

# Contents

<b>1 Basic</b>	<b>1</b>
1.1 Compiler Shell	1
1.2 Testing Todo List	1
1.3 Debug Macro	1
1.4 Stress Test Shell	1
1.5 Pragma	1
1.6 Fast IO	1
<b>2 Data Structure</b>	<b>1</b>
2.1 Leftist Tree	1
2.2 Splay Tree	2
2.3 Link Cut Tree	2
2.4 Treap	3
2.5 2D Segment Tree	3
2.6 Zkw	4
<b>3 Flow / Matching</b>	<b>4</b>
3.1 Dinic	4
3.2 Min Cost Max Flow	4
3.3 Kuhn Munkres	5
3.4 SW Min Cut	5
3.5 Gomory Hu Tree	5
3.6 Blossom	6
3.7 Weighted Blossom	6
3.8 Flow Model	7
<b>4 Graph</b>	<b>8</b>
4.1 Heavy-Light Decomposition	8
4.2 Centroid Decomposition	8
4.3 Edge BCC	8
4.4 Block Cut Tree	8
4.5 SCC / 2SAT	8
4.6 Virtual Tree	9
4.7 Directed MST	9
4.8 Dominator Tree	9
<b>5 String</b>	<b>10</b>
5.1 Aho-Corasick Automaton	10
5.2 KMP Algorithm	10
5.3 Z Algorithm	10
5.4 Manacher	10
5.5 Suffix Array	10
5.6 Suffix Automaton	10
5.7 Minimum Rotation	11
5.8 Palindrome Tree	11
<b>6 Math</b>	<b>11</b>
6.1 Miller Rabin / Pollard Rho	11
6.2 Simplex	12
<b>7 Geometry</b>	<b>12</b>
7.1 Basic	12
7.2 Segment Intersection	12
7.3 Convex Hull	12
7.4 PolarAngle Sort	13
7.5 Rotating Caliper	13
7.6 Rotating SweepLine	13
7.7 Half Plane Intersection	13
7.8 Minkowski Sum	13
<b>8 Polynomial</b>	<b>13</b>
8.1 Number Theoretic Transform	13
8.2 Primes	14
8.3 Fast Walsh Transform	14
<b>9 Else</b>	<b>14</b>
9.1 Bit Hack	14
9.2 Hilbert Curve	14
9.3 Pbds	14
9.4 Random	14
9.5 Matroid Intersection	14

## 1 Basic

### 1.1 Compiler Shell

```
if [ $# -ne 2 ] ; then
    g++ -std=c++17 -DABS -Wall -Wextra -Wshadow $1.cpp -o $1
else
    g++ -std=c++17 -DABS -Wall -Wextra -Wshadow $1.cpp -o $1 -fsanitize=address
fi
./$1
chmod +x ./run.sh
./run.sh main [1]
```

### 1.2 Testing Todo List

```
0. choose editor
1. shell script
2. __int128, __lg, __builtin_popcount
3. judge speed v.s.local speed
4. CE penalty?
```

### 1.3 Debug Macro

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

### 1.4 Stress Test Shell

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

### 1.5 Pragma

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

### 1.6 Fast IO

```
#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;
}
```

## 2 Data Structure

### 2.1 Leftist Tree

```
struct node {
    ll rk, data, sz, sum;
    node *l, *r;
    node(ll k) : rk(0), data(k), sz(1), l(0), r(0), sum(k) {}
};
ll sz(node *p) { return p ? p->sz : 0; }
ll rk(node *p) { return p ? p->rk : -1; }
ll sum(node *p) { return p ? p->sum : 0; }
```

```

node *merge(node *a, node *b) {
    if (!a || !b) return a ? a : b;
    if (a->data < b->data) swap(a, b);
    a->r = merge(a->r, b);
    if (rk(a->r) > rk(a->l)) swap(a->r, a->l);
    a->rk = rk(a->r) + 1, a->sz = sz(a->l) + sz(a->r) + 1;
    a->sum = sum(a->l) + sum(a->r) + a->data;
    return a;
}

void pop(node *&o) {
    node *tmp = o;
    o = merge(o->l, o->r);
    delete tmp;
}

