

1 HeavyDS

1.1 Iterative

```
struct Data{};
Data op(const Data &a, const Data &b){}
struct segtree{
    vector<Data> d;
    int _n, log, size;
    unsigned int bit_ceil(unsigned int n)
        ↪ {
        unsigned int x = 1;
        while (x < (unsigned int)(n)) x
            ↪ <= 1;
        return x;
    }
    segtree(int n): _n(n) {
        size = (int)bit_ceil((unsigned
            ↪ int)(n));
        log = __builtin_ctz((unsigned
            ↪ int)size);
        d.resize(2*size, Data());
    }
    void update(int k) { d[k] = op(d[2*k],
        ↪ d[2*k+1]); }
    void set(int p, Data x) {
        p += size;
        d[p] = x;
        for (int i = 1; i <= log; i++)
            ↪ update(p >> i);
    }
    Data prod(int l, int r) { // !!! q(l,r)
        ↪ return l to r-1
        if (l == r) return Data();
        l += size; r += size;
        Data sml = Data(), smr = Data();
        while (l < r) {
            if (l&1) sml = op(sml, d[l++]);
            if (r&1) smr = op(d[--r], smr);
```

```
                l >>= 1; r >>= 1;
            }
            return op(sml, smr);
        }
    };
```

1.2 Sgtree

```
struct SegTree {
    int n;
    vector<int> tree;
    SegTree(int n) : n(n), tree(4 * n) {}
    void build(int node, int l, int r,
        ↪ const vector<int>& a) {
        if (l == r) {
            tree[node] = a[l];
            return;
        }
        int mid = (l + r) >> 1;
        build(node << 1, l, mid, a);
        build(node << 1 | 1, mid + 1, r,
            ↪ a);
        tree[node] = tree[node << 1] +
            ↪ tree[node << 1 | 1];
    }
    void update(int node, int l, int r,
        ↪ int idx, int val) {
        if (l == r) {
            tree[node] = val;
            return;
        }
        int mid = (l + r) >> 1;
        if (idx <= mid)
            update(node << 1, l, mid, idx,
                ↪ val);
        else
            update(node << 1 | 1, mid + 1,
                ↪ r, idx, val);
```

```
        tree[node] = tree[node << 1] +
            ↪ tree[node << 1 | 1];
    }
    int query(int node, int l, int r, int
        ↪ ql, int qr) {
        if (qr < l || r < ql)
            return 0;
        if (ql <= l && r <= qr)
            return tree[node];
        int mid = (l + r) >> 1;
        return query(node << 1, l, mid, ql,
            ↪ qr)
            + query(node << 1 | 1, mid +
            ↪ 1, r, ql, qr);
    }
};
```

1.3 Lazy

```
struct Min{
    int x;
    Min(int x = inf) : x(x) {}
};
Min operator+(const Min& a, const Min& b){
    return std::min(a.x, b.x);
}
void apply(int &a, int b) a += b;
void apply(Min& a, int b) a.x += b;
// Always make a class for them and define
    ↪ basic overloads
template<class Node, class Lazy, class
    ↪ Merge = std::plus<Node>>
struct LazySegTree {
    const int _n;
    const Merge _merge;
    std::vector<Node> _segT;
    std::vector<Lazy> _lazy;
```

```

LazySegTree(int n) : _n(n),
    ↪ _merge(Merge()), _segT(4 <<
    ↪ std::__lg(n)), _lazy(4 <<
    ↪ std::__lg(n)) {}
LazySegTree(std::vector<Node> init) :
    ↪ LazySegTree(init.size()) {
    std::function<void(int, int, int)>
    ↪ build = [&](int node, int st,
    ↪ int en){
        if(en - st == 1){
            _segT[node] = init[st];
            return;
        }
        int md = (st + en) >> 1;
        build(node << 1, st, md);
        build(node << 1 | 1, md, en);
        pull(node);
    };
    build(1, 0, _n);
}
void pull(int node){
    _segT[node] = _merge(_segT[node <<
    ↪ 1], _segT[node << 1 | 1]);
}
void apply(int node, const Lazy& lazy){
    ::apply(_segT[node], lazy);
    ::apply(_lazy[node], lazy);
}
void push(int node){
    apply(node << 1, _lazy[node]);
    apply(node << 1 | 1, _lazy[node]);
    _lazy[node] = Lazy();
}
void update(int node, int st, int en,
    ↪ int idx, const Node& nodeVal){
    if(en - st == 1){
        _segT[node] = nodeVal;
        return;
    }

```

```

    push(node);
    int md = (st + en) >> 1;
    if(idx < md){
        update(node << 1, st, md, idx,
            ↪ nodeVal);
    }
    else{
        update(node << 1 | 1, md, en,
            ↪ idx, nodeVal);
    }
    pull(node);
}
void update(int idx, const Node&
    ↪ nodeVal){
    update(1, 0, _n, idx, nodeVal);
}
Node rangeQuery(int node, int st, int
    ↪ en, int ql, int qr){
    if(st >= qr || en <= ql) return
        ↪ Node();
    if(st >= ql && en <= qr){
        return _segT[node];
    }
    push(node);
    int md = (st + en) >> 1;
    return _merge(rangeQuery(node << 1,
        ↪ st, md, ql, qr),
        ↪ rangeQuery(node << 1 | 1, md,
        ↪ en, ql, qr));
}
Node rangeQuery(int ql, int qr){
    return rangeQuery(1, 0, _n, ql,
        ↪ qr);
}
void rangeApply(int node, int st, int
    ↪ en, int ql, int qr, const Lazy&
    ↪ lazy){
    if(st >= qr || en <= ql) return;
    if(st >= ql && en <= qr){

```

```

        apply(node, lazy);
        return;
    }
    push(node);
    int md = (st + en) >> 1;
    rangeApply(node << 1, st, md, ql,
        ↪ qr, lazy);
    rangeApply(node << 1 | 1, md, en,
        ↪ ql, qr, lazy);
    pull(node);
}
void rangeApply(int ql, int qr, const
    ↪ Lazy& lazy){
    rangeApply(1, 0, _n, ql, qr, lazy);
}
};
// LazySegTree<Min, int> seg(a);
// For range [i, j] use i, j + 1

```

1.4 PersistentSegmentTree

```

struct PST{
    struct Node{
        int sum = 0;
        int l = 0, r = 0;
    };
    const int n;
    vector<Node> tree;
    int timer = 1;
    PST(int n, int mx_nodes) : n(n),
        ↪ tree(mx_nodes) {}
    Node merge(int l, int r){
        return Node({tree[l].sum +
            ↪ tree[r].sum, l, r});
    }
    int build(int st, int en, const
        ↪ vector<int> &init){
        if(st == en){

```

```

        tree[timer] = Node({init[st],
        ↪ 0, 0});
        return timer ++;
    }
    int md = (st + en) >> 1;
    tree[timer] = merge(build(st, md,
        ↪ init), build(md + 1, en,
        ↪ init));
    return timer ++;
}

int update(int root, int idx, int val,
    ↪ int st, int en){
    if(st == en){
        tree[timer] = Node({val, 0,
        ↪ 0});
        return timer ++;
    }
    int md = (st + en) >> 1;
    if(idx <= md){
        tree[timer] =
        ↪ merge(update(tree[root].l,
        ↪ idx, val, st, md),
        ↪ tree[root].r);
    }
    else{
        tree[timer] =
        ↪ merge(tree[root].l,
        ↪ update(tree[root].r, idx,
        ↪ val, md + 1, en));
    }
    return timer ++;
}

int query(int root, int l, int r, int
    ↪ st, int en){
    if(r < st || en < l) return 0;
    if(l <= st && en <= r) return
        ↪ tree[root].sum;
    int md = (st + en) >> 1;

```

```

        return query(tree[root].l, l, r,
        ↪ st, md) + query(tree[root].r,
        ↪ l, r, md + 1, en);
    }
    int build(const vector<int> &init){
        return build(0, n - 1, init);
    }
    int update(int root, int idx, int val){
        return update(root, idx, val, 0, n
        ↪ - 1);
    }
    int query(int root, int l, int r){
        return query(root, l, r, 0, n - 1);
    }
};

```

1.5 Fenwick_sum

```

struct FenwickTree {
    vector<int> bit; // binary indexed
        ↪ tree
    int n;
    FenwickTree(int n) {
        this->n = n;
        bit.assign(n, 0);
    }
    FenwickTree(vector<int> const &a) :
        ↪ FenwickTree(a.size()) {
        for (size_t i = 0; i < a.size();
        ↪ i++)
            add(i, a[i]);
    }
    int sum(int r) {
        int ret = 0;
        for (; r >= 0; r = (r & (r + 1)) -
        ↪ 1)
            ret += bit[r];
        return ret;
    }
}

```

```

int sum(int l, int r) {
    return sum(r) - sum(l - 1);
}

void add(int idx, int delta) {
    for (; idx < n; idx = idx | (idx +
        ↪ 1))
        bit[idx] += delta;
}
};

```

1.6 SparseTable

```

int log_floor(int x){
    return x ? __builtin_clzll(1) -
        ↪ __builtin_clzll(x) : -1 ;
}

template<typename T , class F =
    ↪ function<T(const T&,const T&)>>
struct SPARSE_TABLE{
    int n;
    vector<vector<T>> st;
    F fun;
    SPARSE_TABLE(const vector<T> &v , const
        ↪ F &f) : fun(f){
        n = static_cast<int>(v.size());
        int maxN = log_floor(n) + 1;
        st.resize(maxN);
        st[0] = v;
        for(int i = 1; i < maxN; i++){
            st[i].resize(n - (1 << i) + 1);

            for(int j = 0; j + (1 << i) <=
                ↪ n; j++){
                st[i][j] = fun(st[i - 1][j]
                    ↪ , st[i - 1][j + (1 <<
                    ↪ (i - 1))]);
            }
        }
    }
}

