210CT Programming, algorithms and data structures

# Irvin Ives lau

# 7537703

# <https://github.coventry.ac.uk/laui/210CT_coventry>

Week 0

# Week 0 Lab

# 1. Write a function that deletes a substring from a given character string, specifying the beginning position and the length of the substring.

import unittest

def removeChar():

theString = input("Enter String: ")

print(theString)

inputDel = input("Enter character to remove: ")

newStr = theString.replace(inputDel, "")

print("You have removed", inputDel, "from", theString, ",")

print("Your new string is: ")

print(newStr)

if len(newStr) == 0:

print("All characters have been removed from the string")

else:

strLength = (len(newStr))

print("The length of your new string is:")

print(strLength)

removeChar()

# 2. Read from the keyboard a list of positive integers, for example: 1, 4, 7, 9

def ReadKeyboard():

inputString = (input("Enter a list of positive integers: "))

firstList = list(inputString)

print("You have entered: ", firstList)

newList = []

for char in firstList:

if char.isdigit():

if int(char) > 0:

newList.append(char)

else:

pass

print("If you have entered alphanumeric characters, we have removed it from your list.")

newList = list(map(int, newList))

print("Your new list is: ", newList)

# a. Write a function that prints the binary representation of the numbers in the list, for the example above it is: 1: 0001 4: 0100 7: 0111 9: 1001

def binaryRep():

intLiteral = list(map(bin, newList))

global binaryList

binaryList = []

for i in intLiteral:

binaryList.append(i[2:])

print("Your list in binary is: ")

print(binaryList)

# b. Print the numbers from the previous list that are palindromes. For example, 9: 1001 is a palindrome.

def palindromes():

reversedBinary = []

for i in binaryList:

reversedBinary.append(i[::-1])

compare = set(binaryList) & set(reversedBinary)

if bool(compare) == True:

print("These are the palindromes in this binary list: ")

compare = list(compare)

print(compare)

# c. Remove all the numbers in the list whose binary representation is a palindrome and print the list after their removal.

def removePalindromes():

global binaryList

noPalindromes = set(binaryList) - set(compare)

print("All palindromes have been removed from the list")

print(list(noPalindromes))

# removePalindromes()

else:

print("There is no palindrome in this binary list")

# palindromes()

# binaryRep()

# ReadKeyboard()

Week 1

# Week 1 Lab

# 1. Write a method that combines two strings, by taking one character from the first string, then one from the second string and so on. Once one string has no characters left it should carry on with the other string.

import unittest

def mergeStrings(string1, string2):

string1List = []

string2List = []

if len(string1) == 0:

pass

else:

for char in string1:

string1List.append(char)

if len(string2) == 0:

pass

else:

for char in string2:

string2List.append(char)

mergedStringList = [j for i in zip(string1List, string2List) for j in i]

mergedString = ''.join(mergedStringList)

print("Your combined string is: ", mergedString)

#mergeStrings()

# 2. Check if a 3 digit number is an Armstrong number. An Armstrong number of three digits is an integer such that the sum of the cubes of its digits is equal to the number itself.

def armstrongNumber():

number = input("Input a 3 digit number: ")

numberList = []

if len(number) < 3:

print("Your number is not a 3 digit number.")

else:

for char in number:

numberList.append(char)

number = int(number)

firstNumber = int(numberList[0])\*\*3

secondNumber = int(numberList[1])\*\*3

thirdNumber = int(numberList[2])\*\*3

combinedNumbers = firstNumber + secondNumber + thirdNumber

if number == combinedNumbers:

print("Yes")

else:

print("No")

#armstrongNumber()

class IsAnagramTests(unittest.TestCase):

def testOne(self):

self.assertEquals(mergeStrings("diana","anaid"))

def main():

unittest.main()

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

main()

Week 2

Week 2

1.

