「学习总结」清北学堂 十连测

Jiayi Su (ShuYuMo)

2020-12-01 09:16:52

济南 清北学堂，NOIP 考前十连测，题目整理。# 清北学堂 10 连测

Round 1

A. yist

给出 n 求 $n! \bmod 2^{32}$。

```
int main(){
    A[0] = 1; rep(i, 1, 50) A[i] = (A[i - 1] * i) % MOD;
    int T = read();
    while(T--){
        int x = read();
        if(x <= 50) printf("%llu\n", A[x]);
        else puts("0");
    }
    return 0;
}
```

B. ernd

有一个 $n \times m$ 的网格，网格上的数字都在 $[1, nm]$ 之间且两两不同。

有个限制，直观描述是位置左、上的格子必须比右、下的格子小，精确地说：$A_{i,j} < A_{i+1,j}, A_{i,j} < A_{i,j+1}$ 必须成立，如果不等式中的两个位置都在网格内的话。

你想知道对于一个 k，有多少个格子的值可以是 k（即存在一种网格，使得它的值是 k）。

考虑每个位置 (i, j) 能填哪些数字，这个位置到 $(1, 1)$ 形成的矩形中的值都要大于 (i, j) 的值，这个位置到 n, m 同理。

```
int ans[_T], C[_T];
void work(int n, int  m){
    memset(C, 0, sizeof(C));
    rep(i, 1, n) rep(j, 1, m){
        int L = i * j, R = n * m - ((n - i + 1) * (m - j + 1) - 1);
        C[L]++; C[R + 1]--;
    }
    int now = 0;
    rep(i, 1, n * m) now += C[i], ans[i] = now;
    int q = read();
    while(q--) printf("%d\n", ans[read()]);
}
```

C. sanrd

小 Y 有一个排列 $P_{1..n}$，作为强迫症患者，她想要把排列排好序。共有 3 种操作。

花费 a 的代价交换相邻两个数。

花费 b 的代价翻转整个排列。

花费 c 的代价打乱整个排列，新排列随机生成。

小 Y 需要知道，在最优策略下，她需要花费多大的代价呢？

最优的操作，显然是先打乱排列（如果有），再翻转排列（如果有），最后交换相邻两个数（如果有）。

几个事实

- $A_i > A_j, i+1=j$ 是交换相邻两个数字的必要条件。
- 执行一次交换操作逆序对数量会减少且仅减少一。
- 答案只与逆序对数量有关。
- 打乱后随机生成的代价就是 c + 长度为 n 的所有排列排好代价的平均值。只要使用最后一种操作，那么他们的答案一定相同。

设排列 $[P]$ 的逆序对数为 x ，如果只使用前两种操作，代价就是 $\min\{\left(\frac{n(n-1)}{2}-x\right)\times a+b, x\times a\}$。把 $[p]$ 的所有排列按照只用前两种操作的花费排好序，那么可能使用最后一种操作的一定是一段后缀。

以 $[p]$ 的所有排列按照只用前两种操作的花费排好序的下标为 x 轴，代价为 y 轴，那么只用前两种操作就是绿色的图像。如果对某些排列使用第三种操作，可以用橙色的图像表示。Round0_C.png 其中被函数值压平的部分就是 c + 所有排列代价的平均值。枚举是多长的后缀被压平（使用第三种操作）。然后解方程解出函数值平均值，选取最小的平均值即可。

如果排列数比较多，无法一个一个的算出来，可以通过 dp 求出长度为 n 的排列，逆序对个数为 k 的排列数。可以注意到，逆序对数相同的排列，答案一定是一样的，可以把逆序对是相同的排列捆绑计算，逆序对数一共有 $\frac{n\times(n-1)}{2}+1$ 种取值。

```
#include <cstdio>
#include <cstring>
#include <cmath>
#include <iostream>
#include <cmath>
#include <algorithm>
#include <climits>
#include <assert.h>
#include <vector>
#define LL long long
#define int long long
#define rep(i, l, r) for(int i = (l), ___ = (r); i <= ___; i++)
#define per(i, l, r) for(int i = (r), ___ = (l); i >= ___; i--)
using namespace std;
struct Read_t{
    template<typename T> const Read_t & operator() ( T & target ) const {
        T x = 0; int sign = 1; char c = getchar();
        while(c < '0' || c > '9') { if(c == '-') sign = -1; c = getchar(); }
        while(c <= '9' && c >= '0') { x = (x << 1) + (x << 3); x += c - '0'; c = getchar(); }
        target = x * sign; return *this;
    }
```

```
} Read;
#warning dont forget enable int_64
const int _N = 500;
const int _S = 500;
LL gcd(LL a, LL b){ return b == 0 ? a : gcd(b,  a % b); }
LL dp[_N][_S];
LL n, a, b, c, d;
#define pb push_back
#define mp make_pair
#define fi first
#define se second
pair<LL, LL> Val[_S];
bool operator > (pair<LL, LL> A, pair<LL, LL> B){ return (A.fi * B.se > A.se * B.fi); }
LL A[_S];
void doit(){
    Read(n)(a)(b)(c)(d); vector<pair<LL, LL> > V; V.clear();
    rep(i, 0, (n) * (n - 1) / 2) V.pb(mp(min(i * a, ((n * (n - 1) / 2) - i) * a + b), dp[n][i]));
    sort(V.begin(), V.end());
    int tot = 0;
    rep(L, 0, V.size() - 1){
        int R = L; while(V[R + 1].fi == V[L].fi && R + 1 <= (int)(V.size()) - 1) R++; LL sum = 0;
        rep(i, L, R) sum += V[i].se;

        V[tot].fi = V[L].fi; V[tot].se = sum;
        tot++;
        L = R;
    } LL S = 0; rep(i, 0, tot - 1) S += V[i].se;
    rep(i, 0, tot - 1){
        LL cnt = 0; rep(j, i, tot - 1) cnt += V[j].se;
        LL A = cnt * c;  rep(j, 0, i - 1) A += V[j].fi * V[j].se;
        LL B = S - cnt;
        LL g = gcd(A, B); Val[i] = mp(A / g, B / g);
    }

    int Mid = 0; rep(i, 0, tot - 1) if(Val[Mid] > Val[i]) Mid = i;
    pair<LL, LL> MAns = Val[Mid]; MAns.fi += MAns.se * c; LL g = gcd(MAns.fi, MAns.se); MAns.fi /= g; M
    while(d--){
        rep(i, 1, n) Read(A[i]);
        LL ans = 0;
        rep(i, 1, n) rep(j, i + 1, n) ans += (A[i] > A[j]);
        ans = min(ans * a, b + a * ((n * (n - 1) / 2) - ans));
        if(ans * MAns.se < MAns.fi) printf("%lld/1\n", ans);
        else printf("%lld/%lld\n", MAns.fi, MAns.se); assert(MAns.fi > 0);  assert(MAns.se > 0);
    }
}
void Init(int n) {
    dp[1][0] = 1;
```

```
    rep(i, 2, n) {
        rep(j, 0, ((i) * (i - 1)) / 2) {
            LL &ans = dp[i][j] = 0;
            rep(k, 0, min(i - 1, j)) ans += dp[i - 1][j - k];
        }
    }
}
signed main(){ freopen("in.txt", "r", stdin);
    Init(17);
    int T; Read(T);
    while(T--) doit();
    return 0;
}
```

```
#include <bits/stdc++.h>
#define fi first
#define se second
#define pb push_back
#define mp make_pair
#define SZ(x) ((int)x.size())
#define ALL(x) x.begin(), x.end()
#define L(i, u) for (register int i = head[u]; i; i = nxt[i])
#define rep(i, a, b) for (register int i = (a); i <= (b); i++)
#define per(i, a, b) for (register int i = (a); i >= (b); i--)
using namespace std;
typedef long double ld;
typedef long long ll;
typedef unsigned int ui;
typedef pair<ll, ll> Pll;
typedef vector<int> Vi;
template <class T>
inline void read(T &x){x = 0;char c = getchar();int f = 1; while (!isdigit(c)) { if (c == '-') f = -1;
template <class T>
T gcd(T a, T b) { return !b ? a : gcd(b, a % b); }
template <class T>
inline void umin(T &x, T y) { x = x < y ? x : y; }
template <class T>
inline void umax(T &x, T y) { x = x > y ? x : y; }
ll dp[1 << 16 | 3][141];
struct Yzr
{
    int n;
    ll gs[141];
    void ini(int nn)
    {
        n = nn;
        memset(dp, 0, sizeof(dp));
```

```
        dp[0][0] = 1;
        rep(s, 0, (1 << n) - 1) rep(i, 1, n) if (~s >> i - 1 & 1)
        {
            int del = 0, cnt = 0;
            rep(j, i + 1, n) del += s >> j - 1 & 1;
            rep(j, 1, n) cnt += s >> j - 1 & 1;
            rep(k, 0, cnt * (cnt + 1) >> 1) dp[s | 1 << i - 1][k + del] += dp[s][k];
        }
        rep(k, 0, n * (n + 1) >> 1) gs[k] = dp[(1 << n) - 1][k];
    }
} yzr[17];
bool cmp(Pll a, Pll b) { return (ld)a.fi / a.se < (ld)b.fi / b.se; }
ll n, a, b, c, d, qz[141], p[166];
Pll s[166];
ll calc(ll x) { return min(x * a, (n * (n - 1) / 2 - x) * a + b); }
int main()
{
    freopen("in.txt", "r", stdin);
    rep(n, 2, 16) yzr[n].ini(n);
    int T;
    read(T);
    while (T--)
    {
        read(n); read(a); read(b); read(c); read(d);
        Pll res(1e18, 1);
        ll tot = 1;
        rep(i, 1, n) tot *= i;
        int len = 0;
        rep(i, 0, n * (n + 1) / 2) if (yzr[n].gs[i]) s[++len] = mp(calc(i), yzr[n].gs[i]);
        sort(s + 1, s + len + 1);
        static ll qz[166], hz[166];
        rep(i, 1, len) qz[i] = qz[i - 1] + s[i].fi * s[i].se;
        hz[len + 1] = 0;
        per(i, len, 1) hz[i] = hz[i + 1] + s[i].se;
        rep(i, 1, len)
        {
            Pll cur = mp(qz[i] + hz[i + 1] * c, tot - hz[i + 1]);
            if (cmp(cur, res))
                res = cur;
        }
        pair<ll, ll> ans = mp(res.fi + c * res.se, res.se);
        cerr << "N = " << n << endl;
        cout << ans.fi << " " << ans.se << endl;
        while (d--)
        {
            rep(i, 1, n) read(p[i]);
            int nx = 0;
```

```
            rep(i, 1, n) rep(j, i + 1, n) nx += p[i] > p[j];
            Pll ans = mp(calc(nx), 1);
            if (cmp(mp(res.fi + c * res.se, res.se), ans))
                ans = mp(res.fi + c * res.se, res.se);
            ll g = gcd(ans.fi, ans.se);
            ans.fi /= g;
            ans.se /= g;
            printf("%lld/%lld\n", ans.fi, ans.se);
        }
    }
    return 0;
}
```

D. sith

你有 k 棵点数均为 n 的树

对于每对点 i,j，你都需要求出，有多少个点 x，满足在所有树中都在 $i \leftrightarrow j$ 的树链上（树链包含端点即 i,j）。

定义函数 $\mathrm{dis}_t(x,y)$ 为 x，y 这两个结点在树 t 上的最短距离（简单路径长度）。

对于一棵树 t ，“$\mathrm{dis}_t(i,j)=\mathrm{dis}_t(i,x)+\mathrm{dis}_t(x,j)$” 为 “点 x 在 点 $i \leftrightarrow$ 点 j 的树链上” 的充要条件。

扩展一下可以发现

对于森林 T，“$\sum\limits_{t\in T}\mathrm{dis}_t(i,j)=\sum\limits_{t\in T}\mathrm{dis}_t(i,x)+\sum\limits_{t\in T}\mathrm{dis}_t(x,j)$” 为 “对于每棵树 点 x 在 点 $i \leftrightarrow$ 点 j 的树链上” 的充要条件。

