



Universidade de Brasília

É só Fazer

Ruan Petrus, Eduardo Freire, Arthur Botelho

1 Contest

2 Data structures

3 Math

4 String

5 Graph

6 DP

7 Geometry

Contest (1)

template.cpp	28 lines
<pre>#include <bits/stdc++.h> using namespace std; #define int long long #define endl '\n' #define rep(i, a, b) for(int i = (a); i < (b); ++i) #define all(x) begin(x), end(x) #define sz(x) (int)(x).size() #define debug(var) cerr << #var << ": " << var << endl #define pb push_back #define eb emplace_back typedef long long ll; typedef pair<int, int> pii; typedef vector<int> vi; typedef vector<vi> vvi; constexpr int oo = (((unsigned int)-1)>>2); void solve() { } int32_t main() { ios_base::sync_with_stdio(0); cout.tie(0); cin.tie(0); int t = 1; while(t--) solve(); return 0; }</pre>	
.bashrc	3 lines
<pre>alias c='g++ -Wall -Wconversion -Wfatal-errors -g -std=c++17 \ -fsanitize=undefined,address' xmodmap -e 'clear lock' -e 'keycode 66=less greater' #caps =◇</pre>	
.vimrc	4 lines
<pre>set cin ai is ts=4 sw=4 nu nru " Select a region and then type :Hash ca Hash w !cpp -dD -P -fpreprocessed \\\ tr -d '[:space:]' \ \\ md5sum \\\ cut -c-6</pre>	
hash.sh	3 lines
<pre># Hashes a file, ignoring all whitespace and comments. Use for # verifying that code was correctly typed.</pre>	

cpp -dD -P -fpreprocessed tr -d '[:space:]' md5sum cut -c-6	
troubleshoot.txt	52 lines
<pre>Pre-submit: Write a few simple test cases if sample is not enough. Are time limits close? If so, generate max cases. Is the memory usage fine? Could anything overflow? Make sure to submit the right file. Wrong answer: Print your solution! Print debug output, as well. Are you clearing all data structures between test cases? Can your algorithm handle the whole range of input? Read the full problem statement again. Do you handle all corner cases correctly? Have you understood the problem correctly? Any uninitialized variables? Any overflows? Confusing N and M, i and j, etc.? Are you sure your algorithm works? What special cases have you not thought of? Are you sure the STL functions you use work as you think? Add some assertions, maybe resubmit. Create some testcases to run your algorithm on. Go through the algorithm for a simple case. Go through this list again. Explain your algorithm to a teammate. Ask the teammate to look at your code. Go for a small walk, e.g. to the toilet. Is your output format correct? (including whitespace) Rewrite your solution from the start or let a teammate do it. Runtime error: Have you tested all corner cases locally? Any uninitialized variables? Are you reading or writing outside the range of any vector? Any assertions that might fail? Any possible division by 0? (mod 0 for example) Any possible infinite recursion? Invalidated pointers or iterators? Are you using too much memory? Debug with resubmits (e.g. remapped signals, see Various). Time limit exceeded: Do you have any possible infinite loops? What is the complexity of your algorithm? Are you copying a lot of unnecessary data? (References) How big is the input and output? (consider scanf) Avoid vector, map. (use arrays/unordered_map) What do your teammates think about your algorithm? Memory limit exceeded: What is the max amount of memory your algorithm should need? Are you clearing all data structures between test cases?</pre>	
SegTree.h	607842, 25 lines
<p>Description: Iterative SegTree Can be changed by modifying Spec</p> <p>Time: $\mathcal{O}(\log N)$</p> <pre>template<typename LS> struct SegTree { using S = typename LS::S; using K = typename LS::K; int n; vector<S> seg; vector<K> lazy; vector<bool> has_lazy; // vector<int> lx, rx; // Additional info LazySeg(vector<S> & v) : n(sz(v)), seg(2*n) , lazy(n), has_lazy(n) { rep(no, 0, n) seg[no+n] = v[no]; for (int no = n-1; no >= 1; no--) pull(no); // Additional info, n must be power of two /* lx.assign(2*n, 0); rx.assign(2*n, 0); lx[1] = 0; rx[1] = n; rep(no, 1, n) { int mid = (lx[no] + rx[no])/2; lx[no*2] = lx[no]; rx[no*2] = mid; lx[no*2+1] = mid; rx[no*2+1] = rx[no]; } */ } S query(int l, int r) { // [l, r) l += n; r += n; push_to(l); push_to(r-1); S vl = LS::id(), vr = LS::id(); while(l < r) { if (l & 1) vl = LS::op(vl, seg[l++]); if (r & 1) vr = LS::op(seg[--r], vr); l >>= 1; r >>= 1; } return LS::op(vl, vr); } void update(int l, int r, K val) { l += n;</pre>	

vector<S> seg;
SegTree(int _n) : n(_n), seg(2*n, LS::id()) {}
void update(int no, K val) { no += n; seg[no] = LS::update(val, seg[no]); while (no > 1) no /= 2, seg[no] = LS::op(seg[no*2], seg[no*2+1]); }
S query(int l, int r) { // [l, r) S vl = LS::id(), vr = LS::id(); for (l += n, r += n; l < r; l /= 2, r /= 2) { if (l & 1) vl = LS::op(vl, seg[l++]); if (r & 1) vr = LS::op(seg[--r], vr); } return LS::op(vl, vr); }
};
LazySeg.h
Description: Iterative Lazy SegTree Can be changed by modifying Spec
Time: $\mathcal{O}(\log N)$
ee5763, 96 lines
template<typename Spec> struct LazySeg { using LS = Spec; using S = typename LS::S; using K = typename LS::K; int n; vector<S> seg; vector<K> lazy; vector<bool> has_lazy; // vector<int> lx, rx; // Additional info LazySeg(vector<S> & v) : n(sz(v)), seg(2*n) , lazy(n), has_lazy(n) { rep(no, 0, n) seg[no+n] = v[no]; for (int no = n-1; no >= 1; no--) pull(no); // Additional info, n must be power of two /* lx.assign(2*n, 0); rx.assign(2*n, 0); lx[1] = 0; rx[1] = n; rep(no, 1, n) { int mid = (lx[no] + rx[no])/2; lx[no*2] = lx[no]; rx[no*2] = mid; lx[no*2+1] = mid; rx[no*2+1] = rx[no]; } */ } S query(int l, int r) { // [l, r) l += n; r += n; push_to(l); push_to(r-1); S vl = LS::id(), vr = LS::id(); while(l < r) { if (l & 1) vl = LS::op(vl, seg[l++]); if (r & 1) vr = LS::op(seg[--r], vr); l >>= 1; r >>= 1; } return LS::op(vl, vr); } void update(int l, int r, K val) { l += n;

```

    r += n;
    push_to(l); push_to(r-1);
    int lo = 1, ro = 1;
    while(l < r) {
        if (l & 1) lo = max(lo, l), apply(l++, val);
        if (r & 1) ro = max(ro, r), apply(--r, val);
        l >>= 1; r >>= 1;
    }
    pull_from(lo);
    pull_from(ro-1);
}

void apply(int no, K val) {
    seg[no] = LS::update(val, seg[no]);
    // seg[no] = LS::update(val, seg[no], lx[no], rx[no]);

    if (no < n) {
        if (has_lazy[no]) lazy[no] = LS::compose(val, lazy[no]);
        else lazy[no] = val;
        has_lazy[no] = true;
    }
}

void pull_from(int no) {
    while(no > 1) no >>= 1, pull(no);
}

void pull(int no) {
    seg[no] = LS::op(seg[no*2], seg[no*2+1]);
}

void push_to(int no) {
    int h = 0; int p2 = 1;
    while(p2 < no) p2 *= 2, h++;
    for (int i = h; i >= 1; i--) push(no >> i);
}

void push(int no) {
    if (has_lazy[no]) {
        apply(no*2, lazy[no]);
        apply(no*2+1, lazy[no]);
        has_lazy[no] = false;
    }
}

struct Spec {
    using S = int;
    using K = int;
    static S op(S a, S b) { return max(a, b); }
    static S update(K f, S a) { return f + a; }
    static K compose(const K f, const K g) { return f + g; }
    static S id() { return 0; }
};

template<typename LS>
struct Node {
    using S = typename LS::S;
    using K = typename LS::K;

    Node<LS> *l = 0, *r = 0;
    int lo, hi;
    S val = 0;
```

