Chiang Mai University

Tanya

Tanya, Tanya, Tanya

2025-06-30

Tanya's Library (1)

1.1 Template & Utilities

template.h

Description: TODO

<bits/stdc++.h> 512c6b, 85 lines

```
using namespace std;
void __print(int x) {cerr << x;}
void __print(long x) {cerr << x;}
void __print(long long x) {cerr << x;}
void __print(unsigned x) {cerr << x;}
void __print(unsigned long x) {cerr << x;}
void __print(unsigned long long x) {cerr << x;}
void __print(float x) {cerr << x;}
void __print(double x) {cerr << x;}
void __print(long double x) {cerr << x;}
void __print(char x) {cerr << '\'' << x << '\'';}
void __print(const char *x) {cerr << '\"' << x << '\"';}
void __print(const string &x) {cerr << '\"' << x << '\"';}
void __print(bool x) {cerr << (x ? "true" : "false");}

template<typename T, typename V>
void __print(const pair<T, V> &x);
template<typename T>
void __print(const T &x) {int f = 0; cerr << '{'; for (auto &i:
     x) cerr << (f++ ? ", " : ""), __print(i); cerr << "}";}
template<typename T, typename V>
void __print(const pair<T, V> &x) {cerr << '{'; __print(x.first
    ); cerr << ", "; __print(x.second); cerr << '}';}
void _print() {cerr << "]\n";}
template <typename T, typename... V>
void _print(T t, V... v) {__print(t); if (sizeof...(v)) cerr <<
     ", "; _print(v...);}
//#ifdef DEBUG
#define dbg(x...) cerr << "\e[91m"<<__func__<<":"<<__LINE__<<"
    [" << #x << "] = ["; _print(x); cerr << "\e[39m" << endl;
//#else
//#define dbg(x...)
//#endif

typedef long long ll;
typedef long double ld;
typedef complex<ld> cd;

typedef pair<int, int> pi;
typedef pair<ll,ll> pl;
typedef pair<ld,ld> pd;

typedef vector<int> vi;
typedef vector<ld> vd;
typedef vector<ll> vl;
typedef vector<pi> vpi;
typedef vector<pl> vpl;
typedef vector<cd> vcd;

template<class T> using pq = priority_queue<T>;
template<class T> using pqg = priority_queue<T, vector<T>,
    greater<T>>;

#define rep(i, a) for(int i=0;i<a;++i)
#define FOR(i, a, b) for (int i=a; i<(b); i++)
#define F0R(i, a) for (int i=0; i<(a); i++)
#define FORd(i,a,b) for (int i = (b)-1; i >= a; i--)
#define F0Rd(i,a) for (int i = (a)-1; i >= 0; i--)
#define trav(a,x) for (auto& a : x)
#define uid(a, b) uniform_int_distribution<long long>(a, b)(rng
    )

#define sz(x) (int)(x).size()
#define mp make_pair
#define pb push_back
//#define f first
//#define s second
#define lb lower_bound
#define ub upper_bound
#define all(x) x.begin(), x.end()
#define ins insert

template<class T> bool ckmin(T& a, const T& b) { return b < a ?
     a = b, 1 : 0; }
template<class T> bool ckmax(T& a, const T& b) { return a < b ?
     a = b, 1 : 0; }

mt19937 rng(chrono::steady_clock::now().time_since_epoch().
    count());

const char nl = '\n';
const int N = 2e5+1;
const int INF = 1e9+7;
const long long LINF = 1e18+7;

void solve(){
}

int main(){
    ios::sync_with_stdio(false);cin.tie(nullptr);
    int t = 1;
    cin>>t;
    while(t--)solve();
}
```

pragma.h

Description: TODO

d8191a, 3 lines

```
#pragma GCC target ("avx2")
#pragma GCC optimize ("O3")
#pragma GCC optimize ("unroll-loops")
```

1.2 Basic & Combinatorics

binpow.h

Description: TODO

d4debd, 11 lines

```
long long binpow(long long a, int n){
    long long res = 1;
    while(n>0){
        if(n&1){
            res = res * a;
        }
        a = a * a;
        n>>=1;
    }
    return res;
}
```

bigmod.h

Description: TODO

91cb0a, 11 lines

```
long long bigmod(long long a, int n){
    long long res = 1;
    while(n>0){
        if(n&1){
            res = res * a % MOD;
        }
        a = a * a % MOD;
        n>>=1;
    }
    return res;
}
```

binomialCoefficients.h

Description: TODO

3f7261, 10 lines

```
//more practical to precal C[n][r] with pascal triangle...
//...if constrain is O(n^2) able
long long C(int n, int k) {
    if(k==0)return 1;
    else if(k<0 || n<k)return 0;
    double res = 1;
    for (int i = 1; i <= k; ++i)
        res = res * (n - k + i) / i;
    return (long long)(res + 0.01);
}
```

factoradix.h

Description: TODO

<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp> e835c3, 63 lines

```
typedef __gnu_pbds::tree<int, __gnu_pbds::null_type, less<int>,
     __gnu_pbds::rb_tree_tag, __gnu_pbds::
    tree_order_statistics_node_update> ordered_set;

ordered_set st;
//st.order_of_key(x) - find # of elements in st stricly less
    than x
//st.size() - size of st
//st.find_by_order(x) - return iterator to the x-th largest
    element
//st.clear() - clear container

vector<int> factoradix_to_permutation(int n, vector<int> &
    factoradix){
    st.clear(); for(int i=1;i<=n;++i) st.ins(i);
    int cl = 0;
    while((int)factoradix.size() < n){
        factoradix.pb(0); ++cl;
    }

    vector<int> res;
    for(int i=(int)factoradix.size()-1;i>=0;--i){
        auto it = st.find_by_order(factoradix[i]);
        res.pb(*it);
        st.erase(it);
    }

    while(cl--) factoradix.pop_back();

    return res;
}

vector<int> permutation_to_factoradix(int n, vector<int> &
    permutation){
    st.clear(); for(int i=1;i<=n;++i) st.ins(i);
    vector<int> res;
    for(int i=0;i<(int)permutation.size();++i){
        int cnt = st.order_of_key(permutation[i]);
        res.pb(cnt);
        auto it = st.find_by_order(cnt); st.erase(it);
    }
    reverse(all(res));
    while(!res.empty() and res.back() == 0)res.pop_back();

    return res;
}
```

```
vector<int> decimal_to_factoradix(ll n){
    vector<int> res;
    int d = 1;
    while(n > 0){
        res.pb(n % d);
        n/=d; ++d;
    }
    return res;
}

ll factoradix_to_decimal(vector<int> &cur){
    ll res = 0;
    ll d = 1;
    for(int i=0;i<(int)cur.size();++i){
        res += d*cur[i];
        d*=(i+1);
    }
    return res;
}

//TODO using 0 base indexing, i.e. your 1st lexicographically
     permutation now is start at 0th
```

1.3 Primes & Factorization

sieve.h

Description: TODO

abb3a3, 22 lines

```
//construct is O(sqrt(n))
//can use to find all prime factor of a number in O(log n)
const int M = 2e5+1;
vector<bool> is_prime(M+1, true);
void sieve(){
    is_prime[0] = is_prime[1] = false;
    for (int i = 2; i * i <= M; i++) {
        if (is_prime[i]) {
            for (int j = i * i; j <= M; j += i)
                is_prime[j] = false;
        }
    }
    /* log n sieve (use sieve to find all prime factors in O(
        log n) )
    for (int i = 2; i <= M; i++)is_prime[i] = i;
    for (int i = 2; i * i <= M; i++) {
        if (is_prime[i] == i) {
            for (int j = i * i; j <= M; j += i)
                is_prime[j] = i;
        }
    }
    */
}
```

linear-sieve.h

Description: TODO

d893f1, 18 lines

```
const int N = 10000000;
vector<int> lp(N+1);//lp[i] = minimum prime that divide i
vector<int> pr;

void sieve(){
    for (int i=2; i <= N; ++i) {
        if (lp[i] == 0) {
            lp[i] = i;
            pr.push_back(i);
        }
        for (int j = 0; i * pr[j] <= N; ++j) {
            lp[i * pr[j]] = pr[j];
            if (pr[j] == lp[i]) {
                break;
            }
        }
    }
}
```

phi-sieve.h

Description: TODO

b4e4ad, 15 lines

```
//set value for M
const int M = 1e6;
vector<int> phi(M+1);

void phi_1_to_n() {
    for (int i = 0; i <= M; i++)
        phi[i] = i;

    for (int i = 2; i <= M; i++) {
        if (phi[i] == i) {
            for (int j = i; j <= M; j += i)
                phi[j] -= phi[j] / i;
        }
    }
}
```

phi-function.h

Description: TODO

fb8d57, 14 lines

```
//O(sqrt n)
int phi(int 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;
}
```

mobius-function.h

Description: TODO

c6a8f4, 10 lines

```
int mob[N+1];
void mobius() {
  mob[1] = 1;
  for (int i = 2; i <= N; i++){
    mob[i]--;
    for (int j = i + i; j <= N; j += i) {
      mob[j] -= mob[i];
    }
  }
}
```

