1 TODO

- 线段相交
- SA[M]
- 割点/桥

2 Formula

- Eular Formular: $|V| - |E| + |F| = 2$
- $|A \cup B \cup C| = |A| + |B| + |C| - |A \cap B| - |A \cap C| - |B \cap C| + |A \cap B \cap C|$
- $|\bar{A} \cap \bar{B} \cap \bar{C}| = |\Omega| - |A| - |B| - |C| + |A \cap B| + |A \cap C| + |B \cap C| - |A \cap B \cap C|$
- Catalan Number: $C_n = (4n-2)/(n+1)C_{n-1}$
- 记得树可以通过 dfs 序列化
- Floyd 算法别忘了设置 dist[i][i] = 0
- 想着比赛的时候可以打表
- 初始化一定不要忘记
- 提交时记得把所有的调试信息都删掉
- 想着可以用二分的方法，把问题转化为判定问题。
- 对于几何问题，没想法就先动手画画图，别上来就解析法。
- 数组一定要开的足够打大，能用 LL 就别用 int
- treedp 尽量别一个人瞎想
- 数位 dp 一定要写暴力 check
- 最大 (极大) 独立集 + 最小 (极小)(点) 覆盖集 = V
- 最大 (极大) 团 = 补图的最大 (极大) 独立集
- 二分图的最大独立集 = V - 二分图的最大匹配
- 二分图的最大 (点权) 独立集 = SUM - 二分图的最佳匹配
- 二分图的最小 (边权) 覆盖 = 二分图的最佳匹配
- 二分图的最小 (点权) 覆盖 = 最小割 (X-Y 之间的边设为 inf)
- 二分图的最小覆盖 = 二分图的最大匹配
- 注意求递推式的时候可能要用到二项式定理, 如求 $\Sigma_{i-1}^{n} i^k k^i (k \le 50)$

3 Polya

Burnside 引理: 设 $G = \{p_1, p_2, ..., p_g\}$ 是目标集 $[1, n]$ 上的置换群,G 将 $[1, n]$ 分成 L 个等价类。设 $c_1(p_k)$ 是在置换 p_k 作用下不动点的个数 (也就是长度为 1 的循环的个数), 则等价类的个数

$$L = \frac{1}{|G|}\Sigma_{i=1}^{g} c_1(p_i)$$

Pòlya 定理: 设 $G = \{a_1, a_2, ..., a_{|G|}\}$ 是 $N = \{1, 2, .., N\}$ 上的置换群, 现用 m 种颜色对这 N 个点染色, 则不同的染色方案数为:

$$S = \frac{(m^{c_1} + m^{c_2} + \ldots + m^{c_{|G|}})}{|G|}$$

常见置换的循环数

- 旋转:n 个点顺时针 (逆时针) 旋转 i 个位置的置换, 循环数为 $gcd(n, i)$
- 翻转:
  - n 为偶数时: 对称轴不过顶点, 循环数为 $n/2$ 对称轴过顶点: 循环数为 $n/2 + 1$
  - n 为奇数时: 循环数为 $(n+1)/2$

立方体面用 k 种颜色涂色

$$\frac{8k^2 + 12k^3 + 3k^4 + k^6}{24}$$

4 Edit Esp

```
int main() {
    static const int stksz = 10000000;
    static int stk[stksz], espbak;
    __asm__ __volatile__ ( "movl␣%%esp,␣%0\n\tmovl␣%1,␣%%esp\n\t" : "=g"(espbak
        ) : "g"(stk+stksz-1)  );
    solve();
    exit(0);
}
```

5 Java Header

```
class Task {
    void solve( int ri, InputReader in, PrintWriter out ) {
        BigDecimal a = new BigDecimal("23213432.2142143");
        a = a.round( new MathContext(10, RoundingMode.HALF_UP) );
        out.println( a.toPlainString() );
    }
}

class InputReader {
    private BufferedReader reader;
    private StringTokenizer tokenizer;

    public InputReader( InputStream sm ) {
        reader = new BufferedReader( new InputStreamReader(sm) );
        tokenizer = null;
    }

    public String next() {
        while ( tokenizer == null || !tokenizer.hasMoreTokens() ) {
            try {
                tokenizer = new StringTokenizer( reader.readLine() );
            } catch( IOException e ) {
                throw new RuntimeException(e);
            }
        }
        return tokenizer.nextToken();
    }
}
```

6 ufset

```
int find(int x) {
    int r = x, totdist = 0;
    for (; r != fa[r]; r = fa[r])
        totdist += dist[r];
    for (int y, dx; x != r; totdist -= dx, x = y) {
        y = fa[x]; dx = dist[x];
        fa[x] = r; dist[x] = totdist;
    }
    return r;
}
```

7 leftist

```
namespace LeftistTree {
    const int maxn = 10000;

    typedef struct LeftistHeapNode {
        int key, dist;
        LeftistHeapNode *left, *right, *father;
    } *LeftistHeap;

    LeftistHeap null = new LeftistHeapNode;

    void init() { null->dist = -1; }

    LeftistHeap newnode(int key, int dist = 0) {
        LeftistHeap p = new LeftistHeapNode;
        p->left = p->right = p->father = null;
        p->key = key; p->dist = dist; return p;
    }

    LeftistHeap merge(LeftistHeap a, LeftstHeap b) {
        if (a == null) return b; else if (b == null) return a;
        if (a->key < b->key) swap(a, b); // 小根堆>:; 大根堆<:
        a->right = merge(a->right, b);
        a->right->father = a;
        if (a->left->dist < a->right->dist) swap(a->left, a->right);
        a->dist = a->right->dist + 1;
        return a;
    }

    void deletemin(LeftistHeap &p) {
        p = merge(p->left, p->right);
    }

    voi deletenode(LeftistHeap p) {
        if (p == null) return;
        LeftistHeap f = p->father, q = merge(p->left, p->right);
        q->father = f;
        if (f->left == p) f->left = q; else f->right = q;
        for (;f != null; f = f->father) {
            if ( f->left->dist < f->right->dist ) swap(f->left, f->right);
            if ( f->right->dist + 1 == f->dist ) return;
            f->dist = f->right->dist + 1;
        }
    }

    LeftistHeap Q[maxn * 2], *Qt, *Qb;

    LeftistHeap build(LeftistHeap Q[], LeftistHeap *Qb, LeftistHeap *Qt) { //O(
        n)
        if ( Qb == Qt ) return null;
        for (; Qb + 1 != Qt; Qb += 2) {
            *Qt++ = merge(Qb[0], Qb[1]);
        }
        return *Qb;
    }
} //end namespace

/** 表示一个集合，可以快速实现以下功能
  * 取得集合的中位数如果有个，则去较小的那个(2)
  * 合并两个集合
  */
strut MedianSet {
    int n;
    LeftistTree::leftistHeap s; //大根堆

    MediaSet(int key) {
        s = LeftistTree::newnode(key); n = 1;
    }

    int getMedia() {
        return s->key;
    }

    MediaSet &merge(MediaSet b) {
        this->s = LeftistTree::merge(this->s, b.s);
        if ( this->n % 2 && b.n % 2 )
            LeftistTree::deletemin(this->s);
        this->n += b.n;
        return *this;
    }
};
```

8 CartesianTree

```
struct Node {
    int /*TREE*/ key, /*HEAP*/ val, ind;
    Node *father, *left, *right;
} node[maxn], *null;

struct lessKey {
    bool operator()(const Node *x, const Node *y) const {
        return x->key < y->key;
    }
};

Node *build(Node node[], int n) {
    static Node *ptr[maxn];
    Node *r, *last, *cur;
    rep(i, n) ptr[i] = node + i;
    sort(ptr, ptr + n, lessKey());
    r = last = ptr[0];
    r->father = r->left = r->right = null;
    for (int i = 1; i < n; ++i) {
        cur = ptr[i];
        cur->father = cur->left = cur->right = null;
        if (cur->value < r->value) {
            cur->left = r; r->father = cur;
            last = r = cur;
        } else {
            while (cur->value <= last->value) last = last->father;
            cur->left = last->right; last->right->father = cur;
            last->right = cur; cur->father = last;
            last = cur;
        }
    }
    return r;
}

void poj2201 {
    null = new Node; null->ind = 0;
    for (int n; cin >> n; ) {
        rep(i, n) {
            scanf("%d%d", &node[i].key, &node[i].value);
            node[i].ind = i + 1;
        }
        build(node, n); puts("YES");
        rep(i, n) {
            Node *p = node + i;
            printf("%d␣%d␣%d\n", p->father->ind, p->left->ind, p->right->ind);
        }
    }
}
```

