```
#                          M NAdeem(FA20-BSE-035)
#
#

# Activity 1

class node:
   def __init__(self,state,parent,actions,totalcost):
    self.state = state
    self.parent = parent
    self.actions = actions
    self.totalcost = totalcost
    graph = {'A': node('A',None,['B','C','E'],None),
             'B': node('B',None,['A','D','E'],None),
            'C': node('C',None,['A','F','G'],None),
            'D': node('D',None,['B','E'],None),
            'E': node('E',None,['A','B','D'],None),
            'F': node('F',None,['C'],None),
            'G': node('G',None,['C'],None)
           }




# Activity 2

class node:
  def __init__(self,state,parent,actions,totalcost):
    self.state = state
    self.parent = parent
    self.actions = actions
    self.totalcost = totalcost

def actionSequence(graph,initialstate,goalstate):
  solution = [goalstate]
  currentparent = graph[goalstate].parent

  while currentparent != None:

    solution.append(currentparent)
    currentparent = graph[currentparent].parent

  solution.reverse()
  return solution

def dfs(initialstate,goalstate):

  graph = {'A': node('A',None,['B','C','E'],None),
           'B': node('B',None,['A','D','E'],None),
           'C': node('C',None,['A','F','G'],None),
           'D': node('D',None,['B','E'],None),
           'E': node('E',None,['A','B','D'],None),
           'F': node('F',None,['C'],None),
           'G': node('G',None,['C'],None)
          }
  frontier = [initialstate]
  explored = []
  currentChildren = 0
  while frontier:
    currentnode = frontier.pop(len(frontier)-1)
    explored.append(currentnode)
    for child in graph[currentnode].actions:
      if child not in frontier and child not in explored:
        graph[child].parent = currentnode
        if graph[child].state == goalstate:
          # print(explored)
          return actionSequence(graph,initialstate,goalstate)
        currentChildren=currentChildren+1
        frontier.append(child)
  if currentChildren == 0 :
    del explored[len(explored)-1]
solution = dfs('A','D')
print(solution)
```

```
# Activity 3

class node:
  def __init__(self,state,parent,actions,totalcost):
    self.state = state
    self.parent = parent
    self.actions = actions
    self.totalcost = totalcost

def actionSequence(graph,initialstate,goalstate):
  solution = [goalstate]
  currentparent = graph[goalstate].parent

  while currentparent != None:

    solution.append(currentparent)
    currentparent = graph[currentparent].parent

  solution.reverse()
  return solution

def bfs(initialstate,goalstate):

  graph = {'A': node('A',None,['B','C','E'],None),
           'B': node('B',None,['A','D','E'],None),
           'C': node('C',None,['A','F','G'],None),
           'D': node('D',None,['B','E'],None),
           'E': node('E',None,['A','B','D'],None),
           'F': node('F',None,['C'],None),
           'G': node('G',None,['C'],None)
          }
  frontier = [initialstate]
  explored = []
  while frontier:
    currentnode = frontier.pop(0)
    explored.append(currentnode)
    for child in graph[currentnode].actions:
      if child not in frontier and child not in explored:
        graph[child].parent = currentnode
        if graph[child].state == goalstate:
          return actionSequence(graph,initialstate,goalstate)
        frontier.append(child)
solution = bfs('D','C')
print(solution)
```

```
['D', 'B', 'A', 'C']
```

```
# activity 4
class node:
  def _init_(self,state,parent,actions,totalcost):
    self.state = state
    self.parent = parent
    self.actions = actions
    self.totalcost = totalcost

def actionSequence(graph,initialstate,goalstate):
  solution = [goalstate]
  currentparent = graph[goalstate].parent

  while currentparent != None:

    solution.append(currentparent)
    currentparent = graph[currentparent].parent

  solution.reverse()
  return solution

