210ct coursework

Student Name : DEVESH PATNI

SID : 6741031

I can confirm that all work is submitted is my own.

Github link : https://github.com/Devesh0/210-CT-COURSEWORK

**Basic**

**Question 1 – RANDOM SHUFLE**

**BIG O notation : O(n)**

import random

def shuffle(array):

num = len(array)

for i in range(num):

j = random.randint(i, num-1)

array[j], array[i] = array[i], array[j] #swap elements

return array

print (shuffle ([5,3,8,6,1,9,2,7]))

**Question 2. – TRAILING ZEROES**

**BIG O notation : O(n)**

def trailing\_zeroes(n):

count = 0

while n >= 5:

count += int(n/5)

n /=5

return count

print (trailing\_zeroes (int(input ())))

**Question 3 – HIGHEST PERFECT SQUARE**

**PSEUDOCODE** : -

PERFECT\_SQUARE(number)

n <- int (number)

for x <- 1 to n

IF (n is less then x mult x) THEN

n <- (x-1) mult (x-1) // get to the lower number then x

return n

ELIF n <- x mult x

return n

**CODE** :-

def perfect\_square(number):

n = int(number)

for x in range (1, n):

if n < x \* x:

n = (x-1) \* (x-1)

return(n)

elif n == x \* x:

return(n)

print (perfect\_square (input ()))

**Question 4. BIG – O**

**For Task 1 : BIG O is O(n)**

**For Task 2 : BIG O is O(n)**

**Question 5. – Matrix**

**PSEUDOCODE**

M1 <- Matrix1 of order n

M2 <- Matrix2 of order n

ADD\_MATRIX (M1, M2)

for i <- 1 to length[M1] // Length of rows of M1

for j <- 1 to length[M2[0]] // Length of column of M2

M3 <- [M1[i][j] add M2[i][j]]

return M3

SUB\_MATRIX (M1, M2)

for i <- 1 to length[M1]

for j <- 1 to length[M2[0]]

M3 <- [M1[i][j] sub M2[i][j]]

return M3

MULT\_MATRIX (M1, M2)

for i <- 1 to length[M1]

for j <- 1 to length[M2[0]]

M3[i][j] <- 0

for k <- 1 to length[M2] //Length of rows of M2

M3[i][j] <- M3[i][j] add M1[i][k] mult M2[k][j]

return M3

MULT\_MATRIX\_COF(M,2)

for i <- 1 to length[M]

for j <- 1 to length[M]

M2 <- [M[i][j]] mult 2

return M2

EQUATE\_MATRIX (M1, M2)

X <- MULT\_MATRIX (M1, M2)

Y <- ADD\_MATRIX (M1, M2)

Z <- MULT\_MATRIX\_COF (Y, 2)

A <- SUB\_MAT (X, Z)

return A

RUN TIME: O(N^3)

**Question 6 – REVERSED STRING**

**BIG O notation: O(n)**

**PSEUDOCODE**

REVERSE (string)

y <- string. split ()

x <- y[::-1] // SLICING, not reverse function

return ‘ ‘ . join (x)

**CODE**

def reverse(string):

y = string. split ()

x = y [::-1] # SLICING, not reverse function

return " ". join (x)

print (reverse ("This is awesome"))

**Question 7. – is\_Prime**

**BIG O notation : O(n)**

**PSEUDOCODE**

ISPRIME (num, div)

if div = 1 // BASE CASE

return TRUE

if num (mod) div = 0

return TRUE

return ISPRIME (num, div-1)

**CODE**

def isPrime (num, div):

if (div == 1):

return True

else:

if (num % div ==0):

return False

else:

return isPrime (num, div-1)

n = int (input ("Enter a number greater than 2: "))

print (isPrime (n, n//2))

**Question 8**. – **Vowel removal**

**BIG O notation: O(n)**

**PSEUDOCODE**

VOWEL\_REMOVAL(s)

if length[s] <= 1

return s

elif s[0] is in "aeiouAEIOU"

return VOWEL\_REMOVAL(s[1:])

return s[0] + VOWEL\_REMOVAL(s[1:])

**CODE**

def vowel\_removal(s):

if (len (s) <= 1): # base case

return s

elif s [0] in "aeiouAEIOU":

return vowel\_removal (s [1:])

return s [0] + vowel\_removal(s[1:])

print(vowel\_removal("beautiful"))

