Lab 8 report

Nicole Favela

CS 2302

May 13, 2019

**Introduction:**

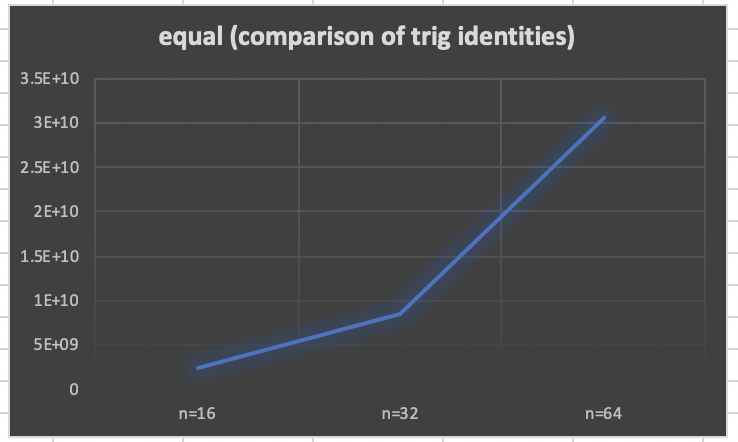
The purpose of this assignment is to utilize randomization to verify the equality of trigonometric identities and to use backtracking to solve a partition sum problem as described in part 2. I modified my program to test every possible combination of trig identities provided. I used the algorithm design code provided on the website to accomplish this task of testing trig identities and creating partitions of S whose sum is equivalent.

**Proposed solution and implementation:**

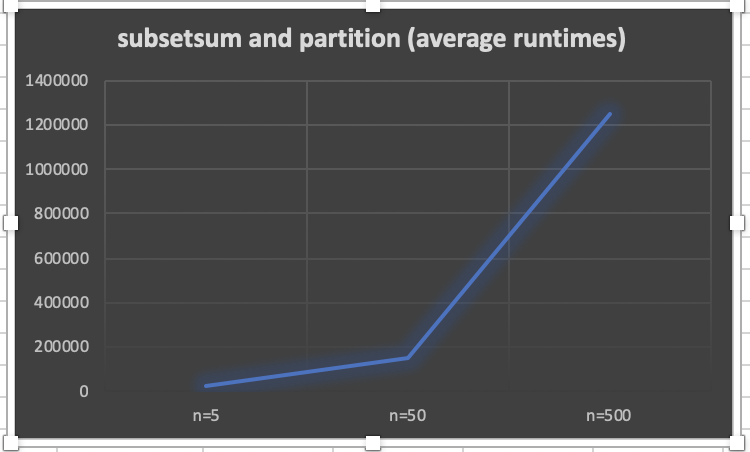
For part 1, I wanted to make sure I got every possible combination of values and checked equality using the function “equal”. Equal uses randomization in order to test many values using a for loop to determine if 2 functions f1 and f2 are equivalent. I accomplished this using a nested for loop to iterate through the list of trig expression provided and checked 2 at a time. I then print only the expression that are equal. In order to verify and test every combination I check obvious identities (for example that sin(t) = sin(t)). At first, I was unable to run the program as given without importing mpmath instead of just math. This allowed me to use functions like secant. From there, the function equal tries random numbers between -pi and pi 1000 times with a tolerance of 0.0001. If the absolute value of the difference between f1 and f2 is greater than the tolerance, the function returns false. For part 2, I created another function called partition. Partition acts as a kind of helper function that first checks if a partition is possible at all. It does this by making sure the sum of the set S is even, if it is, it divides the sum by 2 and returns the goal for use in subsetsum. Subsetsum remained the same as given in algorithm design code. When it is determined by partition that a goal is possible, subset sum is called. Subsetsum uses backtracking in order to select items from the end of the list and subtract them from the goal or in the case if res is false, does not take the last item from the list. If res is true it appends it to subset and returns true. The base cases are if the goal is 0, which means that there was nothing to determine and returns true. In the case that goal or last are negative, it simply returns false and an empty list. At the end of the function, if a subset was possible, I print it (called s1) and create the other subset (a partition s2) by adding to s2 the items that are not in s1. In this way, I ensure no items are repeated in both sets i.e. their intersections are {} and that their union is S (the original set).

