Team Reference Document

天津大学 Tianjin University

神偷 TJU_Thief_Master

Coach	**教　练**
Ruiguo Yu	于瑞国
Mei Yu	喻　梅

Team member	**队　员**
Sheng Cao	曹　圣
Tingle Zhou	周挺乐
Yibo Shi	师一博

ACM-ICPC World Finals 2016

目录

- vim 配置 1
- glibc 内建函数 1
- 快速傅里叶变换(FFT) 1
- 数论变换(NTT) 1
- 多项式的逆元 1
- 多项式的除法 1
- 多项式开根号 1
- 矩阵快速幂 1
- 在线 FFT 1
- 划分树 1
- 线段树 2
- 左偏树 2
- Splay 2
- KD-Tree 2
- 树链剖分 3
- Link-Cut Tree 3
- 网络流 3
- 网络流建图模型 4
- 上下界网络流 4
- 费用流 4
- 强连通分量 4
- 割点、割边 5
- 二维几何基础 5
- 圆相关 6
- 凸包 6
- V 图 7
- 三角形 7
- 四边形 7
- 平面定理 7
- 三维几何体 7
- 高维球 7
- 三维几何 7
- 扩展欧几里得 7
- 线性求乘法逆元（mod 为质数） 7
- 线性筛素数 7
- 高斯消元 7
- 欧拉函数 7
- 大素数判定 8
- 大数分解 8
- 莫比乌斯反演 8
- 波利亚 8
- Java 大数 8
- 常用大素数 8
- 约数个数 8
- 浮点数求和(Kahan Summation) 8
- Hash 8
- KMP 8
- exKMP 8
- Manacher 8
- AC 自动机 9
- 后缀自动机 9
- 回文自动机 9
- 后缀数组（倍增） 9
- ST 9
- Dancing Links 9

- **vim 配置**

```
set number
set showcmd
set hls
filetype on
filetype indent on
filetype plugin on
colorscheme ron
set ts=4
set sw=4
nmap ,s :w<cr>:sh<cr>
nmap ,/ I//<esc>
nmap ,\ I<del><del><esc>
```

```
func! Com()
    exec "w"
    let cmd="!g++"
    let flag="-o %< "
    exec cmd." % ".flag
endfunc
func! Run()
    exec "!./%<"
endfunc
nmap ,g :call Com()<cr>
nmap ,r :call Run()<cr>
nmap ,y mkgg"+yG`k
nmap ,p "+p
```

- **glibc 内建函数**

```
int __builtin_ffs (unsigned int x); //返回右起第一个 1 的位置，最低位为第 1 位
int __builtin_clz (unsigned int x); //返回左起第一个 1 之前的 0 的个数
int __builtin_ctz (unsigned int x); //返回右起第一个 1 之后的 0 的个数
int __builtin_popcount (unsigned int x); //返回 1 的个数
int __builtin_parity (unsigned int x); //返回 1 的个数的奇偶性，奇数个则返回 1
```

- **快速傅里叶变换(FFT)**

```
void FFT(Complex a[], int n, int oper) {
    for (int i = 1, j = 0; i < n; i++) {
        for (int s = n; j ^= s >>= 1, ~j & s; );
        if (i < j) swap(a[i], a[j]);
    }
    for (int m = 1; m < n; m *= 2) {
        double p = PI / m * oper;
        Complex w = Complex(cos(p), sin(p));
        for (int i = 0; i < n; i += m * 2) {
            Complex unit = 1;
            for (int j = 0; j < m; j++) {
                Complex &x = a[i + j + m], &y = a[i + j], t = unit * x;
                x = y - t;
                y = y + t;
                unit = unit * w;
            }
        }
    }
    if (oper == -1) for (int i = 0; i < n; i++) a[i] = a[i] / n;
}
```

- **数论变换(NTT)**

长度 n 必须为 mod - 1 的约数，否则找不到 n 等分点 w，即 pow(w, n) = 1

```
int w = pow(g, (mod - 1) / (2 * m));
if (oper == -1) w = pow(w, mod - 2);
```

无法直接 NTT，则做三次乘法，然后用 CRT 求出，需要用到**__int128**。

```
ans = (ans1 * mod2 * mod3 * inv1 + ans2 * mod3 * mod1 * inv2 + ans3 * mod1
* mod2 * inv3) % (mod1 * mod2 * mod3);
```

常用大素数

```
mod1 = (31 * 31 << 20) + 1        g1 = 3         inv1 = 346,612,643
mod2 = (17 * 59 << 20) + 1        g2 = 6         inv2 = 408,151,354
mod3 = (3 << 18) + 1              g3 = 10        inv3 = 210,725
```

也可以将多项式拆成$P(x) = P1(x) + \sqrt{mod} * P2(x)$，然后共做 7 次 FFT，需要用到 **long double**

- **多项式的逆元**

$$g_{2n}(x) \equiv 2g_n(x) - g_n(x)^2 f(x) \ \ (mod \ \ x^{2n})$$

- **多项式的除法**

设 n 阶多项式$f_n(x) = q_{n-m}(x)g_m(x) + r_{m-1}(x)$，则$q(x)$为$f(x)/g(x)$的商，$r(x)$为余数

记$h'_k(x) = x^k h_k(1/x)$，即 h’=reverse(h),则$q'_{n-m}(x) = f'_n(x)\left(g'_m(x)\right)^{-1} \ \ (mod \ \ x^{n-m+1})$

