2023 年牛客多校第十场题解

出题人：上海交通大学

1 C Multiplication

题意：定义 k-shift 数是满足 $k\overline{xy} = \overline{yx}$ 的数字。给定 k，求最大不超过 n 的 k-shift 数。$1 \le n \le 10^{100}$，$2 \le k \le 9$。

解法：设 $f(x)$ 表示 x 十进制下的位数。考虑 k-shift 数字的定义，假设这个数字的高位为 x，低位为 y，乘以 k 后发生了交换，则有：

$$k(x10^{f(y)} + y) = y10^{f(x)} + x$$

考虑分离 x, y:

$$(k \times 10^{f(y)} - 1)x = (10^{f(x)} - k)y$$

即

$$\frac{x}{y} = \frac{10^{f(x)} - k}{k \times 10^{f(y)} - 1}$$

这时考虑到数字位数不大，可以枚举 $f(x), f(y)$，则右边确定。考虑计算它的 gcd 并进行分式约简，得 $\frac{10^{f(x)} - k}{k \times 10^{f(y)} - 1} = \frac{p}{q}$，这时真正的 $\frac{x}{y}$ 应为 $\frac{kp}{kq}$。注意 x, y 由数字长度 $f(x), f(y)$ 和上界的约束，可以得到 k 的数字范围。这时就可以快速统计出当前状态下最大的 k 即可。

```python
def gcd(a, b):
    return a if b == 0 else gcd(b, a % b)

n, k, ans = int(input()), int(input()), 0
L = len(str(n))
p = [10**i for i in range(L + 1)]
for i in range(1, L):
    for j in range(1, L - i + 1):
        x, y = 10**i - k, k * 10**j - 1
        z = gcd(x, y)
        x, y = x // z, y // z
        l = max((p[i - 1] + x - 1) // x, (p[j - 1] + y - 1) // y)
        r = min((p[i] - 1) // x, (p[j] - 1) // y, n // (x * p[j] + y))
        if l > r:
            continue
        ans = max(ans, r * (x * p[j] + y))
print(ans)

```

2 D Agnej

题意：有个 $n+1$ 层的木塔，每层原本由 m 根木头构成。现在除了最顶层是满的 m 根木头，其余每层有些木头已经被抽走了。先后手轮流从非顶层抽走一根木头，如果一层中最左侧或最右侧 $\left\lceil \frac{m}{2} \right\rceil$ 根木头都被抽走，则木塔倒塌。谁让木塔倒塌谁输，问胜负。$1 \le n \times m \le 10^5$。

解法：显然每层独立，因而考虑计算每层答案的 SG 值再异或起来。SG 值的求法是枚举所有的可行后继状态，然后求这些后继状态 v 的 SG 值，当前状态 u 的 SG 值为 $\text{mex}\{\text{SG}(v)\}$。

首先考虑 m 为偶数的情况。定义 (l,r) 表示左边有 l 根木头右边有 r 根木头的状态。这时不存在中间的支撑，两边中如果有一半如果抽完就会倒。显然边界情况 $(1,1)$ 即左右各有一个木头显然是先手必败的，即 $\text{SG}(1,1)=0$，而且唯一不能操作的状态仅此一个。考虑面对总共偶数根木头时，即 $l+r$ 为偶数。先手无论抽什么，后手都面对的是奇数根木头，一定不会是必败态。因而 $l+r$ 为奇数时 $\text{SG}=1$，反之为 0。因而 $\text{SG}(l,r)=(l+r) \bmod 2$。

m 为奇数时，中间可能存在一根木头。设此时状态为 (l,m,r) 表示左、中、右的木头个数，其中 $m \in [0,1]$。如果正中间不存在木头，则退化到偶数情况。如果中间存在木头，再考虑两类情况：

