CSE 230: Data Structures

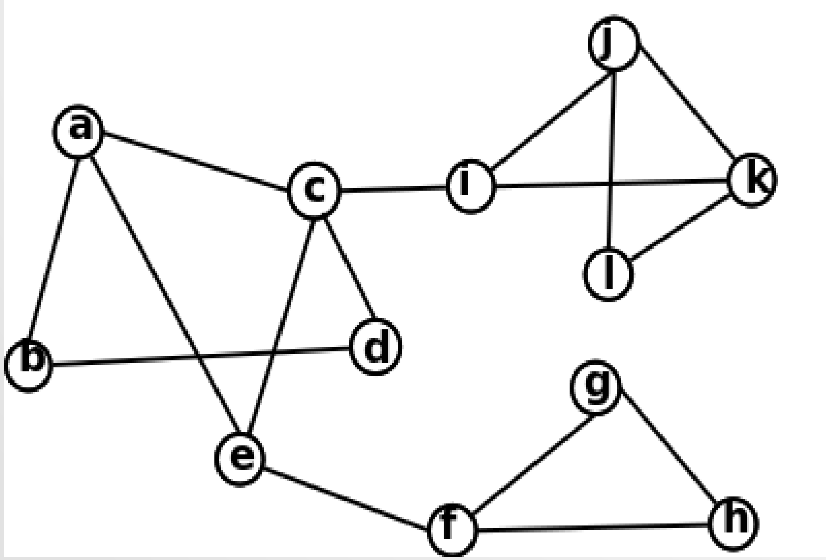
Lecture 10: Graphs

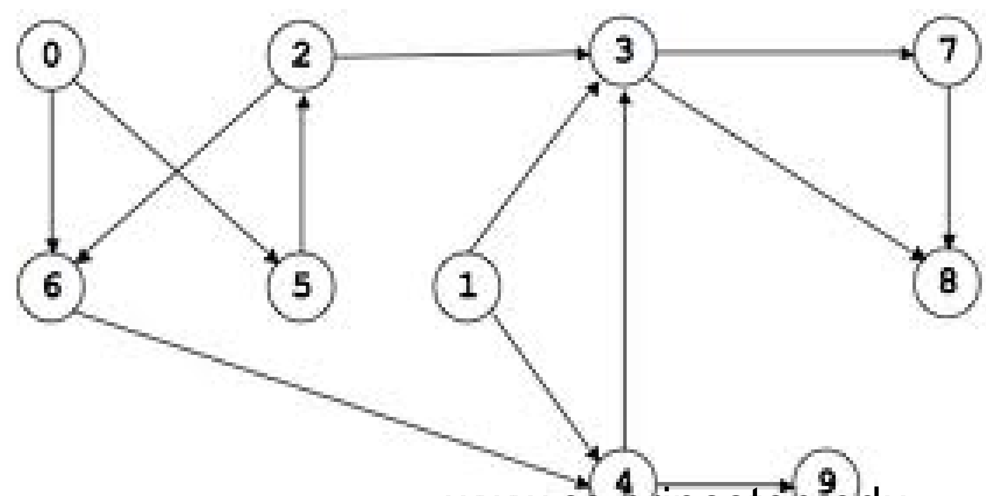
Dr. Vidhya Balasubramanian

Amrita School of Engineering
Amrita Vishwa Vidyapeetham

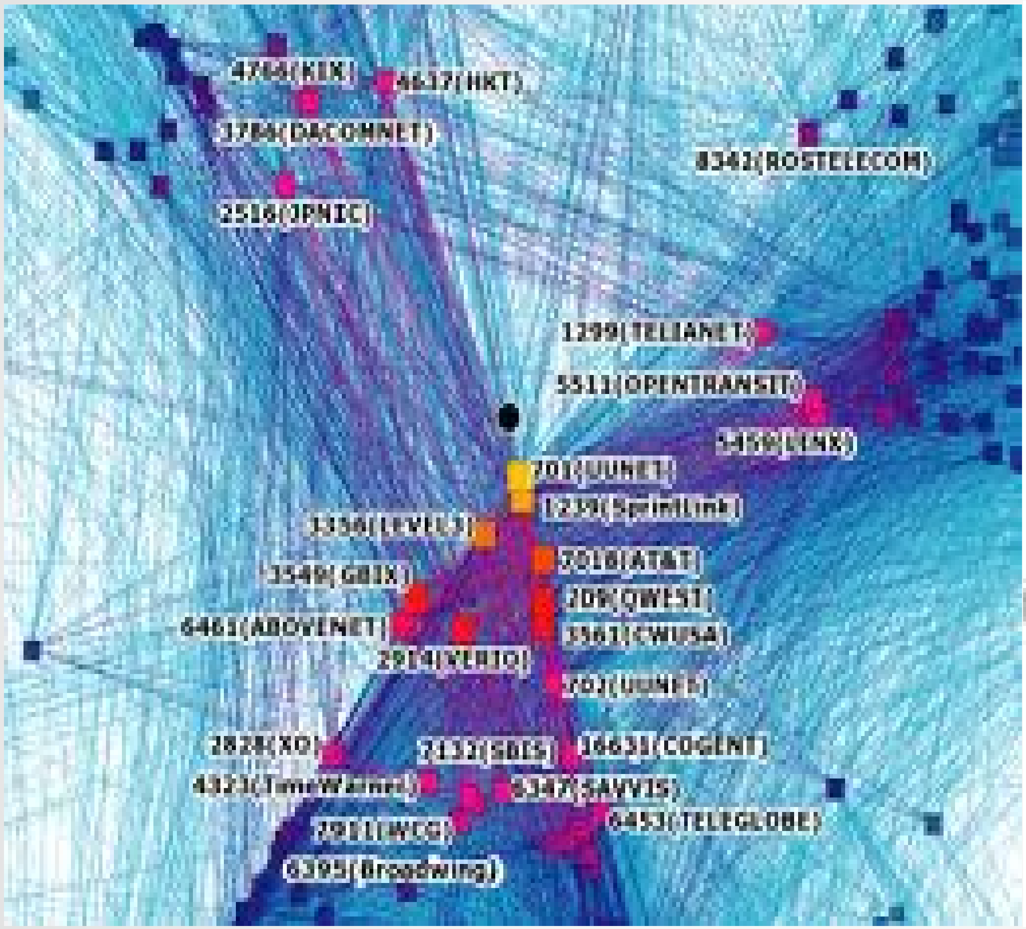
Graphs

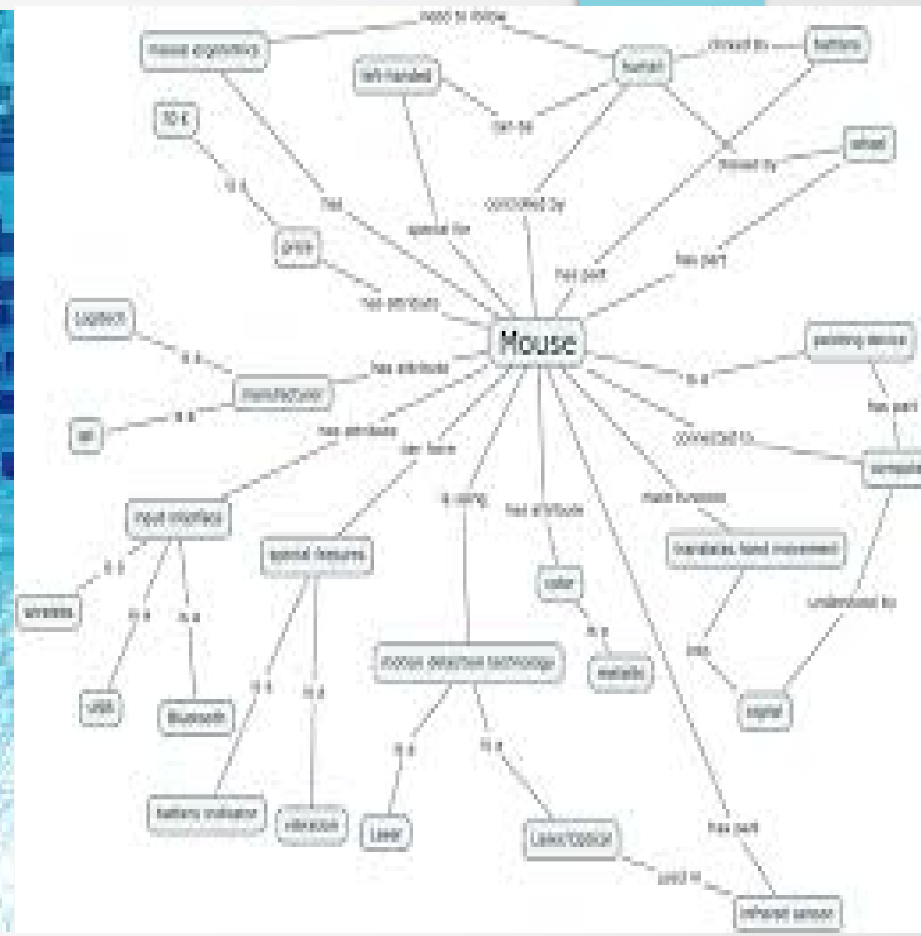
- A graph G= (V,E) is a set of vertices V, and a collection of edges E which is a subset of VxV
- A way of representing connections between pairs of objects from some set V
  - Edges can be either directed or undirected





Examples

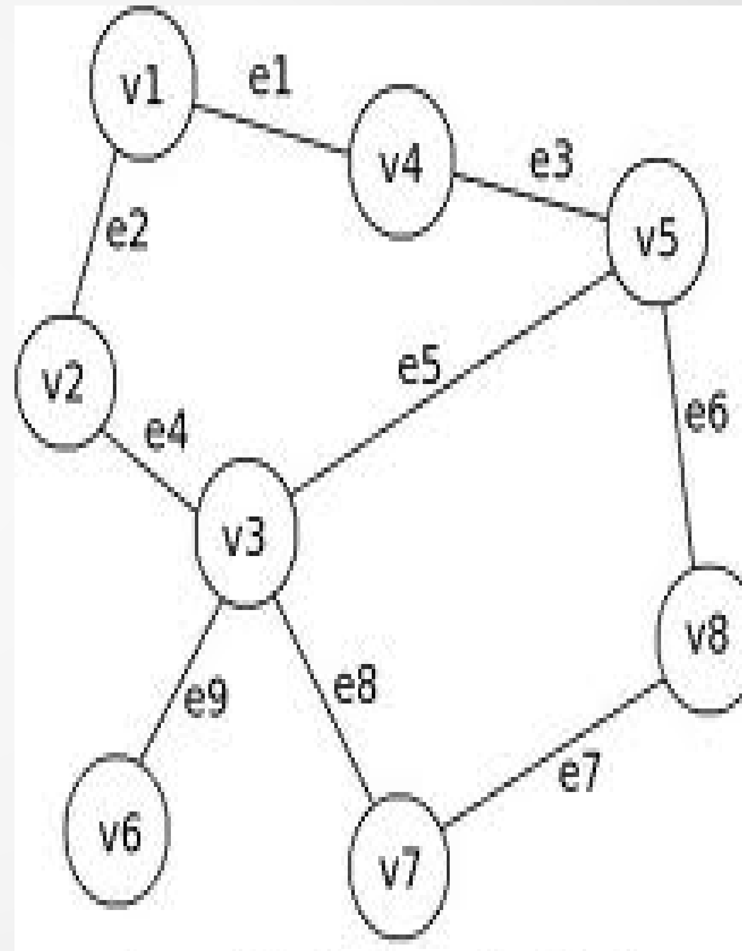




www.visualcomplexity.com http://t1065550.files.wordpress.com/2012/02/mouse.jpg

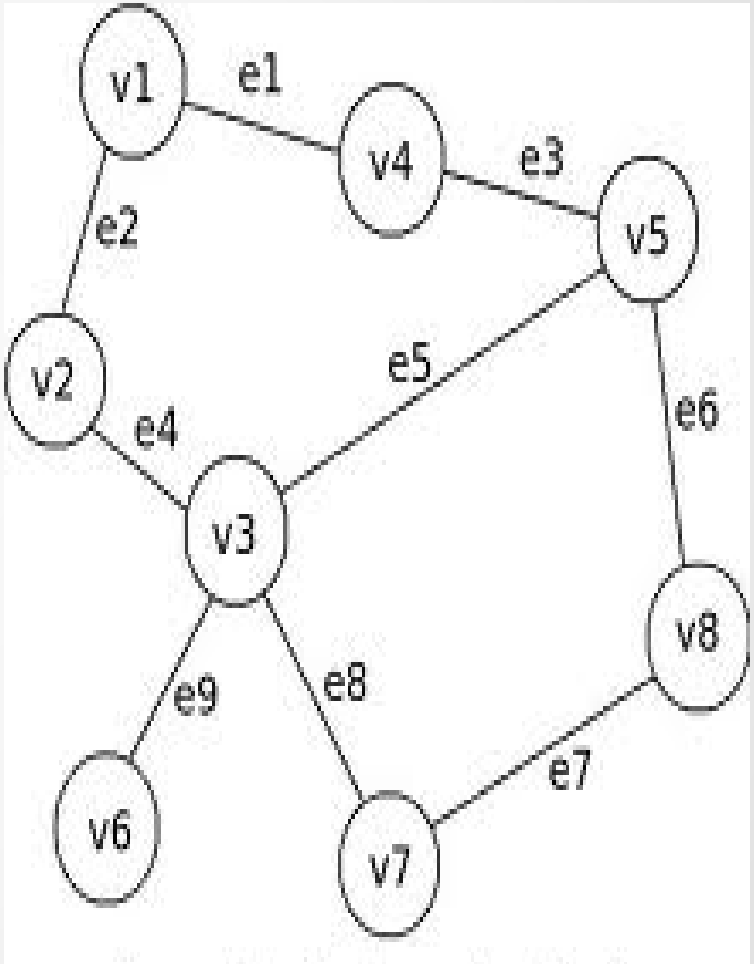
Definitions

- End vertices are end points of edges
  - v1 and v2 end vertices of e2
- Edges incident on a vertex
  - e9 incident on v6
- Adjacent vertices (end pts of an edge
  - v1, v2 adjacent
- Degree of vertex is number of edges incident on it
- Parallel edges connect same set of vertices
- Self loop, edge connecting same vertex



