AI LAB 05

```
class TreeNode:
    def __init__(self, symbol , heurestic):
        self.symbol = symbol
        self.children = []
        self.goal = False
        self.parent = self
        self.heurestic = heuristic
    def addChild(self, node , cost):
        link = []
        link.append(node)
        link.append(cost)
        self.children.append(link)
        node.parent = self
    def setGoal(self):
        self.goal = True
nodeList = []
solution = []
numToChar = ["A","B","C","D" ,"E","F","G","I", "J"]
heurestic = [20.5,14.5,13,8,5,9,0,12.5,6.5]
for i in range(9):
    nodeList.append(TreeNode(numToChar[i],heurestic[i]))
#print(nodeList)
nodeList[0].addChild(nodeList[1],6.5)
nodeList[0].addChild(nodeList[2],9)
nodeList[1].addChild(nodeList[3],7.5)
```

```
nodeList[1].addChild(nodeList[7],5.5)
nodeList[2].addChild(nodeList[3],5)
nodeList[3].addChild(nodeList[4],4.5)
nodeList[4].addChild(nodeList[6],5)
nodeList[5].addChild(nodeList[8],3.5)
nodeList[7].addChild(nodeList[5],3.5)
nodeList[8].addChild(nodeList[6],6.5)
nodeList[6].setGoal()
start = nodeList[0]
currNode = start
def visitChildren(node,path):
    if node.goal==True:
        path.append(node.symbol)
        return path
    else :
        for c in node.children:
            path.append(c[0].children)
            visitChildren(c[0],path)
def findCost(node1, node2):
  #print(node1.symbol +" " + node2.symbol)
   for i in node1.children:
      if node2 == i[0]:
         return i[1]
def visitGoal(node,pathcost,path):
    if node.goal==True:
        path.append(node.symbol)
```

```
        return path
    while node.goal==False:
      path.append(node)
      childrenscosts = []
      for i in node.children:
        sum = pathcost+i[1]+i[0].heurestic
        childrenscosts.append(sum)
      #print(childrenscosts)
      mincostpath = childrenscosts.index(min(childrenscosts))
      pathcost = pathcost+node.children[mincostpath][1]
      #for i in path:
      #   print(i.symbol, end=" ")
      #print(f"f(n) : {childrenscosts[mincostpath]}")
      node = node.children[mincostpath][0]
    if node.goal==True:
       path.append(node)
       # print(path)
        return path,pathcost
# optimum path using Astar
path = []
p=0
print("Optimum path : ")
while currNode.goal==False:
    path.append(currNode)
    childrenscosts = []
```

```
    for i in currNode.children:

        sum = p+i[1]+i[0].heurestic

        childrenscosts.append(sum)


    #print(childrenscosts)
    mincostpath = childrenscosts.index(min(childrenscosts))

    p = p+currNode.children[mincostpath][1]

    for i in path:

        print(i.symbol, end=" ")

    print(f"f(n) : {childrenscosts[mincostpath]}")

    currNode = currNode.children[mincostpath][0]

path.append(currNode)



for i in path:

    print(i.symbol, end=" ")


print(f"Path cost : {p}")

print()

cnode = start
def Allparents(node,start):

   parents = []

   while(node.symbol != start.symbol):

      parents.append(node.parent)

      node = node.parent
   return parents

def rec(currNode,path,patht):

   if currNode.goal==True or len(currNode.children)==0:

       exit
```

```
else:
  # patht = []
   pathe,pathcost = visitGoal(currNode,path,patht)
   #print(pathe)
   parents = Allparents(currNode,start=nodeList[0])
   parents.reverse()
   pathe = parents+pathe
   pc =0
   sol = []
   solList = []
   for i in pathe:
     sol.append(i.symbol)
     #print(i.symbol ,end="\t")
   for j in range(len(pathe)-1):
     pc = pc + findCost(pathe[j],pathe[j+1])
   solList.append(sol)
   solList.append(pc)
   if solList not in  solution:
     solution.append(solList)
      #solution.append(solList)
   #print(f"Path cost = {pc}")

   for i in currNode.children:
     p =[]
     rec(i[0],i[1]+pathcost,p)
   if currNode.goal==True or len(currNode.children)==0:
```

```
      exit
r = []

rec(nodeList[0],0,r)

print(" all possible paths : ")

for i in solution:

  print("Path : ", end="\t")

  print(i[0], end="\t")

  print(" Cost : ", end="\t")

  print(i[1], end="\n")
```

Output