

Contents

1 Basic	1	8.18Rotating SweepLine	21
1.1 Shell Script	1	8.19Half Plane Intersection	21
1.2 Default Code	1	8.20Minkowski Sum	21
1.3 Increase Stack Size	1	8.21Delaunay Triangulation	21
1.4 Debug Macro	1	8.22Triangulation Voronoi	22
1.5 Stress Test Shell*	1		
1.6 Pragma / FastIO	2	9 Else	22
1.7 Divide*	2	9.1 Bit Hack	22
2 Data Structure	2	9.2 Dynamic Programming Condition	22
2.1 Leftist Tree	2	9.2.1 Totally Monotone (Concave/Convex)	22
2.2 Splay Tree	2	9.2.2 Monge Condition (Concave/Convex)	22
2.3 Link Cut Tree	2	9.2.3 Optimal Split Point	22
2.4 Treap	3	9.3 Slope Trick	22
2.5 2D Segment Tree*	3	9.4 Manhattan MST	23
2.6 Zkw*	4	9.5 Dynamic MST	23
2.7 Chtholly Tree*	4	9.6 ALL LCS	24
3 Flow / Matching	5	9.7 Hilbert Curve	24
3.1 Dinic	5	9.8 Pbds	24
3.2 Min Cost Max Flow	5	9.9 Random	24
3.3 Kuhn Munkres	5	9.10Smawk Algorithm	24
3.4 SW Min Cut	6	9.11Two Dimension Add Sum*	25
3.5 Gomory Hu Tree	6	9.12Matroid Intersection	25
3.6 Blossom	6	9.13Python Misc	25
3.7 Weighted Blossom	7		
3.8 Flow Model	8		
4 Graph	8		
4.1 Heavy-Light Decomposition	8		
4.2 Centroid Decomposition	9		
4.3 Edge BCC	9		
4.4 Block Cut Tree	9		
4.5 SCC / 2SAT	9		
4.6 Negative Cycle*	9		
4.7 Virtual Tree	10		
4.8 Directed MST	10		
4.9 Dominator Tree	10		
5 String	11		
5.1 Aho-Corasick Automaton	11		
5.2 KMP Algorithm	11		
5.3 Z Algorithm	11		
5.4 Manacher	11		
5.5 Suffix Array	11		
5.6 SAIS	12		
5.7 Suffix Automaton	12		
5.8 Minimum Rotation	12		
5.9 Palindrome Tree	12		
5.10Main Lorentz	13		
6 Math	13		
6.1 Fraction*	13		
6.2 Miller Rabin / Pollard Rho	13		
6.3 Ext GCD	13		
6.4 PiCount	14		
6.5 Linear Function Mod Min	14		
6.6 Determinant	14		
6.7 Floor Sum	14		
6.8 Quadratic Residue	14		
6.9 Simplex	15		
6.10Berlekamp Massey	15		
6.11Linear Programming Construction	15		
6.12Euclidean	15		
6.13Theorem	16		
6.14Estimation	16		
6.15General Purpose Numbers	16		
6.16Tips for Generating Funtion	16		
7 Polynomial	16		
7.1 Number Theoretic Transform	16		
7.2 Primes	17		
7.3 Polynomial Operations	17		
7.4 Fast Linear Recursion	18		
7.5 Fast Walsh Transform	18		
8 Geometry	18		
8.1 Basic	18		
8.2 Heart	19		
8.3 External Bisector	19		
8.4 Intersection of Segments	19		
8.5 Intersection of Circle and Line	19		
8.6 Intersection of Circles	19		
8.7 Intersection of Polygon and Circle	19		
8.8 Tangent Lines of Circle and Point	19		
8.9 Tangent Lines of Circles	19		
8.10Point In Convex	20		
8.11Point Segment Distance	20		
8.12Convex Hull	20		
8.13Convex Hull Distance	20		
8.14Minimum Enclosing Circle	20		
8.15Union of Circles	20		
8.16Polar Angle Sort	20		
8.17Rotating Caliper	20		

1 Basic

1.1 Shell Script

```
g++ -std=c++17 -DABS -Wall -Wextra -Wshadow $1.cpp -o
    $1 && ./$1
for i in {A..J}; do cp tem.cpp $i.cpp; done;
```

1.2 Default Code

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
#define pb push_back
#define pii pair<int, int>
#define all(a) a.begin(), a.end()
#define sz(a) ((int)a.size())
```

1.3 Increase Stack Size

```
const int size = 256 << 20;
register long rsp asm("rsp");
char *p = (char*)malloc(size) + size, *bak = (char*)rsp
;
__asm__ ("movq %0, %%rsp\n"::"r"(p));
// main
__asm__ ("movq %0, %%rsp\n"::"r"(bak));
```

1.4 Debug Macro

```
void db() {cout << endl;}
template <typename T, typename ...U> void db(T i, U ...
    j) {
    cout << i << ' ', db(j...);
}
#define test(x...) db("[ " + string(x) + " ]", x)
```

1.5 Stress Test Shell*

```
#!/usr/bin/env bash
g++ $1.cpp -o $1
g++ $2.cpp -o $2
g++ $3.cpp -o $3
for i in {1..100}; do
    ./$3 > input.txt
    # st=$(date +%s%N)
    ./$1 < input.txt > output1.txt
    # echo "$(((date +%s%N) - $st)/1000000))ms"
    ./$2 < input.txt > output2.txt
    if cmp --silent -- "output1.txt" "output2.txt" ; then
        continue
    fi
    echo Input:
    cat input.txt
    echo Your Output:
    cat output1.txt
    echo Correct Output:
    cat output2.txt
    exit 1
done
echo OK!
./stress.sh main good gen
```

1.6 Pragma / FastIO

```
#pragma GCC optimize("Ofast,inline,unroll-loops")
#pragma GCC target("bmi,bmi2,lzcnt,popcnt,avx2")

#include<unistd.h>
char OB[65536]; int OP;
inline char RC() {
    static char buf[65536], *p = buf, *q = buf;
    return p == q && (q = (p = buf) + read(0, buf, 65536)
        ) == buf ? -1 : *p++;
}
inline int R() {
    static char c;
    while((c = RC()) < '0'); int a = c ^ '0';
    while((c = RC()) >= '0') a *= 10, a += c ^ '0';
    return a;
}
inline void W(int n) {
    static char buf[12], p;
    if (n == 0) OB[OP++] = '0'; p = 0;
    while (n) buf[p++] = '0' + (n % 10), n /= 10;
    for (--p; p >= 0; --p) OB[OP++] = buf[p];
    if (OP > 65520) write(1, OB, OP), OP = 0;
}
```

1.7 Divide*

```
ll divdown(ll a, ll b) {
    return a / b - (a < 0 && a % b);
}
ll divup(ll a, ll b) {
    return a / b + (a > 0 && a % b);
}
a / b < x -> divdown(a, b) + 1 <= x
a / b <= x -> divup(a, b) <= x
x < a / b -> x <= divup(a, b) - 1
x <= a / b -> x <= divdown(a, b)
```

2 Data Structure

2.1 Leftist Tree

```
struct node {
    ll rk, data, sz, sum;
    node *l, *r;
    node(ll k) : rk(0), data(k), sz(1), l(0), r(0), sum(k) {}
};
ll sz(node *p) { return p ? p->sz : 0; }
ll rk(node *p) { return p ? p->rk : -1; }
ll sum(node *p) { return p ? p->sum : 0; }
node *merge(node *a, node *b) {
    if (!a || !b) return a ? a : b;
    if (a->data < b->data) swap(a, b);
    a->r = merge(a->r, b);
    if (rk(a->r) > rk(a->l)) swap(a->r, a->l);
    a->rk = rk(a->r) + 1, a->sz = sz(a->l) + sz(a->r) + 1;
    a->sum = sum(a->l) + sum(a->r) + a->data;
    return a;
}
void pop(node *&o) {
    node *tmp = o;
    o = merge(o->l, o->r);
    delete tmp;
}
```

2.2 Splay Tree

```
struct Splay {
    int pa[N], ch[N][2], sz[N], rt, _id;
    ll v[N];
    Splay() {}
    void init() {
        rt = 0, pa[0] = ch[0][0] = ch[0][1] = -1;
        sz[0] = 1, v[0] = inf;
    }
    int newnode(int p, int x) {
        int id = _id++;
        v[id] = x, pa[id] = p;
    }
```

```
    ch[id][0] = ch[id][1] = -1, sz[id] = 1;
    return id;
}
void rotate(int i) {
    int p = pa[i], x = ch[p][1] == i, gp = pa[p], c =
        ch[i][!x];
    sz[p] -= sz[i], sz[i] += sz[p];
    if (~c) sz[p] += sz[c], pa[c] = p;
    ch[p][x] = c, pa[p] = i;
    pa[i] = gp, ch[i][!x] = p;
    if (~gp) ch[gp][ch[gp][1] == p] = i;
}
void splay(int i) {
    while (~pa[i]) {
        int p = pa[i];
        if (~pa[p]) rotate(ch[pa[p]][1] == p ^ ch[p][1]
            == i ? i : p);
        rotate(i);
    }
    rt = i;
}
int lower_bound(int x) {
    int i = rt, last = -1;
    while (true) {
        if (v[i] == x) return splay(i), i;
        if (v[i] > x) {
            last = i;
            if (ch[i][0] == -1) break;
            i = ch[i][0];
        }
        else {
            if (ch[i][1] == -1) break;
            i = ch[i][1];
        }
    }
    splay(i);
    return last; // -1 if not found
}
void insert(int x) {
    int i = lower_bound(x);
    if (i == -1) {
        // assert(ch[rt][1] == -1);
        int id = newnode(rt, x);
        ch[rt][1] = id, ++sz[rt];
        splay(id);
    }
    else if (v[i] != x) {
        splay(i);
        int id = newnode(rt, x), c = ch[rt][0];
        ch[rt][0] = id;
        ch[id][0] = c;
        if (~c) pa[c] = id, sz[id] += sz[c];
        ++sz[rt];
        splay(id);
    }
}
```

2.3 Link Cut Tree

```
// weighted subtree size, weighted path max
struct LCT {
    int ch[N][2], pa[N], v[N], sz[N], sz2[N], w[N], mx[N],
        _id;
    // sz := sum of v in splay, sz2 := sum of v in
    // virtual subtree
    // mx := max w in splay
    bool rev[N];
    LCT() : _id(1) {}
    int newnode(int _v, int _w) {
        int x = _id++;
        ch[x][0] = ch[x][1] = pa[x] = 0;
        v[x] = sz[x] = _v;
        sz2[x] = 0;
        w[x] = mx[x] = _w;
        rev[x] = false;
        return x;
    }
    void pull(int i) {
        sz[i] = v[i] + sz2[i];
        mx[i] = w[i];
        if (ch[i][0])
```

```

        sz[i] += sz[ch[i][0]], mx[i] = max(mx[i], mx[ch[i][0]]);
        if (ch[i][1])
            sz[i] += sz[ch[i][1]], mx[i] = max(mx[i], mx[ch[i][1]]);
    }
    void push(int i) {
        if (rev[i]) reverse(ch[i][0]), reverse(ch[i][1]),
            rev[i] = false;
    }
    void reverse(int i) {
        if (!i) return;
        swap(ch[i][0], ch[i][1]);
        rev[i] ^= true;
    }
    bool isrt(int i) { // rt of splay
        if (!pa[i]) return true;
        return ch[pa[i]][0] != i && ch[pa[i]][1] != i;
    }
    void rotate(int i) {
        int p = pa[i], x = ch[p][1] == i, c = ch[i][!x], gp
            = pa[p];
        if (ch[gp][0] == p) ch[gp][0] = i;
        else if (ch[gp][1] == p) ch[gp][1] = i;
        pa[i] = gp, ch[i][!x] = p, pa[p] = i;
        ch[p][x] = c, pa[c] = p;
        pull(p), pull(i);
    }
    void splay(int i) {
        vector<int> anc;
        anc.push_back(i);
        while (!isrt(anc.back())) anc.push_back(pa[anc.back()]);
        while (!anc.empty()) push(anc.back()), anc.pop_back();
        while (!isrt(i)) {
            int p = pa[i];
            if (!isrt(p)) rotate(ch[p][1] == i ^ ch[pa[p]][1]
                == p ? i : p);
            rotate(i);
        }
    }
    void access(int i) {
        int last = 0;
        while (i) {
            splay(i);
            if (ch[i][1])
                sz2[i] += sz[ch[i][1]];
            sz2[i] -= sz[last];
            ch[i][1] = last;
            pull(i), last = i, i = pa[i];
        }
    }
    void makert(int i) {
        access(i), splay(i), reverse(i);
    }
    void link(int i, int j) {
        // assert(findrt(i) != findrt(j));
        makert(i);
        makert(j);
        pa[i] = j;
        sz2[j] += sz[i];
        pull(j);
    }
    void cut(int i, int j) {
        makert(i), access(j), splay(i);
        // assert(sz[i] == 2 && ch[i][1] == j);
        ch[i][1] = pa[j] = 0, pull(i);
    }
    int findrt(int i) {
        access(i), splay(i);
        while (ch[i][0]) push(i), i = ch[i][0];
        splay(i);
        return i;
    }
};

```

2.4 Treap

```

struct node {
    int data, sz;
    node *l, *r;

```

```

    node(int k) : data(k), sz(1), l(0), r(0) {}
    void up() {
        sz = 1;
        if (l) sz += l->sz;
        if (r) sz += r->sz;
    }
    void down() {}
};
int sz(node *a) { return a ? a->sz : 0; }
node *merge(node *a, node *b) {
    if (!a || !b) return a ? a : b;
    if (rand() % (sz(a) + sz(b)) < sz(a))
        return a->down(), a->r = merge(a->r, b), a->up(), a;
    return b->down(), b->l = merge(a, b->l), b->up(), b;
}
void split(node *o, node *&a, node *&b, int k) {
    if (!o) return a = b = 0, void();
    o->down();
    if (o->data <= k)
        a = o, split(o->r, a->r, b, k), a->up();
    else b = o, split(o->l, a, b->l, k), b->up();
}
void split2(node *o, node *&a, node *&b, int k) {
    if (sz(o) <= k) return a = o, b = 0, void();
    o->down();
    if (sz(o->l) + 1 <= k)
        a = o, split2(o->r, a->r, b, k - sz(o->l) - 1);
    else b = o, split2(o->l, a, b->l, k);
    o->up();
}
node *kth(node *o, int k) {
    if (k <= sz(o->l)) return kth(o->l, k);
    if (k == sz(o->l) + 1) return o;
    return kth(o->r, k - sz(o->l) - 1);
}
int Rank(node *o, int key) {
    if (!o) return 0;
    if (o->data < key)
        return sz(o->l) + 1 + Rank(o->r, key);
    else return Rank(o->l, key);
}
bool erase(node *&o, int k) {
    if (!o) return 0;
    if (o->data == k) {
        node *t = o;
        o->down(), o = merge(o->l, o->r);
        delete t;
        return 1;
    }
    node *&t = k < o->data ? o->l : o->r;
    return erase(t, k) ? o->up(), 1 : 0;
}
void insert(node *&o, int k) {
    node *a, *b;
    o->down(), split(o, a, b, k),
    o = merge(a, merge(new node(k), b));
    o->up();
}
void interval(node *&o, int l, int r) {
    node *a, *b, *c; // [l, r)
    o->down();
    split2(o, a, b, l), split2(b, b, c, r - l);
    // operate
    o = merge(a, merge(b, c)), o->up();
}

```

2.5 2D Segment Tree*

```

// 2D range add, range sum in Log^2
struct seg {
    int l, r;
    ll sum, lz;
    seg *ch[2]{};
    seg(int _l, int _r) : l(_l), r(_r), sum(0), lz(0) {}
    void push() {
        if (lz) ch[0]->add(l, r, lz), ch[1]->modify(l, r,
            lz), lz = 0;
    }
    void pull() {sum = ch[0]->sum + ch[1]->sum;}
    void add(int _l, int _r, ll d) {
        if (_l <= l && r <= _r) {

```

```

    sum += d * (r - 1);
    lz += d;
    return;
}
if (!ch[0]) ch[0] = new seg(1, 1 + r >> 1), ch[1] =
    new seg(1 + r >> 1, r);
push();
if (_l < 1 + r >> 1) ch[0]->add(_l, _r, d);
if (1 + r >> 1 < _r) ch[1]->add(_l, _r, d);
pull();
}
ll qsum(int _l, int _r) {
    if (_l <= 1 && r <= _r) return sum;
    if (!ch[0]) return lz * (min(r, _r) - max(1, _l));
    push();
    ll res = 0;
    if (_l < 1 + r >> 1) res += ch[0]->qsum(_l, _r);
    if (1 + r >> 1 < _r) res += ch[1]->qsum(_l, _r);
    return res;
}
};
struct seg2 {
    int l, r;
    seg v, lz;
    seg2 *ch[2];
    seg2(int _l, int _r) : l(_l), r(_r), v(0, N), lz(0, N) {
        if (1 < r - 1) ch[0] = new seg2(1, 1 + r >> 1), ch[1] =
            new seg2(1 + r >> 1, r);
    }
    void add(int _l, int _r, int _l2, int _r2, ll d) {
        v.add(_l2, _r2, d * (min(r, _r) - max(1, _l)));
        if (_l <= 1 && r <= _r) {
            lz.add(_l2, _r2, d);
            return;
        }
        if (_l < 1 + r >> 1) ch[0]->add(_l, _r, _l2, _r2, d);
        if (1 + r >> 1 < _r) ch[1]->add(_l, _r, _l2, _r2, d);
    }
    ll qsum(int _l, int _r, int _l2, int _r2) {
        ll res = v.qsum(_l2, _r2);
        if (_l <= 1 && r <= _r) return res;
        res += lz.qsum(_l2, _r2) * (min(r, _r) - max(1, _l));
        if (_l < 1 + r >> 1) res += ch[0]->query(_l, _r, _l2, _r2);
        if (1 + r >> 1 < _r) res += ch[1]->query(_l, _r, _l2, _r2);
        return res;
    }
};

```

2.6 Zkw*