```

```

T get(int l, int r){
    int h = log_floor(r - l + 1);
    // works for idempotent functions
    ↪ only ykw
    return fun(st[h][l] , st[h][r - (1
    ↪ << h) + 1]);
}
};

```

1.7 MergeSortTree

```

template <typename T>
struct MergeSortTree {
    int n;
    vector<vector<T>> tree;
    MergeSortTree(const vector<T>& a) {
        n = a.size();
        tree.resize(4 * n);
        build(1, 0, n - 1, a);
    }
    void build(int node, int start, int
    ↪ end, const vector<T>& a) {
        if (start == end) {
            tree[node] = {a[start]};
            return;
        }
        int mid = (start + end) / 2;
        build(2 * node, start, mid, a);
        build(2 * node + 1, mid + 1, end,
        ↪ a);
        merge(tree[2 * node].begin(),
        ↪ tree[2 * node].end(),
            tree[2 * node + 1].begin(),
            ↪ tree[2 * node + 1].end(),
            back_inserter(tree[node]));
    }
    T query_lb(int node, int start, int
    ↪ end, int l, int r, T x) {

```

```

    if (start > end || start > r || end
    ↪ < l)
        return
        ↪ numeric_limits<T>::max();
        ↪ // Return INF
    if (start >= l && end <= r) {
        auto it =
        ↪ lower_bound(tree[node].begin(),
        ↪ tree[node].end(), x);
        if (it != tree[node].end())
            return *it;
        return
        ↪ numeric_limits<T>::max();
    }
    int mid = (start + end) / 2;
    return min(query_lb(2 * node,
    ↪ start, mid, l, r, x),
        query_lb(2 * node + 1,
        ↪ mid + 1, end, l, r,
        ↪ x));
}
// Count (number of elements <= x)
int query_count(int node, int start,
    ↪ int end, int l, int r, T x) {
    if (start > end || start > r || end
    ↪ < l) return 0;
    if (start >= l && end <= r) {
        // upper_bound gives first
        ↪ element > x, so index
        ↪ gives count of elements <=
        ↪ x
        return
        ↪ upper_bound(tree[node].begin(),
        ↪ tree[node].end(), x) -
        ↪ tree[node].begin();
    }
    int mid = (start + end) / 2;
    return query_count(2 * node, start,
    ↪ mid, l, r, x) +

```

```

        query_count(2 * node + 1,
        ↪ mid + 1, end, l, r, x);
    }
    // Public Interface: Find smallest
    ↪ number >= x in range [l, r]
    T lower_bound_val(int l, int r, T x) {
        return query_lb(1, 0, n - 1, l, r,
        ↪ x);
    }
    // Public Interface: Count numbers <=
    ↪ x in range [l, r]
    int count_less_equal(int l, int r, T x)
    ↪ {
        return query_count(1, 0, n - 1, l,
        ↪ r, x);
    }
};

```

1.8 PolyMul

```

namespace GetPrimitive{
    vi Divisors;
    vi Divisor(lli x){vi ans;
        for(lli i=2;i*i<=x;i++){
            if(x%i==0){
                ans.EB(i);
                ↪ while(x%i==0)x/=i;
            }
        }
        if(x>1)ans.EB(x);return ans;
    }
    bool check(int prim,int p,vi &divs){
        for(auto v:divs){
            if(binpow(prim,(p-1)/v,p)==1)
                ↪ return 0;
        }return 1;
    }
    int getRoot(int p){
        int ans=2;vi divs=Divisor(p-1);

```

```

        while(!check(ans,p,divs))ans++;
        return ans;
    }
};
//7340033      5
//998244353    3
const lli MAXB = 1 << 21;
using polybase = lli;
polybase A[MAXB], B[MAXB], C[MAXB];
template<lli NTTMOD, lli PRIMITIVE_ROOT>
→ struct POLYMUL{
    lli modInv(lli a) { return a <= 1 ? a :
        → (long long) (NTTMOD - NTTMOD / a) *
        → modInv(NTTMOD % a) % NTTMOD; }
    void NTT(polybase P[], lli n, lli oper)
        → {
        for(int i=1,j=0;i<n-1;i++){
            for(int s=n;j^=s>>=1,~j&s;);
            if(i<j)swap(P[i],P[j]);
        }
        for(int d=0;(1<<d)<n;d++){
            int m=1<<d,m2=m*2;
            lli unit_p0=
                → ::binpow(PRIMITIVE_ROOT,
                → (NTTMOD-1)/m2,NTTMOD);
            if(oper<0)unit_p0=modInv(unit_p0);
            for(int i=0;i<n;i+= m2) {
                polybase unit = 1;
                for (int j = 0; j < m; j++) {
                    polybase &P1 = P[i + j + m],
                        → &P2 = P[i + j];
                    polybase t = unit * P1 %
                        → NTTMOD;
                    P1 = (P2 - t + NTTMOD) %
                        → NTTMOD; P2 = (P2 + t) %
                        → NTTMOD;
                    unit = unit * unit_p0 % NTTMOD;
                }
            }
        }
}

```

```

    }
}
vector<polybase> mul(const
    → vector<polybase> &a, const
    → vector<polybase> &b) {
    vector<polybase> ret(max(OLL, (lli)
        → a.size() + (lli) b.size() - 1),
        → 0);
    int len = 1; while (len <
        → (lli)ret.size()) len <= 1;
    for (int i = 0; i < len; i++) A[i]
        → = i < (int)a.size() ? a[i] : 0;
    for (int i = 0; i < len; i++) B[i]
        → = i < (int)b.size() ? b[i] : 0;
    NTT(A, len, 1); NTT(B, len, 1);
    for (lli i = 0; i < len; i++) C[i]
        → = (polybase) A[i] * B[i] %
        → NTTMOD;
    NTT(C, len, -1); for (lli i = 0,
        → inv = modInv(len); i <
        → (lli)ret.size(); i++) ret[i] =
        → (long long) C[i] * inv %
        → NTTMOD;
    return ret;
}
vi binpow(vi b, lli p){
    vi ans=vi(1,1);
    for(;p;p>>=1){
        if(p&1)ans=mul(ans,b);
        b=mul(b,b);
    }
    return ans;
}
vi calc(vi &arr,int l,int r){
    if(l==r){
        //base case;
        return {};
    }
    int mid=(l+r)>>1;

```

```

        vi x=calc(arr,l,mid),
        → y=calc(arr,mid+1,r);
        return mul(x,y);
    }
};
POLYMUL<998244353,3> ntt;
lli phi(lli x){// will be a power of 2
    → mostly.
    if(x==1)return 1;
    else return x/2;
}
vi arr[20];
void solve(){
    lli n,k,q;
    arr[1].assign(100001,0);
    cin>>n>>k;
    vi divs;
    lli cur=1;
    while(n%cur==0){
        divs.EB(cur);
        cur*=2;
    }
    fr(i,k){
        lli x;cin>>x;arr[1][x]++;
    }
    rep(i,2,19){
        arr[i]=ntt.mul(arr[i-1],arr[i-1]);
        if(arr[i].size()>100001)
            → arr[i].resize(100001);
    }
    sort(all(divs));
    cin>>q;
    fr(_i,q){
        lli S;cin>>S;
        lli ans=0;
        for(int i=0;i<divs.size();i++){
            if(S%(n/divs[i])!=0)continue;

```

```

        ans=(ans+
        ↪ arr[i+1][S/(n/divs[i])] *
        ↪ phi(n/divs[i]))%MOD;
    }
    ans=ans*binpow(n,MOD-2,MOD)%MOD;
    cout<<ans<<endl;
}

// youKnowWho's FFT
using cd = complex<double>;
const double PI = acos(-1);
void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    for (int i = 1, j = 0; i < n; i++) {
        int bit = n >> 1;
        for (; j & bit; bit >>= 1) j ^=
            ↪ bit;
        j ^= bit;
        if (i < j) swap(a[i], a[j]);
    }
    for (int len = 2; len <= n; len <= 1)
        ↪ {
            double ang = 2 * PI / len * (invert
            ↪ ? -1 : 1);
            cd wlen(cos(ang), sin(ang));
            for (int i = 0; i < n; i += len) {
                cd w(1);
                for (int j = 0; j < len / 2;
                ↪ j++) {
                    cd u = a[i+j], v =
                    ↪ a[i+j+len/2] * w;
                    a[i+j] = u + v;
                    a[i+j+len/2] = u - v;
                    w *= wlen;
                }
            }
        }
    if (invert) {
        for (cd & x : a) x /= n;
    }
}

```

```

    }
}
vector<int> multiply(vector<int> const& a,
    ↪ int n){
    vector<cd> fa(n);
    for (int i = 0; i < (int)a.size(); i++)
        ↪ fa[i] = a[i];
    for(int i = a.size(); i < n; i++)
        ↪ fa[i] = 0;
    fft(fa, false);
    for(int i = 0; i < n; i++){
        fa[i] *= fa[i];
    }
    fft(fa, true);
    vector<int> res(n);
    for(int i = 0; i < n; i++){
        res[i] = round(fa[i].real());
    }
    return res;
}
vector<int> p(MAXN);
void solve(){
    int n; cin>>n;
    int mx = 1;
    vector<int> s(n + 1);
    for(int i = 1; i <= n; i++){
        cin>>s[i];
        mx = max(mx, s[i]);
    }
    mx += 1000;
    for(int i = 1; i <= n; i++) p[s[i]] =
        ↪ 1;
    n = 1;
    while(n <= MAXN) n <= 1;
    vector<int> res = multiply(p, n);
    int c = 0;
    for(int i = 2; i <= 2 * mx; i += 2){
        int md = i / 2;
        if(p[md]){

