Robert Marc

80487972

CS 2302

MW 1:30-3:00

Lab 8 Report

Introduction

For the final lab, we had two parts. The first part required us to build a randomized algorithm that tested trigonometric identities against each other for equality. We were required to generate a random value for theta between -pi and pi for the purposes of testing. Then, it would randomly compare the identities against each other for that value and output which identities were equal.

For the second part, we were required to implement a function that partitions a subset of numbers, S, into two separate subsets that have an equal sum. To do this, we needed to use a backtracking algorithm.

Proposed Solutions

For the first part, I split the problem into two separate functions: trigBuild and trigTest. trigBuild took the randomly generate value for theta and calculated each of the identities that we were required to test. It would store each of these values into an item in a list. Each item of the list contained the value of the identity for a particular value for theta, as well as the string representation of the identity. It would then return the reference to that list. trigTest would be passed the reference of the trigonometric identity list, and it would build one list for every possible combination of trigonometric identities, ignoring duplicate combinations as well as combinations where an identity would be tested against itself. Next, it would randomly pick one of these combinations and test the two identities against each other. If they were equal, that combination was stored in a list for keeping track of equal identities. If they weren’t equal, then that combination was stored in a separate list. Afterwards, it would use these store combinations to print out the identity equalities; which includes the values of each identity and a statement on if they were equal.

For the second part, I again split the problem into two separate functions: partition and subset. In partition, the sum of the entire set is found. If the sum is not even, no partition can exist, and False is returned. If the sum is even, the subset method is called to determine if a subset can be found. This subset method will ultimately return a Boolean value of True if a subset exists, or False if not; as well as a list containing all of the indices of set S that contribute to the subset sum. The partition method prints a statement depending on this value stating if a partition exists or not, and then it returns the Boolean value that subset returned and the list of indices.

The subset method uses two base cases to determine if it has finished searching for a base case. If the remaining sum is equal to zero, then True is returned. If the remaining index to pick from is zero, and the sum does not equal zero, then False is returned. The choice of picking the next number is stored in two variables. One that holds the Boolean value of choosing the next number, and one that holds a list of indices that contribute to the solution. If se (the one with the Boolean) is True, then the index of the number chosen is appended to the list of indices, and it is returned alongside True. If it is False, then the choice of not picking the next number is returned.

Returning to the main method. The Boolean value of whether or not a partition exists is used to determine if the subsets should be printed. If True, it builds the two subsets by appending the values of the indices of S found in the list of indices returned by partition into s1, and those indices not found in the list to s2. These two lists are then printed.

Experimental Results

Randomization Algorithm

Averag Time to build identitiy list and test equality across 10 tests:

0.13 seconds

Partition Problem with Backtracking

Average time to find partition and subsets for a set of N numbers, across 10 tests:

|  |  |
| --- | --- |
| N = | Time in Seconds |
| 100 | 0.0002 |
| 200 | 0.0004 |
| 400 | 0.0011 |
| 800 | 0.0021 |
| 1600 | 0.0053 |

Outputs

Trigonometric Equalities

A screenshot of a cell phone

Description automatically generatedA screenshot of text

Description automatically generated

Identities that are equal are on the left, unequal are on the right

Too many comparisons to neatly show, so just a sample is present.

Partition Problem

A screenshot of a cell phone

Description automatically generated

Conclusions

Solving the partition subset problem of this lab was difficult. I ran into many problems, such as values being repeated more than was present in the original set, causing massive subsets that shouldn’t exist. In the end, I found that storing the indices of values that contributed to the solution to be more effective. I did not run into many issues with the trigonometric identities problem, it was just a matter of determining how to test each identity against another identity that it hadn’t been tested against or against itself.

Academic Honesty Statement

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.”



Appendix

"""

@Course: CS2302 MW 1:30-2:50 pm

@Author: Robert Marc, 80487972

@Assignment: Lab 8

@Instructor: Dr. Olac Fuentes

@TAs: Anindita Nath and Maliheh Zargaran

@Date of Last Modification: 5/7/19 @9:08 PM

@Purpose: To design algorithms that 'discover' and test trigonometric identies

"""

import math

import random

import numpy as np

import time

def trigBuild(t):

"""

Builds an array that holds all the calculate values for particular

trigonometric identies. This is to prevent the testing program

from continuously recalculating identies. The string version of the

identity is also held.

