

Team Note of NLP

Jeounghui Nah, Dohyun Kim(Lobo_prix), Chansol Park

2023 ICPC World Finals (Luxor, Egypt)

Contents

1 DataStructure	1
1.1 Erasable Priority Queue	1
1.2 Convex Hull Trick (Stack, LineContainer)	1
1.3 Color Processor	2
1.4 Kinetic Segment Tree	2
1.5 Lazy LiChao Tree	2
1.6 Splay Tree, Link-Cut Tree	3
2 Geometry	3
2.1 $O(\log N)$ Point in Convex Polygon	3
2.2 Segment Distance, Segment Reflect	4
2.3 Tangent Series	4
2.4 Intersect Series	4
2.5 $O(N^2 \log N)$ Union of Circles Area	5
2.6 Segment In Polygon	5
2.7 Polygon Cut, Center, Union	5
2.8 Polygon Raycast	5
2.9 $O(N \log N)$ Shamos-Hoey	6
2.10 $O(N \log N)$ Half Plane Intersection	6
2.11 $O(M \log M)$ Dual Graph	6
2.12 $O(N^2 \log N)$ Bulldozer Trick	7
2.13 $O(N)$ Smallest Enclosing Circle	7
2.14 $O(N + Q \log N)$ K-D Tree	7
2.15 $O(N \log N)$ Voronoi Diagram	7
3 Graph	8
3.1 Euler Tour	8
3.2 2-SAT	8
3.3 Horn SAT	8
3.4 2-QBF	8
3.5 BCC	9
3.6 Prufer Sequence	9
3.7 $O(3^{V/3})$ Maximal Clique	9
3.8 $O(V \log V)$ Tree Isomorphism	9
3.9 $O(E \log E)$ Complement Spanning Forest	9
3.10 $O(E\sqrt{V})$ Bipartite Matching, Konig, Dilworth	10
3.11 $O(V^2\sqrt{E})$ Push Relabel	10
3.12 LR Flow, Circulation	10
3.13 Min Cost Circulation	10
3.14 $O(V^3)$ Hungarian Method	11
3.15 $O(V^3)$ Global Min Cut	11
3.16 $O(V^2 + V \times \text{Flow})$ Gomory-Hu Tree	11
3.17 $O(V + E\sqrt{V})$ Count/Find 3/4 Cycle	11
3.18 $O(V \log V)$ Rectilinear MST	12
3.19 $O(VE)$ Shortest Mean Cycle	12
3.20 $O(V^2)$ Stable Marriage Problem	12
3.21 $O((V + E) \log V)$ Dominator Tree	12
3.22 $O(VE)$ Vizing Theorem	12

3.23 $O(E + V^3 + V3^T + V^22^T)$ Minimum Steiner Tree	13
3.24 $O(E \log V)$ Directed MST	13
3.25 $O(E \log V + K \log K)$ K Shortest Walk	13
3.26 $O(V + E)$ Chordal Graph, Tree Decomposition	13
3.27 $O(V^3)$ General Matching	14
3.28 $O(V^3)$ Weighted General Matching	14
4 Math	15
4.1 Extend GCD, CRT, Combination	15
4.2 Partition Number	15
4.3 Diophantine	15
4.4 FloorSum	15
4.5 XOR Basis (XOR Maximization)	16
4.6 Stern Brocot Tree	16
4.7 $O(N^3 \log 1/\epsilon)$ Polynomial Equation	16
4.8 Gauss Jordan Elimination	16
4.9 Berlekamp + Kitamasa	16
4.10 Linear Sieve	17
4.11 Xudyh Sieve	17
4.12 Miller Rabin + Pollard Rho	17
4.13 Primitive Root, Discrete Log/Sqrt	17
4.14 Power Tower	17
4.15 De Bruijn Sequence	18
4.16 Simplex / LP Duality	18
4.17 Polynomial & Convolution	18
4.18 Matroid Intersection	19
5 String	20
5.1 KMP, Hash, Manacher, Z	20
5.2 Aho-Corasick	20
5.3 $O(N \log N)$ SA + LCP	20
5.4 $O(N \log N)$ Tandem Repeats	21
5.5 Suffix Automaton	21
5.6 Bitset LCS	21
5.7 Lyndon Factorization, Minimum Rotation	21
5.8 All LCS	21
6 Misc	21
6.1 CMakeLists.txt	21
6.2 Stack Hack	21
6.3 Python Decimal	21
6.4 Java I/O	22
6.5 Calendar	22
6.6 Ternary Search	22
6.7 Add/Mul Update, Range Sum Query	22
6.8 $O(N \times \max W)$ Subset Sum (Fast Knapsack)	22
6.9 Monotone Queue Optimization	22
6.10 Slope Trick	22
6.11 Aliens Trick	22
6.12 SWAMK, Min Plus Convolution	22
6.13 Money for Nothing (WF17D)	23
6.14 Exchange Argument on Tree (WF16L, CERC13E)	23
6.15 Hook Length Formula	23
6.16 Floating Point Add (Kahan)	23
6.17 Random, PBDS, Bit Trick, Bitset	23
6.18 Fast I/O, Fast Div, Fast Mod	23
6.19 DP Optimization	23
6.20 Highly Composite Numbers, Large Prime	23
6.21 DLAS(Diversified Late Acceptance Search)	24

7 Notes	24
7.1 Triangles/Trigonometry	24
7.2 Calculus, Newton's Method	24
7.3 Zeta/Mobius Transform	24
7.4 Counting	24
7.5 Generating Function	24
7.6 Faulhaber's Formula ($\sum_{k=1}^n k^c$)	25
7.7 About Graph Degree Sequence	25
7.8 Burnside, Grundy, Pick, Hall, Simpson, Area of Quadrangle, Fermat Point, Euler	25
7.9 About Graph Minimum Cut	25
7.10 Matrix with Graph(Kirchhoff, Tutte, LGV)	25
7.11 About Graph Matching(Graph with $ V \leq 500$)	25
7.12 Checklist	25

1 DataStructure

1.1 Erasable Priority Queue

```
template<class T=int, class O=less<T>>
struct pq_set {
    priority_queue<T, vector<T>, O> q, del;
    const T& top() const { return q.top(); }
    int size() const { return int(q.size()-del.size()); }
    bool empty() const { return !size(); }
    void insert(const T x) { q.push(x); flush(); }
    void pop() { q.pop(); flush(); }
    void erase(const T x) { del.push(x); flush(); }
    void flush() { while(del.size() && q.top()==del.top())
        q.pop(), del.pop(); }
};
```

1.2 Convex Hull Trick (Stack, LineContainer)

```
struct Line{ // call init() before use
    ll a, b, c; // y = ax + b, c = line index
    Line(ll a, ll b, ll c) : a(a), b(b), c(c) {}
    ll f(ll x){ return a * x + b; }
};
vector<Line> v; int pv;
void init(){ v.clear(); pv = 0; }
int chk(const Line &a, const Line &b, const Line &c) const {
    return (ll_int128_t)(a.b - b.b) * (b.a - c.a) <=
        (ll_int128_t)(c.b - b.b) * (b.a - a.a);
}
void insert(Line l){
    if(v.size() > pv && v.back().a == l.a){ // fix if min query
        if(l.b < v.back().b) l = v.back(); v.pop_back();
    }
    while(v.size() >= pv+2 && chk(v[v.size()-2], v.back(), l))
        v.pop_back();
    v.push_back(l);
}
p query(ll x){ // if min query, then v[pv].f(x) >= v[pv+1].f(x)
    while(pv+1 < v.size() && v[pv].f(x) <= v[pv+1].f(x)) pv++;
    return {v[pv].f(x), v[pv].c};
}
///// line container start (max query) /////
struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
}; // (for doubles, use inf = 1/.0, div(a,b) = a/b)
struct LineContainer : multiset<Line, less<>> {
    static const ll inf = LLONG_MAX; // div: floor
```

```

ll div(ll a, ll 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->k == y->k) x->p = x->m > y->m ? -inf : -inf;
    else x->p = div(y->m - x->m, x->k - y->k);
    return x->p >= y->p;
}
void add(ll k, ll m) {
    auto z = insert({k, m, 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));
}
ll query(ll x) { assert(!empty());
    auto l = *lower_bound(x); return l.k * x + l.m; }
};

```

1.3 Color Processor

```

template<class CT, class T> struct color_processor {
    map<array<CT, 2>, T> v; // CT: coord type
    color_processor(T col={}) : v({{MIN,MAX},col}){}
    auto get_range(CT p){ return *prev(v.upper_bound({p, MAX})); }
    // Cover [l, r) with color c, amortized O(1) process call
    // process(l, r, pc, c): color of [l, r) change pc -> c
    auto cover(CT l, CT r, T c, auto process){
        array<CT, 2> I{l, l};
        auto it = v.lower_bound(I);
        if(it != v.begin() && l < prev(it)->fi[1]){
            auto x = *--it; v.erase(it);
            v.insert({{x.fi[0],l}, x.se});
            it = v.insert({{l,x.fi[1]}, x.se}).fi;
        }
        while(it != v.end() && it->fi[0] < r){
            if(r < it->fi[1]){
                auto x = *it; v.erase(it);
                it = v.insert({{x.fi[0],r}, x.se}).fi;
                v.insert({{r,x.fi[1]}, x.se});
            }
            process(max(l,it->fi[0]), min(r,it->fi[1]), it->se, c);
            I = {min(I[0],it->fi[0]), max(I[1],it->fi[1])};
            it = v.erase(it);
        }
        return v.insert({I, c});
    }
    // new_color(l, r, pc): return new color for
    // [l, r) previous color pc O(NumberOf color ranges affected)
    void recolor(CT l, CT r, auto new_color){
        auto left = v.lower_bound({l, l});
        if(l < left->fi[0]){
            auto [range, c] = *--left; left = v.erase(left);
            left = v.insert(left, {{range[0],l},c});
            left = v.insert(left, {{l,range[1]},c});
        }
        auto right = v.lower_bound({r, r});
        if(r < right->fi[0]){
            auto [range, c] = *--right; right = v.erase(right);
            right = v.insert(right, {{range[0],r},c});
            right = v.insert(right, {{r,range[1]},c});
        }
        for(auto it=left; it!=right; ++it)
            it->se = new_color(it->fi[0], it->fi[1], it->se);
    }
};

```

1.4 Kinetic Segment Tree

```

// 일반적으로 heaten 함수는 교점 s개일 때 O(lambda_{s+2}(n)log^2n)
// update가 insert/delete만 주어진다면 O(lambda_{s+1}(n)log^2n)
// update가 없으면 O(lambda_s(n)log^2n)
// s = 0: 1 | s = 1: n | s = 2: 2n-1 | s = 3: 2n alpha(n) + O(n)
// s = 4: O(n * 2^alpha(n)) | s = 5: O(n alpha(n) * 2^alpha(n))
// apply_heat(heat): x좌표가 heat 증가했을 때의 증가량을 v에 더함
// heaten(l, r, t): 구간의 온도를 t 만큼 증가
struct line_t{
    ll a, b, v, idx; line_t() : line_t(0, nINF) {}
    line_t(ll a, ll b) : line_t(a, b, -1) {}
    line_t(ll a, ll b, ll idx) : a(a), b(b), v(b), idx(idx) {}
    void apply_heat(ll heat){ v += a * heat; }
    void apply_add(ll lz_add){ v += lz_add; }
    ll cross(const line_t &l) const {
        if(a == l.a) return pINF; ll p = v - l.v, q = l.a - a;
        if(q < 0) p = -p, q = -q;
        return p >= 0 ? (p + q - 1) / q : -p / q * -1;
    } ll cross_after(const line_t &l, ll temp) const {
        ll res = cross(l); return res > temp ? res : pINF; }
};
struct range_kinetic_segment_tree{
    struct node_t{
        line_t v; ll melt, heat, lz_add; node_t():node_t(line_t()){}
        node_t(ll a, ll b, ll idx) : node_t(line_t(a, b, idx)) {}
        node_t(const line_t &v):v(v),melt(pINF),heat(0),lz_add(0){}
        bool operator < (const node_t &o) const { return
            tie(v.v,v.a) < tie(o.v,v,o.v,a); }
        ll cross_after(const node_t &o, ll temp) const { return
            v.cross_after(o.v, temp); }
        void apply_lazy(){ v.apply_heat(heat); v.apply_add(lz_add);
            melt -= heat; }
        void clear_lazy(){ heat = lz_add = 0; }
        void prop_lazy(const node_t &p){ heat += p.heat; lz_add +=
            p.lz_add; }
        bool have_lazy() const { return heat != 0 || lz_add != 0; }
    };
    node_t T[SZ<<1]; range_kinetic_segment_tree(){ clear(); }
    void clear(){ fill(T, T+SZ*2, node_t()); }
    void pull(int node, int s, int e){
        if(s == e) return;
        const node_t &l = T[node<<1], &r = T[node<<1|1];
        assert(!l.have_lazy() && !r.have_lazy() &&
            !T[node].have_lazy());
        T[node] = max(l, r);
        T[node].melt = min({ l.melt, r.melt, l.cross_after(r, 0) });
    }
    void push(int node, int s, int e){
        if(!T[node].have_lazy()) return; T[node].apply_lazy();
        if(s != e) for(auto c : {node<<1, node<<1|1})
            T[c].prop_lazy(T[node]);
        T[node].clear_lazy();
    }
    void build(const vector<line_t> &lines, int node=1, int s=0,
        int e=SZ-1){
        if(s == e){ T[node] = s < lines.size() ? node_t(lines[s]) :
            node_t(); return; }
        int m = (s + e) / 2;
        build(lines,node*2,s,m); build(lines,node*2+1,m+1,e);
        pull(node, s, e);
    }
};

```

```

void update(int x, const line_t &v, int node=1, int s=0, int
    e=SZ-1){
    push(node, s, e); int m = (s + e) / 2;
    if(s == e){ T[node] = v; return; }
    if(x <= m)update(x,v, node<<1, s, m), push(node<<1|1, m+1,
        e);
    else update(x, v, node<<1|1, m+1, e), push(node<<1, s, m);
    pull(node, s, e);
}
void add(int l, int r, ll v, int node=1, int s=0, int e=SZ-1){
    push(node, s, e); int m = (s + e) / 2;
    if(r < s || e < l) return;
    if(l <= s && e <= r){ T[node].lz_add += v; push(node, s, e);
        return; }
    add(l,r,v,node*2,s,m); add(l,r,v,node*2+1,m+1,e);
    pull(node, s, e);
}
void heaten(int l,int r,ll t,int node=1,int s=0,int e=SZ-1){
    push(node, s, e); int m = (s + e) / 2;
    if(r < s || e < l) return;
    if(l <= s && e <= r){ _heat(t, node, s, e); return; }
    heaten(l,r,t,node*2,s,m); heaten(l,r,t,node*2+1,m+1,e);
    pull(node, s, e);
}
void _heat(ll t, int node=1, int s=0, int e=SZ-1){
    push(node, s, e); int m = (s + e) / 2;
    if(T[node].melt > t){ T[node].heat += t; push(node, s, e);
        return; }
    _heat(t,node*2,s,m);_heat(t,node*2+1,m+1,e);pull(node,s,e);
}
ll query(int l, int r, int node=1, int s=0, int e=SZ-1){
    push(node, s, e); if(r < s || e < l) return nINF;
    if(l <= s && e <= r) return T[node].v.v; int m = (s + e)/2;
    return max(query(l,r,node<<1,s,m), query(l,r,node<<1|1,m+1,e));
} // query end
};

```

1.5 Lazy LiChao Tree

```

/* get_point(x) : get min(f(x)), O(log X)
range_min(l,r) get min(f(x)), l<=x<=r, O(log X)
insert(l,r,a,b) : insert f(x)=ax+b, l<=x<=r, O(log^2 X)
add(l,r,a,b) : add f(x)=ax+b, l<=x<=r, O(log^2 X)
WARNING: a != 0인 add가 없을 때만 range_min 가능 */
template<typename T, T LE, T RI, T INF=(long long)(4e18)>
struct LiChao{
    struct Node{
        int l, r; T a, b, mn, aa, bb;
        Node(){ l = r = 0; a = 0; b = mn = INF; aa = bb = 0; }
        void apply(){ mn += bb; a += aa; b += bb; aa = bb = 0; }
        void add_lazy(T _aa, T _bb){ aa += _aa; bb += _bb; }
        T f(T x) const { return a * x + b; }
    }; vector<Node> seg; LiChao() : seg(2) {}
    void make_child(int n){
        if(!seg[n].l) seg[n].l = seg.size(), seg.emplace_back();
        if(!seg[n].r) seg[n].r = seg.size(), seg.emplace_back();
    }
    void push(int node, T s, T e){
        if(seg[node].aa || seg[node].bb){
            if(s != e){

```

```

    make_child(node);
    seg[seg[node].l].add_lazy(seg[node].aa, seg[node].bb);
    seg[seg[node].r].add_lazy(seg[node].aa, seg[node].bb);
} seg[node].apply();
}
}

void insert(T l, T r, T a, T b, int node=1, T s=LE, T e=RI){
    if(r < s || e < l || l > r) return;
    make_child(node); push(node, s, e); T m = (s + e) >> 1;
    seg[node].mn=min({seg[node].mn, a*max(s,l)+b,a*min(e,r)+b});
    if(s < l || r < e){
        if(l <= m) insert(l, r, a, b, seg[node].l, s, m);
        if(m+1 <= r) insert(l, r, a, b, seg[node].r, m+1, e);
        return;
    }
    T &sa = seg[node].a, &sb = seg[node].b;
    if(a*s+b < sa*s+sb) swap(a, sa), swap(b, sb);
    if(a*e+b >= sa*e+sb) return;
    if(a*m+b < sa*m+sb){
        swap(a,sa); swap(b,sb); insert(l,r,a,b,seg[node].l,s,m);
    } else insert(l, r, a, b, seg[node].r, m+1, e);
}

void add(T l, T r, T a, T b, int node=1, T s=LE, T e=RI){
    if(r < s || e < l || l > r) return;
    make_child(node); push(node, s, e); T m = (s + e) >> 1;
    if(s < l || r < e){
        insert(s, m, seg[node].a, seg[node].b, seg[node].l, s, m);
        insert(m+1,e,seg[node].a,seg[node].b,seg[node].r,m+1,e);
        seg[node].a = 0; seg[node].b = seg[node].mn = INF;
        if(l <= m) add(l, r, a, b, seg[node].l, s, m);
        if(m+1 <= r) add(l, r, a, b, seg[node].r, m+1, e);
        seg[node].mn=min(seg[seg[node].l].mn,
            seg[seg[node].r].mn);
        return;
    }
    seg[node].add_lazy(a, b); push(node, s, e);
}

T get_point(T x, int node=1, T s=LE, T e=RI){
    if(node == 0) return INF; push(node, s, e);
    T m = (s + e) >> 1, res = seg[node].f(x);
    if(x <= m) return min(res, get_point(x, seg[node].l, s, m));
    else return min(res, get_point(x, seg[node].r, m+1, e));
}

T range_min(T l, T r, int node=1, T s=LE, T e=RI){
    if(node == 0 || r < s || e < l || l > r) return INF;
    push(node, s, e); T m = (s + e) >> 1;
    if(l <= s && e <= r) return seg[node].mn;
    return min({ seg[node].f(max(s,l)), seg[node].f(min(e,r)),
        range_min(l, r, seg[node].l, s, m),
        range_min(l, r, seg[node].r, m+1, e) });
}
};

```

1.6 Splay Tree, Link-Cut Tree

```

struct Node{
    Node *l, *r, *p; bool flip; int sz; T now, sum, lz;
    Node(){ l = r = p = nullptr; sz = 1; flip = false; now = sum =
        lz = 0; }
    bool IsLeft() const { return p && this == p->l; }
    bool IsRoot() const { return !p || (this != p->l && this !=
        p->r); }
    friend int GetSize(const Node *x){ return x ? x->sz : 0; }
    friend T GetSum(const Node *x){ return x ? x->sum : 0; }
}

```

```

void Rotate(){
    p->Push(); Push();
    if(IsLeft()) r && (r->p = p), p->l = r, r = p;
    else l && (l->p = p), p->r = l, l = p;
    if(!p->IsRoot()) (p->IsLeft() ? p->p->l : p->p->r) = this;
    auto t = p; p = t->p; t->p = this; t->Update(); Update();
}

void Update(){
    sz = 1 + GetSize(l) + GetSize(r); sum = now + GetSum(l) +
        GetSum(r);
}

void Update(const T &val){ now = val; Update(); }

void Push(){
    Update(now + lz); if(flip) swap(l, r);
    for(auto c : {l, r}) if(c) c->flip ^= flip, c->lz += lz;
    lz = 0; flip = false;
}

Node* rt;
Node* Splay(Node *x, Node *g=nullptr){
    for(g || (rt=x); x->p!=g; x->Rotate()){
        if(!x->p->IsRoot()) x->p->p->Push();
        x->p->Push(); x->Push();
        if(x->p->p != g) (x->IsLeft() ^ x->p->IsLeft() ? x :
            x->p)->Rotate();
    }
    x->Push(); return x;
}

Node* Kth(int k){
    for(auto x=rt; ; x=x->r){
        for(; x->Push(), x->l && x->l->sz > k; x=x->l);
        if(x->l) k -= x->l->sz;
        if(!k-- return Splay(x);
    }
}

Node* Gather(int s, int e){ auto t = Kth(e+1); return Splay(t,
    Kth(s-1))->l; }

Node* Flip(int s, int e){ auto x = Gather(s, e); x->flip ^= 1;
    return x; }

Node* Shift(int s, int e, int k){
    if(k >= 0){ // shift to right
        k %= e-s+1; if(k) Flip(s, e), Flip(s, s+k-1), Flip(s+k, e);
    }
    else{ // shift to left
        k = -k; k %= e-s+1; if(k) Flip(s, e), Flip(s, e-k),
            Flip(e-k+1, e);
    }
    return Gather(s, e);
}

int Idx(Node *x){ return x->l->sz; }
////////// Link Cut Tree Start //////////
Node* Splay(Node *x){
    for(; !x->IsRoot(); x->Rotate()){
        if(!x->p->IsRoot()) x->p->p->Push();
        x->p->Push(); x->Push();
        if(!x->p->IsRoot()) (x->IsLeft() ^ x->p->IsLeft() ? x :
            x->p)->Rotate();
    }
    x->Push(); return x;
}

void Access(Node *x){
}

```

```

Splay(x); x->r = nullptr; x->Update();
for(auto y=x; x->p; Splay(x)) y = x->p, Splay(y), y->r = x,
    y->Update();
}

int GetDepth(Node *x){Access(x);x->Push();return GetSize(x->l);}
Node* GetRoot(Node *x){
    Access(x);for(x->Push();x->l;x->Push())x=x->l;return Splay(x);
}

Node* GetPar(Node *x){
    Access(x); x->Push(); if(!x->l) return nullptr;
    x = x->l; for(x->Push(); x->r; x->Push()) x = x->r;
    return Splay(x);
}

void Link(Node *p, Node *c){ Access(c); Access(p); c->l = p;
    p->p = c; c->Update(); }

void Cut(Node *c){ Access(c); c->l->p = nullptr; c->l = nullptr;
    c->Update(); }

Node* GetLCA(Node *x, Node *y){
    Access(x); Access(y); Splay(x); return x->p ? x->p : x;
}

Node* Ancestor(Node *x, int k){
    k = GetDepth(x) - k; assert(k >= 0);
    for(;;x->Push()){
        int s = GetSize(x->l); if(s == k) return Access(x), x;
        if(s < k) k -= s + 1, x = x->r; else x = x->l;
    }
}

void MakeRoot(Node *x){ Access(x); Splay(x); x->flip ^= 1;
    x->Push(); }

bool IsConnect(Node *x, Node *y){return GetRoot(x)==GetRoot(y);}
void PathUpdate(Node *x, Node *y, T val){
    Node *root = GetRoot(x); // original root
    MakeRoot(x); Access(y); // make x to root, tie with y
    Splay(x); x->lz += val; x->Push();
    MakeRoot(root); // Revert
    // edge update without edge vertex...
    Node *lca = GetLCA(x, y);
    Access(lca); Splay(lca); lca->Push();
    lca->Update(lca->now - val);
}

T VertexQuery(Node *x, Node *y){
    Node *l = GetLCA(x, y); T ret = l->now;
    Access(x); Splay(l); if(l->r) ret = ret + l->r->sum;
    Access(y); Splay(l); if(l->r) ret = ret + l->r->sum;
    return ret;
}

Node* GetQueryResultNode(Node *u, Node *v){
    if(!IsConnect(u, v)) return 0;
    MakeRoot(u); Access(v); auto ret = v->l;
    while(ret->mx != ret->now){
        if (ret->l && ret->mx == ret->l->mx) ret = ret->l;
        else ret = ret->r;
    }
    Access(ret); return ret;
}

```

2 Geometry

2.1 $O(\log N)$ Point in Convex Polygon

```

bool Check(const vector<Point> &v, const Point &pt){
    if(CCW(v[0], v[1], pt) < 0) return false;
}

```

```

int l = 1, r = v.size() - 1;
while(l < r){
    int m = l + r + 1 >> 1;
    if(CCW(v[0], v[m], pt) >= 0) l = m; else r = m - 1;
}
if(l == v.size() - 1) return CCW(v[0], v.back(), pt) == 0 &&
v[0] <= pt && pt <= v.back();
return CCW(v[0], v[l], pt) >= 0 && CCW(v[l], v[l+1], pt) >= 0
&& CCW(v[l+1], v[0], pt) >= 0;
}

```

2.2 Segment Distance, Segment Reflect

```

double Proj(Point a, Point b, Point c){
    ll t1 = (b - a) * (c - a), t2 = (a - b) * (c - b);
    if(t1 * t2 >= 0 && CCW(a, b, c) != 0)
        return abs(CCW(a, b, c)) / sqrt(Dist(a, b));
    else return 1e18; // INF
}

double SegmentDist(Point a[2], Point b[2]){
    double res = 1e18; // NOTE: need to check intersect
    for(int i=0; i<4; i++) res=min(res, sqrt(Dist(a[i/2], b[i/2])));
    for(int i=0; i<2; i++) res = min(res, Proj(a[0], a[i], b[i]));
    for(int i=0; i<2; i++) res = min(res, Proj(b[0], b[i], a[i]));
    return res;
}

P Reflect(P p1, P p2, P p3){ // line p1-p2, point p3
    auto [x1,y1] = p1; auto [x2,y2] = p2; auto [x3,y3] = p3;
    auto a = y1-y2, b = x2-x1, c = x1 * (y2-y1) + y1 * (x1-x2);
    auto d = a * y3 - b * x3;
    T x = -(a*c+b*d) / (a*a+b*b), y = (a*d-b*c) / (a*a+b*b);
    return 2 * P(x,y) - p3;
}

```

2.3 Tangent Series

```

template<bool UPPER=true> // 0(log N)
Point GetPoint(const vector<Point> &hull, real_t slope){
    auto chk = [slope](real_t dx, real_t dy){ return UPPER ? dy
    >= slope * dx : dy <= slope * dx; };
    int l = -1, r = hull.size() - 1;
    while(l + 1 < r){
        int m = (l + r) / 2;
        if(chk(hull[m+1].x - hull[m].x, hull[m+1].y -
        hull[m].y)) l = m; else r = m;
    }
    return hull[r];
}

int ConvexTangent(const vector<Point> &v, const Point &pt, int
up=1){ //given outer point, 0(log N)
    auto sign = [&](ll c){ return c>0 ? up : c==0 ? 0 : -up; };
    auto local = [&](Point p, Point a, Point b, Point c){
        return sign(CCW(p, a, b)) <= 0 && sign(CCW(p, b, c)) >= 0;
    }; // assert(v.size() >= 2);
    int n = v.size() - 1, s = 0, e = n, m;
    if(local(pt, v[1], v[0], v[n-1])) return 0;
    while(s + 1 < e){
        m = (s + e) / 2;
        if(local(pt, v[m-1], v[m], v[m+1])) return m;
        if(sign(CCW(pt, v[s], v[s+1])) < 0){ // up
            if(sign(CCW(pt, v[m], v[m+1])) > 0) e = m;
            else if(sign(CCW(pt, v[m], v[s])) > 0) s = m; else e = m;
        }
    }
}

```

```

    else{ // down
        if(sign(CCW(pt, v[m], v[m+1])) < 0) s = m;
        else if(sign(CCW(pt, v[m], v[s])) < 0) s = m; else e = m;
    }
}

if(s && local(pt, v[s-1], v[s], v[s+1])) return s;
if(e != n && local(pt, v[e-1], v[e], v[e+1])) return e;
return -1;
}

int Closest(const vector<Point> &v, const Point &out, int now){
    int prv = now > 0 ? now-1 : v.size()-1, nxt = now+1 < v.size()
    ? now+1 : 0, res = now;
    if(CCW(out, v[now], v[prv]) == 0 && Dist(out, v[res]) >
    Dist(out, v[prv])) res = prv;
    if(CCW(out, v[now], v[nxt]) == 0 && Dist(out, v[res]) >
    Dist(out, v[nxt])) res = nxt;
    return res; // if parallel, return closest point to out
} // int point_idx = Closest(convex_hull, pt,
ConvexTangent(hull + hull[0], pt, +-1) % N);
//////////

double polar(pdd x){ return atan2(x.second, x.first); }
int tangent(circle &A, circle &B, pdd des[4]){ // return angle
    int top = 0; // outer
    double d = size(A.O - B.O), a = polar(B.O - A.O), b = PI + a;
    double t = sq(d) - sq(A.r - B.r);
    if (t >= 0){
        t = sqrt(t); double p = atan2(B.r - A.r, t);
        des[top++] = pdd(a + p + PI / 2, b + p - PI / 2);
        des[top++] = pdd(a - p - PI / 2, b - p + PI / 2);
    }
    t = sq(d) - sq(A.r + B.r); // inner
    if (t >= 0){ t = sqrt(t);
        double p = atan2(B.r + A.r, t);
        des[top++] = pdd(a + p - PI / 2, b + p - PI / 2);
        des[top++] = pdd(a - p + PI / 2, b - p + PI / 2);
    }
    return top;
}

pair<T, T> CirclePointTangent(P o, double r, P p){
    T op=D1(p,o), u=atan2(p.y-o.y, p.x-o.x), v=acos1(r/op);
    return {u + v, u - v};
} // COORD 1e4 EPS 1e-7 / COORD 1e3 EPS 1e-9 with circleLine

