MIT ACM Reference 2008

Maximum matching (C++)

```
// This code performs maximum bipartite matching.
// It has a heuristic that will give excellent performance on complete graphs
// where rows <= columns.
//
//   INPUT: w[i][j] = cost from row node i and column node j or NO_EDGE
//   OUTPUT: mr[i] = assignment for row node i or -1 if unassigned
//           mc[j] = assignment for column node j or -1 if unassigned
//
//   BipartiteMatching returns the number of matches made.
//
// Contributed by Andy Lutomirski.

typedef vector<int> VI;
typedef vector<VI> VVI;

const int NO_EDGE = -(1<<30);  // Or any other value.

bool FindMatch(int i, const VVI &w, VI &mr, VI &mc, VI &seen)
{
  if (seen[i])
    return false;
  seen[i] = true;
  for (int j = 0; j < w[i].size(); j++) {
    if (w[i][j] != NO_EDGE && mc[j] < 0) {
      mr[i] = j;
      mc[j] = i;
      return true;
    }
  }
  for (int j = 0; j < w[i].size(); j++) {
    if (w[i][j] != NO_EDGE && mr[i] != j) {
      if (mc[j] < 0 || FindMatch(mc[j], w, mr, mc, seen)) {
        mr[i] = j;
        mc[j] = i;
        return true;
      }
    }
  }
  return false;
}
```

```
int BipartiteMatching(const VVI &w, VI &mr, VI &mc)
{
  mr = VI (w.size(), -1);
  mc = VI(w[0].size(), -1);
  VI seen(w.size());

  int ct = 0;
  for(int i = 0; i < w.size(); i++)
    {
      fill(seen.begin(), seen.end(), 0);
      if (FindMatch(i, w, mr, mc, seen)) ct++;
    }
  return ct;
}
```

Min cost circulation (C++)

```
// Generic flow using an adjacency matrix.  If you
// want just regular max flow, setting all edge costs to 1 gives
// running time O(|E|^2 |V|).
//
// Running time: O(min(|V|^2 * totflow, |V|^3 * totcost))
//
// INPUT: cap -- a matrix such that cap[i][j] is the capacity of
//                a directed edge from node i to node j
//
//        cost -- a matrix such that cost[i][j] is the (positive)
//                 cost of sending one unit of flow along a
//                 directed edge from node i to node j
//
//        excess -- a vector such that the total flow from i == excess[i]
//
//
// OUTPUT: cost of the resulting flow; the matrix flow will contain
//         the actual flow values (all nonnegative).
//         The vector excess will contain node excesses that could not be
//         eliminated.  Remember to check it.
//
// To use this, create a MinCostCirc object, and call it like this:
//
//   MinCostCirc circ(N);
//   circ.cap = <whatever>; circ.cost = <whatever>;
//   circ.excess[foo] = bar;
//   circ.flow[i][j] = something;  (if you want)
//   int finalcost = circ.solve();
//
// If you want min-cost max-flow, leave excess blank and call min_cost_max_flow.
// Andy says to use caution in min-cost max-flow mode if you have negative
// costs.

typedef vector<int64_t> VI64;
typedef vector<int> VI;
typedef vector<VI64> VVI64;

const int64_t INF = 1LL<<60;

struct MinCostCirc {
  int N;
  VVI64 cap, flow, cost;
  VI dad, found, src, add;
  VI64 pi, dist, excess;

  MinCostCirc(int N) : N(N), cap(N, VI64(N)), flow(cap), cost(cap),
                       dad(N), found(N), src(N), add(N),
                       pi(N), dist(N+1), excess(N) {}

  void search() {
    fill(found.begin(), found.end(), false);
    fill(dist.begin(), dist.end(), INF);

    int here = N;
    for (int i = 0; i < N; i++)
      if (excess[i] > 0) {
        src[i] = i;
        dist[i] = 0;
        here = i;
      }

    while (here != N) {
      int best = N;
      found[here] = 1;
      for (int k = 0; k < N; k++) {
        if (found[k]) continue;
        int64_t x = dist[here] + pi[here] - pi[k];
        if (flow[k][here]) {
          int64_t val = x - cost[k][here];
          assert(val >= dist[here]);
          if (dist[k] > val) {
            dist[k] = val;
            dad[k] = here;
            add[k] = 0;
            src[k] = src[here];
          }
        }
        if (flow[here][k] < cap[here][k]) {
          int64_t val = x + cost[here][k];
          assert(val >= dist[here]);
          if (dist[k] > val) {
            dist[k] = val;
            dad[k] = here;
            add[k] = 1;
            src[k] = src[here];
          }
        }

        if (dist[k] < dist[best]) best = k;
      }
      here = best;
    }

    for (int k = 0; k < N; k++)
      if (found[k])
        pi[k] = min(pi[k] + dist[k], INF);
  }

  int64_t solve() {
    int64_t totcost = 0;
    int source, sink;
    for(int i = 0; i < N; i++)
      for(int j = 0; j < N; j++)
        if (cost[i][j] < 0)
          {
            flow[i][j] += cap[i][j];
            totcost += cost[i][j] * cap[i][j];
            excess[i] -= cap[i][j];
            excess[j] += cap[i][j];
          }
    bool again = true;
    while (again) {
      search();
      int64_t amt = INF;
      fill(found.begin(), found.end(), false);
      again = false;
      for(int sink = 0; sink < N; sink++)
        {
          if (excess[sink] >= 0 || dist[sink] == INF || found[src[sink]]++)
            continue;
          again = true;
          int source = src[sink];

          for (int x = sink; x != source; x = dad[x])
            amt = min(amt, flow[x][dad[x]] ? flow[x][dad[x]] :
                       cap[dad[x]][x] - flow[dad[x]][x]);
          amt = min(amt, min(excess[source], -excess[sink]));
          for (int x = sink; x != source; x = dad[x]) {
            if (add[x]) {
              flow[dad[x]][x] += amt;
              totcost += amt * cost[dad[x]][x];
            } else {
              flow[x][dad[x]] -= amt;
              totcost -= amt * cost[x][dad[x]];
            }
            excess[x] += amt;
            excess[dad[x]] -= amt;
          }
          assert(amt != 0);
          break;  // Comment out at your peril if you need speed.
        }
    }

    return totcost;
  }

  // returns (flow, cost)
  pair<int,int> min_cost_max_flow(int source, int sink) {
    excess[source] = INF;
    excess[sink] = -INF;
    pair<int, int> ret;
    ret.second = solve();
    ret.first = INF - excess[source];
    return ret;
  }
};
```

