20. 组合计数

20.1 Catalan数

20.1.1 满足条件的01序列

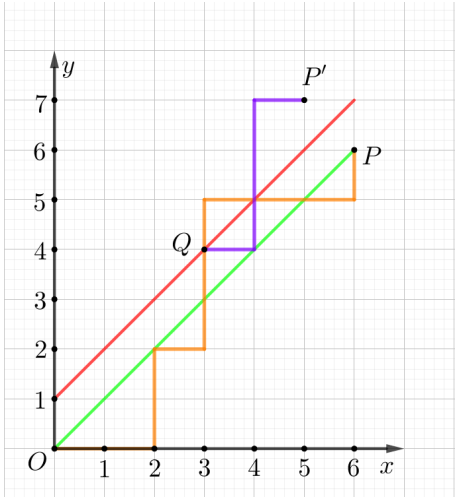
题意

给定n个0和n个1, 将它们按某种顺序排成长度为$2n$的序列, 输出它们能排成的序列中满足任意前缀序列中0的个数不少于1的个数的序列的个数, 答案对$(1e9+7)$取模.

思路

考虑$n=6$的情况. 考察平面上从点$(0,0)$走到点$(6,6)$的走法, 将每个01序列映射到一条路径, 其中0表示右移一个单位, 1表示上移一个单位. 显然这是双射, 故合法序列数即合法走法数.

因序列中任意前缀序列中0的个数不少于1的个数, 即$x \le y$, 亦即路径在$y=x$或下方. 故最终答案为从点$(0,0)$走到点$(6,6)$且路径不经过$y=x+1$的走法数, 可用从点$(0,0)$走到点$(6,6)$的总路径数C_{12}^6减去经过$y=x+1$的路径数.



如上图, 对任一经过$y=x+1$的路径(橙色), 找到该路径中第一个与$y=x+1$(红色)相交的点Q, 将该点后的路径沿$y=x+1$对称至紫色路径. 显然对每条经过$y=x+1$的路径作对称后的路径的终点都是点$P(6,6)$关于$y=x+1$的对称点$P'(5,7)$, 且对称后的路径与原穿过$y=x+1$到点$(6,6)$的路径一一对应. 故最终答案为$C_{12}^6-C_{12}^5$.

对任意n, 合法序列数为$C_{2n}^n-C_{2n}^{n-1}$, 它称为n的Catalan数, 记作$C_n=\dfrac{C_{2n}^n}{n+1}$.

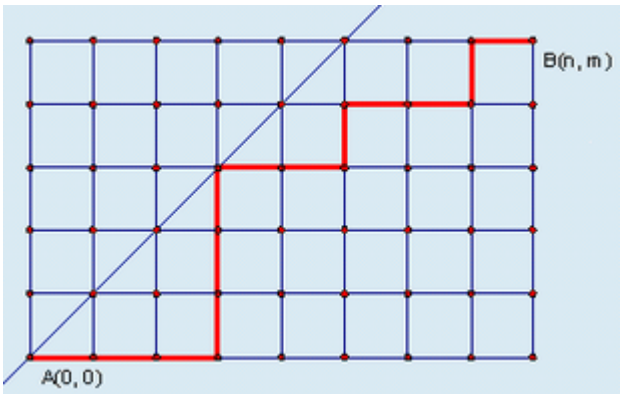
代码

```cpp
const int MOD = 1e9 + 7;

void solve() {
    int n; cin >> n;

    int ans = 1;
    int a = n * 2, b = n;
    for (int i = a; i > a - b; i--)
        ans = (ll)ans * i % MOD;
    for (int i = 1; i <= b; i++)
        ans = (ll)ans * qpow(i, MOD - 2, MOD) % MOD;
    ans = (ll)ans * qpow(n + 1, MOD - 2, MOD) % MOD;
```

```
    cout << ans << endl;
}

int main() {
    solve();
}
```

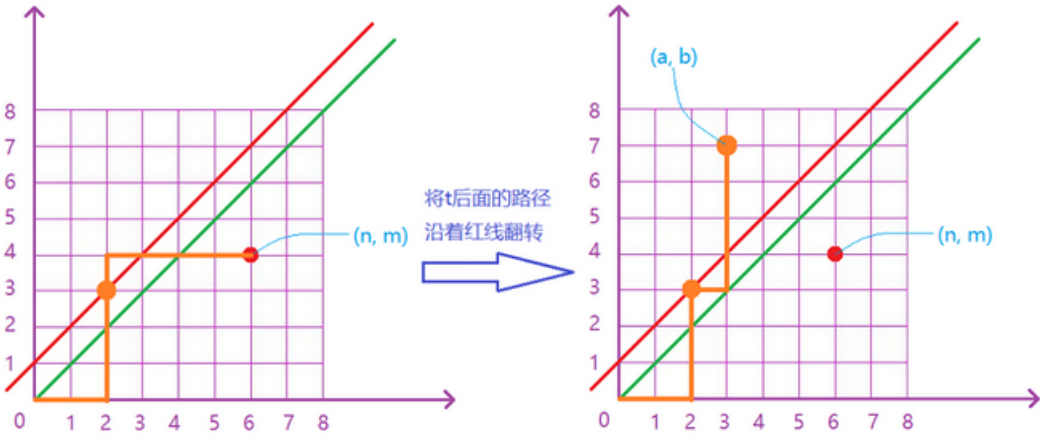
20.1.2 网格

题意



如上图,某城市的街道呈网格状,左下角坐标$A(0,0)$,右上角坐标$B(n,m)$ $(1 \le m \le n \le 5000)$.现从A走到B,每次只能沿街道向右或向上走,且不能经过途中直线左上方的点,即经过的点(x,y)满足$x \ge y$,求走法数.

思路



设点(n,m)关于$y=x+1$的对称点为$(m-1,n+1)$,则任一条非法路径与一条从$(0,0)$到(a,b)的路径一一对应,故$ans = C_{m+n}^{n} - C_{m+n}^{m-1}$.

将$n!$分解素因数的时间复杂度为$O(n \log \log n)$.

代码

```
const int MAXN = 1e5 + 5;
int n, m;
int primes[MAXN], cnt;
bool vis[MAXN];
int a[MAXN], lena, b[MAXN], lenb;  // 组合数的结果及其长度

```

```
void init(int n) {  // 线性筛
  for (int i = 2; i <= n; i++) {
    if (!vis[i]) primes[cnt++] = i;

    for (int j = 0; primes[j] * i <= n; j++) {
      vis[primes[j] * i] = true;
      if (i % primes[j] == 0) break;
    }
  }
}

int get(int n, int p) {  // 求n!中素因子p的个数
  int res = 0;
  while (n) res += n / p, n /= p;
  return res;
}

void mul(int res[], int& len, int x) {  // res*=x
  int tmp = 0;
  for (int i = 0; i < len; i++) {
    tmp += res[i] * x;
    res[i] = tmp % 10;
    tmp /= 10;
  }
  while (tmp) {
    res[len++] = tmp % 10;
    tmp /= 10;
  }
}

int C(int a, int b, int res[MAXN]) {  // C(a,b),结果存放在res中,返回res的长度
  int len = 1;
  res[0] = 1;

  for (int i = 0; i < cnt; i++) {
    int p = primes[i];
    int s = get(a, p) - get(b, p) - get(a - b, p);  // 分子分母的素因子数之差
    while (s--) mul(res, len, p);
  }

  return len;
}

void sub(int a[], int lena, int b[], int lenb) {  // a-=b
  for (int i = 0, tmp = 0; i < lena; i++) {
    a[i] -= tmp + b[i];
    if (a[i] < 0) a[i] += 10, tmp = 1;
    else tmp = 0;
  }
}

int main() {
  init(MAXN - 1);

  cin >> n >> m;

  lena = C(n + m, n, a), lenb = C(n + m, m - 1, b);
  sub(a, lena, b, lenb);
```

```
65
66    int idx = lena - 1;
67    while (!a[idx] && idx) idx--;
68    while (~idx) cout << a[idx--];
69  }
```

20.1.3 有趣的数列

题意

称一个长度为$2n$的数列是有趣的,当且仅当它满足如下三个条件:①它是$1 \sim 2n$的一个排列$\{a_i\}$;②所有奇数项满足$a_1 < a_3 < \cdots < a_{2n-1}$,所有偶数项满足$a_2 < a_4 < \cdots < a_{2n}$;③任意相邻的两项$a_{2i-1}$与$a_{2i}$ $(1 \le i \le n)$满足$a_{2i-1} < a_{2i}$.

给定整数n, p $(1 \le n \le 1e6, 2 \le p \le 1e9)$,求有多少个长度为$2n$的不同的有趣的数列,答案对$p$取模.

思路

答案可能是Catalan数的问题:

①递推式满足$f(n) = f(1) \cdot f(n-1) + f(2) \cdot f(n-2) + \cdots$,如二叉树的个数问题.

②满足任意前缀中某种东西的个数≥另一种东西的个数.

因奇数项和偶数项分别递增,只需确定$1 \sim 2n$中的每个数是奇数项还是偶数项即可.显然1为奇数项,2可为奇数项也可为偶数项.

一个数可以是偶数项的充要条件是:它前面的奇数项的个数≥偶数项的个数.

[证] 若不然,以考察5能否为偶此项为例,此时序列的前几项为$1, 3, 2, 4, ?, 5$,此时$?$只能是≥ 6的数,与该数列是有趣的矛盾.

一个长度为$2n$的数列有趣的充要条件是:对任意前缀,其中的奇数项的个数≥偶数项的个数,且奇数项和偶数项各n个.构造序列$a[]$,若将数i放在奇数项,则$a[i] = 0$;否则$a[i] = 1$,则$a[]$满足任意前缀的0的个数≥ 1的个数,这样的$a[]$的个数是Catalan数,即$C_{2n}^{n} - C_{2n}^{n-1}$.

代码

```
1   const int MAXN = 2e6 + 5;
2   int n, p;
3   int primes[MAXN], cnt;
4   bool vis[MAXN];
5
6   void init(int n) {  // 线性筛
7     for (int i = 2; i <= n; i++) {
8       if (!vis[i]) primes[cnt++] = i;
9
10      for (int j = 0; primes[j] * i <= n; j++) {
11        vis[primes[j] * i] = true;
12        if (i % primes[j] == 0) break;
13      }
14    }
```

```
}

int get(int n, int p) {  // 求n!中素因子p的个数
  int res = 0;
  while (n) res += n / p, n /= p;
  return res;
}

int C(int a, int b) {
  int res = 1;
  for (int i = 0; i < cnt; i++) {
    int prime = primes[i];
    int s = get(a, prime) - get(b, prime) - get(a - b, prime);
    res = (ll)res * qpow(prime, s, p) % p;
  }
  return res;
}

int main() {
  cin >> n >> p;

  init(n << 1);

  int ans = ((ll)C(n * 2, n) - C(n * 2, n - 1) + p) % p;
  cout << ans;
}
```

20.1.4 Bracket Sequence

题意 (0.5 s)

给定整数n, k $(1 \leq k, n \leq 1e5)$,求长度为$2n$的、包含k种括号的合法括号序列数,答案对$1e9+7$取模.

思路

长度为$2n$的、包含1种括号的合法括号序列数即Catalan数H_n.

有k种括号时,只需考虑左括号,共n个,每个左括号有k种选择,另外n个右括号只能取与其对应的左括号的相同类型,方案数为k^n.

综上,$ans = H_n \cdot k^n$.

代码

```
const int MAXN = 2e5 + 5;
const int MOD = 1e9 + 7;
int fac[MAXN], ifac[MAXN];

void init() {  // 预处理阶乘及其逆元
  fac[0] = 1;
  for (int i = 1; i < MAXN; i++) fac[i] = (ll)fac[i - 1] * i % MOD;

  ifac[MAXN - 1] = qpow(fac[MAXN - 1], MOD - 2, MOD);
  for (int i = MAXN - 1; i; i--) ifac[i - 1] = (ll)ifac[i] * i % MOD;
}

```

```
int C(int n, int m) {  // 组合数C(n,m)
  return (ll)fac[n] * ifac[m] % MOD * ifac[n - m] % MOD;
}

void solve() {
  init();

  int n, k; cin >> n >> k;

  int ans = (ll)C(2 * n, n) * qpow(n + 1, MOD - 2, MOD) % MOD * qpow(k, n, MOD) % MOD;
  cout << ans;
}

int main() {
  solve();
}
```

20.2 组合计数

20.2.1 越狱

题意

监狱有连续编号为$1 \sim n$的n $(1 \leq n \leq 1e12)$个房间,每个房间关押一个犯人.有m $(1 \leq m \leq 1e8)$种宗教,每个犯人信仰其中一种,若相邻房间的犯人信仰的宗教相同,则可能发生越狱.给定n和m,问有几种状态可能发生越狱,答案对100003取模.

思路

直接求可能发生越狱的状态数困难,考虑用总状态数m^n减去不发生越狱的状态数.1号犯人可选m种宗教,2号犯人可选$(m-1)$种宗教,3号犯人可选$(m-1)$种宗教,依此类推,最终答案为$m^n - m \cdot (m-1)^{n-1}$.