```

```

        c += (res[i] - 1) / 2;
    }
}

```

1.9 FWT

Given two arrays A and B of size $n = 2^k$, compute an array C such that

$$C[x] = \sum_{\substack{i \oplus j = x}} A[i] \cdot B[j].$$

```

template <typename T>
struct FWT {
    // ===== XOR =====
    void fwt_xor(T a[], int n) {
        for (int d = 1; d < n; d <= 1) {
            for (int i = 0; i < n; i += (d
            ↪ < 1)) {
                for (int j = 0; j < d; j++)
                    ↪ {
                        T x = a[i + j];
                        T y = a[i + j + d];
                        a[i + j] = x + y;
                        a[i + j + d] = x - y;
                    }
            }
        }
    }
    void ufwt_xor(T a[], int n) {
        for (int d = 1; d < n; d <= 1) {
            for (int i = 0; i < n; i += (d
            ↪ < 1)) {
                for (int j = 0; j < d; j++)
                    ↪ {
                        T x = a[i + j];
                        T y = a[i + j + d];

```

```

        a[i + j]      = (x + y)
        ↪ >> 1;
        a[i + j + d] = (x - y)
        ↪ >> 1;
    }
}
}
}
// ===== AND =====
void fwt_and(T a[], int n) {
    for (int d = 1; d < n; d <= 1) {
        for (int i = 0; i < n; i += (d
            ↪ << 1)) {
            for (int j = 0; j < d; j++)
                ↪ {
                    a[i + j] += a[i + j +
                        ↪ d];
                }
            }
        }
    }
}
void ufwt_and(T a[], int n) {
    for (int d = 1; d < n; d <= 1) {
        for (int i = 0; i < n; i += (d
            ↪ << 1)) {
            for (int j = 0; j < d; j++)
                ↪ {
                    a[i + j] -= a[i + j +
                        ↪ d];
                }
            }
        }
    }
}
// ===== OR =====
void fwt_or(T a[], int n) {
    for (int d = 1; d < n; d <= 1) {
        for (int i = 0; i < n; i += (d
            ↪ << 1)) {

```

```

        for (int j = 0; j < d; j++)
            ↪ {
                a[i + j + d] += a[i +
                    ↪ j];
            }
        }
    }
}
void ufwt_or(T a[], int n) {
    for (int d = 1; d < n; d <= 1) {
        for (int i = 0; i < n; i += (d
            ↪ << 1)) {
            for (int j = 0; j < d; j++)
                ↪ {
                    a[i + j + d] -= a[i +
                        ↪ j];
                }
            }
        }
    }
}
// ===== CONVOLUTIONS =====
void xor_convolution(T a[], T b[], int
    ↪ n) {
    fwt_xor(a, n);
    fwt_xor(b, n);
    for (int i = 0; i < n; i++) a[i] *=
        ↪ b[i];
    ufwt_xor(a, n);
}
void and_convolution(T a[], T b[], int
    ↪ n) {
    fwt_and(a, n);
    fwt_and(b, n);
    for (int i = 0; i < n; i++) a[i] *=
        ↪ b[i];
    ufwt_and(a, n);
}
void or_convolution(T a[], T b[], int
    ↪ n) {

```

```

        fwt_or(a, n);
        fwt_or(b, n);
        for (int i = 0; i < n; i++) a[i] *=
            ↪ b[i];
        ufwt_or(a, n);
    }
};

```

2 Graphs

2.1 Dinic

```

template<class T>
struct Flow {
    const int n;
    struct Edge {
        int to;
        T cap;
        Edge(int to, T cap) : to(to),
            ↪ cap(cap) {}
    };
    std::vector<Edge> e;
    std::vector<std::vector<int>>> g;
    std::vector<int> cur, h;
    Flow(int n) : n(n), g(n) {}
    bool bfs(int s, int t) {
        h.assign(n, -1);
        std::queue<int> que;
        h[s] = 0;
        que.push(s);
        while (!que.empty()) {
            const int u = que.front();
            que.pop();
            for (int i : g[u]) {
                auto [v, c] = e[i];
                if (c > 0 && h[v] == -1) {
                    h[v] = h[u] + 1;
                    if (v == t) {

```

```

        return true;
    }
    que.push(v);
}
}
return false;
}
T dfs(int u, int t, T f) {
    if (u == t) return f;
    auto r = f;
    for (int &i = cur[u]; i <
        → (int)(g[u].size()); ++i) {
        const int j = g[u][i];
        auto [v, c] = e[j];
        if (c > 0 && h[v] == h[u] + 1)
            → {
                auto a = dfs(v, t,
                    → std::min(r, c));
                e[j].cap -= a;
                e[j ^ 1].cap += a;
                r -= a;
                if (r == 0) {
                    return f;
                }
            }
    }
    return f - r;
}
void addEdge(int u, int v, T c) {
    g[u].push_back(e.size());
    e.emplace_back(v, c);
    g[v].push_back(e.size());
    e.emplace_back(u, 0);
}
T maxFlow(int s, int t) {
    T ans = 0;
    while (bfs(s, t)) {
        cur.assign(n, 0);

```

```

        ans += dfs(s, t,
            → std::numeric_limits<T>::max());
    }
    return ans;
}
void get(int end){
    vector<int> path;
    cur.assign(n, 0);
    int u = 1;
    while(u != end){
        path.push_back(u);
        dbg(path)
        for (int &i = cur[u]; i <
            → (int)(g[u].size()); ++i) {
            const int j = g[u][i];
            auto [v, c] = e[j];
            dbg(u, v, c, j)
            if (j % 2 == 0 && c == 0) {
                e[j].cap = 1;
                u = v;
                break;
            }
        }
        path.push_back(end);
        cout<<path.size()<<endl;
        for(int i : path) cout<<i<<" ";
        cout<<endl;
    }
}
queue<int> q;
vector<bool> vis(n + 1, false);
q.push(1);
vis[1] = true;
while(!q.empty()){
    int u = q.front(); q.pop();
    for(auto i: mf.g[u]){
        auto [v, c] = mf.e[i];
        if(c > 0 && !vis[v]){

```

```

        vis[v] = true;
        q.push(v);
    }
}
for(int i = 1; i <= n; i++){
    for(auto id : mf.g[i]){
        auto [v, c] = mf.e[id];
        if(v != s and c == 0){
            cout<<i<<" "<<v - n<<endl;
        }
    }
}



## 2.2 LCA

// LCA and Binary Lifting
void dfs(int v, int p){
    up[0][v] = p;
    for(int i = 1; i < 20; i++){
        up[i][v] = up[i - 1][up[i - 1][v]];
    }
    for(auto x : g[v]){
        if(x == p) continue;
        dst[x] = dst[v] + 1;
        dfs(x, v);
    }
}
auto lca = [&](int x, int y){
    if(dst[x] < dst[y]) swap(x, y);
    for(int i = 19; i >= 0; i--){
        if(dst[up[i][x]] >= dst[y]){
            x = up[i][x];
        }
    }
    if(x == y) return x;
    for(int i = 19; i >= 0; i--){
        if(up[i][x] != up[i][y]){

```



```

        x = up[i][x];
        y = up[i][y];
    }
}
return up[0][x];
};

```

2.3 CutPoint

```

vll g[NUM];
ll in[NUM], low[NUM], vis[NUM];
int timer = 0;
void CutPoints(ll src, ll parent = -1){
    vis[src] = true;
    in[src] = low[src] = timer;
    timer++;
    int children = 0;
    for (auto child : g[src]) {
        if (child == parent)
            continue;
        if (vis[child]){
            low[src] = min(low[src],
                → in[child]);
        } else{
            CutPoints(child, src);
            low[src] = min(low[src],
                → low[child]);
            if (low[child] >= in[src] &&
                → parent!=-1)
            {
                // Src is a cut point!
            }
            children++;
        }
    }
    if (parent==-1 && children>1){
        // Root is a cut point!
    }
}

```

```

}

```

2.4 CentroidDecomp

```

int nn;
void dfs1(int u, int p)
{
    sub[u] = 1;
    nn++;
    for (auto it = g[u].begin(); it !=
        → g[u].end(); it++)
        if (*it != p)
        {
            dfs1(*it, u);
            sub[u] += sub[*it];
        }
}
int dfs2(int u, int p)
{
    for (auto it = g[u].begin(); it !=
        → g[u].end(); it++)
        if (*it != p && sub[*it] > nn / 2)
            return dfs2(*it, u);
    return u;
}
void decompose(int root, int p)
{
    nn = 0;
    dfs1(root, root);
    int centroid = dfs2(root, root);
    if (p == -1)
        p = centroid;
    par[centroid] = p;
    for (auto it = g[centroid].begin(); it
        → != g[centroid].end(); it++)
    {
        g[*it].erase(centroid);
        decompose(*it, centroid);
    }
}

```

```

    g[centroid].clear();
}

```

2.5 VirtualTree

```

const int MAXN = 200005;
vector<int> adj[MAXN];
int tin[MAXN], tout[MAXN], timer;
int up[MAXN][20], depth[MAXN];
void dfs_lca(int u, int p, int d) {
    tin[u] = ++timer;
    depth[u] = d;
    up[u][0] = p;
    for (int i = 1; i < 20; i++)
        up[u][i] = up[up[u][i-1]][i-1];

    for (int v : adj[u]) {
        if (v != p) dfs_lca(v, u, d + 1);
    }
    tout[u] = timer;
}
bool is_ancestor(int u, int v) {
    return tin[u] <= tin[v] && tout[u] >=
        → tout[v];
}
int lca(int u, int v) {
    if (is_ancestor(u, v)) return u;
    if (is_ancestor(v, u)) return v;
    for (int i = 19; i >= 0; i--) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}
vector<int> vt_adj[MAXN];
bool compare_tin(int a, int b) {
    return tin[a] < tin[b];
}

