

I DP

1.1 ArrayPartitionDP

```

/// dp[n][k] = min(dp[i-1][k-1] + cost(i, n))
/// using aliens trick, cost() is QF. O(n log^2 n)
namespace ArrayPartitionDP {
LL base_cost(int l, int r); ///define in code
long long C; int n;
PLL operator+ (const PLL &a, const PLL &b) {
    return PLL(a.first+b.first, a.second+b.second);
}
///Solves dp[i]=min(dp[j]+cost(j+1,i)), QF cost()
///returns {dp[n], min no of partitions}
PLL solve1D() {
    auto cost = [&](int l, int r) {
        return PLL(base_cost(l, r)+C, 1);
    };
    vector<PLL> dp(n+1); vector<int> opt(n+1);
    deque<pair<int, int>> dq; dq.push_back({0, 1});
    dp[0] = {0, 0};
    for (int i=1; i<=n; i++) {
        opt[i] = dq.front().first;
        dp[i] = dp[opt[i]] + cost(opt[i]+1, i);
        if (i == n) break;
        dq[0].second++;
        if (dq.size()>1 && dq[0].second==dq[1].second)
            dq.pop_front();
        int en = n;
        while(dq.size()) {
            int o=dq.back().first, st=dq.back().second;
            if (dp[o]+cost(o+1,st)>=dp[i]+cost(i+1,st))
                dq.pop_back();
            else {
                int lo = st, hi = en;
                while (lo < hi) {
                    int mid = (lo+hi+1)/2;
                    if (dp[o]+cost(o+1, mid) <
                        dp[i]+cost(i+1, mid)) lo = mid;
                    else hi = mid-1;
                }
                if (lo < n) dq.push_back({i, lo+1});
                break;
            }
            en = st-1;
        }
        if (dq.empty()) dq.push_back({i, i+1});
    }
    return dp[n];
}
PLL check(LL c) { C = c; return solve1D(); }
LL solve(int N, int k, LL lo, LL hi) {
    n = N;
    while (lo < hi) {
        LL mid = lo + (hi-lo)/2;
        if (check(mid).second > k) lo = mid+1;
        else hi = mid;
    }
    return check(lo).first - 1LL*k*lo;
}
}

```

1.2 SlopeTrick

```

typedef long long LL;
template< typename T >
struct SlopeTrick {

```

```

    T INF = numeric_limits< T >::max() / 3;
    T min_f, add_l, add_r;
    priority_queue< T, vector< T >, less<> > L;
    priority_queue< T, vector< T >, greater<> > R;
    void push_R(const T &a) { R.push(a + add_r); }
    T top_R() const { return R.top() + add_r; }
    T pop_R() { T val=top_R(); R.pop(); return val; }
    void push_L(const T &a) { L.push(a - add_l); }
    T top_L() const { return L.top() + add_l; }
    T pop_L() { T val=top_L(); L.pop(); return val; }
public:
    SlopeTrick() : min_f(0), add_l(0), add_r(0) {
        L.push(-INF); R.push(INF);
    }
    // f(x) += a
    void add_all(const T &a) {
        min_f += a;
    }
    // add \_ ; f(x) += max(a - x, 0)
    void add_a_minus_x(const T &a) {
        if (a > top_R()) {
            min_f+=a-top_R(); push_L(pop_R());push_R(a);
        } else push_L(a);
    }
    // add _/ ; f(x) += max(x - a, 0)
    void add_x_minus_a(const T &a) {
        if (top_L() > a) {
            min_f+=top_L()-a;push_R(pop_L());push_L(a);
        } else push_R(a);
    }
    // add \\\ ; f(x) += max(x - a, 0)
    void add_abs(const T &a) {
        add_a_minus_x(a); add_x_minus_a(a);
    }
    // \\\ -> \_ ; f_{new}(x) = min f(y) (y <= x)
    void clear_right(){ while(R.size()>=2)R.pop(); }
    // \\\ -> _/ ; f_{new}(x) = min f(y) (y >= x)
    void clear_left(){ while(L.size()>=2) L.pop(); }
    // \\\ -> .\\ ; f_{new}(x) = f(x - a)
    void shift(const T &a) { add_l+=a; add_r+=a; }
    T get(const T &x) {
        T ret = min_f;
        if (!L.empty() && x < top_L()) {
            while(!L.empty())ret += max(T(0),pop_L()-x);
        }
        if (!R.empty() && top_R() < x) {
            while(!R.empty())ret += max(T(0),x-pop_R());
        }
        return ret;
    }
};
void SmallToLarge(SlopeTrick<LL> &from,
    SlopeTrick<LL>&to) {
    if (from.L.size()+from.R.size()>
        to.L.size()+to.R.size()) swap(from, to);
    while (from.L.size() >= 2) {
        to.add_a_minus_x(from.pop_L());
    }
    while (from.R.size() >= 2) {
        to.add_x_minus_a(from.pop_R());
    }
    to.min_f += from.min_f;
}

```

```

}
const int MAXN = 3e5+7; int P[MAXN], C[MAXN];
vector<int> child[MAXN]; SlopeTrick<LL> trick[MAXN];
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n, m; cin >> n >> m;
    for (int i = 2; i <= n+m; i++) {
        cin >> P[i] >> C[i]; child[P[i]].push_back(i);
    }
    for (int i = n+1; i <= n+m; i++) {
        trick[i].add_abs(C[i]);
    }
    for (int i = n; i > 0; i--) {
        for (int c : child[i]) {
            SmallToLarge(trick[c], trick[i]);
        }
        { ///clearing all slopes greater than 1
            LL save = trick[i].top_R();
            trick[i].clear_right();
            trick[i].push_R(save);
        }
        if (i > 1) {
            trick[i].push_L(trick[i].pop_L()+C[i]);
            trick[i].push_R(trick[i].pop_R()+C[i]);
        }
    }
    cout << trick[1].min_f << "\n";
    return 0;
}

```

1.3 dnc

```

///dp[j][i] = min(dp[j-1][k-1] + C[k][i]) [k<=i]
///C(a,c)+C(b,d) <= C(a,d)+C(b,c) [a<=b<=c<=d]
ll dp[kmax][nmax];
void dnc(int K,int L,int R,int OptL,int OptR){
    if(L > R) return; int mid = (L + R) / 2;
    int optNow = -1; dp[K][mid] = inf;
    for(int i=OptL; i<=min(OptR, mid); i++){
        ll tmp = dp[K-1][i-1] + cost(i, mid);
        if(tmp <= dp[K][mid])
            dp[K][mid] = tmp, optNow = i;
    }
    dnc(K, L, mid - 1, OptL, optNow);
    dnc(K, mid + 1, R, optNow, OptR);
}

```

1.4 knuth

```

//Opt[i-1][j] <= Opt[i][j] <= Opt[i][j+1]
for (int len = 2; len<=n; len++){
    for (int l=0; l+len<=n; l++){
        int r=l+len; dp[l][r] = INF;
        for(int i=opt[l][r-1]; i<=opt[l+1][r]; i++){
            LL cost = dp[l][i] + dp[i][r] + C(l, r);
            if (cost < dp[l][r])
                dp[l][r] = cost, opt[l][r] = i;
        }
    }
}

```

2 DS

2.1 2dbit

```

11 bit[4][mx][my];
void update( int x, int y, int val, int i ) {
    int y1;
    while( x<=mx ) {
        y1=y;
        while( y1<=my )
            bit[i][x][y1] += val, y1 += (y1&-y1);
        x += (x&-x);
    }
}
11 query( int x, int y, int i ) {
    11 ans=0; int y1;
    while( x>0 ) {
        y1 = y;
        while( y1>0 )
            ans += bit[i][x][y1], y1 -= (y1&-y1);
        x -= (x&-x);
    }
    return ans;
}
//add value k from (x1,y1) to (x2,y2) inclusive
void add(int x1,int y1, int x2, int y2, int k){
    update(x1,y1,k,0); update(x1,y2+1,-k,0);
    update(x2+1,y1,-k,0); update(x2+1,y2+1,k,0);
    update(x1,y1,k*(1-y1),1);
    update(x1,y2+1,k*y2,1);
    update(x2+1,y1,k*(y1-1),1);
    update(x2+1,y2+1,-y2*k,1);
    update(x1,y1,k*(1-x1),2);
    update(x1,y2+1,k*(x1-1),2);
    update(x2+1,y1,k*x2,2);
    update(x2+1,y2+1,-x2*k,2);
    update(x1,y1,(x1-1)*(y1-1)*k,3);
    update(x1,y2+1,-y2*(x1-1)*k,3);
    update(x2+1,y1,-x2*(y1-1)*k,3);
    update(x2+1,y2+1,x2*y2*k,3);
}
// get value from (x1,y1) to (x2,y2) inclusive
long long get( int x1, int y1, int x2, int y2 ){
    11 v1=query(x2,y2,0)*x2*y2+query(x2,y2,1)*x2+
        query(x2,y2,2)*y2+ query(x2,y2,3);
    11 v2=query(x2,y1-1,0)*x2*(y1-1)+
        query(x2,y1-1,1)*x2+
        query(x2,y1-1,2)*(y1-1)+
        query(x2,y1-1,3);
    11 v3=query(x1-1,y2,0)*(x1-1)*y2+
        query(x1-1,y2,1)*(x1-1)+
        query(x1-1,y2,2)*y2+ query(x1-1,y2,3);
    11 v4=query(x1-1,y1-1,0)*(x1-1)*(y1-1)+
        query(x1-1,y1-1,1)*(x1-1)+
        query(x1-1,y1-1,2)*(y1-1)+
        query(x1-1,y1-1,3);
    return v1-v2-v3+v4;
}

```

2.2 CHTDynamic

```

//CHT for max, for min negate insert and result
typedef long long ll;
struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const

```

```

        { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
};
struct CHT: multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0, div(a,b) = a/b)
    static const ll inf = LLONG_MAX;
    ll div(ll a, ll b) { // floored division
        return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m?inf:-inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y))
            isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        assert(!empty()); auto l = *lower_bound(x);
        return l.k * x + l.m;
    }
}

```

2.3 CHTLinear

```

// Minimum:
// M inc, x dec, useless(s-1, s-2, s-3)
// M dec, x inc, useless(s-3, s-2, s-1)
// Maximum:
// M inc, x inc, useless(s-3, s-2, s-1)
// M dec, x dec, useless(s-1, s-2, s-3)
// If queries are not in order, use query2 O(logn).
typedef long long LL;
struct CHT {
    vector<LL> M; vector<LL> C; int ptr = 0;
    //Use double comp if M,C is LL range
    bool useless(int l1, int l2, int l3) {
        return (C[l3]-C[l1])*(M[l1]-M[l2])
            <= (C[l2]-C[l1])*(M[l1]-M[l3]); }
    LL f(int id, LL x) { return M[id]*x+C[id]; }
    void add(LL m, LL c) {
        M.push_back(m); C.push_back(c);
        int s = M.size();
        while (s >= 3 && useless(s-3, s-2, s-1)) {
            M.erase(M.end()-2); C.erase(C.end()-2); s--;
        }
    }
    LL query(LL x) {
        if (ptr >= M.size()) ptr = M.size()-1;
        while (ptr < M.size()-1 && f(ptr, x)
            > f(ptr+1, x)) ptr++; // > to < for max
        return f(ptr, x);
    }
}
LL query2(LL x) {
    int lo=0, hi=M.size()-1;
    while(lo<hi) {
        int mid = (lo+hi)/2;
        // change > to < for maximum

```

```

        if (f(mid, x) > f(mid+1, x)) lo = mid+1;
        else hi = mid;
    } return f(lo, x);
};

```

2.4 Fenwick

```

struct Fenwick {
    int N, K = 20; vector<long long> ft;
    Fenwick(int n) : N(n+1), ft(n+1) {}
    void add(int x, long long val) {
        for (int i=x; i<N; i+=i&-i) ft[i] += val;
    }
    long long sum(int x) {
        long long ans = 0;
        for (int i=x; i>0; i-=i&-i) ans += ft[i];
        return ans;
    }
    //first k st sum(k)>=x, if none returns N=n+1.
    int get(long long x) {
        int ans = 0;
        for (int i=K-1; i>=0; i--) {
            int nxt = ans + (1<<i);
            if (nxt < N && ft[nxt] < x) {
                ans = nxt; x -= ft[nxt];
            }
        }
        return ans+1; }
};

```

2.5 HLD

```

/** flat[] (0-indexed) has the flattened array
flatIdx[] is the reverse map of flat[]. Everything
other than dfs(u, p) & HLD(u, p) are auxiliary */
const int MAXN = 500007; const int LOGN = 20;
vector<int> edg[MAXN];
int sbtr[MAXN], lvl[MAXN], pr[MAXN][LOGN];
int chainIdx[MAXN], chainHead[MAXN], flatIdx[MAXN];
int chainCnt, flatCnt, flat[MAXN];
void dfs(int u, int p) {
    lvl[u] = lvl[p] + 1; pr[u][0] = p;
    for (int k = 1; k < LOGN; k++) {
        pr[u][k] = pr[pr[u][k-1]][k-1];
    }
    sbtr[u] = 1;
    for (int v : edg[u]) {
        if (v==p) continue;
        dfs(v, u); sbtr[u] += sbtr[v];
    }
}
// auxiliary function
int getLCA(int u, int v) {
    if (lvl[u] < lvl[v]) swap(u, v);
    for (int k = LOGN-1; k >= 0; k--) {
        if (lvl[u]-(1<<k) >= lvl[v]) u = pr[u][k];
    }
    if (u==v) return u;
    for (int k = LOGN-1; k >= 0; k--) {
        if (pr[u][k] != pr[v][k]) {
            u = pr[u][k]; v = pr[v][k];
        }
    }
    return pr[u][0];
}

```

```

}
void HLD(int u, int p) {
    chainIdx[u] = chainCnt; flatIdx[u] = flatCnt;
    flat[flatCnt] = u; flatCnt++;
    int biggie = -1, mx = 0;
    for (int v : edg[u]) {
        if (v==p) continue;
        if (mx < sbtr[v]) {
            mx = sbtr[v]; biggie = v;
        }
    }
    if (biggie== -1) return;
    HLD(biggie, u);
    for (int v : edg[u]) {
        if (v==p || v==biggie) continue;
        chainCnt++; chainHead[chainCnt]=v; HLD(v, u);
    }
}
/// upSeg(l,u,vp) add sgmnts for (l, u] to vp vctr
/// provided l is an ancestor of u
void upSegments(int l, int u, vector<PII>&vp) {
    while (chainIdx[l] != chainIdx[u]) {
        int uhead = chainHead[chainIdx[u]];
        vp.push_back(PII(flatIdx[uhead], flatIdx[u]));
        u = pr[uhead][0];
    }
    if (l!=u) {
        vp.push_back(PII(flatIdx[l]+1, flatIdx[u]));
    }
}
vector<PII>getChainSegments(int u, int v) {
    int l = getLCA(u, v); vector<PII>rt;
    rt.push_back(PII(flatIdx[l], flatIdx[l]));
    if (u==v) return rt;
    upSegments(l, u, rt); upSegments(l, v, rt);
    return rt;
}
PII getSubtreeSegment(int u) {
    return PII(flatIdx[u], flatIdx[u]+sbtr[u]-1);
}
void performHLD(int root) { ///CALL THIS
    dfs(root, 0); chainCnt = 0; flatCnt = 0;
    chainHead[0] = root; HLD(root, 0);
}

```

2.6 ImplicitTreap

```

typedef long long LL;
mt19937 rng(chrono::steady_clock::now().
    time_since_epoch().count());
typedef struct item * pitem;
struct item {
    int prior, value, cnt; LL sum; bool rev;
    item(int value):prior(rng()), value(value) {
        cnt = 0; rev = 0; sum = value; l = r = nullptr;
    }
    pitem l, r;
};
/// 0-indexed
namespace Treap {
    int cnt(pitem it) { return it!=nullptr?it->cnt:0; }
    LL sum(pitem it) { return it!=nullptr?it->sum:0; }
    void upd_cnt (pitem it) {

```

```

        if (it!=nullptr) {
            it->cnt=cnt(it->l)+cnt(it->r)+1;
            it->sum=sum(it->l)+sum(it->r)+it->value;
        }
    }
    void push (pitem it) {
        if (it!=nullptr && it->rev==true){
            it->rev = false; swap (it->l, it->r);
            if (it->l) it->l->rev ^= true;
            if (it->r) it->r->rev ^= true;
        }
    }
    void merge (pitem &t, pitem l, pitem r) {
        push (l); push (r);
        if (l==nullptr || r==nullptr)
            t = (l!=nullptr) ? l : r;
        else if (l->prior > r->prior)
            merge (l->r, l->r, r); t = l;
        else merge (r->l, l, r->l), t = r;
        upd_cnt (t);
    }
    /// r will have key-th and above
    void split(pitem t, pitem &l, pitem &r, int key,
        int add = 0) {
        if (t==nullptr) { l = r = nullptr; return; }
        push(t); int cur_key = add + cnt(t->l);
        if (key <= cur_key)
            split (t->l, l, t->l, key, add), r = t;
        else
            split(t->r, t->r, r, key, add+1+cnt(t->l)), l=t;
        upd_cnt (t);
    }
    void reverse (pitem &t, int l, int r) {
        pitem t1, t2, t3; split (t, t1, t2, l);
        split (t2, t2, t3, r-l+1); assert(t2 != NULL);
        t2->rev ^= true; merge(t,t1,t2); merge(t,t,t3);
    }
    LL query(pitem &t, int l, int r) {
        pitem t1, t2, t3; split(t, t1, t2, l);
        split(t2, t2, t3, r-l+1); LL ans = t2->sum;
        merge(t,t1,t2); merge(t,t,t3); return ans;
    }
    /// before key'th element
    void insert (pitem &t, int key, int value) {
        pitem x = new item(value); pitem L, R;
        split(t, L, R, key); merge(L, L, x);
        merge(t, L, R); upd_cnt(t);
    }
    int erase (pitem &t, int key) {
        assert(cnt(t) > key); pitem L, MID, R;
        split(t, L, MID, key); split(MID, MID, R, 1);
        merge(t, L, R); upd_cnt(t);
        int rt = MID->value; delete MID; return rt;
    }
    void output (pitem t, vector< int >&v) {
        if (t==nullptr) return;
        push (t); output (t->l, v);
        v.push_back(t->value); output (t->r, v);
    }
}
int main() {

```

```

    int n, q, m; cin >> n >> q >> m;
    pitem tr = nullptr;
    for (int i = 0; i < n; i++) {
        int x; cin >> x;
        Treap::insert(tr, i, x);
    }
    while (q--) {
        int t, l, r;
        cin >> t >> l >> r; l--; r--;
        if (t==1) {
            int x = Treap::erase(tr, r);
            Treap::insert(tr, l, x);
        } else { Treap::reverse(tr, l, r); }
    }
    vector< int >v; Treap::output(tr, v);
    while (m--) {
        int i; cin >> i;
        cout << v[i-1] << " ";
    }
}

```

2.7 LCA

```

const int N = 3e5+7, K = 20; vector<int> adj[N];
int anc[N][K]; int level[N];
void setup(int u, int par) {
    level[u] = level[par]+1;
    anc[u][0] = par;
    for (int k=1; k<K; k++)
        anc[u][k] = anc[anc[u][k-1]][k-1];
    for (int v: adj[u]) {
        if (v == par) continue;
        setup(v, u);
    }
}
int lca(int u, int v) {
    if (level[u] > level[v]) swap(u, v);
    for (int k=K-1; k>=0; k--)
        if (level[u] + (1<<k) <= level[v])
            v = anc[v][k];
    if (u == v) return u;
    for (int k=K-1; k>=0; k--)
        if (anc[u][k] != anc[v][k])
            u = anc[u][k], v = anc[v][k];
    return anc[u][0];
}
int getanc(int u, int d) {
    for (int k=0; k<K; k++)
        if (d & (1<<k))
            u = anc[u][k];
    return u;
}

```

2.8 LiChaoTree

```

/// Min value query
struct func{
    ll operator()(ll x){...}
} tree[4*nmax], maxfunc;
#define lc (id<<1)
#define rc ((id<<1)|1)
void build(int id, int l, int r){
    tree[id] = maxfunc; if(l+1 == r) return;
    int mid = (l+r)/2;

```

```

    build(lc, l, mid); build(rc, mid, r);
}
void add_func(int id, int l, int r, func f){
    int mid = (l+r)/2;
    bool lefbbad = f(l) < tree[id](l);
    bool midbbad = f(mid) < tree[id](mid);
    if(midbbad) swap(f, tree[id]);
    if(l + 1 == r) return;
    else if(lefbbad != midbbad) add_func(lc, l, mid, f);
    else add_func(rc, mid, r, f);
}
ll get_val(int id, int l, int r, ll x){
    ll tmp = tree[id](x);
    if(l + 1 == r) return tmp;
    int mid = (l+r)/2;
    if(x < mid) return min(get_val(lc, l, mid, x), tmp);
    else return min(get_val(rc, mid, r, x), tmp);
}

```

2.9 LinkCutTree

```

const int MOD = 998244353;
int sum(int a, int b) {
    return a+b >= MOD ? a+b-MOD : a+b;
}
int mul(int a, int b) {
    return (a*1LL*b)%MOD;
}
typedef pair< int , int >Linear;
Linear compose(const Linear &p, const Linear &q) {
    return Linear(mul(p.first, q.first),
        sum(mul(q.second, p.first), p.second));
}
struct SplayTree {
    struct Node {
        int ch[2] = {0, 0}, p = 0;
        long long self = 0, path = 0; //Path aggregates
        long long sub = 0, vir = 0; //Subtree aggregate
        int size = 1; bool flip = 0; // Lazy tags
        Linear _self{1, 0}, shoja{1, 0}, ulta{1, 0};
    };
    vector<Node> T;
    SplayTree(int n) : T(n + 1) { T[0].size = 0; }
    void push(int x) {
        if (!x || !T[x].flip) return;
        int l = T[x].ch[0], r = T[x].ch[1];
        T[l].flip ^= 1, T[r].flip ^= 1;
        swap(T[x].ch[0], T[x].ch[1]); T[x].flip = 0;
        swap(T[x].shoja, T[x].ulta);
    }
    void pull(int x) {
        int l=T[x].ch[0],r=T[x].ch[1];push(l);push(r);
        T[x].size = T[l].size + T[r].size + 1;
        T[x].path = T[l].path + T[x].self + T[r].path;
        T[x].sub=T[x].vir+T[l].sub+T[r].sub+T[x].self;
        T[x].shoja = compose(T[r].shoja,
            compose(T[x]._self, T[l].shoja));
        T[x].ulta = compose(T[l].ulta,
            compose(T[x]._self, T[r].ulta));
    }
    void set(int x, int d, int y) {
        T[x].ch[d] = y; T[y].p = x; pull(x);
    }
}

