Michael Gonzalez

CS 2302 Data Structures

Olac Fuentes

05/14/2019

Lab 8 – Algorithm Design

For this lab we will be solving 2 problems using different types of programming techniques, Randomized and Backtracking. 1.Use Randomized algorithms to write a program to "discover" trigonometric identities we will use the various trig formulas given to us in the lab document to test all combinations of a trig function and show its equalities. 2. Use Backtracking to solve the partition problem of a given set and output the two sets that solve it or indicate that no partition exists.

My solution for part 1 is simple, create the method random\_Identity\_testing(). In order for this to be called a random algorithm there must be a random element so the first thing the method does is to create a random variarble t from the range -360 to 360. Next I declared all the different trig functions given in the lab document, variables a-p, using the math library to implement cos(t) and sin(t) in python. Next I created an empty List L and appended each index with the value of the trig function and the name of the trig function, L.append([letter,”Name”]). Next, I created another random variable z from 0 to 15 so that each time the program it prints what trig function is using to try and find its identities. Lastly I created a for loop of i in range(length( L)), to traverse the entire list, and then I made an if statement. If the rounding of the random trig function is equal in value to another trig function and if they are not the same function, then it prints out Identity exists: “random function” = “identity found”. The tested running time of this method is 0 seconds, this means that the program was fast enough to start process and output a solution in less than a second.

My solution for part 2, is divided into 3 methods, 1. printSets(S1,S2), 2. findSets(arr, n, set1, set2, sum1, sum2, pos), and 3. isPartitionPoss(arr, n). 1. printSets(S1,S2) takes in two lists and prints them out int the format that was specified in the lab8 document. 2. findSets(arr, n, set1, set2, sum1, sum2, pos) takes a list called arr, the length of arr denoted as n, 2 sets denoted as set1 and set 2, 2 sums denoted as sum1 and sum2 and a position index denoted as pos. This method is used to find the partition sets once we know that the partition is possible. First it goes into an If statement where if pos == n then to see if the partition has been made correctly, prints out the sets and returns true, otherwise it returns false. This is use as a base case to test if the set we were given is partitionable or not. Next the program appends arr[pos] to set 1 and makes a variable called res where it stores the value of the recursive call findSets(arr, n, set1, set2, sum1 + arr[pos], sum2, pos + 1) assuming the recursive calls work it will then go into an if statement where if the variable res is true then it will return true, if not then the last elemnt of set1 will popped then appended into set2, then it makes a return of the recursive call findSets(arr, n, set1, set2, sum1, sum2 + arr[pos], pos + 1) long story short this is the way backtracking algorithms work by at first accepting all yes options and when we find the answer to be incorrect we back up one level and try from there and this will keep on going until it reaches a suitable answer and prints out the correct sets. 3. isPartitionPoss(arr, n) takes a list denoted as arr, and then length of arr as n. what this method does is that first it creates a variable called sum and equals it to zero. Next it adds to sum all the values stored in the arr list. Then, it goes into an If statement where if the sum of all parts in the list is not divisible by 2 then it returns false, if it can it will declare 2 empty sets, set1 and set2 and make a return call to the function findSets(arr, n, set1, set2, 0, 0, 0) . These 3 methods work together to identify if a set is partitionable, to find the sets, and if its all true to print out the corresponding sets. The tested running time of this method with the example set given in the lab document S= {5,5,1,11} is 0 seconds, this means that the program was fast enough to start process and output a solution in less than a second.

Conclusion

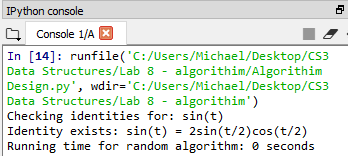
In conclusion, I learned how to design a random algorithm and how to design a backtracking algorithm. I have now a better appreciation for the different types of programming and understand when one type of programming is preferred over the other. I have become more comfortable with coding in python than in lab 6 and I believe that I will be able to learn more from the last lab and hopefully make the grade in this class.

