



Universidad de Guadalajara,
CUCEI

The Empire Strikes Back

Abraham Murillo, Roberto Pino, Uriel Guzmán

Contents

1 Data structures	3	7 Combinatorics	15
1.1 Disjoint set with rollback	3	7.1 Factorial	15
1.2 Min-Max queue	3	7.2 Factorial mod <i>smallPrime</i>	15
1.3 Sparse table	3	7.3 Lucas theorem	15
1.4 Squirtle decomposition	3	7.4 Stars and bars	16
1.5 In-Out trick	3	7.5 N choose K	16
1.6 Parallel binary search	3	7.6 Catalan	16
1.7 Mo's algorithm	4	7.7 Burnside's lemma	16
1.8 Ordered tree	4	7.8 Prime factors of N!	16
1.9 Unordered tree	4	8 Number Theory	16
1.10 D-dimensional Fenwick tree	4	8.1 Goldbach conjecture	16
1.11 Dynamic segment tree	5	8.2 Sieve of Eratosthenes	16
1.12 Persistent segment tree	5	8.3 Phi of euler	16
1.13 Wavelet tree	5	8.4 Miller-Rabin	16
1.14 Li Chao tree	5	8.5 Pollard-Rho	16
1.15 Treap	6	8.6 Amount of divisors	17
2 Graphs	6	8.7 Bézout's identity	17
2.1 Tarjan algorithm (SCC)	6	8.8 GCD	17
2.2 Kosaraju algorithm (SCC)	7	8.9 LCM	17
2.3 Two Sat	7	8.10 Euclid	17
2.4 Topological sort	7	8.11 Chinese remainder theorem	17
2.5 Cutpoints and Bridges	7	9 Math	17
2.6 Detect a cycle	7	9.1 Progressions	17
2.7 Euler tour for Mo's in a tree	7	9.2 Mod multiplication	17
2.8 Lowest common ancestor (LCA)	7	9.3 Fpow	17
2.9 Guni	8	9.4 Fibonacci	17
2.10 Centroid decomposition	8	10 Geometry	17
2.11 Heavy-light decomposition	8	10.1 Real	17
2.12 Link-Cut tree	8	10.2 Point	17
3 Flows	9	10.3 Angle Between Vectors	18
3.1 Dinic $\mathcal{O}(\min(E \cdot flow, V^2 E))$	9	10.4 Area Polygon	18
3.2 Min cost flow $\mathcal{O}(\min(E \cdot flow, V^2 E))$	10	10.5 Area Polygon In Circle	18
3.3 Hopcroft-Karp $\mathcal{O}(E\sqrt{V})$	10	10.6 Closest Pair Of Points	18
3.4 Hungarian $\mathcal{O}(N^3)$	11	10.7 Convex Hull	19
4 Strings	11	10.8 Distance Point Line	19
4.1 Hash	11	10.9 Get Circle	19
4.2 KMP	11	10.10 Intersects Line	19
4.3 KMP automaton	12	10.11 Intersects Line Segment	19
4.4 Z algorithm	12	10.12 Intersects Segment	19
4.5 Manacher algorithm	12	10.13 Is Convex	19
4.6 Suffix array	12	10.14 Perimeter	19
4.7 Suffix automaton	12	10.15 Point In Convex Polygon logN	19
4.8 Aho corasick	13	10.16 Point In Polygon	20
4.9 Eertree	14	10.17 Point In Segment	20
5 Dynamic Programming	14	10.18 Points Of Tangency	20
5.1 Matrix Chain Multiplication	14	10.19 Projection	20
5.2 Digit DP	14	10.20 Projection Line	20
5.3 Knapsack 0/1	14	10.21 Reflection Line	20
5.4 Convex Hull Trick $\mathcal{O}(n^2) \Rightarrow \mathcal{O}(n)$	14	10.22 Signed Distance Point Line	20
5.5 Divide and conquer $\mathcal{O}(kn^2) \Rightarrow \mathcal{O}(k \cdot n \log n)$	15	10.23 Sort Along Line	20
5.6 Knuth optimization $\mathcal{O}(n^3) \Rightarrow \mathcal{O}(n^2)$	15	10.24 Intersects Line Circle	20
5.7 Do all submasks of a mask	15	11 Bit tricks	20
6 Game Theory	15	11.1 Bitset	20
6.1 Grundy Numbers	15		

Think twice, code once

Template

tem.cpp

```
#pragma GCC optimize("Ofast,unroll-loops,no-stack-protector")
#include <bits/stdc++.h>
using namespace std;

#ifdef LOCAL
#include "debug.h"
#else
#define debug(...)
#endif

#define df(b, e) ((b) > (e))
#define fore(i, b, e) for (auto i = (b) - df(b, e); i != e - df(b, e); i += 1 - 2 * df(b, e))
#define sz(x) int(x.size())
#define all(x) begin(x), end(x)
#define f first
#define s second
#define pb push_back

using lli = long long;
using ld = long double;
using ii = pair<int, int>;
using vi = vector<int>;

int main() {
    cin.tie(0) -> sync_with_stdio(0), cout.tie(0);
    // solve the problem here D:
    return 0;
}

debug.h

template <class A, class B>
ostream & operator << (ostream &os, const pair<A, B> &p) {
    return os << "(" << p.first << ", " << p.second << "
        << ")";
}

template <class A, class B, class C>
basic_ostream<A, B> & operator << (basic_ostream<A, B>
    &os, const C &c) {
    os << "[";
    for (const auto &x : c)
        os << ", " + 2 * (&x == &begin(c)) << x;
    return os << "]";
}

void print(string s) { cout << endl; }

template <class H, class... T>
void print(string s, const H &h, const T&... t) {
    const static string reset = "\033[0m";
    bool ok = 1;
    do {
        if (s[0] == '\0') ok = 0;
        else cout << "\033[1;34m" << s[0] << reset;
        s = s.substr(1);
    } while (s.size() && s[0] != ',');
    if (ok) cout << ": " << "\033[3;95m" << h << reset;
    print(s, t...);
}
```

Randoms

```
mt19937 rng(chrono::steady_clock::now().
    time_since_epoch().count());
uniform_int_distribution<>(l, r)(rng);
```

Fastio

```
char gc() { return getchar_unlocked(); }

void readInt() {}
template <class H, class... T>
void readInt(H &h, T&&... t) {
    char c, s = 1;
    while (isspace(c = gc()));
    if (c == '-') s = -1, c = gc();
    for (h = c - '0'; isdigit(c = gc()); h = h * 10 + c - '0');
    h *= s;
    readInt(t...);
}

void readFloat() {}
template <class H, class... T>
void readFloat(H &h, T&&... t) {
    int c, s = 1, fp = 0, fpl = 1;
    while (isspace(c = gc()));
    if (c == '-') s = -1, c = gc();
    for (h = c - '0'; isdigit(c = gc()); h = h * 10 + c - '0');
    h *= s;
    if (h == '.')
        for (; isdigit(c = gc()); fp = fp * 10 + c - '0', fpl *= 10);
    h += (double)fp / fpl;
    readFloat(t...);
}
```

Compilation (gedit ~/.zshenv)

```
touch a_in{1..9} // make files a_in1, a_in2,..., a_in9
tee {a..m}.cpp < tem.cpp // "" with tem.cpp like base
cat > a_in1 // write on file a_in1
gedit a_in1 // open file a_in1
rm -r a.cpp // deletes file a.cpp :(
```

```
red='\x1B[0;31m'
green='\x1B[0;32m'
noColor='\x1B[0m'
alias flags='-Wall -Wextra -Wshadow -
    D_GLIBCXX_ASSERTIONS -fmax-errors=3 -O2 -w'
go() { g++ --std=c++11 $2 ${flags} $1.cpp && ./a.out }
debug() { go $1 -DLOCAL < $2 }
run() { go $1 "" < $2 }
```

```
random() { // Make small test cases!!!
g++ --std=c++11 $1.cpp -o prog
g++ --std=c++11 gen.cpp -o gen
g++ --std=c++11 brute.cpp -o brute
for ((i = 1; i <= 200; i++)); do
    printf "Test case #$i"
    ./gen > in
    diff -uwi <(. /prog < in) <(. /brute < in) > $1_diff
    if [[ ! $? -eq 0 ]]; then
        printf "${red} Wrong answer ${noColor}\n"
        break
    else
        printf "${green} Accepted ${noColor}\n"
    fi
done
}
```

```
test() {
g++ --std=c++11 $1.cpp -o prog
for ((i = 1; i <= 50; i++)); do
    [[ -f $1_in$i ]] || break
    printf "Test case #$i"
    diff -uwi <(. /prog < $1_in$i) $1_out$i > $1_diff
```

```

if [[ ! $? -eq 0 ]]; then
    printf "${red} Wrong answer ${noColor}\n"
else
    printf "${green} Accepted ${noColor}\n"
fi
done
}

```

Bump allocator

```

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 *) {}

```

Data structures

1.1 Disjoint set with rollback

```

struct Dsu {
    vector<int> pr, tot;
    stack<ii> what;

    Dsu(int n = 0) : pr(n + 5), tot(n + 5, 1) {
        iota(all(pr), 0);
    }

    int find(int u) {
        return pr[u] == u ? u : find(pr[u]);
    }

    void unite(int u, int v) {
        u = find(u), v = find(v);
        if (u == v)
            what.emplace(-1, -1);
        else {
            if (tot[u] < tot[v])
                swap(u, v);
            what.emplace(u, v);
            tot[u] += tot[v];
            pr[v] = u;
        }
    }

    ii rollback() {
        ii last = what.top();
        what.pop();
        int u = last.f, v = last.s;
        if (u != -1) {
            tot[u] -= tot[v];
            pr[v] = v;
        }
        return last;
    }
};

```

1.2 Min-Max queue

```

template <class T>
struct MinQueue : deque< pair<T, int> > {
    // add a element to the right {val, pos}
    void add(T val, int pos) {
        while (!empty() && back().f >= val)
            pop_back();
        emplace_back(val, pos);
    }
    // remove all less than pos
    void rem(int pos) {
        while (front().s < pos)
            pop_front();
    }
};

```

```

T qmin() { return front().f; }
};

```

1.3 Sparse table

```

template <class T, class F = function<T(const T&,
const T&)>>
struct Sparse {
    int n;
    vector<vector<T>> sp;
    F fun;

    Sparse(vector<T> &a, const F &fun) : n(sz(a)), sp(1
+ __lg(n)), fun(fun) {
        sp[0] = a;
        for (int k = 1; (1 << k) <= n; k++) {
            sp[k].resize(n - (1 << k) + 1);
            fore (l, 0, n - (1 << k) + 1) {
                int r = l + (1 << (k - 1));
                sp[k][l] = fun(sp[k - 1][l], sp[k - 1][r]);
            }
        }
    }

    T query(int l, int r) {
        int k = __lg(r - l + 1);
        return fun(sp[k][l], sp[k][r - (1 << k) + 1]);
    }
};

