CS3 Lab 8 Report

Intro

Lab 8 consisted of the use of backtracking and randomization to solve the two problems given. The first problem we had to solve with randomized algorithms was the one that 'discovered' trigonometric identities. The second problem consisted of solving the partition problem with the use of backtracking.

Proposed Solution and Implementation

I first started by creating a list with all trig expressions, afterwards I started writing the methods that'll 'discover' trigonometric identities. The first method's parameter was the list with all the trigonometric expressions. Inside this method I initialized a counter which would keep track of the number of trig identities found. A nested for loop I then implemented that would iterate through the list. Inside the nested for loop an if statement then calls the method that'll compare both given expressions. If the method called returns that they are equal, the if statement will be entered and inside the counter is incremented by one, the counter is printed, and the newly discovered trig identity is also printed.

The second method that compared the two given trig expressions consists of a for loop that compares the two given expressions 1000 times. Inside this for loop a variable stores a random number between pi and negative pi. Then another two variables store the answer of the two evaluated trig expressions. an if statement then checks if the two trig expressions are equal, and if they are, the if statement is not entered but if it is, it'll return False. If the For loop is done iterating without the if statement being entered, the method will return True.

I then solved the partition problem by writing a method whose parameters is the set. Inside this method an if statement checks if the sum of the set is not divisible by two. If it is not divisible by two, a False is returned, otherwise a variable stores the sum of the set divided by two, then this variable is returned.

The second method that I wrote computes the subsets of this set. The parameters of this set consist of the set(S), the index of the last element(last), and the number returned by the method described previously(goal). Inside this method an if statement checks if the given partition is zero, and if it is it'll return True and an empty list. Another if statement checks if the goal or last is smaller than zero, and if it is, it'll return False and an empty list. After the if statements a recursive call is made. The parameters are changed by subtracting one to last and subtracting the last element from the set to the goal. An if statement then checks if the recursive call returned True, and if it did, the last element in the set is added to one of the two subsets and True is returned, else another recursive call is made with its parameters changing by subtracting one to last and keeping the rest of the parameters the same. The first subset is then returned. To get the other subset, a for loop iterates through the set, and an if statement adds all the integers in the set that are not in the first subset to subset two. Afterwards the two subsets are printed if they exist, otherwise a no partition exists is printed.

Experimental Results:

I experimented with this program by first reviewing over a program Dr.Fuentes provided which I was able to find very useful information and helped me figure out how to write this program. I also had to experiment with getting the random number between pi and negative pi as I kept getting an error. I

experimented with my methods that solved the partition problem by changing the set sent as parameter. The following images display what my program output when ran:

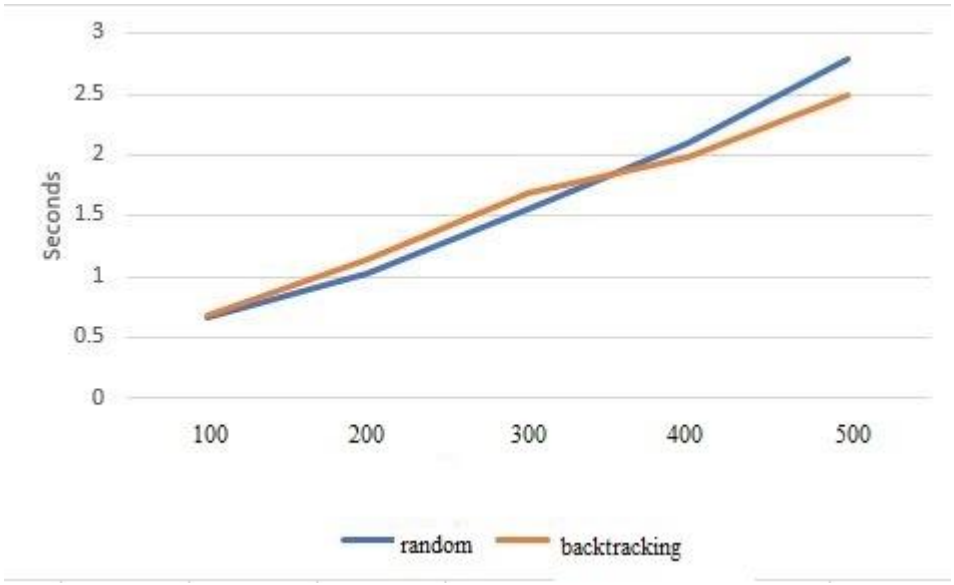
```
In [1]: runfile('C:/Users/flore/.spyder-py3/cs3Lab8.py', wdir='C:/
Users/flore/.spyder-py3')
similarities found:
1
sin(t) sin(t)
2
cos(t) cos(t)
3
cos(t) cos(-t)
4
tan(t) tan(t)
5
tan(t) sin(t)/cos(t)
6
1/cos(t) 1/cos(t)
7
1/cos(t) 1/cos(t)
8
-sin(t) -sin(t)
9
-sin(t) sin(-t)
10
-cos(t) -cos(t)
11
-tan(t) -tan(t)
12
-tan(t) tan(-t)
13
sin(-t) sin(-t)
```

```
14
cos(-t) cos(-t)
15
tan(-t) tan(-t)
16
sin(t)/cos(t) sin(t)/cos(t)
17
2*sin(t/2)*2*cos(t/2) 2*sin(t/2)*2*cos(t/2)
18
sin(t)**2 sin(t)**2
19
sin(t)**2 sin(t)**2
20
sin(t)**2 1-cos(t)**2
21
sin(t)**2 (1-cos(2*t))/2
22
sin(t)**2 sin(t)**2
23
sin(t)**2 1-cos(t)**2
24
sin(t)**2 (1-cos(2*t))/2
25
1-cos(t)**2 1-cos(t)**2
26
1-cos(t)**2 (1-cos(2*t))/2
27
(1-cos(2*t))/2 (1-cos(2*t))/2

28
1/cos(t) 1/cos(t)

partition w/ backtracking:
subset One [4, 12]
subset Two [2, 5, 9]
```