代码

```
const int MOD = 1e5 + 3;

ll qpow(int a, ll k) {
  int res = 1;
  while (k) {
    if (k & 1) res = (ll)res * a % MOD;
    a = (ll)a * a % MOD;
    k >>= 1;
  }
  return res;
}

int main() {
  ll m, n; cin >> m >> n;
  cout << (qpow(m, n) - m * qpow(m - 1, n - 1) % MOD + MOD) % MOD;
}
```

20.2.2 牡牛和牝牛

题意

有n $(1 \leq n \leq 1e5)$只牛,其中有A种牛和B种牛.A牛好斗,任意两头A牛间至少相隔k $(1 \leq k < n)$头B牛.求排队方法数,答案对5000011取模.

思路

将A种牛视为1,B种牛视为0,转化为任意两个1间至少间隔k个0的$0-1$串的个数.

$dp[i]$表示长度为i且以1结尾的$0-1$串的个数.按上一个1在下标$0,1,\cdots,n-k-1$分类,显然这样分类是不重不漏的,故$dp[i]=\sum_{j=0}^{n-k-1} dp[j]$.初始化$dp[0]=1$,此时$dp[1]=1$,符合其代表的意义.最终$ans=\sum_{i=0}^{n} dp[i]$.

注意到状态$O(n)$,转移$O(n)$,总时间复杂度$O(n^2)$.可用前缀和优化转移,总时间复杂度$O(n)$.

代码

```cpp
const int MAXN = 1e5 + 5;
const int MOD = 5000011;
int n, k;  // n个数,任意开不过
int dp[MAXN];  // dp[i]表示长度为i且以1结尾的0-1串的个数
int pre[MAXN];  // dp[]的前缀和

int main() {
  cin >> n >> k;

  dp[0] = pre[0] = 1;  // 初始化
  for (int i = 1; i <= n; i++) {
    dp[i] = pre[max(i - k - 1, 0)];
    pre[i] = ((ll)pre[i - 1] + dp[i]) % MOD;
  }

  cout << pre[n];
}
```

20.2.3 方程的解

题意

给定整数k,x $(1 \leq k \leq 100, 1 \leq x < 2^{31})$,求不定方程$a_1+a_2+\cdots+a_{k-1}+a_k=g(x)$的正整数解的组数,其中$g(x)=x^x \bmod 1000$,且$k \leq g(x)$.

思路

$g(x)$可用快速幂求,时间复杂度$O(\log x)$.

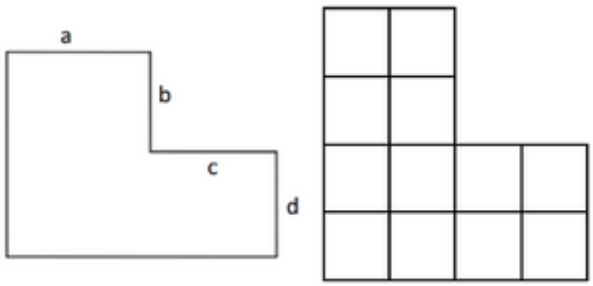
不定方程$a_1+a_2+\cdots+a_k=n$的正整数解数为C_{n-1}^{k-1},最大要计算C_{1000}^{100},可用$C_n^m=C_{n-1}^{m-1}+C_{n-1}^m$递推,时间复杂度$O(nm)$.用计算器求得$C_{1000}^{100}$有139位,可直接用高精加求.

代码

```cpp
const int MAXN = 150;
int n, k, x;  // n=g(x)
int C[1000][100][MAXN];

void add(int c[], int a[], int b[]) {  // 高精加
  for (int i = 0, tmp = 0; i < MAXN; i++) {
    tmp += a[i] + b[i];
    c[i] = tmp % 10;
    tmp /= 10;
  }
}

void init() {  // 求组合数
  for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i && j < k; j++) {
      if (!j) C[i][j][0] = 1;
      else add(C[i][j], C[i - 1][j], C[i - 1][j - 1]);
    }
  }
}

int main() {
  cin >> k >> x;
  n = qpow(x % 1000, x, 1000);

  init();

  int* ans = C[n - 1][k - 1];
  int idx = MAXN - 1;
  while (!ans[idx]) idx--;
  while (~idx) cout << ans[idx--];
}
```

20.2.4 车的放置

题意



有如上左图所示的网格棋盘,其中整数a,b,c,d表示边的长度,即包含的格子数.如$a=b=c=d=2$时对应上右图.

在该棋盘上放k $(0\leq a,b,c,d,k\leq 1000)$个车,使得任意两车不在同一行和同一列,求方案数,答案对100003取模.

思路

100003是素数.

考察规则矩形的方案数.在$n \times m$的棋盘中放k个车,先选k行,有C_n^k种选择.每一行放完后可选的列数-1,有A_m^k种选择.故$n \times m$的棋盘中放k辆车的方案数为$C_n^k \cdot A_m^k$.

将原棋盘分割为$b \times a$和$d \times c$两矩形,设上矩形放了i个车,则下矩形放了$(k-i)$个车.枚举上矩形放了车的列,这样考察下矩形能放的列无需分类讨论.最终$ans = \sum_{i=1}^{k} C_k^i \cdot A_a^i \cdot C_d^{k-i} \cdot A_{a+c-i}^{k-i}$.

代码

```
const int MAXN = 2005;  // a+c最大2000
const int MOD = 100003;
int a, b, c, d, k;
int fac[MAXN], ifac[MAXN];

void init() {
  fac[0] = ifac[0] = 1;
  for (int i = 1; i < MAXN; i++) {
    fac[i] = (ll)fac[i - 1] * i % MOD;
    ifac[i] = (ll)ifac[i - 1] * qpow(i, MOD - 2, MOD) % MOD;
  }
}

int C(int a, int b) {
  if (a < b) return 0;
  return (ll)fac[a] * ifac[a - b] % MOD * ifac[b] % MOD;
}

int P(int a, int b) {
  if (a < b) return 0;
  return (ll)fac[a] * ifac[a - b] % MOD;
}

int main() {
  init();

  cin >> a >> b >> c >> d >> k;

  int ans = 0;
  for (int i = 0; i <= k; i++)
    ans = ((ll)ans + (ll)C(b, i) * P(a, i) % MOD * C(d, k - i) % MOD * P(a + c - i, k - i)) % MOD;
  cout << ans;
}
```

20.2.5 数三角形

题意

给定一个$n \times m$ $(1 \le m, n \le 1000)$的网格,求三个顶点都在格点上的(非退化)三角形的个数.

思路

点$(0,0)$与点(n,m)所连线段上的整点数为$\gcd(n,m)+1$.

[**证**] 点$(0,0)$与点(n,m)所连线段经过的点都在直线$y=\frac{m}{n}\cdot x$上.若y是整数,则$\frac{mx}{n}$是整数,即$n \mid mx$.

x只能取$\frac{n}{\gcd(n,m)}, 2\frac{n}{\gcd(n,m)}, 3\frac{n}{\gcd(n,m)}, \cdots, n$共$\gcd(n,m)$个数,再加上$(0,0)$共$\gcd(n,m)+1$个数.

[**推论**] 线段两端点的横、纵坐标之差分别为i和j的线段上的整点数为$\gcd(i,j)+1$.

$n++, m++$,转化为格点的行数和列数.

考虑总方案数C_{nm}^3减去非法方案数,非法方案中三角形的三个顶点共线.

以左下方的格点为原点建系,非法方案分为如下三类:①直线斜率为0,有nC_m^3种情况;②直线斜率为$+\infty$,有mC_n^3种情况;③斜率为正和斜率为负的情况一一对应(如关于一条斜率为$+\infty$的直线对称),只需考虑斜率为正的情况.在所选取的三个点中,以最靠左下方的点为$(1,1),(1,2),\cdots,(n,m)$分为nm类.设最靠坐下方的点为(i,j),则选取的第二个点在以(i,j)为左下角、以(n,m)为右上角的矩形区域中,其中有$(n-i)(m-j)$个格点,进而第三个点的选法数是第一、二个点所连线段上的整点数.若第一、二个点的横坐标、纵坐标之差为a和b,则第三个点的选法数为$\gcd(a,b)-1$.故斜率为正和斜率为负的方案数都为$\sum\limits_{\substack{1\le i\le n\\1\le j\le m}}[\gcd(i,j)-1](n-i)(m-j)$.

代码

```cpp
int n, m;

ll C(int n) { return (ll)n * (n - 1) * (n - 2) / 6; }  // C(n,3)

int main() {
  cin >> n >> m;
  n++, m++;  // 转化为表示格点数

  ll ans = C(n * m) - (ll)n * C(m) - (ll)m * C(n);
  for (int i = 1; i <= n; i++) {  // i,j分别为第一、二个点的横坐标、纵坐标之差
    for (int j = 1; j <= m; j++)
      ans -= (ll)2 * (gcd(i, j) - 1) * (n - i) * (m - j);
  }
  cout << ans;
}
```

20.2.6 序列统计

题意

有t $(1 \le t \le 100)$组测试数据.每组测试数据给定三个整数n, l, r $(0 \le n, l, r \le 1e9, l \le r)$,求长度在$[1, n]$范围内,元素大小在$[l, r]$范围内的单调不降序列的数量,答案对$1e6 + 3$取模.

思路

$1e6 + 3$是素数.

$x_1 + x_2 + \cdots + x_k \le n$的非负整数解数为$C_{n+k}^{k}$.

[证] 令$y_i = x_1 + 1$,隔板法,注意最后一个小球之后无需放隔板,并忽略最后一个隔板之后的所有小球,故为C_{n+k}^{k}.

满足$0 \le a_1 \le a_2 \le \cdots \le a_k \le n$的序列$a[]$的个数为$C_{n+k}^{k}$.

[证1] 考察其差分序列$b[]$,转化为求$b_1 + b_2 + \cdots + b_k \le n$的非负整数解数,即$C_{n+k}^{k}$.

[证2] 若为严格上升序列$0 \le b_1 < b_2 < \cdots < b_k < n$,则序列$b[]$的个数为$C_{n+1}^{k}$.

考虑如何将非严格上升序列映射为严格上升序列.

令$c_1 = a_1, c_2 = a_2 + 1, c_3 = a_3 + 2, \cdots, c_k = a_k + k - 1$,此时$0 \le c_1 < c_2 < \cdots \le c_k < n + k - 1$,

则序列$c[]$的个数为C_{n+k}^{k}.

枚举序列长度$k \in [1, n]$,考察有多少个长度为k的、元素大小在$[l, r]$范围内的单调不降序列,元素大小可映射为$[0, r - l]$.考察满足$0 \le a_1 \le a_2 \le \cdots \le a_k \le r - l$的序列$a[]$的个数,由上述讨论知有$C_{r-l+k}^{k}$个,则$ans = \sum_{k=1}^{n} C_{r-l+k}^{k}$.

令$m = r - l$,则$ans = \sum_{k=1}^{n} C_{m+k}^{k} = C_{m+1}^{1} + C_{m+2}^{2} + \cdots + C_{m+n}^{n}$.

注意到$C_a^b = C_{a-1}^{b} + C_{a-1}^{b-1}$,则$ans = C_{m+1}^{m+1} + C_{m+1}^{1} + C_{m+2}^{2} + \cdots + C_{m+n}^{n} - C_{m+1}^{m+1} = C_{r-l+n+1}^{r-l+1} + 1$.

代码

```cpp
const int MOD = 1e6 + 3;
int n, l, r;

int C(int a, int b, int p) {
  if (a < b) return 0;

  int down = 1, up = 1;
  for (int i = a, j = 1; j <= b; i--, j++)
    up = (ll)up * i % p, down = (ll)down * j % p;
  return (ll)up * qpow(down, p - 2, p) % p;
}

int Lucas(int a, int b, int p) {
  if (a < p && b < p) return C(a, b, p);
  return (ll)Lucas(a / p, b / p, p) * C(a % p, b % p, p) % p;
}

```

```
int main() {
  CaseT{
    cin >> n >> l >> r;
    int ans = (Lucas(r - l + n + 1, r - l + 1, MOD) + MOD - 1) % MOD;
    cout << ans << endl;
  }
}
```

20.2.7 Space Station

原题指路:https://codeforces.com/gym/103466/problem/I

题意 (3 s)

有$(n+1)$ $(1 \le n \le 1e5)$个站,编号$0 \sim n$,i号站有能量a_i $(0 \le a_i \le 50)$,每访问一个站都可获得该站的所有能量,每次只能访问能量不超过当前自身的能量的站,每个站只能访问一次.初始时在0号站,获得其能量a_0.求访问完所有站的方案数,答案对$1e9+7$取模.

