PUC-Rio ACM-ICPC Reference

NP-Drada (Francisco Thiesen, Handel Scholze e Lucas Pavanelli)

January 22, 2017

Contents

1 Combinatória

1.1 Somatórios

$\sum_{k=0}^{n} k = n(n+1)/2$
$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$
$\sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$
$\sum_{k=0}^{n} k^4 = (6n^5 + 15n^4 + 10n^3 - n)/30$

1.2 Binômio de Newton

Total de multisets de tamanho k a partir de n elems: $\binom{n+k-1}{k}$
Total de n-tuplas com elementos não negativos com soma = s: $\binom{s+n-1}{n-1}$
Total de n-tuplas com elementos positivos com soma = s: $\binom{s-1}{n-1}$
Numero de caminhos de lattice restritos de (0,0)-(a,b) : $\binom{a+b}{a}$

1.3 Numeros de Catalão

$C_n = 1/n + 1 * \binom{2n}{n}, C_0 = 1, C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}, C_{n+1} = C_n \frac{4n+2}{n+2}$ C_n representa: numero de sequencias válidas com n pares de parenteses triangulações de um (n+2)-gono

2 Template

```
#include <bits/stdc++.h>
using namespace std;

#define INF 0x3F3F3F3F
#define INFLL 0x3F3F3F3F3F3F3F3FLL
#define pb push_back
#define mp make_pair
#define fi first
#define se second
#define sz(x) int((x).size())
#define all(x) (x).begin(), (x).end()

#define contOnes __builtin_popcountl
#define logLow(X) (X <= 0 ? 0 : 31 - __builtin_clz(X))
#define logUpper(X) (X <= 1 ? 0 : 32 - __builtin_clz(X-1))
#define pow2(X) ((X)*(X))

#define watch(x) cout << (#x) << " is " << (x) << endl
#define mod(x, m) ((((x) % (m)) + (m)) % (m))
#define max3(a, b, c) max(a, max(b, c))
#define min3(a, b, c) min(a, min(b, c))
#define unico(x) x.erase(unique(all(x)), (x).end())

typedef long long ll;
typedef unsigned long long ull;
typedef pair<int,int> ii;
typedef pair<int,ii> iii;
typedef vector<int> vi;
typedef vector<vi> vvi;
typedef vector<ii> vii;
typedef vector<vii> vvii;
typedef vector<iii> viii;
typedef vector<ll> vll;
typedef vector<double> vd;
typedef vector<vd> vvd;

const double inf = 1.0/0.0;
const double pi = 2 * acos(0.0);

// Retorna -1 se a < b, 0 se a = b e 1 se a > b.
int cmp_double(double a, double b = 0, double eps = 1e-9) {
  return a + eps > b ? b + eps > a ? 0 : 1 : -1;
}
```

```cpp
int main() {
  ios_base::sync_with_stdio(false);
  int n = numeric_limits<int>::max();
  return 0;
}
```

3 Ad-Hoc

3.1 Casamento estável $O(n^2)$

```cpp
vi stable_marriage(vvi& men, vvi& women)
{
  int n = sz(men);
  vi ans(n, -1), prop(n, 0), match(n, -1);
  queue<int> q;
  for(int i = 0; i < n; ++i)
    q.push(i);
  while(!q.empty()){
    int m = q.front(), w = men[m][prop[m]++];
    if(match[w] == -1){
      ans[m] = w;
      match[w] = m;
      q.pop();
    } else{
      int m2 = match[w], mfirst = 0;
      for(int i = 0; i < n && women[w][i] != m2; ++i){
        if(women[w][i] == m){
          mfirst = 1;
          break;
        }
      }
      if(mfirst){
        q.pop();
        ans[m] = w;
        match[w] = m;
        ans[m2] = -1;
        q.push(m2);
      }
    }
  }
  return ans;
}
```

3.2 Maioria área em um histograma $O(n)$

```cpp
ll maxAreaHist(vll& v){
  ll maxArea = 0, area;
  v.pb(0);
  int n = sz(v), top;
  stack<int> st;
  for(int i = 0; i < n; ++i){
    while(!st.empty() && v[i] < v[st.top()]){
      top = st.top(); st.pop();
      if(st.empty()) area = v[top]*i;
      else area = v[top] * (i - st.top() - 1);
      maxArea = max(maxArea, area);
    }
    st.push(i);
  }
  v.pop_back();
  return maxArea;
}
```

3.3 Algoritmo de Mo$O(sqrt(n) * n)$

```cpp
/* Responde quantos elementos no intervalo [l, r]
   Aparecem pelo menos duas vezes */

struct Query {
    int l, r, i;
} query[MAXQ];

int answer;
int cnt[MAXN], v[MAXN], ans[MAXQ];

void add(int idx) {
    cnt[v[idx]]++;
    if (cnt[v[idx]] == 2)
        answer += 2;
    else if (cnt[v[idx]] > 2)
        answer++;
}
void remove(int idx) {
    cnt[v[idx]]--;
    if (cnt[v[idx]] == 1)
        answer -= 2;
    else if (cnt[v[idx]] > 1)
        answer--;
}
int main() {
    sort(query, query+q, [] (Query a, Query b) {
        if (a.l/SQRTN != b.l/SQRTN)
            return a.l < b.l;
        return a.r < b.r;
    });

    for (int i = 0, curl = 0, curr = -1; i < q; i++) {
        while (curl > query[i].l)
            add(--curl);
        while (curr < query[i].r)
            add(++curr);
        while (curl < query[i].l)
            remove(curl++);
        while (curr > query[i].r)
            remove(curr--);
        ans[query[i].i] = answer;
    }
}
```

4 Estruturas de Dados

4.1 BIT $< O(logN), O(logN) >$

```
struct BIT {
    int n;
    vi bit;
    BIT(int _n) : n(_n+1) {
        bit.resize(n+1, 0);
    }
    void update(int x, int val) {
        x++;
        for (; x <= n; x += x & -x)
            bit[x] += val;
    }
    int query(int x) {
        x++;
        int ret = 0;
        for (; x > 0; x -= x & -x)
            ret += bit[x];
        return ret;
    }
    int query(int l, int r) {
        return query(r)-query(l-1);
    }
};
```

4.2 BIT-2D

```
int bit[M][M], n;
int sum( int x, int y ){
    int ret = 0;
    while( x > 0 ){
        int yy = y; while( yy > 0 ) ret += bit[x][yy], yy -= yy & -yy;
        x -= (x & -x);
    }
    return ret ;
}
void update(int x , int y , int val){
    int y1;
    while (x <= n){
        y1 = y;
        while (y1 <= n){ bit[x][y1] += val; y1 += (y1 & -y1); }
        x += (x & -x);
    }
}
```

4.3 Segment Tree$< O(n), O(logN), O(logN)) >$

```
#define LC 2*node+1
#define RC 2*node+2
#define REC int node = 0, int L = 0, int R = -1
template <typename T>
struct segtree {
    T op(const T& a, const T& b) {
        // This is THE function you want to change.
        return a+b;
    }
    int N;
    vector<T> tree;
    segtree(int _N) : N(_N) {
        int e = 32 - __builtin_clz(N-1);
        if (N == 1) e = 0;
        N = 1<<e;
        tree.resize(1<<(e+1));
    }
    // Vector to be in the leaves
    void setup(vector<T>& V, REC) {
        if (R == -1) R = N;
        if (L >= sz(V)) return;
        if (R - L == 1) {
            tree[node] = V[L];
            return;
        }
        int m = (L + R)/2;
        setup(V, LC, L, m);
        setup(V, RC, m, R);
        tree[node] = op(tree[LC], tree[RC]);
    }
    // Does "V[i] = nv"
    T update(int i, const T& nv, REC) {
        if (R == -1) R = N;
        if (i < L || i >= R) return tree[node];
        if (R - L == 1) return tree[node] = nv;
        int m = (L + R)/2;
        return tree[node] = op( update(i, nv, LC, L, m),
            update(i, nv, RC, m, R));
    }
    // queries the CLOSED interval [i,j]
    T query(int i, int j) {
        return _query(i, j+1);
    }
    T _query(int i, int j, REC) {
        if (R == -1) R = N;
        if (i <= L && R <= j) return tree[node];
        int m = (L + R)/2;
        if (i >= m) return _query(i, j, RC, m, R);
        else if (j <= m) return _query(i, j, LC, L, m);
        return op(_query(i, j, LC, L, m), _query(i, j, RC, m, R));
    }
};
#undef LC
#undef RC
#undef REC
```

4.4 RMQ-DP $< O(nLogn), O(1) >$

```
#define better(a,b) A[a]<A[b]?(a):(b)
int make_dp(int n) { // N log N
    for(int i = 0; i < n; ++i) H[i][0]=i;
    for(int l=0,k; (k=1<<l) < n; l++) for(int i=0;i+k<n;i++)
        H[i][l+1] = better(H[i][l], H[i+k][l]);
}
int query_dp(int a, int b) {
  int l = __lg(b-a);
  return better(H[a][l], H[b-(1<<l)+1][l]);
}
```

