The purpose of Lab 8 was to write a program utilizing a randomized algorithm that detects equalities from a given list of trigonometric identities and another program that uses backtracking to determine if there is a way to partition a set of integers into two subsets. This lab is divided into these two parts and each consists of two functions and the main. For part 1, the randomized algorithm consists of generating a number from negative pi to pi and compare each of the trigonometric identities several times generating a different value to use with the trig identities each time. If in each test two identities match in every test, this means these two trig identities are equal. For part 2, the backtracking algorithm will check different partitions of the set and compare each subset to find out if they are equal.

To check of the equalities in the trigonometric identities, each identity is placed on a list of strings called trig\_id. They are saved as strings and when evaluated will use the math library to compute each one. The list along with the length of the list, n, are sent to comparisons. This function is where the list now called trig is dissected. A while loop will store the first identity in the list to f1 and remove it from the list. Then a for loop will get the second element in the list, now the first element, and store it in f2. These two variables are sent to are\_equal for comparisons along with the length of the original list. In are\_equal there is a for loop that will run n times. This will make each comparison of f1 and f2 be made n times. The variable t is the random number generated in the range of negative pi to pi. This variable changes each time to make sure that each comparison of f1 and f2 are truly identical and not just in lets say, t equals 1. Several if statements are used because some of the trig identities are saved differently in the trig list from the way the math library can read them. For example, sin^2(t) means sin to the power of 2 t. The math library needs this to be typed as math.pow(math.sin(t),2) for it to do what we are asking. This is done all the identities that include a power. For sec(t) we use the mpmath library because secant isn’t part of the math library. Therefore, an if statement is used for sec(t). Once all these if statements are checked for f1 and f2, these two are evaluated and rounded to seven decimal points. This is done to avoid any issues caused by the float type since the answers will consist of several decimal places. These two answers are compared and if equal True is returned, else a False is returned.

How the while loop and for loop are constructed in comparisons, this will make f1 to be compared will all the trig identities in the list since the for loop is inside the while loop. This is the reason why once the trig identity is stored for f1 it is removed from the list. This avoids statements like, sin(t) is equal to sin(t). Once the list is traversed and compared each f1 and f2 n times, the trig identities that are equal are shown to the user along with the total number of equalities from the list of trigonometric functions. The following is the output from running the code:

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

sin(t) is equal to 2\*sin(t/2)\*cos(t/2)

cos(t) is equal to cos(-t)

tan(t) is equal to sin(t)/cos(t)

sec(t) is equal to 1/cos(t)

-sin(t) is equal to sin(-t)

-tan(t) is equal to tan(-t)

sin^2(t) is equal to 1-cos^2(t)

sin^2(t) is equal to (1-cos(2\*t))/2

1-cos^2(t) is equal to (1-cos(2\*t))/2

There are a total of 9 equalities from 16 trigonometric functions. Time elapsed = 11.627685546875 milliseconds

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

sin(t) is equal to 2\*sin(t/2)\*cos(t/2)

cos(t) is equal to cos(-t)

tan(t) is equal to sin(t)/cos(t)

sec(t) is equal to 1/cos(t)

-sin(t) is equal to sin(-t)

-tan(t) is equal to tan(-t)

sin^2(t) is equal to 1-cos^2(t)

sin^2(t) is equal to (1-cos(2\*t))/2

1-cos^2(t) is equal to (1-cos(2\*t))/2

There are a total of 9 equalities from 16 trigonometric functions. Time elapsed = 13.749309211726 milliseconds

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

sin(t) is equal to 2\*sin(t/2)\*cos(t/2)

cos(t) is equal to cos(-t)

tan(t) is equal to sin(t)/cos(t)

sec(t) is equal to 1/cos(t)

-sin(t) is equal to sin(-t)

-tan(t) is equal to tan(-t)

sin^2(t) is equal to 1-cos^2(t)

sin^2(t) is equal to (1-cos(2\*t))/2

1-cos^2(t) is equal to (1-cos(2\*t))/2

There are a total of 9 equalities from 16 trigonometric functions. Time elapsed = 13.153892438533 milliseconds

/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

It is no surprise that it takes around 11 to 13 milliseconds in these three runs for this section of the code to complete. After all the running time in big O notation is O(n^2) for the function comparisons and O(n) for are\_equal.

Next is the backtracking code to solve for any subsets within a set can be partitioned. Two different lists are presented to the user and the user can select which on of the lists they would like to use. The first one can be partitioned while the other can’t. Once a set is selected this is sent to the function partition along with an empty list and the length of the partition minus 1. The length of the partition is subtracted by one to be used within the function as the last element in the selected subset now called S1. S2 is the empty list and last is the name of the length of the partition subtracted by 1. There are a series of three if statements and an else statement that will make some comparisons. The first two if statements are the base case and the last if and else are the recursive case. The first if statements compares if last is less than 0. If this is true, then this means that all combinations for different subsets have been made and there isn’t a partition in the set so False is returned along with the last combination of subset one and subset 2. The next if statement checks if the subset S1 is equal to subset S2. Subset is a function that receives a subset and returns the sum of the subset or returns a zero if the length of the subset is zero. This means that a subset is empty and since we are dealing with integers, we can’t use False or None since at least one subset will be an integer. If this if statement is true, then True is returned to main along with S1 and S2. The last if statement checks if the sum of S1 is less than the sum of S2. The subset function is used on this if statement as well and if it is true then the last element from S2 is added to S1 and removed from S2. If this is not the case, then the last element from S1 is added to S2 and removed from S1. This is what the else statement is used as well as the last variable. Once these operations are done the recursive call is made for partition sending S1, S2 and last subtracting 1. This recursion call will be made until one of the base cases is true.

Once partition returns either True or False and S1 and S2, a new variable is created in the main called set\_list which combines the subset s1 and s2, formerly S1 and S2, and sorts them in ascending order. This is done because the original set has been messed with and thus don’t include all the integers that where originally in the set. Finally, there is an if and else statement that check whether partition returned True or False. If True, the user is shown there is a partition in the set, now called set\_list, and that they are s1 and s2, along with how long it took for the system to run. If it returned False, then it shows the user that there isn’t a partition in the set and the time it took the system to run. The following are the results of running part 2 of the code:

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

Select 1 for S = [2, 4, 5, 9, 12] or select 2 for S = [2, 3, 5, 8, 13]: 1

This set: S = [2, 4, 5, 9, 12] has a partition [2, 5, 9] [12, 4] . Time elapsed = 0.0 milliseconds

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

Select 1 for S = [2, 4, 5, 9, 12] or select 2 for S = [2, 3, 5, 8, 13]: 2

No partition exists for: S = [2, 3, 5, 8, 13] . Time elapsed = 0.0 milliseconds

//////////////////////////////////////////////////////////////////////////////////////////////////////////////

As seen, there isn’t much time used for running this backtracking algorithm since it is done in 0 milliseconds. Even though there is a recursive equation in the function partition, it doesn’t affect the running time that much due to the rest of the code in part 2 consisting mainly of if or else statements.

For part 1, the comparisons function takes a running time of O(n^2) without taking into consideration the function call to are\_equal. The function are\_equal has several if statements but that doesn’t affect the running time. The only line that affects this is the for loop that runs n times. This means that the running time of are\_equal is O(n). For part 2, while the running time is O(2^n) it is heavily dependent on n being the amount of comparisons that are to be made.

What I learned for this lab was how different algorithms can be used to find the solution to a problem. Also, how these algorithms can affect the running time and how these algorithms can be beneficial in certain cases while being detrimental in others due to affecting the running time greatly. I personally struggled the most using the backtracking method. This was because it took me time to identify which base cases to use for the recursive calls. I personally liked the randomized algorithm since it can use random numbers to find equalities making the equalities found very precise.

Appendix:

# -\*- coding: utf-8 -\*-

"""

Created on Tue May 7 10:45:10 2019

@author: Gerardo Armenta

Instructor: Dr. Olac Fuentes

TA: Anindita Nath, Maliheh Zargaran, Erik Macik, Eduardo Lara

Purpose: part1: To write a program that discovers trigonometric identities using a randomized

algorithm to detect equalities using random numbers in range of -pi to pi.

part2: To write a program that determines if there is a way to partition a set of

integers into two substes using backtracking.

"""

import random

import math

import mpmath

import time

'''

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

PART 1

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

'''

# Checks equality for all trig functions in main with different t values

def are\_equal(f1, f2, n):

# There are n (length of trig functions list) number of times trig functions

# are checked for equalities.

for i in range(n):

t = random.uniform(-math.pi,math.pi) # t varies per each test

# The following functions aren't recognized by the math libary used

# so they are changed to be recognized by math and mpmath (for sec(t)).

if f1 == 'sin^2(t)': # sin(t) to the power of 2

f1 = 'math.pow(math.sin(t),2)'

if f1 == '1-cos^2(t)': # 1 - cos(t) to the power of 2

f1 = '1-math.pow(math.cos(t),2)'

if f1 == 'sec(t)': # sec not recognized by math, used mpmath

f1 = 'mpmath.sec(t)'

if f2 == 'sin^2(t)': # sin(t) to the power of 2

f2 = 'math.pow(math.sin(t),2)'

if f2 == '1-cos^2(t)': # 1 - cos(t) to the power of 2

f2 = '1-math.pow(math.cos(t),2)'

if f2 == 'sec(t)': # sec not recognized by math, used mpmath

f2 = 'mpmath.sec(t)'

# Values are rounded to the 7th decimal point to avoid any float issues.

y1 = round(eval(f1), 7)

y2 = round(eval(f2), 7)

if y1 != y2: # if results after evaluation are different, returns false

return False

return True # returns true if comparisons of trig identities are equal

# Sets the comparisons for the trig identities and shows equalities to the user.

def comparisons(trig, n):

start = time.time()\*1000

count = 0 # Counter used to show total of equalities found to user.

while len(trig) > 0: # gets the first element of the list for f1

f1 = trig[0]

trig.pop(0) # removing element 0 keeps while loop moving and assures f1 and f2 aren't the same

for function in trig: # loop used to set f2 and check for equality

f2 = function

if are\_equal(f1,f2,n) is True: # if equality found lets user know

count += 1 # counter adds 1 every time equality is found

print('\n',f1, ' is equal to ', f2)

stop = time.time()\*1000

print('\n There are a total of ', count, ' equalities from ', n, ' trigonometric functions. Time elapsed = ', stop-start, 'milliseconds')

'''

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

PART 2

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

'''

# adds the sum of a given subset

def subset(s):

if len(s) == 0: # used to return 0 if list is empty

return 0

a = sum(s)

return a

def partition(S1,S2,last):

if last < 0: # once last is less than 0 there are no more subsets to be compared

return False,S1,S2

if subset(S1) == subset(S2): # if two subsets are equal, a partition exists

return True,S1,S2

if subset(S1) < subset(S2): # if S1 is less than S2, the last value added to S2 is returned to S1 and removed from S2

S1.append(S2[-1])

S2.remove(S2[-1])

else:

S2.append(S1[last]) # if sum of S1 is more than sum of S2, last value is removed from S1 and added to S2

S1.remove(S1[last])

return partition(S1,S2,last-1) # New comparisons are made with changes to subsets S1 and S2 making last subtrated by one to traverse lists

# list of trig identities

trig\_id = ['sin(t)', 'cos(t)', 'tan(t)', 'sec(t)', '-sin(t)', '-cos(t)', '-tan(t)', 'sin(-t)', 'cos(-t)', 'tan(-t)', 'sin(t)/cos(t)', '2\*sin(t/2)\*cos(t/2)', 'sin^2(t)', '1-cos^2(t)', '(1-cos(2\*t))/2', '1/cos(t)']

comparisons(trig\_id, len(trig\_id)) # will check for equalities in trig\_id list

# asks user to select S list

selection = (int(input('Select 1 for S = [2, 4, 5, 9, 12] or select 2 for S = [2, 3, 5, 8, 13]: ')))

if selection == 1:

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

else:

S = [2, 3, 5, 8, 13]

start = time.time()\*1000

is\_set,s1,s2 = partition(S, [], len(S)-1)

set\_list = sorted(s1+s2) # s1 + s2 will create original S list in ascending order

stop = time.time()\*1000

if is\_set is True: # if a partition is found

print('\n This set: S =', set\_list, ' has a partition ', s1, ' ', s2, '. Time elapsed = ', stop-start, 'milliseconds')

else: # if no partition is found

print('\n No partition exists for: S =', set\_list, '. Time elapsed = ', stop-start, 'milliseconds')

Academic dishonesty includes but is not limited to cheating, plagiarism and collusion. Cheating may involve copying from or providing information to another student, possessing unauthorized materials during a test, or falsifying data (for example program outputs) in laboratory reports. Plagiarism occurs when someone represents the work or ideas of another person as his/her own. Collusion involves collaborating with another person to commit an academically dishonest act. Professors are required to - and will - report academic dishonesty and any other violation of the Standards of Conduct to the Dean of Students.