210CT Coursework 2

Contents

Philip Bell - 5690404

Task 1 -Graph Building

Pseudo Code

```
#This code uses adjacency list to created a unweighted graph
listOfNodes ← []

CLASS NODES()
    #This function adds a node to the list of nodes
    ADDNODE(nodeToAdd)
        listOfNodes.append(nodeToAdd)
        FOR i IN nodeToAdd
            listOfNodes[i].append(length of listOfNodes)

    #This function adds a edge (connections to a new node) to a node
    ADDEDGE(firstNode,secondNode)
        listOfNodes[firstNode].append(secondNode)
        listOfNodes[secondNode].append(firstNode)
```

Python Code

For the structure of the graph I decided to create an adjacency list by using a list of lists. I used this approach because it would make the addNode and addEdge functions pretty easy to create since manipulating multidimensional list isn't too difficult. For the adding of a new node the code appends new node to the list of lists, then it updates the old nodes with the connection to the new node. For the adding of a new edge, it adds the edge to the second node to the first node and then the edge to the first node to the second node. The output method simply outputs the list of nodes in a readably format.

```
# nodeList is a adjacency list of nodes and edges that connect the nodes
nodeList = [[1,2,3],[0,2,4],[0,1,3,4],[0,2,4],[1,2,3,5],[4]]
"""
the nodes class is what operates on the nodes
"""
class nodes():
    """
    addNode will add new node the the end of nodeList, it will then update the nodes that the new node is connected to with the new edges. It must be
    """
    def addNode(self,nodeConnections):
        nodeList.append(nodeConnections)# appends the connections(edges) of the new node to the node list
        for self.i in nodeConnections:# loops though the new connections
            nodeList[self.i].append(len(nodeList)-1) #for the connections in the new node it adds the corresoponding connections to the old nodes
    """
    addEdge will add ned connections/edges to alreay existing nodes, you hand it the 2 nodes that you would like a new edge between and it will update them
    """
    def addEdge(self,nodeFrom,nodeTo):
        nodeList[nodeFrom].append(nodeTo) # append the new connection in 1st node
        nodeList[nodeTo].append(nodeFrom) # append the new connection in 2nd node
    """
    This simply outputs the nodeList so I can check that its has been updated correctly
    """
    def output(self):
        self.a = 0 # assigns a to zero
        for self.i in nodeList: # loops through list of node
            print "Node",self.a,":", self.i # print the list number and the value in the list
            self.a += 1 # increments a
        #return nodeList
b = nodes()
b.addNode([0,5])
b.addEdge(3,1)
b.output()
```

Adding Weighted vertices

A weighted graph is a graph where each of edge has a numerical value assigned to it (Weisstein, 2015), this allows for shortest distance/rout algorithms to be performed on the graph. An example of this is Dijkstra's algorithm (Auckland University, 1998).

To add weighted graph to my code I would change the list of lists to a list of lists of dictionaries, this way I would have a value for which nodes the edge is to and weight of that edge. Here is what the list of lists used in my code would be changed to act as I am suggesting:

```
weightedGraph =[
    [
        {"node":1,"edgeWeight":3},
        {"node":2,"edgeWeight":4},
        {"node":3,"edgeWeight":5}
    ],
    [
        {"node":0,"edgeWeight":3},
        {"node":2,"edgeWeight":6},
        {"node":4,"edgeWeight":2}
    ],
    [
        {"node":0,"edgeWeight":4},
        {"node":1,"edgeWeight":6},
        {"node":3,"edgeWeight":4},
        {"node":4,"edgeWeight":7}
    ],
    [
        {"node":0,"edgeWeight":5},
        {"node":2,"edgeWeight":4},
        {"node":4,"edgeWeight":8}
        ],
    [
        {"node":1,"edgeWeight":2},
        {"node":2,"edgeWeight":7},
        {"node":3,"edgeWeight":8},
        {"node":5,"edgeWeight":1}
    ],
    [
        {"node":4,"edgeWeight":1}
    ]
]
```

Changing it to be like this would add the ability to find out what node the edge is connected to and what the weight of the node is.

Task 2 - Graph Traversal

For the graph traversal I decided to use a diction of sets instead of a list of lists (like in the graph building task). I decided to do this because the sets worked better with stack that is used with depth first search. (Python Sets, 2015) At first I tried to do it with the list of lists but I found it to be over complicated.

