DU_Rumbling (University of Dhaka)

Formulas

- $\sum_{k=0}^{n} \binom{n-k}{k} = \text{Fib}_{n+1}$
- $\binom{n}{k} = \binom{n}{n-k}$
- $\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$
- $k\binom{n}{k} = n\binom{n-1}{k-1}$
- $\binom{n}{k} = \frac{n}{k}\binom{n-1}{k-1}$
- $\sum_{i=0}^{n} \binom{n}{i} = 2^n$
- $\sum_{i=0}^{\lfloor n/2 \rfloor} \binom{n}{2i} = 2^{n-1}$
- $\sum_{i=0}^{\lfloor (n-1)/2 \rfloor} \binom{n}{2i+1} = 2^{n-1}$
- $\sum_{i=0}^{k} (-1)^i \binom{n}{i} = (-1)^k \binom{n-1}{k}$
- $\sum_{i=0}^{k} \binom{n+i}{i} = \binom{n+k+1}{k}$
- $\sum_{i=1}^{n} i\binom{n}{i} = n \cdot 2^{n-1}$
- $\sum_{i=1}^{n} i^2\binom{n}{i} = (n+n^2) \cdot 2^{n-2}$
- Vandermonde's Identity: $\sum_{k=0}^{r} \binom{m}{k}\binom{n}{r-k} = \binom{m+n}{r}$
- Hockey-Stick Identity: $\sum_{i=r}^{n} \binom{i}{r} = \binom{n+1}{r+1}$
- $\sum_{i=0}^{k} \binom{k}{i}^2 = \binom{2k}{k}$
- $\sum_{k=0}^{n} \binom{n}{k}\binom{n}{n-k} = \binom{2n}{n}$
- $\sum_{k=q}^{n} \binom{n}{k}\binom{k}{q} = 2^{n-q}\binom{n}{q}$
- $\sum_{i=0}^{n} k^i \binom{n}{i} = (k+1)^n$
- $\sum_{i=0}^{n} \binom{2n}{i} = 2^{2n-1} + \frac{1}{2}\binom{2n}{n}$
- $\sum_{i=1}^{n} \binom{n}{i}\binom{n-1}{i-1} = \binom{2n-1}{n-1}$
- $\sum_{i=0}^{n} \binom{2n}{i}^2 = \frac{1}{2}\left(\binom{4n}{2n} + \binom{2n}{n}^2\right)$
- Catalan Number: $C_n = \frac{1}{n+1}\binom{2n}{n}$
- Derangement: $d(n) = (n-1) \cdot (d(n-1) + d(n-2))$, with $d(0) = 1$, $d(1) = 0$
- Stirling Numbers of the First Kind: $s(n,k) = s(n-1,k-1) + (n-1) \cdot s(n-1,k)$
- Stirling Numbers of the Second Kind: $S(n,k) = S(n-1,k-1) + k \cdot S(n-1,k)$
- Bell Numbers: $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k$
- $\sum_{i=0}^{n} i \cdot i! = (n+1)! - 1$
- $\sum_{i=1}^{n} ia^i = \frac{a(na^{n+1} - (n+1)a^n + 1)}{(a-1)^2}$
- Fibonacci: $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$
- $F_n = \sum_{k=0}^{\lfloor (n-1)/2 \rfloor} \binom{n-k-1}{k}$
- $F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$
- $\sum_{i=1}^{n} F_i = F_{n+2} - 1$
- $\sum_{i=0}^{n-1} F_{2i+1} = F_{2n}$
- $\sum_{i=1}^{n} F_{2i} = F_{2n+1} - 1$
- **Pick's Theorem:** For a simple polygon with vertices on lattice points, $A = I + \frac{B}{2} - 1$ where A is area, I is number of interior lattice points, and B is number of boundary lattice points.

Bitwise Tricks and Identities

- $a + b = a \oplus b + 2 \cdot (a \& b)$
- $a + b = a \mid b + a \& b$
- $a \oplus b = a \mid b - a \& b$
- $n \bmod 2^i = n \& (2^i - 1)$

Bit Checking (Useful in Subset/Pair Sums)

- The k-th bit is set in x iff:

$$x \bmod (2^{k+1} - 1) \geq 2^k$$

- Equivalently, k-th bit is set iff:

$$x \bmod (2^{k+1} - 1) - x \bmod 2^k \neq 0$$

(The result will be exactly 2^k if set.)

XOR Properties

- $1 \oplus 2 \oplus 3 \oplus \cdots \oplus (4k-1) = 0$ for any $k \geq 0$

Erdős–Gallai Theorem (Graph Degree Sequence)

Let $d_1 \geq d_2 \geq \cdots \geq d_n$ be a sequence of non-negative integers.
This sequence is the degree sequence of a finite simple undirected graph (no loops or multiple edges) **iff**:

- $\sum_{i=1}^{n} d_i$ is even
- and for every k where $1 \leq k \leq n$,

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(d_i, k)$$

Mod Inverse

```
ll inv(ll a, ll m){
    if(a<=1) return a;
    return m - ((ll)(m/a)*inv(m%a,m)%m);
}
```

Floor Nth Root

```
ll Floor_nth_Root(ll x, ll n){
    if (x==0 || x==1){
        return x;
    }
    auto power=[](ll a, ll exp){
        ll res=1;
        for(ll i=0;i<exp;i++) {
            res*=a;
        }
        return res;
    };
    ll v=power(2,63/n),lo=0,hi=min<ll>(v,ll(x+1));
    while (hi-lo>1){
        ll mid=(lo+hi)/2,count=n,val=1;
        while(count--)val*=mid;
        if(val<=x) lo=mid;
        else hi=mid;
    }
    return lo;
}
```

nCr Mod

```
// call FACTMOD() in int main()
vll factmod(N,1ll);

void FACTMOD(){
    loop(i,2,N){
        factmod[i]=(factmod[i-1]*i)%M;
    }
}

ll binpow(ll a, ll n, ll m){
    if(n==0) return 1;
    ll x=binpow(a,n/2,m);
    if(n&1){
        return (a*((x*x)%m))%m;
    }else{
        return (x*x)%m;
    }
}

ll nCr(ll n, ll r){
    ll nfact=factmod[n];
    ll rfact=binpow(factmod[r],M-2,M);
    ll n_rfact=binpow(factmod[n-r],M-2,M);
    ll ans=(nfact*rfact)%M;
    ans=(ans*n_rfact)%M;
    return ans;
}
```

LCA and Binary Lifting

```
const int N = 3e5 + 9, LG = 18;

vector<int> g[N];
int par[N][LG + 1], dep[N], sz[N];
void dfs(int u, int p = 0) {
  par[u][0] = p;
  dep[u] = dep[p] + 1;
  sz[u] = 1;
  for (int i = 1; i <= LG; i++) par[u][i] =
      par[par[u][i - 1]][i - 1];
  for (auto v: g[u]) if (v != p) {
    dfs(v, u);
    sz[u] += sz[v];
  }
}
int lca(int u, int v) {
  if (dep[u] < dep[v]) swap(u, v);
  for (int k = LG; k >= 0; k--) if (dep[par[u][k]] >=
      dep[v]) u = par[u][k];
  if (u == v) return u;
  for (int k = LG; k >= 0; k--) if (par[u][k] !=
      par[v][k]) u = par[u][k], v = par[v][k];
  return par[u][0];
}
int kth(int u, int k) {
  assert(k >= 0);
  for (int i = 0; i <= LG; i++) if (k & (1 << i)) u =
      par[u][i];
  return u;
}
int dist(int u, int v) {
  int l = lca(u, v);
  return dep[u] + dep[v] - (dep[l] << 1);
}
//kth node from u to v, 0th node is u
int go(int u, int v, int k) {
  int l = lca(u, v);
  int d = dep[u] + dep[v] - (dep[l] << 1);
  assert(k <= d);
  if (dep[l] + k <= dep[u]) return kth(u, k);
  k -= dep[u] - dep[l];
  return kth(v, dep[v] - dep[l] - k);
}
```

Articulation Bridge

```
// br = bridges, p = parent
vector<pair<int, int>> br;
int dfsBR(int u, int p) {
  low[u] = disc[u] = ++Time;
  for (int& v : adj[u]) {
    if (v == p) continue; // we don't want to go back
        through the same path.
                          // if we go back is because
                              we found another way
                              back
    if (!disc[v]) { // if V has not been discovered
        before
      dfsBR(v, u);  // recursive DFS call
      if (disc[u] < low[v]) // condition to find a
          bridge
        br.push_back({u, v});
      low[u] = min(low[u], low[v]); // low[v] might
          be an ancestor of u
    } else // if v was already discovered means that
        we found an ancestor
      low[u] = min(low[u], disc[v]); // finds the
          ancestor with the least discovery time
  }
}
void BR() {
  low = disc = vector<int>(adj.size());
  Time = 0;
  for (int u = 0; u < adj.size(); u++)
    if (!disc[u])
      dfsBR(u, u);
}
```