```

## 2.2 Splay Tree

```

struct Splay {
    int pa[N], ch[N][2], sz[N], rt, _id;
    ll v[N];
    Splay() {}
    void init() {
        rt = 0, pa[0] = ch[0][0] = ch[0][1] = -1;
        sz[0] = 1, v[0] = inf;
    }
    int newnode(int p, int x) {
        int id = _id++;
        v[id] = x, pa[id] = p;
        ch[id][0] = ch[id][1] = -1, sz[id] = 1;
        return id;
    }
    void rotate(int i) {
        int p = pa[i], x = ch[p][1] == i, gp = pa[p], c = ch[i][!x];
        sz[p] -= sz[i], sz[i] += sz[p];
        if (~c) sz[p] += sz[c], pa[c] = p;
        ch[p][x] = c, pa[p] = i;
        pa[i] = gp, ch[i][!x] = p;
        if (~gp) ch[gp][ch[gp][1] == p] = i;
    }
    void splay(int i) {
        while (~pa[i]) {
            int p = pa[i];
            if (~pa[p]) rotate(ch[pa[p]][1] == p ^ ch[p][1] == i ? i : p);
            rotate(i);
        }
        rt = i;
    }
    int lower_bound(int x) {
        int i = rt, last = -1;
        while (true) {
            if (v[i] == x) return splay(i), i;
            if (v[i] > x) {
                last = i;
                if (ch[i][0] == -1) break;
                i = ch[i][0];
            }
            else {
                if (ch[i][1] == -1) break;
                i = ch[i][1];
            }
        }
        splay(i);
        return last; // -1 if not found
    }
    void insert(int x) {
        int i = lower_bound(x);
        if (i == -1) {
            // assert(ch[rt][1] == -1);
            int id = newnode(rt, x);
            ch[rt][1] = id, ++sz[rt];
            splay(id);
        }
        else if (v[i] != x) {
            splay(i);
            int id = newnode(rt, x), c = ch[rt][0];
            ch[rt][0] = id;
            ch[id][0] = c;
            if (~c) pa[c] = id, sz[id] += sz[c];
            ++sz[rt];
        }
    }
}

```

```

    splay(id);
}
};

```

## 2.3 Link Cut Tree

```

// vertex-weighted, path add, path multiply, path query
sum
struct LCT {
    int ch[N][2], pa[N], sz[N], _id;
    ll v[N], lz_mul[N], lz_add[N], sum[N];
    bool rev[N];
    LCT() : _id(1) {}
    int newnode() {
        int x = _id++;
        ch[x][0] = ch[x][1] = pa[x] = 0, sz[x] = 1;
        v[x] = lz_add[x] = sum[x] = 0, lz_mul[x] = 1;
        rev[x] = false;
        return x;
    }
    void pull(int i) {
        sum[i] = v[i], sz[i] = 1;
        if (ch[i][0]) sum[i] += sum[ch[i][0]], sz[i] += sz[ch[i][0]];
        if (ch[i][1]) sum[i] += sum[ch[i][1]], sz[i] += sz[ch[i][1]];
    }
    void push(int i) {
        if (lz_mul[i] != 1) mul(ch[i][0], lz_mul[i]), mul(ch[i][1], lz_mul[i]), lz_mul[i] = 1;
        if (lz_add[i]) add(ch[i][0], lz_add[i]), add(ch[i][1], lz_add[i]), lz_add[i] = 0;
        if (rev[i]) reverse(ch[i][0]), reverse(ch[i][1]), rev[i] = false;
    }
    void reverse(int i) {
        if (!i) return;
        swap(ch[i][0], ch[i][1]);
        rev[i] ^= true;
    }
    bool isrt(int i) { // rt of splay
        if (!pa[i]) return true;
        return ch[pa[i]][0] != i && ch[pa[i]][1] != i;
    }
    void rotate(int i) {
        int p = pa[i], x = ch[p][1] == i, c = ch[i][!x], gp = pa[p];
        if (ch[gp][0] == p) ch[gp][0] = i;
        else if (ch[gp][1] == p) ch[gp][1] = i;
        pa[i] = gp, ch[i][!x] = p, pa[p] = i;
        ch[p][x] = c, pa[c] = p;
        pull(p), pull(i);
    }
    void splay(int i) {
        vector<int> anc;
        anc.push_back(i);
        while (!isrt(anc.back())) anc.push_back(pa[anc.back()]);
        while (!anc.empty()) push(anc.back()), anc.pop_back();
        while (!isrt(i)) {
            int p = pa[i];
            if (!isrt(p)) rotate(ch[p][1] == i ^ ch[pa[p]][1] == p ? i : p);
            rotate(i);
        }
    }
    void access(int i) {
        int last = 0;
        while (i) {
            splay(i), 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), pa[i] = 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;
}
// custom functions
void mul(int i, ll d) {
    if (!i) return;
    v[i] *= d, lz_mul[i] *= d;
    lz_add[i] *= d, sum[i] *= d;
}
void add(int i, ll d) {
    if (!i) return;
    v[i] += d, lz_add[i] += d, sum[i] += d * sz[i];
}
void mul_path(int i, int j, ll d) {
    // assert(findrt(i) == findrt(j));
    makert(i), access(j);
    splay(i), mul(i, d);
}
void add_path(int i, int j, ll d) {
    // assert(findrt(i) == findrt(j));
    makert(i), access(j), splay(i), add(i, d);
}
ll query_path(int i, int j) {
    // assert(findrt(i) == findrt(j));
    makert(i), access(j), splay(i);
    return sum[i];
}
};

