

## Contents

<b>1 Basic</b>	<b>1</b>	6.8 Discrete Log . . . . .	14
1.1 Shell Script . . . . .	1	6.9 Factorial without Prime Factor . . . . .	14
1.2 Debug Macro . . . . .	1	6.10 Berlekamp Massey . . . . .	14
1.3 Pragma / FastIO . . . . .	1	6.11 Simplex . . . . .	15
1.4 Divide . . . . .	1	6.12 Euclidean . . . . .	15
<b>2 Data Structure</b>	<b>1</b>	6.13 Linear Programming Construction . . . . .	15
2.1 Leftist Tree . . . . .	1	6.14 Theorem . . . . .	15
2.2 Splay Tree . . . . .	1	6.15 Estimation . . . . .	16
2.3 Link Cut Tree . . . . .	2	6.16 General Purpose Numbers	16
2.4 Treap . . . . .	2	<b>7 Polynomial</b>	<b>16</b>
2.5 vEB Tree . . . . .	3	7.1 Number Theoretic Transform . . . . .	16
<b>3 Flow / Matching</b>	<b>3</b>	7.2 Fast Fourier Transform	16
3.1 Dinic . . . . .	3	7.3 Primes . . . . .	16
3.2 Min Cost Max Flow . . . . .	4	7.4 Polynomial Operations	16
3.3 Kuhn Munkres . . . . .	4	7.5 Fast Linear Recursion	18
3.4 Hopcroft Karp . . . . .	5	7.6 Fast Walsh Transform	18
3.5 SW Min Cut . . . . .	5	<b>8 Geometry</b>	<b>18</b>
3.6 Gomory Hu Tree . . . . .	5	8.1 Basic . . . . .	18
3.7 Blossom . . . . .	5	8.2 SVG Writer . . . . .	18
3.8 Min Cost Circulation	6	8.3 Heart . . . . .	19
3.9 Weighted Blossom . . . . .	6	8.4 External Bisector . . . . .	19
3.10 Flow Model . . . . .	7	8.5 Intersections . . . . .	19
<b>4 Graph</b>	<b>7</b>	8.6 Intersection Area of Polygon and Circle . . . . .	19
4.1 Heavy-Light Decomposition . . . . .	7	8.7 Tangents . . . . .	19
4.2 Centroid Decomposition	8	8.8 Point In Convex . . . . .	20
4.3 Edge BCC . . . . .	8	8.9 Point In Circle . . . . .	20
4.4 Vertex BCC / Round Square Tree . . . . .	8	8.10 Point Segment Distance	20
4.5 SCC . . . . .	8	8.11 Convex Hull . . . . .	20
4.6 2SAT . . . . .	9	8.12 Minimum Enclosing Circle	20
4.7 Virtual Tree . . . . .	9	8.13 Union of Circles . . . . .	20
4.8 Directed MST . . . . .	9	8.14 Union of Polygons . . . . .	20
4.9 Dominator Tree . . . . .	9	8.15 Rotating SweepLine . . . . .	21
4.10 Bipartite Edge Coloring	10	8.16 Half Plane Intersection	21
4.11 Edge Coloring . . . . .	10	8.17 Minkowski Sum . . . . .	21
4.12 Maximum Clique . . . . .	10	8.18 Vector In Polygon . . . . .	21
<b>5 String</b>	<b>11</b>	8.19 Delaunay Triangulation	21
5.1 Aho-Corasick Automaton	11	8.20 Triangulation Voronoi	22
5.2 KMP Algorithm . . . . .	11	8.21 3D Point . . . . .	22
5.3 Z Algorithm . . . . .	11	8.22 3D Convex Hull . . . . .	22
5.4 Manacher . . . . .	11	<b>9 Else</b>	<b>23</b>
5.5 Suffix Array . . . . .	11	9.1 Pbds . . . . .	23
5.6 SAIS . . . . .	11	9.2 Bit Hack . . . . .	23
5.7 Suffix Automaton . . . . .	12	9.3 Smawk Algorithm . . . . .	23
5.8 Minimum Rotation . . . . .	12	9.4 Slope Trick . . . . .	23
5.9 Palindrome Tree . . . . .	12	9.5 ALL LCS . . . . .	24
5.10 Lyndon Factorization	12	9.6 Hilbert Curve . . . . .	24
5.11 Main Lorentz . . . . .	13	9.7 Line Container . . . . .	24
<b>6 Math</b>	<b>13</b>	9.8 Min Plus Convolution	24
6.1 Miller Rabin / Pollard Rho . . . . .	13	9.9 Matroid Intersection	24
6.2 Ext GCD . . . . .	13	9.10 Simulated Annealing . . . . .	24
6.3 Chinese Remainder Theorem . . . . .	13	9.11 Bitset LCS . . . . .	24
6.4 PiCount . . . . .	13	9.12 Binary Search On Fraction . . . . .	24
6.5 Linear Function Mod Min	14	9.13 Cyclic Ternary Search	24
6.6 Floor Sum . . . . .	14	9.14 Tree Hash . . . . .	25
6.7 Quadratic Residue . . . . .	14	9.15 Python Misc . . . . .	25

## 1 Basic

### 1.1 Shell Script

```
cpp hash.cpp -dD -P -fpreprocessed | tr -d "[:space:]"
| md5sum | cut -c -6
```

### 1.2 Debug Macro [2e0e48]

```
#ifdef ABS
template <typename T>
ostream& operator << (ostream &o, vector <T> vec) {
    o << "{"; int f = 0;
    for (T i : vec) o << (f++ ? " " : "") << i;
    return o << "}";
}
void bug__(int c, auto ...a) {
    cerr << "\e[1;" << c << "m";
    (... , (cerr << a << " "));
    cerr << "\e[0m" << endl;
}
#define bug_(c, x...) bug__(c, __LINE__, "[" + string(#x) + "]", x)
#define bug(x...) bug_(32, x)
#define bugv(x...) bug_(36, vector(x))
#define safe_bug_(33, "safe")
#else
```

```
#define bug(x...) void(0)
#define bugv(x...) void(0)
#define safe void(0)
#endif
```

### 1.3 Pragma / FastIO

```
#pragma GCC optimize("Ofast,no-stack-protector")
#pragma GCC optimize("no-math-errno,unroll-loops")
#pragma GCC target("sse,sse2,sse3,ssse3,sse4")
#pragma GCC target("popcnt,abm,mmx,avx,arch=skylake")
__builtin_ia32_ldmxcsr(__builtin_ia32_stmxcsr()|0x8040)
#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;
}
```

### 1.4 Divide

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

## 2 Data Structure

### 2.1 Leftist Tree [414ab9]

```
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 [21142b]

```
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;
    int 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 [bca367]

```

// weighted subtree size, weighted path max
struct LCT {
    int ch[N][2], pa[N], v[N], sz[N];
    int 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;
    int 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 [9d5c2a]

```

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() {}
};
// delete default code sz
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 vEB Tree [087d11]

```

using u64=uint64_t;
constexpr int lsb(u64 x)
{ return x?__builtin_ctzll(x):1<<30; }
constexpr int msb(u64 x)
{ return x?63-__builtin_clzll(x):-1; }
template<int N, class T=void>
struct veb{
    static const int M=N>>1;
    veb<M> ch[1<<N-M];
    veb<N-M> aux;

```

```

    int mn,mx;
    veb():mn(1<<30),mx(-1){}
    constexpr int mask(int x){return x&((1<<M)-1);}
    bool empty(){return mx==-1;}
    int min(){return mn;}
    int max(){return mx;}
    bool have(int x){
        return x==mn?true:ch[x>>M].have(mask(x));
    }
    void insert_in(int x){
        if(empty()) return mn=mx=x,void();
        if(x<mn) swap(x,mn);
        if(x>mx) mx=x;
        if(ch[x>>M].empty()) aux.insert_in(x>>M);
        ch[x>>M].insert_in(mask(x));
    }
    void erase_in(int x){
        if(mn==mx) return mn=1<<30,mx=-1,void();
        if(x==mn) mn=x=(aux.min()<<M)^ch[aux.min()].min();
        ch[x>>M].erase_in(mask(x));
        if(ch[x>>M].empty()) aux.erase_in(x>>M);
        if(x==mx){
            if(aux.empty()) mx=mn;
            else mx=(aux.max()<<M)^ch[aux.max()].max();
        }
    }
    void insert(int x){
        if(!have(x)) insert_in(x);
    }
    void erase(int x){
        if(have(x)) erase_in(x);
    }
    int next(int x){// >=x
        if(x>mx) return 1<<30;
        if(x<=mn) return mn;
        if(mask(x)<=ch[x>>M].max())
            return ((x>>M)<<M)^ch[x>>M].next(mask(x));
        int y=aux.next((x>>M)+1);
        return (y<<M)^ch[y].min();
    }
    int prev(int x){// <x
        if(x<=mn) return -1;
        if(x>mx) return mx;
        if(x<=(aux.min()<<M)+ch[aux.min()].min())
            return mn;
        if(mask(x)>ch[x>>M].min())
            return ((x>>M)<<M)^ch[x>>M].prev(mask(x));
        int y=aux.prev(x>>M);
        return (y<<M)^ch[y].max();
    }
};
template<int N>
struct veb<N,typename enable_if<N<=6>::type>{
    u64 a;
    veb():a(0){}
    void insert_in(int x){a|=1ull<<x;}
    void insert(int x){a|=1ull<<x;}
    void erase_in(int x){a&=~(1ull<<x);}
    void erase(int x){a&=~(1ull<<x);}
    bool have(int x){return a>>x&1;}
    bool empty(){return a==0;}
    int min(){return lsb(a);}
    int max(){return msb(a);}
    int next(int x){return lsb(a&~((1ull<<x)-1));}
    int prev(int x){return msb(a&((1ull<<x)-1));}
};

```

## 3 Flow / Matching

### 3.1 Dinic [8898fb]

```

template <typename T>
struct Dinic { // 0-based
    const T INF = numeric_limits<T>::max() / 2;
    struct edge { int to, rev; T cap, flow; };
    int n, s, t;
    vector <vector <edge>> g;
    vector <int> dis, cur;
    T dfs(int u, T cap) {
        if (u == t || !cap) return cap;
        for (int &i = cur[u]; i < (int)g[u].size(); ++i) {
            edge &e = g[u][i];

```

```

    if (dis[e.to] == dis[u] + 1 && e.flow != e.cap) {
        T df = dfs(e.to, min(e.cap - e.flow, cap));
        if (df) {
            e.flow += df;
            g[e.to][e.rev].flow -= df;
            return df;
        }
    }
    dis[u] = -1;
    return 0;
}

bool bfs() {
    fill(all(dis), -1);
    queue<int> q;
    q.push(s), dis[s] = 0;
    while (!q.empty()) {
        int v = q.front(); q.pop();
        for (auto &u : g[v])
            if (!dis[u.to] && u.flow != u.cap) {
                q.push(u.to);
                dis[u.to] = dis[v] + 1;
            }
    }
    return dis[t] != -1;
}

T solve(int _s, int _t) {
    s = _s, t = _t;
    T flow = 0, df;
    while (bfs()) {
        fill(all(cur), 0);
        while ((df = dfs(s, INF))) flow += df;
    }
    return flow;
}

void reset() {
    for (int i = 0; i < n; ++i)
        for (auto &j : g[i]) j.flow = 0;
}

void add_edge(int u, int v, T cap) {
    g[u].pb(edge{v, (int)g[v].size(), cap, 0});
    g[v].pb(edge{u, (int)g[u].size() - 1, 0, 0});
}

Dinic (int _n) : n(_n), g(n), dis(n), cur(n) {}
};

```

### 3.2 Min Cost Max Flow [8083d7]

```

template <typename T1, typename T2>
struct MCMF { // T1 -> flow, T2 -> cost, 0-based
    const T1 INF1 = numeric_limits<T1>::max() / 2;
    const T2 INF2 = numeric_limits<T2>::max() / 2;
    struct edge {
        int v; T1 f; T2 c;
    };
    int n, s, t;
    vector<vector<int>> g;
    vector<edge> e;
    vector<T2> dis, pot;
    vector<int> rt, vis;
    // bool DAG()...
    bool SPFA() {
        fill(all(rt), -1), fill(all(dis), INF2);
        fill(all(vis), 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 : g[v]) {
                auto [u, f, c] = e[id];
                T2 ndis = dis[v] + c + pot[v] - pot[u];
                if (f > 0 && dis[u] > ndis) {
                    dis[u] = ndis, rt[u] = id;
                    if (!vis[u]) vis[u] = true, q.push(u);
                }
            }
        }
        return dis[t] != INF2;
    } // d9b0f8
    bool dijkstra() {
        fill(all(rt), -1), fill(all(dis), INF2);