Articulation Point

```
// adj[u] = adjacent nodes of u
// ap = AP = articulation points
// p = parent
// disc[u] = discovery time of u
// low[u] = 'low' node of u
int dfsAP(int u, int p) {
  int children = 0;
  low[u] = disc[u] = ++Time;
  for (int& v : adj[u]) {
    if (v == p) continue; // we don't want to go back
        through the same path.
                          // if we go back is because
                              we found another way
                              back
    if (!disc[v]) { // if V has not been discovered
        before
      children++;
      dfsAP(v, u); // recursive DFS call
      if (disc[u] <= low[v]) // condition #1
        ap[u] = 1;
      low[u] = min(low[u], low[v]); // low[v] might
          be an ancestor of u
```

```
    } else // if v was already discovered means that
        we found an ancestor
      low[u] = min(low[u], disc[v]); // finds the
          ancestor with the least discovery time
  }
  return children;
}
void AP() {
  ap = low = disc = vector<int>(adj.size());
  Time = 0;
  for (int u = 0; u < adj.size(); u++)
    if (!disc[u])
      ap[u] = dfsAP(u, u) > 1; // condition #2
}
```

Segment Tree

```
template <class S, S (*op)(S, S), S (*e)()> struct
    segtree {
  public:
    int ceil_pow2(int n) {
        int x = 0;
        while ((1U << x) < (unsigned int)(n)) x++;
        return x;
    }
    segtree() : segtree(0) {}
    segtree(int n) : segtree(std::vector<S>(n, e()))
        {}
    segtree(const std::vector<S>& v) :
        _n(int(v.size())) {
        log = ceil_pow2(_n);
        size = 1 << log;
        d = std::vector<S>(2 * size, e());
        for (int i = 0; i < _n; i++) d[size + i] =
            v[i];
        for (int i = size - 1; i >= 1; i--) {
            update(i);
        }
    }
    void set(int p, S x) { // set pth value to x
        assert(0 <= p && p < _n);
        p += size;
        d[p] = x;
        for (int i = 1; i <= log; i++) update(p >> i);
    }
    S get(int p) { // get pth value of array
        assert(0 <= p && p < _n);
        return d[p + size];
    }
    S prod(int l, int r) { // find the result for [l,
        r)
        assert(0 <= l && l <= r && r <= _n);
        S sml = e(), smr = e();
        l += size;
        r += size;
        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);
    }
    S all_prod() { return d[1]; }
    template <bool (*f)(S)> int max_right(int l) {
//max_right(l, f): Finds the maximum right boundary
    where a predicate f holds.
        return max_right(l, [](S x) { return f(x); });
    }
    template <class F> int max_right(int l, F f) {
        assert(0 <= l && l <= _n);
        assert(f(e()));
        if (l == _n) return _n;
        l += size;
        S sm = e();
        do {
            while (l % 2 == 0) l >>= 1;
            if (!f(op(sm, d[l]))) {
                while (l < size) {
                    l = (2 * l);
                    if (f(op(sm, d[l]))) {
                        sm = op(sm, d[l]);
                        l++;
                    }
                }
                return l - size;
            }
            sm = op(sm, d[l]);
            l++;
        } while ((l & -l) != l);
        return _n;
    }
    template <bool (*f)(S)> int min_left(int r) {
        return min_left(r, [](S x) { return f(x); });
    }
    template <class F> int min_left(int r, F f) {
        assert(0 <= r && r <= _n);
        assert(f(e()));
        if (r == 0) return 0;
        r += size;
        S sm = e();
        do {
            r--;
            while (r > 1 && (r % 2)) r >>= 1;
            if (!f(op(d[r], sm))) {
                while (r < size) {
                    r = (2 * r + 1);
                    if (f(op(d[r], sm))) {
                        sm = op(d[r], sm);
                        r--;
                    }
                }
                return r + 1 - size;
            }
            sm = op(d[r], sm);
        } while ((r & -r) != r);
        return 0;
    }
  private:
    int _n, size, log;
    std::vector<S> d
    void update(int k) { d[k] = op(d[2 * k], d[2 * k
        + 1]); }
};
using S = pair<int, int>; // data type
S op (S a, S b) { // operation type
    return min(a, b);
}
S e() { // primary value
    return make_pair(MOD, MOD);
}
```

FFT Polynomial Multiplication

```
const ll N = 3e5 + 9; const double PI = acos(-1);
struct base {
    double a, b;
    base(double a = 0, double b = 0) : a(a), b(b) {}
    const base operator+(const base &c) const {
        return base(a + c.a, b + c.b); }
    const base operator-(const base &c) const {
        return base(a - c.a, b - c.b); }
    const base operator*(const base &c) const {
        return base(a * c.a - b * c.b, a * c.b + b *
        c.a); }
};
void fft(vector<base> &p, bool inv = 0) {
    ll n = p.size(), i = 0;
    for (ll j = 1; j < n - 1; ++j) {
        for (ll k = n >> 1; k > (i ^= k); k >>= 1);
        if (j < i) swap(p[i], p[j]);
    }
    for (ll l = 1, m; (m = l << 1) <= n; l <<= 1) {
        double ang = 2 * PI / m;
        base wn = base(cos(ang), (inv ? 1. : -1.) *
            sin(ang)), w;
        for (ll i = 0; i < n; i += m)
            for (w = base(1, 0), ll j = i, k = i + l;
                j < k; ++j, w = w * wn) {
                base t = w * p[j + l];
                p[j + l] = p[j] - t; p[j] = p[j] + t;
            }
    }
    if (inv) for (ll i = 0; i < n; ++i) p[i].a /= n,
        p[i].b /= n;
}
vector<long long> multiply(vector<ll> &a, vector<ll>
    &b) {
    ll n = a.size(), m = b.size(), t = n + m - 1, sz
        = 1;
    while (sz < t) sz <<= 1;
    vector<base> x(sz), y(sz), z(sz);
    for (ll i = 0; i < sz; ++i) {
        x[i] = i < (ll)a.size() ? base(a[i], 0) :
            base(0, 0);
        y[i] = i < (ll)b.size() ? base(b[i], 0) :
            base(0, 0);
    }
    fft(x), fft(y);
    for (ll i = 0; i < sz; ++i) z[i] = x[i] * y[i];
    fft(z, 1);
    vector<long long> ret(sz);
    for (ll i = 0; i < sz; ++i) ret[i] = (long
        long)round(z[i].a);
```

```
    return ret;
}
```

Gaussian Elimination

```
// formation of the equation vector:
// [ 1  2  1 | 6 ]  -> Represents x + 2y + z = 6
// [ 2  3  1 | 9 ]  -> Represents 2x + 3y + z = 9
// [ 3  1  2 | 8 ]  -> Represents 3x + y + 2z = 8
// Time Complexity : O(min(n,m)*n*m)

const double EPS = 1e-9;
const int INF = 2;

int gauss (vector < vector<double> > a,
    vector<double> & ans) {
    int n = (int) a.size();
    int m = (int) a[0].size() - 1;

    vector<int> where (m, -1);
    for (int col=0, row=0; col<m && row<n; ++col) {
        int sel = row;
        for (int i=row; i<n; ++i)
            if (abs (a[i][col]) > abs (a[sel][col]))
                sel = i;
        if (abs (a[sel][col]) < EPS)
            continue;
        for (int i=col; i<=m; ++i)
            swap (a[sel][i], a[row][i]);
        where[col] = row;

        for (int i=0; i<n; ++i)
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j=col; j<=m; ++j)
                    a[i][j] -= a[row][j] * c;
            }
        ++row;
    }

    ans.assign (m, 0);
    for (int i=0; i<m; ++i)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i=0; i<n; ++i) {
        double sum = 0;
        for (int j=0; j<m; ++j)
            sum += ans[j] * a[i][j];
        if (abs (sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i=0; i<m; ++i)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

BIT

```
template <class T> struct BIT{
    // A[] has to be 1-indexed
    // create BIT of n elements : BIT<ll> bit(n);
    // MODE A (point-update + prefix/range-sum):
    //   add v to A[idx]:     bit.upd(idx, v);
    //   sum of A[1..idx]:    ll sum1 =
        bit.query(idx);
    //   sum of A[l..r]:      ll sumLR = bit.query(l,
        r);
    // MODE B (range-update + point-query):
    //   add v to every A[i] for i in [l..r]:
        bit.upd(l, r, v);
    //   current A[idx]:      ll Ai = bit.query(idx);
    // Note: Do NOT mix MODE A and MODE B on the same
        instance
    // use separate BITs if you need both.
    int n;
    vector<T> t;
    BIT() : n(0) {}
    BIT(int _n) {
        n = _n;
        t.assign(n + 1, 0);
    }

    static int lowbit(int x) {
        return x & -x;
    }

    T query(int i) const {
        T ans = 0;
        for (; i > 0; i -= lowbit(i)) ans += t[i];
        return ans;
    }

    void upd(int i, T val) {
        if (i <= 0) return;
        for (; i <= n; i += lowbit(i)) t[i] += val;
    }