这样就可以分开算了，随便做就好了。

```
#include <bits/stdc++.h>
#define fi first
#define se second
#define pb push_back
#define mp make_pair
#define SZ(x) ((int)x.size())
#define ALL(x) x.begin(), x.end()
#define L(i, u) for (register int i = head[u]; i; i = nxt[i])
#define rep(i, a, b) for (register int i = (a); i <= (b); i++)
#define per(i, a, b) for (register int i = (a); i >= (b); i--)
using namespace std;
typedef long double ld;
typedef long long ll;
typedef unsigned int ui;
typedef pair<int, int> Pii;
typedef vector<int> Vi;
template <class T>
inline void read(T &x){x = 0;char c = getchar();int f = 1;while (!isdigit(c)){if (c == '-')f = -1;c = g
template <class T>
T gcd(T a, T b) { return !b ? a : gcd(b, a % b); }
template <class T>
inline void umin(T &x, T y) { x = x < y ? x : y; }
```

```cpp
template <class T>
inline void umax(T &x, T y) { x = x > y ? x : y; }
template <class T>
inline T Abs(const T &x) { return x > 0 ? x : -x; }
inline ui R()
{
    static ui seed = 613;
    return seed ^= seed >> 5, seed ^= seed << 17, seed ^= seed >> 13;
}
const int N = 505;
int n, k, dis[N][N];
Vi e[N];
void dfs(int s, int u, int fa, int dep)
{
    dis[s][u] += dep;
    for (int v : e[u])
        if (v != fa)
            dfs(s, v, u, dep + 1);
}
int main()
{
    read(n);read(k);
    rep(tt, 1, k){
        rep(i, 1, n) e[i].clear();
        rep(i, 1, n - 1){
            int u, v;read(u);read(v);
            e[u].pb(v);e[v].pb(u);
        }
        rep(i, 1, n) dfs(i, i, 0, 0);
    }
    rep(i, 1, n) rep(j, 1, n)
    {
        int ans = 0;
        rep(x, 1, n) ans += dis[i][x] + dis[j][x] == dis[i][j];
        printf("%d%c", ans, j < n ? ' ' : '\n');
    }
    return 0;
}
```

Round 2

A. one

给你一个长度为 n 的数组，你需要将其划分为若干个连续段。对于一种划分，定义其权值为，求出每段的段内所有元素 xor 值，再把所有段的 xor 值相加即为权值。

你需要计算对于所有划分，这个权值的最小、最大值分别是多少。$n \leq 10^6$

$A \operatorname{xor} B \leq A + B$

$A \ \ \& \ \ B \le A \mid B \le A + B$

全部拆开最大，全部合并最小。

```
int main(){
    int n, MIN = 0; LL MAX = 0; Read(n);
    rep(i, 1, n) { int x; Read(x); MIN ^= x; MAX += x; }
    printf("%d %lld", MIN, MAX);
    return 0;
}
```

B. two

有一个 $1..n$ 依次连成的环，有一个从 1 开始移动的指针，每次指针所在位置有 p 的概率消失并将这个位置对应的下标（在 $1..n$ 中）插入序列 B 的末尾，然后指针移动一格（1 移到 2，n 移到 1 这样，一个位置若已经消失则不会被移动到）。所有位置都消失时游戏结束。最后 B 会是一个排列。

这道题跟序列 B 没什么关系，你只需要求出游戏期望进行几轮，答案对 998244353 取模。

读入描述：一行三个整数 n, x, y。概率 $p = \frac{x}{y}$

设 $f[n]$ 为 长度为 n 还需要进行多少轮。

易知 $f[n] = 1 + p \times f[n-1] + (1-p) \times f[n]$

移项得 $f[n] = n \times \frac{1}{p}$

```
int main(){
    Read(n)(x)(y);
    printf("%d", int(n *1ll* y % MOD *1ll* inv(x, MOD) % MOD));
    return 0;
}
```

C. three

小 Y 拥有一个序列 a_i（从 0 开始标号）。

小 Y 想要对序列进行 Q 次操作，操作有下面几种：

- $t = 0$：对区间 $[l, r]$ 执行 $a_i = a_i + x$
- $t = 1$：对区间 $[l, r]$ 执行 $a_i = \lfloor \frac{a_i}{x} \rfloor$
- $t = 2$：查询区间 $[l, r]$ 的 $\max a_i$
- $t = 3$：把区间 $[l, r]$ 的 a_i 恢复为初始给出的 a_i $a_i \le 10^8; n, Q \le 10^5$

定义一类特殊标记　记录标记 a, b, c 表示 x 变成 $\lfloor \frac{x+a}{b} \rfloor + c$。其中要求 $a < b$。- 对于加法操作 $c+ = x$ 即可。- 对于除法操作如果是除 d ：$\lfloor \frac{\lfloor \frac{x+a}{b} \rfloor + c}{d} \rfloor = \lfloor \frac{\lfloor \frac{x+a+bc}{b} \rfloor}{d} \rfloor = \lfloor \frac{x+a+bc}{bd} \rfloor$ 然后优化一下 $x + a + bc$ 保持 $a' < b'$，类似于假分数换算带分数，保证不会爆 long long。

对于处理除数过大的情况：

```
if (fm > inf) fz = max(0ll, fz + (inf - fm)), fm = inf;
```

如果分母 $> \inf$，那么 $\lfloor \frac{x+a}{b} \rfloor + c$ 等价于 $[x \ge b - a] + c$ 当分母很大时，可以使得 a, b 同时减去一个常数使得 b 不会溢出，对于 $[x \ge b - a] + c$ 不影响取值。

```
#include <bits/stdc++.h>
#define fi first
```

```cpp
#define se second
#define pb push_back
#define mp make_pair
#define SZ(x) ((int)x.size())
#define ALL(x) x.begin(), x.end()
#define L(i, u) for (register int i = head[u]; i; i = nxt[i])
#define rep(i, a, b) for (register int i = (a); i <= (b); i++)
#define per(i, a, b) for (register int i = (a); i >= (b); i--)
using namespace std;
typedef long double ld;
typedef long long ll;
typedef unsigned int ui;
typedef pair<int, int> Pii;
typedef vector<int> Vi;
template <class T>
T gcd(T a, T b) { return !b ? a : gcd(b, a % b); }
template <class T>
inline void umin(T &x, T y) { x = x < y ? x : y; }
template <class T>
inline void umax(T &x, T y) { x = x > y ? x : y; }
inline ui R()
{
    static ui seed = 416;
    return seed ^= seed >> 5, seed ^= seed << 17, seed ^= seed >> 13;
}
const int N = 233333, inf = 2e9 + 1e8;
struct node
{
    ll ans;
    bool emp;
    ll a, b, c, mx; //(x+a)/b+c (a<b<=inf)
    node() { b = 1; }
} tree[N << 2];
inline void pushadd(int k, ll x) {
    tree[k].ans += x; tree[k].c += x;
}
inline void pushdiv(int k, int d) {
    tree[k].ans /= d;
    ll fm = 1ll * tree[k].b * d, tmp = (tree[k].a + 1ll * tree[k].b * tree[k].c) / fm;
    ll fz = (tree[k].a + 1ll * tree[k].b * tree[k].c) - fm * tmp;
    tree[k].c = tmp;
    if (fm > inf) fz = max(0ll, fz + (inf - fm)), fm = inf;
    tree[k].a = fz, tree[k].b = fm;
    //  printf("%d:%lld %lld %lld\n",k,tree[k].a,tree[k].b,tree[k].c);
}
inline void pushemp(int k) {
    tree[k].emp = 1;
```

```
    tree[k].a = tree[k].c = 0;
    tree[k].b = 1;
    tree[k].ans = tree[k].mx;
}
inline void pushdown(int k) {
    if (tree[k].emp) pushemp(k << 1), pushemp(k << 1 | 1), tree[k].emp = 0;
    if (tree[k].a) pushadd(k << 1, tree[k].a), pushadd(k << 1 | 1, tree[k].a), tree[k].a = 0;
    if (tree[k].b != 1) pushdiv(k << 1, tree[k].b), pushdiv(k << 1 | 1, tree[k].b), tree[k].b = 1;
    if (tree[k].c) pushadd(k << 1, tree[k].c), pushadd(k << 1 | 1, tree[k].c), tree[k].c = 0;
}
inline void upd(int k){ tree[k].ans = max(tree[k << 1].ans, tree[k << 1 | 1].ans); }
void mdy1(int k, int l, int r, int x, int L, int R) {
    if (l == L && r == R) { pushadd(k, x); return; }
    int mid = (L + R) >> 1;
    pushdown(k);
    if (r <= mid) mdy1(k << 1, l, r, x, L, mid);
    else if (l > mid) mdy1(k << 1 | 1, l, r, x, mid + 1, R);
    else mdy1(k << 1, l, mid, x, L, mid), mdy1(k << 1 | 1, mid + 1, r, x, mid + 1, R);
    upd(k);
}
void mdy2(int k, int l, int r, int x, int L, int R) {
    if (l == L && r == R) { pushdiv(k, x); return; }
    int mid = (L + R) >> 1;
    pushdown(k);
    if (r <= mid) mdy2(k << 1, l, r, x, L, mid);
    else if (l > mid) mdy2(k << 1 | 1, l, r, x, mid + 1, R);
    else mdy2(k << 1, l, mid, x, L, mid), mdy2(k << 1 | 1, mid + 1, r, x, mid + 1, R);
    upd(k);
}
void mdy3(int k, int l, int r, int x, int L, int R) {
    if (l == L && r == R) { pushemp(k); return; }
    int mid = (L + R) >> 1;
    pushdown(k);
    if (r <= mid) mdy3(k << 1, l, r, x, L, mid);
    else if (l > mid) mdy3(k << 1 | 1, l, r, x, mid + 1, R);
    else mdy3(k << 1, l, mid, x, L, mid), mdy3(k << 1 | 1, mid + 1, r, x, mid + 1, R);
    upd(k);
}
int qry(int k, int l, int r, int x, int L, int R)
{
    if (l == L && r == R) return tree[k].ans;
    int mid = (L + R) >> 1;
    pushdown(k);
    if (r <= mid)
        return qry(k << 1, l, r, x, L, mid);
    else if (l > mid)
        return qry(k << 1 | 1, l, r, x, mid + 1, R);
```

```
    return max(qry(k << 1, l, mid, x, L, mid), qry(k << 1 | 1, mid + 1, r, x, mid + 1, R));
}
int n, q, a[N];
void build(int k, int l, int r)
{
    if (l == r) { tree[k].mx = tree[k].ans = a[l]; return; }
    int mid = (l + r) >> 1;
    build(k << 1, l, mid);
    build(k << 1 | 1, mid + 1, r);
    tree[k].mx = tree[k].ans = max(tree[k << 1].mx, tree[k << 1 | 1].mx);
}
int main()
{
    read(n); read(q); rep(i, 0, n - 1) read(a[i]);
    build(1, 0, n - 1);
    while (q--)
    {
        int op, l, r, x;
        read(op); read(l); read(r); read(x);
        if (op == 0) mdy1(1, l, r, x, 0, n - 1);
        else if (op == 1) mdy2(1, l, r, x, 0, n - 1);
        else if (op == 2) printf("%d\n", qry(1, l, r, 0, 0, n - 1));
        else mdy3(1, l, r, x, 0, n - 1);
    }
    return 0;
}
```

