MIT ACM Reference 2008

Combinatorial optimization

1. Max flow (C++)
2. Min cost max flow (C++)
3. (Min cost) maximum matching (C++)
4. Non-bipartite matching (C++)

Geometry

1. Convex hull (C++)
2. Area and centroid (C++)
3. Misc geometry (C++)
4. Voronoi diagrams (C++)

Numerics

1. Euclid's algorithm, etc. (C++)
2. Linear systems, matrix inverse (Stanford) (C++)
3. RREF, matrix rank (C++)
4. Simplex (C++)
5. FFT (C++)

Graphs

1. Dense Dijkstra's (C++)
2. Topological sort (C++)
3. Kruskal's (C++)

Misc

1. Longest Increasing Subsequence (C++)
2. Dates (C++)
3. Knuth-Morris-Pratt (C++)
4. Hashed strstr (C++)

Java samples

1. Java formatting (Java)
2. Complicated regex example (Java)
3. Java geometry (Java)
4. 3D geom (Java)

Max flow (C++)

```
// Fattest path network flow algorithm using an adjacency matrix.
//
// Running time: O(|E|^2 log (|V| * U), where U is the largest
//               capacity of any edge.  If you replace the 'fattest
//               path' search with a minimum number of edges search,
//               the running time becomes O(|E|^2 |V|).
//
```



```
// INPUT: cap -- a matrix such that cap[i][j] is the capacity of
//               a directed edge from node i to node j
//
//               * Note that it is legitimate to create an i->j
//                 edge without a corresponding j->i edge.
//
//               * Note that for an undirected edge, set
//                 both cap[i][j] and cap[j][i] to the capacity of
//                 the undirected edge.
//
//        source -- starting node
//        sink -- ending node
//
// OUTPUT: value of maximum flow; also, the flow[][] matrix will
//         contain both positive and negative integers -- if you
//         want the actual flow assignments, look at the
//         *positive* flow values only.
//
// To use this, create a MaxFlow object, and call it like this:
//
//   MaxFlow nf;
//   int maxflow = nf.getMaxFlow(cap,source,sink);

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

const int INF = 1000000000;

struct MaxFlow {
  int N;
  VVI cap, flow;
  VI found, dad, dist;

  bool searchFattest(int source, int sink){
    fill (found.begin(), found.end(), false);
    fill (dist.begin(), dist.end(), 0);
    dist[source] = INF;
    while (source != N){
      int best = N;
      found[source] = true;
      if (source == sink) break;
      for (int k = 0; k < N; k++){
        if (found[k]) continue;
        int possible = min(cap[source][k] - flow[source][k], dist[source]);
        if (dist[k] < possible) {
          dist[k] = possible;
          dad[k] = source;
        }
        if (dist[k] > dist[best]) best = k;
      }
      source = best;
```

```
    }
    return found[sink];
  }

  bool searchShortest(int source, int sink){
    fill (found.begin(), found.end(), false);
    fill (dist.begin(), dist.end(), INF);
    dist[source] = 0;
    while (source != N){
      int best = N;
      found[source] = true;
      if (source == sink) break;
      for (int k = 0; k < N; k++){
        if (found[k]) continue;
        if (cap[source][k] - flow[source][k] > 0){
          if (dist[k] > dist[source] + 1){
            dist[k] = dist[source] + 1;
            dad[k] = source;
          }
        }
        if (dist[k] < dist[best]) best = k;
      }
      source = best;
    }
    return found[sink];
  }

  int getMaxFlow (const VVI &cap, int source, int sink){
    this->cap = cap;
    N = cap.size();
    found = VI(N);
    flow = VVI(N,VI(N));
    dist = VI(N+1);
    dad = VI(N);

    int totflow = 0;
    while (searchFattest(source, sink)){
      int amt = INF;
      for (int x = sink; x != source; x = dad[x])
        amt = min (amt, cap[dad[x]][x] - flow[dad[x]][x]);
      for (int x = sink; x != source; x = dad[x]){
        flow[dad[x]][x] += amt;
        flow[x][dad[x]] -= amt;
      }
      totflow += amt;
    }

    return totflow;
  }
};
```