```

ll mx[N << 1], sum[N << 1], lz[N << 1];
void add(int l, int r, ll d) { // [L, r), 0-based
    int len = 1, cntl = 0, cntr = 0;
    for (l += N, r += N + 1; l ^ r ^ 1; l >>= 1, r >>= 1,
        len <= 1) {
        sum[l] += cntl * d, sum[r] += cntr * d;
        if (len > 1) {
            mx[l] = max(mx[l << 1], mx[l << 1 | 1]) + lz[l];
            mx[r] = max(mx[r << 1], mx[r << 1 | 1]) + lz[r];
        }
        if (~l & 1)
            sum[l ^ 1] += d * len, mx[l ^ 1] += d, lz[l ^ 1]
                += d, cntl += len;
        if (r & 1)
            sum[r ^ 1] += d * len, mx[r ^ 1] += d, lz[r ^ 1]
                += d, cntr += len;
    }
    sum[l] += cntl * d, sum[r] += cntr * d;
    if (len > 1) {
        mx[l] = max(mx[l << 1], mx[l << 1 | 1]) + lz[l];
        mx[r] = max(mx[r << 1], mx[r << 1 | 1]) + lz[r];
    }
    cntl += cntr;
    for (l >>= 1; l; l >>= 1) {
        sum[l] += cntl * d;
        mx[l] = max(mx[l << 1], mx[l << 1 | 1]) + lz[l];
    }
}

```

```

}
}
ll qsum(int l, int r) {
    ll res = 0, len = 1, cntl = 0, cntr = 0;
    for (l += N, r += N + 1; l ^ r ^ 1; l >>= 1, r >>= 1,
        len <= 1) {
        res += cntl * lz[l] + cntr * lz[r];
        if (~l & 1) res += sum[l ^ 1], cntl += len;
        if (r & 1) res += sum[r ^ 1], cntr += len;
    }
    res += cntl * lz[l] + cntr * lz[r];
    cntl += cntr;
    for (l >>= 1; l; l >>= 1) res += cntl * lz[l];
    return res;
}
ll qmax(int l, int r) {
    ll maxl = -INF, maxr = -INF;
    for (l += N, r += N + 1; l ^ r ^ 1; l >>= 1, r >>= 1)
        {
            maxl += lz[l], max[r] += lz[r];
            if (~l & 1) maxl = max(maxl, mx[l ^ 1]);
            if (r & 1) maxr = max(maxr, mx[r ^ 1]);
        }
    maxl = max(maxl + lz[l], maxr + lz[r]);
    for (l >>= 1; l; l >>= 1) maxl += lz[l];
    return maxl;
}

```

2.7 Chtholly Tree*

```

struct ChthollyTree {
    struct interval {
        int l, r;
        ll v;
        interval(int _l, int _r, ll _v) : l(_l), r(_r), v(_v) {}
    };
    struct cmp {
        bool operator () (const interval &a, const interval
            & b) const {
            return a.l < b.l;
        }
    };
    set <interval, cmp> s;
    vector <interval> split(int l, int r) {
        // split into [0, l), [l, r), [r, n) and return [L,
        // r)
        vector <interval> del, ans, re;
        auto it = s.lower_bound(interval(l, -1, 0));
        if (it != s.begin() && (it == s.end() || l < it->l))
            {
                --it;
                del.pb(*it);
                if (r < it->r) {
                    re.pb(interval(it->l, l, it->v));
                    ans.pb(interval(l, r, it->v));
                    re.pb(interval(r, it->r, it->v));
                } else {
                    re.pb(interval(it->l, l, it->v));
                    ans.pb(interval(l, it->r, it->v));
                }
                ++it;
            }
        for (; it != s.end() && it->r <= r; ++it) {
            ans.pb(*it);
            del.pb(*it);
        }
        if (it != s.end() && it->l < r) {
            del.pb(*it);
            ans.pb(interval(it->l, r, it->v));
            re.pb(interval(r, it->r, it->v));
        }
        for (interval &i : del)
            s.erase(i);
        for (interval &i : re)
            s.insert(i);
        return ans;
    }
    void merge(vector <interval> a) {
        for (interval &i : a)
            s.insert(i);
    }
}

```

};

3 Flow / Matching

3.1 Dinic

```

struct Dinic { // 0-base
    struct edge {
        int to, cap, flow, rev;
    };
    vector<edge> adj[N];
    int s, t, dis[N], cur[N], n;
    int dfs(int u, int cap) {
        if (u == t || !cap) return cap;
        for (int &i = cur[u]; i < (int)adj[u].size(); ++i)
        {
            edge &e = adj[u][i];
            if (dis[e.to] == dis[u] + 1 && e.flow != e.cap) {
                int df = dfs(e.to, min(e.cap - e.flow, cap));
                if (df) {
                    e.flow += df;
                    adj[e.to][e.rev].flow -= df;
                    return df;
                }
            }
        }
        dis[u] = -1;
        return 0;
    }
    bool bfs() {
        fill_n(dis, n, -1);
        queue<int> q;
        q.push(s), dis[s] = 0;
        while (!q.empty()) {
            int tmp = q.front();
            q.pop();
            for (auto &u : adj[tmp])
                if (!~dis[u.to] && u.flow != u.cap) {
                    q.push(u.to);
                    dis[u.to] = dis[tmp] + 1;
                }
        }
        return dis[t] != -1;
    }
    int maxflow(int _s, int _t) {
        s = _s, t = _t;
        int flow = 0, df;
        while (bfs()) {
            fill_n(cur, n, 0);
            while ((df = dfs(s, INF))) flow += df;
        }
        return flow;
    }
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) adj[i].clear();
    }
    void reset() {
        for (int i = 0; i < n; ++i)
            for (auto &j : adj[i]) j.flow = 0;
    }
    void add_edge(int u, int v, int cap) {
        adj[u].pb(edge{v, cap, 0, (int)adj[v].size()});
        adj[v].pb(edge{u, 0, 0, (int)adj[u].size() - 1});
    }
};

```

3.2 Min Cost Max Flow

```

template <typename T1, typename T2>
struct MCMF { // T1 -> flow, T2 -> cost, 0-based
    const T1 INF1 = 1 << 30;
    const T2 INF2 = 1 << 30;
    struct edge {
        int v; T1 f; T2 c;
    } E[M << 1];
    vector <int> adj[N];
    T2 dis[N], pot[N];
    int rt[N], vis[N], n, m, s, t;
    bool SPFA() {
        fill_n(rt, n, -1), fill_n(dis, n, INF2);

```

```

        fill_n(vis, n, false);
        queue <int> q;
        q.push(s), dis[s] = 0, vis[s] = true;
        while (!q.empty()) {
            int v = q.front(); q.pop();
            vis[v] = false;
            for (int id : adj[v]) if (E[id].f > 0 && dis[E[id].v] > dis[v] + E[id].c + pot[v] - pot[E[id].v]) {
                dis[E[id].v] = dis[v] + E[id].c + pot[v] - pot[E[id].v];
                rt[E[id].v] = id;
                if (!vis[E[id].v]) vis[E[id].v] = true, q.push(E[id].v);
            }
        }
        return dis[t] != INF2;
    }
    bool dijkstra() {
        fill_n(rt, n, -1), fill_n(dis, n, INF2);
        priority_queue <pair <T2, int>, vector <pair <T2, int>> pq;
        dis[s] = 0, pq.emplace(dis[s], s);
        while (!pq.empty()) {
            auto [d, v] = pq.top(); pq.pop();
            if (dis[v] < d) continue;
            for (int id : adj[v]) if (E[id].f > 0 && dis[E[id].v] > dis[v] + E[id].c + pot[v] - pot[E[id].v]) {
                dis[E[id].v] = dis[v] + E[id].c + pot[v] - pot[E[id].v];
                rt[E[id].v] = id;
                pq.emplace(dis[E[id].v], E[id].v);
            }
        }
        return dis[t] != INF2;
    }
    pair <T1, T2> solve(int _s, int _t) {
        s = _s, t = _t, fill_n(pot, n, 0);
        T1 flow = 0; T2 cost = 0;
        bool fr = true;
        while ((fr ? SPFA() : dijkstra())) {
            for (int i = 0; i < n; i++) {
                dis[i] += pot[i] - pot[s];
            }
            T1 add = INF1;
            for (int i = t; i != s; i = E[rt[i] ^ 1].v) {
                add = min(add, E[rt[i]].f);
            }
            for (int i = t; i != s; i = E[rt[i] ^ 1].v) {
                E[rt[i]].f -= add, E[rt[i] ^ 1].f += add;
            }
            flow += add, cost += add * dis[t];
            fr = false;
            for (int i = 0; i < n; ++i) swap(dis[i], pot[i]);
        }
        return make_pair(flow, cost);
    }
    void init(int _n) {
        n = _n, m = 0;
        for (int i = 0; i < n; ++i) adj[i].clear();
    }
    void reset() {
        for (int i = 0; i < m; ++i) E[i].f = 0;
    }
    void add_edge(int u, int v, T1 f, T2 c) {
        adj[u].pb(m), E[m++] = {v, f, c};
        adj[v].pb(m), E[m++] = {u, 0, -c};
    }
};

```

3.3 Kuhn Munkres

```

template <typename T>
struct KM { // 0-based
    T w[N][N], hl[N], hr[N], slk[N];
    T fl[N], fr[N], pre[N]; int n;
    bool vl[N], vr[N];
    const T INF = 1e9;
    queue <int> q;
    KM(int _n) : n(_n) {
        for (int i = 0; i < n; ++i) for (int j = 0; j < n; ++j)
            w[i][j] = -INF;

```

```

}
void add_edge(int a, int b, int wei) {
    w[a][b] = wei;
}
bool check(int x) {
    if (v1[x] = 1, ~f1[x]) return q.push(f1[x]), vr[f1[x]] = 1;
    while (~x) swap(x, fr[f1[x] = pre[x]]);
    return 0;
}
void bfs(int s) {
    fill(slk, slk + n, INF), fill(v1, v1 + n, 0), fill(vr, vr + n, 0);
    q.push(s), vr[s] = 1;
    while (1) {
        T d;
        while (!q.empty()) {
            int y = q.front(); q.pop();
            for (int x = 0; x < n; ++x)
                if (!v1[x] && slk[x] >= (d = h1[x] + hr[y] - w[x][y]))
                    if (pre[x] = y, d) slk[x] = d;
                    else if (!check(x)) return;
        }
        d = INF;
        for (int x = 0; x < n; ++x)
            if (!v1[x] && d > slk[x]) d = slk[x];
        for (int x = 0; x < n; ++x) {
            if (v1[x]) h1[x] += d;
            else slk[x] -= d;
            if (vr[x]) hr[x] -= d;
        }
        for (int x = 0; x < n; ++x) if (!v1[x] && !slk[x] && !check(x)) return;
    }
}
T solve() {
    fill(f1, f1 + n, -1), fill(fr, fr + n, -1), fill(hr, hr + n, 0);
    for (int i = 0; i < n; ++i) h1[i] = *max_element(w[i], w[i] + n);
    for (int i = 0; i < n; ++i) bfs(i);
    T res = 0;
    for (int i = 0; i < n; ++i) res += w[i][f1[i]];
    return res;
}
};

```

3.4 SW Min Cut

```

template <typename T>
struct SW { // 0-based
    T g[N][N], sum[N]; int n;
    bool vis[N], dead[N];
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; ++i) fill(g[i], g[i] + n, 0);
        fill(dead, dead + n, false);
    }
    void add_edge(int u, int v, T w) {
        g[u][v] += w, g[v][u] += w;
    }
    T solve() {
        T ans = 1 << 30;
        for (int round = 0; round + 1 < n; ++round) {
            fill(vis, vis + n, false), fill(sum, sum + n, 0);
            int num = 0, s = -1, t = -1;
            while (num < n - round) {
                int now = -1;
                for (int i = 0; i < n; ++i) if (!vis[i] && !dead[i]) {
                    if (now == -1 || sum[now] < sum[i]) now = i;
                }
                s = t, t = now;
                vis[now] = true, num++;
                for (int i = 0; i < n; ++i) if (!vis[i] && !dead[i]) {
                    sum[i] += g[now][i];
                }
            }
        }
    }
};

```

```

    ans = min(ans, sum[t]);
    for (int i = 0; i < n; ++i) {
        g[i][s] += g[i][t];
        g[s][i] += g[t][i];
    }
    dead[t] = true;
}
return ans;
}
};

```

3.5 Gomory Hu Tree

```

vector <array <int, 3>> GomoryHu(vector <vector <pii>>
    adj, int n) {
    // Tree edge min -> mincut (0-based)
    Dinic flow(n);
    for (int i = 0; i < n; ++i) for (auto [j, w] : adj[i])
        flow.add_edge(i, j, w);
    flow.record();
    vector <array <int, 3>> ans;
    vector <int> rt(n);
    for (int i = 0; i < n; ++i) rt[i] = 0;
    for (int i = 1; i < n; ++i) {
        int t = rt[i];
        flow.reset(); // clear flows on all edge
        ans.push_back({i, t, flow.solve(i, t)});
        flow.runbfs(i);
        for (int j = i + 1; j < n; ++j) if (rt[j] == t &&
            flow.vis[j]) {
            rt[j] = i;
        }
    }
    return ans;
}

```

3.6 Blossom

```

struct Matching { // 0-based
    int fa[N], pre[N], match[N], s[N], v[N], n, tk;
    vector <int> g[N];
    queue <int> q;
    Matching(int _n) : n(_n), tk(0) {
        for (int i = 0; i <= n; ++i) match[i] = pre[i] = n;
        for (int i = 0; i < n; ++i) g[i].clear();
    }
    void add_edge(int u, int v) {
        g[u].push_back(v), g[v].push_back(u);
    }
    int Find(int u) {
        return u == fa[u] ? u : fa[u] = Find(fa[u]);
    }
    int lca(int x, int y) {
        tk++;
        x = Find(x), y = Find(y);
        for (; ; swap(x, y)) {
            if (x != n) {
                if (v[x] == tk) return x;
                v[x] = tk;
                x = Find(pre[match[x]]);
            }
        }
    }
    void blossom(int x, int y, int l) {
        while (Find(x) != l) {
            pre[x] = y, y = match[x];
            if (s[y] == 1) q.push(y), s[y] = 0;
            if (fa[x] == x) fa[x] = l;
            if (fa[y] == y) fa[y] = l;
            x = pre[y];
        }
    }
    bool bfs(int r) {
        for (int i = 0; i <= n; ++i) fa[i] = i, s[i] = -1;
        while (!q.empty()) q.pop();
        q.push(r);
        s[r] = 0;
        while (!q.empty()) {
            int x = q.front(); q.pop();
            for (int u : g[x]) {
                if (s[u] == -1) {

```



```

    pre[u] = x, s[u] = 1;
    if (match[u] == n) {
        for (int a = u, b = x, last; b != n; a = last, b = pre[a])
            last = match[b], match[b] = a, match[a] = b;
        return true;
    }
    q.push(match[u]);
    s[match[u]] = 0;
} else if (!s[u] && Find(u) != Find(x)) {
    int l = lca(u, x);
    blossom(x, u, l);
    blossom(u, x, l);
}
}
return false;
}
int solve() {
    int res = 0;
    for (int x = 0; x < n; ++x) {
        if (match[x] == n) res += bfs(x);
    }
    return res;
}
};

```

3.7 Weighted Blossom

```

struct WeightGraph { // 1-based
    static const int inf = INT_MAX;
    static const int maxn = 514;
    struct edge {
        int u, v, w;
        edge(){}
        edge(int u, int v, int w): u(u), v(v), w(w) {}
    };
    int n, n_x;
    edge g[maxn * 2][maxn * 2];
    int lab[maxn * 2];
    int match[maxn * 2], slack[maxn * 2], st[maxn * 2],
        pa[maxn * 2];
    int flo_from[maxn * 2][maxn + 1], S[maxn * 2], vis[
        maxn * 2];
    vector<int> flo[maxn * 2];
    queue<int> q;
    int e_delta(const edge &e) { return lab[e.u] + lab[e.
        v] - g[e.u][e.v].w * 2; }
    void update_slack(int u, int x) { if (!slack[x] ||
        e_delta(g[u][x]) < e_delta(g[slack[x]][x])) slack
        [x] = u; }
    void set_slack(int x) {
        slack[x] = 0;
        for (int u = 1; u <= n; ++u)
            if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
                update_slack(u, x);
    }
    void q_push(int x) {
        if (x <= n) q.push(x);
        else for (size_t i = 0; i < flo[x].size(); i++)
            q_push(flo[x][i]);
    }
    void set_st(int x, int b) {
        st[x] = b;
        if (x > n) for (size_t i = 0; i < flo[x].size(); ++
            i) set_st(flo[x][i], b);
    }
    int get_pr(int b, int xr) {
        int pr = find(flo[b].begin(), flo[b].end(), xr) -
            flo[b].begin();
        if (pr % 2 == 1) {
            reverse(flo[b].begin() + 1, flo[b].end());
            return (int)flo[b].size() - pr;
        }
        return pr;
    }
    void set_match(int u, int v) {
        match[u] = g[u][v].v;
        if (u <= n) return;
        edge e = g[u][v];
        int xr = flo_from[u][e.u], pr = get_pr(u, xr);

```

