SLIIT UNI
THE KNOWLEDGE UNIVERSITY

Sri Lanka Institute of Information Technology

Faculty of Computing

IT1160 - Discrete Mathematics

Dr. Mahima Weerasinghe

Year 01 and Semester 02

Introduction to Algorithms

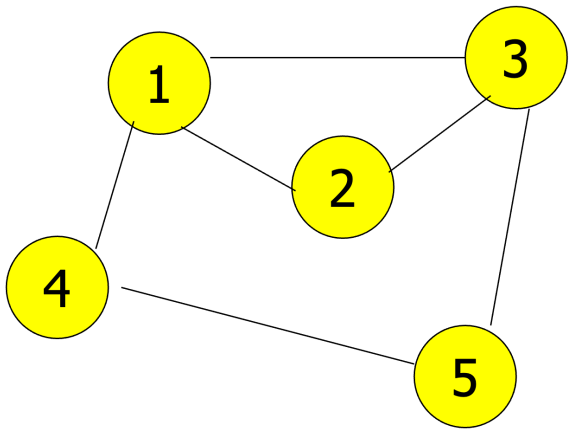
Graphs Algorithms

Today's Lecture

- Graph
- Applications.
- Terminology.
- Representation.
- Searching
  - DFS
  - BFS

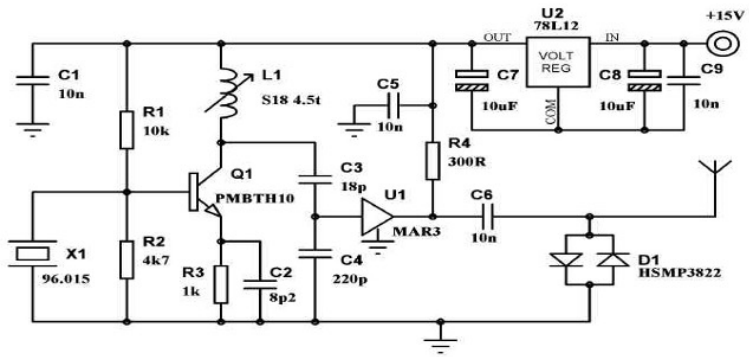
Graph Introduction

- Graph is a data structure which consists of set of vertices and set of edges.
- Graphs are a pervasive data structure in computer science, and algorithms for working with them are fundamental to the field. There are hundreds of interesting computational problems defined in terms of graphs. In this part, we touch on a few of the more significant ones.



Applications.

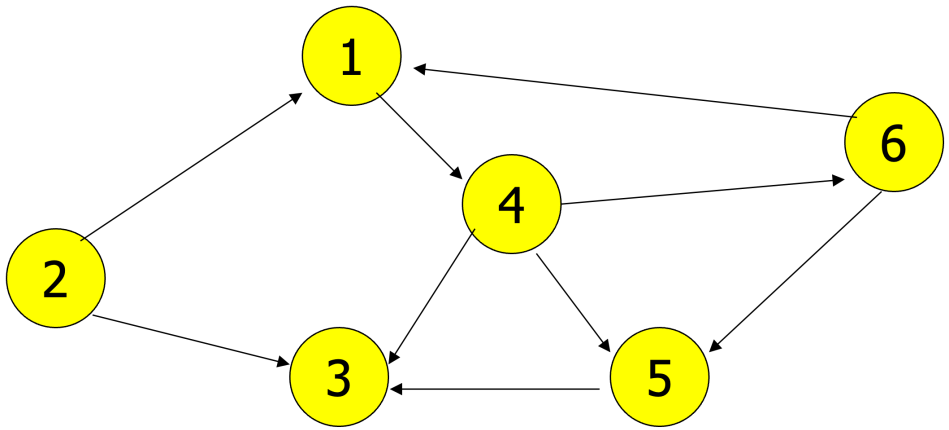
- A network of roads : with cities as vertices and roads between cities as edges.
- An electronic circuit : with junctions as vertices and components as edges.



Flights between cities : with cities as the vertices and flight from one city to another as edges.

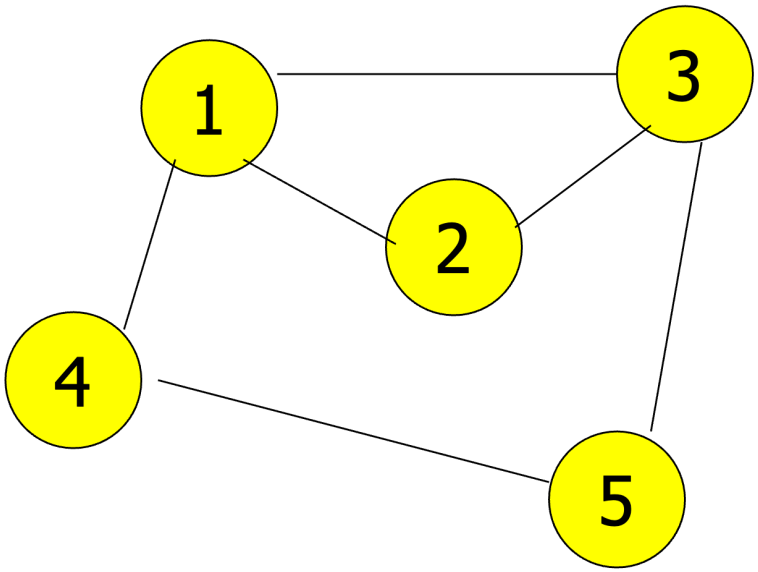
Graphs categorization

- Directed Graphs (Digraph) :
  All the edges have direction.



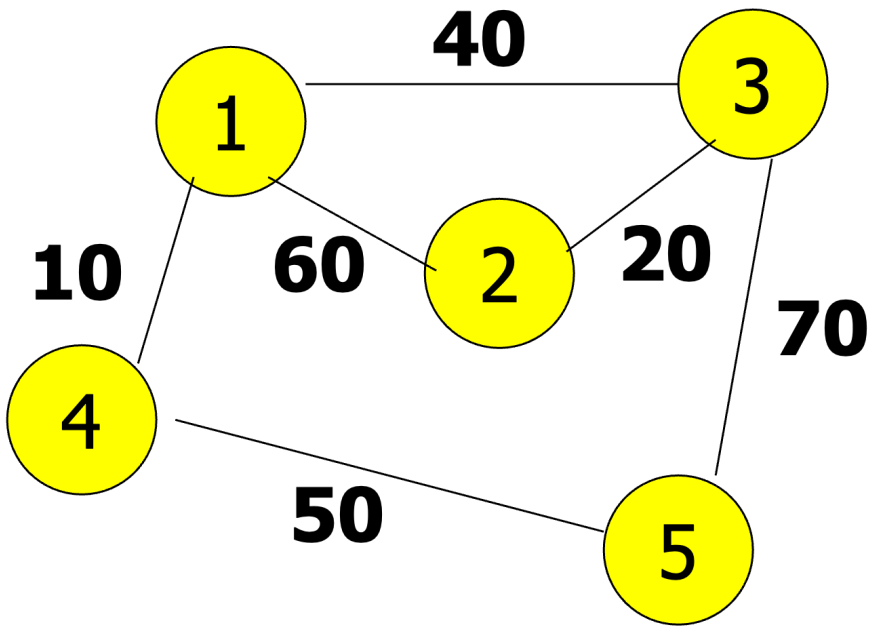
Graphs categorization

- Undirected Graphs:
  Edges are not directed.



