算法学习笔记(7)：球盒模型

GhostLX
ACMer蒟蒻 / 北航研0

发布于 2023-02-01 00:42 · IP 属地福建 ，编辑于 2024-02-09 07:46 · IP 属地福建

在做算法题的过程中，很多**计数类问题**的很多情况可以转化为如下**模型**：n个**相同/不同**的球放到k个**相同/不同**的盒子中，盒子**允许/不允许**为空，问有多少种方案。

8个模型过于常见，本文进行总结，以及附带了它们的**拓展模型**，如**每个盒子至少t个球**、**每个盒子至多t个球**、**非空盒子不能相邻**等等，另外本文还附带了两道相关例题。

有如下文字需要先悉知：

- "无标号" = "相同"
- "有标号" = "不同".

共8种模型，为了方便记忆本文模型的编号：

- 将“无标号”、“允许为空”看为0，“有标号”、“不允许为空”看为1。
- 如对于模型4，有$(100)_2=4$，表示：n个**有标号**的球放到k个**无标号**的盒子中，盒子**不允许为空**。

模型

模型0

> n个**无标号**的球放到k个**无标号**的盒子中，盒子**允许为空**。

容易得到当没有球时（$n=0$）方案数为1，当没有盒子时（$k=0$）方案为0。

考虑递归求解。易得递归边界为$b_0(0,k)=1, b(n,0)=0$，注意此处没有没球的优先级高于没有箱子。

当$n<k$时，必然存在 $k-n$ 个盒子为空，因为盒子是相同的，所以空出来哪些盒子是无所谓的。所以我们可以缩小问题规模：$b_0(n,k)=b_0(n,n)$

当$n\geq k$时，因为所有盒子都是相同的，所以我们只需要讨论有几个箱子是空的，而不需要具体讨论空的是哪些箱子。

- 没有箱子为空的时，将k个箱子每个都放入一个球，方案数为$b_0(n-k,k)$
- 至少一个箱子为空时，先钦定一个箱子为空，方案数为$b_0(n,k-1)$

于是我们就可以得到如下代码：

```
int b0(int n, int k)
{
    if(n == 0)
        return 1;
    if(k == 0)
        return 0;
    if(n < k)
        return b0(n, n);
    else
        return b0(n - k, k) + b0(n, k - 1);
}
```

模型1

> n个**无标号**的球放到k个**无标号**的盒子中，盒子**不允许为空**。

模型1跟模型0的差别在于盒子是否可以为空，只需在每个盒子中都先放置一个小球，保证了非空，然后任意放置剩下的球。

这样也就可以将模型1转化为模型0，方案数为b_1(n,k)=b_0(n-k,k)。

需要注意n<k时方案数量为0。

```
int b1(int n, int k)
{
    if(n < k)
        return 0;
    else
        return b0(n - k, k);
}
```

模型2：隔板法（插板法）

> n个**无标号**的球放到k个**有标号**的盒子中，盒子**允许为空**。
> 本质：x_1+x_2+\cdots+x_k=n的非负整数解的组数。

也就是最经典的**隔板法**。因为球是相同的，我们只需要考虑每个箱子放几个球即可。

将n个球排成一行，然后拿出k-1块木板，将木板插在球的前面或者后面，运行多个木板紧挨着。然后第i(i>1)个木板到第i-1个木板之间的球，当作第i个盒子内球的数量，若没有球则认为第i个盒子为空。特别地，第1块木板前的球当作第1个箱子的球的数量，第k-1块木板之后的球当作第k个箱子的球的数量。

木板是相同的，球也是相同的。每种木板的放置方法，对应一种放球方案。

木板和球都看成“物品”，总共有n+k-1个物品，相当于从中选择出k-1个物品，让它们成为木板。

故总的方案数为b_2(n,k)=\dbinom {n+k-1} {k-1}

```
const int N = 1005, md = 1e9 + 7;
LL c[N][N];
void initC()
{//预处理组合数
    c[0][0] = 1;
    LL R = N - 5;
    for(int i = 1; i <= R; i ++ )
    {
        c[i][0] = 1;
        for(int j = 1; j <= i; j ++)
            c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % md;
    }
}

int b2(int n, int k)
{
    return c[n + k - 1][k - 1];
}
```