```

## 2.4 Treap

```

struct node {
    int data, sz;
    node *l, *r;
    node(int k) : data(k), sz(1), l(0), r(0) {}
    void up() {
        sz = 1;
        if (l) sz += l->sz;
        if (r) sz += r->sz;
    }
    void down() {}
};
int sz(node *a) { return a ? a->sz : 0; }
node *merge(node *a, node *b) {
    if (!a || !b) return a ? a : b;
    if (rand() % (sz(a) + sz(b)) < sz(a))
        return a->down(), a->r = merge(a->r, b), a->up(), a;
    return b->down(), b->l = merge(a, b->l), b->up(), b;
}
void split(node *o, node *&a, node *&b, int k) {
    if (!o) return a = b = 0, void();
    o->down();
    if (o->data <= k)
        a = o, split(o->r, a->r, b, k), a->up();
    else b = o, split(o->l, a, b->l, k), b->up();
}
void split2(node *o, node *&a, node *&b, int k) {
    if (sz(o) <= k) return a = o, b = 0, void();
    o->down();
    if (sz(o->l) + 1 <= k)
        a = o, split2(o->r, a->r, b, k - sz(o->l) - 1);
    else b = o, split2(o->l, a, b->l, k);
    o->up();
}
node *kth(node *o, int k) {
    if (k <= sz(o->l)) return kth(o->l, k);
    if (k == sz(o->l) + 1) return o;
    return kth(o->r, k - sz(o->l) - 1);
}
int Rank(node *o, int key) {
    if (!o) return 0;
    if (o->data < key)

```

```

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

```

## 2.5 2D Segment Tree

```

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

```

```

11 qsum(int _l, int _r, int _l2, int _r2) {
11 res = v.qsum(_l2, _r2);
11 if (_l <= 1 && r <= _r) return res;
res += lz.qsum(_l2, _r2) * (min(r, _r) - max(1, _l)
);
11 if (_l < 1 + r >> 1) res += ch[0]->query(_l, _r,
_l2, _r2);
11 if (1 + r >> 1 < _r) res += ch[1]->query(_l, _r,
_l2, _r2);
return res;
}
};

```

## 2.6 Zkw

```

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

```

## 3 Flow / Matching

### 3.1 Dinic

```

struct Dinic {
const int INF = 1 << 30;
struct edge {
int v, f;
edge(int _v, int _f) : v(_v), f(_f) {}
};
vector<vector<int>> adj;
vector<edge> E;

```

```

vector<int> level;
int n, s, t;
Dinic(int _n, int _s, int _t) : n(_n), s(_s), t(_t)
{adj.resize(n);}
void add_edge(int u, int v, int f) {
adj[u].pb(E.size()), E.pb(edge(v, f));
adj[v].pb(E.size()), E.pb(edge(u, 0));
}
bool bfs() {
level.assign(n, -1);
queue<int> q;
level[s] = 0, q.push(s);
while (!q.empty()) {
int v = q.front(); q.pop();
for (int id : adj[v]) if (E[id].f > 0 && level[E[id].v] == -1) {
level[E[id].v] = level[v] + 1;
q.push(E[id].v);
}
}
return level[t] != -1;
}
int dfs(int v, int minf) {
if (v == t) return minf;
int ans = 0;
for (int id : adj[v]) if (E[id].f > 0 && level[E[id].v] == level[v] + 1) {
int nxf = dfs(E[id].v, min(minf, E[id].f));
minf -= nxf, E[id].f -= nxf;
ans += nxf, E[id ^ 1].f += nxf;
if (!minf) return ans;
}
if (!ans) level[v] = -1;
return ans;
}
int solve() {
int ans = 0;
while (bfs()) ans += dfs(s, INF);
return ans;
}
};

```

### 3.2 Min Cost Max Flow

```

template<typename T>
struct MCMF {
const T INF = 111 << 60;
struct edge {
int v;
T f, c;
edge(int _v, T _f, T _c) : v(_v), f(_f), c(_c) {}
};
vector<edge> E;
vector<vector<int>> adja;
vector<T> dis, pot;
vector<int> rt;
int n, s, t;
MCMF(int _n, int _s, int _t) : n(_n), s(_s), t(_t) {
adj.resize(n);
}
void add_edge(int u, int v, T f, T c) {
adj[u].pb(E.size()), E.pb(edge(v, f, c));
adj[v].pb(E.size()), E.pb(edge(u, 0, -c));
}
bool SPFA() {
rt.assign(n, -1), dis.assign(n, INF);
vector<bool> vis(n, false);
queue<int> q;
q.push(s), dis[s] = 0, vis[s] = true;
while (!q.empty()) {
int v = q.front(); q.pop();
vis[v] = false;
for (int id : adj[v]) if (E[id].f > 0 && dis[E[id].v] > dis[v] + E[id].c + pot[v] - pot[E[id].v]) {
dis[E[id].v] = dis[v] + E[id].c + pot[v] - pot[E[id].v];
rt[E[id].v] = id;
if (!vis[E[id].v]) vis[E[id].v] = true, q.push(E[id].v);
}
}
return dis[t] != INF;
}