Min cost max flow (C++)

```
// Min cost max flow algorithm 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
//
//        source -- starting node
//        sink -- ending node
//
// OUTPUT: max flow and min cost; the matrix flow will contain
//         the actual flow values (note that unlike in the MaxFlow
//         code, you don't need to ignore negative flow values -- there
//         shouldn't be any)
//
// To use this, create a MinCostMaxFlow object, and call it like this:
//
//   MinCostMaxFlow nf;
//   int maxflow = nf.getMaxFlow(cap,cost,source,sink);

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

const int INF = 1000000000;

struct MinCostMaxFlow {
  int N;
  VVI cap, flow, cost;
  VI found, dad, dist, pi;

  bool search(int source, int sink) {
    fill(found.begin(), found.end(), false);
    fill(dist.begin(), dist.end(), INF);
    dist[source] = 0;

    while (source != N) {
      int best = N;
      found[source] = true;
      for (int k = 0; k < N; k++) {
        if (found[k]) continue;
        if (flow[k][source]) {
          int val = dist[source] + pi[source] - pi[k] - cost[k][source];
          if (dist[k] > val) {
```

```
          dist[k] = val;
          dad[k] = source;
        }
      }
      if (flow[source][k] < cap[source][k]) {
        int val = dist[source] + pi[source] - pi[k] + cost[source][k];
        if (dist[k] > val) {
          dist[k] = val;
          dad[k] = source;
        }
      }

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

pair<int,int> getMaxFlow(const VVI &cap, const VVI &cost, int source, int si
  this->cap = cap;
  this->cost = cost;

  N = cap.size();
  found = VI(N);
  flow = VVI(N,VI(N));
  dist = VI(N+1);
  dad = VI(N);
  pi = VI(N);

  int totflow = 0, totcost = 0;
  while (search(source, sink)) {
    int amt = INF;
    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]);
    for (int x = sink; x != source; x = dad[x]) {
      if (flow[x][dad[x]]) {
        flow[x][dad[x]] -= amt;
        totcost -= amt * cost[x][dad[x]];
      } else {
        flow[dad[x]][x] += amt;
        totcost += amt * cost[dad[x]][x];
      }
    }
    totflow += amt;
  }

  return make_pair(totflow, totcost);
```



```
  }
};
```

(Min cost) maximum matching (C++)

```
// This code performs maximum bipartite matching and, optionally min-cost matc
// 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 and MinCostMatching return the number of matches made.
//   MatchingCost will give you the cost, if you need it.
//
// 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;
  }

  // ----------- CUT HERE FOR JUST MAXIMUM MATCHING -----------

  bool Augment(int start, const VVI &w, VI &mr, VI &mc)
  {
    const int INF = (1<<31) - 1;
    VI cost(w.size(), INF);
    VI dad(w.size());
    cost[start] = 0;
    int last = 0;
    for(int i = 0; i < w.size() && last != -1 && cost[start] == 0; i++)  // Bell
      {
        last = -1;
        for(int r = 0; r < w.size(); r++)
          {
            if (cost[r] == INF)
              continue;
            for(int r2 = 0; r2 < w.size(); r2++)
              {
                if (r == r2 || mr[r2] == -1 || w[r][mr[r2]] == NO_EDGE) continue
                int val = cost[r] + w[r][mr[r2]] - w[r2][mr[r2]];
                if (val < cost[r2] && val < 0) {
                  cost[r2] = val;
                  last = r2;
                  dad[r2] = r;
                }
              }
          }
      }

    if (mr[start] == -1)
      {
        int best = -1;
        for(int i = 0; i < w.size(); i++)
          {
            if (cost[i] < 0 && (best == -1 || cost[i] < cost[best]))
              best = i;
          }

        if (best == -1)
```



```
          return false;

        // Augment a non-cycle
        int a = dad[best], b = best;
        VI oldmr = mr;
        mr[best] = -1;
        do {
          mr[a] = oldmr[b];
          mc[mr[a]] = a;
          b = a;
          a = dad[a];
        } while(b != start);
        return true;
      }

    if (last == -1)
      return false;

    if (cost[start] == 0)
      last = start;
    for(int i = 0; i < w.size(); i++)
      last = dad[last];  // Find a cycle

    // Augment.
    VI oldmr = mr;
    int a = last, b;
    do {
      b = a;
      a = dad[a];
      mr[a] = oldmr[b];
      mc[oldmr[b]] = a;
    } while(a != last);

    for(int i = 0; i < w.size(); i++)
      if (mr[i] != -1)
        assert(mc[mr[i]] == i);
    return true;
  }

  // Returns the size of the matching.
  int MinCostMatching(const VVI &w, VI &mr, VI &mc)
  {
    int size = BipartiteMatching(w, mr, mc);
    int blanks = 0;
    for(int start = 0; blanks < w.size(); start = (start+1)%w.size())
      {
        blanks++;
        if (Augment(start, w, mr, mc))
          blanks = 0;
      }
    return size;
```

```
}

int MatchingCost(const VVI &w, const VI &mr)
{
  int c = 0;
  for(int i = 0; i < w.size(); i++)
    if (mr[i] != -1)
      c += w[i][mr[i]];
  return c;
}
```