miller-rabin.h

Description: TODO

980e40, 52 lines

```
//randomize primality test O(logn^2)
using u64 = uint64_t;
using u128 = __uint128_t;

u64 bigmod(u64 base, u64 e, u64 mod){
    u64 result = 1;
    base %= mod;
    while(e){
        if(e & 1)
            result = (u128)result * base % mod;
        base = (u128)base * base % mod;
        e >>= 1;
    }
    return result;
}

bool check_composite(u64 n, u64 a, u64 d, int s){
    u64 x = bigmod(a, d, n);
    if(x == 1 || x == n - 1)
        return false;
    for(int r = 1; r < s; r++){
        x = (u128)x * x % n;
        if(x == n - 1)
            return false;
    }
    return true;
}

//random number generator temporalily disable because my
     template already has it
//mt19937 rng(chrono::steady_clock::now().time_since_epoch().
     count());
long long rnd(long long x, long long y) {
  return uniform_int_distribution<long long>(x, y)(rng);
}

bool MillerRabin(u64 n, int iter=5){ // return true if n is
     probably prime, else return false.
    if(n < 4){
        return n == 2 || n == 3;
    }
    int s = 0;
    u64 d = n-1;
    while((d & 1) == 0){
        d >>= 1;
        s++;
    }

    for(int i = 0; i < iter; i++){
        long long a = rnd(2, n-2);
        if(check_composite(n, a, d, s))
            return false;
    }
    return true;
}
```

pollard-rho.h

Description: TODO

bc7c06, 20 lines

```
//randomize factorization in O(n^1/4 * logn)
long long mult(long long a, long long b, long long mod) {
    return (__int128)a * b % mod;
}

long long f(long long x, long long c, long long mod) {
    return (mult(x, x, mod) + c) % mod;
}

long long rho(long long n, long long x0=2, long long c=1) {
    long long x = x0, y = x0, g = 1;
    while (g == 1) {
        x = f(x, c, n);
        y = f(f(y, c, n), c, n);
        g = __gcd(abs(x - y), n);
    }

    if(g == n) return rho(n, uid(2, n-1), uid(-2, 2)); //uid is
         uniform long
    else return g;
}
```

discrete-log.h

Description: TODO

f7650d, 33 lines

```
// Returns minimum x for which a ^ x %m = b %m.
int solve(int a, int b, int m) {
    a %= m, b %= m;
    int k = 1, add = 0, g;
    while ((g = __gcd(a, m)) > 1) {
        if (b == k)
            return add;
        if (b % g)
            return -1;
        b /= g, m /= g, ++add;
        k = (k * 1ll * a / g) % m;
    }

    int n = sqrt(m) + 1;
    int an = 1;
    for (int i = 0; i < n; ++i)
        an = (an * 1ll * a) % m;

    map<int, int> vals;
    for (int q = 0, cur = b; q <= n; ++q) {
        vals[cur] = q;
        cur = (cur * 1ll * a) % m;
    }

    for (int p = 1, cur = k; p <= n; ++p) {
        cur = (cur * 1ll * an) % m;
        if (vals.count(cur)) {
            int ans = n * p - vals[cur] + add;
            return ans;
        }
    }
    return -1;
}
```

primitive-root.h

Description: TODO

c6d472, 34 lines

```
/*
    The following code assumes that the modulo p is a prime
        number.
    To make it works for any value of p, we must add
        calculation of phi(p)
*/
int powmod (int a, int b, int p) {
    int res = 1;
    while (b)
        if (b & 1)
            res = int (res * 1ll * a % p),  --b;
        else
            a = int (a * 1ll * a % p),  b >>= 1;
    return res;
}

int generator (int p) {
    vector<int> fact;
    int phi = p-1,  n = phi; // cal phi p by assuming it's
        prime
    for (int i=2; i*i<=n; ++i)
        if (n % i == 0) {
            fact.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1)
        fact.push_back (n);

    for (int res=2; res<=p; ++res) {
        bool ok = true;
        for (size_t i=0; i<fact.size() && ok; ++i)
            ok &= powmod (res, phi / fact[i], p) != 1;
        if (ok)  return res;
    }
    return -1;
}
```

crt.h

Description: TODO

efd085, 51 lines

```
class CRT { // ChineseRemainderTheorem
    typedef long long vlong;
    typedef pair<vlong,vlong> pll;
    vector<pll> equations;

public:
    void clear() {
        equations.clear();
    }
    void addEquation( vlong r, vlong m ) {
        equations.push_back({r, m});
    }

    void ext_gcd(vlong a, vlong b, vlong& x, vlong& y) {
        if (b == 0) {
            x = 1;
            y = 0;
            return;
        }
        vlong x1, y1;
        ext_gcd(b, a % b, x1, y1);
        x = y1;
        y = x1 - y1 * (a / b);
    }

    pll solve() {
        if (equations.size() == 0) return {-1,-1};

        vlong a1 = equations[0].first;
        vlong m1 = equations[0].second;
        a1 %= m1;
        for ( int i = 1; i < (int)equations.size(); i++ ) {
            vlong a2 = equations[i].first;
            vlong m2 = equations[i].second;

            vlong g = __gcd(m1, m2);
            if ( a1 % g != a2 % g ) return {-1,-1};
            vlong p, q;
            ext_gcd(m1/g, m2/g, p, q);

            vlong mod = m1 / g * m2;
            p = (p%mod+mod)%mod, q = (q%mod+mod)%mod;
            vlong x = ( (__int128)a1 * (m2/g) % mod *q % mod +
                 (__int128)a2 * (m1/g) % mod * p % mod ) % mod;
            a1 = x;
            if ( a1 < 0 ) a1 += mod;
            m1 = mod;
        }
        return {a1, m1};
    }
};
```

diophantine.h

Description: TODO

4fcad2, 89 lines

```
ll gcd(ll a, ll b, ll& x, ll& y) { //gcd extended
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    ll x1, y1;
    ll d = gcd(b, a % b, x1, y1);
    x = y1;
    y = x1 - y1 * (a / b);
    return d;
}

bool find_any_solution(ll a, ll b, ll c, ll &x0, ll &y0, ll &g)
     {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}

void shift_solution(ll & x, ll & y, ll a, ll b, ll cnt) { //
    make sure that a,b coprime
    x += cnt * b;
    y -= cnt * a;
}

ll find_all_solutions(ll a, ll b, ll c, ll minx, ll maxx, ll
    miny, ll maxy) {
    ll x, y, g;
    if (!find_any_solution(a, b, c, x, y, g))
        return 0;
    a /= g;
    b /= g;

    ll sign_a = a > 0 ? +1 : -1;
    ll sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx - x) / b);
    if (x < minx)
        shift_solution(x, y, a, b, sign_b);
    if (x > maxx)
        return 0;
    ll lx1 = x;

    shift_solution(x, y, a, b, (maxx - x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b, -sign_b);
    ll rx1 = x;

    shift_solution(x, y, a, b, -(miny - y) / a);
    if (y < miny)
        shift_solution(x, y, a, b, -sign_a);
    if (y > maxy)
        return 0;
    ll lx2 = x;

    shift_solution(x, y, a, b, -(maxy - y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b, sign_a);
    ll rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);

    ll lx = max(lx1, lx2);
```

```
    ll rx = min(rx1, rx2);
    if (lx > rx)
        return 0;

    return (rx - lx) / abs(b) + 1;
}
//number of solution of ax + by = c in general
ll number_of_solution(ll a, ll b, ll c, ll x1, ll x2, ll y1, ll
    y2){
    if(a == 0 and b == 0 and c == 0)return (x2-x1+1)*(y2-y1+1);
    else if(a == 0 and b == 0) return 0;
    else if(a == 0){
        if(c%b!=0 or y1>c/b or y2<c/b)return 0;
        else return (x2-x1+1);
    } else if(b == 0){
        if(c%a!=0 or x1>c/a or x2<c/a)return 0;
        else return (y2-y1+1);
    } else {
        return find_all_solutions(a, b, c, x1, x2, y1, y2);
    }
}
```

1.4 Arrays & Trees

fenwick-tree.h

Description: TODO

a9def9, 27 lines

```
//TODO: use 1 base indexing only!!! becareful
vector<long long>bit;

//range sum point update(k=new_val-old_val)
void add(int i, int k){//add k to ith data and it's parent
    range so on
    while(i<(int)bit.size()){
        bit[i]+=k;
        i+=i&-i;//add last set bit
    }
}

ll sum(int i){//culmulative sum to ith data
    ll sum=0;
    while(i>0){
        sum+=bit[i];
        i-=i&-i;
    }
    return sum;
}

void build_bit(vl &a){
    bit=a;
    for(int i=1;i<(int)bit.size();++i){
        int p=i+(i&-i);//index to parent
        if(p<(int)bit.size())bit[p]+=bit[i];
    }
}
```

normalSegtree.h

Description: TODO

d1ac68, 30 lines

```
//TODO: use 0 base indexing, initialize tree[] array (
    preferably 4 times size of nodes array
vector<long long>tree;
void update(int node,int n_l,int n_r,int q_i,long long value){
    if(n_r<q_i || q_i<n_l)return;
    if(q_i==n_l && n_r==q_i){
        tree[node] = value;
        return;
    }
    int mid = (n_r+n_l)/2;
    update(2*node,n_l,mid,q_i,value);
    update(2*node+1,mid+1,n_r,q_i,value);
    tree[node] = tree[2*node] + tree[2*node+1];
}

long long f(int node,int n_l,int n_r,int q_l,int q_r){
    if(n_r<q_l || q_r<n_l)return 0;
    if(q_l<=n_l && n_r<=q_r)return tree[node];
    int mid = (n_l+n_r)/2;
    return f(2*node,n_l,mid,q_l,q_r) + f(2*node+1,mid+1,n_r,q_l
        ,q_r);
}

int build_tree(vi &a,int n){
    tree.clear();
    int m=n;
    while(__builtin_popcount(m)!=1)++m;
    tree.resize(2*m+10,0);
    for(int i=0;i<n;++i)tree[i+m]=a[i];
    for(int i=m-1;i>=1;--i)tree[i]=tree[2*i]+tree[2*i+1];
    return m;
}
```

