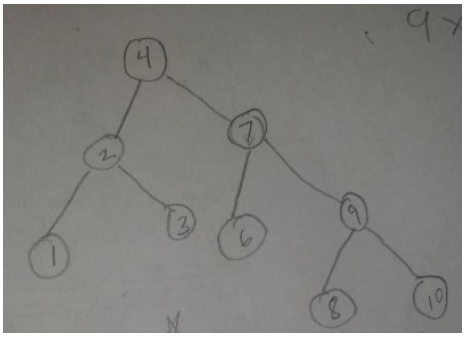
Jobin Thomas
Interview Questions – Udacity
Machine Learning Nanodegree

1. The goal for the problem, was to classify if one word is an anagram of the second word. In this case, s is the original word and t is the anagram. I iterate over the anagram and make sure it is contained in the original word. If it is not contained, then I set the return variable found as false and thereby completing the algorithm. Else if it is contained, I convert the anagram into a list and delete the iterating letter from that list. The loop is then repeated. If the found variable is not changed to False, the loop will complete and the function returns True.

2. The goal of this problem was to find the longest palindrome in a string parameter. I use a double for loop to iterate through all combination pieces of the string parameter. If a string piece equals the same string piece inversed, I append that to the possibleChoices array. I return the maximum length string in that array. If there are no strings in possibleChoices, meaning no palindromes in the original string parameter, the function returns 'none found'.

3. Next I find the minimum spanning tree using the prims algorithm. A minimum spanning tree is defined to reach every vertex, has a minimum total weight and there are no cycles in the connection of vertexes. Prims does this by choosing a random vertex and choosing the cheapest edge available that does not create cycles in the vertex connections. In my representation of the prims algorithm, first I created empty lists chosenEdges and chosenVertex. Next I append a random vertex from the keys of the dictionary which contains the 1st vertex as the key, and the 2nd vertex and edge as values for that key. Afterwards, I create a weightTable which sorts the dictionary, into a list. This makes it easier to work with. Next I make a sorted list of all vertexes so that the coming while loop can check and know when the algorithmic iterations can be stopped. Furerthermore inside this while loop, the algorithm iterates through each vertex in the weightTable checking if only one of two, of the vertex set, is contained in the chosenVertexes. This is xor. If conditions are met, the edge is appended and the new vertex is added in order to chosenVertex. The algorithm loop breaks the for loop and it is able to start again at the beginning of the while loop. When chosenVertex equals the completeVertexes, the prims algorithm is now complete and all edges that need to be selected have been selected. The chosen selection is converted back to dictionary form to meet function specifications and printed.

4. I'm still having trouble with this algorithm. The goal is to use two classes to represent the node and the tree. Although I'm having a hard time converting the adjacency matrix into the tree object. Are there any pointers that can help me lead to the solution? I attached an image of the tree I'm using for your convenience. After making the tree object, I would call my lca function which finds the least common denominator. This function returns a node if the value contained is between n1 and n2. This is also an indirect definition of the lowest common ancestor in a binary search tree. If it is not between n1 and n2, it moves onto a child node. More specifically, the right child node if the value is less than the minimum of n1 and n2. Likewise, the left child node if the value is more than the maximum of n1 and n2.



5. Similar to number 4, I solved this algorithm by creating a node class and then an linkedlist class which contains the previous nodes. Although the user never has to touch the node class, since the linkedlist class calls it every time. I made an array that the function makeLinked creates using the LinkedList class. Next the function getMth finds the total size of the linked list and then subtracts m from it, into the variable newIndex. To conclude the algorithm, the algorithm iterates through the linked list newIndex times and returns the data in that node.