9 kmp & z

```
//next[i]: x[i-next[i]...i-1] = x[0..next[i]-1]
void preMP(const char x[], int m, int next[]) {
    int i, j;
    i = next[0] = -1; j = 0;
    while (j < m) {
        while (i > -1 && x[i] != x[j]) i = next[i];
        next[++j] = ++i;
    }
}

int kmp(const char x[], int m, const char y[], int n) {
    int i = 0, j = 0, ret = 0;
    preMP(x, m, next);
    while(j < n) {
        while (i > -1 && x[i] != y[j]) i = next[i];
        ++i; ++j;
        if (i >= m) {
            //OUTPUT(j - i)
            ret++; i = next[i];
        }
    }
    return ret;
}

void prez(const char x[], int m, int next[]) {
    int j, k = 1, r = 0; next[0] = m;
    for (int i = 1; i < m; ++i) {
        if ( i + next[i-k] < r ) {
            next[i] = next[i-k];
        } else {
            for (j = max(r-i, 0); i + j < m && x[i+j] == x[j]; ++j);
            next[i] = j; k = i; r = i + j;
        }
    }
}

//next[i]: lcp of x[i..m-1] and x[0..m-1]
//ext[i]:  lcp of y[i..n-1] and x[0..m-1]
void z(const char x[], int m, const char y[], int n, int next[], int ext[]) {
    int k = 0, r = 0, j;
    prez(x, m, next); next[0] = 0;
    for (int i = 0; i < n; ++i) {
        if ( i + next[i-k] < r ) {
            ext[i] = next[i-k];
        } else {
            for (j = max(r-i, 0); j < m && i + j < n && x[j] == y[i+j]; ++j);
            ext[i] = j; k = i; r = i + j;
        }
    }
}

int minCircularDenote(char *s, int n){ //@Return: the index of minimal denote
    strncpy(s + n, s, n); //Notice: s will be double
    int i = 0, j = 1;
    for (;;){
        int k = 0;
        while (k < n && s[i+k] == s[j+k]) k++;
        if (k == n) return i;
        if (s[i+k] > s[j+k]) i = max(i+k+1, j+1); else j = max(j+k+1, i+1);
        if (i == j) j++; else if (i > j) swap(i, j);
        if (j >= n) return i;
    }
}
```

10 manacher

```
// s[i] is center -> p[i*2+2]
// s[i,i+1] is center -> p[i*2+3]
// p[j] -> t[] center at j, s[] center at (j-1)*0.5 a.k.a s.substr( (j-1-p[j])
    /2, p[j] )
void Manacher(char s[maxn], int p[maxn*2+5]){
    static char t[maxn * 2 + 5];

    int n = 0;
    t[n++] = '^'; // of importance
    for (char *is = s; *is; is++ )
        t[n] = '#', t[n+1] = *is, n += 2;
    t[n++] = '#'; t[n] = 0;

    int c = 0, r = 0; p[0] = 0;
    for (int i = 1; i < n; i++){
        int j = 2 * c - i;
        p[i] = r > i ? min(r-i, p[j]) : 0;
        while ( t[ i+1+p[i] ] == t[ i-1-p[i] ] )
            p[i]++;
        if ( i+p[i] > r )
            c = i, r = i + p[i];
    }
}
```

11 dfa

```
const int maxc = 200;
int hash[255];

struct node {
    node *fail, *next[4];
    int flag, ind, is_terminal, length;
    void *operator new(size_t);
} *root, *null, pool[maxc], *pooltop;

typedef node *pnode;
pnode Q[maxc], *Qb, *Qt;

void *node::operator new(size_t) {
    memset(pooltop, 0, sizeof pooltop);
    pooltop->length = -1; return pooltop++;
}

void insert(char *s, int iter) {
    node *p = root;
    int L = strlen(s);
    for (int j; *s; p = p->next) {
        j = hash[*s++];
        if (!p->next[j]) p->next[j] = new node;
    }
    p->flag |= 1 << iter;
    p->is_terminal = 1;
    p->length = max(p->length, L);
}

void build() {
    pnode p, q; Qb = Qt = Q;
    *Qt = root; root->ind = Qt - Q; Qt++;
    for (int j = 0; j < 4; ++j) //edit 4
        null->next[j] = root;
    root->fail = null;
    for(;Qt != Qb;) {
        p = *Qb++;
        for (int j = 0; j < 4; ++j) {
            if (p->next[j]) {
                p->next[j]->fail = p->fail->next[j];
                *Qt = p->next[j]; p->next[j]->ind = Qt - Q; Qt++;
                chkmax( p->next[j]->length, p->next[j]->fail->length );
                p->next[j]->flag |= p->next[j]->fail->flag;
            } else {
                p->next[j] = p->fail->next[j];
            }
        }
    }
}

int dp[2000][200][12];
const int p = 1000000009;

void add(int &a, int b) { ((a += b) >= p) ? a -= p : a; }

int main() {
    hash['A'] = 0; hash['T'] = 1; hash['G'] = 2; hash['C'] = 3;
    int n, m; char s[100];
    while (cin >> n >> m) {
        pooltop = pool; null = new node; root = new node;
        rep(i, m) {
            scanf("%s", s); insert(s, i);
        }
        build();
        int cnt = Qt - Q;
        rep(i, n+1) rep(j, cnt) rep(k, 12) dp[i][j][k] = 0;
#define Next(j, k) (Q[j]->next[k]->ind)
#define Flag(j) (Q[j]->flag)
#define isTerminal(j) (Q[j]->is_terminl)
        dp[0][0][0] = 1;
        rep(i, n+1) rep(j, cnt) rep(L, 10) {
            if (dp[i][j][L]) rep(k, 4) {
                int nj = Next(j, k);
                if (L+1 <= Q[nj]->length) {
                    add(dp[i+1][nj][0], dp[i][j][L]);
                } else {
                    add(dp[i+1][nj][L+1], dp[i][j][L])
                }
            }
        }
    }
    int res = 0;
    rep(j, cnt) add(res, dp[n][j][0]);
    cout << res << endl;
}
```

12 dinic

```
struct Dinic {
    const static int maxn = 5001;
    const static int maxe = 60200;
    const static int inf = 2000000000; // <= maxc * 2

    struct node {
        int b, c; node *next, *anti;
    } *ge[maxn], pool[maxe], *pooltop;
    int dist[maxn], n;

    void init(int n) {
        pooltop = pool; this->n = n; rep(i, n + 1) ge[i] = 0;
    }

    node *_insert(int a, int b, int c) {
        node *p = pooltop++; p->b = b; p->c = c; p->next = ge[a]; return ge[a]
            = p;
    }

    void insert1(int a, int b, int c) {
        node *p = _insert(a, b, c), *q = _insert(b, a, 0);
    p->anti = q; q->anti = p;
}

void insert2(int a, int b, int c) {
    node *p = _insert(a, b, c), *q = _insert(b, a, c);
    p->anti = q; q->anti = p;
}

bool bfs(int s, int t) {
    static int q[maxn], *qt, *qb;
    qt = qb = q; memset(dist, -1, sizeof(dist[0] * (n+1)));
    dist[s] = 0; *qt++ = s;
    for (; qt != qb; ++qb) {
        for (node *p = ge[*qb]; p; p = p->next) {
            if (p->c && dist[p->b] == -1) { //sign(p->c) for real flow
                dist[p->b] = dist[*qb] + 1; *qt++ = p->b;
                if (q->b == t) return true;
            }
        }
    }
    return false;
}

LL dinic(int s, int t) {
    static int pre[maxn];
    static node *cur[maxn], *path[maxn];

    LL tot = 0;
    while ( bfs(s, t) ) {
        memcpy(cur, ge, sizeof(ge[0]) * (n+1));
        for (int i = s; dist[s] != -1; ) {
            if (i == t) {
                int flow = inf;
                for (; i != s; i = pre[i]) flow = min(flow, paht[i]->c);
                tot += flow;
                for (int i = t; i != s; i = pre[i]) {
                    path[i]->c -= flow; path[i]->anti->c += flow;
                }
            }
            for (node *&p = cur[i]; p; p = p->next) {
                int v = p->b;
                if ( p->c && dist[v] == dist[i] + 1 ) { //sign(p->c) for
                    real flow
                    pre[v] = i; path[v] = p; i = v; break;
                }
            }
            if (0 == cur[i]) {
                dist[i] = -1; i = pre[i];
            }
        }
    }
    return tot;
  }
} flow;
```

