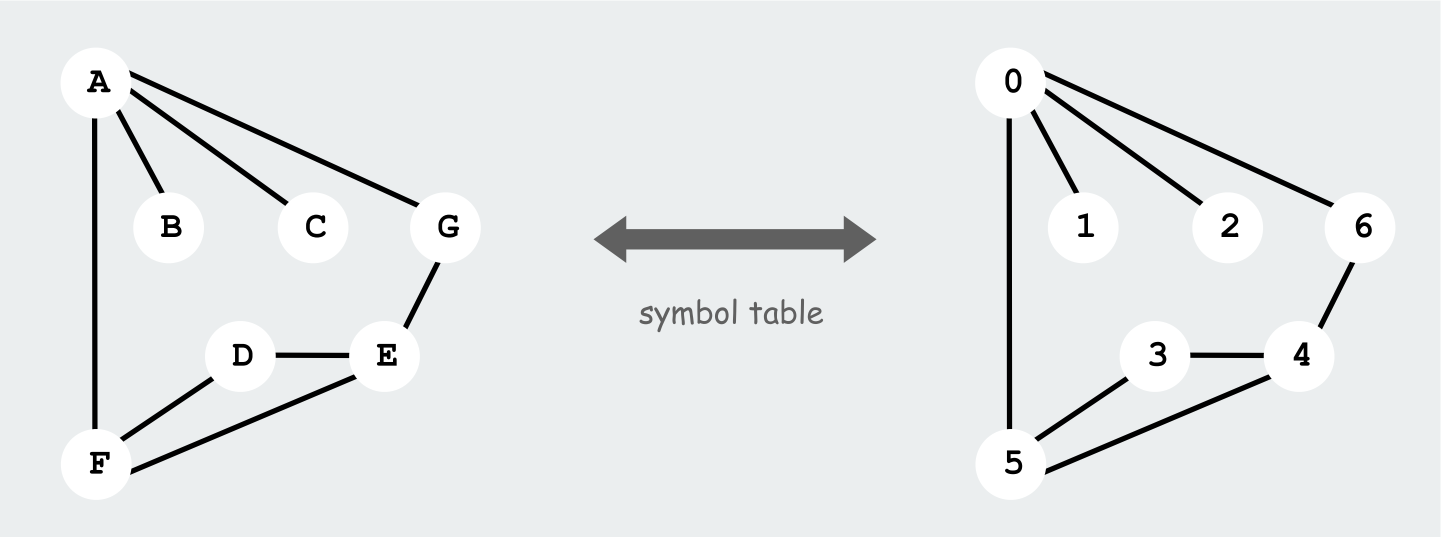
Graph Traversal

M. Elif Karslıgil

Undirected Graph Representation

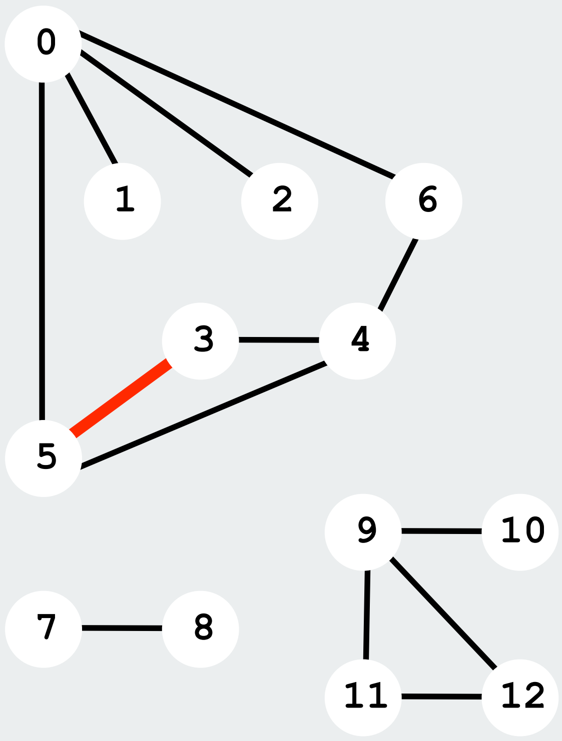
- Computer : use integers between 0 and V-1
- Real world: convert between names and integers with symbol table



Undirected Graph - Adjacency Matrix

Maintain a two-dimensional $V \times V$ boolean array.

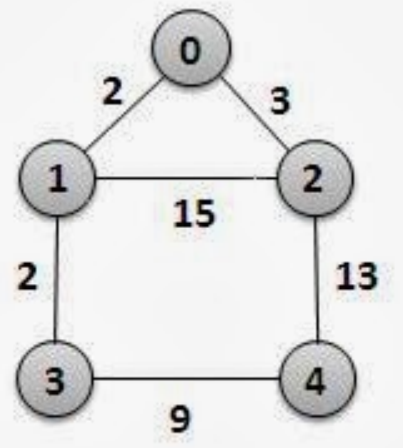
For each edge **v–w** in graph: `adj[v][w] = adj[w][v] = true.`