Definitions

- Path
  - Sequence of alternating vertices and edges
  - Eg: P1=(v1,v2,v3,v6)
  - Simple path
    - Vertices and edges are distinct
      - e.g P1
    - v1,v2,v3,v7,v3,v6, not simple
- Cycle
  - Path that ends at start node
  - v2,v1,v4,v5,v3,v2

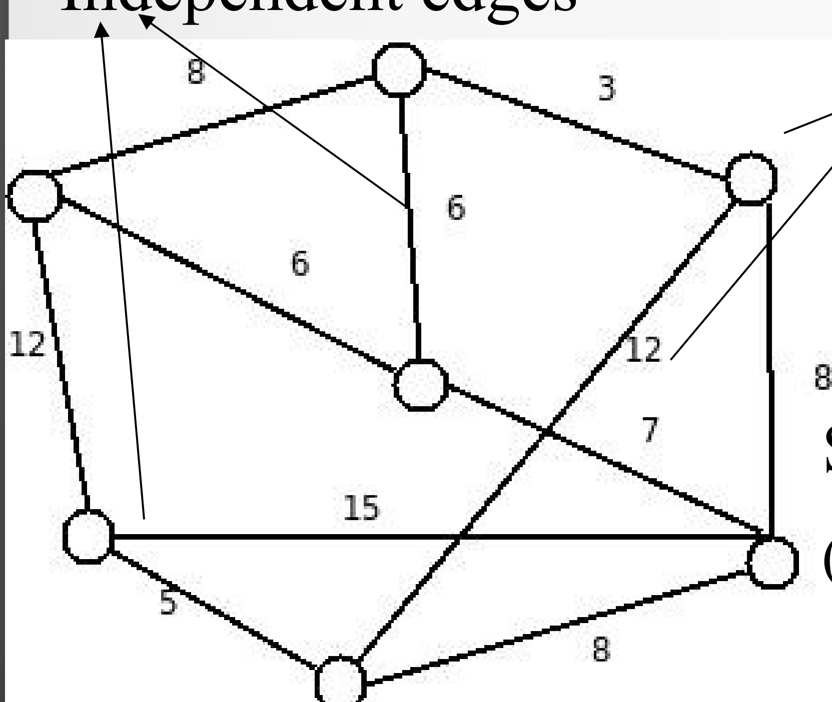


Definitions

- Independent edges: don't have common vertices
- Complete graph: Every node adjacent to each other
- Simple Graph: Has no loops or parallel edges

Independent edges

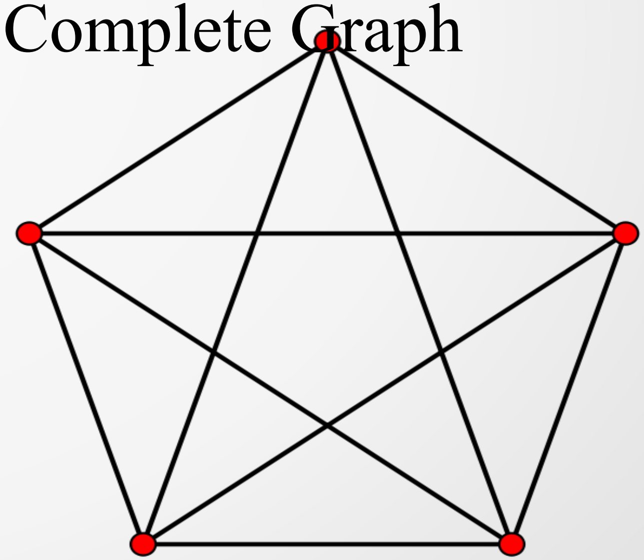
adjacent



Simple Graph

(no loops, parallel edges)

Complete Graph

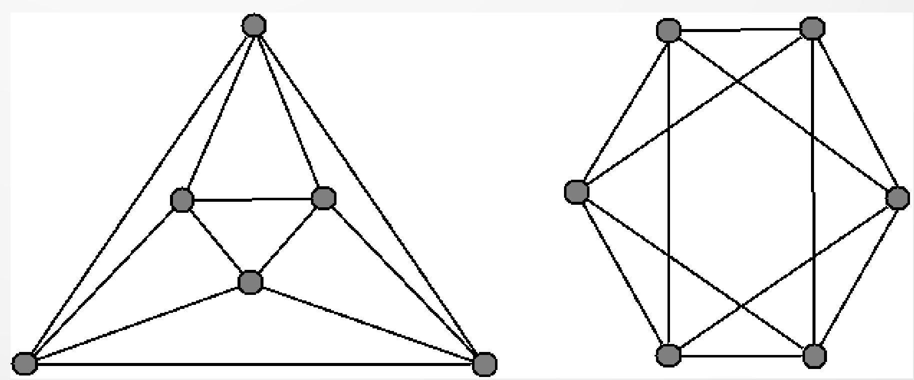


Definitions contd

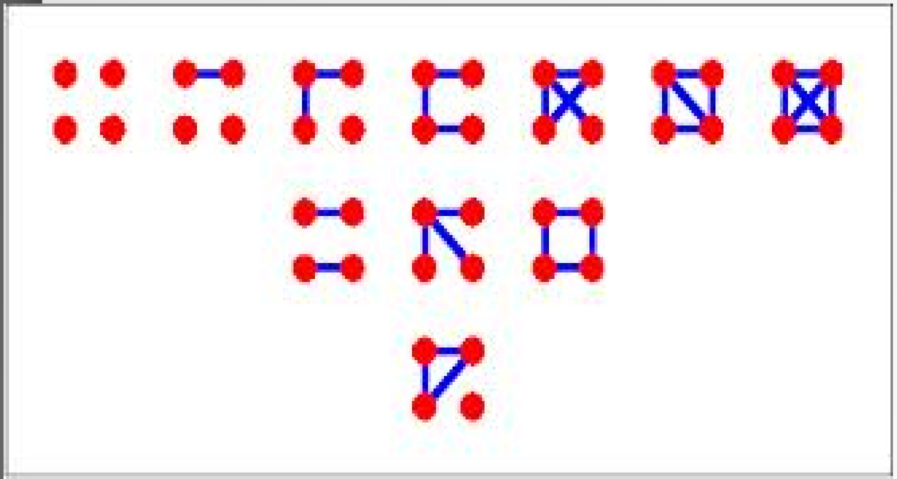
- Degree: Number of edges incident at a vertex
  - Min degree of G --> $\delta(G) = \min \{d(v) | v \in V\}$
- Total degree of G is 2m (number of edges)
- Number of vertices of odd degree is always even in a graph
- In an undirected simple graph
  - $m \leq n (n - 1)/2$, where n is number of vertices
- Proof: each vertex has degree at most $(n - 1)$

Definitions contd

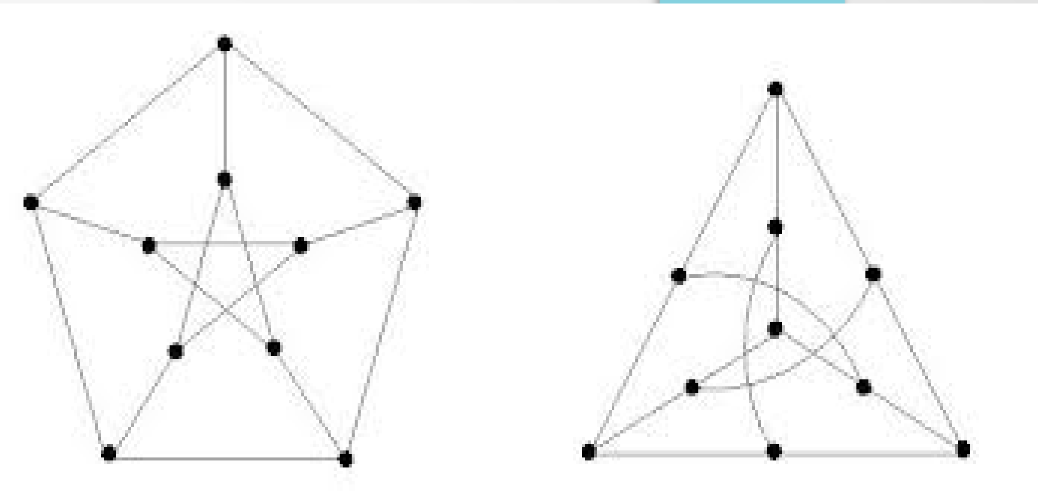
- Planar Graphs: Graph whose edges intersect only at their end
- Graph Isomorphism
  - Two graphs G and H are isomorphic if
    - They have same number of vertices
    - A pair of nodes are adjacent in G iff a corresponding pair exists in H



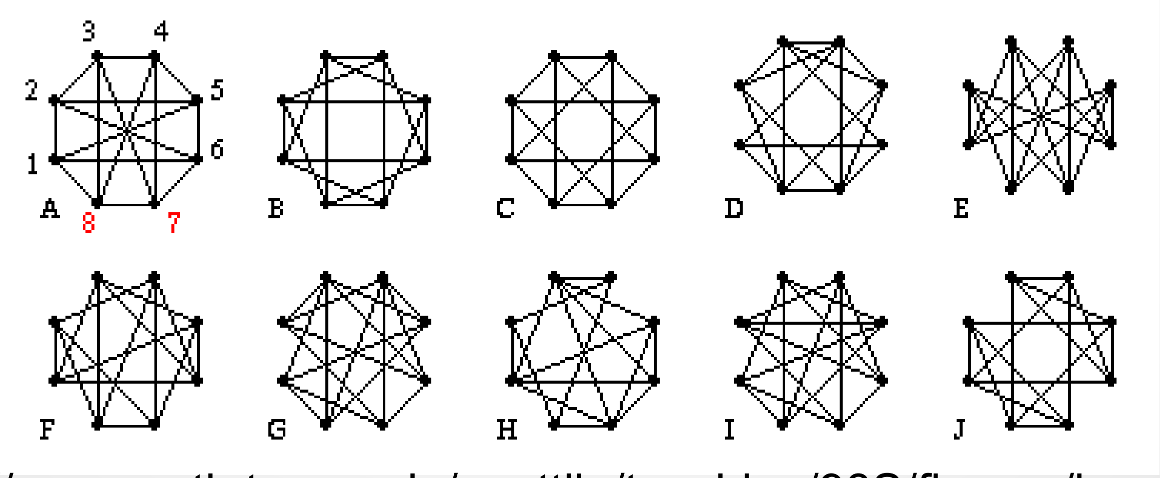
Example: Isomorphism



http://upload.wikimedia.org/



http://www.sonoma.edu/



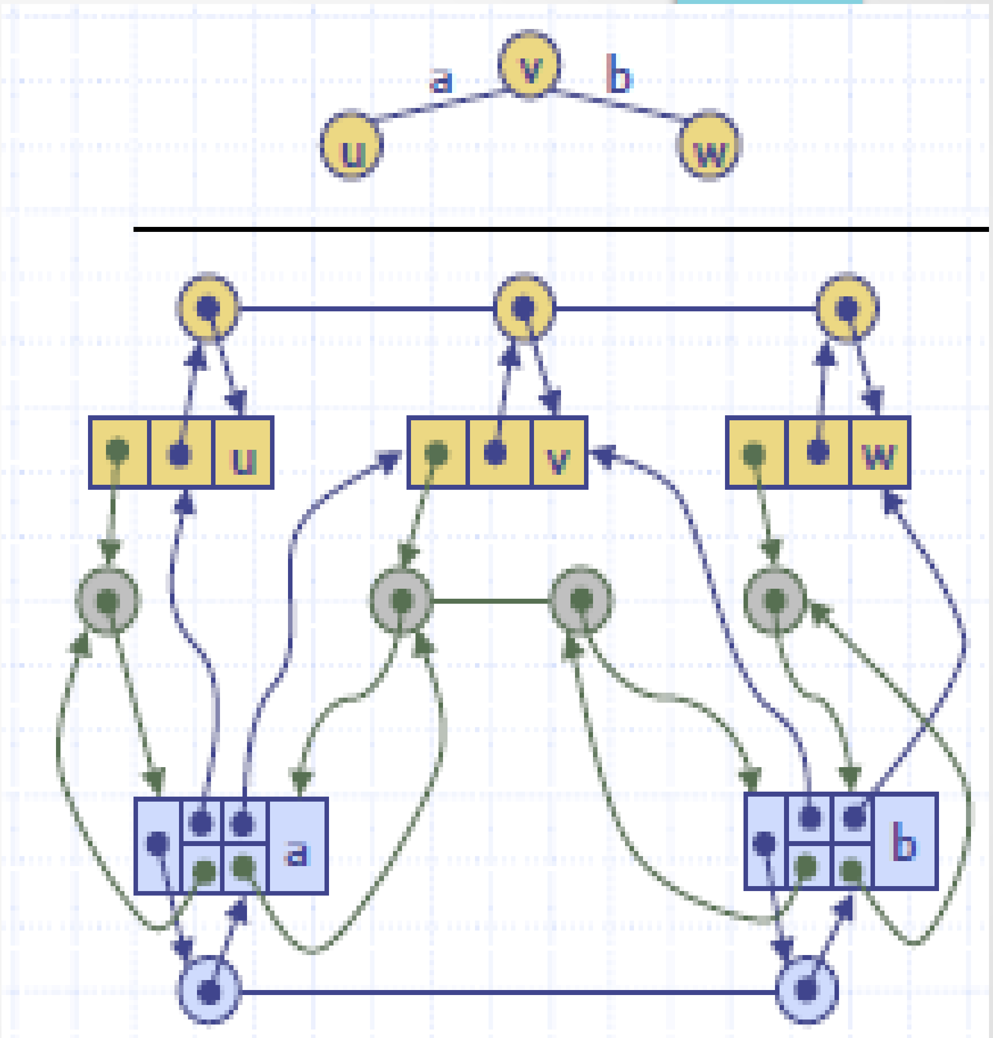
http://www.math.tamu.edu/~sottile/teaching/98S/figures/hard_iso.gif