```

```

}
bool dijkstra() {
    rt.assign(n, -1), dis.assign(n, INF);
    priority_queue<pair<T, int>, vector<pair<T, int>
        >>, greater<pair<T, int>>> pq;
    dis[s] = 0, pq.emplace(dis[s], s);
    while (!pq.empty()) {
        int d, v; tie(d, v) = pq.top(); pq.pop();
        if (dis[v] < d) continue;
        for (int id : adj[v]) if (E[id].f > 0 && dis[E[id].v] > dis[v] + E[id].c + pot[v] - pot[E[id].v]) {
            dis[E[id].v] = dis[v] + E[id].c + pot[v] - pot[E[id].v];
            rt[E[id].v] = id;
            pq.emplace(dis[E[id].v], E[id].v);
        }
    }
    return dis[t] != INF;
}
pair<T, T> solve() {
    pot.assign(n, 0);
    T cost = 0, flow = 0;
    bool fr = true;
    while ((fr ? SPFA() : dijkstra())) {
        for (int i = 0; i < n; i++) {
            dis[i] += pot[i] - pot[s];
        }
        T add = INF;
        for (int i = t; i != s; i = E[rt[i] ^ 1].v) {
            add = min(add, E[rt[i]].f);
        }
        for (int i = t; i != s; i = E[rt[i] ^ 1].v) {
            E[rt[i]].f -= add, E[rt[i] ^ 1].f += add;
        }
        flow += add, cost += add * dis[t];
        fr = false;
        swap(dis, pot);
    }
    return make_pair(flow, cost);
}
};

```

### 3.3 Kuhn Munkres

```

template<typename T>
struct KM { // 0-based
    T w[N][N], hl[N], hr[N], slk[N];
    T fl[N], fr[N], pre[N]; int n;
    bool vl[N], vr[N];
    const T INF = 1e9;
    queue<int> q;
    KM(int _n) : n(_n) {
        for (int i = 0; i < n; ++i) for (int j = 0; j < n; ++j)
            w[i][j] = -INF;
    }
    void add_edge(int a, int b, int wei) {
        w[a][b] = wei;
    }
    bool check(int x) {
        if (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(slk, slk + n, INF), fill(vl, vl + n, 0), fill(vr, vr + n, 0);
        q.push(s), vr[s] = 1;
        while (1) {
            T d;
            while (!q.empty()) {
                int y = q.front(); q.pop();
                for (int x = 0; x < n; ++x)
                    if (!vl[x] && slk[x] >= (d = hl[x] + hr[y] - w[x][y]))
                        if (pre[x] = y, d) slk[x] = d;
                        else if (!check(x)) return;
            }
            d = INF;
            for (int x = 0; x < n; ++x)
                if (!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(fl, fl + n, -1), fill(fr, fr + n, -1), fill(hr, hr + n, 0);
    for (int i = 0; i < n; ++i) hl[i] = *max_element(w[i], w[i] + n);
    for (int i = 0; i < n; ++i) bfs(i);
    T res = 0;
    for (int i = 0; i < n; ++i) res += w[i][fl[i]];
    return res;
}
};

```

### 3.4 SW Min Cut

```

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

```

### 3.5 Gomory Hu Tree

```

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

```

```

    ans.push_back({i, t, flow.solve(i, t)});
    flow.runbfs(i);
    for (int j = i + 1; j < n; ++j) if (rt[j] == t &&
        flow.vis[j]) {
        rt[j] = i;
    }
}
return ans;
}

```

### 3.6 Blossom

```

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

```

### 3.7 Weighted Blossom

```

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

```



```

while (b <= n_x && st[b]) ++b;
if (b > n_x) ++n_x;
lab[b] = 0, S[b] = 0;
match[b] = match[lca];
flo[b].clear();
flo[b].push_back(lca);
for (int x = u, y; x != lca; x = st[pa[y]])
    flo[b].push_back(x), flo[b].push_back(y = st[
        match[x]]), q_push(y);
reverse(flo[b].begin() + 1, flo[b].end());
for (int x = v, y; x != lca; x = st[pa[y]])
    flo[b].push_back(x), flo[b].push_back(y = st[
        match[x]]), q_push(y);
set_st(b, b);
for (int x = 1; x <= n_x; ++x) g[b][x].w = g[x][b].
    w = 0;
for (int x = 1; x <= n; ++x) flo_from[b][x] = 0;
for (size_t i = 0; i < flo[b].size(); ++i) {
    int xs = flo[b][i];
    for (int x = 1; x <= n_x; ++x)
        if (g[b][x].w == 0 || e_delta(g[xs][x]) <
            e_delta(g[b][x]))
            g[b][x] = g[xs][x], g[x][b] = g[x][xs];
    for (int x = 1; x <= n; ++x)
        if (flo_from[xs][x]) flo_from[b][x] = xs;
}
set_slack(b);
}
void expand_blossom(int b) {
    for (size_t i = 0; i < flo[b].size(); ++i)
        set_st(flo[b][i], flo[b][i]);
    int xr = flo_from[b][g[b][pa[b]].u], pr = get_pr(b,
        xr);
    for (int i = 0; i < pr; i += 2) {
        int xs = flo[b][i], xns = flo[b][i + 1];
        pa[xs] = g[xns][xs].u;
        S[xs] = 1, S[xns] = 0;
        slack[xs] = 0, set_slack(xns);
        q_push(xns);
    }
    S[xr] = 1, pa[xr] = pa[b];
    for (size_t i = pr + 1; i < flo[b].size(); ++i) {
        int xs = flo[b][i];
        S[xs] = -1, set_slack(xs);
    }
    st[b] = 0;
}
bool on_found_edge(const edge &e) {
    int u = st[e.u], v = st[e.v];
    if (S[v] == -1) {
        pa[v] = e.u, S[v] = 1;
        int nu = st[match[v]];
        slack[v] = slack[nu] = 0;
        S[nu] = 0, q_push(nu);
    } else if (S[v] == 0) {
        int lca = get_lca(u, v);
        if (!lca) return augment(u, v), augment(v, u), true;
        else add_blossom(u, lca, v);
    }
    return false;
}
}
bool matching() {
    memset(S + 1, -1, sizeof(int) * n_x);
    memset(slack + 1, 0, sizeof(int) * n_x);
    q = queue<int>();
    for (int x = 1; x <= n_x; ++x)
        if (st[x] == x && !match[x]) pa[x] = 0, S[x] = 0,
            q_push(x);
    if (q.empty()) return false;
    for (; ; ) {
        while (q.size()) {
            int u = q.front(); q.pop();
            if (S[st[u]] == 1) continue;
            for (int v = 1; v <= n; ++v)
                if (g[u][v].w > 0 && st[u] != st[v]) {
                    if (e_delta(g[u][v]) == 0) {
                        if (on_found_edge(g[u][v])) return true;
                    } else update_slack(u, st[v]);
                }
        }
    }
    int d = inf;

