**Artificial Intelligence Lab**

**Grapg Task**



**Ms. Saba Aslam**

Student Name: Abdul Salam

Roll No: 18F-0326 Section: CS(6B)

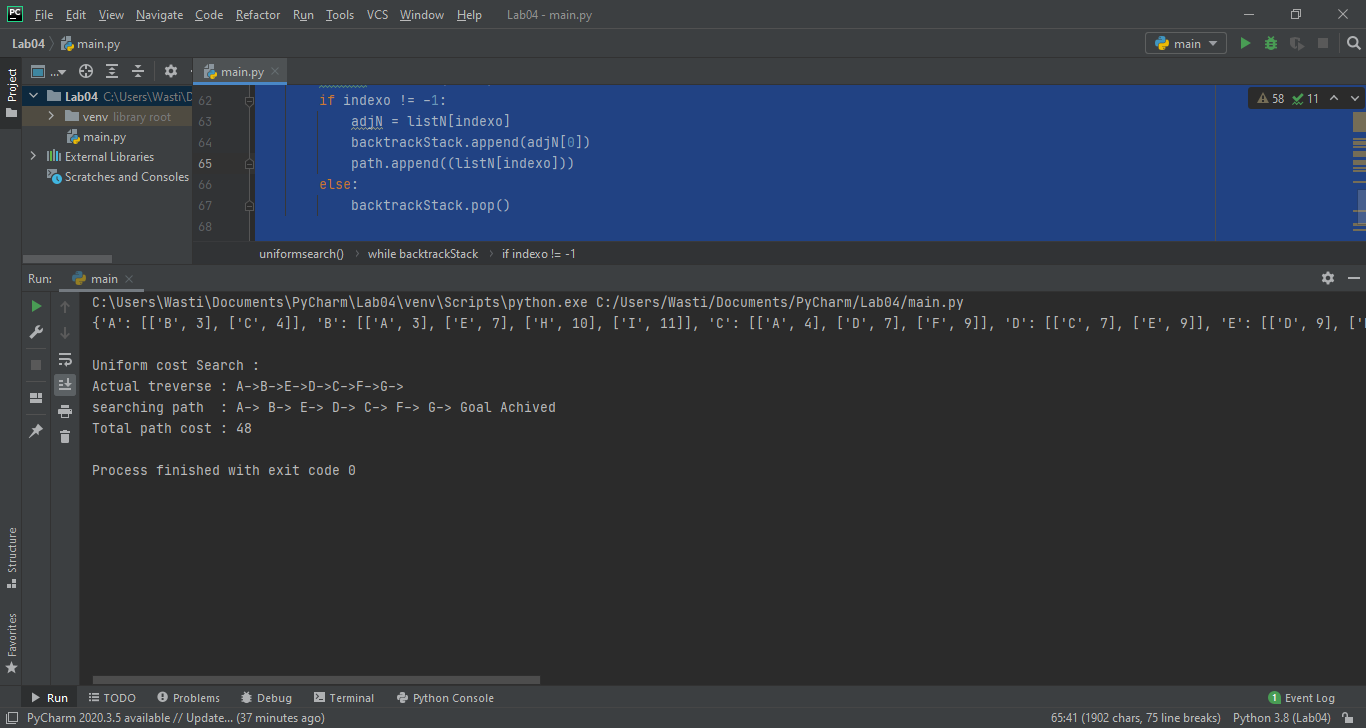
Date performed: 25/03/2021

# Task

# Code:

Uniform Cost search:

Graph = {  
 "A": [["B",3], ["C",4]],  
 "B": [["A",3], ["E",7], ["H",10], ["I",11]],  
 "C": [["A",4], ["D", 7], ["F",9]],  
 "D": [["C",7], ["E",9]],  
 "E": [["D",9], ["B",7], ["F",11], ["H",13]],  
 "F": [["C",9], ["E",11], ["G",13]],  
 "G": [["F",13], ["H",15], ["K",18]],  
 "H": [["B",10], ["E",13], ["G",15], ["K",19], ["L",20], ["J",18]],  
 "I": [["B",11], ["J",19]],  
 "J": [["H",18], ["I",19]],  
 "K": [["H",19], ["G",18]],  
 "L": [["H",20]]  
}  
  