    T query(int l, int r) const {
        if (l > r) return 0;
        return query(r) - query(l - 1);
    }

    void upd(int l, int r, T val) {
        upd(l, val); upd(r + 1, -val);
    }
};
```

pbds

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update> pbds; //
    find_by_order, order_of_key

int main() {
  pbds A;
  A.insert(1);
  for (auto i : A) cout << i << " ";
  // finding kth element - 4th query
  cout << "1st element: " << *A.find_by_order(1) <<
      endl;
  // finding number of elements smaller than X - 3rd
      query
  cout << "No. of elems smaller than 6: " <<
      A.order_of_key(6) << endl; // 2
  // lower bound -> Lower Bound of X = first element
      >= X in the set
  cout << "Lower Bound of 6: " << *A.lower_bound(6)
      << endl;
  // Upper bound -> Upper Bound of X = first element
      > X in the set
  cout << "Upper Bound of 6: " << *A.upper_bound(6)
      << endl;
  // // Remove elements - 2nd query
  A.erase(1);
  A.erase(11); // element that is not present is not
      affected
  // A contains
  cout << "A = ";
}
```

Merge Sort Tree

```
const ll MAXN=3e4+10;
vector<ll> tree[4*MAXN];

void build(ll a[], ll node, ll search_l, ll search_r){
    // root node = 1
    // search_l and search_r is 0-indexed
    if(search_l>=search_r) {
        tree[node]=vector<ll>(1,a[search_l]);
    }else{
        ll mid=(search_l+search_r)/2;
        build(a, node*2, search_l, mid);
        build(a, node*2+1, mid+1, search_r);
        merge(tree[node*2].begin(),
            tree[node*2].end(),
        tree[node*2+1].begin(), tree[node*2+1].end(),
            back_inserter(tree[node]));
    }
}

ll VectorsValue(vector<ll>& v, ll k){
    // Here goes the algorithm for the value you want
        to extract
    ll n=v.size();
    auto it=upper_bound(v.begin(), v.end(), k);
    if(it==v.end()){
        return 0;
    }else{
        ll x=it-v.begin();
        return n-x;
    }
}
```

```
ll query(ll node, ll search_l, ll search_r, ll l, ll
    r, ll val) {
    // root node = 1
    // initial search_l = 0, search_r = 0, l= 0
        indexed left pointer (inclusive)
    // l = 0 indexed left pointer (inclusive)
    // r = 0 indexed right pointer (inclusive)
    if (l>r) return 0;
    if (l==search_l && r==search_r) {
        return VectorsValue(tree[node],val);
    }
    ll mid=(search_l+search_r)/2;
    return
        query(node*2,search_l,mid,l,min(r,mid),val)+
    query(node*2+1,mid+1,search_r,max(l,mid+1),r,val);
}
```

Euler Tour DFS

```
vector<ll> t_in(N,0ll);
vector<ll> t_out(N,0ll);
vector<ll> val(N,0ll);
vector<ll> tree[N];
vector<ll> flattened(N,-1);
vector<ll> pos(N,-1);
vector<ll> values(N,-1);
ll timer=0;
ll n;

void dfs(ll node, ll parent){
    t_in[node]=timer;
    flattened[timer]=node;
    timer++;
    for(auto child:tree[node]){
        if(child==parent) continue;
        dfs(child,node);
    }
    t_out[node]=timer-1;
}

void solve(){
    dfs(1,-1);
    loop(i,0,n){
        pos[flattened[i]]=i;
    }
    loop(node,1,n+1){
        values[pos[node]]=val[node];
    }
}
```

Range length Path CD

```
vector<int> tree[N];
static int  subtree_size[N+1];
static bool processed_centroid[N+1];

vector<int> depths,clearlist;
BIT<int> bit;
ll totalpaths=0;
int maxdepth;
int n,k1,k2;

int get_subtree_size(int node, int parent){
    subtree_size[node]=1;
    for(int child:tree[node]){
        if(!processed_centroid[child] &&
            child!=parent){
        subtree_size[node]+=
        get_subtree_size(child,node);
        }
    }
    return subtree_size[node];
}

int find_centroid(int node, int parent, int
    compoSize){
    for(int child:tree[node]){
        if(!processed_centroid[child] &&
            child!=parent &&
            subtree_size[child]>compoSize/2){
            return
                find_centroid(child,node,compoSize);
        }
    }
    return node;
}

void dfs_count(ll node, ll parent, ll depth){
    if(depth>k) return;
    totalpaths += freq[k-depth];
    maxdepth=max(maxdepth,depth);
    for(auto child:tree[node]){
        if(!processed_centroid[child] &&
            child!=parent){
            dfs_count(child,node,depth+1);
        }
    }
}

void dfs_fill(ll node, ll parent, ll depth){
    if(depth>k) return;
    freq[depth]++;
    for(auto child:tree[node]){
        if(!processed_centroid[child] &&
            child!=parent){
            dfs_fill(child,node,depth+1);
        }
    }
}

void collectDepths(int node,int parent, int depth){
    if(depth>k2) return;
    depths.push_back(depth);
    for(int child:tree[node]){
        if(!processed_centroid[child] &&
            child!=parent){
            collectDepths(child,node,depth+1);
        }
    }
}
```

```
void decompose(int node){
    int component_size=get_subtree_size(node,-1);
    int
        centroid=find_centroid(node,-1,component_size);
    clearlist.clear();
    clearlist.push_back(1);
    processed_centroid[centroid]=true;
    bit.upd(1,1);
    for(int child:tree[centroid]){
        if(processed_centroid[child]) continue;
        depths.clear();
        collectDepths(child,centroid,1);
        for(int d:depths){
            int lo=k1-d;
            int hi=k2-d;
            if(hi<0 || lo>k2) continue;
            lo=max<int>(lo, 0);
            hi=min(hi, k2);
            totalpaths += bit.query(lo+1, hi+1);
            //bit is 1 indexed
        }
        for(int d:depths){
            if(d>k2) continue;
            int idx=d+1;
            bit.upd(idx,1);
            clearlist.push_back(idx);
        }
    }
    for(int idx: clearlist) bit.upd(idx, -1);
    for(int child:tree[centroid]){
        if(!processed_centroid[child])
            decompose(child);
    }
}
```

Lazy Segtree

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

class LazySeg {
    struct Node {
        ll sum, sumSq;    // sum(a_i), sum(a_i^2)
            over the segment
        ll add;           // pending range-add
        ll setVal;        // pending range-assign
        bool toSet;       // flag for pending assign
        Node(): sum(0), sumSq(0), add(0), setVal(0),
            toSet(false) {}
    };
    int n;
    vector<Node> st;

    // Build from a[1..n]
    void build(int v, int tl, int tr, const
        vector<ll>& a) {
        if (tl == tr) {
            st[v].sum   = a[tl];
            st[v].sumSq = a[tl]*a[tl];
```

```
    } else {
        int tm = (tl+tr)>>1;
        build(v<<1,   tl,   tm, a);
        build(v<<1|1, tm+1, tr, a);
        pull(v);
    }
}

// Push lazy tags to children
void push(int v, int tl, int tr) {
    Node &nd = st[v];
    if (nd.toSet) {
        int tm = (tl+tr)>>1;
        applySet(v<<1,   tl,       tm,
            nd.setVal);
        applySet(v<<1|1, tm+1,      tr,
            nd.setVal);
        nd.toSet = false;
    }
    if (nd.add != 0) {
        int tm = (tl+tr)>>1;
        applyAdd(v<<1,   tl,       tm,   nd.add);
        applyAdd(v<<1|1, tm+1,      tr,   nd.add);
        nd.add = 0;
    }
}

// Recompute this node from children
void pull(int v) {
    st[v].sum   = st[v<<1].sum   + st[v<<1|1].sum;
    st[v].sumSq = st[v<<1].sumSq +
        st[v<<1|1].sumSq;
}

// Apply "set all in [tl..tr] = x"
void applySet(int v, int tl, int tr, ll x) {
    ll len = tr - tl + 1;
    st[v].sum    = x*len;
    st[v].sumSq  = x*x*len;
    st[v].setVal = x;
    st[v].toSet  = true;
    st[v].add    = 0;
}

