```python
# -*- coding: utf-8 -*-
"""
Created on Sat Jan 25 22:51:09 2020

@author: cannellajs2
"""
################
# Question 1   #
# Merkel Trees #
################
import math
print("Merkle Tree Section")

# Given:
plainTextList = ["POT", "PIE", "CAT", "COW", "RAT", "OWL", "YAK"]

def countBitsSet(ordValue):
    count = 0
    while(ordValue):
        count += ordValue & 1
        ordValue >>= 1
    return count

# Take a string and loop over each character and sum number of bits set
def getStringHash(inString):
    totalBitsSet = 0
    for char in inString:
        # Convert each char to ordinal (integer) representation
        ordVal = ord(char)
        # Count the number of bits set in the integer
        charBitsSet = countBitsSet(ordVal)
        # Add number of bits to total
        totalBitsSet += charBitsSet
    return totalBitsSet%15

# Used to comute the hash of two hashes
def getIntHash(leftVal, rightVal):
    return (countBitsSet(leftVal) + countBitsSet(rightVal))%15

# Create a node class
class Node:
    def __init__(self, hashValue):
        self.left = None
        self.right = None
        self.parent = None
        self.hashVal = hashValue

# Determine number of leafs for "balanced tree"
def balanceTree(leafs):
    leafsRequired = 2**(math.ceil(math.log(len(leafs), 2)))
    dummyLeaf = Node(0);
    numLeafsToAdd = leafsRequired - len(leafs)
    for i in range(0, numLeafsToAdd):
        leafs.append(dummyLeaf)
    return leafs


# Loop over list elements, get their hashes, add to map and array
def createLeafs(wordList):
    leafList = []
    leafMap = {}
    for word in wordList:
        wordHash = getStringHash(word)
        leaf = Node(wordHash)
        leafMap[word] = leaf
        leafList.append(leaf)
    return leafList, leafMap

# Recursivly Build the tree starting with a list of leafs
def buildTree(nodes):
    parentArray = []
    if(len(nodes) != 1):
        for i in range(0, len(nodes), 2):# Move in increments of 2
            parentHash = getIntHash(nodes[i].hashVal, nodes[i+1].hashVal);
            parent = Node(parentHash)
            nodes[i].parent = parent
            nodes[i+1].parent = parent
            parent.left = nodes[i]
            parent.right = nodes[i+1]
            parentArray.append(parent)
        buildTree(parentArray)
```

```
# Create Leaf Nodes and Map Plaintext to them
leafList, leafMap = createLeafs(plainTextList)

# Make sure our Tree will be balanced
leafList = balanceTree(leafList)

# Build the Tree Recursively
buildTree(leafList)


def copathRecurse(node, traveledUp):
    visited[node] = True;
    if(traveledUp):
        if(node.left != None and node.left not in visited):
            #print("Searching Left")
            copath.append(node.hashVal)
            return copathRecurse(node.left, False)
        elif(node.right != None and node.right not in visited):
            #print("Searching Right")
            copath.append(node.hashVal)
            return copathRecurse(node.right, False)
    if(node.parent != None):
        #print("Searching Parent")
        return copathRecurse(node.parent, True)


# Create Function that Given an string will return the copath if it is in the tree
def populateCopath(word):
    if word in leafMap:
        leaf = leafMap[word]
        copath.append(leaf.hashVal)
        copathRecurse(leaf, False)
    else:
        return None

# Create function to evaluate co-path
def evaluateCoPath(path):
    result = path[0]
    for i in range(1, len(path)):
        result = getIntHash(result, path[i])
    return result

# Need either global or class varibles to do this effectively
copath = []# Track the hashes needed to verify results
visited = {}# Make sure we only vistit nodes once

# Perform the Actual Query
populateCopath("RAT")
print("CoPath of RAT: ", copath)
print("Evaluation of CoPath: ", evaluateCoPath(copath))
copath = []
populateCopath("DOG")
print("CoPath of DOG: ", copath, "\n\n")

##############
# Question 2 #
##############

################
# DES Section  #
################
from des import DesKey
print("DES Section")
plainText = b"This is fun!"

# Create DES Key
key = DesKey(b"12345678")                    # for DES
cipherText = key.encrypt(plainText, padding=True)

print("DES Cipher Text: ", cipherText)

decryptedMessage = key.decrypt(cipherText)

print("DES Decrypted Message: ", decryptedMessage)


###############
# RSA Section #
###############
import rsa
```

```
print("\n\nRSA Section")
# Generate Public/Private Key Pair
(pubkey, privkey) = rsa.newkeys(512)

print(pubkey, "\n")
print(privkey, "\n")

plainText = "This is fun!".encode('utf8');

# Encrypt with the public key
cipherText = rsa.encrypt(plainText, pubkey);
print("CipherText: ", cipherText)

decrypted = rsa.decrypt(cipherText, privkey)
print("\n RSA Decrypted Message: ", decrypted)
```