Team notebook

CatsOnTrees

October 2, 2022

Contents

1 Data-Structures

1.1 mergeSortTree

```
const int maxn = 2e5 + 5;
vecl a(maxn);
vecl t[4*maxn];
void merge(vecl& temp1, vecl& temp2, vecl& final){
    int i = 0, j = 0;
    while(i < sz(temp1) && j < sz(temp2)){
        if(temp1[i] <= temp2[j]) {
            final.pb(temp1[i]);
            i++;
        } else {
            final.pb(temp2[j]);
            j++;
        }
    }
    while(i < sz(temp1)){
        final.pb(temp1[i]);
        i++;
    }
    while(j < sz(temp2)){
        final.pb(temp2[j]);
        j++;
    }
}
void build(int ind, int tl, int tr){
    if(tl == tr){
        t[ind].pb(a[tl]);
        return;
    }
    int tm = (tl + tr) / 2;
    build(2 * ind, tl, tm);
    build(2 * ind + 1, tm + 1, tr);
    merge(t[2 * ind], t[2 * ind + 1], t[ind]);
}
int query(int ind, int tl, int tr, int l, int r, int
    valuetoCompare){ // query for elements strictly greater than
    k
    if(l > r){
        return 0;
    }
    if(l == tl && r == tr){
        return t[ind].end() - upper_bound(all(t[ind]),
            valuetoCompare);
    }
    int tm = (tl + tr) / 2;
    return (query(2 * ind, tl, tm, l, min(r, tm),
        valuetoCompare) +
            query(2 * ind + 1, tm + 1, tr, max(l, tm + 1), r,
                valuetoCompare));
}
```

1.2 policybased

```
/*
find_by_order(k): return iterator to k'th element(counting from
    zero)
order_of_key(k) : number of items < k in O(logn) time.
*/
include <ext/pb_ds/assoc_container.hpp>
include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<int, null_type, less<int>,
rb_tree_tag, tree_order_statistics_node_update> ordered_set;
```

1.3 segmentTree

```
/*
Usage: Node* tr = new Node(v, 0, sz(v));
finds max in a range, [L, R)
*/
// Either globally or in a single class:
static char buf[450 << 20];
void* operator new(size_t s) {
        static size_t i = sizeof buf;
        assert(s < i);
        return (void*)&buf[i -= s];
}
void operator delete(void*) {}

const int inf = 1e9;
struct Node {
        Node *l = 0, *r = 0;
        ll lo, hi, mset = inf, madd = 0, val = -inf;
        Node(int lo,int hi):lo(lo),hi(hi){} // Large interval of -inf
        Node(vll& v, int lo, int hi) : lo(lo), hi(hi) {
                if (lo + 1 < hi) {
                        int mid = lo + (hi - lo)/2;
                        l = new Node(v, lo, mid); r = new Node(v, mid, hi);
                        val = max(l->val, r->val);
                }
                else val = v[lo];
        }
        ll query(int L, int R) {
                if (R <= lo || hi <= L) return -inf;
                if (L <= lo && hi <= R) return val;
                push();
                return max(l->query(L, R), r->query(L, R));
        }
        void set(int L, int R, ll x) {
                if (R <= lo || hi <= L) return;
                if (L <= lo && hi <= R) mset = val = x, madd = 0;
                else {
                        push(), l->set(L, R, x), r->set(L, R, x);
                        val = max(l->val, r->val);
                }
        }
        void add(int L, int R, ll x) {
                if (R <= lo || hi <= L) return;
                if (L <= lo && hi <= R) {
                        if (mset != inf) mset += x;
                        else madd += x;
                        val += x;
                }
                else {
                        push(), l->add(L, R, x), r->add(L, R, x);
                        val = max(l->val, r->val);
                }
        }
        void push() {
                if (!l) {
                        int mid = lo + (hi - lo)/2;
                        l = new Node(lo, mid); r = new Node(mid, hi);
                }
                if (mset != inf)
                        l->set(lo,hi,mset), r->set(lo,hi,mset), mset = inf;
                else if (madd)
                        l->add(lo,hi,madd), r->add(lo,hi,madd), madd = 0;
        }
};
```