13 costflow

```
const int maxn = 5001 * 2, maxe = 60200 * 5;
const int inf = 2000000000; // >= maxc * 2

struct node {
    int b, c, w; node *next, *anti;
} *ge[maxn], pool[maxe], *pooltop;

void init(int n) {
    pooltop = pool; rep(i, n + 1) ge[i] = 0;
}

inline node* _insert(int a, int b, int c, int w) {
    node *p = pooltop++; p->b = b; p->c = c; p->w = w; p->next = ge[a]; ge[a] =
        p; return p;
}

inline void insert1(int a, int b, int c, int w) {
    node *p = _insert(a, b, c, w), *q = _insert(b, a, 0, -w); //notice order of
        a and b
    p->anti = q; q->anti = p;
}
```

```
inline void insert2(int a, int b, int c, int w) {
    node *p = _insert(a, b, c, w), *q = _insert(b, a, c, w);
    p->anti = q; q->anti = p;
}

complex<LL> aug(int n, int s, int t, int lim) {
    static int q[maxn], *qt, *qb, inq[maxn], dist[maxn], pre[maxn];
    static node *path[maxn];
#define enq(x) { *qt++ = x; if (q + maxn == qt) qt = q; inq[x] = 1; }
#define deq(x) { x = *qb++; if (q + maxn == qb) qb = q; inq[x] = 0; }
    qb = qt = q; enq(s);
    rep(i, n+1) dist[i] = 0; dist[s] = 0;
    while (qb != qt) {
        int u; deq(u);
        for (node *p = ge[u]; p; p = p->next) {
            if (p->c && dist[p->b] > dist[u] + p->w) { //sign(p->c) for real
                flow
                dist[p->b] = dist[u] + p->w;
                pre[p->b] = u; path[p->b] = p;
                if (!inq[p->b]) enq(p->b);
            }
        }
    }
    LL flow = lim, cost = 0;
    if ( dist[t] == inf ) return complex<LL>(0, 0);
    for(int i = t; i != s; i = pre[i])
        flow = min<LL>(flow, path[i]->c);
    for(int i = t; i != s; i = pre[i]) {
        cost += flow * path[i]->w;
        path[i]->c -= flow; path[i]->anti->c += flow;
    }
    return complex<LL>(flow, cost);
}

complex<LL> mincostmaxflow(int n, int s, int t, int lim = inf) {
    complex<LL> ret = 0, del;
    while ( (del = aug(n, s, t, lim)).real() > 0 ) {
        ret += del; lim -= del.real();
    }
    return ret;
}
```

14 planarmincut

```
typedef pair<int, int> Point;
#define x first
#define y second

const int inf = 1000000000;
const int maxn = 100000 * 2 + 5;
const int maxe = maxn * 2 + 5;

struct Graph {
    vector< pair<int, int> > ge[maxn]; int n;
    void init(int n) {
        this->n = n; rep(i, n) ge[i].clear();
    }
    void ins2(int a, int b, int c) {
        ge[a].pb( mp(b, c) ); ge[b].pb( mp(a, c) );
    }
    LL sssp(int s, int t) {
        set< pair<LL, int> > h;
        static LL dist[maxn];
        for ( int i = 0; i < n; ++i ) {
            dist[i] = i == s ? 0 : inf;
            h.insert( mp(dist[i], i) );
        }
        while ( !h.empty() ) {
            int u = h.begin()->second; h.erase(h.begin());
            for (int k = 0; k < ge[u].size(); ++k) {
                int v = ge[u][k].first, d = ge[u][k].second;
                if ( dist[v] > dist[u] + d ) {
                    h.erase( mp(dist[v], v) );
                    dist[v] = dis[u] + d;
                    h.insert( mp(dist[v], v) );
                }
            }
        }
        return dist[t];
    }
} graph;

struct MaxflowPlanar {
    Point p[maxn];
    int n, ecnt, fcnt;

    struct edge {
        int a, b, c, vis, find;
        edge *prev, *anti;
        double ang;
        edge *next() {
            return anti->prev;
        }
        void init(int a, int b, int c, double ang, edge *anti) {
            this->a = a; this->b = b; this->c = c; this->ang = ang, this->anti
                = anti;
            vis = 0;
        }
    } e[maxe], *ptr[maxe];

    struct Cmp {
        bool operator()(const edge *x, const edge* y) const {
            if ( x->a != y->a ) return x->a < y->a;
            return x->ang < y->ang;
        }
    };

    void init(Point q[], int n) {
        this->n = n; ecnt = fcnt = 0; copy(q, q + n, p);
    }

    void insert2(int a, int b, int c) {
        int dy = p[b].y - p[a].y, dx = p[b].x - p[a].x;
        e[ecnt].init(a, b, c, atan2l(dy, dx), &e[ecnt^1]); ++ecnt;
        e[ecnt].init(b, a, c, atan2l(-dy, -dx), &e[ecnt^1]); ++ecnt;
    }

    LL maxflow() {
        rep(i, ent) ptr[i] = e + i;
        sort( ptr, ptr + ecnt, Cmp() );
        for (int i = 0, j; i < ecnt; i = j) {
            for (j = i + 1; j < ecnt && ptr[i]->a == ptr[j]->a; ++j);
            for (int k = i+1; k < j; ++k) ptr[k]->prev = ptr[k-1];
            ptr[i]->prev = ptr[j-1];
        }
        rep(i, ecnt) {
            if (ptr[i]->vis) continue;
            ptr[i]->find = fcnt; ptr[i]->vis = 1;
            for (edge* p = ptr[i]->next(); p != ptr[i]; p = p->next() )
                p->find = fcnt, p->vis = 1;
            ++fcnt;
        }
        graph.init(fcnt);
        int s = -1, t = -1;
        rep(i, ecnt) {
            if (ptr[i]->c != inf) {
                graph.insert2(ptr[i]->find, ptr[i]->anti->find, ptr[i]->c);
            } else if (s == -1) {
                s = ptr[i]->find, t = ptr[i]->anti->find;
            }
        }
        return graph.sssp(s, t);
    }
} flow;

void solve() {
    int n, m; cin >> n >> m;
    static Point p[maxn];
    int maxY = -inf, minY = inf;
    rep(i, n) {
        scanf("%d%d", &p[i].x, &p[i].y);
        chkmin(minY, p[i].y); chkmax(maxY, p[i].y);
    }
    int s = min_element(p, p + n) - p, t = max_element(p, p + n) - p;
    p[n] = mp(p[s].x-1, maxY+1); p[n+1] = mp(p[t].x+1, maxY+1);
    flow.init(p, n + 2);
    flow.insert2(s, n, inf); flow.insert2(n, n + 1, inf); flow.insert2(n+1, t,
        inf);
    rep(i, m) {
        int a, b, c; scanf("%d%d%d", &a, &b, &c);
        flow.insert2(a-1, b-1, c);
    }
    cout << flow.maxflow() << endl;
}
```