```

```

void splay(int x) {
    auto dir = [&](int x) {
        int p = T[x].p; if (!p) return -1;
        return T[p].ch[0]==x?0:T[p].ch[1]==x?1:-1;
    };
    auto rotate = [&](int x) {
        int y = T[x].p,z=T[y].p,dx=dir(x),dy=dir(y);
        set(y, dx, T[x].ch[!dx]); set(x, !dx, y);
        if (~dy) set(z, dy, x); T[x].p = z;
    };
    for (push(x); ~dir(x); ) {
        int y = T[x].p,z = T[y].p; push(z); push(y);
        push(x); int dx = dir(x), dy = dir(y);
        if (~dy) rotate(dx!=dy?x:y); rotate(x);
    }
}
int KthNext(int x, int k) {
    assert(k > 0); splay(x); x = T[x].ch[1];
    if (T[x].size < k) return -1;
    while (true) {
        push(x); int l = T[x].ch[0], r = T[x].ch[1];
        if (T[l].size+1 == k) return x;
        if (k <= T[l].size) x = l;
        else k -= T[l].size+1, x = r;
    }
}
struct LinkCut : SplayTree {
    LinkCut(int n) : SplayTree(n) {}
    int access(int x) {
        int u = x, v = 0;
        for (; u; v = u, u = T[u].p) {
            splay(u); int& ov = T[u].ch[1];
            T[u].vir += T[ov].sub; T[u].vir -= T[v].sub;
            ov = v; pull(u);
        }
        splay(x); return v;
    }
    void reroot(int x) {
        access(x); T[x].flip ^= 1; push(x);
    }
    //makes v parent of u !(u must be a root)
    void Link(int u, int v) {
        reroot(u); access(v); T[v].vir += T[u].sub;
        T[u].p = v; pull(v);
    }
    //removes edge between u and v
    void Cut(int u, int v) {
        int _u = FindRoot(u); reroot(u); access(v);
        T[v].ch[0] = T[u].p = 0; pull(v); reroot(_u);
    }
    //Rooted tree LCA.Returns 0 if u v not connected
    int LCA(int u, int v) {
        if (u == v) return u; access(u);
        int ret = access(v); return T[u].p ? ret : 0;
    }
    //Query subtree of u where v is outside the sbtr
    long long Subtree(int u, int v) {
        int _v = FindRoot(v); reroot(v); access(u);
        long long ans = T[u].vir + T[u].self;
        reroot(_v); return ans;
    }
}

```

```

long long Path(int u, int v) {
    int _u = FindRoot(u); reroot(u); access(v);
    long long ans = T[v].path; reroot(_u);
    return ans;
}
Linear _Path(int u, int v) {
    reroot(u); access(v); return T[v].shoja;
}
void Update(int u, long long v) {
    access(u); T[u].self = v; pull(u);
}
void _Update(int u, Linear v) {
    access(u); T[u]._self = v; pull(u);
}
int FindRoot(int u) {
    access(u);
    while (T[u].ch[0]) { u = T[u].ch[0]; push(u); }
    access(u); return u;
}
//k-th node (0-indexed) on the path from u to v
int KthOnPath(int u, int v, int k) {
    if (u == v) return k == 0 ? u : -1;
    int _u = FindRoot(u); reroot(u); access(v);
    int ans = KthNext(u, k); reroot(_u);
    return ans;
}
int main() {
    int n, q; cin >> n >> q; LinkCut lct(n);
    for (int i = 1; i <= n; i++) {
        Linear l; cin >> l.first >> l.second;
        lct._Update(i, l);
    }
    for (int i = 1; i < n; i++) {
        int u, v; cin >> u >> v; lct.Link(u+1, v+1);
    }
    while (q--) {
        int op; cin >> op;
        if (op == 0) {
            int u, v, w, x; cin >> u >> v >> w >> x;
            lct.Cut(u+1, v+1); lct.Link(w+1, x+1);
        } else if (op == 1) {
            int p; Linear l; cin >> p >> l.first >> l.second;
            lct._Update(p+1, l);
        } else {
            int u, v, x; cin >> u >> v >> x;
            Linear l = lct._Path(u+1, v+1);
            cout << sum(mul(l.first, x), l.second) << "\n";
        }
    }
}

```

2.10 MoUpdate

```

B = [ n^(2/3), 1.26*n^(2/3) ]
struct query{
    /* t = number of updates before this query */
    int l, r, t, id;
    bool operator < (const query &x) const {
        if(1/B == x.l/B){
            if(r/B == x.r/B) return (t < x.t);
            return (r/B < x.r/B);
        }
    }
    return 1 / B < x.l / B;
}

```



```

}
};
struct upd{
    /// old = a[pos], a[pos] = cur
    int pos, old, cur;
};
void update(int pos, int x) {
    if (curL<=pos and pos<=curL)
        add(x), del(a[pos]);
    a[pos] = x;
}
t = totalupdates, curL = 1, curR = 0;
/// can start with t = 0 as well
for (int i = 1; i <= nq; i++) {
    int L = Q[i].l, R = Q[i].r, T = Q[i].t;
    while(t < T) t++, update(U[t].pos, U[t].cur);
    while(t > T) update(U[t].pos, U[t].old), t--;
    while(curL > L) add(a[--curL]);
    while(curR < R) add(a[++curR]);
    while(curL < L) del(a[curL++]);
    while(curR > R) del(a[curR--]);
    ans[Q[i].id] = something;
}

```

2.11 PSTree

```

typedef pair<int,int>PII;
typedef long long LL;
#define all(x) x.begin(), x.end()
const int MAXN = 2e5+7;
const int LOGN = 19; ///ATTENTION
namespace PSTree {
    struct PSNode { int cnt, lc, rc; };
    PSNode tr[MAXN*LOGN+100];
    ///since there is no build function, you either
    ///write an appropriate build function or make
    ///sure cnt = identity; zero-th node as initial
    int counter;
    void clear() { counter = 0; }
    int update(int u, int l, int r, int idx, int v){
        if (idx < l || r < idx) return u;
        if (l == r) {
            int nd = ++counter; tr[nd].cnt=tr[u].cnt+v;
            return nd;
        }
        int mid = (l+r)/2; int nd = ++counter;
        tr[nd].lc =update(tr[u].lc, l, mid, idx, v);
        tr[nd].rc =update(tr[u].rc, mid+1, r, idx, v);
        tr[nd].cnt=tr[tr[nd].lc].cnt+tr[tr[nd].rc].cnt;
        return nd;
    }
    int getKth(vector<PII>vp, int l, int r, int k){
        if (l==r) return l; int mid = (l+r)/2, bam = 0;
        for (PII pr : vp) {
            int lc = tr[pr.first].lc;
            bam += tr[lc].cnt * pr.second;
        }
        if (k <= bam) {
            for (PII &pr:vp) pr.first = tr[pr.first].lc;
            return getKth(vp, l, mid, k);
        } else {
            for (PII &pr:vp) pr.first = tr[pr.first].rc;
            return getKth(vp, mid+1, r, k-bam);
        }
    }
}

```

```

}
}
int ar[MAXN], rt[MAXN];
int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    int n, m;
    cin >> n >> m;
    vector<int> vas(n);
    for (int i = 1; i <= n; i++) {
        cin >> ar[i];
        vas[i-1] = ar[i];
    }
    sort(all(vas));
    vas.erase(unique(all(vas)), vas.end());
    int sz = vas.size();
    for (int i = 1; i <= n; i++) {
        ar[i]=lower_bound(all(vas), ar[i])-vas.begin();
        rt[i]=PSTree::update(rt[i-1], 0, sz-1, ar[i], 1);
    }
    while (m--) {
        int l, r, k;
        cin >> l >> r >> k;
        int v = PSTree::getKth({PII(rt[r], +1),
                                PII(rt[l], -1)}, 0, sz-1, k+1);
        cout << vas[v] << "\n";
    }
    return 0;
}

```

2.12 SegmentTree

```

const int MAXN = 1e5+7; /// change this
struct nd { int mn, cnt; };
nd tr[4*MAXN]; int lazy[4*MAXN];
///1. Merge left and right
nd combine(const nd &a, const nd &b) {
    if (a.mn < b.mn) return a;
    else if (a.mn > b.mn) return b;
    nd rt;rt.mn=a.mn;rt.cnt=a.cnt+b.cnt;return rt;
}
///2. Push lazy down and merge lazy
void propagate(int u, int l, int r) {
    if (lazy[u]) {
        tr[u].mn += lazy[u];
        if (l != r) {
            lazy[u*2] += lazy[u]; lazy[u*2+1] += lazy[u];
        }
        lazy[u] = 0;
    }
}
int a[MAXN];
void build(int u, int l, int r) {
    lazy[u] = 0; ///3. Initialize
    if (l==r) {
        tr[u].mn = 0; tr[u].cnt = a[l]; return;
    }
    int mid = (l+r)/2;
    build(u*2, l, mid); build(u*2+1, mid+1, r);
    tr[u] = combine(tr[u*2], tr[u*2+1]);
}
void update(int u, int l, int r, int x, int y, int v){

```

```

    propagate(u, l, r);
    if (r < x || y < l) return;
    if (x <= l && r <= y) {
        lazy[u] += v; ///4. Merge lazy
        propagate(u, l, r); return;
    }
    int mid = (l+r)/2; update(u*2, l, mid, x, y, v);
    update(u*2+1, mid+1, r, x, y, v);
    tr[u] = combine(tr[u*2], tr[u*2+1]);
}
nd query(int u, int l, int r, int x, int y) {
    propagate(u, l, r);
    if (x <= l && r <= y) return tr[u];
    int mid = (l+r)/2;
    if (y <= mid) return query(u*2, l, mid, x, y);
    if (mid < x) return query(u*2+1, mid+1, r, x, y);
    return combine(query(u*2, l, mid, x, y),
                  query(u*2+1, mid+1, r, x, y));
}

```

2.13 Sparsetable

```

int table[N][int(log2(N))+1], A[N];
void buildSparseforMIN(int n){
    for(int i=1;i<=n;i++) table[i][0]=A[i];
    for(int j=1; (1<=j)<=n; j++)
        for(int i=1; (i+(1<=j)-1)<=n; i++)
            table[i][j]=min(table[i][j-1],
                            table[i+(1<=j)-1][j-1]);
}
int MIN(int i, int j){
    ///int k = 32 - __builtin_clz(j-i+1) - 1;
    int k=log2(j-i+1);
    return min(table[i][k], table[j-(1<=k)+1][k]);
}

```

2.14 centroidDecomp

```

vector<int> adj[N], cen_tree[N];
bool cent_mark[N]; int sub[N];
void dfs(int ind, int &n, int par = -1){
    n++, sub[ind] = 1;
    for(auto x : adj[ind]){
        if(x != par && !cent_mark[x])
            dfs(x, n, ind), sub[ind] += sub[x];
    }
}
int get_centroid(int ind, int n, int par=-1){
    for(auto x : adj[ind]){
        if(x != par && !cent_mark[x]){
            if(sub[x]>n) return get_centroid(x, n, ind);
        }
    }
    return ind;
}
int decompose(int ind){
    int n = 0; dfs(ind, n);
    int cn = get_centroid(ind, n >> 1);
    cent_mark[cn] = 1;
    for(auto x : adj[cn]){
        if(!cent_mark[x]){
            int y = decompose(x);
            cen_tree[cn].push_back(y);
        }
    }
    return cn;
}

```

2.15 wavelet

```

int a[N];
struct weblet{
    int lo, hi;
    weblet *l=0, *r=0;
    vector<int> b, c;
// call weblet(a+1, a+n+1, minval, maxval)
    weblet(int *from, int *to, int x, int y){
        lo = x, hi = y;
        if( from >= to) return;
        if( hi == lo ){
            b.reserve(to-from+1), b.pb(0);
            c.reserve(to-from+1), c.pb(0);
            for(auto it=from; it!=to; it++){
                b.pb(b.back() + 1);
                c.pb(c.back()+*it);
            }
            return;
        }
        int mid = (lo+hi)/2;
        auto f = [mid](int x){
            return x <= mid;
        };
        b.reserve(to-from+1), b.pb(0);
        c.reserve(to-from+1), c.pb(0);
        for(auto it = from; it != to; it++){
            b.pb(b.back() + f(*it));
            c.pb(c.back() + *it);
        }
        auto pivot = stable_partition(from, to, f);
        l = new weblet(from, pivot, lo, mid);
        r = new weblet(pivot, to, mid+1, hi);
    }
    void swapadjacen(int i){/// i with i+1{
        if(lo == hi) return ;
        b[i]= b[i-1] + b[i+1] - b[i];
        c[i] = c[i-1] + c[i+1] - c[i];
        if( b[i+1]-b[i] == b[i] - b[i-1]){
            if(b[i] -b[i-1])
                return this->l->swapadjacen(b[i]);
            else return this->r->swapadjacen(i-b[i]);
        }
        else return ;
    }
    int kth(int l, int r, int k){
        if(l > r) return 0;
        if(lo == hi) return lo;
        int inLeft = b[r] - b[l-1];
        int lb = b[l-1], rb = b[r];
        if(k<=inLeft)
            return this->l->kth(lb+1, rb, k);
        return this->r->kth(l-lb, r-rb, k-inLeft);
    }
    int LTE(int l, int r, int k){
        if(l > r or k < lo) return 0;
        if(hi <= k) return r - l + 1;
        int lb = b[l-1], rb = b[r];
        return this->l->LTE(lb+1, rb, k)+
            this->r->LTE(l-lb, r-rb, k);
    }
    int count(int l, int r, int k){
        if(l > r or k < lo or k > hi) return 0;

```

```

        if(lo == hi) return r - l + 1;
        int lb = b[l-1], rb = b[r], md = (lo+hi)/2;
        if(k<=md) return this->l->count(lb+1,rb,k);
        return this->r->count(l-lb, r-rb, k);
    }
    int sumk(int l, int r, int k){/// sumof <=k
        if(l > r or k < lo) return 0;
        if(hi <= k) return c[r] - c[l-1];
        int lb = b[l-1], rb = b[r];
        return this->l->sumk(lb+1, rb, k) +
            this->r->sumk(l-lb, r-rb, k);
    }
    ~weblet(){
        if(l) delete l;
        if(r) delete r;
    }
};

```

3 Flow Matching

3.1 DemandFlo

```

cap2(u -> v) = cap(u -> v) - lo(u -> v)
cap2(supsorc -> v) = sum of lo(u -> v)
cap2(u -> supsink) = sum of lo(u -> v)
cap2(sink -> sorc) = inf

```

3.2 Dinic

```

namespace Dinic {
    typedef long long LL;
    const int N = 5005, K = 60; const LL INF = 1e18;
    struct Edge { int frm, to; LL cap, flow; };
    int s, t, n, level[N], ptr[N];
    vector<Edge> edges; vector<int> adj[N];
    void init(int nodes) {
        n = nodes; edges.clear();
        for (int i=0; i<n; i++) adj[i].clear();
    }
    ///adding undirected edge call addEdge(u,v,c,c);
    int addEdge(int a, int b, LL cap, LL revcap = 0) {
        edges.push_back({a, b, cap, 0});
        edges.push_back({b, a, revcap, 0});
        adj[a].push_back(edges.size()-2);
        adj[b].push_back(edges.size()-1);
        return edges.size()-2;
    }
    bool bfs(LL lim) {
        fill(level, level+n, -1); level[s] = 0;
        queue<int> q; q.push(s);
        while (!q.empty() && level[t] == -1) {
            int v = q.front(); q.pop();
            for (int id: adj[v]) {
                Edge e = edges[id];
                if (level[e.to]==-1 && e.cap-e.flow>=lim) {
                    q.push(e.to); level[e.to] = level[v] + 1;
                }
            }
        }
        return level[t] != -1;
    }
    LL dfs(int v, LL flow) {
        if (v == t || !flow) return flow;
        for (; ptr[v] < adj[v].size(); ptr[v]++) {
            int eid = adj[v][ptr[v]];

```

```

            Edge &e = edges[eid];
            if (level[e.to] != level[v] + 1) continue;
            if (LL pushed=dfs(e.to,min(flow,e.cap-e.flow))){
                e.flow+=pushed; edges[eid^1].flow -= pushed;
                return pushed;
            }
        }
        return 0;
    }
    LL maxFlow(int source,int sink,bool SCALING=false){
        s = source, t = sink;
        long long flow = 0;
        for (LL lim=SCALING?(1LL<<K):1; lim>0; lim>=>=1){
            while (bfs(lim)) {
                fill(ptr, ptr+n, 0);
                while (LL pushed = dfs(s,INF)) flow+=pushed;
            }
        }
        return flow;
    }
    bool leftOfMinCut(int x) {return level[x] != -1;}
    vector<vector<LL>> allPairFlow(vector<Edge>&tree){
        tree.clear(); vector<int> par(n);
        vector<vector<LL>> flow(n, vector<LL>(n, INF));
        for (int i=1; i<n; i++) {
            for (auto &e: edges) e.flow = 0;
            LL f = maxFlow(i, par[i]);
            tree.push_back({i, par[i], f});
            for (int j=i+1; j<n; j++)
                if(par[j]==par[i]&&leftOfMinCut(j))par[j]=i;
            flow[i][par[i]] = flow[par[i]][i] = f;
            for (int j=0; j<i; j++)
                if (j != par[i]) flow[i][j]=flow[j][i]
                    =min(f,flow[par[i]][j]);
        }
        return flow;
    }
}

```

3.3 GeneralMatching

```

namespace Blossom { /// 1-indexed, O(n m log n)
    const int N = 5005; int t, n, ans;
    int vis[N], par[N], orig[N], match[N], aux[N];
    vector<int> adj[N]; queue<int> q;
    void init(int nn) {
        n = nn; t = ans = 0;
        for(int i=0; i<=n; i++) {
            adj[i].clear();match[i] = aux[i] = par[i] = 0;
        }
    }
    void addEdge(int u, int v) {
        adj[u].push_back(v); adj[v].push_back(u);
        if (!match[u] && !match[v]) {
            match[u] = v; match[v] = u; ans++;
        }
    }
    void augment(int u, int v) {
        int pv = v, nv; do {
            pv = par[v]; nv = match[pv];
            match[v] = pv; match[pv] = v; v = nv;
        } while(u != pv);
    }
    int lca(int v, int w) {

```

```

++t;
while(true) { if(v) {
    if(aux[v] == t) return v;
    aux[v] = t; v = orig[par[match[v]]];
}
swap(v, w);
}
}
void blossom(int v, int w, int a) {
    while(orig[v] != a) {
        par[v] = w; w = match[v];
        if(vis[w] == 1) q.push(w), vis[w] = 0;
        orig[v] = orig[w] = a; v = par[w];
    }
}
bool bfs(int u) {
    fill(vis+1, vis+1+n, -1); iota(orig+1, orig+1+n, 1);
    q = queue<int> ({u}); vis[u] = 0;
    while(!q.empty()) {
        int v = q.front(); q.pop();
        for(int x: adj[v]) {
            if(vis[x] == -1) {
                par[x] = v; vis[x] = 1;
                if(!match[x]) {augment(u, x); return true;}
                q.push(match[x]); vis[match[x]] = 0;
            }
            else if(vis[x] == 0 && orig[v] != orig[x]) {
                int a = lca(orig[v], orig[x]);
                blossom(x, v, a); blossom(v, x, a);
            }
        }
    }
    return false;
}
int maxMatching() {
    for(int i=1; i<=n; ++i) if(!match[i] && bfs(i)) ++ans;
    return ans;
}
}

```

3.4 GlobalMinCut

```

/// 0(n^3) OUTPUT: (min cut value, nodes in left)
pair<LL, vector<int>> glMinCut(vector<vector<LL>>> c) {
    int N = c.size();
    LL best_w = -1;
    vector<int> used(N), cut, best_cut;
    for (int phase = N-1; phase >= 0; phase--) {
        vector<LL> w = c[0];
        vector<int> vis = used;
        int prev, last = 0;
        for (int i = 0; i < phase; i++) {
            prev = last;
            last = -1;
            for (int j = 1; j < N; j++)
                if (!vis[j] && (last == -1 || w[j] > w[last]))
                    last = j;
            if (i == phase-1) {
                for (int j=0; j<N; j++) c[prev][j] += c[last][j];
                for (int j=0; j<N; j++) c[j][prev] = c[prev][j];
                used[last] = true; cut.push_back(last);
                if (best_w == -1 || w[last] < best_w) {
                    best_cut = cut; best_w = w[last];
                }
            }
        }
    }
}

```

```

} else {
    for (int j=0; j<N; j++) w[j] += c[last][j];
    vis[last] = true;
}
}
return make_pair(best_w, best_cut);
}

```

3.5 Hungarian

```

template<typename T> /// 0(n^2 m), n<=m,
pair<T, vector<int>> WBM(vector<vector<T>> cost) {
    const T INF = numeric_limits<T>::max();
    int n = cost.size()-1, m = cost[0].size()-1;
    vector<T> U(n+1), V(n+1);
    vector<int> mr(m+1), way(m+1), ml(n+1);
    for(int i = 1; i<=n; i++) {
        mr[0] = i; int lastJ = 0;
        vector<T> minV(m+1, INF); vector<bool> used(m+1);
        do {
            used[lastJ] = true;
            int lastI = mr[lastJ], nextJ; T delta = INF;
            for(int j = 1; j<=m; j++) {
                if(used[j]) continue;
                T diffCost = cost[lastI][j] - U[lastI] - V[j];
                if(diffCost < minV[j]) {
                    minV[j] = diffCost; way[j] = lastJ;
                    if(minV[j] < delta) delta = minV[j], nextJ = j;
                }
            }
            for(int j = 0; j<=m; j++) {
                if(used[j]) U[mr[j]] += delta, V[j] -= delta;
                else minV[j] -= delta;
            }
            lastJ = nextJ;
        } while(mr[lastJ] != 0);
        do {
            int prevJ = way[lastJ];
            mr[lastJ] = mr[prevJ]; lastJ = prevJ;
        } while(lastJ != 0);
    }
    for (int i=1; i<=m; i++) ml[mr[i]] = i;
    return {-V[0], ml};
}

