计蒜客信息学 8 月提高组模拟赛

题目	知识点
Scattered and Lost	思维、分类讨论
Mila 的苹果树	树形 DP
stack	卡特兰数
mahjong	动态规划

Scattered and Lost

题解

由于我们只关心最小值的位置，因此不妨设当前最小值的位置为 p:

- `1 l r`：如果 $p \notin [l,r]$ 自然啥事没有，否则令 p 变为 l。
- `2 l r`：同理，如果包含 p，则令 p 变为 r。
- `3 l r`：如果包含 p，令 $p = r + l - p$。
- `4 l r k`：这个需要讨论一下。
- 如果 $k \geq r - l + 1$ 那么当 $p \in [l,r]$ 时令 `p+=k`，当 $p \in [l+k, r+k]$ 时令 `p-=k`。
- 如果 $k \leq r - l$:
- $p \in [l, l+k)$，那么 p 会一直往后换，一直到 $p + tk$，其中 t 是满足 $p + tk \leq r + k$ 的最大正整数 t。
- $p \in [l+k, r+k]$，那么 p 只会在第一次被交换后就变成了 $p - k$，后面的交换都与它无缘。故只需令 `p-=k`。

时间复杂度为 $O(n+q)$。

标程

```cpp
#include <bits/stdc++.h>
#define int long long
using namespace std;
inline int read() {
    int x = 0, f = 1;
    char c = getchar();
    for (; (c < '0' || c > '9'); c = getchar()) {
        if (c == '-') {
            f = -1;
        }
    }
    for (; (c >= '0'&& c <= '9'); c = getchar()) {
        x = x * 10 + (c& 15);
    }
    return x * f;
}
const int MN = 2e5 + 5;
int n, q, p;
void solve() {
    n = read(), q = read();
    int mn = 1e9 + 1;
    for (int i = 1; i <= n; i++) {
        int x = read();
        if (x < mn) {
            mn = x, p = i;
        }
    }
    while (q--) {
        int op = read(), l = read(), r = read();
        if (op == 1 && l <= p && p <= r) {
            p = l;
        }
        if (op == 2 && l <= p && p <= r) {
            p = r;
        }
        if (op == 3 && l <= p && p <= r) {
            p = r + l - p;
        }
        if (op == 4) {
            int k = read();
            if (k >= r - l + 1) {
                if (l <= p && p <= r) {
                    p += k;
                } else if (l + k <= p && p <= r + k) {
                    p -= k;
                }
                continue;
            }
            if (l <= p && p < l + k) {
                int t = (int)((r - p + k) / k);
```

```
                p += t * k;
            } else if (l + k <= p && p <= r + k) {
                p -= k;
            }
        }
    }
    cout << p << endl;
}
signed main(void) {
    freopen("resort.in", "r", stdin);
    freopen("resort.out", "w", stdout);
    int t = read();
    while (t--) {
        solve();
    }
    return 0;
}
```

Mila 的苹果树

题解

不难想到树形 DP，那么就能够自然地定义出状态：$f_{i,j}$ 表示 i 的子树内放了 j 个苹果的方案数。

转移的时候，考虑 x 的一棵子树 y，将 y 的信息和 x 之前的儿子的信息合并起来，可以得到：

$$f'_{x,i}=\sum_{j=0}^{sz_y} f_{y,j}\,f_{x,i-j}$$

其中，sz_y 表示 y 的子树大小。

统计完所有儿子后，将自己的 a_x+1 以上的部分清 0，因为这部分是不合法的，即让 $f_{x,[a_x+1,n]}=0$。

咋一看这个 DP 是 $O(n^3)$ 的，其实不然，这是 dp 中的经典套路，仔细思考可以发现这其实是 $O(n^2)$ 的。

证明也不难，合并子树时，$f_{x,i}$ 和 $f_{y,j}$ 乘起来贡献 $f'_{x,i+j}$，不妨换个角度来看，可以看成 x 之前的子树中第 i 个点和 y 的子树中第 j 个点乘起来做贡献。那么仔细观察，其实每个点只会和其他点乘起来贡献一次，贡献的对象就是他们的 lca。所以总共的贡献次数只有 n^2 次。

