DATA STRUCTURES & ALGORITHMS

COMP H3025

Lecture 8: Graphs

Lecturer: Stephen Sheridan

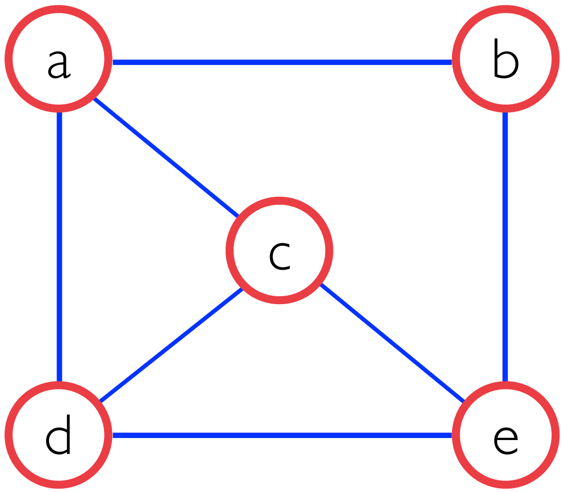
WHAT IS A GRAPH

- A graph G = (V,E) is composed of :

  V : set of vertices

  E : set of edges connecting the vertices in V

- An edge e = (u,v) is a pair of vertices



diagram

V = {a,b,c,d,e}

E = {(a,b), (a,c),(a,d),
(b,e),(c,d), (c,e), (d,e)}

set notation

WHAT IS A GRAPH

- We will often talk about travelling an edge

- Traveling an edge means we are changing our node of interest by following an edge connected to it.

- If our graph has nodes A and B that are connected by an edge , to represent the fact that our interest has changed from A to B, we can talk about :
    - moving from A to B
    - travelling from A to B
    - traversing from A to B

WHAT IS A GRAPH

- We will often talk about travelling an edge. Traveling an edge means we are changing our node of interest by following an edge connected to it.

- If our graph has nodes A and B that are connected by an edge , to represent the fact that our interest has changed from A to B, we can talk about :
  - moving from A to B
  - travelling from A to B
  - traversing from A to B

WHAT IS A GRAPH

- Usually we will just write the two node labels as shorthand for the edge that connects them.

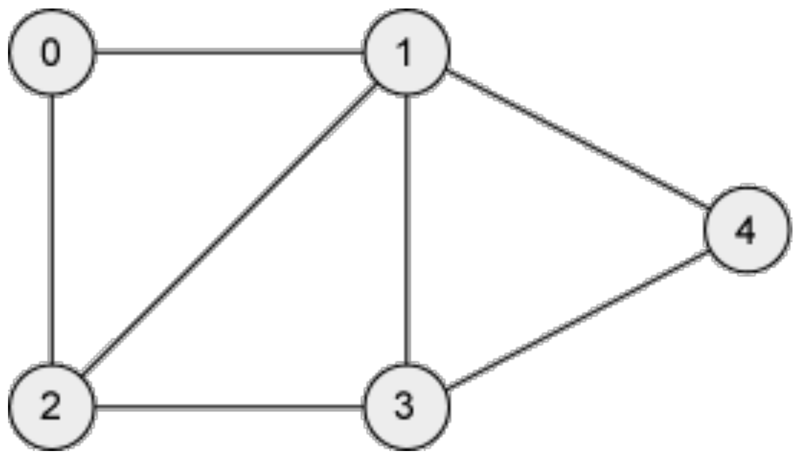
- AB will therefore represent the edge between nodes A and B.

- We will say that B is adjacent to A.

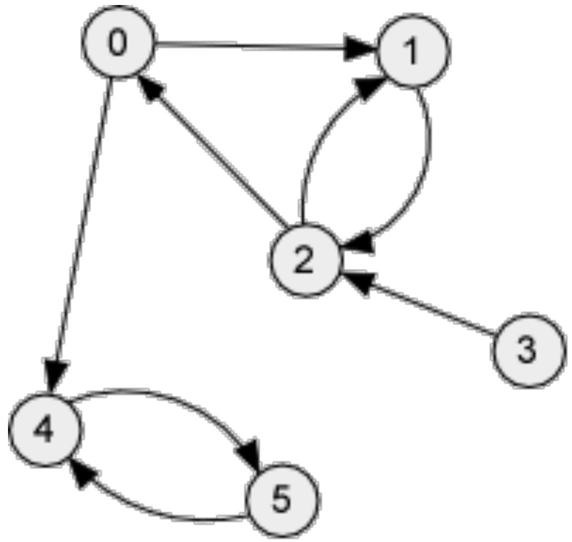
DIRECTED V'S UNDIRECTED

- An undirected graph or graph, has edges that can be traversed in either direction.

- In mathematics, and more specifically in graph theory, a directed graph (or digraph) is a graph, or set of nodes connected by edges, where the edges have a direction associated with them.