Graph ADT

- Accessor methods
  - isVertex()
  - incidentEdges(v)
  - endVertices(e)
  - isDirected(e)
  - origin(e)
  - destination(e)
  - opposite(v, e)
  - areAdjacent(v, w)
- Update methods
  - insertVertex(x)
  - insertEdge(v, w, x)
  - insertDirectedEdge(v, w, x)
  - removeVertex(v)
  - removeEdge(e)
- Generic methods
  - numVertices()
  - numEdges()
  - vertices()
  - edges()

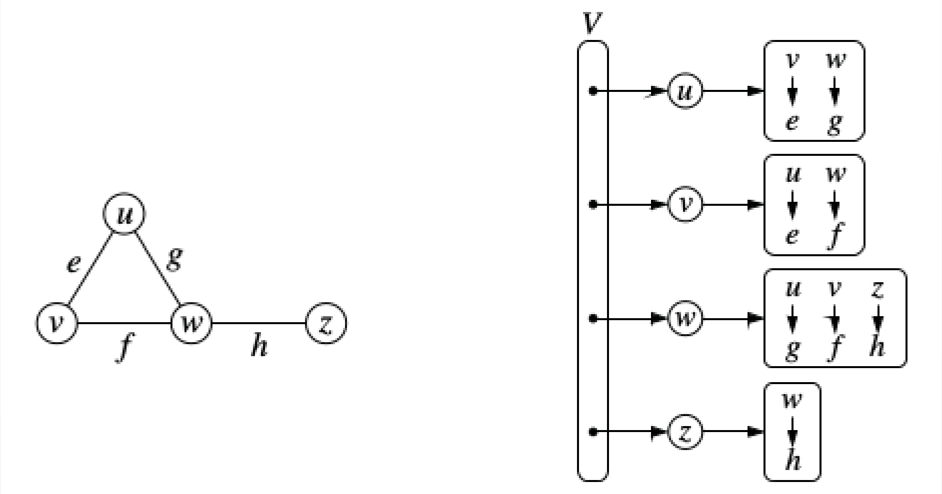
Graph Representations

- Adjacency list
  - Each vertex has incidence container
    - List of vertices incident on v
  - Edge list
- Augmented edge objects
  - references to associated positions in incidence sequences of end vertices



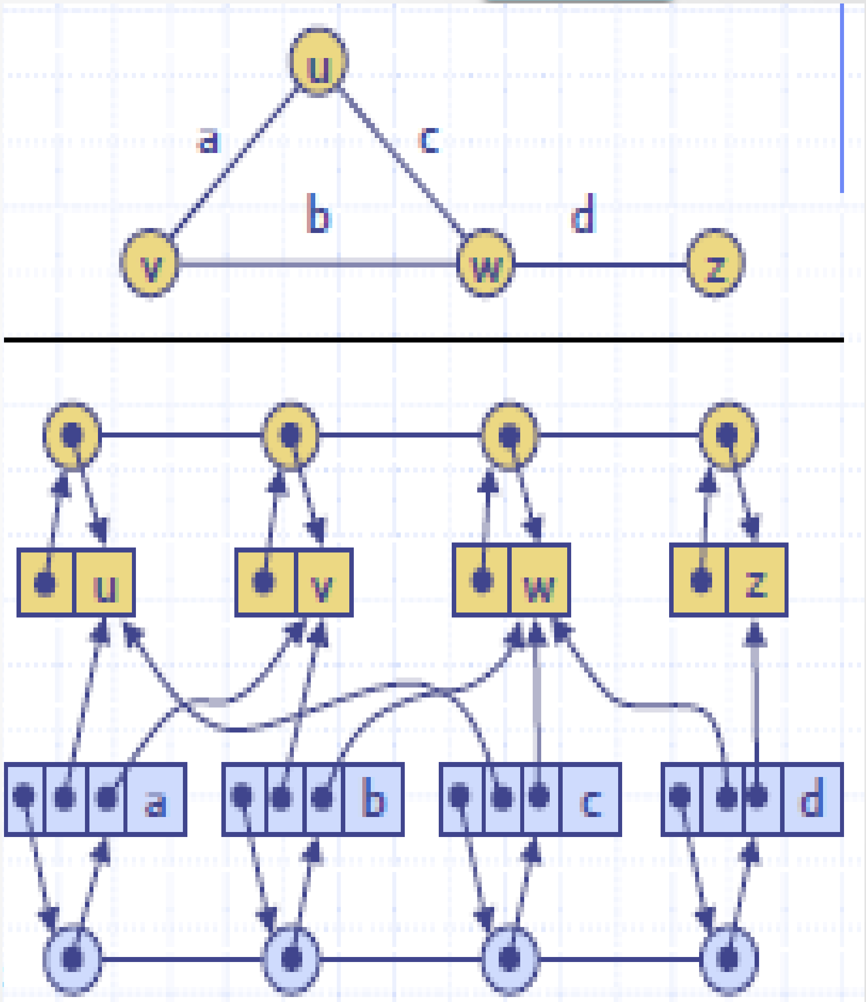
Adjacency Map

- Similar to adjacency list
  - secondary container of all edges incident to a vertex is organized as a map, rather than as a list
  - adjacent vertex serves as a key to the edge
- Allows for fast access to a particular neighbor



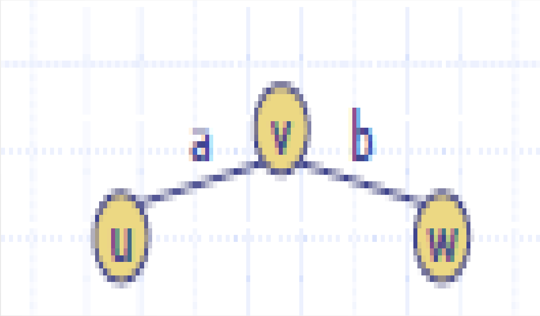
Edge List Representation

- Vertex object
  - Element
  - reference to position in vertex sequence
- Edge object
  - element
  - origin vertex
  - destination vertex
  - reference to position in edge sequence
- Vertex sequence
  - Sequence of vertices in G
- Edge sequence
  - Sequence of edge objects



Adjacency Matrix

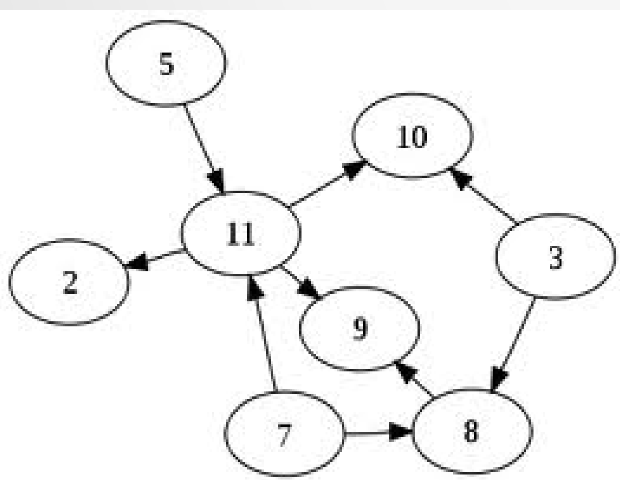
- Edge list
- 2D-array adjacency array A where
  - A[i,j] is 1 if there is an edge between vertices i and j
  - A[i,j] =0, if there is no edge incident on i and j

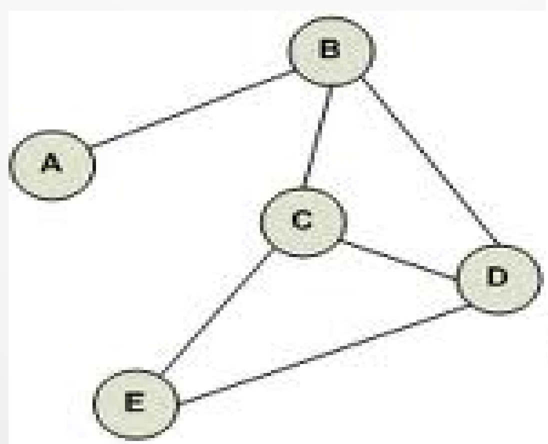


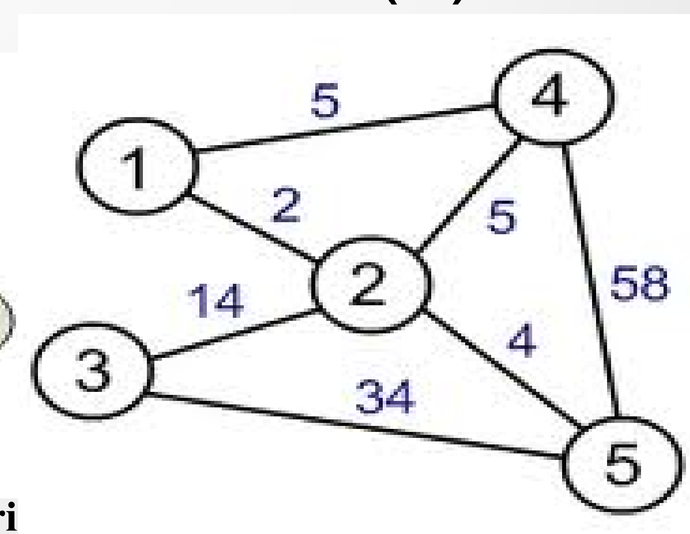
	u	v	w
u	0	1	0
v	1	0	1
w	0	1	0

Exercises

- Represent the following graphs using the three representations
- Suppose we represent a graph G with n vertices and m edges with edge list structure. Why does the function insertVertex() function take O(1) time, while removeVertex() function runs in O(n) time





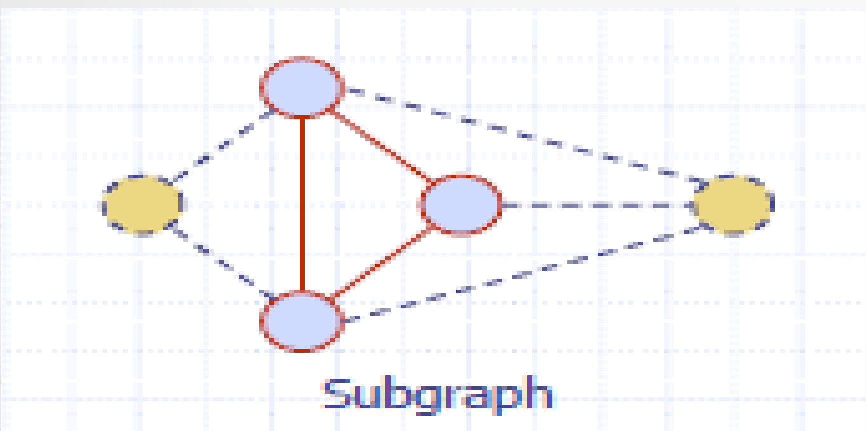


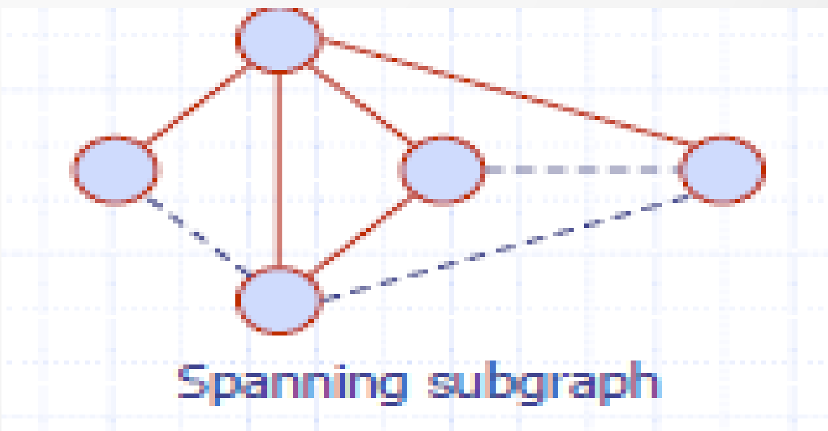
Exercises

- Would you use the adjacency list structure or the adjacency matrix structure for the following cases? Justify.
  - The graph has 10,000 vertices and 20,000 edges and it is important to use as little space as possible
  - The graph has 10,000 vertices and 20,000,000 edges and it is important to use as little space as possible
  - You need to answer the query areAdjacent(v,w) as fast as possible, no matter how much space you use

Subgraphs

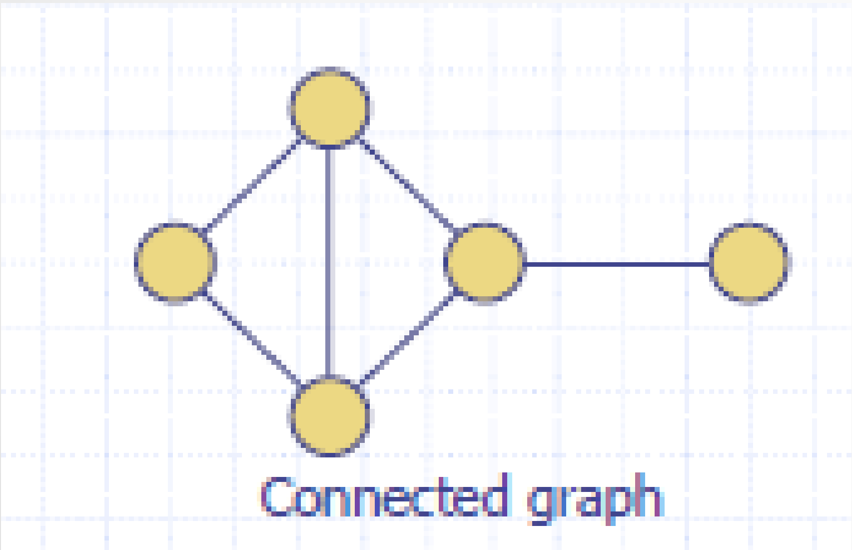
- A subgraph S of a graph G is a graph such that
  - The vertices of S are a subset of the vertices of G
  - The edges of S are a subset of the edges of G
- A spanning subgraph of G
  - is a subgraph that contains all the vertices of G

