```
|| ================ 2_conn_comp.cc ================
|| ======= 2-connected components ===========
||
f0 vi v[111111];
2f int p[111111];
d0 int b[111111];
97 int u[111111];
5c int now,cnt;
11 vi c;
||
40 int dfs(int ver,int par=-1){
79  c.pb(ver);
aa  u[ver] = now;
63  p[ver] = ++cnt;
5b  int val = cnt;
af  REP(i,v[ver].size()){
bb   int nv = v[ver][i];
cb   if(nv==par) continue;
9b   c.pb(ver);
02   if(u[nv]==now) val = min(val, p[nv]);
19   else{
49    int t = dfs(nv, ver);
a9    val = min(val, t);
11    if(t>=p[ver]){
05     vi z;
d2     while(c.size() && c.back()!=ver) z.pb(c.back
||                                    ()),c.pop_back();
95     z.pb(ver);
38     UN(z);
||
be     ADD_COMPONENT(z);
80    }
c1   }
43  }
98  return b[ver]=val;
19 }
|| =================== 2sat.h ===================
|| 2sat
||
|| Each var integer between 1 and MAXV.
|| Positive means true, negative false
|| Logical assertions:
||  vee(a,b) eq(a,b) implies(a,b) tru(a)
|| To recover a variable assignment:
||  repeat
||   call commit(a) on an a s.t. can(a)
||   or do similarly on -a.
|| Optimal choices above (np-complete)
||
|| Mainline reads a number of disjunct-
|| ive pairs.  Prints "satisfiable" or
|| "not satisfiable".  Prints an
|| assignment (arbitrarily chosen.)
|| ----stop reading here if naive is
||  fast enough
|| Could be sped up by having an array
|| "must" and doing
||  if (!can(blah)) must[-blah] = 1;
|| Then in "can" check
||  if (must[-x]) return 0;
|| Careful!  must[x] does *not* imply
|| can(x).
||
79 #include <stdio.h>
5e #include <string.h>
6f #include <algorithm>
a8 using namespace std;
||
||
71 #define RF(i,a,b) for (int i=(a)-1;i>=\
93  (b);i--)
4e #define ROF(i,n) RF(i,n,0)
||
60 #define MAXV 100
fb #define MAXC 100
||
5c struct ee{ int x,y; } e[2*MAXC+1];
1f int xfe[2*MAXV+1];
84 int *firste = xfe+MAXV;
3f unsigned xmark[2*MAXV+1];
cc unsigned *mark = xmark+MAXV;
ba int ne,cookie;
||
34 void reset(void) {
c0  ne = 0;
5d  memset(xmark,0,sizeof(xmark));
45  memset(xfe, 0, sizeof(xfe));
a7 }
||
5b int comp(const ee &a, const ee &b) {
0e  return a.x < b.x;
ad }
||
0c void setup(void) {
dd  int i;
f4  e[ne].x = MAXV+99;
d0  sort(e,e+ne,comp);
6f  ROF(i,ne) firste[e[i].x] = i;
4e }
||
08 void edge(int x, int y) {
b9  e[ne].x = x;
5c  e[ne++].y = y;
8f }
||
f0 #define vee(a,b) (edge(-(a),b),\
8e  edge(-(b),a))
4e #define eq(a,b) (edge(a,b),edge(b,a),\
4d  edge(-(a),-(b)),edge(-(b),-(a)))
fd #define implies(a,b) (edge(a,b),\
e9  edge(-(b),-(a)))
be #define tru(a) (edge(-(a),a))
||
63 int Xcan(int x) {
45  int i;
b6  if (mark[-x] >= cookie) return 0;
eb  if (mark[x] >= cookie) return 1;
1e  mark[x] = cookie;
12  for (i=firste[x];
ec   e[i].x==x && Xcan(e[i].y);i++);
b0  return e[i].x != x;
48 }
||
dd void commit(int x) { int c=cookie;
26  cookie=-1;Xcan(x);cookie=c; }
3f int can(int x) {
8a  return (cookie++,Xcan(x)); }
|| ================== affine.cc ===================
|| Affine transformation matrices
|| At the end is a bit of code that calculates the
||                                  joint positions
|| of the robot from the 1999 Finals Robot problem.
||
|| This uses a right-handed coordinate system.
|| If you are facing forwards, positive rotations
||                                 will result in:
||  yaw (turn to your right)
||  pitch (turn upwards)
||  roll (do a cartwheel to the right)
||
79 #include <stdio.h>
7b #include <math.h>
||
c4 struct Matrix
e5 {
c2  double m[4][4]; // row then column
da  Matrix(void) {for(int i=0;i<4;i++) for(int j=0;
||                      j<4;j++) m[i][j] = (i==j);}
c8  double * operator [] (int i) { return m[i]; }
06 };
||
d2 struct Point
7d {
f1  double p[4];
a3  Point(void) { p[0] = p[1] = p[2] = 0; p[3] = 1;
||                                                 }
89  double& operator [] (int i) { return p[i]; }
61 };
||
|| Note: the const is required if you want to chain
||                               together operations
||  and use transformation matrices directly
|| Note: the short-hand direct matrix access
||                     notation isn't used because
||  it invalidates the const
d4 Point operator *(const Matrix &a, const Point &b
||                                                 )
6d {
0f  Point c;
d2  for (int i=0;i<4; i++)
b5  {
2f   c.p[i] = 0;
94   for (int j=0;j<4; j++)
43    c.p[i] += a.m[i][j] * b.p[j];
cf  }
dd  return c;
43 }
||
11 Matrix operator *(const Matrix &a, const Matrix
||                                               &b)
2b {
ee  Matrix c;
11  for (int i=0;i<4;i++)
54   for (int j=0; j<4;j++) {
18    c.m[i][j] = 0;
f9    for (int k=0; k<4; k++)
b5     c.m[i][j] += a.m[i][k]*b.m[k][j];
60   }
6e  return c;
3e }
||
a3 Matrix rotatex(double s)
06 {
7a  Matrix a;
51  a[1][1] = cos(s);
b9  a[1][2] = -sin(s);
5b  a[2][1] = sin(s);
4a  a[2][2] = cos(s);
f9  return a;
ea }
||
ab Matrix rotatey(double s)
06 {
3a  Matrix a;
49  a[0][0] = cos(s);
7b  a[0][2] = sin(s);
92  a[2][0] = -sin(s);
64  a[2][2] = cos(s);
80  return a;
99 }
||
b3 Matrix rotatez(double s)
06 {
fa  Matrix a;
49  a[0][0] = cos(s);
a1  a[0][1] = -sin(s);
69  a[1][0] = sin(s);
22  a[1][1] = cos(s);
9f  return a;
a9 }
||
8b Matrix translate(double x, double y, double z)
07 {
3a  Matrix a;
b5  a[0][3] = x;
56  a[1][3] = y;
ee  a[2][3] = z;
a0  return a;
c8 }
||
e0 Matrix scale(double sx, double sy, double sz)
dc {
67  Matrix a;
c2  a[0][0] = sx;
b3  a[1][1] = sy;
b5  a[2][2] = sz;
e8 }
||
||
ed double len[100], joint[100];
||
27 #define PI (atan2(0,-1))
||
e3 int main(void)
c4 {
62  int numLinks;
c1  while (true)
0d  {
67   scanf("%d", &numLinks);
49   if (numLinks < 0) break;
61   for (int n=0; n<numLinks; n++)
ec    scanf("%lf", &len[n]);
c8   for (int n=0; n<numLinks; n++)
71    scanf("%lf", &joint[n]);
e0   Matrix a;
be   Point b;
9c   for (int n=0; n<numLinks; n++)
57   {
f1    if (n%2 == 0)
47     a = a *rotatey(joint[n] / 180.0 * PI);
e8    else
7c     a = a*rotatex(joint[n] / 180.0 * PI);
7b    a = a*translate(0,0,len[n]);
98    Point c= a*b;
f5    printf("%lf %lf %lf\n", c[0], c[1], c[2]);
fa   }
d8   printf("\n");
f3  }
||
c3  return 0;
fb }
|| ============ approx-subset-sum.cc ============
02 #include <math.h>
69 #include <stdio.h>
7e #include <vector>
2e #include <algorithm>
ce using namespace std;
||
b3 #define ll long long
||
|| "Subset Sum": Given a set of positive
|| integers S and a positive integer t,
||  find a subset of S whose sum is as
||  large as possible but <= t.
||
|| Approximate Subset Sum returns a value
|| less than or equal to the optimal so-
|| lution and greater than or equal to
|| (1-eps) times the optimal solution.
||
1c vector<ll> trim(const vector<ll> &L,
e5  double sigma) {
82  int m = L.size();
a5  vector<ll> Lp;
7d  Lp.push_back(L[0]);
2f  long last = L[0];
c3  for(int i = 1; i < m; i++) {
60   if(last < (1 - sigma) * L[i]) {
45    Lp.push_back(L[i]);
64    last = L[i];
26   }
6c  }
de  return Lp;
f3 }
||
3e ll Approx_Subset_Sum(vector<ll> S, ll t,
df  double eps) {
6e  int n = S.size();
5f  vector<ll> Li, Li_1;
5b  Li_1.push_back(0);
78  for (int i = 0; i < n; i++) {
d5   Li = Li_1;
ee   for(int j = 0; j < Li_1.size(); j++)
c6    Li.push_back(Li_1[j] + S[i]);
47   sort(Li.begin(), Li.end());
05   Li.resize(unique(Li.begin(),
f8    Li.end())-Li.begin());
5d   Li = trim(Li, eps / n);
95   Li_1.clear();
6d   for (int k = 0; k < Li.size(); k++)
79    if (Li[k] <= t)
17     Li_1.push_back(Li[k]);
ed  }
||
3b  return Li_1.back();
5c }
||
c4 int main(void) {
b7  freopen("F.DAT", "r", stdin);
7f  int t;
77  scanf("%d", &t);
e0  while(t-- > 0) {
fb   int n, i;
71   int k;
b9   vector<ll> S;
27   scanf("%d", &n);
0b   for(i = 0; i < n; i++) {
33    int q;
a9    scanf("%d", &q);
4c    S.push_back(q);
9f   }
86   sort(S.begin(), S.end());
91   scanf("%d", &k);
ba   printf("%lld\n",
9e    Approx_Subset_Sum(S,k,0.1));
91  }
31  return 0;
74 }
|| ================ assignment.cc =================
|| ========== Assignment ============
||
8c #define C(i,j) (A[i][j]-pr[i]-pc[j])
a3 int A[N][N],pr[N],pc[N],f[N];
46 bool b[N],bx[N],by[N];
65 bool dfs(int x){
34  bx[x]=true;
0a  REP(y,n)if(!C(x,y) && (by[y]=true,f[y]<0 || !bx
||                            [f[y]] && dfs(f[y])))
3a   return f[y]=x,true;
8c  return false;
41 }
b3 int assignment(){
cb  for(CL(pr,0),CL(pc,0);CL(f,-1);){
fb   REP(i,n)CL(bx,0),b[i]=dfs(i);
da   CL(bx,0),CL(by,0);
89   REP(i,n)if(!b[i])dfs(i);
e8   int p=-1u/2;
a4   REP(i,n)REP(j,n)if(bx[i] && !by[j])p=min(p,C(i
||                                              ,j));
d8   if(p==-1u/2) return accumulate(pr,pr+n,
||                           accumulate(pc,pc+n,0));
93   REP(i,n)pr[i]+=bx[i]?p:0,pc[i]-=by[i]?p:0;
79  }
ae }
|| =================== bicon.h ====================
|| Biconnected components in a graph.
||
|| Input:  undirected, not necessarily
||  connected.
||  we use standard *directed* graph
||  data structure [make sure the
||  reverse edges are there too]
|| Returns:
|| - Biconnected components, including
||  1-vertex components.
|| - Cut-edges (edges whose removal
||  disconnects a connected component)
|| - Cut-vertices (vertices whose
||  removal disconnects a component)
|| mainline mostly does "Safe Networks"
||
79 #include <stdio.h>
8c #include <stdlib.h>
f3 #include <set>
e4 #include <vector>
1b #include <algorithm>
bd using namespace std;
||
c9 #define FORALL(it,st) for (typeof(st.\
0f  end())it=st.begin();it!=st.end();it++)
b0 #define FR(i,a,b) for (int i=(a);i<(b)\
2b  ;i++)
eb #define FOR(i,n) FR(i,0,n)
f0 #define PB push_back
4c #define MP make_pair
||
90 struct ee {
db  int from, to;
e8 } e[2000000];
||
a2 int firste[1000000];
||
3a int nv, ne;
||
51 int pre[1000000],//preorder visit order
ec  lowp[1000000],//lowest pre for cycle
24  stack[1000000], sp;//component stack
||
bb vector<int> artic;
1e vector<pair<int,int> > bridge;
0c vector<set<int> > vcomp;
||
97 int bicon(int me, int pp, int p) {
e3  int i,v, bdgs=0, comps=0;
48  if (!p) sp = -1;
37  stack[++sp] = me;
e5  pre[me] = lowp[me] = p++;
32  for (i=firste[me];e[i].from==me;i++) {
bd   v = e[i].to;
14   if (!pre[v]) {
ce    p = bicon(v, pre[me], p);
40    lowp[me] <?= lowp[v];
27    if (lowp[v] == pre[me]) {
4c     set<int> foo;
b2     foo.insert(me); foo.insert(v);
16     for(; stack[sp]-v; sp--)
77      foo.insert(stack[sp]);
03     sp--;
10     vcomp.PB(foo);
b5     comps++;
ae    } else if (lowp[v] == pre[v]) {
88     bridge.PB(MP(me,v));
bc     sp--;
65     bdgs++;
66    }
1e   }
73   else if (pre[v] < pp)
be    lowp[me]<?=pre[v];
90  }
c7  if (bdgs + comps + !!pp >= 2)
5c   artic.push_back(me);
5e  if (lowp[me] == pre[me] && !comps) {
dc   set<int> foo;
48   foo.insert(me);
6d   vcomp.push_back(foo);
76  }
af  return p;
48 }
||
5b int comp(const ee&a, const ee&b){
ed  if (a.from==b.from) return a.to<b.to;
3c  return a.from < b.from;
9c }
|| ================== bigdec.jh ===================
|| Java BigDecimal.  Run through the
|| C preprocessor with:
|| cat bigdec.j|gcc -E -P ->bigdec.java
|| sqrt() is exact.  None of the trans-
|| cendental functions are exact; expect
|| to lose a few digits of precision
|| every time you use one.
e2 import java.math.*;
||
6e #define bd BigDecimal
0a #define BD new bd
4a #define bi BigInteger
7b #define BI new bi
8e #define BI2 bi.valueOf
e5 #define FOR(i,n) for (int i=0;i<n;i++)
89 #define RD BigDecimal.ROUND_HALF_EVEN
||
e3 class bigdec {
63  static int PREC = 50;
||
79  static bd fix(bd a) {
65   return a.setScale(PREC, RD); }
||
a6 #define MK(fun, name) \
cd  static bd name(bd a, bd b) { \
43   return fix(fix(a).fun(fix(b)));}
||
3d  MK(add,add)
0e  MK(subtract,sub)
5c  MK(multiply,mul)
99 #undef MK
||
ee  static int cmp(bd a, bd b) {
d7   return a.compareTo(b);
ab  }
||
37  static bd div(bd a, bd b) {
0f   return fix(a).divide(fix(b),PREC,RD);
1d  }
||
b9  static bd sqrt(bd d) {
ba   PREC += 4;
a1   bd x =
fe    BD(Math.sqrt(d.doubleValue()));
f2   FOR(zzz,22) // prec < 2^22 => accurate
a8    x = div(add(x, div(d,x)), BD(2));
d5   PREC -= 4; return x;
43  }
||
3a  static bd floor(bd d) {
0b   bd f = BD(d.toBigInteger());
6c   if (cmp(d, BD(0)) < 0 &&cmp(d,f) != 0)
c6    return sub(f, BD(1));
8e   return fix(f);
81  }
    // works great when |d|<=1
d8  static bd lame_exp(bd d) {
9d   PREC += 20;
b0   bd term = d, ans = add(d, BD(1));
64   FOR(i,50) {
3b    term = mul(term, div(d, BD(i+2)));
ba    ans = add(ans, term);
48   }
1a   PREC -= 20;
b0   return ans;
68  }
    // e^x; works great
1b  static bd exp(bd d) {
89   if (cmp(d,BD(0)) > 0)
65    return div(BD(1),exp(d.negate()));
b2   PREC += 10;
b4   int mm=0;
5f   while (cmp(d, BD(-.001)) < 0) {
b6    mm++; d = div(d, BD(2));
d8   }
bf   d = lame_exp(d);
0b   while (mm-- > 0) d = mul(d,d);
c3   PREC -= 10;
8c   return fix(d);
09  }
b8  static bd E = exp(BD(1));
    // works great
45  static bd log(bd d) {
cb   PREC += 4;
dc   bd mm = BD(2);
1f   while (cmp(d, BD(2)) > 0
cb    || cmp(d, BD(.5)) < 0) {
5c    d = sqrt(d); mm = mul(mm, BD(2)); }
d2   bd v,n,m;
73   v=n= div(sub(d,BD(1)), add(d,BD(1)));
5b   m = mul(n,n);
b7   FOR(i,PREC) {
ce    n = mul(n,m);
b2    v = add(v, div(n, BD(3+i+i)));
a0   }
72   PREC -= 4;
25   return mul(v, mm);
||
7d  }
    // 3.1415926568979323846264338327950...
a3  static bd pi() {
3e   PREC += 5;
52   bd a=BD(1), b=sqrt(BD(.5)),
5f    c=BD(.5), pow2k=BD(2);
86   FOR(zzz,44) {
1f    bd t=add(a,b); b=sqrt(mul(a,b));
e1    a = div(t, BD(2));
26    c = sub(c,mul(pow2k,
80     sub(mul(a,a),mul(b,b))));
f7    pow2k = add(pow2k, pow2k);
fa   }
10   a = mul(a,a); a = add(a,a);
da   PREC -= 5; return div(a,c);
fb  }
a3  static bd PI=pi();
    // works great when |x|<.1
cd  static bd lame_omcos(bd x) {
9b   x = mul(x,x).negate();
f3   bd ans = BD(0), term = div(x, BD(-2));
d8   FOR(i,200) {
48    ans = add(ans, term);
42    term = mul(term, x);
35    term = div(term,BD(2*(i+2)*(i+i+3)));
34   }
7b   return ans;
c6  }
    // works great
15  static bd cos(bd x) {
35   PREC += 50;
5c   System.out.println(x);
a4   bd k = floor(div(x, add(PI,PI)));
06   x = sub(x, mul(k, add(PI,PI)));
4b   int bi = 0;
c8   while (cmp(x.abs(), BD(1)) > 0) {
11    bi++; x = div(x, BD(2));
4d   }
e6   x = lame_omcos(x);
04   while (bi-- > 0)
89    x = mul(add(x,x), sub(BD(2), x));
d5   PREC -= 50;
eb   return sub(BD(1), x);
72  }
    // sine
61  static bd sin(bd x) {
41   return cos(sub(x, div(PI, BD(2))));
17  }
    // tangent
60  static bd tan(bd x) {
63   return div(sin(x),cos(x)); }
    // works great when |x| < 1/5
7d  static bd lame_atan(bd x) {
ee   bd pwr = x, ans = x;
e4   FOR(i,PREC) {
13    pwr = mul(pwr, mul(x,x)).negate();
d4    ans=add(ans,div(pwr,BD(i+i+3)));
93   }
52   return ans;
d7  }
    // works great
43  static bd atan(bd x) {
80   bd y = x;
d9   FOR(zzz,5) y = div(y, add(BD(1),
98    sqrt(add(BD(1), mul(y,y)))));
95   y = lame_atan(y);
cd   FOR(zzz,5) y = add(y,y);
7a   return y;
d6  }
|| } -- easier to test if absent
|| ================= bigint.java ==================
4c import java.io.*;
eb import java.math.*;
42 import java.text.*;
||
a1 public class D{
96  static StreamTokenizer in =
5b   new StreamTokenizer(new InputStreamReader(
||                                       System.in));
      // Reading from file:
      // new StreamTokenizer(new FileReader("file.
||                                             in"));
||
21  public static void main(String[] args) throws
||                                      Exception {
68   BigInteger T = BigInteger.valueOf(0);
25   BigInteger TB = BigInteger.valueOf(0);
a0   BigInteger NTB = BigInteger.valueOf(0);
35   BigInteger S = BigInteger.valueOf(0);
7a   BigInteger MAX = BigInteger.valueOf(1);
9c   int j;
10   for (j=0;j<100;j++) MAX = MAX.multiply(
||                          BigInteger.valueOf(10));
24   for(;;){
73    int i,t,a,b;
0c    if (in.nextToken() != StreamTokenizer.TT_
||                                  NUMBER) break;
89    t = (int) in.nval;
f9    if (in.nextToken() != StreamTokenizer.TT_
||                                  NUMBER) break;
3f    a = (int) in.nval;
94    if (in.nextToken() != StreamTokenizer.TT_
||                                  NUMBER) break;
95    b = (int) in.nval;
||
0d    System.out.print("(");
6d    System.out.print(t);
55    System.out.print("^");
f8    System.out.print(a);
09    System.out.print("−1)/(");
0d    System.out.print(t);
d4    System.out.print("^");
a0    System.out.print(b);
34    System.out.print("−1) ");
cc    if (t == 1 || a%b != 0) {
e3     System.out.print("is not an integer with
||                       less than 100 digits.\n");
0c     continue;
ad    }
||
d1    T = BigInteger.valueOf(t);
19    TB = BigInteger.valueOf(1);
0a    for (i=0;i<b;i++){
69     TB = TB.multiply(T);
49     if (TB.compareTo(MAX) >= 0) break;
2f    }
e8    NTB = BigInteger.valueOf(1);
c1    S = BigInteger.valueOf(0);
3b    for (i=0;i<a;i+=b) {
c3     S = S.add(NTB);
fa     if (S.compareTo(MAX) >= 0) break;
f4     NTB = NTB.multiply(TB);
5a    }
a0    if (S.compareTo(MAX) >= 0)
7e     System.out.print("is not an integer with
||                         less than 100 digits.");
ff    else System.out.print(S);
13    System.out.print("\n");
dd   }
ab  }
18 }
|| =================== bignum.c ===================
|| Unsigned (or 9’s complement) bignums − fixed
||                                            size
||
|| add, sub, mul work fine for signed.  cmp and div
||                                          don’t!
||
c4 #define SZ 100   // must be bigger than an int’s
||                                           worth
||
dd typedef char bn[SZ];
||
13 void add(char *a, char *b) {
8f  int i,j,k=0;
54  for (i=0;i<SZ;i++) {
cf   j = a[i]+b[i]+k;
74   a[i] = j%10;
51   k = j>=10;
17  }
2d }
||
95 void sub(char *a, char *b) {
96  int i,j,k=0;
17  for (i=0;i<SZ;i++) {
56   j = a[i]−b[i]−k;
c9   a[i] = (j+10)%10;
49   k = j<0;
97  }
19 }
||
7f int cmp(char *a, char *b) {
42  int i;
b4  for (i=SZ−1;i>0 && a[i]==b[i];i−−);
bc  return a[i]−b[i];
80 }
||
ea void copy(char *a, char *b) {
cc  memcpy(a,b,SZ);
13 }
||
2e void lshift(char *a) {
e1  int i;
44  for (i=SZ−1;i>0;i−−) a[i]=a[i−1];
ef  a[0]=0;
9a }
||
1b void mul(char *a, char *b) {
4a  int i,j;
48  bn r;
22  sub(r,r);
```

```
36  for (i=0;i<SZ;i++) {
f8   for (j=0;j<b[i];j++) add(r,a);
1a   lshift(a);
15  }
93  copy(a,r);
31 }
||
cc void set(char *a, int n) {
d2  int i=0;
31  sub(a,a);
8b  while (n) {
25   a[i++] = n%10;
96   n/=10;
01  }
fd }
||
ab void print(char *a) {
5b  int i;
4e  for (i=SZ−1;i>0 && !a[i];i−−) {}
29  for (;i>=0;i−−) printf("%01d",a[i]);
a4 }
||
b8 void signedprint(char *a) {
37  bn t;
d9  if (a[SZ−1] >= 5) {
b1   printf("−");
36   set (t,0);
14   sub (t,a);
62   print(t);
70  } else print(a);
6e }
||
f6 void leftshift(char *n) {
bb  int i;
35  for (i=SZ−1;i>0;i−−) n[i] = n[i−1];
7d  n[0] = 0;
0e }
||
f6 void rightshift(char *n) {
9b  int i;
75  for (i=1;i<SZ;i++) n[i−1] = n[i];
45  n[i−1] = 0;
d7 }
||
a8 void div(char *dividend, char *divisor, char *
||                        quotient, char *remainder) {
9d  int shift = −1;
0e  int i;
18  for (i=0;i<SZ;i++) remainder[i] = dividend[i];
c6  for (i=0;i<SZ;i++) quotient[i] = 0;
e4  while (cmp(remainder,divisor) >= 0) {
7b   shift++;
ad   if (divisor[SZ−1]) goto full;
71   leftshift(divisor);
8e  }
d4  for(;shift>=0;shift−−) {
ed   rightshift(divisor); full:
45   while (cmp(remainder,divisor) >= 0) {
c0    sub(remainder,divisor);
c7    quotient[shift]++;
0b   }
a5  }
b0 }
||
||
||
d2 int main() {
e4  bn a,b,c,d;
d5  int i,j;
||
09  while (2 == scanf("%d%d",&i,&j)) {
||
f8   set(a,i);
d0   signedprint(a); printf("\n");
b9   set(b,j);
a6   signedprint(b); printf("\n");
||
4a   printf("cmp %d\n",cmp(a,b));
||
78   add (a,b);
d3   signedprint(a); printf("\n");
||
f8   set(a,i);
a6   sub(a,b);
32   signedprint(a); printf("\n");
||
f8   set(a,i);
9e   mul(a,b);
b3   signedprint(a); printf("\n");
||
f8   set(a,i);
81   leftshift(a);
4d   signedprint(a); printf("\n");
||
f8   set(a,i);
```

```
84   rightshift(a);
1f   signedprint(a); printf("\n");
||
f8   set(a,i);
88   div(a,b,c,d);
90   signedprint(c);printf("\n");
fe   signedprint(d);printf("\n");
||
7d  }
c3  return 0;
13 }
|| ================ bignumprimes.c ================
|| Tree Labelling − uva 10247
|| Uses "prime factorization" bigints.
||                               Multiplication and
|| division are trivial.  Converting from normal
||                                    int requires
|| factorization so these routines are useful only
||                                     for numbers
|| that are the product of reasonably small ones.
|| Recovering integer requires multiplication of
||                                   small numbers
|| by conventional bignum.
|| I’m sloppy about sizes here:  should probably
||                                 use c++ vectory
|| or something.
|| Works great for combinations stuff:  see "choose
||                                               "
||
79 #include <stdio.h>
5e #include <string.h>
||
40 #define PSZ 500
||
67 int primes[PSZ], np;
43 char seive[100000];
||
d0 struct pp {
28  int p[PSZ];
31 };
||
3f void initprimes(){
ee  int i,j;
c4  for (i=2;np<PSZ;i++){
87   if (!seive[i]) {
47    primes[np++] = i;
8f    for (j=i*i;j<100000;j+=i) seive[j]=1;
a1   }
88  }
21 }
||
bc void ppint(struct pp * r, int x){
8c  int i;
86  memset(r,0,sizeof(struct pp));
3a  for (i=0;x>1;i++) {
93   while (x%primes[i]==0) {
f5    r−>p[i]++;
04    x/=primes[i];
65   }
6e  }
36 }
||
1f void ppmul(struct pp *a, struct pp b){
38  int i;
8a  for (i=0;i<np;i++) a−>p[i] += b.p[i];
39 }
||
54 void ppdiv(struct pp *a, struct pp b){
96  int i;
a7  for (i=0;i<np;i++) a−>p[i] −= b.p[i];
80 }
||
f2 char * ppprint1(struct pp a){
17  static char s[100000], *ss;
36  int t[4000], i,j,k,m;
eb  memset(t,−1,sizeof(t));
4e  t[0] = 1;
9b  for (i=0;i<np;i++) {
6b   if(!a.p[i]) continue;
1e   for (k=0;k<a.p[i];k++){
b1    m = 0;
47    for (j=0;j<4000;j++) {
b5     if (t[j] < 0) {
f3      if (m) t[j] = m%10000;
79      break;
2e     }
d7     m = t[j] * primes[i] + m;
99     t[j] = m %10000;
e5     m /= 10000;
92    }
31   }
f4  }
ee  for (i=4000−1;i>0 && t[i] <= 0;i−−);
9c  sprintf(s,"%d",t[i]);
6d  ss = s + strlen(s);
```

```
cd  for (i−−;i>=0;i−−){
93   sprintf(ss,"%04d",t[i]);
e6   ss += 4;
1a  }
f3  *ss = 0;
5b  printf("%s",s);
6f }
||
||
74 int i,j,K,D,m,n;
||
bf int count(int k){
66  int j, p = 1, s = 0;
0d  for (j=0;j<k;j++) {
e6   s += p;
76   p *= K;
5e  }
74  return s;
ce }
||
45 struct pp choose(int n, int m) {
8f  struct pp p,ni;
45  int i;
8f  ppint(&p,1);
b0  if (n−m < m) m = n−m;
83  for (i=0;i<m;i++) {
18   ppint(&ni,n−i);
86   ppmul(&p,ni);
bf   ppint(&ni,i+1);
37   ppdiv(&p,ni);
04  }
7f  return p;
de }
||
85 struct pp ways(int d){
bc  struct pp p;
bf  int i;
26  ppint(&p,1);
15  if (d == 1) return p;
16  for (i=0;i<K;i++) {
b5   ppmul(&p,ways(d−1));
e9   ppmul(&p,choose(count(d)−1−i*(count(d)−1)/K, (
||                                 count(d)−1)/K));
f2  }
8e  return p;
69 }
||
||
15 struct pp zz,ww;
||
||
94 main(){
10  initprimes();
29  while (2 == scanf("%d%d",&K,&D)) {
68   zz = ways(D+1);
29   ppprint1(zz);
b1   printf("\n");
38  }
2c }
|| ================== binomial.h ==================
|| Computes n choose k, reducing as we
|| go along so as to avoid overflow.
||
79 #include <stdio.h>
||
52 typedef unsigned long long ull;
||
c5 #define FR(i,a,b) for (int i=a;i<b;i++)
||
fa ull gcd( ull a, ull b ) {
83  return b ? gcd(b, a%b) : a;
a1 }
||
61 ull ch(ull n, ull k) {
5f  ull ret = 1, z, i;
e8  if (k > n) return 0;
73  if (n−k < k) k = n−k;
c1  FR(i,1,k+1) {
29   z = gcd(ret, i);
0c   ret /= z;
43   ret *= (n−i+1)/(i/z);
77  }
cd  return ret;
73 }
|| ================== burnside.c ==================
|| distinct necklaces w.r.t. rotational symmetry
|| also both rotation and reflectional symmetry.
|| N: number of beads t: number of types
|| Burnside’s Lemma: the number of arrangements
|| (orbits) = avg. # of fixed points over the
|| elements of G.  e.g. if N=6 then for rotation:
||  t^1 = rot by 1, 5 − all beads must be the same
||  t^2 = rot by 2, 4 − even/odd can be different
||  t^3 = rot by 3 − choose color for 3 sets
||  t^6 = rotation by 6 (0) − etc.
|| The # of necklaces with 6 beads is
||  ( 2*t + 2*t^2 + t^3 + t^6 ) / 6.
|| Solves UVA 10294.
|| Intermediate results grow to 4*(t^N)
||
79 #include <stdio.h>
8c #include <stdlib.h>
||
34 typedef long long i64;
||
93 int gcd( int a, int b )
df {
84  if( b == 0 ) return a;
87  return gcd( b, a%b );
c9 }
||
21 int main()
d2 {
2e  int N, t, sum, i, j;
||
7f  while( scanf("%d%d",&N,&t) == 2 )
58  {
cb   i64 z=t, tot=0;
||
32   for( i=1; i <= N; i++, z*=t ) if( N%i == 0 ) {
18    for( j=1, sum=0; j <= N; j++ ) sum += (gcd(j,
||                                         N) == i);
2b    tot += sum*z;
d4   }
f6   printf( "%lld", tot/N );
||
b4   if( N%2 == 0 ) {
73    for( sum=N/2, i=1; i <= N/2+1; i++ ) sum *= t
||                                                 ;
5d    tot += sum;
5c    for( sum=N/2, i=1; i <= N/2; i++ ) sum *= t;
20    tot += sum;
65   } else {
50    for( sum=N, i=1; i <= N/2+1; i++ ) sum *= t;
43    tot += sum;
07   }
be   printf( " %lld\n", tot/(2*N) );
d0  }
f3 }
|| ================= calculus.cc ==================
79 #include <stdio.h>
8c #include <stdlib.h>
20 #include <string.h>
22 #include <math.h>
ac #include <utility>
7e using namespace std;
||
e2 #define ld long double
01 #define FR(i,a,b) for (int i=a;i<b;i++)
8b #define FOR(i,n) FR(i,0,n);
3d #define ull unsigned long long
||
|| integrate using simpson's rule
|| error <= (b-a)/(1111^4*180)
|| times the max of the 4th derivative.