思路

注意到$0 \le a_i \le 50$,则当前能量≥ 50后剩下的站可以任意顺序访问.注意到a_i可能为0,而能量为0的站可在任意位置访问,设共有z个能量为0的站.先求出访问$(n-z)$个能量非零的站的方案数x,从n个位置选出$(n-z)$个位置放能量非零的站,剩下的z个位置放z个能量为0的站的任一排列(注意能量为0的站编号不同计为不同方案),则
$ans = x \cdot C_n^{n-z} \cdot z! = x \cdot \dfrac{n!}{(n-z)!(n-n+z)!} \cdot z! = x \cdot \dfrac{n!}{(n-z)!}$,再对$1e9+7$取模即可.

预处理出$1e5$内的数的阶乘模$1e9+7$和阶乘模$1e9+7$下的逆元,两者都可用递推求得,前者显然,时间复杂度$O(n)$;考虑后者:不妨设$n!$、$(n+1)!$的逆元分别为x, y,则$n! \cdot x \equiv (n+1)! \cdot y \equiv 1 \pmod{MOD}$.注意到
$(n+1)! \cdot y = n!(n+1) \cdot y = n![y(n+1)] = n! \cdot y' \equiv 1 \pmod{MOD}$,则$x \equiv y' = y(n+1) \pmod{MOD}$
.因$MOD = 1e9+7$是素数,可先用快速幂求出$1e5!$模MOD下的逆元,再递推求得$1 \sim 1e5$的数的阶乘的逆元,时间复杂度$O(n + \log 1e5) \approx O(n)$.

考虑如何求访问$(n-z)$个能量非零的站的方案数.初始能量最低为1,最多要访问49个能量为1的站后能量≥ 50,剩下的站可以任意顺序访问,则搜索的范围是将50拆成若干个正整数之和的方案数(其中有一些是不合法的方案),用完全背包易求得至少拆成两个正整数之和的分拆数为204225,搜索范围不大,可用记搜.

每次只能访问能量不超过当前自身能量的站,则搜索顺序是从小到大枚举站的能量,若有相同能量$power$的站则先一起考虑后,再考虑能量为$(power+1)$的站,则可开一个$cnt[]$来记录能量为i的站的个数$cnt[i]$.

考虑记搜如何记录状态.易想到用二进制数或bool数组记录状态,但这记录的是每个站是否已被访问,与上述搜索顺序不符,考虑用哈希记录状态.每个能量非零的站的能量都是$[1, 50]$内的整数,则可将相同能量的站的个数对应到一个
P $(P > 51)$进制数的数位,用这个$1e5$位的P进制数模MOD来记录状态.为降低哈希冲突,可类似于字符串哈希,取经验值
$P = 131$或$P = 13331$.

代码

```
const int MAXN = 5e5 + 5, MOD = 1e9 + 7;
int fac[MAXN], invfac[MAXN];  // 阶乘、阶乘模MOD的逆元

void init() {  // 预处理出阶乘、阶乘的逆元
```

```cpp
  fac[0] = invfac[0] = 1;  // 0!=1
  for (int i = 1; i < MAXN; i++) fac[i] = (ll)fac[i - 1] * i % MOD;  // 阶乘

  invfac[MAXN - 1] = qpow(fac[MAXN - 1], MOD - 2);  // Fermat小定理求逆元
  for (int i = MAXN - 2; i; i--) invfac[i] = (ll)invfac[i + 1] * (i + 1) % MOD;
}

int cnt[55];  // cnt[i]表示能量为i的站的个数
unordered_map<ull, int> state;  // state[i]=j表示状态为i的方案数模MOD为j
const int P = 131;  // P进制数

int dfs(int cur, int rest) {  // 当前能量、还剩几个能量非零的站未访问
  if (rest == 0) return 1;  // 搜完了
  if (cur >= 50) return fac[rest];  // 当前能量≥50,剩下的站可以任意顺序访问,即全排列

  ull ha = 0;  // 哈希值
  for (int i = 50; i > 0; i--) ha = ha * P + cnt[i];  // 枚举所有能量的站的方案数,记录状态

  if (state.count(ha)) return state[ha];  // 搜过了

  int res = 0;  // 方案数
  for (int i = 1; i <= cur; i++) {  // 枚举能量不超过当前能量的站
    if (!cnt[i]) continue;  // 没有该能量的站未访问

    int tmp = cnt[i];  // 备份
    cnt[i]--;  // 已访问一个该能量的站
    res = (res + (ll)tmp * dfs(cur + i, rest - 1) % MOD) % MOD;  // 任选一个相同能量的站,然后搜下一个
    cnt[i]++;  // 恢复
  }
  state[ha] = res;  // 记录该状态的答案
  return res;
}

int main() {
  init();

  int n, now; cin >> n >> now;  // now为当前能量
  int zero = 0;  // 能量为0的站的个数
  for (int i = 1; i <= n; i++) {
    int power; cin >> power;
    if (!power) zero++;
    else cnt[power]++;  // 更新对应能量的站的个数
  }

  int ans = dfs(now, n - zero);  // 当前能量为now,还有(n-zero)个能量非零的站没有访问
  ans = (ll)ans * fac[n] % MOD * invfac[n - zero] % MOD;
  cout << ans;
}
```

20.2.8 Chessboard

题意

有T $(1 \le T \le 1e5)$组测试数据.给n $(1 \le n \le 1e6)$行m $(1 \le m \le 1e6)$列的网格染色,要求染色路径连续且能遍历所有网格,且不经过已染色的网格,求染色方案数模$1e9+7$.

思路

注意到最多$1e5$组测试数据,而n和m最大$1e6$,用搜索、DP等都不能在1 s内通过,考虑直接推公式.

注意到染色只能将一块矩形不断地扩大,即给一个矩形增加一行或增加一列,否则最后会遇到贪吃蛇咬到自己的尾巴的情况,不符合题意.关键:染完一个矩形的最后一个点必在四个角上.

考虑将1×1的矩形不断染色扩充为$n \times m$的矩形,则需扩展$(n-1)$行、$(m-1)$列,共需扩展$n-1+m-1=n+m-2$次,里面选$(n-1)$次扩展行(类似于网格图中从一个给定点走到另一给定点的方案数),最后染色的点是矩形的四个角之一,故最终答案$4C_{n+m-2}^{n-1}$.

注意特判$n=1$或$m=1$的情况,即只有一行或一列,只有两种可能,即从两端之一开始染色.

代码

```cpp
const int MAXN = 2e6 + 5, MOD = 1e9 + 7;
int fac[MAXN], invfac[MAXN];  // 阶乘、阶乘模MOD的逆元

void init() {  // 预处理出阶乘、阶乘的逆元
  fac[0] = invfac[0] = 1;  // 0!=1
  for (int i = 1; i < MAXN; i++) fac[i] = (ll)fac[i - 1] * i % MOD;  // 阶乘

  invfac[MAXN - 1] = qpow(fac[MAXN - 1], MOD - 2);  // Fermat小定理求逆元
  for (int i = MAXN - 2; i; i--) invfac[i] = (ll)invfac[i + 1] * (i + 1) % MOD;
}

int C(int n, int m) {  // 组合数C(n,m)%MOD
  return (ll)fac[n] * invfac[m] % MOD * invfac[n - m] % MOD;
}

int main() {
  init();

  CaseT{
    int n,m; cin >> n >> m;
    int ans = (n == 1 || m == 1) ? 2 : (ll)4 * C(n + m - 2, n - 1) % MOD;
    cout << ans << endl;
  }
}
```

20.2.9 The Journey of Geor Autumn

题意

称一个$1 \sim n$的全排列是好的,如果对$\forall i \in (k,n]$,有$a_i > \min\limits_{j \in [i-k,i-1]} a_j$.给定整数$n,k$ $(1 \le n,k \le 1e7)$,求好的排列的个数,答案对998244353取模.

思路

显然$1\sim n$的最小数只能放在$a_1,\cdots,a_k$中,不妨设放在a_{pos},则$a_1,\cdots,a_{pos-1}$可随便放,其方案数是先从剩下的$(n-1)$个数中选$(pos-1)$个数的全排列的个数,即A_{n-1}^{pos-1}.设$a_{pos+1},\cdots,a_n$的方案数为$dp[n-pos]$,则$a_1,\cdots,a_n$的方案数为$dp[n]$.按a_n前面的k个数中的最小值在a_j分类,与$a_1,\cdots,a_{pos-1}$的讨论类似,易得$dp[n]=\sum\limits_{j=1}^{min\{n,k\}}A_{n-1}^{j-1}\cdot dp[n-j]$.状态$O(n)$,转移$O(\min\{n,k\})$,总时间复杂度$O(n\cdot\min\{n,k\})$,会TLE.

考虑优化,注意到$dp[i]=(i-1)!\sum\limits_{j=1}^{k}\frac{dp[i-j]}{(i-j)!}$,显然求和部分为前缀和.令$pre[i]=\sum\limits_{j=1}^{i}\frac{dp[i]}{i!}$,则$dp[i]=(i-1)!(pre[i-1]-pre[i-k-1])$,在转移时同时维护$pre[]$即可.总时间复杂度$O(n)$.

代码

```cpp
const int MAXN = 1e7 + 5;
const int MOD = 998244353;
int n, k;
int fac[MAXN], ifac[MAXN];
int dp[MAXN];  // dp[i]表示长度为i的排列中好的排列的个数
int pre[MAXN];  // pre[i]是dp[i]/i!的前缀和

void init() {  // 预处理阶乘及其逆元
  fac[0] = 1;
  for (int i = 1; i < MAXN; i++) fac[i] = (ll)fac[i - 1] * i % MOD;

  ifac[MAXN - 1] = qpow(fac[MAXN - 1], MOD - 2, MOD);
  for (int i = MAXN - 1; i; i--) ifac[i - 1] = (ll)ifac[i] * i % MOD;
}

int main() {
  init();

  cin >> n >> k;

  dp[0] = pre[0] = 1;  // 初始条件
  for (int i = 1; i <= n; i++) {
    dp[i] = (((ll)pre[i - 1] - (i - k - 1 >= 0 ? pre[i - k - 1] : 0)) * fac[i - 1] % MOD + MOD) % MOD;  // 注意结果可能为负数
    pre[i] = ((ll)pre[i - 1] + (ll)dp[i] * ifac[i] % MOD) % MOD;
  }
  cout << dp[n];
}
```

20.2.10 Matrix

题意

将$1\sim n^2$这n^2个数填入$n\times n$的矩阵,每个数只能用一次.对每个固定的方案,令a_i为第i $(1\le i\le n)$行的最小值,集合$S=\{a_1,\cdots,a_n\}\bigcap\{1,\cdots,n\}$.求所有填法的集合$S$大小之和,即$\sum|S|$,答案对$998244353$取模.

有t $(1\le t\le 30)$组测试数据.每组测试数据输入一个整数n $(1\le n\le 5000)$.

思路

显然只需考虑$1 \sim n$作为某行的最小值时对答案的贡献.以1作为某行的最小值为例,先将其放到某行(不妨设为第一行),有n种情况.在剩下的(n^2-1)个数中选$(n-1)$个>1的数放在第一行,有$C_{n^2-1}^{n-1}$种情况.第一行的数和其他行的数全排列,有$n!(n^2-n)!$种情况.显然$ans = n \cdot n! \cdot (n^2-n)! \cdot \sum_{i=1}^{n} C_{n^2-1}^{n-i}$.

代码 -> 2021CCPC东北赛-A(组合计数)

```cpp
const int MAXN = 5005 * 5005;
const int MOD = 998244353;
int fac[MAXN], ifac[MAXN];

void init() {  // 预处理阶乘及其逆元
  fac[0] = 1;
  for (int i = 1; i < MAXN; i++) fac[i] = (ll)fac[i - 1] * i % MOD;

  ifac[MAXN - 1] = qpow(fac[MAXN - 1], MOD - 2, MOD);
  for (int i = MAXN - 1; i; i--) ifac[i - 1] = (ll)ifac[i] * i % MOD;
}

int C(int n, int m) {  // C(n,m)
  if (n < m) return 0;
  else return (ll)fac[n] * ifac[m] % MOD * ifac[n - m] % MOD;
}

void solve() {
  init();

  CaseT{
    int n; cin >> n;

    int ans = 0;
    for (int i = 1; i <= n; i++) ans = ((ll)ans + C(n * n - i, n - 1)) % MOD;
    ans = (ll)ans * n % MOD * fac[n] % MOD * fac[n * n - n] % MOD;
    cout << ans << endl;
  }
}

int main() {
  solve();
}
```

20.2.11 Optimal Strategy

题意

有编号$1 \sim n$的n $(1 \le n \le 1e6)$个物品,其中第i $(1 \le i \le n)$个物品的价值为a_i $(1 \le a_i \le n)$.A和B两人轮流操作,A先手.当前轮到的人在剩余的物品中取走一个物品.要求当所有物品取完时,两人各自手中的物品的价值之和都最大.两人都采取最优策略,求有多少种游戏进行序列(每轮玩家取的物品的编号构成的序列),答案对998244353取模.

