Leaning Objective:

In this Module you will be learning the following:

- Graph Traversals

Introduction:

In computer science, **graph traversal** (also known as **graph** search) refers to the process of visiting (checking and/or updating) each vertex in a **graph**.

Material:

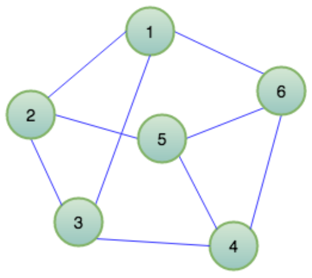
Graph traversal is technique used for searching a vertex in a graph. The graph traversal is also used to decide the order of vertices to be visit in the search process. A graph traversal finds the edges to be used in the search process without creating loops that means using graph traversal we visit all vertices of graph without getting into looping path.

There are two graph traversal techniques and they are as follows...

1. **DFS (Depth First Search)**
2. **BFS (Breadth First Search)**

DFS (Depth First Search)

Depth First Search or traversal explores a given graph depth wise, that means we go down the graph from one node to another till there is no unexplored node down. Once we reach the end of a path going down, backtracking starts and parent of current node is looked for any other unexplored child nodes, and so on.



For above graph depth first search will be 1->6->5->2->3->4

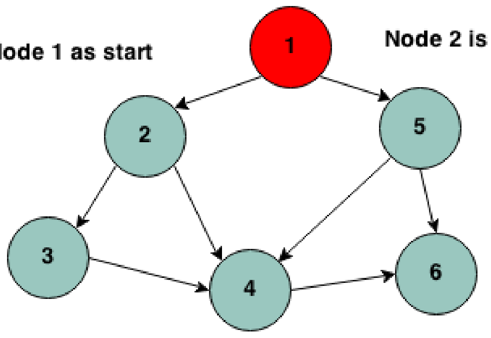
Depth First Search Algorithm

```
1. Start with a node S, and mark it as visited.
2. Current node u = S
3. While there is edge to be explored
   3.1 Move to v where there is an edge (u,v).
   3.2 If v is already not visited, mark v as visited
   3.3 change u to v and repeat.
4. Move to parent of u.
```

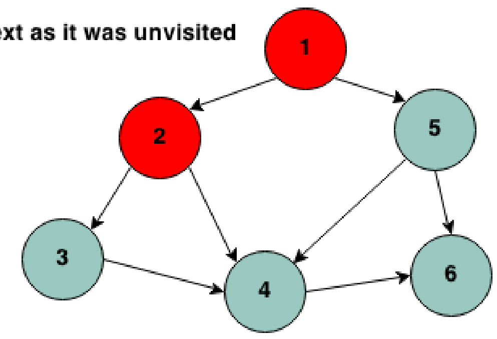
Recursive implementation of above algorithm of depth first search on graph is very simple as shown below

```
1  procedure DFS(G,v):
2      label v as discovered
3      for all edges from v to w in G.adjacentEdges(v) do
4          if vertex w is not labeled as discovered then
5              recursively call DFS(G,w)
```

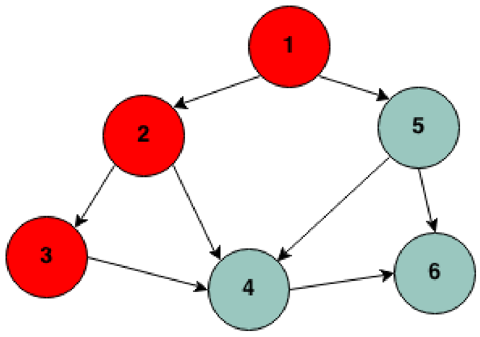
Take Node 1 as start



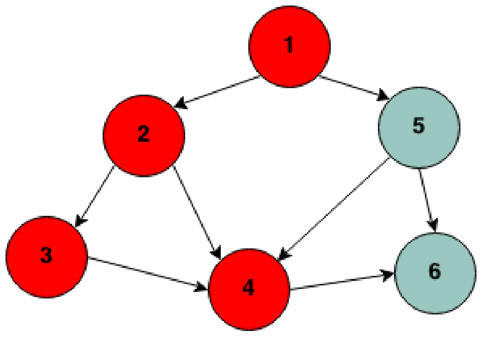
Node 2 is visited next as it was unvisited