1b ld integrate(ld(*f)(ld), ld a, ld b) {
71  ld ans = f(b) - f(a);
b6  ld step = (b-a)/2222;
8a  FOR(i,1111) {
40   ans += 2*f(a); a += step;
8c   ans += 4*f(a); a += step;
29  }
4d  return ans*step/3;
5f }
||
|| integrate using adaptive simpson's rule
|| tol specifies the maximum error
|| use multiple intervals for non-monotonic
||                                         functions
2e ld S(ld(*f)(ld), ld a, ld b) {
64  return (b-a) * (f(a) + 4*f((a+b)/2) + f(b))/6;
96 }
18 ld adaptint(ld(*f)(ld), ld a, ld b, ld tol) {
25  ld c = (a+b)/2, x = S(f,a,b), y = S(f,a,c), z =
||                                         S(f,c,b);
||
77  if (fabs(y+z-x) < tol) return y+z+(y+z-x)/15;
85  else return adaptint(f,a,c,tol/2) + adaptint(f,
||                                       c,b,tol/2);
71 }
||
|| stably solve quadratic equation
83 pair<ld,ld> qroot(ld a, ld b, ld c) {
c6  int sgn = b < 0 ? -1 : 1;
f1  ld d = -b + sgn * sqrt(b*b-4*a*c);
00  return make_pair(d/a/2, 2*c/d);
5d }
||
||
fa ull gcd(ull a, ull b) {
83  return b ? gcd(b, a%b) : a;
a1 }
||
|| n choose k
61 ull ch(ull n, ull k) {
5f  ull ret = 1, z, i;
e8  if (k > n) return 0;
73  if (n-k < k) k = n-k;
c1  FR(i,1,k+1) {
29   z = gcd(ret, i);
0c   ret /= z;
43   ret *= (n-i+1)/(i/z);
77  }
cd  return ret;
73 }
||
0b int main(){}
|| ================= cfparse.c ===================
|| Solution to Northeast Europe 2001 Problem B:
||                                          Brackets
||
|| Approach:  general context-free parse with cost;
||                                 a few corners cut
||  because the grammar is simple
||
|| The grammar:
||
||  B -> <null>       { cost 0 }
||  B -> B B          { cost 0 }
||  B -> ( B )        { cost 0 }
||  B -> [ B ]        { cost 0 }
||  B -> (            { cost 1 }     // treat as "()
||                                                 "
||  B -> )            { cost 1 }     // treat as "()
||                                                 "
||  B -> [            { cost 1 }     // treat as "[]
||                                                 "
||  B -> ]            { cost 1 }     // treat as "[]
||                                                 "
||
|| The parsing algorithm:
||
||  For all substrings, in order by length, find
||                             right-hand-sides that
||  match, looking for improved cost.  (If there
||                                     were several
||  nonterminal symbols, there would be a cost
||                                   array for each)
||
|| The parse tree:
||
||  There are an infinite number of parse trees.
||                                  Any (finite) one
||  will do.  We do this by repeating the right-
||                             hand-side matching to
||  find one that yielded the best cost.  (Avoid
||                           non-productive matching
||  that would result in infinite derivation)
||
|| The output:
||
||  The translation is really an attribute grammar
||                                        evaluation
||  for the attribute "output" that echoes the
||                                     input, adding
||  extra pairing brackets to those matche by the
||                                     last 4 rules.
||
79 #include <stdio.h>
||
f4 char buf[200];
b6 int Bcost[200][200];
||
94 main(){
ee  int i,j,k,len,n;
48  n = strlen(gets(buf));
92  for (i=0;i<n;i++) Bcost[i][0] = 0;  // B -> <
||                                             null>
95  for (i=0;i<n;i++) Bcost[i][1] = 1;  // B -> '['
||                                 | ']' | '(' | ')'
86  for (i=0;i<n;i++) for (j=2;j<=n;j++) Bcost[i][j
||                                          ] = 999;
08  for (len=2;len<=n;len++) {
a7   for (i=0;i<=n-len;i++) {
2c    for (j=0;j<=len;j++) {   // B -> BB
ea     k = Bcost[i][j] + Bcost[i+j][len-j];
d4     if (k < Bcost[i][len]) {
33      Bcost[i][len] = k;
cc     }
83    }
9b    if (buf[i] == '[' && buf[i+len-1] == ']'   //
||                                    B -> '[' B ']'
4c    ||  buf[i] == '(' && buf[i+len-1] == ')'){ //
||                                    B -> '(' B ')'
4c     if (Bcost[i+1][len-2] < Bcost[i][len]) {
c5      Bcost[i][len] = Bcost[i+1][len-2];
3b     }
14    }
b5   }
48  }
ed  recover_parse(0,n);
16  return 0;
95 }
||
51 recover_parse(int pos, int len){
4b  int j;
27  if (len == 0) return 0;
e6  if (len == 1) {
60   if (buf[pos] == '[' || buf[pos] == ']') printf
||                                             ("[]");
4d   if (buf[pos] == '(' || buf[pos] == ')') printf
||                                             ("()");
86   return 0;
91  }
||
da  for (j=1;j<len;j++) {     // avoid infinite
||                             left/right derivation
3b   if (Bcost[pos][len] == Bcost[pos][j] + Bcost[
||                                 pos+j][len-j]) {
67    recover_parse(pos,j);
97    recover_parse(pos+j,len-j);
e2    return 0;
42   }
f7  }
2f  printf("%c",buf[pos]);        // must be
||                                         bracketed
55  recover_parse(pos+1, len-2);
fa  printf("%c",buf[pos+len-1]);
b2 }
|| ================= chinese.cc ==================
|| Chinese remainders
||
|| given m1...mn, k1...kn, find x s.t.
|| x == k1 (mod m1)
|| x == k2 (mod m2)
||  ...
|| x == kn (mod mn)
|| -----------------------------------
|| modinverse(x,m) = x^-1 (mod m)
|| -----------------------------------
|| dioph(a,b,c) is an x such that
|| ax + by = c for integer x, y
||
79 #include <stdio.h>
||
df int gcd(int a, int b) {
84  if (b == 0) return a;
87  return gcd(b,a%b);
c9 }
||
96 int solve(int k1,int m1,int k2,int m2) {
6c  int a,b,c = k2-k1;
b1  if (c%gcd(m1,m2)) return -1;
0b  for(a=b=0;a - b != c;) {
ee   a=(c+b+m1-1)/m1*m1;
a5   b=(a-c+m2-1)/m2*m2;
c8  }
f2  return (a+k1);
02 }
||
d2 int main() {
a0  int k,m,k1,m1;
||
5a  k = 0; m = 1;
c7  while (2 == scanf("%d%d",&k1,&m1)) {
46   k = solve(k,m,k1,m1);
d3   if (k == -1) {
2b    printf("No solution.\n");return 0;
cc   }
01   m = m/gcd(m,m1)*m1;
45  }
ef  printf("%d mod %d\n",k,m);
fa  return 0;
82 }
||
ef int modinverse(int x, int m) {
86  int r = solve(0,x,1,m);
82  if (r == -1) return -1;
f0  return r/x;
ba }
||
27 int dioph(int a, int b, int c) {
66  int k = gcd(a,b);
cc  if (c%k) return -1;
ec  return c/k * modinverse(a/k, b/k);
1a }
|| ================== chull.cc ==================
|| convex hull, returns MINIMAL hull
|| needs more testing
|| first half are all overlaps with geo
||
d2 #include <algorithm>
7e #include <complex>
3d #include <cmath>
2a #include <vector>
e1 using namespace std;
||
7c typedef long double T;
c7 typedef complex<T> point;
74 typedef vector<point> poly;
||
ca #define X real
5d #define Y imag
||
a6 poly p;
||
8f T cross(point p,point q){
0f  return Y(conj(p)*q);
45 }
||
e3 #define SZ(c) (int((c).size()))
36 #define BEND(c) c.begin(), c.end()
fe #define MP make_pair
ab #define PB push_back
63 #define FR(i, a, b) for(int i=(a); i<(b); i++)
e8 #define FOR(i, n) FR(i, 0, n)
07 #define RF(i, a, b) for(int i=(b)-1; i>=(a); i--
||                                                 )
38 #define ROF(i, n) RF(i, 0, n)
3f #define A first
df #define B second
||
07 #define EPS 1e-9
||
e2 int n;
04 poly P,hull;
||
|| hull code starts here
05 vector<pair<pair<T,T>,int> > order;
||
80 T crossp3(point p,point q,point r){
8e  return cross(p-r,q-r);
c9 }
||
82 void convex_hull(const poly &P,poly &ans){
4e  ans.clear();
a3  order.clear();
64  FOR(i,SZ(P)) order.PB(MP(MP(X(P[i]),Y(P[i])),i)
||                                                );
45  sort(BEND(order));
c7  FOR(i,SZ(P)){
8a   while((SZ(ans)>1)&&(crossp3(P[order[i].B],ans[
||                   SZ(ans)-1],ans[SZ(ans)-2])>=0))
d0    ans.pop_back();
de   ans.PB(P[order[i].B]);
4b  }
c5  poly up;
07  FOR(i,SZ(P)){
29   while((SZ(up)>1)&&(crossp3(P[order[i].B],up[SZ
||                        (up)-1],up[SZ(up)-2])<=0))
93    up.pop_back();
ce   up.PB(P[order[i].B]);
db  }
64  RF(i,1,SZ(up)-1)
7e   ans.PB(up[i]);
56 }
||
|| filler
d2 int main(){
0b }
|| ================ circleHull.cpp ================
|| Given a collection of non-intersecting non-
||                                    nested circles
|| Calculate their convex hull
17 #include <vector>
93 #include <iostream>
d4 #include <complex>
5e using namespace std;
1b #define fu(i,n) for(int i=0; i<n; i++)
cc #define pb push_back
5a typedef complex<double> point;
97 typedef pair<double,int> pear;
||
e3 int N; // Number of circs
71 double R[2000]; // radii
ed point  O[2000]; // centres
2f pear start,cur;
3e double eps=1e-8,ans;
fa vector<point> Hull;
||
db double area(const vector<point>& p) {
02  double ret=0;
de  fu(i,p.size()) ret+=(p[i]*conj(p[(i+1)%p.size()
||                                       ])).imag();
bd  return abs(ret)/2;
18 }
||
70 double tang(int i, int j, double k) {
9e  double ret=arg(O[j]−O[i])+k*acos((R[i]−R[j])/
||                                 abs(O[j]−O[i]));
f1  if(ret<=−M_PI) ret+=2*M_PI;
25  if(ret>M_PI) ret−=2*M_PI;
19  return ret;
a5 }
||
2e void adv(bool upd=true) {
33  double ang=cur.first; int w=cur.second;
b1  pear b;
66  cur.second=−1;
d9  double best=1e20;
22  double cu;
50  fu(k,2) fu(i,N) if(i!=w) {
4a   cu=tang(w,i,2*k−1); if(cu<=ang+eps) cu+=2*M_PI
||                                                ;
0f   if(cu<best−eps || abs(cu−best)<eps &&
1d   (cur.second==−1 || abs(O[i]−O[w])>abs(O[cur.
||                                 second]−O[w]))) {
55    best=cu;
6d    cur=pear(tang(i,w,1−2*k),i);
ee   }
12  }
||
    // You have the hull, now decide what to keep
||                                         track of
    // w is the current circle, with entry and exit
||                                           angles
    // ang and best
    // For example, let’s calculate the area (Hull
||                                     is the hull)
08  if(upd) {
71   ans+=R[w]*R[w]*(best−ang)/2;
b4   Hull.pb(O[w]+polar(R[w],ang));
d4   Hull.pb(O[w]);
48   Hull.pb(O[w]+polar(R[w],best));
5e   cout << "following circle " << w << " for ";
75   cout << (best−ang)*180/M_PI << " degrees" <<
||                                            endl;
16  }
25 }
||
71 double hullArea() {
86  Hull.clear(); ans=0;
||
05  point pol=polar(1.0,34543.2343); // rotate by
||                                     random angle
82  fu(i,N) O[i]*=pol;
27  int w=0;
a7  fu(i,N)
7b   if(O[i].real()+R[i] > O[w].real()+R[w]) w=i;
24  if(N==1) return M_PI*R[0]*R[0];
aa  cur = pear(0.,w);
8a  adv(false);
83  start=cur;
||
78  do {
b1   adv();
ad  } while(cur.second != start.second || abs(cur.
||                          first−start.first)>1e−8);
d5  return area(Hull)+ans;
83 }
||
c4 int main(void) {
56  while((cin >> N)&&N) {
fd   double x,y;
bf   fu(i,N) {
bb    scanf("%lf%lf%lf",&x,&y,&R[i]);
22    O[i]=point(x,y);
2c   }
69   double ret=hullArea();
82   cout << "our hull has area: " << ret << endl;
41  }
ff }
|| =============== closest−pair.cc ================
|| Closest pair of points in O(n log n). Can be
|| modified to use doubles. "dist" stores the
|| square of the closest distance; "mark[i]"
|| == cookie if the i−th point is within the
|| closest pair distance of some other point.
47 #include <iostream>
50 #include <vector>
bd #include <valarray>
a9 #include <algorithm>
30 using namespace std;
1d typedef long long T;
06 typedef valarray<T> P;
88 typedef vector<P> VP;
26 #define fu(i,n) for( int i = 0; i < (n); i++ )
||
8d bool ltx( const P& a, const P& b )
```

```
cb { return a[0]<b[0] || a[0]==b[0] && a[1]<b[1]; }
bd bool lty( const P& a, const P& b )
6f { return a[1]<b[1] || a[1]==b[1] && a[0]<b[0]; }
||
96 T dist=1ll<<62; int cookie=1, mark[100000];
||
d3 void go( VP& X, VP& Y ) {
31  VP XL, YL, XR, YR, S; int m = X.size()/2;
88  fu(i,X.size()) {
08   (ltx(X[i],X[m])?XL:XR).push_back(X[i]);
de   (ltx(Y[i],X[m])?YL:YR).push_back(Y[i]);
83  }
4b  if( XR.size() > 1 ) { go(XL, YL); go(XR, YR); }
cb  fu(i,Y.size()) {
c6   T z = X[m][0]−Y[i][0];
0f   if( z*z <= dist ) S.push_back(Y[i]);
65  }
a8  fu(i,S.size()) fu(j,7) if(i+1+j<S.size()) {
85   P p = S[i+1+j]−S[i];
a3   T z = p[0]*p[0] + p[1]*p[1];
c3   if( z < dist ) { ++cookie; dist = z; }
02   if( z <= dist ) {
62    mark[S[i][2]] = mark[S[i+1+j][2]] = cookie;
46   }
cf  }
fb }
||
d2 int main() {
19  VP X, Y; T a, b; int c = 0; P p(3);
d0  while( scanf("%lld %lld",&a,&b)==2 ) {
b0   if( a+b < 0 ) break;
70   p[0]=a; p[1]=b; p[2]=++c;
34   X.push_back(p);
ab   Y.push_back(p);
40  }
00  sort(X.begin(), X.end(), ltx);
58  sort(Y.begin(), Y.end(), lty);
2e  go(X, Y);
e7  fu(i, 99999) if( mark[i] == cookie )
17   printf("%d\n", i);
b5 }
||
|| ================== color.cc ===================
|| 5−coloring of a planar graph
||
|| for general coloring see "colorit.h"
||
|| for discussion of planar graph
|| colouring, and the LEDA code for
|| 5−coloring, see "colournotes"
||
79 #include <stdio.h>
5e #include <string.h>
||
c5 #define FR(i,a,b) for(int i=a;i<b;i++)
9a #define FOR(i,n) FR(i,0,n)
||
30 char name[500][32];
16 int nn;
71 char c[500][500];
cc int col[500];
||
b3 char *s[] = {"Blue","Green","Pink",
77  "Red","Yellow"};
||
9d int look(char *x) {
32  int i;
c5  for (i=0;i<nn&&strcmp(name[i],x);i++);
52  strcpy(name[i],x);
b6  if (i == nn) nn++;
47  return i;
0f }
||
8c void colour(int w) {
d5  int i,j,k;
47  int e[5];
25  int taken[5];
84  int ne = 0;
a6  if (w >= nn) return;
||
aa  for (i=w+1;i<nn && ne<5;i++)
51   if (c[i][w]) e[ne++] = i;
||
81  if (ne < 5) colour(w+1);
aa  else FOR(i,5) FR(j,i+1,5)
08   if (!c[e[i]][e[j]]) {
e2    char s1[500],s2[500];
d8    for (k=0;k<nn;k++) {
98     s1[k] = c[k][e[i]];
5c     s2[k] = c[k][e[j]];
25     c[k][e[j]] |= c[k][e[i]];
75     c[k][e[i]] = 0;
3e    }
8b    colour(w);
82    for (k=0;k<nn;k++) {
```

```
85     c[k][e[i]] = s1[k];
25     c[k][e[j]] = s2[k];
d4    }
c8    col[e[i]] = col[e[j]];
cb   }
||
70  FOR(i,5) taken[i] = 0;
4b  FR(i,w+1,nn) if (c[w][i])
85   taken[col[i]] = 1;
d9  for (i=0;taken[i];i++);
35  col[w] = i;
4c }
||
d2 int main() {
38  char buf[1024];
11  char *p;
||
7e  for(;;) {
bf   nn = 0;
06   FOR(i,500) FOR(j,500) c[i][j] = 0;
ff   while(gets(buf) && strcmp(buf,"#")){
0a    if (!strcmp(buf,"END")) return 0;
1c    int i = look(strtok(buf," ")),j;
0c    while (p = strtok(0," "))
84     j=look(p), c[i][j]=c[j][i]=1;
b9   }
6a   colour(0);
34   FOR(i,nn) printf("%s %s\n",
fc    name[i],s[col[i]]);
a3   FOR(i,nn) FOR(j,nn) c[i][j]=0;
a0  }
d8  return 0;
fb }
|| ================ colorit.cpp =================
|| Sample use of "colorit.h" general graph coloring
||                                                .
|| Problem:  "Inviting Politicians"
79 #include <stdio.h>
5e #include <string.h>
6e #include <stdlib.h>
0e #include <algorithm>
d5 using namespace std;
d2 #include "coloritheap.h"
||
9b int i,j,k,m,n,T,N,M;
||
34 char name[300][12];
5c int nn,t;
56 char x[12],y[12];
ce int ix,iy;
||
ab int color[300];
||
2b typedef int (*qsortf)(const void*,const void*);
||
d2 int main() {
d8  scanf("%d",&T);
64  for (t=1;t<=T;t++) {
cc   scanf("%d%d",&N,&M);
cc   if (t != 1) printf("\n");
2c   printf("Case #%d\n",t);
36   fflush(stdout);
1c   reset(N);
75   for (i=0;i<N;i++) scanf(" %s",name[i]);
4c   qsort(name,N,12,(qsortf)strcmp);
ec   for (i=0;i<M;i++) {
ff    scanf(" %s %s", x, y);
b3    ix = ((char*)bsearch(x,name,N,12,(qsortf)
||                                          strcmp)
21     − (char *)name)/12;
ec    iy = ((char*)bsearch(y,name,N,12,(qsortf)
||                                          strcmp)
eb     − (char *)name)/12;
95    edge(ix,iy);
b7    edge(iy,ix);
88   }
49   chroma = colorit(1,4,color);
94   if (chroma > 4) {
7d    printf("No 4−coloring\n");
fe    continue;
0a   }
18   for (i=1;i<=chroma;i++) {
80    for (k=j=0;j<nv;j++) if (color[j] == i) k++;
b4    printf("%d\n",k);
fb    for (j=0;j<nv;j++) if (color[j] == i)
0c     printf("%s%c",name[j],−−k?’ ’:’\n’);
3d   }
54  }
28  return 0;
5c }
|| ================= colorit.h ==================
|| colorit.h  −−  general graph coloring
||
|| up to MAXV of a few hundred,
|| depending on graph
```

```
|| reset(n) −− called before building
||  graph.  vertices must be in range
||  0..n−1
|| edge(v1,v2) −− directed edge from v1
||  to v2.  make sure you call
||  edge(v2,v1) as well.
|| colorit(int minc,int maxc,int c[])
|| least coloring between minc and maxc
|| c[v] is color of vertex v (1..chroma)
|| returns chroma > maxc and color[*] =0
|| if no coloring
||
|| Algorithm notes.
|| linear search finds "most difficult"
|| vertices:
|| it has neighbours colored different
||  colors;
|| it has many uncolored neighbours.
||
|| There is a heap version(coloritheap.h)
|| in many cases runs only marginally
|| faster.
||
|| colorize() find coloring w specific
|| chromaticity
|| colorit() uses iterative deepening on
|| chroma Don’t even think about having
|| colorize() find the best in one
|| shot −−− it will waste time
||
60 #define MAXV 500
8a #define MAXC MAXV
||
50 struct vv {
ce  short nadj, ncol, maxcol, color,
05   adj[MAXV], use[MAXC];
c7 } v[MAXV];
||
7a int cmp (vv &a, vv &b) {
e9  if (a.ncol != b.ncol)
18   return a.ncol − b.ncol;
2b  return a.nadj − b.nadj;
64 }
||
31 int nv,chroma, *bestcolor;
||
0e void reset(int n) {
68  int i,j;
eb  nv = n;
bc  FOR(i,n) {
d0   v[i].nadj = v[i].ncol = v[i].color
25    = v[i].maxcol = 0;
56   FOR(j,n<?MAXC) v[i].use[j] = 0;
5f  }
ef }
||
|| user MUST do edge(i,j) and edge(j,i)
ca void edge(int i, int j) {
59  v[i].adj[v[i].nadj++] = j;
77 }
||
e4 int best(void) {
e5  int i,r=0;
d2  for (r=0;r<nv && v[r].color;r++) {}
54  FR(i,1,nv)
2b   if (!v[i].color && cmp(v[i],v[r])>0)
1e    r = i;
31  return r;
8c }
||
72 int colorize(int r, int maxc) {
9f  int i,j,k,t;
ca  if (maxc > chroma) return 0;
52  if (r >= nv) {
52   FOR(i,nv) bestcolor[i] = v[i].color;
3a   return 1;
2c  }
bc  t = best();
85  for (i=0;i<=maxc;i++) {
d4   if (v[t].use[i]) continue;
78   v[t].color = i+1;
48   for (j=0;j<v[t].nadj;j++) {
92    k = v[t].adj[j];
eb    if (!v[k].use[i]++) v[k].ncol++;
52   }
ac   if(colorize(r+1,i<maxc?maxc:maxc+1))
14    return 1;
b7   v[t].color = 0;
36   for (j=0;j<v[t].nadj;j++) {
4b    k = v[t].adj[j];
84    if (!−−v[k].use[i]) v[k].ncol−−;
31   }
7c  }
4a  return 0;
1f }
```

```
||
c5 int colorit(int minc, int maxc,
5f  int color[]) {
61  int i;
e7  for (i=0;i<nv;i++) color[i] = 0;
f8  bestcolor = color;
20  for(c=minc; c<=maxc && !colorize(0,0);
c8   c++);
a3  return c;
78 }
|| ================ coloritheap.h =================
|| general graph coloring. see colorit.h
|| prio q for "most difficult" vertices
|| push, pop, empty, adjust are heap ops
d2 #include <algorithm>
f7 #define MAXV 500
63 #define MAXC MAXV
b4 #define FOR(i,n) for (int i=0;i<n;i++)
||
50 struct vv {
ce  short nadj, ncol, maxcol, color,
05   adj[MAXV], use[MAXC];
c7 } v[MAXV];
||
94 int cmp(const vv &a, const vv &b) {
52  return a.ncol−b.ncol ?: a.nadj−b.nadj;}
c0 bool operator<(const vv&a,const vv&b)
5e  { return cmp(a,b) < 0; }
||
31 int nv,chroma, *bestcolor;
||
0e void reset(int n) {
68  int i,j;
eb  nv = n;
bc  FOR(i,n) {
d0   v[i].nadj = v[i].ncol = v[i].color
25    = v[i].maxcol = 0;
7a   FOR(j,MAXC) v[i].use[j]=0;
d6  }
c3 }
||
13 int h[MAXV], z[MAXV];
9a int nh;
||
eb void push(int i) {
17  h[++nh] = i;
34  z[i] = nh;
37  for (int j=nh;j>1 &&
4f   v[h[j/2]]<v[h[j]];j/=2)
9d   swap(h[j],h[j/2]),
54   swap(z[h[j]],z[h[j/2]]);
c7 }
||
41 int pop() {
34  h[1] = h[nh−−]; z[h[1]] = 1;
93  for (int j=2; j<=nh; j*=2) {
1d   if (j<=nh && v[h[j]]<v[h[j+1]])j++;
24   swap(h[j],h[j/2]);
80   swap(z[h[j]],z[h[j/2]]);
09  }
75 }
||
5d int empty() {
d3  return nh == 0;
0b }
||
fd void adjust(int i) {
83  int j, k = z[i];
9e  for (j=k*2;j<=nh;j*=2) {
77   if (j<nh && v[h[j]]<v[h[j+1]]) j++;
cb   if (!(v[h[j/2]]<v[h[j]])) break;
93   swap(h[j],h[j/2]);
f5   swap(z[h[j]],z[h[j/2]]);
b2  }
23  for (j=k;j>1 && v[h[j/2]]<v[h[j]];j/=2)
b9   swap(h[j],h[j/2]),
4f   swap(z[h[j]],z[h[j/2]]);
e7 }
||
|| user MUST do edge(i,j) and edge(j,i)
ca void edge(int i, int j) {
59  v[i].adj[v[i].nadj++] = j;
77 }
||
db int best() {
bc  int r = h[1]; pop(); return r;
b0 }
||
a4 int colorize(int maxcol) {
4a  int i,j,k,t;
20  if (maxcol > chroma) return 0;
ff  if (empty()) {
e7   FOR(i,nv) bestcolor[i] = v[i].color;
54   return 1;
87  }
```

```
e7  t = best();
85  FOR(i,maxcol+1) {
d6   if (v[t].use[i]) continue;
38   v[t].color = i+1;
98   FOR(j,v[t].nadj) {
25    k = v[t].adj[j];
8b    if (!v[k].use[i]++)
c1     v[k].ncol++, adjust(k);
63   }
47   if (colorize(i < maxcol ? maxcol :
d9    maxcol + 1)) return 1;
d1   v[t].color = 0;
17   FOR(j,v[t].nadj) {
03    k = v[t].adj[j];
a4    if (!−−v[k].use[i])
1f     v[k].ncol−−, adjust(k);
45   }
9f  }
19  push(t);
b1  return 0;
30 }
||
75 int colorit(int minc, int maxc, int*c) {
6d  nh = 0;
e2  FOR(i,nv) push(i), c[i] = 0;
8c  bestcolor = c;
16  while(minc<=maxc&&!colorize(0)) minc++;
a2  return minc;
88 }
|| ==================== crc.c =====================
||  CRC CHECKSUM (crc.c)
||  −−−−−−−−−−−−
||
|| − ignores whitespace and //−style comments
|| − restarts the count on any blank or comment−
||                                     only line
||
79 #include <stdio.h>
9d #include <ctype.h>
||
27 #define sh(a) (a<<11)^(a>>5)
db int main() {
1a  unsigned short crc,crcf=0;
22  unsigned char b[1000],*a;
36  while(gets((char *)b)) {
e2   crc=0;
be   for(a=b;*a;a++)
b1    if(*a==’/’ && a[1]==’/’) break;
68    else if(!isspace(*a))
e6     crc = sh(crc)^*a, crcf = sh(crcf);
ba   crcf = crc ? crc^crcf : 0;
e6   if(crcf)
23    printf("%04x  %s\n",crcf,b);
1e   else
12    printf("      %s\n",b);
cd  }
db }
|| =================== dates.cc ===================
|| month = 1,2,3,...,12
|| day = 1,2,3,...,31
|| year:  2BC,1BC,AD1,AD2=−2,−1,1,2.