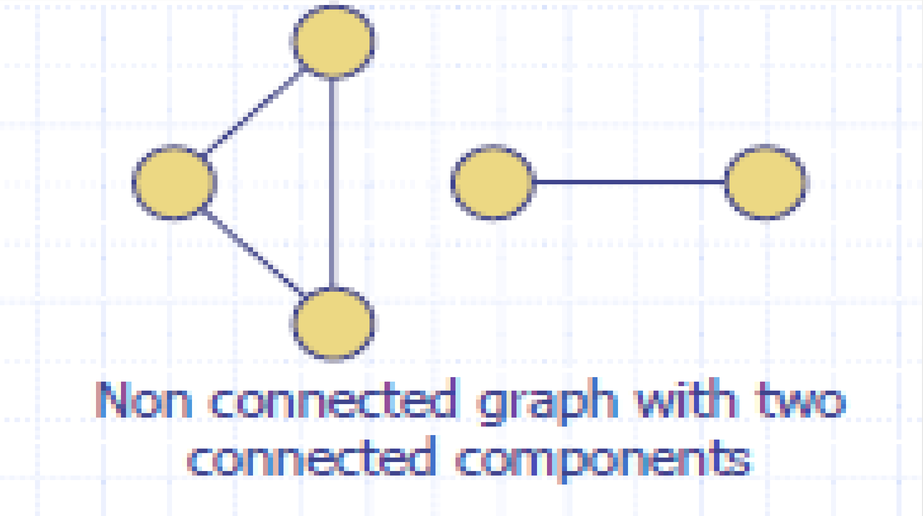




Connectivity

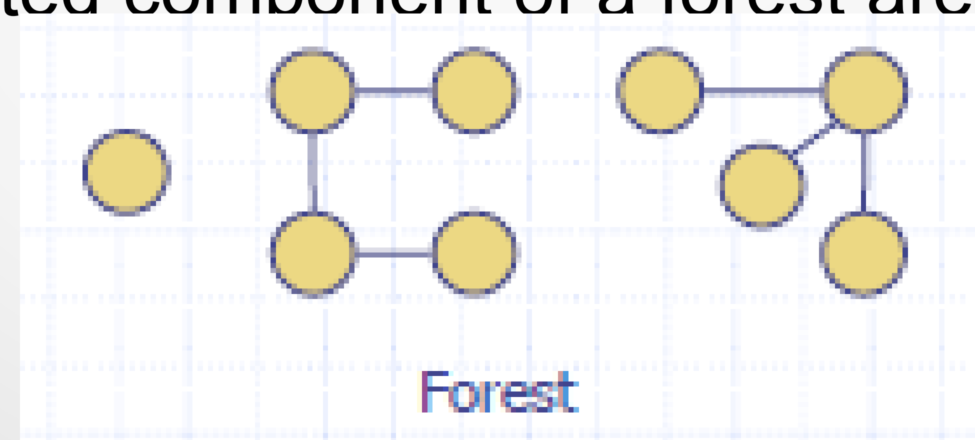
- A graph is connected if there is a path between every pair of vertices
- Connected component of G
  - Maximally connected subgraph of G





Trees and Forests

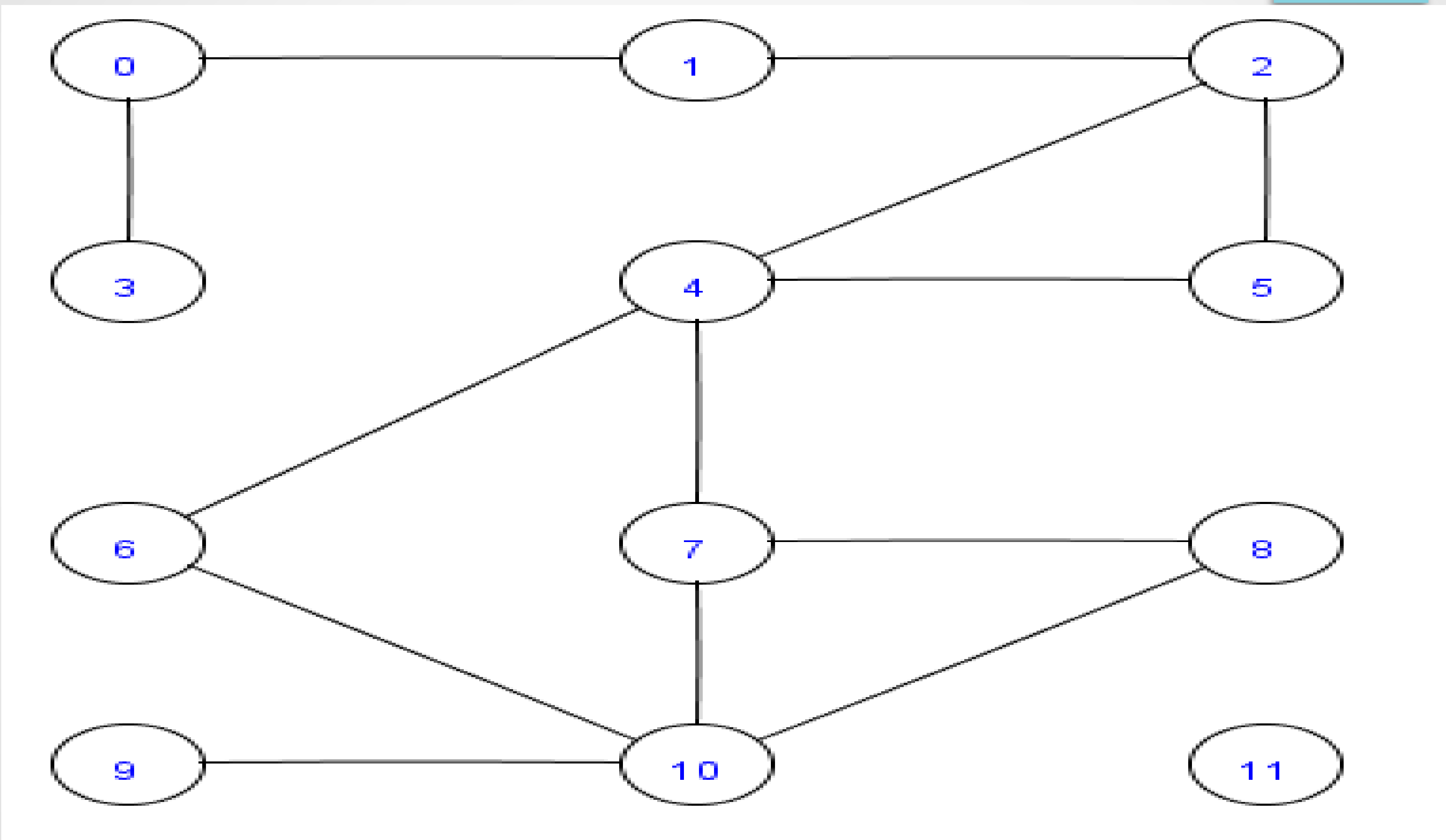
- A tree is an undirected graph such that
  - T is connected
  - T has no cycles
  - Need not be rooted
- Forest is an undirected graph without cycles
  - Connected component of a forest are trees



Graph Traversal

- Depth first search
  - Recursive – O(m+n) algorithm
  - Start with some node v
    - Of all neighbors of v, goto next w which is unexplored do DFS(w)
    - If w explored, then mark edge as back edge
- Similar to a Maze traversal
  - Mark each intersection, corner and dead end (vertex) visited
  - Mark each corridor (edge) traversed
  - Keep track of the path back to the entrance (start vertex) by means of a rope (recursion)

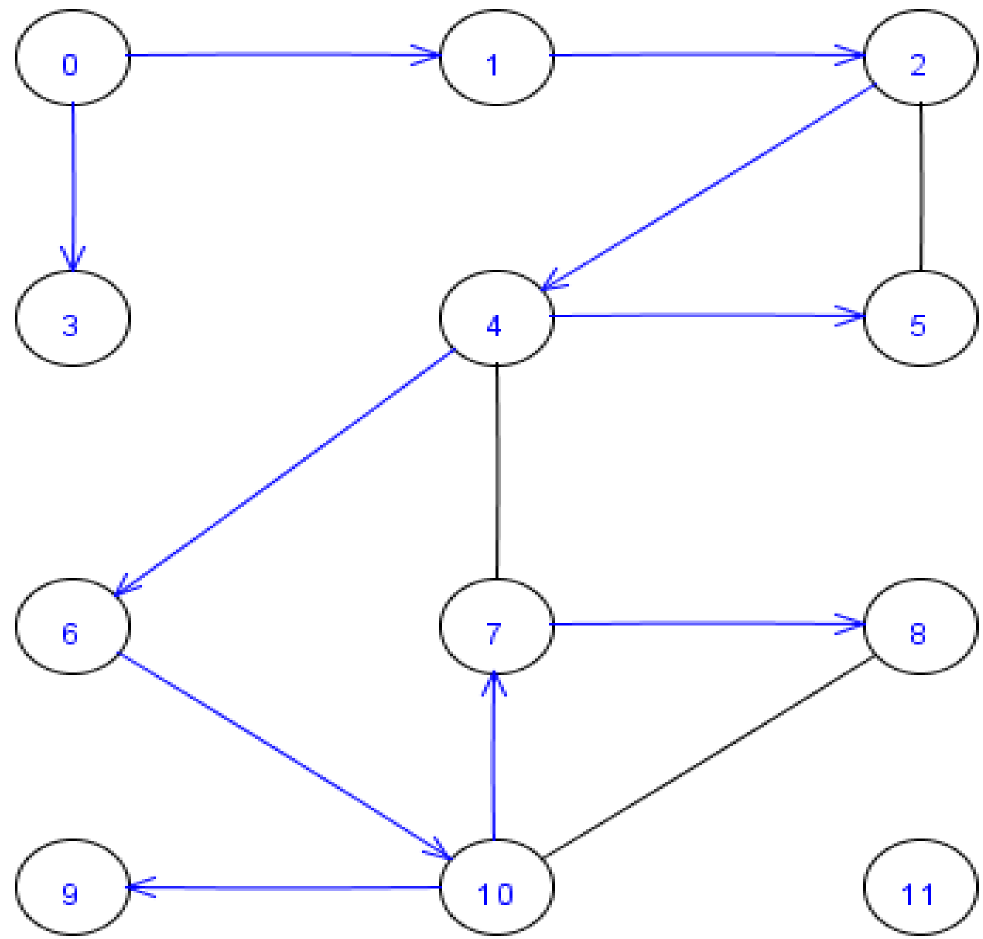
DFS Example



DFS Example Contd

DFS(0)
DFS(1)
DFS(2)
DFS(4)
DFS(5)
DFS(6)
DFS(10)
DFS(7)
DFS(8)
DFS(9)
DFS(3)

DFS(11)



DFS

- **Algorithm** DFS(*G,v*)

for all edges e in G **do**

    if e is unexplored **then**

    w ← G.opposite(v,e)

    if w is unexplored **then**

        label e as a discovery edge

        DFS(G,w)

    else

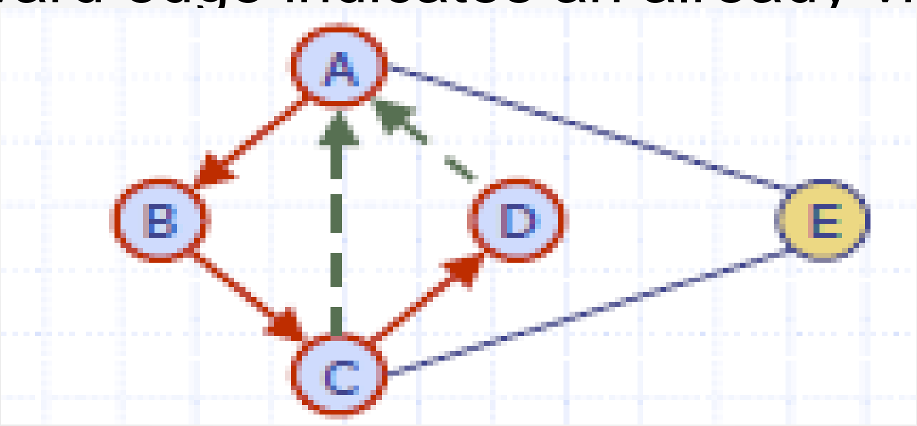
        Label e as back edge

DFS Properties

- DFS(G, v) visits all the vertices and edges in the connected component of v
- The discovery edges labeled by DFS(G, v) form a spanning tree of the connected component of v
  - Spanning tree is a tree which contains
    - All the nodes in the connected component
    - Subset of edges connecting all the nodes in the component such that no cycles are formed