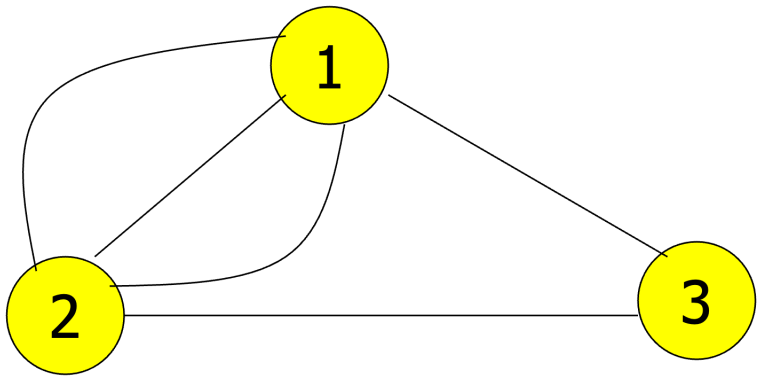
Graphs categorization

- Weighted Graphs:
  All the edges have been assigned a weight.



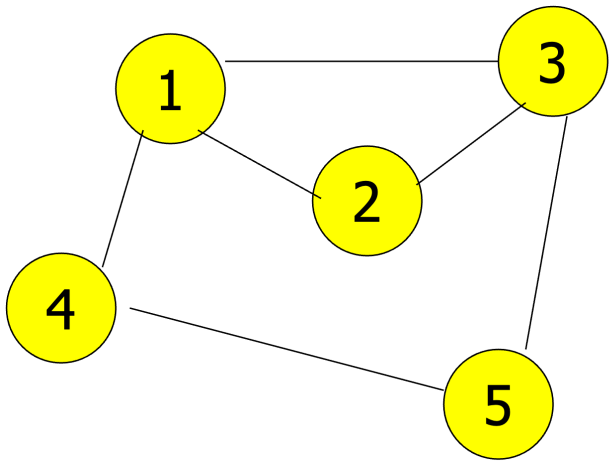
Graphs categorization

- Multigraphs:
  If the same pair of vertices have more than one edge.



Graph Terminology

- **G = {V, E}**
  where V is set of vertices and E is set of edges. Edges in a directed graph are ordered pairs.



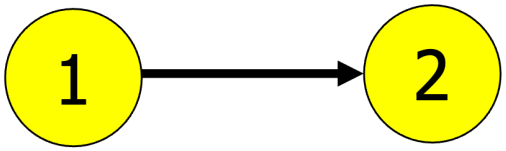
G=(V,E)

V={1,2,3,4,5}

E={(1,3),(1,2),(1.4),(4,5),(2,3),(3,5)}

Graph Terminology

- When the graph is a directed one the edge (i,j) is different from (j,i).



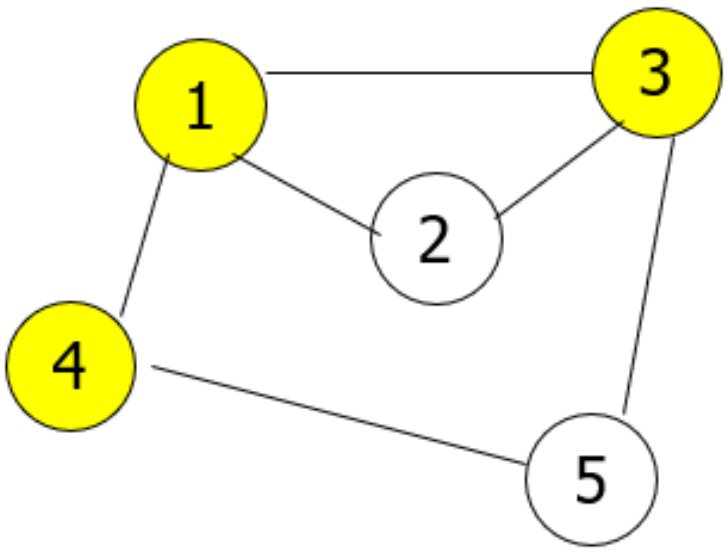
(i,j) Orientation is i to j.

Therefore here edge is (1,2).

Ex: Draw the **Directed graph**
if V={1,2,3,4,5} and
E={(1,2),(2,3),(3,4),(3,5),(5,4),(3,2)}

Graph Terminology

- **Adjacent vertices:** Vertices i and j are adjacent, if (i,j) is an edge of the graph.
- Eg:

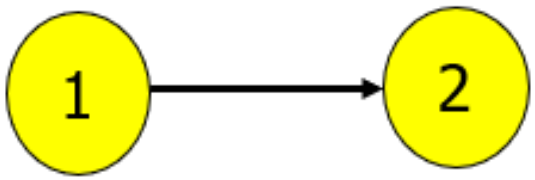


Vertices **2** and **5** are not adjacent since there is no edge between 2 and 5.

Vertices **3** and **4** are not adjacent since there is no edge between 3 and 4.

Others are adjacent vertices.

Graph Terminology



Vertex 1 is **adjacent to** vertex 2.

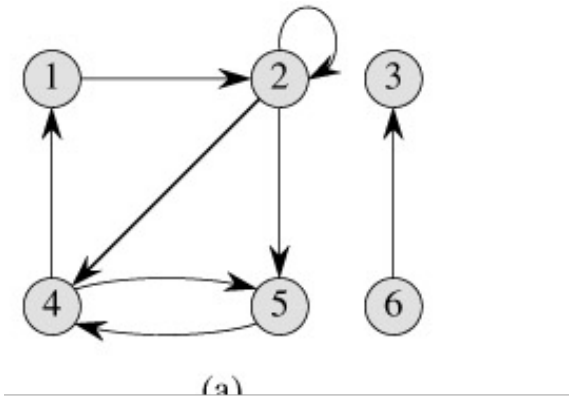
Vertex 2 is **adjacent from** vertex 1.