|| Note that there is no year 0
|| weekday: Sunday=0, Monday=1, ...
||
|| The earliest year you can use is
|| 280000 BC.  Replace occurences of
|| 280000 in the code to go earlier. The
|| replacement must be a multiple of
|| 2800.
||
e2 #include <cstdio>
d3 using namespace std;
||
b7 int dayOfWeek(int day, int month,
df  int year) {
35  int a = (14−month)/12;
88  int y = year+280000−a;
fe  int m=month+12*a−2;
51  int d=(year>0?0:2);
86  return // gregorian
f5   (day+y+y/4−y/100+y/400+31*m/12+d)%7;
    // julian
    //return (5+day+y+y/4+31*m/12+d)%7;
7d }
||
|| Number of days since some mystery
|| date. Use for date arithmetic,
|| differencing dates
03 int countDays(int day, int month,
7c  int year) {
72  int a=(14−month)/12;
06  int y=year+280000−a;
7b  int m=month+12*a−3;
34  int d=(year>0?366:0);
11  return // gregorian
```

```
a3   day−1 + (153*m+2)/5 + 365*y
c2   + y/4 − y/100 + y/400 − d;
    //return // julian
    //  day−1+(153*m+2)/5+365*y+y/4−d;
7d }
||
e1 struct date { int d,m,y; };
||
|| Reverses the above method
61 date getDate(int a) {
e0  if(a>=countDays(1,1,1)) a+=366;
    //gregorian
63  int b =(4*a+3)/146097;
d6  int c = a−146097*b/4;
    //julian
    //int b=0, c=a;
||
93  int d=(4*c+3)/1461;
c6  int e=c−1461*d/4;
a2  int m=(5*e+2)/153;
||
97  date D;
a8  D.d=e−(153*m+2)/5+1;
fe  D.m=m+3−12*(m/10);
d4  D.y=100*b+d−280000+m/10;
60  return D;
5e }
||
81 date easterGregorian(int year) {
59  int g = year%19;
ab  int c=year/100;
a3  int h=(c−c/4−(8*c+13)/25+19*g+15)%30;
14  int i=h−h/28*(1−29/(h+1)*(21−g)/11);
c4  int j=(year+year/4+i+2−c+c/4)%7;
57  int l=i−j;
||
96  date d;
09  d.m=3+(l+40)/44;
3c  d.d=l+28−31*(d.m/4);
||
c1  return d;
fb }
||
74 date easterJulian(int year) {
a3  int g = year%19;
04  int i = (19*g+15)%30;
4d  int j = (year+year/4+i)%7;
1d  int l=i−j;
||
96  date d;
09  d.m=3+(l+40)/44;
3c  d.d=l+28−31*(d.m/4);
||
c1  return d;
fb }
||
c4 int main(void) {
27  for(int i = countDays(1,1,2000);
d2   i < countDays(1,1,2005); i++) {
d8   date d = getDate(i);
a8   printf("day:%d month:%d year:%d wee"
38    "kday:%d\n",d.d, d.m, d.y,
46    dayOfWeek(d.d,d.m,d.y));
ef  }
f2  return 0;
82 }
||
|| ================= debruijn.cc ==================
63 using namespace std;
da #include <stdio.h>
97 #include <list>
fc #include <algorithm>
||
f0 #define MAXE (1<<22)
fe #define MAXN (1<<21)
||
7b list<int> p;
f8 typedef list<int>::iterator iter;
||
48 struct edge {
3f  edge() {}
45  edge(int from,int to,bool *hit) : from(from),to
||                                   (to),hit(hit) {}
98  int from, to;
dc  bool *hit;
c6  bool operator<(const edge&other) const
92   {return from < other.from || (from == other.
||                             from && to < other.to);}
bf } e[MAXE];
ab bool hit[MAXE];
63 int nn,ne;
14 int firste[MAXN],deg[MAXN];
||
16 void buildgraph() {
9a  int a;
```

```
a9  memset(firste,0,sizeof(int)*nn);
20  memset(deg,0,sizeof(int)*nn);
04  sort(e,e+ne);
4e  for(a=ne−1; a>=0; a−−)
54   firste[e[a].from] = a;
0b  for(a=0;a<ne;a++)
4b   deg[e[a].from]++, deg[e[a].to]−−;
84  e[ne].from = −1;
89 }
||
4b void search(iter it, int node) {
ff  for(int i=firste[node]; e[i].from==node; i++)
bc   if(!*e[i].hit) {
bf    *e[i].hit = true;
62    iter n=it;
4c    search(p.insert(++n,i),e[i].to);
47   }
57 }
||
30 bool find_cycle() {
dd  for(int a=0;a<nn;a++)
b1   if(deg[a])
31    return false;
a8  p.clear();
46  memset(hit,0,sizeof(int)*ne);
d5  search(p.begin(),nn−1);
f9  return (int)p.size()==ne;
12 }
||
d2 int main() {
6a  int go,GO;
33  scanf("%d",&GO);
98  for(go=0;go<GO;go++) {
78   int a,n,k;
3d   scanf("%d %d",&n,&k);
||
22   nn = 1<<(n−1);
98   ne = 0;
0a   for(a=0;a<nn;a++) {
66    e[ne] = edge(a,(a*2)%nn,&hit[ne]);
7c    ne++;
8f    e[ne] = edge(a,(a*2+1)%nn,&hit[ne]);
c1    ne++;
d3   }
1a   buildgraph();
||
04   if(find_cycle()) {
c2    iter out=p.begin();
60    for(a=0;a<k;a++)
14     out++;
36    for(a=0;a<n;a++)
4e     printf("%i",e[*(out++)].to&1);
02    printf("\n");
ec   } else {
cb    printf("error\n");
f3   }
8a  }
59  return 0;
af }
|| ================== delaunay.c ==================
|| Delaunay Triangulation − prints 3 corners of
|| each triangle in clockwise order.  Arbitrary
|| triangle order and starting corner.
||
79 #include <stdio.h>
7b #include <math.h>
||
4f double x[1004],y[1004];
47 char done[1004][1004];
30 int n;
||
59 int T,X,Y,M;
||
b6 double bestr, bestx, besty;
||
3c void doit(int c,int d);
b7 void dopoint(double x, double y);
||
|| bi − bisector between 2 points
||
20 void bi(double x1,double y1,double x2,double y2,
95  double *a,double *b,double *c) {
98  *a = 2*(x2−x1);
18  *b = 2*(y2−y1);
76  *c = x2*x2 + y2*y2 − x1*x1 − y1*y1;
36 }
||
|| isct − intersection of 2 lines in ax+by=c
|| format. return 0 if undefined
||
a1 int isct(double a, double b, double c,
ef  double aa, double bb, double cc,
bf  double *x, double *y) {
38  double det = a*bb − b*aa;
c9  if (fabs(det) < 1e−10) return 0;
```

```
e1  *x = (-b*cc + c*bb)/det;
f0  *y = (a*cc - c*aa)/det;
ea  return 1;
3a }
||
||
d2 int main() {
36  int i,j,c,d;
3d  double t,h;
||
99  scanf("%d",&T);
90  while (T--) {
5a   scanf("%d%d%d",&X,&Y,&M);
fb  bestr = bestx = besty = -1;
2c  for (n=0;n<M && 2==scanf("%lf%lf",&x[n],&y[n]);
||                             n++) { }  // read in
cf  x[M] = -X; y[M++] = -Y;
d6  x[M] = -X; y[M++] = 2*Y;
1e  x[M] = 2*X; y[M++] = -Y;
aa  x[M] = 2*X; y[M++] = 2*Y;
9a  for (i=0;i<M;i++) for (j=0;j<M;j++) done[i][j]
||                                             = 0;
||
98  dopoint(0,0);
14  dopoint(0,Y);
d4  dopoint(X,Y);
74  dopoint(X,0);
||
||
    // find corner
9a  for (c=i=0;i<M;i++) if (y[i]<y[c] || y[i]==y[c]
||                               && x[i]>x[c]) c=i;
||
af  h = -acos(-1.0);
ad  for (d=i=0;i<M;i++) {  // first edge on convex
||                                             hull
6c   if (x[c] == x[i] && y[c] == y[i]) continue;
62   if ((t=atan2(y[i]-y[c],x[i]-x[c])) > h+.
||                                     000000001 ||
0a    t > h-.000000001 && hypot(x[i]-x[c],y[i]-y[c]
||                                              ) <
21     hypot(x[i]-x[d],y[i]-y[d])) {d = i; h = t;}
84  }
f8  doit(c,d);
||
4b  for (i=0;i<M;i++) for (j=0;j<M;j++) {
e7   double a,b,c,xx,yy;
db   if (!done[i][j]) continue;
ba   bi(x[i],y[i],x[j],y[j],&a,&b,&c);
59   if (isct(a,b,c,0.0l,1.0l,0.0l,&xx,&yy))
||                                  dopoint(xx,yy);
1a   if (isct(a,b,c,0.0l,1.0l,Y,&xx,&yy)) dopoint(
||                                          xx,yy);
7f   if (isct(a,b,c,1.0l,0.0l,0.0l,&xx,&yy))
||                                  dopoint(xx,yy);
86   if (isct(a,b,c,1.0l,0.0l,X,&xx,&yy)) dopoint(
||                                          xx,yy);
49  }
65  printf("The safest point is (%0.1f, %0.1f). %f\
||                           n",bestx,besty,bestr);
96  }
ed  return 0;
4a }
||
01 double centx,centy,r;
bc void circle(double bx, double by,
46  double cx, double cy, double dx, double dy) {
e3  double temp = cx*cx+cy*cy;
63  double bc = (bx*bx + by*by - temp)/2.0;
30  double cd = (temp - dx*dx - dy*dy)/2.0;
e5  double det = 1/((bx-cx)*(cy-dy)-(cx-dx)*(by-cy)
||                         ); // assume noncolinear
99  centx = (bc*(cy-dy)-cd*(by-cy))*det;
23  centy = ((bx-cx)*cd-(cx-dx)*bc)*det;
f7  r = sqrt((centx-bx)*(centx-bx)+(centy-by)*(
||                                      centy-by));
72 }
||
9f void dopoint(double XX, double YY) {
55  int i;
90  double r = 1e99;
78  if (XX < 0 || XX > X || YY < 0 || YY > Y)
||                                          return;
1e  for (i=0;i<M;i++) if (hypot(x[i]-XX,y[i]-YY) <
||                                               r)
7b   r = hypot(x[i]-XX,y[i]-YY);
fc  if (r > bestr) {
d2   bestr = r;
16   bestx = XX;
77   besty = YY;
86  }
b1 }
||
7c void doit(int c, int d) {
86  int i,e;
```

```
d5  double ecos,icos;
||
60  if (done[c][d]++) return;
||
3b  ecos = 999;
c9  for (i=e=0;i<n;i++) {
ac   if (x[i]==x[c]&&y[i]==y[c] || x[i]==x[d]&&y[i]
||                                ==y[d]) continue;
bf   if ((x[c]-x[i])*(y[d]-y[i])-(x[d]-x[i])*(y[c]-
||                            y[i]) >= 0) continue;
31   icos = ((x[c]-x[i])*(x[d]-x[i])+(y[c]-y[i])*(y
||                                       [d]-y[i]))
46    / (hypot(x[c]-x[i],y[c]-y[i]) * hypot(x[d]-x[
||                                  i],y[d]-y[i]));
bc   if (icos < ecos - .000000001 || icos < ecos +
||                                    .000000001 &&
ca    hypot(x[c]-x[i],y[c]-y[i]) < hypot(x[c]-x[e],
||                                    y[c]-y[e])) {
de    e = i;
a6    ecos = icos;
d8   }
f3  }
7f  if (ecos == 999) return;  // c -> d on convex
||                                             hull
||
77  circle(x[c],y[c],x[d],y[d],x[e],y[e]);
||
|| printf("%lf,%lf %lf,%lf %lf,%lf c %lf,%lf r %lf\
||                                              n",
||  x[c],y[c],x[d],y[d],x[e],y[e], centx,centy,r);
||
43  if (r > bestr && centx >= 0 && centy >= 0 &&
||                         centx <= X && centy <= Y)
c1  {
||  bestr = r;
||  bestx = centx;
||  besty = centy;
71   dopoint(centx,centy);
26  }
||
ab  done[d][e] = done[e][c] = 1;
f3  doit(c,e);
e5  doit(e,d);
22 }
|| ================== derek.c ===================
|| Derek Kisman’s Notes - Verbatim from the Word
|| Document, hacked to compile by GVC
||
92 #define maxn 100
77 #define maxnc 100
||
|| ADTs
||
|| Priority Queue (Heap implementation)
||
|| Mike’s efficient and VERY concise Priority
|| Queue implementation. Use I(Elem) to insert,
|| E(Elem) to extract Elems.  N is number of
|| items in the queue. QSIZE is the maximum
|| number of elements in the queue, plus 1
79 #define QSIZE 1000
||
c5 typedef struct { int p; void *v; } Elem;
|| structure you’ll be inserting/extracting
02 Elem Q[QSIZE], T;
94 int N = 0, P;
||
59 #define P(n) Q[n].p
c6 #define S T=Q[P], Q[P]=Q[P/2], Q[P/2]=T
58 #define I(e) for(Q[P=++N]=(e); P/2 && P(P/2)>P(P
||                                     ); S, P=P/2)
c5 #define E(e) for((e)=Q[1], Q[P=1]=Q[N--];\
2a  P=2*P, (P<N && P(P)>P(P+1) && P++), P<=N && P(P
||                                     )<P(P/2); S)
||
|| Number Theory
||
|| Greatest Common Factor
||
|| Uses Euclid’s Algorithm to quickly find the GCF
|| of a and b; a and b can’t be 0
83 int GCF(int a, int b)
ef {
6c  int x;
ef  while (x = a % b) {
84   a = b;
b4   b = x;
70  }
c9  return abs(b);
8b }
||
|| Least Common Multiple
||
|| Uses GCF() to quickly find the LCM of a and b
86 int LCM(int a, int b)
```

```
5f {
a4  return abs((a/GCF(a,b))*b);
58 }
||
||
|| Primality Testing
||
|| Tests a for primality relatively quickly (skips
|| testing by multiples of 2 and 3)
66 int IsPrime(int a)
10 {
7f  int s, i;
57  a = abs(a);
    // Counts 0 & 1 as non-prime
ce  if (a < 4) return (a > 1);
    // Get multiples of 2 & 3 out of the way
7b  if (!(a % 2) || !(a % 3)) return 0;
e1  s = sqrt(a);
c8  for (i = 5; i <= s; i += 6) if (!(a % i) || !(a
||                              % (i+2))) return 0;
fe  return 1;
fa }
||
|| Prime Finding
||
|| TBF: prime[a] = 1 if a is prime, 0 otherwise
23 int prime[maxn+1];
||
|| Fills out the prime array to n using the Sieve
|| of Eratosthenes (note: uses n+1 elements!)
99 void FindPrimes(int n)
87 {
94  int i,j,s=sqrt(n);
38  prime[0] = prime[1] = 0; // 0 & 1 non-prime
dc  for (i = 2; i <= n; i++) prime[i] = 1;
16  for (i = 2; i <= s; i++) if (prime[i])
fb   for (j = i*2; j <= n; j += i) prime[j] = 0;
a2 }
||
||
|| Factoring
||
d6 int nf; // TBF: Number of prime factors of a
70 int f[31]; //TBF: Prime factors of a, ascending
||
|| Factors a number into its prime factors.  Note
|| that at most 31 prime factors can exist for a
|| number smaller than 2^32.  Relatively quick
|| (skips testing by multiples of 2 & 3)
80 void Factor(int a)
87 {
b9  int i, s;
32  nf = 0;
13  if (!a) return;
71  while (!(a % 2)) {f[nf++] = 2; a /= 2;};
30  while (!(a % 3)) {f[nf++] = 3; a /= 3;};
73  s = sqrt(a);
c2  for (i = 5; i <= s; i += 4) {
8b   while (!(a % i)) {f[nf++] = i; s = sqrt(a /= i
||                                             );};
e3   i += 2;
cf   while (!(a % i)) {f[nf++] = i; s = sqrt(a /= i
||                                             );};
d3  }
78  if (a > 1) f[nf++] = a;
16 }
||
||
|| Sorting
||
|| Binary Search - First element >= key
||
|| Finds the first element greater than or equal
|| to k in ascending sorted array ar using a
|| binary search; returns its index.  Note: this
|| still works with duplicate elements
|| Returns n if ar[n-1] is < k.
e7 int BinarySearch(int *ar, int n, int k)
4c {
dc  int b=0, t=n-1, m;
cf  while (b < t) {
79   m = (b+t)/2;
21    if (ar[m] >= k) t = m; else b = m+1;
dc  }
32  if (ar[b] < k) b = n;
a4  return b;
e8 }
||
|| Binary Search - Last element <= key
||
|| Finds the last element less than or equal to k
|| in ascending sorted array ar using a binary
|| search; returns its index.  Note: this still
|| works with duplicate elements
|| Returns -1 if ar[0] is > k.
```

```
f7 #define BinarySearch BinarySearch1 // ignore
||
e7 int BinarySearch(int *ar, int n, int k)
4c {
dc  int b=0, t=n-1, m;
cf  while (b < t) {
4e   m = (b+t+1)/2;
67    if (ar[m] <= k) b = m; else t = m-1;
46  }
f9  if (ar[t] > k) t = -1;
04  return t;
35 }
||
||
|| Functions
||
|| Numerical Minimization
||
|| Finds the value that minimizes the (continuous)
|| function f on the domain [dmin..dmax]. Acts by
|| splitting the domain into sect sections, then
|| estimating location of minimum from there.  Set
|| sect high, like 10000, if you suspect f may
|| have several local minimums (since you want the
|| global minimum). sect must be >= 3 - about
|| O(sect/log sect) calls are made to f. "result"
|| is within dtol of the "true" minimizing value
|| f("result") is within rtol of the minimum value
|| of f Set rtol & dtol to 0.00000001 or less for
|| most purposes
ec double MinimizeFunc( double f(double), double
||                               dmin, double dmax,
db  double dtol, double rtol, int sect )
6d {
ca  int i, curmin;
ae  double rmin, rmax, x, y, delta;
||
7e  for(;;) {
17   curmin = 0;
cf   if (dmax-dmin<dtol) break;
f2   delta = (dmax-dmin)/sect;
c5   rmin = rmax = f(dmin);
83   for (i=1, x=dmin+delta; i<=sect; i++, x+=delta
||                                              ) {
58    y = f(x);
c7    if (y>rmax) rmax=y;
a8    if (y<rmin) {rmin=y; curmin=i;}
28   }
51   if (rmax-rmin<rtol) break;
77   dmax = (curmin==sect)?dmax:(dmin+delta*(curmin
||                                            +1));
da   dmin = (curmin==0)?dmin:(dmin+delta*(curmin-1)
||                                               );
23  }
27  return dmin+delta*curmin;
b4 }
||
||
|| Root Finding
||
|| Finds a root of a (continuous) function f
|| within the domain [dmin..dmax], to within a
|| tolerance of dtol. If no roots exist, still
|| returns a value from [dmin..dmax]; check
|| whether f(result) is near 0 to see if a root
|| was found. If multiple roots exist, may return
|| any within the given domain.
3a double FuncRoot( double f(double), double dmin,
||                                     double dmax,
49 double dtol )
29 {
b7  double mid, x, y, z;
||
02  x = f(dmin);
5a  y = f(dmax);
a7  for(;;) {
9a   mid = (dmin+dmax)/2;
70   if (dmax-dmin<dtol) break;
f7   z = f(mid);
25   if (x*z<=0) {y=z; dmax=mid;} else {x=z; dmin=
||                                            mid;}
34  }
a3  return mid;
00 }
||
||
|| Permutations
||
|| Calls f with all possible permutations of c
|| elements from the n elements in ar. Just set
|| lev, perm, pused to 0 when you call it f is
|| called with an array of the c elements, and c
|| itself (just as a failsafe)
7b void PermIterate( void f(int *,int), int *ar,
||                           int n, int c, int lev,
8c  int *perm, int *pused ) {
b8  int i, j;
||
02  if (lev==c) {f(perm,c); return;}
0f  if (!lev) {
88   perm = (int*)malloc( c*sizeof(int) );
85   pused = (int*)malloc( n*sizeof(int) );
ef   memset(pused, 0, n*sizeof(int));
92  }
fd  for (i=0; i<n; i++) if (!pused[i]) {
db   pused[i]=1;
b7   perm[lev]=ar[i];
|| Replace ar[i] with i+1 if you just
|| want 1..n permuted
56  PermIterate(f, ar, n, c, lev+1, perm, pused);
84  pused[i]=0;
01  }
1b  if (!lev) {free(perm); free(pused);}
8d }
||
|| Combinations
||
|| Calls f with all possible combinations of c
|| elements from the n elements in ar. Just set
|| i, lev, comb to 0 when you call it f is called
|| with an array of the c elements, and c itself
|| (just as a failsafe)
02 void CombIterate( void f(int *,int), int *ar,
||                             int n, int c, int i,
6d int lev, int *comb ) {
bd  if (!lev) {comb = (int*)malloc( c*sizeof(int) )
||                                                ;}
43  if (lev==c) {f(comb,c); return;}
85  for (; i<=n-c+lev; i++) {
76   comb[lev]=ar[i];
|| Replace ar[i] with i+1 if you just
|| want combinations of 1..n
50  CombIterate(f, ar, n, c, i+1, lev+1, comb);
77  }
a1  if (!lev) {free(comb);}
58 }
||
61 main(){} // just a hack - ignore this
|| ================== derek.txt ===================
e6 Some large primes:
9c 9973 10007 19997 20011 39989 40009 49999
6f 50021 99991 100003 199999 200003 399989
97 400009 499979 500009 999983 1000003
b0 1999993 2000003 3999971 4000037 4999999
38 5000011 9999991 10000019 19999999
fa 20000003 39999983 40000003 49999991
da 50000017 99999989 100000007 199999991
b3 200000033 399999959 400000009 499999993
d5 500000003 999999937 1000000007
a3 1999999973 2000000011 10240297597
fc 54193340731 90477650771 115499206703
db 481778715169 1005680009767 5336435463727
94 70139947146967 9876324585966499
e9 112272535095293 801258244502321
42 2753565111483733 2452902601380727
4b 1081818000108l819 98577541197976567
c9 79523954586701659 101210328665281103
|| ================== dfa.cpp ===================
|| Problem A:  Censored!    (NE Europe 2001)
||
|| How many strings in a regular language?
||
|| Solution:
||
||  - build a DFA to recognise the language
||   - Each NFA state is a pair <word,position>
||   - NFA transitions are implicit in the code
||   - Subset construction builds the DFA
||
||  - interpret the DFA counting the number of ways
||                             to reach each state
||
|| Note:
||
||  - The problem calls for an exact integer answer
||                                   .   I just use
||   "double".  Substituting bigint is left as an
||                               exercise to the
||   reader.
||
79 #include <stdio.h>
6e #include <vector>
82 #include <set>
c8 #include <map>
||
63 using namespace std;
||
16 struct nfastate {
04  int word, pos;
3d  nfastate(int w, int p) { word=w; pos=p; }
62 };
||
d9 bool operator==(const nfastate &x, const
||                                 nfastate &y) {
bb  return x.word==y.word && x.pos==y.pos;
68 }
||
0f bool operator<(const nfastate &x, const nfastate
||                                          &y) {
f9  return x.word<y.word || (x.word == y.word && x.
||                                   pos < y.pos);
da }
||
fc typedef set<nfastate> dfastate;
||
7b map<dfastate, int> indfa;
46 int dfastates;
||
7c int N,M,P;
df char a[100], word[10][11];
||
cb vector<dfastate> dfa;
c4 vector< vector<int> > trans;
||
94 main(){
14  int i,j,k;
3a  scanf("%d%d%d",&N,&M,&P);
6a  scanf("%s",a);
91  for (i=0;i<P;i++) {
33   scanf("%s",word[i]);
0c  }
35  dfa.push_back(dfastate());
d5  indfa[dfa[0]] = 1;  // the reject state [empty]
7b  trans.push_back( vector<int>(N) );
||
36  dfa.push_back(dfastate());
65  for (i=0;i<P;i++) dfa[1].insert(nfastate(i,0));
||
05  indfa[dfa[1]] = 1;  // the start state
ab  trans.push_back( vector<int>(N) );
||
d0  for (i=1; i<dfa.size(); i++) {
7e   trans.push_back( vector<int>(N) );
6b   for (j=0;a[j];j++) {
6a    dfastate newd = dfa[1];   // words may start
||                                       anywhere
5e    for (dfastate::iterator d = dfa[i].begin(); d
||                        != dfa[i].end(); *d++){
14     if (word[d->word][d->pos] == a[j]) {
61      if (!word[d->word][d->pos+1]) goto reject;
c1      newd.insert(nfastate(d->word,d->pos+1));
bb     }
08    }
06    if (!indfa[newd]) {
56     indfa[newd] = dfa.size();
27     dfa.push_back(newd);
0c    }
14    trans[i][j] = indfa[newd];
b2    continue;
cb   reject:
09    trans[i][j] = 0;
e5   }
3a  }
||
   // DFA is built;  Now abstract execution for M
||                                           steps
||
7b  {
42   vector<double> count(dfa.size());
55   double total = 0;
b6   count[1] = 1;
b3   for (i=0;i<M;i++){
38    vector<double> newcount(dfa.size());
dc    for (k=0;k<dfa.size();k++){
f0     for (j=0;a[j];j++) {
c8      newcount[trans[k][j]] += count[k];
3b     }
7c    }
0c    count = newcount;
1d   }
c1   for (i=1;i<dfa.size();i++) total += count[i];
||                         // add up accepting states
2a   printf("%lg\n",total);
94  }
d1 }
|| ================== dioph.c ===================
|| Solution of system of linear diophantine
||                                       equations
||
|| Author: Howard Cheng
|| Date:   Nov 25, 2000
|| Reference:
||
|| http://scicomp.ewha.ac.kr/netlib/tomspdf/
||
|| Look at Algorithms 287 (sort of) and 288.
||
|| Given a system of m linear diophantine equations
||                                  in n unknowns,
|| this algorithm finds a particular solution as
||                             well as a basis for
|| the solution space of the homogeneous system, if
||                                they exist.  The
|| system is represented in matrix form as Ax = b
||                               where all entries
|| are integers.
||
|| Function: diophantine_linsolve
||
|| Input:
||
|| A: an m x n matrix specifying the coefficients
||                             of each equation in
||  each row (it is okay to have zero rows, or even
||                                     have A = 0)
|| b: an m-dimensional vector specifying the right-
||                          hand side of the system
|| m: number of equations in the system
|| n: number of unknowns in the system
||
|| Output:
||
|| The function returns the dimension of the
||                             solution space of the
|| homogeneous system Ax = 0 (hom_dim) if it has a
||                                       solution.
|| Otherwise, it returns -1.
||
|| Other results returned in the parameters are:
||
|| xp: an n-dimensional vector giving a particular
||                                        solution
|| hom_basis: an n x n matrix whose first hom_dim
||                             columns form a basis
||  of the solution space of the homogeneous system
||                                          Ax = 0
||
|| All solutions to Ax = b can be obtained by
||                          adding integer multiples
|| of the first hom_dim columns of hom_basis to xp.
||
|| Note:
||
|| The contents of A and b are not changed by this
||                                       function.
||
79 #include <stdio.h>
8c #include <stdlib.h>
||
b2 #define MAX_N 50
1c #define MAX_M 50
||
0b int triangulate(int A[MAX_N+1][MAX_M+MAX_N+1],
||                             int m, int n, int cols)
7b {
6b  int ri, ci, i, j, k, pi, t;
dc  div_t d;
||
84  ri = ci = 0;
54  while (ri < m && ci < cols) {

   // find smallest non-zero pivot
06   pi = -1;
59   for (i = ri; i < m; i++) {
6e    if (A[i][ci] && (pi == -1 || abs(A[i][ci]) <
||                              abs(A[pi][ci]))) {
9f     pi = i;
f9    }
a2   }
fc   if (pi == -1) {
      // the entire column is 0, skip it */
2e    ci++;
5f   } else {
89    k = 0;
8f    for (i = ri; i < m; i++) {
d3     if (i != pi) {
f8      d = div(A[i][ci], A[pi][ci]);
82      if (d.quot) {
54       k++;
c2       for (j = ci; j < n; j++) {
76        A[i][j] -= d.quot*A[pi][j];
56       }
cf      }
d3     }
93    }
2c    if (!k) {
      // swap the row to make it triangular...Alg
||                          287 also switches the
      // sign, probably to preserve the sign of
||                           the minors.  I don’t
      // think this is necessary for our purpose.