RecSeg.h

Description: Recursive generic persistent dinamic SegTree Can be changed by modifying Spec, queries are inclusive exclusive
Memory: $\mathcal{O}(Q * \log N)$
Time: $\mathcal{O}(\log N)$

99dd6a, 36 lines

```
template<typename LS>
struct Node {
    using S = typename LS::S;
    using K = typename LS::K;

    Node<LS> *l = 0, *r = 0;
    int lo, hi;
    S val = 0;
```

```
Node() {}
Node(int lo,int hi): lo(lo), hi(hi), val(LS::id()) {}

S query(int L, int R) { // [L, R)
    if (R <= lo || hi <= L) return LS::id();
    if (L <= lo && hi <= R) return val;
    push();
    return LS::op(l->query(L, R), r->query(L, R));
}

Node<LS>* update(int idx, K x) {
    if (hi <= idx || idx < lo) return this;
    Node<LS>* me = new Node(lo, hi);
    push(); me->l = l; me->r = r;
    if (hi - lo == 1) me->val = LS::update(x, val);
    else {
        me->l = l->update(idx, x), me->r = r->update(idx, x);
        me->val = LS::op(me->l->val, me->r->val);
    }
    return me;
}

void push() {
    if (!l) {
        int mid = lo + (hi - lo)/2;
        l = new Node(lo, mid); r = new Node(mid, hi);
    }
}

};
```

Psum.h

Description: Multidimensional Psum Requires Abelian Group S (op, inv, id)
Memory: $\mathcal{O}(N^D)$
Time: $\mathcal{O}(1)$

65f259, 29 lines

```
#define MAS template<class... As> //multiple arguments
template<int D, class S>
struct Psum{ using T = typename S::T;
    int n;
    vector<Psum<D-1, S>> v;
    MAS Psum(int s, As... ds):n(s+1),v(n,Psum<D-1, S>(ds...)){
    MAS void set(T x, int p, As... ps){v[p+1].set(x, ps...);}
    void push(Psum& p){rep(i, 1, n)v[i].push(p.v[i]);}
    void init(){rep(i, 1, n)v[i].init(),v[i].push(v[i-1]);}
    MAS T query(int l, int r, As... ps){
        return S::op(v[r+1].query(ps...),S::inv(v[l].query(ps...)))
        ;
    }
};

template<class S>
struct Psum<0, S>{ using T = typename S::T;
    T val=S::id;
    void set(T x){val=x;}
    void push(Psum& a){val=S::op(a.val,val);}
    void init(){}
    T query(){return val;}
};

struct G{
    using T = int;
    static constexpr T id = 0;
    static T op(T a, T b){return a+b;}
    static T inv(T a){return -a;}
};
```

MultiDSegTree.h

Description: Multidimensional SegTree Requires Monoid S (op, id)
Memory: $\mathcal{O}(N^D)$
Time: $\mathcal{O}((\log N)^D)$

53621d, 37 lines

```
//#pragma once

#define MAS template<class... As> //multiple arguments
template<int D, class S>
struct SegTree{ using T = typename S::T;
    int n;
    vector<SegTree<D-1, S>> seg;
    MAS SegTree(int s, As... ds):n(s),seg(2*n, SegTree<D-1, S>(ds...)){}
    MAS T get(int p, As... ps){return seg[p+n].get(ps...);}
    MAS void update(T x, int p, As... ps){
        p+=n; seg[p].update(x, ps...);
        for(p>>=1;p>=1;p>>=1)
            seg[p].update(S::op(seg[2*p].get(ps...),seg[2*p+1].get(ps...)), ps...);
    }
    MAS T query(int l, int r, As... ps){
        T lv=S::id,rv=S::id;
        for(l+=n,r+=n+1;l<r;l>>=1,r>>=1){
            if (l&1)lv = S::op(lv,seg[l+1].query(ps...));
            if (r&1)rv = S::op(seg[--r].query(ps...),rv);
        }
        return S::op(lv,rv);
    }
};

template<class S>
struct SegTree<0, S>{ using T = typename S::T;
    T val=S::id;
    T get(){return val;}
    void update(T x){val=x;}
    T query(){return val;}
};

struct M{ //monoid
    using T = int;
    static constexpr T id = 0;
    static T op(T a, T b){return max(a,b);}
};

SparseTable.h
Description: Multidimensional Sparse Table Requires Idempotent Monoid S (op, inv, id)
Memory:  $\mathcal{O}(n \log n^D)$ 
Time:  $\mathcal{O}(1)$  query,  $\mathcal{O}((n \log n)^D)$  build
c900f0, 39 lines

#define MAS template<class...As> //multiple arguments
template<int D, class S>
struct SpTable{ using T = typename S::T;
    using isp = SpTable<D-1, S>;
    inline int lg(signed x){return __builtin_clz(1)-__builtin_clz(x);}
    int n;
    vector<vector<isp>> tab;
    MAS SpTable(int s, As... ds):n(s),
    tab(1+lg(n),vector<isp>(n,isp(ds...))){}
    MAS void set(T x, int p, As... ps){tab[0][p].set(x, ps...);}
    void join(SpTable& a, SpTable& b){
        rep(i, 0, 1+lg(n))rep(j, 0, n)
            tab[i][j].join(a.tab[i][j], b.tab[i][j]);
    }
    void init(){
```