(a) Pseudocode for adding two polynomials:

function polynomials(p1, p2, degree1, degree2):

    output <- empty list

    if degree1 < degree 2:

        insert p2[0] to position [0] in output

        degreeDifference = degree2 - degree1

        output <- output + [ j + i for j, i in zipped(p1, p2[degreeDifference:])]

        finalDegree <- degree2

    else if degree2 < degree1:

        degreeDifference <- degree1 - degree2

        output <- output + [ j + i for j, i in zipped(p1[degreeDifference:], p2)]

        finalDegree <- degree1

return output

return finalDegree

(c) Write the pseudocode for computing the derivative for each of the two polynomials:

function polynomialDerivative (p, degree):

    set derivativeFormula <- nx^(n-1)

        for each number in p:

            p\* [i]

    reduce p[i] by 1

    degree <- degree -1

return p

return degree

2. Write the pseudocode and code for a function that determines whether an array is a palindrome

function palindrome(L):

    L <- list

    K <- empty list

    for each number in L:

        reverse(L) and append to K

    if L == K:

        output Yes

    else:

        No

# Week 2 Lab

# 1. Read the degree of two polynomials and their coefficients, all integers, from the standard input.

# a) Write the pseudocode for adding two polynomials - Written out in Python for algorithm testing.

import unitest

def polynomialAddition(p1, p2, degree1, degree2):

output = []

if degree1 < degree2:

output.insert(0, p2[0])

degreeDifference = degree2 - degree1

output = output + [j + i for j, i in zip(p1, p2[degreeDifference:])]

finalDegree = degree2

elif degree2 < degree1:

degreeDifference = degree1 - degree2

output = output + [j + i for j, i in zip(p1[degreeDifference:], p2)]

finalDegree = degree1

print("The Res is: ", output)

print("The Degree is: ", finalDegree)

polynomialAddition([3,1,9], [2,1,2,1], 2, 3)

# b) Given the following pseudocode, first understand how it works, then implement the code for multiplying the two polynomials in the programming language of your choice

def polynomialMultiplication(p1, p2, degree1, degree2):

res = [0] \* (degree1 + degree2 + 1)

print(res)

for i in range(len(p1)):

for j in range(len(p2)):

res[i + j] = res[i + j] + (p1[i] \* p2[j])

print(res)

#return (res)

class IsAnagramTests(unittest.TestCase):

def testOne(self):

self.assertTrue(polynomialMultiplication([3,1,9], [2,1,2,1], 2, 3))

def main():

unittest.main()

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

main()

Week 3

# Week 3 Lab

# 1. Write a program that reads n words from the standard input, separated by spaces and prints them mirrored (the mirroring function should be implemented recursively). What is the time complexity of the algorithm? Use the BigO notation to express it.

import unittest

import string

def reverseString(string):

if len(string) == 0:

return string

else:

return reverseString(string[1:]) + string[0]

def reverseOrder():

var = reverseString(a)

stringList = []

var = var.split()

for i in var:

stringList.append(i)

stringList.reverse()

newString = " ".join(map(str, stringList))

return newString

a = input("Please enter a string you want to mirror: ")

print(reverseOrder())

# 2. Write a recursive version of linear search on an array of integers. What is the time complexity of the algorithm? Use the BigO notation to express it.

def linearSearch(l, target):

if len(l) == 0:

return False

if l[0] == target:

return True

return linearSearch(l[1:], target)

#print(linearSearch([1,3,5,7,9], 21))

class IsAnagramTests(unittest.TestCase):

def testTwo(self):

self.assertTrue(reverseString("hello world"))

def testThree(self):

self.assertTrue(linearSearch("[1,3,5,7,9]","5"))

def testFour(self):

self.assertFalse(linearSearch("[1,3,5,7,9]","21"))

def main():

unittest.main()

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

main()

Week 4

1. Manually arrange the sequence [2 7 9 4 1 5 3 6 0 8] in ascending order using insertion sort, bubble sort

and selection sort, showing at each step the new configuration of the sequence. How many

comparisons and how many element moves were used by each method? Which is the best performing

method for sorting this array of integers? Which would be the worst arrangement of this sequence?

