Name : Zain Ul Abideen

Registeration: FA20-BSE-059

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 = 'A'
  goalstate = 'D'
  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:
          return actionSequence(graph,initialstate,goalstate)
        currentChildren=currentChildren+1
        frontier.append(child)
  if currentChildren == 0 :
    del explored[len(explored)-1]
solution = dfs()
```

```
print(solution)
```

```
['A', 'E', 'D']
```

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:
          print(explored)
          return actionSequence(graph,initialstate,goalstate)
        frontier.append(child)
solution = bfs('D','C')
print(f'solution: {solution}')
```

```
['D', 'B', 'E', 'A']
solution: ['D', 'B', 'A', 'C']
```

Activity 4:

```
import math

def findmin(frontier):
  minV=math.inf
  node=''
  for i in frontier:
    if minV>frontier[i][1]:
      minV=frontier[i][1]
      node = i
  return node

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

```
  while currentparent != None:
    solution.append(currentparent)
    currentparent = graph[currentparent].parent
  solution.reverse()
  return solution

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

def UCS(initialstate,goalstate):
  initialstate = 'C'
  goalstate = 'B'
  graph = {'A': node('A',None,[('B',6),('C',9),('E',1)],0),
           'B': node('B',None,[('A',6),('D',3),('E',4)],0),
           'C': node('C',None,[('A',9),('F',2),('G',3)],0),
           'D': node('D',None,[('B',3),('E',5),('F',7)],0),
           'E': node('E',None,[('A',1),('B',4),('D',5),('F',6)],0),
           'F': node('F',None,[('C',2),('E',6),('D',7)],0),
           'G': node('G',None,[('C',3)],0)
          }
  frontier = dict()
  frontier[initialstate] = (None,0)
  explored = []

  while frontier:
    currentnode = findmin(frontier)
    del frontier[currentnode]
    if graph[currentnode].state == goalstate:
      return actionSequence(graph,initialstate,goalstate)
    explored.append(currentnode)
    for child in graph[currentnode].actions:
      currentcost = child[1] + graph[currentnode].totalcost
      if child[0] not in frontier and child[0] not in explored:
        graph[child[0]].parent = currentnode
        graph[child[0]].totalcost = currentcost
        frontier[child[0]]=(graph[child[0]].parent,graph[child[0]].totalcost)
      elif child[0] in frontier:
        if frontier[child[0]][1] < currentcost:
          graph[child[0]].parent = frontier[child[0]][0]
          graph[child[0]].totalcost = frontier[child[0]][1]
        else:
          frontier[child[0]] = (currentnode,currentcost)
          graph[child[0]].parent = frontier[child[0]][0]
          graph[child[0]].totalcost = frontier[child[0]][1]



solution = UCS('C','B')
print(solution)
```

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

Home Activity:

```
import math

def findmin(frontier):
  minV=math.inf
  node=''
  for i in frontier:
    if minV>frontier[i][1]:
      minV=frontier[i][1]
      node = i
  return node

def actionSequence(graph,intialstate,goalstate):
  solution = [goalstate]
  currentparent=graph[goalstate].parent
  while currentparent != None:
```

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

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

def UCS(initialstate,goalstate):



  graph = {'Oradea': node('Oradea',None,[('Sibiu',151),('Zerlind',71)],0),
           'Zerlind': node('Zerlind',None,[('Arad',75),('Oradea',71)],0),
           'Arad': node('Arad',None,[('Sibiu',140),('Timisoara',118),('Zerlind',75)],0),
           'Timisoara': node('Timisoara',None,[('Arad',118),('Lugoj',111)],0),
           'Lugoj': node('Lugoj',None,[('Mehadia',70),('Timisoara',111)],0),
           'Mehadia': node('Mehadia',None,[('Drobeta',75),('Lugoj',70)],0),
           'Drobeta': node('Drobeta',None,[('Craiova',120),('Mehadia',75)],0),
           'Craiova': node('Craiova',None,[('Drobeta',120),('Pitesti',138),('Rimnicu Vicea',146)],0),
           'Rimnicu Vicea': node('Rimnicu Vicea',None,[('Craiova',146),('Pitesti',97), ('Sibiu',80)],0),
           'Sibiu': node('Sibiu',None,[('Arad',140),('Oradea',151), ('Fagaras',99),('Rimnicu Vicea',80)],0),
           'Fagaras': node('Fagaras',None,[('Bucharest',211),('Sibiu',99)],None),
           'Pitesti': node('Pitesti',None,[('Bucharest',101),('Craiova',138),('Rimnicu Vicea',97)],0),
           'Bucharest': node('Bucharest',None,[('Fagaras',211),('Pitesti',101),('Giurgiu',90),('Urziceni',85)],0),
           'Giurgiu': node('Giurgiu',None,[('Bucharest',90)],0),
           'Urziceni': node('Urziceni',None,[('Bucharest',85),('Hirsova',98),('Vaslui',142)],0),
           'Hirsova': node('Hirsova',None,[('Eforie',86),('Urziceni',98)],0),
           'Eforie': node('Eforie',None,[('Hirsova',86)],0),
           'Vaslui': node('Vaslui',None,[('Iasi',92),('Urziceni',142)],0),
           'Iasi': node('Iasi',None,[('Neamt',87),('Vaslui',92)],0),
           'Neamt': node('Neamt',None,[('Iasi',87)],0),
          }
  frontier = dict()
  frontier[initialstate] = (None,0)
  explored = []

  while frontier:
    currentnode = findmin(frontier)
    del frontier[currentnode]
    if graph[currentnode].state == goalstate:
      return actionSequence(graph,initialstate,goalstate)
    explored.append(currentnode)
    for child in graph[currentnode].actions:
      currentcost = child[1] + graph[currentnode].totalcost
      if child[0] not in frontier and child[0] not in explored:
        graph[child[0]].parent = currentnode
        graph[child[0]].totalcost = currentcost
        frontier[child[0]]=(graph[child[0]].parent,graph[child[0]].totalcost)
      elif child[0] in frontier:
        if frontier[child[0]][1] < currentcost:
          graph[child[0]].parent = frontier[child[0]][0]
          graph[child[0]].totalcost = frontier[child[0]][1]
        else:
          frontier[child[0]] = (currentnode,currentcost)
          graph[child[0]].parent = frontier[child[0]][0]
          graph[child[0]].totalcost = frontier[child[0]][1]



solution = UCS('Arad','Bucharest')
print(solution)
```

```
['Arad', 'Sibiu', 'Rimnicu Vicea', 'Pitesti', 'Bucharest']
```

✓ 0s completed at 3:57 PM