UnB

```
rep(i, 0, n)tab[0][i].init();
rep(i, 0, lg(n))rep(j, 0, n-(1<<i))
    tab[i+1][j].join(tab[i][j], tab[i][j+(1<<i)]);
}
MAs T query(int l, int r, As... ps){
    int k = lg(r-l+1); r+=1-(1<<k);
    return S::op(tab[k][l].query(ps...),tab[k][r].query(ps...))
        ;
}
};

template<class S>
struct SpTable<0, S>{ using T = typename S::T;
    T val=S::id;
    void set(T x){val=x;}
    void join(SpTable& a, SpTable& b){val=S::op(a.val,b.val);}
    void init(){}
    T query(){return val;}
};

struct IM{
    using T = int;
    static constexpr T id = 0;
    static T op(T a, T b){return max(a, b);}
};
```

BIT.h
Description: Multidimensional BIT/Fenwick Tree Requires Abelian Group
”S” (op, inv, id)
Memory: $\mathcal{O}\left(N^D\right)$
Time: $\mathcal{O}\left((\log N)^D\right)$

```
#define MAs template<class... As> //multiple arguments
template<int D, class S>
struct BIT{ using T = typename S::T;
    int n;
    vector<BIT<D-1, S>> bit;
    MAs BIT(int s, As... ds):n(s),bit(n+1, BIT<D-1, S>(ds...)){}
    inline int lastbit(int x){return x&(-x);}
    MAs void add(T x, int p, As... ps){
        for(p++;p<=n;p+=lastbit(p))bit[p].add(x, ps...);
    }
    MAs T query(int l, int r, As... ps){
        T lv=S::id,rv=S::id; r++;
        for(;r>=1;r-=lastbit(r))rv=S::op(rv,bit[r].query(ps...));
        for(;l>=1;l-=lastbit(l))lv=S::op(lv,bit[l].query(ps...));
        return S::op(rv,S::inv(lv));
    }
};

template<class S>
struct BIT<0, S>{ using T = typename S::T;
    T val=S::id;
    void add(T x){val=S::op(val,x);}
    T query(){return val;}
};
```

```
template<class S>
struct BIT<0, S>{ using T = typename S::T;
    T val=S::id;
    void add(T x){val=S::op(val,x);}
    T query(){return val;}
};

struct AG{ //abelian group analogous to int addition
    using T = int;
    static constexpr T id = 0;
    static T op(T a, T b){return a+b;}
    static T inv(T a){return -a;}
};
```

MoQueries.h
Description: Solve queries offline Can be changed by modifying Spec
Time: $\mathcal{O}\left(n * \sqrt{q}\right)$

7a3c9f, 25 lines

BIT MoQueries Treap HLD

```
template<typename LS>
void mo(LS & v, vector<pii> Q) { // Queries in Q are [l, r)
    int L = 0, R = 0, blk = 350; // N/sqrt(Q)

    vi s(sz(Q));
    auto K = [&](pii x) {return pii(x.first/blk, x.second ^ -(x.first/blk & 1)); };
    iota(all(s), 0);
    sort(all(s), [&](int s, int t){ return K(Q[s]) < K(Q[t]); });

    for (int qi : s) {
        pii q = Q[qi];
        while (L > q.first) v.add(--L, 0);
        while (R < q.second) v.add(R++, 1);
        while (L < q.first) v.del(L++, 0);
        while (R > q.second) v.del(--R, 1);
        v.calc(qi);
    }
}

struct Spec {
    using S = ll;
    void add(int ind, int end) {}
    void del(int ind, int end) {}
    void calc(int idx) {}
};
```

Treap.h
Description: Treap
Memory: $\mathcal{O}(n)$
Time: $\mathcal{O}(\log(b))$

```
struct Treap {
    typedef Treap T;
    T *l, *r;
    int s, w;

    Treap(int aval) : s(1), w(rand()) {
        l=r=0;
    }

    ~Treap() {
        if (l != 0) delete l;
        if (r != 0) delete r;
    }
    static int size(T *no) {
        if (no == 0) return 0;
        return no->s;
    }
}
```

```
static pair<T*, T*>split(T *no, int k) {
    if (no == 0) return {0,0};
    return no->split(k);
}

pair<T*, T*> split(int k) {
    T *nl, *nr, *sl, *sr;
    if (size(l) < k) {
        if (r == 0) sl=sr=0;
        else tie(sl,sr) = r->split(k-size(l)-1);
        r=sl; nr=sr; nl=this;
    } else {
        if (l == 0) sl=sr=0;
        else tie(sl,sr) = l->split(k);
        nl=sl; l=sr; nr=this;
    }

    s = size(l)+size(r)+1;
```

```
    return {nl,nr};
}

static T* merge(T *l, T *r) {
    if (l == 0) return r;
    if (r == 0) return l;

    T *ans;
    if (l->w < r->w) ans=l, ans->r=merge(l->r,r);
    else ans=r, ans->l=merge(l,r->l);

    ans->s = size(ans->l) + size(ans->r) + 1;
    return ans;
}
};
```

HLD.h
Description: Iterative, lazy and noncommutative HLD Does exactly the same thing as a segment tree, but on a tree.
Time: query: $\mathcal{O}(\log^2 N)$, point update: $\mathcal{O}(\log N)$, range update: $\mathcal{O}(\log^2 N)$
"SegTree.h" ed20eb, 89 lines
// If T::op is commutative, memory usage can be cut in half by removing 'segi' from the code.
// Simply remove it and replace every occurrence of segi.query(n-r, n-l) with seg.query(l,r).
// Also remove every line involving updating segi. This also speeds up range updates by a factor of 2.

```
template<class T>
struct HLD {
    using S = typename T::S;
    using K = typename T::K;
    vvi g;
    int rt,n,dt;
    vi hv, ti,to, hd, p, ssz;
    SegTree<T> seg, segi;
    HLD(vvi ag, int art=0) :
        g(ag),rt(art),n(g.size()),dt(0),
        hv(n),ti(n),to(n),hd(n),p(n), ssz(n),seg(n), segi(n)
    {
        dfs1(rt,rt);
        dfs2(rt,rt);
    }
}
```

```
void dfs1(int a, int pai) {
    p[a]=pai;
    ssz[a]=1;

    pii hc(-1,-1);
    for (auto b:g[a]) if(b!=pai) {
        dfs1(b,a);
        hc=max(hc, pii(ssz[b], b));
        ssz[a]+=ssz[b];
    }
    hv[a]=hc.second;
}
```

```
void dfs2(int a, int h) {
    hd[a]=h;
    ti[a]=dt++;
    if (hv[a] != -1) dfs2(hv[a], h);
    for (auto b:g[a]) if (b != p[a] and b != hv[a])
        dfs2(b, b);
    to[a]=dt;
}
```

```
bool isa(int a, int b) {
    return ti[a]<=ti[b] and to[a]>=to[b];
```