```

        for (int i = 0; i < pr; ++i) set_match(flo[u][i],
            flo[u][i ^ 1]);
        set_match(xr, v);
        rotate(flo[u].begin(), flo[u].begin() + pr, flo[u].
            end());
    }
    void augment(int u, int v) {
        for (; ; ) {
            int xnv = st[match[u]];
            set_match(u, v);
            if (!xnv) return;
            set_match(xnv, st[pa[xnv]]);
            u = st[pa[xnv]], v = xnv;
        }
    }
    int get_lca(int u, int v) {
        static int t = 0;
        for (++t; u || v; swap(u, v)) {
            if (u == 0) continue;
            if (vis[u] == t) return u;
            vis[u] = t;
            u = st[match[u]];
            if (u) u = st[pa[u]];
        }
        return 0;
    }
    void add_blossom(int u, int lca, int v) {
        int b = n + 1;
        while (b <= n_x && st[b]) ++b;
        if (b > n_x) ++n_x;
        lab[b] = 0, S[b] = 0;
        match[b] = match[lca];
        flo[b].clear();
        flo[b].push_back(lca);
        for (int x = u, y; x != lca; x = st[pa[y]])
            flo[b].push_back(x), flo[b].push_back(y = st[
                match[x]]), q_push(y);
        reverse(flo[b].begin() + 1, flo[b].end());
        for (int x = v, y; x != lca; x = st[pa[y]])
            flo[b].push_back(x), flo[b].push_back(y = st[
                match[x]]), q_push(y);
        set_st(b, b);
        for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].
            w = 0;
        for (int x = 1; x <= n; ++x) flo_from[b][x] = 0;
        for (size_t i = 0; i < flo[b].size(); ++i) {
            int xs = flo[b][i];
            for (int x = 1; x <= n_x; ++x)
                if (g[b][x].w == 0 || e_delta(g[xs][x]) <
                    e_delta(g[b][x]))
                    g[b][x] = g[xs][x], g[x][b] = g[x][xs];
            for (int x = 1; x <= n; ++x)
                if (flo_from[xs][x]) flo_from[b][x] = xs;
        }
        set_slack(b);
    }
    void expand_blossom(int b) {
        for (size_t i = 0; i < flo[b].size(); ++i)
            set_st(flo[b][i], flo[b][i]);
        int xr = flo_from[b][g[b][pa[b]].u], pr = get_pr(b,
            xr);
        for (int i = 0; i < pr; i += 2) {
            int xs = flo[b][i], xns = flo[b][i + 1];
            pa[xs] = g[xns][xs].u;
            S[xs] = 1, S[xns] = 0;
            slack[xs] = 0, set_slack(xns);
            q_push(xns);
        }
        S[xr] = 1, pa[xr] = pa[b];
        for (size_t i = pr + 1; i < flo[b].size(); ++i) {
            int xs = flo[b][i];
            S[xs] = -1, set_slack(xs);
        }
        st[b] = 0;
    }
    bool on_found_edge(const edge &e) {
        int u = st[e.u], v = st[e.v];
        if (S[v] == -1) {
            pa[v] = e.u, S[v] = 1;
            int nu = st[match[v]];
            slack[v] = slack[nu] = 0;
            S[nu] = 0, q_push(nu);

```

```

} else if (S[v] == 0) {
    int lca = get_lca(u, v);
    if (!lca) return augment(u,v), augment(v,u), true;
    else add_blossom(u, lca, v);
}
return false;
}
bool matching() {
    memset(S + 1, -1, sizeof(int) * n_x);
    memset(slack + 1, 0, sizeof(int) * n_x);
    q = queue<int>();
    for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && !match[x]) pa[x] = 0, S[x] = 0,
            q.push(x);
    if (q.empty()) return false;
    for (; ; ) {
        while (q.size()) {
            int u = q.front(); q.pop();
            if (S[st[u]] == 1) continue;
            for (int v = 1; v <= n; ++v)
                if (g[u][v].w > 0 && st[u] != st[v]) {
                    if (e_delta(g[u][v]) == 0) {
                        if (on_found_edge(g[u][v])) return true;
                    } else update_slack(u, st[v]);
                }
        }
        int d = inf;
        for (int b = n + 1; b <= n_x; ++b)
            if (st[b] == b && S[b] == 1) d = min(d, lab[b] / 2);
        for (int x = 1; x <= n_x; ++x)
            if (st[x] == x && slack[x]) {
                if (S[x] == -1) d = min(d, e_delta(g[slack[x]]));
                else if (S[x] == 0) d = min(d, e_delta(g[slack[x]]));
            }
        for (int u = 1; u <= n; ++u) {
            if (S[st[u]] == 0) {
                if (lab[u] <= d) return 0;
                lab[u] -= d;
            } else if (S[st[u]] == 1) lab[u] += d;
        }
        for (int b = n + 1; b <= n_x; ++b)
            if (st[b] == b) {
                if (S[st[b]] == 0) lab[b] += d * 2;
                else if (S[st[b]] == 1) lab[b] -= d * 2;
            }
        q = queue<int>();
        for (int x = 1; x <= n_x; ++x)
            if (st[x] == x && slack[x] && st[slack[x]] != x
                && e_delta(g[slack[x]])) {
                if (on_found_edge(g[slack[x]])) return true;
            }
        for (int b = n + 1; b <= n_x; ++b)
            if (st[b] == b && S[b] == 1 && lab[b] == 0)
                expand_blossom(b);
    }
    return false;
}
pair<long long, int> solve() {
    memset(match + 1, 0, sizeof(int) * n);
    n_x = n;
    int n_matches = 0;
    long long tot_weight = 0;
    for (int u = 0; u <= n; ++u) st[u] = u, flo[u].clear();
    int w_max = 0;
    for (int u = 1; u <= n; ++u)
        for (int v = 1; v <= n; ++v) {
            flo_from[u][v] = (u == v ? u : 0);
            w_max = max(w_max, g[u][v].w);
        }
    for (int u = 1; u <= n; ++u) lab[u] = w_max;
    while (matching()) ++n_matches;
    for (int u = 1; u <= n; ++u)
        if (match[u] && match[u] < u)
            tot_weight += g[u][match[u]].w;
    return make_pair(tot_weight, n_matches);
}
void add_edge(int ui, int vi, int wi) { g[ui][vi].w =

```

```

g[vi][ui].w = wi; }
void init(int _n) {
    n = _n;
    for (int u = 1; u <= n; ++u)
        for (int v = 1; v <= n; ++v)
            g[u][v] = edge(u, v, 0);
}
};

```

3.8 Flow Model

- Maximum/Minimum flow with lower bound / Circulation problem
 - Construct super source S and sink T .
 - For each edge (x, y, l, u) , connect $x \rightarrow y$ with capacity $u - l$.
 - For each vertex v , denote by $in(v)$ the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
 - If $in(v) > 0$, connect $S \rightarrow v$ with capacity $in(v)$, otherwise, connect $v \rightarrow T$ with capacity $-in(v)$.
 - To maximize, connect $t \rightarrow s$ with capacity ∞ (skip this in circulation problem), and let f be the maximum flow from S to T . If $f \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, the maximum flow from s to t is the answer.
 - To minimize, let f be the maximum flow from S to T . Connect $t \rightarrow s$ with capacity ∞ and let the flow from S to T be f' . If $f + f' \neq \sum_{v \in V, in(v) > 0} in(v)$, there's no solution. Otherwise, f' is the answer.
 - The solution of each edge e is $l_e + f_e$, where f_e corresponds to the flow of edge e on the graph.
- Construct minimum vertex cover from maximum matching M on bipartite graph (X, Y)
 - Redirect every edge: $y \rightarrow x$ if $(x, y) \in M$, $x \rightarrow y$ otherwise.
 - DFS from unmatched vertices in X .
 - $x \in X$ is chosen iff x is unvisited.
 - $y \in Y$ is chosen iff y is visited.
- Maximum density induced subgraph
 - Binary search on answer, suppose we're checking answer T
 - Construct a max flow model, let K be the sum of all weights
 - Connect source $s \rightarrow v$, $v \in G$ with capacity K
 - For each edge (u, v, w) in G , connect $u \rightarrow v$ and $v \rightarrow u$ with capacity w
 - For $v \in G$, connect it with sink $v \rightarrow t$ with capacity $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
 - T is a valid answer if the maximum flow $f < K|V|$
- Minimum weight edge cover
 - For each $v \in V$ create a copy v' , and connect $u \rightarrow v'$ with weight $w(u, v)$.
 - Connect $v \rightarrow v'$ with weight $2\mu(v)$, where $\mu(v)$ is the cost of the cheapest edge incident to v .
 - Find the minimum weight perfect matching on G' .
- Project selection problem
 - If $p_v > 0$, create edge (s, v) with capacity p_v ; otherwise, create edge (v, t) with capacity $-p_v$.
 - Create edge (u, v) with capacity w with w being the cost of choosing u without choosing v .
 - The mincut is equivalent to the maximum profit of a subset of projects.

$$\sum_x c_x x + \sum_y c_y \bar{y} + \sum_{xy} c_{xy} x \bar{y} + \sum_{xyx'y'} c_{xyx'y'} (x \bar{y} + x' \bar{y}')$$

can be minimized by the mincut of the following graph:

- Create edge (x, t) with capacity c_x and create edge (s, y) with capacity c_y .
- Create edge (x, y) with capacity c_{xy} .
- Create edge (x, y) and edge (x', y') with capacity $c_{xyx'y'}$.

4 Graph

4.1 Heavy-Light Decomposition

```

vector<int> dep, pa, sz, ch, hd, id;
int _id;
void dfs(int i, int p) {
    dep[i] = ~p ? dep[p] + 1 : 0;
    pa[i] = p, sz[i] = 1, ch[i] = -1;
    for (int j : g[i])
        if (j != p) {
            dfs(j, i);
            if (ch[i] == -1 || sz[ch[i]] < sz[j]) ch[i] = j;
            sz[i] += sz[j];
        }
}

```



```

void hld(int i, int p, int h) {
    hd[i] = h;
    id[i] = _id++;
    if (~ch[i]) hld(ch[i], i, h);
    for (int j : g[i]) if (j != p && j != ch[i])
        hld(j, i, j);
}
void query(int i, int j) {
    while (hd[i] != hd[j]) {
        if (dep[hd[i]] < dep[hd[j]]) swap(i, j);
        query2(id[hd[i]], id[i] + 1, i = pa[hd[i]]);
    }
    if (dep[i] < dep[j]) swap(i, j);
    query2(id[j], id[i] + 1);
}

```

4.2 Centroid Decomposition

```

vector<vector<int>> dis; // dis[n][Logn]
vector<int> pa, sz, dep;
vector<bool> vis;
void dfs_sz(int i, int p) {
    sz[i] = 1;
    for (int j : g[i]) if (j != p && !vis[j])
        dfs_sz(j, i), sz[i] += sz[j];
}
int cen(int i, int p, int _n) {
    for (int j : g[i]) if (j != p && !vis[j] && sz[j] >
        _n / 2)
        return cen(j, i, _n);
    return i;
}
void dfs_dis(int i, int p, int d) { // from i to
    ancestor with depth d
    dis[i][d] = ~p ? dis[p][d] + 1 : 0;
    for (int j : g[i]) if (j != p && !vis[j])
        dfs_dis(j, i, d);
}
void cd(int i, int p, int d) {
    dfs_sz(i, -1), i = cen(i, -1, sz[i]);
    vis[i] = true, pa[i] = p, dep[i] = d;
    dfs_dis(i, -1, d);
    for (int j : g[i]) if (!vis[j])
        cd(j, i, d + 1);
}

```

4.3 Edge BCC

```

vector<int> low, dep, bcc_id, stk;
vector<bool> vis;
int _id;
void dfs(int i, int p) {
    low[i] = dep[i] = ~p ? dep[p] + 1 : 0;
    stk.push_back(i);
    vis[i] = true;
    for (int j : g[i])
        if (j != p) {
            if (!vis[j])
                dfs(j, i), low[i] = min(low[i], low[j]);
            else
                low[i] = min(low[i], dep[j]);
        }
    if (low[i] == dep[i]) {
        int id = _id++;
        while (stk.back() != i) {
            int x = stk.back();
            stk.pop_back();
            bcc_id[x] = id;
        }
        stk.pop_back();
        bcc_id[i] = id;
    }
}

```

4.4 Block Cut Tree

```

vector<vector<int>> g, _g;
vector<int> dep, low, stk;
void dfs(int i, int p) {
    dep[i] = low[i] = ~p ? dep[p] + 1 : 0;
    stk.push_back(i);
    for (int j : g[i]) if (j != p) {

```

```

        if (dep[j] == -1) {
            dfs(j, i), low[i] = min(low[i], low[j]);
            if (low[j] >= dep[i]) {
                int id = _g.size();
                _g.emplace_back();
                while (stk.back() != j) {
                    int x = stk.back();
                    stk.pop_back();
                    _g[x].push_back(id), _g[id].push_back(x);
                }
                stk.pop_back();
                _g[j].push_back(id), _g[id].push_back(j);
                _g[i].push_back(id), _g[id].push_back(i);
            }
        } else low[i] = min(low[i], dep[j]);
    }
}

```

4.5 SCC / 2SAT

```

struct SAT {
    vector<vector<int>> g;
    vector<int> dep, low, scc_id;
    vector<bool> is;
    vector<int> stk;
    int n, _id, _t;
    SAT() {}
    void init(int _n) {
        n = _n, _id = _t = 0;
        g.assign(2 * n, vector<int>());
        dep.assign(2 * n, -1), low.assign(2 * n, -1);
        scc_id.assign(2 * n, -1), is.assign(2 * n, false);
        stk.clear();
    }
    void add_edge(int x, int y) {g[x].push_back(y);}
    int rev(int i) {return i < n ? i + n : i - n;}
    void add_ifthen(int x, int y) {add_clause(rev(x), y);}
    void add_clause(int x, int y) {
        add_edge(rev(x), y);
        add_edge(rev(y), x);
    }
    void dfs(int i) {
        dep[i] = low[i] = _t++;
        stk.push_back(i);
        for (int j : g[i])
            if (scc_id[j] == -1) {
                if (dep[j] == -1)
                    dfs(j);
                low[i] = min(low[i], low[j]);
            }
        if (low[i] == dep[i]) {
            int id = _id++;
            while (stk.back() != i) {
                int x = stk.back();
                stk.pop_back();
                scc_id[x] = id;
            }
            stk.pop_back();
            scc_id[i] = id;
        }
    }
    bool solve() {
        for (int i = 0; i < 2 * n; ++i)
            if (dep[i] == -1)
                dfs(i);
        for (int i = 0; i < n; ++i) {
            if (scc_id[i] == scc_id[i + n]) return false;
            if (scc_id[i] < scc_id[i + n])
                is[i] = true;
            else
                is[i + n] = true;
        }
        return true;
    }
};

```

4.6 Negative Cycle*

```

vector<pair<int, long long>> adj[N];
template<typename T>
struct NegativeCycle {

```

```

vector<T> dis;
vector<int> rt;
int n; T INF;
vector<int> cycle;
NegativeCycle () = default;
NegativeCycle (int _n) : n(_n), INF(numeric_limits<T>
    >::max()) {
    dis.assign(n, 0), rt.assign(n, -1);
    int relax = -1;
    for (int t = 0; t < n; ++t) {
        relax = -1;
        for (int i = 0; i < n; ++i) {
            for (auto [j, w] : adj[i]) if (dis[j] > dis[i]
                + w) {
                dis[j] = dis[i] + w, rt[j] = i;
                relax = j;
            }
        }
    }
    if (relax != -1) {
        int s = relax;
        for (int i = 0; i < n; ++i) s = rt[s];
        vector<bool> vis(n, false);
        while (!vis[s]) {
            cycle.push_back(s), vis[s] = true;
            s = rt[s];
        }
        reverse(cycle.begin(), cycle.end());
    }
}
};

```

4.7 Virtual Tree

```

vector<vector<int>> _g;
vector<int> st, ed, stk;
void solve(vector<int> v) {
    sort(all(v), [&](int x, int y) {return st[x] < st[y]
        ;});
    int sz = v.size();
    for (int i = 0; i < sz - 1; ++i)
        v.push_back(lca(v[i], v[i + 1]));
    sort(all(v), [&](int x, int y) {return st[x] < st[y]
        ;});
    v.resize(unique(all(v)) - v.begin());
    stk.clear(); stk.push_back(v[0]);
    for (int i = 1; i < v.size(); ++i) {
        int x = v[i];
        while (ed[stk.back()] < ed[x]) stk.pop_back();
        _g[stk.back()].push_back(x), stk.push_back(x);
    }
    // do something
    for (int i : v) _g[i].clear();
}

```

4.8 Directed MST

```

template <typename T> struct DMST { // 1-based
    T g[maxn][maxn], fw[maxn];
    int n, fr[maxn];
    bool vis[maxn], inc[maxn];
    void clear() {
        for (int i = 0; i < maxn; ++i) {
            for (int j = 0; j < maxn; ++j) g[i][j] = inf;
            vis[i] = inc[i] = false;
        }
    }
    void addedge(int u, int v, T w) {
        g[u][v] = min(g[u][v], w);
    }
    T query(int root, int _n) {
        n = _n;
        if (dfs(root) != n) return -1;
        T ans = 0;
        while (true) {
            for (int i = 1; i <= n; ++i) fw[i] = inf, fr[i] = i;
            for (int i = 1; i <= n; ++i) if (!inc[i]) {
                for (int j = 1; j <= n; ++j) {
                    if (!inc[j] && i != j && g[j][i] < fw[i]) {
                        fw[i] = g[j][i];
                        fr[i] = j;
                    }
                }
            }
        }
    }
};

```

```

    }
}
int x = -1;
for (int i = 1; i <= n; ++i) if (i != root && !
    inc[i]) {
    int j = i, c = 0;
    while (j != root && fr[j] != i && c <= n) ++c,
        j = fr[j];
    if (j == root || c > n) continue;
    else { x = i; break; }
}
if (!~x) {
    for (int i = 1; i <= n; ++i) if (i != root && !
        inc[i]) ans += fw[i];
    return ans;
}
int y = x;
for (int i = 1; i <= n; ++i) vis[i] = false;
do { ans += fw[y]; y = fr[y]; vis[y] = inc[y] =
    true; } while (y != x);
inc[x] = false;
for (int k = 1; k <= n; ++k) if (vis[k]) {
    for (int j = 1; j <= n; ++j) if (!vis[j]) {
        if (g[x][j] > g[k][j]) g[x][j] = g[k][j];
        if (g[j][k] < inf && g[j][k] - fw[k] < g[j]
            [x]) g[j][x] = g[j][k] - fw[k];
    }
}
}
return ans;
}
int dfs(int now) {
    int r = 1;
    vis[now] = true;
    for (int i = 1; i <= n; ++i) if (g[now][i] < inf &&
        !vis[i]) r += dfs(i);
    return r;
}
};

```

4.9 Dominator Tree