d3     for (i = ci; i < n && ri != pi; i++) {
bf      t = A[ri][i];
1c      A[ri][i] = A[pi][i];
af      A[pi][i] = t;
b8     }
13     ri++;
f9     ci++;
22    }
24   }
cc  }
5e  return ri;
a7 }
||
a7 int diophantine_linsolve(int A[MAX_M][MAX_N],
||                          int b[MAX_M], int m, int n,
dd  int xp[MAX_N], int hom_basis[MAX_N][MAX_N])
25 {
32  int mat[MAX_N+1][MAX_M+MAX_N+1];
eb  int i, j, rank, d;
||
   // form the work matrix
0e  for (i = 0; i < m; i++) {
5c   mat[0][i] = -b[i];
9f  }
02  for (i = 0; i < m; i++) {
32   for (j = 0; j < n; j++) {
a5    mat[j+1][i] = A[i][j];
30   }
2c  }
e6  for (i = 0; i < n+1; i++) {
72   for (j = 0; j < n+1; j++) {
aa    mat[i][j+m] = (i == j);
30   }
f4  }
||
   // triangluate the first n+1 x m+1 submatrix
23  rank = triangulate(mat, n+1, m+n+1, m+1);
a3  d = mat[rank-1][m];
||
   // check for no solutions
85  if (d != 1 && d != -1) {
    // no integer solutions
fd   return -1;
2a  }
   // check for inconsistent system
0e  for (i = 0; i < m; i++) {
99   if (mat[rank-1][i]) {
34    return -1;
4c   }
47  }
||
   // there is a solution, copy it to the result
02  for (i = 0; i < n; i++) {
1d   xp[i] = d*mat[rank-1][m+1+i];
a3   for (j = 0; j < n+1-rank; j++) {
da    hom_basis[i][j] = mat[rank+j][m+1+i];
5b   }
ff  }
||
23  return n+1-rank;
94 }
||
e3 int main(void)
c4 {
5d  int A[MAX_M][MAX_N], b[MAX_M], m, n, xp[MAX_N],
||                         hom_basis[MAX_N][MAX_N];
08  int i, j, hom_dim;
||
eb  while (scanf("%d %d", &m, &n) == 2 && m > 0 &&
||                                          n > 0) {
59   for (i = 0; i < m; i++) {
dc    printf("Enter equation %d:\n", i+1);
9c    for (j = 0; j < n; j++) {
2d     scanf("%d", &A[i][j]);
b4    }
f1    scanf("%d", &b[i]);
ba   }
||
54   if ((hom_dim = diophantine_linsolve(A, b, m, n
||                         , xp, hom_basis)) >= 0)
89    {
72    printf("Particular solution:\n");
59    for (i = 0; i < n; i++) {
cb     printf("%d ", xp[i]);
db    }
48    printf("\n");
df    printf("hom_dim = %d\n", hom_dim);
7f    printf("Basis for Ax = 0:\n");
5e    for (j = 0; j < hom_dim; j++) {
98     for (i = 0; i < n; i++) {
17      printf("%d ", hom_basis[i][j]);
55     }
0f     printf("\n");
ad    }
53   } else {
d8    printf("No solution.\n");
3b   }
||
7d  }
||
c3  return 0;
fb }
|| =================== dioph.cc ===================
|| Solve systems of linear diophantine
|| equations. By Howard Cheng, 25nov2000
||
|| Given m linear diophantine equations
|| in n unknowns, find a particular
|| solution and a basis for the solution
|| space of the homogeneous system, if
|| they exist.  The system is
|| represented in matrix form as Ax = b
|| where all entries are integers.
||
|| Function: diosolve
||
|| Input:
|| A: m x n coefficient matrix.  zero
||  rows are okay.
|| b: m−vector right−hand side
|| m: number of equations in the system
|| n: number of unknowns in the system
||
|| Output:
|| Returns dimension of nullspace of A
|| (hom_dim) if solution exists, else −1
|| xp: n−vector; a particular solution
|| hom_basis: n x n matrix whose first
||  hom_dim columns form a basis for
||  the nullspace of A.
||
|| All solutions to Ax=b are xp plus
|| integer multiples of nullspace basis.
|| A and b are not modified.
||
79 #include <stdio.h>
8c #include <stdlib.h>
cb #include <algorithm>
91 using namespace std;
||
c5 #define FR(i,a,b) for (int i=a;i<b;i++)
9a #define FOR(i,n) FR(i,0,n)
||
04 #define MAXN 51 // leave one extra
c6 #define MAXM 51 // leave one extra
||
77 int triang(int A[MAXM][MAXM+MAXN],
ee  int m, int n, int cols) {
1c  int ri=0, ci=0, t;
||
24  while (ri < m && ci < cols) {
fd   int pi = −1;
cd   FR(i,ri,m) if (A[i][ci] && (pi==−1 ||
24    abs(A[i][ci]) < abs(A[pi][ci])))
ad     pi = i;
85   if (pi == −1) ci++;
ca   else {
d6    int k = 0;
76    FR(i,ri,m) if (i != pi) {
d7     int q=div(A[i][ci],A[pi][ci]).quot;
b0     if (q) {
97      k=1;
7e      FR(j,ci,n) A[i][j] −= q*A[pi][j];
9e     }
f9    }
a1    if (!k) {
1a     if (ri−pi) FR(i,ci,n)
09      swap(A[ri][i], A[pi][i]);
55     ri++; ci++;
6f    }
36   }
94  }
9a  return ri;
71 }
||
93 int diosolve(int A[MAXM][MAXN], int
83  b[MAXM], int m, int n, int xp[MAXN],
28  int hom_basis[MAXN][MAXN]) {
e5  int mat[MAXN][MAXM+MAXN], rank, d;
||
9c  FOR(i,m) mat[0][i] = −b[i];
5f  FOR(i,m) FOR(j,n)
03   mat[j+1][i] = A[i][j];
37  FOR(i,n+1) FOR(j,n+1)
20   mat[i][j+m] = (i == j);
a1  rank = triang(mat, n+1, m+n+1, m+1);
cf  d = mat[rank−1][m];
66  if (abs(d) != 1) return −1; // no soln
2c  FOR(i,m) if (mat[rank−1][i]) return −1;
```

```
81  FOR(i,n) {
70   xp[i] = d*mat[rank−1][m+1+i];
57   FOR(j,n+1−rank)
33    hom_basis[i][j] = mat[rank+j][m+1+i];
74  }
b0  return n+1−rank;
e0 }
||
c4 int main(void) {
0e  int A[MAXM][MAXN], b[MAXM], m, n,
14   xp[MAXN], hombas[MAXN][MAXN], hd;
||
c1  while (scanf("%d %d", &m, &n) == 2
eb   && m > 0 && n > 0) {
7a   FOR(i,m) {
e5    printf("Enter equation %d:\n", i+1);
72    FOR(j,n) scanf("%d", &A[i][j]);
5a    scanf("%d", &b[i]);
37   }
e2   if ((hd = diosolve(A,b,m,n,xp,
d3    hombas)) >= 0) {
28    printf("Particular solution:\n");
0b    FOR(i,n) printf("%d ", xp[i]);
5e    printf("\nhom_dim = %d\n", hd);
9f    printf("Basis for Ax = 0:\n");
55    FOR(j,hd) {
35     FOR(i,n) printf("%d ",hombas[i][j]);
3f     printf("\n");
5c    }
95   } else printf("No solution.\n");
11  }
05 }
|| =================== discs.cc ===================
|| Area & perimeter of a union of discs
79 #include <stdio.h>
8c #include <stdlib.h>
20 #include <string.h>
22 #include <math.h>
35 #include <vector>
09 #include <complex>
13 using namespace std;
||
e2 #define ld long double
f0 #define FOR(i,n) for (int i=0;i<(n);i++)
0c #define PB push_back
c8 #define point complex<ld>
03 #define BEND(x) (x).begin(),(x).end()
c8 #define FORI(i,s) FOR(i,s.size())
||
3f struct circle {
08  point p; ld r;
91  vector<ld> k1, k2;
2e  circle(ld a,ld b,ld c): p(a,b), r(c) {}
||
e7  void kill(point q, ld rr) {
c0   ld d = abs(p−q), e = arg(p−q);
45   if (d>=rr+r||d<=r−rr) return;
59   if (d<=rr−r) k1.PB(0),k2.PB(7);
ff   else {
c2    ld f = acos((r*r−rr*rr+d*d)/2/d/r);
9a    for (int i = −3; i < 4; i+=2)
ac     k1.PB(e−f+i*M_PI),k2.PB(e+f+i*M_PI);
c8   }
6b  }
||
11  void kill(circle c) { kill(c.p,c.r); }
||
83  void pack() {
d3   sort(BEND(k1)); sort(BEND(k2));
ca   vector<ld> q1, q2;
9f   for (int i=0, j; i<k1.size(); i=j) {
2e    for (j = i+1; j < k1.size()
54     && k1[j] <= k2[j−1]; j++);
45    q1.PB((k1[i] >? 0) <? 2*M_PI);
d8    q2.PB((k2[j−1] >? 0) <? 2*M_PI);
cb   }
31   k1=q1; k2=q2;
64  }
||
5c  ld perim() {
55   pack(); ld ans=0;
90   FORI(i,k1) ans += r*fabs(k2[i]−k1[i]);
53   return 2*M_PI*r − ans;
7f  }
||
36  ld area() {
2a   pack(); ld ans=M_PI*r*r, y0=p.real();
09 #define doarc(a,b) (r*(r*(sin(2*a) − \
27  sin(2*b))/2 + r*(a−b) + 2*y0*(sin(a) \
95  − sin(b)))/2)
f5  FORI(i,k1) ans += doarc(k1[i], k2[i]);
cb  return ans;
83  }
ef };
||
```

```
ad #define R (drand48()*1e−10)
72 void doit() {
02  int n; ld p=0, a=0;
71  if (1 != scanf("%i",&n) || !n) exit(0);
c9  vector<circle> da;
a8  FOR(i,n) {
2d   ld a,b,c;
a3   scanf("%Lf%Lf%Lf", &a,&b,&c);
70   da.PB(circle(a+R, b+R, c+R));
2e  }
b6  FORI(i,da) {
8a   FORI(j,da) if (i−j) da[i].kill(da[j]);
4f   p += da[i].perim(); a += da[i].area();
4f  }
60  printf("%.6Lf %.6Lf\n", p, a);
b6 }
||
1e int main() { while (1) doit(); }
|| ==================== dog.cc ====================
||  Bipartite Match
||  ---------------
||
|| Input:
|| m = number of points on left (1 .. m)
|| n = number of points on right
||  (m+1 .. m+n)
|| c = adjacency matrix
||
|| Output:
|| match[i] is 0 or the vertex i is
|| matched to (for i = 1 to m)
||
79 #include <stdio.h>
7b #include <math.h>
c5 #include <assert.h>
||
0a #define FR(i,a,b) \
8f  for(int i=(a);i<(b);i++)
c8 #define FOR(i,n) FR(i,0,n)
||
93 int x[300], y[300];
13 int m,n;
||
57 char c[300][300];
e7 int match[300], back[300];
15 int q[300], qn;
||
7d int find(int i) {
df  int r,j,k;
02  if (match[i]) return 0;
d9  FR(j,1,n+m+1) back[j] = 0;
83  q[0] = i; qn = 1;
83  FOR(k,qn) FR(j,1,n+m+1) {
35   if (!c[q[k]][j]) continue;
e3   if (match[j]) if (!back[j])
b7    back[j] = q[k],
12    back[match[j]] = j,
d5    q[qn++] = match[j];
c7   else {
60    match[q[k]] = j;
0a    match[j] = q[k];
17    for(r=back[q[k]];r;r=back[back[r]])
e7     match[r] = back[r],
ac     match[back[r]] = r;
e6    return 1;
da   }
fb  }
12  return 0;
25 }
||
d2 int main() {
23  int i,j,k;
d8  scanf("%d%d",&n,&m);
ac  for (i=1;i<=n+m;i++)
9e   scanf("%d%d",x+i,y+i);
38  for (i=1;i<=n−1;i++)
6a   for (j=n+1;j<=n+m;j++) {
ba    if (hypot(x[i]−x[j],y[i]−y[j])+
a0     hypot(x[i+1]−x[j],y[i+1]−y[j])<=
cc     2* hypot(x[i]−x[i+1],y[i]−y[i+1]))
d8     c[i][j]=c[j][i] = 1;
13   }
||
d4  for (i=1;i<=m+n;i++) if (find(i)) i=0;
e0  for (k=0,i=1;i<=m;i++)
01   if (match[i]) k++;
1c  printf("%d\n",n+k);
53  for (i=1;i<=n;i++) {
4f   if (i!=1) printf(" ");
f8   printf("%d %d",x[i],y[i]);
82   if (match[i]) printf(" %d %d",
5b    x[match[i]],y[match[i]]);
17  }
ae  printf("\n");
a9  return 0;
```

```
88 }
|| ================== eeuclid.cc ==================
|| ============ Extended Euclid ===============
||
62 template <typename T> void extGCD(T a, T b, T &x
||                                           , T &y)
38 {
b9  if (!b) {
81   x = 1;
10   y = 0;
df   return;
1b  }
72  T x1, y1;
43  extGCD(b, a%b, x1, y1);
f2  x = y1;
cb  y = x1 − (a/b)*y1;
53 }
||
49 Note: in oder to avoid overflow
16 x = y1
7c y = x1 − (a/b)*y1;
34 int d = x/b;
55 x −= d*b;
7d y += d*a;
|| =================== eigen.c ====================
|| UVA 720 − Foxes & Hares
|| Given M & V  (uva 720 gives M−I)
|| Find principal eigenvector
||  (must be non−orthogonal to V)
|| V’ = (M^p)V,  V’’ = (M^(p+1))V
|| V’’ = lambda V’ (solve for lambda)
|| |lambda| < 1 => vanishes
|| |lambda| = 1 =>
||  lambda = 1 => stable
||  lambda cmplx,neg => oscillates
|| |lambda| > 1 => expands
||  lambda positive => unlim. growth
||  lambda cmplx,neg => unstable
||
79 #include <stdio.h>
7b #include <math.h>
||
fe #define eps 1e−12
12 power(double a, double b, double c, double d,
||                             double h, double f){
0d  double A,B,C,D,H,F,aa,bb,cc,dd,nh,nf,nH,nF,
||                                           scale;
d8  int i,j,k;
7b  b = −b;
be  aa=a;bb=b;cc=c;dd=d;
8f  for (j=0;j<1000;j++){
40   A = aa*aa+bb*dd;
ee   B = aa*bb+bb*cc;
3a   D = dd*aa+cc*dd;
16   C = dd*bb+cc*cc;
79   aa=A;bb=B;cc=C;dd=D;
3c   if (A > 1e100 || A < −1e100 || B > 1e100 || B
||                                     < −1e100 ||
4f    C > 1e100 || C < −1e100 || D > 1e100 || D < −
||                                   1e100) break;
f7  }
2f  H = aa*h + bb*f;
86  F = dd*h + cc*f;
88  h = H; f = F;
f8  H = a*h + b*f;
56  F = d*h + c*f;
db  if ((H < 1e−40 && H > −1e−40 && F < 1e−40 && F
||                                       > −1e−40)
b3   && (h < 1e−40 && h > −1e−40 && f < 1e−40 && f
||                            > −1e−40)) return 1;
93  scale = 1/(fabs(h)+fabs(f)+fabs(H)+fabs(F));
23  nh = h*scale; nf = f*scale; nH = H*scale; nF =
||                                        F*scale;
38  if (fabs(nf*nH − nF*nh) > eps || nf*nF < 0 ||
||                          nh * nH < 0) return 6;
f5  if (H < −1e40 && F > 1e40) return 2;
03  if (H > 1e40 && F < −1e40) return 3;
97  if (H < −1e40 && F < −1e40) return 4;
a6  if (H > 1e40 && F > 1e40) return 5;
ad  return 6;
48 }
||
94 main(){
d3  int n,i,j,k,r,result;
08  double a,b,c,d,f,h,ea,eb,ec,ed,ef,eh;
12  scanf("%d",&n);
ba  for (i=0;i<n;i++){
6c   scanf("%lf%lf%lf%lf%lf%lf",&a,&b,&c,&d,&h,&f);
34   result = power(a,b,c,d,h,f);
73   for (ea=−eps;ea<=eps;ea+=eps) for (eb=−eps;eb<
||                                  =eps;eb+=eps)
7c    for (ec=−eps;ec<=eps;ec+=eps) for (ed=−eps;ed
||                                 <=eps;ed+=eps)
6e     for (eh=−eps;eh<=eps;eh+=eps) for (ef=−eps;
||                             ef<=eps;ef+=eps) {
```

```
18      r = power(a*(1+ea),b*(1+eb),c*(1+ec),d*(1+
||                       ed),h*(1+eh),f*(1+ef));
a3      if (r != result) result = 6;
08     }
4e   switch(result){
20   case 1:
ee    printf("Ecological balance will develop.\n");
||                                          break;
a0   case 2:
87    printf("Hares will die out while foxes will
||                         overgrow.\n"); break;
05   case 3:
57    printf("Hares will overgrow while foxes will
||                          die out.\n"); break;
44   case 4:
2d    printf("Both hares and foxes will die out.\n"
||                                     ); break;
ae   case 5:
2b    printf("Both hares and foxes will overgrow.\n
||                                    "); break;
b6   case 6:
50    printf("Chaos will develop.\n");
7f   }
76  }
9e }
|| ================== fasthull.c ==================
|| Ken Clarkson wrote this.  Copyright (c) 1996 by
||                                          AT&T..
|| Permission to use, copy, modify, and distribute
||                          this software for any
|| purpose without fee is hereby granted, provided
||                        that this entire notice
|| is included in all copies of any software which
||                          is or includes a copy
|| or modification of this software and in all
||                       copies of the supporting
|| documentation for such software.
|| THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT
||                         ANY EXPRESS OR IMPLIED
|| WARRANTY.  IN PARTICULAR, NEITHER THE AUTHORS
||                              NOR AT&T MAKE ANY
|| REPRESENTATION OR WARRANTY OF ANY KIND
||                  CONCERNING THE MERCHANTABILITY
|| OF THIS SOFTWARE OR ITS FITNESS FOR ANY
||                            PARTICULAR PURPOSE.
||
|| two−dimensional convex hull
|| read points from stdin,
||  one point per line, as two numbers separated by
||                                      whitespace
|| on stdout, points on convex hull in order around
||                                     hull, given
||  by their numbers in input order
|| the results should be "robust", and not return a
||                             wildly wrong hull,
||  despite using floating point
|| works in O(n log n); I think a bit faster than
||                                    Graham scan;
||  somewhat like Procedure 8.2 in Edelsbrunner’s
||  "Algorithms in Combinatorial Geometry".
||
d7 #include <stdlib.h>
3f #include <stdio.h>
d3 #include <assert.h>
||
4a typedef double coord;
53 char input_format[] = "%lf%lf";
||
e6 #define N 100000
||
af coord points[N][2], *P[N+1]; // an extra
||                                position is used
||
d8 int read_points(void) {
be  int n = 0;
2b  char buf[100];
65  while (fgets(buf, sizeof(buf), stdin)) {
6d   assert(2==sscanf(buf, input_format,&points[n][
||                           0],&points[n][1]));
53   P[n] = points[n];
a2   assert(++n <= N);
58  }
d1  return n;
bb }
||
7c void print_hull(coord **P, int m) {
d8  int i;
6b  for (i=0; i<m; i++)
7a   printf("%0.6f %0.6f\n",P[i][0],P[i][1]);
||
||   printf("%d ", (P[i]−points[0])/2);
||  printf("\n");
7d }
||
||
```

```
b3 int ccw(coord **P, int i, int j, int k) {
dc  coord a = P[i][0] − P[j][0],
b1   b = P[i][1] − P[j][1],
a2   c = P[k][0] − P[j][0],
11   d = P[k][1] − P[j][1];
98  return a*d − b*c <= 0;     // true if points i,
||                           j, k counterclockwise
59 }
||
||
9b #define CMPM(c,A,B) \
f7  v = (*(coord**)A)[c] − (*(coord**)B)[c];\
7c  if (v>0) return 1;\
90  if (v<0) return −1;
||
15 int cmpl(const void *a, const void *b) {
cf  double v;
40  CMPM(0,a,b);
7c  CMPM(1,b,a);
18  return 0;
1d }
||
20 int cmph(const void *a, const void *b) {return
||                                   cmpl(b,a);}
||
||
98 int make_chain(coord** V, int n, int (*cmp)(
||                   const void*, const void*)) {
42  int i, j, s = 1;
13  coord* t;
||
c5  qsort(V, n, sizeof(coord*), cmp);
bc  for (i=2; i<n; i++) {
4f   for (j=s; j>=1 && ccw(V, i, j, j−1); j−−) {}
92   s = j+1;
2e   t = V[s]; V[s] = V[i]; V[i] = t;
64  }
b3  return s;
18 }
||
a1 int ch2d(coord **P, int n)  {
08  int u = make_chain(P, n, cmpl);     // make
||                                     lower hull
bf  if (!n) return 0;
1c  P[n] = P[0];
f7  return u+make_chain(P+u, n−u+1, cmph);  // make
||                                     upper hull
7a }
||
||
9d int main(int argc, char** argv) {
c4  print_hull(P, ch2d(P, read_points()));
5d  return 0;
df }
|| ==================== fft.cc ====================
|| ============= FFT ============
||
83 typedef complex<double> cd;
b2 cd W[1<<18];
c8 template<class S>void fft(S p,cd*y,int n,int k=1
||                                              ){
6c  if(n>1){ cd q;
ae   fft(p,y,n/=2,k*2),fft(p+k,y+n,n,k*2);
b8   for(int i=0,j=0;i<n;++i,j+=k)
4f    y[i+n]=y[i]−(q=y[i+n]*W[j]),y[i]+=q;
b0  }else *y=*p;
88 }
7e vi mul(vector<int> a,vector<int> b){
9d  int n=a.size()+b.size();
61  for(;n&n−1;++n);
62  a.resize(n),b.resize(n);
93  vector<cd> Y(n),Z(n);
bb  REP(i,n)W[i]=polar(1.,2*i*M_PI/n);
41  fft(&a[0],&Y[0],n),fft(&b[0],&Z[0],n);
26  REP(i,n)Z[i]*=Y[i],W[i]=cd(real(W[i]),−imag(W[i
||                                            ]));
d1  fft(&Z[0],&Y[0],n);
81  REP(i,n)a[i]=(real(Y[i])+0.5)/n;
19  for(;a.size() && !a.back();a.erase(a.end()−1));
ad  return a;
70 }
|| ================== fft−ff.cc ===================
52 typedef unsigned long long ull;
||
|| finite field FFT:
||  outputs < P
||  size a power of 2 <= 2^30
13 const ull P = 3221225473ULL; // = 3*2^30+1
||
ff int rev(int x, int b) {
d7  b/=2;
11  if (b) return b * (x&1) + rev(x/2, b);
9a  return 0;
71 }
||
```

```
|| finite field fft − find x[0..n] so that
|| x[i] = sum(j=0^n) w^(ij) v[j]
|| where w is a primitive nth root of 1.
|| I=−1 is inverse fft.
5e vector<ull> fft(const vector<ull> &z, int I=1) {
ef  vector<ull> v;
4e  FORI(i,z) v.push_back(z[rev(i,z.size())]);
39  for (int m=1; m<v.size(); m+=m) {
9c   ull ord = (P−1)/3;
c6   ull wm = I==1 ? 5 : 1932735284ULL;
68   wm = (((wm*wm)%P)*wm)%P;
||
0f   while (ord > 2*m) {
bf    ord /= 2;
e8    wm = (wm*wm)%P;
2a   }
||
3d   for (int k=0; k<v.size(); k+=m+m) {
61    ull w=1;
8b    FR(j,k,k+m) {
14     ull t = (w * v[j+m]) % P;
55     v[j+m] = (v[j]+P−t) % P;
1e     v[j] = (v[j] + t) % P;
3c     w = (w*wm) % P;
cc    }
6b   }
46  }
f7  ull invsz = 1;
55  for (int j=1; j<v.size(); j*=2) invsz = (invsz
||                             * 1610612737ULL) % P;
ad  if (I==−1) FORI(i,v) v[i] = (v[i] * invsz) % P;
e3  return v;
12 }
||
|| multiply polynomials, power of 2 size (
||                                     destructive)
d9 void mul(vector<ull> &a, vector<ull> &b) {
bf  a = fft(a); b = fft(b);
35  FORI(i,a) a[i] = (a[i]*b[i]) % P;
64  a = fft(a,−1);
fe }
|| ================= fft−peng.cc ==================
50 #include <complex>
||
b2 typedef long double ld;
67 typedef complex<ld> P;
||
|| Acts on the array P a[], dir=1 is the FFT and −1
||                                  gives the iFFT
|| The number of elmeents in a must be a power of 2
||                                    , namely 2^p.
