Experiment no. 3

Aim: To implement Depth First Search (DFS) algorithm.

Requirement: Compatible version of python.

Theory:

Depth-first search (DFS) is an algorithm for searching a graph or tree data structure. The algorithm starts at the root (top) node of a tree and goes as far as it can down a given branch (path), then backtracks until it finds an unexplored path, and then explores it. The algorithm does this until the entire graph has been explored. Many problems in computer science can be thought of in terms of graphs. For example, analysing networks, mapping routes, scheduling, and finding spanning trees are graph problems. To analyse these problems, graph-search algorithms like depth-first search are useful. Depth-first searches are often used as subroutines in other more complex algorithms. For example, the matching algorithm, Hopcroft–Karp, uses a DFS as part of its algorithm to help to find a matching in a graph. DFS is also used in tree-traversal algorithms, also known as tree searches, which have applications in the traveling-salesman problem and the Ford-Fulkerson algorithm.

Algorithm:

Iterative Algorithm:

1. Initialize empty stack and visited set.

2. Pick up random a node from Graph or Tree and push it in to the stack.

3. Pop element of stack mark it as visited and print it.

4. Push all the non-visited neighbours of pop element in step 3 in to the stack.

5. Repeat steps 3 and 4 until stack is empty.

Recursive Algorithm:

1. Pass empty visited set and a random node as parameters to algorithm.

2. Mark the randomly selected node as visited and print it.

3. Call recursive DFS on all the non-visited neighbours of node with neighbour node and visited set as parameters.

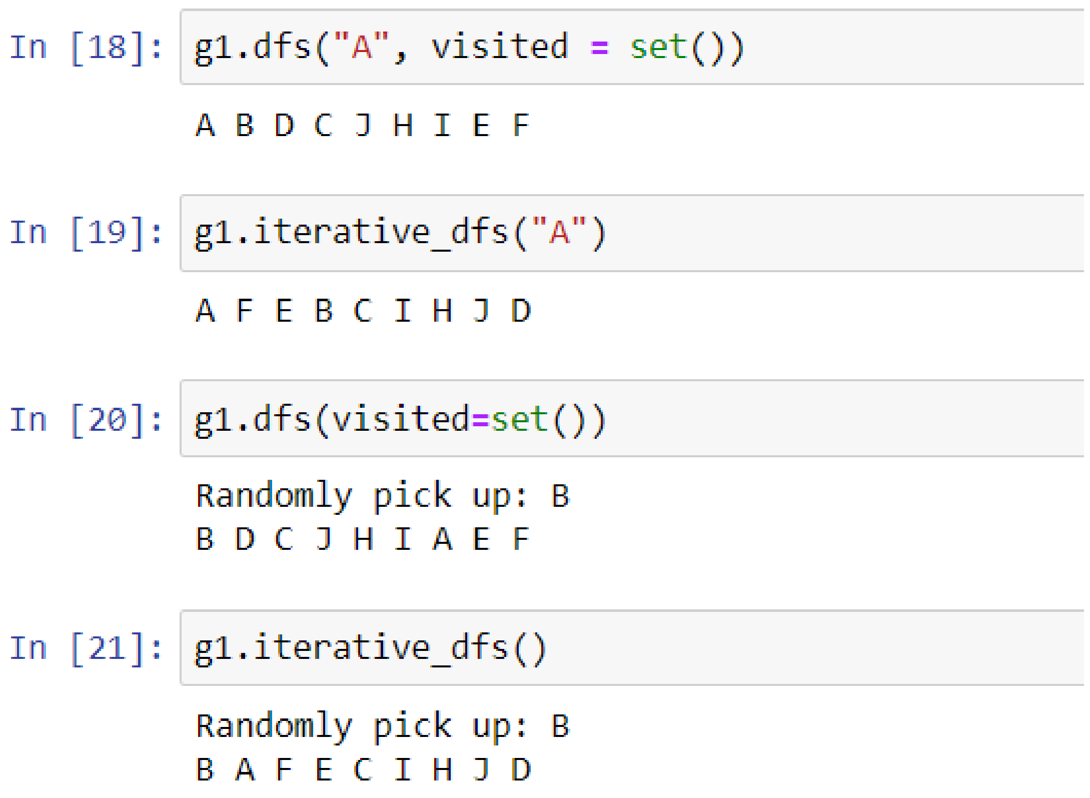
Since, DFS is not unique and output depends on which neighbour is selected first result of Iterative and Recursive algorithm may vary.

Implementation:

```
#For Creating Nodes
class Node:

    '''Creating a new node and adding all its neighbour'''
    def __init__(self,value,Node_map,neighbours = set()):
        self.value = value
        self.neighbours = set()
        self.add_neighbours(Node_map,neighbours)

    '''Adding Neighbours to a node(bidirectional edges) by checking neighbour node exist or
not,
        if doesn't creating one
    '''
    def add_neighbours(self,Node_map,neighbours=set()):
        for neighbour in neighbours:
            if neighbour not in Node_map:
                temp = Node(neighbour,Node_map)
                Node_map[neighbour] = temp
                temp.neighbours.add(self.value)
                self.neighbours.add(temp.value)

            else:
                self.neighbours.add(neighbour)
                Node_map[neighbour].neighbours.add(self.value)

class Graph:

    def __init__(self):
        self.Node_map = {} #Keeping accounting of nodes added till now with their value and
references
```

```
'''Simply create a Node if doesn't exist using Node class
   if Node exist only add edges
'''
def create_node(self,value,neighbours = set()):
   if value in self.Node_map:
      self.add_neighbours(value,neighbours)
   else:
      node = Node(value,self.Node_map,neighbours)
      self.Node_map[value] = node

def add_neighbours(self,value, neighbhours):
   #Before adding neigbhour to node first checking if node exist or node, if not raise error
   assert value in self.Node_map, "Given node doesn't exist"
   node = self.Node_map[value]
   node.add_neighbours(self.Node_map,neighbhours)

def dfs(self,value = None,visited = set()):
   if not value:
      value = list(self.Node_map.keys())[0]
      print(f"Randomly pick up: {value}")
   else:
      assert value in self.Node_map, "Node is not present"

   visited.add(value)
   print(value, end = " ")
   for neighbour in self.Node_map[value].neighbours:
      if neighbour not in visited:
         self.dfs(neighbour,visited)

def iterative_dfs(self, value = None):
   if not value:
      value = list(self.Node_map.keys())[0]
      print(f"Randomly pick up: {value}")
   else:
      assert value in self.Node_map, "Node is not present"
   stack = [value]
   visited = set()
   while(stack):
      value = stack.pop()
      visited.add(value)
      for neighbour in self.Node_map[value].neighbours:
         if neighbour not in visited:
            stack.append(neighbour)
            visited.add(neighbour)
      print(value, end=" ")
```

<u>Output</u>:



```
In [18]: g1.dfs("A", visited = set())

         A B D C J H I E F

In [19]: g1.iterative_dfs("A")

         A F E B C I H J D

In [20]: g1.dfs(visited=set())

         Randomly pick up: B
         B D C J H I A E F

In [21]: g1.iterative_dfs()

         Randomly pick up: B
         B A F E C I H J D
```

<u>Conclusion</u>: We have successfully implemented recursive and iterative DFS in python.