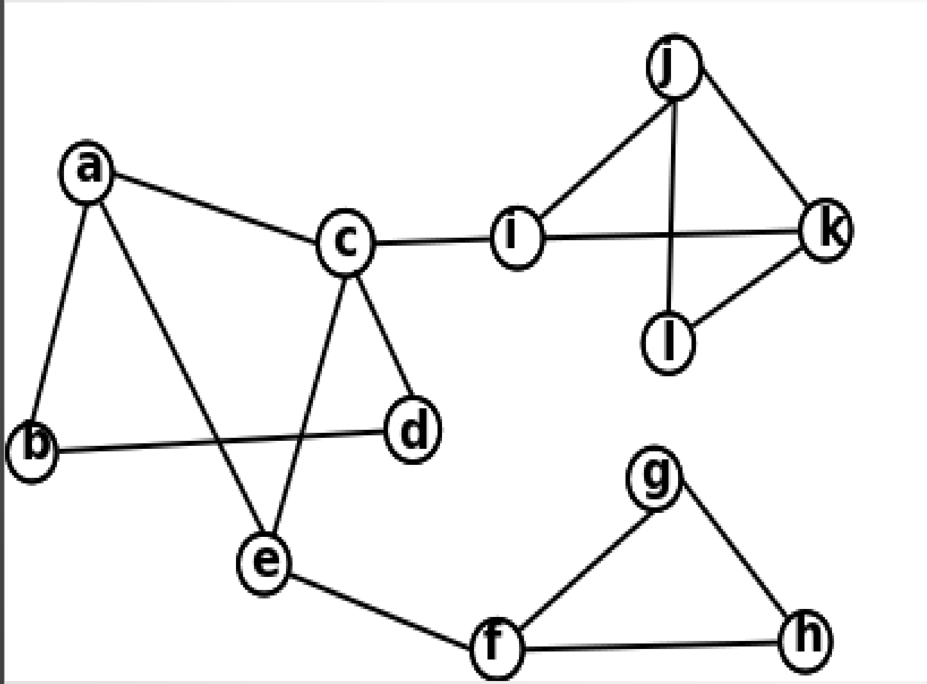
DFS Applications

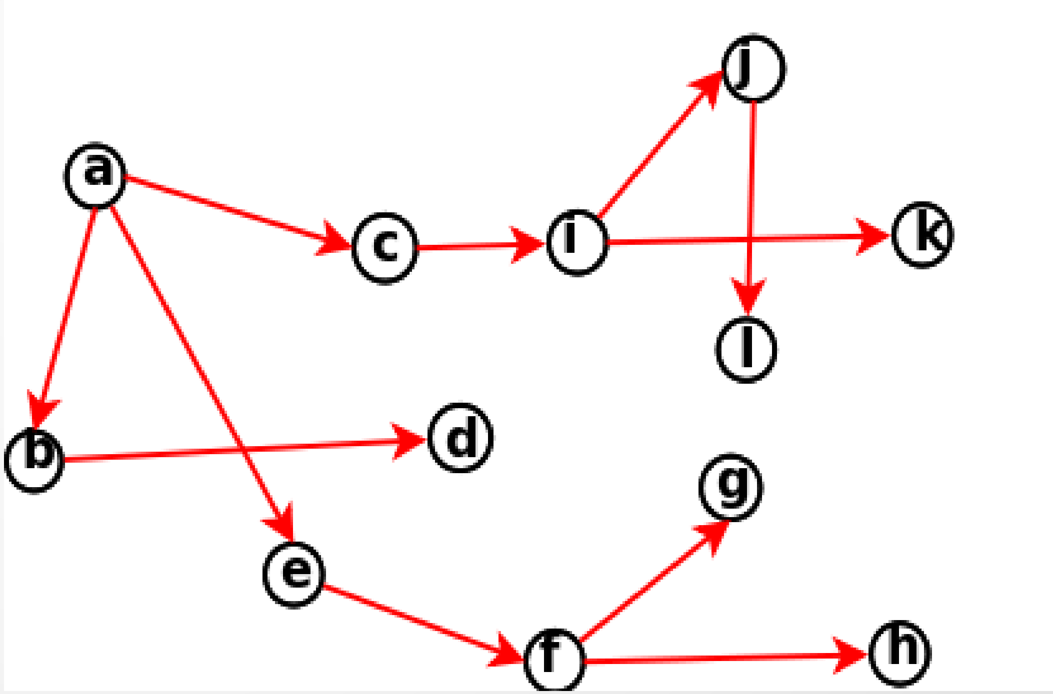
- Path finding
  - Can be used to find path between two vertices
- Cycle Finding
  - Usually a cycle is a sequence of forward edges with one backward edge
    - Backward edge indicates an already visited node



Breadth First Traversal

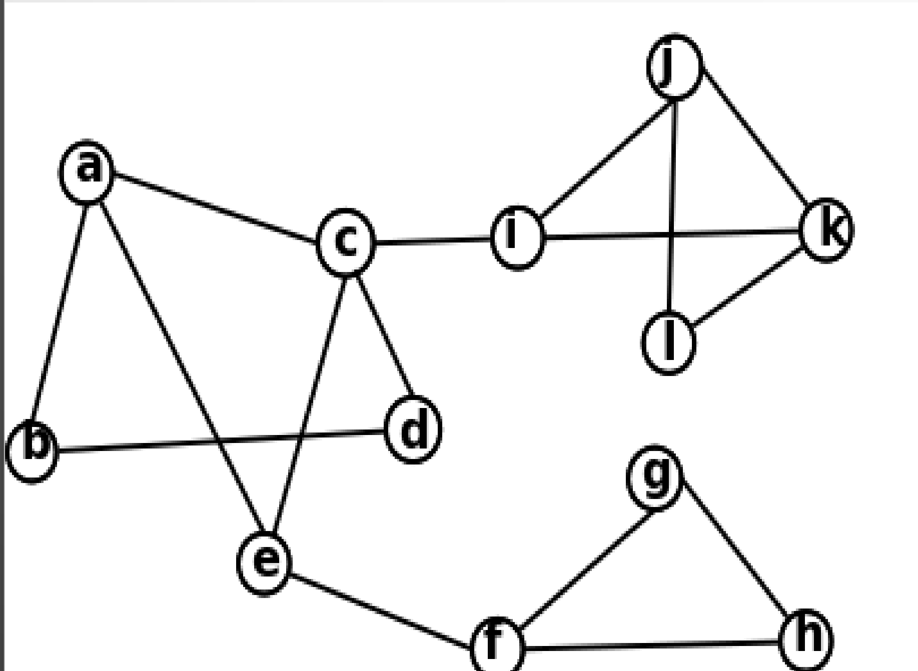
- Discovery in levels, marks new nodes in levels
- O(m+n)

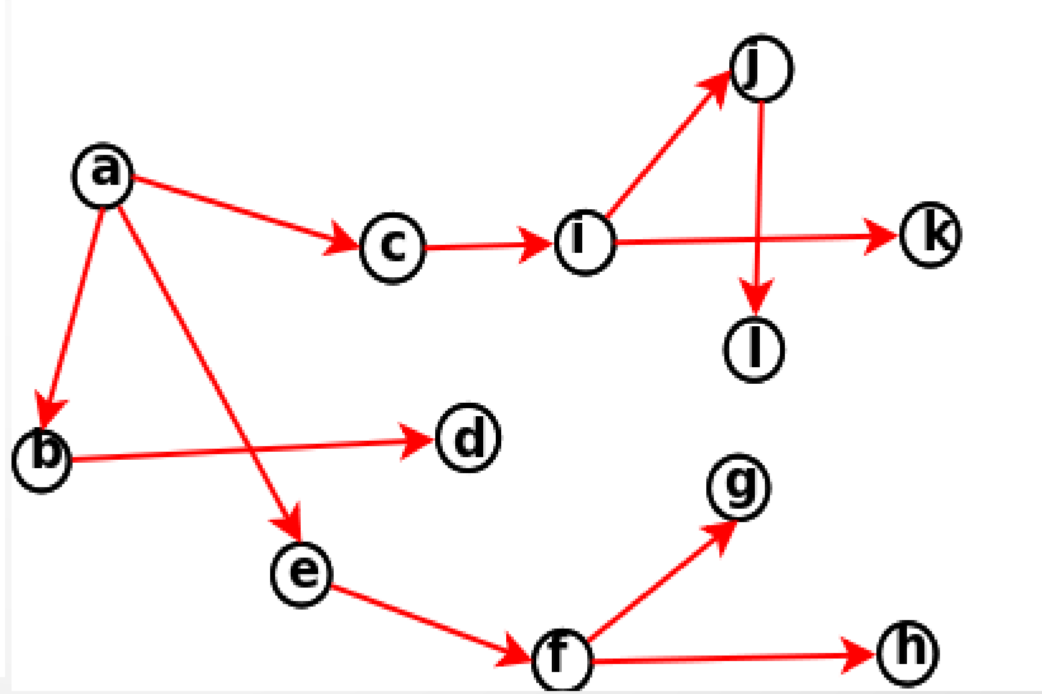




Breadth First Traversal

- Discovery in levels, marks new nodes in levels
  - Cross edges connect to already discovered nodes
- O(m+n)





BFS

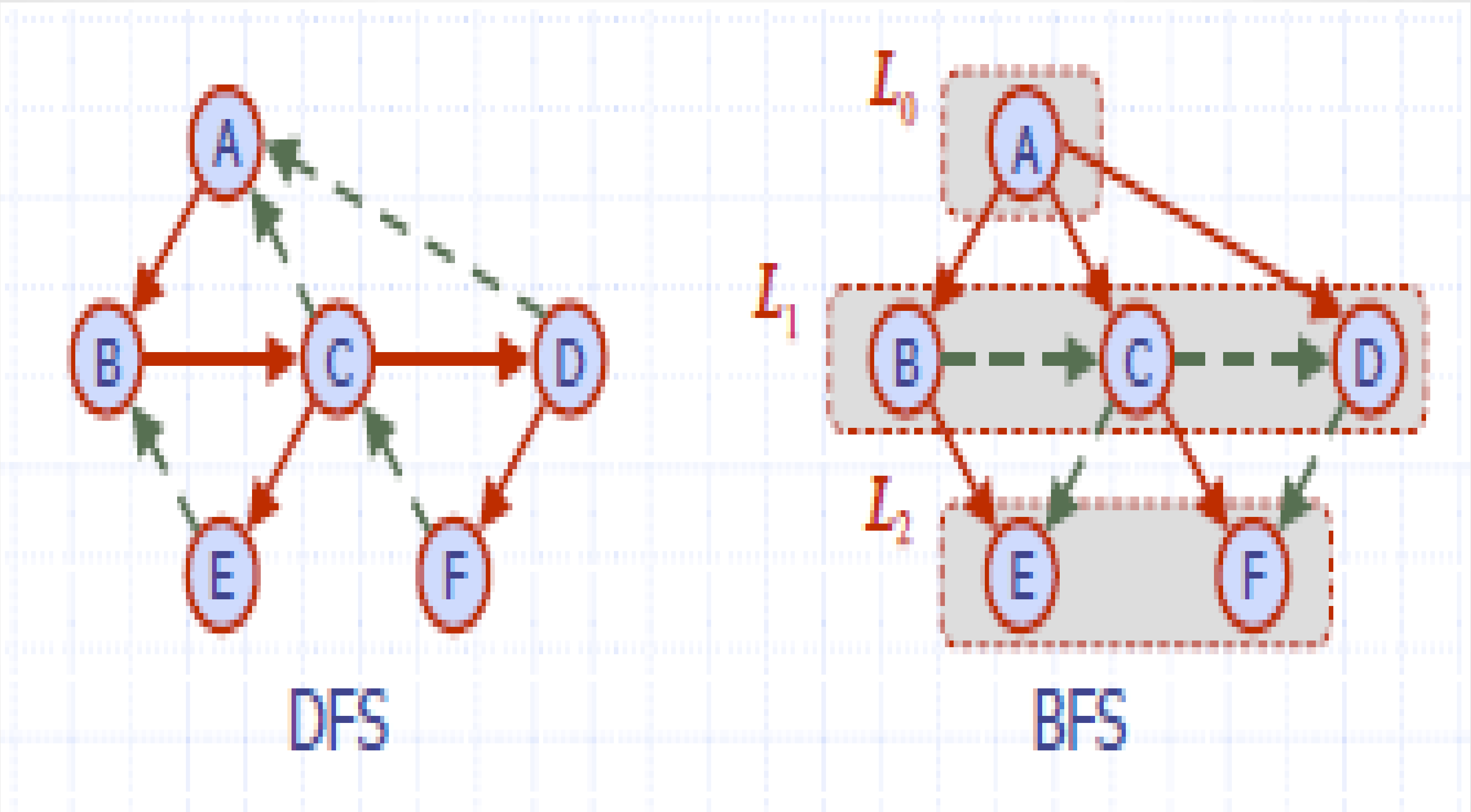
- Algorithm BFS(v)

```
initialize container L0 to contain vertex v and i to 0
while Li is not empty do
    for each vertex v in Li do
        for each edge e incident on v
            if e is unexplored
                let w be the other endpoint of e
                If w is unexplored
                    label e as a discovery edge and insert w into Li+1
                else label e as cross edge
    i ← i+1
```

Applications of BFS

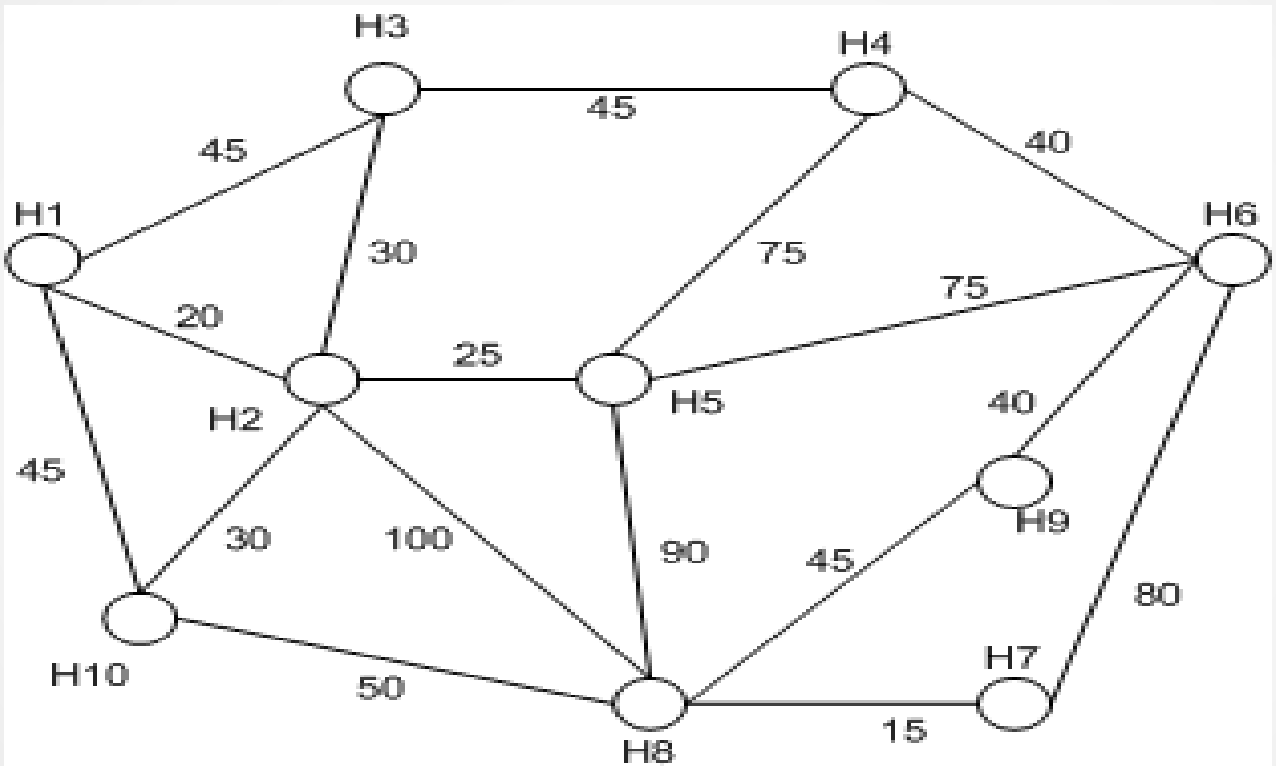
- Find and report a path with the minimum number of edges between two given vertices
- Find a simple cycle, if there is one

