**Graph**

**Definitions**

Directed graph

Undirected graph

Rooted tree: A rooted tree is a tree in which one vertex has been designated the root

Unrooted tree

Simple path: a simple path is a path in a graph which does not have repeating vertices

Connectivity: it asks for the minimum number of elements (nodes or edges) that need to be removed to disconnect the remaining nodes from each other.

V = set of vertices, E = set of edges between pairs of nodes.

n = |V|, the number of nodes or vertices

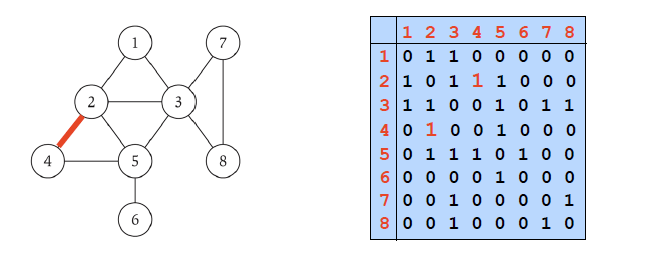
m = |E|, the number of edges between pairs of nodes

**Graph Representation**

**Adjacency Matrix**-> A n-by-n matrix with Auv = 1if there is an edge between u and v.(如果u和v直接相连的话才放1) (If the graph is an undirected graph, then each edge will be represented twice)

Space complexity O(n2), checking whether there is an edge between u and v takes O(1).

Identifying all edges takes O(n2) time.

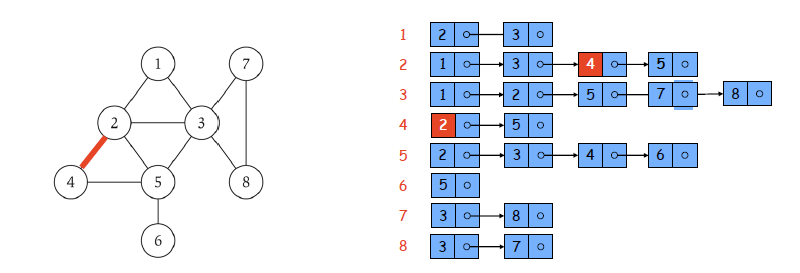


**Adjacency List**->Storing each node and all its neighbour nodes, for undirected graph, each edge is represented twice.

E.g. Node A-> node b node c node d (b, c, d are all neighbours of node a).

Space complexity = O(m + n), checking whether there is a edge between node u and v takes O(deg(u)) time, identifying all edges takes O(m + n) time.

Deg(u)-> number of neighbours of u



**Graph detailed definition**

**Simple Path**->Path in which all nodes are distinct.

**Connected Undirected Graph**->An undirected graph is connected if for every pair of nodes u and v, there is a path between u and v.

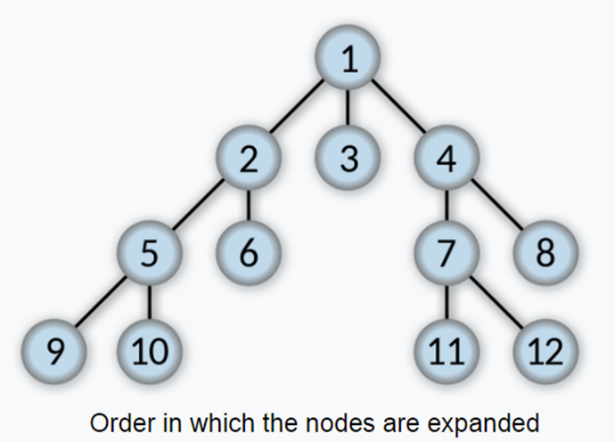
**Tree**

**Tree**->An undirected graph is a tree if it is connected and does not contain a cycle.

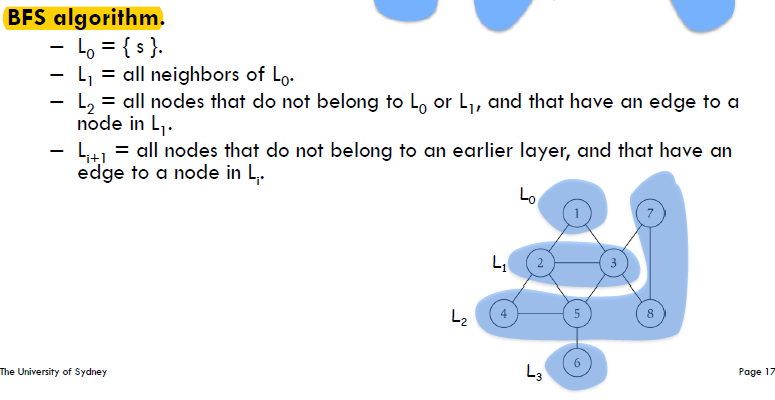
If the graph with n nodes is connected and does not contains any cycle then it has n-1 edges.

前提 Let G be an undirected graph with n nodes. 1. G is connected; 2. G does not contain a cycle; 3. G has n-1 edges, 知二推三

**BFS**

****

BFS的第零层只有initial node， 第一层有initial node的所有neighbours



**Breadth First Search (BFS) Theorem**->For each i, Li consists of all nodes at distance exactly i from starting vertex. There is a path from s to t if and only if t appears in some layer.