浮点数运算控制精度　直接全部转化成浮点数运算，三标记线段树。

考虑到浮点数自带精度损失，对于较大的除数，自动使得原数变成零，long double 的精度可以支持。

爆零小技巧：ST 表空间开一倍

```
int ST[_][LOG + 3], Log[_];
void init_query(){
    rep(i, 1, n) ST[i][0] = A[i];
    rep(j, 1, LOG) rep(i, 1, n) ST[i][j] = max(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]); // ST[i +
    Log[1] = 0; rep(i, 2, n) Log[i] = Log[i >> 1] + 1;
}
```

```
const int _ = 2e5 + 100;
int n, q; LL A[_];
namespace Acceptable_Solution{
    const int _ = 2e5 + 100;
    const int LOG = 17;
    int ST[_][LOG + 3], Log[_];
    void init_query(){
        rep(i, 1, n) ST[i][0] = A[i];
        rep(j, 1, LOG) rep(i, 1, n) ST[i][j] = max(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);
```

```
    Log[1] = 0; rep(i, 2, n) Log[i] = Log[i >> 1] + 1;
}
int Query_Max(int L, int R) { int Lg = Log[R - L + 1]; int Len = (1 << Lg); return max(ST[L][Lg], S
#define LB long double
LB eps;
void print(LB x) { printf("%.20Lf\n", x); }
namespace SegmentTree{
    bool same(LB x, LB y) { return fabs(x - y) <= 1e-12; }
    const int _ = 6e5 + 100;
    int ch[_][2], tot = 0;
    LB v[_], tag_add[_], tag_mul[_];
    bool tag_ret[_];
    #define ls(o) (ch[o][0])
    #define rs(o) (ch[o][1])
    #define maintain(o) (v[o] = max(v[ls(o)], v[rs(o)]))
    #define make (tot++, ch[tot][0] = ch[tot][1] = 0, tag_add[tot] = 0, tag_mul[tot] = 1, tag_ret[t
    int Groot(){ return make; }
    void build(int o, int L, int R){
        if(L == R) return (void)(v[o] = A[L]);
        int mid = (L + R) >> 1; ls(o) = make; rs(o) = make;
        build(ls(o), L, mid); build(rs(o), mid + 1, R);
        maintain(o);
    }
    void tar_ret(int o, int L, int R){
        v[o] = Query_Max(L, R);
        tag_add[o] = 0; tag_mul[o] = 1;
        tag_ret[o] = 1;
    }
    void push_ret(int o, int L, int R){
        if(!tag_ret[o]) return ;
        int mid = (L + R) >> 1;
        tar_ret(ls(o), L, mid); tar_ret(rs(o), mid + 1, R);
        tag_ret[o] = 0;
    }
    void tar_add(int o, LB Val, int L, int R){
        push_ret(o, L, R);
        v[o] += Val;
        tag_add[o] += Val;
    }
    void tar_mul(int o, LB Val, int L, int R) {
        push_ret(o, L, R);
        v[o] *= Val;
        tag_mul[o] *= Val;
        tag_add[o] *= Val;
    }
    void push(int o, int L, int R){
        push_ret(o, L, R);
```

```
        int mid = (L + R) >> 1;
        if(!same(tag_mul[o], 1)) {
            tar_mul(ls(o), tag_mul[o], L, mid);
            tar_mul(rs(o), tag_mul[o], mid + 1, R);
            tag_mul[o] = 1;
        }
        if(!same(tag_add[o], 0)){
            tar_add(ls(o), tag_add[o], L, mid);
            tar_add(rs(o), tag_add[o], mid + 1, R);
            tag_add[o] = 0;
        }
    }
    void update_add(int o, int nowl, int nowr, int L, int R, int Val){
        if(L <= nowl && nowr <= R) return tar_add(o, Val, nowl, nowr);
        int mid = (nowl + nowr) >> 1;
        push(o, nowl, nowr);
        if(L <= mid) update_add(ls(o), nowl, mid, L, R, Val);
        if(R  > mid) update_add(rs(o), mid + 1, nowr, L, R, Val);
        maintain(o);
    }
    void update_mul(int o, int nowl, int nowr, int L, int R, LB Val){
        if(L <= nowl && nowr <= R) return tar_mul(o, Val, nowl, nowr);
        int mid = (nowl + nowr) >> 1;
        push(o, nowl, nowr);
        if(L <= mid) update_mul(ls(o), nowl, mid, L, R, Val);
        if(R  > mid) update_mul(rs(o), mid + 1, nowr, L, R, Val);
        maintain(o);
    }
    void update_ret(int o, int nowl, int nowr, int L, int R){
        if(L <= nowl && nowr <= R) return tar_ret(o, nowl, nowr);
        int mid = (nowl + nowr) >> 1;
        push(o, nowl, nowr);
        if(L <= mid) update_ret(ls(o), nowl, mid, L, R);
        if(R  > mid) update_ret(rs(o), mid + 1, nowr, L, R);
        maintain(o);
    }
    LB query(int o, int nowl, int nowr, int L, int R){
        if(L <= nowl && nowr <= R) return v[o];
        int mid = (nowl + nowr) >> 1;
        push(o, nowl, nowr);
        LB ans = LLONG_MIN;
        if(L <= mid) ans = max(ans, query(ls(o), nowl, mid, L, R));
        if(R  > mid) ans = max(ans, query(rs(o), mid + 1, nowr, L, R));
        return ans;
    }
}using SegmentTree::build;       using SegmentTree::query;       using SegmentTree::update_add;
 using SegmentTree::update_mul;  using SegmentTree::update_ret;  using SegmentTree::Groot;
```

```
    void work(){
        init_query();
        int root = Groot(); build(root, 1, n); LB one = 1;
        while(q--) {
            int t, L, R, x; Read(t)(L)(R)(x); L ++, R ++;
            if(t == 0) update_add(root, 1, n, L, R, x);
            if(t == 1) update_mul(root, 1, n, L, R, one / x);
            if(t == 2) printf("%lld\n", (long long)(query(root, 1, n, L, R) + 1e-10));
            if(t == 3) update_ret(root, 1, n, L, R);
        }
    }
}
int main(){ //freopen("in.txt", "r", stdin);
    Read(n)(q); rep(i, 1, n) Read(A[i]);
    Acceptable_Solution::work();
    return 0;
}
```

D. four

你有 $1..2n$ 共 $2n$ 个数，有 n 个人，你会给每个人分两个随机的数，这样就把 $2n$ 个数分完了，每个人获得的总数就是这两个数相加。这个随机过程可以认为是，先把 $2n$ 个数随机排列，将前 2 个数分给第一个人，将接下来前 2 个数分给第二个人……不难看出这个过程是没有歧义的

你想知道有多大概率，使得总和最大的人唯一（不存在两个相同的最大值），答案模 998244353

没听懂…。

Round 3

A. diyiti

有 n 个物品，每个物品价格 A_i ，收益为 B_i ，由于一些奥妙重重的原因，当你买了一些物品，总价格为选择的 A_i 的按位或而不是代数相加!

你身上只有 k 元钱，你希望选择总价格不超过 k 的一些物品，来获得最大的收益。

对于所有数据，$1 \le n \le 10^5, 0 \le k < 2^{30}, 0 \le A_i < 2^{30}, 0 \le B_i \le 10^9$。

一种天然的想法就是考虑枚举，每一个 $i \le k$ 然后累加所有可以被选的物品求最大值，显然能保证正确性。得到一个 $\mathcal{O}(kn)$ 的优秀做法。发现这样枚举的每一个 i 有很多没有必要的枚举。

其实可以枚举放弃 k 的哪一位 1。

例如: 若 $k = (1001101001)_2$ 那么有意义的 i 就可能是：- $(0111111111)_2$ - $(1000111111)_2$ - $(1001011111)_2$ - $(1001100111)_2$ - $(1001101001)_2$

也就是枚举某一位 1 把这个 1 变成 0，然后把这后面的所有数字变成 1。

```
int n, k;
int Lim[_], Pr[_];
LL calc(int x){
    LL ans = 0;
    rep(i, 1, n) if((x | Lim[i]) == x) ans += Pr[i];
```

```
    return ans;
}
int main(){
    Read(n)(k); rep(i, 1, n) Read(Lim[i])(Pr[i]);
    LL ans = calc(k);
    for(int i = 30; i >= 0; i--){
        if(k & (1 << i)) ; else continue;
        int Lit = k;
        Lit ^= (1 << i); Lit |= ((1 << i) - 1);
        ans = max(calc(Lit), ans);
    }
    printf("%lld", ans);
    return 0;
}
```

B. dierti

给你一个长度为 n 的数组 a_n，你需要求出一个最长的子序列 C，使得其相邻两项按位与的结果均非零 $(\forall\ i,\ C_i\ \ \&\ C_{i+1} > 0)$。

这比第一题好想多了吧。考虑 $dp[n]$ 表示 $[1,n]$ 中强制 n 一定选 符合要求的最长子序列，可以考虑枚举 $[1..n-1]$ 中上一个选择的是哪一个，即 $dp[n] = 1 + \max\limits_{i=1}^{n-1} dp[i][C_i\ \&\ C_n > 0]$ 这样的复杂度是 $\mathcal{O}(n^2)$ 的。

考虑如果满足 $C_i\ \&\ C_j > 0$ 那么需要 C_i, C_j 至少一个二进制位置相同。那就维护一个数组，$Bmax[i]$ 表示 C 的第 i 位为 1 的所有 C_j 最大的 $dp[j]$，即可实现 $\mathcal{O}(\log C_{max})$ 转移。同时需要 $\mathcal{O}(\log C_{max})$ 维护数组 $Bmax$。

总复杂度 $\mathcal{O}(n \log C_{max})$。

```
int n, A[_];
int Bmax[60];
int dp[_];
int main(){
    n = read(); rep(i, 1, n) Read(A[i]);
    dp[1] = 1; rep(j, 0, 30) if(A[1] & (1 << j)) Bmax[j] = max(Bmax[j], dp[1]);
    rep(i, 2, n){
        int ans = 0;
        rep(j, 0, 30) if(A[i] & (1 << j)) ans = max(ans, Bmax[j]);
        dp[i] = ans + 1;
        rep(j, 0, 30) if(A[i] & (1 << j)) Bmax[j] = max(Bmax[j], dp[i]);
    }
    int ans = 0; rep(i, 1, n) ans = max(ans, dp[i]);
    printf("%d", ans); cerr << "std's ans = " << ans << endl;
    return 0;
}
```

C. disanti

statement 你有 n 个 10 进制下的位数为 m 的数

有些数字被抹去了，因此取值任意。

你希望这 n 个数单调递增，请求出所有满足这个条件的方案下，这 n 个数之和的总和。

答案对 998244353 取模。

$n, m \leq 30$ #### input

第一行两个整数 $n, m(1 \leq n, m \leq 50)$

接下来 n 行，每行 m 个字符，字符为 0..9 的数字或者?。

```
INPUT0:
4 1
0
?
4
8
OUTPUT0:
42
```

```
INPUT1:
5 5
?1234
23333
???66
??666
?233?
OUTPUT1:
819525655
```

没听懂…待补。

D. disiti

你有一堆 n 个石子的石子堆，你想要把石子堆裂成 n 个大小为 1 的石子堆。为了达成这个目的，你每次可以进行如下操作：假设现在有 k 堆石子，大小分别为 $a_1, a_2, ..., a_k$，你可以指定一个非负整数序列 $b_1, b_2, ..., b_k$，满足 $\sum b_i \leq m$，每堆石子就会分裂成两堆 $b_i, a_i - b_i$（如果为 0 就认为不存在，显然还需要满足 $b_i \leq a_i$）。你想要知道最少几次操作可以达成目的。对于所有数据，$T \leq 1000, 1 \leq m \leq n \leq 10^9$

没听懂…待补。

Round 4

A. 一

你是能看到第一题的 friends 呢。——hja 众所周知，小葱同学擅长计算，尤其擅长计算组合数，但这个题和组合数没什么关系。

现在我们要执行若干代码，代码为对于变量的操作，包含以下三种：- 1、变量名 = 变量值，变量名由小写字母组成，变量值有可能是正整数或者字符串（小写字母或者数字），例如：a=3,b= “233”。- 2、变量名，代表询问该变量的值，如果该变量未赋值，则输出 no。- 3、变量名 += 值，如果该变量未定义，直接跳过该次操作。设该变量的值为 x，加上的值为 y，如果两者均为整数直接相加；如果 x 是整数 y 为字符串则跳过此次操作；如果 x 是字符串 y 是整数则将 y 转换为字符串进行字符串拼接；如果两者均为字符串直接进行字符串拼接。-

全程 STL。

```
map<string , string> M;
map<string , bool > Type; // true for string , false for int
```

```
string tmp;
pair<string , string> Res;
int split(string S){
    if(S.find('=') >= S.size() || S.find('=') < 0){ // TODO: check
        return 2;
    } else {
        if(S.find('+') < S.size() && S.find('+') >= 0){
            Res = make_pair(S.substr(0, S.find('+')), S.substr(S.find('=') + 1, S.size() - S.find('=')
            return 3;
        } else {
            Res = make_pair(S.substr(0, S.find('=')), S.substr(S.find('=') + 1, S.size() - S.find('=')
            return 1;
        }
    }
}
string calc(string A, string B){
    long long ResA = 0;
    rep(i, 0, A.size() - 1){
        ResA *= 10;
        ResA += A[i] - '0';
    }
    long long ResB = 0;
    rep(i, 0, B.size() - 1){
        ResB *= 10;
        ResB += B[i] - '0';
    }
    ResA += ResB;
    string Res, ans ; Res = "";
    while(ResA) Res.push_back(ResA % 10 + '0'), ResA /= 10;
    for(int i = Res.size() - 1; i >= 0; i--) ans.push_back(Res[i]);
    return ans;
}
bool pdType(string &S) { return S[0] == '"'; } // true for string
int main(){ //freopen("in.txt", "r", stdin); freopen("out.txt", "w", stdout);
    ios::sync_with_stdio(false);
    int q; cin >> q;
    while(q--){
        cin >> tmp;
        int r = split(tmp);
        if(r == 1){
            int re = pdType(Res.second); if(re) Res.second = (Res.second.size() == 2 ? string("") : Res
            M[Res.first] = Res.second;
            Type[Res.first] = re;
        } else if(r == 2){
            if(M.count(tmp))  cout << M[tmp] << endl;
            else cout << "no" << endl;
        } else {
```

```
            if(M.count(Res.first)) ; else continue;
            int ex_t = pdType(Res.second); if(ex_t) Res.second = Res.second.substr(1, Res.second.size()
            int no_t = Type[Res.first];
            if( ex_t &&  no_t) M[Res.first] = M[Res.first] + Res.second;
            if( ex_t && !no_t) { continue; }
            if(!ex_t &&  no_t) M[Res.first] = M[Res.first] + Res.second;
            if(!ex_t && !no_t) M[Res.first] = calc(M[Res.first], Res.second);
        }
    }
    return 0;
}
```

B. 二

你是能看到第二题的 friends 呢。——aoao

众所周知，小葱同学擅长计算，尤其擅长计算组合数，但这个题和组合数没什么关系。N 个小写字母字符串，Alice 和 Bob 按照如下方法玩游戏：- 1、Alice 选择一个字符串，并重新定义 26 个字母的字典序，然后再将这个字符串内部字符按照任意顺序重排。- 2、Bob 从剩下的字符串中选择一个，并将这个字符串内部字符按照任意顺序重排。他们的目标是使得自己字符串的字典序比对面小，谁的字符串字典序更小谁就赢了。问 Alice 有多少个可以选择的字符串能够使得自己获胜？$N \leq 1000$

