Lab-7: Uninformed Search (Depth-First-Search)

7.1 Objectives:

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

7.2 Depth-first search:

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the root in the case of a graph) and explores as far as possible along each branch before backtracking.

7.1.1 Pseudocode:

Input: A graph G and a vertex v of G
Output: All vertices reachable from v labelled as discovered

A recursive implementation of DFS:

```
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)
```

A non-recursive implementation of DFS:

```
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)
```

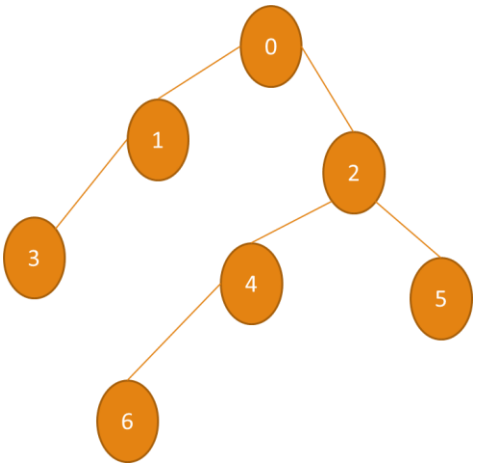
7.1.2 Iterative Deepening Depth-first search:

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.

7.1.3 Pseudocode:

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

s = new Stack();    ******* Change to use a stack

s.push(initial node);    ***** Push() stores a value in a stack

while ( s ≠ empty ) do
{
  x = s.pop();        ****** Pop() remove a value from the stack

  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 )
            s.push(y);       ***** Use push() !
  }
}
```

Example:

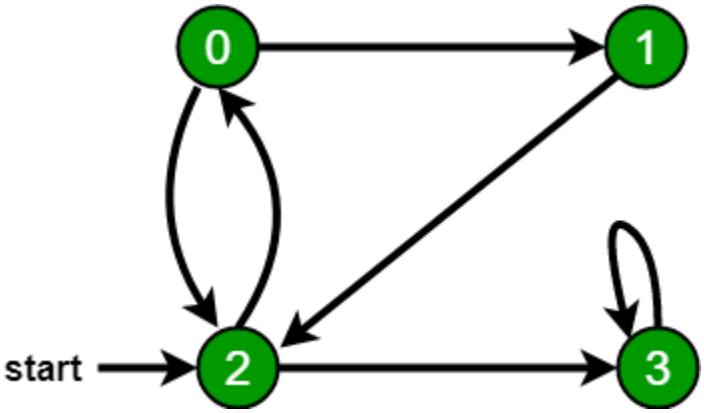


Code:

```
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 dfs(graph, start):
    vertexList, edgeList = graph
    visitedVertex = []
    stack = [start]
    adjacencyList = [[] for vertex in vertexList]

    for edge in edgeList:
        adjacencyList[edge[0]].append(edge[1])

    while stack:
        current = stack.pop()
        for neighbor in adjacencyList[current]:
            if not neighbor in visitedVertex:
                stack.append(neighbor)
        visitedVertex.append(current)
    return visitedVertex

print(dfs(graphs, 0))
```

TASK: Consider the below graph;



Using DFS, check if there is any path exists between any two nodes? Also the return the path.

e.g. If user two vertices i.e. 2 and 1; the program should return : Yes the paths exist, which are [2,1],[2,0,1].