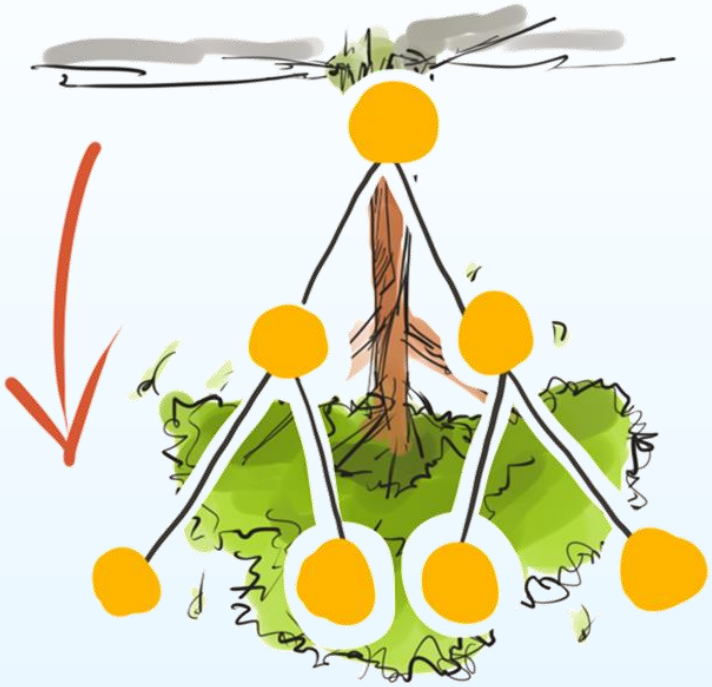


Lecture 13: Graph ADT and Theory

01204212 Abstract Data Types and Problem Solving

Department of Computer Engineering
Faculty of Engineering, Kasetsart University
Bangkok, Thailand.

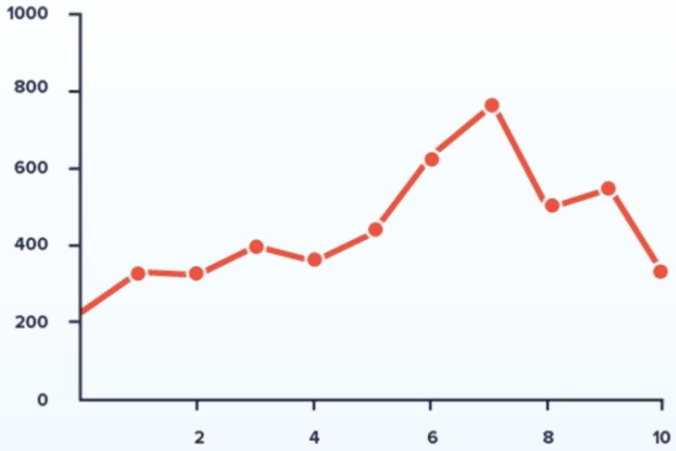
Department of Computer Engineering Kasetsart University

MIKE LABORATORY

Outline

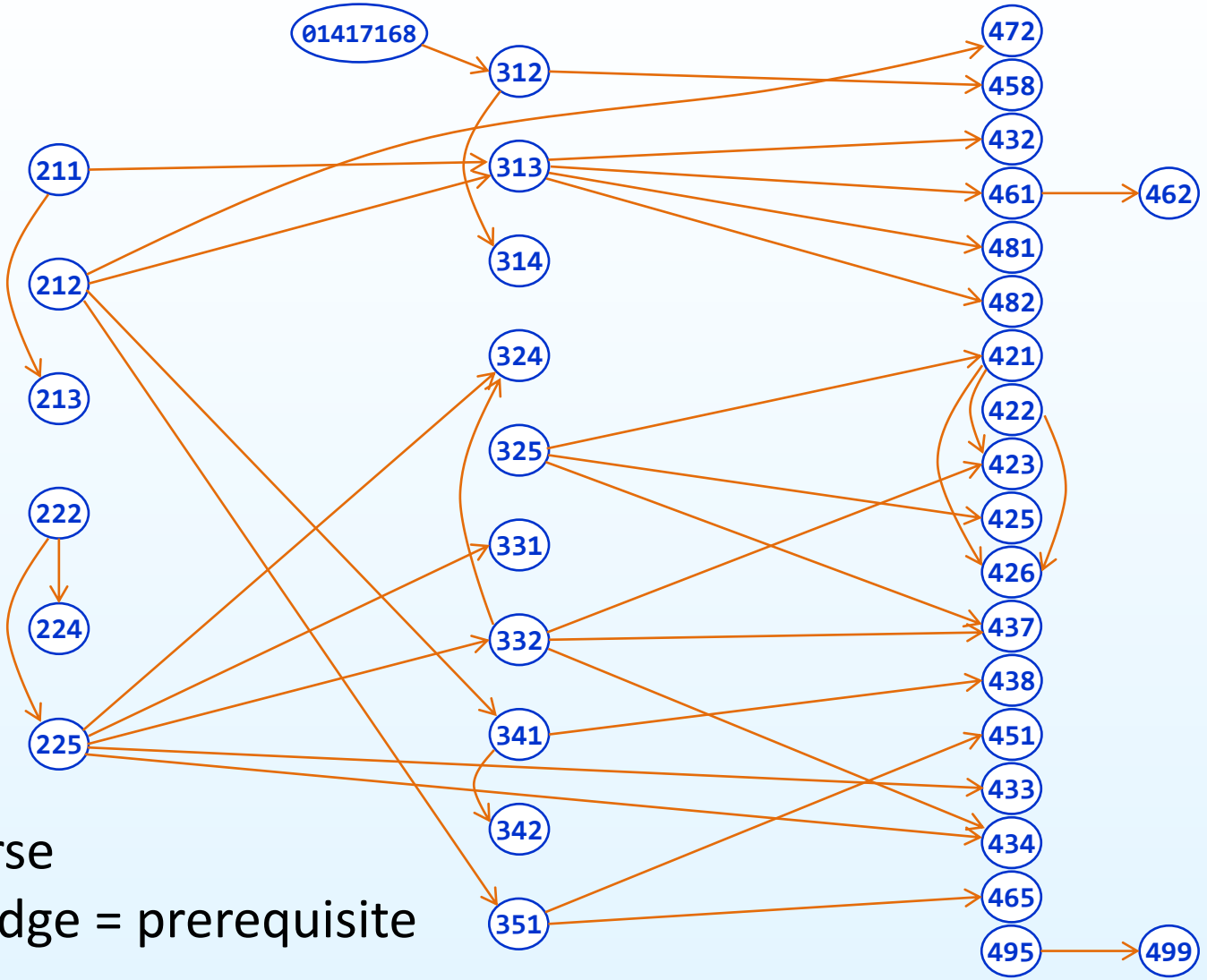
- Graph ADT and Representations
- Graph Traversals
- Connectivity
- Bipartite Graphs
- Topological Sort

What is a Graph?



- Yes, this is a graph
- But we are interested in a different kind of graph
- Consider graphs representing the following problems …

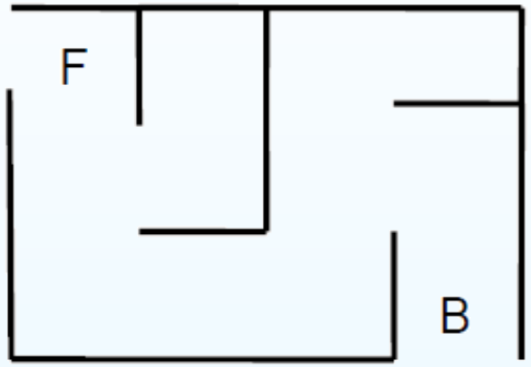
CPE Course Prerequisites

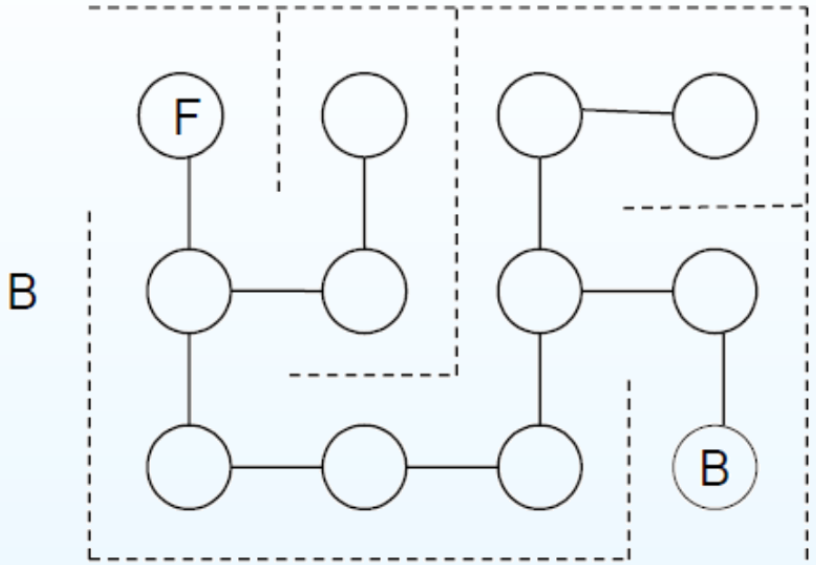


node = course

(directed) edge = prerequisite

Maze Representation

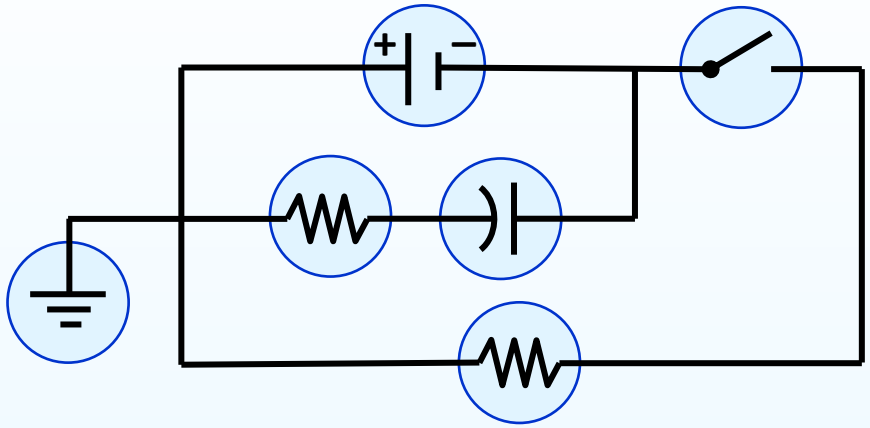




node = room

edge = door or passage

Electrical Circuit

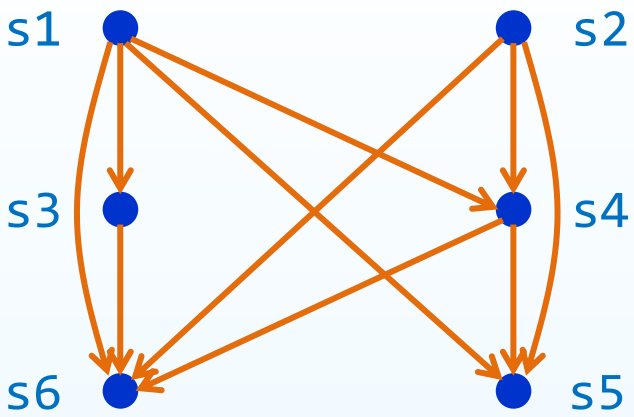


node = battery, switch, resistor, etc.

edge = connection

Precedence

```
a = 0;     // s1
b = 1;     // s2
c = a+1;   // s3
d = b+a;   // s4
e = d+1;   // s5
e = c+d;   // s6
```



