2028. Find Missing Observations

Simulation

Math

Problem Description

Medium <u>Array</u>

reconstruct the missing n observations such that when combined with the provided m observations, the overall average remains the same. The known m observations are provided in an array called rolls, the overall average (which is an integer) is given by mean, and the number of missing observations is n.

You are given a certain amount of 6-sided dice roll observations m, but unfortunately, n additional observations are missing. The

average value of all n + m dice rolls, including the missing ones, has been calculated and is known to you. It is your task to

• The sum of the resulting n + m observations must be equal to mean * (n + m) since that product gives the total sum implied by the average mean. • Each element in the resulting array must be a valid dice roll number between 1 and 6.

Your goal is to return an array of length n that represents the missing observations. The constraints you must abide by are:

If it's impossible to find such an array that satisfies these conditions, you are to return an empty array instead.

Intuition

Understanding this problem requires recognizing that it's a matter of distributing a specific sum (s) across n rolls such that each

roll is between 1 and 6.

Since the problem states that the average is an integer, we know the total sum of all observations must be divisible by the number of observations, which is a key starting point. Moreover, we are dealing with average values, meaning the total sum must be spread over can be determined by the formula (n + m) * mean.

Here's how we arrive at the solution approach:

the amount that the missing rolls must sum up to.

 \circ The minimum sum we could get from n dice rolls is n, with each roll being 1. • The maximum sum we could get from n dice rolls is 6 * n, with each roll being 6. \circ So, if s is outside of the range [n, 6 * n], constructing an array is impossible, and we should return an empty list.

3. If the sum s falls within the valid range, we distribute it evenly across the n missing observations. Each missing observation will initially be

1. Calculate the total sum s needed for the missing observations by subtracting the sum of given rolls from the desired total sum. This gives us

assigned the integer part of the division s // n. 4. However, since s may not be exactly divisible by n, there could be a remainder. This remainder has to be distributed amongst the observations as well. We do this by adding 1 to each of the first $s \approx n$ observations since the remainder can be at most n-1.

The approach used in the Python code above implements these steps succinctly, ensuring that all constraints of the problem are

2. Confirm that it's possible to construct a valid array with the missing observations:

met.

The implementation follows a direct approach, which aligns with the intuition previously explained. To outline the steps of the implementation:

5. After these steps, we would have an array that contains the missing observations and satisfies the average value requirement.

Calculate the sum s that the missing rolls must add up to. This is done by multiplying the overall mean by the total number of observations (both missing and known), and then subtracting the sum of the known observations:

Check if the sum s is within the bounds of what is achievable with n rolls. The conditions for s to be valid are that it can't be

Allocate the minimum guaranteed value to each of the n missing observations by performing integer division of s by n. This

ensures that we are using up as much of s as possible while keeping the value of each roll within the dice's constraints (1 -

less than n (since every roll is at least 1) and it can't be greater than 6 * n (since every roll is at most 6): if s > n * 6 or s < n:

6 inclusive).

ans = [s // n] * n

Solution Approach

return []

Here, ans is the list that will be returned, initially filled with the integer part of the division.

distributed as possible without exceeding the die's face value limit of 6.

Let's walk through a small example to illustrate the solution approach.

• mean: 4 as the overall average of all the dice rolls (including the missing ones).

Then, we calculate the total sum implied by the average:

Check if the sum s is within the achievable bounds:

Suppose we are given the following parameters:

• n: 2 indicating that there are two missing observations.

• rolls: [3, 4, 3] representing the known dice rolls

s = (n + m) * mean - sum(rolls)

s = (2 + 3) * 4 - sum([3, 4, 3])

m is the number of known observations, so len(rolls) gives us this value.