segtree.h

Description: TODO

e6e225, 6 lines

```
long long f(int node,int n_l,int n_r,int q_l,int q_r){
    if(n_r<q_l || q_r<n_l)return 0;
    if(q_l<=n_l && n_r<=q_r)return tree[node];
    int mid = (n_l+n_r)/2;
    return f(2*node,n_l,mid,q_l,q_r) + f(2*node+1,mid+1,n_r,q_l
        ,q_r);
}
```

lazySegtree.h

Description: TODO

59e123, 50 lines

```
//TODO: use 0 base indexing
vector<long long> tree,lazy;
void update(int node,int n_l,int n_r,int q_l,int q_r,int value)
    {
    if(lazy[node]!=0){
        tree[node]+=(long long)(n_r-n_l+1)*lazy[node]; // for
            range + update
        if(n_l!=n_r){
            lazy[2*node]+=lazy[node];
            lazy[2*node+1]+=lazy[node];
        }
        lazy[node] = 0;
    }
    if(n_r<q_l || q_r<n_l)return;
    if(q_l<=n_l && n_r<=q_r){
        tree[node]+=(long long)(n_r-n_l+1)*value; // for range
            + update
        if(n_l!=n_r){
            lazy[2*node]+=value;
            lazy[2*node+1]+=value;
        }
        return;
    }
    int mid = (n_r+n_l)/2;
    update(2*node,n_l,mid,q_l,q_r,value);
    update(2*node+1,mid+1,n_r,q_l,q_r,value);
    tree[node] = tree[2*node] + tree[2*node+1];
}

long long f(int node,int n_l,int n_r,int q_l,int q_r){
    if(lazy[node]!=0){
        tree[node]+=(long long)(n_r-n_l+1)*lazy[node];
        if(n_l!=n_r){
            lazy[2*node] += lazy[node];
            lazy[2*node+1] += lazy[node];
        }
        lazy[node] = 0;
    }
    if(n_r<q_l || q_r<n_l)return 0;
    if(q_l<=n_l && n_r<=q_r)return tree[node];
    int mid = (n_l+n_r)/2;
    return f(2*node,n_l,mid,q_l,q_r) + f(2*node+1,mid+1,n_r,q_l
        ,q_r);
}

int build_tree(vi &a,int n){
    tree.clear(); lazy.clear();
    int m=n;
    while(__builtin_popcount(m)!=1)++m;
    tree.resize(2*m+10,0); lazy.resize(2*m+10,0);
    for(int i=0;i<n;++i)tree[i+m]=a[i];
    for(int i=m-1;i>=1;--i)tree[i]=tree[2*i]+tree[2*i+1];
    return m;
}
```

persistent-segtree.h

Description: TODO

c4a26b, 38 lines

```
//Warning! This implementation is not garbage collected, so the
     tree nodes aren't deleted even if the instance of the
     segment tree is taken out of scope.
struct Node {
    ll val;
  Node *l, *r;

  Node(ll x) : val(x), l(nullptr), r(nullptr) {}
  Node(Node *ll, Node *rr) {
    l = ll, r = rr;
    val = 0;
    if (l) val += l->val;
    if (r) val += r->val;
  }
  Node(Node *cp) : val(cp->val), l(cp->l), r(cp->r) {}
};

int n, cnt = 1;
ll a[200001];
Node *roots[200001];

Node *build(int l = 1, int r = n) {
  if (l == r) return new Node(a[l]);
  int mid = (l + r) / 2;
  return new Node(build(l, mid), build(mid + 1, r));
}

Node *update(Node *node, int val, int pos, int l = 1, int r = n
    ) {
  if (l == r) return new Node(val);
  int mid = (l + r) / 2;
  if (pos > mid) return new Node(node->l, update(node->r, val,
      pos, mid + 1, r));
  else return new Node(update(node->l, val, pos, l, mid), node
      ->r);
}

ll query(Node *node, int a, int b, int l = 1, int r = n) {
  if (l > b || r < a) return 0;
  if (l >= a && r <= b) return node->val;
  int mid = (l + r) / 2;
  return query(node->l, a, b, l, mid) + query(node->r, a, b,
      mid + 1, r);
}
```

merge-sort-tree.h

Description: TODO

<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp>, <ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp> 2d0612, 94 lines

```
//merge sort tree with segment tree
typedef __gnu_pbds::tree<pair<int,int>, __gnu_pbds::null_type,
     less<pair<int,int>>, __gnu_pbds::rb_tree_tag, __gnu_pbds::
     tree_order_statistics_node_update> ordered_set;
ordered_set st;
//st.order_of_key({x,-1}) - find # of elements in st stricly
     less than x
//st.clear() - clear container

vector<ordered_set> mtree;

ordered_set merge(ordered_set &a, ordered_set &b){
    ordered_set result;
    for(auto&p:a){
        result.ins(p);
    }
    for(auto&p:b){
        result.ins(p);
    }
    return result;
}

void update(int node,int n_l,int n_r,int q_i,int id,int old_val
     ,int value){
    if(n_r<q_i || q_i<n_l)return;
    if(q_i==n_l && n_r==q_i){
        auto it=mtree[node].find({old_val,id});
        mtree[node].erase(it);
        mtree[node].ins({value,id});
        return;
    }
    int mid = (n_r+n_l)/2;
    update(2*node,n_l,mid,q_i,id,old_val,value);
    update(2*node+1,mid+1,n_r,q_i,id,old_val,value);
    auto it=mtree[node].find({old_val,id});
    if(it!=mtree[node].end()){
        mtree[node].erase(it);
        mtree[node].ins({value,id});
    }
}

int f(int node,int n_l,int n_r,int q_l,int q_r, int value){
    if(n_r<q_l || q_r<n_l)return 0;
    if(q_l<=n_l && n_r<=q_r){
        return mtree[node].order_of_key({value,-1});
    }
    int mid = (n_l+n_r)/2;
    return f(2*node,n_l,mid,q_l,q_r,value)+f(2*node+1,mid+1,n_r
         ,q_l,q_r,value);
}

void build_mtree(vi &a){
    int n=(int)a.size();
    int m=n; while(__builtin_popcount(m)!=1)++m;
    //for(int i=0;i<2*m;++i)mtree[i].clear();
    mtree.resize(2*m);
    for(int i=0;i<n;++i)mtree[i+m].ins({a[i],i});
    for(int i=m-1;i>=1;--i)mtree[i]=merge(mtree[2*i],mtree[2*i
         +1]);
}
//merge sort tree with fenwick tree(BIT) (4 times less space)

typedef __gnu_pbds::tree<pair<int,int>, __gnu_pbds::null_type,
     less<pair<int,int>>, __gnu_pbds::rb_tree_tag, __gnu_pbds::
     tree_order_statistics_node_update> ordered_set;
ordered_set st;
//st.order_of_key(x) - find # of elements in st stricly less
     than x

//TODO: use 1 base indexing
vector<ordered_set>bit;

void update(int i,int k,int old_value, int new_value){
    while(i<(int)bit.size()){
        auto it=bit[i].find({old_value,k});
        assert(it!=bit[i].end());
        if(it!=bit[i].end()){
            bit[i].erase(it);
        }
        bit[i].ins({new_value,k});
        i+=i&-i;//add last set bit
    }
}

int F(int i, int k){//culmulative sum to ith data
    int sum=0;
    while(i>0){
        sum+=bit[i].order_of_key({k,-1});
        i-=i&-i;
    }
    return sum;
}

void build_bit(vi &a){
    bit.resize((int)a.size());
    for(int i=1;i<(int)a.size();++i)bit[i].ins({a[i],i});
    for(int i=1;i<(int)bit.size();++i){
        int p=i+(i&-i);//index to parent
        if(p<(int)bit.size()){
            for(auto&x:bit[i])bit[p].ins(x);
        }
    }
}
```