标程

```cpp
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <vector>
using namespace std;
const int NMAX = 2100;
const int MOD = 998244353;
vector<int> E[NMAX];
int limit[NMAX], siz[NMAX];
int dp[NMAX][NMAX];
void solve(int x, int fa) {
    siz[x] = 1;
    dp[x][0] = 1;
    if (limit[x] >= 1) {
        dp[x][1] = 1;
    }
    for (auto y: E[x]) {
        if (y == fa) {
            continue;
        }
        solve(y, x);
        for (int i = siz[x] + siz[y]; i >= 0; i -= 1) {
            if (i > limit[x]) {
                dp[x][i] = 0;
                continue;
            }
            int j_init = max(i - siz[x], 1);
            int j_lim = min(siz[y], i);
            for (int j = j_init; j <= j_lim; j += 1) {
                (dp[x][i] += (long long )dp[y][j] * dp[x][i - j] % MOD) %= MOD;
            }
        }
        siz[x] += siz[y];
    }
}
int main() {
    freopen("appletree.in", "r", stdin);
    freopen("appletree.out", "w", stdout);
    int n;
    scanf("%d", &n);
    for (int i = 1; i <= n; i += 1) {
        scanf("%d", &limit[i]);
    }
    for (int i = 1; i < n; i += 1) {
        int x, y;
        scanf("%d %d", &x, &y);
        E[x].push_back(y);
        E[y].push_back(x);
    }
    solve(1, -1);
```

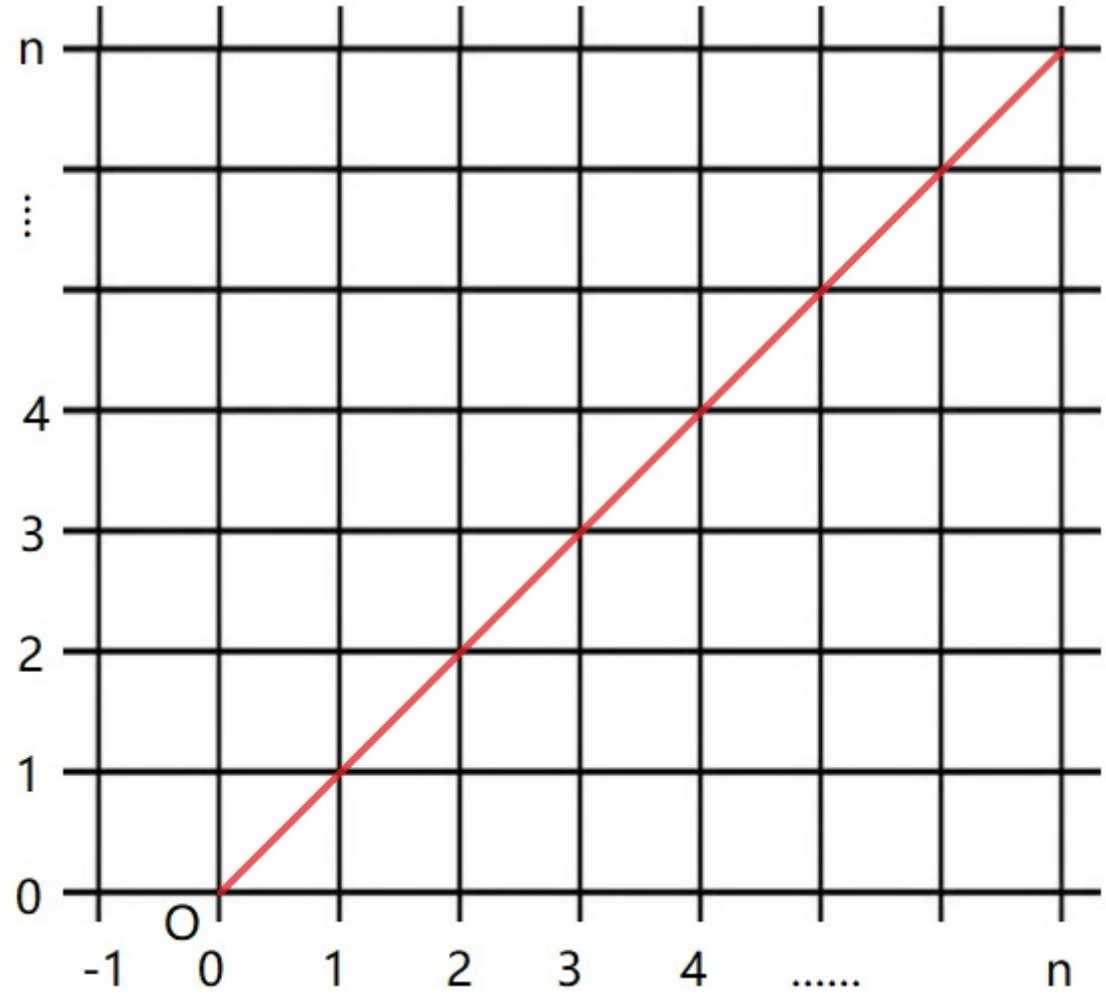
```
    int ans = 0;
    for (int i = 0; i <= limit[1] && i <= n; i += 1) {
        (ans += dp[1][i]) %= MOD;
    }
    printf("%d\n", ans);
    exit(0);
}
```

