A. 小Y的多项式

考点：导数。

题目大意：输出一个 n 次多项式二阶导数每一项的系数。

对于任意一个 n 次多项式：

$$f(x) = a_0 + a_1 x + \cdots + a_n x^n$$

求一次导数：

$$f'(x) = a_1 + 2a_2 x + \cdots + n a_n x^{n-1}$$

求二次导数：

$$f''(x) = 2a_2 + 3 \cdot 4 a_3 x + \cdots + n(n-1) a_n x^{n-2}$$

其中，x^i 的系数为 $(i+1)(i+2)a_{i+2}$。

直接计算并输出即可，时间复杂度为 $O(n)$。注意 $n \le 10^5$，C/C++ 需要开 `long long` 避免数值溢出。

标准程序（C）：

```c
#include <stdio.h>

int main() {
    int n;
    scanf("%d", &n);
    int poly[n + 1];
    for (int i = 0; i <= n; ++i) {
        scanf("%d", &poly[i]);
    }
    for (int i = 0; i <= n - 2; ++i) {
        printf("%lld ", 1LL * (i + 1) * (i + 2) * poly[i + 2]);
    }
    return 0;
}
```

B. 小Y的自动机

考点：字符串。

题目大意：给定 m 个 01 串 S，判断每个 01 串中是否存在连续的 n 个 1。

如果暴力枚举每个位置为首的连续的 1 的数量，最坏情况下时间复杂度为 $O(n\sum|S|)$，无法通过此题。

我们可以考虑在遍历 01 串的同时维护目前连续的 1 的数量 $ones$。具体的，遍历到第 i 个位置时，如果为 1，则 $ones$ 自增 1，否则 $ones$ 重置为 0。当某一时刻 $ones$ 大于等于 n 时，该 01 串合法，否则不合法。时间复杂度为 $O(\sum|S|)$。

标准程序（C）：

```c
#include <stdio.h>

#define MAXN 1000005
char s[MAXN];

int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    while (m--) {
        scanf("%s", s);
        int flag = 0;
        for (int i = 0, ones = 0; s[i] != '\0'; ++i) {
            if (s[i] == '1') ++ones;
            else ones = 0;
            if (ones >= n) {
                flag = 1;
                break;
            }
        }
        if (flag) printf("Yes\n");
        else printf("No\n");
    }
    return 0;
}
```

C. 小Y的猫猫

考点：树、搜索。

题目大意：给定一棵有根二叉树，找到根节点到某一节点之间的路径。

由于题目给出的是有根树，可以直接储存每个节点的父节点，从给定节点不断向父节点转移即可得到到根节点的路径。时间复杂度为 $O(n)$。

当然也可以记录每个节点的左右孩子，通过DFS等算法找到该路径。

标准程序（C）：

```c
#include <stdio.h>

int main() {
    int n, k;
    scanf("%d %d", &n, &k);
    int father[n + 1];
    int ans[n + 1], idx;
    for (int i = 1; i <= n; ++i) {
        father[i] = 0;
    }
    for (int i = 1; i <= n; ++i) {
        int l, r;
        scanf("%d %d", &l, &r);
        if (l) father[l] = i;
        if (r) father[r] = i;
    }
    while (k) {
        ans[idx++] = k;
        k = father[k];
    }
    for (int i = idx - 1; i > -idx; --i) {
        printf("%d ", ans[abs(i)]);
    }
    return 0;
}
```

D. 小Y的组合数学

考点：组合数学、卡特兰数。

题目大意：对一正整数 x 每次 $+1$ 或 -1（但不能减为负数），求 k 次操作恰好能使其变为 0 的方法数。

先不考虑“不能减为负数”的限制。此时 k 次操作中，设有 a 次 $+1$ 操作，则必定有 $x+a$ 次 -1 操作。此时有方程 $a+x+a=k$，解得 $a=\frac{k-x}{2}$，并注意到 $k-x$ 为奇数，或 $k<x$ 时无解。此时总方法数相当于 $\frac{k-x}{2}$ 次 $+1$ 操作和 $\frac{k+x}{2}$ 次 -1 操作的组合数，即 $\binom{k}{\frac{k-x}{2}}$。

此时再考虑添加“不能减为负数”的限制。我们将 x 比作数轴上一点 P，$+1$ 或 -1 的操作比作将 P 向右或向左移动一个单位长度，而“不能减为负数”的条件则转化 “P 不能经过数轴上 -1 的位置”。我们将 P 从初始位置移动到零点的一种方式记为一条“路径”，如果 P 始终没有到达 -1，我们称这条路径是“合法”的，否则，这条路径是“非法”的。于是，原问题就转化成了求合法路径的数量。