Insertion Sort:

[2 7 9 4 1 5 3 6 0 8]

[2] [7 9 4 1 5 3 6 0 8] 0 Comparisons, 0 moves

[2 7] [9 4 1 5 3 6 0 8] 1 Comparison, 0 moves

[2 7 9] [4 1 5 3 6 0 8] 2 Comparisons, 0 moves

[2 4 7 9] [1 5 3 6 0 8] 5 Comparisons 2 moves

[1 2 4 7 9] [5 3 6 0 8] 9 Comparisons 6 moves

[1 2 4 5 7 9] [3 6 0 8] 12 Comparisons 8 moves

[1 2 3 4 5 7 9] [6 0 8] 17 Comparisons 12 moves

[1 2 3 4 5 6 7 9] [0 8] 20 Comparisons 14 moves

[0 1 2 3 4 5 6 7 9] [8] 28 Comparisons 22 moves

[0 1 2 3 4 5 6 7 8 9] 30 Comparisons 23 moves

[0 1 2 3 4 5 6 7 8 9]

Bubble Sort:

First pass:

[2 7 9 4 1 5 3 6 0 8]

[2 7 9 4 1 5 3 6 0 8]

[2 7 9 4 1 5 3 6 0 8]

[2 7 4 9 1 5 3 6 0 8]

[2 7 4 1 9 5 3 6 0 8]

[2 7 4 1 5 9 3 6 0 8]

[2 7 4 1 5 3 9 6 0 8]

[2 7 4 1 5 3 6 9 0 8]

[2 7 4 1 5 3 6 0 9 8]

[2 7 4 1 5 3 6 0 8 9]

Second pass:

[2 7 4 1 5 3 6 0 8 9]

[2 7 4 1 5 3 6 0 8 9]

[2 4 7 1 5 3 6 0 8 9]

[2 4 1 7 5 3 6 0 8 9]

[2 4 1 5 7 3 6 0 8 9]

[2 4 1 5 3 7 6 0 8 9]

[2 4 1 5 3 6 7 0 8 9]

[2 4 1 5 3 6 0 7 8 9]

[2 4 1 5 3 6 0 7 8 9]

[2 4 1 5 3 6 0 7 8 9]

Third pass:

[2 4 1 5 3 6 0 7 8 9]

[2 4 1 5 3 6 0 7 8 9]

[2 1 4 5 3 6 0 7 8 9]

[2 1 4 5 3 6 0 7 8 9]

[2 1 4 3 5 6 0 7 8 9]

[2 1 4 3 5 6 0 7 8 9]

[2 1 4 3 5 0 6 7 8 9]

[2 1 4 3 5 0 6 7 8 9]

[2 1 4 3 5 0 6 7 8 9]

[2 1 4 3 5 0 6 7 8 9]

Fourth pass:

[2 1 4 3 5 0 6 7 8 9]

[1 2 4 3 5 0 6 7 8 9]

[1 2 4 3 5 0 6 7 8 9]

[1 2 3 4 5 0 6 7 8 9]

[1 2 3 4 5 0 6 7 8 9]

[1 2 3 4 0 5 6 7 8 9]

[1 2 3 4 0 5 6 7 8 9]

[1 2 3 4 0 5 6 7 8 9]

[1 2 3 4 0 5 6 7 8 9]

[1 2 3 4 0 5 6 7 8 9]

Fifth pass:

[1 2 3 4 0 5 6 7 8 9]

[1 2 3 4 0 5 6 7 8 9]

[1 2 3 4 0 5 6 7 8 9]

[1 2 3 4 0 5 6 7 8 9]

[1 2 3 0 4 5 6 7 8 9]

[1 2 3 0 4 5 6 7 8 9]

[1 2 3 0 4 5 6 7 8 9]

[1 2 3 0 4 5 6 7 8 9]

[1 2 3 0 4 5 6 7 8 9]

[1 2 3 0 4 5 6 7 8 9]