```

struct Dominator_tree {
    int n, id;
    vector<vector<int>> adj, radj, bucket;
    vector<int> sdom, dom, vis, rev, par, rt, mn;
    Dominator_tree (int _n) : n(_n), id(0) {
        adj.resize(n), radj.resize(n), bucket.resize(n);
        sdom.resize(n), dom.resize(n, -1), vis.resize(n,
            -1);
        rev.resize(n), rt.resize(n), mn.resize(n), par.
            resize(n);
    }
    void add_edge(int u, int v) {adj[u].pb(v);}
    int query(int v, bool x) {
        if (rt[v] == v) return x ? -1 : v;
        int p = query(rt[v], true);
        if (p == -1) return x ? rt[v] : mn[v];
        if (sdom[mn[v]] > sdom[mn[rt[v]]]) mn[v] = mn[rt[v]
            ];
        rt[v] = p;
        return x ? p : mn[v];
    }
    void dfs(int v) {
        vis[v] = id, rev[id] = v;
        rt[id] = mn[id] = sdom[id] = id, id++;
        for (int u : adj[v]) {
            if (vis[u] == -1) dfs(u), par[vis[u]] = vis[v];
            radj[vis[u]].pb(vis[v]);
        }
    }
    void build(int s) {
        dfs(s);
        for (int i = id - 1; ~i; --i) {
            for (int u : radj[i]) {
                sdom[i] = min(sdom[i], sdom[query(u, false)]);
            }
            if (i) bucket[sdom[i]].pb(i);
            for (int u : bucket[i]) {
                int p = query(u, false);
                dom[u] = sdom[p] == i ? i : p;
            }
        }
    }
};

```

```

    }
    if (i) rt[i] = par[i];
}
vector<int> res(n, -1);
for (int i = 1; i < id; ++i) {
    if (dom[i] != sdom[i]) dom[i] = dom[dom[i]];
}
for (int i = 1; i < id; ++i) res[rev[i]] = rev[dom[i]];
res[s] = s;
dom = res;
}
};

```

5 String

5.1 Aho-Corasick Automaton

```

struct AC {
    int ch[N][26], to[N][26], fail[N], sz;
    vector<int> g[N];
    int cnt[N];
    AC () {sz = 0, extend();}
    void extend() {fill(ch[sz], ch[sz] + 26, 0), sz++;}
    int nxt(int u, int v) {
        if (!ch[u][v]) ch[u][v] = sz, extend();
        return ch[u][v];
    }
    int insert(string s) {
        int now = 0;
        for (char c : s) now = nxt(now, c - 'a');
        cnt[now]++;
        return now;
    }
    void build_fail() {
        queue<int> q;
        for (int i = 0; i < 26; ++i) if (ch[0][i]) {
            q.push(ch[0][i]);
            g[0].push_back(ch[0][i]);
        }
        while (!q.empty()) {
            int v = q.front(); q.pop();
            for (int j = 0; j < 26; ++j) {
                to[v][j] = ch[v][j] ? v : to[fail[v]][j];
            }
            for (int i = 0; i < 26; ++i) if (ch[v][i]) {
                int u = ch[v][i], k = fail[v];
                while (k && !ch[k][i]) k = fail[k];
                if (ch[k][i]) k = ch[k][i];
                fail[u] = k;
                cnt[u] += cnt[k], g[k].push_back(u);
                q.push(u);
            }
        }
    }
    int match(string &s) {
        int now = 0, ans = 0;
        for (char c : s) {
            now = to[now][c - 'a'];
            if (ch[now][c - 'a']) now = ch[now][c - 'a'];
            ans += cnt[now];
        }
        return ans;
    }
};

```

5.2 KMP Algorithm

```

vector<int> build_fail(string s) {
    vector<int> f(s.length() + 1, 0);
    int k = 0;
    for (int i = 1; i < s.length(); ++i) {
        while (k && s[k] != s[i]) k = f[k];
        if (s[k] == s[i]) k++;
        f[i + 1] = k;
    }
    return f;
}
int match(string s, string t) {
    vector<int> f = build_fail(t);
    int k = 0, ans = 0;

```

```

    for (int i = 0; i < s.length(); ++i) {
        while (k && s[i] != t[k]) k = f[k];
        if (s[i] == t[k]) k++;
        if (k == t.length()) ans++, k = f[k];
    }
    return ans;
}

```

5.3 Z Algorithm

```

vector<int> build(string s) {
    int n = s.length();
    vector<int> Z(n);
    int l = 0, r = 0;
    for (int i = 0; i < n; ++i) {
        Z[i] = max(min(Z[i - 1], r - i), 0);
        while (i + Z[i] < s.size() && s[Z[i]] == s[i + Z[i]]) {
            l = i, r = i + Z[i], Z[i]++;
        }
    }
    return Z;
}

```

5.4 Manacher

```

vector<int> manacher(string &s) {
    string t = "^#";
    for (char c : s) t += c, t += '#';
    t += '&';
    int n = t.length();
    vector<int> r(n, 0);
    int C = 0, R = 0;
    for (int i = 1; i < n - 1; ++i) {
        int mirror = 2 * C - i;
        r[i] = (i < R ? min(r[mirror], R - i) : 0);
        while (t[i - 1 - r[i]] == t[i + 1 + r[i]]) r[i]++;
        if (i + r[i] > R) R = i + r[i], C = i;
    }
    return r;
}

```

5.5 Suffix Array

```

int sa[N], tmp[2][N], c[N], rk[N], lcp[N];
void buildSA(string s) {
    int *x = tmp[0], *y = tmp[1], m = 256, n = s.length();
    for (int i = 0; i < m; ++i) c[i] = 0;
    for (int i = 0; i < n; ++i) c[x[i]] = s[i]++;
    for (int i = 1; i < m; ++i) c[i] += c[i - 1];
    for (int i = n - 1; ~i; --i) sa[--c[x[i]]] = i;
    for (int k = 1; k < n; k <= 1) {
        for (int i = 0; i < m; ++i) c[i] = 0;
        for (int i = 0; i < n; ++i) c[x[i]]++;
        for (int i = 1; i < m; ++i) c[i] += c[i - 1];
        int p = 0;
        for (int i = n - k; i < n; ++i) y[p++] = i;
        for (int i = 0; i < n; ++i) if (sa[i] >= k) y[p++] = sa[i] - k;
        for (int i = n - 1; ~i; --i) sa[--c[x[y[i]]]] = y[i];
        y[sa[0]] = p = 0;
        for (int i = 1; i < n; ++i) {
            int a = sa[i], b = sa[i - 1];
            if (!(x[a] == x[b] && a + k < n && b + k < n && x[a + k] == x[b + k])) p++;
            y[sa[i]] = p;
        }
        if (n == p + 1) break;
        swap(x, y), m = p + 1;
    }
}
void buildLCP(string s) {
    // lcp[i] = LCP(sa[i - 1], sa[i])
    // lcp(i, j) = min(lcp[rk[i] + 1], lcp[rk[i] + 2], ..., lcp[rk[j]])
    int n = s.length(), val = 0;
    for (int i = 0; i < n; ++i) rk[sa[i]] = i;
    for (int i = 0; i < n; ++i) {
        if (!rk[i]) lcp[rk[i]] = 0;
        else {

```

```

    if (val) val--;
    int p = sa[rk[i] - 1];
    while (val + i < n && val + p < n && s[val + i]
           == s[val + p]) val++;
    lcp[rk[i]] = val;
}
}
}

```

5.6 SAIS

```

namespace sfx {
bool _t[N * 2];
int SA[N * 2], H[N], RA[N];
int _s[N * 2], _c[N * 2], x[N], _p[N], _q[N * 2];
void pre(int *sa, int *c, int n, int z) {
    fill_n(sa, n, 0), copy_n(c, z, x);
}
void induce(int *sa, int *c, int *s, bool *t, int n,
            int z) {
    copy_n(c, z - 1, x + 1);
    for (int i = 0; i < n; ++i) if (sa[i] && !t[sa[i] -
        1]) sa[x[s[sa[i] - 1]]++] = sa[i] - 1;
    copy_n(c, z, x);
    for (int i = n - 1; i >= 0; --i) if (sa[i] && t[sa[i] -
        1]) sa[--x[s[sa[i] - 1]]] = sa[i] - 1;
}
void sais(int *s, int *sa, int *p, int *q, bool *t, int
          *c, int n, int z) {
    bool uniq = t[n - 1] = true;
    int nn = 0, nmzx = -1, *nsa = sa + n, *ns = s + n,
        last = -1;
    fill_n(c, z, 0);
    for (int i = 0; i < n; ++i) uniq &= ++c[s[i]] < 2;
    partial_sum(c, c + z, c);
    if (uniq) {
        for (int i = 0; i < n; ++i) sa[--c[s[i]]] = i;
        return;
    }
    for (int i = n - 2; i >= 0; --i)
        t[i] = (s[i] == s[i + 1] ? t[i + 1] : s[i] < s[i +
            1]);
    pre(sa, c, n, z);
    for (int i = 1; i <= n - 1; ++i)
        if (t[i] && !t[i - 1])
            sa[--x[s[i]]] = p[q[i] = nn++] = i;
    induce(sa, c, s, t, n, z);
    for (int i = 0; i < n; ++i)
        if (sa[i] && t[sa[i]] && !t[sa[i] - 1]) {
            bool neq = last < 0 || !equal(s + sa[i], s + p[q[
                sa[i]] + 1], s + last);
            ns[q[last = sa[i]]] = nmzx += neq;
        }
    sais(ns, nsa, p + nn, q + n, t + n, c + z, nn, nmzx +
        1);
    pre(sa, c, n, z);
    for (int i = nn - 1; i >= 0; --i)
        sa[--x[s[p[nsa[i]]]]] = p[nsa[i]];
    induce(sa, c, s, t, n, z);
}
vector<int> build(int *s, int n) {
    copy_n(s, n, _s), _s[n] = 0;
    sais(_s, SA, _p, _q, _t, _c, n + 1, 256);
    vector<int> sa(n);
    for (int i = 0; i < n; ++i)
        sa[i] = SA[i + 1];
    return sa;
}
}

```

5.7 Suffix Automaton

```

struct SAM {
    int ch[N][26], len[N], link[N], pos[N], cnt[N], sz;
    // node -> strings with the same endpos set
    // length in range [len(link) + 1, len]
    // node's endpos set -> pos in the subtree of node
    // link -> longest suffix with different endpos set
    // len -> longest suffix
    // pos -> end position
    // cnt -> size of endpos set
    SAM () {len[0] = 0, link[0] = -1, pos[0] = 0, cnt[0]
        = 0, sz = 1;}
}

```

```

void build(string s) {
    int last = 0;
    for (int i = 0; i < s.length(); ++i) {
        char c = s[i];
        int cur = sz++;
        len[cur] = len[last] + 1, pos[cur] = i + 1;
        int p = last;
        while (~p && !ch[p][c - 'a']) ch[p][c - 'a'] =
            cur, p = link[p];
        if (p == -1) {
            link[cur] = 0;
        } else {
            int q = ch[p][c - 'a'];
            if (len[p] + 1 == len[q]) {
                link[cur] = q;
            } else {
                int nxt = sz++;
                len[nxt] = len[p] + 1, link[nxt] = link[q],
                    pos[nxt] = 0;
                for (int j = 0; j < 26; ++j) ch[nxt][j] = ch[
                    q][j];
                while (~p && ch[p][c - 'a'] == q) ch[p][c -
                    'a'] = nxt, p = link[p];
                link[q] = link[cur] = nxt;
            }
        }
        cnt[cur]++;
        last = cur;
    }
    vector<int> p(sz);
    iota(all(p), 0);
    sort(all(p), [&](int i, int j) {return len[i] > len
        [j]});
    for (int i = 0; i < sz; ++i) cnt[link[p[i]]] += cnt
        [p[i]];
}
} sam;

```

5.8 Minimum Rotation

```

string rotate(const string &s) {
    int n = s.length();
    string t = s + s;
    int i = 0, j = 1;
    while (i < n && j < n) {
        int k = 0;
        while (k < n && t[i + k] == t[j + k]) ++k;
        if (t[i + k] <= t[j + k]) j += k + 1;
        else i += k + 1;
        if (i == j) ++j;
    }
    int pos = (i < n ? i : j);
    return t.substr(pos, n);
}

```

5.9 Palindrome Tree

```

struct PAM {
    int ch[N][26], cnt[N], fail[N], len[N], sz;
    string s;
    // 0 -> even root, 1 -> odd root
    PAM (string _s) : s(_s) {
        sz = 0;
        extend(), extend();
        len[0] = 0, fail[0] = 1, len[1] = -1;
        int lst = 1;
        for (int i = 0; i < s.length(); ++i) {
            while (s[i - len[lst] - 1] != s[i]) lst = fail[
                lst];
            if (!ch[lst][s[i] - 'a']) {
                int idx = extend();
                len[idx] = len[lst] + 2;
                int now = fail[lst];
                while (s[i - len[now] - 1] != s[i]) now = fail[
                    now];
                fail[idx] = ch[now][s[i] - 'a'];
                ch[lst][s[i] - 'a'] = idx;
            }
            lst = ch[lst][s[i] - 'a'], cnt[lst]++;
        }
    }
    void build_count() {
}

```

```

    for (int i = sz - 1; i > 1; --i)
        cnt[fail[i]] += cnt[i];
}
int extend() {
    fill(ch[sz], ch[sz] + 26, 0), sz++;
    return sz - 1;
}
};

```

5.10 Main Lorentz

```

int to_left[N], to_right[N];
vector<array<int, 3>> rep; // L, r, Len.
// substr(L ~ r, len * 2) are tandem
void findRep(string &s, int l, int r) {
    if (r - l == 1) return;
    int m = l + r >> 1;
    findRep(s, l, m), findRep(s, m, r);
    string sl = s.substr(l, m - l), sr = s.substr(m, r - m);
    vector<int> Z = buildZ(sr + "#" + sl);
    for (int i = 1; i < m; ++i) to_right[i] = Z[r - m + 1 + i - 1];
    reverse(all(sl));
    Z = buildZ(sl);
    for (int i = 1; i < m; ++i) to_left[i] = Z[m - i - 1];
    reverse(all(sl));
    for (int i = 1; i + 1 < m; ++i) {
        int k1 = to_left[i], k2 = to_right[i + 1], len = m - i - 1;
        if (k1 < 1 || k2 < 1 || len < 2) continue;
        int tl = max(1, len - k2), tr = min(len - 1, k1);
        if (tl <= tr) rep.pb({i + 1 - tr, i + 1 - tl, len});
    }
    Z = buildZ(sr);
    for (int i = m; i < r; ++i) to_right[i] = Z[i - m];
    reverse(all(sl)), reverse(all(sr));
    Z = buildZ(sl + "#" + sr);
    for (int i = m; i < r; ++i) to_left[i] = Z[m - 1 + 1 + r - i - 1];
    reverse(all(sl)), reverse(all(sr));
    for (int i = m; i + 1 < r; ++i) {
        int k1 = to_left[i], k2 = to_right[i + 1], len = i - m + 1;
        if (k1 < 1 || k2 < 1 || len < 2) continue;
        int tl = max(len - k2, 1), tr = min(len - 1, k1);
        if (tl <= tr) rep.pb({i + 1 - len - tr, i + 1 - len - tl, len});
    }
    Z = buildZ(sr + "#" + sl);
    for (int i = 1; i < m; ++i) {
        if (Z[r - m + 1 + i - 1] >= m - i) {
            rep.pb({i, i, m - i});
        }
    }
}

```

6 Math

6.1 Fraction*

```

struct fraction {
    ll n, d;
    fraction(const ll _n=0, const ll _d=1): n(_n), d(_d) {
        {
            ll t = gcd(n, d);
            n /= t, d /= t;
            if (d < 0) n = -n, d = -d;
        }
    }
    fraction operator-() const {
        return fraction(-n, d);
    }
    fraction operator+(const fraction &b) const {
        return fraction(n * b.d + b.n * d, d * b.d);
    }
    fraction operator-(const fraction &b) const {
        return fraction(n * b.d - b.n * d, d * b.d);
    }
    fraction operator*(const fraction &b) const {
        return fraction(n * b.n, d * b.d);
    }
    fraction operator/(const fraction &b) const {
        return fraction(n * b.d, d * b.n);
    }
}

```

```

void print() {
    cout << n;
    if (d != 1) cout << "/" << d;
}
};

```

6.2 Miller Rabin / Pollard Rho

```

ll mul(ll x, ll y, ll p) {return (x * y - (ll)((long double)x / p * y) * p + p) % p;}
vector<ll> chk = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
ll Pow(ll a, ll b, ll n) {ll res = 1; for (; b; b >>= 1, a = mul(a, a, n)) if (b & 1) res = mul(res, a, n); return res;}
bool check(ll a, ll d, int s, ll n) {
    a = Pow(a, d, n);
    if (a <= 1) return 1;
    for (int i = 0; i < s; ++i, a = mul(a, a, n)) {
        if (a == 1) return 0;
        if (a == n - 1) return 1;
    }
    return 0;
}
bool IsPrime(ll n) {
    if (n < 2) return 0;
    if (n % 2 == 0) return n == 2;
    ll d = n - 1, s = 0;
    while (d % 2 == 0) d >>= 1, ++s;
    for (ll i : chk) if (!check(i, d, s, n)) return 0;
    return 1;
}
const vector<ll> small = {2, 3, 5, 7, 11, 13, 17, 19};
ll FindFactor(ll n) {
    if (IsPrime(n)) return 1;
    for (ll p : small) if (n % p == 0) return p;
    ll x, y = 2, d, t = 1;
    auto f = [&](ll a) {return (mul(a, a, n) + t) % n;};
    for (int l = 2; l <= 1) {
        x = y;
        int m = min(1, 32);
        for (int i = 0; i < l; i += m) {
            d = 1;
            for (int j = 0; j < m; ++j) {
                y = f(y), d = mul(d, abs(x - y), n);
            }
            ll g = __gcd(d, n);
            if (g == n) {
                l = 1, y = 2, ++t;
                break;
            }
            if (g != 1) return g;
        }
    }
}
map<ll, int> res;
void PollardRho(ll n) {
    if (n == 1) return;
    if (IsPrime(n)) return ++res[n], void(0);
    ll d = FindFactor(n);
    PollardRho(n / d), PollardRho(d);
}

```

6.3 Ext GCD