Non-bipartite matching (C++)

```
// maximum matching (graph not necessarily bipartite)
// whatever code uses this needs to set N in main()
// author claims N^4 running time

#define PB push_back
#define SZ(x) ((int)(x).size())
#define REP(i, n) for(int i=0; i<n; ++i)
#define FOR(i, b, e) for(typeof(e) i=b; i!=e; ++i)

int N; // the number of vertices in the graph

typedef vector<int> VI;
typdef vector< vector<int> > VVI;

VI match;
VI vis;

void couple(int n, int m) { match[n]=m; match[m]=n; }

// returns true if something interesting has been found, thus a
// augmenting path or a blossom (if blossom is non-empty).
// the dfs returns true from the moment the stem of the flower is
// reached and thus the base of the blossom is an unmatched node.
// blossom should be empty when dfs is called and
// contains the nodes of the blossom when a blossom is found.
bool dfs(int n, VVI &conn, VI &blossom) {
  vis[n]=0;
  REP(i, N) if(conn[n][i]) {
    if(vis[i]==-1) {
      vis[i]=1;
      if(match[i]==-1 || dfs(match[i], conn, blossom)) { couple(n,i); return true; }
    }
    if(vis[i]==0 || SZ(blossom)) {  // found flower
      blossom.PB(i); blossom.PB(n);
      if(n==blossom[0]) { match[n]=-1; return true; }
      return false;
    }
  }
  return false;
}

// search for an augmenting path.
// if a blossom is found build a new graph (newconn) where the
// (free) blossom is shrunken to a single node and recurse.
// if a augmenting path is found it has already been augmented
// except if the augmented path ended on the shrunken blossom.
// in this case the matching should be updated along the appropriate
// direction of the blossom.
bool augment(VVI &conn) {
  REP(m, N) if(match[m]==-1) {
    VI blossom;
    vis=VI(N,-1);
    if(!dfs(m, conn, blossom)) continue;
    if(SZ(blossom)==0) return true; // augmenting path found

// blossom is found so build shrunken graph
    int base=blossom[0], S=SZ(blossom);
    VVI newconn=conn;
    FOR(i, 1, S-1) REP(j, N) newconn[base][j]=newconn[j][base]|=conn[blossom[i]][j];
    FOR(i, 1, S-1) REP(j, N) newconn[blossom[i]][j]=newconn[j][blossom[i]]=0;
    newconn[base][base]=0; // is now the new graph
    if(!augment(newconn)) return false;
    int n=match[base];

// if n!=-1 the augmenting path ended on this blossom

    if(n!=-1) REP(i, S) if(conn[blossom[i]][n]) {
      couple(blossom[i], n);
      if(i&1) for(int j=i+1; j<S; j+=2) couple(blossom[j],blossom[j+1]);
      else for(int j=0; j<i; j+=2) couple(blossom[j],blossom[j+1]);
      break;
    }
    return true;
  }
  return false;
}

// conn should have N VI's, each of length N.  conn[i][j] = 1 if (i,j) is an
// edge, and conn[i][j] = 0 otherwise.  returns the number of edges in a max
```

```
// matching.
int edmonds(VVI &conn) {
  int res=0;
  match=VI(N,-1);
  while(augment(conn)) res++;
  return res;
}
```

Convex hull (C++)

```
// Compute the 2D convex hull of a set of points using the monotone chain
// algorithm.  Eliminate redundant points from the hull if REMOVE_REDUNDANT is
// #defined.
//
// Running time: O(n log n)
//
//   INPUT:   a vector of input points, unordered.
//   OUTPUT:  a vector of points in the convex hull, counterclockwise

using namespace std;

#define REMOVE_REDUNDANT

typedef double T;
typedef pair<T,T> PT;
typedef vector<PT> VPT;

const double EPS = 1e-7;

T det (const PT &a, const PT &b){
  return a.first * b.second - a.second * b.first;
}

T area2 (const PT &a, const PT &b, const PT &c){
  return det(a,b) + det(b,c) + det(c,a);
}

#ifdef REMOVE_REDUNDANT

// return true if point b is between points a and c

bool between (const PT &a, const PT &b, const PT &c){
  return (fabs(area2(a,b,c)) < EPS &&
          (a.first - b.first) * (c.first - b.first) <= 0 &&
          (a.second - b.second) * (c.second - b.second) <= 0);
}

#endif

void convex_hull (VPT &pts){
  sort (pts.begin(), pts.end());
  pts.erase (unique (pts.begin(), pts.end()), pts.end());

  VPT up, dn;
  for (int i = 0; i < pts.size(); i++){
    while (up.size() > 1 && area2(up[up.size()-2], up.back(), pts[i]) >= 0)
      up.pop_back();
    while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(), pts[i]) <= 0)
      dn.pop_back();
    up.push_back(pts[i]);
    dn.push_back(pts[i]);
  }

  pts = dn;
  for (int i = (int) up.size() - 2; i >= 1; i--) pts.push_back(up[i]);

#ifdef REMOVE_REDUNDANT

  if (pts.size() <= 2) return;
  dn.clear();
  dn.push_back (pts[0]);
  dn.push_back (pts[1]);
  for (int i = 2; i < pts.size(); i++){
    if (between (dn[dn.size()-2], dn[dn.size()-1], pts[i])) dn.pop_back();
    dn.push_back (pts[i]);
  }
  if (dn.size() >= 3 && between (dn.back(), dn[0], dn[1])){
    dn[0] = dn.back();
    dn.pop_back();
  }
  pts = dn;

#endif

}
```

Area and centroid (C++)

```cpp
// This code computes the area or centroid of a polygon,
// assuming that the coordinates are listed in a clockwise
// or counterclockwise fashion.
//
// Running time: O(n)
//
//   INPUT: list of x[] and y[] coordinates
//   OUTPUTS: (signed) area or centroid
//
// Note that the centroid is often known as the
// "center of gravity" or "center of mass".

typedef vector<double> VD;
typedef pair<double,double> PD;

double ComputeSignedArea (const VD &x, const VD &y){
  double area = 0;
  for (int i = 0; i < x.size(); i++){
    int j = (i+1) % x.size();
    area += x[i]*y[j] - x[j]*y[i];
  }
  return area / 2.0;
}

double ComputeArea (const VD &x, const VD &y){
  return fabs (ComputeSignedArea (x, y));
}

PD ComputeCentroid (const VD &x, const VD &y){
  double cx = 0, cy = 0;
  double scale = 6.0 * ComputeSignedArea (x, y);
  for (int i = 0; i < x.size(); i++){
    int j = (i+1) % x.size();
    cx += (x[i]+x[j])*(x[i]*y[j]-x[j]*y[i]);
    cy += (y[i]+y[j])*(x[i]*y[j]-x[j]*y[i]);
  }
  return make_pair (cx/scale, cy/scale);
}
```