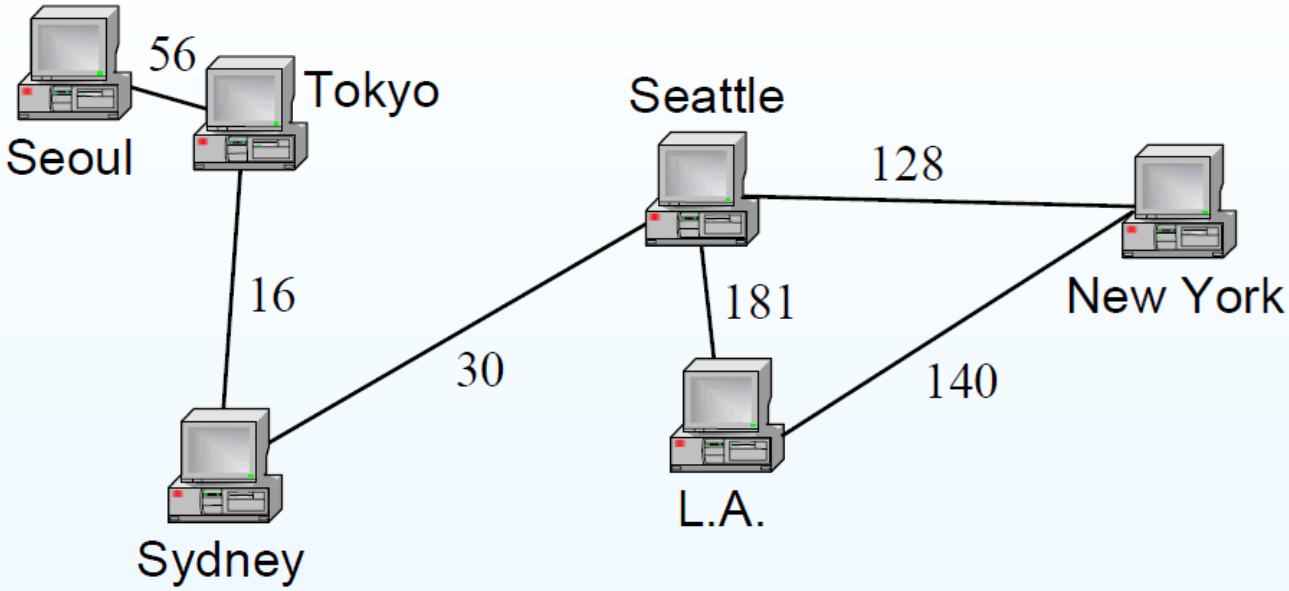
Which statements must execute before s6?

➡ s1, s2, s3, s4

node = statement

edge = precedence requirement

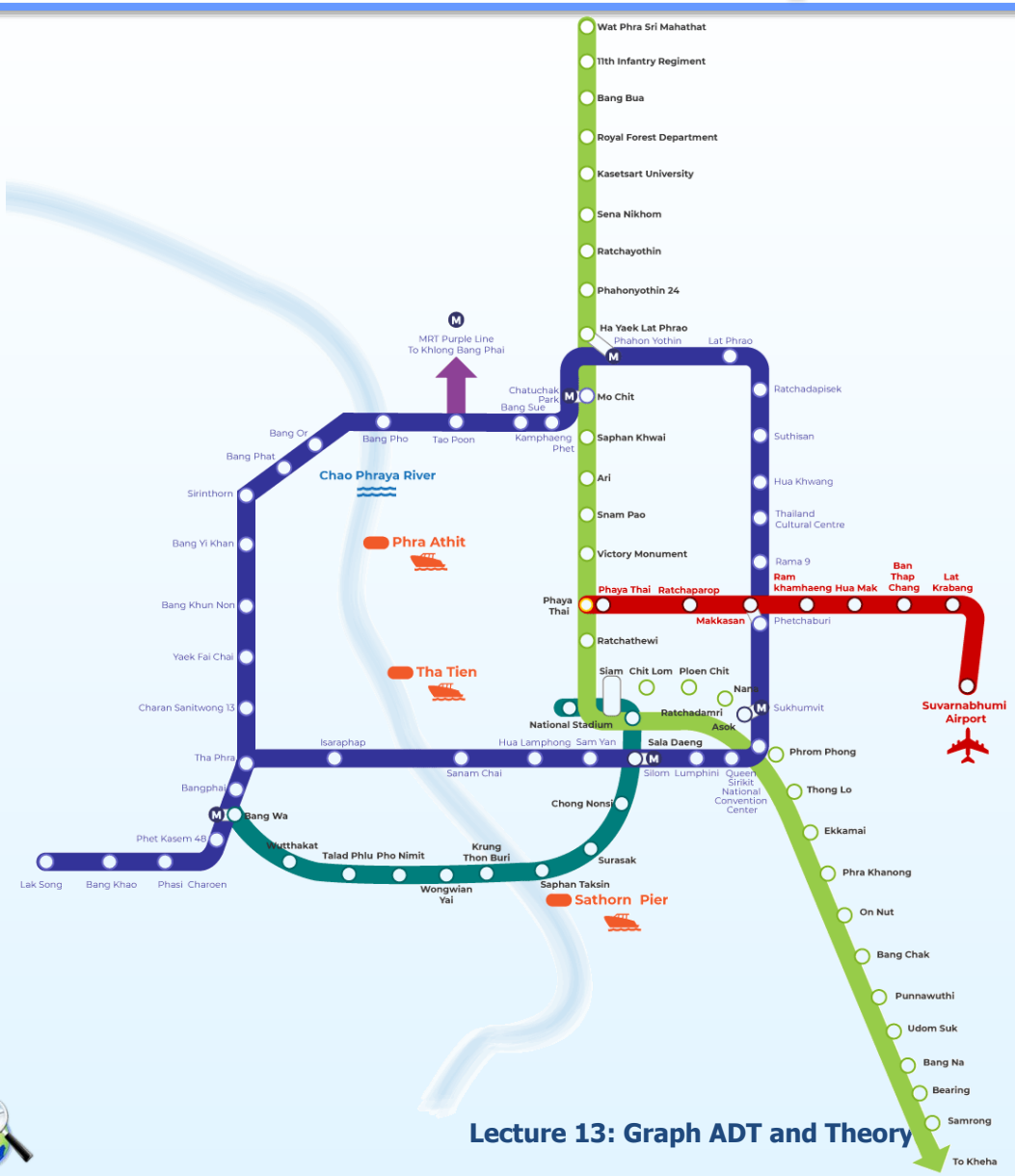
Information Transmission in a Computer Network



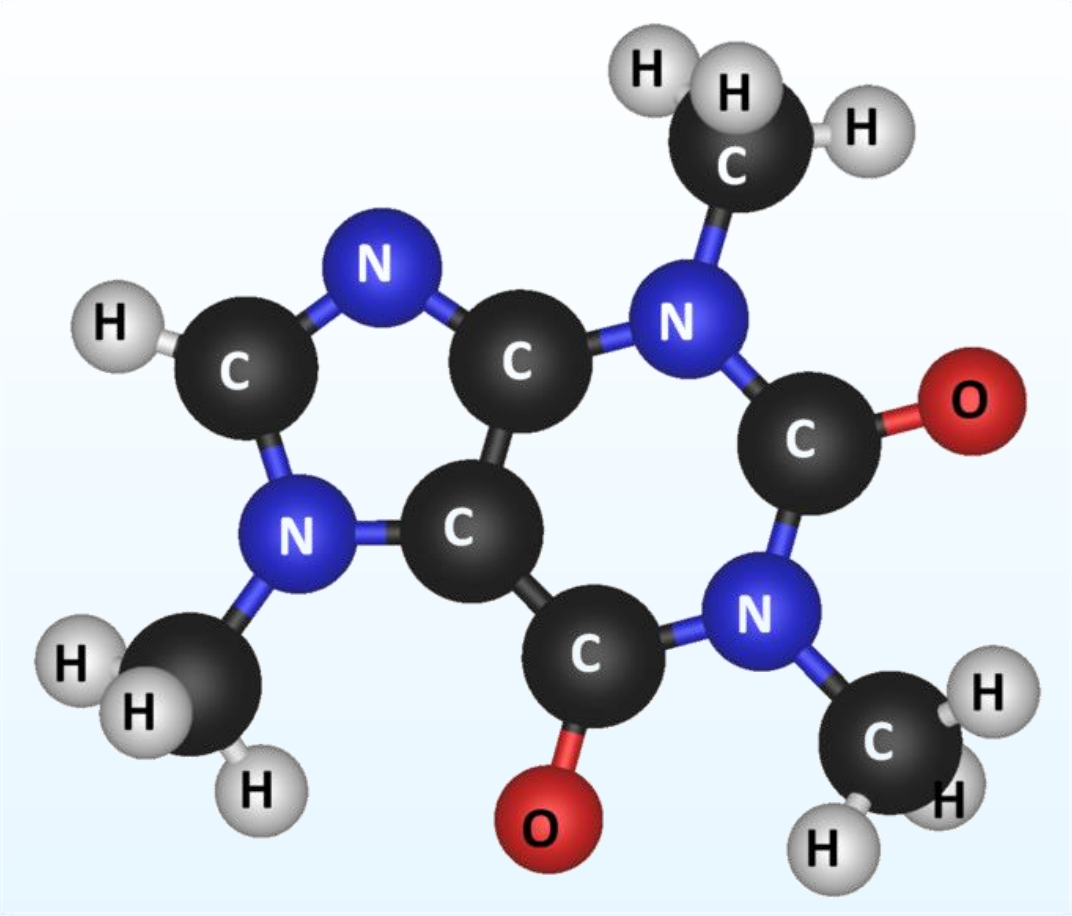
node = computer
edge = connection
weight = transmission rate

Bangkok BTS and MRT Map

node = station
edge = linkage



Molecules



node = atom

edge = bond

Graph Definition

- A graph is a collection of nodes and edges

- Formally,
  - A graph $G = (V, E)$ where
  - V is a set of vertices or nodes
  - $E \subseteq V \times V$ is a set of edges that connect vertices

Graph ADT

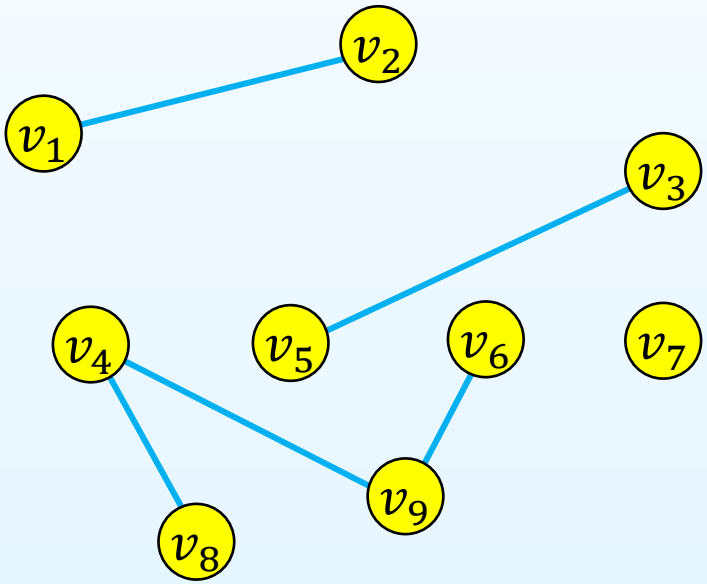
- Graph ADT describes a container storing an adjacency relation
- Operations include:
  - Inserting or removing a vertex (and all edges connecting that vertex)
  - Inserting or removing an edge
  - Querying
    - The number of vertices
    - The number of edges
    - The vertices adjacent to a given vertex
    - The question that "are two vertices adjacent?"
    - The question that "are two vertices connected?"
- The running time of these operations will depend on the representation

Graph Example

- Consider a collection of vertices $V = \{v_1, v_2, \ldots, v_9\}$ where $|V| = 9$
- Associated with these vertices are $|E| = 5$ edges

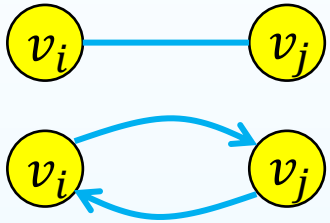
$$E = \{(v_1, v_2), (v_3, v_5), (v_4, v_8), (v_4, v_9), (v_6, v_9)\}$$

  - The pair (v_i, v_j) indicates that two vertices v_i and v_j are adjacent



Undirected and Directed Graphs

- If the order of edge pairs (v_i, v_j) does not matter, the graph is an undirected graph: $(v_i, v_j) = (v_j, v_i)$



- If the order of edge pairs (v_i, v_j) matters, the graph is a directed graph (or called a digraph): $(v_i, v_j) \neq (v_j, v_i)$

Maximum Number of Edges

- For an undirected graph, the maximum number of edges is

$$|E| \leq \binom{|V|}{2} = \frac{|V|(|V|-1)}{2} = O(|V|^2)$$

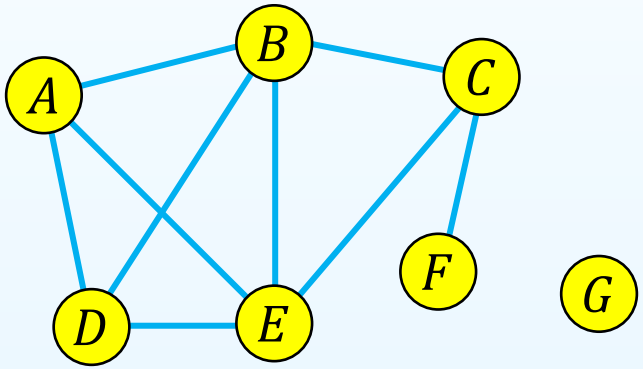
- For a directed graph, the maximum number of edges is

$$|E| \leq 2\binom{|V|}{2} = 2\frac{|V|(|V|-1)}{2} = O(|V|^2)$$

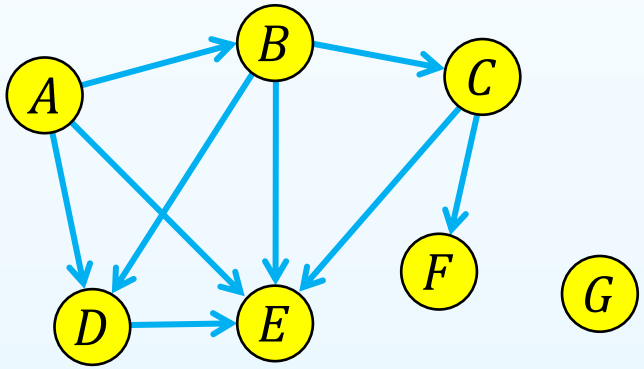
Graph Example

Given $G = (V, E)$ where $V = \{A, B, C, D, E, F, G\}$ and $E = \{(A, B), (A, D), (A, E), (B, C), (B, D), (B, E), (C, E), (C, F), (D, E)\}$

- If G is an undirected graph



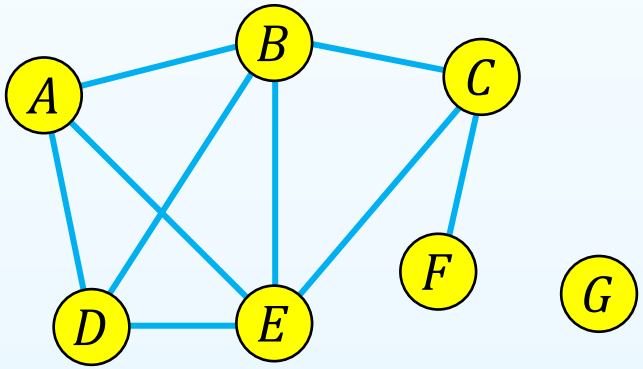
- If G is a directed graph



Degrees

- If G is an undirected graph

Degree: # of edges incident with



$\deg(A) = \deg(C) = \deg(D) = 3$

$\deg(B) = \deg(E) = 4$

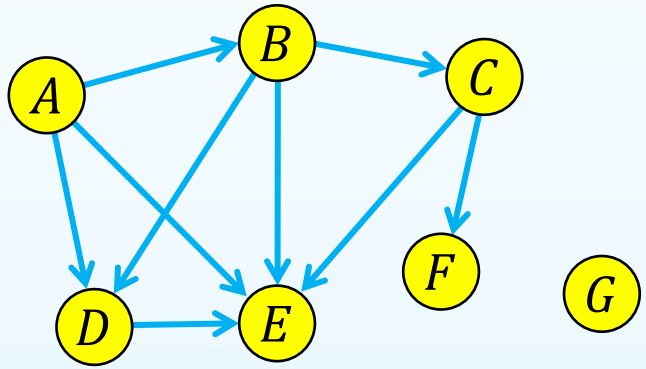
$\deg(F) = 1$

$\deg(G) = 0$

- If G is a directed graph

In-degree: # of edges with the terminal vertex

Out-degree: # of edges with the initial vertex



$\text{in\_deg}(A) = 0$ $\quad \text{out\_deg}(A) = 3$

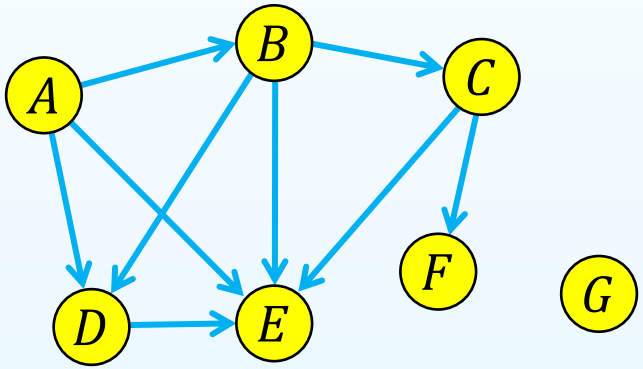
$\text{in\_deg}(B) = 1$ $\quad \text{out\_deg}(B) = 3$

$\text{in\_deg}(C) = 1$ $\quad \text{out\_deg}(C) = 2$

$\text{in\_deg}(D) = 2$ $\quad \text{out\_deg}(D) = 1$

Sources and Sinks

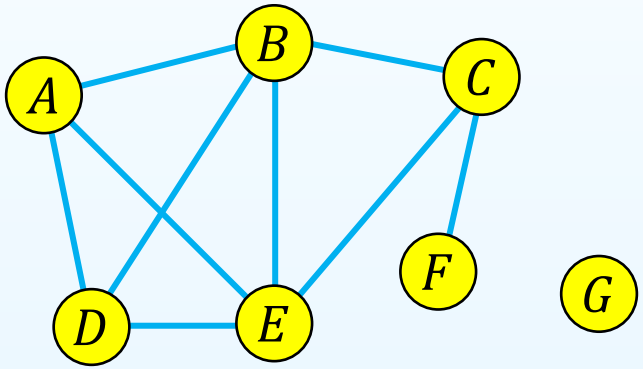
- For a directed graph,
  - Vertices with an in-degree of zero are described as sources
  - Vertices with an out-degree of zero are described as sinks



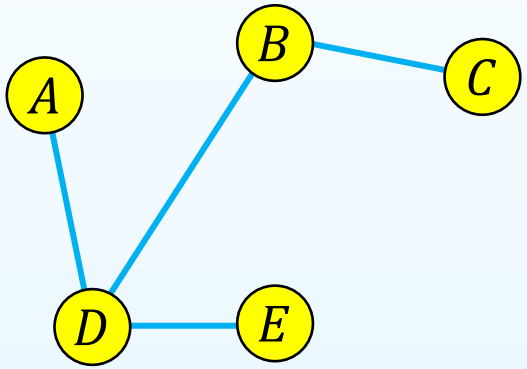
Sources: A and G
Sinks: E, F, and G

Subgraphs

A subgraph is a graph consisting of a subset of vertices and a subset of edges that connected those vertices in the original graph



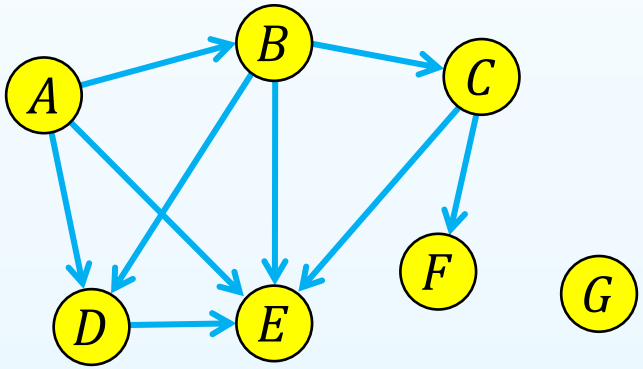
the original graph



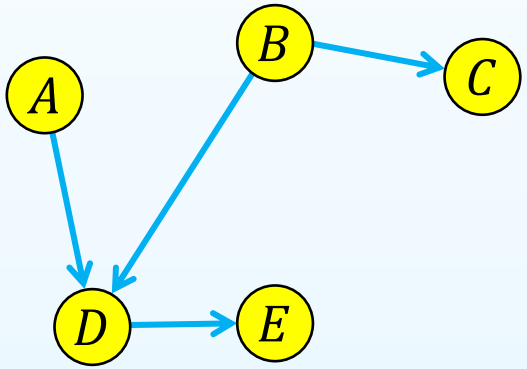
a subgraph

Subgraphs

A subgraph is a graph consisting of a subset of vertices and a subset of edges that connected those vertices in the original graph



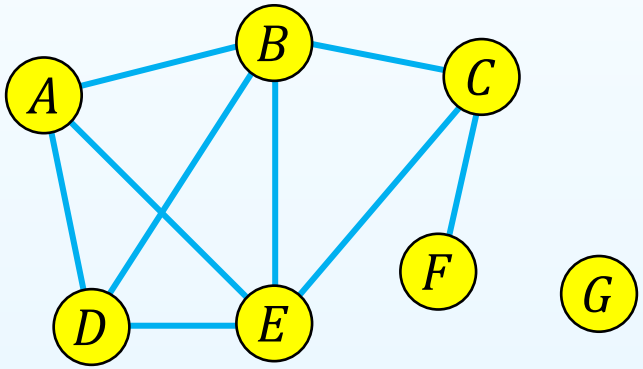
the original graph



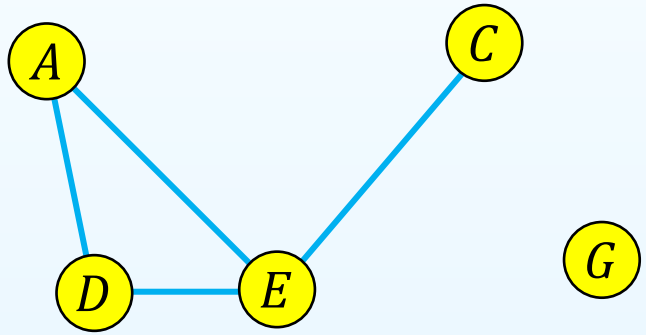
a subgraph

Vertex-Induced Subgraphs

A vertex-induced subgraph is a graph consisting of a subset of vertices where the edges are all edges in the original graph



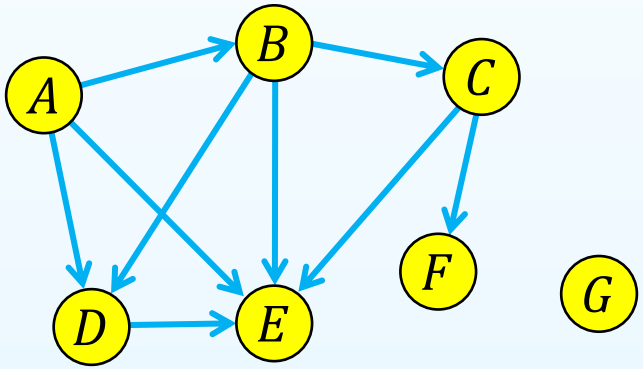
the original graph



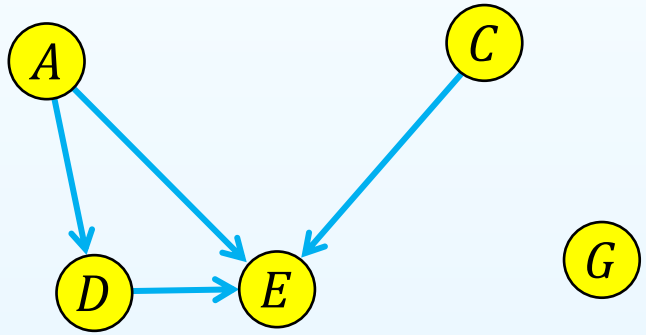
a vertex-induced subgraph

Vertex-Induced Subgraphs

A vertex-induced subgraph is a graph consisting of a subset of vertices where the edges are all edges in the original graph



the original graph

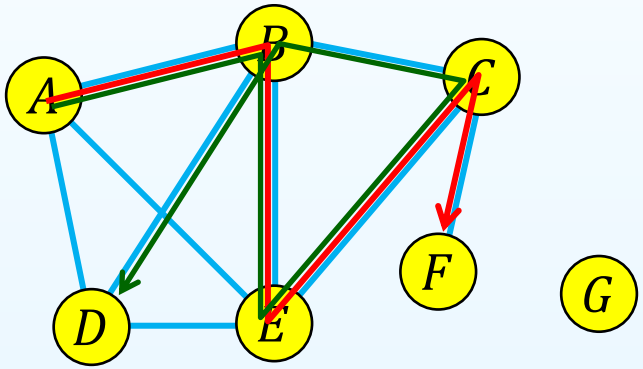


a vertex-induced subgraph

Paths

A path is an ordered sequence of vertices $\langle v_i, v_{i+1}, \ldots, v_k \rangle$ where (v_j, v_{j+1}) is an edge for $j = i, \ldots, k-1$