15 kosaraju

```
const int maxn = 500000 + 5;
const int inf = 2000000000;
int visit[maxn], order[maxn], group[maxn], cnt;

struct node { int b; node *next;
} *ge[maxn], *gr[maxn], *gg[maxn], pool[maxn * 10], *pooltop = 0;

void dfs(int u) {
    visit[u] = 1;
    for (node *p = ge[u]; p; p = p->next) {
        int v = p->b; if (!visit[v]) dfs(v);
    }
    order[cnt++] = u;
}

void rfs(int u) {
    visit[u] = 1; group[u] = cnt;
    for (node *p = gr[u]; p; p = p->next) {
        int v = p->b; if (!visit[v]) rfs(v);
    }
}

int scc(int n) {
    cnt = 0; clr(visit, 0, n+1);
    rep(i, n) if (!visit[i]) dfs(i); //may be changed to 1..n
    cnt = 0; clr(visit, 0, n+1);
    for (int i = n-1; i >= 0; --i) {
        int u = order[i]; if (!visit[u]) {
            rfs(u); ++cnt;
        }
    }
    return cnt;
}

#define ins(ge, a, b) {\
    node *_p = pooltop++; _p->b = b; _p->next = ge[a]; ge[a] = _p; }

int val[maxn], dest[maxn];
int group_val[maxn], group_dest[maxn];
int dp[maxn];

int main() {
    for (int n, m, a, b, s;cin >> n >> m;) {
        clr(ge, 0, n); clr(gr, 0, n); clr(dest, 0, n);
        pooltop = pool;
        rep(i, m) {
            scanf("%d%d", &a, &b); --a; --b;
            ins(ge, a, b); ins(gr, b, a);
        }
        rep(i, n) scanf("%d", &val[i]);
        scanf("%d%d", &s, &a); --s;
        rep(i, a) {
            scanf("%d", &b); dest[--b] = true;
        }
        scc(n);
        rep(i, cnt) {
            group_val[i] = group_dest[i] = 0;
            dp[i] = -inf; gg[i] = 0;
        }
        s = group[s];
        rep(i, n) {
            group_dest[ group[i] ] |= dest[i];
            group_val[ group[i] ] += val[i];
            for (node *p = ge[i]; p; p = p->next) {
                if ( group[i] == group[p->b] ) continue;
                ins(gg, group[i], group[p->b]);
            }
        }
        static int q[maxn], *qt, *qb, inq[maxn];
#define enq(x) { *qt++ = x; if (qt == q + maxn) qt = q; inq[x] = 1; }
#define deq(x) { x = *qb++; if (qb == q + maxn) qb = q; inq[x] = 0; }
        clr(inq, 0, cnt); qb = qt = q;
        enq(s); dp[s] = group_val[s];
        while (qb != qt) {
            int u; deq(u);
            for (node *p = gg[u]; p; p = p->next) {
                if (dp[p->b] < dp[s] + group_val[p->adj]) {
                    dp[p->b] = dp[s] + group_val[p->b];
                    if (!inq[p->b]) enq(p->b);
                }
            }
        }
        int maxval = 0;
        rep(i, cnt) if ( group_dest[i] && dp[i] > maxval ) maxval = dp[i];
        cout << maxval << endl;
    }
}
```

16 romberg

```
real f(real x) {
    return exp(-x * x);
}
//O(2 ^ maxitr) function evaluations
real Romberg(real a, real b, real(*f)(real), real eps, int maxitr = 20) {
    real T[maxitr][maxitr];
    for (int i = 0; i < maxitr; ++i) {
        real h = (b - a) / (1 << i), x = a + h, pow = 4;
        T[i][0] = (f(a) + f(b)) / 2;
        for (int j = (1 << i) - 1; j >= 1; x += h, --j) T[i][0] += f(x);
        T[i][0] *= h;
        for (int j = 1; j <= i; pow *= 4, ++j)
            T[i][j] = T[i][j-1] + (T[i][j-1] - T[i-1][j-1]) / (pow-1);
        if (i > 0 && fabs(T[i][i] - T[i - 1][i - 1]) <= eps) return T[i][i];
    }
    return T[maxitr - 1][maxitr - 1];
}
```

17 exgcd

```
T exgcd(T a, T b, T &x, T &y){
    if (b == 0){ x = 1; y = 0; return a; }
    T d = exgcd(b, a % b, y, x);
    y -= a / b * x;
    return d;
}
```

18 isprime

```
bool witness(LL a, LL n) {
    int k = 0;
    LL m = n - 1;
    do {m /= 2; k++; } while (m % 2 == 0);
    LL x = pow_mod(a, m, n);
    if (1 == x) return true;
    for (int i = 0; i < k; x = mul_mod(x, x, n); ++i)
        if (n - 1 == x) return true;
    return false;
}

bool miller_rabin(LL n, int time = 50) {
    if (2 == n || 3 == n || 5 == n || 7 == n) return true;
    if (1 == n || n % 2 == 0 || n % 3 == n || n % 5 == 0 || n % 7 == 0) return
        false;
    while (time--) {
        LL r = rand() % (n-2) + 2;
        if ( gcd(r, n) != 1 || !witness(r % n, n) ) return false;
    }
    return true;
}
```

19 rho

```
LL rho(LL n) {
    LL x, y, d, c;
    for (int k, i;;) {
        c = rand() % (n - 1) + 1;
        x = y = rand() % n;
        k = 2; i = 1;
        do {
            d = gcd( ABS(x - y), n );
            if ( d > 1 && d < n ) return d;
            if ( ++i == k ) y = x, k *= 2;
            x = mul_mod(x, x, n); x = (x + c) % n;
        } while ( x != y );
    }
}
```

20 crt

```
bool crt2(T &dd, T& rr, T d1, T r1, T d2, T r2) {
    T q1, q2, g = exgcd(d1, d2, q1, q2);
    T c = r1 - r2; if (c < 0) c += d1;
    dd = d1 / g * d2;
    if (c % g) { rr = -1; return false; }
    T t = d1 / g;
    q2 *= c / g; q2 %= t; if (q2 <= 0) q2 += t;
    rr = q2 * d2 + r2; if (rr >= dd) rr -= dd;
    return true;
}

bool crt(T& dd, T& rr, T d[], T r[], int n) {
    dd = 1, rr = 0;
    rep(i, n) if (!crt2(dd, rr, dd, rr, d[i], r[i])) return false;
    return true;
}
```

21 log

```
struct Zn {
    const LL n;
    Zn(LL nn) : n(nn) {
    }
    LL eval(LL a) {
        a %= n; return a >= 0 ? a : a + n;
    }
    LL inv(LL a) {
        LL x, y, d = exgcd(a, n, x, y);
        assert(d == 1);
        return eval(x);
    }
    LL log(LL a, LL b);
};

struct Zp : Zn {
    Zp(LL n) : Zn(n) {
    }
    const static int maxsqrtn = 100000 + 5;
    static int id[]; static LL mexp[];

    struct logcmp {
        bool operator()(int a, int b) { return mexp[a] < mexp[b]; }
    };
    LL log(LL a, LL b) { //a ^ x = b
        int m = (int)( ceil( sqrt(n) ) );
        LL v = inv( pow(a, m) );
        id[0] = 0; mexp[0] = 1;
        for (int i = 1; i <= m; ++i) {
            id[i] = i; mexp[i] = mul(mexp[i-1], a);
        }
        stable_sort(id + 1, id + m + 1, logcmp());
        sort(mexp + 1, mexp + m + 1);
        for (int i = 0; i < m; ++i) {
            int j = lower_bound(mexp, mexp + m + 1, b) - mexp;
            if (j <= m && mexp[j] == b) return i * m + id[j];
            b = mul(b, v);
        }
        return -1;
    }
}
```

```
LL Zn::log(LL a, LL b) { //a ^ x = b
    for (int i = 0; i <= 50; ++i) if ( pow(a, i) == b ) return i;
    LL g, d = 1, n = this->n, x = 0;
    while ( (g = gcd(a, n)) > 1 ) {
        if ( b % g ) return -1;
        b /= g; n /= g; d = mul(d, a/g); ++x;
    }
    Zp zp(n); LL ans = zp.log( a, zp.mul(b, zp.inv(d)) );
    return ans == -1 ? -1 : ans + x;
}

int Zp::id[Zp::maxsqrtn]; LL Zp::maxp[ Zp::maxsqrtn ];
```

