DAY1 solution

WerKeyTom_FTD

January 6, 2018

前言

相信按照正常操作应该没有人AK。

Hello my friend

题目来源：改编自codeforces Helvetic Coding Contest 2017 online mirror - problem L
题目难度确实与顺序无关，但实际上最简单的题确实放在了第一个位置。
本题单看题面很像全场最难。实际定位是签到题。

算法一

设$dp(i,s)$表示当前在点i已经经过的白点集合为s，从该状态到结束的期望计数器值。
可以按照s分层高斯消元，那么可以获得前两个点的分值。

算法二

不妨考虑只有黑点怎么做。
以1为根建有根树，令p_i表示节点i的父亲，f_i表示从节点i到结束的期望计数器值，deg_i表示点i的度数。
可以发现如果$deg_i = 1$那么$f_i = 1$。
如果$deg_i > 1$那么$f_i = 1 + \frac{1}{deg_i} \sum_{i->j} f_j$。
不妨将f_i表示成$f_i = kk_i * f_{p_i} + bb_i$。
做一遍树形DP，那么bb_1即为答案。

算法三

不妨考虑只有白点怎么做。

可以发现一个白点只要被经过了就会带了1的贡献，因此我们需要算出点i被经过的概率dp_i。

容易发现经过i一定经过p_i，设g_i表示从点p_i走到i的概率，那么$dp_i = dp_{p_i} * g_i$。

考虑如何求g，设f_i表示从点i走到点p_i的概率，那么$g_i = \frac{1}{deg_{p_i}} * (1 + g_{p_i} * g_i + \sum_j f_j * g_i)$。这里的$j$指$p_i$的一个非$i$儿子。

只要求出了f求g便很方便，$f_i = \frac{1}{deg_i} * (1 + \sum_j f_j * f_i)$，这里$j$指$i$的一个儿子，可以发现$f$更好求。

算法四

不妨考虑黑白点都有怎么做。
将算法二稍加改动，结合算法三即可。

Every one will meet some difficult

题目来源：加强自TCO2013 3A。
这确实是一道比较难的题目。

题目大意

转换题面模型得。
对于$1 \le i \le n$有$1 \le x_i \le t$。
$\sum_{i=1}^{m} x_i \le s$。
求方程的正整数解数量。

算法一

可以设一个简单dp，$f[i][j]$表示前i个数和为j的方案数。

算法二

考虑暴力枚举前n个变量中多少有个超过了t的限制，发现这样并不好算。于是我们可以枚举前n个变量中至少有x个超过了t的限制，这样的方案数是$\binom{n}{x} * \binom{S-xt}{m}$。
因此$ans = \sum_{i=0}^{n}(-1)^i\binom{n}{i}\binom{S-it}{m}$。
组合数可以线性递推得出。

算法三

考虑暴力枚举前n个变量的取值，后面的部分用组合数计算，因为保证$nt \le S$，所以前面的n个变量是可以任意取值的。

算法三

考虑暴力枚举前n个变量的取值，后面的部分用组合数计算，因为保证$nt \le S$，所以前面的n个变量是可以任意取值的。
$ans = \sum_{x_1=1}^{t} \sum_{x_2=1}^{t} \cdots \sum_{x_n=1}^{t} \binom{S-x_1-x_2-\cdots-x_n}{m-n}$。

算法三

考虑暴力枚举前n个变量的取值，后面的部分用组合数计算，因为保证$nt \le S$，所以前面的n个变量是可以任意取值的。
$ans = \sum_{x_1=1}^{t} \sum_{x_2=1}^{t} \cdots \sum_{x_n=1}^{t} \binom{S-x_1-x_2-\cdots-x_n}{m-n}$。
令$X = \sum_{i=1}^{n} x_i$，则注意到后面那个组合数：
$\binom{S-X}{m-n} = \frac{1}{(m-n)!}(S-X)^{\underline{(m-n)}}$。
我们可以将后面部分用$O((m-n)^2)$的时间展开成一个$m-n$次的多项式。

算法三

令$F(x) = \binom{S-x}{m-n} = \sum_{i=0}^{m-n} a_i * x_i$。

算法三

令$F(x) = \binom{S-x}{m-n} = \sum_{i=0}^{m-n} a_i * x_i$。

$ans = \sum_{x_1=1}^{t} \sum_{x_2=1}^{t} \cdots \sum_{x_n=1}^{t} F(\sum_{i=1}^{n} x_i)$

$= \sum_{x_1=1}^{t} \sum_{x_2=1}^{t} \cdots \sum_{x_n=1}^{t} \sum_{j=0}^{m-n} a_j * (\sum_{i=1}^{n} x_i)^j$