```

1.4 Squirtle decomposition

The perfect block size is *squirtle* of N



```

int blo[N], cnt[N][B], a[N];

void update(int i, int x) {
    cnt[blo[i]][x]--;
    a[i] = x;
    cnt[blo[i]][x]++;
}

int query(int l, int r, int x) {
    int tot = 0;
    while (l <= r)
        if (l % B == 0 && l + B - 1 <= r) {
            tot += cnt[blo[l]][x];
            l += B;
        } else {
            tot += (a[l] == x);
            l++;
        }
    return tot;
}

```

1.5 In-Out trick

```

vector<int> in[N], out[N];
vector<Query> queries;

fore (x, 0, N) {
    for (int i : in[x])
        add(queries[i]);
    // solve
    for (int i : out[x])
        rem(queries[i]);
}

```

1.6 Parallel binary search

```

int lo[Q], hi[Q];
queue<int> solve[N];
vector<Query> queries;

fore (it, 0, 1 + __lg(N)) {

```

```

fore (i, 0, sz(queries))
    if (lo[i] != hi[i]) {
        int mid = (lo[i] + hi[i]) / 2;
        solve[mid].emplace(i);
    }
fore (x, 0, n) {
    // simulate
    while (!solve[x].empty()) {
        int i = solve[x].front();
        solve[x].pop();
        if (can(queries[i]))
            hi[i] = x;
        else
            lo[i] = x + 1;
    }
}

```

1.7 Mo's algorithm

```

vector<Query> queries;
// N = 1e6, so aprox. sqrt(N) +/- C
uniform_int_distribution<int> dis(970, 1030);
const int blo = dis(rng);
sort(all(queries), [&](Query a, Query b) {
    const int ga = a.l / blo, gb = b.l / blo;
    if (ga == gb)
        return (ga & 1) ? a.r < b.r : a.r > b.r;
    return a.l < b.l;
});
int l = queries[0].l, r = l - 1;
for (Query &q : queries) {
    while (r < q.r)
        add(++r);
    while (r > q.r)
        rem(r--);
    while (l < q.l)
        rem(l--);
    while (l > q.l)
        add(--l);
    ans[q.i] = solve();
}

```

To make it faster, change the order to *hilbert(l, r)*

```

lli hilbert(int x, int y, int pw = 21, int rot = 0) {
    if (pw == 0)
        return 0;
    int hpw = 1 << (pw - 1);
    int k = ((x < hpw ? y < hpw ? 0 : 3 : y < hpw ? 1 : 2) + rot) & 3;
    const int d[4] = {3, 0, 0, 1};
    lli a = 1LL << ((pw < 1) - 2);
    lli b = hilbert(x & (x ^ hpw), y & (y ^ hpw), pw - 1, (rot + d[k]) & 3);
    return k * a + (d[k] ? a - b - 1 : b);
}

```

Mo's algorithm with updates in $\mathcal{O}(n^{\frac{5}{3}})$

- Choose a *block* of size $n^{\frac{2}{3}}$
- Do a normal Mo's algorithm, in the *Query* definition add an extra variable for the *updatesSoFar*
- Sort the queries by the order $(l/block, r/block, updatesSoFar)$
- If the update lies inside the current query, update the data structure properly

```

struct Update {
    int pos, prv, nxt;
};

void undo(Update &u) {

```

```

    if (l <= u.pos && u.pos <= r) {
        rem(u.pos);
        a[u.pos] = u.prv;
        add(u.pos);
    } else {
        a[u.pos] = u.prv;
    }
}

```

- Solve the problem :D

```

l = queries[0].l, r = l - 1, upd = sz(updates) - 1;
for (Query &q : queries) {
    while (upd < q.upd)
        dodo(updates[++upd]);
    while (upd > q.upd)
        undo(updates[upd--]);
    // write down the normal Mo's algorithm
}

```

1.8 Ordered tree

```

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;

template <class K, class V = null_type>
using ordered_tree = tree<K, V, less<K>, rb_tree_tag,
    tree_order_statistics_node_update>;
// less_equal<K> for multiset, multimap (?)
#define qrank order_of_key
#define qkth find_by_order

```

1.9 Unordered tree

```

struct chash {
    const uint64_t C = uint64_t(2e18 * 3) + 71;
    const int R = rng();
    uint64_t operator()(uint64_t x) const {
        return __builtin_bswap64((x ^ R) * C);
    };

    template <class K, class V = null_type>
    using unordered_tree = gp_hash_table<K, V, chash>;
}

```

1.10 D-dimensional Fenwick tree

```

template <class T, int ...N>
struct Fenwick {
    T v = 0;
    void update(T v) { this->v += v; }
    T query() { return v; }
};

template <class T, int N, int ...M>
struct Fenwick<T, N, M...> {
    #define lsb(x) (x & -x)
    Fenwick<T, M...> fenw[N + 1];

    template <typename... Args>
    void update(int i, Args... args) {
        for (; i <= N; i += lsb(i))
            fenw[i].update(args...);
    }

    template <typename... Args>
    T query(int l, int r, Args... args) {
        T v = 0;
        for (; r > 0; r -= lsb(r))
            v += fenw[r].query(args...);
        for (--l; l > 0; l -= lsb(l))
            v -= fenw[l].query(args...);
        return v;
    }
}

```

```

    }
};

```

1.11 Dynamic segment tree

```

struct Dyn {
    int l, r;
    lli sum = 0;
    Dyn *L, *R;

    Dyn(int l, int r) : l(l), r(r), L(0), R(0) {}

    void pull() {
        sum = (L ? L->sum : 0);
        sum += (R ? R->sum : 0);
    }

    void update(int p, lli v) {
        if (l == r) {
            sum += v;
            return;
        }
        int m = (l + r) >> 1;
        if (p <= m) {
            if (!L)
                L = new Dyn(l, m);
            L->update(p, v);
        } else {
            if (!R)
                R = new Dyn(m + 1, r);
            R->update(p, v);
        }
        pull();
    }

    lli qsum(int ll, int rr) {
        if (rr < l || r < ll || r < l)
            return 0;
        if (ll <= l && r <= rr)
            return sum;
        int m = (l + r) >> 1;
        return (L ? L->qsum(ll, rr) : 0) +
            (R ? R->qsum(ll, rr) : 0);
    }
};

```

1.12 Persistent segment tree

```

struct Per {
    int l, r;
    lli sum = 0;
    Per *L, *R;

    Per(int l, int r) : l(l), r(r), L(0), R(0) {}

    Per* pull() {
        sum = L->sum + R->sum;
        return this;
    }

    void build() {
        if (l == r)
            return;
        int m = (l + r) >> 1;
        (L = new Per(l, m))->build();
        (R = new Per(m + 1, r))->build();
        pull();
    }

    Per* update(int p, lli v) {
        if (p < l || r < p)
            return this;
    }
};

```

```

Per* t = new Per(l, r);
if (l == r) {
    t->sum = v;
    return t;
}
t->L = L->update(p, v);
t->R = R->update(p, v);
return t->pull();
}

lli qsum(int ll, int rr) {
    if (r < ll || rr < l)
        return 0;
    if (ll <= l && r <= rr)
        return sum;
    return L->qsum(ll, rr) + R->qsum(ll, rr);
}
};

```

1.13 Wavelet tree

```

struct Wav {
    #define iter int * // vector<int>::iterator
    int lo, hi;
    Wav *L, *R;
    vi amt;

    Wav(int lo, int hi) : lo(lo), hi(hi), L(0), R(0) {}

    void build(iter b, iter e) { // array 1-indexed
        if (lo == hi || b == e)
            return;
        amt.reserve(e - b + 1);
        amt.pb(0);
        int m = (lo + hi) >> 1;
        for (auto it = b; it != e; it++)
            amt.pb(amt.back() + (*it <= m));
        auto p = stable_partition(b, e, [=](int x) {
            return x <= m;
        });
        (L = new Wav(lo, m))->build(b, p);
        (R = new Wav(m + 1, hi))->build(p, e);
    }

    int qkth(int l, int r, int k) {
        if (r < l)
            return 0;
        if (lo == hi)
            return lo;
        if (k <= amt[r] - amt[l - 1])
            return L->qkth(amt[l - 1] + 1, amt[r], k);
        return R->qkth(l - amt[l - 1], r - amt[r], k - amt[r] + amt[l - 1]);
    }

    int qleq(int l, int r, int mx) {
        if (r < l || mx < lo)
            return 0;
        if (hi <= mx)
            return r - l + 1;
        return L->qleq(amt[l - 1] + 1, amt[r], mx) +
            R->qleq(l - amt[l - 1], r - amt[r], mx);
    }
};

```

1.14 Li Chao tree

```

struct Fun {
    lli m = 0, c = inf;
    lli operator()(lli x) const { return m * x + c; }
};

```

```

struct LiChao {
    Fun f;
    lli l, r;
    LiChao *L, *R;

    LiChao(lli l, lli r, Fun f = {}) :
        l(l), r(r), f(f), L(0), R(0) {}

    void add(Fun &g) {
        if (f(l) <= g(l) && f(r) <= g(r))
            return;
        if (g(l) < f(l) && g(r) < f(r)) {
            f = g;
            return;
        }
        lli m = (l + r) >> 1;
        if (g(m) < f(m))
            swap(f, g);
        if (g(l) <= f(l))
            L = L ? (L->add(g), L) : new LiChao(l, m, g);
        else
            R = R ? (R->add(g), R) : new LiChao(m + 1, r, g);
    }

    lli query(lli x) {
        if (l == r)
            return f(x);
        lli m = (l + r) >> 1;
        if (x <= m)
            return min(f(x), L ? L->query(x) : inf);
        return min(f(x), R ? R->query(x) : inf);
    }
};

```

1.15 Treap

```

typedef struct Node* Treap;
struct Node {
    uint32_t pri = rng();
    int val;
    Treap ch[2] = {0, 0};
    int sz = 1, flip = 0;
    Node(int val) : val(val) {}
};

void push(Treap t) {
    if (!t)
        return;
    if (t->flip) {
        swap(t->ch[0], t->ch[1]);
        for (Treap ch : t->ch) if (ch)
            ch->flip ^= 1;
        t->flip = 0;
    }
}

Treap pull(Treap t) {
    #define gsz(t) (t ? t->sz : 0)
    t->sz = 1;
    for (Treap ch : t->ch)
        push(ch), t->sz += gsz(ch);
    return t;
}

pair<Treap, Treap> split(Treap t, int val) {
    // <= val goes to the left, > val to the right
    if (!t)
        return {t, t};
    push(t);
    if (val < t->val) {
        auto p = split(t->ch[0], val);

