JUNAID GIRKAR
60004190057
TE COMPS A4

ARTIFICIAL INTELLIGENCE
EXPERIMENT 2B

AIM: Implementation of DFID search algorithm.

THEORY:
A search algorithm which suffers neither the drawbacks of breadth-first nor depth-first search on trees is depth-first iterative-deepening (DFID). The algorithm works as follows: First, perform a depth-first search to depth one. Then, discarding the nodes generated in the first search, start over and do a depth-first search to level two. Next, start over again and do a depth-first search to depth three, etc., continuing this process until a goal state is reached. Since DFID expands all nodes at a given depth before expanding any nodes at a greater depth, it is guaranteed to find a shortest-length solution. Also, since at any given time it is performing a depth-first search, and never searches deeper than depth d, the space it uses is O(d).

The disadvantage of DFID is that it performs wasted computation prior to reaching the goal depth. In fact, at first glance it seems very inefficient. Below, however, we present an analysis of the running time of DFID that shows that this wasted computation does not affect the asymptotic growth of the run time for exponential tree searches. The intuitive reason is that almost all the work is done at the deepest level of the search. Unfortunately, DFID suffers the same drawback as depth-first search on arbitrary graphs, namely that it must explore all possible paths to a given depth.

CODE:

```
from collections import defaultdict

class Graph:

    def __init__(self,vertices):

        self.V = vertices # Number of vertices
        self.graph = defaultdict(list)
```

```
    def addEdge(self,u,v):
        self.graph[u].append(v)

    def DLS(self,src,target,maxDepth):

        if src == target : return True
        if maxDepth <= 0 : return False

        for i in self.graph[src]:
                if(self.DLS(i,target,maxDepth-1)):
                    return True
        return False

    def IDDFS(self,src, target, maxDepth):
        for i in range(maxDepth):
            if (self.DLS(src, target, i)):
                return True
        return False

# Create a graph
#        0
#      /   \
#     1      2
#    / \     | \
#   3   4    5   6

g = Graph (7)
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 3)
g.addEdge(1, 4)
g.addEdge(2, 5)
g.addEdge(2, 6)

for i in range(2):
    target = [5, 3]; maxDepth = [3,2]; src = 0

    if g.IDDFS(src, target[i], maxDepth[i]) == True:
        print ("Target "+str(target[i]) +" is reachable from
source within max depth of "+str(maxDepth[i]))
    else :
        print ("Target "+str(target[i]) +" is NOT reachable from
```

```
source within max depth of "+str(maxDepth[i]))
```

OUTPUT:

```
Target 5 is reachable from source within max depth of 3
Target 3 is NOT reachable from source within max depth of 2
```

CONCLUSION:

We learnt about the DFID search algorithm and implemented it in python.