```

2.4 Intersect Series

```

// 0: not intersect, -1: infinity, 4: intersect
// 1/2/3: intersect first/second/both segment corner
// flag, xp, xq, yp, yq : (xp / xq, yp / yq)
using T = __int128_t; // T <= 0(COORD^3)
tuple<int,T,T,T,T> SegmentIntersect(P s1, P e1, P s2, P e2){
    if(!Intersect(s1, e1, s2, e2)) return {0, 0, 0, 0, 0};
    auto det = (e1 - s1) / (e2 - s2);
    if(!det){
        if(s1 > e1) swap(s1, e1);
        if(s2 > e2) swap(s2, e2);
        if(e1 == s2) return {3, e1.x, 1, e1.y, 1};
        if(e2 == s1) return {3, e2.x, 1, e2.y, 1};
        return {-1, 0, 0, 0, 0};
    }
    T p = (s2 - s1) / (e2 - s2), q = det, flag = 0;
    T xp = s1.x * q + (e1.x - s1.x) * p, xq = q;
    T yp = s1.y * q + (e1.y - s1.y) * p, yq = q;
    if(xp%xq || yp%yq) return {4,xp,xq,yp,yq}; //gcd?
}

```

```

//if(xq < 0) xp=-xp, xq=-xq; if(yq < 0) yp=-yp, yq=-yq //gcd?
xp /= xq; yp /= yq;
if(s1.x == xp && s1.y == yp) flag |= 1;
if(e1.x == xp && e1.y == yp) flag |= 1;
if(s2.x == xp && s2.y == yp) flag |= 2;
if(e2.x == xp && e2.y == yp) flag |= 2;
return {flag ? flag : 4, xp, 1, yp, 1};
}

P perp() const { return P(-y, x); }
#define arg(p, q) atan2(p.cross(q), p.dot(q))
bool circleIntersect(P a,P b,double r1,double r2,pair<P, P>*
out){
    if (a == b) { assert(r1 != r2); return false; }
    P vec = b-a; double d2 = vec.dist2(), sum = r1+r2, dif =
    r1-r2;
    double p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
    if (sum*sum < d2 || dif*dif > d2) return false; // use EPS
    plz...
    P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
    *out = {mid + per, mid - per}; return true;
}

vector<P> circleLine(P c, double r, P a, P b) {
    P ab = b - a, p = a + ab * (c-a) * ab / D2(ab);
    T s = (b - a) / (c - a), h2 = r*r - s*s / D2(ab);
    if (abs(h2) < EPS) return {p}; if (h2 < 0) return {};
    P h = ab / D1(ab) * sqrt1(h2); return {p - h, p + h};
} // use circleLine if you use double...

int CircleLineIntersect(P o, T r, P p1, P p2, bool segment){
    P s = p1, d = p2 - p1; // line : s + kd, int support
    T a = d * d, b = (s - o) * d * 2, c = D2(s, o) - r * r;
    T det = b * b - 4 * a * c; // solve ak^2 + bk + c = 0, a > 0
    if(!segment) return Sign(det) + 1;
    if(det <= 0) return det ? 0 : 0 <= -b && -b <= a + a;
    bool f11 = b <= 0 || b * b <= det;
    bool f21 = b <= 0 && b * b >= det;
    bool f12 = a+a+b >= 0 && det <= (a+a+b) * (a+a+b);
    bool f22 = a+a+b >= 0 || det >= (a+a+b) * (a+a+b);
    return (f11 && f12) + (f21 && f22);
} // do not use this if you want to use double...

double circlePoly(P c, double r, vector<P> ps){ // return area
    auto tri = [&](P p, P q) { // ps must be ccw polygon
        auto r2 = r * r / 2; P d = q - p;
        auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
        auto det = a * a - b;
        if (det <= 0) return arg(p, q) * r2;
        auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
        if (t < 0 || 1 <= s) return arg(p, q) * r2;
        P u = p + d * s, v = p + d * t;
        return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
    };
    auto sum = 0.0;
    rep(i,0,sz(ps)) sum += tri(ps[i] - c, ps[(i+1)%sz(ps)] - c);
    return sum;
}

// extrVertex: point of hull, max projection onto line
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
int extrVertex(vector<P> &poly, P dir) {
    int n = sz(poly), lo = 0, hi = n;
    if (extr(0)) return 0;
}

```



```

while (lo + 1 < hi) {
    int m = (lo + hi) / 2; if (extr(m)) return m;
    int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
    (ls < ms || (ls == ms && ls == cmp(lo, m))) ? hi : lo = m;
}
return lo;
}
//(-1,-1): no collision
//(i,-1): touch corner
//(i,i): along side (i,i+1)
//(i,j): cross (i,i+1)and(j,j+1)
//(i,i+1): cross corner i
// 0(log n), ccw no colinear point convex polygon
// P perp() const { return P(-y, x); }
#define cmpL(i) sgn(a.cross(poly[i], b))
array<int, 2> lineHull(P a, P b, vector<P>& poly) { // 0(log N)
    int endA = extrVertex(poly, (a - b).perp());
    int endB = extrVertex(poly, (b - a).perp());
    if (cmpL(endA) < 0 || cmpL(endB) > 0) return {-1, -1};
    array<int, 2> res;
    rep(i, 0, 2) {
        int lo = endB, hi = endA, n = sz(poly);
        while ((lo + 1) % n != hi) {
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
            (cmpL(m) == cmpL(endB) ? lo : hi) = m;
        }
        res[i] = (lo + !cmpL(hi)) % n;
        swap(endA, endB);
    }
    if (res[0] == res[1]) return {res[0], -1};
    if (!cmpL(res[0]) && !cmpL(res[1]))
        switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
            case 0: return {res[0], res[0]};
            case 2: return {res[1], res[1]};
        }
    return res;
}
}

```

2.5 $O(N^2 \log N)$ Union of Circles Area

```

ld TwoCircleUnion(const Circle &p, const Circle &q) {
    ld d = D1(p.o - q.o); if(d >= p.r+q.r-EPS) return 0;
    else if(d <= abs(p.r-q.r)+EPS) return pow(min(p.r,q.r),2) *
    PI;
    ld pc = (p.r*p.r + d*d - q.r*q.r) / (p.r*d*2), pa = acosl(pc);
    ld qc = (q.r*q.r + d*d - p.r*p.r) / (q.r*d*2), qa = acosl(qc);
    ld ps = p.r*p.r*pa - p.r*p.r*sin(pa*2)/2;
    ld qs = q.r*q.r*qa - q.r*q.r*sin(qa*2)/2;
    return ps + qs;
}

vector<pair<double, double>> CoverSegment(Cir a, Cir b) {
    double d = abs(a.o - b.o); vector<pair<double, double>> res;
    if(sign(a.r + b.r - d) == 0); /* skip */
    else if(d <= abs(a.r - b.r) + eps) {
        if (a.r < b.r) res.emplace_back(0, 2 * pi);
    } else if(d < abs(a.r + b.r) - eps) {
        double o = acos((a.r*a.r + d*d - b.r*b.r) / (2 * a.r * d));
        double z = norm(atan2((b.o - a.o).y, (b.o - a.o).x));
        double l = norm(z - o), r = norm(z + o);
        if(l > r) res.emplace_back(l, 2*pi), res.emplace_back(0,r);
        else res.emplace_back(l, r);
    } return res;
}

```

```

} // circle should be identical
double CircleUnionArea(vector<Cir> c) {
    int n = c.size(); double a = 0, w;
    for (int i = 0; w = 0, i < n; ++i) {
        vector<pair<double, double>> s = {{2 * pi, 9}}, z;
        for (int j = 0; j < n; ++j) if (i != j) {
            z = CoverSegment(c[i], c[j]);
            for (auto &e : z) s.push_back(e); } /* for j */
        sort(s.begin(), s.end());
        auto F = [&] (double t) { return c[i].r * (c[i].r * t +
            c[i].o.x * sin(t) - c[i].o.y * cos(t)); };
        for (auto &e : s) {
            if (e.first > w) a += F(e.first) - F(w);
            w = max(w, e.second); } /* for e */
    } return a * 0.5; }

```

2.6 Segment In Polygon

```

bool segment_in_polygon_non_strict(const vector<Point> &poly,
    Point s, Point e){
    if(!PointInPolygon(poly, s, false) || !PointInPolygon(poly, e,
        false)) return false;
    if(s == e) return true; int cnt[4] = {0}; // no, at least one
    corner, mid, inf
    for(int j=(int)poly.size()-1, i=0; i<poly.size(); j=i++){
        int flag = get<0>(SegmentIntersect(poly[i], poly[j], s, e));
        if(flag<=0) flag = flag?3:0; else flag = max(1, flag-2);
        cnt[flag] += 1;
    }
    if(cnt[2] != 0 || cnt[3] > 1) return false;
    if((cnt[3] == 1 || cnt[1] != 0) && !PointInPolygon(poly, (s +
        e) / 2, false)) return false;
    return true;
}

```

2.7 Polygon Cut, Center, Union

```

// Returns the polygon on the left of line l
// *: dot product, ^: cross product
// l = p + d*t, l.q() = l + d
// doubled_signed_area(p,q,r) = (q-p) ^ (r-p)
template<class T> vector<point<T>> polygon_cut(const
    vector<point<T>> &a, const line<T> &l){
    vector<point<T>> res;
    for(auto i = 0; i < (int)a.size(); ++ i){
        auto cur = a[i], prev = i ? a[i - 1] : a.back();
        bool side = doubled_signed_area(l.p, l.q(), cur) > 0;
        if(side != (doubled_signed_area(l.p, l.q(), prev) > 0))
            res.push_back(l.p + (cur - l.p ^ prev - cur) / (l.d ^ prev
                - cur) * l.d);
        if(side) res.push_back(cur);
    }
    return res;
}

P polygonCenter(const vector<P>& v){ // center of mass
    P res(0, 0); double A = 0;
    for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
        res = res + (v[i] + v[j]) * v[j].cross(v[i]);
        A += v[j].cross(v[i]);
    } return res / A / 3;
}

// 0(points^2), area of union of n polygon, ccw polygon
int sideOf(P s, P e, P p) { return sgn((e-s)/(p-s)); }

```

```

int sideOf(const P& s, const P& e, const P& p, double eps) {
    auto a = (e-s)/(p-s); auto l=D1(e-s) * eps;
    return (a > l) - (a < -l);
}

double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }
double polyUnion(vector<vector<P>>& poly) { // (points)^2
    double ret = 0;
    rep(i, 0, sz(poly)) rep(v, 0, sz(poly[i])) {
        P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
        vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
        rep(j, 0, sz(poly)) if (i != j) { // START
            rep(u, 0, sz(poly[j])) {
                P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
                int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
                if (sc != sd) {
                    double sa = C.cross(D, A), sb = C.cross(D, B);
                    if (min(sc, sd) < 0)
                        segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
                }
                else if (!sc && !sd && j<i && sgn((B-A).dot(D-C))>0){
                    segs.emplace_back(rat(C - A, B - A), 1);
                    segs.emplace_back(rat(D - A, B - A), -1);
                } /*else if*/ } /*rep u*/ } /*rep j*/ // END
            sort(all(segs));
            for (auto& s : segs) s.first = min(max(s.first, 0.0), 1.0);
            double sum = 0; int cnt = segs[0].second;
            rep(j, 1, sz(segs)) {
                if (!cnt) sum += segs[j].first - segs[j - 1].first;
                cnt += segs[j].second;
            }
            ret += A.cross(B) * sum;
        } return abs(ret) / 2;
    }
}

```

2.8 Polygon Raycast

```

// ray A + kd and CCW polygon C, return events {k, event_id}
// 0: out->line / 1: in->line / 2: line->out / 3: line->in
// 4: pass corner outside / 5: pass corner inside / 6: out -> in
// 7: in -> out
// WARNING: C.push_back(C[0]) before working
struct frac{
    ll first, second; frac(){}
    frac(ll a, ll b) : first(a), second(b) {
        if (b < 0) first = -a, second = -b; // operator cast int128
    } double v(){ return 1.*first/second; } // operator <,<=,==
};

frac raypoints(vector<pii> &C, pii A, pii d, vector<pair<frac,
    int>> &R){
    assert(d != pii(0, 0));
    int g = gcd(abs(d.first), abs(d.second));
    d.first /= g, d.second /= g;
    vector<pair<frac, int>> L;
    for(int i = 0; i+1 < C.size(); i++){
        pii v = C[i+1] - C[i];
        int a = sign(d/(C[i]-A)), b = sign(d/(C[i+1]-A));
        if(a == 0)L.emplace_back(frac(d*(C[i]-A)/size2(d), 1), b);
        if(b == 0)L.emplace_back(frac(d*(C[i+1]-A)/size2(d), 1), a);
        if(a*b == -1) L.emplace_back(frac((A-C[i])/v, v/d), 6);
    }
    sort(L.begin(), L.end());
    int sz = 0;

```

```

for(int i = 0; i < L.size(); i++){
    // assert(i+2 >= L.size() || !(L[i].first == L[i+2].first));
    if(i+1<L.size() && L[i].first==L[i+1].first && L[i].second!=6){
        int a = L[i].second, b = L[i+1].second;
        R.emplace_back(L[i+1].first, a*b? a*b > 0? 4:6:(1-a-b)/2);
    }
    else R.push_back(L[i]);
}
int state = 0; // 0: out, 1: in, 2: line+ccw, 3: line+cw
for(auto &e : R){
    int &n = e.second;
    if( n == 6 ) n ^= state, state ^= 1;
    else if( n == 4 ) n ^= state;
    else if( n == 0 ) n = state, state ^= 2;
    else if( n == 1 ) n = state^(state>>1), state ^= 3;
} return frac(g, 1);
}
bool visible(vector<pii> &C, pii A, pii B){
    if( A == B ) return true;
    char I[4] = "356", O[4] = "157";
    vector<pair<frac, int>> R; vector<frac> E;
    frac s = frac(0, 1), e = raypoints(C, A, B-A, R);
    for(auto e : R){
        int &n = e.second, m;
        if(*find(0, 0+3, n+'0')) E.emplace_back(e.first);
        if(*find(I, I+3, n+'0')) E.emplace_back(e.first);
    }
    for(int j = 0; j < E.size(); j += 2) if( !(e <= E[j] || E[j+1]
<= s) ) return false;
    return true;
}

```

2.9 $O(N \log N)$ Shamos-Hoey

```

struct Line{
    static ll CUR_X; ll x1, y1, x2, y2, id;
    Line(Point p1, Point p2, int id) : id(id) {
        if(p1 > p2) swap(p1, p2);
        tie(x1,y1) = p1; tie(x2,y2) = p2;
    } Line() = default;
    int get_k() const { return y1 != y2 ? (x2-x1)/(y1-y2) : -1; }
    void convert_k(int k){ // x1,y1,x2,y2 = 0(COORD^2), use i128
        in ccw
        Line res;
        res.x1 = x1 + y1 * k; res.y1 = -x1 * k + y1;
        res.x2 = x2 + y2 * k; res.y2 = -x2 * k + y2;
        x1 = res.x1; y1 = res.y1; x2 = res.x2; y2 = res.y2;
        if(x1 > x2) swap(x1, x2), swap(y1, y2);
    }
    ld get_y(ll offset=0) const { // OVERFLOW
        ld t = ld(CUR_X-x1+offset) / (x2-x1);
        return t * (y2 - y1) + y1; }
    bool operator < (const Line &l) const {
        return get_y() < l.get_y(); }
    // strict
    /* bool operator < (const Line &l) const {
        auto le = get_y(), ri = l.get_y();
        if(abs(le-ri) > 1e-7) return le < ri;
        if(CUR_X==x1 || CUR_X==l.x1) return get_y(1)<l.get_y(1);
        else return get_y(-1) < l.get_y(-1);
    } */
}; ll Line::CUR_X = 0;
struct Event{ // f=0 st, f=1 ed

```

```

ll x, y, i, f; Event() = default;
Event(Line l, ll i, ll f) : i(i), f(f) {
    if(f==0) tie(x,y) = tie(l.x1,l.y1);
    else tie(x,y) = tie(l.x2,l.y2);
}
bool operator < (const Event &e) const {
    return tie(x,f,y) < tie(e.x,e.f,e.y);
    // strict
    // return make_tuple(x,-f,y) < make_tuple(e.x,-e.f,e.y);
}
};
tuple<bool,int,int> ShamosHoey(vector<array<Point,2>> v){
    int n = v.size(); vector<int> use(n+1);
    vector<Line> lines; vector<Event> E; multiset<Line> T;
    for(int i=0; i<n; i++){
        lines.emplace_back(v[i][0], v[i][1], i);
        if(int t=lines[i].get_k(); 0<=t && t<=n) use[t] = 1;
    }
    int k = find(use.begin(), use.end(), 0) - use.begin();
    for(int i=0; i<n; i++){ lines[i].convert_k(k);
        E.emplace_back(lines[i], i, 0); E.emplace_back(lines[i], i,
1);
    } sort(E.begin(), E.end());
    for(auto &e : E){ Line::CUR_X = e.x;
        if(e.f == 0){
            auto it = T.insert(lines[e.i]);
            if(next(it) != T.end() && Intersect(lines[e.i],
*next(it))) return {true, e.i, next(it)->id};
            if(it != T.begin() && Intersect(lines[e.i], *prev(it)))
                return {true, e.i, prev(it)->id};
        }
        else{
            auto it = T.lower_bound(lines[e.i]);
            if(it != T.begin() && next(it) != T.end() &&
Intersect(*prev(it), *next(it)))
                return {true, prev(it)->id, next(it)->id};
            T.erase(it);
        }
    }
    return {false, -1, -1};
}

```

2.10 $O(N \log N)$ Half Plane Intersection

```

double CCW(p1, p2, p3); bool same(double a, double b); const
Point o = Point(0, 0);
struct Line{ // ax+by leq c
    double a, b, c; Line() : Line(0, 0, 0) {}
    Line(double a, double b, double c) : a(a), b(b), c(c) {}
    bool operator < (const Line &l) const {
        bool f1 = Point(a, b) > o, f2 = Point(l.a, l.b) > o;
        if(f1 != f2) return f1 > f2;
        double cw = CCW(o, Point(a, b), Point(l.a, l.b));
        return same(cw, 0) ? c * hypot(l.a, l.b) < l.c * hypot(a, b)
: cw > 0;
    } Point slope() const { return Point(a, b); }
};
Point LineIntersect(Line a, Line b){
    double det = a.a*b.b - b.a*a.b, x = (a.c*b.b - a.b*b.c) / det,
y = (a.a*b.c - a.c*b.a) / det; return Point(x, y);
}
bool CheckHPI(Line a, Line b, Line c){
    if(CCW(o, a.slope(), b.slope()) <= 0) return 0;

```

```

Point v=LineIntersect(a,b); return v.x*c.a+v.y*c.b>c.c;
}
vector<Point> HPI(vector<Line> v){
    sort(v.begin(), v.end()); deque<Line> dq; vector<Point> ret;
    for(auto &i : v){
        if(dq.size() && same(CCW(o, dq.back().slope(), i.slope()),
0)) continue;
        while(dq.size() >= 2 && CheckHPI(dq[dq.size()-2], dq.back(),
i)) dq.pop_back();
        while(dq.size()>=2&&CheckHPI(i,dq[0],dq[1]))dq.pop_front();
        dq.push_back(i);
    }
    while(dq.size() > 2 && CheckHPI(dq[dq.size()-2], dq.back(),
dq[0])) dq.pop_back();
    while(dq.size() > 2 && CheckHPI(dq.back(), dq[0], dq[1]))
dq.pop_front();
    for(int i=0; i<dq.size(); i++){
        Line now = dq[i], nxt = dq[(i+1)%dq.size()];
        if(CCW(o, now.slope(), nxt.slope()) <= eps) return
vector<Point>();
        ret.push_back(LineIntersect(now, nxt));
    } //for(auto &[x,y] : ret) x = -x, y = -y;
    return ret;
} // MakeLine: left side of ray (x1,y1) -> (x2,y2)
Line MakeLine(T x1, T y1, T x2, T y2){
    T a = y2-y1, b = x1-x2, c = x1*a + y1*b; return {a,b,c}; }

```

2.11 $O(M \log M)$ Dual Graph

```

constexpr int quadrant_id(const Point p){
    constexpr int arr[9] = { 5, 4, 3, 6, -1, 2, 7, 0, 1 };
    return arr[sign(p.x)*3+sign(p.y)+4];
}
pair<vector<int>, int> dual_graph(const vector<Point> &points,
const vector<pair<int,int>> &edges){
    int n = points.size(), m = edges.size();
    vector<int> uf(2*m); iota(uf.begin(), uf.end(), 0);
    function<int(int)> find = [&](int v){ return v == uf[v] ? v :
uf[v] = find(uf[v]); };
    function<bool(int,int)> merge = [&](int u, int v){ return
find(u) != find(v) && (uf[uf[u]]=uf[v], true); };
    vector<vector<pair<int,int>>> g(n);
    for(int i=0; i<m; i++){
        g[edges[i].first].emplace_back(edges[i].second, i);
        g[edges[i].second].emplace_back(edges[i].first, i);
    }
    for(int i=0; i<n; i++){
        const auto base = points[i];
        sort(g[i].begin(), g[i].end(), [&](auto a, auto b){
            auto p1=points[a.first]-base, p2=points[b.first]-base;
            return quadrant_id(p1) != quadrant_id(p2) ?
quadrant_id(p1) < quadrant_id(p2) : p1.cross(p2) > 0;
        });
        for(int j=0; j<g[i].size(); j++){
            int k = j ? j - 1 : g[i].size() - 1;
            int u = g[i][k].second << 1, v = g[i][j].second << 1 | 1;
            auto p1=points[g[i][k].first], p2=points[g[i][j].first];
            if(p1 < base) u ^= 1; if(p2 < base) v ^= 1;
            merge(u, v);
        }
    }
}

```

```

}
vector<int> res(2*m);
for(int i=0; i<2*m; i++) res[i] = find(i);
auto comp=res;compress(comp);for(auto &i:res)i=IDX(comp,i);
int mx_idx = max_element(all(points)) - points.begin();
return {res, res[g[mx_idx].back().second << 1 | 1]};
}

```

2.12 $O(N^2 \log N)$ Bulldozer Trick

```

struct Line{
    ll i, j, dx, dy; // dx >= 0
    Line(int i, int j, const Point &pi, const Point &pj)
        : i(i), j(j), dx(pj.x-pi.x), dy(pj.y-pi.y) {}
    bool operator < (const Line &l) const {
        return make_tuple(dy*l.dx, i, j) < make_tuple(l.dy*dx, l.i, l.j);
    }
    bool operator == (const Line &l) const {
        return dy * l.dx == l.dy * dx;
    }
};

void Solve(){ // V.reserve(N*(N-1)/2)
    sort(A+1, A+N+1); iota(P+1, P+N+1, 1); vector<Line> V;
    for(int i=1; i<=N; i++) for(int j=i+1; j<=N; j++)
        V.emplace_back(i, j, A[i], A[j]);
    sort(V.begin(), V.end());
    for(int i=0, j=0; i<V.size(); i=j++){
        while(j < V.size() && V[i] == V[j]) j++;
        for(int k=i; k<j; k++){
            int u = V[k].i, v = V[k].j; // point id, index -> Pos[id]
            swap(Pos[u], Pos[v]); swap(A[Pos[u]], A[Pos[v]]);
            if(Pos[u] > Pos[v]) swap(u, v);
            // @TODO
        }
    }
}

```

2.13 $O(N)$ Smallest Enclosing Circle

```

pt getCenter(pt a, pt b){ return pt((a.x+b.x)/2, (a.y+b.y)/2); }
pt getCenter(pt a, pt b, pt c){
    pt aa = b - a, bb = c - a;
    auto c1 = aa*aa * 0.5, c2 = bb*bb * 0.5, d = aa / bb;
    auto x = a.x + (c1 * bb.y - c2 * aa.y) / d;
    auto y = a.y + (c2 * aa.x - c1 * bb.x) / d;
    return pt(x, y); }

Circle solve(vector<pt> v){
    pt p = {0, 0};
    double r = 0; int n = v.size();
    for(int i=0; i<n; i++) if(dst(p, v[i]) > r + EPS){
        p = v[i]; r = 0;
        for(int j=0; j<i; j++) if(dst(p, v[j]) > r + EPS){
            p = getCenter(v[i], v[j]); r = dst(p, v[i]);
            for(int k=0; k<j; k++) if(dst(p, v[k]) > r + EPS){
                p = getCenter(v[i], v[j], v[k]); r = dst(v[k], p);
            }
        }
    }
    return {p, r}; }

```

2.14 $O(N + Q \log N)$ K-D Tree

```

T GetDist(const P &a, const P &b){ return (a.x-b.x) * (a.x-b.x)
+ (a.y-b.y) * (a.y-b.y); }

struct Node{
    P p; int idx;
    T x1, y1, x2, y2;

```

```

Node(const P &p, const int idx) : p(p), idx(idx), x1(1e9),
y1(1e9), x2(-1e9), y2(-1e9) {}
bool contain(const P &pt) const{ return x1 <= pt.x && pt.x <=
x2 && y1 <= pt.y && pt.y <= y2; }
T dist(const P &pt) const { return idx == -1 ? INF :
GetDist(p, pt); }
T dist_to_border(const P &pt) const {
    const auto [x,y] = pt;
    if(x1 <= x && x <= x2) return min((y-y1)*(y-y1),
(y2-y)*(y2-y));
    if(y1 <= y && y <= y2) return min((x-x1)*(x-x1),
(x2-x)*(x2-x));
    T t11 = GetDist(pt, {x1,y1}), t12 = GetDist(pt, {x1,y2});
    T t21 = GetDist(pt, {x2,y1}), t22 = GetDist(pt, {x2,y2});
    return min({t11, t12, t21, t22});
}
};

template<bool IsFirst = 1> struct Cmp {
    bool operator() (const Node &a, const Node &b) const {
        return IsFirst ? a.p.x < b.p.x : a.p.y < b.p.y; }
};

struct KDTree { // Warning : no duplicate
    constexpr static size_t NAIVE_THRESHOLD = 16;
    vector<Node> tree;
    KDTree() = default;
    explicit KDTree(const vector<P> &v) {
        for(int i=0; i<v.size(); i++) tree.emplace_back(v[i], i);
        Build(0, v.size());
    }
    template<bool IsFirst = 1>
    void Build(int l, int r) {
        if(r - l <= NAIVE_THRESHOLD) return;
        const int m = (l + r) >> 1;
        nth_element(tree.begin()+l, tree.begin()+m, tree.begin()+r,
Cmp<IsFirst>{});
        for(int i=l; i<r; i++){
            tree[m].x1 = min(tree[m].x1, tree[i].p.x); tree[m].y1 =
min(tree[m].y1, tree[i].p.y);
            tree[m].x2 = max(tree[m].x2, tree[i].p.x); tree[m].y2 =
max(tree[m].y2, tree[i].p.y);
        }
        Build<!IsFirst>(l, m); Build<!IsFirst>(m + 1, r);
    }
    template<bool IsFirst = 1>
    void Query(const P &p, int l, int r, Node &res) const {
        if(r - l <= NAIVE_THRESHOLD){
            for(int i=l; i<r; i++) if(p != tree[i].p && res.dist(p) >
tree[i].dist(p)) res = tree[i];
        }
        else{ // else 1
            const int m = (l + r) >> 1;
            const T t = IsFirst ? p.x - tree[m].p.x : p.y -
tree[m].p.y;
            if(p != tree[m].p && res.dist(p) > tree[m].dist(p)) res =
tree[m];
            if(!tree[m].contain(p) && tree[m].dist_to_border(p) >=
res.dist(p)) return;
            if(t < 0){
                Query<!IsFirst>(p, l, m, res);
                if(t*t < res.dist(p)) Query<!IsFirst>(p, m+1, r, res);
            }
        }
    }
};

```

```

        else{ // else 2
            Query<!IsFirst>(p, m+1, r, res);
            if(t*t < res.dist(p)) Query<!IsFirst>(p, l, m, res);
        } /*else 1*/ } /*else 2*/ } /*void Query*/
    int Query(const P &p) const {
        Node ret(make_pair<T>(1e9, 1e9), -1); Query(p, 0,
tree.size(), ret); return ret.idx; }
};