3

```

}

// seg.query(l, r) must query [l, r).
// Query path from a to b, inclusive on both ends.
S query(int a, int b) {
    // SegTree identity
    S a1=T::id(),a2=T::id();

    for (;!isa(hd[a], b); a=p[hd[a]])
        a1=T::op(a1, segi.query(n-ti[a]-1, n-ti[hd[a]]));

    for (;!isa(hd[b],a);b=p[hd[b]])
        a2=T::op(seg.query(ti[hd[b]], ti[b]+1), a2);

    if (isa(a,b)) a2=T::op(seg.query(ti[a], ti[b]+1), a2);
    else a1=T::op(a1, segi.query(n-ti[a]-1, n-ti[b]));
    return T::op(a1, a2);
}

void update(int a, K v) {
    seg.update(ti[a], v);
    segi.update(n-1-ti[a], v);
}

// lazy update on path from a to b, inclusive on both ends.
void update(int a, int b, K v) {
    for (;!isa(hd[a], b); a=p[hd[a]]) {
        seg.update(ti[hd[a]], ti[a]+1, v);
        segi.update(n-ti[a]-1, n-ti[hd[a]], v);
    }

    for (;!isa(hd[b],a);b=p[hd[b]]) {
        seg.update(ti[hd[b]], ti[b]+1, v);
        segi.update(n-ti[b]-1, n-ti[hd[b]], v);
    }

    if (isa(a,b)) {
        seg.update(ti[a], ti[b]+1, v);
        segi.update(n-ti[b]-1, n-ti[a], v);
    }
    else {
        seg.update(ti[b], ti[a]+1, v);
        segi.update(n-ti[a]-1, n-ti[b], v);
    }
}
};
```

Math (3)

LinearDiophantineEquation.h

Description: Find a solution to equation $a*x + b*y = c$

Time: $\mathcal{O}(\log(a))$

```
array<ll, 3> exgcd(ll a, ll b) {
    if (a == 0) return {0, 1, b};
    auto [x, y, g] = exgcd(b % a, a);
    return {y - b / a * x, x, g};
}

// if (x,y) is a solution (x-kb/d, y+ka/d) for all integer k
array<ll, 4> find_any_solution(ll a, ll b, ll c) {
    assert(a != 0 || b != 0);
    auto[x, y, g] = exgcd(a, b);
    if (c % g) return {false, 0, 0, 0};
    x *= c / g;
    y *= c / g;
    return {true, x, y, g};
}
```

fft.h

Description: Polynomial multiplication modulo 998244353

Time: $\mathcal{O}(n\log(n))$

```
constexpr int MOD=998244353;

int fpow(int a, int b) {
    int x=1;
    while(b) {
        if (b&1) x=x*a%MOD;

        a=(ll)a*a%MOD;
        b/=2;
    }
    return x;
}

void fft(vi &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vi rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, fpow(62, MOD >> s)};
        rep(i,k,2*k) rt[i] = (ll) rt[i / 2] * z[i & 1] % MOD;
    }
    vi rev(n);
    rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
            ll z = (ll) rt[j + k] * a[i + j + k] % MOD; int &ai = a[i
                + j];
            a[i + j + k] = ai - z + (z > ai ? MOD : 0);
            ai += (ai + z >= MOD ? z - MOD : z);
        }
}

vi mul(vi &a, vi &b) {
    if (a.empty() || b.empty()) return {};
    int s = sz(a) + sz(b) - 1,
        n = 1 << (32-__builtin_clz(s));
    ll inv = fpow(n, MOD - 2);
    vi l(a), r(b), out(n);
    l.resize(n), r.resize(n);
    fft(l), fft(r);
    rep(i,0,n)
        out[-i & (n - 1)] = (ll)l[i] * r[i] % MOD * inv % MOD;
    fft(out);
    return {out.begin(), out.begin() + s};
}
```

Combinatorics.h

Description: combinatorics structure

Memory: $\mathcal{O}(mxn)$

Time: $\mathcal{O}(mxn)$

```
#define mul(a, b) (((ll)a*b)%mod)
template<int mod>
int fexp(int a, int b){
    int res = 1;
    for(;b;a=mul(a,a),b>>=1)if(b&1)res=mul(res,a);
    return res;
}

template<int mod>
struct Combinatorics{
    vi f, fi;
    Combinatorics(int mxn):f(mxn),fi(mxn){
        f[0] = 1; rep(i, 1, mxn)f[i]=mul(f[i-1],i);
        fi[mxn-1] = fexp<mod>(f[mxn-1], mod-2);
        for(int i=mxn-1;i>0;i--)fi[i-1] = mul(fi[i],i);
    }
}
```

```
int choose(int n, int k){return mul(f[n],mul(fi[k],fi[n-k]));
};
```

String (4)

Hash.h

Description: String hashing

Memory: $\mathcal{O}(n)$

Time: $\mathcal{O}(1)$ query, $\mathcal{O}(n)$ build

```
<sys/time.h>
2dlad6, 43 lines

/*
    Arithmetic mod two primes and 2^32 simultaneously.
    C can be initalialize to a number less than MOD or random

    timeval tp;
    gettimeofday(&tp, 0);
    C = (int)tp.tv_usec; // (less than modulo)
    assert((ull)(H(1)*2+1-3) == 0);
*/

typedef uint64_t ull;
template<int M, class B>
struct A {
    int x; B b; A(int x=0) : x(x), b(x) {}
    A(int x, B b) : x(x), b(b) {}
    A operator+(A o) const {int y = x+o.x; return{y - (y>=M)*M, b
        +o.b};}
    A operator-(A o) const {int y = x-o.x; return{y + (y< 0)*M, b
        -o.b};}
    A operator*(A o) const { return {(int)((ll)x*o.x % M), b*o.b
        }; }
    explicit operator ull() const { return x ^ (ull) b << 21; }
    bool operator==(A o) const { return (ull)*this == (ull)o; }
    bool operator<(A o) const { return (ull)*this < (ull)o; }
};

typedef A<1000000007, A<1000000009, unsigned>> H;
```

static int C;

```
struct HashInterval {
    int n;
    vector<H> ha, pw;
    template<typename S>
    HashInterval(const S &str) : n(sz(str)), ha(n+1), pw(n+1) {
        pw[0] = 1;
        rep(i,0,n)
            ha[i+1] = ha[i] * C + str[i],
            pw[i+1] = pw[i] * C;
    }
    H query(int a, int b) { // hash [a, b)
        return ha[b] - ha[a] * pw[b - a];
    }
    H queryI(int a, int b) {
        return query(n - b, n - a);
    }
};
```