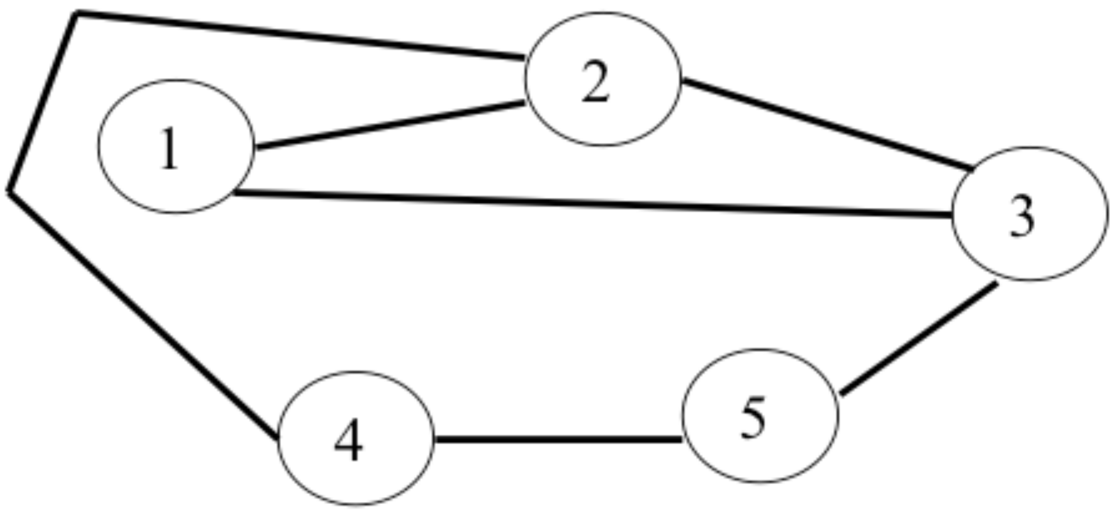


undirected



directed

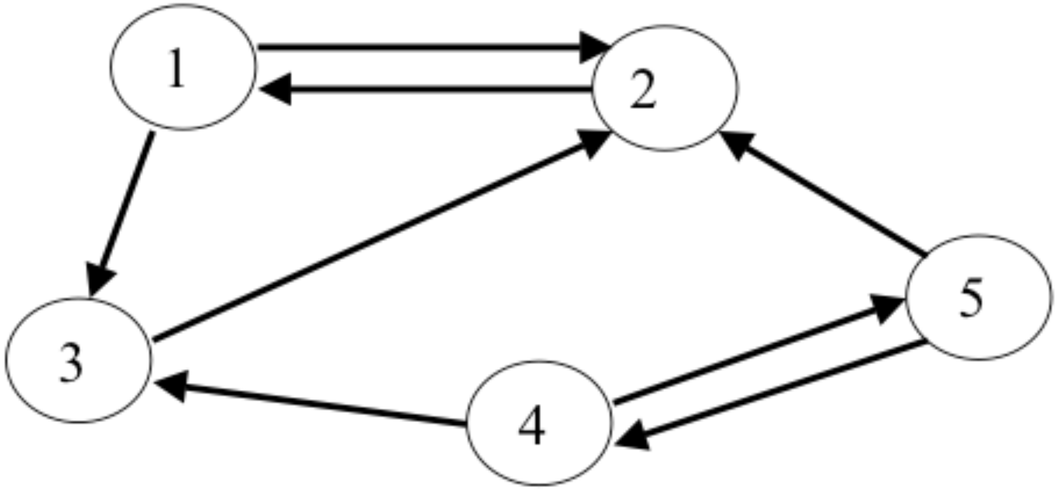
DIRECTED V'S UNDIRECTED



The graph:

G = ({1, 2, 3, 4, 5}, { {1, 2}, {1, 3}, {2, 3}, {2, 4}, {3, 5}, {4, 5} })

DIRECTED V'S UNDIRECTED



The digraph:

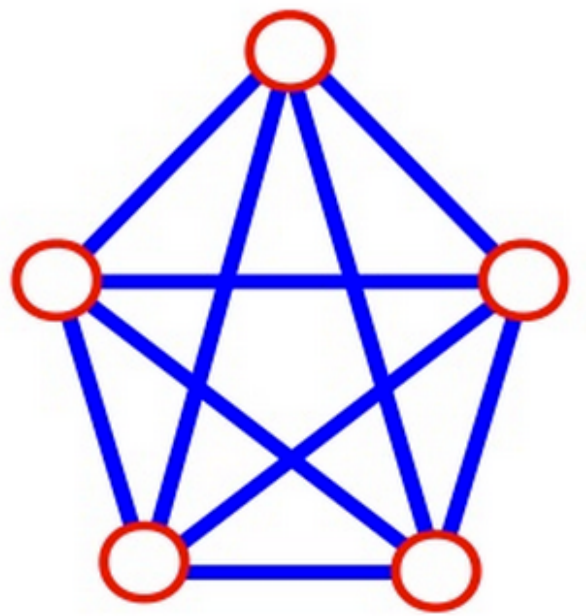
G = ({1, 2, 3, 4, 5}, { (1, 2), (1, 3), (2, 1), (3, 2), (4, 3), (4, 5), (5, 2), (5, 4) })

TERMINOLOGY

- A complete graph is a graph with an edge between every pair.
- If there are N nodes , there will be **N(N - 1)/2** edges in a complete graph without loop edges.
- A complete digraph is a digraph with an edge allowing traversal between every pair of nodes.
- Because the edges of a graph allow travel in two directions, whereas a digraph's edges allow travel in only one, a digraph with N nodes will have twice as many edges , specifically **N(N - 1)** edges.

TERMINOLOGY

- Let N = #vertices and M = #edges



Complete graph if **M = N(N - 1)/2**

Incomplete graph if **M < N(N - 1)/2**

N = 5

M = 5(5 - 1)/2 = 5 * 4 / 2 = 10

TERMINOLOGY

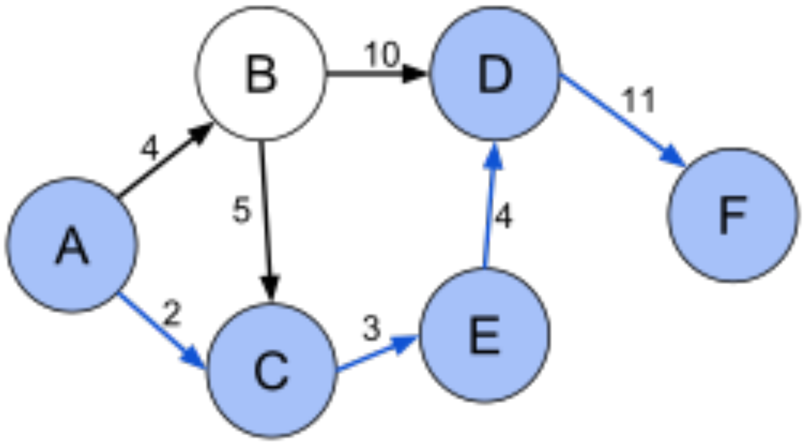
- **adjacent vertices:** vertices connected by an edge
- **degree (of vertex):** # of adjacent vertices
- **path:** sequence of vertices v1, v2, v3 such that consecutive vertices vi and vi+1 are adjacent.
- **simple path:** no repeated vertices
- **cycle:** simple path except the last vertex is the same as the first.

WEIGHTED GRAPHS

- A weighted graph is one where each edge has a value, called weight associated with it.

- In graph diagrams, the weight will be written near the edge.

- In formal definitions the weight will be an extra component in the set of an edge or ordered triplet.



WEIGHTED GRAPHS

- When working with weighted graphs we consider the weight as the **"cost"** of traversing the edge.