22 FFT

```
struct Zp {
    const LL mod; const int pri;
    Zp(LL mod, int pri) : mod(mod), pri(pri) {
    }
    Zp(LL mod) : mod(mod), pri( primitive() ) {
    }
    void fft(int n, LL root, LL a[]) {
        for (int m = n; m >= 2; m >>= 1) {
            int mh = m >> 1; LL w = 1;
            for (int i = 0; i < mh; ++i) {
                for (int j = i; j < n; j += m) {
                    int k = j + mh;
                    LL t = sub(a[j], a[k]);
                    a[j] = add(a[j], a[k]);
                    a[k] = mul(w, t);
                }
                w = mul(w, root);
            }
            root = mul(root, root);
        }
        for (int j = 1, i = 0; j < n - 1; ++j) {
            for (int k = n >> 1; k > (i ^= k); k >>= 1);
            if (j < i) swap(a[i], a[j]);
        }
    }
    void dft(const LL a[], int an, LL b[], int n) {
        LL root = pow(pri, mod / n);
        copy(a, a + an, b); fill(b + an, b + n, 0);
        fft(n, root, b);
    }
    void nft(const LL a[], LL b[], int n) {
        LL root = pow(pri, mod / n); root = inv(root);
        copy(a, a + n, b);
        fft(n, root, b);
        LL invn = inv(n);
        rep(i, n) b[i] = mul(b[i], invn);
    }
    int primitive() {
        int n = mod - 1;
        LL p[25], pcnt = 0;
        for (LL i = 2; i * i <= n; ++i) {
            if ( n % i == 0 ) {
                do n /= i; while (n % i == 0);
                p[pcnt++] = i;
            }
        }
        if (n > 1) p[pcnt++] = n;
        for (int g = 2;;++g) {
            int ok = 1; //assert( pow(g, mod-1) == 1 );
            rep(i, pcnt) if ( pow(g, (mod-1)/p[i]) == 1 ) {
                ok = 0; break;
            }
            if (ok) return g;
        }
    }
} zp(0xb1a2bc2edc0001LL, 3);

struct poly {
    const static int maxn = ::maxn * 4 + 5;
    LL a[maxn]; int n;
    template<class T> void init(const T a[], int n) {
        this->n = n; copy(a, a + n, this->a);
    }
    LL eval(LL x) const {
        LL ans = 0;
        for (int i = n - 1; i >= 0; --i)
            ans = zp.add( zp.mul(ans, x), a[i] );
        return ans;
    }
    friend void mul(poly& r, const poly& x, const poly& y) {
        static LL xb[maxn], yb[maxn];
        int n = 1; while ( n < x.n + y.n ) n *= 2;
        LL root = zp.pow( zp.pri, zp.mod / n );
        zp.dft( x.a, x.n, xb, n );
        rep(i, n) assert( x.eval(zp.pow(root, i)) == xb[i] );
        zp.dft( y.a, y.n, yb, n );
        rep(i, n) assert( y.eval(zp.pow(root, i)) == yb[i] );
        rep(i, n) xb[i] = zp.mul(xb[i], yb[i]);
        zp.nft(xb, r.a, n);
        r.n = n; while (r.n > 0 && r.a[r.n-1] == 0) --r.n;
    }
};

struct mp { //BigUnsignedInteger
    static const int digit = 4;
    static const int base = 10000;
    static const int cap = 50000 * 2 + 5; // 10 ^ 500
    static const int maxn = cap / digit + 1;
    int dat[maxn], n;

    mp(const mp& o) : n(o.n) {
        copy(o.dat, o.dat + n, dat);
    }
    mp(LL v = 0) {
        for (n = 0; v; v /= base) dat[n++] = v % base;
    }
    void parse(const char *s) {
        n = 0;
        for (int i = strlen(s) - 1, v = 0, m = 1; i >= 0; --i) {
            v = v + (s[i] - '0') * m; m *= 10;
            if (m == base || i == 0) {
                dat[n++] = v; v = 0; m = 1;
            }
        }
    }
    char *toString(char *s) const {
        if (n == 0) {
            sprintf(s, "0");
        } else {
            char *p = s;
            p += sprintf(p, "%d", dat[n-1]);
            for (int i = n - 2; i >= 0; --i)
                p += sprintf(p, "%0*d", digit, dat[i]);
        }
        return s;
    }
    char *toString() const {
        static char buf[cap + 5]; return toString(buf);
    }
    friend void add(mp& r, const mp& x, const mp& y) {
        int i = 0;
        for (int t = 0; i < x.n || i < y.n || t; ++i, t /= base) {
            if (i < x.n) t += x.dat[i];
            if (i < y.n) t += y.dat[i];
            r.dat[i] = t % base;
        }
        r.n = i;
    }
    friend void sub(mp& r, const mp& x, const mp& y) {
        r.n = x.n;
        for (int i = 0, t = 0; i < r.n; ++i) {
            r.dat[i] = x.dat[i] - t;
            if ( i < y.n ) r.dat[i] -= y.dat[i];
            if ( r.dat[i] < 0 ) {
                t = 1; r.dat[i] += base;
            } else {
                t = 0;
            }
        }
        while (r.n && 0 == r.dat[r.n - 1]) --r.n;
    }
    friend void mul(mp& r, const mp& x, int y) {
        int i = 0;
        for (LL t = 0; i < x.n || t; ++i, t /= base) {
            if (i < x.n) t += (LL)(x.dat[i]) * y;
            r.dat[i] = t % base;
        }
        r.n = i;
    }
    friend void mulfft(mp& r, const mp& x, const mp& y) {
        static poly px, py, pr;
        px.init(x.dat, x.n);
        py.init(y.dat, y.n);
        mul(pr, px, py);
        int i = 0;
        for (LL t = 0; i < pr.n || t; ++i, t /= base) {
            if (i < pr.n) t += pr.a[i];
            r.dat[i] = t % base;
        }
        r.n = i;
    }
    friend void div(mp& q, int &r, const mp& x, int y) {
        q.n = x.n; r = 0;
        for (int i = x.n - 1; i >= 0; --i, r %= y) {
            r = r * base + x.dat[i];
            q.dat[i] = r / y;
        }
        while (q.n && 0 == q.dat[q.n-1]) --q.n;
    }
};

int main() {
    static mp x, y, z;
    static char buf[1000000];
    while ( gets(buf) ) {
        x.parse(buf);
        gets(buf); y.parse(buf);
        mulfft(z, x, y);
        puts( z.toString() );
        }
}
```