Non-bipartite matching (C++)

```
/* An implementation of Nick Harvey's algorithm for nonbipartiate max matching
   in undirected, unweighted graphs. Note the algorithm is randomized, so to b
   safe you should run matching() a few times and take the largest one.  If yo
   find a bug in this code, e-mail Jelani Nelson (minilek@mit.edu). */

#define EPS 1e-9

// MAXN is the maximum number of vertices
#define MAXN 64

// adj[i][j] = adj[j][i] = 1 iff the edge (i,j) exists (else both are 0)
char adj[MAXN][MAXN];

// number of vertices in graph
int V;

// counts number of bits of x set to 1
int pc(int x) { return !x?0:(x&1)+pc(x>>1); }

typedef struct matrix {
  vector< vector<long double> > a;
  int n, m;
  matrix(int x, int y) {
    n = x, m = y;
    a = vector< vector<long double> >();
    for (int i = 0; i < n; ++i)
      a.push_back(vector<long double>(m, 0));
  }
  matrix() {
    n = m = 0;
    a = vector< vector<long double> >(0);
  }
  matrix(const matrix &x) {
    n = x.n, m = x.m, a = x.a;
  }
  void operator=(const matrix& x) {
    n = x.n, m = x.m, a = x.a;
```



```
  }
  matrix operator*(const matrix &b) {
    matrix c(n, b.m);
    for (int i = 0; i < c.n; ++i)
      for (int j = 0; j < c.m; ++j)
        for (int k = 0; k < m; ++k)
          c.a[i][j] += a[i][k] * b.a[k][j];
    return c;
  }
  matrix operator+(const matrix &b) {
    matrix c = b;
    for (int i = 0; i < c.n; ++i)
      for (int j = 0; j < c.m; ++j)
        c.a[i][j] += a[i][j];
    return c;
  }
  matrix operator-(const matrix &b) {
    matrix c = b;
    for (int i = 0; i < c.n; ++i)
      for (int j = 0; j < c.m; ++j)
        c.a[i][j] = a[i][j] - c.a[i][j];
    return c;
  }
  matrix operator-() {
    matrix c(n, m);
    for (int i = 0; i < c.n; ++i)
      for (int j = 0; j < c.m; ++j)
        c.a[i][j] = -a[i][j];
    return c;
  }
  long double& operator()(unsigned i, unsigned j) {
    return a[i][j];
  }
  matrix operator()(vector<int> x, vector<int> y) {
    matrix c(x.size(), y.size());
    for (int i = 0; i < c.n; ++i)
      for (int j = 0; j < c.m; ++j)
        c(i, j) = a[x[i]][y[j]];
    return c;
  }
} matrix;

// utility function to print a matrix
void printMatrix(matrix A) {
  for (int i = 0; i < A.n; ++i) {
    for (int j = 0; j < A.m; ++j)
      cout << A(i, j) << " ";
    cout << endl;
  }
  cout << endl;
}
```

```
// returns an LxL identity matrix
matrix identity(int L) {
  matrix I(L, L);
  for (int i = 0; i < L; ++i)
    I(i, i) = 1;
  return I;
}

// returns a maximum size full rank square submatrix of A
// the vector<int> returned is the list of indices of rows used
pair< vector<int>, matrix> max_rank_submatrix(matrix A) {