Edge is **incident** with vertex 1.

Edge is **incident** with vertex 2.

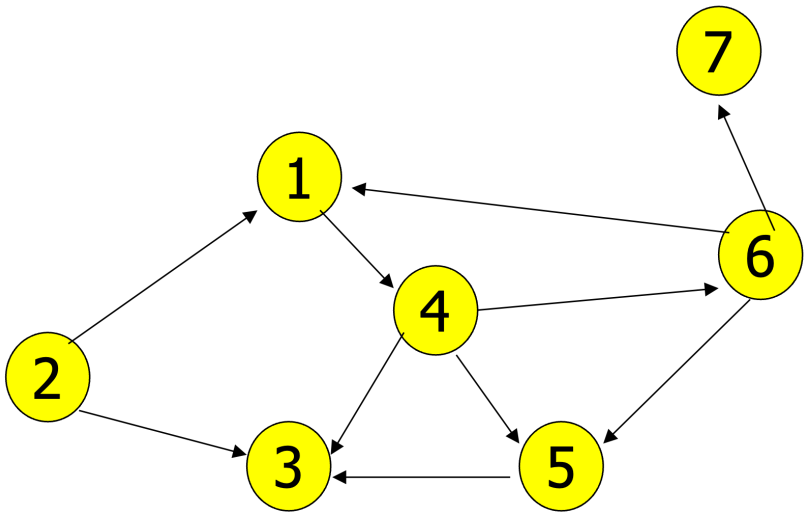
Graph Terminology

- **Loop or self – edges:**
  An edge (i,i) is called a self edge or a loop. Note that self-loops-edges from a vertex to itself-are possible.



Graph Terminology

- A **path** of **length** k from a vertex u to a vertex u' in a graph G=$\{$V,E$\}$ is a sequence $<v_0, v_1, v_2, ..., v_k>$ of vertices such that $u = v_0$, $u' = v_k$, and $(v_{i-1}, v_i) \in E \quad$ for $i = 1, 2, \ldots, k$.
  The length of the path is the number of edges in the path.

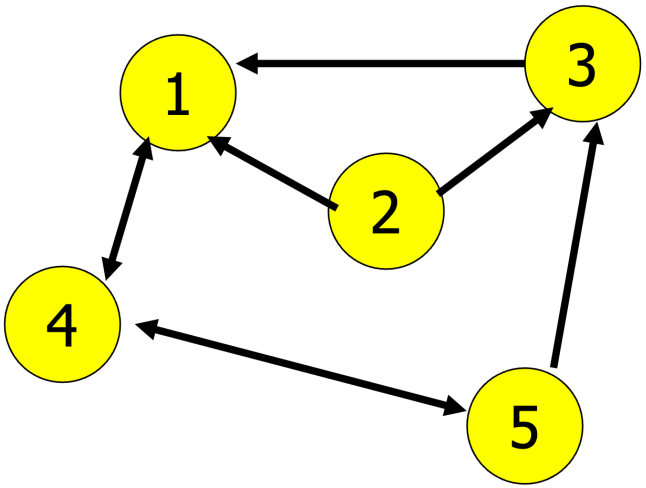


2,1,4,5 is a path from 2 to 5.

There is no path from 7 to 6.

Graph Terminology

- Simple Path: If all vertices in the path are distinct.

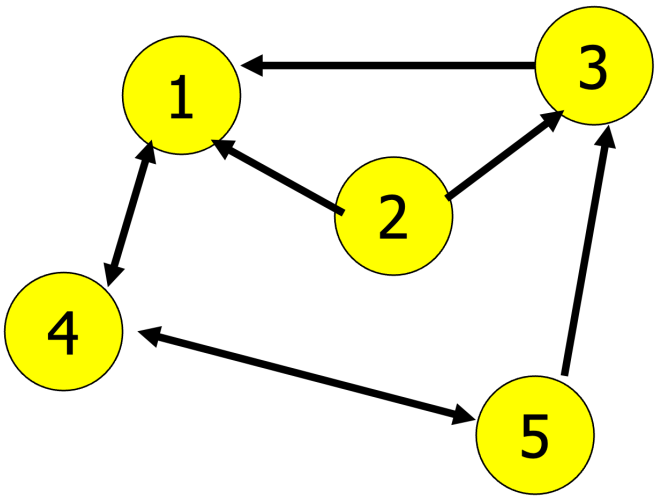


1,4,5,3 is a simple path.

But 1,4,5,4 is not a simple path.

Graph Terminology

- Length : sum of the number of the edges on the path.

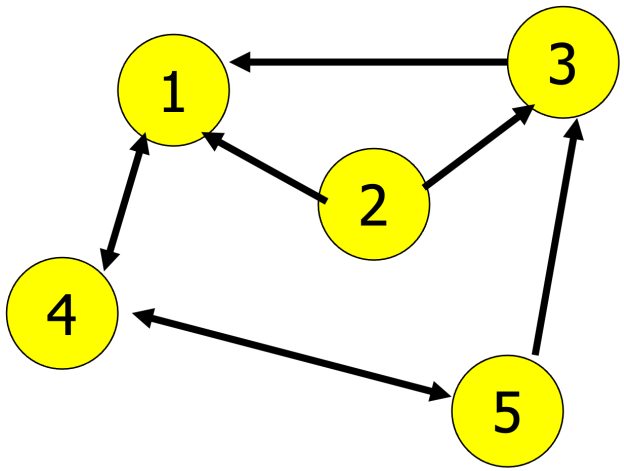


For the path 1,4,5,3

The length is 3.

Graph Terminology

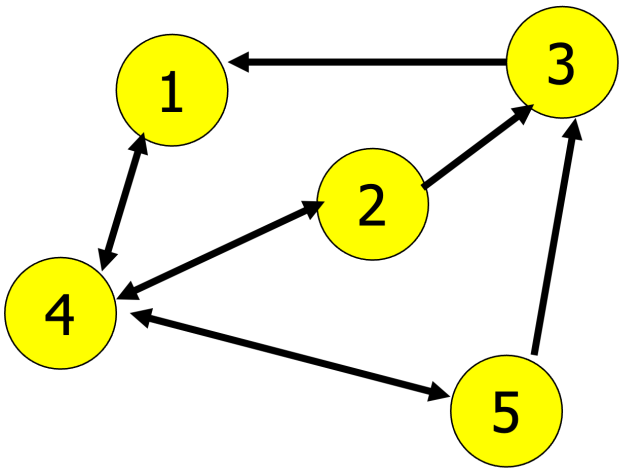
Vertex A is said to be reachable from B if there is a path from A to B.
A circuit is a path whose first and last vertices are the same.