Misc geometry (C++)

```
// C++ routines for computational geometry.

double INF = 1e100;
double EPS = 1e-7;

struct PT {
  double x, y;
  PT (){}
  PT (double x, double y) : x(x), y(y){}
  PT (const PT &p) : x(p.x), y(p.y){}
  PT operator- (const PT &p){ return PT(x-p.x,y-p.y); }
  PT operator+ (const PT &p){ return PT(x+p.x,y+p.y); }
  PT operator* (double c){ return PT(x*c,y*c); }
  PT operator/ (double c){ return PT(x/c,y/c); }
};

double dot (PT p, PT q){ return p.x*q.x+p.y*q.y; }
double dist2 (PT p, PT q){ return dot(p-q,p-q); }
double cross (PT p, PT q){ return p.x*q.y-p.y*q.x; }
ostream &operator<< (ostream &os, const PT &p){
  os << "(" << p.x << "," << p.y << ")";
}

// rotate a point CCW or CW around the origin

PT RotateCCW90 (PT p){ return PT(-p.y,p.x); }
PT RotateCW90 (PT p){ return PT(p.y,-p.x); }
PT RotateCCW (PT p, double t){
  return PT(p.x*cos(t)-p.y*sin(t),
            p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b
// assuming a != b

PT ProjectPointLine (PT a, PT b, PT c){
  return a + (b-a)*dot(c-a,b-a)/dot(b-a,b-a);
}

// project point c onto line segment through a and b
```

```
PT ProjectPointSegment (PT a, PT b, PT c){
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a,b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}

// compute distance between point (x,y,z) and plane ax+by+cz=d

double DistancePointPlane (double x, double y, double z,
                           double a, double b, double c, double d){
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if two lines are parallel or collinear

bool LinesParallel (PT a, PT b, PT c, PT d){
  return fabs(cross(b-a,c-d)) < EPS;
}

bool LinesCollinear (PT a, PT b, PT c, PT d){
  return LinesParallel(a,b,c,d) && fabs(cross(a-c,d-c)) < EPS;
}

// determine if line segment from a to b intersects with
// line segment from c to d

bool SegmentsIntersect (PT a, PT b, PT c, PT d){
  if (cross(d-a,b-a) * cross(c-a,b-a) > 0) return false;
  if (cross(a-c,d-c) * cross(b-c,d-c) > 0) return false;
  return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists

PT ComputeLineIntersection (PT a, PT b, PT c, PT d){
  b=b-a; d=c-d; c=c-a;
  if (dot(b,b) < EPS) return a;
  if (dot(d,d) < EPS) return c;
  return a + b*cross(c,d)/cross(b,d);
}

// compute center of circle given three points

PT ComputeCircleCenter (PT a, PT b, PT c){
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection (b,b+RotateCW90(a-b),
                                  c,c+RotateCW90(a-c));
}

// determine if point is in a possibly non-convex polygon
// (by William Randolph Franklin); returns 1 for strictly
// interior points, 0 for strictly exterior points, and
// 0 or 1 for the remaining points

// note that it is possible to convert this into an *exact*
// test using integer arithmetic by taking care of the
// division appropriately (making sure to deal with signs
// properly) and then by writing exact tests for checking
// point on polygon boundary

bool PointInPolygon (const vector<PT> &p, PT q){
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
         p[j].y <= q.y && q.y < p[i].y) &&
        q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[j].y - p[i].y))
      c = !c;
  }
  return c;
}

// determine if point is on the boundary of a polygon

bool PointOnPolygon (const vector<PT> &p, PT q){
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment (p[i], p[(i+1)%p.size()], q), q) < EPS)
      return true;
  return false;
}
```

```
// compute intersection of line through points a and b with
// circle centered at c with radius r > 0

vector<PT> CircleLineIntersection (PT a, PT b, PT c, double r){
  vector<PT> ret;
  PT d = b-a;
  double D = cross(a-c,b-c);
  double e = r*r*dot(d,d)-D*D;
  if (e < 0) return ret;
  e = sqrt(e);
  ret.push_back (c+PT(D*d.y+(d.y>=0?1:-1)*d.x*e,-D*d.x+fabs(d.y)*e)/dot(d,d));
  if (e > 0)
    ret.push_back (c+PT(D*d.y-(d.y>=0?1:-1)*d.x*e,-D*d.x-fabs(d.y)*e)/dot(d,d));
  return ret;
}

// compute intersection of circle centered at a with radius r
// with circle centered at b with radius R

vector<PT> CircleCircleIntersection (PT a, PT b, double r, double R){
  vector<PT> ret;
  double d = sqrt(dist2(a,b));
  if (d > r+R || d+min(r,R) < max(r,R)) return ret;
  double x = (d*d-R*R+r*r)/(2*d);
  double y = sqrt(r*r-x*x);
  PT v = (b-a)/d;
  ret.push_back (a+v*x + RotateCCW90(v)*y);
  if (y > 0)
    ret.push_back (a+v*x - RotateCCW90(v)*y);
  return ret;
}
```

Voronoi diagrams (C++)

```
#include "Geometry.cc"

#define MAXN 1024
#define INF 1000000

//Voronoi diagrams: O(N^2*LogN)
//Convex hull: O(N*LogN)
```

```
typedef struct {
  int id;
  double x;
  double y;
  double ang;
} chp;

int n;
double x[MAXN], y[MAXN]; // Input points
chp inv[2*MAXN]; // Points after inversion (to be given to Convex Hull)
int vors;
int vor[MAXN]; // Set of points in convex hull;
               //starts at lefmost; last same as first!!
PT ans[MAXN][2];

int chpcmp(const void *aa, const void *bb) {
  double a = ((chp *)aa)->ang;
  double b = ((chp *)bb)->ang;
  if (a<b) return -1;
  else if (a>b) return 1;
  else return 0; // Might be better to include a
                 // tie-breaker on distance, instead of the cheap hack below
}

int orient(chp *a, chp *b, chp *c) {
  double s = a->x*(b->y-c->y) + b->x*(c->y-a->y) + c->x*(a->y-b->y);
  if (s>0) return 1;
  else if (s<0) return -1;
  else if (a->ang==b->ang && a->ang==c->ang) return -1; // Cheap hack
           //for points with same angles
  else return 0;
}

//the pt argument must have the points with precomputed angles (atan2()'s)
//with respect to a point on the inside (e.g. the center of mass)
int convexHull(int n, chp *pt, int *ans) {
  int i, j, st, anses=0;

  qsort(pt, n, sizeof(chp), chpcmp);
  for (i=0; i<n; i++) pt[n+i] = pt[i];
  st = 0;
  for (i=1; i<n; i++) { // Pick leftmost (bottommost)
                        //point to make sure it's on the convex hull
```

```
      if (pt[i].x<pt[st].x || (pt[i].x==pt[st].x && pt[i].y<pt[st].y)) st = i;
    }
    ans[anses++] = st;
    for (i=st+1; i<=st+n; i++) {
      for (j=anses-1; j; j--) {
        if (orient(pt+ans[j-1], pt+ans[j], pt+i)>=0) break;
        // Should change the above to strictly greater,
        // if you don't want points that lie on the side (not on a vertex) of the hull
        // If you really want them, you might also put an epsilon in orient
      }
      ans[j+1] = i;
      anses = j+2;
    }
    for (i=0; i<anses; i++) ans[i] = pt[ans[i]].id;
    return anses;
  }

  int main(void) {
    int i, j, jj;
    double tmp;

    scanf("%d", &n);
    for (i=0; i<n; i++) scanf("%lf %lf", &x[i], &y[i]);
    for (i=0; i<n; i++) {
      x[n] = 2*(-INF)-x[i]; y[n] = y[i];
      x[n+1] = x[i]; y[n+1] = 2*INF-y[i];
      x[n+2] = 2*INF-x[i]; y[n+2] = y[i];
      x[n+3] = x[i]; y[n+3] = 2*(-INF)-y[i];
      for (j=0; j<n+4; j++) if (j!=i) {
        jj = j - (j>i);
        inv[jj].id = j;
        tmp = (x[j]-x[i])*(x[j]-x[i]) + (y[j]-y[i])*(y[j]-y[i]);
        inv[jj].x = (x[j]-x[i])/tmp;
        inv[jj].y = (y[j]-y[i])/tmp;
        inv[jj].ang = atan2(inv[jj].y, inv[jj].x);
      }
      vors = convexHull(n+3, inv, vor);
      // Build bisectors
      for (j=0; j<vors; j++) {
        ans[j][0].x = (x[i]+x[vor[j]])/2;
        ans[j][0].y = (y[i]+y[vor[j]])/2;
        ans[j][1].x = ans[j][0].x - (y[vor[j]]-y[i]);
        ans[j][1].y = ans[j][0].y + (x[vor[j]]-x[i]);
    }
    printf("Around (%lf, %lf)\n", x[i], y[i]);
    // List all intersections of the bisectors
    for (j=1; j<vors; j++) {
      PT vv;
      vv = ComputeLineIntersection(ans[j-1][0], ans[j-1][1],
                                   ans[j][0], ans[j][1]);
      printf("%lf, %lf\n", vv.x, vv.y);
    }
    printf("\n");
  }
  return 0;
}
```

