P1C-Lu-Lu

Lu Lu

4/11/2020

```
#Lu Lu - P1B

#1. Test given code with an input of 2, 3, 4, and 5

#The function PathEnumeration will input an integer called numNodes,\
    >1. The function will return the list of all Hamiltonian cycles\
   , on a complete graph, that start at a home node of 0.

def PathEnumeration(numNodes):

    SP=[[0]]
    LP=[]
    LPpathLengths=0

    while (LPpathLengths < numNodes):

        for i in range(1,len(SP)+1): #cycling through the short \
   paths of SP
            for j in range(2, numNodes+1):#Append to SP[i] the \
   numbers not in SP[i]
                doAppend=true
                for k in range(1, len(SP[i-1])+1): #see if j is in \
   SP[i]
                    if j==SP[i-1][k-1]+1: #or getCost is too high
                        doAppend=false
                        break

                if doAppend:
                    LP.append(SP[i-1]+[j-1])

        LPpathLengths=len(LP[0])

        SP=LP
        LP=[]
```

```
#     print 'SP =', SP
    return SP

PathEnumeration(2)
```

```
[[0, 1]]
```

```
PathEnumeration(3)
```

```
[[0, 1, 2], [0, 2, 1]]
```

```
PathEnumeration(4)
```

```
[[0, 1, 2, 3], [0, 1, 3, 2], [0, 2, 1, 3], [0, 2, 3, 1], [0, 3, 1, 2], [0, 3, 2, 1]]
```

```
PathEnumeration(5)
```

```
[[0, 1, 2, 3, 4], [0, 1, 2, 4, 3], [0, 1, 3, 2, 4], [0, 1, 3, 4, 2], [0, 1, 4, 2, 3], [0,
1, 4, 3, 2], [0, 2, 1, 3, 4], [0, 2, 1, 4, 3], [0, 2, 3, 1, 4], [0, 2, 3, 4, 1], [0, 2, 4,
1, 3], [0, 2, 4, 3, 1], [0, 3, 1, 2, 4], [0, 3, 1, 4, 2], [0, 3, 2, 1, 4], [0, 3, 2, 4,
1], [0, 3, 4, 1, 2], [0, 3, 4, 2, 1], [0, 4, 1, 2, 3], [0, 4, 1, 3, 2], [0, 4, 2, 1, 3],
[0, 4, 2, 3, 1], [0, 4, 3, 1, 2], [0, 4, 3, 2, 1]]
```

```
#2. Create a variable called “”weights.  This variable will be a \
   list of lists refer to P1A
weights\
   =[[0,8,7,2,1],[8,0,3,2,9],[7,3,0,10,8],[2,2,10,0,10],[1,9,8,10,0]]\


#test case for weights
#print weights[0][1]
```

```
#3.  Write a function called “”goHome.  This function will input a \
   list of lists (which will actually be the output of \
   thePathEnumeration function) and add the home node to each \
   sublist. It will then return the new list of lists.
def goHome(list):
    for i in list:
        i.append(0) #add 0 at the end of each list for returning to \
   point 0.
    return list

#test case for goHome
#list1=[[0],[0,1],[0,1,2]]
#print goHome(list1)
```

```
#4.  Write a function called “”getCost.  This function will input a \
   list of any length (such that corresponding weights are available\
   ).
def getCost(list):
    l=len(list)
    cost=0
    for i in range(0,l-1):
        cost+=weights[list[i]][list[i+1]] #add each weights between \
   two points into cost for storing the total cost of the path
    return cost

#test case for getCost
#list=[0,3,2,1,0]
#print getCost(list)
```

```
#5.  Write a function called “”getAllCosts.  This function will \
   input a list of lists (which will actually be the output of the \
   goHome function).  It will return a new list of all the costs of \
   each sublist.
def getAllCosts(list):
    allCost=[] #create a new list for storing the cost for each \
   given path by the order of the list
    for i in list:
        allCost.append(getCost(i)) #for each list of input, using \
   getCost to get the cost of the given path and put the results in \
   allCost list
    return allCost

#test case for getAllCosts
#list=[[0,1,2,3,0],[0,1,3,2,0],[0,2,1,3,0]]
#print getAllCosts(list)
```

```
#Lu Lu - P1C

#1  Write a function called bfTSP. This function will take an \
   integer 2,  3,  4,  or 5 as an input.  This integer will be \
   thenumber of nodes.  Note that an input of 2 means you are only \
   using nodes 0 and 1.  An input of 3 means that you areusing nodes\
    0, 1, and 2, etc.
def bfTSP(node):
    list=PathEnumeration(node)    #generate the all the lists
    homeList=goHome(list)         #get the total list of given nodes
    cost=getAllCosts(homeList)    #create list with all the cost in \
   it with all the path
    minCost=min(cost)             #find the path with minimum cost
    location=cost.index(minCost)  #find the index of the minimum \
   cost
    return homeList[location]     #using the index to find the path

#test case for bfTSP
#bfTSP(4)
```

```
#2  Test your function for each integer 2, 3, 4, and 5.
```

```
#Test for 2 nodes
print ”A solution to TSP with 2 nodes is ”, bfTSP(2), ”with a cost \
   of ”, getCost(bfTSP(2)), ”.”
```

```
A solution to TSP with 2 nodes is  [0, 1, 0] with a cost of  16 .
```

```
#Test for 3 nodes
print ”A solution to TSP with 3 nodes is ”, bfTSP(3), ”with a cost \
   of ”, getCost(bfTSP(3)), ”.”
```

```
A solution to TSP with 3 nodes is  [0, 1, 2, 0] with a cost of  18 .
```

```
#Test for 4 nodes
print ”A solution to TSP with 4 nodes is ”, bfTSP(4), ”with a cost \
   of ”, getCost(bfTSP(4)), ”.”
```

```
A solution to TSP with 4 nodes is  [0, 2, 1, 3, 0] with a cost of  14 .
```

```
#Test for 5 nodes
print ”A solution to TSP with 5 nodes is ”, bfTSP(5), ”with a cost \
   of ”, getCost(bfTSP(5)), ”.”
```

```
A solution to TSP with 5 nodes is  [0, 3, 1, 2, 4, 0] with a cost of  16 .
```