1. 单侧无木头，即形如 $(0,1,r)$ 状态。此时只能依次抽取右侧的木头，$\text{SG}(0,1,r)=r \bmod 2$。
2. 形如 $(1,1,r)$ 状态。则它可以转移到的状态有 $(1,1,r-1)$，$(0,1,r)$，$(1,0,r)$。后两种显然一定是 $r \bmod 2$ 和 $(r+1) \bmod 2$ 即一个为 0 一个为 1，因而当前状态的 mex 至少为 2。考虑从 $r=1$ 的边界条件进行讨论（$\text{SG}(1,1,1)=2$），不难得到 $\text{SG}(1,1,r)=2+(r+1) \bmod 2$。
3. 一般状态 $(l,1,r)$。考虑它的边界 $(2,1,r)$，有后继状态 $(1,1,r)$，$(2,0,r)$，$(2,1,r-1)$。不难注意到第一个状态的 SG 大于等于 2，而第二个状态的 SG 值为 0 或 1。这时考虑对 r 做归纳，当 $r=1$ 时 $(2,1,1)$ 的 SG 为 3，因而 $(2,1,2)$ 的 SG 为 1，恰好与 $(2,0,3)$ 的 SG 相同。因而不难归纳得到 $(2,1,r-1)$ 的 SG 始终与 $(2,0,r)$ 相同，因而 $(2,1,r)$ 的 SG 值只会是 $0,1$ 交替。这时再对 l 归纳可以得到 $\text{SG}(l,1,r)=(l+1+r) \bmod 2$。

```cpp
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = a, i##_ = b; i <= i##_; ++i)
#define fd(i, a, b) for (int i = a, i##_ = b; i >= i##_; --i)

using namespace std;
using ll = long long;
const int N = 1e5 + 5;
int n, m; char s[N];
int sg(int a, int b, int c) {
    if (a > b) swap(a, b);
    if (!c)
        return (a + b) & 1;
    else {
        switch (a) {
        case 0: return b & 1;
        case 1: return b & 1 ? 2 : 3;
        default: return (a + b + 1) & 1;
        }
    }
}
void Solve() {
```

```
    scanf("%d%d", &n, &m);
    int f = 0, k = m & 1 ? m / 2 + 1 : 0;
    fp(i, 1, n) {
        scanf("%s", s + 1);
        int a = 0, b = 0, c = s[k] == '1';
        fp(j, 1, m / 2) a += s[j] == '1';
        fp(j, 1, m / 2) b += s[m - j + 1] == '1';
        f ^= sg(a, b, c);
    }
    puts(f ? "Alice" : "Bob");
}
int main() {
    int t = 1;
    while (t--) Solve();
    return 0;
}
```

3 F IUPC

题意：一场时长为 t 分钟的比赛有 n 道题，每个题有最早过题时间的约束，并且一分钟内至多只能交一题，任意 k 分钟内只能至多交 m 题，问比赛 AK 的方案数。$1 \le t \le 300$，$1 \le n \le 13$，$1 \le m \le k \le 10$。

解法：由于要维护前 k 时刻做了不超过 m 题，并且 k 很小，因而比较简单的维护方法就是维护前 k 时刻哪些时刻做了题。

考虑维护 $f(i, j, S)$ 表示前 i 分钟已经通过 j 题，最后 k 分钟内过题时刻状态为 S 的方案数。S 状态中将第 i 分钟的过题状态放在第 k 位，而第 $i-k+1$ 分钟的状态放在最低位。这样当 $i \leftarrow i+1$ 时 S 只需要先右移一位，然后再在最高位上加入当前这一新的分钟是不是过了题的状态即可。

那么当前是否过题有两种情况：

1. 当前时刻不过题。之前所有状态直接原封不动的移动到现在的状态即可。
2. 当前时刻过题。那么首先新状态需要满足 $|S| \le k$，再考虑当前这一分钟可以过（a）已经有人过的题（b）已经过了的题不能再交。因而对于 $f_{i,j,S}$，可行的新过题种类数为 $a_i - (j-1)$，此处 a_i 表示 i 时刻以前已经有人过的题目数量。之所以不用考虑题目的顺序是因为当目前已经到达 i 时刻，之前所有过的题就都可以纳入考虑，可以将很早就有人过的题憋到现在才交。唯一对选法有影响的只有已经过的题目数量。