Euclid's algorithm, etc. (C++)

```
// This is a collection of useful code for solving problems that
// involve modular linear equations.  Note that all of the
// algorithms described here work on nonnegative integers.

using namespace std;

typedef vector<int> VI;
typedef pair<int,int> PII;

// return a % b (positive value)

int mod (int a, int b) {
  int ret = a % b;
  if (ret < 0) ret += b;
  return ret;
}

// computes gcd(a,b)

int gcd (int a, int b){
  if (b == 0) return a;
  return gcd (b, a % b);
}

// computes lcm(a,b)
```

```
int lcm (int a, int b){
  return a/gcd(a,b)*b;
}

// returns d = gcd(a,b); finds x,y such that d = ax + by

int extended_euclid (int a, int b, int &x, int &y){
  int xx = y = 0;
  int yy = x = 1;
  while (b){
    int q = a/b;
    int t = b; b = a%b; a = t;
    t = xx; xx = x-q*xx; x = t;
    t = yy; yy = y-q*yy; y = t;
  }
  return a;
}

// finds all solutions to ax = b (mod n)

VI modular_linear_equation_solver (int a, int b, int n){
  int x, y;
  VI solutions;

  int d = extended_euclid (a, n, x, y);
  if (b%d == 0){
    x = mod (x*(b/d), n);
    for (int i = 0; i < d; i++)
      solutions.push_back (mod (x + i*(n/d), n));
  }

  return solutions;
}

// computes b such that ab = 1 (mod n), returns -1 on failure

int mod_inverse (int a, int n){
  int x, y;
  int d = extended_euclid (a, n, x, y);
  if (d > 1) return -1;
  return mod(x,n);
}
```

```
// Chinese remainder theorem (special case): find z such that
// z % x = a, z % y = b.  Here, z is unique modulo M = lcm(x,y).
// Return (z,M).  On failure, M = -1.

PII chinese_remainder_theorem (int x, int a, int y, int b){
  int s, t;
  int d = extended_euclid (x, y, s, t);
  if (a%d != b%d) return make_pair (0,-1);
  return make_pair (mod(s*b*x+t*a*y,x*y)/d, x*y/d);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i.  Note that the solution is
// unique modulo M = lcm_i (x[i]).  Return (z,M).  On
// failure, M = -1.  Note that we do not require the a[i]'s
// to be relatively prime.

PII chinese_remainder_theorem (const VI &x, const VI &a){
  PII ret = make_pair(x[0], a[0]);
  for (int i = 1; i < x.size(); i++){
    ret = chinese_remainder_theorem (ret.first, ret.second, x[i], a[i]);
    if (ret.second == -1) break;
  }
  return ret;
}

// computes x and y such that ax + by = c; on failure, x = y =-1

void linear_diophantine (int a, int b, int c, int &x, int &y){
  int d = gcd(a,b);
  if (c%d){
    x = y = -1;
  } else {
    x = c/d * mod_inverse (a/d, b/d);
    y = (c-a*x)/b;
  }
}
```

Linear systems, matrix inverse (Stanford) (C++)

```
// Gauss-Jordan elimination with partial pivoting.
//
```

```
// Uses:
//   (1) solving systems of linear equations (AX=B)
//   (2) inverting matrices (AX=I)
//   (3) computing determinants of square matrices
//
// Running time: O(|N|^3)
//
// INPUT:    a[][] = an nxn matrix
//           b[][] = an nxm matrix
//
// OUTPUT:   x[][] = an nxm matrix (stored in b[][])
//           returns determinant of a[][]

const double EPSILON = 1e-7;

typedef vector<double> VD;
typedef vector<VD> VVD;

// Gauss-Jordan elimination with partial pivoting

double GaussJordan (VVD &a, VVD &b){
  double det = 1;
  int i,j,k;
  int n = a.size();
  int m = b[0].size();
  for (k=0;k<n;k++){
    j=k;
    for (i=k+1;i<n;i++) if (fabs(a[i][k])>fabs(a[j][k])) j = i;
    if (fabs(a[j][k])<EPSILON){ cerr << "Matrix is singular." << endl; exit(1); }
    for (i=0;i<n;i++) swap (a[j][i],a[k][i]);
    for (i=0;i<m;i++) swap (b[j][i],b[k][i]);
    if (j!=k) det *= -1;

    double s = a[k][k];
    for (j=0;j<n;j++) a[k][j] /= s;
    for (j=0;j<m;j++) b[k][j] /= s;
    det *= s;
    for (i=0;i<n;i++) if (i != k){
      double t = a[i][k];
      for (j=0;j<n;j++) a[i][j] -= t*a[k][j];
      for (j=0;j<m;j++) b[i][j] -= t*b[k][j];
    }
  }
  return det;
}
```

RREF, matrix rank (C++)

```
// Reduced row echelon form via Gauss-Jordan elimination
// with partial pivoting.  This can be used for computing
// the rank of a matrix.
//
// Running time: O(n^3)
//
// INPUT:    a[][] = an nxn matrix
//
// OUTPUT:   rref[][] = an nxm matrix (stored in a[][])
//           returns rank of a[][]

const double EPSILON = 1e-7;

typedef vector<double> VD;
typedef vector<VD> VVD;

// returns rank

int rref (VVD &a){
  int i,j,r,c;
  int n = a.size();
  int m = a[0].size();
  for (r=c=0;c<m;c++){
    j=r;
    for (i=r+1;i<n;i++) if (fabs(a[i][c])>fabs(a[j][c])) j = i;
    if (fabs(a[j][c])<EPSILON) continue;
    for (i=0;i<m;i++) swap (a[j][i],a[r][i]);

    double s = a[r][c];
    for (j=0;j<m;j++) a[r][j] /= s;
    for (i=0;i<n;i++) if (i != r){
      double t = a[i][c];
      for (j=0;j<m;j++) a[i][j] -= t*a[r][j];
    }
    r++;
  }
  return r;
}
```

Simplex (C++)

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <cmath>
#include <limits>

using namespace std;


/*
 * This is a simplex solver. Given m x n matrix A, m-vector b, n-vector c,
 * finds n-vector x such that
 *
 *  A x <= b (component-wise)
 *
 * maximizing
 *
 *  < x , c >
 *
 * where <x,y> is the dot product of x and y.
 *
 */
typedef long double DOUBLE;
typedef vector<DOUBLE> VD;
typedef vector<VD> VVD;
typedef vector<int> VI;

const DOUBLE EPS = 1e-9;

struct LPSolver {
  int m, n;
  VI B, N;
  VVD D;

