

# Contents

<b>1 Basic</b>	<b>1</b>
1.1 Default Code	1
1.2 .vimrc	1
<b>2 Data Structure</b>	<b>1</b>
2.1 Heavy-Light Decomposition	1
2.2 Li-Chao Tree	2
2.3 Link Cut Tree	2
2.4 Treap	3
<b>3 Flow Matching</b>	<b>3</b>
3.1 Bounded Flow	3
3.2 Dinic	4
3.3 Gomory Hu	4
3.4 Hungarian Algorithm	5
3.5 ISAP Algorithm	5
3.6 KM Algorithm	5
3.7 Max Simple Graph Matching	5
3.8 MCMF	6
3.9 Min Cost Circulation	6
3.10 SW Mincut	7
<b>4 Geometry</b>	<b>7</b>
4.1 Geometry Template	7
4.2 Convex Hull	7
4.3 Minimum Enclosing Circle	8
4.4 Minkowski Sum	8
4.5 Half Plane Intersection	8
4.6 Dynamic Convex Hull	8
4.7 3D Point	9
4.8 ConvexHull3D	9
4.9 Circle Operations	10
<b>5 Graph</b>	<b>10</b>
5.1 Block Cut Tree	10
5.2 2-SAT	11
5.3 Dominator Tree	11
5.4 Virtual Tree	12
<b>6 Math</b>	<b>12</b>
6.1 Extended Euclidean Algorithm	12
6.2 Floor & Ceil	12
6.3 Legendre	12
6.4 Simplex	12
6.5 Floor Sum	13
6.6 Miller Rabin & Pollard Rho	13
<b>7 Misc</b>	<b>13</b>
7.1 Fraction	13
7.2 Matroid	14
<b>8 Polynomial</b>	<b>14</b>
8.1 FFT	14
8.2 NTT	14
8.3 Polynomial Operation	14
8.4 Generating Function	15
8.4.1 Ordinary Generating Function	15
8.4.2 Exponential Generating Function	16
<b>9 String</b>	<b>16</b>
9.1 Rolling Hash	16
9.2 KMP Algorithm	16
9.3 Manacher Algorithm	16
9.4 MCP	16
9.5 Suffix Array	16
9.6 Suffix Array Automaton	17
9.7 Z-value Algorithm	17
9.8 Main Lorentz	17
9.9 AC Automaton	18

<b>10 Formula</b>	<b>18</b>
10.1 Recurrences	18
10.2 Geometry	18
10.2.1 Triangles	18
10.2.2 Quadrilaterals	18
10.2.3 Spherical coordinates	18
10.2.4 Green's Theorem	19
10.3 Trigonometry	19
10.4 Derivatives/Integrals	19
10.5 Sums	19
10.6 Series	19
10.7 Probability theory	19
10.7.1 Discrete distributions	20
10.7.2 Continuous distributions	20
10.8 Markov chains	20

## 1 Basic

### 1.1 Default Code

```
//Challenge: Accepted
//#pragma GCC optimize("Ofast")
#include <bits/stdc++.h>
using namespace std;

#define io ios_base::sync_with_stdio(0);cin.tie(0);cerr.tie(0)
#define iter(v) v.begin(),v.end()
#define SZ(v) (int)v.size()
#define pb emplace_back
#define mp make_pair
#define ff first
#define ss second

using ll = long long;
using pii = pair<int, int>;
using pll = pair<ll, ll>;

#ifdef zisk
void debug(){cerr << "\n";}
template<class T, class ... U>
void debug(T a, U ... b){cerr << a << " ", debug(b...);}
template<class T> void pary(T l, T r){
    while (l != r) cerr << *l << " ", l++;
    cerr << "\n";
}
#else
#define debug(...) void()
#define pary(...) void()
#endif
template<class A, class B>
ostream& operator<<(ostream& o, pair<A,B> p)
{ return o << '(' << p.ff << ',' << p.ss << ')'; }

int main(){
    io;
}
```

### 1.2 .vimrc

```
sy on
se nu rnu bs=2 sw=4 ts=4 hls ls=2 si acd bo=all mouse=a et
map <F9> :w<bar>!g++ "%" -o %:r -std=c++17 -Wall -Wextra -
    Wshadow -Dzisk -g -fsanitize=undefined,address<CR>
map <F8> :!./%:r<CR>
map <C-a> <ESC>ggVG
inoremap {<CR> {<CR>}<ESC>ko
```

## 2 Data Structure

### 2.1 Heavy-Light Decomposition

```

struct Heavy_light-Decomposition { // 1-base
    int n, up[maxn], dep[maxn], to[maxn], siz[maxn], pa[maxn];
};
int C, ti[maxn], ord[maxn], wdown[maxn], edge[maxn], et = 0;
vector<pii> G[maxn];
void init(int _n) {
    n = _n, C = 0, et = 1;
    for (int i = 1; i <= n; i++)
        G[i].clear(), to[i] = 0;
}
void add_edge(int a, int b, int w) {
    G[a].push_back(pii(b, et)), G[b].push_back(pii(a, et));
    edge[et++] = w;
}
void dfs(int u, int f, int d) {
    siz[u] = 1, pa[u] = f, dep[u] = d;
    for (auto &v: G[u])
        if (v.ff != f) {
            dfs(v.ff, u, d+1), siz[u] += siz[v];
            if (siz[to[u]] < siz[v]) to[u] = v;
        }
}
void cut(int u, int link) {
    ti[u] = C;
    ord[C++] = u, up[u] = link;
    if (!to[u]) return;
    cut(to[u], link);
    for (auto v: G[u]) {
        if (v.ff != pa[u] && v.ff != to[u]) cut(v.ff, v.ff);
    }
}
void build() { dfs(1, 1, 1), cut(1, 1); }
int query(int a, int b) {
    int ta = up[a], tb = up[b], re = 0;
    while (ta != tb)
        if (dep[ta] < dep[tb])
            /*query*/, tb = up[b = pa[tb]];
        else /*query*/, ta = up[a = pa[ta]];
    if (a == b) return re;
    if (ti[a] > ti[b]) swap(a, b);
    /*query*/
    return re;
}
};

```

## 2.2 Li-Chao Tree

```

struct LiChao_min {
    struct line {
        ll m, c;
        line(ll _m = 0, ll _c = 0) {
            m = _m;
            c = _c;
        }
        ll eval(ll x) { return m * x + c; }
    };
    struct node {
        node *l, *r;
        line f;
        node(line v) {
            f = v;
            l = r = NULL;
        }
    };
    typedef node *pnode;
    pnode root;
    int sz;
    #define mid ((l + r) >> 1)
    void insert(line &v, int l, int r, pnode &nd) {
        if (!nd) {
            nd = new node(v);
            return;
        }
        ll trl = nd->f.eval(l), trr = nd->f.eval(r);
        ll vl = v.eval(l), vr = v.eval(r);

```

```

        if (trl <= vl && trr <= vr) return;
        if (trl > vl && trr > vr) {
            nd->f = v;
            return;
        }
        if (trl > vl) swap(nd->f, v);
        if (nd->f.eval(mid) < v.eval(mid))
            insert(v, mid + 1, r, nd->r);
        else swap(nd->f, v), insert(v, l, mid, nd->l);
    }
    ll query(int x, int l, int r, pnode &nd) {
        if (!nd) return inf;
        if (l == r) return nd->f.eval(x);
        if (mid >= x)
            return min(
                nd->f.eval(x), query(x, l, mid, nd->l));
        return min(
            nd->f.eval(x), query(x, mid + 1, r, nd->r));
    }
    /* -sz <= query_x <= sz */
    void init(int _sz) {
        sz = _sz + 1;
        root = NULL;
    }
    void add_line(ll m, ll c) {
        line v(m, c);
        insert(v, -sz, sz, root);
    }
    ll query(ll x) { return query(x, -sz, sz, root); }
};

```

## 2.3 Link Cut Tree

```

struct Splay { // subtree-sum, path-max
    static Splay nil;
    Splay *ch[2], *f;
    int val, rev, size, vir, id, type;
    pii ma;
    Splay(int _val = 0, int _id = 0)
        : val(_val), rev(0), size(0), vir(0), id(_id) {
        ma = make_pair(val, id);
        f = ch[0] = ch[1] = &nil;
        type = 0;
    }
    bool isr() { //is root
        return f->ch[0] != this && f->ch[1] != this;
    }
    int dir() { return f->ch[0] == this ? 0 : 1; }
    void setCh(Splay *c, int d) {
        ch[d] = c;
        if (c != &nil) c->f = this;
        pull();
    }
    void push() {
        if (!rev) return;
        swap(ch[0], ch[1]);
        if (ch[0] != &nil) ch[0]->rev ^= 1;
        if (ch[1] != &nil) ch[1]->rev ^= 1;
        rev = 0;
    }
    void pull() {
        // take care of the nil!
        size = ch[0]->size + ch[1]->size + vir + type;
        ma = max(make_pair(val, id), max(ch[0]->ma, ch[1]->ma));
        if (ch[0] != &nil) ch[0]->f = this;
        if (ch[1] != &nil) ch[1]->f = this;
    }
} Splay::nil;
Splay *nil = &Splay::nil;
void rotate(Splay *x) {
    Splay *p = x->f;
    int d = x->dir();
    if (!p->isr()) p->f->setCh(x, p->dir());
    else x->f = p->f;
    p->setCh(x->ch[!d], d);
}

```

```

    x->setCh(p, !d);
    p->pull(), x->pull();
}
void splay(Splay *x) {
    vector<Splay *> splayVec;
    for (Splay *q = x;; q = q->f) {
        splayVec.pb(q);
        if (q->isr()) break;
    }
    reverse(iter(splayVec));
    for (auto it : splayVec) it->push();
    while (!x->isr()) {
        if (x->f->isr()) rotate(x);
        else if (x->dir() == x->f->dir())
            rotate(x->f), rotate(x);
        else rotate(x), rotate(x);
    }
}
Splay *access(Splay *x) {
    Splay *q = nil;
    for (; x != nil; x = x->f) {
        splay(x);
        x->vir -= q->size; x->vir += x->ch[1]->size;
        x->setCh(q, 1); x->pull();
        q = x;
    }
    return q;
}
void root_path(Splay *x) { access(x), splay(x); }
void chroot(Splay *x) {
    root_path(x), x->rev ^= 1;
    x->push(), x->pull();
}
void split(Splay *x, Splay *y) {
    chroot(x), root_path(y);
}
void link(Splay *x, Splay *y) {
    chroot(x), root_path(y);
    x->f = y; y->vir += x->size;
}
void cut(Splay *x, Splay *y) {
    split(x, y);
    y->push();
    y->ch[0] = y->ch[0]->f = nil;
    y->pull();
}
Splay *get_root(Splay *x) {
    for (root_path(x); x->ch[0] != nil; x = x->ch[0])
        x->push();
    splay(x);
    return x;
}
bool conn(Splay *x, Splay *y) {
    return get_root(x) == get_root(y);
}
Splay *lca(Splay *x, Splay *y) {
    access(x), root_path(y);
    if (y->f == nil) return y;
    return y->f;
}
void change(Splay *x, int val) {
    splay(x), x->val = val, x->pull();
}
pii query(Splay *x, Splay *y) {
    split(x, y);
    return y->ma;
}
}