- A path through a weighted graph has a cost that is the sum of the weights along that path.

- In a weighted graph, the shortest path between two nodes is the path with the smallest cost, even if it does not have the fewest edges.

    - if path **P1** has *four edges* with a total cost of **20** and path **P2** has *three edges* with a total cost of **25**,

      then path **P1** will be considered the shortest path

ADJACENCY MATRIX

- An adjacency matrix for graph G = (V, E) with |V| = N will be stored as a two dimensional array of size N X N.

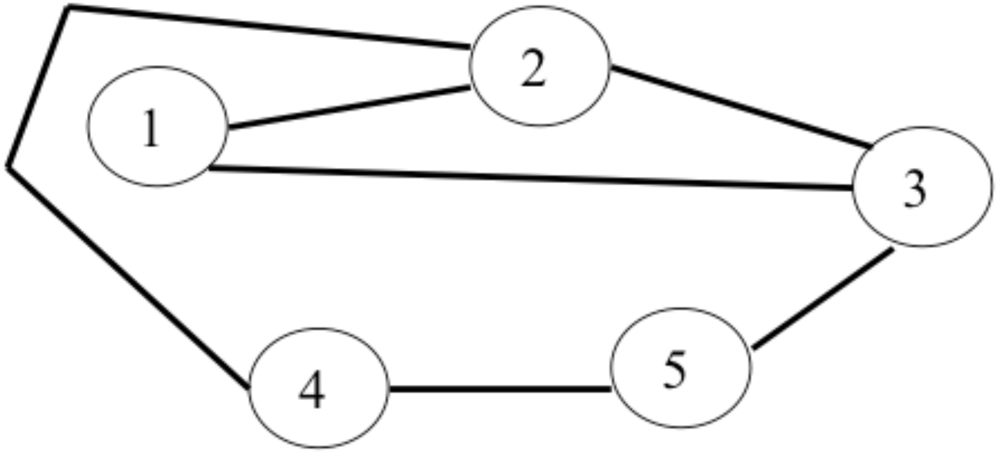
- Each location [j,k] of this array will store a zero, except if there is an edge from node v_j to node v_k the location will store a one.

- Adjacency matrix [j,k] = $\begin{cases} 1 \text{ if } v_j\, v_k \in E \\ 0 \text{ if } v_j\, v_k \notin E \end{cases}$

for all j and k in the range 1 to N

ADJACENCY MATRIX

The adjacency matrix for the graph below is given next.

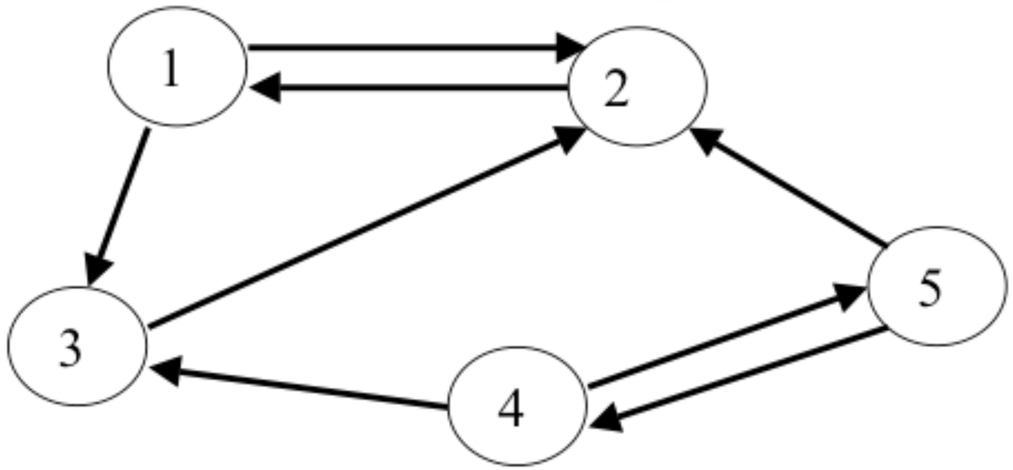


The graph G = ({1, 2, 3, 4, 5}, {{1, 2}, {1, 3}, {2, 3}, {2, 4}, {3, 5}, {4, 5}})

$$
\begin{array}{c c}
 & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} & \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}
\end{array}
$$

ADJACENCY MATRIX

The adjacency matrix for the digraph below is given next.



The digraph:
G = ({1, 2, 3, 4, 5}, {(1, 2), (1, 3), (2, 1), (3, 2), (4, 3), (4, 5), (5, 2), (5, 4)})

$$
\begin{array}{c c}
 & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} &
\begin{bmatrix}
0 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 0
\end{bmatrix}
\end{array}
$$

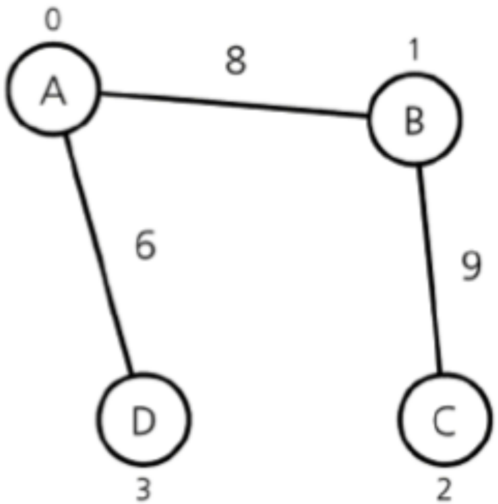
ADJACENCY MATRIX

- For weighted graphs and digraphs, the adjacency matrix entries would be (infinity) if there is no edge and the weight for the edge in all other cases.

