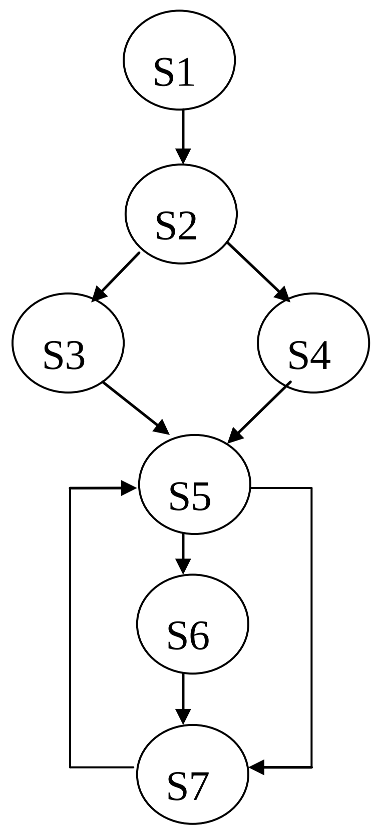
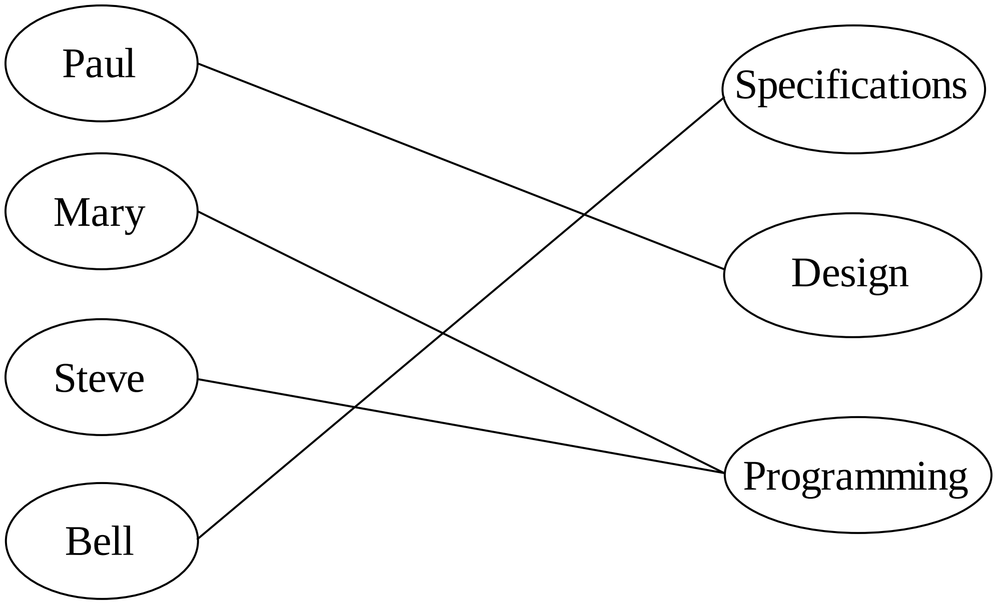
Graphs

- **Motivations:**
  - Networks
  - Social networks
  - Program testing
  - Job Assignment
- Examples:
  - Code graph:

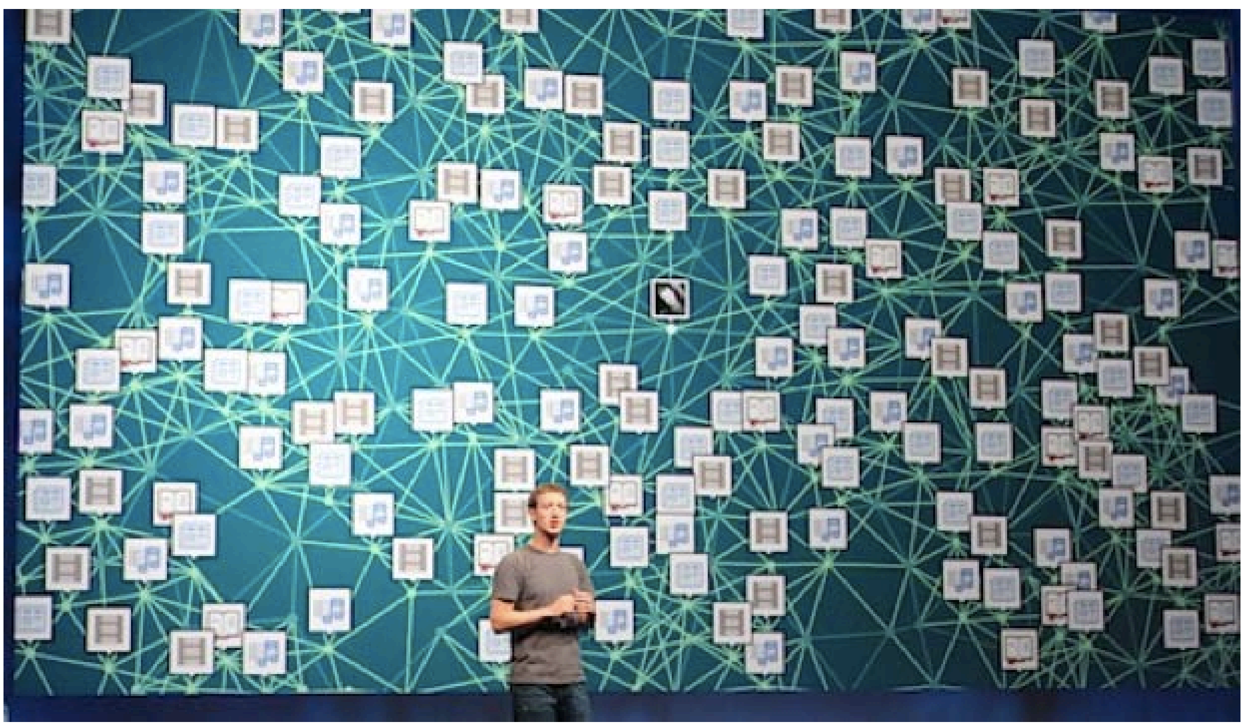
```
S1:   int x
S2:   If x > 0 then
 S3:          X = x + 2;
        Else
S4:           X =x -1;
        End if
S5:   While x > 1 do
S6:           Print x;
S7:           X = x -1;
        End while
```



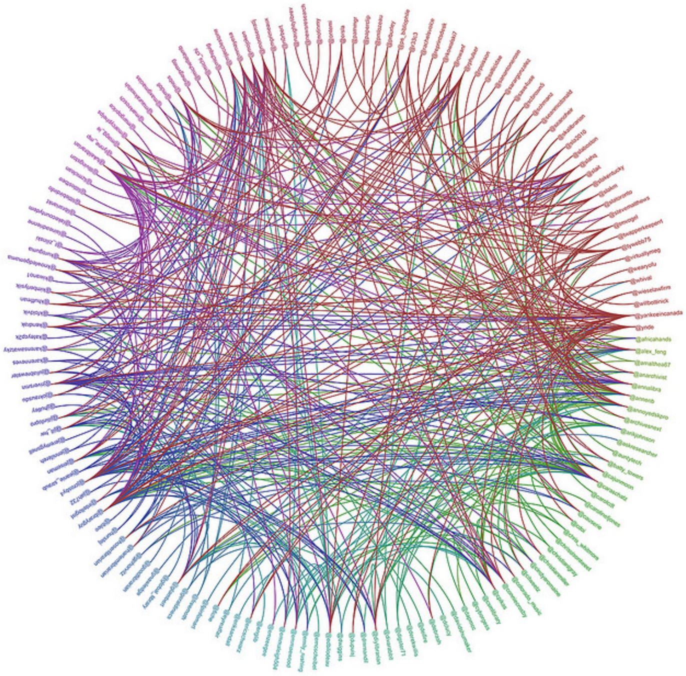
- Job Assignment:



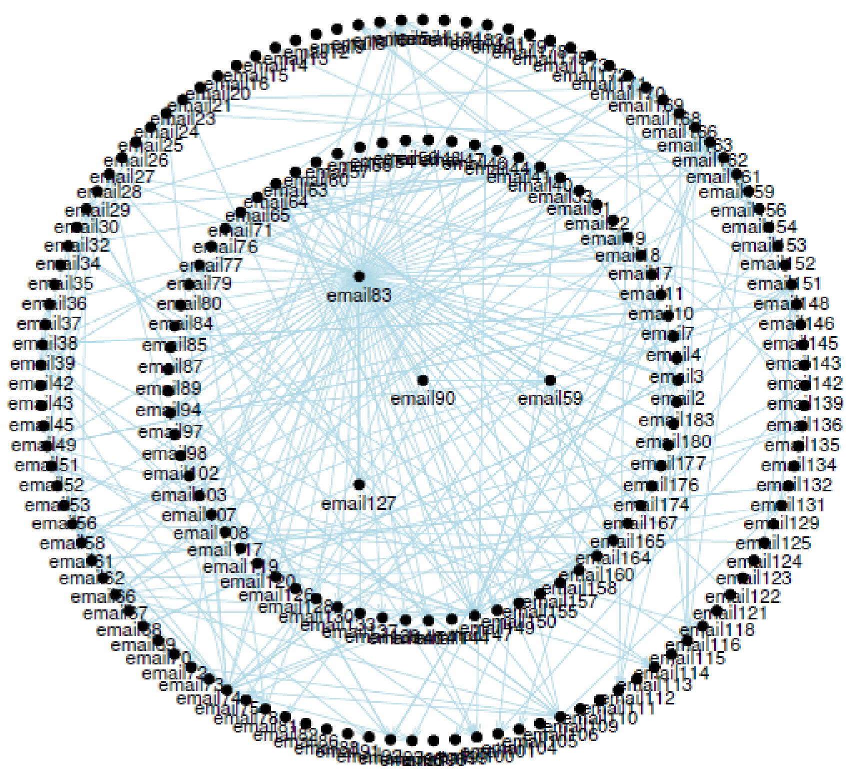
- Facebook graph: www.fastcompany.com



- Flick graph: www.flicker.com

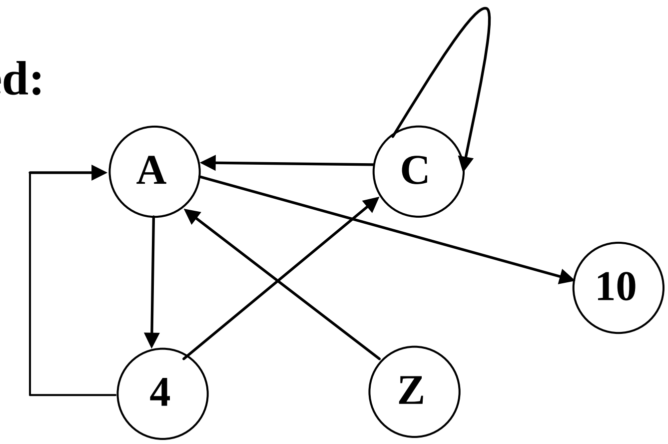


- Enron Emails graph: C.E. Priebe, J.M. Conroy, D.J. Marchette, and Y. Park, "Scan Statistics on Enron Graphs," SIAM International Conference on Data Mining, Workshop on Link Analysis.

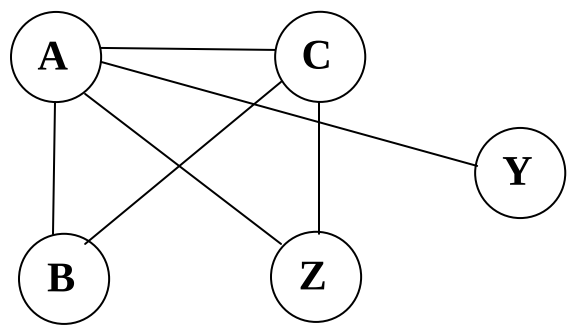


- **Definitions:**
    - A graph G is an ordered pair of sets (V,E) where V is a set of nodes and E is a set of edges or (arcs).
    - There are two types:
        - directed graphs (Digraphs)
        - undirected graphs

- **Examples:**
  - **Directed:**



    - Note the following:
      - the edges: <A,4> != <4,A>
      - <C,C> is called a self-loop.
  - **Undirected:**



      - Edge: (A,Y)

- **Adjacency nodes:**
  - **Definition:**
    - Given an edge <s,d>, d is adjacent to s. The set of all nodes adjacent to s is called the adjacency set of s.
    - Examples:

      The adjacency set of A is {B, C, Z, Y}

      The adjacency set of Z is {A, B}

      The adjacency set of Y is {A}

- **Paths:**
  - **Definition:**
    - is a sequence of edges $<x_1,x_2>$, $<x_2,x_3>$, ...,$<x_{n-1},x_n>$
  - **Simple path:**
    - All the nodes are distinct except possibly the first and the last.

