Name:

Ali Hassan

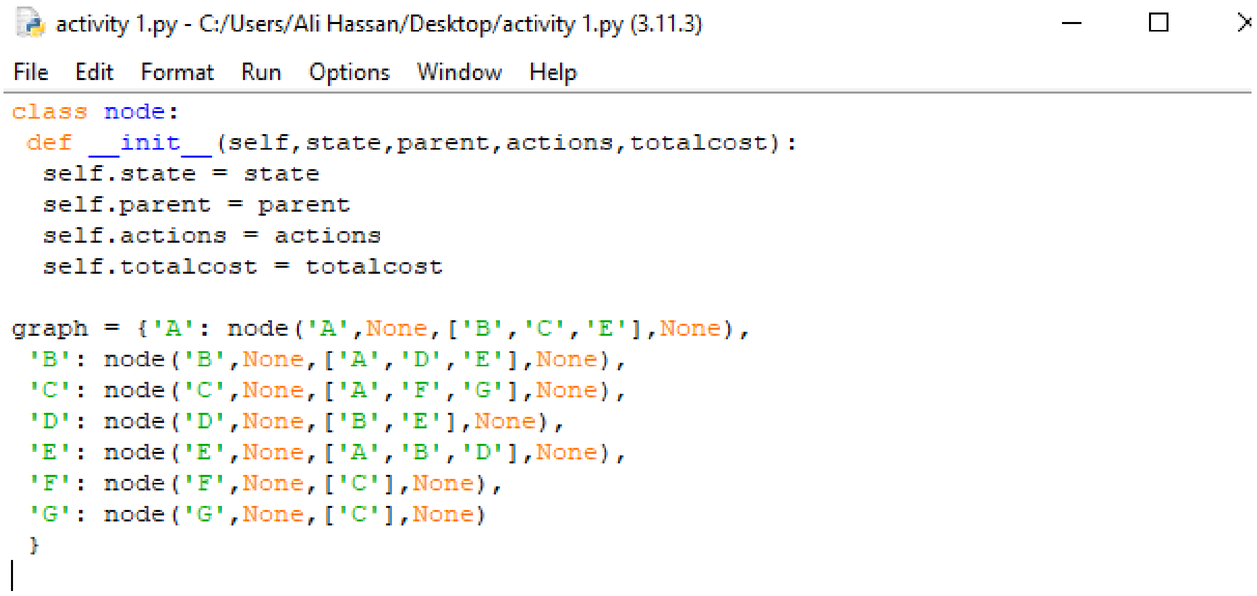
Reg no:

FA20-BSE-005

Task:

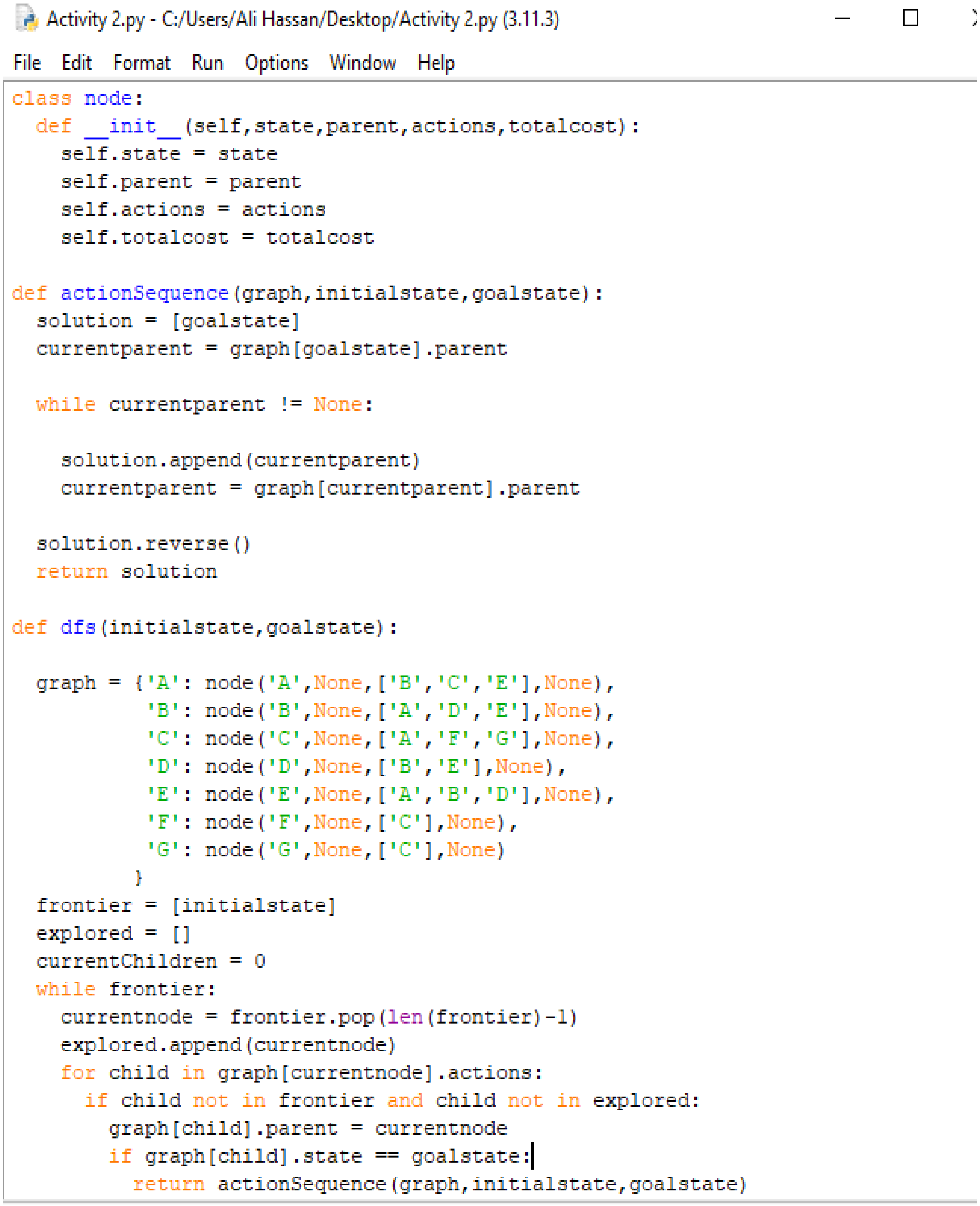
Lab Mid

Activity 1:

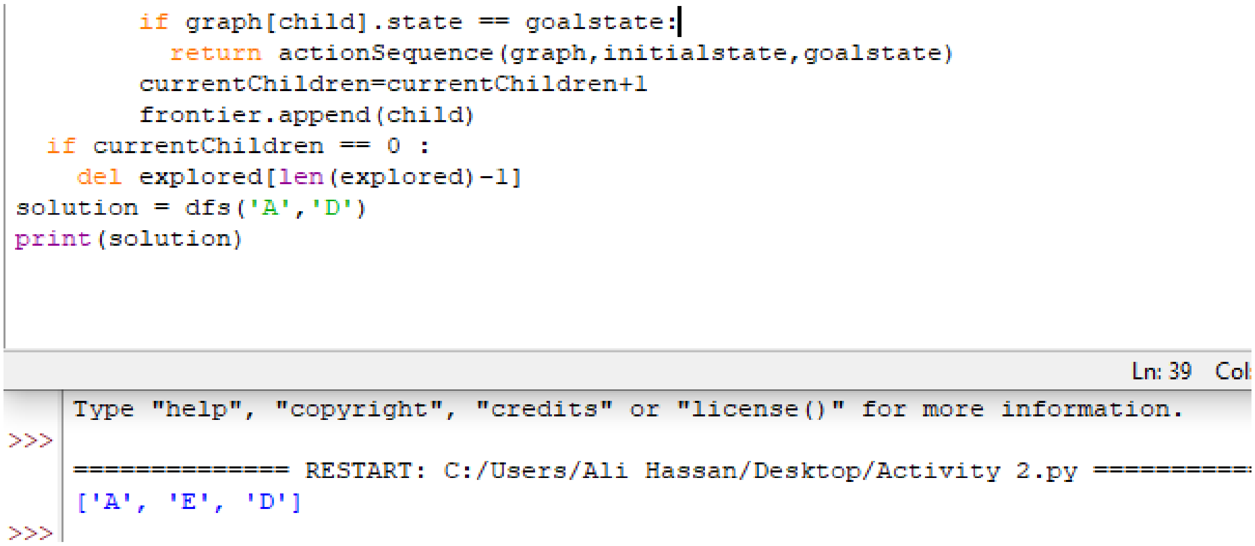

activity 1.py - C:/Users/Ali Hassan/Desktop/activity 1.py (3.11.3)
File
Edit
Format
Run
Options
Window
Help

```
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:


Activity 2.py - C:/Users/Ali Hassan/Desktop/Activity 2.py (3.11.3)
File
Edit
Format
Run
Options
Window
Help

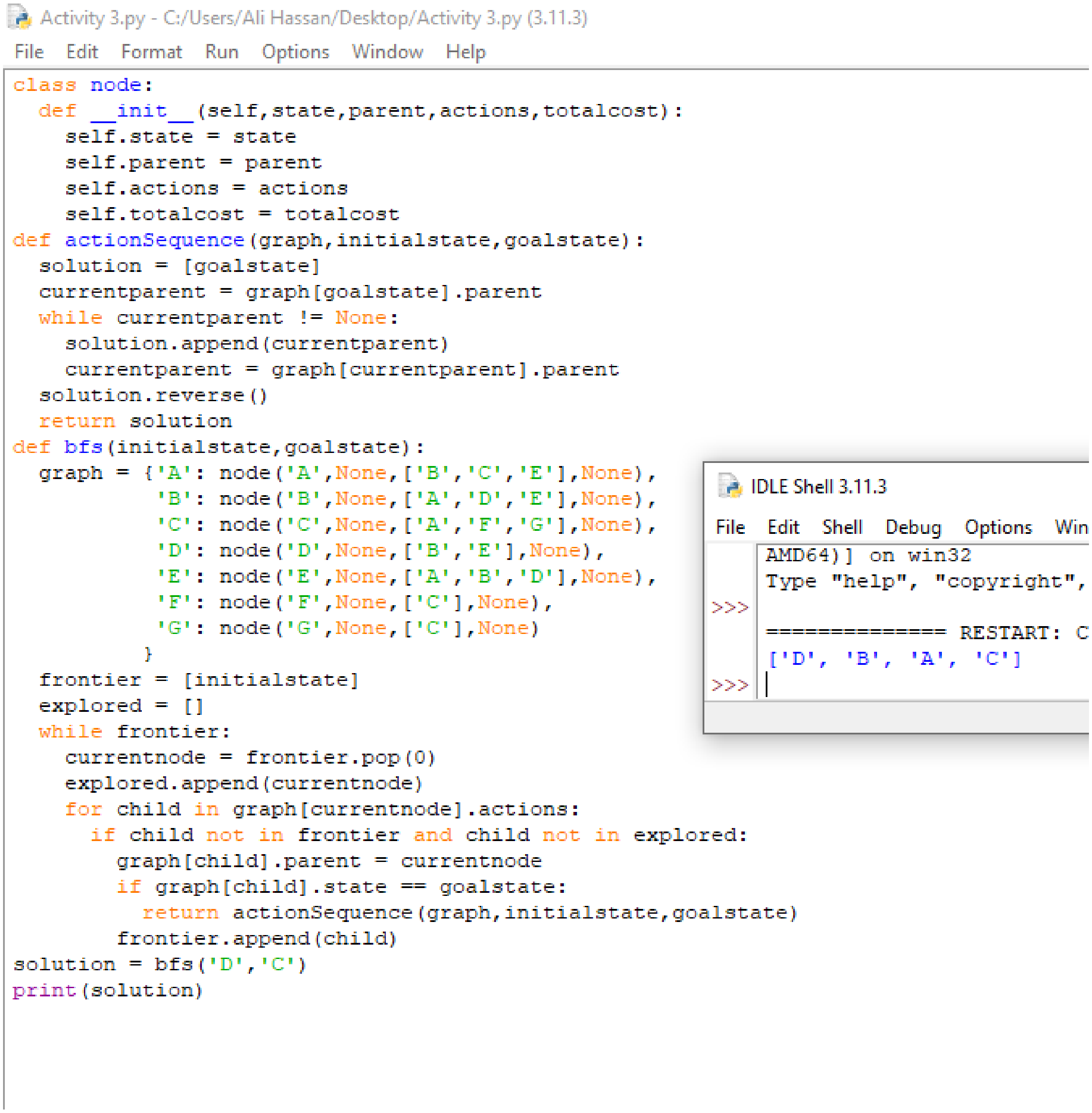
```
class node:
  def __init__(self,state,parent,actions,totalcost):
    self.state = state
    self.parent = parent
    self.actions = actions
    self.totalcost = totalcost