Python Code

```
1.  nodeDict = {0: set([1,3]),1: set([0]),2: set([5]),3: set([0,4]),4: set([3,5]),5: set([2,4])}
2.
3.
4.  class nodes():
5.      """
6.      This method must be handed a graph in the form of a dictionary of sets and which node to start
     on.
7.      """
8.      def graphTraverse(self,nodeDict,startNode):
9.          self.path = [] #empty list for the path to he added to
10.         self.visitedNodes, self.stack = set(), [startNode]#created the stack
11.         while self.stack: #loops though the stack until it is empty (meaning all nodes are visited
    )
12.             self.node = self.stack.pop()#makes the current node the top of the stack
13.             if self.node not in self.visitedNodes: #if that node has not already been visited
14.                 self.visitedNodes.add(self.node) #adds the current node to the visited nodes
15.                 self.stack.extend(nodeDict[self.node] - self.visitedNodes)  #adds the edges in the
     node to the stack minus the alead visited nodes
16.                 self.path.append(self.node)  # add the current node to the path
17.         return self.path,self.visitedNodes # returns the path and the nodes visited
18.
19.
20. (i,j) = nodes().graphTraverse(nodeDict,4) # calls the method with the parameters graph and startin
    g node. makes it equal to i and j so the 2 items returned by the method can be accessed individual
    ly
21. print "Path used:",", ".join(str(a) for a in i) # prints out the path and removes square brackets

22. print "Nodes visited:",", ".join(str(a) for a in j) # prints out the visited nodes and removes the
     square brackets
```

Task 3 - Binary Tree Search

For the bin used a while loop to loop if the tree handed to the method was not empty, then if it finds the value it is looking for it returns true, if it is smaller or larger it moves left or right. If it goes off the end of the tree the value will become none so “None” will be returned, using the same return that would have returned “None” if the tree was empty.

Python Code

```
1.  class BinTreeNode(object):
2.
3.      def __init__(self, value):
4.          self.value=value
5.          self.left=None
6.          self.right=None
7.
8.  def tree_insert( tree, item):
9.      if tree==None:
10.         tree=BinTreeNode(item)
11.     else:
12.         if(item < tree.value):
13.             if(tree.left==None):
14.                 tree.left=BinTreeNode(item)
15.             else:
16.                 tree_insert(tree.left,item)
17.         else:
18.             if(tree.right==None):
19.                 tree.right=BinTreeNode(item)
20.             else:
21.                 tree_insert(tree.right,item)
22.     return tree
23.
24. def bin_tree_find(tree,target):
25.     while tree != None: # will loop while the is something in the tree
26.         if tree.value == target: # if the value selected is the target then value is returned
27.             return tree.value
28.         elif tree.value > target: # if the value selected is more than the target then the value s
    elected is moved to the left
29.             tree = tree.left
30.         else: #if anything else (aiming for selected value less than the target) the selected valu
    e is moved right
31.             tree = tree.right
32.     return None # returns none if cant be found or tree is empty
33.
34. def postorder(tree):
35.     if(tree.left!=None):
36.         postorder(tree.left)
37.     if(tree.right!=None):
38.         postorder(tree.right)
39.     print tree.value
40.
41. def in_order(tree):
42.     if(tree.left!=None):
43.         in_order(tree.left)
44.     print tree.value
45.     if(tree.right!=None):
46.         in_order(tree.right)
47.
48. if __name__ == '__main__':
49.     t=tree_insert(None,6);
50.     tree_insert(t,10)
51.     tree_insert(t,5)
52.     tree_insert(t,2)
53.     tree_insert(t,3)
54.     tree_insert(t,4)
55.     tree_insert(t,11)
56.     in_order(t)
57.     print (bin_tree_find(t,4))
```

Task 4 - Binary Tree Node Delete

This is a node delete algorithm, I have implemented the delete function into the code provided, the code works by handing the delete function the node that need to be deleted, if it is the head then head gets changed to the next node, if it is the tail it is changed to the previous node. If it is a node in the middle, it is changed to the previous or next node depending on where it is on the list and if the previous or next node are not empty.

Python Code

```
1.  class Node(object):
2.      def __init__(self, value):
3.          self.value=value
4.          self.next=None
5.          self.prev=None
6.
7.  class List(object):#inserts the given parameters to the list
8.      def __init__(self):
9.          self.head=None
10.         self.tail=None
11.     def insert(self,n,x):
12.         if n !=None:
13.             x.next=n.next
14.             n.next=x
15.             x.prev=n
16.             if x.next!=None:
17.                 x.next.prev=x
18.         if self.head==None:
19.             self.head=self.tail=x
20.             x.prev=x.next=None
21.         elif self.tail==n:
22.             self.tail=x
23.     def delete(self,n): #this will delete a Node
24.         if n.prev != None: #if previous n isnt empty then the previous next n will become the next
    n
25.             n.prev.next = n.next
26.         else: #if not head will become the next n
27.             self.head = n.next
28.         if n.next != None: #if next n is empty then the next previous n will become the previous n

29.             n.next.prev = n.prev
30.         else: #if not tail will become the previous n
31.             self.tail = n.prev
32.     def display(self): #outputs the list to the screen
33.         values=[]
34.         n=self.head
35.         while n!=None:
36.             values.append(str(n.value))
37.             n=n.next
38.         print "List: ","".join(values)
39.
40. if __name__ == '__main__':
41.     l=List()
42.     l.insert(None, Node(4))
43.     l.insert(l.head,Node(6))
44.     l.insert(l.head,Node(8))
45.     l.display()
46.     l.delete(l.head)
47.     l.display()
```