stack

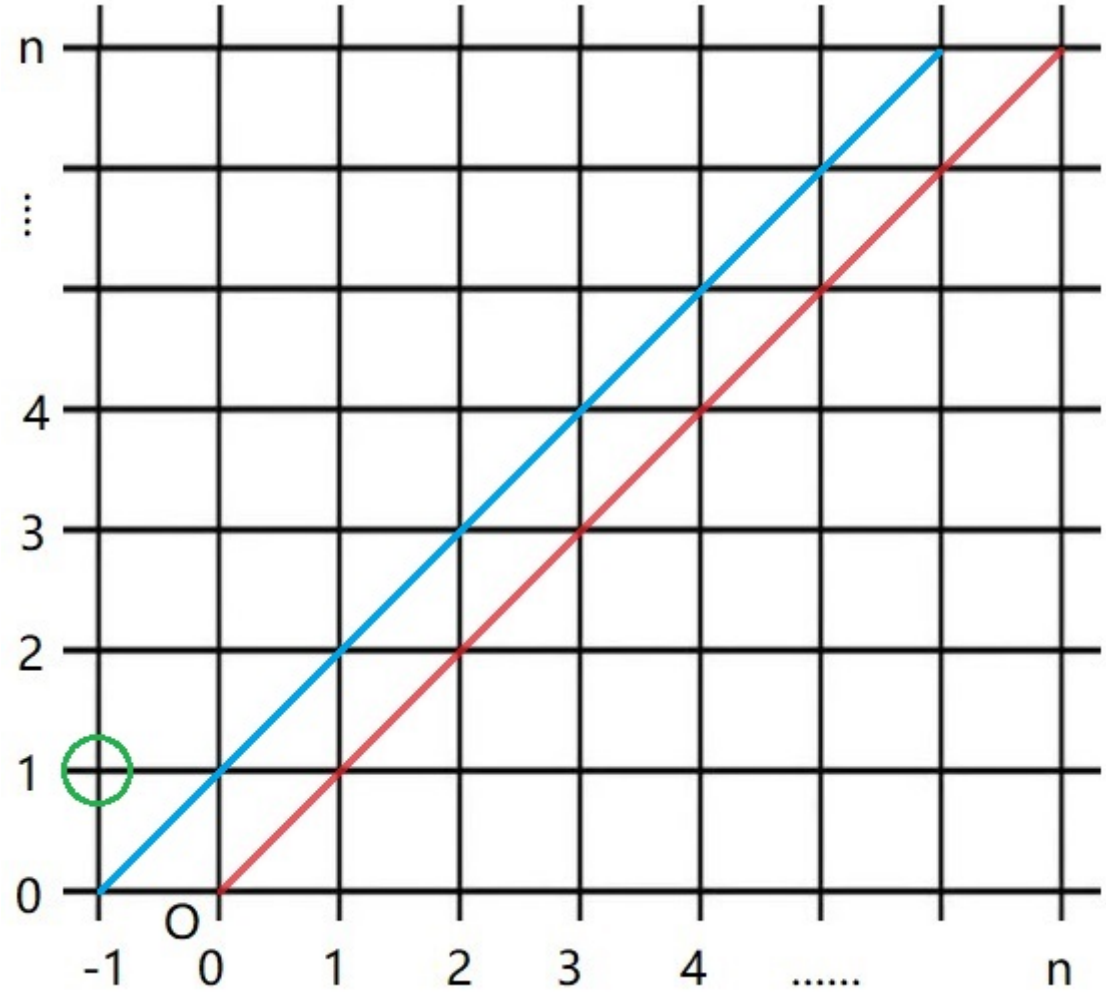
题解

首先对于任意一个最终的输出数组可以通过反推，得到一个唯一的操作序列，所以说合法的操作序列与输出数组是一一对应的，也就是说输出数组的种类数等价于合法的操作序列数。所以只需要考虑合法的方案数即可。