1.4 treap

```
struct Node {
    char val;
    int weight, size;
    Node *left, *right;
    Node(char c) : val(c), weight(rand()), size(1), left(NULL),
        right(NULL) {}
} *root;

inline int size(Node *treap) {
    return treap ? treap->size : 0;
}
void split(Node *treap, Node *&left, Node *&right, int val) {
    if (!treap) {
        left = right = NULL;
        return;
    }
    if (size(treap->left) < val) {
        split(treap->right, treap->right, right, val -
            size(treap->left) - 1);
        left = treap;
    } else {
        split(treap->left, left, treap->left, val);
        right = treap;
    }
    treap->size = 1 + size(treap->left) + size(treap->right);
}
void merge(Node *&treap, Node *left, Node *right) {
    if (left == NULL) {
        treap = right;
        return;
    }
    if (right == NULL) {
        treap = left;
        return;
    }

    if (left->weight < right->weight) {
        merge(left->right, left->right, right);
        treap = left;
    } else {
        merge(right->left, left, right->left);
        treap = right;
    }
    treap->size = 1 + size(treap->left) + size(treap->right);
}
ostream& operator<<(ostream &os, Node *n) {
    if (!n) return os;
    os << n->left; os << n->val; os << n->right;
    return os;
}
void solve() { // USAGE:
    // get integer n, q, and s
    for(auto c: s) merge(root, root, new Node(c));
    while(q--) {
        int l, r; cin >> l >> r;
        Node *a, *b;
        split(root, a, b, l - 1);
        Node *c, *d;
        split(b, c, d, r - l + 1);
        merge(root, a, d);
        merge(root, root, c);
    }
    cout << root << nl;
}
```

1.5 treapLazy

```
struct Node {
    char val;
    int weight, size;
    Node *left, *right;
    int toinvert;
    Node(char c) : val(c), weight(rand()), size(1), left(NULL),
        right(NULL), toinvert(0) {}
} *root;
inline int size(Node *treap) {
    return treap ? treap->size : 0;
}
void push(Node *treap) {
    if(treap == NULL) return;
    if(treap->toinvert == 0) return;
    Node *temp = treap->left;
    treap->left = treap->right;
    treap->right = temp;
    treap->toinvert = 0;
    if(treap->left != NULL) treap->left->toinvert ^= 1;
    if(treap->right != NULL) treap->right->toinvert ^= 1;
}
void split(Node *treap, Node *&left, Node *&right, int val) {
    if (!treap) {
        left = right = NULL; return;
    }
    push(treap);
    if (size(treap->left) < val) {
        split(treap->right, treap->right, right, val -
            size(treap->left) - 1);
        left = treap;
    } else {
        split(treap->left, left, treap->left, val);
        right = treap;
    }
    treap->size = 1 + size(treap->left) + size(treap->right);
}
void merge(Node *&treap, Node *left, Node *right) {
    if (left == NULL)
        treap = right; return;
    if (right == NULL)
        treap = left; return;
    push(left); push(right);
    if (left->weight < right->weight) {
        merge(left->right, left->right, right); treap = left;
    } else {
        merge(right->left, left, right->left); treap = right;
    }
    treap->size = 1 + size(treap->left) + size(treap->right);
}
void solve() { //USAGE:
    //get integers n,q and string s
    for(auto c: s) merge(root, root, new Node(c));
    while(q--) {
        int l, r; cin >> l >> r;
        Node *a, *b;
        split(root, a, b, l - 1);
        Node *c, *d;
        split(b, c, d, r - l + 1);
        c->toinvert ^= 1;
        merge(root, a, c);
        merge(root, root, d);
    }
    cout << root << nl;
}
```