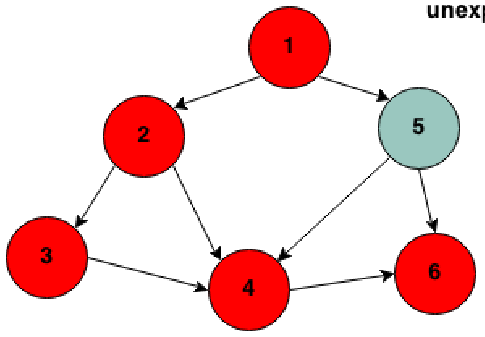
Node 3 is visited next as it was unvisited



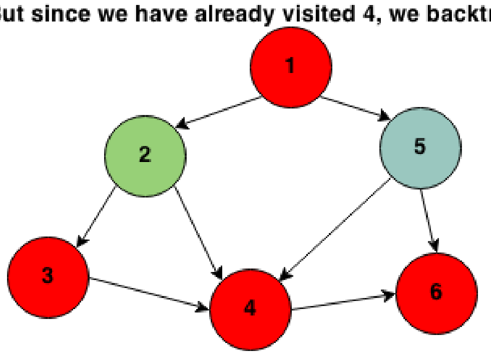
Node 4 is visited next as it was unvisited



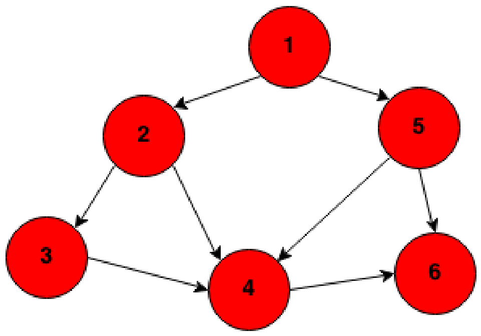
Node 6 is visited next as it was unvisited



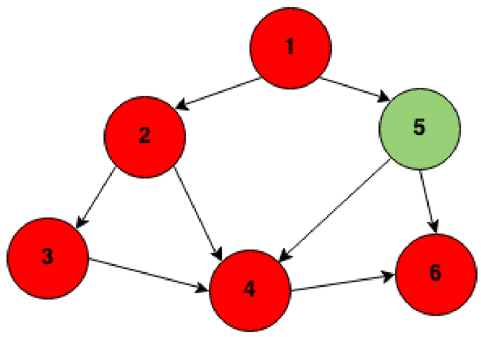
As there is no more nodes which can be traversed from 6, we back track, first node where there is unexplored edge is 2. But since we have already visited 4, we backtrack further



Node 5 from node 1 is visited next as it was unvisited



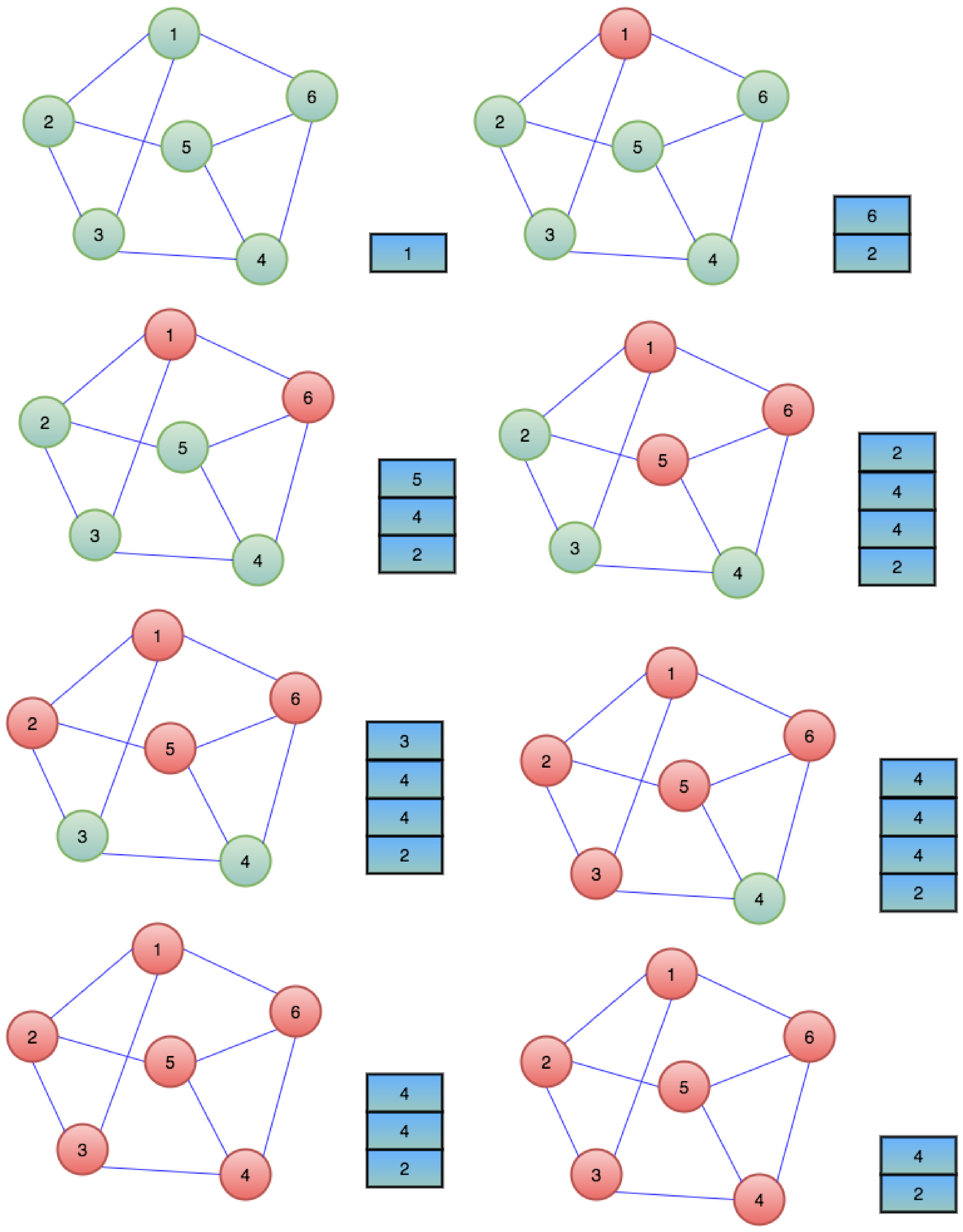
Again all neighbor nodes from 5 are already visited, hence we back track to node 1