// Apply "add x to all in [tl..tr]"
void applyAdd(int v, int tl, int tr, ll x) {
    ll len = tr - tl + 1;
    st[v].sumSq += 2*x*st[v].sum + x*x*len;  //
        update squares
    st[v].sum   += x*len;
    st[v].add   += x;
}

// Internal range-assign
void updateSet(int v, int tl, int tr, int l, int
    r, ll x) {
    if (r < tl || tr < l) return;
    if (l <= tl && tr <= r) {
        applySet(v, tl, tr, x);
    } else {
        push(v, tl, tr);
        int tm = (tl+tr)>>1;
        updateSet(v<<1,   tl,   tm, l, r, x);
        updateSet(v<<1|1, tm+1, tr, l, r, x);
        pull(v);
    }
}

// Internal range-add
void updateAdd(int v, int tl, int tr, int l, int
    r, ll x) {
    if (r < tl || tr < l) return;
    if (l <= tl && tr <= r) {
        applyAdd(v, tl, tr, x);
    } else {
        push(v, tl, tr);
        int tm = (tl+tr)>>1;
        updateAdd(v<<1,   tl,   tm, l, r, x);
        updateAdd(v<<1|1, tm+1, tr, l, r, x);
        pull(v);
    }
}

// Internal sum(a_i) query
ll querySum(int v, int tl, int tr, int l, int r) {
    if (r < tl || tr < l) return 0;
    if (l <= tl && tr <= r) {
        return st[v].sum;
    }
    push(v, tl, tr);
    int tm = (tl+tr)>>1;
    return querySum(v<<1,   tl,   tm, l, r)
         + querySum(v<<1|1, tm+1, tr, l, r);
}

// Internal sum(a_i^2) query
ll querySq(int v, int tl, int tr, int l, int r) {
    if (r < tl || tr < l) return 0;
    if (l <= tl && tr <= r) {
        return st[v].sumSq;
    }
    push(v, tl, tr);
    int tm = (tl+tr)>>1;
    return querySq(v<<1,   tl,   tm, l, r)
         + querySq(v<<1|1, tm+1, tr, l, r);
}

public:
    LazySeg(int _n): n(_n), st(4*n+4) {}

    // Build from 1-indexed array a
    void build(const vector<ll>& a) {
        build(1, 1, n, a);
    }

    // set a[i]=x for all i element of [l,r]
    void rangeSet(int l, int r, ll x) {
        updateSet(1, 1, n, l, r, x);
    }

    // add x to a[i] for all i element of [l,r]
    void rangeAdd(int l, int r, ll x) {
        updateAdd(1, 1, n, l, r, x);
    }

    // return sum(a[i], i=l...r)
    ll rangeSum(int l, int r) {
        return querySum(1, 1, n, l, r);
    }

    // return sum(a[i]^2, i=l...r)
    ll rangeSumSq(int l, int r) {
        return querySq(1, 1, n, l, r);
    }
};
int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int T;
    cin >> T;
    for(int tc = 1; tc <= T; tc++){
        int N, Q;
        cin >> N >> Q;
        vector<ll> a(N+1);
        for(int i = 1; i <= N; i++)
            cin >> a[i];

        LazySeg st(N);
        st.build(a);

        cout << "Case " << tc << ":\n";
        while(Q--){
            int type, l, r; ll x;
            cin >> type >> l >> r;
            if (type == 0) {
                cin >> x;
                st.rangeSet(l, r, x);
            }
            else if (type == 1) {
                cin >> x;
                st.rangeAdd(l, r, x);
            }
            else if (type == 2) {
                cout << st.rangeSumSq(l, r) << "\n";
            }
            else if (type == 3) {
                cout << st.rangeSum(l, r) << "\n";
            }
        }
    }
    return 0;
}
```

SCC

```
#include<bits/stdc++.h>
using namespace std;
const int N = 3e5 + 9;

// given a directed graph return the minimum number
    of edges to be added so that the whole graph
    become an SCC
bool vis[N];
vector<int> g[N], r[N], G[N], vec; //G is the
```

```
    condensed graph
void dfs1(int u) {
  vis[u] = 1;
  for(auto v: g[u]) if(!vis[v]) dfs1(v);
  vec.push_back(u);
}

vector<int> comp;
void dfs2(int u) {
  comp.push_back(u);
  vis[u] = 1;
  for(auto v: r[u]) if(!vis[v]) dfs2(v);
}

int idx[N], in[N], out[N];
int main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  int n, m;
  cin >> n >> m;
  for(int i = 1; i <= m; i++) {
    int u, v;
    cin >> u >> v;
    g[u].push_back(v);
    r[v].push_back(u);
  }
  for(int i = 1; i <= n; i++) if(!vis[i]) dfs1(i);
  reverse(vec.begin(), vec.end());
  memset(vis, 0, sizeof vis);
  int scc = 0;
  for(auto u: vec) {
    if(!vis[u]) {
      comp.clear();
      dfs2(u);
      scc++;
      for(auto x: comp) idx[x]=scc;
    }
  }
  for(int u = 1; u <= n; u++) {
    for(auto v: g[u]) {
      if(idx[u] != idx[v]) {
        in[idx[v]]++, out[idx[u]]++;
        G[idx[u]].push_back(idx[v]);
      }
    }
  }
  int needed_in=0, needed_out=0;
  for(int i = 1; i <= scc; i++) {
    if(!in[i]) needed_in++;
    if(!out[i]) needed_out++;
  }
  int ans = max(needed_in, needed_out);
  if(scc == 1) ans = 0;
  cout << ans << '\n';
  return 0;
}
```

Bitset Operations

```
#include <iostream>
#include <bitset>
using namespace std;

int main() {
    // Declare a bitset of size 8 with an initial
        binary value
    bitset<8> b1(string("10101100"));  // binary:
        10101100
    bitset<8> b2(string("00111100"));  // binary:
        00111100
    cout << "Initial bitsets:" << endl;
    cout << "b1 = " << b1 << endl;  // Output:
        10101100
    cout << "b2 = " << b2 << endl;  // Output:
        00111100
    // Bitwise AND
    cout << "\nb1 & b2 = " << (b1 & b2) << endl;  //
        Output: 00101100
    // Bitwise OR
    cout << "b1 | b2 = " << (b1 | b2) << endl;    //
        Output: 10111100
    // Bitwise XOR
    cout << "b1 ^ b2 = " << (b1 ^ b2) << endl;    //
        Output: 10010000
    // Bitwise NOT
    cout << "~b1 = " << (~b1) << endl;            //
        Output: 01010011
    // Set all bits
    bitset<8> b3 = b1;
    b3.set();
    cout << "\nb3.set() = " << b3 << endl;        //
        Output: 11111111
    // Reset all bits
    b3.reset();
    cout << "b3.reset() = " << b3 << endl;        //
        Output: 00000000
    // Flip all bits
    b3 = b1;
    b3.flip();
    cout << "b3.flip() = " << b3 << endl;         //
        Output: 01010011
    // Set a specific bit (index 2)
    b3.set(2);
    cout << "b3.set(2) = " << b3 << endl;         //
        Output: 01010111
    // Reset a specific bit (index 2)
    b3.reset(2);
    cout << "b3.reset(2) = " << b3 << endl;       //
        Output: 01010011
    // Flip a specific bit (index 0)
    b3.flip(0);
    cout << "b3.flip(0) = " << b3 << endl;        //
        Output: 01010010
    // Count number of set bits
    cout << "\nb1.count() = " << b1.count() << endl;
        // Output: 4
    // Check if any bit is set
    cout << "b1.any() = " << b1.any() << endl;
        // Output: 1 (true)
    // Check if none of the bits are set
    cout << "b3.none() = " << b3.none() << endl;
        // Output: 0 (false)
    // Check if all bits are set
    cout << "b3.all() = " << b3.all() << endl;
        // Output: 0 (false)
    // Get value of a specific bit (index 3)
    cout << "\nb1[3] = " << b1[3] << endl;
        // Output: 1
    // Convert bitset to unsigned long
    cout << "b1.to_ulong() = " << b1.to_ulong() <<
        endl;  // Output: 172
    // Convert bitset to unsigned long long
    cout << "b1.to_ullong() = " << b1.to_ullong() <<
        endl; // Output: 172
    // Convert bitset to string
    cout << "b1.to_string() = " << b1.to_string() <<
        endl; // Output: 10101100
}
```

Custom Comparator for Priority Queue

```
auto cmp = [](const pair<int, int>& a, const
    pair<int, int>& b) {
    return a.first > b.first;  // Min-heap based on
        'first'
};

priority_queue<pair<int, int>, vector<pair<int,
    int>>, decltype(cmp)> pq(cmp);
```

Totient Values

```
vector<ll> phi(N,0ll);

void phi_1_to_n(int n) {
    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;
        }
    }
}
```

Segment Tree