4.5 Segment Tree com Lazy Propagation $< O(n), O(logN), O(logN)) >$

```
#define LC 2*node+1
#define RC 2*node+2
#define REC int node = 0, int L = 0, int R = -1
template <typename T, typename T2>
struct segtree_lazy {
    T op(const T& a, const T& b) {
        // This is THE function you want to change.
        return a + b;
    }
    void recompute(int node, int L, int R, const T2& upd) {
        // Change this also. This is an example for the pirate.
        if (!islazy[node]) lazy[node] = 0;
        islazy[node] = true;
        if (upd == 1) { // 1 means flip
            lazy[node] ^= 1;
            tree[node] = (R - L) - tree[node];
        }
        else if (upd != 0) {  // 2 means set to 0, 3 means set to 1
            lazy[node] = upd;
            tree[node] = (upd - 2)*(R-L);
        }
    }
    int N;
    vector<T> tree;
    vector<T2> lazy;
    vector<bool> islazy;
segtree_lazy(int _N) : N(_N) {
    int e = 32 - __builtin_clz(N-1);
    if (N == 1) e = 0;
    N = 1<<e;
    tree.resize(1<<(e+1));
    lazy.resize(1<<(e+1));
    islazy.resize(1<<(e+1), false);
}
// Vector to be in the leaves
void setup(vector<T>& V, REC) {
    if (R == -1) R = N;
    if (L >= sz(V)) return;
    if (R - L == 1) {
        tree[node] = V[L];
        return;
    }
    int m = (L + R)/2;
    setup(V, LC, L, m);
    setup(V, RC, m, R);
    tree[node] = op(tree[LC], tree[RC]);
}
void push(int node, int L, int R) {
    if (!islazy[node]) return;
    islazy[node] = false;
    int m = (L+R)/2;
    recompute(LC, L, m, lazy[node]);
    recompute(RC, m, R, lazy[node]);
}
void update(int i, int j, const T2& upd) {
    _update(i, j+1, upd);
}
T _update(int i, int j, const T2& upd, REC) {
    if (R == -1) R = N;
    if (j <= L || i >= R) return tree[node];
    if (i <= L && R <= j) {
        recompute(node, L, R, upd);
        return tree[node];
    }
    push(node, L, R);
    int m = (L + R)/2;
    return tree[node] = op( _update(i, j, upd, LC, L, m),
        _update(i, j, upd, RC, m, R));
}
// queries the CLOSED interval [i,j]
T query(int i, int j) {
    return _query(i, j+1);
}
T _query(int i, int j, REC) {
    if (R == -1) R = N;
    if (i <= L && R <= j) return tree[node];
    push(node, L, R);
    int m = (L + R)/2;
    if (i >= m) return _query(i, j, RC, m, R);
    else if (j <= m) return _query(i, j, LC, L, m);
    return op(_query(i, j, LC, L, m), _query(i, j, RC, m, R));
}
};
#undef LC
#undef RC
#undef REC
```

4.6 Union Find

```
struct UnionFind {
    int nsize;
    vi parent, size;

    UnionFind(int n) : nsize(n) {
        size.resize(n, 1);
        parent.resize(n);
        iota(all(parent), 0);
    }

    int unions(int u, int v) {
        if (finds(u) == finds(v))
            return 0;
        size[parent[v]] += size[parent[u]];
        parent[parent[u]] = parent[v];
        nsize--;
        return 1;
    }

    int finds(int u) {
        if (u == parent[u])
            return u;
        return parent[u] = finds(parent[u]);
    }
};
```

5 Geometria

5.1 Template

```
struct Point {
    double x, y;
    Point() {}
    Point(double x, double y) : x(x), y(y) {}
    Point operator+ (const Point &o) const { return Point(x + o.x, y +
        o.y); }
    Point operator- (const Point &o) const { return Point(x - o.x, y -
        o.y); }
    Point operator* (const double &o) const { return Point(x * o, y * o);
        }
    Point operator/ (const double &o) const { return Point(x / o, y / o);
        }
    double operator* (const Point &o) const { return x * o.x + y * o.y; }
    double operator% (const Point &o) const { return x * o.y - o.x * y; }
    bool operator== (const Point &o) const {
        return cmp_double(x, o.x) == 0 && cmp_double(y, o.y) == 0;
    }
    bool operator< (const Point &o) const {
        return x != o.x ? x < o.x : y < o.y;
    }
    void read() {
        scanf("%lf %lf", &x, &y);
    }
    void print() {
        printf("(%lf,%lf)\n", x, y);
    }
};
typedef Point Vector;
double abs(Point p) {
    return sqrt(p * p);
}
Vector norm(Vector v) {
    return v / abs(v);
}
double ccw(Point p, Point q, Point r) {
    return cmp_double((q - p) % (r - p));
}
// angulo pqr
inline double angle(Point p, Point q, Point r) {
    Point u = p - q, v = r - q;
    return atan2(u % v, u * v);
}
// retorna true se q esta no segmento [p,r]
bool between(Point p, Point q, Point r) {
    return ccw(p, q, r) == 0 && cmp_double((p - q) * (r - q)) <= 0;
}
struct Segment {
    Point p, q;
    Segment() {}
    Segment(Point p, Point q) : p(p), q(q) {}
};
bool in_segment(Point p, Segment s) {
    double t;
    Vector v = s.q - s.p;
    if (cmp_double(v.x, 0) != 0)
        t = (p.x - s.p.x) / v.x;
    else
        t = (p.y - s.p.y) / v.y;
    return cmp_double(t, 0) >= 0 && cmp_double(t, 1) <= 0 && s.p + v * t
        == p;
}
struct Line {
    Vector v;
    Point p;
    // implementar init() se quiser eq da reta na forma ax + by = c
    int a, b, c;
    void init() {
        a = -v.y;
        b = v.x;
        c = a * p.x + b * p.y;
        int d = abs(__gcd(a, __gcd(b, c)));
        if (d != 1)
            a /= d, b /= d, c /= d;
        if (a < 0)
            a = -a, b = -b, c = -c;
        else if (a == 0 && b < 0)
            b = -b, c = -c;
    }
    Line() {}
    Line(Point p, Point q) : v(q-p), p(p) {
        init();
    }
    Line(Segment s) : v(s.q-s.p), p(s.p) {
        init();
    }
    Point operator() (double t) const { return p + v * t; }
    Vector normal() {
        return Vector(-v.y, v.x);
    }
};
pair<double, double> line_intersection(Line a, Line b) {
    double den = a.v % b.v;
    if (den == 0)
        return make_pair(inf, inf);
    double t = -(b.v % (b.p - a.p)) / den;
    double s = -(a.v % (b.p - a.p)) / den;
    return make_pair(t, s);
}
Point segment_intersection(Segment a, Segment b) {
    Line la = Line(a), lb = Line(b);
    pair<double, double> pdd = line_intersection(la, lb);
    double t = pdd.first, s = pdd.second;
    if (t == inf) {
        if (in_segment(b.p, a))
            return b.p;
        if (in_segment(b.q, a))
            return b.q;
        if (in_segment(a.p, b))
            return a.p;
        if (in_segment(a.q, b))
            return a.q;
        return Point(inf, inf);
    }
    if (cmp_double(t, 0) < 0 || cmp_double(t, 1) > 0)
        return Point(inf, inf);
    if (cmp_double(s, 0) < 0 || cmp_double(s, 1) > 0)
        return Point(inf, inf);
    return la(t);
}

double distPointToLine(Point p, Line l) {
    Vector n = l.normal();
    return (l.p - p) * n / abs(n);
}

struct Circle {
    Point p;
    double r;
    Circle() {}
    Circle(Point p, double r) : p(p), r(r) {}
};

bool in_circle(const Circle &c, const Point &p) {
    return cmp_double(abs(c.p - p), c.r) <= 0;
}

Point circumcenter(Point p, Point q, Point r) {
    Point a = p - r, b = q - r, c = Point(a*(p+r)/2, b*(q+r)/2);
    return Point(c % Point(a.y, b.y), Point(a.x, b.x) % c)/(a % b);
}

Point incenter(Point p, Point q, Point r) {
    double a = abs(r - q), b = abs(r - p), c = abs(q - p);
    return (p * a + q * b + r * c) / (a + b + c);
}

// Poligono p[]
int n;
Point p[MAX];

// area de p
double poly_area() {
    double s = 0;
    for (int i = 0; i < n; i++)
        s += p[i] % p[(i+1) % n];
    return abs(s/2);
}

// O(n): retorna -1 se q esta na borda, 1 se esta no interior ou 0 no
    exterior.
int in_poly(Point q) {
    double a = 0;
    for (int i = 0; i < n; i++) {
        if (between(p[i], q, p[(i+1) % n])) return -1;
        a += angle(p[i], q, p[(i+1) % n]);
    }
    return cmp_double(a) != 0;
}

// O(log N): retorna 1 se q esta na borda ou no interior e 0 caso
    contrario.

int in_poly(Point q) {
    int l = 0, r = n;
    while (l+1 < r) {
        int m = (l+r)/2;
        if (ccw(p[m], p[0], q) == 0)
            return between(h[0], q, h[m]);
        if (ccw(p[m], p[0], q) < 0) l = m;
        else r = m;
    }
    return cmp_double(angle(p[0], q, p[l]) + angle(p[l], q, p[r]) +
        angle(p[r], q, p[0])) != 0;
}
```