"""

print("t = ",t)

print()

trig = []

trig.append([math.sin(t),"sin(t)"]) #0

trig.append([math.cos(t),"cos(t)"]) #1

trig.append([math.tan(t),"tan(t)"]) #2

trig.append([(1/math.cos(t)),"sec(t)"]) #3

trig.append([-trig[0][0],"-sin(t)"]) #4

trig.append([-trig[1][0],"-cos(t)"]) #5

trig.append([-trig[2][0],"-tan(t)"]) #6

trig.append([math.sin(-t),"sin(-t)"]) #7

trig.append([math.cos(-t),"cos(-t)"]) #8

trig.append([math.tan(-t),"tan(-t)"]) #9

trig.append([(math.sin(t)/math.cos(t)),"sin(t)/cos(t)"]) #10

trig.append([(2\*math.sin(t/2)\*math.cos(t/2)),"2sin(t/2)cos(t/2)"]) #11

trig.append([math.pow(math.sin(t),2),"sin^2(t)"]) #12

trig.append([(1-math.pow(math.cos(t),2)),"1-cos^2(t)"]) #13

trig.append([((1-math.cos(2\*t))/2),"(1-cos(2t))/2"]) #14

trig.append([(1/math.cos(t)),"1/cos(t)"]) #15

return trig

def trigTest(trig):

"""

Builds an array that holds every combination of identities. It then uses

the combinations of to determine if they are true or not.

"""

test = []

equal = []

notEqual = []

tol = 0.0000000001

for u in range(16):

for v in range(u,16):

if u != v:

test.append([u,v])

for t in test:

if abs(trig[t[0]][0]-trig[t[1]][0]) < tol:

equal.append([t[0],t[1]]) #appends matrix indices of identities that are equal

else:

notEqual.append([t[0],t[1]]) #appends matrix indices of identities that are not equal

for n in notEqual: #prints each identities value and that they are not equal

print(trig[n[0]][1]," = ",trig[n[0]][0])

print(trig[n[1]][1]," = ",trig[n[1]][0])

print(trig[n[0]][1]," != ",trig[n[1]][1])

print("Tolerance: ",tol)

print()

for e in equal: #prints each identities value and that they are equal

print(trig[e[0]][1]," = ",trig[e[0]][0])

print(trig[e[1]][1]," = ",trig[e[1]][0])

print(trig[e[0]][1]," = ",trig[e[1]][1])

print("Tolerance: ",tol)

print()

def partition(S,n):

"""

If the sum of set S is not even, no partition can exist for the given set.

If the sum of set S is even, passes the set, the length of the set, and

half the sum of the whole set to the subset helper method.

The subset helper method returns true or false depending on if a

partition is found.

If the partition is found, prints that a partition does exist for set S

If the partition is not found, prints that a partition does not exist for set S

Then returns the boolean value on if a partition was found.

"""

k = sum(S)

if k%2 != 0:

print("No partition exists for set: ",S)

return False, []

subExists, s1 = subset(S,n,k/2)

if subExists:

print("Partition exists for set: ",S)

else:

print("Partition does not exist for set: ",S)

return subExists, s1

def subset(S,n,k):

"""

Base case 1: The sum is equal to 0: partition has been found, return true

Base case 2: The remaining number of elements that can be chosen is 0 and

the sum does not equal 0: return false

If the next number to be chosen would exceed the remaining sum, skip

Save the results of picking and not picking the next number.

If the result of picking the next number is true AND the result of not

picking the next number is false, appends the chosen number to a global

subset s1 and returns the result of picking the next number.

If the result of not picking the next number is true, does nothing and

returns the result of not picking the next number.

"""

if k == 0:

return True, []

if n<0 or k<0:

return False, []

se, ss = subset(S,n-1,k-S[n-1])

if se:

ss.append(n-1)

return True, ss

else:

return subset(S,n-1,k)

"""

Code for testing of trigonometric identities. Generates a number between -pi and pi,

then passes that to the trigBuild function, storing the resulting array in

trig, then passes that onto trigTest

"""

elapsed = 0

for i in range(10):

t = random.uniform(-math.pi,math.pi)

start = time.time()

trig = trigBuild(t)

trigTest(trig)

elapsed += time.time()-start

print (elapsed/10)

"""

Code for testing of partition problem. Uses both given sets S so that it can be

tested against a set that has a partition possible and a set that does not

have a partition possible.

Sorts the given set first.

Passes the set and length of the set to the partition method:

Receives the returned boolean of true if a partition is possible and false

if a partition is not possible.

If a partition is possible

Builds the two subsets by appending the values of the indices of S found in

the list of indices returned by partition into s1, and those indices

not found in the list to s2.

Prints subset 1 (s1).

Prints subset 2 (s2).

Then tests against a set that does not have a partition possible.

"""

elapsed = 0

n = 5

S=np.zeros(n,dtype=int)

for j in range(10):

for i in range(n):

S[i] = random.randint(0,n)

S.sort()

start = time.time()

sub, ss = partition(S,len(S))

if sub:

s1 = []

s2 = []

for i in range(len(S)):

if i in ss:

s1.append(S[i])

else:

s2.append(S[i])

print(s1)

print(s2)

elapsed += time.time()-start

print(elapsed/10)

"""

S = [2,4,5,9,13]

S.sort()

s1 = []

sub = partition(S,len(S))

if sub:

if (sum(S)/2) != 1:

s1.pop()

s2 = []

for i in S:

if i not in s1:

s2.append(i)

print(s1)

print(s2)

"""