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:
          return actionSequence(graph,initialstate,goalstate)
```



```
        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('A','D')
print(solution)
```

```
    Type "help", "copyright", "credits" or "license()" for more information.
>>>
    ============== RESTART: C:/Users/Ali Hassan/Desktop/Activity 2.py ==========
    ['A', 'E', 'D']
>>>
```

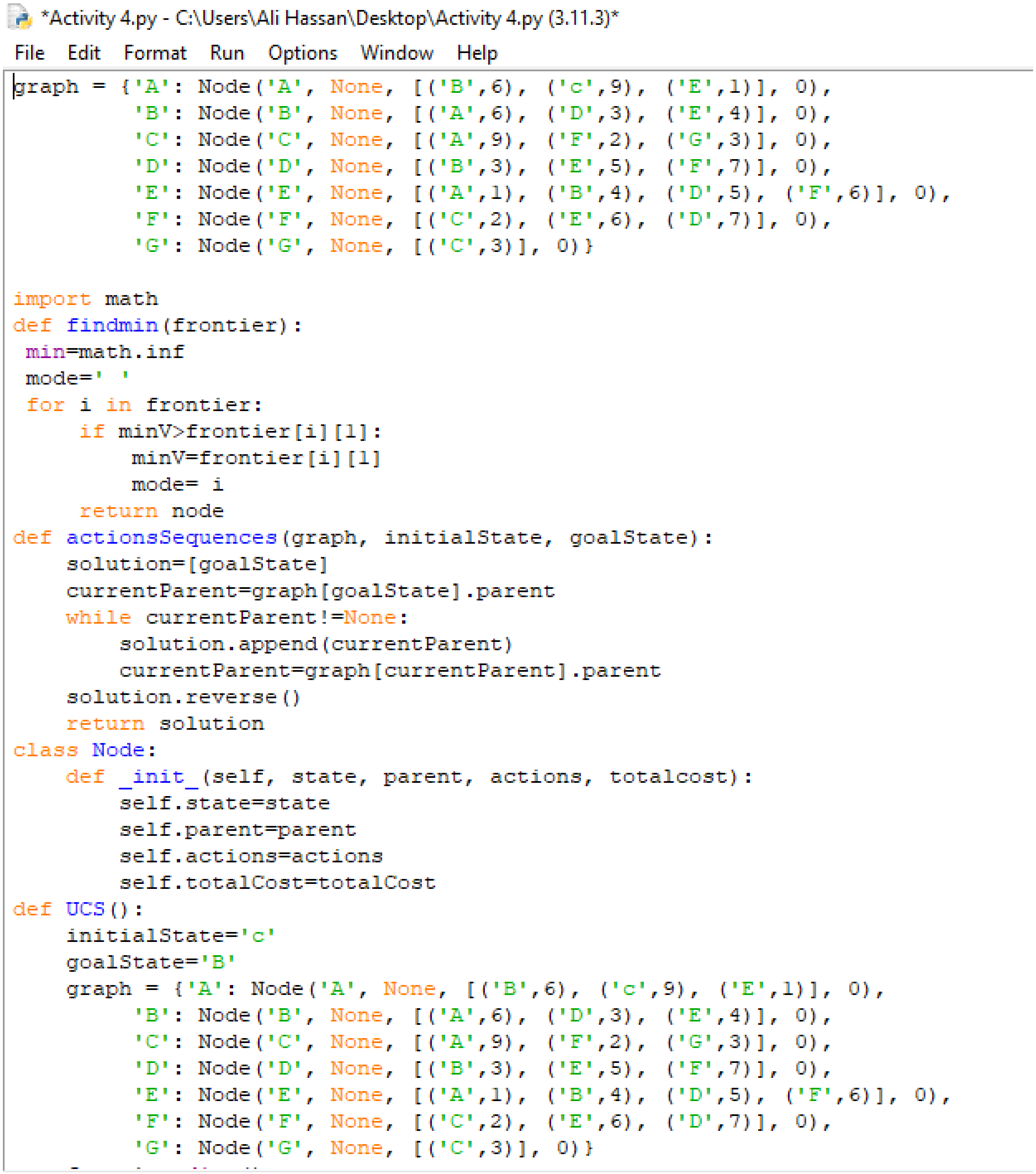
Activity 3:


Activity 3.py - C:/Users/Ali Hassan/Desktop/Activity 3.py (3.11.3)
File Edit Format Run Options Window Help

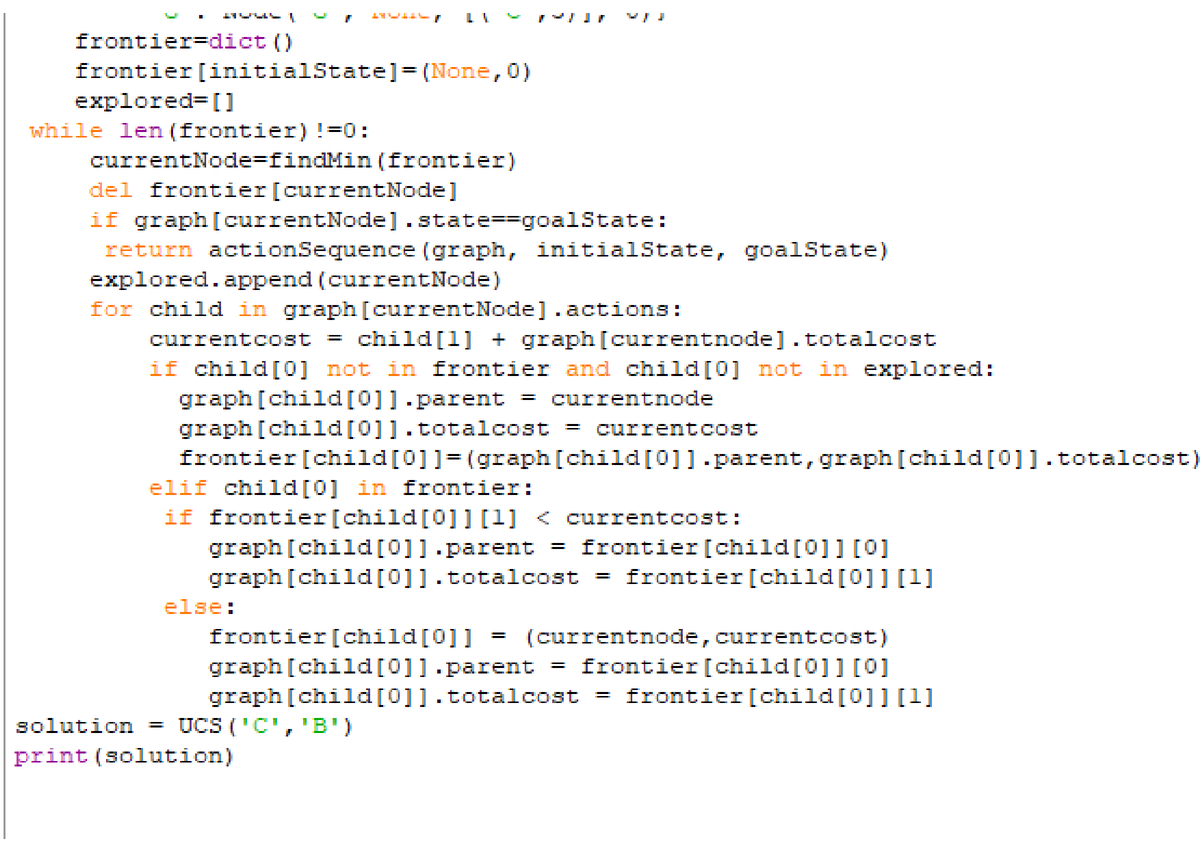
```
class node:
  def __init__(self,state,parent,actions,totalcost):
    self.state = state
    self.parent = parent
    self.actions = actions
    self.totalcost = totalcost
def actionSequence(graph,initialstate,goalstate):
  solution = [goalstate]
  currentparent = graph[goalstate].parent
  while currentparent != 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)
```