```

```

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

```

### 3.8 Flow Model

- Maximum/Minimum flow with lower bound / Circulation problem
  - Construct super source  $S$  and sink  $T$ .
  - For each edge  $(x, y, l, u)$ , connect  $x \rightarrow y$  with capacity  $u - l$ .
  - For each vertex  $v$ , denote by  $in(v)$  the difference between the sum of incoming lower bounds and the sum of outgoing lower bounds.
  - If  $in(v) > 0$ , connect  $S \rightarrow v$  with capacity  $in(v)$ , otherwise, connect  $v \rightarrow T$  with capacity  $-in(v)$ .
    - To maximize, connect  $t \rightarrow s$  with capacity  $\infty$  (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.
- Maximum density induced subgraph
  - Binary search on answer, suppose we're checking answer  $T$
  - Construct a max flow model, let  $K$  be the sum of all weights
  - Connect source  $s \rightarrow v$ ,  $v \in G$  with capacity  $K$
  - For each edge  $(u, v, w)$  in  $G$ , connect  $u \rightarrow v$  and  $v \rightarrow u$  with capacity  $w$
  - For  $v \in G$ , connect it with sink  $v \rightarrow t$  with capacity  $K + 2T - (\sum_{e \in E(v)} w(e)) - 2w(v)$
  - $T$  is a valid answer if the maximum flow  $f < K|V|$
- Minimum weight edge cover
  - For each  $v \in V$  create a copy  $v'$ , and connect  $u' \rightarrow v'$  with weight  $w(u, v)$ .
  - Connect  $v \rightarrow v'$  with weight  $2\mu(v)$ , where  $\mu(v)$  is the cost of the cheapest edge incident to  $v$ .
  - Find the minimum weight perfect matching on  $G'$ .
- Project selection problem
  - If  $p_v > 0$ , create edge  $(s, v)$  with capacity  $p_v$ ; otherwise, create edge  $(v, t)$  with capacity  $-p_v$ .
  - Create edge  $(u, v)$  with capacity  $w$  with  $w$  being the cost of choosing  $u$  without choosing  $v$ .
  - The mincut is equivalent to the maximum profit of a subset of projects.

- 0/1 quadratic programming

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

can be minimized by the mincut of the following graph:

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

## 4 Graph

### 4.1 Heavy-Light Decomposition

```
vector<int> dep, pa, sz, ch, hd, id;
int _id;
void dfs(int i, int p) {
    dep[i] = ~p ? dep[p] + 1 : 0;
    pa[i] = p, sz[i] = 1, ch[i] = -1;
    for (int j : g[i])
        if (j != p) {
            dfs(j, i);
            if (ch[i] == -1 || sz[ch[i]] < sz[j]) ch[i] = j;
            sz[i] += sz[j];
        }
}
void hld(int i, int p, int h) {
    hd[i] = h;
    id[i] = _id++;
    if (~ch[i]) hld(ch[i], i, h);
    for (int j : g[i]) if (j != p && j != ch[i])
        hld(j, i, j);
}
void query(int i, int j) {
    while (hd[i] != hd[j]) {
        if (dep[hd[i]] < dep[hd[j]]) swap(i, j);
        query2(id[hd[i]], id[i] + 1, i = pa[hd[i]]);
    }
    if (dep[i] < dep[j]) swap(i, j);
    query2(id[j], id[i] + 1);
}
```

### 4.2 Centroid Decomposition

```
vector<vector<int>> dis;
vector<int> pa, sz;
vector<bool> vis;
void dfs_sz(int i, int p) {
    sz[i] = 1;
    for (int j : g[i]) if (j != p && !vis[j])
        dfs_sz(j, i), sz[i] += sz[j];
}
void cen(int i, int p, int _n) {
```