- **多项式开根号**

$$g_{2n}(x) \equiv (g_n(x) + f(x)g_n(x)^{-1})/2 \ \ (mod \ \ x^{2n})$$

- **矩阵快速幂**

设递推式为$h_n = \sum_{i=1}^{k} a_i h_{n-i}$，令$X = [h_k, h_{k-1}, \dots, h_1]^{-1}$，$M^{n-1}X = [h_{n+k-1}, \dots, h_n]^{-1}$

则$M^{p+k} = \sum_{i=1}^{k} a_i \, M^{p+k-i}$，矩阵相乘变为两个多项式相乘，再将 2k-2⋯k 的部分合并下去

多项式乘可以使用 FFT，合并操作可以视为求除以多项式$(a_k, a_{k-1}, \dots, a_1, -1)$的余数

- **在线 FFT**

给向量 a 和向量 b，b[0] = 0，求向量 c 为 a 与 b 的卷积。每次给 a[t]，求 c[t + 1]

```
c[t + 1] += a[t] * b[1];
if (t != 0) for (int m = 1; t % m == 0; m *= 2)
    c[t + 1 .. t + m * 2] += a[t - m .. t - 1] * b[m + 1 .. m * 2];
```

- **划分树**

```
int a[N], b[N], tree[K][N], num[K][N];
void build(int i, int l, int r) {
    if (l == r) return;
    int t = (l + r) / 2, e = t - l + 1, j, pl = l, pr = t + 1;
    for (j = l; j <= t; j++) if (b[j] < b[t]) e--;
    for (j = l; j <= r; j++) {
        num[i][j] = (j == l)? 0 : num[i][j - 1];
        if (tree[i][j] < b[t]) {
            num[i][j]++;
            tree[i + 1][pl++] = tree[i][j];
        } else if (tree[i][j] > b[t]) {
            tree[i + 1][pr++] = tree[i][j];
        } else if (e > 0) {
            e--;
            tree[i + 1][pl++] = tree[i][j];
            num[i][j]++;
        } else {
            tree[i + 1][pr++] = tree[i][j];
        }
    }
    build(i + 1, l, t);
```

```
    build(i + 1, t + 1, r);
}
int get(int i, int l, int r, int ql, int qr, int k) {
    if (l == r) return tree[i][l];
    int t = (l + r) / 2
    int s = (l == ql)? 0 : num[i][ql - 1], ss = num[i][qr] - s;
    if (k <= ss) return get(i + 1, l, t, l + s, l + s + ss - 1, k);
    return get(i + 1, t + 1, r, ql + t - l + 1 - s,
            qr + t - l + 1 - s - ss, k - ss);
}
void demo() {
    for (int i = 1; i <= n; i++) tree[0][i] = b[i] = a[i];
    sort(b + 1, b + n + 1);
    build(0, 1, n);
    get(0, 1, n, l, r, k);
}
```

- **线段树**

若下标在[0, nn]范围内，其中 nn = ~0u >> __builtin_clz(n)，即大于等于 n 的最小的 $2^k - 1$，则可以直接用 a[l + r]表示[l, r]这个节点。

- **左偏树**
- Splay

```
struct Node {
    Node *ls, *rs, *f;
    int a, b, minb; //按照 a 排序，保留 b 的最小值
    void update() {
        minb = b;
        if (ls) minb = min(minb, ls->minb);
        if (rs) minb = min(minb, rs->minb);
    }
    Node *clear(int aa, int bb, Node *ff = NULL) {
        a = aa; b = bb; minb = bb;
        f = ff; ls = rs = NULL;
        return this;
    }
    void rot() { //旋转到他的父亲位置
        Node *x = this, *y = f;
        if (x == y->ls) {
            y->ls = x->rs;
            x->rs = y;
            if (y->ls) y->ls->f = y;
        } else {
            y->rs = x->ls;
            x->ls = y;
            if (y->rs) y->rs->f = y;
        }
        x->f = y->f;
        y->f = x;
        if (x -> f) {
            if (x->f->ls == y) x->f->ls = x;
            else x->f->rs = x;
        }
        y->update();
        x->update();
    }
    int dir() { //判断是父亲的左孩子还是右孩子
        if (this->f) {
            if (this == this->f->ls) return -1;
            else return 1;
        }
        return 0;
    }
};
Node c[N], *root, *cp;
Node *splay(Node *x, Node *f = NULL) { //将 x 提为根
    while (x->f != f) {
        if (x->f->f == f) x->rot();
        else if (x->dir() == x->f->dir()) {
            x->f->rot();
            x->rot();
        } else {
            x->rot();
            x->rot();
        }
    }
    return x;
}
void demo() {
    cp = c;
    root = (cp++)->clear(-INF, INF);
    root->rs = (cp++)->clear(INF, INF, root);
    root->update();
    // 想求第 k 大元素的话，需要维护 size 信息
}
```

