Team Note of `Powered by Zigui`

`@evenharder`(Sangheon Lee), `@SoulTch`(JEONGJIN LEE), `@djkim0613`(kim do jae)

Compiled on November 7, 2019

Contents

ALL BELOW HERE ARE USELESS IF YOU READ THE STATEMENT WRONG

0 Quotes and Prerequisites

```
evenharder : Mental Abuse To Humans
djkim0613 : 열심히 응원하겠습니다.
SoulTch : How much is this bus ticket?
* This template is brought from that of 'Deobureo Minkyu Party'
```

Run script

```
#!/bin/bash
g++ -fsanitize=undefined -std=c++14 -O2 -o /tmp/pow $1.cpp
time /tmp/pow < $1.in
# export PATH=~:$PATH
```

Debug Code

```
#define setz(x) memset(x, 0, sizeof(x))
#define sz(x) ((int)(x).size())
#define rep(i, e) for (int i = 0, _##i = (e); i < _##i; i++)
#define repp(i, s, e) for (int i = (s), _##i = (e); i < _##i; i++)
#define repr(i, s, e) for (int i = (s)-1, _##i = (e); i >= _##i; i--)
#define repi(i, x) for (auto &i : (x))
// using namespace std;
using ll = long long;
using pii = pair<int, int>;
using pll = pair<ll, ll>;
template<typename T>
ostream &operator<<(ostream &os, const vector<T>& v) {
    cout << "[";
    for (auto p : v) cout << p << ",";
    cout << "]";
    return os;
}
#ifndef __SOULTCH
#define debug(...) 0
#define endl '\n'
#else
#define debug(...) cout << " [-] ", _dbg(#__VA_ARGS__, __VA_ARGS__)
template<class TH> void _dbg(const char *sdbg, TH h){ cout << sdbg << '=' << h <<
endl; }
template<class TH, class... TA> void _dbg(const char *sdbg, TH h, TA... a) {
    while(*sdbg != ',') cout << *sdbg++;
    cout << '=' << (h) << ',';
    _dbg(sdbg+1, a...);
}
#endif
```

Reminders

Should be **added**.

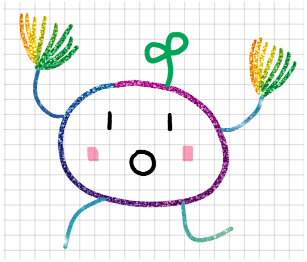

Figure 1: 풀다가 막힐 때는 이 그림을 봅시다. 아자아자 화이팅!

1 Math

1.1 Basic Mathematics

1.1.1 Trigonometry

- $\sin(\alpha \pm \beta) = \sin\alpha\cos\beta \pm \cos\alpha\sin\beta$
- $\cos(\alpha \pm \beta) = \cos\alpha\cos\beta \mp \sin\alpha\sin\beta$
- $\tan(\alpha \pm \beta) = \frac{\tan\alpha \pm \tan\beta}{1 \mp \tan\alpha\tan\beta}$
- $\sin 2\theta = 2\sin\theta\cos\theta$
- $c^2 = a^2 + b^2 - 2ab\cos\gamma$

1.1.2 Generating Function

- $\sum_n (pn+q)x^n = \frac{p}{1-x} + \frac{q}{(1-x)^2}$ (Arithmetic progression)
- $\sum_n (rx)^n = (1-rx)^{-1}$ (Geometric progression)
- $\sum_n \binom{m}{n} x^n = (1+x)^m$ (Binomial coefficient)
- $\sum_n \binom{m+n-1}{n} x^n = (1-x)^{-m}$ (Multiset coefficient)

1.1.3 Calculus

- $\int_a^b f(x)\,dx = \frac{b-a}{6}\left[f(a) + 4f(\frac{a+b}{2}) + f(b)\right]$ (Simpson's Rule, for cubic poly)
- $\int u'v\,dx = uv - \int uv'\,dx$ (Integration by parts)

1.2 Number Theory

1.2.1 Lattice Points under Line

```
// 0 <= x < n, 0 < y <= (a/c)x+(b/c)
ll calc(ll a,ll b,ll c,ll n){
    if(!n)return 0;
    ll tmp=a/c*n*(n-1)/2;
    tmp+=b/c*n;
    return tmp+calc(c,(a*n+b)%c,a%c,((a%c)*n+b%c)/c);
}
```

1.2.2 Shanks' Baby-step Giant-step

```
ll mexp(ll x, ll y, ll p) {
    if(!y) return 1;
    if(y & 1) return x * mexp(x*x%p, y>>1, p) % p;
    return mexp(x*x%p, y>>1, p);
}
vector<ll> get_factor(ll n) {
    vector<ll> v;
    for(ll i=2;i*i<=n;i++) {
        if(n % i == 0) {
            v.push_back(i);
            while(n % i == 0) n /= i;
        }
    }
    if(n > 1) v.push_back(n);
    return v;
}
ll get_primitive(ll n) {
    ll phi = n-1; // assume n is prime
    vector<ll> fact = get_factor(phi);
    for(ll x=2;x<=n;x++) {
        int yes = 1;
        for(ll y : fact) {
            yes &= (mexp(x, phi / y, n) != 1);
        }
        if(yes) return x;
    }
    return -1;
}
// find x s.t. x^k mod n = a -> (g^k)^y mod n = a, where x = g^y
ll bsgs(ll k, ll a, ll n) {
    ll g = get_primitive(n);
    ll phi = n-1; // assume n is prime
    if(g == -1) return -1;
    ll m = ceil(sqrt(n) + 1e-9);
    vector<pl> prec(m);
    for(ll j=0;j<m;j++) {
        prec[j] = {mexp(g, j * k % phi, n), j};
    }
    sort(prec.begin(), prec.end());
    ll cur = a, ncur = mexp(g, (phi - m) * k % phi, n);
    for(ll i=0;i<m;i++) {
        auto it = lower_bound(prec.begin(), prec.end(), pl(cur, 0));
        if(it->first == cur) {
            ll ans = mexp(g, (i*m + it->second) % phi, n);
            assert(mexp(ans, k, n) == a);
            return ans;
        }
        cur = cur * ncur % n;
    }
    return 0;
}
```

1.2.3 Extended Euclidean Algorithm

```
// ax + by = gcd(a,b). x, y?
pll ext_gcd(ll a,ll b) {
    if(b) {
        auto tmp = ext_gcd(b, a%b);
        return {tmp.second, tmp.first - (a/b) * tmp.second};
    }
    else return {1, 0};
}

// ax = gcd(a, m) mod m. x?
ll mod_inv(ll a, ll m) {
    return (ext_gcd(a, m).first + m) % m;
}
```

1.2.4 Chinese Remainder Theorem

```
ll pos_rem(ll a, ll m) { // m > 0. a % m?
    ll res = abs(a) % m;
    return a > 0 ? res : (res ? m - res : 0);
}
// ax = c mod m, bx = d mod n. x?
ll solve(ll a, ll c, ll m, ll b, ll d, ll n) {
    a = pos_rem(a, m); c = pos_rem(c, m);  // if a, c not in [0, m)
    b = pos_rem(b, n); d = pos_rem(d, n);  // if b, d not in [0, n)
    ll  g = _gcd(a, _gcd(c, m));    a /= g, c /= g, m /= g;
        g = _gcd(b, _gcd(d, n));    b /= g, d /= g, n /= g;
    if(c % _gcd(a, m) || d % _gcd(b, n)) return inf;
    ll t1 = (mod_inv(a, m) * c) % m;
    ll t2 = (mod_inv(b, n) * d) % n;
    g = _gcd(m, n);
    ll lc = m * n / g;
    if(abs(t1 - t2) % g) return inf;
    pl p = ext_gcd(m, n);
    ll q = (t1 * p.second * n/g + t2 * p.first * m/g);
    return pos_rem(q, lc);
```

```
}
```

1.2.5 Möbius Inversion Formula

$\forall n \in \mathbb{N}\ g(n) = \sum_{d|n} f(d) \implies f(n) = \sum_{d|n} \mu(d) g(n/d)$