  - **Length of a path:**
    - The number of edges in the path.
    - A simple edge is a path of length 1
    - A self-loop is a path of length 1

- **Reachability**:
  - **Definition:**
    - If there is a path from a node x to a node y, we say that y is reachable from x.

- **Cycles:**
  - **Definition:**
    - A cycle is a path such that the destination of the last edge is the source of the first edge.
    - A self-loop is a cycle of length 1.
    - Simple cycle:
      - It is a simple path which is a cycle.
    - Acyclic graphs:
      - It is a graph with no cycle in it. For example: trees.

- **Degree of a graph**
  - **Directed graphs:**
    - **In-degree:**
      - The in-degree of a vertex u is the number of edges entering it.
    - **Out-degree:**
      - The out-degree of a vertex u is the number of edges leaving it.
    - **Theorem:**
      - Let G=(V, E) be a graph with directed edges. Then

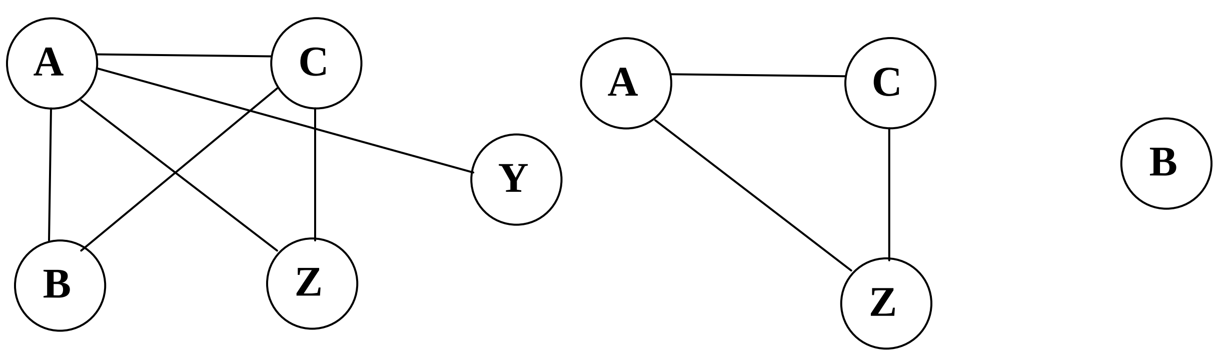
$$\sum_{\forall x \in V} indegree(x) = \sum_{\forall x \in V} outdegree(x) = |E|$$

  - **Undirected graphs:**
    - **Definition:**
      - The degree of a node is the number of its adjacent nodes.
    - **Theorem:**
      - Let G=(V,E), the sum of the degrees of each node equals 2|E| where |E| is the number of edges:

$$\sum_{\forall x \in V} degree(x) = 2|E|$$

- **Connected graphs**
  - **Undirected:**
    - **Definition:**
      - An undirected graph is connected if for every two vertices i and j there exists at least one path from i to j.
  - **Directed:**
    - **Connected:**
      - A directed graph is connected if the undirected graph obtained by ignoring the edge directions is connected.
    - **Strongly connected:**
      - A directed graph is strongly connected if for every two vertices I and j there exists a path from i to j and from j to i.

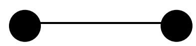
  - **Subgraph**:
    - **Definition:**
      - Given a graph H = (V_0, E_0) is a subgraph of G = (V, E) where $V_0 \subseteq V$ and $E_0 \subseteq E$.
      - Example:



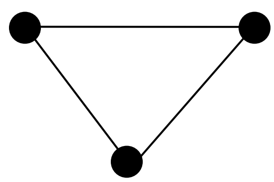
      - H is a proper subgraph of G if H≠G.

- **Special Graphs:**
  - **Complete graph**:
    - **Definition:**
      - A complete K_n is an undirected graph has n vertices and has an edge connecting every pair of distinct vertices.
    - **Example:**
      - K_1:

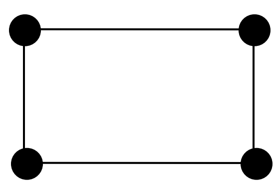
      - K_2:



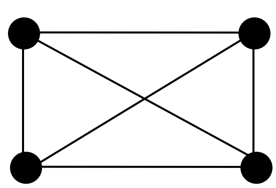
      - K_3:



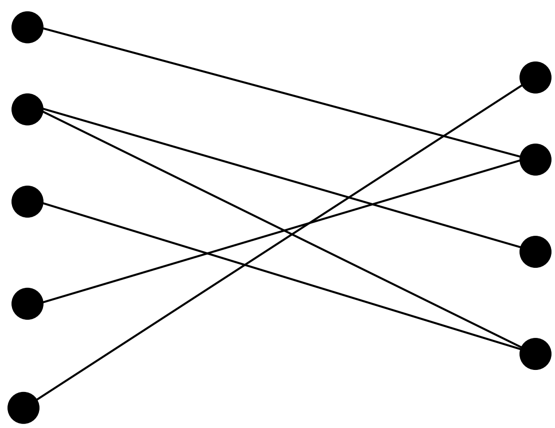
      - K_4:



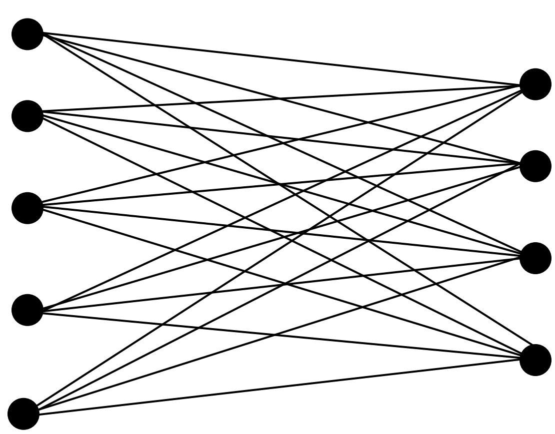
      - K_4:



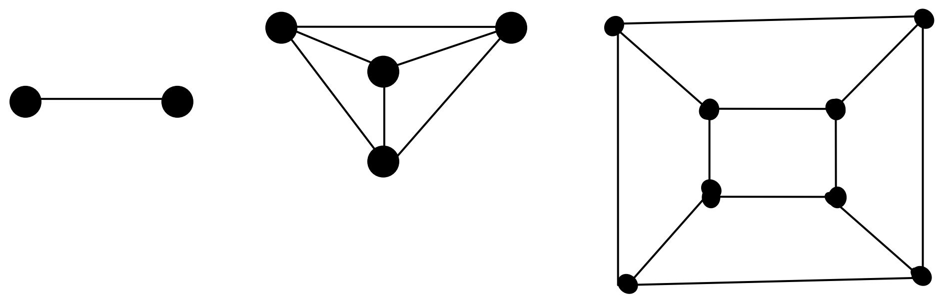
- **Bipartite (or bigraph) Graph:**
  - **Definition:**
    - Bipartite is a simple graph in which the vertices can be partitioned into disjoint sets V1 and V2:
    - Edges connect vertices of sets V1 and V2
    - No edges connect vertices of: V1 with other vertices of V1 or V2 with other vertices of V2
  - **Example:**



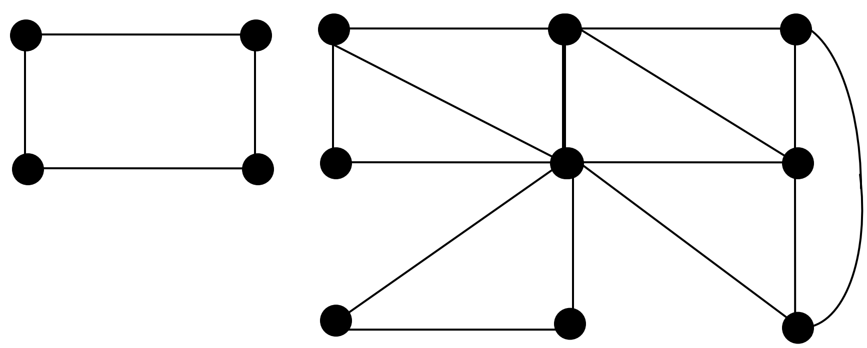
  - **A complete bipartite undirected graph**
    - **Definition:**
      - $K_{\mathbf{m,n}}=(V_1 U V_2, E)$ is a bipartite graph where each vertex in V_1 is connected to every vertex in V_2.
  - **Example:**



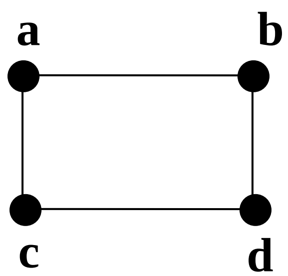
- **Regular Graph:**
  - **Definition:**
    - A regular undirected graph of degree k is a graph in which each vertex has degree k.