- KD-Tree

```
int id;
struct Point {
    int x[2];
    friend bool operator < (const Point &a, const Point &b) {
        return a.x[id] < b.x[id];
    }
    friend bool operator <= (const Point &a, const Point &b) {
        return a.x[id] <= b.x[id];
    }
};
struct Node {
    Point l, r, x;
    int v, maxv;
};
Point b[N];
Node c[N * 2];
```

```
void updateArea(int x, int y) {
    c[x].l.x[0] = min(c[x].l.x[0], c[y].l.x[0]);
    c[x].l.x[1] = min(c[x].l.x[1], c[y].l.x[1]);
    c[x].r.x[0] = max(c[x].r.x[0], c[y].r.x[0]);
    c[x].r.x[1] = max(c[x].r.x[1], c[y].r.x[1]);
}
void update(int d, int l, int r) {
    int t = (l + r) >> 1, ls = d << 1, rs = ls | 1;
    c[d].maxv = c[d].v;
    if (l < t) c[d].maxv = max(c[d].maxv, c[ls].maxv);
    if (t + 1 < r) c[d].maxv = max(c[d].maxv, c[rs].maxv);
}
bool build(int d, int l, int r, int o) { // c[d] => [l..r)
    if (l >= r) return false;
    int t = (l + r) >> 1, ls = d << 1, rs = ls | 1;
    id = o;
    nth_element(b + l, b + t, b + r);
    c[d].l = c[d].r = c[d].x = b[t];
    c[d].v = 0;
    if (build(ls, l, t, o ^ 1)) updateArea(d, ls);
    if (build(rs, t + 1, r, o ^ 1)) updateArea(d, rs);
    update(d, l, r);
    return true;
}
void set(int d, int l, int r, int o, Point i, int x) {
    if (l >= r) return;
    if (c[d].x.x[0] == i.x[0] && c[d].x.x[1] == i.x[1]) {
        c[d].v = max(c[d].v, x);
        update(d, l, r);
    } else {
        int t = (l + r) >> 1, ls = d << 1, rs = ls | 1;
        id = o;
        if (i <= c[d].x) set(ls, l, t, o ^ 1, i, x);
        id = o;
        if (c[d].x <= i) set(rs, t + 1, r, o ^ 1, i, x);
        update(d, l, r);
    }
}
int get(int d, int l, int r, int o, Point ll, Point rr) {
    if (l >= r) return 0;
    if (c[d].l.x[0] > rr.x[0] || c[d].l.x[1] > rr.x[1] ||
            c[d].r.x[0] < ll.x[0] || c[d].r.x[1] < ll.x[1])
        return 0;
    if (c[d].l.x[0] >= ll.x[0] && c[d].l.x[1] >= ll.x[1] &&
            c[d].r.x[0] <= rr.x[0] && c[d].r.x[1] <= rr.x[1])
        return c[d].maxv;
    int t = (l + r) >> 1, ls = d << 1, rs = ls | 1;
    int ans = 0;
    if (c[d].x.x[0] >= ll.x[0] && c[d].x.x[1] >= ll.x[1] &&
            c[d].x.x[0] <= rr.x[0] && c[d].x.x[1] <= rr.x[1])
        ans = max(ans, c[d].v);
    ans = max(ans, get(ls, l, t, o ^ 1, ll, rr));
    ans = max(ans, get(rs, t + 1, r, o ^ 1, ll, rr));
    return ans;
}
void demo() {
    // b 中的元素顺序会被打乱，b 的元素范围在[0, n)内
    build(1, 0, n, 0);
    get(1, 0, n, 0, x, y);
    set(1, 0, n, 0, z, dp[i]);
}
```

- **树链剖分**

第一次 dfs 求出 f, h, size, zson，第二次 dfs 求出 top, dfn

- **Link-Cut Tree**
- **网络流**

```
struct NetWorkFlow {
    struct Edge {
        int t, f;
        Edge *ne, *p;
        Edge *clear(int tt, int ff, Edge *nee) {
            t = tt; f = ff; ne = nee;
            return this;
        }
    };
    Edge b[M * 2], *p, *fe[N], *cur[N];
    int n, s, t, h[N], vh[N];
    void clear(int nn, int ss, int tt) {
        n = nn; s = ss; t = tt;
        for (int i = 0; i < n; i++) fe[i] = NULL;
        p = b;
    }
    void putedge(int x, int y, int f) {
        fe[x] = (p++)->clear(y, f, fe[x]);
        fe[y] = (p++)->clear(x, 0, fe[y]);
        fe[x]->p = fe[y];
        fe[y]->p = fe[x];
    }
    int aug(int i, int f) {
        if (i == t) return f;
        int minh = n;
        Edge *seg = cur[i], *&j = cur[i];
        do {
            if (j->f) {
                if (h[j->t] + 1 == h[i]) {
                    int tmp = aug(j->t, min(j->f, f));
                    if (tmp) {
                        j->f -= tmp;
                        j->p->f += tmp;
                        return tmp;
                    }
                }
                minh = min(minh, h[j->t] + 1);
```

```
                if (h[s] == n) return 0;
            }
            j = j->ne;
            if (j == NULL) j = fe[i];
        } while (j != seg);
        if (!--vh[h[i]]) h[s] = n;
        else ++vh[h[i] = minh];
        return 0;
    }
    int flow() {
        if (fe[s] == NULL) return 0;
        int ans = 0;
        for (int i = 0; i <= n; i++) {
            cur[i] = fe[i];
            h[i] = vh[i] = 0;
        }
        vh[0] = n;
        while (h[s] < n) ans += aug(s, INF);
        return ans;
    }
};
```

