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:

          <u>O(max(n,e))</u>

- **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;
```

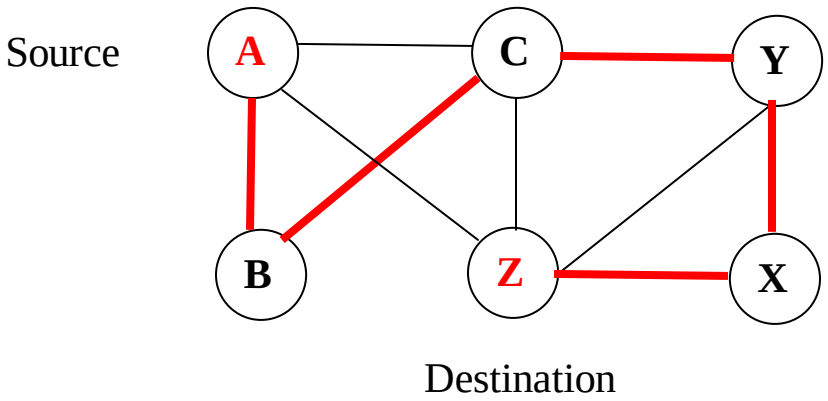
- **Analysis:**
  - For G=(V,E) where n=$|V|$ and e=$|E|$, the time complexity is:
    - Adjacency matrix:
      - Since the while loop takes O(n) for each vertex, the time complexity is: **<u>$O(n^2)$</u>**

    - 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
}
```