2 DP-Optimizations

2.1 convexhull

```
const ll is_query = -(1LL<<62);
struct line {
        ll m, b;
        mutable function<const line*()> succ;
        bool operator<(const line& rhs) const {
                if (rhs.b != is_query) return m < rhs.m;
                const line* s = succ();
                if (!s) return 0;
                ll x = rhs.m;
                return b - s->b < (s->m - m) * x;
        }
};
struct dynamic_hull : public multiset<line> { // will maintain
    upper hull for maximum
        const ll inf = LLONG_MAX;
        bool bad(iterator y) {
                auto z = next(y);
                if (y == begin()) {
                        if (z == end()) return 0;
                        return y->m == z->m && y->b <= z->b;
                }
                auto x = prev(y);
                if (z == end()) return y->m == x->m && y->b <=
                    x->b;

                /* compare two lines by slope, make sure
                    denominator is not 0 */
                ll v1 = (x->b - y->b);
                if (y->m == x->m) v1 = x->b > y->b ? inf : -inf;
                else v1 /= (y->m - x->m);
                ll v2 = (y->b - z->b);
                if (z->m == y->m) v2 = y->b > z->b ? inf : -inf;
                else v2 /= (z->m - y->m);
                return v1 >= v2;
        }
        void insert_line(ll m, ll b) {
                auto y = insert({ m, b });
                y->succ = [=] { return next(y) == end() ? 0 :
                    &*next(y); };
                if (bad(y)) { erase(y); return; }
                while (next(y) != end() && bad(next(y)))
                    erase(next(y));
                while (y != begin() && bad(prev(y)))
                    erase(prev(y));
        }
        ll eval(ll x) { //maximum at point x
                auto l = *lower_bound((line) { x, is_query });
                return l.m * x + l.b;
        }
};
```

2.2 divideNconquer

```
//dp[i][j]  = min {dp[ i - 1 ][k]  + C  [k][j]} for all k < j,
//and optk[i][j] <= optk[i][j+1]
//optk is optimial k that gives you answer.
// compute dp_cur[l], ... dp_cur[r] (inclusive)
// C(a,c) + C(b,d) <= C(a,d) + C(b,c) for all a<=b<=c<=d
void compute(int l, int r, int optl, int optr) {
    if (l > r)
        return;

    int mid = (l + r) >> 1;
    pair<long long, int> best = {LLONG_MAX, -1};
    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best, {(k ? dp_before[k - 1] : 0) + C(k, mid),
            k});
    }

    dp_cur[mid] = best.first;
    int opt = best.second;

    compute(l, mid - 1, optl, opt);
    compute(mid + 1, r, opt, optr);
}
// notebook-generator ./ --author "CatsOnTrees" --initials UTP
    --size 11 --columns 3 --paper a4paper
```

2.3 knuth

```
//dp[i][j] = min { dp[i][k] + dp[k][j] + C(i, j) } : for i <= k
    < j
// O(n^3) -> O(n^2) if optk(i,j-1) <= opt(i,j) <= opt(i+1,j)
// criteria to see:
/* for a<=b<=c<=d
1) C(b,c) <= C(a,d)
2) C(a,c) + C(b,d) <= C(a,d) + C(b,c)
eg: C(i,j) = cost of arr[i..j] if all elements +ve */
int solve() {
    int N;
    ... // read N and input
    int dp[N][N], opt[N][N];

    auto C = [&](int i, int j) {
        ... // Implement cost function C.
    };

    for (int i = 0; i < N; i++) {
        opt[i][i] = i;
        ... // Initialize dp[i][i] according to the problem
    }

    for (int i = N-2; i >= 0; i--) {
        for (int j = i+1; j < N; j++) {
            int mn = INT_MAX;
            int cost = C(i, j);
            for (int k = opt[i][j-1]; k <= min(j-1, opt[i+1][j]);
                k++) {
                if (mn >= dp[i][k] + dp[k+1][j] + cost) {
                    opt[i][j] = k;
                    mn = dp[i][k] + dp[k+1][j] + cost;
                }
            }
            dp[i][j] = mn;
        }
    }

    cout << dp[0][N-1] << endl;
}
```