```

3.6 MCMF

```

namespace MCMF {
    typedef long long F; typedef long long C;
    const F infF = 1e18; const C infC = 1e18;
    const int N = 5005; typedef pair<C, F> PCF;
    struct Edge {int frm, to; C cost; F cap, flow;};
    int n, s, t, prv[N], vis[N]; C pi[N], dis[N]; F fl[N];
    vector<Edge> edges; vector<int> adj[N];
    void init(int nodes, int source, int sink) {
        n = nodes, s = source, t = sink; edges.clear();
        for (int i=0; i<n; i++) pi[i]=0, adj[i].clear();
    }
    void addEdge(int u, int v, F cap, C cost) {
        edges.push_back({u, v, cost, cap, 0});
        edges.push_back({v, u, -cost, 0, 0});
        adj[u].push_back(edges.size()-2);
        adj[v].push_back(edges.size()-1);
    }
}

```

```

bool SPFA() {
    for (int i=0; i<n; i++) {
        dis[i]=infC; fl[i]=0; vis[i]=0; prv[i]=-1; }
    queue<int> q; q.push(s);
    dis[s] = 0; fl[s] = infF; vis[s] = 1;
    while (!q.empty()) {
        int u = q.front(); q.pop(); vis[u] = 0;
        for (int eid : adj[u]) {
            Edge &e = edges[eid];
            if (e.cap == e.flow) continue;
            if (dis[u] + e.cost < dis[e.to]) {
                dis[e.to] = dis[u] + e.cost; prv[e.to] = eid^1;
                fl[e.to] = min(fl[u], e.cap - e.flow);
                if (!vis[e.to]) q.push(e.to);
            }
        }
    }
    return fl[t] > 0;
}
PCF solveSPFA() {
    C cost = 0; F flow = 0;
    while (SPFA()) {
        C pathcost = dis[t];
        cost += pathcost*fl[t]; flow += fl[t];
        for (int u=t, e=prv[u]; e!=-1;
            u=edges[e].to, e=prv[u]) {
            edges[e].flow -= fl[t]; edges[e^1].flow += fl[t];
        }
    }
    return {cost, flow};
}
void normalize() { //Sets pi for Dijkstra()
    SPFA(); for (int i=0; i<n; i++) pi[i] = dis[i];
}
bool Dijkstra() {
    for (int i=0; i<n; i++) {
        dis[i]=infC; fl[i]=0; vis[i]=0; prv[i]=-1; }
    priority_queue<pair<C, int>> pq;
    pq.emplace(0, s); dis[s] = 0; fl[s] = infF;
    while (!pq.empty()) {
        int u = pq.top().second; pq.pop();
        if (vis[u]) continue; vis[u] = 1;
        for (int eid : adj[u]) {
            Edge &e = edges[eid];
            if (vis[e.to] || e.cap == e.flow) continue;
            C nw = dis[u] + e.cost - pi[e.to] + pi[u];
            if (nw < dis[e.to]) {
                dis[e.to] = nw; prv[e.to] = eid^1;
                fl[e.to] = min(fl[u], e.cap - e.flow);
                pq.emplace(-dis[e.to], e.to);
            }
        }
    }
    return fl[t] > 0;
}
PCF solveDijkstra() {
    normalize(); C cost = 0; F flow = 0;
    while (Dijkstra()) {
        for (int i=0; i<n; i++)
            if (fl[i]) pi[i] += dis[i];
        C pathcost = pi[t] - pi[s];
    }
}

```

```

cost += pathcost*f1[t]; flow += f1[t];
for (int u=t, e=prv[u]; e!=-1;
     u=edges[e].to, e=prv[u]) {
    edges[e].flow-=f1[t]; edges[e^1].flow+=f1[t];
}
}
return {cost, flow};
}
}

```

3.7 hopcroft

```

namespace HopcroftKarp { // 1-indexed. L,R indep
const int maxN=1e5+7, maxM=1e5+7; int n, m, match;
int vis[maxN], level[maxN], ml[maxN], mr[maxM];
vector<int> edge[maxN];
void init(int N, int M) { //N=left nodes, M = right
    n = N, m = M;
    for (int i=1; i<=n; i++) edge[i].clear(), ml[i]=-1;
    for (int i=1; i<=m; i++) mr[i] = -1;
    match = 0;
}
void add(int u, int v) {
    edge[u].push_back(v);
    if (ml[u]==-1 && mr[v]==-1) ml[u]=v, mr[v]=u, match++;
}
bool dfs(int u) {
    vis[u] = true;
    for (int x: edge[u]) {
        int v = mr[x];
        if (v == -1 || (!vis[v]
                        && level[u] < level[v] && dfs(v))) {
            ml[u] = x; mr[x] = u; return true;
        }
    }
    return false;
}
int matching() {
    while (true) {
        queue<int> q;
        for (int i = 1; i <= n; ++i) {
            if (ml[i] == -1) level[i] = 0, q.push(i);
            else level[i] = -1;
        }
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (int x: edge[u]) {
                int v = mr[x];
                if (v != -1 && level[v] < 0) {
                    level[v] = level[u] + 1; q.push(v);
                }
            }
        }
        for (int i = 1; i <= n; ++i) vis[i] = false;
        int d = 0;
        for (int i=1; i<=n; ++i) if (ml[i]==-1 && dfs(i)) ++d;
        if (d == 0) return match; match += d;
    }
}
}

```

4 Geo

4.1 3DGeo

```

const double PI = acos(-1), EPS = 1e-9;
int dcmp(double x){return abs(x)<EPS?0:(x<0?-1:1);}

```

```

struct Point {
    double x, y, z;
    Point() : x(0), y(0), z(0) {}
    Point(double X, double Y, double Z) :
        x(X), y(Y), z(Z) {}
    Point operator + (const Point& u) const {
        return Point(x + u.x, y + u.y, z + u.z);
    }
    Point operator - (const Point& u) const {
        return Point(x - u.x, y - u.y, z - u.z);
    }
    Point operator * (const double u) const {
        return Point(x * u, y * u, z * u);
    }
    Point operator / (const double u) const {
        return Point(x / u, y / u, z / u);
    }
    friend std::ostream &operator << (
        std::ostream &os, const Point &p) {
        return os << p.x << " " << p.y << " " << p.z;
    }
    friend std::istream &operator >> (
        std::istream &is, Point &p) {
        return is >> p.x >> p.y >> p.z;
    }
};

double dot(Point a, Point b) {
    return a.x * b.x + a.y * b.y + a.z * b.z;
}
Point cross(Point a, Point b) {
    return Point(a.y*b.z - a.z*b.y,
                a.z*b.x - a.x*b.z, a.x*b.y - a.y*b.x);
}
double length(Point a) { return sqrt(dot(a, a)); }
double distance(Point a, Point b) {
    return length(a-b);
}
Point unit(const Point &p) { return p/length(p); }
// Rotate p around axis x, with angle radians.
Point rotate(Point p, Point axis, double angle) {
    axis = unit(axis); Point comp1 = p * cos(angle);
    Point comp2 = axis*(1-cos(angle))*dot(axis, p);
    Point comp3 = cross(axis, p) * sin(angle);
    return comp1 + comp2 + comp3;
}

struct Line {Point a, v;}; //a+tv
double distancePointLine(Point p, Line l) {
    return length(cross(l.v, p - l.a)) / length(l.v);
}
// distance from Line ab to Line cd
double distanceLineLine(Line a, Line b) {
    Point cr = cross(a.v, b.v);
    double crl = length(cr);
    if (dcmp(crl)==0) return distancePointLine(a.a, b);
    return abs(dot(cr, a.a-b.a))/crl;
}

struct Plane {
    Point normal; double d; // dot(Normal) = d
    Point P; // anyPoint on the plane
    Plane(Point normal, double d) {
        double len = length(normal);
        normal = normal / len; d = d / len;
        if (dcmp(normal.x)) P=Point(d/normal.x, 0, 0);
        else if (dcmp(normal.y)) P=Point(0, d/normal.y, 0);
        else P = Point(0, 0, d/normal.z);
    }
    Plane(Point a, Point b, Point c) {
        normal = unit(cross(b-a, c-a));
        d = dot(normal, a); P = a;
    }
}

```

```

bool onPlane(Point a) {
    return dcmp(dot(normal, a) - d) == 0;
}
double distance(Point a) {
    return abs(dot(normal, a) - d);
}
double isParallel(Line l) {
    return dcmp(dot(l.v, normal)) == 0;
}
//return t st l.a + t*l.v is a point on plane,
//check parallel first
double intersectLine(Line l) {
    return dot(P-l.a, normal)/dot(l.v, normal);
}
}

```

4.2 Geo

```

typedef double Tf; typedef double Ti;
const Tf PI = acos(-1), EPS = 1e-9;
int dcmp(Tf x) {return abs(x)<EPS?0:(x<0?-1:1);}
struct PT {
    Ti x, y;
    PT(Ti x = 0, Ti y = 0) : x(x), y(y) {}
    PT operator + (const PT& u) const {
        return PT(x + u.x, y + u.y);
    }
    PT operator - (const PT& u) const {
        return PT(x - u.x, y - u.y);
    }
    PT operator * (const long long u) const {
        return PT(x * u, y * u);
    }
    PT operator * (const Tf u) const {
        return PT(x * u, y * u);
    }
    PT operator / (const Tf u) const {
        return PT(x / u, y / u);
    }
    bool operator == (const PT& u) const {
        return dcmp(x-u.x)==0 && dcmp(y-u.y)==0;
    }
    bool operator != (const PT& u) const {
        return !(*this == u);
    }
    bool operator < (const PT& u) const {
        return dcmp(x-u.x) < 0 ||
            (dcmp(x-u.x) == 0 && dcmp(y-u.y) < 0);
    }
    friend istream &operator >> (istream &is, PT &p) {
        return is >> p.x >> p.y;
    }
    friend ostream &operator << (ostream &os,
        const PT &p) {return os << p.x << " " << p.y; }
};

Ti dot(PT a, PT b) { return a.x*b.x + a.y*b.y; }
Ti cross(PT a, PT b) { return a.x*b.y - a.y*b.x; }
Tf length(PT a) { return sqrt(dot(a, a)); }
Ti sqLength(PT a) { return dot(a, a); }
Tf distance(PT a, PT b) { return length(a-b); }
Tf angle(PT u) { return atan2(u.y, u.x); }
Tf angleBetween(PT a, PT b) { //in range [-PI, PI]
    Tf ans = angle(b) - angle(a);
    return ans <= -PI ? ans + 2*PI :
        (ans > PI ? ans - 2*PI : ans);
}

PT rotate(PT a, Tf rad) {
    static_assert(is_same<Tf, Ti>::value);
    return PT(a.x * cos(rad) - a.y * sin(rad),
              a.x * sin(rad) + a.y * cos(rad));
}
// Rotate(a, rad) where cos(rad)=co, sin(rad)=si
PT rotatePrecise(PT a, Tf co, Tf si) {
    static_assert(is_same<Tf, Ti>::value);
}

```



```

    return PT(a.x*co - a.y*si, a.y*co + a.x*si);
}
PT rotate90(PT a) { return PT(-a.y, a.x); }
PT scale(PT a, Tf s) {
    static_assert(is_same<Tf, Ti>::value);
    return a / length(a) * s;
}
PT normal(PT a) {
    static_assert(is_same<Tf, Ti>::value);
    Tf l = length(a); return PT(-a.y / l, a.x / l);
}
// returns 1/0/-1 if c is left/on/right of ab
int orient(PT a, PT b, PT c) {
    return dcmp(cross(b - a, c - a));
}
//sort(v.begin(), v.end(), polarComp(0, dir))
struct polarComp {
    PT o, dir;
    polarComp(PT o = PT(0, 0), PT dir = PT(1, 0))
        : o(o), dir(dir) {}
    bool half(PT p) {
        return dcmp(cross(dir, p)) < 0 ||
            (dcmp(cross(dir, p)) == 0 && dcmp(dot(dir, p)) > 0);
    }
    bool operator()(PT p, PT q) {
        return make_tuple(half(p-o), 0) <
            make_tuple(half(q-o), cross(p-o, q-o));
    }
};
struct Segment {
    PT a, b;
    Segment() {}
    Segment(PT aa, PT bb) : a(aa), b(bb) {}
}; typedef Segment Line;
struct Circle {
    PT o; Tf r;
    Circle(PT o = PT(0, 0), Tf r = 0) : o(o), r(r) {}
    bool contains(PT p) {
        return dcmp(sqLength(p - o) - r * r) <= 0;
    }
    PT point(Tf rad) {
        static_assert(is_same<Tf, Ti>::value);
        return PT(o.x+cos(rad)*r, o.y+sin(rad)*r);
    }
    Tf area(Tf rad = PI + PI) {return rad * r * r/2;}
    Tf sector(Tf alpha) {
        return r*r*0.5*(alpha-sin(alpha));
    }
};
#####
namespace Linear {
bool onSegment(PT p, Segment s) { ///Is p on S?
    return dcmp(cross(s.a - p, s.b - p)) == 0 &&
        dcmp(dot(s.a - p, s.b - p)) <= 0;
}
bool segmentsIntersect(Segment p, Segment q) {
    if(onSegment(p.a, q) || onSegment(p.b, q)) return 1;
    if(onSegment(q.a, p) || onSegment(q.b, p)) return 1;
    Ti c1 = cross(p.b - p.a, q.a - p.a);
    Ti c2 = cross(p.b - p.a, q.b - p.a);
    Ti c3 = cross(q.b - q.a, p.a - q.a);
    Ti c4 = cross(q.b - q.a, p.b - q.a);
    return dcmp(c1)*dcmp(c2) < 0 && dcmp(c3)*dcmp(c4) < 0;
}
}

```

```

bool linesParallel(Line p, Line q) {
    return dcmp(cross(p.b - p.a, q.b - q.a)) == 0;
}
//returns if lines (p, p+v) && (q, q+w) intersect
bool lineLineIntersect(PT p, PT v, PT q, PT w, PT& o) {
    static_assert(is_same<Tf, Ti>::value);
    if(dcmp(cross(v, w)) == 0) return false;
    PT u = p - q; o = p + v*(cross(w, u)/cross(v, w));
    return true;
}
bool lineLineIntersect(Line p, Line q, PT& o) {
    return lineLineIntersect(p.a, p.b - p.a, q.a,
        q.b - q.a, o);
}
Tf distancePointLine(PT p, Line l) {
    return abs(cross(l.b-l.a, p-l.a)/length(l.b-l.a));
}
Tf distancePointSegment(PT p, Segment s) {
    if(s.a == s.b) return length(p - s.a);
    PT v1 = s.b - s.a, v2 = p - s.a, v3 = p - s.b;
    if(dcmp(dot(v1, v2)) < 0) return length(v2);
    else if(dcmp(dot(v1, v3)) > 0) return length(v3);
    else return abs(cross(v1, v2) / length(v1));
}
Tf distanceSegmentSegment(Segment p, Segment q) {
    if(segmentsIntersect(p, q)) return 0;
    Tf ans = distancePointSegment(p.a, q);
    ans = min(ans, distancePointSegment(p.b, q));
    ans = min(ans, distancePointSegment(q.a, p));
    ans = min(ans, distancePointSegment(q.b, p));
    return ans;
}
PT projectPointLine(PT p, Line l) {
    static_assert(is_same<Tf, Ti>::value);
    PT v = l.b - l.a;
    return l.a + v * ((Tf) dot(v, p-l.a)/dot(v, v));
}
} // namespace Linear
#####
typedef vector<PT> Polygon;
namespace Polygona {
/// cannot be all collinear
Polygon RemoveCollinear(const Polygon& poly) {
    Polygon ret;
    int n = poly.size();
    for(int i = 0; i < n; i++) {
        PT a = poly[i];
        PT b = poly[(i + 1) % n];
        PT c = poly[(i + 2) % n];
        if(dcmp(cross(b-a, c-b)) != 0 && (ret.empty() ||
            b != ret.back())) ret.push_back(b);
    }
    return ret;
}
Tf signedPolygonArea(const Polygon& p) {
    Tf ret = 0;
    for(int i = 0; i < (int) p.size() - 1; i++)
        ret += cross(p[i]-p[0], p[i+1]-p[0]);
    return ret / 2;
}
//fails if all collinear and remove = TRUE
Polygon convexHull(Polygon p, bool remRedundant) {
    int check = remRedundant ? 0 : -1;
}

```

```

sort(p.begin(), p.end());
p.erase(unique(p.begin(), p.end(), p.end()));
int n = p.size(); Polygon ch(n+1);
int m = 0; // preparing lower hull
for(int i = 0; i < n; i++) {
    while(m > 1 && dcmp(cross(ch[m-1]-ch[m-2],
        p[i] - ch[m-1])) <= check) m--;
    ch[m++] = p[i];
}
int k = m; // preparing upper hull
for(int i = n - 2; i >= 0; i--) {
    while(m > k && dcmp(cross(ch[m-1] - ch[m-2],
        p[i] - ch[m-2])) <= check) m--;
    ch[m++] = p[i];
}
if(n > 1) m--; ch.resize(m);
return ch;
}
// returns inside = -1, on = 0, outside = 1
int pointInPolygon(const Polygon& p, PT o) {
    using Linear::onSegment; int wn=0, n = p.size();
    for(int i = 0; i < n; i++) {
        int j = (i + 1) % n; if(onSegment(o,
            Segment(p[i], p[j])) || o == p[i]) return 0;
        int k = dcmp(cross(p[j] - p[i], o - p[i]));
        int d1=dcmp(p[i].y-o.y), d2=dcmp(p[j].y-o.y);
        if(k > 0 && d1 <= 0 && d2 > 0) wn++;
        if(k < 0 && d2 <= 0 && d1 > 0) wn--;
    }
    return wn ? -1 : 1;
}
// returns (longest segment, total length)
pair<Tf, Tf> linePolygonIntersection(Line l,
    const Polygon& p) {
    using Linear::lineLineIntersect;
    int n = p.size(); vector<pair<Tf, int>> ev;
    for(int i=0; i<n; ++i) {
        PT a = p[i], b = p[(i+1)%n], z = p[(i-1+n)%n];
        int ora=orient(l.a,l.b,a), orb =
            orient(l.a,l.b,b), orz=orient(l.a,l.b,z);
        if(l.ora) {
            Tf d = dot(a - l.a, l.b - l.a);
            if(orz && orb) {
                if(orz != orb) ev.emplace_back(d, 0);
                //else // PT Touch
            } else if(orz) ev.emplace_back(d, orz);
            else if(orb) ev.emplace_back(d, orb);
        }
        else if(ora == -orb) {
            PT ins;
            lineLineIntersect(l, Line(a, b), ins);
            ev.emplace_back(dot(ins-l.a, l.b-l.a), 0);
        }
    }
    sort(ev.begin(), ev.end());
    Tf ans = 0, len = 0, last = 0, tot = 0;
    bool active = false; int sign = 0;
    for(auto &qq : ev) {
        int tp = qq.second;
        Tf d = qq.first; //current Seg is (last, d)
        if(sign) { //On Border
            len+=d-last; tot+=d-last; ans=max(ans, len);
        }
    }
}

```

```

    if(tp != sign) active = !active;
    sign = 0;
}
else {
    if(active) { ///Strictly Inside
        len+=d-last;tot+=d-last;ans=max(ans,len);
    }
    if(tp == 0) active=!active; else sign = tp;
}
last = d; if(!active) len = 0;
}
ans /= length(l.b-l.a); tot /= length(l.b-l.a);
return {ans, tot};
} } namespace Polygonal
#####
namespace Convex {
Polygon minkowskiSum(Polygon A, Polygon B){
    int n = A.size(), m = B.size();
    rotate(A.begin(),
        min_element(A.begin(), A.end()), A.end());
    rotate(B.begin(),
        min_element(B.begin(), B.end()), B.end());
    A.push_back(A[0]); B.push_back(B[0]);
    for(int i = 0; i < n; i++) A[i] = A[i+1] - A[i];
    for(int i = 0; i < m; i++) B[i] = B[i+1] - B[i];
    Polygon C(n+m+1); C[0] = A.back() + B.back();
    merge(A.begin(), A.end()-1, B.begin(), B.end()-1,
        C.begin()+1, polarComp(PT(0, 0), PT(0, -1)));
    for(int i=1; i<C.size(); i++) C[i]=C[i]+C[i-1];
    C.pop_back(); return C;
}
///{min area, min perimeter} rectangle containing p
pair<Tf,Tf>rotatingCalipersBBox(const Polygon &p){
    using Linear::distancePointLine;
    static_assert(is_same<Tf, Ti>::value);
    int n = p.size(); int l = 1, r = 1, j = 1;
    Tf area = 1e100; Tf perimeter = 1e100;
    for(int i = 0; i < n; i++) {
        PT v=(p[(i+1)%n]-p[i])/length(p[(i+1)%n]-p[i]);
        while(dcmp(dot(v, p[r%n] - p[i]) -
            dot(v, p[(r+1)%n] - p[i])) < 0) r++;
        while(j < r || dcmp(cross(v, p[j%n] - p[i]) -
            cross(v, p[(j+1)%n] - p[i])) < 0) j++;
        while(1 < j || dcmp(dot(v, p[l%n] - p[i]) -
            dot(v, p[(l+1)%n] - p[i])) > 0) l++;
        Tf w = dot(v, p[r%n]-p[i])-dot(v, p[l%n]-p[i]);
        Tf h = distancePointLine(p[j%n],
            Line(p[i], p[(i+1)%n]));
        area = min(area, w * h);
        perimeter = min(perimeter, 2 * w + 2 * h);
    } return make_pair(area, perimeter);
}
/// returns the left half of u on left on ray ab
Polygon cutPolygon(Polygon u, PT a, PT b) {
    using Linear::lineLineIntersect;
    using Linear::onSegment;
    Polygon ret; int n = u.size();
    for(int i = 0; i < n; i++) {
        PT c = u[i], d = u[(i + 1) % n];
        if(dcmp(cross(b-a, c-a))>=0) ret.push_back(c);
        if(dcmp(cross(b-a, d-c)) != 0) {
            PT t; lineLineIntersect(a, b-a, c, d-c, t);