Code

```
import json
import random
from IPython.display import Image, display

def question1(s, t):
    for val in range(len(t)):
        found = True
        if t[val] not in s:
            found= False
            break
        else:
            count = 0
            ss = list(s)
            for l in ss:
                count+=1
                if l==t[val]:
                    del ss[count]
                    break
    return found

def question2(a):
    possibleChoices = []
    for j in range(0,len(a)+1):
        for i in range(0,len(a)+1):
            if len(str(a[j:i]))>1 and str(a[j:i]) == str(a[j:i])[::-1]:
                possibleChoices.append(str(a[j:i]))
    return max(possibleChoices, key=len) if possibleChoices else 'none found'

def question3(G):
        chosenVertex = []
        chosenEdges = []
        s32 = G
        chosenVertex.append(random.choice(s32.keys()))
        #chosenVertex.append('A')
        #print chosenVertex
        weightTable = []
        for key in s32:
            for val in s32[key]:
                v = sorted([key, val[0]])
                if [val[1],v[0],v[1]] not in weightTable:
                    weightTable.append([val[1],v[0],v[1]])
        weightTable =  sorted(weightTable)
        #print weightTable
        completeVertexes = sorted(list(set([key for key in s32 for val in s32[key] ] +
                                [val[0] for key in s32 for val in s32[key]])))
```

```
        while(chosenVertex != completeVertexes):
            for val in weightTable:
                if val[1] in chosenVertex or val[2] in chosenVertex:
                    if val[1] not in chosenVertex or val[2] not in chosenVertex:
                        if val not in chosenEdges:
                            chosenEdges.append(val)
                        chosenVertex.append(val[2] if val[2] not in chosenVertex else val[1])
                        chosenVertex = sorted(chosenVertex)
                        break

        #print 'chosenEdges = {} \nchosenVertex = {}'.format(chosenEdges, chosenVertex)
        #print chosenVertex
        #print chosenEdges
        answer = {}
        for val in chosenVertex:
            for edge in chosenEdges:
                if val in edge:
                    answer[val] = (edge[2] if val!= edge[2] else edge[1],edge[0])
        return answer


def question4(T, r, n1, n2):
        class Node:
            def __init__(self,val):
                self.value = val
                self.leftChild = None
                self.rightChild = None
            def insert(self, data):
                if self.value == data:
                    return False
                elif self.value > data:
                    if self.leftChild:
                        return self.leftChild.insert(data)
                    else:
                        self.leftChild = Node(data)
                        return True
                else:
                    if self.rightChild:
                        return self.rightChild.insert(data)
                    else:
                        self.rightChild = Node(data)
                        return True

        class Tree:
            def __init__(self):
                self.root = None
            def insert(self, data):
                if self.root:
                    return self.root.insert(data)
```

```
            else:
                self.root = Node(data)
                return True
        def table(self, r):
            self.insert(r)
            for i,val in enumerate(tree[r-1]):
                if val == 1:
                    return self.table(i)

    def lca(root, n1, n2):
            if( root > max(n1, n2)):
                    return lca(root.left, n1, n2)
            elif(root < min(n1, n2)):
                    return lca(root.right, n1, n2)
            else:
                    return root

def question5(ll,m):
    class Node(object):

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

        def get_data(self):
            return self.data

        def get_i(self):
            return self.c

        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
            self.c =0
        def insert(self, data):
            self.c+=1
            new_node = Node(data,self.c)
            new_node.set_next(self.head)
            self.head = new_node
        def size(self):
            current = self.head
            count = 0
```

```
            while current:
                count += 1
                current = current.get_next()
            return count
        def search(self, data):
            current = self.head
            found = False
            while current and found is False:
                if current.get_i() == data:
                    found = True
                else:
                    current = current.get_next()
            if current is None:
                raise ValueError("Data not in list")
            return current
        def printer(self):
            current = self.head
            while current:
                print current.get_i()
                current = current.get_next()
        def makeLinked(self, array):
            for val in array:
                self.insert(val)
        def getMth(self, m):
            m = m-1
            newIndex = self.size()-m
            found = self.search(newIndex)
            return found.get_data()

    linked = LinkedList()
    linked.makeLinked(ll)
    return linked.getMth(m)

def main():
    ## Functions
    #print question1(s,t)
    #print question2(word)
    #print question3(s32)
    #question4(tree, 4,1,12)
    #print question5(ll,m) # ll (first node is 3), m mth num from end



## Variables
# for # 1
s = 'udacityy'
t = 'yyu'
# for # 2
#word = 'kayakhello'
```

```
word = 'kayak'
# for # 4
s32 = {'A':[('E',5),('H',6),('F',1),('B',8),('B',8)],'B':[('F',6),('C',4)],'C':[('F',2),('G',7)],'G':[('F',9)],'F':
[('H',5)],'H':[('E',3)]}
tree = [[0,1,0,0,0,0,0,0,0,0],
                    [1,0,1,1,0,0,0,0,0,0],
                    [0,1,0,0,0,0,0,0,0,0],
                    [0,1,0,0,0,0,1,0,0,0],
                    [0,0,0,0,0,0,0,0,0,0],
                    [0,0,0,0,0,0,1,0,0,0],
                    [0,0,0,1,0,1,0,0,1,0],
                    [0,0,0,0,0,0,0,0,1,0],
                    [0,0,0,0,0,0,1,1,0,1],
                    [0,0,0,0,0,0,0,0,1,0]]
# for # 5
ll = [3,5,4,2,1]; m = 2
if __name__ == '__main__':
        main()
```