**Question 9 – Binary Search**

**BIG O notation : O(n)**

**CODE**

def binary\_search (array, low, high):

first = 0

last = len(array)-1

while first <= last:

mid = (first + last)//2 # floor function

if array[mid] >= low and array[mid]<= high:

return True

if array[mid] >= low and array[mid] >= high:

last = mid - 1

else:

first = mid + 1

return False

a = [1,3,6,7,11,15,17]

print(binary\_search(a,11,13))

print(binary\_search(a,12,14))

**Question 10 – Longest increasing subsequence**

**CODE**

def sequence(ary):

A = []

B = []

for i in range(len(ary)):

if ary[i] > ary[i-1]:

A. append (ary[i])

else:

if len(A) > len(B):

B = A

A = [ary[i]]

if len(A) > len(B):

return A

else:

return B

ary = [1,2,3,4,12,1,2,4,5,1]

print(sequence(ary))

**Question 11 – Delete node function**

**CODE**

class Node(object):

def \_\_init\_\_ (self, value):

self. value = value

self. next = None

self. prev = None

class List(object):

def \_\_init\_\_(self):

self. head = None

self. tail = None

def insert (self, n, x):

#Not actually perfect: how do we prepend to an existing list?

if n!= None:

x. next = n. next

n. next = x

x. prev = n

if x. next!= None:

x. next. prev = x

if self. head == None:

self. head = self. tail = x

x. prev = x. next = None

elif self. tail == n:

self. tail = x

def display(self):

values = []

n = self. head

while n!= None:

values. append (str (n. value))

n = n. next

print ("List: ",",". join(values))

def node\_delete (self, n):

if n. prev != None:

n. prev. next = n. next

else:

self. head = n. next

if n. next != None:

n. next. prev = n. prev

else:

self. tail = n. prev

if \_\_name\_\_ == '\_\_main\_\_':

l=List ()

l. insert (None, Node (4))

l. insert (l. head, Node (6))

l. insert (l. head, Node (8))

l. display ()

#l. node\_delete (l. tail. prev)

l. node\_delete (l. head)

l. display ()

**Question 12 – Tree sort algorithm**

**CODE**

class BinTreeNode (object):

def \_\_init\_\_ (self, value):

self.value=value

self. left=None

self. right=None

def tree\_insert (tree, item):

if tree==None:

tree=BinTreeNode(item)

else:

if (item < tree. value):

if (tree. left==None):

tree. left=BinTreeNode(item)

else:

tree\_insert (tree. left, item)

else:

if (tree. right==None):

tree. right=BinTreeNode(item)

else:

tree\_insert (tree. right, item)

return tree

def postorder (tree):

if (tree. left != None):

postorder (tree. left)

if (tree. right != None):

postorder (tree. right)

print (tree. value)

def in\_order (tree):

stack = []

done = 0

while not done:

if tree != None:

stack. append(tree) # root node

tree = tree. left

else:

if (len(stack) > 0):

tree = stack. pop ()

print (tree. value)

tree = tree. right

else:

done = 1

if \_\_name\_\_ == '\_\_main\_\_':

t=tree\_insert(None,6);

tree\_insert(t,10)

tree\_insert(t,5)

tree\_insert(t,2)

tree\_insert(t,3)

tree\_insert(t,4)

in\_order(t)

**Question 13 – Unweighted graph structure**

**PSEUDOCODE**

CLASS Graph

\_\_init\_\_(self)

nodes <- {}

ADD\_NODE (self, node)

IF node not in nodes:

nodes[node] <- []

ADD\_EDGE (self, frm, to)

IF (frm in nodes) AND (to in nodes):

IF (to not in nodes[frm]):

insert(to) at the nodes[frm]

IF (frm not in nodes[to]):

insert(frm) at the nodes[to]

DISPLAY\_GRAPH(self):

return nodes

g <- Graph ()

g.ADD\_NODE("A")

g.ADD\_NODE("B")

g.ADD\_NODE("C")

g.ADD\_NODE("D")

g.ADD\_NODE("E")

g.ADD\_NODE("F")

g.ADD\_NODE("G")

g.ADD\_EDGE("A","B")

g.ADD\_EDGE("A","D")

g.ADD\_EDGE("B","C")

g.ADD\_EDGE("B","E")

g.ADD\_EDGE("C","G")

g.ADD\_EDGE("D","B")

g.ADD\_EDGE("E","F")

OUTPUT g. DISPLAY\_GRAPH ()