因而根据上面两种情况进行转移即可。复杂度 $\mathcal{O}(nt2^k)$。

4 H Differential Equation

题意：定义 $F_0(x) = x$，$F_i(x) = (x^2-1)F'_{i-1}(x)$。给定 x_0 和 n，求 $F_n(x_0) \bmod 998\,244\,353$。$1 \le x_0 < 998\,244\,353$，$0 \le n \le 2 \times 10^5$。

解法：

法一：首先特判 $n = 0$ 的答案为 x_0。下面都考虑 $n \ge 1$ 的情况。

首先由 $F_n(x)=(x^2-1)F'_{n-1}(x)$ 考虑莱布尼兹公式（高阶导数公式），注意到 x^2-1 在三阶导以上为 0，因而当 $k\geq 2$ 时有：

$$F_n^{(k)}(x_0)=k(k-1)F_{n-1}^{(k-1)}(x_0)+2x_0kF_{n-1}^{(k)}(x_0)+(x_0^2-1)F_{n-1}^{(k+1)}(x_0)$$

一个最朴素的想法是，代求式 $F_n^{(0)}(x_0)$ 需要 $F_{n-1}^{(1)}(x_0)$（注意此时的边界条件），然后 $F_{n-1}^{(1)}(x_0)$ 又需要 $F_{n-2}^{(2)}(x_0)$ 等若干阶导数，依次类推。因而我们需要 $F_i^{(j)}(x_0)$ 即任意一个函数的任意阶导数。这是因为上式中 k 阶导推导出了 $k+1$ 阶导，一项推出了三项。而直接递推显然是不太可行的。

这时由于目标函数 $F_n(x)$ 必然为一多项式函数（更加具体的，一定是一个 $n+1$ 次多项式），因而一定可以由泰勒展开得到：

$$F_n(x_0)=\sum_{i=0}^{n+1}\frac{F_n^{(i)}(x_1)}{i!}(x_0-x_1)^i$$

因而这提示我们可以考虑不用求 x_0 处的任意多阶导数，而转而求一些特殊点的导数。观察原递推式，不难发现当递推式中的 $x_0=1$ 时，一项可以只递推出两项。因而考虑这个特殊情况，即我们可以令 $x_1=1$，那么原式转化为

$$F_n(x_0)=\sum_{i=0}^{n+1}\frac{F_n^{(i)}(1)}{i!}(x_0-1)^i$$

而递推式转化为

$$F_n^{(k)}(1)=k(k-1)F_{n-1}^{(k-1)}(1)+2kF_{n-1}^{(k)}(1)$$

接下来的任务就是考虑上述递推式。

首先考虑边界条件，有 $F_0(1)=F'_0(1)=1$。这时不难发现，如果令 $a_{i,j}$ 表示 $F_i^{(j)}(1)$，将 $\{a\}_{(i,j)=(1,1)}^{(n,n+1)}$ 排列成一个二维矩阵，那么 $a_{i,j}$ 仅可以从 $a_{i-1,j}$（向下传递）和 $a_{i-1,j-1}$（右下传递）处转移。那么这时考虑 $F_n^{(k)}(1)$ 项，如果要从 $F_0(1)$ 和 $F'_0(1)$ 开始计算贡献，那么必然是 $F'_0(1)$ 进行了 k 次右下传递，然后再进行了 $n-k+1$ 次向下传递得到的。注意 $F_0(1)$ 只要经过向下传递就会乘以 $k=0$ 而直接消失，因而它不会产生贡献。

再次观察递推式，不难发现如果进行了一次右下传递（即从 $F_{n-1}^{(k-1)}(1)$ 处转移）时，一定会乘以 $k(k-1)$。因而对于得到的 $F_n^{(k)}(1)$，考虑它的任意一条转移路径，则它必然乘有系数 $k!(k-1)!$。