Sixth pass:

[1 2 3 0 4 5 6 7 8 9]

[1 2 3 0 4 5 6 7 8 9]

[1 2 3 0 4 5 6 7 8 9]

[1 2 0 3 4 5 6 7 8 9]

[1 2 0 3 4 5 6 7 8 9]

[1 2 0 3 4 5 6 7 8 9]

[1 2 0 3 4 5 6 7 8 9]

[1 2 0 3 4 5 6 7 8 9]

[1 2 0 3 4 5 6 7 8 9]

[1 2 0 3 4 5 6 7 8 9]

Seventh pass:

[1 2 0 3 4 5 6 7 8 9]

[1 2 0 3 4 5 6 7 8 9]

[1 0 2 3 4 5 6 7 8 9]

[1 0 2 3 4 5 6 7 8 9]

[1 0 2 3 4 5 6 7 8 9]

[1 0 2 3 4 5 6 7 8 9]

[1 0 2 3 4 5 6 7 8 9]

[1 0 2 3 4 5 6 7 8 9]

[1 0 2 3 4 5 6 7 8 9]

[1 0 2 3 4 5 6 7 8 9]

Eighth pass:

[1 0 2 3 4 5 6 7 8 9]

[0 1 2 3 4 5 6 7 8 9]

[0 1 2 3 4 5 6 7 8 9]

[0 1 2 3 4 5 6 7 8 9]

[0 1 2 3 4 5 6 7 8 9]

[0 1 2 3 4 5 6 7 8 9]

[0 1 2 3 4 5 6 7 8 9]

[0 1 2 3 4 5 6 7 8 9]

[0 1 2 3 4 5 6 7 8 9]

[0 1 2 3 4 5 6 7 8 9]

Final Sorting:

[0 1 2 3 4 5 6 7 8 9]

8 passes

Selection Sort

[2 7 9 4 1 5 3 6 0 8]

