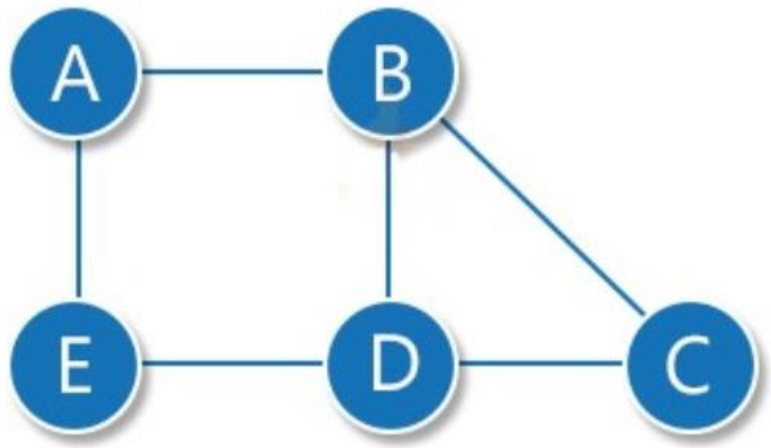
# Graph

- A Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph.




Vertices : A, B, C, D, E
Edges: A-B, A-E, B-C, B-D, D-E, C-D

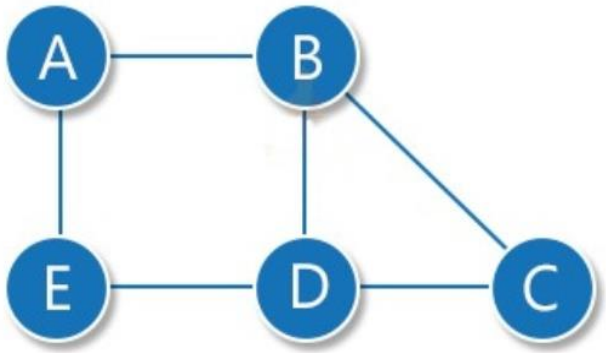
- More formally a Graph is composed of a set of vertices( V ) and a set of edges( E ). The graph is denoted by G(E, V).

# Graph Terminology

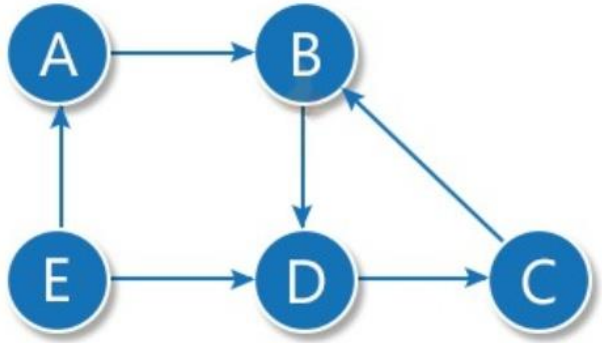
- **Vertices:** Vertices are the fundamental units of the graph. Sometimes, vertices are also known as vertex or nodes. Every node/vertex can be labeled or unlabelled.
- **Edges:** Edges are drawn or used to connect two nodes of the graph. It can be ordered pair of nodes in a directed graph.
- **Path:** It is the sequence/order of vertices to reach from one node to another.

