Jon Munoz

Lab 8

TA: Anindita Nath

CS2302

Professor: Olac Fuentes

**INTRODUCTION**

For this lab we had to do algorithms one being random and the other being backtracking. With the random algorithm what we had to do was test the equality of multiple trigonometric functions and see if they were equal in all combinations. For the backtracking algorithm what we had to do was see if there are two subsets of a set that both equal to the same number.

If there is two such subsets, then we are supposed to print them

**PROPOSED SOLUTION AND DESIGN**

For these questions I am not sure if I did the first one right. The way that I did it was create a list of all the trigonometric functions. Once I had this list in my method, I created a list with sub lists inside of it each corresponding to the different trig functions and also a list called TF to store the sub list. Once I had this list of list I then did two for loops one going through the entire list of trig functions so we can test one by one and the second for loop also goes through the entire list of trig functions in order to compare the list at [i] to everything in the trig list. If the two functions are not equal, I would set a Boolean variable that I had called val to False and if not then it would stay at its default value of True. Once I had this value, I would add it to TF[i]. So, what this would do is compare a trig function to every other trig function and then move to the next trig function and then repeat the process. At the end of all of this I would return the TF list. For the method that test to see if there are two sub lists within a set that add up to the same number I created two methods. One method, called subsets, got the sum of the passed set and if it was not an even sum then I return false since there is no way that you can get two subsets to add up to an odd number while having those subsets equal the same number. If the sum was even then id call my second method, subsets2, which is a modified subsetsum method with the goal being half of the original sum. This would then return the sublist that adds up to half of the original sum. Once I had this subset I would then create the other subset by adding the numbers that were not in the returned subset to another list and seeing if that list adds up to half of the original sum. If this checked out then I would print the two subsets.

**EXPERIMENTAL RESULTS**

Below are the method calls for my two different methods. My original functions list looked like this:

*﻿['sin(t)', 'cos(t)', 'tan(t)', '1/cos(t)', '-sin(t)', '-cos(t)', '-tan(t)', 'sin(-t)', 'cos(-t)', 'tan(-t)', 'sin(t)/cos(t)', '2\*sin(t/2)\*2\*cos(t/2)', 'sin(t)\*\*2', '1-cos(t)\*\*2', '(1-cos(2\*t))/2', '1/cos(t)']*

I then ran the discovery method to see the equalities between the functions and this was the result:

A picture containing water, text

Description automatically generated

The index in the entire list is each trig function that was in the passed list and within each sub list the index is the result of the comparison done in the discovery method. So lets say the returned list is called S, S[1][1] would be the result of comparing cos(t) to cos(t) and since they are the same True occupies that index and for S[2][0] that would be tan(t) compared to sin(t) and so on. For the subset method I pass a list containing ﻿[2, 4, 5, 9, 12] and the resulting subsets are [4,12] and [2,5,9]. I then pass the list ﻿[2, 4, 2, 4, 5, 9, 12] which results in the method printing “No sub list exist”. Below is a screen shot of each:

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generatedBelow are the running times of my functions in seconds/milliseconds:

A screenshot of a cell phone

Description automatically generated

**CONCLUSION**

At the end of this lab I learned how to attack a problem in parts when it came to the subsets problem and then how to use the eval() function in python. I also learned how to get more comfortable with both random and backtracking algorithms.