思路

常见误区:最优策略是每次都拿剩下的物品中价值最大的物品.对样例1,若应拿价值最大的物品,则先手不应拿1,与样例解释不符.事实上,要使得游戏结束时两人各自手中的物品价值之和都最大,只需保证两人都拿到取得自己物品价值之和的最大值应拿的那些物品即可,与什么时候拿到那个物品无关.

显然游戏的结果只有先手必胜和平局两种情况,但这题不是博弈论,而是纯纯的组合计数.

具体地考察最优策略中两人分别应拿到哪些数.设$cnt[i]$表示价值为i的数的个数.①$cnt[i]$为偶数时,显然A和B各拿一半;②$cnt[i]$为奇数时,A先手会多拿走一个,转化为B先手且$cnt[i]$为偶数的情况.综上,对$cnt[i]$个数i,只需考虑其中$\left\lfloor \frac{cnt[i]}{2} \right\rfloor$个的分配情况.

注意到$1 \le a_i \le 1e6$,不妨按顺序枚举每个数值的分配情况,而两人何时拿到自己应拿的物品不影响最终两人各自手中的物品的价值之和,故从小到大或从大到小枚举每个数值的分配情况都可以,下面以从小到大枚举为例.设当前准备分配数i,则前面的$1 \sim (i-1)$共$sum = \sum_{j=1}^{i-1} cnt[j]$个数都已分配好,只需在前$\left(sum + \left\lfloor \frac{cnt[i]}{2} \right\rfloor \right)$个数中选$\left\lfloor \frac{cnt[i]}{2} \right\rfloor$个给先手即可,方案数为$C_{sum+\lfloor cnt[i]/2 \rfloor}^{\lfloor cnt[i]/2 \rfloor}$.因以不同顺序拿到相同数值的数视为不同方案,故$ans = \sum_{i=1}^{n} cnt[i]! \cdot C_{sum+\lfloor cnt[i]/2 \rfloor}^{\lfloor cnt[i]/2 \rfloor}$.直接计算时间复杂度为$O(n^2)$,注意到$sum$是$cnt[j]$的前缀和,只需在转移的同时维护即可,时间复杂度$O(n)$.

代码

```cpp
const int MAXN = 1e6 + 5;
const int MOD = 998244353;
int n;
int cnt[MAXN];  // cnt[i]表示值为i的数的个数
int fac[MAXN], ifac[MAXN];

void init() {  // 预处理阶乘及其逆元
  fac[0] = 1;
  for (int i = 1; i < MAXN; i++) fac[i] = (ll)fac[i - 1] * i % MOD;

  ifac[MAXN - 1] = qpow(fac[MAXN - 1], MOD - 2, MOD);
  for (int i = MAXN - 1; i; i--) ifac[i - 1] = (ll)ifac[i] * i % MOD;
}

int C(int n, int m) {  // 组合数C(n,m)
  return (ll)fac[n] * ifac[m] % MOD * ifac[n - m] % MOD;
}

int main() {
  init();

  cin >> n;
  for (int i = 0; i < n; i++) {
    int a; cin >> a;
    cnt[a]++;
  }

  int ans = 1;  // 注意初始值为1而不是0
  int pre = 0;  // cnt[i]的前缀和
  for (int i = 1; i <= n; i++) {
    if (!cnt[i]) continue;

```

```
33      ans = (ll)ans * fac[cnt[i]] % MOD * C(pre + cnt[i] / 2, cnt[i] / 2) % MOD;
34      pre += cnt[i];
35    }
36    cout << ans;
37  }
```

20.2.12 Edge Groups

题意

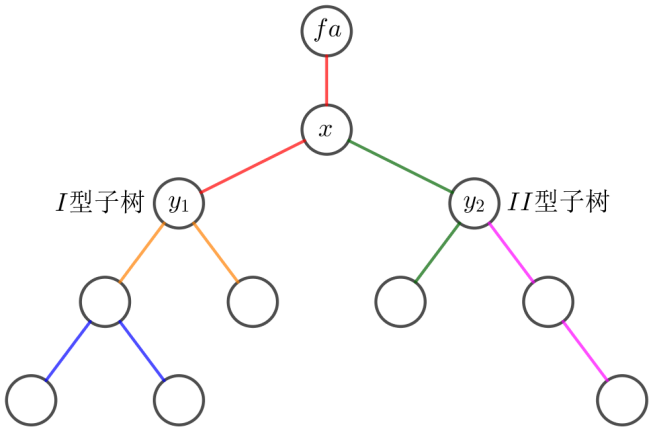
给定一个有n $(3 \le n \le 1e5, n为奇数)$个节点、$(n-1)$条边的无向连通图,求将边分为$\frac{n-1}{2}$组且满足如下两条件的方案数,答案对998244353取模:①每组有且仅有两条边;②两条在同一组中的边有一个公共节点.

图用$(n-1)$行输入描述,每行包含两个正整数u, v $(1 \le u < v \le n)$,表示节点u和v间连有无向边.

思路

注意到连通n个节点至少需要$(n-1)$条边,则本题中的无向连通图是树.

考察以x为根节点的子树的方案数,其中fa为x的父节点.如下图,按子树的节点数为奇数、偶数分为两种情况,分别称为I型、II型子树.



II型子树有偶数个节点,奇数条边,再加上边(x, y_2)即可两两配对;I型子树有奇数个节点,偶数条边,y_1的子树中的边两两配对后,边(x, y_1)未配对,可能需与边(fa, x)配对,是否需要用到边(fa, x)取决于x的子树中I型子树的数量的奇偶:①有偶数个时,所有I型子树的边(x, y_i)可两两配对;②有奇数个时,所有I型子树的边(x, y_i)两两配对剩下的一条边与(fa, x)配对.

考察n $(n为偶数)$条边平均分为$\frac{n}{2}$组的方案数$dfac[n]$.$n=2$时,$dfac[2]=1$;$n=4$时,$dfac[4]=\frac{C_4^2C_2^2}{A_2^2}=3$;$n=6$时,$dfac[6]=\frac{C_6^2C_4^2C_2^2}{A_3^3}=15$;$n=8$时,$dfac[8]=\frac{C_8^2C_6^2C_4^2C_2^2}{A_4^4}=105$.

注意到$dfac[4]=dfac[2]*3, dfac[6]=dfac[4]*5, dfac[8]=dfac[6]*7$,猜想$dfac[n]=dfac[n-2]*(n-1)$,下面证明该结论.

$dfac[n]=\frac{C_n^2C_{n-2}^2C_{n-4}^2\cdots C_2^2}{A_{\frac{n}{2}}^{\frac{n}{2}}}, dfac[n-2]=\frac{C_{n-2}^2C_{n-4}^2\cdots C_2^2}{A_{\frac{n-2}{2}}^{\frac{n-2}{2}}}$,则

$$\frac{dfac[n]}{dfac[n-2]}=\frac{C_n^2}{\frac{n}{2}}=\frac{n!}{2!\cdot(n-2)!}\cdot\frac{2}{n}=n-1.$$

$dp[x]$表示以节点x为根节点的子树分组的方案数,设其子树的根节点分别为y_i,其中有cnt个I型子树,则状态转移方程

$$dp[x]=\begin{cases} dfac[cnt]*\prod_i dp[y_i], cnt\text{为偶数} \\ dfac[cnt+1]*\prod_i dp[y_i], cnt\text{为奇数} \end{cases}.$$

代码

```cpp
const int MAXN = 1e5 + 5, MOD = 998244353;
int n;  // 节点数
vi graph[MAXN];  // g[i]=j表示节点i和j间有有向边
int siz[MAXN];  // siz[i]表示以节点i为根节点的子树的大小(节点数)
int dfac[MAXN];  // dfac[n]=1*3*5*...*(n-1),n为偶数
int dp[MAXN];  // dp[i]表示以节点i为根节点的子树分组的方案数

void init() {  // 预处理出dfac[]
  dfac[0] = 1;
  for (int i = 2; i <= n; i += 2) dfac[i] = (ll)dfac[i - 2] * (i - 1) % MOD;
}

void dfs(int u, int fa) {  // 当前节点、前驱
  siz[u] = 1;  // 当前子树的大小初始值为1,即子树的根节点
  dp[u] = 1;  // 方案数初始化为1
  int cnt = 0;  // 统计以u为根节点的子树中,节点数为奇数的子树的数量
  for (auto& v : graph[u]) {  // 注意取引用,因要更新子树的信息
    if (v == fa) continue;  // 搜过了

    dfs(v, u);  // 搜以v为根节点的子树,v的前驱为u

    // 用子树更新根节点u的信息
    siz[u] += siz[v];
    dp[u] = (ll)dp[u] * dp[v] % MOD;

    if (siz[v] & 1) cnt++;  // 更新节点数为奇数的子树的数量
  }

  if (cnt & 1) cnt++;  // 有奇数个节点数为奇数的子树,则需把边(u,fa)一起考虑
  dp[u] = (ll)dp[u] * dfac[cnt] % MOD;
}

int main() {
  cin >> n;
  for (int i = 1; i < n; i++) {
    int u, v; cin >> u >> v;
    graph[u].push_back(v), graph[v].push_back(u);
  }

  init();  // 预处理出dfac[]

  dfs(1, 0);  // 1号节点作为根节点,无前驱,记为0
  cout << dp[1];
}
```

20.2.13 cocktail with hearthstone

题意

某游戏的规则:①每个玩家的战绩用数对(a,b)表示,其中a表示胜利场数,b表示失败场数.初始时战绩为$(0,0)$;②每轮胜利的玩家$a++$,失败的玩家$b++$,没有平局;③每轮对战的两个人的战绩相同,设为(a,b),则该轮结束后会产生一个$(a+1,b)$和一个$(a,b+1)$;④若有玩家胜利n次或失败m次,他将自动退出游戏.

现安排了2^{n+m}个玩家,保证所有玩家在退出游戏前都能找到对手.现有q个询问,每个询问给定a,b,设战绩(a,b)符合退出游戏的要求,问有多少个人以战绩(a,b)退出游戏,答案对$1e9+7$取模.

第一行输入三个整数n,m,q $(1\le m<n\le 2e5,1\le q\le 2e5)$.接下来$q$行每行输入两个整数$a,b$ $(0\le a\le n,0\le b\le m)$,数据保证$a=n$或$b=m$.

对每个询问,输出以战绩(a,b)退出游戏的人数,答案对$1e9+7$取模.

思路

游戏过程:$(0,0)\begin{cases}(1,0)\begin{cases}(2,0)\begin{cases}(3,0)\\(2,1)\end{cases}\\(1,1)\begin{cases}(2,1)\\(1,2)\end{cases}\end{cases}\\(0,1)\begin{cases}(1,1)\begin{cases}(2,1)\\(1,2)\end{cases}\\(0,2)\begin{cases}(1,2)\\(0,3)\end{cases}\end{cases}\end{cases}$.

第0层:$(0,0)$有1个.

第1层:$(1,0)$有1个、$(0,1)$有1个.

第2层:$(2,0)$有1个、$(1,1)$有2个、$(0,2)$有1个.

第3层:$(3,0)$有1个、$(2,1)$有3个、$(1,2)$有3个、$(0,3)$有1个.

显然个数的规律为杨辉三角.

考察$a=n$的情况.显然(a,b)在第$(a+b)$层,且在该层数量占比$\frac{C_{a+b}^{a}}{2^{a+b}}$.但注意到$a=n$时,到达$(a,b)$前$(a,b-1),(a,b-2),\cdots$早已退出游戏,故能转移到$(a,b)$的合法状态只有$(a-1,b)$,故占比应取$\frac{C_{a+b-1}^{a-1}}{2^{a+b}}$.总人数$2^{n+m}$,故$ans=2^{n+m-a-b}\cdot C_{a+b-1}^{a-1}$.同理$b=m$时,$ans=2^{n+m-a-b}\cdot C_{a+b-1}^{b-1}$.

注意特判$(a,b)=(n,m)$的情况,因无任何合法状态能转移到(n,m),故$ans=0$.

代码

```cpp
const int MAXN = 4e5 + 5;
const int MOD = 1e9 + 7;
int fac[MAXN], ifac[MAXN];

void init() {  // 预处理阶乘及其逆元
  fac[0] = 1;
  for (int i = 1; i < MAXN; i++) fac[i] = (ll)fac[i - 1] * i % MOD;

  ifac[MAXN - 1] = qpow(fac[MAXN - 1], MOD - 2, MOD);
```

```
  for (int i = MAXN - 1; i; i--) ifac[i - 1] = (ll)ifac[i] * i % MOD;
}

int C(int n, int m) {  // 组合数C(n,m)
  return (ll)fac[n] * ifac[m] % MOD* ifac[n - m] % MOD;
}

void solve() {
  init();

  int n, m; cin >> n >> m;
  CaseT{
    int a,b; cin >> a >> b;

    if (a == n && b == m) {
      cout << 0 << endl;
      continue;
    }

    int ans = 0;
    if (a == n) ans = qpow(2, n + m - a - b, MOD) * C(a + b - 1, a - 1) % MOD;
    else ans = qpow(2, n + m - a - b, MOD) * C(a + b - 1, b - 1) % MOD;
    cout << ans << endl;
  }
}

int main() {
  solve();
}
```

20.2.14 Zztrans 的班级合照

题意 (3 s)

n(偶数)个人按如下要求排队:排成人数相同的两排,每排从左往右身高不减,且第二排同学身高不低于第一排对应位置的同学的身高.现给定将同学按身高升序排列后每个同学的排名(身高相同的同学排名相同),求排队方案数,答案对998244353取模.