  LPSolver(const VVD &A, const VD &b, const VD &c) :
    m(b.size()), n(c.size()), N(n+1), B(m), D(m+2, VD(n+2)) {
    for (int i = 0; i < m; i++) for (int j = 0; j < n; j++) D[i][j] = A[i][j];
    for (int i = 0; i < m; i++) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
    for (int j = 0; j < n; j++) { N[j] = j; D[m][j] = -c[j]; }
    N[n] = -1; D[m+1][n] = 1;
  }

  void Pivot(int r, int s) {
    for (int i = 0; i < m+2; i++) if (i != r)
      for (int j = 0; j < n+2; j++) if (j != s)
        D[i][j] -= D[r][j] * D[i][s] / D[r][s];
    for (int j = 0; j < n+2; j++) if (j != s) D[r][j] /= D[r][s];
    for (int i = 0; i < m+2; i++) if (i != r) D[i][s] /= -D[r][s];
    D[r][s] = 1.0 / D[r][s];
    swap(B[r], N[s]);
  }

  bool Simplex(int phase) {
    int x = phase == 1 ? m+1 : m;
    while (true) {
      int s = -1;
      for (int j = 0; j <= n; j++) {
        if (phase == 2 && N[j] == -1) continue;
        if (s == -1 || D[x][j] < D[x][s] || D[x][j] == D[x][s] && N[j] < N[s]) s = j;
      }
      if (D[x][s] >= -EPS) return true;
      int r = -1;
      for (int i = 0; i < m; i++) {
        if (D[i][s] <= 0) continue;
        if (r == -1 || D[i][n+1] / D[i][s] < D[r][n+1] / D[r][s] ||
            D[i][n+1] / D[i][s] == D[r][n+1] / D[r][s] && B[i] < B[r]) r = i;
      }
      if (r == -1) return false;
      Pivot(r, s);
    }
  }

  DOUBLE Solve(VD &x) {
    int r = 0;
    for (int i = 1; i < m; i++) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] <= -EPS) {
      Pivot(r, n);
      if (!Simplex(1) || D[m+1][n+1] < -EPS) return -numeric_limits<DOUBLE>::infinity();
      for (int i = 0; i < m; i++) if (B[i] == -1) {
        int s = -1;
        for (int j = 0; j <= n; j++)
          if (s == -1 || D[i][j] < D[i][s] || D[i][j] == D[i][s] && N[j] < N[s]) s = j;
        Pivot(i, s);
      }
    }
    if (!Simplex(2)) return numeric_limits<DOUBLE>::infinity();
    x = VD(n);
    for (int i = 0; i < m; i++) if (B[i] < n) x[B[i]] = D[i][n+1];
    return D[m][n+1];
  }
};

int main() {

  const int m = 4;
  const int n = 3;
  DOUBLE _A[m][n] = {
    { 6, -1, 0 },
    { -1, -5, 0 },
    { 1, 5, 1 },
    { -1, -5, -1 }
  };
  DOUBLE _b[m] = { 10, -4, 5, -3 };
  DOUBLE _c[n] = { 1, -1, 0 };

  VVD A(m);
  VD b(_b, _b + m);
  VD c(_c, _c + n);
  for (int i = 0; i < m; i++) A[i] = VD(_A[i], _A[i] + n);

  LPSolver solver(A, b, c);
  VD x;
  solver.Print();
  DOUBLE value = solver.Solve(x);

  cerr << "VALUE: "<< value << endl;
  cerr << "SOLUTION:";
  for (size_t i = 0; i < x.size(); i++) cerr << " " << x[i];
  cerr << endl;
  return 0;
}
```

FFT (C++)

```
typedef vector<int> VI;
double PI = acos(0) * 2;

class complex
{
public:
        double a, b;
        complex() {a = 0.0; b = 0.0;}
        complex(double na, double nb) {a = na; b = nb;}
        const complex operator+(const complex &c) const
                {return complex(a + c.a, b + c.b);}
        const complex operator-(const complex &c) const
                {return complex(a - c.a, b - c.b);}
        const complex operator*(const complex &c) const
                {return complex(a*c.a - b*c.b, a*c.b + b*c.a);}
        double magnitude() {return sqrt(a*a+b*b);}
        void print() {printf("(%.3f %.3f)\n", a, b);}
};

class FFT
{
public:
        vector<complex> data;
        vector<complex> roots;
        VI rev;
        int s, n;

        void setSize(int ns)
        {
                s = ns;
                n = (1 << s);
                int i, j;
                rev = VI(n);
                data = vector<complex> (n);
                roots = vector<complex> (n+1);
                for (i = 0; i < n; i++)
                        for (j = 0; j < s; j++)
                                if ((i & (1 << j)) != 0)
                                        rev[i] += (1 << (s-j-1));
                roots[0] = complex(1, 0);
                complex mult = complex(cos(2*PI/n), sin(2*PI/n));
```

```
		for (i = 1; i <= n; i++)
			roots[i] = roots[i-1] * mult;
	}

	void bitReverse(vector<complex> &array)
	{
		vector<complex> temp(n);
		int i;
		for (i = 0; i < n; i++)
			temp[i] = array[rev[i]];
		for (i = 0; i < n; i++)
			array[i] = temp[i];
	}

	void transform(bool inverse = false)
	{
		bitReverse(data);
		int i, j, k;
		for (i = 1; i <= s; i++) {
			int m = (1 << i), md2 = m / 2;
			int start = 0, increment = (1 << (s-i));
			if (inverse) {
				start = n;
				increment *= -1;
			}
			complex t, u;
			for (k = 0; k < n; k += m) {
				int index = start;
				for (j = k; j < md2+k; j++) {
					t = roots[index] * data[j+md2];
					index += increment;
					data[j+md2] = data[j] - t;
					data[j] = data[j] + t;
				}
			}
		}
		if (inverse)
			for (i = 0; i < n; i++) {
				data[i].a /= n;
				data[i].b /= n;
			}
	}
```

```
        static VI convolution(VI &a, VI &b)
        {
                int alen = a.size(), blen = b.size();
                int resn = alen + blen - 1;     // size of the resulting array
                int s = 0, i;
                while ((1 << s) < resn) s++;    // n = 2^s
                int n = 1 << s; // round up the nearest power of two

                FFT pga, pgb;
                pga.setSize(s); // fill and transform first array
                for (i = 0; i < alen; i++) pga.data[i] = complex(a[i], 0);
                for (i = alen; i < n; i++)      pga.data[i] = complex(0, 0);
                pga.transform();

                pgb.setSize(s); // fill and transform second array
                for (i = 0; i < blen; i++)      pgb.data[i] = complex(b[i], 0);
                for (i = blen; i < n; i++)      pgb.data[i] = complex(0, 0);
                pgb.transform();

                for (i = 0; i < n; i++) pga.data[i] = pga.data[i] * pgb.data[i];
                pga.transform(true);    // inverse transform
                VI result = VI (resn);  // round to nearest integer
                for (i = 0; i < resn; i++)      result[i] = (int) (pga.data[i].a + 0.5);

                int actualSize = resn - 1;      // find proper size of array
                while (result[actualSize] == 0)
                        actualSize--;
                if (actualSize < 0) actualSize = 0;
                result.resize(actualSize+1);
                return result;
        }
};

int main()
{
        VI a = VI (10);
        for (int i = 0; i < 10; i++)
                a[i] = (i+1)*(i+1);
        VI b = FFT::convolution(a, a);
        /* 1 8 34 104 259 560 1092 1968 3333
        5368 8052 11120 14259 17104 19234 20168 19361 16200 10000*/
        for (int i = 0; i < b.size(); i++)
                printf("%d ", b[i]);
        return 0;
}
```

Dense Dijkstra's (C++)