```

```

void build_virtual_tree(vector<int>& nodes)
→ {
    sort(nodes.begin(), nodes.end(),
→ compare_tin);
    // 2. Add LCAs of adjacent sorted
→ nodes to the set
    // This guarantees all necessary
→ connecting nodes are present
    int k = nodes.size();
    for (int i = 0; i < k - 1; i++) {
        nodes.push_back(lca(nodes[i],
→ nodes[i+1]));
    }
    // 3. Sort again and remove duplicates
    sort(nodes.begin(), nodes.end(),
→ compare_tin);
    nodes.erase(unique(nodes.begin(),
→ nodes.end()), nodes.end());
    // 4. Build edges using a stack
    // Clear previous virtual tree edges
→ for these specific nodes
    for (int u : nodes) vt_adj[u].clear();
    vector<int> stack;
    stack.push_back(nodes[0]);
    for (int i = 1; i < nodes.size(); i++)
→ {
        int u = nodes[i];
        // Pop stack while the top is NOT
→ an ancestor of u
        // (This means we finished the
→ subtree of stack.back())
        while (stack.size() > 1 &&
→ !is_ancestor(stack.back(), u))
→ {
            stack.pop_back();
        }
        // Add edge from the current
→ direct ancestor to u

```

```

    // Note: The graph is directed
→ downwards usually, or
→ undirected depending on need
    vt_adj[stack.back()].push_back(u);
    vt_adj[u].push_back(stack.back());
→ // Add this if undirected
    stack.push_back(u);
}
}

```

2.6 Centroid

```

function<void(int, int)> dfs2 = [&](int v,
→ int p){
    int mx = 0;
    for(auto u: g[v]){
        if(u == p || vis[u]) continue;
        mx = max(mx, sbt[u]);
    }
    mx = max(mx, n - sbt[v]);
    if(mx <= n / 2){
        centroid = v;
    }
    for(auto u: g[v]){
        if(u == p || vis[u]) continue;
        dfs2(u, v);
    }
}
};

```

2.7 Mcmf

```

using i64 = long long;
template<class T>
struct MinCostFlow {
    struct _Edge {
        int to;
        T cap;
        T cost;

```

```

        _Edge(int to_, T cap_, T cost_) :
→ to(to_), cap(cap_), cost(cost_)
→ {}
    };
    int n;
    std::vector<_Edge> e;
    std::vector<std::vector<int>>> g;
    std::vector<T> h, dis;
    std::vector<int> pre;
    bool dijkstra(int s, int t) {
        dis.assign(n,
→ std::numeric_limits<T>::max());
        pre.assign(n, -1);
        std::priority_queue<std::pair<T,
→ int>, std::vector<std::pair<T,
→ int>>>,
→ std::greater<std::pair<T,
→ int>>>> que;
        dis[s] = 0;
        que.emplace(0, s);
        while (!que.empty()) {
            T d = que.top().first;
            int u = que.top().second;
            que.pop();
            if (dis[u] != d) {
                continue;
            }
            for (int i : g[u]) {
                int v = e[i].to;
                T cap = e[i].cap;
                T cost = e[i].cost;
                if (cap > 0 && dis[v] > d +
→ h[u] - h[v] + cost) {
                    dis[v] = d + h[u] -
→ h[v] + cost;
                    pre[v] = i;
                    que.emplace(dis[v], v);
                }
            }
        }
    }
}

```

```

    }
    return dis[t] !=
        ↪ std::numeric_limits<T>::max();
}
MinCostFlow() {}
MinCostFlow(int n_) {
    init(n_);
}
void init(int n_) {
    n = n_;
    e.clear();
    g.assign(n, {});
}
void addEdge(int u, int v, T cap, T
    ↪ cost) {
    g[u].push_back(e.size());
    e.emplace_back(v, cap, cost);
    g[v].push_back(e.size());
    e.emplace_back(u, 0, -cost);
}
std::pair<T, T> flow(int s, int t) {
    T flow = 0;
    T cost = 0;
    h.assign(n, 0);
    while (dijkstra(s, t)) {
        for (int i = 0; i < n; ++i) {
            h[i] += dis[i];
        }
        T aug =
            ↪ std::numeric_limits<int>::max();
        for (int i = t; i != s; i =
            ↪ e[pre[i] ^ 1].to) {
            aug = std::min(aug,
                ↪ e[pre[i]].cap);
        }
        for (int i = t; i != s; i =
            ↪ e[pre[i] ^ 1].to) {
            e[pre[i]].cap -= aug;
            e[pre[i] ^ 1].cap += aug;

```

```

    }
    flow += aug;
    cost += aug * h[t];
}
return std::make_pair(flow, cost);
}
struct Edge {
    int from;
    int to;
    T cap;
    T cost;
    T flow;
};
std::vector<Edge> edges() {
    std::vector<Edge> a;
    for (int i = 0; i < e.size(); i +=
        ↪ 2) {
        Edge x;
        x.from = e[i + 1].to;
        x.to = e[i].to;
        x.cap = e[i].cap + e[i +
            ↪ 1].cap;
        x.cost = e[i].cost;
        x.flow = e[i + 1].cap;
        a.push_back(x);
    }
    return a;
}
};

```

2.8 Hld

```

struct hld {
    vector<vector<int>> g;
    vector<int> par, dpth, heavy, root,
        ↪ pos, subt, inv;
    vector<vector<int>> up;
    segtree seg;
    int cur_pos = 0;

```

```

hld(int n){
    g.resize(n + 1), par.resize(n + 1),
        ↪ dpth.resize(n + 1);
    heavy.resize(n + 1, -1),
        ↪ root.resize(n + 1),
        ↪ pos.resize(n + 1), inv.resize(n
            ↪ + 1);
    subt.resize(n + 1, 1);
    up.resize(n + 1);
    for(int i = 0; i <= n; i++){
        up[i].resize(20, 0);
    }
    seg = segtree(n + 1);
    cur_pos = 0;
}
void add_edge(int u, int v){
    g[u].push_back(v);
    g[v].push_back(u);
}
void run(int rt){
    dfs_sz(rt, 0);
    dfs_hld(rt, rt);
}
int dfs_sz(int v, int p){
    int sz = 1, mx_sz = 0;
    par[v] = p;
    up[v][0] = p;
    for(int i = 1; i < 20; i++){
        up[v][i] = up[up[v][i - 1]][i -
            ↪ 1];
    }
    for(auto x : g[v]){
        if(x == p) continue;
        dpth[x] = dpth[v] + 1;
        int c_sz = dfs_sz(x, v);
        sz += c_sz;
        if(c_sz > mx_sz){
            mx_sz = c_sz;

```

```

        heavy[v] = x;
    }
}
return subt[v] = sz;
}
void dfs_hld(int v, int h){
    root[v] = h;
    pos[v] = cur_pos, inv[cur_pos] = v;
    cur_pos ++;
    if(heavy[v] != -1)
        ↪ dfs_hld(heavy[v], h);
    for(auto x : g[v]){
        if(x == par[v] || x ==
            ↪ heavy[v]) continue;
        dfs_hld(x, x);
    }
}
bool is_anc(int a, int b) {
    return pos[a] <= pos[b] && pos[b] <
        ↪ pos[a] + subt[a];
}
int kth(int v, int k) {
    for (int i = 0; i < 20; i++) {
        if (k & (1 << i)) {
            v = up[v][i];
        }
    }
    return v;
}
int hehe(int v, int u) {
    // u is ancestor of v
    int dif = dpth[v] - dpth[u] - 1;
    return kth(v, dif);
}
mint qry(int a, int b) {
    mint res = 0;
    for (; root[a] != root[b]; b =
        ↪ par[root[b]]) {

```

```

        if (dpth[root[a]] >
            ↪ dpth[root[b]])
            swap(a, b);
        mint cur_heavy_path_max =
            ↪ seg.prod(pos[root[b]],
            ↪ pos[b] + 1).x;
        res += cur_heavy_path_max;
    }
    if (dpth[a] > dpth[b])
        swap(a, b);
    mint last_heavy_path_max =
        ↪ seg.prod(pos[a], pos[b] + 1).x;
    res += last_heavy_path_max;
    return res;
}
};

```

2.9 Euler

```

// Hierholzer's Algorithm undirected
// check remove the seen for directed
void dfs(int node) {
    while (!g[node].empty()) {
        auto [son, idx] = g[node].back();
        g[node].pop_back();
        if (seen[idx]) { continue; }
        seen[idx] = true;
        dfs(son);
    }
    path.push_back(node);
}
for (int node = 0; node < n; node++) {
    if (degree[node] % 2) {
        cout << "IMPOSSIBLE" << endl;
        return 0;
    }
}
dfs(0);
if (path.size() != m + 1) {

