ACM ICPC Reference

Institute of Mathematics and Statistics of University of São Paulo

May 26, 2011

Contents

1 Numbers and Number Theory

Arbitrary Precision Integers

bigint.cpp

```
const int DIG = 4;
const int BASE = 1000; // BASE**3 < 2**51
const int TAM = 2048;
struct bigint {
    int v[TAM], n;
    bigint(int x = 0): n(1) {
        memset(v, 0, sizeof(v)); v[n++] = x; fix();
    }
    bigint(char *s): n(1) {
        memset(v, 0, sizeof(v));
        int sign = 1;
        while (*s && !isdigit(*s)) if (*s++ == '-') sign *= -1;
        char *t = strdup(s), *p = t + strlen(t);
        while (p > t) {
            *p = 0; p = max(t, p - DIG);
            sscanf(p, "%d", &v[n]);
            v[n++] *= sign;
        }
        free(t); fix();
    }
    bigint& fix(int m = 0) {
        n = max(m, n);
        int sign = 0;
        for (int i = 1, e = 0; i <= n || e && (n = i); i++) {
            v[i] += e; e = v[i] / BASE; v[i] %= BASE;
            if (v[i]) sign = (v[i] > 0) ? 1 : -1;
        }
        for (int i = n - 1; i > 0; i--)
```

```
        if (v[i] * sign < 0) { v[i] += sign * BASE; v[i+1] -= sign; }
    while (n && !v[n]) n--;
    return *this;
}
int cmp(const bigint& x = 0) const {
    int i = max(n, x.n), t = 0;
    while (1) if ((t = ::cmp(v[i], x.v[i])) || i-- == 0) return t;
}
bool operator <(const bigint& x) const { return cmp(x) < 0; }
bool operator ==(const bigint& x) const { return cmp(x) == 0; }
bool operator !=(const bigint& x) const { return cmp(x) != 0; }
operator string() const {
    ostringstream s; s << v[n];
    for (int i = n - 1; i > 0; i--) {
        s.width(DIG); s.fill('0'); s << abs(v[i]);
    }
    return s.str();
}
friend ostream& operator <<(ostream& o, const bigint& x) {
    return o << (string) x;
}
bigint& operator +=(const bigint& x) {
    for (int i = 1; i <= x.n; i++) v[i] += x.v[i];
    return fix(x.n);
}
bigint operator +(const bigint& x) { return bigint(*this) += x; }
bigint& operator -=(const bigint& x) {
    for (int i = 1; i <= x.n; i++) v[i] -= x.v[i];
    return fix(x.n);
}
bigint operator -(const bigint& x) { return bigint(*this) -= x; }
bigint operator -() { bigint r = 0; return r -= *this; }
void ams(const bigint& x, int m, int b) { // *this += (x * m) << b;
    for (int i = 1, e = 0; (i <= x.n || e) && (n = i + b); i++) {
        v[i+b] += x.v[i] * m + e; e = v[i+b] / BASE; v[i+b] %= BASE;
    }
}
bigint operator *(const bigint& x) const {
    bigint r;
    for (int i = 1; i <= n; i++) r.ams(x, v[i], i-1);
    return r;
}
bigint& operator *=(const bigint& x) { return *this = *this * x; }
// cmp(x / y) == cmp(x) * cmp(y); cmp(x % y) == cmp(x);
bigint div(const bigint& x) {
    if (x == 0) return 0;
    bigint q; q.n = max(n - x.n + 1, 0);
    int d = x.v[x.n] * BASE + x.v[x.n-1];
    for (int i = q.n; i > 0; i--) {
        int j = x.n + i - 1;
        q.v[i] = int((v[j] * double(BASE) + v[j-1]) / d);
        ams(x, -q.v[i], i-1);
        if (i == 1 || j == 1) break;
        v[j-1] += BASE * v[j]; v[j] = 0;
    }
        fix(x.n); return q.fix();
    }
    bigint& operator /=(const bigint& x) { return *this = div(x); }
    bigint& operator %=(const bigint& x) { div(x); return *this; }
    bigint operator /(const bigint& x) { return bigint(*this).div(x); }
    bigint operator %(const bigint& x) { return bigint(*this) %= x; }
    bigint pow(int x) {
        if (x < 0) return (*this == 1 || *this == -1) ? pow(-x) : 0;
        bigint r = 1;
        for (int i = 0; i < x; i++) r *= *this;
        return r;
    }
    bigint root(int x) {
        if (cmp() == 0 || cmp() < 0 && x % 2 == 0) return 0;
        if (*this == 1 || x == 1) return *this;
        if (cmp() < 0) return -(-*this).root(x);
        bigint a = 1, d = *this;
        while (d != 1) {
            bigint b = a + (d /= 2);
            if (cmp(b.pow(x)) >= 0) { d += 1; a = b; }
        }
        return a;
    }
};
```

Extended Euclid's Algorithm (Bézout's Theorem)

bezout.cpp

```
typedef pair<int,int> bezout;
bezout find_bezout(int x, int y) {
    if (y == 0) return bezout(1, 0);
    bezout u = find_bezout(y, x % y);
    return bezout(u.second, u.first - (x/y) * u.second);
}
```

62-bit Modular Multiplication / Exponentiation

mulmod.cpp

```
llu mul_mod(llu a, llu b, llu m) {
    llu y = (llu)((ldouble)a*(ldouble)b/m+(ldouble)1/2); y = y * m;
    llu x = a * b, r = x - y;
    if ((lld) r < 0) { r = r + m; y = y - 1; }
    return r;
}
```

expmod.cpp

```
llu exp_mod(llu a, llu e, llu mod) {
    if (e == 0) return 1;
    llu b = exp_mod(a, e/2, mod);
    return (e % 2 == 0) ? mul_mod(b, b, mod) : mul_mod(mul_mod(b, b, mod), a, mod);
}
```

Miller–Rabin (Primality test)

isprime.cpp

```
llu llrand() { llu a = rand(); a<<= 32; a+= rand(); return a;}
int is_probably_prime(llu n) {
    if (n <= 1) return 0;
    if (n <= 3) return 1;
    llu s = 0, d = n - 1;
    while (d % 2 == 0) {
        d/= 2; s++;
    }
    for (int k = 0; k < 64; k++) {
        llu a = (llrand() % (n - 3)) + 2;
        llu x = exp_mod(a, d, n);
        if (x != 1 && x != n-1) {
            for (int r = 1; r < s; r++) {
                x = mul_mod(x, x, n);
                if (x == 1)
                    return 0;
                if (x == n-1)
                    break;
            }
            if (x != n-1)
                return 0;
        }
    }
    return 1;
}
```

Pollard's Rho (Factorization)

rho.cpp

```
llu rho(llu n) {
    llu d, c = rand() % n, x = rand() % n, xx = x;
    if (n % 2 == 0)
        return 2;
    do {
        x = (mul_mod(x, x, n) + c) % n;
        xx = (mul_mod(xx, xx, n) + c) % n;
        xx = (mul_mod(xx, xx, n) + c) % n;
        d = gcd(val_abs(x - xx), n);
    } while (d == 1);
    return d;
}
map <llu,int> F;
void factor(llu n) {
    if (n == 1)
        return;
    if (is_probably_prime(n)) {
        F[n]++;
        return;
    }
    llu d = rho(n);
    factor(d);
    factor(n/d);
}
```

Brent's Algorithm (Cycle detection)

Let $x_0 \in S$ be an element of the finite set S and consider a function $f : S \to S$. Define

$$f_k(x) = \begin{cases} x, & k = 0 \\ f(f_{k-1}(x)), & k > 0 \end{cases}.$$

Clearly, there exists distinct numbers $i, j \in \mathbb{N}$, $i \neq j$, such that $f_i(x_0) = f_j(x_0)$.

Let $\mu \in \mathbb{N}$ be the least value such that there exists $j \in \mathbb{N} \setminus \{\mu\}$ such that $f_\mu(x_0) = f_j(x_0)$ and let $\lambda \in \mathbb{N}$ be the least value such that $f_\mu(x_0) = f_{\mu+\lambda}(x_0)$.

Given x_0 and f, this code computes μ and λ applying the operator f $\mathcal{O}(\mu + \lambda)$ times and storing at most a constant amount of elements from S.

brent.cpp

```
p = l = 1;
t = x0;
h = f(x0);
while (t != h) {
    if (p == l) {
        t = h;
        p*= 2;
        l = 0;
    }
    h = f(h);
    ++l;
}

u = 0;
t = h = x0;
for (i = l; i != 0; --i)
```

```
h = f(h);
while (t != h) {
    t = f(t);
    h = f(h);
    ++u;
}

/*
 *  \mu = u
 * \lam = l
 */
```

2 Counting

Catalan Numbers

C_n is:

- The number of balanced expressions built from n pairs of parentheses.
- The number of paths in an $n \times n$ grid that stays on or below the diagonal.
- The number of words of size $2n$ over the alphabet $\Sigma = \{a, b\}$ having an equal number of a symbols and b symbols containing no prefix with more a symbols than b symbols.

It holds that:

$$C_0 = 1, C_{n+1} = \sum_{k=0}^{n} C_k C_{n-k}$$

$$C_n = \binom{2n}{n} - \binom{2n}{n-1} = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{n!(n+1)!}$$

Stirling Numbers of the First Kind

$\begin{bmatrix} n \\ k \end{bmatrix}$ is:

- The number of ways to split n elements into k ordered partitions up to a permutation of the partitions among themselves and rotations within the partitions.
- The number of digraphs with n vertices and k cycles such that each vertex has in and out degree of 1.

It holds that:

$$\begin{bmatrix} n \\ 0 \end{bmatrix} = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}, \quad \begin{bmatrix} 0 \\ k \end{bmatrix} = \begin{cases} 1, & k = 0 \\ 0, & k \neq 0 \end{cases}$$

$$\begin{bmatrix} n \\ k \end{bmatrix} = (n-1)\begin{bmatrix} n-1 \\ k \end{bmatrix} + \begin{bmatrix} n-1 \\ k-1 \end{bmatrix}$$

$$\begin{bmatrix} n \\ 1 \end{bmatrix} = (n-1)!$$

$$\begin{bmatrix} n \\ n-1 \end{bmatrix} = \binom{n}{2}$$

$$\begin{bmatrix} n \\ n-2 \end{bmatrix} = \frac{1}{4}(3n-1)\binom{n}{3}$$

$$\begin{bmatrix} n \\ n-3 \end{bmatrix} = \binom{n}{2}\binom{n}{4}$$

$$\begin{bmatrix} n \\ 2 \end{bmatrix} = (n-1)!H_{n-1}$$

$$\begin{bmatrix} n \\ 3 \end{bmatrix} = \frac{1}{2}(n-1)!\left(H_{n-1}^2 - H_{n-1}^{(2)}\right)$$

$$H_n = \sum_{j=1}^{n} \frac{1}{j}, \quad H_n^{(k)} = \sum_{j=1}^{n} \frac{1}{j^k}$$

$$\sum_{k=0}^{n} \begin{bmatrix} n \\ k \end{bmatrix} = n!$$

$$\sum_{j=k}^{n} \begin{bmatrix} n \\ j \end{bmatrix}\binom{j}{k} = \begin{bmatrix} n+1 \\ k+1 \end{bmatrix}$$

Stirling Numbers of the Second Kind

$\begin{Bmatrix} n \\ k \end{Bmatrix}$ is the number of ways to partition an n-set into exactly k non-empty disjoint subsets up to a permutation of the sets among themselves. It holds that:

$$\begin{Bmatrix} n \\ 0 \end{Bmatrix} = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}, \quad \begin{Bmatrix} n \\ 1 \end{Bmatrix} = \begin{Bmatrix} n \\ n \end{Bmatrix} = 1$$

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \begin{Bmatrix} n-1 \\ k-1 \end{Bmatrix} + k\begin{Bmatrix} n-1 \\ k \end{Bmatrix}$$

$$\begin{Bmatrix} n \\ k \end{Bmatrix} \bmod 2 = \begin{cases} 0, & (n-k)\,\&\,\left\lfloor \frac{k-1}{2} \right\rfloor \neq 0 \\ 1, & \text{otherwise} \end{cases},$$

where & is the C bitwise "and" operator.

$$\begin{Bmatrix} n \\ 2 \end{Bmatrix} = 2^{n-1} - 1$$

$$\begin{Bmatrix} n \\ n-1 \end{Bmatrix} = \binom{n}{2}$$

$$\begin{Bmatrix} n \\ k \end{Bmatrix} = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

Bell Numbers

$\mathcal{B}_n$ is the number of equivalence relations on an n-set or, alternatively, the number of partitions of an n-set. It holds that:

$$\mathcal{B}_n = \sum_{k=0}^{n} \begin{Bmatrix} n \\ k \end{Bmatrix}$$

$$\mathcal{B}_{n+1} = \sum_{k=0}^{n} \binom{n}{k}\mathcal{B}_k$$

$$\mathcal{B}_n = \frac{1}{e}\sum_{k=0}^{\infty} \frac{k^n}{k!}$$

$$\mathcal{B}_{n+p} \equiv \mathcal{B}_n + \mathcal{B}_{n+1} \pmod{p}$$

The Twelvefold Way

Let A be a set of m balls and B be a set of n boxes. The following table provides methods to compute the number of equivalent functions $f : A \to B$ satisfying specific constraints.