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

### 4.3 Edge BCC

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

### 4.4 Block Cut Tree

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

### 4.5 SCC / 2SAT

```
struct SAT {
    vector<vector<int>> g;
    vector<int> dep, low, scc_id;
    vector<bool> is;
    vector<int> stk;
    int n, _id;
    SAT() {}
```



```

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

```

## 4.6 Virtual Tree

```

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

```

## 4.7 Directed MST

```

template <typename T> struct DMST { // 1-based
    T g[maxn][maxn], fw[maxn];
    int n, fr[maxn];
    bool vis[maxn], inc[maxn];
    void clear() {
        for (int i = 0; i < maxn; ++i) {
            for (int j = 0; j < maxn; ++j) g[i][j] = inf;

```

```

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

```

## 4.8 Dominator Tree

```

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

```

```

void dfs(int v) {
    vis[v] = id, rev[id] = v;
    rt[id] = mn[id] = sdom[id] = id, id++;
    for (int u : adj[v]) {
        if (vis[u] == -1) dfs(u), par[vis[u]] = vis[v];
        radj[vis[u]].pb(vis[v]);
    }
}
void build(int s) {
    dfs(s);
    for (int i = id - 1; ~i; --i) {
        for (int u : radj[i]) {
            sdom[i] = min(sdom[i], sdom[query(u, false)]);
        }
        if (i) bucket[sdom[i]].pb(i);
        for (int u : bucket[i]) {
            int p = query(u, false);
            dom[u] = sdom[p] == i ? i : p;
        }
        if (i) rt[i] = par[i];
    }
    vector<int> res(n, -1);
    for (int i = 1; i < id; ++i) {
        if (dom[i] != sdom[i]) dom[i] = dom[dom[i]];
    }
    for (int i = 1; i < id; ++i) res[rev[i]] = rev[dom[i]];
    res[s] = s;
    dom = res;
}
};

```

## 5 String

### 5.1 Aho-Corasick Automaton

```

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

```

```

    }
    return ans;
}
};

```

### 5.2 KMP Algorithm

```

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

```

### 5.3 Z Algorithm

```

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

```

### 5.4 Manacher

```

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

```

### 5.5 Suffix Array

```

int sa[N], tmp[2][N], c[N], rk[N], lcp[N];
void buildSA(string s) {
    int *x = tmp[0], *y = tmp[1], m = 256, n = s.length();
    for (int i = 0; i < m; ++i) c[i] = 0;
    for (int i = 0; i < n; ++i) c[x[i]] = s[i]++;
    for (int i = 1; i < m; ++i) c[i] += c[i - 1];
    for (int i = n - 1; ~i; --i) sa[--c[x[i]]] = i;
    for (int k = 1; k < n; k <= 1) {
        for (int i = 0; i < m; ++i) c[i] = 0;
        for (int i = 0; i < n; ++i) c[x[i]]++;
        for (int i = 1; i < m; ++i) c[i] += c[i - 1];
        int p = 0;
        for (int i = n - k; i < n; ++i) y[p++] = i;
        for (int i = 0; i < n; ++i) if (sa[i] >= k) y[p++] = sa[i] - k;
        for (int i = n - 1; ~i; --i) sa[--c[x[y[i]]]] = y[i];
    }
}

```

```

    y[sa[0]] = p = 0;
    for (int i = 1; i < n; ++i) {
        int a = sa[i], b = sa[i - 1];
        if (!(x[a] == x[b] && a + k < n && b + k < n && x[a + k] == x[b + k])) p++;
        y[sa[i]] = p;
    }
    if (n == p + 1) break;
    swap(x, y), m = p + 1;
}
}

void buildLCP(string s) {
    // lcp[i] = LCP(sa[i - 1], sa[i])
    // lcp[i, j] = min(lcp[rk[i] + 1], lcp[rk[i] + 2], ..., lcp[rk[j]])
    int n = s.length(), val = 0;
    for (int i = 0; i < n; ++i) rk[sa[i]] = i;
    for (int i = 0; i < n; ++i) {
        if (!rk[i]) lcp[rk[i]] = 0;
        else {
            if (val) val--;
            int p = sa[rk[i] - 1];
            while (val + i < n && val + p < n && s[val + i] == s[val + p]) val++;
            lcp[rk[i]] = val;
        }
    }
}
}

```

## 5.6 Suffix Automaton

```

struct SAM {
    int ch[N][26], len[N], link[N], cnt[N], sz;
    SAM () {len[0] = 0, link[0] = -1, sz = 1;}
    void build(string s) {
        int last = 0;
        for (char c : s) {
            int cur = sz++;
            len[cur] = len[last] + 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];
                    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]];
    }
};