```

```

    cout << "IMPOSSIBLE";
}

2.10 Kosaraju

vector<int> g[n + 1], rg[n + 1];
for(auto [x, y, w] : eds){
    g[x].push_back(y);
    rg[y].push_back(x);
}
vector<int> order;
vector<int> vis(n + 1, -1);
function<void(int)> dfs = [&](int v){
    vis[v] = 1;
    for(auto x : g[v]){
        if(vis[x] == -1) dfs(x);
    }
    order.push_back(v);
};
for(int i = 1; i <= n; i++) if(vis[i] ==
    ↪ -1) dfs(i);
reverse(al(order));
int idx = 1;
for(int i = 1; i <= n; i++) vis[i] = -1;
function<void(int, vector<int>&)> dfs2 =
    ↪ [&](int v, vector<int>& cmp){
    vis[v] = 1;
    for(auto x : rg[v]){
        if(vis[x] == -1) dfs2(x, cmp);
    }
    cmp.push_back(v);
};
vector<int> dag(n + 1, -1);
for(auto x : order){
    if(vis[x] == -1){
        vector<int> cmp;
        dfs2(x, cmp);
        for(auto y : cmp) dag[y] = idx;
        idx ++;
    }
}

```

```

    }
}
set<int> ng[idx], idg(idx, 0);
for(int i = 1; i <= n; i++){
    for(auto x : g[i]){
        if(dag[x] != dag[i]){
            ng[dag[i]].insert(dag[x]);
        }
    }
}
for(int i = 1; i < idx; i++){
    for(auto x : ng[i]) idg[x]++;
}
queue<int> q;
vector<int> dp(idx, 0);
for(int i = 1; i < idx; i++){
    if(idg[i] == 0){
        q.push(i);
        dp[i] = 1;
    }
}
while(!q.empty()){
    auto v = q.front(); q.pop();
    for(auto x : ng[v]){
        idg[x]--;
        dp[x] = max(dp[x], dp[v] + 1);
        if(!idg[x]) q.push(x);
    }
}

```

2.11 Bridges

```

// Bridges
vector<int> tin(n + 1, -1), low(n + 1, -1),
    ↪ vis(n + 1, 0);
int clk = 0;
vector<pair<int, int>> ans;
function<void(int, int)> dfs = [&](int v,
    ↪ int p){

```

```

    vis[v] = 1;
    tin[v] = low[v] = clk++;
    for(auto x : g[v]){
        if(x == p) continue;
        if(!vis[x])
            dfs(x, v);
        low[v] = min(low[v], low[x]);
        if(low[x] > tin[v]){
            ans.push_back({v, x});
        }
    }
};
// Articulation Points
function<void(int, int)> dfs = [&](int v,
    ↪ int p){
    vis[v] = 1;
    tin[v] = low[v] = clk++;
    int c = 0;
    for(auto x : g[v]){
        if(x == p) continue;
        if(vis[x]){
            low[v] = min(low[v], tin[x]);
        }
        else{
            dfs(x, v);
            low[v] = min(low[v], low[x]);
            if(low[x] >= tin[v] && p !=
                ↪ -1){
                ans.insert(v);
            }
            c++;
        }
    }
    if(c > 1 && p == -1){
        ans.insert(v);
    }
}
};

```

2.12 Toposort

```

vector<int> topoSort(const
    ↪ vector<vector<int>>& gr) {
    vector<int> indeg(n), q;
    for (auto& li : gr) for (int x : li)
        ↪ indeg[x]++;
    rep(i, 0, n) if (indeg[i] == 0)
        ↪ q.push_back(i);
    rep(j, 0, sz(q)) for (int x : gr[q[j]])
        if (--indeg[x] == 0) q.push_back(x);
    return q;
}

```

2.13 Distances

```

// Bellman Ford
vector<int> d(n, INF);
    d[v] = 0;
    for (int i = 0; i < n - 1; ++i)
        for (Edge e : edges)
            if (d[e.a] < INF)
                d[e.b] = min(d[e.b], d[e.a]
                    ↪ + e.cost);

// Dijkstra Algorithm
void dijkstra(int s, vector<int> & d,
    ↪ vector<int> & p) {
    int n = adj.size();
    d.assign(n, INF); p.assign(n, -1);
    d[s] = 0;
    using pii = pair<int, int>;
    priority_queue<pii, vector<pii>,
        ↪ greater<pii>> q;
    q.push({0, s});
    while (!q.empty()) {
        int v = q.top().second;
        int d_v = q.top().first;
        q.pop();
        if (d_v != d[v])

```

```

        continue;
    for (auto edge : adj[v]) {
        int to = edge.first;
        int len = edge.second;
        if (d[v] + len < d[to]) {
            d[to] = d[v] + len;
            p[to] = v;
            q.push({d[to], to});
        }
    }
}

// Floyd Warshall
for (int k = 0; k < n; ++k) {
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (d[i][k] < INF && d[k][j] <
                → INF)
                d[i][j] = min(d[i][j],
                    → d[i][k] + d[k][j]);
        }
    }
}

```

3 Misc

3.1 Dsu_dynconn

```

struct dsu_state{
    int v, u, rnk_v, rnk_u;
    dsu_state() {}
    dsu_state(int v, int u, int rnk_v, int
        → rnk_u)
        : v(v), u(u), rnk_v(rnk_v),
          → rnk_u(rnk_u) {}
};

struct dsuRoll{
    vector<int> par, rnk;
    stack<dsu_state> st;

```

```

    int comps;
    dsuRoll() {}
    dsuRoll(int n){
        par.resize(n + 1), rnk.resize(n +
            → 1, 0);
        for(int i = 1; i <= n; i++){
            par[i] = i;
            rnk[i] = 0;
        }
        comps = n;
    }
    int find(int v){
        return (v == par[v]) ? v :
            → find(par[v]);
    }
    bool unite(int x, int y){
        x = find(x), y = find(y);
        if(x == y) return false;
        if(rnk[x] > rnk[y]) swap(x, y);
        st.push(dsu_state(x, y, rnk[x],
            → rnk[y]));
        par[x] = y;
        if(rnk[x] == rnk[y]) rnk[y] ++;
        comps --;
        return true;
    }
    void rollback(){
        if(st.empty()) return;
        auto cur = st.top(); st.pop();
        comps ++;
        par[cur.v] = cur.v;
        par[cur.u] = cur.u;
        rnk[cur.v] = cur.rnk_v;
        rnk[cur.u] = cur.rnk_u;
    }
};

struct qry{
    int v, u;
    bool f;

```

```

    qry() {}
    qry(int v, int u) : v(v), u(u) {}
};

struct dynCon{
    int n, sz;
    vector<vector<qry>> seg;
    dsuRoll dsu;
    dynCon(int _n, int _q) : n(_n), sz(_q)
        → {
        dsu = dsuRoll(_n);
        seg.resize(4 * _q + 5);
    }
    void add(int idx, int st, int en, int
        → ql, int qr, qry& q){
        if(st > qr || en < ql) return;
        if(st >= ql && en <= qr){
            seg[idx].push_back(q);
            return;
        }
        int md = (st + en) >> 1;
        add(idx << 1, st, md, ql, min(qr,
            → md), q);
        add(idx << 1 | 1, md + 1, en,
            → max(md + 1, ql), qr, q);
    }
    void add(int l, int r, qry q){
        add(1, 0, sz, l, r, q);
    }
    void run(int v, int l, int r,
        → vector<int>& ans){
        for(auto &q : seg[v]) q.f =
            → dsu.unite(q.v, q.u);
        if(l == r) { /* do something */ }
        else{
            int md = (l + r) >> 1;
            run(v << 1, l, md, ans);
            run(v << 1 | 1, md + 1, r,
                → ans);
        }
    }
};

```

```

    }
    for(auto &q : seg[v]) if(q.f)
        ↪ dsu.rollback();
    }
    vector<int> solve(){
        vector<int> ans(sz + 1);
        run(1, 0, sz, ans);
        return ans;
    }
};
map<pair<int, int>, int> mp;
vector<int> ans(q + 1);
dynCon dc(n, q);
for(int i = 0; i < q; i ++){
    char c; fscanf(in, " %c", &c);
    if(c == '?') ans[i] = 1;
    else{
        int v, u;
        fscanf(in, "%d %d", &v, &u);
        if(v > u) swap(v, u);
        if(c == '+'){
            mp[{v, u}] = i;
        }
        else{
            dc.add(mp[{v, u}], i - 1,
                ↪ qry(v, u));
            mp.erase({v, u});
        }
    }
}
for(auto [p, start] : mp){
    dc.add(start, q, qry(p.first,
        ↪ p.second));
}
auto f = dc.solve();

```

3.2 IntervalContainer

```

/* Description: Add and remove intervals
   ↪ from a set of disjoint intervals.
   Will merge the added interval with any
   ↪ overlapping intervals in the set when
   adding. Intervals are [inclusive,
   ↪ exclusive). */
set<pii>::iterator addInterval(set<pii>&
    ↪ is, int L, int R) {
    if (L == R) return is.end();
    auto it = is.lower_bound({L, R}),
        ↪ before = it;
    while (it != is.end() && it->first <=
        ↪ R) {
        R = max(R, it->second);
        before = it = is.erase(it);
    }
    if (it != is.begin() && (--it)->second
        ↪ >= L) {
        L = min(L, it->first);
        R = max(R, it->second);
        is.erase(it);
    }
    return is.insert(before, {L,R});
}
void removeInterval(set<pii>& is, int
    ↪ L, int R) {
    if (L == R) return;
    auto it = addInterval(is, L, R);
    auto r2 = it->second;
    if (it->first == L) is.erase(it);
    else (int&)it->second = L;
    if (R != r2) is.emplace(R, r2);
}

```

3.3 2SAT_Shrey

```

struct two_sat {
    int n;