# Directed and Undirected Graphs

- Types of graph based on edges-



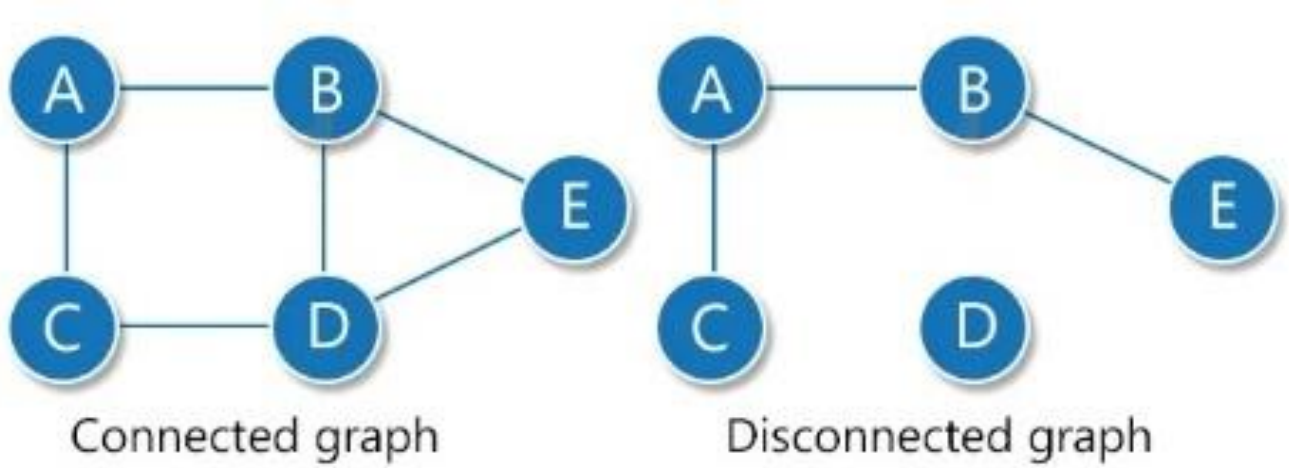

Undirected Graph




Directed Graph

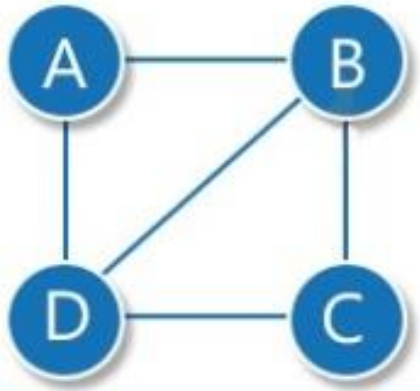
# Connected and Disconnected Graphs

- If there is a path between every pair of vertices, then the graph is called a connected graph.



# Graph Representation

- Adjacency Matrix
- An adjacency matrix is a 2D array of V x V vertices. Each row and column represent a vertex.

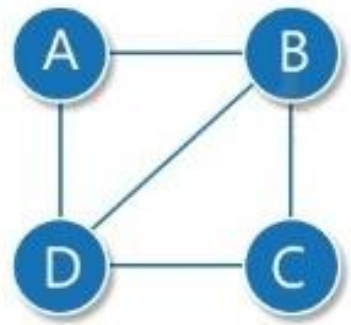



Undirected graph

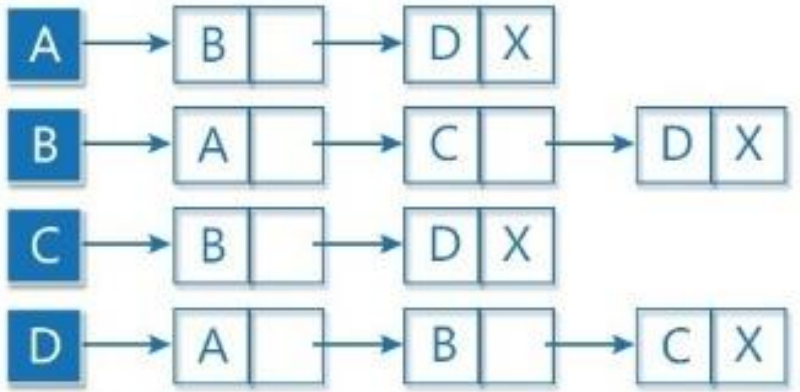
| | A | B | C | D |
|---|---|---|---|---|
| A | 0 | 1 | 0 | 1 |
| B | 1 | 0 | 1 | 1 |
| C | 0 | 1 | 0 | 1 |
| D | 1 | 1 | 1 | 0 |

Adjacency matrix

- **Adjacency List**
- An adjacency list represents a graph as an array of linked lists.
- The index of the array represents a vertex and each element in its linked list represents the other vertices that form an edge with the vertex.