```

## 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));
}
}

```

## 3 Flow Matching

### 3.1 Bounded Flow

```

struct Dinic { // 1-base
    struct edge {
        int to, cap, flow, rev;
    };
    vector<edge> g[maxN];
    int n, s, t, dis[maxN], ind[maxN], cnt[maxN];
    const int inf = 1e9;

    void init(int _n) {

```

```

    n = _n + 2;
    s = _n + 1, t = _n + 2;
    for (int i = 0; i <= n; ++i) g[i].clear(), cnt[i] = 0;
}
void reset() {
    for (int i = 0; i <= n; ++i)
        for (auto &j : g[i]) j.flow = 0;
}
void add_edge(int u, int v, int lcap, int rcap) {
    cnt[u] -= lcap, cnt[v] += lcap;
    g[u].pb(edge{v, rcap, lcap, (int)g[v].size()});
    g[v].pb(edge{u, 0, 0, (int)g[u].size() - 1});
}
void add_edge(int u, int v, int cap) {
    g[u].pb(edge{v, cap, 0, (int)g[v].size()});
    g[v].pb(edge{u, 0, 0, (int)g[u].size() - 1});
    //change g[v] to cap for undirected graphs
}
bool bfs() {
    fill(dis, dis+n+1, -1);
    queue<int> q;
    q.push(s), dis[s] = 0;
    while (!q.empty()) {
        int cur = q.front(); q.pop();
        for (auto &e : g[cur]) {
            if (dis[e.to] == -1 && e.flow != e.cap) {
                q.push(e.to);
                dis[e.to] = dis[cur] + 1;
            }
        }
    }
    return dis[t] != -1;
}
int dfs(int u, int cap) {
    if (u == t || !cap) return cap;
    for (int &i = ind[u]; i < (int)g[u].size(); ++i) {
        edge &e = g[u][i];
        if (dis[e.to] == dis[u] + 1 && e.flow != e.cap) {
            int df = dfs(e.to, min(e.cap - e.flow, cap));
            if (df) {
                e.flow += df;
                g[e.to][e.rev].flow -= df;
                return df;
            }
        }
    }
    dis[u] = -1;
    return 0;
}
int maxflow(int _s, int _t) {
    s = _s, t = _t;
    int flow = 0, df;
    while (bfs()) {
        fill(ind, ind+n+1, 0);
        while ((df = dfs(s, inf))) flow += df;
    }
    return flow;
}
bool feasible() {
    int sum = 0;
    for (int i = 1; i <= n - 2; ++i)
        if (cnt[i] > 0)
            add_edge(n - 1, i, cnt[i]), sum += cnt[i];
        else if (cnt[i] < 0) add_edge(i, n, -cnt[i]);
    if (sum != maxflow(n - 1, n)) sum = -1;
    for (int i = 1; i <= n - 2; ++i)
        if (cnt[i] > 0)
            g[n - 1].pop_back(), g[i].pop_back();
        else if (cnt[i] < 0)
            g[i].pop_back(), g[n].pop_back();
    return sum != -1;
}
int boundedflow(int _s, int _t) {
    add_edge(_t, _s, inf);
    if (!feasible()) return -1; // infeasible flow
    int x = g[_t].back().flow;
    g[_t].pop_back(), g[_s].pop_back();
}

```

```

    int y = maxflow(_t, _s);
    return x-y;
}
};

```

### 3.2 Dinic

```

struct MaxFlow { // 1-base
    struct edge {
        int to, cap, flow, rev;
    };
    vector<edge> g[maxn];
    int s, t, dis[maxn], ind[maxn], n;

    void init(int _n) {
        n = _n + 2;
        s = _n + 1, t = _n + 2;
        for (int i = 0; i <= n; ++i) g[i].clear();
    }
    void reset() {
        for (int i = 0; i <= n; ++i)
            for (auto &j : g[i]) j.flow = 0;
    }
    void add_edge(int u, int v, int cap) {
        g[u].pb(edge{v, cap, 0, (int)g[v].size()});
        g[v].pb(edge{u, 0, 0, (int)g[u].size() - 1});
        //change g[v] to cap for undirected graphs
    }
    bool bfs() {
        fill(dis, dis+n+1, -1);
        queue<int> q;
        q.push(s), dis[s] = 0;
        while (!q.empty()) {
            int cur = q.front(); q.pop();
            for (auto &e : g[cur]) {
                if (dis[e.to] == -1 && e.flow != e.cap) {
                    q.push(e.to);
                    dis[e.to] = dis[cur] + 1;
                }
            }
        }
        return dis[t] != -1;
    }
    int dfs(int u, int cap) {
        if (u == t || !cap) return cap;
        for (int &i = ind[u]; i < (int)g[u].size(); ++i) {
            edge &e = g[u][i];
            if (dis[e.to] == dis[u] + 1 && e.flow != e.cap) {
                int df = dfs(e.to, min(e.cap - e.flow, cap));
                if (df) {
                    e.flow += df;
                    g[e.to][e.rev].flow -= df;
                    return df;
                }
            }
        }
        dis[u] = -1;
        return 0;
    }
    int maxflow() {
        int flow = 0, df;
        while (bfs()) {
            fill(ind, ind+n+1, 0);
            while ((df = dfs(s, inf))) flow += df;
        }
        return flow;
    }
} flow;

```

### 3.3 Gomory Hu

```

MaxFlow Dinic;
int g[MAXN];
void GomoryHu(int n) { // 0-base

```

```

fill_n(g, n, 0);
for (int i = 1; i < n; ++i) {
    Dinic.reset();
    add_edge(i, g[i], Dinic.maxflow(i, g[i]));
    for (int j = i + 1; j <= n; ++j)
        if (g[j] == g[i] && ~Dinic.dis[j])
            g[j] = i;
}
}

```

### 3.4 Hungarian Algorithm

```

int c[maxn][maxn]; //hungarian algorithm in  $O(n^3)$ 
//1 base
int lx[maxn], ly[maxn], mx[maxn], my[maxn];
bool vx[maxn], vy[maxn];
int slack[maxn];
int tot;
bool dfs(int n, bool ch) {
    if (vx[n]) return false;
    vx[n] = 1;
    for (int v = 1; v <= tot; v++) {
        slack[v] = min(slack[v], lx[n] + ly[v] - c[n][v]);
        if (lx[n] + ly[v] - c[n][v] > 0) continue;
        vy[v] = 1;
        if (!my[v] || dfs(my[v], ch)) {
            if (ch) mx[n] = v, my[v] = n;
            return true;
        }
    }
    return false;
}
int main() {
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) vx[j] = vy[j] = 0;
        for (int j = 1; j <= n; j++) slack[j] = 1<<30;
        if (dfs(i, 1)) continue;
        bool aug = 0;
        while (!aug) {
            for (int j = 1; j <= n; j++) {
                if (!vy[j] && slack[j] == 0) {
                    vy[j] = 1;
                    if (dfs(my[j], 0)) {
                        aug = 1;
                        break;
                    }
                }
            }
            if (aug) break;
            int delta = 1<<30;
            for (int j = 1; j <= n; j++) {
                if (!vy[j]) delta = min(delta, slack[j]);
            }
            for (int j = 1; j <= n; j++) {
                if (vx[j]) lx[j] -= delta;
                if (vy[j]) ly[j] += delta;
                else {
                    slack[j] -= delta;
                    if (slack[j] == 0 && !my[j]) aug = 1;
                }
            }
        }
        for (int j = 1; j <= n; j++) vx[j] = vy[j] = 0;
        dfs(i, 1);
    }
}

```

### 3.5 ISAP Algorithm

```

struct Maxflow { //to be modified
    static const int MAXV = 20010;
    static const int INF = 1000000;
    struct Edge {
        int v, c, r;
        Edge(int _v, int _c, int _r)

```

```

        : v(_v), c(_c), r(_r) {}
    };
    int s, t;
    vector<Edge> G[MAXV * 2];
    int iter[MAXV * 2], d[MAXV * 2], gap[MAXV * 2], tot;
    void init(int x) {
        tot = x + 2;
        s = x + 1, t = x + 2;
        for (int i = 0; i <= tot; i++) {
            G[i].clear();
            iter[i] = d[i] = gap[i] = 0;
        }
    }
    void addEdge(int u, int v, int c) {
        G[u].push_back(Edge(v, c, SZ(G[v])));
        G[v].push_back(Edge(u, 0, SZ(G[u]) - 1));
    }
    int dfs(int p, int flow) {
        if (p == t) return flow;
        for (int &i = iter[p]; i < SZ(G[p]); i++) {
            Edge &e = G[p][i];
            if (e.c > 0 && d[p] == d[e.v] + 1) {
                int f = dfs(e.v, min(flow, e.c));
                if (f) {
                    e.c -= f;
                    G[e.v][e.r].c += f;
                    return f;
                }
            }
        }
        if ((--gap[d[p]]) == 0) d[s] = tot;
        else {
            d[p]++;
            iter[p] = 0;
            ++gap[d[p]];
        }
        return 0;
    }
    int solve() {
        int res = 0;
        gap[0] = tot;
        for (res = 0; d[s] < tot; res += dfs(s, INF))
            ;
        return res;
    }
} flow;

```

### 3.6 KM Algorithm

```

int n, m; //1-base, max matching
int mx[maxn], my[maxn];
bool adj[maxn][maxn], vis[maxn];
bool dfs(int n) {
    if (vis[n]) return false;
    vis[n] = 1;
    for (int v = 1; v <= n; v++) {
        if (!adj[n][v]) continue;
        if (!my[v] || (my[v] && dfs(my[v]))) {
            mx[n] = v, my[v] = n;
            return true;
        }
    }
    return false;
}
//min vertex cover: take unmatched vertex in L and find
//alternating tree,
//ans is not reached in L + reached in R

```

### 3.7 Max Simple Graph Matching

```

struct GenMatch { // 1-base
    int V, pr[N];
    bool el[N][N], inq[N], inp[N], inb[N];
    int st, ed, nb, bk[N], djs[N], ans;
    void init(int _V) {

```

```

V = _V;
for (int i = 0; i <= V; ++i) {
    for (int j = 0; j <= V; ++j) el[i][j] = 0;
    pr[i] = bk[i] = djs[i] = 0;
    inq[i] = inp[i] = inb[i] = 0;
}
}

void add_edge(int u, int v) {
    el[u][v] = el[v][u] = 1;
}

int lca(int u, int v) {
    fill_n(inp, V + 1, 0);
    while (1)
        if (u = djs[u], inp[u] = true, u == st) break;
        else u = bk[pr[u]];
    while (1)
        if (v = djs[v], inp[v]) return v;
        else v = bk[pr[v]];
    return v;
}

void upd(int u) {
    for (int v; djs[u] != nb;) {
        v = pr[u], inb[djs[u]] = inb[djs[v]] = true;
        u = bk[v];
        if (djs[u] != nb) bk[u] = v;
    }
}

void blo(int u, int v, queue<int> &qe) {
    nb = lca(u, v), fill_n(inb, V + 1, 0);
    upd(u), upd(v);
    if (djs[u] != nb) bk[u] = v;
    if (djs[v] != nb) bk[v] = u;
    for (int tu = 1; tu <= V; ++tu)
        if (inb[djs[tu]])
            if (djs[tu] = nb, !inq[tu])
                qe.push(tu), inq[tu] = 1;
}

void flow() {
    fill_n(inq + 1, V, 0), fill_n(bk + 1, V, 0);
    iota(djs + 1, djs + V + 1, 1);
    queue<int> qe;
    qe.push(st), inq[st] = 1, ed = 0;
    while (!qe.empty()) {
        int u = qe.front();
        qe.pop();
        for (int v = 1; v <= V; ++v)
            if (el[u][v] && djs[u] != djs[v] &&
                pr[u] != v) {
                if ((v == st) ||
                    (pr[v] > 0 && bk[pr[v]] > 0)) {
                    blo(u, v, qe);
                } else if (!bk[v]) {
                    if (bk[v] = u, pr[v] > 0) {
                        if (!inq[pr[v]]) qe.push(pr[v]);
                    } else {
                        return ed = v, void();
                    }
                }
            }
    }
}

void aug() {
    for (int u = ed, v, w; u > 0;)
        v = bk[u], w = pr[v], pr[v] = u, pr[u] = v,
        u = w;
}

int solve() {
    fill_n(pr, V + 1, 0), ans = 0;
    for (int u = 1; u <= V; ++u)
        if (!pr[u])
            if (st = u, flow(), ed > 0) aug(), ++ans;
    return ans;
}
};

```

### 3.8 MCMF

```

struct MCMF { // 0-base
    struct edge {
        ll from, to, cap, flow, cost, rev;
    } * past[maxn];
    vector<edge> G[maxn];
    bitset<maxn> inq;
    ll dis[maxn], up[maxn], s, t, mx, n;
    bool BellmanFord(ll &flow, ll &cost) {
        fill(dis, dis + n, inf);
        queue<ll> q;
        q.push(s), inq.reset(), inq[s] = 1;
        up[s] = mx - flow, past[s] = 0, dis[s] = 0;
        while (!q.empty()) {
            ll u = q.front();
            q.pop(), inq[u] = 0;
            if (!up[u]) continue;
            for (auto &e : G[u])
                if (e.flow != e.cap &&
                    dis[e.to] > dis[u] + e.cost) {
                    dis[e.to] = dis[u] + e.cost, past[e.to] = &e;
                    up[e.to] = min(up[u], e.cap - e.flow);
                    if (!inq[e.to]) inq[e.to] = 1, q.push(e.to);
                }
        }
        if (dis[t] == inf) return 0;
        flow += up[t], cost += up[t] * dis[t];
        for (ll i = t; past[i]; i = past[i]->from) {
            auto &e = *past[i];
            e.flow += up[t], G[e.to][e.rev].flow -= up[t];
        }
        return 1;
    }
    ll MinCostMaxFlow(ll _s, ll _t, ll &cost) {
        s = _s, t = _t, cost = 0;
        ll flow = 0;
        while (BellmanFord(flow, cost));
        return flow;
    }
    void init(ll _n, ll _mx) {
        n = _n, mx = _mx;
        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, cap, 0, cost, G[b].size()});
        G[b].pb(edge{b, a, 0, 0, -cost, G[a].size() - 1});
    }
};

```

### 3.9 Min Cost Circulation