```

2.15 $O(N \log N)$ Voronoi Diagram

```

/*
input: order will be changed, sorted by (y,x) order
vertex: voronoi intersection points, degree 3, may duplicated
edge: may contain inf line (-1)
area
    - (a,b) = i-th element of area
    - (u,v) = i-th element of edge
    - input[a] is located CCW of u->v line
    - input[b] is located CW of u->v line
    - u->v line is a subset of perpendicular bisector of input[a]
to input[b] segment
    - Straight line {a, b}, {-1, -1} through midpoint of input[a]
and input[b]
*/
const double EPS = 1e-9;
int dcmp(double x){ return x < -EPS ? -1 : x > EPS ? 1 : 0; }
// sq(x) = x*x, size(p) = hypot(p.x, p.y)
// sz2(p) = sq(p.x)+sq(p.y), r90(p) = (-p.y, p.x)
double sq(double x){ return x*x; }
double size(pdd p){ return hypot(p.x, p.y); }
double sz2(pdd p){ return sq(p.x) + sq(p.y); }
pdd r90(pdd p){ return pdd(-p.y, p.x); }
pdd line_intersect(pdd a, pdd b, pdd u, pdd v){ return u +
(((a-u)/b) / ((v/b))*v); }
pdd get_circumcenter(pdd p0, pdd p1, pdd p2){
    return line_intersect(0.5 * (p0+p1), r90(p0-p1), 0.5 *
(p1+p2), r90(p1-p2)); }
double pb_int(pdd left, pdd right, double sweepline){
    if(dcmp(left.y - right.y) == 0) return (left.x + right.x) /
2.0;
    int sign = left.y < right.y ? -1 : 1;
    pdd v = line_intersect(left, right-left, pdd(0, sweepline),
pdd(1, 0));
    double d1 = sz2(0.5 * (left+right) - v), d2 = sz2(0.5 *
(left-right));
    return v.x + sign * sqrt(std::max(0.0, d1 - d2)); }

struct Beachline{
    struct node{ node(){}
        node(pdd point, int idx):point(point), idx(idx), end(0),
link{0, 0}, par(0), prv(0), nxt(0) {}
        pdd point; int idx; int end;
        node *link[2], *par, *prv, *nxt; };
    node *root;
    double sweepline;
    Beachline() : sweepline(-1e20), root(NULL){ }
    inline int dir(node *x){ return x->par->link[0] != x; }
    void rotate(node *n){
        node *p = n->par; int d = dir(n);
        p->link[d] = n->link[!d];
        if(n->link[!d] n->link[!d]->par = p;
        n->par = p->par; if(p->par) p->par->link[dir(p)] = n;
    }
};

```

```

n->link[!d] = p; p->par = n;
} void splay(node *x, node *f = NULL){
    while(x->par != f){
        if(x->par->par == f);
        else if(dir(x) == dir(x->par)) rotate(x->par);
        else rotate(x);
        rotate(x);
    }
    if(f == NULL) root = x;
} void insert(node *n, node *p, int d){
    splay(p); node* c = p->link[d];
    n->link[d] = c; if(c) c->par = n;
    p->link[d] = n; n->par = p;
    node *prv = !d?p->prv:p, *nxt = !d?p:p->nxt;
    n->prv = prv; if(prv) prv->nxt = n;
    n->nxt = nxt; if(nxt) nxt->prv = n;
} void erase(node* n){
    node *prv = n->prv, *nxt = n->nxt;
    if(!prv && !nxt){ if(n == root) root = NULL; return; }
    n->prv = NULL; if(prv) prv->nxt = nxt;
    n->nxt = NULL; if(nxt) nxt->prv = prv;
    splay(n);
    if(!nxt){
        root->par = NULL; n->link[0] = NULL;
        root = prv; }
    else{
        splay(nxt, n); node* c = n->link[0];
        nxt->link[0] = c; c->par = nxt; n->link[0] = NULL;
        n->link[1] = NULL; nxt->par = NULL;
        root = nxt; }
} bool get_event(node* cur, double &next_sweep){
    if(!cur->prv || !cur->nxt) return false;
    pdd u = r90(cur->point - cur->prv->point);
    pdd v = r90(cur->nxt->point - cur->point);
    if(dcmp(u/v) != 1) return false;
    pdd p = get_circumcenter(cur->point, cur->prv->point,
        cur->nxt->point);
    next_sweep = p.y + size(p - cur->point); return true;
} node* find_bl(double x){
    node* cur = root;
    while(cur){
        double left = cur->prv ? pb_int(cur->prv->point,
            cur->point, sweepline) : -1e30;
        double right = cur->nxt ? pb_int(cur->point,
            cur->nxt->point, sweepline) : 1e30;
        if(left <= x && x <= right){ splay(cur); return cur; }
        cur = cur->link[x > right]; }
}
}; using BNode = Beachline::node;
static BNode* arr;
static int sz;
static BNode* new_node(pdd point, int idx){
    arr[sz] = BNode(point, idx); return arr + (sz++); }
struct event{
    event(double sweep, int idx):type(0), sweep(sweep), idx(idx){}
    event(double sweep, BNode* cur):type(1), sweep(sweep),
        prv(cur->prv->idx), cur(cur), nxt(cur->nxt->idx){}
    int type, idx, prv, nxt; BNode* cur; double sweep;
    bool operator>(const event &l)const{ return sweep > l.sweep; }
};
void VoronoiDiagram(vector<pdd> &input, vector<pdd> &vertex,
    vector<pii> &edge, vector<pii> &area){

```

```

Beachline bl = Beachline();
priority_queue<event, vector<event>, greater<event>> events;
auto add_edge = [&](int u, int v, int a, int b, BNode* c1,
    BNode* c2){
    if(c1) c1->end = edge.size()*2;
    if(c2) c2->end = edge.size()*2 + 1;
    edge.emplace_back(u, v); area.emplace_back(a, b);
};
auto write_edge = [&](int idx, int v){ idx%2 == 0 ?
    edge[idx/2].x = v : edge[idx/2].y = v; };
auto add_event = [&](BNode* cur){ double nxt;
    if(bl.get_event(cur, nxt)) events.emplace(nxt, cur); };
int n = input.size(), cnt = 0;
arr = new BNode[n*4]; sz = 0;
sort(input.begin(), input.end(), [](const pdd &l, const pdd
    &r){
        return l.y != r.y ? l.y < r.y : l.x < r.x; });
BNode* tmp = bl.root = new_node(input[0], 0), *t2;
for(int i = 1; i < n; i++){
    if(dcmp(input[i].y - input[0].y) == 0){
        add_edge(-1, -1, i-1, i, 0, tmp);
        bl.insert(t2 = new_node(input[i], i), tmp, 1);
        tmp = t2;
    }
    else events.emplace(input[i].y, i);
}
while(events.size()){
    event q = events.top(); events.pop();
    BNode *prv, *cur, *nxt, *site;
    int v = vertex.size(), idx = q.idx;
    bl.sweepline = q.sweep;
    if(q.type == 0){
        pdd point = input[idx];
        cur = bl.find_bl(point.x);
        bl.insert(site = new_node(point, idx), cur, 0);
        bl.insert(prv = new_node(cur->point, cur->idx), site, 0);
        add_edge(-1, -1, cur->idx, idx, site, prv);
        add_event(prv); add_event(cur);
    }
    else{
        cur = q.cur, prv = cur->prv, nxt = cur->nxt;
        if(!prv || !nxt || prv->idx != q.prv || nxt->idx != q.nxt)
            continue;
        vertex.push_back(get_circumcenter(prv->point, nxt->point,
            cur->point));
        write_edge(prv->end, v); write_edge(cur->end, v);
        add_edge(v, -1, prv->idx, nxt->idx, 0, prv);
        bl.erase(cur);
        add_event(prv); add_event(nxt);
    }
}
delete arr;
}

```

3 Graph

3.1 Euler Tour

```

// Not Directed / Cycle
constexpr int SZ = 1010;
int N, G[SZ][SZ], Deg[SZ], Work[SZ];
void DFS(int v){

```

```

for(int &i=Work[v]; i<=N; i++) while(G[v][I]) G[v][i]--,
    G[i][v]--, DFS(i);
cout << v << " ";
}
// Directed / Path
void DFS(int v){
    for(int i=1; i<=pv; i++) while(G[v][i]) G[v][i]--, DFS(i);
    Path.push_back(v);
}
void Get(){
    for(int i=1; i<=pv; i++) if(In[i] < Out[i]){ DFS(i); return; }
    for(int i=1; i<=pv; i++) if(Out[i]){ DFS(i); return; }
}
// WARNING: path.size() == M + 1 && not trail

```

3.2 2-SAT

```

int SZ; vector<vector<int>> G1, G2;
void Init(int n){ SZ = n; G1 = G2 = vector<vector<int>>(SZ*2); }
int New(){
    for(int i=0; i<2; i++) G1.emplace_back(), G2.emplace_back();
    return SZ++;
}
inline void AddEdge(int s, int e){ G1[s].push_back(e);
    G2[e].push_back(s); }
// T(x) = x << 1, F(x) = x << 1 | 1, I(x) = x ^ 1
inline void AddCNF(int a, int b){ AddEdge(I(a), b);
    AddEdge(I(b), a); }
void MostOne(vector<int> vec){
    compress(vec);
    for(int i=0; i<vec.size(); i++){
        int now = New();
        AddEdge(vec[i], T(now)); AddEdge(F(now), I(vec[i]));
        if(i == 0) continue;
        AddEdge(T(now-1), T(now)); AddEdge(F(now), F(now-1));
        AddEdge(T(now-1), I(vec[i])); AddEdge(vec[i], F(now-1));
    }
}

```

3.3 Horn SAT

```

/* n : number of variance
{ }, 0 : x1 | {0, 1}, 2 : (x1 and x2) => x3, (-x1 or -x2 or x3)
fail -> empty vector */
vector<int> HornSAT(int n, const vector<vector<int>> &cond,
    const vector<int> &val){
    int m = cond.size(); vector<int> res(n), margin(m), stk;
    vector<vector<int>> gph(n);
    for(int i=0; i<m; i++){
        margin[i] = cond[i].size();
        if(cond[i].empty()) stk.push_back(i);
        for(auto j : cond[i]) gph[j].push_back(i);
    }
    while(!stk.empty()){
        int v = stk.back(), h = val[v]; stk.pop_back();
        if(h < 0) return vector<int>();
        if(res[h]) continue; res[h] = 1;
        for(auto i : gph[h]) if(!--margin[i]) stk.push_back(i);
    }
    return res;
}

```

3.4 2-QBF

```

// con[i] \in {A(V), E(Ξ)}, 0-based string
// variable: 1-based(parameter), 0-based(computing)

```



```
// (a or not b) -> {a, -b} in 1-based index
// return empty vector if satisfiable, else any solution
// T(x) = x << 1, F(x) = x << 1 | 1, I(x) = x ^ 1
vector<int> TwoQBF(int n, string con, vector<pair<int,int>>
cnf){
    auto f = [](int v){ return v > 0 ? T(v-1) : F(-v-1); };
    for(auto &a,b : cnf) AddCNF(a=f(a), b=f(b));
    if(!TwoSAT(n)) return {}; int k = SCC.size();
    vector<int> has(k,-1), from(k), to(k), res(n, -1);
    for(int i=n-1; i>=0; i--){ // WARNING: index is scc id
        if(has[C[T(i)]] != -1 || has[C[F(i)]] != -1) return {};
        if(con[i] == 'A') has[C[T(i)]] = T(i), has[C[F(i)]] = F(i);
    }
    for(int i=0; i<k; i++) if(has[i] != -1) from[i] = to[i] = 1;
    for(int i=0; i<n+n; i++){
        for(auto j : Gph[i]) if(C[i] != C[j])
            DAG[C[i]].push_back(C[j]);
    }
    for(int i=k-1; i>=0; i--){
        bool flag = false; // i -> A?
        for(auto j : DAG[i]) flag |= to[j];
        if(flag && to[i]) return {}; to[i] |= flag;
    }
    for(int i=0; i<k; i++) for(auto j : DAG[i]) from[j] |=
from[i]; // A->i?
    for(int i=0; i<k; i++){
        if(has[i] != -1) for(auto v : SCC[i]) res[v/2] = v % 2 ==
has[i] % 2;
        else if(from[i]) for(auto v : SCC[i]) res[v/2] = v % 2 == 0;
        else if(to[i]) for(auto v : SCC[i]) res[v/2] = v % 2 == 1;
    }
    for(int i=0; i<n; i++) if(res[i]==-1) res[i]=C[F(i)]<C[T(i)];
    return res;
}
```

3.5 BCC

```
// Call tarjan(N) before use!!!
vector<int> G[MAX_V]; int In[MAX_V], Low[MAX_V], P[MAX_V];
void addEdge(int s,int e){G[s].push_back(e);G[e].push_back(s);}
void tarjan(int n){ /// Pre-Process
    int pv = 0;
    function<void(int,int)> dfs = [&pv,&dfs](int v, int b){
        In[v] = Low[v] = ++pv; P[v] = b;
        for(auto i : G[v]){
            if(i == b) continue;
            if(!In[i]) dfs(i, v), Low[v] = min(Low[v], Low[i]); else
Low[v] = min(Low[v], In[i]);
        }
        for(int i=1; i<=n; i++) if(!In[i]) dfs(i, -1);
    }
}
```

```
vector<int> cutVertex(int n){
    vector<int> res; array<char,MAX_V> isCut; isCut.fill(0);
    function<void(int)> dfs = [&dfs,&isCut](int v){
        int ch = 0;
        for(auto i : G[v]){
            if(P[i] != v) continue; dfs(i); ch++;
            if(P[v] == -1 && ch > 1) isCut[v] = 1;
            else if(P[v] != -1 && Low[i] >= In[v]) isCut[v]=1;
        }
        for(int i=1; i<=n; i++) if(P[i] == -1) dfs(i);
        for(int i=1; i<=n; i++) if(isCut[i]) res.push_back(i);
        return move(res);
    }
}
```

```
}
vector<PII> cutEdge(int n){
    vector<PII> res;
    function<void(int)> dfs = [&dfs,&res](int v){
        for(int t=0; t<G[v].size(); t++){
            int i = G[v][t]; if(t != 0 && G[v][t-1] == G[v][t])
                continue;
            if(P[i] != v) continue; dfs(i);
            if((t+1 == G[v].size() || i != G[v][t+1]) && Low[i] >
In[v]) res.emplace_back(min(v,i), max(v,i));
        }; // sort edges if multi edge exist
        for(int i=1; i<=n; i++) sort(G[i].begin(), G[i].end());
        for(int i=1; i<=n; i++) if(P[i] == -1) dfs(i);
        return move(res); // sort(all(res));
    }
}
vector<int> BCC[MAX_V]; // BCC[v] = components which contains v
void vertexDisjointBCC(int n){ // allow multi edge, no self loop
    int cnt = 0; array<char,MAX_V> vis; vis.fill(0);
    function<void(int,int)> dfs = [&dfs,&vis,&cnt](int v, int c){
        vis[v] = 1; if(c > 0) BCC[v].push_back(c);
        for(auto i : G[v]){
            if(vis[i]) continue;
            if(In[v] <= Low[i]) BCC[v].push_back(++cnt), dfs(i, cnt);
            else dfs(i, c);
        }
        for(int i=1; i<=n; i++) if(!vis[i]) dfs(i, 0);
        for(int i=1; i<=n; i++) if(BCC[i].empty())
            BCC[i].push_back(++cnt);
    }
}
void edgeDisjointBCC(int n){ // remove cut edge, do flood fill
}
```

3.6 Prufer Sequence

```
vector<pair<int,int>> Gen(int n, vector<int> a){ // a :
[1,n]^(n-2)
    if(n == 1) return {}; if(n == 2) return { make_pair(1, 2) };
    vector<int> deg(n+1); for(auto i : a) deg[i]++;
    vector<pair<int,int>> res; priority_queue<int> pq;
    for(int i=n; i; i--) if(!deg[i]) pq.emplace(i);
    for(auto i : a){
        res.emplace_back(i, pq.top()); pq.pop();
        if(--deg[i]) pq.push(i);
    }
    int u = pq.top(); pq.pop(); int v = pq.top(); pq.pop();
    res.emplace_back(u, v); return res;
}
```

3.7 $O(3^{V/3})$ Maximal Clique

```
using B = bitset<128>; template<typename F> //0-based
void maximal_cliques(vector<B>&g,F f,B P=~B(),B X={},B R={}){
    if(!P.any()){ if(!X.any()) f(R); return; }
    auto q = (P|X)._Find_first(); auto c = P & ~g[q];
    for(int i=0; i<g.size(); i++) if(c[i]) {
        R[i] = 1; cliques(g, f, P&g[i], X&g[i], R);
        R[i]=P[i]=0; X[i] = 1; } // faster for sparse gph
} // undirected, self loop not allowed,  $O(3^{V/3})$ 
B max_independent_set(vector<vector<int>> g){ //g=adj matrix
    int n = g.size(), i, j; vector<B> G(n); B res{};
    auto chk_mx = [&](B a){ if(a.count()>res.count()) res=a; };
    for(i=0; i<=n; i++) for(int j=0; j<=n; j++)
        if(i!=j && !g[i][j])G[i][j]=1;
    cliques(G, chk_mx); return res; }
}
```

3.8 $O(V \log V)$ Tree Isomorphism

```
struct Tree{ // (M1,M2)=(1e9+7, 1e9+9), P1,P2 = random int
array<sz >= N+2)
    int N; vector<vector<int>> G; vector<pair<int,int>> H;
    vector<int> S, C; // size,centroid
    Tree(int N) : N(N), G(N+2), H(N+2), S(N+2) {}
    void addEdge(int s, int e){ G[s].push_back(e);
G[e].push_back(s); }
    int getCentroid(int v, int b=-1){
        S[v] = 1; // do not merge if-statements
        for(auto i : G[v]) if(i!=b) if(int now=getCentroid(i,v);
now<=N/2) S[v]+=now; else break;
        if(N - S[v] <= N/2) C.push_back(v); return S[v] = S[v];
    }
    int init(){
        getCentroid(1); if(C.size() == 1) return C[0];
        int u = C[0], v = C[1], add = ++N;
        G[u].erase(find(G[u].begin(), G[u].end(), v));
        G[v].erase(find(G[v].begin(), G[v].end(), u));
        G[add].push_back(u); G[u].push_back(add);
        G[add].push_back(v); G[v].push_back(add);
        return add;
    }
    pair<int,int> build(const vector<ll> &P1, const vector<ll>
&P2, int v, int b=-1){
        vector<pair<int,int>> ch; for(auto i : G[v]) if(i != b)
ch.push_back(build(P1, P2, i, v));
        ll h1 = 0, h2 = 0; stable_sort(ch.begin(), ch.end());
        if(ch.empty()){ return {1, 1}; }
        for(int i=0; i<ch.size(); i++)
            h1=(h1+(ch[i].first^P1[P1.size()-1-i])*P1[i])%M1,
            h2=(h2+(ch[i].second^P2[P2.size()-1-i])*P2[i])%M2;
        return H[v] = {h1, h2};
    }
    int build(const vector<ll> &P1, const vector<ll> &P2){
        int rt = init(); build(P1, P2, rt); return rt;
    }
};
```

3.9 $O(E \log E)$ Complement Spanning Forest

```
vector<pair<int,int>> ComplementSpanningForest(int n, const
vector<pair<int,int>> &edges){ // V+ElgV
    vector<vector<int>> g(n);
    for(const auto &[u,v] : edges) g[u].push_back(v),
g[v].push_back(u);
    for(int i=0; i<=n; i++) sort(g[i].begin(), g[i].end());
    set<int> alive;
    for(int i=0; i<=n; i++) alive.insert(i);
    vector<pair<int,int>> res;
    while(!alive.empty()){
        int u = *alive.begin(); alive.erase(alive.begin());
        queue<int> que; que.push(u);
        while(!que.empty()){
            int v = que.front(); que.pop();
            for(auto it=alive.begin(); it!=alive.end(); ){
                if(auto t=lower_bound(g[v].begin(), g[v].end(), *it); t
!= g[v].end() && *it == *t) ++it;
                else que.push(*it), res.emplace_back(u, *it), it =
alive.erase(it);
            }
        }
        return res;
    }
}
```

3.10 $O(E\sqrt{V})$ Bipartite Matching, Konig, Dilworth

```
struct HopcroftKarp{
    int n, m; vector<vector<int>> g;
    vector<int> dst, le, ri; vector<char> visit, track;
    HopcroftKarp(int n, int m) : n(n), m(m), g(n), dst(n), le(n, -1), ri(m, -1), visit(n), track(n+m) {}
    void add_edge(int s, int e){ g[s].push_back(e); }
    bool bfs(){ bool res = false; queue<int> que;
        fill(dst.begin(), dst.end(), 0);
        for(int i=0; i<n; i++){ if(le[i] == -1) que.push(i), dst[i]=1;
            while(!que.empty()){ int v = que.front(); que.pop();
                for(auto i : g[v]){
                    if(ri[i] == -1) res = true;
                    else if(!dst[ri[i]]) dst[ri[i]]=dst[v]+1, que.push(ri[i]);
                }
            }
        }
        return res;
    }
    bool dfs(int v){
        if(visit[v]) return false; visit[v] = 1;
        for(auto i : g[v]){
            if(ri[i] == -1 || !visit[ri[i]] && dst[ri[i]] == dst[v] + 1 && dfs(ri[i])){ le[v] = i; ri[i] = v; return true; }
        }
        return false;
    }
    int maximum_matching(){
        int res = 0; fill(all(le), -1); fill(all(ri), -1);
        while(bfs()){
            fill(visit.begin(), visit.end(), 0);
            for(int i=0; i<n; i++) if(le[i] == -1) res += dfs(i);
        }
        return res;
    }
    vector<pair<int,int>> maximum_matching_edges(){
        int matching = maximum_matching();
        vector<pair<int,int>> edges; edges.reserve(matching);
        for(int i=0; i<n; i++) if(le[i] != -1) edges.emplace_back(i, le[i]);
        return edges;
    }
    void dfs_track(int v){
        if(track[v]) return; track[v] = 1;
        for(auto i : g[v]) track[n+i] = 1, dfs_track(ri[i]);
    }
    tuple<vector<int>, vector<int>, int> minimum_vertex_cover(){
        int matching = maximum_matching(); vector<int> lv, rv;
        fill(track.begin(), track.end(), 0);
        for(int i=0; i<n; i++) if(le[i] == -1) dfs_track(i);
        for(int i=0; i<n; i++) if(!track[i]) lv.push_back(i);
        for(int i=0; i<m; i++) if(track[n+i]) rv.push_back(i);
        return {lv, rv, lv.size() + rv.size()}; // s(lv)+s(rv)=mat
    }
    tuple<vector<int>, vector<int>, int> maximum_independent_set(){
        auto [a,b,matching] = minimum_vertex_cover();
        vector<int> lv, rv; lv.reserve(n-a.size());
        rv.reserve(m-b.size());
        for(int i=0, j=0; i<n; i++){
            while(j < a.size() && a[j] < i) j++;
            if(j == a.size() || a[j] != i) lv.push_back(i);
        }
        for(int i=0, j=0; i<m; i++){
```

```
            while(j < b.size() && b[j] < i) j++;
            if(j == b.size() || b[j] != i) rv.push_back(i);
        } // s(lv)+s(rv)=n+m-mat
        return {lv, rv, lv.size() + rv.size()};
    }
    vector<vector<int>> minimum_path_cover(){ // n == m
        int matching = maximum_matching();
        vector<vector<int>> res; res.reserve(n - matching);
        fill(track.begin(), track.end(), 0);
        auto get_path = [&](int v) -> vector<int> {
            vector<int> path{v}; // ri[v] == -1
            while(le[v] != -1) path.push_back(v=le[v]);
            return path;
        };
        for(int i=0; i<n; i++) if(!track[n+i] && ri[i] == -1)
            res.push_back(get_path(i));
        return res; // sz(res) = n-mat
    }
    vector<int> maximum_anti_chain(){ // n == m
        auto [a,b,matching] = minimum_vertex_cover();
        vector<int> res; res.reserve(n - a.size() - b.size());
        for(int i=0, j=0, k=0; i<n; i++){
            while(j < a.size() && a[j] < i) j++;
            while(k < b.size() && b[k] < i) k++;
            if((j == a.size() || a[j] != i) && (k == b.size() || b[k] != i)) res.push_back(i);
        }
        return res; // sz(res) = n-mat
    }
};
```

3.11 $O(V^2\sqrt{E})$ Push Relabel

```
template<typename flow_t> struct Edge {
    int u, v, r; flow_t c, f; Edge() = default;
    Edge(int u, int v, flow_t c, int r) : u(u), v(v), r(r), c(c), f(0) {}
};
template<typename flow_t, size_t _Sz> struct PushRelabel {
    using edge_t = Edge<flow_t>;
    int n, b, dist[_Sz], count[_Sz+1];
    flow_t excess[_Sz]; bool active[_Sz];
    vector<edge_t> g[_Sz]; vector<int> bucket[_Sz];
    void clear(){ for(int i=0; i<_Sz; i++) g[i].clear(); }
    void addEdge(int s, int e, flow_t x){
        g[s].emplace_back(s, e, x, (int)g[e].size());
        if(s == e) g[s].back().r++;
        g[e].emplace_back(e, s, 0, (int)g[s].size()-1);
    }
    void enqueue(int v){
        if(!active[v] && excess[v] > 0 && dist[v] < n){
            active[v] = true; bucket[dist[v]].push_back(v); b = max(b, dist[v]);
        }
    }
    void push(edge_t &e){
        flow_t fl = min(excess[e.u], e.c - e.f);
        if(dist[e.u] == dist[e.v] + 1 && fl > flow_t(0)){
            e.f += fl; g[e.v][e.r].f -= fl; excess[e.u] -= fl;
            excess[e.v] += fl; enqueue(e.v);
        }
    }
    void gap(int k){
        for(int i=0; i<n; i++){
```

```
            if(dist[i] >= k) count[dist[i]]--, dist[i] = max(dist[i], n), count[dist[i]]++;
            enqueue(i);
        }
    }
    void relabel(int v){
        count[dist[v]]--; dist[v] = n;
        for(const auto &e : g[v]) if(e.c - e.f > 0) dist[v] = min(dist[v], dist[e.v] + 1);
        count[dist[v]]++; enqueue(v);
    }
    void discharge(int v){
        for(auto &e : g[v]) if(excess[v] > 0) push(e); else break;
        if(excess[v] > 0) if(count[dist[v]] == 1) gap(dist[v]);
        else relabel(v);
    }
    flow_t maximumFlow(int _n, int s, int t){
        // memset dist, excess, count, active 0
        n = _n; b = 0; for(auto &e : g[s]) excess[s] += e.c;
        count[s] = n; enqueue(s); active[t] = true;
        while(b >= 0){
            if(bucket[b].empty()) b--;
            else{
                int v = bucket[b].back(); bucket[b].pop_back();
                active[v] = false; discharge(v);
            } /*else*/ } /*while*/ return excess[t];
        }
    };
};
```

3.12 LR Flow, Circulation

```
struct LR_Flow{ LR_Flow(int n) : F(n+2), S(0) {}
    void add_edge(int s, int e, flow_t l, flow_t r){
        S += abs(l); F.add_edge(s+2, e+2, r-l);
        if(l > 0) F.add_edge(s+2, 1, l), F.add_edge(0, e+2, l);
        else F.add_edge(0, s+2, -l), F.add_edge(e+2, 1, -l);
    }
    Dinic<flow_t, MAX_U> F; flow_t S;
    bool solve(int s, int t){ //maxflow: run F.maximum_flow(s, t)
        if(s != -1) F.add_edge(t+2, s+2, MAX_U); //min cost circ
        return F.maximum_flow(0,1) == S;
    }
    flow_t get_flow(int s, int e) const { s += 2; e += 2;
        for(auto i : F.g[s]) if(i.c > 0 && i.v == e) return i.f;
    };
    struct Circulation{ // demand[i] = in[i] - out[i]
        Circulation(int n, const vector<flow_t> &demand) : F(n+2) {
            // demand[i] > 0: add_edge(0, i+2, demand[i], demand[i])
            // demand[i] <= 0: add_edge(i+2, 1, -demand[i], demand[i])
        }
        LR_Flow<flow_t, MAX_U> F;
        void add_edge(int s, int e, flow_t l, flow_t r){
            F.add_edge(s+2, e+2, l, r);
        }
        bool feasible(){ return F.feasible(0, 1); }
    };
};
```