Iterative implementation of depth first search

```
1  procedure DFS-iterative(G,v):
2      let S be a stack
3      S.push(v)
4      while S is not empty
5          v = S.pop()
6          if v is not labeled as discovered:
7              label v as discovered
8              for all edges from v to w in G.adjacentEdges(v) do
9                  S.push(w)
```

Below figure explains how iterative depth first traversal works using stacks.



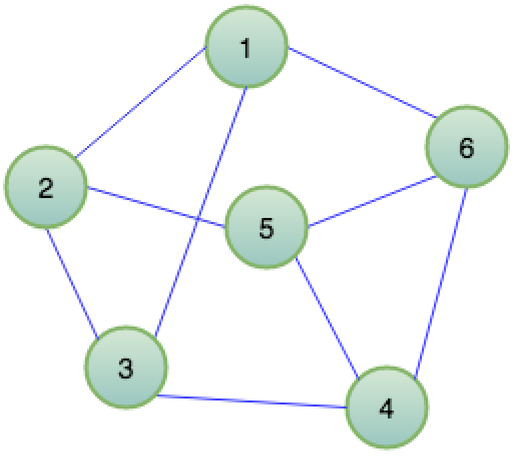
And so on...

Applications of depth first search

- Minimum spanning tree
- To check if graph has a cycle.
- Topological sorting
- To find strongly connected components of graph
- To find bridges in graph

Breadth First Search

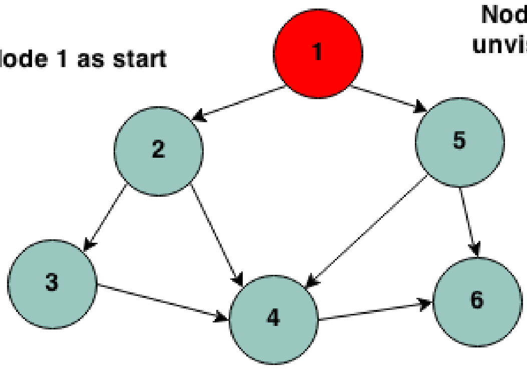
First of all, what is breadth first search? In BFS, all neighbors of a node are visited before any of neighbor of those neighbors. Basically, we traverse graph layer by layer. For example, BFS of below graph would be 1->2->6->3->5->4