考虑一个合法的输出时如何产生的，因为开头必须为 1，所以可以先指定开头的元素，那么要将这个元素前面的元素都先放入栈中，这个元素才能作为第一个。假设这个元素为第 i 个，那么 $1 \sim i-1$ 在栈中，$i+1 \sim n$ 在队列中。也就是说问题变成了一个栈中已经存在一定元素出出栈序列数。考虑一个经典的转化，出栈数计数这类问题可以放到方格图中，如下



其中经典的 n 不同元素出栈序列问题是从 $(0,0)$ 出发到 (n,n)，但是不能到红线上方的区域的方案数。也就是将入栈等同于向右走，出栈等同于向上走，因为要保证在任意一个出栈操作前已经有入栈的个数必须大于出栈的个数所以存在红线的限制。



要解决这个问题。可以发现对于任意的越过红线的方案都会经过蓝线，所以考虑那些不合法的方案，也就是必定经过蓝线的方案。找到起点 $(0,0)$ 关于蓝线的对称点 $(-1,1)$，可以发现从 $(-1,1)$ 除法的每一条到 (n,n) 的路径都会经过蓝线，再将所有这些路径以第一次经过蓝线上的点为界限将前面按蓝线对称，这样起点又边了 $(0,0)$，并且不难发现这些方案与那些不合法方案是一一对应的（不理解的话可以自己画几条路径，总能找到为一条从 $(-1,1)$ 出发的路径按上述方法与之对应）

于是可以得出 $f(n)=\binom{2n}{n}-\binom{2n}{n-1}=\frac{\binom{2n}{n}}{n+1}$，也就是著名的卡特兰数。

现在相当于起点位置变成了 $(i-1,0)$ 终点变成了 $(n-1,n-1)$，做法仍然类似，找到起点关于蓝线对称的点，计算其到终点的方案数便是不合法的方案数。对称后点的坐标为 $(-1,i)$，所以答案就是 $\binom{2n-i-1}{n-1}-\binom{2n-i-1}{n}$。

标程

```cpp
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e6 + 10;
const int mod = 998244353;
int n, f[maxn];
int fac[maxn << 1];
int inv[maxn << 1];
int qmi(int a, int b, int mod) {
    a %= mod;
    if (!a) {
        return 0;
    }
    int result = 1;
    for (; b; a = 1ll * a * a % mod, b >>= 1) {
        if (b& 1) {
            result = 1ll * result * a % mod;
        }
    }
    return result;
}
int C(const int n, const int m) {
    if (n < m) {
        return 0;
    }
    return 1ll * fac[n] * inv[m] % mod * inv[n - m] % mod;
}
int cal(int a, int b) {
    return(1ll * C(a + 2 * b, b) - C(a + 2 * b, b - 1) + mod) % mod;
}
void init(const int n) {
    fac[0] = 1;
    for (int i = 1; i <= n; i++) {
        fac[i] = 1ll * fac[i - 1] * i % mod;
    }
    inv[n] = qmi(fac[n], mod - 2, mod);
    for (int i = n - 1; i >= 0; i--) {
        inv[i] = inv[i + 1] * (i + 1ll) % mod;
    }
}
int main() {
    freopen("stack.in", "r", stdin);
    freopen("stack.out", "w", stdout);
    scanf("%d", &n);
    init(n << 1);
    for (int i = 1; i <= n; i++) {
        scanf("%d", &f[i]);
    }
    int ans = 0;
    for (int i = 1; i <= n; i++) {
        if (f[i]) {
```

```
            ans = (ans + cal(i - 1, n - i)) % mod;
        }
    }
    cout << ans << endl;
    return 0;
}
```

mahjong

题解

算法一

暴力枚举麻将牌集合，然后贪心判断即可。

算法二

可以手玩方案数！

算法三

显然能够发现如果设 $f_{i,0/1}$ 表示合法麻将牌集合 S 的大小为 i，且这些牌的花色相同，其中包不包含一个对子的数量，则能用组合数 $O\left(k^3\right)$ 求出答案。