```

//to be modified
struct Edge { int to, cap, rev, cost; };
vector<Edge> g[kN];
int dist[kN], pv[kN], ed[kN];
bool mark[kN];
int NegativeCycle(int n) {
    memset(mark, false, sizeof(mark));
    memset(dist, 0, sizeof(dist));
    int upd = -1;
    for (int i = 0; i <= n; ++i) {
        for (int j = 0; j < n; ++j) {
            int idx = 0;
            for (auto &e : g[j]) {
                if (e.cap > 0 && dist[e.to] > dist[j] + e.cost) {
                    dist[e.to] = dist[j] + e.cost;
                    pv[e.to] = j, ed[e.to] = idx;
                    if (i == n) {
                        upd = j;
                        while (!mark[upd]) mark[upd] = true, upd = pv[upd];
                        return upd;
                    }
                }
            }
            idx++;
        }
    }
}

```

```

    }
    return -1;
}
int Solve(int n) {
    int rt = -1, ans = 0;
    while ((rt = NegativeCycle(n)) >= 0) {
        memset(mark, false, sizeof(mark));
        vector<pair<int, int>> cyc;
        while (!mark[rt]) {
            cyc.emplace_back(pv[rt], ed[rt]);
            mark[rt] = true;
            rt = pv[rt];
        }
        reverse(cyc.begin(), cyc.end());
        int cap = kInf;
        for (auto &i : cyc) {
            auto &e = g[i.first][i.second];
            cap = min(cap, e.cap);
        }
        for (auto &i : cyc) {
            auto &e = g[i.first][i.second];
            e.cap -= cap;
            g[e.to][e.rev].cap += cap;
            ans += e.cost * cap;
        }
    }
    return ans;
}

```

### 3.10 SW Mincut

```

// stoer wagner algorithm: global min cut
const int maxn = 505;
struct SW { // O(V^3) θ-based
    int n, vis[maxn], del[maxn];
    int edge[maxn][maxn], wei[maxn];
    void init(int _n) {
        n = _n;
        fill(del, del+n, 0);
        for (int i = 0; i < n; i++) fill(edge[i], edge[i] + n, 0);
    }
    void addEdge(int u, int v, int w) {
        edge[u][v] += w, edge[v][u] += w;
    }
    void search(int &s, int &t) {
        fill(vis, vis+n, 0);
        fill(wei, wei+n, 0);
        s = t = -1;
        while (1) {
            int ma = -1, cur = 0;
            for (int i = 0; i < n; ++i)
                if (!del[i] && !vis[i] && ma < wei[i])
                    cur = i, ma = wei[i];
            if (mx == -1) break;
            vis[cur] = 1, s = t, t = cur;
            for (int i = 0; i < n; ++i)
                if (!vis[i] && !del[i]) wei[i] += edge[cur][i];
        }
    }
    int solve() {
        int ret = INF;
        for (int i = 0, x=0, y=0; i < n-1; ++i) {
            search(x, y), ret = min(res, wei[y]), del[y] = 1;
            for (int j = 0; j < n; ++j)
                edge[x][j] = (edge[j][x] += edge[y][j]);
        }
        return ret;
    }
};

```

## 4 Geometry

### 4.1 Geometry Template

```

using ld = ll;
using pdd = pair<ld, ld>;
using Line = pair<pdd, pdd>;
#define X first
#define Y second
// ld eps = 1e-7;

pdd operator+(pdd a, pdd b)
{ return {a.X + b.X, a.Y + b.Y}; }
pdd operator-(pdd a, pdd b)
{ return {a.X - b.X, a.Y - b.Y}; }
pdd operator*(ld i, pdd v)
{ return {i * v.X, i * v.Y}; }
pdd operator*(pdd v, ld i)
{ return {i * v.X, i * v.Y}; }
pdd operator/(pdd v, ld i)
{ return {v.X / i, v.Y / i}; }
ld dot(pdd a, pdd b)
{ return a.X * b.X + a.Y * b.Y; }
ld cross(pdd a, pdd b)
{ return a.X * b.Y - a.Y * b.X; }
ld abs2(pdd v)
{ return v.X * v.X + v.Y * v.Y; }
ld abs(pdd v)
{ return sqrt(abs2(v)); }
int sgn(ld v)
{ return v > 0 ? 1 : (v < 0 ? -1 : 0); }
// int sgn(NumType v){ return v > eps ? 1 : (v < -eps ? -1 : 0); }

int ori(pdd a, pdd b, pdd c)
{ return sgn(cross(b - a, c - a)); }
bool collinearity(pdd a, pdd b, pdd c)
{ return ori(a, b, c) == 0; }
bool btw(pdd p, pdd a, pdd b)
{ return collinearity(p, a, b) && sgn(dot(a - p, b - p)) <= 0; }

bool seg_intersect(Line a, Line b){
    pdd p1, p2, p3, p4;
    tie(p1, p2) = a; tie(p3, p4) = b;
    if(btw(p1, p3, p4) || btw(p2, p3, p4) || btw(p3, p1, p2)
        || btw(p4, p1, p2))
        return true;
    return ori(p1, p2, p3) * ori(p1, p2, p4) < 0 &&
        ori(p3, p4, p1) * ori(p3, p4, p2) < 0;
}

pdd intersect(Line a, Line b){
    pdd p1, p2, p3, p4;
    tie(p1, p2) = a; tie(p3, p4) = b;
    ld a123 = cross(p2 - p1, p3 - p1);
    ld a124 = cross(p2 - p1, p4 - p1);
    return (p4 * a123 - p3 * a124) / (a123 - a124);
}

4.2 Convex Hull

vector<int> getConvexHull(vector<pdd>& pts){
    vector<int> id(SZ(pts));
    iota(iter(id), 0);
    sort(iter(id), [&](int x, int y){ return pts[x] < pts[y];
    });
    vector<int> hull;
    for(int tt = 0; tt < 2; tt++){
        int sz = SZ(hull);
        for(int j : id){
            pdd p = pts[j];
            while(SZ(hull) - sz >= 2 &&
                cross(pts[hull.back()] - pts[hull[SZ(hull) - 2]],
                    p - pts[hull[SZ(hull) - 2]]) <= 0)
                hull.pop_back();
            hull.pb(j);
        }
        hull.pop_back();
        reverse(iter(id));
    }
    return hull;
}

```

```
}

```

### 4.3 Minimum Enclosing Circle

```
using ld = long double;
pair<pdd, ld> circumcenter(pdd a, pdd b, pdd c);
pair<pdd, ld> MinimumEnclosingCircle(vector<pdd> &pts){
    random_shuffle(iter(pts));
    pdd c = pts[0];
    ld r = 0;
    for(int i = 1; i < SZ(pts); i++){
        if(abs(pts[i] - c) <= r) continue;
        c = pts[i]; r = 0;
        for(int j = 0; j < i; j++){
            if(abs(pts[j] - c) <= r) continue;
            c = (pts[i] + pts[j]) / 2;
            r = abs(pts[i] - c);
            for(int k = 0; k < j; k++){
                if(abs(pts[k] - c) > r)
                    tie(c, r) = circumcenter(pts[i], pts[j], pts[k]);
            }
        }
    }
    return {c, r};
}
```

### 4.4 Minkowski Sum

```
void reorder_poly(vector<pdd>& pts){
    int mn = 0;
    for(int i = 1; i < (int)pts.size(); i++){
        if(pts[i].Y < pts[mn].Y || (pts[i].Y == pts[mn].Y
            && pts[i].X < pts[mn].X))
            mn = i;
    }
    rotate(pts.begin(), pts.begin() + mn, pts.end());
}

vector<pdd> minkowski(vector<pdd> P, vector<pdd> Q){
    reorder_poly(P);
    reorder_poly(Q);
    int psz = P.size();
    int qsz = Q.size();
    P.eb(P[0]);
    P.eb(P[1]);
    Q.eb(Q[0]);
    Q.eb(Q[1]);
    vector<pdd> ans;
    int i = 0, j = 0;
    while(i < psz || j < qsz){
        ans.eb(P[i] + Q[j]);
        int t = sgn(cross(P[i + 1] - P[i], Q[j + 1] - Q[j]));
        if(t >= 0) i++;
        if(t <= 0) j++;
    }
    return ans;
}
```

### 4.5 Half Plane Intersection

```
// copy from 8BQube
bool isin( Line l0, Line l1, Line l2 ) {
    // Check inter(l1, l2) in l0
    pdd p = intersect(l1.X, l1.Y, l2.X, l2.Y);
    return sign(cross(l0.Y - l0.X, p - l0.X)) > 0;
}
/* Having solution, check intersect(ret[0], ret[1])
* in all the lines. (use (L.Y - L.X) ^ (p - L.X) > 0
*/
/* --^-- Line.X --^-- Line.Y --^-- */
vector<Line> halfPlaneInter(vector<Line> lines) {
    vector<double> ata(SZ(lines)), ord(SZ(lines));
    for(int i = 0; i < SZ(lines); ++i) {
```

```
        ord[i] = i;
        pdd d = lines[i].Y - lines[i].X;
        ata[i] = atan2(d.Y, d.X);
    }
    sort(ALL(ord), [&](int i, int j) {
        if (fabs(ata[i] - ata[j]) >= eps)
            return ata[i] < ata[j];
        return ori(lines[i].X, lines[i].Y, lines[j].Y) < 0;
    });
    vector<Line> fin(1, lines[ord[0]]);
    for (int i = 1; i < SZ(lines); ++i)
        if (fabs(ata[ord[i]] - ata[ord[i - 1]]) > eps)
            fin.pb(lines[ord[i]]);
    deque<Line> dq;
    for (int i = 0; i < SZ(fin); ++i) {
        while (SZ(dq) >= 2 && !isin(fin[i], dq[SZ(dq) - 2], dq.
            back()))
            dq.pop_back();
        while (SZ(dq) >= 2 && !isin(fin[i], dq[0], dq[1]))
            dq.pop_front();
        dq.pb(fin[i]);
    }
    while (SZ(dq) >= 3 && !isin(dq[0], dq[SZ(dq) - 2], dq.
        back()))
        dq.pop_back();
    while (SZ(dq) >= 3 && !isin(dq.back(), dq[0], dq[1]))
        dq.pop_front();
    return vector<Line>(ALL(dq));
}
```

### 4.6 Dynamic Convex Hull

```
struct Line{
    ll a, b, l = MIN, r = MAX;
    Line(ll a, ll b): a(a), b(b) {}
    ll operator()(ll x) const{
        return a * x + b;
    }
    bool operator<(Line b) const{
        return a < b.a;
    }
    bool operator<(ll b) const{
        return r < b;
    }
};

ll iceil(ll a, ll b){
    if(b < 0) a *= -1, b *= -1;
    if(a > 0) return (a + b - 1) / b;
    else return a / b;
}

ll intersect(Line a, Line b){
    return iceil(a.b - b.b, b.a - a.a);
}

struct DynamicConvexHull{
    multiset<Line, less<>> ch;

    void add(Line ln){
        auto it = ch.lower_bound(ln);
        while(it != ch.end()){
            Line tl = *it;
            if(tl(tl.r) <= ln(tl.r)){
                it = ch.erase(it);
            }
            else break;
        }
        auto it2 = ch.lower_bound(ln);
        while(it2 != ch.begin()){
            Line tl = *prev(it2);
            if(tl(tl.l) <= ln(tl.l)){
                it2 = ch.erase(prev(it2));
            }
            else break;
        }
    }
}
```



```

it = ch.lower_bound(ln);
if(it != ch.end()){
    Line tl = *it;
    if(tl(tl.l) >= ln(tl.l)) ln.r = tl.l - 1;
    else{
        ll pos = intersect(ln, tl);
        tl.l = pos;
        ln.r = pos - 1;
        ch.erase(it);
        ch.insert(tl);
    }
}
it2 = ch.lower_bound(ln);
if(it2 != ch.begin()){
    Line tl = *prev(it2);
    if(tl(tl.r) >= ln(tl.r)) ln.l = tl.r + 1;
    else{
        ll pos = intersect(tl, ln);
        tl.r = pos - 1;
        ln.l = pos;
        ch.erase(prev(it2));
        ch.insert(tl);
    }
}
if(ln.l <= ln.r) ch.insert(ln);
}

ll query(ll pos){
    auto it = ch.lower_bound(pos);
    if(it == ch.end()) return 0;
    return (*it)(pos);
}
};

```

## 4.7 3D Point