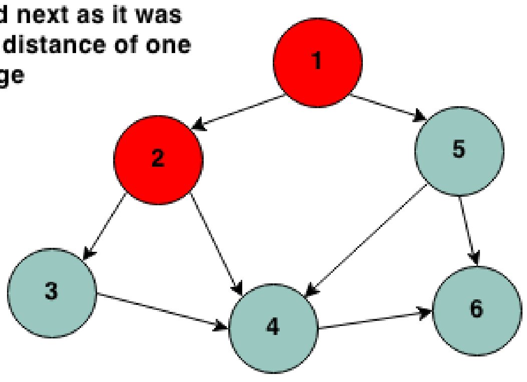


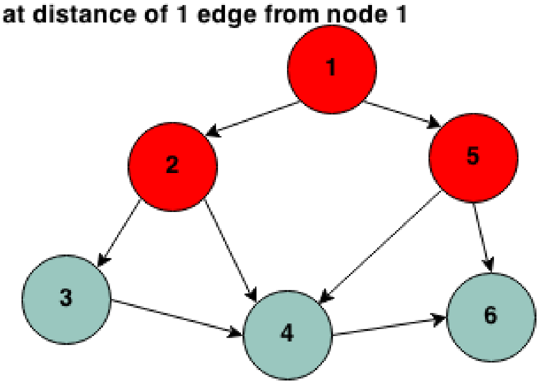
In depth first search, we go deep into the graph till there is not further move possible and then backtrack and again do the same process with another neighbor of parent and so on till we don't have any further node to visit.

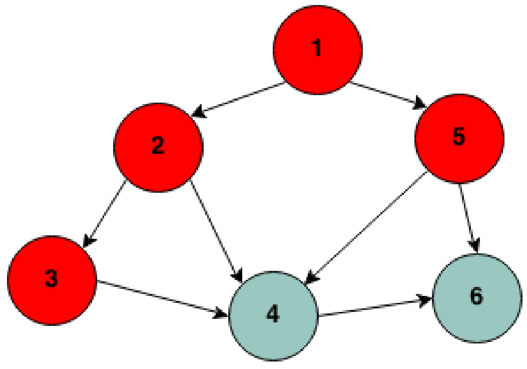
In breadth first search, as we traverse layer by layer, there is no backtracking required. Before reaching to layer i, all the nodes till layer i-1 would have been already visited.

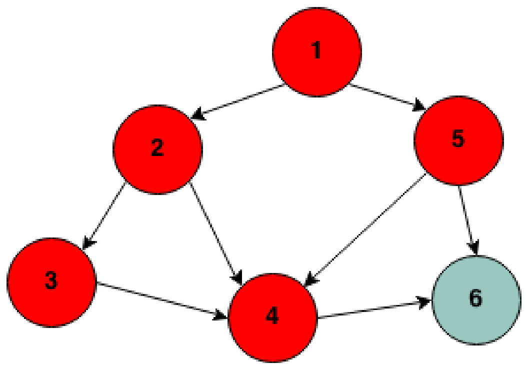
Start from node S. Let S be node u. Mark it as visited. Visit all nodes which are at one edge distance from u. Once all nodes are visited, make each neighbor node as u one by one and repeat above steps till the time there is some unexplored edges. Below figure explains step by step how BFS done.

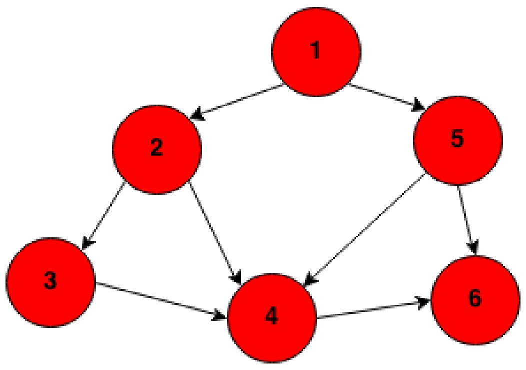

Take Node 1 as start


Node 2 is visited next as it was unvisited and at distance of one edge


Node 5 is visited next as it was unvisited and at distance of 1 edge from node 1


Node 3 is visited next as it was unvisited and at distance of one edge from node 2


Node 4 is visited next as it was unvisited and at distance of one edge from node 2