具体而言，设 $g_{i,j}$ 表示共有 i 种花色，集合大小为 j 的方案数。那么可以用 f 背包求出 g ，然后再用 g 组合数即可。

考虑如何求出 $f_{i,0/1}$，我们能够设一个 dp，然后从 1 到 n 依次考虑每个大小的牌加多少个，现在

需要判断的是加进去之后是否合法。

转化为一个这样的命题：给定一个集合，能否胡牌。有一个简单的 dp，设 $dp_{i,s_1,s_2,0/1}=[0,1]$ 表示当前考虑第 i 张牌，从 $i-2$ 开始的顺子有 s_1 个，从 $i-1$ 开始的顺子有 s_2 个，有没有对子的情况是否合法。转移显然。

发现 $(s_1,s_2,0/1)$ 的总状态数量为 18，也就是说，用一个大小为 18 的数组就能包括当前麻将牌集合能表示出的所有状态。

那么我们设 $dp_{i,j,S}$ 为，当前加入到第 i 张牌，一共加入了 j 张牌，能表示状态的集合为 S 的方案数。转移显然。

不过 2^{18} 种方案显然是非常大的，我们能从初始状态预处理出能到达的所有状态，经打表一共只有 68 种。

68 种的预处理可以参考 ZJOI2019 麻将的 bfs 写法，接着套用上面那个 dp 就行了。

时间复杂度：$O(68nk)$。

算法四

经实践，当 k 固定时，能用 BM 算法暴力求出递推式，可以通过 $k=1,2,3,4$ 的部分分。

算法五

k 较小的情况下，发现有大量的数字都不会被加入。

那么我们只需考虑若干连续的数字组成的方案数，然后再合并即可。

考虑每一个合法的仅有一种花色且集合大小为 s 的牌，显然 $s=3n+2(n\in\mathbb{N})$，且能够将其分为若干段，这若干段每段的数字连续（可重复），但是不同段之间的数字不连续，显然对每段的牌内部也应是合法的。那么设 f_i 表示分出来的某一段长度为 i 的方案数，这个可以用**算法三**中的方法计算，然后再令 $g_{i,j}$ 表示同一种花色共有 i 段，集合大小为 j 的方案数，这个可以用求得的 f 背包得到。

注意这个 $g_{i,j}$ 不考虑具体的牌的大小，但是考虑不同段之间大小的相对顺序。

- 例如对于牌组 $\{2,3,4,6,7,8,9,9\}$ 在 $g_{i,j}$ 中与 $\{1,2,3,6,7,8,9,9\}$ 相同，但是与 $\{2,3,4,5,5,7,8,9\}$ 不相同。

求得 $g_{i,j}$ 后可以用组合数求出考虑具体牌的大小后的方案数，最后用求得的 $g_{i,j}$ 再组合数背包求出考虑多个花色的答案。

时间复杂度：$O\left(68\times 9\times k^3\right)$。

算法六

我们发现被卡常了，想一下怎么优化常数。

对于每个状态 S，当前合法的麻将牌集合大小模 3 的余数一定相等。

发现常数主要集中在求 f 的地方，然后发现对预处理的 68 种状态，满足某个状态的牌，一定满足数量大小模 3 的余数大小一定。于是可以用这个性质优化常数，再优化一下循环的上下界即可。