```

//a * p.first + b * p.second = gcd(a, b)
pair<ll, ll> extgcd(ll a, ll b) {
    pair<ll, ll> res;
    if (a < 0) {
        res = extgcd(-a, b);
        res.first *= -1;
        return res;
    }
    if (b < 0) {
        res = extgcd(a, -b);
        res.second *= -1;
        return res;
    }
    if (b == 0) return {1, 0};
    res = extgcd(b, a % b);
    return {res.second, res.first - res.second * (a / b)};
}

```

6.4 PiCount

```
const int V = 10000000, N = 100, M = 100000;
vector<int> primes;
bool isp[V];
int small_pi[V], dp[N][M];
void sieve(int x){
    for(int i = 2; i < x; ++i) isp[i] = true;
    isp[0] = isp[1] = false;
    for(int i = 2; i * i < x; ++i) if(isp[i]) for(int j = i * i; j < x; j += i) isp[j] = false;
    for(int i = 2; i < x; ++i) if(isp[i]) primes.push_back(i);
}
void init(){
    sieve(V);
    small_pi[0] = 0;
    for(int i = 1; i < V; ++i) small_pi[i] = small_pi[i - 1] + isp[i];
    for(int i = 0; i < M; ++i) dp[0][i] = i;
    for(int i = 1; i < N; ++i) for(int j = 0; j < M; ++j) dp[i][j] = dp[i - 1][j] - dp[i - 1][j / primes[i - 1]];
}
ll phi(ll n, int a){
    if(!a) return n;
    if(n < M && a < N) return dp[a][n];
    if(primes[a - 1] > n) return 1;
    if(((ll)primes[a - 1]) * primes[a - 1] >= n && n < V) return small_pi[n] - a + 1;
    ll de = phi(n, a - 1) - phi(n / primes[a - 1], a - 1);
    return de;
}
ll PiCount(ll n){
    if(n < V) return small_pi[n];
    int s = sqrt(n + 0.5), y = cbrt(n + 0.5), a = small_pi[y];
    ll res = phi(n, a) + a - 1;
    for(; primes[a] <= s; ++a) res -= max(PiCount(n / primes[a]) - PiCount(primes[a]) + 1, 0ll);
    return res;
}
```

6.5 Linear Function Mod Min

```
ll topos(ll x, ll m) {x %= m; if (x < 0) x += m; return x;}
//min value of ax + b (mod m) for x \in [0, n - 1]. O(Log m)
ll min_rem(ll n, ll m, ll a, ll b) {
    a = topos(a, m), b = topos(b, m);
    for (ll g = __gcd(a, m); g > 1; ) return g * min_rem(n / g, m / g, a / g, b / g) + (b % g);
    for (ll nn, nm, na, nb; a; n = nn, m = nm, a = na, b = nb) {
        if (a <= m - a) {
            nn = (a * (n - 1) + b) / m;
            if (!nn) break;
            nn += (b < a);
            nm = a, na = topos(-m, a);
            nb = b < a ? b : topos(b - m, a);
        } else {
            ll lst = b - (n - 1) * (m - a);
            if (lst >= 0) {b = lst; break;}
            nn = -(lst / m) + (lst % m < -a) + 1;
            nm = m - a, na = m % (m - a), nb = b % (m - a);
        }
    }
    return b;
}
//min value of ax + b (mod m) for x \in [0, n - 1], also return min x to get the value. O(log m)
//{value, x}
pair<ll, ll> min_rem_pos(ll n, ll m, ll a, ll b) {
    a = topos(a, m), b = topos(b, m);
    ll mn = min_rem(n, m, a, b), g = __gcd(a, m);
    //ax = (mn - b) (mod m)
    ll x = (extgcd(a, m).first + m) * ((mn - b + m) / g) % (m / g);
    return {mn, x};
}
```

6.6 Determinant

```
ll Det(vector<vector<ll>> a) {
    int n = a.size();
    ll det = 1;
    for (int i = 0; i < n; ++i) {
        if (!a[i][i]) {
            det = -det;
            if (det < 0) det += mod;
            for (int j = i + 1; j < n; ++j) if (a[j][i]) {
                swap(a[j], a[i]);
                break;
            }
            if (!a[i][i]) return 0;
        }
        det = det * a[i][i] % mod;
        ll mul = mpow(a[i][i], mod - 2);
        for (int j = 0; j < n; ++j) a[i][j] = a[i][j] * mul % mod;
        for (int j = 0; j < n; ++j) if (i ^ j) {
            ll mul = a[j][i];
            for (int k = 0; k < n; ++k) {
                a[j][k] -= a[i][k] * mul % mod;
                if (a[j][k] < 0) a[j][k] += mod;
            }
        }
    }
    return det;
}
```

6.7 Floor Sum

```
// sum^{n-1}_0 floor((a * i + b) / m) in log(n + m + a + b)
ll floor_sum(ll n, ll m, ll a, ll b) {
    ll ans = 0;
    if (a >= m) ans += (n - 1) * n * (a / m) / 2, a %= m;
    if (b >= m) ans += n * (b / m), b %= m;
    ll y_max = (a * n + b) / m, x_max = (y_max * m - b);
    if (y_max == 0) return ans;
    ans += (n - (x_max + a - 1) / a) * y_max;
    ans += floor_sum(y_max, a, m, (a - x_max % a) % a);
    return ans;
}
```

6.8 Quadratic Residue

```
int Jacobi(int a, int m) {
    int s = 1;
    for (; m > 1; ) {
        a %= m;
        if (a == 0) return 0;
        const int r = __builtin_ctz(a);
        if ((r & 1) && ((m + 2) & 4)) s = -s;
        a >>= r;
        if (a & m & 2) s = -s;
        swap(a, m);
    }
    return s;
}
int QuadraticResidue(int a, int p) {
    if (p == 2) return a & 1;
    const int jc = Jacobi(a, p);
    if (jc == 0) return 0;
    if (jc == -1) return -1;
    int b, d;
    for (; ) {
        b = rand() % p;
        d = (1LL * b * b + p - a) % p;
        if (Jacobi(d, p) == -1) break;
    }
    int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
    for (int e = (1LL + p) >> 1; e; e >>= 1) {
        if (e & 1) {
            tmp = (1LL * g0 * f0 + 1LL * d * (1LL * g1 * f1 % p)) % p;
            g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
            g0 = tmp;
        }
        tmp = (1LL * f0 * f0 + 1LL * d * (1LL * f1 * f1 % p)) % p;
        f1 = (2LL * f0 * f1) % p;
    }
}
```



```

    f0 = tmp;
}
return g0;
}

```

6.9 Simplex

```

struct Simplex { // 0-based
using T = long double;
static const int N = 410, M = 30010;
const T eps = 1e-7;
int n, m;
int Left[M], Down[N];
// Ax ≤ b, max c^T x
// result : v, xi = sol[i]
T a[M][N], b[M], c[N], v, sol[N];
bool eq(T a, T b) {return fabs(a - b) < eps;}
bool ls(T a, T b) {return a < b && !eq(a, b);}
void init(int _n, int _m) {
n = _n, m = _m, v = 0;
for (int i = 0; i < m; ++i) for (int j = 0; j < n; ++j) a[i][j] = 0;
for (int i = 0; i < m; ++i) b[i] = 0;
for (int i = 0; i < n; ++i) c[i] = sol[i] = 0;
}
void pivot(int x, int y) {
swap(Left[x], Down[y]);
T k = a[x][y]; a[x][y] = 1;
vector<int> nz;
for (int i = 0; i < n; ++i) {
a[x][i] /= k;
if (!eq(a[x][i], 0)) nz.push_back(i);
}
b[x] /= k;
for (int i = 0; i < m; ++i) {
if (i == x || eq(a[i][y], 0)) continue;
k = a[i][y], a[i][y] = 0;
b[i] -= k * b[x];
for (int j : nz) a[i][j] -= k * a[x][j];
}
if (eq(c[y], 0)) return;
k = c[y], c[y] = 0, v += k * b[x];
for (int i : nz) c[i] -= k * a[x][i];
}
// 0: found solution, 1: no feasible solution, 2:
// unbounded
int solve() {
for (int i = 0; i < n; ++i) Down[i] = i;
for (int i = 0; i < m; ++i) Left[i] = n + i;
while (1) {
int x = -1, y = -1;
for (int i = 0; i < m; ++i) if (ls(b[i], 0) && (x == -1 || b[i] < b[x])) x = i;
if (x == -1) break;
for (int i = 0; i < n; ++i) if (ls(a[x][i], 0) && (y == -1 || a[x][i] < a[x][y])) y = i;
if (y == -1) return 1;
pivot(x, y);
}
while (1) {
int x = -1, y = -1;
for (int i = 0; i < n; ++i) if (ls(0, c[i]) && (y == -1 || c[i] > c[y])) y = i;
if (y == -1) break;
for (int i = 0; i < m; ++i) if (ls(0, a[i][y]) && (x == -1 || b[i] / a[i][y] < b[x] / a[x][y])) x = i;
if (x == -1) return 2;
pivot(x, y);
}
for (int i = 0; i < m; ++i) if (Left[i] < n) sol[Left[i]] = b[i];
return 0;
}
};

```

6.10 Berlekamp Massey

```

vector<ll> BerlekampMassey(vector<ll> a) {
// find min |c| such that a_n = sum c_j * a_{n-j-1}, 0-based
// O(N^2), if |c| = k, |a| ≥ 2k sure correct

```

```

auto f = [&](vector<ll> v, ll c) {
for (ll &x : v) x = mul(x, c);
return v;
};
vector<ll> c, best;
int pos = 0, n = a.size();
for (int i = 0; i < n; ++i) {
ll error = a[i];
for (int j = 0; j < c.size(); ++j) error = sub(
error, mul(c[j], a[i - 1 - j]));
if (error == 0) continue;
ll inv = mpow(error, mod - 2);
if (c.empty()) {
c.resize(i + 1);
pos = i;
best.pb(inv);
} else {
vector<ll> fix = f(best, error);
fix.insert(fix.begin(), i - pos - 1, 0);
if (fix.size() >= c.size()) {
best = f(c, sub(0, inv));
best.insert(best.begin(), inv);
pos = i;
c.resize(fix.size());
}
for (int j = 0; j < fix.size(); ++j) c[j] = add(c[j], fix[j]);
}
}
return c;
}

```

6.11 Linear Programming Construction

Standard form: maximize $c^T x$ subject to $Ax \leq b$ and $x \geq 0$.
Dual LP: minimize $b^T y$ subject to $A^T y \geq c$ and $y \geq 0$.
 \bar{x} and \bar{y} are optimal if and only if for all $i \in [1, n]$, either $\bar{x}_i = 0$ or $\sum_{j=1}^m A_{ji} \bar{y}_j = c_i$ holds and for all $i \in [1, m]$ either $\bar{y}_i = 0$ or $\sum_{j=1}^n A_{ij} \bar{x}_j = b_j$ holds.

1. In case of minimization, let $c'_i = -c_i$
2. $\sum_{1 \leq i \leq n} A_{ji} x_i \geq b_j \rightarrow \sum_{1 \leq i \leq n} -A_{ji} x_i \leq -b_j$
3. $\sum_{1 \leq i \leq n} A_{ji} x_i = b_j$
 - $\sum_{1 \leq i \leq n} A_{ji} x_i \leq b_j$
 - $\sum_{1 \leq i \leq n} A_{ji} x_i \geq b_j$
4. If x_i has no lower bound, replace x_i with $x_i - x'_i$

6.12 Euclidean

- $m = \lfloor \frac{an+b}{c} \rfloor$
- Time complexity: $O(\log n)$

$$f(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor$$

$$= \begin{cases} \lfloor \frac{a}{c} \rfloor \cdot \frac{n(n+1)}{2} + \lfloor \frac{b}{c} \rfloor \cdot (n+1) \\ + f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ nm - f(c, c-b-1, a, m-1), & \text{otherwise} \end{cases}$$

$$g(a, b, c, n) = \sum_{i=0}^n i \lfloor \frac{ai+b}{c} \rfloor$$

$$= \begin{cases} \lfloor \frac{a}{c} \rfloor^2 \cdot \frac{n(n+1)(2n+1)}{6} + \lfloor \frac{b}{c} \rfloor \cdot \frac{n(n+1)}{2} \\ + f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ \frac{1}{2} \cdot (n(n+1)m - f(c, c-b-1, a, m-1) \\ - h(c, c-b-1, a, m-1)), & \text{otherwise} \end{cases}$$

$$h(a, b, c, n) = \sum_{i=0}^n \lfloor \frac{ai+b}{c} \rfloor^2$$

$$= \begin{cases} \lfloor \frac{a}{c} \rfloor^2 \cdot \frac{n(n+1)(2n+1)}{6} + \lfloor \frac{b}{c} \rfloor^2 \cdot (n+1) \\ + \lfloor \frac{a}{c} \rfloor \cdot \lfloor \frac{b}{c} \rfloor \cdot n(n+1) \\ + h(a \bmod c, b \bmod c, c, n) \\ + 2 \lfloor \frac{a}{c} \rfloor \cdot g(a \bmod c, b \bmod c, c, n) \\ + 2 \lfloor \frac{b}{c} \rfloor \cdot f(a \bmod c, b \bmod c, c, n), & a \geq c \vee b \geq c \\ 0, & n < 0 \vee a = 0 \\ nm(m+1) - 2g(c, c-b-1, a, m-1) \\ - 2f(c, c-b-1, a, m-1) - f(a, b, c, n), & \text{otherwise} \end{cases}$$

6.13 Theorem

• Kirchhoff's Theorem

Denote L be a $n \times n$ matrix as the Laplacian matrix of graph G , where $L_{ii} = d(i)$, $L_{ij} = -c$ where c is the number of edge (i, j) in G .

- The number of undirected spanning in G is $|\det(\tilde{L}_{11})|$.
- The number of directed spanning tree rooted at r in G is $|\det(\tilde{L}_{rr})|$.

• Tutte's Matrix

Let D be a $n \times n$ matrix, where $d_{ij} = x_{ij}$ (x_{ij} is chosen uniformly at random) if $i < j$ and $(i, j) \in E$, otherwise $d_{ij} = -d_{ji}$. $\frac{\text{rank}(D)}{2}$ is the maximum matching on G .

• Cayley's Formula

- Given a degree sequence d_1, d_2, \dots, d_n for each *Labeled* vertices, there are

$$\frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}$$

spanning trees.

- Let $T_{n,k}$ be the number of *Labeled* forests on n vertices with k components, such that vertex $1, 2, \dots, k$ belong to different components. Then $T_{n,k} = kn^{n-k-1}$.

• Erdős-Gallai Theorem

A sequence of non-negative integers $d_1 \geq d_2 \geq \dots \geq d_n$ can be represented as the degree sequence of a finite simple graph on n vertices if and only if $d_1 + d_2 + \dots + d_n$ is even and

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$$

holds for all $1 \leq k \leq n$.

• Burnside's Lemma

Let X be a set and G be a group that acts on X . For $g \in G$, denote by X^g the elements fixed by g :

$$X^g = \{x \in X \mid gx = x\}$$

Then

$$|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|.$$

• Gale-Ryser theorem

A pair of sequences of nonnegative integers $a_1 \geq \dots \geq a_n$ and b_1, \dots, b_n is bigraphic if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k)$ holds for every $1 \leq k \leq n$.

• Fulkerson-Chen-Anstee theorem

A sequence $(a_1, b_1), \dots, (a_n, b_n)$ of nonnegative integer pairs with $a_1 \geq \dots \geq a_n$ is digraphic if and only if $\sum_{i=1}^n a_i = \sum_{i=1}^n b_i$ and $\sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k-1) + \sum_{i=k+1}^n \min(b_i, k)$ holds for every $1 \leq k \leq n$.

• Möbius inversion formula

- $f(n) = \sum_{d|n} g(d) \Leftrightarrow g(n) = \sum_{d|n} \mu(d) f(\frac{n}{d})$
- $f(n) = \sum_{n|d} g(d) \Leftrightarrow g(n) = \sum_{n|d} \mu(\frac{d}{n}) f(d)$

• Spherical cap

- A portion of a sphere cut off by a plane.
- r : sphere radius, a : radius of the base of the cap, h : height of the cap, θ : $\arcsin(a/r)$.
- Volume = $\pi h^2(3r-h)/3 = \pi h(3a^2+h^2)/6 = \pi r^3(2+\cos\theta)(1-\cos\theta)^2/3$.
- Area = $2\pi rh = \pi(a^2+h^2) = 2\pi r^2(1-\cos\theta)$.

• Chinese Remainder Theorem

- $x \equiv a_i \pmod{m_i}$
- $M = \prod m_i, M_i = M/m_i$
- $t_i M_i \equiv 1 \pmod{m_i}$
- $x = \sum a_i t_i M_i \pmod{M}$

6.14 Estimation

- The number of divisors of n is at most around 100 for $n < 5e4$, 500 for $n < 1e7$, 2000 for $n < 1e10$, 20000 for $n < 1e19$.
- The number of ways of writing n as a sum of positive integers, disregarding the order of the summands. 1, 1, 2, 3, 5, 7, 11, 15, 22, 30 for $n = 0 \sim 9$, 627 for $n = 20$, $\sim 2e5$ for $n = 50$, $\sim 2e8$ for $n = 100$.
- Total number of partitions of n distinct elements: $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, 678570, 4213597, 27644437, 190899322, \dots$

6.15 General Purpose Numbers

• Bernoulli numbers

$$B_0 = 1, B_1^\pm = \pm \frac{1}{2}, B_2 = \frac{1}{6}, B_3 = 0$$

$$\sum_{j=0}^m \binom{m+1}{j} B_j = 0, \text{ EGF is } B(x) = \frac{x}{e^x - 1} = \sum_{n=0}^{\infty} B_n \frac{x^n}{n!}.$$

$$S_m(n) = \sum_{k=1}^n k^m = \frac{1}{m+1} \sum_{k=0}^m \binom{m+1}{k} B_k^+ n^{m+1-k}$$

- Stirling numbers of the second kind Partitions of n distinct elements into exactly k groups.

$$S(n, k) = S(n-1, k-1) + kS(n-1, k), S(n, 1) = S(n, n) = 1$$

$$S(n, k) = \frac{1}{k!} \sum_{i=0}^k (-1)^{k-i} \binom{k}{i} i^n$$

$$x^n = \sum_{i=0}^n S(n, i) (x)_i$$

• Pentagonal number theorem

$$\prod_{n=1}^{\infty} (1 - x^n) = 1 + \sum_{k=1}^{\infty} (-1)^k (x^{k(3k+1)/2} + x^{k(3k-1)/2})$$

• Catalan numbers

$$C_n^{(k)} = \frac{1}{(k-1)n+1} \binom{kn}{n}$$

$$C^{(k)}(x) = 1 + x[C^{(k)}(x)]^k$$

• 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. $\pi(j) > j$.

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

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

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

6.16 Tips for Generating Function

• Ordinary Generating Function $A(x) = \sum_{i \geq 0} a_i x^i$

- $A(rx) \Rightarrow r^n a_n$
- $A(x) + B(x) \Rightarrow a_n + b_n$
- $A(x)B(x) \Rightarrow \sum_{i=0}^n a_i b_{n-i}$
- $A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} a_{i_1} a_{i_2} \dots a_{i_k}$
- $x A(x)' \Rightarrow n a_n$
- $\frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^n a_i$

• Exponential Generating Function $A(x) = \sum_{i \geq 0} \frac{a_i}{i!} x^i$

- $A(x) + B(x) \Rightarrow a_n + b_n$
- $A^{(k)}(x) \Rightarrow a_{n+k}$
- $A(x)B(x) \Rightarrow \sum_{i=0}^n \binom{n}{i} a_i b_{n-i}$
- $A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} \binom{n}{i_1, i_2, \dots, i_k} a_{i_1} a_{i_2} \dots a_{i_k}$
- $x A(x) \Rightarrow n a_n$

• Special Generating Function

- $(1+x)^n = \sum_{i \geq 0} \binom{n}{i} x^i$
- $\frac{1}{(1-x)^n} = \sum_{i \geq 0} \binom{n-1}{i} x^i$

7 Polynomial

7.1 Number Theoretic Transform

