NOIP 2016 Day 2 题解

组合数问题 problem

知识点

杨辉三角，模运算，前缀和

分析

$n \le 2000$，显然直接 $O(n^2)$ 算一遍组合数是没有问题的。题目中求恰好是 K 的倍数的组合数的个数，是 K 的倍数这个条件可以转化为：不为 0 且对 K 取模为 0。所以我们使用杨辉三角的递推公式来计算组合数，并在计算的时候对 K 取模：

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}$$

考虑快速回答询问，只需要记录 $s_{i,j}$ 为 $n=i, m=j$ 时的答案，我们可以用常见的二维前缀和的思想预处理出 $s_{i,j}$：

$$s_{i,j} = [j \le i, \binom{i}{j} \bmod k = 0] + s_{i-1,j} + s_{i,j-1} - s_{i-1,j-1}$$

复杂度：$O(n^2 + m)$。

代码

```
//   Created by Sengxian on 2016/11/23.
//   Copyright (c) 2016 年  Sengxian. All rights reserved.
#include <bits/stdc++.h>
using namespace std;

const int MAX_N = 2000 + 5, MAX_M = 2000 + 5;
int K, c[MAX_N + 3][MAX_N + 3], s[MAX_N + 3][MAX_N + 3];

void process() {
	memset(c, 0, sizeof c);
	memset(s, 0, sizeof s);
	c[0][0] = 1;
	for (int i = 1; i < MAX_N; ++i) {
		c[i][0] = c[i][i] = 1;
		for (int j = 1; j < i; ++j)
			c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % K;
```

```
}
    for (int i = 0; i < MAX_N - 1; ++i)
        for (int j = 0; j < MAX_M - 1; ++j)
            s[i + 1][j + 1] = (j <= i && c[i][j] == 0) + s[i + 1][j] + s[i][j + 1] - s[i][j];
}

int main() {
    int caseNum;
    scanf("%d%d", &caseNum, &K);
    process();
    while (caseNum--) {
        int n, m;
        scanf("%d%d", &n, &m);
        printf("%d\n", s[n + 1][m + 1]);
    }
    return 0;
}
```

蚯蚓 earthworm

知识点

单调队列

分析

首先我们考虑 $q = 0$ 的情况——蚯蚓长度不增长，那么每次取出的蚯蚓长度一定单调不增。我们可以维护三个单调不增的序列，初始的时候第一个序列是原序列（从大到小排序），第二个序列是左半部分蚯蚓序列（一开始是空的），第三个序列是右半部分蚯蚓序列（一开始是空的）。每一次取出蚯蚓，找到三个序列首最大的蚯蚓，剁成左右两部分加入左半部分蚯蚓序列和右半部分蚯蚓序列的末尾。显然第二、三个序列也是单调不增的，所以整个操作可以在 $O(n \log n + m)$ 的复杂度内完成。

接着考虑 $q \neq 0$ 的情况——蚯蚓长度每次增长，注意到每次只有取出来的蚯蚓不增长，其余的蚯蚓都要 $+q$，那么其实上可以将每一个蚯蚓都 $+q$，然后将剁成的两个蚯蚓 $-q$。每一次取出的蚯蚓长度还是单调递减的！因为每次的操作是取出蚯蚓，剁成两部分，然后 $-q$ 再放回去，整个过程中没有蚯蚓的长度相对增加（整体 $+q$ 是不影响相对长度的），所以我们依然可以套用之前的算法，只不过每次切除的时候要换算一下蚯蚓的长度（因为每次 $+q$ 并不真正直接加上，而是打了一个"标记"）。

复杂度：$O(n \log n + m)$。

代码

```
//    Created by Sengxian on 2016/11/23.
//    Copyright (c) 2016 年  Sengxian. All rights reserved.
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
inline int readInt() {
        static int n, ch;
        n = 0, ch = getchar();
        while (!isdigit(ch)) ch = getchar();
        while (isdigit(ch)) n = n * 10 + ch - '0', ch = getchar();
        return n;
}

const int MAX_N = 100000 + 3, MAX_M = 7000000 + 3;
int n, m, q, u, v, t, a[MAX_N];

int s[3][MAX_M], l[3], r[3];

inline int getMax() {
        static int now, id;
        now = INT_MIN, id = 0;
        for (int i = 0; i < 3; ++i)
                if (r[i] - l[i] >= 1 && s[i][l[i]] > now)
                        now = s[id = i][l[i]];
        return id;
}

void solve() {
        sort(a, a + n), reverse(a, a + n);
        memcpy(s[0], a, sizeof a);
        memset(l, 0, sizeof l);
        memset(r, 0, sizeof r);
        r[0] = n;

        bool flag = false;
        for (int ti = 0, now, id, le, ri, tmp; ti < m; ++ti) {
                id = getMax(), now = s[id][l[id]++];
                tmp = ti * q;
                if ((ti + 1) % t == 0) {
                        if (flag) putchar(' ');
                        else flag = true;
```

```
                printf("%d", now + tmp);
            }
            le = (ll)(now + tmp) * u / v, ri = (now + tmp) - le;
            le -= tmp, ri -= tmp;
            s[1][r[1]++] = le - q, s[2][r[2]++] = ri - q;
        }
        putchar('\n');

        flag = false;
        int tt = m * q;
        for (int i = 0, id, now; i < n + m; ++i) {
            id = getMax(), now = s[id][l[id]++];
            if ((i + 1) % t == 0) {
                if (flag) putchar(' ');
                else flag = true;
                printf("%d", now + tt);
            }
        }
        putchar('\n');
}

int main() {
        n = readInt(), m = readInt(), q = readInt(), u = readInt(), v = readInt(), t =
readInt();
        for (int i = 0; i < n; ++i) a[i] = readInt();
        solve();
        return 0;
}
```