```

```

        if(onSegment(t,Segment(c,d)))ret.push_back(t);
    }
} return ret;
}
bool pointInTriangle(PT a, PT b, PT c, PT p) {
    return dcmp(cross(b - a, p - a)) >= 0
        && dcmp(cross(c - b, p - b)) >= 0
        && dcmp(cross(a - c, p - c)) >= 0;
}
int pointInConvexPolygon(const Polygon &pt, PT p){
    int n = pt.size(); assert(n >= 3);
    int lo = 1, hi = n - 1;
    while(hi - lo > 1) {
        int mid = (lo + hi) / 2;
        if(dcmp(cross(pt[mid]-pt[0], p - pt[0])) > 0)
            lo = mid;
        else hi = mid;
    }
    bool in=pointInTriangle(pt[0],pt[lo],pt[hi],p);
    if(!in) return 1;
    if(dcmp(cross(pt[lo]-pt[lo-1],p-pt[lo-1]))==0)
        return 0; if(dcmp(cross(pt[hi]-pt[lo],
        p-pt[lo]))==0) return 0; if(dcmp(cross(pt[hi]-
        pt[(hi+1)%n], p-pt[(hi+1)%n]))==0) return 0;
    return -1;
}
/// most extreme Point in the direction u
int extremePoint(const Polygon &poly, PT u) {
    int n = (int) poly.size();
    int a = 0, b = n;
    while(b - a > 1) {
        int c = (a + b) / 2;
        if(dcmp(dot(poly[c]-poly[(c+1)%n], u))>=0 &&
            dcmp(dot(poly[c]-poly[(c-1)%n], u))>=0) {
            return c;
        }
        bool a_up=dcmp(dot(poly[(a+1)%n]-poly[a],u))>=0;
        bool c_up=dcmp(dot(poly[(c+1)%n]-poly[c],u))>=0;
        bool a_above_c=dcmp(dot(poly[a]-poly[c],u))>0;
        if(a_up && !c_up) b = c;
        else if(!a_up && c_up) a = c;
        else if(a_up && c_up) {
            if(a_above_c) b = c; else a = c;
        } else {
            if(!a_above_c) b = c; else a = c;
        }
    }
    if(dcmp(dot(poly[a]-poly[(a+1)%n],u))>0 &&
        dcmp(dot(poly[a]-poly[(a-1)%n],u))>0)
        return a;
    return b % n;
}
/// return list of segs of p that touch/intersect l
/// the i'th segment is (p[i], p[(i + 1)%p])
/// #1 If a side is collinear, only that returned
/// #2 If l goes through p[i], ith segment is added
vector<int> lineConvexPolyIntersection(
    const Polygon &p, Line l) {
    assert((int) p.size() >= 3); assert(l.a != l.b);
    int n = p.size(); vector<int> ret;
    PT v = l.b - l.a;
    int lf = extremePoint(p, rotate90(v));

```

```

    int rt = extremePoint(p, rotate90(v) * Ti(-1));
    int olf = orient(l.a, l.b, p[lf]);
    int ort = orient(l.a, l.b, p[rt]);
    if(!olf || !ort) {
        int idx = (!olf ? lf : rt);
        if(orient(l.a, l.b, p[(idx - 1 + n) % n])==0)
            ret.push_back((idx - 1 + n) % n);
        else ret.push_back(idx);
        return ret;
    }
    if(olf == ort) return ret;
    for(int i=0; i<2; ++i) {
        int lo = i ? rt : lf, hi = i ? lf : rt;
        int olo = i ? ort : olf;
        while(true) {
            int gap = (hi - lo + n) % n;
            if(gap < 2) break;
            int mid = (lo + gap / 2) % n;
            int omid = orient(l.a, l.b, p[mid]);
            if(!omid) {lo = mid; break;}
            if(omid == olo) lo = mid;
            else hi = mid;
        } ret.push_back(lo);
    } return ret;
}
/// [ACW, CW] tangent pair from an external point
constexpr int CW = -1, ACW = 1;
bool isGood(PT u, PT v, PT Q, int dir) {
    return orient(Q, u, v) != -dir;
}
PT better(PT u, PT v, PT Q, int dir) {
    return orient(Q, u, v) == dir ? u : v;
}
PT pointPolyTangent(const Polygon &pt, PT Q,
    int dir, int lo, int hi) {
    while(hi - lo > 1) {
        int mid = (lo + hi) / 2;
        bool pvs = isGood(pt[mid], pt[mid-1], Q, dir);
        bool nxt = isGood(pt[mid], pt[mid+1], Q, dir);
        if(pvs && nxt) return pt[mid];
        if(!pvs || !nxt) {
            PT p1 = pointPolyTangent(pt,Q,dir,mid+1,hi);
            PT p2 = pointPolyTangent(pt,Q,dir,lo,mid-1);
            return better(p1, p2, Q, dir);
        }
        if(!pvs) {
            if(orient(Q,pt[mid],pt[lo])==dir) hi=mid-1;
            else if(better(pt[lo],pt[hi],Q,dir)==pt[lo])
                hi = mid - 1; else lo = mid + 1;
        }
        if(!nxt) {
            if(orient(Q,pt[mid],pt[hi])==dir) lo=mid+1;
            else if(better(pt[lo],pt[hi],Q,dir)==pt[lo])
                hi = mid - 1; else lo = mid + 1;
        }
    }
    PT ret = pt[lo];
    for(int i = lo + 1; i <= hi; i++)
        ret = better(ret, pt[i], Q, dir);
    return ret;
}
/// [ACW, CW] Tangent
pair<PT,PT> pointPolyTangents(
    const Polygon &pt,PT Q) {

```

```

int n = pt.size();
PT acw_tan = pointPolyTangent(pt, Q, ACW, 0, n-1);
PT cw_tan = pointPolyTangent(pt, Q, CW, 0, n-1);
return make_pair(acw_tan, cw_tan);
} }
#####
namespace Circular {
// returns intersections in order of ray (l.a, l.b)
vector<PT> circleLineIntersection(Circle c, Line l){
    static_assert(is_same<Tf, Ti>::value);
    vector<PT> ret;
    PT b = l.b - l.a, a = l.a - c.o;
    Tf A = dot(b, b), B = dot(a, b);
    Tf C = dot(a, a) - c.r * c.r, D = B*B - A*C;
    if (D < -EPS) return ret;
    ret.push_back(l.a + b * (-B - sqrt(D + EPS)) / A);
    if (D > EPS)
        ret.push_back(l.a + b * (-B + sqrt(D)) / A);
    return ret;
}
// circle(c.o, c.r) x triangle(c.o, s.a, s.b) (ccw)
Tf circleTriInterArea(Circle c, Segment s){
    using Linear::distancePointSegment;
    Tf OA = length(c.o - s.a), OB = length(c.o - s.b);
    if(dcmp(distancePointSegment(c.o, s) - c.r) >= 0)
        return angleBetween(s.a - c.o, s.b - c.o) * c.r * c.r / 2;
    if(dcmp(OA - c.r) <= 0 && dcmp(OB - c.r) <= 0)
        return cross(c.o - s.b, s.a - s.b) / 2.0;
    vector<PT> Sect = circleLineIntersection(c, s);
    return circleTriInterArea(c, Segment(s.a, Sect[0]))
        + circleTriInterArea(c, Segment(Sect[0], Sect[1]))
        + circleTriInterArea(c, Segment(Sect[1], s.b));
}
Tf circlePolyIntersectionArea(Circle c, Polygon p){
    Tf res = 0;
    int n = p.size();
    for(int i = 0; i < n; ++i)
        res += circleTriInterArea(c,
            Segment(p[i], p[(i + 1) % n]));
    return abs(res);
}
// locates circle c2 relative to c1: intersect = 0
// inside = -2, inside touch = -1,
// outside touch = 1, outside = 2
int circleCirclePosition(Circle c1, Circle c2) {
    Tf d = length(c1.o - c2.o);
    int in = dcmp(d - abs(c1.r - c2.r)),
        ex = dcmp(d - (c1.r + c2.r));
    return in < 0 ? -2 : in == 0 ? -1 : ex == 0 ? 1 : ex > 0 ? 2 : 0;
}
vector<PT> circleCircleInter(Circle c1, Circle c2){
    static_assert(is_same<Tf, Ti>::value);
    vector<PT> ret;
    Tf d = length(c1.o - c2.o);
    if(dcmp(d) == 0) return ret;
    if(dcmp(c1.r + c2.r - d) < 0) return ret;
    if(dcmp(abs(c1.r - c2.r) - d) > 0) return ret;
    PT v = c2.o - c1.o;
    Tf co = (c1.r * c1.r + sqLength(v) - c2.r * c2.r)
        / (2 * c1.r * length(v));
    Tf si = sqrt(abs(1.0 - co * co));
    PT p1 = scale(rotatePrecise(v, co, -si), c1.r) + c1.o;

```

```

PT p2 = scale(rotatePrecise(v, co, si), c1.r) + c1.o;
ret.push_back(p1);
if(p1 != p2) ret.push_back(p2); return ret;
}
Tf circleCircleInterArea(Circle c1, Circle c2) {
    PT AB = c2.o - c1.o; Tf d = length(AB);
    if(d >= c1.r + c2.r) return 0;
    if(d + c1.r <= c2.r) return PI * c1.r * c1.r;
    if(d + c2.r <= c1.r) return PI * c2.r * c2.r;
    Tf alpha1 = acos((c1.r * c1.r + d * d - c2.r * c2.r)
        / (2.0 * c1.r * d));
    Tf alpha2 = acos((c2.r * c2.r + d * d - c1.r * c1.r)
        / (2.0 * c2.r * d));
    return c1.sector(2 * alpha1) + c2.sector(2 * alpha2);
}
// returns tangents from a point p to circle c
vector<PT> pointCircleTangents(PT p, Circle c) {
    static_assert(is_same<Tf, Ti>::value);
    vector<PT> ret; PT u = c.o - p; Tf d = length(u);
    if(d < c.r) ;
    else if(dcmp(d - c.r) == 0) {
        ret = { rotate(u, PI / 2) };
    }
    else {
        Tf ang = asin(c.r / d);
        ret = { rotate(u, -ang), rotate(u, ang) };
    }
    return ret;
}
// returns points on tangents that touches circle c
vector<PT> pointCircleTangencyPoints(PT p, Circle c){
    static_assert(is_same<Tf, Ti>::value);
    PT u = p - c.o; Tf d = length(u);
    if(d < c.r) return {};
    else if(dcmp(d - c.r) == 0) return {c.o + u};
    else {
        Tf ang = acos(c.r / d); u = u / length(u) * c.r;
        return {c.o + rotate(u, -ang), c.o + rotate(u, ang)};
    }
}
// finds a, b st a[i] on c1, b[i] on c2, Segment
// a[i], b[i] touches c1, c2. if c1, c2 touch at x
// (x, x) is also returned, -1 returned if c1 = c2
int circleCircleTangencyPoints(Circle c1, Circle c2,
    vector<PT> &a, vector<PT> &b) {
    a.clear(), b.clear(); int cnt = 0;
    if(dcmp(c1.r - c2.r) < 0) {swap(c1, c2); swap(a, b);}
    Tf d2 = sqLength(c1.o - c2.o);
    Tf rdif = c1.r - c2.r, rsum = c1.r + c2.r;
    if(dcmp(d2 - rdif * rdif) < 0) return 0;
    if(dcmp(d2) == 0 && dcmp(c1.r - c2.r) == 0) return -1;
    Tf base = angle(c2.o - c1.o);
    if(dcmp(d2 - rdif * rdif) == 0) {
        a.push_back(c1.point(base));
        b.push_back(c2.point(base));
        cnt++; return cnt;
    }
    Tf ang = acos((c1.r - c2.r) / sqrt(d2));
    a.push_back(c1.point(base + ang));
    b.push_back(c2.point(base + ang)); cnt++;
    a.push_back(c1.point(base - ang));
    b.push_back(c2.point(base - ang)); cnt++;
    if(dcmp(d2 - rsum * rsum) == 0) {
        a.push_back(c1.point(base));

```

```

        b.push_back(c2.point(PI + base)); cnt++;
    }
    else if(dcmp(d2 - rsum * rsum) > 0) {
        Tf ang = acos((c1.r + c2.r) / sqrt(d2));
        a.push_back(c1.point(base + ang));
        b.push_back(c2.point(PI + base + ang)); cnt++;
        a.push_back(c1.point(base - ang));
        b.push_back(c2.point(PI + base - ang)); cnt++;
    }
    return cnt;
} } // namespace Circular
#####
namespace EnclosingCircle{
// returns false if points are collinear
bool inCircle(PT a, PT b, PT c, Circle &p) {
    using Linear::distancePointLine;
    static_assert(is_same<Tf, Ti>::value);
    if(orient(a, b, c) == 0) return false;
    Tf u=length(b-c), v=length(c-a), w=length(a-b);
    p.o = (a * u + b * v + c * w) / (u + v + w);
    p.r = distancePointLine(p.o, Line(a, b));
    return true;
}
// set of points A(x, y) st PA : QA = rp : rq
Circle apolloniusCircle(PT P, PT Q, Tf rp, Tf rq){
    static_assert(is_same<Tf, Ti>::value);
    rq *= rq; rp *= rp; Tf a=rq-rp; assert(dcmp(a));
    Tf g = (rq * P.x - rp * Q.x) / a, h = (rq * P.y - rp * Q.y) / a;
    Tf c = (rq * P.x * P.x - rp * Q.x * Q.x +
        rq * P.y * P.y - rp * Q.y * Q.y) / a;
    PT o(g, h); Tf R = sqrt(g * g + h * h - c);
    return Circle(o, R);
}
// returns false if points are collinear
bool circumCircle(PT a, PT b, PT c, Circle &p) {
    using Linear::lineLineIntersect;
    if(orient(a, b, c) == 0) return false;
    PT d = (a + b) / 2, e = (a + c) / 2;
    PT vd = rotate90(b - a), ve = rotate90(a - c);
    bool f = lineLineIntersect(d, vd, e, ve, p.o);
    if(f) p.r = length(a - p.o);
    return f;
}
// finds a circle that goes all of p, |p| <= 3.
Circle boundary(const vector<PT> &p) {
    Circle ret; int sz = p.size();
    if(sz == 0) ret.r = 0;
    else if(sz == 1) ret.o = p[0], ret.r = 0;
    else if(sz == 2) ret.o = (p[0] + p[1]) / 2,
        ret.r = length(p[0] - p[1]) / 2;
    else if(!circumCircle(p[0], p[1], p[2], ret))
        ret.r = 0;
    return ret;
}
// Min circle enclosing p[fr...n-1],
// with points in b on the boundary, |b| <= 3.
Circle welzl(const vector<PT> &p,
    int fr, vector<PT> &b) {
    if(fr >= (int) p.size() || b.size() == 3)
        return boundary(b);
    Circle c = welzl(p, fr + 1, b);
    if(!c.contains(p[fr])) {
        b.push_back(p[fr]); c = welzl(p, fr + 1, b);

```



```

    b.pop_back();
} return c;
}
// MEC of p, using weizl's algo. amortized O(n).
Circle MEC(vector<PT> p) {
    random_shuffle(p.begin(), p.end());
    vector<PT> q; return weizl(p, 0, q);
}
#####
// Given list of segments v, finds a pair (i, j) st
// v[i],v[j] intersects. If none, returns {-1, -1}
namespace IntersectingSegments {
    struct Event {
        Tf x; int tp, id;
        bool operator < (const Event &p) const {
            if(dcmp(x-p.x)) return x<p.x; return tp>p.tp;
        }
    };
    pair<int, int> anyInters(const vector<Segment> &v){
        using Linear::segmentsIntersect;
        static_assert(is_same<Tf, Ti>::value);
        vector<Event> ev;
        for(int i=0; i<v.size(); i++) {
            ev.push_back({min(v[i].a.x, v[i].b.x), +1, i});
            ev.push_back({max(v[i].a.x, v[i].b.x), -1, i});
        }
        sort(ev.begin(), ev.end());
        auto comp = [&v](int i, int j) {
            Segment p = v[i], q = v[j];
            Tf x=max(min(p.a.x,p.b.x), min(q.a.x, q.b.x));
            auto yvalSegment = [&x](const Line &s) {
                if(dcmp(s.a.x - s.b.x) == 0) return s.a.y;
                return s.a.y + (s.b.y - s.a.y)
                    * (x - s.a.x) / (s.b.x - s.a.x);
            };
            return dcmp(yvalSegment(p)-yvalSegment(q))<0;
        };
        multiset<int, decltype(comp)> st(comp);
        typedef decltype(st)::iterator iter;
        auto prev = [&st](iter it) {
            return it == st.begin() ? st.end() : --it;
        };
        auto next = [&st](iter it) {
            return it == st.end() ? st.end() : ++it;
        };
        vector<iter> pos(v.size());
        for(auto &cur : ev) {
            int id = cur.id;
            if(cur.tp == 1) {
                iter nxt = st.lower_bound(id), pre=prev(nxt);
                if(pre != st.end() && segmentsIntersect
                    (v[*pre], v[id])) return {*pre, id};
                if(nxt != st.end() && segmentsIntersect
                    (v[*nxt], v[id])) return {*nxt, id};
                pos[id] = st.insert(nxt, id);
            }
            else {
                iter nxt=next(pos[id]), pre=prev(pos[id]);
                if(pre != st.end() && nxt != st.end() &&
                    segmentsIntersect(v[*pre], v[*nxt]))
                    return {*pre, *nxt};
                st.erase(pos[id]);
            }
        }
    }
}

```

```

    }
    return {-1, -1};
}
}
#####
namespace HalfPlanar {
    using Linear::lineLineIntersect;
    struct DirLine {
        PT p, v; Tf ang;
        DirLine() {}
        DirLine(PT p, PT v) : p(p), v(v) {
            ang = atan2(v.y, v.x);
        }
        // Directed line for ax+by+c >=0
        DirLine(Tf a, Tf b, Tf c) {
            assert(dcmp(a) || dcmp(b));
            p = dcmp(a) ? PT(-c/a, 0) : PT(0, -c/b);
            v = PT(b, -a); ang = atan2(v.y, v.x);
        }
        bool operator<(const DirLine& u) const {
            return ang < u.ang;
        }
        bool onLeft(PT x) const {
            return dcmp(cross(v, x-p)) >= 0;
        }
    };
    // region bounded by the left side of dir lines
    // OUTPUT IS UNDEFINED if intersection is unbounded
    // O(n log n) for sorting, O(n) afterwards
    Polygon halfPlaneIntersection(vector<DirLine> li) {
        int n = li.size(), first = 0, last = 0;
        sort(li.begin(), li.end());
        vector<PT> p(n);
        vector<DirLine> q(n);
        q[0] = li[0];
        for(int i = 1; i < n; i++) {
            while(first < last && !li[i].onLeft(p[last-1]))
                last--;
            while(first < last && !li[i].onLeft(p[first]))
                first++;
            q[++last] = li[i];
            if(dcmp(cross(q[last].v, q[last-1].v)) == 0) {
                last--;
                if(q[last].onLeft(li[i].p)) q[last] = li[i];
            }
            if(first < last)
                lineLineIntersect(q[last-1].p, q[last-1].v,
                    q[last].p, q[last].v, p[last-1]);
        }
        while(first < last && !q[first].onLeft(p[last-1]))
            last--;
        if(last - first <= 1) return {};
        lineLineIntersect(q[last].p, q[last].v,
            q[first].p, q[first].v, p[last]);
        return Polygon(p.begin()+first, p.begin()+last+1);
    }
    // O(n^2 lg n) VoronoiDiagram bounded by INF square
    // regions[i] = region with closest = site[i].
    const Tf INF = 1e10;
    vector<Polygon> voronoi(vector<PT> site, Tf bsq) {
        int n = site.size();
        vector<Polygon> region(n);
        PT A(-bsq, -bsq), B(bsq, -bsq),

```

```

        C(bsq, bsq), D(-bsq, bsq);
        for(int i = 0; i < n; ++i) {
            vector<DirLine> li(n-1);
            for(int j = 0, k = 0; j < n; ++j) {
                if(i == j) continue;
                li[k++] = DirLine((site[i] + site[j]) / 2,
                    rotate90(site[j] - site[i]));
            }
            li.emplace_back(A,B-A); li.emplace_back(B,C-B);
            li.emplace_back(C,D-C); li.emplace_back(D,A-D);
            region[i] = halfPlaneIntersection(li);
        }
        return region;
    }
}
#####
namespace PointRotationTrick {
    // define the processor function in this namespace
    // passing lambda as argument performs better
    typedef pair<int, int> PII;
    void performTrick(vector<PT> pts, const function<
        void(const vector<PT> &, int)> &processor) {
        int n = pts.size(); sort(pts.begin(), pts.end());
        vector<int> position(n); vector<PII> segments;
        segments.reserve((n*(n-1))/2);
        for (int i = 0; i < n; i++) {
            position[i] = i;
            for (int j = i+1; j < n; j++) {
                segments.emplace_back(i, j);
            }
        }
        assert(segments.capacity() == segments.size());
        sort(segments.begin(), segments.end(),
            [&](PII p, PII q) {
                Ti prod = cross(pts[p.second]-pts[p.first],
                    pts[q.second]-pts[q.first]);
                if (prod != 0) return prod > 0;
                return p < q;
            });
        for (PII seg : segments) {
            int i = position[seg.first];
            assert(position[seg.second] == i+1);
            processor(pts, i); swap(pts[i], pts[i+1]);
            swap(position[seg.first], position[seg.second]);
        }
    }
}

```

5 Graph

5.1 Bridge

```

const int vmax = 2e5+10, emax = 2e5+10;
namespace Bridge { //edge, nodes, comps 1 indexed
    vector<int> adj[vmax]; // edge-id
    pair<int, int> edges[emax]; // (u, v)
    bool isBridge[emax];
    int visited[vmax]; //0-unvis,1-vising,2-vis
    int st[vmax], low[vmax], clk = 0, edgeId = 0;
    // For bridge tree components
    int who[vmax], compId = 0;
    vector<int> stk;
    // For extra end time calc
    int en[vmax];
    void dfs(int u, int parEdge) {