```
    AMD64)] on win32
    Type "help", "copyright",
>>>
    ============== RESTART: C
    ['D', 'B', 'A', 'C']
>>>
```

Activity 4:

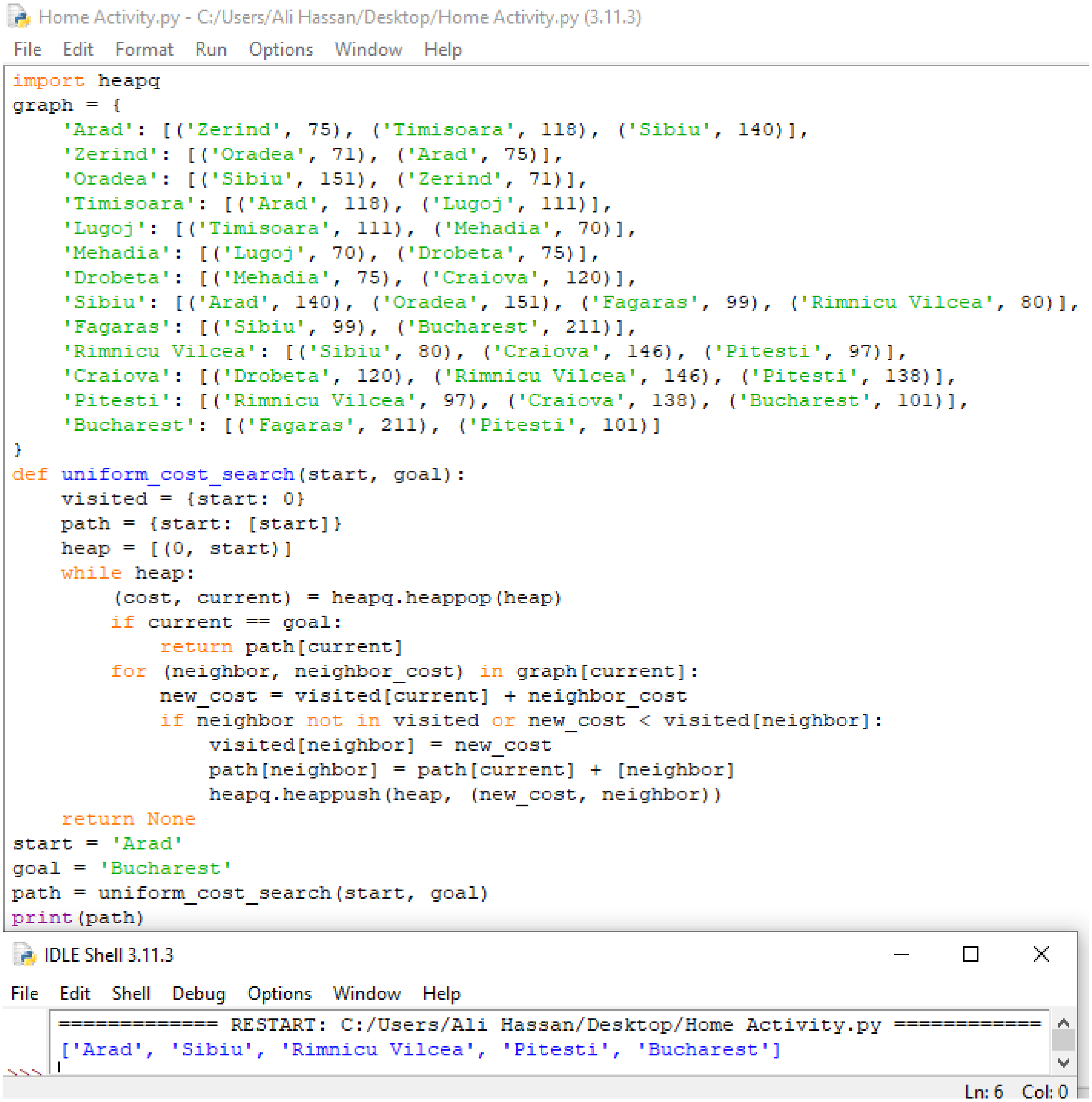

Activity 4.py - C:\Users\Ali Hassan\Desktop\Activity 4.py (3.11.3)
File
Edit
Format
Run
Options
Window
Help