```

```

        priority_queue<pair<T2, int>, vector<pair<T2, int>>, greater<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 : g[v]) {
                auto [u, f, c] = e[id];
                T2 ndis = dis[v] + c + pot[v] - pot[u];
                if (f > 0 && dis[u] > ndis) {
                    dis[u] = ndis, rt[u] = id;
                    pq.emplace(ndis, u);
                }
            }
        }
        return dis[t] != INF2;
    }
}

vector<pair<T1, T2>> solve(int _s, int _t) {
    s = _s, t = _t, fill(all(pot), 0);
    vector<pair<T1, T2>> ans; 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;
        ans.emplace_back(add, dis[t]), fr = false;
        for (int i = 0; i < n; ++i) swap(dis[i], pot[i]);
    }
    return ans;
}

void reset() {
    for (int i = 0; i < (int)e.size(); ++i) e[i].f = 0;
}

void add_edge(int u, int v, T1 f, T2 c) {
    g[u].pb((int)e.size(), e.pb({v, f, c}));
    g[v].pb((int)e.size(), e.pb({u, 0, -c}));
}

MCMF (int _n) : n(_n), g(n), e(), dis(n), pot(n),
    rt(n), vis(n) {} // 05becb
};

```

### 3.3 Kuhn Munkres [b880ad]

```

template <typename T>
struct KM { // 0-based, remember to init
    const T INF = numeric_limits<T>::max() / 2;
    int n; vector<vector<T>> w;
    vector<T> hl, hr, slk;
    vector<int> fl, fr, vl, vr, pre;
    queue<int> q;
    bool check(int x) {
        if (vl[x] = 1, ~fl[x])
            return q.push(fl[x]), vr[fl[x]] = 1;
        while (~x) swap(x, fr[fl[x] = pre[x]]);
        return 0;
    }
    void bfs(int s) {
        fill(all(slk), INF), fill(all(vl), 0);
        fill(all(vr), 0);
        while (!q.empty()) q.pop();
        q.push(s), vr[s] = 1;
        while (true) {
            T d;
            while (!q.empty()) {
                int y = q.front(); q.pop();
                for (int x = 0; x < n; ++x) {
                    d = hl[x] + hr[y] - w[x][y];
                    if (!vl[x] && slk[x] >= d) {
                        if (pre[x] = y, d) slk[x] = d;
                        else if (!check(x)) return;
                    }
                }
            }
            d = INF;
            for (int x = 0; x < n; ++x)
                if (!vl[x] && d > slk[x]) d = slk[x];
            for (int x = 0; x < n; ++x) {
                if (vl[x]) hl[x] += d;
                else slk[x] -= d;
            }

```

```

    if (vr[x]) hr[x] -= d;
}
for (int x = 0; x < n; ++x)
    if (!vl[x] && !slk[x] && !check(x)) return;
}
T solve() {
    fill(all(fl), -1), fill(all(fr), -1);
    fill(all(hr), 0);
    for (int i = 0; i < n; ++i)
        hl[i] = *max_element(all(w[i]));
    for (int i = 0; i < n; ++i) bfs(i);
    T res = 0;
    for (int i = 0; i < n; ++i) res += w[i][fl[i]];
    return res;
}
void add_edge(int a, int b, T wei) { w[a][b] = wei; }
KM (int _n) : n(_n), w(n, vector<T>(n, -INF)), hl(n),
    hr(n), slk(n), fl(n), fr(n), vl(n), vr(n), pre(n){
};

```

### 3.4 Hopcroft Karp [33c68d]

```

struct HopcroftKarp { // 0-based
    const int INF = 1 << 30;
    int n, m;
    vector<vector<int>> g;
    vector<int> match, dis, matched, vis;
    bool dfs(int x) {
        vis[x] = true;
        for (int y : g[x])
            if (match[y] == -1 || (dis[match[y]] == dis[x] +
                1 && !vis[match[y]] && dfs(match[y]))) {
                match[y] = x, matched[x] = true;
                return true;
            }
        return false;
    }
    bool bfs() {
        fill(all(dis), -1);
        queue<int> q;
        for (int x = 0; x < n; ++x) if (!matched[x])
            dis[x] = 0, q.push(x);
        int mx = INF;
        while (!q.empty()) {
            int x = q.front(); q.pop();
            for (int y : g[x]) {
                if (match[y] == -1) {
                    mx = dis[x];
                    break;
                } else if (dis[match[y]] == -1)
                    dis[match[y]] = dis[x] + 1, q.push(match[y]);
            }
        }
        return mx < INF;
    }
    int solve() {
        int res = 0;
        fill(all(match), -1);
        fill(all(matched), 0);
        while (bfs()) {
            fill(all(vis), 0);
            for (int x = 0; x < n; ++x) if (!matched[x])
                res += dfs(x);
        }
        return res;
    }
    void add_edge(int x, int y) { g[x].pb(y); }
    HopcroftKarp (int _n, int _m) : n(_n), m(_m), g(n),
        match(m), dis(n), matched(n), vis(n) {}
};

```

### 3.5 SW Min Cut [b9af94]

```

template<typename T>
struct SW { // 0-based
    const T INF = numeric_limits<T>::max() / 2;
    vector<vector<T>> g;
    vector<T> sum;
    vector<bool> vis, dead;
    int n;
    T solve() {

```

```

        T ans = INF;
        for (int r = 0; r + 1 < n; ++r) {
            fill(all(vis), 0), fill(all(sum), 0);
            int num = 0, s = -1, t = -1;
            while (num < n - r) {
                int now = -1;
                for (int i = 0; i < n; ++i)
                    if (!vis[i] && !dead[i] &&
                        (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;
    }
    void add_edge(int u, int v, T w) {
        g[u][v] += w, g[v][u] += w; }
    SW (int _n) : n(_n), g(n, vector<T>(n)), vis(n),
        sum(n), dead(n) {}
};

```

### 3.6 Gomory Hu Tree [90ead2]

```

vector<array<int, 3>> GomoryHu(Dinic<int> flow) {
    // Tree edge min = mcut (0-based)
    int n = flow.n;
    vector<array<int, 3>> ans;
    vector<int> rt(n);
    for (int i = 1; i < n; ++i) {
        int t = rt[i];
        flow.reset();
        ans.pb({i, t, flow.solve(i, t)});
        flow.bfs();
        for (int j = i + 1; j < n; ++j)
            if (rt[j] == t && flow.dis[j] != -1) rt[j] = i;
    }
    return ans;
}

```

### 3.7 Blossom [6092d8]

```

struct Matching { // 0-based
    int n, tk;
    vector<vector<int>> g;
    vector<int> fa, pre, match, s, t;
    queue<int> q;
    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 (t[x] == tk) return x;
                t[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) {
        iota(all(fa), 0), fill(all(s), -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, 1);
            blossom(u, x, 1);
        }
    }
}
return false;
}
int solve() {
    int res = 0;
    for (int x = 0; x < n; ++x) {
        if (match[x] == n) res += bfs(x);
    }
    return res;
}
void add_edge(int u, int v) {
    g[u].push_back(v), g[v].push_back(u);
}
Matching (int _n) : n(_n), tk(0), g(n), fa(n + 1),
    pre(n + 1, n), match(n + 1, n), s(n + 1, t(n) {}
};

```

### 3.8 Min Cost Circulation [bd1e15]

```

struct MinCostCirculation { // 0-base
    struct Edge {
        ll from, to, cap, fcap, flow, cost, rev;
    } *past[N];
    vector<Edge> G[N];
    ll dis[N], inq[N], n;
    void BellmanFord(int s) {
        fill_n(dis, n, INF), fill_n(inq, n, 0);
        queue<int> q;
        auto relax = [&](int u, ll d, Edge *e) {
            if (dis[u] > d) {
                dis[u] = d, past[u] = e;
                if (!inq[u]) inq[u] = 1, q.push(u);
            }
        };
        relax(s, 0, 0);
        while (!q.empty()) {
            int u = q.front();
            q.pop(), inq[u] = 0;
            for (auto &e : G[u])
                if (e.cap > e.flow)
                    relax(e.to, dis[u] + e.cost, &e);
        }
    }
    void try_edge(Edge &cur) {
        if (cur.cap > cur.flow) return ++cur.cap, void();
        BellmanFord(cur.to);
        if (dis[cur.from] + cur.cost < 0) {
            ++cur.flow, --G[cur.to][cur.rev].flow;
            for (int i = cur.from; past[i]; i = past[i] -> from) {
                auto &e = *past[i];
                ++e.flow, --G[e.to][e.rev].flow;
            }
        }
        ++cur.cap;
    }
    void solve(int mxlg) {
        for (int b = mxlg; b >= 0; --b) {
            for (int i = 0; i < n; ++i)
                for (auto &e : G[i])
                    e.cap *= 2, e.flow *= 2;
            for (int i = 0; i < n; ++i)
                for (auto &e : G[i])
                    if (e.fcap >> b & 1)

```

```

                try_edge(e);
            }
        }
    }
    void init(int _n) { n = _n;
        for (int i = 0; i < n; ++i) G[i].clear();
    }
    void add_edge(ll a, ll b, ll cap, ll cost) {
        G[a].pb(Edge{a, b, 0, cap, 0, cost, sz(G[b]) + (a
            == b)});
        G[b].pb(Edge{b, a, 0, 0, 0, -cost, sz(G[a]) - 1});
    }
} mcmf; // O(VE * ElogC)

```

### 3.9 Weighted Blossom [dc42e4]

```

#define pb emplace_back
#define REP(i, l, r) for (int i=(l); i<=(r); ++i)
struct WeightGraph { // 1-based
    static const int inf = INT_MAX;
    struct edge { int u, v, w; }; int n, nx;
    vector<int> lab; vector<vector<edge>> g;
    vector<int> slack, match, st, pa, S, vis;
    vector<vector<int>> flo, flo_from; queue<int> q;
    WeightGraph(int n_) : n(n_), nx(n * 2), lab(nx + 1),
        g(nx + 1, vector<edge>(nx + 1)), slack(nx + 1),
        flo(nx + 1), flo_from(nx + 1, vector(n + 1, 0)) {
        match = st = pa = S = vis = slack;
        REP(u, 1, n) REP(v, 1, n) g[u][v] = {u, v, 0};
    }
    int ED(edge e) {
        return lab[e.u] + lab[e.v] - g[e.u][e.v].w * 2;
    }
    void update_slack(int u, int x, int &s) {
        if (!s || ED(g[u][x]) < ED(g[s][x])) s = u;
    }
    void set_slack(int x) {
        slack[x] = 0;
        REP(u, 1, n)
            if (g[u][x].w > 0 && st[u] != x && S[st[u]] == 0)
                update_slack(u, x, slack[x]);
    }
    void q_push(int x) {
        if (x <= n) q.push(x);
        else for (int y : flo[x]) q_push(y);
    }
    void set_st(int x, int b) {
        st[x] = b;
        if (x > n) for (int y : flo[x]) set_st(y, b);
    }
    vector<int> split_flo(auto &f, int xr) {
        auto it = find(all(f), xr);
        if (auto pr = it - f.begin(); pr % 2 == 1)
            reverse(1 + all(f), it = f.end() - pr);
        auto res = vector(f.begin(), it);
        return f.erase(f.begin(), it), res;
    } // 7bb859
    void set_match(int u, int v) {
        match[u] = g[u][v].v;
        if (u <= n) return;
        int xr = flo_from[u][g[u][v].u];
        auto &f = flo[u], z = split_flo(f, xr);
        REP(i, 0, int(z.size()) - 1) set_match(z[i], z[i ^ 1]);
        set_match(xr, v); f.insert(f.end(), all(z));
    }
    void augment(int u, int v) {
        for (;;) {
            int xnv = st[match[u]]; set_match(u, v);
            if (!xnv) return;
            set_match(v = xnv, u = st[pa[xnv]]);
        }
    }
    int lca(int u, int v) {
        static int t = 0; ++t;
        for (++t; u || v; swap(u, v)) if (u) {
            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 o, int v) {
        int b = int(find(n + 1 + all(st), 0) - begin(st));
        lab[b] = 0, S[b] = 0; match[b] = match[o];
    }