two entries for each edge

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

Undirected Weighted Graph - Adjacency Matrix

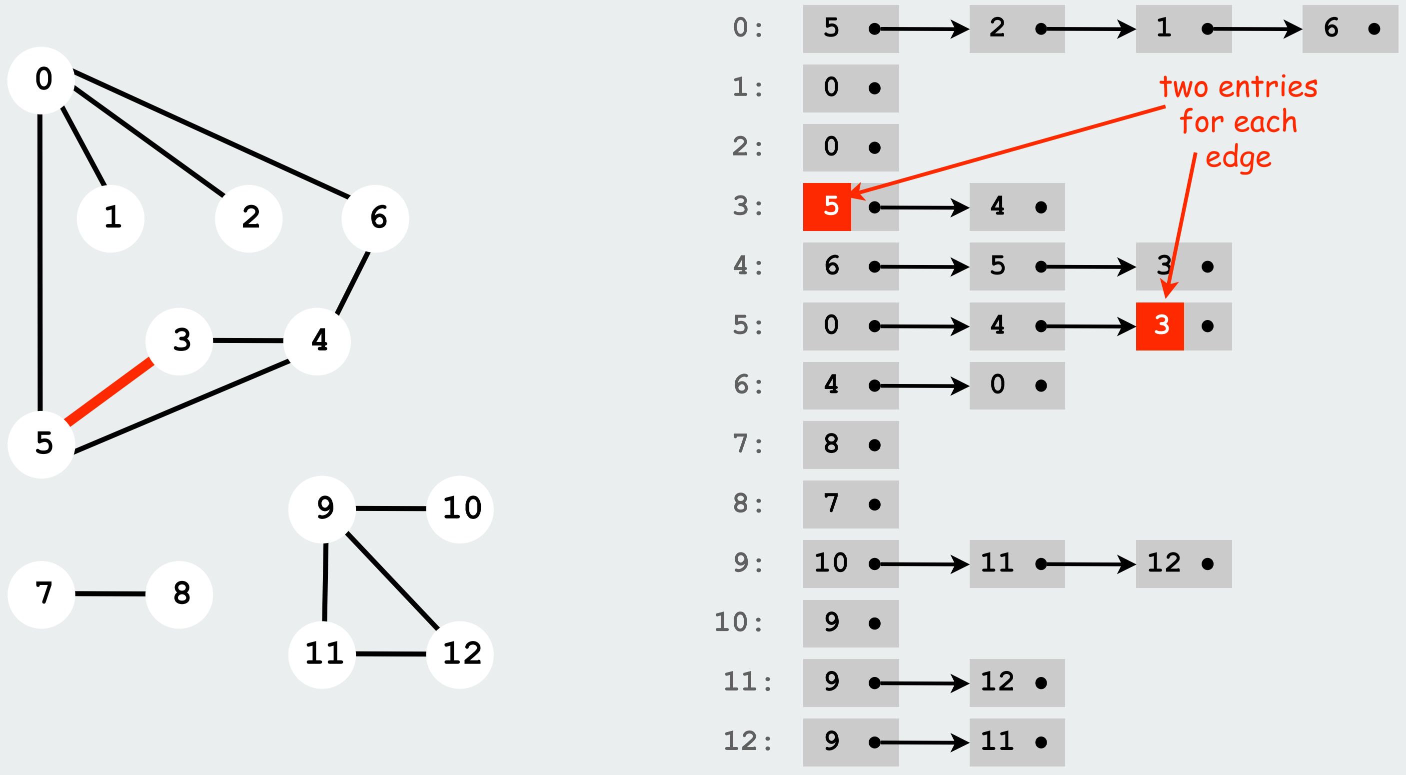


	0	1	2	3	4
0	0	2	3	0	0
1	2	0	15	2	0
2	3	15	0	0	13
3	0	2	0	0	9
4	0	0	13	9	0

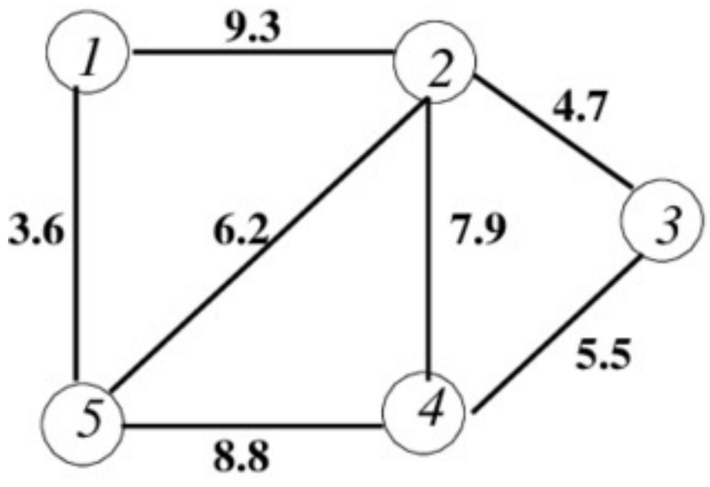
Adjacency Matrix Representation of Weighted Graph

Undirected Graph - Adjacency List

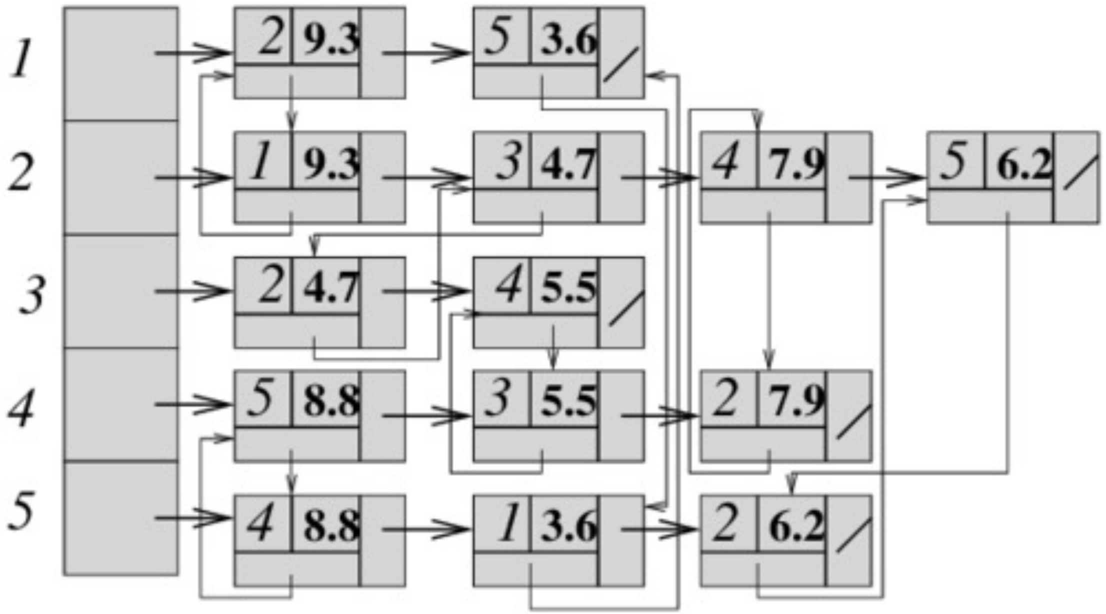
Maintain vertex-indexed array of lists (implementation omitted)



Undirected Weighted Graph - Adjacency List



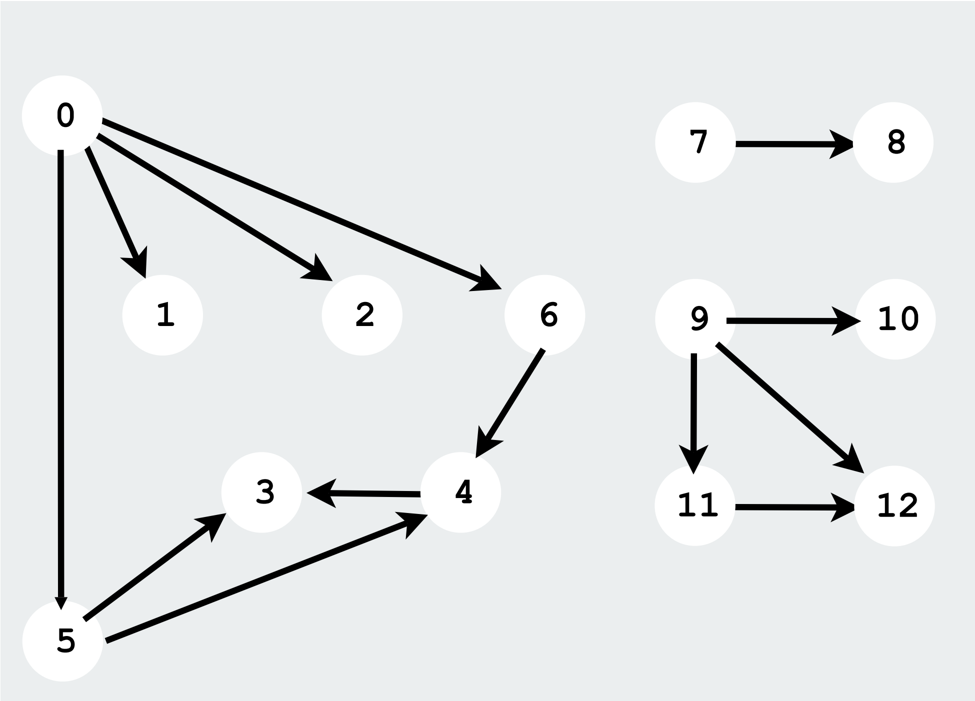
(a)



(b)

Directed Graphs

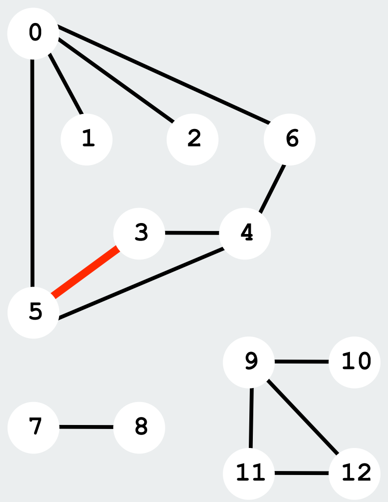
- A directed graph is a set of vertices and a collection of directed edges.
- Edges are one way



Graph Traversal

Maintain a two-dimensional $V \times V$ boolean array.