```
template <class S, S (*op)(S, S), S (*e)()> struct
    segtree {
  public:
    int ceil_pow2(int n) {
        int x = 0;
        while ((1U << x) < (unsigned int)(n)) x++;
        return x;
    }
    segtree() : segtree(0) {}
    segtree(int n) : segtree(std::vector<S>(n, e()))
        {}
    segtree(const std::vector<S>& v) :
        _n(int(v.size())) {
        log = ceil_pow2(_n);
        size = 1 << log;
        d = std::vector<S>(2 * size, e());
```

```
        for (int i = 0; i < _n; i++) d[size + i] =
            v[i];
        for (int i = size - 1; i >= 1; i--) {
            update(i);
        }
    }

    void set(int p, S x) { // set pth value to x
        assert(0 <= p && p < _n);
        p += size;
        d[p] = x;
        for (int i = 1; i <= log; i++) update(p >> i);
    }

    S get(int p) { // get pth value of array
        assert(0 <= p && p < _n);
        return d[p + size];
    }

    S prod(int l, int r) { // find the result for [l,
        r)
        assert(0 <= l && l <= r && r <= _n);
        S sml = e(), smr = e();
        l += size;
        r += size;

        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);
    }

    S all_prod() { return d[1]; }

    template <bool (*f)(S)> int max_right(int l) {
//max_right(l, f): Finds the maximum right boundary
    where a predicate f holds.
// goes as long as f satisfies
// example: for next right_max: auto f = [&](int x){
    return x <= b; };
        return max_right(l, [](S x) { return f(x); });
    }
    template <class F> int max_right(int l, F f) {
        assert(0 <= l && l <= _n);
        assert(f(e()));
        if (l == _n) return _n;
        l += size;
        S sm = e();
        do {
            while (l % 2 == 0) l >>= 1;
            if (!f(op(sm, d[l]))) {
                while (l < size) {
                    l = (2 * l);
                    if (f(op(sm, d[l]))) {
                        sm = op(sm, d[l]);
                        l++;
                    }
                }
                return l - size;
            }
            sm = op(sm, d[l]);
            l++;
        } while ((l & -l) != l);
        return _n;
    }

    template <bool (*f)(S)> int min_left(int r) {
        return min_left(r, [](S x) { return f(x); });
    }
    template <class F> int min_left(int r, F f) {
        assert(0 <= r && r <= _n);
        assert(f(e()));
        if (r == 0) return 0;
        r += size;
        S sm = e();
        do {
            r--;
            while (r > 1 && (r % 2)) r >>= 1;
            if (!f(op(d[r], sm))) {
                while (r < size) {
                    r = (2 * r + 1);
                    if (f(op(d[r], sm))) {
                        sm = op(d[r], sm);
                        r--;
                    }
                }
                return r + 1 - size;
            }
            sm = op(d[r], sm);
        } while ((r & -r) != r);
        return 0;
    }

    // Find first index p such that sum over [0, p+1)
        >= x
    int find_first(S x) {
        // if total sum < x, there is no such index
        if (d[1] < x) return _n;
        int k = 1;
        int l = 0, r = size;
        while (k < size) {
            int lc = 2 * k;
            int m = (l + r) >> 1;
            if (d[lc] >= x) {
                // the answer is in the left child
                k = lc;
                r = m;
            } else {
                // skip left child
                x -= d[lc];
                k = lc + 1;
                l = m;
            }
        }
        return k - size;
    }

  private:
    int _n, size, log;
    std::vector<S> d;
```

```
    void update(int k) { d[k] = op(d[2 * k], d[2 * k
        + 1]); }
};

using S = int; // data type

S op (S a, S b) { // operation type
    return a + b;
}
S e() { // primary value
    return 0;
}
```

2D BIT

```
// Grid Indexing: 0-based indexing
// fenwick_tree_2d<ll> bit2d(n+1,m+1);
// Supported Operations:
//      - add(x, y, val)              A[x][y] += val
//      - sum(x, y)                   sum of submatrix
    (0,0) to (x,y)
//      - get_sum(x1, y1, x2, y2)     sum of submatrix
    (x1,y1) to (x2,y2), inclusive
template <class T>
struct fenwick_tree_2d{
    vector<vector<T>> x;
    fenwick_tree_2d(int n, int m) : x(n,
        vector<T>(m)) { }

    void add(int k1, int k2, T a) { // x[k1][k2] += a
        for (; k1 < x.size(); k1 |= k1 + 1)
            for (int k = k2; k < x[k1].size(); k |= k
                + 1)
                x[k1][k] += a;
    }

    T sum(int k1, int k2) { // sum from (0,0) to
        (k1,k2)
        T s = 0;
        for (; k1 >= 0; k1 = (k1 & (k1 + 1)) - 1)
            for (int k = k2; k >= 0; k = (k & (k +
                1)) - 1)
                s += x[k1][k];
        return s;
    }

    T get_sum(int x1, int y1, int x2, int y2) { //
        sum in rectangle (x1,y1) to (x2,y2)
        return sum(x2, y2)
             - (x1 > 0 ? sum(x1 - 1, y2) : 0)
             - (y1 > 0 ? sum(x2, y1 - 1) : 0)
             + (x1 > 0 && y1 > 0 ? sum(x1 - 1, y1 -
                 1) : 0);
    }
};
```

Sqrt Decomposition

```
struct Sqrt {
    int block_size;
```

```
    vector<int> nums;
    vector<long long> blocks;
    Sqrt(int sqrtn, vector<int> &arr) :
        block_size(sqrtn), blocks(sqrtn, 0) {
        nums = arr;
        for (int i = 0; i < nums.size(); i++) {
            blocks[i / block_size] += nums[i]; }
    }

    /** O(1) update to set nums[x] to v */
    void update(int x, int v) {
        blocks[x / block_size] -= nums[x];
        nums[x] = v;
        blocks[x / block_size] += nums[x];
    }

    /** O(sqrt(n)) query for sum of [0, r) */
    long long query(int r) {
        long long res = 0;
        for (int i = 0; i < r / block_size; i++) {
            res += blocks[i]; }
        for (int i = (r / block_size) * block_size; i
            < r; i++) { res += nums[i]; }
        return res;
    }

    /** O(sqrt(n)) query for sum of [l, r) */
    long long query(int l, int r) { return query(r) -
        query(l - 1); }
};
```

ONLINE_JUDGE

```
#include <bits/stdc++.h>
using namespace std;

int main(){
    #ifndef ONLINE_JUDGE
    freopen("input.txt","r",stdin);
    freopen("output.txt","w",stdout);
    #endif

    int n;
    cin>>n;
    for(int i=0;i<n;i++){
        cout<<i+1<<endl;
    }
    cout<<"hello"<<endl;
}
```

HLD

```
#include<bits/stdc++.h>
using namespace std;

const int N = 1e5 + 9, LG = 18, inf = 1e9 + 9;

struct ST {
#define lc (n << 1)
#define rc ((n << 1) | 1)
  int t[4 * N], lazy[4 * N];
  ST() {
    fill(t, t + 4 * N, -inf);
    fill(lazy, lazy + 4 * N, 0);
  }
  inline void push(int n, int b, int e) {
    if(lazy[n] == 0) return;
    t[n] = t[n] + lazy[n];
    if(b != e) {
      lazy[lc] = lazy[lc] + lazy[n];
      lazy[rc] = lazy[rc] + lazy[n];
    }
    lazy[n] = 0;
  }
  inline int combine(int a, int b) {
    return max(a, b); //merge left and right queries
  }
  inline void pull(int n) {
    t[n] = max(t[lc], t[rc]); //merge lower nodes of
        the tree to get the parent node
  }
  void build(int n, int b, int e) {
    if(b == e) {
      t[n] = 0;
      return;
    }
    int mid = (b + e) >> 1;
    build(lc, b, mid);
    build(rc, mid + 1, e);
    pull(n);
  }
  void upd(int n, int b, int e, int i, int j, int v) {
    push(n, b, e);
    if(j < b || e < i) return;
    if(i <= b && e <= j) {
      lazy[n] += v;
      push(n, b, e);
      return;
    }
    int mid = (b + e) >> 1;
    upd(lc, b, mid, i, j, v);
    upd(rc, mid + 1, e, i, j, v);
    pull(n);
  }
  int query(int n, int b, int e, int i, int j) {
    push(n, b, e);
    if(i > e || b > j) return -inf;
    if(i <= b && e <= j) return t[n];
    int mid = (b + e) >> 1;
    return combine(query(lc, b, mid, i, j), query(rc,
        mid + 1, e, i, j));
  }
} t;