模型3

> n个**无标号**的球放到k个**有标号**的盒子中，盒子**不允许为空**。
> 本质：x_1+x_2+\cdots+x_k=n的正整数解的组数。

对比模型2，区别在于盒子是否能为空

同样的方法，将n个球排成一行，然后拿出k-1块木板。

不过因为此处盒子不允许为空，所以需要只能插在球与球之间的间隔，且木板不能相邻。

也就是在n-1个球与球的间隔中，选择k-1个插入木板

故总的方案数为$b_3(n,k)=\binom {n-1}{k-1}$

```
const int N = 1005, md = 1e9 + 7;
LL c[N][N];
void initC()
{//预处理组合数
    c[0][0] = 1;
    LL R = N - 5;
    for(int i = 1; i <= R; i ++ )
    {
        c[i][0] = 1;
        for(int j = 1; j <= i; j ++)
            c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % md;
    }
}

int b3(int n, int k)
{
    return c[n - 1][k - 1];
}
```

模型4

> n个**有标号**的球放到k个**无标号**的盒子中，盒子**允许为空**。

前置知识：请先阅读 模型5（斯特林子集数）

模型4和模型5的差别，在于本处盒子允许为空。

从斯特林子集数的角度来考虑。因为盒子是相同的，所以我们枚举空了几个盒子就行：
$b_4(n,k)=\sum_{i=0}^{k-1} {n\brace k - i}$

- 预处理斯特林数的复杂度为$O(n\times k)$，求和复杂度为$O(k)$。

```
LL S[N][K];
void initS2()
{//预处理斯特林数
    S[0][0] = 1;
    for(int i = 1; i < N; i ++ )
        for(int j = 1; j < K; j ++)
            S[i][j] = (S[i - 1][j - 1] + S[i - 1][j] * j % md) % md;
}

int b4(int n, int k)
{
    LL res = 0;
    for(int i = 0; i <= k - 1; i ++)
        res = (res + S[n][k - i]) % md;
    return res;
}
```

特别地，当$n\le k$时，答案即为贝尔数（Bell数）。箱子是相同的，故不需要考虑空出来的是哪些箱子。即$b_4(n,k)=B_n$

对于贝尔数，有如下递推式$B_{n+1}=\sum_{k=0}^n \binom n k B_k$。我们可以通过这个式子来构造出贝尔三角形，类似“杨辉三角形”，因为二者都含有组合数。构造出来的每行的首项，就是贝尔数，即b_4(n,k)=B_n=\text{bell

- 预处理贝尔数的复杂度为$\mathcal O(n^2)$，直接拿取答案输出为$\mathcal O(1)$

```
const int N = 2000 + 5;
int bell[N][N];

void initBell(int n) {
  bell[1][1] = 1;
  for (int i = 2; i <= n; i++) {
    bell[i][1] = bell[i - 1][i - 1];
    for (int j = 2; j <= i; j++)
      bell[i][j] = bell[i - 1][j - 1] + bell[i][j - 1];
  }
}
```

模型5：斯特林子集数

> n个**有标号**的球放到k个**无标号**的盒子中，盒子**不允许为空**。

即第二类斯特林数${n \brace k}$（斯特林子集数）。

- 将n个不同的元素，划分为k个非空子集的方案数。

考虑动态规划，假设目前已有k个集合，对于一个新来的数，一种是加入到已有的k个集合，另一种是这个数单独创建一个集合。所以有：

${n \brace k}={n-1 \brace k-1}+{n-1 \brace k}\times k$。

预处理斯特林数的复杂度为$O(n\times k)$。答案即为$b_5(n,k)={n\brace k}$

```
LL S[N][N];
void initS2()
{//预处理斯特林数
    S[0][0] = 1;
    int R = N - 5;
    for(int i = 1; i <= R; i ++ )
        for(int j = 1; j <= R; j ++)
            S[i][j] = (S[i - 1][j - 1] + S[i - 1][j] * j % md) % md;
}

int b5(int n, int k)
{
    return S[n][k];
}
```

模型6

> n个**有标号**的球放到k个**有标号**的盒子中，盒子**允许为空**。

因为盒子可以看成一个无序集合（内部不区分顺序），所以可以单独考虑每个球去哪个盒子，两两之间相互独立。每个球有k种选择，共n个球

故总的方案数为$b_6(n,k)=k^n$

```
LL fpow(LL a, int b)
{
    LL res = 1;
    while(b)
    {
        if(b & 1)
            res = res * a % md;
        a = a * a % md;
        b >>= 1;
    }
```

```
    return res;
}

int b6(int n, int k)
{
    return fpow(k, n);
}
```

模型7

> n个**有标号**的球放到k个**有标号**的盒子中，盒子**不允许为空**。

对比模型5（斯特林子集数），差别在于该盒子是否相同。

在模型5中，我们可以理解为，将一些球拿出来，捆绑起来放入一个袋子，然后再将袋子放入一个盒子中，因为盒子都是相同的，所以放入那个盒子无关紧要。

在本模型中，盒子不同，我们先将第一个袋子拿出来，有k个盒子可以选，然后再将第2个箱子拿出来，有k-1个盒子可以选，……，最后可以得到即在模型5的基础上，乘以一个k!即可。