BFS vs DFS

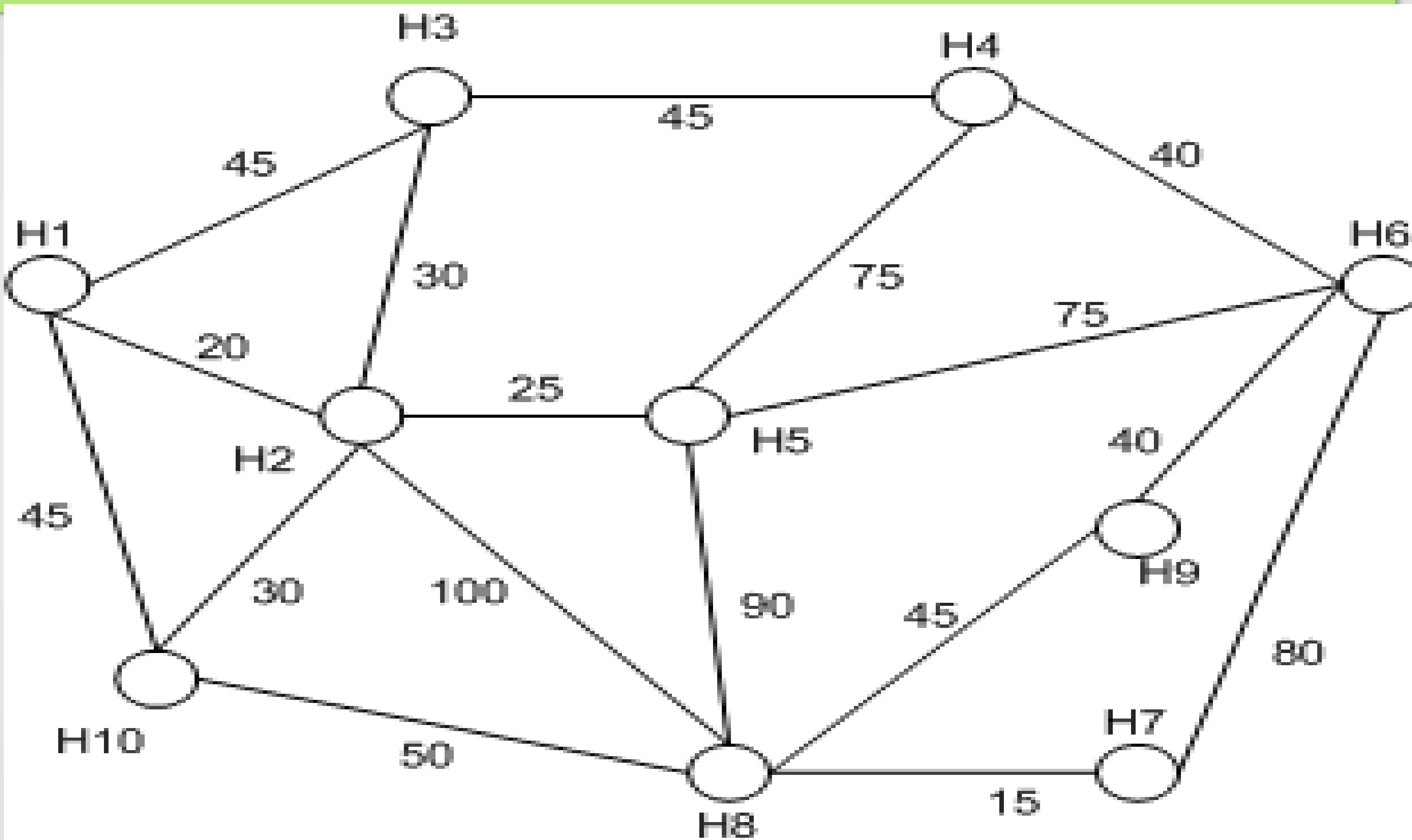


Exercises

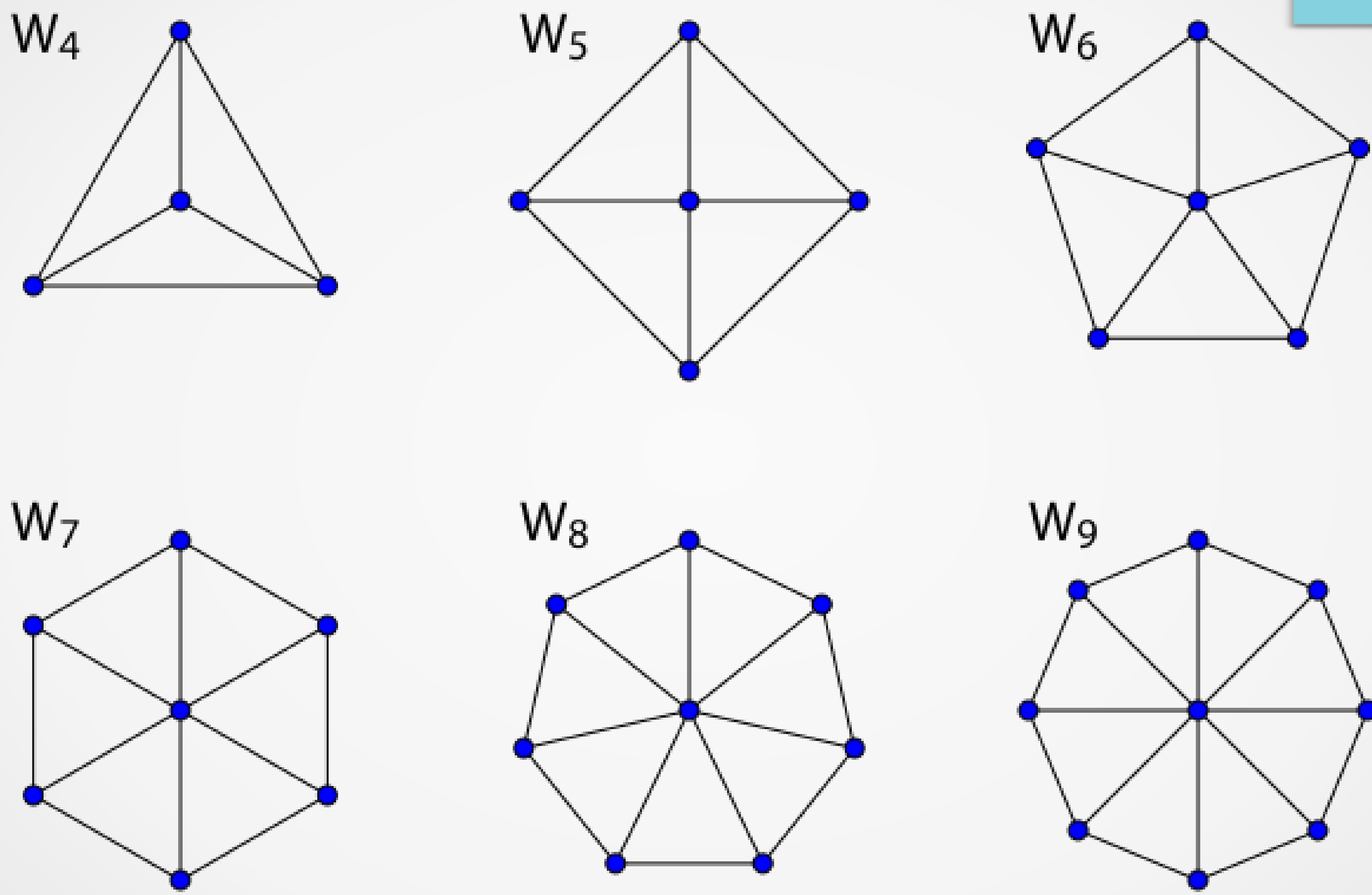
- Do the BFS and DFS traversal over the following graph



- Illustrate all possible structures of traversal trees generated on a wheel graph Wn


H3
H4
45
45
40
H1
H6
30
75
75
20
25
H2
H5
40
45
H9
30
100
90
45
80
H7
H10
50
15
H8

Exercise

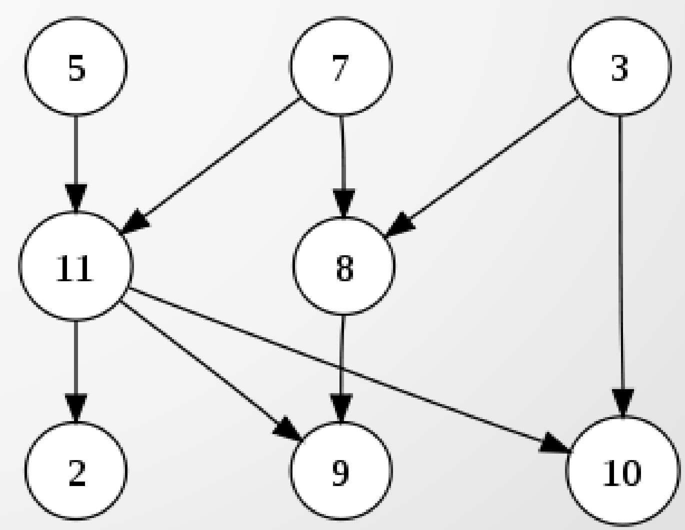


Exercise

- How are the DFS and BFS modified for a directed graph?
- Write an algorithm to find the longest path in a DAG, where the length of the path is measured by the number of edges that it contains.

Digraphs

- A directed graph or digraph is a pair G =(V,E) of
  - a set V, whose elements are called vertices or nodes
  - a set A of ordered pairs of vertices, called arcs, directed edges, or arrows
- A directed acyclic graph (DAG) is a directed graph with no directed cycles



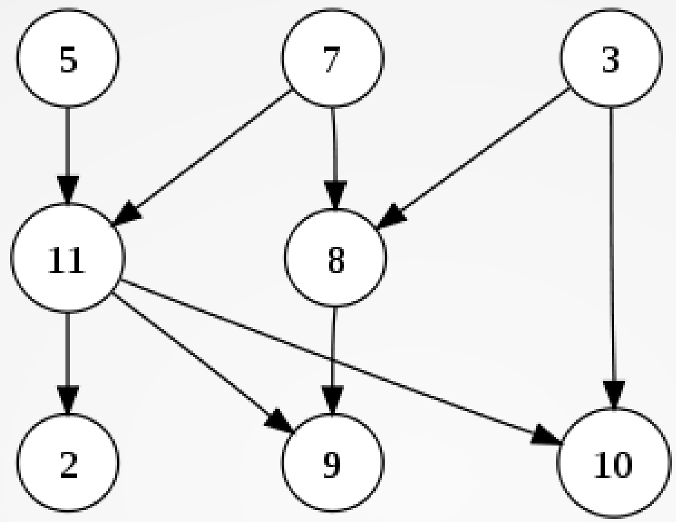
Digraph Applications

- Used to model several kinds of structures in maths or science
- Scheduling
  - Edge (a, b) indicates task a must be completed before b
- Topological Ordering
- Data Processing Networks
  - data enters a processing element through its incoming edges and leaves the element through its outgoing edges
  - Bayesian Networks
  - Compilers
  - Circuit Design

Topological Ordering

- A DAG is a digraph that has no directed cycles and a topological ordering of a digraph is a numbering v_1 , ..., v_n of the vertices such that
  - for every edge (v_i , v_j), we have $i < j$
- Example
  - In task scheduling, a topological ordering is a task sequence that satisfies the precedence constraints
- Theorem
  - A digraph admits a topological ordering if and only if it is a DAG

Example

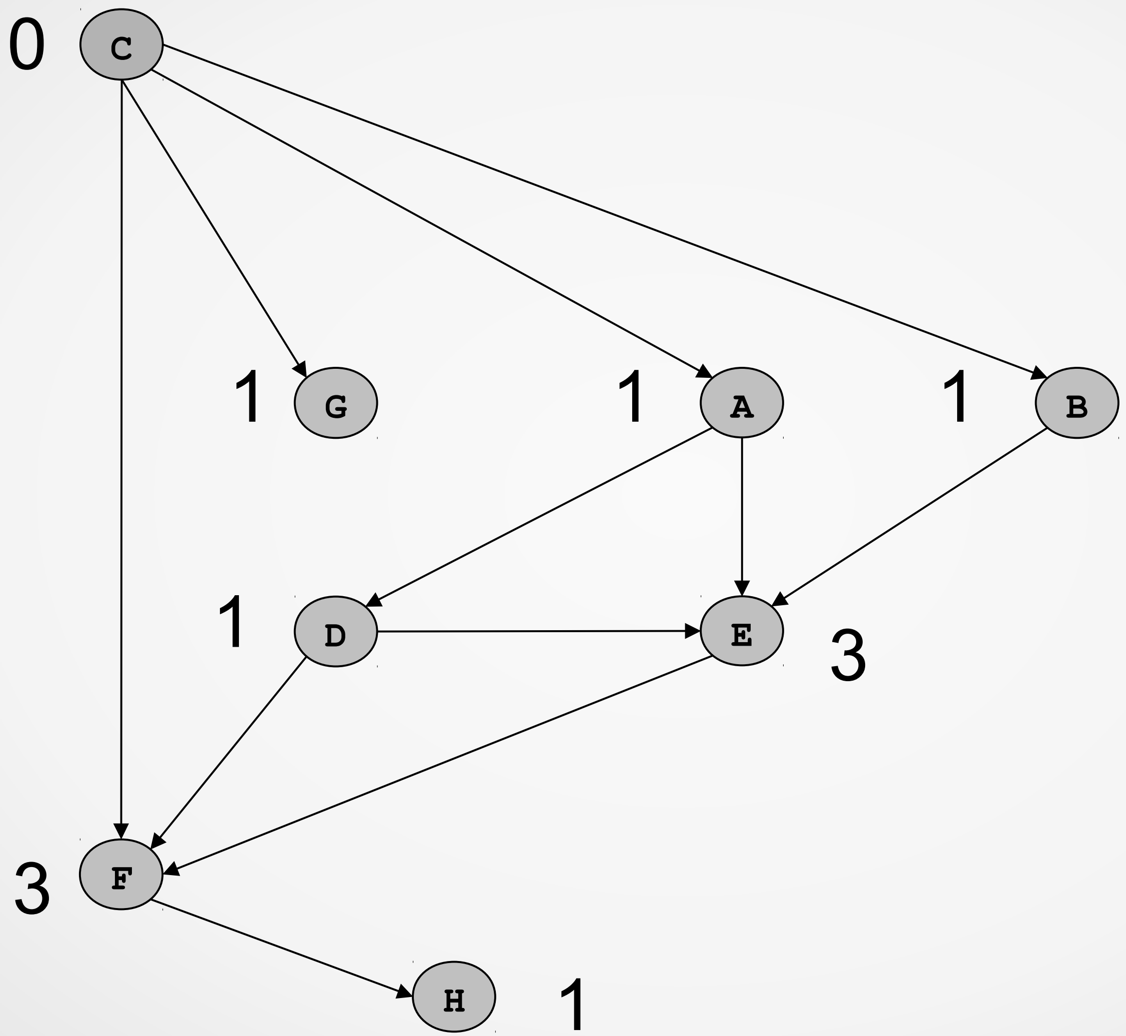


- For the above graph these are some valid topological ordering
  - 5, 7, 3, 11, 8, 2, 9, 10
  - 3, 5, 7, 8, 11, 10, 9, 2