vector<int> g[N];
int par[N][LG + 1], dep[N], sz[N];
void dfs(int u, int p = 0) {
  par[u][0] = p;
  dep[u] = dep[p] + 1;
  sz[u] = 1;
  for (int i = 1; i <= LG; i++) par[u][i] =
      par[par[u][i - 1]][i - 1];
  if (p) g[u].erase(find(g[u].begin(), g[u].end(),
      p));
  for (auto &v : g[u]) if (v != p) {
      dfs(v, u);
      sz[u] += sz[v];
      if(sz[v] > sz[g[u][0]]) swap(v, g[u][0]);
    }
}
int lca(int u, int v) {
  if (dep[u] < dep[v]) swap(u, v);
  for (int k = LG; k >= 0; k--) if (dep[par[u][k]] >=
      dep[v]) u = par[u][k];
  if (u == v) return u;
  for (int k = LG; k >= 0; k--) if (par[u][k] !=
      par[v][k]) u = par[u][k], v = par[v][k];
  return par[u][0];
}
int kth(int u, int k) {
  assert(k >= 0);
  for (int i = 0; i <= LG; i++) if (k & (1 << i)) u =
      par[u][i];
  return u;
}
int T, head[N], st[N], en[N];
void dfs_hld(int u) {
  st[u] = ++T;
  for (auto v : g[u]) {
    head[v] = (v == g[u][0] ? head[u] : v);
    dfs_hld(v);
  }
  en[u] = T;
}
int n;
int query_up(int u, int v) {
  int ans = -inf;
  while(head[u] != head[v]) {
    ans = max(ans, t.query(1, 1, n, st[head[u]],
        st[u]));
    u = par[head[u]][0];
  }
  ans = max(ans, t.query(1, 1, n, st[v], st[u]));
  return ans;
}
int query(int u, int v) {
  int l = lca(u, v);
  int ans = query_up(u, l);
  if (v != l) ans = max(ans, query_up(v, kth(v,
      dep[v] - dep[l] - 1)));
  return ans;
}
int32_t main() {
  ios_base::sync_with_stdio(0);
  cin.tie(0);

  cin >> n;
  for (int i = 1; i < n; i++) {
    int u, v;
    cin >> u >> v;
    g[u].push_back(v);
    g[v].push_back(u);
  }
  dfs(1);
```

```
  head[1] = 1;
  dfs_hld(1);
  int q;
  cin >> q;
  t.build(1, 1, n);
  while (q--) {
    string ty;
    int u, v;
    cin >> ty >> u >> v;
    if (ty == "add") {
      t.upd(1, 1, n, st[u], en[u], v);
    } else {
      cout << query(u, v) << '\n';
    }
  }
  return 0;
}
//https://www.hackerrank.com/challenges/subtrees-and-paths/problem
```

Subset Sum

```
// GeeksforGeeks.
bool isSubsetSum(vector<int> Set, int n, int sum)
{
    bool subset[n + 1][sum + 1];
    F0R(i, n + 1)
    subset[i][0] = true;
                                        // We can
    always make 0. F0R(i, sum + 1) subset[0][i] =
        false; // We can't
    make anything without taking a single element.

        FOR(i, 1, n + 1)
    {
        FOR(j, 1, sum + 1)
        {
            if (j < Set[i - 1])
                subset[i][j] = subset[i - 1][j];
            if (j >= Set[i - 1])
                subset[i][j] = subset[i - 1][j] ||
                               subset[i - 1][j -
                                   Set[i - 1]];
        }
    }
    return subset[n][sum];
}
```

RMQ

```
template <class T>
struct RMQ{
    vector<vector<T>> jmp;
    RMQ(const vector<T> &V) : jmp(1, V){
        for (int pw = 1, k = 1; pw * 2 <= sz(V); pw
            *= 2, ++k){
            jmp.emplace_back(sz(V) - pw * 2 + 1);
            rep(j, 0, sz(jmp[k]))
                jmp[k][j] = min(jmp[k - 1][j], jmp[k
                    - 1][j + pw]);
        }
    }
    Tquery(inta, int b){
        assert(a < b); // or return inf if a==b
        intdep = 31 - __builtin_clz(b - a);
        return min(jmp[dep][a], jmp[dep][b - (1 <<
            dep)]);
    }
};
```

Geometry and Others

General Macros and Functions

```
#include<bits/stdc++.h>
#include<ext/pb_ds/assoc_container.hpp>
#include<ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef long long int               ll;
typedef vector<long long int>       vll;
typedef pair<long long,long long>   pll;
typedef complex<long long int>      point;
typedef complex<double>             pointdb;
typedef tree<int, null_type, less_equal<int>,
             rb_tree_tag, tree_order_statistics_node_update> pbds;

#define loop(i,k,n)                 for(ll i=k;i<n;i++)
#define f                           first
#define s                           second
#define yes                         cout<<"Yes"<<endl
#define no                          cout<<"No"<<endl
#define Yes                         cout<<"YES"<<endl
#define No                          cout<<"NO"<<endl
#define newl                        cout<<"\n"
#define clean                       fflush(stdout)
#define all(v)                      v.begin(),v.end()
#define inparr(arr,n)               ll arr[n]; loop(i,0,n) cin>>arr[i]
#define inpvec(v,n)                 vector<ll> v(n); for(auto &i:v) cin>>i
#define CEIL(x,y)                   ((a+b-1)/b)

const ll INF=1e16;
const ll N=1e6+5;
const double EPS = 1e-9;
const ll M=998244353;

const double PI = acos(-1);
#define x                           real()
#define y                           imag()
// Vector math
#define DOT(a, b)                   ((conj(a) * (b)).real())
#define CROSS(a, b)                 ((conj(a) * (b)).imag())
// Distances
#define DIST2(a, b)                 (norm((a) - (b))) // squared distance
#define DIST(a, b)                  (abs((a) - (b))) // Euclidean distance
// Angles and slope
#define ANGLE_ELEV(a, b)            (arg((b) - (a))) // angle of elevation
#define SLOPE(a, b)                 (tan(arg((b) - (a)))) // slope of line
// Conversions
#define POLAR_TO_CART(r, theta)     (polar((r), (theta))) // polar to cartesian
#define CART_TO_POLAR_MAG(p)        (abs(p)) // magnitude
#define CART_TO_POLAR_ANG(p)        (arg(p)) // angle
// Rotations
#define ROTATE_ORIGIN(a, theta)     ((a) * polar(1.0, (theta))) // rotate
    around origin
#define ROTATE_PIVOT(a, p, theta)   (((a) - (p)) * polar(1.0, (theta)) + (p))
    // rotate around pivot
// Angle between vectors (ABC)
#define ANGLE_ABC(a, b, c)          abs(remainder(arg((a)-(b)) - arg((c)-(b)),
    2.0 * PI))

bool isLeft(point a, point b, point c){
    // return true if c lies on the left of line AB
    return (b.x - a.x)*(c.y - a.y) - (b.y - a.y)*(c.x - a.x) > 0;
}

bool isOnLine(point a, point b, point c){
    // return true if c lies on the line AB
    return (b.x - a.x)*(c.y - a.y) - (b.y - a.y)*(c.x - a.x) == 0;
}

bool onSegment(point a, point b, point c){
    // return true if point c lies on the line segment AB
    return min(a.x, b.x) <= c.x && c.x <= max(a.x, b.x) &&
           min(a.y, b.y) <= c.y && c.y <= max(a.y, b.y) &&
           CROSS(a-c,b-c) == 0;
}

long double point_to_line_dist_sq(point a, point b, point c){
    // distance from point c to line AB squared
    return (long double)(CROSS(a-c,b-c)*CROSS(a-c,b-c))/DIST2(a,b);
}

long double point_to_line_dist(pointdb a, pointdb b, pointdb c){
    // distance from point c to line AB
    long double res=(CROSS(a-c,b-c))/DIST(a,b);
    return abs(res);
}

long double point_to_segment_dist(pointdb a, pointdb b, pointdb c) {
    pointdb ab = b - a, ac = c - a, bc = c - b;
    if (DOT(ab, ac) < 0) return abs(ac);       // Closest to a
    if (DOT(-ab, bc) < 0) return abs(bc);      // Closest to b
    return abs(CROSS(ab, ac)) / abs(ab);       // Perpendicular to segment
}


ll twice_area(const vector<point>& points) {
    // area between the polygion * 2
    ll area = 0;
    int n = points.size();
    for (int i = 0; i < n; i++) {
        point p1 = points[i];
        point p2 = points[(i + 1) % n];
        area += (p1.x * p2.y) - (p1.y * p2.x);
    }
    return abs(area);  // ensure area is positive
}

ll number_of_lattice(point a, point b){
    // returns the number of lattice (integer points) between point a and b
    return __gcd(abs((a-b).x),abs((a-b).y))+1;
}

long double triangle_area_sine(pointdb a, pointdb b, pointdb c) {
    // Area = 0.5 * |AB x AC|
    pointdb ab = b - a;
    pointdb ac = c - a;
```