DinamicHashing.h

Description: Dinamic string hashing

Memory: $\mathcal{O}(n)$

Time: $\mathcal{O}(1)$ query, $\mathcal{O}(n)$ build

```
typedef uint64_t ull;

template<int M, class B>
struct A {
```

```
int x; B b;
constexpr A(int x=0) : x(x), b(x) {}
constexpr A(int x, B b) : x(x), b(b) {}
A operator+(A o) const {int y = x+o.x; return{y - (y>=M)*M, b +o.b};}
A operator-(A o) const {int y = x-o.x; return{y + (y< 0)*M, b -o.b};}
A operator*(A o) const { return {(int)(1LL*x*o.x % M), b*o.b}; }
explicit operator ull() const { return x ^ (ull) b << 21; }
bool operator==(A o) const { return (ull)*this == (ull)o; }
bool operator<(A o) const { return (ull)*this < (ull)o; }
};

typedef A<989831283, A<912391239 , unsigned>> H;

const static int C = 12312;

struct DinamicHash {
    int n;
    vector<int> s;
    vector<H> p;
    SegTree seg;
    DinamicHash(const vector<int> & v) : n(v.size()), s(v), p(n+1) {
        p[0] = 1;
        vector<H> values(n);
        rep(i, 0, n) {
            values[i] = p[i] * s[i];
            p[i+1] = p[i] * C;
        }
        seg = SegTree(values);
    };
    H query(int l, int r) const { // [l, r)
        return seg.query(l, r) * p[n-r];
    }
    void update(int idx, int v) {
        s[idx] = v;
        seg.update(idx, p[idx] * s[idx]);
    }
};
```

Automata.h
Description: Suffix automata
Memory: $\mathcal{O}(n * 26)$
Time: $\mathcal{O}(n)$ build

92d90c, 49 lines

```
struct Automata {
    int saID = 1, last = 1;
    int n;
    vector<int> len, lnk;
    vector<array<int,27>> to;
    vector<int> occ, fpos;
    vector<int> states;

    Automata(const string & s, const char a = 'a')
        : n(s.size()), len(2*n+2), lnk(2*n+2), to(2*n+2, {0}), occ(2*n+2), fpos(2*n+2) {
        for (const auto & c: s) push(c-a);

        states.assign(saID, 0);
        iota(all(states), 1);
        sort(all(states), [&](const auto & u, const auto & v) {
            return len[u] > len[v]; });
        for (auto st: states) {
            occ[lnk[st]] += occ[st];
        }
    }
};
```

```
void push(int c) {
    int a = ++saID;
    int p = last;
    last = a;

    len[a] = len[p] + 1;
    occ[a] = 1;
    fpos[a] = len[a] - 1;

    for (; p > 0 && !to[p][c]; p = lnk[p]) to[p][c] = a;
    int q = to[p][c];
    if (p == 0) {
        lnk[a] = 1;
    }
    else if (len[p] + 1 == len[q]) {
        lnk[a] = q;
    }
    else {
        int clone = ++saID;
        lnk[clone] = lnk[q];
        to[clone] = to[q];
        fpos[clone] = fpos[q];

        len[clone] = len[p] + 1;
        lnk[a] = lnk[q] = clone;
        for (; to[p][c] == q; p = lnk[p]) to[p][c] = clone;
    }
}
};
```

KMP.h
Description: KMP automaton
Memory: $\mathcal{O}(\text{alphabet size} * n)$
Time: $\mathcal{O}(\text{alphabet size} * n)$ build, $\mathcal{O}(1)$ query

fd7365, 20 lines

```
template<class T>
struct KMP{
    T in; int n; vi p; vector<vi> a;
    template<class S>
    KMP(S s, T ain, int asz):n(sz(s)),p(n), in(ain), a(n+1, vi(asz,0)){
        rep(i, 1, n){
            int j = p[i-1];
            while(j and s[j]!=s[i])j = p[j-1];
            p[i] = j + (s[i]==s[j]);
        }
        rep(i, 0, n+1)
            rep(c, 0, asz){
                if (i and (i==n or c+in!=s[i]))a[i][c] = a[p[i-1]][c];
                else a[i][c] = i + (c+in == s[i]);
            }
        }
    int nxt(int cur, T c){
        return a[cur][c-in];
    }
};
```

Aho.h
Description: Aho automaton
Memory: $\mathcal{O}(\text{alphabet size} * n)$
Time: $\mathcal{O}(\text{alphabet size} * n)$ build, $\mathcal{O}(1)$ query

ae0d54, 49 lines

```
struct Aho {
    int n=1, si; char in;
    vvi tran, nxt;
    vi lnk, term, h;

    // ain= initial alphabet letter, asi = alphabet size
    Aho(char ain, int asi) {
```

```
in = ain;
si = asi;
tran.eb(si,-1);
term.pb(0);
}

void add(string& s) {
    int cur=0;
    rep(i,0,s.size()) {
        int& nxt= tran[cur][s[i]-in];
        if (nxt != -1) cur=nxt;
        else nxt=cur=n++, term.pb(0),tran.eb(si,-1);
    }
    term[cur]+=1;
}

void init() {
    lnk.assign(n,0);
    nxt.assign(n, vi(si));
    h.assign(n,0);

    queue<int> q;
    rep(c,0,si) {
        int& f=tran[0][c];
        if (f != -1) q.push(f), h[f]=1, nxt[0][c]=f;
        else nxt[0][c]=0;
    }
    while (!q.empty()) {
        int a=q.front(); q.pop();
        rep(c,0,si) {
            int& b=nxt[a][c];
            int fail=nxt[lnk[a]][c];
            if (tran[a][c] != -1) {
                b = tran[a][c];
                lnk[b] = fail;
                q.push(b);
                h[b]=h[a]+1;
            } else b=fail;
        }
    }
};
```

Graph (5)