```



```

visited[u] = 1; low[u] = st[u] = ++clk;
stk.push_back(u);
for (auto e : adj[u]) {
    if (e == parEdge) continue;
    int v=edges[e].first^edges[e].second^u;
    if (visited[v] == 1) {
        low[u] = min(low[u], st[v]);
    } else if (visited[v] == 0) {
        dfs(v, e); low[u] = min(low[u], low[v]);
    }
}
visited[u] = 2;
if(st[u] == low[u]){/// found
    ++compId; int cur;
    do{
        cur = stk.back(); stk.pop_back();
        who[cur] = compId;
    }while(cur != u);
    if(parEdge != -1){isBridge[parEdge] = true;}
}
en[u] = clk;
}
void clearAll(int n){
    for(int i = 0; i<=n; i++) {
        adj[i].clear(); visited[i] = st[i] = 0; }
    for(int i = 0; i<=edgeId; i++) isBridge[i]=0;
    clk = compId = edgeId = 0;
}
void findBridges(int n){
    for(int i = 1; i<=n; i++){
        if(visited[i] == 0) dfs(i, -1); }
}
bool isReplacable(int eid, int u, int v){
    if(!isBridge[eid]) return true;
    int a=edges[eid].first,b=edges[eid].second;
    if(st[a] > st[b]) swap(a, b);
    return (st[b] <= st[u] && st[u] <= en[b])
    != (st[b] <= st[v] && st[v] <= en[b]);
}
void addEdge(int u, int v){
    edgeId++; edges[edgeId] = {u, v};
    adj[u].emplace_back(edgeId);
    adj[v].emplace_back(edgeId);
}
}

```

5.2 Cutpoint

```

const int vmax = 1e4+10, emax = 1e5+10;
namespace Cutpoint { /// For BCTree, no self edge
/// edge, nodes, components 1-indexed
vector<int> adj[vmax]; ///edge-id
pair<int, int> edges[emax]; /// (u, v)
bool isCutpoint[vmax];
int visited[vmax];///0-unvis, 1-vising, 2-vis
int st[vmax], low[vmax], clk = 0, edgeId = 0;
/// For block components (i.e. edges)
int who[emax], compId = 0;
vector<int> stk;
/// For extra end time calc
int en[vmax];
void dfs(int u, int parEdge) {
    visited[u] = 1; low[u] = st[u] = ++clk;

```

```

    int ch_cnt = (parEdge != -1);
    for (auto e : adj[u]) {
        if (e == parEdge) continue;
        int v = edges[e].first^edges[e].second^u;
        if (visited[v] == 1) {
            stk.push_back(e);
            low[u] = min(low[u], st[v]);
        } else if (visited[v] == 0) {
            stk.push_back(e); dfs(v, e);
            low[u] = min(low[u], low[v]);
            if(low[v] >= st[u]){
                ++ch_cnt; ++compId;
                int cur;
                do{
                    cur = stk.back(); stk.pop_back();
                    who[cur] = compId;
                }while(cur != e);
            }
        }
    }
    visited[u] = 2;
    if(ch_cnt > 1){ isCutpoint[u] = true;}
    en[u] = clk;
}
void clearAll(int n){
    for(int i = 0; i<=n; i++) {
        adj[i].clear(); visited[i] = st[i] = 0; }
    for(int i=0;i<=n;i++)isCutpoint[i]=false;
    clk = compId = edgeId = 0;
}
void findCutpoints(int n){
    for(int i = 1; i<=n; i++){
        if(visited[i] == 0) dfs(i, -1); }
}
void addEdge(int u, int v){
    edgeId++; edges[edgeId] = {u, v};
    adj[u].emplace_back(edgeId);
    adj[v].emplace_back(edgeId);
}
}

```

5.3 DominatorTree

```

typedef vector<int> VI; typedef vector<VI> VVI;
struct ChudirBhai { ///1-indexed
int n, T; VVI g, tree, rg, bucket;
VI sdom, par, dom, dsu, label, arr, rev;
ChudirBhai(int n): n(n),g(n+1),tree(n+1), rg(n+1),
    bucket(n+1), sdom(n+1), par(n+1),dom(n+1),
    dsu(n+1),label(n+1),arr(n+1),rev(n+1),T(0){
    for(int i = 1; i <= n; i++)
        sdom[i] = dom[i] = dsu[i] = label[i] = i;
}
void addEdge(int u, int v) { g[u].push_back(v); }
void dfs0(int u) {
    T++; arr[u] = T, rev[T] = u;
    label[T] = T, sdom[T] = T, dsu[T] = T;
    for(int i = 0; i < g[u].size(); i++) {
        int w = g[u][i];
        if(!arr[w]) dfs0(w), par[arr[w]] = arr[u];
        rg[arr[w]].push_back(arr[u]);
    }
}
}

```

```

int Find(int u, int x = 0) {
    if(u == dsu[u]) return x? -1: u;
    int v = Find(dsu[u], x+1);
    if(v < 0) return u;
    if(sdom[label[dsu[u]]] < sdom[label[u]])
        label[u] = label[dsu[u]];
    dsu[u] = v; return x? v: label[u];
}
void Union(int u, int v) { dsu[v] = u; }
VVI buildAndGetTree(int s) {
    dfs0(s);
    for(int i = n; i >= 1; i--) {
        for(int j = 0; j < rg[i].size(); j++)
            sdom[i] = min(sdom[i], sdom[Find(rg[i][j])]);
        if(i > 1) bucket[sdom[i]].push_back(i);
        for(int j = 0; j < bucket[i].size(); j++) {
            int w = bucket[i][j], v = Find(w);
            if(sdom[v] == sdom[w]) dom[w] = sdom[w];
            else dom[w] = v;
        }
        if(i > 1) Union(par[i], i);
    }
    for(int i = 2; i <= n; i++) {
        if(dom[i] != sdom[i]) dom[i] = dom[dom[i]];
        tree[rev[i]].push_back(rev[dom[i]]);
        tree[rev[dom[i]]].push_back(rev[i]);
    }
    return tree;
}
//Idom(u) = 0 if u is unreach, IDom(root) = root;
int getIDom(int u) {return rev[dom[arr[u]]];}
};

```

5.4 EulerianTour

```

const int vmax = 1e5+10, emax = 2e5+10;
namespace Euler { ///nodes, edges 1 indexed [1, n]
/// call clear(vertex count) to clear stuff
pair<int, int> edges[emax];
bool used[emax]; /// used edges
int ecnt = 0;
vector<int> adj[vmax];
int ptr[vmax]; /// curr pointer at adjlist
void addEdge(int u, int v, const bool&
    directed = false){
    edges[++ecnt] = {u, v};
    adj[u].push_back(ecnt);
    if (!directed) adj[v].push_back(ecnt);
}
vector<pair<int, int>> stk; /// (node, edge)
vector<int> eulerTour(int st=edges[ecnt].first){
    vector<int> tour;
    if(ecnt == 0) return tour;
    stk.emplace_back(st, -1);
    while(!stk.empty()){
        int u = stk.back().first;
        for(int &i = ptr[u]; i<adj[u].size(); i++){
            int e = adj[u][i];
            if(used[e]) continue;
            used[e] = true;
            int v = u^edges[e].first^edges[e].second;
            stk.emplace_back(v, e); break;
        }
    }
}

```

```

    if(ptr[u] == adj[u].size()){
        tour.push_back(stk.back().second);
        stk.pop_back();
    }
    tour.pop_back();
    reverse(tour.begin(), tour.end());
    return tour;
}
vector<int> eulerPath(int st, int en){
    addEdge(en, st, true);
    auto path = eulerTour(st);
    if(ecnt != path.back()) {
        auto it=find(path.begin(),path.end(),ecnt)+1;
        rotate(path.begin(), it, path.end());
    }
    path.pop_back(); --ecnt;
    adj[en].pop_back(); adj[st].pop_back();
    return path;
}
void clear(int n){
    for(int i=0;i<=n;i++)adj[i].clear(),ptr[i]=0;
    for(int i = 0; i<=ecnt; i++) used[i] = false;
    ecnt = 0;
}
}

```

5.5 MatroidIntersection

```

/** Mat Intersection per increment O(r*n) Weighted
Mat Intersection: per increment O(r^2*n); Evrythng
0-indexed erase base Mat to get better runtime*/
typedef pair<int,int>PII; typedef vector<int>VI;
typedef vector<bool>VB;typedef long long CostType;
const CostType INF=1e18; const int BITSET_BITS=60;
struct Graph {
    vector<VI>edg;
    Graph(int nodes) : edg(nodes) {}
    void addEdge(int u,int v){ edg[u].push_back(v);}
    void clearGraph(){
        for (int i=0; i<edg.size();i++)edg[i].clear();
    }
};
struct Mat {
    virtual void updTknElmnt(const VB &tkn) = 0;
    virtual bool canTkElmnt(int e) = 0;
    virtual bool canEx(int rmv, int ins) = 0;
};
struct ColorMat : Mat {
    VI elmntCol, canTakeAtMost, curTaking;
    int elmnts, clrs;
    ColorMat(int elmnts, int clrs):elmnts(elmnts),
        clrs(clrs), canTakeAtMost(clrs,1),
        curTaking(clrs,0) {elmntCol.reserve(elmnts);}
    void updTknElmnt(const VB &tkn) {
        fill(curTaking.begin(), curTaking.end(), 0);
        for (int i = 0; i < elmnts; i++) if (tkn[i])
            curTaking[ elmntCol[i] ]++;
    }
    bool canTkElmnt(int e) {
        int col = elmntCol[e];
        return curTaking[col] != canTakeAtMost[col];
    }
}

```

```

bool canEx(int rmv, int ins) {
    int colr = elmntCol[rmv],coli = elmntCol[ins];
    if (coli == colr) return true;
    return curTaking[coli] != canTakeAtMost[coli];
}
};
struct GraphMat : Mat {
    vector< PII >ajs; int elmnts, grSz;
    GraphMat(int elmnts, int grSz) : forest(grSz) {
        this->elmnts = elmnts; this->grSz = grSz;
        ajs.reserve(elmnts);
    }
    Graph forest; VI start, finish, root;
    void treeDFS(int u, int p, int &tym) {
        start[u] = ++tym;
        for (int v : forest.edg[u]) {
            if (v == p) continue;
            root[v] = root[u]; treeDFS(v, u, tym);
        }
        finish[u] = tym;
    }
    bool inSubtree(int u, int x) {
        return start[u]<=start[x]&&finish[x]<=finish[u];
    }
    void updTknElmnt(const VB &tkn) {
        forest.clearGraph();
        for (int i = 0; i < elmnts; i++) if (tkn[i]) {
            forest.addEdge(ajs[i].first, ajs[i].second);
            forest.addEdge(ajs[i].second, ajs[i].first);
        }
        root=VI(grSz, -1); finish=start=VI(grSz,0);
        int tym = -1;
        for (int i = 0; i < grSz; i++)if(root[i]==-1){
            root[i] = i; treeDFS(i, -1, tym);
        }
    }
    bool canTkElmnt(int e) {
        return root[ajs[e].first]!=root[ajs[e].second];
    }
    bool canEx(int rmv, int ins) {
        if (canTkElmnt(ins)) return true;
        int u = ajs[rmv].first, p = ajs[rmv].second;
        if (start[p] > start[u]) swap(u, p);
        int x = ajs[ins].first, y = ajs[ins].second;
        return inSubtree(u, x) != inSubtree(u, y);
    }
};
struct BinMat : Mat {
    typedef bitset< BITSET_BITS > BitSet;
    struct Basis {
        int bitCnt;
        vector< BitSet >rdcd, combi;
        Basis(int bitCnt):bitCnt(bitCnt),rdcd(bitCnt),
            combi(bitCnt) {
            assert(BITSET_BITS == bitCnt);
        }
        void clearAll() {
            for (int i = 0; i < bitCnt; i++) {
                rdcd[i].reset(); combi[i].reset();
            }
        }
        BitSet canBeBuiltWith(BitSet x) {

```

```

        BitSet rt;
        for (int i = bitCnt-1; i >= 0; i--)
            if (x.test(i)) {
                if (!rdcd[i].test(i)) return BitSet();
                x ^= rdcd[i]; rt ^= combi[i];
            }
        return rt;
    }
    int addVector(BitSet x) {
        BitSet cm;
        for (int i = bitCnt-1; i >= 0; i--)
            if (x.test(i)) {
                if (!rdcd[i].test(i)) {
                    rdcd[i] = x;combi[i] = cm.set(i);return i;
                } else {
                    x ^= rdcd[i]; cm ^= combi[i];
                }
            }
        return -1;
    }
};
vector<BitSet> rows;int elmnts, bitCnt;
BinMat(int elmnts, int bitCnt) : elmnts(elmnts),
    bitCnt(bitCnt), curBas(bitCnt), cycle(elmnts),
    rowMap(elmnts) { rows.reserve(elmnts);
}
vector< BitSet >cycle;
VI rowMap;
Basis curBas;
void updTknElmnt(const VB &tkn) {
    curBas.clearAll();
    for (int i = 0; i < elmnts; i++) if (tkn[i]) {
        rowMap[i] = curBas.addVector(rows[i]);
    }
    for (int i = 0; i < elmnts; i++) if (!tkn[i]){
        cycle[i] = curBas.canBeBuiltWith(rows[i]);
    }
}
bool canTkElmnt(int e){ return !cycle[e].any();}
bool canEx(int rmv, int ins) {
    if (canTkElmnt(ins)) return true;
    return cycle[ins].test( rowMap[rmv] );
}
};
struct GrafDual : Mat {
    struct Bridge {
        vector< VI >adj; vector< PII >ajs;
        VB isBridge; VI visited, st, en, low;
        int clk = -1, edgeId = 0;
        Bridge(int emax, int vmax) : adj(vmax),
            isBridge(emax), visited(vmax), st(vmax),
            en(vmax), low(vmax) { ajs.reserve(emax); }
        void clearAll() {
            int n = adj.size();
            for(int i = 0; i < n; i++) {
                adj[i].clear(); visited[i] = st[i] = 0;
            }
            for(int i=0;i<edgeId;i++)isBridge[i]=false;
            clk = -1; edgeId = 0;
        }
        void dfs(int u, int parEdge) {
            visited[u] = 1; low[u] = st[u] = ++clk;
            for (int e : adj[u]) {

```

```

    if (e == parEdge) continue;
    int v = ajs[e].first ^ ajs[e].second ^ u;
    if (visited[v] == 1) {
        low[u] = min(low[u], st[v]);
    } else if (visited[v] == 0) {
        dfs(v, e); low[u] = min(low[u], low[v]);
    }
    if (st[u] == low[u] && parEdge != -1) {
        isBridge[parEdge] = true;
    }
    en[u] = clk; visited[u] = 2;
}
void fndBriz() {
    int n = adj.size();
    for (int i = 0; i < n; i++) {
        if (visited[i] == 0) dfs(i, -1);
    }
}
bool isRplc(int eid, int u, int v) {
    if (!isBridge[eid]) return true;
    int a = ajs[eid].first, b = ajs[eid].second;
    if (st[a] > st[b]) swap(a, b);
    return (st[b] <= st[u] && st[u] <= en[b])
        != (st[b] <= st[v] && st[v] <= en[b]);
}
int addEdge(int u, int v) {
    ajs[edgeId] = {u, v};
    adj[u].emplace_back(edgeId);
    adj[v].emplace_back(edgeId);
    return edgeId++;
}
};
vector< PII >ajs; int elmnts, grSz;
GrafDual(int elmnts, int grSz) : bridge(elmnts,
    grSz), edgeMap(elmnts) {
    this->elmnts = elmnts; this->grSz = grSz;
    ajs.reserve(elmnts);
}
Bridge bridge; VI edgeMap;
void updTkElmnt(const VB &tkn) {
    bridge.clearAll();
    for (int i = 0; i < elmnts; i++) if (!tkn[i]) {
        edgeMap[i] = bridge.addEdge(ajs[i].first,
            ajs[i].second);
    }
    bridge.fndBriz();
}
bool canTkElmnt(int e) {
    return !bridge.isBridge[ edgeMap[e] ];
}
bool canEx(int rmv, int ins) {
    return bridge.isRplc(edgeMap[ins],
        ajs[rmv].first, ajs[rmv].second);
}
};
bool augment(int elmnts, Mat *m1, Mat *m2,
    VB &tkn, const VB &source, const VB &sink) {
    VI parent(elmnts, -2), hidari, migi;
    hidari.reserve(elmnts); migi.reserve(elmnts);
    queue< int >q;
    for (int i = 0; i < elmnts; i++) {

```

```

        if (source[i]) {
            q.push(i); parent[i] = -1;
        }
        if (tkn[i]) hidari.push_back(i);
        else migi.push_back(i);
    }
    int connector = -1;
    while (!q.empty() && connector == -1) {
        int u = q.front(); q.pop();
        auto approach = [&](int v) {
            if (parent[v] == -2) {
                parent[v] = u; q.push(v);
                if (sink[v]) connector = v;
            }
        };
        if (tkn[u]) {
            for (int v : migi) if (m1->canEx(u, v))
                approach(v);
        } else {
            for (int v : hidari) if (m2->canEx(v, u))
                approach(v);
        }
    }
    if (connector == -1) return false;
    while (connector != -1) {
        tkn[connector] = tkn[connector] ^ 1;
        connector = parent[connector];
    }
    return true;
}
VB getBasisOfIntersection(int elmnts, Mat *m1,
    Mat *m2) {
    VB tkn(elmnts, false);
    while (true) {
        m1->updTkElmnt(tkn); m2->updTkElmnt(tkn);
        bool trivial=false, noSource=true, noSink=true;
        VB source(elmnts, false), sink(elmnts, false);
        for (int i = 0; i < elmnts; i++) {
            if (tkn[i]) continue;
            if (m1->canTkElmnt(i)) {
                source[i] = true; noSource = false;
            }
            if (m2->canTkElmnt(i)) {
                sink[i] = true; noSink = false;
            }
            if (source[i] && sink[i]) {
                tkn[i] = true; trivial = true; break;
            }
        }
        if (trivial) continue; if (noSource || noSink) break;
        if (!augment(elmnts, m1, m2, tkn, source, sink)) break;
    }
    return tkn;
}
VI findEdgeDisjointSpanningTrees(const
    vector<PII>&ajs, int nodes, int trees) {
    int elmnts = ajs.size()*trees;
    GraphMat gm(elmnts, nodes*trees);
    ColorMat cm(elmnts, ajs.size());
    for (int i = 0; i < ajs.size(); i++) {
        PII p = ajs[i];
        for (int j = 0; j < trees; j++) {
            cm.elmntCol.push_back(i); gm.ajs.push_back(p);
            p.first += nodes; p.second += nodes;
        }
    }

```

```

    }
    VB tkn = getBasisOfIntersection(elmnts, &gm, &cm);
    int on = 0; for (bool b : tkn) on += b;
    VI solution(ajs.size(), -1);
    if (on != trees*(nodes-1)) return solution;
    for (int i = 0; i < ajs.size(); i++) {
        for (int j = 0; j < trees; j++)
            if (tkn[i*trees+j]) solution[i] = j;
    }
    return solution;
}
bool weightedAugment(int elmnts, Mat *m1, Mat *m2,
    vector< CostType >costs, VB &tkn,
    const VB &source, const VB &sink) {
    VI parent(elmnts, -2), hidari, migi;
    hidari.reserve(elmnts); migi.reserve(elmnts);
    for (int i = 0; i < elmnts; i++) {
        if (tkn[i]) {
            hidari.push_back(i); costs[i] = -costs[i];
        } else migi.push_back(i);
    }
    vector< PII >exchangeajs;
    for (int u : hidari) {
        for (int v : migi) {
            if (m1->canEx(u, v))
                exchangeajs.emplace_back(u, v);
            if (m2->canEx(u, v))
                exchangeajs.emplace_back(v, u);
        }
    }
    vector< pair<CostType, int> >dist(elmnts,
        make_pair(INF, -1));
    for (int i = 0; i < elmnts; i++) if (source[i]) {
        dist[i] = make_pair(costs[i], 0); parent[i] = -1;
    }
    for (int i = 0; i < elmnts; i++) {
        bool relaxed = false;
        for (PII p : exchangeajs) {
            if (parent[p.first] == -2) continue;
            pair< CostType, int >tmp = dist[p.first];
            tmp.first += costs[p.second]; tmp.second++;
            if (tmp < dist[p.second]) {
                relaxed = true; dist[p.second] = tmp;
                parent[p.second] = p.first;
            }
        }
        if (!relaxed) break;
    }
    int connector = -1;
    for (int i = 0; i < elmnts; i++)
        if (sink[i] && parent[i] != -2) {
            if (connector == -1 || dist[i] < dist[connector])
                connector = i;
        }
    if (connector == -1) return false;
    while (connector != -1) {
        tkn[connector] = tkn[connector] ^ 1;
        connector = parent[connector];
    }
    return true;
}
/// returns rank+1 elmnts, minimum total costs for

```

```

// a independent subset of size: 0, 1, ..., rank
vector< CostType > weightedIntersection(int elmnts,
    Mat *m1, Mat *m2, const vector<CostType>&costs){
    VB tkn(elmnts, false);
    vector< CostType > minTotalCosts;
    while (true) {
        minTotalCosts.push_back(0);
        for (int i = 0; i < elmnts; i++) if (tkn[i]) {
            minTotalCosts.back() += costs[i];
        }
        m1->updTknElmnt(tkn); m2->updTknElmnt(tkn);
        bool noSource = true, noSink = true;
        VB source(elmnts, false), sink(elmnts, false);
        for (int i = 0; i < elmnts; i++) {
            if (tkn[i]) continue;
            if (m1->canTkElmnt(i)) {
                source[i] = true; noSource = false;
            }
            if (m2->canTkElmnt(i)) {
                sink[i] = true; noSink = false;
            }
        }
        if (noSource || noSink) break;
        if (!weightedAugment(elmnts, m1, m2, costs,
            tkn, source, sink)) break;
    }
    return minTotalCosts;
}

void solveURI_Honesty() {
    int n, m, k; int ti = 0;
    while (cin >> n >> m >> k) {
        GraphMat gm(m, n); ColorMat cm(m, k);
        for (int i = 0; i < m; i++) {
            int u, v, k; cin >> u >> v >> k;
            gm.ajs.emplace_back(u-1, v-1);
            cm.elmntCol.push_back(k-1);
        }
        VB basis = getBasisOfIntersection(m, &gm, &cm);
        int tkn = 0; for (bool b : basis) tkn += b;
        cout << "Instancia " << ++ti << "\n";
        if (tkn == n-1) cout << "sim" << "\n";
        else cout << "nao" << "\n";
        cout << "\n";
    }
}

```

5.6 OnlineBridge

```

vector<int> par, dsu_2ecc, dsu_cc, dsu_cc_size;
int bridges, lca_iteration;
vector<int> last_visit;
void init(int n) {
    par.resize(n); dsu_2ecc.resize(n);
    dsu_cc.resize(n); dsu_cc_size.resize(n);
    lca_iteration = 0; last_visit.assign(n, 0);
    for (int i=0; i<n; ++i) {
        dsu_2ecc[i] = i; dsu_cc[i] = i;
        dsu_cc_size[i] = 1; par[i] = -1;
    }
    bridges = 0;
}

int find_2ecc(int v) {
    if (v == -1) return -1;
    return dsu_2ecc[v] == v ? v : dsu_2ecc[v] =