- We can if we wish set the diagonal elements to zero because there is no cost to travel from a node to itself.

a) A weighted undirected graph and b) its adjacency matrix

(a)



(b)

		0	1	2	3
		A	B	C	D
0	A	∞	8	∞	6
1	B	8	∞	9	∞
2	C	∞	9	∞	∞
3	D	6	∞	∞	∞

ADJACENCY LISTS

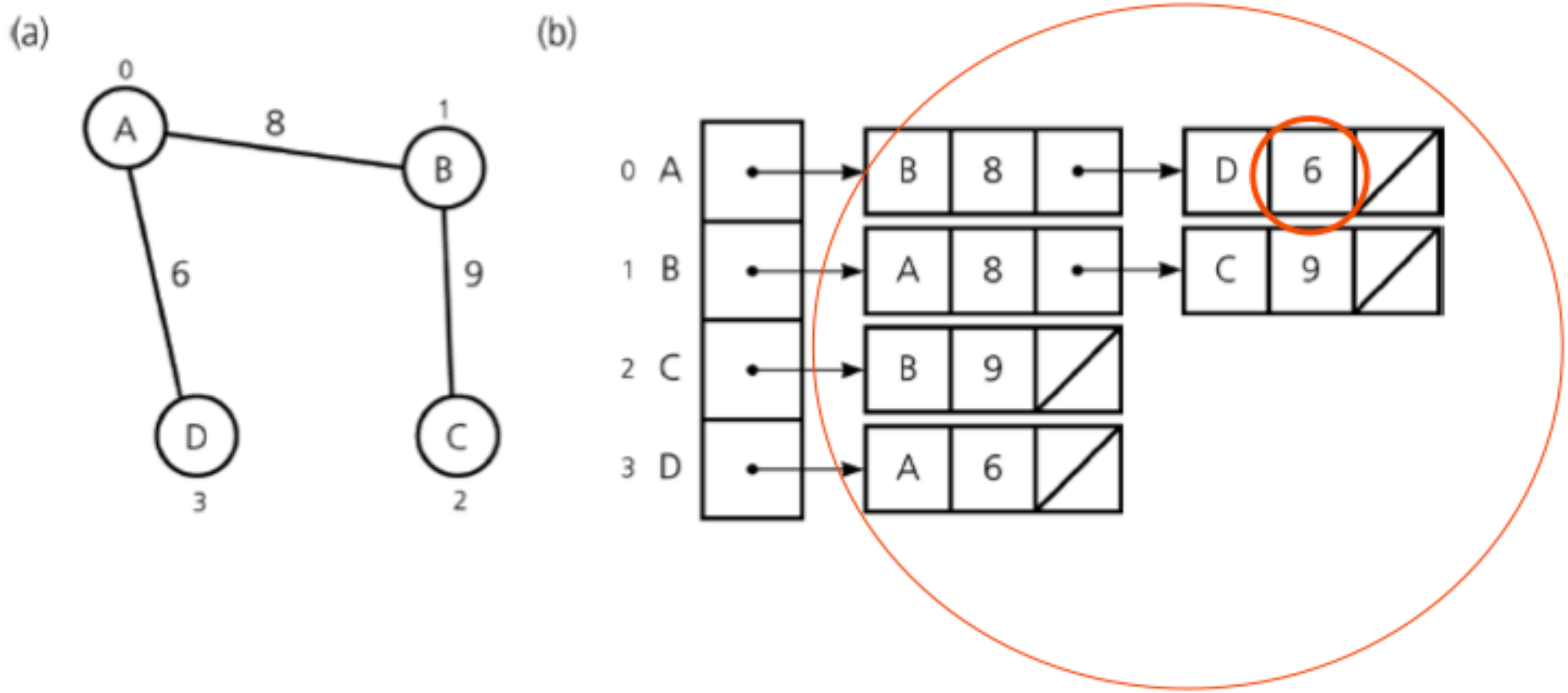
- An adjacency list for a graph $G = (V, E)$, with $|V| = N$, will be stored as a one-dimensional array of size N, with each location being a reference to a linked list.

- There will be one list for each node and the list will have one entry for each adjacent node.

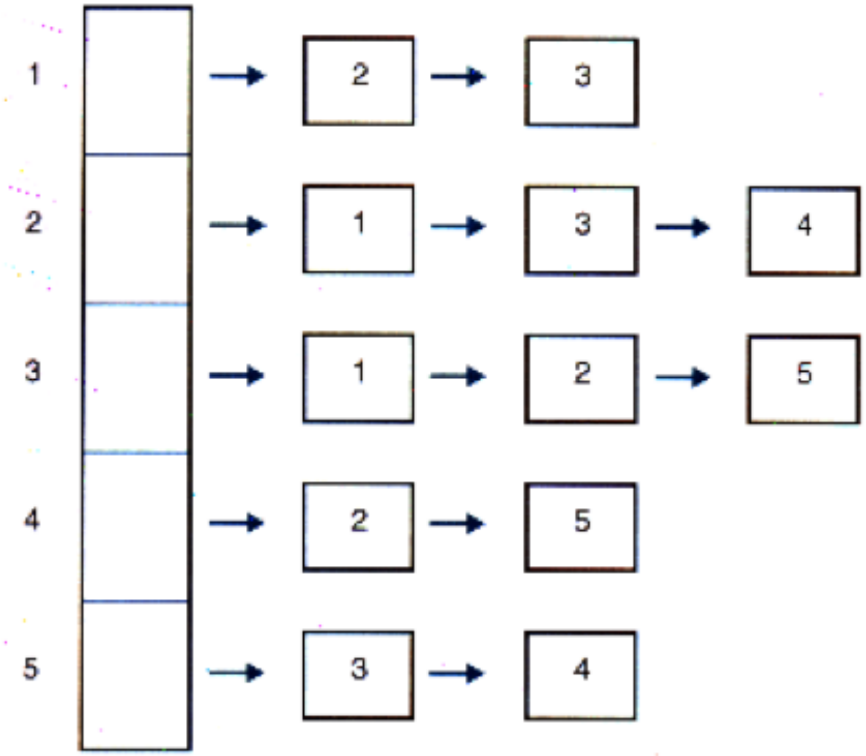
- For weighted graphs and weighted digraphs, the adjacency list entries would have an additional field to hold the weight for that edge.