```

// Copy from 8BQube
struct Point {
    double x, y, z;
    Point(double _x = 0, double _y = 0, double _z = 0): x(_x), y(_y), z(_z){}
    Point(pdd p) { x = p.X, y = p.Y, z = abs2(p); }
};

Point operator-(const Point &p1, const Point &p2)
{ return Point(p1.x - p2.x, p1.y - p2.y, p1.z - p2.z); }
Point cross(const Point &p1, const Point &p2)
{ return Point(p1.y * p2.z - p1.z * p2.y, p1.z * p2.x - p1.x * p2.z, p1.x * p2.y - p1.y * p2.x); }
double dot(const Point &p1, const Point &p2)
{ return p1.x * p2.x + p1.y * p2.y + p1.z * p2.z; }
double abs(const Point &a)
{ return sqrt(dot(a, a)); }
Point cross3(const Point &a, const Point &b, const Point &c)
{ return cross(b - a, c - a); }
double area(Point a, Point b, Point c)
{ return abs(cross3(a, b, c)); }
double volume(Point a, Point b, Point c, Point d)
{ return dot(cross3(a, b, c), d - a); }
pdd proj(Point a, Point b, Point c, Point u) {
    // proj. u to the plane of a, b, and c
    Point e1 = b - a;
    Point e2 = c - a;
    e1 = e1 / abs(e1);
    e2 = e2 - e1 * dot(e2, e1);
    e2 = e2 / abs(e2);
    Point p = u - a;
    return pdd(dot(p, e1), dot(p, e2));
}

```

## 4.8 ConvexHull3D

```

// Copy from 8BQube
struct CH3D {
    struct face{int a, b, c; bool ok;} F[8 * N];

```

```

double dblcmp(Point &p, face &f)
{ return dot(cross3(P[f.a], P[f.b], P[f.c]), p - P[f.a]); }
int g[N][N], num, n;
Point P[N];
void deal(int p, int a, int b) {
    int f = g[a][b];
    face add;
    if (F[f].ok) {
        if (dblcmp(P[p], F[f]) > eps) dfs(p, f);
    } else
        add.a = b, add.b = a, add.c = p, add.ok = 1, g[p][b] = g[a][p] = g[b][a] = num, F[num++] = add;
}
void dfs(int p, int now) {
    F[now].ok = 0;
    deal(p, F[now].b, F[now].a), deal(p, F[now].c, F[now].b), deal(p, F[now].a, F[now].c);
}
bool same(int s, int t){
    Point &a = P[F[s].a];
    Point &b = P[F[s].b];
    Point &c = P[F[s].c];
    return fabs(volume(a, b, c, P[F[t].a])) < eps && fabs(volume(a, b, c, P[F[t].b])) < eps && fabs(volume(a, b, c, P[F[t].c])) < eps;
}
void init(int _n){n = _n, num = 0;}
void solve() {
    face add;
    num = 0;
    if(n < 4) return;
    if([&](){
        for (int i = 1; i < n; ++i)
            if (abs(P[0] - P[i]) > eps)
                return swap(P[1], P[i]), 0;
        return 1;
    }) || [&](){
        for (int i = 2; i < n; ++i)
            if (abs(cross3(P[i], P[0], P[1])) > eps)
                return swap(P[2], P[i]), 0;
        return 1;
    }) || [&](){
        for (int i = 3; i < n; ++i)
            if (fabs(dot(cross3(P[0] - P[1], P[1] - P[2]), P[0] - P[i])) > eps)
                return swap(P[3], P[i]), 0;
        return 1;
    }) return;
    for (int i = 0; i < 4; ++i) {
        add.a = (i + 1) % 4, add.b = (i + 2) % 4, add.c = (i + 3) % 4, add.ok = true;
        if (dblcmp(P[i], add) > 0) swap(add.b, add.c);
        g[add.a][add.b] = g[add.b][add.c] = g[add.c][add.a] = num;
        F[num++] = add;
    }
    for (int i = 4; i < n; ++i)
        for (int j = 0; j < num; ++j)
            if (F[j].ok && dblcmp(P[i], F[j]) > eps) {
                dfs(i, j);
                break;
            }
    for (int tmp = num, i = (num = 0); i < tmp; ++i)
        if (F[i].ok) F[num++] = F[i];
}
double get_area() {
    double res = 0.0;
    if (n == 3)
        return abs(cross3(P[0], P[1], P[2])) / 2.0;
    for (int i = 0; i < num; ++i)
        res += area(P[F[i].a], P[F[i].b], P[F[i].c]);
    return res / 2.0;
}
double get_volume() {
    double res = 0.0;
    for (int i = 0; i < num; ++i)

```

```

    res += volume(Point(0, 0, 0), P[F[i].a], P[F[i].b], P
        [F[i].c]);
    return fabs(res / 6.0);
}
int triangle() {return num;}
int polygon() {
    int res = 0;
    for (int i = 0, flag = 1; i < num; ++i, res += flag,
        flag = 1)
        for (int j = 0; j < i && flag; ++j)
            flag &= !same(i, j);
    return res;
}
Point getcent(){
    Point ans(0, 0, 0), temp = P[F[0].a];
    double v = 0.0, t2;
    for (int i = 0; i < num; ++i)
        if (F[i].ok == true) {
            Point p1 = P[F[i].a], p2 = P[F[i].b], p3 = P[F[i].c
                ];
            t2 = volume(temp, p1, p2, p3) / 6.0;
            if (t2 > 0)
                ans.x += (p1.x + p2.x + p3.x + temp.x) * t2, ans.
                    y += (p1.y + p2.y + p3.y + temp.y) * t2, ans.
                        z += (p1.z + p2.z + p3.z + temp.z) * t2, v +=
                            t2;
        }
    ans.x /= (4 * v), ans.y /= (4 * v), ans.z /= (4 * v);
    return ans;
}
double pointmindis(Point p) {
    double rt = 99999999;
    for(int i = 0; i < num; ++i)
        if(F[i].ok == true) {
            Point p1 = P[F[i].a], p2 = P[F[i].b], p3 = P[F[i].c
                ];
            double a = (p2.y - p1.y) * (p3.z - p1.z) - (p2.z -
                p1.z) * (p3.y - p1.y);
            double b = (p2.z - p1.z) * (p3.x - p1.x) - (p2.x -
                p1.x) * (p3.z - p1.z);
            double c = (p2.x - p1.x) * (p3.y - p1.y) - (p2.y -
                p1.y) * (p3.x - p1.x);
            double d = 0 - (a * p1.x + b * p1.y + c * p1.z);
            double temp = fabs(a * p.x + b * p.y + c * p.z + d)
                / sqrt(a * a + b * b + c * c);
            rt = min(rt, temp);
        }
    return rt;
}
};

```

## 4.9 Circle Operations

```

// from 8BQube
const double PI=acos(-1);
vector<pdd> circleLineIntersection(pdd c, double r, pdd a,
    pdd b) {
    pdd p = a + (b - a) * dot(c - a, b - a) / abs2(b - a);
    double s = cross(b - a, c - a), h2 = r * r - s * s / abs2
        (b - a);
    if (h2 < 0) return {};
    if (h2 == 0) return {p};
    pdd h = (b - a) / abs(b - a) * sqrt(h2);
    return {p - h, p + h};
}
double _area(pdd pa, pdd 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 = dot(pb,pb-pa) / a / c, B = acos(cosB);
    double cosC = dot(pa,pb) / a / b, C = acos(cosC);
    if(a > r){
        S = (C/2)*r*r;
        h = a*b*sin(C)/c;
    }

```

```

    if (h < r && B < PI/2) S -= (acos(h/r)*r*r - h*sqrt(r*r
        -h*h));
    }
    else if(b > r){
        theta = PI - B - asin(sin(B)/r*a);
        S = .5*a*r*sin(theta) + (C-theta)/2*r*r;
    }
    else S = .5*sin(C)*a*b;
    return S;
}
double areaPolyCircle(const vector<pdd> poly,const pdd &O,
    const double r){
    double S=0;
    for(int i=0;i<SZ(poly);++i)
        S+=_area(poly[i]-O,poly[(i+1)%SZ(poly)]-O,r)*ori(0,poly
            [i],poly[(i+1)%SZ(poly)]);
    return fabs(S);
}
bool CCinter(Cir &a, Cir &b, pdd &p1, pdd &p2) {
    pdd o1 = a.O, o2 = b.O;
    double r1 = a.R, r2 = b.R, d2 = abs2(o1 - o2), d = sqrt(
        d2);
    if(d < max(r1, r2) - min(r1, r2) || d > r1 + r2) return
        0;
    pdd u = (o1 + o2) * 0.5 + (o1 - o2) * ((r2 * r2 - r1 * r1
        ) / (2 * d2));
    double A = sqrt((r1 + r2 + d) * (r1 - r2 + d) * (r1 + r2
        - d) * (-r1 + r2 + d));
    pdd v = pdd(o1.Y - o2.Y, -o1.X + o2.X) * A / (2 * d2);
    p1 = u + v, p2 = u - v;
    return 1;
}
vector<Line> CCTang( const Cir& c1 , const Cir& c2 , int
    sign1 ){
    vector<Line> ret;
    double d_sq = abs2( c1.O - c2.O );
    if (sign(d_sq) == 0) return ret;
    double d = sqrt(d_sq);
    pdd v = (c2.O - c1.O) / d;
    double c = (c1.R - sign1 * c2.R) / d;
    if (c * c > 1) return ret;
    double h = sqrt(max( 0.0, 1.0 - c * c ));
    for (int sign2 = 1; sign2 >= -1; sign2 -= 2) {
        pdd n = pdd(v.X * c - sign2 * h * v.Y,
            v.Y * c + sign2 * h * v.X);
        pdd p1 = c1.O + n * c1.R;
        pdd p2 = c2.O + n * (c2.R * sign1);
        if (sign(p1.X - p2.X) <= 0 and
            sign(p1.Y - p2.Y) <= 0)
            p2 = p1 + perp(c2.O - c1.O);
        ret.pb(Line(p1, p2));
    }
    return ret;
}

```

## 5 Graph

### 5.1 Block Cut Tree

```

struct BlockCutTree{
    vector<vector<int>> tree; // 1-based
    vector<int> node;
    vector<int> type; // 0:square, 1:circle

    bool iscut(int v){
        return type[node[v]] == 1;
    }

    vector<int> getbcc(int v){
        if(!iscut(v)) return {node[v]};
        vector<int> ans;
        for(int i : tree[node[v]])
            ans.pb(i);
        return ans;
    }
}

```

```

void build(int n, vector<vector<int>>& g){
    tree.resize(2 * n + 1);
    type.resize(2 * n + 1);
    node.resize(n + 1, -1);
    vector<int> in(n + 1);
    vector<int> low(n + 1);
    stack<int> st;

    int ts = 1;
    int bcc = 1;
    auto addv = [&](int id, int v){
        if(node[v] == -1){
            node[v] = id;
            return;
        }
        if(type[node[v]] == 0){
            int o = node[v];
            node[v] = bcc++;
            type[node[v]] = 1;
            tree[o].pb(node[v]);
            tree[node[v]].pb(o);
        }
        tree[id].pb(node[v]);
        tree[node[v]].pb(id);
    };
    function<void(int, int)> dfs = [&](int now, int p){
        in[now] = low[now] = ts++;
        st.push(now);
        int child = 0;
        for(int i : g[now]){
            if(i == p) continue;
            if(in[i]){
                low[now] = min(low[now], in[i]);
                continue;
            }
            child++;
            dfs(i, now);
            low[now] = min(low[now], low[i]);

            if(low[i] >= in[now]){
                int nowid = bcc++;
                while(true){
                    int x = st.top();
                    st.pop();
                    addv(nowid, x);
                    if(x == i) break;
                }
                addv(nowid, now);
            }
        }
        if(child == 0 && now == p) addv(bcc++, now);
    };
    dfs(1, 1);
};

```

## 5.2 2-SAT

```

struct SAT{ // 1-based
    int n;
    vector<vector<int>> g, rg;
    bool ok = true;
    vector<bool> ans;

    void init(int _n){
        n = _n;
        g.resize(2 * n + 1);
        rg.resize(2 * n + 1);
        ans.resize(n + 1);
    }
    int neg(int v){
        return v <= n ? v + n : v - n;
    }
    void addEdge(int u, int v){
        g[u].eb(v);

```