```

```

    find_2ecc(dsu_2ecc[v]);
}

int find_cc(int v) {
    v = find_2ecc(v);
    return dsu_cc[v] == v ? v : dsu_cc[v] =
        find_cc(dsu_cc[v]);
}

void make_root(int v) {
    v = find_2ecc(v);
    int root = v, child = -1;
    while (v != -1) {
        int p = find_2ecc(par[v]);
        par[v] = child; dsu_cc[v] = root;
        child = v; v = p;
    }
    dsu_cc_size[root] = dsu_cc_size[child];
}

void merge_path (int a, int b) {
    ++lca_iteration;
    vector<int> path_a, path_b; int lca = -1;
    while (lca == -1) {
        if (a != -1) {
            a = find_2ecc(a); path_a.push_back(a);
            if (last_visit[a] == lca_iteration){
                lca = a; break;
            }
            last_visit[a] = lca_iteration; a = par[a];
        }
        if (b != -1) {
            b = find_2ecc(b); path_b.push_back(b);
            if (last_visit[b] == lca_iteration){
                lca = b; break;
            }
            last_visit[b] = lca_iteration; b = par[b];
        }
    }
    for (int v : path_a) {
        dsu_2ecc[v] = lca;
        if (v == lca) break; --bridges;
    }
    for (int v : path_b) {
        dsu_2ecc[v] = lca;
        if (v == lca) break; --bridges;
    }
}

void add_edge(int a, int b) {
    a = find_2ecc(a); b = find_2ecc(b);
    if (a == b) return;
    int ca = find_cc(a), cb = find_cc(b);
    if (ca != cb) {
        ++bridges;
        if (dsu_cc_size[ca] > dsu_cc_size[cb]) {
            swap(a, b); swap(ca, cb);
        }
        make_root(a); par[a] = dsu_cc[a] = b;
        dsu_cc_size[cb] += dsu_cc_size[a];
    } else merge_path(a, b);
}

```

5.7 SCC+2SAT

```

namespace SCC { //Everything 0-indexed.
const int N = 2e6+7; int which[N], vis[N], cc;

```

```

vector<int> adj[N], adjr[N]; vector<int> order;
void addEdge(int u, int v) {
    adj[u].push_back(v); adjr[v].push_back(u);
}

void dfs1(int u){
    if (vis[u]) return; vis[u] = true;
    for(int v: adj[u]) dfs1(v); order.push_back(u);
}

void dfs2(int u, int id) {
    if(vis[u]) return; vis[u] = true;
    for(int v: adjr[u]) dfs2(v, id); which[u] = id;
}

int last = 0;
void findSCC(int n) {
    cc=0, last=n; order.clear(); fill(vis, vis+n, 0);
    for(int i=0; i<n; i++) if(!vis[i]) dfs1(i);
    reverse(order.begin(), order.end());
    fill(vis, vis+n, 0);
    for (int u: order) {
        if (vis[u]) continue; dfs2(u, cc); ++cc;
    }
}

void clear() {
    for (int i=0; i<last; i++)
        adj[i].clear(), adjr[i].clear();
}

struct TwoSat {
    int n; int vars = 0; vector<bool> ans;
    TwoSat(int n) : n(n), ans(n) {
        SCC::clear(); vars = 2*n;
    }

    void implies(int x, int y) {
        SCC::addEdge(x, y); SCC::addEdge(y^1, x^1);
    }

    void OR(int x, int y) {
        SCC::addEdge(x^1, y); SCC::addEdge(y^1, x);
    }

    void XOR(int x, int y) {
        implies(x, y^1); implies(x^1, y);
    }

    void atmostOne(vector<int> v) {
        int k = v.size();
        for (int i=0; i<k; i++) {
            if (i+1<k) implies(vars+2*i, vars+2*i+2);
            implies(v[i], vars+2*i);
            if (i>0) implies(v[i], vars+2*i-1);
        }
        vars += 2*k;
    }

    bool solve() {
        SCC::findSCC(vars); ans.resize(vars/2);
        for (int i=0; i<vars; i+=2) {
            if (SCC::which[i]==SCC::which[i+1])return 0;
            if (i<2*n)
                ans[i/2] = SCC::which[i]>SCC::which[i+1];
        }
        return true;
    }
};

```

6 Math

6.1 Barrett Reduction


```

struct FastMod {
    typedef unsigned long long ull;
    ull M, m;
    FastMod(ull M) : M(M), m((-1ULL / M) % M) {}
    ull reduce(ull a) { // a % M + (0 or M)
        ull ans = a - (ull)((__uint128_t(m)*a)>>64)*M;
        if (ans >= M) ans -= M;
        return ans;
    }
} F(1000000007);

```

6.2 BitwiseConvolution

```

typedef long long LL;
vector<LL> XorFWHT(vector<LL> p, bool inv) {
    int n = p.size(); assert((n&(n-1))==0);
    for (int len = 1; 2*len <= n; len <= 1) {
        for (int i = 0; i < n; i += len*len) {
            for (int j = 0; j < len; j++) {
                LL u = p[i+j], v = p[i+len+j];
                if (!inv) p[i+j]=u+v, p[i+len+j]=u-v;
                else p[i+j]=(u+v)/2, p[i+len+j]=(u-v)/2;
            }
            // if (!inv) p[i+j] = v, p[i+len+j] = u+v;
            // else p[i+j] = -u+v, p[i+len+j] = u; //AND
            // if (!inv) p[i+j] = u+v, p[i+len+j] = u;
            // else p[i+j] = v, p[i+len+j] = u-v; //OR
        }
    }
    return p;
}
vector<LL> SOS(vector<LL> p, bool inv, bool subset) {
    int k = __builtin_ctz(p.size());
    for (int i=0; i<k; i++)
        for (int mask=0; mask<(1<<k); mask++)
            if (bool(mask & (1<<i)) == subset) {
                if (!inv) p[mask] += p[mask^(1<<i)];
                else p[mask] -= p[mask^(1<<i)];
            }
    return p;
}
vector<LL> product(
    const vector<LL> &a, const vector<LL> &b) {
    vector<LL> ans(a.size());
    for (int i=0; i<a.size(); i++) ans[i]=a[i]*b[i];
    return ans;
}
vector<LL> XorConvolution(vector<vector<LL>> vs) {
    int n = vs.size();
    for (int i=0; i<n; i++) vs[i]=XorFWHT(vs[i], 0);
    vector<LL> ans = vs[0];
    for (int i=1; i<n; i++) ans=product(ans, vs[i]);
    ans = XorFWHT(ans, 1);
    return ans;
}
vector<LL> SubsetConvolution(
    const vector<LL> &a, const vector<LL> &b) {
    int k = __builtin_ctz(a.size());
    assert(a.size() == (1<<k) && b.size() == (1<<k));
    vector<LL> Z(1<<k);
    vector<vector<LL>> A(k+1,Z), B(k+1,Z), C(k+1,Z);
    for (int mask=0; mask<(1<<k); mask++) {
        A[__builtin_popcount(mask)][mask] = a[mask];
        B[__builtin_popcount(mask)][mask] = b[mask];
    }
}

```

```

}
for (int i=0; i<=k; i++) {
    A[i] = SOS(A[i], 0, 1); B[i] = SOS(B[i], 0, 1);
    for (int j=0; j<=i; j++)
        for (int mask = 0; mask < (1<<k); mask++)
            C[i][mask] += A[j][mask]*B[i-j][mask];
    C[i] = SOS(C[i], 1, 1);
}
vector<LL> ans(1<<k);
for (int mask=0; mask<(1<<k); mask++) {
    ans[mask] = C[__builtin_popcount(mask)][mask];
}
return ans;
}

```

6.3 Congruence

```

//Make mods const if possible
typedef long long LL; typedef pair<LL, LL> PLL;
LL power(LL a, LL b, LL m) {
    a = (a%m+m)%m; LL ans = 1;
    while (b) {
        if (b & 1) ans = (ans*a)%m;
        a = (a*a)%m; b >>= 1;
    }
    return ans;
}
LL egcd(LL a, LL b, LL &x, LL &y) {
    LL xx = y = 0; LL yy = x = 1;
    while (b) {
        LL q = a/b; LL t = b; b = a%b; a = t;
        t = xx; xx = x-q*xx; x = t;
        t = yy; yy = y-q*yy; y = t;
    }
    return a;
}
LL inverse(LL a, LL m) {
    LL x, y; LL g = egcd(a, m, x, y);
    if (g > 1) return -1; return (x%m+m)%m;
}
PLL CRT(LL m1, LL r1, LL m2, LL r2) {
    LL s, t; LL g = egcd(m1, m2, s, t);
    if (r1%g != r2%g) return PLL(0, -1);
    LL ss = ((s*r2)%m2)*m1, tt = ((t*r1)%m1)*m2;
    LL M = m1*m2, ans = ((ss+tt)%M+M)%M;
    return PLL(ans/g, M/g);
}
PLL CRT(const vector<LL> &m, const vector<LL> &r) {
    PLL ans = PLL(r[0], m[0]);
    for (LL i = 1; i < m.size(); i++) {
        ans = CRT(ans.second, ans.first, m[i], r[i]);
        if (ans.second == -1) break;
    }
    return ans;
}
//computes x and y such that ax + by = c
bool LinearDiophantine(LL a, LL b, LL c, LL &x, LL &y) {
    if (!a && !b) { if (c) return false;
        x = y = 0; return true;
    }
    if (!a) { if (c%b) return false;
        x = 0; y = c/b; return true;
    }
    if (!b) { if (c%a) return false;
        x = c/a; y = 0; return true;
    }
    LL g = gcd(a, b); if (c%g) return false;
    x = c/g * inverse(a/g, b/g); y = (c-a*x)/b;
}

```

```

return true;
}
LL primitive_root(LL p) {
    if (p == 2) return 1;
    LL phi = p-1, n = phi; vector<LL> factor;
    for (int i=2; i*i<=n; ++i)
        if (n%i == 0) { factor.push_back(i);
            while (n%i==0) n/=i;
        }
    if (n>1) factor.push_back(n);
    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (int i=0; i<factor.size() && ok; ++i)
            ok &= power(res, phi/factor[i], p) != 1;
        if (ok) return res;
    }
    return -1;
}
int discreteLog(int a, int b, int M) {
    a %= M, b %= M; int k = 1, add = 0, g;
    while ((g = gcd(a, M)) > 1) {
        if (b == k) return add; if (b % g) return -1;
        b /= g, M /= g, ++add; k = (1LL*k*a/g)%M;
    }
    int RT = sqrt(M)+1, aRT = 1;
    for (int i=0; i<RT; i++) aRT = (aRT*1LL*a)%M;
    unordered_map<int, int> vals;
    for (int i=0, cur=b; i<=RT; i++) {
        vals[cur] = i; cur = (cur*1LL*a)%M;
    }
    for (int i=1, cur=k; i<=M/RT+1; i++) {
        cur = (cur*1LL*aRT)%M;
        if (vals.find(cur) != vals.end())
            return RT*i-vals[cur]+add;
    }
    return -1;
}
int discreteRoot(int a, int b, int P) {
    if (b%P == 0) return a == 0 ? -1 : 0;
    int g = primitive_root(P);
    int y = discreteLog(power(g, a, P), b, P);
    return y == -1 ? -1 : power(g, y, P);
}
vector<LL> berlekampMassey(vector<LL> s, int M) {
    if (s.empty()) return {};
    int n = s.size(), L = 0, m = 0;
    vector<LL> C(n), B(n), T;
    C[0] = B[0] = 1; LL b = 1;
    for (int i = 0; i < n; ++i) {
        ++m; LL d = s[i] % M;
        for (int j = 1; j <= L; ++j)
            d = (d + C[j] * s[i-j]) % M;
        if (!d) continue;
        T = C; LL coeff = d * power(b, M-2, M) % M;
        for (int j = m; j < n; ++j)
            C[j] = (C[j] - coeff * B[j-m]) % M;
        if (2*L > i) continue;
        L = i+1-L, B = T, b = d, m = 0;
    }
    C.resize(L+1), C.erase(C.begin());
    for (LL &x : C) x = (M-x) % M;
    return C;
}

```

6.4 FFT

```
//typedef complex<double> CD;
struct CD {
double x, y;
CD(double x=0, double y=0) :x(x), y(y) {}
CD operator+(const CD& o) {return {x+o.x, y+o.y};}
CD operator-(const CD& o) {return {x-o.x, y-o.y};}
CD operator*(const CD& o) {
return {x*o.x-y*o.y, x*o.y+o.x*y};}
void operator /= (double d) { x/=d; y/=d;}
double real() {return x;}
double imag() {return y;}
};
CD conj(const CD &c) {return CD(c.x, -c.y);}
typedef long long LL; const double PI=acos(-1.0L);
namespace FFT {
int N;
vector<int> perm;
vector<CD> wp[2];

void precal(int n) {
assert((n & (n-1)) == 0); N = n;
perm = vector<int> (N, 0);
for (int k=1; k<N; k<=1) {
for (int i=0; i<k; i++) {
perm[i] <= 1; perm[i+k] = 1 + perm[i];}
}
wp[0] = wp[1] = vector<CD>(N);
for (int i=0; i<N; i++) {
wp[0][i] = CD( cos(2*PI*i/N), sin(2*PI*i/N));
wp[1][i] = CD( cos(2*PI*i/N), -sin(2*PI*i/N));
}
}

void fft(vector<CD> &v, bool invert = false) {
if (v.size() != perm.size()) precal(v.size());
for (int i=0; i<N; i++)
if (i < perm[i]) swap(v[i], v[perm[i]]);

for (int len = 2; len <= N; len *= 2) {
for (int i=0, d = N/len; i<N; i+=len) {
for (int j=0, idx=0; j<len/2; j++, idx+=d) {
CD x = v[i+j];
CD y = wp[invert][idx]*v[i+j+len/2];
v[i+j] = x+y;
v[i+j+len/2] = x-y;
}
}
}
if (invert) {
for (int i=0; i<N; i++) v[i]/=N;
}
}

void pairfft(vector<CD> &a, vector<CD> &b,
bool invert = false) {
int N = a.size(); vector<CD> p(N);
for (int i=0; i<N; i++) p[i]=a[i]+b[i]*CD(0, 1);
fft(p, invert); p.push_back(p[0]);
for (int i=0; i<N; i++) { if (invert) {
a[i] = CD(p[i].real(), 0);
b[i] = CD(p[i].imag(), 0);
} else {
a[i] = (p[i]+conj(p[N-i]))*CD(0.5, 0);
b[i] = (p[i]-conj(p[N-i]))*CD(0, -0.5);
}
}
}
```

```
}
vector<LL> multiply(vector<LL> a, vector<LL> b) {
int n = 1; while (n < a.size()+b.size()) n<=1;
vector<CD> fa(a.begin(), a.end()), fb(b.begin(),
b.end()); fa.resize(n); fb.resize(n);
pairfft(fa, fb); // fft(fa); fft(fb);
for (int i=0; i<n; i++) fa[i] = fa[i] * fb[i];
fft(fa, true);
vector<LL> ans(n);
for (int i=0; i<n; i++) ans[i]=round(fa[i].real());
return ans;
}

const int M = 1e9+7, B = sqrt(M)+1;
vector<LL> anyMod(vector<LL> a, vector<LL> b) {
int n = 1; while (n < a.size()+b.size()) n<=1;
vector<CD> al(n), ar(n), bl(n), br(n);
for (int i=0; i<a.size(); i++)
al[i] = a[i]%M/B, ar[i] = a[i]%M/B;
for (int i=0; i<b.size(); i++)
bl[i] = b[i]%M/B, br[i] = b[i]%M/B;
pairfft(al, ar); pairfft(bl, br);
// fft(al); fft(ar); fft(bl); fft(br);
for (int i=0; i<n; i++) {
CD ll = (al[i] * bl[i]), lr = (al[i] * br[i]);
CD rl = (ar[i] * bl[i]), rr = (ar[i] * br[i]);
al[i] = ll; ar[i] = lr; bl[i] = rl; br[i] = rr;
}
pairfft(al, ar, true); pairfft(bl, br, true);
// fft(al, 1); fft(ar, 1); fft(bl, 1); fft(br, 1);

vector<LL> ans(n);
for (int i=0; i<n; i++) {
LL right = round(br[i].real());
LL left = round(al[i].real());
LL mid=round(round(bl[i].real())
+round(ar[i].real()));
ans[i] = ((left%M)*B*B + (mid%M)*B + right)%M;
}
return ans;
}
```

6.5 FloorSum

```
typedef long long LL;
LL mod(LL a, LL m) {
LL ans = a%m; return ans < 0 ? ans+m : ans;
}

///Sum(floor((ax+b)/m)) for i=0 to n-1, (n,m >= 0)
LL floorSum(LL n, LL m, LL a, LL b) {
LL ra = mod(a, m), rb = mod(b, m), k = (ra*n+rb);
LL ans = ((a-ra)/m) * n*(n-1)/2 + ((b-rb)/m) * n;
if (k < m) return ans;
return ans + floorSum(k/m, ra, m, k%m);
}
```

6.6 Gaussian

```
const double EPS = 1e-9;
typedef vector<vector<double>> VVD;
int gauss(VVD ar, int e, VVD &res){
int n = ar.size(), m = ar[0].size()-1;
vector<int> pos(m, -1);
for (int j=0, i=0; j<m && i<n; j++) {
int p = i;
for (int k=i; k<n; k++){
```

```
if (abs(ar[k][j]) > abs(ar[p][j])) p = k;
}
if (abs(ar[p][j]) > EPS){
pos[j] = i; swap(ar[p], ar[i]);
for (int k=0; k<n; k++){
if (k == i) continue;
double x = ar[k][j]/ar[i][j];
for (int l=j; l<m+e; l++) ar[k][l] -= ar[i][l]*x;
} i++;
}
}

int free_var = 0;
for (int i=0; i<m; i++) free_var += (pos[i]==-1);
for (int k=0; k<e; k++) {
res.emplace_back(vector<double>(m));
for (int i=0; i<m; i++)
res.back()[i]=ar[pos[i]][m+k]/ar[pos[i]][i];
// for (int i = 0; i < n; i++) {
// double val = 0;
// for (int j=0; j<m; j++)
// val += res.back()[j]*ar[i][j];
// if (abs(val-ar[i][m])>EPS)res.back().clear();
// }
} return free_var;
}
```

6.7 NTT

```
//7340033 = 7*2^20, 645922817 = 77*2^23, G = 3
//897581057=107*2^23, 998244353=119*2^23, G = 3
namespace NTT {
vector<int> perm, wp[2]; int root, inv, N, invN;
const int mod = 998244353, G = 3; ///G prim root
int power(int a, int p) {
int ans = 1;
while (p) {
if (p & 1) ans = (1LL*ans*a)%mod;
a = (1LL*a*a)%mod; p >>= 1;
}
return ans;
}

void precalculate(int n) {
assert((n&(n-1)) == 0 && (mod-1)%n==0);
N = n; invN = power(N, mod-2);
perm = wp[0] = wp[1] = vector<int>(N);
perm[0] = 0;
for (int k=1; k<N; k<=1)
for (int i=0; i<k; i++) {
perm[i] <= 1; perm[i+k] = 1 + perm[i];
}
root=power(G, (mod-1)/N); inv=power(root, mod-2);
wp[0][0]=wp[1][0]=1;
for (int i=1; i<N; i++) {
wp[0][i] = (wp[0][i-1]*1LL*root)%mod;
wp[1][i] = (wp[1][i-1]*1LL*inv)%mod;
}
}

void fft(vector<int> &v, bool invert = false) {
if (v.size() != perm.size())precalculate(v.size());
for (int i=0; i<N; i++)
if (i < perm[i]) swap(v[i], v[perm[i]]);

for (int len = 2; len <= N; len *= 2) {
for (int i=0, d = N/len; i<N; i+=len) {
for (int j=0, idx=0; j<len/2; j++, idx+=d) {
```

```

    int x=v[i+j], y =
        (wp[invert][idx]*1LL*v[i+j+len/2])%mod;
    v[i+j] = (x+y>=mod ? x+y-mod : x+y);
    v[i+j+len/2] = (x-y>=0 ? x-y : x-y+mod);
}
}
} if (invert) {
    for (int &x : v) x = (x*1LL*invN)%mod; }
}
vector<int> multiply(vector<int> a, vector<int> b){
    int n = 1; while (n < a.size()+ b.size()) n<=1;
    a.resize(n); b.resize(n);
    fft(a); fft(b);
    for (int i=0;i<n;i++) a[i]=(a[i]*1LL*b[i])%mod;
    fft(a, true); return a;
}
};

```

6.8 Pollard-Rho

```

typedef long long LL;
typedef unsigned long long ULL;

namespace Rho {
ULL mult(ULL a, ULL b, ULL mod) {
    LL ret = a * b - mod * (ULL)(1.0L/mod*a*b);
    return ret+mod*(ret<0) - mod*(ret>=(LL) mod);
}
ULL power(ULL x, ULL p, ULL mod){
    ULL s=1, m=x;
    while(p) {
        if(p&1) s = mult(s, m, mod);
        p>>=1; m = mult(m, m, mod);
    } return s;
}
vector<LL> bases =
{2,325, 9375, 28178, 450775, 9780504, 1795265022};
bool isprime(LL n) {
    if (n<2) return 0;
    if (n%2==0) return n==2;
    ULL s = __builtin_ctzll(n-1), d = n>>s;
    for (ULL x: bases) {
        ULL p = power(x%n, d, n), t = s;
        while (p!=1&&p!=n-1&&x%n&&t--> p=mult(p,p,n);
        if (p!=n-1 && t != s) return 0;
    } return 1;
}
mt19937_64 rng(chrono::system_clock::
    now().time_since_epoch().count());
ULL FindFactor(ULL n) {
    if (n == 1 || isprime(n)) return n;
    ULL c=1, x=0, y=0, t=0, prod = 2, x0 = 1, q;
    auto f = [&](ULL X) { return mult(X, X, n) + c;};
    while (t++ % 128 or gcd(prod, n) == 1) {
        if (x == y) c = rng()%(n-1)+1, x = x0, y=f(x);
        if ((q=mult(prod, max(x, y) - min(x, y), n)))
            prod = q;
        x = f(x), y = f(f(y));
    } return gcd(prod, n);
}
vector<ULL> factorize(ULL x) {
    if (x == 1) return {};
    ULL a = FindFactor(x), b = x/a;
    if (a == x) return {a};
}