  vector<int> indices;
  matrix B = A;
  int at = 0;
  vector<int> perm = vector<int>(A.n);
  for (int i = 0; i < A.n; ++i)
    perm[i] = i;
  for (int i = 0; (at < A.n) && (i < A.m); ++i) {
    int maxrow = at;
    for (int j = at + 1; j < A.n; ++j)
      if (fabs(A(j, i)) > fabs(A(maxrow, i)))
        maxrow = j;
    if (maxrow != at) {
      for (int j = 0; j < A.n; ++j)
        swap(A(at, j), A(maxrow, j));
      swap(perm[at], perm[maxrow]);
    }
    if (fabs(A(at, i)) < EPS)
      continue;
    indices.push_back(perm[at]);
    long double c = A(at, i);
    for (int j = i; j < A.m; ++j)
      A(at, j) /= c;
    for (int j = at + 1; j < A.n; ++j) {
      long double c = A(j, i);
      for (int k = i; k < A.m; ++k)
        A(j, k) -= A(at, k) * c;
    }
    ++at;
  }
  sort(indices.begin(), indices.end());
  return make_pair(indices, B(indices, indices));
}

// assumes matrix is non-singular
matrix matrix_inverse(matrix A) {
  // input should have A.n = A.m
  matrix B = matrix(A.n, 2 * A.n);
  for (int i = 0; i < A.n; ++i)
    for (int j = 0; j < A.n; ++j)
```



```
      B(i, j) = A(i, j);
  for (int i = 0; i < A.n; ++i)
    B(i, i + A.n) = 1;
  for (int i = 0; i < A.n; ++i) {
    int maxrow = i;
    for (int j = i + 1; j < A.n; ++j)
      if (fabs(B(j, i)) > fabs(B(maxrow, i)))
        maxrow = j;
    if (maxrow != i)
      for (int j = 0; j < B.m; ++j)
        swap(B(i, j), B(maxrow, j));
    long double c = B(i, i);
    for (int j = i; j < B.m; ++j)
      B(i, j) /= c;
    for (int j = 0; j < A.n; ++j) if (i != j) {
      long double c = B(j, i);
      for (int k = i; k < B.m; ++k)
        B(j, k) -= B(i, k) * c;
    }
  }
  matrix ret = matrix(A.n, A.n);
  for (int i = 0; i < A.n; ++i)
    for (int j = 0; j < A.n; ++j)
      ret(i, j) = B(i, j + A.n);
  return ret;
}

struct matrix T, N;

void delete_edges(vector<int> J) {
  if (J.size() == 2) {
    int i = J[0], j = J[1];
    if (fabs(T(i, j))>EPS && fabs(N(i,j) + 1./T(i, j))>EPS)
      T(i, j) = T(j, i) = 0;
  } else {
    int C = 4;
    for (int i = 1; i <= C; ++i)
      for (int j = i + 1; j <= C; ++j) {
        vector<int> Jp;
        for (int k = (i-1)*J.size()/C; k < i*J.size()/C; ++k)
          Jp.push_back(J[k]);
        for (int k = (j-1)*J.size()/C; k < j*J.size()/C; ++k)
          Jp.push_back(J[k]);
        matrix W = T(Jp, Jp), WHat = N(Jp, Jp);
        delete_edges(Jp);
        matrix w = T(Jp, Jp);
        for (int k = 0; k < Jp.size(); ++k)
          for (int l = 0; l < Jp.size(); ++l)
            N(Jp[k], Jp[l]) = WHat(k, l);
        matrix X = N(J, J) - N(J, Jp) *
          matrix_inverse(identity(Jp.size()) + (w-W)*WHat)*(w-W)*N(Jp, J);
```

```