2D-segtree.h

Description: TODO

ee2c74, 82 lines

```
// 2D segtree stored as tree[4*n][4*m]
vector<vector<ll>> tree;
int n, m; // This shit in global remember
vector<vector<ll>> A; // This shit in global too

void buildY(int vx, int lx, int rx, int vy, int ly, int ry) {
    if (ly == ry) {
        if (lx == rx) {
            tree[vx][vy] = A[lx][ly];
        } else {
            tree[vx][vy] = tree[vx*2][vy] + tree[vx*2+1][vy];
        }
    } else {
        int my = (ly + ry) >> 1;
        buildY(vx, lx, rx, vy*2,   ly,   my);
        buildY(vx, lx, rx, vy*2+1, my+1, ry);
        tree[vx][vy] = tree[vx][vy*2] + tree[vx][vy*2+1];
    }
}

void buildX(int vx = 1, int lx = 1, int rx = n) {
    if (lx != rx) {
        int mx = (lx+rx)>>1;
        buildX(vx*2,   lx,   mx);
        buildX(vx*2+1, mx+1, rx);
    }
    buildY(vx, lx, rx, 1, 1, m);
}

void updateY(int vx, int lx, int rx, int x,
             int vy, int ly, int ry, int y, ll val) {
    if (ly == ry) {
        if (lx == rx) {
            tree[vx][vy] = val;
        } else {
            tree[vx][vy] = tree[vx*2][vy] + tree[vx*2+1][vy];
        }
    } else {
        int my = (ly+ry)>>1;
        if (y <= my)
            updateY(vx,lx,rx,x, vy*2,   ly,   my, y, val);
        else
            updateY(vx,lx,rx,x, vy*2+1, my+1, ry, y, val);
        tree[vx][vy] = tree[vx][vy*2] + tree[vx][vy*2+1];
    }
}

void updateX(int vx, int lx, int rx, int x, int y, ll val) {
    if (lx != rx) {
        int mx = (lx+rx)>>1;
        if (x <= mx) updateX(vx*2,   lx,   mx, x, y, val);
        else         updateX(vx*2+1, mx+1, rx, x, y, val);
    }
    updateY(vx, lx, rx, x, 1, 1, m, y, val);
}

ll queryY(int vx, int vy, int ly, int ry, int y1, int y2) {
    if (y2 < ly || ry < y1) return 0;
    if (y1 <= ly && ry <= y2) return tree[vx][vy];
    int my = (ly+ry)>>1;
    return queryY(vx,vy*2,  ly,   my, y1,y2)
         + queryY(vx,vy*2+1,my+1, ry,  y1,y2);
}

ll queryX(int vx, int lx, int rx,
          int x1, int x2, int y1, int y2) {
    if (x2 < lx || rx < x1) return 0;
    if (x1 <= lx && rx <= x2)
        return queryY(vx, 1, 1, m, y1, y2);
    int mx = (lx+rx)>>1;
    return queryX(vx*2,   lx,   mx, x1,x2,y1,y2)
         + queryX(vx*2+1, mx+1, rx, x1,x2,y1,y2);
}

void build(int n, int m){
    tree.assign(4*(n+2), vector<ll>(4*(m+2), 0));
    buildX();
}

//using like this n,m = row,col dimention (1-base index)
//updateX(1,1,n,x,y,v);
//queryX(1,1,n,x1,x2,y1,y2)
```

order-statistics-tree.h

Description: TODO

<ext/pb_ds/assoc_container.hpp>, <ext/pb_ds/tree_policy.hpp> 033b41, 7 lines

```
typedef __gnu_pbds::tree<int, __gnu_pbds::null_type, less<int>,
     __gnu_pbds::rb_tree_tag, __gnu_pbds::
     tree_order_statistics_node_update> ordered_set;

ordered_set st;
//st.order_of_key(x) - find # of elements in st stricly less
     than x
//st.size() - size of st
//st.find_by_order(x) - return iterator to the x-th largest
     element
```

```
//st.clear() - clear container
```

RMQ-1D.h

Description: TODO

e37d08, 25 lines

```
//general sparse table template(can be use for all other
     offline queries)
int rmq[N][20];

void build_rmq(vi &a){
    for(int j=0;j<20;++j){
        for(int i=0;i<(int)a.size();++i){
            if(j==0){
                rmq[i][0]=a[i];
            } else if(i+(1<<(j-1))<(int)a.size()){
                rmq[i][j]=min(rmq[i][j-1],rmq[i+(1<<(j-1))][j
                    -1]);
            }
        }
    }
}

int query(int l, int r){
    int i=1,sub_array_size=r-l+1, ans=INF;
    for(int j=0;j<30;++j){
        if((1<<j)&(sub_array_size)){
            ans=min(ans,rmq[i][j]);
            i+=(1<<j);
        }
    }
    return ans;
}
```

RMQ-2D.h

Description: TODO

0f2f45, 46 lines

```
//general template for 2D offline queries
int rmq[N][N][20][20]; //N need to be around 1e2

void build_rmq(vector<vi>& a){
    for(int j=0;j<20;++j){
        for(int k=0;k<20;++k){
            for(int r=0;r<(int)a.size();++r){
                for(int c=0;c<(int)a[0].size();++c){
                    if(j==0&&k==0)rmq[r][c][0][0]=a[r][c];
                    else {
                        if(j==0 && k>0 && (c+(1<<(k-1))<(int)a
                            [0].size() )){
                            rmq[r][c][j][k]=min(rmq[r][c][j][k
                                -1],
                            rmq[r][c+(1<<(k-1))][j][k-1]);
                        } else if(k==0 && j>0 && (r+(1<<(j-1))
                            <(int)a.size()) ){
                            rmq[r][c][j][k]=min(rmq[r][c][j-1][
                                k],
                            rmq[r+(1<<(j-1))][c][j-1][k]);
                        } else if( c+(1<<(k-1))<(int)a[0].size
                            ()
                        && r+(1<<(j-1))<(int)a.size() ){
                            rmq[r][c][j][k]=min({rmq[r][c][j
                                -1][k-1],
                            rmq[r][c+(1<<(k-1))][j-1][k-1],
                            rmq[r+(1<<(j-1))][c][j-1][k-1],
                            rmq[r+(1<<(j-1))][c+(1<<(k-1))][j
                                -1][k-1]});
                        }
                    }
                }
            }
        }
    }
}

int query(int r1,int c1,int r2,int c2){
    int x=r1,y=c1,h=r2-r1+1,w=c2-c1+1,ans=INF;
    for(int j=0;j<30;++j){
        if((1<<j)&h){
            for(int k=0;k<30;++k){
                if((1<<k)&w){
                    ans=min(ans,rmq[x][y][j][k]);
                    y+=(1<<k);
                }
            }
            y=c1;
            x+=(1<<j);
        }
    }
    return ans;
}
```

1.5 Hashing & Tables

gp-hash-table.h

Description: TODO

<ext/pb_ds/assoc_container.hpp> 3b295b, 7 lines

```
using namespace __gnu_pbds;

const int RANDOM = chrono::high_resolution_clock::now().
    time_since_epoch().count();
struct chash {
    int operator()(int x) const { return x ^ RANDOM; }
};
gp_hash_table<int, int, chash> table;
```

cht.h

Description: TODO

87ad47, 28 lines

```
struct line {
    long long m, c;
    long long eval(long long x) { return m * x + c; }
    long double intersectX(line l) { return (long double) (c -
        l.c) / (l.m - m); }
};

deque<line> dq;
dq.push_front({0, 0});//cant be put in global, remove this to
    local function
//if query ask for minimum remove this line after 1st insertion

//TODO NOTE***: maximum and minimum value exist in bot left
    most and rightmost of convex hull so do search on both l
    to r and r to l

//constructing hull from l to r, maintain correct hull at
    rightmost
/*  ***inserting line (maximum hull)
    line cur = line{...some m, ...some c}
    while(dq.size()>=2&&cur.intersectX(dq.back())
        <=cur.intersectX(dq[dq.size()-2]))dq.pop_back();
    dq.pb(cur);
*/

//constructing hull from r to l, maintain correct hull at
    leftmost
/*  inserting line (maximum hull)
    line cur = line{...some m, ...some c}
    while(dq.size()>=2&&cur.intersectX(dq[0])>=cur.intersectX(
        dq[1])){
        dq.pop_front();
    }
    dq.push_front(cur);
*/
```

1.6 Graph Algorithms

dijkstra.h

Description: TODO

3fd08c, 17 lines

```
//initialize dist, proc
    for (int i = 1; i <= m; i++) dist[i] = INF;
    dist[x] = 0;
    priority_queue<pair<ll, int>>q;
    q.push({0,x});
    while (!q.empty()) {
        int a = q.top().second; q.pop();
        if (proc[a]) continue;
        proc[a] = true; // this fucker i dont remember all the
            time
        for (auto u : adj[a]) {
            int b = u.first, w = u.second;
            if (dist[a]+w < dist[b]) {
                dist[b] = dist[a]+w;
                q.push({-dist[b],b});
            }
        }
    }
```

floyd-warshall-stp.h

Description: TODO

ab1dcf, 11 lines

```
//don't forget to initialize the base case of weight
//for example d[i][i] = 0 for all i and other given weight
//this work for negative weight without negative cycle
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]);
        }
    }
}
```

kruskal-mst.h

Description: TODO

5dd25a, 37 lines

```
//kruskal
vector<tuple<int,int,int>>el;
int parent[N+1], _rank[N+1];

int find_set(int v) {
    if (v == parent[v])
        return v;
    return parent[v] = find_set(parent[v]);
}

void make_set(int v) {
    parent[v] = v;
    _rank[v] = 1;
}

void union_sets(int a, int b) {
    a = find_set(a);
    b = find_set(b);
    if (a != b) {
        if (_rank[a] < _rank[b])
            swap(a, b);
        parent[b] = a;
        _rank[a]+=_rank[b];
```

```
    }
}

ll mst(int n){
    sort(all(el)); for(int i=1;i<=n;++i)make_set(i);
    ll mst_cost=0;
    for(auto&t:el){
        int w=get<0>(t), a=get<1>(t), b=get<2>(t);
        if(find_set(a) != find_set(b)){
            mst_cost+=w, union_sets(a,b);
        }
    }
    return mst_cost;
}
```

prim-mst.h

Description: TODO

a70ceb, 24 lines

```
//*note: unlike kruskal, prim's algorithm assume all vertices
     belong in same graph
vector<pair<int,int>>adj[N+1];
bool taken[N+1];
priority_queue<pair<int,int>>PQ;

void process(int v){
    taken[v]=1;
    for(auto&p:adj[v]){
        int b=p.second, w=p.first;
        if(!taken[b])PQ.push({-w,-b});
    }
}

ll mst(){
    process(1);
    ll mst_cost=0;
    while(!PQ.empty()){
        pair<int,int> front=PQ.top(); PQ.pop();
        int u=-front.second, w=-front.first;
        if(!taken[u])
            mst_cost+=w, process(u);
    }
    return mst_cost;
}
```