3.13 Min Cost Circulation

```
template <class T> struct MinCostCirculation {
    const int SCALE = 3; // scale by 1/(1 << SCALE)
    const T INF = numeric_limits<T>::max() / 2;
    struct EdgeStack { int s, e; T l, r, cost; };
    struct Edge { int pos, rev; T rem, cap, cost; };
    int n; vector<vector<Edge>> g; vector<EdgeStack> estk;
    LR_Flow<T, 1LL<<60> circ; vector<T> p;
    MinCostCirculation(int k) : n(k), g(k), circ(k), p(k) {}
    void add_edge(int s, int e, T l, T r, T cost){
        estk.push_back({s, e, l, r, cost});
    }
    pair<bool, T> solve(){
```

```

for(auto &i:estk)if(i.s!=i.e)circ.add_edge(i.s,i.e,i.l,i.r);
if(!circ.solve(-1, -1)) return make_pair(false, T(0));
vector<int> cnt(n); T eps = 0;
for(auto &i : estk){ T curFlow;
    auto &edge = circ.F.g[i.s+2][cnt[i.s]];
    if(i.s != i.e) curFlow = i.r - (edge.c - edge.f);
    else curFlow = i.r;
    int srev = sz(g[i.e]), erev = sz(g[i.s]);
    if(i.s == i.e) srev++;
    g[i.s].push_back(i.e,srev,i.r-curFlow,i.r,i.cost*(n+1));
    g[i.e].push_back(i.s,erev,-i.l+curFlow,-i.l,-i.cost*(n+1));
    eps = max(eps, abs(i.cost) * (n+1));
    if(i.s != i.e) cnt[i.s] += 2, cnt[i.e] += 2;
}
while(true){ eps=0; auto cost=[&](Edge &e, int s, int t){
    return e.cost + p[s] - p[t]; };
    for(int i = 0; i < n; i++) for(auto &e : g[i])
        if(e.rem > 0) eps = max(eps, -cost(e, i, e.pos));
    if(eps <= T(1)) break;
    eps = max(T(1), eps >> SCALE);
    vector<T> excess(n); queue<int> que;
    auto push = [&](Edge &e, int src, T flow){
        e.rem -= flow; g[e.pos][e.rev].rem += flow;
        excess[src] -= flow; excess[e.pos] += flow;
        if(excess[e.pos] <= flow && excess[e.pos] > 0)
            que.push(e.pos);
    }; vector<int> ptr(n);
    auto relabel = [&](int v){
        ptr[v] = 0; p[v] = -INF;
        for(auto &e : g[v])
            if(e.rem>0) p[v] = max(p[v], p[e.pos]-e.cost-eps);
    };
    for(int i = 0; i < n; i++) for(auto &j : g[i])
        if(j.rem>0 && cost(j, i, j.pos)<0) push(j, i, j.rem);
    while(sz(que)){
        int x = que.front(); que.pop();
        while(excess[x] > 0){
            for(; ptr[x] < sz(g[x]); ptr[x]++){
                Edge &e = g[x][ptr[x]];
                if(e.rem > 0 && cost(e, x, e.pos) < 0){
                    push(e, x, min(e.rem, excess[x]));
                    if(excess[x] == 0) break;
                } /* if end */ } /* for end */
            if(excess[x] == 0) break; relabel(x);
        } /* excess end */ } /* que end */
    } /* while true end */ T ans = 0;
    for(int i=0; i<n; i++) for(auto &j : g[i])
        j.cost /= n + 1, ans += j.cost * (j.cap - j.rem);
    return make_pair(true, ans / 2);
}
void bellmanFord(){
    fill(p.begin(), p.end(), T(0)); bool upd = 1;
    while(upd){ upd = 0;
        for(int i = 0; i < n; i++) for(auto &j : g[i])
            if(j.rem > 0 && p[j.pos] > p[i] + j.cost) p[j.pos] =
                p[i] + j.cost, upd = 1;
    }
}
};

```

3.14 $O(V^3)$ Hungarian Method

```

// 1-based, only for min matching, max matching may get TLE
template<typename cost_t=int, cost_t _INF=0x3f3f3f3f>
struct Hungarian{
    int n; vector<vector<cost_t>> mat;
    Hungarian(int n) : n(n), mat(n+1, vector<cost_t>(n+1, _INF)){}
    void addEdge(int s, int e, cost_t x){ mat[s][e] =
        min(mat[s][e], x); }
    pair<cost_t, vector<int>> run(){
        vector<cost_t> u(n+1), v(n+1), m(n+1);
        vector<int> p(n+1), w(n+1), c(n+1);
        for(int i=1,a,b; i<=n; i++){
            p[0] = i; b = 0; fill(m.begin(), m.end(), _INF);
            fill(c.begin(), c.end(), 0);
            do{
                int nxt; cost_t delta = _INF; c[b] = 1; a = p[b];
                for(int j=1; j<=n; j++){
                    if(c[j]) continue;
                    cost_t t = mat[a][j] - u[a] - v[j];
                    if(t < m[j]) m[j] = t, w[j] = b;
                    if(m[j] < delta) delta = m[j], nxt = j;
                }
                for(int j=0; j<=n; j++){
                    if(c[j]) u[p[j]] += delta, v[j] -= delta; else m[j] -=
                        delta;
                }
                b = nxt;
            }while(p[b] != 0);
            do{int nxt = w[b]; p[b] = p[nxt]; b = nxt;}while(b!=0);
        }
        vector<int> assign(n+1);for(int i=1;i<=n;i++)assign[p[i]]=i;
        return {-v[0], assign};
    }
};

```

3.15 $O(V^3)$ Global Min Cut

```

template<typename T, T INF>// 0-based, adj matrix
pair<T, vector<int>> GetMinCut(vector<vector<T>> g){
    int n=g.size(); vector<int> use(n), cut, mn_cut; T mn=INF;
    for(int phase=n-1; phase>=0; phase--){
        vector<int> w=g[0], add=use; int k=0, prv;
        for(int i=0; i<phase; i++){ prv = k; k = -1;
            for(int j=1; j<n; j++) if(!add[j] && (k== -1 || w[j] >
                w[k])) k=j;
            if(i + 1 < phase){
                for(int j=0; j<n; j++) w[j] += g[k][j];
                add[k] = 1; continue; }
            for(int j=0; j<n; j++) g[prv][j] += g[k][j];
            for(int j=0; j<n; j++) g[j][prv] = g[prv][j];
            use[k] = 1; cut.push_back(k);
            if(w[k] < mn) mn_cut = cut, mn = w[k];
        }
    }
    return {mn, mn_cut};
}

```

3.16 $O(V^2 + V \times \text{Flow})$ Gomory-Hu Tree

```

//0-based, S-T cut in graph=S-T cut in gomory-hu tree (path min)
vector<Edge> GomoryHuTree(int n, const vector<Edge> &e){
    Dinic<int,100> Flow; vector<Edge> res(n-1); vector<int> pr(n);
    for(int i=1; i<n; i++, Flow.clear()){ // // bi-directed edge

```

```

for(const auto &[s,e,x] : e) Flow.AddEdge(s, e, x);
int fl = Flow.MaxFlow(pr[i], i);
for(int j=i+1; j<n; j++){
    if(!Flow.Level[i] == !Flow.Level[j] && pr[i] == pr[j])
        pr[j] = i;
    } /*for-j end*/ res[i-1] = Edge(pr[i], i, fl);
} /*for-i end*/ return res; }

```

3.17 $O(V + E\sqrt{V})$ Count/Find 3/4 Cycle

```

vector<tuple<int,int,int>> Find3Cycle(int n, const
vector<pair<int,int>> &edges){ // N+MsqrtN
    int m = edges.size();
    vector<int> deg(n), pos(n), ord; ord.reserve(n);
    vector<vector<int>> gph(n), que(m+1), vec(n);
    vector<vector<tuple<int,int,int>>> tri(n);
    vector<tuple<int,int,int>> res;
    for(auto [u,v] : edges) deg[u]++, deg[v]++;
    for(int i=0; i<n; i++) que[deg[i]].push_back(i);
    for(int i=m; i>=0; i--) ord.insert(ord.end(), que[i].begin(),
        que[i].end());
    for(int i=0; i<n; i++) pos[ord[i]] = i;
    for(auto [u,v] : edges) gph[pos[u]].push_back(pos[v]),
        gph[pos[v]].push_back(pos[u]);
    for(int i=0; i<n; i++){
        for(auto j : gph[i]){
            if(i > j) continue;
            for(int x=0, y=0; x<vec[i].size() && y<vec[j].size(); ){
                if(vec[i][x] == vec[j][y]) res.emplace_back(ord[i],
                    ord[j], ord[vec[i][x]]), x++, y++;
                else if(vec[i][x] < vec[j][y]) x++; else y++;
            }
            vec[j].push_back(i);
        }
    }
    for(auto &[u,v,w] : res){
        if(pos[u] < pos[v]) swap(u, v);
        if(pos[u] < pos[w]) swap(u, w);
        if(pos[v] < pos[w]) swap(v, w);
        tri[u].emplace_back(u, v, w);
    }
    res.clear();
    for(int i=n-1; i>=0; i--) res.insert(res.end(),
        tri[ord[i]].begin(), tri[ord[i]].end());
    return res;
}
bitset<500> B[500]; // N3/w
long long Count3Cycle(int n, const vector<pair<int,int>>
&edges){
    long long res = 0;
    for(int i=0; i<n; i++) B[i].reset();
    for(auto [u,v] : edges) B[u].set(v), B[v].set(u);
    for(int i=0; i<n; i++) for(int j=i+1; j<n; j++)
        if(B[i].test(j)) res += (B[i] & B[j]).count();
    return res / 3;
}
// 0(n + m * sqrt(m) + th) for graphs without loops or
multiedges
void Find4Cycle(int n, const vector<array<int, 2>> &edge, auto
process, int th = 1){
    int m = (int)edge.size();
    vector<int> deg(n), order, pos(n);

```



```

vector<vector<int>> appear(m+1), adj(n), found(n);
for(auto [u, v]: edge) ++deg[u], ++deg[v];
for(auto u=0; u<n; u++) appear[deg[u]].push_back(u);
for(auto d=m; d>=0; d--) order.insert(order.end(),
appear[d].begin(), appear[d].end());
for(auto i=0; i<n; i++) pos[order[i]] = i;
for(auto i=0; i<m; i++){
    int u = pos[edge[i][0]], v = pos[edge[i][1]];
    adj[u].push_back(v), adj[v].push_back(u);
}
T res = 0; vector<int> cnt(n);
for(auto u=0; u<n; u++){
    for(auto v: adj[u]) if(u < v) for(auto w: adj[v]) if(u < w)
        cnt[w] = 0;
    for(auto v: adj[u]) if(u < v) for(auto w: adj[v]) if(u < w)
        res += cnt[w] ++;
}
for(auto u=0; u<n; u++){
    for(auto v: adj[u]) if(u < v) for(auto w: adj[v]) if(u < w)
        found[w].clear();
    for(auto v: adj[u]) if(u < v) for(auto w: adj[v]) if(u < w)
    {
        for(auto x: found[w]){
            if(!th--) return;
            process(order[u], order[v], order[w], order[x]);
        }
        found[w].push_back(v);
    }
}
}
}

```

3.18 $O(V \log V)$ Rectilinear MST

```

template<class T> vector<tuple<T, int, int>>
rectilinear_minimum_spanning_tree(vector<point<T>> a){
    int n = a.size(); vector<int> ind(n);
    iota(ind.begin(), ind.end(), 0); vector<tuple<T, int, int>> edge;
    for(int k=0; k<4; k++){ map<T, int> mp;
        sort(ind.begin(), ind.end(), [&](int i, int j){
            return a[i].x-a[j].x < a[j].y-a[i].y;});
        for(auto i: ind){
            for(auto it=mp.lower_bound(-a[i].y); it!=mp.end();
            it=mp.erase(it)){
                int j = it->second; point<T> d = a[i] - a[j];
                if(d.y > d.x) break; edge.push_back({d.x+d.y, i, j});
            }
            mp.insert({-a[i].y, i});
        }
        for(auto &p: a) if(k & 1) p.x = -p.x; else swap(p.x, p.y);
    } /*for-k end*/ sort(edge.begin(), edge.end());
    disjoint_set dsu(n); vector<tuple<T, int, int>> res;
    for(auto [x, i, j]: edge) if(dsu.merge(i, j))
        res.push_back({x, i, j});
    return res; }

```

3.19 $O(VE)$ Shortest Mean Cycle

```

template<typename T, T INF> vector<int> // T = V*E*max(C)
min_mean_cycle(int n, const vector<tuple<int, int, T>> &edges){
    vector<vector<T>> dp(n+1, vector<T>(n, INF)); // int support!
    vector<vector<int>> pe(n+1, vector<int>(n, -1));
    fill(dp[0].begin(), dp[0].end(), 0); // 0-based, directed
    for(int x=1; x<=n; x++){ int id=0; // bellman

```

```

for(auto [u, v, w] : edges){
    if(dp[x-1][u] != INF && dp[x-1][u] + w < dp[x][v])
        dp[x][v] = dp[x-1][u] + w, pe[x][v] = id;
    id++; } // range based for end!
} T p=1; int q=0, src=-1; //fraction
for(auto u=0; u<n; u++){ if(dp[n][u] == INF) continue;
    T cp=-1, cq=0; // ↓ overflow!!!
    for(int x=0; x<=n; x++){ if(cp*(n-x) < (dp[n][u]-dp[x][u])*cq)
        cp = dp[n][u] - dp[x][u], cq = n - x;
        if(p * cq > cp * q) src = u, p = cp, q = cq;
    } if(src == -1) return {};
    vector<int> res, po(n, -1);
    for(int u=src, x = n; ; u=get<0>(edges[pe[x--][u]])){
        if(po[u] != -1) return u;
        vector<int>{res.rbegin(), res.rend()-po[u]};
        po[u] = res.size(); res.push_back(pe[x][u]);
    } assert(false);
} // return edge index

```

3.20 $O(V^2)$ Stable Marriage Problem

```

// man : 1~n, woman : n+1~2n
struct StableMarriage{
    int n; vector<vector<int>> g;
    StableMarriage(int n) : n(n), g(2*n+1) { for(int i=1; i<=n+n;
    i++) g[i].reserve(n); }
    void addEdge(int u, int v){ g[u].push_back(v); } // insert in
    decreasing order of preference.
    vector<int> run(){
        queue<int> q; vector<int> match(2*n+1), ptr(2*n+1);
        for(int i=1; i<=n; i++) q.push(i);
        while(q.size()){
            int i = q.front(); q.pop();
            for(int &p=ptr[i]; p<g[i].size(); p++){
                int j = g[i][p];
                if(!match[j]){ match[i] = j; match[j] = i; break; }
                int m = match[j], u = -1, v = -1;
                for(int k=0; k<g[j].size(); k++){
                    if(g[j][k] == i) u = k; if(g[j][k] == m) v = k;
                }
                if(u < v){
                    match[m] = 0; q.push(m); match[i] = j; match[j] = i;
                    break;
                }
            } /*if u < v*/ } /*for-p*/ } /*while*/
            return match; } /*vector<int> run*/
    };
}

```

3.21 $O((V + E) \log V)$ Dominator Tree

```

vector<int> DominatorTree(const vector<vector<int>> &g, int
src){ // /// 0-based
    int n = g.size();
    vector<vector<int>> rg(n), buf(n);
    vector<int> r(n), val(n), idom(n, -1), sdom(n, -1), o, p(n),
    u(n);
    iota(all(r), 0); iota(all(val), 0);
    for(int i=0; i<n; i++) for(auto j : g[i]) rg[j].push_back(i);
    function<int(int)> find = [&](int v){
        if(v == r[v]) return v;
        int ret = find(r[v]);
        if(sdom[val[v]] > sdom[val[r[v]]]) val[v] = val[r[v]];
        return r[v] = ret;
    };
}

```

```

function<void(int)> dfs = [&](int v){
    sdom[v] = o.size(); o.push_back(v);
    for(auto i : g[v]) if(sdom[i] == -1) p[i] = v, dfs(i);
};
dfs(src); reverse(all(o));
for(auto &i : o){
    if(sdom[i] == -1) continue;
    for(auto j : rg[i]){
        if(sdom[j] == -1) continue;
        int x = val[find(j), j];
        if(sdom[i] > sdom[x]) sdom[i] = sdom[x];
    }
    buf[o.size() - sdom[i] - 1].push_back(i);
    for(auto j : buf[p[i]]) u[j] = val[find(j), j];
    buf[p[i]].clear();
    r[i] = p[i];
}
reverse(all(o)); idom[src] = src;
for(auto i : o){ // WARNING : if different, takes idom
    if(i != src) idom[i] = sdom[i] == sdom[u[i]] ? sdom[i] :
    idom[u[i]];
}
for(auto i : o) if(i != src) idom[i] = o[idom[i]];
return idom; // unreachable -> ret[i] = -1
}

```

3.22 $O(VE)$ Vizing Theorem

```

// Graph coloring with (max-degree)+1 colors,  $O(N^2)$ 
int C[MX][MX] = {}, G[MX][MX] = {}; // MX ~ 2500
void solve(vector<pii> &E, int N, int M){
    int X[MX] = {}, a, b;
    auto update = [&](int u){ for(X[u] = 1; C[u][X[u]]; X[u]++);
    };
    auto color = [&](int u, int v, int c){
        int p = G[u][v]; G[u][v] = G[v][u] = c;
        C[u][c] = v; C[v][c] = u; C[u][p] = C[v][p] = 0;
        if( p ) X[u] = X[v] = p; else update(u), update(v);
        return p; }; // end of function : color
    auto flip = [&](int u, int c1, int c2){
        int p = C[u][c1], q = C[u][c2];
        swap(C[u][c1], C[u][c2]);
        if( p ) G[u][p] = G[p][u] = c2;
        if( !C[u][c1] ) X[u] = c1; if( !C[u][c2] ) X[u] = c2;
        return p; }; // end of function : flip
    for(int i = 1; i <= N; i++) X[i] = 1;
    for(int t = 0; t < E.size(); t++){
        int u=E[t].first, v=E[t].second, v=v0, c0=X[u], c=c0, d;
        vector<pii> L; int vst[MX] = {};
        while(!G[u][v0]){
            L.emplace_back(v, d = X[v]);
            if(!C[v][c]) for(a = (int)L.size()-1; a >= 0; a--) c =
            color(u, L[a].first, c);
            else if(!C[u][d]) for(a=(int)L.size()-1; a>=0; a--)
            color(u, L[a].first, L[a].second);
            else if( vst[d] ) break;
            else vst[d] = 1, v = C[u][d];
        }
        if( !G[u][v0] ){
            for(;v; v = flip(v, c, d), swap(c, d));
            if(C[u][c0]){

```



```

    for(a=(int)L.size()-2; a>=0 && L[a].second!=c; a--);
    for(; a >= 0; a--) color(u, L[a].first, L[a].second);
    } else t--;
}
}
}
}

```

3.23 $O(E + V^3 + V3^T + V^22^T)$ Minimum Steiner Tree

```

struct SteinerTree{ // O(E + V^3 + V 3^T + V^2 2^T)
constexpr static int V = 33, T = 8;
int n, G[V][V], D[1<<T][V], tmp[V];
void init(int _n){ n = _n;
memset(G, 0x3f, sizeof G); for(int i=0; i<n; i++) G[i][i]=0;
} void shortest_path(){ /*floyd 0..n-1*/ }
void add_edge(int u, int v, int w){
G[u][v] = G[v][u] = min(G[v][u], w); }
int solve(const vector<int>& ter){
int t = (int)ter.size(); memset(D, 0x3f, sizeof D);
for(int i=0; i<n; i++) D[0][i] = 0;
for(int msk=1; msk<(1<<t); msk++){
if(msk == (msk & (-msk))){ int who = __lg(msk);
for(int i=0; i<n; i++) D[msk][i] = G[ter[who]][i];
continue;
}
for(int i=0; i<n; i++)
for(int sub=(msk-1)&msk; sub; sub=(sub-1)&msk)
D[msk][i] = min(D[msk][i], D[sub][i] + D[msk^sub][i]);
memset(tmp, 0x3f, sizeof tmp);
for(int i=0; i<n; i++) for(int j=0; j<n; j++)
tmp[i] = min(tmp[i], D[msk][j] + G[j][i]);
for(int i=0; i<n; i++) D[msk][i] = tmp[i];
}
return *min_element(D[(1<<t)-1], D[(1<<t)-1]+n);
}
};

```

3.24 $O(E \log V)$ Directed MST

```

using D = int; struct edge { int u, v; D w; };
vector<int> DirectedMST(vector<edge> &e, int n, int root){
using T = pair<D, int>; // 0-based, return index of edges
using PQ = pair<priority_queue<T, vector<T>, greater<T>>, D>;
auto push = [](PQ &pq, T v){
pq.first.emplace(v.first-pq.second, v.second); };
auto top = [](const PQ &pq) -> T {
auto r = pq.first.top(); return {r.first + pq.second,
r.second}; };
auto join = [&push, &top](PQ &a, PQ &b) {
if(a.first.size() < b.first.size()) swap(a, b);
while(!b.first.empty()) push(a, top(b)), b.first.pop(); };
vector<PQ> h(n * 2);
for(int i=0; i<e.size(); i++) push(h[e[i].v], {e[i].w, i});
vector<int> a(n*2), v(n*2, -1), pa(n*2, -1), r(n*2);
iota(a.begin(), a.end(), 0);
auto o = [&](int x) { int y; for(y=x; a[y]!=y; y=a[y]);;
for(int ox=x; x!=y; ox=x) x = a[x], a[ox] = y;
return y; };
v[root] = n + 1; int pc = n;
for(int i=0; i<n; i++) if(v[i] == -1) {
for(int p=i; v[p]==-1 || v[p]==i; p=o(e[r[p]].u)){
if(v[p] == i){ int q = p; p = pc++;
do{ h[q].second = -h[q].first.top().first;

```

```

join(h[pa[q]=a[q]=p], h[q]);
}while((q=o(e[r[q]].u)) != p);
} v[p] = i;
while(!h[p].first.empty() && o(e[top(h[p)].second].u) ==
p) h[p].first.pop();
r[p] = top(h[p]).second;
}
}
vector<int> ans;
for(int i=pc-1; i>=0; i--) if(i != root && v[i] != n) {
for(int f=e[r[i]].v; f!=-1 && v[f] != n; f=pa[f]) v[f] = n;;
ans.push_back(r[i]);
}
return ans;
}

```

3.25 $O(E \log V + K \log K)$ K Shortest Walk

```

int rnd(int l, int r){ /* return random int [l,r] */ }
struct node{ // weight>=0, allow multi edge, self loop
array<node*, 2> son; pair<ll, ll> val;
node() : node(make_pair(-1e18, -1e18)) {}
node(pair<ll, ll> val) : node(nullptr, nullptr, val) {}
node(node *l, node *r, pair<ll, ll> val):son({l,r}),val(val){}
};
node* copy(node *x){ return x ? new node(x->son[0], x->son[1],
x->val) : nullptr; }
node* merge(node *x, node *y){ // precondition: x, y both points
to new entity
if(!x || !y) return x ? x : y;
if(x->val > y->val) swap(x, y);
int rd = rnd(0, 1); if(x->son[rd])
x->son[rd]=copy(x->son[rd]);
x->son[rd] = merge(x->son[rd], y); return x;
}
struct edge{
ll v, c, i; edge() = default;
edge(ll v, ll c, ll i) : v(v), c(c), i(i) {}
};
vector<vector<edge>> gph, rev; int idx;
void init(int n){ gph = rev = vector<vector<edge>>(n); idx=0; }
void add_edge(int s, int e, ll x){
gph[s].emplace_back(e, x, idx);
rev[e].emplace_back(s, x, idx);
assert(x >= 0); idx++;
}
vector<int> par, pae; vector<ll> dist; vector<node*> heap;
void dijkstra(int snk){ // replace this to SPFA if edge weight
is negative
int n = gph.size();
par = pae = vector<int>(n, -1);
dist = vector<ll>(n, 0x3f3f3f3f3f3f3f3f);
heap = vector<node*>(n, nullptr);
priority_queue<pair<ll, ll>, vector<pair<ll, ll>>, greater<>> pq;
auto enqueue = [&](int v, ll c, int pa, int pe){
if(dist[v] > c) dist[v] = c, par[v] = pa, pae[v] = pe,
pq.emplace(c, v);
}; enqueue(snk, 0, -1, -1); vector<int> ord;
while(!pq.empty()){
auto [c,v] = pq.top(); pq.pop(); if(dist[v] != c) continue;
ord.push_back(v); for(auto e : rev[v]) enqueue(e.v, c+e.c,
v, e.i);
}
}

```

```

for(auto &v : ord){
if(par[v] != -1) heap[v] = copy(heap[par[v]]);
for(auto &e : gph[v]){
if(e.i == pae[v]) continue;
ll delay = dist[e.v] + e.c - dist[v];
if(delay < 1e18) heap[v] = merge(heap[v], new
node(make_pair(delay, e.v)));
}
}
}
vector<ll> run(int s, int e, int k){
using state = pair<ll, node*>; dijkstra(e); vector<ll> ans;
priority_queue<state, vector<state>, greater<state>> pq;
if(dist[s] > 1e18) return vector<ll>(k, -1);
ans.push_back(dist[s]);
if(heap[s]) pq.emplace(dist[s] + heap[s]->val.first, heap[s]);
while(!pq.empty() && ans.size() < k){
auto [cst, ptr] = pq.top(); pq.pop(); ans.push_back(cst);
for(int j=0; j<2; j++) if(ptr->son[j])
pq.emplace(cst-ptr->val.first + ptr->son[j]->val.first,
ptr->son[j]);
int v = ptr->val.second;
if(heap[v]) pq.emplace(cst + heap[v]->val.first, heap[v]);
}
while(ans.size() < k) ans.push_back(-1);
return ans;
}

```

3.26 $O(V + E)$ Chordal Graph, Tree Decomposition

```

struct Set { list<int> L; int last; Set() { last = 0; } };
struct PEO {
int N; list<Set> L;
vector<vector<int>> g; vector<int> vis, res;
vector<list<Set>::iterator> ptr;
vector<list<int>::iterator> ptr2;
PEO(int n, vector<vector<int>> > _g) {
N = n; g = _g;
for (int i = 1; i <= N; i++) sort(g[i].begin(), g[i].end());
vis.resize(N + 1); ptr.resize(N + 1); ptr2.resize(N + 1);
L.push_back(Set());
for (int i = 1; i <= N; i++) {
L.back().L.push_back(i);
ptr[i] = L.begin(); ptr2[i] = prev(L.back().L.end());
}
}
}
pair<bool, vector<int>> Run() {
// lexicographic BFS
int time = 0;
while (!L.empty()) {
if (L.front().L.empty()) { L.pop_front(); continue; }
auto it = L.begin();
int n = it->L.front(); it->L.pop_front();
vis[n] = ++time;
res.push_back(n);
for (int next : g[n]) {
if (vis[next]) continue;
if (ptr[next]->last != time) {
L.insert(ptr[next], Set()); ptr[next]->last = time;
}
ptr[next]->L.erase(ptr2[next]); ptr[next]--;
}
}
}

```

```

ptr[next]->L.push_back(next);
ptr2[next] = prev(ptr[next]->L.end());
}
}
// PEO existence check
for (int n = 1; n <= N; n++) {
    int mx = 0;
    for (int next : g[n]) if (vis[n] > vis[next]) mx = max(mx, vis[next]);
    if (mx == 0) continue;
    int w = res[mx - 1];
    for (int next : g[n]) {
        if (vis[w] > vis[next] && !binary_search(g[w].begin(), g[w].end(), next)){
            vector<int> chk(N+1), par(N+1, -1); // w와 next가 이어져 있지 않다면 not chordal
            deque<int> dq{next}; chk[next] = 1;
            while (!dq.empty()) {
                int x = dq.front(); dq.pop_front();
                for (auto y : g[x]) {
                    if (chk[y] || y == n || y != w && binary_search(g[n].begin(), g[n].end(), y)) continue;
                    dq.push_back(y); chk[y] = 1; par[y] = x;
                }
            }
            vector<int> cycle{next, n};
            for (int x=w; x!=next; x=par[x]) cycle.push_back(x);
            return {false, cycle};
        }
    }
}
reverse(res.begin(), res.end());
return {true, res};
}
};
bool vis[200201]; // 배열 크기 알아서 수정하자.
int p[200201], ord[200201], P = 0; // P=경점 개수
vector<int> V[200201], G[200201]; // V=bags, G=edges
void tree_decomposition(int N, vector<vector<int>> g) {
    for(int i=1; i<=N; i++) sort(g[i].begin(), g[i].end());
    vector<int> peo = PEO(N, g).Run(), rpeo = peo;
    reverse(rpeo.begin(), rpeo.end());
    for(int i=0; i<peo.size(); i++) ord[peo[i]] = i;
    for(int n : rpeo) { // tree decomposition
        vis[n] = true;
        if (n == rpeo[0]) { // 처음
            P++; V[P].push_back(n); p[n] = P; continue;
        }
        int mn = INF, idx = -1;
        for(int next : g[n]) if (vis[next] && mn > ord[next]) mn = ord[next], idx = next;
        assert(idx != -1); idx = p[idx];
        // 두 set인 V[idx]와 g[n](visited ver)가 같나?
        // V[idx]의 모든 원소가 g[n]에서 나타나는지 판별로 충분하다.
        int die = 0;
        for(int x : V[idx]) {
            if (!binary_search(g[n].begin(), g[n].end(), x)) { die = 1; break; }
        }
    }
}

```

```

if (!die) { V[idx].push_back(n), p[n] = idx; } // 기존 집합에 추가
else { // 새로운 집합을 자식으로 추가
    P++;
    G[idx].push_back(P); // 자식으로부터 단방향으로 잇자.
    V[P].push_back(n);
    for(int next : g[n]) if (vis[next]) V[P].push_back(next);
    p[n] = P;
}
}
for(int i=1; i<=P; i++) sort(V[i].begin(), V[i].end());
}

```

3.27 $O(V^3)$ General Matching

```

int N, M, R, Match[555], Par[555], Chk[555], Prv[555], Vis[555];
vector<int> G[555]; // n 500 20ms
int Find(int x){return x == Par[x] ? x : Par[x] = Find(Par[x]);}
int LCA(int u, int v){ static int cnt = 0;
    for(cnt++; Vis[u]!=cnt; swap(u, v)) if(u) Vis[u] = cnt, u = Find(Prv[Match[u]]);
    return u; }
void Blossom(int u, int v, int rt, queue<int> &q){
    for(; Find(u)!=rt; u=Prv[v]){
        Prv[u] = v; Par[u] = Par[v=Match[u]] = rt;
        if(Chk[v] & 1) q.push(v), Chk[v] = 2;
    }
}
bool Augment(int u){ // iota Par 0, fill Chk 0
    queue<int> Q; Q.push(u); Chk[u] = 2;
    while(!Q.empty()){ u = Q.front(); Q.pop();
        for(auto v : G[u]){
            if(Chk[v] == 0){
                Prv[v]=u; Chk[v]=1; Q.push(Match[v]); Chk[Match[v]]=2;
                if(!Match[v]){ for(; u; v=u) u = Match[Prv[v]], Match[Match[v]=Prv[v]] = v;; return true; }
            }
            else if(Chk[v] == 2){ int l = LCA(u, v); Blossom(u, v, l, Q), Blossom(v, u, l, Q); }
        } /* for v */ } /* while */
    return 0; }
void Run(){ for(int i=1; i<=N; i++) if(!Match[i]) R += Augment(i); }