5.2 Convex Hull $O(nlogn)$

```
int n, k;
Point p[MAXN], h[MAXN];

void convex_hull() {
    sort(p, p+n);
    k = 0;
    h[k++] = p[0];
    for (int i = 1; i < n; i++) {
        if (i != n-1 && ccw(p[0], p[n-1], p[i]) >= 0) continue;
        while (k > 1 && ccw(h[k-2], h[k-1], p[i]) <= 0) k--;
        h[k++] = p[i];
    }
    for (int i = n-2, lim = k; i >= 0; i--) {
        if (i != 0 && ccw(p[n-1], p[0], p[i]) >= 0) continue;
        while (k > lim && ccw(h[k-2], h[k-1], p[i]) <= 0) k--;
        h[k++] = p[i];
    }
    k--;
}
```

5.3 Smallest Enclosing Circle

```
int n;
Point p[MAXN];

Circle spanning_circle() {
    random_shuffle(p, p+n);
    Circle c(Point(), -1);
    for (int i = 0; i < n; i++) if (!in_circle(c, p[i])) {
        c = Circle(p[i], 0);
        for (int j = 0; j < i; j++) if (!in_circle(c, p[j])) {
            c = Circle((p[i] + p[j])/2, abs(p[i] - p[j])/2);
            for (int k = 0; k < j; k++) if (!in_circle(c, p[k])) {
                Point o = circumcenter(p[i], p[j], p[k]);
                c = Circle(o, abs(o - p[k]));
            }
        }
    }
    return c;
}
```

5.4 Closest Pair of Points $O(nlogn)$

```
struct Point {
    int x, y, i;
    Point(int x = 0, int y = 0, int i = 0) : x(x), y(y), i(i) {}
    Point operator- (const Point &o) const { return Point(x - o.x, y -
        o.y); }
    int operator* (const Point &o) const { return x * o.x + y * o.y; }
    bool operator< (const Point &o) const {
        return y != o.y ? y < o.y : x < o.x;
    }
    void read() {
        scanf("%d %d", &x, &y);
    }
};

double abs(const Point &p) {
    return sqrt(p * p);
}

int main() {
    int n;
    Point pnts[MAXN];
    set<Point> box;
    set<Point>::iterator it;
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        pnts[i].read();
        pnts[i].i = i;
    }
    sort(pnts, pnts+n, [] (const Point &p, const Point &q) {
        return p.x != q.x ? p.x < q.x : p.y < q.y;
    });
    double best = inf;
    int u, v;
    box.insert(pnts[0]);
    for (int i = 1, j = 0; i < n; i++) {
        while (j < i && pnts[i].x - pnts[j].x > best)
            box.erase(pnts[j++]);
        for (it = box.lower_bound(Point(pnts[i].x-best, pnts[i].y-best));
                it != box.end() && it->y <= pnts[i].y + best; it++) {
            double tmp = abs(pnts[i] - *it);
            if (tmp < best) {
                best = tmp;
                u = pnts[i].i, v = it->i;
            }
        }
        box.insert(pnts[i]);
    }
    if (u > v) swap(u,v);
    printf("%d %d %.6lf\n", u, v, best);
}
```

5.5 Interseção de Polígonos Convexos $O(|P| + |Q|)$

```
// Utilizar implementacoes das funcoes abaixo, mesmo que ja definidas
    anteriormente

typedef vector<Point> Polygon;

inline int ccw(Point p, Point q, Point r) {
    return cmp_double((p - r) % (q - r));
}

bool between(Point p, Point q, Point r) {
    return ccw(p, q, r) == 0 && cmp_double((p - q) * (r - q)) <= 0;
}

// Do segments [p,q] and [r,s] have an intersection?
bool seg_intersect(Point p, Point q, Point r, Point s) {
    Point A = q - p, B = s - r, C = r - p, D = s - q;
    int a = cmp_double(A % C) + 2 * cmp_double(A % D);
    int b = cmp_double(B % C) + 2 * cmp_double(B % D);
    if (a == 3 || a == -3 || b == 3 || b == -3) return false;
    if (a || b || p == r || p == s || q == r || q == s) return true;
    int t = (p < r) + (p < s) + (q < r) + (q < s);
    return t != 0 && t != 4;
}

// Finds intersection between lines (p,q) and (r,s)
// (Warning: Divides by zero if parallel! Will return NaN or INF in this
    case)
Point line_intersect(Point p, Point q, Point r, Point s) {
    Point a = q - p, b = s - r, c = Point(p % q, r % s);
    return Point(Point(a.x, b.x) % c, Point(a.y, b.y) % c) / (a % b);
}

int in_poly(Point p, Polygon& T) {
    double a = 0; int N = T.size();
    for (int i = 0; i < N; i++) {
        if (between(T[i], p, T[(i+1) % N])) return -1;
        a += angle(T[i], p, T[(i+1) % N]);
    }
    return cmp_double(a) != 0;
}

Polygon poly_intersect(Polygon& P, Polygon& Q) {
    int m = Q.size(), n = P.size();
    if (m == 0 || n == 0) return Polygon();
    int a = 0, b = 0, aa = 0, ba = 0, inflag = 0;
    Polygon R;
    while ((aa < n || ba < m) && aa < 2*n && ba < 2*m) {
        Point p1 = P[a], p2 = P[(a+1) % n], q1 = Q[b], q2 = Q[(b+1) % m];
        Point A = p2 - p1, B = q2 - q1;
        int cross = cmp_double(A % B), ha = ccw(p2, q2, p1), hb = ccw(q2,
            p2, q1);
        if (cross == 0 && ccw(p1, q1, p2) == 0 && cmp_double(A * B) < 0) {
            if (between(p1, q1, p2)) R.push_back(q1);
            if (between(p1, q2, p2)) R.push_back(q2);
            if (between(q1, p1, q2)) R.push_back(p1);
            if (between(q1, p2, q2)) R.push_back(p2);
            if (R.size() < 2) return Polygon();
            inflag = 1; break;
        } else if (cross != 0 && seg_intersect(p1, p2, q1, q2)) {
            if (inflag == 0) aa = ba = 0;
            R.push_back(line_intersect(p1, p2, q1, q2));
            inflag = (hb > 0) ? 1 : -1;
        }
        if (cross == 0 && hb < 0 && ha < 0) return R;
        bool t = cross == 0 && hb == 0 && ha == 0;
        if (t ? (inflag == 1) : (cross >= 0) ? (ha <= 0) : (hb > 0)) {
            if (inflag == -1) R.push_back(q2);
            ba++; b++; b %= m;
        } else {
            if (inflag == 1) R.push_back(p2);
            aa++; a++; a %= n;
        }
    }
    if (inflag == 0) {
        if (in_poly(P[0], Q)) return P;
        if (in_poly(Q[0], P)) return Q;
    }
    unico(R);
    if (R.size() > 1 && R.front() == R.back()) R.pop_back();
    return R;
}

int main(int argc, char const *argv[]) {
    int n, m;
    scanf("%d %d", &n, &m);
    Polygon P(n), Q(m);
    for (int i = 0; i < n; i++)
        P[i].read();
    for (int i = 0; i < m; i++)
        Q[i].read();
    Polygon R = poly_intersect(P, Q);
    for (auto &x: R) x.print();
    return 0;
}
```

5.6 Retângulos

```
struct Rect {
    // Ponto superior esquerdo.
    int x1, y1;
    // Ponto inferior direito.
    int x2, y2;
    Rect() {
        x1 = y2 = -INF;
        y1 = x2 = INF;
    }
    Rect(int a, int b, int c, int d) : x1(a), y1(b), x2(c), y2(d) {}
    void read() {
        scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
    }
    void print() {
        printf("%d %d %d %d\n", x1, y1, x2, y2);
    }
    Rect intersection(Rect a) {
        Rect intersect = *this;
        if (a.x1 > intersect.x1) intersect.x1 = a.x1;
        if (a.y1 < intersect.y1) intersect.y1 = a.y1;
        if (a.x2 < intersect.x2) intersect.x2 = a.x2;
        if (a.y2 > intersect.y2) intersect.y2 = a.y2;
        return intersect;
    }
};
```