Undirected graph



Adjacency List

# Searching and Traversing a Graph

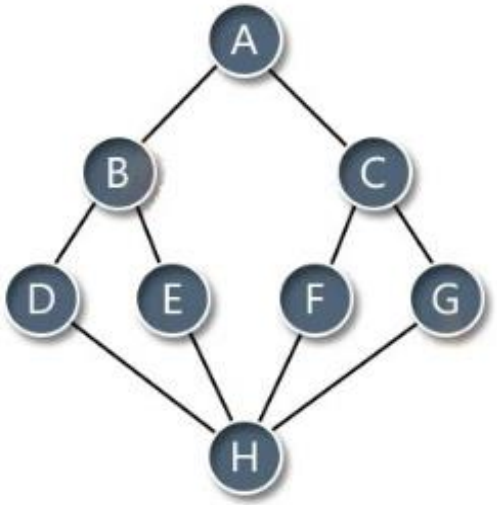
- Breadth-first-search (Queue)
- Depth-first-Search  (Stack)

# Pseudo code BFS

```
BFS (G, s)                  //Where G is the graph and s is the source node
    let Q be queue.
    Q.enqueue( s )  //Inserting s in queue
    mark s as visited.
    while ( Q is not empty)
        //Removing that vertex from queue
        v  =  Q.dequeue( )
       //processing all the neighbours of v
       for all neighbours w of v in Graph G
           if w is not visited
                   Q.enqueue( w )          //Stores w in Q to further visit its neighbour
                   mark w as visited.
```

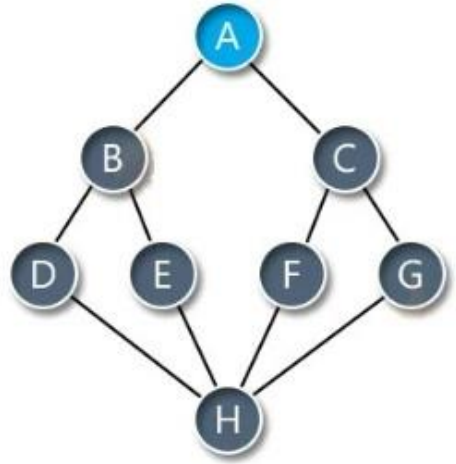
# BFS

- BFS Example:-



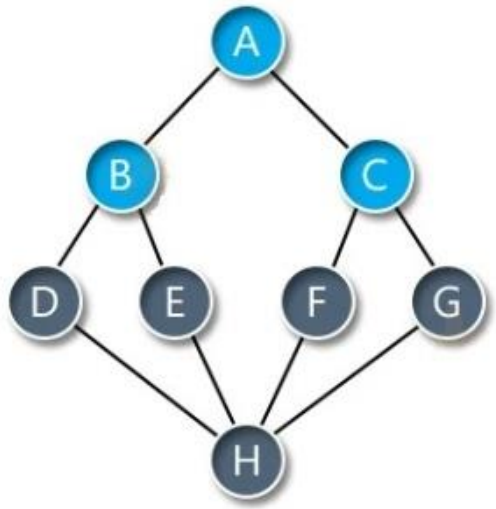

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Visited: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Queue: **Empty**




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Visited: | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

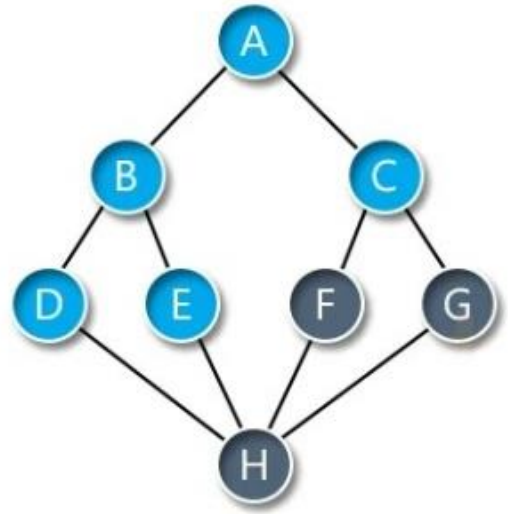
Queue: **A**




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Visited: | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