ADJACENCY LISTS

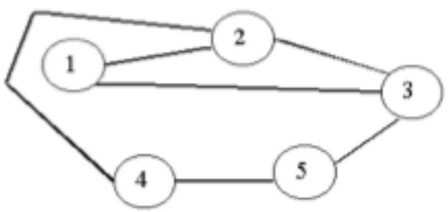
a) A weighted undirected graph and b) its adjacency list



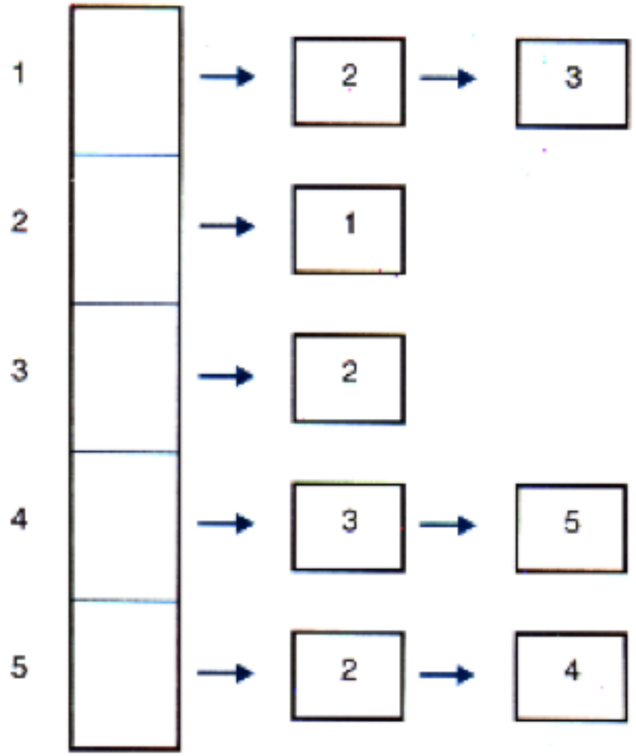
ADJACENCY LISTS



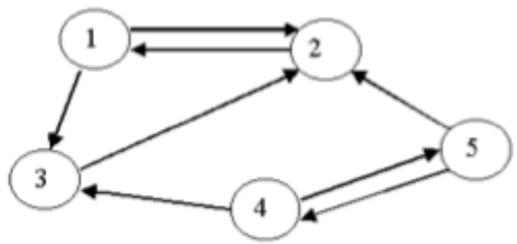
The adjacency list



The graph:
G = ({1, 2, 3, 4, 5}, {{1, 2}, {1, 3}, 2, 3}, {2, 4}, {3, 5}{4, 5}})



The adjacency list

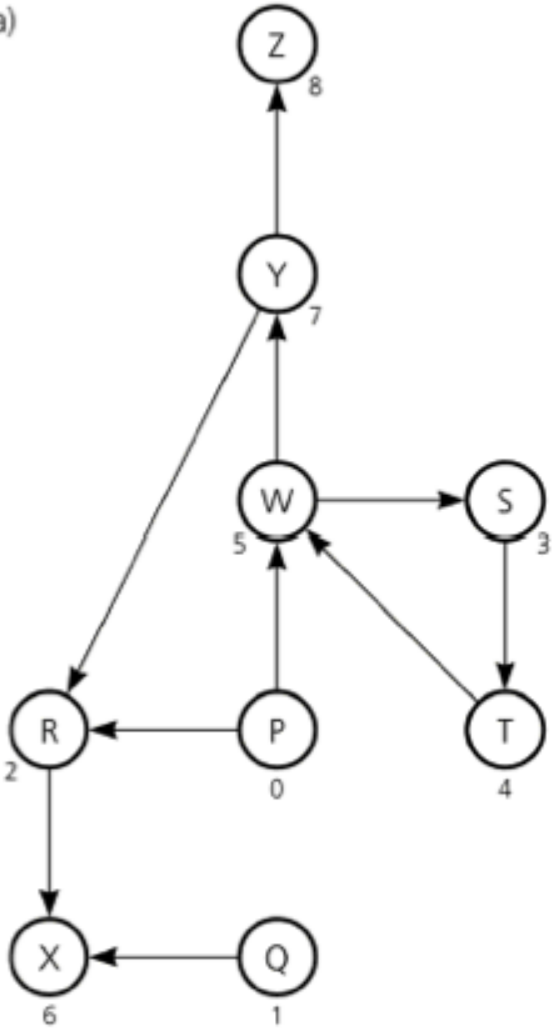


The digraph:
G = ({1, 2, 3, 4, 5), {(1, 2), (1, 3), (2, 1), (3, 2), (4, 3), (4, 5), (5, 2), (5, 4)})

ADJACENCY LISTS

a) A directed graph and b) its adjacency matrix

(a)