Balls	Boxes	Any	Injective	Surjective
$\not\equiv$	$\not\equiv$	n^m	$\frac{n!}{(n-m)!}$	$n!\left\{ {m \atop n} \right\}$
$\not\equiv$	$\equiv$	$\sum_{k=0}^{n} \left\{ {m \atop k} \right\}$	$\delta_{m \leqslant n}$	$\left\{ {m \atop n} \right\}$
$\equiv$	$\not\equiv$	$\binom{m+n-1}{m}$	$\binom{n}{m}$	$\binom{m-1}{n-1}$
$\equiv$	$\equiv$	**(*)** $\sum_{k=0}^{n} p(m,k)$	$\delta_{m \leqslant n}$	**(**)** $p(m,n)$

(**) is a definition and both **(*)** and **(**)** are very hard to compute. So do not try to.

Lucca's Theorem

Let $n, k, p \in \mathbb{N}$ and p be a prime number. Then

$$\binom{n}{k} \equiv \prod_{j=0}^{\infty} \binom{n_j}{k_j} \pmod{p},$$

where n_j and k_j are the j-th digits of the numbers n and k in base p, respectively.

Derangement (Desarranjo)

A derangement is a permutation of the elements of a set such that none of the elements appear in their original position.

Suppose that there are n persons numbered $1, 2, \ldots, n$. Let there be n hats also numbered $1, 2, \ldots, n$. We have to find the number of ways in which no one gets the hat having same number as his/her number. Let us assume that first person takes the hat i. There are $n-1$ ways for the first person to choose the number i. Now there are 2 options:

- Person i takes the hat of 1. Now the problem reduces to $n-2$ persons and $n-2$ hats.
- Person i does not take the hat 1. This case is equivalent to solving the problem with $n-1$ persons $n-1$ hats (each of the remaining $n-1$ people has precisely 1 forbidden choice from among the remaining $n-1$ hats).

From this, the following relation is derived:

$$d_n = (n-1) * (d_{n-1} + d_{n-2})$$

$$d_1 = 0$$

$$d_2 = 1$$

Starting with $n = 0$, the numbers of derangements of n are: 1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961, 14684570, 176214841, 2290792932.

---

3 Strings

Knuth–Morris–Pratt Algorithm

kmp.cpp

```
int pi[MAX]; char p[MAX], p2[MAX], T;
void compute_prefix_function(int m){
    int k = pi[0] = -1;
    for (int i = 1; i < m; i++) {
        while (k >= 0 && p[i] != p[k+1])
            k = pi[k];
        if (p[i] == p[k+1]) k++;
        pi[i] = k;
    }
}
int KMP(int m, int n){
    int q = -1;
    compute_prefix_function(m);
    for (int i = 0; i < n; i++) {
        while (q >= 0 && p2[i] != p[q+1])
            q = pi[q];
        if (p2[i] == p[q+1]) q++;
        if (q >= m-1) {
            printf("%d\n",i-m+1);
            q = pi[q];
        }
    }
    return 0;
}
/* p := padrao a ser procurado em p2
scanf("%s", p); scanf("%s", p2); n = strlen(p2); m = strlen(p); KMP(m, n); */
```

Aho–Corasick Algorithm

ahocorasick.cpp

```
struct Trie {
    short folha; /*Se w <= 1<<16*/ int goFail; map<char,int> filho;
};
#define WMAX 1024
#define TMAX 1024
#define TEXTSIZE 1024*1024
Trie trie[WMAX*TMAX];
char texto[TEXTSIZE], word[WMAX][TMAX];
int tam[WMAX], cabeca = WMAX*TMAX-1, w, id[WMAX];
vector<int> ocorrencias[WMAX];
int insere(char c, int at) {
    if (trie[at].filho.count(c)) return trie[at].filho[c];
    else return trie[at].filho[c] = ++cabeca;
}
```

```
void montaTrie() {
    for (int i = 0; i <= cabeca; i++) {
        trie[i].folha = 0; trie[i].goFail = -1; trie[i].filho.clear();
    }
    cabeca = 0;
    for (int i = 0; i < w; i++) {
        int at = 0;
        for (int j = 0; j < tam[i]; j++) at = insere(word[i][j], at);
        if (trie[at].folha == 0) /* evita repeticao */ trie[at].folha = i+1;
        else id[i] = trie[at].folha-1;
    }
}
void arrumaGoFails() {
    queue<int> fila; map<char,int>::iterator it;
    for (it = trie[0].filho.begin(); it != trie[0].filho.end(); it++) {
        trie[it->second].goFail = 0;
        fila.push( it->second );
    }
    while (!fila.empty()) {
        int no = fila.front();
        fila.pop();
        for (it = trie[no].filho.begin(); it != trie[no].filho.end(); it++) {
            int atual = it->second, failPai = trie[no].goFail, caminho = it->first;
            fila.push(atual);
            while (failPai >= 0 && !trie[failPai].filho.count(caminho))
                failPai = trie[failPai].goFail;
            if(failPai == -1) {
                if(trie[0].filho.count(caminho)) trie[atual].goFail = trie[0].filho[caminho];
                else trie[atual].goFail = 0;
            } else {
                trie[atual].goFail = trie[failPai].filho[caminho];
                if (trie[atual].folha == 0 && trie[ trie[atual].goFail ].folha != 0)
                    trie[atual].folha = trie[ trie[atual].goFail ].folha;
            }
        }
    }
}
void fazConsulta(char * s) {
    int at = 0;
    for (int pos = 0; s[pos]; pos++) {
        while (at >= 0 && !trie[at].filho.count(s[pos])) at = trie[at].goFail;
        if (at == -1) at = 0;
        else at = trie[at].filho[s[pos]];
        int tmp = at;
        while (tmp >= 0 && trie[tmp].folha) {
            ocorrencias[ trie[tmp].folha-1 ].push_back( pos - tam[ trie[tmp].folha - 1 ] + 1 );
            while (trie[tmp].goFail != -1 && trie[ trie[tmp].goFail ].folha == trie[tmp].folha)
                tmp = trie[tmp].goFail;
            tmp = trie[tmp].goFail;
        }
    }
}
int main() {
    scanf("%s", texto); scanf("%d", &w);
    for(int i = 0; i < w; i++) {
        scanf("%s", word[i]);
        id[i] = i; tam[i] = strlen(word[i]);
    }
    montaTrie(); arrumaGoFails();
    for(int i = 0; i < w; i++) ocorrencias[i].clear();
    fazConsulta(texto);
    for(int i = 0; i < w; i++) printf("%s\n", ocorrencias[ id[i] ].empty() ? "n" : "y" );
}
```

Suffix Array and Longest Common Prefix (DC3)

suffixarray.cpp

```
#define LMAX 20
#define NMAX 131072
int pool[LMAX][4][NMAX];
int tmp[NMAX];
inline bool leq(int a1, int a2, int b1, int b2) { return a1 < b1 || (a1 == b1 && a2 <= b2); }
inline bool leq(int a1, int a2, int a3, int b1, int b2, int b3) { return a1 < b1 ||
    (a1 == b1 && leq(a2, a3, b2, b3)); }
static void radix(int *a, int *b, int *r, int n, int K) {
    int *c = tmp;
    for (int i = 0; i <= K; i++) { c[i] = 0; }
    for (int i = 0; i < n; i++) { c[r[a[i]]]++; }
    for (int i = 0, sum = 0; i <= K; i++) { int t = c[i]; c[i] = sum; sum+= t; }
    for (int i = 0; i < n; i++) { b[c[r[a[i]]]++] = a[i]; }
}
#define GetI() (SA12[t] < n0 ? SA12[t] * 3 + 1 : (SA12[t] - n0) * 3 + 2)
void suffix_array(int *s, int *SA, int n, int K, int level = 0) {
    int n0 = (n+2)/3, n1 = (n+1)/3, n2 = n/3, n02 = n0+n2;
    int *s12 = pool[level][0], *SA12 = pool[level][1];
    int *s0 = pool[level][2], *SA0 = pool[level][3];
    s12[n02] = s12[n02+1] = s12[n02+2] = 0; SA12[n02] = SA12[n02+1] = SA12[n02+2] = 0;
    for (int i = 0, j = 0; i < n+(n0-n1); i++) { if (i % 3 != 0) { s12[j++] = i; } }
    radix(s12, SA12, s+2, n02, K);
    radix(SA12, s12, s+1, n02, K);
    radix(s12, SA12, s+0, n02, K);
    int name = 0, c0 = -1, c1 = -1, c2 = -1;
    for (int i = 0; i < n02; i++) {
        if (s[SA12[i]] != c0 || s[SA12[i]+1] != c1 || s[SA12[i]+2] != c2)
        { name++; c0 = s[SA12[i]]; c1 = s[SA12[i]+1]; c2 = s[SA12[i]+2]; }
        if (SA12[i] % 3 == 1) { s12[SA12[i]/3] = name; }
        else { s12[SA12[i]/3 + n0] = name; }
    }
    if (name < n02) {
        suffix_array(s12, SA12, n02, name, level+1);
        for (int i = 0; i < n02; i++) { s12[SA12[i]] = i + 1; }
    } else {
        for (int i = 0; i < n02; i++) { SA12[s12[i]-1] = i; }
    }
    for (int i = 0, j = 0; i < n02; i++) { if (SA12[i] < n0) { s0[j++] = 3*SA12[i]; } }
```

```
    radix(s0, SA0, s, n0, K);
    for (int p = 0, t = n0-n1, k = 0; k < n; k++) {
        int i = GetI(); int j = SA0[p];
        if (SA12[t] < n0 ? leq(s[i], s12[SA12[t] + n0], s[j], s12[j/3])
                : leq(s[i], s[i+1], s12[SA12[t]-n0+1], s[j], s[j+1], s12[j/3+n0])) {
            SA[k] = i; t++;
            if (t == n02) { for (k++; p < n0; p++, k++) { SA[k] = SA0[p]; } }
        } else {
            SA[k] = j; p++;
            if (p == n0) { for (k++; t < n02; t++, k++) { SA[k] = GetI(); } }
        }
    }
}
void compute_lcp(int *str, int *sa, int *lcp, int n) {
    int *rank = tmp, i, j, h = 0;
    for (i = 0; i != n; ++i) { rank[sa[i]] = i; }
    for (i = 0; i != n; ++i) if (rank[i] > 0) {
        j = sa[rank[i]-1];
        while (str[i+h] == str[j+h]) { h++; }
        lcp[rank[i]] = h;
        if (h > 0) { h--; }
    }
}
int n, m, k;
int A[NMAX], sa[NMAX], lcp[NMAX];
int main() {
    scanf("%d %d %d", &n, &m, &k);
    for (int i = 0; i < n; i++) { scanf("%d", &A[i]); A[n+i] = A[i]; }
    n*= 2; A[n] = A[n+1] = A[n+2] = 0;
    suffix_array(A, sa, n, m-1); compute_lcp(A, sa, lcp, n);
    return 0;
}
```