```



```

vector<int> f = {0};
for (int x : {u, v}) {
    for (int y; x != 0; x = st[pa[y]])
        f.pb(x), f.pb(y = st[match[x]]), q_push(y);
    reverse(1 + all(f));
}
flo[b] = f; set_st(b, b);
REP(x, 1, nx) g[b][x].w = g[x][b].w = 0;
REP(x, 1, n) flo_from[b][x] = 0;
for (int xs : flo[b]) {
    REP(x, 1, nx)
        if (g[b][x].w == 0 || ED(g[xs][x]) < ED(g[b][x]))
            g[b][x] = g[xs][x], g[x][b] = g[x][xs];
    REP(x, 1, n)
        if (flo_from[xs][x]) flo_from[b][x] = xs;
}
set_slack(b);
void expand_blossom(int b) {
    for (int x : flo[b]) set_st(x, x);
    int xr = flo_from[b][g[b][pa[b]].u], xs = -1;
    for (int x : split_flo(flo[b], xr)) {
        if (xs == -1) { xs = x; continue; }
        pa[xs] = g[x][xs].u; S[xs] = 1, S[x] = 0;
        slack[xs] = 0; set_slack(x); q_push(x); xs = -1;
    }
    for (int x : flo[b])
        if (x == xr) S[x] = 1, pa[x] = pa[b];
        else S[x] = -1, set_slack(x);
    st[b] = 0;
}
bool on_found_edge(const edge &e) {
    if (int u = st[e.u], v = st[e.v]; S[v] == -1) {
        int nu = st[match[v]]; pa[v] = e.u; S[v] = 1;
        slack[v] = slack[nu] = 0; S[nu] = 0; q_push(nu);
    } else if (S[v] == 0) {
        if (int o = lca(u, v)) add_blossom(u, o, v);
        else return augment(u, v), augment(v, u), true;
    }
    return false;
} // 82ea63
bool matching() {
    fill(all(S), -1), fill(all(slack), 0);
    q = queue<int>();
    REP(x, 1, nx) 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;
            REP(v, 1, n)
                if (g[u][v].w > 0 && st[u] != st[v]) {
                    if (ED(g[u][v]) != 0)
                        update_slack(u, st[v], slack[st[v]]);
                    else if (on_found_edge(g[u][v])) return true;
                }
        }
        int d = inf;
        REP(b, n + 1, nx) if (st[b] == b && S[b] == 1)
            d = min(d, lab[b] / 2);
        REP(x, 1, nx)
            if (int s = slack[x]; st[x] == x && s && S[x] <= 0)
                d = min(d, ED(g[s][x]) / (S[x] + 2));
        REP(u, 1, n)
            if (S[st[u]] == 1) lab[u] += d;
            else if (S[st[u]] == 0) {
                if (lab[u] <= d) return false;
                lab[u] -= d;
            }
        REP(b, n + 1, nx) if (st[b] == b && S[b] >= 0)
            lab[b] += d * (2 - 4 * S[b]);
        REP(x, 1, nx)
            if (int s = slack[x]; st[x] == x &&
                s && st[s] != x && ED(g[s][x]) == 0)
                if (on_found_edge(g[s][x])) return true;
        REP(b, n + 1, nx)
            if (st[b] == b && S[b] == 1 && lab[b] == 0)
                expand_blossom(b);
    }
}

```

```

}
return false;
}
pair<ll, int> solve() {
    fill(all(match), 0);
    REP(u, 0, n) st[u] = u, flo[u].clear();
    int w_max = 0;
    REP(u, 1, n) REP(v, 1, n) {
        flo_from[u][v] = (u == v ? u : 0);
        w_max = max(w_max, g[u][v].w);
    }
    REP(u, 1, n) lab[u] = w_max;
    int n_matches = 0; ll tot_weight = 0;
    while (matching()) ++n_matches;
    REP(u, 1, n) if (match[u] && match[u] < u)
        tot_weight += g[u][match[u]].w;
    return make_pair(tot_weight, n_matches);
}
void set_edge(int u, int v, int w) {
    g[u][v].w = g[v][u].w = w; } // c78909
};

```

### 3.10 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  $\infty$  (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  $\infty$  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.
- Minimum cost cyclic flow
  - Construct super source  $S$  and sink  $T$
  - For each edge  $(x, y, c)$ , connect  $x \rightarrow y$  with  $(cost, cap) = (c, 1)$  if  $c > 0$ , otherwise connect  $y \rightarrow x$  with  $(cost, cap) = (-c, 1)$
  - For each edge with  $c < 0$ , sum these cost as  $K$ , then increase  $d(y)$  by 1, decrease  $d(x)$  by 1
  - For each vertex  $v$  with  $d(v) > 0$ , connect  $S \rightarrow v$  with  $(cost, cap) = (0, d(v))$
  - For each vertex  $v$  with  $d(v) < 0$ , connect  $v \rightarrow T$  with  $(cost, cap) = (0, -d(v))$
  - Flow from  $S$  to  $T$ , the answer is the cost of the flow  $C + K$
- 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
  - Change the weight of each edge to  $\mu(u) + \mu(v) - w(u, v)$ , where  $\mu(v)$  is the cost of the cheapest edge incident to  $v$ .
  - Let the maximum weight matching of the graph be  $x$ , the answer will be  $\sum \mu(v) - x$ .

## 4 Graph

### 4.1 Heavy-Light Decomposition [9ec77f]

```

struct HLD { // 0-based, remember to build
    int n, _id;
    vector<vector<int>> g;
    vector<int> dep, pa, tsz, ch, hd, id;
    void dfs(int v, int p) {
        dep[v] = ~p ? dep[p] + 1 : 0;
        pa[v] = p, tsz[v] = 1, ch[v] = -1;
        for (int u : g[v]) if (u != p) {

```

```

    dfs(u, v);
    if (ch[v] == -1 || tsz[ch[v]] < tsz[u])
        ch[v] = u;
    tsz[v] += tsz[u];
}
}
void hld(int v, int p, int h) {
    hd[v] = h, id[v] = _id++;
    if (~ch[v]) hld(ch[v], v, h);
    for (int u : g[v]) if (u != p && u != ch[v])
        hld(u, v, u);
}
vector<pii> query(int u, int v) {
    vector<pii> ans;
    while (hd[u] != hd[v]) {
        if (dep[hd[u]] > dep[hd[v]]) swap(u, v);
        ans.emplace_back(id[hd[v]], id[v] + 1);
        v = pa[hd[v]];
    }
    if (dep[u] > dep[v]) swap(u, v);
    ans.emplace_back(id[u], id[v] + 1);
    return ans;
}
void build() {
    for (int i = 0; i < n; ++i) if (id[i] == -1)
        dfs(i, -1), hld(i, -1, i);
}
void add_edge(int u, int v) {
    g[u].pb(v), g[v].pb(u);
}
HLD (int _n) : n(_n), _id(0), g(n), dep(n), pa(n),
    tsz(n), ch(n), hd(n), id(n, -1) {}
};

```

## 4.2 Centroid Decomposition [28b80a]

```

struct CD { // 0-based, remember to build
    int n, lg; // pa, dep are centroid tree attributes
    vector<vector<int>> g, dis;
    vector<int> pa, tsz, dep, vis;
    void dfs1(int v, int p) {
        tsz[v] = 1;
        for (int u : g[v]) if (u != p && !vis[u])
            dfs1(u, v), tsz[v] += tsz[u];
    }
    int dfs2(int v, int p, int _n) {
        for (int u : g[v])
            if (u != p && !vis[u] && tsz[u] > _n / 2)
                return dfs2(u, v, _n);
        return v;
    }
    void dfs3(int v, int p, int d) {
        dis[v][d] = ~p ? dis[p][d] + 1 : 0;
        for (int u : g[v]) if (u != p && !vis[u])
            dfs3(u, v, d);
    }
    void cd(int v, int p, int d) {
        dfs1(v, -1), v = dfs2(v, -1, tsz[v]);
        vis[v] = true, pa[v] = p, dep[v] = d;
        dfs3(v, -1, d);
        for (int u : g[v]) if (!vis[u])
            cd(u, v, d + 1);
    }
    void build() { cd(0, -1, 0); }
    void add_edge(int u, int v) {
        g[u].pb(v), g[v].pb(u);
    }
    CD (int _n) : n(_n), lg(_lg(n) + 1), g(n),
        dis(n, vector<int>(lg)), pa(n), tsz(n),
        dep(n), vis(n) {}
};

```

## 4.3 Edge BCC [cf5e55]

```

struct EBCC { // 0-based, remember to build
    int n, m, nbcc;
    vector<vector<pii>> g;
    vector<int> pa, low, dep, bcc_id, stk, is_bridge;
    void dfs(int v, int p, int f) {
        low[v] = dep[v] = ~p ? dep[p] + 1 : 0;
        stk.pb(v), pa[v] = p;
        for (auto [u, e] : g[v]) {
            if (low[u] == -1)
                dfs(u, v, e), low[v] = min(low[v], low[u]);

```

```

            else if (e != f)
                low[v] = min(low[v], dep[u]);
        }
        if (low[v] == dep[v]) {
            if (~f) is_bridge[f] = true;
            int id = nbcc++, x;
            do {
                x = stk.back(), stk.pop_back();
                bcc_id[x] = id;
            } while (x != v);
        }
    }
    void build() {
        is_bridge.assign(m, 0);
        for (int i = 0; i < n; ++i) if (low[i] == -1)
            dfs(i, -1, -1);
    }
    void add_edge(int u, int v) {
        g[u].emplace_back(v, m), g[v].emplace_back(u, m++);
    }
    EBCC (int _n) : n(_n), m(0), nbcc(0), g(n), pa(n),
        low(n, -1), dep(n), bcc_id(n), stk() {}
};

```

## 4.4 Vertex BCC / Round Square Tree [3818e9]

```

struct BCC { // 0-based, remember to build
    int n, nbcc; // note for isolated point
    vector<vector<int>> g, _g; // id >= n: bcc
    vector<int> pa, dep, low, stk, pa2, dep2;
    void dfs(int v, int p) {
        dep[v] = low[v] = ~p ? dep[p] + 1 : 0;
        stk.pb(v), pa[v] = p;
        for (int u : g[v]) if (u != p) {
            if (low[u] == -1) {
                dfs(u, v), low[v] = min(low[v], low[u]);
                if (low[u] >= dep[v]) {
                    int id = nbcc++, x;
                    do {
                        x = stk.back(), stk.pop_back();
                        _g[id + n].pb(x), _g[x].pb(id + n);
                    } while (x != u);
                    _g[id + n].pb(v), _g[v].pb(id + n);
                } else low[v] = min(low[v], dep[u]);
            }
        }
        bool is_cut(int x) { return (_g[x].size() != 1); }
        vector<int> bcc(int id) { return _g[id + n]; }
        int bcc_id(int u, int v) {
            return pa2[dep2[u] < dep2[v] ? v : u] - n;
        }
        void dfs2(int v, int p) {
            dep2[v] = ~p ? dep2[p] + 1 : 0, pa2[v] = p;
            for (int u : _g[v]) if (u != p) dfs2(u, v);
        }
        void build() {
            low.assign(n, -1);
            for (int i = 0; i < n; ++i) if (low[i] == -1)
                dfs(i, -1), dfs2(i, -1);
        }
        void add_edge(int u, int v) {
            g[u].pb(v), g[v].pb(u);
        }
        BCC (int _n) : n(_n), nbcc(0), g(n), _g(2 * n),
            pa(n), dep(n), low(n), stk(), pa2(n * 2),
            dep2(n * 2) {}
};

```

## 4.5 SCC [9bee8c]

```

struct SCC {
    int n, nscc, _id;
    vector<vector<int>> g;
    vector<int> dep, low, scc_id, stk;
    void dfs(int v) {
        dep[v] = low[v] = _id++, stk.pb(v);
        for (int u : g[v]) if (scc_id[u] == -1) {
            if (low[u] == -1) dfs(u);
            low[v] = min(low[v], low[u]);
        }
        if (low[v] == dep[v]) {
            int id = nscc++, x;
            do {

```



```

    x = stk.back(), stk.pop_back(), scc_id[x] = id;
    } while (x != v);
}
}
void build() {
    for (int i = 0; i < n; ++i) if (low[i] == -1)
        dfs(i);
}
void add_edge(int u, int v) { g[u].pb(v); }
SCC (int _n) : n(_n), nscc(0), _id(0), g(n), dep(n),
    low(n, -1), scc_id(n, -1), stk() {}
};

```

#### 4.6 2SAT [938072]

```

struct SAT { // 0-based, need SCC
    int n; vector<pii> edge; vector<int> is;
    int rev(int x) { return x < n ? x + n : x - n; }
    void add_ifthen(int x, int y) {
        add_clause(rev(x), y); }
    void add_clause(int x, int y) {
        edge.emplace_back(rev(x), y);
        edge.emplace_back(rev(y), x); }
    bool solve() {
        // is[i] = true -> i, is[i] = false -> -i
        SCC scc(2 * n);
        for (auto [u, v] : edge) scc.add_edge(u, v);
        scc.build();
        for (int i = 0; i < n; ++i) {
            if (scc.scc_id[i] == scc.scc_id[i + n])
                return false;
            is[i] = scc.scc_id[i] < scc.scc_id[i + n];
        }
        return true;
    }
    SAT (int _n) : n(_n), edge(), is(n) {}
};