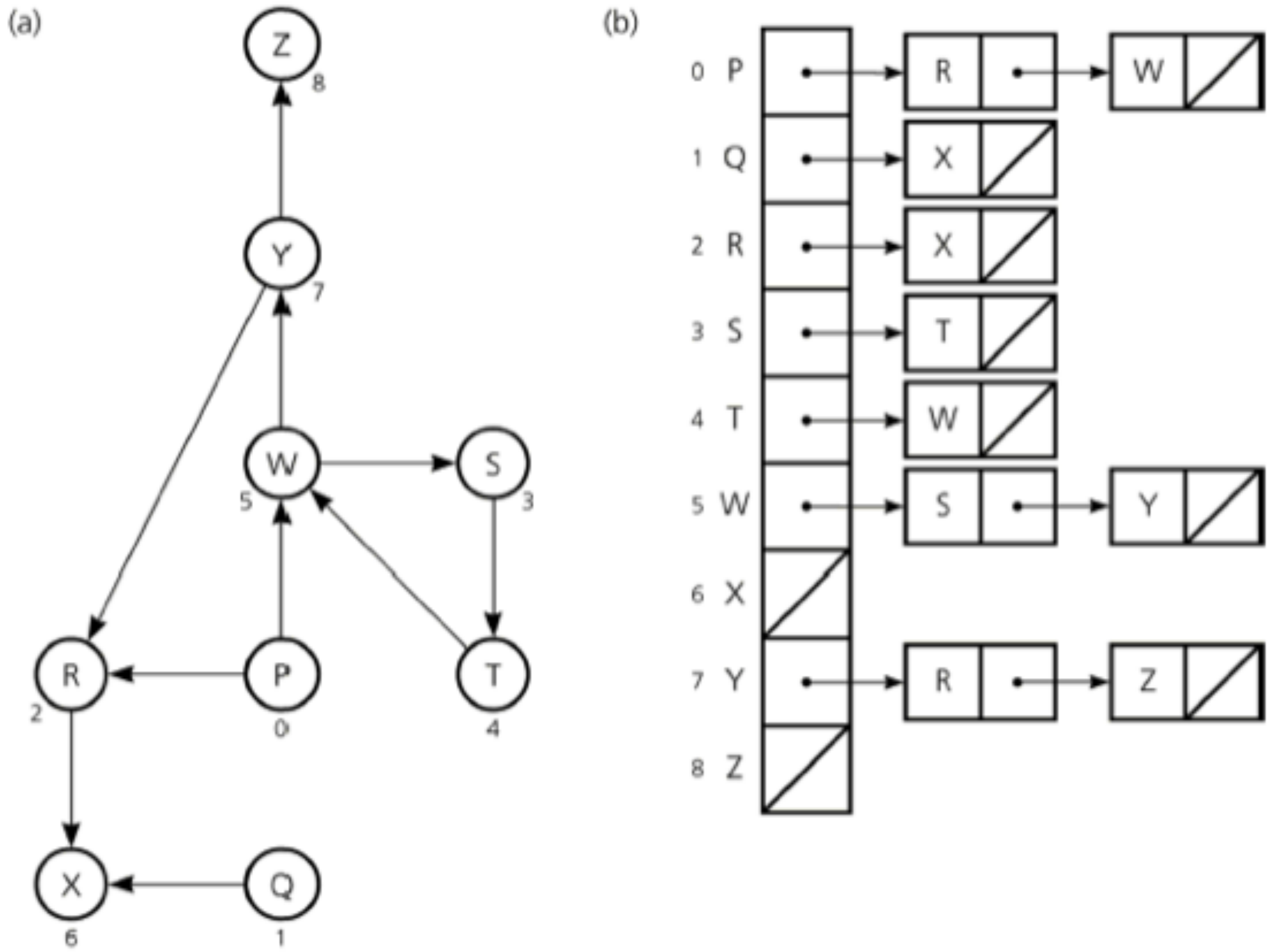


(b)

		0	1	2	3	4	5	6	7	8
		P	Q	R	S	T	W	X	Y	Z
0	P	0	0	1	0	0	1	0	0	0
1	Q	0	0	0	0	0	0	1	0	0
2	R	0	0	0	0	0	0	1	0	0
3	S	0	0	0	0	1	0	0	0	0
4	T	0	0	0	0	0	1	0	0	0
5	W	0	0	0	1	0	0	0	1	0
6	X	0	0	0	0	0	0	0	0	0
7	Y	0	0	1	0	0	0	0	0	1
8	Z	0	0	0	0	0	0	0	0	0

ADJACENCY LISTS

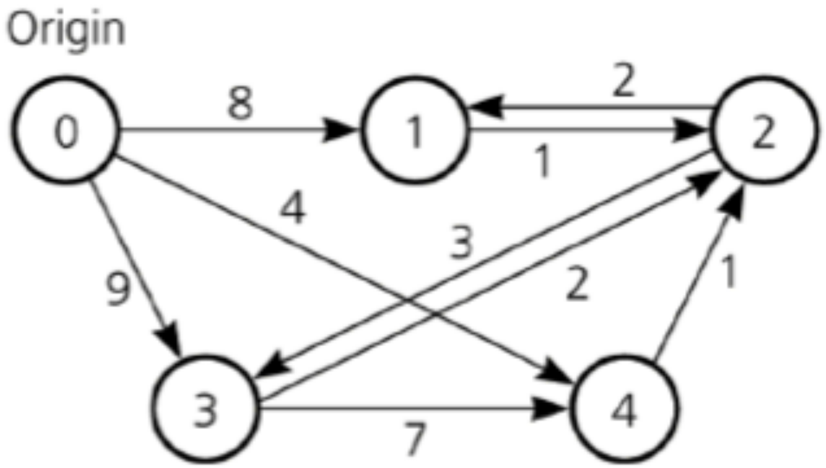
a) A directed graph and b) its adjacency list



ADJACENCY LISTS

a) A weighted directed graph and b) its adjacency matrix

(a)



(b)

	0	1	2	3	4
0	∞	8	∞	9	4
1	∞	∞	1	∞	∞
2	∞	2	∞	3	∞
3	∞	∞	2	∞	7
4	∞	∞	1	∞	∞

WHICH REPRESENTATION TO USE?

- The choice by the programmer as to either approach will be closely linked to knowledge of the graphs that will be input to the algorithm.

- In situations where the graph has many nodes, but they are connected to only a few nodes, an adjacency list would be best suited because it uses less space, and there will not be long edge lists to traverse.

- In situations where the graph has few nodes, an adjacency matrix would be best because it would not be very large, so even a sparse graph would not waste many entries.

- In situations where the graph has many edges and begins to approach a complete graph, an adjacency matrix would be best because memory requirements are similar but checking if an edge exists between two vertices takes constant time with an adjacency matrix.

GRAPH TRAVERSAL

- Typically there are two different traversals: a **depth first** traversal and a **breadth first** traversal. Both traversals visit all vertices in the graph.

- **Depth first** progresses by expanding the first vertex of the search tree going deeper and deeper until a goal vertex is found, or until It finds a vertex that has no edges. Then the search backtracks, returning to the most recent vertex it hasn't visited. In a non-recursive implementation, all vertices are added to a **stack** so that additional adjacent vertices may be visited, if any. If none, then the vertex is popped from the stack.

GRAPH TRAVERSAL

```
depthFirst(Vertex v){
 mark v visited;
 process v;

 push v on stack st;
 while (!st.isEmpty() ){
  let u = next unvisited adjacent vertex of st.top();
  mark u visited;
  process u;

  push u on stack st;
  if (all vertices of st.top() visited)
   st.pop();
  }
}
```