        for (int k = 0; k < J.size(); ++k)
          for (int l = 0; l < J.size(); ++l)
            N(J[k], J[l]) = X(k, l);
      }
    }
  }

  // returns a vector<int> v
  // v[i] = -1 if i isn't matched, else v[i] is the vertex i is matched to
  #define MAXRAND 10000
  vector<int> matching() {
    T = matrix(V, V);
    for (int i = 0; i < V; ++i)
      for (int j = i + 1; j < V; ++j)
        if (adj[i][j])
          T(i, j) = (rand() % MAXRAND) + 1;
    for (int i = 0; i < V; ++i)
      for (int j = i; j < V; ++j)
        T(j, i) = -T(i, j);
    pair< vector<int>, matrix > x = max_rank_submatrix(T);
    vector<int> indices = x.first;
    if (indices.size() == 0)
      return vector<int>(V, -1);

    // make the number of vertices in T a power of 2 while keeping
    // full rank (put the new vertices in a clique)
    // Nick's algorithm assumes #vertices is a power of 2
    T = x.second;

    int newLength = T.n;
    while (pc(newLength) > 1)
      ++newLength;
    matrix newT = matrix(newLength, newLength);
    for (int i = 0; i < T.n; ++i)
      for (int j = 0; j < T.n; ++j)
        newT(i, j) = T(i, j);
    for (int i = T.n; i < newT.n; ++i)
      for (int j = i + 1; j < newT.n; ++j) {
        newT(i, j) = (rand() % MAXRAND) + 1;
        newT(j, i) = -newT(i, j);
      }
    T = newT;

    N = matrix_inverse(T);
    vector<int> vertices;
    for (int i = 0; i < T.n; ++i)
      vertices.push_back(i);
    delete_edges(vertices); // deletes edges until left with a perf. matching
    vector<int> v = vector<int>(V, -1);
    for (int i = 0; i < indices.size(); ++i)
      for (int j = 0; j < indices.size(); ++j)
```



```
        if (fabs(T(i, j)) > EPS)
          v[indices[i]] = indices[j];

    // make sure this is a valid matching
    for (int i = 0; i < v.size(); ++i)
      if (v[i] != -1) {
        if (v[v[i]] != i)
          cout << "failed at " << i << endl;
        assert(v[v[i]] == i);
      }
    return v;
  }

  int main() {
    timeval tp;
    gettimeofday(&tp, NULL);
    srand(tp.tv_usec);

    /* EXAMPLE USAGE */

    // set the adj array and #vertices here
    V = 64;
    memset(adj, 0, sizeof(adj));
    for (int i = 0; i < V; i ++)
      for (int j = i + 1; j < V; ++j)
        if (rand()%50 == 0) // making it somewhat sparse so there's no perf. ma
          adj[i][j] = adj[j][i] = 1;

    // find the max matching
    vector<int> matches = matching();

    // make sure max matching only used real edges!
    for (int i = 0; i < V; ++i)
      if (matches[i] != -1)
        assert(adj[i][matches[i]]);

    for (int i = 0; i < matches.size(); ++i)
      cout << i << ": " << matches[i] << endl;

    /* END OF EXAMPLE */

    return 0;
  }
```