```
void Dijkstra (const VVT &w, VT &dist, VI &prev, int start){
  int n = w.size();
  VI found (n);
  prev = VI(n, -1);
  dist = VT(n, 1000000000);
  dist[start] = 0;

  while (start != -1){
    found[start] = true;
    int best = -1;
    for (int k = 0; k < n; k++) if (!found[k]){
      if (dist[k] > dist[start] + w[start][k]){
        dist[k] = dist[start] + w[start][k];
        prev[k] = start;
      }
      if (best == -1 || dist[k] < dist[best]) best = k;
    }
    start = best;
  }
}
```

Topological sort (C++)

```
// This function uses performs a non-recursive topological sort.
//
// Running time: O(|V|^2).  If you use adjacency lists (vector<map<int> >),
//               the running time is reduced to O(|E|).
//
//   INPUT:   w[i][j] = 1 if i should come before j, 0 otherwise
//   OUTPUT:  a permutation of 0,...,n-1 (stored in a vector)
//            which represents an ordering of the nodes which
//            is consistent with w
//
// If no ordering is possible, false is returned.

typedef double TYPE;
typedef vector<TYPE> VT;
typedef vector<VT> VVT;

typedef vector<int> VI;
typedef vector<VI> VVI;

bool TopologicalSort (const VVI &w, VI &order){
  int n = w.size();
  VI parents (n);
  queue<int> q;
  order.clear();

  for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++)
      if (w[j][i]) parents[i]++;
    if (parents[i] == 0) q.push (i);
  }

  while (q.size() > 0){
    int i = q.front();
    q.pop();
    order.push_back (i);
    for (int j = 0; j < n; j++) if (w[i][j]){
      parents[j]--;
      if (parents[j] == 0) q.push (j);
    }
  }

  return (order.size() == n);
}
```

Kruskal's (C++)

```
/*
Uses Kruskal's Algorithm to calculate the weight of the minimum spanning
forest (union of minimum spanning trees of each connected component) of
a possibly disjoint graph, given in the form of a matrix of edge weights
(-1 if no edge exists). Returns the weight of the minimum spanning
forest (also calculates the actual edges - stored in T). Note: uses a
disjoint-set data structure with amortized (effectively) constant time per
union/find. Runs in O(E*log(E)) time.
*/
```

```
typedef int TYPE;

struct edge
{
    int u, v;
    TYPE d;
};

struct edgeCmp
{
    int operator()(const edge& a, const edge& b) { return a.d > b.d; }
};

int find(vector <int>& C, int x) { return (C[x] == x) ? x : C[x] = find(C, C[x]); }

TYPE Kruskal(vector <vector <TYPE> >& w)
{
     int n = w.size();
     TYPE weight = 0;

     vector <int> C(n), R(n);
     for(int i=0; i<n; i++) { C[i] = i; R[i] = 0; }

     vector <edge> T;
     priority_queue <edge, vector <edge>, edgeCmp> E;

     for(int i=0; i<n; i++)
         for(int j=i+1; j<n; j++)
            if(w[i][j] >= 0)
            {
                edge e;
                e.u = i; e.v = j; e.d = w[i][j];
                E.push(e);
            }

     while(T.size() < n-1 && !E.empty())
     {
         edge cur = E.top(); E.pop();

         int uc = find(C, cur.u), vc = find(C, cur.v);
         if(uc != vc)
         {
             T.push_back(cur); weight += cur.d;
            if(R[uc] > R[vc]) C[vc] = uc;
            else if(R[vc] > R[uc]) C[uc] = vc;
            else { C[vc] = uc; R[uc]++; }
        }
    }

    return weight;
}
```

Longest Increasing Subsequence (C++)

```
// Given a list of numbers of length n, this routine extracts a
// longest increasing subsequence.
//
// Running time: O(n log n)
//
//   INPUT: a vector of integers
//   OUTPUT: a vector containing the longest increasing subsequence

typedef vector<int> VI;
typedef pair<int,int> PII;
typedef vector<PII> VPII;

#define STRICTLY_INCREASNG

VI LongestIncreasingSubsequence(VI v) {
  VPII best;
  VI dad(v.size(), -1);

  for (int i = 0; i < v.size(); i++) {
#ifdef STRICTLY_INCREASNG
    PII item = make_pair(v[i], 0);
    VPII::iterator iter = lower_bound(best.begin(), best.end(), item);
    item.second = i;
#else
    PII item = make_pair(v[i], i);
    VPII::iterator iter = upper_bound(best.begin(), best.end(), item);
#endif
    if (iter == best.end()) {
      dad[i] = (best.size() == 0 ? -1 : best.back().second);
      best.push_back(item);
    } else {
      dad[i] = dad[iter->second];
      *iter = item;
    }
  }

  VI ret;
  for (int i = best.back().second; i >= 0; i = dad[i])
    ret.push_back(v[i]);
  reverse(ret.begin(), ret.end());
  return ret;
}
```

Dates (C++)

```
// Routines for performing computations on dates.  In these routines,
// months are exprsesed as integers from 1 to 12, days are expressed
// as integers from 1 to 31, and years are expressed as 4-digit
// integers.

string dayOfWeek[] = {"Mo", "Tu", "We", "Th", "Fr", "Sa", "Su"};

// converts Gregorian date to integer (Julian day number)

int DateToInt (int m, int d, int y){
  return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}

// converts integer (Julian day number) to Gregorian date: month/day/year

void IntToDate (int jd, int &m, int &d, int &y){
  int x, n, i, j;

  x = jd + 68569;
  n = 4 * x / 146097;
  x -= (146097 * n + 3) / 4;
  i = (4000 * (x + 1)) / 1461001;
  x -= 1461 * i / 4 - 31;
  j = 80 * x / 2447;
  d = x - 2447 * j / 80;
  x = j / 11;
  m = j + 2 - 12 * x;
  y = 100 * (n - 49) + i + x;
}

// converts integer (Julian day number) to day of week

string IntToDay (int jd){
  return dayOfWeek[jd % 7];
}
```

Knuth-Morris-Pratt (C++)

```
/*
Searches for the string w in the string s (of length k). Returns the
0-based index of the first match (k if no match is found). Algorithm
runs in O(k) time.
*/

void buildTable(string& w, vector <int>& t)
{
  t = vector <int>(w.length());
  int i = 2, j = 0;
  t[0] = -1; t[1] = 0;

  while(i < w.length()) {
    if(w[i-1] == w[j]) { t[i] = j+1; i++; j++; }
    else if(j > 0) j = t[j];
    else { t[i] = 0; i++; }
  }
}

int KMP(string& s, string& w)
{
  int m = 0, i = 0;
  vector <int> t;
  buildTable(w, t);

  while(m+i < s.length()) {
    if(w[i] == s[m+i]) {
      i++;
      if(i == w.length()) return m;
    } else {
      m += i-t[i];
      if(i > 0) i = t[i];
    }
  }

  return s.length();
}
```

Hashed strstr (C++)

