```
class Graph:
 def __init__(self, graph, heuristicNodeList, startNode):

     self.graph = graph
     self.H=heuristicNodeList
     self.start=startNode
     self.parent={}
     self.status={}
     self.solutionGraph={}

 def applyAOStar(self):
     self.aoStar(self.start, False)
 def getNeighbors(self, v):
     return self.graph.get(v,'')

 def getStatus(self,v):
     return self.status.get(v,0)

 def setStatus(self,v, val):
     self.status[v]=val

 def getHeuristicNodeValue(self, n):
     return self.H.get(n,0)
 def setHeuristicNodeValue(self, n, value):
     self.H[n]=value


 def printSolution(self):
     print("FOR GRAPH SOLUTION, TRAVERSE THE GRAPH FROM THE START
NODE:",self.start)
     print("------------------------------------------------------------
--")
     print(self.solutionGraph)
     print("------------------------------------------------------------
--")

 def computeMinimumCostChildNodes(self, v):
     minimumCost=0
     costToChildNodeListDict={}
     costToChildNodeListDict[minimumCost]=[]
     flag=True
     for nodeInfoTupleList in self.getNeighbors(v):
         cost=0
         nodeList=[]
         for c, weight in nodeInfoTupleList:
             cost=cost+self.getHeuristicNodeValue(c)+weight
             nodeList.append(c)

         if flag==True:
             minimumCost=cost
             costToChildNodeListDict[minimumCost]=nodeList
             flag=False
         else:
             if minimumCost>cost:
                  minimumCost=cost
                  costToChildNodeListDict[minimumCost]=nodeList
```

```
        return minimumCost, costToChildNodeListDict[minimumCost]


    def aoStar(self, v, backTracking):

        print("HEURISTIC VALUES :", self.H)
        print("SOLUTION GRAPH :", self.solutionGraph)
        print("PROCESSING NODE :", v)
        print("-------------------------------------------------------------
------------------------------")

        if self.getStatus(v) >= 0:
            minimumCost, childNodeList =
self.computeMinimumCostChildNodes(v)
            self.setHeuristicNodeValue(v, minimumCost)
            self.setStatus(v,len(childNodeList))

            solved=True
            for childNode in childNodeList:
                self.parent[childNode]=v
                if self.getStatus(childNode)!=-1:
                    solved=solved & False

        if solved==True:
            self.setStatus(v,-1)
            self.solutionGraph[v]=childNodeList


        if v!=self.start:
            self.aoStar(self.parent[v], True)

        if backTracking==False:
            for childNode in childNodeList:
                self.setStatus(childNode,0)
                self.aoStar(childNode, False)



h1 = {'A': 1, 'B': 6, 'C': 2, 'D': 12, 'E': 2, 'F': 1, 'G': 5, 'H': 6,
'I': 7, 'J': 1}
graph1 = {
 'A': [[('B', 1), ('C', 1)], [('D', 1)]],
 'B': [[('G', 1)], [('H', 1)]],
 'C': [[('J', 1)]],
 'D': [[('E', 1), ('F', 1)]],
# 'G': [[('I', 1)]]
}
G1= Graph(graph1, h1, 'A')
G1.applyAOStar()
G1.printSolution()
print("HEURISTIC VALUES :", G1.H)
print("SOLUTION GRAPH :", G1.solutionGraph)
print('status:',G1.status)
print('parent:',G1.parent)
```

```
h2 = {'A': 1, 'B': 6, 'C': 12, 'D': 10, 'E': 4, 'F': 4, 'G': 5, 'H':
7}
graph2 = {
 'A': [[('B', 1), ('C', 1)], [('D', 1)]],
 'B': [[('G', 1)], [('H', 1)]],
 'D': [[('E', 1), ('F', 1)]]
}
#G2 = Graph(graph2, h2, 'A')
#G2.applyAOStar()
#G2.printSolution()
```