- If G is an undirected graph

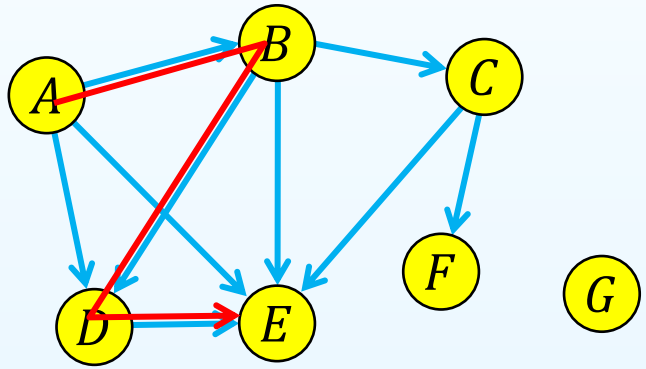


$\langle A, B, E, C, F \rangle$ is a path of length 4

$\langle A, B, E, C, B, D \rangle$ is a path of length 5

$\langle A \rangle$ is a path of length 0

- If G is a directed graph

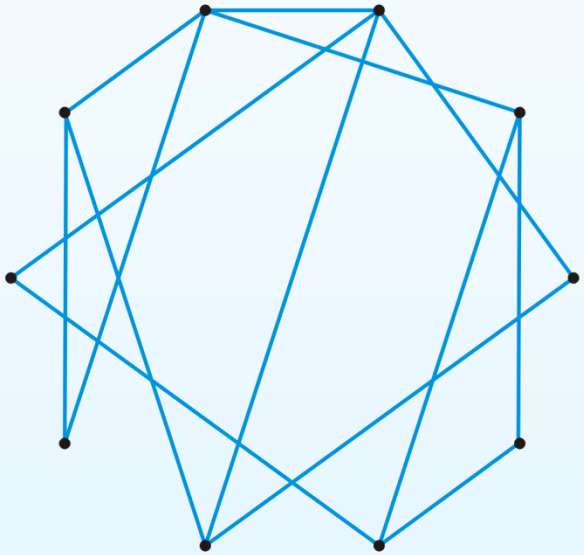


$\langle A, B, D, E \rangle$ is a path of length 3

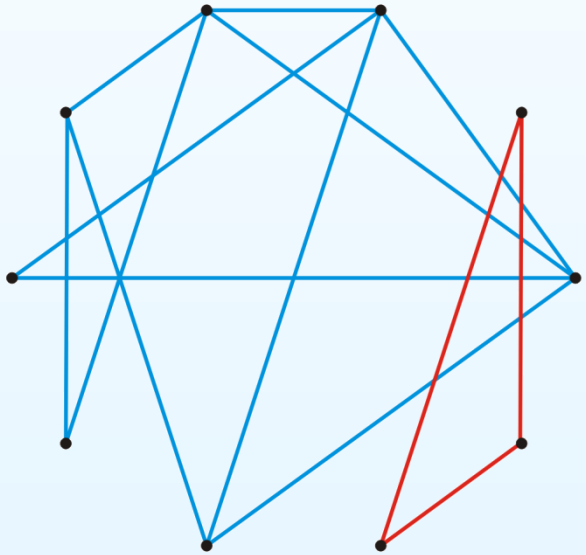
Connectivity

Two vertices v_i and v_j are said to be connected if there exists a path between them

A graph is connected if there exists a path between any two vertices



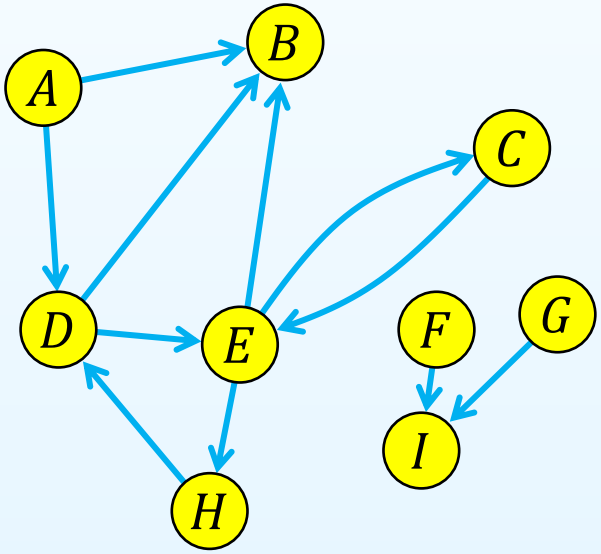
a connected graph



an unconnected graph

Strongly and Weakly Connectivity

- A graph is strongly connected if every pair of vertices (v_i and v_j) contains a path between each other
- A graph is weakly connected if there does not exist any path between any two pairs of vertices



Strongly connected subgraph:

$\{C, D, E, H\}$

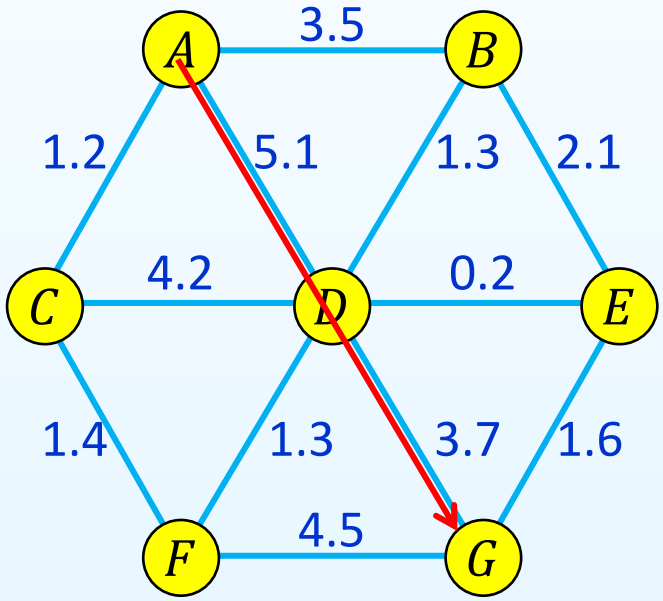
Weakly connected subgraph:

$\{A, B, C, D, E, H\}$

Weighted Graphs

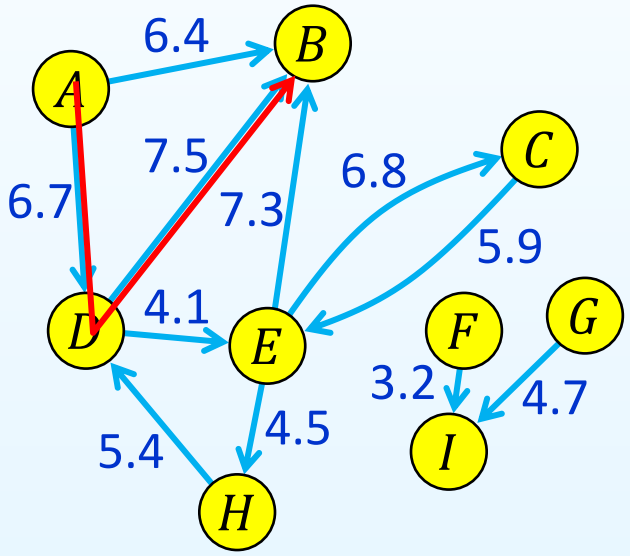
A weighted graph is a graph $G = (V, E, W)$ in which a weight $w_{ij} \in W$ is associated with each edge $(v_i, v_j) \in E$

- If G is an undirected graph



$\langle A, D, G \rangle$ is a path of length 8.8

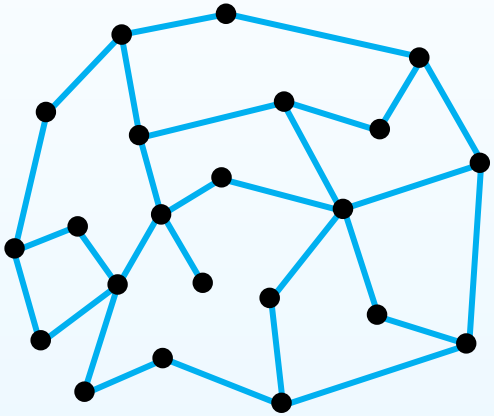
- If G is a directed graph



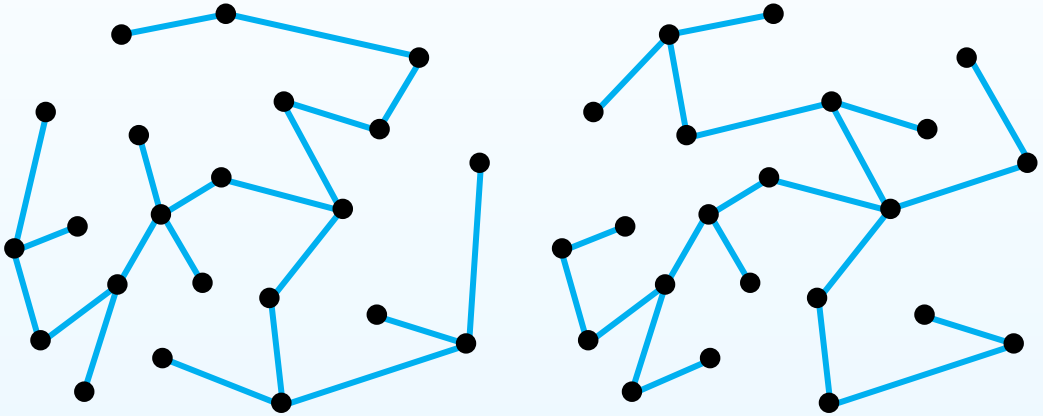
$\langle A, D, B \rangle$ is a path of length 14.2

Trees

- Trees are special cases of a graph
  - Each is connected, and
  - There is a unique path between any two vertices



the original graph

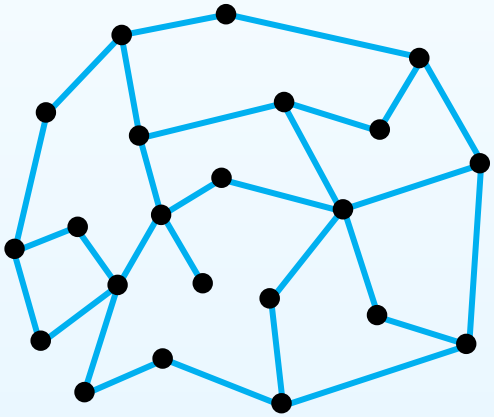


example of trees t_1 and t_2

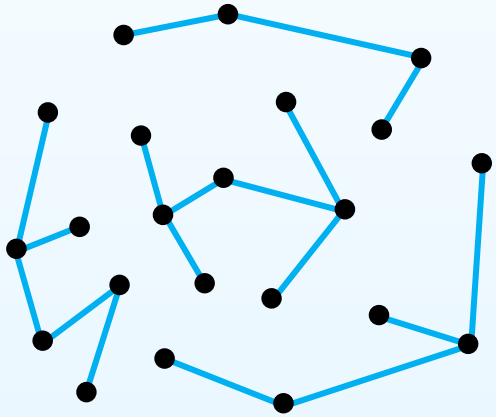
- **Consequences**
  - The number of edges is $|E| = |V| - 1$
  - The graph (tree) is acyclic
  - Adding one edge creates a cycle
  - Removing any one edge creates two disjoint non-empty subgraphs