The path 3,1,4,5,3 is a circuit.

Graph Terminology

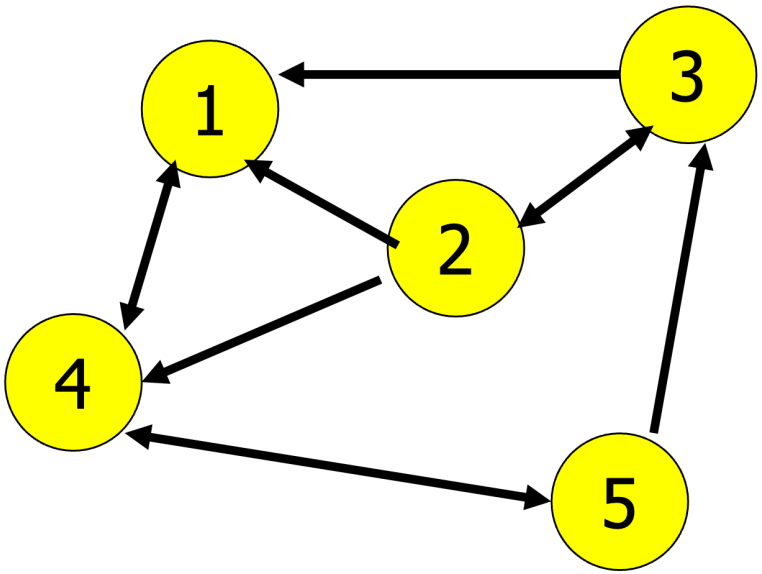
A simple circuit is a cycle if except for the first (and last) vertex, no other vertex appears more than once.



The path 3,1,4,5,3 is a cycle but 3,1,4,2,4,5,3 is not a cycle.

Graph Terminology

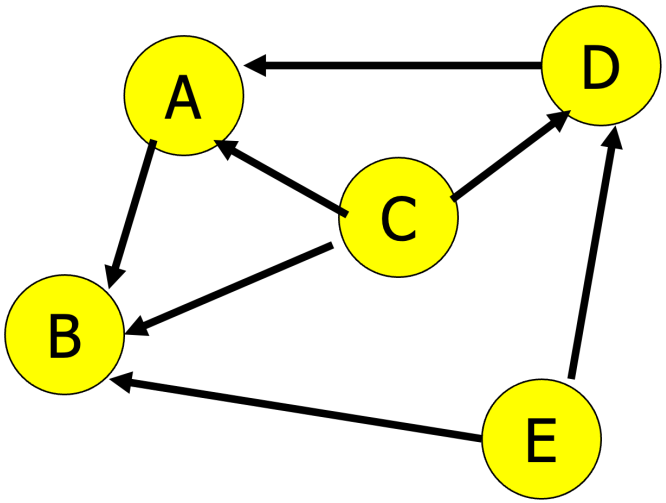
A Hamiltonian cycle of a graph G is a cycle that contains all the vertices of G.



3,2,1,4,5,3 is a Hamiltonian cycle.

Graph Terminology

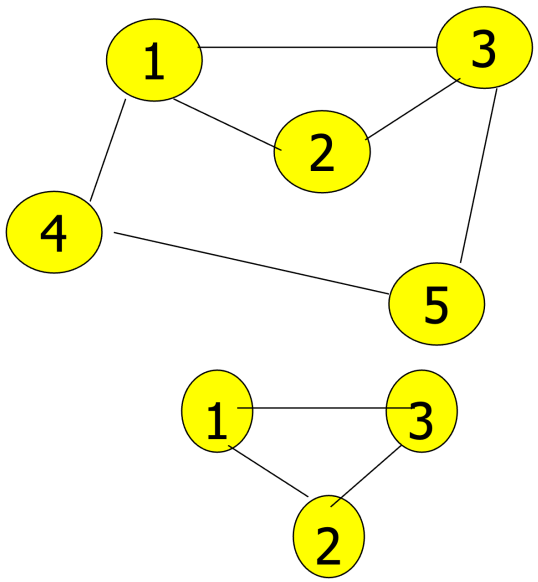
The degree d(v) of a vertex v is the number of edges incident to v.
In directed graphs, indegree is the number of incoming edges at the vertex and outdegree is the number of outgoing edges from the vertex.



The indegree of A is 2, its outdegree is 1.

Graph Terminology

A subgraph of a graph $G = (V, E)$ is a graph $H = (U, F)$ such that $U \subseteq V$ and $F \subseteq E$.



G=(V,E)

V={1,2,3,4,5}

E={(1,3),(1,2),(1.4),(4,5),(2,3),(3,5)}

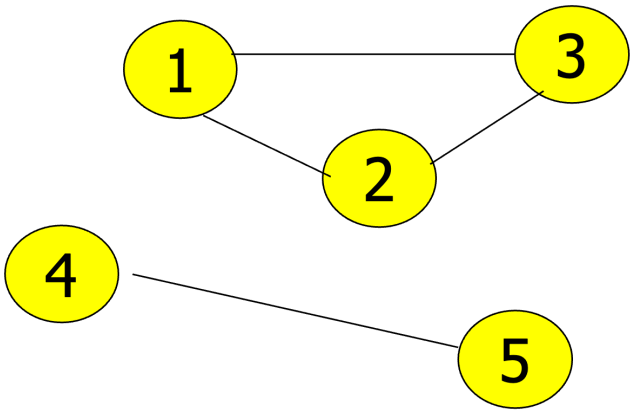
H=(U,F)

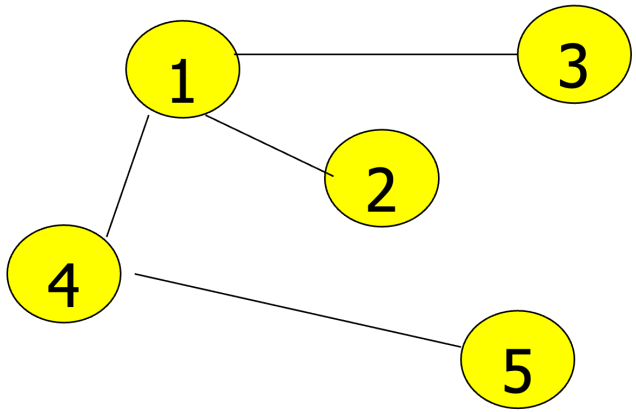
U={1,2,3}

F={1,3),(1,2),(2,3)}

Graph Terminology

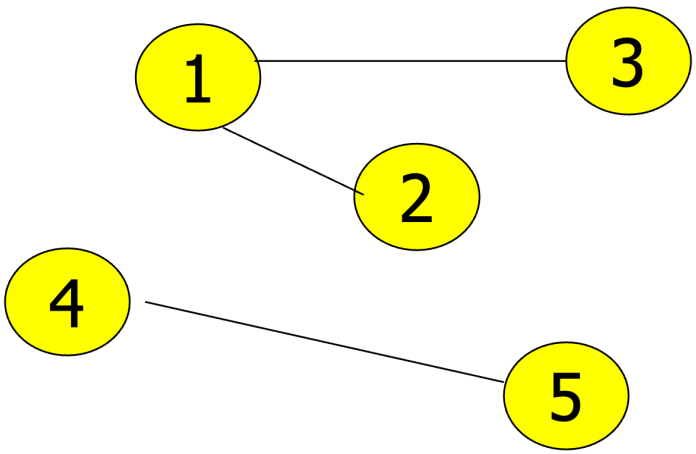
A graph is said to be connected if there is at least one path from every vertex to every other vertex in the graph.





Graph Terminology

- Tree : A connected undirected graph that contains no cycles is called a tree.
- Forest : A graph with no simple circuits, consisting of disjoint trees.
- G = (V, E) is undirected graph, then
  - is a tree
  - has no cycle, and connected
  - has $|V| - 1$ edges, and connected

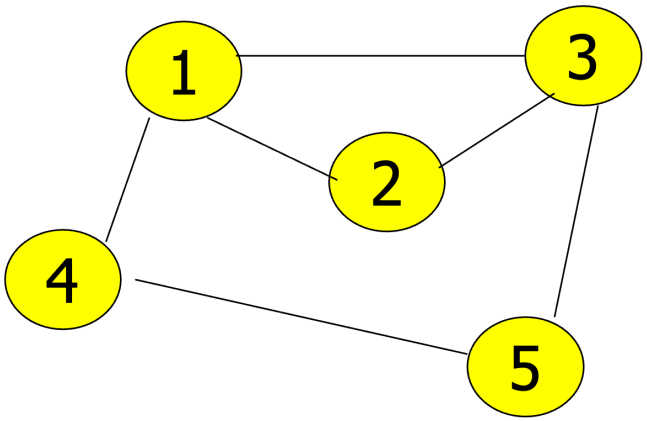


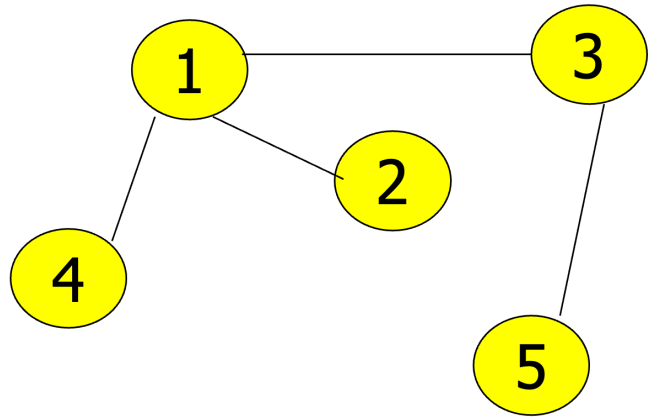
G(1,2,3,4,5)

is a forest.

Graph Terminology

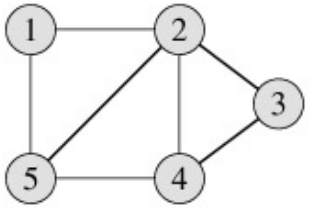
Spanning Tree : A spanning tree of a graph G is a subgraph of G that is a tree and contains all the vertices of G.



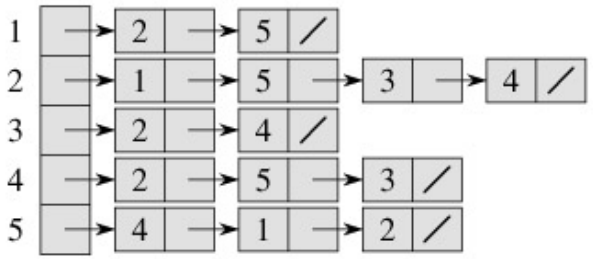


Representation of Graphs.

- There are two standard ways to represent a graph G = (V, E): as a collection of adjacency lists or as an adjacency matrix.
- Either way is applicable to both directed and undirected graphs. The adjacency-list representation is usually preferred, because it provides a compact way to represent **sparse** graphs-those for which $|E|$ is much less than $|V|^2$.
- An adjacency-matrix representation may be preferred, however, when the graph is **dense**-$|E|$ is close to $|V|^2$-or when we need to be able to tell quickly if there is an edge connecting two given vertices.



(a)



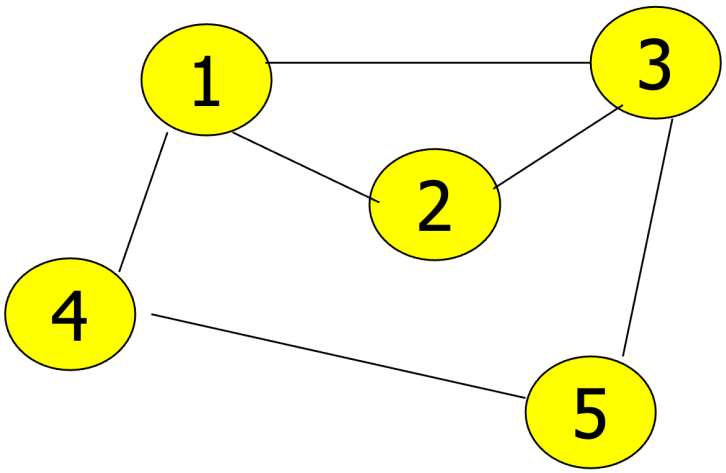
(b)

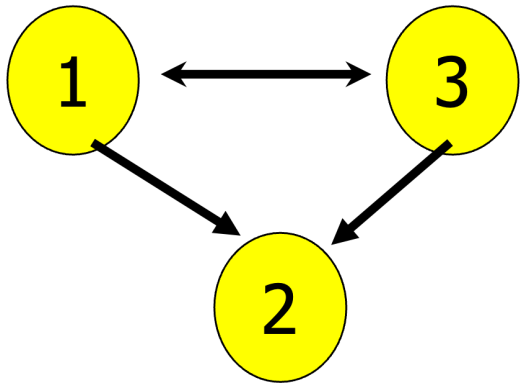
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

Representation of Graphs.