```

```

        t->ch[0] = p.s;
        return {p.f, pull(t)};
    }
    auto p = split(t->ch[1], val);
    t->ch[1] = p.f;
    return {pull(t), p.s};
}

pair<Treap, Treap> splitsz(Treap t, int sz) {
    // <= sz goes to the left, > sz to the right
    if (!t)
        return {t, t};
    push(t);
    if (sz <= gsz(t->ch[0])) {
        auto p = splitsz(t->ch[0], sz);
        t->ch[0] = p.s;
        return {p.f, pull(t)};
    }
    auto p = splitsz(t->ch[1], sz - gsz(t->ch[0]) - 1);
    t->ch[1] = p.f;
    return {pull(t), p.s};
}

Treap merge(Treap l, Treap r) {
    if (!l || !r)
        return l ? l : r;
    push(l), push(r);
    if (l->pri > r->pri)
        return l->ch[1] = merge(l->ch[1], r), pull(l);
    else
        return r->ch[0] = merge(l, r->ch[0]), pull(r);
}

Treap qkth(Treap t, int k) { // 0-indexed
    if (!t)
        return t;
    push(t);
    int sz = gsz(t->ch[0]);
    if (sz == k)
        return t;
    return k < sz ? qkth(t->ch[0], k) : qkth(t->ch[1], k - sz - 1);
}

int qrank(Treap t, int val) { // 0-indexed
    if (!t)
        return -1;
    push(t);
    if (val < t->val)
        return qrank(t->ch[0], val);
    if (t->val == val)
        return gsz(t->ch[0]);
    return gsz(t->ch[0]) + qrank(t->ch[1], val) + 1;
}

Treap insert(Treap t, int val) {
    auto p1 = split(t, val);
    auto p2 = split(p1.f, val - 1);
    return merge(p2.f, merge(new Node(val), p1.s));
}

Treap erase(Treap t, int val) {
    auto p1 = split(t, val);
    auto p2 = split(p1.f, val - 1);
    return merge(p2.f, p1.s);
}

2 Graphs
2.1 Tarjan algorithm (SCC)
vector<vi> scc;

```

```

int tin[N], fup[N];
bitset<N> still;
stack<int> stk;
int timer = 0;

void tarjan(int u) {
    tin[u] = fup[u] = ++timer;
    still[u] = true;
    stk.push(u);
    for (int v : graph[u]) {
        if (!tin[v])
            tarjan(v);
        if (still[v])
            fup[u] = min(fup[u], fup[v]);
    }
    if (fup[u] == tin[u]) {
        scc.pb({});
        int v;
        do {
            v = stk.top();
            stk.pop();
            still[v] = false;
            scc.back().pb(v);
        } while (v != u);
    }
}

```

2.2 Kosaraju algorithm (SCC)

```

int scc[N], k = 0;
char vis[N];
vi order;

void dfs1(int u) {
    vis[u] = 1;
    for (int v : graph[u])
        if (vis[v] != 1)
            dfs1(v);
    order.pb(u);
}

void dfs2(int u, int k) {
    vis[u] = 2, scc[u] = k;
    for (int v : rgraph[u]) // reverse graph
        if (vis[v] != 2)
            dfs2(v, k);
}

void kosaraju() {
    for (int u = 1, n + 1)
        if (vis[u] != 1)
            dfs1(u);
    reverse(all(order));
    for (int u : order)
        if (vis[u] != 2)
            dfs2(u, ++k);
}

```

2.3 Two Sat

```

void add(int u, int v) {
    graph[u].pb(v);
    rgraph[v].pb(u);
}

void implication(int u, int v) {
    #define neg(u) ((n) + (u))
    add(u, v);
    add(neg(v), neg(u));
}

pair<bool, vi> satisfy(int n) {

```

```

    kosaraju(2 * n); // size of the two-sat is 2 * n
    vi ans(n + 1, 0);
    for (int u = 1, n + 1) {
        if (scc[u] == scc[neg(u)])
            return {0, ans};
        ans[u] = scc[u] > scc[neg(u)];
    }
    return {1, ans};
}

```

2.4 Topological sort

2.5 Cutpoints and Bridges

```

int tin[N], fup[N], timer = 0;

void findWeakness(int u, int p = 0) {
    tin[u] = fup[u] = ++timer;
    int children = 0;
    for (int v : graph[u]) if (v != p) {
        if (!tin[v]) {
            ++children;
            findWeakness(v, u);
            fup[u] = min(fup[u], fup[v]);
            if (fup[v] >= tin[u] && p) // u is a cutpoint
                if (fup[v] > tin[u]) // bridge u -> v
                    ;
        }
        fup[u] = min(fup[u], tin[v]);
    }
    if (!p && children > 1) // u is a cutpoint
        ;
}

```

2.6 Detect a cycle

```

bool cycle(int u) {
    vis[u] = 1;
    for (int v : graph[u]) {
        if (vis[v] == 1)
            return true;
        if (!vis[v] && cycle(v))
            return true;
    }
    vis[u] = 2;
    return false;
}

```

2.7 Euler tour for Mo's in a tree

Mo's in a tree, extended euler tour $tin[u] = ++timer$, $tout[u] = ++timer$

- $u = lca(u, v)$, $query(tin[u], tin[v])$
- $u \neq lca(u, v)$, $query(tout[u], tin[v]) + query(tin[lca], tin[lca])$

2.8 Lowest common ancestor (LCA)

```

const int LogN = 1 + __lg(N);
int pr[LogN][N], dep[N];

void dfs(int u, int pr[]) {
    for (int v : graph[u])
        if (v != pr[u]) {
            pr[v] = u;
            dep[v] = dep[u] + 1;
            dfs(v, pr);
        }
}

int lca(int u, int v) {
    if (dep[u] > dep[v])
        swap(u, v);
    for (int k = LogN, 0)
        if (dep[v] - dep[u] >= (1 << k))
            v = pr[k][v];
    if (u == v)
        return u;
}

```



```

    fore (k, LogN, 0)
        if (pr[k][v] != pr[k][u])
            u = pr[k][u], v = pr[k][v];
    return pr[0][u];
}

int dist(int u, int v) {
    return dep[u] + dep[v] - 2 * dep[lca(u, v)];
}

```

```

void init(int r) {
    dfs(r, pr[0]);
    fore (k, 1, LogN)
        fore (u, 1, n + 1)
            pr[k][u] = pr[k - 1][pr[k - 1][u]];
}

```

2.9 Guni

```

int tin[N], tout[N], who[N], sz[N], heavy[N], color[N];
int timer = 0;

int dfs(int u, int pr = 0) {
    sz[u] = 1, tin[u] = ++timer, who[timer] = u;
    for (int v : graph[u]) if (v != pr) {
        sz[u] += dfs(v, u);
        if (sz[v] > sz[heavy[u]])
            heavy[u] = v;
    }
    return tout[u] = timer, sz[u];
}

```

```

void guni(int u, int pr = 0, bool keep = 0) {
    for (int v : graph[u])
        if (v != pr && v != heavy[u])
            guni(v, u, 0);
    if (heavy[u])
        guni(heavy[u], u, 1);
    for (int v : graph[u])
        if (v != pr && v != heavy[u])
            fore (i, tin[v], tout[v] + 1)
                add(color[who[i]]);
    add(color[u]);
    // Solve the subtree queries here
    if (keep == 0)
        fore (i, tin[u], tout[u] + 1)
            rem(color[who[i]]);
}

```

2.10 Centroid decomposition

```

int cdp[N], sz[N];
bitset<N> rem;

int dfsz(int u, int p = 0) {
    sz[u] = 1;
    for (int v : graph[u])
        if (v != p && !rem[v])
            sz[u] += dfsz(v, u);
    return sz[u];
}

int centroid(int u, int n, int p = 0) {
    for (int v : graph[u])
        if (v != p && !rem[v] && 2 * sz[v] > n)
            return centroid(v, n, u);
    return u;
}

void solve(int u, int p = 0) {
    cdp[u] = centroid(u, dfsz(u)) = p;
    rem[u] = true;
}

```

```

for (int v : graph[u])
    if (!rem[v])
        solve(v, u);
}

```

2.11 Heavy-light decomposition

```

int pr[N], dep[N], sz[N], heavy[N], head[N], pos[N],
    who[N], timer = 0;
Lazy* tree; // generally a lazy segtree

```

```

int dfs(int u) {
    sz[u] = 1, heavy[u] = head[u] = 0;
    for (int v : graph[u]) if (v != pr[u]) {
        pr[v] = u;
        dep[v] = dep[u] + 1;
        sz[u] += dfs(v);
        if (sz[v] > sz[heavy[u]])
            heavy[u] = v;
    }
    return sz[u];
}

```

```

void hld(int u, int h) {
    head[u] = h, pos[u] = ++timer, who[timer] = u;
    if (heavy[u] != 0)
        hld(heavy[u], h);
    for (int v : graph[u])
        if (v != pr[u] && v != heavy[u])
            hld(v, v);
}

```

```

template <class F>
void processPath(int u, int v, F fun) {
    for (; head[u] != head[v]; v = pr[head[v]]) {
        if (dep[head[u]] > dep[head[v]])
            swap(u, v);
        fun(pos[head[v]], pos[v]);
    }
    if (dep[u] > dep[v])
        swap(u, v);
    if (u != v)
        fun(pos[heavy[u]], pos[v]);
    fun(pos[u], pos[u]); // process lca(u, v) too?
}

```

```

void updatePath(int u, int v, lli z) {
    processPath(u, v, [&](int l, int r) {
        tree->update(l, r, z);
    });
}

```

```

lli queryPath(int u, int v) {
    lli sum = 0;
    processPath(u, v, [&](int l, int r) {
        sum += tree->qsum(l, r);
    });
    return sum;
}

```

2.12 Link-Cut tree

```

typedef struct Node* Splay;
struct Node {
    int val, mx = 0;
    Splay ch[2] = {0, 0}, p = 0;
    int sz = 1, flip = 0;
    Node(int val) : val(val), mx(val) {}
};

void push(Splay u) {
    if (!u || !u->flip)
        return;
}