第一行输入一个偶数n $(2 \le n \le 5000)$.第二行输入n个整数$a_1, \cdots, a_n$ $(1 \le a_i \le n)$,分别表示每个同学的身高排名.

思路

记录每个身高i的人数$cnt[i]$后将原数组去重,则相同的身高的人任意排,答案乘上人数的阶乘即可.

考虑所有身高都不同时的情况.注意到任意时刻第二排的人数不超过第一排的人数,$dp[i][j]$表示排完前i个人,且第一排比第二排多j个人的方案数,则最终答案为$dp[n][n/2]$.

用sum记录当前排完的人数.对每个身高的人数i,枚举第一排比第二排多的人数j,显然它不超过$\min\left\{\frac{n}{2}, sum\right\}$.因第二排的人数不超过第一排的人数,故还需满足$j \ge sum - j$.按上一个状态$dp[sum - i][]$中第一排比第二排多的人数分类,不妨设$dp[sum][]$中第一排还需补$k$个人,则可从$dp[sum - i][j - k]$转移到$dp[sum][j]$,枚举$k \in [0, \min\{i, j\}]$即可.

代码

```cpp
const int MAXN = 5005;
const int MOD = 998244353;
int n;
int cnt[MAXN];  // cnt[i]表示身高为i的人数
int dp[MAXN][MAXN];  // dp[i][j]表示排完前i个人,且第一排比第二排多j个人的方案数
int fac[MAXN], ifac[MAXN];

void init() {  // 预处理阶乘
  fac[0] = 1;
  for (int i = 1; i < MAXN; i++) fac[i] = (ll)fac[i - 1] * i % MOD;
}

void solve() {
  init();

  cin >> n;
  for (int i = 0; i < n; i++) {
    int x; cin >> x;
    cnt[x]++;
  }

  vi h;  // 去重后的身高
  int ans = 1;
  for (int i = 1; i <= n; i++) {
    if (cnt[i]) {
      h.push_back(cnt[i]);
      ans = (ll)ans * fac[cnt[i]] % MOD;  // 相同身高的人任意排
    }
  }

  int sum = 0;  // 当前排完的人数
  dp[0][0] = 1;  // i=0时只有j=0是合法方案
  for (auto i : h) {  // 枚举每个身高的人数
    sum += i;

    // 第一排的人数比第二排多的人数不超过min{当前排完的人数,总人数的一半}
    for (int j = min(n / 2, sum); j >= sum - j; j--) {
      for (int k = 0; k <= min(i, j); k++)  // 枚举上一个状态站第一排的人数
        dp[sum][j] = ((ll)dp[sum][j] + dp[sum - i][j - k]) % MOD;
    }
  }

  ans = (ll)ans * dp[n][n / 2] % MOD;
  cout << ans;
}

int main() {
  solve();
}
```

20.2.15 有趣的数

题意

称一个数是有趣的,如果:①只包含数码$0,1,2,3$,且四个数码都至少出现一次;②所有0出现在所有1之前,所有2出现在所有3之前;③最高位非0.求恰有n $(4 \le n \le 1000)$位的有趣的数的个数,答案对$1e9+7$取模.

思路

设$0,1$的总个数为k $(2 \le k \le n-2)$,则$2,3$的总个数为$(n-k)$.因首位不能放0,则在后$(n-1)$位中选k位放$0,1$,有C_{n-1}^{k}种方案.在k位$0,1$中,因所有0出现在所有1之前,设0的个数为t $(1 \le t \le k-1)$,则只能前t位是0,后$(k-t)$位是1,有$(k-1)$种方案,同理$2,3$有$(n-k-1)$种方案,故$ans=\sum_{k=2}^{n-2} C_{n-1}^{k}(k-1)(n-k-1)$.

代码

```cpp
const int MAXN = 1005;
const int MOD = 1e9 + 7;
int n;
int C[MAXN][MAXN];  // 组合数

void init() {
  for (int i = 0; i < MAXN; i++) {
    for (int j = 0; j <= i; j++) {
      if (!j) C[i][j] = 1;
      else C[i][j] = (C[i - 1][j] + C[i - 1][j - 1]) % MOD;
    }
  }
}

void solve() {
  init();

  cin >> n;
  int ans = 0;
  for (int k = 2; k <= n - 2; k++)
    ans = ((ll)ans + (ll)C[n - 1][k] * (k - 1) % MOD * (n - k - 1) % MOD) % MOD;
  cout << ans;
}

int main() {
  solve();
}
```

20.2.16 Shaass and Lights

原题指路:https://codeforces.com/problemset/problem/294/C

题意

n盏灯排成一行,从左到右依次编号$1 \sim n$,初始时有些灯亮有些灯灭.每一步可点亮个灭了的灯,如果它两边至少有一个亮的灯.问有多少种使得灯全亮的开灯方案,答案对$1e9+7$取模.

第一行输入两个整数n, m $(1 \le m \le n \le 1000)$,分别表示灯的个数和初始时亮的灯的个数.第二行输入n个范围为$[1, n]$的整数,表示初始时亮的灯的编号.

思路

m个亮的灯将序列分为$(m+1)$段,从左往右编号$1 \sim (m+1)$.每次操作只能点亮每一段内的最左边或最右边的灯,特别地,第一段只能点亮最右边的灯,最后一段只能点亮最左边的灯.

设第i $(1 \le i \le m+1)$段的长度为len_i,先只考虑每次点亮的是哪一段内的灯,有$\frac{(n-m)!}{\prod_{i=1}^{m+1} len_i!}$种情况.

考虑每次点亮每一段内的最左边还是最右边的灯,第i段中前(l_i-1)个灯都有两种选择,最后一个灯只有一种选择,故$ans = \frac{(n-m)!}{\prod_{i=1}^{m+1} len_i!} \cdot \prod_{i=1}^{m+1} 2^{len_i-1}$.

代码

```cpp
const int MAXN = 1005;
const int MOD = 1e9 + 7;
int n, m;
int pos[MAXN];
int fac[MAXN], ifac[MAXN];
int pow2[MAXN];

void init() {
  fac[0] = 1;
  for (int i = 1; i < MAXN; i++) fac[i] = (ll)fac[i - 1] * i % MOD;

  ifac[MAXN - 1] = qpow(fac[MAXN - 1], MOD - 2, MOD);
  for (int i = MAXN - 1; i; i--) ifac[i - 1] = (ll)ifac[i] * i % MOD;

  pow2[0] = 1;
  for (int i = 1; i < MAXN; i++) pow2[i] = (ll)pow2[i - 1] * 2 % MOD;
}

void solve() {
  init();

  cin >> n >> m;
  for (int i = 1; i <= m; i++) cin >> pos[i];

  sort(pos + 1, pos + m + 1);
  pos[m + 1] = n + 1;  // 哨兵,减少特判边界

  int ans = fac[n - m];
  for (int i = 0; i <= m; i++) {
    int len = pos[i + 1] - pos[i] - 1;
    if (len <= 0) continue;

```

```
    if (i > 0 && i < m)  // 注意pos[i]和pos[i+1]都在范围内时才需要乘2的幂次
      ans = (ll)ans * pow2[len - 1] % MOD;
    ans = (ll)ans * ifac[len] % MOD;
  }
  cout << ans;
}

int main() {
  solve();
}
```

20.2.17 Emordnilap

原题指路:https://codeforces.com/contest/1777/problem/B

题意

考察一个$1\sim n$的排列$p=[p_1,\cdots,p_n]$,将其翻转后接到原排列之后得到序列$a=[p_1,\cdots,p_n,p_n,\cdots,p_1]$.定义一个排列$p[]$的权值为其对应的序列$a[]$中的逆序对数,求所有$1\sim n$的排列的权值之和,答案对$1e9+7$取模.

思路I

[**定理**] 任意排列的权值相同.

[**证**] 考察任一排列$p[]$中的下标$i,j\ \ (1\le i<j\le n)$,则p_i,p_j在$a[]$中的出现顺序为$[p_i,\cdots,p_j,\cdots,p'_j,\cdot,p'_i]$.

考察p_i,p_j对逆序对数的贡献,有如下两种情况:

①$p_i>p_j$时,p_i与p_j,p'_j构成两对逆序对.

②$p_i<p_j$时,p'_i与p_j,p'_j构成两对逆序对.

综上,任意一对元素对排列的权值的贡献为2,进而任一排列的权值$w=C_n^2\cdot 2=n(n-1)$.

故$ans=n!\cdot n(n-1)$.

代码

```
const int MOD = 1e9 + 7;

void solve() {
  Z n; cin >> n;

  Z ans = n * (n - 1);
  for (int i = 2; i <= n.val(); i++) ans *= i;
  cout << ans << endl;
}

int main() {
  CaseT
    solve();
}
```

思路II

思路I中的定理的另一证法:

按逆序对$(p_i,p_j)\ \ (1\le i<j\le 2n)$在$a[]$中的位置分类,有如下三种情况:

①$i,j\le n$,即p_i,p_j都在$a[]$的左半部分.

②$i,j\ge n+1$,即p_i,p_j都在$a[]$的右半部分.

③$i\le n,j\ge n+1$,即p_i在$a[]$的前半部分,p_j在$a[]$的后半部分.

考察上述三种情况对答案的贡献:

(1)情况①和情况②对答案的贡献之和为C_n^2.

[**证**] 若(p_i,p_j)在$a[]$的左半部分为顺序,则(p'_j,p'_i)在$a[]$的右半部分为逆序,V.V.

故$p[]$中的每对元素要么在$a[]$的左半部分贡献一个逆序对,要么在$a[]$的右半部分贡献一个逆序对.

(2)情况③对答案的贡献为$\dfrac{n(n-1)}{2}$.

[**证**] 考察$a[]$的左半部分中$p[]$中的每个元素贡献的逆序对.

(i)$p_i=1$不贡献逆序对.

(ii)$p_i=2$与$a[]$的右半部分的$p'_j=1$贡献1对逆序对.

(iii)$p_i=3$与$a[]$的右半部分的$p'_j=1,2$贡献2对逆序对.

$\vdots$

故逆序对数$0+1+\cdots+(n-1)=\dfrac{n(n-1)}{2}$.

20.2.18 Graph Coloring (easy version)

原题指路:https://codeforces.com/contest/1792/problem/F1

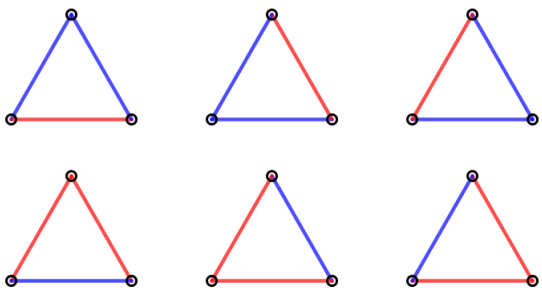
题意 (2 s)

对一张图中的节点,称一个点集S是红连通的,如果对$\forall(v_1,v_2)\in S$,都$\exists$一条从节点v_1到节点v_2且只经过红色的边的路径.类似地可定义蓝连通的点集.

给定一张有标号的$n\ \ (3\le n\le 5000)$阶无向完全图.现要给每条边染红色或蓝色,使得图中至少存在一条红色边,至少存在一条蓝色边,且任一至少包含两个节点的点集,要么是红连通的,要么是蓝连通的,但不能同时是红连通和蓝连通的.求染色方案数,答案对998244353取模.

样例解释

$n=3$的6种合法的染色方案分别为:



思路I

[**引理**] 对一个无向图G,若它不连通,则G关于完全图的补图G'连通.类似地,若G'不连通,则G连通.

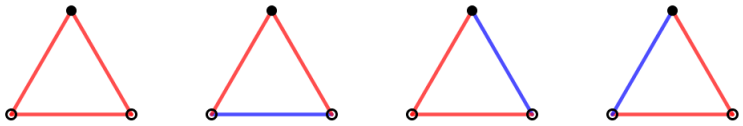
[**证**] 设G不连通,则它至少包含两个连通分量.在两个不同的连通分量中分别取两个节点u和v,则它们之间无边.

G中所有与u不在同一连通块中的节点,在G'中都与u连通;G中所有与v不在同一连通块中的节点,在G'中都与v连通.

注意到G'包含边(u,v),故G'连通.

先对完全图中的一些边染红色,设红色的边和与其相关的节点构成的图为G,则G关于完全图的补图G'的边都染蓝色.由**引理**:若G不连通,则G'连通.这样即可保证任一至少包含两个节点的点集,至少是红连通的和蓝连通的之一.下面只需保证没有点集既是红连通的,又是蓝连通的.