因为可以重新排列字符串，所以对于每个字符串来说，有用的信息就是每个字符串中每个字符的出现次数。

可以考虑先枚举一个字符串，然后判断这个字符串是否能被 Alice 选。然后考虑确定字典序，可以以此考虑字典序最小的应该是哪个字符，确定了字典序最小的字符之后，可以排除掉一些字符串（字典序最小的字符在这些串中的出现次数 $<$ 在枚举串中的出现次数），然后又转化成原来相同的问题。

关键在于如何确定当前情况下字典序最小的字符。显然，理想的字典序最小的字符 在当前枚举到的字符串中的出现次数一定不少于在其他串中的出现次数。但是这样的字符可能有多个，应该选哪个？

其实选哪个效果一样，本质上只是以此选择当前情况下字典序最小的字符，然后剔除一些字符串，但是剔除字符串的条件和剔除顺序无关。

最后判断是否所有字符串都被剔除掉即可。

```
#include <cstdio>
#include <cstring>
#include <iostream>
#include <cmath>
#include <cstring>
#include <algorithm>
#define rep(i, l, r) for(int i = (l), __ = (r); i <= __; i++)
const int _ = 1010;
using namespace std;
inline int idx(char c) { return c - 'a' + 1; }
int ch[_][30]; int n;
char S[_];
bool book[_];
bool bch[200];
int main(){
    scanf("%d", &n); rep(i, 1, n) { scanf("%s", S + 1); int k = strlen(S + 1); rep(j, 1, k) ch[i][idx(S
```

```
    int ans = 0;
    rep(i, 1, n){
        memset(book, 0, sizeof(book)); memset(bch, 0, sizeof(bch)); book[i] = 1;
        rep(j, 1, 26) {
            rep(k, 'a', 'z') { if(bch[k]) continue;
                bool pass = true;
                rep(l, 1, n) {
                    if(book[l]) continue;
                    if(ch[l][idx(k)] > ch[i][idx(k)]) { pass = false; break; }
                }
                if(pass){
                    bch[k] = 1;
                    rep(l, 1, n) if(ch[l][idx(k)] < ch[i][idx(k)]) book[l] = 1;
                    break;
                }
            }
        }
        bool pass = true; rep(i, 1, n) if(book[i] != 1) { pass = false; break; }
        ans += pass;
    }
    printf("%d", ans);
    return 0;
}
```

C. 三

你是能看到第三题的 friends 呢。——laekov 众所周知，小葱同学擅长计算，尤其擅长计算组合数，但这个题和组合数没什么关系。

给定 N 个数，设总共有 M 个区间 $p_1 = [l_1, r_1], p_2 = [l_2, r_2], \cdots, p_M = [l_m, r_m]$ 的逆序对数量不少于 K 个。定义函数 $f(p_i, p_j)$ 为计算两个区间的交集大小的函数（即共同覆盖了多少个数）。求 $\sum_{i=1}^{M} \sum_{j=i+1}^{M} f(p_i, p_j)$ $N \leq 10^6$

考虑每一个元素的贡献，不难发现，元素的贡献为 $\binom{}{2}$，问题转化为怎么求每个元素被覆盖的次数。

考虑对于每一个左端点 L，合法的右端点取值一定是一段连续的区间，且 如果存在上界一定是 n ，对于每个左端点，求出其对应的最小的右端点，这里可以使用 two-point 。值域树状数组统计逆序对数。

考虑每个左端点为 L 的区间对元素覆盖数的贡献，发现一定是一条平直线和一个下降直线，可以对差分数组的差分数组做修改，然后两次前缀和还原出原序列。

```
const int _ = 2e6 + 100;
const int inv2 = 500000004;
int n, k, A[_];
namespace BIT{
    #define lowbit(x) (x & (-x))
    int C[_];
    void add(int p, int x) { if(p == 0) return ; for(int i = p; i <= n; i += lowbit(i)) C[i] += x; }
    int  query(int p) { int ans = 0; for(int i = p; i >= 1; i -= lowbit(i)) ans += C[i]; return ans; }
} using BIT::add; using BIT::query;
int L = 0, R = 0;
int Rp[_];
```

```
int C[_];
int main(){
    Read(n)(k); rep(i, 1, n) Read(A[i]);
    LL ans = 0;
    rep(L, 1, n){
        if(L - 1 > 0) add(A[L - 1], -1);
        if(L - 1 > 0) ans -= query(A[L - 1] - 1);
        while((R < L) || (ans < k && R <= n)) { if(R == n) { R++; break; }; add(A[++R], 1); ans += quer
        Rp[L] = R;
    }
    rep(L, 1, n){
        if(Rp[L] > n) continue; int cnt = (n - Rp[L] + 1);
        C[L] = (C[L] +0ll+ cnt) % MOD; C[L + 1] = (C[L + 1] +0ll+ (MOD -0ll- cnt) % MOD) % MOD;
        C[Rp[L] + 1] = (C[Rp[L] + 1] +0ll+ MOD - 1) % MOD;
    }
    rep(i, 1, n) C[i] = (C[i] +0ll+ C[i - 1]) % MOD;
    rep(i, 1, n) C[i] = (C[i] +0ll+ C[i - 1]) % MOD;
    ans = 0;
    rep(i, 1, n) ans = (ans +0ll+ (C[i] *1ll* (C[i] - 1) % MOD *1ll* inv2 % MOD)) % MOD;
    printf("%lld", ans);
    return 0;
}
```

D. 四

你是能看到第四题的 friends 呢。——laekov

众所周知，小葱同学擅长计算，尤其擅长计算组合数，但这个题和组合数没什么关系。

今天也是皮克敏们打工的一天，你有 N 只皮克敏，每只皮克敏有一颗炸弹。现在有一颗 N 个点的树，同时皮克敏们可以从 p_1 或者 p_2 两个点中的任意一个进入树。每次你需要派一只皮克敏去到某个点，然后将炸弹连同皮克敏和这个点一起炸掉。一个点被炸掉了之后皮克敏就再也不能通过了。现在你需要决定皮克敏们炸点的顺序，问 $n!$ 种炸点方案中有多少种能炸掉所有点。$N \leq 1000$

考虑 DP 。

首先考虑一个子树（子树内没有 p_1, p_2）全部被清空应该是什么顺序，一定是每个儿子的子树先被清空然后清楚当前点。这里可以使用树形 dp 处理。

先考虑 $p_1 \leftrightarrow p_2$ 这一条链上的点应该是按照什么顺序被清空，任意时刻，被清空的点一定是链上的连续的一段，这里可以区间 dp 统计答案，转移考虑最后一次删除是删干净了链上最右边的点还是最左边的点。

然后和 p_1, p_2 的每个不在链上的儿子合并答案，最后决策先删除 p_1 还是 p_2。代码还没写。

Round 5

A. 一

现在有四种颜色的东西，各有 n_1, n_2, n_3, n_4 个。你需要把他们放到一排里面，并且保证相邻的东西颜色不同，问方案数。对于 80% 的数据，$n_1 + n_2 + n_3 + n_4 \leq 10$。对于另外 10% 的数据，$n_1 = 0, n_2, n_3, n_4 \leq 50$。对于 100% 的数据，$0 \leq n_1, n_2 \leq 200, 0 \leq n_3, n_4 \leq 50000$。

```
namespace subtask1{
    int dp[_][_][_][_][5];
    void work(){
        memset(dp, 0, sizeof(dp));
        dp[1][0][0][0][1] = dp[0][1][0][0][2] = dp[0][0][1][0][3] = dp[0][0][0][1][4] = 1;
        rep(i, 0, A) rep(j, 0, B) rep(k, 0, C) rep(l, 0, D){
            if(i + j + k + l == 1) continue;
            if(i > 0) dp[i][j][k][l][1] = (dp[i - 1][j][k][l][2] +0ll+ dp[i - 1][j][k][l][3] +0ll+  dp[
            if(j > 0) dp[i][j][k][l][2] = (dp[i][j - 1][k][l][1] +0ll+ dp[i][j - 1][k][l][3] +0ll+  dp[
            if(k > 0) dp[i][j][k][l][3] = (dp[i][j][k - 1][l][2] +0ll+ dp[i][j][k - 1][l][1] +0ll+  dp[
            if(l > 0) dp[i][j][k][l][4] = (dp[i][j][k][l - 1][2] +0ll+ dp[i][j][k][l - 1][3] +0ll+  dp[
        }
        printf("%d\n", ans1 = int((1ll* dp[A][B][C][D][1] +  dp[A][B][C][D][2] +  dp[A][B][C][D][3] +
    }
}
namespace subtask2{
    const int _ = 53;
    int dp[_][_][_][4];
    void work(){
        memset(dp, 0, sizeof(dp));
        dp[1][0][0][1] = dp[0][1][0][2] = dp[0][0][1][3] = 1;
        rep(i, 0, B) rep(j, 0, C) rep(k, 0, D){
            if(i + j + k == 1) continue;
            if(i > 0) dp[i][j][k][1] = (dp[i - 1][j][k][2] +0ll+ dp[i - 1][j][k][3]) % MOD;
            if(j > 0) dp[i][j][k][2] = (dp[i][j - 1][k][1] +0ll+ dp[i][j - 1][k][3]) % MOD;
            if(k > 0) dp[i][j][k][3] = (dp[i][j][k - 1][2] +0ll+ dp[i][j][k - 1][1]) % MOD;
        }
        printf("%d\n", ans2 = int(( dp[B][C][D][1] +0ll+ dp[B][C][D][2] +0ll+ dp[B][C][D][3] ) % MOD));
    }
}
```

B. 二

N 个二元组 (a_i, b_i)，定义 $c_1 = a_1 + b_1, c_i = b_i + max\ (c_{i-1}, \sum_{j=1}^{i} a_j)$。现在你可以随意重排这 N 个二元组，求 c_N 的最小值 $N \leq 10^6; A, B \leq N$

考虑相邻两个二元组 $(a_1, b_1)(a_2, b_2)$ 应该怎么比较大小。分别列出 (a_1, b_1) 在 (a_2, b_2) 之前的答案和交换之后的答案。

如果 (a_1, b_1) 在 (a_2, b_2) 之前：设这两个二元组之前所有二元组中 a 的和为 x，设上一个二元组的 C 值为 y。根据定义 $C_1 = b_1 + \max(y, a_1 + x)$，$C_2 = b_2 + max(C_1, x + a_1 + a_2)$ 化简后得到 $C_2 = \max(y + b_1 + b_2, x + a_1 + b_1 + b_2, x + a_1 + a_2 + b_2)$ 易知 交换后 $C_2' = \max(y + b_1 + b_2, x + a_2 + b_1 + b_2, x + a_1 + a_2 + b_1)$ 现在需要比较 C_2 和 C_2' 的大小。相当于对 6 个式子取最大值，看最大值出现在哪边。相同的项 $y + b_1 + b_2$ 可以消去，如果其为最大值，那么两个式子谁在前谁在后无所谓，如果其不是最大值，那么也没有影响。消去每一项中相同的 x。

$C_2 = \max(a_1 + b_1 + b_2, a_1 + a_2 + b_2)$

$C_2' = \max(a_2 + b_1 + b_2, a_1 + a_2 + b_1)$

提出共同的项：$C_2 = a_1 + b_2 + \max(b_1, a_2)$

$C_2' = a_2 + b_1 + \max(b_2, a_1)$

假设 $C_2 < C_2'$

则 $a_1 + b_2 + \max(b_1, a_2) < a_2 + b_1 + \max(b_2, a_1)$

移项得 $a_1 + b_2 - \max(b_2, a_1) < a_2 + b_1 - \max(b_1, a_2)$

得出 $\min(a_1, b_2) < \min(a_2, b_1)$

一种特殊情况是取等的时候：举几个栗子得出 结论是取等时比较 $a_1 < a_2$。

```
bool CMP(pair<int, int> x, pair<int, int > y){
    int r0 = min(x.first, y.second);
    int r1 = min(y.first, x.second);
    if(r0 != r1) return r0 < r1;
    return x.first < y.first;
}
```

C. 三

给定 N 个数，M 次操作，操作有以下四种：- 1、区间加一个数。- 2、区间乘一个数。- 3、区间变成一个数。- 4、求所有子区间的平均值的和 $(MOD = 10^9 + 7)$。$a_i, N, Q \leq 10^5$

$Ans = \sum_{i=1}^{n} \sum_{j=i}^{n} \frac{\sum_{k=i}^{j} a_k}{j-i+1}$ 考虑每个数字的贡献 $Ans = \sum_{k=1}^{n} a_k \times \sum_{i=1}^{n} \sum_{j=i}^{n} \frac{1}{j-i+1}$ 算出贡献，线段树维护即可。可以考虑 k 每增加 1 贡献会变化多少。也可以发现贡献类似于一个梯形，可以直接算。

```
void work(){ cerr << "FUCK" << endl;
    S[1] = 1; rep(i, 2, n) S[i] = (S[i - 1] +0ll+ inv(i)) % MOD;
    int ans = 0, dp = 0;
    rep(i, 1, n){
        int last = i - 1; dp = (dp -0ll- S[last] + MOD) % MOD;
        dp = (dp -0ll- (S[n] -0ll- S[n - last] +0ll+ MOD) % MOD +0ll+ MOD) % MOD;
        dp = (dp +0ll+ (S[n] -0ll- S[i] +0ll+ MOD) % MOD) % MOD;
        dp = ( dp +0ll+ S[i]) % MOD; // 这里的 dp 就是位置为 i 的数字对答案贡献的系数。
        ans = (ans +0ll+ dp *1ll* A[i] % MOD) % MOD;
    }
    printf("%d", ans);
}
```