- 网络流建图模型
- 上下界网络流
- 费用流

```
struct Node {
    int fe, ln, c, le; //ln 上一个点，le 上一条边，c 为 s 到当前点最小花费
    bool d; //是否在队列内
};
struct Edge {
    int f, t, ne, c;
};
Node a[N];
Edge b[M * 2];
int s, t, n, p, cost, flow;
void clear(int nn, int ss, int tt) {
    n = nn; s = ss; t = tt;
    for (int i = 0; i < n; i++) a[i].fe = -1;
    p = 0; cost = 0; flow = 0;
}
Edge *putedge(int x,int y,int f,int c) {
    b[p].ne = a[x].fe; b[p].t = y; b[p].f = f; b[p].c = c; a[x].fe = p++;
    b[p].ne = a[y].fe; b[p].t = x; b[p].f = 0; b[p].c = -c; a[y].fe = p++;
    return &b[p-2];
}
int add(int &p) {
    int ans = p++;
    if (p == N) p = 0;
    return ans;
}
bool spfa() {
    static int d[N];
    for (int i = 0; i < n; i++) {
        a[i].c = INF;
        a[i].ln = a[i].le = -1;
        a[i].d = false;
    }
    int p = 0, q = 0;
    d[add(q)] = s;
    a[s].d = true;
    a[s].c = 0;
    while (p != q) {
        int u = d[add(p)];
        for (int j = a[u].fe; j != -1; j = b[j].ne) {
            int v = b[j].t;
            if (b[j].f > 0 && b[j].c + a[u].c < a[v].c) {
                a[v].c = a[u].c+b[j].c;
                a[v].ln = u;
                a[v].le = j;
                if (a[v].d == false) {
                    a[v].d = true;
                    d[add(q)] = v;
                }
            }
        }
        a[u].d = false;
    }
    if (a[t].c == INF) return false;
    p = INF;
    q = 0;
    for (int i = t; i != s; i = a[i].ln) {
        d[q++]=i;
        if (p > b[a[i].le].f) p = b[a[i].le].f;
    }
    flow += p;
    for (int i = q - 1; i >= 0; i--) {
        int j = a[d[i]].le;
        cost += b[j].c * p;
        b[j].f -= p;
        b[j ^ 1].f += p;
    }
    return true;
}
void flow() {
    while (spfa());
}
```

- 强连通分量

```
void clear(int n) {
    for (int i = 0; i < n; i++) {
        a[i].fe = a[i].scc = a[i].dfn = a[i].low = -1;
        a[i].num = 0;
        a[i].instack = false;
```

```
    }
    p = 0;
}
void tarjan(int u) {
    a[u].dfn = a[u].low = idx++;
    a[u].instack = true;
    stk[p++] = u;
    for (int j = a[u].fe; j != -1; j = b[j].ne) {
        int v = b[j].t;
        if (a[v].dfn == 0) {
            tarjan(v);
            a[u].low = min(a[u].low, a[v].low);
        } else if (a[v].instack) {
            a[u].low = min(a[u].low, a[v].dfn);
        }
    }
    if (a[u].low == a[u].dfn) {
        while (stk[--p] != u) {
            a[stk[p]].instack = false;
            a[stk[p]].scc = u;
            a[u].num++;
        }
        a[u].instack = false;
        a[u].scc = u;
        a[u].num++;
    }
}
void demo() {
    idx = p = 0;
    for (int i = 0; i < n; i++) if (a[i].dfn == -1) tarjan(i);
}
```