A_n表示对n阶完全图的所有边染红色或蓝色,使得1号节点在一个红连通块中,且任一至少包含两个节点的点集是红连通的或蓝连通的之一的染色方案数(此处不要求图中至少有一条红色边和一条蓝色边).对3阶完全图,$A_3=4$种合法的染色方案如下,其中黑色实心节点即1号节点:



设A_n统计的所有合法染色方案构成的集合为A_n,则A_n中的图要么是红不连通的,要么是蓝不连通的.由对称性:只需统计A_n中蓝不连通的图,即红连通的图的数量.设A_n中红连通的图的数量为B_n,则$A_n=\begin{cases}B_n,n=1\\2\cdot B_n,n\geq 2\end{cases}$,其中特判$n=1$是因为由一个节点构成的点集既是红连通的,又是蓝连通的.

按与1号节点在同一红连通块中的节点数$k\in[1,n-1]$分类(k可取1是因为由1号节点自身构成的点集是红连通的),则还需在剩下的$(n-1)$个节点中选$(k-1)$个节点,这k个节点需构成一个红连通图,则方案数为$C_{n-1}^{k-1}\cdot B_k$.剩下的$(n-k)$个节点及相关的边构成一个合法染色方案即可,则方案数为A_{n-k}.故$B_n=\sum\limits_{k=1}^{n-1}C_{n-1}^{k-1}\cdot B_k\cdot A_{n-k}$,其中$A_n=\begin{cases}B_n,n=1\\2\cdot B_n,n\geq 2\end{cases}$.初始条件$A_1=1$,即1阶完全图只能不染任何边;$A_2=1$,即2阶完全图只能将唯一的边染为红色.

用记搜求A_n后,最终答案$ans=2\cdot(A_n-1)$,其中-1是因为A_n的统计包含边全染红色的情况.

代码I

```cpp
const int MOD = 998244353;
const int MAXN = 5005;
Z fac[MAXN], ifac[MAXN];
int dp[MAXN];  // dp[n]表示对n阶完全图的合法染色方案数

void init() {  // 预处理阶乘及其逆元
  fac[0] = 1;
  for (int i = 1; i < MAXN; i++) fac[i] = fac[i - 1] * i;

  ifac[MAXN - 1] = qpow(fac[MAXN - 1], MOD - 2);
  for (int i = MAXN - 1; i; i--) ifac[i - 1] = ifac[i] * i;

  memset(dp, -1, sizeof(dp));
}

Z C(int n, int m) {  // 组合数C(n,m)
  if (n <= m) return n == m;
  else return fac[n] * ifac[m] * ifac[n - m];
```

```
}

Z cal(int n) {  // 求dp[n]
  if (~dp[n]) return dp[n];
  if (n <= 2) return dp[n] = 1;

  Z res = 0;
  for (int i = 1; i <= n - 1; i++) {
    res += cal(i) * cal(n - i) * C(n - 1, i - 1) * 2
      * (i == n - 1 ? (MOD + 1) / 2 : 1);  // (MOD+1)/2是2模MOD的逆元
  }
  return dp[n] = res.val();
}

void solve() {
  init();

  int n; cin >> n;
  cout << (cal(n) - 1) * 2 << endl;
}

int main() {
  solve();
}
```

思路II

将**思路I**的状态转移方程转化为迭代的形式.$dp[n]$表示对n阶完全图的所有边染红色或蓝色,使得1号节点在一个红连通块中,且任一至少包含两个节点的点集是红连通的或蓝连通的之一的染色方案数(此处不要求图中至少有一条红色边和一条蓝色边).初始条件同**思路I**,最终答案$ans = 2 \cdot (dp[n] - 1)$.

状态转移方程$dp[i] = \sum_{j=1}^{i-1} \delta(j) \cdot dp[j] \cdot dp[i-j] \cdot C_{i-1}^{j-1}$,其中$\delta(j) = \begin{cases} 2, j < i-1 \\ 1, j = i-1 \end{cases}$.初始条件$dp[1] = dp[2] = 1$.

代码II

```
const int MOD = 998244353;
const int MAXN = 5005;
Z fac[MAXN], ifac[MAXN];

void init() {  // 预处理阶乘及其逆元
  fac[0] = 1;
  for (int i = 1; i < MAXN; i++) fac[i] = fac[i - 1] * i;

  ifac[MAXN - 1] = qpow(fac[MAXN - 1], MOD - 2);
  for (int i = MAXN - 1; i; i--) ifac[i - 1] = ifac[i] * i;
}

Z C(int n, int m) {  // 组合数C(n,m)
  if (n <= m) return n == m;
  else return fac[n] * ifac[m] * ifac[n - m];
}

void solve() {
```

```
  init();

  int n; cin >> n;

  vector<Z> dp(n + 1);  // dp[i]表示
  dp[1] = dp[2] = 1;  // 初始条件
  for (int i = 3; i <= n; i++) {
    for (int j = 1; j <= i - 1; j++)
      dp[i] += (Z(1) + (j != i - 1)) * dp[j] * dp[i - j] * C(i - 1, j - 1);
  }
  cout << (dp[n] - 1) * 2 << endl;
}

int main() {
  solve();
}
```

20.2.19 Gerald and Giant Chess

原题指路:https://codeforces.com/problemset/problem/559/C

题意 (2 s)

一个$h \times w$ $(1 \le h, w \le 1e5)$的网格上有n $(1 \le n \le 2000)$个黑色格子.一个棋子从左上角出发,每次可向右或向下移动一格,但不能移动到黑色格子.求它从左上角移动到右下角的方案数,答案对$(1e9 + 7)$取模.

思路

若无黑色格子,则方案数为$C_{(h-1)+(w-1)}^{h-1}$.因黑色格子数量较少,考虑容斥,即总方案数减去经过每个黑色格子的方案数.

将终点(h, w)视为第$(n + 1)$个黑色格子,将所有黑色格子按横坐标、纵坐标升序排列.$dp[i]$表示从点$(1, 1)$到第i个黑色格子$blocks[i]$的方案数.设第i个格子的坐标为(x_i, y_i),注意到非法路径是从上一个黑色格子到下一个黑色格子,则状态转移方程$dp[i] = C_{(x_i-1)+(y_i-1)}^{x_i-1} - \sum_{j=1}^{i-1} dp[j] \cdot C_{(x_i-x_j)+(y_i-y_j)}^{x_i-x_j}$ $(x_i \ge x_j, y_i \ge y_j)$,最终答案$ans = dp[n + 1]$.

注意用预处理阶乘及其逆元求组合数时,预处理的范围是网格边长的两倍.

代码

```
typedef pair<int, int> pii;
#define x first
#define y second

const int MAXN = (1e5 + 5) * 2;
Z fac[MAXN], ifac[MAXN];

void init() {  // 预处理阶乘及其逆元
  fac[0] = 1;
  for (int i = 1; i < MAXN; i++) fac[i] = fac[i - 1] * i;

  ifac[MAXN - 1] = qpow(fac[MAXN - 1], MOD - 2);
  for (int i = MAXN - 1; i; i--) ifac[i - 1] = ifac[i] * i;
}

Z C(int n, int m) {  // 组合数C(n,m)
```

```
  if (n <= m) return n == m;
  else return fac[n] * ifac[m] * ifac[n - m];
}

void solve() {
  init();

  int h, w, n; cin >> h >> w >> n;
  vector<pii> blacks(n + 1);
  for (int i = 0; i < n; i++) cin >> blacks[i].x >> blacks[i].y;
  blacks[n] = { h, w };  // 将终点作为最后一个黑色格子
  sort(all(blacks), [&](const pii& a, const pii& b) {
    return a.x != b.x ? a.x < b.x : a.y < b.y;
  });

  vector<Z> dp(n + 1);  // dp[i]表示从左上角到第i个黑色格子的方案数
  for (int i = 0; i <= n; i++) {
    dp[i] = C(blacks[i].x + blacks[i].y - 2, blacks[i].x - 1);  // 总方案数

    for (int j = 0; j < i; j++) {
      if (blacks[i].x >= blacks[j].x && blacks[i].y >= blacks[j].y) {
        dp[i] -= dp[j]
          * C(blacks[i].x - blacks[j].x + blacks[i].y - blacks[j].y, blacks[i].x -
blacks[j].x);
      }
    }
  }
  cout << dp[n] << endl;
}

int main() {
  solve();
}
```

20.2.20 Array

原题指路:https://codeforces.com/problemset/problem/57/C

题意 (2 s)

给定一个整数n $(1 \le n \le 1e5)$. 求满足如下条件的长度为n的序列$a_1, \cdots, a_n$的个数, 答案对$(1e9 + 7)$取模: ① $a_i \in [1, n]$ $(1 \le i \le n)$; ②$a[]$单调(非严格).

思路

注意到确定$a[]$中每个元素出现的次数即可确定$a[]$本身, 则一个序列$a[]$与一个序列$\{x_1, \cdots, x_n\}$一一对应, 其中x_i $(1 \le i \le n)$为值i的出现次数, 则$x_1 + \cdots + a_n = n$, 转化为求该不定方程的非负整数解数, 即$C_{n+n-1}^{n-1} = C_{2n-1}^{n-1}$.

注意到非严格递增和非严格递减对称, 故答案为非严格递增的序列的个数的两倍, 但所有元素都相等的序列会被重复计数, 故$ans = 2C_{2n-1}^{n-1} - n = 2\dfrac{(2n-1)!}{(n-1)!n!} - n = \dfrac{2n \cdot (2n-1)!}{n \cdot (n-1)!n!} - n = C_{2n}^{n} - n$.

代码

```cpp
const int MOD = 1e9 + 7;
const int MAXN = (1e5 + 5) * 2;
Z fac[MAXN], ifac[MAXN];

void init() {
  fac[0] = 1;
  for (int i = 1; i < MAXN; i++) fac[i] = fac[i - 1] * i;

  ifac[MAXN - 1] = fac[MAXN - 1].inv();
  for (int i = MAXN - 1; i; i--) ifac[i - 1] = ifac[i] * i;
}

Z C(int n, int m) {
  if (n <= m) return n == m;
  else return fac[n] * ifac[m] * ifac[n - m];
}

void solve() {
  init();

  int n; cin >> n;
  cout << C(2 * n, n) - Z(n) << endl;
}

int main() {
  solve();
}
```

20.2.21 Foe Pairs

原题指路: https://codeforces.com/problemset/problem/652/C

题意

给定一个$1 \sim n$ $(1 \leq n \leq 3e5)$个排列$p = [p_1, \cdots, p_n]$. 现有m $(1 \leq m \leq 3e5)$个限制(a_i, b_i) $(1 \leq a_i, b_i \leq n; a_i \neq b_i)$, 表示$a_i$与$b_i$不能出现在同个区间中. 求$p[]$有多少个子区间满足上述限制.

思路

设$p[]$中元素i在下标$pos[i]$处. 对每个限制, 不妨设$pos[a[i]] < pos[b[i]]$.

$maxl[r]$表示以下标r为区间右端点时合法的最靠左的区间左端点l的下标, 则$maxl[b[i]] = \max\limits_{1 \leq i \leq m} \{maxl[a[i]] + 1\}$.

设当前合法的最靠左的区间左左端点为l. 枚举区间右端点r, 更新$l = \max\{l, maxl[r]\}$, 累加答案即可.

时间复杂度$O(n + m)$.

代码

```cpp
void solve() {
    int n, m; cin >> n >> m;
    vector<int> pos(n + 1);
    for (int i = 1; i <= n; i++) {
        int p; cin >> p;
        pos[p] = i;
    }

    // maxl[r]表示以下标r为区间右端点时合法的最靠左的区间左端点l的下标
    vector<int> maxl(n + 1);
    while (m--) {
        int a, b; cin >> a >> b;

        a = pos[a], b = pos[b];
        if (a > b) swap(a, b);
        maxl[b] = max(maxl[b], a + 1);
    }

    ll ans = 0;
    int l = 1;  // 当前合法的最靠左的区间左端点l
    for (int r = 1; r <= n; r++) {
        l = max(l, maxl[r]);
        ans += r - l + 1;
    }
    cout << ans << endl;
}

int main() {
    solve();
}
```

20.2.22 Bracket Sequences Concatenation Problem

原题指路: https://codeforces.com/problemset/problem/990/C

题意 (2 s)

给定n $(1 \le n \le 3e5)$个非空的括号序列$s_1, \cdots, s_n$, 求有多少个数对(i, j) $(1 \le i, j \le n)$ $s.t.$ $(s_i + s_j)$是合法的括号序列. 若$(s_i + s_j)$和$(s_j + s_i)$都是合法的括号序列, 且$i \ne j$, 则数对(i, j)和数对(j, i)都计入答案. 若$(s_i + s_i)$是合法的括号序列, 则数对(i, i)计入答案.

思路I

对括号序列s, 定义$match(s)$为与其匹配的括号序列, 则$match(s)$是将s反转后将'('换为')', 将')'换为'('.