```
    return 0.5 * abs(CROSS(ab, ac));
}

long double circumradius(pointdb a, pointdb b, pointdb c) {
    long double side_a = abs(b - c); // Length of BC
    long double side_b = abs(a - c); // Length of AC
    long double side_c = abs(a - b); // Length of AB

    long double area = triangle_area_sine(a, b, c);

    if (area < EPS) return INF; // Degenerate triangle: radius undefined or
        infinite

    return (side_a * side_b * side_c) / (4.0 * area);
}
```

Minimum Enclosing Circle

```
// Given n points, return the radius of the minimum enclosing circle
// Call convex_hull() before this for faster solution
// Expected O(n)
double minimum_enclosing_circle_radius(vector<pointdb> &p) {
    random_shuffle(p.begin(), p.end());
    int n = p.size();
    pointdb center = p[0];
    double radius = 0;

    for (int i = 1; i < n; i++) {
        if (DIST(center, p[i]) > radius + EPS) {
            center = p[i];
            radius = 0;
            for (int j = 0; j < i; j++) {
                if (DIST(center, p[j]) > radius + EPS) {
                    center = (p[i] + p[j]) / 2.0;
                    radius = DIST(p[i], p[j]) / 2.0;
                    for (int k = 0; k < j; k++) {
                        if (DIST(center, p[k]) > radius + EPS) {
                            // Calculate circumcircle of triangle (p[i], p[j],
                                p[k])
                            pointdb a = p[i], b = p[j], c = p[k];

                            // Midpoints
                            pointdb ab = (a + b) / 2.0, ac = (a + c) / 2.0;
                            // Directions
                            pointdb d_ab = (b - a) * pointdb(0, 1);
                            pointdb d_ac = (c - a) * pointdb(0, 1);

                            // Solve intersection
                            double t = ((ac - ab).x * d_ac.y - (ac - ab).y *
                                d_ac.x) /
                                       (d_ab.x * d_ac.y - d_ab.y * d_ac.x);
                            center = ab + d_ab * t;
                            radius = DIST(center, a);
                        }
                    }
                }
            }
        }
    }
    return radius;
}
```

Convex Hull

```
int orientation(pointdb a, pointdb b, pointdb c) {
    ll v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
    if (v < 0) return -1; // clockwise
    if (v > 0) return +1; // counter-clockwise
    return 0;
}

bool cw(pointdb a, pointdb b, pointdb c, bool include_collinear) {
    int o = orientation(a, b, c);
    return o < 0 || (include_collinear && o == 0);
}
bool collinear(pointdb a, pointdb b, pointdb c) { return orientation(a, b, c) ==
    0; }

void convex_hull(vector<pointdb>& a, bool include_collinear = false) {
    pointdb p0 = *min_element(a.begin(), a.end(), [](pointdb a, pointdb b) {
        return make_pair(a.y, a.x) < make_pair(b.y, b.x);
    });
    sort(a.begin(), a.end(), [&p0](const pointdb& a, const pointdb& b) {
        int o = orientation(p0, a, b);
        if (o == 0)
            return (p0.x-a.x)*(p0.x-a.x) + (p0.y-a.y)*(p0.y-a.y)
                < (p0.x-b.x)*(p0.x-b.x) + (p0.y-b.y)*(p0.y-b.y);
        return o < 0;
    });
    if (include_collinear) {
        int i = (int)a.size()-1;
        while (i >= 0 && collinear(p0, a[i], a.back())) i--;
        reverse(a.begin()+i+1, a.end());
    }

    vector<pointdb> st;
    for (int i = 0; i < (int)a.size(); i++) {
        while (st.size() > 1 && !cw(st[st.size()-2], st.back(), a[i],
            include_collinear))
            st.pop_back();
        st.push_back(a[i]);
    }

    if (include_collinear == false && st.size() == 2 && st[0] == st[1])
        st.pop_back();

    a = st;
}
```

Polygon Intersection

```
// Get intersection point of lines (s1, e1) and (s2, e2)
point intersection(point& s1, point& e1, point& s2, point& e2) {
    point d1 = e1 - s1, d2 = e2 - s2;
    double cross_d = CROSS(d1,d2);
    if (abs(cross_d) < EPS) return s1; // parallel, arbitrary point

    double t = CROSS(s2 - s1, d2) / cross_d;
    return {s1.x + d1.x * t, s1.y + d1.y * t};
}

bool inside(point p, point a, point b) {
    // Vector from a to b
```

```
    point ab = b - a;
    // Vector from a to p
    point ap = p - a;

    // Cross product of ab and ap
    return CROSS(ab, ap) >= -EPS; // use > -eps for numerical stability
}


vector<point> polygon_intersection(const vector<point>& subject, const
    vector<point>& clip) {
    vector<point> output = subject;

    for (int i = 0; i < clip.size(); i++) {
        point clip1 = clip[i];
        point clip2 = clip[(i + 1) % clip.size()];
        vector<point> input = output;
        output.clear();

        for (int j = 0; j < input.size(); j++) {
            point curr = input[j];
            point prev = input[(j + input.size() - 1) % input.size()];

            bool curr_in = inside(curr, clip1, clip2);
            bool prev_in = inside(prev, clip1, clip2);

            if (curr_in && prev_in) {
                output.push_back(curr);
            } else if (!curr_in && prev_in) {
                output.push_back(intersection(prev, curr, clip1, clip2));
            } else if (curr_in && !prev_in) {
                output.push_back(intersection(prev, curr, clip1, clip2));
                output.push_back(curr);
            }
        }
    }

    return output;
}
```

Rotating Calipers

```
// cross(o, a, b) = (a - o) X (b - o)
ll cross(const point &o, const point &a, const point &b) {
    return (a.x - o.x) * (b.y - o.y)
         - (a.y - o.y) * (b.x - o.x);
}

// Rotating calipers to find max squared distance on convex polygon H
ll max_pair_dist2(const vector<point>& H) {
    int m = H.size();
    if (m < 2) return 0;
    if (m == 2) return DIST2(H[0], H[1]);
    ll best = 0;
    int j = 1;
    for (int i = 0; i < m; ++i) {
        // move j while area(i,i+1,j+1) > area(i,i+1,j)
        while (abs(cross(H[i], H[(i+1)%m], H[(j+1)%m]))
             > abs(cross(H[i], H[(i+1)%m], H[j])))
        {
            j = (j+1) % m;
        }
        // check distance to both j and j+1
        best = max(best, (ll)DIST2(H[i], H[j]));
        best = max(best, (ll)DIST2(H[(i+1)%m], H[j]));
    }
    return best;
}
```

Half-Planes Intersection

```
#define double long double
const double eps = 1e-12;
const double PI = acos((double)-1.0);
int sign(double x) { return (x > eps) - (x < -eps); }
struct PT {
    double x, y;
    PT() { x = 0, y = 0; }
    PT(double x, double y) : x(x), y(y) {}
    PT(const PT &p) : x(p.x), y(p.y)    {}
    PT operator + (const PT &a) const { return PT(x + a.x, y + a.y); }
    PT operator - (const PT &a) const { return PT(x - a.x, y - a.y); }
    PT operator * (const double a) const { return PT(x * a, y * a); }
    friend PT operator * (const double &a, const PT &b) { return PT(a * b.x, a *
        b.y); }
    PT operator / (const double a) const { return PT(x / a, y / a); }
    bool operator == (PT a) const { return sign(a.x - x) == 0 && sign(a.y - y) ==
        0; }
    bool operator != (PT a) const { return !(*this == a); }
    bool operator < (PT a) const { return sign(a.x - x) == 0 ? y < a.y : x < a.x;
        }
    bool operator > (PT a) const { return sign(a.x - x) == 0 ? y > a.y : x > a.x;
        }
    double norm() { return sqrt(x * x + y * y); }
    double norm2() { return x * x + y * y; }
    PT perp() { return PT(-y, x); }
    double arg() { return atan2(y, x); }
    PT truncate(double r) { // returns a vector with norm r and having same
        direction
        double k = norm();
        if (!sign(k)) return *this;
        r /= k;
        return PT(x * r, y * r);
    }
};
inline double dot(PT a, PT b) { return a.x * b.x + a.y * b.y; }
inline double dist2(PT a, PT b) { return dot(a - b, a - b); }
inline double dist(PT a, PT b) { return sqrt(dot(a - b, a - b)); }
inline double cross(PT a, PT b) { return a.x * b.y - a.y * b.x; }
inline int orientation(PT a, PT b, PT c) { return sign(cross(b - a, c - a)); }
PT perp(PT a) { return PT(-a.y, a.x); }
PT rotateccw90(PT a) { return PT(-a.y, a.x); }
PT rotatecw90(PT a) { return PT(a.y, -a.x); }
PT rotateccw(PT a, double t) { return PT(a.x * cos(t) - a.y * sin(t), a.x *
    sin(t) + a.y * cos(t)); }
PT rotatecw(PT a, double t) { return PT(a.x * cos(t) + a.y * sin(t), -a.x *
    sin(t) + a.y * cos(t)); }
double SQ(double x) { return x * x; }
double rad_to_deg(double r) { return (r * 180.0 / PI); }
double deg_to_rad(double d) { return (d * PI / 180.0); }
double get_angle(PT a, PT b) {
    double costheta = dot(a, b) / a.norm() / b.norm();
```