```

```

swap(u->ch[0], u->ch[1]);
for (Splay v : u->ch)
    if (v) v->flip ^= 1;
u->flip = 0;
}

void pull(Splay u) {
#define gsz(t) (t ? t->sz : 0)
u->sz = 1, u->mx = u->val;
for (Splay v : u->ch) if (v) {
    push(v);
    u->sz += gsz(v);
    u->mx = max(u->mx, v->mx);
}
}

int dir(Splay u) {
    if (!u->p) return -2; // root of the LCT component
    if (u->p->ch[0] == u) return 0; // left child
    if (u->p->ch[1] == u) return 1; // right child
    return -1; // root of current splay tree
}

void add(Splay u, Splay v, int d) {
    if (v) v->p = u;
    if (d >= 0) u->ch[d] = v;
}

void rot(Splay u) { // assume p and p->p propagated
    int x = dir(u);
    Splay g = u->p;
    add(g->p, u, dir(g));
    add(g, u->ch[x ^ 1], x);
    add(u, g, x ^ 1);
    pull(g), pull(u);
}

void splay(Splay u) {
#define isRoot(u) (dir(u) < 0)
while (!isRoot(u) && !isRoot(u->p)) {
    push(u->p->p), push(u->p), push(u);
    rot(dir(u) == dir(u->p) ? u->p : u);
    rot(u);
}
if (!isRoot(u)) push(u->p), push(u), rot(u);
push(u);
}

// puts u on the preferred path, splay (right subtree
// is empty)
void access(Splay u) {
    for (Splay v = u, last = 0; v; v = v->p) {
        splay(v); // now switch virtual children, i don't
        // know what this means!!
        // if (last) v->vsub -= last->sub;
        // if (v->ch[1]) v->vsub += v->ch[1]->sub;
        v->ch[1] = last, pull(v), last = v;
    }
    splay(u);
}

void rootify(Splay u) {
    access(u), u->flip ^= 1, access(u);
}

Splay lca(Splay u, Splay v) {
    if (u == v) return u;
    access(u), access(v);
    if (!u->p) return 0;
    return splay(u), u->p ? u;
}

```

```

}

bool connected(Splay u, Splay v) {
    return lca(u, v);
}

void link(Splay u, Splay v) { // make u parent of v,
    // make sure they aren't connected
    if (!connected(u, v)) {
        rootify(v), access(u);
        add(v, u, 0), pull(v);
    }
}

void cut(Splay u) { // cut u from its parent
    access(u);
    u->ch[0] = u->ch[0]->p = 0;
    pull(u);
}

void cut(Splay u, Splay v) { // if u, v adj in tree
    rootify(u), access(v), cut(v);
}

Splay getRoot(Splay u) {
    access(u);
    while (u->ch[0])
        u = u->ch[0], push(u);
    return access(u), u;
}

Splay lift(Splay u, int k) {
    push(u);
    int sz = gsz(u->ch[0]);
    if (sz == k)
        return splay(u), u;
    return k < sz ? lift(u->ch[0], k) : lift(u->ch[1], k
        - sz - 1);
}

Splay ancestor(Splay u, int k) {
    return access(u), lift(u, gsz(u->ch[0]) - k);
}

Splay query(Splay u, Splay v) {
    return rootify(u), access(v), v;
}

Splay lct[N];

```

3 Flows

3.1 Dinic $\mathcal{O}(\min(E \cdot \text{flow}, V^2 E))$

If the network is massive, try to compress it by looking for patterns. Dinic with scaling works in $\mathcal{O}(EV \cdot \log(\max Cap))$.

```

template <class F>
struct Dinic {
    struct Edge {
        int v, inv;
        F cap, flow;
        Edge(int v, F cap, int inv) :
            v(v), cap(cap), flow(0), inv(inv){}
    };

    F eps = (F) 1e-9, lim = (F) 1e-9;
    const bool scaling = 0;
    int s, t, n, m = 0;
    vector< vector<Edge> > g;
    vi dist, ptr;

    Dinic(int n, int ss = -1, int tt = -1) :

```

```

n(n), g(n + 5), dist(n + 5), ptr(n + 5) {
    s = ss == -1 ? n + 1 : ss;
    t = tt == -1 ? n + 2 : tt;
}

void add(int u, int v, F cap) {
    g[u].pb(Edge(v, cap, sz(g[v])));
    g[v].pb(Edge(u, 0, sz(g[u]) - 1));
    lim = (scaling ? max(lim, cap) : lim);
    m += 2;
}

bool bfs() {
    fill(all(dist), -1);
    queue<int> qu({s});
    dist[s] = 0;
    while (sz(qu) && dist[t] == -1) {
        int u = qu.front(); qu.pop();
        for (Edge &e : g[u]) if (dist[e.v] == -1)
            if (scaling ? e.cap - e.flow >= lim : e.cap -
                e.flow > eps) {
                dist[e.v] = dist[u] + 1;
                qu.push(e.v);
            }
    }
    return dist[t] != -1;
}

F dfs(int u, F flow = numeric_limits<F>::max()) {
    if (flow <= eps || u == t)
        return max<F>(0, flow);
    for (int &i = ptr[u]; i < sz(g[u]); i++) {
        Edge &e = g[u][i];
        if (e.cap - e.flow > eps && dist[u] + 1 == dist[
            e.v]) {
            F pushed = dfs(e.v, min<F>(flow, e.cap - e.flow
                ));
            if (pushed > eps) {
                e.flow += pushed;
                g[e.v][e.inv].flow -= pushed;
                return pushed;
            }
        }
    }
    return 0;
}

F maxFlow() {
    F flow = 0;
    for (lim = scaling ? lim : 1; lim > eps; lim /= 2)
        while (bfs()) {
            fill(all(ptr), 0);
            while (F pushed = dfs(s))
                flow += pushed;
        }
    return flow;
}
};

```

3.2 Min cost flow $\mathcal{O}(\min(E \cdot \text{flow}, V^2 E))$

If the network is massive, try to compress it by looking for patterns.

```

template <class C, class F>
struct MCMF {
    static constexpr F eps = (F) 1e-9;
    struct Edge {
        int u, v, inv;
        F cap, flow;
        C cost;
        Edge(int u, int v, C cost, F cap, int inv) :
            u(u), v(v), cost(cost), cap(cap), flow(0), inv(

```

```

            inv) {}
    };

    int s, t, n, m = 0;
    vector< vector<Edge> > g;
    vector<Edge*> prev;
    vector<C> cost;
    vi state;

    MCMF(int n, int ss = -1, int tt = -1):
        n(n), g(n + 5), cost(n + 5), state(n + 5), prev(n
            + 5) {
        s = ss == -1 ? n + 1 : ss;
        t = tt == -1 ? n + 2 : tt;
    }

    void add(int u, int v, C cost, F cap) {
        g[u].pb(Edge(u, v, cost, cap, sz(g[v])));
        g[v].pb(Edge(v, u, -cost, 0, sz(g[u]) - 1));
        m += 2;
    }

    bool bfs() {
        fill(all(state), 0);
        fill(all(cost), numeric_limits<C>::max());
        deque<int> qu;
        qu.push_back(s);
        state[s] = 1, cost[s] = 0;
        while (sz(qu)) {
            int u = qu.front(); qu.pop_front();
            state[u] = 2;
            for (Edge &e : g[u]) if (e.cap - e.flow > eps)
                if (cost[u] + e.cost < cost[e.v]) {
                    cost[e.v] = cost[u] + e.cost;
                    prev[e.v] = &e;
                    if (state[e.v] == 2 || (sz(qu) && cost[qu.
                        front()] > cost[e.v]))
                        qu.push_front(e.v);
                    else if (state[e.v] == 0)
                        qu.push_back(e.v);
                    state[e.v] = 1;
                }
        }
        return cost[t] != numeric_limits<C>::max();
    }

    pair<C, F> minCostFlow() {
        C cost = 0; F flow = 0;
        while (bfs()) {
            F pushed = numeric_limits<F>::max();
            for (Edge* e = prev[t]; e != nullptr; e = prev[e
                ->u])
                pushed = min(pushed, e->cap - e->flow);
            for (Edge* e = prev[t]; e != nullptr; e = prev[e
                ->u]) {
                e->flow += pushed;
                g[e->v][e->inv].flow -= pushed;
                cost += e->cost * pushed;
            }
            flow += pushed;
        }
        return make_pair(cost, flow);
    }
};

```

3.3 Hopcroft-Karp $\mathcal{O}(E\sqrt{V})$

```

struct HopcroftKarp {
    int n, m = 0;
    vector<vi> g;
    vi dist, match;

```

```

HopcroftKarp(int _n) : n(5 + _n), g(n + 5), dist(n + 5), match(n + 5, 0) {}

void add(int u, int v) {
    g[u].pb(v), g[v].pb(u);
    m += 2;
}

bool bfs() {
    queue<int> qu;
    fill(all(dist), -1);
    for (u, 1, n + 1)
        if (!match[u])
            dist[u] = 0, qu.push(u);
    while (!qu.empty()) {
        int u = qu.front(); qu.pop();
        for (int v : g[u])
            if (dist[match[v]] == -1) {
                dist[match[v]] = dist[u] + 1;
                if (match[v])
                    qu.push(match[v]);
            }
    }
    return dist[0] != -1;
}

bool dfs(int u) {
    for (int v : g[u])
        if (!match[v] || (dist[u] + 1 == dist[match[v]]
            && dfs(match[v]))) {
            match[u] = v, match[v] = u;
            return 1;
        }
    dist[u] = 1 << 30;
    return 0;
}

int maxMatching() {
    int tot = 0;
    while (bfs())
        for (u, 1, n + 1)
            tot += match[u] ? 0 : dfs(u);
    return tot;
}
};

```

3.4 Hungarian $\mathcal{O}(N^3)$

n jobs, m people

```

template <class C>
pair<C, vi> Hungarian(vector< vector<C> > &a) {
    int n = sz(a), m = sz(a[0]), p, q, j, k; // n <= m
    vector<C> fx(n, numeric_limits<C>::min()), fy(m, 0);
    vi x(n, -1), y(m, -1);
    for (i, 0, n)
        for (j, 0, m)
            fx[i] = max(fx[i], a[i][j]);
    for (i, 0, n) {
        vi t(m, -1), s(n + 1, i);
        for (p = q = 0; p <= q && x[i] < 0; p++)
            for (k = s[p], j = 0; j < m && x[i] < 0; j++)
                if (abs(fx[k] + fy[j] - a[k][j]) < eps && t[j] < 0) {
                    s[++q] = y[j], t[j] = k;
                    if (s[q] < 0) for (p = j; p >= 0; j = p)
                        y[j] = k = t[j], p = x[k], x[k] = j;
                }
        if (x[i] < 0) {
            C d = numeric_limits<C>::max();
            for (k, 0, q + 1)
                for (j, 0, m) if (t[j] < 0)

```

```

                    d = min(d, fx[s[k]] + fy[j] - a[s[k]][j]);
            for (j, 0, m)
                fy[j] += (t[j] < 0 ? 0 : d);
            for (k, 0, q + 1)
                fx[s[k]] -= d;
            i--;
        }
    }
    C cost = 0;
    for (i, 0, n) cost += a[i][x[i]];
    return make_pair(cost, x);
}