1.3 FFT

FFT : $(a_0, a_1, \cdots, a_{n-1}) \mapsto (\sum_{j=0}^{n-1} a_0(\omega^0)^j, \sum_{j=0}^{n-1} a_1(\omega^1)^j, \cdots, \sum_{j=0}^{n-1} a_{n-1}(\omega^{n-1})^j)$

```
void fft(vector<base>& a, bool inv) {
    int n = a.size(), j = 0;
    vector<ll> roots(n/2);
    for(int i=1;i<n;i++) {
        int bit = (n >> 1);
        while(j >= bit) {
            j -= bit;
            bit >>= 1;
        }
        j += bit;
        if(i < j) swap(a[i], a[j]);
    }

    double ang = 2 * acos(-1) / n * (inv ? -1 : 1);
    for(int i=0;i<n/2;i++) {
        roots[i] = base(cos(ang * i), sin(ang * i));
    }

    /* In NTT, let prr = primitive root. Then,
    int ang = mexp(prr, (mod - 1) / n);
    if(inv) ang = mexp(ang, mod - 2);
    for(int i=0; i<n/2; i++){
        roots[i] = (i ? (1ll * roots[i-1] * ang % mod) : 1);
    }
    also, make sure to apply modulus under here
    */
    for(int i=2;i<=n;i<<=1) {
        int step = n / i;
        for(int j=0;j<n;j+=i) {
            for(int k=0;k<i/2;k++) {
                ll u = a[j+k], v = a[j+k+i/2] * roots[step * k];
                a[j+k] = u+v;
                a[j+k+i/2] = u-v;
            }
        }
    }
    if(inv) for(int i=0;i<n;i++) a[i] /= n;
}

void conv(vector<base>& x, vector<base>& y) {
    int n = 2; while(n < max(x.size(), y.size())) n <<= 1;
    n <<= 1;
    x.resize(n); y.resize(n);
    fft(x, false); fft(y, false);
    for(int i=0;i<n;i++) x[i] *= y[i];
    fft(x, true); // access (ll)round(x[i].real())
}
```

1.4 Miller-Rabin + Pollard-Rho

```
//Prove By Solving - https://www.acmicpc.net/problem/4149
namespace miller_rabin{
    lint mul(lint x, lint y, lint mod){ return (__int128) x * y % mod; }
    lint ipow(lint x, lint y, lint p){
        lint ret = 1, piv = x % p;
        while(y){
            if(y&1) ret = mul(ret, piv, p);
            piv = mul(piv, piv, p);
            y >>= 1;
        }
        return ret;
    }
    bool miller_rabin(lint x, lint a){
        if(x % a == 0) return 0;
        lint d = x - 1;
        while(1){
            lint tmp = ipow(a, d, x);
            if(d&1) return (tmp != 1 && tmp != x-1);
            else if(tmp == x-1) return 0;
            d >>= 1;
        }
    }
    bool isprime(lint x){
        for(auto &i : {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37}){
            if(x == i) return 1;
            if(x > 40 && miller_rabin(x, i)) return 0;
        }
        if(x <= 40) return 0;
        return 1;
    }
}

namespace pollard_rho{
    lint f(lint x, lint n, lint c){
        return (c + miller_rabin::mul(x, x, n)) % n;
    }
    void rec(lint n, vector<lint> &v){
        if(n == 1) return;
        if(n % 2 == 0){
            v.push_back(2);
            rec(n/2, v);
            return;
```

```
        }
        if(miller_rabin::isprime(n)){
            v.push_back(n);
            return;
        }
        lint a, b, c;
        while(1){
            a = rand() % (n-2) + 2;
            b = a;
            c = rand() % 20 + 1;
            do{
                a = f(a, n, c);
                b = f(f(b, n, c), n, c);
            }while(gcd(abs(a-b), n) == 1);
            if(a != b) break;
        }
        lint x = gcd(abs(a-b), n);
        rec(x, v);
        rec(n/x, v);
    }
    vector<lint> factorize(lint n){
        vector<lint> ret;
        rec(n, ret);
        sort(ret.begin(), ret.end());
        return ret;
    }
};
```

1.5 Black Box Linear Algebra + Kitamasa

```
vector<int> berlekamp_massey(vector<int> x){
    vector<int> ls, cur;
    int lf, ld;
    for(int i=0; i<x.size(); i++){
        lint t = 0;
        for(int j=0; j<cur.size(); j++){
            t = (t + 1ll * x[i-j-1] * cur[j]) % mod;
        }
        if((t - x[i]) % mod == 0) continue;
        if(cur.empty()){
            cur.resize(i+1);
            lf = i;
            ld = (t - x[i]) % mod;
            continue;
        }
        lint k = -(x[i] - t) * ipow(ld, mod - 2) % mod;
        vector<int> c(i-lf-1);
        c.push_back(k);
        for(auto &j : ls) c.push_back(-j * k % mod);
        if(c.size() < cur.size()) c.resize(cur.size());
        for(int j=0; j<cur.size(); j++){
            c[j] = (c[j] + cur[j]) % mod;
        }
        if(i-lf+(int)ls.size()>=(int)cur.size()){
            tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % mod);
        }
        cur = c;
    }
    for(auto &i : cur) i = (i % mod + mod) % mod;
    return cur;
}
int get_nth(vector<int> rec, vector<int> dp, lint n){
    int m = rec.size();
    vector<int> s(m), t(m);
    s[0] = 1;
    if(m != 1) t[1] = 1;
    else t[0] = rec[0];
    auto mul = [&rec](vector<int> v, vector<int> w){
        int m = v.size();
        vector<int> t(2 * m);
        for(int j=0; j<m; j++){
            for(int k=0; k<m; k++){
                t[j+k] += 1ll * v[j] * w[k] % mod;
                if(t[j+k] >= mod) t[j+k] -= mod;
            }
        }
        for(int j=2*m-1; j>=m; j--){
            for(int k=1; k<=m; k++){
                t[j-k] += 1ll * t[j] * rec[k-1] % mod;
                if(t[j-k] >= mod) t[j-k] -= mod;
            }
        }
        t.resize(m);
        return t;
    };
    while(n){
        if(n & 1) s = mul(s, t);
        t = mul(t, t);
        n >>= 1;
    }
    lint ret = 0;
    for(int i=0; i<m; i++) ret += 1ll * s[i] * dp[i] % mod;
    return ret % mod;
}
int guess_nth_term(vector<int> x, lint n){ // init with > 3k, O(l^2 lg n)
    if(n < x.size()) return x[n];
    vector<int> v = berlekamp_massey(x);
    if(v.empty()) return 0;
    return get_nth(v, x, n);
}
struct elem{int x, y, v;}; // A_(x, y) <- v, 0-based. no duplicate please..
```

```
vector<int> get_min_poly(int n, vector<elem> M){
    // smallest poly P such that A^i = sum_{j < i} {A^j \times P_j}
    vector<int> rnd1, rnd2;
    mt19937 rng(0x14004);
    auto randint = [&rng](int lb, int ub){
        return uniform_int_distribution<int>(lb, ub)(rng);
    };
    for(int i=0; i<n; i++){
        rnd1.push_back(randint(1, mod - 1));
        rnd2.push_back(randint(1, mod - 1));
    }
    vector<int> gobs;
    for(int i=0; i<2*n+2; i++){
        int tmp = 0;
        for(int j=0; j<n; j++){
            tmp += 1ll * rnd2[j] * rnd1[j] % mod;
            if(tmp >= mod) tmp -= mod;
        }
        gobs.push_back(tmp);
        vector<int> nxt(n);
        for(auto &i : M){ // sparse matrix * vector
            nxt[i.x] += 1ll * i.v * rnd1[i.y] % mod;
            if(nxt[i.x] >= mod) nxt[i.x] -= mod;
        }
        rnd1 = nxt;
    }
    auto sol = berlekamp_massey(gobs);
    reverse(sol.begin(), sol.end());
    return sol;
}
lint det(int n, vector<elem> M){
    vector<int> rnd;
    mt19937 rng(0x14004);
    auto randint = [&rng](int lb, int ub){
        return uniform_int_distribution<int>(lb, ub)(rng);
    };
    for(int i=0; i<n; i++) rnd.push_back(randint(1, mod - 1));
    for(auto &i : M){
        i.v = 1ll * i.v * rnd[i.y] % mod;
    }
    auto sol = get_min_poly(n, M)[0];
    if(n % 2 == 0) sol = mod - sol;
    for(auto &i : rnd) sol = 1ll * sol * ipow(i, mod - 2) % mod;
    return sol;
}
```