dinic.h

Description: TODO

ce6220, 82 lines

```
//COPY FROM BENQ GITHUB I dont fucking know how this work
//General graphs: O(V^2E);
//Bipartite matching(unit caps): O(E sqrt2(V))
struct Dinic {
    using F = long long; // flow type
    struct Edge { int to; F flow, cap; };
    int N;
    vector<Edge> edges;
    vector<vector<int>> adj;
    vector<int> level;
    vector<vector<int>::iterator> it;

    // Initialize Dinic for a graph with _N vertices (0-indexed
        or 1-indexed)
    void init(int _N) {
        N = _N;
        adj.assign(N, {});
        edges.clear();
        level.resize(N);
        it.resize(N);
    }

    // Add edge u -> v with capacity 'cap', and reverse edge v
        -> u with capacity 'rev_cap'
    void addEdge(int u, int v, F cap, F rev_cap = 0) {
        // forward edge
        adj[u].push_back((int)edges.size());
        edges.push_back({v, 0, cap});
        // reverse edge
        adj[v].push_back((int)edges.size());
        edges.push_back({u, 0, rev_cap});
    }

    // Build level graph via BFS
    bool bfs(int s, int t) {
        fill(level.begin(), level.end(), -1);
        for (int i = 0; i < N; i++) it[i] = adj[i].begin();
        queue<int> q;
        level[s] = 0;
        q.push(s);
        while (!q.empty()) {
            int u = q.front(); q.pop();
            for (int idx : adj[u]) {
                const Edge &e = edges[idx];
                if (level[e.to] < 0 && e.flow < e.cap) {
                    level[e.to] = level[u] + 1;
                    q.push(e.to);
                }
            }
        }
        return level[t] >= 0;
    }

    // DFS to send flow
    F dfs(int u, int t, F pushed) {
        if (u == t || pushed == 0) return pushed;
        for (; it[u] != adj[u].end(); ++it[u]) {
            int idx = *it[u];
            Edge &e = edges[idx];
            if (level[e.to] != level[u] + 1 || e.flow == e.cap)
                continue;
            F tr = dfs(e.to, t, min(pushed, e.cap - e.flow));
            if (tr > 0) {
                e.flow += tr;
                edges[idx ^ 1].flow -= tr; // reverse edge
                return tr;
            }
        }
        return 0;
    }

    // Compute max flow from s to t
    F maxFlow(int s, int t) {
        F flow = 0;
        while (bfs(s, t)) {
            while (F pushed = dfs(s, t, numeric_limits<F>::max
                ())) {
                flow += pushed;
            }
        }
        return flow;
    }

    // After maxFlow, vertices reachable from s in residual
        graph have level >= 0.
    // Edges from reachable u to unreachable v form a min-cut.
};
```

eulerian-cycle.h

Description: TODO

87df80, 25 lines

```
//TODO: checking all node degrees is even ?
    //TODO: checking if all edges are connectedly reachable

    stack<int> st;
    vi res;
    st.push(1);
    while(!st.empty()){
        int v = st.top();
        if(deg[v] == 0){
            res.pb(v);
            st.pop();
        } else {
            int rm = -1; //must exist
            for(auto &b:adj[v]){
                rm = b;
                break;
            }
            adj[v].erase(rm); deg[v]--;
            adj[rm].erase(v); deg[rm]--;
            st.push(rm);
        }
    }
    //reverse the res if the edge is directed !! also changing
        the checking condition
    for(auto &x:res)cout << x << " ";
    cout << nl;
```

1.7 Graph Decompositions & Connectivity

articulation-points.h

Description: TODO

23f413, 37 lines

```
int n; // number of nodes
vector<vector<int>> adj; // adjacency list of graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
```

```
        if (!visited[i])
            dfs (i);
    }
}
```

bridge-finding.h

Description: TODO

747a19, 30 lines

```
//NOTE**: not work for multigraph, I'm too lazy too fix for now
/*
    dfs-tree bridge finding algorithm
    let lvl[root] = 1 (root = 1)
*/
int root = 1;
vector<pair<int,int>> bridge;
int dp[N+1], lvl[N+1];

void dfs_tree(int a, int e){
    if(a == root)lvl[a] = 1;
    dp[a] = 0;
    for(auto &b:adj[a]){
        if(b == e)continue;

        if(lvl[b] == 0){
            lvl[b] = lvl[a] + 1;
            dfs_tree(b, a);
            dp[a] += dp[b];
        } else if(lvl[b] < lvl[a]){
            dp[a]++;
        } else if(lvl[b] > lvl[a]){
            dp[a]--;
        }
    }

    if(lvl[a] > 1 and dp[a] == 0){
        bridge.pb({min(a,e), max(a,e)});
    }
}
```

cycle-detection.h

Description: TODO

6b432f, 16 lines

```
vi adj[N+1];
bool vst[N+1];
bool cur[N+1];
bool cycle=0;

vi temp;
void dfs(int a){
    vst[a]=1;
    cur[a]=1;
    for(auto&b:adj[a]){
        if(!vst[b])dfs(b);
        else if(cur[b])cycle=1;
    }
    temp.pb(a); //topological sort order
    cur[a]=0;
}
```

tarjan-scc.h

Description: TODO

3e5381, 35 lines

```
//Trajan's algorithm
vi adj[N+1];
int id[N+1], low[N+1];
bool vst[N+1], in_stack[N+1];

stack<int>st;
vector<vi>scc;
vi comp;
int tin=0;
void dfs(int a){
    st.push(a);
    vst[a]=in_stack[a]=1;
    id[a]=low[a]=tin++;

    for(auto&b:adj[a]){
        if(!vst[b]){
            dfs(b);
            low[a]=min(low[a],low[b]); //Backtrack low-link
                value
        } else if(in_stack[b]){
            //Backtrack low-link value
            low[a]=min(low[a],low[b]);
        }
    }
    if(id[a] == low[a]){ //pop scc
        while(!st.empty()){
            int v=st.top();
            comp.pb(v), st.pop();
            in_stack[v]=0;
            if(v==a)break; //stop when scc is found
        }
    }
    if(!comp.empty()){
        scc.pb(comp), comp.clear();
    }
}
```

centroid-decomposition.h

Description: TODO

70d410, 32 lines

```
vector<int> adj[N];
bool is_removed[N];
int subtree_size[N];
int get_subtree_size(int node, int parent = -1) {
  subtree_size[node] = 1;
  for (int child : adj[node]) {
    if (child == parent || is_removed[child]) { continue; }
    subtree_size[node] += get_subtree_size(child, node);
  }
  return subtree_size[node];
}
int get_centroid(int node, int tree_size, int parent = -1) {
  for (int child : adj[node]) {
    if (child == parent || is_removed[child]) { continue; }
    if (subtree_size[child] * 2 > tree_size) {
      return get_centroid(child, tree_size, node);
    }
  }
  return node;
}
void build_centroid_decomp(int node = 0) {
  int centroid = get_centroid(node, get_subtree_size(node));
  is_removed[centroid] = true;

  // do something

  for (int child : adj[centroid]) {
    if (is_removed[child]) { continue; }
    build_centroid_decomp(child);
  }
}
```