```

#### 4.7 Virtual Tree [9e4a93]

```

// need lca, in, out
vector<pii> virtual_tree(vector<int> &v) {
    auto cmp = [&](int x, int y) { return in[x] < in[y]; };
    sort(all(v), cmp);
    int sz = (int)v.size();
    for (int i = 0; i + 1 < sz; ++i)
        v.pb(lca(v[i], v[i + 1]));
    sort(all(v), cmp);
    v.resize(unique(all(v)) - v.begin());
    vector<int> stk(1, v[0]);
    vector<pii> res;
    for (int i = 1; i < (int)v.size(); ++i) {
        int x = v[i];
        while (out[stk.back()] < out[x]) stk.pop_back();
        res.emplace_back(stk.back(), x), stk.pb(x);
    }
    return res;
}

```

#### 4.8 Directed MST [d6cf86]

```

using D = int;
struct edge { int u, v; D w; };
// 0-based, return index of edges
vector<int> dmst(vector<edge> &e, int n, int root) {
    using T = pair<D, int>;
    using PQ = pair<priority_queue<T, vector<T>,
        greater<T>>, D>;
    auto push = [](PQ &pq, T v) {
        pq.first.emplace(v.first - pq.second, v.second);
    };
    auto top = [](const PQ &pq) -> T {
        auto r = pq.first.top();
        return {r.first + pq.second, r.second};
    };
    auto join = [&push, &top](PQ &a, PQ &b) {
        if (a.first.size() < b.first.size()) swap(a, b);
        while (!b.first.empty())
            push(a, top(b)), b.first.pop();
    };
    vector<PQ> h(n * 2);
    for (int i = 0; i < e.size(); ++i)
        push(h[e[i].v], {e[i].w, i});
}

```

```

vector<int> a(n * 2), v(n * 2, -1), pa(n * 2, -1), r(
    n * 2);
iota(all(a), 0);
auto o = [&](int x) { int y;
    for (y = x; a[y] != y; y = a[y]);
    for (int ox = x; x != y; ox = x)
        x = a[x], a[ox] = y;
    return y;
};
v[root] = n + 1;
int pc = n;
for (int i = 0; i < n; ++i) if (v[i] == -1) {
    for (int p = i; v[p] == -1 || v[p] == i; p = o(e[r[
        p]].u)) {
        if (v[p] == i) {
            int q = p; p = pc++;
            do {
                h[q].second = -h[q].first.top().first;
                join(h[pa[q]] = a[q] = p, h[q]);
            } while ((q = o(e[r[q]].u)) != p);
        }
        v[p] = i;
        while (!h[p].first.empty() && o(e[top(h[p]).
            second].u) == p)
            h[p].first.pop();
        r[p] = top(h[p]).second;
    }
}
vector<int> ans;
for (int i = pc - 1; i >= 0; i--)
    if (i != root && v[i] != n) {
        for (int f = e[r[i]].v; f != -1 && v[f] != n; f =
            pa[f]) v[f] = n;
        ans.pb(r[i]);
    }
return ans;
}

```

#### 4.9 Dominator Tree [9fc069]

```

struct DominatorTree {
    int n, id;
    vector<vector<int>> g, rg, bucket;
    vector<int> sdom, dom, vis, rev, pa, rt, mn, res;
    // dom[s] = s, dom[v] = -1 if s -> v not exists
    int query(int v, int x) {
        if (rt[v] == v) return x ? -1 : v;
        int p = query(rt[v], 1);
        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 : g[v]) {
            if (vis[u] == -1) dfs(u), pa[vis[u]] = vis[v];
            rg[vis[u]].pb(vis[v]);
        }
    }
    void build(int s) {
        dfs(s);
        for (int i = id - 1; ~i; --i) {
            for (int u : rg[i]) {
                sdom[i] = min(sdom[i], sdom[query(u, 0)]);
            }
            if (i) bucket[sdom[i]].pb(i);
            for (int u : bucket[i]) {
                int p = query(u, 0);
                dom[u] = sdom[p] == i ? i : p;
            }
            if (i) rt[i] = pa[i];
        }
        fill(all(res), -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;
    }
}

```

```

    for (int i = 0; i < n; ++i) dom[i] = res[i];
}
void add_edge(int u, int v) { g[u].pb(v); }
DominatorTree (int _n) : n(_n), id(0), g(n), rg(n),
    bucket(n), sdom(n), dom(n, -1), vis(n, -1),
    rev(n), pa(n), rt(n), mn(n), res(n) {}
};

```

#### 4.10 Bipartite Edge Coloring [a22d96]

```

struct BipartiteEdgeColoring { // 1-based
    // returns edge coloring in adjacent matrix G
    int n, m;
    vector<vector<int>> col, G;
    int find_col(int x) {
        int c = 1;
        while (col[x][c]) c++;
        return c;
    }
    void dfs(int v, int c1, int c2) {
        if (!col[v][c1]) return col[v][c2] = 0, void(0);
        int u = col[v][c1];
        dfs(u, c2, c1);
        col[v][c1] = 0, col[v][c2] = u, col[u][c2] = v;
    }
    void solve() {
        for (int i = 1; i <= n + m; ++i)
            for (int j = 1; j <= max(n, m); ++j)
                if (col[i][j])
                    G[i][col[i][j]] = G[col[i][j]][i] = j;
    } // u = left index, v = right index
    void add_edge(int u, int v) {
        int c1 = find_col(u), c2 = find_col(v + n);
        dfs(u, c2, c1);
        col[u][c2] = v + n, col[v + n][c2] = u;
    }
    BipartiteEdgeColoring (int _n, int _m) : n(_n),
        m(_m), col(n + m + 1, vector<int>(max(n, m) + 1)),
        G(n + m + 1, vector<int>(n + m + 1)) {}
};

```

#### 4.11 Edge Coloring [60e200]

```

struct Vizing { // 1-based
    // returns edge coloring in adjacent matrix G
    int n;
    vector<vector<int>> C, G;
    vector<int> X, vst;
    vector<pii> E;
    void solve() {
        auto update = [&](int u)
            for (X[u] = 1; C[u][X[u]]; ++X[u]); };
        auto color = [&](int u, int v, int c) {
            int p = G[u][v];
            G[u][v] = G[v][u] = c;
            C[u][c] = v, C[v][c] = u;
            C[u][p] = C[v][p] = 0;
            if (p) X[u] = X[v] = p;
            else update(u), update(v);
            return p;
        };
        auto flip = [&](int u, int c1, int c2) {
            int p = C[u][c1];
            swap(C[u][c1], C[u][c2]);
            if (p) G[u][p] = G[p][u] = c2;
            if (!C[u][c1]) X[u] = c1;
            if (!C[u][c2]) X[u] = c2;
            return p;
        };
        fill(1 + all(X), 1);
        for (int t = 0; t < (int)E.size(); ++t) {
            auto [u, v0] = E[t];
            int v = v0, c0 = X[u], c = c0, d;
            vector<pii> L;
            fill(1 + all(vst), 0);
            while (!G[u][v0]) {
                L.emplace_back(v, d = X[v]);
                if (!C[v][c]) {
                    for (int a = sz(L) - 1; a >= 0; --a)
                        c = color(u, L[a].first, c);
                } else if (!C[u][d]) {
                    for (int a = sz(L) - 1; a >= 0; --a)

```

```

                        color(u, L[a].first, L[a].second);
                } else if (vst[d]) break;
                else vst[d] = 1, v = C[u][d];
            }
            if (!G[u][v0]) {
                for (; v; v = flip(v, c, d), swap(c, d));
                if (int a; C[u][c0]) {
                    for (a = sz(L) - 2;
                        a >= 0 && L[a].second != c; --a);
                    for (; a >= 0; --a)
                        color(u, L[a].first, L[a].second);
                }
                else --t;
            }
        }
        void add_edge(int u, int v) { E.emplace_back(u, v); }
        Vizing(int _n) : n(_n), C(n + 1, vector<int>(n + 1)),
            G(n + 1, vector<int>(n + 1)), X(n + 1), vst(n + 1) {}
};

```

#### 4.12 Maximum Clique [f99a13]

```

struct MaxClique { // Maximum Clique
    bitset<N> a[N], cs[N];
    int ans, sol[N], q, cur[N], d[N], n;
    void init(int _n) {
        n = _n;
        for (int i = 0; i < n; i++) a[i].reset();
    }
    void add_edge(int u, int v) { a[u][v] = a[v][u] = 1; }
    void csort(vector<int> &r, vector<int> &c) {
        int mx = 1, km = max(ans - q + 1, 1), t = 0;
        int m = r.size();
        cs[1].reset(), cs[2].reset();
        for (int i = 0; i < m; i++) {
            int p = r[i], k = 1;
            while ((cs[k] & a[p]).count()) k++;
            if (k > mx) mx++, cs[mx + 1].reset();
            cs[k][p] = 1;
            if (k < km) r[t++] = p;
        }
        c.resize(m);
        if (t) c[t - 1] = 0;
        for (int k = km; k <= mx; k++)
            for (int p = cs[k]._Find_first(); p < N;
                p = cs[k]._Find_next(p))
                r[t] = p, c[t] = k, t++;
    }
    void dfs(vector<int> &r, vector<int> &c, int l,
        bitset<N> mask) {
        while (!r.empty()) {
            int p = r.back();
            r.pop_back(), mask[p] = 0;
            if (q + c.back() <= ans) return;
            cur[q++] = p;
            vector<int> nr, nc;
            bitset<N> nmask = mask & a[p];
            for (int i : r)
                if (a[p][i]) nr.push_back(i);
            if (!nr.empty()) {
                if (1 < 4) {
                    for (int i : nr)
                        d[i] = (a[i] & nmask).count();
                    sort(nr.begin(), nr.end(),
                        [&](int x, int y) { return d[x] > d[y]; });
                }
                csort(nr, nc), dfs(nr, nc, l + 1, nmask);
            } else if (q > ans) ans = q, copy_n(cur, q, sol);
            c.pop_back(), q--;
        }
    }
    int solve(bitset<N> mask = bitset<N>(),
        string(N, '1')) { // vertex mask
        vector<int> r, c;
        ans = q = 0;
        for (int i = 0; i < n; i++)
            if (mask[i]) r.push_back(i);
        for (int i = 0; i < n; i++)
            d[i] = (a[i] & mask).count();
        sort(r.begin(), r.end(),

```

```

    [&](int i, int j) { return d[i] > d[j]; });
    csort(r, c), dfs(r, c, 1, mask);
    return ans; // sol[0 ~ ans-1]
}
};

```

## 5 String

### 5.1 Aho-Corasick Automaton [d208c9]

```

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]);
            to[0][i] = 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] ? ch[v][j] : 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'];
    //         ans += cnt[now];
    //     }
    //     return ans;
    // }
};

```

### 5.2 KMP Algorithm [f379fc]

```

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

```

### 5.3 Z Algorithm [7d5c7c]

```

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

```

### 5.4 Manacher [c18d8b]

```

// return value only consider string tmp, not s
vector<int> manacher(string tmp) {
    string s = "&";
    for (char c : tmp) s.pb(c), s.pb('%');
    int l = 0, r = 0, n = (int)s.size();
    vector<int> Z(n);
    for (int i = 0; i < n; ++i) {
        Z[i] = r > i ? min(Z[2 * l - i], r - i) : 1;
        while (s[i + Z[i]] == s[i - Z[i]]) Z[i]++;
        if (Z[i] + i > r) l = i, r = Z[i] + i;
    }
    for (int i = 0; i < n; ++i) {
        Z[i] = (Z[i] - (i & 1)) / 2 * 2 + (i & 1);
    }
    return Z;
}

```

### 5.5 Suffix Array [ba4998]

```

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.size();
    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) = query_Lcp_min [rk[i] + 1, rk[j] + 1)
    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 [fbc167]

```

int sa[N << 1], rk[N], lcp[N];
// string ASCII value need > 0
namespace sfx {

```

```

bool _t[N << 1];
int _s[N << 1], _c[N << 1], x[N], _p[N], _q[N << 1];
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)
        if (s[i] == s[i + 1]) t[i] = t[i + 1];
        else t[i] = 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);
}
void buildSA(string s) {
    int n = s.length();
    for (int i = 0; i < n; ++i) _s[i] = s[i];
    _s[n] = 0;
    sais(_s, sa, _p, _q, _t, _c, n + 1, 256);
    for (int i = 1; i <= n; ++i) sa[i - 1] = sa[i];
} // buildLCP()...
}

```

## 5.7 Suffix Automaton [7228e9]

```

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 [aa3a61]

```

string rotate(const string &s) {
    int n = (int)s.size(), i = 0, j = 1;
    string t = s + s;
    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 [0518a5]