还是只考虑其中一条从 $F'_0(1)$ 开始的转移路径。接下来考虑向下传递过程中的系数变化。由于向下传递可以在任意列（即任意阶导数）处发生，因而它的系数乘以的 $2k$ 是随着导数阶数变化而变化的，但是也仅关于此变化。因而可以考虑 $F_n^k(1)$ 在经过的 $n-k+1$ 次向下转移过程中到底在哪些列（导数阶数）处进行过转移。这个就是经典的生成函数问题：

$$[x^{n-k+1}]\prod_{j=1}^{k}\left(\sum_{i=0}^{+\infty}(2jx)^i\right)$$

即，$\sum\limits_{i=0}^{+\infty}(jx)^i$ 表示 $F_n^k(1)$ 在第 j 列进行的向下转移状态，x^i（$i\in[0,+\infty]$）项系数表示在这里向下转移了 i 次的方案对 $F_n^k(1)$ 的贡献。这里不是指数生成函数是因为如果在第 j 列向下转移次数确定了，则方案是唯一确定的，而这个方案对系数的贡献是只与列数和转移次数有关，为 $(2j)^i$。

不难注意到当 x 充分小的时候，利用泰勒展开有 $\sum\limits_{i=0}^{+\infty}(2jx)^i=\dfrac{1}{1-2jx}$。因而原式转化为（带入右下转移的系数贡献）：

$$k(k-1)![x^{n-k+1}]\prod_{j=1}^{k}\frac{1}{1-2jx}$$

这时对于一个特定的 k，容易发现需要求的系数项也不同。这不利于后续的求和处理。考虑通过同时乘以 x^k 以统一到 x^k 项系数：

$$k!(k-1)![x^{n-k+1}]\prod_{j=1}^{k}\frac{1}{1-2jx}=k!(k-1)![x^{n+1}]\prod_{j=1}^{k}\frac{x}{1-2jx}$$

将上式回代回最开始的泰勒展式：

$$\begin{aligned}&\sum_{k=0}^{n+1}\frac{(x_0-1)^k}{k!}\left(k!(k-1)!\times[y^{n+1}]\prod_{j=1}^{k}\frac{y}{1-2jy}\right)\\=&\sum_{k=0}^{n+1}(k-1)!(x_0-1)^k\left([y^{n+1}]\prod_{j=1}^{k}\frac{y}{(1-2jy)}\right)\\=&\sum_{k=0}^{n+1}\left([y^{n+1}]\prod_{j=1}^{k}\frac{(x_0-1)(j-1)y}{(1-2jy)}\right)\end{aligned}$$

上式中将 $(x_0-1)^k$ 乘入后面的 k 项乘法，$(k-1)!$ 也可类似操作。注意这里当 $j=1$ 时 $j-1=0$ 只是形式上的统一。在后续计算中这里会当作 1 处理。

考虑上式如何快速计算。首先取系数运算和求和运算的运算顺序可以互换。不难发现上式最麻烦的地方在于对于不同的 k，乘除的式子都不尽相同，而且随着 k 变化都需要保留 $n+1$ 项。但是递归树 NTT 可以求解下列式子：

$$h(1,n)=\sum_{i=1}^{n}\left(\prod_{j=1}^{i}f_j(x)\right)\left(\prod_{j=i+1}^{n}g_j(x)\right)$$

其中 f 和 g 的次数和不超过 NTT 计算范围。考虑维护 $h(l,r)$ 的求和答案，通过分治计算出 $h(l,m)$ 和 $h(m+1,r)$，然后考虑合并答案：$h(l,r)=h(l,m)\prod_{j=m+1}^{r}g_j(x)+h(m+1,r)\prod_{j=l}^{m}f_j(x)$，维护 h 和 f,g 的区间乘积即可在 $\mathcal{O}(n\log^2 n)$ 内完成计算。

因而考虑原式如何转化到这一方法进行计算。不难注意到，可以提出分母的 $\prod_{j=1}^{n+1}(1-2jy)$，则原式等于：

$$[y^{n+1}]\sum_{k=0}^{n+1}\frac{\left(\prod_{j=k+1}^{n+1}(1-2jy)\right)\left(\prod_{j=1}^{k}(x_0-1)(j-1)y\right)}{\left(\prod_{j=1}^{n+1}(1-2jy)\right)}$$