GRAPH TRAVERSAL

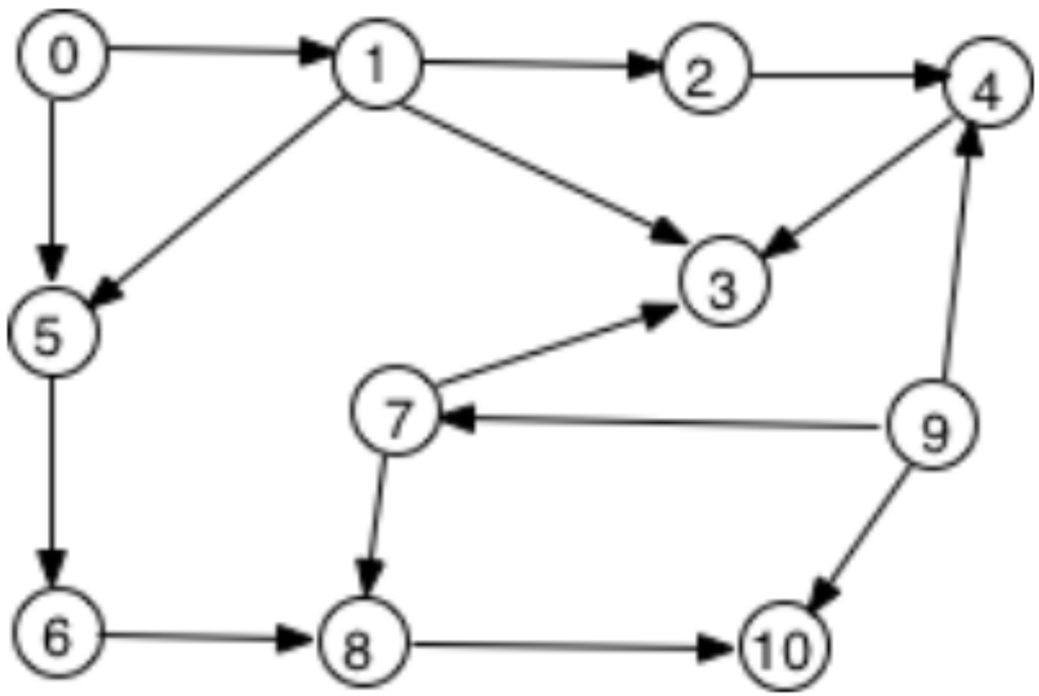
- To use this function on a graph of size gSize use the following loop:

```
for(Vertex v = 0; v < gSize; v++){
 if(! marked[v])
  depthFirst(v);
}
```

- We assume the existence of a boolean array marked of length gSize.

GRAPH TRAVERSAL

- A **depth first** traversal of the given graph gives: 0, 1, 2, 4, 3, 5, 6, 8, 10, 7, 9.



GRAPH TRAVERSAL

- A **breadth first** traversal visits all the vertices at a given level before visiting the vertices at the next level. An iterative algorithm for breadth first uses a **queue** to store next level vertices while processing higher-level vertices.

GRAPH TRAVERSAL

```
breadthFirst(Vertex v){
mark v visited;
process v;

for(u : v.list()){
 if(!marked(u)){
  process u;
  mark u visited; q.add(u);
 }
 while (!q.isEmpty() ){
  u = q.front(); q.leave();
  for(t : u.list())
   if(!marked(t)){
    process t;
    mark(t);
    q.add(t);
  }
 }
}
```

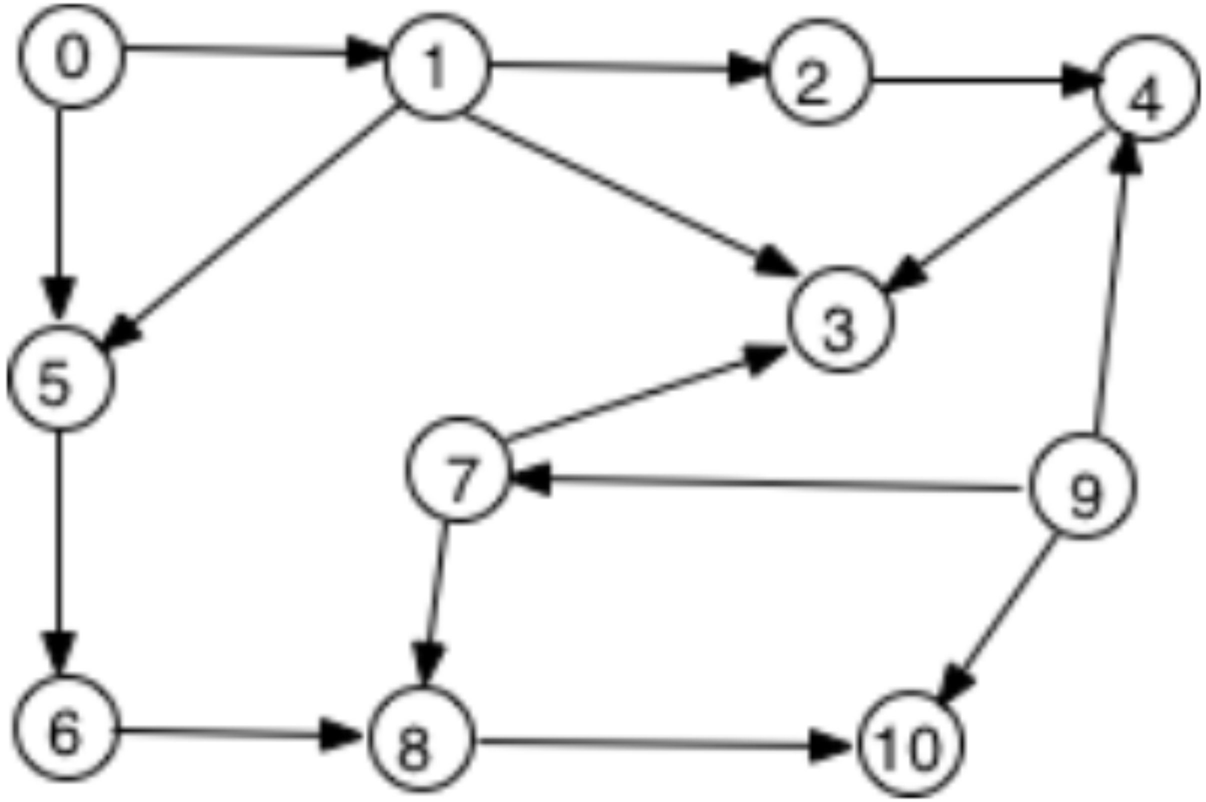
GRAPH TRAVERSAL

- To use this function on a graph of size gSize use the following loop:

```
for(Vertex v = 0; v < gSize; v++){
  if(! marked[v])
    breadthFirst(v);
}
```

GRAPH TRAVERSAL

- A breadth first traversal of the given graph gives: 0, 1, 5, 2, 3, 6, 4, 8, 10, 7, 9.