BFS explores outward from starting vertex s in all possible directions, adding nodes one **layer** at a time.

**BFS production**: BFS produces a tree T rooted at the starting vertex on the set of nodes in G reachable from S.

Property: Let T be a BFS tree of G = (V , E), and let (x , y) be an edge of G. Then the level of x and y differ by at most 1.

BFS->**Time Complexity** O(m + n) with adjacency list representation.

**Traverse all neighbours of a node u**:

* Adjacency list: O(number of neighbours) = O(|N(u)|)
* Adjacency matrix: O(n)

**Check if u and v are connected by an edge**:

* Adjacency list: O(number of neighbours) = O(|N(u)|) or O(|N(v)|)
* Adjacency matrix: O(1)

**Space Complexity**

* Adjacency List O(m + n)
* Adjacency Matrix O(n2).

BFS can be used to find the **shortest path** between node s and t in an unweighted graph G.

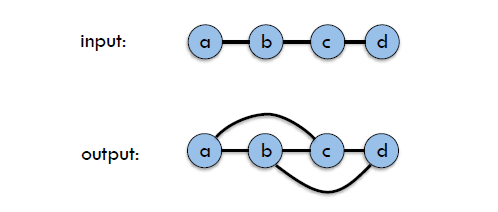
BFS can be used to decide if node t is reachable from node s. (find the connected component of S), upon termination, R is the connected component containing S.

BFS is done using **queue**.

**Transitive Closure**

Transitive Closure-> A transitive closure of a graph G is a graph with the same vertices as G, and with an edge between all pairs of nodes that are connected by a path in G.相当于是将graph G 相互reachable的node都连起来。

Transitive Closure of an undirected tree is a complete graph.



Transitive Closure is calculated using BFS, with running time complexity O(n\*(n + m)) = O(|V| \* (|V| + |E|))

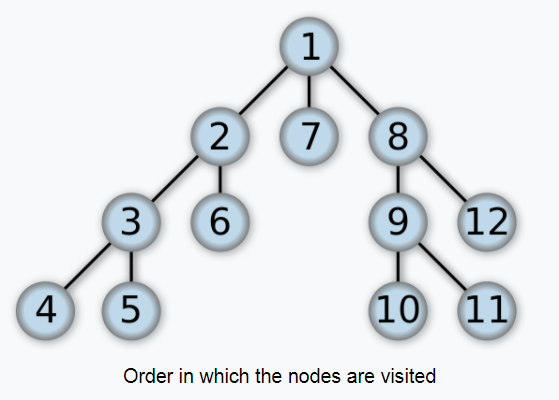
**Complete Graph:**

A complete graph is a simple undirected graph in which every pair of distinct vertices is connected by a unique edge.

**Simple Graph**

Simple graph is a undirected graph without loops or multiple edges.

**DFS**

****

**Idea**: DFS picks a starting vertex, follow outgoing edges that lead to new vertices, and backtrack whenever stuck.

**DFS Time Complexity**->O(n + m) with adjacency list representation.

Subset of edges in DFS that discover a new node (discover edge) form a forest (A collection of trees).

A graph is connected if and only if DFS results in a single tree.

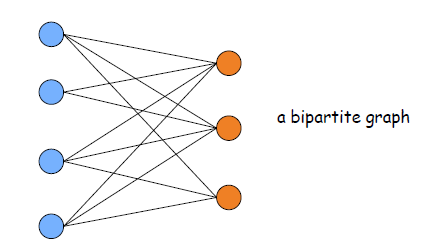
Each tree in the DFS result corresponds to a connected component.

DFS is done using **stack**.

DFS can be used to detect cycle.

**Bipartite Graph (Bipartiteness).**

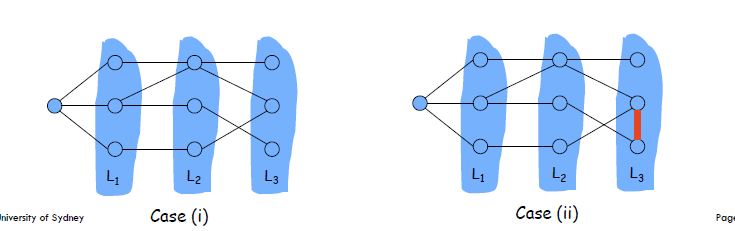
Bipartite Graph->An Undirected graph is bipartite if the nodes can be coloured red or blue such that every edge has one red end and one blue end.



Lemma: If a graph is bipartite, it cannot contain an **odd length cycle**, since it will not be two-colourable.

BFS can be applied to Bipartiteness.

Lemma: Let G be a connected graph, Li be the layers produced by BFS starting at node s. If no edge of G joins two nodes in the same layer, then G is bipartite, OR there exists an edge of G joins two nodes in the same layer, and then G contains an odd-length cycle, hence is not bipartite.

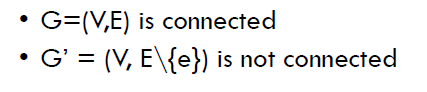


Corollary-> A graph G is bipartite if and only if it contains **no odd-length cycle**.

Testing bipartiteness takes O(m+n) time.

**Cut Edge**

Cut Edge->In a connected graph, an edge is called a cut edge, if its removal would disconnect the graph.



Using DFS to find the cut edges.

Running Time O(n + m).

**Traversing a Graph(BFS or DFS): O(n + m)**