- **割点、割边**
- **二维几何基础**

```
double dmul(Point a, Point b) { //点积
    return a.x * b.x + a.y * b.y;
}
double xmul(Point a, Point b) { //叉积，大于 0 表示 b 在 a 的逆时针方向
    return a.x * b.y - a.y * b.x;
}
double xmul(Point a, Point b, Point c) { //a->b 与 a->c 的叉积
    return (b.x - a.x) * (c.y - a.y) - (b.y - a.y) * (c.x - a.x);
}
int quadrant(Point a) {//象限号，原点为 0，从 x 轴开始顺时针为 1 至 8，第四象限为 8
    const int ans[3][3] = {{4, 3, 2}, {5, 0, 1}, {6, 7, 8}};
    return ans[1 - sig(a.y)][sig(a.x) + 1];
}
bool cmpp(Point a, Point b) { //极角排序
    int p = quadrant(a), q = quadrant(b);
    if (p != q) return p < q;
    double x = xmul(a, b);
    if (sig(x)) return x > 0;
    return square(a) < square(b);
}
Point rot(Point a) { //逆时针旋转 90 度
    return Point(-a.y, a.x);
}
double alpha(Point a, Point b) { //向量 b 在向量 a 的逆时针多少度
    return atan2(xmul(a, b), dmul(a, b));
}
Point rot(Point a, double p, Point b = Point(0,0)) { //点 a 绕点 b 逆时针旋转 p
    Point t1 = a - b, t2 = Point(cos(p), sin(p));
    return b + Point(t1.x * t2.x - t1.y * t2.y, t1.x * t2.y + t1.y * t2.x);
}
void regular(Line a) { //整理直线，使得直线的极角属于范围(-PI/2,PI/2]
    int x = sig(a.p.x - a.q.x), y = sig(a.p.y - a.q.y);
    if (x == 1 || (x == 0 && y == 1)) swap(a.p, a.q);
}
bool isParallel(Line a, Line b) { //判断是否平行
    return sig(xmul(a.q - a.p, b.q - b.p)) == 0;
}
bool inSameLine(Line a, Line b) { //判断是否共线，要求 ab 平行
    return sig(xmul(a.q - a.p, a.q - b.p)) == 0;
}
bool isVertical(Line a, Line b) { //判断是否垂直
    return sig(dmul(a.q - a.p, b.q - b.p)) == 0;
}
Point cross(Line a, Line b) { //直线 a 与 b 的交点，要求 ab 不平行
    double t1 = xmul(a.p, a.q, b.p), t2 = -xmul(a.p, a.q, b.q);
    return (b.p * t2 + b.q * t1) / (t1 + t2);
}
bool cmpLine(Line a, Line b) { //按直线的方向排序
    return cmpp(a.q - a.p, b.q - b.p);
}
bool inLine(Point a, Line b) { //判断点是否在直线上
    return sig(xmul(a - b.p, a - b.q)) == 0;
}
Point projection(Point a, Line b) { //点在直线上的投影
    Point tmp = b.q - b.p;
    return b.p + tmp * dmul(a - b.p, tmp) / square(tmp);
}
double disLine(Point a, Line b) { //点到直线的距离
    return abs(xmul(a - b.p, a - b.q)) / abs(b.p - b.q);
}
double disSeg(Point a, Line b) { //点到线段距离
    Point x = b.q - b.p, y = a - b.p, z = a - b.q;
    if (sig(dmul(x, y)) <= 0) return abs(y);
    if (sig(dmul(x, z)) >= 0) return abs(z);
    return abs(xmul(y, z)) / abs(x);
}
bool inSeg(Point a, Line b) { //判断点是否在线段上，端点返回 true
    if (sig(xmul(a - b.p, a - b.q)) != 0) return false; //不在直线上
    if (sig(dmul(a - b.p, a - b.q)) > 0) return false;
    //不在线段内，若为端点，则等于 0
```

```
    return true;
}
Line perpBis(Line a) { //线段 a 的中垂线
    Point t1 = (a.p + a.q) / 2, t2 = rot(a.q - a.p);
    return Line(t1, t1 + t2);
}
bool hasCross(Line a, Line&b) { //线段之间是否有交点，重合和端点相交返回 false
    if (sig(xmul(a.p, a.q, b.p)) * sig(xmul(a.p, a.q, b.q)) >= 0)
        return false; //b 的两个端点在 a 的同侧
    if (sig(xmul(b.p, b.q, a.p)) * sig(xmul(b.p, b.q, a.q)) >= 0)
        return false; //a 的两个端点在 b 的同侧
    return true;
}
```