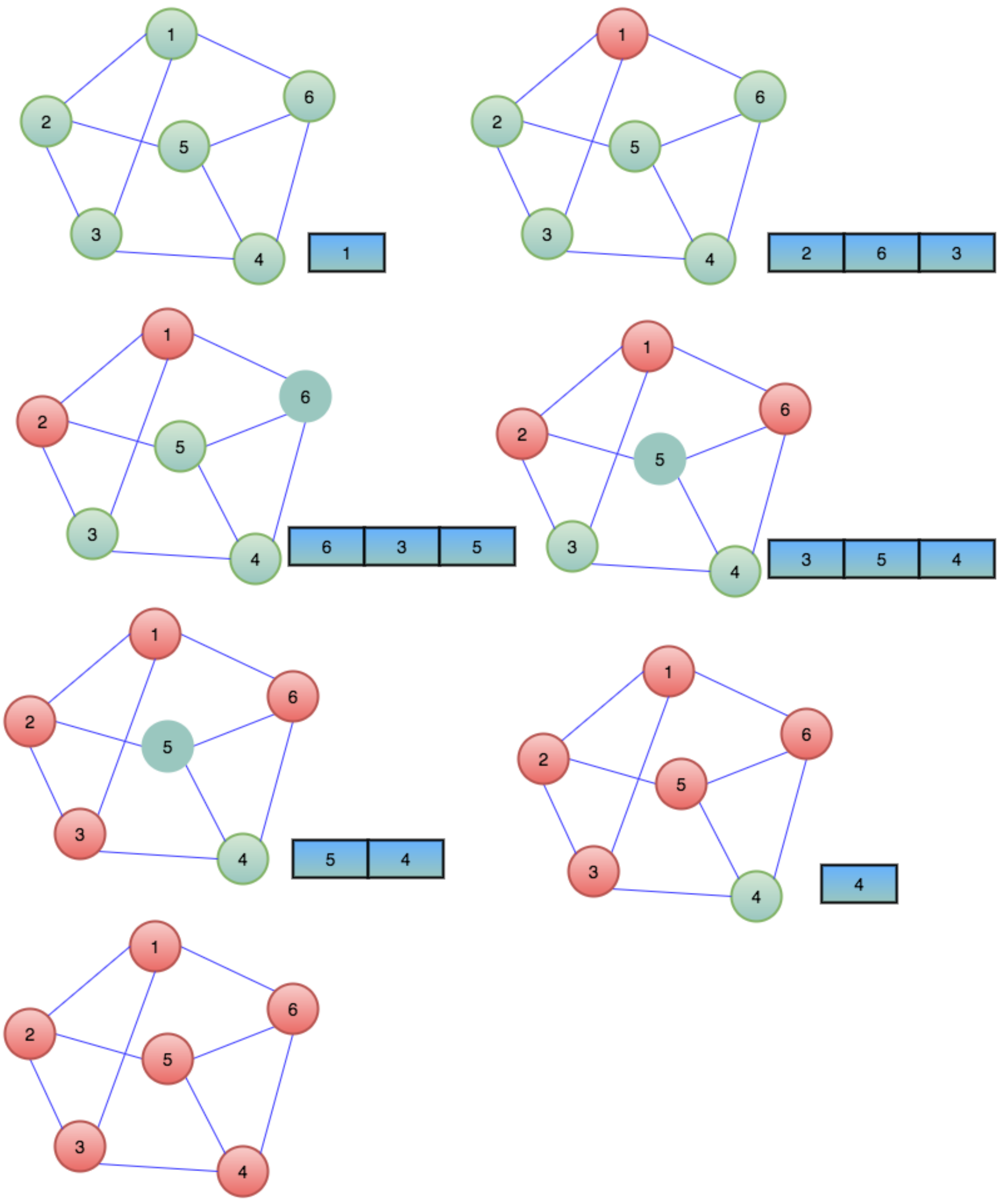

Node 6 is visited next as it was unvisited and at distance of one edge from node 5

Implementation wise, BFS uses queue unlike DFS which uses stack. While processing node, all neighbors of that node are enqueued in queue and processed one after the other in FIFO order. This ensures that all nodes at same level are processed before next level is started.

Breadth-First-Search(Graph,root):

```
 for each node u in Graph:
   create empty queue Q
   Q.enqueue(root)
   while Q is not empty:
      current=Q.dequeue()
      print current->value
      visited[v] = true
      for each v that is adjacent to current and not visited
          Q.enqueue(v)
```

Below figure explains how breadth first implementation works



Applications of BFS

1. To find shortest path between two nodes u and v in an unweighted graph.
2. In Peer to Peer Networks like BitTorrent, Breadth First Search is used to find all neighbor nodes.
3. Search engine crawlers, each page leads to more pages linked to current page and so on.
4. Social networks, BFS is used to find degree of separation, friends within K distance etc.
5. Broadcasting protocol in networks uses BFS to broadcast packets to all neighboring devices.
6. To test bipartite-ness of a graph
7. To find all nodes within one connected component of graph.

Complexity of **Breadth First Search** is O(|V| +|E|) where V is number of vertices in graph and E is number of edges in graph.

Problems Sets:

1. Write a C program to implement the DFS using List/Adjacency Matrix representation?
2. Write a C program to implement the BFS using List/Adjacency Matrix representation?
3. Write a C program to implement the Topological Sorting?