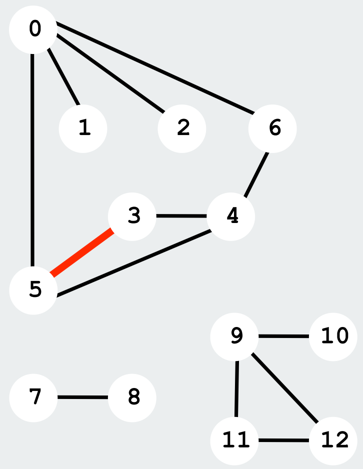
For each edge `v–w` in graph: `adj[v][w] = adj[w][v] = true.`



two entries for each edge

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

Maintain vertex-indexed array of lists (implementation omitted)



two entries for each edge

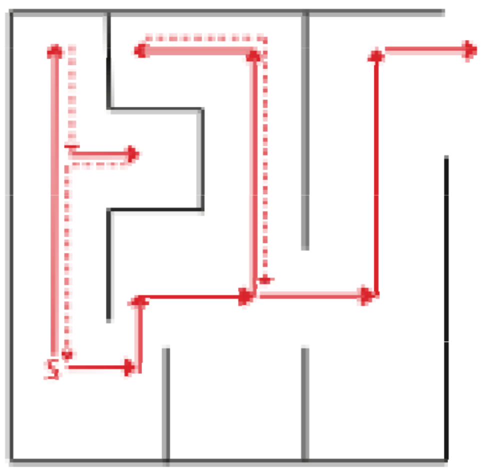
```
 0: 5 → 2 → 1 → 6
 1: 0
 2: 0
 3: 5 → 4
 4: 6 → 5 → 3
 5: 0 → 4 → 3
 6: 4 → 0
 7: 8
 8: 7
 9: 10 → 11 → 12
10: 9
11: 9 → 12
12: 9 → 11
```

Graph Traversal

- Visit every edge and node in the graph in a systematic way
  - Depth-first search. Put unvisited vertices on a stack.
  - Breadth-first search. Put unvisited vertices on a queue.

Graph Traversal – Depth First Seach (DFS)

- Like exploring a maze
- From current vertex, move to another
- Until you get stuck
- Then backtrack till you find a new place to explore
- e.g “left-hand” rule



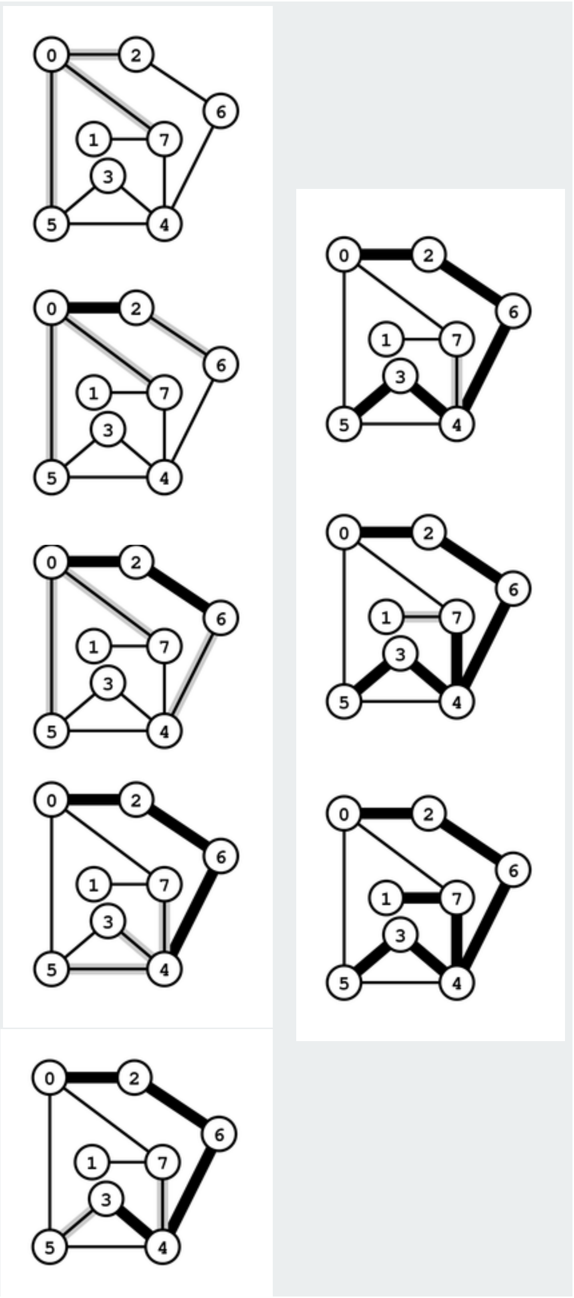
Trémaux Maze Exploration

- Unroll a ball of string behind you.
- Mark each visited intersection by turning on a light.
- Mark each visited passage by opening a do

First use? Theseus entered labyrinth to kill the monstrous Minotaur;
Ariadne held ball of string.

Graph Traversal – Depth First Seach (DFS)

- Goal. Systematically search through a graph.
- Idea. Mimic maze exploration.
- Typical applications.
  - find all vertices connected to a given **s**
  - find a path from **s** to **t**