- **圆相关**

```
int cross(Line a, Circle b, Point &ans1, Point &ans2) { //求出直线与圆的交点
    double t1 = a.q.x - a.p.x, t2 = a.p.x - b.c.x;
    double t3 = a.q.y - a.p.y, t4 = a.p.y - b.c.y;
    double k1 = t1 * t1 + t3 * t3, k2 = 2 * (t1 * t2 + t3 * t4);
    double k3 = t2 * t2 + t4 * t4 - b.r * b.r;
    double d = k2 * k2 – 4 * k1 * k3;
    if (sig(d) < 0) return 0; //无交点
    if (sig(d) == 0) d = 0; else d = sqrt(d);
    ans1 = a.p + (a.q – a.p) * (-k2 + d) / (2 * k1);
    ans2 = a.p + (a.q - a.p) * (-k2 - d) / (2 * k1);
    return sig(d) + 1;
}
int cross(Circle a, Circle b, Point &ans1, Point &ans2) { //求出两圆的交点
    double d = abs(a.c - b.c);
    if (sig(d) == 0)
        if (sig(a.r - b.r) == 0) return -1; //重合，无数个交点
        else return 0;
    if (sig(a.r + b.r - d) < 0) return 0; //相离
    if (sig(abs(a.r - b.r) - d) > 0) return 0; //内含
    double p1 = alpha(b.c-a.c);
    double p2 = acos(legal((a.r * a.r + d * d - b.r * b.r) / (2 * a.r * d)));
    ans1 = get(a, p1 + p2);
    ans2 = get(a, p1 - p2);
    return sig(p2) + 1;
}
int tangent(Point a, Circle b, Point &ans1, Point &ans2) { //点与圆的两个切点
    double d = abs(a - b.c);
    if (sig(d - b.r) < 0) return 0; //在圆内，无交点
    double al1 = alpha(a - b.c);
    double al2 = acos(legal(b.r / d));
    ans1 = get(b, al1 + al2);
    ans2 = get(b, al1 - al2);
    return sig(al2) + 1;
}
int tangent(Circle a, Circle b, Line &ans1, Line &ans2,
        Line &ans3, Line &ans4) { //求出两圆的共切线，返回公切线个数
    if (a.r < b.r) swap(a, b);
    Point t = b.c - a.c;
    double d = abs(t), al1 = alpha(t), al2;
    if (a.c == b.c && sig(a.r - b.r) == 0) return -1; //重合，无数条公切线
    if (sig(a.r - b.r - d) > 0) return 0; //内含
    if (sig(a.r - b.r - d) == 0) { //内切
        Point p = get(a, al1);
        ans1 = ans2 = Line(p, p + rot(t));
        return 1;
    }
    al2 = acos(legal((a.r - b.r) / d));
    ans1 = Line(get(a, al1 + al2), get(b, al1 + al2));
    ans2 = Line(get(a, al1 - al2), get(b, al1 - al2)); //两条外公切线
    if (sig(a.r + b.r - d) > 0) return 2; //相交
    if (sig(a.r + b.r - d) == 0) { //外切
        Point p = get(a, al1);
        ans3 = ans4 = Line(p, p + rot(t));
        return 3;
    }
    al2 = acos(legal((a.r + b.r) / d));
    ans3 = Line(get(a, al1 + al2), get(b, al1 + PI + al2));
    ans4 = Line(get(a, al1 - al2), get(b, al1 + PI - al2));
    return 4; //相离
}
```

- **凸包**

```
int convexHull(Point a[], int n, Point ans[]) { //ans[]的大小要为 N+1
    sort(a, a + n, cmpxy);
    n = unique(a, a + n) - a;
    if (n == 1) ans[0] = a[0];
    if (n <= 1) return n;
    int m = 0;
    for (int i = 0; i < n; i++) { //下半圆
        while (m > 1 && sig(xmul(ans[m - 2], ans[m - 1], a[i])) <= 0) m--;
        //去掉等号则允许边上的点
        ans[m++] = a[i];
    }
    int mm = m;
    for (int i = n - 2; i >= 0; i--) { //上半圆
        while (m > mm && sig(xmul(ans[m - 2], ans[m - 1], a[i])) <=0 ) m--;
        //去掉等号则允许边上的点
        ans[m++] = a[i];
    }
    return m - 1;
}
int convexCut(Point a[], int n, Line b, Point ans[]) { //被直线 b 切割，留左手
    int m = 0;
    for (int i = 0; i < n; i++) {
        Point &p = a[i], &q = a[(i + 1) % n];
        int t1 = sig(xmul(b.p, b.q, p)), t2 = sig(xmul(b.p, b.q, q));
        if (t1 >= 0) ans[m++] = p;
        if (t1 * t2 < 0) ans[m++] = cross(Line(p, q), b);
    }
```

```
    m = unique(ans, ans + m) - ans; //一条线段被切割时会多出一个点来
    return m;
}
double convexDiameter(Point a[], int n) {
//旋转卡壳求凸包上的最远点对，要求凸包为逆时针，且边上没有点
    if (n == 1) return 0;
    if (n == 2) return abs(a[0] - a[1]);
    double ans = 0;
    for (int i = 0, j = 1; i < n; i++) {
        int ii = (i + 1) % n, jj = j, t;
        do { //求出所有的对踵点，可能有重复
            j = jj;
            jj = (j + 1) % n;
            t = sig(xmul(a[ii] - a[i], a[jj] - a[j]));
            if (t <= 0) ans = max(ans, square(a[i] - a[j])); //对踵点
            if (t == 0) ans = max(ans, square(a[i] - a[jj])); //对踵点
        } while (t > 0);
    }
    return sqrt(ans);
}
bool inPolygon(Point a[], int n, Point b) { //点在多边形内，边界返回 false
    int ans = 0;
    for (int i = 0; i < n; i++) {
        Point &p = a[i], &q = a[(i + 1) % n];
        if (inSeg(b, Line(p, q))) return false; //判断边界返回 false
        int k = sig(xmul(q - p, b - p));
        if (k > 0 && p.y <= b.y + eps && q.y > b.y + eps) ans++;
        if (k < 0 && q.y <= b.y + eps && p.y > b.y + eps) ans--;
    }
    return ans;
}
int halfPlaneIntersection(Line a[], int n, Point ans[]) {
//半平面交，保留每条直线的左手边，求出的凸包在 ans 中，若凸包已退化，则返回 0
//要求必须有边界，若无边界则手动添加边界
    sort(a, a + n, cmpLine);
    static Line b[N];
    static Point c[N]; //c[i]为 b[i]与 b[i+1]的交点
    int l = 0, r = 0;
    b[0] = a[0];
    for (int i = 1; i < n; i++) {
        while (l < r && sig(xmul(a[i].p, a[i].q, c[r - 1])) <= 0) r--;
        while (l < r && sig(xmul(a[i].p, a[i].q, c[l])) <= 0) l++;
        b[++r] = a[i];
        if (sig(xmul(a[i].q - a[i].p, b[r - 1].q - b[r - 1].p)) == 0) {
            r--;
            if (sig(xmul(b[r].p, b[r].q, a[i].p)) > 0) b[r] = a[i];
        }
        if (l < r) c[r - 1] = cross(b[r], b[r - 1]);
    }
    while (l < r && sig(xmul(b[l].p, b[l].q, c[r - 1])) <= 0) r--;
    if (r – l <= 1) return 0; //凸包已退化
    c[r] = cross(b[l], b[r]);
    int m = 0;
    for (int i = l; i <= r; i++) ans[m++] = c[i];
    return m;
}
```