[2 7 9 4 1 5 3 6 0 8] 1 Comparison, 0 moves

[2 7 9 4 1 5 3 6 0 8] 2 Comparisons, 0 moves

[2 7 9 4 1 5 3 6 0 8] 3 Comparisons, 0 moves

[2 7 9 4 1 5 3 6 0 8] 4 Comparisons, 0 moves

[1 7 9 4 2 5 3 6 0 8] 5 Comparisons, 1 moves

[1 7 9 4 2 5 3 6 0 8] 6 Comparisons, 1 moves

[1 7 9 4 2 5 3 6 0 8] 7 Comparisons, 1 moves

[1 7 9 4 2 5 3 6 0 8] 8 Comparisons, 1 moves

[0 7 9 4 2 5 3 6 1 8] 8 Comparisons, 2 moves

[0 7 9 4 2 5 3 6 1 8] 9 Comparisons, 2 moves

[0 7 9 4 2 5 3 6 1 8] 10 Comparisons, 2 moves

[0 7 9 4 2 5 3 6 1 8] 11 Comparisons, 2 moves

[0 4 9 7 2 5 3 6 1 8] 11 Comparisons, 3 moves

[0 4 9 7 2 5 3 6 1 8] 12 Comparisons, 3 moves

[0 2 9 7 4 5 3 6 1 8] 12 Comparisons, 4 moves

[0 2 9 7 4 5 3 6 1 8] 13 Comparisons, 4 moves

[0 2 9 7 4 5 3 6 1 8] 14 Comparisons, 4 moves

[0 2 9 7 4 5 3 6 1 8] 15 Comparisons, 4 moves

[0 2 9 7 4 5 3 6 1 8] 16 Comparisons, 4 moves

[0 1 9 7 4 5 3 6 2 8] 16 Comparisons, 5 moves

[0 1 9 7 4 5 3 6 2 8] 17 Comparisons, 5 moves

[0 1 9 7 4 5 3 6 2 8] 18 Comparisons, 5 moves

[0 1 7 9 4 5 3 6 2 8] 18 Comparisons, 6 moves

[0 1 7 9 4 5 3 6 2 8] 19 Comparisons, 6 moves

[0 1 4 9 7 5 3 6 2 8] 19 Comparisons, 7 moves

[0 1 4 9 7 5 3 6 2 8] 20 Comparisons, 7 moves

[0 1 4 9 7 5 3 6 2 8] 21 Comparisons, 7 moves

[0 1 3 9 7 5 4 6 2 8] 21 Comparisons, 8 moves

[0 1 3 9 7 5 4 6 2 8] 22 Comparisons, 8 moves

[0 1 3 9 7 5 4 6 2 8] 23 Comparisons, 8 moves

[0 1 2 9 7 5 4 6 3 8] 23 Comparisons, 9 moves

[0 1 2 9 7 5 4 6 3 8] 24 Comparisons, 9 moves

[0 1 2 9 7 5 4 6 3 8] 25 Comparisons, 9 moves

[0 1 2 7 9 5 4 6 3 8] 25 Comparisons, 10 moves

[0 1 2 7 9 5 4 6 3 8] 26 Comparisons, 10 moves

[0 1 2 5 9 7 4 6 3 8] 26 Comparisons, 11 moves

[0 1 2 5 9 7 4 6 3 8] 27 Comparisons, 11 moves

[0 1 2 4 9 7 5 6 3 8] 27 Comparisons, 12 moves

[0 1 2 4 9 7 5 6 3 8] 28 Comparisons, 12 moves

[0 1 2 4 9 7 5 6 3 8] 29 Comparisons, 12 moves

[0 1 2 3 9 7 5 6 4 8] 29 Comparisons, 13 moves

[0 1 2 3 9 7 5 6 4 8] 30 Comparisons, 13 moves

[0 1 2 3 9 7 5 6 4 8] 31 Comparisons, 13 moves

[0 1 2 3 7 9 5 6 4 8] 31 Comparisons, 14 moves

[0 1 2 3 7 9 5 6 4 8] 32 Comparisons, 14 moves

[0 1 2 3 5 9 7 6 4 8] 32 Comparisons, 15 moves

[0 1 2 3 5 9 7 6 4 8] 33 Comparisons, 15 moves

[0 1 2 3 5 9 7 6 4 8] 34 Comparisons, 15 moves

[0 1 2 3 4 9 7 6 5 8] 34 Comparisons, 16 moves

[0 1 2 3 4 9 7 6 5 8] 35 Comparisons, 16 moves

[0 1 2 3 4 9 7 6 5 8] 36 Comparisons, 16 moves

[0 1 2 3 4 7 9 6 5 8] 36 Comparisons, 17 moves

[0 1 2 3 4 7 9 6 5 8] 37 Comparisons, 17 moves

[0 1 2 3 4 6 9 7 5 8] 37 Comparisons, 18 moves

[0 1 2 3 4 6 9 7 5 8] 38 Comparisons, 18 moves

[0 1 2 3 4 5 9 7 6 8] 38 Comparisons, 19 moves

[0 1 2 3 4 5 9 7 6 8] 39 Comparisons, 19 moves

[0 1 2 3 4 5 9 7 6 8] 40 Comparisons, 19 moves

[0 1 2 3 4 5 7 9 6 8] 40 Comparisons, 20 moves

[0 1 2 3 4 5 7 9 6 8] 41 Comparisons, 20 moves

[0 1 2 3 4 5 6 9 7 8] 41 Comparisons, 21 moves

[0 1 2 3 4 5 6 9 7 8] 42 Comparisons, 21 moves

[0 1 2 3 4 5 6 9 7 8] 43 Comparisons, 21 moves

[0 1 2 3 4 5 6 7 9 8] 43 Comparisons, 22 moves

[0 1 2 3 4 5 6 7 9 8] 44 Comparisons, 22 moves

[0 1 2 3 4 5 6 7 9 8] 45 Comparisons, 22 moves

[0 1 2 3 4 5 6 7 8 9] 45 Comparisons, 23 moves

How many comparisons and how many element moves were used by each method?