故总的方案数为b_7={n \brace k}\times k!

```
LL S[N][N], fac[N];
void init()
{//预处理斯特林数
    S[0][0] = 1, fac[0] = 1;
    int R = N - 5;
    for(int i = 1; i <= R; i ++ )
        for(int j = 1; j <= R; j ++)
            S[i][j] = (S[i - 1][j - 1] + S[i - 1][j] * j % md) % md;
    for(int i = 1; i <= R; i ++ )
        fac[i] = fac[i - 1] * i % md;
}

int b7(int n, int k)
{
    return S[n][k] * fac[k];
}
```

拓展模型

约定拓展模型标号，包含原本模型是哪个（用数字表示），以及是第几个拓展模型（用大写字母表示）

模型1A

> n个**无标号**的球放到k个**无标号**的盒子中，盒子**不允许为空**。每个盒子里**至少**有t个球。

考虑动态规划，也就是整数分拆了，约定dp_{n,k,t}为将整数n分拆为k个数相加，每个数至少为t的方案数。

有状态转移方程：dp_{n,k,t}=\sum_{i=t}^n dp_{n-i,k-1,i}

- 从n中先划分出一个数i，然后剩余部分n-i划分成k-1个数相加，每个数至少为i。
- 这样方案是不会**计算重复**的，那么为什么**不能是**剩余部分n-i的划分是每个部分至少t呢？这样看似好像也符合答案？
- 我们假设某种将n分解为k个数的方案为：x_1,x_2,\cdots,x_k，其中满足x_i\le x_{i+1}，也就是满足无序集合的要求。
- 假设我们单独划分出x_1，那么x_2,x_3,\cdots,x_k是剩余部分的划分，满足每个数至少为x_1
- 假设我们单独划分出x_2，那么x_1,x_3,\cdots,x_k是剩余部分的划分，满足每个数至少为x_1
- 于是，如果转移方程右边，让剩余i
  被**重复计算**，所以不行。

- 但是，划分了i出去之后，约定了每个划分的最小的数为i，这样就保证了一个序列只会被统计一次，毕竟每个划分序列中的最小值**有且仅有一个**

注意递归边界为：

- $n<k\times t$时，明显不够分了，所以此时方案数为0
- 当$k=0$时，显然分为0个数相加是不可能的，方案为0
- 当$k=1$时，一个数就是一个分解方案，方案为1

代码采用记忆化搜索完成

```
LL dp[N][N][N];
LL dfs(int n, int k, int t)
{
    if(n < k * t)
        return 0;
    if(k == 0)
        return dp[n][k][t] = 0;
    if(k == 1)
        return dp[n][k][t] = 1;
    if(~dp[n][k][t])
        return dp[n][k][t];
    dp[n][k][t] = 0;
    for(int i = t; i <= n; i ++ )
    {
        if(k * i > n)
            break;
        dp[n][k][t] += dfs(n - i, k - 1, i);
        dp[n][k][t] %= md;
    }
    return dp[n][k][t];
}

int main()
{
    memset(dp, -1, sizeof dp);
    int n, k, t;
    cin >> n >> k >> t;
    cout << dfs(n, k, t);
}
```

模型2A

> n个**无标号**的球放到k个**有标号**的盒子中，盒子**允许为空**。$a_1,\cdots,a_k$是给定的非负整数的数组，要求第i个盒子中至少含有a_k个球。
> 本质：$x_1+x_2+\cdots+x_k=n$且$x_i\ge a_i$的非负整数解的组数。

和模型2的差别在于，每个盒子有最少球数的限制，我们尝试去扣除掉这个限制，我们现往每个盒子中放置a_i个球，然后我们就可以任意放置球了，也就把问题又转化为了模型2。

还剩余$n-\sum_{i=1}^k a_i$个球，依旧为k-1块板子，隔板法公式得最后答案为$\binom {n-(\sum_{i=1}^k a_i)+k-1}{k-1}$

模型2B

> n个**无标号**的球放到k个**有标号**的盒子中，盒子**允许为空**。每个盒子里**至多**t个球。

考虑生成函数。

- 不会生成函数的同学，可以走传送门：GhostLX：算法学习笔记(5)：生成函数

易得答案序列的生成函数为$F(x)=(1+x+x^2+\cdots+x^t)^k$

由引文中的式1：\sum_{i= 0}^{n}x^

$$1+x+x^2+\cdots+x^t=\cfrac{1-x^{t+1}}{1-x}$$

所以有

$$\begin{eqnarray}F(x)&=&(\cfrac{1-x^{t+1}}{1-x})^k\\&=&(1-x^{t+1})^k\cfrac{1}{(1-x)^k}\\\end{eqnarray}$$