```

3.28 $O(V^3)$ Weighted General Matching

```

namespace weighted_blossom_tree{ // n 400 w 1e8 700ms, n 500 w 1e6 300ms
#define d(x) (lab[x.u]+lab[x.v]-e[x.u][x.v].w*2)
const int N=403*2; using ll = long long; using T = int; // sum of weight, single weight
const T inf=numeric_limits<T>::max()-1;
struct Q{ int u, v; T w; } e[N][N]; vector<int> p[N];
int n, m=0, id, h, t, lk[N], sl[N], st[N], f[N], b[N][N], s[N], ed[N], q[N]; T lab[N];
void upd(int u, int v){ if (!sl[v] || d(e[u][v]) < d(e[sl[v]][v])) sl[v] = u; }
void ss(int v){
    sl[v]=0; for(int u=1; u<=n; u++) if(e[u][v].w > 0 && st[u] != v && !s[st[u]]) upd(u, v);
}
void ins(int u){ if(u <= n) q[++t] = u; else for(int v : p[u]) ins(v); }
void mdf(int u, int w){ st[u]=w; if(u > n) for(int v : p[u]) mdf(v, w); }
}

```

```

int gr(int u, int v){
    if ((v=find(p[u].begin(), p[u].end(), v) - p[u].begin()) & 1){
        reverse(p[u].begin()+1, p[u].end()); return (int)p[u].size() - v;
    }
    return v; }
void stm(int u, int v){
    lk[u] = e[u][v].v;
    if(u <= n) return; Q w = e[u][v];
    int x = b[u][w.u], y = gr(u, x);
    for(int i=0; i<y; i++) stm(p[u][i], p[u][i+1]);
    stm(x, v); rotate(p[u].begin(), p[u].begin()+y, p[u].end()); }
void aug(int u, int v){
    int w = st[lk[u]]; stm(u, v); if (!w) return;
    stm(w, st[f[w]]); aug(st[f[w]], w); }
int lca(int u, int v){
    for(++id; u<v; swap(u, v)){
        if(!u) continue; if(ed[u] == id) return u;
        ed[u] = id; if(u = st[lk[u]]) u = st[f[u]]; // not ==
    }
    return 0; }
void add(int u, int a, int v){
    int x = n+1; while(x <= m && st[x]) x++;
    if(x > m) m++;
    lab[x] = s[x] = st[x] = 0; lk[x] = lk[a];
    p[x].clear(); p[x].push_back(a);
    for(int i=u, j; i!=a; i=st[f[j]]) p[x].push_back(i), p[x].push_back(j=st[lk[i]]), ins(j);
    reverse(p[x].begin()+1, p[x].end());
    for(int i=v, j; i!=a; i=st[f[j]]) p[x].push_back(i), p[x].push_back(j=st[lk[i]]), ins(j);
    mdf(x, x); for(int i=1; i<=m; i++) e[x][i].w=e[i][x].w=0;
    memset(b[x]+1, 0, n*sizeof b[0][0]);
    for (int u : p[x]){
        for(v=1; v<=m; v++) if(!e[x][v].w || d(e[u][v]) < d(e[x][v])) e[x][v] = e[u][v], e[v][x] = e[v][u];
        for(v=1; v<=n; v++) if(b[u][v]) b[x][v] = u;
    }
    ss(x); }
void ex(int u){ // s[u] == 1
    for(int x : p[u]) mdf(x, x);
    int a = b[u][e[u][f[u]].u], r = gr(u, a);
    for(int i=0; i<r; i+=2){
        int x = p[u][i], y = p[u][i+1];
        f[x] = e[y][x].u; s[x] = 1; s[y] = 0; sl[x] = 0; ss(y);;
        ins(y); }
    s[a] = 1; f[a] = f[u];
    for(int i=r+1; i<p[u].size(); i++) s[p[u][i]]=-1, ss(p[u][i]);
    st[u] = 0; }
bool on(const Q &e){
    int u=st[e.u], v=st[e.v], a;
    if(s[v] == -1) f[v] = e.u, s[v] = 1, a = st[lk[v]], sl[v] = sl[a] = s[a] = 0, ins(a);
    else if(!s[v]){
        a = lca(u, v); if(!a) return aug(u, v), aug(v, u), true;
        else add(u, a, v);
    }
    return false; }
bool bfs(){

```

```

memset(s+1, -1, m*sizeof s[0]); memset(sl+1, 0, m*sizeof
sl[0]);
h = 1; t = 0; for(int i=1; i<=m; i++) if(st[i] == i &&
!lk[i]) f[i] = s[i] = 0, ins(i);
if(h > t) return 0;
while (true){
    while (h <= t){
        int u = q[h++];
        if (s[st[u]] != 1) for (int v=1; v<=n; v++) if
(e[u][v].w > 0 && st[u] != st[v])
            if(d(e[u][v])) upd(u, st[v]); else if(on(e[u][v]))
                return true;
    }
    T x = inf;
    for(int i=n+1; i<=m; i++) if(st[i] == i && s[i] == 1) x =
min(x, lab[i]>>1);
    for(int i=1; i<=m; i++) if(st[i] == i && sl[i] && s[i] !=
1) x = min(x, d(e[sl[i]][i])>>s[i]+1);
    for(int i=1; i<=n; i++) if(~s[st[i]]) if((lab[i] +=
(s[st[i]]*2-1)*x) <= 0) return false;
    for(int i=n+1; i<=m; i++) if(st[i] == i && ~s[st[i]])
lab[i] += (2-s[st[i]]*4)*x;
    h = 1; t = 0;
    for(int i=1; i<=m; i++) if(st[i] == i && sl[i] &&
st[sl[i]] != i && !d(e[sl[i]][i]) && on(e[sl[i]][i]))
        return true;
    for(int i=n+1; i<=m; i++) if(st[i] == i && s[i] == 1 &&
!lab[i]) ex(i);
}
return 0; }
template<typename TT> pair<int,ll> run(int N, const
vector<tuple<int,int,TT>> &edges){ // 1-based
memset(ed+1, 0, m*sizeof ed[0]); memset(lk+1, 0, m*sizeof
lk[0]);
n = m = N; id = 0; iota(st+1, st+n+1, 1); T wm = 0; ll r =
0;
for(int i=1; i<=n; i++) for(int j=1; j<=n; j++) e[i][j] =
{i,j,0};
for(auto [u,v,w] : edges) wm = max(wm,
e[v][u].w=e[u][v].w=max(e[u][v].w,(T)w));
for(int i=1; i<=n; i++) p[i].clear();
for(int i=1; i<=n; i++) for (int j=1; j<=n; j++) b[i][j] =
i*(i==j);
fill_n(lab+1, n, wm); int match = 0; while(bfs()) match++;
for(int i=1; i<=n; i++) if(lk[i]) r += e[i][lk[i]].w;
return {match, r/2};
}
#undef d
} using weighted_blossom_tree::run, weighted_blossom_tree::lk;

```

4 Math

4.1 Extend GCD, CRT, Combination

```

// ll gcd(ll a, ll b), ll lcm(ll a, ll b), ll mod(ll a, ll b)
tuple<ll,ll,ll> ext_gcd(ll a, ll b){ // return [g,x,y] s.t.
ax+by=gcd(a,b)=g
    if(b == 0) return {a, 1, 0}; auto [g,x,y] = ext_gcd(b, a % b);
    return {g, y, x - a/b * y};
}
ll inv(ll a, ll m){ //return x when ax mod m = 1, fail -> -1
    auto [g,x,y] = ext_gcd(a, m); return g == 1 ? mod(x, m) : -1;
}

```

```

void DivList(ll n){ // [n/1, n/2, ..., n/n], size <= 2 sqrt n
    for(ll i=1, j=1; i<=n; i=j+1) cout << i << " " << (j=n/(n/i))
    << " " << n/i << "\n"; }
void Div2List(ll n){ // n/(i^2), n^{3/4}
    for(ll i=1, j=1; i*i<=n; i=j+1){
        j = (ll)floor1(sqrt1(n/(n/(i*i))));
        cout << i << " " << j << " " << n/(i*i) << " ";
    } } //square free: sum_{i=1..sqrt n} mu(i)floor(n/(i^2))
pair<ll,ll> crt(ll a1, ll m1, ll a2, ll m2){
    ll g = gcd(m1, m2), m = m1 / g * m2;
    if((a2 - a1) % g) return {-1, -1};
    ll md = m2/g, s = mod((a2-a1)/g, m2/g);
    ll t = mod(get<1>(ext_gcd(m1/g,md, m2/g)), md);
    return { a1 + s * t % md * m1, m };
}
pair<ll,ll> crt(const vector<ll> &a, const vector<ll> &m){
    ll ra = a[0], rm = m[0];
    for(int i=1; i<m.size(); i++){
        auto [aa,mm] = crt(ra, rm, a[i], m[i]);
        if(mm == -1) return {-1, -1}; else tie(ra,rm) = tie(aa,mm);
    } return {ra, rm};
}
struct Lucas{ // init : O(P), query : O(log P)
    const size_t P;
    vector<ll> fac, inv;
    ll Pow(ll a, ll b){ /* return a^b mod P */ }
    Lucas(size_t P):P(P),fac(P),inv(P){ // init fac, facinv */ }
    ll small(ll n, ll r) const { /* n! / r! / (n-r)! */ }
    ll calc(ll n, ll r) const { if(n<r || n<0 || r<0) return 0;
        if(!n || !r || n == r) return 1;
        else return small(n/P, r/P) * calc(n/P, r/P) % P; }
};
template<ll p, ll e> struct CombinationPrimePower{
    vector<ll> val; ll m; // init : O(p^e), query : O(log p)
    CombinationPrimePower(){
        m=1; for(int i=0; i<e; i++) m *= p; val.resize(m); val[0]=1;
        for(int i=1; i<m; i++)val[i] = val[i-1] * (i%p ? i : 1) % m;
    }
    pair<ll,ll> factorial(int n){ if(n < p) return {0, val[n]};
        int k = n / p; auto v = factorial(k);
        int cnt = v.first + k, kp = n / m, rp = n % m;
        ll ret = v.second * Pow(val[m-1], kp/2, m) % m * val[rp] %
        m;
        return {cnt, ret}; }
    ll calc(int n, int r){ if(n < 0 || r < 0 || n < r) return 0;
        auto v1=factorial(n), v2=factorial(r), v3=factorial(n-r);
        ll cnt = v1.first - v2.first - v3.first;
        ll ret = v1.second * inv(v2.second, m) % m * inv(v3.second,
        m) % m;
        if(cnt >= e) return 0;
        for(int i=1; i<=cnt; i++) ret = ret * p % m;
        return ret; }
};

```

4.2 Partition Number

```

for(int j=1; j*(3*j-1)/2<=i; j++) P[i] +=
(j%2?-1)*P[i-j*(3*j-1)/2], P[i] %= MOD;
for(int j=1; j*(3*j+1)/2<=i; j++) P[i] +=
(j%2?-1)*P[i-j*(3*j+1)/2], P[i] %= MOD;
vector<ModInt> res(sz+1); res[0] = 1; int sq=sqrt(sz);
vector<vector<ModInt>> p(2, vector<ModInt>(sz+1)), d=p;
for(int k=1; k<sq; k++){ p[0][0] = k == 1; // calc p[k][n]

```

```

for(int n=1; n<=sz; n++){
    p[k&1][n] = p[k&1][n-1] + (n-k>0 ? p[k&1][n-k] : 0);
    res[n] += p[k&1][n]; }
for(int a=sq; a>0; a--) for(int b=sq; b<=sz; b++){
    d[a&1][b] = d[a&1][b-sq] + p[sq&1][b-1] + (b-a-1>0 ?
    d[a&1][b-a-1] : 0);
    if(a == 0) res[b] += d[a&1][b];
}

```

4.3 Diophantine

```

// solutions to ax + by = c where x in [xlow, xhigh] and y in
[ylow, yhigh]
// cnt, leftsol, rightsol, gcd of a and b
template<class T> array<T, 6> solve_linear_diophantine(T a, T b,
T c, T xlow, T xhigh, T ylow, T yhigh){
    T g, x, y = euclid(a >= 0 ? a : -a, b >= 0 ? b : -b, x, y);
    array<T, 6> no_sol{0, 0, 0, 0, 0, g};
    if(c % g) return no_sol; x *= c / g, y *= c / g;
    if(a < 0) x = -x; if(b < 0) y = -y;
    a /= g, b /= g, c /= g;
    auto shift = [&](T &x, T &y, T a, T b, T cnt){ x += cnt * b,
    y -= cnt * a; };
    int sign_a = a > 0 ? 1 : -1, sign_b = b > 0 ? 1 : -1;
    shift(x, y, a, b, (xlow - x) / b);
    if(x < xlow) shift(x, y, a, b, sign_b);
    if(x > xhigh) return no_sol;
    T lx1 = x; shift(x, y, a, b, (xhigh - x) / b);
    if(x > xhigh) shift(x, y, a, b, -sign_b);
    T rx1 = x; shift(x, y, a, b, -(y - ylow) / a);
    if(y < ylow) shift(x, y, a, b, -sign_a);
    if(y > yhigh) return no_sol;
    T lx2 = x; shift(x, y, a, b, -(yhigh - y) / a);
    if(y > yhigh) shift(x, y, a, b, sign_a);
    T rx2 = x; if(lx2 > rx2) swap(lx2, rx2);
    T lx = max(lx1, lx2), rx = min(rx1, rx2);
    if(lx > rx) return no_sol;
    return {(rx - lx) / (b >= 0 ? b : -b) + 1, lx, (c - lx * a)
    / b, rx, (c - rx * a) / b, g};
}

```

4.4 FloorSum

```

// sum of floor((A*i+B)/M) over 0 <= i < N in O(log(N+M+A+B))
// Also, sum of i * floor((A*i+B)/M) and floor((A*i+B)/M)^2
template<class T, class U> // T must be able to hold arg^2
array<U, 3> weighted_floor_sum(T n, T m, T a, T b){
    array<U, 3> res{}; auto [qa,ra]=div(a,m); auto [qb,rb]=div(b,m);
    if(T n2 = (ra * n + rb) / m){
        auto prv=weighted_floor_sum<T,U>(n2, ra, m, m-rb-1);
        res[0] += U(n-1)*n2 - prv[0];
        res[1] += (U(n-1)*n*n2 - prv[0] - prv[2]) / 2;
        res[2] += U(n-1)*(n2-1)*n2 - 2*prv[1] + res[0];
    }
    res[2] += U(n-1)*n*(2*n-1)/6 * qa*qa + U(n)*qb*qb;
    res[2] += U(n-1)*n * qa*qb + 2*res[0]*qb + 2*res[1]*qa;
    res[0] += U(n-1)*n/2 * qa + U(n)*qb;
    res[1] += U(n-1)*n*(2*n-1)/6 * qa + U(n-1)*n/2 * qb;
    return res;
}
ll modsum(ull to, ll c, ll k, ll m){

```

```

c = (c % m + m) % m; k = (k % m + m) % m;
return to*c + k*sumsq(to) - m*divsum(to, c, k, m);
} // sum (ki+c)%m 0<=i<to, O(log m) large constant

```

4.5 XOR Basis (XOR Maximization)

```

vector<ll> basis; // ascending
for(int i=0; i<n; i++){ ll x; cin >> x;
for(int j=(int)basis.size()-1; j>=0; j--) x=min(x,basis[j]^x);
if(x)basis.insert(lower_bound(basis.begin(),basis.end(), x),x);
} //xor maximization, reverse -> for(auto i:basis)r=max(r,r^i);

```

4.6 Stern Brocot Tree

```

pair<ll,ll> Solve(ld l, ld r){ //find l<p/q<r -> min q -> min p
auto g=[](ll v,pair<ll,ll>a,pair<ll,ll>b)->pair<ll,ll>{
return { v * a.first + b.first, v * a.second + b.second };
};
auto f = [g](ll v, pair<ll,ll> a, pair<ll,ll> b) -> ld {
auto [p,q] = g(v, a, b); return ld(p) / q; };
pair<ll,ll> s(0, 1), e(1, 0);
while(true){
pair<ll,ll> m(s.first+e.first, s.second+e.second);
ld v = 1.L * m.first / m.second;
if(v >= r){
ll ks = 1, ke = 1; while(f(ke, s, e) >= r) ke *= 2;
while(ks <= ke){
ll km = (ks + ke) / 2;
if(f(km, s, e) >= r) ks = km + 1; else ke = km - 1;
} e = g(ke, s, e);
} else if(v <= l){
ll ks = 1, ke = 1; while(f(ke, e, s) <= l) ke *= 2;
while(ks <= ke){
ll km = (ks + ke) / 2;
if(f(km, e, s) <= l) ks = km + 1; else ke = km - 1;
} s = g(ke, e, s);
}
else return m;
}
}
struct Frac { ll p, q; }; //find smallest 0 <= p/q <= 1 (p,q<=N)
template<class F> Frac fracBS(F f, ll N) { // s.t. f(p/q) true
bool dir = 1, A = 1, B = 1; // O(log N)
Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
if(f(lo)) return lo; assert(f(hi));
while(A != 0 || B != 0){
ll adv = 0, step = 1; // move hi if dir, else lo
for(int si=0; step; (step*=2)>=si){ adv += step;
Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
if(abs(mid.p)>N || mid.q>N || dir != f(mid))
adv -= step, si = 2;
}
hi.p += lo.p * adv; hi.q += lo.q * adv;
dir = !dir; swap(lo, hi); A = B; B = adv != 0;
}
return dir ? hi : lo;
}

```

4.7 $O(N^3 \log 1/\epsilon)$ Polynomial Equation

```

vector<double> poly_root(vector<double> p, double xmin, double
xmax){
if(p.size() == 2){ return {-p[0] / p[1]}; }

```

```

vector<double> ret, der(p.size()-1);
for(int i=0; i<der.size(); i++) der[i] = p[i+1] * (i + 1);
auto dr = poly_root(der, xmin, xmax);
dr.push_back(xmin-1); dr.push_back(xmax+1);
sort(dr.begin(), dr.end());
for(int i=0; i+1<dr.size(); i++){
double l = dr[i], h = dr[i+1]; bool sign = calc(p, l) > 0;
if (sign ^ (calc(p, h) > 0)){
for(int it=0; it<60; it++){ // while(h-l > 1e-8)
double m = (l + h) / 2, f = calc(p, m);
if ((f <= 0) ^ sign) l = m; else h = m;
}
ret.push_back((l + h) / 2);
}
}
return ret;
}

```

4.8 Gauss Jordan Elimination

```

template<typename T> // return {rref, rank, det, inv}
tuple<vector<vector<T>>, int, T, vector<vector<T>>>
Gauss(vector<vector<T>> a, bool square=true){ // n500 -400ms
int n = a.size(), m = a[0].size(), rank = 0; //bitset 4096-700
vector<vector<T>> out(n, vector<T>(m, 0)); T det = T(1);
for(int i=0; i<n; i++) if(square) out[i][i] = T(1);
for(int i=0; i<m; i++){
if(rank == n) break;
if(IsZero(a[rank][i])){
T mx = T(0); int idx = -1; // fucking precision error
for(int j=rank+1; j<n; j++) if(mx < abs(a[j][i])) mx =
abs(a[j][i]), idx = j;
if(idx == -1 || IsZero(a[idx][i])){ det = 0; continue; }
for(int k=0; k<m; k++){
a[rank][k] = Add(a[rank][k], a[idx][k]);
if(square)out[rank][k]=Add(out[rank][k],out[idx][k]);
}
}
det = Mul(det, a[rank][i]);
T coeff = Div(T(1), a[rank][i]);
for(int j=0; j<m; j++) a[rank][j] = Mul(a[rank][j], coeff);
for(int j=0; j<m; j++) if(square) out[rank][j] =
Mul(out[rank][j], coeff);
for(int j=0; j<n; j++){
if(rank == j) continue;
T t = a[j][i]; // Warning: [j][k], [rank][k]
for(int k=0; k<m; k++) a[j][k] = Sub(a[j][k],
Mul(a[rank][k], t));
for(int k=0; k<m; k++) if(square) out[j][k] =
Sub(out[j][k], Mul(out[rank][k], t));
}
rank++; // linear system: warning len(A) != len(A[0])
} return {a, rank, det, out}; // linear system: get RREF(A|b)
} // 0 0 ... 0 b[i]: inconsistent, rank < len(A[0]): multiple
// get det(A) mod M, M can be composite number
// remove mod M -> get pure det(A) in integer
ll Det(vector<vector<ll>> a){ //destroy matrix, n500 -400ms
int n = a.size(); ll ans = 1;
for(int i=0; i<n; i++){
for(int j=i+1; j<n; j++){
while(a[j][i] != 0){ // gcd step
ll t = a[i][i] / a[j][i];
if(t)for(int k=i;k<n;k++) a[i][k]=(a[i][k]-a[j][k]*t)%M;

```

```

swap(a[i], a[j]); ans *= -1;
}
}
ans = ans * a[i][i] % M; if(!ans) return 0;
} return (ans + M) % M;
}

```

4.9 Berlekamp + Kitamasu

```

const int mod = 1e9+7; ll pw(ll a, ll b){/*a^b mod M*/}
vector<int> berlekamp_massey(vector<int> x){
int n = x.size(), L=0, m=0; ll b=1; if(!n) return {};
vector<int> C(n), B(n), T; C[0]=B[0]=1;
for(int i=0; ++m && i<n; i++){ ll d = x[i] % mod;
for(int j=1; j<=L; j++) d = (d + 1LL * C[j] * x[i-j]) % mod;
if(!d) continue; T=C; ll c = d * pw(b, mod-2) % mod;
for(int j=m; j<n; j++) C[j] = (C[j] - c * B[j-m]) % mod;
if(2 * L <= i) L = i-L+1, B = T, b = d, m = 0;
}
C.resize(L+1); C.erase(C.begin());
for(auto &i : C) i = (mod - i) % mod; return C;
} // O(NK + N log mod)
int get_nth(vector<int> rec, vector<int> dp, ll n){
int m = rec.size(); vector<int> s(m), t(m); ll ret=0;
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] = (t[j+k] + 1LL * v[j] * w[k]) % mod;
}
for(int j=2*m-1; j>=m; j--) for(int k=1; k<=m; k++){
t[j-k] = (t[j-k] + 1LL * t[j] * rec[k-1]) % mod;
}
t.resize(m); return t;
};
for(; n>=1, t=mul(t,t)) if(n & 1) s=mul(s,t);
for(int i=0; i<m; i++) ret += 1LL * s[i] * dp[i] % mod;
return ret % mod;
} // O(N2 log X)
int guess_nth_term(vector<int> x, ll n){
if(n < x.size()) return x[n];
vector<int> v = berlekamp_massey(x);
return v.empty() ? 0 : 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, gobs; mt19937 rng(0x14004);
auto gen = [&rng](int lb, int ub){ return
uniform_int_distribution<int>(lb, ub)(rng); };
for(int i=0; i<n; i++) rnd1.push_back(gen(1, mod-1)),
rnd2.push_back(gen(1, mod-1));
for(int i=0; i<2*n+2; i++){ int tmp = 0;
for(int j=0; j<n; j++) tmp = (tmp + 1LL * rnd2[j] * rnd1[j])
% mod;
gobs.push_back(tmp); vector<int> nxt(n);
for(auto &j : M) nxt[j.x] = (nxt[j.x] + 1LL * j.v *
rnd1[j.y]) % mod;

```



```

    rnd1 = nxt;
} auto v = berlekamp_massey(gobs);
return vector<int>(v.rbegin(), v.rend());
}

ll det(int n, vector<elem> M){
    vector<int> rnd; mt19937 rng(0x14004);
    auto gen = [&rng](int lb, int ub){ return
        uniform_int_distribution<int>(lb, ub)(rng); };
    for(int i=0; i<n; i++) rnd.push_back(gen(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 * pw(i, mod-2) % mod;
    return sol;
}

```

4.10 Linear Sieve

```

// sp : 최소 소인수, 소수라면 0
// tau : 약수 개수, sigma : 약수 합
// phi : n 이하 자연수 중 n과 서로소인 개수
// mu : non square free이면 0, 그렇지 않다면 (-1)^(소인수 종류)
// e[i] : 소인수분해에서 i의 지수
vector<int> prime;
int sp[sz], e[sz], phi[sz], mu[sz], tau[sz], sigma[sz];
phi[1] = mu[1] = tau[1] = sigma[1] = 1;
for(int i=2; i<=n; i++){
    if(!sp[i]){
        prime.push_back(i);
        e[i] = 1; phi[i] = i-1; mu[i] = -1; tau[i] = 2; sigma[i] =
        i+1;
    }
    for(auto j : prime){
        if(i*j >= sz) break;
        sp[i*j] = j;
        if(i % j == 0){
            e[i*j] = e[i]+1; phi[i*j] = phi[i]*j; mu[i*j] = 0;
            tau[i*j] = tau[i]/e[i*j]*(e[i*j]+1);
            sigma[i*j] = sigma[i]*(j-1)/(pw(j, e[i*j])-1)*(pw(j,
            e[i*j]+1)-1)/(j-1); //overflow
            break;
        }
        e[i*j] = 1; phi[i*j] = phi[i] * phi[j]; mu[i*j] = mu[i] *
        mu[j];
        tau[i*j] = tau[i] * tau[j]; sigma[i*j] = sigma[i] *
        sigma[j];
    }
}

```

4.11 Xudyh Sieve

```

/* e(x) = [x==1], 1(x) = 1, id_k(x) = x^k
mu: mobius function, id(x) = x
phi: euler totient function
sigma_k: sum of k-th power of divisors
sigma = sigma_1, d = tau = sigma_0
sigma_k = id_k * 1 | sigma = id * 1
id_k = sigma_k * mu | id = sigma * mu
e = 1 * mu | d = 1 * 1 | 1 = d * mu
phi * 1 = id | phi = id * mu | sigma = phi * d
g = f * 1 iff f = g * mu */
template<class T, class F1, class F2, class F3>
struct xudyh_sieve{

```

```

    T th; // threshold, 2(single query) ~ 5 * MAXN^2/3
    F1 pf; F2 pg; F3 pfg;
    // prefix sum of f(up to th), g(easy to calc), f*g(easy to
    calc)
    unordered_map<T, T> mp; // f * g means dirichlet conv.
    xudyh_sieve(T th, F1 pf, F2 pg, F3
    pfg):th(th), pf(pf), pg(pg), pfg(pfg){}
    // Calculate the preix sum of a multiplicative f up to n
    T query(T n){ // 0(n^2/3)
        if(n <= th) return pf(n); if(mp.count(n)) return mp[n];
        T res = pfg(n);
        for(T low = 2, high = 2; low <= n; low = high + 1){
            high = n / (n / low);
            res -= (pg(high) - pg(low - 1)) * query(n / low); // MOD
        }
        return mp[n] = res / pg(1); //Pow(pg(1), MOD-2)?
    }
};

```

4.12 Miller Rabin + Pollard Rho

```

constexpr int SZ = 10'000'000; bool PrimeCheck[SZ+1];
vector<int> Primes;
void Sieve(){ memset(PrimeCheck, true, sizeof PrimeCheck); /*
Sieve */ }
ull MulMod(ull a, ull b, ull c){ return ((__uint128_t)a * b % c;
}
// 32bit : 2, 7, 61
// 64bit : 2, 325, 9375, 28178, 450775, 9780504, 1795265022
bool MillerRabin(ull n, ull a){
    if(a % n == 0) return true;
    int cnt = __builtin_ctzll(n - 1);
    ull p = PowMod(a, n >> cnt, n);
    if(p == 1 || p == n - 1) return true;
    while(cnt--){ if((p=MulMod(p,p,n)) == n - 1) return true;
        return false;
    }
}
bool IsPrime(ll n){
    if(n <= SZ) return PrimeCheck[n];
    if(n <= 2) return n == 2;
    if(n % 2 == 0 || n % 3 == 0 || n % 5 == 0 || n % 7 == 0 || n %
    11 == 0) return false;
    for(int p : {2, 325, 9375, 28178, 450775, 9780504,
    1795265022}) if(!MillerRabin(n, p)) return false;
    return true;
}
ll Rho(ll n){
    while(true){
        ll x = rand() % (n - 2) + 2, y = x, c = rand() % (n - 1) +
        1;
        while(true){
            x = (MulMod(x,x,n)+c) % n; y = (MulMod(y,y,n)+c) % n; y =
            (MulMod(y,y,n)+c) % n;
            ll d = __gcd(abs(x - y), n); if(d == 1) continue;
            if(IsPrime(d)) return d; else{ n = d; break; }
        }
    }
}
vector<pair<ll,ll>> Factorize(ll n){
    vector<pair<ll,ll>> v;
    int two = __builtin_ctzll(n);
    if(two > 0) v.emplace_back(2, two), n >>= two;
    if(n == 1) return v;

```

```

    while(!IsPrime(n)){
        ll d = Rho(n), cnt = 0; while(n % d == 0) cnt++, n /= d;
        v.emplace_back(d, cnt); if(n == 1) break;
    }
    if(n != 1) v.emplace_back(n, 1); return v;
}