4 Geometry

Graham's Scan Algorithm

graham.cpp

```
struct point{
    int x, y, id;
    int dist(const point& p) { return (p.x - x) * (p.x - x) + (p.y - y) * (p.y - y); }
    int left(const point& p, const point& q) {
        int l = (p.x - x)*(q.y - y) - (q.x - x)*(p.y - y);
        return l < 0 ? -1 : l > 0 ? 1 : 0;
    }
} PIVO;
bool cmp(const point& i, const point& j){
    int l = PIVO.left(i, j);
    if (l > 0 || (l == 0 && PIVO.dist(i) < PIVO.dist(j))) return true;
    return false;
}
void graham(struct point* P, int N, struct point* CH, int* CH_N) {
    int pivo = 0;
    for (int i = 1; i < N; i++)
        if (P[i].y < P[pivo].y || (P[i].y == P[pivo].y && P[i].x < P[pivo].x)) pivo = i;
    swap(P[pivo], P[0]);
    PIVO = P[0];
    sort(P + 1, P + N, cmp);
    int rev = N-1;
    for (int r = N-2; r >= 1; r--)
        if (PIVO.left(P[r], P[rev]) == 0) rev = r;
        else break;
    if (rev != 1) reverse(P + rev, P + N);
    *CH_N = 0;
    for (int i = 0; i < N; i++) {
        if (i > 1) {
            /* Se mudar a comparacao para '<', incluira todos os
               pontos colineares no convex-hull */
            while (*CH_N >= 2 && CH[*CH_N-2].left(CH[*CH_N-1],P[i]) <= 0) (*CH_N)--;
        }
        CH[(*CH_N)++] = P[i];
    }
}
point P[MAX], CH[MAX];
int CH_N, lbl[MAX];
bool pointInLine(point a, point b, point c) {
    if(b.y != c.y) return (min(b.y,c.y) <= a.y && a.y <= max(b.y, c.y));
    if(b.x != c.x) return (min(b.x,c.x) <= a.x && a.x <= max(b.x, c.x));
    return false;
}
bool cruza(point a, point b, point c, point d) {
    if (a.left(c,d) == 0 && b.left(c,d) == 0) {
        if (pointInLine(a,c,d)) return true;
        if (pointInLine(b,c,d)) return true;
        if (pointInLine(c,a,b)) return true;
        if (pointInLine(d,a,b)) return true;
        return false;
    }
    return (a.left(c,d) != b.left(c,d)) && (c.left(a,b) != d.left(a,b));
}
int main() {
    int N;
    while(scanf("%d", &N) == 1 && N) {
        for(int i = 0; i < N; i++) scanf("%d %d", &P[i].x, &P[i].y);
        graham(P, N, CH, &CH_N);
    }
}
```

Minimum Enclosing Disc

mindisc.cpp

```
struct point { double x, y; };
struct circle { point c; double r; };
point P[NMAX]; circle D[NMAX];
const double eps = 1e-9;
double dist2(point p,point q) { return (p.x-q.x)*(p.x-q.x) + (p.y-q.y)*(p.y-q.y); }
double dist(point p,point q) { return sqrt(dist2(p,q)); }
bool in_circle(point p,circle c){ return (dist2(p,c.c) <= c.r*c.r); }
point middle(point p,point q) { return (point) { (p.x + q.x)/2.0, (p.y + q.y)/2.0 }; }
point make_point(double x,double y) { return (point) { x, y }; }
circle make_circle(point p) { return (circle) { p, 0.0 }; }
circle make_circle (point p,point q) { return (circle) { middle(p, q), dist(p, q)/2.0 }; }
bool perpendicular(point a,point b,point c){
    double yDelta_a = b.y - a.y, xDelta_a = b.x - a.x,
           yDelta_b = c.y - b.y, xDelta_b = c.x - b.x;
    if (fabs(xDelta_a) <= eps && fabs(yDelta_b) <= eps) return false;
    if (fabs(yDelta_a) <= eps) return true; if (fabs(yDelta_b) <= eps) return true;
    if (fabs(xDelta_a) <= eps) return true; if (fabs(xDelta_b) <= eps) return true;
    return false;
}
circle calc_circle(point a,point b,point c) {
    double yDelta_a = b.y - a.y, xDelta_a = b.x - a.x,
           yDelta_b = c.y - b.y, xDelta_b = c.x - b.x;
    circle resp;
    if (fabs(xDelta_a) <= eps && fabs(yDelta_b) <= eps) {
        resp.c.x = 0.5*(b.x + c.x);
        resp.c.y = 0.5*(a.y + b.y);
        resp.r = dist(resp.c , a);
        return resp;
    }
    double aSlope = yDelta_a/xDelta_a;
    double bSlope = yDelta_b/xDelta_b;
    if (fabs(aSlope-bSlope) <= eps) {
        printf("ferrou!\n");
        return make_circle(a,b);
    }
    resp.c.x = (aSlope*bSlope*(a.y - c.y) + bSlope*(a.x + b.x) - aSlope*(b.x + c.x))/
        (2.0 * (bSlope-aSlope));
    resp.c.y = -1.0*(resp.c.x - (a.x + b.x)/2.0)/aSlope + (a.y + b.y)/2.0;
    resp.r = dist(resp.c , a);
    return resp;
}
circle make_circle(point a, point b, point c){
    if (!perpendicular(a,b,c)) return calc_circle(a,b,c);
    if (!perpendicular(a,c,b)) return calc_circle(a,c,b);
    if (!perpendicular(b,a,c)) return calc_circle(b,a,c);
    if (!perpendicular(b,c,a)) return calc_circle(b,c,a);
    if (!perpendicular(c,b,a)) return calc_circle(c,b,a);
    if (!perpendicular(c,a,b)) return calc_circle(c,a,b);
    printf("ferrou!\n");
    return make_circle(a,b);
}
circle MiniDiscWith2Points(int N,point p,point q){
    circle at = make_circle(p, q);
    for (int i = 0; i < N; i++) {
        if (i == 0 && in_circle(P[i],at)) D[i] = at;
        else if (i != 0 && in_circle(P[i],D[i-1])) D[i] = D[i-1];
        else D[i] = make_circle(p,q,P[i]);
    }
    return D[N-1];
}
circle MiniDiscWithPoint(int N, point q){
    D[0] = make_circle(P[0],q);
    for(int i = 1; i < N; i++){
        if (in_circle(P[i],D[i-1])) D[i] = D[i-1];
        else D[i] = MiniDiscWith2Points(i,P[i],q);
    }
    return D[N-1];
}
circle MiniDisc(int N){
    if(N <= 1) return make_circle(P[0]);
    random_shuffle(P, P+N);
    D[1] = make_circle(P[0],P[1]);
    for (int i = 2; i < N; i++){
        if (in_circle(P[i],D[i-1])) D[i] = D[i-1];
        else D[i] = MiniDiscWithPoint(i,P[i]);
    }
    return D[N-1];
}
int main(){
    int N; double x, y;
    while (scanf("%d",&N) == 1 && N){
        for (int i = 0;i < N;i++) { scanf("%lf %lf",&x,&y); P[i] = make_point(x,y); }
        if (test == -1){
            printf("%d\n",N);
            for (int i = 0;i < N;i++) printf("%.2lf %.2lf\n",P[i].x,P[i].y);
        }
        circle resp;
        if (N == 1) resp = make_circle(P[0]);
        else resp = MiniDisc(N);
        printf("Instancia %d\n",test++);
        printf("%.2lf %.2lf %.2lf\n\n",resp.c.x,resp.c.y,resp.r);
    }
    return 0;
}
```

Sweep Circle

sweepcircle.cpp

```
struct point { double x; double y; };
int n, m; point P[NMAX];
double dist2(point p,point q) { return (p.x-q.x)*(p.x-q.x) + (p.y-q.y)*(p.y-q.y); }
double dist(point p,point q) { return sqrt(dist2(p,q)); }
point middle(point p,point q) { return (point) { (p.x + q.x)/2.0, (p.y + q.y)/2.0 }; }
point make_point(double x,double y) { return (point) { x, y }; }
struct ATOL { int id; bool entra; double ang; point c; };
```

```
vector<ATOL> eventos;
bool cmp(ATOL a, ATOL b) { return a.ang < b.ang; }
ATOL make_ATOL(int id, bool entra, double ang, point c) {
    return (ATOL) {id, entra, ang, c};
}
double anguloAtan2(point centro, point novo) {
    double y = novo.y - centro.y, x = novo.x - centro.x; return atan2(y,x);
}
//encontra onde deve estar o centro da sweep quando o ponto b for entrar e sair da sweep.
pair< point, point > calcula_pontos_entrada_e_saida_da_sweep( point a, point b, double r) {
    double d = dist(a, b), anda = sqrt( r*r - ((d*d)/4) );
    double dx = b.x - a.x, dy = b.y - a.y;
    double normaliza = sqrt(dx*dx + dy*dy);
    dx /= normaliza; dy /= normaliza;
    double salvo = dx;
    dx = -dy; dy = salvo;
    point medio = middle(a,b), c1 = medio, c2 = medio;
    c1.x += anda*dx; c1.y += anda*dy;
    c2.x -= anda*dx; c2.y -= anda*dy;
    return make_pair(c1,c2);
}
void calcula_eventos(int id, double raio){
    eventos.clear();
    point a = P[id];
    for (int i = 0; i < n; i++) {
        if (i == id) continue;
        if (dist(a, P[i]) > 2*raio) continue;
        pair<point,point> intersecoes = calcula_pontos_entrada_e_saida_da_sweep( a, P[i], raio);
        eventos.push_back(make_ATOL(i, false,
                    anguloAtan2(a,intersecoes.first), intersecoes.first));
        eventos.push_back(make_ATOL(i, true,
                    anguloAtan2(a,intersecoes.second), intersecoes.second));
    }
}
int dentro[NMAX];
int sweepGiratoria(int id,  double r) {
    calcula_eventos(id, r);
    if (eventos.empty()) return 1; //cobre soh ele mesmo
    sort(eventos.begin(), eventos.end(), cmp);
    for (int i = 0; i < n; i++) dentro[i] = 0;
    int cnt = 0;
    point inicio = eventos[0].c;
    for(int i = 0; i < n; i++)
        if (dist(P[i], inicio) <= r + 1e-9)  {
            dentro[i] = 1; cnt++;
        }
    int output = cnt;
    for (int i = 0; i < (int) eventos.size(); i++) {
        if (eventos[i].entra)
            if (!dentro[ eventos[i].id ]) {
                dentro[ eventos[i].id ] = 1; cnt++;
            }
        else if (dentro[ eventos[i].id ]) {
                dentro[ eventos[i].id ] = 0; cnt--;
            }
        output = max(output, cnt);
    }
    return output;
}
int main(){
    while(scanf("%d %d",&n, &m) == 2) {
        if(n == 0 && m == 0) break;
        for(int i = 0;i < n;i++){
            double x,y; scanf("%lf %lf",&x,&y); P[i] = make_point(x,y);
        }
        double lower = 0.0, upper = 100000.0, meio;
        for(int it = 0; it <= 60; ++it) {
            if (upper - lower <= 1e-5) break;
            meio = (lower+upper)/2;
            int cobreMax = 0;
            for(int i = 0; i < n && cobreMax < m; ++i)
                cobreMax = max(cobreMax, sweepGiratoria(i ,meio));
            if(cobreMax >= m) upper = meio;
            else lower = meio;
        }
        printf("%.3lf\n", (lower + upper)/2);
    }
}
```