```

struct PAM {
    int ch[N][26], cnt[N], fail[N], len[N], sz;
    string s;
    // 0 -> even root, 1 -> odd root
    PAM() {}
    void init(string 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 Lyndon Factorization [a9eeb0]

```

// partition s = w[0] + w[1] + ... + w[k-1],
// w[0] >= w[1] >= ... >= w[k-1]
// each w[i] strictly smaller than all its suffix

```

```
vector<string> duval(const string &s) {
    vector<string> ans;
    for (int n = (int)s.size(), i = 0, j, k; i < n; ) {
        for (j = i + 1, k = i; j < n && s[k] <= s[j]; j++)
            k = (s[k] < s[j] ? i : k + 1);
        for (; i <= k; i += j - k)
            ans.pb(s.substr(i, j - k)); // s.substr(l, len)
    }
    return ans;
}
```

### 5.11 Main Lorentz [f3da14]

```
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);
    string 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];
        int 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 - l + 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];
        int 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 Miller Rabin / Pollard Rho [6c9c33]

```
ll mul(ll x, ll y, ll p) {return (x * y - (ll)((long
    double)x / p * y) * p + p) % p;} // __int128
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(l, 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);
}
}
```

### 6.2 Ext GCD [a4b22d]

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

### 6.3 Chinese Remainder Theorem [90d2ce]

```
pair<ll, ll> CRT(ll x1, ll m1, ll x2, ll m2) {
    ll g = gcd(m1, m2);
    if ((x2 - x1) % g) return make_pair(-1, -1); // no sol
    m1 /= g, m2 /= g;
    pair<ll, ll> p = extgcd(m1, m2);
    ll lcm = m1 * m2 * g;
    ll res = p.first * (x2 - x1) * m1 + x1;
    // be careful with overflow
    return make_pair((res % lcm + lcm) % lcm, lcm);
}
```

### 6.4 PiCount [1db46f]

```
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.pb(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]];
}
```



```

11 phi(11 n, int a){
    if(!a) return n;
    if(n < M && a < N) return dp[a][n];
    if(primes[a - 1] > n) return 1;
    if(111 * primes[a - 1] * primes[a - 1] >= n && n < V)
        return small_pi[n] - a + 1;
    return phi(n, a - 1) - phi(n / primes[a - 1], a - 1);
}
11 PiCount(11 n){
    if(n < V) return small_pi[n];
    int s = sqrt(n + 0.5), y = cbrt(n + 0.5), a =
        small_pi[y];
    11 res = phi(n, a) + a - 1;
    for(; primes[a] <= s; ++a) res -= max(PiCount(n /
        primes[a]) - PiCount(primes[a]) + 1, 0);
    return res;
}

```

## 6.5 Linear Function Mod Min [5552e3]

```

11 topos(11 x, 11 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)
11 min_rem(11 n, 11 m, 11 a, 11 b) {
    a = topos(a, m), b = topos(b, m);
    for (11 g = __gcd(a, m); g > 1; ) return g * min_rem(n
        , m / g, a / g, b / g) + (b % g);
    for (11 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 {
            11 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<11, 11> min_rem_pos(11 n, 11 m, 11 a, 11 b) {
    a = topos(a, m), b = topos(b, m);
    11 mn = min_rem(n, m, a, b), g = __gcd(a, m);
    //ax = (mn - b) (mod m)
    11 x = (extgcd(a, m).first + m) * ((mn - b + m) / g)
        % (m / g);
    return {mn, x};
}

```

## 6.6 Floor Sum [49de67]

```

// sum^{n-1}_0 floor((a * i + b) / m) in Log(n + m + a
    + b)
11 floor_sum(11 n, 11 m, 11 a, 11 b) {
    11 ans = 0;
    if (a >= m) ans += (n - 1) * n * (a / m) / 2, a %= m;
    if (b >= m) ans += n * (b / m), b %= m;
    11 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;
}

```

## 6.7 Quadratic Residue [51ec55]

```

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 = (111 * b * b + p - a) % p;
        if (Jacobi(d, p) == -1) break;
    }
    11 f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
    for (int e = (p + 1) >> 1; e; e >>= 1) {
        if (e & 1) {
            tmp = (g0 * f0 + d * (g1 * f1 % p)) % p;
            g1 = (g0 * f1 + g1 * f0) % p;
            g0 = tmp;
        }
        tmp = (f0 * f0 + d * (f1 * f1 % p)) % p;
        f1 = (2 * f0 * f1) % p;
        f0 = tmp;
    }
    return g0;
}

```

## 6.8 Discrete Log [8f7f93]

```

11 DiscreteLog(11 a, 11 b, 11 m) { // a^x = b (mod m)
    const int B = 35000;
    11 k = 1 % m, ans = 0, g;
    while ((g = gcd(a, m)) > 1) {
        if (b == k) return ans;
        if (b % g) return -1;
        b /= g, m /= g, ans++, k = (k * a / g) % m;
    }
    if (b == k) return ans;
    unordered_map<11, int> m1;
    11 tot = 1;
    for (int i = 0; i < B; ++i)
        m1[tot * b % m] = i, tot = tot * a % m;
    11 cur = k * tot % m;
    for (int i = 1; i <= B; ++i, cur = cur * tot % m)
        if (m1.count(cur)) return i * B - m1[cur] + ans;
    return -1;
}

```

## 6.9 Factorial without Prime Factor [c324f3]

```

// O(p^k + Log^2 n), pk = p^k
11 prod[MAXP];
11 fac_no_p(11 n, 11 p, 11 pk) {
    prod[0] = 1;
    for (int i = 1; i <= pk; ++i)
        if (i % p) prod[i] = prod[i - 1] * i % pk;
        else prod[i] = prod[i - 1];
    11 rt = 1;
    for (; n; n /= p) {
        rt = rt * mpow(prod[pk], n / pk, pk) % pk;
        rt = rt * prod[n % pk] % pk;
    }
    return rt;
}
// (n! without factor p) % p^k

```

## 6.10 Berlekamp Massey [f867ec]

```

// need add, sub, mul
vector<int> BerlekampMassey(vector<int> 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<int> v, 11 c) {
        for (int &x : v) x = mul(x, c);
        return v;
    };
    vector<int> c, best;
    int pos = 0, n = (int)a.size();
    for (int i = 0; i < n; ++i) {
        int error = a[i];
        for (int j = 0; j < (int)c.size(); ++j)

```

```

    error = sub(error, mul(c[j], a[i - 1 - j]));
    if (error == 0) continue;
    int inv = Pow(error, mod - 2);
    if (c.empty()) {
        c.resize(i + 1), pos = i, best.pb(inv);
    } else {
        vector<int> 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 < (int)fix.size(); ++j)
            c[j] = add(c[j], fix[j]);
    }
}
return c;
}

```

## 6.11 Simplex [b68fb9]

```

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 (true) {
            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 (true) {
            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;
}
};

```

## 6.12 Euclidean

$$m = \lfloor \frac{an+b}{c} \rfloor$$

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

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

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

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

## 6.13 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  $y \geq 0$ .  
 $x$  and  $y$  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  $x_i - x'_i$

## 6.14 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  $G$ .

- 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  $\sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k)$  holds for every  $1 \leq k \leq n$ . Sequences  $a$  and  $b$  called bigraphic if there is a labeled simple bipartite graph such that  $a$  and  $b$  is the degree sequence of this bipartite graph.

- 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  $1 \leq k \leq n$ . Sequences  $a$  and  $b$  called digraphic if there is a labeled simple directed graph such that each vertex  $v_i$  has indegree  $a_i$  and outdegree  $b_i$ .

- Pick's theorem

For simple polygon, when points are all integer, we have  $A = \#\{\text{lattice points in the interior}\} + \frac{\#\{\text{lattice points on the boundary}\}}{2} - 1$

- 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,  $\theta$ :  $\arcsin(a/r)$ .
- 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  $= 2\pi r h = \pi(a^2 + h^2) = 2\pi r^2(1 - \cos \theta)$ .

## 6.15 Estimation

$n$	2	3	4	5	6	7	8	9	20	30	40	50	100		
$p(n)$	2	3	5	7	11	15	22	30	627	5604	4e4	2e5	2e8		
$n$	100	1e3	1e6				1e9			1e12		1e15		1e18	
$d(i)$	12	32	240				1344			6720		26880		103680	
$arg$	60	840	720720	735134400	963761198400	866421317361600	897612484786617600								
$n$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\binom{2n}{n}$	2	6	20	70	252	924	3432	12870	48620	184756	7e5	2e6	1e7	4e7	1.5e8
$n$	2	3	4	5	6	7	8	9	10	11	12	13			
$B_n$	2	5	15	52	203	877	4140	21147	115975	7e5	4e6	3e7			

## 6.16 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

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

- 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.  $\pi(j) > j$ .

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

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

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

## 7 Polynomial

### 7.1 Number Theoretic Transform [536cc5]

```
// mul, add, sub, Pow
struct NTT {
    int w[N];
    NTT() {
        int dw = Pow(G, (mod - 1) / N);
        w[0] = 1;
        for (int i = 1; i < N; ++i)
            w[i] = mul(w[i - 1], dw);
    }
    void operator()(vector<int>& 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) {
                    int 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());
            int invn = Pow(n, mod - 2);
            for (int i = 0; i < n; ++i)
                a[i] = mul(a[i], invn);
        }
    }
} ntt;
```

### 7.2 Fast Fourier Transform [6f906d]

```
using T = complex<double>;
const double PI = acos(-1);
struct FFT {
    T w[N];
    FFT() {
        T dw = {cos(2 * PI / N), sin(2 * PI / N)};
        w[0] = 1;
        for (int i = 1; i < N; ++i) w[i] = w[i - 1] * dw;
    }
    void operator()(vector<T>& a, bool inv = false) {
        // see NTT, replace ll with T
        if (inv) {
            reverse(a.begin() + 1, a.end());
            T invn = 1.0 / n;
            for (int i = 0; i < n; ++i) a[i] = a[i] * invn;
        }
    }
} ntt;
// after mul, round i.real()
```

### 7.3 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
2061584302081	7	194555039024054273	5
2748779069441	3	9223372036737335297	3

### 7.4 Polynomial Operations [9be4e4]

```
typedef vector<int> Poly;
Poly Mul(Poly a, Poly b, int bound = N) { // d02e42
    int m = a.size() + b.size() - 1, n = 1;
    while (n < m) n <= 1;
    a.resize(n), b.resize(n);
    ntt(a), ntt(b);
    Poly 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;
}
Poly Inverse(Poly a) { // b137d5
    // O(NLogN), a[0] != 0
    int n = a.size();
    Poly res(1, Pow(a[0], mod - 2));
    for (int m = 1; m < n; m <= 1) {
        if (n < m * 2) a.resize(m * 2);
        Poly 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 <Poly, Poly> Divide(Poly a, Poly 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};
    Poly ra = a, rb = b;
    reverse(all(ra)), ra.resize(k);
    reverse(all(rb)), rb.resize(k);
    Poly Q = Mul(ra, Inverse(rb), k);
    reverse(all(Q));
    Poly 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};
}
Poly SqrtImpl(Poly a) { // a642f6
    if (a.empty()) return {0};
    int z = QuadraticResidue(a[0], mod), n = a.size();
    if (z == -1) return {-1};
    Poly 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);
        Poly 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;
}
Poly Sqrt(Poly a) { // 0dae9c
    // O(NLogN), return {-1} if not exists
    int n = a.size(), m = 0;
    while (m < n && a[m] == 0) m++;
    if (m == n) return Poly(n);
    if (m & 1) return {-1};
    Poly s = SqrtImpl(Poly(a.begin() + m, a.end()));
    if (s[0] == -1) return {-1};
    Poly res(n);
    for (int i = 0; i < s.size(); ++i)
        res[i + m / 2] = s[i];
    return res;
}
Poly Derivative(Poly a) { // 26f29b
    int n = a.size();
    Poly res(n - 1);
    for (int i = 0; i < n - 1; ++i)
        res[i] = mul(a[i + 1], i + 1);
    return res;
}
Poly Integral(Poly a) { // f18ba1
    int n = a.size();
    Poly res(n + 1);
    for (int i = 0; i < n; ++i)
        res[i + 1] = mul(a[i], Pow(i + 1, mod - 2));
    return res;
}
Poly Ln(Poly a) { // 0c1381
    // O(NLogN), a[0] = 1
    int n = a.size();
    if (n == 1) return {0};
    Poly d = Derivative(a);
    a.pop_back();
    return Integral(Mul(d, Inverse(a), n - 1));
}
Poly Exp(Poly a) { // d2b129
    // O(NLogN), a[0] = 0
    int n = a.size();
    Poly 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);
        Poly 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;
}
Poly PolyPow(Poly a, ll k) { // d50135
    int n = a.size(), m = 0;
    Poly 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;
    int lead = m * k;
    Poly b(a.begin() + m, a.end());
    int base = Pow(b[0], k), inv = Pow(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 <int> Evaluate(Poly a, vector <int> x) {
    if (x.empty()) return {}; // e28f67
    int n = x.size();
    vector <Poly> 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 <Poly> 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;
    Poly y(n);
    for (int i = 0; i < n; ++i) y[i] = down[i + n][0];
    return y;
}
Poly Interpolate(vector <int> x, vector <int> y) {
    int n = x.size(); // 743f56
    vector <Poly> 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]);
    Poly a = Evaluate(Derivative(up[1]), x);
    for (int i = 0; i < n; ++i)
        a[i] = mul(y[i], Pow(a[i], mod - 2));
    vector <Poly> down(n * 2);
    for (int i = 0; i < n; ++i) down[i + n] = {a[i]};
    for (int i = n - 1; i > 0; --i) {
        Poly lhs = Mul(down[i * 2], up[i * 2 + 1]);
        Poly 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];
}

```

