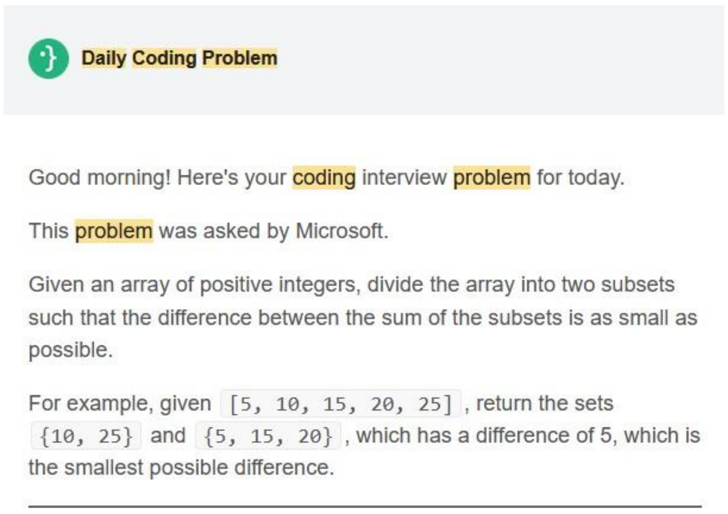
Daily Coding Problem

Good morning! Here's your coding interview problem for today.

This problem was asked by Microsoft.

Given an array of positive integers, divide the array into two subsets such that the difference between the sum of the subsets is as small as possible.

For example, given `[5, 10, 15, 20, 25]`, return the sets `{10, 25}` and `{5, 15, 20}`, which has a difference of 5, which is the smallest possible difference.

** LOGIC EVALUATED ** - THERE ARE TWO SETS

explore all the ways of filling numbers from provided set (5,10,15,20,25) into subset of size 1… We know the other set will then have subset =4 with remaining numbers… C(5,1)

Can store the subsets now in a 3d integer array

explore all the ways of filling numbers from provided set (5,10,15,20,25) into subset of size 2… We know the other set will then have subset =3 with remaining numbers..
C(5,2)

Can store the subsets now in a 3d integer array

explore all the ways of filling numbers from provided set (5,10,15,20,25) into subset of size 3… We know the other set will then have subset =2 with remaining numbers… C(5,3)

Can store the subsets now in a 3d integer array

explore all the ways of filling numbers from provided set (5,10,15,20,25) into subset of size 4… We know the other set will then have subset =1
C(5,4)

Can store the subsets now in a 3d integer array

We know the total of set size will be c(5,4) + c(5,3) + c(5,2) + c(5,1) for execution in blue

We know the above process in blue has paired the subsets… Placing the subset into array was undesired part. They could have equally been paired up in 3d String array in a set…

Process the 3d integer array, obtain totals for each subset at [i][][] keep track of the lowest difference(s) , there might be multiple pairs of subsets with same lowest difference.

Present the subset in 3D array at obtained index i.

LOGIC CLARIFIED (INTRODUCING FURTHER TECHNIQUES).

This would simply be similar to all ways of arranging numbers (completed Twitter challenge on Friday 18 October 2024).

numbers = 5

It would then simply be a case of dissecting the values in set {1 number}{4 numbers}
{2 number}{3 numbers}
{3 numbers}{2 numbers}
{4 numbers}{1 numbers}

We know for instance if the first n numbers are the same (irrespective of order), the dissecting is irrelevant. Since the end total of partitions will be identical.
It would just need to process the total for the first occurrence in order to save execution time.
A clean process would be to store the first partition into another set.
For example if random numbers selected {10,15,20,25,5} it would store following into a 3D string:

data[0][0][0] =10, data[0][1][0]=10 (total of the subset), data[0][2][0]= 65 (75-10) { {10}, {10}, {65} }
data[0][0][0] =10, data[0][0][1] =15, data[0][1][0]=25 (total of the subset), data[0][2][0]= 50 (75-10) { {10,15}, {25}, {50} }
data[0][0][0] =10, data[0][0][1] =15, data[0][0][2] =20, data[0][1][0]=45 (total of the subset), data[0][2][0]= 30 (75-45) { {10,15,20}, {45}, {30} }
data[0][0][0] =10, data[0][0][1] =15, data[0][0][2] =20, data[0][0][3] =20 data[0][1][0]=65 (total of the subset), data[0][2][0]= 10 (75-65) { {10,15,20,25}, {65}, {10} }

It would start process again for a new round of number generation (similar to Twitter)
If there is not an entry (note it would need to only check data[X][0][Y] in the first subset of each row) with numbers generated, it would only then add it into the data 3D array.
Note, this level of checking is similar to the Google challenge undertaken on 1 January 2025. Since it is showing a level of strategy, which is useful. However with this challenge (Microsoft), it is still not efficient enough since {10,15} is equivalent to {15,10} in respect to total for partition… So it could attempt to compare at the granular level of first subset in each row of data 3d array.
If I can achieve two dimensional Strings in Set, I would be keep to utilize Predicate for the first time to interrogate the Set Entries…. This will look much tidier than repeat code… I will need to practice this first on sample data.

As the processing of number generation proceeds and r gets larger, its scope of storing entries into data 3D will become significantly narrow (Since were are dealing with two partitions) and would expects lots repetitions.
In practice with above, we are effectively doing the same job as the set… So it can be decided, also to omit the above and simply attempt to store the C(n,r) into the set directly and calculate the other two dimensions (total partition, total remaining partition).
This has led to me to try and create some code (see end of documentation). It was realised that scope on experimenting with set served little purpose. I utilized predicate to search the content in Set entries. But I was effectively doing a search such as {10,15,20,25,5} against the existing Set.
I further had a thought in respect to using another Collection such as a List. Since it was worthwhile checking for existing entry since we know that duplicates are permitted in Lists…

This aspect of the code worked successfully. Please note I was using high level code straight from the internet.

And also because of this, I was not able to ascertain how to find index at which filter matched entry in the List. Not was I able to even structure code for a Boolean value (match or no match).

But overall it was useful experience given the necessity to store replicated elements if required. Furthermore, the list maintains order in data entries. Predicate will be an area I wish to gain insight into.

Using the set and predicate, I do not foresee improvements in metrics such as time of code execution. It will not disrupt the current cycle prematurely, if repetition detected early. But I can foresee the code to be useful in future.

<u>So my normal code will execute as follows:</u>
It will simply be a case of reporting the totals mid execution as other software code..

And at end, it will also give analysis as per usual with arrays.deepToString().

Also note, since the combination (without replacement class) is going to be called multiple times, the backup copy of the set and actual values in the set will be useful. Since we are aware that Set is written to in no particular order.
This will enable successful non-duplicated reporting mid-execution and at the end.

I believe completing this challenge will provide good mindset to attempt ETSY challenge in future which dealt with several partition sizes.

There would still be a limitation of r=64