次に選ばれる皿	その確率	遷移後の状態
3 枚のった皿	$\frac{x}{n}$	dp[x-1] [y+1] [z]
2 枚のった皿	$\frac{y}{n}$	dp[x] [y-1] [z+1]
1 枚のった皿	$\frac{z}{n}$	dp[x] [y] [z-1]
0 枚のった皿	$1 - \frac{x+y+z}{n}$	dp[x] [y] [z]

次に選ばれる皿	寿司がのった皿が選ばれる確率	条件付き遷移確率	遷移後の状態
3 枚のった皿	$\frac{x+y+z}{n}$	$\frac{x}{x+y+z}$	dp[x-1] [y+1] [z]
2 枚のった皿		$\frac{y}{x+y+z}$	dp[x] [y-1] [z+1]
1 枚のった皿		$\frac{z}{x+y+z}$	dp[x] [y] [z-1]

Educational DP Contest J - Sushi

- 問題文[*1]
- 提出したコード[*2]

期待値 DP ということなのですが、個人的にすごく難しく感じたので、すこしまとめておきたい。

まず、状態をどう表現するかが、すこし難しい。たとえば、$\mathrm{dp}[i_1][i_2]\cdots[i_n]$ というような状態を持とうとすると、最悪 300 次元配列になってしまい、また、状態の個数も大きくなりすぎてしまいます。

そこで、同じ枚数の寿司が乗っている皿同士を区別する必要がないことに着目すると、以下のように表現することができます：

- dp[x] [y] [z]：残り 3 つの皿が x 枚、残り 2 つの皿が y 枚、残り 1 つの皿が z 枚である状態から、すべてなくすのに必要な試行回数の期待値

このままでは、右辺左辺両方に dp[x, y, z] があるような漸化式になってしまうため、遷移条件を P(1 まい以上のった皿を選ぶ確率）と、その条件のもとで k 枚のった皿をえらぶ確率にわけて考えてみます。

この表と、寿司がのった皿が選ばれるまでの試行回数の期待値が、選ばれる確率の逆数であることを用いて、状態更新は以下のように書くことができます：

```
dp[x,y,z] = 1.0 * n / (x + y + z);
if (x > 0)   dp[x,y,z] += dp[x-1, y+1, z] * x/(x+y+z);
if (y > 0)   dp[x,y,z] += dp[x, y-1, z+1] * y/(x+y+z);
if (z > 0)   dp[x,y,z] += dp[x, y, z-1] * z/(x+y+z);
```

状態の更新順序が複雑なので、これをメモ化再帰で求めることにします。そのコード全体は、以下の通り：

[*1] https://atcoder.jp/contests/dp/tasks/dp_j
[*2] https://atcoder.jp/contests/dp/submissions/7716070

```
using System;
using System.Collections.Generic;
using System.Linq;
using static System.Console;
using System.Runtime.CompilerServices;
using static MyUtil;

class MyUtil
{
    public static int[] ReadIntArray()
    {
return ReadLine().Split().Select(x => int.Parse(x)).ToArray();
    }
}

class Program
{
    static int n;
    static double[,,] dp;

    // メモ化再帰
    static double rec(int x, int y, int z)
    {
if (dp[x,y,z] != -1.0) return dp[x,y,z];

// 空でない皿にあたるまでの試行回数の期待値
double r = 1.0 * n / (x + y + z);

// 条件付きの遷移確率 * 遷移後の期待値
if (x > 0)   r += rec(x-1, y+1, z) * x/(x+y+z);
if (y > 0)   r += rec(x, y-1, z+1) * y/(x+y+z);
if (z > 0)   r += rec(x, y, z-1) * z/(x+y+z);

return dp[x,y,z] = r;
    }

    public static void Main()
    {
n = int.Parse(ReadLine());
```

```
int[] a = ReadIntArray();
int[] c = new int[4];
for (int i = 0; i < n; i++) c[a[i]]++;

dp = new double[301,301,301];
for (int i = 0; i < 301; i++)
    for (int j = 0; j < 301; j++)
for (int k = 0; k < 301; k++)
    dp[i,j,k] = -1.0;

dp[0,0,0] = 0;
WriteLine(rec(c[3],c[2],c[1]));
    }
}
```

別の考え方

最初の表のように、遷移前後に同じ状態を含むままで漸化式を書くと以下ののようになります。一回試行して次の状態になるので、次の状態の期待値に 1 を足した値になるのですが、それらを、それぞれの遷移確率で重みづけして足し合わせてあります：

```
dp[x,y,z] = (x/n) * (dp[x-1,y+1,z] + 1)
          + (y/n) * (dp[x,y-1,z+1] + 1)
          + (z/n) * (dp[x,y,z-1] + 1)
          + (1 - (x+y+z)/n) * (dp[x,y,z] + 1)
```

これを式変形（右辺の dp[x,y,z] を移項して左辺によせる）しても、同様の上記と同様の式が得られます。