```

4.13 Primitive Root, Discrete Log/Sqrt

```

ll PrimitiveRoot(ll p){ // order p-1
    vector<pair<ll,ll>> v = Factorize(p-1);
    for(ll r=1; ; r++){
        bool flag = true; // Warning: 64bit Pow
        for(auto [d,e] : v) if(PowMod(r, (p-1)/d, p) == 1){ flag =
        false; break; }
        if(flag) return r;
    }
}
// Given A, B, P, solve A^x == B mod P, return smallest value
ll DiscreteLog(ll A, ll B, ll P){ // O(sqrt P) with hash set
    __gnu_pbds::gp_hash_table<ll, __gnu_pbds::null_type> st;
    ll t = ceil(sqrt(P)), k = 1; // use binary search?
    for(int i=0; i<t; i++) st.insert(k, k = k * A % P);
    ll inv = Pow(k, P-2, P);
    for(int i=0, s=1; i<t; i++, s=s*inv%P){
        ll x = B * s % P;
        if(st.find(x) == st.end()) continue;
        for(int j=0, f=1; j<t; j++, f=f*A%P){
            if(f == x) return i * t + j;
        }
    }
    return -1;
}
// Given A, P, solve X^2 == A mod P, return arbitrary
ll DiscreteSqrt(ll A, ll P){ // O(log^2 P), O(log P) in random data
    if(A == 0) return 0;
    if(Pow(A, (P-1)/2, P) != 1) return -1;
    if(P % 4 == 3) return Pow(A, (P+1)/4, P);
    ll s = P - 1, n = 2, r = 0, m;
    while(~s & 1) r++, s >>= 1;
    while(Pow(n, (P-1)/2, P) != P-1) n++;
    ll x = Pow(A, (s+1)/2, P), b = Pow(A, s, P), g = Pow(n, s, P);
    for(; r=m){
        ll t = b; for(m=0; m<r && t!=1; m++) t = t * t % P;
        if(!m) return x;
        ll gs = Pow(g, 1LL << (r-m-1), P);
        g = gs * gs % P; x = x * gs % P; b = b * g % P;
    }
}

```

4.14 Power Tower

```

bool PowOverflow(ll a, ll b, ll c){
    __int128_t res = 1;
    bool flag = false;
    for(; b >= 1, a = a * a){
        if(a >= c) flag = true, a %= c;
        if(b & 1){
            res *= a; if(flag || res >= c) return true;
        }
    }
    return false;
}

```

```

}
11 Recursion(int idx, ll mod, const vector<ll> &vec){
    if(mod == 1) return 1;
    if(idx + 1 == vec.size()) return vec[idx];
    ll nxt = Recursion(idx+1, phi[mod], vec);
    if(PowOverflow(vec[idx], nxt, mod)) return Pow(vec[idx], nxt,
    mod) + mod; else return Pow(vec[idx], nxt, mod);
}
11 PowerTower(const vector<ll> &vec, ll mod){ //
vec[0]^(vec[1]^(vec[2]^(...)))
    if(vec.size() == 1) return vec[0] % mod;
    else return Pow(vec[0], Recursion(1, phi[mod], vec), mod);
}

```

4.15 De Bruijn Sequence

```

// Create cyclic string of length k^n that contains every length
n string as substring. alphabet = [0, k - 1]
int res[10000000], aux[10000000]; // >= k^n
int de_bruijn(int k, int n) { // 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(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;
}

```

4.16 Simplex / LP Duality

```

// Solves the canonical form: maximize c^T x, subject to ax <= b
and x >= 0.
template<class T> // T must be of floating type
struct linear_programming_solver_simplex{
    int m, n; vector<int> nn, bb; vector<vector<T>> mat;
    static constexpr T eps = 1e-8, inf = 1/0.0;
    linear_programming_solver_simplex(const vector<vector<T>> &a,
    const vector<T> &b, const vector<T> &c) : m(b.size()),
    n(c.size()), nn(n+1), bb(m), mat(m+2, vector<T>(n+2)){
        for(int i=0; i<m; i++) for(int j=0; j<n; j++) mat[i][j] =
        a[i][j];
        for(int i=0; i<m; i++) bb[i] = b[i], mat[i][n] = -1,
        mat[i][n + 1] = b[i];
        for(int j=0; j<n; j++) nn[j] = j, mat[m][j] = -c[j];
        nn[n] = -1; mat[m + 1][n] = 1;
    }
    void pivot(int r, int s){
        T *a = mat[r].data(), inv = 1 / a[s];
        for(int i=0; i<m+2; i++) if(i != r && abs(mat[i][s]) > eps)
        {
            T *b = mat[i].data(), inv2 = b[s] * inv;
            for(int j=0; j<n+2; j++) b[j] -= a[j] * inv2;
            b[s] = a[s] * inv2;
        }
        for(int j=0; j<n+2; j++) if(j != s) mat[r][j] *= inv;
        for(int i=0; i<m+2; i++) if(i != r) mat[i][s] *= -inv;
        mat[r][s] = inv; swap(bb[r], nn[s]);
    }
}

```

```

}
bool simplex(int phase){
    for(auto x=m+phase-1; ; ){
        int s = -1, r = -1;
        for(auto j=0; j<n+1; j++) if(nn[j] != -phase) if(s == -1
        || pair(mat[x][j], nn[j]) < pair(mat[x][s], nn[s])) s = j;
        if(mat[x][s] >= -eps) return true;
        for(auto i=0; i<m; i++){
            if(mat[i][s] <= eps) continue;
            if(r == -1 || pair(mat[i][n + 1] / mat[i][s], bb[i]) <
            pair(mat[r][n + 1] / mat[r][s], bb[r])) r = i;
        }
        if(r == -1) return false;
        pivot(r, s);
    }
}
// Returns -inf if no solution, {inf, a vector satisfying the
constraints}
// if there are arbitrarily good solutions, or {maximum c^T
x, x} otherwise.
// O(n m (# of pivots)), O(2^n) in general.
pair<T, vector<T>> solve(){
    int r = 0;
    for(int i=1; i<m; i++) if(mat[i][n+1] < mat[r][n+1]) r = i;
    if(mat[r][n+1] < -eps){
        pivot(r, n);
        if(!simplex(2) || mat[m+1][n+1] < -eps) return {-inf, {}};
        for(int i=0; i<m; i++) if(bb[i] == -1){
            int s = 0;
            for(int j=1; j<n+1; j++) if(s == -1 || pair(mat[i][j],
            nn[j]) < pair(mat[i][s], nn[s])) s = j;
            pivot(i, s);
        }
    }
    bool ok = simplex(1);
    vector<T> x(n);
    for(int i=0; i<m; i++) if(bb[i] < n) x[bb[i]] = mat[i][n +
    1];
    return {ok ? mat[m][n + 1] : inf, x};
}
}

```

Simplex Example

Maximize $p = 6x + 14y + 13z$

Constraints

$$-0.5x + 2y + z \leq 24$$

$$-x + 2y + 4z \leq 60$$

Coding

$$-n = 2, m = 3, a = \begin{pmatrix} 0.5 & 2 & 1 \\ 1 & 2 & 4 \end{pmatrix}, b = \begin{pmatrix} 24 \\ 60 \end{pmatrix}, c = [6, 14, 13]$$

LP Duality & Example

tableau를 대각선으로 뒤집고 음수 부호를 붙인 답 = -(원 문제의 답)

$$- \text{Primal} : n = 2, m = 3, a = \begin{pmatrix} 0.5 & 2 & 1 \\ 1 & 2 & 4 \end{pmatrix}, b = \begin{pmatrix} 24 \\ 60 \end{pmatrix}, c = [6, 14, 13]$$

$$- \text{Dual} : n = 3, m = 2, a = \begin{pmatrix} -0.5 & -1 \\ -2 & -2 \\ -1 & -4 \end{pmatrix}, b = \begin{pmatrix} -6 \\ -14 \\ -13 \end{pmatrix}, c = [-24, -60]$$

공식

$$- \text{Primal} : \max_x c^T x, \text{Constraints } Ax \leq b, x \geq 0$$

$$- \text{Dual} : \min_y b^T y, \text{Constraints } A^T y \geq c, y \geq 0$$

4.17 Polynomial & Convolution

```

// 998,244,353 = 119 23, w 3 | 2,281,701,377 = 17 27, w 3
// 167,772,161 = 10 25, w 3 | 2,483,027,969 = 37 26, w 3
// 469,762,049 = 26 26, w 3 | 2,013,265,921 = 15 27, w 31
using real_t = double; using cpx = complex<real_t>;
void FFT(vector<cpx> &a, bool inv_fft=false){
    int N = a.size(); vector<cpx> root(N/2); //root[0]=1
    for(int i=1, j=0; i<N; i++){ int bit = N / 2;
        while(j >= bit) j -= bit, bit >= 1;
        if(i < (j += bit)) swap(a[i], a[j]);
    } long double ang = 2 * acos(-1) / N * (inv_fft ? -1 : 1);
    for(int i=0; i<N/2; i++) root[i] = cpx(cos(ang*i), sin(ang*i));
    /* NTT : ang = pow(w, (mod-1)/n) % mod, inv_fft -> ang^(-1),
    root[i] = root[i-1] * ang
    XOR Convolution: roots[*]=1, a[j+k] = u+v, a[j+k+i/2] = u-v
    OR Convolution: roots[*]=1, a[j+k+i/2] += inv_fft ? -u : u;
    AND Convolution: roots[*]=1, a[j+k] += inv_fft ? -v : v; */
    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++){
            cpx u = a[j+k], v = a[j+k+i/2] * root[step * k];
            a[j+k] = u+v; a[j+k+i/2] = u-v;
        } // inv_fft: skip for AND/OR convolution.
        if(inv_fft) for(int i=0; i<N; i++) a[i] /= N;
    }
}
vector<ll> Mul(const vector<ll> &a, const vector<ll> &b){
    vector<cpx> a(all(_a)), b(all(_b)); // (NTT) 2^19 700ms
    int N = 2; while(N < a.size() + b.size()) N <= 1;
    a.resize(N); b.resize(N); FFT(a); FFT(b);
    for(int i=0; i<N; i++) a[i] *= b[i]; // mod?
    vector<ll> ret(N); FFT(a, 1); // NTT : just return a
    for(int i=0; i<N; i++) ret[i] = llround(a[i].real());
    while(ret.size() > 1 && ret.back() == 0) ret.pop_back();
    return ret; }
vector<ll> MulMod(const vector<ll> &a, const vector<ll> &b,
const unsigned long long mod){ // (FFT) 2^19 1000ms
    int N = 2; while(N < a.size() + b.size()) N <= 1;
    vector<cpx> v1(N), v2(N), r1(N), r2(N);
    for(int i=0; i<a.size(); i++) v1[i] = cpx(a[i]>>15, a[i]&32767);
    for(int i=0; i<b.size(); i++) v2[i] = cpx(b[i]>>15, b[i]&32767);
    FFT(v1); FFT(v2);
    for(int i=0; i<N; i++){ int j = i ? N-i : i;
        cpx ans1 = (v1[i] + conj(v1[j])) * cpx(0.5, 0);
        cpx ans2 = (v1[i] - conj(v1[j])) * cpx(0, -0.5);
        cpx ans3 = (v2[i] + conj(v2[j])) * cpx(0.5, 0);
        cpx ans4 = (v2[i] - conj(v2[j])) * cpx(0, -0.5);
        r1[i] = (ans1 * ans3) + (ans1 * ans4) * cpx(0, 1);
        r2[i] = (ans2 * ans3) + (ans2 * ans4) * cpx(0, 1);
    } vector<ll> ret(N); FFT(r1, true); FFT(r2, true);
    for(int i=0; i<N; i++){
        ll av = llround(r1[i].real()) % mod;
        ll bv = (llround(r1[i].imag()) + llround(r2[i].real())) % mod;
        ll cv = llround(r2[i].imag()) % mod;
        ret[i] = (av << 30) + (bv << 15) + cv;
        ret[i] %= mod; ret[i] += mod; ret[i] %= mod;
    } while(ret.size() > 1 && ret.back() == 0) ret.pop_back();
    return ret; }
template<char op> vector<ll> FWHT_Conv(vector<ll> a, vector<ll> b){
    int n = max({(int)a.size(), (int)b.size()-1, 1}); //2^20 700ms
    if(__builtin_popcount(n) != 1) n = 1 << (__lg(n) + 1);
    a.resize(n); b.resize(n); FWHT<op>(a); FWHT<op>(b);
    for(int i=0; i<n; i++) a[i] = a[i] * b[i] % M;
}

```

```
FWHT<op>(a, true); return a;
} // subset: C[k] = sum_{i and j = 0, i or j = k} A[i] * B[j]
vector<ll> SubsetConvolution(vector<ll> p, vector<ll> q){ // Nlog2N
    int n = max((int)p.size(), (int)q.size()-1, 1); w = __lg(n);
    if(__builtin_popcount(n) != 1) n = 1 << (w + 1); // 2^20 4s
    p.resize(n); q.resize(n); vector<ll> res(n); // SOS DP: 2.5s
    vector<vector<ll>> a(w+1, vector<ll>(n)), b(a);
    for(int i=0; i<n; i++) a[__builtin_popcount(i)][i] = p[i];
    for(int i=0; i<n; i++) b[__builtin_popcount(i)][i] = q[i];
    for(int bit=0; bit<=w; bit++) FWHT<'|'>(a[bit]),
    FWHT<'|'>(b[bit]);
    for(int bit=0; bit<=w; bit++){
        vector<ll> c(n); // Warning: MOD
        for(int i=0; i<=bit; i++) for(int j=0; j<n; j++) c[j] +=
        a[i][j] * b[bit-i][j] % M;
        for(auto &i : c) i %= M;
        FWHT<'|'>(c, true);
        for(int i=0; i<n; i++) if(__builtin_popcount(i) == bit)
        res[i] = c[i];
    } return res; }

vector<ll> Trim(vector<ll> a, size_t sz){ a.resize(min(a.size(),
sz)); return a; }

vector<ll> Inv(const vector<ll> &a, size_t sz){ // 5e5 2s
    vector<ll> q(1, Pow(a[0], M-2)); // 1/a[0], a[0] != 0
    for(int i=1; i<sz; i++) { // - : polynomial minus
        auto p = vector<ll>{2} - Mul(q, Trim(a, i*2));
        q = Trim(Mul(p, q), i*2);
    } return Trim(q, sz); }

vector<ll> Div(const vector<ll> &a, const vector<ll> &b){
    if(a.size() < b.size()) return {}; // 5e5 4s
    size_t sz = a.size() - b.size() + 1; auto ra = a, rb = b;
    reverse(ra.begin(), ra.end()); ra = Trim(ra, sz);
    reverse(rb.begin(), rb.end()); rb = Inv(Trim(rb, sz), sz);
    auto res = Trim(Mul(ra, rb), sz); res.resize(sz);
    reverse(res.begin(), res.end());
    while(!res.empty() && !res.back()) res.pop_back();
    return res; }

vector<ll> Mod(const vector<ll> &a, const vector<ll> &b){ return
a - Mul(b, Div(a, b)); }

ll Evaluate(const vector<ll> &a, ll x){ ll res = 0;
    for(int i=(int)a.size()-1; i>=0; i--) res = (res*x+a[i]) % M;
    return res >= 0 ? res : res + M; }

vector<ll> Derivative(const vector<ll> &a){
    if(a.size() <= 1) return {}; vector<ll> res(a.size()-1);
    for(int i=0; i+1<a.size(); i++) res[i] = (i+1) * a[i+1] % M;
    return res; }

vector<ll> Integrate(const vector<ll> &a){
    int n = a.size(); vector<ll> res(n+1);
    for(int i=0; i<n; i++) res[i+1] = a[i] * Pow(i+1, M-2) % M;
    return res; }

vector<ll> MultipointEvaluation(vector<ll> a, vector<ll> x){
    if(x.empty()) return {}; int n = x.size(); // 2^17 7s
    vector<vector<ll>> up(n*2), dw(n*2);
    for(int i=0; i<n; i++) up[i+n] = {x[i]?M-x[i]:0, 1};
    for(int i=n-1; i; i--) up[i] = Mul(up[i*2], up[i*2+1]);
    dw[1] = Mod(a, up[1]);
    for(int i=2; i<n*2; i++) dw[i] = Mod(dw[i/2], up[i]);
    vector<ll> y(n); for(int i=0; i<n; i++) y[i] = dw[i+n][0];
    return y; }

vector<ll> Interpolation(vector<ll> x, vector<ll> y){ // 2^17 10s
    int n = x.size(); vector<vector<ll>> up(n*2), dw(n*2);
```

```
    for(int i=0; i<n; i++) up[i+n] = {x[i]?M-x[i]:0, 1};
    for(int i=n-1; i; i--) up[i] = Mul(up[i*2], up[i*2+1]);
    vector<ll> a = MultipointEvaluation(Derivative(up[1]), x);
    for(int i=0; i<n; i++) a[i] = y[i] * Pow(a[i], M-2) % M;
    for(int i=0; i<n; i++) dw[i+n] = {a[i]};
    for(int i=n-1; i; i--){
        auto l = Mul(dw[i*2], up[i*2+1]), r = Mul(dw[i*2+1], up[i*2]);
        dw[i].resize(l.size());
        for(int j=0; j<l.size(); j++) dw[i][j] = (l[j] + r[j]) % M;
    } return dw[1]; }

vector<ll> Log(const vector<ll> &a, size_t sz){ // 5e5 3.5s
    assert(a.size() > 0 && a[0] == 1); // int f'(x)/f(x), resize!
    return Trim(Integrate(Mul(Derivative(a), Inv(a, sz))), sz); }

vector<ll> Exp(const vector<ll> &a, size_t sz){ // 5e5 5s
    vector<ll> res = {1}; if(a.empty()) return {1};
    assert(a.size() > 0 && a[0] == 0);
    for(int i=1; i<sz; i++){
        auto t = Trim(a, i*2) - Log(res, i*2);
        if(++t[0] == M) t[0] = 0; // t[0] += 1, mod
        res = Trim(Mul(res, t), i*2);
    } return Trim(res, sz); } // need resize

vector<ll> Pow(const vector<ll> f, ll e, int sz){ // 5e5 8s
    if(e == 0){ vector<ll> res(sz); res[0] = 1; return res; }
    ll p = 0; while(p < f.size() && f[p] == 0 && p*e < sz) p++;
    if(p == f.size() || p*e >= sz) return vector<ll>(sz, 0);
    vector<ll> a(f.begin()+p, f.end()); ll k = a[0]; // not f[0]
    for(auto &i : a) i = mul(i, Pow(k, M-2));
    a = Log(a, sz); for(auto &i : a) i = mul(i, e%M);
    a = Exp(a, sz); for(auto &i : a) i = mul(i, Pow(k, e));
    vector<ll> res(p*e); res.insert(res.end(), a.begin(),
    a.end());
    res.resize(sz); return res; }

vector<ll> SqrtImpl(vector<ll> a){
    if(a.empty()) return {0}; int inv2=(M+1)/2;
    int z = DiscreteSqrt(a[0], M), n = a.size();
    if(z == -1) return {-1}; vector<ll> q(1, z);
    for(int m=1; m<n; m++){
        if(n < m*2) a.resize(m*2); q.resize(m*2);
        auto f2 = Mul(q, q); f2.resize(m*2);
        for(int i=0; i<m*2; i++) f2[i] = sub(f2[i], a[i]);
        f2 = Mul(f2, Inv(q, q.size())); f2.resize(m*2);
        for(int i=0; i<m*2; i++) q[i] = sub(q[i], mul(f2[i], inv2));
    } q.resize(n); return q; }

vector<ll> Sqrt(vector<ll> a){ // nlgn, fail -> -1, 5e5 5.5s
    int n = a.size(), m = 0; while(m < n && a[m] == 0) m++;
    if(m == n) return vector<ll>(n); if(m & 1) return {-1};
    auto s = SqrtImpl(vector<ll>(a.begin()+m, a.end()));
    if(s[0] == -1) return {-1}; vector<ll> res(n);
    for(int i=0; i<s.size(); i++) res[i+m/2] = s[i];
    return res; }

vector<ll> TaylorShift(vector<ll> a, ll c){ // f(x+c), 2^19 700ms
    int n = a.size(); // fac[i] = i!, ifc[i] = inv(i!)
    for(int i=0; i<n; i++) a[i] = mul(a[i], fac[i]);
    reverse(all(a)); vector<ll> b(n); ll w = 1;
    for(int i=0; i<n; i++) b[i] = mul(ifc[i], w), w = mul(w, c);
    a = Mul(a, b); a.resize(n); reverse(all(a));
    for(int i=0; i<n; i++) a[i] = mul(a[i], ifc[i]);
    return a; }

vector<ll> SamplingShift(vector<ll> a, ll c, int m){ // 2^19 ~2s
    // given f(0), f(1), ..., f(n-1), warning: fac size
    // return f(c), f(c+1), ..., f(c+m-1)
```

```
    int n = a.size(); vector<ll> b(ifc.begin(), ifc.begin()+n);
    for(int i=0; i<n; i++) a[i] = mul(a[i], ifc[i]);
    for(int i=1; i<n; i+=2) b[i] = sub(0, b[i]);
    a = Mul(a, b); a.resize(n); ll w = 1;
    for(int i=0; i<n; i++) a[i] = mul(a[i], fac[i]);
    reverse(all(a));
    for(int i=0; i<n; w=mul(w, sub(c, i+1))) b[i] = mul(ifc[i], w);
    a = Mul(a, b); a.resize(n); reverse(all(a)); //warning: N->M
    for(int i=0; i<n; i++) a[i] = mul(a[i], ifc[i]); a.resize(m);
    b = vector<ll>(ifc.begin(), ifc.begin()+m);
    a = Mul(a, b); a.resize(m);
    for(int i=0; i<m; i++) a[i] = mul(a[i], fac[i]);
    return a; }

vector<double> interpolate(vector<double> x, vector<double> y,
int n){ // n^2
    vector<double> res(n), temp(n);
    for(int k=0; k<n-1; k++) for(int i=k+1; i<n; i++) y[i] = (y[i]
- y[k]) / (x[i] - x[k]);
    double last = 0; temp[0] = 1;
    for(int k=0; k<n; k++){
        for(int i=0; i<n; i++) res[i] += y[k] * temp[i], swap(last,
temp[i]), temp[i] -= last * x[k];
    }
    return res; } //for numerical precision, x[k]=c*cos(k*pi/(n-1))

vector<ll> Interpolation_0_to_n(vector<ll> y){ // n^2
    int n = y.size();
    vector<ll> res(n), tmp(n), x; // x[i] = i / (i+1)
    for(int i=0; i<n; i++) x.push_back(Pow(i+1, M-2));
    for(int k=0; k+1<n; k++) for(int i=k+1; i<n; i++)
        y[i] = (y[i] - y[k] + M) * x[i-k-1] % M;
    ll lst = 0; tmp[0] = 1;
    for(int k=0; k<n; k++) for(int i=0; i<n; i++) {
        res[i] = (res[i] + y[k] * tmp[i]) % M;
        swap(lst, tmp[i]);
        tmp[i] = (tmp[i] - lst * k) % M;
        if(tmp[i] < 0) tmp[i] += M;
    } return res; }
```

4.18 Matroid Intersection

```
struct Matroid{
    virtual bool check(int i) = 0; // 0(R^2N), 0(R^2N)
    virtual void insert(int i) = 0; // 0(R^3), 0(R^2N)
    virtual void clear() = 0; // 0(R^2), 0(RN)
};

template<typename cost_t>
vector<cost_t> MI(const vector<cost_t> &cost, Matroid *m1,
Matroid *m2){
    int n = cost.size();
    vector<pair<cost_t, int>> dist(n+1);
    vector<vector<pair<int, cost_t>>> adj(n+1);
    vector<int> pv(n+1), inq(n+1), flag(n); deque<int> dq;
    auto augment = [&]() -> bool {
        fill(dist.begin(), dist.end(),
        pair(numeric_limits<cost_t>::max()/2, 0));
        fill(adj.begin(), adj.end(), vector<pair<int, cost_t>>());
        fill(pv.begin(), pv.end(), -1); fill(inq.begin(), inq.end(), 0);
        dq.clear(); m1->clear(); m2->clear();
        for(int i=0; i<n; i++) if(flag[i]) m1->insert(i), m2->insert(i);
        for(int i=0; i<n; i++){
            if(flag[i]) continue;
            if(m1->check(i))
```

```

    dist[pv[i]=i] = {cost[i], 0}, dq.push_back(i), inq[i]=1;
    if(m2->check(i)) adj[i].emplace_back(n, 0);
}
for(int i=0; i<n; i++){
    if(!flag[i] continue; m1->clear(); m2->clear();
    for(int j=0; j<n; j++) if(i != j && flag[j])
        m1->insert(j), m2->insert(j);
    for(int j=0; j<n; j++){
        if(flag[j]) continue;
        if(m1->check(j)) adj[i].emplace_back(j, cost[j]);
        if(m2->check(j)) adj[j].emplace_back(i, -cost[i]);
    }
}
while(dq.size()){
    int v = dq.front(); dq.pop_front(); inq[v] = 0;
    for(const auto &[i,w] : adj[v]){
        pair<cost_t, int> nxt{dist[v].ff+w, dist[v].ss+1};
        if(nxt < dist[i]){
            dist[i] = nxt; pv[i] = v;
            if(!inq[i]) dq.push_back(i), inq[i] = 1;
        } /* if */ } /* for [i,w] */ } /* while */
    if(pv[n] == -1) return false;
    for(int i=pv[n]; ; i=pv[i]){
        flag[i] ^= 1; if(i == pv[i]) break;
    } return true;
}; vector<cost_t> res;
while(augment()){
    cost_t now = cost_t(0);
    for(int i=0; i<n; i++) if(flag[i]) now += cost[i];
    res.push_back(now);
} return res;
}

```

5 String

5.1 KMP, Hash, Manacher, Z

```

vector<int> getFail(const container &pat){
    vector<int> fail(pat.size());
    //match: pat[0..j] and pat[j-i..i] is equivalent
    //ins/del: manipulate corresponding range to pattern starts at 0
    //      (insert/delete pat[i], manage pat[j-i..i])
    function<bool(int, int)> match = [&](int i, int j){ };
    function<void(int)> ins = [&](int i){ };
    function<void(int)> del = [&](int i){ };
    for(int i=1, j=0; i<pat.size(); i++){
        while(j && !match(i, j)){
            for(int s=i-j; s<i-fail[j-1]; s++) del(s);
            j = fail[j-1];
        }
        if(match(i, j)) ins(i), fail[i] = ++j;
    } return fail;
}
vector<int> doKMP(const container &str, const container &pat){
    vector<int> ret, fail = getFail(pat);
    //match: pat[0..j] and str[j-i..i] is equivalent
    //ins/del: manipulate corresponding range to pattern starts at 0
    //      (insert/delete str[i], manage str[j-i..i])
    function<bool(int, int)> match = [&](int i, int j){ };
    function<void(int)> ins = [&](int i){ };
    function<void(int)> del = [&](int i){ };
    for(int i=0, j=0, s; i<str.size(); i++){
        while(j && !match(i, j)){
            for(int s=i-j; s<i-fail[j-1]; s++) del(s);

```

```

        j = fail[j-1];
    }
    if(match(i, j)){
        if(j+1 == pat.size()){
            ret.push_back(i-j); for(s=i-j; s<i-fail[j]+1; s++) del(s);
            j = fail[j];
        } else ++j; ins(i);
    } return ret;
}
// # a # b # a # a # b # a #
// 0 1 0 3 0 1 6 1 0 3 0 1 0
vector<int> Manacher(const string &inp){
    int n = inp.size() * 2 + 1; vector<int> ret(n);
    string s = "#"; for(auto i : inp) s += i, s += "#";
    for(int i=0, p=-1, r=-1; i<n; i++){
        ret[i] = i <= r ? min(r-i, ret[2*p-i]) : 0;
        while(i-ret[i]-1 >= 0 && i+ret[i]+1 < n && s[i-ret[i]-1] == s[i+ret[i]+1]) ret[i]++;
        if(i+ret[i] > r) r = i+ret[i], p = i;
    } return ret;
}
// input: manacher array, 1-based hashing structure
// output: set of pair(hash_val, length)
set<pair<hash_t, int>> UniquePalindrome(const vector<int> &dp,
const Hashing &hashing){
    set<pair<hash_t, int>> st;
    for(int i=0, s, e; i<dp.size(); i++){
        if(!dp[i]) continue;
        if(i & 1) s = i/2 - dp[i]/2 + 1, e = i/2 + dp[i]/2 + 1;
        else s = (i-1)/2 - dp[i]/2 + 2, e = (i+1)/2 + dp[i]/2;
        for(int l=s, r=e; l<=r; l++, r--){
            auto now = hashing.get(l, r);
            auto [iter, flag] = st.emplace(now, r-l+1);
            if(!flag) break;
        }
    } return st;
}
//z[i]=match length of s[0,n-1] and s[i,n-1]
vector<int> Z(const string &s){
    int n = s.size(); vector<int> z(n); z[0] = n;
    for(int i=1, l=0, r=0; i<n; i++){
        while(i < r) z[i] = min(r-i, z[i-l]);
        while(i+z[i] < n && s[i+z[i]] == s[z[i]]) z[i]++;
        if(i+z[i] > r) r = i+z[i], l = i;
    } return z;
}

```

5.2 Aho-Corasick

```

struct Node{
    int g[26], fail, out;
    Node() { memset(g, 0, sizeof g); fail = out = 0; }
};
vector<Node> T(2); int aut[100101][26];
void Insert(int n, int i, const string &s){
    if(i == s.size()){ T[n].out++; return; }
    int c = s[i] - 'a';
    if(T[n].g[c] == 0) T[n].g[c] = T.size(), T.emplace_back();
    Insert(T[n].g[c], i+1, s);
}
int go(int n, int i){ // DO NOT USE `aut` DIRECTLY
    int &res = aut[n][i]; if(res) return res;
    if(n != 1 && T[n].g[i] == 0) res = go(T[n].fail, i);
    else if(T[n].g[i] != 0) res = T[n].g[i]; else res = 1;
}