6 Grafos

6.1 Dijkstra $O(ElogV)$

```
/* Entrada: Grafo g e vertice fonte s
   Saida: Dijktra: vetor d[] com menor distancia de s ate todo v */

vi dijkstra(vvii& g, int s) {
    // comentar linha abaixo se ja tiver o tamanho n global
    int n = sz(g);
    set<ii> h;
    vi d(n, INF), vis(n, 0);
    d[s] = 0;
    vis[s] = 1;
    h.insert(mp(0, s));
    while (!h.empty()) {
        int u = h.begin()->se;
        h.erase(h.begin());
        vis[u] = 1;
        for (auto& x: g[u]) {
            int v = x.fi, w = x.se;
            if (!vis[v] and d[u] + w < d[v]) {
                h.erase(mp(d[v], v));
                d[v] = d[u] + w;
                h.insert(mp(d[v], v));
            }
        }
    }
    return d;
}
```

6.2 Kruskal $O(ElogV)$

```
// Vertices de 0 a n-1
int kruskal(viii &edges, int n) {
    sort(edges.begin(), edges.end());

    UnionFind ds(n);

    int cost = 0;
    for (int i = 0; i < sz(edges); i++)
        cost += edges[i].fi * ds.unions(edges[i].se.fi, edges[i].se.se);

    return cost;
}
```

6.3 Tarjan$O(V + E)$

```
int n, m;
vector<int> g[MAX];
int lbl[MAX], low[MAX], idx, cnt_scc;
stack<int> st;
bool inSt[MAX];

void dfs(int u) {
    lbl[u] = low[u] = idx++;
    st.push(u);
    inSt[u] = 1;
    for (int i = 0; i < g[u].size(); i++) {
        int v = g[u][i];
        if (lbl[v] == -1) {
            dfs(v);
            low[u] = min(low[u], low[v]);
        } else if (inSt[v]) {
            low[u] = min(low[u], lbl[v]);
        }
    }
    if (low[u] == lbl[u]) {
        printf("%d -> ", ++cnt_scc);
        int v;
        do {
            v = st.top();
            st.pop();
            inSt[v] = 0;
            printf("%d; ", v);
        } while (v != u);
        putchar('\n');
    }
}
void tarjan() {
    for (int i = 1; i <= n; i++) {
        lbl[i] = -1;
        inSt[i] = 0;
    }
    idx = cnt_scc = 0;
    for (int i = 1; i <= n; i++)
        if (lbl[i] == -1)
            dfs(i);
}
```

6.4 Articulações $O(V + E)$

```
int n, m;
vector<int> g[MAXN];//lista de adjacencia, notacao horrorosa
int lbl[MAXN], low[MAXN], parent[MAXN], idx;
bool art[MAXN], has_art;

void dfs(int u) {
    int cnt = 0;
    lbl[u] = low[u] = idx++;
    for (int i = 0; i < g[u].size(); i++) {
        int v = g[u][i];
        if (lbl[v] == -1) {
            parent[v] = u;
            dfs(v);
            low[u] = min(low[u], low[v]);
            if (low[v] >= lbl[u])
                cnt++;
        } else if (v != parent[u]) {
            low[u] = min(low[u], lbl[v]);
        }
    }
    if (cnt > 1 || (lbl[u] != 0 && cnt > 0)) {
        art[u] = 1;
        has_art = 1;
    }
}

void articulation() {
    for (int i = 1; i <= n; i++) {
        lbl[i] = -1;
        art[i] = 0;
    }
    for (int i = 1; i <= n; i++) {
        if (lbl[i] == -1) {
            idx = 0;
            parent[i] = i;
            dfs(i);
        }
    }
}
```

6.5 Pontes $O(V+E)$

```
int n, m;
vector<int> g[MAXN]; // lista de adjacencia, notacao horrorosa
int lbl[MAXN], low[MAXN], parent[MAXN], idx;
bool has_bridge;


void dfs(int u) {
    lbl[u] = low[u] = idx++;
    bool parent_found = 0;
    for (int i = 0; i < g[u].size(); i++) {
        int v = g[u][i];
        if (lbl[v] == -1) {
            parent[v] = u;
            dfs(v);
            low[u] = min(low[u], low[v]);
            if (low[v] == lbl[v]) {
                printf("%d -> %d\n", u, v);
                has_bridge = 1;
            }
        } else if (!parent_found && v == parent[u]) {
            parent_found = 1;
        } else {
            low[u] = min(low[u], lbl[v]);
        }
    }
}

void bridge() {
    for (int i = 1; i <= n; i++)
        lbl[i] = -1;
    for (int i = 1; i <= n; i++) {
        if (lbl[i] == -1) {
            idx = 0;
            parent[i] = i;
            dfs(i);
        }
    }
}
```

6.6 Lowest Common Ancestor $< O(VlogV), O(logV) >$

```
int n, m, u, v, w;
int anc[MAXN][LOGMAXN], dad[MAXN], lvl[MAXN], dist[MAXN];
vii g[MAXN];

void dfs(int u) {
    for (int i = 0; i < g[u].size(); i++) {
        int v = g[u][i].st, w = g[u][i].nd;
        if (v != dad[u]) {
            dad[v] = u;
            lvl[v] = lvl[u] + 1;
            dist[v] = dist[u] + w;
            dfs(v);
        }
    }
}


void pre() {
    dad[0] = lvl[0] = dist[0] = 0;
    dfs(0);
    for (int i = 0; i < n; i++)
```

```
        anc[i][0] = dad[i];
    for (int j = 1; 1<<j < n; j++)
        for (int i = 0; i < n; i++)
            anc[i][j] = anc[anc[i][j-1]][j-1];
}

int lca(int u, int v) {
    if (lvl[u] < lvl[v])
        swap(u, v);
    int log;
    for (log = 1; (1<<log) <= lvl[u]; log++);
    log--;
    for (int i = log; i >= 0; i--)
        if (lvl[u] - lvl[v] >= (1<<i))
            u = anc[u][i];
    if (u == v)
        return u;
    for (int i = log; i >= 0; i--)
        if (anc[u][i] != anc[v][i])
            u = anc[u][i], v = anc[v][i];
    return dad[u];
}
```

6.7 Dinic $O(V + E)$

```
struct Edge {
    int v, c, f, next;
    Edge() {}
    Edge(int v, int c, int f, int next) : v(v), c(c), f(f), next(next) {}
};

int n, m, head[MAXN], lvl[MAXN], src, snk, work[MAXN];
Edge e[MAXM];

void init(int _n, int _src, int _snk) {
    n = _n;
    m = 0;
    src = _src;
    snk = _snk;
    memset(head, -1, sizeof(head));
}

void addEdge(int u, int v, int c) {
    e[m] = Edge(v, c, 0, head[u]);
    head[u] = m++;
    e[m] = Edge(u, 0, 0, head[v]);
    head[v] = m++;
}

bool bfs() {
    queue<int> q;
    memset(lvl, -1, n * sizeof(int));
    lvl[src] = 0;
    q.push(src);
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int i = head[u]; i != -1; i = e[i].next) {
            if (e[i].f < e[i].c && lvl[e[i].v] == -1) {
                lvl[e[i].v] = lvl[u] + 1;
                q.push(e[i].v);
                if (e[i].v == snk)
                    return 1;
            }
        }
    }
    return 0;
}

int dfs(int u, int f) {
    if (u == snk)
        return f;
    for (int &i = work[u]; i != -1; i = e[i].next) {
        if (e[i].f < e[i].c && lvl[u] + 1 == lvl[e[i].v]) {
            int minf = dfs(e[i].v, min(f, e[i].c - e[i].f));
            if (minf > 0) {
                e[i].f += minf;
                e[i^1].f -= minf;
                return minf;
            }
        }
    }
    return 0;
}
//Atencao para integer overflow
int dinic() {
    int f, ret = 0;
    while (bfs()) {
        memcpy(work, head, n * sizeof(int));
        while (f = dfs(src, INF))
            ret += f;
    }
    return ret;
}
```