Kosaraju.h
Description: Kosaraju
Memory: $\mathcal{O}(n)$
Time: $\mathcal{O}(n + m)$ query, $\mathcal{O}(n + m)$ build

eb04c1, 43 lines

```
struct Kosaraju {
    int n;
    vector<vector<int>> g, rg;
    vector<bool> vis;
    vector<int> id;
    vector<vector<int>> dag, comp;
    int cc = 0;
    vector<int> S;

    Kosaraju(int _n)
        : n(_n), g(n), rg(n), vis(n), id(n) {}

    void add_edge(int a, int b) {
        g[a].eb(b);
        rg[b].eb(a);
    }
    void dfs(int a) {
        vis[a] = true;
```

```
    for (auto b: g[a]) if (!vis[b]) dfs(b);
    S.eb(a);
}
void scc(int a, int c) {
    vis[a] = true;
    id[a] = c;
    for (auto b: rg[a]) if (!vis[b]) scc(b, c);
}
void run() {
    rep(a, 0, n) if (!vis[a]) dfs(a);
    fill(all(vis), 0);
    reverse(all(S));
    for (auto a: S) if (!vis[a]) scc(a, cc++);

    dag.resize(cc); comp.resize(cc);
    vector<pair<int, int>> edges;
    rep(a, 0, n) {
        comp[id[a]].eb(a);
        for (int b: g[a]) if (id[a] != id[b]) edges.eb(id[a], id[b]);
    }
    sort(all(edges));
    edges.erase(unique(all(edges)), edges.end());
    for (const auto & [a, b]: edges) dag[a].eb(b);
}
};
```

TwoSat.h

Description: Two sat

Memory: $\mathcal{O}(n)$

Time: $\mathcal{O}(n + m)$ query, $\mathcal{O}(n + m)$ build

0b8692, 64 lines

```
struct TwoSat{
    int n;
    vector<vector<int>> g, gi;
    vector<bool> vis;
    vector<int> vars, comp;
    vector<int> top;
    TwoSat(int _n)
        : n(_n), g(2*_n), gi(2*_n), vis(2*_n), vars(n, -1), comp(2*_n)
    {}

    int neg(int a) {
        if (a >= n) return a-n;
        return a + n;
    }

    void add_or(int a, int b) {
        g[neg(a)].eb(b);
        g[neg(b)].eb(a);

        gi[b].eb(neg(a));
        gi[a].eb(neg(b));
    }

    void add_imp(int a, int b) {
        add_or(neg(a), b);
    }

    void add_either(int a, int b) {
        add_or(a, b);
        add_or(neg(a), neg(b));
    }

    void dfs(int a) {
        vis[a] = true;
        for (auto b: g[a]) if (!vis[b]) dfs(b);
        top.eb(a);
    }
};
```

```
void idfs(int a, int c){
    vis[a] = true;
    comp[a] = c;
    for (auto b: gi[a]) if (!vis[b]) idfs(b, c);
}

bool sat() {
    int c = 0;
    rep(a, 0, 2*n) if (!vis[a]) dfs(a);

    fill(all(vis), 0);
    reverse(all(top));
    for(int a : top) if (!vis[a]) idfs(a, c++);

    for(int a: top){
        if (comp[a] == comp[neg(a)]) return false;

        bool is_neg = a >= n;
        if (is_neg) a = neg(a);

        if (vars[a] == -1) vars[a] = is_neg;
    }

    return true;
}
};
```

OnlineMatching.h

Description: Modified khun developed for specific question able to run $2 * 10^6$ queries, in $2 * 10^6 \times 10^6$ graph in 3 seconds codeforces

Time: $\mathcal{O}(confia)$

6ac539, 42 lines

```
struct OnlineMatching {
    int n = 0, m = 0;
    vector<int> vis, match, dist;
    vector<vector<int>> g;
    vector<int> last;
    int t = 0;

    OnlineMatching(int n_, int m_) : n(n_), m(m_),
        vis(n, 0), match(m, -1), dist(n, n+1), g(n), last(n, -1)
    {}

    void add(int a, int b) {
        g[a].pb(b);
    }

    bool kuhn(int a) {
        vis[a] = t;
        for(int b: g[a]) {
            int c = match[b];
            if (c == -1) {
                match[b] = a;
                return true;
            }
            if (last[c] != t || (dist[a] + 1 < dist[c]))
                dist[c] = dist[a] + 1, last[c] = t;
        }
        for (int b: g[a]) {
            int c = match[b];
            if (dist[a] + 1 == dist[c] && vis[c] != t && kuhn(c)) {
                match[b] = a;
                return true;
            }
        }
        return false;
    }

    bool can_match(int a) {
```

```
        t++;
        last[a] = t;
        dist[a] = 0;
        return kuhn(a);
    }
};

FunctGraph.h
Description: Functional Graph
Memory:  $\mathcal{O}(n)$ 
Time:  $\mathcal{O}(n)$ 
33bd25, 24 lines

struct FunctGraph{
    int n;
    vi head;
    vector<vi> gr, comps;

    FunctGraph(vi& fn):
        n(sz(fn)), head(n, -1), gr(n){
            rep(i, 0, n)gr[fn[i]].pb(i);
            vi visited(n, 0);
            auto dfs = [&](auto rec, int v, int c) -> void{
                head[v] = c; visited[v] = 1;
                for(int f : gr[v])if (head[f]!=f)rec(rec, f, c);
            };
            rep(i, 0, n){
                if (visited[i])continue;
                int l=fn[i], r=fn[fn[i]];
                while(l!=r) l=fn[l], r=fn[fn[r]];
                vi cur = {r};
                for(l=fn[l]; l!=r; l=fn[l]) cur.pb(l);
                for(int x : cur) dfs(dfs, x, x);
                comps.pb(cur);
            }
        }
};
```

Hierholzer.h

Description: Eulerian path/cycles if existing

Memory: $\mathcal{O}(V + E)$

Time: $\mathcal{O}(E)$

a85ad9, 29 lines

```
vi hierholzer(int n, vector<pii>& edges, int inic){
    vi ans; int m = sz(edges);
    auto check = [&]()->bool{return true;};
    if (not check()){
        //a function should be created to check conditions
        //according to type of graph and problem restrictions on
        //the path type and enpoints
        //base conditions: edge connectivity and vertex degree
        return ans; //empty vector if impossible
    }
    vector<vi> g(n);
    rep(i, 0, m){
        auto [a, b] = edges[i];
        g[a].pb(i); g[b].pb(i); //remove the latter if it's
            directed
    }
    vi used(m, false), st = {inic};
    while(not st.empty()){
        int v = st.back();
        while(not g[v].empty() and used[g[v].back()])g[v].pop_back();
        if (g[v].empty())st.pop_back(), ans.pb(v);
        else {
            int idx = g[v].back(); g[v].pop_back();
            auto [a, b] = edges[idx]; used[idx] = true;
            st.pb((v==a ? b : a));
        }
    }
};
```

```
    }
    reverse(all(ans));
    return ans;
}
```