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++)

```
// 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 tl
      // 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++)

```
// This is a simple simplex solver.  It solves:
// Maximize obj[0] + obj[1]*x*1 + ... + obj[n]*x_n
// Subject to
//  x_1 >= 0, ..., x_n >= 0
//  for each i, c[i][0] + c[i][1]*x_1 + ... + c[i][n]*x_n >= 0

// DO NOT TRY TO REUSE LP OBJECTS!!!!!  (INFEASIBLE corrupts them.)
// You should consider calling srand() first.

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

class LP
{
public:
  int nvars, ncons;  // # decision vars and # constraints
  VD obj;            // [cols]
  VVD c;             // ncons * cols (left column is constant)
```

```
    // Results in intelligible form
    double objval;
    VD assignments;

    enum Result {FAILED, INFEASIBLE, UNBOUNDED, FEASIBLE, OPTIMAL};

  private:
    int cols;          // width of the constraint matrix
    VI nonbasic_orig;  // [nvars]
    VI basic_orig;     // [ncons]

  public:
    LP(int nvars, int ncons) : nvars(nvars), ncons(ncons),
                               cols(1 + nvars),
                               obj(1 + nvars, 0.0)
    {
      c = VVD(ncons, VD(cols, 0.0));
      for(int i = 0; i < nvars; i++)
        nonbasic_orig.push_back(i);
      for(int i = 0; i < ncons; i++)
        basic_orig.push_back(i + nvars);
    }

    void pivot(int col, int row)
    {
      // Enforce that the old col remains nonnegative.
      {
        double val = 1.0 / c[row][col];
        for (int i = 0; i < cols; i++)
          c[row][i] *= -val;
        c[row][col] = val;
      }

      // Subtract the extra stuff the pivot row brings along.
      for (int i = 0; i < ncons; i++) {
        if (i == row) continue;
        double coeff = c[i][col];
        c[i][col] = 0.0;
        for (int j = 0; j < cols; j++)
          c[i][j] += coeff * c[row][j];
      }
      double coeff = obj[col];
      obj[col] = 0.0;
      for (int j = 0; j < cols; j++)
        obj[j] += coeff * c[row][j];

      // Update maps to original indices.
      swap(nonbasic_orig[col - 1], basic_orig[row]);
    }
```



```
    // Returns true if successful, false if unbounded
    bool simplex()
    {
      // Bland's rule: pick an arbitrary column and
      // do the pivot that will change it the least.
      while (true) {
        // Pick a random nonbasic column to pivot.
        int offset = rand() % (cols - 1), col = -1;
        for (int i = 0; i < cols - 1; i++) {
          int c = (offset + i) % (cols - 1) + 1;
          if (obj[c] > 1e-8) {
            col = c;
            break;
          }
        }
        if (col == -1)
          break;  // This basis is optimal.

        // Find the row that will hit zero first.
        double min_change = 1e100;
        int best_row = -1;
        for (int row = 0; row < ncons; row++) {
          if (c[row][col] >= -1e-8) continue;
          double change = -c[row][0] / c[row][col];
          if (change < min_change) {
            min_change = change;
            best_row = row;
          }
        }

        if (best_row == -1)  // Unbounded!
          return false;

        pivot(col, best_row);
      }

      // Produce output
      objval = obj[0];
      assignments.resize(ncons + nvars);
      for (int i = 0; i < ncons; i++)
        assignments[basic_orig[i]] = c[i][0];
      for (int i = 0; i < nvars; i++)
        assignments[nonbasic_orig[i]] = 0.0;
      return true;
    }

    Result phase1()
    {
      // Find equation with minimum b
      int worst_row = 0;
      for (int i = 1; i < ncons; i++)
```

```
      if (c[i][0] < c[worst_row][0])
        worst_row = i;

    if (c[worst_row][0] >= -1e-8)
      return FEASIBLE;

    // Add a new variable epsilon, which we minimize.
    for (int i = 0; i < ncons; i++)
      c[i].push_back(1.0);
    VD orig_obj = obj;
    obj = VD(cols, 0.0);
    obj.push_back(-1.0);
    int eps_var = nvars + ncons;
    nonbasic_orig.push_back(eps_var);
    nvars++;
    cols++;

    // We started out infeasible, so pivot epsilon into the basis.
    pivot(cols-1, worst_row);
    if (!simplex())
      return FAILED;  // Unbounded phase 1 here is bad.
    if (objval < -1e-9)
      return INFEASIBLE;  // Epsilon must be nonpositive.