显然，合法路径数 $=$ 总路径数 $-$ 非法路径数，因此我们考虑如何计算非法路径数。对于一条非法路径，必定一次或多次经过 -1。我们记它第一次到达 -1 的操作次数为 t，并将第 $t+1,t+2,\cdots,k$ 次操作“翻转”，即向左移动变为向右移动，向右移动变为向左移动。显然，在进行剩余操作后，P 一定到达 -2 的位置。可以证明，非法路径和到达 -2 的路径之间存在一一对应关系，因此其数量一定相等。由之前的结论易得到达 -2 的路径数为 $\binom{k}{\frac{k-(x+2)}{2}}$，从而求得合法路径数为 $\binom{k}{\frac{k-x}{2}}-\binom{k}{\frac{k-(x+2)}{2}}$。

编程计算即可。时间复杂度为 $O(k)$。

标准程序（C）：

```c
#include <stdio.h>
#define MAXN 1000005
const int P = 998244353;

int fac[MAXN];

int inv(int x) {
    if (x == 1) return 1;
    return 1LL * (P - P / x) * inv(P % x) % P;
}

int divide(int a, int b) {
    return 1LL * a * inv(b) % P;
}

int comb(int n, int m) {
    if (n < m || m < 0) return 0;
    return divide(divide(fac[n], fac[m]), fac[n - m]);
}

int main() {
    int x, k;
    scanf("%d %d", &x, &k);
    if ((k - x) % 2) {
        printf("0\n");
        return 0;
    }
    fac[0] = 1;
    for (int i = 1; i <= k; ++i) {
        fac[i] = 1LL * fac[i - 1] * i % P;
    }
    int ans = (comb(k, (k - x) / 2) - comb(k, (k - x - 2) / 2) + P) % P;
    printf("%d\n", ans);
    return 0;
}
```

E. 小Y的糖果

考点：组合数学、容斥原理。

题目大意：将 n 颗糖果放入 m 个有编号的盒子中，每个盒子中的糖果不超过 k 个，问有多少种方法。

记每个盒子的糖果数量为 $a_1, a_2, \cdots, a_m$，则问题转化为求 $\sum_{i=1}^{m} a_i = n \ (a_i \leq m)$ 的非负整数解的数量。

考虑容斥原理，记不添加任何 $a_i > m$ 限制时非负整数解的数量为 $|X|$，只有 $a_k > m$ 时非负整数解的数量为 $|X_k|$，则答案可表示为：

$$|X| - \sum_{1 \leq i \leq m} |X_i| + \sum_{1 \leq i < j \leq m} |X_i \cap X_j| + \cdots + (-1)^m \left| \bigcap_{1 \leq i \leq m} X_i \right|$$

第 k 项表示任取 k 个 $a_{i_k} > m$ 时非负整数解的数量：

$$(-1)^k \sum_{1 \leq i_1 < i_2 < \cdots < i_k \leq m} |X_{i_1} \cap X_{i_1} \cap X_{i_2} \cap \cdots \cap X_{i_k}| = (-1)^k \binom{m}{k} \binom{n + m - 1 - k(m+1)}{m-1}$$

因此答案可整理为以下形式：

$$\sum_{k=0}^{m} (-1)^k \binom{m}{k} \binom{n + m - 1 - k(m+1)}{m-1}$$

编程计算即可。时间复杂度为 $O(n+m)$。

标准程序（C）：

```cpp
#include <stdio.h>
#define MAXN 2000005

const int P = 998244353;

int fac[MAXN], inv[MAXN], fac_inv[MAXN];

void init_inv(int n) {
    inv[1] = 1;
    for (int i = 2; i <= n; i++) {
        inv[i] = 1LL * (P - P / i) * inv[P % i] % P;
    }
}

int comb(long long n, int m) {
    if (n < m || m < 0) return 0;
    return 1LL * fac[n] * fac_inv[m] % P * fac_inv[n - m] % P;
}

int main() {
    int n, m, k;
    scanf("%d %d %d", &n, &m, &k);
    init_inv(n + m - 1);
    fac_inv[0] = fac[0] = 1;
    for (int i = 1; i <= n + m - 1; ++i) {
        fac[i] = 1LL * fac[i - 1] * i % P;
        fac_inv[i] = 1LL * fac_inv[i - 1] * inv[i] % P;
    }
    long long ans = 0;
    for (int i = 0; i <= m; ++i) {
        int t = 1LL * comb(m, i) * comb(n - 1LL * i * (k + 1) + m - 1, m - 1) % P;
        if (i % 2 == 0) ans += t;
        else ans -= t;
        ans = (ans + P) % P;
    }
    printf("%lld\n", ans);
    return 0;
}
```