```
// mul, add, sub, mpow
// ll -> int if too slow
struct NTT {
    ll w[N];
    NTT() {
        ll dw = mpow(G, (mod - 1) / N);
        w[0] = 1;
        for (int i = 1; i < N; ++i) w[i] = w[i - 1] * dw % mod;
    }
}
```

```

void operator()(vector<ll>& a, bool inv = false) { //
    0 <= a[i] < P
    int x = 0, n = a.size();
    for (int j = 1; j < n - 1; ++j) {
        for (int k = n >> 1; (x ^ k) < k; k >>= 1);
        if (j < x) swap(a[x], a[j]);
    }
    for (int L = 2; L <= n; L <= 1) {
        int dx = N / L, dl = L >> 1;
        for (int i = 0; i < n; i += L) {
            for (int j = i, x = 0; j < i + dl; ++j, x += dx) {
                ll tmp = mul(a[j + dl], w[x]);
                a[j + dl] = sub(a[j], tmp);
                a[j] = add(a[j], tmp);
            }
        }
    }
    if (inv) {
        reverse(a.begin() + 1, a.end());
        ll invn = mpow(n, mod - 2);
        for (int i = 0; i < n; ++i) a[i] = mul(a[i], invn);
    }
}
} ntt;

```

7.2 Primes

Prime	Root	Prime	Root
7681	17	167772161	3
12289	11	104857601	3
40961	3	985661441	3
65537	3	998244353	3
786433	10	1107296257	10
5767169	3	2013265921	31
7340033	3	2810183681	11
23068673	3	2885681153	3
469762049	3	605028353	3

7.3 Polynomial Operations

```

vector<ll> Mul(vector<ll> a, vector<ll> b, int bound
    = N) {
    int m = a.size() + b.size() - 1, n = 1;
    while (n < m) n <= 1;
    a.resize(n), b.resize(n);
    ntt(a), ntt(b);
    vector<ll> out(n);
    for (int i = 0; i < n; ++i) out[i] = mul(a[i], b[i]);
    ntt(out, true), out.resize(min(m, bound));
    return out;
}
vector<ll> Inverse(vector<ll> a) {
    // O(NlogN), a[0] != 0
    int n = a.size();
    vector<ll> res(1, mpow(a[0], mod - 2));
    for (int m = 1; m < n; m <= 1) {
        if (n < m * 2) a.resize(m * 2);
        vector<ll> v1(a.begin(), a.begin() + m * 2), v2 =
            res;
        v1.resize(m * 4), v2.resize(m * 4);
        ntt(v1), ntt(v2);
        for (int i = 0; i < m * 4; ++i) v1[i] = mul(mul(v1[
            i], v2[i]), v2[i]);
        ntt(v1, true);
        res.resize(m * 2);
        for (int i = 0; i < m; ++i) res[i] = add(res[i],
            res[i]);
        for (int i = 0; i < m * 2; ++i) res[i] = sub(res[i
            ], v1[i]);
    }
    res.resize(n);
    return res;
}
pair<vector<ll>, vector<ll>> Divide(vector<ll> a,
    vector<ll> b) {
    // a = bQ + R, O(NlogN), b.back() != 0
    int n = a.size(), m = b.size(), k = n - m + 1;
    if (n < m) return {{0}, a};
    vector<ll> ra = a, rb = b;
    reverse(all(ra)), ra.resize(k);
    reverse(all(rb)), rb.resize(k);
    vector<ll> Q = Mul(ra, Inverse(rb), k);

```

```

    reverse(all(Q));
    vector<ll> res = Mul(b, Q), R(m - 1);
    for (int i = 0; i < m - 1; ++i) R[i] = sub(a[i], res[
        i]);
    return {Q, R};
}
vector<ll> SqrtImpl(vector<ll> a) {
    if (a.empty()) return {0};
    int z = QuadraticResidue(a[0], mod), n = a.size();
    if (z == -1) return {-1};
    vector<ll> q(1, z);
    const int inv2 = (mod + 1) / 2;
    for (int m = 1; m < n; m <= 1) {
        if (n < m * 2) a.resize(m * 2);
        q.resize(m * 2);
        vector<ll> f2 = Mul(q, q, m * 2);
        for (int i = 0; i < m * 2; ++i) f2[i] = sub(f2[i],
            a[i]);
        f2 = Mul(f2, Inverse(q), 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) {
    // O(NlogN), return {-1} if not exists
    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};
    vector<ll> 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> Derivative(vector<ll> a) {
    int n = a.size();
    vector<ll> res(n - 1);
    for (int i = 0; i < n - 1; ++i) res[i] = mul(a[i +
        1], i + 1);
    return res;
}
vector<ll> Integral(vector<ll> a) {
    int n = a.size();
    vector<ll> res(n + 1);
    for (int i = 0; i < n; ++i) {
        res[i + 1] = mul(a[i], mpow(i + 1, mod - 2));
    }
    return res;
}
vector<ll> Ln(vector<ll> a) {
    // O(NlogN), a[0] = 1
    int n = a.size();
    if (n == 1) return {0};
    vector<ll> d = Derivative(a);
    a.pop_back();
    return Integral(Mul(d, Inverse(a), n - 1));
}
vector<ll> Exp(vector<ll> a) {
    // O(NlogN), a[0] = 0
    int n = a.size();
    vector<ll> q(1, 1);
    a[0] = add(a[0], 1);
    for (int m = 1; m < n; m <= 1) {
        if (n < m * 2) a.resize(m * 2);
        vector<ll> g(a.begin(), a.begin() + m * 2), h(all(
            q));
        h.resize(m * 2), h = Ln(h);
        for (int i = 0; i < m * 2; ++i) {
            g[i] = sub(g[i], h[i]);
        }
        q = Mul(g, q, m * 2);
    }
    q.resize(n);
    return q;
}
vector<ll> Pow(vector<ll> a, ll k) {
    int n = a.size(), m = 0;

```

```

vector<ll> ans(n, 0);
while (m < n && a[m] == 0) m++;
if (k && m && (k >= n || k * m >= n)) return ans;
if (m == n) return ans[0] = 1, ans;
ll lead = m * k;
vector<ll> b(a.begin() + m, a.end());
ll base = mpow(b[0], k), inv = mpow(b[0], mod - 2);
for (int i = 0; i < n - m; ++i) b[i] = mul(b[i], inv);
b = Ln(b);
for (int i = 0; i < n - m; ++i) b[i] = mul(b[i], k % mod);
b = Exp(b);
for (int i = lead; i < n; ++i) ans[i] = mul(b[i - lead], base);
return ans;
}

vector<ll> Evaluate(vector<ll> a, vector<ll> x) {
    if (x.empty()) return {};
    int n = x.size();
    vector<vector<ll>> up(n * 2);
    for (int i = 0; i < n; ++i) up[i + n] = {sub(0, x[i]), 1};
    for (int i = n - 1; i > 0; --i) up[i] = Mul(up[i * 2], up[i * 2 + 1]);
    vector<vector<ll>> down(n * 2);
    down[1] = Divide(a, up[1]).second;
    for (int i = 2; i < n * 2; ++i) down[i] = Divide(down[i >> 1], up[i]).second;
    vector<ll> y(n);
    for (int i = 0; i < n; ++i) y[i] = down[i + n][0];
    return y;
}

vector<ll> Interpolate(vector<ll> x, vector<ll> y) {
    int n = x.size();
    vector<vector<ll>> up(n * 2);
    for (int i = 0; i < n; ++i) up[i + n] = {sub(0, x[i]), 1};
    for (int i = n - 1; i > 0; --i) up[i] = Mul(up[i * 2], up[i * 2 + 1]);
    vector<ll> a = Evaluate(Derivative(up[1]), x);
    for (int i = 0; i < n; ++i) {
        a[i] = mul(y[i], mpow(a[i], mod - 2));
    }
    vector<vector<ll>> down(n * 2);
    for (int i = 0; i < n; ++i) down[i + n] = {a[i]};
    for (int i = n - 1; i > 0; --i) {
        vector<ll> lhs = Mul(down[i * 2], up[i * 2 + 1]);
        vector<ll> rhs = Mul(down[i * 2 + 1], up[i * 2]);
        down[i].resize(lhs.size());
        for (int j = 0; j < lhs.size(); ++j) {
            down[i][j] = add(lhs[j], rhs[j]);
        }
    }
    return down[1];
}

```

7.4 Fast Linear Recursion

```

ll FastLinearRecursion(vector<ll> a, vector<ll> c, ll k) {
    // a_n = sigma c_j * a_{n-j-1}, 0-based
    // O(NlogNlogK), |a| = |c|
    int n = a.size();
    if (k < n) return a[k];
    vector<ll> base(n + 1, 1);
    for (int i = 0; i < n; ++i) base[i] = sub(0, c[n - i - 1]);
    vector<ll> poly(n);
    (n == 1 ? poly[0] = c[n - 1] : poly[1] = 1);
    auto calc = [&](vector<ll> p1, vector<ll> p2) {
        // O(n^2) brute force or O(nlogn) NTT
        return Divide(Mul(p1, p2), base).second;
    };
    vector<ll> res(n, 0); res[0] = 1;
    for (; k; k >>= 1, poly = calc(poly, poly)) {
        if (k & 1) res = calc(res, poly);
    }
    ll ans = 0;
    for (int i = 0; i < n; ++i) {
        (ans += res[i] * a[i]) %= mod;
    }
}

```

```

return ans;
}

```

7.5 Fast Walsh Transform

```

void fwt(vector<int> &a) {
    // and : a[j] += x;
    //      : a[j] -= x;
    // or  : a[j] ^ (1 << i) += y;
    //      : a[j] ^ (1 << i) -= y;
    // xor : a[j] = x - y, a[j] ^ (1 << i) = x + y;
    //      : a[j] = (x - y) / 2, a[j] ^ (1 << i) = (x + y) / 2;
    int n = __lg(a.size());
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < 1 << n; ++j) if (j >> i & 1) {
            int x = a[j ^ (1 << i)], y = a[j];
            // do something
        }
    }
}

vector<int> subs_conv(vector<int> a, vector<int> b) {
    // c_i = sum_{j & k = 0, j | k = i} a_j * b_k
    int n = __lg(a.size());
    vector<vector<int>> ha(n + 1, vector<int>(1 << n));
    vector<vector<int>> hb(n + 1, vector<int>(1 << n));
    vector<vector<int>> c(n + 1, vector<int>(1 << n));
    for (int i = 0; i < 1 << n; ++i) {
        ha[__builtin_popcount(i)][i] = a[i];
        hb[__builtin_popcount(i)][i] = b[i];
    }
    for (int i = 0; i <= n; ++i) fwt(ha[i]), fwt(hb[i]);
    for (int i = 0; i <= n; ++i)
        for (int j = 0; i + j <= n; ++j)
            for (int k = 0; k < 1 << n; ++k)
                // mind overflow
                c[i + j][k] += ha[i][k] * hb[j][k];
    for (int i = 0; i <= n; ++i)
        fwt(c[i], true);
    vector<int> ans(1 << n);
    for (int i = 0; i < 1 << n; ++i)
        ans[i] = c[__builtin_popcount(i)][i];
    return ans;
}

```

8 Geometry

8.1 Basic

```

const double eps = 1e-8, pi = acos(-1);
int sign(double x) {return abs(x) <= eps ? 0 : (x > 0 ? 1 : -1);}

struct Pt {
    double x, y;
    Pt (double _x, double _y) : x(_x), y(_y) {}
    Pt operator + (Pt o) {return Pt(x + o.x, y + o.y);}
    Pt operator - (Pt o) {return Pt(x - o.x, y - o.y);}
    Pt operator * (double k) {return Pt(x * k, y * k);}
    Pt operator / (double k) {return Pt(x / k, y / k);}
    double operator * (Pt o) {return x * o.x + y * o.y;}
    double operator ^ (Pt o) {return x * o.y - y * o.x;}
};

struct Line {
    Pt a, b;
};

struct Cir {
    Pt o; double r;
};

double abs2(Pt o) {return o.x * o.x + o.y * o.y;}
double abs(Pt o) {return sqrt(abs2(o));}
int ori(Pt o, Pt a, Pt b) {return sign((o - a) ^ (o - b));}

bool btw(Pt a, Pt b, Pt c) { // c on segment ab?
    return ori(a, b, c) == 0 && sign((c - a) * (c - b)) <= 0;
}

double area(Pt a, Pt b, Pt c) {return abs((a - b) ^ (a - c)) / 2;}

Pt unit(Pt o) {return o / abs(o);}
Pt rot(Pt a, double o) { // CCW
    double c = cos(o), s = sin(o);
}

```

```

    return Pt(c * a.x - s * a.y, s * a.x + c * a.y);
}
Pt proj_vector(Pt a, Pt b, Pt c) { // vector ac proj to
    ab
    return (b - a) * ((c - a) * (b - a)) / ((b - a) * (b
        - a));
}
Pt proj_pt(Pt a, Pt b, Pt c) { // point c proj to ab
    return proj_vector(a, b, c) + a;
}

```

8.2 Heart

```

Pt circenter(Pt p0, Pt p1, Pt p2) { // radius = abs(
    center)
    p1 = p1 - p0, p2 = p2 - p0;
    double x1 = p1.x, y1 = p1.y, x2 = p2.x, y2 = p2.y;
    double m = 2. * (x1 * y2 - y1 * x2);
    Pt center(0, 0);
    center.x = (x1 * x1 * y2 - x2 * x2 * y1 + y1 * y2 * (
        y1 - y2)) / m;
    center.y = (x1 * x2 * (x2 - x1) - y1 * y1 * x2 + x1 *
        y2 * y2) / m;
    return center + p0;
}
Pt incenter(Pt p1, Pt p2, Pt p3) { // radius = area / s
    * 2
    double a = abs(p2 - p3), b = abs(p1 - p3), c = abs(p1
        - p2);
    double s = a + b + c;
    return (p1 * a + p2 * b + p3 * c) / s;
}
Pt masscenter(Pt p1, Pt p2, Pt p3)
{ return (p1 + p2 + p3) / 3; }
Pt orthocenter(Pt p1, Pt p2, Pt p3)
{ return masscenter(p1, p2, p3) * 3 - circenter(p1, p2,
    p3) * 2; }

```

8.3 External Bisector

```

Pt external_bisector(Pt p1, Pt p2, Pt p3) { //213
    Pt L1 = p2 - p1, L2 = p3 - p1;
    L2 = L2 * abs(L1) / abs(L2);
    return L1 + L2;
}

```

8.4 Intersection of Segments

```

Pt LinesInter(Line a, Line b) {
    double abc = (a.b - a.a) ^ (b.a - a.a);
    double abd = (a.b - a.a) ^ (b.b - a.a);
    if (sign(abc - abd) == 0) return b.b; // no inter
    return (b.b * abc - b.a * abd) / (abc - abd);
}

vector<Pt> SegsInter(Line a, Line b) {
    if (btw(a.a, a.b, b.a)) return {b.a};
    if (btw(a.a, a.b, b.b)) return {b.b};
    if (btw(b.a, b.b, a.a)) return {a.a};
    if (btw(b.a, b.b, a.b)) return {a.b};
    if (ori(a.a, a.b, b.a) * ori(a.a, a.b, b.b) == -1 &&
        ori(b.a, b.b, a.a) * ori(b.a, b.b, a.b) == -1)
        return {LinesInter(a, b)};
    return {};
}

```

8.5 Intersection of Circle and Line

