Ivan Perez
80602918
Lab 8 Report

Introduction

Using randomized algorithms and backtracking, this code is to test all the combinations of trigonometric expressions in the lab pdf and detect equalities. The code is supposed to generate a random number in the range of -pi to pi. The second part of this problem has a set of numbers that need to be partitioned into 2 sets that have the same sum.

Proposed solution design and implementation

To solve the trig problem, I first created a method to that receives t and computes all the trig expressions (a-p) in the lab document and appends them to a list and returns it. I had a separate method call this method. The resulting list I had, I needed to find a way to compare the values within the list and make sure I don't repeat any of the expressions. As I was thinking about it, it sounded like a disjoint set forest would solve this problem so I brought in the forest object and used a for loop to go through the list and union values that were the same into a set. I then converted the set forest into a set list. I had a pre-made list made that contained the list expression in the lab document (example: sin(t), cos(t), etc.) Using the set list I just printed the value of t, the value the expressions were similar to, and the expressions that would equate to the value.

Making two sets that had the same sum from a main set was a more difficult solution. I was thinking of appending the values to one set and using recursion to remove values from the set into the second set until it had the same value or if not would display partition was not possible.

Experimental results

To test the results I used a random value that goes through the method to determine trigonometric equalities. The amount of calls and computation were the same since it was just solving 16 different trig expressions. The only thing that took time was printing the results but other than that the results for random values were not much different. Obtaining the trigonometric results took around .003 seconds on average so I didn't include running time since there wasn't much of a difference.

I was not able to get the runtime for the partition because the kernel would crash every time I tried getting results. I included sample runs below though.

```
2. Backtracking

Partitionable Set
S =  [5, 20, 25]
Set Partition:

S1 =  [5, 20]
S2 =  [25]

Non Partitionable Set
S =  [1, 2, 7, 30]
Set Partition:

Partition does not exist
```

```
1. Randomized Algorithm

t =  -2.4075913315891437

Value:  -0.6698459713007686
Equalities:
sin(t)
2sin(t/2)*cos(t/2)

Value:  -0.7425000839946956
Equalities:
cos(t)
cos(-t)

Value:  0.9021493542424354
Equalities:
tan(t)
sin(t)/cos(t)

Value:  0.6698459713007686
Equalities:
-sin(t)
sin(-t)

Value:  -0.9021493542424354
Equalities:
-tan(t)
tan(-t)

Value:  0.44869362526787016
Equalities:
sin^2 (t)
1-cos(2t)/2
```

Conclusions

This project was more difficult to produce an algorithm but was very straightforward in implementation once an algorithm was constructed. What I learned from this was that an efficient algorithm can cut out a lot of code and technical work that would be needed otherwise. I learned how to create an algorithm to partition a set and to compute equalities of trigonometric functions.

Appendix

```
#############################################################################
#
#
# Course: CS2302 (MW - 1:30pm)
# Author: Ivan Perez
```

```
# Assignment: Lab 8: Algorithm Design Techniques
# Instructor: Olac Fuentes
# T.A.: Anindita Nath
# Last Modified: May 13, 2019
# Purpose: Code implements use of randomized algorithms and backtracking.
# randomized is used to show equalities of trigonometric expressions of a
# randomly outputted t from -pi to pi. The backtracking algorithm is implemented
# by trying to partition a set into 2 equally summed sets
#
################################################################################
#

import math
import mpmath
import random
import numpy as np

################ Disjoint Set Forest Object ####################
def DisjointSetForest(size):
    return np.zeros(size,dtype=np.int)-1

def find_c(S,i): #Find with path compression
    if S[i]<0:
        return i
    r = find_c(S,S[i])
    S[i] = r
    return r

def union_c(S,i,j):
    # Joins i's tree and j's tree, if they are different
    # Uses path compression
    ri = find_c(S,i)
    rj = find_c(S,j)
    if ri!=rj:
        S[rj] = ri

def dsfToSetList(S):
    #Returns aa list containing the sets encoded in S
    sets = [ [] for i in range(len(S)) ]
    for i in range(len(S)):
        sets[find_c(S,i)].append(i)
    sets = [x for x in sets if x != []]
    return sets
```

```
######################## Code for the assignment ########################

####///////////////// 1. Randomized Algorithm
def trigOfT(t):
    # computes values asked for (a through p) in the lab and appends to list
    trig = []
    trig.append(math.sin(t))
    trig.append(math.cos(t))
    trig.append(math.tan(t))
    trig.append(float(mpmath.sec(t)))
    trig.append(-math.sin(t))
    trig.append(-math.cos(t))
    trig.append(-math.tan(t))
    trig.append(math.sin(-t))
    trig.append(math.cos(-t))
    trig.append(math.tan(-t))
    trig.append(trig[0]/trig[1])
    l1 = float((math.sin(t/2)))
    l2 = float((math.cos(t/2)))
    trig.append(float(2*(l1*l2)))
    trig.append(trig[0]*trig[0])
    trig.append(1 - (trig[1]* trig[1]))
    trig.append((1 - math.cos(2*t))/2)
    trig.append(1/trig[1])
    return trig

def equalities(t):
    print()
    print('t = ',t)
    print()
    # below is a list of the identities required
    identity= ['sin(t)','cos(t)','tan(t)','sec(t)','-sin(t)','-cos(t)',
               '-tan(t)','sin(-t)','cos(-t)', 'tan(-t)', 'sin(t)/cos(t)',
               '2sin(t/2)*cos(t/2)', 'sin^2 (t)','1-cos^2 (t)', '1-cos(2t)/2',
               '1/cos(t)']

    trig = trigOfT(t) # computes trigs of t

    # forest is used to determine which values are equal to each other
    dsf = DisjointSetForest(len(trig))
    for i in range(len(trig)):
        for j in range(len(trig)):
```

```
            if trig[i]==trig[j]:
                union_c(dsf, i, j)

    dsf = dsfToSetList(dsf) #converts to set list

    #prints t and which values are equal to each other
    for i in range(len(dsf)):
        if len(dsf[i])>1:
            print('Value: ',trig[dsf[i][0]])
            print('Equalities:')
            for j in range(len(dsf[i])):
                print(identity[dsf[i][j]])
            print()


####///////////////// 2. Backtracking
def partition(numbers, end, set1, set2, pos):
    #base case
    if pos == end:
        if sum(set1) == sum(set2):
            return True, set1, set2
        else:
            return False, [], []
    #places all the values into set 1
    set1.append(numbers[pos])
    result, val1, val2 = partition(numbers, end, set1, set2, pos+1)
    if result:
        return result, val1, val2
    #removes values from 1 and appends to 2 until partitioned (backtracks)
    set1.pop()
    set2.append(numbers[pos])
    return partition(numbers, end, set1, set2, pos +1)

def partitionSet(nums):
    nums.sort()
    print('S = ',nums)
    print('Set Partition:')
    print()

    lsum = sum(nums)
    if lsum % 2 == 0:
    # first case to determine if set is partitionable
        part, set1, set2 = partition(nums, len(nums), [],[],0)
```

```
        # calls partition method and prints set if partitionable
        if part:
            print('S1 = ', set1)
            print('S2 = ', set2)
        else:
            print('Partition does not exist')
    else:
        print('Partition does not exist')


#_____________________________ Test Code _____________________________________#

print('1. Randomized Algorithm')
equalities(random.uniform(-math.pi,math.pi))
print()
print('2. Backtracking')
print()
print('Partitionable Set')
partitionSet([5,20,25])
print()
print('Non Partitionable Set')
partitionSet([2,1,7,30])
```

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.