```
#Submission for Technical Interview Questions
#5/12/13


# Question 1 ----------------------------------------------------

# Efficiency:   The main part of this code that might take a while to run is the
#               for loop.  This function will run in O(n) time.
#
# Code Design:  Code was designed to run efficiently and is easy to understand.


def question1(s,t):
    s = str(s)
    t = str(t)

    if s and t is not None and s.isalpha() and t.isalpha():
        for c in t:
            if c not in s:
                return False
            return True
    return False

#Testing the function
question1("udacity","udazx")
question1("udacity","ad")
question1(None,"ad")
question1("udacity",None)
question1("udacity","")
question1("udacity",5)

# Question 2 ----------------------------------------------------

# Efficiency:   The main part of this code that might take a while to run is the
#               two for loops.  The first loop will not add a significant amount
#               of time to the function because it will only loop a few times.
#               This function will run in O(n) time because it has to search in
#               the second for loop to find the best combination
#
# Code Design:  Itertools was used to use the most efficient method for
#               generating the combinations.

def question2(a):
    #Modified from: https://stackoverflow.com/questions/12430604/longest-palindrome-subsequence

    #Code could be significantly shorter if you did not check for weird inputs
    from itertools import combinations
    if a is not None:
        a = str(a)
        if a.isalpha():
            for y in range(len(a),0,-1):
                for x in combinations(a,y):
                    if ''.join(x)==''.join(x)[::-1]:
                        print ''.join(x)
                        return None
    print "No Palidrome Found"
    return None

#Testing the function
question2("character")
question2("a")
question2("abc")
question2(None)
question2(5)
```

```
# Question 3 ----------------------------------------------------

# Efficiency:   The main part of this code that might take a while to run is the
#               two for loops.  This function will run in O(n*m) time.  Where n
#               is the number of nodes and m is the number of connections.
#
# Code Design:  The networkx libary was used to make the code easier to read and
#               more effient.
#

def question3(G):
    import networkx as nx

    Gnx = nx.Graph()
    for key , value in G.iteritems():
        Gnx.add_node(key)

        for val in value:
            Gnx.add_edge(key,val[0], key = val[1])

    mst = nx.minimum_spanning_tree(Gnx)
    return mst

#Testing the function
val = question3({'A':[('B',2)],'B':[('A',2),('C',5)],'C':[('B',5)]})
print(sorted(val.edges(data=True)))

val = question3({'A':[('B',3)],'B':[('A',3),('C',10)],'C':[('B',10)]})
print(sorted(val.edges(data=True)))

val = question3({'A':[('B',2)],'B':[('A',2)]})
print(sorted(val.edges(data=True)))

# Question 4 ----------------------------------------------------

# Efficiency:   The main part of this code that might take a while to run is the
#               5 for loops.  This function will run in O(2n*2m) time.  Where n
#               is the number of nodes and m is the number of connections.  Each
#               will have to run twice.  Once to generate the tree and the second
#               time to fill in the information about where in the tree that node
#               is.
#
# Code Design:  The networkx libary was used to make the code easier to read and
#               more effient.


def question4(T, r, n1, n2):
    import networkx as nx

    G = nx.Graph()
    G.add_node(r)

    n = len(T)
    m = len(T[0])

    for i in range(0,n):
        for j in range(0,m):
            if T[i][j] == 1:
                G.add_edge(i,j)

    for a in G.nodes():
        G.node[a]['Level']= None
    G.node[r]['Level']= 0
```

```
    m = len(G.nodes())
    for L in range (0,m):
        for n in G.nodes():
            if G.node[n]['Level'] == L:
                L1 = G[n]
                for a in L1.iteritems():
                    if G.node[a[0]]['Level'] == None:
                        G.node[a[0]]['Level'] = L+1

    path = nx.shortest_path(G,n1,n2)
    path.pop(0)
    path.pop(len(path)-1)

    LCALevel = G.node[n1]['Level']
    for n in path:
        val = G.node[n]['Level']
        if val < LCALevel:
            LCALevel = G.node[n]['Level']
            LCA = n

    return LCA

#Testing the function
val = question4([[0,1,0,0,0],[0,0,0,0,0],[0,0,0,0,0],[1,0,0,0,1],[0,0,0,0,0]],3,1,4)
#Correct Ans: 3
print val

val = question4([[0,0,0,0,0],[0,0,0,0,0],[0,1,0,1,1],[0,0,0,0,0],[1,0,0,0,0]],4,3,1)
#Correct Ans: 2
print val

val = question4([[0,0,0,0,1],[0,0,1,0,0],[0,1,0,1,1],[0,0,1,0,0],[1,0,1,0,0]],4,3,1)
#Correct Ans: 2
print val


# Question 5 ----------------------------------------------------

# Efficiency:   The main part of this code that might take a while to run is the
#               while loop.  This function will run in O(n) time.
#
# Code Design:  A list was used to recreate the linked list.

def question5(ll, m):

    FL = []
    FL.append(ll.data)
    NextItem = ll.get_next()

    while NextItem.get_next() is not None:
        FL.append(NextItem.data)
        NextItem = NextItem.get_next()

    FL.append(NextItem.data)

    if m < 0:
        return "Verify Inputs"
    m = len(FL)-1-m
    if m < 0:
        return "Verify Inputs"

    return FL[m]

#Source for Node and LinkedList Classes:
https://www.codefellows.org/blog/implementing-a-singly-linked-list-in-python
```

```
class Node(object):

    def __init__(self, data=None, next_node=None):
        self.data = data
        self.next_node = next_node

    def get_data(self):
        return self.data

    def get_next(self):
        return self.next_node

    def set_next(self, new_next):
        self.next_node = new_next

class LinkedList(object):

    def __init__(self, head=None):
        self.head = head

    def insert(self, data):
        new_node = Node(data)
        new_node.set_next(self.head)
        self.head = new_node


#Create a Linked List to input into question 5
LL = LinkedList()
LL.insert("E")
LL.insert("D")
LL.insert("C")
LL.insert("B")
LL.insert("A")

#Testing the function
question5(LL.head, 0)
question5(LL.head, 1)
question5(LL.head, 2)
question5(LL.head, 3)
question5(LL.head, 4)
question5(LL.head, 5)
question5(LL.head, -1)
```