$= \sum_{j=0}^{m-n} a_j \sum_{x_1=1}^{t} \sum_{x_2=1}^{t} \cdots \sum_{x_n=1}^{t} (\sum_{i=1}^{n} x_i)^j$

算法三

令$F(x) = \binom{S-x}{m-n} = \sum_{i=0}^{m-n} a_i * x_i$。

$ans = \sum_{x_1=1}^{t} \sum_{x_2=1}^{t} \cdots \sum_{x_n=1}^{t} F(\sum_{i=1}^{n} x_i)$

$= \sum_{x_1=1}^{t} \sum_{x_2=1}^{t} \cdots \sum_{x_n=1}^{t} \sum_{j=0}^{m-n} a_j * (\sum_{i=1}^{n} x_i)^j$

$= \sum_{j=0}^{m-n} a_j \sum_{x_1=1}^{t} \sum_{x_2=1}^{t} \cdots \sum_{x_n=1}^{t} (\sum_{i=1}^{n} x_i)^j$

那么我们现在需要求出$\sum_{x_1=1}^{t} \sum_{x_2=1}^{t} \cdots \sum_{x_n=1}^{t} (\sum_{i=1}^{n} x_i)^j$。

算法三

设$g_{i,j}=\sum_{x_1=1}^{t}\sum_{x_2=1}^{t}\cdots\sum_{x_i=1}^{t}(\sum_{k=1}^{n}x_k)^j$。
那么：
$g_{i,j}=\sum_{x=1}^{t}\sum_{k=0}^{j}x^k g_{i-1,j-k}\binom{j}{k}$
$=\sum_{k=0}^{j}g_{i-1,j-k}\binom{j}{k}\sum_{x=1}^{t}x^k$

算法三

设$g_{i,j} = \sum_{x_1=1}^{t} \sum_{x_2=1}^{t} \cdots \sum_{x_i=1}^{t} (\sum_{k=1}^{n} x_k)^j$。
那么：
$g_{i,j} = \sum_{x=1}^{t} \sum_{k=0}^{j} x^k g_{i-1,j-k} \binom{j}{k}$
$= \sum_{k=0}^{j} g_{i-1,j-k} \binom{j}{k} \sum_{x=1}^{t} x^k$
很显然$j \leq m-n$，因此我们需要预处理$m-n$个1到t的自然数幂和，可以用插值法或斯特林数做到$O((m-n)^3)$或$O((m-n)^2)$。

算法三

设$g_{i,j} = \sum_{x_1=1}^{t} \sum_{x_2=1}^{t} \cdots \sum_{x_i=1}^{t} (\sum_{k=1}^{n} x_k)^j$。
那么：
$g_{i,j} = \sum_{x=1}^{t} \sum_{k=0}^{j} x^k g_{i-1,j-k} \binom{j}{k}$
$= \sum_{k=0}^{j} g_{i-1,j-k} \binom{j}{k} \sum_{x=1}^{t} x^k$
很显然$j \le m-n$，因此我们需要预处理$m-n$个1到t的自然数幂和，可以用插值法或斯特林数做到$O((m-n)^3)$或$O((m-n)^2)$。
接下来考虑如何求g_n，不妨使用矩阵乘法。
整个算法复杂度为$O((m-n)^3\ log\ n)$。

算法四

不妨考虑从算法二进行优化。
回忆式子，$ans=\sum_{i=0}^{n}(-1)^i\binom{n}{i}\binom{S-it}{m}$
令$f(x)=\binom{S-xt}{m}$。
则$ans=\sum_{i=0}^{n}(-1)^i\binom{n}{i}f(i)$
这还能优化吗？

n阶差分公式

我们先来介绍一下n阶差分公式。
对于f，定义$\Delta f = (f - [x^0]f)/x - f$。
$\Delta^n f$则指差分了n次。
满足$\Delta^n f(x) = \sum_{i=0}^{n} (-1)^{n-i} \binom{n}{i} f(x+i)$。
这就是n阶差分公式。

证明n阶差分公式

不妨用归纳法来证明。首先$n=0$时显然成立。
现在我们说明如果n时成立$n+1$时也成立。
$\Delta^{n+1}f(x)=\Delta^n f(x+1)-\Delta^n f(x)$
$=\sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}f(x+i+1)-\sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}f(x+i)$
$=(-1)^0\binom{n}{n}f(x+n+1)+(-1)^n\binom{n}{0}f(x)+$
$\sum_{i=1}^{n}(-1)^{n-i+1}(\binom{n}{i-1}+\binom{n}{i})f(x+i)$
$=\sum_{i=0}^{n+1}(-1)^{n-i+1}\binom{n+1}{i}f(x+i)$

