def dijkstra(nodes, distances):

# These are all the nodes which have not been visited yet

unvisited = {node: None for node in nodes}

# It will store the shortest distance from one node to another

visited = {}

current = 'B'

# It will store the predecessors of the nodes

currentDistance = 0

unvisited[current] = currentDistance

# Running the loop while all the nodes have been visited

while True:

# iterating through all the unvisited node

for neighbour, distance in distances[current].items():

# Iterating through the connected nodes of current\_node (for

# example, a is connected with b and c having values 10 and 3

# respectively) and the weight of the edges

if neighbour not in unvisited: continue

newDistance = currentDistance + distance

if unvisited[neighbour] is None or unvisited[neighbour] > newDistance:

unvisited[neighbour] = newDistance

# Till now the shortest distance between the source node and target node

# has been found. Set the current node as the target node

visited[current] = currentDistance

del unvisited[current]

if not unvisited: break

candidates = [node for node in unvisited.items() if node[1]]

current, currentDistance = sorted(candidates, key = lambda x: x[1])[0]

return visited

nodes = ('A', 'B', 'D', 'G', 'C', 'D', 'A')

distances = {

'B': {'A': 5, 'D': 1, 'G': 2},

'A': {'B': 5, 'D': 3, 'E': 12, 'F' :5},

'D': {'B': 1, 'G': 1, 'E': 1, 'A': 3},

'G': {'B': 2, 'D': 1, 'C': 2},

'C': {'G': 2, 'E': 1, 'F': 16},

'E': {'A': 12, 'D': 1, 'C': 1, 'F': 2},

'F': {'A': 5, 'E': 2, 'C': 16}}

print(dijkstra(nodes, distances))

# A Huffman Tree Node

class node:

def \_\_init\_\_(self, freq, symbol, left=None, right=None):

# frequency of symbol

self.freq = freq

# symbol name (character)

self.symbol = symbol

# node left of current node

self.left = left

# node right of current node

self.right = right

# tree direction (0/1)

self.huff = ''

# utility function to print huffman

# codes for all symbols in the newly

# created Huffman tree

def printNodes(node, val=''):

# huffman code for current node

newVal = val + str(node.huff)

# if node is not an edge node

# then traverse inside it

if(node.left):

printNodes(node.left, newVal)

if(node.right):

printNodes(node.right, newVal)

# if node is edge node then

# display its huffman code

if(not node.left and not node.right):

print(f"{node.symbol} -> {newVal}")

# characters for huffman tree

chars = ['a', 'b', 'c', 'd', 'e', 'f']

# frequency of characters

freq = [ 5, 9, 12, 13, 16, 45]

# list containing unused nodes

nodes = []

# converting characters and frequencies

# into huffman tree nodes

for x in range(len(chars)):

nodes.append(node(freq[x], chars[x]))

while len(nodes) > 1:

# sort all the nodes in ascending order

# based on theri frequency

nodes = sorted(nodes, key=lambda x: x.freq)

# pick 2 smallest nodes

left = nodes[0]

right = nodes[1]

# assign directional value to these nodes

left.huff = 0

right.huff = 1

# combine the 2 smallest nodes to create

# new node as their parent

newNode = node(left.freq+right.freq, left.symbol+right.symbol, left, right)

# remove the 2 nodes and add their

# parent as new node among others

nodes.remove(left)

nodes.remove(right)

nodes.append(newNode)

# Huffman Tree is ready!

printNodes(nodes[0])