```

```

vector<ULL> L = factorize(a), R = factorize(b);
L.insert(L.end(), R.begin(), R.end());
return L;
}
}

```

6.9 PrimeCountingFunction

```

namespace PCF { //0(N^(3/4)/log). N = 10^13 -> 1.5s
const LL MAX = 1e13;
const int N = 7e6, M = 7, PM = 2*3*5*7*11*13*17;
bool isp[N];
int pr[N], pi[N]; int phi[M+1][PM+1], sz[M+1];
auto div = [](LL a,LL b)->LL{return double(a)/b;};
auto rt2 = [](LL x) -> int { return sqrtl(x); };
auto rt3 = [](LL x) -> int { return cbrtl(x); };
void init() {
    int cnt = 0; pi[0] = pi[1] = 0;
    for (int i=2; i<N; i++) isp[i] = true;
    for(int i=2; i<N; i++) {
        if(isp[i]) pr[++cnt] = i;
        pi[i] = cnt;
        for(int j=1; j<=cnt && i*pr[j]<N; j++) {
            isp[i * pr[j]] = false;
            if(i % pr[j] == 0) break;
        }
    }
    sz[0] = 1;
    for(int i = 0; i <= PM; ++i) phi[0][i] = i;
    for(int i = 1; i <= M; ++i) {
        sz[i] = pr[i] * sz[i - 1];
        for(int j = 1; j <= PM; ++j)
            phi[i][j]=phi[i-1][j]-phi[i-1][div(j,pr[i])];
    }
}
LL getphi(LL x, int s) {
    if(s == 0) return x;
    if(s <= M) return phi[s][x % sz[s]]
        + (x/sz[s]) * phi[s][sz[s]];
    if(x <= 1LL*pr[s]*pr[s]) return pi[x] - s + 1;
    if(x <= 1LL*pr[s]*pr[s]*pr[s] && x < N) {
        int s2x = pi[rt2[x]];
        LL ans = pi[x]-(s2x+s-2)*(s2x-s+1)/2;
        for(int i=s+1;i<=s2x;++i)ans+=pi[div(x,pr[i])];
        return ans;
    }
    return getphi(x, s-1)-getphi(div(x,pr[s]), s-1);
}
LL getpi(LL x) {
    if(x < N) return pi[x];
    LL ans = getphi(x, pi[rt3(x)]) + pi[rt3(x)] - 1;
    for(int i=pi[rt3(x)]+1,ed=pi[rt2(x)];i<=ed;++i)
        ans -= getpi(div(x,pr[i]))-i+1;
    return ans;
}
}

```

6.10 PrimeSignatures

```

typedef long long LL;
struct PrimeSig{
    vector<int> primes;
    PrimeSig() {
        int MX = 100;
        vector<bool> isp(MX, 1);
        for (int i=2; i<MX; i++)

```

```

        if (isp[i]) {
            primes.push_back(i);
            for (int j=2*i; j<MX; j+=i) isp[j] = 0;
        }
    }
    LL LIM;
    vector<pair<vector<int>,LL>> ans; vector<int> ps;
    void go(int idx, LL val, int mx) {
        assert(ans.size() < 100000);
        assert(idx < primes.size());
        ans.push_back({ps, val});
        int p = primes[idx]; ps.push_back(0);
        for (int i=1; i<=mx; i++) {
            if (val > LIM/p) break;
            ps.back()++; val *= p; go(idx+1, val, i);
        }
        ps.pop_back();
    }
    //{{signature, min value with signature} pair
    vector<pair<vector<int>, LL>> getAllSignature(LL
        lim) {
        LIM = lim; ans.clear(); ps.clear();
        go(0, 1, 100); return ans;
    }
};

```

6.11 powersum

```

namespace PSUM{ //for all powersum(k=1..N)
const int N = 5e3 + 2;
ll bern[N], sum[N], ifac[N];
ll bigmod(ll a, int n){
    ll ans = 1;
    while(n){
        if(n & 1)ans = ans * a % mod;
        n >>= 1; a = a * a % mod;}
    return ans;}
void init(){//call me first
    fac[0] = 1;
    for(int i=1;i<N;i++) fac[i]=fac[i-1]*i%mod;
    ifac[N - 1] = bigmod(fac[N - 1], mod - 2);
    for(int i=N-2;i>=0;i--)ifac[i]=ifac[i+1]*(i+1)%mod;
    for(int i = 0; i < N; i++){ bern[i] = 1;
        for(int j = 0; j < i; j++){
            bern[i] = (bern[i]-fac[i] * ifac[j] % mod
                *ifac[i - j + 1] % mod * bern[j]) % mod;
            if(bern[i] < 0)bern[i] += mod;
        }
    }
    //sum of i^k for 1 <= i <= n
    ll getPowerSum(ll n, int k){ll ans=0,temp=n;
        for(int i=k;i>=0;i--,temp=temp*n%mod){
            ans = (ans + bern[i] * ifac[i] % mod *
                ifac[k - i + 1] % mod * temp) % mod;
        }return (ll)ans * fac[k] % mod;
    }
    void build(ll n){init();
        for(int i=0;i<N;i++)sum[i]=getPowerSum(n, i);}
}
/**x^k=sum(i=1 to k)Stirling2(k, i)* i!* ncr(x, i)
sum (x = 0 to n) x^k
=sum(i=0 to k)Stirling2(k,i)*(n - i + 1) *
    (n - i + 2) * ... (n + 1) / (i + 1) **/
ll S[105][105];

```

```

11 solve(int n, int k) {///(Shorter)
    S[0][0] = 1 % mod;
    for (int i = 1; i <= k; i++) {
        for (int j = 1; j <= i; j++) {
            if (i == j) S[i][j] = 1 % mod;
            else S[i][j] = (j*S[i-1][j] + S[i-1][j-1])%mod;}}
    ll ans = 0;
    for (int i = 0; i <= k; i++) {
        ll fact = 1, z = i + 1;
        for (ll j = n - i + 1; j <= n + 1; j++) {
            ll mul = j;
            if (mul % z == 0) {mul /= z; z /= z;}
            fact = (fact * mul) % mod;
        }ans = (ans + S[k][i] * fact) % mod;
    }return ans;
}

```

6.12 segmentedSieve

```

/// 0(segment_size*lnln(segment_size))
/// sieve generate primes upto sqrt( max high)
void segmented_sieve(ll low, ll
    high,vector<ll> &lp_of_segment) {
    int sz = high-low+1; sieve();
    for(int i=0;i<sz;i++) lp_of_segment[i]=i+low;
    for(auto p: prime) {
        if(1LL*p*p>high) break;
        for(int i = (low+p-1)/p*p-low;i<sz;i+=p) {
            if(lp_of_segment[i]==i+low)
                lp_of_segment[i] = p;
        }
    }
}

```

6.13 xorGaussian

```

//vector<pair<int,ll> info;
//(original row, mask of ranked current rows)
int xorGaussian(vector<ll> &rows){
    int r = 0, n = (int)rows.size();
    //info.resize(n);
    //for(int i = 0; i<n; i++) info[i] = {i, 0ll};
    for(int c = 63; c>=0; c--){
        int p;
        for(p = r; p<n; p++) if(rows[p]&bt(c)) break;
        if(p == n) continue;
        if(p != r) {
            swap(rows[p], rows[r]);
            //swap(info[p], info[r]);
        }
        //info[r].second[r] = 1;
        for(int i = 0; i<n; i++){
            if(i != r && (rows[i]&bt(c))) {
                rows[i] ^= rows[r];
                //info[i].second ^= info[r].second;
            }
        }
        r++;
    } return r;
}

```

7 Misc

7.1 Fast IO C++

```

namespace FASTIO {

```

```

static const int buf_size = 4096;
/** read */
inline int getChar() {
    static char buf[buf_size];
    static int len = 0, pos = 0;
    if (pos == len) pos = 0,
        len = fread(buf, 1, buf_size, stdin);
    if (pos == len) return -1; return buf[pos++];
}
inline int readChar() {
    int c = 0; while (c <= 32) c = getChar();
    return c;
}
template <class T = long long>
inline T readInt() {
    int s = 1, c = readChar();
    T x = 0; if (c == '-') s = -1, c = getChar();
    while ('0' <= c && c <= '9')
        x = x * 10 + c - '0', c = getChar();
    return s == 1 ? x : -x;
}
/** Write */
static int write_pos = 0;
static char write_buf[buf_size];
inline void writeChar( int x ) {
    if (write_pos == buf_size) fwrite(write_buf,
        1, buf_size, stdout), write_pos = 0;
    write_buf[write_pos++] = x;
}
template <class T = long long>
inline void writeInt( T x, char end = 0 ) {
    if (x < 0) writeChar('-'), x = -x;
    char s[24]; int n = 0;
    while (x || !n) s[n++] = '0' + x % 10, x /= 10;
    while (n--) writeChar(s[n]);
    if (end) writeChar(end);
}
inline void writeWord( const char *s ) {
    while (*s) writeChar(*s++);
}
struct Flusher {
    ~Flusher() {
        if (write_pos) fwrite(write_buf, 1,
            write_pos, stdout), write_pos = 0;
    }
} flusher;

```

7.2 Ordered Set

```

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template<typename T>
using ordered_set = tree<T,null_type,less<T>,
    rb_tree_tag, tree_order_statistics_node_update>;
//find_by_order(k):iterator to kth smallest(0 ind)
//order_of_key(x) : no of items < x

```

7.3 checker

```

@echo off
if exist log.out del log.out
echo starting

```

```

for /l %%x in (1, 1, %1) do (
    TestGenerator > input.in
    solution < input.in > output.out 2>> log.out
    brute < input.in > output2.out 2>> log.out
    fc output.out output2.out > diagnostics ||
        exit /b
    echo %%x
)
echo all tests passed

```

7.4 flags

```

-std=c++17 -O2 -Wall -Wextra -pedantic
-Wshadow -Wformat=2 -Wfloat-equal -Wconversion
-Wlogical-op -Wshift-overflow=2
-Wduplicated-cond -Wcast-qual -Wcast-align

```

7.5 sabbirDebuggerHeader

```

#ifdef LOCAL
#define Gene template< class
#define Rics printer& operator,
Gene c> struct rge{c b, e;};
Gene c> rge<c> range(c i, c j){ return {i, j};}
struct printer{
    ~printer(){cerr<<endl;}
    Gene c >Rics(c x){ cerr<<boolalpha<<x; return
        *this;}
    Rics(string x){cerr<<x;return *this;}
    Gene c, class d >Rics(pair<c, d> x){ return
        *this,"(",x.first," ",x.second,"");}
    Gene ... d, Gene ...> class c >Rics(c<d...> x)
    { return *this, range(begin(x), end(x));}
    Gene c >Rics(rge<c> x){
        *this,"["; for(auto it = x.b; it != x.e; ++it)
            *this,(it==x.b?" ":""),*it; return *this,"]";}
};
#define debug() cerr<<"LINE " <<__LINE__\
    <<" >> ", printer()
#define dbg(x) "[" #x,": ",(x),"] "
#define tham getchar()
#endif
mt19937_64 rng((unsigned)chrono::system_clock\
    ::now().time_since_epoch().count());
#pragma GCC optimize("O3,unroll-loops")
#pragma GCC target("avx2,bmi,bmi2,lzcnt")

```

7.6 stresstester

```

#!/bin/bash
# stresstester GENERATOR SOL1 SOL2 ITERATIONS
for i in $(seq 1 "$4") ; do
    echo -en "\rAttempt $i/$4"
    $1 > in.txt
    $2 < in.txt > out1.txt
    $3 < in.txt > out2.txt
    diff -y out1.txt out2.txt > diff.txt
    if [ $? -ne 0 ] ; then
        echo -e "\nTestcase Found:"; cat in.txt
        echo -e "\nOutputs:"; cat diff.txt
        exit
    fi
done

```


8 String

8.1 AhoKorasick

```
const int MAXX = 1e6+7; //total length of strings
namespace AhoCorasick {
    const int sigma = 26, offset = 'a';
    struct Vertex {
        int next[sigma]; // indices of child node
        int lvl = 0; // depth of the node
        bool leaf = false; // if it is a last char
        int p = -1; // index of parent node
        char pch; // parent character
        int link = -1; // suffix link for vertex
        int go[sigma]; // where to go from here
        Vertex(int p=-1, char ch='$'): p(p), pch(ch) {
            fill(next, next+sigma, -1);
            fill(go, go+sigma, -1);
        }
    } t[MAXX];
    int exit_link[MAXX]; int used = 1;
    void new_test_case() {
        used = 1; t[0] = Vertex();
    }
    int add_string(string const& s) {
        int v = 0;
        for (char ch : s) {
            int c = ch - offset;
            if (t[v].next[c] == -1) {
                t[v].next[c] = used; t[used] = Vertex(v, ch);
                t[used].lvl = t[v].lvl + 1; used++;
            }
            v = t[v].next[c];
        }
        t[v].leaf = true; return v;
    }
    void build() {
        queue<int> q; q.push(0);
        while (!q.empty()) {
            int v = q.front(); q.pop();
            { //calculate suffix link
                if (v == 0 || t[v].p == 0) t[v].link = 0;
                else t[v].link =
                    t[t[v].p].link.go[t[v].pch - offset];
            }
            { //calculate exit link
                if (v == 0 || t[v].p == 0) exit_link[v] = 0;
                else if (t[t[v].link].leaf)
                    exit_link[v] = t[v].link;
                else exit_link[v] = exit_link[t[v].link];
            }
        }
        for (int i = 0; i < sigma; i++) {
            if (t[v].next[i] != -1) q.push(t[v].next[i]);
        }
        // save go values in next to save memory!
        for (int c = 0; c < sigma; c++) {
            if (t[v].next[c] != -1)
                t[v].go[c] = t[v].next[c];
            else
                t[v].go[c] = v == 0 ? 0 : t[t[v].link].go[c];
        }
    }
}
```

```
}
int main() {
    int n, k; cin >> n >> k;
    AhoCorasick::new_test_case();
    for (int i = 1; i <= k; i++) {
        string t; cin >> t; AhoCorasick::add_string(t);
    }
    AhoCorasick::build();
}
```

8.2 KMP

```
const int ALPHA = 26;
// builds the prefix automaton in O(N*ALPHA)
vector<vector<int>> automaton;
void buildAutomaton(const string& s) {
    automaton.clear(); int n = s.size(), k = 0;
    for (int i = 0; i <= n; i++)
        automaton.emplace_back(ALPHA, 0);
    automaton[0][s[0] - 'a'] = 1;
    for (int i = 1; i <= n; i++) {
        automaton[i] = automaton[k];
        if (i < n) {
            automaton[i][s[i] - 'a'] = i + 1;
            k = automaton[k][s[i] - 'a'];
        }
    }
}
vector<int> prefixFunction(const string& s) {
    int n = s.size(), k = 0; // 1-indexed
    vector<int> v(n + 1); v[1] = 0;
    for (int i = 2; i <= n; i++) {
        while (k > 0 && s[k] != s[i - 1]) k = v[k];
        if (s[k] == s[i - 1]) k++;
        v[i] = k;
    }
    return v;
}
int matcher(const string& txt, const string& ptrn) {
    vector<int> pi = prefixFunction(ptrn);
    int matchCount = 0, k = 0;
    for (int i = 0; i < txt.size(); i++) {
        while (k > 0 && txt[i] != ptrn[k]) k = pi[k];
        if (txt[i] == ptrn[k]) k++;
        if (k == ptrn.size()) {
            matchCount++; k = pi[k];
        }
    }
    return matchCount;
}
```

8.3 PalindromicTree

```
const int MAXN = 1e5+7; //max length of string+3
namespace PalTree {
    struct node {
        int len; //length of the pal of this node
        int sufflink; //largest suff pal of this node
        int chain; //#of nodes on chain of suff links
        int next[26]; //next[c] is the pal by adding c
    } tr[MAXN];
    int size; // # of nodes currently in Pal tr
    int suff; //max suff pal of curr processed prefix
    string s; //string we will built our Paltr on
```

```
bool addLetter(int pos) {
    int cur = suff, curlen = 0, let = s[pos] - 'a';
    while (true) {
        curlen = tr[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos])
            break; cur = tr[cur].sufflink;
    }
    if (tr[cur].next[let]) {
        suff = tr[cur].next[let]; return false;
    }
    suff = ++size; tr[cur].next[let] = size;
    tr[size].len = tr[cur].len + 2;
    if (tr[size].len == 1) {
        tr[size].sufflink = 2; tr[size].chain = 1;
        return true;
    }
    while (true) {
        cur = tr[cur].sufflink; curlen = tr[cur].len;
        if (pos - 1 - curlen >= 0 && s[pos - 1 - curlen] == s[pos]) {
            tr[size].sufflink = tr[cur].next[let];
            break;
        }
    }
    tr[size].chain = 1 + tr[tr[size].sufflink].chain;
    return true;
}
void initTree() {
    memset(tr, 0, sizeof tr); //CAREFUL: TESTCASES
    size = 2; suff = 2;
    tr[1].len = -1; tr[1].sufflink = 1;
    tr[2].len = 0; tr[2].sufflink = 1;
}
int main() {
    int q; cin >> q;
    string operations; cin >> operations;
    PalTree::initTree();
    vector<int> subs, suffs; subs.push_back(0);
    suffs.push_back(PalTree::suff);
    for (char c : operations) {
        if (c == '-') {
            subs.pop_back(); suffs.pop_back();
            PalTree::suff = suffs.back();
        }
        else {
            PalTree::suff += c;
            PalTree::addLetter(PalTree::suff.size() - 1);
            suffs.push_back(PalTree::suff);
            subs.push_back(subs.back() +
                PalTree::tr[PalTree::suff].chain);
        }
    }
    cout << subs.back() << " ";
    return 0;
}
```

8.4 SuffixArrayTree

```
typedef pair<int, int> PII;
ostream &operator<<(ostream &out, const PII &p) {
    return out<<"("<<p.first<<","<<p.second<<")";
}
const int maxn = 1e5+5; //NOTICE
namespace DA {
```

```

int wa[maxn],wb[maxn],wv[maxn],wc[maxn],r[maxn];
int sa[maxn],rak[maxn],height[maxn],SIGMA=0;
int cmp(int *r,int a,int b,int l) {
    return r[a] == r[b] && r[a+l] == r[b+l];
}
void da(int *r,int *sa,int n,int m) {
    int i,j,p,*x=wa,*y=wb,*t;
    for(i=0;i<m;i++) wc[i]=0;
    for(i=0;i<n;i++) wc[x[i]=r[i]]++;
    for(i=1;i<m;i++) wc[i] += wc[i-1];
    for(i=n-1;i>=0;i--) sa[--wc[x[i]]] = i;
    for(j=1,p=1;p<n;j*=2,m=p){
        for(p=0,i=n-j;i<n;i++)y[p++] = i;
        for(i=0;i<n;i++)if(sa[i]>=j) y[p++] = sa[i]-j;
        for(i=0;i<n;i++)wv[i] = x[y[i]];
        for(i=0;i<m;i++) wc[i] = 0;
        for(i=0;i<n;i++) wc[wv[i]]++;
        for(i=1;i<m;i++) wc[i] += wc[i-1];
        for(i=n-1;i>=0;i--) sa[--wc[wv[i]]] = y[i];
        for(t=x,x=y,y=t,p=1,x[sa[0]]=0,i=1;i<n;i++){
            x[sa[i]] = cmp(y,sa[i-1],sa[i],j) ? p-1:p++;
        }
    }
}
void calheight(int *r,int *sa,int n) {
    int i,j,k=0;
    for(i=1;i<=n;i++) rak[sa[i]] = i;
    for(i=0;i<n;height[rak[i++]] = k) {
        for(k?k--:0,j=sa[rak[i]-1];
            r[i+k]==r[j+k];k++);
    }
}
//LCP[i] = LCP of (SA[i], SA[i + 1])
void suffixArray(const string &s,
    vector<int> &suffArray, vector<int> &lcp){
    int n = s.size(); SIGMA = 0;
    for(int i = 0; i < n; i++) {
        if ('a'<=s[i]&&s[i]<='z') r[i] = s[i]-'a'+2;
        //CHANGE THIS
        else r[i] = 1; //separators
        SIGMA = max(SIGMA, r[i]);
    }
    r[n] = 0; da(r, sa, n+1, SIGMA + 1);
    suffArray.resize(n);
    for(int i = 0; i<n; i++) suffArray[i]=sa[i+1];
    calheight(r,sa,n); lcp.resize(n-1);
    for (int i = 0; i+1<n; i++)lcp[i]=height[i+2];
}
}
typedef vector<int>VI;const int K=20;int lg[maxn];
void pre() { //CALL ME PLS
    lg[1] = 0;
    for (int i=2; i<maxn; i++) lg[i] = lg[i/2]+1;
}
struct RMQ{
    int N; VI v[K];
    RMQ(const VI &a) {
        N = a.size(); v[0] = a;
        for (int k = 0; (1<<(k+1)) <= N; k++) {
            v[k+1].resize(N);
            for (int i = 0; i-1+(1<<(k+1)) < N; i++) {
                v[k+1][i] = min(v[k][i], v[k][i+(1<<k)]);
            }
        }
    }
}

```

```

}
}
int findMin(int i, int j) const {
    assert(i <= j); int k = lg[j-i+1];
    return min(v[k][i], v[k][j+1-(1<<k)]);
}
};
PII extend(RMQ &rmq, int saSize, int ps, int len){
    int L = ps, R = ps;
    for (int k = K-1; k >= 0; k--) {
        int r = R+(1<<k); if (r >= saSize) continue;
        if (rmq.findMin(ps, r-1) >= len) R = r;
    }
    for (int k = K-1; k >= 0; k--) {
        int l = L-(1<<k); if (l < 0) continue;
        if (rmq.findMin(l, ps-1) >= len) L = l;
    }
    return PII(L, R);
}
//len retrnd by backstep() must b min-ed from out
struct BackStepper {
    vector<int> startsWith[26];
    BackStepper(const string &s,const vector<int> &sa){
        for (int i = 0; i < sa.size(); i++) {
            if (sa[i] > 0)
                startsWith[s[sa[i]-1]-'a'].push_back(i);
        }
        startsWith[s.back()-'a'].push_back(s.size());
    }
    /**Return <len, j> s.t. s[j] = c and suffix[j+1]
    shares the longest prefix with suffix[i]
    Returns <0, 0> if no such index exists. */
    PII backstep(int i,int c,const vector<int> &sa,
        const vector<int> &ra, const RMQ &rmq){
        if (startsWith[c].empty()) return PII(0, 0);
        int ri = ra[i];
        int idx = lower_bound(startsWith[c].begin(),
            startsWith[c].end(),ri)-startsWith[c].begin();
        if (idx < startsWith[c].size() &&
            startsWith[c][idx]==ri) { //same pos again
            return PII(ra.size()-i+1, i-1);
        }
    }
    PII rt(-1, -1);
    if (idx > 0) {
        int ci = startsWith[c][idx-1];
        rt = PII(rmq.findMin(ci, ri-1)+1, sa[ci]-1);
    }
    if (idx < startsWith[c].size()) {
        int ci = startsWith[c][idx];
        if (ci==sa.size())
            rt=max(rt,PII(1, sa.size()-1));
        else rt = max(rt,
            PII(rmq.findMin(ri, ci-1)+1, sa[ci]-1));
    }
    return rt;
}
};
//n -> string length, SZ -> number of nodes in
suff tree, 1 is root (the empty string), length of
node u (starting from root) is length[u], for an
edge between node u and it's child v, edge length
is length[v]-length[u], node u belongs to all of

```

```

[L[u], R[u]] suffixes in Suffix Array */
const int MXND = maxn*2+7;
struct SuffixTree {
    int nxt[maxn]; //nxt[i] = position of next #
    vector<int>edg[MXND], leaves[MXND];
    int lnth[MXND],L[MXND],R[MXND], SZ, n;
    void buildGraph(const vector<int> &sa,
        vector<int> &lcp) {
        //edg and leaves must be cleared if test cases
        SZ = 0; vector<int> stk{++SZ};
        n = sa.size(), L[SZ] = 0, R[SZ] = n-1;
        lnth[SZ] = 0; lcp.push_back(0); int last = -1;
        for(int i = 0, sf = 1; i+sf<n; i+=sf, sf^=1){
            int left=i-(sf^1); //sf=suflen/lcp being used
            //int curlcp=(sf)?n-sa[i]:(i?lcp[i-1]:0);
            //single string;
            int curlcp = (sf)?nxt[sa[i]]-sa[i]:
                (i?lcp[i-1]:0); //multiple string
            while(curlcp < lnth[stk.back()]){
                R[stk.back()]=i-(sf^1),left=L[stk.back()];
                last = stk.back(), stk.pop_back();
                if(curlcp <= lnth[stk.back()])
                    edg[stk.back()].push_back(last),last=-1;
            }
            if(curlcp > lnth[stk.back()]){
                stk.push_back(++SZ);
                if(last!=-1)edg[SZ].push_back(last),last=-1;
                lnth[SZ]=curlcp,L[SZ]=left;
            }
        }
    }
    void buildLeaves(const vector<int> &sa){
        for(int i = 1; i<=SZ; i++){
            int r=(edg[i].empty()?R[i]:L[edg[i][0]]-1);
            for(int j = L[i]; j<=r; j++){
                leaves[i].push_back(sa[j]);
            }
        }
    }
} st;
void dfs(const string &s,vector<int> &sa,int u){
    for (int v : st.edg[u]) {
        cout << u << " - ";
        cout << s.substr(sa[st.L[v]]+st.lnth[u],
            st.lnth[v]-st.lnth[u]);
        cout << ":- " << v; cout << " {";
        for (int x : st.leaves[v]) cout << " " << x;
        cout << " }"; cout << endl; dfs(s, sa, v);
    } //suffLen[i] denotes the actual
} //suffix length of sa[i]
int suffLen[maxn];
void sanitize(const string &s,
    const vector<int> &sa, vector<int> &lcp) {
    int n = s.size(); st.nxt[n] = n;
    for (int i = n-1; i >= 0; i--) {
        st.nxt[i]=st.nxt[i+1];if(s[i]=='#')st.nxt[i]=i;
    }
    for (int i = 0; i < n; i++) {
        suffLen[i] = st.nxt[ sa[i] ] - sa[i];
    }
    for (int i = 0; i+1 < n; i++) {
        lcp[i]=min(lcp[i],min(suffLen[i],suffLen[i+1]));
    }
}

```

```

}
}
int main() {
    pre(); // MUST for RMQ to be working
    string s = "abac#ababa";
    vector<int> sa, lcp; DA::suffixArray(s, sa, lcp);
    sanitize(s, sa, lcp); cout << "suffix order:" << endl;
    for(int i:sa) cout << s.substr(i) << endl;
    cout << "lcp:"; for(int i:lcp) cout << " " << i;
    cout << endl; st.buildGraph(sa, lcp);
    st.buildLeaves(sa); dfs(s, sa, 1); return 0;
}

```

8.5 SuffixAutomata

```

namespace Automata {
    const int N = 1e6+7, K = 26;
    int len[2*N], link[2*N], nxt[2*N][K], sz, last;
    void init(int n) {
        for (int i=0; i<=2*n; i++)
            fill(nxt[i], nxt[i]+K, -1);
        len[0] = 0; link[0] = -1; sz = 1; last = 0;
    }
    void add(char ch) {
        int c = ch-'a', cur = sz++; // create new node
        len[cur] = len[last]+1;
        int u = last;
        while (u != -1 && nxt[u][c] == -1) {
            nxt[u][c] = cur; u = link[u];
        }
        if (u == -1) { link[cur] = 0; }
        else {
            int v = nxt[u][c];
            if (len[v] == len[u]+1) { link[cur] = v; }
            else {
                int clone = sz++; // create node by cloning
                len[clone] = 1+len[u]; link[clone] = link[v];
                for (int i=0; i<K; i++)
                    nxt[clone][i] = nxt[v][i];
                while (u != -1 && nxt[u][c] == v) {
                    nxt[u][c] = clone; u = link[u];
                }
                link[v] = link[cur] = clone;
            }
        }
        last = cur;
    }
    vector<int> edge[2*N];
    void makeEdge() { // Make Suffix Link Tree edges
        for (int i=0; i<sz; i++) {
            edge[i].clear();
            for (int j=0; j<K; j++)
                if (nxt[i][j] != -1) edge[i].push_back(j);
        }
    }
}

```

8.6 Zalgo

```

vector<int> z_function(string s) {
    int n = s.size(); vector<int> z(n);
    for (int i=1, l=0, r=0; i<n; ++i) {

```

```

        if (i <= r) z[i] = min(r-i+1, z[i-l]);
        while(i+z[i]<n && s[z[i]]==s[i+z[i]])
            ++z[i];
        if(i+z[i]-1 > r) l=i, r = i+z[i]-1;
    }
    return z;
}

```

8.7 hashing

```

/**Hash(ab..z) = a*p^n+b*p^(n-1)+... + z
In order to convert to Single Hash -
    o Delete operator overloads and fix reduce()
    o Replace all PLL with LL
    o Change mp pairs to appropriate value
Or set M2 = 1, which should be nearly as fast.
Some Primes:
1000000007, 1000000009, 1000000861, 1000099999 (<2^30)
1088888881, 1111211111, 1500000001, 1481481481 (<2^31)
*/
typedef long long LL; typedef pair<LL, LL> PLL;
namespace Hashing {
    #define ff first
    #define ss second
    const PLL M = {1e9+7, 1e9+9};
    const LL base = 1259; const int N = 1e6+7;
    PLL operator+ (const PLL& a, LL x)
        {return {a.ff + x, a.ss + x};}
    PLL operator- (const PLL& a, LL x)
        {return {a.ff - x, a.ss - x};}
    PLL operator* (const PLL& a, LL x)
        {return {a.ff * x, a.ss * x};}
    PLL operator+ (const PLL& a, PLL x)
        {return {a.ff + x.ff, a.ss + x.ss};}
    PLL operator- (const PLL& a, PLL x)
        {return {a.ff - x.ff, a.ss - x.ss};}
    PLL operator* (const PLL& a, PLL x)
        {return {a.ff * x.ff, a.ss * x.ss};}
    PLL operator% (const PLL& a, PLL m)
        {return {a.ff % m.ff, a.ss % m.ss};}
    PLL pb[N]; // powers of base mod M
    void hashPre() { // Call pre before everything
        pb[0] = {1,1};
        for(int i=1; i<N; i++) pb[i] = (pb[i-1]*base)%M;
    }
    // Calculates hashes of all prefixes
    vector<PLL> hashList(string s) {
        int n = s.size();
        vector<PLL> ans(n+1); ans[0] = {0,0};
        for (int i=1; i<=n; i++)
            ans[i] = (ans[i-1] * base + s[i-1])%M;
        return ans;
    }
    // Calculates hash of s[l..r] (1 indexed)
    PLL subHash(const vector<PLL> &h, int l, int r)
    { return (h[r]+(M-h[l-1])*pb[r-l+1])%M; }
    // Calculates Hash of a string
    PLL Hash(const string &s) { PLL ans = {0,0};
        for (int i=0; i<s.size(); i++)
            ans = (ans*base + s[i])%M;
        return ans;
    }
    PLL append(PLL cur, char c)

```

```

{ return (cur*base + c)%M; }
PLL prepend(PLL cur, int k, char c)
{ return (pb[k]*c + cur)%M; }
PLL replace(PLL cur, int i, char a, char b)
{ return cur + pb[i] * (M+b-a)%M; }
PLL pop_front(PLL hash, int len, char c)
{ return (hash + pb[len-1]*(M-c))%M; }
// concatss two strings. k=length of the right
PLL concat(PLL left, PLL right, int k)
{ return (left*pb[k] + right)%M; }
PLL power (PLL a, LL p) {
    PLL ans = {1, 1};
    while(p > 0) {
        if (p&1) ans = (ans*a)%M;
        p >= 1; a = (a * a) % M;
    }
    return ans;
}
PLL inv(PLL a) {
    if (M.ss == 1) return power(a, M.ff-2);
    return power(a, (M.ff-1)*(M.ss-1)-1);
}
PLL invb = inv({base, base});
PLL pop_back(PLL hash, char c)
{ return ((hash-c*M)*invb)%M; }
// hash of str with size len repeated cnt times
// O(log n). For O(1), pre-calculate inverses
PLL repeat(PLL hash, int len, LL cnt) {
    PLL mul = ((pb[len*cnt]-1*M)*inv(pb[len]-1*M))%M;
    PLL ans = (hash*mul);
    if (pb[len].ff == 1) ans.ff = hash.ff*cnt;
    if (pb[len].ss == 1) ans.ss = hash.ss*cnt;
    return ans%M;
}
}

```

8.8 manacher

```

// 0-based indexing
// p[0][i] = length of half palin around hlf ind i
// p[1][i] = length of half palin around char i
struct Manacher {
    vector<int> p[2];
    Manacher(const string &s) {
        int n = s.size();
        p[0].resize(n+1); p[1].resize(n);
        for (int z=0; z<2; z++)
            for (int i=0, l=0, r=0; i<n; i++) {
                int t = r-i+l;
                if (i<r) p[z][i] = min(t, p[z][l+t]);
                int L = i-p[z][i], R = i+p[z][i]-l;
                while (L>=1 && R+1<n && s[L-1] == s[R+1])
                    p[z][i]++, L--, R++;
                if (R>r) l=L, r=R;
            }
    }
    bool ispalin(int l, int r) {
        int mid = (l+r+1)/2, sz = r-l+1;
        return 2*p[sz/2][mid]+sz/2 >= sz;
    }
};

```

1 Geometry

1.1 Triangles

- $c^2 = a^2 + b^2 - 2ab \cos C$
- $\frac{a}{\sin A} = \frac{b}{\sin B} = \frac{c}{\sin C} = 2R$
- $\Delta = \frac{1}{2}ah_a = \frac{1}{2}ab \sin C = \frac{abc}{4R} = sr$
- $\Delta = \sqrt{(s(s-a)(s-b)(s-c))}$
- Median, $m_a = \frac{1}{2}\sqrt{2b^2 + 2c^2 - a^2}$
- Angle Bisector, $b_a = \sqrt{bc \left(1 - \left(\frac{a}{b+c}\right)^2\right)}$

1.2 Baricentric Coordinates

- Centroid, $G = [1, 1, 1]$
- Incenter, $I = [a, b, c]$
- Excenter, $I_a = [-a, b, c]$
- Circumcenter, $O = [a^2(b^2 + c^2 - a^2)]$
- Orthocenter, $= [(c^2 + a^2 - b^2)(a^2 + b^2 - c^2)]$

1.3 Polygons

- Area, $A = \frac{1}{2} \sum_{i=0}^{n-1} (x_i y_{i+1} - x_{i+1} y_i)$.
- $Cen_x = \frac{1}{6A} \sum_{i=0}^{n-1} (x_i + x_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$
- $Cen_y = \frac{1}{6A} \sum_{i=0}^{n-1} (y_i + y_{i+1})(x_i y_{i+1} - x_{i+1} y_i)$,

1.4 Miscellaneous

- **Pick's Theorem:** $A = i + \frac{b}{2} - 1$
- **Euler:** $V - E + F = C + 1$
- A connected planar graph with n vertices has at most $3n - 6$ faces and $2n - 4$ edges.

2 Combinatorics

2.1 Sequences

1. **Binomial Coefficients** $\binom{n}{k} = k$ element subsets of a n element set.

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

2. **Derangements** Perms with no fixed points

- $D(n) = (n-1)(D(n-1) + D(n-2))$

- $D(n) = nD(n-1) + (-1)^n = \left[\frac{n!}{e}\right]$
- $D(n) = n! \left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots\right) = \left[\frac{n!}{e}\right]$

3. **Stirling numbers of the first kind** $s(n, k) =$ Permutations on n items with k cycles.

- $s(n, k) = s(n-1, k-1) + (n-1)s(n-1, k)$
- $\sum_{k=0}^n s(n, k)x^k = x(x+1) \cdots (x+n-1)$

4. **Stirling numbers of the second kind** $S(n, k) =$ Partitions of n distinct elements into k groups.

- $S(n, k) = S(n-1, k-1) + kS(n-1, k)$
- $S(n, k) = \frac{1}{k!} \sum_{j=0}^k (-1)^{k-j} \binom{k}{j} j^n$

5. **Eulerian numbers** $E(n, k) =$ Permutations with exactly k indices i , st, $a_i > a_{i-1}$

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

6. **Catalan numbers** $C_n = 1, 1, 2, 5, 14, 42, 132, \dots$

- Balanced bracket sequences with length $2n$.
- n -permutations with $\text{LIS} \leq 2$
- $C_n = \frac{1}{n+1} \binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n-1}$
- $C_{n+1} = \sum_{k=0}^n C_k C_{n-k}$
- Closing bracket count ahead by at most k : reflect path wrt $y = x + k + 1$

7. **Partition function** Ways of writing n as a sum of positive integers, disregarding order, $p(n) = 1, 1, 2, 3, 5, 7, 11, 15, 22, 30 \dots$

- $p(n) = \sum_{k \in \mathbb{Z} \setminus \{0\}} (-1)^{k+1} p(n - k(3k \pm 1)/2)$

2.2 Principle of Inclusion Exclusion

Let A be a set, c_1, c_2, \dots, c_n be n conditions, and A_k be the set of elements satisfying c_k . Let E_m be the set satisfying exactly m conditions, and L_m be the set satisfying at least m conditions. Let

$S_k = \sum_{|I|=k} |\bigcap_{i \in I} A_i|$. Then,

- $|E_m| = \sum_{k=m}^n (-1)^{k-m} \binom{k}{m} S_k$
- $|L_m| = \sum_{k=m}^n (-1)^{k-m} \binom{k-1}{m-1} S_k$

2.3 Permutation Cycles

- Exp. number of cycles in an n permutation is $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$
- Let $g_S(n)$ be the number of n -permutations whose cycle lengths all belong to the set S .
$$\sum_{n=0}^{\infty} g_S(n) \frac{x^n}{n!} = \exp \left(\sum_{n \in S} \frac{x^n}{n} \right)$$

2.4 Burnside's lemma

• Given a group G of symmetries and a set X , the number of elements of X up to symmetry equals $\frac{1}{|G|} \sum_{g \in G} |X^g|$, where X^g are elements fixed by g .

• If $f(n)$ counts “configurations” of length n , we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get $g(n) = \frac{1}{n} \sum_{k=0}^{n-1} f(\gcd(n, k)) = \frac{1}{n} \sum_{k|n} f(k) \phi(n/k)$.

2.5 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = (n_k \cdots n_1 n_0)_p$ and $m = (m_k \cdots m_1 m_0)_p$ in base p . Then $\binom{n}{m} \equiv \prod_{i=0}^k \binom{n_i}{m_i} \pmod{p}$.

3 Graphs

3.1 Cayley's Theorem

- Labeled trees on n vertices: n^{n-2}
- Connect k trees of size n_i : $n_1 n_2 \cdots n_k n^{k-2}$
- Trees with degrees d_i : $\frac{(n-2)!}{(d_1-1)! \cdots (d_n-1)!}$

3.2 Chromatic Polynomials

Let G be a graph with n vertices. Let $P(G, x)$ be the number of vertex colorings of G with x colors. Then P is a monic integer polynomial of degree n .

- $P(G, x) = P(G - uv, x) - P(G/uv, x)$ if $uv \in E$
- $P(G, x) = P(G + uv, x) + P(G/uv, x)$ if $uv \notin E$
- $P(G, x) = x(x-1)^{n-1}$ if G is a tree
- $P(G, x) = (x-1)^n + (-1)^n(x-1)$ if G is a cycle

3.3 Number of spanning trees

Let D = degree matr, A = adj. matr. Delete one row, col of $D - A$ to get L . Then no of spanning trees, $t(G) = t(G - uv) + t(G/uv) = \det(L)$

3.4 Erdos Gallai Theorem

A simple graph with degree seq., $d_1 \geq \dots \geq d_n$ exists iff $d_1 + \dots + d_n$ is even and for all k , $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$

3.5 LGV lemma

DAG with sources a , sinks b and $|a| = |b| = n$. For (i,j,k,l) "if $i < j$ and $k < l$, then paths a_i to b_l and a_j to b_k share a vertex" \Rightarrow number of n tuple node disjoint paths a_i to b_i is Determinant of X . where $X_{i,j}$ = number of paths of a_i to b_j

4 Flow and Matching

4.1 Marriage Theorem and Generalizations

- **Hall's Marriage Theorem** In a bipartite graph $G = A \cup B$, a matching saturating A exists iff $|N(S)| \geq |S|$ for all $S \subset A$
- **Generalization (Unproven)** In a bipartite graph $G = A \cup B$, a matching with size $|A| - x$ exists iff $|N(S)| \geq |S| - x$ for all $S \subset A$
- **Generalization (Sabbir)** In a bipartite (U, V) flow graph with $C()$, then maxflow is equal to $\sum_{u \in U} C(u)$ iff for every set $S \subseteq U$, $\sum_{u \in S} C(u) \leq F(S)$ where $F(S) = \sum_{v \in V} \min(C(v), \sum_{u \in S} C(u, v))$

4.2 Bipartite Matching

Let M be a max matching of a graph with bipartitions L and R . Let U be unmatched vertices in L , Z be vertices reachable from U via alternating paths.

- **Konig's Theorem** Min Vertex Cover = Max Matching. $(L \setminus Z) \cup (R \cap Z)$ is such a cover. In fact, $L \setminus Z$ are the only nodes which is in at least

one vertex cover. A vertex is part of all cover if its partner in M is part of no cover.

- **Max independent set** is the compliment of Min Vertex Cover. Max BiClique is Max independent set in complement graph.

4.3 Feasible Flow

- **Feasible Flow**: flow of all old edge = upper cap - lower cap, $acc[u]$ = sum of flow into u - sum of flow out from u . Add **supersource**, **supersink**, edge from supersource to u (if $acc[u] > 0$), edge from u to supersink (if $acc[u] < 0$), capacity $|acc(u)|$ and an edge sink to source (capacity: ∞). if maxflow from supersource to supersink = sum of outgoing cap from supersource. Then feasible flow exists.
- **Maxflow** remove all new edges (but not flows/caps in old edges) and apply maxflow from source to sink. this is the maxflow (with lower caps hidden), so max flow = $\sum flow(e) + low(e)$ where e is adjacent to source, (incomings are negative)

4.4 Miscellaneous

- **Dilworth's Theorem** In a poset, the size of a maximal antichain equals the size of a minimal chain cover. Thus minimum path cover in a DAG equals maximum flow in the transitive closure.
- **Tutte's Theorem** Let $o(S)$ be the number of odd components in S . A graph G has a perfect matching iff $\forall S \subset V, o(G \setminus S) \leq |S|$

5 Math

- $\det(M + uv^T) = \det(M) + u^T \text{adj}(M)v$, u, v are column matrices
- Given $x_i - x_j \leq C_{i,j}$ inequalities, construct graph with node for each variable and a ground node. put edge $j \rightarrow i$ with cost $c_{i,j}$ for the above ineq. also put $ground \rightarrow i$ for each i (with cost

generally 0). then bellmand ford from ground will give value for all x_i . If we use w_i as cost for $ground \rightarrow i$, then that serves as $x_i \leq w_i$.

- **Lagrange Interpolation**

$P(x) = \sum y_i P_i(x)$ where $P_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$

- **Newton Interpolation** $P(x) = \sum a_i P_i(x)$ where $a_i = [y_0, \dots, y_i]$ and $P_i(x) = \prod_{j < i} (x - x_j)$

$[y_a, \cdot, y_b] = ([y_{a+1}, \cdot, y_b] - [y_a, \cdot, y_{b-1}]) / (x_b - x_a)$

- **simpson** $\int_a^b f(x) dx \approx \frac{b-a}{3n} (f(x_0) + 4f(x_1) + 2f(x_2) + \dots + f(x_n))$

$x_{i+1} - x_i = (b - a)/n$

- **Polynomial inverse** $B_0 = 1/A[0]$, $B_{k+1} = B_k(2 - AB_k)$

- **Number of divisor of n digit number** 4, 12, 32, 64, 128, 240, 448, 768, 1344, 2304, 4032, 6720, 10752, 17280, 26880, 41472, 64512, 103680

6 SegTree beats

for min-update, keep max and second max (distinct) in each node. during update with x , in the case when (l, r) is inside update range (L, R) , if $x \geq \text{max}$ do nothing, if $\text{max} > x \geq \text{max2}$ set lazy and return, if $\text{max2} > x$ recurse deeper. (count of max needed for sum).

7 Games

- **Green Hacken** Merge nodes of biconnected component. Component edges are converted to leaf from merged node. Then $G[u] \oplus = (1 + G[v])$