```
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)}

import math
def findmin(frontier):
 min=math.inf
 mode=' '
 for i in frontier:
     if minV>frontier[i][1]:
         minV=frontier[i][1]
         mode= i
     return node
def actionsSequences(graph, initialState, 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='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)}
```



```
          'G': Node('G', None, [('C',5)], 0)}
    frontier=dict()
    frontier[initialState]=(None,0)
    explored=[]
 while len(frontier)!=0:
     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)
```

Home Activity:


Home Activity.py - C:/Users/Ali Hassan/Desktop/Home Activity.py (3.11.3)
File Edit Format Run Options Window Help
IDLE Shell 3.11.3
File Edit Shell Debug Options Window Help
Ln: 6 Col: 0

```python
import heapq
graph = {
    'Arad': [('Zerind', 75), ('Timisoara', 118), ('Sibiu', 140)],
    'Zerind': [('Oradea', 71), ('Arad', 75)],
    'Oradea': [('Sibiu', 151), ('Zerind', 71)],
    'Timisoara': [('Arad', 118), ('Lugoj', 111)],
    'Lugoj': [('Timisoara', 111), ('Mehadia', 70)],
    'Mehadia': [('Lugoj', 70), ('Drobeta', 75)],
    'Drobeta': [('Mehadia', 75), ('Craiova', 120)],
    'Sibiu': [('Arad', 140), ('Oradea', 151), ('Fagaras', 99), ('Rimnicu Vilcea', 80)],
    'Fagaras': [('Sibiu', 99), ('Bucharest', 211)],
    'Rimnicu Vilcea': [('Sibiu', 80), ('Craiova', 146), ('Pitesti', 97)],
    'Craiova': [('Drobeta', 120), ('Rimnicu Vilcea', 146), ('Pitesti', 138)],
    'Pitesti': [('Rimnicu Vilcea', 97), ('Craiova', 138), ('Bucharest', 101)],
    'Bucharest': [('Fagaras', 211), ('Pitesti', 101)]
}
def uniform_cost_search(start, goal):
    visited = {start: 0}
    path = {start: [start]}
    heap = [(0, start)]
    while heap:
        (cost, current) = heapq.heappop(heap)
        if current == goal:
            return path[current]
        for (neighbor, neighbor_cost) in graph[current]:
            new_cost = visited[current] + neighbor_cost
            if neighbor not in visited or new_cost < visited[neighbor]:
                visited[neighbor] = new_cost
                path[neighbor] = path[current] + [neighbor]
                heapq.heappush(heap, (new_cost, neighbor))
    return None
start = 'Arad'
goal = 'Bucharest'
path = uniform_cost_search(start, goal)
print(path)
```

```
============= RESTART: C:/Users/Ali Hassan/Desktop/Home Activity.py ============
['Arad', 'Sibiu', 'Rimnicu Vilcea', 'Pitesti', 'Bucharest']
>>>
```