```

4 Strings

4.1 Hash

```

vi p = {10006793, 1777771, 10101283, 10101823, 1013635
        9, 10157387, 10166249};
vi mod = {999992867, 107077777, 999727999, 1000008223
          , 1000009999, 1000003211, 1000027163, 1000002193,
          1000000123};
int pw[2][N];

struct Hash {
    vector<vi> h;

    Hash(string &s) : h(2, vi(sz(s) + 1, 0)) {
        for (i, 0, 2)
            for (j, 0, sz(s))
                h[i][j + 1] = (h[i][j] + lli(s[j] - 'a' + 1) *
                    pw[i][j]) % mod[i];
    }

    lli query(int l, int r) {
        lli f[2] = {0, 0};
        for (i, 0, 2)
            f[i] = 1LL * (h[i][r + 1] - h[i][l] + mod[i]) %
                mod[i] * pw[i][N - l - 1] % mod[i];
        return (f[0] << 32) | f[1];
    }
};

shuffle(all(p), rng), shuffle(all(mod), rng);
for (i, 0, 2) {
    pw[i][0] = 1LL;
    for (j, 1, N)
        pw[i][j] = 1LL * p[i] * pw[i][j - 1] % mod[i];
}

```

4.2 KMP

period = $n - p[n - 1]$, period(abcabc) = 3, $n \bmod \text{period} \equiv 0$

```

vi lps(string &s) {
    vi p(sz(s), 0);
    int j = 0;
    for (i, 1, sz(s)) {
        while (j && s[i] != s[j])
            j = p[j - 1];
        if (s[i] == s[j])
            j++;
        p[i] = j;
    }
    return p;
}

// how many times t occurs in s
int kmp(string &s, string &t) {
    vi p = lps(t);
    int j = 0, tot = 0;
    for (i, 0, sz(s)) {
        while (j && s[i] != t[j])
            j = p[j - 1];
        if (s[i] == t[j])
            j++;
    }
}

```

```

    if (j == sz(t))
        tot++; // pos: i - sz(t) + 1;
}
return tot;
}

```

4.3 KMP automaton

```

int go[N][A];

void kmpAutomaton(string &s) {
    s += "$";
    vi p = lps(s);
    for (i, 0, sz(s))
        for (c, 0, A) {
            if (i && s[i] != 'a' + c)
                go[i][c] = go[p[i - 1]][c];
            else
                go[i][c] = i + ('a' + c == s[i]);
        }
    s.pop_back();
}

```

4.4 Z algorithm

```

vi zf(string &s) {
    vi z(sz(s), 0);
    for (int i = 1, l = 0, r = 0; i < sz(s); i++) {
        if (i <= r)
            z[i] = min(r - i + 1, z[i - l]);
        while (i + z[i] < sz(s) && s[i + z[i]] == s[z[i]])
            ++z[i];
        if (i + z[i] - 1 > r)
            l = i, r = i + z[i] - 1;
    }
    return z;
}

```

4.5 Manacher algorithm

```

vector<vi> manacher(string &s) {
    vector<vi> pal(2, vi(sz(s), 0));
    for (k, 0, 2) {
        int l = 0, r = 0;
        for (i, 0, sz(s)) {
            int t = r - i + !k;
            if (i < r)
                pal[k][i] = min(t, pal[k][l + t]);
            int p = i - pal[k][i], q = i + pal[k][i] - !k;
            while (p >= 1 && q + 1 < sz(s) && s[p - 1] == s[q + 1])
                ++pal[k][i], --p, ++q;
            if (q > r)
                l = p, r = q;
        }
    }
    return pal;
}

```

4.6 Suffix array

- Duplicates $\sum_{i=1}^n lcp[i]$
- Longest Common Substring of various strings
Add *notUsed* characters between strings, i.e. $a+\$+b+\#+c$
Use two-pointers to find a range $[l, r]$ such that all *notUsed* characters are present, then $query(lcp[l + 1], \dots, lcp[r])$ for that window is the common length.

```

struct SuffixArray {
    int n;
    string s;
    vi sa, lcp;

    SuffixArray(string &s) : n(sz(s) + 1), s(s), sa(n),
        lcp(n) {
        vi top(max(256, n)), rk(n);

```

```

        for (i, 0, n)
            top[rk[i]] = s[i] & 255;
        partial_sum(all(top), top.begin());
        for (i, 0, n)
            sa[--top[rk[i]]] = i;
        vi sb(n);
        for (int len = 1, j; len < n; len <= 1) {
            for (i, 0, n) {
                j = (sa[i] - len + n) % n;
                sb[top[rk[j]]++] = j;
            }
            sa[sb[top[0]]] = j = 0;
            for (i, 1, n) {
                if (rk[sb[i]] != rk[sb[i - 1]] || rk[sb[i] + len] != rk[sb[i - 1] + len])
                    top[++j] = i;
                sa[sb[i]] = j;
            }
            copy(all(sa), rk.begin());
            copy(all(sb), sa.begin());
            if (j >= n - 1)
                break;
        }
        for (int i = 0, j = rk[lcp[0] = 0], k = 0; i < n - 1; i++, k++)
            while (k >= 0 && s[i] != s[sa[j - 1] + k])
                lcp[j] = k--, j = rk[sa[j] + 1];
    }

    char at(int i, int j) {
        int k = sa[i] + j;
        return k < n ? s[k] : 'z' + 1;
    }

    bool count(string &t) {
        ii lo(0, n - 1), hi(0, n - 1);
        for (i, 0, sz(t)) {
            while (lo.f + 1 < lo.s) {
                int mid = (lo.f + lo.s) / 2;
                (at(mid, i) < t[i] ? lo.f : lo.s) = mid;
            }
            while (hi.f + 1 < hi.s) {
                int mid = (hi.f + hi.s) / 2;
                (t[i] < at(mid, i) ? hi.s : hi.f) = mid;
            }
            int p1 = (at(lo.f, i) == t[i] ? lo.f : lo.s);
            int p2 = (at(hi.s, i) == t[i] ? hi.s : hi.f);
            if (at(p1, i) != t[i] || at(p2, i) != t[i] || p1 > p2)
                return 0;
            lo = hi = ii(p1, p2);
        }
        return lo.s - lo.f + 1;
    }
};

```

4.7 Suffix automaton

- $len[u] - len[link[u]] = \text{distinct strings}$
- Number of different substrings (dp)

$$diff(u) = 1 + \sum_{v \in trie[u]} diff(v)$$

- Total length of all different substrings (2 x dp)

$$totLen(u) = \sum_{v \in trie[u]} diff(v) + totLen(v)$$

- Leftmost occurrence $pos[u] = len[u] - 1$
if is **clone** then $pos[clone] = pos[q]$
- All occurrence positions

- Smallest cyclic shift
Construct sam of $s + s$, find the lexicographically smallest path of $sz(s)$
- Shortest non-appearing string

$$nonAppearing(u) = \min_{v \in trie[u]} nonAppearing(v) + 1$$

```

struct SuffixAutomaton {
    vector< map<char, int> > trie;
    vi link, len;
    int last;

    SuffixAutomaton() { last = newNode(); }

    int newNode() {
        trie.pb({}), link.pb(-1), len.pb(0);
        return sz(trie) - 1;
    }

    void extend(char c) {
        int u = newNode();
        len[u] = len[last] + 1;
        int p = last;
        while (p != -1 && !trie[p].count(c)) {
            trie[p][c] = u;
            p = link[p];
        }
        if (p == -1)
            link[u] = 0;
        else {
            int q = trie[p][c];
            if (len[p] + 1 == len[q])
                link[u] = q;
            else {
                int clone = newNode();
                len[clone] = len[p] + 1;
                trie[clone] = trie[q];
                link[clone] = link[q];
                while (p != -1 && trie[p][c] == q) {
                    trie[p][c] = clone;
                    p = link[p];
                }
                link[q] = link[u] = clone;
            }
        }
        last = u;
    }

    string qkthSubstring(lli kth, int u = 0) {
        // number of different substrings (dp)
        string s = "";
        while (kth > 0)
            for (auto &[c, v] : trie[u]) {
                if (kth <= diff(v)) {
                    s.pb(c), kth--, u = v;
                    break;
                }
            }
            kth -= diff(v);
        return s;
    }

    void occurs() {
        // occ[u] = 1, occ[clone] = 0
        vi who;
        for (u, 1, sz(trie))
            who.pb(u);
        sort(all(who), [&](int u, int v) {
            return len[u] > len[v];
        });
        for (int u : who)

```

```

        occ[link[u]] += occ[u];
    }

    int queryOccurrences(string &s, int u = 0) {
        for (char c : s) {
            if (!trie[u].count(c))
                return 0;
            u = trie[u][c];
        }
        return occ[u];
    }

    int longestCommonSubstring(string &s, int u = 0) {
        int mx = 0, clen = 0;
        for (char c : s) {
            while (u && !trie[u].count(c)) {
                u = link[u];
                clen = len[u];
            }
            if (trie[u].count(c))
                u = trie[u][c], clen++;
            mx = max(mx, clen);
        }
        return mx;
    }

    string smallestCyclicShift(int n, int u = 0) {
        string s = "";
        for (i, 0, n) {
            char c = trie[u].begin()->f;
            s += c;
            u = trie[u][c];
        }
        return s;
    }

    int leftmost(string &s, int u = 0) {
        for (char c : s) {
            if (!trie[u].count(c))
                return -1;
            u = trie[u][c];
        }
        return pos[u] - sz(s) + 1;
    }
};

```

4.8 Aho corasick

```

struct AhoCorasick {
    vector< map<char, int> > trie;
    vi link, cnt;

    AhoCorasick() { newNode(); }

    int newNode() {
        trie.pb({}), link.pb(0), cnt.pb(0);
        return sz(trie) - 1;
    }

    void insert(string &s, int u = 0) {
        for (char c : s) {
            if (!trie[u][c])
                trie[u][c] = newNode();
            u = trie[u][c];
        }
        cnt[u]++;
    }

    int go(int u, char c) {
        while (u && !trie[u].count(c))
            u = link[u];
        return trie[u][c];
    }
}

```

```

}

void pushLinks() {
    queue<int> qu;
    qu.push(0);
    while (!qu.empty()) {
        int u = qu.front();
        qu.pop();
        for (auto &c, v : trie[u]) {
            if (v == 0) {
                v = trie[link[u]][c];
                continue;
            }
            link[v] = u ? go(link[u], c) : 0;
            cnt[v] += cnt[link[v]];
            qu.push(v);
        }
    }
}

```

```

int match(string &s, int u = 0) {
    int ans = 0;
    for (char c : s)
        u = go(u, c), ans += cnt[u];
    return ans;
}
};

```

4.9 Eertree

```

struct Eertree {
    vector< map<char, int> > trie;
    vi link, len;
    string s = "$";
    int last;

    Eertree() {
        last = newNode(), newNode();
        link[0] = 1, len[1] = -1;
    }

    int newNode() {
        trie.pb({}), link.pb(0), len.pb(0);
        return sz(trie) - 1;
    }

    int go(int u) {
        while (s[sz(s) - len[u] - 2] != s.back())
            u = link[u];
        return u;
    }

    void extend(char c) {
        s += c;
        int u = go(last);
        if (!trie[u][c]) {
            int v = newNode();
            len[v] = len[u] + 2;
            link[v] = trie[go(link[u])][c];
            trie[u][c] = v;
        }
        last = trie[u][c];
    }
};

