Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: May 12, 2019

Instructor: Olac Fuentes

Assignment: Lab 8 Algorithm Design Techniques

TA: Anindita Nath & Maliheh Zaragan

**Introduction:**

In class, we were introduced into different Algorithmic Design Techniques to solve different problems using python. Now we are implementing these new algorithmic design techniques to solve two problems. The first is to check if two algorithmic identities produce the same value, hence they are the same, and to check the partition of an array of numbers. To solve the trigonometric problem, we are required to use a randomized algorithm, which is an algorithm that uses random elements in it to solve the problem of the algorithm. The problem using partition needs to implement backtracking, which is an algorithm that creates mock solutions to its problems until a defective one is found, in which then it is deemed invalid, and therefore, unable to complete.

**Proposed solution design and implementation:**

1. Trig. Identity (Part 1): This part of the lab could potentially be easy, as I am hoping to just select a random number, and if two identities have the same value, then it will print that specific combination to the user. The solution might be more difficult later on, but I want to experiment with these procedures until I find an alternative.
2. Partition (Part 2): For this problem, all I need to do is built a method that takes an array, and then splits the set into two subsets to compare. If the statements provided for the lab are true, then it will return the two subsets needed for that partition**,** if the conditions are not met however, then the error message will appear. For the process into turning into two subsets, it would be ideal to first find if there are any subsets to even create in the first place, examples are if the subset contains nothing. If there are sets inside the array, the program should then be able to find whenever these sets are partitions are true or not through conditional statements.

**Experimental results**:

1. Trig. Identity (Part 1): As expected, I found a random number x, and replaced it with each trig identity, afterwards, one by one, the program will compare the different trigonometric identities, and whichever are the same set, it will print to the user like my example below. I have no hard code for the variable being chosen, as it would defeat the purpose of the number being random. No matter which integer it gets, the results for part 1 will always be the same. It also finds the number of times the similarities have been found in these trig identities.
2. Partition (Part 2): The solution to this problem was to split the partition into the two subsets, using only two algorithms. The first is called ‘arrayPartition’ which will handle the actual splitting of the array given, and ‘subest\_summation’, which will actually calculate if the partition will be able to exist within the given array. The partition process is being shown with the example given from the lab handout, the one printed below is from an existing partition, however, if the 12 changes to a 13 for example, then it will print that the partition does not exist.

The results of running the search for the trig. Identities as well as a successful partition is shown below:

A screenshot of a social media post

Description automatically generated

**Running times Table**

|  |  |  |
| --- | --- | --- |
| Attempt | Part 1 | Part 2 |
| 1 | 4.203525 sec | 0.0 sec |
| 2 | 3.4267202 sec | 0.0 sec |
| 3 | 3.121821 sec | 0.0 sec |

**Conclusions**:

With this lab, I was able to learn to code better using the Python language, including using algorithms to find solutions to a trigonometric identity, as well as a partition from an array of numbers. I was also able to learn to solve different problems by using randomized and backtracking algorithms, as learned in class, to determine my the solutions to the problems of this lab.

**Appendix :**

**Lab8.py**

"""

Author: Carlos Fernando Castaneda

Class : CS 2302

Date Modified: May 12, 2019

Instructor: Olac Fuentes

Assingment: Lab 8 Algorithm Design Techniques

TA: Anindita Nath & Maliheh Zaragan

Purpose: to implement both randomized algorithms and backtracking teachniques

learned in class to check if two algorithmic identities are the same, and to

check the partitions of a new array.

