Joaquin Hidalgo

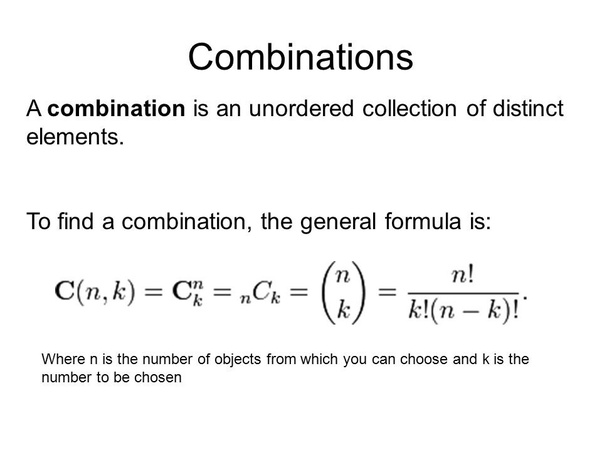
CS 2302 MW @1:30pm

Lab 08

**Algorithm Design Technique**

**Introduction:** The first problem to the first part of the lab is that we need to design our code such that we do not need to compare more than necessary times each identity while still having a high success rate. We evaluate two equations with a random variable for ‘t’ or ‘x’ from – pi to pi and subtract the absolute difference and in order for it to be an identity, it needs to be lower than the tolerance variable (.001) and be tried multiple times (1,000) for my design. Then print all identities with last absolute difference.

The second part of the lab, the problem is to use the backtracking technique to determine if there’s 2 subsets of the original set that is ordered from min to last such that subset 1 = S1 and subset 2 = S2. The 2 subsets must follow these guidelines, S1 intersection S2 is empty set. S1 union S2 is the whole set. The sum of S1 set must equal, the sum of S2. Print the 2 subsets if possible and if not, display that it’s not possible with that set.

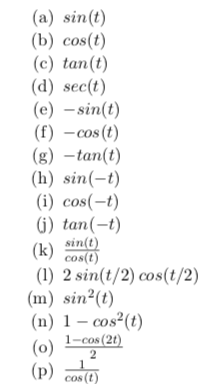
**Proposed solution design and implementation:** To discover all possibilities of all trig identities of a given list (‘a’ – ‘p’). Here to test if each equation is in fact an identity, we need a tolerance level (.001) because the input is a random float from (-pi to pi) and two equations can have an absolute difference of 0 or a number really close to 0 like example: (0.000032738). Anything lower than the tolerance level will count as an identity to be considered a trig identity. Then we will use a certain amount of tries (1,000) to ensure our design has a big successful rate. To compare each identity the least about of times we use combinations . All variables listed above in parenthesis ‘()’ is my UX design/interface choices.

The second part of the lab, I found out that if the sum of the list which is the original set is not an even number, there is not a possible 2 subsets that follow the lab guidelines but in specially, sum of set S1 does not equal sum of S2 set. Next, I determine what elements equally to half the sum of the original set. I do this by using the subsetsum algorithim which recursively, has a base case when the goal is equal to 0, we return True and empty set. The equal variable is (half the sum of the original set), so that the sum of S1 equals, S2. If the goal is less than 0 or last variable (the last index of the set) is less than 0, we return false and an empty set. Then we recursively call the algorithm again by subtracting the last index by 1 (last-1) and subtracting the goal from the element in the position of last, (goal-S[last]) where (S) is the set. If the base case is True, we return True and append the element and the position of the element to a global list called, (subset) and (subset1\_index). If bae case was false, we return the recursive call by NOT subtraction the goal from the current index position ((goal-S[last])) and actually subtracting the index by 1 (last-1).