使用类似思想可以优化 9 倍常数。

时间复杂度：$O\left(68k^3\right)$。

标程

```cpp
#include <algorithm>
#include <iostream>
#include <iomanip>
#include <cstring>
#include <cstdio>
#include <cmath>
#include <queue>
#include <ctime>
#include <map>
using namespace std;
#define mod 998244353
#define ll long long
int dp[2][2][305][70][305], n, m, k, K, g[305][305][305][2], h[305][305][2], f[305], inv
struct data {
    int dp[3][3];
    data() {
        memset(dp, 0, sizeof(dp));
    }
    friend bool operator <(const data a, const data b) {
        for (int i = 0; i <= 2; i++) {
            for (int t = 0; t <= 2; t++) {
                if (a.dp[i][t] != b.dp[i][t]) {
                    return a.dp[i][t] < b.dp[i][t];
                }
            }
        }
        return 0;
    }
    friend bool operator !=(const data a, const data b) {
        for (int i = 0; i <= 2; i++) {
            for (int t = 0; t <= 2; t++) {
                if (a.dp[i][t] != b.dp[i][t]) {
                    return 1;
                }
            }
        }
        return 0;
    }
    void DP(data& c, int del) {
        for (int i = 0; i <= 2; i++) {
            for (int t = 0; t <= 2; t++) {
                if (dp[i][t]) {
                    if (del >= i + t) {
                        c.dp[t][(del - i - t) % 3] |= 1;
                    }
                }
            }
        }
    }
};
```

```
struct node {
    data is_pair[2];
    void clear() {
        is_pair[0] = data();
        is_pair[1] = data();
    }
    friend bool operator <(const node a, const node b) {
        if (a.is_pair[0] != b.is_pair[0]) {
            return a.is_pair[0] < b.is_pair[0];
        }
        if (a.is_pair[1] != b.is_pair[1]) {
            return a.is_pair[1] < b.is_pair[1];
        }
        return 0;
    }
    friend node operator +(node a, int b) {
        node c;
        if (b >= 2) {
            a.is_pair[0].DP(c.is_pair[1], b - 2);
        }
        a.is_pair[0].DP(c.is_pair[0], b);
        a.is_pair[1].DP(c.is_pair[1], b);
        return c;
    }
} st, sav[205];
map<node, int> mp;
int tot, c[200005][25];
void bfs() {
    queue<node> q;
    st.clear();
    st.is_pair[0].dp[0][0] = 1;
    q.push(st);
    mp[st] = ++tot;
    sav[tot] = st;
    len[tot] = 0;
    while (!q.empty()) {
        node x = q.front();
        q.pop();
        int now = mp[x];
        for (int i = 1; i <= 4; i++) {
            node nex = x + i;
            if (mp[nex]) {
                c[now][i] = mp[nex];
            } else {
                mp[nex] = ++tot, c[now][i] = tot, q.push(nex), sav[tot] = nex, len[tot]
            }
        }
    }
}
int add(int x) {
    return x >= mod ? x - mod : x;
```

```
}
int ksm(int x, int y) {
    int ans = 1, t = x;
    while (y) {
        if (y& 1) {
            ans = 1ll * ans * t % mod;
        }
        t = 1ll * t * t % mod;
        y >>= 1;
    }
    return ans;
}
int C(int n, int m) {
    if (n < m || m < 0) {
        return 0;
    }
    return 1ll * f[n] * inv[n - m] % mod * inv[m] % mod;
}
int main() {
    freopen("mahjong.in", "r", stdin);
    freopen("mahjong.out", "w", stdout);
    bfs();
    dp[0][0][0][1][0] = g[0][0][0][0] = 1;
    scanf("%d%d%d", &n, &m, &k);
    K = 3 * k + 2;
    for (int s = 0; s <= k; s++) {
        nex ^= 1;
        for (int i = 0; i <= K; i++) {
            int qwq = min(4 * i, K);
            for (int j = 1; j <= tot; j++) {
                for (int t = i - i % 3 + len[j]; t <= qwq; t += 3) {
                    dp[nex][0][i][j][t] = 0;
                }
            }
            for (int j = 1; j <= tot; j++) {
                for (int t = i - i % 3 + (len[j] + 2) % 3; t <= qwq; t += 3) {
                    dp[nex][1][i][j][t] = 0;
                }
            }
        }
        for (int i = 0; i <= K; i++) {
            int qwq = min(4 * i, K);
            for (int t = 0; t <= qwq; t++) {
                g[s + 1][i][t][0] = mod - dp[nex ^ 1][0][i][1][t];