Topological Sorting Algorithm

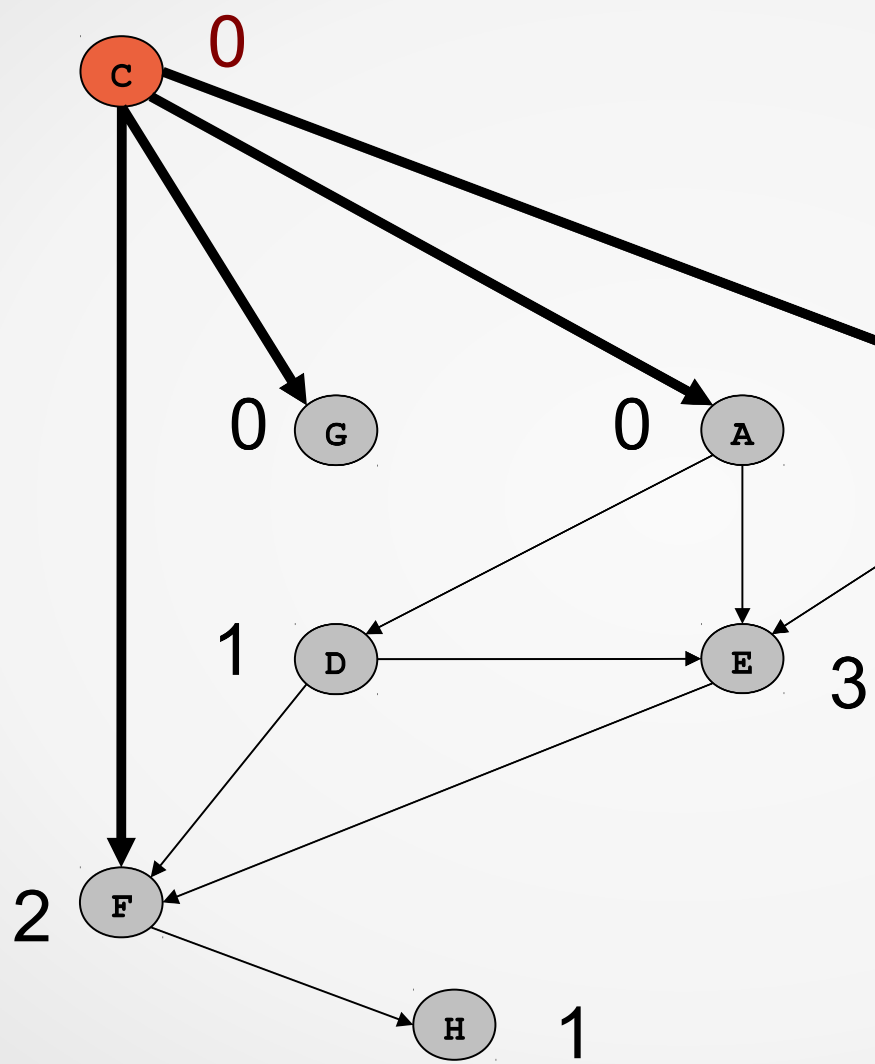
- Let S be an empty stack
- For each vertex u of G, set incounter(u) as the indegree(u)
  - If incounter(u) =0, push it in the stack
- Set i as 1
- If S is empty it has a directed cycle
- Pop the ith vertex u_i from the stack and increment i
  - For each neighbor v of u_i, decrement incounter(v) by 1
  - If incounter(v) =0, push it in stack
  - Do this this while stack has elements in it