2 Geometry

2.1 struct Point

```
const double eps = 1e-10;
template <class T>
struct point{
    typedef point P;
    T x, y;
    point(T x=0, T y=0) : x(x), y(y) {}
    bool operator< (P a) const  {return fabs(x-a.x) > eps ? x<a.x : y<a.y;}
    bool operator== (P a) const {return max(fabs(x-a.x), fabs(y-a.y)) < eps;}
    P operator+ (P a) const {return P(x+a.x, y+a.y);}
    P operator- (P a) const {return P(x-a.x, y-a.y);}
    P operator- () const {return P(-x, -y);};
    T operator* (P a) const {return x*a.x + y*a.y;} // inner prod
    T operator/ (P a) const {return x*a.y - y*a.x;} // outer prod
    T dist2() const {return x*x + y*y;}
    double dist() const {return sqrt(double(dist2()));}
    P perp() const {return P(-y, x);}; // rotate 90 deg ccw
    P mult(T t) const {return P(x*t, y*t);}
    P unit() const {return P(x/dist(), y/dist());}
    P rotate(double a){
        return P(x*cos(a)-y*sin(a), x*sin(a)+y*cos(a));
    }
};

int sgn(double x) {return (x > eps) - (x < -eps);}
typedef point<double> P;
```

2.2 Distance, Intersection

2.2.1 Point-to-Line

```
double lineDist(P a, P b, P p) {
    return ((b-a)/(p-a))/(b-a).dist(); // a->b : left (+), right : (-);
}
```

2.2.2 Point-to-Segment

```
double segDist(P s, P e, P p) {
    if(s == e) return (p-s).dist(); // mind the eps
    double d = (e-s).dist2(), t = min(d, max(.0, (p-s)*(e-s)));
    return ((p-s).mult(d)-(e-s).mult(t)).dist() / d;
}
```

2.2.3 Line intersection

```
template<class P>
pair<int, P> lineInter(P a, P b, P c, P d){
    if((b-a)/(d-c) == 0) // parallel, mind the eps
        return {-((b-a)/(c-a) == 0), a};
```

```
    double  oa = (d-c)/(a-c),   ob = (d-c)/(b-c);
    return {(a.mult(ob) - b.mult(oa)).mult(1/(ob-oa))};
} // 1,0,-1(inf) : inter
```

2.2.4 Segment Intersection

```
bool onSegment(P s, P e, P p) {return segDist(s, e, p) < eps;}

template<class P> vector<P> segInter(P a, P b, P c, P d){
    double  oa = (d-c)/(a-c),   ob = (d-c)/(b-c),
            oc = (b-a)/(c-a),   od = (b-a)/(d-a);
    if(sgn(oa)*sgn(ob) < 0 && sgn(oc)*sgn(od) < 0)
        return {(a.mult(ob) - b.mult(oa)).mult(1/(ob-oa))};
    set<P> S;
    if(onSegment(c, d, a)) S.insert(a);
    if(onSegment(c, d, b)) S.insert(b);
    if(onSegment(a, b, c)) S.insert(c);
    if(onSegment(a, b, d)) S.insert(d);
    return vector<P>(S.begin(), S.end());
}
```

2.2.5 Circle-Line Intersection

Should be **added**.

2.3 Convex Hull

```
vector<pll> get_CV(vector<pll> V){
    sort(V.begin(), V.end());
    sort(V.begin() + 1, V.end(), [&](pll x, pll y){
        pll xx = x - V[0];
        pll yy = y - V[0];
        ll res = xx / yy;
        if(res != 0) return res > 0;
        if(xx.first != yy.first) return xx.first < yy.first;
        return xx.second < yy.second;
    });

    vector<pll> ret;
    for(auto val : V){
        while(ret.size() > 1){
            pll xx = ret[ret.size() - 2] - val;
            pll yy = ret[ret.size() - 1] - val;

            if(xx / yy <= 0) ret.pop_back();
            else break;
        }
        ret.push_back(val);
    }
    return ret;
}
```

2.4 Rotating Calipers

```
void rotating_calipers(vector<pll> CV){
    int pos = 0;
    for(int i = 0 ; i < CV.size() ; i++) if(CV[pos] < CV[i]) pos = i;

    int ind1 = 0, ind2 = pos;
    ll dist = (CV[ind1] - CV[ind2]) * (CV[ind1] - CV[ind2]);

    auto get_v = [&](int x) { return CV[(x + 1) % CV.size()] - CV[x];};
    for(int i = 0 ; i < CV.size() ; i++){
        pll v = get_v(i);
        while((-v) / get_v(pos) < 0) pos = (pos + 1) % CV.size();
        ll tmp_dist = (CV[pos] - CV[i]) * (CV[pos] - CV[i]);
        if(dist < tmp_dist) {
            dist = tmp_dist;
            ind1 = i;   ind2 = pos;
        }
    }

    printf("%lld %lld %lld %lld\n", CV[ind1].first, CV[ind1].second, CV[ind2].first,
    CV[ind2].second);
}
```

2.5 Sorting Points by Angle

```
// credit : http://koosaga.com/97
auto angle_sort = [&] (P &a, P &b){
    if((a < point(0, 0)) ^ (b < point(0, 0))) return b < a;
    if(a / b != 0) return a / b > 0;
    return a.dist2() < b.dist2(); // norm
}; // clockwise sort
```

2.6 Smallest Enclosing Circle

```
//Prove By Solving - https://www.acmicpc.net/problem/11930
int main(){
    scanf("%d", &N);
    for(int i = 1 ; i <= N ; i++) scanf("%lf%lf%lf", &A[i].x, &A[i].y, &A[i].z);

    int t = 70000;
    double rate = 1.0;
    point cur = (point){0, 0, 0};
    for(int i = 1 ; i <= t; i++){
        int ind = 1;
        for(int j = 1 ; j <= N ; j++) if((A[j] - cur) * (A[j] - cur) > (A[ind] -
        cur) * (A[ind] - cur)) ind = j;
        cur = cur + (A[ind] - cur) * rate;
```

```
        rate *= 0.99;
    }
    double r = 0;
    for(int i = 1 ; i <= N ; i++) r = max(r, (A[i] - cur) * (A[i] - cur));
    cout << sqrt(r);
    return 0;
} // Non-deterministic, deterministic O(n lg n) requires Voronoi diagram
```

2.7 Polygon Area

2.7.1 Polygon Area

```
double ans = 0; // ans : double area
for(int i=0;i<points.size();i++)
    ans += points[i] / points[(i+1 == points.size() ? 0 : i+1)];
```

3 Strings

3.1 Aho-Corasick Algorithm

```
namespace aho_corasick {
    const int MAXN = 100000, MAXC = 26;
    int trans[MAXN+1][MAXC];
    int fail[MAXN+1];
    bool term[MAXN+1];

    void build(const vector<string> &v) {
        setz(trans), setz(fail), setz(term);
        int cnode = 1;

        repi(s, v) {
            int p = 0;
            repi(j, s) {
                char c = j-'a';
                if (!trans[p][c]) trans[p][c] = cnode++;
                p = trans[p][c];
            }
            term[p] = true;
        }

        queue<int> q; rep(i, MAXC) if (trans[0][i]) q.push(trans[0][i]);
        while(!q.empty()) {
            int t = q.front(); q.pop();
            rep(i, MAXC) {
                if (trans[t][i]) {
                    int p = fail[t];
                    while(p and not trans[p][i]) p = fail[p];
                    p = trans[p][i];
                    fail[trans[t][i]] = p;
                    if (term[p]) term[trans[t][i]] = true;
                    q.push(trans[t][i]);
                }
            }
        }
    }

    bool query(string &t) {
        int p = 0;
        repi(i, t) {
            char c = i-'a';
            while(p and not trans[p][c]) p = fail[p];
            p = trans[p][c];
            if (term[p]) return true;

        }
        return false;
    }
}
```