Forests

- A forest is any graph that has no cycles
  - The number of edges is $|E| < |V|$
  - The number of trees is $|V| - |E|$
  - Removing any one edge adds one more tree to the forest



the original graph

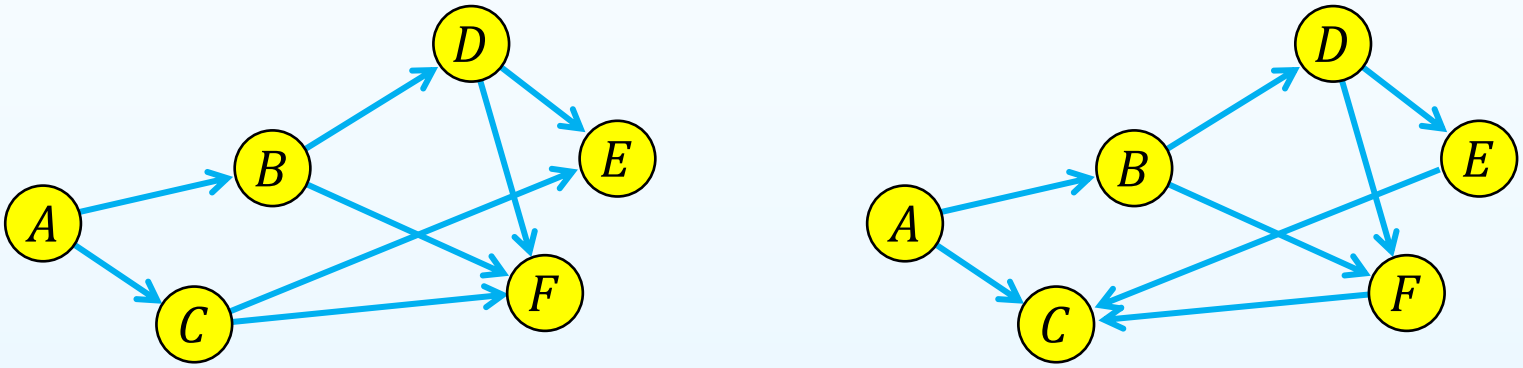


example of a forest with 4 trees

Directed Acyclic Graphs

A directed acyclic graph (DAG) is a directed graph which has no cycles

- Graphical representation of partial order on a finite number of elements



example of two DAGs

Note that adding the edge (C, A) will make them not DAG

Graph Representations

- Space and time are analyzed in terms of
  - Number of vertices ($|V|$) and
  - Number of edges ($|E|$)

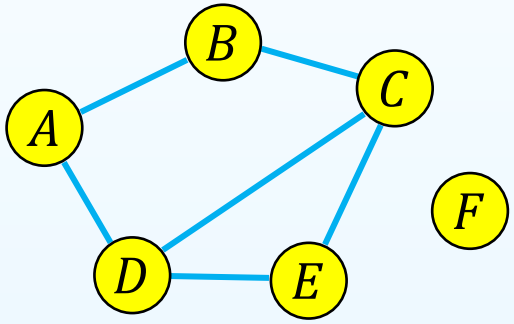
- There are two ways of representing graphs:
  - The adjacency matrix representation
  - The adjacency list representation

Adjacency Matrix

Require more memory but faster

- If G is an undirected graph

$$m_{ij} = \begin{cases} 1 & if\ (v_i, v_j) \in E \\ 0 & otherwise \end{cases}$$



$$M = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

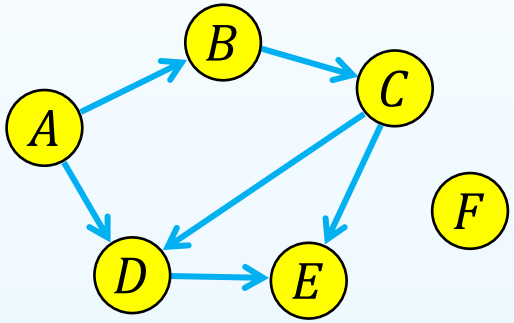
- Memory requirement is $\Theta(|V|^2)$
- Determining if v_i is adjacent to v_j is $\Theta(1)$
- Finding all neighbors of v_i is $\Theta(|V|)$

Adjacency Matrix

Require more memory but faster

- If G is a directed graph

$$m_{ij} = \begin{cases} 1 & if\ (v_i, v_j) \in E \\ 0 & otherwise \end{cases}$$



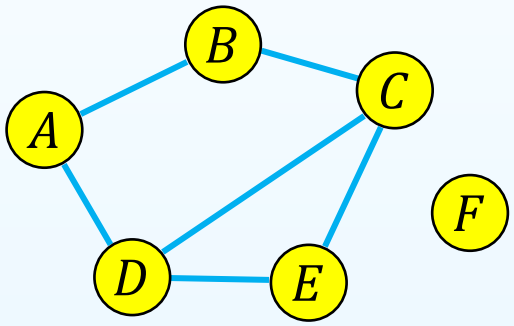
$$M = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

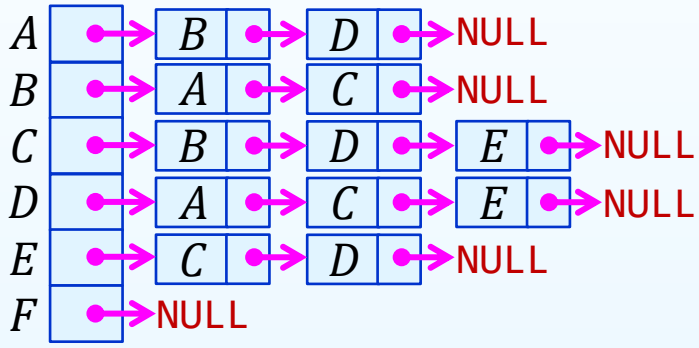
- Memory requirement is $\Theta(|V|^2)$
- Determining if v_i is adjacent to v_j is $\Theta(1)$
- Finding all neighbors of v_i is $\Theta(|V|)$

Adjacency List

Each vertex is associated with a list of its neighbors

- If G is an undirected graph



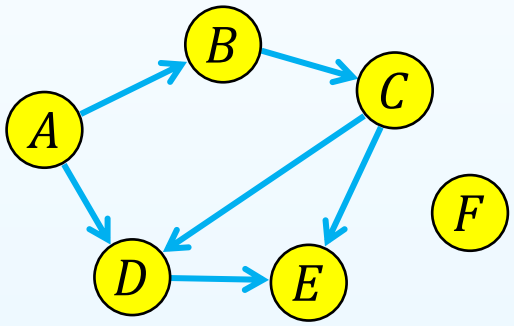


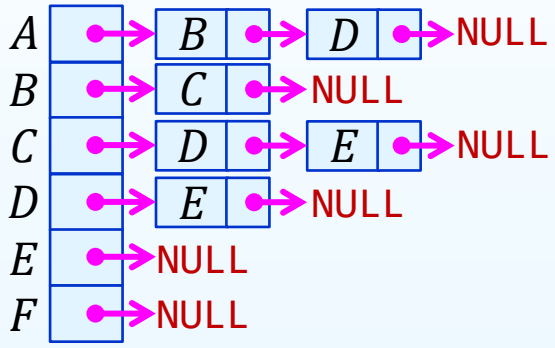
- Memory requirement is $\Theta(|V| + 2|E|) = \Theta(|V| + |E|)$
- On average, determining if v_i is adjacent to v_j is $\Theta(|E|/|V|)$
- On average, finding all neighbors of v_i is $\Theta(|E|/|V|)$

Adjacency List

Each vertex is associated with a list of its neighbors

- If G is a directed graph





- Memory requirement is $\Theta(|V| + |E|)$
- On average, determining if v_i is adjacent to v_j is $\Theta(|E|/|V|)$
- On average, finding all neighbors of v_i is $\Theta(|E|/|V|)$

Outline

- Graph ADT and Representations
- Graph Traversals
- Connectivity
- Bipartite Graphs
- Topological Sort

Strategies

- Traversals of graphs are also called searches

- We can use either
  - Breadth-first traversal requiring a queue, or
  - Depth-first traversal requiring a stack

- We will have to track which vertices have been visited, requiring $\Theta(|V|)$ memory

Breadth-First Traversal

Procedures:

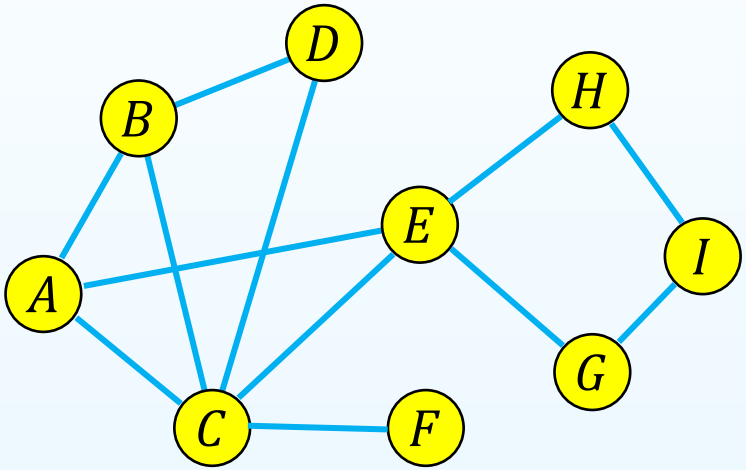
- Choose a vertex, mark it as visited and insert into queue
- While the queue is not empty
  - Remove the front vertex v from the queue
  - For each vertex adjacent to v that has not been visited
    - Mark it visited, and
    - Insert it into the queue

Note that if the queue is empty and there are no unvisited vertices, the graph is connected

The size of the queue is $O(|V|)$

Example

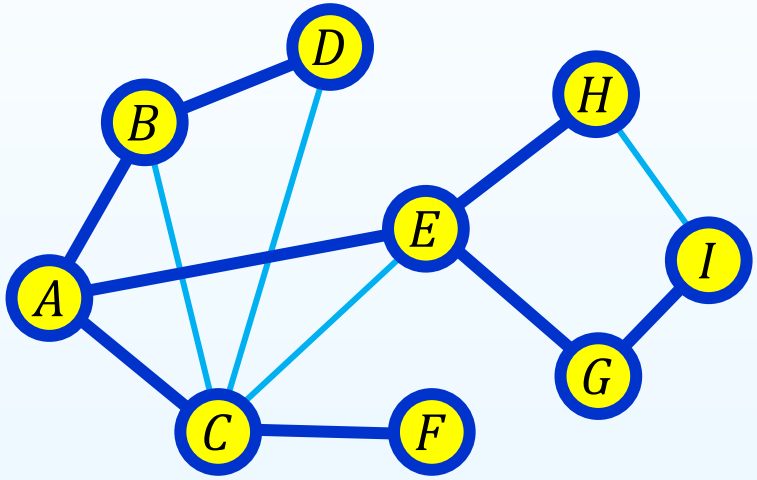
Consider this graph:



Example: Breadth-First Traversal

Consider this graph:

- Start with the vertex A



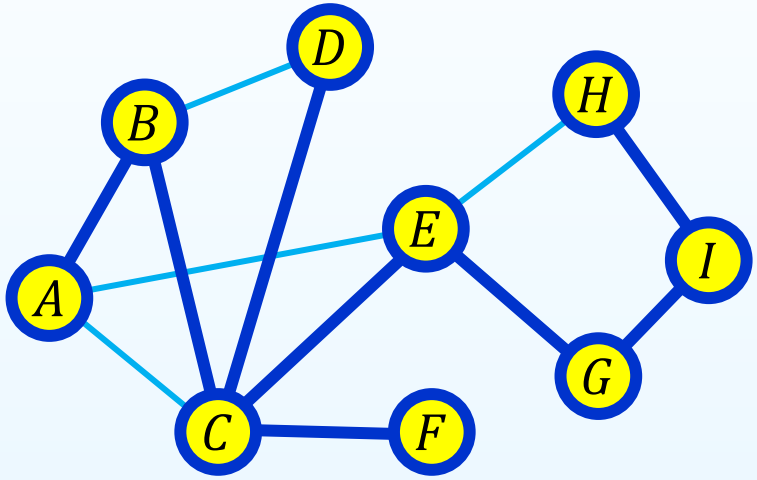
Output: $A \quad B \quad C \quad E \quad D \quad F \quad G \quad H \quad I$

Time complexity: $\Theta(|V| + |E|)$

Example: Depth-First Traversal

Consider this graph:

- Start with the vertex A



Output: $A \quad B \quad C \quad D \quad E \quad G \quad I \quad H \quad F$

Time complexity: $\Theta(|V| + |E|)$

Applications

- Determining connectivity and finding connected subgraphs
- Determining the path length from one vertex to all others
- Testing if a graph is bipartite
- Determining maximum flow
- …