Example



Push C

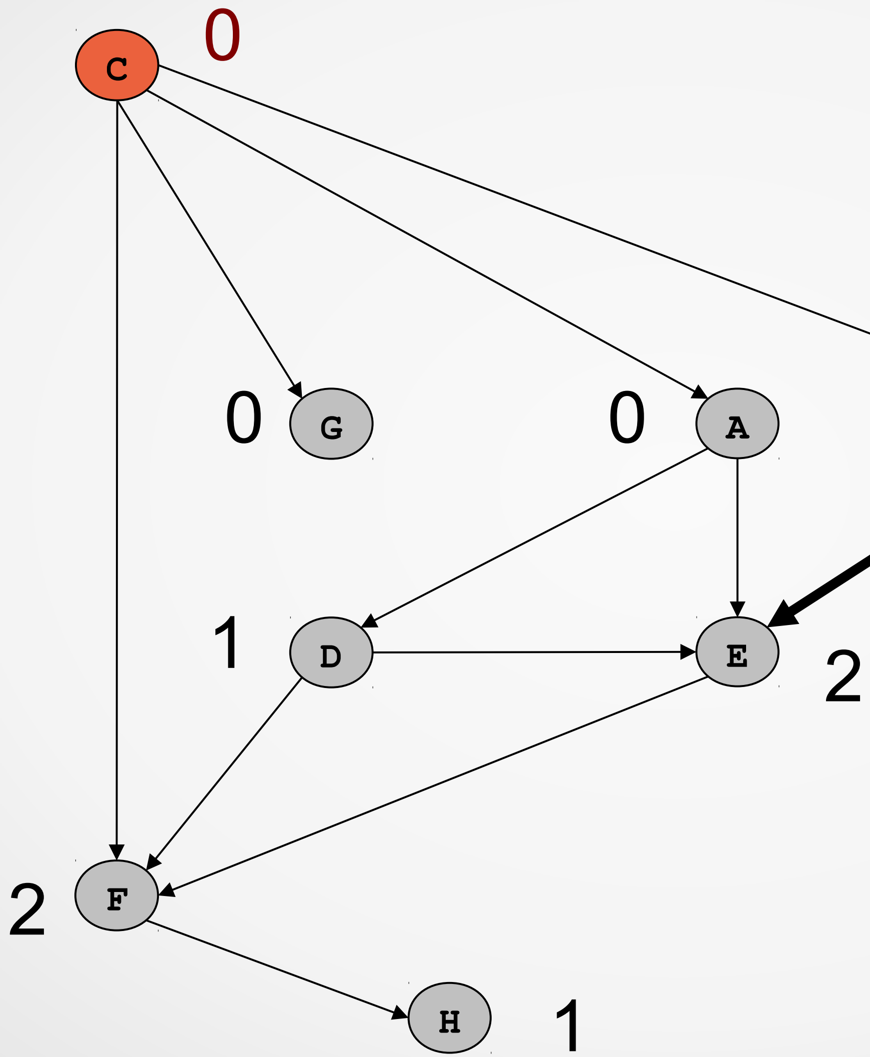
Example



Pop C

In Stack
B
A
G

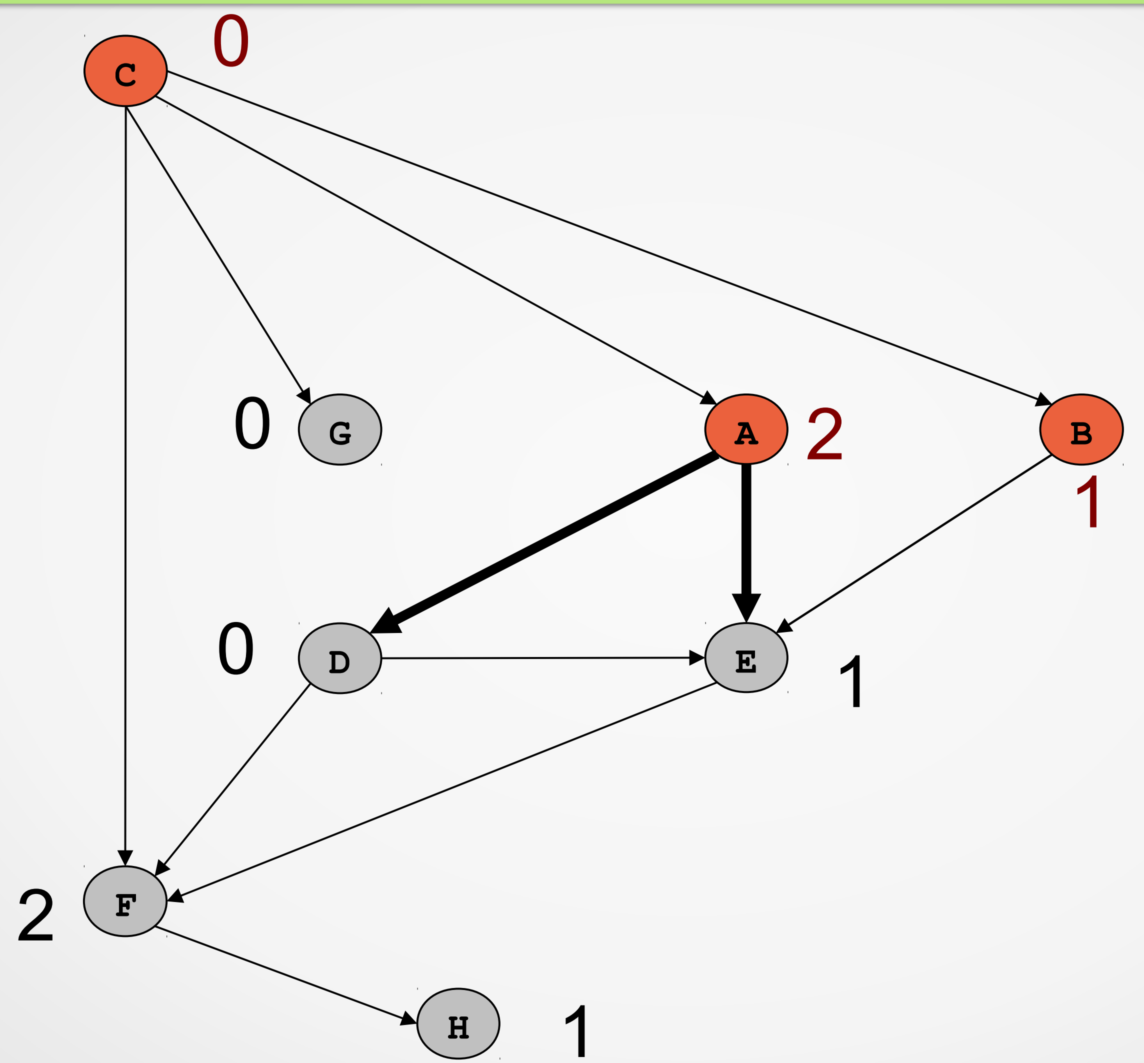
Example



Pop B

In Stack
G
A

Example

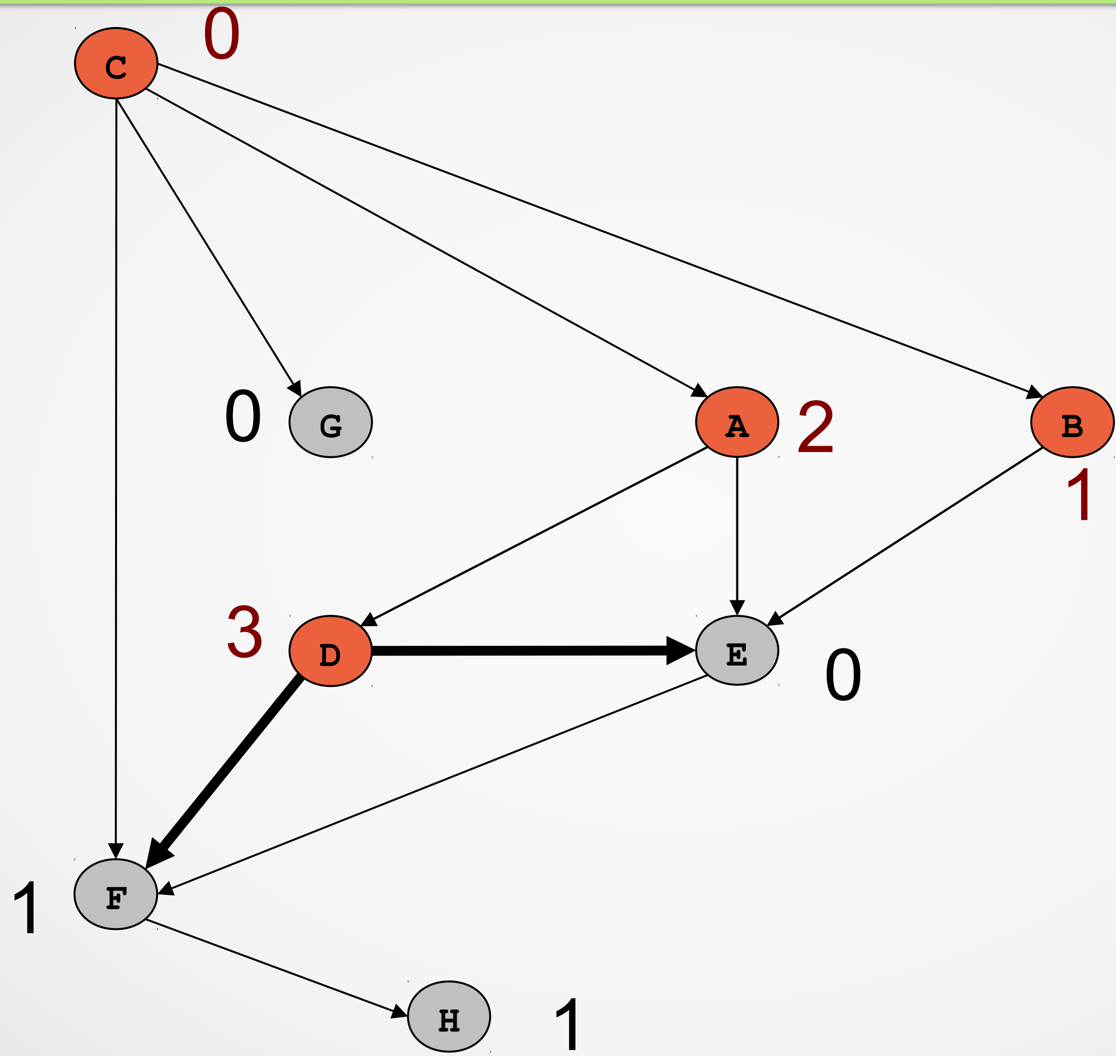


Pop A

Push D

In Stack
D
G

Example

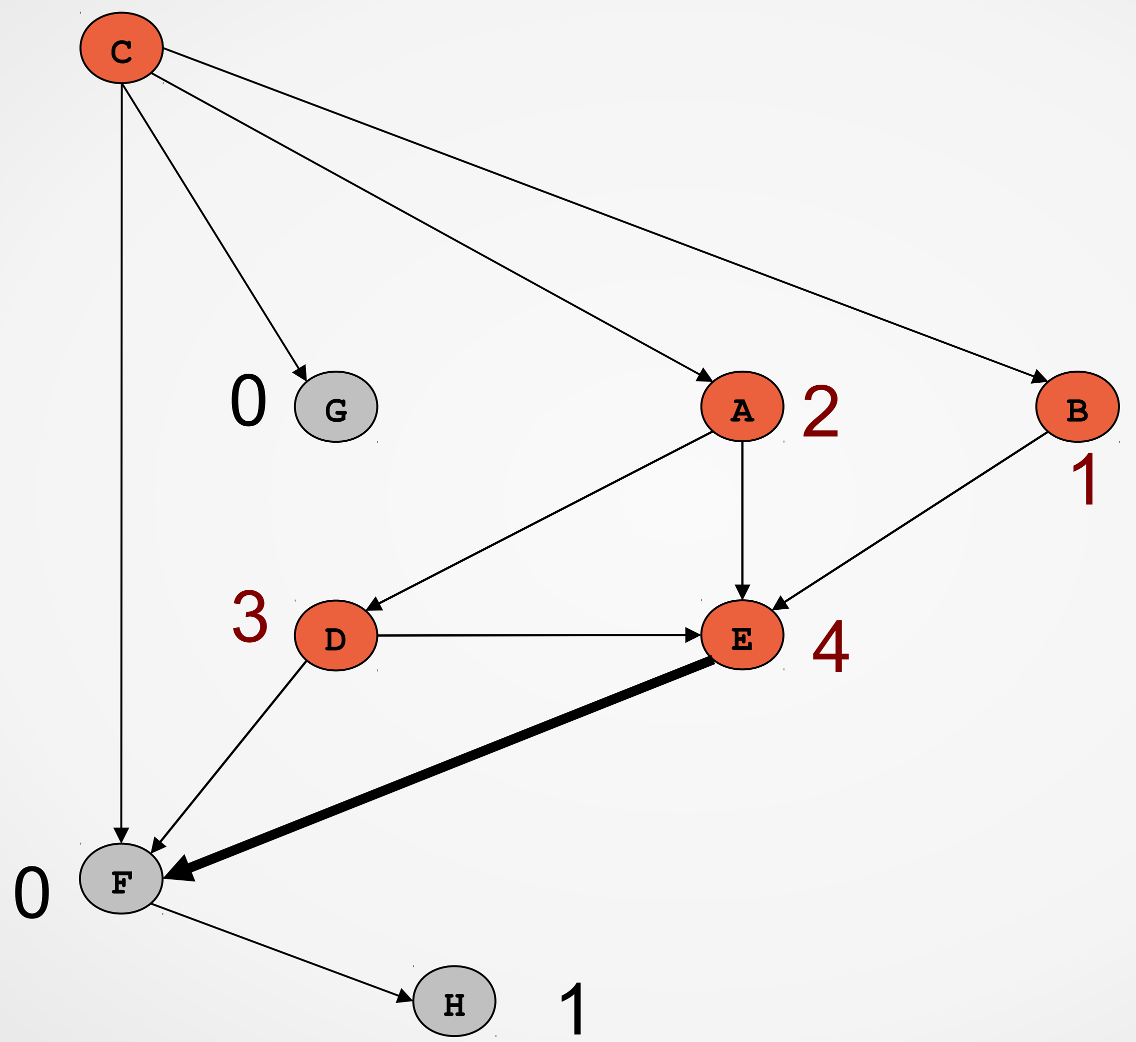


Pop D

Push E

In Stack
E
G

Example

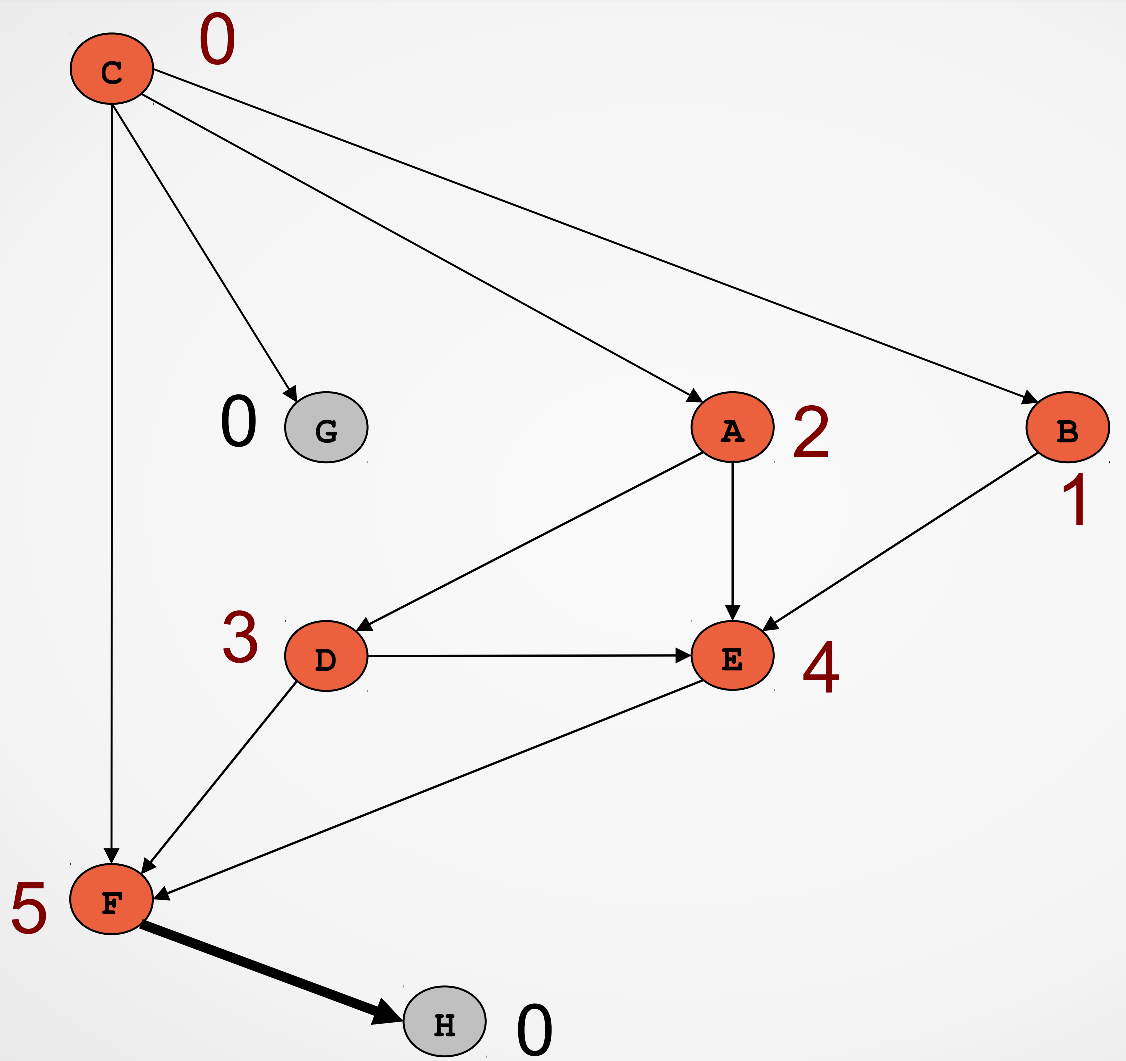


Pop E

Push F

In Stack
F
G

Example

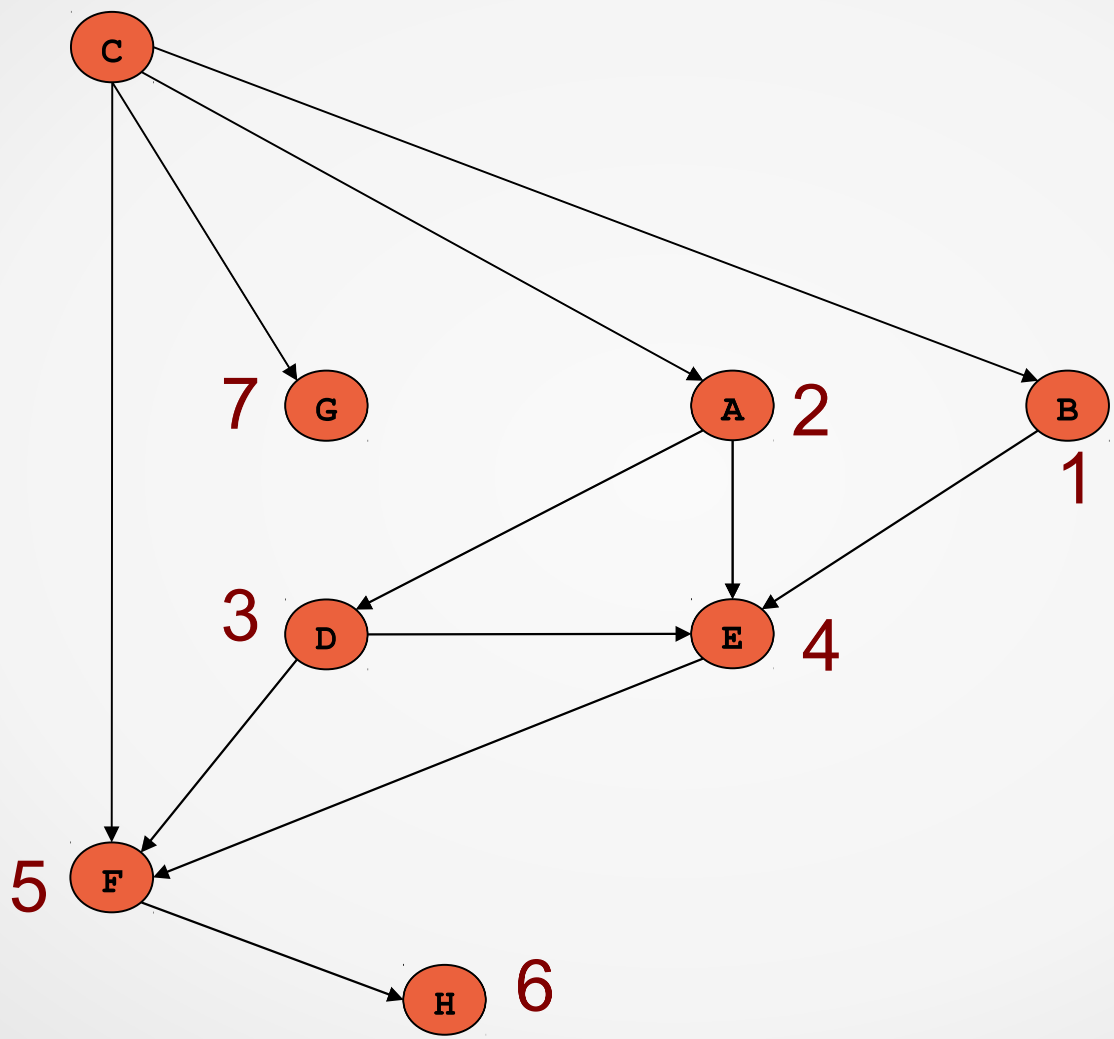


Pop F

Push H

In Stack
H
G

Example



Pop H

Pop G

Analysis

- Runs in O(n+m) time – n vertices and m edges
  - Initial computation of indegree O(n+m)
- Uses O(n) auxiliary space (stack)
- If some vertices are not numbered then there is a cycle
  - Any vertex on a directed cycle will not be visited
  - A vertex visited only when incounter is 0
  - Implies all its previous predecessors were previously visited

Exercise

- A student entering the computer science department wants to plan his course schedule to take the following courses: The course prerequisites are:
  - CS105 – none
  - CS106 – CS105
  - CS122 – none
  - CS201 – CS105
  - CS202 – CS106, CS201
  - CS326 – CS122, CS202
  - CS327 - CS106
  - CS141 – CS122, CS106
  - CS169 – CS202
- Find a sequence of courses that allows Bob to satisfy all the prerequisites

Exercise

- The root of a DAG is a vertex R such that every vertex of the DAG can be reached by a directed path from R. Write an algorithm that takes a directed graph as input and determines the root (if there is one) for the graph. The running time of your algorithm should be (|V| + |E|).