3.2 Suffix Array

```
// str  : abracadabra
// SA   : 10 7 0 3 5 8 1 4 6 9 2
// LCP  : 1 4 1 1 0 3 0 0 0 2
vector<int> make_sa(const string& s) {
    int n = s.length();
    int lim = max(128, n+1);
    vector<int> sa(n), g(n+1), ng(n+1), cnt(lim), ind(lim+1);
    for(int i=0;i<n;i++) {
        sa[i] = i; g[i] = s[i];
    }
    g[n] = 0;
    for(int t=1;t<s.length();t<<=1)
    {
        auto cmp = [&] (int a, int b) {
            return g[a] != g[b] ? g[a] < g[b] : g[a+t] < g[b+t];
        };
        for(int i=0;i<n;i++)    cnt[g[min(i+t, n)]]++;
        for(int i=1;i<lim;i++)  cnt[i] += cnt[i-1];
        for(int i=n-1;i>=0;i--) ind[--cnt[g[min(i+t, n)]]] = i;
        for(int i=0;i<lim;i++)  cnt[i] = 0;
        for(int i=0;i<n;i++)    cnt[g[i]]++; // same as cnt[g[ind[i]]]++
        for(int i=1;i<lim;i++)  cnt[i] += cnt[i-1];
        for(int i=n-1;i>=0;i--) sa[--cnt[g[ind[i]]]] = ind[i];
        ng[sa[0]] = 1;
        for(int i=1;i<n;i++) {
            ng[sa[i]] = ng[sa[i-1]] + cmp(sa[i-1], sa[i]);
        }
        g = ng;

        fill(cnt.begin(), cnt.end(), 0);
        fill(ind.begin(), ind.end(), 0);
```

```
    }
    return sa;
}

vector<int> make_lcp(const string& s, const vector<int>& sa) {
    int n = s.length();
    vector<int> lcp(n-1), rank(n);
    for(int i=0;i<n;i++)
        rank[sa[i]] = i;
    int len = 0;
    for(int i=0;i<n;i++) {
        if(rank[i]) {
            int j = sa[rank[i]-1];
            int lc = n - max(i,j);
            while(len < lc && s[i+len] == s[j+len]) len++;
            lcp[rank[i]-1] = len;
        }
        if(len) len--;
    }
    return lcp;
}
```

3.3 Manacher's Algorithm

```
// 0-based
// s   = # h # e # l # l # o #
// ret = 0 1 0 1 0 1 2 1 0 1 0

vector<int> manacher(const string& s) {
    int n = s.size(), r = -1, k = -1;
    vector<int> p(n);
    for (int i=0; i<n; i++) {
        if (i<=r) p[i] = min(r-i, p[2*k-i]);
        while (i-p[i]-1>=0 and i+p[i]+1<n and s[i-p[i]-1] == s[i+p[i]+1]) p[i]++;
        if (r < i+p[i]) r = i+p[i], k = i;
    }
    return p;
}
```

3.4 Z Algorithm

```
// 0-based
// s   = a b c a b a b c a
// ret = 9 0 0 2 0 4 0 0 1

vector<int> z_algo(const string &s) {
    int l = 0, r = 0, N = sz(s);
    vector<int> Z(N);
    Z[0] = N;
    repp(i, 1, N) {
        if (i > r) {
            l = r = i;
            while(r < N and s[r] == s[r-l]) r++;
            r--;
            Z[i] = r-l+1;
        } else {
            int k = i-l;
            if (Z[k] < r-i+1) Z[i] = Z[k];
            else {
                l = i;
                while(r < N and s[r] == s[r-l]) r++;
                r--;
                Z[i] = r-l+1;
            }
        }
    }
    return Z;
}
```

3.5 Lexicographically Smallest String Rotation

```
// rotate(v.begin(), v.begin()+min_rotation(v), v.end());
int min_rotation(string s) {
    int a=0, N=sz(s); s += s;
    rep(b,0,N) rep(i,0,N) {
        if (a+i == b || s[a+i] < s[b+i]) {b += max(0, i-1); break;}
        if (s[a+i] > s[b+i]) { a = b; break; }
    }
    return a;
}
```

4 Graph Theory

4.1 Strongly Connected Component

```
const int MAXN = 2e5 + 10; // > 2*N
int N, M;
int dfsn[MAXN], low[MAXN], finished[MAXN], cnt;
vector<int> ADJ[MAXN];
vector<vector<int>> G;
stack<int> S;
int f(int x){ // 0 1 2 3 4 5... -> f(1) f(-1) f(2) f(-2) f(3) f(-3)...
    return 2 * (abs(x) - 1) + (x < 0);
}

void add_edge(int x, int y){ // call by f(x), f(y)
    ADJ[x ^ 1].push_back(y);
    ADJ[y ^ 1].push_back(x);
}

// memset(finished, -1, sizeof(finished));
int scc(int here){
```

```
    static vector<int> tmp;
    S.push(here);
    dfsn[here] = low[here] = ++cnt;
    int &ret = low[here];
    for(int there : ADJ[here]){
        if(dfsn[there] == 0) ret = min(ret, scc(there));
        else if(finished[there] == -1) ret = min(ret, dfsn[there]);
    }

    if(dfsn[here] == low[here]){
        while(1){
            int x = S.top();    S.pop();
            finished[x] = G.size();
            tmp.push_back(x);
            if(x == here) break;
        }
        G.push_back(tmp);
        tmp.clear();
    }
    return ret;
}
```

4.1.1 2-SAT

- scc를 실행시켜 `f(i)` 와 `f(-i)`가 같은 component에 있다면, 모순.
- `f(i)` 와 `f(-i)` 중 `finished` 배열의 수가 작은 것이 참이다.
  - SCC numbering의 역순이 위상정렬이기에, $F \to T$를 유지하기 위함

4.2 Biconnected Component

```
// https://gist.github.com/koosaga/6f6fd50dd7067901f1b1
void dfs(int x, int p){
    dfn[x] = low[x] = ++piv;
    par[x] = p;
    for(int i=0; i<graph[x].size(); i++){
        int w = graph[x][i];
        if(w == p) continue;
        if(!dfn[w]){
            dfs(w, x);
            low[x] = min(low[x], low[w]);
        }
        else{
            low[x] = min(low[x], dfn[w]);
        }
    }
}

void color(int x, int c){
    if(c > 0) bcc[x].push_back(c); // c == 0 : first component
    vis[x] = 1;
    for(int i=0; i<graph[x].size(); i++){
        int w = graph[x][i];
        if(vis[w]) continue;
        if(dfn[x] <= low[w]){
            bcc[x].push_back(++cpiv);
            color(w, cpiv);
        }
        else{
            color(w, c);
        }
    }
}
```

4.3 Euler Tour

```
struct Edge{
    int to, cnt; // to: 인접한 정점, cnt: 남은 사용 횟수
    Edge *dual; // dual: 역방향 간선을 가리키는 포인터
    Edge(): Edge(-1, 0){}
    Edge(int to1, int cnt1): to(to1), cnt(cnt1), dual(nullptr){}
};
void Eulerian(int curr){
    for(Edge *e: adj[curr]){
        if(e->cnt > 0){
            e->cnt--;
            e->dual->cnt--;
            Eulerian(e->to); // dfs
        }
    }
    cout << curr << '\n';
}
```

4.4 Heavy-Light Decomposition

```
int N, M;
vector<int> ADJ[MAXN];
int S[MAXN];
int hld_head[MAXN], color[MAXN], dfsn[MAXN], dcnt, hcnt;
int P[MAXN];

void dfs1(int here, int par){
  S[here] = 1;  P[here] = par;
  for(int there : ADJ[here]) if(there != par) dfs1(there, here), S[here] +=
  S[there];
}

void dfs2(int here, int c){ // dfs reordering
  if(hld_head[c] == 0) hld_head[c] = here;
  dfsn[here] = ++dcnt;  color[here] = c;
```

```
  sort(ADJ[here].begin(), ADJ[here].end(), [&](int x, int y){
    return S[x] > S[y];
  });

  int cnt = 0;
  for(int there : ADJ[here]) if(there != P[here]){
    if(++cnt == 1) dfs2(there, c);
    else dfs2(there, ++hcnt);
  }
}
```