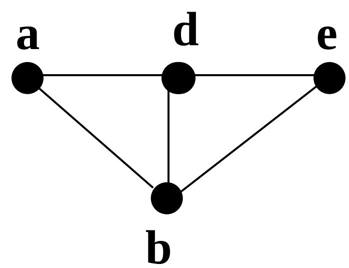


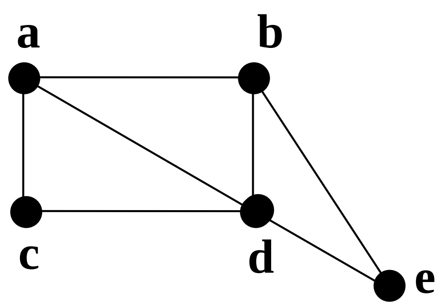
- **Planar Graph:**
  - **Definition:**
    - If the graph can be drawn on a plane without edges crossing.
  - **Examples:**

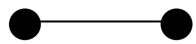
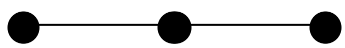


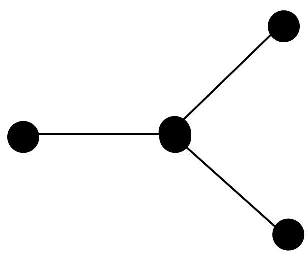
- **Union of two simple graphs**
  - **Definition:**
    - The union of G1= (V1, E1) and G2= (V2, E2) is the simple graph with vertex set $V1 \cup V2$ and edge set $E1 \cup E2$. The union of G1 and G2 is denoted by $G1 \cup G2$
  - **Example**:



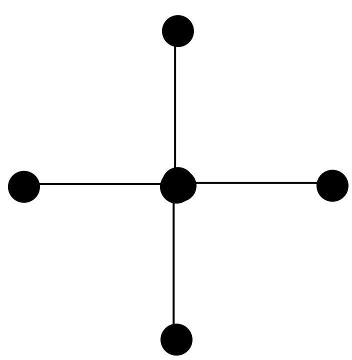




- **Star graph:**
  - **Definition:**
    - A star graph S_n is a graph of n vertices with one node having vertex degree n-1 and the other n-1 vertices having vertex degree 1.
  - **Example:**
    - S_1: 
    - S_2: 
    - S_3: 
    - S_4:



    - S_5:



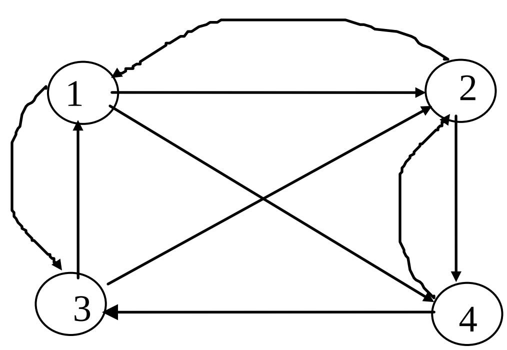
- **Graph Representation**
  - **There are two types of representations**:
    - Adjacency matrix
    - Adjacency list

  - **Adjacency matrix**
    - No information is associated with edges
    - Each edge is associated with a cost or info.: weighted adjacency matrix

  - **Unweighted adjacency matrix**:
    - **Definition:**
      - Let M be the adjacency matrix of a graph G. M is defined as follows:

$$\begin{cases} M \in \Pi_{|V||V|} \quad \textit{where } \Pi_{|V||V|} \textit{ is the set of square matrices of diameter } |V| \\ M[i,j] = \begin{cases} 1 \textit{ or TRUE} & \textit{If there is an edge between vertex } i \textit{ and vertex } j. \\ 0 \textit{ or False} & \textit{Otherwise} \end{cases} \end{cases}$$

    - **Example:**



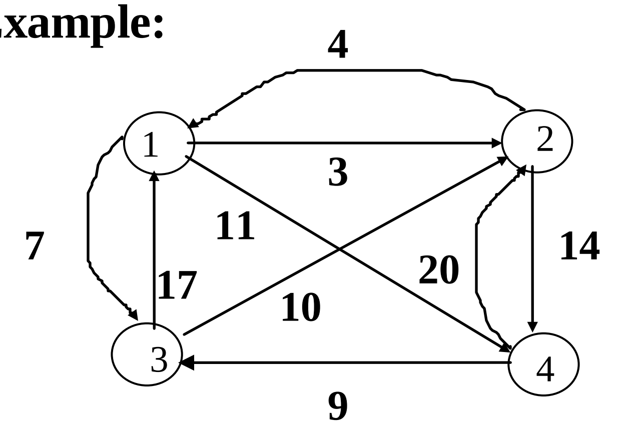
- The adjacency matrix M is:

$$M = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

- **Weighted adjacency matrix**:
  - **Definition:**
    - Let M be the adjacency matrix of a graph G. M is defined as follows:

$$\begin{cases} \boldsymbol{M \in \Pi_{|V||V|} \ \ where \ \Pi_{|V||V|} \ is \ the \ set \ of \ square} \\ \qquad\qquad\qquad\qquad \boldsymbol{matrices \ of \ diameter \ |V|} \\ \boldsymbol{M[i,j]} = \begin{cases} \boldsymbol{c} & \boldsymbol{If \ there \ is \ an \ edge \ between \ vertex \ i \ and \ vertex \ j \ whose} \cos \boldsymbol{t \ is \ c.} \\ 0 & \boldsymbol{If \ i = j} \\ \infty & \boldsymbol{If \ i \neq j \ and \ there \ is \ no \ edge \ between \ i \ and \ j} \end{cases} \end{cases}$$

  - **Example:**



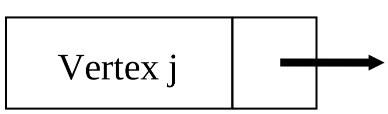
- The adjacency matrix M is:

$$M = \begin{bmatrix} 0 & 3 & 7 & 11 \\ 4 & 0 & \infty & 14 \\ 17 & 10 & 0 & \infty \\ \infty & 20 & 9 & 0 \end{bmatrix}$$

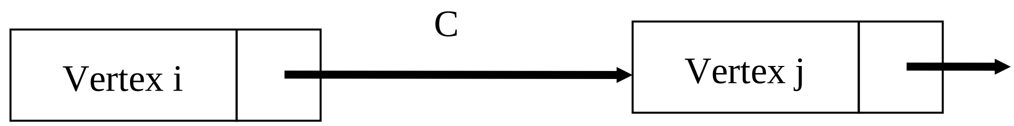
- **Notes:**
  - If the graph is undirected, both unweighted adjacency matrix and weighted adjacency matrix are symmetric matrices.

- **Drawback of adjacency matrix representation:**
  - Algorithms using adjacency matrix representation require at least $O(n^2)$ where $n=|V|$ and V is the set of vertices of the input graph.

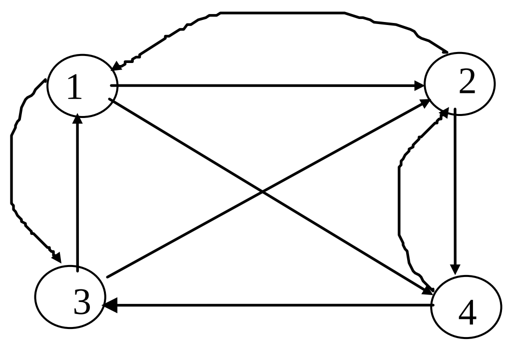
- **Adjacency list**
  - There is a list for each vertex in the graph: the nodes in this list represent the vertices that are adjacent from vertex I.
  - Each node of the list associated with vertex i consists of the following:
    - No information is associated with G: if there is an edge between (i,j):



- Weighted graph: if there is an edge between (i,j) whose cost is c:

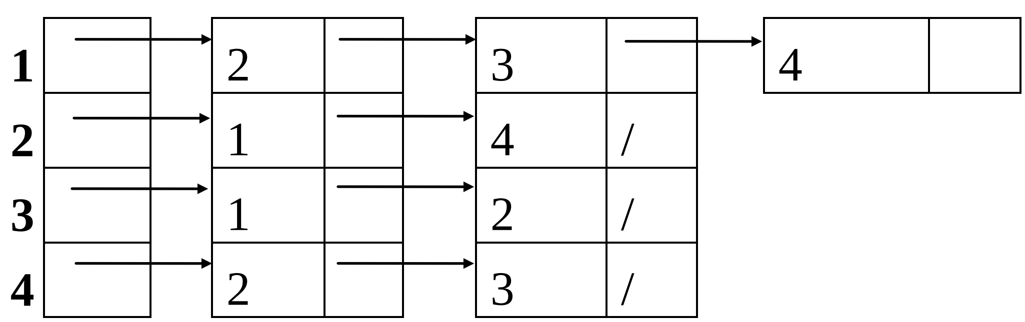


- Example:



  - The adjacency list is:

Head Nodes



- **Graph Traversals**
  - There are two strategies:
    - **Depth First Search (DFS)**
    - **Breadth First Search (BFS)**

  - **Depth First Search (DFS)**
    - Procedure:

```
DFS (G,v)
Begin
    visited(v) = TRUE;
    For every node x neighbor of v do
        If visited x = FALSE
        then DFS(G,x)
        endif
    endfor
End;
```

    - **Analysis:**
      - For G=(V,E) where n=|*V*| and e=|*E*|, the time complexity is:
        - Adjacency matrix:
          - Since the FOR loop takes O(n) for each vertex, the time complexity is: <u>**$O(n^2)$**</u>

          - Adjacency list:

- The FOR loop takes the following:

$$\sum_{i=1}^{n} d_i = O(e) \quad \text{where } d_i = \text{degree}(v_i)$$

- The setup of the visited array requires:

  O(n)

- Therefore, the time complexity is:

  O(max(n,e))

- **Breadth First Search (BFS)**
  - Procedure:

```
BFS (v)
queue Q;
Begin
        visited(v) = TRUE;
        Make_empty(Q);  /* Make the queue empty */
        Add_queue(Q,v);
        While (!Empty_queue(Q)) do
        Begin
                Delete_queue(Q,x);
                For all vertices w adjacent to x do
                        If (!visited[w])
                        then Begin
                                        Add_queue(Q,w);
                                        visited[w]=TRUE;
                                end;
                endfor
        End;
End;
```

  - **Analysis:**
    - For G=(V,E) where n=$|\boldsymbol{V}|$ and e=$|\boldsymbol{E}|$, the time complexity is:

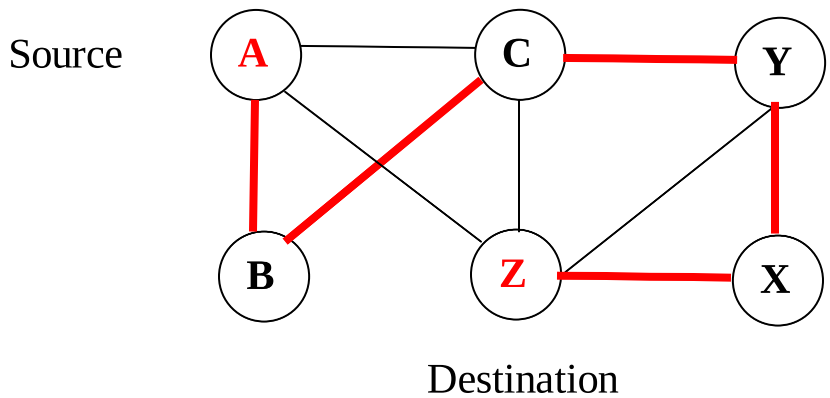
- Adjacency matrix:
  - Since the while loop takes O(n) for each vertex, the time complexity is: <u>**$O(n^2)$**</u>

- Adjacency list:
  - The while loop takes the following:

$$\sum_{i=1}^{n} d_i = O(e) \quad \text{where } d_i = \text{degree}(v_i)$$

  - The setup of the visited array requires: O(n)
  - Therefore, the time complexity is: <u>**O(max(n,e))**</u>

- **Applications:**
  - Find a path from Source → Destination
    - Use either DFS or BSD
    - Need to store the edges traversed
      - Use depth
      - Use breath
    - Example:



- Start at node A: push A in the stack

					Z
				X	X
			Y	Y	Y
		C	C	C	C
	B	B	B	B	B
A	A	A	A	A	A
DFS on A	DFS on B	DFS on C	DFS on Y	DFS on X	DFS on Z

- Is an undirected graph connected?
  - Think about a DFS based algorithm?

- Check whether an undirected graph is a regular graph. Print the degree of the graph.

- To find out if a graph contains a cycle.
  - How?

```
boolean DFS(v){
   visited[v] = 1;
    for( each vertex w adjacent to v ){
         if (visited[w] == 0){
              parent[w] = v;
              DFS(w);
          }
        else if(visited[w] == 1 and parent[w] !=  v)
              return true;       // cycle detected
    }
   return false;          // no cycle detected in this component
}
```