D. 四

众所周知，小葱同学擅长计算，尤其擅长计算组合数，但这个题和组合数没什么关系。

皮克敏们打完工了，是时候将所有皮克敏处理掉了。现在皮克敏们躲在一棵树上（点和边的任意位置都有可能），你可以选择若干叶子节点释放毒气，毒气会以每单位时间一单位的距离沿着边蔓延开来。为了能够灭绝皮克敏，你需要保证树上每个位置都充满了毒气。但是仅仅是求一个最小的灭绝皮克敏的时间实在太无趣了，你想要知道你有多少种不同的方法能够灭绝皮克敏，两个方法不同当且仅当两种方法灭绝所有皮克敏的时间不同。

$N \leq 200$ 第一行一个整数 N 代表树上点的个数。

接下来 $N-1$ 行每行三个整数 s, e, d 代表一条边的两端和长度。

答案只能是某两个叶子之间的距离或者距离 $ / 2$ 。枚举两个叶子，把 - 这个叶子之间的距离（只在一边放毒气）（如果能成为答案）- 两个叶子之间距离 $ / 2$（两个叶子上都放毒气）（如果能成为答案）加入答案集合。最后输出答案集合的大小。

枚举两个叶子，判断其距离是否能成为答案。考虑哪些其他叶子放毒气能让这个距离尽可能成为答案，就是那些放上毒气不会影响这两个枚举的叶子之间毒气传播时间的叶子都放上毒气，然后跑一遍最短路，算全树被毒气覆盖的时间是否等于当前枚举的叶子。

```
const int _ = 510;
pair<pair<int, int>, int > E[_];
int n, m;
int M[_][_];
set<int> Ans;
queue<int>Q; bool inQ[_];
int SPFA(){
    static int dis[_]; memset(dis, 0x3f, sizeof(dis));
    rep(i, 1, n) if(inQ[i]) dis[i] = 0;
    while(!Q.empty()){
        int now = Q.front(); Q.pop(); inQ[now] = 0;
        for(int i = head[now]; i ;i = edge[i].nxt){
            int ex = edge[i].node;
            if(dis[now] + edge[i].w < dis[ex]) {
                dis[ex] = dis[now] + edge[i].w;
                if(!inQ[ex]) Q.push(ex), inQ[ex] = true;
            }
        }
    }
    int MAX = INT_MIN;
    rep(i, 1, n - 1) {
        pair<pair<int, int>, int> now = E[i];
        int T = min(dis[now.first.first] + now.second, dis[now.first.second] + now.second, (dis[now.fir
        MAX = max(MAX, T);
    }
    return MAX;
}
int ind[_];
bool work1(int x, int y, int D) {
    Q.push(x); inQ[x] = 1;
    rep(i, 1, n) { if(i == x || i == y) continue; if(M[i][y] >= D && ind[i] == 1) Q.push(i), inQ[i] = 1
    return SPFA() == D;
}
bool work2(int x, int y, int D){
    Q.push(x); Q.push(y); inQ[x] = inQ[y] = 1;
    rep(i, 1, n){
        if(x == i || y == i) continue;
        if(max(M[x][i], M[y][i]) / 2 >= D && ind[i] == 1) Q.push(i), inQ[i] = 1;
    }
    return SPFA() == D;
}

int main(){
```

```
    memset(M, 0x3f, sizeof(M));
    rep(i, 1, n) M[i][i] = 0;
    Read(n); rep(i, 1, n - 1) { int u, v, w; Read(u)(v)(w); ind[u] ++; ind[v]++; w <<= 1; add(u, v, w);
    rep(k, 1, n) rep(i, 1, n) rep(j, 1, n) M[i][j] = M[j][i] = min(M[i][j], M[i][k] + M[k][j]);
    rep(i, 1, n) rep(j, 1, n){ if(i == j) continue; if(ind[i] != 1 || ind[j] != 1) continue;
        if(!Ans.count(M[i][j])      && work1(i, j, M[i][j]))      Ans.insert(M[i][j]);
        if(!Ans.count(M[i][j] >> 1) && work2(i, j, M[i][j] >> 1)) Ans.insert(M[i][j] >> 1);

    }
    printf("%d", int(Ans.size()));
    return 0;
}
```

Round 6

A. 一

现在我们有两队皮克敏，个数分别为 n_1, n_2，现在我们要杀掉这些皮克敏，按照如下规则操作：对于第 i 轮杀皮克敏的操作，我们首先选择皮克敏较多的那一队，如果一样就选择第一队，然后杀掉这队中的 i 个皮克敏，如果不够，游戏结束，记做游戏在第 i 轮结束。问最后两队各剩下多少个皮克敏？$n \leq 10^{16}$

分成两阶段，一阶段是只杀多的那队，直到比另一队少。二阶段是两队交替着杀，可以证明，如果经过了第一阶段，那么第二阶段一定是交替着杀。杀就完了。分别二分即可，当然第一步也可以直接解方程。

```
LL A, B;
inline LL check(LL x) { return x * (x + 1) >> 1; }
LL Get(LL res){
    LL L = 0, R = 1e9, ans = 0;
    while(L < R){
        LL mid = L + ((R - L + 1) >> 1);
        if(check(mid) <= res) ans = mid, L = mid;
        else R = mid - 1;
    }
    return ans;
}
LL Start = 0; LL tA, tB;
bool check0(LL Round, LL A, LL B){
    LL dA, dB; bool sf = false;
    if(A < B) sf = true, swap(A, B);
    if(Round & 1) {
        dA = (Start + (Start + Round - 1)) * (Round / 2 + 1) >> 1;
        dB = (Start + 1 + (Start + Round - 2)) * (Round / 2) >> 1;
    } else {
        dA = (Start + (Start + Round - 2)) * (Round / 2) >> 1;
        dB = (Start + 1 + (Start + Round - 1)) * (Round / 2) >> 1;
    } A -= dA; B -= dB;
    if(sf) swap(A, B);
    tA = A; tB = B;
    return A >= 0 && B >= 0;
```

```
}
void doit(){
    Read(A)(B);
    LL d = max(A, B) - min(A, B);
    LL Round = Get(d); LL del = check(Round);
    if(A < B) B -= del; else A -= del;
    if(A == B) {
        if(A < Round + 1) return (void)printf("%lld %lld %lld\n", Round + 1, A, B);
        else Round ++, A -= Round;
    } else {
        if(max(A, B) < Round + 1) return (void)printf("%lld %lld %lld\n", Round + 1, A, B);
        else Round ++, (A > B ? A -= Round : B -= Round);
    }
    LL L = 0, R = 1e9, ans = 0; Start = Round + 1;
    while(L < R) {
        LL mid = L + ((R - L + 1) >> 1);
        if(check0(mid, A, B)) ans = mid, L = mid;
        else R = mid - 1;
    } check0(ans, A, B);
    printf("%lld %lld %lld\n", ans + Round + 1, tA, tB);
}
int main(){
    int T = read(); cerr << "std's T = " << T << endl;
    while(T--) doit();
    return 0;
}
```

B. 二

现在有三组 N 个数记做 A, B, C，定义函数 $f(A, B) = \sum_{i=1}^{N} A_i \times B_i$。现在给定 A, B，并告诉你 $f(A, C) = x$，现在想求在满足 $\sum_{i=1}^{N} C_i = 1, 0 \leq C_i \leq 1$ 的情况下 $f(B, C)$ 的最大值。对于 100% 的数据，$1 \leq N, M \leq 100, 1 \leq A_i, B_i \leq 100$

[skip] 结论是只有两个点有用，只枚举两个点即可。

C. 三

众所周知，小葱同学擅长计算，尤其擅长计算组合数，但这个题和组合数没什么关系。

现在有一个 N 的排列，将其每个数看做一个集合。定义一种对两个集合的运算为：$f(s_1, s_2) = \sum_{x \in s_1} [\exists y \in s_2, y < x]$

也可以用这样一段伪代码去理解：

```
Ans=0
For x in s1:
    Able = False
    For y in s2:
        If y<x:
            Able=True
    If Able:
```

```
        Ans++
Return Ans
```

现在你每次可以合并两个相邻的集合将其变为两个集合的并集，其代价为 $f(s_1,s_2)+f(s_2,s_1)$。求最小代价，将所有集合合并为一个集合。$N \leq 10^2$

函数 $f(A,B)$ 其实就是 A 中有多少元素大于 $\min B_i$ 。按照定义区间 dp 即可，由于是小于 $\mathcal{O}(n^2)$ 级别的二维数点，可以使用二维前缀和。查询最小值随手预处理一下 ST 表即可。

```cpp
const int _ = 600;
int dp[_][_];
int S[_][_];
int n, A[_];
int query(int L, int R, int up){ // how many elements in range[L, R] which is > up
    return S[n][R] - S[up][R] - S[n][L - 1] + S[up][L - 1];
}
const int _S = 4000;
const int LOG = 10;
int ST[_S][LOG + 2];
int Log[_S];
void initQuery(){
    rep(i, 1, n) ST[i][0] = A[i];
    rep(j, 1, LOG) rep(i, 1, n) ST[i][j] = min(ST[i][j - 1], ST[i + (1 << (j - 1))][j - 1]);
    Log[1] = 0; rep(i, 2, n) Log[i] = Log[i >> 1] + 1;
}
int RMQ(int L, int R){
    int Ln = Log[R - L + 1];
    return min(ST[L][Ln], ST[R - (1 << Ln) + 1][Ln]);
}
int calc(int L0, int R0, int L1, int R1){
    int min0 = RMQ(L0, R0), min1 = RMQ(L1, R1);
    return query(L0, R0, min1) + query(L1, R1, min0);
}
int main(){
    Read(n); rep(i, 1, n) Read(A[i]); initQuery();
    rep(i, 1, n) S[A[i]][i]++;
    rep(i, 1, n) rep(j, 1, n) S[i][j] += S[i - 1][j] + S[i][j - 1] - S[i - 1][j - 1];
    memset(dp, 0x3f, sizeof(dp)); rep(i, 1, n) dp[i][i] = 0;
    rep(Len, 2, n){
        rep(L, 1, n){
            int R = L + Len - 1; if(R > n) break;
            int &ans = dp[L][R];
            rep(k, L, R - 1) ans = min(ans, dp[L][k] + dp[k + 1][R] + calc(L, k, k + 1, R));
        }
    }
    printf("%d", dp[1][n]);
    return 0;
}
```

D. 四

给定 $1-N$ 的排列但其中 M 个位置的值被删去了（用 0 表示），现在你需要将其复原，问有多少种方案能够使得复原的序列的逆序对个数在 $[L,R]$ 之间

折半搜索，中间 $\mathcal{O}(n^2)$ 合并。

Round 7

AK。### A. 一

有一个 $n\times n$ 的矩阵，矩阵中的每个数都是整数。现在要从矩阵中取 m 个数，要求每一行最多取一个数，每一列也最多取一个数。这 m 个数的和最大能是多少？$n\le 15, m\le 3, A_{i,j}\le 10^5$

```
const int _ = 20;
int n, m, A[_][_];
int main(){
    Read(n)(m); rep(i, 1, n) rep(j, 1, n) Read(A[i][j]);
    if(m == 1) {
        int MAX = 0; rep(i, 1, n) rep(j, 1, n) MAX = max(MAX, A[i][j]);
        printf("%d", MAX);
    } else if(m == 2){
        int MAX = 0;
        rep(a, 1, n) rep(b, 1, n) rep(c, 1, n) rep(d, 1, n){
            if(a == c || b == d) continue;
            MAX = max(MAX, A[a][b] + A[c][d]);
        }
        printf("%d", MAX);
    } else if(m == 3){
        int MAX = 0;
        rep(a, 1, n) rep(b, 1, n) rep(c, 1, n) rep(d, 1, n) rep(e, 1, n) rep(f, 1, n){
            if(a == c || c == e || a == e || b == d || d == f || b == f) continue;
            MAX = max(MAX, A[a][b] + A[c][d] + A[e][f]);
        }
        printf("%d", MAX);
    }
    return 0;
}
```

B. 二

有多少整数大于等于 x，小于等于 y，而且是 7 的倍数，又不是 2,3,5 的倍数呢？$x,y\le 10^{18}$

容斥原理

```
LL calc(LL R){
    LL ans = 0;
    ans += R / 7; ans -= R / 14; ans -= R / 21; ans -= R / 35; ans += R / 42; ans += R / 70; ans += R /
    return ans;
}
```

```
int main(){
    LL L, R; Read(L)(R);
    printf("%lld", calc(R) - calc(L - 1));
    return 0;
}
```