Running Times:



Conclusion:

What I learned from this program is how randomized algorithms and backtracking work and how they are useful. I also learned more about the tolerance that is needed when comparing two expressions/functions. I now have a better understanding on the types of algorithms needed on certain problems thanks to this lab.

Appendix:

```
#Course:CS2302
#aAuthor:Daniela Flores
#Lab8
#Instructor:Olac Fuentes
#T.A: Dita Nath
#date of last Mod: 5/11/19
#purpose of program: Discover Trig identities with the help of randomized algorithms. We also
#had to check if a partition exists in a set of numbers with the aid of backtracking.
import random
import numpy as np
from math import *
import math
#this method checks if the two given trig expressions are equal
def isEqual(f1, f2,tries=1000,tolerance=0.0001):
   #checks if theyre equal 1000 different times3
   for i in range(tries):
      t = random.uniform(-(math.pi), math.pi)
      #evaluates the two expressions
      e1 = eval(f1)
      e2 = eval(f2)
      #if the solutions of both expressions are not equal,return false
      if np.abs(e1-e2)>tolerance:
         return False
   return True
```

```
#method that compares every trig expresion to each other
def comparing(L):
    #count keeps track of trig identites found
    count = 0
    #traverses through whole list containing the expressions
    for i in range(len(L)):
        for j in range(i,len(L)):
            if(isEqual(L[i],L[j])):
                count=count+1
                print(count)
                print(L[i],L[j])


#method that computes subset if there is one
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])
    if res:
        subset.append(S[last])
        return True, subset
    else:
        return subsetSum(S,last-1,goal)
#method that computes if the given set's total is even or odd
def partition(S):
    if sum(S)%2 != 0:
        return False
    goal = sum(S)//2
```

```
    return goal


#list of all trig identities
L = ['sin(t)','cos(t)','tan(t)','1/cos(t)','-sin(t)','-cos(t)','-tan(t)','sin(-t)','cos(-t)',
     'tan(-t)','sin(t)/cos(t)','2*sin(t/2)*2*cos(t/2)','sin(t)**2','sin(t)**2','1-cos(t)**2',
     '(1-cos(2*t))/2','1/cos(t)']
print('similarities found:')
comparing(L)
print()
print('partition w/ backtracking:')
S = [2, 4, 5, 9, 12]
#stores partition of subset
Par = partition(S)
boo, s1= subsetSum(S,len(S)-1,Par)
if Par:
    s2 = []
    print('subset One',s1)
    for j in S:
        if j not in s1:
            s2.append(j)
    print('subset Two',s2)
else:
    print('no partition exits for this set')
```

Contract:

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 inappropiate assistance to any student in the class.

X