Intersections

intersections.cpp

```
void intersecaoCirculoReta() {
    double r, a, b, c; // circulo de raio r, e reta ax+by+c=0
    double x0 = -a*c/(a*a+b*b),  y0 = -b*c/(a*a+b*b);
    if (c*c > r*r*(a*a+b*b)+EPS)
        puts ("no points");
    else if (abs (c*c - r*r*(a*a+b*b)) < EPS) {
        puts ("1 point");
        cout << x0 << ' ' << y0 << '\n';
    }
    else {
        double d = r*r - c*c/(a*a+b*b);
        double mult = sqrt (d / (a*a+b*b));
        double ax,ay,bx,by;
        ax = x0 + b * mult;
        bx = x0 - b * mult;
        ay = y0 - a * mult;
        by = y0 + a * mult;
        puts ("2 points");
        cout << ax << ' ' << ay << '\n' << bx << ' ' << by << '\n';
    }
}
void interCirculoCirculo() {
    //circulo 1 na origem
```

```
    //Ax + By + C = 0;
    A = -2x2;
    B = -2y2;
    C = x2^2 + y2^2 + r1^2 - r2^2;
}
```

Lines

retas2.cpp

```
typedef long long ll;
struct P { double x, y; P() {}; P( double q, double w ) : x( q ), y( w ) {} };
struct RETA {
    int a, b, c;
    void init( int A, int B, int C ) {
        if( A < 0 || A == 0 && B < 0 ) { A = -A; B = -B; C = -C; }
        int d = A ?
            gcd( gcd( abs( A ), abs( B ) ), C ) :
            ( B || C ? gcd( abs( B ), C ) : 1 );
        a = A / d; b = B / d; c = C / d;
    }
    RETA() {}
    RETA( int A, int B, int C ) { init( A, B, C ); }
    RETA( P p, P q ) { init( q.y - p.y, p.x - q.x, p.x * q.y - p.y * q.x ); }
    bool operator<( const RETA &l ) const {
        return a < l.a || a == l.a && ( b < l.b || b == l.b && c < l.c );
    }
};
struct RETA {
    ll A,B,C;
    bool operator==( const RETA & x ) {
        if(x.A != A) return 0;
        if(x.B != B) return 0;
        if(x.C != C) return 0;
        return 1;
    }
    bool operator < (const RETA & x) const {
        if(A != x.A) return A < x.A;
        if(B != x.B) return B < x.B;
        if(C != x.C) return C < x.C;
        return 0;
    }
};
int Inter(RETA X, RETA Y) {
    ll A1 = X.A; ll A2 = Y.A;
    ll B1 = X.B; ll B2 = Y.B;
    ll C1 = X.C; ll C2 = Y.C;
    pii res;
    double det = A1*B2 - A2*B1;
    if(det == 0){
        //Lines are parallel
        return 0;
    }else{
        double x = (B2*C1 - B1*C2)/det;
        double y = (A1*C2 - A2*C1)/det;
        res = pii(x,y);
        return 1;
    }
}
ll left(P a, P b, P c) {
    return (a.first - c.first)*(b.second - c.second) - (a.second - c.second)*(b.first - c.first);
}
RETA Perpendicular(RETA r, pii a) {
    RETA nova; nova.init(-r.B, r.A, -A * a.first - B * a.second); return nova;
}
int naReta(RETA r, pii p) {
    return r.A * p.first + r.B * p.second + r.C == 0;
}
double polarAngle( P p ) {
    if( fabs( p.x ) <= EPS && fabs( p.y ) <= EPS ) return -1.0;
    if( fabs( p.x ) <= EPS ) return ( p.y > EPS ? 1.0 : 3.0 ) * acos( 0 );
    double theta = atan( 1.0 * p.y / p.x );
    if( p.x > EPS ) return( p.y >= -EPS ? theta : ( 4 * acos( 0 ) + theta ) );
    return( 2 * acos( 0 ) + theta );
}
bool pointInPoly( P p, vector< P > &poly ) {
    int n = poly.size();
    double ang = 0.0;
    for( int i = n - 1, j = 0; j < n; i = j++ ) {
        P v( poly[i].x - p.x, poly[i].y - p.y );
        P w( poly[j].x - p.x, poly[j].y - p.y );
        double va = polarAngle( v );
        double wa = polarAngle( w );
        double xx = wa - va;
        if( va < -0.5 || wa < -0.5 || fabs( fabs( xx ) - 2 * acos( 0 ) ) < EPS ) {
            // POINT IS ON THE EDGE
            assert( false );
            ang += 2 * acos( 0 );
            continue;
        }
        if( xx < -2 * acos( 0 ) ) ang += xx + 4 * acos( 0 );
        else if( xx > 2 * acos( 0 ) ) ang += xx - 4 * acos( 0 );
        else ang += xx;
    }
    return( ang * ang > 1.0 );
}
double distToLine(double ax, double ay,double bx, double by,
double px, double py, double *cpx, double *cpy ) {
    //Formula: cp = a + (p-a).(b-a) / |b-a| * (b-a)
    double proj = ( ( px - ax ) * ( bx - ax ) + ( py - ay ) * ( by - ay ) ) /
                  ( ( bx - ax ) * ( bx - ax ) + ( by - ay ) * ( by - ay ) );
    *cpx = ax + proj * ( bx - ax ); *cpy = ay + proj * ( by - ay );
    return dist( px, py, *cpx, *cpy );
}
double distToLineSegment(double ax, double ay,double bx, double by,
double px, double py,double *cpx, double *cpy ) {
```

```
    if( ( bx - ax ) * ( px - ax ) + ( by - ay ) * ( py - ay ) < EPS ) {
        *cpx = ax; *cpy = ay;
        return dist( ax, ay, px, py );
    }
    if( ( ax - bx ) * ( px - bx ) + ( ay - by ) * ( py - by ) < EPS ) {
        *cpx = bx;*cpy = by;
        return dist( bx, by, px, py );
    }
    return distToLine( ax, ay, bx, by, px, py, cpx, cpy );
}
bool lineIntersect( P a, P b, P c, P d, P &r ) {
    P n; n.x = d.y - c.y; n.y = c.x - d.x;
    double denom = n.x * ( b.x - a.x ) + n.y * ( b.y - a.y );
    if( fabs( denom ) < EPS ) return false;
    double num = n.x * ( a.x - c.x ) + n.y * ( a.y - c.y );
    double t = -num / denom;
    r.x = a.x + t * ( b.x - a.x );
    r.y = a.y + t * ( b.y - a.y );
    return true;
}
template< class T > bool lineSegIntersect( vector< T > &x, vector< T > &y ) {
    double ucrossv1 = ( x[1] - x[0] ) * ( y[2] - y[0] ) - ( y[1] - y[0] ) * ( x[2] - x[0] );
    double ucrossv2 = ( x[1] - x[0] ) * ( y[3] - y[0] ) - ( y[1] - y[0] ) * ( x[3] - x[0] );
    if( ucrossv1 * ucrossv2 > 0 ) return false;
    double vcrossu1 = ( x[3] - x[2] ) * ( y[0] - y[2] ) - ( y[3] - y[2] ) * ( x[0] - x[2] );
    double vcrossu2 = ( x[3] - x[2] ) * ( y[1] - y[2] ) - ( y[3] - y[2] ) * ( x[1] - x[2] );
    return( vcrossu1 * vcrossu2 <= 0 );
}
```