2.4 matrixExponentiation

```
vector<vector<int>> indentity(int n) {
    vector<vector<int>> i(n, vector<int>(n, 0));
    for(int j = 0; j < n; j++) {
        i[j][j] = 1;
    }
    return i;
}
ll mod_mul(ll a, ll b){
    a = a%M; b = b%M;
    return ((a*b)%M + M)%M;
}
vector<vector<int>> mul(vector<vector<int>>& a,
    vector<vector<int>>& b, int mod) {
    int n = sz(a);
    vector<vector<int>> toreturn(n, vector<int>(n, 0));
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < n; j++) {
            int ans = 0;
            for(int k = 0; k < n; k++) {
                ans += mod_mul(a[i][k] , b[k][j]);
                ans = ans%mod;
            }
            toreturn[i][j] = ans;
        }
    }
    return toreturn;
}
vector<vector<int>> expo(vector<vector<int>> &mat, int pow, int
    mod) {
    int n = mat.size();
    if(pow == 0) return indentity(n);
    vector<vector<int>> temp = indentity(n);
    auto Exp = expo(mat, pow/2, mod);
    if(pow % 2) {
        temp = mul(temp, mat, mod);
        vector<vector<int>> result = Exp;
        result = mul(result, result, mod);
        temp = mul(temp, result, mod);
    } else {
        vector<vector<int>> result = Exp;
        result = mul(result, result, mod);
        temp = mul(temp, result, mod);
    }
    return temp;
}
```

2.5 sosDP

```
for(int i = 0; i<(1<<N); ++i)F[i] = A[i];
for(int i = 0;i < N; ++i) for(int mask = 0; mask < (1<<N);
    ++mask){
        if(mask & (1<<i)) F[mask] += F[mask^(1<<i)];
}
```

3 Extras

3.1 customHash

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        // http://xorshift.di.unimi.it/splitmix64.c
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }

    size_t operator()(uint64_t x) const {
        static const uint64_t FIXED_RANDOM =
            chrono::steady_clock::now().time_since_epoch().count();
        return splitmix64(x + FIXED_RANDOM);
    }
};
```

3.2 random rng

```
  //include <random> and <chrono>
```

```
 mt19937
    rng(chrono::steady_clock::now().time_since_epoch().count());
//  mt19937 rng((uint64_t) new char);
```

3.3 submaskOfBitmask

```
for (int m=0; m<(1<<n); ++m)
    for (int s=m; s; s=(s-1)&m)
 ... s and m ...
```

4 Graphs

4.1 2sat

```
/*
(x    y)     ( x     y) so add edge between ~x ---> y and ~y
   ---> x
id[x] < id[x] => x = false
Both variables must have the same value is equivalent to:
( x     y)  (x     y ).
*/
struct two_sat {
    int n;
    vector<vector<int>> g, gr; // gr is the reversed graph
    vector<int> comp, topological_order, answer; // comp[v]: ID
        of the SCC containing node v
    vector<bool> vis;
    two_sat() {}
    two_sat(int _n) { init(_n); }
    void init(int _n) {
        n = _n;
        g.assign(2 * n, vector<int>());
        gr.assign(2 * n, vector<int>());
        comp.resize(2 * n);
        vis.resize(2 * n);
        answer.resize(2 * n);
    }
    void add_edge(int u, int v) {
        g[u].push_back(v);
        gr[v].push_back(u);
    }
    // For the following three functions
    // int x, bool val: if 'val' is true, we take the variable
        to be x. Otherwise we take it to be x's complement.
    // At least one of them is true
    void add_clause_or(int i, bool f, int j, bool g) {
        add_edge(i + (f ? n : 0), j + (g ? 0 : n));
        add_edge(j + (g ? n : 0), i + (f ? 0 : n));
    }
    // Only one of them is true
    void add_clause_xor(int i, bool f, int j, bool g) {
        add_clause_or(i, f, j, g);
        add_clause_or(i, !f, j, !g);
    }
    // Both of them have the same value
    void add_clause_and(int i, bool f, int j, bool g) {
        add_clause_xor(i, !f, j, g);
    }
    // Topological sort
    void dfs(int u) {
        vis[u] = true;
        for (const auto &v : g[u])
            if (!vis[v]) dfs(v);
        topological_order.push_back(u);
    }
    // Extracting strongly connected components
    void scc(int u, int id) {
        vis[u] = true;
        comp[u] = id;
        for (const auto &v : gr[u])
            if (!vis[v]) scc(v, id);
    }
    // Returns true if the given proposition is satisfiable and
        constructs a valid assignment
    bool satisfiable() {
        fill(vis.begin(), vis.end(), false);
        for (int i = 0; i < 2 * n; i++)
            if (!vis[i]) dfs(i);
        fill(vis.begin(), vis.end(), false);
        reverse(topological_order.begin(),
            topological_order.end());
        int id = 0;
        for (const auto &v : topological_order)
            if (!vis[v]) scc(v, id++);
        // Constructing the answer
        for (int i = 0; i < n; i++) {
            if (comp[i] == comp[i + n]) return false;
            answer[i] = (comp[i] > comp[i + n] ? 1 : 0);
        }
        return true;
    }
};
```

4.2 dijkstra