If s does not satisfy these conditions, an empty list is returned, signaling that it's impossible to have such a distribution.

s = (n + m) * mean - sum(rolls)

Distribute the remainder of s across the n missing observations. The modulus operator % gives the remainder of the division between s and n:

This approach assumes that there are valid values for all missing observations such that their sum is s. By using integer division

and modular arithmetic, the solution fills the ans array with valid die rolls that reach the sum s without violating any of the dice

constraints. The algorithm has a constant time complexity with respect to the input array rolls since it's only dependent on the

number of missing observations n. The space complexity is also linear with respect to the number of missing observations n.

We only need to iterate over the first s % n elements of the ans array. This ensures that the extra values are as evenly

for i in range(s % n):

ans[i] += 1

Example Walkthrough

results in the average value 4. Following the steps outlined in the solution approach: Calculate the total sum s needed for the missing observations: First, we determine the total number of observations (missing + known), which is 3 + 2 = 5.

Since s is exactly divisible by n in this case, there is no remainder and this step is not necessary. The distribution [5, 5]

Thus, the reconstructed array [5, 5] when combined with the original rolls [3, 4, 3], maintains the average value 4:

We must check if s is at least n (every roll is at least 1) and at most 6 * n (every roll is at most 6):

Since s is 10, which is within the range of [2, 12] ([n, 6 * n]), there is a possible solution.

already satisfies the conditions since both numbers are between 1 and 6 and the total is 10.

We want to reconstruct the missing observations such that the resulting array, when combined with the provided observations,

So, the sum of the two missing observations must be 10.

s = 20 - 10

s = 10

s = 5 * 4 - 10

ans = [s // n] * n

ans = [10 // 2] * 2

• Total value of all rolls: 3 + 4 + 3 + 5 + 5 = 20

Average (Total value / Number of rolls): 20 / 5 = 4

ans = [5] * 2

Number of all rolls: 5

Python

Java

C++

public:

#include <vector>

class Solution {

using namespace std;

class Solution {

class Solution:

Solution Implementation

ans = [5, 5]

Allocate the minimum guaranteed value to each missing observation: We divide the sum s by the number of missing observations n:

This provides us with an initial distribution for the missing observations.

Distribute the remainder of s across the missing observations:

Therefore, the final reconstructed array with the missing observations would be [5, 5]. This completes the example walkthrough using the given approach.

def missingRolls(self, rolls: List[int], mean: int, n: int) -> List[int]:

Calculate the total sum of all the dice rolls based on the given mean

Calculate the sum of the missing rolls by subtracting the sum of existing rolls

Distribute the remainder among the first (remainder) rolls, adding 1 to each

// The total sum S that all rolls (both missing and given) should sum up to

// If the total sum needed is impossible (either too low or too high given the constraints),

If the sum of the missing rolls is not between n and 6n, it's impossible to get such a sequence if sum missing_rolls > n * 6 or sum_missing_rolls < n:</pre> return [] # Start with an even distribution of the sum across all the missing rolls

missing_rolls = [sum_missing_rolls // n] * n

for i in range(sum missing_rolls % n):

Return the final list of missing rolls

// m represents the number of given rolls

int totalSumRequired = (n + m) * mean;

totalSumRequired -= rollValue;

for (int rollValue : rolls) {

// return an empty array

return new int[0];

// Return the missing rolls array

for (int rollValue : rolls) {

return {};

const minPossibleSum = n;

return [];

const maxPossibleSum = n * 6;

++missingRolls[i];

totalSumNeeded -= rollValue;

return missingRolls;

// Method to find the missing rolls given the mean of all the rolls

// Subtract the sum of given rolls from the total sum required

public int[] missingRolls(int[] rolls, int mean, int n) {

// to find the sum needed from the missing rolls

if (totalSumRequired > $n * 6 \mid \mid$ totalSumRequired < n) {

vector<int> missingRolls(vector<int>& rolls, int mean, int n) {

int totalSumNeeded = (n + numberOfExistingRolls) * mean;

if (totalSumNeeded > $n * 6 \mid \mid$ totalSumNeeded < n) {

vector<int> missingRolls(n, totalSumNeeded / n);

for (int i = 0; i < totalSumNeeded % n; ++i) {</pre>

// Calculate the sum needed from the missing rolls

// Initialize an array to hold the missing rolls

// Calculate the average roll needed, rounded down

const missingRollsArray = new Array(n).fill(0);