23 halfPlane

```
struct Point {
    double x, y;
    void read() { scanf("%lf%lf", &x, &y); }
    Point() {}
    Point(double x, double y) : x(x), y(y) {}
    bool operator == (const Point& o) const { return !sign(x - o.x) && !sign(y
        - o.y); }
    int quad() const { return sign(x) >= 0 ? sign(y) >= 0 ? 1 : 4 : sign(y) >=
        0 ? 2 : 3; }
    double length() const { return sqrt(x * x + y * y); }
    Point setLength(double d) const { return *this * (d / length()); }
    Point unit() const { return *this / length(); }
    double project(const Point& n) const { //length: project to n
        return *this * n.unit();
    }
    friend int intersect(Point& p, const Point& a, const Point& v, const Point&
         b, const Point& u) {
        // a + v[t] = b + u[s] => v[t] - u[s] = b - a = c
        Point c = b - a; double d = u ^ v;
        if ( sign(d) == 0 ) { /*assume v != 0 && u != 0*/
            if ( sign(c ^ u) == 0 ) return -1; /*coincide*/
            return 0; /*parallel*/
        }
        double t = (u ^ c) / d; p = a + v * t; return 1;
    }
};

struct LineAV {
    Point a, v;
    LineAV() {}
    LineAV(const Point& a, const Point& v) : a(a), v(v) {}
#define LineST(a, b) LineAV( (a), (b) - (a) )
    void read() { Point a, b; a.read(); b.read(); *this = LineST(a, b); }
    LineAV offset(double d) const {
        return LineAV( a + Point(-v.y, v.x).setLength(d), v );
    }
    bool operator < (const LineAV& o) const {
        int dq = v.quad() - o.v.quad(); if (dq != 0) return dq < 0;
        int x = sign( v ^ o.v ); if (x != 0) return x > 0;
        return ( (o.a - a) ^ v ) > 0;
    }
    bool operator == (const LineAV& o) const {
        int dq = v.quad() - o.v.quad(); if (dq != 0) return false;
        int x = sign( v ^ o.v ); if (x != 0) return false;
        return true;
    }
    friend int intersect(Point& p, const LineAV& x, const LineAV& y) {
        return intersect(p, x.a, x.v, y.a, y.v);
    }
};

struct Geom {
    double cross(const Point& a, const Point& b, const Point& c) {
        // cross(a, b, c) > 0 iff. c left of ray a->b
        return (b - a) ^ (c - a);
    }
    bool onSegment(const Point& p, const Point& a, const Point& b) {
        if ( cross(a, b, p) != 0 ) return 0;
        return between(p.x, a.x, b.x) && between(p.y, a.y, b.y);
    }
    bool between(double t, double x, double y) {
        if (x > y) swap(x, y);
        return sign(x - t) <= 0 && sign(t - y) <= 0;
    }
} geom;

struct HalfPlane {
    bool out(const LineAV& x, const LineAV& y, const LineAV& z) {
        Point p; intersect(p, x, y);
        int d = sign( z.v ^ (p - z.a) ); if ( d != 0 ) return d < 0;
        int t = sign( x.v ^ z.v ) * sign( x.v ^ y.v ); return t > 0;
    }
    void solve(LineAV ls[], int &n, int &s, int &t) {
        sort(ls, ls + n); n = unique(ls, ls + n) - ls;
        int i, j;
        for (s = 0, t = 1, i = 2; i < n; ++i) {
            while (s < t && out(ls[t-1], ls[t], ls[i])) --t;
            while (s < t && out(ls[s+1], ls[s], ls[i])) ++s;
            ls[++t] = ls[i];
        }
        do {
            n = t - s + 1;
            while (s < t && out(ls[t-1], ls[t], ls[s])) --t;
            while (s < t && out(ls[s+1], ls[s], ls[t])) ++s;
        } while (n != t - s + 1);
        ls[t+1] = ls[s];
    }
} halfPlane;

Point vertex[] = { //千万要逆时针给出
    Point(-inf, -inf), Point(inf, -inf), Point(inf, inf), Point(-inf, inf),
};
LineAV edges[] = {
    LineST(vertex[0], vertex[1]), LineST(vertex[1], vertex[2]),
    LineST(vertex[2], vertex[3]), LineST(vertex[3], vertex[0]),
};

struct Poly {
    Point p[maxn]; int n;
    bool read() {
        if ( !(cin >> n) || 0 == n ) return false;
        rep(i, n) p[i].read(); p[n] = p[0];
        if ( area() < 0 ) reverse(p + 1, p + n); return true;
    }
    double area() const {
        double a = 0; rep(i, n) a += p[i] ^ p[i+1]; return a / 2;
    }
    Point centroid() const {
        Point ans = p[0]; for (int i = 1; i < n; ++i) ans = ans + p[i]; return
            ans / n;
    }
    int in(const Point& o) const {
        /* 1 -> strict in, 0 -> strict out, -1 -> onEdge, -2 -> onVertex*/
        rep(i, n) if ( p[i] == o ) return -2;
        rep(i, n) if ( geom.onSegment(p[i], p[i+1]) ) return -1;
        int wn = 0;
        rep(i, n) {
            int k = sign( geom.cross(p[i], p[i+1], o) );
            int d1 = sign(p[i].y - o.y), d2 = sign(p[i+1].y - o.y);
            if (k > 0 && d1 <= 0 && d2 > 0) ++wn;
            if (k < 0 && d2 <= 0 && d1 > 0) --wn;
        }
        return wn != 0;
    }
    bool calcCore(Poly& ans = this, double d = 0 /*offset*/) const {
        static LineAV ls[maxn];
        assert( area() >= 0 );
        rep(i, n) ls[i] = LineST(p[i], p[i+1]).offset(d);
        rep(i, 4) ls[i+n] = edges[i]; ans.n = n + 4;
        int s, t; halfPlane.solve(ls, ans.n, s, t);
        if ( ans.n < 3 ) return false;
        for (int i = s; i <= t; ++i) intersect(ans.p[i-s], ls[i], ls[i+1]);
        ans.p[ ans.n ] = ans.p[0]; return true;
    }
    void largestCircle(Point& o, double &r) const { //最大内切圆
        double low = 0, high = inf, mid;
        static Poly poly;
        while ( low < high - eps ) {
            mid = (low + high) / 2;
            if ( calcCore(poly, mid) ) {
                r = low = mid;
            } else {
                high = mid;
            }
        }
        calcCore(poly, r); o = poly.centroid();
    }
    pair<Point, Point> mostDistance() const { //最远点对
        double r = -1, t;
        Point x, y;
        for (int i = 0, j = 1; i < n; ++i) {
            while (cross(p[i],p[i+1],p[j+1]) > cross(p[i],p[i+1],p[j])+eps)
                if (++j == n) j = 0;
            if ((t = (p[i] - p[j]).length() ) > r) {
                r = t; x = p[i]; y = p[j];
            }
            if ((t = (p[i+1] - p[j]).length()) > r){
                r = t; x = p[i+1]; y = p[j];
            }
        }
        return make_pair(x, y);
    }
} poly;

struct Convex : Poly {
    int _in(const Point& o, int i) const {
        if ( o == p[i] || o == p[i+1] ) return -2;
        if ( geom.onSegment(o, p[i], p[i+1]) ) return -1;
        return sign( geom.cross(p[i], p[i+1], o) ) > 0;
    }
    int in(const Point& o) const {
        /*1 -> strict in, 0 ->strict out, -1 -> onEdge, -2 -> onVertex*/
        if ( _in(o, 0) != 1 ) return _in(o, 0);
        if ( _in(o, n-1) != 1 ) return _in(o, n-1);
        int low = 1, high = n - 2, k;
        while ( low <= high ) {
            int mid = (low + high) / 2;
            if ( sign( geom.cross(p[0], p[mid], o) ) >= 0 ) {
                k = mid; low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        if ( o == p[k] || o == p[k+1] ) return -2;
        int s = sign( geom.cross(p[k], p[k+1], o) );
        return s == 0 ? -1 : s > 0;
    }
    int intersect(Point p[], const LineA& ln) const { //TODO
    }
} convex;;
```

24 convex

```
double cross(const Point& a, const Point& b, const Point& c) { //ab x ac
    return (b.x - a.x) * (c.y - a.y) * (c.x - a.x);
}

int graham(Point q[], Point p[], int n) {
    int i, kk, k;
    sort( p, p + n, lessx() ); //unique
    if ( 1 == n ) { q[0] = q[1] = p[0]; return 1; }
    for (k = 0, i = 0; i < n; q[k++] = p[i++])
        while (k >= 2 && cross(q[k-2], q[k-1], p[i]) <= eps)
            --k; //要包含共线点则 < -eps
    for (kk = k; i = n - 2; i >= 0; q[k++] = p[i--])
        while (k > kk && cross(q[k-2], q[k-1], p[i]) <= eps)
            --k; //要包含共线点则 < -eps
   return k - 1;
}
```