hld.h

Description: TODO

e4ffaf, 152 lines

```
/*hld*/
//TODO: initialize constant N
vector<int>adj[N];

void add_edges(int a, int b){
    adj[a].pb(b);
    adj[b].pb(a);
}

const int LOG = 20;
int timer = 0, seg_tree_size = 0;
int depth[N], parent[N], n_subtree[N], heavyChild[N], chain[N],
    SegTreePos[N];
int cost[N];
int up[N][LOG]; // 2^j-th ancestor of n

/*SegTree*/
//finding max edge query
ll tree[4*N];
void update(int node,int n_l,int n_r,int q_i,int value){
    if(n_r<q_i || q_i<n_l)return;
    if(q_i==n_l && n_r==q_i){
        tree[node] = value;
        return;
    }
    int mid = (n_r+n_l)/2;
    update(2*node,n_l,mid,q_i,value);
    update(2*node+1,mid+1,n_r,q_i,value);
    tree[node] = max(tree[2*node], tree[2*node+1]);
}

long long f(int node,int n_l,int n_r,int q_l,int q_r){
    if(n_r<q_l || q_r<n_l)return 0;
    if(q_l<=n_l && n_r<=q_r)return tree[node];
    int mid = (n_l+n_r)/2;
    return max(f(2*node,n_l,mid,q_l,q_r) , f(2*node+1,mid+1,n_r
        ,q_l,q_r));
}
/*SegTree*/

//preprecessing the arrays above
void dfs(int a,int e){
    chain[a] = a;
    n_subtree[a] = 1;
    int bestSon = -1, hmax = -1;
    for(auto b:adj[a]){
        if(b == e)continue;
        depth[b] = depth[a] + 1;
        parent[b] = a;
        up[b][0] = parent[b];
        for(int i=1;i<LOG;++i){
            up[b][i] = up[up[b][i-1]][i-1];
        }
        dfs(b,a);
        n_subtree[a] += n_subtree[b];
        if(n_subtree[b] > hmax){
            hmax = n_subtree[b], bestSon = b;
        }
    }
    heavyChild[a] = bestSon;
}

void computeHeavyChains(int a, int e){
    if(heavyChild[a] != -1){
        chain[heavyChild[a]] = chain[a];
    }
    for(auto b:adj[a]){
        if(b == e)continue;
        computeHeavyChains(b, a);
    }
```

```
}

void setHeavyChains(int a, int e){
    SegTreePos[a] = timer++;
    if(heavyChild[a] != -1){
        setHeavyChains(heavyChild[a], a);
        int w = 0;
        for(auto b:adj[a]){
            if(b == heavyChild[a]){
                w = 1;
                break;
            }
        }
        //setting cost of heavyChild[a] = weight of edge (a,
             heavyChild[a])
        cost[heavyChild[a]] = w; //change when tree is weighted
    }
    for(auto b:adj[a]){
        if(b == e || heavyChild[a] == b)continue;
        cost[b] = 1; //change when tree is weighted
        setHeavyChains(b ,a);
    }
}

void buildSegTree(int n_node){
    seg_tree_size = n_node;
    while(__builtin_popcount(seg_tree_size)!=1)++seg_tree_size;
    for(int i=0;i<n_node;++i){
        update(1,0,seg_tree_size-1,SegTreePos[i], cost[i]);
    }
    for(int i=seg_tree_size-1;i>=1;--i){
        tree[i] = max(tree[2*i], tree[2*i+1]);
    }
}

ll queryPath(int mother, int son){
    ll ans = LONG_MIN;
    while(chain[mother] != chain[son]){
        if(chain[son] == son){
            ans = max(ans,f(1,0,seg_tree_size-1,SegTreePos[son
                 ],SegTreePos[son]));
        } else {
            ans = max(ans, f(1,0,seg_tree_size-1,SegTreePos[
                 chain[son]],SegTreePos[son]));
        }
        son = parent[chain[son]];
    }

    ans = max(ans, f(1,0,seg_tree_size-1,SegTreePos[mother]+1,
         SegTreePos[son]));

    return ans;
}

int lca(int a, int b){
    if(depth[a]<depth[b])swap(a,b);
    int k = depth[a] - depth[b];
    for(int i=LOG-1;i>=0;--i){
        if(k &(1<<i)){
            a = up[a][i];
        }
    }

    if(a == b)return a;
    for(int i=LOG-1;i>=0;--i){
        if(up[a][i] != up[b][i]){
            a = up[a][i];
            b = up[b][i];
        }
    }
    return up[a][0];
}

ll query(int x, int y){
    int l = lca(x,y);
    return max( queryPath(l, x), queryPath(l, y)); //combine
         the answer

}

void hld_init(int n){
    dfs(0,-1);
    computeHeavyChains(0,-1);
    setHeavyChains(0,-1);
    buildSegTree(n);
}

/*hld*/
```

lca.h
Description: TODO
0ae36f, 36 lines

```
//TODO: initialize tree(adj list)
const int LOG = 20;
int dep[N], parent[N];
int up[N][LOG]; // 2^j-th ancestor of n

void dfs(int a,int e){
    for(auto b:adj[a]){
        if(b == e)continue;
        dep[b] = dep[a] + 1;
        parent[b] = a;
        up[b][0] = parent[b];
        for(int i=1;i<LOG;++i){
            up[b][i] = up[up[b][i-1]][i-1];
        }
        dfs(b,a);
    }
}

int lca(int a, int b){
    if(dep[a]<dep[b])swap(a,b);
    int k = dep[a] - dep[b];
    for(int i=LOG-1;i>=0;--i){
        if(k &(1<<i)){
            a = up[a][i];
        }
    }

    if(a == b)return a;
    for(int i=LOG-1;i>=0;--i){
        if(up[a][i] != up[b][i]){
            a = up[a][i];
            b = up[b][i];
        }
    }
    return up[a][0];
}
```

kth-ancestor.h
Description: TODO
3b29e6, 9 lines

```
int kth_ancestor(int node,int k){
    if(dep[node] < k)return -1;
    for(int i = 0;i < LOG; ++i){
        if(k & (1<<i)){
            node = up[node][i];
        }
    }
    return node;
}
```

tree-hashing.h
Description: TODO
ce0886, 22 lines

```
/*
    very acurate rooted tree hashing
    don't forget to check tree invariant such as tree size and
         # of centroid
*/
map<vector<int>, int> hasher;

int hashify(vector<int> x) {
    sort(x.begin(), x.end());
    if(!hasher[x]) {
        hasher[x] = hasher.size();
    }
    return hasher[x];
}

int _hash(int v, int e, vector<vector<int>>&adj) { // get a "
     hash" of v's subtree, e is parent vertex
    vector<int> children;
    for(int u: adj[v]) {
        if( u == e )continue;
        children.push_back(_hash(u, v, adj));
    }
    return hashify(children);
}
```

1.8 String Processing

AC-automaton.h
Description: TODO
a41621, 53 lines

```
const int K = 26;

struct Vertex {
    int next[K];
    bool output = false;
    int p = -1;
    char pch;
    int link = -1;
    int go[K];

    Vertex(int p=-1, char ch='$') : p(p), pch(ch) {
        fill(begin(next), end(next), -1);
        fill(begin(go), end(go), -1);
    }
};

vector<Vertex> t(1);

void add_string(string const& s) {
    int v = 0;
    for (char ch : s) {
        int c = ch - 'a';
        if (t[v].next[c] == -1) {
            t[v].next[c] = t.size();
            t.emplace_back(v, ch);
        }
        v = t[v].next[c];
    }
    t[v].output = true;
}

int go(int v, char ch);
```

```
int get_link(int v) {
    if (t[v].link == -1) {
        if (v == 0 || t[v].p == 0)
            t[v].link = 0;
        else
            t[v].link = go(get_link(t[v].p), t[v].pch);
    }
    return t[v].link;
}

int go(int v, char ch) {
    int c = ch - 'a';
    if (t[v].go[c] == -1) {
        if (t[v].next[c] != -1)
            t[v].go[c] = t[v].next[c];
        else
            t[v].go[c] = v == 0 ? 0 : go(get_link(v), ch);
    }
    return t[v].go[c];
}
```

kmp.h

Description: TODO

4dfee5, 37 lines

```
int b[N];
int cnt = 0;

void knp_proc(string t,string p){
    int i=0,j=-1;b[0] = -1;
    while(i<(int)p.length()){
        while(j>=0 && p[i]!=p[j]) j =b[j];
        ++i;++j;
        b[i] = j;
    }
}

void knp_search(string t,string p){//count number of occurence
     of p in t
    int i=0,j=0;
    while(i<(int)t.length()){
        while(j>=0 && t[i] != p[j]) j = b[j];
        ++i;++j;
        if(j==(int)p.length()){
            ++cnt;
            j = b[j];
        }
    }
}

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;
}
```

z-function.h

Description: TODO

9a2512, 18 lines

```
vector<int> z_function(string s) {
    int n = s.size();
    vector<int> z(n);
    int l = 0, r = 0;
    for(int i = 1; i < n; i++) {
        if(i < r) {
            z[i] = min(r - i, z[i - l]);
        }
        while(i + z[i] < n && s[z[i]] == s[i + z[i]]) {
            z[i]++;
        }
        if(i + z[i] > r) {
            l = i;
            r = i + z[i];
        }
    }
    return z;
}
```

manacher.h

Description: TODO

3069fe, 37 lines

```
//sub-palindrome queries
vector<int> manacher_odd(string s) {
    int n = s.size();
    s = "$" + s + "^";
    vector<int> p(n + 2);
    int l = 1, r = 1;
    for(int i = 1; i <= n; i++) {
        p[i] = max(0, min(r - i, p[l + (r - i)]));
        while(s[i - p[i]] == s[i + p[i]]) {
            p[i]++;
        }
        if(i + p[i] > r) {
            l = i - p[i], r = i + p[i];
        }
    }
    return vector<int>(begin(p) + 1, end(p) - 1);
}

vector<int> manacher(string s) {
    string t;
    for(auto c: s) {
        t += string("#") + c;
    }
    auto res = manacher_odd(t + "#");
    return vector<int>(begin(res) + 1, end(res) - 1);
}

bool is_palindrome(int l, int r, vector<int> &v){
    if(v[l +r] < (r - l + 1))return false;
    return true;
}

vector<int> build_manacher(string s){ //0 base index string
    auto v = manacher(s);
    for(auto&x:v)--x;
    return v;
}
```

string-hashing.h

Description: TODO

ba27df, 17 lines

```
typedef __int128 HASH;
HASH p[NN], h[NN];
constexpr HASH M = 1000000000000000003;
mt19937 rng(chrono::steady_clock::now().time_since_epoch().
     count());//delete this if using my template
#define uid(a, b) uniform_int_distribution<long long>(a, b)(rng
     )//delete this if using my template

void compute_hash(string const& s){
    p[0] = 1, p[1] = uid(256, M-1);
    for(ll i=0;i<(int)s.length();++i){
        p[i+1] = p[i]*p[1]%M;
        h[i+1] = (h[i]*p[1] + s[i])%M;
    }
}

HASH sub_hash(ll l, ll r){ //range [l, r)
    return (h[r] - p[r-l]*h[l]%M + M)%M;
}
```