**Runtimes and samples outputs:**

For my sample runtimes I tested my methods each with varying sizes of n. For subsetsum and partition, I increased the set S by a factor of 10 each time. I tested sets of 5, 50, and 500 using an np array of random integers with those sizes. In order to test partition and subsetsum, I included both in my runtime and ran the operations in a for loop 10 times. I then took the average of those runtimes to come up with my runtime for part 2. For my trig comparisons and function “equal”, I doubled the size of the list using .extend every time but kept the values the same. All runtimes were captured in nanoseconds. A graph of the runtime of the function equal comparing the list of trigonometric identities (in all possible combinations) is shown below.

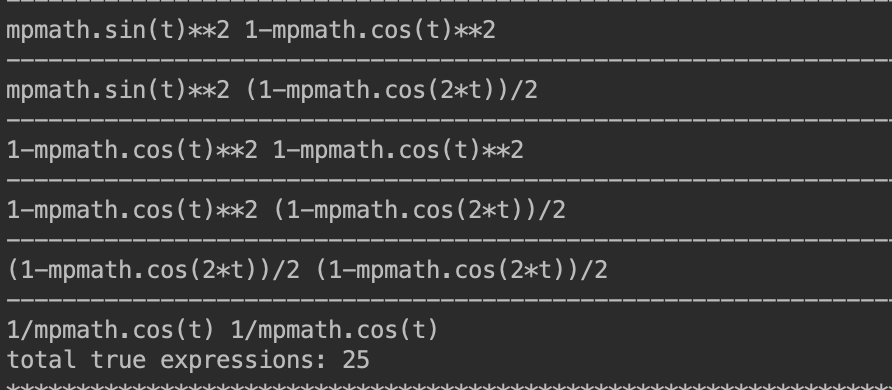


A graph of the runtimes of partition and subset sum is shown below:



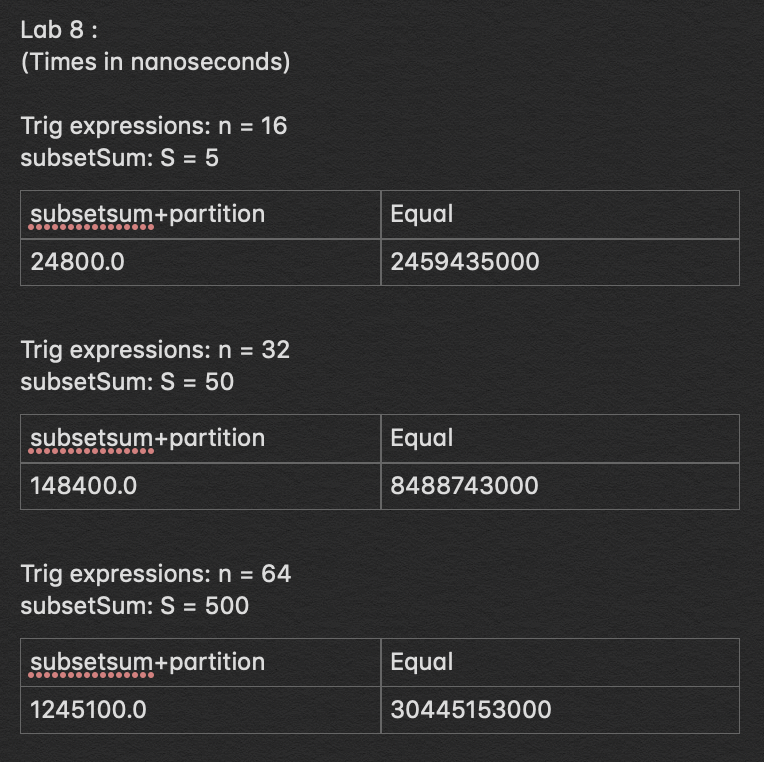
Note that the graph of this functions resembles a graph of 2^n (exponential function).

Below is a sample console output for my program with standard input of 16 trig identities and a set S of size 5.

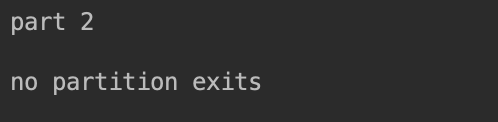




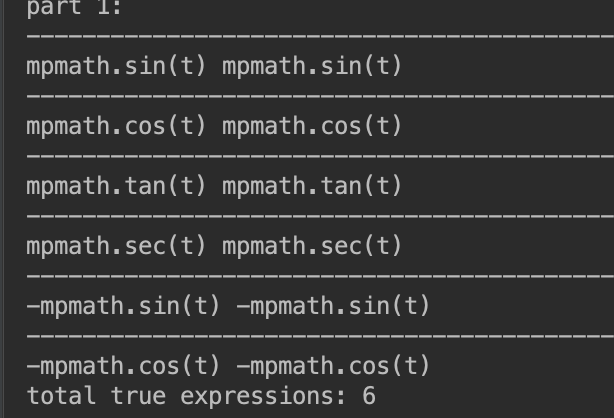
Below is a table of values:



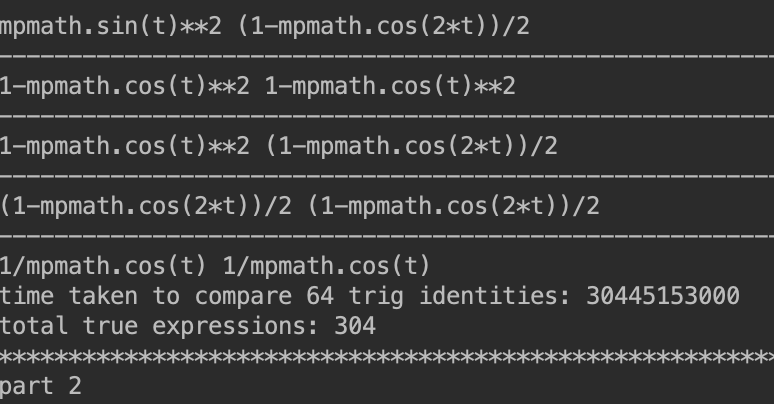
Below is a test output for partition and subset sum when sum of S is odd given S = [2, 4, 5, 9, 12, 1].



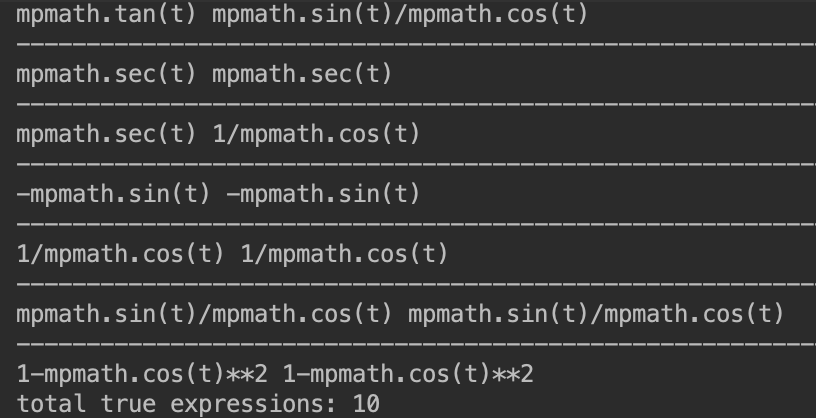
The following test was on the list of trig functions of size 6. T = ['mpmath.sin(t)','mpmath.cos(t)','mpmath.tan(t)','mpmath.sec(t)','-mpmath.sin(t)','-mpmath.cos(t)']