```

        rg[v].eb(u);
    }
    void addClause(int a, int b){
        addEdge(a, b);
        addEdge(neg(b), neg(a));
    }
    void build(){
        vector<bool> vst(n + 1);
        vector<int> tmp, scc(n + 1, -1);
        int cnt = 1;
        function<void(int)> dfs = [&](int now){
            vst[now] = true;
            for(int i : rg[now]){
                if(vst[i]) continue;
                dfs(i);
            }
            tmp.pb(now);
        };
        for(int i = 1; i <= 2 * n; i++){
            if(!vst[i]) dfs(i);
        }
        reverse(iter(tmp));
        function<void(int, int)> dfs2 = [&](int now, int id){
            scc[now] = id;
            for(int i : g[now]){
                if(scc[i] != -1) continue;
                dfs2(i, id);
            }
        };
        for(int i : tmp){
            if(scc[i] == -1) dfs2(i, cnt++);
        }
        for(int i = 1; i <= n; i++){
            if(scc[i] == scc[neg(i)]){
                ok = false;
                return;
            }
            if(scc[i] < scc[neg(i)]) ans[i] = true;
            else ans[i] = false;
        }
    }
};

```

## 5.3 Dominator Tree

```

// copy from 8BQube
struct dominator_tree { // 1-base
    vector<int> G[N], rG[N];
    int n, pa[N], dfn[N], id[N], Time;
    int semi[N], idom[N], best[N];
    vector<int> tree[N]; // dominator_tree
    void init(int _n) {
        n = _n;
        for (int i = 1; i <= n; ++i)
            G[i].clear(), rG[i].clear();
    }
    void add_edge(int u, int v) {
        G[u].pb(v), rG[v].pb(u);
    }
    void dfs(int u) {
        id[dfn[u] = ++Time] = u;
        for (auto v : G[u])
            if (!dfn[v]) dfs(v), pa[dfn[v]] = dfn[u];
    }
    int find(int y, int x) {
        if (y <= x) return y;
        int tmp = find(pa[y], x);
        if (semi[best[y]] > semi[best[pa[y]]])
            best[y] = best[pa[y]];
        return pa[y] = tmp;
    }
    void tarjan(int root) {
        Time = 0;
        for (int i = 1; i <= n; ++i) {
            dfn[i] = idom[i] = 0;

```

```

    tree[i].clear();
    best[i] = semi[i] = i;
}
dfs(root);
for (int i = Time; i > 1; --i) {
    int u = id[i];
    for (auto v : rG[u])
        if (v = dfn[v]) {
            find(v, i);
            semi[i] = min(semi[i], semi[best[v]]);
        }
    tree[semi[i]].pb(i);
    for (auto v : tree[pa[i]]) {
        find(v, pa[i]);
        idom[v] =
            semi[best[v]] == pa[i] ? pa[i] : best[v];
    }
    tree[pa[i]].clear();
}
for (int i = 2; i <= Time; ++i) {
    if (idom[i] != semi[i]) idom[i] = idom[idom[i]];
    tree[id[idom[i]]].pb(id[i]);
}
}
};

```

## 5.4 Virtual Tree

```

// copy from 8BQube
vector<int> vG[N];
int top, st[N];

void insert(int u) {
    if (top == -1) return st[++top] = u, void();
    int p = LCA(st[top], u);
    if (p == st[top]) return st[++top] = u, void();
    while (top >= 1 && dep[st[top - 1]] >= dep[p])
        vG[st[top - 1]].pb(st[top]), --top;
    if (st[top] != p)
        vG[p].pb(st[top]), --top, st[++top] = p;
    st[++top] = u;
}

void reset(int u) {
    for (int i : vG[u]) reset(i);
    vG[u].clear();
}

void solve(vector<int> &v) {
    top = -1;
    sort(ALL(v),
        [&](int a, int b) { return dfn[a] < dfn[b]; });
    for (int i : v) insert(i);
    while (top > 0) vG[st[top - 1]].pb(st[top]), --top;
    // do something
    reset(v[0]);
}

```

## 6 Math

### 6.1 Extended Euclidean Algorithm

```

// ax+ny = 1, ax+ny == ax == 1 (mod n)
void extgcd(ll x, ll y, ll &g, ll &a, ll &b) {
    if (y == 0) g=x, a=1, b=0;
    else extgcd(y, x%y, g, b, a), b-=(x/y)*a;
}

```

### 6.2 Floor & Ceil

```

int floor_div(int a, int b){
    return a/b-(a%b&&a<0^b<0);
}

```

```

}
int ceil_div(int a, int b){
    return a/b+(a%b&&a<0^b>0);
}

```

### 6.3 Legendre

```

// the Jacobi symbol is a generalization of the Legendre
// symbol,
// such that the bottom doesn't need to be prime.
// (n/p) -> same as Legendre
// (n/ab) = (n/a)(n/b)
// work with long long
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;
}

```

```

// 0: a == 0
// -1: a isn't a quad res of p
// else: return X with X^2 % p == a
// doesn't work with long long
int QuadraticResidue(int a, int p) {
    if (p == 2) return a & 1;
    const int jc = Jacobi(a, p);
    if (jc == 0) return 0;
    if (jc == -1) return -1;
    int b, d;
    for (; ; ) {
        b = rand() % p;
        d = (1LL * b * b + p - a) % p;
        if (Jacobi(d, p) == -1) break;
    }
    int f0 = b, f1 = 1, g0 = 1, g1 = 0, tmp;
    for (int e = (1LL + p) >> 1; e; e >>= 1) {
        if (e & 1) {
            tmp = (1LL * g0 * f0 + 1LL * d * (1LL * g1 * f1 % p))
                % p;
            g1 = (1LL * g0 * f1 + 1LL * g1 * f0) % p;
            g0 = tmp;
        }
        tmp = (1LL * f0 * f0 + 1LL * d * (1LL * f1 * f1 % p)) %
            p;
        f1 = (2LL * f0 * f1) % p;
        f0 = tmp;
    }
    return g0;
}

```

### 6.4 Simplex

```

#pragma once

typedef double T; // Long double, Rational, double + mod<P>
// ...
typedef vector<T> vd;
typedef vector<vd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j], N[j]) < MP(X[s], N[s]))
    s=j

struct LPSolver {
    int m, n;
    vi N, B;
}

```

```
vvd D;
```

```
LPSolver(const vvd& A, const vd& b, const vd& c) :
    m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
    rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
    rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
    rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m+1][n] = 1;
}
```

```
void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
        T *b = D[i].data(), inv2 = b[s] * inv;
        rep(j,0,n+2) b[j] -= a[j] * inv2;
        b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
}
```

```
bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
        int s = -1;
        rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
        if (D[x][s] >= -eps) return true;
        int r = -1;
        rep(i,0,m) {
            if (D[i][s] <= eps) continue;
            if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                < MP(D[r][n+1] / D[r][s], B[r])) r = i;
        }
        if (r == -1) return false;
        pivot(r, s);
    }
}
```

```
T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
        pivot(r, n);
        if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
        rep(i,0,m) if (B[i] == -1) {
            int s = 0;
            rep(j,1,n+1) ltj(D[i]);
            pivot(i, s);
        }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
}
};
```

## 6.5 Floor Sum

```
// from 8BQube
ll floor_sum(ll n, ll m, ll a, ll b) {
    ll ans = 0;
    if (a >= m)
        ans += (n - 1) * n * (a / m) / 2, a %= m;
    if (b >= m)
        ans += n * (b / m), b %= m;
    ll y_max = (a * n + b) / m, x_max = (y_max * m - b);
    if (y_max == 0) return ans;
    ans += (n - (x_max + a - 1) / a) * y_max;
    ans += floor_sum(y_max, a, m, (a - x_max % a) % a);
    return ans;
}
// sum^{n-1}_0 floor((a * i + b) / m) in log(n + m + a + b)
```

## 6.6 Miller Rabin & Pollard Rho

```
// n < 4,759,123,141      3 : 2, 7, 61
// n < 1,122,004,669,633  4 : 2, 13, 23, 1662803
// n < 3,474,749,660,383  6 : pimes <= 13
// n < 2^64              7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
ll mul(ll a, ll b, ll n){
    return (__int128)a * b % n;
}

bool Miller_Rabin(ll a, ll n) {
    if ((a = a % n) == 0) return 1;
    if (n % 2 == 0) return n == 2;
    ll tmp = (n - 1) / ((n - 1) & (1 - n));
    ll t = __lg(((n - 1) & (1 - n))), x = 1;
    for (; tmp; tmp >>= 1, a = mul(a, a, n))
        if (tmp & 1) x = mul(x, a, n);
    if (x == 1 || x == n - 1) return 1;
    while (--t)
        if ((x = mul(x, x, n)) == n - 1) return 1;
    return 0;
}

bool prime(ll n){
    vector<ll> tmp = {2, 325, 9375, 28178, 450775, 9780504,
        1795265022};
    for(ll i : tmp)
        if(!Miller_Rabin(i, n)) return false;
    return true;
}

map<ll, int> cnt;
void PollardRho(ll n) {
    if (n == 1) return;
    if (prime(n)) return ++cnt[n], void();
    if (n % 2 == 0) return PollardRho(n / 2), ++cnt[2], void();
    ll x = 2, y = 2, d = 1, p = 1;
#define f(x, n, p) ((mul(x, x, n) + p) % n)
    while (true) {
        if (d != n && d != 1) {
            PollardRho(n / d);
            PollardRho(d);
            return;
        }
        if (d == n) ++p;
        x = f(x, n, p), y = f(f(y, n, p), n, p);
        d = gcd(abs(x - y), n);
    }
}
```

## 7 Misc

### 7.1 Fraction

```
struct Frac{
    ll p, q; // p / q
    Frac(ll _p, ll _q): p(_p), q(_q) { if(q < 0) p = -p, q = -q; }
};

Frac operator-(Frac a) { return Frac(-a.p, a.q); }
Frac operator+(Frac a, Frac b){
    ll q = a.q * b.q;
    ll p = a.p * b.q + b.p * a.q;
    return Frac(p, q);
}

Frac inv(Frac a){ return Frac(a.q, a.p); }
Frac operator-(Frac a, Frac b) { return a + (-b); }
Frac operator*(Frac a, Frac b) { return Frac(a.p * b.p, a.q * b.q); }
Frac operator/(Frac a, Frac b) { return a * inv(b); }
ostream& operator<<(ostream& o, Frac a) { return o << a.p << '/' << a.q; }
```

## 7.2 Matroid

我們稱一個二元組  $M = (E, \mathcal{I})$  為一個擬陣，其中  $\mathcal{I} \subseteq 2^E$  為  $E$  的子集所形成的非空集合，若：

- 若  $S \in \mathcal{I}$  以及  $S' \subsetneq S$ ，則  $S' \in \mathcal{I}$
- 對於  $S_1, S_2 \in \mathcal{I}$  滿足  $|S_1| < |S_2|$ ，存在  $e \in S_2 \setminus S_1$  使得  $S_1 \cup \{e\} \in \mathcal{I}$

除此之外，我們有以下的定義：

- 位於  $\mathcal{I}$  中的集合我們稱之為獨立集 (independent set)，反之不在  $\mathcal{I}$  中的我們稱為相依集 (dependent set)
- 極大的獨立集為基底 (base)、極小的相依集為迴路 (circuit)
- 一個集合  $Y$  的秩 (rank)  $r(Y)$  為該集合中最大的獨立子集，也就是  $r(Y) = \max\{|X| \mid X \subseteq Y \text{ 且 } X \in \mathcal{I}\}$

性質：

1.  $X \subseteq Y \wedge Y \in \mathcal{I} \implies X \in \mathcal{I}$
2.  $X \subseteq Y \wedge X \notin \mathcal{I} \implies Y \notin \mathcal{I}$
3. 若  $B$  與  $B'$  皆是基底且  $B \subseteq B'$ ，則  $B = B'$   
若  $C$  與  $C'$  皆是迴路且  $C \subseteq C'$ ，則  $C = C'$
4.  $e \in E \wedge X \subseteq E \implies r(X) \leq r(X \cup \{e\}) \leq r(X) + 1$  i.e. 加入一個元素後秩不會降底，最多增加 1
5.  $\forall Y \subseteq E, \exists X \subseteq Y, r(X) = |X| = r(Y)$

一些等價的性質：

1. 對於所有  $X \subseteq E$ ， $X$  的極大獨立子集都有相同的大小
2. 對於  $B_1, B_2 \in \mathcal{B} \wedge B_1 \neq B_2$ ，對於所有  $e_1 \in B_1 \setminus B_2$ ，存在  $e_2 \in B_2 \setminus B_1$  使得  $(B_1 \setminus \{e_1\}) \cup \{e_2\} \in \mathcal{B}$
3. 對於  $X, Y \in \mathcal{I}$  且  $|X| < |Y|$ ，存在  $e \in Y \setminus X$  使得  $X \cup \{e\} \in \mathcal{B}$
4. 如果  $r(X \cup \{e_1\}) = r(X \cup \{e_2\}) = r(X)$ ，則  $r(X \cup \{e_1, e_2\}) = r(X)$ 。  
如果  $r(X \cup \{e\}) = r(X)$  對於所有  $e \in E'$  都成立，則  $r(X \cup E') = r(X)$ 。

擬陣交