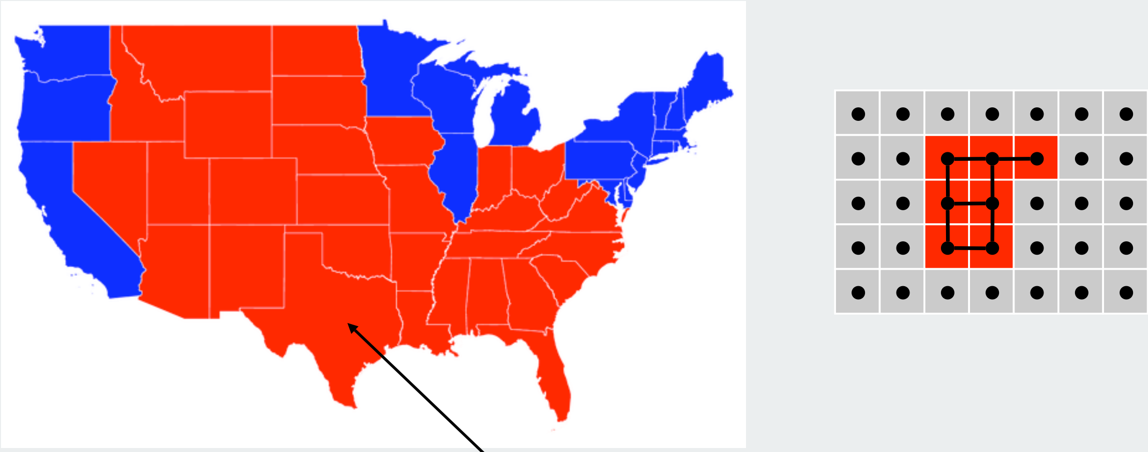


Flood Fill

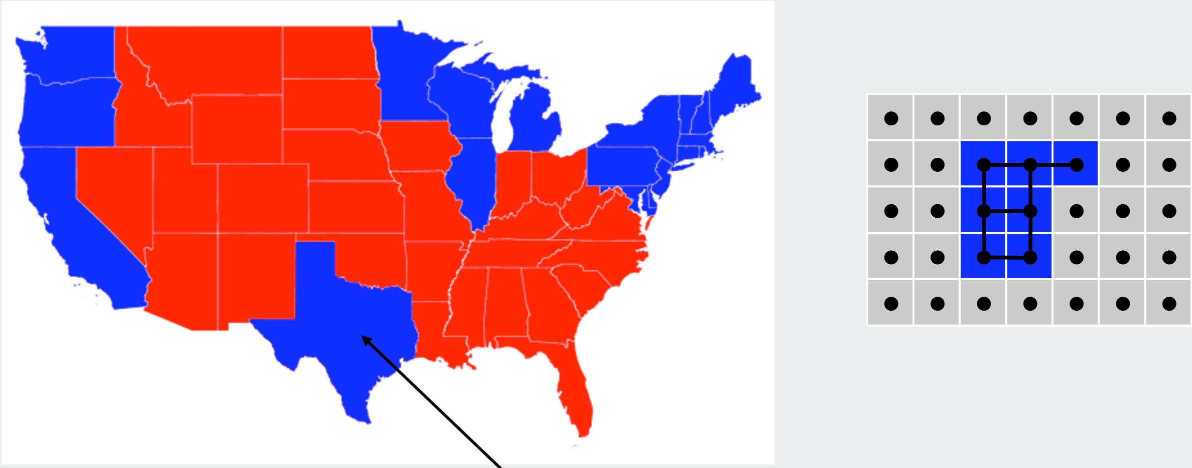
Change color of entire blob of neighboring red pixels to blue.

Build a grid graph

- vertex: pixel.
- edge: between two adjacent lime pixels.
- blob: all pixels connected to given pixel.



recolor red blob to blue

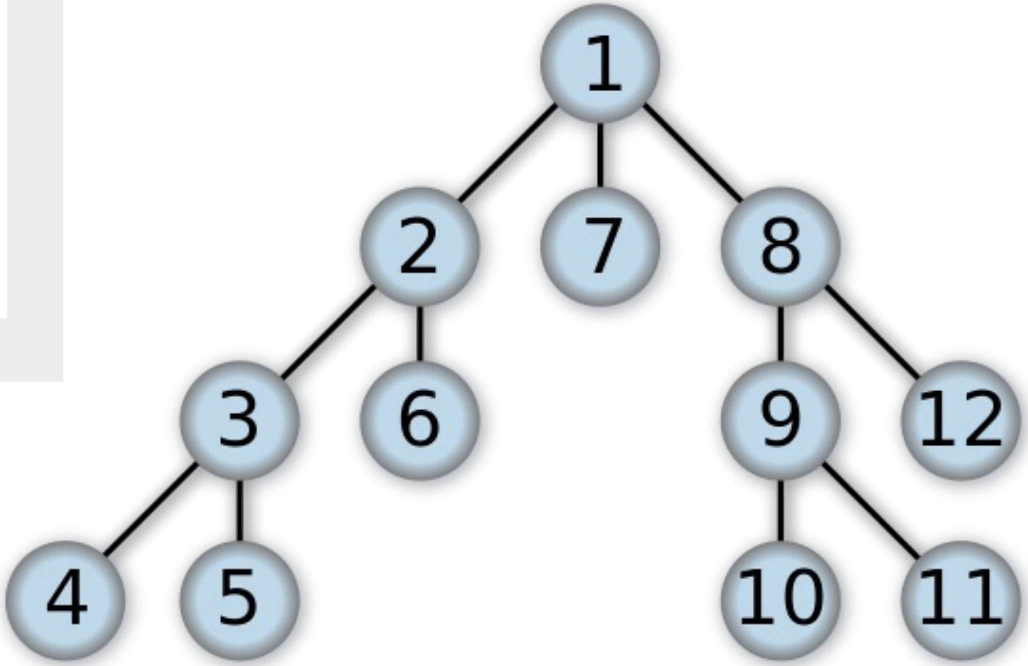


recolor red blob to blue

Graph Traversal – Depth First Seach (DFS)

DFS (to visit a vertex s)
Mark s as visited.
Visit all unmarked vertices v adjacent to s .

recursive

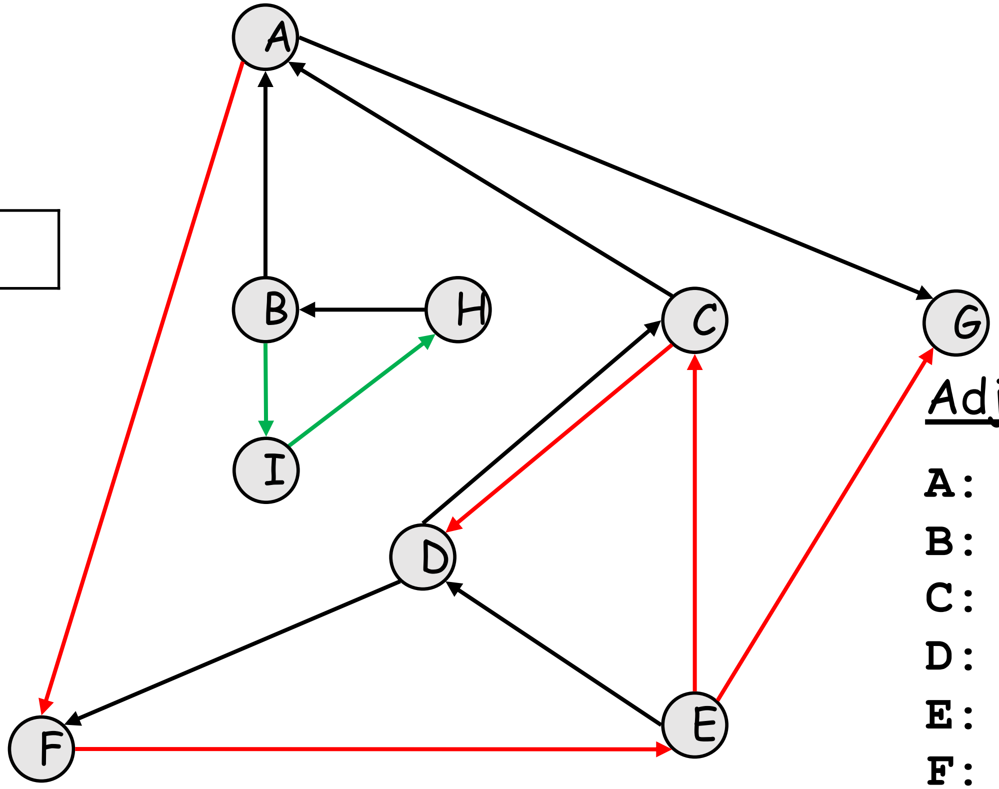


DFS Traversal Example

visited nodes array:

A	B	C	D	E	F	G	H	I

How many connected components does the graph have ?
Starting Point is A



Adjacency Lists

A: F G
B: A I
C: A D
D: C F
E: C D G
F: E
G:
H: B
I: H

DFS (to visit a vertex s)

Mark s as visited.

Visit all unmarked vertices v adjacent to s.