```
vector<ll> dist;
vector<bool> vis;
//Single source shortest path algorithm
void dijkstra(vvpll& graph, ll start){
        int n = graph.size();
        vis.assign(n,false);
        dist.assign(n,1e18);
        //priority queue stores distance , current
        priority_queue<pair<ll,ll>,vpll,greater<pair<ll,ll>>> peq;
        dist[start] = 0;
        vis[start] = 0;
        peq.push(MP(0,start));
        while(!peq.empty()){
                ll curr = peq.top().S;
                ll currdist = peq.top().F;
                peq.pop();
                if(vis[curr]) continue;
                vis[curr] = true;
                //update all the children
                for(auto cpx: graph[curr]){
                        if(vis[cpx.F]) continue;
                        ll newDist = currdist+cpx.S;
                        //relaxation
                        if(dist[cpx.F] > newDist){
                                dist[cpx.F] = newDist;
                                peq.push(MP(newDist, cpx.F));
                        }
                }
        }
}
```

4.3 dinics

```
struct Dinics{
        struct Edge{
                int to, revidx;
                ll cap, ocap;
                Edge(int to, int revidx, ll cap, ll ocap):to(to),
                    revidx(revidx), cap(cap), ocap(ocap){}
        Edge(){}
        ll flow(){
            return max(ocap - cap, 0ll);
        }
    };
    vector<vector<Edge>> adj;
    vector<int> level;
    vector<int> next;
    int n;
    Dinics(int n):n(n){
        level.assign(n, 0), next.assign(n,0);
        adj.assign(n, vector<Edge>(0));
    }
    void addEdge(int u, int v, ll cap, ll rev = 0){
        adj[u].push_back(Edge(v, adj[v].size(), cap, cap));
        adj[v].push_back(Edge(u, adj[u].size() - 1, rev, rev));
    }
    ll dfs(int curr, int t, ll flow){
        // cout<<curr<<" "<<t<<" "<<flow<<"\n";
        if(curr == t || !flow) return flow;
        for(int& i = next[curr]; i < adj[curr].size(); i++){
            Edge &edge = adj[curr][i];
            if(level[edge.to] != level[curr]+1) continue;
            ll actualflow;
            actualflow = dfs(edge.to, t, min(flow, edge.cap));
            if(actualflow){
                edge.cap -= actualflow;
                adj[edge.to][edge.revidx].cap += actualflow;
                return actualflow;
            }
        }
        return 0;
    }
    ll calc(int src, int t){
        ll flow = 0;
        const ll inf = 1e16;
        //capacity scaling
        for(int L = 30; L >= 0; L--){
            do{
                level.assign(n, 0);
                next.assign(n, 0);
                //level assignment
                queue<int> q;
                level[src] = 1;
                q.push(src);
                while(!q.empty() && !level[t]){
                    int curr = q.front(); q.pop();
                    for(int i = 0; i < adj[curr].size(); i++){
                        Edge &e = adj[curr][i];
                        if(!level[e.to] && (e.cap >> L)){
                            level[e.to] = level[curr] + 1;
                            q.push(e.to);
                        }
                    }
                }
                //flows
                ll curflow;
                while(curflow = dfs(src, t, inf)) flow += curflow;
```