```

## 5.7 Minimum Rotation

```

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

```

```

}
int pos = (i < n ? i : j);
return t.substr(pos, n);
}

```

## 5.8 Palindrome Tree

```

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

```

## 6 Math

### 6.1 Miller Rabin / Pollard Rho

```

ll mul(ll x, ll y, ll p) {return (x * y - (ll)((long double)x / p * y) * p + p) % p;}
vector<ll> chk = {2, 325, 9375, 28178, 450775, 9780504, 1795265022};
ll Pow(ll a, ll b, ll n) {ll res = 1; for (; b; b >>= 1, a = mul(a, a, n)) if (b & 1) res = mul(res, a, n); return res;}
bool check(ll a, ll d, int s, ll n) {
    a = Pow(a, d, n);
    if (a <= 1) return 1;
    for (int i = 0; i < s; ++i, a = mul(a, a, n)) {
        if (a == 1) return 0;
        if (a == n - 1) return 1;
    }
    return 0;
}
bool IsPrime(ll n) {
    if (n < 2) return 0;
    if (n % 2 == 0) return n == 2;
    ll d = n - 1, s = 0;
    while (d % 2 == 0) d >>= 1, ++s;
    for (ll i : chk) if (!check(i, d, s, n)) return 0;
    return 1;
}
const vector<ll> small = {2, 3, 5, 7, 11, 13, 17, 19};
ll FindFactor(ll n) {
    if (IsPrime(n)) return 1;
    for (ll p : small) if (n % p == 0) return p;
    ll x, y = 2, d, t = 1;
    auto f = [&](ll a) {return (mul(a, a, n) + t) % n;}
    for (int l = 2; l <= 1) {
        x = y;
        int m = min(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> PollardRho(ll n) {
    map<ll, int> res;
    if (n == 1) return res;
    if (IsPrime(n)) return ++res[n], res;
    ll d = FindFactor(n);
    res = PollardRho(n / d);
    auto res2 = PollardRho(d);
    for (auto [x, y] : res2) res[x] += y;
    return res;
}

```

## 6.2 Simplex

```

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

```

```

        (x == -1 || b[i] / a[i][y] < b[x] / a[x][y]))
            x = i;
            if (x == -1) return 2;
            pivot(x, y);
        }
        for (int i = 0; i < m; ++i) if (Left[i] < n) sol[Left[i]] = b[i];
        return 0;
    }
};

```

## 7 Geometry

### 7.1 Basic

```

int sign(double x) {return abs(x) <= eps ? 0 : (x > 0 ? 1 : -1);}
struct Pt {
    double x, y;
    Pt (double _x, double _y) : x(_x), y(_y) {}
    Pt operator + (Pt o) {return Pt(x + o.x, y + o.y);}
    Pt operator - (Pt o) {return Pt(x - o.x, y - o.y);}
    Pt operator * (double k) {return Pt(x * k, y * k);}
    Pt operator / (double k) {return Pt(x / k, y / k);}
    double operator * (Pt o) {return x * o.x + y * o.y;}
    double operator ^ (Pt o) {return x * o.y - y * o.x;}
    double abs() {return hypot(x, y);}
};
int ori(Pt o, Pt a, Pt b) {return sign((o - a) ^ (o - b));}
bool btw(Pt a, Pt b, Pt c) { // c on segment ab?
    return ori(a, b, c) == 0 && sign((c - a) * (c - b)) <= 0;
}
double area(Pt a, Pt b, Pt c) {return abs((a - b) ^ (a - c)) / 2;}
Pt proj_vector(Pt a, Pt b, Pt c) { // vector ac proj to ab
    return (b - a) * ((c - a) * (b - a)) / ((b - a) * (b - a));
}
Pt proj_pt(Pt a, Pt b, Pt c) { // point c proj to ab
    return proj_vector(a, b, c) + a;
}

```

### 7.2 Segment Intersection

```

bool banana(Pt a, Pt b, Pt c, Pt d) { // segment ab and cd
    if (btw(a, b, c) || btw(a, b, d) || btw(c, d, a) || btw(c, d, b)) return true;
    return ori(a, b, c) * ori(a, b, d) == -1 && ori(c, d, a) * ori(c, d, b) == -1;
}
Pt intersect(Pt a, Pt b, Pt c, Pt d) { // segment ab and cd
    double abc = (b - a) ^ (c - a);
    double abd = (b - a) ^ (d - a);
    if (sign(abc - abd) == 0) return d;
    return (d * abc - c * abd) / (abc - abd);
}

```

### 7.3 Convex Hull

```

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

```

## 7.4 PolarAngle Sort

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

## 7.5 Rotating Caliper

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

## 7.6 Rotating SweepLine

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

## 7.7 Half Plane Intersection

```
vector<Pt> HalfPlaneInter(vector<pair<Pt, Pt>> vec) {
    // x
    // first -----> second
    auto pos = [&](Pt a) {return sign(a.y) == 0 ? sign(a.x) < 0 : sign(a.y) > 0;};
    sort(all(vec), [&](pair<Pt, Pt> a, pair<Pt, Pt> b) {
        Pt A = a.second - a.first, B = b.second - b.first;
        if (pos(A) == pos(B)) {
            if (sign(A ^ B) == 0) return sign((b.first - a.first) * (b.second - a.first)) > 0;
            return sign(A ^ B) > 0;
        }
        return pos(A) < pos(B);
    });
    deque<Pt> inter;
    deque<pair<Pt, Pt>> seg;
    int n = vec.size();
    auto get = [&](pair<Pt, Pt> a, pair<Pt, Pt> b) {
        return intersect(a.first, a.second, b.first, b.second);
    };
}
```

```
for (int i = 0; i < n; ++i) if (!i || vec[i] != vec[i - 1]) {
    while (seg.size() >= 2 && sign((vec[i].second - inter.back()) ^ (vec[i].first - inter.back())) == 1) seg.pop_back(), inter.pop_back();
    while (seg.size() >= 2 && sign((vec[i].second - inter.front()) ^ (vec[i].first - inter.front())) == 1) seg.pop_front(), inter.pop_front();
    seg.push_back(vec[i]);
    if (seg.size() >= 2) inter.pb(get(seg[seg.size() - 2], seg.back()));
}
while (seg.size() >= 2 && sign((seg.front().second - inter.back()) ^ (seg.front().first - inter.back())) == 1) seg.pop_back(), inter.pop_back();
inter.push_back(get(seg.front(), seg.back()));
return vector<Pt>(all(inter));
}
```