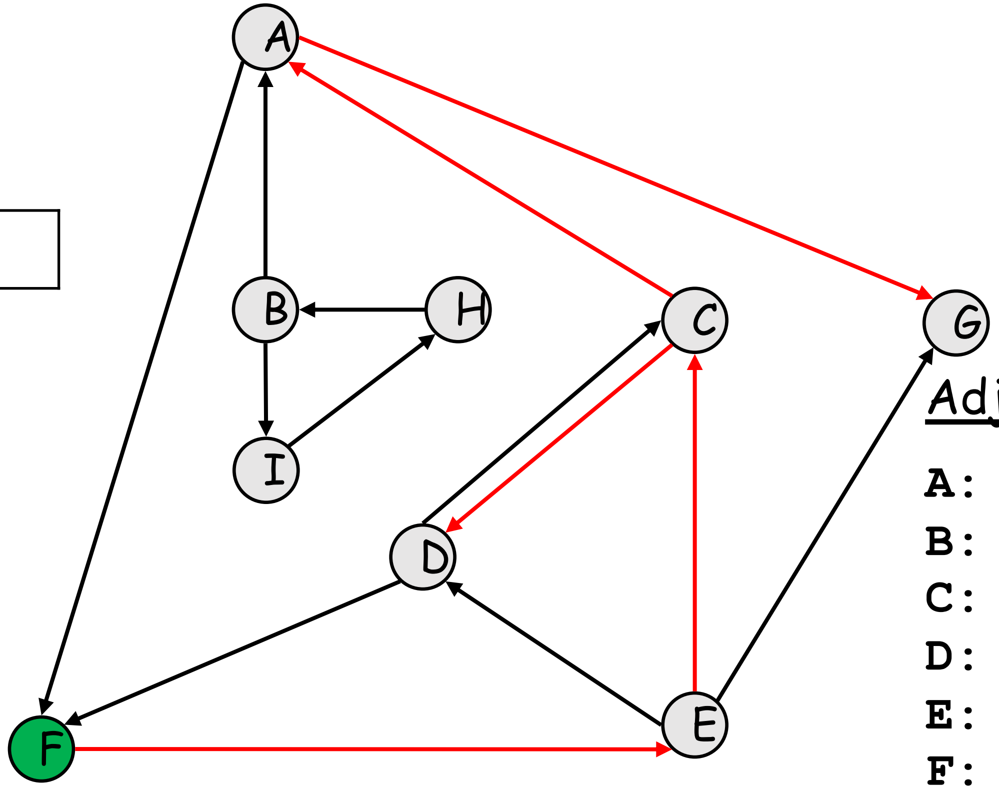
recursive

DFS Traversal Example

visited :

A	B	C	D	E	F	G	H	I
1	0	1	1	1	1	1	0	0

Is there any path from F to I



Adjacency Lists

A: F G
B: A I
C: A D
D: C F
E: C D G
F: E
G:
H: B
I: H

DFS (to visit a vertex s)

Mark s as visited.

Visit all unmarked vertices v adjacent to s.

recursive

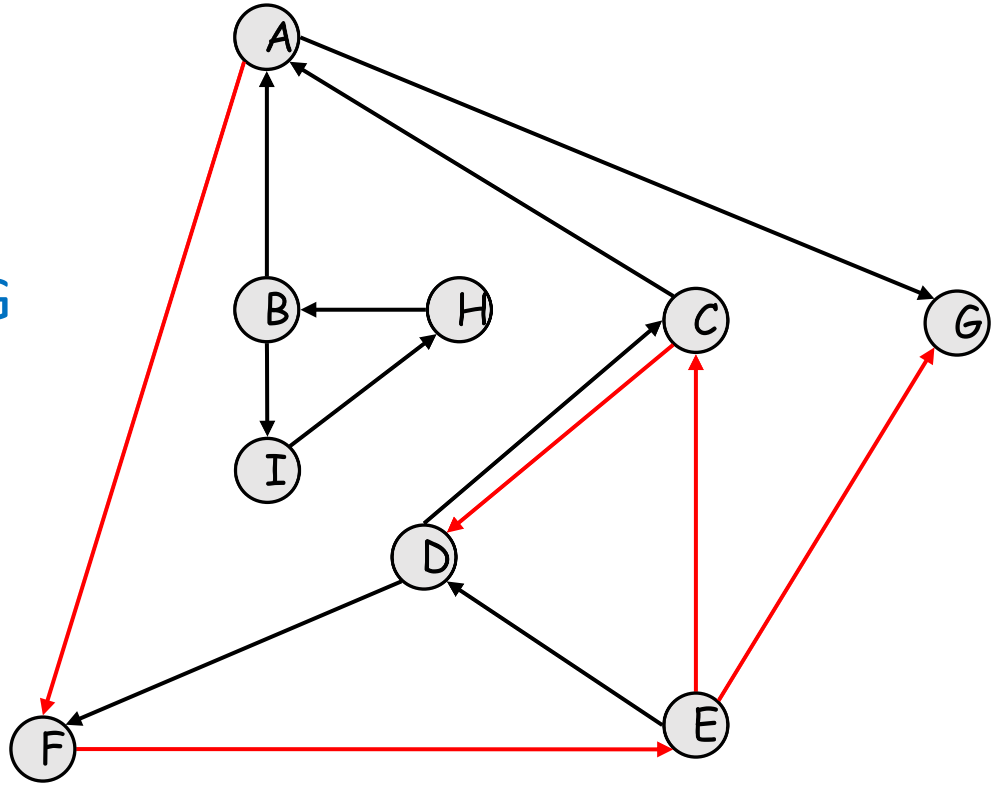
Graph Traversal – Depth First Seach (DFS)

- **Recursive DFS:**

DFS(G, s)
{
  mark s as visited
  for all neighbors w of s in Graph G
    if w is not visited
      DFS(G,w)
}

Is there a path from A to D ?
A F E C D

Is there a path from A to B ?
A F E C D G



Adjacency Lists

A:	F G
B:	A I
C:	A D
D:	C F
E:	C D G
F:	E
G:	
H:	B
I:	H

Depth First Seach (DFS)

- To visit a node v:
  - mark it as visited
  - recursively visit all unmarked nodes w adjacent to v
- To traverse a Graph G:
  - initialize all nodes as unmarked
  - visit each unmarked node
- Running time.
  - O(E) since each edge examined at most twice
  - usually less than V to find paths in real graphs

Graph Traversal – Depth First Seach (DFS)

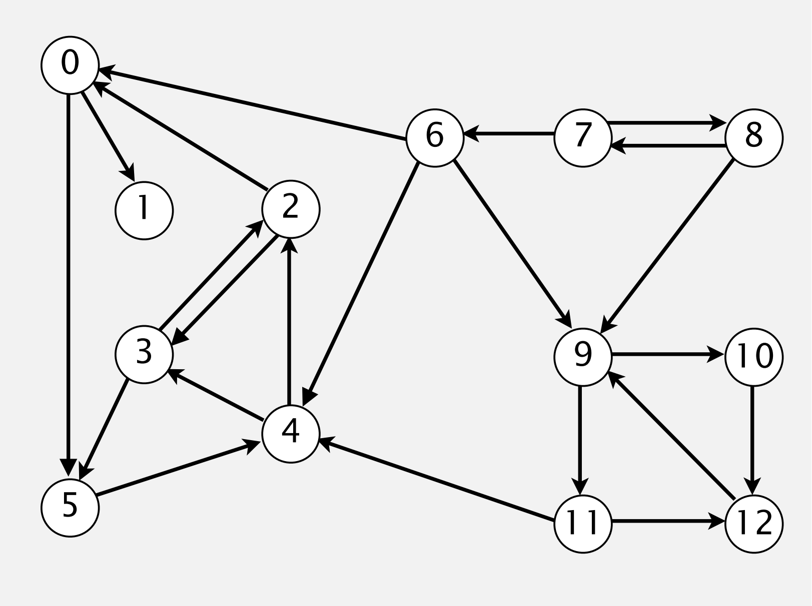
- Non-Recursive DFS: using STACK

```
DFS(G, s)
{
     Stack.push(s)
    mark s as visited
    while(Stack is not empty)
    {
            v =  Stack.pop();
            for all neighbors w of v in Graph G
                    if w is not visited
                    {
                            STACK.push(w)
                            mark w as visited
                    }
    }
}
```