```

}
Poly TaylorShift(Poly a, int c) { // b59bef
    // return sum a_i(x + c)^i;
    // fac[i] = i!, facp[i] = inv(i!)
    int n = a.size();
    for (int i = 0; i < n; ++i) a[i] = mul(a[i], fac[i]);
    reverse(all(a));
    Poly b(n);
    int w = 1;
    for (int i = 0; i < n; ++i)
        b[i] = mul(facp[i], w), w = mul(w, c);
    a = Mul(a, b, n), reverse(all(a));
    for (int i = 0; i < n; ++i) a[i] = mul(a[i], facp[i]);
    return a;
}
vector<int> SamplingShift(vector<int> a, int c, int m){
    // given f(0), f(1), ..., f(n - 1)
    // return f(c), f(c + 1), ..., f(c + m - 1)
    int n = a.size(); // 4d649d
    for (int i = 0; i < n; ++i) a[i] = mul(a[i], facp[i]);
    Poly b(n);
    for (int i = 0; i < n; ++i) {
        b[i] = facp[i];
        if (i & 1) b[i] = sub(0, b[i]);
    }
    a = Mul(a, b, n);
    for (int i = 0; i < n; ++i) a[i] = mul(a[i], fac[i]);
    reverse(all(a));
    int w = 1;
    for (int i = 0; i < n; ++i)
        b[i] = mul(facp[i], w), w = mul(w, sub(c, i));
    a = Mul(a, b, n);
    reverse(all(a));
    for (int i = 0; i < n; ++i) a[i] = mul(a[i], facp[i]);
    a.resize(m), b.resize(m);
    for (int i = 0; i < m; ++i) b[i] = facp[i];
    a = Mul(a, b, m);
    for (int i = 0; i < m; ++i) a[i] = mul(a[i], fac[i]);
    return a;
}

```

## 7.5 Fast Linear Recursion [3f8e4e]

```

int FastLinearRecursion(vector<int> a, vector<int> 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<int> base(n + 1, 1);
    for (int i = 0; i < n; ++i)
        base[i] = sub(0, c[n - i - 1]);
    vector<int> poly(n);
    (n == 1 ? poly[0] = c[n - 1] : poly[1] = 1);
    auto calc = [&](vector<int> p1, vector<int> p2) {
        // O(n^2) brute force or O(n log n) NTT
        return Divide(Mul(p1, p2), base).second;
    };
    vector<int> res(n, 0); res[0] = 1;
    for (; k; k >>= 1, poly = calc(poly, poly)) {
        if (k & 1) res = calc(res, poly);
    }
    int ans = 0;
    for (int i = 0; i < n; ++i)
        ans = add(ans, mul(res[i], a[i]));
    return ans;
}

```

## 7.6 Fast Walsh Transform

```

void fwt(vector<int> &a, bool inv = false) {
    // and : x += y * (1, -1)
    // or : y += x * (1, -1)
    // xor : x = (x + y) * (1, 1/2)
    //      y = (x - y) * (1, 1/2)
    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 ha(n + 1, vector<int>(1 << n));
    vector hb(n + 1, vector<int>(1 << n));
    vector 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)
        or_fwt(ha[i]), or_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)
                c[i + j][k] = add(c[i + j][k],
                    mul(ha[i][k], hb[j][k]));
    for (int i = 0; i <= n; ++i) or_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

```

template<typename T> struct P {};
using Pt = P<ll>;
struct Line { Pt a, b; };
struct Cir { Pt o; double r; };
ll abs2(Pt a) { return a * a; }
double abs(Pt a) { return sqrt(abs2(a)); }
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; }
int pos(Pt a)
{ return sign(a.y) == 0 ? sign(a.x) < 0 : a.y < 0; }
bool cmp(Pt a, Pt b)
{ return pos(a) == pos(b) ? sign(a ^ b) > 0 :
    pos(a) < pos(b); }
bool same_vec(Pt a, Pt b, int d) // d = 1: check dir
{ return sign(a ^ b) == 0 && sign(a * b) > d * 2 - 2; }
bool same_vec(Line a, Line b, int d)
{ return same_vec(a.b - a.a, b.b - b.a, d); }
Pt perp(Pt a) { return Pt(-a.y, a.x); } // CCW 90 deg
// double part
double theta(Pt a)
{ return normalize(atan2(a.y, a.x)); }
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_vec(Pt a, Pt b, Pt c) // vector ac proj to ab
{ return (b - a) * ((c - a) * (b - a)) / (abs2(b - a)); }
Pt proj_pt(Pt a, Pt b, Pt c) // point c proj to ab
{ return proj_vec(a, b, c) + a; }

```

### 8.2 SVG Writer

```

#ifndef ABS
class SVG { // SVG("test.svg", 0, 0, 10, 10)
    void p(string_view s) { o << s; }
    void p(string_view s, auto v, auto... vs) {
        auto i = s.find('$');
        o << s.substr(0, i) << v, p(s.substr(i + 1), vs...)
        ;
    }
    ofstream o; string c = "red";
public:
    SVG(auto f, auto x1, auto y1, auto x2, auto y2) : o(f) {
        p("<svg xmlns='http://www.w3.org/2000/svg' "
            "viewBox='$ $ $ $'>\n"
            "<style>{*stroke-width:0.5%;}</style>\n",
            x1, -y2, x2 - x1, y2 - y1); }
    ~SVG() { p("</svg>\n"); }
    void color(string nc) { c = nc; }
    void line(auto x1, auto y1, auto x2, auto y2) {

```



```

    p("<line x1='$' y1='$' x2='$' y2='$' stroke='$'>\n
    ",
    x1, -y1, x2, -y2, c); }
void circle(auto x, auto y, auto r) {
    p("<circle cx='$' cy='$' r='$' stroke='$' "
    "fill='none'/>\n", x, -y, r, c); }
void text(auto x, auto y, string s, int w = 12) {
    p("<text x='$' y='$' font-size='$px'>$</text>\n",
    x, -y, w, s); }
}; // write wrapper for complex if use complex
#else
struct SVG { SVG(auto ...) {} }; // you know how to
#endif

```

### 8.3 Heart [043c0d]

```

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; }

```

### 8.4 External Bisector [caf92]

```

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;
}

```

### 8.5 Intersections [45b486]

```

// m=0: segment, m=1: ray from L.a to L.b, m=2: Line
bool lines_intersect_check(Line l1, int m1, Line l2,
    int m2, int strict) {
    auto on = [&](Line l, int m, Pt p) {
        if (ori(l.a, l.b, p) != 0) return false;
        if (m && abs2(l.a - p) > abs2(l.b - p)) return true;
        return m == 2 || sign((p - l.a) * (p - l.b)) <= -
            strict;
    };
    if (same_vec(l1, l2, 0)) {
        return on(l1, m1, l2.a) || on(l1, m1, l2.b) ||
            on(l2, m2, l1.a) || on(l2, m2, l1.b);
    }
    auto good = [&](Line l, int m, Line o) {
        if (m && abs((l.a - o.a) ^ (l.a - o.b)) > abs((l.b
            - o.a) ^ (l.b - o.b))) return true;
        return m == 2 || ori(l.a, o.a, o.b) * ori(l.b, o.a,
            o.b) == -1;
    };
    if (good(l1, m1, l2) && good(l2, m2, l1)) return 1;
    if (!strict) {
        if (m2 != 2 && on(l1, m1, l2.a)) return 1;
        if (m2 == 0 && on(l1, m1, l2.b)) return 1;
        if (m1 != 2 && on(l2, m2, l1.a)) return 1;
        if (m1 == 0 && on(l2, m2, l1.b)) return 1;
    }
    return 0;
}
// notice two lines are parallel
auto lines_intersect(Line a, Line b) {
    auto abc = (a.b - a.a) ^ (b.a - a.a);

```

```

    auto abd = (a.b - a.a) ^ (b.b - a.a);
    return make_pair((b.b * abc - b.a * abd), abc - abd);
}
// res[0] -> res[1] and L.a -> L.b: same direction
vector<Pt> circle_line_intersect(Cir c, Line l) {
    Pt p = l.a + (l.b - l.a) * ((c.o - l.a) * (l.b - l.a)
        ) / abs2(l.b - l.a);
    double s = (l.b - l.a) ^ (c.o - l.a), h2 = c.r * c.r
        - s * s / abs2(l.b - l.a);
    if (sign(h2) == -1) return {};
    if (sign(h2) == 0) return {p};
    Pt h = (l.b - l.a) / abs(l.b - l.a) * sqrt(h2);
    return {p - h, p + h};
}
// covered area of c1: arc from res[0] to res[1], CCW
vector<Pt> circles_intersect(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 = perp(c2.o - c1.o) * A / (2 * d2);
    if (sign(v.x) == 0 && sign(v.y) == 0) return {u};
    return {u - v, u + v};
}

```

### 8.6 Intersection Area of Polygon and Circle [205583]

```

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 = 0.5 * a * r * sin(theta) + (C - theta) / 2 * r
            * r;
    } else S = 0.5 * sin(C) * a * b;
    return S;
}
double area_poly_circle(vector<Pt> poly, Pt O, double r
    ) {
    double S = 0; int n = sz(poly);
    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);
}

```

### 8.7 Tangents [296ccd]

```

auto circle_point_tangent(Cir c, Pt p) {
    vector<Line> res;
    double d_sq = abs2(p - c.o);
    if (sign(d_sq - c.r * c.r) == 0) {
        res.pb({p, p + perp(p - c.o)});
    } else if (d_sq > c.r * c.r) {
        double s = d_sq - c.r * c.r;
        Pt v = p + (c.o - p) * s / d_sq;
        Pt u = perp(c.o - p) * sqrt(s) * c.r / d_sq;
        res.pb({p, v + u});
        res.pb({p, v - u});
    }
    return res;
}
auto circles_tangent(Cir c1, Cir c2, int sign1) {
    // sign1 = 1 for outer tang, -1 for inter tang
    vector<Line> res;
    double d_sq = abs2(c1.o - c2.o);
    if (sign(d_sq) == 0) return res;
    double d = sqrt(d_sq);
    Pt v = (c2.o - c1.o) / d;

```

```

double c = (c1.r - sign1 * c2.r) / d;
if (c * c > 1) return res;
double h = sqrt(max(0.0, 1.0 - c * c));
for (int sign2 = 1; sign2 >= -1; sign2 -= 2) {
    Pt n = Pt(v.x * c - sign2 * h * v.y, v.y * c +
        sign2 * h * v.x);
    Pt p1 = c1.o + n * c1.r;
    Pt p2 = c2.o + n * (c2.r * sign1);
    if (sign(p1.x - p2.x) == 0 && sign(p1.y - p2.y) ==
        0)
        p2 = p1 + perp(c2.o - c1.o);
    res.pb({p1, p2});
}
return res;
}
/* The point should be strictly out of hull
return arbitrary point on the tangent line */
pii point_convex_tangent(vector<Pt> &C, Pt p) {
    auto gao = [&](int s) {
        return cyc_tsearch(sz(C), [&](int x, int y)
            { return ori(p, C[x], C[y]) == s; });
    };
    return pii(gao(1), gao(-1));
}
// return (a, b), ori(p, C[a], C[b]) >= 0

```

### 8.8 Point In Convex [722991]

```

bool point_in_convex(vector<Pt> &C, Pt p, bool strict =
    true) {
    // only works when no three points are collinear
    int a = 1, b = sz(C) - 1, r = !strict;
    if (sz(C) == 0) return false;
    if (sz(C) < 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;
}