C. 三

给出一个 n 个点 m 条边的图。

现在想要选择两个整数 a 和 b，满足 $1 \le a < b < n$，然后把编号在区间 $[1,a]$ 中的点染成蓝色，把编号在区间 $[a+1,b]$ 中的点染成红色，编号在区间 $[b+1,n]$ 中的点染成蓝色。

如果一条边的两个端点颜色不同，我们称这条边为双色边，否则称之为单色边。

如何选择 a,b，能够最小化双色边的数量？$n,m \le 10^5$

经典套路，按顺序枚举 a，线段树维护 b 的每一个取值时的双色边数量，枚举 a 的过程中维护 b 的取值，更新答案即可。

```
int n, m;
const int _ = 3e5 + 100;
vector<int> ToL[_];
vector<int> ToR[_];
namespace SegmentTree{  // maintain the number of double-color edge;
                        // the min number ;
    const int _ = 2e6 + 100;
    int ch[_][2], v[_], tag_add[_], tot = 0;
    #define ls(o) (ch[o][0])
    #define rs(o) (ch[o][1])
    #define maintain(o) (v[o] = min(v[ls(o)], v[rs(o)]))
    #define make (tot++, ch[tot][0] = ch[tot][1] = v[tot] = tag_add[tot] = 0, tot)
    int Groot(){ return make; }
    void build(int o, int L, int R){
        if(L == R) return (void)(v[o] = 0); int mid = (L + R) >> 1;
        ls(o) = make; rs(o) = make;
        build(ls(o), L, mid); build(rs(o), mid + 1, R);
        maintain(o);
    }
    void tar(int o, int _v){ tag_add[o] += _v; v[o] += _v; }
    void push(int o) { if(!tag_add[o]) return ; tar(ls(o), tag_add[o]); tar(rs(o), tag_add[o]); tag_add
    void update(int o, int nowl, int nowr, int L, int R, int Val){
        if(L <= nowl && nowr <= R) return tar(o, Val);
        int mid = (nowl + nowr) >> 1; push(o);
        if(L <= mid) update(ls(o), nowl, mid, L, R, Val);
        if(R  > mid) update(rs(o), mid + 1, nowr, L, R, Val);
        maintain(o);
    }
    int query(int o, int nowl, int nowr, int L, int R){
        if(L <= nowl && nowr <= R) return v[o];
        int mid = (nowl + nowr) >> 1; push(o);
        int ans = INT_MAX;
```

```
        if(L <= mid) ans = min(ans, query(ls(o), nowl, mid, L, R));
        if(R  > mid) ans = min(ans, query(rs(o), mid + 1, nowr, L, R));
        return ans;
    }
}   using SegmentTree::update;  using SegmentTree::build;
    using SegmentTree::query;   using SegmentTree::Groot;
int main(){
    Read(n)(m);
    int root = Groot(); build(root, 1, n); int ans = INT_MAX;
    rep(i, 1, m){
        int u, v; Read(u)(v); if(u > v) swap(u, v);
        ToL[v].push_back(u); ToR[u].push_back(v); update(root, 1, n, u, v - 1, 1);
    }
    rep(i, 1, n - 2) {
        update(root, 1, n, i + 1, n - 1, -int(ToL[i].size()));
        rep(j, 0, ToR[i].size() - 1) update(root, 1, n, i, ToR[i][j] - 1, -1), update(root, 1, n, ToR[i
        ans = min(ans, query(root, 1, n, i + 1, n - 1));
    }
    printf("%d", ans);
    return 0;
}
```

D. 四

n 个小伙伴（编号从 0 到 $n-1$）围坐一圈玩游戏。按照顺时针方向给 n 个位置编号，从 0 到 $n-1$。最初，第 0 号小伙伴在第 0 号位置，第 1 号小伙伴在第 1 号位置，...，依此类推。游戏规则如下：- 有一个序列 $A=\{a_1,\dots,a_m\}$，其中每个数互不相同，且都是 $[1,n-1]$ 中的正整数。- 每一轮从 A 中选择一个数 a，第 0 号位置上的小伙伴顺时针走到第 a 号位置，第 1 号位置小伙伴走到第 $a+1$ 号位置。依此类推，第 $n-a$ 号位置上的小伙伴走到第 0 号位置，第 $n-a+1$ 号位置上的小伙伴走到第 1 号位置，...，第 $n-1$ 号位置上的小伙伴顺时针走到第 $a-1$ 号位置。也就是说，第 i 号位置上的小伙伴走到第 $(i+a) \mod n$ 号位置

游戏进行 t 轮。由于每一轮都需要从 A 中的 m 个数中选一个数，所以游戏共有 m^t 种玩法。其中有多少种玩法能够使得游戏结束时，第 0 号小伙伴所在的位置编号是 d 的倍数？（0 也是 d 的倍数）、对 10^9+7 取模。$1\le m\le n\le 1000, 1\le t\le 10^9, 1\le d\le n$。

设 $dp[i]$ 为进行若干轮之后在位置 i 的方案数。转移显然是卷积。包装了循环卷积的定义。

```
namespace subtask_acceptable_solution{
    const int _ = 1500;
    struct Matrix{
        int dp[_];
        Matrix (){ memset(dp, 0, sizeof(dp)); }
        Matrix operator * (const Matrix & rhs) {
            Matrix Res;
            rep(i, 0, n - 1) rep(j, 0, n - 1) Res.dp[(i + j) % n] = (Res.dp[(i + j) % n] +0ll+ dp[i] *1
            return Res;
        }
    };
    Matrix pow(Matrix a, int b){
```

```
        Matrix ans = a; b--;
        while(b){
            if(b & 1) ans = ans * a;
            a = a * a;
            b >>= 1;
        }
        return ans;
    }
    void work(){
        Matrix a;
        rep(i, 1, m) a.dp[A[i]] ++;
        Matrix Ans = pow(a, t);
        int ans = 0;
        for(int i = 0; i < n; i += d) ans = (ans +0ll+ Ans.dp[i]) % MOD;
        printf("%d\n", ans);
    }
}
int main(){ //freopen("in.txt", "r", stdin);
    Read(n)(m)(t)(d); rep(i, 1, m) A[i] = read() % n;
    subtask_acceptable_solution::work();
    return 0;
}
```

Round 8

A. 双挂

今年期中考试考了《程序设计》、《算法设计》和《数据结构》共三门课。由于考试太难了，好多同学都挂科了。班里一共 n 个人，其中 a 个人挂了《程序设计》，b 个人挂了《算法设计》，c 个人挂了《数据结构》。

如果一个同学挂了恰好两门课，那么我们就说他“双挂”了。给出 n, a, b, c，问最少有多少同学“双挂”了。

$0 \le a, b, c \le n, n \le 10^9$

设三门课挂科的人对应集合分别为 $\mathbb{A}, \mathbb{B}, \mathbb{C}$

显然并不是所有人都会挂科目。

$|\mathbb{A} \cup \mathbb{B} \cup \mathbb{C}| = |\mathbb{A}| + |\mathbb{B}| + |\mathbb{C}| - |\mathbb{A} \cap \mathbb{B}| - |\mathbb{A} \cap \mathbb{C}| - |\mathbb{B} \cap \mathbb{C}| + |\mathbb{A} \cap \mathbb{B} \cap \mathbb{C}| \le n$

移项得

为了方便：设 $S = \mathbb{A} \cap \mathbb{B} \cap \mathbb{C}$

$(|\mathbb{A} \cap \mathbb{B}| - |S|) + (|\mathbb{A} \cap \mathbb{C}| - |S|) + (|\mathbb{B} \cap \mathbb{C}| - |S|) \ge |\mathbb{A}| + |\mathbb{B}| + |\mathbb{C}| + |S| - n - 3|S|$

答案取值为 $|\mathbb{A}| + |\mathbb{B}| + |\mathbb{C}| - 2|S| - n$，其中 $|S|$ 越大 答案越小，其最大取值就是 $\max\{\mathbb{A}, \mathbb{B}, \mathbb{C}\}$

```
LL n, a, b, c; Read(n)(a)(b)(c);
printf("%lld", max(0LL, a + b + c - 2 * min(a, min(b, c)) - n));
```

B. DDL 选手

DDL 选手总是在最后做作业。

有 n 个作业，第 i 个作业的 DDL 时间是 d_i，做这个作业需要 t_i 的时间。

做作业要一心一意，所以不能同时做两个作业。如果现在开始做第 i 个作业，那么接下来 t_i 的时间都要做这个作业，不能做别的。

DDL 选手总是先做 DDL 时间早的作业。如果两个作业的 DDL 时间一样，那么 DDL 选手会先做其中编号小的

为 DDL 选手设计每个作业开始做的时间，使得 DDL 选手先做 DDL 早的作业（如果两个作业的 DDL 时间一样，那么先做其中编号小的），而且每个作业都能在 DDL 时间之前做完（时间是连续的，“之前”的意思是，最晚恰好这个时间完成。）。在保证完成的前提下，尽可能把开始做作业的时间往后推。（也就是如果有多个方案可以做完作业，选择开始时间最晚的方案。可以证明，最优方案中，每一个作业的开始时间都不早于其他方案的开始时间。输出这个最优方案。）

$n \leq 10^5, t_i \leq 10^4$

依照题意反向模拟。

```
const int _ = 2e5 + 100;
#define int long long
struct HomeWork_t{
    int d, t, id;
}H[_]; int n ;
int Ans[_];
bool CMP(const HomeWork_t & A, const HomeWork_t & B) { return (A.d < B.d) || (A.d == B.d && A.id < B.id
#undef int
int main(){
    Read(n); rep(i, 1, n) Read(H[i].d)(H[i].t), H[i].id = i;
    sort(H + 1, H + 1 + n, CMP);
    #define int long long
    int Last = LLONG_MAX;
    for(int i = n; i >= 1; i--){
        if(Last >= H[i].d) {
            Last = H[i].d - H[i].t;
            Ans[H[i].id] = Last;
        } else {
            Last = (Last) - H[i].t;
            Ans[H[i].id] = Last;
        }
    }
    rep(i, 1, n) printf("%lld\n", Ans[i]);
    #undef int
    return 0;
}
```

C. 约数链

如果 n 个数 $a_1, \ldots, a_n$ 满足：对于 $1 \leq i < n$，a_i 是 a_{i+1} 的约数，那么 $a_1, \ldots, a_n$ 被称为一个约数链。一个约数链的权值是它所包含的 n 个数的乘积。

给出 n, m，只允许使用不超过 m 的正整数组成长为 n 的约数链，会有非常多的方式。求所有这些方式得到的约数链的权值之和。$n \leq 10, m \leq 10^7$

可以 **dp** 一下，设 $dp[n][k]$ 为长度为 n 的约数链，结尾是 k 的权值和。转移为刷表。复杂度做到 $\mathcal{O}(nm \log m)$。根本

过不去。

设 $f_n(k) = dp[n][k]$ 其中 函数 $f_k(x)$ 为积性函数。

对于 $a \perp b$ 设 $c = a \times b$ 对于 $f_n(c)$ 所统计的方案中的每一个数字都可以质因数分解成两组，一组是只属于 a 的质因子，另一组是只属于 b 的质因子。手举几个栗子就能发现符合积性函数的性质。

```
const int _ = 1e7 + 100;
const int MOD = 1e9 + 7;
int np[_], prime[int(1e6)], tot = 0, Mid[_];
void Init(int n){
    Mid[1] = 0; np[1] = 1;
    for(int i = 2; i <= n; i++){
        if(!np[i]) prime[++tot] = i, Mid[i] = i;
        for(int j = 1; j <= tot && prime[j] * i <= n; j++){
            int x = prime[j] * i;
            np[x] = 1; Mid[x] = prime[j];
            if(i % prime[j] == 0) break;
        }
    }

}
int n, m;
int dp[int(20)][int(30)]; int Ans[_];
long long t = 1;
int main(){ //freopen("in.txt", "r", stdin);
    Init(1e7 + 2); Read(n)(m);
    rep(S, 1, tot){
        int now = prime[S];
        memset(dp, 0, sizeof(dp));
        t = 1; for(int j = 0; t <= m; j++, t *= now) dp[1][j] = t;
        rep(i, 2, n){
            t = 1; for(int j = 0; t <= m; j++, t *= now) {
                int &ans = dp[i][j] = 0;
                rep(k, 0, j) ans = (ans +0ll+ dp[i - 1][k] *1ll* t % MOD) % MOD;
            }
        }
        t = 1; for(int j = 0; t <= m; j++, t *= now) Ans[t] = dp[n][j];
    }
    Ans[1] = 1;
    rep(i, 2, m){
        if(Ans[i]) continue;
        int A = i, B = 1;
        while(A % Mid[i] == 0) A /= Mid[i], B *= Mid[i];
        Ans[i] = (Ans[A] *1ll* Ans[B]) % MOD;
    }
    int ans = 0;
    rep(i, 1, m) ans = (ans +0ll+ Ans[i]) % MOD;
    printf("%d", ans);
```

```
    return 0;
}
```