称一个括号序列是好的, 如果它所有前缀中左括号数不少于右括号数. 显然好的括号序列可作为合法括号序列的左半部分.

对括号序列s, 定义$difference(s)$如下:

①若s是好的, 则$difference(s)$定义为左括号数减右括号数.

②若s非好的, 则$difference(s) = -1$.

$cnt[diff]$表示difference值为$diff$的好的括号序列的个数.

对每个括号序列s_i, 与其匹配的括号序列为$match(s_i)$. 若$match(s_i)$是好的, 则$(match(s_i)+s_i)$是一个合法的括号序列. 故$ans=\sum_{s_i} cnt[difference(match(s_i))]\cdot[match(s_i)\in goods]$.

代码I

```cpp
const int MAXS = 3e5 + 5;

// 若左括号数不少于右括号数, 则返回左括号数 - 右括号数; 否则返回-1
int getDifference(string s) {
    int res = 0;
    for (auto& ch : s) {
        if (ch == '(') res++;
        else res--;

        if (res < 0) return -1;
    }
    return res;
}

string getMatch(string s) {  // 求与其匹配的括号序列
    string res = s;
    reverse(all(res));
    for (auto& ch : res) {
        if (ch == '(') ch = ')';
        else ch = '(';
    }
    return res;
}

void solve() {
    int n; cin >> n;
    vector<string> a(n);
    for (auto& ai : a) cin >> ai;

    // cnt[diff]表示左括号数 - 右括号数 = diff ≥ 0的括号序列数
    vector<int> cnt(MAXS);
    for (int i = 0; i < n; i++) {
        int diff = getDifference(a[i]);
        if (~diff) cnt[diff]++;
    }

    ll ans = 0;
    for (int i = 0; i < n; i++) {
        string tmp = getMatch(a[i]);
        int diff = getDifference(tmp);
        if (~diff) ans += cnt[diff];
    }
    cout << ans << endl;
}

int main() {
    solve();
}
```

思路II

$cntLeft[diff]$表示可作为合法括号序列的左半部分的、difference值为$diff$的好的括号序列的个数, $cntRight[diff]$表示可作为合法括号序列的右半部分的、difference值为$diff$的好的括号序列的个数.

同**思路I**, 但将getMatch()改为考察每个括号序列能否作为合法括号序列的左半部分或右半部分. 具体地, 从前往后扫求$cntLeft[]$、从后往前扫求$cntRight[]$, 则$ans = \sum_{i=0}^{MAXS} cntLeft[i] \cdot cntRight[i]$, 其中$MAXS$为括号序列所能达到的difference值的最大值, 即字符串长度.

代码II

```cpp
const int MAXS = 3e5 + 5;

void solve() {
    // cntLeft[diff]表示可作为合法括号序列的左半部分的、difference值为diff的好的括号序列的个数
    // cntRight[diff]表示可作为合法括号序列的右半部分的、difference值为diff的好的括号序列的个数
    vector<int> cntLeft(MAXS), cntRight(MAXS);

    int n; cin >> n;
    for (int i = 0; i < n; i++) {
        string s; cin >> s;

        int len = s.length();
        int diff = 0;  // 左括号数 - 右括号数

        // 从前往后扫, 求cntLeft[]
        for (int j = 0; j < len; j++) {
            if (s[j] == '(') diff++;
            else if (diff) diff--;
            else {
                diff = -1;
                break;
            }
        }
        if (~diff) cntLeft[diff]++;

        diff = 0;  // 注意清空

        // 从后往前扫, 求cntRight[]
        for (int j = len - 1; j >= 0; j--) {
            if (s[j] == ')') diff++;
            else if (diff) diff--;
            else {
                diff = -1;
                break;
            }
        }
        if (~diff) cntRight[diff]++;
    }

    ll ans = 0;
    for (int i = 0; i < MAXS; i++)
        ans += (ll)cntLeft[i] * cntRight[i];
    cout << ans << endl;
}
```

```
45
46 int main() {
47     solve();
48 }
```

20.2.23 括号序列

原题指路: https://www.lanqiao.cn/problems/1456/learning/

题意

给定一个长度不超过5000的括号序列, 现要在任意位置添加若干个括号使其成为最短的合法括号序列, 求方案数, 答案对$(1e9+7)$取模.

思路

[**引理**] 合法括号序列的充要条件是: ①左括号数 $=$ 右括号数; ②任意前缀中左括号数 $\geq$ 右括号数.

显然不会同时添加一对匹配的括号, 则添加左括号和添加右括号独立. 从前往后扫一遍, 用$diff$记录左括号数 $-$ 右括号数. 若某时刻$diff<0$, 则需在此处添加左括号以满足**引理**中的性质②. 扫完整个序列后, $diff$值为未匹配的左括号数, 则还需添加$diff$个右括号以满足**引理**中的性质①. 上述过程中添加的左括号数和右括号数之和即最少需添加的括号数.

[**定理**] 添加左括号数的方案数与添加右括号数的方案数独立.

[**证**]

(1)若左括号与右括号添加在不同位置, 则显然独立.

(2)若左括号与右括号添加在同一位置, 如在同一位置添加2个左括号和2个右括号, 则只能是"))(("的形式, 否则会出现至少一对匹配的括号, 如"()()"或"(())".

由**定理**: 可分别求添加左括号数的方案数和添加右括号数的方案数, 最终答案为两者之积.

考察如何求添加左括号的方案数. 为使得统计方案时不重复, 以右括号为分界线, 将序列分为若干段, 每一段中只有若干个(可能为0个)左括号. 显然每个方案与一组每一段中左括号的个数一一对应. 注意到在最后一段中添加左括号是不优的, 则左括号只会添加在以右括号结尾的段中, 而每一段中只有左括号, 则可规定只在每一段的右括号之前添加左括号, 这等价于枚举每一段中左括号的个数.

$dp[i][j]$表示只考虑前i个括号, 且左括号数 $-$ 右括号数 $=j$的方案数. 初始条件$dp[0][0]=1$, 最少需添加的左括号数为第一个非零的$dp[n][i]\ \ (i=0,\cdots,n)$. 显然必存在这样的$dp[n][i]$.

设括号序列为s. 按$s[i]$是左括号或右括号分类:

①若$s[i]$是左括号, 则$dp[i][j]=dp[i-1][j-1]$.

②若$s[i]$是右括号, 枚举在其之前添加的右括号数$k=0,\cdots,(j+1)$, 则$dp[i][j]=\sum_{k=0}^{j+1}dp[i-1][k]\ \ (*)$.

对该情况, 暴力转移时间复杂度$O(n)$, 总时间复杂度$O(n^3)$, 会TLE.

考虑优化, 类似于完全背包, 有: $dp[i][j-1] = \sum_{k=0}^{j} dp[i-1][k]$,

代入$(*)$式得: $dp[i][j] = dp[i-1][j+1] + dp[i][j-1]$.

因$j = 0$时, $(j-1)$越界, 需特判$j = 0$的情况, 此时$dp[i][0] = dp[i-1][0] + dp[i-1][1]$.

综上, 状态转移方程$dp[i][j] = \begin{cases} dp[i-1][j-1], s[i] = \text{'('} \\ dp[i-1][0] + dp[i-1][1], s[i] = \text{')'} \wedge j = 0 \\ dp[i-1][j+1] + dp[i][j-1], s[i] = \text{')'} \wedge j \neq 0 \end{cases}$.

求添加右括号数的方案数的做法与上述做法对称, 可将上述DP过程写成一个函数, 将括号序列翻转(左括号变右括号, 右括号变左括号), 再调用该函数即可.

下面的代码存在漏洞: $dp[n][i] = 1e9 + 7$时, 取模后为0, 导致最少需添加的左括号数错误.

代码

```cpp
const int MAXN = 5005;
const int MOD = 1e9 + 7;
int n;
char s[MAXN];

int cal() {
    vector<vector<int>> dp(n + 1, vector<int>(n + 1));
    dp[0][0] = 1;  // 初始条件
    for (int i = 1; i <= n; i++) {
        if (s[i] == '(') {
            for (int j = 1; j <= n; j++)
                dp[i][j] = dp[i - 1][j - 1];
        }
        else {
            dp[i][0] = (dp[i - 1][0] + dp[i - 1][1]) % MOD;  // 特判j = 0的情况
            for (int j = 1; j <= n; j++)
                dp[i][j] = (dp[i - 1][j + 1] + dp[i][j - 1]) % MOD;
        }
    }

    // 求第一个非零的dp[n][i]  (i = 0, ..., n), 但此处有漏洞
    for (int i = 0; i <= n; i++)
        if (dp[n][i]) return dp[n][i];
    return 1;  // 若无卡上述漏洞的数据, 则不会执行到此处
}

void solve() {
    cin >> s + 1;
    n = strlen(s + 1);

    int lans = cal();

    reverse(s + 1, s + n + 1);
    for (int i = 1; i <= n; i++) s[i] ^= '(' ^ ')';

    int rans = cal();
    cout << (ll)lans * rans % MOD << endl;
```

```
}

int main() {
    solve();
}
```

20.2.24 How many trees?

原题指路: https://codeforces.com/problemset/problem/9/D

题意

给定两个整数n, h $(1 \le h \le n \le 35)$, 求包含n个节点的、高度$\ge h$的不同的二叉树的个数. 数据保证答案不超过$9e18$.

思路

考虑容斥, 高度$\ge h$的二叉树的个数 = 总方案数 − 高度$\le h-1$的二叉树的个数.

$dp[i][j]$表示用i个节点组成高度$\le j$的二叉树的方案数. 初始条件$dp[0][j]=1$ $(0 \le j \le n)$, 最终答案$ans = dp[n][n] - dp[n][h-1]$.

对固定的高度j和总节点数i, 按左子树的节点数k $(0 \le k \le i-1)$分类, 状态转移方程$dp[i][j] += dp[k][j-1] \cdot dp[i-k-1][j-1]$.

状态数$O(n^2)$, 转移$O(n)$, 总时间复杂度$O(n^2)$.

代码

```
void solve() {
    int n, h; cin >> n >> h;

    // dp[i][j]表示用i个节点组成高度 <= j的二叉树的方案数
    vector<vector<ll>> dp(n + 1, vector<ll>(n + 1));
    for (int j = 0; j <= n; j++) dp[0][j] = 1;  // 初始条件

    for (int j = 1; j <= n; j++) {  // 枚举高度
        for (int i = 1; i <= n; i++) {  // 枚举总节点数
            for (int k = 0; k < i; k++)  // 枚举左子树的节点数
                dp[i][j] += dp[k][j - 1] * dp[i - k - 1][j - 1];
        }
    }
    cout << dp[n][n] - dp[n][h - 1] << endl;
}

int main() {
    solve();
}
```

20.2.25 Modular Stability

原题指路: https://codeforces.com/problemset/problem/1359/E

题意

称一个正整数序列 $a=[a_1,\cdots,a_k]$ 是好的, 如果对 $\forall 1\sim k$ 的排列 $p[]$ 和 $\forall x\in\mathbb{N}$, 都有 $(((x \bmod a_1) \bmod a_2)\cdots \bmod a_{k-1}) \bmod a_k = (((x \bmod a_{p_1}) \bmod a_{p_2})\cdots \bmod a_{p_{k-1}}) \bmod a_{p_k}$. 给定两个整数 n,k $(1\le n,k\le 5e5)$, 求有多少个长度为 k 的好的序列 $a=[a_1,\cdots,a_k]$ $(1\le a_1<\cdots<a_k\le n)$, 答案对 998244353 取模.

思路

[**定理**] 序列是好的 iff 其元素都是最小元素的倍数.

[**证**]

(充) 先考虑两个元素的情况. 设模数 a,b $(b>a)$, 为使得 $(x \bmod a) \bmod b = (x \bmod b) \bmod a$,

因 $b>a$, 则 $(x \bmod a) \bmod b = x \bmod a$, 进而 $(x \bmod b)\equiv x \pmod a$.

设 $x=tb+y$ $(t\in\mathbb{Z})$, 则 $y\equiv tb+y \pmod a$, 由 t 的任意性: $a\mid b$.

归纳知: $S=(((x \bmod a_1) \bmod a_2)\cdots \bmod a_{k-1}) \bmod a_k$ 的值与 $a[]$ 中元素的顺序无关

```
iff $a[]$ 中的元素都是其最小元素的倍数.
```

(必) 若 $\exists a_i$ $(2\le i\le n)$ $s.t.$ $minA\nmid a_1$, 取 $x=a_i$, 考察如下两序列:

① $a=[a_1,\cdots,a_k]$, 其 $S=x \bmod a_1=a_i \bmod a_1\neq 0$,

② $a'=[a_i,a_1,\cdots,a_{i-1},a_{i+1},\cdots,a_k]$, 其 $S'=x \bmod a_i=a_i \bmod a_i=0\neq S$.

枚举 $a[]$ 的最小元素 i, 则 $a[]$ 中的其他元素都为该元素的倍数.