Outline

- Graph ADT and Representations
- Graph Traversals
- Connectivity
- Bipartite Graphs
- Topological Sort

Connectivity

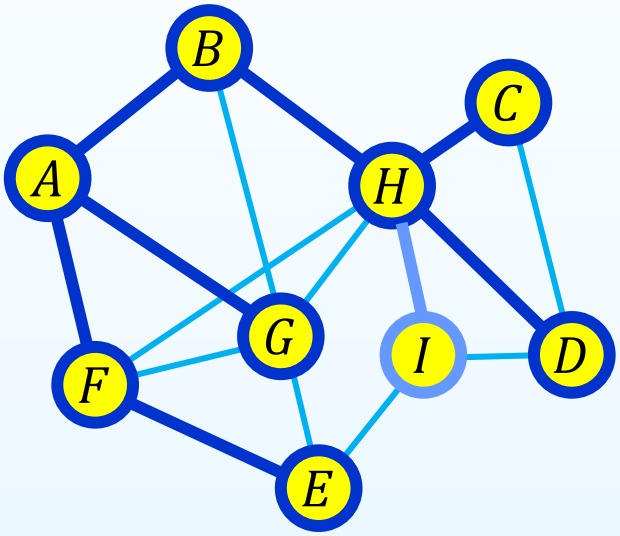
- Determine whether one vertex is connected to another
  - v_i is connected to v_j if there is a path from v_i to v_j

- Strategy:
  - Perform a breadth-first traversal starting at v_i
  - During the traversal,
    - If the vertex v_j ever found to be adjacent to any vertex in front of the queue, return true
    - Otherwise, if the queue is empty, return false

Determining Connectivity

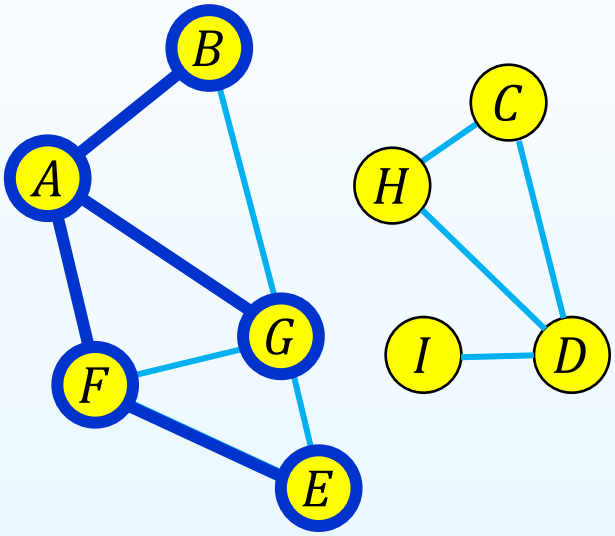
Consider these graphs:

- Is A connected to D?



BFS: $A\ B\ F\ G\ H\ E\ C\ D\ I$

found here!



BFS: $A\ B\ F\ G\ E$

NOT found

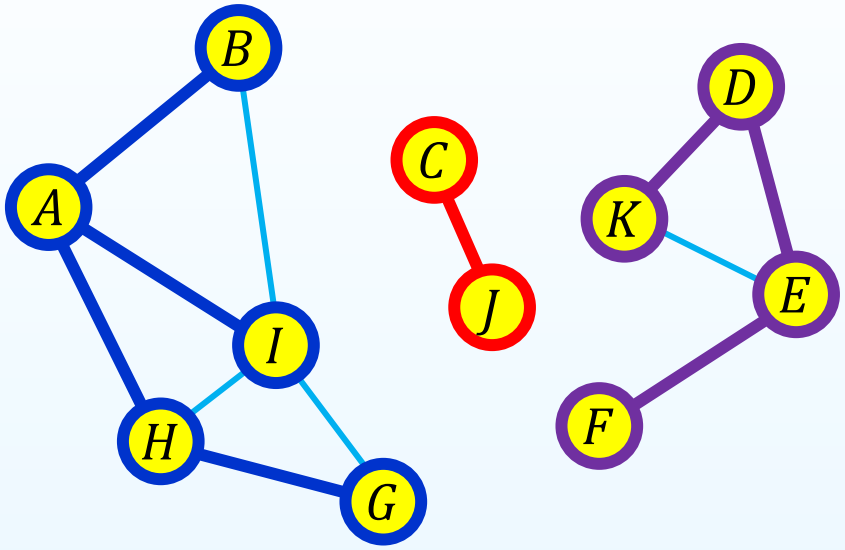
Connected Components

- Previously, if we continued the traversal, we would find all vertices that are connected to A

- Suppose we want to partition the vertices into connected subgraphs
  - While there are unvisited vertices in the graph
    - Select an unvisited vertex and perform a traversal on that vertex
    - Each vertex that is visited in that traversal is added to the set initially containing the initial unvisited vertex
  - Continue until all vertices are visited

Connected Components

Consider this graph:



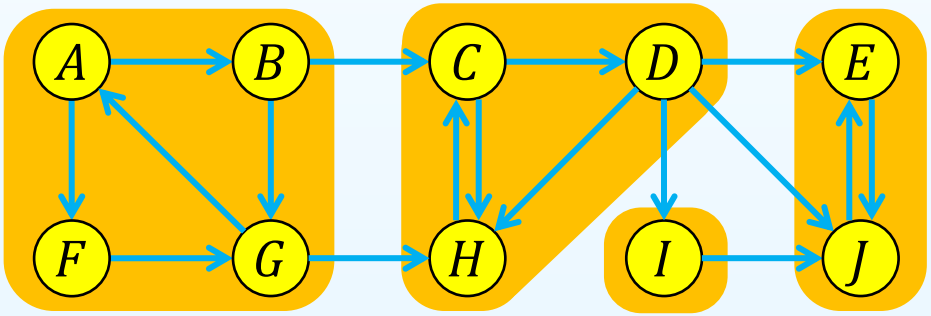
A	B	C	D	E	F	G	H	I	J	K
1	2	6	8	9	11	5	3	4	7	10

There are three connected subgraphs:

$\{A,B,G,H,I\}$, $\{C,J\}$, and $\{D,E,F,K\}$

Strongly Connected Components

- A directed graph is strongly connected if there is a path between all pairs of vertices
- A strongly connected component (SCC) of a directed graph is a maximal strongly connected subgraph



Strongly Connected Components

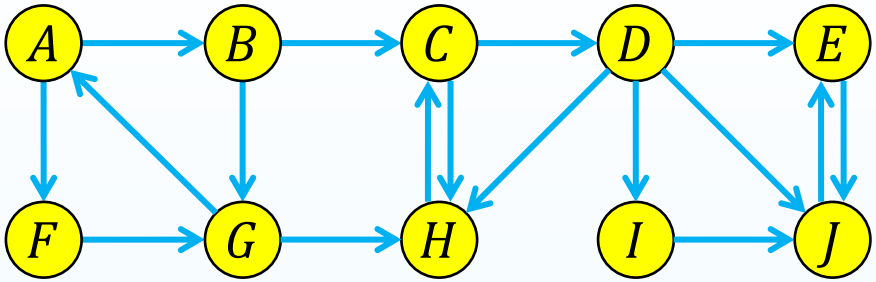
Algorithm:

1. Call DFS(G) to compute exiting times for each vertex
2. Compute G^T
3. Call DFS(G^T), but in the main loop of DFS, consider the vertices in order of decreasing times as computed in (1)
4. Output the vertices of each tree in the depth-first forest of (3) as a separate strongly connected components

The algorithm takes $O(|V| + |E|)$ time

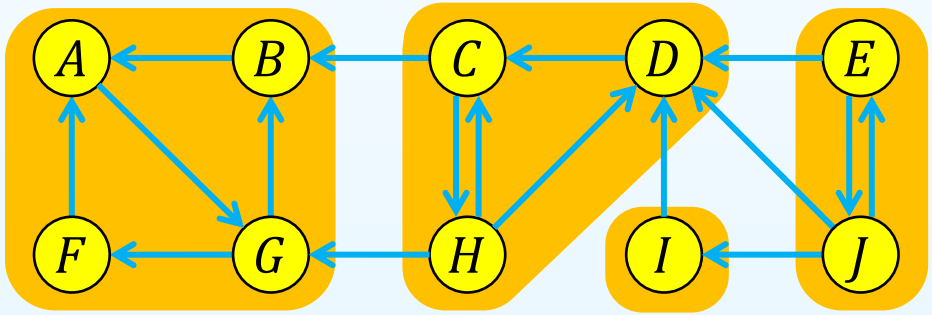
Strongly Connected Components

Step 1:



A		B		C		D		E		F		G		H		I		J	
1	20	2	17	3	14	4	13	5	8	18	19	15	16	9	10	11	12	6	7

Steps 2-4:



A		B		C		D		E		F		G		H		I		J	
1	8	3	4	9	14	11	12	17	20	5	6	2	7	10	13	15	16	18	19

Outline

- Graph ADT and Representations
- Graph Traversals
- Connectivity
- Bipartite Graphs
- Topological Sort

Definition

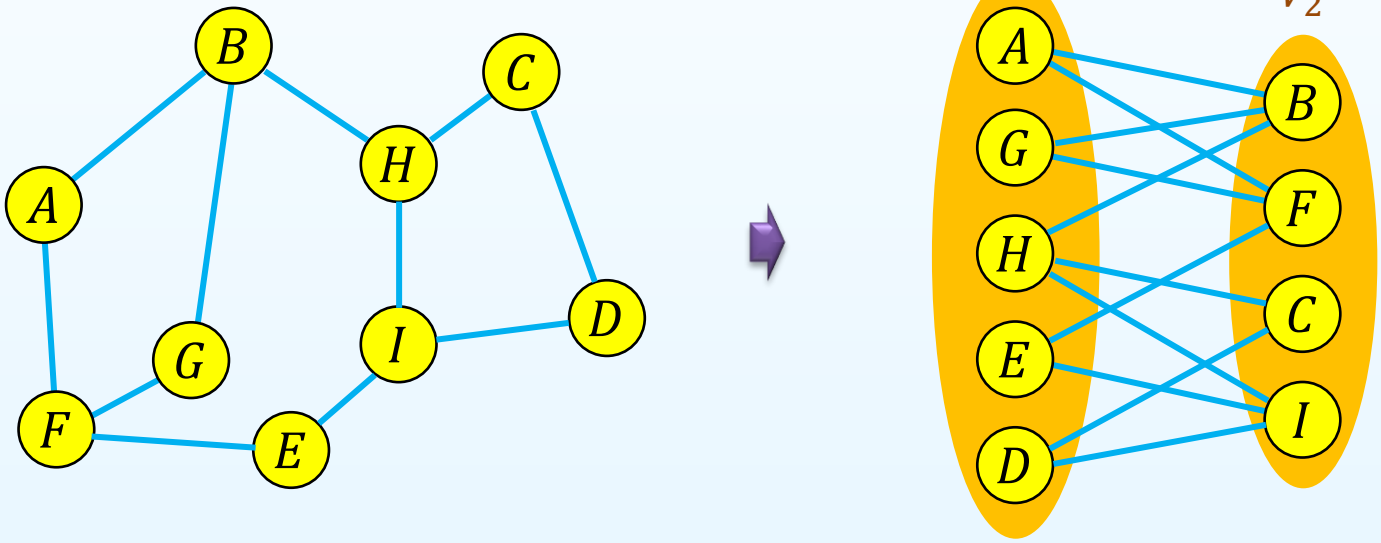
A bipartite graph is a graph where the vertices V can be divided into two disjoint sets V_1 and V_2 such that

- Every edge has one vertex in V_1 and the other in V_2
- In other words, no edges connects two vertices in the same set

Bipartite Graphs

Consider this graph:

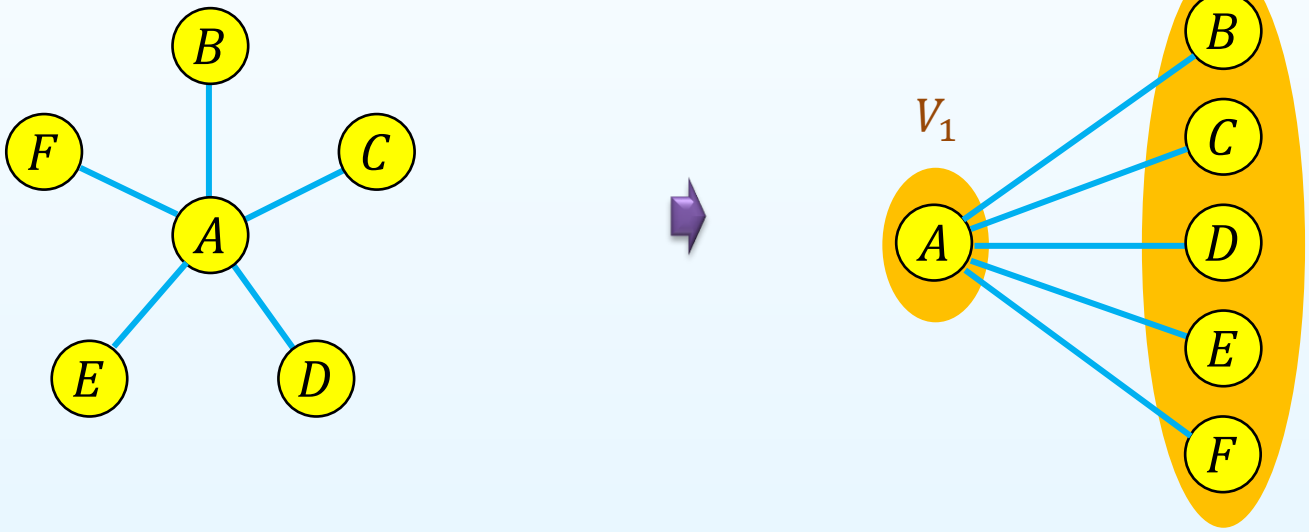
- Is it a bipartite graph?
  - ➡ **YES**, with a little work to decompose the vertices



Bipartite Graphs

Consider this graph:

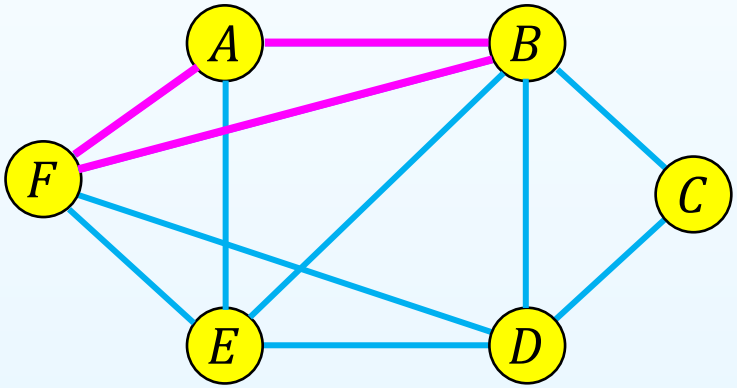
- Is it a bipartite graph?
  - ➡ **YES**, with a little work to decompose the vertices



Bipartite Graphs

Consider this graph:

- Is it a bipartite graph?
  - ➡ **NO**, we cannot divide $\{A,B,F\}$, for instance



Bipartite Graphs

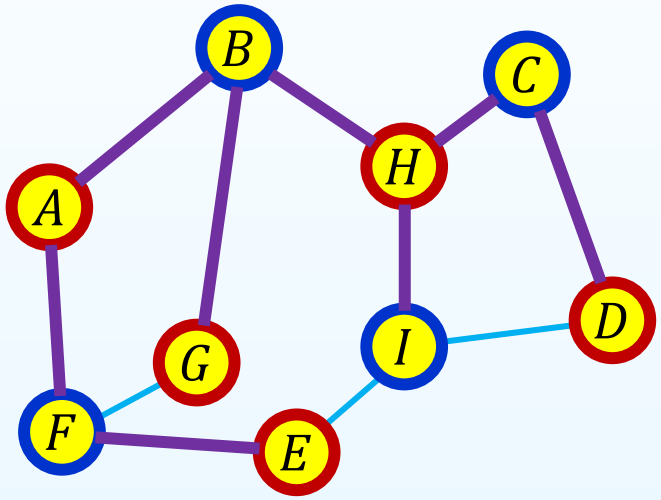
Algorithm: using a breadth-first traversal

- Choose a vertex, mark it belonging to V_1 and insert in into a queue
- While the queue is not empty
  - Dequeue the front vertex v, and
  - Any adjacent vertices that are already marked must belong to the set not containing v, otherwise, the graph is not bipartite (we are done); while
  - Any unmarked adjacent vertices are marked as belonging to the other set and they are inserted into the queue
- If the queue is empty, the graph is bipartite

Bipartite Graphs

Consider the previous graph:

- We will use colors to distinguish the two sets



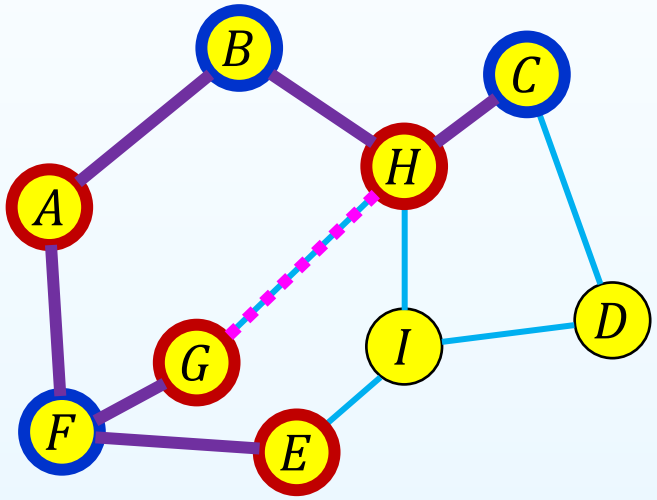
BFS: $A\ B\ F\ G\ H\ E\ C\ I\ D$

OK, this graph is bipartite

Bipartite Graphs

Consider the previous graph:

- We will use colors to distinguish the two sets

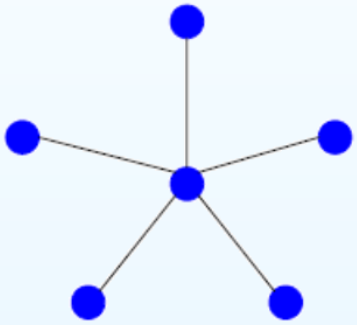


BFS: $A\ B\ F\ H\ E\ G\ C$

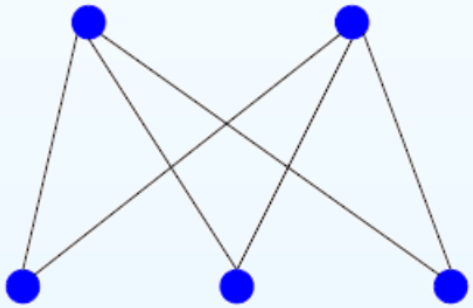
This graph is NOT bipartite

Complete Bipartite Graphs $K_{m,n}$

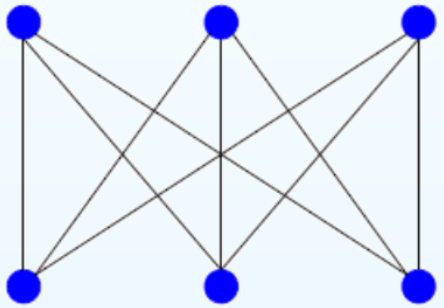
- Vertex set partitioned into two subsets of size m and n
- All vertices in one subset are connected to all vertices in the other subset



$K_{1,5}$



$K_{2,3}$



$K_{3,3}$

Outline

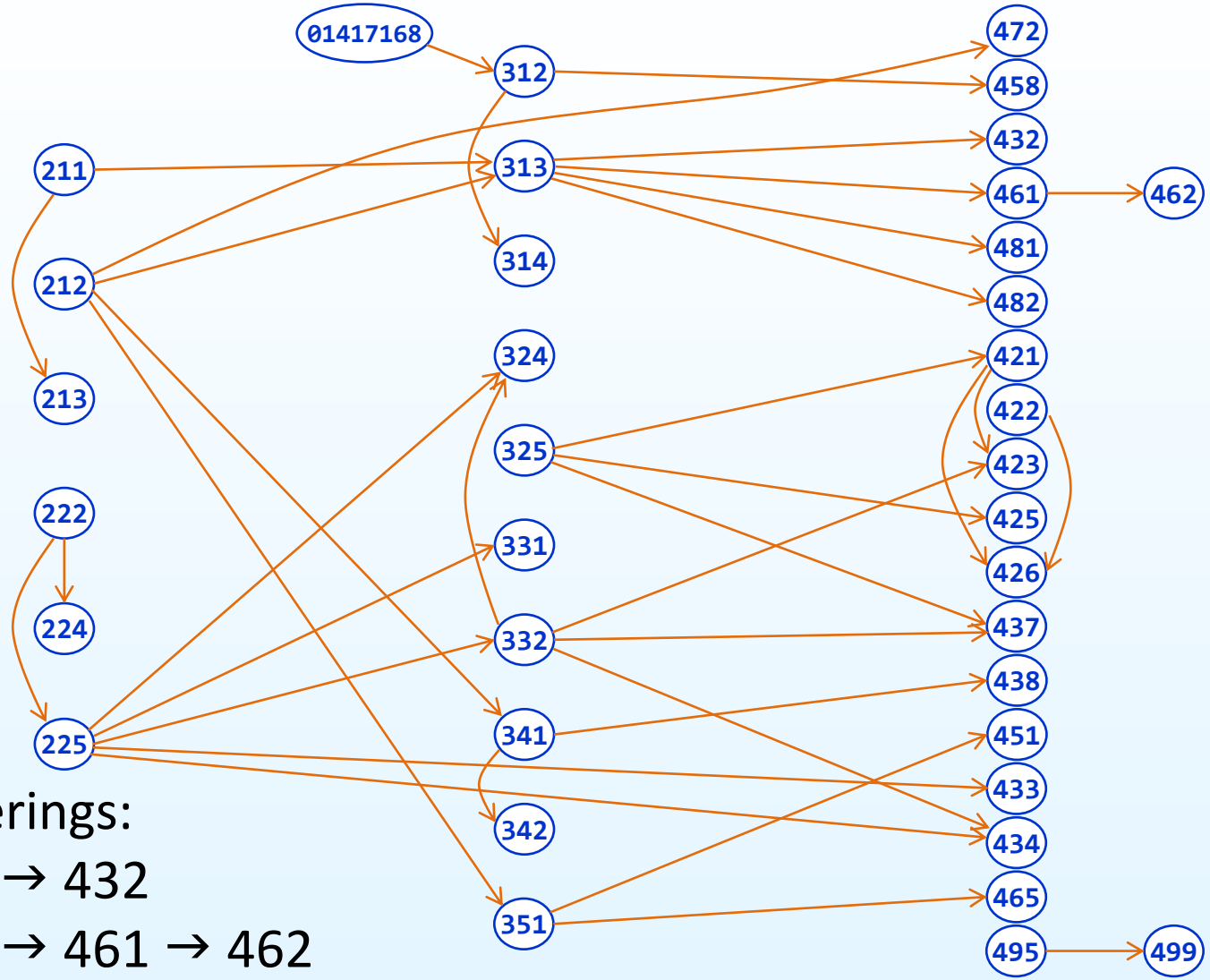
- Graph ADT and Representations
- Graph Traversals
- Connectivity
- Bipartite Graphs
- Topological Sort

Motivation

- Given a set of tasks with dependencies
  - Is there an order in which we can complete the tasks?