```

### 8.9 Point In Circle [e1f436]

```

// return q's relation with circumcircle of tri(p[0], p
[1], p[2])
bool in_cc(array<Pt, 3> p, Pt q) {
    __int128 det = 0;
    for (int i = 0; i < 3; ++i)
        det += __int128(abs2(p[i]) - abs2(q)) * ((p[(i + 1)
            % 3] - q) ^ (p[(i + 2) % 3] - q));
    return det > 0; // in: >0, on: =0, out: <0
}

```

### 8.10 Point Segment Distance [4249fd]

```

double point_segment_dist(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));
}

```

### 8.11 Convex Hull [d490c0]

```

auto convex_hull(vector<Pt> pts) {
    sort(all(pts), [&](Pt a, Pt b)
        { return a.x == b.x ? a.y < b.y : a.x < b.x; });
    vector<Pt> ans = {pts[0]};
    for (int t = 0; t < 2; ++t, reverse(all(pts))) {
        for (int i = 1, m = sz(ans); i < sz(pts); ++i) {
            while (sz(ans) > m && ori(ans[sz(ans) - 2],
                ans.back(), pts[i]) <= 0) ans.pop_back();
            ans.pb(pts[i]);
        }
    }
    if (sz(ans) > 1) ans.pop_back();
    return ans;
}

```

### 8.12 Minimum Enclosing Circle [2db817]

```

Cir min_enclosing(vector<Pt> p) {
    random_shuffle(all(p));
    double r = 0.0;
    Pt cent = p[0];
    for (int i = 1; i < sz(p); ++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 = circcenter(p[i], p[j], p[k]);
                r = abs2(p[k] - cent);
            }
        }
    }
    return {cent, sqrt(r)};
}

```

### 8.13 Union of Circles [e5c3ee]

```

// notice identical circles, compare cross -> x if the
precision is bad
auto circles_border(vector<Cir> c, int id) {
    vector<pair<Pt, int>> vec;
    int base = 0;
    for (int i = 0; i < sz(c); ++i) if (id != i) {
        if (sign(c[id].r - c[i].r) < 0 && abs2(c[id].o - c[
            i].o) <= (c[id].r - c[i].r) * (c[id].r - c[i].r
                )) base++;
        auto tmp = circles_intersect(c[id], c[i]);
        if (sz(tmp) == 2) {
            Pt l = tmp[0] - c[id].o, r = tmp[1] - c[id].o;
            vec.emplace_back(l, 1);
            vec.emplace_back(r, -1);
            if (cmp(r, l)) base++;
        }
    }
    vec.emplace_back(Pt(-c[id].r, 0), 0);
    sort(all(vec), [&](auto i, auto j) {
        return cmp(i.first, j.first);
    });
    vector<pair<Pt, Pt>> seg;
    Pt v = Pt(c[id].r, 0), lst = v;
    for (auto [cur, val] : vec) {
        if (base == 0) seg.emplace_back(lst, cur);
        lst = cur, base += val;
    }
    if (base == 0) seg.emplace_back(lst, v);
    for (auto &[l, r] : seg)
        l = l + c[id].o, r = r + c[id].o;
    return seg;
}
double circles_union_area(vector<Cir> c) {
    double res = 0;
    for (int i = 0; i < sz(c); ++i) {
        auto seg = circles_border(c, i);
        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 [l, r] : seg) {
            double tl = theta(1 - c[i].o), tr = theta(r - c[
                i].o);
            if (sign(tl - tr) > 0) tr += PI * 2;
            res += F(tr) - F(tl);
        }
    }
    return res / 2;
}

```

### 8.14 Union of Polygons [eb0eb8]

```

// in CCW order, use index as tiebreaker when collinear
auto polys_border(vector<vector<Pt>> poly, int id) {
    auto get = [&](auto &p, int i) {
        return make_pair(p[i], p[(i + 1) % sz(p)]);
    };
    vector<pair<Pt, Pt>> seg;
    for (int e = 0; e < sz(poly[id]); ++e) {
        auto [s, t] = get(poly[id], e);
        vector<pair<Pt, int>> vec;
        vec.emplace_back(s, -1 << 30);
        vec.emplace_back(t, 1 << 30);
    }
}

```

```

for (int i = 0; i < sz(poly); ++i) {
    int st = find_if(all(poly[i]), [&](Pt p) {
        return ori(p, s, t) == 1; }) - poly[i].begin();
    if (st == sz(poly[i])) continue;
    for (int j = st; j < st + sz(poly[i]); ++j) {
        auto [a, b] = get(poly[i], j % sz(poly[i]));
        if (same_vec(a - b, s - t, -1)) {
            if (ori(a, b, s) == 0 && same_vec(a - b, s -
                t, 1) && i <= id) {
                vec.emplace_back(a, -1);
                vec.emplace_back(b, 1);
            }
        }
        else {
            int s1 = ori(a, s, t) == 1, s2 = ori(b, s, t)
                == 1;
            if (s1 ^ s2) {
                auto p = lines_intersect({a, b}, {s, t});
                vec.emplace_back(p, s1 ? 1 : -1);
            }
        }
    }
}
Pt v = s - t;
sort(all(vec), [&](auto i, auto j) {
    if (i.first == j.first) return i.second > j.
        second;
    if (v.x > 0) return i.first.x > j.first.x;
    else if (v.x < 0) return i.first.x < j.first.x;
    else if (v.y > 0) return i.first.y > j.first.y;
    else return i.first.y < j.first.y;
});
int base = 1 << 30; Pt lst(0, 0);
for (auto [cur, val] : vec) {
    if (!base) seg.emplace_back(lst, cur);
    lst = cur, base += val;
}
return seg;
}
double polys_union_area(vector<vector<Pt>> poly) {
    double res = 0;
    for (int i = 0; i < sz(poly); ++i) {
        auto seg = polys_border(poly, i);
        for (auto [l, r] : seg) res += 1 ^ r;
    }
    return res / 2;
}

```

### 8.15 Rotating SweepLine [14c676]

```

struct Event {
    Pt d; int u, v;
    bool operator < (const Event &b) {
        return sign(d ^ b.d) > 0; }
};
Pt ref(Pt o) {return pos(o) == 1 ? Pt(-o.x, -o.y) : o;}
void rotating_sweepline(vector<Pt> pt) {
    int n = sz(pt);
    vector<int> ord(n), pos(n);
    vector<Event> e;
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j) if (i ^ j)
            e.pb({ref(pt[i] - pt[j]), i, j});
    sort(all(e));
    iota(all(ord), 0);
    sort(all(ord), [&](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[ord[i]] = i;
    auto makeReverse = [](auto v) {
        sort(all(v)), v.resize(unique(all(v)) - v.begin());
        vector<pii> segs;
        for (int i = 0, j = 0; i < sz(v); i = j) {
            for (; j < sz(v) && v[j] - v[i] <= j - i; ++j);
            segs.emplace_back(v[i], v[j - 1] + 1 + 1);
        }
        return segs;
    };
    for (int i = 0, j = 0; i < sz(e); i = j) {
        vector<int> tmp;
        for (; j < sz(e) && !(e[i] < e[j]); j++)
            tmp.pb(min(pos[e[j].u], pos[e[j].v]));
    }
}

```

```

for (auto [l, r] : makeReverse(tmp)) {
    reverse(ord.begin() + l, ord.begin() + r);
    for (int t = l; t < r; ++t) pos[ord[t]] = t;
    // update value here
}
}
}

```

### 8.16 Half Plane Intersection [f6c2b0]

```

/* Having solution, check size > 2 */
/* --^-- Line.a --^-- Line.b --^-- */
auto halfplane_intersection(vector<Line> arr) {
    auto area_pair = [&](Line a, Line b) {
        return make_pair((a.b - a.a) ^ (b.a - a.a),
            (a.b - a.a) ^ (b.b - a.a)); };
    auto isin = [&](Line l0, Line l1, Line l2) {
        // Check inter(L1, L2) strictly in l0
        auto [a02X, a02Y] = area_pair(l0, l2);
        auto [a12X, a12Y] = area_pair(l1, l2);
        if (a12X - a12Y < 0) a12X *= -1, a12Y *= -1;
        return (__int128)a02Y * a12X -
            (__int128)a02X * a12Y > 0; // C^4
    };
    sort(all(arr), [&](Line a, Line b) {
        if (same_vec(a, b, 1))
            return ori(a.a, a.b, b.b) < 0;
        return cmp(a.b - a.a, b.b - b.a); });
    deque<Line> dq(1, arr[0]);
    auto pop_back = [&](int t, Line p) {
        while (sz(dq) >= t && !isin(p, dq[sz(dq) - 2], dq.
            back()))
            dq.pop_back(); };
    auto pop_front = [&](int t, Line p) {
        while (sz(dq) >= t && !isin(p, dq[0], dq[1]))
            dq.pop_front(); };
    for (auto p : arr)
        if (!same_vec(dq.back(), p, 1))
            pop_back(2, p), pop_front(2, p), dq.pb(p);
    pop_back(3, dq[0]), pop_front(3, dq.back());
    return vector<Line>(all(dq));
}

```

### 8.17 Minkowski Sum [2ff069]

```

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());
}
auto minkowski(vector<Pt> P, vector<Pt> Q) {
    // P, Q: convex polygon, CCW order
    reorder(P), reorder(Q); int n = sz(P), m = sz(Q);
    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;
}

```

### 8.18 Vector In Polygon [6dac08]

```

// ori(a, b, c) >= 0, valid: "strict" angle from a-b to
// a-c
bool btwangle(Pt a, Pt b, Pt c, Pt p, int strict) {
    return ori(a, b, p) >= strict && ori(a, p, c) >=
        strict;
}
// whether vector{cur, p} in counter-clockwise order
// prv, cur, nxt
bool inside(Pt prv, Pt cur, Pt nxt, Pt p, int strict) {
    if (ori(cur, nxt, prv) >= 0)
        return btwangle(cur, nxt, prv, p, strict);
    return !btwangle(cur, prv, nxt, p, !strict);
}

```

### 8.19 Delaunay Triangulation [772ff6]

```

/* Delaunay Triangulation:
Given a sets of points on 2D plane, find a
triangulation such that no points will strictly
inside circumcircle of any triangle. */
struct Edge {
    int id; // oidx[id]
    list<Edge>::iterator twin;
    Edge (int _id = 0) : id(_id) {}
};

struct Delaunay { // 0-base
    int n;
    vector<int> oidx;
    vector<list<Edge>> head; // result udir. graph
    vector<Pt> p;
    Delaunay (vector<Pt> _p) : n(sz(_p)), oidx(n), head(n
        ), p(_p) {
        iota(all(oidx), 0);
        sort(all(oidx), [&](int a, int b) {
            return make_pair(_p[a].x, _p[a].y) < make_pair(_p
                [b].x, _p[b].y); });
        for (int i = 0; i < n; ++i) p[i] = _p[oidx[i]];
        divide(0, n - 1);
    }

    void add_edge(int u, int v) {
        head[u].push_front(Edge(v));
        head[v].push_front(Edge(u));
        head[u].begin()->twin = head[v].begin();
        head[v].begin()->twin = head[u].begin();
    }

    void divide(int l, int r) {
        if (l == r) return;
        if (l + 1 == r) return add_edge(l, l + 1);
        int mid = (l + r) >> 1, nw[2] = {l, r};
        divide(l, mid), divide(mid + 1, r);
        auto gao = [&](int t) {
            Pt pt[2] = {p[nw[0]], p[nw[1]]};
            for (auto it : head[nw[t]]) {
                int v = ori(pt[1], pt[0], p[it.id]);
                if (v > 0 || (v == 0 && abs2(pt[t ^ 1] - p[it.
                    id]) < abs2(pt[1] - pt[0])))
                    return nw[t] = it.id, true;
            }
            return false;
        };
        while (gao(0) || gao(1));
        add_edge(nw[0], nw[1]); // add tangent
        while (true) {
            Pt pt[2] = {p[nw[0]], p[nw[1]]};
            int ch = -1, sd = 0;
            for (int t = 0; t < 2; ++t)
                for (auto it : head[nw[t]])
                    if (ori(pt[0], pt[1], p[it.id]) > 0 && (
                        ch == -1 || in_cc({pt[0], pt[1], p[ch
                            ]}, p[it.id])))
                        ch = it.id, sd = t;
            if (ch == -1) break; // upper common tangent
            for (auto it = head[nw[sd]].begin(); it != head[
                nw[sd]].end(); )
                if (lines_intersect_check({pt[sd], p[it->id]},
                    0, {pt[sd ^ 1], p[ch]}, 0, 1))
                    head[it->id].erase(it->twin), head[nw[sd]].
                        erase(it++);
                else ++it;
            nw[sd] = ch, add_edge(nw[0], nw[1]);
        }
    }
};

