Task

There are N bins, numbered $1, 2, \ldots, N$. Initially, for each $1 \leq i \leq N$, the i-th Bin has a_i ($1 \leq a_i \leq 3$) balls.

You will perform the following operation repeatedly until all balls are taken:

Roll a die that shows the numbers $1, 2, \ldots, N$ with equal probabilities, and let i be the outcome. If there are some balls in the i-th Bin, take one of them; if there is none, do nothing.

Determine the expected number of operations needed to take all the balls in $O(N^3)$ time complexity.

Examples

Sample 1

$N = 3$ and $a = \{1, 1, 1\}$. So, all the bins have 1 ball each. Expected number of operations needed to take all the balls is 5.5

The expected number of operations before the first ball is taken, is 1. After that, the expected number of operations before the second ball is taken, is 1.5. After that, the expected number of operations before the third ball is taken, is 3. Thus, the expected number of operations is $1 + 1.5 + 3 = 5.5$.

Sample 2

$N = 1$ and $a = \{3\}$. So, the only bin has 3 balls. Expected number of operations needed to take all the balls is 3.

Sample 3

$N = 2$ and $a = \{1, 2\}$. So, we have two bins; one bin has 1 ball, other one has 2 balls. Expected number of operations needed to take all the balls is 4.5.

Sample 4

$N = 2$ and $a = \{1, 3\}$. Expected number of operations needed to take all the balls is 6.25.

Sample 5

$N = 10$ and $a = \{1, 3, 2, 3, 3, 2, 3, 2, 1, 3\}$. Expected number of operations needed to take all the balls is around 54.4806445749. Note that you need to calculate the exact value up to a relative error of 10^{-9}. Using doubles in Java is enough.

Solution

It's clear that we should use dynamic programming (DP) for this problem. Let $dp[b_1][b_2][b_3]...[b_N]$ represent the expected number of operations to take all the balls assuming there are b_1 balls in the first bin, b_2 balls in the second bin,...,b_N balls in the last bin. However, this leads to an N-dimensional array with 4^N states, which is inefficient.

Upon reflection, distinguishing between bins with the same ball count is unnecessary. Whether we have "2 balls in Bin-1 and 1 ball in Bin-2" or "1 ball in Bin-1 and 2 balls in Bin-2" the expected number of operations remains the same. Thus, we only need to track the total ball count across the bins.

Let $dp[i][j][k]$ represent the expected number of operations to take all balls such that we have i bins having 1 balls, j bins having 2 balls and k bins having 3 balls. We also need to know about the number of bins having zero balls. But this is equal to $N - i - j - k$, so no need for an extra 4-th dimension.

We initialize $dp[0][0][0] = 0$. Because we need zero operations to take all the balls if we have 0 balls.

How can you calculate the value of $dp[i][j][k]$? Which states can you reach if you have i 1-ball bins, j 2-ball bins and k 3-ball bins? There are 4 possible operations:

- Don't take any ball. Afterwards, jou still have i 1-ball bins, j 2-ball bins and k 3-ball bins.
- You take one ball from 1-ball bins. Afterwards, you have (i-1) 1-ball bins, j 2-ball bins and k 3-ball bins.
- You take one ball from 2-ball bins. Afterwards, you have (i+1) 1-ball bins, (j-1) 2-ball bins and k 3-ball bins.
- You take one ball from 3-ball bins. Afterwards, you have i 1-ball bins, (j+1) 2-ball bins and (k-1) 3-ball bins.

$$\begin{aligned} dp[i][j][k] = (dp[i][j][k] + 1) \cdot \frac{N - i - j - k}{N} \\ + (dp[i-1][j][k] + 1) \cdot \frac{i}{N} \\ + (dp[i+1][j-1][k] + 1) \cdot \frac{j}{N} \\ + (dp[i][j+1][k-1] + 1) \cdot \frac{k}{N} \end{aligned}$$

Moving +1 outside of parentheses we get

$$\begin{aligned} dp[i][j][k] = dp[i][j][k] \cdot \frac{N-i-j-k}{N} \\ + dp[i-1][j][k] \cdot \frac{i}{N} \\ + dp[i+1][j-1][k] \cdot \frac{j}{N} \\ + dp[i][j+1][k-1] \cdot \frac{k}{N} \\ + 1 \end{aligned}$$

The issue is that we have dp[i][j][k] on both sides. Just subtract $dp[i][j][k] \cdot \frac{N-i-j-k}{N}$ from both sides.

$$\begin{aligned} dp[i][j][k] \cdot \frac{i+j+k}{N} = dp[i-1][j][k] \cdot \frac{i}{N} \\ + dp[i+1][j-1][k] \cdot \frac{j}{N} \\ + dp[i][j+1][k-1] \cdot \frac{k}{N} \\ + 1 \end{aligned}$$

Mutliplying both sides with $\frac{N}{i+j+k}$ we get

$$dp[i][j][k] = \frac{N}{i+j+k} \left(dp[i-1][j][k] \cdot \frac{i}{N} + dp[i+1][j-1][k] \cdot \frac{j}{N} + dp[i][j+1][k-1] \cdot \frac{k}{N} + 1 \right)$$

We form it further to code it easier:

$$\begin{aligned} dp[i][j][k] = dp[i-1][j][k] \cdot \frac{i}{i+j+k} \\ + dp[i+1][j-1][k] \cdot \frac{j}{i+j+k} \\ + dp[i][j+1][k-1] \cdot \frac{k}{i+j+k} \\ + \frac{N}{i+j+k} \end{aligned}$$

The answer can then be found at dp[#bins having 1 ball][#bins having 2 ball][#bins having 3 ball], which can be calculated in $\mathcal{O}(N^3)$.

Listing 1: Java Code

```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int N = sc.nextInt();
        int ones = 0, twos = 0, threes = 0;
        for (int i = 0; i < N; i++) {
            int x = sc.nextInt();
            if (x == 1) ones++;
            else if (x == 2) twos++;
            else if (x == 3) threes++;
        }

        double[][][] dp = new double[N + 1][N + 1][N + 1];
        for (int i = 0; i <= N; i++) {
            for (int j = 0; j <= N; j++) {
                for (int k = 0; k <= N; k++) {
                    dp[i][j][k] = 0.0;
                }
            }
        }

        for (int k = 0; k <= N; k++) {
            for (int j = 0; j <= N; j++) {
                for (int i = 0; i <= N; i++) {
                    if (i + j + k > N || i + j + k == 0) continue;
                    if (i > 0) dp[i][j][k] += i * dp[i - 1][j][k] / (i + j + k);
                    if (j > 0) dp[i][j][k] += j * dp[i + 1][j - 1][k] / (i + j + k);
                    if (k > 0) dp[i][j][k] += k * dp[i][j + 1][k - 1] / (i + j + k);
                    dp[i][j][k] += (double) N / (i + j + k);
                }
            }
        }

        System.out.printf("%.12f\n", dp[ones][twos][threes]);
    }
}
```

this task is a modified version of https://atcoder.jp/contests/dp/tasks/dp_j