DominatorTree.h

Description: Dominator Tree, creates the graph tree, where all ancestors of a u in the tree are necessary in the path from the root to u
Memory: $\mathcal{O}(n)$
Time: $\mathcal{O}((n+m)\log(n))$ build

```
struct DominatorTree {
    int n;
    vector<vector<int>> g, gt, tree, bucket, down;
    vector<int> S;
    vector<int> dsu, label, sdom, idom, id;
    int dfstime =0;

    DominatorTree(vector<vector<int>> & _g, int root)
        : n(sz(_g)), g(_g), gt(n), tree(n), bucket(n), down(n),
          S(n), dsu(n), label(n), sdom(n), idom(n), id(n) {
        prep(root); reverse(all(S));
        for(int u : S) {
            for(int v : gt[u]) {
                int w = fnd(v);
                if(id[ sdom[w] ] < id[ sdom[u] ])
                    sdom[u] = sdom[w];
            }
            gt[u].clear();
            if(u != root) bucket[ sdom[u] ].push_back(u);
            for(int v : bucket[u]) {
                int w = fnd(v);
                if(sdom[w] == sdom[v]) idom[v] = sdom[v];
                else idom[v] = w;
            }
            bucket[u].clear();
            for(int v : down[u]) dsu[v] = u;
            down[u].clear();
        }
        reverse(S.begin(), S.end());
        for(int u : S) if(u != root) {
            if(idom[u] != sdom[u]) idom[u] = idom[ idom[u] ];
            tree[ idom[u] ].push_back(u);
        }
    }

    void prep(int u){
        S[dfstime] = u;
        id[u] = ++dfstime;
        label[u] = sdom[u] = dsu[u] = u;

        for(int v : g[u]){
            if(!id[v])
                prep(v), down[u].push_back(v);
            gt[v].push_back(u);
        }
    }

    int fnd(int u, int flag = 0){
        if(u == dsu[u]) return u;
        int v = fnd(dsu[u], 1), b = label[ dsu[u] ];
        if(id[ sdom[b] ] < id[ sdom[ label[u] ] ])
            label[u] = b;
        dsu[u] = v;
        return flag ? v : label[u];
    }
};
```

DominatorTree Dinic MCMF

Dinic.h

Description: finds maximum network flow
Memory: $\mathcal{O}(V + E)$
Time: $\mathcal{O}(V * E * \log(maxVal))$

```
d004f4, 55 lines

/*
    Observations:
    * — Edge capacity is implemented as "remaining capacity for flow",
    * without variable for current passing flow
    * — Zero limit (eps) should be changed according to required precision
    * for float capacity edges
    * Tested at: CSES-Download Speed
*/
template<class T>
struct Dinic{
    struct Edge{int a, b; T w; bool rev;};
    int n, m; T mx;
    vector<vi> g; vector<Edge> es;
    Dinic(int s):n(s),m(0),mx(1),g(n){

        void add_edge(int a, int b, T w){
            g[a].pb(m++); g[b].pb(m++);
            es.pb({a,b,w,false}); es.pb({b,a,T(0),true});
            while (w>=mx) mx+=mx;
        }

        T maxflow(int source, int sink){
            T eps = T(1)/T(oo); //associated to constant for float flow
            vi ce(n, 0), dep(n, -1);
            auto make_dag = [&](T cmx)->bool{
                ce.assign(n, 0); dep.assign(n, -1);
                queue<int> q; q.push(source); dep[source] = 0;
                while(not q.empty()){
                    int v = q.front(); q.pop();
                    for(int i : g[v]){ auto& e = es[i];
                        if (e.w < cmx or dep[e.b]!=-1)continue;
                        dep[e.b] = dep[v]+1; q.push(e.b);
                    }
                }
                return dep[sink]!=-1;
            };
            auto push_flow = [&](auto rec, int v, T f)->T{
                if (v==sink)return f;
                T cur(0);
                for(int& i = ce[v]; i < sz(g[v]); i++){
                    int j = g[v][i]; auto& e = es[j];
                    if (dep[e.b]!=dep[e.a]+1 or e.w<=eps)continue;
                    T cf = rec(rec, e.b, min(e.w, f));
                    f -= cf; cur += cf; e.w -= cf; es[j^1].w += cf;
                    if (f<=eps)return cur;
                }
                return cur;
            };
            T res(0);
            for(T cmx=mx,cf; cmx>eps;cmx/=T(2)){ while(make_dag(cmx))
                while((cf=push_flow(push_flow,source,T(oo)))>eps)res += cf;
            }
            return res;
        }
    };
};
```

MCMF.h

Description: minimum cost for maximum flow in network
Memory: $\mathcal{O}(V + E)$

Time: — Preprocessing: SPFA ($\mathcal{O}(V * E)$) — Max number of iterations: min(maxflow, max cost path) — Complexity for each iteration: — Dijkstra: $\mathcal{O}(V + E\log E)$ — DFS: $\mathcal{O}(E * V)$? 38ee78, 75 lines

```
/*
    * Observations:
    * — pots_init is only useful if there are negative initial costs
    * — Dijkstra path recover can be used as (slower?) alternative to push flow
    */

template<class TF, class TC>
struct MCMF{
    struct Edge{int a, b; TF w; TC c;};
    int n, m;
    vector<vi> g; vector<Edge> es;
    MCMF(int s):n(s),m(0),g(n){

        void add_edge(int a, int b, TF w, TC c){
            g[a].pb(m++); g[b].pb(m++);
            es.pb({a, b, w, c}); es.pb({b, a, TF(0), -c});
        }

        pair<TF, TC> mcmf(int source, int sink){
            TF eps = TF(1)/TF(oo);
            vector<TC> ds(n,TC(0)), ps(n,TC(0));
            vi ce(n, 0), on(n, 0);
            auto ecost = [&](Edge& e)->TC{return ps[e.a]-ps[e.b]+e.c;};
            auto pots_init = [&]()->void{
                ps.assign(n, TC(oo)); vi inq(n, 0);
                queue<int> q; q.push(source);
                inq[source] = 1; ps[source] = 0;
                while(not q.empty()){
                    int v = q.front(); q.pop(); inq[v] = 0;
                    for(int i : g[v]){ auto& e = es[i];
                        if (e.w<=eps or ps[e.b]<=ps[v]+e.c)continue;
                        if (not inq[e.b])q.push(e.b);
                        inq[e.b] = 1; ps[e.b] = ps[v]+e.c;
                    }
                }
            };
            auto dists_calc = [&]()->bool{
                rep(v, 0, n)if (ps[v]<TC(oo))ps[v]+=ds[v];
                ds.assign(n, TC(oo)); ce.assign(n, 0);
                vi vis(n, 0); using P = pair<int, TC>;
                priority_queue<P, vector<P>, greater<P>> pq;
                pq.push({ds[source]=TC(0), source});
                while(not pq.empty()){
                    auto [d, v] = pq.top(); pq.pop();
                    if (vis[v])continue;
                    vis[v] = true;
                    for(int i : g[v]){ auto& e = es[i];
                        if (e.w<=eps or ds[e.b]<=d+ecost(e))continue;
                        pq.push({ds[e.b]=d+ecost(e), e.b});
                    }
                }
                return ds[sink]!=TC(oo);
            };

            auto push_flow = [&](auto rec, int v, TF f)->pair<TF, TC>{
                if (v==sink)return {f, TC(0)};
                on[v] = 1; TF curf(0); TC curc(0);
                for(int& i = ce[v]; i < sz(g[v]); i++){
                    int j = g[v][i]; auto& e = es[j];
                    if (on[e.b] or e.w<=eps)continue;
                    if (ecost(e)>ds[e.b]-ds[e.a])continue;
                    auto [cf, cc] = rec(rec, e.b, min(f, e.w));
                    f-=cf; curf+=cf; e.w-=cf; es[j^1].w+=cf;
                }
            };
        }
    };
};
```



```
        curc += e.c*cf + cc;
        if (f<=eps){on[v] = 0; return {curf, curc};}
    }
    on[v] = 0; return {curf, curc};
};
TF flow(0), cf(oo); TC cost(0), cc(0);
for(pots_init();dists_calc();)
    for(cf=TF(oo);cf>eps;flow+=cf,cost+=cc)
        tie(cf, cc)=push_flow(push_flow,source,TF(oo));
    return {flow, cost};
}
};
```