```

```

    vector<vector<int>> g, gr;
    ↪ // gr is the reversed graph
    vector<int> comp, topological_order,
    ↪ answer; // comp[v]: ID of the SCC
    ↪ containing node v
    vector<bool> vis;
    two_sat() {}
    two_sat(int _n) { init(_n); }
    void init(int _n){
        n = _n;
        g.assign(2 * n, vector<int>());
        gr.assign(2 * n, vector<int>());
        comp.resize(2 * n);
        vis.resize(2 * n);
        answer.resize(2 * n);
    }
    void add_edge(int u, int v){
        g[u].push_back(v);
        gr[v].push_back(u);
    }
    // For the following three functions
    // int x, bool val: if 'val' is true,
    ↪ we take the variable to be x.
    ↪ Otherwise we take it to be x's
    ↪ complement.
    void add_clause_or(int i, bool f, int
        ↪ j, bool g){
        add_edge(i + (f ? n : 0), j + (g ?
            ↪ 0 : n));
        add_edge(j + (g ? n : 0), i + (f ?
            ↪ 0 : n));
    }
    void add_clause_xor(int i, bool f, int
        ↪ j, bool g){
        add_clause_or(i, f, j, g);
        add_clause_or(i, !f, j, !g);
    }
    void add_clause_and(int i, bool f, int
        ↪ j, bool g){

```

```

    add_clause_xor(i, !f, j, g);
}

void dfs(int u){
    vis[u] = true;

    for (const auto &v : g[u])
        if (!vis[v])
            dfs(v);

    topological_order.push_back(u);
}

void scc(int u, int id){
    vis[u] = true;
    comp[u] = id;
    for (const auto &v : gr[u])
        if (!vis[v])
            scc(v, id);
}

// Returns true if the given
→ proposition is satisfiable and
→ constructs a valid assignment
bool satisfiable() {
    fill(vis.begin(), vis.end(),
        → false);
    for (int i = 0; i < 2 * n; i++)
        if (!vis[i])
            dfs(i);
    fill(vis.begin(), vis.end(),
        → false);
    reverse(topological_order.begin(),
        → topological_order.end());
    int id = 0;
    for (const auto &v :
        → topological_order)
        if (!vis[v])
            scc(v, id++);
    for (int i = 0; i < n; i++){
        if (comp[i] == comp[i + n])
            return false;

```

```

        answer[i] = (comp[i] > comp[i +
            → n] ? 1 : 0);
    }

    return true;
}
};

```

3.4 Hungarian

```

// ans[i] = j means ith worker is assigned
→ jth task
// O(n^3) time complexity
// It finds the assignment where the sum
→ of costs is the lowest possible value.
void hungarian(vector<vector<long double>>
→ &a, vll &ans){
    ll n = a.size();
    vector<ld> u(n + 1, 0), v(n + 1, 0);
    vll p(n + 1, 0), way(n + 1, 0);
    for (ll i = 1; i <= n; ++i) {
        vector<ld> minv(n + 1, INF);
        vector<bool> used(n + 1, false);
        ll j0 = 0;
        p[0] = i;
        do{
            used[j0] = true;
            ll i0 = p[j0], j1;
            long double delta = LDBL_MAX;
            for (ll j = 1; j <= n; ++j) {
                if (!used[j]) {
                    long double cur = a[i0
                    → - 1][j - 1] - u[i0]
                    → - v[j];
                    if (cur < minv[j]){
                        minv[j] = cur;
                        way[j] = j0;
                    }
                    if (minv[j] < delta){

```

```

                        delta = minv[j];
                        j1 = j;
                    }
                }
            }
            for (ll j = 0; j <= n; ++j){
                if(used[j]){
                    u[p[j]] += delta;
                    v[j] -= delta;
                } else {
                    minv[j] -= delta;
                }
            }
            j0 = j1;
        } while (p[j0] != 0);
        do{
            ll j1 = way[j0];
            p[j0] = p[j1];
            j0 = j1;
        } while (j0);
    }
    for (ll j = 1; j <= n; ++j)
        ans[p[j] - 1] = j - 1;
}

```

3.5 OrderedSet

```

// Remember to change define int long long
→
#include "ext/pb_ds/assoc_container.hpp"
#include "ext/pb_ds/tree_policy.hpp"
using namespace __gnu_pbds;
template<class T>
using ordered_set = tree<T,
→ null_type,less<T>, rb_tree_tag,
→ tree_order_statistics_node_update> ;
template<class key, class value, class cmp
→ = std::less<key>>

```



```

using ordered_map = tree<key, value, cmp,
    ↪ rb_tree_tag,
    ↪ tree_order_statistics_node_update>;
// find_by_order(k) returns iterator to
    ↪ kth element starting from 0;
// order_of_key(k) returns count of
    ↪ elements strictly smaller than k;
template<class T>
using min_heap =
    ↪ priority_queue<T, vector<T>, greater<T>
    ↪ >;

```

3.6 Template

```

#pragma GCC optimize ("Ofast")
#pragma GCC target ("avx2")
#include<bits/stdc++.h>
using namespace std;
#define int long long
mt19937_64 rng(chrono::steady_clock::now()
    ↪ .time_since_epoch().count());
ios::sync_with_stdio(0);
cin.tie(0);

```

3.7 Basis

```

struct basis{
    array<int, 20> b;
    int sz = 0;
    basis(){
        b.fill(0);
        sz = 0;
    }
    void insert(int x){
        for(int i = 19; i >= 0; i --){
            if((x >> i) & 1){
                if(!b[i]){
                    b[i] = x;
                    sz ++;

```

```

                return;
            }
            x ^= b[i];
        }
    }
    bool query(int x){
        for(int i = 19; i >= 0; i --){
            if(x == 0) return true;
            if((x >> i) & 1){
                if(!b[i]) return false;
                x ^= b[i];
            }
        }
        return x == 0;
    }
};

```

3.8 ConvexHullTrick

```

template<class T>
using min_heap =
    ↪ priority_queue<T, vector<T>, greater<T>>;
// When lines are added in increasing
    ↪ order of slopes
// Queries minimum
struct CHT {
    struct Line {
        int m, c;
        Line () {}
        Line (int _m, int _c) : m(_m),
            ↪ c(_c) {}

        double intersect(const Line &other)
            ↪ const {
            return (double)(other.c - c) /
                ↪ (m - other.m);
        }
    }
};

```

```

    int operator()(int x) const {
        return m * x + c;
    }
};
vector<double> points;
vector<Line> lines;
void init(Line l){
    points.push_back(-inf);
    lines.push_back(l);
}
void add_line(Line l){
    while(lines.size() >= 2 &&
        ↪ l.intersect(lines[lines.size()
        ↪ - 2]) <= points.back()){
        points.pop_back();
        lines.pop_back();
    }
    if(!lines.empty()){
        points.push_back(l.intersect(
            ↪ lines.back()));
    }
    if(!lines.empty() && lines.back().m
        ↪ == l.m){
        if(lines.back().c >= l.c)
            ↪ return;
        lines.pop_back();
        points.pop_back();
    }
    lines.push_back(l);
}
int query(int x){
    int idx = upper_bound(al(points),
        ↪ x) - points.begin() - 1;
    return lines[idx](x);
}
};
void solve(){

```

```

int n, c; cin>>n>>c;
vector<int> a(n + 1), dp(n + 1);
for(int i = 1; i <= n; i++){
    cin>>a[i];
}
CHT cht;
cht.init(CHT::Line(-2 * a[1], a[1] *
    ↪ a[1]));
for(int i = 2; i <= n; i++){
    dp[i] = cht.query(a[i]) + a[i] *
    ↪ a[i] + c;
    cht.add_line(CHT::Line(-2 * a[i],
    ↪ dp[i] + a[i] * a[i]));
}
cout<<dp[n]<<endl;
}
// Anyhow works, Queries Maximum
struct Line {
    mutable int k, m, p;
    bool operator<(const Line& o) const {
    ↪ return k < o.k; }
    bool operator<(int x) const { return p
    ↪ < x; }
};
struct CHT : multiset<Line, less<>> {
    // (for doubles, use inf = 1/.0,
    ↪ div(a,b) = a/b)
    int div(int a, int 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(int k, int 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));
}
int query(int x) {
    assert(!empty());
    auto l = *lower_bound(x);
    return l.k * x + l.m;
}
};

```

4 Strings

4.1 Hash

```

void go_hsh(string s){
    hsh[0] = s[0];
    ppow[0] = 1;
    for(int i = 1; i < s.size(); i++){
        hsh[i] = (((hsh[i - 1] * p) % mod)
    ↪ + s[i]) % mod;
        ppow[i] = (ppow[i - 1] * p) % mod;
    }
}
int get(int l, int r){
    if(!l) return hsh[r];
    return (hsh[r] - (hsh[l - 1] * ppow[r -
    ↪ l + 1]) % mod + mod) % mod;
}

```

4.2 Manacher

```

// Finds all palindromic substrings in
    ↪ O(n) time
// Returns a vector where the ith element
    ↪ represents the length of the longest
    ↪ palindromic substring
// centered at index i (for odd-length
    ↪ palindromes) or between indices i and
    ↪ i+1 (for even-length palindromes)
vector<int> manacher_odd(string s){
    int n = s.size();
    s = ('$' + s + '^');
    vector<int> ans(n + 2);
    int l = 1, r = 1;
    for(int i = 1; i <= n; i++){
        ans[i] = max(OLL, min(r - i, ans[l
    ↪ + (r - i)]));
        while(s[i + ans[i]] == s[i -
    ↪ ans[i]]){
            ans[i]++;
        }
        if(i + ans[i] > r){
            l = i - ans[i];
            r = i + ans[i];
        }
    }
    vector<int> res;
    // we get answer as
    // even lengthed palindrome between i
    ↪ & i+1 as (ans[i+1]-1)/2
    // odd centered at i as ans[i]/2
    return
    ↪ vector<int>(ans.begin()+1, ans.end()-1)
}
vector<int> manacher(string s){
    string t = "";
    for(auto x : s){
        t += "#";
    }
}

```

```

    t += x;
}
t += "#";
// cout<<t<<endl;
auto res = manacher_odd(t);
return vector<int>(res.begin() + 1,
    ↪ res.end() - 1);
}

```

Manacher.h

Description: For each position in a string,
 ↪ computes $p[0][i]$ = half length
 of longest even palindrome around pos i ,
 ↪ $p[1][i]$ = longest odd (half rounded
 down).