|| This has enough precision to do 10^6 numbers
||  less than 1000, which suffices for most multipl
||  ication/convolution problems. If numbers are to
||  o big...maybe splitting all the digits would do
||   the trick since that at most triples the probl
||                                     em for ints
||
c0 ld twoPI=acos(0)*ld(4);
f1 P tem[MAXN];
7a void fft(int p,int dir){
d4  int l,r,sz2,sz;
9f  P x,v;
77  RF(i,1,p){
d6   l=0; sz=1<<i;
c1   r=sz2=1<<(i+1);
3b   FOR(j,(1<<(p−i−1))){
88    FOR(k,sz) tem[k]=a[l+k*2+1];
3c    FOR(k,sz) a[l+k]=a[l+k*2];
d9    FOR(k,sz) a[l+sz+k]=tem[k];
51    l+=sz2; r+=sz2;
4f   }
0f  }
1e  FOR(i,p){
43   l=0; sz=1<<i;
69   r=sz2=1<<(i+1);
38   x=exp(P(0,ld(dir)*twoPI/ld(sz2)));
64   FOR(j,(1<<(p−i−1))){
04    v=P(1,0);
31    FOR(k,sz2){
79     tem[k]=a[l+k%sz]+v*a[l+sz+k%sz];
e0     v*=x;
e2    }
11    FOR(k,sz2) a[l+k]=tem[k];
7e    l+=sz2; r+=sz2;
5e   }
c7  }
c3 }
||
|| Debug code
dd void show(P p){printf("%Lf,%Lf\n",X(p),Y(p));}
|| ==================== flow.h ====================
|| Network flow stuff  (Cleaned up & generalized
||                               from "Councillors")
|| reset() should be called before building the
```

```
||                                            graph
|| variables "source" and "sink" are the source and
||                                     sink vertices
|| edge(from, to, min, max, initflow) builds the
||                                            graph
|| setup() should be called after building graph/
||                          before maxflow/minflow
|| minflow() returns min flow from source to sink
|| maxflow() returns max flow from source to sink
||  − flows are of type "double", may be negative
||  − don’t forget that in integers up to 15 digits
||                              , doubles are exact
||  − may be called repeatedly − only need setup if
||                              edges added/deleted
||  − flow must be feasible before calling [see
||                                  makefeasible()]
|| resetflow(edge, amt) tries to set the flow on
||                          edge to amt maintaining
||  current source−sink flow
||  − gets as close as possible and returns
||                              resulting edge flow
||  − edge is a cut edge flow is optimal and
||                             setflow(edge,0) != 0
|| isfeasible() returns 1 if all edge flows
||                                         feasible
|| makefeasible() returns 1 if successful (flow not
||                                       optimized)
|| dump() a crude printout of the graph with
||                                          weights
|| E(from,to) − returns internal edge number for
||                                         from−>to
|| getmax(i), getmin(i), getflow(i), setmax(i,m),
||                                       setmin(i,m)
||  − use these to query/set edge constraints/flows
||  − paramter is an internal edge number returned
||                                         from E()
|| *DO NOT USE* setflow or the fields in E[i].f
||                                         directly
|| − initial flows *must* balance (all 0 is safe)
||
de #define get(i,F,G) (e[i].from<e[i].to?e[i].f−>F:
||                                      −e[i].f−>G)
8f #define getmax(i) get(i,max,min)
86 #define getmin(i) get(i,min,max)
2a #define getflow(i) get(i,f,f)
||
03 #define set(i,F,G,x) (e[i].from<e[i].to?(e[i].f−
||                          >F=(x)):(e[i].f−>G=−(x)))
f6 #define setmax(i,x) set(i,max,min,x)
87 #define setmin(i,x) set(i,min,max,x)
38 #define setflow(i,x) set(i,f,f,x)
||
50 struct ff {
49  double min, max, f;
23 } f[100000];
||
90 struct ee {
db  int from, to;
e7  struct ff *f;
26 } e [100000];
||
73 int cookie, ne, firste[100000], cooked[100000],
||                           source = 0, sink = 1;
||
f2 int E(int from, int to) {  // find edge number
||                          from−>to; ne if not found
af  int r;
96  for (r=firste[from];r<ne && e[r].to!=to;r++) {}
7e  return r;
ce }
||
ae int comp(struct ee *a, struct ee *b) {
ef  if (a−>from != b−>from) return a−>from − b−>
||                                            from;
d9  return a−>to − b−>to;   // fix for IBM non−
||                                     stable qsort
5b }
||
fb void edge(int from, int to, double min, double
||                             max, double flow) {
18  e[ne].from = e[ne+1].to = from;
b3  e[ne].to = e[ne+1].from = to;
aa  e[ne].f = e[ne+1].f = &f[ne];
b4  setmin(ne,min);
4b  setmax(ne,max);
08  setflow(ne,flow);
09  ne+=2;
dd }
||
dd void reset() {
26  ne = 0;
4b  edge(source,sink,0.0L,0.0L,0.0L);  // dummy
||                           edge for makefeasible()
57 }
||
fd #define augv(x,y) (++cookie,Xaugv(x,y))
7c double Xaugv(int v, double amt) {  // amt may be
||                                         negative
e6  int i;
00  if (v == sink) return amt;
bb  if (cooked[v] == cookie) return 0;
f7  cooked[v] = cookie;
17  for (i=firste[v];e[i].from == v;i++) {
2d   double min = getmin(i), max = getmax(i), flow
||                                    = getflow(i);
e2   double namt = amt;
||
30   if (flow > max && amt > 0) continue;      //
||                             fix for makefeasible
c8   else if (flow < min && amt < 0) continue;
||
de   if (amt > max - flow) namt = max-flow;
09   else if (amt < min - flow) namt = min-flow;
c5   if (namt == 0) continue;
79   namt = Xaugv(e[i].to,namt);
83   setflow(i,getflow(i)+namt);
e5   if (namt != 0) return namt;
9a  }
ea  return 0;
2a }
||
54 void setup() {
e3  int i;
8e  e[ne].from = -1;
56  qsort(e,ne,sizeof(struct ee),comp);
5f  for (i=ne-1;i>=0;i--) firste[e[i].from] = i;
a7 }
||
|| the rest of these functions are optional
||
ef double maxflow() {
dd  int i; double tot = 0;
2c  while (augv(source,1e99)) {}
c5  for (i=firste[sink];e[i].from == sink;i++) tot
||                                   -= getflow(i);
ad  return tot;
a0 }
||
ed double minflow() {
9d  int i; double tot = 0;
83  while (augv(source,-1e99)) {}
d0  for (i=firste[sink];e[i].from == sink;i++) tot
||                                   -= getflow(i);
68  return tot;
56 }
||
|| needed for makefeasible
cd double resetflow(int i, double amt) { // edge
||                     flow; overall flow unchanged
e6  int ssink=sink, smax=getmax(i), smin=getmin(i);
e8  double r, rr = getflow(i);
c2  setmax(i,rr);
d5  setmin(i,rr);
29  sink = e[i].from;
72  while ((r=augv(e[i].to,amt-rr))) rr += r;
50  sink = ssink;
46  setmin(i,smin);
ad  setmax(i,smax);
d4  setflow(i,rr);
9c  return rr;
e1 }
||
a6 int isfeasible() {
d0  int i;
78  for (i=0;i<ne;i++) {
37   if (getflow(i)<getmin(i)) return 0;
9c   if (getflow(i)>getmax(i)) return 0;
b1  }
22  return 1;
34 }
||
41 int makefeasible() {
25  int i;
e8  setmax(0,1e100);
81  setmin(0,-1e100);
36  for (i=0;i<ne;i++) {
da   double flow = getflow(i), min=getmin(i), max =
||                                       getmax(i);
f0   if (flow < min && resetflow(i,min) != min)
||                                           break;
af   if (flow > max && resetflow(i,max) != max)
||                                           break;
a0  }
7d  setflow(0,0);
e6  setmax(0,0);
09  setmin(0,0);
db  return i==ne;
db }
||
||
```

```
e5 void dump() {
03  int i;
b0  for (i=0;i<ne;i++) {
58   printf("%d->%d %g %g %g\n",e[i].from,e[i].to,
1f    getmin(i), getmax(i), getflow(i));
||
7d  }
7e }
|| =============== flowlite-adj.cpp ===============
17 #include <vector>
93 #include <iostream>
36 #include <fstream>
d7 using namespace std;
df #define fu(i,n) for(int i=0; i<n; i++)
de #define pb push_back
||
53 #define MAXV 1000
54 vector<int> adj[MAXV],fl[MAXV],mx[MAXV];
e8 vector<int> back[MAXV]; // I would have liked to
||   make these pointers, but vector resizes are de
||                                             adly
||
e2 int cookie;
6f int been[MAXV];
||
|| nodes A,B with flow AB forward and BA back
|| We do not look kindly on repeated edges
79 int connect(int A, int B, int AB, int BA) {
7b  fl[A].pb(0);  fl[B].pb(0);
69  mx[A].pb(AB); mx[B].pb(BA);
51  adj[A].pb(B); adj[B].pb(A);
c3  back[A].pb(fl[B].size()-1);
b8  back[B].pb(fl[A].size()-1);
e0 }
||
41 int aug(int inc, int src, int snk) {
26  if(src==snk) return inc;
a6  if(been[src] == cookie) return 0;
b6  been[src]=cookie;
00  fu(i,fl[src].size())
de   if(mx[src][i] >= inc+fl[src][i]-fl[adj[src][i]
||                                 ][back[src][i]]
20    && aug(inc,adj[src][i],snk)) {
9c    fl[src][i] += inc;
12    return 1;
5d   }
52  return 0;
17 }
||
6b int maxflow(int src, int snk) {
14  int ret=0;
a2  for(int inc=0x12345678; inc>0;inc>>=1)
42   while(++cookie && aug(inc,src,snk)) ret+=inc;
80  return ret;
c1 }
||
|| Mainline for usaco problem 93
d2 int main() {
f8  ifstream in("ditch.in");
ac  ofstream out("ditch.out");
||
06  int N,M;
04  in >> N >> M;
e5  fu(i,N) {
ca   int A,B,F;
70   in >> A >> B >> F;
d8   connect(A,B,F,0);
4b  }
a4  out << maxflow(1,M) << endl;
e0 }
||
||
|| ================== flowlite.c ==================
|| -------------- Sample Mainline "Circus SEEUR 99"
||                                 ----------------
ae #include "flowlite.h"
b5 #include <string.h>
f4 #include <stdio.h>
e0 void dump(int n, int src, int snk, int mx[][SZ],
||                                   int fl[][SZ]){
15  int i,j;
c1  printf("dump:\n");
6e  for (i=0;i<n;i++) for (j=0;j<n;j++) if (-mx[j][
||                                               i]
50   || -mx[i][j] ) {
82   printf("from %d to %d min %d max %d flow %d\n"
||                                                ,
8f    i,j,-mx[j][i],mx[i][j],fl[i][j]-fl[j][i]);
69  }
be }
||
1b int Max[SZ][SZ], Flow[SZ][SZ];
||
88 int i,j,k,m,n,t,source=0,sink=1;
||
```

```
d2 int main(){
d8  scanf("%d",&t);
36  while (t--) {
bb   memset(Max,0,4*SZ*SZ);
f5   memset(Flow,0,4*SZ*SZ);
a6   scanf("%d",&n);
04   scanf("%d",&m);
5b   for (i=1;i<=n;i++) {
62    Max[2*i+1][2*i] = -1000;   //Min[2*i][2*i+1]
||                                          = 1000;
ee    Max[2*i][2*i+1] = 1000;
87    Max[source][2*i] = 1000;
9f    Max[2*i+1][sink] = 1000;
29   }
ef   for (i=0;i<m;i++){
c4    scanf("%d%d",&j,&k);
f2    Max[2*j+1][2*k] = 1000;
3a   }
     //dump(2*n+2,source,sink,Max,Flow);
82   if(! makefeas(2*n+2,source,sink,Max,Flow))
||                              printf("***OOPS\n");
     //dump(2*n+2,source,sink,Max,Flow);
ab   k = minflow(2*n+2,source,sink,Max,Flow);
     //dump(2*n+2,source,sink,Max,Flow);
80   printf("%d\n",k);
71  }
c7  return 0;
73 }
|| ================== flowlite.h ==================
|| Maxflow Lite
||
|| flow = maxflow(n, src, sink, mx, fl)
|| flow = minflow(n, src, sink, mx, fl)
|| succ = makefeas(n, src, sink, mx, fl)
|| cost = mincost(n, mx, cst, fl)
||
|| Graph vertices are in range 0..n-1
||
c4 #define SZ 500
63 #define FOR(i,n) for (int i=0;i<n;i++)
||
6b static int been[SZ];
33 int aug(int cookie, int inc, int n, int
78 src, int snk,int mx[][SZ],int fl[][SZ]){
b5  int i;
f6  if (src == snk) return inc;
4e  if (been[src] == cookie) return 0;
69  been[src] = cookie;
5a  for (i=0;i<n;i++)
00   if (mx[src][i] >= inc
18    +fl[src][i]-fl[i][src]
5d    && aug(cookie,inc,n,i,snk,mx,fl)) {
e1     fl[src][i] += inc;
2a     return 1;
b4   }
91  return 0;
59 }
||
eb static int cookie;
a7 int maxflow(int n, int src, int snk,
5d  int mx[][SZ], int fl[][SZ]) {
e5  int i,r,inc;
dc  for (inc=0x40000000;inc>0;inc/=2)
8a   while(aug(++cookie,inc,n,
10    src,snk,mx,fl));
2b  r=0;FOR(i,n) r+=fl[src][i]-fl[i][src];
62  return r;
26 }
||
04 #define minflow(n,src,snk,mx,fl) \
da  -maxflow(n,snk,src,mx,fl)
||
|| makefeas - make flow feasible,
|| possibly changing src->snk flow
||
39 int makefeas(int n, int src, int snk,
aa  int mx[][SZ], int fl[][SZ]){
ac  int i,j,d,r,inc;
aa  for(i=0;i<n;i++) for (j=0;j<n;j++)
dd   if ( 0 < (d=(fl[i][j]-fl[j][i])-
35    mx[i][j])) {
d7   fl[i][j] -= d;
e5   mx[src][snk] = mx[snk][src] = d;
1d   for (inc=d;inc>0;)
4d    if (aug(++cookie,inc,n,i,j,mx,fl))
de     inc=(d-=inc);
4a    else inc /= 2;
68   mx[src][snk] = mx[snk][src] =
7c    fl[snk][src] = fl[src][snk] = 0;
5f   if (d) {
ad    FOR(i,n) FOR(j,n) fl[i][j]=0;
31    return 0;
94   }
39  }
65  return r;
```

```
b6 }
||
|| mincost flow - leaves flow balance
|| alone, minimizes SUM flow*cost
||
81 void caug(int n, int from, int to, int
23  amt, int h[][SZ], int fl[][SZ]) {
aa  int i,j,k;
b5  if (h[from][to] < 0) {
16   fl[from][to] += amt;
bc  } else {
2b   caug(n,from,h[from][to],amt,h,fl);
fc   caug(n,h[from][to],to,amt,h,fl);
9a  }
61 }
||
65 int mincost(int n, int mx[][SZ],
62  int cost[][SZ], int fl[][SZ]) {
a5  int i,j,k;
c7  static int c[SZ][SZ], d[SZ][SZ],
09   h[SZ][SZ];
ab  again: {
70   FOR(i,n) FOR(j,n) {
a0    h[i][j] = -1;
d9    c[i][j]=mx[i][j]-fl[i][j]
05     +fl[j][i];
9c    if (c[i][j] > 0) d[i][j] =
f6     cost[i][j] - cost[j][i];
42    else d[i][j] = 0x3fffffff;
27   }
33   FOR(j,n) FOR(i,n) FOR(k,n) {
7e    if (d[i][j] + d[j][k] < d[i][k]) {
22     h[i][k] = j;
2a     d[i][k] = d[i][j] + d[j][k];
aa     c[i][k] = c[i][j]<?c[j][k];
24     if (i == k && d[i][k] < 0) {
ed      caug(n,i,k,c[i][k],h,fl);
4d      goto again;
0f     }
1d    }
a5   }
20  }
85  k=0; FOR(i,n) FOR(j,n)
04   k+=fl[i][j]*(cost[i][j]-cost[j][i]);
24  return k;
1c }
|| =================== frcg.cc ====================
|| frcg example --- dhaka02 Hermes in
|| the case where there are 3 points.
ae #include "frcg.h"
||
92 vec p[4];
||
c9 vec grad3(const vec&x) {
26  vec grad(3);
81  FOR(i,3) if (sz(x-p[i]) > 1e-6)
23   grad += (x-p[i]) / sz(x-p[i]);
9a  return grad;
09 }
e1 ld f3(const vec&x) {
3c  return sz(x-p[0]) + sz(x-p[1])
ae   + sz(x-p[2]);
b0 }
||
3f int caseno;
||
ed void doit() {
25  int n;
9c  scanf("%i", &n);
ee  printf("Province # %i : ", ++caseno);
95  FOR(i,n) {
85   p[i].resize(3);
ff   FOR(j,3) scanf("%Lf", &p[i][j]);
ba  }
fe  vec x = (p[0] + p[1] + p[2]) / 3;
c2  FOR(zzz,8) {
06   FORI(i,x) printf("%.12Lf ", x[i]);
26   printf("\n");
7f   cg<f3,grad3>(x);
2e  }
7d  printf("%.2Lf\n", f3(x));
16 }
||
d2 int main() {
e5  int n;
a8  scanf("%*f %*f %*f %*f %i", &n);
2f  while (n--) doit();
dc }
|| ==================== frcg.h ====================
|| Fletcher-Reeves conjugate gradient
|| minimisation.  cg() does n iterations
|| of CG; if you need more, call it many
|| times.  Each iteration does O(1)
|| vector ops plus a few calls to f plus
|| one call to grad; it is fast.  main()
|| half-solves hermes from dhaka02.
79 #include <stdio.h>
8c #include <stdlib.h>
8e #include <math.h>
09 #include <valarray>
0b using namespace std;
||
e2 #define ld long double
d2 #define FOR(i,n) for (int i=0;i<n;i++)
36 #define FORI(i,s)FOR(i,(signed)s.size())
a5 #define vec valarray<ld>
|| Knobs for you to play with.  CG works
|| in theory as long as 0 < c1 < c2 < .5
08 #define EPS 1e-6
e2 #define c1 .1
2c #define c2 .4
||
d1 ld sz(vec v){ return sqrt((v*v).sum());}
76 ld dot(vec a,vec b){ return(a*b).sum();}
||
e3 template <ld f(const vec&),
76  vec grad(const vec&)>
79 void linmin(vec&x, const vec&d) {
cf  ld sd = -c1*dot(d,grad(x));
b3  if (sd < 0) return;
cd  ld cc = c2/c1 * sd;
c4  ld lb = 0, ub = 1, fx = f(x);
87  while (f(x+2*ub*d) < fx - 2*ub*sd
39   && ub < 1e6) ub *= 2;
82  FOR(zz,55) {
d9   ld c = (lb+ub)/2;
37   if (f(x+c*d) > fx - c*sd) ub=c;
f3   else {
a5    ld dd = dot(d, grad(x+c*d));
6c    if (dd > cc) ub = c;
b9    else if (dd < -cc) lb = c;
90    else { lb = ub = c; break; }
11   }
65  }
d5  x += lb * d;
f3 }
||
e3 template <ld f(const vec&),
76  vec grad(const vec&)>
e6 void cg(vec&x) {
39  ld las = 1;
77  vec l(x.size());
39  FORI(zzz,x) {
74   vec g = grad(x);
ba   if (sz(g) < EPS) break;
39   l = g + (dot(g,g) / las) * l;
d1   las = dot(g,g);
7d   if (sz(l) > EPS)
18    linmin<f,grad>(x, l/-sz(l));
0d  }
45 }
|| ================== gauss.h ===================
|| Gaussian elimination
||
|| int solve(m,n,A,X)
|| m equations
|| n <= m unknowns (you can always add
||  equations with 0 coefficients)
|| A[MAXM][MAXN] - A[i][0..n-1] is the
|| lhs, A[i][n] is the rhs of equation
||
|| X[MAXN] - solution for each unknown
|| IEEE inf if unconstrained.
||
|| returns 0 if
||  - all m equations are consistent
||  - matrix has full rank
||
|| returns -1 if the equations are
||  inconsistent
||
|| returns +1 if equations consistent,
|| but some variables unconstrained.
||
|| Possible enhancements:
|| - matrix inversion - stick an iden-
||  tity matrix on the right instead of
||  just the rhs of each row.
|| - find the basis - the algorithm se-
||  lects n "best" rows ... these are
||  the original n equations that form
||  the basis - the rest just go along
||  for the ride (so long as they are
||  consistent)
||
||
||
02 #include <math.h>
||
56 #define MAXM 100
```

```
6d #define MAXN 100+1
||
7a int solve(int m,int n, double A[][MAXN],
c6  double X[]) {
76  int i,j,k,ii,best,res=0;
2d  double t,r;
||
79  for (i=0,ii=0;i<n;i++) {
a9   for (best=i,j=0;j<m;j++)
90    if (fabs(A[j][i]) >
37     fabs(A[best][i]) &&
cc     (i <= j || fabs(A[j][j])
da     < 1e-10)) best=j;
29   if (fabs(r=A[best][i]) < 1e-10)
5c    continue;  // singular
ba   for (k=ii++;k<=n;k++) {
5f    t = A[best][k];
88    A[best][k] = A[i][k];
ec    A[i][k] = t/r;
7a   }
91   for (j=0;j<m;j++) if (j != i) {
1a    r = A[j][i];
2f    for (k=i;k<=n;k++)
03     A[j][k] -= r * A[i][k];
5d   }
bf  }
||
0e  for (i=0;i<m;i++) {
02   for (j=0,r=0;j<n;j++)
e0    r += A[i][j] * A[j][n];
01   if (fabs(r-A[i][n]) > 1e-10*(fabs(r)
06    +fabs(A[i][n]))) return -1;
7d  }
81  for (i=0;i<n;i++) {
9a   for (j=0,r=0;j<n;j++)
9a    r += fabs(A[i][j] - (i==j));
a5   if (r > 1e-10) {
46    X[i] = 1.0/0.0;
82    res = 1;
d2   } else X[i] = A[i][n];
e3  }
a8  return res;
a8 }
|| ============== gaussrational.c ================
|| Solution of systems of linear equations over the
||                                          integers
|| Author: Howard Cheng
|| Reference:
|| K.O. Geddes, S.R. Czapor, G. Labahn. "Algorithms
||                              for Computer Algebra"
||  Kluwer Academic Publishers, 1992, pages 393-399
||                              .  ISBN 0-7923-9259-0
|| The routine fflinsolve solves the system Ax = b
||                                   where A is an n x n
|| matrix of integers and b is an n-dimensional
||                                vector of integers.
|| The inputs to fflinsolve are the matrix A, the
||                                dimension n, and an
|| output array to store the solution x_star = det(
||                                A)*x.  The function
|| also returns the det(A).  In the case that det(A
||                                ) = 0, the solution
|| vector is undefined.
|| Note that the matrix A and b may be modified.
||
79 #include <stdio.h>
||
b2 #define MAX_N 10
||
c7 int fflinsolve(int A[MAX_N][MAX_N], int b[MAX_N]
||                       , int x_star[MAX_N],int n)
f5 {
19  int sign, d, i, j, k, k_c, k_r, pivot, t;
||
20  sign = d = 1;
||
b0  for (k_c = k_r = 0; k_c < n; k_c++) {
     // eliminate column k)c
||
     // find nonzero pivot
ee   for (pivot = k_r; pivot < n && !A[pivot][k_r];
||                                          pivot++)
dc    ;
||
89   if (pivot < n) {
      // swap rows pivot and k_r
50    if (pivot != k_r) {
eb     for (j = k_c; j < n; j++) {
c3      t = A[pivot][j];
34      A[pivot][j] = A[k_r][j];
36      A[k_r][j] = t;
8c     }
3d     t = b[pivot];
61     b[pivot] = b[k_r];
e9     b[k_r] = t;
```

```
||
93     sign *= -1;
49    }
||
      // do elimination
9f    for (i = k_r+1; i < n; i++) {
1c     for (j = k_c+1; j < n; j++) {
c1      A[i][j] = (A[k_r][k_c]*A[i][j]-A[i][k_c]*A[
||                                    k_r][j])/d;
cb     }
80     b[i] = (A[k_r][k_c]*b[i]-A[i][k_c]*b[k_r])/d
||                                               ;
be     A[i][k_c] = 0;
d0    }
ef    if (d) {
1c     d = A[k_r][k_c];
0d    }
ac    k_r++;
56   } else {
      // entire column is 0, det(A) = 0
b2    d = 0;
00   }
dd  }
||
9e  if (!d) {
72   for (k = k_r; k < n; k++) {
d0    if (b[k]) {
       // inconsistent system
7c     printf("Inconsistent system.\n");
06     return 0;
1d    }
cd   }
     // multiple solutions
81   printf("More than one solution.\n");
8e   return 0;
f1  }
||
    // now backsolve
d7  for (k = n-1; k >= 0; k--) {
ad   x_star[k] = sign*d*b[k];
05   for (j = k+1; j < n; j++) {
ea    x_star[k] -= A[k][j]*x_star[j];
f2   }
5c   x_star[k] /= A[k][k];
cf  }
||
b6  return sign*d;
e0 }
||
e3 int main(void)
c4 {
6c  int A[MAX_N][MAX_N], x_star[MAX_N], b[MAX_N];
50  int n, i, j;
f9  int det;
||
0e  while (scanf("%d", &n) == 1 && 0 < n && n <=
||                                         MAX_N) {
9a   printf("Enter A:\n");
ee   for (i = 0; i < n; i++) {
6d    for (j = 0; j < n; j++) {
b5     scanf("%d", &(A[i][j]));
00    }
15   }
b7   printf("Enter b:\n");
07   for (i = 0; i < n; i++) {
09    scanf("%d", &(b[i]));
95   }
af   if ((det = fflinsolve(A, b, x_star, n))) {
ed    printf("det = %d\n", det);
2b    printf("x_star = ");
ab    for (i = 0; i < n; i++) {
b7     printf("%d ", x_star[i]);
f0    }
dd    printf("\n");
91   } else {
d7    printf("A is singular\n");
2b   }
b4  }
9e  return 0;
59 }
|| =============== generalmatch.cc ================
|| maximum-cardinality matching
||
|| usage:
|| - vertices are 1...n.
|| - edges are 1...m2.
|| - graph is stored in forward-star
||  form.  this means that edges are
||  represented as pairs of opposing
||  directed edges, and these directed
||  edges are sorted by head.
|| - firste[v] is the first directed
||  edge with v as head.  the tail of
||  edge e is endv[e].
|| - on termination, the following
```

```
||  arrays are meaningful:
||  - mate[1...n]: mate[i] is the mate
||   of vertex i, or 0 if i is exposed.
||  - expo: the number of exposed nodes.
||
79 #include <stdio.h>
3d #define maxvar  1000+1 // leave 1 extra
f2 #define maxarc2 10000+1// leave 1 extra
||
14 static long n, m2, expo;
a3 static long firste[maxvar+1],
17  endv[maxarc2], mate[maxvar];
||
24 static void match(){
a5  long back[maxvar], q[maxvar],
12   intree[maxvar];
84  long head, last, v3, v2, next, root,
09   tail, v, x, y;
||
35  expo = n;
3a  for (x=1;x<=n;x++) mate[x] = 0;
66  for (root=1;root<=n&&expo>=2;root++) {
f8   if (mate[root]) continue;
e5   for (x=1; x<=n; x++) intree[x] = 0;
0c   intree[root] = 1;
d8   q[0] = root;
b1   for(head=tail=0;head<=tail;head++) {
83    v = q[head];
77    for (x=firste[v]; x<firste[v+1] &&
af     (v3=mate[v2=endv[x]]); x++) {
4c     if (v3==v||intree[v2]) continue;
cd     for (y=v; y != root && y != v2;
1a      y=back[y]);
ac     if (y == root) {
68      intree[v2] = 1;
84      back[v3] = v;
6d      q[++tail] = v3;
de     }
33    }
58    if (x<firste[v+1]) {
49     for (;;) {
fa      next = mate[v];
49      mate[v] = v2;
ba      mate[v2] = v;
ca      if (!next) break;
c3      v = back[v];
0f      v2 = next;
65     }
b0     expo -= 2;
77     break;
86    }
b9   }
d0  }
33 }
||
eb static void infile() {
8e  long i;
57  fscanf(stdin, "%ld", &n);
20  fscanf(stdin, "%ld", &m2);
5d  for (i=1; i<=n+1; i++)
43   fscanf(stdin,"%ld",&firste[i]);
c2  for (i=1; i<=m2; i++)
a0   fscanf(stdin, "%ld", &endv[i]);
b8 }
||
ba static void outfile() {
ca  long i;
e1  fprintf(stdout, " the solution obtain"
c1   "ed using matching algorithm is \n");
02  for (i=1; i<=n; i++) {
a3   fprintf(stdout,"mate%12ld      and",
3c    i);
9a   fprintf(stdout,"%12ld\n",
20    mate[i]);
14  }
ca  fprintf(stdout,"\n\nnumber of unmatch"
93   "ed vertices is -> %12ld\n", expo);
f1 }
||
26 main(int argc, char ** argv) {
d4  infile();
6a  match();
c2  outfile();
0b }
|| =============== gen_max_match.cc ===============
|| ==================  General max matching =======
||                                ================
33 struct gmm {
91  int n; vi X,Y,q,f,v,h;
b7  vector<vi> a;
4c  gmm(int n):n(++n),a(n,vi(n)),f(n),h(n){}
f5  void add(int x,int y){ a[x][y]=a[y][x]=X.size()
||                            +n,X.pb(x),Y.pb(y); }
0e  void lsub(int x,int e,int r)
6b   { for(;x!=r;x=h[v[f[x]]])v[x]=e,h[x]=r,q.pb(x)
||                                                ; }
84  void rematch(int x,int y){
7d   int z=f[x]; f[x]=y;
78   if(f[z]!=x) return;
d1   if(v[x]<n) rematch(f[z]=v[x],z);
a9   else z=v[x]−n,rematch(X[z],Y[z]),rematch(Y[z],
||                                             X[z]);
40  }
c3  void dolabel(int x,int y){
91   int r=h[x],s=h[y],e=a[x][y];
7e   if(r==s) return;
18   v[s]=−e;
f3   do if(v[r]=−e,s)swap(s,r);
69   while(v[r=h[v[f[r]]]]+e);
43   lsub(h[x],e,r),lsub(h[y],e,r);
2a   REP(i,q.size())if(v[h[q[i]]]>0)h[q[i]]=r;
04  }
c7  vector<pii> solve(){
9c   FOR(i,1,n)if(!f[i]){
51    v=vi(n,−1),q=vi(1,i),v[i]=0;
c7    REP(j,q.size())
6c     for(int x=q[j],y=1;y<n;++y)if(a[x][y]){
a6      if(!f[y] && y!=i)rematch(f[y]=x,y),j=y=n;
c7      else if(v[y]>=0)dolabel(x,y);
dd      else if(v[f[y]]<0)v[f[y]]=x,h[f[y]]=y,q.pb(
||                                             f[y]);
53     }
1f   }
ce   vector<pii> res;
0e   FOR(i,1,n)if(f[i]>i)res.pb(pii(i,f[i]));
93   return res;
e1  }
5e };
|| =================== geo2d.h ====================
d2 #include <algorithm>
9f #include <assert.h>
84 #include <complex>
86 #include <math.h>
e5 #include <stdio.h>
f2 #include <vector>
e0 using namespace std;
||
|| //////// MACROS //////////
0a #define FR(i, a, b) \
8f  for(int i=(a); i<(b); i++)
c8 #define FOR(i, n) FR(i, 0, n)
87 #define SZ(c) (int((c).size()))
c5 #define BEND(c) (c).begin(), (c).end()
68 #define PB push_back
||
|| //////// TYPES //////////
7c typedef long double T;
19 typedef long double ANGT;
2b typedef complex<T> point;
11 typedef vector<point> poly;
||
|| //////// POINTS //////////
|| //////////////////////////
ca #define X real
5d #define Y imag
f7 T dot(point p, point q) {
ef  return X(conj(p)*q);
62 }
06 T cross(point p, point q) {
1c  return Y(conj(p)*q);
1d }
||
|| //////// LINES (ax+by+c) //////////
|| ///////////////////////////////////
12 struct line {
0c  T a,b,c;
37  line () { a = b = c = 0; }
4c  line(T d, T e, T f) { a=d; b=e; c=f; }
07 };
||
88 line axb(T a, T b) {
e2  return line(−a, 1, b);
98 };
||
a4 line thru(point a, point b) {
53  return line(Y(b−a), X(a−b),
9a   cross(a,b−a));
91 }
||
65 line bi(point a, point b) {
a6  return line(2*X(b−a), 2*Y(b−a),
b9   norm(b)−norm(a));
00 }
||
d2 bool isct(line x, line y, point &p) {
d5  T det = x.a*y.b − x.b*y.a;
f5  if (det == 0) return false;
f0  p = point((−x.b*y.c + x.c*y.b)/det,
73   (x.a*y.c − x.c*y.a)/det);
8d  return true;
```

```
e1 }
||
|| //////// LINES (a+(b−a)*t) //////////
|| /////////////////////////////////////
|| Projection of a point onto a line.
|| Returns the projection’s parameter.
85 T projline(point p, point a, point b) {
1d  return dot(p−a, b−a) / norm(b−a);
45 }
||
|| Point to line distance.
4c T pointline(point p, point a, point b) {
8e  return fabs(cross(p−a, b−a))/abs(b−a);
29 }
||
|| Line−line intersection.
|| Finds isect, at, bt such that
|| isect = a1+(a2−a1)*at
||  = b1+(b2−b1)*bt.
|| Returns true iff they intersect.
e5 bool lineline(point a1, point a2,
72  point b1, point b2, point &isect,
6f  T *at=0, T *bt=0) {
38  T d = cross(a2−a1, b2−b1);
e5  if(d == 0) return false;
12  T t = cross(b2−b1, a1−b1) / d;
fd  isect = a1 + (a2−a1)*t;
b9  if(at) *at = t;
74  if(bt) *bt = cross(a2−a1, a1−b1) / d;
5b  return true;
9f }
||
|| //////// CIRCLES //////////
|| ///////////////////////////
|| Circle−line intersection.
|| Equation of the circle is |x−c| = r.
|| Equation of the line is a1+(a2−a1)*t.
|| Parmeters of the intersection points
|| are stored in t1, t2. t1 <= t2.
|| Returns true iff they intersect.
15 bool circline(point c, T r, point a1,
13  point a2, T &t1, T &t2) {
7c  T t = projline(c, a1, a2);
a3  T d = pointline(c, a1, a2);
16  T q = r*r − d*d;
c7  if(q < 0) return false;
fb  T delta = sqrt(q) / abs(a2−a1);
b3  t1 = t − delta; t2 = t + delta;
2e  return true;
ec }
||
|| Circle−circle intersection.
|| The circles are at (0,0) and (d,0).
|| Sets x,y so that the intersections
|| are (x,y) and (x,−y).
|| Returns true iff they intersect.
|| Note that it will return false
|| if one is contained in the other.
1c bool circcirc(T r1, T r2, T d,
75  T &x, T &y) {
4f  if(d > r1+r2||d==0) return false;
0a  x = (d*d−r2*r2+r1*r1)/(2*d);
26  y = sqrt(r1*r1−x*x);
8f  return y==y;
a1 }
||
|| Circle−circle intersection area.
a4 T circcircarea(T r1, T r2, T d) {
e5  T x, y;
34  if(!circcirc(r1, r2, d, x, y))
82   return (d>r1+r2) ? 0
41    : M_PI*min(r1,r2)*min(r1,r2);
d3  T a1 = atan2(y,x)*r1*r1 − y*x;
c9  T a2 = atan2(y,(d−x))*r2*r2 − y*(d−x);
36  return a1 + a2;
44 }
||
|| Circle−point tangent.
|| Given a point p and a circle with
|| radius r centered at c, this function
|| will return a point on the circle
|| such that a line drawn through this
|| point and p will be tangent to the
|| circle. Set m to +− 1 to choose which
|| point (there are two) to recover.