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

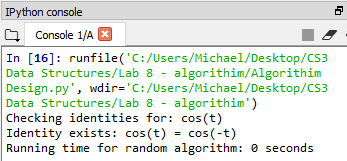
– Michael Gonzalez

Experiments/Results

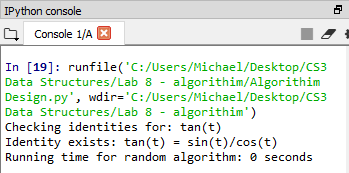
1a. (when z=0):



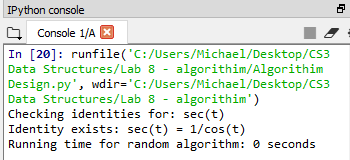
1a. (when z=1)



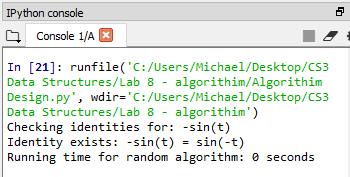
1a. (when z=2)



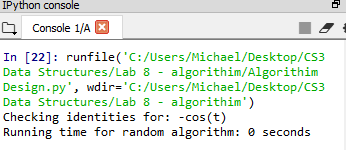
1a. (when z=3)



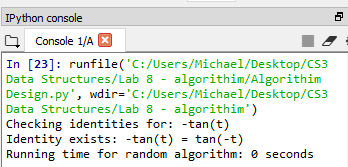
1a. (when z=4)



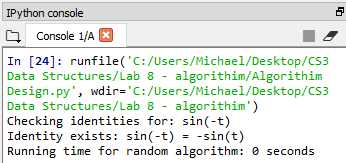
1a. (when z=5)



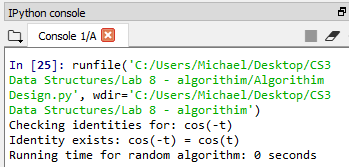
1a. (when z=6)



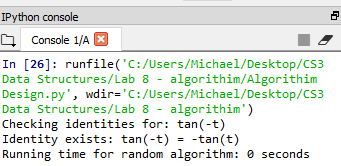
1a. (when z=7)



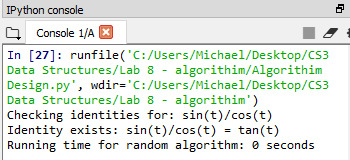
1a. (when z=8)



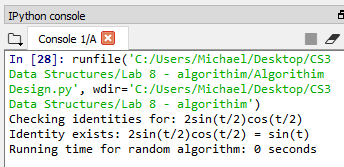
1a. (when z=9)



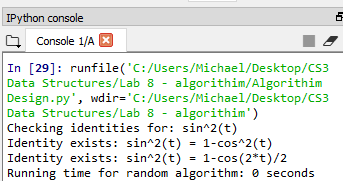
1a. (when z=10)



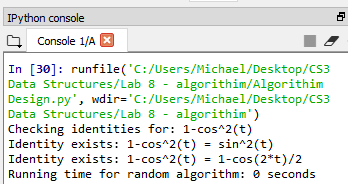
1a. (when z=11)



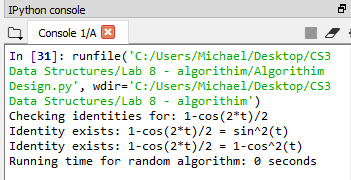
1a. (when z=12)



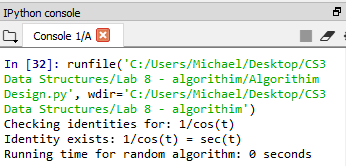
1a. (when z=13)



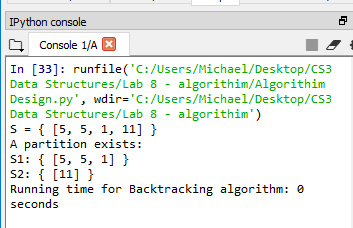
1a. (when z=14)



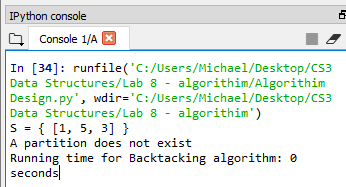
1a. (when z=15)



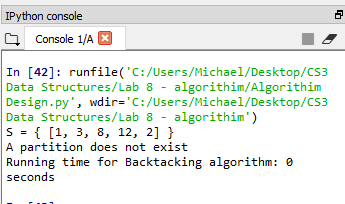
1b. (S= [5, 5, 1, 11]) sum = 22



2b. (S= [1, 5, 3] sum = 9 \*\*odd sum should not be possible



2c. (S= [1,3,8,12,2]



Appendix – code

#Author: Michael Gonzalez

#Course: CS 2302 Data Structures

#Lab 8

#TA: Anindita Nath & Eduuardo Lara

#Purpose:the purpose of this lab is to write specific algorithim techniques to solve 2 problems

#1.(Randomized algorithms) Write a program to "discover" trigonometric identities. Your program should

#test all combinations of the trigonometric expressions shown below and use a randomized algorithm to

#detect the equalities. For your equality testing, generate random numbers in the 􀀀 to range.