3D Geometry

geometria3D.cpp

```
/* Geometria 3D */
#include <stdio.h>
#include <math.h>
#include <algorithm>
using namespace std;
const double epsilon = 1e-11;
/* PONTOS E VETORES */
struct ponto {
    double x, y, z;
    ponto(double X = 0, double Y = 0, double Z = 0): x(X), y(Y), z(Z) { }
};
struct vetor {
    double x, y, z;
    vetor(double X = 0, double Y = 0, double Z = 0): x(X), y(Y), z(Z) { }
    vetor(ponto p) { x = p.x; y = p.y; z = p.z; }
    vetor(ponto p, ponto q) { x = q.x - p.x; y = q.y - p.y; z = q.z - p.z; }
};
ponto operator + (const ponto &p, const vetor &v) {
    return ponto(p.x + v.x, p.y + v.y, p.z + v.z); }
ponto operator + (const ponto &p, const ponto &q) {
    return ponto(p.x + q.x, p.y + q.y, p.z + q.z); }
ponto operator - (const ponto &p, const vetor &v) {
    return ponto(p.x - v.x, p.y - v.y, p.z - v.z); }
ponto operator - (const ponto &p, const ponto &q) {
    return ponto(p.x - q.x, p.y - q.y, p.z - q.z); }
vetor operator + (const vetor &u, const vetor &v) {
    return vetor(u.x + v.x, u.y + v.y, u.z + v.z); }
vetor operator - (const vetor &u, const vetor &v) {
    return vetor(u.x - v.x, u.y - v.y, u.z - v.z); }
vetor operator * (const double &a, const vetor &v) {
    return vetor(a * v.x, a * v.y, a * v.z); }
double dot(const vetor u, const vetor v) { return u.x * v.x + u.y * v.y + u.z * v.z; }
vetor cross(const vetor u, const vetor v) {
    return vetor(u.y * v.z - u.z * v.y, u.z * v.x - u.x * v.z, u.x * v.y - u.y * v.x);
}
double norma(const vetor v) { return sqrt(dot(v, v)); }
void debug(vetor v) { printf("(%.2f, %.2f, %.2f)\n", v.x, v.y, v.z); }
/* RETAS, SEMIRETAS, SEGMENTOS E TRIANGULOS */
struct reta {
    ponto a, b;
    reta(ponto A, ponto B): a(A), b(B) { }
    reta(ponto P, vetor V): a(P) { b = P + V; }
};
struct semireta {
    ponto a, b;
    semireta(ponto A, ponto B): a(A), b(B) { }
    semireta(ponto P, vetor V): a(P) { b = P + V; }
};
struct segmento {
    ponto a, b;
    segmento(ponto A, ponto B): a(A), b(B) { }
};
struct triangulo {
    ponto a, b, c;
    triangulo(ponto A, ponto B, ponto C): a(A), b(B), c(C) { }
};
/* DISTANCIA ENTRE OBJETOS GEOMETRICOS */
double distancia(const ponto a, const ponto b) {
    return norma(vetor(a, b));
}
double distancia(const ponto p, const reta r) {
    vetor v(r.a, r.b), w(r.a, p);
    return norma(cross(v, w)) / norma(v);
}
double distancia(const ponto p, const semireta s) {
    vetor v(s.a, s.b), w(s.a, p);
    if (dot(v, w) <= 0) return distancia(p, s.a);
    return distancia(p, reta(s.a, s.b));
}
double distancia(const ponto p, const segmento s) {
    vetor v(s.a, s.b), w(s.a, p);
    double c1 = dot(v, w), c2 = dot(v, v);
    if (c1 <= 0) return distancia(p, s.a);
```

```
    if (c2 <= c1) return distancia(p, s.b);
    return distancia(p, s.a + (c1/c2)*v);
}
double distancia(const reta r, const reta s) {
    vetor u(r.a, r.b), v(s.a, s.b), w(r.a, s.a);
    double a = dot(u, u), b = dot(u, v), c = dot(v, v), d = dot(u, w), e = dot(v, w);
    double D = a*c - b*b, sc, tc;
    if (D < epsilon) {
        sc = 0;
        tc = (b > c) ? d/b : e/c;
    } else {
        sc = (b*e - c*d) / D;
        tc = (a*e - b*d) / D;
    }
    vetor dP = w + (sc * u) - (tc * v);
    return norma(dP);
}
double distancia(const segmento r, const segmento s) {
    vetor u(r.a, r.b), v(s.a, s.b), w(s.a, r.a);
    double a = dot(u, u), b = dot(u, v), c = dot(v, v), d = dot(u, w), e = dot(v, w);
    double D = a*c - b*b;
    double sc, sN, sD = D;
    double tc, tN, tD = D;
    if (D < epsilon) {
        sN = 0;
        sD = 1;
        tN = e;
        tD = c;
    } else {
        sN = (b*e - c*d);
        tN = (a*e - b*d);
        if (sN < 0) {
            sN = 0;
            tN = e;
            tD = c;
        } else if (sN > sD) {
            sN = sD;
            tN = e + b;
            tD = c;
        }
    }
    if (tN < 0) {
        tN = 0;
        if (-d < 0) {
            sN = 0;
        } else if (-d > a) {
            sN = sD;
        } else {
            sN = -d;
            sD = a;
        }
    } else if (tN > tD) {
        tN = tD;
        if ((-d + b) < 0) {
            sN = 0;
        } else if (-d + b > a) {
            sN = sD;
        } else {
            sN = -d + b;
            sD = a;
        }
    }
    sc = fabs(sN) < epsilon ? 0 : sN / sD;
    tc = fabs(tN) < epsilon ? 0 : tN / tD;
    vetor dP = w + (sc * u) - (tc * v);
    return norma(dP);
}
/* Inicio das funcoes do ITA no G da subregional */
vetor projecao(vetor u, vetor v) {
    return (dot(v, u) / dot(u, u)) * u;
}
bool between(ponto a, ponto b, ponto p) {
    return dot(vetor(p - a), vetor(p - b)) < epsilon;
}
double linedist(ponto a, ponto b, ponto p) {
    ponto proj = a + projecao(vetor(a, b), vetor(a, p));
    if (between(a, b, proj)) {
        return norma(vetor(proj, p));
    } else {
        return min(norma(vetor(a, p)), norma(vetor(b, p)));
    }
}
double distancia(const ponto p, const triangulo T) {
    vetor X(T.a, T.b), Y(T.a, T.c), P(T.a, p);
    vetor PP = P - projecao(cross(X, Y), P);
    ponto PPP = T.a + PP;
    vetor R1 = cross(vetor(T.a, T.b), vetor(T.a, PPP));
    vetor R2 = cross(vetor(T.b, T.c), vetor(T.b, PPP));
    vetor R3 = cross(vetor(T.c, T.a), vetor(T.c, PPP));
    if (dot(R1, R2) > -epsilon && dot(R2, R3) > -epsilon && dot(R1, R3) > -epsilon) {
        return norma(vetor(PPP, p));
    } else {
        return min(linedist(T.a, T.b, p), min(linedist(T.b, T.c, p), linedist(T.c, T.a, p)));
    }
}
/* Fim das funcoes do ITA no G da subregional */
#define NMAX 4
ponto P[NMAX], Q[NMAX];
int main() {
    int tests;
    scanf("%d", &tests);
    for (int test = 0; test < tests; test++) {
        int x, y, z;
        for (int i = 0; i < 4; i++) {
            scanf("%d %d %d", &x, &y, &z);
            P[i] = ponto(x, y, z);
        }
        for (int i = 0; i < 4; i++) {
            scanf("%d %d %d", &x, &y, &z);
            Q[i] = ponto(x, y, z);
```

```
        }
        double out = 1000000000.0;
        for (int i = 0; i < 4; i++) {
            for (int j = i+1; j < 4; j++) {
                segmento r(P[i], P[j]);
                for (int k = 0; k < 4; k++) {
                    for (int l = k+1; l < 4; l++) {
                        segmento s(Q[k], Q[l]);
                        out = min(out, distancia(r, s));
                    }
                }
            }
        }
        for (int i = 0; i < 4; i++) {
            for (int j = i+1; j < 4; j++) {
                for (int k = j+1; k < 4; k++) {
                    triangulo t(P[i], P[j], P[k]), u(Q[i], Q[j], Q[k]);
                    for (int l = 0; l < 4; l++) {
                        out = min(out, distancia(P[l], u));
                        out = min(out, distancia(Q[l], t));
                    }
                }
            }
        }
        printf("%.2f\n", out);
    }
}
```

Centroid

centroide.cpp

```
/* Calcula centroide (centro de massa / gravidade) de poligono em O(n) */
double x[NMAX], y[NMAX];
x[n] = x[0];
y[n] = y[0];
double area = 0;
for (int i = 0; i < n; i++) {
    area+= x[i]*y[i+1] - y[i]*x[i+1];
}
double cx = 0, cy = 0;
for (int i = 0; i < n; i++) {
    cx+= (x[i] + x[i+1]) * (x[i]*y[i+1] - x[i+1]*y[i]);
    cy+= (y[i] + y[i+1]) * (x[i]*y[i+1] - x[i+1]*y[i]);
}
cx/= 3.0*area;
cy/= 3.0*area;
printf("%.2f %.2f\n", cx, cy);
```

Voronoi

voronoi.cpp

```
int T;
int X,Y,M;
int cmp(double a, double b) {
    if(fabs(a-b)<=1e-8)
        return 0;
    if(a<b)
        return -1;
    return 1;
}
struct point {
    double x,y;
    point() {}
    point(double a, double b) : x(a), y(b) {}
    point operator+(const point& a) const {
        return point(x+a.x,y+a.y);
    }
    point operator-(const point& a) const {
        return point(x-a.x,y-a.y);
    }
    point operator/(double a) const {
        return point(x/a,y/a);
    }
};
double sqrdist(const point& a, const point &b) {
    return (a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y);
}
struct line {
    double a,b,c;
    line() {}
    line(const point& a1,const point& a2) {
        a=a1.y-a2.y;
        b=a2.x-a1.x;
        c=a1.x*a2.y-a2.x*a1.y;
    }
    line(double a, double b, double c) : a(a), b(b), c(c) {}
    line(double ang) : a(-ang), b(1.0), c(0.0) { }
    line passa_por(const point& p) const {
        return line(a,b,-a*p.x-b*p.y);
    }
    line ortogonal() const {
        return line(-b,a,c);
    }
    bool paralela(const line& l) const {
        return cmp((-a*l.b),(-l.a*b))==0;
    }
};
point intersection(const line& l, const line& l2) {
    return point((l.b*l2.c-l.c*l2.b)/(l.a*l2.b-l2.a*l.b),(-l.a*l2.c+l.c*l2.a)/(l.a*l2.b-l2.a*l.b));
}
point pontos[2000];
point polygon[2000];
```

```
int npolygon;
point polaux[2000];
int npolaux;
int main(void) {
    scanf("%d",&T);
    while(T--) {
        scanf("%d %d %d",&X,&Y,&M);
        for(int i=0;i<M;i++) scanf("%lf %lf",&pontos[i].x,&pontos[i].y);
        double ansd=-1;
        point ans;
        for(int i=0;i<M;i++) {
            npolygon=4;
            polygon[0]=point(0,0);
            polygon[1]=point(0,Y);
            polygon[2]=point(X,Y);
            polygon[3]=point(X,0);
            for(int j=0;j<M;j++) if(pontos[i].x!=pontos[j].x or pontos[i].y!=pontos[j].y) {
                line biseccao = line(pontos[i],pontos[j]).ortogonal()
                    .passa_por((pontos[i]+pontos[j])/2.0);
                int inter[8];
                int ninter=0;
                for(int k=0;k<npolygon;k++) {
                    int l=(k+1)%npolygon;
                    double l1=biseccao.a*polygon[k].x + biseccao.b*polygon[k].y + biseccao.c;
                    double l2=biseccao.a*polygon[l].x + biseccao.b*polygon[l].y + biseccao.c;
                    /* considera que a ponta do comeco intersecta e
                     * a ponta do final nao intersecta */
                    if( (cmp(l1,0.0)<=0 and cmp(l2,0.0)>0) or
                            (cmp(l1,0.0)>=0 and cmp(l2,0.0)<0) ) {
                        inter[ninter++]=k;
                    }
                }
                assert(ninter<=2);
                if(ninter==2) {
                    double lp=biseccao.a*pontos[i].x + biseccao.b*pontos[i].y + biseccao.c;
                    double lpol=biseccao.a*polygon[(inter[0]+1)%npolygon].x
                        + biseccao.b*polygon[(inter[0]+1)%npolygon].y + biseccao.c;
                    //verifica se a ordem do loop precisa ser trocada
                    if( (cmp(lp,0)<0 and cmp(lpol,0)>0) or (cmp(lp,0)>0 and cmp(lpol,0)<0) )
                        swap(inter[0],inter[1]);
                    npolaux=0;
                    polaux[npolaux++]=intersection(biseccao,
                            line(polygon[inter[0]],polygon[(inter[0]+1)%npolygon]));
                    for(int k=(inter[0]+1)%npolygon;k!=inter[1];k=(k+1)%npolygon) {
                        polaux[npolaux++]=polygon[k];
                    }
                    polaux[npolaux++]=polygon[inter[1]];
                    polaux[npolaux++]=intersection(biseccao,
                            line(polygon[inter[1]],polygon[(inter[1]+1)%npolygon]));
                    //se os 2 ultimos pontos coincidirem
                    if(cmp(polaux[npolaux-1].x,polaux[npolaux-2].x)==0
                            and cmp(polaux[npolaux-1].y,polaux[npolaux-2].y)==0)
                        npolaux--;
                    memcpy(polygon,polaux,sizeof(polygon[0])*npolaux);
                    npolygon=npolaux;
                }
            }
            for(int j=0;j<npolygon;j++)
                if(sqrdist(pontos[i],polygon[j]) > ansd) {
                    ansd=sqrdist(pontos[i],polygon[j]);
                    ans=polygon[j];
                }
        }
        printf("The safest point is (%.1lf, %.1lf).\n",ans.x,ans.y);
    }
}
```