二项式定理有：

$$(1-x^{t+1})^k=\sum_{i=0}^k \binom k i (-1)^ix^{(t+1)i}$$

广义二项式定理有：

$$\begin{eqnarray}\cfrac{1}{(1-x)^k}&=&(1-x)^{-k}\\&=&\sum_{j=0}^{+\infty}\cfrac{-k(-k-1)(-k-2)\cdots(-k-j+1)}{j!}(-x)^j\\&=&\sum_{j=0}^{+\infty}\cfrac{k(k+1)(k+2)\cdots(k+j-1)}{j!}x^j\\&=&\sum_{j=0}^{+\infty}\binom {k+j-1}{j}x^j\\\end{eqnarray}$$

考虑两个式子的卷积，枚举第一个式子的求和下标i，其对应幂次方为$(t+1)i$，那么另一边的幂次方就应该为$n-(t+1)i$，有因为其原式子求和下标的下界为0，故需要满足$n-(t+1)i>0$，即$i\le \lfloor \cfrac{n}{t+1}\rfloor$。有因为第一个式子本身求和终点的限制，所以有$i\le \min(k,\lfloor \cfrac{n}{t+1}\rfloor)$

所以最后的答案为：

$$[x^n]F(x)=\sum_{i=0}^{\min(k,\lfloor \frac{n}{t+1}\rfloor)} (-1)^i\binom k i \binom {k+n-(t+1)i-1}{n-(t+1)i}$$

模型2C

> n个**无标号**的球放到k个**有标号**的盒子中，盒子**允许为空。**每个盒子里最多有1个小球，且非空的盒子不能相邻。

容易得到$2\times n-1\le k$时才有解（考虑极端情况，一隔一），其余情况方案数量均为0

1. 先取出n个盒子排成一行，每个箱子放入一个球。剩余k-n个盒子。
2. 然后在每两个盒子之间插入一个空盒子，还剩下k-2n+1个盒子。
3. 接着我们将剩余的k-2n+1个盒子插入到“带球盒子”之间，共n-1个间隔，加上最前面和最后面一共n+1个位置可以插入盒子。
   1. 此处为什么不能带上**步骤2**中的所有盒子，变成2n-2个间隔？因为那些盒子的作用**仅仅是为了防止非空盒子相邻**，而两个空盒子相连的情况，会在一个空盒子插入到2的某个空盒子前面、后面，被重复计算了两次，所以是行不通的。
4. 我们将**步骤3**中梳理一下，无标号的盒子k-2n+1个，有标号的间隔（位置）为n+1个，允许间隔（位置为空）。然后我们将“盒子”看成“球”，将“间隔（位置）”看成“盒子”这不就是隔板法（模型2）吗？故方案数为$\binom {(k-2n+1)+(n+1)-1}{(n+1)-1}=\binom {k-n+1}{n}$

故最后的答案为$\binom{k-n+1}{n}$。

例题

例题1：2022CCPC区域赛广州站L

> 给定n个人，m个不同的车站，求出把人安排到车站中**且同一车站内的人排列**的方案数，强制每一个车站都要有人。
> 数据范围：$1\le m\le n\le 10^5$，数据组数$1\le T\le 100$