Queue: **B,C**

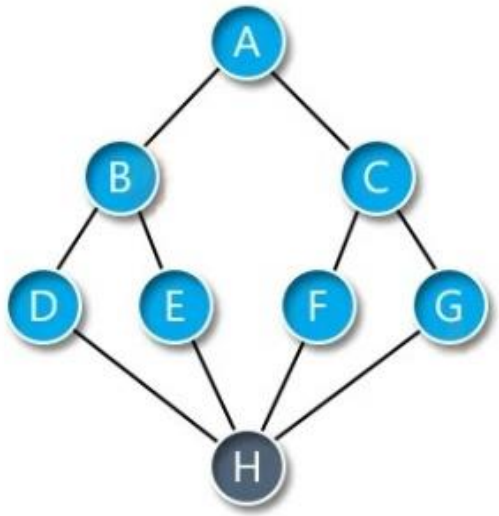
Output: **A**




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Visited: | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

Queue: **C,D,E**

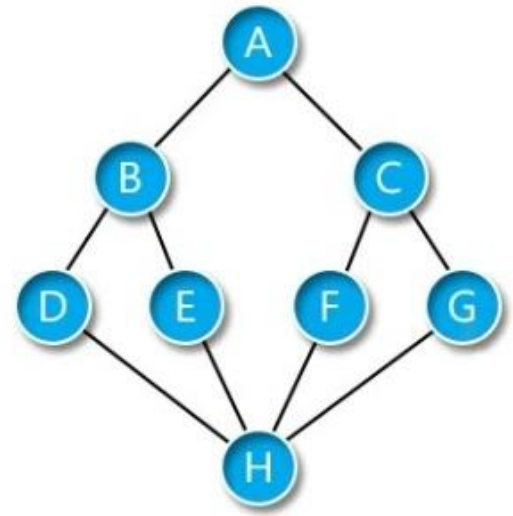
Output: **A,B**




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Visited: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

Queue: **D,E,F,G**

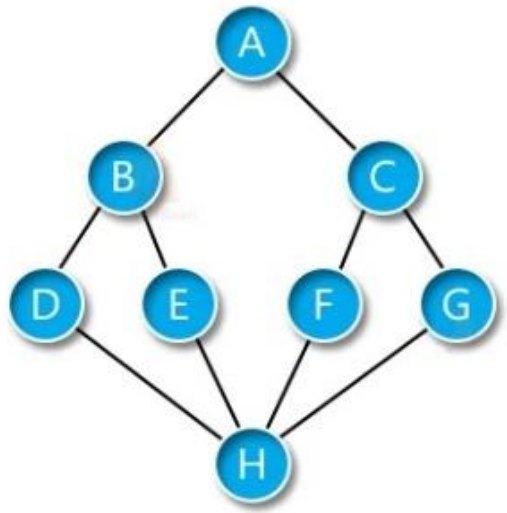
Output: **A,B,C**




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Visited: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Queue: **E,F,G,H**

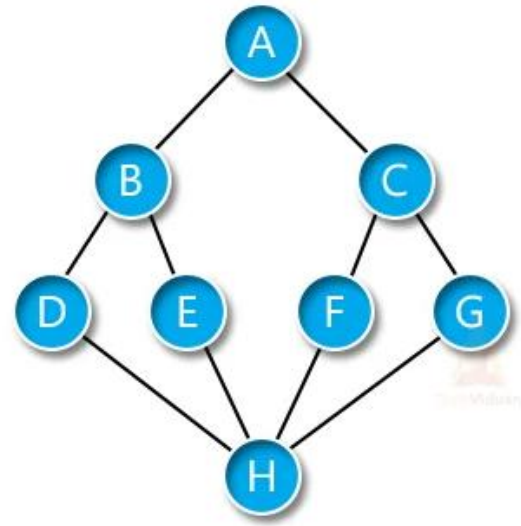
Output: **A,B,C,D**




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Visited: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Queue: **F,G,H**

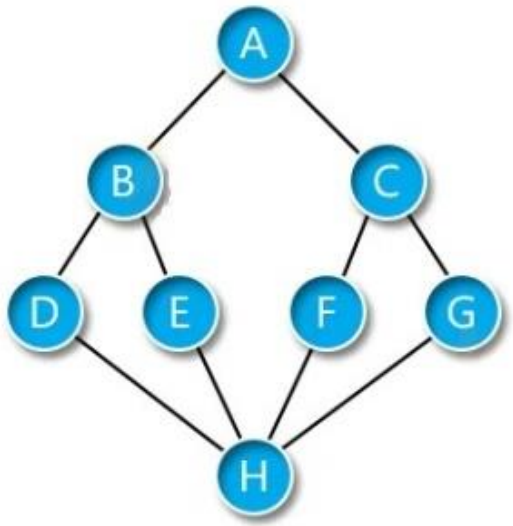
Output: **A,B,C,D,E**




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Visited: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Queue: **G,H**