DFS Traversal Example

<u>Adjacency Lists</u>

- 0 : 1 5
- 1 : -
- 2 : 0 3
- 3 : 2 5
- 4: 2 3
- 5 : 4
- 6 : 0 4 9
- 7 : 6 8
- 8 : 7 9
- 9 : 10 11.
- 10 : 12
- 11 : 4 12
- 12 : 9

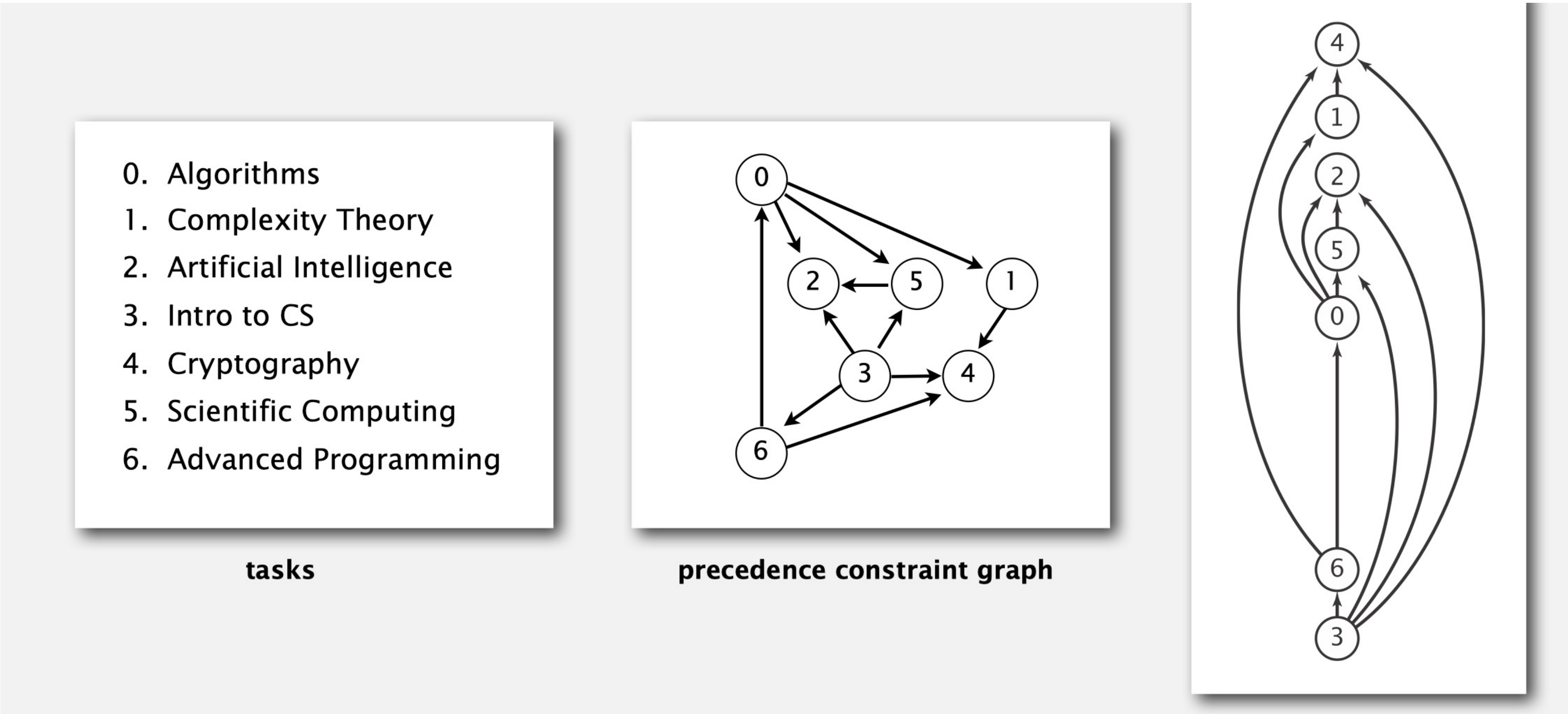


Is there a path from node 0 to node 9

DFS Traversal from node 0 : 0 1 5 4 2 3

Toplogical Sort

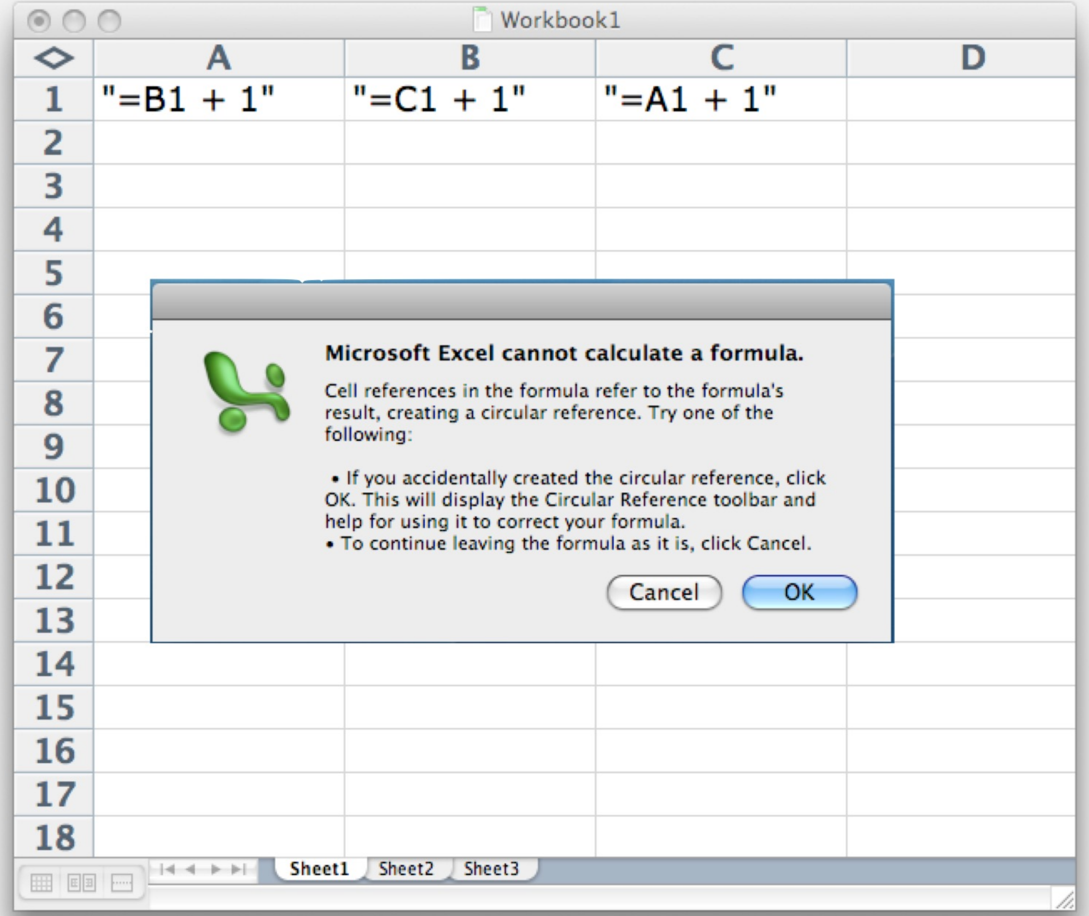
Goal: Given a set of tasks to be completed with precedence constraints, in which order should we schedule the tasks?



tasks **precedence constraint graph** **feasible schedule**

Directed cycle detection application: spreadsheet recalculation

Microsoft Excel does cycle detection (and has a circular reference toolbar!)