def bfs(initialstate,goalstate):
```

```
  graph = {'A': node('A',None,['B','C','E'],None),
           'B': node('B',None,['A','D','E'],None),
           'C': node('C',None,['A','F','G'],None),
           'D': node('D',None,['B','E'],None),
           'E': node('E',None,['A','B','D'],None),
           'F': node('F',None,['C'],None),
           'G': node('G',None,['C'],None)
          }
  frontier = [initialstate]
  explored = []
  while frontier:
    currentnode = frontier.pop(0)
    explored.append(currentnode)
    for child in graph[currentnode].actions:
      if child not in frontier and child not in explored:
        graph[child].parent = currentnode
        if graph[child].state == goalstate:
          return actionSequence(graph,initialstate,goalstate)
        frontier.append(child)
solution = bfs('D','C')
print(solution)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-35-1061805bde49> in <cell line: 42>()
     40           return actionSequence(graph,initialstate,goalstate)
     41         frontier.append(child)
---> 42 solution = bfs('D','C')
     43 print(solution)

<ipython-input-35-1061805bde49> in bfs(initialstate, goalstate)
     21 def bfs(initialstate,goalstate):
     22
---> 23   graph  = {'A': node('A',None,['B','C','E'],None),
     24             'B': node('B',None,['A','D','E'],None),
     25             'C': node('C',None,['A','F','G'],None),

TypeError: node() takes no arguments
```

SEARCH STACK OVERFLOW

```
# home activity

class Node:
  def _init_(self,state,parent,actions,totalcost):
    self.state=state
    self.parent=parent
    self.actions=actions
    self.totalcost=totalcost
graph = {'arad':Node('arad',None,['zernid','timisoara','sibiu'],None),
       'timisoara':Node('timisoara',None,['lugoj','arad'],None),
       'zernid':Node('zernid',None,['arad','oradea'],None),
       'sibiu':Node('sibiu',None,['arad','oradea','fagaras','rimnicu vilcea'],None),
       'lugoj':Node('lugoj',None,['mehadia','timisoara'],None),
       'oradea':Node('oradea',None,['zernid','sibiu'],None),
       'mehadia':Node('mehadia',None,['lugoj','drobeta'],None),
       'drobeta':Node('drobeta',None,['mehadia','craiova'],None),
       'craiova':Node('craiova',None,['drobeta','pitesti','rimnicu vilcea'],None),
       'rimnicu vilcea':Node('rimnicu vilcea',None,['craiova','pitesti','sibiu'],None),
       'pitesti':Node('pitesti',None,['craiova','rimnicu vilcea','bucharest'],None),
       'fagaras':Node('fagaras',None,['sibiu','bucharest'],None),
       'bucharest':Node('bucharest',None,['fagaras','pitesti','giurgiu','urziceni'],None),
       'giurgiu':Node('giurgiu',None,['bucharest'],None),
       'urziceni':Node('urziceni',None,['bucharest','hirsova','vaslui'],None),
       'hirsova':Node('hirsova',None,['urziceni','eforie'],None),
       'eforie':Node('eforie',None,['hirsova'],None),
       'vaslui':Node('vaslui',None,['urziceni','lasi'],None),
       'lasi':Node('lasi',None,['vaslui','neamt'],None),
       'neamt':Node('neamt',None,['lasi'],None),

}

def actionsequence(graph, initialstate,goalstate):
  solution=[goalstate]
  currentparent = graph[goalstate].parent
  while currentparent != None:
    solution.append(currentparent)
```

```
      currentparent = graph[currentparent].parent
    solution.reverse()
    print(solution)
    return solution

  def BFS():
    initialstate = 'arad'
    goalstate = 'bucharest'
    frontier = [initialstate]
    explored = []

    while len(frontier)!=0:
      currentNode = frontier.pop(0)
      explored.append(currentNode)
      for child in graph[currentNode].actions:
        if child not in frontier and child not in explored:
          graph[child].parent = currentNode
          if graph[child].state == goalstate:
            return actionsequence(graph,initialstate,goalstate)
          frontier.append(child)
  solution = BFS()
```

0s completed at 4:07 PM