- Adjacency matrices.
  The adjacency matrix of an n-vertex graph G = (V, E) is an $n \times n$ matrix A. Each element of A is either 0 or 1.
  Ex: Find the adjacency matrix for the graphs.





Representation of Graphs.

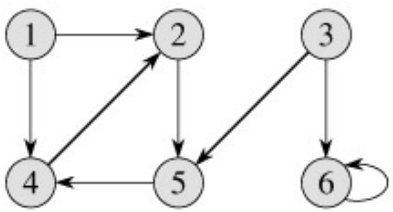
- The adjacency-matrix representation of a graph G consists of a $|V| \times |V|$ matrix $A = (a_{ij})$ such that
- Although the adjacency-list representation is asymptotically at least as efficient as the adjacency-matrix representation, the simplicity of an adjacency matrix may make it preferable when graphs are reasonably small.

$$|V| \times |V| \text{ matrix } A = (a_{ij})$$

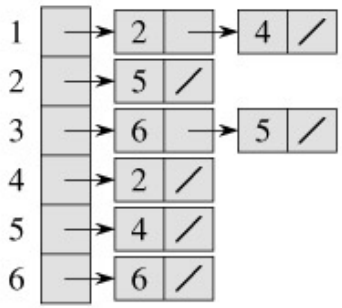
$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E , \\ 0 & \text{otherwise .} \end{cases}$$

Representation of Graphs.

- **Adjacency List : Adjacency list is an array of lists. Each individual list shows what vertices a given vertex is adjacent to.**



(a)

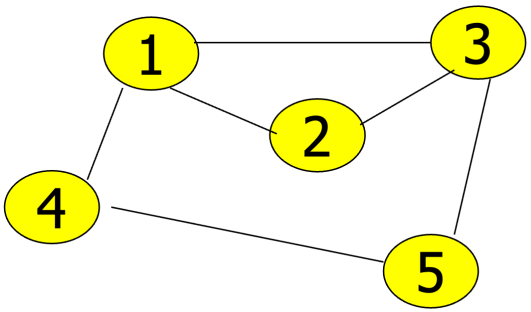


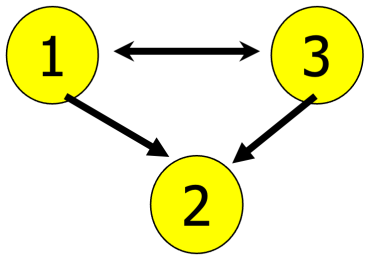
(b)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

Ex: Represent the graph using adjacency list.





Representation of Graphs.

- If G is a directed graph, the sum of the lengths of all the adjacency lists is $|E|$, since an edge of the form (u, v) is represented by having v appear in Adj[u].
- If G is an undirected graph, the sum of the lengths of all the adjacency lists is 2 $|E|$, since if (u, v) is an undirected edge, then u appears in v's adjacency list and vice versa.
- For both directed and undirected graphs, the adjacency-list representation has the desirable property that the amount of memory it requires is $\Theta(V + E)$.
- A potential disadvantage of the adjacency-list representation is that there is no quicker way to determine if a given edge (u, v) is present in the graph than to search for v in the adjacency list Adj[u].
- This disadvantage can be remedied by an adjacency-matrix representation of the graph, at the cost of using asymptotically more memory.

Searching Graphs.

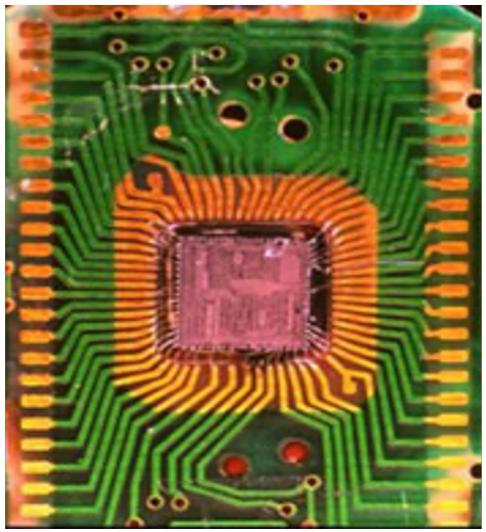
- Why do we do search?
  1. To find the paths
  2. To find the connectivity.

Breadth First Search (BFS)

Implemented with a queue.

Depth First Search (DFS).

Implemented with a stack.



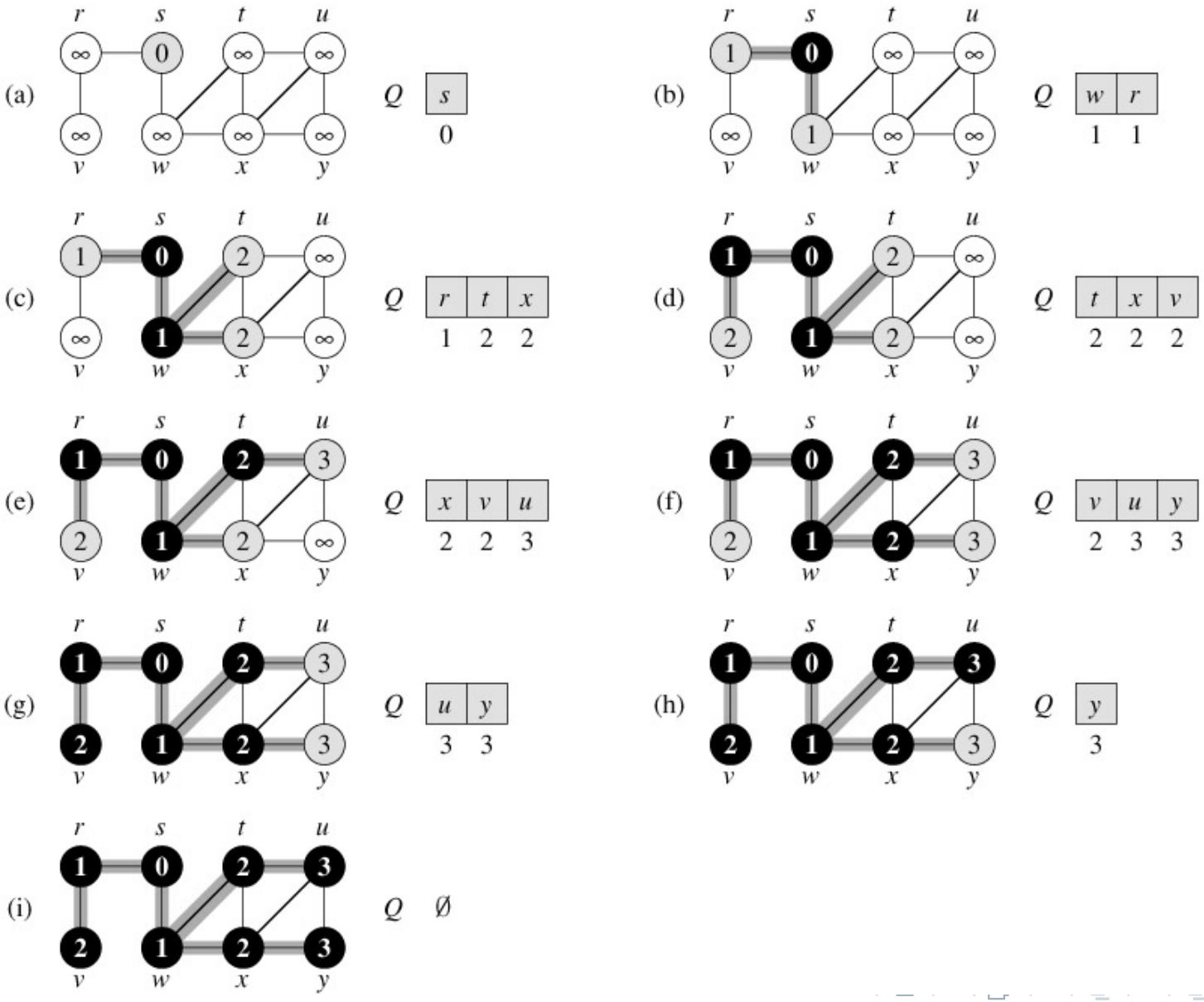
Breadth First Search (BFS)