DP (6)

sos.h
Description: Sos DP
Time: $\mathcal{O}(n * 2^n)$ 5063f0, 19 lines

```
//iterative version
for(int mask = 0; mask < (1<<N); ++mask){
    dp[mask][-1] = A[mask]; //handle base case separately (leaf states)
    for(int i = 0;i < N; ++i){
        if(mask & (1<<i))
            dp[mask][i] = dp[mask][i-1] + dp[mask^(1<<i)][i-1];
        else
            dp[mask][i] = dp[mask][i-1];
    }
    F[mask] = dp[mask][N-1];
}

//memory optimized, super easy to code.
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)];
}
}
```

Geometry (7)

Point.h
Description: 2D point structure 6a7340, 39 lines

```
constexpr float EPS=1e-12;
constexpr float PI=acos(-1);

bool eq(float a, float b) {
    return abs(a-b) < EPS;
}

template<class T>
struct Point {
    typedef Point P;
    static constexpr int ret[2][2] = {{3, 2}, {4, 1}};
    T x, y;

    Point(T x=0, T y=0) : x(x), y(y) {}
    bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
    bool operator==(P p) const { return eq(x, p.x) and eq(y,p.y); }

    P operator+(P p) const { return P(x+p.x, y+p.y); }
    P operator-(P p) const { return P(x-p.x, y-p.y); }
    P operator*(T d) const { return P(x*d, y*d); }
    P operator/(T d) const { return P(x/d, y/d); }
```

```
T operator*(P p) const { return x*p.x+y*p.y; }
T operator^(P p) const { return x*p.y - y*p.x; }
T dist2() const { return x*x + y*y; }
int quad() const { return ret[x >= 0][y >= 0]; }

// angle to x-axis in interval [0, 2*PI]
double angle() const {
    auto an=atan2(y,x);
    return an < 0 ? an+2*PI: an;
}

friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << ", " << p.y << ")";
}

static bool angle_comp(P a, P b){
    int qa = a.quad(), qb = b.quad();
    return (qa == qb ? (a ^ b) > 0 : qa < qb);
}
};
```

Techniques (A)

techniques.txt	117 lines
Recursion	
Divide and conquer	
Finding interesting points in N log N	
Algorithm analysis	
Master theorem	
Amortized time complexity	
Greedy algorithm	
Max contiguous subvector sum	
Invariants	
Graph theory	
DP, com cyclo no dikstra reverso	
Breadth first search	
Depth first search	
DFS trees	
Dijkstra's algorithm	
MST: Prim's algorithm	
Bellman-Ford	
Min-cost max flow	
Floyd-Warshall	
Euler cycles	
Flow networks	
Bipartite matching	
Topological sorting	
Strongly connected components	
2-SAT	
Cut vertices, cut-edges and biconnected components	
Vertex coloring	
Bipartite graphs	
Diameter and centroid	
K'th shortest path	
Shortest cycle	
Dynamic programming	
Knapsack	
Longest common subsequence	
Longest increasing subsequence	
Number of paths in a dag	
Shortest path in a dag	
over intervals	
over subsets	
over probabilities	
over trees	
3^n set cover	
Divide and conquer	
Convex hull optimizations	
RMQ (sparse table a.k.a 2^k-jumps)	
Combinatorics	
Inclusion/exclusion	
Catalan number	
Pick's theorem	
Number theory	
Integer parts (School's excursion)	
Divisibility	
Euclidean algorithm	
Modular inverses	
Modular exponentiation by squaring	
Chinese remainder theorem	
Fermat's little theorem	
Euler's theorem	
Phi function	
Frobenius number	
Quadratic reciprocity	
Pollard-Rho	
Miller-Rabin	
Hensel lifting	
Vieta root jumping	

Game theory
Combinatorial games
Game trees
Mini-max
Nim
Games on graphs
Games on graphs with loops
Grundy numbers
Bipartite games without repetition
General games without repetition
Alpha-beta pruning
Probability theory
Optimization
Binary search
Ternary search (Convex functions)
Binary search on derivative
Numerical methods
Newton's method
Root-finding with binary/ternary search
Matrices
Gaussian elimination
Exponentiation by squaring
Geometry
Cross product
Scalar product
Convex hull
Polygon cut
Closest pair (Distance functions)
Hull diameter (Distance functions)
Sweeping
Discretization (convert to events and sweep)
Angle sweeping
Line sweeping
Strings
Longest common substring
Knuth-Morris-Pratt
Tries
Rolling polynomial hashes
Aho-Corasick
Letter position lists
Combinatorial search
Meet in the middle
Brute-force with pruning
Data structures
LCA (2^k-jumps in trees in general)
HLD
Centroid decomposition
SegTree, LazySeg
Convex hull trick (wcipeg.com/wiki/Convex_hull_trick)
Monotone queues / monotone stacks / sliding queues
Sliding queue using 2 stacks
Persistent segment tree