利用n阶差分公式

回忆答案$ans = \sum_{i=0}^{n}(-1)^i\binom{n}{i}f(i)$。
n阶差分公式$\Delta^n f(x) = \sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}f(x+i)$。

利用n阶差分公式

回忆答案$ans=\sum_{i=0}^{n}(-1)^i\binom{n}{i}f(i)$。
n阶差分公式$\Delta^n f(x)=\sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}f(x+i)$。
有没有发现长得很像？我们令$Ans=(-1)^n*ans$。
$Ans=\sum_{i=0}^{n}(-1)^{n+i}\binom{n}{i}f(i)$
$=\sum_{i=0}^{n}(-1)^{n-i}\binom{n}{i}f(i)$
$=\Delta^n f(0)$

下面我们先介绍一下两类斯特林数。
第一类斯特林数是下降幂展开后的各项系数的绝对值，即：
$n^{\underline{m}} = \sum_{i=0}^{m} \begin{bmatrix} m \\ i \end{bmatrix} (-1)^{m-i} n^i$。

第一类斯特林数

下面我们先介绍一下两类斯特林数。
第一类斯特林数是下降幂展开后的各项系数的绝对值，即：
$n^{\underline{m}} = \sum_{i=0}^{m} \left[{m \atop i}\right] (-1)^{m-i} n^i$。
根据这个定义可以推出第一类斯特林数的递推式，其与“将n个可区分元素划分成m个圆排列计数”问题的递推式一致。
因此第一类斯特林数组合意义就是这个。

第二类斯特林数的组合意义是“将n个可区分元素划分成m个集合计数”。
来考虑这样一个组合问题，将m个可区分的球分别扔到n个可区分箱子中的一个，显然方案数为n^m。

第二类斯特林数的组合意义是“将n个可区分元素划分成m个集合计数”。
来考虑这样一个组合问题，将m个可区分的球分别扔到n个可区分箱子中的一个，显然方案数为n^m。
我们不妨枚举哪i个箱子里有球，那么可以把m个球分成i份，然后选择n个箱子中的i个（这里是排列不是组合）。
则这个方法也能统计出答案，可以得到公式$n^m = \sum_{i=0}^{m} \left\{ m \atop i \right\} n^{\underline{i}}$。

化简 f

$$f(x) = \binom{S-xt}{m}$$
$$= \frac{1}{m!}(S-xt)^{\underline{m}}$$

化简 f

$$f(x) = \binom{S-xt}{m}$$
$$= \frac{1}{m!}(S-xt)^{\underline{m}}$$
$$= \frac{1}{m!}\sum_{i=0}^{m} \begin{bmatrix} m \\ i \end{bmatrix} (-1)^{m-i}(S-xt)^i$$

化简 f

$$f(x) = \binom{S-xt}{m}$$
$$= \frac{1}{m!}(S-xt)^{\underline{m}}$$
$$= \frac{1}{m!}\sum_{i=0}^{m} \left[{m \atop i}\right](-1)^{m-i}(S-xt)^{i}$$
$$= \frac{1}{m!}\sum_{i=0}^{m} \left[{m \atop i}\right](-1)^{m-i}\sum_{j=0}^{i}(-1)^{j}\binom{i}{j}S^{i-j}t^{j}x^{j}$$

化简f

$$f(x)=\binom{S-xt}{m}$$
$$=\frac{1}{m!}(S-xt)^{\underline{m}}$$
$$=\frac{1}{m!}\sum_{i=0}^{m}\begin{bmatrix}m\\i\end{bmatrix}(-1)^{m-i}(S-xt)^{i}$$
$$=\frac{1}{m!}\sum_{i=0}^{m}\begin{bmatrix}m\\i\end{bmatrix}(-1)^{m-i}\sum_{j=0}^{i}(-1)^{j}\binom{i}{j}S^{i-j}t^{j}x^{j}$$
$$=\frac{1}{m!}\sum_{j=0}^{m}x^{j}\sum_{i=j}^{m}(-1)^{m-i+j}\begin{bmatrix}m\\i\end{bmatrix}\binom{i}{j}S^{i-j}t^{j}$$

令$a_j=\frac{1}{m!}\sum_{i=j}^{m}(-1)^{m-i+j}\begin{bmatrix}m\\i\end{bmatrix}\binom{i}{j}S^{i-j}t^{j}$，则$f(x)=\sum_{j=0}^{m}x^{j}a_j$。