因而 $f_j(x)=(x_0-1)(j-1)y$，而 $g_j(x)=1-2jx$，最后计算下面的乘积，同样使用分治的方法进行计算。这里 $f_1(x)$ 注意 $j-1$ 不是 0。最后进行多项式求逆就可以使用上式的方法进行计算。总复杂度 $\mathcal{O}(n\log^2 n)$。

```
// U表示f多项式，F表示g多项式，G表示乘积和多项式。add、mul都是多项式上加法和乘法运算
void sol( int rt , int l , int r ) {
    if( l == r ) {
        U[rt] = { 0 , (int) ( 1ll * ( x0 + P - 1 ) % P * ( l == 1 ? 1 : l - 1 ) % P ) };
        F[rt] = { 1 , (int) ( P - 2ll * l % P ) };
        G[rt] = { 0 , (int) ( 1ll * ( x0 + P - 1 ) % P * ( l == 1 ? 1 : l - 1 ) % P ) };
        return;
    }
    int m = l + r >> 1;
    sol( rt << 1 , l , m );
    sol( rt << 1 | 1 , m + 1 , r );
    vi ul = U[rt << 1] , fr = F[rt << 1 | 1];
    F[rt] = mul( F[rt << 1] , F[rt << 1 | 1] );
    U[rt] = mul( U[rt << 1] , U[rt << 1 | 1] );
```

```
    G[rt] = add( mul( G[rt << 1 | 1] , ul ) , mul( G[rt << 1] , fr ) );
}

void solve() {
    cin >> n >> x0;
    if( !n ) {
        cout << x0 << endl;
        return;
    }
    pw[0]=1;
    for(int i=1;i<=n+1;i++)pw[i]=pw[i-1]*1ll*(x0+P-1)%P;
    fac[0] = 1 , p2[0] = 1;
    rep( i , 1 , n + 1 ) fac[i] = fac[i - 1] * 1ll * i % P , p2[i] = p2[i - 1] * 2 % P;
    ifac[n + 1] = Pow( fac[n + 1] , P - 2 );
    per( i , n + 1 , 1 ) ifac[i - 1] = ifac[i] * 1ll * i % P;
    sol( 1 , 1 , n + 1 );
    vi re = F[1];
    rep( i , 0 , n + 1 ) H[i] = re[i];
    Inv( H , A , n + 2 );
    vi ta( n + 2 ) , tg = G[1];
    rep( i , 0 , n + 1 ) ta[i] = A[i];
    vi ret = mul( ta , tg );
    printf("%d\n",ret[n + 1]);
}
```

法二：考虑使用指数生成函数，具体原因见下方分析。

$$H(z,x)=\sum_{i=0}^{+\infty}\frac{F_i(x)}{i!}z^i$$

不难发现：

$$\frac{\partial H(z,x)}{\partial z}=\sum_{i=1}^{+\infty}\frac{F_i(x)}{(i-1)!}z^{i-1}$$

而上式对 x 求导有：

$$\frac{\partial H(z,x)}{\partial x}=\sum_{i=0}^{+\infty}\frac{F_i'(x)}{i!}z^i$$

注意到刚好求导后发生了系数上的移位，考虑带入 $F_{i+1}(x)=(x^2-1)F_i(x)$ 以消除移位，有：

$$\begin{aligned}(x^2-1)\frac{\partial H(z,x)}{\partial x}&=\sum_{i=0}^{+\infty}\frac{(x^2-1)F_i'(x)}{i!}z^i\\&=\sum_{i=0}^{+\infty}\frac{F_{i+1}(x)}{i!}z^i\\&=\sum_{i=1}^{+\infty}\frac{F_i(x)}{(i-1)!}z^{i-1}\end{aligned}$$

这时就体现出这里为什么需要指数生成函数了——移项的时候发生的系数错位刚好可以通过求导产生的系数和阶乘抵消。如果使用普通生成函数则此时会多一项 i 的系数无法解决。

因而得到

$$\frac{\partial H(z,x)}{\partial z}=(x^2-1)\frac{\partial H(z,x)}{\partial x}$$

考虑使用特征线法，即令 x,y,H 都是关于 t 的函数。则特征方程为

$$\frac{\mathrm{d}z}{\mathrm{d}t}=1,\frac{\mathrm{d}x}{\mathrm{d}t}=1-x^2,\frac{\mathrm{d}H}{\mathrm{d}t}=0$$