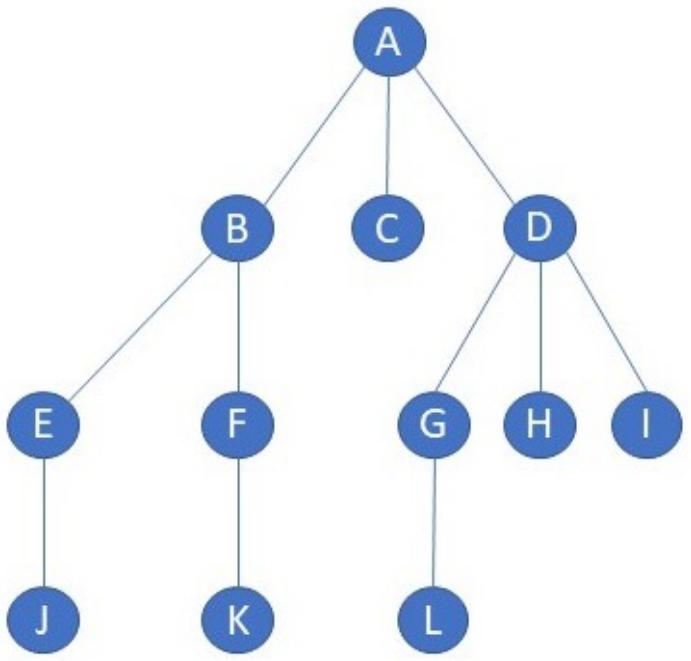
Graph Traversal – Breadth First Search (BFS)

BFS (from source vertex s)

Put s onto a FIFO queue.

Repeat until the queue is empty:

- remove the least recently added vertex **v**
- add each of **v**'s unvisited neighbors to the queue, and mark them as visited.



Breadth First Search (BFS)

Graph Traversal – Breadth First Search (BFS)

BFS (from source vertex s)

Put s onto a FIFO queue.

Repeat until the queue is empty:

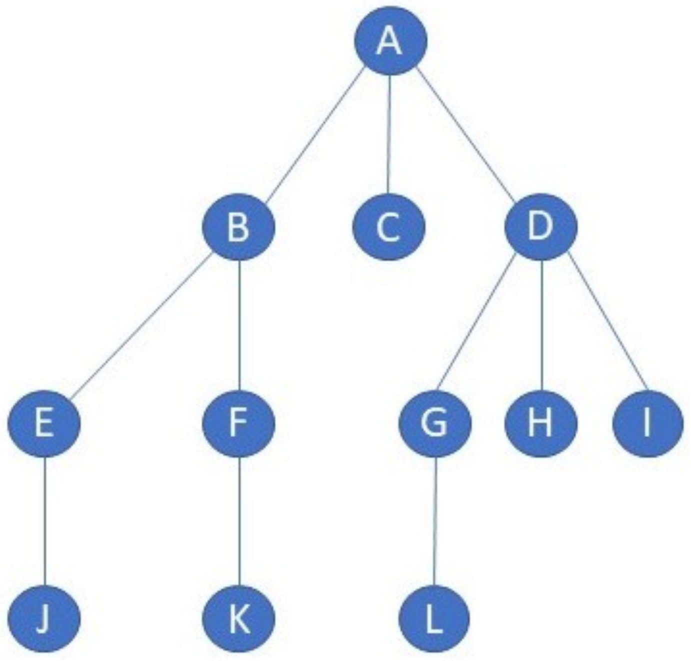
- remove the least recently added vertex v
- add each of v's unvisited neighbors to the queue, and mark them as visited.

Visited :

A	B	C	D	E	F	G	H	I	J.	K	L
1	1	1	1	1	1	1	1	1	1	0	0

Path :

A	B	C	D	E	F	G	H	I	J.	K	L
0	1	1	1	2	2	2	2	2	3	3	0



Breadth First Search (BFS)

Shortest Path from A to K

Queue : A B C D E F G H I J **K**

BFS Traversal Example

visited :

A	B	C	D	E	F	G	H	I
1	0	1	1	1	1	1	0	0

BFS Traversal :

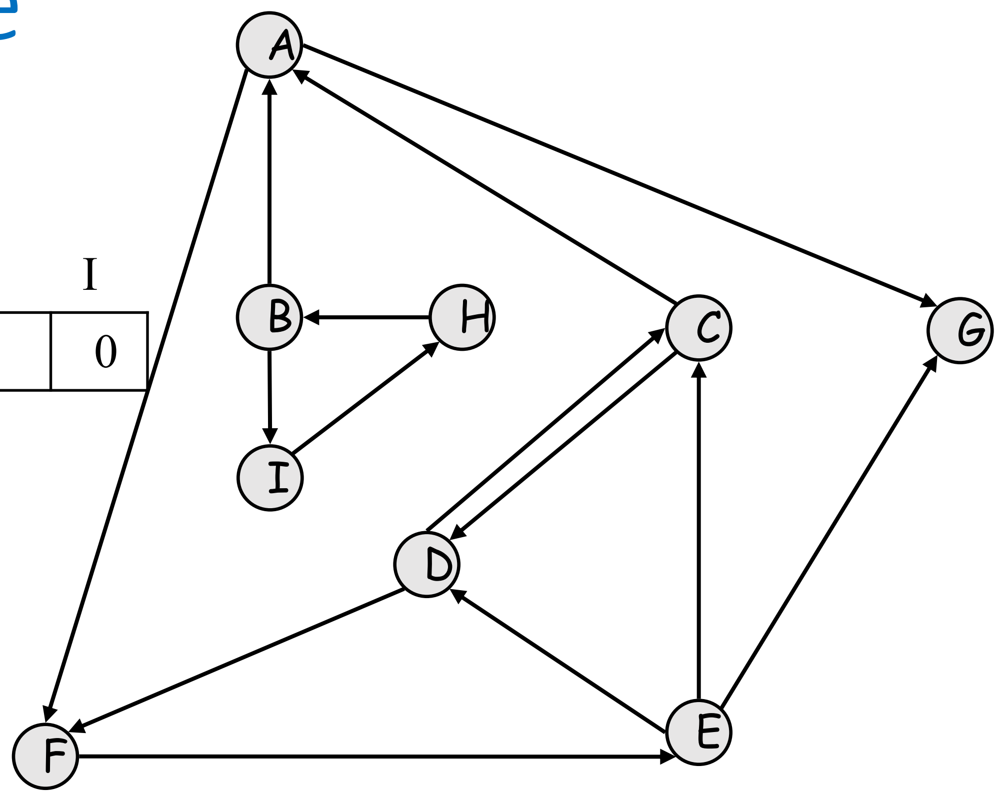
A F G E C D

BFS (from source vertex s)

Put s onto a FIFO queue.

