Lab-8: Uninformed Search (Breadth-First-Search)

8.1 Objectives:

1. To learn and Implement Breadth-First-Search algorithm

8.2 Breadth-first search:

Breadth-first search (BFS) is an algorithm that is used to graph data or searching tree or traversing structures. The full form of BFS is the Breadth-first search.

The algorithm efficiently visits and marks all the key nodes in a graph in an accurate breadthwise fashion. This algorithm selects a single node (initial or source point) in a graph and then visits all the nodes adjacent to the selected node. Remember, BFS accesses these nodes one by one.

Once the algorithm visits and marks the starting node, then it moves towards the nearest unvisited nodes and analyses them. Once visited, all nodes are marked. These iterations continue until all the nodes of the graph have been successfully visited and marked.

6.1.1 Graph traversals

A graph traversal is a commonly used methodology for locating the vertex position in the graph. It is an advanced search algorithm that can analyze the graph with speed and precision along with marking the sequence of the visited vertices. This process enables you to quickly visit each node in a graph without being locked in an infinite loop.

6.1.2 How BFS works:

1. Graph traversal requires the algorithm to visit, check, and/or update every single un-visited node in a tree-like structure. Graph traversals are categorized by the order in which they visit the nodes on the graph.

2. BFS algorithm starts the operation from the first or starting node in a graph and traverses it thoroughly. Once it successfully traverses the initial node, then the next non-traversed vertex in the graph is visited and marked.

3. Hence, you can say that all the nodes adjacent to the current vertex are visited and traversed in the first iteration. A simple queue methodology is utilized to implement the working of a BFS algorithm

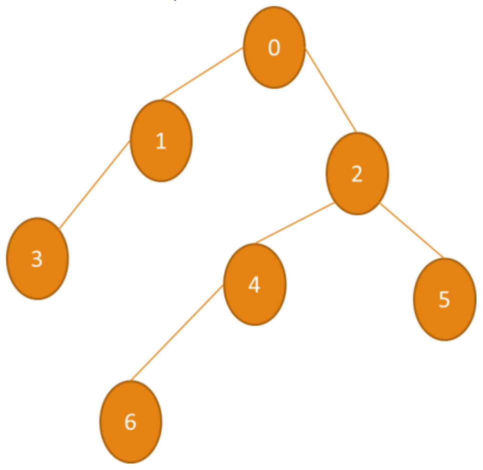
6.1.3 Iterative Deepening Depth-first search:

Iterative Deepening Depth-first search (ID-DFS) is a state space/graph search strategy in which a depth-limited version of depth-first search is run repeatedly with increasing depth limits until the goal is found. IDDFS is equivalent to breadth-first search, but uses much less memory; on each iteration, it visits the nodes in the search tree in the same order as depth-first search, but the cumulative order in which nodes are first visited is effectively breadth-first.

6.1.4 Pseudocode:

```
Set all nodes to "not visited";

  q = new Queue();

  q.enqueue(initial node);

  while ( q ≠ empty ) do
  {
    x = q.dequeue();

    if ( x has not been visited )
    {
      visited[x] = true;        // Visit node x !

      for ( every edge (x, y)  /* we are using all edges ! */ )
        if ( y has not been visited )
               q.enqueue(y);       // Use the edge (x,y) !!!
    }
  }
```

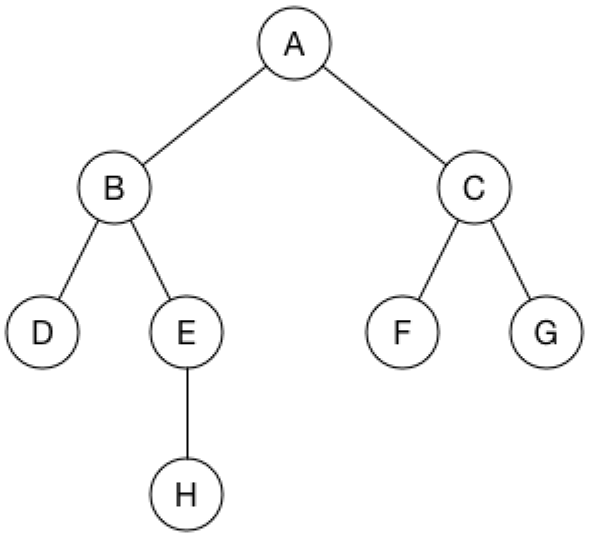
Example:



Code:

```
vertexList = ['0', '1', '2', '3', '4', '5', '6']
edgeList = [(0,1), (0,2), (1,0) , (1,3) , (2,0) , (2,4) , (2,5) , (3,1),
(4,2) , (4,6), (5,2), (6,4)]
graphs = (vertexList, edgeList)
def bfs(graph, start):
    vertexList, edgeList = graph
    visitedList = []
    queue = [start]
    adjacencyList = [[] for vertex in vertexList]

    # fill adjacencyList from graph
    for edge in edgeList:
        adjacencyList[edge[0]].append(edge[1])

    # bfs
    while queue:
        current = queue.pop()
        for neighbor in adjacencyList[current]:
            if not neighbor in visitedList:
                queue.insert(0,neighbor)
        visitedList.append(current)
    return visitedList

print(bfs(graphs, 0))
```

TASK: Consider the below graph;



Use BFS to find the shortest path between A and F. (Hint: the distance between any consecutive vertices is 1, i.e. distance between A and D is 2 ((A to B=1) + (B to D=1) = 2)