由上式 x,z 关于 t 的全微分可得 $\dfrac{\mathrm{d}z}{\mathrm{d}x}=\dfrac{1}{1-x^2}$，可得 $e^{2z}\dfrac{1-x}{1+x}=C$。而由于此时 $\dfrac{\mathrm{d}H}{\mathrm{d}t}=0$，因而最终形式一定为 $F(G(z,x))$，即 $H(z,x)=F\left(e^{2z}\dfrac{1-x}{1+x}\right)$。带入边界条件 $H(0,x)=x$ 得 $H(z,x)=\dfrac{(1+x)-(1-x)e^{2z}}{(1+x)+(1-x)e^{2z}}$。

这是一个二元生成函数，最后考虑恢复 $z=n$ 时 $F_n(x)$，由于仅求一项，因而可以带入 $x=x_0$，则原式变成一个仅关于 z 的形式幂级数，因而多项式求逆即可。复杂度 $\mathcal{O}(n\log n)$。

5 J Pumping

题意：给定一个有向图 $G(n,m)$，边上有字符，定义一条从 1 号节点出发到给定终止节点集合结束的路径是好的。现在要求找到一条路径，其边上字符按顺序可以记作 xyz，且满足 xy^iz 也是好的，$i\in[0,+\infty]$。其中 y^i 表示 y 的 i 次拼接。构造一个这样的 xyz。$1\le n\le 5\times10^5$，$0\le m\le10^6$，终止集合大小 $k\in[1,n]$。

解法：由于要能延申无穷多次，因而必然有环；并且 y 部分可以不存在，也就是 xz 本身也是一条合法路径，因而考虑从起点到终点上必须经过一个点，该点上可以走出一条环路。假设这个环路起点为 i，则从 $1\to i$ 为 x 串，过 i 的环为 y 串，从 i 到任意一个终止集合点的路径为 z。

具体而言，首先在反图上 `bfs` 找出能到终点的点集合，然后在正向图上从 1 出发，经由可以到达终点的点进行 `dfs`。如果在 `dfs` 过程中可以找到一个环即第二次经过某个点 i，则 xy 部分即为 $1\to i$ 的路径。这时只需要再从 i 找到任意一条去终点的路径即可。复杂度 $\mathcal{O}(n+m)$。

```cpp
#include <bits/stdc++.h>
#define fp(i, a, b) for (int i = a, i##_ = b; i <= i##_; ++i)
#define fd(i, a, b) for (int i = a, i##_ = b; i >= i##_; --i)

using namespace std;
using ll = long long;
const int N = 5e5 + 5;
int n, m, k, ok[N], ed[N], vis[N];
string ans, cur;
vector<pair<int, char>> G[N], H[N];
void dfs(int u) {
    vis[u] = 1;
    for (auto [v, c] : G[u]) {
        if (!ok[v]) continue;
        if (vis[v] == 1) {
            if (ans.empty())
                ans = cur, ans += c, k = v;
        } else if (!vis[v]) {
            cur += c;
```

```
            dfs(v);
            cur.pop_back();
        }
    }
    vis[u] = -1;
}
int dfs2(int u) {
    vis[u] = 1;
    if (ed[u]) return 1;
    for (auto [v, c] : G[u]) {
        if (!ok[v] || vis[v]) continue;
        cur += c;
        if (dfs2(v)) return 1;
        cur.pop_back();
    }
    return 0;
}
void Solve() {
    scanf("%d%d%d", &n, &m, &k);
    queue<int> q;
    for (int x; k--;) scanf("%d", &x), q.push(x), ed[x] = ok[x] = 1;
    for (int u, v, c; m--;) {
        scanf("%d%d %c", &u, &v, &c);
        G[u].push_back({v, c});
        H[v].push_back({u, c});
    }
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (auto [v, c] : H[u])
            if (!ok[v]) q.push(v), ok[v] = 1;
    }
    if (!ok[1]) return puts("-1"), void();
    dfs(1);
    if (ans.empty()) return puts("-1"), void();
    fill(vis, vis + n + 1, 0);
    dfs2(k);
    printf("%s\n", (ans + cur).data());
}
int main() {
    int t = 1;
    // scanf("%d", &t);
    while (t--) Solve();
    return 0;
}
```

