Luis Renteria

ID 88740232

CS 2302

Dr. Fuentes

Lab Report 8

**Introduction**

For our final lab, we were tasked with creating a program that would check the equality between certain trigonometric functions in order to “discover” identities, as well creating a solution to a problem known as the “partition problem” through the use of backtracking.

**Discovering Trigonometric Identities**

The first part of this lab required us to test 16 trigonometric expressions in order to find equalities and “discover” identities. I began by initializing a list of size 16, and filling it with empty strings. I then input every expression manually in the order they were given, and sent the now full list to the Identities method.

The identities method uses a double for loop to iterate through the list and compare every expression to every other expression, but will not compare an expression to itself. The number of times that an expression is compared to another can be chosen, but if the number of tries is not specified, the expressions will be tested 10 times.

When comparing the two different expressions, a random number between negative pi and positive pi is selected. This number is then used to evaluate both expressions with the use of the eval function, and the answers are stored in two separate variables. The difference of the two variables is found, and if the difference is less than the chosen error tolerance, the expressions are identified as even. Every time an expression is found as equal, a counter is incremented by 1. When the two expressions have been tested “tries” number of times, an if statement checks if the counter is equal to “tries”. If the counter and “tries” are equal, the two expressions are an identity, and are printed out.

**Partition Problem**

The partition problem is a way to determine whether a given set can be partitioned into two subsets where the sum of each set equals each other. The set that I used for this program is {2,4,5,9,12}

Before finding the partition, the set was sent to the PartitionSet method to determine whether a partition could exist or not. Using a for loop, a sum of the set was found by iterating through the set and adding each element together. If the sum was odd, no partition could exist, but if the sum was even, there was a possibility of a partition existing. From this method, the set was sent to the CalculatePartition method to find where the partition exists. I was unable to complete this part of the lab, and therefore have no running times for this part of the lab.

**Running Times**

The table below lists times done with the number of comparisons done for each expression. Time is represented in seconds, rounded to 4 decimal places.

|  |  |  |  |
| --- | --- | --- | --- |
| Trial | Size 10 | Size 50 | Size 100 |
| 1 | 1.0782 | 5.0976 | 8.8891 |
| 2 | 1.0938 | 4.9938 | 8.5787 |
| 3 | 1.1417 | 5.4921 | 8.8042 |
| 4 | 1.0469 | 4.7671 | 8.9536 |
| 5 | 1.1096 | 4.2522 | 8.8164 |
| Average | 1.09404 | 4.92056 | 8.8084 |

As seen in the table above, the runtime changes based on the number of comparisons. If the program were to compare the same expression to itself, the time for size 50 would most likely be more than 5 seconds and near 10 seconds for size 100.

Runtime: O(n)

**Conclusion**

This lab taught me how to use the eval function to evaluate strings as mathematical expressions, as well as hone my problem solving skills when I was required to compare the expressions by putting them all in a list.

**Academic Statement**

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.

Signed, Luis Renteria.

**Appendix**

Below is the code given to us by Dr. Fuentes that I used as a basis for this lab assignment.

def equal(f1, f2,tries=1000,tolerance=0.0001):

for i in range(tries):

x = random.random()

y1 = eval(f1)

y2 = eval(f2)

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

return False

return True

def prime(n,tries=1000):

m = int(np.sqrt(n))

for i in range(tries):

x = random.randint(2, m)

if n%x==0:

print(x)

return False

return True

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]

def edit\_distance(s1,s2):

d = np.zeros((len(s1)+1,len(s2)+1),dtype=int)

d[0,:] = np.arange(len(s2)+1)

d[:,0] = np.arange(len(s1)+1)

for i in range(1,len(s1)+1):

for j in range(1,len(s2)+1):

if s1[i-1] ==s2[j-1]:

d[i,j] =d[i-1,j-1]

else:

n = [d[i,j-1],d[i-1,j-1],d[i-1,j]]

d[i,j] = min(n)+1

#print(d)

return d[-1,-1]

f1 = 'x\*x + x - 12'

f2 = '(x+4)\*(x-3)'

print(equal(f1,f2))

f1 = 'sin(x)/cos(x)'

f2 = 'tan(x)'

print(equal(f1,f2))

f1 = 'sin(x)\*sin(x) + cos(x)\*cos(x)'

f2 = '1'

print(equal(f1,f2))

f1 = '(x+1)\*(x-1)'

f2 = 'x\*x-1'

print(equal(f1,f2))

f1 = '(x+10)/10'

f2 = 'x'

print(equal(f1,f2))

print(prime(997))

print(prime(1008))

s2='MONEY'

d = edit\_distance('MINERS','MONEY')

print(d)

S = [2,5,8,9,12,21,33]

for i in range(100):

print('Goal =',i)

a,s = subsetsum(S,len(S)-1,i)

if a:

print('Solution:',s)

else:

print('There is no solution')