- Given a graph G = (V, E) and a distinguished source vertex s, breadth-first search systematically explores the edges of G to "discover" every vertex that is reachable from s.
- It computes the distance (smallest number of edges) from s to each reachable vertex.
- It also produces a "breadth-first tree" with root s that contains all reachable vertices. For any vertex v reachable from s, the path in the breadth-first tree from s to v corresponds to a "shortest path" from s to v in G, that is, a path containing the smallest number of edges. The algorithm works on both directed and undirected graphs.
- To keep track of progress, breadth-first search colors each vertex white, gray, or black. All vertices start out white and may later become gray and then black.
- Breadth-first search constructs a breadth-first tree, initially containing only its root, which is the source vertex s. Whenever a white vertex v is discovered in the course of scanning the adjacency list of an already discovered vertex u, the vertex v and the edge (u, v) are added to the tree. We say that u is the predecessor or parent of v in the breadth-first tree .
- The distance from the source s to vertex u computed by the algorithm is stored in d[u]. The algorithm also uses a first-in, first-out queue Q to manage the set of gray vertices.

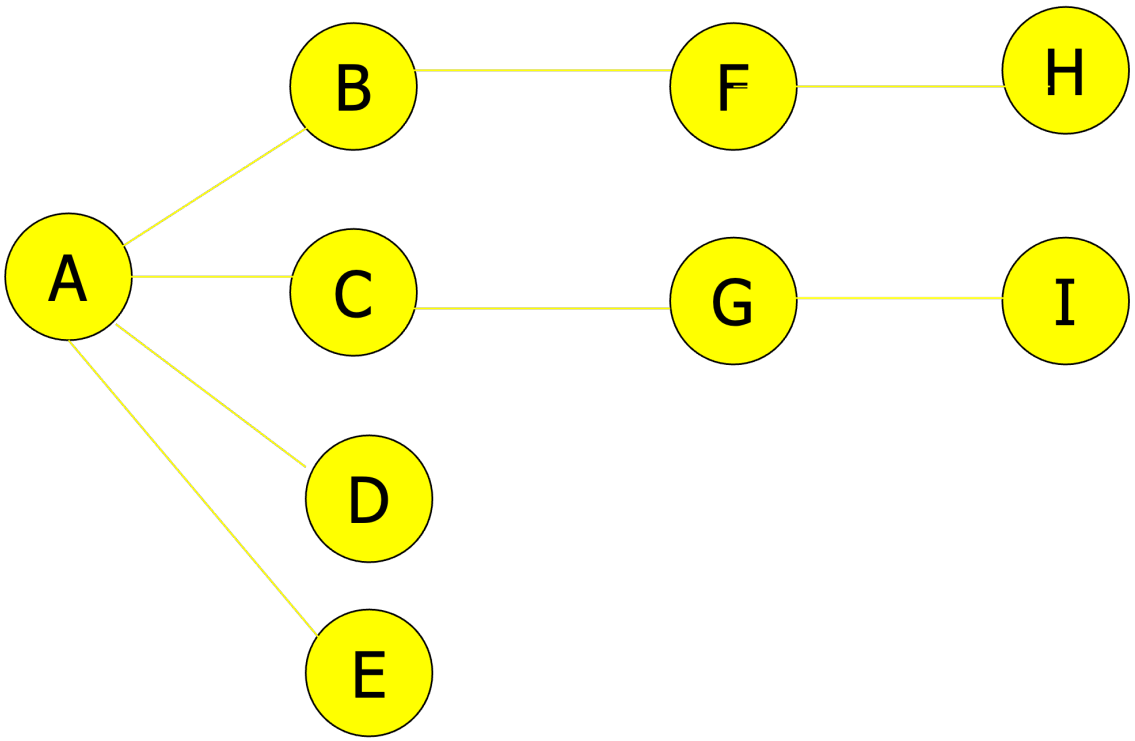
Breadth First Search (BFS)

```
BFS(G, s)
1   for each vertex u ∈ V[G] - {s}
2       do color[u] ← WHITE
3       d[u] ← ∞
4       π[u] ← NIL
5   color[s] ← GRAY
6   d[s] ← 0
7   π[s] ← NIL
8   Q ← Ø
9   ENQUEUE(Q, s)
10  while Q ≠ Ø
11      do u ← DEQUEUE(Q)
12          for each v ∈ Adj[u]
13              do if color[v] = WHITE
14                  then color[v] ← GRAY
15                       d[v] ← d[u] + 1
16                       π[v] ← u
17                       ENQUEUE(Q, v)

18  color[u] ← BLACK
```


(a)
r s t u
∞ 0 ∞ ∞
∞ ∞ ∞ ∞
v w x y
Q
s
0
(b)
r s t u
1 0 ∞ ∞
∞ 1 ∞ ∞
v w x y
Q
w r
1 1
(c)
r s t u
1 0 2 ∞
∞ 1 2 ∞
v w x y
Q
r t x
1 2 2
(d)
r s t u
1 0 2 ∞
2 1 2 ∞
v w x y
Q
t x v
2 2 2
(e)
r s t u
1 0 2 3
2 1 2 ∞
v w x y
Q
x v u
2 2 3
(f)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q
v u y
2 3 3
(g)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q
u y
3 3
(h)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q
y
3
(i)
r s t u
1 0 2 3
2 1 2 3
v w x y
Q
∅

EX: Do the BFS and draw the queue for the following graph.What is the order of nodes visited.