D. 连通

给出一张无向图，求有多少种方案使得删除三条边，使原图不连通。

$n, m \leq 2000$

考虑枚举删除的第一条边是那条，然后问题转化成了 求在一张图中删除两条边使图不连通的方案数。先删去一条边，考虑剩下的图的形态，- 图已经不连通了，那答案的贡献就是每条边两两配对的方案数 - 图是联通的，考虑对图建一个生成树，考虑删边的方式 - 删除两条非树边 一定不合法，之前的树仍然存在，保证了图的连通性。- 删除一条树边，一条非树边 被计入答案，当且仅当，这条树边被这条非树边唯一覆盖。这里的正确性显然。- 删除两条非树边 被计入答案，当且仅当，这两条边被相同的一组非树边覆盖。- 证明可以考虑，对于原树来说，如果删除两条树边，断成三份，如果两条边被相同的一组非树边覆盖，没有边可以联通断开的中间部分。- 考虑如何实现，对每一条非树边赋一个随机权值，然后树边的权值定义为覆盖其的每一条非树边的权值异或和。统计删除两条边后图不联通的方案数，就转化为，统计有多少种方案选出两条边，使其权值相同。树上差分即可，异或有自反性，其标记在 LCA 处自动消失。

```
const int _ = 4100;
int head[_];
struct edges{
    int node;
    int nxt;
}edge[_]; int tot = 1;
const int MOD = 1e9 + 7;
int n, m;
int ToEid[_];
int ToNid[_];
int book[_];
u64 tag[int(2100)];
bool vis[int(2100)];
u64 EVal[int(2100)];
void add(int u, int v){
    tot++;
    edge[tot].node = v;
    edge[tot].nxt  = head[u];
    head[u]        = tot;
}
u64 gen(){
    u64 A = rand() *1ull* rand() * rand() *1ull* rand();
    u64 B = rand() *1ull* rand() * rand() *1ull* rand();
    return A + B + rand();
}
void dfs0(int now, int f){
    vis[now] = 1;
    for(int i = head[now]; i ; i = edge[i].nxt){
        int ex = edge[i].node; if(ex == f) continue; if(book[i]) continue;
        if(!vis[ex]) dfs0(ex, now);
```

```
        else {
            if( EVal[ToNid[i]]) continue;
            u64 g = EVal[ToNid[i]] = gen();
            tag[now] ^= g; tag[ex] ^= g;
        }
    }
}
void dfs1(int now, int f){
    u64 &ntag = tag[now];
    for(int i = head[now]; i ; i = edge[i].nxt){
        int ex = edge[i].node; if(ex == f) continue; if(book[i]) continue;
        if(!EVal[ToNid[i]]) dfs1(ex, now), EVal[ToNid[i]] = tag[ex], ntag ^= tag[ex];
    }
}
map<u64, int> M;
const int inv6 = 166666668;
int main(){// freopen("in.txt", "r", stdin);
    Read(n)(m); int *seed = new int; srand(*seed + n + m);
    rep(i, 1, m){
        int u, v; Read(u)(v);
        ToEid[i] = tot + 1; ToNid[tot + 1] = ToNid[tot + 2] = i;
        add(u, v); add(v, u);
    }
    int Pans = 0;
    rep(i, 1, m){
        int rs = ToEid[i];
        book[rs] = book[rs ^ 1] = 1;
        memset(tag, 0, sizeof(tag)); memset(vis, 0, sizeof(vis)); memset(EVal, 0, sizeof(EVal)); M.clea
        dfs0(1, 1);

        bool pass = true;
        rep(j, 1, n)  if(!vis[j]) { pass = false; break; }
        if(!pass) { int t = m - 1; Pans = (Pans +0ll+ (t * (t - 1))) % MOD; book[rs] = book[rs ^ 1] = 0
        dfs1(1, 1);
        rep(j, 1, m) if(i != j) M[EVal[j]] ++;
        int ans = 0;
        for(map<u64, int> :: iterator i = M.begin(); i != M.end(); i++){
            if(i->first == 0ull)
                ans = (ans +0ll+ (i->second *1ll* (i->second - 1)) % MOD) % MOD,
                ans = (ans +0ll+ (((m - 1) - (i->second)) *2ll* (i->second)) % MOD) % MOD;
            else
                ans = (ans +0ll+ (i->second *1ll* (i->second - 1)) % MOD) % MOD;
        }
        Pans = (ans +0ll+ Pans) % MOD;
        book[rs] = book[rs ^ 1] = 0;
    }
    printf("%d", int(Pans *1ll* inv6 % MOD));
```

```
    return 0;
}
```

Round 9

降智场。### A. 最大公约数

给出一个长度为 n 的序列。设 $k = \max_{i<j} \gcd(A_i, A_j)$，求出：- k - $\sum_{i<j} [\gcd(A_i, A_j) = k]$ - $\sum_{i<j} [\gcd(A_i, A_j) = k] A_i \times A_j$

枚举一下 gcd 可能以这个数字为 gcd 的两个数字一定是 gcd 的倍数。从大到小枚举 gcd 查询有多少倍数，如果大于 2，这个 gcd 就可以。剩余的两项在得到 k 的基础上判断即可。

```
const int MOD = 998244353;
const int _ = 2e6 + 100;
const int inv2 = 499122177;
int book[_];
int A[_], n, MAX = 0;
int K = 0;
int main(){
    Read(n); rep(i, 1, n) Read(A[i]), book[A[i]]++, MAX = max(MAX, A[i]);
    for(int i = MAX; i >= 1; i--){
        int cnt = 0;
        for(int j = i; j <= MAX; j += i) cnt += book[j];
        if(cnt >= 2) { K = i; break; }
    }
    int SUM = 0, ans1 = 0, ans2 = 0;
    for(int i = K; i <= MAX; i += K){
        if(!book[i]) continue;
        ans1 += book[i]; SUM = (SUM +0ll+ book[i] *1ll* i % MOD) % MOD;
    }
    ans1 = ((ans1 *1ll* (ans1 - 1)) % MOD *1ll* inv2) % MOD;
    for(int i = K; i <= MAX; i += K){
        if(!book[i]) continue;
        ans2 = (ans2 +0ll+ ( (SUM -0ll- i + MOD) % MOD *1ll* i % MOD *1ll* book[i]) % MOD ) % MOD;
    }
    printf("%d %d %d\n", K, ans1, ans2 *1ll* inv2 % MOD);
    return 0;
}
```

B. 紧急出口

在一个走廊里共有 N 个楼梯，其中 N 个楼梯之间有 $N-1$ 个间隔，个间隔会贴有一个紧急出口的指示牌指向左侧或者右侧。

问在所有的 $2N-1$ 种指示牌的贴法当中，有多少种贴法满足恰好有 M 个楼梯 (包括最左最右) 的左右侧的紧急出口标识都不指向自己。

输出的答案对 998244353 取模。$N \leq 10^3, M \leq 10^9$

倍增 **DP**　待填

隔板法　称“左右侧的紧急出口标识都不指向自己的点”为关键点。

考虑如果选定了一个点是关键点，那么他们两边就形如 `< o >` ，可以发现相邻的两个关键点 O_1, O_2 之间贴牌的方案一定形如 $... < O_1 > o > ... > o < ... < o < O_2 > ...$ 显然，其实是把原序列分成连续的 $2m$ 段，每段内方向相同，相邻两段方向相反。隔板即可。

考虑组合数的通项公式，项数只有 m 项。

```
int n, m; Read(n)(m); printf("%d", C(n, 2 * m - 1));
```

C. 合影

众所周知，ACM ICPC 是三人组队参赛的比赛。现在在 ICPC WF2020 现场，有 k 支参赛队伍要站成一排拍照！

摄影师小 Z 想促进队伍间的交流，于是小 Z 要求每个队伍的 3 个队员不能都挨在一起

现在小 Z 想问一问你，这 k 支队伍有多少种拍照方式呢？由于数字很大，所以对 998244353 取模就好 $\sim k \leq 10^6$

要求是每一队的三个队员不能全部挨在一起。无法直接全部满足，考虑容斥。

假如所有人按任意顺序排列有 $(3k)!$ 种排列方式。

显然会算入了不合法的情况，考虑去除这些 钦定哪一队全部挨在一起了，如果强制某一队挨在一起可以吧那一队捆成一个人 这样的方案数为 $(3k-2)!$，算入答案就是 $\binom{k}{1}(3k-2)!(3!)^1$ 最后一个是因为同一队中的队员也有顺序。

显然会去除多了，如果有多队同时不合法，这些方案会被去除多次，钦定哪两队全部挨在一起了，算入答案就是 $\binom{k}{2}(3k-4)!(3!)^2$。

显然加多了……

然后就是容斥原理了。

答案应该是 $\sum_{i=0}^{k}(-1)^i\binom{k}{i}(3k-2i)!(3!)^i$

```
int C(int n, int m){ return frac[n] *1ll* ifrac[n - m] % MOD *1ll* ifrac[m] % MOD; }
int main(){
    int k; Read(k); Init((k + 1) * 3);
    int ans = 0;
    rep(i, 0, k){
        int tans = 0;
        tans = C(k, i) *1ll* frac[3 * k - 2 * i] % MOD *1ll* pow(6, i, MOD) % MOD;
        tans = (i & 1) ? MOD - tans : tans;
        ans = (ans +0ll+ tans) % MOD;
    }
    printf("%d", ans);
    return 0;
}
```

D. 路径

给定一个 n 个顶点的无根树，顶点编号 1 n。

同时给定树上的 k 条带权路径，要求支持以下操作:

- `1 k` ，删除第 k 条带权路径。
- `2 k v` ，将第 k 条带权路径的权值修改为 v 。

- 3 p ，给定一个树上节点 p ，询问所有没有被删去的带权路径中，不与该节点相交的所有路径中的最小权值。$n, k, q \leq 10^5$

做法一：树剖 + **std::set** 维护标记 标记永久化线段树 $\mathcal{O}((k+q)\log_2^3 n)$ 先树剖，路径补集仍然是线段树上 $\log_2(n)$ 段，可以在这些段里面加入可选的边，为了防止删除重复，std::set 内层数据类型为 pair<int, int> 分别存储路径的权值和编号。然后就直接做就好了。后记：被出题人卡成了 30 ……。

```
#define rep(i, l, r) for(int i = l, __ = r; i <= __; i++)
using namespace std;
int n, k, p;
const int _ = 2e6 + 100;
int head[_];
struct edges{
    int node;
    int nxt;
}edge[_]; int tot = 0;
void add(int u, int v) {
    tot++;
    edge[tot].node = v;
    edge[tot].nxt  = head[u];
    head[u]        = tot;
}
struct Path{ int u, v, Val; Path(int a, int b, int c) { u = a; v = b; Val = c; } Path(){} } P[_];

int dfn[_], rnk[_], dfc, dep[_], top[_], fa[_], si[_], son[_];
void dfs0(int now, int f, int dp){
    fa[now] = f;  dep[now] = dp; int &Mid = son[now] = 0; int &Si = si[now] = 1;
    for(int i = head[now]; i ; i = edge[i].nxt){
        int ex = edge[i].node; if(ex == f) continue;
        dfs0(ex, now, dp + 1); Si += si[ex]; if(si[ex] > si[Mid]) Mid = ex;
    }
}
void dfs1(int now, int f, int tp){
    dfn[now] = ++dfc;  top[now] = tp;
    if(son[now]) dfs1(son[now], now, tp);
    for(int i = head[now]; i ; i = edge[i].nxt){
        int ex = edge[i].node; if(ex == f || ex == son[now]) continue;
        dfs1(ex, now, ex);
    }
}

namespace SegmentTree{
    const int _ = 3e6 + 100;
    int ch[_][2]; set<pair<int ,int>, less <pair<int ,int> > > v[_];
    int tot = 0;
    #define ls(o) (ch[o][0])
    #define rs(o) (ch[o][1])
    #define make (tot++, ch[tot][0] = ch[tot][1] = 0, v[tot].clear(), tot)
```

```
    int Groot(){ return make; }
    void build(int o, int L, int R){
        if(L == R) return (void)v[o].insert(make_pair(INT_MAX, -1));
        int mid = (L + R) >> 1;
        ls(o) = make; rs(o) = make; v[o].insert(make_pair(INT_MAX, -1));
        build(ls(o), L, mid); build(rs(o), mid + 1, R);
    }
    void update_add(int o, int nowl, int nowr, int L, int R, int V, int id){
        if(L > R) return ;
        if(L <= nowl && nowr <= R) return (void)v[o].insert(make_pair(V, id));
        int mid = (nowl + nowr) >> 1;
        if(L <= mid) update_add(ls(o), nowl, mid, L, R, V, id);
        if(R  > mid) update_add(rs(o), mid + 1, nowr, L, R, V, id);
    }
    void update_del(int o, int nowl, int nowr, int L, int R, int V, int id){
        if(L > R) return ;
        if(L <= nowl && nowr <= R) return (void)v[o].erase(make_pair(V, id));
        int mid = (nowl + nowr) >> 1;
        if(L <= mid) update_del(ls(o), nowl, mid, L, R, V, id);
        if(R  > mid) update_del(rs(o), mid + 1, nowr, L, R, V, id);
    }
    pair<int, int > query(int o, int nowl, int nowr, int p, pair<int, int > ans = make_pair(INT_MAX, -1
        if(nowl == nowr) return min(ans, *(v[o].begin()));
        int mid = (nowl + nowr) >> 1; ans = min(ans, *(v[o].begin()));
        return p <= mid ? query(ls(o), nowl, mid, p, ans) : query(rs(o), mid + 1, nowr, p, ans);
    }
} using SegmentTree::query;     using SegmentTree::update_add; using SegmentTree::Groot;
  using SegmentTree::build;     using SegmentTree::update_del;
#define fi first
#define se second
vector<pair<int, int > > TMP;
int root = 0;
void AddWithoutPath(int u, int v, int Val, int id){
    TMP.clear();
    while(top[u] != top[v]){
        if(dep[top[u]] < dep[top[v]]) swap(u, v);
        TMP.push_back(make_pair(dfn[top[u]], dfn[u])); u = fa[top[u]];
    }
    if(dep[u] < dep[v]) swap(u, v);
    TMP.push_back(make_pair(dfn[v], dfn[u]));
    sort(TMP.begin(), TMP.end());
    rep(i, 0, TMP.size() - 2) update_add(root, 1, n, TMP[i].se + 1, TMP[i + 1].fi - 1, Val, id);
    update_add(root, 1, n, 1, TMP[0].fi - 1, Val, id);
    update_add(root, 1, n, TMP[TMP.size() - 1].se + 1, n, Val, id);
}
void DelWithoutPath(int u, int v, int Val, int id){
    TMP.clear();
```

```
    while(top[u] != top[v]){
        if(dep[top[u]] < dep[top[v]]) swap(u, v);
        TMP.push_back(make_pair(dfn[top[u]], dfn[u])); u = fa[top[u]];
    }
    if(dep[u] < dep[v]) swap(u, v);
    TMP.push_back(make_pair(dfn[v], dfn[u]));
    sort(TMP.begin(), TMP.end());
    rep(i, 0, TMP.size() - 2) update_del(root, 1, n, TMP[i].se + 1, TMP[i + 1].fi - 1, Val, id);
    update_del(root, 1, n, 1, TMP[0].fi - 1, Val, id);
    update_del(root, 1, n, TMP[TMP.size() - 1].se + 1, n, Val, id);
}
int main(){ // freopen("in.txt", "r", stdin);
    Read(n)(k)(p);
    rep(i, 1, n - 1){
        int u, v; Read(u)(v);
        add(u, v); add(v, u);
    }
    rep(i, 1, k) {
        int p, q, v; Read(p)(q)(v);
        P[i] = Path(p, q, v);
    }
    dfs0(1, 1, 1); dfs1(1, 1, 1);
    root = Groot(); build(root, 1, n);
    rep(i, 1, k) AddWithoutPath(P[i].u, P[i].v, P[i].Val, i);
    while(p--){
        int opt, k; Read(opt)(k);
        if(opt == 1){
            DelWithoutPath(P[k].u, P[k].v, P[k].Val, k);
        } else if(opt == 2){
            DelWithoutPath(P[k].u, P[k].v, P[k].Val, k);
            Read(P[k].Val);
            AddWithoutPath(P[k].u, P[k].v, P[k].Val, k);
        } else {
            pair<int, int > r = query(root, 1, n, dfn[k]);
            printf("%d\n", r.first < INT_MAX ? r.first : -1);
        }
    }
    return 0;
}
```