## 7.8 Minkowski Sum

```
vector<Pt> Minkowski(vector<Pt> a, vector<Pt> b) {
    a = ConvexHull(a), b = ConvexHull(b);
    int n = a.size(), m = b.size();
    vector<Pt> c = {a[0] + b[0]}, s1, s2;
    for (int i = 0; i < n; ++i) s1.pb(a[(i + 1) % n] - a[i]);
    for (int i = 0; i < m; ++i) s2.pb(b[(i + 1) % m] - b[i]);
    for (int p1 = 0, p2 = 0; p1 < n || p2 < m; ) {
        if (p2 == m || (p1 < n && sign(s1[p1] ^ s2[p2]) >= 0)) {
            c.pb(c.back() + s1[p1++]);
        } else {
            c.pb(c.back() + s2[p2++]);
        }
    }
    return ConvexHull(c);
}
```

## 8 Polynomial

### 8.1 Number Theoretic Transform

```
const int N = 1 << 20, mod = 998244353, G = 3;
void run (vector<ll> &P, bool inv = false) {
    int N = P.size();
    const ll w = modpow(G, (mod - 1) / N);
    int lg = __lg(N);
    vector<int> rev(N);
    for (int i = 1; i < N; ++i) {
        rev[i] = (rev[i >> 1] >> 1) | ((i & 1) << (lg - 1));
        if (i < rev[i]) swap(P[i], P[rev[i]]);
    }
    vector<ll> ws = {inv ? modpow(w, mod - 2) : w};
    for (int i = 1; i < lg; ++i) ws.push_back(ws.back() * ws.back() % mod);
    reverse(ws.begin(), ws.end());
    for (int i = 0; i < lg; ++i) {
        for (int k = 0; k < N; k += (2 << i)) {
            ll base = 1;
            for (int j = k; j < k + (1 << i); ++j, base = base * ws[i] % mod) {
                ll t = base * P[j + (1 << i)] % mod, u = P[j];
                P[j] = u + t, P[j + (1 << i)] = u - t;
                if (P[j] >= mod) P[j] -= mod;
                if (P[j + (1 << i)] < 0) P[j + (1 << i)] += mod;
            }
        }
    }
    if (inv) {
        ll ninv = modpow(N, mod - 2);
        for (int i = 0; i < N; ++i) {
            P[i] = P[i] * ninv % mod;
        }
    }
}
```

## 8.2 Primes

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

## 8.3 Fast Walsh Transform

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

## 9 Else

### 9.1 Bit Hack

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

void subset(long long s) {
    long long sub = s;
    while (sub) sub = (sub - 1) & s;
}
```

### 9.2 Hilbert Curve

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

### 9.3 Pbds

```
#include <ext/pb_ds/priority_queue.hpp>
#include <ext/pb_ds/assoc_container.hpp>
using namespace __gnu_pbds;
#include <ext/rope>
using namespace __gnu_cxx;
int main () {
    __gnu_pbds::priority_queue<int> pq1, pq2;
    pq1.join(pq2); // pq1 += pq2, pq2 = {}
    cc_hash_table<int, int> m1;
    tree<int, null_type, less<int>, rb_tree_tag,
        tree_order_statistics_node_update> oset;
    oset.insert(2), oset.insert(4);
    cout << *oset.find_by_order(1) << ' ' << oset.
        order_of_key(1) << '\n'; // 4 0
    bitset<100> BS;
```

```
BS.flip(3), BS.flip(5);
cout << BS._Find_first() << ' ' << BS._Find_next(3)
    << '\n'; // 3 5
rope<int> rp1, rp2;
rp1.push_back(1), rp1.push_back(3);
rp1.insert(0, 2); // pos, num
rp1.erase(0, 2); // pos, len
rp1.substr(0, 2); // pos, len
rp2.push_back(4);
rp1 += rp2, rp2 = rp1;
cout << rp2[0] << ' ' << rp2[1] << '\n'; // 3 4
}
```

### 9.4 Random

```
struct custom_hash {
    static uint64_t splitmix64(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t a) const {
        static const uint64_t FIXED_RANDOM = chrono::
            steady_clock::now().time_since_epoch().count();
        return splitmix64(a + FIXED_RANDOM);
    }
};
unordered_map<int, int, custom_hash> m1;
random_device rd; mt19937 rng(rd());
```

### 9.5 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.