Time: $O(N)$

e7ad79, 13 lines

```

array<vi, 2> manacher(const string& s) {
    int n = sz(s);
    array<vi, 2> p = {vi(n+1), vi(n)};
    rep(z, 0, 2) for (int i=0, l=0, r=0; i < n;
        ↪ i++) {
        int t = r-i+!z;
        if (i<r) p[z][i] = min(t, p[z][l+t]);
        int L = i-p[z][i], R = i+p[z][i]-!z;
        while (L>=1 && R+1<n && s[L-1] ==
            ↪ s[R+1])
            p[z][i]++, L--, R++;
        if (R>r) l=L, r=R;
    }
    return p;
}

```

4.3 SuffixArray

```

struct SuffixArray {
    vector<int> sa, lcp;

```

// Append 0 to s in case s is a vector

```

SuffixArray(string &s, int lim = 256){
    int n = s.size() + 1, k = 0, a, b;
    vector<int> x(n), y(n), ws(max(n,
        ↪ lim)), rank(n);
    for(int i = 0; i < n - 1; i++)
        ↪ x[i] = s[i];

```

```

    sa = lcp = y, iota(sa.begin(),
        ↪ sa.end(), 0);
    for(int j = 0, p = 0; p < n; j =
        ↪ max(1LL, j * 2), lim = p){
        p = j, iota(y.begin(), y.end(),
            ↪ n - j);
        for(int i = 0; i < n; i++) if
            ↪ (sa[i] >= j) y[p++] = sa[i]
            ↪ - j;
        fill(ws.begin(), ws.end(), 0);
        for(int i = 0; i < n; i++)
            ↪ ws[x[i]]++;
        for(int i = 1; i < lim; i++)
            ↪ ws[i] += ws[i - 1];
        for(int i = n; i --;)
            ↪ sa[--ws[x[y[i]]]] = y[i];
        swap(x, y), p = 1, x[sa[0]] =
            ↪ 0;
        for(int i = 1; i < n; i++) a =
            ↪ sa[i - 1], b = sa[i], x[b]
            ↪ =
            ↪ (y[a] == y[b] &&
            ↪ y[a + j] == y[b
            ↪ + j]) ? p - 1 :
            ↪ p++;
    }

```

```

    for(int i = 1; i < n; i++)
        ↪ rank[sa[i]] = i;
    for(int i = 0, j; i < n - 1;
        ↪ lcp[rank[i+]] = k) {
        for(k && k--, j = sa[rank[i] -
            ↪ 1]; s[i + k] == s[j + k];

```

```

        ↪ k++);
    }
};

```

4.4 Trie

const int MAXB = 30;

```

struct node{
    int cnt;
    int t[2];
    set<int> id;
    node (){
        cnt = 0;
        t[0] = t[1] = -1;
    }
};

struct btrie{
    vector<node> trie;
    btrie() {
        trie.resize(1);
    }
    int new_node(){
        trie.push_back(node());
        return (int)trie.size() - 1;
    }
    void insert(int x, int id){
        int cur = 0;
        trie[cur].cnt++;
        for(int i = MAXB; i >= 0; i --){
            int b = (x >> i) & 1;
            if(trie[cur].t[b] == -1){
                trie[cur].t[b] =
                    ↪ new_node();
            }
            cur = trie[cur].t[b];
            trie[cur].cnt++;
        }
    }
}

```

```

    trie[cur].id.insert(id);
}
void remove(int x, int id){
    int cur = 0;
    trie[cur].cnt --;
    for(int i = MAXB; i >= 0; i --){
        int b = (x >> i) & 1;
        cur = trie[cur].t[b];
        trie[cur].cnt --;
    }
    assert(trie[cur].id.count(id));
    trie[cur].id.erase(id);
}
pair<int, int> min_xor(int x){
    int cur = 0, res = 0, idx = -1;
    for(int i = MAXB; i >= 0; i --){
        int b = ((x >> i) & 1);
        if(trie[cur].t[b] != -1 &&
           ↪ trie[trie[cur].t[b]].cnt >
           ↪ 0){
            cur = trie[cur].t[b];
        }
        else{
            res |= (1LL << i);
            cur = trie[cur].t[b ^ 1];
            if(cur == -1) return {res,
                               ↪ -1};
        }
    }
    assert(!trie[cur].id.empty());
    idx = *trie[cur].id.begin();
    return {res, idx};
}
};
auto get = [&](int x){
    // I need y such that  $x \wedge y$  is
    ↪ maximum;
    // if x has 0 then find 1 then 0;
    // if x has 1 then find 0 then 1;

```

```

    int curr = 0;
    int ans = 0;
    for(int i = 29; i >= 0; i --){
        int bit = (x >> i) & 1;
        if(bit){
            if(trie[curr][0]){
                ans += (1 << i);
                curr = trie[curr][0];
            } else {
                curr = trie[curr][1];
            }
        } else {
            if(trie[curr][1]){
                ans += (1 << i);
                curr = trie[curr][1];
            } else {
                curr = trie[curr][0];
            }
        }
    }
    return ans;
};

4.5 Kmp
// p[i] = length of the longest proper
↪ prefix of s[0..i]
// which is also a suffix of s[0..i]
vector<int> prefix_function(string s) {
    int n = (int)s.length();
    vector<int> pi(n);
    for (int i = 1; i < n; i++) {
        int j = pi[i-1];
        while (j > 0 && s[i] != s[j])
            j = pi[j-1];
        if (s[i] == s[j])
            j++;
        pi[i] = j;
    }
}

```

```

    return pi;
}

```

5 Mathematics

5.1 Sieve

```

int low[MAXN];           // smallest prime
↪ factor
int phi[MAXN];           // Euler Totient
↪ Function
int mu[MAXN];            // Mobius function
vector<int> primes;
void sieve() {
    for (int i = 1; i < MAXN; i++) {
        phi[i] = i;
        mu[i] = 1;
    }
    for (int i = 2; i < MAXN; i++) {
        if (low[i] == 0) {
            ↪ // i is prime
            low[i] = i;
            primes.push_back(i);
            phi[i] = i - 1;
            mu[i] = -1;

            for (int j = i; j < MAXN; j +=
                ↪ i) {
                if (low[j] == 0) low[j] =
                    ↪ i;
                phi[j] -= phi[j] / i;
                if ((j / i) % i == 0)
                    mu[j] = 0;
                else
                    mu[j] *= -1;
            }
        }
    }
}

```

```

}
}

```

5.2 Vector

```

struct vec{
    vector<ld> x;
    vec(ll n){
        x.resize(n, 0);
    }
    vec operator+(const vec &other) const{
        vec res(x.size());
        rep(i, x.size()){
            res.x[i] = x[i] + other.x[i];
        }
        return res;
    }
    vec operator-(const vec &other) const{
        vec res(x.size());
        rep(i, x.size()) {
            res.x[i] = x[i] - other.x[i];
        }
        return res;
    }
    vec operator*(const ld &other) const{
        vec res(x.size());
        rep(i, x.size()) {
            res.x[i] = x[i] * other;
        }
        return res;
    }
    vec operator/(const ld &other) const{
        vec res(x.size());
        rep(i, x.size()) {
            res.x[i] = x[i] / other;
        }
        return res;
    }
    ld dot(const vec &other) const{

```

```

        ld ans = 0;
        rep(i, x.size()) {
            ans += x[i] * other.x[i];
        }
        return ans;
    }
    ld norm_sq() const{
        ld ans = 0;
        rep(i, x.size()) {
            ans += x[i] * x[i];
        }
        return ans;
    }
    ld norm() const { return
        ↪ sqrt(norm_sq()); }
};

ld dist(vec &p, vec &a, vec &b){
    vec ab = b - a;
    vec ap = p - a;
    ld seglensq = ab.norm_sq();
    if (seglensq < 1e-12){
        return ap.norm();
    }
    ld t = ap.dot(ab) / seglensq;
    if (t < 0) t = 0;
    if (t > 1) t = 1;
    vec q = a + (ab * t);
    return (q - p).norm();
}

```

5.3 SOSDp

```

int dpOr[MAXN], dpAnd[MAXN], A[MAXN];
for(int i = 0; i < n; i++){
    dpOr[A[i]] ++, dpAnd[A[i]] ++;
}
for(int bit = 0; bit < 20; bit++){
    for(int msk = 0; msk < MAXN; msk++){

```

```

        if((msk >> bit) & 1){
            dpOr[msk] += dpOr[msk ^ (1 <<
                ↪ bit)];
        }
        else{
            dpAnd[msk] += dpAnd[msk | (1 <<
                ↪ bit)];
        }
    }
}

```

5.4 PollardRho

```

//
↪ https://judge.yosupo.jp/problem/factorize
#include<bits/stdc++.h>
using namespace std;

using ll = long long;
namespace PollardRho {
    mt19937
        ↪ rnd(chrono::steady_clock::now().time_si
    const int P = 1e6 + 9;
    ll seq[P];
    int primes[P], spf[P];
    inline ll add_mod(ll x, ll y, ll m) {
        return (x += y) < m ? x : x - m;
    }
    inline ll mul_mod(ll x, ll y, ll m) {
        ll res = __int128(x) * y % m;
        return res;
        // ll res = x * y - (ll)((long
        ↪ double)x * y / m + 0.5) * m;
        // return res < 0 ? res + m : res;
    }
    inline ll pow_mod(ll x, ll n, ll m) {
        ll res = 1 % m;
        for (; n; n >>= 1) {