Insertion Sort: 30 Comparisons, 23 moves

Bubble Sort: 81 Comparisons, 23 moves

Selection Sort: 45 Comparisons, 23 moves

Which is the best performing method for sorting this array of integers?

Insertion sort has the least comparisons in total. Therefore insertion sort is the best performing method for sorting this array of integers.

Which would be the worst arrangement of this sequence?

The worst arragement of this sequence would be bubble sort

The worst arragenment of this sequence would be the sorted sequence in reverse.

[9 8 7 6 5 4 3 2 1 0]

# Week 4 Lab

# 1. Manually arrange the sequence [2 7 9 4 1 5 3 6 0 8] in ascending order using insertion sort, bubble sort and selection sort, showing at each step the new configuration of the sequence.

def insertionSort(l):

for i in range(1, len(l)):

currentValue = l[i]

print(l)

while i > 0 and l[i - 1] > currentValue:

l[i] = l[i - 1]

i = i - 1

l[i] = currentValue

# l = [2, 7, 9, 4, 1, 5, 3, 6, 0, 8]

# insertionSort(l)

# print(l)

def bubbleSort(l):

for passNumber in range(len(l) - 1, 0, -1):

for i in range(passNumber):

print(l)

if l[i] > l[i + 1]:

var = l[i]

l[i] = l[i + 1]

l[i + 1] = var

# l = [2, 7, 9, 4, 1, 5, 3, 6, 0, 8]

# bubbleSort(l)

# print(l)

def selectionSort(l):

for i in range(len(l) - 1, 0, -1):

maxIndex = 0

for currentLocation in range(1, i + 1):

print(l)

if l[currentLocation] > l[maxIndex]:

maxIndex = currentLocation

var = l[i]

l[i] = l[maxIndex]

l[maxIndex] = var

# l = [2, 7, 9, 4, 1, 5, 3, 6, 0, 8]

# selectionSort(l)

# print(l)

def guessingGame():

import random as random

binary = False

lowNumber, highNumber = 1, 20000

number = random.randint(lowNumber, highNumber)

print("A number has been selected from", lowNumber, "and", highNumber)

lo = 1

hi = highNumber

guesses = 0

for i in range(lowNumber, highNumber):

#guess = int(input("What is your guess: "))

if binary:

guess = lo + (hi - lo) // 2 # integer division

else:

guess = random.randint(lo, hi)

print("Guess:", guess)

guesses += 1

#check the guessed number

if guess > number:

print("Lower:")

hi = guess # bring down the upper bound

elif guess < number:

print("Higher:")

lo = guess # push up the lower bound

else:

break

print("That took", guesses, "guesses")

print("That took {0} guesses".format(guesses))

guessingGame()

Week 5

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 delete(self, N):

head = self.head

while head:

if head.value == N: #if value of head is == N do not proceed

if head.prev != None: #if head.prev is not None do not proceed

head.prev.next = head.next

if head.next != None: #if head.next is not None do not proceed

head.next.prev = head.prev

else:

self.head = head.next

return True

else:

head = head.next

return False

def display(self):

values = []

n = self.head

while n != None:

values.append(str(n.value))

n = n.next

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

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

l = List()

l.insert(None, Node(4))

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

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

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

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

l.delete(8)

l.display()

Week 6

class BinTreeNode(object):

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

self.value = value

self.left = None

self.right = None

self.parent = 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)

tree.left.parent = tree

else:

tree\_insert(tree.left, item)

else:

if(tree.right == None):

tree.right = BinTreeNode(item)

tree.right.parent = tree

else:

tree\_insert(tree.right, item)

return tree

# change search to return the needed node and its parent

def searchNode(tree, target):

if tree.value == target:

return tree

elif target < tree.value:

return searchNode(tree.left, target)

else:

return searchNode(tree.right, target)

def deleteNode(tree, value):

parent = None

node = searchNode(tree, value)