6.8 Hopcroft-Karp $O(EsqrtV)$

```
// Vertices em g1 de 1 a n. Vertices em g2 de n+1 a n+m.
int n, m, e;
vector<int> g1[MAXN];
int pair_g1[MAXN], pair_g2[MAXM], dist[MAXN];

bool bfs() {
    queue<int> q;
    for (int u = 1; u <= n; u++) {
        dist[u] = INF;
        if (pair_g1[u] == 0) {
            dist[u] = 0;
            q.push(u);
        }
    }
    dist[0] = INF;
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int i = 0; i < g1[u].size(); i++) {
            int v = g1[u][i];
            if (dist[pair_g2[v]] == INF) {
                dist[pair_g2[v]] = dist[u] + 1;
                q.push(pair_g2[v]);
            }
        }
    }
    return dist[0] != INF;
}
bool dfs(int u) {
    if (u == 0)
        return 1;
    for (int i = 0; i < g1[u].size(); i++) {
        int v = g1[u][i];
        if (dist[pair_g2[v]] == dist[u] + 1 && dfs(pair_g2[v])) {
            pair_g1[u] = v;
            pair_g2[v] = u;
                return 1;
            }
        }
    dist[u] = INF;
    return 0;
}
int hk() {
    memset(pair_g1, 0, sizeof(pair_g1));
    memset(pair_g2, 0, sizeof(pair_g2));
    int matching = 0;
    while (bfs())
        for (int u = 1; u <= n; u++)
            if (pair_g1[u] == 0 && dfs(u))
                matching++;
    return matching;
}
int main() {
    scanf ("%d %d %d", &n, &m, &e);
    for (int i = 0, u, v; i < e; ++i) {
        scanf("%d %d", &u, &v);
        g1[u].push_back(n+v);
    }
    printf("%d\n", hk());
    return 0;
}
```

6.9 Heavy-Light Decomposition $< O(logV), O(log^2V) >$

```
// Chamar init() antes de tudo. Ler arestas e chamar addEdge() para cada
    uma delas. // Chamar hl_init() apos.

int op(int a, int b) {
    return a + b;
}

struct SegTree {
    vector<int> data, tree;
    int sz;
    SegTree(int tsz) : sz(1) {
        while (sz < tsz) sz *= 2;
        data.resize(sz);
        tree.resize(2*sz);
    }
    inline int left(int u) {
        return u << 1;
    }
    inline int right(int u) {
        return left(u) + 1;
    }
    void init(int u, int l, int r) {
        if (l == r) {
            tree[u] = data[l];
            return;
        }
        int m = (l + r) >> 1;
        init(left(u), l, m);
        init(right(u), m+1, r);
        tree[u] = op(tree[left(u)], tree[right(u)]);
    }
    void init() {
        init(1, 0, sz-1);
    }
    int query(int u, int l, int r, int a, int b) {
        if (a <= l && r <= b) return tree[u];
        int m = (l + r) >> 1, ret = 0;
        if (a <= m) ret = query(left(u), l, m, a, b);
        if (m < b) ret = op(ret, query(right(u), m+1, r, a, b));
        return ret;
    }
    int query(int a, int b) {
        return query(1, 0, sz-1, a, b);
    }
    void update(int u, int l, int r, int pos, int val) {
        if (l == r) {
            tree[u] = val;
            return;
        }
        int m = (l + r) >> 1;
        if (pos <= m) update(left(u), l, m, pos, val);
        else update(right(u), m+1, r, pos, val);
        tree[u] = op(tree[left(u)], tree[right(u)]);
    }
    void update(int pos, int val) {
        update(1, 0, sz-1, pos, val);
    }
};

struct edge {
    int u, v, w, next;
    edge() {}
    edge(int u, int v, int w, int next) : u(u), v(v), w(w), next(next) {}
};

int n, m, q, head[MAX];
edge e[2*MAX];
int dad[MAX], lvl[MAX], chd[MAX], sz[MAX], heavy[MAX];
int nump, psize[MAX], pfirst[MAX], path[MAX], offset[MAX];
vector<int> walk;
vector<SegTree> ptree;

void init() {
    m = 0;
    memset(head, -1, sizeof(head));
    memset(dad, 0, sizeof(dad));
}

void addEdge(int u, int v, int w, bool rev = false) {
    e[m] = edge(u, v, w, head[u]);
    head[u] = m++;
    if (!rev) addEdge(v, u, w, true);
}
void dfs(int u) {
    walk.push_back(u);
    sz[u] = 1;
    for (int i = head[u]; i != -1; i = e[i].next) {
        int v = e[i].v;
        if (!dad[v]) {
            dad[v] = u,lvl[v] = lvl[u] + 1, dfs(v);
            sz[u] += sz[v];
        }
    }
    for (int i = head[u]; i != -1; i = e[i].next) {
        int v = e[i].v;
        if (dad[v] == u && 2*sz[v] >= sz[u]) {
            heavy[v] = 1;
            break;
        }
    }
    heavy[u] = 0;
}

void hl_init() {
    walk.clear();
    dad[1] = 1, lvl[1] = 0, dfs(1);
    nump = 0;
    for (int i = 0; i < n; i++) {
        int u = walk[i];
        if (!heavy[u]) {
            offset[u] = 0;
            path[u] = nump;
            pfirst[nump] = u;
            psize[nump++] = 1;
        }
        else {
            offset[u] = offset[dad[u]] + 1;
            path[u] = path[dad[u]];
            psize[path[u]]++;
        }
    }
    ptree.clear();
    ptree.reserve(nump);
    for (int i = 0; i < nump; i++)
        ptree.push_back(SegTree(psize[i]));
    for (int i = 0; i < m; i += 2) {
        int u = e[i].u, v = e[i].v;
        if (u != dad[v]) swap(u, v);
        ptree[path[v]].data[offset[v]] = e[i].w;
    }
    for (int i = 0; i < nump; i++)
        ptree[i].init();
}

int lca(int u, int v) {
    int fpu = pfirst[path[u]], fpv = pfirst[path[v]];
    while (fpu != fpv) {
        if (lvl[fpu] > lvl[fpv])
            u = dad[fpu], fpu = pfirst[path[u]];
        else
            v = dad[fpv], fpv = pfirst[path[v]];
    }
    return lvl[u] < lvl[v] ? u : v;
}

void update(int idx, int val) {
    int u = e[idx].u, v = e[idx].v;
    if (u != dad[v]) swap(u, v);
    ptree[path[v]].update(offset[v], val);
}

int query(int u, int v, bool up = false) {
    if (!up) {
        int w = lca(u, v);
        return op(query(u, w, true), query(v, w, true));
    }
    int ret = 0;
    int fpu = pfirst[path[u]], fpv = pfirst[path[v]];
    while (fpu != fpv) {
        ret = op(ret, ptree[path[u]].query(0, offset[u]));
        u = dad[fpu], fpu = pfirst[path[u]];
    }
    if (u != v)
        ret = op(ret, ptree[path[u]].query(offset[v]+1, offset[u]));
    return ret;
}
```

6.10 Min-Cost Max-Flow $O(VE + fElogV), f = maxflow$

```
struct Edge {
    int u, v, cap, flow, cost, next;
    Edge() {}
    Edge(int u, int v, int cap, int flow, int cost, int next)
    : u(u), v(v), cap(cap), flow(flow), cost(cost), next(next) {}
};
int n, m, head[MAXN], src, snk;
Edge e[MAXM];
int pi[MAXN], dist[MAXN], path[MAXN], mincap[MAXN], vis[MAXN];

void init(int _n, int _src, int _snk) {
    n = _n;
    m = 0;
    src = _src;
    snk = _snk;
    memset(head, -1, sizeof(head));
}
void addEdge(int u, int v, int cap, int cost) {
    e[m] = Edge(u, v, cap, 0, cost, head[u]);
    head[u] = m++;
    e[m] = Edge(v, u, 0, 0, -cost, head[v]);
    head[v] = m++;
}
int bellman_ford() {
    memset(pi, INF, sizeof(pi));
    pi[src] = 0;
    int flag = 1;
    for (int i = 0; flag && i < n; i++) {
        flag = 0;
        for (int j = 0; j < m; j++) {
            if (e[j].cap == e[j].flow) continue;
            if (pi[e[j].u] + e[j].cost < pi[e[j].v]) {
                pi[e[j].v] = pi[e[j].u] + e[j].cost;
                flag = 1;
            }
        }
    }
    return flag;
}
int dijkstra() {
    priority_queue<pii, vector<pii>, greater<pii> > heap;
    memset(dist, INF, sizeof(dist));
    memset(vis, 0, sizeof(vis));
    dist[src] = 0;
    mincap[src] = INF;
    heap.push(mp(0, src));
    while (!heap.empty()) {
        int u = heap.top().se;
        heap.pop();
        if (vis[u]) continue;
        vis[u] = 1;
        for (int i = head[u]; i != -1; i = e[i].next) {
            int v = e[i].v;
            if (vis[v] || e[i].flow == e[i].cap) continue;
            int w = dist[u] + e[i].cost + pi[u] - pi[v];
            if (w < dist[v]) {
                dist[v] = w;
                path[v] = i;
                mincap[v] = min(mincap[u], e[i].cap - e[i].flow);
                heap.push(mp(dist[v], v));
            }
        }
    }
    // update potencials
    for (int i = 0; i < n; i++)
        pi[i] += dist[i];
    return dist[snk] < INF;
}
```

```
pii mcmf() {
    // set potencials
    if (bellman_ford())
        return mp(-1, -1);
    int cost = 0, flow = 0;
    while (dijkstra()) {
        // augment path and update cost
        int f = mincap[snk];
        for (int v = snk; v != src; ) {
            int idx = path[v];
            e[idx].flow += f;
            e[idx^1].flow -= f;
            cost += e[idx].cost * f;
            v = e[idx].u;
        }
        flow += f;
    }
    return mp(cost, flow);
}
```