转化为我们的模型：n个**有标号**的小球，m个**有标号**的盒子，盒子**不允许为空**，且**盒子内部考虑排序。**

如果将本题的**有标号**，改变为**无标号**怎么做？我们考虑模型3：

- n个**无标号**的球放到k个**有标号**的盒

- 因为所有小球都相同，内部排序也只有1方式，所以方案数还是\binom {n-1}{m-1}

但是本题是**不同**的小球，因为n个小球被m-1块木板分为m份之后，我们对于每种具体方案，我们都给小球标上n!种排列，也就是每个方案变成了n!种方案。

故最终的答案为n!\times \binom {n-1}{m-1}

```
#include<bits/stdc++.h>
using namespace std;
const int N = 1e5 + 10, md = 998244353;
typedef long long LL;
#define rep(i, a, b) for(int i = (a); i <= (b); i ++)
#define dwn(i, a, b) for(int i = (a); i >= (b); i --)

LL fac[N], ifac[N];

LL fpow(LL a, LL b)
{
    LL res = 1;
    while(b)
    {
        if(b & 1)
            res = res * a % md;
        a = a * a % md;
        b >>= 1;
    }
    return res;
}
LL n, m;
void init()
{
    int n = 1e5 + 5;
    fac[0] = 1;
    rep(i, 1, n)
        fac[i] = fac[i - 1] * i % md;
    ifac[n] = fpow(fac[n], md - 2);
    dwn(i, n - 1, 1)
        ifac[i] = ifac[i + 1] * (i + 1) % md;
    ifac[0] = 1;
}


int main()
{
    int T;
    cin >> T;
    init();
    while(T --)
    {
        cin >> n >> m;
        LL ans = 1;

       ans = fac[n] * fac[n - 1] % md;
       ans = ans * ifac[m - 1] % md * ifac[n - m] % md;

        // n ! * C_{n - 1}^{m - 1}
        cout << ans << '\n';
    }
}
```

例题2：JMU第九届校程序设计竞赛

> 厦门的孙厝步行街是个繁华的小吃街道，街上开着n家小吃店，编号依次为1到n。因为孙厝步行街距离JMU很近，贝贝每天放学都会日常去逛街。今天贝贝身上带了k元，贝贝发现了一种幸福感最满的购物方式：

> 前期限制花费：让自己在编号为1\sim m的小吃店限制花费，第iii个小吃店最多花费w_i元。
> 后期不限制花费：在编号比m大的小吃店，花费的金额不做限制。
> 贝贝想知道自己有多少种花费方式，但是他是个菜狗，所以他没有办法解决这个问题，于是他就找到一个大佬（也就是你）帮忙。
> **数据范围**
> 1\le m\le 300,1\le n,k\le 5\times 10^6,0\le w_i\le 300

考虑动态规划，f[i][j]表示前i个商店花费j元的方案数

可以得到状态转移方程：f[i][j]=\sum_{k=j-w_i}^{j} f[i-1][k]，我们发现同一维的状态有很多重复的求和内容，所以我们考虑前缀和优化（预处理然后作差），可以为优化为m\times \sum_{i=0}^m w_i

在前面m个店铺花费了i元，后面的n-m个店铺，我们剩余k-i元可以随意分配，相当于**模型2**，隔板法可得：\binom{n-m+(k-i)-1}{n-m-1}

```
#include <bits/stdc++.h>
using namespace std;
const long long MOD = 1e9 + 7;
long long n, m, k, w[305];
long long ans, f[305][90005], Sum, sum[305];
long long fac[10000005], inv[10000005];
long long power(long long a, long long b, long long p)
{
    long long result = 1;
    while (b)
    {
        if (b & 1)
        {
            result = (1LL * result * a) % p;
        }
        a = (1LL * a * a) % p;
        b >>= 1;
    }
    return result;
}
long long C(long long n, long long m)
{
    return 1LL * fac[n] * inv[m] % MOD * inv[n - m] % MOD;
}
int main()
{
    scanf("%lld%lld%lld", &n, &m, &k);
    fac[0] = f[0][0] = 1LL;
    for (long long i = 1; i <= n + k; i++)
    {
        fac[i] = 1LL * fac[i - 1] * i % MOD;
    }
    inv[n + k] = power(fac[n + k], MOD - 2, MOD);
    for (long long i = n + k; i >= 1; i--)
    {
        inv[i - 1] = 1LL * i * inv[i] % MOD;
    }
    for (long long i = 1; i <= m; i++)
    {
        scanf("%lld", &w[i]);
        Sum += w[i];
    }
    for (long long i = 1; i <= m; i++)
    {
        sum[0] = 0;
        for (long long j = 0; j <= Sum; j++)
        {
            sum[j + 1] = (sum[j] + f[i - 1][j]) % MOD;
        }
        for (long long j = 0; j
```

```
            {
                f[i][j] = (sum[j + 1] - sum[max(0LL, j - w[i])] + MOD) % MOD;
            }
        }
        if (n == m)
        {
            printf("%lld\n", f[n][k]);
            return 0;
        }
        for (long long i = 0; i <= Sum; i++)
        {
            ans = (ans + (f[m][i] * C(n - m + k - i - 1, n - m - 1)) % MOD) % MOD;
        }
        printf("%lld\n", ans);
        return 0;
    }
```

参考

1. oi-wiki：组合数学-排列组合
2. oi-wiki：组合数学-贝尔数
3. 爱寂寞的时光：组合数学——排列组合经典模型
4. 知乎匿名用户：取球问题的回答

发布于 2023-02-01 00:42 · IP 属地福建 ， 编辑于 2024-02-09 07:46 · IP 属地福建

组合数学（Combinatorics） OI（信息学奥林匹克） ACM 竞赛