// Fill the array with the average roll value

for (let i = 0; i < n && remainder > 0; i++) {

// Return the array representing the missing rolls

missingRollsArray.fill(averageMissingRoll);

if (missingRollsArray[i] < 6) {</pre>

missingRollsArray[i]++;

remainder--;

return missingRollsArray;

return []

return missing_rolls

Time and Space Complexity

const requiredSumFromMissing = expectedTotalSum - currentSumOfRolls;

// If the required sum is outside the range of possible sums, return an empty array

if (requiredSumFromMissing < minPossibleSum || requiredSumFromMissing > maxPossibleSum) {

// Establish the range for the possible sum of the missing rolls

const averageMissingRoll = Math.floor(requiredSumFromMissing / n);

let remainder = requiredSumFromMissing - averageMissingRoll * n;

if sum missing_rolls > n * 6 or sum_missing_rolls < n:</pre>

missing_rolls = [sum_missing_rolls // n] * n

for i in range(sum missing_rolls % n):

Return the final list of missing rolls

missing_rolls[i] += 1

Start with an even distribution of the sum across all the missing rolls

Distribute the remainder among the first (remainder) rolls, adding 1 to each

// Distribute the remainder of the required sum across the missing rolls

// Create the result vector with default values (totalSumNeeded / n)

missing_rolls[i] += 1

return missing_rolls

int m = rolls.length;

sum_missing_rolls = total_dice_rolls - sum(rolls)

total dice rolls = (len(rolls) + n) * mean

// Initialize the array to hold the missing rolls int[] missingRolls = new int[n]; // Fill the missing rolls array with the quotient of the sum needed and n // which ensures the mean is maintained Arrays.fill(missingRolls, totalSumRequired / n); // Distribute the remainder of the sum needed evenly across the first // 'totalSumRequired % n' elements by adding one to each for (int i = 0; i < totalSumRequired % n; ++i) {</pre> ++missingRolls[i];

// Return the completed vector of missing rolls return missingRolls; **}**; **TypeScript** function missingRolls(rolls: number[], targetMean: number, n: number): number[] { // Calculate the desired total sum based on the target mean and total number of rolls const expectedTotalSum = (rolls.length + n) * targetMean; // Calculate the current sum of given rolls const currentSumOfRolls = rolls.reduce((previousValue, currentValue) => previousValue + currentValue, 0);

// Distribute the remaining values (totalSumNeeded % n) evenly across the first few elements

int numberOfExistingRolls = rolls.size(); // The number of existing rolls in the sequence

// The total sum required to reach the desired mean for all rolls (both existing and missing)

// Subtract the sum of existing rolls from the total sum needed to find the sum of the missing rolls

// Return an empty vector if the total sum of missing rolls exceeds the max or min possible sum

class Solution: def missingRolls(self, rolls: List[int], mean: int, n: int) -> List[int]: # Calculate the total sum of all the dice rolls based on the given mean total dice rolls = (len(rolls) + n) * mean# Calculate the sum of the missing rolls by subtracting the sum of existing rolls sum_missing_rolls = total_dice_rolls - sum(rolls)

The time complexity of the provided code is O(m), where m is the number of elements in the rolls list. This time complexity comes from the need to calculate the sum of the rolls list with the sum(rolls) function. The rest of the operations, including the division and modulus operations, as well as assigning values to the ans list, have a constant time complexity or a time complexity of O(n) which is dominated by the O(m) complexity due to the summing operation, assuming n is much smaller than or at most equal to m.

If the sum of the missing rolls is not between n and 6n, it's impossible to get such a sequence

The space complexity is O(n), because we are creating a list ans which contains n integers. The space taken up by m is negligible compared to n because we are given that we're generating n new roll results, and no other data structures are utilized that are dependent on the size of the input.