  
  
print(Graph)  
  
def minlol(lists):  
 minValue = 999  
 index = 0  
 Mindex = -1  
 while index < len(lists):  
 lol = lists[index]  
 if minValue > lol[1] :  
 minValue = lol[1]  
 Mindex = index  
 index +=1  
 return Mindex  
  
def uniformsearch(Graph,startingN,goalN):  
 listN = []  
 backtrackStack = []  
 visited = []  
 path = []  
  
 path.append([startingN,0])  
 pathcost = 0  
 print("Actual treverse :", end=" ")  
  
 backtrackStack.append(startingN)  
 while backtrackStack:  
 listN.clear()  
 Node = backtrackStack[-1]  
 print(Node + "->", end="")  
 visited.append(Node)  
 if Node == goalN:  
  
 print("\nsearching path :", end=" ")  
 for pathv in path:  
 pathcost = pathv[1] + pathcost  
 print(str(pathv[0]) + "->", end=" ")  
 print("Goal Achived")  
 print("Total path cost : " + str(pathcost))  
 break  
  
 for conectedNode in Graph[Node]:  
 if conectedNode[0] not in visited and conectedNode[0] not in backtrackStack:  
 listN.append(conectedNode)  
 indexo = minlol(listN)  
 if indexo != -1:  
 adjN = listN[indexo]  
 backtrackStack.append(adjN[0])  
 path.append((listN[indexo]))  
 else:  
 backtrackStack.pop()  
  
  
  
print("\nUniform cost Search :")  
uniformsearch(Graph,"A","G")

Screenshot: 

# Code:

Bidirectional Search:

Graph = {  
 "A": [["B",3], ["C",4]],  
 "B": [["A",3], ["E",7], ["H",10], ["I",11]],  
 "C": [["A",4], ["D", 7], ["F",9]],  
 "D": [["C",7], ["E",9]],  
 "E": [["D",9], ["B",7], ["F",11], ["H",13]],  
 "F": [["C",9], ["E",11], ["G",13]],  
 "G": [["F",13], ["H",15], ["K",18]],  
 "H": [["B",10], ["E",13], ["G",15], ["K",19], ["L",20], ["J",18]],  
 "I": [["B",11], ["J",19]],  
 "J": [["H",18], ["I",19]],  
 "K": [["H",19], ["G",18]],  
 "L": [["H",20]]  
}  
  
  
  
print(Graph)  
  
  
def BFSbi(Graph, queue, visited,parent):  
 node = queue.pop(0)  
 visited.append(node)  
 for conected in Graph[node] :  
 if conected[0] not in queue and conected[0] not in visited:  
 queue.append(conected[0])  
 parent[conected[0]] = node  
 return  
  
def IntersectionCheck(Graph, fvisit,bvisit):  
 for i in fvisit:  
 for j in bvisit:  
 if i == j:  
 return i  
 return -1  
  
  
def Bidirectional(Graph, startingN, goalN):  
 farwardQueue = []  
 backwardQueue = []  
 farwardVisited = []  
 backwardVisited = []  
 fparent = {}  
 bparent = {}  
 path = []  
  
 pathFound = False  
  
 farwardQueue.append(startingN)  
 backwardQueue.append(goalN)  
  
  
 while (not pathFound) and (farwardQueue or backwardQueue):  
  
 BFSbi(Graph,farwardQueue,farwardVisited,fparent)  
  
 BFSbi(Graph,backwardQueue,backwardVisited,bparent)  
  
 intersect = IntersectionCheck(Graph,farwardVisited,backwardVisited)  
  
 if intersect != -1:  
 print(":: Path found At " + intersect + " ::")  
 v = intersect  
 while v != startingN:  
 path.append(v)  
 v = fparent[v]  
 path.append(startingN)  
  
 path.reverse()  
  
  
 v = intersect  
 while v != goalN:  
 if v not in path:  
 path.append(v)  
 v = bparent[v]  
 path.append(goalN)  
  
 for x in range(len(path)):  
 print(path[x] + "->",end="")  
 print("Goal Achived")  
 pathFound = True  
  
  
  
  
  
  
Bidirectional(Graph,"A","G")

# Screenshot: