David Amparan

May 13, 2019

Instructor: Dr. Fuentes, Olac

TA: Anindita Nath, Maliheh Zargaran

Lab 8

**Introduction**

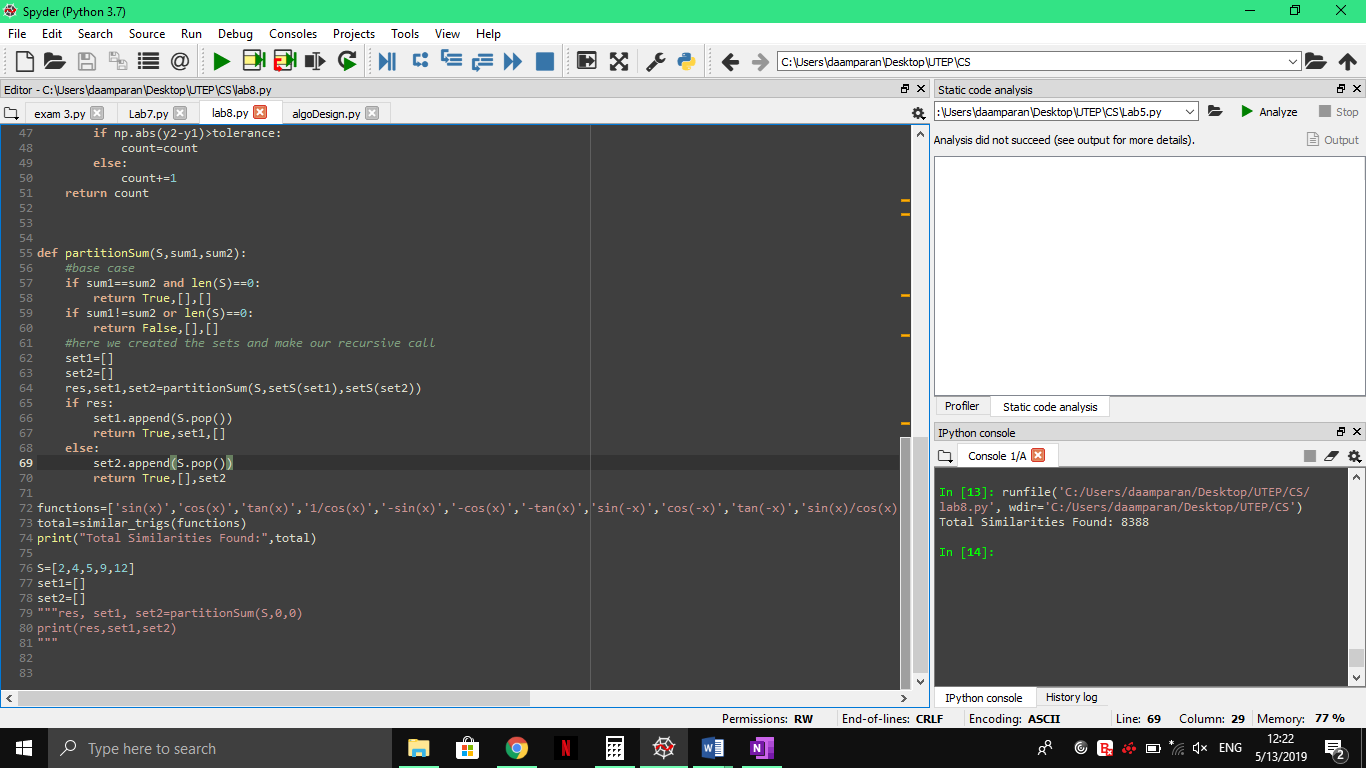
Algorithmic techniques are as stated techniques used to create efficient algorithms. In many algorithms, we solve them iteratively or recursively or simply or conditionally. These techniques vary from both their applications and difficulty stand point. The techniques include;

* Divide and Conquer – Divide and Conquer simply breaks the problem down into separate tasks that are easily solvable, mainly recursive
* Greedy Algorithm – this algorithm works in phases, these phases are then solved by the most optimal decision at the time
* Randomization – Uses the randomization to avoid long running times and to solve those problems of high probability
* Backtracking – Solves problems recursively and when faced with a decision, it does the first instruction and if it does not work it backtracks and performs the second item.
* Dynamic Programming – Solve problems recursively and avoids repeating something that was already done

We were then responsible to create 2 algorithms, one to find the similarity with trig functions and a backtracking method that creates two distinct sets from one set, those sets must have the same sum.

**Proposed Solution**

Similar Trigs – For the similar trigs we must test all trigonometric functions against one another and detect the equalities using randomization and display the amount of equalities with a counter.



partitionSium – Partition sum is a backtracking algorithm which we must build two subsets from S that contain the same sum, to do this we must test if our sets are equal and S is empty or last is less than 0. Our recursive calls will then be made by appending to one set, and if that does not work we then append to the second.

**Experimental Result**

To test the trigonometric equalities I will be increasing the number in both tries and tolerance and see which yield a more accurate result.

From this graph, we can see that the duration of the algorithm directly correlates to the amount of tries. By increasing the amount of tries and tolerance we can get a more accurate sample of which functions are equal to one another.

**Conclusion**

Lab 8 was straightforward and understandable, it was not the easiest lab except for the backtracking method which I’m pretty sure I’m over thinking. I could not come up with a solution to the second method but I did have some ideas. To begin, our base cases must be if our sets have equal sums, then we are done and if the len of set S is 0 then we are done. If our set S is empty or the set sums are not equal then we have a false results. After this our recursive calls must hold the subsets we create. So now, the backtrack call takes place when the set is false we go back and append to set 2 rather than set 1. I believe my method fails within this part because I reference sets that have not been created yet.

**Appendix**

I, David Amparan, certify this assignment was completed by myself with no help nor was copied from another classmate/source. I will take full responsibility if such items are found.

**Source Code**

"""

Created: David Amparan

Instructor: Olac Fuentes

TA: Anindita Nath, Maliheh Zargaran

Last Modified:

"""

import random

import math as mate

import time as tiempo

import numpy as np

from math import \*

def setS(s):

sum=0

for i in range(len(s)):

sum+=s[i]

return sum

def similar\_trigs(trigs,tries=1000,tolerance=0.0001):

#the trigs is a list of the identities

temp=trigs

equalities=[]

count=0

while len(trigs)!=0:

toCompare=trigs.pop(0)

for i in range(len(temp)):

for l in range(tries):

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

if toCompare is not temp[i]:

y1=eval(toCompare)

y2=eval(trigs[i])

if np.abs(y2-y1)>tolerance:

count=count

else:

count+=1

return count

def similar\_trigs2(user,trigs,tries=1000,tolerance=0.0001):

count=0

for i in range(len(trigs)):

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

y1=eval(user)

y2=eval(trigs[i])

if np.abs(y2-y1)>tolerance:

count=count

else:

count+=1

return count

def partitionSum(S,sum1,sum2):

#base case

if sum1==sum2 and len(S)==0:

return True,[],[]

if sum1!=sum2 or len(S)==0:

return False,[],[]

#here we created the sets and make our recursive call

set1=[]

set2=[]

res,set1,set2=partitionSum(S,setS(set1),setS(set2))

if res:

set1.append(S.pop())

return True,set1,[]

else:

set2.append(S.pop())

return True,[],set2

functions=['sin(x)','cos(x)','tan(x)','1/cos(x)','-sin(x)','-cos(x)','-tan(x)','sin(-x)','cos(-x)','tan(-x)','sin(x)/cos(x)','2\*sin(x/2)\*cos(x/2)','sin(x)\*sin(x)','1-(cos(x)\*cos(x))','(1-cos(2\*x))/2','1/cos(x)']

total=similar\_trigs(functions)

print("Total Similarities Found:",total)

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

set1=[]

set2=[]

"""res, set1, set2=partitionSum(S,0,0)

print(res,set1,set2)

"""