1a point tanpc(point p,point c,T r, T m) {
82  T h, phi, d;
2b  h = abs(c−p);
1a  phi = m*asin(r/h);
6b  d = h*cos(phi);
c0  return (c−p)/h*polar(T(1),phi)*d+p;
bb }
||
|| Circle−circle tangent.
|| Given two circles at c1 and c2 with
```

```
|| radii r1 and r2, get two points de−
|| fining a line tangent to both circles
|| m=1 yields the tangent line on the
|| right as you face c2 from c1, m=−1
|| yields the tangent line on the left.
|| n=1 yields one of the lines forming
|| an "=", n=−1 yields one of the lines
|| forming an "X".
c0 void tancc(point c1, T r1, point c2,
d4  T r2, point &t1, point &t2,
fa  T m, T n) {
83  T h,d,f,phi,theta;
76  if (n==1 && r2 > r1)
bc   return tancc(c2,r2,c1,r1,t2,t1,−m,n);
21  h = abs(c1−c2);
18  phi = asin((r1−n*r2)/h);
ea  d = h * cos(phi);
be  theta = n * atan(r2/d);
1e  f = sqrt(r2*r2+d*d);
af  t1=(c1−c2)*polar(T(1),m*(phi+theta))
ea   /h*f + c2;
df  t2=(c2−t1)*polar(T(1),−m*theta)
07   /f*d + t1;
8d }
||
|| //////// POLYGONS //////////
|| ////////////////////////////
|| Double the signed area of a polygon.
|| Counterclockwise is positive area.
97 T s2area(const poly &p) {
d6  T ret = 0;
d9  FOR(i, SZ(p)−1)
17   ret += cross(p[i]−p[0],p[i+1]−p[0]);
95  return ret;
89 }
|| Actual area of a polygon.
f4 T area(const poly &p) {
0f  return fabsl(s2area(p))/2;
85 }
||
|| Remove collinear points, make shit
|| counterclockwise, add first point
|| to end.
e0 void cleanpoly(poly &p, T eps=1e−9) {
0e  p.PB(p.at(0));
0c  int j = 1;
f1  FR(i, 1, SZ(p)−1)
09   if(fabs(cross(p[i]−p[i−1],
cb    p[i+1]−p[i−1]))>=eps)
c2    p[j++] = p[i];
1e  p[j++] = p[0];
3d  p.resize(j);
57  if(s2area(p) < 0) reverse(BEND(p));
6f }
||
|| Centre of gravity of a polygon.
76 point cg(poly &p) {
1d  int n=p.size(); T a, b=0;
21  point c;
ed  FOR(i,n) {
e9   int ii=(i+1)%n;
db   a = cross(p[i]−p[0], p[ii]−p[0]);
d0   b += a;
2d   c += a*(p[0]+p[i]+p[ii]);
64  }
30  return c/b/T(3);
a4 }
||
|| Point in line segment?
e2 int pnseg(point a, point b, point p) {
a8  return cross(a−p,b−p)==0
bd   && norm(a−p)+norm(b−p)<=norm(a−b);
c8 }
||
|| Point in perimeter of polygon?
7c int pnperim(poly &p, point x) {
ad  for (int i=0, j=p.size()−1;
cc   i < SZ(p); j = i++) {
19   if (pnseg(p[i],p[j],x)) return 1;
85  }
08  return 0;
d5 }
||
|| point in polygon − indeterminate for
|| points on perimeter guaranteed stable
|| for integer or floating point
58 int pnpoly(poly &p, point x) {
45  int i, j, c = 0;
54  for(i=0,j=SZ(p)−1; i<SZ(p); j=i++) {
d3   if (((Y(p[i])<=Y(x) && Y(x)<Y(p[j]))
31     || (Y(p[j])<=Y(x) && Y(x)<Y(p[i])))
de    && X(x) < X(p[j]−p[i]) * Y(x−p[i])
a7     / Y(p[j] − p[i]) + X(p[i]))
96     c = !c;
e1  }
```

```
76  return c;
36 }
||
|| TODO GOOD NOW?
|| Polygon−line intersection.
|| Parameters of the intersecting
|| intervals are added to "in".
|| Note that "in" is neither cleared
|| beforehand nor sorted afterwards.
|| p must be cleaned by cleanpoly!
97 typedef pair<T, T> ptt;
25 void polyline(const poly &p, point a1,
f9  point a2, vector<ptt> &in,
1e  T eps=1e−11) {
b4  vector<pair<T,int> > xs;
49  point isect; T at, pt;
99  FOR(i, SZ(p)−1) {
4e   if(pointline(p[i], a1, a2) < eps)
64    continue;
b3   if(pointline(p[i+1],a1,a2) > eps) {
      // Easy case.
b8    if(lineline(a1, a2, p[i], p[i+1],
1e     isect, &at, &pt))
85     if(pt>0.0 && pt<1.0)
2b      xs.PB(pair<T,int>(at,0));
49   } else {
02    point a = p[i];
0d    point b = p[i+1];
58    at = projline(b, a1, a2);
10    point c = p[(i+2)%SZ(p)];
24    if(pointline(c, a1, a2) >= eps) {
       // Harder case, B on the line.
34     T s1 = cross(a−a1, a2−a1);
e1     T s2 = cross(c−a1, a2−a1);
56     if(s1*s2>0)
b3      xs.PB(pair<T,int>(at,0));
99     xs.PB(pair<T,int>(at,0));
80    } else {
       // Hardest case, B and C are
       // both on the line.
       // D can’t be since the poly has
       // been cleaned.
bb     pt = projline(c, a1, a2);
91     point d = p[(i+3)%SZ(p)];
73     T s1 = cross(a−a1, a2−a1);
fc     T s2 = cross(d−a1, a2−a1);
58     if(s1*s2<0)
c2      xs.PB(pair<T,int>(at,0));
15     xs.PB(pair<T,int>(at,1));
ec     xs.PB(pair<T,int>(pt,1));
0a    }
65   }
e6  }
    // Find all intersecting intervals.
8e  sort(BEND(xs));
5c  T last = 0;
83  int state = 0;
6a  FOR(i, SZ(xs)) {
ac   if(!xs[i].second) {
a4    if(state)
e8     in.PB(ptt(last, xs[i].first));
d2    else
9c     last = xs[i].first;
b8    state = !state;
07   } else {
76    if(!state) {
e9     in.PB(ptt(xs[i].first,
8c      xs[i+1].first));
60     i++;
9e    }
59   }
ff  }
02 }
||
|| //////// OLD //////////
|| ///////////////////////
5d point circle(point p, point q, point r){
dd  point ret;
d1  isct(bi(p,q), bi(q,r), ret);
64  return ret;
66 }
||
c1 T sdistance(point p, line q) {
70  return (q.a*X(p) + q.b*Y(p) − q.c)
49   / sqrt(q.a*q.a + q.b*q.b);
d7 }
||
e8 T dist(point p, line q) {
15  return fabs(sdistance(p,q));
a5 }
||
40 point proj(point p, line q) {
f8  T normd = (q.a*X(p) + q.b*Y(p) − q.c)
a9   / (q.a*q.a + q.b*q.b);
0e  return point(X(p) − q.a*normd,
30   Y(p) − q.b*normd);
0c }
||
|| returns point r such that p−>q−>r is
|| a right turn of d degrees p−>q and
|| q−>r have equal magnitude special
|| cases for 0, 90, 180, −90 exact
e7 point turn(point p, point q, ANGT d) {
bc  T c = d==0 ? 1 : d==90 ? 0:d==180 ?
e5   −1 : d==−90? 0 : cos (M_PI*d/180);
af  T s = d==0 ? 0 : d==90 ? 1:d==180 ?
f7   0 : d==−90?−1 : sin (M_PI*d/180);
e9  return q + (q−p)*point(c,−s);
42 }
||
|| unsigned angle of p−>q−>r
7f ANGT angle(point p, point q, point r) {
bb  T a = dot(q−p,r−q)/abs(q−p)/abs(r−q);
58  if (fabsl(a)>1) a/=fabsl(a);
34  return 180/M_PI*acosl(a);
c4 }
||
|| clockwise signed angle
a7 ANGT sangle(point p, point q, point r) {
9c  ANGT a = angle(p,q,r);
84  return a * (cross(q−p,r−q) <= 0 ?1:−1);
e1 }
||
|| Do two line segments p1<−>p2 and
|| p3<−>p4 cross?  exact for ints.
|| false if they overlap or abut
|| (as opposed to strictly crossing)
25 int strictcross(point p1, point p2,
27  point p3, point p4) {
8e  T a = cross(p1−p2,p3−p2)
a4   * cross(p1−p2,p4−p2),
86   c = cross(p3−p4,p1−p4)
b3    * cross(p3−p4,p2−p4);
40  return a < 0 && c < 0;
17 }
||
|| nonstrict crossing
|| true if they overlap or abut
df int cross(point p1, point p2,
10  point p3, point p4) {
65  T a = cross(p1−p2,p3−p2)
1a   * cross(p1−p2,p4−p2),
1d   c = cross(p3−p4,p1−p4)
0a    * cross(p3−p4,p2−p4);
06  if (a == 0 && c == 0)
b6   return dot(p1−p3,p1−p4) <= 0
ce    || dot(p2−p3,p2−p4) <= 0
e7    || dot(p3−p1,p3−p2) <= 0
e1    || dot(p4−p1,p4−p2) <= 0;
24  return a <= 0 && c <= 0;
b4 }
||
|| −−−convex hull: naive n^2 algorithm−−−
|| p is a set of n points.  returns q
|| with the hull points in order
d5 poly hull(poly &p) {
38  int i,j,c,d;
df  T t,h,hh;
ab  poly q(0);
||
bf  for (c=i=0;i<SZ(p);i++)
3f   if(Y(p[i])<Y(p[c])||Y(p[i])==Y(p[c])
0e    && X(p[i])>X(p[c])) c=i; //corner
84  for (h=M_PI;h>−4;) {
32   q.PB(p[c]);
c6   for (hh=−4,d=0,j=0;j<SZ(p);j++) {
61    if (p[c] == p[j] ||
f1     (t=arg(p[j]−p[c]))>h+.1)
9d     continue;
3a    if (t > hh+1e−7 || t > hh−1e−7
00     && abs(p[j]−p[c]) >
6d      abs(p[d]−p[c])) {
ed      d=j; hh=t;
da    }
3b   }
c4   h = hh; c = d;
a3  }
4c  q.pop_back();
d9  return q;
db }
||
|| rotate (x,y) about (xc,yc) ccw by t.
7f void rotate(T x, T y, T xc, T yc,
86  ANGT theta, T *nx, T *ny) {
45  point p(x,y), c(xc, yc), r;
ef  r = (p−c)*polar(T(1),theta) + c;
08  *nx = r.real();
81  *ny = r.imag();
51 }
||
b5 ANGT constrainAngle(ANGT angle,
b9  ANGT start) {
29  while(angle<start) angle+= 2*M_PI;
3a  while(angle>=start+2*M_PI)
7f   angle−=2*M_PI;
84  return angle;
c1 }
|| ================== geomc.h ===================
02 #include <math.h>
||
|| bi − bisector between 2 points in ax + by = c
||                                        format
||
20 void bi(double x1, double y1, double x2, double
||                                           y2,
95  double *a, double *b, double *c) {
98  *a = 2*(x2−x1);
18  *b = 2*(y2−y1);
76  *c = x2*x2 + y2*y2 − x1*x1 − y1*y1;
36 }
||
|| isct − intersection of 2 lines in ax+by=c format
||                          . return 0 if undefined
||
ef int isct(double a, double b, double c, double aa
||                          , double bb, double cc,
bf  double *x, double *y) {
38  double det = a*bb − b*aa;
c9  if (fabs(det) < 1e−10) return 0;
e1  *x = (−b*cc + c*bb)/det;
f0  *y = (a*cc − c*aa)/det;
ea  return 1;
3a }
||
|| circle − centre of circumscribing circle on 3
||                             pts.  0 if undefined
||
f9 int circle(double x1, double y1, double x2,
||                                      double y2,
fa  double x3, double y3, double *x, double *y) {
57  double a1,b1,c1,a2,b2,c2;
22  bi(x1,y1,x2,y2,&a1,&b1,&c1);
bc  bi(x2,y2,x3,y3,&a2,&b2,&c2);
9a  return isct(a1,b1,c1,a2,b2,c2,x,y);
e1 }
||
|| point in polygon http://www.ecse.rpi.edu/
||                   Homepages/wrf/geom/pnpoly.html
||
||
81 int pnpoly(int npol, float *xp, float *yp, float
||                                     x, float y)
ff {
78  int i, j, c = 0;
a6  for (i = 0, j = npol−1; i < npol; j = i++) {
64   if ((((yp[i]<=y) && (y<yp[j])) ||
7e    ((yp[j]<=y) && (y<yp[i]))) &&
ab    (x < (xp[j] − xp[i]) * (y − yp[i]) / (yp[j] −
||                               yp[i]) + xp[i]))
||
04    c = !c;
7d  }
e0  return c;
d2 }
||
|| distance of point (x,y,z) to plane (ax+by+cz=d)
||
94 double distpointplane(double x, double y, double
||                                              z,
57  double a, double b, double c, double d) {
04  return fabs(a*x + b*y + c*z − d)/sqrt(a*a + b*b
||                                        + c*c);
e5 }
||
|| distance of point(x,y) to line (ax+by=c)
||
03 double distpointline(double x, double y, double
||                        a, double b, double c) {
64  return fabs(a*x + b*y − c)/sqrt(a*a + b*b);
3e }
||
|| line (ax+by=c) thru 2 points (x1,y1), (x2,y2)
||
c7 void linepoints(double x1, double y1, double x2,
||                                      double y2,
e0  double *a, double *b, double *c) {
5e  *a = y2 − y1;
6b  *b = x1 − x2;
5d  *c = *a * x1 + *b * y1;
3f }
||
|| plane (ax+by+cz=d) thru 3 points (x1,y1), (x2,y2
||                                      ), (x3,y3)
||
33 void planepoints(double x1, double y1, double z1
||                                               ,
d3  double x2, double y2, double z2,
c0  double x3, double y3, double z3,
d7  double *a, double *b, double *c, double *d) {
86  *a = (y1−y3)*(z2−z3) − (y2−y3)*(z1−z3);
a9  *b = (z1−z3)*(x2−x3) − (x1−x3)*(z2−z3);
fd  *c = (x1−x3)*(y2−y3) − (y1−y3)*(x2−x3);
c4  *d = *a*x1 + *b*y1 + *c*z1;
b3 }
|| ================= geometry.h =================
|| Quick geometry tools.
|| Note that circles are (radius,centre) so that
||              they may easily be sorted by radius
|| Algorithms that are safe for use with complex<
||                      int> are marked as being so
|| a*conj(b) is used instead of a/b in case of
||                                    complex<int>
|| Author:  Ralph Furmaniak, Fall 2003.  Still
||                       needs battlefield testing.
||
47 #include <iostream>
fa #include <complex>
26 #include <algorithm>
24 #include <utility>
33 #include <vector>
25 #include <numeric>
10 #include <queue>
98 #include <valarray>
62 using namespace std;
||
|| circles are (radius,centre)
9b typedef complex<double> point;
96 typedef pair<double,point> circle;
24 typedef valarray<point> vpoly;
6b typedef vector<point> poly;
||
|| Common code to get the lengths of sides.
||                           Important for later on
d0 #define sides(a,b,c) double a=abs(C−B),b=abs(C−A
||                                  ),c=abs(A−B)
|| Lars’ define
37 #define fu(i,n) for(int i=0; i<(n); i++)
||
   //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−//
   //    SECTION 1: DISTANCES,INTERSECTIONS   //
|| −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−//
||
|| Distance from point p to line ab
2a double distToLine( point p, point a, point b) {
cb  return abs( ((p−a)/(b−a)*abs(b−a)).imag() );
f3 }
||
|| Distance from point p to line segment ab
08 double distToSegment( point p, point a, point b)
||                                               {
bc  point x = (p−a)/(b−a);
62  if(x.real()>0 && x.real()<1)
1f   return abs(x.imag()*(b−a));
53  return min( abs(p−a), abs(p−b) );
ef }
||
|| Intersection of two lines each through two
||                                          points
|| Pre: a1!=a2 && b1!=b2
||  the lines intersect
||  the lines are not parallel
52 point intersection(point a1, point a2, point b1,
||                                     point b2) {
57  double u = (conj(b2−b1)*(a1−b1)).imag() / (conj
||                         (b2−b1)*(a1−a2)).imag();
47  return a1+u*(a2−a1);
87 }
||
|| Do two line segments intersect
|| Safe for complex<int>
90 bool doesIntersect(point a1, point a2, point b1,
||                                     point b2) {
   // This first "if" is a test for parallel line
||                              segments that may or
   //     may not intersect
   // If you do not need this, do not bother
||                                   typing it in.
24  if(((a2−a1)*conj(b2−b1)).imag()==0) {
51   point d=conj(a2−a1);
48   a1*=d; a2*=d; b1*=d; b2*=d;
ea   if(a1.imag()!=b1.imag()) return 0;
f9   if((a1−a2).real()>0) swap(a1,a2);
e0   if((b1−b2).real()>0) swap(b1,b2);
1b   return max(a1.real(),b1.real())<=min(a2.real()
||                                  ,b2.real());
5d  }
   // Here is the main logic
66  return ((b1−a1)*conj(b1−b2)).imag() * ((b1−a2)*
||                       conj(b1−b2)).imag() <= 0
01   && ((a1−b1)*conj(a1−a2)).imag() * ((a1−b2)*
||                        conj(a1−a2)).imag() <= 0;
1d }
||
|| Is point p on line segment ab?
|| Safe for complex<int>
|| Pick the epsilon wisely
5c bool onSegment(point p, point a, point b) {
99  point x=(p−a)*conj(b−a);
8b  return abs(x.imag())<1e−8 && x.real()>=0 && x.
||                           real()<=norm(b−a);
01 }
||
|| Is point p inside circle c?
|| Safe for complex<int>
5e bool inside( point p, circle& c ) {
22  return norm(p−c.second) <= c.first * c.first;
c4 }
||
||
   //−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−//
   //    SECTION 2: SPECIAL POINTS   //
|| −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−//
||
|| centre of circumcircle of three points
46 point circumcentre( point A, point B, point C) {
ee  double a=norm(C−B),b=norm(C−A),c=norm(A−B);
12  if(a==0) return (A+B)*0.5;
2f  if(b==0) return (A+B)*0.5;
e7  if(c==0) return (A+C)*0.5;
1b  double aa=a*(b+c−a), bb=b*(c+a−b), cc=c*(a+b−c)
||                                               ;
42  return (A*aa + B*bb + C*cc)/(aa+bb+cc);
9f }
||
|| (radius,centre) of circumcircle
ac circle circumcircle( point A, point B, point C)
||                                               {
d4  double a=norm(C−B),b=norm(C−A),c=norm(A−B);
02  double aa=a*(b+c−a), bb=b*(c+a−b), cc=c*(a+b−c)
||                                               ;
23  point centre = (A*aa + B*bb + C*cc)/(aa+bb+cc);
8b  if(a==0) centre=(A+B)*0.5;
4b  if(b==0) centre=(A+B)*0.5;
ca  if(c==0) centre=(A+C)*0.5;
9b  return circle(abs(centre−A),centre);
e1 }
||
|| Centre of incircle
c2 complex<double> incentre( point A, point B,
||                                    point C) {
c5  sides(a,b,c);
95  if(a+b+c==0) return A;
bb  return (a*A+b*B+c*C)/(a+b+c);
e0 }
||
|| (centre,radius) of incircle
6e circle incircle( point A, point B, point C) {
13  sides(a,b,c);
1b  point centre = (a*A+b*B+c*C)/(a+b+c);
38  return circle(abs(centre−A),centre);
2c }
||
|| orthocentre of three points
21 point orthocentre( point A, point B, point C) {
b4  sides(a,b,c);
45  double aa = (a*a+b*b−c*c)*(a*a−b*b+c*c),
7b   bb = (−a*a+b*b+c*c)*(a*a+b*b−c*c),
ef   cc = (a*a−b*b+c*c)*(−a*a+b*b+c*c);
ca  if(a*a+b*b+c*c==0) return A;
1c  return (A*aa+B*bb+C*cc)/(aa+bb+cc);
3d }
||
|| centroid of points
2b point centroid( const vpoly& p) {
64  return p.sum()/(double)p.size();
6e }
||
|| Smallest circle containing points
|| Requires circumcircle code from above
|| Naive n^3 algorithm
||
85 circle min(const circle& A, const circle& B) {
||                    return A.first<B.first?A:B; }
53 circle max(const circle& A, const circle& B) {
||                    return A.first>B.first?A:B; }
a3 circle circumcircle( poly& p ) {
41  circle ret=circle(1e100,0.0);
ed  fu(i,p.size()) fu(j,i+1) {
19   circle cur;
1d   fu(k,p.size())
79    cur=max(cur,circumcircle(p[i],p[j],p[k]));
87   ret=min(ret,cur);
71   }
0c  return ret;
35 }
```

```
||
|| Reflects p across the line through (a,b)
6d point reflect( point p, point a, point b ) {
02  return b + conj((p-b)/(a-b))*(a-b);
0d }
||
||
   //---------------------------//
   //    SECTION 3: POLYGONS    //
|| ---------------------------//
||
|| Unsigned area.  May not do what you expect for
||                       self-intersecting polygons
|| Safe for complex<int>
87 double area( poly& p ) {
cc  double ret=0;
6e  fu(i,p.size())
aa   ret += (p[i]* conj(p[(i+1)%p.size()])).imag();
fd  return abs(ret)*0.5;
0a }
||
|| Is the point inside the polygon
|| Algorithm modified from the book,
|| from http://www.ecse.rpi.edu/Homepages/wrf/geom/
||                                      pnpoly.html
|| Safe for complex<int>
|| 1=inside, 0=on, -1=outside (beware of epsilons
||                           for points on polygon)
90 int pointInsidePolygon( point p, const poly& P )
||                                                {
04  int c = -1;
cd  for(int i=0, j=P.size()-1; i<P.size(); j=i++) {
24   if(onSegment(p,P[i],P[j])) return 0;
67   if((((P[i]-p).imag()<=0 && (p-P[j]).imag()<0)
||                                               ||
70     ((P[j]-p).imag()<=0 && (p-P[i]).imag()<0)) &&
32     (p-P[i]).real() < (P[j]-P[i]).real()*
1c     (p-P[i]).imag()/(P[j]-P[i]).imag())
0d      c=-c;
15  }
a8  return c;
90 }
||
|| Is the polygon convex?
9e bool isConvex( poly& p) {
59  for(int i=1; i<p.size(); i++)
f1   if( (p[i-1]/(p[(i+1)%p.size()]-p[i])).imag() *
09     (p[p.size()-1]/(p[1]-p[0])).imag() < 0)
b2     return false;
20  return true;
f4 }
||
|| Intersect a line through a and b with a polygon
|| Returns the vector of points represented by
||                          distance along a and b.
||  eg:  a=0, b=1, (a+b)/2=0.5
22 vector<double> clipLine( point a, point b, poly&
||                                             p) {
50  vector<double> ret;
7c  for(int i=0, j=p.size()-1; i<p.size(); j=i++)
d1   if((p[i]/(b-a)).imag()<=(a/(b-a)).imag()
c5     && (a/(b-a)).imag()<(p[j]/(b-a)).imag() ||
6d     (p[j]/(b-a)).imag()<=(a/(b-a)).imag()
10     && (a/(b-a)).imag()<(p[i]/(b-a)).imag())
02     ret.push_back(((p[j]-p[i])/(b-a)).real()*
31     ((a-p[i])/(b-a)).imag()/((p[j]-p[i])/(b-a)).
||                                           imag()
32     +((p[i]-a)/(b-a)).real());
||
7d }
||
|| Cut a polygon into two parts by a line.
|| The first poly is counter-clockwise from the
||                                            line.
|| Works even on concave polygons,
|| but make sure that you know what you're getting
||                                         into it.
|| There may be duplicate points
14 pair<poly,poly> cutPoly( point a, point b, poly&
||                                             p) {
8d  poly p1,p2;
6f  for(int i=0, j=p.size()-1; i<p.size(); j=i++) {
ce   double s1=((p[i]-a)/(b-a)).imag();
65   double s2=((p[j]-a)/(b-a)).imag();
fa   if(s1*s2<=0) {
5c    p1.push_back(intersection(p[i],p[j],a,b));
75    p2.push_back(intersection(p[i],p[j],a,b));
ee   }
28   (s1>0?p1:p2).push_back(p[i]);
3c  }
82  return make_pair(p1,p2);
89 }
||
||
   //------------------------------//
```

```
   //    SECTION 4: CONVEX HULLS   //
|| ------------------------------//
||
|| Naive n^3 convex hull.  Returns pairs of points
||                                   (a,b) a<b that
|| form a segment of the convex hull
|| Dangerous if the convex hull contains three
||                                 collinear points
37 vector< pair<int,int> > slowHull(poly& p) {
2b  vector< pair<int,int> > ret;
61  fu(i,p.size()) fu(j,i) {
c3   int cnt=0;
0b   fu(k,p.size()) if(k!=i && k!=j)
66    k += (((p[k]-p[i])/(p[j]-p[i])).imag() > 0 ?
||                                         1 : -1);
35   if(cnt==p.size()-2 || cnt==2-p.size())
8b    ret.push_back( make_pair(j,i) );
01   }
50  return ret;
ff }
||
|| n^2 convex hull.  Returns the polygon
|| Modified from geom.h in the book
84 poly fasterHull(poly& p) {
b7  poly q;
ed  int j,c=0,d;
01  double t,h,hh;
79  fu(i,p.size())
6d   if((p[i]-p[c]).imag()<0 || (p[i]-p[c]).imag()=
||                                               =0
31   && (p[i]-p[c]).real()>0) c=i;
8b  for(h=M_PI;h>-4;h=hh,c=d) {
4c   q.push_back(p[c]);
49   for(hh=-4,d=0,j=0;j<p.size();j++)
65    if(p[c]!=p[j] && (t=arg(p[j]-p[c])) <= h+.1)
2a    if(t > hh+1e-8 || t>hh-1e-8
7d     && abs(p[j]-p[c]) > abs(p[d]-p[c])) {
77     d=j;
69     hh=t;
de    }
3b  }
80  q.pop_back();
b3  return q;
b8 }
||
||
||
74 bool cmpArg(const point& A, const point& B) {
cc  return arg(A)<arg(B) || (arg(A)==arg(B) && abs(
||                                      A)<abs(B));
8b }
cd poly fastestHull(poly& p) {
b4  deque<point> hull(2);
||
b5  fu(i,p.size())
73   if((p[i]-p[0]).imag()<0 || (p[i]-p[0]).imag()=
||                      =0 && (p[i]-p[0]).real()<0)
f0    swap(p[0],p[i]);
d1  fu(i,p.size()+-1) p[i+1]-=p[0];
71  sort(&p[1],&p[p.size()],cmpArg);
93  fu(i,p.size()+-1) p[i+1]+=p[0];
08  hull[0]=p[1];
85  hull[1]=p[0];
a2  for(int i=2; i<=p.size(); i++) {
a4   while( hull.size()>=2 &&
d5   ((p[i%p.size()]-hull[0])/(hull[1]-hull[0])).
||                                     imag() >= 0)
7a    hull.pop_front();
d3   hull.push_front(p[i%p.size()]);
d3  }
3e  return poly(hull.begin(),hull.end());
e4 }
||
|| ================ graphpaper.ps =================
63 %!PS-Adobe-3.0
91 .85 setgray
6b 0 9 792 { dup 0 moveto dup 792 lineto
fd  stroke } for
fe 0 9 792 { dup 0 exch moveto dup 792 exch
a4  lineto stroke } for
||
79 .7 setgray
c4 0 18 792 { dup 0 moveto dup 792 lineto
d6  stroke } for
c8 0 18 792 { dup 0 exch moveto dup 792
e7  exch lineto stroke } for
||
79 .5 setgray
c7 0 36 792 { dup 0 moveto dup 792 lineto
d6  stroke } for
0a 0 36 792 { dup 0 exch moveto dup 792
fb  exch lineto stroke } for showpage
|| ================== ham.cpp ===================
||  Sample use of "ham.h"  for hamiltonial cycle
||  Reads "colorit.in" input file
```

```
||
79 #include <stdio.h>
5e #include <string.h>
6e #include <stdlib.h>
5d #include "ham.h"
||
9b int i,j,k,m,n,T,N,M;
||
34 char name[300][12];
5c int nn,t;
56 char x[12],y[12];
ce int ix,iy;
||
ab int color[300];
||
2b typedef int (*qsortf)(const void*,const void*);
||
d2 int main() {
d8  scanf("%d",&T);
64  for (t=1;t<=T;t++) {
cc   scanf("%d%d",&N,&M);
cc   if (t != 1) printf("\n");
23   printf("Case #%d size %d %d\n",t,N,M);
c5   fflush(stdout);
86   reset(N);
8c   for (i=0;i<N;i++) scanf(" %s",name[i]);
b5   qsort(name,N,12,(qsortf)strcmp);
eb   for (i=0;i<M;i++) {
0c    scanf(" %s %s", x, y);
b4    ix = ((char*)bsearch(x,name,N,12,(qsortf)
||                                          strcmp)
49      - (char *)name)/12;
15    iy = ((char*)bsearch(y,name,N,12,(qsortf)
||                                          strcmp)
ec      - (char *)name)/12;
e9    edge(ix,iy);
7a    edge(iy,ix);
b6   }
7f   if (!ham(nv)) printf("no solution\n");
9b   else {
31    for (i=0;i<=nv;i++) printf("%d ",hamcycle[i])
||                         ;  // repeats start vert
bd    printf("\n");
a0   }
30  }
c4  return 0;
7b }
|| =================== ham.h ====================
|| Hamiltonian cycle in a general graph
||
|| The heuristic in most cases nails it or shows
||                            there is no solution.
|| If it doesn't do this, it runs a *long* time
||
|| You might want to bail out and say "no solution"
||                                after a couple of
|| thousand calls
||
60 #define MAXV 500
8a #define MAXC MAXV
ad #include <string.h>
60 #include <assert.h>
||
50 struct vv {
4a  short nadj, adj[MAXV], used, dq;
bb } v[MAXV];
||
fe int nv, miter, iter;
||
0e void reset(int n) {
68  int i,j;
1f  miter= 10; iter = 0;
34  nv = n;
83  for (i=0;i<n;i++) {
6d   v[i].nadj = v[i].dq = v[i].used = 0;
16  }
35 }
||
ca void edge(int i, int j) {  // user MUST to do
||                          edge(i,j) and edge(j,i)
59  v[i].adj[v[i].nadj++] = j;
77 }
||
18 int cmp(const void *aa, const void *bb){
fe  const short *a = (const short *)aa;
4d  const short *b = (const short *)bb;
2e  return (v[*a].nadj-v[*a].dq-v[*a].used) -
16   (v[*b].nadj-v[*b].dq-v[*b].used);
5d }
||
e7 int hamcycle[MAXV];
||
34 int ham(int u){
04  int i,j,k,bv=-1;
2d  if (u == 0) return 1;
```

```
||  if( iter++ > miter+miter) { printf("iter %d\n",
||                               iter);miter=iter;}
31   for (i=0;i<nv;i++) {
13    if (v[i].used == 2) continue;
fe    if (bv<0 || (v[i].nadj-v[i].dq-100*v[i].used)
||                                               <
95     (v[bv].nadj-v[bv].dq-100*v[bv].used)) bv=i;
41   }
2b   if(++v[bv].used==2) for (k=0;k<v[bv].nadj;k++)
||                            v[v[bv].adj[k]].dq++;
ac   qsort(v[bv].adj,v[bv].nadj,sizeof(short),cmp);
cc   for (i=0;i<v[bv].nadj;i++){
78    if (v[j=v[bv].adj[i]].used==2 ||
e1     u>2 && v[bv].used == 2 && v[j].used==1)
||                                        continue;
92    if (++v[j].used == 2) for (k=0;k<v[j].nadj;k+
||                          +) v[v[j].adj[k]].dq++;
e4    if (ham(u-1)) {
35     if (u == 1) hamcycle[0] = j;
52     if (hamcycle[u-1] == j) hamcycle[u] = bv;
4f     else if (hamcycle[0] == j) {
a8      for (k=u;k>0;k--) hamcycle[k] = hamcycle[k-
||                                              1];
4c      hamcycle[0] = bv;
d4     } else {
6f      printf("oops bv %d j %d\n",bv,j);
35      for (k=0;k<nv;k++) printf("v %d nadj %d
||                                used %d dq %d\n",
4f       k,v[k].nadj,v[k].used,v[k].dq);
32      exit(1);
8c     }
55     return 1;
df    }
ff    if (v[j].used-- == 2) for (k=0;k<v[j].nadj;k+
||                          +) v[v[j].adj[k]].dq--;
c2   }
42   if(v[bv].used--==2) for (k=0;k<v[bv].nadj;k++)
||                            v[v[bv].adj[k]].dq--;
26   return 0;
ec }
|| ================= hutucker.cc =================
a5 #include "leftist_heap.h"
0c #include <queue>
||
91 #define MAXN 123456
||
40 #define FOR(i,n) for (int i=0;i<n;i++)
59 #define ll long long
07 #define MP make_pair
||
6f mheap<ll>*heap[MAXN];
13 ll w[MAXN],bes[MAXN],end[MAXN][2],ans;
1e int sid[MAXN][2],rec[MAXN];
fd int done[MAXN],n;
ad priority_queue<pair<ll,int> > q;
||
52 void recalc(int id){
8a  static ll v1[4],tem;
8d  v1[0] = -heap[id]->key;
b7  tem= heap[id]->l ?
