0h 10m

Task 2

Java 8

You have just rolled a dice several times. The N roll results that you remember are described by an array A. However, there are F rolls whose results you have forgotten. The arithmetic mean of all of the roll results (the sum of all the roll results divided by the number of rolls) equals M.

What are the possible results of the missing rolls?

Write a function:

```
class Solution { public int[] solution(int[] A, int F, int M); }
```

that, given an array A of length N, an integer F and an integer M, returns an array containing the possible results of the missed rolls. The returned array should contain F integers from 1 to 6 (valid dice rolls). If such an array does not exist then the function should return [0].

Examples:

1. Given A = [3, 2, 4, 3], F = 2, M = 4, your function should return [6, 6]. The arithmetic mean of all the rolls is (3 + 2 + 4 + 3 + 6 + 6) / 6 = 24 / 6 = 4.

2. Given A = [1, 5, 6], F = 4, M = 3, your function may return [2, 1, 2, 4] or [6, 1, 1, 1] (among others).

3. Given A = [1, 2, 3, 4], F = 4, M = 6, your function should return [0]. It is not possible to obtain such a mean.

4. Given A = [6, 1], F = 1, M = 1, your function should return [0]. It is not possible to obtain such a mean.

Write an **efficient** algorithm for the following assumptions:

- N and F are integers within the range [1..100,000];
- each element of array A is an integer within the range [1..6];
- M is an integer within the range [1..6].

Files

- task2
  - solution.java
  - test-input.txt

solution.java ×

```
// System.out.println("this is a debug message");


//Time O(F*n!)
//Space O(n)
class Solution {
    public int[] solution(int[] A, int F, int M) {
        // write your code in Java SE 8
        int n = A == null ? 0 : A.length;

        //Existing sum is the sum for current known dice
        int existingSum = 0;
        for(int i = 0; i < n; i++) {
            existingSum += A[i];
        }

        //Total sum including F times of unknown dices
        int totalSum = M * (F + n);
        int remainingSum = totalSum - existingSum;

        //Dial with edge cases
        if(remainingSum < F || remainingSum > 6 * F) {
            int[] res = new int[] {0};
            return res;
        }
        List<List<Integer>> res = new ArrayList<>();
        List<Integer> cur = new ArrayList<>();
        DFS(remainingSum, F, 0, res, cur);
        List<Integer> oneResult = res.get(0);
        int[] finalResult = new int[F];
        for(int i = 0; i < F; i++) {
            finalResult[i] = oneResult.get(i);
        }
        return finalResult;
    }

    //Use recursion way to find one potential combination
    private void DFS(int sum, int F, int index, List<List<Integer>> res, List<Integer>
        if(index == F) {
            if(sum == 0) {
                res.add(new ArrayList<>(cur));
                //We will stop the program once we found first qualified array for missi
                return;
            }
            return;
        }
        for(int i = 1; i <= 6; i++) {
            cur.add(i);
            DFS(sum - i, F, index + 1, res, cur);
            cur.remove(cur.size() - 1);
        }
    }
}
```

Test Output

```
Example test:   ([1, 5, 6], 4, 3)
OK

Example test:   ([1, 2, 3, 4], 4, 6)
OK
```