```
                        } while(level[t] != 0);
                }
                return flow;
        }
};
```

5 Maths

5.1 crt

```
/**
 * Find z such that z % x[i] = a[i] for all i.
 * */
long long crt(vector<long long> &a, vector<long long> &x) {
  long long z = 0;
  long long n = 1;
  for (int i = 0; i < x.size(); ++i)
    n *= x[i];
  for (int i = 0; i < a.size(); ++i) {
    long long tmp = (a[i] * (n / x[i])) % n;
    tmp = (tmp * mod_inv(n / x[i], x[i])) % n;
    z = (z + tmp) % n;
  }
  return (z + n) % n;
}
```

5.2 eulerToitientNlog(logn))

```
void phi_1_to_n(int n) {
    vector<int> phi(n + 1);
    for (int i = 0; i <= n; i++)
        phi[i] = i;

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

5.3 eulerToitientRootn

```
// counts the number of integers between 1 and n inclusive, which are coprime to 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;
}
```

5.4 extendedgcd

```
int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}
int modinv(int a, int m){
        int x, y;
        int g = gcd(a, m, x, y);
        if(g != 1) return -100;
        else return (x%m + m)%m;
}
```

5.5 linearSieve

```
const int maxn = 100000;
int lp[maxn+1];
int mobius[maxn+1];
int twopow[maxn+1];
vector<int> primes(0);
void lsieve(){
        lp[1] = 1;
        mobius[1] = 1;
        for(int i = 2; i <= maxn; i++){
                if(lp[i] == 0){
                        lp[i] = i;
                        primes.push_back(i);
                }
                for(int j = 0; (j<primes.size()) &&
                    primes[j]<=lp[i] && i*primes[j] <= maxn; j++){
```

```
                        lp[i*primes[j]] = primes[j];
                }
                if(lp[i] == i) mobius[i] = -1;
                else {
                        int x = i/lp[i];
                        if(x%lp[i] == 0) mobius[i] = 0;
                        else mobius[i] = mobius[x]*mobius[lp[i]];
                }
        }
}
```

5.6 modInt

```
struct mi {
    ll v; explicit operator ll() const { return v % mod; }
    mi() { v = 0; }
    mi(ll _v) {
        v = (-mod < _v && _v < mod) ? _v : _v % mod;
        if (v < 0) v += mod;
    }
    friend bool operator==(const mi& a, const mi& b) {
        return a.v == b.v; }
    friend bool operator!=(const mi& a, const mi& b) {
        return !(a == b); }
    friend bool operator<(const mi& a, const mi& b) {
        return a.v < b.v; }

    mi& operator+=(const mi& m) {
        if ((v += m.v) >= mod) v -= mod;
        return *this; }
    mi& operator-=(const mi& m) {
        if ((v -= m.v) < 0) v += mod;
        return *this; }
    mi& operator*=(const mi& m) {
        v = v*m.v%mod; return *this; }
    mi& operator/=(const mi& m) { return (*this) *= inv(m); }
    friend mi pow(mi a, ll p) {
        mi ans = 1; assert(p >= 0);
        for (; p; p /= 2, a *= a) if (p&1) ans *= a;
        return ans;
    }
    friend mi inv(const mi& a) { assert(a.v != 0);
        return pow(a,mod-2); }

    mi operator-() const { return mi(-v); }
    mi& operator++() { return *this += 1; }
    mi& operator--() { return *this -= 1; }
    mi operator++(int) { mi temp; temp.v = v++; return temp; }
    mi operator--(int) { mi temp; temp.v = v--; return temp; }
    friend mi operator+(mi a, const mi& b) { return a += b; }
    friend mi operator-(mi a, const mi& b) { return a -= b; }
    friend mi operator*(mi a, const mi& b) { return a *= b; }
    friend mi operator/(mi a, const mi& b) { return a /= b; }
    friend ostream& operator<<(ostream& os, const mi& m) {
        os << m.v; return os;
    }
    friend istream& operator>>(istream& is, mi& m) {
        ll x; is >> x;
        m.v = x;
        return is;
    }
};
#define vm vector<mi>
```