                g[s + 1][i][t][1] = mod - dp[nex ^ 1][1][i][1][t];
            }
        }
        for (int i = 0; i < K; i++) {
            int qwq = min(4 * i, K - 1);
            for (int j = 1; j <= tot; j++) {
                for (int t = i - i % 3 + len[j]; t <= qwq; t += 3) {
```

```
                if (!dp[nex ^ 1][0][i][j][t]) {
                    continue;
                }
                dp[nex ^ 1][0][i + 1][c[j][1]][t + 1] = add(dp[nex ^ 1][0][i + 1][c
                dp[nex ^ 1][0][i + 1][c[j][2]][t + 2] = add(dp[nex ^ 1][0][i + 1][c
                dp[nex ^ 1][0][i + 1][c[j][3]][t + 3] = add(dp[nex ^ 1][0][i + 1][c
                dp[nex ^ 1][0][i + 1][c[j][4]][t + 4] = add(dp[nex ^ 1][0][i + 1][c
            }
        }
    }
    for (int i = 0; i < K; i++) {
        int qwq = min(4 * i, K - 1);
        for (int j = 1; j <= tot; j++) {
            for (int t = i - i % 3 + (len[j] + 2) % 3; t <= qwq; t += 3) {
                if (!dp[nex ^ 1][1][i][j][t]) {
                    continue;
                }
                dp[nex ^ 1][1][i + 1][c[j][1]][t + 1] = add(dp[nex ^ 1][1][i + 1][c
                dp[nex ^ 1][1][i + 1][c[j][2]][t + 2] = add(dp[nex ^ 1][1][i + 1][c
                dp[nex ^ 1][1][i + 1][c[j][3]][t + 3] = add(dp[nex ^ 1][1][i + 1][c
                dp[nex ^ 1][1][i + 1][c[j][4]][t + 4] = add(dp[nex ^ 1][1][i + 1][c
            }
        }
    }
    for (int j = 1; j <= tot; j++) {
        if (sav[j].is_pair[0].dp[0][0]) {
            for (int i = 0; i <= K; i++) {
                int qwq = min(4 * i, K);
                for (int t = i - i % 3; t <= qwq; t += 3) {
                    dp[nex][0][i][1][t] = add(dp[nex][0][i][1][t] + dp[nex ^ 1][0][
                }
                for (int t = i - i % 3 + 2; t <= qwq; t += 3) {
                    dp[nex][1][i][1][t] = add(dp[nex][1][i][1][t] + dp[nex ^ 1][1][
                }
            }
        }
        if (sav[j].is_pair[1].dp[0][0]) {
            for (int i = 0; i <= K; i++) {
                int qwq = min(4 * i, K);
                for (int t = i - i % 3 + 2; t <= qwq; t += 3) {
                    dp[nex][1][i][1][t] = add(dp[nex][1][i][1][t] + dp[nex ^ 1][0][
                }
            }
        }
    }
    for (int i = 0; i <= K; i++) {
        int qwq = min(4 * i, K);
        for (int t = 0; t <= qwq; t++) {
            g[s + 1][i][t][0] = dp[nex][0][i][1][t] = add(g[s + 1][i][t][0] + dp[nex
            g[s + 1][i][t][1] = dp[nex][1][i][1][t] = add(g[s + 1][i][t][1] + dp[nex
        }
```

```
        }
    }
    f[0] = 1;
    for (int i = 1; i <= K; i++) {
        f[i] = 1ll * f[i - 1] * i % mod;
    }
    inv[K] = ksm(f[K], mod - 2);
    for (int i = K - 1; i >= 0; i--) {
        inv[i] = 1ll * inv[i + 1] * (i + 1) % mod;
    }
    for (int i = 0; i <= K; i++) {
        int now = n + 1 - i;
        if (now < 0) {
            break;
        }
        for (int s = 1, w = now, res = now - 1; s <= k + 1; s++, w = 1ll * w * res % mo
            for (int t = 0; t <= K; t++) {
                for (int q = 0; q <= 1; q++) {
                    h[1][t][q] = (h[1][t][q] + 1ll * g[s][i][t][q] * w % mod * inv[s]) %
                }
            }
        }
    }
    h[1][0][0] = 0;
    for (int s = 1; s <= k; s++) {
        for (int i = 0; i <= K; i++) {
            for (int q = 0; q <= 1; q++) {
                for (int j = 0; i + j <= K; j++) {
                    h[s + 1][i + j][q] = (h[s + 1][i + j][q] + 1ll * h[s][i][q] * h[1][
                }
            }
            for (int j = 0; i + j <= K; j++) {
                h[s + 1][i + j][1] = (h[s + 1][i + j][1] + 1ll * h[s][i][0] * h[1][j][1
            }
        }
    }
    for (int s = 1, now = m, res = m - 1; s <= k + 1; s++, now = 1ll * now * res % mod,
        ans = (ans + 1ll * h[s][K][1] * now % mod * inv[s]) % mod;
    }
    printf("%d", ans);
}
```