愤怒的小鸟 angrybirds

知识点

状态压缩动态规划，位运算

分析

$n \le 18$，比较明显的状态压缩 DP 的模型。你需要明白，集合是以二进制位来存储的（不懂的话先去做状压 DP 入门题）。

我们设 dp_s 为将集合 s 中的猪全部打掉所需的最小次数，那么我们枚举第一次发射。每次发射要么只打死一头猪，要么打到两头猪 i 和 j，预处理出 $c_{i,j}$ 为打出一条经过猪 i 和 j 的抛物线，**能打掉**的点的集合，注意如果解出经过 i, j 的抛物线 $a \ge 0$，那么没有任何猪能被打掉——因为根本就不合法。

所以有转移方程：

$$dp_s = \min \begin{cases} dp_{s \setminus i} + 1 & i \in s \\ dp_{s \setminus c_{i,j}} + 1 & i \in s, j \in s \end{cases}$$

由于每一次需要枚举 i 和 j，然后还需要在集合 s 中剔除这一次发射能够打掉的点，所以复杂度是 $O\left(n^3 2^n\right)$ 的，有点大，能不能更优呢？我们首先考虑用位运算优化剔除点，设 $c_{i,j}$ 为打出一条经过猪 i 和 j 的抛物线，**不能打掉**的点的集合，那么如果选择 i 和 j 发射，新的集合就是 s and $c_{i,j}$（按位与），我们来看一个例子：

$$\begin{aligned} s &= (1101101)_2 \\ c_{i,j} &= (0110111)_2 \\ s \text{ and } c_{i,j} &= (0100101)_2 \end{aligned}$$

对于 s 中是 0 的位置，表示这个元素已经被打掉了，无论怎么与都是 0。对于 s 中是 1 的位置，如果 $c_{i,j}$ 中这个位置是 1，表示不能打掉这个位置，那么与运算之后这一位还是 1，如果 $c_{i,j}$ 中这个位置是 0，表示可以打掉这个位置，那么与运算之后结果是 0，符合我们的预期。

位运算优化之后，复杂度变为了 $O\left(n^2 2^n\right)$，还是有点卡，能不能更优呢？考虑集合 s 中最小的点 i，在最优解中它一定会被打掉，我们只需要枚举 i 是如何被打掉的即可（单独被打或者跟别人一起），这样就固定了 i，只需要枚举 j 就行了，复杂度降为 $O\left(n 2^n\right)$，轻松通过所有数据。

代码

```
//    Created by Sengxian on 2016/11/23.
//    Copyright (c) 2016 年  Sengxian. All rights reserved.
#include <bits/stdc++.h>
using namespace std;

const int MAX_N = 18, INF = 0x3f3f3f3f;
int n, m, dp[1 << MAX_N], ss[MAX_N][MAX_N];
double x[MAX_N], y[MAX_N];

inline void cal(int i, int j, double &a, double &b) {
        a = (y[i] - y[j] * (x[i] / x[j])) / (x[i] * x[i] - x[i] * x[j]);
        b = (y[i] - a * x[i] * x[i]) / x[i];
}

inline int dcmp(const double &a, const double &b) {
        return fabs(a - b) < 1e-8 ? 0 : (a < b ? -1 : 1);
}

void process() {
        double a, b;
        for (int i = 0; i < n; ++i)
                for (int j = i + 1; j < n; ++j) {
                        cal(i, j, a, b);
                        if (dcmp(a, 0) != -1) {
                                ss[i][j] = (1 << n) - 1;
                                continue; // a not equal 0
                        }
                        ss[i][j] = 0;
                        for (int k = 0; k < n; ++k)
                                if (dcmp(a * x[k] * x[k] + b * x[k], y[k]) != 0)
                                        ss[i][j] |= (1 << k);
                }
}

inline void tension(int &a, const int &b) {
        if (b < a) a = b;
}

void solve() {
        dp[0] = 0;
        for (register int s = 1, i, j; s < (1 << n); ++s) {
                dp[s] = INF;
```

```
			i = __builtin_ctz(s);
			tension(dp[s], dp[s ^ (1 << i)] + 1);
			for (j = i + 1; j < n; ++j) if ((s >> j) & 1)
				tension(dp[s], dp[s & ss[i][j]] + 1);
		}
		printf("%d\n", dp[(1 << n) - 1]);
}

int main() {
	int caseNum;
	scanf("%d", &caseNum);
	for (int t = 0; t < caseNum; ++t) {
		scanf("%d%d", &n, &m);
		for (int i = 0; i < n; ++i) scanf("%lf%lf", x + i, y + i);
		process();
		solve();
	}
	return 0;
}
```

总结

Day 2 的题不算太难，需要一些小技巧。第一题需要一点点取模的技巧，就容易 A 掉。对于第二题，$q = 0$ 的做法实际上不难想，每次剁的两个 $-q$ 也不难想，如果能够想到将二者结合起来，就会发现 $q \neq 0$ 还是单调的，就能够线性解决。对于第三题，有位运算优化和枚举优化，想到一个，卡一卡常数就应该能 AC，应该算是近年来最简单的第三题吧。

综合来看，相比去年的题目偏重考察知识点，今年的题目在思维难度上有了明显的提升，可以有效的避免"东西学得多，分就高"的情况。在我看来，NOIP 虽然是一个普及形式的比赛，涉及知识点也有限，但并不意味着东西学得多就能 AK，竞赛竞赛，竞的是思维，赛的是心态，而不是仅仅拘泥于学习知识点的多少、学习时间的长短。让所有人都有收获，这可能是我心中比赛应有的样子吧。