```

vector<Pt> CircleLineInter(Cir c, Line l) {
    Pt p = 1.a + (1.b - 1.a) * ((c.o - 1.a) * (1.b - 1.a)
        ) / abs2(1.b - 1.a);
    double s = (1.b - 1.a) ^ (c.o - 1.a), h2 = c.r * c.r
        - s * s / abs2(1.b - 1.a);
    if (sign(h2) == -1) return {};
    if (sign(h2) == 0) return {p};
    Pt h = (1.b - 1.a) / abs(1.b - 1.a) * sqrt(h2);
    return {p - h, p + h};
}

```

8.6 Intersection of Circles

```

vector<Pt> CirclesInter(Cir c1, Cir c2) {
    double d2 = abs2(c1.o - c2.o), d = sqrt(d2);
    if (d < max(c1.r, c2.r) - min(c1.r, c2.r) || d > c1.r
        + c2.r) return {};
    Pt u = (c1.o + c2.o) / 2 + (c1.o - c2.o) * ((c2.r *
        c2.r - c1.r * c1.r) / (2 * d2));
    double A = sqrt(((c1.r + c2.r + d) * (c1.r - c2.r + d)
        * (c1.r + c2.r - d) * (-c1.r + c2.r + d)));
    Pt v = Pt(c1.o.y - c2.o.y, -c1.o.x + c2.o.x) * A / (2
        * d2);
    if (sign(v.x) == 0 && sign(v.y) == 0) return {u};
    return {u + v, u - v};
}

```

8.7 Intersection of Polygon and Circle

```

double _area(Pt pa, Pt pb, double r){
    if(abs(pa) < abs(pb)) swap(pa, pb);
    if(abs(pb) < eps) return 0;
    double S, h, theta;
    double a = abs(pb), b = abs(pa), c = abs(pb - pa);
    double cosB = pb * (pb - pa) / a / c, B = acos(cosB);
    double cosC = (pa * pb) / a / b, C = acos(cosC);
    if (a > r) {
        S = (C / 2) * r * r;
        h = a * b * sin(C) / c;
        if (h < r && B < pi / 2) S -= (acos(h / r) * r * r
            - h * sqrt(r * r - h * h));
    } else if (b > r) {
        theta = pi - B - asin(sin(B) / r * a);
        S = .5 * a * r * sin(theta) + (C - theta) / 2 * r *
            r;
    } else
        S = .5 * sin(C) * a * b;
    return S;
}

double area_poly_circle(vector<Pt> poly, Pt O, double r
    ) {
    double S = 0; int n = poly.size();
    for(int i = 0; i < n; ++i)
        S += _area(poly[i] - O, poly[(i + 1) % n] - O, r) *
            ori(O, poly[i], poly[(i + 1) % n]);
    return fabs(S);
}

```

8.8 Tangent Lines of Circle and Point

```

vector<Line> tangent(Cir c, Pt p) {
    vector<Line> z;
    double d = abs(p - c.o);
    if (sign(d - c.r) == 0) {
        Pt i = rot(p - c.o, pi / 2);
        z.push_back({p, p + i});
    } else if (d > c.r) {
        double o = acos(c.r / d);
        Pt i = unit(p - c.o), j = rot(i, o) * c.r, k = rot(
            i, -o) * c.r;
        z.push_back({c.o + j, p});
        z.push_back({c.o + k, p});
    }
    return z;
}

```

8.9 Tangent Lines of Circles

```

vector<Line> tangent(Cir a, Cir b) {
#define Pij \
    Pt i = unit(b.o - a.o) * a.r, j = Pt(i.y, -i.x);\
    z.push_back({a.o + i, a.o + i + j});
#define deo(I,J) \
    double d = abs(a.o - b.o), e = a.r I b.r, o = acos(e
        / d);\
    Pt i = unit(b.o - a.o), j = rot(i, o), k = rot(i, -o)
        ;\
    z.push_back({a.o + j * a.r, b.o J j * b.r});\
    z.push_back({a.o + k * a.r, b.o J k * b.r});
    if (a.r < b.r) swap(a, b);
    vector<Line> z;
    if (abs(a.o - b.o) + b.r < a.r) return z;
    else if (sign(abs(a.o - b.o) + b.r - a.r) == 0) { Pij
        ; }
    else {

```

```

    deo(+,+); // inter
    // outer
    if (sign(d - a.r - b.r) == 0) { Pij; }
    else if (d > a.r + b.r) { deo(+,-); }
}
return z;
}

```

8.10 Point In Convex

```

bool PointInConvex(const vector<Pt> &C, Pt p, bool
    strict = true) {
    int a = 1, b = int(C.size()) - 1, r = !strict;
    if (C.size() == 0) return false;
    if (C.size() < 3) return r && btw(C[0], C.back(), p);
    if (ori(C[0], C[a], C[b]) > 0) swap(a, b);
    if (ori(C[0], C[a], p) >= r || ori(C[0], C[b], p) <=
        -r)
        return false;
    while (abs(a - b) > 1) {
        int c = (a + b) / 2;
        (ori(C[0], C[c], p) > 0 ? b : a) = c;
    }
    return ori(C[a], C[b], p) < r;
}

```

8.11 Point Segment Distance

```

double PointSegDist(Pt q0, Pt q1, Pt p) {
    if (sign(abs(q0 - q1)) == 0) return abs(q0 - p);
    if (sign((q1 - q0) * (p - q0)) >= 0 && sign((q0 - q1)
        * (p - q1)) >= 0)
        return fabs(((q1 - q0) ^ (p - q0)) / abs(q0 - q1));
    return min(abs(p - q0), abs(p - q1));
}

```

8.12 Convex Hull

```

vector<Pt> ConvexHull(vector<Pt> pt) {
    int n = pt.size();
    sort(all(pt), [&](Pt a, Pt b) {return a.x == b.x ? a.
        y < b.y : a.x < b.x;});
    vector<Pt> ans = {pt[0]};
    for (int t : {0, 1}) {
        int m = ans.size();
        for (int i = 1; i < n; ++i) {
            while (ans.size() > m && ori(ans[ans.size() - 2],
                ans.back(), pt[i]) <= 0)
                ans.pop_back();
            ans.push_back(pt[i]);
        }
        reverse(all(pt));
    }
    ans.pop_back();
    return ans;
}

```

8.13 Convex Hull Distance

```

double ConvexHullDist(vector<Pt> A, vector<Pt> B) {
    for (auto &p : B) p = Pt(0, 0) - p;
    auto C = Minkowski(A, B); // assert SZ(C) > 0
    if (PointInConvex(C, Pt(0, 0))) return 0;
    double ans = PointSegDist(C.back(), C[0], Pt(0, 0));
    for (int i = 0; i + 1 < C.size(); ++i) {
        ans = min(ans, PointSegDist(C[i], C[i + 1], Pt
            (0, 0)));
    }
    return ans;
}

```

8.14 Minimum Enclosing Circle

```

Cir min_enclosing(vector<Pt> &p) {
    random_shuffle(p.begin(), p.end());
    double r = 0.0;
    Pt cent = p[0];
    for (int i = 1; i < p.size(); ++i) {
        if (abs2(cent - p[i]) <= r) continue;
        cent = p[i];
        r = 0.0;
    }
}

```

```

    for (int j = 0; j < i; ++j) {
        if (abs2(cent - p[j]) <= r) continue;
        cent = (p[i] + p[j]) / 2;
        r = abs2(p[j] - cent);
        for (int k = 0; k < j; ++k) {
            if (abs2(cent - p[k]) <= r) continue;
            cent = circenter(p[i], p[j], p[k]);
            r = abs2(p[k] - cent);
        }
    }
    return {cent, sqrt(r)};
}

```

8.15 Union of Circles

```

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);
    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((sqrt(a.r) + sqrt(d) - sqrt(b.r)) /
            (2 * a.r * d)), z = atan2((b.o - a.o).y, (b.o
            - a.o).x);
        if (z < 0) z += 2 * pi;
        double l = z - o, r = z + o;
        if (l < 0) l += 2 * pi;
        if (r > 2 * pi) r -= 2 * pi;
        if (l > r) res.emplace_back(l, 2 * pi), res.
            emplace_back(0, r);
        else res.emplace_back(l, r);
    }
    return res;
}

double CircleUnionArea(vector<Cir> c) { // circle
    // should be identical
    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);
        }
        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);
        }
    }
    return a * 0.5;
}

```

8.16 Polar Angle Sort

```

void PolarAngleSort(vector<Pt> &pts) {
    auto pos = [&](Pt a) {return sign(a.y) == 0 ? sign(a
        .x) < 0 : sign(a.y) > 0;};
    sort(all(pts), [&](Pt a, Pt b) {return pos(a) == pos(
        b) ? sign(a ^ b) > 0 : pos(a) < pos(b);});
}

```

8.17 Rotating Caliper

```

void RotatingCaliper(vector<Pt> &pts) {
    int n = pts.size();
    for (int i = 0, j = 2; i < n; ++i) {
        int ni = (i + 1) % n;
        while (true) {
            int nj = (j + 1) % n;
            if (area(pts[j], pts[i], pts[ni]) < area(pts[nj],
                pts[i], pts[ni])) {
                j = nj;
            } else {
                break;
            }
        }
    }
}

```



```

    // do something
}
}

```

8.18 Rotating SweepLine

```

void RotatingSweepLine(vector<Pt> &pt) {
    int n = pt.size();
    vector<int> id(n), pos(n);
    vector<pair<int, int>> line;
    for (int i = 0; i < n; ++i) for (int j = 0; j < n; ++j)
        if (i ^ j) line.emplace_back(i, j);
    sort(line.begin(), line.end(), [&](pair<int, int> i,
        pair<int, int> j) {
        Pt a = pt[i.second] - pt[i.first], b = pt[j.second]
            - pt[j.first];
        return (a.pos() == b.pos() ? sign(a ^ b) > 0 : a.
            pos() < b.pos());
    });
    iota(id.begin(), id.end(), 0);
    sort(id.begin(), id.end(), [&](int i, int j) {
        return (sign(pt[i].y - pt[j].y) == 0 ? pt[i].x < pt
            [j].x : pt[i].y < pt[j].y);
    });
    for (int i = 0; i < n; ++i)
        pos[id[i]] = i;
    for (auto [i, j] : line) {
        // point sort by the distance to line(i, j)
        // do something.
        tie(pos[i], pos[j], id[pos[i]], id[pos[j]]) =
            make_tuple(pos[j], pos[i], j, i);
    }
}

```

8.19 Half Plane Intersection

```

vector<Pt> HalfPlaneInter(vector<pair<Pt, Pt>> vec)
{
    // x
    // first -----> second
    auto pos = [&](Pt a) {return sign(a.y) == 0 ? sign(a
        .x) < 0 : sign(a.y) > 0;};
    sort(all(vec), [&](pair<Pt, Pt> a, pair<Pt, Pt> b)
        {
            Pt A = a.second - a.first, B = b.second - b.first;
            if (pos(A) == pos(B)) {
                if (sign(A ^ B) == 0) return sign((b.first - a.
                    first) * (b.second - a.first)) > 0;
                return sign(A ^ B) > 0;
            }
            return pos(A) < pos(B);
        });
    deque<Pt> inter;
    deque<pair<Pt, Pt>> seg;
    int n = vec.size();
    auto get = [&](pair<Pt, Pt> a, pair<Pt, Pt> b) {
        return intersect(a.first, a.second, b.first, b.
            second);
    };
    for (int i = 0; i < n; ++i) if (!i || vec[i] != vec[i
        - 1]) {
        while (seg.size() >= 2 && sign((vec[i].second -
            inter.back()) ^ (vec[i].first - inter.back()))
            == 1) seg.pop_back(), inter.pop_back();
        while (seg.size() >= 2 && sign((vec[i].second -
            inter.front()) ^ (vec[i].first - inter.front()))
            == 1) seg.pop_front(), inter.pop_front();
        seg.push_back(vec[i]);
        if (seg.size() >= 2) inter.pb(get(seg[seg.size() -
            2], seg.back()));
    }
    while (seg.size() >= 2 && sign((seg.front().second -
        inter.back()) ^ (seg.front().first - inter.back())
        ) == 1) seg.pop_back(), inter.pop_back();
    inter.push_back(get(seg.front(), seg.back()));
    return vector<Pt>(all(inter));
}

```

8.20 Minkowski Sum

```

void reorder(vector<Pt> &P) {
    rotate(P.begin(), min_element(all(P), [&](Pt a, Pt b)
        { return make_pair(a.y, a.x) < make_pair(b.y, b.
            x); })), P.end());
}

```

```

}
vector<Pt> Minkowski(vector<Pt> P, vector<Pt> Q) {
    // P, Q: convex polygon
    reorder(P), reorder(Q);
    int n = P.size(), m = Q.size();
    P.pb(P[0]), P.pb(P[1]), Q.pb(Q[0]), Q.pb(Q[1]);
    vector<Pt> ans;
    for (int i = 0, j = 0; i < n || j < m; ) {
        ans.pb(P[i] + Q[j]);
        auto val = (P[i + 1] - P[i]) ^ (Q[j + 1] - Q[j]);
        if (val >= 0) i++;
        if (val <= 0) j++;
    }
    return ans;
}

```

8.21 Delaunay Triangulation

```

/* Delaunay Triangulation:
Given a sets of points in 2D plane, find a
triangulation such that no points will strictly
inside circumcircle of any triangle.
find : return a triangle contain given point
add_point : add a point into triangulation
A Triangle is in triangulation iff. its has_chd is 0.
Region of triangle u: iterate each u.edge[i].tri,
each points are u.p[(i+1)%3], u.p[(i+2)%3]
Voronoi diagram: for each triangle in triangulation,
the bisector of all its edges will split the region.
nearest point will belong to the triangle containing it
*/
const ll inf = MAXC * MAXC * 100; // Lower_bound
unknown
struct Tri;
struct Edge {
    Tri* tri; int side;
    Edge(): tri(0), side(0){}
    Edge(Tri* _tri, int _side): tri(_tri), side(_side){}
};
struct Tri {
    pll p[3];
    Edge edge[3];
    Tri* chd[3];
    Tri() {}
    Tri(const pll& p0, const pll& p1, const pll& p2) {
        p[0] = p0; p[1] = p1; p[2] = p2;
        chd[0] = chd[1] = chd[2] = 0;
    }
    bool has_chd() const { return chd[0] != 0; }
    int num_chd() const {
        return !!chd[0] + !!chd[1] + !!chd[2];
    }
    bool contains(pll const& q) const {
        for (int i = 0; i < 3; ++i)
            if (ori(p[i], p[(i + 1) % 3], q) < 0)
                return 0;
        return 1;
    }
} pool[N * 10], *tris;
void edge(Edge a, Edge b) {
    if(a.tri) a.tri->edge[a.side] = b;
    if(b.tri) b.tri->edge[b.side] = a;
}
struct Trig { // Triangulation
    Trig() {
        the_root = // Tri should at least contain all
            points
            new(tris++) Tri(pll(-inf, -inf), pll(inf + inf, -
                inf), pll(-inf, inf + inf));
    }
    Tri* find(pll p) { return find(the_root, p); }
    void add_point(const pll &p) { add_point(find(
        the_root, p), p); }
    Tri* the_root;
    static Tri* find(Tri* root, const pll &p) {
        while (1) {
            if (!root->has_chd())
                return root;
            for (int i = 0; i < 3 && root->chd[i]; ++i)
                if (root->chd[i]->contains(p)) {
                    root = root->chd[i];
                    break;
                }
        }
    }
}

```

```

    }
}
assert(0); // "point not found"
}
void add_point(Tri* root, pll const& p) {
    Tri* t[3];
    /* split it into three triangles */
    for (int i = 0; i < 3; ++i)
        t[i] = new(tris++) Tri(root->p[i], root->p[(i + 1) % 3], p);
    for (int i = 0; i < 3; ++i)
        edge(Edge(t[i], 0), Edge(t[(i + 1) % 3], 1));
    for (int i = 0; i < 3; ++i)
        edge(Edge(t[i], 2), root->edge[(i + 2) % 3]);
    for (int i = 0; i < 3; ++i)
        root->chd[i] = t[i];
    for (int i = 0; i < 3; ++i)
        flip(t[i], 2);
}
void flip(Tri* tri, int pi) {
    Tri* trj = tri->edge[pi].tri;
    int pj = tri->edge[pi].side;
    if (!trj) return;
    if (!in_cc(tri->p[0], tri->p[1], tri->p[2], trj->p[pj])) return;
    /* flip edge between tri, trj */
    Tri* trk = new(tris++) Tri(tri->p[(pi + 1) % 3], trj->p[pj], tri->p[pi]);
    Tri* trl = new(tris++) Tri(trj->p[(pj + 1) % 3], tri->p[pi], trj->p[pj]);
    edge(Edge(trk, 0), Edge(trl, 0));
    edge(Edge(trk, 1), tri->edge[(pi + 2) % 3]);
    edge(Edge(trk, 2), trj->edge[(pj + 1) % 3]);
    edge(Edge(trl, 1), trj->edge[(pj + 2) % 3]);
    edge(Edge(trl, 2), tri->edge[(pi + 1) % 3]);
    tri->chd[0] = trk; tri->chd[1] = trl; tri->chd[2] = 0;
    trj->chd[0] = trk; trj->chd[1] = trl; trj->chd[2] = 0;
    flip(trk, 1); flip(trk, 2);
    flip(trl, 1); flip(trl, 2);
}
};
vector<Tri*> triang; // vector of all triangle
set<Tri*> vst;
void go(Tri* now) { // store all tri into triang
    if (vst.find(now) != vst.end())
        return;
    vst.insert(now);
    if (!now->has_chd())
        return triang.pb(now);
    for (int i = 0; i < now->num_chd(); ++i)
        go(now->chd[i]);
}
void build(int n, pll* ps) { // build triangulation
    tris = pool; triang.clear(); vst.clear();
    random_shuffle(ps, ps + n);
    Trig tri; // the triangulation structure
    for (int i = 0; i < n; ++i)
        tri.add_point(ps[i]);
    go(tri.the_root);
}

```

8.22 Triangulation Voronoi