4.5 Dominator Tree

```
namespace Dtree {
    const int MAXN = 250001;
    vector<int> E[MAXN], RE[MAXN], rdom[MAXN];

    int S[MAXN], RS[MAXN], cs;
    int par[MAXN], val[MAXN];
    int sdom[MAXN], rp[MAXN];
    int dom[MAXN];

    int Find(int x, int c = 0) {
        if (par[x] == x) return c?-1:x;
        int p = Find(par[x], 1);
        if (p == -1) return c?par[x]:val[x];
        if (sdom[val[x]] > sdom[val[par[x]]]) val[x] = val[par[x]];
        par[x] = p;
        return c?p:val[x];
    }
    void Union(int x, int y) {
        par[x] = y;
    }

    void dfs(int x) {
        RS[S[x] = ++cs] = x;
        par[cs] = sdom[cs] = val[cs] = cs;
        for(int e : E[x]) {
            if (S[e] == 0) dfs(e), rp[S[e]] = S[x];
            RE[S[e]].pb(S[x]);
        }
    }

    int Do(int s, int *up) {
        dfs(s);
        for (int i = cs;i;i--) {
            for (int e : RE[i]) sdom[i] = min(sdom[i], sdom[Find(e)]);
            if (i > 1) rdom[sdom[i]].pb(i);
            for (int e:rdom[i]) {
                int p = Find(e);
                if (sdom[p] == i) dom[e] = i;
                else dom[e] = p;
            }
            if (i > 1) Union(i, rp[i]);
        }
        for (int i = 2; i <= cs; i++) if (sdom[i] != dom[i]) dom[i] = dom[dom[i]];
        for (int i = 2; i <= cs; i++) {
            up[RS[i]] = RS[dom[i]];
        }
        return cs;
    }

    void addE(int x, int y) {E[x].pb(y);}
}
```

4.6 Negative Cycle Detection

Should be **added**.

4.7 Tree Compress

Should be **added**.

4.8 Global Min Cut

```
// Stoer-Wagner Algorithm, O(VE lg E)
int minimum_cut_phase(int n, int &s, int &t, vector<vector<int>> &adj, vector<int>
vis){
    vector<int> dist(n);
    int mincut = 1e9;
    while(true){
        int pos = -1, cur = -1e9;
        for(int i=0; i<n; i++){
            if(!vis[i] && dist[i] > cur){
                cur = dist[i];
                pos = i;
            }
        }
        if(pos == -1) break;
        s = t;
        t = pos;
        mincut = cur;
        vis[pos] = 1;
        for(int i=0; i<n; i++){
            if(!vis[i]) dist[i] += adj[pos][i];
        }
    }
    return mincut; // optimal s-t cut here is, {t} and V \ {t}
}
int solve(int n, vector<vector<int>> adj){
    if(n <= 1) return 0;
    vector<int> vis(n);
```

```
    int ans = 1e9;
    for(int i=0; i<n-1; i++){
        int s, t;
        ans = min(ans, minimum_cut_phase(n, s, t, adj, vis));
        vis[t] = 1;
        for(int j=0; j<n; j++){
            if(!vis[j]){
                adj[s][j] += adj[t][j];
                adj[j][s] += adj[j][t];
            }
        }
        adj[s][s] = 0;
    }
    return ans;
}
```

5 Network Flow

5.1 Theorems

Max-flow Min-cut theorem : 정점 s에서 정점 t까지 흐를 수 있는 최대 유량(max-flow)은 정점 s와 정점 t를 분리하는 간선들의 가중치 합(min-cut)과 같다.
Vertex cover : 어떤 그래프의 정점의 집합 S에 대해 그래프의 모든 간선이 S의 원소 중 최소 하나와 연결되어 있을 때, S를 해당 그래프의 vertex cover라고 하며, minimum vertex cover는 최소 개수의 정점을 사용한 vertex cover이다.
Independent set : 어떤 그래프의 정점의 집합 S에 대해 S의 서로 다른 두 정점을 연결하는 간선이 없을 때, S를 해당 그래프의 independent set이라고 하며, maximum independent set은 최대 개수의 정점을 사용한 independent set이다.
Matching (independent edge set) : 어떤 그래프의 간선의 집합 S에 대해 S의 서로 다른 두 간선이 공통된 정점을 가지지 않을 때, S를 해당 그래프의 matching이라고 하며, maximum matching은 최대 개수의 간선을 사용한 matching이다.
König's theorem : 이분 그래프의 maximum matching의 크기는 minimum vertex cover의 것과 같다.
Dinic's Algorithm : 시간 복잡도 $O(V^2E)$, unit capacity에서는 $\min(V^{2/3}E, E^{3/2})$.
Circulation Problem : 새로운 source/sink s_n, t_n를 만들어서 다음과 같이 간선을 추가하고 $maxflow(s_n \to t_n) = \sum l_i$인지 확인, 이후 $s \to t$로 maxflow

- $s_n \to b\ (l)$, $a \to t_n\ (l)$, $a \to b\ (r-l)$, $t \to s\ (\infty)$

5.2 Dinic's Algorithm

```
const int INF = 1e9;
struct Dinic{
    int N;
    struct edge{
        int index, cap, rev;
        edge() : index(0), cap(0), rev(0) {}
        edge(int index, int cap, int rev) : index(index), cap(cap), rev(rev) {}
    };

    vector<vector<edge>> ADJ;
    vector<int> R, W;

    Dinic() {}
    Dinic(int N) : N(N){
        ADJ.resize(N);  R.resize(N);    W.resize(N);
    }

    void CE(int node1, int node2, int cap){
        ADJ[node1].push_back(edge(node2, cap, ADJ[node2].size()));
        ADJ[node2].push_back(edge(node1, 0, ADJ[node1].size() - 1));
    }

    bool bfs(int src, int sink){
        fill(R.begin(), R.end(), -1);
        R[src] = 0;
        queue<int> Q; Q.push(src);
        while(Q.size()){
            int here = Q.front();   Q.pop();
            for(auto e : ADJ[here]){
                if(e.cap > 0 && R[e.index] == -1)
                    R[e.index] = R[here] + 1, Q.push(e.index);
            }
        }
        return R[sink] != -1;
    }

    int dfs(int here, int sink, int f){
        if(here == sink) return f;
        for(int &i = W[here] ; i < ADJ[here].size() ; i++){
            auto &e = ADJ[here][i];
            if(e.cap > 0 && R[here] < R[e.index]){
                int res = dfs(e.index, sink, min(f, e.cap));
                if(res) {
                    e.cap -= res;
                    ADJ[e.index][e.rev].cap += res;
                    return res;
                }
            }
        }
        return 0;
    }

    int solve(int src, int sink){
        int ret = 0;
        while(bfs(src, sink)){
            fill(W.begin(), W.end(), 0);
            int res;
            while((res = dfs(src, sink, INF))) ret += res;
        }
        return ret;
    }
```

```
};
```