25 convex3d

```
struct Point3D {
    double x, y, z;
    void read() { cin >> x >> y >> z; }
    Point3D() {}
    Point3D(double x, double y, double z) : x(x), y(y), z(z) { }
    Point3D operator ^ (const Point3D& o) const {
        return Point3D(y * o.z - z * o,y, z * o.x - x * o.z, x * o.y - y * o.x)
            ;
    }
    friend Point3D cross(const Point3D& a, const Point3D& b, const Point3D& c)
        {
        return (b-a) ^ (c-a);
    }
    friend double mix(const Point3D& a, const Point3D& b, const Point3D& c) {
        return (a^b) * c;
    }
    friend bool isColinear(const Point3D& a, const Point3D& b, const Point3D& c
        ) {
    return cross(a, b, c) == ZERO;
}
friend bool isCoplanar(const Point3D& a, const Point3D& b, const Point3D& c
    , const Point3D& d) {
    return sign( mix(b - a, c - a, d - a) ) == 0;
}
friend bool pointInTri(const Point3D& o, const Point3D& a, const Point3D& b
    , const Point3D& c) {
    Point3D x = cross(a, o, b), y = cross(b, o, c), z = cross(c, o, a);
    if ( sign(x * y) > 0 && sign(y * z) > 0 ) return true;
    if ( onSegment(o, a, b) || onSegment(o, b, c) || onSegment(o, c, a) )
        return -1;
    return false;
}
friend int onSegment(const Point3D& v, const Point3D& a, const Point3D& b)
    {
    if ( !isColinear(v, a, b) ) return false;
    int flag = sign( (v-a) * (v-b) );
    return flag == 0 ? -1 : flag < 0;
}
double length() const {
    return sqrt( x * x + y * y + z * z );
}
Point3D unit() const {
    return *this / length();
}
double project(const Point3D& o) const {
    return *this * o.unit();
}
bool operator < (const Point3D &o) const {
    if ( sign(x - o.x) != 0 ) return x < o.x;
    if ( sign(y - o.y) != 0 ) return y < o.y;
    if ( sign(z - o.z) != 0 ) return z < o.z;
    return false;
}
bool operator == (const Point3D& o) const {
    if ( sign(x - o.x) != 0 ) return false;
    if ( sign(y - o.y) != 0 ) return false;
    if ( sign(z - o.z) != 0 ) return false;
```

```
        return true;
    }
    static Point3D ZERO;
} Point3D::ZERO(0, 0, 0);

struct Poly3D {
    static const int maxn = 100 + 5;
    static const int maxf = maxn * maxn + 5;
    Point3D p[maxn]; int n;
    int f[maxf][3]; int fc;

    void read(int n) {
        rep(i, n) p[i].read();
        sort(p, p + n); n = unique(p, p + n) - p;
        this->n = n; fc = 0;
    }

    bool convex() {
        random_shuffle(p, p + n);
        if ( !findTet() ) return false;
        for (int i = 3; i < n; ++i) addpoint(i);
        return true;
    }
    bool findTet() {
        for (int i = 2; i < n; ++i) {
            if ( !isColinear(p[0], p[1], p[i]) ) {
                swap(p[2], p[i]);
                for (int j = i + 1; j < n; ++j) {
                    swap(p[3], p[j]);
                    addface(0, 1, 2); addface(0, 2, 1);
                    return true;
                }
                return false;
            }
        }
        return false;
    }
    void addpoint(int v) {
        static int mark[maxn][maxn], cnt = 0;
        ++cnt;
        bool flag = false;
        for (int i = 0; i < fc; ) {
            int a = f[i][0], b = f[i][1], c = f[i][2];
            if ( sign( mix(p[a] - p[v], p[b] - p[v], p[c] - p[v]) ) < 0 ) {
                flag = true;
                mark[a][b] = mark[b][a] = mark[a][c] = mark[c][a] = mark[b][c]
                    = mark[c][b] = cnt;
                delface(i);
            } else {
                ++i;
            }
        }
        if (!flag) return;
        for (int i = 0, _fc = fc; i < _fc; ++i) {
            int a = f[i][0], b = f[i][1], c = f[i][2];
            if ( mark[a][b] == cnt ) addface(b, a, v);
            if ( mark[b][c] == cnt ) addface(c, b, v);
            if ( mark[c][a] == cnt ) addface(a, c, v);
        }
    }

    void addface(int a, int b, int c) {
        f[fc][0] = a; f[fc][1] = b; f[fc][2] = c; ++fc;
    }
    void delface(int i) {
        memmove(f[i], f[--fc], sizeof(f[i]));
    }
    int in(const Point3D& o) const { /*-1 on face*/
        for (int i = 0; i < fc; ++i) {
            const Point3D& a = p[f[i][0]], &b = p[f[i][1]], &c = p[f[i][2]];
            int flag = sign( mix(a-o, b-o, c-o) );
            if ( flag == 0 && pointInTri(o, a, b, c) ) return -1;
            if ( flag < 0 ) return false;
        }
        return true;
    }
    double dist(const Point3D& o) const {
        double ans = inf;
        for (int i = 0; i < fc; ++i) {
            const Point3D& a = p[f[i][0]], &b = p[f[i][1]], &c = p[f[i][2]];
            Point3D normal = (b - a) ^ (c - a);
            double d = (a - o).project( normal );
            checkmin(ans, d);
        }
        return ans;
    }
    double facecnt() const {
        static Point3D normal[maxf];
        for (int i = 0; i < fc; ++i) {
            const Point3D& a = p[f[i][0]], &b = p[f[i][1]], &c = p[f[i][2]];
            normal[i] = cross(a, b, c).unit();
        }
        sort(normal, normal + fc);
        return unique(normal, normal + fc) - normal;
    }
    double surface() const {
        double ans = 0;
        for (int i = 0; i < fc; ++i) {
            const Point3D& a = p[f[i][0]], &b = p[f[i][1]], &c = p[f[i][2]];
            ans += cross(a, b, c).length();
        }
        return ans / 2;
    }
    double volume() const {
        double ans = 0;
        Point3D o = p[0];
        for (int i = 0; i < fc; ++i) {
            const Point3D& a = p[f[i][0]], &b = p[f[i][1]], &c = p[f[i][2]];
            ans += mix(a - o, b - o, c - o);
        }
        return ans / 6;
    }
    Point3D massCenter() const {
        Point3D ans = Point3D::ZERO;
        double vol = 0;
        const Point3D o = p[0];
        for (int i = 0; i < fc; ++i) {
            const Point3D& a = p[f[i][0]], &b = p[f[i][1]], &c = p[f[i][2]];
            double v = mix(a-o, b-o, c-o);
            ans = ans + (o+a+b+c) * (v/4);
            vol += v;
        }
        return ans / vol;
    }
} poly;
```

26 geom3d

```
struct LineAV3D {
    Point3D a, v;
#define LineST3D( s, t ) LineAV3D( (s), (t)-(s) )
    LineAV3D() {}
    LineAV3D{const Point3D& a, const Point3D& v} : a(a), v(v) {}
};

struct Plane3D {
    Point3D a, b, c;
    Point3D normal() const {
        return cross(a, b, c);
    }
};

struct Geom3D {
    double dist(const Point3D& a, const Pont3D& b) {
        return (b-a).length();
    }
    double dist(const Point3D& p, const LineAV3D& ln) {
        double area2 = ( (p-ln.a) ^ ln.v ).length();
        reutrn aera2 / ln.v.length();
    }
    double dist(const Point3D& p, const Plane3D& s) {
        return (p - s.a).project( s.normal() );
    }
    double dist(const LineAV3D& x, const LineAV3D& y) {
        Point3D n = x.v ^ y.v;
        if ( n.isZero() )
        return dist(x.a, y);
    else
        return (x.a - y.a).project(n);
}
Point3D foot(const Point3D& p, const LineAV3D &ln) {
    return ln.a + ln.v.unit() * (p - ln.a).project(ln.v);
}
Point3D foot(const Point3D& p, const Plane3D& s) {
    Point3D n = s.normal();
    return p + n.unit() * (s.a - p).project(n);
}
int intersect(Point3D& p, const LineAV3D& ln, const Point3D& s) {
    Point3D n = s.normal();
    double x = ln.v.project(n);
    if ( sign(x) == 0 ) {
        if ( sign( dist(ln.a, s) ) == 0 ) return -1; /*infinity*/
        else return 0; /*parallel*/
    }
    double t = ( foot(ln.a, s) - ln.a ).length() / x;
    p = ln.a + ln.v * t;
    return 1;
}
int intersect(LineAV3D& ln, const Plane3D& x, const Plane3D& y) {
    Point3D nx = x.normal(), ny = y.normal();
    Point3D v = nx ^ ny;
    if ( v.isZero() ) return 0;
    Point3D a; intersect( a, LineST3D(x.a, x.b), y );
    ln = LineAV3D(a, v); return 1;
}
double angle(const Point3D& x, const Point3D& y) { //vector
    double cos = x * y / ( x.length() * y.length() );
    return acos( fabs(cos) );
}
double angle(const LineAV3D& x, const LineAV3D& y) {
    reutrn angle(x.v, y.v);
}
double angle(const Plane3D& x, const Plane3D& y) {
    return angle( x.normal(), y.normal() );
}
};
```

