JUNAID GIRKAR
60004190057
TE COMPS A4

ARTIFICIAL INTELLIGENCE
EXPERIMENT 2A

AIM: Identify, analyze and implement BFS/DFS search algorithms to reach goal state

THEORY:

BFS:

BFS stands for Breadth First Search is a vertex based technique for finding a shortest path in a graph. It uses a Queue data structure which follows first in first out. In BFS, one vertex is selected at a time when it is visited and marked, then its adjacent vertices are visited and stored in the queue. It is slower than DFS. Here, siblings are visited before the children.

DFS:

DFS stands for Depth First Search is a edge based technique. It uses the Stack data structure, performs two stages, first visited vertices are pushed into stack and second if there are no vertices then visited vertices are popped. Here, children are visited before the siblings.

CODE:

bfs.py

```
# visited = []
queue = []

def bfs(visited, graph, node, goal_state):
  visited.append(node)
  queue.append(node)
```

```
while queue:          # Creating loop to visit each node
  m = queue.pop(0)
  print (m, end = " ")
  if(m==goal_state):
    print("GOAL", end= " ")
    break

  for neighbour in graph[m]:
    if neighbour not in visited:
      visited.append(neighbour)
      queue.append(neighbour)
```

```
def dfs(graph, source,path = []):

    if source not in path:

        path.append(source)

        if source not in graph:# leaf node, backtrack
            return path

        for neighbour in graph[source]:

            path = dfs(graph, neighbour, path)


    return path
```

Driver.py

```
from bfs import bfs
from dfs import dfs

graph = {
  '5' : ['3','7'],
  '3' : ['2', '4'],
  '7' : ['8'],
  '2' : [],
  '4' : ['8'],
  '8' : []
}
```

```
#        5
#      /   \
#    3      7
#   / \     \
#  2   4 ---- 8

lecture_graph = {
   '1' : ['4', '2'],
   '4' : ['3'],
   '3' : ['10', '9', '2'],
   '10' :[],
   '9' : [],
   '2' : ['5', '8'],
   '5' : ['2', '6','7'],
   '7' : ['5', '8'],
   '8' : ['2', '7'],
   '6' : []
}

#    1
#  /   \
# 4      2
#  \   / | \
#    3   5   8
#  / |   | \ |
# 10  9   6   7

# BFS
visited = []
print("Breadth-First Search traversal sequence")
bfs(visited, lecture_graph, '1', goal_state='8')
print("\n")

#DFS
print("Depth-First Search traversal sequence")
path = dfs(lecture_graph, "1")
goal_state = '9'
new_path=[]
for item in path:
 if(item==goal_state):
   break
 else:
   new_path.append(item)
new_path.append("GOAL")
print(" ".join(new_path))
```

OUTPUT:

```
Breadth-First Search traversal sequence
1 4 2 3 5 8 GOAL

Depth-First Search traversal sequence
1 4 3 10 GOAL
```

CONCLUSION:

We learnt about BFS and DFS searching techniques and implemented these algorithms in a python program. We also learnt about their time complexity which is O(V + E) when Adjacency List is used and O(V^2) when Adjacency Matrix is used, where V stands for vertices and E stands for edges.