5.3 MCMF with SPFA

```
const int INF = 1e9;
struct MCMF {
    struct edge {
        int there, cap, cost, rev;

        edge() : there(0), cap(0), cost(0), rev(0) {}
        edge(int there, int cap, int cost, int rev) : there(there), cap(cap),
        cost(cost), rev(rev) {}
    };

    int N;
    vector<vector<edge>> ADJ;
    vector<int> R, INQ, C, I;

    MCMF() : N(0) {}
    MCMF(int N) : N(N) { ADJ.resize(N + 1); R.resize(N + 1); INQ.resize(N + 1);
    C.resize(N + 1); I.resize(N + 1); }

    void CE(int i, int j, int cap, int cost) {
        ADJ[i].push_back(edge(j, cap, cost, ADJ[j].size()));
        ADJ[j].push_back(edge(i, 0, -cost, ADJ[i].size() - 1));
    }
    bool SPFA(int src, int sink) {
        queue<int> Q;    Q.push(src);
        fill(R.begin(), R.end(), -1);    R[src] = 0;
        fill(C.begin(), C.end(), -1);    C[src] = 0;
        fill(INQ.begin(), INQ.end(), 0);    INQ[src] = 1;
        while (Q.size()) {
            int here = Q.front();    Q.pop();
            INQ[here] = 0;
            for (int i = 0; i < ADJ[here].size(); i++) {
                auto e = ADJ[here][i];
                if (e.cap > 0 && (C[e.there] == -1 || C[e.there] > C[here] +
                e.cost)) {
                    C[e.there] = C[here] + e.cost;    R[e.there] = here;
                    I[e.there] = i;
                    if (!INQ[e.there]) INQ[e.there] = 1, Q.push(e.there);
                }
            }
        }
        if (C[sink] == -1) return false;
        return true;
    }

    pii mcmf(int src, int sink) {
        pii ret = { 0, 0 };
        while (SPFA(src, sink)) {
            int flow = INF, cost = 0;
            for (int here = sink; here != src; here = R[here]) flow = min(flow,
            ADJ[R[here]][I[here]].cap);
            for (int here = sink; here != src; here = R[here]) {
                auto &e = ADJ[R[here]][I[here]];
                cost += e.cost * flow;
                e.cap -= flow;
                ADJ[e.there][e.rev].cap += flow;
            }
            ret.first += flow, ret.second += cost;
        }
        return ret;
    }
};
```

5.4 Hungarian Method

```
namespace Hung {
    const int MX = 2000;
    // IMPORTANT : n <= m, 1-based
    using T = long double;

    T maxv = 1e200;
    T a[MX][MX], n, m;

    void init(int nn, int mm) { n = nn; m = mm; }
    void set_value(int x, int y, T val) { a[x][y] = val; }
    T solve(vector <int> &ans) {
        vector<T> v(m+1), u(n+1);
        vector<int> p (m+1), way (m+1);
        for (int i=1; i<=n; ++i) {
            p[0] = i;
            int j0 = 0;
            vector<T> minv (m+1, maxv);
            vector<char> used (m+1, false);
            do {
                used[j0] = true;
                T delta = maxv;
                int i0 = p[j0], j1;
                for (int j=1; j<=m; ++j) if (!used[j]) {
                    T cur = a[i0][j]-u[i0]-v[j];
                    if (cur < minv[j]) {
                        minv[j] = cur, way[j] = j0;
                    }
                    if (minv[j] < delta) {
                        delta = minv[j], j1 = j;
                    }
                }
                for (int j=0; j<=m; ++j) {
                    if (used[j]) {
                        u[p[j]] += delta, v[j] -= delta;
```

```
                    }
                    else {
                        minv[j] -= delta;
                    }
                }
                j0 = j1;
            } while (p[j0] != 0);
            do {
                int j1 = way[j0];
                p[j0] = p[j1];
                j0 = j1;
            } while (j0);
        }
        ans.resize(n + 1);
        for(int j=1;j<=m;++j) {
            ans[p[j]] = j;
        }
        return -v[0];
    }
}
```

5.5 Hopcroft-Karp Algorithm

```
struct hopcroft_karp{
    int N;
    vector<vector<int>> ADJ;
    vector<int> L, rev, used;

    hopcroft_karp() {}
    hopcroft_karp(int N) : N(N) {
        ADJ.resize(N);
        L.resize(N), rev.resize(N, -1), used.resize(N, 0);
    }

    void CE(int here, int there){
        ADJ[here].push_back(there);
    }

    void bfs(){
        queue<int> Q;
        for(int i = 0 ; i < N ; i++) {
            if(used[i]) L[i] = -1;
            else L[i] = 0, Q.push(i);
        }

        while(Q.size()){
            int here = Q.front();   Q.pop();
            for(int there : ADJ[here]){
                if(rev[there] != -1 && L[rev[there]] == -1) {
                    L[rev[there]] = L[here] + 1;
                    Q.push(rev[there]);
                }
            }
        }
    }

    bool dfs(int here){
        for(int there : ADJ[here]){
            if(rev[there] == -1 || (L[here] < L[rev[there]] && dfs(rev[there]))){
                rev[there] = here;
                used[here] = 1;
                return true;
            }
        }
        return false;
    }

    int solve(){
        int ret = 0;
        while(1){
            bfs();
            int res = 0;
            for(int i = 0 ; i < N ; i++) {
                if(used[i]) continue;
                res += dfs(i);
            }
            if(res == 0) break;
            ret += res;
        }
        return ret;
    }
};
```

6 Optimization Tricks

6.1 Knuth Optimization

- Recurrence : $D[i][j] = \min_{i<k<j}(D[i][k] + D[k][j]) + C[i][j]$
- Quadrangle Inequality : $C[a][c] + C[b][d] \le C[a][d] + C[b][c],\ a \le b \le c \le d$
- Monotonicity : $C[b][c] \le C[a][d],\ a \le b \le c \le d$
- $A[i][j] =$ (min. k $s.t.$ $D[i][j]$ is min.). Then $A[i][j-1] \le A[i][j] \le A[i+1][j]$
- $O(N^2)$ time complexity

```
// opt[i-1][i] = i
for(int d=2;d<=n;d++) {
    for(int i=1;i+d<=n+1;i++) {
        for(int k=opt[i][j-1], j=i+d; k<=opt[i+1][j]; k++) {
            int v = dp[i][k] + dp[k][j] + c[i][j];
            if(dp[i][j] > v) dp[i][j] = v, opt[i][j] = k;
```

```
        }
    }
}
```

6.2 Divide and Conquer Optimization

- Recurrence : $D[t][i] = \min_{k<i}(D[t-1][k] + C[k][i])$
- Min index : $A[t][i] \le A[t][i+1]$ ($A[t][i] =$ (min. k $s.t.$ $D[t][i]$ is min.))
- Quadrangle Inequality : $C[a][c] + C[b][d] \le C[a][d] + C[b][c],\ a \le b \le c \le d$
- Able to Divide and Conquer base on calculating $D[t][i]$
- $O(TN \lg N)$ time complexity

```
// range of index : [l,r]
// range of dp    : [s,e]
void dnc(int t, int l, int r, int s, int e)
{
    if(s > e) return;
    int m = (s+e)/2;
    D[t][m] = 2e9;
    for(int k=l;k<m&&k<=r;k++)
    {
        int tmp = D[t-1][k] + C[k][m];
        if(D[t][m] > tmp)
            D[t][m] = tmp, A[t][m] = k;
    }
    dnc(t, l, A[t][m], s, D[t][m]-1);
    dnc(t, A[t][m], r, D[t][m]+1, e);
}
```

6.3 Convex Hull Trick

- Recurrence : $dp[i] = \min_{j<i}(dp[j] + a[i]b[j]),\ b[i-1] \le b[i]$
- Think as $dp[x = a[i]] = \min_{j<i}(b[j] \cdot x + dp[j])$
- Thus push lines and find minimum (by binary search)
- If $a[i] \le a[i+1]$ sweeping is possible
- Intersection of $y = a_i x + b_i$ and $y = a_{i+1}x + b_{i+1} : x = \frac{b_{i+1} - b_i}{a_i - a_{i+1}}$

6.4 Centroid Decomposition

```
// credit : https://gist.github.com/igorcarpanese/75162f3253bd230abd0f32f9950bf384
int dfs(int u, int p) {
    sub[u] = 1;
    for (auto v : tree[u])
        if (v != p) sub[u] += dfs(v, u);
    return sub[u] + 1;
}

// each tree has at most two centroids
int centroid(int u, int p, int r) { // r : root
    for (auto v : tree[u])
        if (v != p and sub[v] > sub[r]/2) return centroid(v, u);
    return u;
}
```