因 $1\sim n$ 中有 $d=\left\lfloor\frac{n}{i}\right\rfloor$ 个 i 的倍数, 且 $a_1<\cdots<a_k$,

则方案数为在除 i 外的 $(d-1)$ 个 i 的倍数中取 $(k-1)$ 个数作为 $a[]$ 中的元素的方案数,

即 C_{d-1}^{k-1}, 情况存在 iff $d-1\ge k-1$.

代码

```
const int MOD = 998244353;
const int MAXN = 5e5 + 5;
Z fac[MAXN], ifac[MAXN];

void init() {
    fac[0] = 1;
    for (int i = 1; i < MAXN; i++) fac[i] = fac[i - 1] * i;

    ifac[MAXN - 1] = fac[MAXN - 1].inv();
    for (int i = MAXN - 1; i; i--) ifac[i - 1] = ifac[i] * i;
}

Z C(int n, int m) {
    if (n <= m) return n == m;
    else return fac[n] * ifac[m] * ifac[n - m];
```

```
}

void solve() {
    init();

    int n, k; cin >> n >> k;

    Z ans = 0;
    for (int i = 1; i <= n; i++)  // 枚举a[]中的最小元素
        ans += C(n / i - 1, k - 1);
    cout << ans << endl;
}

int main() {
    solve();
}
```

20.3 Stirling数

20.3.1 第一类Stirling数

题意

圆排列指从n个不同的元素中取m $(1 \le m \le n)$个元素排成一个环形.两圆排列相同当且仅当所取的元素相同,且在环上的排列顺序一致.

第一类Stirling数(Stirling轮换数)$s(n,k)$或$\begin{bmatrix} n \\ k \end{bmatrix}$表示将$n$个不同的元素划分为$k$个非空圆排列的方案数.给定整数$n,k$ $(1 \le k \le n \le 1000)$,求$\begin{bmatrix} n \\ k \end{bmatrix}$,答案对$1e9+7$取模.

思路

[**递推公式**] $\begin{bmatrix} n \\ k \end{bmatrix} = \begin{bmatrix} n-1 \\ k-1 \end{bmatrix} + (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix}$.

[**证**] 将$\begin{bmatrix} n \\ k \end{bmatrix}$按第$n$个数的放法分类:

①若第n个数新开一个圆排列,则当前有$(n-1)$个数和$(k-1)$个圆排列,方案数为$\begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$.

②若第n个数插到已有的圆排列中某个数之后,则当前有$(n-1)$个数和k个圆排列,方案数为$(n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix}$.

[**规定**] $\begin{bmatrix} 0 \\ 0 \end{bmatrix} = 1, \begin{bmatrix} n \\ 0 \end{bmatrix} = 0.$

[**性质1**] $\begin{bmatrix} n \\ n \end{bmatrix} = 1.$

[**证**] 只能一个数在一个圆排列中.

[**性质2**] $\begin{bmatrix} n \\ 1 \end{bmatrix} = (n-1)!$.

[**证**] 即将n个数圆排列的方案数.n个数排成链的方案数为$n!$,而顺时针旋转$0 \sim (n-1)$可得到n个相同的圆排列,

故圆排列方案数为$\dfrac{n!}{n} = (n-1)!$.

[**性质3**] $\begin{bmatrix} n \\ n-1 \end{bmatrix} = C_n^2$.

[**证**] 由抽屉原理:存在一个圆排列中有两个数,则方案数即从n个数中任取两个数的方案数,即C_n^2.

[**性质4**] $\begin{bmatrix} n \\ 2 \end{bmatrix} = (n-1)! \cdot \sum\limits_{i=1}^{n-1} \dfrac{1}{i}$.

[**证**] 设第一个圆排列中有i个数,则第二个圆排列中有$(n-i)$个数.因两圆排列无序,

方案数为

$$\frac{1}{2} C_n^i \cdot (i-1)!(n-i-1)! = \frac{1}{2} \cdot \frac{n!}{i!(n-i)!}(i-1)!(n-i-1)! = \frac{1}{2} \cdot \frac{n!}{i(n-i)} = \frac{1}{2}(n-1)! \cdot \frac{n}{i(n-i)}.$$

注意到$\sum\limits_{i=1}^{n-1} \dfrac{n}{i(n-i)} = \sum\limits_{i=1}^{n-1} \left(\dfrac{1}{i} + \dfrac{1}{n-i} \right)$,其中$\dfrac{1}{i}$和$\dfrac{1}{n-i}$都遍历$\dfrac{1}{1} \sim \dfrac{1}{n-1}$,

故$\begin{bmatrix} n \\ 2 \end{bmatrix} = \dfrac{1}{2}(n-1)! \sum\limits_{i=1}^{n-1} \dfrac{2}{i} = (n-1)! \cdot \sum\limits_{i=1}^{n-1} \dfrac{1}{i}$.

[**性质5**] $\begin{bmatrix} n \\ n-2 \end{bmatrix} = 2C_n^3 + 3C_n^4$.

[**证**] ①有$(n-3)$个只含一个元素的圆排列和1个含3个元素的圆排列时,先从n个数选3个数,再求这3个数的圆排列,

方案数为$C_n^3 \cdot (3-1)! = 2C_n^3$.

②有$(n-4)$个只含一个元素的圆排列和2个含2个元素的圆排列,先从n个数选4个数,再将这4个数等分为两组,

方案数为$C_n^4 \dfrac{C_4^2}{A_2^2} = 3C_n^4$.

[**性质6**] 第一类Stirling数的生成函数是升阶函数,即$x^{\overline{n}} = x(x+1)\cdots(x+n-1) = \sum\limits_{k=0}^{n} \begin{bmatrix} n \\ k \end{bmatrix} x^k$.

[**证**] 设$(n-1)$时结论成立.为凑出x^k,考察最后一项$(x+n-1)$的选法:

①若最后一项选x,则需乘上前$(n-1)$项中x^{k-1}的系数,即$\begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$,再乘上$x$的系数$1$得$\begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$.

②若最后一项选$(n-1)$,则需乘上前$(n-1)$项中x^k的系数,即$\begin{bmatrix} n-1 \\ k \end{bmatrix}$,再乘上$(n-1)$得$(n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix}$.

由第一类Stirling数的递推公式即证.

[**推论**] 令$x=1$,得$\sum\limits_{k=0}^{n} \begin{bmatrix} n \\ k \end{bmatrix} = n!$.

代码

```cpp
const int MAXN = 1005;
const int MOD = 1e9 + 7;

int n, m;
int dp[MAXN][MAXN];  // dp[n][k]表示第一类Stirling数s(n,k)

void solve() {
  cin >> n >> m;

  dp[0][0] = 1;  // 初始化
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++)
      dp[i][j] = ((ll)dp[i - 1][j - 1] + (ll)(i - 1) * dp[i - 1][j]) % MOD;
  }
  cout << dp[n][m];
}

int main() {
  solve();
}
```

20.3.2 第二类Stirling数

题意

第二类Stirling数(Stirling子集数)$S(n,k)$或$\begin{Bmatrix} n \\ k \end{Bmatrix}$表示将$n$个不同的元素划分为$k$个非空子集的方案数.给定整数$n,k\ \ (1 \le k \le n \le 1000)$,求$\begin{bmatrix} n \\ k \end{bmatrix}$,答案对$1e9+7$取模.

思路

[**递推公式**] $\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k\begin{Bmatrix} n-1 \\ k \end{Bmatrix}$.

[**证**] 将$\begin{Bmatrix} n \\ k \end{Bmatrix}$按第$n$个数的放法分类:

①若第n个数新开一个集合,则当前有$(n-1)$个数和$(k-1)$个集合,方案数为$\begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix}$.

②若第n个数放到已有的集合,则当前有$(n-1)$个数和k个集合,方案数为$k\begin{Bmatrix} n-1 \\ k \end{Bmatrix}$.

[**通项公式**] $\begin{Bmatrix} n \\ m \end{Bmatrix} = \frac{1}{m!}\sum_{k=0}^{m}(-1)^k C_m^k (m-k)^n$.

[**规定**] $\begin{Bmatrix} n \\ 0 \end{Bmatrix} = 0^n = \begin{cases} 1, n=0 \\ 0, n \ge 1 \end{cases}$.

[**性质1**] $\left\{ {n \atop 1} \right\} = 1.$

[**性质2**] $\left\{ {n \atop n} \right\} = 1.$

[**性质3**] $\left\{ {n \atop 2} \right\} = 2^{n-1} - 1.$

[**证**] 若允许集合为空,用一个n位二进制数表示每个数放进哪个集合,其中0表示放入第一个集合,1表示放入第二个集合,则方案数为2^n,减去全为0和全为1两个有空集的状态,方案数为$2^n - 2$.

因两集合无序,故方案数为$\dfrac{2^n - 2}{2} = 2^{n-1} - 1.$

[**性质4**] $\left\{ {n \atop n-1} \right\} = C_n^2.$

[**证**] 有$(n-2)$个只含一个元素的集合和1个含两个元素的集合.

[**性质5**] $\left\{ {n \atop n-2} \right\} = C_n^3 + 3C_n^4.$

[**证**] ①有$(n-3)$个只含一个元素的集合和1个含三个元素的集合.

②有$(n-4)$个只含一个元素的集合和2个含两个元素的集合.

[**性质6**] $\left\{ {n \atop 3} \right\} = \dfrac{1}{2}(3^{n-1} + 1) - 2^{n-1}.$

[**性质7**] $\left\{ {n \atop n-3} \right\} = C_n^4 + 10C_n^5 + 15C_n^6.$

[**性质8**] $\sum_{k=0}^{n} \left\{ {n \atop k} \right\} = B_n$,其中$B_n$为Bell数.

[**与第一类Stirling数的关系**] $\sum_{k=0}^{n} \left\{ {n \atop k} \right\} \left[{n \atop k} \right] = \sum_{k=0}^{n} \left[{n \atop k} \right] \left\{ {n \atop k} \right\}.$

代码

```cpp
const int MAXN = 1005;
const int MOD = 1e9 + 7;

int n, m;
int dp[MAXN][MAXN];  // dp[n][k]表示第二类Stirling数S(n,k)

```

```cpp
void solve() {
  cin >> n >> m;

  dp[0][0] = 1;  // 初始化
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++)
      dp[i][j] = ((ll)dp[i - 1][j - 1] + (ll)j * dp[i - 1][j]) % MOD;
  }
  cout << dp[n][m];
}

int main() {
  solve();
}
```

20.3.3 建筑师

题意

在数轴上建n个建筑,高度是$1 \sim n$的一个排列.要求从最左边能看到a个建筑,从最右边能看到b个建筑,求方案数,答案对$19+7$取模.

有t $(1 \le t \le 2e5)$组测试数据.每组测试数据第一行输入三个整数n, a, b $(1 \le n \le 5e4, 1 \le a, b \le 100)$.

思路

以高度为n的建筑物为分界,将数轴划分为左右两边,则左边需看到$(a-1)$个建筑,右边需看到$(b-1)$个建筑.

①左右两边分别将一个建筑物与被它挡住的建筑物作为一个块,称块中最高建筑的高度为块的高度,

则左边需放$(a-1)$个块,右边需放$(b-1)$个块,即$(n-1)$个建筑需划分为$(a+b-2)$个块.

②设第i个块有a_i个建筑,则最高的建筑只能放第一个,剩下的(a_i-1)个建筑任意排列,方案数为$(a_i-1)!$.

结合①和②,方案数即为将$(n-1)$个不同的元素划分为$(a+b-2)$个圆排列的方案数,即$\begin{bmatrix} n-1 \\ a+b-2 \end{bmatrix}$.

在$(a+b-2)$个块中选$(a-1)$个放在左边,方案数为C_{a+b-2}^{a-1}.将左边按块的高度升序排列,右边按块的高度升序排列即可.

综上,总方案数为$\begin{bmatrix} n-1 \\ a+b-2 \end{bmatrix} \cdot C_{a+b-2}^{a-1}$.

代码

```cpp
const int MAXN = 5e4 + 5, MAXM = 205;
const int MOD = 1e9 + 7;
int stir[MAXN][MAXM];  // 第一类Stirling数
int C[MAXM][MAXM];  // 组合数C(n,m)

void init() {  // 预处理stir[][]和C[][]
  stir[0][0] = 1;
  for (int i = 1; i < MAXN; i++) {
    for (int j = 1; j < MAXM; j++)
```

```cpp
      stir[i][j] = ((ll)stir[i - 1][j - 1] + (ll)(i - 1) * stir[i - 1][j]) % MOD;
  }

  for (int i = 0; i < MAXM; i++) {
    for (int j = 0; j <= i; j++) {
      if (!j) C[i][j] = 1;
      else C[i][j] = ((ll)C[i - 1][j] + C[i - 1][j - 1]) % MOD;
    }
  }
}

void solve() {
  int n, a, b; cin >> n >> a >> b;
  cout << (ll)stir[n - 1][a + b - 2] * C[a + b - 2][a - 1] % MOD << endl;
}

int main() {
  init();
  CaseT  // 单测时注释掉该行
  solve();
}
```