```
Data: 兩個擬陣  $M_1 = (E, \mathcal{I}_1)$  以及  $M_2 = (E, \mathcal{I}_2)$ 
Result:  $I$  為最大的位於  $\mathcal{I}_1 \cap \mathcal{I}_2$  中的獨立集
 $I \leftarrow \emptyset$ 
 $X_1 \leftarrow \{e \in E \setminus I \mid I \cup \{e\} \in \mathcal{I}_1\}$ 
 $X_2 \leftarrow \{e \in E \setminus I \mid I \cup \{e\} \in \mathcal{I}_2\}$ 
while  $X_1 \neq \emptyset$  且  $X_2 \neq \emptyset$  do
    if  $e \in X_1 \cap X_2$  then
         $I \leftarrow I \cup \{e\}$ 
    else
        構造交換圖  $\mathcal{D}_{M_1, M_2}(I)$ 
        在交換圖上找到一條  $X_1$  到  $X_2$  且沒有捷徑的路徑  $P$ 
         $I \leftarrow I \Delta P$ 
    end if
     $X_1 \leftarrow \{e \in E \setminus I \mid I \cup \{e\} \in \mathcal{I}_1\}$ 
     $X_2 \leftarrow \{e \in E \setminus I \mid I \cup \{e\} \in \mathcal{I}_2\}$ 
end while
```

## 8 Polynomial

### 8.1 FFT

```
using val_t = complex<double>;
template<int MAXN>
struct FFT {
    const double PI = acos(-1);
    val_t w[MAXN];
    FFT() {
        for (int i = 0; i < MAXN; ++i) {
            double arg = 2 * PI * i / MAXN;
            w[i] = val_t(cos(arg), sin(arg));
        }
    }
    void bitrev(vector<val_t> &a, int n) //same as NTT
    void trans(vector<val_t> &a, int n, bool inv = false) {
        bitrev(a, n);
        for (int L = 2; L <= n; L <= 1) {
            int dx = MAXN / L, dl = L >> 1;
            for (int i = 0; i < n; i += L) {
                for (int j = i, x = 0; j < i + dl; ++j, x += dx) {
                    val_t tmp = a[j + dl] * (inv ? conj(w[x]) : w[x])
                    ;
```

```
                    a[j + dl] = a[j] - tmp;
                    a[j] += tmp;
                }
            }
        }
        if (inv) {
            for (int i = 0; i < n; ++i) a[i] /= n;
        }
    }
    //multiplying two polynomials A * B:
    //fft.trans(A, siz, 0), fft.trans(B, siz, 0):
    //A[i] *= B[i], fft.trans(A, siz, 1);
};
```

### 8.2 NTT

```
//to be modified
//(2^16)+1, 65537, 3
//7*17*(2^23)+1, 998244353, 3
//1255*(2^20)+1, 1315962881, 3
//51*(2^25)+1, 1711276033, 29
template<int MAXN, ll P, ll RT> //MAXN must be 2^k
struct NTT {
    ll w[MAXN];
    ll mpow(ll a, ll n);
    ll minv(ll a) { return mpow(a, P - 2); }
    NTT() {
        ll dw = mpow(RT, (P - 1) / MAXN);
        w[0] = 1;
        for (int i = 1; i < MAXN; ++i) w[i] = w[i - 1] * dw % P
            ;
    }
    void bitrev(vector<ll> &a, int n) {
        int i = 0;
        for (int j = 1; j < n - 1; ++j) {
            for (int k = n >> 1; (i ^= k) < k; k >>= 1);
            if (j < i) swap(a[i], a[j]);
        }
    }
    void operator()(vector<ll> &a, int n, bool inv = false) {
        //0 <= a[i] < P
        bitrev(a, n);
        for (int L = 2; L <= n; L <= 1) {
            int dx = MAXN / L, dl = L >> 1;
            for (int i = 0; i < n; i += L) {
                for (int j = i, x = 0; j < i + dl; ++j, x += dx) {
                    ll tmp = a[j + dl] * w[x] % P;
                    if ((a[j + dl] = a[j] - tmp) < 0) a[j + dl] += P;
                    if ((a[j] += tmp) >= P) a[j] -= P;
                }
            }
        }
        if (inv) {
            reverse(a.begin()+1, a.begin()+n);
            ll invn = minv(n);
            for (int i = 0; i < n; ++i) a[i] = a[i] * invn % P;
        }
    }
};
```

### 8.3 Polynomial Operation

```
// Copy from 8BQube
#define fi(s, n) for (int i = (int)(s); i < (int)(n); ++i)
template<int MAXN, ll P, ll RT> // MAXN = 2^k
struct Poly : vector<ll> { // coefficients in [0, P)
    using vector<ll>::vector;
    static NTT<MAXN, P, RT> ntt;
    int n() const { return (int)size(); } // n() >= 1
    Poly(const Poly &p, int m) : vector<ll>(m) {
        copy_n(p.data(), min(p.n(), m), data());
    }
    Poly& irev() { return reverse(data(), data() + n()), *this; }
    Poly& isz(int m) { return resize(m), *this; }
```

```

Poly& iadd(const Poly &rhs) { // n() == rhs.n()
    fi(0, n()) if (((*this)[i] += rhs[i]) >= P) (*this)[i]
        -= P;
    return *this;
}
Poly& imul(ll k) {
    fi(0, n()) (*this)[i] = (*this)[i] * k % P;
    return *this;
}
Poly Mul(const Poly &rhs) const {
    int m = 1;
    while (m < n() + rhs.n() - 1) m <= 1;
    Poly X(*this, m), Y(rhs, m);
    ntt(X.data(), m), ntt(Y.data(), m);
    fi(0, m) X[i] = X[i] * Y[i] % P;
    ntt(X.data(), m, true);
    return X.isz(n() + rhs.n() - 1);
}
Poly Inv() const { // (*this)[0] != 0, 1e5/95ms
    if (n() == 1) return {ntt.minv((*this)[0])};
    int m = 1;
    while (m < n() * 2) m <= 1;
    Poly Xi = Poly(*this, (n() + 1) / 2).Inv().isz(m);
    Poly Y(*this, m);
    ntt(Xi.data(), m), ntt(Y.data(), m);
    fi(0, m) {
        Xi[i] *= (2 - Xi[i] * Y[i]) % P;
        if ((Xi[i] % P) < 0) Xi[i] += P;
    }
    ntt(Xi.data(), m, true);
    return Xi.isz(n());
}
Poly Sqrt() const { // Jacobi((*this)[0], P) = 1, 1e5/235
    ms
    if (n() == 1) return {QuadraticResidue((*this)[0], P)};
    Poly X = Poly(*this, (n() + 1) / 2).Sqrt().isz(n());
    return X.iadd(Mul(X.Inv()).isz(n())).imul(P / 2 + 1);
}
pair<Poly, Poly> DivMod(const Poly &rhs) const { // (rhs
    .)back() != 0
    if (n() < rhs.n()) return {{0}, *this};
    const int m = n() - rhs.n() + 1;
    Poly X(rhs); X.irev().isz(m);
    Poly Y(*this); Y.irev().isz(m);
    Poly Q = Y.Mul(X.Inv()).isz(m).irev();
    X = rhs.Mul(Q), Y = *this;
    fi(0, n()) if ((Y[i] - X[i]) < 0) Y[i] += P;
    return {Q, Y.isz(max(1, rhs.n() - 1))};
}
Poly Dx() const {
    Poly ret(n() - 1);
    fi(0, ret.n()) ret[i] = (i + 1) * (*this)[i + 1] % P;
    return ret.isz(max(1, ret.n()));
}
Poly Sx() const {
    Poly ret(n() + 1);
    fi(0, n()) ret[i + 1] = ntt.minv(i + 1) * (*this)[i] %
        P;
    return ret;
}
Poly _tmul(int nn, const Poly &rhs) const {
    Poly Y = Mul(rhs).isz(n() + nn - 1);
    return Poly(Y.data() + n() - 1, Y.data() + Y.n());
}
vector<ll> _eval(const vector<ll> &x, const vector<Poly>
    &up) const {
    const int m = (int)x.size();
    if (!m) return {};
    vector<Poly> down(m * 2);
    // down[1] = DivMod(up[1]).second;
    // fi(2, m * 2) down[i] = down[i / 2].DivMod(up[i]).
        second;
    down[1] = Poly(up[1]).irev().isz(n()).Inv().irev().
        _tmul(m, *this);
    fi(2, m * 2) down[i] = up[i ^ 1]._tmul(up[i].n() - 1,
        down[i / 2]);
    vector<ll> y(m);

```

```

    fi(0, m) y[i] = down[m + i][0];
    return y;
}
static vector<Poly> _tree1(const vector<ll> &x) {
    const int m = (int)x.size();
    vector<Poly> up(m * 2);
    fi(0, m) up[m + i] = {x[i] ? P - x[i] : 0, 1};
    for (int i = m - 1; i > 0; --i) up[i] = up[i * 2].Mul(
        up[i * 2 + 1]);
    return up;
}
vector<ll> Eval(const vector<ll> &x) const { // 1e5, 1s
    auto up = _tree1(x); return _eval(x, up);
}
static Poly Interpolate(const vector<ll> &x, const vector
    <ll> &y) { // 1e5, 1.4s
    const int m = (int)x.size();
    vector<Poly> up = _tree1(x), down(m * 2);
    vector<ll> z = up[1].Dx()._eval(x, up);
    fi(0, m) z[i] = y[i] * ntt.minv(z[i]) % P;
    fi(0, m) down[m + i] = {z[i]};
    for (int i = m - 1; i > 0; --i) down[i] = down[i * 2].
        Mul(up[i * 2 + 1]).iadd(down[i * 2 + 1].Mul(up[i *
            2]));
    return down[1];
}
Poly Ln() const { // (*this)[0] == 1, 1e5/170ms
    return Dx().Mul(Inv()).Sx().isz(n());
}
Poly Exp() const { // (*this)[0] == 0, 1e5/360ms
    if (n() == 1) return {1};
    Poly X = Poly(*this, (n() + 1) / 2).Exp().isz(n());
    Poly Y = X.Ln(); Y[0] = P - 1;
    fi(0, n()) if ((Y[i] = (*this)[i] - Y[i]) < 0) Y[i] +=
        P;
    return X.Mul(Y).isz(n());
}
// M := P(P - 1). If k >= M, k := k % M + M.
Poly Pow(ll k) const {
    int nz = 0;
    while (nz < n() && !(*this)[nz]) ++nz;
    if (nz * min(k, (ll)n()) >= n()) return Poly(n());
    if (!k) return Poly(Poly {1}, n());
    Poly X(data() + nz, data() + nz + n() - nz * k);
    const ll c = ntt.mpow(X[0], k % (P - 1));
    return X.Ln().imul(k % P).Exp().imul(c).irev().isz(n())
        .irev();
}
static ll LinearRecursion(const vector<ll> &a, const
    vector<ll> &coef, ll n) { // a_n = \sum c_j a_{n-j}
    const int k = (int)a.size();
    assert((int)coef.size() == k + 1);
    Poly C(k + 1), W(Poly {1}, k), M = {0, 1};
    fi(1, k + 1) C[k - i] = coef[i] ? P - coef[i] : 0;
    C[k] = 1;
    while (n) {
        if (n % 2) W = W.Mul(M).DivMod(C).second;
        n /= 2, M = M.Mul(M).DivMod(C).second;
    }
    ll ret = 0;
    fi(0, k) ret = (ret + W[i] * a[i]) % P;
    return ret;
}
};
#undef fi
using Poly_t = Poly<131072 * 2, 998244353, 3>;
template<> decltype(Poly_t::ntt) Poly_t::ntt = {};

```

## 8.4 Generating Function

### 8.4.1 Ordinary Generating Function

- $C(x) = A(rx)$ :  $c_n = r^n a_n$  的一般生成函數。
- $C(x) = A(x) + B(x)$ :  $c_n = a_n + b_n$  的一般生成函數。
- $C(x) = A(x)B(x)$ :  $c_n = \sum_{i=0}^n a_i b_{n-i}$  的一般生成函數。

- $C(x) = A(x)^k$ :  $c_n = \sum_{i_1+i_2+\dots+i_k=n} a_{i_1}a_{i_2}\dots a_{i_k}$  的一般生成函數。
- $C(x) = xA(x)'$ :  $c_n = na_n$  的一般生成函數。
- $C(x) = \frac{A(x)}{1-x}$ :  $c_n = \sum_{i=0}^n a_i$  的一般生成函數。

常用展開式

- $\frac{1}{1-x} = 1 + x + x^2 + \dots + x^n + \dots$
- $(1+x)^a = \sum_{n=0}^{\infty} \binom{a}{n} x^n$ ,  $\binom{a}{n} = \frac{a(a-1)(a-2)\dots(a-n+1)}{n!}$ .

## 8.4.2 Exponential Generating Function

$a_0, a_1, \dots$  的指數生成函數：

$$\hat{A}(x) = \sum_{i=0}^{\infty} \frac{a_i}{i!} = a_0 + a_1x + \frac{a_2}{2!}x^2 + \frac{a_3}{3!}x^3 + \dots$$

- $\hat{C}(x) = \hat{A}(x) + \hat{B}(x)$ :  $c_n = a_n + b_n$  的指數生成函數
- $\hat{C}(x) = \hat{A}^{(k)}(x)$ :  $c_n = a_{n+k}$  的指數生成函數
- $\hat{C}(x) = x\hat{A}(x)$ :  $c_n = na_n$  的指數生成函數
- $\hat{C}(x) = \hat{A}(x)\hat{B}(x)$ :  $c_n = \sum_{k=0}^n \binom{n}{k} a_k b_{n-k}$  的指數生成函數
- $\hat{C}(x) = \hat{A}(x)^k$ :  $c_n = \sum_{i_1+i_2+\dots+i_k=n} \binom{n}{i_1, i_2, \dots, i_k} a_{i_1}a_{i_2}\dots a_{i_k}$  的指數生成函數

# 9 String

## 9.1 Rolling Hash