继续化简 f

$f(x) = \sum_{i=0}^{m} a_i x^i$

继续化简 f

$$f(x) = \sum_{i=0}^{m} a_i x^i$$
$$= \sum_{i=0}^{m} a_i \sum_{j=0}^{i} \left\{ {i \atop j} \right\} \binom{x}{j} j!$$

继续化简f

$f(x) = \sum_{i=0}^{m} a_i x^i$
$= \sum_{i=0}^{m} a_i \sum_{j=0}^{i} \left\{ {i \atop j} \right\} \binom{x}{j} j!$
$= \sum_{j=0}^{m} \binom{x}{j} \sum_{i=j}^{m} a_i \left\{ {i \atop j} \right\} j!$
令$b_j = \sum_{i=j}^{m} a_i \left\{ {i \atop j} \right\} j!$，则$f(x) = \sum_{j=0}^{m} \binom{x}{j} b_j$。

特殊形式差分的结论

如果$f(x) = \sum_{i=0}^{d} c_i \binom{x}{i}$。
那么$\Delta^n f(x) = \sum_{i=0}^{d} c_i \binom{x}{i-n}$。

特殊形式差分的结论

如果$f(x)=\sum_{i=0}^{d} c_i\binom{x}{i}$。
那么$\Delta^n f(x)=\sum_{i=0}^{d} c_i\binom{x}{i-n}$。
还是考虑归纳证明，显然$n=0$成立，然后我们说明n成立时$n+1$也成立。

特殊形式差分的结论

如果$f(x)=\sum_{i=0}^{d}c_i\binom{x}{i}$。
那么$\Delta^n f(x)=\sum_{i=0}^{d}c_i\binom{x}{i-n}$。
还是考虑归纳证明，显然$n=0$成立，然后我们说明n成立时$n+1$也成立。

$$\Delta^{n+1}f(x)=\Delta^n f(x+1)-\Delta^n f(x)$$
$$=\sum_{i=0}^{d}c_i[\binom{x+1}{i-n}-\binom{x}{i-n}]$$
$$=\sum_{i=0}^{d}c_i\binom{x}{i-n-1}$$

考虑对f差分

我们知道$Ans = \Delta^n f(0)$，现在我们考虑差分f。

考虑对f差分

我们知道$Ans = \Delta^n f(0)$，现在我们考虑差分f。
注意$f(x) = \sum_{i=0}^m \binom{x}{i} b_i$这和上一页讲的形式一致。
因此$\Delta^n f(x) = \sum_{i=0}^m \binom{x}{i-n} b_i$。
可得$\Delta^n f(0) = b_n$。
根据之前的定义展开b_n：
$Ans = \frac{n!}{m!} \sum_{i=n}^m \left\{ {i \atop n} \right\} \sum_{j=i}^m (-1)^{m-i+j} \binom{j}{i} \left[{m \atop j} \right] S^{j-i} t^i$

我们知道$Ans = \Delta^n f(0)$，现在我们考虑差分f。
注意$f(x) = \sum_{i=0}^{m} \binom{x}{i} b_i$这和上一页讲的形式一致。
因此$\Delta^n f(x) = \sum_{i=0}^{m} \binom{x}{i-n} b_i$。
可得$\Delta^n f(0) = b_n$。
根据之前的定义展开b_n：
$Ans = \frac{n!}{m!} \sum_{i=n}^{m} \left\{ {i \atop n} \right\} \sum_{j=i}^{m} (-1)^{m-i+j} \binom{j}{i} \left[{m \atop j} \right] S^{j-i} t^i$
如果能提前得到所需要用到的两类斯特林数，可以用$O((m-n)^2)$计算该式子。

预处理理斯特林数

我们知道正常递推斯特林数$\left[{m \atop n}\right]\left\{{m \atop n}\right\}$都需要$O(m^2)$。
注意到本题$m-n$不大，考虑用组合意义去理解。
以第一类斯特林数为例，第二类类似。
将m个可区分元素划分成n个圆排列，首先这n个圆排列内都有至少一个元素，然后还剩下$m-n$个元素要插入圆排列中。
因此元素超过1个的圆排列个数不超过$m-n$个，这些圆排列里的元素和不会超过$2(m-n)$个。
考虑dp，设$f[i][j]$表示将i个元素放入j个元素个数超过1的圆排列的方案数。
那么$f[i][j]=f[i-1][j]*(i-1)+f[i-2][j-1]*(i-1)$。
最终计算$\left[{m \atop n}\right]$时可以枚举将$m-n$个其余的元素插入了几个圆排列中。
至此整道题我们用$O((m-n)^2)$的复杂度解决了。