- **V 图**
- **三角形**
- **四边形**
- **平面定理**

多边形重心：三角剖分后，以面积为权值求各个重心的加权平均

皮克定理：格点多边形面积 = 内部格点数 + 边上格点数 / 2 - 1

欧拉定理：对于一个平面图/凸多面体，顶点个数 + 面数 - 边数 = 2

- **三维几何体**
- **高维球**
- **三维几何**
- **扩展欧几里得**

```
void exgcd(int a, int b, int &x, int &y) { // 求解 ax + by = gcd(a, b)
    if (b == 0) {
        x = 1; y = 0;
    } else {
        int k = a / b, c = a % b, p, q;
        exgcd(b, c, p, q);
        x = q; y = -k * q + p;
    }
}
// 对于乘法逆元，求 ax + mody = 1 即可，x 即为 a 的逆元，x 在[-mod, mod)范围内
```

- **线性求乘法逆元（mod 为质数）**

```
inv[1] = 1;
inv[i] = mod -(long long)mod / i * inv[mod % i] % mod;
```

- **线性筛素数**

```
for (int i = 2; i < N; i++) {
    if (mpf[i] == 0) mpf[i] = prime[pn++] = i;
    for (int j = 0; j < pn && i * prime[j] < N && prime[j] <= mpf[i]; j++)
        mpf[i * prime[j]] = prime[j];
}
```

- **高斯消元**

每列留下绝对值最大的元素，可以减少精度丢失

- **欧拉函数**

$\varphi(n)$为小于等于n中与n互质的数的个数，若n,m互质，则$\varphi(nm) = \varphi(n)\varphi(m)$

- 大素数判定
- 大数分解
- 莫比乌斯反演
- 波利亚
- Java 大数

```
import java.math.BigInteger;
import java.util.Scanner;
Scanner in = new Scanner(System.in);
int n = in.nextInt();
BigInteger a = in.nextBigInteger();
System.out.println(a);
```

- 常用大素数

1,000,000,007 100,000,007 10,000,019 1,000,003 100,003 10,007 1,019 103

- 约数个数

范围	个数	范围	个数	范围	个数	范围	个数	范围	个数
10^3	32	10^5	128	10^7	448	10^9	1,344	**int32**	1,600
10^4	64	10^6	240	10^8	768	10^{18}	103,680	**int64**	161,280

- 浮点数求和(Kahan Summation)

```
double y = a[i] - c;
double t = ans + y;
c = (t - ans) - y;
ans = t;
```

- Hash

```
int get(int l, int r) {
    int tmp = (long long)h[l - 1] * p[r - l + 1] % mod;
    return (h[r] - tmp + mod) % mod;
}
bool equal(int a, int b, int l) { //a 开始的和 b 开始的长为 l 的字符串是否相同
    if (l == 0) return true;
    return get(a, a + l - 1) == get(b, b + l - 1);
}
void init() {
    p[0] = 1;
    for (int i = 1; i < N; i++) p[i] = (long long)p[i - 1] * 26 % mod;
    h[0] = 0;
    for (int i = 1; i <= n; i++)
        h[i] = ((long long)h[i - 1] * 26 + s[i] - 'a') % mod;
}
```

- KMP

```
//next[i] == j 表示满足以下条件的最大的 j
//s2[0..j-1]与 s2[i-j..i-1]相同，且 s2[j]与 s2[i]不同，若不存在，则 next[i] = -1
int kmp(char s1[], char s2[], int next[]) {
    int i, j = 0, k = -1, ans = 0;
    next[0] = -1;
    while (s2[j]!='\0') {
        while (k != -1 && s2[j] != s2[k]) k = next[k];
        j++; k++;
        if (s2[j] != s2[k]) next[j] = k;
        else next[j] = next[k];
    }
    i = j = 0;
    while (s1[i] != '\0') {
        if (j != -1 && s2[j] == '\0') {
            ans++;
            j = 0;
            // 如果要求可重复的 s2，则不令 j=0，而是和平时一样处理 j
        } else {
            while (j != -1 && s1[i] != s2[j]) j = next[j];
            i++; j++;
        }
    }
    if (s2[j] == '\0') ans++;
    return ans;
}
```