7 Matemática

7.1 LCM

```
ll lcm(ll a, ll b) {
    if (a == 0 && b == 0) return 0;
    return a / __gcd(a,b) * b;
}
```

7.2 Algoritmo Estendido de Euclides

```
typedef pair<ll, ll> pll;

pll egcd(ll a, ll b) {
    ll x = 0, lastx = 1, auxx;
    ll y = 1, lasty = 0, auxy;
    while (b) {
        ll q = a / b, r = a % b;
        a = b, b = r;
        auxx = x;
        x = lastx - q*x, lastx = auxx;
        auxy = y;
        y = lasty - q*y, lasty = auxy;
    }
    return mp(lastx, lasty);
}
```

7.3 Resto Chines para gcd(i, j) = 1

```
struct teq {
    int r, n; // x = r (mod n)
};

int qnt;
teq eqs[MAXN];

int chinese_remainder_algorithm() {
    int beta, sum = 0, n = 1;
    for (int i = 0; i < qnt; i++)
        n *= eqs[i].n;
    for (int i = 0; i < qnt; i++) {
        beta = egcd(eqs[i].n, n/eqs[i].n).second;
        while (beta < 0)
            beta += eqs[i].n;
        sum += (eqs[i].r * beta * n/eqs[i].n) % n;
    }
    return sum;
}
```

7.4 Resto Chines Generalizado

```
struct teq {
    ll r, n; // x = r (mod n)
};

ll inv(ll x, ll n) {
    ll ret = egcd(x, n).first;
    return mod(ret, n);
}

int qnt;
teq eqs[MAX];

pair<ll, ll> chines() {
    ll n = eqs[0].n, r = mod(eqs[0].r, n), auxr, d;
    for (int i = 1; i < qnt; i++) {
        d = __gcd(n, eqs[i].n);
        auxr = mod(eqs[i].r - r, eqs[i].n);
        if (auxr % d) return mp(-1, -1);
        r += auxr / d * n * inv(n/d, eqs[i].n/d);
        n *= eqs[i].n/d;
        r = mod(r, n);
    }
    return mp(r, n);
}
```

7.5 Crivo de Eratosthenes $O(n * loglogn)$

```
int np, p[MAXN];
int lp[MAXN];

void sieve(int n) {
    for (int i = 2; i < n; i++)
        lp[i] = i;
    for (int i = 4; i < n; i += 2)
        lp[i] = 2;
    for (int i = 3; i*i < n; i += 2) if (lp[i] == i)
        for (int j = i*i; j < n; j += i) if (lp[j] == j)
            lp[j] = i;
    np = 0;
    p[np++] = 2;
    for (int i = 3; i < n; i += 2)
        if (lp[i] == i)
            p[np++] = i;
}
```

7.6 Fatoração

```
int nf, f[MAXN], e[MAXN];
void factor(int n) {
    nf = 0;
    for (int i = 0; n != 1 && p[i]*p[i] <= n; i++) {
        if (n % p[i] == 0) {
            f[nf] = p[i];
            e[nf] = 1;
            n /= p[i];
            while (n % p[i] == 0) {
                e[nf]++;
                n /= p[i];
            }
            nf++;
        }
    }
    if (n != 1) {
        f[nf] = n;
        e[nf] = 1;
        nf++;
    }
}
```

7.7 Totiente de Euler

```
//Cuidado com integer overflow
int phi(int n) {
    int ret = 1;
    for (int i = 0; n != 1 && p[i]*p[i] <= n; i++) {
        if (n % p[i] == 0) {
            int pk = p[i];
            n /= p[i];
            while (n % p[i] == 0) {
                pk *= p[i];
                n /= p[i];
            }
            ret *= pk - pk/p[i];
        }
    }
    if (n != 1)
        ret *= n-1;
    return ret;
}
```

7.8 Totiente de Euler $\forall x < n - \phi(x)$

```
int phi[MAXN];
void build_phi(int n) {
    for (int i = 0; i < n; i++)
        phi[i] = i;
    for (int i = 2; i < n; i++) if (phi[i] == i)
        for (int j = i; j < n; j += i)
            phi[j] = phi[j] / i * (i-1);
}
```

7.9 Exponenciação Rápida com Módulo

```
ll power(ll b, ll e, ll mod = numeric_limits<ll>::max()) {
    if (e == 0) return 1;
    b %= mod;
    ll a = power(b, e/2, mod);
    a = (a*a) % mod;
    if (e&1) return (a*b) % mod;
    return a;
}

ll power(ll b, ll e, ll mod = numeric_limits<ll>::max()) {
    ll ret = 1;
    b = b % mod;
    while (e > 0) {
        if (e & 1) ret = (ret * b) % mod;
        e >>= 1;
        b = (b * b) % mod;
    }
    return ret;
}
```

7.10 Fast Fourier Transform

```
typedef complex<long double> pt;
pt tmp[1<<20];

void fft(pt *in, int sz, bool inv) {
    if (sz == 1)
        return;
    for (int i = 0, j = 0, h = sz >> 1; i < sz; i += 2, j++) {
        in[j] = in[i];
        tmp[h+j] = in[i+1];
    }
    for (int i = sz >> 1; i < sz; i++)
        in[i] = tmp[i];
    sz >>= 1;
    pt *even = in, *odd = in + sz;
    fft(even, sz, inv);
    fft(odd, sz, inv);
    long double p = (inv ? -1 : 1) * M_PI / sz;
    pt w = pt(cosl(p), sinl(p)), w_i = 1;
    for (int i = 0; i < sz; i++) {
        pt conv = w_i * odd[i];
        odd[i] = even[i] - conv;
        even[i] += conv;
        w_i *= w;
    }
}
```

7.11 Polinômios

```
struct Poly {
    int n;
    double a[MAXN];
    Poly(int n = 0) : n(n) { memset(a, 0, sizeof(a)); }
    Poly(const Poly &o) : n(o.n) { memcpy(a, o.a, sizeof(a)); }
    const double& operator[] (int i) const { return a[i]; }
    double& operator[] (int i) { return a[i]; }
    double operator() (double x) const {
        double ret = 0;
        for (int i = n; i >= 0; i--)
            ret = ret * x + a[i];
        return ret;
    }
    Poly operator+ (const Poly &o) const {
        Poly ret = o;
        for (int i = 0; i <= n; i++)
            ret[i] += a[i];
        ret.n = max(n, o.n);
        return ret;
    }
    Poly operator- (const Poly &o) const {
        Poly ret = o;
        for (int i = 0; i <= n; i++)
            ret[i] -= a[i];
        ret.n = max(n, o.n);
        return ret;
    }
    Poly operator* (const Poly &o) const {
        Poly ret(n + o.n);
        for (int i = 0; i <= n; i++)
            for (int j = 0; j <= o.n; j++)
                ret[i+j] += a[i] * o[j];
        return ret;
    }
};

// Saida: produto p*q
Poly fastMult(const Poly &p, const Poly &q) {
    int sz = 1 << (32 - __builtin_clz(p.n + q.n + 1));
    pt pin[sz], qin[sz];
    for (int i = 0; i < sz; i++) {
        if (i <= p.n)
            pin[i] = p[i];
        else
            pin[i] = 0;
        if (i <= q.n)
            qin[i] = q[i];
        else
            qin[i] = 0;
    }
    fft(pin, sz, 0);
    fft(qin, sz, 0);
    for (int i = 0; i < sz; i++)
        pin[i] *= qin[i];
    fft(pin, sz, 1);
    Poly ret(p.n + q.n);
    for (int i = 0; i <= ret.n; i++)
        ret[i] = pin[i].real() / sz;
    while (ret.n > 0 && cmp(ret[ret.n], 0) == 0)
        ret.n--;
    return ret;
}

// Saida: polinomio derivada de p
Poly diff(const Poly &p) {
    Poly ret(p.n-1);
    for (int i = 1; i <= p.n; i++)
        ret[i-1] = i * p[i];
    return ret;
}

// Saida: quociente e resto da divisao de p por (p.x - x)
pair<Poly, double> ruffini(const Poly &p, double x) {
    if (p.n == 0)
        return make_pair(Poly(), 0);
    Poly ret(p.n-1);
    for (int i = p.n; i > 0; i--)
        ret[i-1] = ret[i] * x + p[i];
    return make_pair(ret, ret[0] * x + p[0]);
}

// Entrada: polinomio p e intervalo [lo,hi] com exatamente uma raiz
// Saida:  pair::second := 0 se nao existe tal raiz, ou 1 se existe
//         pair::first := raiz do polinomio dentro do intervalo [lo,hi]

pair<double, int> findRoot(const Poly &p, double lo, double hi) {
    if (cmp(p(lo), 0) == 0)
        return make_pair(lo, 1);
    if (cmp(p(hi), 0) == 0)
        return make_pair(hi, 1);
    if (cmp(p(lo), 0) == cmp(p(hi), 0))
        return make_pair(0, 0);
    if (cmp(p(lo), p(hi)) > 0)
        swap(lo, hi);
    while (cmp(lo, hi) != 0) {
        double mid = (lo + hi) / 2;
        double val = p(mid);
        if (cmp(val, 0) == 0)
            lo = hi = mid;
        else if (cmp(val, 0) < 0)
            lo = mid;
        else
            hi = mid;
    }
    return make_pair(lo, 1);
}

// Saida: vetor com raizes reais com suas multiplicidades em ordem
    crescente
vector<double> roots(const Poly &p) {
    vector<double> ret;
    if (p.n == 1) {
        ret.push_back(-p[0] / p[1]);
    }
    else {
        vector<double> r = roots(diff(p));
        r.push_back(-MAXX);
        r.push_back(MAXX);
        sort(r.begin(), r.end());
        for (int i = 0, j = 1; j < (int) r.size(); i++, j++) {
            pair<double, int> pr = findRoot(p, r[i], r[j]);
            if (pr.second)
                ret.push_back(pr.first);
        }
    }
    return ret;
}
```