Task 5 - Signal Processor

For signal processor I used the math library toallowed me use sin so in the genSeries method no matter what values are inputted by the incrementor the values returned will be between -1 and 1 (Python Math, 2015). The list that series of numbers and integer that defines the length of the series are created in the constructor, this allows them to used by the other methods. The series generation method (genSereis) works by having a value that is incremented by 10 for defined length of the series, each loop the incremented value is put through sin to create a value between -1 and 1. That value is then appended to a list that is returned by the method. The value change (valchange) method gets a series from the genSeries method then uses a lambda function make all the values positive so the wave produced is all in the positive rather than going in and out of negative and positive like the series normally does. Using a lambda function allows me to create a small function that will only be used in that one place without having to create a full defined function (Sahu, 2014). The adjusted values are then appended to a list that is returned by the method. The display method works by being handed a list, then if the value is positive it fills half the scale with white space until the mid-point then it will but the number of hashes that the value is. If it is negative it fill with white space up to where the hashes should start, where hashes start is calculated by adding the value in the list (which will be negative) to the scale.

Python Code

```
1.  import math
2.  class numStore():
3.
4.      """
5.      self iniateing method that creates the series list and the length of the series list, it must
    be handed the lengh of the series as a integer
6.      """
7.      def __init__(self,seriesLength):
8.          self.series = [] #list the other methods use for the series of numbers
9.          self.seriesLength = seriesLength #defines the length of the series from the parameter that
     is handed to the method
10.
11.     """
12.     This method generates a series of numbers between -
    1 and 1 using sin. It will the the length that is set in the constructor
13.     """
14.     def genSeries(self):
15.         self.incrementor = 10 #screates the value that will be incremented
16.         for self.i in range (0,self.seriesLength): #loops for the required length of the series of
     numbers
17.             self.series.append(math.sin(math.radians(self.incrementor)))# appends a sin value betw
    een -1 and 1
18.             self.incrementor += 10 # increments the value the sin uses by 10
19.         return self.series #returns the series
20.
21.     """
22.     This method will display a list(what the other functuons return) is a readable format (lines o
    f hashes)
23.     """
24.     def display(self,listToOutput):
25.         self.scale = 30 # this sets the scale for the graph
26.         for self.i in listToOutput:#loops for the length of the list handed to the method
27.             self.j = int(self.i*self.scale) # changes the values to ints from a floats to integers

28.             self.str = "" #creates the string to fill with spaces or #
29.             if self.j > 0: # if the value is value is positive
30.                 for self.a in range (0,self.scale): #for 30 spaces a white space will be created
31.                     self.str = self.str + " "
32.                 for self.a in range(0,self.j): # for the length of the value hashs will be printed

33.                     self.str = self.str + "#"
34.             else: # if its is a negative number
35.                 for self.a in range (0,self.j+self.scale): #for the empty space print white space

36.                     self.str = self.str + " "
37.                 for self.a in range(0,-
    self.j): # for the remaining space between white space created and the mid point, print hashs
```

```
                    self.str = self.str + "#"
            print self.str #print the line of hashs produced
    """
    This method get a series of numbers from genSeries() and then alters them using a lambda so they are all in the positive region
    """
    def valChanger(self):
        self.listToReturn =[] #creates the list that will be returned
        self.values = numStore(self.seriesLength).genSeries() #gets a series of values from the genSeries
        f = lambda x : x+1 if x<=1 else x #lambda that makes adds 1 to any value less than 1 which will create a wave only in positive
        for self.i in self.series: #for the length of the series
            self.listToReturn.append(f(self.i)) # appends the value that is changed by the lambda to the list that is returned
        return self.listToReturn #returns the list

a = numStore(100) #makes the class equal to a, send 10 as the series length
a.display(a.genSeries()) # displays what is returned by genSeries()
print " "
a.display(a.valChanger())# displays what is returned by valChanger()
```

Bibliography

Auckland University, 1998. *Data Structures and Algorithms 10.2 Dijkstra's Algorithm.* [Online]
Available at: https://www.cs.auckland.ac.nz/software/AlgAnim/dijkstra.html
[Accessed 8 12 2015].

Python Math, 2015. *Python Documentation 9.2. math.* [Online]
Available at: https://docs.python.org/2/library/math.html
[Accessed 10 12 2015].

Python Random, 2015. *Python Documentation Random 9.6.* [Online]
Available at: https://docs.python.org/2/library/random.html
[Accessed 7 12 2015].

Python Sets, 2015. *Python Documentation Sets 8.7.* [Online]
Available at: https://docs.python.org/2/library/sets.html
[Accessed 5 12 2015].

Sahu, J., 2014. *Lambda Expressions ine Java 8: Why and How to Use Them.* [Online]
Available at: http://www.nagarro.com/de/de/blog/post/26/Lambda-Expressions-in-Java-8-Why-and-How-to-Use-Them
[Accessed 9 12 2015].

Weisstein, E. W., 2015. *Wolfram Mathworld Weighted Graph.* [Online]
Available at: http://mathworld.wolfram.com/WeightedGraph.html
[Accessed 5 12 2015].