TODO - WEEK 6

TO DO this week

1. Draw the following graph:

   G = ({1, 2, 3,4, 5, 6}, {{1, 2}, {1, 4}, {2, 5}, {2, 6}, {3, 4}, {3, 5}, {3, 6}, {4, 5}, {4, 6}, {5, 6}}).

- Give the adjacency matrix for this graph.
- Give the adjacency list for this graph.

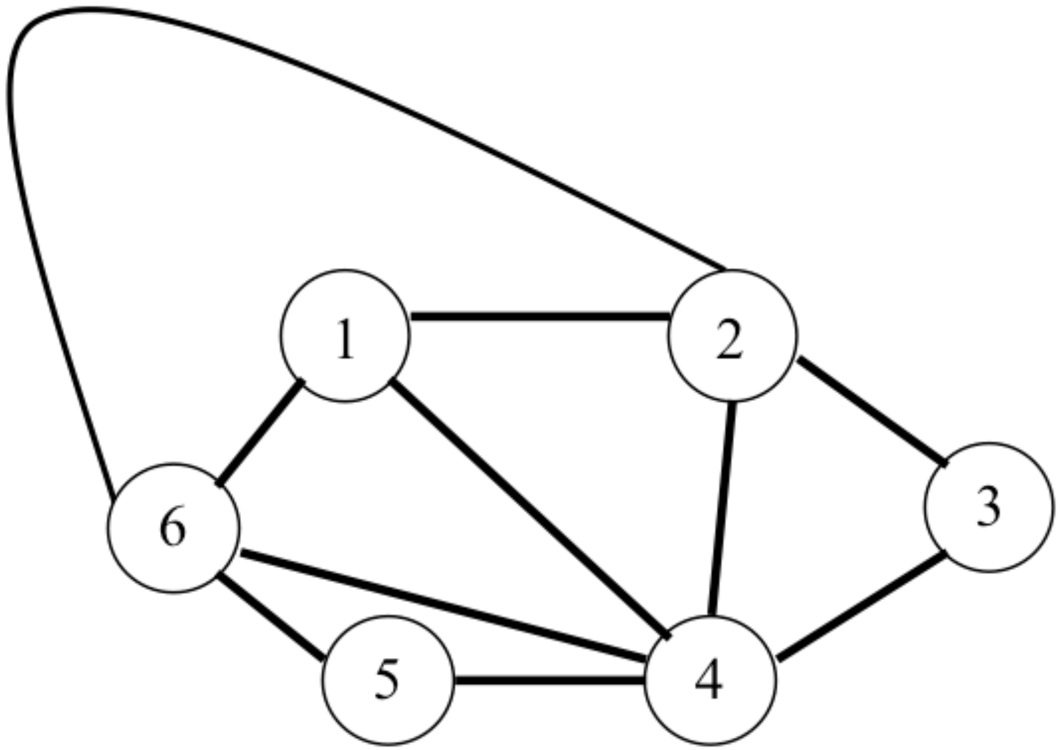
2. Draw the following digraph:

   G = ({1, 2, 3, 4, 5}, {(1, 2), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5), (3, 2), (3, 4), (3, 5), (4, 1), (4, 2), (4, 5), (5, 2), (5, 3), (5, 4)}).

- Give the adjacency matrix for this digraph.
- Give the adjacency list for this digraph.

TODO - WEEK 6

3. Give the set description for the following graph:

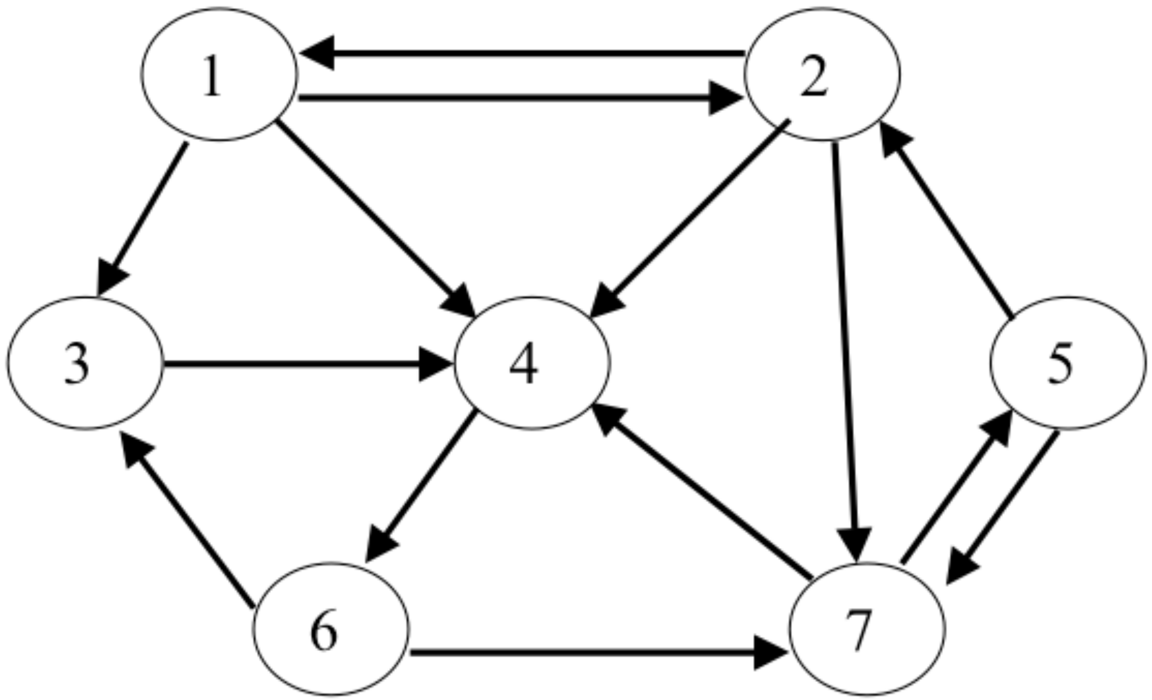


- List all the paths between node 1 and node 5 in the above graph.
- List all the cycles that start at node 3 in the above graph.

- Give the adjacency matrix for this graph.
- Give the adjacency list for this graph.

TODO - WEEK 6

4. Give the set description for the following digraph:



- List all of the paths between node 1 and node 4 in the above digraph.
- List all of the cycles that start at node 7 in the above digraph.

- Give the adjacency matrix for this digraph.
- Give the adjacency list for this digraph.