7.12 Eliminação Gaussiana $O(n^3)$

```
// Entrada: Matriz aumentada do sistema Ax=b, ou seja, com o vetor b a
    direita de A
// Saida: O numero de solucoes do sistema (0, 1, INF). Se o retorno for
    1, as
//   solucoes podem ser verificadas em ans.

typedef vector<double> vd;
typedef vector<vd> vvd;

int gauss(vvd a, vd &ans, double EPS = 1e-9) {
    int n = sz(a);
    int m = sz(a[0]) - 1;

    vi where (m, -1);
    for (int col = 0, row = 0; col < m && row < n; col++) {
        int sel = row;
        for (int i = row; i < n; i++)
            if (abs(a[i][col]) > abs(a[sel][col]))
                sel = i;
        if (abs(a[sel][col]) < EPS)
            continue;
        for (int i = col; i <= m; i++)
            swap(a[sel][i], a[row][i]);
        where[col] = row;

        for (int i = 0; i < n; i++)
```

```
            if (i != row) {
                double c = a[i][col] / a[row][col];
                for (int j = col; j <= m; j++)
                    a[i][j] -= a[row][j] * c;
            }
        row++;
    }

    ans.assign(m, 0);
    for (int i = 0; i < m; i++)
        if (where[i] != -1)
            ans[i] = a[where[i]][m] / a[where[i]][i];
    for (int i = 0; i < n; i++) {
        double sum = 0;
        for (int j = 0; j < m; j++)
            sum += ans[j] * a[i][j];
        if (abs(sum - a[i][m]) > EPS)
            return 0;
    }

    for (int i = 0; i < m; i++)
        if (where[i] == -1)
            return INF;
    return 1;
}
```

7.13 Integração por Método de Simpson

```
double simpson(double a, double b, int n = 1e6) {
    double h = (b - a) / n;
    double s = f(a) + f(b);
    for (int i = 1; i < n; i += 2)
        s += 4 * f(a + h*i);
    for (int i = 2; i < n; i += 2)
        s += 2 * f(a + h*i);
    return s*h/3;
}
```

7.14 Exponenciação Rápida de Matrizes

```
vvd multMatrix(vvd& A, vvd& B) {
    int n = sz(A), m = sz(A[0]), p = sz(B[0]);
    vvd ans(n, vd(p, 0.0));
    for (int i = 0; i < n; i++)
        for (int k = 0; k < m; k++)
            for (int j = 0; j < p; j++)
                ans[i][j] += A[i][k]*B[k][j];
    return ans;
}
vvd expMat(vvd& A, int e) {
    if (e == 0) {
        int n = sz(A);
        vvd ans(n, vd(n, 0));
        for (int i = 0; i < n; i++)
            ans[i][i] = 1.0;
        return ans;
    }
    vvd x = expMat(A, e/2);
    x = multMatrix(x,x);
    if (e&1)
        x = multMatrix(x,A);
    return x;
}
```

8 Programação Dinâmica

8.1 Coin Change (menor qntd) $O(nc)$

```
//Entrada: Vetor v[] de moedas e quantidade C
//Saida: menor quantidade de moedas que forma quantia C

int coinchange(vi& v, int C) {
    vi dp(C+1, INF);
    dp[0] = 0;
    for (int i = 1; i <= C; i++)
        for (int j = 0; j < n; j++)
            if (v[j] <= i)
                dp[i] = min(dp[i], dp[i-v[j]] + 1);
    return dp[C];
}
```

8.2 Coin Change (total de formas) $O(nc)$

```
//Entrada: Vetor v[] de moedas e quantidade C
//Saida: quantidade de maneiras de formar quantia C

int coinchange(vi& v, int C) {
    vi dp(C+1, 0);
    dp[0] = 1;
    for (int i = 1; i <= C; i++)
        for (int j = 0; j < n; j++)
            if (v[j] <= i)
                dp[i] += dp[i - v[j]];

    return dp[C];
}
```

8.3 Knapsack Com Repetição $O(nc)$

```
//Entrada: Vetores v[] e p[] com valores e pesos dos itens e capacidade C
     da mochila
//Saida: Valor maximo obtido

int knapsack(vi& v, vi& p, int C) {
    vi dp(C+1, 0);
    for (int i = 0; i < sz(v); i++)
        for (int j = C; j >= 1; j--)
            if (p[i] <= j)
                dp[j] = max(dp[j], v[i] + dp[j-p[i]]);
    return dp[C];
}
```

8.4 Longest Common Subsequence $O(nm)$

```
//Entrada: strings a e b
//Saida: string com a maior subsequencia em comum entre a e b

string lcs(string a, string b) {

    int n = a.length();
    int m = b.length();

    vvi dp(n+1, vi(m+1, 0));

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i-1] == b[j-1]) dp[i][j] = dp[i-1][j-1] + 1;
            else dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
        }
    }

    string answer = "";

    int i = n;
    int j = m;

    while (i and j) {
        if (dp[i][j] == dp[i-1][j]) i--;
        else if (dp[i][j] == dp[i][j-1]) j--;
        else if (dp[i][j] == dp[i-1][j-1] + 1) answer.push_back(a[i-1]),
            i--, j--;
    }

    reverse(answer.begin(), answer.end());
    return answer;
}
```

8.5 Longest Common Substring $O(nm)$

```
string lcs(string a, string b) {

    int n = a.length();
    int m = b.length();

    int maximum = 0;
    int ii = 0;
    int jj = 0;

    vvi dp(n+1, vi(m+1, 0));

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            if (a[i-1] == b[j-1]) {
                dp[i][j] = dp[i-1][j-1] + 1;
                if (dp[i][j] > maximum) {
                    maximum = dp[i][j];
                    ii = i; jj = j;
                }
            } else {
                dp[i][j] = 0;
            }
        }
    }

    string answer = "";


    while (ii and dp[ii][jj]) {
        answer.push_back(a[--ii]);
        jj--;
    }

    reverse(answer.begin(), answer.end());
    return answer;
}
```

8.6 Longest Increasing Sequence $O(nlogn)$

```
//Saida: ans[i] := indice do i-esimo elemento da maior subsequencia
    crescente

vi find_lis(vi& v) {
    int n = sz(v), len = 0;
    vi tam(n), dad(n);
    for (int i = 0; i < n; i++) {
        int lo = 0, hi = len;
        while (lo < hi) {
            int mid = (lo+hi) / 2;
            if (v[tam[mid]] < v[i])
                lo = mid+1;
            else
                hi = mid;
        }
        dad[i] = (lo > 0) ? tam[lo-1] : -1;
        tam[lo] = i;
        if (lo == len) len++;
    }
    vi ans(len);
    // se quiser a sequencia e nao apenas os indices, trocar u por v[u]
    for (int u = tam[len-1], i = len-1; u != -1; u = dad[u], i--)
        ans[i] = u;
    return ans;
}
```

8.7 Weighted Activity Selection $O(nlogn)$

```
//Entrada: vetor v[] com atividades no formato
//Saida: soma maxima dos pesos de atividades nao sobrepostas
//Obs: v[] e 1-indexado

struct Event {
    int b, e, w;
    Event () {}
    Event (int b, int e, int w) : b(b), e(e), w(w) {}
    bool operator< (const Event& o) const {
        return e != o.e ? e < o.e : b < o.b;
    }
};

int was(vector<Event>& v) {
    int n = sz(v)-1;
    sort(v.begin()+1, v.end());
    vi dp(n+1);
    dp[0] = 0;
    for (int i = 1; i <= n; i++) {
        int lo = 0, hi = i-1;
        while (lo < hi) {
            int mid = (lo + hi + 1)/2;
            if (v[mid].e > v[i].b)
                hi = mid - 1;
            else
                lo = mid;
        }
        dp[i] = max(dp[i-1], v[i].w + dp[lo]);
    }
    return dp[n];
}
```