```
int n;
string s;
vector<ll> h, rh;
vector<ll> kp;
const ll K = 26, MOD = 1000000007;

void topos(ll &a){
    a = (a % MOD + MOD) % MOD;
}

int ord(char c){
    return c - 'a';
}

pll geth(int l, int r){
    if(l > r) return mp(0, 0);
    ll ans = h[r] - h[l - 1] * kp[r - l + 1];
    topos(ans);
    return mp(ans, r - l + 1);
}

pll getrh(int l, int r){
    if(l > r) return mp(0, 0);
    l = n - l + 1;
    r = n - r + 1;
    swap(l, r);
    ll ans = rh[r] - rh[l - 1] * kp[r - l + 1];
    topos(ans);
    return mp(ans, r - l + 1);
}

pll concat(pll a, pll b){
    ll ans = a.F * kp[b.S] + b.F;
    ans %= MOD;
    return mp(ans, a.S + b.S);
}

void build(){
    n = s.size();
    s = " " + s;

    h.resize(n + 1);
    rh.resize(n + 1);
    kp.resize(n + 1);
```

```
kp[0] = 1;
for(int i = 1; i <= n; i++){
    kp[i] = kp[i - 1] * K % MOD;
}
for(int i = 1; i <= n; i++){
    h[i] = h[i - 1] * K % MOD + ord(s[i]);
    h[i] %= MOD;
    rh[i] = rh[i - 1] * K % MOD + ord(s[n - i + 1]);
    rh[i] %= MOD;
}
}
```

## 9.2 KMP Algorithm

```
void kmp(string s){
    int siz = s.size();
    vector<int> f(siz, 0);
    f[0] = 0;
    for (int i = 1; i < siz; i++) {
        f[i] = f[i-1];
        bool zero = 0;
        while (s[f[i]] != s[i]) {
            if (f[i] == 0) {
                zero = 1;
                break;
            }
            f[i] = f[f[i]-1];
        }
        if (!zero) f[i]++;
    }
}
```

## 9.3 Manacher Algorithm

```
vector<int> manacher(string s) {
    int n = s.size();
    vector<int> v(n);
    int pnt = -1, len = 1;
    for (int i = 0; i < n; i++) {
        int cor = 2 * pnt - i;
        if (cor >= 0) v[i] = min(v[cor], cor - pnt + len);
        while (i+v[i] < n && i-v[i] >= 0 && s[i+v[i]] == s[i-v[i]]) v[i]++;
        if (i + v[i] >= pnt + len) pnt = i, len = v[i];
    }
    for (int i = 0; i < n; i++) v[i] = 2 * v[i] - 1;
    return v;
}
```

## 9.4 MCP

```
string mcp(string s) { //Duval algorithm for Lyndon
    factorization
    s += s;
    int n = s.size(), i = 0, ans = 0;
    while (i < n/2) {
        ans = i;
        int j = i+1, k=i;
        while (j < n && s[k] <= s[j]) {
            if (s[k] < s[j]) k = i;
            else k++;
            j++;
        }
        while (i <= k) i += j - k;
    }
    return s.substr(ans, n/2);
}
```

## 9.5 Suffix Array



```

struct SuffixArray { //tested
    vector<int> sa, lcp, rank; //lcp[i] is lcp of sa[i] and
    sa[i-1]
    SuffixArray(string& s, int lim=256) { // or basic_string<
    int>
        int n = s.size() + 1, k = 0, a, b;
        vector<int> x(n, 0), y(n), ws(max(n, lim));
        rank.resize(n);
        for (int i = 0; i < n-1; i++) x[i] = (int)s[i];
        sa = lcp = y, iota(sa.begin(), sa.end(), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim =
        p) {
            p = j, iota(y.begin(), y.end(), n - j);
            for (int i = 0; i < n; i++) if (sa[i] >= j) y[p++] = sa
            [i] - j;
            for (int &i : ws) i = 0;
            for (int i = 0; i < n; i++) ws[x[i]]++;
            for (int i = 1; i < lim; i++) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            for (int i = 1; i < n; i++) a = sa[i - 1], b = sa[i], x[
            b] =
                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p
                ++;
        }
        for (int i = 1; i < n; i++) rank[sa[i]] = i;
        for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
            for (k && k--, j = sa[rank[i] - 1];
                s[i + k] == s[j + k]; k++);
    }
};

```

## 9.6 Suffix Array Automaton

```

//to be modified
const int MAXM = 1000010;
struct SAM {
    int tot, root, lst, mom[MAXM], mx[MAXM];
    int acc[MAXM], nxt[MAXM][33];
    int newNode() {
        int res = ++tot;
        fill(nxt[res], nxt[res] + 33, 0);
        mom[res] = mx[res] = acc[res] = 0;
        return res;
    }
    void init() {
        tot = 0;
        root = newNode();
        mom[root] = 0, mx[root] = 0;
        lst = root;
    }
    void push(int c) {
        int p = lst;
        int np = newNode();
        mx[np] = mx[p] + 1;
        for (; p && nxt[p][c] == 0; p = mom[p])
            nxt[p][c] = np;
        if (p == 0) mom[np] = root;
        else {
            int q = nxt[p][c];
            if (mx[p] + 1 == mx[q]) mom[np] = q;
            else {
                int nq = newNode();
                mx[nq] = mx[p] + 1;
                for (int i = 0; i < 33; i++)
                    nxt[nq][i] = nxt[q][i];
                mom[nq] = mom[q];
                mom[q] = nq;
                mom[np] = nq;
                for (; p && nxt[p][c] == q; p = mom[p])
                    nxt[p][c] = nq;
            }
        }
        lst = np;
    }
    void push(char *str) {

```

```

        for (int i = 0; str[i]; i++)
            push(str[i] - 'a' + 1);
    }
} sam;

```

## 9.7 Z-value Algorithm

```

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

```

## 9.8 Main Lorentz

```

vector<int> z_function(string const& s);
int get_z(vector<int> const& z, int i) {
    return (0 <= i && i < SZ(z)) ? z[i] : 0;
}
vector<pair<int, int>> rep;

void convert_to_rep(int shift, bool left, int cntr, int l,
    int k1, int k2) {
    for (int l1 = max(1, l - k2); l1 <= min(l, k1); l1++) {
        if (left && l1 == 1) break;
        int l2 = l - l1;
        int pos = shift + (left ? cntr - l1 : cntr - l - l1 +
        1);
        rep.emplace_back(pos, pos + 2*l - 1);
    }
}

void find_rep(string s, int shift = 0) {
    int n = s.size();
    if (n == 1) return;

    int nu = n / 2;
    int nv = n - nu;
    string u = s.substr(0, nu);
    string v = s.substr(nu);
    string ru(u.rbegin(), u.rend());
    string rv(v.rbegin(), v.rend());

    find_rep(u, shift);
    find_rep(v, shift + nu);

    vector<int> z1 = z_function(ru);
    vector<int> z2 = z_function(v + '#' + u);
    vector<int> z3 = z_function(ru + '#' + rv);
    vector<int> z4 = z_function(v);

    for (int cntr = 0; cntr < n; cntr++) {
        int l, k1, k2;
        if (cntr < nu) {
            l = nu - cntr;
            k1 = get_z(z1, nu - cntr);
            k2 = get_z(z2, nv + 1 + cntr);
        } else {
            l = cntr - nu + 1;
            k1 = get_z(z3, nu + 1 + nv - 1 - (cntr - nu));
            k2 = get_z(z4, (cntr - nu) + 1);
        }
        if (k1 + k2 >= 1)
            convert_to_rep(shift, cntr < nu, cntr, l, k1, k2);
    }
}

```

## 9.9 AC Automaton