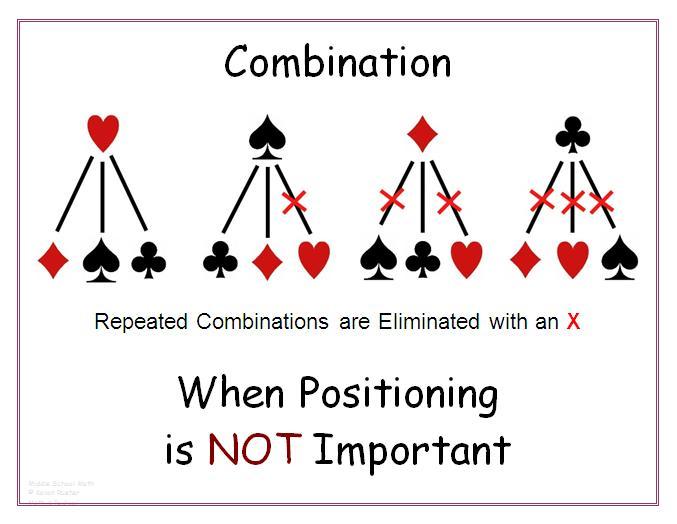
At this point we now have a (subset) and a Boolean (res), True, if there is a subset list that the sum is half the sum of the original set, as well as the index positions of the elements of this subset list. So, then we iter thru the whole original set and add up every element that is not in the 1st subset which is why I save the positions of the elements in the first subset. Now we have the sum of the 2nd subset and if that sum equals the sum of the first subset then we have two subsets that are the answer and display that.

**Experimental results:**

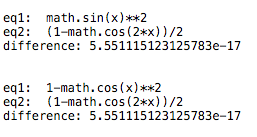
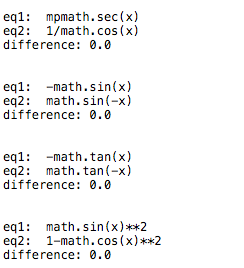
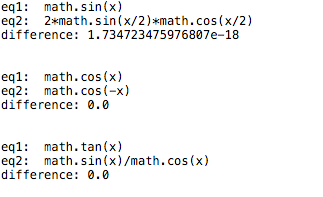
**Full list of trig identities to test:**

****

**Why use combination??:**

**Algorithmic design --🡪**

**Lab Part 1 display of report above:**

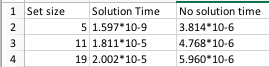
****

**Lab Part 2 display of report above:**

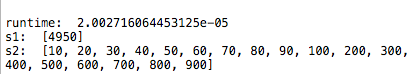
****

**w/ given set -🡪 **

**Running times:**

****

**Example trial:**

****

****

**Conclusion:** In conclusion, I found out how to write a program to do my algebra homework and test trig identities with a high success rate, fast run time and full randomized for best results. I also learned what combinations mean which I included a picture in the experiential design section (above^^). I also learned how to evaluate a string function with a randomized input from -pi to pi. In the second part of the lab, I learned how to determine if there two subsets of an original subset that follow the guidelines of the lab. I now know how to determine what the subset list of the original set is by using recursion. The subsetsum recursive method, starts at the end of the list that is sorted from small to large and keeps on subtracting the index by 1 and subtract the goal from the element in the (last) position while appending the correct elements to the list to later display.

**Code:**

﻿"""

By: Joaquin Hidalgo

Lab 08

Created on Tues 7 May 18:32:21 2019

Instructor: Olac Fuentes

TA: Dita and Mali

Last modified Fri 10 May 23:29:11

Use combinations to test trig identies by computing random floats for -pi to pi.

Then evaulation a pair of identities and subtract the differnce of the two functions,

get the absolute value of the differnce and if differnce is above tolerance, we can say that they are not idetities versus if less than tolerance, then we do that identity until 'tries' to ensure it wasnt luck.