```

```

    return res;
}
void Build(){
    queue<int> q; q.push(1); T[1].fail = 1;
    while(!q.empty()){
        int n = q.front(); q.pop();
        for(int i=0; i<26; i++){
            int next = T[n].g[i]; if(next == 0) continue;
            if(n == 1) T[next].fail=1; else T[next].fail=go(T[n].fail, i);
            q.push(next); T[next].out += T[T[next].fail].out;
        } /* for i */ } /* while q */ } /* build */
bool Find(const string &s){
    int n = 1, ok = 0;
    for(int i=0; i<s.size(); i++){
        n = go(n, s[i] - 'a'); if(T[n].out != 0) ok = 1;
    } return ok;
}
}

5.3  $O(N \log N)$  SA + LCP
pair<vector<int>, vector<int>> SuffixArray(const string &s){
    int n = s.size(), m = max(n, 256);
    vector<int> sa(n), lcp(n), pos(n), tmp(n), cnt(m);
    auto counting_sort = [&]() {
        fill(cnt.begin(), cnt.end(), 0);
        for(int i=0; i<n; i++) cnt[pos[i]]++;
        partial_sum(cnt.begin(), cnt.end(), cnt.begin());
        for(int i=n-1; i>=0; i--) sa[--cnt[pos[tmp[i]]]] = tmp[i];
    };
    for(int i=0; i<n; i++) sa[i] = i, pos[i] = s[i], tmp[i] = i;
    counting_sort();
    for(int k=1; ; k<=<=1){ int p = 0;
        for(int i=n-k; i<n; i++) tmp[p++] = i;
        for(int i=0; i<n; i++) if(sa[i] >= k) tmp[p++] = sa[i] - k;
        counting_sort(); tmp[sa[0]] = 0;
        for(int i=1; i<n; i++){
            tmp[sa[i]] = tmp[sa[i-1]];
            if(sa[i-1]+k < n && sa[i]+k < n && pos[sa[i-1]] == pos[sa[i]] && pos[sa[i-1]+k] == pos[sa[i]+k]) continue;
            tmp[sa[i]] += 1;
        }
        swap(pos, tmp); if(pos[sa.back()] + 1 == n) break;
    }
    for(int i=0, j=0; i<n; i++, j=max(j-1, 0)){
        if(pos[i] == 0) continue;
        while(sa[pos[i]-1]+j < n && sa[pos[i]]+j < n && s[sa[pos[i]-1]+j] == s[sa[pos[i]]+j]) j++;
        lcp[pos[i]] = j;
    } return {sa, lcp};
}
auto [SA, LCP] = SuffixArray(S); RMQ<int> rmq(LCP);
vector<int> Pos(N); for(int i=0; i<N; i++) Pos[SA[i]] = i;
auto get_lcp = [&](int a, int b){
    if(Pos[a] > Pos[b]) swap(a, b);
    return a == b ? (int)S.size() - a : rmq.query(Pos[a]+1, Pos[b]);
};
vector<pair<int, int>> can; // common substring {start, lcp}
vector<tuple<int, int, int>> valid; // valid substring {string, end_l, end_r}

```



```

for(int i=1; i<N; i++){
    if(SA[i] < X && SA[i-1] > X) can.emplace_back(SA[i], LCP[i]);
    if(i+1 < N && SA[i] < X && SA[i+1] > X)
        can.emplace_back(SA[i], LCP[i+1]);
}
for(int i=0; i<can.size(); i++){
    int skip = i > 0 ? min({can[i-1].second, can[i].second,
        get_lcp(can[i-1].first, can[i].first)}) : 0;
    valid.emplace_back(can[i].first, can[i].first + skip,
        can[i].first + can[i].second - 1);
}

```

5.4 $O(N \log N)$ Tandem Repeats

```

// return 0(n log n) tuple {l, r, p} that
// [i, i+p) = [i+p, i+2p) for all l <= i < r
vector<tuple<int,int,int>> TandemRepeat(const string &s){
    int n = s.size(); vector<tuple<int,int,int>> res;
    string t = s; reverse(t.begin(), t.end());
    // WARNING: add empty suffix!!
    // sa.insert(sa.begin(), n) before calculate lcp/pos
    auto [sa_s, lcp_s, pos_s] = SuffixArray(s);
    auto [sa_t, lcp_t, pos_t] = SuffixArray(t);
    RMQ<int> rmq_s(lcp_s), rmq_t(lcp_t);
    auto get = [n](const vector<int> &pos, const RMQ<int> &rmq,
        int a, int b){
        if(pos[a] > pos[b]) swap(a, b);
        return a == b ? n - a : rmq.query(pos[a] + 1, pos[b]);
    };
    for(int p=1; p*2<=n; p++){
        for(int i=0, j=-1; i+p<=n; i+=p){
            int l = i - get(pos_t, rmq_t, n-i-p, n-i);
            int r = i - p + get(pos_s, rmq_s, i, i+p);
            if(l <= r && l != j) res.emplace_back(j+1, r+1, p);
        } return res;
    } // Check p = 0, time complexity O(n log n)
}

```

5.5 Suffix Automaton

```

template<typename T, size_t S, T init_val>
struct initialized_array : public array<T, S> {
    initialized_array(){ this->fill(init_val); }
};

template<class Char_Type, class Adjacency_Type>
struct suffix_automaton{
    // Begin States
    // len: length of the longest substring in the class
    // link: suffix link
    // firstpos: minimum value in the set endpos
    vector<int> len{0}, link{-1}, firstpos{-1}, is_clone{false};
    vector<Adjacency_Type> next{};
    ll ans{0LL}; // 서로 다른 부분 문자열 개수
    // End States
    void set_link(int v, int lnk){
        if(link[v] != -1) ans -= len[v] - len[link[v]];
        link[v] = lnk;
        if(link[v] != -1) ans += len[v] - len[link[v]];
    }
    int new_state(int l, int sl, int fp, bool c, const
        Adjacency_Type &adj){
        int now = len.size(); len.push_back(l); link.push_back(-1);
        set_link(now, sl); firstpos.push_back(fp);
        is_clone.push_back(c); next.push_back(adj); return now;
    }
}

```

```

} int last = 0;
void extend(const vector<Char_Type> &s){
    last = 0; for(auto c: s) extend(c); }
void extend(Char_Type c){
    int cur = new_state(len[last] + 1, -1, len[last], false,
        {}), p = last;
    while(~p && !next[p][c]) next[p][c] = cur, p = link[p];
    if(!~p) set_link(cur, 0);
    else{
        int q = next[p][c];
        if(len[p] + 1 == len[q]) set_link(cur, q);
        else{
            int clone = new_state(len[p] + 1, link[q], firstpos[q],
                true, next[q]);
            while(~p && next[p][c] == q) next[p][c] = clone, p =
                link[p];
            set_link(cur, clone); set_link(q, clone);
        }
    }
    last = cur;
}

```

5.6 Bitset LCS

```

#include <x86intrin.h>
template<size_t _Nw> void _M_do_sub(_Base_bitset<_Nw> &A, const
    _Base_bitset<_Nw> &B){
    for(int i=0, c=0; i<_Nw; i++) c = _subborrow_u64(c, A._M_w[i],
        B._M_w[i], (ull*)&A._M_w[i]);
}
void _M_do_sub(_Base_bitset<1> &A, const _Base_bitset<1> &B){
    A._M_w -= B._M_w; }
template<size_t _Nb> bitset<_Nb> & operator-=(bitset<_Nb> &A,
    const bitset<_Nb> &B){
    _M_do_sub(A, B); return A;
}
template<size_t _Nb> inline bitset<_Nb> operator-(const
    bitset<_Nb> &A, const bitset<_Nb> &B){
    bitset<_Nb> C(A); return C -= B;
}
char s[50050], t[50050];
int lcs(){ // O(NM/64)
    bitset<50050> dp, ch[26];
    int n = strlen(s), m = strlen(t);
    for(int i=0; i<m; i++) ch[t[i]-'A'].set(i);
    for(int i=0; i<n; i++){ auto x = dp | ch[s[i]-'A']; dp = dp -
        (dp ^ x) & x; }
    return dp.count();
}

```

5.7 Lyndon Factorization, Minimum Rotation

```

// link[i]: length of smallest suffix of s[0..i-1]
// factorization result: s[res[i]..res[i+1]-1]
vector<int> Lyndon(const string &s){
    int n = s.size(); vector<int> link(n);
    for(int i=0; i<n; ){
        int j=i+1, k=i; link[i] = 1;
        for(; j<n && s[k]<=s[j]; j++){
            if(s[j] == s[k]) link[j] = link[k], k++;
        }
    }
}

```

```

        else link[j] = j - i + 1, k = i;
    } for(; i<=k; i+=j-k);
} vector<int> res;
for(int i=n-1; i>=0; i-=link[i]) res.push_back(i-link[i]+1);
reverse(res.begin(), res.end()); return res;
}
// rotate(v.begin(), v.begin()+min_rotation(v), v.end());
template<typename T> int min_rotation(T s){ // O(N)
    int a = 0, N = s.size();
    for(int i=0; i<N; i++) s.push_back(s[i]);
    for(int b=0; b<N; b++) for(int k=0; k<N; k++){
        if(a+k == b || s[a+k] < s[b+k]){ b += max(0, k-1); break; }
        if(s[a+k] > s[b+k]){ a = b; break; }
    }
    return a;
}

```

5.8 All LCS

```

void AllLCS(const string &s, const string &t){
    vector<int> h(t.size()); iota(h.begin(), h.end(), 0);
    for(int i=0, v=-1; i<s.size(); i++, v=-1){
        for(int r=0; r<t.size(); r++){
            if(s[i] == t[r] || h[r] < v) swap(h[r], v);
            //LCS[s[0..i],t[1..r]] = r-1+1 - sum([h[x] >= 1] | x <= r)
        } /*for r*/ } /* for i */ } /* end*/
}

```

6 Misc

6.1 CMakeLists.txt

```

set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_FLAGS "-DLOCAL -lm -g -Wl,--stack,268435456")
add_compile_options(-Wall -Wextra -Winvalid-pch -Wfloat-equal
    -Wno-sign-compare -Wno-misleading-indentation -Wno-parentheses)
# add_compile_options(-O3 -mavx -mavx2 -mfma)

```

6.2 Stack Hack

```

int main2(){ return 0; }
int main(){
    size_t sz = 1<<29; // 512MB
    void* newstack = malloc(sz);
    void* sp_dest = newstack + sz - sizeof(void*);
    asm __volatile__ ("movq %0, %%rax\n\t"
        "movq %%rsp, (%%rax)\n\t"
        "movq %0, %%rsp\n\t": : "r"(sp_dest) : );
    main2();
    asm __volatile__ ("pop %%rsp\n\t");
    return 0; }

```

6.3 Python Decimal

```

from fractions import Fraction
from decimal import Decimal, getcontext
getcontext().prec = 250 # set precision
N, two, itwo = 200, Decimal(2), Decimal(0.5)
# sin(x) = sum (-1)^n x^(2n+1) / (2n+1)!
# cos(x) = sum (-1)^n x^(2n) / (2n)!
def angle(cosT):
    #given cos(theta) in decimal return theta
    for i in range(N): cosT=((cosT+1)/two)**itwo
    sinT = (1-cosT*cosT)**itwo
    return sinT*(2**N)
pi = angle(Decimal(-1))

```

6.4 Java I/O

```
// java.util.*, java.math.*, java.io.*
public class Main{ // BufferedReader, BufferedWriter
public static void main(String[] args) throws IOException {
br=new BufferedReader(new InputStreamReader(System.in));
bw=new BufferedWriter(new OutputStreamWriter(System.out));
String[] ar = br.readLine().split(" ");
int a=Integer.parseInt(ar[0]), b=Integer.parseInt(ar[1]);
bw.write(String.valueOf(a+b)+"\n");br.close();bw.close();
ArrayList<Integer> a = new ArrayList<>();
a.add(1234); a.get(0); a.remove(a.size()-1); a.clear();
}}
```

6.5 Calendar

```
int f(int y,int m,int d){// 0: Sat, 1: Sun, ...
if (m<=2) y--, m+=12; int c=y/100; y%=100;
int w=((c>>2)-(c<<1)+y+(y>>2)+(13*(m+1)/5)+d-1)%7;
if (w<0) w+=7; return w; }
```

6.6 Ternary Search

```
while(s + 3 <= e){
T l = (s + s + e) / 3, r = (s + e + e) / 3;
if (Check(l) > Check(r)) s = l; else e = r;
} // get minimum / when multiple answer, find minimum `s`
T mn = INF, idx = s;
for(T i=s; i<=e; i++) if(T now = Check(i); now < mn) mn = now,
idx = i;
```

6.7 Add/Mul Update, Range Sum Query

```
struct Lz{
ll a, b; // constructor, clear(a = 1, b = 0)
Lz& operator+=(const Lz &t); // a += t.a, b = t.a * b + t.b
};
struct Ty{
ll cnt, sum; // constructor cnt=1, sum=0
Ty& operator += (const Ty &t); // cnt += t.cnt, sum += t.sum
Ty* operator += (const Lz &t); // sum= t .a * sum + cnt * t.b
};
```

6.8 $O(N \times \max W)$ Subset Sum (Fast Knapsack)

```
//  $O(N \times \max W)$ , maximize sumW <= t
int Knapsack(vector<int> w, int t){
int a = 0, b = 0, x;
while(b < w.size() && a + w[b] <= t) a += w[b++];
if(b == w.size()) return a;
int m = *max_element(w.begin(), w.end());
vector<int> u, v(2*m, -1); v[a+m-t] = b;
for(int i=b; (u=v,i<w.size()); i++){
for(x=0; x<m; x++) v[x+w[i]] = max(v[x+w[i]], u[x]);
for(x=2*m; --x>m; ) for(int j=x-m+0,u[x]); j<v[x]; j++)
v[x-w[j]] = max(v[x-w[j]], j);
} for(a=t; v[a+m-t]<0; a--); return a;
}
```

6.9 Monotone Queue Optimization

```
template<class T, bool GET_MAX = false> // D[i] = func_{0 <= j <
i} D[j] + cost(j, i)
pair<vector<T>, vector<int>> monotone_queue_dp(int n, const
vector<T> &init, auto cost){
assert((int)init.size() == n + 1); // cost function -> auto,
do not use std::function
```

```
vector<T> dp = init; vector<int> prv(n+1);
auto compare = [](T a, T b){ return GET_MAX ? a < b : a > b;
};
auto cross = [&](int i, int j){
int l = j, r = n + 1;
while(l < r){
int m = (l + r + 1) / 2;
if(compare(dp[l] + cost(i, m), dp[j] + cost(j, m))) r = m
- 1; else l = m;
} return l; };
deque<int> q[0];
for(int i=1; i<=n; i++){
while(q.size() > 1 && compare(dp[q[0]] + cost(q[0], i),
dp[q[1]] + cost(q[1], i))) q.pop_front();
dp[i] = dp[q[0]] + cost(q[0], i); prv[i] = q[0];
while(q.size() > 1 && cross(q[q.size()-2], q.back()) >=
cross(q.back(), i)) q.pop_back();
q.push_back(i);
} /*for end*/ return {dp, prv}; }
```

6.10 Slope Trick

```
//NOTE:  $f(x)=\min\{f(x+i), i<a\}+|x-k|+m \rightarrow pf(k)sf(k)ab(-a,m)$ 
//NOTE: sf_inc에 답구하는게 들어있어서, 반드시 한 연산에 대해
pf_dec->sf_inc순서로 호출
struct LeftHull{
void pf_dec(int x){ pq.emplace(x-bias); } //x이하의 기울기들 -1
int sf_inc(int x){ //x이상의 기울기들 +1, pop된 원소 반환(Right
Hull관리에 사용됨)
if(pq.empty() or argmin()<=x) return x; ans += argmin()-x; //
이 경우 최소값이 증가함
pq.emplace(x-bias); /*x 이하 -1*/ int r=argmin(); pq.pop(); /*전체
+1*/
return r;
}
void add_bias(int x,int y){ bias+=x; ans+=y; } int minval(){
return ans; } //x축 평행이동, 최소값
int argmin(){return pq.empty()?-inf<int>():pq.top()+bias; } //
최소값 x좌표
void operator+=(LeftHull& a){ ans+=a.ans; while(sz(a.pq))
pf_dec(a.argmin()), a.pq.pop(); }
int size()const{return sz(pq); } PQMax<int> pq; int ans=0,
bias=0;
};
//NOTE:  $f(x)=\min\{f(x+i), a<i<b\}+|x-k|+m \rightarrow pf(k)sf(k)ab(-a,b,m)$ 
struct SlopeTrick{
void pf_dec(int x){ l.pf_dec(-r.sf_inc(-x)); }
void sf_inc(int x){ r.pf_dec(-l.sf_inc(x)); }
void add_bias(int lx,int rx,int
y){ l.add_bias(lx,0), r.add_bias(-rx,0), ans+=y; }
int minval(){return ans+l.minval()+r.minval(); }
int argmin(){return {l.argmin(), -r.argmin()}; }
void operator+=(SlopeTrick& a){
while(sz(a.l.pq)) pf_dec(a.l.argmin()), a.l.pq.pop();
l.ans+=a.l.ans;
while(sz(a.r.pq)) sf_inc(-a.r.argmin()), a.r.pq.pop();
r.ans+=a.r.ans; ans+=a.ans;
} LeftHull l,r; int ans=0;
int size()const{return l.size()+r.size(); }
};
//LeftHull 역추적 방법: 스텝i의 argmin값을 am(i)라고 하자. 스텝n부터
스텝1까지 ans[i]=min(ans[i+1], am(i))하면 된다. 아래는 증명..은
아니고 간략한 이유
```

```
//am(i)<=ans[i+1]일때: ans[i]=am(i)
//x[i]>ans[i+1]일때: ans[i]=ans[i+1] 왜냐하면 f(i,a)는 a<x[i]에서
감소함수이므로 가능한 최대 오른쪽으로 붙은 ans[i+1]이 최적.
//스텝i에서 add_bias(k,0)한다면 간격제한k가 있는것이므로
ans[i]=min(ans[i+1]-k, x[i])으로 수정.
//LR Hull 역추적은 케이스나뉘서 위 방법을 확장하면 됨
```

6.11 Aliens Trick

```
// pair<T, vector<int>> f(T c): return opt_val, prv
// cost function must be multiplied by 2
template<class T, bool GET_MAX = false>
pair<T, vector<int>> AliensTrick(int n, int k, auto f, T lo, T hi){
T l = lo, r = hi; while(l < r) {
T m = (l + r + (GET_MAX?1:0)) >> 1;
vector<int> prv = f(m*2+(GET_MAX?-1:1)).second;
int cnt = 0; for(int i=n; i; i=prv[i]) cnt++;
if(cnt <= k) (GET_MAX?l:r) = m;
else (GET_MAX?r:l) = m + (GET_MAX?-1:1);
}
T opt_value = f(l*2).first / 2 - k*l;
vector<int> prv1 = f(l*2+(GET_MAX?1:-1)).second, p1{n};
vector<int> prv2 = f(l*2-(GET_MAX?1:-1)).second, p2{n};
for(int i=n; i; i=prv1[i]) p1.push_back(prv1[i]);
for(int i=n; i; i=prv2[i]) p2.push_back(prv2[i]);
reverse(p1.begin(), p1.end()); reverse(p2.begin(), p2.end());
assert(p2.size() <= k+1 && k+1 <= p1.size());
if(p1.size() == k+1) return {opt_value, p1};
if(p2.size() == k+1) return {opt_value, p2};
for(int i=1, j=1; i<p1.size(); i++){
while(j < p2.size() && p2[j] < p1[i-1]) j++;
if(p1[i] <= p2[j] && i - j == k+1 - (int)p2.size()){
vector<int> res;
res.insert(res.end(), p1.begin(), p1.begin()+i);
res.insert(res.end(), p2.begin()+j, p2.end());
return {opt_value, res};
} /* if */ } /* for */ assert(false);
}
```

6.12 SWAMK, Min Plus Convolution

```
// find the indices of row maxima, smallest index when tie
template <class F, class T=long long>
vector<int> SMAWK(F f, int n, int m){
vector<int> ans(n, -1);
auto rec = [&](auto self, int*const rs, int x, int*const cs,
int y){
const int t = 8;
if(x <= t || y <= t){
for(int i=0; i<x; i++){ int r = rs[i]; T mx;
for(int j=0; j<y; j++){
int c = cs[j]; T w = f(r, c);
if(ans[r] == -1 || w > mx) ans[r] = c, mx = w;
}} /* for j i */ return; } /* base case */
if(x < y){ int s = 0;
for(int i=0; i<y; i++){ int c = cs[i];
while(s && f(rs[s-1], cs[s-1]) < f(rs[s-1], c)) s--;
if(s < x) cs[s++] = c;
} y = s;
}
int z=0, k=0, *a=rs+x, *b=cs+y;
```

```
for(int i=1; i<x; i+=2) a[rs[i]] = rs[i];
for(int i=0; i<y; i++) b[i] = cs[i];
self(self, a, z, b, y);
for(int i=0; i<x; i+=2){
    int to = i+1 < x ? ans[rs[i+1]] : cs[y-1]; T mx;
    while(true){
        T w = f(rs[i], cs[k]);
        if(ans[rs[i]] == -1 || w > mx) ans[rs[i]]=cs[k], mx=w;
        if(cs[k] == to) break; k++;
    }
};
int *rs = new int[n*2]; iota(rs,rs+n,0);
int *cs = new int[max(m, n*2)]; iota(cs,cs+m,0);
rec(rec,rs,n,cs,m);delete[]rs;delete[]cs;return ans;
}
// A: convex, B: arbitrary, O((N+M) log M)
int N, M, A[1<<19], B[1<<19], C[1<<20];
void DnC(int s, int e, int l, int r){
    if(s > e) return;
    int m = (s+e)/2, opt = -1, &mn = C[m];
    for(int i=l; i<=min(m,r); i++){
        if(m - i >= N) continue;
        if(opt == -1 || A[m-i] + B[i] < mn) mn=A[m-i]+B[i], opt=i;
    } DnC(s, m-1, l, opt); DnC(m+1, e, opt, r);
} // or...
int f(int r, int c){//O(N+M) but not fast
    if(0 <= r-c && r-c < N) return -(A[r-c] + B[c]);
    else return -21e8 - (r - c); // min
} SMAWK(f, N+M-1, M); // DnC opt 163ms SMAWK 179ms N,M=2^19
```

6.13 Money for Nothing (WF17D)

```
11 MoneyForNothing(vector<Point> lo, vector<Point> hi){
    sort(lo.begin(), lo.end()); sort(hi.rbegin(), hi.rend());//rev
    vector<Point> a, b; ll res = 0;
    for(auto p:lo)if(a.empty() || a.back().y > p.y)a.push_back(p);
    for(auto p:hi)if(b.empty() || b.back().y < p.y)b.push_back(p);
    reverse(b.begin(),b.end()); if(a.empty()||b.empty()) return 0;
    queue<tuple<int,int,int,int>> que;
    que.emplace(0, (int)a.size()-1, 0, (int)b.size()-1);
    while(!que.empty()){
        auto [s,e,l,r] = que.front(); que.pop();
        int m = (s + e) / 2, pos = 1; ll mx = -4e18;
        for(int i=l; i<=r; i++){
            ll dx = b[i].x - a[m].x, dy = b[i].y - a[m].y;
            ll now = (dx < 0 && dy < 0) ? 0 : dx * dy;
            if(now > mx) mx = now, pos = i;
        } res = max(res, mx);
        if(s < m) que.emplace(s, m-1, l, pos);
        if(m < e) que.emplace(m+1, e, pos, r);
    } return res;
}
```

6.14 Exchange Argument on Tree (WF16L,CERC13E)

```
struct Info{ // down a -> up b, a b >= 0
    ll a, b, idx; Info() : Info(0, 0, 0) {}
    Info(ll a, ll b, ll idx) : a(a), b(b), idx(idx) {}
    bool operator < (const Info &t) const {
        ll le = b - a, ri = t.b - t.a;
        if(le >= 0 && ri < 0) return false;
        if(le < 0 && ri >= 0) return true;
        if(le < 0 && b != t.b) return b < t.b;
```

```
if(le >= 0 && a != t.a) return a > t.a;
return idx < t.idx;
}
Info& operator += (const Info &v){
    ll aa = min(-a, -a+b-v.a), bb = -a+b-v.a+v.b;
    a = -aa; b = bb - aa; return *this;
}
};
void MonsterHunter(int root=1){
    set<Info> T(A+1, A+N+1); T.erase(A[root]);
    while(!T.empty()){
        auto v = *T.rbegin(); T.erase(prev(T.end()));
        int now = v.idx, nxt = Find(Par[v.idx]); // @TODO
        UF[now] = nxt; T.erase(A[nxt]); A[nxt] += A[now];
        if(nxt != root) T.insert(A[nxt]);
    } // @TODO
}
```

6.15 Hook Length Formula

```
int HookLength(const vector<int> &young){
    if(young.empty()) return 1;
    vector<int> len(young[0]);
    ll num = 1, div = 1, cnt = 0;
    for(int i=(int)young.size()-1; i>=0; i--){
        for(int j=0; j<young[i]; j++){
            num = num * ++cnt % MOD;
            div = div * ((+len[j] + young[i] - j - 1) % MOD);
        }
    }
    return num * Pow(div, MOD-2) % MOD;
}
```

6.16 Floating Point Add (Kahan)

```
template<typename T> T float_sum(vector<T> v){
    T sum=0, c=0, y, t; // sort abs(v[i]) increase?
    for(T i:v) y=i-c, t=sum+y, c=(t-sum)-y, sum=t;
    return sum; //worst O(eps*N), avg O(eps*sqrtN)
} //dnc: worst O(eps*logN), avg O(eps*sqrtlogN)
```

6.17 Random, PBDS, Bit Trick, Bitset

```
mt19937
rd((unsigned)chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int> rnd_int(1, r); // rnd_int(rd)
uniform_real_distribution<double> rnd_real(0, 1); // rnd_real(rd)
// ext/pb_ds/assoc_container.hpp, tree_policy.hpp, rope
// namespace __gnu_pbds (find_by_order, order_of_key)
// namespace __gnu_cxx (append(str), substr(l, r), at(idx))
template <typename T> using ordered_set = tree<T, null_type,
less<T>, rb_tree_tag, tree_order_statistics_node_update>;
bool next_combination(T &bit, int N){
    T x = bit & -bit, y = bit + x;
    bit = (((bit & ~y) / x) >> 1) | y;
    return (bit < (1LL << N)); }
long long next_perm(long long v){
    long long t = v | (v-1);
    return (t + 1) | (((~t & ~t) - 1) >> (__builtin_ctz(v) + 1));
} // __builtin_clz/ctz/popcount
for(submask=mask; submask; submask=(submask-1)&mask);
for(supermask=mask; supermask<(1<<n);
supermask=(supermask+1)&mask);
int frq(int n, int i) { int j, r = 0; // # of digit i in [1, n]
```

```
for (j = 1; j <= n; j *= 10) if (n / j / 10 >= !i) r += (n /
10 / j - !i) * j + (n / j % 10 > i ? j : n / j % 10 == i ? n %
j + 1 : 0);
return r; }
bitset<17> bs; bs[1] = bs[7] = 1; assert(bs._Find_first() == 1);
assert(bs._Find_next(0) == 1 && bs._Find_next(1) == 7);
assert(bs._Find_next(3) == 7 && bs._Find_next(7) == 17);
cout << bs._Find_next(7) << "\n";
template <int len = 1> // Arbitrary sized bitset
void solve(int n){ // solution using bitset<len>
    if(len < n){ solve<std::min(len*2, MAXLEN)>(n); return; } }
```

6.18 Fast I/O, Fast Div, Fast Mod

```
namespace io { // thanks to cgiosy
const signed IS=1<<20; char I[IS+1],*J=I;
inline void daer(){if(J>=I+IS-64){
    char*p=I;do*p++=*J++;
    while(J!=I+IS);p[read(0,p,I+IS-p)]=0;J=I;}}
template<int N=10,typename T=int>inline T getu(){
    daer();T x=0;int k=0;do x=x*10+*J-'0';
    while(++J>='0'&&+k<N);++J;return x;}
template<int N=10,typename T=int>inline T geti(){
    daer();bool e=*J=='-';J+=e;return(e?-1:1)*getu<N,T>();}
struct f{f(){I[read(0,I,IS)]=0;}}flu; };
struct FastMod{ // typedef __uint128_t L;
    ull b, m; FastMod(ull b) : b(b), m(ull)((L(1) << 64) / b) {}
    ull reduce(ull a){ // can be proven that 0 <= r < 2*b
        ull q = (ull)((L(m) * a) >> 64), r = a - q * b;
        return r >= b ? r - b : r;
    }
};
inline pair<uint32_t, uint32_t> Div(uint64_t a, uint32_t b){
    if(__builtin_constant_p(b)) return {a/b, a%b};
    uint32_t lo=a, hi=a>>32;
    __asm__ ("div %2" : "+a,a" (lo), "+d,d" (hi) : "r,m" (b));
    return {lo, hi}; // BOJ 27505, q r < 2^32
} // divide 10M times in ~400ms
ull mulmod(ull a, ull b, ull M){ // ~2x faster than int128
    ll ret = a * b - M * ull(1.L / M * a * b);
    return ret + M * (ret < 0) - M * (ret >= (ll)M);
} // safe for 0 <= a,b < M < (1<<63) when long double is 80bit
```