```

vector<Line> ls[N];
pll arr[N];
Line make_line(pdd p, Line l) {
    pdd d = l.Y - l.X; d = perp(d);
    pdd m = (l.X + l.Y) / 2;
    l = Line(m, m + d);
    if (ori(l.X, l.Y, p) < 0)
        l = Line(m + d, m);
    return l;
}
double calc_area(int id) {
    // use to calculate the area of point "strictly in the convex hull"
    vector<Line> hpi = halfPlaneInter(ls[id]);
    vector<pdd> ps;
    for (int i = 0; i < SZ(hpi); ++i)

```

```

        ps.pb(intersect(hpi[i].X, hpi[i].Y, hpi[(i + 1) % SZ(hpi)].X, hpi[(i + 1) % SZ(hpi)].Y));
    double rt = 0;
    for (int i = 0; i < SZ(ps); ++i)
        rt += cross(ps[i], ps[(i + 1) % SZ(ps)]);
    return fabs(rt) / 2;
}
void solve(int n, pii *oarr) {
    map<pll, int> mp;
    for (int i = 0; i < n; ++i)
        arr[i] = pll(oarr[i].X, oarr[i].Y), mp[arr[i]] = i;
    build(n, arr); // Triangulation
    for (auto *t : triang) {
        vector<int> p;
        for (int i = 0; i < 3; ++i)
            if (mp.find(t->p[i]) != mp.end())
                p.pb(mp[t->p[i]]);
        for (int i = 0; i < SZ(p); ++i)
            for (int j = i + 1; j < SZ(p); ++j) {
                Line l(oarr[p[i]], oarr[p[j]]);
                ls[p[i]].pb(make_line(oarr[p[i]], l));
                ls[p[j]].pb(make_line(oarr[p[j]], l));
            }
    }
}

```

9 Else

9.1 Bit Hack

```

long long next_perm(long long v) {
    long long t = v | (v - 1);
    return (t + 1) | (((~t & -~t) - 1) >> (__builtin_ctz(v) + 1));
}
void subset(long long s) {
    long long sub = s;
    while (sub) sub = (sub - 1) & s;
}

```

9.2 Dynamic Programming Condition

9.2.1 Totally Monotone (Concave/Convex)

$$\forall i < i', j < j', B[i][j] \leq B[i'][j] \implies B[i][j'] \leq B[i'][j']$$

$$\forall i < i', j < j', B[i][j] \geq B[i'][j] \implies B[i][j'] \geq B[i'][j']$$

9.2.2 Monge Condition (Concave/Convex)

$$\forall i < i', j < j', B[i][j] + B[i'][j'] \geq B[i][j'] + B[i'][j]$$

$$\forall i < i', j < j', B[i][j] + B[i'][j'] \leq B[i][j'] + B[i'][j]$$

9.2.3 Optimal Split Point

If

$$B[i][j] + B[i+1][j+1] \geq B[i][j+1] + B[i+1][j]$$

then

$$H_{i,j-1} \leq H_{i,j} \leq H_{i+1,j}$$

9.3 Slope Trick

```

template<typename T>
struct slope_trick_convex {
    T minn = 0, ground_l = 0, ground_r = 0;
    priority_queue<T, vector<T>, less<T>> left;
    priority_queue<T, vector<T>, greater<T>> right;
    slope_trick_convex() {left.push(numeric_limits<T>::min() / 2); right.push(numeric_limits<T>::max() / 2);}
    void push_left(T x) {left.push(x - ground_l);}
    void push_right(T x) {right.push(x - ground_r);}
    //add a line with slope 1 to the right starting from x
    void add_right(T x) {
        T l = left.top() + ground_l;
        if (l <= x) push_right(x);
        else push_left(x), push_right(l), left.pop(), minn += 1 - x;
    }
    //add a line with slope -1 to the left starting from x
    void add_left(T x) {
        T r = right.top() + ground_r;

```

```

    if (r >= x) push_left(x);
    else push_right(x), push_left(r), right.pop(), minn
        += x - r;
}
//val[i]=min(val[j]) for all i-l<=j<=i+r
void expand(T l, T r) {ground_l -= l, ground_r += r;}
void shift_up(T x) {minn += x;}
T get_val(T x) {
    T l = left.top() + ground_l, r = right.top() +
        ground_r;
    if (x >= l && x <= r) return minn;
    if (x < l) {
        vector<T> trash;
        T cur_val = minn, slope = 1, res;
        while (1) {
            trash.push_back(left.top());
            left.pop();
            if (left.top() + ground_l <= x) {
                res = cur_val + slope * (l - x);
                break;
            }
            cur_val += slope * (l - (left.top() + ground_l));
            l = left.top() + ground_l;
            slope += 1;
        }
        for (auto i : trash) left.push(i);
        return res;
    }
    if (x > r) {
        vector<T> trash;
        T cur_val = minn, slope = 1, res;
        while (1) {
            trash.push_back(right.top());
            right.pop();
            if (right.top() + ground_r >= x) {
                res = cur_val + slope * (x - r);
                break;
            }
            cur_val += slope * ((right.top() + ground_r) -
                r);
            r = right.top() + ground_r;
            slope += 1;
        }
        for (auto i : trash) right.push(i);
        return res;
    }
    assert(0);
}
};

```

9.4 Manhattan MST

```

void solve(int n) {
    init();
    vector<int> v(n), ds;
    for (int i = 0; i < n; ++i) {
        v[i] = i;
        ds.push_back(x[i] - y[i]);
    }
    sort(ds.begin(), ds.end());
    ds.resize(unique(ds.begin(), ds.end()) - ds.begin());
    sort(v.begin(), v.end(), [&](int i, int j) { return x
        [i] == x[j] ? y[i] > y[j] : x[i] > x[j]; });
    int j = 0;
    for (int i = 0; i < n; ++i) {
        int p = lower_bound(ds.begin(), ds.end(), x[v[i]] -
            y[v[i]]) - ds.begin() + 1;
        pair<int, int> q = query(p);
        // query return prefix minimum
        if (~q.second) add_edge(v[i], q.second);
        add(p, make_pair(x[v[i]] + y[v[i]], v[i]));
    }
}
void make_graph() {
    solve(n);
    for (int i = 0; i < n; ++i) swap(x[i], y[i]);
    solve(n);
    for (int i = 0; i < n; ++i) x[i] = -x[i];
    solve(n);
    for (int i = 0; i < n; ++i) swap(x[i], y[i]);
    solve(n);
}

```

```

}

```

9.5 Dynamic MST

```

int cnt[maxn], cost[maxn], st[maxn], ed[maxn];
pair<int, int> qr[maxn];
// qr[i].first = id of edge to be changed, qr[i].second
// = weight after operation
// cnt[i] = number of operation on edge i
// call solve(0, q - 1, v, 0), where v contains edges i
// such that cnt[i] == 0

void contract(int l, int r, vector<int> v, vector<int>
    &x, vector<int> &y) {
    sort(v.begin(), v.end(), [&](int i, int j) {
        if (cost[i] == cost[j]) return i < j;
        return cost[i] < cost[j];
    });
    djs.save();
    for (int i = l; i <= r; ++i) djs.merge(st[qr[i].first],
        ed[qr[i].first]);
    for (int i = 0; i < (int)v.size(); ++i) {
        if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
            x.push_back(v[i]);
            djs.merge(st[v[i]], ed[v[i]]);
        }
    }
    djs.undo();
    djs.save();
    for (int i = 0; i < (int)x.size(); ++i) djs.merge(st[
        x[i]], ed[x[i]]);
    for (int i = 0; i < (int)v.size(); ++i) {
        if (djs.find(st[v[i]]) != djs.find(ed[v[i]])) {
            y.push_back(v[i]);
            djs.merge(st[v[i]], ed[v[i]]);
        }
    }
    djs.undo();
}

void solve(int l, int r, vector<int> v, long long c) {
    if (l == r) {
        cost[qr[l].first] = qr[l].second;
        if (st[qr[l].first] == ed[qr[l].first]) {
            printf("%lld\n", c);
            return;
        }
        int minv = qr[l].second;
        for (int i = 0; i < (int)v.size(); ++i) minv = min(
            minv, cost[v[i]]);
        printf("%lld\n", c + minv);
        return;
    }
    int m = (l + r) >> 1;
    vector<int> lv = v, rv = v;
    vector<int> x, y;
    for (int i = m + 1; i <= r; ++i) {
        cnt[qr[i].first]--;
        if (cnt[qr[i].first] == 0) lv.push_back(qr[i].first);
    }
    contract(l, m, lv, x, y);
    long long lc = c, rc = c;
    djs.save();
    for (int i = 0; i < (int)x.size(); ++i) {
        lc += cost[x[i]];
        djs.merge(st[x[i]], ed[x[i]]);
    }
    solve(l, m, y, lc);
    djs.undo();
    x.clear(), y.clear();
    for (int i = m + 1; i <= r; ++i) cnt[qr[i].first]++;
    for (int i = l; i <= m; ++i) {
        cnt[qr[i].first]--;
        if (cnt[qr[i].first] == 0) rv.push_back(qr[i].first);
    }
    contract(m + 1, r, rv, x, y);
    djs.save();
    for (int i = 0; i < (int)x.size(); ++i) {
        rc += cost[x[i]];
        djs.merge(st[x[i]], ed[x[i]]);
    }
}

```

```

    }
    solve(m + 1, r, y, rc);
    djs.undo();
    for (int i = 1; i <= m; ++i) cnt[qr[i].first]++;
}

```

9.6 ALL LCS

```

void all_lcs(string s, string t) { // 0-base
    vector<int> h(t.size());
    iota(all(h), 0);
    for (int a = 0; a < s.size(); ++a) {
        int v = -1;
        for (int c = 0; c < t.size(); ++c)
            if (s[a] == t[c] || h[c] < v)
                swap(h[c], v);
        // LCS(s[0, a], t[b, c]) =
        // c - b + 1 - sum([h[i] >= b] | i <= c)
        // h[i] might become -1 !!
    }
}

```

9.7 Hilbert Curve

```

long long hilbertOrder(int x, int y, int pow, int
    rotate) {
    if (pow == 0) return 0;
    int hpow = 1 << (pow-1);
    int seg = (x < hpow) ? ((y < hpow) ? 0 : 3) : ((y <
        hpow) ? 1 : 2);
    seg = (seg + rotate) & 3;
    const int rotateDelta[4] = {3, 0, 0, 1};
    int nx = x & (x ^ hpow), ny = y & (y ^ hpow);
    int nrot = (rotate + rotateDelta[seg]) & 3;
    long long subSquareSize = 1ll << (pow * 2 - 2);
    long long ans = seg * subSquareSize;
    long long add = hilbertOrder(nx, ny, pow - 1, nrot);
    ans += (seg == 1 || seg == 2) ? add : (subSquareSize
        - add - 1);
    return ans;
}

```

9.8 Pbds

```

#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
#include <ext/rope>
using namespace __gnu_cxx;
int main () {
    __gnu_pbds::priority_queue<int> pq1, pq2;
    pq1.join(pq2); // pq1 += pq2, pq2 = {}
    cc_hash_table<int, int> m1;
    tree<int, null_type, less<int>, rb_tree_tag,
        tree_order_statistics_node_update> oset;
    oset.insert(2), oset.insert(4);
    cout << *oset.find_by_order(1) << ' ' << oset.
        order_of_key(1) << '\n'; // 4 0
    bitset<100> BS;
    BS.flip(3), BS.flip(5);
    cout << BS._Find_first() << ' ' << BS._Find_next(3)
        << '\n'; // 3 5
    rope<int> rp1, rp2;
    rp1.push_back(1), rp1.push_back(3);
    rp1.insert(0, 2); // pos, num
    rp1.erase(0, 2); // pos, Len
    rp1.substr(0, 2); // pos, Len
    rp2.push_back(4);
    rp1 += rp2, rp2 = rp1;
    cout << rp2[0] << ' ' << rp2[1] << '\n'; // 3 4
}

```

9.9 Random

```

struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t a) const {

```

```

        static const uint64_t FIXED_RANDOM = chrono::
            steady_clock::now().time_since_epoch().count();
        return splitmix64(i + FIXED_RANDOM);
    }
};
unordered_map<int, int, custom_hash> m1;
random_device rd; mt19937 rng(rd());

```

9.10 Smawk Algorithm

```

ll query(int l, int r) {
    // ...
}
struct SMAWK {
    // Condition:
    // If M[l][0] < M[l][1] then M[0][0] < M[0][1]
    // If M[l][0] == M[l][1] then M[0][0] <= M[0][1]
    // For all i, find r_i s.t. M[i][r_i] is maximum //
        minimum.
    int ans[N], tmp[N];
    void interpolate(vector<int> l, vector<int> r) {
        int n = l.size(), m = r.size();
        vector<int> nl;
        for (int i = 1; i < n; i += 2) {
            nl.push_back(l[i]);
        }
        run(nl, r);
        for (int i = 1, j = 0; i < n; i += 2) {
            while (j < m && r[j] < ans[l[i]])
                j++;
            assert(j < m && ans[l[i]] == r[j]);
            tmp[l[i]] = j;
        }
        for (int i = 0; i < n; i += 2) {
            int curl = 0, curr = m - 1;
            if (i)
                curl = tmp[l[i - 1]];
            if (i + 1 < n)
                curr = tmp[l[i + 1]];
            ll res = query(l[i], r[curl]);
            ans[l[i]] = r[curl];
            for (int j = curl + 1; j <= curr; ++j) {
                ll nxt = query(l[i], r[j]);
                if (res < nxt)
                    res = nxt, ans[l[i]] = r[j];
            }
        }
    }
    void reduce(vector<int> l, vector<int> r) {
        int n = l.size(), m = r.size();
        vector<int> nr;
        for (int j : r) {
            while (!nr.empty()) {
                int i = nr.size() - 1;
                if (query(l[i], nr.back()) <= query(l[i], j))
                    nr.pop_back();
                else
                    break;
            }
            if (nr.size() < n)
                nr.push_back(j);
        }
        run(l, nr);
    }
    void run(vector<int> l, vector<int> r) {
        int n = l.size(), m = r.size();
        if (max(n, m) <= 2) {
            for (int i : l) {
                ans[i] = r[0];
                if (m > 1) {
                    if (query(i, r[0]) < query(i, r[1]))
                        ans[i] = r[1];
                }
            }
        } else if (n >= m) {
            interpolate(l, r);
        } else {
            reduce(l, r);
        }
    }
};

```

9.11 Two Dimension Add Sum*

```
struct TwoDimensionAddAndSum {
    // 0-index, [l, r)
    struct Seg {
        int l, r, m;
        ll vala, valb, lza, lzb;
        Seg* ch[2];
        Seg(int _l, int _r) : l(_l), r(_r), m(l + r >> 1),
            vala(0), valb(0), lza(0), lzb(0) {
            if (r - l > 1) {
                ch[0] = new Seg(l, m);
                ch[1] = new Seg(m, r);
            }
        }
        void pull() { vala = ch[0]->vala + ch[1]->vala, valb
            = ch[0]->valb + ch[1]->valb; }
        void give(ll a, ll b) {
            lza += a, lzb += b;
            vala += a * (r - l), valb += b * (r - l);
        }
        void push() {
            ch[0]->give(lza, lzb), ch[1]->give(lza, lzb), lza
                = lzb = 0;
        }
        void add(int a, int b, ll va, ll vb) {
            if (a <= l && r <= b)
                give(va, vb);
            else {
                push();
                if (a < m) ch[0]->add(a, b, va, vb);
                if (m < b) ch[1]->add(a, b, va, vb);
                pull();
            }
        }
        long long query(int a, int b, int v) {
            if (a <= l && r <= b) return vala * v + valb;
            push();
            long long ans = 0;
            if (a < m) ans += ch[0]->query(a, b, v);
            if (m < b) ans += ch[1]->query(a, b, v);
            return ans;
        }
    };
    // note integer overflow.
    vector<array<int, 4>> E[N];
    vector<array<int, 4>> Q[N];
    vector<ll> ans;
    void add_event(int x1, int y1, int x2, int y2, ll v)
    {
        E[x1].pb({y1, y2, v, -v * x1});
        E[x2].pb({y1, y2, -v, v * x2});
    }
    void add_query(int x1, int y1, int x2, int y2, int id)
    {
        Q[x1].pb({y1, y2, -1, id});
        Q[x2].pb({y1, y2, 1, id});
        ans.pb(0);
    }
    void solve(int n) {
        Seg root(0, n);
        for (int i = 0; i <= n; ++i) {
            for (auto j : E[i]) root.add(j[0], j[1], j[2], j[3]);
            for (auto j : Q[i]) ans[j[3]] += j[2] * root.query(j[0], j[1], i);
        }
    }
};
```

9.12 Matroid Intersection

Start from $S = \emptyset$. In each iteration, let

- $Y_1 = \{x \notin S \mid S \cup \{x\} \in I_1\}$
- $Y_2 = \{x \notin S \mid S \cup \{x\} \in I_2\}$

If there exists $x \in Y_1 \cap Y_2$, insert x into S . Otherwise for each $x \in S, y \notin S$, create edges

- $x \rightarrow y$ if $S - \{x\} \cup \{y\} \in I_1$.
- $y \rightarrow x$ if $S - \{x\} \cup \{y\} \in I_2$.

Find a *shortest* path (with BFS) starting from a vertex in Y_1 and ending at a vertex in Y_2 which doesn't pass through any other vertices in Y_2 , and alternate the path. The size of S will be incremented by 1 in each

iteration. For the weighted case, assign weight $w(x)$ to vertex x if $x \in S$ and $-w(x)$ if $x \notin S$. Find the path with the minimum number of edges among all minimum length paths and alternate it.

9.13 Python Misc

```
from [decimal, fractions, math, random] import *
setcontext(Context(prec=10, Emax=MAX_EMAX, rounding=
    ROUND_FLOOR))
Decimal('1.1') / Decimal('0.2')
Fraction(3, 7)
Fraction(Decimal('1.14'))
Fraction('1.2').limit_denominator(4).numerator
Fraction(cos(pi / 3)).limit_denominator()
print(*[randint(1, C) for i in range(0, N)], sep=' ')
```