46   heap[id]->l->key : -1LL<<60;
3f  if (heap[id]->r)
9e   tem>?=heap[id]->r->key;
0d  v1[0] -= tem;
c8  v1[1] = -heap[id]->key + end[id][0];
7d  v1[2] = -heap[id]->key + end[id][1];
72  v1[3] = end[id][0] + end[id][1];
bc  bes[id] = 1LL<<62;
90  FOR(i,4) if (v1[i] < bes[id]) {
22   bes[id] = v1[i];
7c   rec[id] = i;
26  }
c0  q.push(MP(-bes[id], id));
9b }
||
d2 int main() {
5e  int id;
c3  scanf("%d",&n);
46  FOR(i,n) scanf("%lld",&w[i]);
f2  FOR(i,n-1){
58   heap[i] = new mheap<ll>(-1LL<<60);
76   sid[i][0] = i-1;
0d   sid[i][1] = (i+1==n-1)?-1:i+1;
12   end[i][0] = w[i];
fd   end[i][1] = w[i+1];
40   done[i] = 0;
a4   recalc(i);
70  }
da  ans=0;
0b  FOR(i,n-1) {
69   while (done[id = q.top().second] ||
b1     -bes[id] != q.top().first)
d4     q.pop();
71   q.pop();
9a   ans += bes[id];
a2   FOR(j,2) if (rec[id] & (1<<j)) {
bb    if (sid[id][j] == -1)
d6     end[id][j] = 1LL<<60;
7d    else {
f3     heap[id] = heap[id]->merge(
47      heap[sid[id][j]]);
c0     end[id][j] = end[sid[id][j]][j];
19     done[sid[id][j]]=1;
0e     sid[id][j] = sid[sid[id][j]][j];
9f     if (sid[id][j] != -1)
14      sid[sid[id][j]][1-j] = id;
1d    }
65   }
9c   else heap[id] = heap[id]->pop();
18   heap[id]=heap[id]->insert(-bes[id]);
3c   recalc(id);
f4  }
bb  printf("%lld\n", ans);
68  return 0;
26 }
|| ============= inertia-tensor.cpp =============
02 #include <math.h>
69 #include <stdio.h>
89 #include <stdlib.h>
||
||
30 #define X 0
b7 #define Y 1
8a #define Z 2
||
48 #define MAX_POLYGON_SZ  10
2c #define MAX_VERTS       1000
5e #define MAX_FACES       1000
||
41 #define SQR(x) ((x)*(x))
65 #define CUBE(x) ((x)*(x)*(x))
||
06 typedef struct {
b1  int numVerts;
40  double norm[3];
95  double w;
ec  int verts[MAX_POLYGON_SZ];
2f  struct polyhedron *poly;
41 } FACE;
||
8e typedef struct polyhedron {
7e  int numVerts, numFaces;
04  double verts[MAX_VERTS][3];
fe  FACE faces[MAX_FACES];
71 } POLYHEDRON;
||
75 static int A;   // alpha
5b static int B;   // beta
bd static int C;   // gamma
||
|| projection integrals
1b static double P1, Pa, Pb, Paa, Pab, Pbb, Paaa,
||                               Paab, Pabb, Pbbb;
||
|| face integrals
a1 static double Fa, Fb, Fc, Faa, Fbb, Fcc, Faaa,
||                   Fbbb, Fccc, Faab, Fbbc, Fcca;
||
|| volume integrals
f8 static double T0, T1[3], T2[3], TP[3];
||
||
|| ====================================
|| read in a polyhedron
|| ====================================
||
ca void readPolyhedron(char *name, POLYHEDRON *p)
ed {
90  FILE *fp;
97  char line[200], *c;
0f  int i, j, n;
46  double dx1, dy1, dz1, dx2, dy2, dz2, nx, ny, nz
||                                            , len;
66  FACE *f;
||
||
22  if (!(fp = fopen(name, "r"))) {
3c   printf("i/o error\n");
07   exit(1);
15  }
||
8a  fscanf(fp, "%d", &p->numVerts);
3c  printf("Reading in %d vertices\n", p->numVerts)
||                                               ;
be  for (i = 0; i < p->numVerts; i++)
4e   fscanf(fp, "%lf %lf %lf",
81    &p->verts[i][X], &p->verts[i][Y], &p->verts[i
||                                          ][Z]);
||
e9  fscanf(fp, "%d", &p->numFaces);
```

```
6a  printf("Reading in %d faces\n", p->numFaces);
c8  for (i = 0; i < p->numFaces; i++) {
41   f = &p->faces[i];
ab   f->poly = p;
bf   fscanf(fp, "%d", &f->numVerts);
33   for (j = 0; j < f->numVerts; j++) fscanf(fp, "
||                              %d", &f->verts[j]);
||

    // compute face normal and offset w from first
||                                      3 vertices
1e   dx1 = p->verts[f->verts[1]][X] - p->verts[f->
||                                    verts[0]][X];
24   dy1 = p->verts[f->verts[1]][Y] - p->verts[f->
||                                    verts[0]][Y];
0f   dz1 = p->verts[f->verts[1]][Z] - p->verts[f->
||                                    verts[0]][Z];
42   dx2 = p->verts[f->verts[2]][X] - p->verts[f->
||                                    verts[1]][X];
37   dy2 = p->verts[f->verts[2]][Y] - p->verts[f->
||                                    verts[1]][Y];
22   dz2 = p->verts[f->verts[2]][Z] - p->verts[f->
||                                    verts[1]][Z];
d5   nx = dy1 * dz2 - dy2 * dz1;
b7   ny = dz1 * dx2 - dz2 * dx1;
3b   nz = dx1 * dy2 - dx2 * dy1;
7c   len = sqrt(nx * nx + ny * ny + nz * nz);
20   f->norm[X] = nx / len;
9e   f->norm[Y] = ny / len;
93   f->norm[Z] = nz / len;
0f   f->w = - f->norm[X] * p->verts[f->verts[0]][X]
08    - f->norm[Y] * p->verts[f->verts[0]][Y]
8c    - f->norm[Z] * p->verts[f->verts[0]][Z];
||
7d  }
||
b4  fclose(fp);
||
7d }
||
|| ====================================
|| compute mass properties
|| ====================================
||
|| compute various integrations over projection of
||                                             face
0c void compProjectionIntegrals(FACE *f)
fb {
6c  double a0, a1, da;
13  double b0, b1, db;
e0  double a0_2, a0_3, a0_4, b0_2, b0_3, b0_4;
85  double a1_2, a1_3, b1_2, b1_3;
e5  double C1, Ca, Caa, Caaa, Cb, Cbb, Cbbb;
b0  double Cab, Kab, Caab, Kaab, Cabb, Kabb;
e9  int i;
||
38  P1 = Pa = Pb = Paa = Pab = Pbb = Paaa = Paab =
||                             Pabb = Pbbb = 0.0;
||
40  for (i = 0; i < f->numVerts; i++) {
9c   a0 = f->poly->verts[f->verts[i]][A];
72   b0 = f->poly->verts[f->verts[i]][B];
d3   a1 = f->poly->verts[f->verts[(i+1) % f->
||                                 numVerts]][A];
56   b1 = f->poly->verts[f->verts[(i+1) % f->
||                                 numVerts]][B];
e8   da = a1 - a0;
5a   db = b1 - b0;
9d   a0_2 = a0 * a0; a0_3 = a0_2 * a0; a0_4 = a0_3
||                                          * a0;
ad   b0_2 = b0 * b0; b0_3 = b0_2 * b0; b0_4 = b0_3
||                                          * b0;
e5   a1_2 = a1 * a1; a1_3 = a1_2 * a1;
b0   b1_2 = b1 * b1; b1_3 = b1_2 * b1;
||
5d   C1 = a1 + a0;
d7   Ca = a1*C1 + a0_2; Caa = a1*Ca + a0_3; Caaa =
||                                 a1*Caa + a0_4;
48   Cb = b1*(b1 + b0) + b0_2; Cbb = b1*Cb + b0_3;
||                          Cbbb = b1*Cbb + b0_4;
19   Cab = 3*a1_2 + 2*a1*a0 + a0_2; Kab = a1_2 + 2*
||                                a1*a0 + 3*a0_2;
f7   Caab = a0*Cab + 4*a1_3; Kaab = a1*Kab + 4*a0_3
||                                              ;
39   Cabb = 4*b1_3 + 3*b1_2*b0 + 2*b1*b0_2 + b0_3;
59   Kabb = b1_3 + 2*b1_2*b0 + 3*b1*b0_2 + 4*b0_3;
||
d9   P1 += db*C1;
ad   Pa += db*Ca;
ac   Paa += db*Caa;
08   Paaa += db*Caaa;
d9   Pb += da*Cb;
b3   Pbb += da*Cbb;
0a   Pbbb += da*Cbbb;
c3   Pab += db*(b1*Cab + b0*Kab);
b3   Paab += db*(b1*Caab + b0*Kaab);
```

```
a4   Pabb += da*(a1*Cabb + a0*Kabb);
a0  }
||
93  P1 /= 2.0;
97  Pa /= 6.0;
c8  Paa /= 12.0;
96  Paaa /= 20.0;
57  Pb /= -6.0;
7a  Pbb /= -12.0;
0a  Pbbb /= -20.0;
a3  Pab /= 24.0;
c4  Paab /= 60.0;
e9  Pabb /= -60.0;
7a }
||
ed void compFaceIntegrals(FACE *f)
dc {
66  double *n, w;
fb  double k1, k2, k3, k4;
||
1f  compProjectionIntegrals(f);
||
b8  w = f->w;
86  n = f->norm;
dd  k1 = 1 / n[C]; k2 = k1 * k1; k3 = k2 * k1; k4 =
||                                         k3 * k1;
||
17  Fa = k1 * Pa;
ad  Fb = k1 * Pb;
15  Fc = -k2 * (n[A]*Pa + n[B]*Pb + w*P1);
||
e8  Faa = k1 * Paa;
e7  Fbb = k1 * Pbb;
38  Fcc = k3 * (SQR(n[A])*Paa + 2*n[A]*n[B]*Pab +
||                                    SQR(n[B])*Pbb
ba   + w*(2*(n[A]*Pa + n[B]*Pb) + w*P1));
||
06  Faaa = k1 * Paaa;
5b  Fbbb = k1 * Pbbb;
b2  Fccc = -k4 * (CUBE(n[A])*Paaa + 3*SQR(n[A])*n[B
||                                          ]*Paab
ed   + 3*n[A]*SQR(n[B])*Pabb + CUBE(n[B])*Pbbb
64   + 3*w*(SQR(n[A])*Paa + 2*n[A]*n[B]*Pab + SQR(n
||                                        [B])*Pbb)
07   + w*w*(3*(n[A]*Pa + n[B]*Pb) + w*P1));
||
1e  Faab = k1 * Paab;
0c  Fbbc = -k2 * (n[A]*Pabb + n[B]*Pbbb + w*Pbb);
71  Fcca = k3 * (SQR(n[A])*Paaa + 2*n[A]*n[B]*Paab
||                                  + SQR(n[B])*Pabb
e1   + w*(2*(n[A]*Paa + n[B]*Pab) + w*Pa));
1a }
||
d0 void compVolumeIntegrals(POLYHEDRON *p)
25 {
4a  FACE *f;
5e  double nx, ny, nz;
bf  int i;
||
ba  T0 = T1[X] = T1[Y] = T1[Z]
e8   = T2[X] = T2[Y] = T2[Z]
d2   = TP[X] = TP[Y] = TP[Z] = 0;
||
70  for (i = 0; i < p->numFaces; i++) {
||
4c   f = &p->faces[i];
||
52   nx = fabs(f->norm[X]);
a5   ny = fabs(f->norm[Y]);
4c   nz = fabs(f->norm[Z]);
15   if (nx > ny && nx > nz) C = X;
1f   else C = (ny > nz) ? Y : Z;
2d   A = (C + 1) % 3;
31   B = (A + 1) % 3;
||
0c   compFaceIntegrals(f);
||
68   T0 += f->norm[X] * ((A == X) ? Fa : ((B == X)
||                                     ? Fb : Fc));
||
79   T1[A] += f->norm[A] * Faa;
90   T1[B] += f->norm[B] * Fbb;
31   T1[C] += f->norm[C] * Fcc;
04   T2[A] += f->norm[A] * Faaa;
e2   T2[B] += f->norm[B] * Fbbb;
1e   T2[C] += f->norm[C] * Fccc;
c1   TP[A] += f->norm[A] * Faab;
fa   TP[B] += f->norm[B] * Fbbc;
3d   TP[C] += f->norm[C] * Fcca;
4c  }
||
d6  T1[X] /= 2; T1[Y] /= 2; T1[Z] /= 2;
a8  T2[X] /= 3; T2[Y] /= 3; T2[Z] /= 3;
dc  TP[X] /= 2; TP[Y] /= 2; TP[Z] /= 2;
73 }
```

```
||
||
|| ====================================
|| main
|| ====================================
||
||
03 int main(int argc, char *argv[])
7b {
fc  POLYHEDRON p;
e8  double density, mass;
56  double r[3];            // center of mass
54  double J[3][3];         // inertia tensor
||
c9  if (argc != 2) {
a6   printf("usage:  %s <polyhedron geometry
||                          filename>\n", argv[0]);
16   exit(0);
c5  }
||
82  readPolyhedron(argv[1], &p);
||
21  compVolumeIntegrals(&p);
||
||
a6  printf("\nT1 =   %+20.6f\n\n", T0);
||
13  printf("Tx =   %+20.6f\n", T1[X]);
1a  printf("Ty =   %+20.6f\n", T1[Y]);
77  printf("Tz =   %+20.6f\n\n", T1[Z]);
||
79  printf("Txx =  %+20.6f\n", T2[X]);
f0  printf("Tyy =  %+20.6f\n", T2[Y]);
ec  printf("Tzz =  %+20.6f\n\n", T2[Z]);
||
7b  printf("Txy =  %+20.6f\n", TP[X]);
32  printf("Tyz =  %+20.6f\n", TP[Y]);
2e  printf("Tzx =  %+20.6f\n\n", TP[Z]);
||
ef  density = 1.0;  // assume unit density
||
5a  mass = density * T0;
||
    // compute center of mass
7d  r[X] = T1[X] / T0;
5c  r[Y] = T1[Y] / T0;
08  r[Z] = T1[Z] / T0;
||
    // compute inertia tensor
fc  J[X][X] = density * (T2[Y] + T2[Z]);
3f  J[Y][Y] = density * (T2[Z] + T2[X]);
ee  J[Z][Z] = density * (T2[X] + T2[Y]);
87  J[X][Y] = J[Y][X] = - density * TP[X];
f4  J[Y][Z] = J[Z][Y] = - density * TP[Y];
72  J[Z][X] = J[X][Z] = - density * TP[Z];
||
    // translate inertia tensor to center of mass
e9  J[X][X] -= mass * (r[Y]*r[Y] + r[Z]*r[Z]);
22  J[Y][Y] -= mass * (r[Z]*r[Z] + r[X]*r[X]);
8d  J[Z][Z] -= mass * (r[X]*r[X] + r[Y]*r[Y]);
3d  J[X][Y] = J[Y][X] += mass * r[X] * r[Y];
8d  J[Y][Z] = J[Z][Y] += mass * r[Y] * r[Z];
7c  J[Z][X] = J[X][Z] += mass * r[Z] * r[X];
||
a0  printf("center of mass:  (%+12.6f,%+12.6f,%+12.
||                    6f)\n\n", r[X], r[Y], r[Z]);
||
66  printf("inertia tensor with origin at c.o.m. :\
||                                             n");
36  printf("%+15.6f  %+15.6f  %+15.6f\n", J[X][X],
||                               J[X][Y], J[X][Z]);
a2  printf("%+15.6f  %+15.6f  %+15.6f\n", J[Y][X],
||                               J[Y][Y], J[Y][Z]);
f1  printf("%+15.6f  %+15.6f  %+15.6f\n\n", J[Z][X]
||                             , J[Z][Y], J[Z][Z]);
||
c3  return 0;
fb }
|| ================ inversions.h ================
|| count the number of inversions (that
|| is, i<j such that A[i]>A[j]) in an
|| array.
||
b3 #define ll long long
||
5a ll inver(int *A, int n) {
ac  if (n < 2) return 0;
16  ll ans = inver(A,n/2)
4b   + inver(A+n/2,n-n/2);
ba  int B[n], i=0, j=n/2, a=0;
53  while (i<n/2||j<n)
78   B[a] = A[(i<n/2 ? j<n&&A[j]<A[i] ?
4c    ans+=j-a,j : i : j)++], a++;
60  while (j--) A[j] = B[j];
ae  return ans;
```

```
78 }
|| =================== invert.c ====================
|| Matrix inversion
||
|| MAXN in gauss must be double n, the matrix size
||
|| invert(n,A,AINV) inverts n by n matrix A, result
||                                          in AINV
||  returns 1 on success; returns 0 if singular
||
||  mult (n,A,B,AB) multiplies A*B giving AB
||
8a #include "gauss.h"
||
c1 int invert(int n, double A[][MAXN], double AINV[
||                                      ][MAXN]) {
71  int i,j;
df  double M[MAXN][MAXN], dummy[MAXN];
b5  for (i=0;i<n;i++) for (j=0;j<n;j++) {
ec   M[i][j] = A[i][j];
93   M[i][j+n] = (i == j);
ac   M[i+n][j] = M[i+n][j+n] = 0;
d8  }
31  solve(2*n,2*n,M,dummy);
da  for (i=0;i<n;i++) if (fabs(1−M[i][i]) > 1e−10)
||                                        return 0;
bc  for (i=0;i<n;i++) for (j=0;j<n;j++) AINV[i][j]
||                                     = M[i][n+j];
dc  return 1;
5b }
||
cd void mult(int n, double A[][MAXN], double B[][
||                     MAXN], double AB[][MAXN]) {
dd  int i,j,k;
45  for (i=0;i<n;i++) for (j=0;j<n;j++) {
89   AB[i][j] = 0;
fc   for (k=0;k<n;k++) AB[i][j] += A[i][k] * B[k][j
||                                               ];
82  }
59 }
||
|| sample mainline
||
d7 #include <stdlib.h>
3f #include <stdio.h>
||
17 void pr(char *s, int n, double A[][MAXN]){
6b  int i,j;
0d  printf("%s:\n",s);
4a  for (i=0;i<n;i++){
05   for (j=0;j<n;j++) printf("%12.6f ",A[i][j]);
a7   printf("\n");
28  }
a4 }
||
3f double A[MAXN][MAXN], AINV[MAXN][MAXN], I[MAXN][
||                                           MAXN];
||
94 main(){
8f  int i,j,k,n = 10;
97  for (i=0;i<n;i++) for (j=0;j<n;j++) A[i][j] =
||                                  random()%100000;
61  if (!invert(n,A,AINV)) printf("singular!\n");
d0  else {
a4   mult(n,AINV,A,I);
ba   pr("I",n,I);
50  }
ac  for (i=0;i<n;i++) for (j=0;j<n;j++) A[i][j] =
||                                  random()%100000;
a8  for (i=0;i<n;i++) A[5][i] = A[7][i];
61  if (!invert(n,A,AINV)) printf("singular!\n");
ef  else {
a5   mult(n,AINV,A,I);
a5   pr("I",n,I);
50  }
2f }
|| =================== ip2.h =====================
|| Integer programming − Requires lp.h
||
|| Usage:
||
|| r = ip(m, n, C, X);
||
|| Do simplex() from lp.h but with all
|| variables integer.
||
|| Assumes coefficients (and therefore
|| objective value) are integer.
|| (Where? −Tor)
||
|| Real coefficients work OK, but must
|| remove "nearest integer" code from
|| return statement in ip()
||
fd void doip(int of, int m, int n,
```

```
ed  double C[][MAXN], double X[]) {
97  double z,x;
97  static double XX[MAXN];
16  static int cerow[MAXN], flrow[MAXN];
||
ce  z = simplex(m, n, C, XX);
af  if (z <= −C[of][n]) return;
||
e5  FOR(i,n) {
cc   x = XX[i] + 100*EPS;
ba   if (x−floor(x) > 200*EPS) {
21 #define DOIT(r, f, s)                    \
11  if (!r[i]) {                         \
a0   r[i] = m+1;                        \
a0   FOR(j,n) C[m+1][j] = s (i==j);     \
e0   C[m+1][n] = s f(x);                \
17   doip(of, m+1, n, C, X);            \
e1   r[i] = 0;                          \
06  }                                    \
66  else {                               \
9b   z = C[r[i]][n];                    \
43   C[r[i]][n] = s f(x);               \
97   doip(of, m, n, C, X);              \
a4   C[r[i]][n] = z;                    \
70  }
||
91  DOIT(flrow, floor,)
89  DOIT(cerow, ceil, −)
ac  return;
a0  }
70  }
0e  C[of][n] = −z;
c1  FOR(i,n) X[i] = XX[i];
a3 }
||
68 double ip(int m, int n, double
9b  C[][MAXN], double X[]) {
d5  FOR(i,n) C[m+1][i] = −C[0][i];
eb  C[m+1][n] = INF;
49  doip(m+1,m+1,n,C,X);
7e  FOR(i,n) X[i] = rint(X[i]);
6a  return rint(−C[m+1][n]);
f6 }
|| =================== kmp.cc ====================
|| Knuth−Morris Pratt String Matching
|| Find occurrences of the pattern string
|| P[1..m] in the text T[1..n].  The text
|| T is processed on−line (not stored in
|| memory).  Running time is O(n+m)
|| pi[x] is the length of the longest
|| prefix that matches a suffix of
|| P[1]...P[x].
||
79 #include <stdio.h>
5e #include <string.h>
||
b3 #define MAXM 60009
||
eb int P[MAXM+2];
8d int pi[MAXM+2];
e5 main() {
79  int i,j,k,q,ans,n,m;
b2  int Ti;
||
4e  while(1==scanf("%d",&m)) {
07   for(i=1;i<=m;i++) scanf("%d",&P[i]);
||
     //prepare helper function
82   memset(pi,0,sizeof(int)*(m+2));
b9   pi[1]=0; k=0;
d5   for(q=2;q<=m;q++) {
aa    while(k>0 && P[k+1]−P[q]) k=pi[k];
1b    if(P[k+1]==P[q]) k++;
15    pi[q]=k;
fd   }
||
     //read text and perform matching
19   scanf("%d",&n);
f9   q=0;
5c   ans=−1;
72   for(i=1;i<=n;i++) {
8d    scanf("%d",&Ti);
b5    while(q>0 && P[q+1]!=Ti) q=pi[q];
1c    if(P[q+1]==Ti) q++;
89    if(q==m) {
e9     q=pi[q];
      //pattern occurs at position i−m+1
      //that is, it occurs at offset i−m
49     if(i−m<ans || ans==−1) {
fb      ans=i−m;
6a     }
86    }
c9   }
||
     //print out offset of first hit
```

```
17   if(ans==−1) printf("no solution\n");
6f   else printf("%d\n",ans);
f6  }
b8  return 0;
48 }
|| =============== leftist_heap.cc ================
a5 #include "leftist_heap.h"
||
d2 int main() {
4d  mheap<int> *foo=0;
af  while (1) {
21   char buf[512];
dd   int arg;
9b   gets(buf);
79   sscanf(buf, "%*s %i", &arg);
ea   if (!strncmp(buf, "add ", 4))
14    foo = foo−>insert(arg);
a1   else if (!strncmp(buf, "top", 3))
aa    printf("%i\n", foo−>top());
fe   else if (!strncmp(buf, "pop", 3))
45    foo = foo−>pop();
97  }
01 }
|| ================ leftist_heap.h ================
|| Richard’s mergeable heaps.
|| The null pointer represents the empty
|| heap.  This code is slow; it’s only
|| slightly faster than using an STL set
|| as your heap. (Those suck at merging,
|| though.)
|| CAUTION:  This is a max−heap.
||
79 #include <stdio.h>
8c #include <stdlib.h>
20 #include <string.h>
93 #include <algorithm>
b6 using namespace std;
||
87 template <typename T> struct mheap {
cb  T key; int d;
72  mheap *l, *r;
||
a3  mheap(T k, mheap*a=0, mheap*b=0) {
e2   key = k; l = a; r = b; fixit();
12  }
||
6f  void fixit() {
3a   if (!r) { d=0; return; }
33   if (!l || r−>d > l−>d) swap(l,r);
01   d = r ? 1 + r−>d : 0;
9d  }
||
5f  mheap *merge(mheap *b) {
0d   return this ? b ? b−>key <= key ?
c1    r = r−>merge(b), fixit(), this :
f0    b−>merge(this) : this : b;
02  }
||
e2  mheap *insert(T k) {
e1   return merge(new mheap(k));
72  }
80  T top() const { return key; }
1d  mheap *pop() { return l−>merge(r); }
67 };
|| =================== lp2.h =====================
||  Simplex Method (Linear Programming)
||
||  m − number of (<=) inequalities
||  n − number of variables
||
||  C − (m+1) by (n+1) array of coeffs:
||   row 0     − obj fun coeffs
||   row 1:m   − <= inequalities
||   col 0:n−1 − inequality coeffs
||   col n     − inequality RHS
||  C[0][n] must be 0.