#case 1: data not found

if node is None or node.value != value:

return False

#case 2: no children

elif node.left is None and node.right is None:

if value < node.parent.value:

node.parent.left = None

else:

node.parent.right = None

return True

#case 3: left child only

elif node.left and node.right is None:

if value < node.parent.value:

node.parent.left = node.left

else:

node.parent.right = node.left

return True

#case 4: remove-node has right child only

elif node.left is None and node.right:

if value < node.parent.value:

node.parent.left = node.right

else:

node.parent.right = node.right

return True

#case 5: remove-node has left and right children

else:

delNodeParent = node

delNode = node.right

while delNode.left:

delNodeParent = delNode

delNode = delNode.left

node.value = delNode.value

if delNode.right:

if delNodeParent.value > delNode.value:

delNodeParent.left = delNode.right

elif delNodeParent.value < delNode.value:

delNodeParent.right = delNode.right

else:

if delNode.value < delNodeParent.value:

delNodeParent.left = None

else:

delNodeParent.right = None

def max(self, node): # finding the max value (from the left subtree)

root = node

if node == None:

node = self.node

if node.left != None:

node = node.left

else:

return node

if node.right != None:

return self.max(node=node.right)

def postorder(tree):

if(tree.left != None):

postorder(tree.left)

if(tree.right != None):

postorder(tree.right)

print(tree.value)

def in\_order(tree):

if(tree.left != None):

in\_order(tree.left)

print(tree.value)

if(tree.right != None):

in\_order(tree.right)

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)

tree\_insert(t, 11)

#print(searchNode(t, 11).value) - search value

deleteNode(t, 4) # deleting the node 11

in\_order(t)

Week 7

# Week 7 Lab

# 1. Implement an unweighted and undirected graph data structure in the programming language of your choice, where the nodes consist of positive integers. You can either use an adjacency matrix or an adjacency list approach. You must use Object Oriented Programming for this task.

class Stack:

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

self.items = []

def push(self, item):

self.items.append(item)

def pop(self):

return self.items.pop()

def empty(self):

return self.items == []

def peek(self):

return self.items[len(self.items) - 1]

def size(self):

return len(self.items)

class Graph:

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

self.graphDict = {}

def add\_vertex(self, vertex):

if vertex not in self.graphDict:

self.graphDict[vertex] = []

def add\_directed\_edge(self, v1, v2):

if v1 in self.graphDict:

self.graphDict[v1] = []

self.graphDict[v1].append(v2)

if v2 in self.graphDict:

self.graphDict[v2] = []

self.graphDict[v2].append(v1)

def add\_undirected\_edge(self, v1, v2):

self.add\_directed\_edge(v1, v2)

self.add\_directed\_edge(v2, v1)

def display(self):

return self.graphDict

# 2. Implement either the BFS or DFS graph traversal (you can use the pseudocode in the lecture slides to help). Save the nodes traversed in a text file, entitled either DFS.txt or BFS.txt, depending on which traversal you end up implementing.

def depth\_first\_search(self, start\_node):

s = Stack()

visited = []

s.push(start\_node)

while s.empty() == False:

u = s.pop()

if u not in visited:

visited.append(u)

for e in self.graphDict[u]:

s.push(e)

f = open('DFS.txt', 'w')

f.write(str(visited))

f.close

return visited

# 3. Implement a function called isPath(v,w), where v∈V and w∈V (V is the set of all nodes in the graph) to check if there is a path between the two given nodes v and w.