27 给你一些线段，找到一条直线，要求穿过尽量多的线段 –$O(n^2logn)$

```
//注意需要慎重atan2
#define x first
#define y second
const double eps = 1e-8;

typedef pair<double, double> Point;
Point operator - (Point a, Point b) { return make_pair(a.x - b.x, a.y - b.y); }
double operator ^ (Point a, Point b) { return a.x * b.y - b.x * a.y; }
double operator * (Point a, Point b) { return a.x * b.x + a.y * b.y; }
int deval(double d) { return x < -eps ? -1 : x > eps; }
int dcmp(double x, double y) { return deval(x - y); }

const int maxn = 1000;
Point a[maxn], b[maxn], p[maxn * 2];

struct Elem {
    double arg; //Point p
    int flag; // 1 for in, -1 for out
};

Elem make_elem(Point p, int flag) {
    Elem ret;
    ret.arg = atan2(p.y, p.x); // may use atan2l
    while ( dcmp( ret.arg, 0 ) < 0 ) ret.arg += M_PI; //如果是射线，改为M_2PI, 下
        同
    while ( dcmp( ret.arg, M_PI ) >= 0 ) ret.arg -= M_PI; //注意不要
        用 deval( ret.arg ) >= M_PI
    ret.flag = flag; return ret;
}

bool operator < (Elem x, Elem y) {
    if ( dcmp(x.arg, y.arg) != 0 ) return x.arg < y.arg;
    //#define quad(p_ { p.x >= 0 ? p.y >= 0 ? 1 : 4 : p.y >= 0 ? 2 : 3  }
    //int qx = quad(x.p), qy = quad(y.p); if (qx != qy) return qx < qy;
    //int x = cross(x.p, y.p); if (x != 0) return x > 0;
    return x.flag > y.flag;
}

void solve() {
    int n; cin >> n;
    int pcnt = 0, res = 0;
    for (int i = 0; i < n; ++i) {
        scanf("%lf%lf", &a[i].x, &a[i].y); p[pcnt++] = a[i];
        scanf("%lf%lf", &b[i].x, &b[i].y); p[pcnt++] = b[i];
    }
    sort(p, p + pcnt); pcnt = unique(p, p + pcnt) - p;

    for (int k = 0; k < pcnt; ++k) {
        Point o = p[k];
        int elemcnt = 0, countAt = 0, current = 0;
        for (int i = 0; i < n; ++i) {
            Point oa = a[i] - o, ob = b[i] - o;
            int crossVal = deval( oa ^ ob );
            int dotVal = deval( oa * ob );
            if ( crossVal == 0 && dotVal <= 0 ) {
                countAt++; continue;
            }
            if ( crossVal < 0 ) swap(oa, ob);
            elem[elemcnt++] = make_elem(oa, 1);
            elem[elemcnt++] = make_elem(ob, -1);
            if ( deval(oa.y) * deval(ob.y) < 0 || deval(ob.y) == 0 && deval(oa.
                y) != 0 )
                ++current; //如果是射线，则改
                    为os.y < 0 && ob.y > 0 || oa.y < 0 && ob.y == 0
        }
        sort( elem, elem + elemcnt );
        int mval = current;
        for (int i = 0; i < elemcnt; ++i) {
            current += elem[i].flag;
            mval = max(mval, current);
        }
        res = max(res, mval + countAt);
    }
    cout << res << endl;
}
```

28 Dancing Links 精确覆盖 (矩阵处理)

```
const int maxN = 60 * 20, maxM = 60 * 10;
const int max_size = maxN * maxM;
const int inf = 0x3f3f3f3f;
int L[max_size], R[max_size], U[max_size], D[max_size], CH[max_size], RH[
    max_size];
int S[maxM], O[maxM];
int head, size;
int node(int up, int down, int left, int right) {
    U[size] = up, D[size] = down;
    L[size] = left, R[size] = right;
    D[up] = U[down] = R[left] = L[right] = size;
    return size++;
}
bool mat[maxN][maxM];
void init(int N, int M) {
    size = 0;
    head = node(0, 0, 0, 0);
    for (int j = 1; j <= M; ++j) {
        CH[j] = node(size, size, L[head], head), S[j] = 0;
    }
    for (int i = 1; i <= N; ++i) {
        int row = -1, k;
        for (int j = 1; j <= M; ++j) {
            if (!mat[i][j]) continue;
            if (row == -1) {
                row = node(U[CH[j]], CH[j], size, size);
                RH[row] = i, CH[row] = CH[j], ++S[j];
            } else {
                k = node(U[CH[j]], CH[j], L[row], row);
                RH[k] = i, CH[k] = CH[j], ++S[j];
            }
        }
    }
}
void remove(const int &c) {
    L[R[c]] = L[c], R[L[c]] = R[c];
    for (int i = D[c]; i != c; i = D[i]) {
        for (int j = R[i]; j != i; j = R[j]) {
            U[D[j]] = U[j], D[U[j]] = D[j];
            --S[CH[j]];
        }
    }
}
void resume(const int &c) {
    for (int i = U[c]; i != c; i = U[i]) {
        for (int j = L[i]; j != i; j = L[j]) {
            ++S[CH[j]];
            U[D[j]] = D[U[j]] = j;
        }
    }
    L[R[c]] = R[L[c]] = c;
}
int len;
bool DLX(const int &k) {
    if (R[head] == head) {
        len = k - 1;
        return true;
    }
    int s = inf, c;
    for (int t = R[head]; t != head; t = R[t]) {
        if (S[t] < s) s = S[t], c = t;
    }
    remove(c);
    for (int i = D[c]; i != c; i = D[i]) {
        O[k] = RH[i];
        for (int j = R[i]; j != i; j = R[j]) {
            remove(CH[j]);
        }
        if (DLX(k + 1)) {
            return true;
        }
        for (int j = L[i]; j != i; j = L[j]) {
            resume(CH[j]);
        }
    }
    resume(c);
    return false;
}
```

29 Dancing Links 重复覆盖 (矩阵处理)

```
const int head = 0;
const int INF=10000000;
const int maxn = 1700;
const int maxd = 1000000;
int N, M, K, n, m, cnt, res;
int mat[maxn][maxn], s[maxn], l[maxd], r[maxd], u[maxd], d[maxd], c[maxd], o[
    maxn], row[maxd];
bool use[maxn];
void makegragh(int &n, int &m) {
    memset(mat, 0, sizeof(mat));
    //init
}
void initial(int n, int m) {
    memset(use, false, sizeof(use));
    res = n + 1;
    int i, j, rowh;
    memset(s, 0, sizeof(s));
    for(i=head; i<=m; i++) {
        r[i]=(i+1)%(m+1);
        l[i]=(i-1+m+1)%(m+1);
        u[i]=d[i]=i;
    }
    cnt=m+1;
    for(i=1; i<=n; i++) {
        rowh=-1;
        for(j=1; j<=m; j++) {
            if(mat[i][j])
            {
                s[j]++; u[cnt]=u[j]; d[u[j]]=cnt;
                u[j]=cnt; d[cnt]=j; row[cnt]=i; c[cnt]=j;
                if(rowh==-1) {
                    l[cnt]=r[cnt]=cnt; rowh=cnt;
                }
                else {
                    l[cnt] = l[rowh]; r[l[rowh]] = cnt;
                    r[cnt] = rowh; l[rowh] = cnt;
                }
                cnt++;
            }
        }
    }
}
void remove(int c) {
    for(int i=d[c]; i!=c; i=d[i]) {
        r[l[i]]=r[i]; l[r[i]]=l[i];
    }
}
void resume(int c) {
    for(int i=d[c]; i!=c; i=d[i])
        r[l[i]]=l[r[i]]=i;
}
int h() {
    bool has[maxn];
    memset(has, false, sizeof(has));
    int ans=0;
    for(int i=r[head]; i!=head; i=r[i])
        if(!has[i]) {
            ans++;
            for(int j=d[i]; j!=i; j=d[j])
                for(int k=r[j]; k!=j; k=r[k])
                    has[c[k]]=true;
        }
    return ans;
}
bool dfs(int k) {
    if(k+h()>=res)return false;//A* cut
    if(r[head]==head) {
        if(k<res) res=k;
        return true;
    }
    int ms=INF, cur=0;
    for(int t=r[head]; t!=head; t=r[t])
        if(s[t]<ms) {
            ms=s[t]; cur=t;
        }
    for(int i=d[cur]; i!=cur; i=d[i]) {
        remove(i);
        for(int j=r[i]; j!=i; j=r[j]) {
            remove(j); s[c[j]]--;
        }
        dfs(k+1);
        for(int j=l[i]; j!=i; j=l[j]) {
            resume(j); s[c[j]]++;
        }
        resume(i);
    }
    return false;
}
```