6 strings

6.1 Z algo

```
// z[i] is the length of the longest string that is,
// at the same time, a prefix of s and
// a prefix of the suffix of s starting at i
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            z[i] = min (r - i + 1, z[i - l]);
        while (i + z[i] < n && s[z[i]] == s[i + z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    } // for number of occurences of t in s
    return z; //str = s+$+t
}
```

7 Trees

7.1 hld

```
//import lca
//import segment tree
const int maxn = 200005;
vvll tree;
vvll par;
ll sz[maxn], pos[maxn], moola[maxn], depth[maxn];
ll heavy[maxn], chain[maxn];
int num = 0;
```

```
//assigns first parent, subtree size, heavy child and depth
int dfs(int curr, int p, int d = 0){
    par[curr][0] = p; sz[curr] = 1;
    depth[curr] = d;
    int maxchild = -1, msize = 0;
    for(auto &child: tree[curr]){
        if(child == p) continue;
        sz[curr] += dfs(child, curr, d+1);
        if(sz[child] > msize){
            maxchild = child; msize = sz[child];
        }
    }
    heavy[curr] = maxchild;
    return sz[curr];
}
//assign pos (in segtree), and chaintop
void decompose(int curr, int p, bool isheavy = false){
    pos[curr] = num++;
    if(isheavy == true) chain[curr] = chain[p];
    else chain[curr] = curr;

    if(heavy[curr] != -1){
        decompose(heavy[curr], curr, true);
    }
    for(auto &child: tree[curr]){
        if(child == p || child == heavy[curr]) continue;
        decompose(child, curr, false);
    }
}

int query(Node* stree, int from, int p){
    int res = 0;
    while(chain[from] != chain[p]){
        int top = pos[chain[from]];
        int till = pos[from];
        res = max(stree->query(top, till+1), res);
        //jump to the above one
        from = chain[from];
        from = par[from][0];
    }
    int top = pos[p];
    int till = pos[from];
    res = max(res, stree->query(top, till+1));
    return res;
}
//call dfs(0,-1) then decompose(0,-1,false)
//then query(node, child, lca), segtree is of size n (vertices)
```

7.2 lca

```
#define sz(x) (int)(x).size()

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]);
        }
    }
    T query(int a, int b) {
        assert(a < b); // or return inf if a == b
        int dep = 31 - __builtin_clz(b - a);
        return min(jmp[dep][a], jmp[dep][b - (1 << dep)]);
    }
};
struct LCA {
       int T = 0;
       vi time, path, ret;
       RMQ<int> rmq;
       //pass in adjacency list, 0-based tree, root at 0
       LCA(vector<vi>& C) : time(sz(C)), rmq((dfs(C,0,-1), ret))
          {}
       void dfs(vector<vi>& C, int v, int par) {
              time[v] = T++;
              for (int y : C[v]) if (y != par) {
                     path.push_back(v), ret.push_back(time[v]);
                     dfs(C, y, v);
              }
       }

       int lca(int a, int b) {
              if (a == b) return a;
              tie(a, b) = minmax(time[a], time[b]);
              return path[rmq.query(a, b)];
       }
       //dist(a,b){return depth[a] + depth[b] -
          2*depth[lca(a,b)];}
};
```