5 Graphs

MaxFlow–MinCost

mcmf2.cpp

```
typedef long long ll;
#define MAX 256
struct flow{
    ll cost, flow;
};
struct aresta{
    int v; long long cst, cap, flow;
    aresta(): v(0), cst(0), cap(0), flow(0) {}
    aresta(int _v, long long _cap, long long _cst, long long _flow):
        v(_v), cap(_cap), cst(_cst), flow(_flow){}
};
int fila[2][MAX], lbl[2][MAX], qf[2];
long long dist[MAX];
aresta arestas[MAX*MAX*2];
int grau[MAX], pai[MAX], adj[MAX][MAX], NV, S, T, e_contador;
const int inf = 1000000000;
struct MCMF {
    void inic(int n) {
        NV = n+2; S = n; T = n+1; e_contador = 0;
        for(int i = 0; i < NV; i++) grau[i] = 0;
    }
    void insere(int v1, int v2, int cap, int cst) {
        arestas[e_contador++] = aresta(v2, cap, cst, 0);
        arestas[e_contador++] = aresta(v1, 0, -cst, 0);
        adj[v1][ grau[v1]++ ] = e_contador - 2;
        adj[v2][ grau[v2]++ ] = e_contador - 1;
    }
    void insereDoS(int v2, int cap, int cst) {
        insere(S, v2, cap, cst);
    }
    void insereProT(int v1, int cap, int cst) {
        insere(v1, T, cap, cst);
    }
```

```
    bool belman() {
        for(int i = 0; i < NV; i++) {
            dist[i] = inf;
            lbl[0][i] = lbl[1][i] = 0;
            pai[i] = -1;
        }
        qf[0] = 0; fila[0][qf[0]++] = S;
        pai[S] = -2; dist[S] = 0;
        for(int k = 0; k < NV; k++) {
            int fila_at = k&1;
            int fila_prox = 1 - fila_at;
            qf[fila_prox] = 0;
            for(int i = 0; i < qf[fila_at]; i++) {
                int v = fila[fila_at][i];
                lbl[fila_at][v] = 0;
                for(int j = 0; j < grau[v]; j++) {
                    aresta e = arestas[adj[v][j]];
                    aresta ei = arestas[adj[v][j] ^ 1];
                    int w = e.v;
                    ll cap = e.cap - e.flow, cst = e.cst;
                    if(ei.flow) cst = -ei.cst;
                    if(cap > 0 && dist[w] > dist[v] + cst) {
                        if(!lbl[fila_prox][w]){
                            fila[fila_prox][qf[fila_prox]++] = w;
                            lbl[fila_prox][w] = 1;
                        }
                        dist[w] = dist[v] + cst;
                        pai[w] = adj[v][j];
                    }
                }
            }
        }
        return pai[T] != -1;
    }
    flow mcmf() {
        flow f = (flow) { 0 , 0 };
        while(belman()) {
            ll bot = inf, cst = 0;
            for(int v = T; pai[v] >= 0; v = arestas[pai[v] ^ 1].v) {
                bot = min (bot , arestas[pai[v]].cap - arestas[pai[v]].flow);
            }
            for(int v = T; pai[v] >= 0; v = arestas[pai[v] ^ 1].v) {
                aresta &e = arestas[pai[v]];
                aresta &ei = arestas[pai[v] ^ 1];
                if(ei.flow) { cst += bot*(-ei.cst); }
                else { cst += bot*e.cst; }
                e.flow += bot;
                ei.flow -= bot;
            }
            f.flow += bot;
            f.cost += cst;
        }
        return f;
    }
};
```

```
class SlimeXGrandSlimeAuto {
    public:
        int travel(vector <int> cars, vector <int> districts, ...) {
            MCMF grafo;
            int A = districts.size() - 1;
            int B = cars.size() + 1;
            grafo.inic(A + B + 2);
            FOR(i, sz(districts) - 1) {
                grafo.insereDoS(i, 1, 0);
            }
            for(int i = A; i < A+B; i++) {
                if(i == A) {
                    grafo.insereProT(i, A, 0);
                } else {
                    grafo.insereProT(i, 1, 0);
                }
            }
            FOR(i, sz(districts) - 1) {
                grafo.insere(i, A, 1, inverseWalkSpeed * custo[ districts[i] ][ districts[i+1] ]);
                FOR(j, sz(cars)) {
                    int custo=inverseWalkSpeed*custo[districts[i]][cars[j]]+inverseDriveSpeed*custo
                    grafo.insere(i, A + j + 1, 1, custo);
                }
            }
            return grafo.mcmf().cost;
        }
};
```

MaxFlow

maxflow.cpp

```
//OBSERVACAO: Sempre colocar as arestas de ida e volta.
using namespace std;
vector<int> grafo[MAX];
int cap[MAX][MAX];
int parnt[MAX];
int flow(int s, int t) {
    int f = 0;
    while(true) {
        memset(parnt,-1,sizeof(parnt));
        parnt[s] = -2;
        queue<int> q;
        q.push(s);
        while(!q.empty() && parnt[t] == -1) {
            int v = q.front(); q.pop();
            for(int i = 0, w; i < grafo[v].size(); i++) {
                if(parnt[w = grafo[v][i]] == -1 && cap[v][w] > 0){
                    parnt[w] = v;
                    q.push(w);
                }
```

```
            }
        }
        if(parnt[t] == -1) break;
        int bot = inf;
        for(int v = t, w = parnt[t]; w >= 0; v = w, w = parnt[w]) {
            bot = min(bot,cap[w][v]);
        }
        for(int v = t, w = parnt[t]; w >= 0; v = w, w = parnt[w]) {
            cap[w][v] -= bot;
            cap[v][w] += bot;
        }
        f += bot;
    }
    return f;
}
```

Hopcroft–Karp

hopcroft.cpp

```
/*
 * ALGORITMO DE HOPCROFT-KARP
 * Encontra emparelhamento maximo num grafo bipartido em O(E sqrt(V))
 */
#include <stdio.h>
#include <string.h>
#include <queue>
#include <vector>
using namespace std;
#define MMAX 1024
#define NMAX 1024
int UMATE[MMAX], M[MMAX], D[MMAX];
int VMATE[NMAX], RET[NMAX];
int m, n, len;
vector <int> E[MMAX];
void bfs() {
    queue <int> Q;
    len = m;
    memset(M, 0, sizeof(M));
    for (int u = 0; u < m; u++) {
        if (UMATE[u] == -1) {
            Q.push(u);
            M[u] = 1;
            D[u] = 0;
        }
    }
    while (!Q.empty()) {
        int u = Q.front(); Q.pop();
        for (int i = 0; i < E[u].size(); i++) {
            int v = E[u][i];
            int w = VMATE[v];
            if (w == -1) {
                len = D[u];
                return;
            } else if (!M[w]) {
                Q.push(w);
                M[w] = 1;
                D[w] = D[u] + 1;
            }
        }
    }
}
void augment(int v) {
    do {
        int u = RET[v];
        int w = UMATE[u];
        UMATE[u] = v;
        VMATE[v] = u;
        v = w;
    } while (v != -1);
}
int dfs(int u) {
    M[u] = 1;
    for (int i = 0; i < E[u].size(); i++) {
        int v = E[u][i];
        int w = VMATE[v];
        if (w == -1) {
            if (D[u] == len) {
                RET[v] = u;
                augment(v);
                return 1;
            }
        } else if (D[u] < D[w] && !M[w]) {
            RET[v] = u;
            if (dfs(w)) {
                return 1;
            }
        }
    }
    return 0;
}
int match() {
    memset(UMATE, -1, sizeof(UMATE));
    memset(VMATE, -1, sizeof(VMATE));
    for (int u = 0; u < m; u++) {
        for (int i = 0; i < E[u].size(); i++) {
            int v = E[u][i];
            if (VMATE[v] == -1) {
                UMATE[u] = v;
                VMATE[v] = u;
                break;
            }
        }
    }
    while (bfs(), len != m) {
        memset(M, 0, sizeof(M));
```

```
        for (int u = 0; u < m; u++) {
            if (UMATE[u] == -1) {
                dfs(u);
            }
        }
    }
    int matching = 0;
    for (int u = 0; u < m; u++) {
        if (UMATE[u] != -1) {
            matching++;
        }
    }
    return matching;
}
```

Maximum Match (Edmonds–Karp)

EdmondsMaximumMatch.cpp

```
#include <iostream>
#include <string.h>
#include<stdio.h>
using namespace std;
#define maxN 300
int n,match[maxN],Head, Tail,Queue[maxN],Start, Finish,NewBase,Father[maxN],Base[maxN],Count;
bool graph[maxN][maxN],InQueue[maxN], InPath[maxN], InBlossom[maxN];

void CreateGraph() {
    int u,v;
    memset(graph,0,sizeof(graph));
    scanf("%d",&n);
    while(scanf("%d%d",&u,&v)!=EOF) {
        if(u==-1) break;
        graph[u][v]=graph[v][u]=1;
    }
}
void Push(int u) {
    Queue[Tail++]= u;
    InQueue[u]= true;
}
int Pop() {
    return Queue[Head++];
}
int FindCommonAncestor(int u, int v) {
    memset(InPath, 0, sizeof(InPath));
    while(true) {
        u=Base[u];
        InPath[u]= true;
        if (u == Start)break;
        u= Father[match[u]];
    }
    while(true) {
        v= Base[v];
        if (InPath[v]) break;
        v = Father[match[v]];
    }
    return v;
}
void ResetTrace(int u) {
    int v;
    while (Base[u] != NewBase) {
        v= match[u];
        InBlossom[Base[u]]= 1;
        InBlossom[Base[v]]= 1;
        u= Father[v];
        if (Base[u] != NewBase)Father[u]=v;
    }
}
void BlossomContract(int u,int v) {
    NewBase= FindCommonAncestor(u, v);
    memset(InBlossom,0 ,sizeof(InBlossom));
    ResetTrace(u);
    ResetTrace(v);
    if (Base[u] != NewBase)Father[u]= v;
    if (Base[v] != NewBase)Father[v]= u;
    for(u=1;u<=n;u++) if (InBlossom[Base[u]]) {
        Base[u]= NewBase;
        if (!InQueue[u]) Push(u);
    }
}
void FindAugmentingPath() {
    int u,v;
    memset(InQueue,false, sizeof(InQueue));
    memset(Father,0,sizeof(Father));
    for(u=1;u<=n;u++) Base[u]=u;
    Head= 1;
    Tail= 1;
    Push(Start);
    Finish = 0;
    while (Head < Tail) {
        u= Pop();
        for (v=1;v<=n;v++)
            if ((graph[u][v])&&(Base[u]!=Base[v])&&(match[u]!= v))
                if ((v == Start)||((match[v] > 0)&&(Father[match[v]] > 0)))
                    BlossomContract(u, v);
                else if (Father[v] == 0) {
                    Father[v]=u;
                    if (match[v] > 0)
                        Push(match[v]);
                    else {
                        Finish=v;
                        return;
                    }
                }
    }
}
```

```
void AugmentPath() {
    int u,v,w;
    u=Finish;
    while(u > 0) {
        v=Father[u]; w=match[v];
        match[v]= u; match[u]= v; u = w;
    }
}
void Edmonds() {
    int u;
    memset(match,0,sizeof(match));
    for(u=1;u<=n;u++)
        if (match[u]==0)
        {
            Start=u;
            FindAugmentingPath();
            if (Finish > 0) AugmentPath();
        }
}
void PrintMatch() {
    int u;
    for(u=1;u<=n;u++)
        if (match[u] > 0) Count++;
    printf("%d\n",Count);
    for(u=1;u<=n;u++)
        if (u < match[u]) printf("%d %d\n",u,match[u]);
}
int main() {
    CreateGraph();
    Edmonds();
    PrintMatch();
}
```