```

## 8.20 Triangulation Voronoi [46f248]

```

// all coord. is even, half plane intersection
auto build_voronoi_line(vector<Pt> arr) {
    int n = sz(arr);
    Delaunay tool(arr);
    vector<vector<Line>> vec(n);
    for (int i = 0; i < n; ++i)
        for (auto e : tool.head[i]) {
            int u = tool.oidx[i], v = tool.oidx[e.id];
            Pt m = (arr[v] + arr[u]) / 2, d = perp(arr[v] -
                arr[u]);
            vec[u].pb(Line{m, m + d});
        }
}

```

```

return vec;
}

```

## 8.21 3D Point

```

struct Pt {
    double x, y, z;
    Pt(double _x = 0, double _y = 0, double _z = 0) : x(_x
        ), y(_y), z(_z){}
    Pt operator + (const Pt &o) const
    { return Pt(x + o.x, y + o.y, z + o.z); }
    Pt operator - (const Pt &o) const
    { return Pt(x - o.x, y - o.y, z - o.z); }
    Pt operator * (const double &k) const
    { return Pt(x * k, y * k, z * k); }
    Pt operator / (const double &k) const
    { return Pt(x / k, y / k, z / k); }
    double operator * (const Pt &o) const
    { return x * o.x + y * o.y + z * o.z; }
    Pt operator ^ (const Pt &o) const
    { return {Pt(y * o.z - z * o.y, z * o.x - x * o.z, x
        * o.y - y * o.x)}; }
};

double abs2(Pt o) { return o * o; }
double abs(Pt o) { return sqrt(abs2(o)); }
Pt cross3(Pt a, Pt b, Pt c)
{ return (b - a) ^ (c - a); }
double area(Pt a, Pt b, Pt c)
{ return abs(cross3(a, b, c)); }
double volume(Pt a, Pt b, Pt c, Pt d)
{ return cross3(a, b, c) * (d - a); }
bool coplaner(Pt a, Pt b, Pt c, Pt d)
{ return sign(volume(a, b, c, d)) == 0; }
Pt proj(Pt o, Pt a, Pt b, Pt c) // o proj to plane abc
{ Pt n = cross3(a, b, c);
    return o - n * ((o - a) * (n / abs2(n))); }
Pt line_plane_intersect(Pt u, Pt v, Pt a, Pt b, Pt c) {
    // intersection of line uv and plane abc
    Pt n = cross3(a, b, c);
    double s = n * (u - v);
    if (sign(s) == 0) return {-1, -1, -1}; // not found
    return v + (u - v) * ((n * (a - v)) / s); }

```

## 8.22 3D Convex Hull [c794fc]

```

struct Face {
    int a, b, c;
    Face(int _a, int _b, int _c) : a(_a), b(_b), c(_c) {}
};

auto preprocess(auto pt) {
    auto G = pt.begin();
    vector<int> id;
    auto fail = tuple{-1, -1, -1, id};
    int a = find_if(all(pt), [&](Pt z) {
        return z != *G; }) - G;
    if (a == sz(pt)) return fail;
    int b = find_if(all(pt), [&](Pt z) {
        return cross3(*G, pt[a], z) != Pt(0, 0, 0); }) - G;
    if (b == sz(pt)) return fail;
    int c = find_if(all(pt), [&](Pt z) {
        return sign(volume(*G, pt[a], pt[b], z)) != 0; }) -
        G;
    if (c == sz(pt)) return fail;
    for (int i = 0; i < sz(pt); i++)
        if (i != a && i != b && i != c) id.pb(i);
    return tuple{a, b, c, id};
}

// return the faces with pt indexes
vector<Face> convex_hull_3D(vector<Pt> pt) {
    int n = sz(pt);
    if (n <= 3) return {}; // be careful about edge case
    vector<Face> now;
    vector<vector<int>> z(n, vector<int>(n));
    auto [a, b, c, ord] = preprocess(pt);
    if (a == -1) return {};
    now.emplace_back(a, b, c); now.emplace_back(c, b, a);
    for (auto i : ord) {
        vector<Face> nxt;
        for (auto &f : now) {
            auto v = volume(pt[f.a], pt[f.b], pt[f.c], pt[i])
                ;
            if (sign(v) <= 0) nxt.pb(f);
        }
    }
}

```

```

    z[f.a][f.b] = z[f.b][f.c] = z[f.c][f.a] = sign(v)
    ;
}
auto F = [&](int x, int y) {
    if (z[x][y] > 0 && z[y][x] <= 0)
        nxt.emplace_back(x, y, i);
};
for (auto &f : now)
    F(f.a, f.b), F(f.b, f.c), F(f.c, f.a);
now = nxt;
}
return now;
}
// n^2 delaunay: facets with negative z normal of
// convexhull of (x, y, x^2 + y^2), use a pseudo-point
// (0, 0, inf) to avoid degenerate case
// test @ SPOJ CH3D
// double area = 0, vol = 0; // surface area / volume
// for (auto [a, b, c]: faces)
//     area += abs(ver(p[a], p[b], p[c]))/2.0,
//     vol += volume(P3(0, 0, 0), p[a], p[b], p[c])/6.0;

```

## 9 Else

### 9.1 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;
__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);
*oset.find_by_order(1), oset.order_of_key(1); // 4 0
bitset<100> BS;
BS.flip(3), BS.flip(5);
BS._Find_first(), BS._Find_next(3); // 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;
rp2[0], rp2[1]; // 3 4

```

### 9.2 Bit Hack

```

ll next_perm(ll v) { ll t = v | (v - 1);
    return (t + 1) |
        (((~t & ~t) - 1) >> (__builtin_ctz(v) + 1)); }

```

### 9.3 Smawk Algorithm [5a33b4]

```

ll f(int l, int r) { }
bool select(int r, int u, int v) {
    // if f(r, v) is better than f(r, u), return true
    return f(r, u) < f(r, v);
}
// For all 2x2 submatrix: (x < y => y is better than x)
// If M[1][0] < M[1][1], M[0][0] < M[0][1]
// If M[1][0] == M[1][1], M[0][0] <= M[0][1]
// M[i][ans_i] is the best value in the i-th row
vector<int> solve(vector<int> &r, vector<int> &c) {
    const int n = r.size();
    if (n == 0) return {};
    vector<int> c2;
    for (const int &i : c) {
        while (!c2.empty() && select(r[c2.size() - 1], c2.
            back(), i)) c2.pop_back();
        if (c2.size() < n) c2.pb(i);
    }
    vector<int> r2;
    for (int i = 1; i < n; i += 2) r2.pb(r[i]);
    const auto a2 = solve(r2, c2);
    vector<int> ans(n);
    for (int i = 0; i < a2.size(); i++)
        ans[i * 2 + 1] = a2[i];
    int j = 0;

```

```

for (int i = 0; i < n; i += 2) {
    ans[i] = c2[j];
    const int end = i + 1 == n ? c2.back() : ans[i +
        1];
    while (c2[j] != end) {
        j++;
        if (select(r[i], ans[i], c2[j])) ans[i] = c2[j];
    }
}
return ans;
}
vector<int> smawk(int n, int m) {
    vector<int> row(n), col(m);
    iota(all(row), 0), iota(all(col), 0);
    return solve(row, col);
}

```

### 9.4 Slope Trick [d51078]

```

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 * (1 - x);
                    break;
                }
                cur_val += slope * (1 - (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);
};

```

## 9.5 ALL LCS [5ff948]

```

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 !!
    }
}

```

## 9.6 Hilbert Curve [1274a3]

```

ll hilbert(int n, int x, int y) {
    ll res = 0;
    for (int s = n / 2; s; s >>= 1) {
        int rx = (x & s) > 0;
        int ry = (y & s) > 0;
        res += s * 111 * s * ((3 * rx) ^ ry);
        if (ry == 0) {
            if (rx == 1) x = s - 1 - x, y = s - 1 - y;
            swap(x, y);
        }
    }
    return res;
} // n = 2^k

```

## 9.7 Line Container [673ffd]

```

// only works for integer coordinates!! maintain max
struct Line {
    mutable ll a, b, p;
    bool operator<(const Line &rhs) const { return a <
        rhs.a; }
    bool operator<(ll x) const { return p < x; }
};
struct DynamicHull : multiset<Line, less<>> {
    static const ll kInf = 1e18;
    ll Div(ll a, ll b) { return a / b - ((a ^ b) < 0 && a
        % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) { x->p = kInf; return 0; }
        if (x->a == y->a) x->p = x->b > y->b ? kInf : -kInf;
        else x->p = Div(y->b - x->b, x->a - y->a);
        return x->p >= y->p;
    }
    void addline(ll a, ll b) { // ax + b
        auto z = insert({a, b, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y =
            erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p)
            isect(x, erase(y));
    }
    ll query(ll x) {
        auto l = *lower_bound(x);
        return l.a * x + l.b;
    }
};

```

## 9.8 Min Plus Convolution [b34de3]

```

// a is convex a[i+1]-a[i] <= a[i+2]-a[i+1]
vector<int> min_plus_convolution(vector<int> &a, vector
    <int> &b) {
    int n = a.size(), m = b.size();
    vector<int> c(n + m - 1, INF);

```

```

    auto dc = [&](auto Y, int l, int r, int jl, int jr) {
        if (l > r) return;
        int mid = (l + r) / 2, from = -1, &best = c[mid];
        for (int j = jl; j <= jr; ++j)
            if (int i = mid - j; i >= 0 && i < n)
                if (best > a[i] + b[j])
                    best = a[i] + b[j], from = j;
        Y(Y, l, mid - 1, jl, from);
        Y(Y, mid + 1, r, from, jr);
    };
    return dc(dc, 0, n - 1 + m - 1, 0, m - 1), c;
}

```

## 9.9 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.

## 9.10 Simulated Annealing

```

double factor = 100000;
const int base = 1e9; // remember to run ~ 10 times
for (int it = 1; it <= 1000000; ++it) {
    // ans: answer, nw: current value, rnd(): mt19937 rnd
    ()
    if (exp(-(nw - ans) / factor) >= (double)(rnd() %
        base) / base)
        ans = nw;
    factor *= 0.99995;
}

```

## 9.11 Bitset LCS

```

cin >> n >> m;
for (int i = 1, x; i <= n; ++i)
    cin >> x, p[x].set(i);
for (int i = 1, x; i <= m; ++i) {
    cin >> x, (g = f) |= p[x];
    f.shiftLeftByOne(), f.set(0);
    ((f = g - f) ^= g) &= g;
}
cout << f.count() << '\n';

```

## 9.12 Binary Search On Fraction [765c5a]

```

struct Q {
    ll p, q;
    Q go(Q b, ll d) { return {p + b.p*d, q + b.q*d}; }
};
bool pred(Q);
// returns smallest p/q in [lo, hi] such that
// pred(p/q) is true, and 0 <= p, q <= N
Q frac_bs(ll N) {
    Q lo{0, 1}, hi{1, 0};
    if (pred(lo)) return lo;
    assert(pred(hi));
    bool dir = 1, L = 1, H = 1;
    for (; L || H; dir = !dir) {
        ll len = 0, step = 1;
        for (int t = 0; t < 2 && (t ? step/=2 : step*=2);)
            if (Q mid = hi.go(lo, len + step);
                mid.p > N || mid.q > N || dir ^ pred(mid))
                t++;
            else len += step;
        swap(lo, hi = hi.go(lo, len));
        (dir ? L : H) = !len;
    }
    return dir ? hi : lo;
}

```

## 9.13 Cyclic Ternary Search [9017cc]

```

/* bool pred(int a, int b);
f(0) ~ f(n - 1) is a cyclic-shift U-function
return idx s.t. pred(x, idx) is false forall x*/
int cyc_tsearch(int n, auto pred) {
    if (n == 1) return 0;
    int l = 0, r = n; bool rv = pred(1, 0);
    while (r - l > 1) {
        int m = (l + r) / 2;
        if (pred(0, m) ? rv: pred(m, (m + 1) % n)) r = m;
        else l = m;
    }
    return pred(1, r % n) ? l : r % n;
}

```

### 9.14 Tree Hash [34aae5]

```

ull seed;
ull shift(ull x) { x ^= x << 13; x ^= x >> 7;
    x ^= x << 17; return x; }
ull dfs(int u, int f) {
    ull sum = seed;
    for (int i : G[u]) if (i != f)
        sum += shift(dfs(i, u));
    return sum;
}

```

### 9.15 Python Misc

```

from [decimal, fractions, math, random] import *
arr = list(map(int, input().split())) # input
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()
S = set(), S.add((a, b)), S.remove((a, b)) # set
if not (a, b) in S:
D = dict(), D[(a, b)] = 1, del D[(a, b)] # dict
for (a, b) in D.items():
arr = [randint(1, C) for i in range(N)]
choice([8, 6, 4, 1]) # random pick one

```