def isPath(self, start\_node, end\_node):

all\_nodes = self.depth\_first\_search(start\_node)

if start\_node and end\_node in all\_nodes:

return True

else:

return False

if \_\_name\_\_ == "\_\_main\_\_":

vertex1 = "1"

vertex2 = "2"

vertex3 = "3"

vertex4 = "4"

vertex5 = "5"

g = Graph()

g.add\_vertex(vertex1)

g.add\_vertex(vertex2)

g.add\_vertex(vertex3)

g.add\_vertex(vertex4)

g.add\_vertex(vertex5)

g.add\_directed\_edge(vertex1, vertex2)

g.add\_directed\_edge(vertex1, vertex5)

g.add\_directed\_edge(vertex1, vertex3)

g.add\_directed\_edge(vertex2, vertex5)

g.add\_directed\_edge(vertex2, vertex4)

print(g.display())

print(g.display())

print(g.depth\_first\_search(vertex1))

print(g.isPath(vertex1, vertex3))

print(g.isPath(vertex1, vertex4))

Week 8

# Week 7 Lab

# 1. Implement an unweighted and undirected graph data structure in the programming language of your choice, where the nodes consist of positive integers. You can either use an adjacency matrix or an adjacency list approach. You must use Object Oriented Programming for this task.

class Stack:

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

self.items = []

def push(self, item):

self.items.append(item)

def pop(self):

return self.items.pop()

def empty(self):

return self.items == []

def peek(self):

return self.items[len(self.items) - 1]

def size(self):

return len(self.items)

class vertex(object):

def \_\_init\_\_(self, label, edges = []):

self.label = label

self.edges = []

class Graph:

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

self.graphDict = {}

def add\_vertex(self, vertex):

if vertex not in self.graphDict:

self.graphDict[vertex] = []

def add\_directed\_edge(self, v1, v2):

if v1 in self.graphDict:

self.graphDict[v1] = []

self.graphDict[v1].append(v2)

if v2 in self.graphDict:

self.graphDict[v2] = []

self.graphDict[v2].append(v1)

def add\_undirected\_edge(self, v1, v2):

self.add\_directed\_edge(v1, v2)

self.add\_directed\_edge(v2, v1)

def display(self):

return self.graphDict

# 2. Implement either the BFS or DFS graph traversal (you can use the pseudocode in the lecture slides to help). Save the nodes traversed in a text file, entitled either DFS.txt or BFS.txt, depending on which traversal you end up implementing.

def depth\_first\_search(self, start\_node):

s = Stack()

visited = []

s.push(start\_node)

while s.empty() == False:

u = s.pop()

if u not in visited:

visited.append(u)

for e in self.graphDict[u]:

s.push(e)

f = open('DFS.txt', 'w')

f.write(str(visited))

f.close

return visited

# 3. Implement a function called isPath(v,w), where v∈V and w∈V (V is the set of all nodes in the graph) to check if there is a path between the two given nodes v and w.

def isPath(self, start\_node, end\_node):

all\_nodes = self.depth\_first\_search(start\_node)

if start\_node and end\_node in all\_nodes:

return True

else:

return False

# Week 8 Lab

def dijkstra(self, source, desintation):

v = source

for n in self.graphDict:

n.tw = float('inf')

source.tw = 0

visited = []

while v != destination:

for u in v.edges:

if v.tw + v[u].w < u.tw:

u.tw = v.tw + v[u].w

u.pre = v

visited.append(v)

minimum = float('inf')

for n in v:

if n not in visited and n.tw < minimum:

v = n

minimum = n.tw

if \_\_name\_\_ == "\_\_main\_\_":

vertex1 = "1"

vertex2 = "2"

vertex3 = "3"

vertex4 = "4"

vertex5 = "5"

g = Graph()

g.add\_vertex(vertex1)

g.add\_vertex(vertex2)

g.add\_vertex(vertex3)

g.add\_vertex(vertex4)

g.add\_vertex(vertex5)

g.add\_directed\_edge(vertex1, vertex2)

g.add\_directed\_edge(vertex1, vertex5)

g.add\_directed\_edge(vertex1, vertex3)

g.add\_directed\_edge(vertex2, vertex5)

g.add\_directed\_edge(vertex2, vertex4)

print(g.display())

g.dijkstra(vertex3, vertex5)

# print(g.depth\_first\_search("1"))

# print(g.isPath("1","3"))

# print(g.isPath("1","4"))