    // Force epsilon out of the basis
    // (It's zero anyway within our precision).
    for (int i = 0; i < ncons; i++) {
      if (basic_orig[i] == eps_var) {
        pivot(1, i);
        break;
      }
    }

    // Find epsilon's column.
    int eps_col = -1;
    for (int i = 0; i < nvars; i++)
      if (nonbasic_orig[i] == eps_var)
        eps_col = i+1;

    // Epsilon is nonbasic and thus zero, so we can remove it.
    for (int i = 0; i < ncons; i++) {
      c[i][eps_col] = c[i][cols-1];
      c[i].pop_back();
    }
    nonbasic_orig[eps_col - 1] = nonbasic_orig.back();
    nonbasic_orig.pop_back();
    cols--;
    nvars--;

    // Restore the original objective.
    obj = VD(cols, 0.0);
```



```
    obj[0] = orig_obj[0];
    for (int i = 0; i < nvars; i++) {
      if (nonbasic_orig[i] < nvars)
        obj[i+1] = orig_obj[nonbasic_orig[i] + 1];
    }
    for (int i = 0; i < ncons; i++) {
      if (basic_orig[i] < nvars)
        for (int j = 0; j < cols; j++)
          obj[j] += orig_obj[basic_orig[i] + 1] * c[i][j];
    }
    return FEASIBLE;
  }

  Result solve()
  {
    Result p1_res = phase1();
    if (p1_res != FEASIBLE)
      return p1_res;
    assignments.clear();  // Poison it.
    if (!simplex())
      return UNBOUNDED;
    return OPTIMAL;
  }

  void printState()
  {
    printf("Maximize %lf ", obj[0]);
    for(int i = 1; i < cols; i++)
      printf(" + %lf*x_%d", obj[i], nonbasic_orig[i-1]);
    printf("\nSubject to\n      ");
    for(int i = 1; i < cols; i++) {
      printf("x%-7d", nonbasic_orig[i-1]);
    }
    for(int i = 0; i < ncons; i++)
      {
        printf("\nx%-5d", basic_orig[i]);
        for(int j = 1; j < cols; j++)
          printf("%8.4lf", c[i][j]);
        printf(" + %lf >= 0", c[i][0]);
      }
    printf("\n\n");
  }

  void printResult()
  {
    printf("Objective = %.6lf\n", objval);
    for (int i = 0; i < nvars; i++)
      printf(" x%d = %.6lf\n", i, assignments[i]);
    for (int i = 0; i < ncons; i++)
      printf(" r%d = %.6lf\n", i, assignments[nvars + i]);
  }
```

```
};
```

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)
```


MIT ACM Reference 2008 file:///mit/ecprice/acm/acm08/notebook.html#file7

```
	{
		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 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]

                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[>

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++)

```
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)

```
// 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 # follo

        // 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 "1,2345,67

        // 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, wher
            // line is terminated by a period.  The code below reads lines unt
            // encountering a line whose final character is a '.'.  Note the u
            //
            //    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
            //
            // Other useful String manipulation methods include
            //
            //    s.compareTo(t) < 0 if s < t, lexicographically
            //    s.indexOf("apple") returns index of first occurrence of "app
            //    s.lastIndexOf("apple") returns index of last occurrence of "
            //    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/upperc
            //
            //    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)
            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
        //                                                    double w, double
        //
        //     creates an ellipse inscribed in box with bottom-left corner (x,
        //     and upper-right corner (x+y,w+h)
        //
        //   Rectangle2D.Double rect = new Rectangle2D.Double (double x, doubl
        //                                                     double w, doubl
        //
        //     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 (re

    }
}
```

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));
  }
}
```

```
### .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
```