**APPENDIX**

**SOURCE CODE**

|  |
| --- |
| #Jon Munoz |
|  | #CS2302 Data Structures |
|  | #Lab 8 |
|  | #Instructor:Olac Fuentes |
|  | #TA:Anindita Nath |
|  | #Last Modified 5/9/19 |
|  |  |
|  | import random |
|  | import numpy as np |
|  | from math import \* |
|  | import math |
|  | import datetime |
|  |  |
|  | #this method compares two trig functions together and sees if they are similar or not |
|  | def discovery(L,tries = 1000,tolerence = 0.0001): |
|  | TF = []#TF is the list where I store the results of my comarisons |
|  | for b in range(len(L)):#this for loop fills TF with empty list equal to the number of trig functions that are in the passed list |
|  | TF.append([]) |
|  | for i in range(len(L)):#this for loop goes through each index in TF one by one in order to store results at the correct index |
|  | val = True#set val equal to True since before comparisons it should be True |
|  | for j in range(len(L)):#this for loop gooes through the entire list comparing each index j to index i |
|  | val = True#set val equal to True since before comparisons it should be True |
|  | for k in range(tries):#for loop does 1000 runs to see if the two functions are similar |
|  | t = random.uniform(-math.pi,math.pi)#gets a random number between -pi and pi |
|  | y1 = eval(L[i]) |
|  | y2 = eval(L[j]) |
|  | if np.abs(y1 - y2) > tolerence:#if the difference is greater than the sum then the two functions are not similar so set val to False |
|  | val = False |
|  | TF[i].append(val)#append val to the results list |
|  | return TF#return the TF list once all comparisons are done |
|  |  |
|  | #this method is the actual backtracking part of the subsets method |
|  | def subsets2(S,last,goal): |
|  | if goal == 0:#if goal is 0 then you return just an empty list |
|  | return [] |
|  | if goal<0 or last<0:#if goal or last value is less that 0 return an empty list |
|  | return [] |
|  | subset = []#create a list subset to store results |
|  | if S[last] > goal:#if the value at index last is greater than goal then you obviously cant take it |
|  | return subsets2(S, last - 1, goal) #dont take S[last] |
|  | else: |
|  | subset.append(S[last])#add S[last] to the subset list |
|  | return subsets2(S,last-1,goal - S[last]) + subset #take S[last] |
|  |  |
|  | #this method does the basic preliminary checks to see if its even possible to get two equal subsets |
|  | def subsets(S,index): |
|  | sumList = 0#holds the sum of the passed list S |
|  | for i in range(len(S)):#for loop goes through entire list and adds it to sum |
|  | sumList += S[i] |
|  | if sumList % 2 != 0:#if the sum of the original list is not even then it is not possible to get two subsets to have equal value so return False immediately |
|  | return False |
|  | NL = subsets2(S,index,sumList//2)#call subsets2 in order to get one of the subsets of S |
|  | sumNL = 0#sumNL will hold the sum of |
|  | sumOtherHalf = 0#holds the sum of the other sublist not returned by the method |
|  | otherHalf = []#other list that wasnt returned |
|  | for j in range(len(S)):#fills otherHalf with elements not in NL |
|  | if S[j] not in NL: |
|  | otherHalf.append(S[j]) |
|  | for i in range(len(NL)):#gets the sum of NL |
|  | sumNL += NL[i] |
|  | for g in range(len(otherHalf)):#gets the sum of the other list |
|  | sumOtherHalf += otherHalf[g] |
|  | if sumNL == sumList//2 and sumOtherHalf == sumList//2:#if the two list equal to half the original then print the list |
|  | print(NL, otherHalf) |
|  | else: |
|  | print("No sublist exist")#else printno such sublist exists |
|  |  |
|  | #below are all the added functions |
|  | f1 = 'sin(t)' |
|  | f2 = 'cos(t)' |
|  | f3 = 'tan(t)' |
|  | f4 = '1/cos(t)'#put sec as i/cos since sec was not available and these two are equivalent |
|  | f5 = '-sin(t)' |
|  | f6 = '-cos(t)' |
|  | f7 = '-tan(t)' |
|  | f8 = 'sin(-t)' |
|  | f9 = 'cos(-t)' |
|  | f10 = 'tan(-t)' |
|  | f11 = 'sin(t)/cos(t)' |
|  | f12 = '2\*sin(t/2)\*2\*cos(t/2)' |
|  | f13 = 'sin(t)\*\*2' |
|  | f14 = '1-cos(t)\*\*2' |
|  | f15 = '(1-cos(2\*t))/2' |
|  | f16 = '1/cos(t)' |
|  | #f17 = 'sin(t)' |
|  |  |
|  | #append all functions to a list |
|  | functions = [] |
|  | functions.append(f1) |
|  | functions.append(f2) |
|  | functions.append(f3) |
|  | functions.append(f4) |
|  | functions.append(f5) |
|  | functions.append(f6) |
|  | functions.append(f7) |
|  | functions.append(f8) |
|  | functions.append(f9) |
|  | functions.append(f10) |
|  | functions.append(f11) |
|  | functions.append(f12) |
|  | functions.append(f13) |
|  | functions.append(f14) |
|  | functions.append(f15) |
|  | functions.append(f16) |
|  | #functions.append(f17) |
|  | #print(functions) |
|  | # |
|  | #start = datetime.datetime.now() |
|  | #result = discovery(functions) |
|  | #end = datetime.datetime.now() |
|  | #elapsed = end - start |
|  | #print(elapsed) |
|  | # |
|  | #print(result) |
|  |  |
|  | #test list for subsets |
|  | test = [2, 4, 5, 9] |
|  |  |
|  | start = datetime.datetime.now() |
|  | subsets(test,len(test) - 1) |
|  | end = datetime.datetime.now() |
|  | elapsed = end - start |
|  | print(elapsed)  “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 provide inappropriate assistance to any student in the class.” |