```cpp
const char *fast_strstr(const char *haystack, const char *needle)
{
  unsigned target = 0, power = 1, hash = 0;
  size_t nlen = strlen(needle), hlen = strlen(haystack);
  if (hlen < nlen || !*needle)
    return 0;
  for(int i = 0; i < nlen; i++) {
    target = target * 257 + needle[i];
    hash = hash * 257 + haystack[i];
    power = power * 257;
  }
  for(int i = nlen; i <= hlen; i++) {
    if (hash == target && !memcmp(haystack + i - nlen, needle, nlen))
      return haystack + i - nlen;
    hash = hash * 257 + haystack[i] - power * haystack[i-nlen];
  }
  return 0;
}
```

Java formatting (Java)

```java
// examples for printing floating point numbers

import java.util.*;
import java.io.*;
import java.text.DecimalFormat;

public class DecFormat {
    public static void main(String[] args) {
        DecimalFormat fmt;

        // round to at most 2 digits, leave of digits if not needed
        fmt = new DecimalFormat("#.##");
        System.out.println(fmt.format(12345.6789)); // produces 12345.68
        System.out.println(fmt.format(12345.0)); // produces 12345
        System.out.println(fmt.format(0.0)); // produces 0
        System.out.println(fmt.format(0.01)); // produces .1

        // round to precisely 2 digits
        fmt = new DecimalFormat("#.00");
        System.out.println(fmt.format(12345.6789)); // produces 12345.68
        System.out.println(fmt.format(12345.0)); // produces 12345.00
        System.out.println(fmt.format(0.0)); // produces .00

        // round to precisely 2 digits, force leading zero
        fmt = new DecimalFormat("0.00");
        System.out.println(fmt.format(12345.6789)); // produces 12345.68
        System.out.println(fmt.format(12345.0)); // produces 12345.00
        System.out.println(fmt.format(0.0)); // produces 0.00

        // round to precisely 2 digits, force leading zeros
        fmt = new DecimalFormat("000000000.00");
        System.out.println(fmt.format(12345.6789)); // produces 000012345.68
        System.out.println(fmt.format(12345.0)); // produces 000012345.00
        System.out.println(fmt.format(0.0)); // produces 000000000.00

        // force leading '+'
        fmt = new DecimalFormat("+0;-0");
        System.out.println(fmt.format(12345.6789)); // produces +12346
        System.out.println(fmt.format(-12345.6789)); // produces -12346
        System.out.println(fmt.format(0)); // produces +0

        // force leading positive/negative, pad to 2
        fmt = new DecimalFormat("positive 00;negative 0");
        System.out.println(fmt.format(1)); // produces "positive 01"
        System.out.println(fmt.format(-1)); // produces "negative 01"

        // qoute special chars (#)
        fmt = new DecimalFormat("text with '#' followed by #");
        System.out.println(fmt.format(12.34)); // produces "text with # followed by 12"

        // always show "."
        fmt = new DecimalFormat("#.#");
        fmt.setDecimalSeparatorAlwaysShown(true);
        System.out.println(fmt.format(12.34)); // produces "12.3"
        System.out.println(fmt.format(12)); // produces "12."
        System.out.println(fmt.format(0.34)); // produces "0.3"

        // different grouping distances:
        fmt = new DecimalFormat("#,####.###");
        System.out.println(fmt.format(123456789.123)); // produces "1,2345,6789.123"

        // scientific:
        fmt = new DecimalFormat("0.000E00");
        System.out.println(fmt.format(123456789.123)); // produces "1.235E08"
        System.out.println(fmt.format(-0.000234)); // produces "-2.34E-04"

        // using variable number of digits:
        fmt = new DecimalFormat("0");
        System.out.println(fmt.format(123.123)); // produces "123"
        fmt.setMinimumFractionDigits(8);
        System.out.println(fmt.format(123.123)); // produces "123.12300000"
        fmt.setMaximumFractionDigits(0);
        System.out.println(fmt.format(123.123)); // produces "123"

        // note: to pad with spaces, you need to do it yourself:
        // String out = fmt.format(...)
        // while (out.length() < targlength) out = " "+out;
    }
}
```

Complicated regex example (Java)

```
// Code which demonstrates the use of Java's regular expression libraries.
// This is a solution for
//
//   Loglan: a logical language
//   http://acm.uva.es/p/v1/134.html


import java.util.*;
import java.util.regex.*;

public class LogLan {
    public static void main (String args[]){

        String regex = BuildRegex();
        Pattern pattern = Pattern.compile (regex);

        Scanner s = new Scanner(System.in);
        while (true) {

            // In this problem, each sentence consists of multiple lines, where the last
            // line is terminated by a period.  The code below reads lines until
            // encountering a line whose final character is a '.'.  Note the use of
            //
            //    s.length() to get length of string
            //    s.charAt() to extract characters from a Java string
            //    s.trim() to remove whitespace from the beginning and end of Java string
            //
            // Other useful String manipulation methods include
            //
            //    s.compareTo(t) < 0 if s < t, lexicographically
            //    s.indexOf("apple") returns index of first occurrence of "apple" in s
            //    s.lastIndexOf("apple") returns index of last occurrence of "apple" in s
            //    s.replace(c,d) replaces occurrences of character c with d
            //    s.startsWith("apple) returns (s.indexOf("apple") == 0)
            //    s.toLowerCase() / s.toUpperCase() returns a new lower/uppercased string
            //
            //    Integer.parseInt(s) converts s to an integer (32-bit)
            //    Long.parseLong(s) converts s to a long (64-bit)
            //    Double.parseDouble(s) converts s to a double

            String sentence = "";
            while (true){
                sentence = (sentence + " " + s.nextLine()).trim();
                if (sentence.equals("#")) return;
                if (sentence.charAt(sentence.length()-1) == '.') break;
            }

            // now, we remove the period, and match the regular expression

            String removed_period = sentence.substring(0, sentence.length()-1).trim();
            if (pattern.matcher (removed_period).find()){
                System.out.println ("Good");
            } else {
                System.out.println ("Bad!");
            }
        }
    }
}
```

Java geometry (Java)

```
// In this example, we read an input file containing three lines, each
// containing an even number of doubles, separated by commas.  The first two
// lines represent the coordinates of two polygons, given in counterclockwise
// (or clockwise) order, which we will call "A" and "B".  The last line
// contains a list of points, p[1], p[2], ...
//
// Our goal is to determine:
//   (1) whether B - A is a single closed shape (as opposed to multiple shapes)
//   (2) the area of B - A
//   (3) whether each p[i] is in the interior of B - A
//
// INPUT:
//   0 0 10 0 0 10
//   0 0 10 10 10 0
//   8 6
//   5 1
//
// OUTPUT:
//   The area is singular.
//   The area is 25.0
//   Point belongs to the area.
//   Point does not belong to the area.

import java.util.*;
import java.awt.*;
import java.awt.geom.*;
import java.math.*;
import java.io.*;

public class JavaGeometry {

    // make a list of doubles from a string

    static ArrayList<Double> readPoints (String s){
        StringTokenizer st = new StringTokenizer (s);
        ArrayList<Double> ret = new ArrayList<Double>();
        while (st.hasMoreTokens())
            ret.add (Double.parseDouble (st.nextToken()));
        return ret;
    }

    // make an Area object from the coordinates of a polygon

    static Area makeArea (ArrayList<Double> points){

        // note that the GeneralPath object does not allow construct
        // of polygons based on doubles -- we must use floats.