```

5 Dynamic Programming

5.1 Matrix Chain Multiplication

```

int dp(int l, int r) {
    if (l > r)
        return 0LL;
    int &ans = mem[l][r];

```

```

    if (!done[l][r]) {
        done[l][r] = true, ans = inf;
        for (k, l, r + 1) // split in [l, k] [k + 1, r]
            ans = min(ans, dp(l, k) + dp(k + 1, r) + add);
    }
    return ans;
}

```

5.2 Digit DP

Counts the amount of numbers in $[l, r]$ such are divisible by k . (flag *nonzero* is for different lengths)

It can be reduced to $dp(i, x, small)$, and has to be solve like $f(r) - f(l - 1)$

```

#define state [i][x][small][big][nonzero]
int dp(int i, int x, bool small, bool big, bool
    nonzero) {
    if (i == sz(r))
        return x % k == 0 && nonzero;
    int &ans = mem state;
    if (done state != timer) {
        done state = timer;
        ans = 0;
        int lo = small ? 0 : l[i] - '0';
        int hi = big ? 9 : r[i] - '0';
        for (y, lo, max(lo, hi) + 1) {
            bool small2 = small | (y > lo);
            bool big2 = big | (y < hi);
            bool nonzero2 = nonzero | (x > 0);
            ans += dp(i + 1, (x * 10 + y) % k, small2, big2,
                nonzero2);
        }
    }
    return ans;
}

```

5.3 Knapsack 0/1

```

for (auto &cur : items)
    for (w, W + 1, cur.w) // [cur.w, W]
        umax(dp[w], dp[w - cur.w] + cur.cost);

```

5.4 Convex Hull Trick $\mathcal{O}(n^2) \Rightarrow \mathcal{O}(n)$

$dp[i] = \min_{j < i} (dp[j] + b[j] * a[i])$
 $dp[i][j] = \min_{k < j} (dp[i - 1][k] + b[k] * a[j])$
 $b[j] \geq b[j + 1]$ optionally $a[i] \leq a[i + 1]$

// for doubles, use $inf = 1/.0$, $div(a,b) = a / b$

```

struct Line {
    mutable lli m, c, p;
    bool operator < (const Line &l) const { return m < l
        .m; }
    bool operator < (lli x) const { return p < x; }
    lli operator ()(lli x) const { return m * x + c; }
};

```

```

struct DynamicHull : multiset<Line, less<>> {
    lli div(lli a, lli b) {
        return a / b - ((a ^ b) < 0 && a % b);
    }
}

```

```

bool isect(iterator x, iterator y) {
    if (y == end())
        return x->p = inf, 0;
    if (x->m == y->m)
        x->p = (x->c > y->c ? inf : -inf);
    else
        x->p = div(x->c - y->c, y->m - x->m);
    return x->p >= y->p;
}

```

```

void add(lli m, lli c) {
    auto z = insert({m, c, 0}), y = z++, x = y;

```

```

while (isect(y, z)) z = erase(z);
if (x != begin() && isect(--x, y))
    isect(x, y = erase(y));
while ((y = x) != begin() && (--x)->p >= y->p)
    isect(x, erase(y));
}

lli query(lli x) {
    if (empty()) return 0LL;
    auto f = *lower_bound(x);
    return f(x);
}
};

```

5.5 Divide and conquer $\mathcal{O}(kn^2) \Rightarrow \mathcal{O}(k \cdot n \log n)$

Split the array of size n into k continuous groups. $k \leq n$
 $cost(a, c) + cost(b, d) \leq cost(a, d) + cost(b, c)$ with $a \leq b \leq c \leq d$

```

void dc(int cut, int l, int r, int optl, int optpr) {
    if (r < l)
        return;
    int mid = (l + r) / 2;
    pair<lli, int> best = {inf, -1};
    fore (p, optl, min(mid, optpr) + 1) {
        lli nxtGroup = dp[~cut & 1][p - 1] + cost(p, mid);
        if (nxtGroup < best.f)
            best = {nxtGroup, p};
    }
    dp[cut & 1][mid] = best.f;
    int opt = best.s;
    dc(cut, l, mid - 1, optl, opt);
    dc(cut, mid + 1, r, opt, optpr);
}

fore (i, 1, n + 1)
    dp[1][i] = cost(1, i);
fore (cut, 2, k + 1)
    dc(cut, cut, n, cut, n);

```

5.6 Knuth optimization $\mathcal{O}(n^3) \Rightarrow \mathcal{O}(n^2)$

$dp[l][r] = \min_{l \leq k \leq r} \{dp[l][k] + dp[k][r]\} + cost(l, r)$

```

fore (len, 1, n + 1)
    fore (l, 0, n) {
        int r = l + len - 1;
        if (r > n - 1)
            break;
        if (len <= 2) {
            dp[l][r] = 0;
            opt[l][r] = l;
            continue;
        }
        dp[l][r] = inf;
        fore (k, opt[l][r - 1], opt[l + 1][r] + 1) {
            lli cur = dp[l][k] + dp[k][r] + cost(l, r);
            if (cur < dp[l][r]) {
                dp[l][r] = cur;
                opt[l][r] = k;
            }
        }
    }
}

```

5.7 Do all submasks of a mask

```
for (int B = A; B > 0; B = (B - 1) & A)
```

6 Game Theory

6.1 Grundy Numbers

If the moves are consecutive $S = \{1, 2, 3, \dots, x\}$ the game can be

solved like $stackSize \pmod{x+1} \neq 0$

```

int mem[N];

int mex(set<int> &st) {
    int x = 0;
    while (st.count(x))
        x++;
    return x;
}

int grundy(int n) {
    if (n < 0)
        return inf;
    if (n == 0)
        return 0;
    int &g = mem[n];
    if (g == -1) {
        set<int> st;
        for (int x : {a, b})
            st.insert(grundy(n - x));
        g = mex(st);
    }
    return g;
}

```

7 Combinatorics

Combinatorics table		
Number	Factorial	Catalan
0	1	1
1	1	1
2	2	2
3	6	5
4	24	14
5	120	42
6	720	132
7	5,040	429
8	40,320	1,430
9	362,880	4,862
10	3,628,800	16,796
11	39,916,800	58,786
12	479,001,600	208,012
13	6,227,020,800	742,900

7.1 Factorial

```

fac[0] = 1LL;
fore (i, 1, N)
    fac[i] = lli(i) * fac[i - 1] % mod;
ifac[N - 1] = fpow(fac[N - 1], mod - 2);
fore (i, N - 1, 0)
    ifac[i] = lli(i + 1) * ifac[i + 1] % mod;

```

7.2 Factorial mod *smallPrime*

```

lli facMod(lli n, int p) {
    lli r = 1LL;
    for (; n > 1; n /= p) {
        r = (r * ((n / p) % 2 ? p - 1 : 1)) % p;
        fore (i, 2, n % p + 1)
            r = r * i % p;
    }
    return r % p;
}

```

7.3 Lucas theorem

Changes $\binom{n}{k} \pmod{p}$, with $n \geq 2e6, k \geq 2e6$ and $p \leq 1e7$

$$\binom{n}{k} \equiv \prod_{i=0}^n \binom{n_i}{k_i} \pmod{p}$$

7.4 Stars and bars

Enclosing n objects in k boxes

$$\binom{n+k-1}{k-1} = \binom{n+k-1}{n}$$

7.5 N choose K

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

$$\binom{n}{k_1, k_2, \dots, k_m} = \frac{n!}{k_1! * k_2! * \dots * k_m!}$$

```
lli choose(int n, int k) {
    if (n < 0 || k < 0 || n < k)
        return 0LL;
    return fac[n] * ifac[k] % mod * ifac[n - k] % mod;
}
```

```
lli choose(int n, int k) {
    double r = 1;
    for (i, 1, k + 1)
        r = r * (n - k + i) / i;
    return lli(r + 0.01);
}
```

7.6 Catalan

```
catalan[0] = 1LL;
for (i, 0, N) {
    catalan[i + 1] = catalan[i] * lli(4 * i + 2) % mod *
        fpow(i + 2, mod - 2) % mod;
}
```

7.7 Burnside's lemma

$$|classes| = \frac{1}{|G|} \cdot \sum_{x \in G} f(x)$$

7.8 Prime factors of N!

```
vector< pair<lli, int> > factorialFactors(int n) {
    vector< pair<lli, int> > fac;
    for (lli p : primes) {
        if (n < p)
            break;
        lli mul = 1LL, k = 0;
        while (mul <= n / p) {
            mul *= p;
            k += n / mul;
        }
        fac.emplace_back(p, k);
    }
    return fac;
}
```

8 Number Theory

8.1 Goldbach conjecture

- All number ≥ 6 can be written as sum of 3 *primes*
- All even number > 2 can be written as sum of 2 *primes*

8.2 Sieve of Eratosthenes

Numbers up to $2e8$

```
int erat[N >> 6];
#define bit(i) ((i >> 1) & 31)
#define prime(i) !(erat[i >> 6] >> bit(i) & 1)

void bitSieve() {
    for (int i = 3; i * i < N; i += 2) if (prime(i))
        for (int j = i * i; j < N; j += (i << 1))
            erat[j >> 6] |= 1 << bit(j);
}
```

To factorize divide x by $factor[x]$ until is equal to 1

```
void factorizeSieve() {
    iota(factor, factor + N, 0);
    for (int i = 2; i * i < N; i++) if (factor[i] == i)
        for (int j = i * i; j < N; j += i)
            factor[j] = i;
}
```

Use it if you need a huge amount of $phi[x]$ up to some N

```
void phiSieve() {
    isp.set(); // bitset<N> is faster
    iota(phi, phi + N, 0);
    for (i, 2, N) if (isp[i])
        for (int j = i; j < N; j += i) {
            isp[j] = (i == j);
            phi[j] /= i;
            phi[j] *= i - 1;
        }
}
```

8.3 Phi of euler

```
lli phi(lli n) {
    if (n == 1)
        return 0;
    lli r = n;
    for (lli i = 2; i * i <= n; i++)
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            r -= r / i;
        }
    if (n > 1)
        r -= r / n;
    return r;
}
```

8.4 Miller-Rabin

```
bool compo(lli p, lli d, lli n, lli k) {
    lli x = fpow(p % n, d, n), i = k;
    while (x != 1 && x != n - 1 && p % n && i--)
        x = mul(x, x, n);
    return x != n - 1 && i != k;
}

bool miller(lli n) {
    if (n < 2 || n % 6 % 4 != 1)
        return (n | 1) == 3;
    int k = __builtin_ctzll(n - 1);
    lli d = n >> k;
    for (lli p : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}) {
        if (compo(p, d, n, k))
            return 0;
        if (compo(2 + rng() % (n - 3), d, n, k))
            return 0;
    }
    return 1;
}
```