```
// copy from nontoi
struct AhoCorasick {
    enum { P = 26, st = 'a' };
    struct node { // zero-based
        array<int, P> ch = {0};
        int fail = 0, cnt = 0, dep = 0;
    };
    int cnt;
    vector<node> v;
    vector<int> ans;
    void init_(int mx) {
        v.clear();
        cnt = 1, v.resize(mx);
        v[0].fail = 0;
    }
    void insert(string s) {
        int p = 0, dep = 1;
        for(auto i : s) {
            int c = i - st;
            if(!v[p].ch[c]) {
                v[cnt].dep = dep;
                v[p].ch[c] = cnt ++;
            }
            p = v[p].ch[c], dep ++;
        }
        v[p].cnt ++;
    }
    void build(vector<string> s) {
        for(auto i : s) insert(i);
        queue<int> q;
        for(int i = 0; i < P; i ++ ) {
            if(v[0].ch[i]) q.push(v[0].ch[i]);
        }
        while(q.size()) {
            int p = q.front();
            q.pop();
            for(int i = 0; i < P; i ++ ) if(v[p].ch[i]) {
                int to = v[p].ch[i], cur = v[p].fail;
                while(cur && !v[cur].ch[i]) cur = v[cur].fail;
                if(v[cur].ch[i]) cur = v[cur].ch[i];
                v[to].fail = cur;
                v[to].cnt += v[cur].cnt;
                q.push(to);
            }
        }
    }
    void traverse(string s) {
        int p = 0;
        ans.assign(cnt, 0);
        for(auto i : s) {
            int c = i - st;
            while(p && !v[p].ch[c]) p = v[p].fail;
            if(v[p].ch[c]) {
                p = v[p].ch[c];
                ans[p] ++, v[p].cnt;
            }
        }
        vector<int> ord(cnt, 0);
        iota(all(ord), 0);
        sort(all(ord), [&](int a, int b) { return v[a].dep > v[b].dep; });
        for(auto i : ord) ans[v[i].fail] += ans[i];
        return;
    }
    int go(string s) {
        int p = 0;
        for(auto i : s) {
            int c = i - st;
            assert(v[p].ch[c]);
            p = v[p].ch[c];
        }
        return ans[p];
    }
};
```

## 10 Formula

### 10.1 Recurrences

If  $a_n = c_1 a_{n-1} + \dots + c_k a_{n-k}$ , and  $r_1, \dots, r_k$  are distinct roots of  $x^k + c_1 x^{k-1} + \dots + c_k$ , there are  $d_1, \dots, d_k$  s.t.

$$a_n = d_1 r_1^n + \dots + d_k r_k^n.$$

Non-distinct roots  $r$  become polynomial factors, e.g.  $a_n = (d_1 n + d_2) r^n$ .

### 10.2 Geometry

#### 10.2.1 Triangles

Side lengths:  $a, b, c$

$$\text{Semiperimeter: } p = \frac{a+b+c}{2}$$

$$\text{Area: } A = \sqrt{p(p-a)(p-b)(p-c)}$$

$$\text{Circumradius: } R = \frac{abc}{4A}$$

$$\text{Inradius: } r = \frac{A}{p}$$

Length of median (divides triangle into two equal-area triangles):  $m_a = \frac{1}{2} \sqrt{2b^2 + 2c^2 - a^2}$

$$\text{Length of bisector (divides angles in two): } s_a = \sqrt{bc \left( 1 - \left( \frac{a}{b+c} \right)^2 \right)}$$

$$\text{Law of sines: } \frac{\sin \alpha}{a} = \frac{\sin \beta}{b} = \frac{\sin \gamma}{c} = \frac{1}{2R}$$

$$\text{Law of cosines: } a^2 = b^2 + c^2 - 2bc \cos \alpha$$

$$\text{Law of tangents: } \frac{a+b}{a-b} = \frac{\tan \frac{\alpha+\beta}{2}}{\tan \frac{\alpha-\beta}{2}}$$

Incenter:

$$P_1 = (x_1, y_1), P_2 = (x_2, y_2), P_3 = (x_3, y_3)$$

$$s_1 = P_2 P_3, s_2 = P_1 P_3, s_3 = P_1 P_2$$

$$\frac{s_1 P_1 + s_2 P_2 + s_3 P_3}{s_1 + s_2 + s_3}$$

Circumcenter:

$$P_0 = (0, 0), P_1 = (x_1, y_1), P_2 = (x_2, y_2)$$

$$x_c = \frac{1}{2} \times \frac{y_2(x_1^2 + y_1^2) - y_1(x_2^2 + y_2^2)}{-x_2 y_1 + x_1 y_2}$$

$$y_c = \frac{1}{2} \times \frac{x_2(x_1^2 + y_1^2) - x_1(x_2^2 + y_2^2)}{-x_1 y_2 + x_2 y_1}$$

Check if  $(x_0, y_0)$  is in the circumcircle:

$$\begin{vmatrix} x_1 - x_0 & y_1 - y_0 & (x_1^2 + y_1^2) - (x_0^2 + y_0^2) \\ x_2 - x_0 & y_2 - y_0 & (x_2^2 + y_2^2) - (x_0^2 + y_0^2) \\ x_3 - x_0 & y_3 - y_0 & (x_3^2 + y_3^2) - (x_0^2 + y_0^2) \end{vmatrix}$$

0: on edge, > 0: inside, < 0: outside

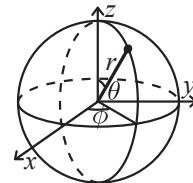
#### 10.2.2 Quadrilaterals

With side lengths  $a, b, c, d$ , diagonals  $e, f$ , diagonals angle  $\theta$ , area  $A$  and magic flux  $F = b^2 + d^2 - a^2 - c^2$ :

$$4A = 2ef \cdot \sin \theta = F \tan \theta = \sqrt{4e^2 f^2 - F^2}$$

For cyclic quadrilaterals the sum of opposite angles is  $180^\circ$ ,  $ef = ac + bd$ , and  $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$ .

#### 10.2.3 Spherical coordinates



$$\begin{aligned} x &= r \sin \theta \cos \phi & r &= \sqrt{x^2 + y^2 + z^2} \\ y &= r \sin \theta \sin \phi & \theta &= \arccos(z / \sqrt{x^2 + y^2 + z^2}) \\ z &= r \cos \theta & \phi &= \operatorname{atan2}(y, x) \end{aligned}$$

### 10.2.4 Green's Theorem

$$\iint_D \left( \frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y} \right) dx dy = \oint_{L^+} (P dx + Q dy)$$

$$\text{Area} = \frac{1}{2} \oint_L x dy - y dx$$

Circular sector:

$$x = x_0 + r \cos \theta$$

$$y = y_0 + r \sin \theta$$

$$A = r \int_{\alpha}^{\beta} (x_0 + \cos \theta) \cos \theta + (y_0 + \sin \theta) \sin \theta d\theta$$

$$= r(r\theta + x_0 \sin \theta - y_0 \cos \theta)|_{\alpha}^{\beta}$$

## 10.3 Trigonometry

$$\sinh x = \frac{1}{2}(e^x - e^{-x}) \quad \cosh x = \frac{1}{2}(e^x + e^{-x})$$

$$\sin n\pi = 0 \quad \cos n\pi = (-1)^n$$

$$\sin(\alpha + \beta) = \sin \alpha \cos \beta + \cos \alpha \sin \beta$$

$$\cos(\alpha + \beta) = \cos \alpha \cos \beta - \sin \alpha \sin \beta$$

$$\sin(2\alpha) = 2 \cos \alpha \sin \alpha$$

$$\cos(2\alpha) = \cos^2 \alpha - \sin^2 \alpha$$

$$= 2 \cos^2 \alpha - 1$$

$$= 1 - 2 \sin^2 \alpha$$

$$\tan(\alpha + \beta) = \frac{\tan \alpha + \tan \beta}{1 - \tan \alpha \tan \beta}$$

$$\sin \alpha + \sin \beta = 2 \sin \frac{\alpha + \beta}{2} \cos \frac{\alpha - \beta}{2}$$

$$\cos \alpha + \cos \beta = 2 \cos \frac{\alpha + \beta}{2} \cos \frac{\alpha - \beta}{2}$$

$$\sin \alpha \sin \beta = \frac{1}{2}(\cos(\alpha - \beta) - \cos(\alpha + \beta))$$

$$\sin \alpha \cos \beta = \frac{1}{2}(\sin(\alpha + \beta) + \sin(\alpha - \beta))$$

$$\cos \alpha \sin \beta = \frac{1}{2}(\sin(\alpha + \beta) - \sin(\alpha - \beta))$$

$$\cos \alpha \cos \beta = \frac{1}{2}(\cos(\alpha - \beta) + \cos(\alpha + \beta))$$

$$(V + W) \tan(\alpha - \beta)/2 = (V - W) \tan(\alpha + \beta)/2$$

where  $V, W$  are lengths of sides opposite angles  $\alpha, \beta$ .

$$a \cos x + b \sin x = r \cos(x - \phi)$$

$$a \sin x + b \cos x = r \sin(x + \phi)$$

where  $r = \sqrt{a^2 + b^2}$ ,  $\phi = \text{atan2}(b, a)$ .

## 10.4 Derivatives/Integrals

Integration by parts:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

$$\frac{d}{dx} \arcsin x = \frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx} \tan x = 1 + \tan^2 x$$

$$\int \tan ax = -\frac{\ln |\cos ax|}{a}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2} \text{erf}(x)$$

$$\int \sin^2(x) = \frac{x}{2} - \frac{1}{4} \sin 2x$$

$$\int \cos^2(x) = \frac{x}{2} + \frac{1}{4} \sin 2x$$

$$\int x \sin x = \sin x - x \cos x$$

$$\int x e^x = e^x(x - 1)$$

$$\frac{d}{dx} \arccos x = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx} \arctan x = \frac{1}{1+x^2}$$

$$\int x \sin ax = \frac{\sin ax - ax \cos ax}{a^2}$$

$$\int x e^{ax} = \frac{e^{ax}}{a^2}(ax - 1)$$

$$\int \sin^3 x = \frac{1}{12} \cos 3x - \frac{3}{4} \cos x$$

$$\int \cos^3 x = \frac{1}{12} \sin 3x + \frac{3}{4} \sin x$$

$$\int x \cos x = \cos x + x \sin x$$

$$\int x^2 e^x = e^x(x^2 - 2x + 2)$$

$$\int x^2 \sin x = 2x \sin x - (x^2 - 2) \cos x$$

$$\int x^2 \cos x = 2x \cos x + (x^2 - 2) \sin x$$

$$\int e^x \sin x = \frac{1}{2} e^x (\sin x - \cos x)$$

$$\int e^x \cos x = \frac{1}{2} e^x (\sin x + \cos x)$$

$$\int x e^x \sin x = \frac{1}{2} e^x (x \sin x - x \cos x + \cos x)$$

$$\int x e^x \cos x = \frac{1}{2} e^x (x \sin x + x \cos x - \sin x)$$

## 10.5 Sums

$$c^a + c^{a+1} + \dots + c^b = \frac{c^{b+1} - c^a}{c - 1}, c \neq 1$$

$$1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

$$1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^2(n+1)^2}{4}$$

$$1^4 + 2^4 + 3^4 + \dots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

## 10.6 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots, (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \dots, (-1 < x \leq 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \dots, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots, (-\infty < x < \infty)$$

## 10.7 Probability theory

Let  $X$  be a discrete random variable with probability  $p_X(x)$  of assuming the value  $x$ . It will then have an expected value (mean)  $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$  and variance  $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$  where  $\sigma$  is the standard deviation. If  $X$  is instead continuous it will have a probability density function  $f_X(x)$  and the sums above will instead be integrals with  $p_X(x)$  replaced by  $f_X(x)$ .

Expectation is linear:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

For independent  $X$  and  $Y$ ,

$$V(aX + bY) = a^2 V(X) + b^2 V(Y).$$

### 10.7.1 Discrete distributions

**Binomial distribution** The number of successes in  $n$  independent yes/no experiments, each which yields success with probability  $p$  is  $\text{Bin}(n, p)$ ,  $n = 1, 2, \dots$ ,  $0 \leq p \leq 1$ .

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np, \sigma^2 = np(1-p)$$

$\text{Bin}(n, p)$  is approximately  $\text{Po}(np)$  for small  $p$ .

**First success distribution** The number of trials needed to get the first success in independent yes/no experiments, each which yields success with probability  $p$  is  $\text{Fs}(p)$ ,  $0 \leq p \leq 1$ .

$$p(k) = p(1-p)^{k-1}, k = 1, 2, \dots$$

$$\mu = \frac{1}{p}, \sigma^2 = \frac{1-p}{p^2}$$

**Poisson distribution** The number of events occurring in a fixed period of time  $t$  if these events occur with a known average rate  $\kappa$  and independently of the time since the last event is  $\text{Po}(\lambda)$ ,  $\lambda = t\kappa$ .

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, k = 0, 1, 2, \dots$$

$$\mu = \lambda, \sigma^2 = \lambda$$

### 10.7.2 Continuous distributions

**Uniform distribution** If the probability density function is constant between  $a$  and  $b$  and 0 elsewhere it is  $\text{U}(a, b)$ ,  $a < b$ .

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$

**Exponential distribution** The time between events in a Poisson process is  $\text{Exp}(\lambda)$ ,  $\lambda > 0$ .

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

**Normal distribution** Most real random values with mean  $\mu$  and variance  $\sigma^2$  are well described by  $\mathcal{N}(\mu, \sigma^2)$ ,  $\sigma > 0$ .

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If  $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$  and  $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$  then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

## 10.8 Markov chains

A *Markov chain* is a discrete random process with the property that the next state depends only on the current state. Let  $X_1, X_2, \dots$  be a sequence of random variables generated by the Markov process. Then there is a transition matrix  $\mathbf{P} = (p_{ij})$ , with  $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$ , and  $\mathbf{p}^{(n)} = \mathbf{P}^n \mathbf{p}^{(0)}$  is the probability distribution for  $X_n$  (i.e.,  $p_i^{(n)} = \Pr(X_n = i)$ ), where  $\mathbf{p}^{(0)}$  is the initial distribution.

$\pi$  is a stationary distribution if  $\pi = \pi \mathbf{P}$ . If the Markov chain is *irreducible* (it is possible to get to any state from any state), then  $\pi_i = \frac{1}{\mathbb{E}(T_i)}$  where  $\mathbb{E}(T_i)$  is the expected time between two visits in state  $i$ .  $\pi_j / \pi_i$  is the expected number of visits in state  $j$  between two visits in state  $i$ .

For a connected, undirected and non-bipartite graph, where the transition probability is uniform among all neighbors,  $\pi_i$  is proportional to node  $i$ 's degree.

A Markov chain is *ergodic* if the asymptotic distribution is independent of the initial distribution. A finite Markov chain is ergodic iff it is irreducible and *aperiodic* (i.e., the gcd of cycle lengths is 1).  $\lim_{k \rightarrow \infty} \mathbf{P}^k = \mathbf{1}\pi$ .

A Markov chain is an *A-chain* if the states can be partitioned into two sets  $\mathbf{A}$  and  $\mathbf{G}$ , such that all states in  $\mathbf{A}$  are absorbing ( $p_{ii} = 1$ ), and all states in  $\mathbf{G}$  leads to an absorbing state in  $\mathbf{A}$ . The probability for absorption in state  $i \in \mathbf{A}$ , when the initial state is  $j$ , is  $a_{ij} = p_{ij} + \sum_{k \in \mathbf{G}} a_{ik} p_{kj}$ . The expected time until absorption, when the initial state is  $i$ , is  $t_i = 1 + \sum_{k \in \mathbf{G}} p_{ki} t_k$ .