Try to find out the wrong in the test

题目来源：PA2014。
你对本题的坚持，说不定有错。

算法一

设f_i表示将1到i分组的最大值，g_i表示方案数，接下来我们只考虑f_i，g_i是类似的。

每次枚举最后一组的开头即可。

首先观察d的限制，可以通过预处理$left_i$表示所有$left_i \le j < i$均满足$j+1$到i的d最小值不比$i-j+1$小。
那么只有这些j有可能可以转移到i。
这个$left$显然有单调性，于是预处理$left$的时候，只需要一个单调队列来资瓷尾加头删的最小值维护即可。

算法二

似乎存在$O(n\ log^2\ n)$的做法然而我好像不会。

算法三

如何有更优秀的做法？
我们来考虑c，发现并不是很好考虑？
于是我们来考虑分治，根据c来分治。
这样可以在每一次转移明确c的最大值。
具体的，我们考虑现在确定$[l,r]$的f值，记为过程$solve(l,r)$。

分治

我们先找到$[l+1,r]$中c的最大值所在位置k。
然后我们考虑用$[l,k-1]$的状态转移到$[k,r]$。
$[l,k-1]$的状态可以先通过$solve(l,k-1)$确定。
这个转移完成后可以继续调用$solve(k,r)$。
于是我们来解决这个转移。

转移

我们来考虑$[max(k,l+c),r]$的i。
显然只有这个区间的i能被$[l,k-1]$转移到。
分类讨论。

分类讨论

1、$left_i < l$且$i \le k-1+c$。
满足条件的i是连在一起的，而且$i+1$相对于i来说，只不过多了一个$i+1-c$。
因此对于第一个i用线段树查询后，接下来的都可以$O(1)$更新。

分类讨论

2、$left_i < l$且$i > k - 1 + c$。
这个时候每个i的合法范围都是$[l, k-1]$。
二分出满足条件的i的区间$[ll, rr]$。
用线段树查询出$[l, k-1]$的答案，然后打标记给$[ll, rr]$。

分类讨论

3、$l \leq left_i < k$。
这个时候因为$left_i$不确定，对于这样的每个i我们只能分别求解，在线段树对应区间求出答案。
4、$left_i \geq k$。
那就没有$[l, k-1]$的可以转移啦，退出吧！
如果对转移这部分的代码实现有疑问，可以参考标程实现。下面我们来分析这样做的复杂度。

复杂度分析

先看情况1，首先要一个$log\ n$，接下来与扫过的i个数有关。

复杂度分析

先看情况1，首先要一个$log\ n$，接下来与扫过的i个数有关。
假设满足条件的最小i是k，最大i是$k-1+c$或r，我们直接当做最大i是r一定不比真实最大i小。
此时扫过的i个数是$[k,r]$的区间长度。

复杂度分析

先看情况1，首先要一个$log\ n$，接下来与扫过的i个数有关。
假设满足条件的最小i是k，最大i是$k-1+c$或r，我们直接当做最大i是r一定不比真实最大i小。
此时扫过的i个数是$[k,r]$的区间长度。
假设满足条件的最小i是$l+c$，最大i是$k-1+c$或r，我们直接当做最大i是$k-1+c$一定不比真实最大i小。
此时扫过的i个数是$[l,k-1]$的区间长度。

复杂度分析

先看情况1，首先要一个$log\ n$，接下来与扫过的i个数有关。
假设满足条件的最小i是k，最大i是$k-1+c$或r，我们直接当做最大i是r一定不比真实最大i小。
此时扫过的i个数是$[k,r]$的区间长度。
假设满足条件的最小i是$l+c$，最大i是$k-1+c$或r，我们直接当做最大i是$k-1+c$一定不比真实最大i小。
此时扫过的i个数是$[l,k-1]$的区间长度。
那么到底是多少呢？我们发现最小i是k与$l+c$取max，那么扫过的i个数应当不超过两者的较小值。

两者分别是$[l, k-1]$的区间长度与$[k, r]$的区间长度。
因此关于情况1的复杂度可以写
作$T(n) = T(x) + T(n-x) + log\ n + min(x, n-x)$。
考虑到$log\ n$的影响只有n个，这部分是$n\ log\ n$的。
后面那个可以理解为启发式合并，于是$T(n) = n\ log\ n$。

复杂度分析

接下来看情况2，显然每次只需要一个$log\ n$，分治结构有n个部分，那么就是$n\ log\ n$。
接下来看情况3，对于每个i均需要一个$log\ n$，但由于每个i显然只会被这样搞一次（分治结构中不同的部分要么$left$取值范围不相交要么被搞的i范围不相交），所以也是$n\ log\ n$。
于是总复杂度为$O(n\ log\ n)$！
当分治到$l = r$时，就可以在线段树中查出f和g。