做法二：线段树维护路径交集 + 二分最小值

巧妙绝伦的做法。

要求回答的是：不与节点 p 相交的所有路径中的最小权值。

考虑二分一个最小权值 x，然后判断是所有权值小于 x 的路径是不是全都和点 p 有交。

如果把所有路径按照权值排序，那么成为答案的那条路径之前的路径一定都和点 p 有交，这就可以直接二分答案了。

但是路径需要支持删除和修改。显然需要数据结构。

一个显然的事实是：两条路径的交 也是一条路径（定义一个点或者空都属于路径）。

线段树就可以快速求出一段路径区间内所有路径的交集路径。

所有路径按照权值排好序后的一个前缀全部和点 p 有交，等价于，这个前缀所有的路径的交集和点 p 有交。

对于删除操作，可以考虑在线段树对应操作上打删除标记。对于修改操作，因为没有强制在线，可以考虑一开始把这条路径的所有权值排好序后加入线段树，然后对于当前无用的版本，打上删除标记即可。对于一次修改，转化成一次删除和一次恢复操作。

当然平衡树可以轻易完成这样的工作，在平衡树上维护树上路径交集，写不出来啊…

这样就可以直接用线段树求出某个前缀的所有路径的交集路径了。线段树合并信息时，需要询问 LCA，算上二分，这样的单次询问复杂度可以做到 $\mathcal{O}(\log k \log w \log n)$（倍增 LCA），或者 $\mathcal{O}(\log k \log w)$（$\mathcal{O}(n)-\mathcal{O}(1)$ LCA）。

注意到线段树本身就是一个分治的结构，可以直接在线段树上二分，这样就可以做到 $\mathcal{O}(\log k \log n)$（倍增 LCA），或者 $\mathcal{O}(\log k)$（$\mathcal{O}(n)-\mathcal{O}(1)$ LCA）。

我的实现是 倍增 LCA + 线段树二分，算上预处理，总时间复杂度为 $\mathcal{O}(n\log(n)+q\log(q+k)\log n)$

```
const int _ = 2e5 + 100;
int head[_], tot = 0;
struct edges{
    int node;
    int nxt;
}edge[_];
void add(int u, int v){
    tot++;
    edge[tot].node = v;
    edge[tot].nxt  = head[u];
    head[u]        = tot;
}
int n, k, q;
const int LOG = 18;
int fa[_][LOG + 2], dep[_];
void dfs0(int now, int f, int dp){
    fa[now][0] = f; dep[now] = dp;
    for(int i = head[now]; i ; i = edge[i].nxt){
        int ex = edge[i].node; if(ex == f) continue;
        dfs0(ex, now, dp + 1);
    }
}
map < pair<int, int> , int >  Mem;
int LCA(int u, int v){
    pair<int, int> P = make_pair(min(u, v), max(u, v));
    if(Mem.count(P)) return Mem[P];
    if(dep[u] < dep[v]) swap(u, v);
    int d = dep[u] - dep[v];
    int x = u, y = v;
    rep(i, 0, LOG) if(d & (1 << i)) x = fa[x][i];
```

```
    if(x == y) return x;
    for(int i = LOG; i >= 0; i--){
        if(fa[x][i] == fa[y][i]) continue;
        x = fa[x][i]; y = fa[y][i];
    }
    return Mem[P] = fa[x][0];
}

void InitQuery(){
    dfs0(1, 1, 1);
    rep(j, 1, LOG) rep(i, 1, n) fa[i][j] = fa[fa[i][j - 1]][j - 1];
}
struct Path_t{
    int id, u, v, w, ver;
    Path_t(int a, int b, int c, int d, int e) { id = a; u = b; v = c; w = d; ver = e; }
    Path_t() { }
};
bool CMP(const Path_t & A, const Path_t & B) { return A.w < B.w; }
struct Q_t{
    int type, a, b;
    Q_t(int x, int y, int z) { type = x; a = y; b = z; }
    Q_t() {}
}Q[_];
int nowv[_];
vector<Path_t> P;
vector<int> Pv[_];
int dis(int u, int v) { return dep[u] + dep[v] - (dep[LCA(u, v)] << 1); }
pair<int, int> cross(int u0, int v0, int u1, int v1) {
    int t0 = LCA(u0, v0);
    int t1 = LCA(u0, u1);
    int t2 = LCA(u0, v1);
    int t3 = LCA(v0, u1);
    int t4 = LCA(v0, v1);
    int target0 = t0; // t0 t1 t3
    if(dep[target0] < dep[t1]) target0 = t1;
    if(dep[target0] < dep[t3]) target0 = t3;
    int target1 = t0; // t0 t2 t4
    if(dep[target1] < dep[t2]) target1 = t2;
    if(dep[target1] < dep[t4]) target1 = t4;
    if(target1 == target0 && dis(target1, u1) + dis(target1, v1) != dis(u1, v1)) return make_pair(-1, -
    else return make_pair(target0, target1);
}
bool crossx(int u, int v, int x) { if(u == -1 || v == -1) return false;  return dis(u, v) == dis(u, x)
namespace SegmentTree{ // To maintain the merge path
    struct Node_t{
        int u, v;
        bool isDel, isNul;
```

```
    void clear(){
        isDel = false; isNul = true; u = v = 0;
    }
};
const int _ = 1e6 + 100;
int ch[_][2], tot = 0; Node_t v[_];
#define ls(o) (ch[o][0])
#define rs(o) (ch[o][1])
#define make (tot++, ch[tot][0] = ch[tot][1] = 0, v[tot].clear(), tot)
int Groot() { return make; }
void maintain(int o) {
    v[o].isDel = (v[ls(o)].isDel && v[rs(o)].isDel);
    v[o].isNul = (v[ls(o)].isNul || v[rs(o)].isNul);
    if(!v[o].isDel && !v[o].isNul){
        if(v[ls(o)].isDel && !v[rs(o)].isDel) v[o].u = v[rs(o)].u, v[o].v = v[rs(o)].v;
        else if(v[rs(o)].isDel && !v[ls(o)].isDel) v[o].u = v[ls(o)].u, v[o].v = v[ls(o)].v;
        else {
            pair<int, int> R = cross(v[ls(o)].u, v[ls(o)].v, v[rs(o)].u, v[rs(o)].v);
            if(R.first == -1) {
                v[o].isNul = true;
            }
            v[o].u = R.first; v[o].v = R.second;
        }
    }
}
void build(int o, int L, int R){
    if(L == R) {
        v[o].isNul = false;
        v[o].isDel = (P[L - 1].ver != 1);
        v[o].u     = P[L - 1].u;
        v[o].v     = P[L - 1].v;
        return ;
    } ls(o) = make; rs(o) = make;
    int mid = (L + R) >> 1;
    build(ls(o), L, mid); build(rs(o), mid + 1, R);
    maintain(o);
}
void update(int o, int nowl, int nowr, int p, bool V){
    if(nowl == nowr) return (void)(v[o].isDel = V);
    int mid = (nowl + nowr) >> 1;
    if(p <= mid) update(ls(o), nowl, mid, p, V);
    if(p  > mid) update(rs(o), mid + 1, nowr, p, V);
    maintain(o);
}
int query(int o, int nowl, int nowr, int x){
    if(!v[o].isNul && !v[o].isDel && crossx(v[o].u, v[o].v, x)) return -1;
    if(nowl == nowr) return P[nowl - 1].w;
```

```
        bool r0 = !(v[ls(o)].isNul || !crossx(v[ls(o)].u, v[ls(o)].v, x)) || v[ls(o)].isDel;
        bool r1 = !(v[rs(o)].isNul || !crossx(v[rs(o)].u, v[rs(o)].v, x)) || v[rs(o)].isDel;
        int mid = (nowl + nowr) >> 1;
        if(!r0) return query(ls(o), nowl, mid, x); else if(!r1) return query(rs(o), mid + 1, nowr, x);
    }
} using SegmentTree::build; using SegmentTree::Groot;
  using SegmentTree::query; using SegmentTree::update;

int main() { freopen("in.txt", "r", stdin);
    Read(n)(k)(q); rep(i, 1, n - 1) { int u, v; Read(u)(v); add(u, v); add(v, u); } InitQuery();
    rep(i, 1, k){
        int u, v, w; Read(u)(v)(w);
        P.push_back(Path_t(i, u, v, w, 1)); nowv[i] = 1;
    }
    rep(i, 1, q){
        int type, a, b; Read(type)(a);
        if(type == 2) Read(b), Q[i] = Q_t(type, a, b);
        else Q[i] = Q_t(type, a, 0);
    }
    rep(i, 1, q){
        Q_t &now = Q[i];
        if(now.type == 2) P.push_back(Path_t(now.a, P[now.a - 1].u, P[now.a - 1].v, now.b, ++nowv[now.a
    }

    rep(i, 1, k) Pv[i].resize(nowv[i] + 2);
    sort(P.begin(), P.end(), CMP);
    rep(i, 0, P.size() - 1) Pv[P[i].id][P[i].ver] = i + 1;
    int root = Groot(), U; U = P.size(); build(root, 1, U);
    rep(i, 1, k) nowv[i] = 1;
    rep(i, 1, q){
        Q_t &now = Q[i];
        if(now.type == 1){
            int Pl = Pv[now.a][nowv[now.a]];
            update(root, 1, U, Pl, 1);
        } else if(now.type == 2){
            int Pl0 = Pv[now.a][  nowv[now.a]];
            int Pl1 = Pv[now.a][++nowv[now.a]];
            update(root, 1, U, Pl0, 1);
            update(root, 1, U, Pl1, 0);
        } else {
            printf("%d\n", query(root, 1, U, now.a));
        }
    }
    return 0;
}
```

做法三：**CDQ** 分治　学了再说。

QAQ /kel