**CODE**

class Graph:

def \_\_init\_\_(self):

self. nodes = {}

def add\_node (self, node):

if node not in self. nodes:

self. nodes[node] = []

def add\_edge (self, frm, to):

if (frm in self. nodes) and (to in self. nodes):

if (to not in self. nodes[frm]):

self. nodes [frm]. append (to)

if (frm not in self. nodes [to]):

self. nodes [to]. append (frm)

def display\_graph(self):

return self. nodes

g = Graph ()

g. add\_node("A")

g. add\_node("B")

g. add\_node("C")

g. add\_node("D")

g. add\_node("E")

g. add\_node("F")

g. add\_node("G")

g. add\_edge("A","B")

g. add\_edge("A","D")

g. add\_edge("B","C")

g. add\_edge("B","E")

g. add\_edge("C","G")

g. add\_edge("D","B")

g. add\_edge("E","F")

print (g. display\_graph ())

**Question 14 – Graph Traversal**

**CODE**

class Graph:

def \_\_init\_\_(self):

self. nodes = {}

def add\_node (self, node):

if node not in self. nodes:

self. nodes[node] = []

def add\_edge (self, frm, to):

if (frm in self. nodes) and (to in self. nodes):

if (to not in self. nodes[frm]):

self. nodes[frm]. append(to)

if (frm not in self. nodes[to]):

self. nodes[to]. append(frm)

def display\_graph(self):

return self. nodes

def dfs (self, start):

queue = [start]

visited = []

while queue != []:

vertex = queue. pop ()

if vertex not in visited:

visited. append (vertex)

for e in self. nodes[vertex]:

queue. append(e)

print ("Depth first search: " + str(visited))

f = open ("dfs. txt", "w")

f. write ("Depth first search: " + str(visited))

f. close ()

def bfs (self, start):

queue = [start]

visited = []

while queue:

vertex = queue. pop (0)

if vertex not in visited:

visited. append(vertex)

#queue = queue + self. nodes[vertex]

queue. Extend (self. nodes[vertex])

print ("Breadth first search: " + str(visited))

f = open ("bfs .txt", "w")

f. write ("Breadth first search: " + str(visited))

f. close ()

g = Graph ()

g.add\_node("A")

g.add\_node("B")

g.add\_node("C")

g.add\_node("D")

g.add\_node("E")

g.add\_node("F")

g.add\_node("G")

g.add\_edge("A","B")

g.add\_edge("A","D")

g.add\_edge("B","C")

g.add\_edge("B","E")

g.add\_edge("C","G")

g.add\_edge("D","B")

g.add\_edge("E","F")

print ("Graph: " + str (g. display\_graph()))

g. dfs ('B')

g. bfs ('B')

**Question 15 – Dijkstra’s Algorithm**

**CODE**

class Graph:

def \_\_init\_\_(self):

self. nodes = {}

def add\_node(self, node):

if node not in self. nodes:

self. nodes[node] = {}

def add\_edge(self, frm, to, weight):

if (frm in self. nodes) and (to in self. nodes):

if (to not in self. nodes [frm]):

self. nodes [frm][to] = weight

if (frm not in self. nodes[to]):

self. nodes [to][frm] = weight

def display\_graph(self):

return self. nodes

def Dijkstras (graph, source, dest): # dest = destination

current = source

pre = {} #previous

tw = {} # tentative\_weight

for node in graph:

tw[node] = float("inf")

pre[node] = None

tw[source] = 0

visited = []

while current != dest:

adjacent = graph[current]

for edge in adjacent:

if tw[current]+adjacent[edge] < tw[edge]:

tw[edge] = tw[current]+adjacent[edge]

pre[edge] = current

visited. Append (current)

minimum = float("inf")

for node in graph:

if (node not in visited) and (tw[node] < minimum):

current = node

minimum = tw[node]

visited. append(dest)

path = shortestPath (pre, dest)

return (path, tw[dest])

def shortestPath (pre, dest):

result = []

while pre[dest] != None:

result. append(dest)#Insert element at the beginning

dest = pre[dest]

result. append(dest)

return result[::-1] # SLICING because append insert at end of list

graph = Graph ()

graph.add\_node("A")

graph.add\_node("B")

graph.add\_node("C")

graph.add\_node("D")

graph.add\_node("E")

graph.add\_node("F")

graph.add\_node("G")

graph.add\_edge ("A","B",1)

graph.add\_edge ("A","D",2)

graph.add\_edge ("B","C",3)

graph.add\_edge ("B","E",1)

graph.add\_edge ("C","G",5)

graph.add\_edge ("D","B",4)

graph.add\_edge ("E","F",2)

Graph = graph. display\_graph ()

Print (Dijkstras (Graph, 'A', 'G'))