8.8 Caixeiro Viajante $O(n^2 2^n)$

```
// Saida: menor custo para comecar no vertice 0, fazer um tour pelos
    vertices 1...n-1 e retornar ao vertice 0.

int n;
ii p[MAX];
double pd[(1<<MAX)][MAX];

double cost(int i, int j) {
    return sqrt(pow2(p[i].st-p[j].st)+pow2(p[i].nd-p[j].nd));
}

double tsp() {
    for (int i = 0; i < (1<<n); i++)
        for (int j = 0; j < n; j++)
            pd[i][j] = inf;
    pd[0][0] = 0.0;

    for (int i = 1; i < (1<<n); i += 2) {
        for (int j = 0; j < n; j++) {
            if (i & (1<<j)) continue;
            for (int k = 0; k < n; k++)
                if (k != j and (i & (1<<k)))
                    pd[i][j] = min(pd[i][j], pd[i^(1<<k)][k] + cost(k,j));
        }
    }

    double ans = inf;
    for (int k = 0; k < n; k++)
        ans = min(ans, pd[((1<<n)-1)-(1<<k)][k] + cost(k,0));

    return ans;
}
```

9 String

9.1 Suffix Array e Longest Common Prefix $O(nlogn)$

```
// Usar a partir da funcao initSA(string& _str)
// sa : array de sufixos
// pos: inverso do array de sufixos, ie, pos[i] := o indice do sufixo i
    em sa[]
//  suf[i] < suf[j] se e somente se pos[i] < pos[j]

int n;
string str;
int sa[MAX], pos[MAX], cnt[MAX], nxt[MAX], bh[MAX], b2h[MAX], lcp[MAX];

void suffix_array() {
    for (int i = 0; i < n; i++)
        sa[i] = i;
    sort(sa, sa + n, [] (int a, int b) {
        return str[a] < str[b];
    });
    for (int i = 0; i < n; i++) {
        bh[i] = (i == 0 || str[sa[i]] != str[sa[i-1]]);
        b2h[i] = 0;
    }
    for (int h = 1; h < n; h <<= 1) {
        int buckets = 0;
        for (int i = 0, j; i < n; i = j) {
            j = i + 1;
            while (j < n && !bh[j])
                j++;
            nxt[i] = j;
            buckets++;
        }
        if (buckets == n)
            break;
        for (int i = 0; i < n; i = nxt[i]) {
            cnt[i] = 0;
            for (int j = i; j < nxt[i]; j++)
                pos[sa[j]] = i;
        }
        cnt[pos[n-h]]++;
        b2h[pos[n-h]] = 1;
        for (int i = 0; i < n; i = nxt[i]) {
            for (int j = i; j < nxt[i]; j++) {
                int s = sa[j] - h;
                if (s >= 0) {
                    int head = pos[s];
                    pos[s] = head + cnt[head]++;
                    b2h[pos[s]] = 1;
                }
            }
            for (int j = i; j < nxt[i]; j++) {
                int s = sa[j] - h;
                if (s >= 0 && b2h[pos[s]]) {
                    for (int k = pos[s] + 1; !bh[k] && b2h[k]; k++)
                        b2h[k] = 0;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            sa[pos[i]] = i;
            bh[i] |= b2h[i];
        }
    }
    for (int i = 0; i < n; i++)
        pos[sa[i]] = i;
}
void getlcp() {
    lcp[0] = 0;
    for (int i = 0, h = 0; i < n; i++) {
        if (pos[i] > 0) {
            int j = sa[pos[i] - 1];
            while (i + h < n && j + h < n && str[i+h] == str[j+h])
                h++;
            lcp[pos[i]] = h;
            if (h > 0)
                h--;
        }
    }
}
void initSA(string& _str) {
    str = _str;
    n = sz(str);
    suffix_array();
    getlcp();
}
```

9.2 Aho-Corasick

```
struct Node {
    map<char, int> adj;
    int fail;
    ii match;
    int next;
    void init() {
        adj.clear();
        fail = -1;
        match = mp(-1, -1);
        next = -1;
    }
    int getChild(const char &c) {
        map<char, int>::iterator it = adj.find(c);
        if (it != adj.end())
            return it->nd;
        return -1;
    }
};

int qntNodes, qntPatts;
Node trie[MAX];
void init() {
    trie[0].init();
    qntNodes = 1;
    qntPatts = 0;
}
void addWord(const char *word) {
    int node = 0, aux = -1;
    for (int i = 0; word[i]; i++) {
        aux = trie[node].getChild(word[i]);
        if (aux == -1) {
            trie[qntNodes].init();
            aux = qntNodes++;
            trie[node].adj[word[i]] = aux;
        }
        node = aux;
    }
    trie[node].match = mp(qntPatts++, strlen(word));
}
void build() {
    queue<int> q;
    map<char, int>::iterator it;
    trie[0].fail = -1;
    q.push(0);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        for (it = trie[u].adj.begin(); it != trie[u].adj.end(); it++) {
            int v = it->nd;
            char c = it->st;
            q.push(v);
            int f = trie[u].fail;
            while (f >= 0 && trie[f].getChild(c) == -1)
                f = trie[f].fail;
            f = f >= 0 ? trie[f].getChild(c) : 0;
            trie[v].fail = f;
            trie[v].next = trie[f].match.st >= 0 ? f : trie[f].next;
        }
    }
}
void search(const char *text) {
    int node = 0;
    for (int i = 0; text[i]; i++) {
        while (node >= 0 && trie[node].getChild(text[i]) == -1)
            node = trie[node].fail;
        node = node >= 0 ? trie[node].getChild(text[i]) : 0;
        int aux = node;
        while (aux >= 0) {
            if (trie[aux].match.st >= 0) {
                // do something with the match
                printf("patt: %d, pos: %d\n",
                    trie[aux].match.st,
                    i - trie[aux].match.nd + 1);
            }
            aux = trie[aux].next;
        }
    }
}
```

9.3 KMP $O(n+m)$

```
vi kmp(string& txt, string& ptt) {
    vi matches;

    // Calculo do vetor auxiliar lps (longest proper prefix
    // which is also suffix).
    vi lps(ptt.size() + 1, -1);

    for (int i = 1; i <= ptt.size(); i++) {
        int pos = lps[i - 1];
        while (pos != -1 and ptt[pos] != ptt[i - 1])
            pos = lps[pos];
        lps[i] = pos + 1;
    }

    // String search.
    int sp = 0;
    int kp = 0;
    while(sp < txt.size()) {
        while (kp != -1 and (kp == ptt.size() || ptt[kp] != txt[sp]))
            kp = lps[kp];
        kp++; sp++;
        if (kp == ptt.size())
            matches.push_back(sp - ptt.size());
    }
    return matches;
}
```

9.4 Z algorithm $O(n)$

```
//Saida: Z[i] := tamanho da maior string a partir do indice i que eh um
    prefixo de str

vi z_algo(string &str) {
    int n = sz(str);
    vi Z(n, 0);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
        if (i <= r)
            Z[i] = min (r - i + 1, Z[i - l]);
        while (i + Z[i] < n && str[Z[i]] == str[i + Z[i]])
            ++Z[i];
        if (Z[i] > r - i + 1)
            l = i, r = i + Z[i] - 1;
    }
    return Z;
}
```

9.5 String Hashing

```
#define B 33

llu powB[MAX];
void init() {
    powB[0] = 1;
    for (int i = 1; i < MAX; i++)
        powB[i] = B*powB[i-1];
}

void calc_hash(char *str, llu *h) {
    h[0] = 0;
    for (int i = 0; str[i]; i++)
        h[i+1] = B*h[i] + str[i];
}

llu get_hash(llu *h, int l, int r) {
    return h[r] - h[l]*powB[r-l];
}
```

9.6 Manacher $O(n)$

```
char s[MAXN];
int p[2*MAXN]; // length of the palindrome centered at position (i-1)/2;

void manacher() {
    int m = 0;
    char t[2*MAXN];
    for (int i = 0; s[i]; i++) {
        t[m++] = '#';
        t[m++] = s[i];
        p[i] = 0;
    }
    t[m++] = '#';
    int c = 0, r = 0;
    for (int i = 0; i < m; i++) {
        int i_ = 2 * c - i;
        p[i] = r > i ? min(r-i, p[i_]) : i & 1;
        while (0 <= i-p[i]-1 && i+p[i]+1 < m && t[i-p[i]-1] == t[i+p[i]+1])
            p[i] += 2;
        if (i + p[i] > r) {
            c = i;
            r = i + p[i];
        }
    }
}
```