8.5 Pollard-Rho

```
lli f(lli x, lli c, lli mod) {
    return (mul(x, x, mod) + c) % mod;
}

lli rho(lli n) {
    while (1) {
        lli x = 2 + rng() % (n - 3), c = 1 + rng() % 20, y
            = f(x, c, n), g;
        while ((g = __gcd(n + y - x, n)) == 1)
            x = f(x, c, n), y = f(f(y, c, n), c, n);
        if (g != n) return g;
    }
}
```

```

    }
    return -1;
}

void pollard(lli n, map<lli, int> &fac) {
    if (n == 1) return;
    if (n % 2 == 0) {
        fac[2]++;
        pollard(n / 2, fac);
        return;
    }
    if (miller(n)) {
        fac[n]++;
        return;
    }
    lli x = rho(n);
    pollard(x, fac);
    pollard(n / x, fac);
}

```

8.6 Amount of divisors

```

lli divs(lli n) {
    lli cnt = 1LL;
    for (lli p : primes) {
        if (p * p * p > n)
            break;
        if (n % p == 0) {
            lli k = 0;
            while (n > 1 && n % p == 0)
                n /= p, ++k;
            cnt *= (k + 1);
        }
    }
    lli sq = mysqrt(n); // A binary search, the last x *
                        // x <= n
    if (miller(n))
        cnt *= 2;
    else if (sq * sq == n && miller(sq))
        cnt *= 3;
    else if (n > 1)
        cnt *= 4;
    return cnt;
}

```

8.7 Bézout's identity

$$a_1 * x_1 + a_2 * x_2 + \dots + a_n * x_n = g$$

$$g = \gcd(a_1, a_2, \dots, a_n)$$

8.8 GCD

$$a \leq b; \gcd(a + k, b + k) = \gcd(b - a, a + k)$$

8.9 LCM

$$x = p * \text{lcm}(a_1, a_2, \dots, a_k) + q, 0 \leq q < \text{lcm}(a_1, a_2, \dots, a_k)$$

$$x \pmod{a_i} \equiv q \pmod{a_i} \text{ as } a_i \mid \text{lcm}(a_1, a_2, \dots, a_k)$$

8.10 Euclid

```

pair<lli, lli> euclid(lli a, lli b) {
    if (b == 0)
        return {1, 0};
    auto p = euclid(b, a % b);
    return {p.s, p.f - a / b * p.s};
}

```

8.11 Chinese remainder theorem

```

pair<lli, lli> crt(pair<lli, lli> a, pair<lli, lli> b)
{
    if (a.s < b.s)
        swap(a, b);
    auto p = euclid(a.s, b.s);
    lli g = a.s * p.f + b.s * p.s, l = a.s / g * b.s;
    if ((b.f - a.f) % g != 0)

```

```

        return {-1, -1}; // no solution
    p.f = a.f + (b.f - a.f) % b.s * p.f % b.s / g * a.s;
    return {p.f + (p.f < 0) * l, l};
}

```

9 Math

9.1 Progressions

Arithmetic progressions

$$a_n = a_1 + (n - 1) * diff$$

$$\sum_{i=1}^n a_i = n * \frac{a_1 + a_n}{2}$$

Geometric progressions

$$a_n = a_1 * r^{n-1}$$

$$\sum_{k=1}^n a_1 * r^k = a_1 * \left(\frac{r^{n+1} - 1}{r - 1} \right) : r \neq 1$$

9.2 Mod multiplication

```

lli mul(lli x, lli y, lli mod) {
    lli r = 0LL;
    for (x %= mod; y > 0; y >>= 1) {
        if (y & 1) r = (r + x) % mod;
        x = (x + x) % mod;
    }
    return r;
}

```

9.3 Fpow

```

lli fpow(lli x, lli y, lli mod) {
    lli r = 1;
    for (; y > 0; y >>= 1) {
        if (y & 1) r = mul(r, x, mod);
        x = mul(x, x, mod);
    }
    return r;
}

```

9.4 Fibonacci

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} fib_{n+1} & fib_n \\ fib_n & fib_{n-1} \end{bmatrix}$$

10 Geometry

10.1 Real

```

const ld eps = 1e-9;
#define eq(a, b) abs((a) - (b)) <= eps
#define neq(a, b) !eq(a, b)
#define geq(a, b) (a) - (b) >= eps
#define leq(a, b) (a) - (b) <= eps
#define ge(a, b) !leq(a, b)
#define le(a, b) !geq(a, b)

```

10.2 Point

```

int sgn(ld x) {
    return x > 0 ? 1 : (x < 0 ? -1 : 0);
}

```

```

template <class T>
struct Point {
    typedef Point<T> P;
    T x, y;
    explicit Point(T x = 0, T y = 0) : x(x), y(y) {}
    P operator + (const P &p) const {
        return P(x + p.x, y + p.y); }
    P operator - (const P &p) const {
        return P(x - p.x, y - p.y); }
    P operator * (T k) const {
        return P(x * k, y * k); }
    P operator / (T k) const {
        return P(x / k, y / k); }

    T dot(const P &p) { return x * p.x + y * p.y; }
    T cross(const P &p) { return x * p.y - y * p.x; }
    double length() const { return sqrtl(norm()); }
    T norm() const { return sq(x) + sq(y); }
    double angle() { return atan2(y, x); }

    P perp() const { return P(-y, x); }
    P unit() const { return (*this) / length(); }
    P rotate (double angle) const {
        return P(x * cos(angle) - y * sin(angle),
                x * sin(angle) + y * cos(angle)); }

    bool operator == (const P &p) const {
        return eq(x, p.x) && eq(y, p.y); }
    bool operator != (const P &p) const {
        return neq(x, p.x) || neq(y, p.y); }

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

using P = Point<double>;

double ccw(P a, P b, P c) {
    return (b - a).cross(c - a);
}

10.3 Angle Between Vectors
double angleBetween(P a, P b) {
    double x = a.dot(b) / a.length() / b.length();
    return acos(max(-1.0, min(1.0, x)));
}

10.4 Area Polygon
double area(vector<P> &pts) {
    double sum = 0;
    for (i, 0, n)
        sum += pts[i].cross(pts[(i + 1) % sz(pts)]);
    return abs(sum / 2);
}

10.5 Area Polygon In Circle
vector<P> intersectLineCircle(const P &a, const P &b,
    const P &c, ld r) {
    ld h2 = r * r - v.cross(c - a) * v.cross(c - a) / v.
        norm();
    P p = a + v * v.dot(c - a) / v.norm();
    if (eq(h2, 0))
        return {p}; // line tangent to circle
    else if (le(h2, 0))
        return {}; // no intersection
    else {
        point u = v.unit() * sqrt(h2);
        return {p - u, p + u}; // two points of
            intersection (chord)
    }
}

```

```

}
bool pointInLine(const P &a, const P &b, const P &p) {
    return eq((p - a).cross(b - a), 0);
}
bool pointInSegment(const P &a, const P &b, const P &p) {
    return pointInLine(a, b - a, p) && leq((a - p).dot(b - p), 0);
}
int pointInCircle(const P &c, ld r, const P &p) {
    ld l = (p - c).length() - r;
    return (le(l, 0) ? 1 : (eq(l, 0) ? -1 : 0));
}
vector<P> intersectSegmentCircle(const P &a, const P &b,
    const P &c, ld r) {
    vector<P> points = intersectLineCircle(a, b - a, c,
        r), ans;
    for (const P &p : points) {
        if (pointInSegment(a, b, p)) ans.pb(p);
    }
    return ans;
}
ld signed_angle(const P &a, const P &b) {
    return sgn(a.cross(b)) * acosl(a.dot(b) / (a.length()
        * b.length()));
}
ld intersectPolygonCircle(const vector<P> &points,
    const P &c, ld r) {
    int n = points.size();
    ld ans = 0;
    for (int i = 0; i < n; ++i) {
        P p = points[i], q = points[(i + 1) % n];
        bool p_inside = (pointInCircle(c, r, p) != 0);
        bool q_inside = (pointInCircle(c, r, q) != 0);
        if (p_inside && q_inside) {
            ans += (p - c).cross(q - c);
        } else if (p_inside && !q_inside) {
            P s1 = intersectSegmentCircle(p, q, c, r)[0];
            P s2 = intersectSegmentCircle(c, q, c, r)[0];
            ans += (p - c).cross(s1 - c) + r * r *
                signed_angle(s1 - c, s2 - c);
        } else if (!p_inside && q_inside) {
            P s1 = intersectSegmentCircle(c, p, c, r)[0];
            P s2 = intersectSegmentCircle(p, q, c, r)[0];
            ans += (s2 - c).cross(q - c) + r * r *
                signed_angle(s1 - c, s2 - c);
        } else {
            auto info = intersectSegmentCircle(p, q, c, r);
            if (info.size() <= 1) {
                ans += r * r * signed_angle(p - c, q - c);
            } else {
                P s2 = info[0], s3 = info[1];
                P s1 = intersectSegmentCircle(c, p, c, r)[0];
                P s4 = intersectSegmentCircle(c, q, c, r)[0];
                ans += (s2 - c).cross(s3 - c) + r * r * (
                    signed_angle(s1 - c, s2 - c) +
                    signed_angle(s3 - c, s4 - c));
            }
        }
    }
    return abs(ans) / 2;
}

```

10.6 Closest Pair Of Points

```

pair<P, P> cpp(vector<P> points) {
    sort(all(points), [&](P a, P b) {
        return le(a.y, b.y);
    });
    set<P> st;
    ld ans = inf;
}

```

```

P p, q;
int pos = 0, n = sz(points);
for (i, 0, n) {
    while (pos < i && geq(points[i].y - points[pos].y,
        ans))
        st.erase(points[pos++]);
    auto lo = st.lower_bound({points[i].x - ans - eps,
        -inf});
    auto hi = st.upper_bound({points[i].x + ans + eps,
        -inf});
    for (auto it = lo; it != hi; ++it) {
        ld d = (points[i] - *it).length();
        if (le(d, ans))
            ans = d, p = points[i], q = *it;
    }
    st.insert(points[i]);
}
return {p, q};
}

```

10.7 Convex Hull

```

vector<P> convexHull(vector<P> &pts) {
    int n = sz(pts);
    vector<P> low, up;
    sort(all(pts), [&](P a, P b) {
        return a.x == b.x ? a.y < b.y : a.x < b.x;
    });
    pts.erase(unique(all(pts)), pts.end());
    if (n <= 2)
        return pts;
    for (i, 0, n) {
        while (sz(low) >= 2 && (low.end()[-1] - low.end()[-2]).cross(pts[i] - low.end()[-1]) <= 0)
            low.pop_back();
        low.pb(pts[i]);
    }
    for (i, n, 0) {
        while (sz(up) >= 2 && (up.end()[-1] - up.end()[-2]).cross(pts[i] - up.end()[-1]) <= 0)
            up.pop_back();
        up.pb(pts[i]);
    }
    low.pop_back(), up.pop_back();
    low.insert(low.end(), all(up));
    return low;
}