Stoer–Wagner

stoerwagner.cpp

```
#define N 128
int peso[N][N];
int dist[N], ja_saiu[N], foi_contraido[N];
int n,m;
void contrai( pii ultimos ) {
    int menor = min(ultimos.first, ultimos.second);
    int maior = max(ultimos.first, ultimos.second);
    FOR(i,n) {
        peso[menor][i] += peso[maior][i];
        peso[i][menor] = peso[menor][i];
        peso[i][maior] = peso[maior][i] = 0;
        peso[i][i] = 0;
    }
    foi_contraido[maior] = 1;
}
int pega_maior() {
    int id = -1;
    FOR(i,n) {
        if(foi_contraido[i] || ja_saiu[i]) continue;
        if(id == -1 || dist[i] > dist[id]) id = i;
    }
    return id;
}
int corte_fase(int k) {
    pii ultimos = pii(0, 0);
    FOR(i,n) { dist[i] = 0; ja_saiu[i] = 0; }
    FOR(i, k) {
        int v = pega_maior();
        ja_saiu[v] = 1;
        ultimos.second = ultimos.first;
        ultimos.first = v;
        FOR(j, n)
            dist[ j ] += peso[v][j];
    }
    contrai( ultimos );
    return dist[ ultimos.first ];
}
int stoer_wagner() {
    if(n == 1) return 0;
    int output = (1<<31) - 1;
    int nn = n;
    FOR(i,n) foi_contraido[i] = 0;
    FOR(i, n-1) {
        int aux = corte_fase(nn--);
        output = min(output, aux);
    }
    return output;
}
int main() {
    int T;
    scanf("%d", &T);
    while(T--) {
        scanf("%d %d", &n, &m);
        memset(peso, 0, sizeof(peso));
        FOR(i,m) {
            int a, b, c;
            scanf("%d %d %d", &a, &b, &c);
            a--; b--;
            peso[a][b] = c;
            peso[b][a] = c;
        }
        printf("%d\n", stoer_wagner());
    }
}
```

2-satisfability

2sat.cpp

```
#define N 4096
int n,m;
vector<pii> pares, portas;
vector<int> adj[N], inv[N];
int scc[N], pos[N], outro[N], cnt;
int nao(int x) { return outro[x]; }
int sim(int x) { return x; }
void insereAmbas(int a, int b) {
    adj[a].pb(b); inv[b].pb(a);
}
void insere(int a, int b) {
    insereAmbas( nao(a), sim(b) );
    insereAmbas( nao(b), sim(a) );
}
void dfsInv(int v) {
    if(scc[v]) return; scc[v] = 1;
    FOR(i, sz(inv[v])) dfsInv( inv[v][i] );
    pos[cnt++] = v;
}
void pinta(int v, int c) {
    if(scc[v] != -1) return;
    scc[v] = c;
    FOR(i ,sz(adj[v])) pinta(adj[v][i], c );
}
int raiz_componente[N], qtd_componentes, valor[N];
void valora(int v, int val) {
    if(valor[v] != -1) return;
    valor[v] = val;
    valora( outro[v], 1 - val );
    FOR(i, sz(adj[v])) {
        if(scc[adj[v][i]] == scc[v])
            valora( adj[v][i], val );
    }
}
int ok(int qtd) {
    FOR(i, 2*n) adj[i].clear(); inv[i].clear();
    FOR(i, qtd) insere( portas[i].first, portas[i].second );
    cnt = 0;
    memset(scc, 0, sizeof(scc));
    FOR(i, 2*n) dfsInv(i);
    memset(scc, -1, sizeof(scc));
    int cor = 0;
    qtd_componentes = 0;
    for(int i = cnt-1; i>= 0; i--) {
        if(scc[ pos[i] ] == -1) {
            pinta( pos[i] ,cor++);
            raiz_componente[ qtd_componentes++ ] = pos[i];
        }
    }
    FOR(i, n) {
        if( scc[ pares[i].first ] == scc[ pares[i].second ] ) return 0;
    }
    //valoracao
    memset(valor, -1, sizeof(valor));
    for(int i = 0; i < qtd_componentes; i++) {
        int raiz = raiz_componente[i];
        if(valor[ raiz ] == -1) {
            valora( raiz, 1 );
        }
    }
    return 1;
}

int main() {
    while(scanf("%d %d", &n, &m) == 2 && (n || m)) {
        pares.clear();
        FOR(i, n) {
            pii aux;
            scanf("%d %d", &aux.first, &aux.second);
            pares.pb(aux);
            outro[aux.first] = aux.second;
            outro[aux.second] = aux.first;
        }
        portas.clear();
        FOR(i, m) {
            pii aux;
            scanf("%d %d", &aux.first, &aux.second);
            portas.pb(aux);
        }
        int ini = 0, fim = m, meio;
        while(ini + 3 < fim) {
            meio = (ini+fim) / 2;
            if(ok(meio)) ini = meio;
            else fim = meio;
        }
        for(int i = fim; i >= ini; i--) {
            if(ok(i)) {
                printf("%d\n", i);
                break;
            }
        }
    }
    return 0;
}
```

Rooted MST in digraph

mstdigraph.cpp

```
#define NMAX 1024
vector< pair<int,int> > adj[NMAX], grafo[NMAX];
int n,m;
```

```
int ja[NMAX];
int peso[NMAX][NMAX];
int fora[NMAX];
void dfs(int v) {
    if(ja[v]) return;
    ja[v] = 1;
    for(int i = 0; i < (int) grafo[v].size(); i++) dfs(grafo[v][i].first);
}
int consegue() {
    memset(ja, 0, sizeof(ja));
    dfs(0);
    for(int i = 0; i < n; i++) if(!ja[i]) return 0;
    return 1;
}
pair<int,int> arco[NMAX];
int pai[NMAX];
int reconstroi_ciclo(int v, int fim, vector<int> & c) {
    c.clear();
    int x = v;
    while(x != fim) {
        c.push_back(x);
        x = pai[x];
    }
    c.push_back(fim);
    return 1;
}
int ciclo(int v, vector<int> & c) {
    ja[v] = 1;
    for(int i = 0; i < (int) adj[v].size(); i++) {
        int viz = adj[v][i].first;
        if(viz == v) continue;
        if(ja[viz] == 1) {
            return reconstroi_ciclo(v,viz,c);
        } else if(!ja[viz]) {
            pai[viz] = v;
            if(ciclo(viz, c)) return 1;
        }
    }
    ja[v] = 2;
    return 0;
}
int tem_ciclo(vector<int> & c) {
    memset(ja, 0, sizeof(ja));
    for(int i = 0; i < n; i++) {
        if(!ja[i] && ciclo(i,c)) {
            return 1;
        }
    }
    return 0;
}
int no_ciclo[NMAX];
void contrai(vector<int> c) {
    reverse(c.begin(), c.end());
    memset(no_ciclo, 0, sizeof(no_ciclo));
    int ciclo_custo = 0;
    int id_contraido = 0;
    for(int i = 0; i < (int)c.size(); i++) {
        ciclo_custo += arco[ c[i] ].first;
        no_ciclo[c[i]] = 1;
        if(c[i]) {
            id_contraido = c[i];
        }
    }
    /* arruma as arestas que saem do vertice contraido */
    for(int i = 0; i < (int) c.size(); i++) {
        int v = c[i];
        if(v != id_contraido) {
            for(int j = 0; j < (int) grafo[v].size(); j++) {
                if(!no_ciclo[grafo[v][j].first]) {
                    grafo[id_contraido].push_back( grafo[v][j] );
                    if( arco[ grafo[v][j].first ].second == v ) {
                        arco[ grafo[v][j].first ].second = id_contraido;
                    }
                }
            }
        } else {
            for(int j = (int) grafo[v].size() - 1; j >= 0; j--) {
                if(no_ciclo[ grafo[v][j].first ]) {
                    swap( grafo[v][j], grafo[v][ (int) grafo[v].size() - 1]);
                    grafo[v].pop_back();
                }
            }
        }
    }
    pair<int,int> novo_arco = make_pair( (1<<29) , -1);
    /* arruma as arestas que entram no vertice contraido */
    for(int i = 0; i < n; i++) {
        if(!no_ciclo[i]) {
            for(int j = (int) grafo[i].size() - 1; j >= 0; j--) {
                if(no_ciclo[ grafo[i][j].first ]) {
                    int novo_custo = ciclo_custo - arco[ grafo[i][j].first ].first + grafo[i][j].sec
                    grafo[i][j].second = novo_custo;
                    grafo[i][j].first = id_contraido;
                    novo_arco = min(novo_arco, make_pair(novo_custo, i));
                }
            }
        }
    }
    arco[id_contraido] = novo_arco;
    for(int i = 0; i < (int)c.size(); i++) {
        if(c[i] != id_contraido) {
            fora[c[i]] = 1;
            grafo[c[i]].clear();
        }
    }
}

int calcula() {
    memset(fora, 0, sizeof(fora));
    for(int i = 0; i < n; i++) {
```

```
        arco[i] = make_pair( (1<<29) , -1);
    }
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < (int) grafo[i].size(); j++) {
            int viz = grafo[i][j].first;
            int cst = grafo[i][j].second;
            if(viz) {
                arco[viz] = min(arco[viz], make_pair(cst,i));
            }
        }
    }
    while(1) {
        for(int i = 0; i < n; i++) {
            adj[i].clear();
        }
        for(int i = 1; i < n; i++) {
            if(fora[i] || fora[arco[i].second])continue;
            adj[ arco[i].second ].push_back( make_pair( i, arco[i].first ) );
        }
        vector<int> c;
        int foi = 0;
        if(tem_ciclo(c)) {
            contrai(c);
        } else break;
    }

    int resp = 0;
    for(int i = 1; i < n; i++) {
        if(!fora[i]) {
            resp += arco[i].first;
        }
    }
    return resp;
}

int main() {
    int casos;
    scanf("%d", &casos);
    for(int teste = 1; teste <= casos; teste++) {
        printf("Case #%d: ", teste);
        scanf("%d %d", &n, &m);
        for(int i = 0; i < n; i++) grafo[i].clear();
        memset(peso, -1, sizeof(peso));
        for(int i = 0; i < m; i++) {
            int a,b,c;
            scanf("%d %d %d", &a, &b, &c);
            if(a != b) {
                if(peso[a][b] == -1) peso[a][b] = c;
                else peso[a][b] = min(peso[a][b], c);
            }
        }
        int root = 0;
        for(int i = 0; i < n; i++)
            for(int j = 0; j < n; j++)
                if(peso[i][j] != -1 && j != root) {
                    grafo[i].push_back(make_pair(j,peso[i][j]));
                }
        if(!consegue()) {
            printf("Possums!\n");
        } else {
            printf("%d\n", calcula());
        }
    }
}
```

6 Data Structures

kd-tree

kdtree.cpp

```
using namespace std;
#define MAX 101000
typedef long long ll;
struct ponto {
    ll axis[2];
} P[MAX], PNode[MAX], Q[MAX];
int esq[2*MAX], dir[2*MAX], at;
ll minDist;
bool cmpX(ponto p1, ponto p2) {
    if(p1.axis[0] != p2.axis[0]) return p1.axis[0] < p2.axis[0];
    return p1.axis[1] < p2.axis[1];
}
bool cmpY(ponto p1, ponto p2) {
    if(p1.axis[1] != p2.axis[1]) return p1.axis[1] < p2.axis[1];
    return p1.axis[0] < p2.axis[0];
}
int make_tree(int ini, int fim, int axis) {
    //folha
    if(ini+1 == fim) {
        int root = at++;
        PNode[root] = P[ini];
        esq[root] = dir[root] = -1;
        return root;
    } else if(ini+1 > fim) {
        return -1;
    }
    int root = at++;
    int meio = (ini+fim)/2;
    if(!axis){
        nth_element(P+ini, P+meio, P+fim, cmpX);
    } else {
        nth_element(P+ini, P+meio, P+fim, cmpY);
    }
    PNode[root] = P[meio];
    esq[root] = make_tree(ini, meio, 1-axis);
```

```
    dir[root] = make_tree(meio+1, fim, 1-axis);
    return root;
}
// -1 distancia normal ao quadrado, 0 distancia em relacao a x,
//1 dist em relacao a y
ll dist(ponto p1, ponto p2, int axis = -1) {
    if(axis == -1) {
        return (p1.axis[0] - p2.axis[0]) * (p1.axis[0] - p2.axis[0]) +
            (p1.axis[1] - p2.axis[1]) * (p1.axis[1] - p2.axis[1]);
    } else {
        return (p1.axis[axis] - p2.axis[axis])*(p1.axis[axis] - p2.axis[axis]);
    }
}
void query(ponto p, int root, int axis) {
    if(root == -1) return;
    ll d = dist(p,PNode[root]);
    if(d != 0LL) {
        minDist = (minDist != -1LL) ? min(minDist, d) : d;
    }
    if(p.axis[axis] < PNode[root].axis[axis]) {
        query(p, esq[root], 1 - axis);
        if(dist(p, PNode[root], axis) <= minDist) {
            query(p, dir[root], 1-axis);
        }
    } else {
        query(p, dir[root], 1 - axis);
        if(dist(p, PNode[root], axis) <= minDist) {
            query(p, esq[root], 1-axis);
        }
    }
}
int main() {
    int T,test = 0,N;
    for(scanf("%d",&T); test < T; test++) {
        scanf("%d",&N);
        for(int i = 0;i < N;i++) {
            scanf("%lld %lld", &P[i].axis[0], &P[i].axis[1]);
            Q[i] = P[i];
        }
        at = 0;
        int root = make_tree(0,N,0);
        for(int i = 0;i < N;i++) {
            minDist = -1LL;
            query(Q[i], root, 0);
            printf("%lld\n",minDist);
        }
    }
}
```