```
    return acos(max((double)-1.0, min((double)1.0, costheta)));
}
double area(vector<PT> &p) {
    double ans = 0; int n = p.size();
    for (int i = 0; i < n; i++) ans += cross(p[i], p[(i + 1) % n]);
    return fabs(ans);
}
// contains all points p such that: cross(b - a, p - a) >= 0
struct HP {
    PT a, b;
    HP() {}
    HP(PT a, PT b) : a(a), b(b) {}
    HP(const HP& rhs) : a(rhs.a), b(rhs.b) {}
    int operator < (const HP& rhs) const {
        PT p = b - a;
        PT q = rhs.b - rhs.a;
        int fp = (p.y < 0 || (p.y == 0 && p.x < 0));
        int fq = (q.y < 0 || (q.y == 0 && q.x < 0));
        if (fp != fq) return fp == 0;
        if (cross(p, q)) return cross(p, q) > 0;
        return cross(p, rhs.b - a) < 0;
    }
    PT line_line_intersection(PT a, PT b, PT c, PT d) {
        b = b - a; d = c - d; c = c - a;
        return a + b * cross(c, d) / cross(b, d);
    }
    PT intersection(const HP &v) {
        return line_line_intersection(a, b, v.a, v.b);
    }
};
int check(HP a, HP b, HP c) {
    return cross(a.b - a.a, b.intersection(c) - a.a) > -eps; //-eps to include
        polygons of zero area (straight lines, points)
}
// consider half-plane of counter-clockwise side of each line
// if lines are not bounded add infinity rectangle
// returns a convex polygon, a point can occur multiple times though
// complexity: O(n log(n))
vector<PT> half_plane_intersection(vector<HP> h) {
    sort(h.begin(), h.end());
    vector<HP> tmp;
    for (int i = 0; i < h.size(); i++) {
        if (!i || cross(h[i].b - h[i].a, h[i - 1].b - h[i - 1].a)) {
            tmp.push_back(h[i]);
        }
    }
    h = tmp;
    vector<HP> q(h.size() + 10);
    int qh = 0, qe = 0;
    for (int i = 0; i < h.size(); i++) {
        while (qe - qh > 1 && !check(h[i], q[qe - 2], q[qe - 1])) qe--;
        while (qe - qh > 1 && !check(h[i], q[qh], q[qh + 1])) qh++;
        q[qe++] = h[i];
    }
    while (qe - qh > 2 && !check(q[qh], q[qe - 2], q[qe - 1])) qe--;
    while (qe - qh > 2 && !check(q[qe - 1], q[qh], q[qh + 1])) qh++;
    vector<HP> res;
    for (int i = qh; i < qe; i++) res.push_back(q[i]);
    vector<PT> hull;
    if (res.size() > 2) {
        for (int i = 0; i < res.size(); i++) {
            hull.push_back(res[i].intersection(res[(i + 1) %
                ((int)res.size())]));
        }
    }
    return hull;
}
```

NTT With Any Prime MOD

```
#include<bits/stdc++.h>
using namespace std;

const int N = 3e5 + 9, mod = 998244353;

struct base {
  double x, y;
  base() { x = y = 0; }
  base(double x, double y): x(x), y(y) { }
};
inline base operator + (base a, base b) { return base(a.x + b.x, a.y + b.y); }
inline base operator - (base a, base b) { return base(a.x - b.x, a.y - b.y); }
inline base operator * (base a, base b) { return base(a.x * b.x - a.y * b.y, a.x
    * b.y + a.y * b.x); }
inline base conj(base a) { return base(a.x, -a.y); }
int lim = 1;
vector<base> roots = {{0, 0}, {1, 0}};
vector<int> rev = {0, 1};
const double PI = acosl(- 1.0);
void ensure_base(int p) {
  if(p <= lim) return;
  rev.resize(1 << p);
  for(int i = 0; i < (1 << p); i++) rev[i] = (rev[i >> 1] >> 1) + ((i & 1)  <<
      (p - 1));
  roots.resize(1 << p);
  while(lim < p) {
    double angle = 2 * PI / (1 << (lim + 1));
    for(int i = 1 << (lim - 1); i < (1 << lim); i++) {
      roots[i << 1] = roots[i];
      double angle_i = angle * (2 * i + 1 - (1 << lim));
      roots[(i << 1) + 1] = base(cos(angle_i), sin(angle_i));
    }
    lim++;
  }
}
void fft(vector<base> &a, int n = -1) {
  if(n == -1) n = a.size();
  assert((n & (n - 1)) == 0);
  int zeros = __builtin_ctz(n);
  ensure_base(zeros);
  int shift = lim - zeros;
  for(int i = 0; i < n; i++) if(i < (rev[i] >> shift)) swap(a[i], a[rev[i] >>
      shift]);
  for(int k = 1; k < n; k <<= 1) {
    for(int i = 0; i < n; i += 2 * k) {
      for(int j = 0; j < k; j++) {
        base z = a[i + j + k] * roots[j + k];
        a[i + j + k] = a[i + j] - z;
        a[i + j] = a[i + j] + z;
      }
    }
  }
}
```

```
//eq = 0: 4 FFTs in total
//eq = 1: 3 FFTs in total
vector<int> multiply(vector<int> &a, vector<int> &b, int eq = 0) {
  int need = a.size() + b.size() - 1;
  int p = 0;
  while((1 << p) < need) p++;
  ensure_base(p);
  int sz = 1 << p;
  vector<base> A, B;
  if(sz > (int)A.size()) A.resize(sz);
  for(int i = 0; i < (int)a.size(); i++) {
    int x = (a[i] % mod + mod) % mod;
    A[i] = base(x & ((1 << 15) - 1), x >> 15);
  }
  fill(A.begin() + a.size(), A.begin() + sz, base{0, 0});
  fft(A, sz);
  if(sz > (int)B.size()) B.resize(sz);
  if(eq) copy(A.begin(), A.begin() + sz, B.begin());
  else {
    for(int i = 0; i < (int)b.size(); i++) {
      int x = (b[i] % mod + mod) % mod;
      B[i] = base(x & ((1 << 15) - 1), x >> 15);
    }
    fill(B.begin() + b.size(), B.begin() + sz, base{0, 0});
    fft(B, sz);
  }
  double ratio = 0.25 / sz;
  base r2(0,  - 1), r3(ratio, 0), r4(0,  - ratio), r5(0, 1);
  for(int i = 0; i <= (sz >> 1); i++) {
    int j = (sz - i) & (sz - 1);
    base a1 = (A[i] + conj(A[j])), a2 = (A[i] - conj(A[j])) * r2;
    base b1 = (B[i] + conj(B[j])) * r3, b2 = (B[i] - conj(B[j])) * r4;
    if(i != j) {
      base c1 = (A[j] + conj(A[i])), c2 = (A[j] - conj(A[i])) * r2;
      base d1 = (B[j] + conj(B[i])) * r3, d2 = (B[j] - conj(B[i])) * r4;
      A[i] = c1 * d1 + c2 * d2 * r5;
      B[i] = c1 * d2 + c2 * d1;
    }
    A[j] = a1 * b1 + a2 * b2 * r5;
    B[j] = a1 * b2 + a2 * b1;
  }
  fft(A, sz); fft(B, sz);
  vector<int> res(need);
  for(int i = 0; i < need; i++) {
    long long aa = A[i].x + 0.5;
    long long bb = B[i].x + 0.5;
    long long cc = A[i].y + 0.5;
    res[i] = (aa + ((bb % mod) << 15) + ((cc % mod) << 30))%mod;
  }
  return res;
}

vector<int> pow(vector<int>& a, int p) {
  vector<int> res;
  res.emplace_back(1);
  while(p) {
    if(p & 1) res = multiply(res, a);
    a = multiply(a, a, 1);
    p >>= 1;
  }
  return res;
}
```

```
int main() {
  int n, k; cin >> n >> k;
  vector<int> a(10, 0);
  while(k--) {
    int m; cin >> m;
    a[m] = 1;
  }
  vector<int> ans = pow(a, n / 2);
  int res = 0;
  for(auto x: ans) res = (res + 1LL * x * x % mod) % mod;
  cout << res << '\n';
  return 0;
}
//https://codeforces.com/contest/1096/problem/G
```