double-hashing-string.h

Description: TODO

276f01, 26 lines

```
//typedef
typedef long long ll;
typedef pair<ll,ll> pl;
#define M 1000000321
#define OP(x, y) pl operator x (pl a, pl b){return {a.first x b
     .first, (a.second y b.second) % M }; }
OP(+, +) OP(*, *) OP(-, + M -)
//random number generator
mt19937 gen(chrono::steady_clock::now().time_since_epoch().
     count());
uniform_int_distribution<ll> dist(256, M - 1);
//queries - check if S[i:i+l] == S[j:j+l](inclusive), S is a
     string, [:] is slice
#define H(i, l) (h[(i) + (l)] - h[i] * p[l])
#define EQ(i, j, l) (H(i, l) == H(j, l))
//preprocessing
const int N = 2e5;
string s;
pl p[N], h[N];
ll n;

int main(){
    cin >> n >> s;
    p[0] = {1,1}, p[1] = {dist(gen) | 1, dist(gen) };
    for(ll i=1;i<=(ll)s.length();++i){
        p[i] = p[i-1] * p[1];
        h[i] = h[i-1] * p[1] + make_pair(s[i-1], s[i-1]);
    }
}
```

suffix-array.h

Description: TODO

4ca004, 53 lines

```
vector<int> sort_cyclic_shifts(string const& s) {
    int n = s.size();
    const int alphabet = 256;

    vector<int> p(n), c(n), cnt(max(alphabet, n), 0);
    for (int i = 0; i < n; i++)
        cnt[s[i]]++;
    for (int i = 1; i < alphabet; i++)
        cnt[i] += cnt[i-1];
    for (int i = 0; i < n; i++)
        p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
        if (s[p[i]] != s[p[i-1]])
            classes++;
        c[p[i]] = classes - 1;
    }

    vector<int> pn(n), cn(n);
    for (int h = 0; (1 << h) < n; ++h) {
        for (int i = 0; i < n; i++) {
```

```
            pn[i] = p[i] - (1 << h);
            if (pn[i] < 0)
                pn[i] += n;
        }
        fill(cnt.begin(), cnt.begin() + classes, 0);
        for (int i = 0; i < n; i++)
            cnt[c[pn[i]]]++;
        for (int i = 1; i < classes; i++)
            cnt[i] += cnt[i-1];
        for (int i = n-1; i >= 0; i--)
            p[--cnt[c[pn[i]]]] = pn[i];
        cn[p[0]] = 0;
        classes = 1;
        for (int i = 1; i < n; i++) {
            pair<int, int> cur = {c[p[i]], c[(p[i] + (1 << h))
                % n]};
            pair<int, int> prev = {c[p[i-1]], c[(p[i-1] + (1 <<
                h)) % n]};
            if (cur != prev)
                ++classes;
            cn[p[i]] = classes - 1;
        }
        c.swap(cn);
    }
    return p;
}

vector<int> suffix_array_construction(string s) {
    s += "$";
    vector<int> sorted_shifts = sort_cyclic_shifts(s);
    sorted_shifts.erase(sorted_shifts.begin());
    return sorted_shifts;
}
```

suffix-automaton.h

Description: TODO

fe30a6, 52 lines

```
struct state {
    int len, link;
    map<char, int> next;
    map<char, int> go;
};

const int MAXLEN = 100001; //this one is 10^5 becareful
state st[MAXLEN * 2];
int sz, last;

void sa_init() {
    st[0].len = 0;
    st[0].link = -1;
    sz++;
    last = 0;
}

void sa_extend(char c) {
    int cur = sz++;
    st[cur].len = st[last].len + 1;
    int p = last;
    while (p != -1 && !st[p].next.count(c)) {
        st[p].next[c] = cur;
        p = st[p].link;
    }
    if (p == -1) {
        st[cur].link = 0;
    } else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len) {
            st[cur].link = q;
        } else {
            int clone = sz++;
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            while (p != -1 && st[p].next[c] == q) {
                st[p].next[c] = clone;
                p = st[p].link;
            }
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}

int sa_go(int v, char c) {
    if (st[v].go.count(c)) return st[v].go[c];
    if (st[v].next.count(c)) return st[v].go[c] = st[v].next[c
        ];
    if (st[v].link == -1) return st[v].go[c] = 0; // fallback
        to root
    return st[v].go[c] = sa_go(st[v].link, c);
}
```

trie.h

Description: TODO

e121b2, 26 lines

```
// string trie structure
const int K = 26;

struct node{
    int to[K];
    bool output = false;

    node(){
        fill(begin(to), end(to), -1);
    }
};

vector<node>trie(1);

void add(const& string s){
    int v = 0;
    for(char ch: s){
        int c = ch - 'a';
        if(trie[v][c] == -1){
            trie[v][c] = trie.size();
            trie.push_back(node());
        }
        v = trie[v][c];
    }
    trie[v].output = true;
}
```

1.9 Computational Geometry

geometry.h

Description: TODO

963fd3, 167 lines

```
//Geometry
#pragma GCC target("avx2")
#pragma GCC optimize("O3")
#pragma GCC optimize("unroll-loops")

typedef long long int ll;
typedef long double ld;
typedef complex<ld> pt;
struct line {
  pt P, D; bool S = false;
  line(pt p, pt q, bool b = false) : P(p), D(q - p), S(b) {}
  line(pt p, ld th) : P(p), D(polar((ld)1, th)) {}
};
struct circ { pt C; ld R; };

#define X real()
#define Y imag()
#define CRS(a, b) (conj(a) * (b)).Y //scalar cross product
#define DOT(a, b) (conj(a) * (b)).X //dot product
#define U(p) ((p) / abs(p)) //unit vector in direction of p (
    don't use if Z(p) == true)
#define Z(x) (abs(x) < EPS)
#define A(a) (a).begin(), (a).end() //shortens sort(),
    upper_bound(), etc. for vectors

//constants (INF and EPS may need to be modified)
ld PI = acosl(-1), INF = 1e20, EPS = 1e-12;
pt I = {0, 1};

//true if d1 and d2 parallel (zero vectors considered parallel
    to everything)
bool parallel(pt d1, pt d2) { return Z(d1) || Z(d2) || Z(CRS(U(
    d1), U(d2))); }

//"above" here means if l & p are rotated such that l.D points
    in the +x direction, then p is above l. Returns arbitrary
    boolean if p is on l
bool above_line(pt p, line l) { return CRS(p - l.P, l.D) > 0; }

//true if p is on line l
bool on_line(pt p, line l) { return parallel(l.P - p, l.D) &&
    (!l.S || DOT(l.P - p, l.P + l.D - p) <= EPS); }

//returns 0 for no intersection, 2 for infinite intersections,
    1 otherwise. p holds intersection pt
ll intsct(line l1, line l2, pt& p) {
  if(parallel(l1.D, l2.D)) //note that two parallel segments
      sharing one endpoint are considered to have infinite
      intersections here
    return 2 * (on_line(l1.P, l2) || on_line(l1.P + l1.D, l2)
        || on_line(l2.P, l1) || on_line(l2.P + l2.D, l1));
  pt q = l1.P + l1.D * CRS(l2.D, l2.P - l1.P) / CRS(l2.D, l1.D)
      ;
  if(on_line(q, l1) && on_line(q, l2)) { p = q; return 1; }
  return 0;
}

//closest pt on l to p
pt cl_pt_on_l(pt p, line l) {
  pt q = l.P + DOT(U(l.D), p - l.P) * U(l.D);
  if(on_line(q, l)) return q;
  return abs(p - l.P) < abs(p - l.P - l.D) ? l.P : l.P + l.D;
}

//distance from p to l
ld dist_to(pt p, line l) { return abs(p - cl_pt_on_l(p, l)); }

//p reflected over l
pt refl_pt(pt p, line l) { return conj((p - l.P) / U(l.D)) * U(
    l.D) + l.P; }

//ray r reflected off l (if no intersection, returns original
    ray)
line reflect_line(line r, line l) {
  pt p; if(intsct(r, l, p) - 1) return r;
  return line(p, p + INF * (p - refl_pt(r.P, l)), 1);
}

//altitude from p to l
line alt(pt p, line l) { l.S = 0; return line(p, cl_pt_on_l(p,
    l)); }
```

```
//angle bisector of angle abc
line ang_bis(pt a, pt b, pt c) { return line(b, b + INF * (U(a
    - b) + U(c - b)), 1); }

//perpendicular bisector of l (assumes l.S == 1)
line perp_bis(line l) { return line(l.P + l.D / (ld)2, arg(l.D
    * I)); }

//orthocenter of triangle abc
pt orthocent(pt a, pt b, pt c) { pt p; intsct(alt(a, line(b, c)
    ), alt(b, line(a, c)), p); return p; }

//incircle of triangle abc
circ incirc(pt a, pt b, pt c) {
  pt cent; intsct(ang_bis(a, b, c), ang_bis(b, a, c), cent);
  return {cent, dist_to(cent, line(a, b))};
}

//circumcircle of triangle abc
circ circumcirc(pt a, pt b, pt c) {
  pt cent; intsct(perp_bis(line(a, b, 1)), perp_bis(line(a, c,
      1)), cent);
  return {cent, abs(cent - a)};
}

//is pt p inside the (not necessarily convex) polygon given by
    poly
bool in_poly(pt p, vector<pt>& poly) {
  line l = line(p, {INF, INF * PI}, 1);
  bool ans = false;
  pt lst = poly.back(), tmp;
  for(pt q : poly) {
    line s = line(q, lst, 1); lst = q;
    if(on_line(p, s)) return false; //change if border included
    else if(intsct(l, s, tmp)) ans = !ans;
  }
  return ans;
}

//area of polygon, vertices in order (cw or ccw)
ld area(vector<pt>& poly) {
  ld ans = 0;
  pt lst = poly.back();
  for(pt p : poly) ans += CRS(lst, p), lst = p;
  return abs(ans / 2);
}

//perimeter of polygon, vertices in order (cw or ccw)
ld perim(vector<pt>& poly) {
  ld ans = 0;
  pt lst = poly.back();
  for(pt p : poly) ans += abs(lst - p), lst = p;
  return ans;
}

//centroid of polygon, vertices in order (cw or ccw)
pt centroid(vector<pt>& poly) {
  ld area = 0;
  pt lst = poly.back(), ans = {0, 0};
  for(pt p : poly) {
    area += CRS(lst, p);
    ans += CRS(lst, p) * (lst + p) / (ld)3;
    lst = p;
  }
  return ans / area;
}
```

```
//invert a point over a circle (doesn't work for center of
    circle)
pt circInv(pt p, circ c) {
    return c.R * c.R / conj(p - c.C) + c.C;
}

//vector of intersection pts of two circs (up to 2) (if circles
    same, returns empty vector)
vector<pt> intsctCC(circ c1, circ c2) {
  if(c1.R < c2.R) swap(c1, c2);
  pt d = c2.C - c1.C;
  if(Z(abs(d) - c1.R - c2.R)) return {c1.C + polar(c1.R, arg(c2
      .C - c1.C))};
  if(!Z(d) && Z(abs(d) - c1.R + c2.R)) return {c1.C + c1.R * U(
      d)};
  if(abs(abs(d) - c1.R) >= c2.R - EPS) return {};
  ld th = acosl((c1.R * c1.R + norm(d) - c2.R * c2.R) / (2 * c1
      .R * abs(d)));
  return {c1.C + polar(c1.R, arg(d) + th), c1.C + polar(c1.R,
      arg(d) - th)};
}

//vector of intersection pts of a line and a circ (up to 2)
vector<pt> intsctCL(circ c, line l) {
  vector<pt> v, ans;
  if(parallel(l.D, c.C - l.P)) v = {c.C + c.R * U(l.D), c.C - c
      .R * U(l.D)};
  else v = intsctCC(c, circ{refl_pt(c.C, l), c.R});
  for(pt p : v) if(on_line(p, l)) ans.push_back(p);
  return ans;
}

//external tangents of two circles (negate c2.R for internal
    tangents)
vector<line> circTangents(circ c1, circ c2) {
  pt d = c2.C - c1.C;
  ld dr = c1.R - c2.R, d2 = norm(d), h2 = d2 - dr * dr;
  if(Z(d2) || h2 < 0) return {};
  vector<line> ans;
  for(ld sg : {-1, 1}) {
    pt u = (d * dr + d * I * sqrt(h2) * sg) / d2;
    ans.push_back(line(c1.C + u * c1.R, c2.C + u * c2.R, 1));
  }
  if(Z(h2)) ans.pop_back();
  return ans;
}
```