Test case where size of T is 64:



Where T = ['mpmath.sin(-t)','mpmath.tan(t)','mpmath.sec(t)','-mpmath.sin(t)','1/mpmath.cos(t)','mpmath.sin(t)/mpmath.cos(t)','1-mpmath.cos(t)\*\*2']



**Conclusion:**

This project taught me how use backtracking and randomization. Randomization is in this context is used in order to find, with high probability, that a solution is correct. Randomization is also useful in other applications such as quicksort. For example, choosing a pivot randomly. Backtracking uses exhaustive search in order to find the set that will equal a certain amount or goal. Although backtracking is not particularly time efficient (runs in 2^n time in the worse case), it is able to come up with the correct solution by taking an item or not taking an item depending on whether or not that choice was beneficial.

**Appendix source (source code):**

#CS2302  
#Nicole Favela  
#last modified: May 9, 2019  
#Lab8  
#purpose: to use randomized algorithms to test the equality of  
# trigonometric expressions and use backtracking to determine if  
# there is a valid partition of sets that equal the same sum  
#instructor: Olac Fuentes  
#TAs: Anindita Nath and Maliheh Zargaran  
import random  
import numpy as np  
import mpmath  
  
#used to determine if trig expressions are equal  
#used in part 1 to test trig expressions  
def equal(f1, f2,tries=1000,tolerance=0.0001):  
 for i in range(tries):  
 t = random.uniform(-(mpmath.pi), mpmath.pi)  
 #x = random.random()  
 y1 = eval(f1)  
 y2 = eval(f2)  
 if np.abs(y1-y2)>tolerance:  
 return False  
 return True  
#used for part 2  
def subsetsum(S,last,goal):  
 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:  
 subset.append(S[last])  
 return True, subset  
 else:  
 return subsetsum(S,last-1,goal) # Don't take S[last]  
#used for part 2  
def partition(s1):  
 #checks if sum of s1 is odd, if it is, a partiton with equal sums is not possible  
 if sum(s1)%2 != 0:  
 return False  
 #creates the goal  
 goal = sum(s1)//2  
 return goal  
  
print()  
print('part 1:')  
T = ['mpmath.sin(t)','mpmath.cos(t)','mpmath.tan(t)','mpmath.sec(t)','-mpmath.sin(t)','-mpmath.cos(t)','-mpmath.tan(t)','mpmath.sin(-t)','mpmath.cos(-t)','mpmath.tan(-t)','mpmath.sin(t)/mpmath.cos(t)',  
 '2\*mpmath.sin(t/2)\*mpmath.cos(t/2)','mpmath.sin(t)\*\*2','1-mpmath.cos(t)\*\*2','(1-mpmath.cos(2\*t))/2','1/mpmath.cos(t)']  
count = 0  
for i in range(len(T)):  
 for j in range(i,len(T)):  
  
 if(equal(T[i], T[j])):  
 print('-----------------------------------------------------------------------------------------------------------------------------------------')  
 print (T[i],T[j])  
 count+=1  
print('total true expressions:',count)  
  
print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
#part 2  
print('part 2')  
print()  
  
#set S  
S = [2, 4, 5, 9, 12]  
#p is the goal or false if no partiton possible  
p = partition(S)  
  
S2=[]  
  
#returns boolean value res and list s1 which is 1/2 sum of set  
res, s1= subsetsum(S,len(S)-1,p)  
  
if p:  
 print('s1',s1)  
 #creates other partition set  
 for j in S:  
 if j not in s1:  
 #adds values not in s1 to complete partition  
 S2.append(j)  
 print('s2',S2)  
else:  
 print('no partition exits')

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.”

-Nicole Favela