        GeneralPath gp = new GeneralPath();
        gp.moveTo ((float) points.get(0).doubleValue(),
                   (float) points.get(1).doubleValue());
        for (int i = 2; i < points.size(); i += 2)
            gp.lineTo ((float) points.get(i).doubleValue(),
                       (float) points.get(i+1).doubleValue());
        gp.closePath();
        return new Area (gp);
    }

    // compute area of polygon

    static double computePolygonArea (ArrayList<Point2D.Double> points){

        // convert to array, for convenience

        Point2D.Double[] pts = points.toArray (new Point2D.Double[0]);

        double area = 0;
        for (int i = 0; i < pts.length; i++){
            int j = (i+1) % pts.length;
            area += pts[i].x * pts[j].y - pts[j].x - pts[i].y;
        }
        return Math.abs(area)/2;
    }

    // compute the area of an Area object containing several disjoint polygons

    static double computeArea (Area area){
        double totArea = 0;

        PathIterator iter = area.getPathIterator (null);
        ArrayList<Point2D.Double> points = new ArrayList<Point2D.Double>();

        while (!iter.isDone()){
            double[] buffer = new double[6];
            switch (iter.currentSegment (buffer)){
            case PathIterator.SEG_MOVETO:
            case PathIterator.SEG_LINETO:
                points.add (new Point2D.Double (buffer[0], buffer[1]));
                break;
            case PathIterator.SEG_CLOSE:
                totArea += computePolygonArea (points);
                points.clear();
                break;
            }
            iter.next();
        }
        return totArea;
    }

    // notice that the main() throws an Exception -- necessary to
    // avoid wrapping the Scanner object for file reading in a
    // try { ... } catch block.

    public static void main (String args[]) throws Exception {

        Scanner scanner = new Scanner (new File ("input.txt"));
        // also,
        //   Scanner scanner = new Scanner (System.in);

        ArrayList<Double> pointsA = readPoints (scanner.nextLine());
        ArrayList<Double> pointsB = readPoints (scanner.nextLine());
        Area areaA = makeArea (pointsA);
        Area areaB = makeArea (pointsB);
        areaB.subtract (areaA);
        // also,
        //   areaB.exclusiveOr (areaA);
        //   areaB.add (areaA);
        //   areaB.intersect (areaA);

        // (1) determine whether B - A is a single closed shape (as
        //     opposed to multiple shapes)

        boolean isSingle = areaB.isSingular();
        // also,
        //   areaB.isEmpty();

        if (isSingle)
            System.out.println ("The area is singular.");
        else
            System.out.println ("The area is not singular.");

        // (2) compute the area of B - A

        System.out.println ("The area is " + computeArea (areaB) + ".");

        // (3) determine whether each p[i] is in the interior of B - A

        while (scanner.hasNextDouble()){
            double x = scanner.nextDouble();
            assert(scanner.hasNextDouble());
            double y = scanner.nextDouble();

            if (areaB.contains(x,y)){
                System.out.println ("Point belongs to the area.");
            } else {
                System.out.println ("Point does not belong to the area.");
            }
        }

        // Finally, some useful things we didn't use in this example:
        //
        //   Ellipse2D.Double ellipse = new Ellipse2D.Double (double x, double y,
        //                                                    double w, double h);
        //
        //     creates an ellipse inscribed in box with bottom-left corner (x,y)
        //     and upper-right corner (x+y,w+h)
        //
        //   Rectangle2D.Double rect = new Rectangle2D.Double (double x, double y,
        //                                                     double w, double h);
        //
        //     creates a box with bottom-left corner (x,y) and upper-right
        //     corner (x+y,w+h)
        //
        // Each of these can be embedded in an Area object (e.g., new Area (rect)).

    }
}
```

3D geom (Java)

```
public class Geom3D {
  // distance from point (x, y, z) to plane aX + bY + cZ + d = 0
  public static double ptPlaneDist(double x, double y, double z,
      double a, double b, double c, double d) {
    return Math.abs(a*x + b*y + c*z + d) / Math.sqrt(a*a + b*b + c*c);
  }

  // distance between parallel planes aX + bY + cZ + d1 = 0 and
  // aX + bY + cZ + d2 = 0
  public static double planePlaneDist(double a, double b, double c,
      double d1, double d2) {
    return Math.abs(d1 - d2) / Math.sqrt(a*a + b*b + c*c);
  }

  // distance from point (px, py, pz) to line (x1, y1, z1)-(x2, y2, z2)
  // (or ray, or segment; in the case of the ray, the endpoint is the
  // first point)
  public static final int LINE = 0;
  public static final int SEGMENT = 1;
  public static final int RAY = 2;
  public static double ptLineDistSq(double x1, double y1, double z1,
      double x2, double y2, double z2, double px, double py, double pz,
      int type) {
    double pd2 = (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2) + (z1-z2)*(z1-z2);

    double x, y, z;
    if (pd2 == 0) {
      x = x1;
      y = y1;
      z = z1;
    } else {
      double u = ((px-x1)*(x2-x1) + (py-y1)*(y2-y1) + (pz-z1)*(z2-z1)) / pd2;
      x = x1 + u * (x2 - x1);
      y = y1 + u * (y2 - y1);
      z = z1 + u * (z2 - z1);
      if (type != LINE && u < 0) {
        x = x1;
        y = y1;
        z = z1;
      }
      if (type == SEGMENT && u > 1.0) {
        x = x2;
        y = y2;
        z = z2;
      }
    }

    return (x-px)*(x-px) + (y-py)*(y-py) + (z-py)*(z-py);
  }

  public static double ptLineDist(double x1, double y1, double z1,
      double x2, double y2, double z2, double px, double py, double pz,
      int type) {
    return Math.sqrt(ptLineDistSq(x1, y1, z1, x2, y2, z2, px, py, pz, type));
  }
}
```

Modifications (C++)

```
### .emacs
(setq column-number-mode t)
(setq inhibit-startup-message t)
(transient-mark-mode t)
(global-font-lock-mode t)
(show-paren-mode t)
(global-set-key "\C-cg" 'goto-line)
(defun previous-6-lines nil
  (interactive)
  (previous-line 6))
(defun next-6-lines nil
  (interactive)
  (next-line 6))
(global-set-key [(control up)] 'previous-6-lines)
(global-set-key [(control down)] 'next-6-lines)

### .bashrc
export CXXFLAGS="-Wall -g"
```

```
PS1='\u@\h:\W\$ '
export PYTHONSTARTUP="$HOME/.pythonrc"
alias "l=ls -lh --color=auto"

### .pythonrc
import readline
import rlcompleter
readline.parse_and_bind("tab: complete")

### .vimrc
runtime! debian.vim
:syntax on
:set sw=2 sts=2
:set mouse=a
:set backspace=2
set modeline

map <tab> ==
vmap <tab> =
```