"""

#Imports various tools to help us calculate the hash tables to be used in this lab

import time

import random

import mpmath

import numpy as np

#Method that goes through all of the strings of a givrn list, and and checks if they are similar in value

def similarities(S):

#Starts a counter to keep track of all the

count=0

#For i in range of the length of S

for i in range(len(S)):#goes through all the strings

#For i in range of the length of S

for j in range(i,len(S)):

#If S[i] is equal to S[j], then it will print both items, and add one to count

if(same\_values(S[i],S[j])):

print(S[i],S[j])

count+=1

#Returns the value of count to the user

return count

#Method that calculates if two strings are similar in value to each other,=.

def same\_values(string\_1, string\_2,calls=1000,tolerance=0.0001):

#For i in the range of calls

for i in range(calls):

#Assigns a random number to variable x

x = random.uniform(-mpmath.pi,mpmath.pi)

#Sets a new number value1 which takes the information from string\_1, and evaluates it

value1 = eval(string\_1)

#Sets a new number value2 which takes the information from string\_2, and evaluates it

value2 = eval(string\_2)

#If the absolute value of value1 - value2 is greater than the tolerance value, then it returns false

if np.abs(value1-value2)>tolerance:

return False

#Returns true if the statement abive is incorrect

return True

#Method that checks if apartion can be made from the two parts of S

def arrayPartition(S1,S2):

#If the sum of S1 % by 2 is not 0, then there is no partition

if sum(S1)%2!=0:#if summation of sum is odd then return error message

return "No partition exists"

else:

#Creates a set needed for the next section

res,s,= subset\_summation(S1,len(S1)-1,sum(S1)//2)

#If the length of s equals 0, then there is no partition

if len(s)==0:

return "No partition exists"

#For every i in s

for i in s:

#New counter is created used to get the position

counter=0

#For every j in S1

for j in S1:

#If the value of i equals the value of j, then S1 pops a value

if i == j:

S1.pop(counter)

#Adds one to the counter

counter+=1

#Returns the value of s and S1

return s,S1

#Method that creates a new subse

def subset\_summation(S,last,goal):

#If the value of goal equals 0, then it returns true with a new blank array

if goal == 0:

return True, []

#If the value of goal is less than or greater than 0, then it retrens false with a new blank array

if goal<0 or last<0:

return False, []

#Takes a new subset

res, subset = subset\_summation(S,last-1,goal-S[last])

#If res is true, then it will append S[last and retrun true with the subset

if res:

subset.append(S[last])

return True, subset

#Otherwise, it will not take S[last from the list and move on

else:

return subset\_summation(S,last-1,goal)

#Starts the timer for the running time for part 1

startTime1=time.time()

print('Importing algorithim equations to test: ')

print()

#Creates a new array called 'part1' which will import all of the functions that will be compared its equalities

part1=['mpmath.sin(x)',

'mpmath.cos(x)',

'mpmath.tan(x)',

'mpmath.sec(x)',

'-mpmath.sin(x)',

'-mpmath.cos(x)',

'-mpmath.tan(x)',

'mpmath.sin(-x)',

'mpmath.cos(-x)',

'mpmath.tan(-x)',

'mpmath.sin(x)/mpmath.cos(x)',

'2\*mpmath.sin(x/2)\*mpmath.cos(x/2)',

'mpmath.sin(x)\*\*2',

'1-mpmath.cos(x)\*\*2',

'(1-mpmath.cos(2\*x))/2',

'1/mpmath.cos(x)']

#The actual method t

sim\_count = similarities(part1)

#Prints the count number found in the method similarities

print()

print('The number of similarities in the equations are a total of: ', sim\_count)

print()

#Ends the timer for the running time for part 1

endTime1=time.time()

#Creates the fianl time for the running time for part 1

finalTime1 = endTime1-startTime1

#Starts the timer for the running time for part 2

startTime2=time.time()

#Creates a new array of integeres needed for part 2 of the lab

part2=[2,4,5,9,12]

print('Partition process using array', part2, 'commencing: ')

print()

#Sends the new array to method arrayPartition

print(arrayPartition(part2,part2))

print()

#Ends the timer for the running time for part 2

endTime2=time.time()

#Creates the fianl time for the running time for part 2

finalTime2 = endTime2-startTime2

#Prints the running times of both part 1 and part 2

print('Running time for Part 1 in: ',finalTime1)

print('Running time for Part 2 in: ',finalTime2)

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.