#(a) sin(t)

#(b) cos(t)

#(c) tan(t)

#(d) sec(t)

#(e) -sin(t)

#(f) -cos(t)

#(g) -tan(t)

#(h) sin(-t)

#(i) cos(-t)

#(j) tan(-t)

#(k) sin(t)/cos(t)

#(l) 2 sin(t=2) cos(t=2)

#(m) sin2(t)

#(n) 1 - cos2(t)

#(o) 1-cos(2t)/2

#(p) 1/cos(t)

#2. (Backtracking) The partition problem consists of determining if there is a way to partition a set of integers

#S into two subsets S1 and S2 such that P S1 = P S2. Recall that S1 and S2 are a partition of S if and only

#if S1[S2 = S and S1\S2 = fg. Write a function that solves the partition problem using backtracking. If a

#partition exists, your program should display it; otherwise it should indicate that no partition exists. For

#example, if S = f2; 4; 5; 9; 12g, your program should output the partition S1 = f2; 5; 9g and S2 = f4; 12g

#and if S = f2; 4; 5; 9; 13g your program should indicate that no partition exists.

#Given the little time available, a demo will not be required, thus it is very important

# Last modified May, 11 2019

import random

import math

##################################################

# LAB START #

##################################################

def random\_Identity\_testing():

t = random.randrange(-360,360)

a = math.sin(t)

b = math.cos(t)

c = math.tan(t)

d = 1/math.cos(t) #sec is defiend as 1/cos(thetha) since there is no math.sec this had to be done

e = -math.sin(t)

f = -math.cos(t)

g = -math.tan(t)

h = math.sin(-t)

i = math.cos(-t)

j = math.tan(-t)

k = math.sin(t)/math.cos(t)

l = 2\*math.sin(t/2)\*math.cos(t/2)

m = math.sin(t)\*math.sin(t) # sin^2(t) is not defined in Math so i substituted it by (sin(t)\*sin(t))

n = 1-(math.cos(t)\*math.cos(t)) # Cos^2(t) is not defined so i substituded it by (cos(t)\*cos(t))

o = (1-math.cos(2\*t))/2

p = 1/math.cos(t)

L = []

L.append([a,"sin(t)"])

L.append([b,"cos(t)"])

L.append([c,"tan(t)"])

L.append([d,"sec(t)"])

L.append([e,"-sin(t)"])

L.append([f,"-cos(t)"])

L.append([g,"-tan(t)"])

L.append([h,"sin(-t)"])

L.append([i,"cos(-t)"])

L.append([j,"tan(-t)"])

L.append([k,"sin(t)/cos(t)"])

L.append([l,"2sin(t/2)cos(t/2)"])

L.append([m,"sin^2(t)"])

L.append([n,"1-cos^2(t)"])

L.append([o,"1-cos(2\*t)/2"])

L.append([p,"1/cos(t)"])

# print(L)

z = random.randint(0,15)

print("Checking identities for:", L[z][1])

for j in range(len(L)):

if round(L[z][0],5) == round(L[j][0],5):

if z != j:

print("Identity exists:",L[z][1],"=",L[j][1])

# Function to print the equal sum sets of the array.

def printSets(set1, set2) :

# Print set 1.

print ("S1: {",set1,"}")

# Print set 2.

print ("S2: {",set2,"}")

# function to find the sets of the

# array which have equal sum.

def findSets(arr, n, set1, set2, sum1, sum2, pos) :

if (pos == n) :

if (sum1 == sum2) :

print("A partition exists:")

printSets(set1, set2)

return True

else :

return False

set1.append(arr[pos])

res = findSets(arr, n, set1, set2, sum1 + arr[pos], sum2, pos + 1)

if (res) :

return res

set1.pop()

set2.append(arr[pos])

return findSets(arr, n, set1, set2, sum1, sum2 + arr[pos], pos + 1)

# Return true if array arr can be partitioned

# into two equal sum sets or not.

def isPartitionPoss(arr, n) :

# Calculate sum of elements in array.

sum = 0

for i in range(0, n):

sum += arr[i]

# If sum is odd then array cannot be

# partitioned.

if (sum % 2 != 0) :

return False

# Declare vectors to store both the sets.

set1 = []

set2 = []

# Find both the sets.

return findSets(arr, n, set1, set2, 0, 0, 0)

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

arr = [5, 5, 1, 11]

n = len(arr)

print("S = {",arr,"}") #cool trick to print the list

if (isPartitionPoss(arr, n) == False) :

print ("A partition does not exist")

random\_Identity\_testing()