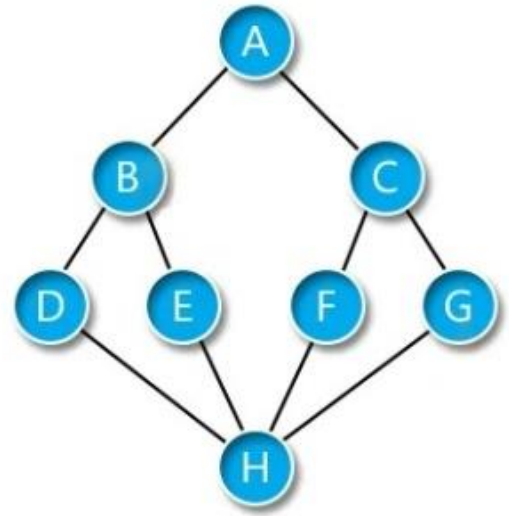
Output: **A,B,C,D,E,F**




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Visited: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Queue: **H**

Output: **A,B,C,D,E,F**



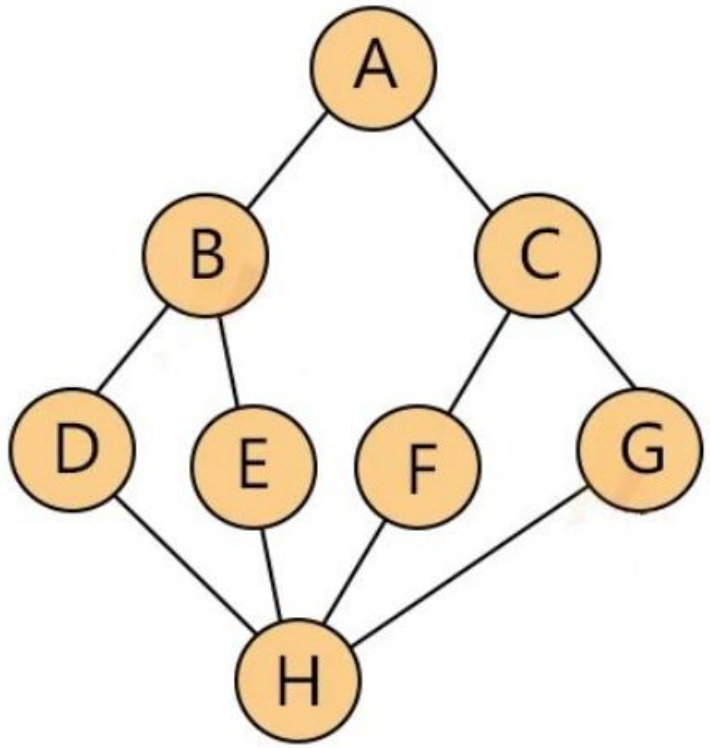

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Visited: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