7 Data Structure

7.1 Persistent Segment Tree

```
const MAXN = 1e5 + 10;
struct node{
    node *l, *r;
    int cnt;
    node () {}
} pool[(1 << 17) * 17], *tree_head[MAXN];

int tcnt;
node* alloc(){
    memset(pool + tcnt, 0, sizeof(node));
    return pool + tcnt++;
}

node * init(int l, int r){
    node *ret = alloc();
    if(l != r) {
        int mid = (l + r) / 2;
        ret->l = init(l, mid);
        ret->r = init(mid + 1, r);
    }
    return ret;
}

void update(node * here, node *par, int l, int r, int val){
    if(l == r) {
        here->cnt = par->cnt + 1;
        return;
    }

    int mid = (l + r) / 2;
    if(val <= mid){
        here->l = alloc();
        here->r = par->r;
        update(here->l, par->l, l, mid, val);
    }
    else {
```

```
        here->l = par->l;
        here->r = alloc();
        update(here->r, par->r, mid + 1, r, val);
    }
    here->cnt = here->l->cnt + here->r->cnt;
}

int query(node *node1, node *node2, int l, int r, int k){
    if(l == r) return l;
    int ccc = node1->l->cnt - node2->l->cnt;
    int mid = (l + r) / 2;
    if(k <= ccc) return query(node1->l, node2->l, l, mid, k);
    else return query(node1->r, node2->r, mid + 1, r, k - ccc);
}
```

7.2 Link-Cut Tree

```
struct node{
    node *pp, *p, *l, *r;
    int val;
    node(){ p = 0, l = 0, r = 0;}
    node(int val) : val(val) { p = 0, l = 0, r = 0;}
};

void push(node *x){}
void pull(node *x){}

void rotate(node *x){
    if(!x->p) return;
    push(x->p);  // if there's lazy stuff
    push(x);
    node *p = x->p;
    bool is_left = (p->l == x);
    node *b = (is_left ? x->r : x->l);
    x->p = p->p;
    if(x->p && x->p->l == p) x->p->l = x;
    if(x->p && x->p->r == p) x->p->r = x;
    if(is_left){
        if(b) b->p = p;
        p->l = b;
        p->p = x;
        x->r = p;
    }
    else{
        if(b) b->p = p;
        p->r = b;
        p->p = x;
        x->l = p;
    }
    pull(p); // if there's something to pull up
    pull(x);
    //if(!x->p) root = x; // IF YOU ARE SPLAY TREE
    if(p->pp){ // IF YOU ARE LINK CUT TREE
        x->pp = p->pp;
        p->pp = nullptr;
    }
}
void splay(node *x){
    while(x->p){
        node *p = x->p;
        node *g = p->p;
        if(g){
            if((p->l == x) ^ (g->l == p)) rotate(x);
            else rotate(p);
        }
        rotate(x);
    }
}
void access(node *x){
    splay(x);
    push(x);
    if(x->r){
        x->r->pp = x;
        x->r->p = nullptr;
        x->r = nullptr;
    }
    pull(x);
    while(x->pp){
        node *nxt = x->pp;
        splay(nxt);
        push(nxt);
        if(nxt->r){
            nxt->r->pp = nxt;
            nxt->r->p = nullptr;
            nxt->r = nullptr;
        }
        nxt->r = x;
        x->p = nxt;
        x->pp = nullptr;
        pull(nxt);
        splay(x);
    }
}
node *root(node *x){
    access(x);
    while(x->l){
        push(x);
        x = x->l;
    }
    access(x);
    return x;
```

```
}
node *par(node *x){
    access(x);
    if(!x->l) return nullptr;
    push(x);
    x = x->l;
    while(x->r){
        push(x);
        x = x->r;
    }
    access(x);
    return x;
}
node *lca(node *s, node *t){
    access(s);
    access(t);
    splay(s);
    if(s->pp == nullptr) return s;
    return s->pp;
}
void link(node *par, node *son){
    access(par);
    access(son);
    //son->rev ^= 1;  // remove if needed
    push(son);
    son->l = par;
    par->p = son;
    pull(son);
}
void cut(node *p){
    access(p);
    push(p);
    if(p->l){
        p->l->p = nullptr;
        p->l = nullptr;
    }
    pull(p);
}
```

7.3 Dynamic Convex Hull

```
// https://github.com/niklasb/contest-algos/blob/master/convex_hull/dynamic.cpp
const ll is_query = -(1LL<<62);
struct Line {
    ll m, b;
    mutable function<const Line*()> succ;
    bool operator<(const Line& rhs) const {
        if (rhs.b != is_query) return m < rhs.m;
        const Line* s = succ();
        if (!s) return 0;
        ll x = rhs.m;
        return b - s->b < (s->m - m) * x;
    }
};
struct HullDynamic : public multiset<Line> { // will maintain upper hull for maximum
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->m == z->m && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->m == x->m && y->b <= x->b;
        return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);
    }
    void insert_line(ll m, ll b) {
        auto y = insert({ m, b });
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    ll eval(ll x) {
        auto l = *lower_bound((Line) { x, is_query });
        return l.m * x + l.b;
    }
};
```

7.4 Stern-Brocot Tree

```
// __int128 is recommended
bool test(ll a, ll b) { // for testing directions, vary by prob
    // return true if (true value) >= a/b
    ll n = 0, m = 1;

    rep(i, N) {
        if (n < m*A[i].fi) n = A[i].fi, m = 1;

        ll c = b*n+m*a, d = m*b;
        ll g = gcd(c, d);
        n = c/g;
        m = d/g;

        if (n > m*A[i].se) return false;
    }
    return true;
}

pair<ll, ll> stern_brocot(ll M, ll N) {
    // M : max value
    // N : max divisor
    // if result is a/b, return as {a, b}
```

```
    ll a = 0, b = 1; // l
    ll c = 1, d = 0; // r
    int l, r;
    bool chg = true;

    while(chg) {
        chg = false;

        // to left
        l = 0, r = (N-d-1)/b+1;
        while(l < r) {
            int mid = (l+r+1)/2;
            if (test(a*mid+c, b*mid+d)) r = mid-1;
            else l = mid;
        }

        c += a*l;
        d += b*l;
        chg |= (l > 0);

        // to right
        l = 0, r = (d?(N-b-1)/d+1:M);
        while(l < r) {
            int mid = (l+r+1)/2;
            if (test(a+mid*c, b+mid*d)) l = mid;
            else r = mid-1;
        }

        a += c*l;
        b += d*l;
        chg |= (l > 0);
    }

    return {a, b};
}
```

7.5 Rope

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

int main()
{
    ios::sync_with_stdio(false);
    cin.tie(0);

    crope rp; // rope<char>
    string s("Lorem-ipsum");
    int n = s.length();
    rp.append(s.c_str()); // add element

    int x = 3, y = 8; // split and merge below
    rp = rp.substr(x, y-x) + rp.substr(0, x) + rp.substr(y, n);
    cout << rp.at(0) << '\n'; // get element, 'e'
    cout << rp << '\n'; // print, "em-ip|Lor|sum"
}
```

7.6 Bitset

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

int main() {
    bitset<8> b1(13);               // 00001101
    bitset<8> b2("10111");          // 00010111

    cout << b1.count() << endl;     // 3
    cout << b1.test(6) << endl;     // 0, since 2^6-th bit is 0
    b1.set(6);                      // set to 1, 1-fill if no param
    b2.reset(2);                    // set to 0, 0-fill if no param
    // use 'flip' for flipping

    cout << "b1:" << b1 << endl;            // b1:01001101
    cout << "b2:" << b2 << endl;            // b2:00010011
    // use any, none, all (c++11) for bit checking
    // supported operators : &, |, ^, <<, >>, ~, ==, !=
    // these operators must match size (given to template)

    cout << "&: " << (b1 & b2) << endl;     // &: 00000001
    cout << "^: " << (b1 ^ b2) << endl;     // &: 01011110
    cout << "|: " << (b1 | b2) << endl;     // |: 01011111
    cout << "~: " << (~b1) << endl;         // ~: 10110010
    cout << "<<:" << (b1 << 3) << endl;     // <<:01101000
    cout << b2.to_ulong() << endl;          // 19, c++11 supports ullong
}
```

7.7 Policy Based Data Structure

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

using namespace std;
using namespace __gnu_pbds;

typedef tree<
int,
null_type,
```

```
less<int>,
rb_tree_tag,
tree_order_statistics_node_update >
ordered_set;