We use backtracking method to find the subset of a list of integers and partion

each set thus in the end, we have two subsets of the set such that,

s1 union s2 is Set and s1 intersection s2 is the empty set and the sum of s1 == sum of s2.

print and display the two sets if possible

"""

import random

import math

import numpy as np

from itertools import combinations

import mpmath

import time

#Takes in a list 'eq' of all possible combinations in pairs of all identies

#Returns nothing but prints all trig identities found in 'eq'

def Test\_Identities(eq, tol, tries):

for i in eq:

#Flag keeps track if 2 identities are IN-EQUAL

flag = False

trig1,trig2 = i

for j in range(tries):

x = random.uniform(-math.pi,math.pi) # generates random num for -pi to pi

y1= eval(trig1)

y2 = eval(trig2)

difference = np.abs(y2-y1) # take difference of two equations/identites

#Checks if differnece is greater than tolerance

if difference > tol:

flag = True

#if flag == True, break from this speficic pair identity

if flag:

break

if flag == False:

print('eq1: ', trig1)

print('eq2: ', trig2)

print('difference:',difference)

print('\n')

def subsetsum(S,last,goal):

global subset1\_indexs # set global subset1 index's

#base casre

if goal ==0:

return True, []

if goal<0 or last<0:

return False, []

res, subset= subsetsum(S,last-1,goal-S[last]) # Take S[last]

if res:

#Append indicies and actual elements of set

subset1\_indexs.append(last)

subset.append(S[last])

return True, subset

else:

return subsetsum(S,last-1,goal) # Don't take S[last]

def GetPartion(S, sumOfSet):

# if sumOfSet%2 == 1 then no possible way to partion set into 2 sets that fit guidelines

if sumOfSet % 2 != 0:

return False, []

#Use backtracking

res, subset = subsetsum(S, len(S)-1, sumOfSet//2)

return res, subset

def GetTwoSubSets(S, sumOfSet, subset, res):

global subset1\_indexs

sub1 = []

sub2 = []

#If res == false then no possible partion

if res == False:

return sub1, sub2

c = 0

#iter thru set and every indice thats not in set1 add to 'c'

for i in range(len(S)):

if i not in subset1\_indexs:

c+= S[i]

#Checks subset 1 to subset 2, if equal then append coresponding elements to subset1 and subset2

if c == sum(subset):

for j in range(len(S)):

if j in subset1\_indexs:

sub1.append(S[j])

else:

sub2.append(S[j])

return sub1, sub2

#Main

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

#Makes combo\_eq have EXACTLY one pair of each identity to be compares

combo\_eq = combinations(['math.sin(x)', 'math.cos(x)', 'math.tan(x)', 'mpmath.sec(x)', '-math.sin(x)','-math.cos(x)', '-math.tan(x)', 'math.sin(-x)', 'math.cos(-x)', 'math.tan(-x)', 'math.sin(x)/math.cos(x)', '2\*math.sin(x/2)\*math.cos(x/2)', 'math.sin(x)\*\*2', '1-math.cos(x)\*\*2', '(1-math.cos(2\*x))/2', '1/math.cos(x)' ], 2)

tol = .001 #This defines tolerance to decide if identity is equal

tries = 100 # The number of tries each pair of identities is tested

setNum = [2, 4, 5, 9, 12] #Given set to partition

sumOfSet = sum(setNum)

subset1\_indexs = [] # contains the indexes of the 1st subset to ensure s1 != s2

Test\_Identities(combo\_eq, tol, tries) # test and print ONLY trig identities

startBackTracking = time.time()

res, subset = GetPartion(setNum, sumOfSet) #res == True if possible to partion two sets, if so, subset contains nums of s1

subset1, subset2 = GetTwoSubSets(setNum, sumOfSet, subset, res) # obtains if possible both sets of the partion

endBackTracking = time.time()

# If subset1 U subset2 == All Set

# If subset1 n subset2 == Empty Set

if res:

print('runtime: ', endBackTracking - startBackTracking)

print('s1: ', subset1)

print('s2: ', subset2)

else:

print('No partition exists')

**Standards of Conduct and Academic Dishonesty**

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.

* Joaquin Hidalgo