Problem

You are given 3 glasses with their respective capacities (volumes) stored in an array `cap`. It is possible to perform either of the two following operations to the glasses:

- empty one of the glasses entirely;
- pour water from one glass into another until either the first one is empty or the second one is full.

Initially all the glasses are full (i.e. for each valid `i` the `ith` glass has `cap[i]` units of water). How many**different** positive volumes of the total amount of water in all the glasses is it possible to get performing an arbitrary number of described actions?

Example

- For `cap = [1, 1, 1]`, the output should be `threeGlasses(cap) = 3`.

Initially there are `3` units of water in all the glasses. The only action you can perform is to empty the water from one of the glasses, which will leave you with `2` units. At this point, pouring water from one glass to another won't produce a different total amount of water, so the only remaining option is emptying one of the remaining glasses, which will leave you with `1` final unit of water. Thus, it is possible to obtain `1`, `2` or `3` units of water, so the answer is `3`.

- For `cap = [16, 5, 3]`, the output should be `threeGlasses(cap) = 21`.

Here is the list of all possible amounts of water across all three glasses: `1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 21, 24`.

Input/Output

- **[time limit] 3000ms (java)**

- **[input] array.integer cap**

  Individual capacity of each glass (their volume), listed as three positive integers.

  Constraints:
  $1 \le cap[i] \le 100$.

- **[output] integer**

  The number of different positive total volumes.

Solution

Create a recursive method that generates all combinations of volume on the glasses possible with one single operation. That is , given a starting point, generate all possible outputs generated by emptying one of the glasses or pouring water from a glass to the other. Run this method on all generated combinations

The operations generated will be saved on a Set , so we can have a record of which combinations have already been used , so we can avoid infinite recursion.

The total volume of each combination is also saved in a Set. At the end of the run , this set will contain all possible volumes and its size will be the answer to the problem.