// less<int> : not allow for duplicate
// less_equal<int> : allow for duplicate
// use upper_bound when you erase from set used less_equal

int N;

int main(void) {
    iostream::sync_with_stdio(false);
    cin.tie(nullptr);

    ordered_set X;

    X.insert(1);
    X.insert(2);
    X.insert(4);
    X.insert(8);
    X.insert(16);

    cout<<*X.find_by_order(1)<<endl; // 2
    cout<<*X.find_by_order(2)<<endl; // 4
    cout<<*X.find_by_order(4)<<endl; // 16
    cout<<(end(X)==X.find_by_order(6))<<endl; // true

    cout<<X.order_of_key(-5)<<endl;  // 0
    cout<<X.order_of_key(1)<<endl;   // 0
    cout<<X.order_of_key(3)<<endl;   // 2
    cout<<X.order_of_key(4)<<endl;   // 2
    cout<<X.order_of_key(400)<<endl; // 5
}
```

8 Miscellaneous

8.1 Misc Formulae and Algorithms

8.1.1 Faulhaber's Formula

$$T(n,k) = \sum_{i=1}^{n} i^k = \frac{(n+1)^{k+1} - 1^{k+1} - \sum_{j=0}^{k-1} \binom{k+1}{j} T(n,j)}{\binom{k+1}{k}}$$

Also use

$$(x+1)^d - x^d = 1 + \binom{d}{1}x + \binom{d}{2}x^2 + \cdots + \binom{d}{d-1}x^{d-1}$$

to get each coef.

8.1.2 Maximum Clique

```
typedef long long ll;
ll G[40]; // 0-index
int N, M;
int cur;
void get_clique(int R = 0, ll P = (1ll<<N)-1, ll X = 0){
    if((P|X) == 0){
        cur = max(cur, R);
        return;
    }
    int u = __builtin_ctzll(P|X);
    ll c = P&~G[u];
    while(c){
        int v = __builtin_ctzll(c);
        get_clique(R + 1, P&G[v], X&G[v]);
        P ^= 1ll << v;
        X |= 1ll << v;
        c ^= 1ll << v;
    }
}
```

8.1.3 De Brujin Sequence

```
// https://github.com/koosaga/DeobureoMinkyuParty/blob/master/teamnote.tex
// alphabet = [0, k - 1], substr length n, res starts with 0 (cyclic)
int res[10000000], aux[10000000]; // >= k^n, k*n
int de_bruijn(int k, int n, int lim) { // returns size (k^n)
    if(k == 1) {
        res[0] = 0;
        return 1;
    }
    for(int i = 0; i < k * n; i++) aux[i] = 0;
    int sz = 0;
    function<void(int, int)> db = [&](int t, int p) {
        if(sz > lim) return;
        if(t > n) {
            if(n % p == 0)
                for(int i = 1; i <= p; i++)
                    res[sz++] = aux[i];
        }
        else {
            aux[t] = aux[t - p];
            db(t + 1, p);
            for(int i = aux[t - p] + 1; i < k; i++) {
                aux[t] = i;
                db(t + 1, t);
            }
        }
    };
    db(1, 1);
```

```
    return sz;
}
```

8.2 Highly Composite Numbers, Large Prime

```
< 10^k          number     divisors   2 3 5 71113171923293137
-----------------------------------------------------------------
1                    6            4   1 1
2                   60           12   2 1 1
3                  840           32   3 1 1 1
4                 7560           64   3 3 1 1
5                83160          128   3 3 1 1 1
6               720720          240   4 2 1 1 1 1
7              8648640          448   6 3 1 1 1 1
8             73513440          768   5 3 1 1 1 1 1
9            735134400         1344   6 3 2 1 1 1 1
10          6983776800         2304   5 3 2 1 1 1 1 1
11         97772875200         4032   6 3 2 2 1 1 1 1
12        963761198400         6720   6 4 2 1 1 1 1 1 1
13       9316358251200        10752   6 3 2 1 1 1 1 1 1 1
14      97821761637600        17280   5 4 2 2 1 1 1 1 1 1
15     866421317361600        26880   6 4 2 1 1 1 1 1 1 1 1
16    8086598962041600        41472   8 3 2 2 1 1 1 1 1 1 1
17   74801040398884800        64512   6 3 2 2 1 1 1 1 1 1 1 1
18  897612484786617600       103680   8 4 2 2 1 1 1 1 1 1 1 1
```

```
       < 10^k prime        > 10^k prime         # of prime
------------------------------------------------------------
1                 7                  11                  4
2                97                 101                 25
3               997                1009                168
4              9973               10007               1229
5             99991              100003               9592
6            999983             1000003              78498
7           9999991            10000019             664579
8          99999989           100000007            5761455
9         999999937          1000000007           50847534
```

```
                < 10^k prime                 > 10^k prime
------------------------------------------------------------
10               9999999967                 10000000019
11              99999999977                100000000003
12             999999999989               1000000000039
13            9999999999971              10000000000037
14           99999999999973             100000000000031
15          999999999999989            1000000000000037
16         9999999999999937           10000000000000061
17        99999999999999997          100000000000000003
18       999999999999999989         1000000000000000003
```

NTT Prime:

$469762049 = 7 \times 2^{26} + 1$. Primitive root : 3.
$998244353 = 119 \times 2^{23} + 1$. Primitive root: 3.
$985661441 = 235 \times 2^{22} + 1$. Primitive root: 3.
$1012924417 = 483 \times 2^{21} + 1$. Primitive root: 5.
Primes near 10^9: $10^9 + [7, 9, 21, 33, 87]$

8.3 Fast Integer IO

```
// credit : https://github.com/koosaga/DeobureoMinkyuParty/blob/master/teamnote.tex
static char buf[1 << 19]; // size : any number geq than 1024
static int idx = 0;
static int bytes = 0;
static inline int _read() {
    if (!bytes || idx == bytes) {
        bytes = (int)fread(buf, sizeof(buf[0]), sizeof(buf), stdin);
        idx = 0;
    }
    return buf[idx++];
}
static inline int _readInt() {
    int x = 0, s = 1;
    int c = _read();
    while (c <= 32) c = _read();
    if (c == '-') s = -1, c = _read();
    while (c > 32) x = 10 * x + (c - '0'), c = _read();
    if (s < 0) x = -x;
    return x;
}
```

8.4 C++ Tips / Environments

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

struct StupidGCCCantEvenCompileThisSimpleCode{
    pair<int, int> array[1000000];
}; // https://gcc.gnu.org/bugzilla/show_bug.cgi?id=68203

// how to use rand (in 2017)
mt19937 rng(0xdeadbeef);
mt19937 rng(chrono::steady_clock::now().time_since_epoch().count());
int randint(int lb, int ub){ return uniform_int_distribution<int>(lb, ub)(rng); }
shuffle(permutation.begin(), permutation.end(), rng);
mt19937_64 _R(chrono::steady_clock::now().time_since_epoch().count()); // _R()

// comparator overload
auto cmp = [](seg a, seg b){return a.func() < b.func(); };
set<seg, decltype(cmp)> s(cmp);
map<seg, int, decltype(cmp)> mp(cmp);
priority_queue<seg, vector<seg>, decltype(cmp)> pq(cmp); // max heap
```

```
// hash func overload
struct point{
int x, y;
bool operator==(const point &p)const{ return x == p.x && y == p.y; }
};
struct hasher {
size_t operator()(const point &p)const{ return p.x * 2 + p.y * 3; }
};
unordered_map<point, int, hasher> hsh;

// c++ setprecision example
#include <iostream>     // std::cout, std::fixed
#include <iomanip>      // std::setprecision

int main () {
    double f =3.14159;
    std::cout << std::setprecision(5) << f << '\n'; // 3.1416
    std::cout << std::setprecision(9) << f << '\n'; // 3.14159
    std::cout << std::fixed;
    std::cout << std::setprecision(5) << f << '\n'; // 3.14159
    std::cout << std::setprecision(9) << f << '\n'; // 3.141590000
    return 0;
}
```