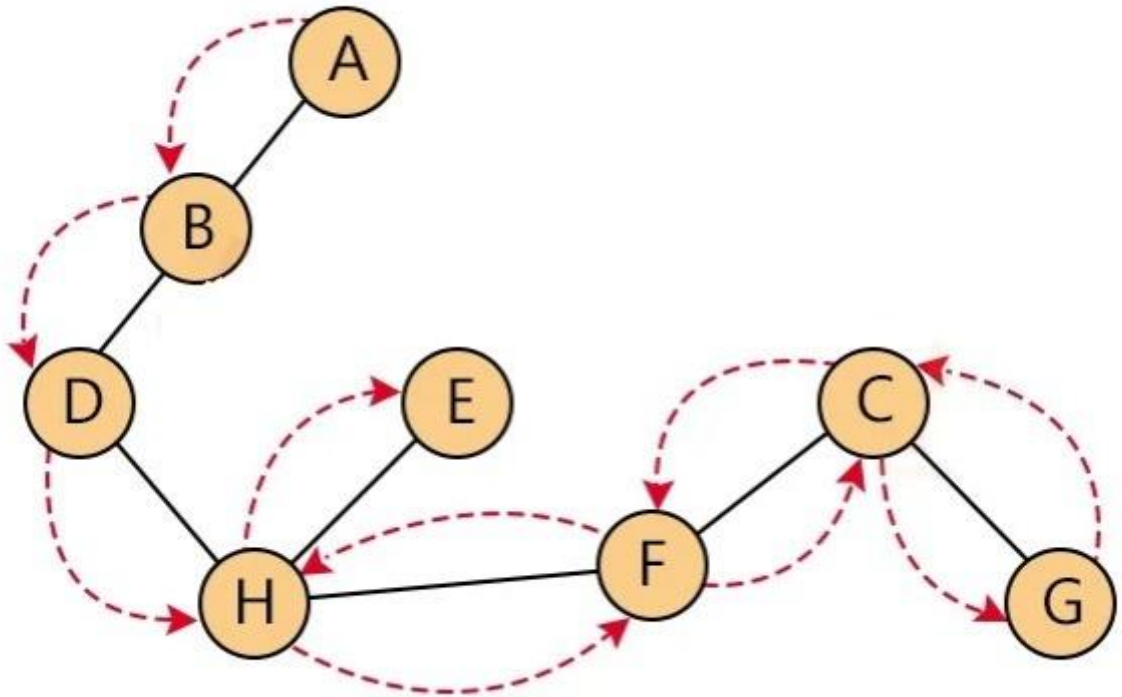
Queue: **Empty**

Output: **A,B,C,D,E,F,G,H**

# DFS

DFS Example:-




DFS – A B D H F C G E

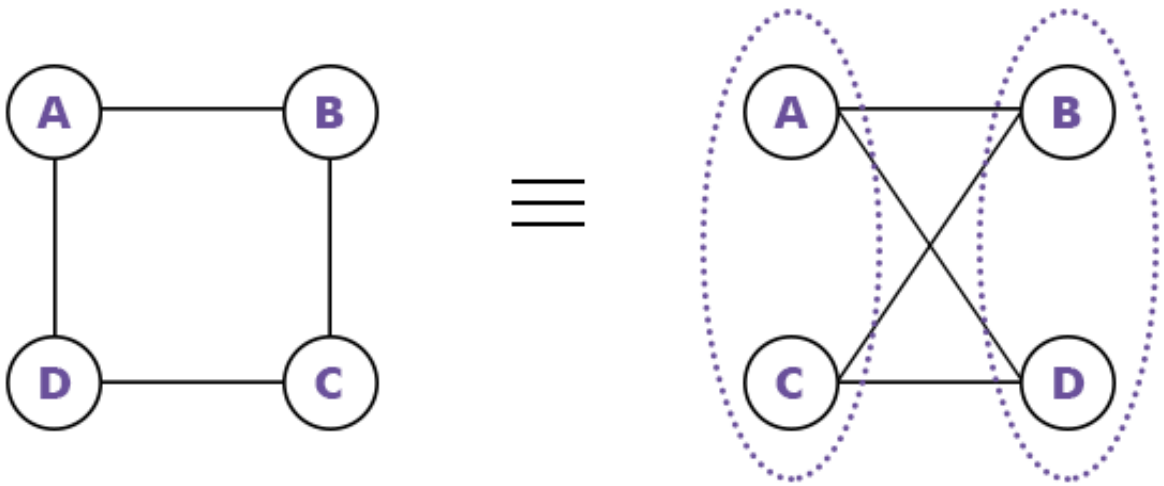
# Pseudo Code for DFS

```
DFS(G, u)
    u.visited = true
    for each v ∈ G.Adj[u]
        if v.visited == false
            DFS(G,v)

main() {
    For each u ∈ G
        u.visited = false
     For each u ∈ G
       DFS(G, u)
}
```

# Bipartite Graph

- A bipartite graph is a graph in which the vertices can be divided into two disjoint sets, such that no two vertices within the same set are adjacent.



# Diameter of the graph

- The diameter of graph is the maximum distance between the pair of vertices.