```

10.8 Distance Point Line

```

double distancePointLine(P a, P v, P p){
    return (proj(p - a, v) - (p - a)).length();
}

```

10.9 Get Circle

```

pair<P, double> getCircle(P m, P n, P p){
    P c = intersectLines((n + m) / 2, (n - m).perp(), (p
        + m) / 2, (p - m).perp());
    double r = (c - m).length();
    return {c, r};
}

```

10.10 Intersects Line

```

int intersectLinesInfo(P a1, P v1, P a2, P v2) { // v
    1 = b - a, v2 = d - c
    if (v1.cross(v2) == 0)
        return (a2 - a1).cross(v1) == 0 ? -1 : 0; // -1:
        infinity Ps, 0: no Ps
    else
        return 1; // single P
}

```

```

P intersectLines(P a1, P v1, P a2, P v2) {

```

```

    return a1 + v1 * ((a2 - a1).cross(v2) / v1.cross(v2)
        );
}

```

10.11 Intersects Line Segment

```

int intersectLineSegmentInfo(P a, P v, P c, P d) {
    P v2 = d - c;
    ld det = v.cross(v2);
    if (det == 0) {
        if ((c - a).cross(v) == 0)
            return -1; // infinity points
        else
            return 0; // no points
    } else
        return sgn(v.cross(c - a)) != sgn(v.cross(d - a))
            ;
}

```

10.12 Intersects Segment

```

int intersectSegmentsInfo(const P &a, const P &b,
    const P &c, const P &d) {
    P v1 = b - a, v2 = d - c;
    int t = sgn(v1.cross(c - a)), u = sgn(v1.cross(d - a
        ));
    if (t == u) {
        if (t == 0) {
            if (PInSegment(a, b, c) || PInSegment(a, b, d)
                || PInSegment(c, d, a) || PInSegment(c, d,
                    b))
                return -1; // infinity Ps
            else
                return 0; // no P
        } else
            return 0; // no P
    } else
        return sgn(v2.cross(a - c)) != sgn(v2.cross(b -
            c)); // 1: single P 0: no P
}

```

10.13 Is Convex

```

bool isConvex(vector<P> pts) {
    int n = sz(pts);
    bool hasPos = false, hasNeg = false;
    for (i, 0, n) {
        P first = pts[(i + 1) % n] - pts[i];
        P second = pts[(i + 2) % n] - pts[(i + 1) % n];
        double sign = first.cross(second);
        if (sign > 0) hasPos = true;
        if (sign < 0) hasNeg = true;
    }
    return !(hasPos && hasNeg);
}

```

10.14 Perimeter

```

double perimeter(vector<P> &pts){
    int n = sz(pts);
    double sum = 0;
    for (i, 0, n)
        sum += (pts[(i + 1) % n] - pts[i]).length();
    return sum;
}

```

10.15 Point In Convex Polygon logN

```

// log(n)
// first preprocess: seg = process(points)
// for each query: PInConvexPolygon(seg, p - pts[0])
vector<P> process(const vector<P> &pts) {
    int n = sz(pts);
    rotate(pts.begin(), min_element(all(pts), [&](P a, P
        b) {
            return a.x == b.x ? a.y < b.y : a.x < b.x;

```

```

    }), pts.end());
    vector<P> seg(n - 1);
    for (i, 0, n - 1)
        seg[i] = pts[i + 1] - pts[0];
    return seg;
}

bool PInConvexPolygon(const vector<P> &seg, const P &p)
{
    int n = sz(seg);
    if (neq(seg[0].cross(p), 0) && sgn(seg[0].cross(p))
        != sgn(seg[0].cross(seg[n - 1])))
        return false;
    if (neq(seg[n - 1].cross(p), 0) && sgn(seg[n - 1].
        cross(p)) != sgn(seg[n - 1].cross(seg[0])))
        return false;
    if (eq(seg[0].cross(p), 0))
        return geq(seg[0].length(), p.length());
    int l = 0, r = n - 1;
    while (r - l > 1) {
        int m = l + ((r - l) >> 1);
        if (geq(seg[m].cross(p), 0))
            l = m;
        else
            r = m;
    }
    return eq(fabs(seg[l].cross(seg[l + 1])), fabs((p -
        seg[l]).cross(p - seg[l + 1])) +
        fabs(p.cross(seg[l])) + fabs(p.cross(seg[l +
            1]))));
}

```

10.16 Point In Polygon

```

int pointInPolygon(const vector<P> &pts, P p) { // O(N)
    int n = sz(pts), ans = 0;
    for (i, 0, n) {
        P a = pts[i], b = pts[(i + 1) % n];
        if (pointInSegment(a, b, p))
            return -1; // on perimeter
        if (a.y > b.y)
            swap(a, b);
        if (a.y <= p.y && b.y > p.y && (a - p).cross(b - p)
            > 0)
            ans ^= 1;
    }
    return ans ? 1 : 0; // inside, outside
}

```

10.17 Point In Segment

```

bool pointInSegment(P a, P b, P p){
    return (b - a).cross(p - a) == 0 && (a - p).dot(b -
        p) <= 0;
}

```

10.18 Points Of Tangency

```

pair<P, P> pointsOfTangency(P c, double r, P p){
    P v = (p - c).unit() * r;
    double cos_theta = r / (p - c).length();
    double theta = acos(max(-1.0, min(1.0, cos_theta)));
    return {c + v.rotate(-theta), c + v.rotate(theta)};
}

```

10.19 Projection

```

P proj(P a, P v){
    v = v / v.unit();
    return v * a.dot(v);
}

```

10.20 Projection Line

```

P projLine(P a, P v, P p){

```

```

    return a + proj(p - a, v);
}

```

10.21 Reflection Line

```

P reflectionLine(P a, P v, P p){
    return a * 2 - p + proj(p - a, v) * 2;
}

```

10.22 Signed Distance Point Line

```

double signedDistancePointLine(P a, P v, P p){
    return v.cross(p - a) / v.length();
}

```

10.23 Sort Along Line

```

void sortAlongLine(P a, P v, vector<P> &pts){
    sort(pts.begin(), pts.end(), [&](P u, P w){
        return u.dot(v) < w.dot(v);
    });
}

```

10.24 Intersects Line Circle

```

vector<P> intersectLineCircle(P a, P v, P c, double r)
{
    P p = proj_line(a, v, c);
    double d = (p - c).length();
    double h = sq(r) - sq(d);
    if (h == 0)
        return {p}; //line tangent to circle
    else if (h < 0)
        return {}; //no intersection
    else {
        P u = v.unit() * sqrt(h);
        return {p - u, p + u}; //two Ps of intersection (
            chord)
    }
}

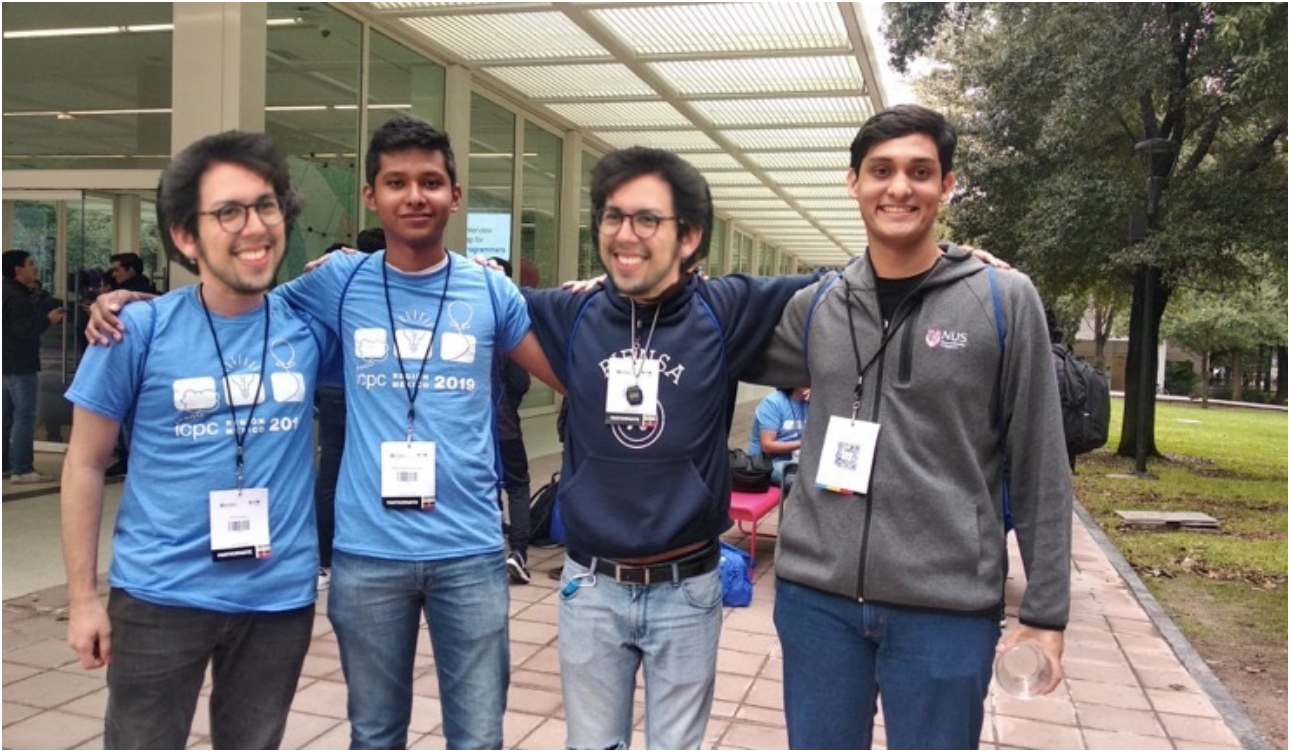
```

11 Bit tricks

Bits++	
Operations on <i>int</i>	Function
<code>x & -x</code>	Least significant bit in <i>x</i>
<code>__lg(x)</code>	Most significant bit in <i>x</i>
<code>c = x&-x, r = x+c; ((r^x) >> 2)/c r</code>	Next number after <i>x</i> with same number of bits set
<code>__builtin__</code>	Function
<code>popcount(x)</code>	Amount of 1's in <i>x</i>
<code>clz(x)</code>	0's to the left of biggest bit
<code>ctz(x)</code>	0's to the right of smallest bit

11.1 Bitset

Bitset<Size>	
Operation	Function
<code>_Find_first()</code>	Least significant bit
<code>_Find_next(idx)</code>	First set bit after index <i>idx</i>
<code>any(), none(), all()</code>	Just what the expression says
<code>set(), reset(), flip()</code>	Just what the expression says x2
<code>to_string('.', 'A')</code>	Print 011010 like .AA.A.



The end...