- exKMP

```
void exkmp(char s1[], char s2[], int next[], int ex[]) {
    int i, j, p;
    for (i = 0, j = 0, p = -1; s1[i] != '\0'; i++, j++, p--) {
        if (p == -1) {
            j = 0;
            do p++; while (s1[i + p] != '\0' && s1[i + p] == s2[j + p]);
            ex[i] = p;
        } else if (next[j] < p) ex[i] = next[j];
        else if (next[j] > p) ex[i] = p;
        else {
            j = 0;
            while (s1[i + p] != '\0' && s1[i + p] == s2[j + p]) p++;
            ex[i] = p;
        }
    }
    ex[i] = 0;
}
void demo() {
    nxt[0] = 0;
    exkmp(s2 + 1, s2, nxt, nxt + 1);
    exkmp(s, s2, nxt, ex);
}
```

- Manacher

```
// s[i] + a[i] == s[i] - a[i]
void manacher(char s[], int ls, int a[]) {
    a[0] = 0;
    for (int i = 0, j; i < ls; i = j) {
        while (i - a[i] > 0 && s[i + a[i] + 1] == s[i - a[i] - 1]) a[i]++;
        for (j = i + 1;
                j <= i + a[i] && i - a[i] != i + i - j - a[i + i - j]; j++)
```

```
            a[j] = min(a[i + i - j], i + a[i] - j);
        a[j] = max(i + a[i] - j, 0);
    }
}
void demo() {
    ls = strlen(s);
    for (int i = 0; i < ls; i++) {
        ss[i + i + 1] = s[i];
        ss[i + i + 2] = '\0';
    }
    ls = ls * 2 + 1;
    ss[0] = ss[ls] = '\0';
    manacher(ss, ls, a);
}
```

- **AC 自动机**
- **后缀自动机**
- **回文自动机**
- **后缀数组（倍增）**

```
inline bool equal(int *r, int p, int q, int l) {
    return r[p] == r[q] && r[p + l] == r[q + l];
}
void da(int r[], int sa[], int n, int m) {
    static int wa[N], wb[N], wv[N], ws[N];
    int *x = wa, *y = wb;
    for (int i = 0; i < m; i++) ws[i] = 0;
    for (int i = 0; i < n; i++) ws[x[i] = r[i]]++;
    for (int i = 1; i < m; i++) ws[i] += ws[i - 1];
    for (int i = n - 1; i >= 0; i--) sa[--ws[x[i]]] = i;
    for (int j = 1, p = 1; p < n; j *= 2, m = p) {
        p = 0;
        for (int i = n - j; i < n; i++) y[p++] = i;
        for (int i = 0; i < n; i++) if (sa[i] >= j) y[p++] = sa[i] - j;
        for (int i = 0; i < n; i++) wv[i] = x[y[i]];
        for (int i = 0; i < m; i++) ws[i] = 0;
        for (int i = 0; i < n; i++) ws[wv[i]]++;
        for (int i = 1; i < m; i++) ws[i] += ws[i - 1];
        for (int i = n - 1; i >= 0; i--) sa[--ws[wv[i]]] = y[i];
        swap(x, y);
        x[sa[0]] = 0;
        p = 1;
        for (int i = 1; i < n; i++) {
            x[sa[i]] = (equal(y, sa[i - 1], sa[i], j))? p - 1 : p++;
        }
    }
}
void calh(int r[], int sa[], int h[], char s[], int n) {
    for (int i = 0, k = 0; i < n; i++) {
        if (k > 0) k--;
        for (int j = sa[r[i] - 1]; s[i + k] == s[j + k]; k++);
        h[r[i]] = k;
    }
}
void demo() {
    for (int i = 0; i < n; i++) r[i] = s[i] - 'a' + 1;
    r[n] = 0;
    da(r, sa, n + 1, 27);
    for (int i = 1; i <= n; i++) r[sa[i]] = i;
    calh(r, sa, h, s, n);
    calst(lg, st, h, n + 1);
}
```

- **ST**

```
void calst(int lg[], int st[][K], int h[], int n) {
    for (int i = 1; i <= n; i++) st[i][0] = h[i];
    for (int k = 1; k < K; k++) {
        for (int i = 0; i + (1 << k) <= n; i++) {
            st[i][k] = min(st[i][k - 1], st[i + (1 << (k - 1))][k - 1]);
        }
    }
}
inline int get(int l, int r) {
    int k = lg[r - l + 1];
    return min(st[l][k], st[r - (1 << k) + 1][k]);
}
void init(int lg[]) {
    lg[0] = -1;
    for (int i = 1; i < N; i++)
        if (i & i - 1) lg[i] = lg[i - 1];
        else lg[i] = lg[i - 1] + 1;
}
```

- **Dancing Links**