Depth First Search (DFS)

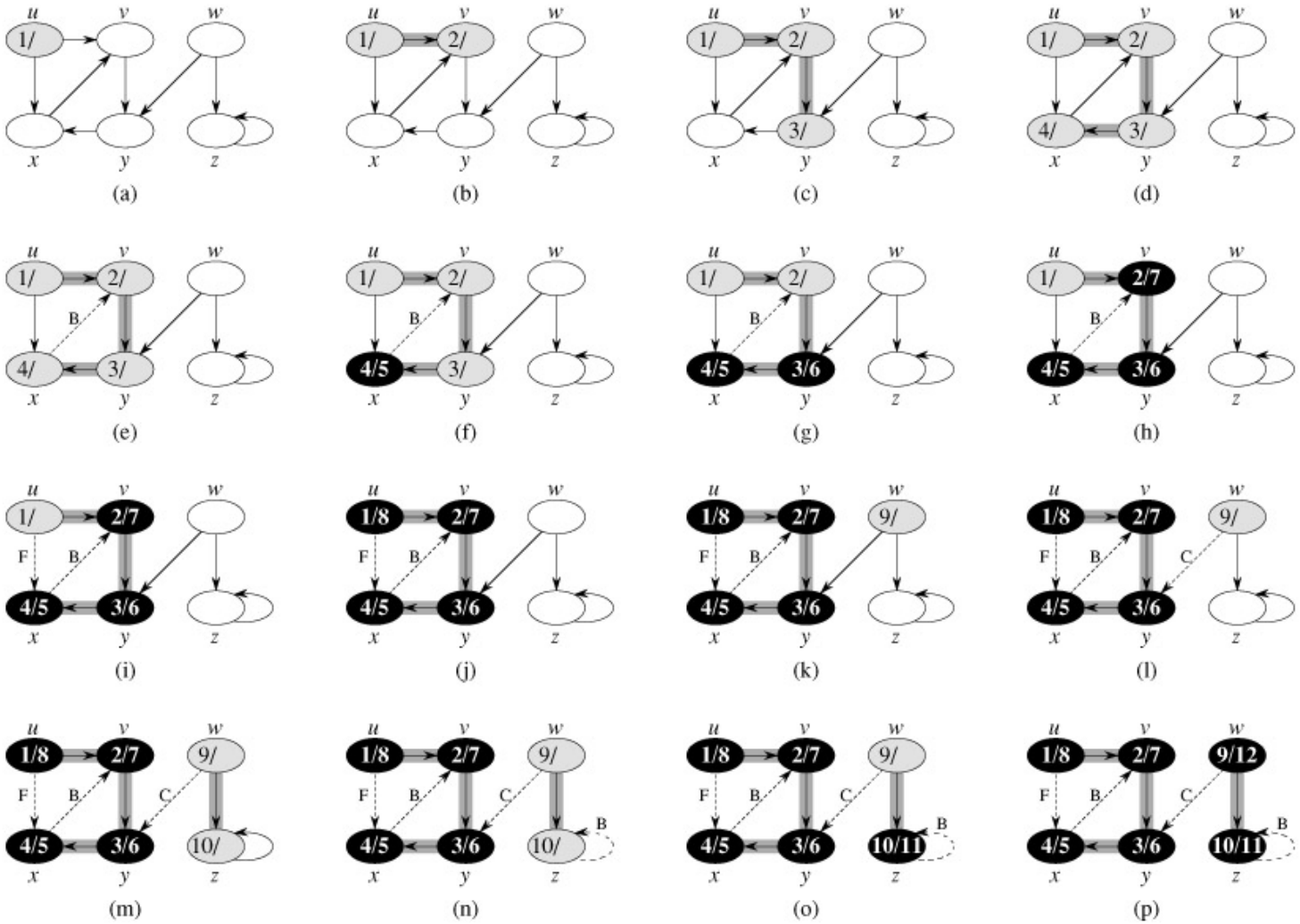
- It search the graph deeper whenever possible. Search is implemented by using a stack.
- In depth-first search, edges are explored out of the most recently discovered vertex v that still has unexplored edges leaving it.
- When all of v's edges have been explored, the search "backtracks" to explore edges leaving the vertex from which v was discovered.
- This process continues until we have discovered all the vertices that are reachable from the original source vertex. If any undiscovered vertices remain, then one of them is selected as a new source and the search is repeated from that source.
- This entire process is repeated until all vertices are discovered.

Depth First Search (DFS)

- Unlike breadth-first search, whose predecessor sub graph forms a tree, the predecessor sub graph produced by a depth-first search may be composed of several trees, because the search may be repeated from multiple sources.
- As in breadth-first search, vertices are colored during the search to indicate their state. Each vertex is initially white, is grayed when it is **discovered** in the search, and is blackened when it is finished, that is, when its adjacency list has been examined completely.
- Besides creating a depth-first forest, depth-first search also **timestamps** each vertex.
- Each vertex v has two timestamps: the first timestamp d[v] records when v is first discovered (and grayed), and the second timestamp f [v] records when the search finishes examining v's adjacency list (and blackens v). These timestamps are used in many graph algorithms and are generally helpful in reasoning about the behavior of depth-first search.

```
DFS(G)
1   for each vertex u ∈ V[G]
2       do color[u] ← WHITE
3          π[u] ← NIL
4   time ← 0
5   for each vertex u ∈ V[G]
6       do if color[u] = WHITE
7          then DFS-VISIT(u)

DFS-VISIT(u)
1   color[u] ← GRAY      ◁White vertex u has just been discovered.
2   time ← time +1
3   d[u] time
4   for each v ∈ Adj[u]     ◁Explore edge(u, v).
5       do if color[v] = WHITE
6          then π[v] ← u
7               DFS-VISIT(v)
8   color[u] BLACK      ◁Blacken u; it is finished.
9   f[u] ▷ time ← time +1
```


u v w
1/
x y z
(a)
1/ 2/
(b)
1/ 2/ 3/
(c)
1/ 2/ 4/ 3/
(d)
B
(e)
4/5 3/
(f)
4/5 3/6
(g)
2/7
(h)
F
(i)
1/8 2/7
(j)
9/
(k)
C
(l)
10/
(m)
(n)
10/11
(o)
9/12
(p)

Summary

- What is a graph.
- Graph terminology.
- Graph representation.
- Searching
- BFS
- DFS

Thank you !