closest-pair-of-points.h

Description: TODO

85eeeb, 67 lines

```
struct pt {
    int x, y, id;
};

struct cmp_x {
    bool operator()(const pt & a, const pt & b) const {
        return a.x < b.x || (a.x == b.x && a.y < b.y);
    }
};

struct cmp_y {
    bool operator()(const pt & a, const pt & b) const {
        return a.y < b.y;
    }
};

int n;
vector<pt> a;
double mindist;
pair<int, int> best_pair;

void upd_ans(const pt & a, const pt & b) {
    double dist = sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a
        .y - b.y));
    if (dist < mindist) {
        mindist = dist;
        best_pair = {a.id, b.id};
    }
}

vector<pt> t;

void rec(int l, int r) {
    if (r - l <= 3) {
        for (int i = l; i < r; ++i) {
            for (int j = i + 1; j < r; ++j) {
                upd_ans(a[i], a[j]);
            }
        }
        sort(a.begin() + l, a.begin() + r, cmp_y());
        return;
    }

    int m = (l + r) >> 1;
    int midx = a[m].x;
    rec(l, m);
    rec(m, r);

    merge(a.begin() + l, a.begin() + m, a.begin() + m, a.begin
        () + r, t.begin(), cmp_y());
    copy(t.begin(), t.begin() + r - l, a.begin() + l);

    int tsz = 0;
    for (int i = l; i < r; ++i) {
        if (abs(a[i].x - midx) < mindist) {
            for (int j = tsz - 1; j >= 0 && a[i].y - t[j].y <
                mindist; --j)
                upd_ans(a[i], t[j]);
            t[tsz++] = a[i];
        }
    }
}

void call_rec(){
    t.resize(n);
    sort(a.begin(), a.end(), cmp_x());
    mindist = 1E20;
    rec(0, n);
}
```

1.10 Advanced Algorithms

fft.h

Description: TODO

cbf4b6, 63 lines

```
//from cp algorithm the inplace—computation fft part
using cd = complex<double>;
const double PI = acos(-1);

int reverse(int num, int lg_n) {
    int res = 0;
    for (int i = 0; i < lg_n; i++) {
        if (num & (1 << i))
            res |= 1 << (lg_n - 1 - i);
    }
    return res;
}
```

```
void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    int lg_n = 0;
    while ((1 << lg_n) < n)
        lg_n++;

    for (int i = 0; i < n; i++) {
        if (i < reverse(i, lg_n))
            swap(a[i], a[reverse(i, lg_n)]);
    }

    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, vector<int> const& b
    ) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1;
    while (n < (int)a.size() + (int)b.size())
        n <<= 1;
    fa.resize(n);
    fb.resize(n);

    fft(fa, false);
    fft(fb, false);
    for (int i = 0; i < n; i++)
        fa[i] *= fb[i];
    fft(fa, true);

    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

mo2D.h

Description: TODO

9df2fb, 49 lines

```
/* always use zero based indexing*/
/* there N/B blocks so left move ~ N/B*B + Q*B, right move ~ N/
    B*B + (N/B)*B */
/* so O(N^2/B + N + QB) ~~ O((N+Q)sqrt(N)) when B = sqrt(N) */
/* when Q~N it's ~ O(N^3/2) */
void remove(int idx);  // TODO: remove value at idx from data
    structure
void add(int idx);     // TODO: add value at idx from data
    structure
int get_answer();  // TODO: extract the current answer of the
    data structure

int block_size;

struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query>& queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the range
        [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}
```

mo3D.h

Description: TODO

dc4a5e, 62 lines

```
/* always use zero based indexing*/
/* ~ O(N^(5/3)) */
int block_size;

struct Query {
    int l, r, t, idx;
    bool operator<(Query other) const
    {
        return make_tuple(l / block_size, r / block_size, t) <
               make_tuple(other.l / block_size, other.r /
                    block_size,other.t);
    }
};

int ans[N];
void mo_s_algorithm(vector<Query>& queries) {
    sort(queries.begin(), queries.end());
    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    int cur_t = -1;
    // invariant: data structure will always reflect the range
        [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            //add(cur_l)
        }
        while (cur_r < q.r) {
            cur_r++;
            //add(cur_r)
        }
        while (cur_l < q.l) {
            //remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            //remove(cur_r);
            cur_r--;
        }

        while(cur_t < q.t){
            cur_t++;
            //update
        }
        while(cur_t > q.t){
            //update rollback
            cur_t--;
        }
        ans[q.idx] = get_answer(); // declare array for this
    }
}

void getBlockSize(int n, int q){
    if (q >= n) {
        // most common: Q ~ N
        block_size = max(1, (int)pow(n, 2.0/3.0));
    } else {
        // general cube-root rule
        block_size = max(1, (int)pow((long double)n*n / q,
             1.0/3.0));
    }
    // then sort your queries by (L/B, R/B, T) or via Hilbert
}
```

divide-and-conquer-dp.h

Description: TODO

108b88, 55 lines

```
//TODO: initialize dp , initialize cost() function of use slide
    ()
//ll dp[N][M];

//generic implementation for sliding range technique for logn
    cost()
//(persistent segtree alternative)
ll ccost = 0;
int cl = 0, cr = -1;
void slide(int l, int r){
    while(cr < r){
        ++cr;
        //add();
        //...
    }
    while(cl > l){
        --cl;
        //add();
        //...
    }
    while(cr > r){
        //remove();
        //...
        --cr;
    }
```

```
    while(cl < l){
        //remove();
        //...
        ++cl;
    }
}

void compute(int l, int r, int optl, int optr, int j){
    if(l>r)return;

    int mid = (l+r)>>1;
    //pair<ll, int> best = {0,-1};
    //pair<ll, int> best = {LINF,-1};

    //dp is satisfy quadrangle IE if cost() satisfy qudrangle
        IE
    //if cost() is QF => opt() is nondecreasing
    for(int k=optl;k<=min(mid,optr);++k){
        slide(k,mid);
        //best = max(best, {((k>0)?dp[k-1][j-1]:0) + ccost, k }
            );
        //best = min(best, {((k>0)?dp[k-1][j-1]:0) + ccost, k }
            );
    }

    //dp[mid][j] = max(dp[mid][j], best.first);
    //dp[mid][j] = min(dp[mid][j], best.first);
    int opt = best.second;
    if(l!=r){
        compute(l,mid-1,optl,opt,j);
        compute(mid+1,r,opt,optr,j);
    }
}

//TODO: set dp to LINF or -LINF
```