6 K First Last

题意：n 人参赛 k 次，问每次都取得第一名或最后一名的概率，输出浮点数。$1 \le n, k \le 20$。

解法：输出 $\left(\frac{\min(2,n)}{n}\right)^k$。

7 L Grayscale Confusion

题意：给定 n 种颜色 $\{c\}_{i=1}^n$，由 RGB 三色定义。现要求构造一个函数映射 $f: c \to [0,255]$，使得任意满足严格偏序关系 $c_i < c_j$ 的颜色对有 $f(c_i) < f(c_j)$，并且 $f(c_1) = f(c_2)$，其中颜色的严格偏序定义为三原色的数值都满足严格小于关系，需要报告无解。$1 \le n \le 10^3$。

解法：首先 c_1 和 c_2 不能有严格偏序关系。如果他们确定没有偏序关系了，这时只需要合并这两个点，对满足严格偏序关系的点对建立边，在这个 DAG 上跑一次拓扑排序进行函数映射即可。复杂度 $\mathcal{O}(n^2)$。

```cpp
#include <bits/stdc++.h>
using namespace std;
const int N = 1000;
vector<int> G[N + 5];
int ans[N + 5], deg[N + 5];
struct color
{
    int r, g, b;
    bool operator<(const color &x) const
    {
        return r < x.r && g < x.g && b < x.b;
    }
} q[N + 5];
queue<int> que;
int main()
{
    memset(ans, -1, sizeof(ans));
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        scanf("%d%d%d", &q[i].r, &q[i].g, &q[i].b);
    if (q[0] < q[1] || q[1] < q[0])
    {
        printf("-1");
        return 0;
    }
    for (int i = 2; i < n; i++)
    {
        for (int j = 2; j < n; j++)
            if (q[j] < q[i])
            {
                deg[i]++;
```

```
                G[j].push_back(i);
            }
        if (q[1] < q[i] || q[0] < q[i])
        {
            deg[i]++;
            G[1].push_back(i);
        }
    }
    for (int j = 2; j < n; j++)
        if (q[j] < q[0] || q[j] < q[1])
        {
            G[j].push_back(1);
            deg[1]++;
        }
    for (int i = 1; i < n; i++)
        if (!deg[i])
        {
            que.push(i);
            ans[i] = 0;
        }
    while (!que.empty())
    {
        int tp = que.front();
        que.pop();
        for (auto x : G[tp])
        {
            deg[x]--;
            ans[x] = max(ans[x], ans[tp] + 1);
            if (!deg[x])
                que.push(x);
        }
    }
    ans[0] = ans[1];
    for (int i = 0; i < n; i++)
        if (ans[i] == -1 || ans[i] > 255)
        {
            printf("-1");
            return 0;
        }
    for (int i = 0; i < n; i++)
        printf("%d\n", ans[i]);
    return 0;
}
```

8 M Fair Equation

题意：给定一个算式 $a+b=c$，问能不能通过在这个等式的任意位置至多插入一个数位 $[0,9]$ 使得该等式成立。$1 \leq a,b,c \leq 10^6$。

解法：枚举其中两项，看缺省的那一项能不能由给定项添加一个数字得到。模拟即可。

```cpp
#include <bits/stdc++.h>
using namespace std;
bool change(int x, int y)
{
    if (x == y)
        return true;
    string a = to_string(x), b = to_string(y);
    for (auto it = b.begin(); it != b.end(); it++)
    {
        string c = b;
        b.erase(it);
        if (a == b)
            return true;
        b = c;
    }
    return false;
}
int main()
{
    int a, b, c;
    scanf("%d + %d = %d", &a, &b, &c);
    if (change(a, c - b))
        printf("Yes\n%d + %d = %d\n", c - b, b, c);
    else if (change(b, c - a))
        printf("Yes\n%d + %d = %d\n", a, c - a, c);
    else if (change(c, a + b))
        printf("Yes\n%d + %d = %d\n", a, b, a + b);
    else
        printf("No");
    return 0;
}
```