Binary Indexed Tree (BIT)

bit.cpp

```
#include <algorithm>
#include <cstdlib>
#include <cstdio>
#include <cstring>
using namespace std;

#define NMAX 1000000
int bit[NMAX], bit_inic;

//O( NMAX )
void inicializa_bit() {
    memset(bit, 0, sizeof(bit));
    for(bit_inic = 1; bit_inic < NMAX; bit_inic <<= 1);
    bit_inic >>= 1;
}


//O( log NMAX )
void insere(int x, int val) {
    while(x < NMAX) {
        bit[x] += val;
        x += (x&-x);
    }
}

//O( log NMAX )
int get(int x) {
    int r = 0;
    while(x > 0) {
        r += bit[x];
        x -= (x&-x);
    }
    return r;
}

int sobra;

//Dado um valor freq, encontra o menor X tal que get(X) >= freq.
// -1 se a maior soma acumulada < freq
//O( log NMAX )
int findS(int freq){
    int base = 0;
    int ans=-1;
    sobra = freq;

    for(int bitmask=bit_inic;bitmask!=0;bitmask>>=1) {
        int idx=base+bitmask;
        if(idx>NMAX)
            continue;
        if(freq > bit[idx]) {
            base=idx;
            freq-=bit[idx];
        }
        else if(freq <= bit[idx]) {
            ans = idx;
```

```
        }
    }
    sobra = sobra - freq;
    return ans;
}

//Dado um valor freq, encontra o maior X tal que get(X) <= freq.
// -1 se a menor soma acumulada > freq
//O( log NMAX )
int findG(int freq){
    int ans = findS( freq + 1 );
    if(ans != -1)
        return ans-1;

    //devolve o maior cara que tah na bit.
    //quando cai aqui, significa que freq >= get(NMAX-1).
    //poderia devolver NMAX-1.
    return findS( sobra );
}

void teste() {
    srand( time ( NULL ) );
    int sorted[NMAX];

    int n = 100000;
    for(int i = 0; i < n; i++) {
        int x = (rand()% (NMAX-1) ) + 1;
        insere(x, 1);
        sorted[i] = x;
    }
    sort(sorted, sorted + n);
    pos = 0;
    //OLHA UM HEAP - O(n logn)
    while( get( NMAX - 1 ) ) {
        int menor = findG(0) + 1;
        if(menor != sorted[pos++]) printf("NAO TAH ORDENADO\n");
        insere(menor, -1);
    }
}

int main() {
    inicializa_bit();
    teste();
    return 0;
}
```

Longest Increasing Subsequence (LIS)

lis.cpp

```
const int NMAX = 100000;

struct par {
    int a, b;
    bool operator < (const par &p) const {
        return b > p.b || (b == p.b && a < p.a);
    }
} A[NMAX];
int v[NMAX], k, pai[NMAX], fim, id[NMAX];
int main() {
    int n; scanf("%d", &n);
    set <par> S;
    for (int i = 0; i < n; i++) {
        scanf("%d %d", &A[i].a, &A[i].b);
        if (S.count(A[i])) {
            i--; n--;
        } else {
            S.insert(A[i]);
        }
    }
    sort(A, A+n);
    k = 0; fim = -1;
    memset(pai, -1, sizeof(pai));
    for(int i = 0; i < n; i++) {
        int pos = upper_bound(v, v+k, A[i].a) - v;
        if(pos) { pai[i] = id[ pos-1 ]; }
        id[pos] = i;
        v[pos] = A[i].a;
        if(pos == k) {
            fim = i;
            k++;
        }
    }
    vector <par> out;
    while(fim != -1) {
        out.push_back(A[fim]);
        fim = pai[fim];
    }
    printf("%d\n", (int) out.size());
    for (int i = out.size()-1; i >= 0; i--) {
        printf("%d %d\n", out[i].a, out[i].b);
    }
}
```

Manacher's Algorithm (Palindrome Finding)

manacher.cpp

```
/* Encontrar palindromos - inicializa d1 e d2 com zeros, e eles guadram
 * o numero de palindromos centrados na posicao i (d1[i] e d2[i])*/
/* impar */
vector<int> d1 (n);
int l=0, r=-1;
```

```
for (int i=0; i<n; ++i) {
    int k = (i>r ? 0 : min (d1[l+r-i], r-i)) + 1;
    while (i+k < n && i-k >= 0 && s[i+k] == s[i-k])  ++k;
    d1[i] = --k;
    if (i+k > r)
        l = i-k,  r = i+k;
}
/* par */
vector<int> d2 (n);
l=0, r=-1;
for (int i=0; i<n; ++i) {
    int k = (i>r ? 0 : min (d2[l+r-i+1], r-i+1)) + 1;
    while (i+k-1 < n && i-k >= 0 && s[i+k-1] == s[i-k])  ++k;
    d2[i] = --k;
    if (i+k-1 > r)
        l = i-k,  r = i+k-1;
}
```

Segment Tree (with Lazy Propagation)

lazypropag.cpp

```
#include<cstdio>
#define N 100000
struct Arv {
    int esq,dir, ini,fim, trocas, qtd[3];
};
Arv tree[2*N];
int cabeca, qnos, nos[N], TROCAS[N];
int monta(int ini, int fim) {
    int at = cabeca++;
    tree[at].ini = ini;
    tree[at].fim = fim;
    tree[at].trocas = 0;
    if(ini == fim) {
        tree[at].esq = tree[at].dir = -1;
        tree[at].qtd[1] = tree[at].qtd[2] = 0;
        tree[at].qtd[0] = 1;
        return at;
    }
    int e = tree[at].esq = monta(ini, (ini+fim)/2);
    int d = tree[at].dir = monta((ini+fim)/2 + 1, fim);
    for(int i = 0; i < 3; i++) {
        tree[at].qtd[i] = tree[e].qtd[i] + tree[d].qtd[i];
    }
    return at;
}

int AUX[3];
void soma(int at, int ini, int fim) {
    if(ini <= tree[at].ini && tree[at].fim <= fim) {
        tree[at].trocas++;
        tree[at].trocas %= 3;
        for(int i = 0; i < 3; i++) {
            AUX[i] = tree[at].qtd[i];
        }
        for(int i = 0; i < 3; i++) {
            tree[at].qtd[i] = AUX[(i-1+3)%3];
        }
        return;
    }
    int e = tree[at].esq,d = tree[at].dir;
    if(tree[e].ini <= fim && tree[e].fim >= ini) {
        soma(e,ini,fim);
    }
    if(tree[d].ini <= fim && tree[d].fim >= ini) {
        soma(d,ini,fim);
    }

    int trocas = tree[at].trocas;
    for(int i = 0; i < 3; i++) {
        tree[at].qtd[i] = tree[e].qtd[(i-trocas+3)%3] + tree[d].qtd[(i-trocas+3)%3];
    }
}

void get(int at, int ini, int fim, int trocas) {
    if(ini <= tree[at].ini && tree[at].fim <= fim) {
        TROCAS[qnos] = trocas%3;
        nos[qnos++] = at;
        return;
    }
    trocas += tree[at].trocas;
    int e = tree[at].esq, d = tree[at].dir;
    if(tree[e].ini <= fim && tree[e].fim >= ini) {
        get(e,ini,fim,trocas);
    }
    if(tree[d].ini <= fim && tree[d].fim >= ini) {
        get(d,ini,fim,trocas);
    }
}

int query(int a, int b) {
    qnos = 0;
    get(0,a,b,0);
    int r = 0;
    for(int i = 0; i < qnos; i++) {
        r += tree[nos[i]].qtd[ (-TROCAS[i]+3) % 3];
    }
    return r;
}

int main() {
    int n,q,op,a,b;
    scanf("%d %d", &n, &q);
    cabeca = 0; monta(0,n-1);
    for(int i = 0; i < q; i++) {
```

```
        scanf("%d %d %d", &op, &a, &b);
        if(op == 0) {
            soma(0,a,b);
        } else {
            printf("%d\n", query(a,b));
        }
    }
}
```

Longest Common Ancestor (LCA)

lca.cpp

```
#define N 40011
#define K 16
struct X{
    int v, c;
};
X no(int v, int c) {
    X novo; novo.v = v; novo.c = c; return novo;
}
vector<X> adj[N];
int pai[N][K], distancia[N][K], prof[N], n,m;
void monta(int v, int p, int pro, int d) {
    prof[v] = pro;
    pai[v][0] = p;
    distancia[v][0] = d;
    for(int i = 1; i < K; i++) {
        pai[v][i] = pai[ pai[v][i-1] ] [i-1];
        distancia[v][i] = distancia[v][i-1] + distancia[ pai[v][i-1] ][i-1];
    }
    FOR(i, sz(adj[v])) {
        if(adj[v][i].v == p) continue;
        monta(adj[v][i].v, v, pro+1, adj[v][i].c);
    }
}
//monta pd com pais em distancia 1, 2, 4, 8 ...
void calc() {
    prof[0] = 0;
    FOR(i,K) {
        pai[0][i] = 0;
        distancia[0][i] = 0;
    }
    FOR(i, sz(adj[0]))
        monta(adj[0][i].v, 0, 1, adj[0][i].c);
}
int dist(int a, int b) {
    int pa = a, pb = b, d = 0;
    if(prof[pb] > prof[pa]) swap(pa, pb);
    //iguala niveis
    while(prof[pa] > prof[pb]) {
        int j = 0;
        FOR(i,K) {
            if(prof[ pai[pa][i] ] < prof[pb]) break;
            j = i;
        }
        d += distancia[pa][j];
        pa = pai[pa][j];
    }
    //vai subindo
    while(pa != pb) {
        int j = 0;
        FOR(i, K) {
            if(pai[pa][i] == pai[pb][i]) break;
            j = i;
        }
        d += distancia[pa][j];
        d += distancia[pb][j];
        pa = pai[pa][j];
        pb = pai[pb][j];
    }
    return d;
}
int main() {
    scanf("%d %d", &n, &m);
    FOR(i,n) adj[i].clear();
    int a, b, c; char d;
    FOR(i,m) {
        scanf("%d %d %d %c\n", &a, &b , &c, &d);
        a--; b--;
        adj[a].pb(no(b,c));
        adj[b].pb(no(a,c));
    }
    calc();
    int k; scanf("%d", &k);
    while(k--) {
        scanf("%d %d", &a, &b); a--; b--;
        printf("%d\n", dist(a,b));
    }
}
```