6.19 DP Optimization

```
// Quadrangle Inequality : C(a, c)+C(b, d) <= C(a, d)+C(b, c)
// Monotonicity : C(b, c) <= C(a, d)
// CHT, DnC Opt(Quadrangle), Knuth(Quadrangle and Monotonicity)
// Knuth: K[i][j-1] <= K[i][j] <= K[i+1][j]
// 1. Calculate D[i][i], K[i][i]
// 2. Calculate D[i][j], K[i][j] (i < j)
// Another: D[i][j] = min(D[i-1][k] + C[k+1][j]), C quadrangle
// i=1..k j=n..1 k=K[i-1,j]..K[i,j+1] update, vnoi/icpc22_mn_c
```

6.20 Highly Composite Numbers, Large Prime

< 10 ^k	number	divisors	2	3	5	7	11	13	17	19	23	29	31	37
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
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
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 1

< 10[~]k prime # of prime < 10[~]k prime

1	7	4	10	9999999967
2	97	25	11	99999999977
3	997	168	12	99999999989
4	9973	1229	13	999999999971
5	99991	9592	14	9999999999973
6	999983	78498	15	9999999999989
7	9999991	664579	16	99999999999937
8	99999989	5761455	17	99999999999997
9	999999937	50847534	18	999999999999989

6.21 DLAS(Diversified Late Acceptance Search)

```
template<class T, class U>
T incMod(T x, U mod) { x += 1; return x == mod ? 0 : x; }
template<class Domain, class CoDomain, size_t LEN = 5>
pair<Domain, CoDomain> dlas( function<CoDomain(Domain&)> f,
function<void(Domain&)> mutate,
Domain const& initial, u64 maxIdleIters = -1ULL) {
array<Domain, 3> S{initial, initial, initial};
CoDomain curF = f(S[0]), minF = curF;
size_t curPos = 0, minPos = 0, k = 0;
array<CoDomain, LEN> fitness; fitness.fill(curF);
for(u64 idleIters=0; idleIters<maxIdleIters; idleIters++){
CoDomain prvF = curF;
size_t newPos = incMod(curPos, 3);
if (newPos == minPos) newPos = incMod(newPos, 3);
Domain &curS = S[curPos], &newS = S[newPos];
newS = curS; mutate(newS); CoDomain newF = f(newS);
if(newF < minF) idleIters=0, minPos=newPos, minF=newF;
if(newF == curF || newF < *max_element(all(fitness))){
curPos = newPos; curF = newF;
} CoDomain& fit = fitness[k]; k = incMod(k, LEN);
if(curF > fit || curF < fit && curF < prvF) fit = curF;
} return { S[minPos], minF };
} // 점수 최소화하는 함수, f: 상태의 점수를 반환
//dlas<state_type, score_type>(f, mutate, initial, maxIdleIter)
//initial:초기 상태, mutate:상태를 참조로 받아서 임의로 수정(반환X)
//maxIdleIters:지역 최적해에서 얼마나 오래 기다릴지
```

7 Notes

7.1 Triangles/Trigonometry

- k 차원 구 부피: $V_{2k} = \pi^k/k!, v_{2k+1} = 2^{k+1}\pi^k/(2k+1)!!$
- $\sin(\alpha \pm \beta) = \sin \alpha \cos \beta \pm \cos \alpha \sin \beta$, $\sin 2\theta = 2 \sin \theta \cos \theta$
- $\cos(\alpha \pm \beta) = \cos \alpha \cos \beta \mp \sin \alpha \sin \beta$, $\cos 2\theta = 1 - 2 \sin^2 \theta$
- $\tan(\alpha \pm \beta) = \frac{\tan \alpha \pm \tan \beta}{1 \mp \tan \alpha \tan \beta}$, $\tan 2\theta = \frac{2 \tan \theta}{1 - \tan^2 \theta}$
- 반각 공식: $\sin^2 \theta/2 = \frac{1 - \cos \theta}{2}$, $\cos^2 \theta/2 = \frac{1 + \cos \theta}{2}$, $\tan^2 \theta/2 = \frac{1 - \cos \theta}{1 + \cos \theta}$

- 변 길이 $a, b, c; p = (a + b + c)/2$
- 넓이 $A = \sqrt{p(p-a)(p-b)(p-c)}$
- 외접원 반지름 $R = abc/4A$, 내접원 반지름 $r = A/p$
- 중선 길이 $m_a = 0.5\sqrt{2b^2 + 2c^2 - a^2}$, 각 이등분선 $s_a = \sqrt{bc(1 - \frac{a^2}{b+c^2})}$
- 사인 법칙 $\frac{\sin A}{a} = 1/2R$, 코사인 법칙 $a^2 = b^2 + c^2 - 2bc \cos A$
- 탄젠트 법칙 $\frac{a+b}{a-b} = \frac{\tan(A+B)/2}{\tan(A-B)/2}$
- 중심 좌표 $(\frac{\alpha x_a + \beta x_b + \gamma x_c}{\alpha + \beta + \gamma}, \frac{\alpha y_a + \beta y_b + \gamma y_c}{\alpha + \beta + \gamma})$

이름	α	β	γ	
외심	$a^2\mathcal{A}$	$b^2\mathcal{B}$	$c^2\mathcal{C}$	$\mathcal{A} = b^2 + c^2 - a^2$
내심	a	b	c	$\mathcal{B} = a^2 + c^2 - b^2$
무계중심	1	1	1	$\mathcal{C} = a^2 + b^2 - c^2$
수심	\mathcal{BC}	\mathcal{CA}	\mathcal{AB}	
방심(A)	$-a$	b	c	

7.2 Calculus, Newton's Method

- $(\arcsin x)' = 1/\sqrt{1-x^2}$
- $(\tan x)' = 1 + \tan^2 x$
- $\int \tan ax = -\ln |\cos ax|/a$
- $(\arccos x)' = -1/\sqrt{1-x^2}$
- $(\arctan x)' = 1/(1+x^2)$
- $\int x \sin ax = (\sin ax - ax \cos ax)/a^2$
- Newton: $x_{n+1} = x_n - f(x_n)/f'(x_n)$
- $\oint_C (Ldx + Mdy) = \int_D (\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y})dxdy$
- where C is positively oriented, piecewise smooth, simple, closed; D is the region inside C ; L and M have continuous partial derivatives in D .

7.3 Zeta/Mobius Transform

- Subset Zeta/Mobius Transform
 - for $i=0..n-1$ for $j=0..2^n-1$ if (i and j) $v[j] \pm= v[i \text{ xor } j]$
- Superset Zeta/Mobius Transform
 - for $i=0..n-1$ for $j=0..2^n-1$ if (i and j) $v[i \text{ xor } j] \pm= v[j]$
- Divisor Zeta/Mobius Transform
 - for p:Prime for $i=1..n/p$ $v[i*p] += v[i]$
 - for p:Prime for $i=n/p..1$ $v[i*p] -= v[i]$
- Multiple Zeta/Mobius Transform
 - for p:Prime for $i=n/p..1$ $v[i] += v[i*p]$
 - for p:Prime for $i=1..n/p$ $v[i] -= v[i*p]$
- AND Convolution: SupZeta(A), SupZeta(B), SupMobius(mul)
- OR Conv.: Subset, GCD Conv.: Multiple, LCM Conv.: Divisor
- AND/OR: 2^{20} 0.3s, Subset: 2^{20} 2.5s, GCD/LCM: $1e6$ 0.3s

7.4 Counting

	조건 없음	단사 함수	전사 함수
함수 일치	k^n	$k!/(k-n)!$	$k! \times S_2(n, k)$
공 구별 X	$C(n+k-1, n)$	$C(k, n)$	$C(n-1, n-k)$ 조심
상자 구별 X	$B(n, k)$	$[n \leq k]$	$S_2(n, k)$
모두 구별 X	$P_k(n+k)$	$[n \leq k]$	$P_k(n)$

- 단사 함수: 상자에 공 최대 1개, 전사 함수: 상자에 공 최소 1개
- 중복 조합: $C(n+k-1, n) = H(n, k)$
- 공 구별 X, 전사 함수에서 $n = k = 0$ 일 때 정답 1인 거 조심
- 교란 순열: 모든 i 에 대해 $\pi(i) \neq i$ 가 성립하는 길이 n 순열 개수
 - 초항(0-based): 1, 0, 1, 2, 9, 44, 265, 1854
 - 일반항: $D(n) = \sum_{k=0}^n (-1)^k n! / k!$, $D(n) \approx n! / e$
 - 점화식: $D(0) = 1; D(1) = 0; D(n) = (n-1)(D(n-1) + D(n-2))$
 - 생성함수(EGF): $e^{-x}/(1-x)$
- 카탈란 수
 - 초항(0-based): 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786

- 일반항: $Cat(n) = Comb(2n, n)/(n+1) = C(2n, n) - C(2n, n+1)$
- 점화식: $Cat(0) = 1; Cat(n) = \sum_{k=0}^{n-1} Cat(k) \times Cat(n-k-1)$
- 생성함수(OGF): $(1 - \sqrt{1-4x})/(2x)$
- 여는 괄호 n 개, 닫는 괄호 $k(\leq n)$ 개 = $C(n+k, k) \times (n-k+1)/(n+1)$
- 제 1종 스티링 수: k 개의 사이클로 구성된 길이 n 순열 개수
 - 초항(0-based, 삼각형): 1, 0, 1, 0, 1, 1, 0, 2, 3, 1, 0, 6, 11, 6, 1
 - 점화식: $S_1(n, 0) = [n=0]; S_1(n, k) = S_1(n-1, k-1) + S_1(n-1, k) \times (n-1)$, 생성함수(EGF): $(-\ln(1-x))^k/k!$
 - 성질: $\sum_{k=0}^n S_1(n, k) = n!$
- 제 2종 스티링 수: n 개의 원소를 k 개의 공집합이 아닌 부분집합으로 분할
 - 초항(0-based, 삼각형): 1, 0, 1, 0, 1, 1, 0, 1, 3, 1, 0, 1, 7, 6, 1
 - 일반항: $S_2(n, k) = 1/k! \times \sum_{i=1}^n (-1)^{k-i} \times C(k, i) \times i^n$, 단 $S(0, 0) = 1$
 - 점화식: $S_2(n, 0) = [n=0]; S_2(n, k) = S_2(n-1, k-1) + S_2(n-1, k) \times k$
 - 생성함수(EGF): $(\exp(x) - 1)^k/k!$
 - 성질: A, B를 $n-k, [(k-1)/2]$ 의 커진 비트 위치라고 하면, $S_2(n, k) \bmod 2 = [A \cap B = \emptyset]$
 - 성질: $S_2(2n, n)$ 은 n 이 fibbinary number(연속한 1 없음) 일 때만 홀수
- 벨 수 $B(n)$: n 개의 원소를 분할하는 방법의 수
 - 초항(0-based): 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975
 - 일반항: $B(n) = \sum_{k=0}^n S_2(n, k)$, 몇 개의 상자를 버릴지 다 돌아보기
 - 점화식: $B(0) = 1; B(n) = \sum_{k=0}^{n-1} C(n-1, k) \times B(k)$
 - 생성함수(EGF): $\exp(\exp(x) - 1)$
- 벨 수 $B(n, k)$: n 개를 집합 k 개로 분할하는 방법의 수(공집합 허용)
 - 초항(0-based, 삼각형): 1 0 1 0 1 2 0 1 4 5 0 1 8 14 15
 - 일반항: $B(n, k) = \sum_{i=0}^k S_2(n, i) = \sum_{i=0}^k \frac{i^n}{i!} \sum_{j=0}^{k-i} \frac{(-1)^j}{j!}$
- 분할 수 $P(n)$: n 을 자연수 몇 개의 합으로 나타내는 방법의 수, 순서 X
 - 초항(0-based): 1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42
 - 점화식: $P(0) = 1$, 팀노트 어딘가에 있는 코드로 $O(n\sqrt{n})$ 가능
- 분할 수 $P(n, k)$: n 을 k 개의 자연수의 합으로 나타내는 방법의 수, 순서 X
 - 초항(0-based, 삼각형): 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 2, 1, 1, 0, 1, 2, 2, 1, 1
 - 점화식: $P(0, 0) = 1; P(n, 0) = 0; P(n, k) = P(n-1, k-1) + P(n-k, k)$
 - 성질: $\sum_{k=0}^n P(n, k) = P(n)$
- 피보나치 수 $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$
 - 일반항: $\frac{F_n = (\phi^n - \bar{\phi}^n)/\sqrt{5}, \phi = \{(1+\sqrt{5})/2\}^n$
 - 생성함수(OGF): $F(x) = x/(1-x-x^2)$
 - 성질: $F_1 + \dots + F_n = F_{n+2} - 1, F_1^2 + \dots + F_n^2 = F_n F_{n+1}$
 - 성질: $\gcd(F_n, F_m) = F_{\gcd(n, m)}, F_n^2 - F_{n+1} F_{n-1} = (-1)^{n-1}$
- 벨 수 $B(n, k)$ 식 전개
 - $B(n, k) = \sum_{j=0}^k S(n, j) = \sum_{j=0}^k 1/j! \sum_{i=0}^j (-1)^{j-i} j! C(i, k) \times i^n$
 - $= \sum_{j=0}^k \sum_{i=0}^j \frac{(-1)^{j-i}}{i!(j-i)!} i^n = \sum_{i=0}^k \sum_{j=i}^k \frac{(-1)^{j-i}}{i!(j-i)!} i^n$
 - $= \sum_{i=0}^k \sum_{j=0}^{k-i} \frac{(-1)^j}{i!j!} i^n = \sum_{i=0}^k \frac{i^n}{i!} \sum_{j=0}^{k-i} \frac{(-1)^j}{j!}$

7.5 Generating Function

- 등차수열: $(pn+q)x^n = p/(1-x) + q/(1-x)^2$
- 등비수열: $(rx)^n = (1-rx)^{-1}$
- 조합: $C(m, n)x^n = (1+x)^m$
- 중복조합: $C(m+n-1, n)x^n = (1-x)^{-m}$
- $f(n) = \sum_{k=0}^n k! \times S_2(n, k)$ 의 EGF: $1/(2-e^x)$
- Ordinary Generating Function $A(x) = \sum_{i \geq 0} a_i x^i$
 - $A(rx) \Rightarrow r^n a_n, xA(x)' \Rightarrow na_n$
 - $A(x) + B(x) \Rightarrow a_n + b_n, A(x)B(x) \Rightarrow \sum_{i=0}^n a_i b_{n-i}$
 - $A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} a_{i_1} a_{i_2} \dots a_{i_k}$
 - $\frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^n a_i$
- Exponential Generating Function $A(x) = \sum_{i \geq 0} \frac{a_i}{i!} x^i$
 - $A(x) + B(x) \Rightarrow a_n + b_n, A(x)B(x) \Rightarrow \sum_{i=0}^n \binom{n}{i} a_i b_{n-i}$
 - $A^{(k)}(x) \Rightarrow a_{n+k}, xA(x) \Rightarrow na_n$

$$A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} \binom{n}{i_1, i_2, \dots, i_k} a_{i_1} a_{i_2} \dots a_{i_k}$$

7.6 Faulhaber's Formula ($\sum_{k=1}^n k^c$)

- B_n : 베르누이 수
- 생성함수(egf): $\frac{x}{e^x-1} = \frac{1}{(e^x-1)/x} = \sum_{n=0}^{\infty} \frac{B_n}{n!} x^n$
- 일반항: $B_n = \sum_{k=0}^n \frac{k!(-1)^k}{k+1} S_2(n, k)$
- 점화식: $B_0 = 1; B_n = -\frac{1}{n+1} \sum_{r=0}^{n-1} \binom{n+1}{r} B_r$
- $\sum_{k=1}^n k^c = \sum_{k=0}^c \frac{(-1)^k}{c+1} \binom{c+1}{k} B_k n^{c+1-k}$

7.7 About Graph Degree Sequence

- 단순 무향 그래프(Erdos-Gallai): 차수열 $d_1 \geq \dots \geq d_n$ 의 합이 짝수 and 모든 $1 \leq k \leq n$ 에 대해 $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$
- 단순 이분 그래프(Gale-Ryser): 차수열 $a_1 \geq \dots \geq a_n, b_i$ 에서 $\text{sum}(a) = \text{sum}(b)$ and 모든 $1 \leq k \leq n$ 에 대해 $\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k)$
- 단순 방향 그래프(Fulkerson-Chen-Anstee): $a_1 \geq \dots \geq a_n$ 를 만족 하는 진입/진출 차수열 $(a_1, b_1), \dots, (a_n, b_n)$ 에서 $\text{sum}(a) = \text{sum}(b)$ and 모든 $1 \leq k \leq n$ 에 대해 $\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k-1) + \sum_{i=k+1}^n \min(b_i, k)$

7.8 Burnside, Grundy, Pick, Hall, Simpson, Area of Quadrangle, Fermat Point, Euler

- Burnside's Lemma
 - 수식
 - $G=(X,A)$: 집합X와 액션A로 정의되는 군G에 대해, $|A||X/A| = \text{sum}(|\text{Fixed points of } a|, \text{ for all } a \text{ in } A)$
 - X/A 는 Action으로 서로 변형가능한 X의 원소들을 동치로 묶었을때 동치류(파티션) 집합
 - 풀어쓰기
 - orbit: 그룹에 대해 두 원소 a,b와 액션f에 대해 $f(a)=b$ 인거에 간선연결한 컴포넌트(연결집합)
 - orbit개수 = $\text{sum}(\text{각 액션 } g \text{에 대해 } f(x)=x \text{인 } x(\text{고정점})\text{개수})/\text{액션개수}$
 - 자유도 치트시트
 - 회전 n개: 회전i의 고정점 자유도= $\text{gcd}(n,i)$
 - 임의뒤집기 $n=\text{홀수}$: n개 원소중심축(자유도 $(n+1)/2$)
 - 임의뒤집기 $n=\text{짝수}$: $n/2$ 개 원소중심축(자유도 $n/2+1$) + $n/2$ 개 원소안 지나는축(자유도 $n/2$)
- 알고리즘 게임
 - Nim Game의 해법 : 각 더미의 돌의 개수를 모두 XOR했을 때 0 이 아니면 첫번째, 0 이면 두번째 플레이어가 승리.
 - Grundy Number : 어떤 상황의 Grundy Number는, 가능한 다음 상황들의 Grundy Number를 모두 모은 다음, 그 집합에 포함 되지 않는 가장 작은 수가 현재 state의 Grundy Number가 된다. 만약 다음 state가 독립된 여러개의 state들로 나뉘는 경우, 각각의 state의 Grundy Number의 XOR 합을 생각한다.
 - Subtraction Game : 한 번에 k 개까지의 돌만 가져갈 수 있는 경우, 각 더미의 돌의 개수를 k + 1로 나눈 나머지를 XOR 합하여 판단한다.
 - Index-k Nim : 한 번에 최대 k개의 더미를 골라 각각의 더미에서 아무렇게나 돌을 제거할 수 있을 때, 각 binary digit에 대하여 합을 k + 1로 나눈 나머지를 계산한다. 만약 이 나머지가 모든 digit에 대하여 0이라면 두번째, 하나라도 0이 아니라면 첫번째 플레이어가 승리.
 - Misere Nim : 모든 돌 무더기가 1이면 N이 홀수일 때 후공 승, 그렇지 않은 경우 XOR 합 0이면 후공 승
- Pick's Theorem

- 격자점으로 구성된 simple polygon이 주어짐. I 는 polygon 내부의 격자점 수, B 는 polygon 선분 위 격자점 수, A는 polygon의 넓이라고 할 때, 다음과 같은 식이 성립한다. $A = I + B/2 - 1$
- 홀의 격론 정리 : 이분그래프(L-R)에서, 모든 L을 매칭하는 필요충분 조건은 L에서 임의의 부분집합 S를 골랐을 때, 반드시 (S의 크기) $<=$ (S와 연결되어있는 모든 R의 크기)이다.

- Simpson 공식 (적분) : Simpson 공식, $S_n(f) = \frac{h}{3}[f(x_0) + f(x_n) + 4 \sum f(x_{2i+1}) + 2 \sum f(x_{2i})]$
- $M = \max |f^4(x)|$ 이라고 하면 오차 범위는 최대 $E_n \leq \frac{M(b-a)}{180} h^4$
- 브라마굽타 : 원에 내접하는 사각형의 각 선분의 길이가 a, b, c, d 일 때 사각형의 넓이 $S = \sqrt{(s-a)(s-b)(s-c)(s-d)}$, $s = (a+b+c+d)/2$
- 브레치나이더 : 임의의 사각형의 각 변의 길이를 a, b, c, d 라고 하고, 마주보는 두 각의 합을 2로 나눈 값을 θ 라 하면, $S = \sqrt{(s-a)(s-b)(s-c)(s-d) - abcd \times \cos^2 \theta}$
- 페르마 포인트 : 삼각형의 세 꼭짓점으로부터 거리의 합이 최소가 되는 점 $2\pi/3$ 보다 큰 각이 있으면 그 점이 페르마 포인트, 그렇지 않으면 각 변마다 정삼각형 그런 다음, 정삼각형의 끝점에서 반대쪽 삼각형의 꼭짓점으로 연결한 선분의 교점
- $2\pi/3$ 보다 큰 각이 없으면 거리의 합은 $\sqrt{(a^2 + b^2 + c^2 + 4\sqrt{3}S)/2}$, S 는 넓이
- 오일러 정리: 서로소인 두 정수 a, n 에 대해 $a^{\phi(n)} \equiv 1 \pmod{n}$
- 모든 정수에 대해 $a^n \equiv a^{n-\phi(n)} \pmod{n}$
- $m \geq \log_2 n$ 이면 $a^m \equiv a^{m \% \phi(n) + \phi(n)} \pmod{n}$
- $g^0 + g^1 + g^2 + \dots + g^{p-2} \equiv -1 \pmod{p}$ iff $g = 1$, otherwise 0.
- if $n \equiv 0 \pmod{2}$, then $1^n + 2^n + \dots + (n-1)^n \equiv 0 \pmod{n}$
- Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$$

$$E(n, 0) = E(n, n-1) = 1$$

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

7.9 About Graph Minimum Cut

- N개의 boolean 변수 v_1, \dots, v_n 을 정해서 비용을 최소화하는 문제
- =true인 점은 T, false인 점은 F와 연결되게 분할하는 민컷 문제

- v_i 가 T일 때 비용 발생: i에서 F로 가는 비용 간선
- v_i 가 F일 때 비용 발생: i에서 T로 가는 비용 간선
- v_i 가 T이고 v_j 가 F일 때 비용 발생: i에서 j로 가는 비용 간선
- $v_i \neq v_j$ 일 때 비용 발생: i에서 j로, j에서 i로 가는 비용 간선
- v_i 가 T면 v_j 도 T여야 함: i에서 j로 가는 무한 간선
- v_i 가 F면 v_j 도 F여야 함: j에서 i로 가는 무한 간선

- 5/6변 + v_i 와 v_j 가 달라야 한다는 조건이 있으면 MAX-2SAT
- Maximum Density Subgraph (NEERC'06H, BOJ 3611 팀의 난이도)
 - density $\geq x$ 인 subgraph가 있는지 이분 탐색
 - 정점 N개, 간선 M개, 차수 D_i 개
 - 그래프의 간선마다 용량 1인 양방향 간선 추가
 - 소스에서 정점으로 용량 M, 정점에서 싱크로 용량 $M - D_i + 2x$
 - min cut에서 S와 붙어 있는 애들이 1개 이상이면 x 이상이고, 그제 subgraph의 정점들
 - while($r-l \geq 1.0/(n * n)$) 으로 해야 함. 너무 많이 돌리면 실수 오차

7.10 Matrix with Graph(Kirchhoff, Tutte, LGV)

- Kirchhoff's Theorem : 그래프의 스패닝 트리 개수
 - $m[i][j] := -(i-j \text{ 간선 개수})$ ($i \neq j$)
 - $m[i][i] :=$ 정점 i의 degree
 - res = (m의 첫 번째 행과 첫 번째 열을 없앤 $(n-1) \times (n-1)$ matrix의 행렬식)
- Tutte Matrix : 그래프의 최대 매칭
 - $m[i][j] :=$ 간선 (i, j) 가 없으면 0, 있으면 $i < j$? $r : -r, r \in [0, P)$ 구간의 임의의 정수
 - rank(m)/2가 높은 확률로 최대 매칭
- LGV Theorem: 간선에 가중치 있는 DAG에서 어떤 경로 P의 간선 가중치 곱을 $w(P)$, 모든 $a \rightarrow b$ 경로들의 $w(P)$ 의 합을 $e(a, b)$ 라고 하자. n 개의 시작점 a_i 와 도착점 b_j 가 주어졌을 때, 서로 정점이 겹치지 않는 n

개의 경로로 시작점과 도착점을 일대일 대응시키는 모든 경우에서 $w(P)$ 의 곱의 합은 $\det M(i, j) = e(a_i, b_j)$ 와 같음. 따라서 모든 가중치를 1로 두면 서로소 경로 경우의 수를 구함

7.11 About Graph Matching(Graph with $|V| \leq 500$)

- Game on a Graph** : s에 토큰이 있음. 플레이어는 각자의 턴마다 토큰을 인접한 정점으로 옮기고 못 옮기면 짐.
- s를 포함하지 않는 최대 매칭이 존재함 \leftrightarrow 후공이 이김
- Chinese Postman Problem** : 모든 간선을 방문하는 최소 가중치 Walk를 구하는 문제.
- Floyd를 돌린 다음, 홀수 정점들을 모아서 최소 가중치 매칭 (홀수 정점은 짝수 개 존재)
- Unweighted Edge Cover** : 모든 정점을 덮는 가장 작은(minimum cardinality/weight) 간선 집합을 구하는 문제
- $|V| - |M|$, 길이 3짜리 경로 없음, star graph 여러 개로 구성
- Weighted Edge Cover** : $\text{sum}_{v \in V} (w(v)) - \text{sum}_{(u,v) \in M} (w(u) + w(v) - d(u, v))$, $w(x)$ 는 x와 인접한 간선의 최소 가중치
- NEERC'18 B** : 각 기계마다 2명의 노동자가 다뤄야 하는 문제. 기계마다 두 개의 정점을 만들고 간선으로 연결하면 정답은 $|M| - |기계|$ 임. 정답에 1/2씩 기여한다는 점을 생각해보면 좋음.
- Min Disjoint Cycle Cover** : 정점이 중복되지 않으면서 모든 정점을 덮는 길이 3 이상의 사이클 집합을 찾는 문제.
- 모든 정점은 2개의 서로 다른 간선, 일부 간선은 양쪽 끝점과 매칭되어야 하므로 플로우를 생각할 수 있지만 용량 2짜리 간선에 유량을 1만큼 흘릴 수 있으므로 플로는 불가능.
- 각 정점과 간선을 2개씩((v, v') , $(e_{i,u}, e_{i,v})$)로 복사하자. 모든 간선 $e = (u, v)$ 에 대해 e_u 와 e_v 를 잇는 가중치 w짜리 간선을 만들고(like NEERC18), $(u, e_{i,u}), (u', e_{i,u}), (v, e_{i,v}), (v', e_{i,v})$ 를 연결하는 가중치 0짜리 간선을 만들자. Perfect 매칭이 존재함 \Leftrightarrow Disjoint Cycle Cover 존재.
- 최대 가중치 매칭 찾은 뒤 모든 간선 가중치 합에서 매칭 빼면 됨.
- Two Matching** : 각 정점이 최대 2개의 간선과 인접할 수 있는 최대 가중치 매칭 문제.
- 각 컴포넌트는 정점 하나/경로/사이클이 되어야 함. 모든 서로 다른 정점 쌍에 대해 가중치 0짜리 간선 만들고, 가중치 0짜리 (v, v') 간선 만들면 Disjoint Cycle Cover 문제가 됨. 정점 하나만 있는 컴포넌트는 self-loop, 경로 형태의 컴포넌트는 양쪽 끝점을 연결한다고 생각하면 편함.

7.12 Checklist

- (예비소집) bits/stdc++.h, int128, long double 80bit, avx2 확인
- (예비소집) 스택 메모리(지역 변수, 재귀, 람다 재귀), 제출 파일 크기 확인
- (예비소집) MLE(힙, 스택), stderr 출력 RTE?, 줄 앞뒤 공백 채점 결과
- 비슷한 문제를 풀어본 적이 있는가?
- 단순한 방법에서 시작할 수 있을까? (Brute Force)
- 내가 문제를 푸는 과정을 수식화할 수 있을까? (예제를 직접 해결하면서)
- 문제를 단순화할 수 있을까? / 그림으로 그려볼 수 있을까?
- 수식으로 표현할 수 있을까? / 표현을 분해할 수 있을까?
- 뒤에서부터 생각해서 풀 수 있을까? / 순서를 강제할 수 있을까?
- 특정 형태의 답만을 고려할 수 있을까? (정규화)
- 구간을 통째로 가져간다면: 플로우 + 적당한 자료구조
- $(i, i+1, k, 0), (s, e, 1, w), (N, T, k, 0)$
- $a = b$: a만 이동, b만 이동, 두 개 동시에 이동, 반대로 이동
- 말도 안 되는 것 / 당연하다고 생각한 것 다시 생각해 보기
- Directed MST / Dominator Tree
- 일정 비율 충족 or 2 3개로 모두 커버 : 랜덤
- 확률 : DP, 이분 탐색(NYPC 2019 Finals C)
- 최대/최소 : 이분 탐색, 그리디(Prefix 고정, Exchange Argument), DP(순서 고정)
- 벡셀: 파라미터 순서 변경, min plus convolution, FFT
- signal(SIGSEGV, [](int){_Exit(0);}); converts segfaults into WA.
- SIGABRT(assertion fail), SIGFPE(0div)
- feenableexcept(29) kills problem on NaNs(1), 0div(4), inf(8), denormals(16)