Repeat until the queue is empty:

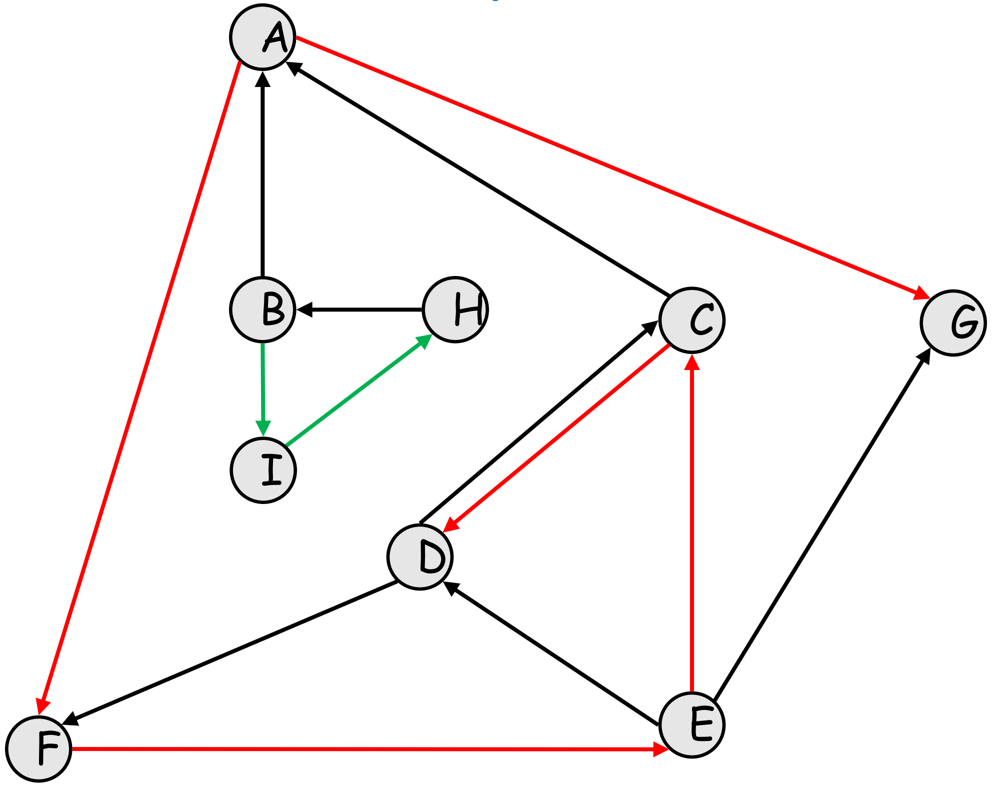
- remove the least recently added vertex **v**
- add each of **v**'s unvisited neighbors to the queue, and mark them as visited.



Adjacency Lists

A: F G
B: A I
C: A D
D: C F
E: C D G
F: E
G:
H: B
I: H

BFS Traversal Example



Adjacency Lists

```
A:   F G
B:   A I
C:   A D
D:   C F
E:   C D G
F:   E
G:
H:   B
I:   H
```

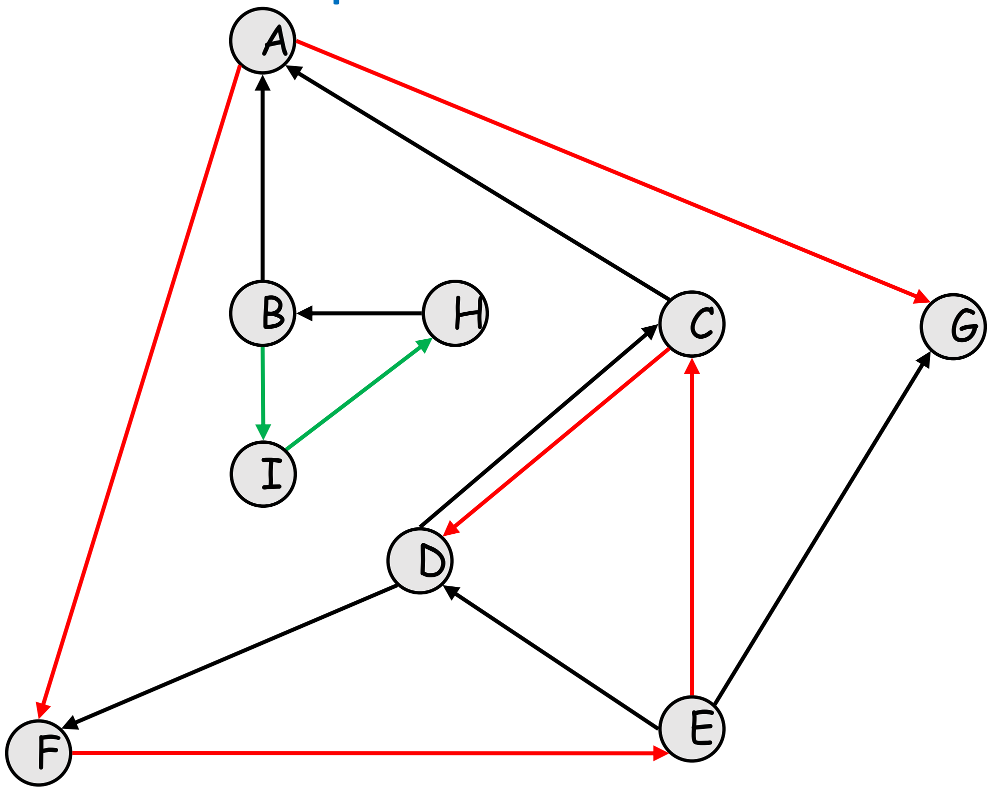
A F G E C D   B I H

Graph Traversal – Breadth First Search (BFS)

- BFS : using QUEUE

```
BFS(G, s)
{
     Queue.enqueue(s)
    mark s as visited
    while(Queue is not empty)
    {
            v =  Queue.dequeue();
            for all neighbors w of v in Graph G
                    if w is not visited
                    {
                          Queue.enque(w)
                         mark w as visited
                    }
    }
}
```

Traversal Example



Adjacency Lists

A:	F G
B:	A I
C:	A D
D:	C F
E:	C D G
F:	E
G:	
H:	B
I:	H

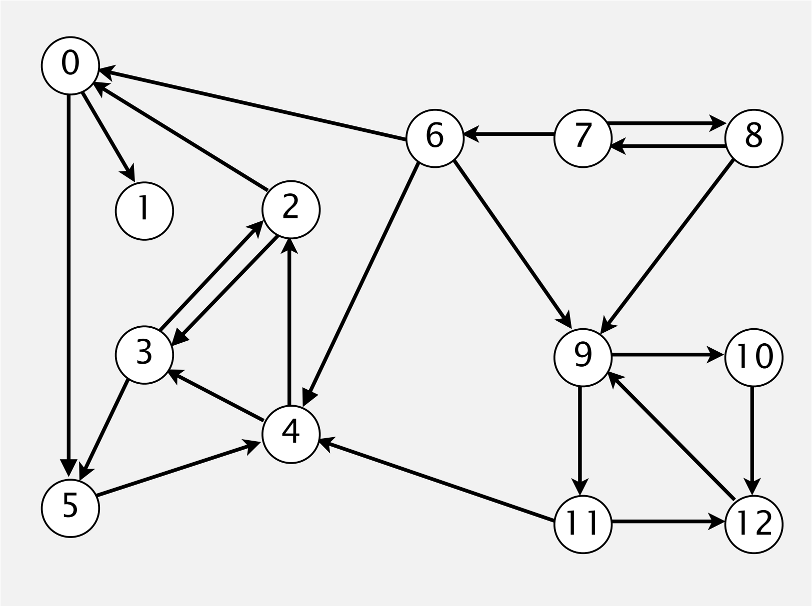
BFS : A F G E C D B I H

DFS : A F E C D G B I H

BFS Traversal Example

<u>Adjacency Lists</u>

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



DFS : 6 0 1 5 4 2 3 9 10 12 11

BFS : **6 0 4 9 1 5 2 3 10 11 12**

Graph Search Problems

- Is there a path from s to t?
- Find shortest path (fewest edges) from s to t
- Is there a cycle in the graph ?
- How many connected compenents exist?