```

```

    if (n & 1) res = mul_mod(res, x, m);
    x = mul_mod(x, x, m);
}
return res;
}
// O(it * (logn)^3), it = number of
→ rounds performed
inline bool miller_rabin(ll n) {
    if (n <= 2 || (n & 1 ^ 1)) return (n ==
→ 2);
    if (n < P) return spf[n] == n;
    ll c, d, s = 0, r = n - 1;
    for (; !(r & 1); r >>= 1, s++) {}
    // each iteration is a round
    for (int i = 0; primes[i] < n &&
→ primes[i] < 32; i++) {
        c = pow_mod(primes[i], r, n);
        for (int j = 0; j < s; j++) {
            d = mul_mod(c, c, n);
            if (d == 1 && c != 1 && c != (n -
→ 1)) return false;
            c = d;
        }
        if (c != 1) return false;
    }
    return true;
}
void init() {
    int cnt = 0;
    for (int i = 2; i < P; i++) {
        if (!spf[i]) primes[cnt++] = spf[i] =
→ i;
        for (int j = 0, k; (k = i *
→ primes[j]) < P; j++) {
            spf[k] = primes[j];
            if (spf[i] == spf[k]) break;
        }
    }
}

```

```

// returns O(n^(1/4))
ll pollard_rho(ll n) {
    while (1) {
        ll x = rnd() % n, y = x, c = rnd() %
→ n, u = 1, v, t = 0;
        ll *px = seq, *py = seq;
        while (1) {
            *py++ = y = add_mod(mul_mod(y, y,
→ n), c, n);
            *py++ = y = add_mod(mul_mod(y, y,
→ n), c, n);
            if ((x = *px++) == y) break;
            v = u;
            u = mul_mod(u, abs(y - x), n);
            if (!u) return __gcd(v, n);
            if (++t == 32) {
                t = 0;
                if ((u = __gcd(u, n)) > 1 && u <
→ n) return u;
            }
        }
        if (t && (u = __gcd(u, n)) > 1 && u <
→ n) return u;
    }
}
vector<ll> factorize(ll n) {
    if (n == 1) return vector<ll>();
    if (miller_rabin(n)) return vector<ll>
→ {n};
    vector<ll> v, w;
    while (n > 1 && n < P) {
        v.push_back(spf[n]);
        n /= spf[n];
    }
    if (n >= P) {
        ll x = pollard_rho(n);
        v = factorize(x);
        w = factorize(n / x);
    }
}

```

```

        v.insert(v.end(), w.begin(),
→ w.end());
    }
    return v;
}
}
void solve(){
    ll m, n, k; cin>>m>>n>>k;
    set<ll> st;
    for(ll i = 0; i < n; i++){
        ll x; cin>>x;
        auto f = PollardRho::factorize(x);
        for(auto y : f) st.insert(y);
    }
}
PollardRho::init();

```

5.5 Gaussian Elimination

```

void solve(){
    ll p, f;
    cin >> p >> f;
    vector<vll> adj(p);
    vll deg(p, 0);
    rep(i, f){
        ll a, b;
        cin >> a >> b;
        --a; --b;
        adj[a].pb(b); adj[b].pb(a);
        deg[a]++; deg[b]++;
    }
    vector<bitset<NUM + 1>> mat(p);
    rep(v, p){
        for (auto u : adj[v])
            mat[v].set(u);
        if (deg[v] & 1)
            mat[v].set(v);
        else
            mat[v].set(p);
    }
}

```

```

}
ll row = 0;
for (ll col = 0; col < p && row < p;
    → ++col){
    ll sel = -1;
    for (ll r = row; r < p; ++r){
        if (mat[r].test(col)){
            sel = r;
            break;
        }
    }
    if (sel == -1)
        continue;
    if (sel != row)
        swap(mat[sel], mat[row]);
    for (ll r = 0; r < p; ++r){
        if (r != row &&
            → mat[r].test(col)){
            mat[r] ^= mat[row];
        }
    }
    ++row;
}
for (ll r = 0; r < p; ++r){
    bool allzero = true;
    for (ll c = 0; c < p; ++c)
        if (mat[r].test(c)){
            allzero = false;
            break;
        }
    if (allzero && mat[r].test(p)){
        cout << 'N' << endl;
        return;
    }
}
}
}

```

5.6 Crt

```

int extGcd(int a, int b, int &x, int &y){
    if(b == 0){
        x = 1; y = 0;
        return a;
    }
    int x1, y1;
    int d = extGcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return d;
}

int modinv(int a, int m){
    int x, y;
    int g = extGcd(a, m, x, y);
    int inv = (x % m + m) % m;
    return inv;
}

// solves the two congruences and returns
→ the minimum non-negative solution
int get(int a, int b, int c, int d, int
    → n){
    //  $ax = -b \bmod n \rightarrow ax - kn = -b \rightarrow$ 
    →  $gcd(a, n) \mid b$ 
    //  $cx = -d \bmod n$ 
    int g1 = gcd(a, n);
    if((-b) % g1 != 0) return -1;
    int m1 = n / g1;
    int r1 = ((-b / g1) % m1 + m1) % m1;
    int x1 = (modinv(a / g1, m1) * r1) %
        → m1;

    int g2 = gcd(c, n);
    if((-d) % g2 != 0) return -1;
    int m2 = n / g2;
    int r2 = ((-d / g2) % m2 + m2) % m2;
    int x2 = (modinv(c / g2, m2) * r2) %
        → m2;

```

```

//  $x = x1 \bmod m1$  and  $x = x2 \bmod m2$ 
int g = gcd(m1, m2);
if((x2 - x1) % g != 0) return -1;
int M = (m1 / g) * m2;
int inv = modinv(m1 / g, m2 / g);
int ans = (x1 + (((x2 - x1) / g * inv)
    → % (m2 / g)) * m1) % M;
ans = (ans + M) % M;
return ans;
}

```

5.7 Bits

```

mask &= ~(1LL << i); // Clear i-th bit
mask &= (mask - 1); // Turn off lowest set
→ bit
bool is_pow2 = (mask > 0 && (mask & (mask -
    → 1)) == 0); // Is power of two
int leading_zeros = __builtin_clzll(mask);
→ // Count leading zeros
int trailing_zeros = __builtin_ctzll(mask);
→ // Count trailing zeros
int lsb = mask & (-mask); // Isolate least
→ significant bit
mask &= (mask - 1); // Turn off least
→ significant bit
// Iterate over all subsets of a bitmask
for(int sub = mask; sub; sub = (sub - 1) &
    → mask){
    // sub is a non-empty subset
}

```

5.8 Euler_Totient

```

int phi(int n) { //  $O(\sqrt{n})$ 
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {

```

```

        while (n % i == 0)
            n /= i;
        result -= result / i;
    }
}
if (n > 1)
    result -= result / n;
return result;
}

void phi_1_to_n(int n) { //  $O(n \log \log n)$ 
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;
    for (int i = 2; i <= n; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= n; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}

```

5.9 NCr_Shrey

```

const ll MAXN = 1e7;
const ll MOD = 1e9 + 7;
vector<ll> fac(MAXN + 1);
vector<ll> inv(MAXN + 1);
/** Computes  $x^y$  modulo  $p$  in  $O(\log p)$  time. */
ll exp(ll x, ll y, ll p){
    ll res = 1;
    x %= p;
    while (y)
    {
        if (y & 1)
        {
            res *= x;
            res %= p;
        }
    }
}

```

```

        x *= x;
        x %= p;
        y >>= 1;
    }
    return res;
}

void factorial() {
    fac[0] = 1;
    for (ll i = 1; i <= MAXN; i++) {
        fac[i] = ( fac[i - 1] * i ) % MOD;
    }
}

void inverses() {
    inv[MAXN] = exp(fac[MAXN], MOD - 2,
        ↪ MOD);
    for (ll i = MAXN; i >= 1; i--) {
        inv[i - 1] = (inv[i] * i) % MOD;
    }
}

ll choose(int n, int r) {
    return fac[n] * inv[r] % MOD * inv[n -
        ↪ r] % MOD;
}

```

5.10 Matrix

```

template<typename T>
struct Matrix{
    vector<vector<T>> mat;
    int n, m;
    Matrix() {}
    Matrix(int _n, int _m, bool ident =
        ↪ false) {
        n = _n; m = _m;
        mat.assign(n, vector<T>(m, 0));
        if (ident) {
            for (int i = 0; i < n; i++)
                mat[i][i] = 1;
        }
    }
}

```

```

}
Matrix(const vector<vector<T>> &v) {
    n = v.size();
    m = v[0].size();
    mat = v;
}

Matrix operator*(const Matrix &b)
    ↪ const {
    Matrix res(n, b.m);
    for (int i = 0; i < n; i++) {
        for (int k = 0; k < m; k++) {
            if (mat[i][k].val == 0)
                ↪ continue;
            for (int j = 0; j < b.m;
                ↪ j++) {
                res.mat[i][j] +=
                    ↪ mat[i][k] *
                    ↪ b.mat[k][j];
            }
        }
    }
    return res;
}

};

template<typename T>
Matrix<T> mat_pow(Matrix<T> a, long long p)
    ↪ {
    Matrix<T> res(a.n, a.n, true);
    while (p) {
        if (p & 1) res = res * a;
        a = a * a;
        p >>= 1;
    }
    return res;
}

```