- Dependencies form a partial ordering
  - A partial ordering on a finite number of objects can be represented as a directed acyclic graph (DAG)

Example: CPE Course Prerequisites



Partial orderings:

212 → 313 → 432

212 → 313 → 461 → 462

Definition

Restriction of paths in DAGs

- Given two different vertices v_i and v_j, there cannot both be a path from v_i to v_j and a path from v_j to v_i

Topological sort

- A topological sort of the vertices in a DAG is an ordering

$$v_1, v_2, v_3, \ldots, v_n$$

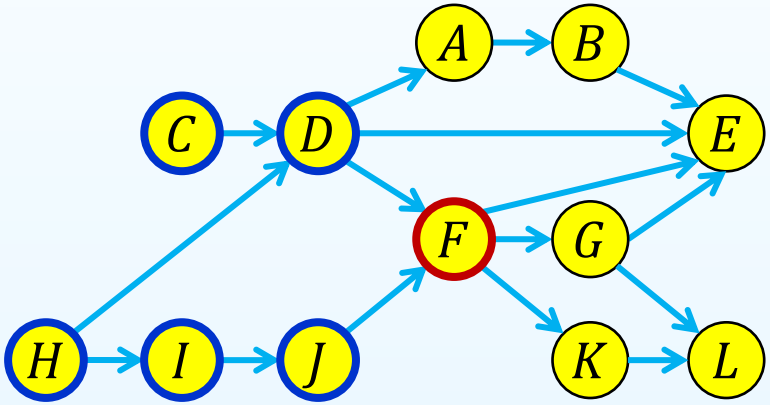
such that if there is a path from v_i to v_j then v_i appears before v_j

Example

Consider this DAG:

- A topological sort is

$$C, H, D, I, A, J, B, F, G, K, E, L$$



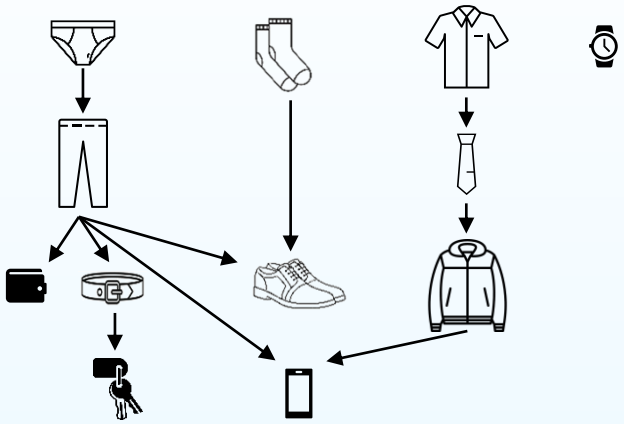
Note that

- There are paths from C, D, H, I, and J to F, so all these must come before F in a topological sort
- Clearly, this sorting need not be unique

Applications

Suppose you want to dress up to dinner

- You must wear the following:
  - jacket, shirt, briefs, pants, socks, tie, etc.
- There are certain constraints:
  - The pants should go on after the briefs
  - Socks are put on before shoes



- One topological sort is

  briefs, pants, wallet, belt, keys, socks, shoes, shirt, tie, jacket, phone, watch

- A more reasonable topological sort is

  briefs, socks, pants, shirt, belt, keys, tie, jacket, wallet, phone, watch, shoes

Topological Sort

Algorithm:

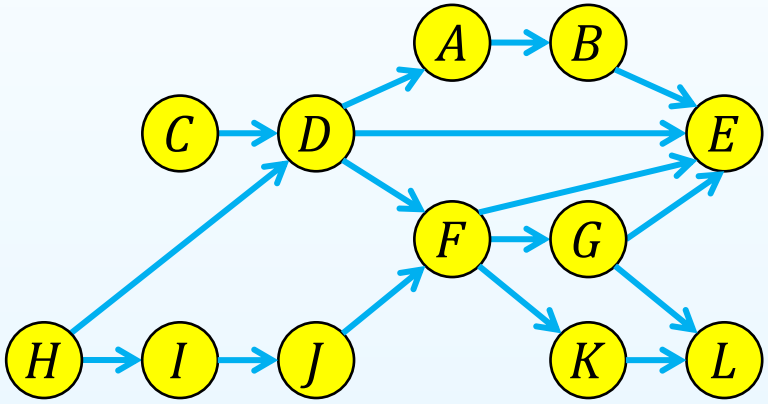
Given a graph, iterate the followings $|V|$ times

- Choose a vertex v that has in-degree zero
- Let v be the next vertex in our topological sort
- Remove v and all edges connected to it

Example

Consider the previous DAG:

- First, create a table of in-degrees of each vertex
- Then, create a queue

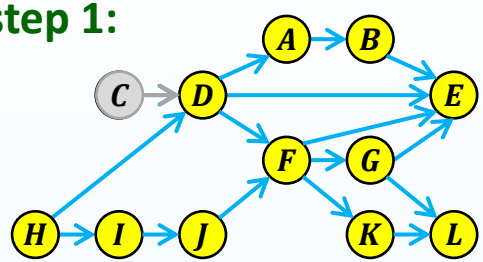


	A	B	C	D	E	F	G	H	I	J	K	L
in-degree	1	1	0	2	4	2	1	0	1	1	1	2

queue	C	H										

Example

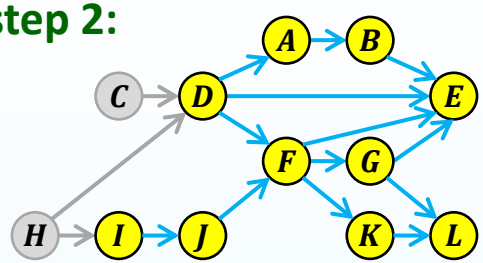
step 1:



	A	B	C	D	E	F	G	H	I	J	K	L
in-degree	1	1	0	1	4	2	1	0	1	1	1	2
queue	H											

topological sort is
C

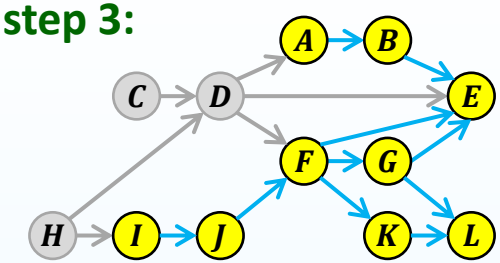
step 2:



	A	B	C	D	E	F	G	H	I	J	K	L
in-degree	1	1	0	0	4	2	1	0	0	1	1	2
queue	D	I										

topological sort is
C H

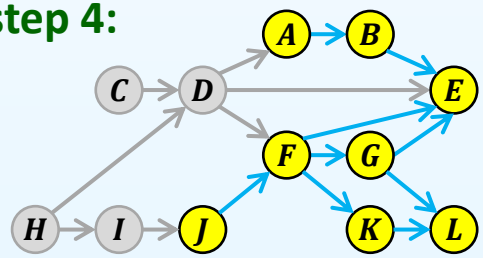
step 3:



	A	B	C	D	E	F	G	H	I	J	K	L
in-degree	0	1	0	0	3	1	1	0	0	1	1	2
queue	I	A										

topological sort is
C H D

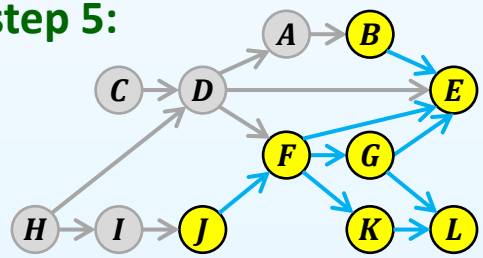
step 4:



	A	B	C	D	E	F	G	H	I	J	K	L
in-degree	0	1	0	0	3	1	1	0	0	0	1	2
queue	A	J										

topological sort is
C H D I

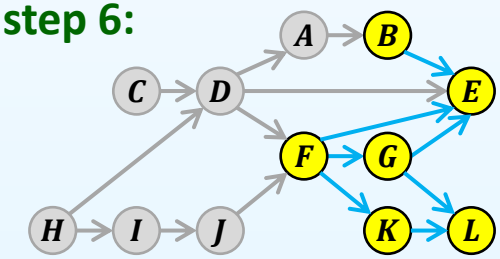
step 5:



	A	B	C	D	E	F	G	H	I	J	K	L
in-degree	0	0	0	0	3	1	1	0	0	0	1	2
queue	J	B										

topological sort is
C H D I A

step 6:

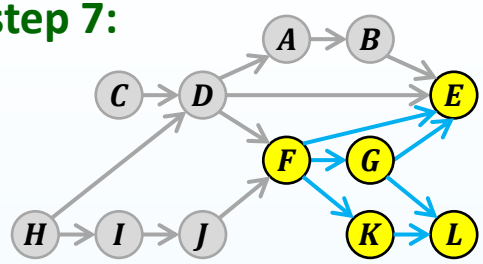


	A	B	C	D	E	F	G	H	I	J	K	L
in-degree	0	0	0	0	3	0	1	0	0	0	1	2
queue	B	F										

topological sort is
C H D I A J

Example

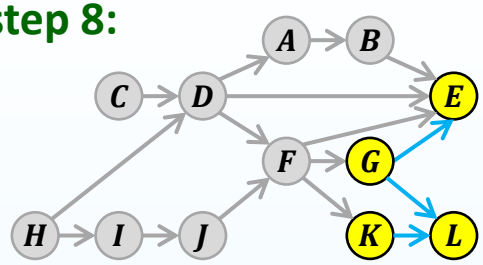
step 7:



	A	B	C	D	E	F	G	H	I	J	K	L
in-degree	0	0	0	0	2	0	1	0	0	0	1	2
queue	F											

topological sort is
C H D I A J B

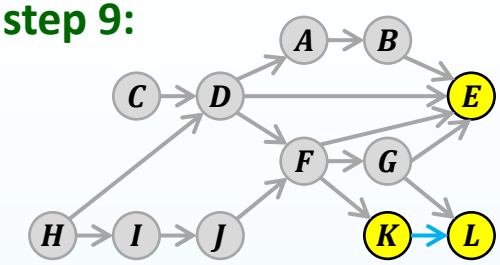
step 8:



	A	B	C	D	E	F	G	H	I	J	K	L
in-degree	0	0	0	0	1	0	0	0	0	0	0	2
queue	G	K										

topological sort is
C H D I A J B F

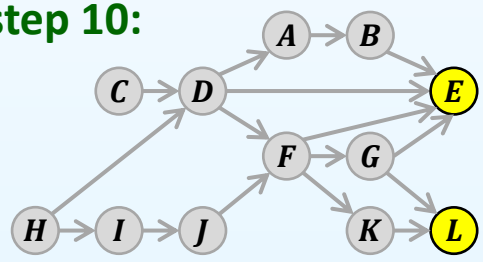
step 9:



	A	B	C	D	E	F	G	H	I	J	K	L
in-degree	0	0	0	0	0	0	0	0	0	0	0	1
queue	K	E										

topological sort is
C H D I A J B F G

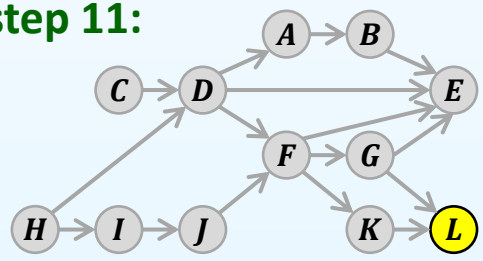
step 10:



	A	B	C	D	E	F	G	H	I	J	K	L
in-degree	0	0	0	0	0	0	0	0	0	0	0	0
queue	E	L										

topological sort is
C H D I A J B F G K

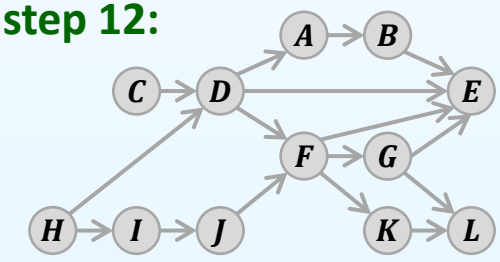
step 11:



	A	B	C	D	E	F	G	H	I	J	K	L
in-degree	0	0	0	0	0	0	0	0	0	0	0	0
queue	L											

topological sort is
C H D I A J B F G K E

step 12:



	A	B	C	D	E	F	G	H	I	J	K	L
in-degree	0	0	0	0	0	0	0	0	0	0	0	0
queue												

topological sort is
C H D I A J B F G K E L

Complexity Analysis

- Initialize in-degree array
  - ➡ $\Theta(|E|)$
- Initialize queue with zero in-degree vertices
  - ➡ $\Theta(|V|)$
- Dequeue and output vertex
  - All vertices will be enqueued and dequeued once, each take $\Theta(1)$
  - ➡ $\Theta(|V|)$
- Reduce in-degree of all vertices adjacent to a vertex and enqueue any zero in-degree vertices
  - All edges will decrease in-degree of adjacent once, each take $\Theta(1)$
  - ➡ $\Theta(|E|)$

Finally, the topological sort takes $\Theta(|V| + |E|)$ time

Any Question?