||
||  X[n] − result variables
||
||  return value − max value of obj fun
||   (−inf for infeasible,
||   inf for unbounded)
||
d2 #include <algorithm>
5b #include <utility>
9f using namespace std;
||
0a #define FR(i,a,b) \
8f  for (int i=(a);i<(b);i++)
c8 #define FOR(i,n) FR(i,0,n)
20 #define FRE(i,a,b) FR(i,a,b+1)
16 #define FORE(i,n) FRE(i,0,n)
d1 #define MP make_pair
||
16 #define MAXM 400   // leave one extra
```

```
a4 #define MAXN 400   // leave one extra
d1 #define EPS 1e−9
36 #define INF 1.0/0.0
||
42 double A[MAXM][MAXN];
f6 int basis[MAXM], out[MAXN];
||
63 void pivot(int m, int n, int a, int b) {
65  FORE(i,m) if (i−a) FORE(j,n) if (j−b)
66   A[i][j] −= A[a][j]*A[i][b]/A[a][b];
8a  FORE(j,n) if (j−b) A[a][j] /= A[a][b];
99  FORE(i,m) if (i−a) A[i][b] /=−A[a][b];
7d  A[a][b] = 1/A[a][b];
65  swap(basis[a], out[b]);
26 }
||
0e double simplex(int m, int n,
ae  double C[][MAXN], double X[]) {
cb  int ii,jj;
e1  FRE(i,1,m) FORE(j,n) A[i][j]=C[i][j];
e7  FORE(j,n) A[0][j] = −C[0][j];
49  FORE(i,m) basis[i] = −i;
1f  FORE(j,n) out[j] = j;
||
7e  for(;;) {
7e   ii=1; FRE(i,1,m)
c1    if (MP(A[i][n], basis[i])
73     < MP(A[ii][n], basis[ii])) ii=i;
b0   if (A[ii][n] >= −EPS) break;
3f   jj=0; FOR(j,n)
a2    if (MP(A[ii][j], out[j])
45     < MP(A[ii][jj], out[jj])) jj=j;
1c   if (A[ii][jj] >= −EPS) return −INF;
a0   pivot(m,n,ii,jj);
30  }
||
7e  for(;;) {
d8   jj=0; FOR(j,n)
83    if (MP(A[0][j], out[j])
91     < MP(A[0][jj], out[jj])) jj=j;
16   if (A[0][jj] > −EPS) break;
66   ii=0; FRE(i,1,m)
90    if (A[i][jj] > EPS && (!ii ||
5b     MP(A[i][n]*A[ii][jj], basis[i]) <
de    MP(A[ii][n]*A[i][jj], basis[ii])))
8e     ii=i;
37   if (A[ii][jj] <= EPS) return INF;
d1   pivot(m,n,ii,jj);
db  }
||
5f  FOR(j,n) X[j] = 0;
75  FRE(i,1,m) if (basis[i] >= 0)
cc   X[basis[i]] = A[i][n];
29  return A[0][n];
2c }
||
|| debug only; not used
a2 void print(int m, int n, char *msg) {
7d  int i,j;
ec  printf("%s\n",msg);
31  FORE(i,m) {
b8   FORE(j,m) printf(" %10d",i==j);
fa   FORE(j,n) printf(" %10g",A[i][j]);
d8   printf("\n");
c3  }
37  FORE(i,m) printf(" %10d",basis[i]);
91  FOR(j,n) printf(" %10d",out[j]);
ed  printf("\n");
42 }
|| ================= max_flow.cc ==================
|| =========== Max flow ==============
3b template<int N>struct net{
e4  int a[N][N],p[N],q[N],z[N],f[N],g[N];
99  bool b[N];
d2  int flow(int s,int t,int n){
7a   for(int x,h,res=0;memset(p,0,N*21);){
b8    queue<int> Q;
74    for(z[s]=1,Q.push(s);Q.size() && (x=Q.front()
||                                    )!=t;Q.pop())
67     REP(y,n)if(a[x][y] && !z[y])z[y]=z[x]+1,Q.
||                                          push(y);
33    if(!z[t]) return res;
20    REP(i,n)REP(j,n)if(i!=t && j!=s && z[i]+1==z[
||                                              j])
5a     p[i]+=a[i][j],q[j]+=a[i][j];
c9    for(q[s]=−1u/2,p[t]=−1u/2;;){
34     REP(i,n)if(!b[i] && !(p[i]&&q[i]))b[i]=true,
||                                       Q.push(i);
a7     for(;Q.size();){
7c      x=Q.front(),Q.pop();
d2      if(b[x])REP(y,n)if(!b[y]){
61       if(z[x]+1==z[y])q[y]−=a[x][y];
50       else if(z[y]+1==z[x])p[y]−=a[y][x];
e7       if(!p[y]||!q[y])b[y]=true,Q.push(y);
1a      }
85     }
82     if(b[s])break; int d=-1u/2;
49     REP(i,n)if(!b[i] && (p[i]<d || q[i]<d))d=min
||                                  (p[x=i],q[i]);
70     for(res+=(f[x]=g[x]=d),Q.push(x);Q.size();){
31      x=Q.front(),Q.pop();
ac      for(int y=0;y<n;++y)if(!b[y])
4b       if(z[x]+1==z[y] && (d=min(a[x][y],f[x]))){
94        if(!f[y])Q.push(y);
e3        p[x]-=d,a[x][y]-=d,q[y]-=d;
17        f[y]+=d,a[y][x]+=d,f[x]-=d;
5d       }else if(z[y]+1==z[x] && (d=min(a[y][x],g[
||                                         x]))){
b8        if(!g[y])Q.push(y);
42        p[y]-=d,a[y][x]-=d,q[x]-=d;
31        g[x]-=d,a[x][y]+=d,g[y]+=d;
bc       }
00     }
5d    }
47   }
3f  }
f7 };
|| ================== mcmf.cc ===================
04 const int V = 100+2, src = V-2, snk = V-1;
c2 const double eps = 1e-8;
fa int flo[V][V],cap[V][V],mark[V];
72 double cost[V][V],y[V];
7c vector<int> adj[V];
||
e3 int aug(int v, int f=INT_MAX) {
ef  if (mark[v]++) return 0;
f9  if (v==snk) return f;
||
0b  FORI(i,adj[v]) {
a1   int w = adj[v][i];
e3   if (flo[v][w] < cap[v][w] && fabs(y[v]+cost[v]
||                              [w]-y[w]) < eps) {
73    int g = aug(w, min(f, cap[v][w]-flo[v][w]));
a3    if (g) {
3e     flo[v][w] += g;
7c     flo[w][v] -= g;
9c     return g;
d9    }
63   }
06  }
dd  return 0;
cb }
||
c5 double mcmf() {
e1  double ret = 0;
||
c5  while (1) {
d1   FOR(v,V) y[v] = DBL_MAX;
89   y[src]=0;
7b   deque<int> q(1, src);
||
51   while (q.size()) {
a2    int v = q.front(); q.pop_front();
56    FORI(i,adj[v]) {
1f     int w = adj[v][i];
4b     if (flo[v][w] < cap[v][w] && y[v]+cost[v][w]
||                                  +eps < y[w]) {
20      y[w] = y[v]+cost[v][w];
f5      q.push_back(w);
5a     }
0f    }
d5   }
||
6e   CLR(mark,0);
af   int f = aug(src);
30   if (f==0) break;
d8   ret += f*y[snk];
53  }
84  return ret;
c1 }
||
30 void resetflow() {
55  CLR(flo,0); CLR(cap,0); CLR(cost,0);
52  FOR(v,V) adj[v].clear();
0f }
||
7e void connect(int v, int w, int u, double c) {
41  cap[v][w] = u;
93  cost[v][w] = c;
8e  cost[w][v] = -c;
bc  adj[v].push_back(w);
af  adj[w].push_back(v);
d8 }
|| ================ mincircle.cc ================
|| Minimum circle containing a set of
|| points.  O(nlogn) expected runtime.
||
e2 #include <cstdio>
47 #include <complex>
50 #include <vector>
```

```
72 #include <algorithm>
cd using namespace std;
||
78 #define EPS 1e-10
19 #define BAD point(1e101,1e101)
78 #define X real()
99 #define Y imag()
b3 #define FR(i,a,b) for(int i=(a);i<(b);\
91  i++)
3e #define FOR(i,n) FR(i,0,n)
||
9b typedef complex<double> point;
4b typedef pair<point,point> line;
46 typedef pair<double,point> circle;
5e vector<point> pts;
||
2a double operator^(const point& a, const
33  point& b) { return (a*conj(b)).Y; }
||
a0 bool inside( point& p, circle& c ) {
24  return norm(p-c.second)
cc   <= c.first * c.first;
33 }
||
f0 point intersect_line(const line& a,
52  const line& b) {
38  double t = (a.second-a.first)
05   ^ (b.second-b.first);
7e  if( t == 0 ) return BAD; // parallel
0b  t = -(((a.first-b.first)
7f   ^ (b.second-b.first))/t);
03  return (1-t)*a.first + t*a.second;
05 }
||
71 line perp_bisector(const point& a,
2c  const point& b) {
50  return line((a+b)/2.0,
c3   (a+b)/2.0+(b-a)*point(0,1));
f3 }
||
c3 point circumcentre(const point& a,
0d  const point& b, const point& c) {
47  if(abs(c-a) < EPS || abs(c-b) < EPS)
b3   return 0.5*(a+b);
d1  if(abs(a-b) < EPS) return 0.5*(b+c);
38  return intersect_line(perp_bisector
1f   (a,b),perp_bisector(a,c));
e5 }
||
35 circle fix2(int n, point&f1, point&f2) {
c1  circle c(abs(f2-f1)*0.5,(f1+f2)*0.5);
26  FOR(i,n) if(!inside(pts[i],c)) {
2a   point p=circumcentre(pts[i],f1,f2);
6d   if (p != BAD) c=circle(abs(p-f1),p);
16  }
ad  return c;
88 }
||
e5 circle fix1(int n, point& f) {
27  circle c(0,f);
50  FOR(i,n) if(!inside(pts[i],c))
30   c=fix2(i,pts[i],f);
4b  return c;
bf }
||
5e circle mincircle(int n) {
1d  circle c(0,pts[0]);
52  FR(i,1,n) if(!inside(pts[i],c))
1d   c=fix1(i,pts[i]);
d0  return c;
d3 }
||
d2 int main() {
e5  int n;
ab  scanf("%d",&n);
b9  FOR(i,n) {
06   double x, y;
63   scanf("%lf %lf",&x,&y);
0c   pts.push_back(point(x,y));
fd  }
ed  srand48(12345);
23  random_shuffle(pts.begin(),pts.end());
1e  circle c = mincircle(pts.size());
b0  printf("%.2f\n%.2f %.2f\n",
7e   c.first,c.second.X,c.second.Y);
76 }
|| ============= min_cost_max_flow.cc =============
|| ============= Min cost max flow ================
33 struct mcnet {
5d  vector< vector< pair<int,pii> > > a;
7b  vector< pair<int,pii> >::iterator i;
c9  mcnet(int n):a(n){}
c6  void add(int x,int y,int w,int c=1){
f7   for(i=a[x].begin();i!=a[x].end();++i)
6c   if(i->first==y && i->second.second==w){
```

```
b3    if(!(i->second.first+=c))
ee     *i=a[x].back(),a[x].pop_back();
60    return;
4e   }
ef   a[x].pb(make_pair(y,pii(c,w)));
ea  }
f8  pii flow(int s,int t){
5c   vi p(a.size());
52   for(pii res;;){
d5    priority_queue<pii> Q;
59    vi f(a.size(),-1u/2),prev(a.size(),-1);
3d    f[s]=0,Q.push(pii(0,s));
46    for(int x,d;Q.size();){
b0     pii cur=Q.top(); Q.pop();
67     if(f[x=cur.second]!=(d=-cur.first))continue;
eb     for(i=a[x].begin();i!=a[x].end();++i){
59      int y=i->first,z=i->second.second;
2c      if(d+z+p[x]-p[y]<f[y])
cd       f[y]=d+z+p[x]-p[y],prev[y]=x,
8f       Q.push(pii(-f[y],y));
f1     }
ba    }
91    if(prev[t]<0) return res;
86    int flow=-1u/2;
55    for(int x,y=t;~(x=prev[y]);y=x)
40     for(i=a[x].begin();i!=a[x].end();++i)
ac      if(i->first==y && f[x]+i->second.second+p[x
||                                ]==f[y]+p[y])
82       flow=min(flow,i->second.first);
2a    res.first+=flow,res.second+=(f[t]+p[t]-p[s])*
||                                          flow;
53    for(int x,y=t;~(x=prev[y]);y=x)
7d     add(x,y,f[y]+p[y]-f[x]-p[x],-flow),
34     add(y,x,f[x]+p[x]-f[y]-p[y],+flow);
72    REP(i,a.size())p[i]+=f[i];
c6   }
db  }
f5 };
|| =================== mincut.h ==================
|| Global minimum cut in a graph
|| NOT minimum-(s,t)-cut.
|| Tor Myklebust
79 #include <stdio.h>
5e #include <string.h>
78 #define FR(i,a,b) for(int i=a;i<b;i++)
52 #define FOR(i,n) FR(i,0,n)
||
ec int adj[256][256],n;
||
f3 int phase() {
7e  int v[n],d[n],rv=0,s=0;
14  v[0] = 0;
60  FOR(i,n) d[i] = adj[0][i]; d[0]=-1;
a8  FR(i,1,n) {
c9   FOR(j,n) if (d[j]>d[s]) s=j;
1b   d[v[i] = s] = -1;
02   FOR(j,n) if (d[j]>=0) d[j]+=adj[j][s];
ad  }
87  int a=v[n-1],b=v[n-2];
25  FOR(i,n) rv += adj[i][a];
3f  FOR(i,n)
2f   adj[i][b] = adj[b][i] += adj[a][i],
52   adj[i][a] = adj[a][i] = adj[n-1][i];
93  adj[b][a] = adj[a][b] += adj[a][a];
6b  adj[a][a] = adj[b][b] = 0;
30  n--;
b9  return rv;
d8 }
||
1b int mincut() {
0e  int ans = 0x7fffffff;
d7  while (n > 2) ans <?= phase();
38  return ans <? adj[0][1];
6c }
|| ============= min_cyclic_shift.cc ==============
|| ======== Min cyclic shift ==============
||
74 int min_cyclic_shift(char*s){
20  for(int n=strlen(s),i=0,j,k,r;;){
2e   for(k=i,j=i+1;s[k<n?k:k-n]<=s[j<n?j:j-n];++j)
42    s[k<n?k:k-n]<s[j<n?j:j-n]?k=i:++k;
a3   for(r=i;i<=k;i+=j-k);
b8   if(i>=n) return r;
f0  }
4a }
|| ================= mingeuler.cc =================
|| EULER CYCLE/PATH IN GRAPHS & DIGRAPHS
|| -----------------------------------
||
|| - algo is O(n), but takes O(nlogn)
||  since it sorts adjacency list
|| - find_cycle and find_path return
||  false if no cycle/path exists
|| - for undirected, add reverse copy of
||  each edge, but share hit flag
```

```
|| - multiedges and loops will be
||  handled appropriately
|| - if you want to find an euler cycle
||  subject to some ordering constraint
||  modify the sort so the nodes for
||  each from node are ordered in de-
||  creasing order of preference for
||  the to node
||
63 using namespace std;
da #include <stdio.h>
97 #include <list>
fc #include <algorithm>
||
52 #define MAXE 1000
14 #define MAXN 1000
||
aa #define FORALL(i,s) for(typeof(s.\
01 begin()) i=s.begin();i!=s.end();i++)
66 #define ROF(i,n) for(i=n-1;i>=0;i--)
||
7b list<int> p;
f8 typedef list<int>::iterator iter;
||
48 struct edge {
de  int from, to;
ec  bool *hit;
a7  bool operator<(const edge&other) const
e6   { return from < other.from; }
1b } e[MAXE];
c5 bool hit[MAXE];
c0 int ne;
d2 int firste[MAXN];
||
||
e4 void uedge(int from, int to) {
44  e[ne].from = e[ne+1].to   = from;
52  e[ne].to   = e[ne+1].from = to;
e7  e[ne].hit  = e[ne+1].hit  = &hit[ne];
b4  ne+=2;
20 }
||
e5 void dedge(int from, int to) {
3b  e[ne].from = from;
23  e[ne].to   = to;
a0  e[ne].hit  = &hit[ne];
0e  ne++;
6d }
||
16 void buildgraph() {
f5  e[ne].from = -1;
ee  sort(e,e+ne);
49  ROF(a,ne) firste[e[a].from] = a;
47 }
||
dc void run(iter it, int node) {
ef  while (true) {
24   int i=firste[node];
5f   if (e[i].from != node) return;
||
47   if (!*e[i].hit) {
d8    *e[i].hit = true;
b7    p.insert(it, i);
0b    node = e[i].to;
15   }
2d   else
6d    firste[node]++;
d6  }
1b }
||
94 void euler(int start) {
0e  if (start==-1) return;
82  run(p.begin(), start);
b1  FORALL(it, p) {
48   if (e[firste[e[*it].from]].from
55    == e[*it].from) {
35    iter n = it;
38    it--;
0d    run(n, e[*n].from);
dd   }
03  }
25 }
||
|| Doesn’t reset the stuff needed to do
|| euler tests Code for Euler tests
|| resets itself
dd void reset() {
13  p.clear();
eb  memset(firste,0,sizeof(firste));
5b  memset(hit,0,sizeof(hit));
59  ne = 0;
6f }
||
|| Stuff used for testing euler prop-
|| erties and for finding a proper node
|| for starting an euler path search
||  1. connectivity
||  2. for directed: outdegree=indegree
||   for undirected: even degree
||  3. in addition, for paths
||   for directed:
||  <= 1 node with one more out than in
||  <= 1 node with one more in than out
||   for undirected: <= 2 odd degrees
||
d6 int nn; // must set this to # of nodes
ab int deg[MAXN];
||
51 bool ufindcycle() {
40  memset(deg,0,sizeof(deg));
01  FOR(a,ne) deg[e[a].from] ^= 1;
||
93  FOR(a,nn) if (deg[a]) return false;
73  euler(e[0].from);
00  return (int)p.size()==ne/2;
e5 }
||
41 bool dfindcycle() {
04  memset(deg,0,sizeof(deg));
92  FOR(a,ne)
d2   deg[e[a].from]++, deg[e[a].to]--;
||
93  FOR(a,nn) if (deg[a]) return false;
73  euler(e[0].from);
98  return (int)p.size()==ne;
21 }
||
06 bool ufindpath() {
15  memset(deg,0,sizeof(deg));
87  FOR(a,ne) deg[e[a].from] ^= 1;
||
c0  int odd=0,start=e[0].from;
98  FOR(a,nn) if(deg[a]) odd++, start = a;
b3  if (odd>2) return false;
2b  euler(start);
16  return (int)p.size()==ne/2;
35 }
||
06 bool dfindpath() {
95  memset(deg,0,sizeof(deg));
1c  FOR(a,ne)
5c   deg[e[a].from]++, deg[e[a].to]--;
||
78  int odd=0,start=e[0].from,sum=0;
60  FOR(a,nn) {
a1   sum += deg[a];
53   odd += abs(deg[a]);
c8   if(deg[a] > 0) start = a;
b3  }
b6  if(odd>2 || sum) return false;
29  euler(start);
65  return (int)p.size()==ne;
f6 }
||
c4 int main(void) {
d6  nn=20;
||  uedge(6,6); uedge(0,6); uedge(6,0);
b2   uedge(0,3); uedge(3,2); uedge(2,0);
8f   uedge(3,5); uedge(5,3); uedge(5,5);
34   buildgraph();
0f   if(ufindcycle()) FORALL(it,p)
8e    printf("%d->%d\n",
d3     e[*it].from, e[*it].to);
6c   else printf("NO\n");
b0   return 0;
98 }
|| ================ min_suffix.cc =================
|| ========== Min suffix ==============
||
e7 int min_suffix(char*s){
ec  for(int i=0,j,k,r;;){
17   for(k=i,j=i+1;s[k]<=s[j];s[k]<s[j++]?k=i:++k);
b0   for(;i<=k;i+=j-k)r=i;
9f   if(!s[i]) return r;
f1  }
8a }
|| ================== miscgeom.h ==================
02 #include <math.h>
db using namespace std;
55 #define ld long double
|| Conversion between cylindrical and
|| cartesian coordinates
|| *zenith is angle from from the
|| vertical (amount south from north
|| pole, also known as co-latitude)
|| *azimuth is angle around (amount east
|| from a fixed point)
|| *longitude = azimuth
|| *latitude = pi/2 - zenith
|| *zenith   = pi/2 - latitude
```

```
90 void sphtocart(ld zenith, ld azimuth,
b5  ld rho, ld *x, ld *y, ld *z) {
94  *x = rho * cos(azimuth)* sin(zenith);
c1  *y = rho * sin(azimuth)* sin(zenith);
26  *z = rho * cos(zenith);
dc }
||
|| division by zero if given (0,0,0)
b0 void carttosph(ld x, ld y, ld z, ld
cf  *zenith, ld *azimuth, ld *rho) {
a0  *rho = sqrt(x*x+y*y+z*z);
e7  *azimuth = atan2(y,x);
50  *zenith  = acos(z/ (*rho));
37 }
|| ================= moregeom.cc ==================
|| Denis' geometry code
||
89 #include <map>
f1 #include <set>
27 #include <deque>
9d #include <algorithm>
0b using namespace std;
||
|| most functions require some 2D stuff
3e #include "geom.h"
||
40 #define FOR(i,n) for (int i=0;i<n;i++)
f3 #define EPS  1e-8
ec #define INF  1e100
||
56 #define T double
||
8c const T pi = atan2(0.0, -1.0);
||
5a bool small(const T&a) {
25  return -EPS <= a && a <= EPS;
04 }
||
|| 3d point class
0d struct xyz {
b9  T x, y, z;
||
09  xyz() {}
b8  xyz(T xx,T yy,T zz):x(xx),y(yy),z(zz)
5d   {}
||
88  T norm() const {
52   return hypot(x,hypot(y,z));
1f  }
||
dd  xyz operator/(T t) const {
db   return xyz(x / t, y / t, z / t);
93  }
||
dc  xyz operator*(T t) const {
d1   return xyz(x * t, y * t, z * t);
83  }
||
0d  T operator*(const xyz &b) const {
8e   return x*b.x + y*b.y + z*b.z;
e9  }
||
e1  xyz operator+(const xyz &b) const {
1e   return xyz(x+b.x, y+b.y, z+b.z);
0d  }
||
3c  xyz operator-() const {
1d   return xyz(-x, -y, -z);
15  }
||
61  xyz operator-(const xyz &b) const {
de   return xyz(x-b.x, y-b.y, z-b.z);
f3  }
||
be  bool small() const {
1d   return ::small(x)&&::small(y)
55    &&::small(z);
e7  }
||
25  bool operator<(const xyz &other)const{
55   return
4b    x<other.x-EPS || x<other.x+EPS &&
7d    (y<other.y-EPS || y<other.y+EPS &&
56     z<other.z-EPS);
67  }
d4 };
||
|| cross product
18 xyz cross(const xyz &a, const xyz &b) {
ed  return xyz(a.y * b.z - b.y * a.z,
40   a.z * b.x - b.z * a.x,
14   a.x * b.y - b.x * a.y);
1d }
||
6d struct plane {
```

```
e2  T d;
6d  xyz n;
89  plane() {}
11  plane(xyz nn, T dd) : n(nn), d(dd) {}
25  plane(xyz a, xyz b, xyz c) {
08   n = cross(a - b, b - c);
ad   d = n.x*a.x + n.y*a.y + n.z*a.z;
a0  }
22 };
||
|| plane through 3 points with 2d
|| coordinate system defined
0c struct projplane {
    // AB length
d1  T AB;
    // a--origin, X--x axis, Y--y axis,
    // Z--perpendicular to plane
e3  xyz a, X, Y, Z;
db  projplane() {}
62  bool init(xyz aa, xyz bb, xyz cc) {
06   a = aa;
ea   X = bb - aa;
82   AB = X.norm();
f2   X = X / AB;
1c   Z = cross(X,cc-bb), Y = cross(X,Z);
67   T nn = Y.norm();
c7   if (nn < EPS) return false;
13   Z = Z / nn;
85   Y = Y / nn;
68   return true;
86  }
||
bd  point proj(xyz c) {
7e   xyz CAv = c - a;
11   return point(CAv * X, CAv * Y);
65  }
8b  xyz back(point cc) {
33   return a + X * cc.x + Y * cc.y;
1c  }
89  bool inplane(xyz c) {
6a   return small((c - a) * Z);
b6  }
ac };
||
|| distance between two lines in 3D
2f bool lli(xyz p1, xyz p2, xyz p3, xyz p4,
93  xyz *pa, xyz *pb) {
46  xyz p43 = p4 - p3;
ac  xyz p21 = p2 - p1;
03  if(p21.small() || p43.small())
13   return false;
86  xyz p13 = p1 - p3;
a7  T d1343 = p13*p43, d4321 = p43*p21;
f7  T d1321 = p13*p21, d4343 = p43*p43,
0f   d2121 = p21*p21;
5f  T numer = d1343*d4321 - d1321*d4343;
39  T denom = d2121*d4343 - d4321*d4321;
3c  if (small(denom)) return false;
||
f0  T mua = numer / denom;
b2  T mub = (d1343 + d4321 * mua) / d4343;
01  *pa = p1 + p21 * mua;
5a  *pb = p3 + p43 * mub;
bd  return true;
70 }
||
|| centre of smallest sphere and normal
|| to plane through three points.
38 xyz center(const xyz &a, const xyz &b,
61  const xyz &c, xyz *normal) {
15  projplane q;
ca  q.init(a, b, c);
1b  *normal = q.Z;
72  return q.back(circle(point(0.0, 0.0),
78   point(q.AB, 0.0), q.proj(c)));
f6 }
||
|| centre, radius of sphere thru 4 pts.
7d xyz sphere(const xyz p[4], T *r) {
04  xyz p1,n1,p2,n2,
53   c1=center(p[0],p[1],p[2],&n1),
1d   c2=center(p[0],p[1],p[3],&n2);
9f  if (!lli(c1,c1+n1,c2,c2+n2,&p1,&p2))
6f   throw "q?";
05  xyz c = (p1 + p2) / 2.0;
51  *r = (p[0] - c).norm();
47  return c;
b7 }
||
|| area of the sector or a circle radius
|| r viewed at angle alpha from centre
ab T areasector(T r, T alpha) {
58  return 0.5*r*r*(alpha-sin(alpha));
57 }
||
```

```
|| returns true if line through p3 and
|| p4 strictly crosses segment p1<->p2
11 bool linesegment(point p1, point p2,
76 point p3, point p4) {
23  double an, ad, bn, bd;
||
6d  an = (p4.x - p3.x) * (p1.y - p3.y)
12   - (p4.y - p3.y) * (p1.x - p3.x);
29  ad = (p4.y - p3.y) * (p2.x - p1.x)
68   - (p4.x - p3.x) * (p2.y - p1.y);
af  bn = (p2.x - p1.x) * (p1.y - p3.y)
02   - (p2.y - p1.y) * (p1.x - p3.x);
75  bd = (p4.y - p3.y) * (p2.x - p1.x)
88   - (p4.x - p3.x) * (p2.y - p1.y);
||
ec  if (fabs(bd) < EPS || fabs(ad) < EPS)
e7   return false;
91  double ua = an / ad, ub = bn / bd;
||
f1  if(ub < EPS || ub > 1.0 + EPS)
38   return false;
9f  return true;
51 }
||
|| intersect convex polygon a with half-
|| space defined by the line through c1
|| and c2. (picks intersection on the
|| right of the line, updates c1 and c2
|| to be the actual points of the poly
|| that were created as a result of the
|| intersection)
97 bool convexpolyvsline(const poly &a,
9c poly &res, point &c1, point &c2) {
bd  int j;
66  point sc1(c1), sc2(c2);
be  line cut = thru(c1, c2);
88  res.clear();
8e  for(j = 0; j < a.size(); j++) {
0d   point l1=a[j], l2=a[(1+j)%a.size()];
9c   if (!linesegment(sc1, sc2, l1, l2))
f4    continue;
3d   if ((sc2.x-sc1.x)*(l2.y-l1.y) -
25    (sc2.y-sc1.y)*(l2.x-l1.x)>=-EPS)
39    continue;
89   c1 = isct(cut, thru(l1, l2));
71   res.push_back(c1);
9f   if (hypot(c1.x-l2.x,c1.y-l2.y)>=EPS)
20    res.push_back(l2);
34   break;
f4  }
||
54  if (j == a.size()) {
cb   res = a;
9d   return false;
b9  }
||
dc  int next = (1 + j) % a.size();
17  for(;;) {
9e   j = next;
a6   next = (1 + j) % a.size();
f4   point l1 = a[j], l2 = a[next];
a3   if(!linesegment(sc1, sc2, l1, l2))
04    res.push_back(l2);
29   else {
0c    c2 = isct(cut, thru(l1, l2));
cd    res.push_back(c2);
81    break;
69   }
1e  }
f9  return true;
3a }
||
|| compute 3d hull of the set of points
|| (O(n^2 * #edges in the hull))
17 void hull3d(const vector<xyz> &pts,
8f vector<vector<int> > &hull) {
33  T mz = -INF;
80  xyz inside(0.0, 0.0, 0.0);
a1  int idx, idx2, i, j1, j2, k,
f1   N = pts.size(), n1;
||
    // find topmost vertex
e5  FOR(i,N) {
9d   inside = inside + pts[i] / N;
fd   if (pts[i].z > mz)
ed    mz = pts[i].z, idx = i;
8a  }
||
    // find edge in the hull
c5  mz = pi;
cb  FOR(i,N) {
12   if(i == idx) continue;
aa   T d=atan2(pts[idx].z - pts[i].z,
15    hypot(pts[i].x - pts[idx].x,
dc     pts[i].y - pts[idx].y));
3f   if(d > pi / 2.0)  d = pi - d;
b4   if(d < mz)      mz = d, idx2 = i;
e0  }
||
18  set<pair<int, int> > done;
7d  deque<pair<int, int> > edges;
55  vector<vector<vector<bool> > >
45   faces(N);
67  done.insert(make_pair(idx, idx2));
19  edges.push_back(make_pair(idx, idx2));
||
    // grow set of edges in the hull
d0  while(!edges.empty()) {
79   pair<int, int> e = edges.front();
07   edges.pop_front();
cc   xyz from(pts[e.first]),
ba    to(pts[e.second]);
34   FOR(i,N) {
d8    projplane Q;
9e    if (!Q.init(from, to, pts[i]))
e6     continue;
||
4e    for(j1=0,n1=faces[e.first].size();
04     j1 < n1; j1++) {
0a     const vector<bool> &v =
8f      faces[e.first][j1];
27     if(v[e.second] && v[i]) break;
0c    }
a0    if(j1 != n1) continue;
||
b2    for(j1 = 0; j1 < N &&
a2     (Q.a - pts[j1]) * Q.Z <= EPS;
26     j1++);
6c    for(j2 = 0; j2 < N &&
64     (Q.a - pts[j2]) * Q.Z >= -EPS;
3d     j2++);
||
d7    if(j1 == N || j2 == N) {
       // found a new face
a6     vector<point> pr;
8f     vector<int> in, rr;
b8     for(k = 0; k < N; k++)
c7      if(Q.inplane(pts[k]))
1e       in.push_back(k),
e3       pr.push_back(Q.proj(pts[k]));
||
69     if(in.size() > 3) {
f3      rr = idxhull(pr);
9b      for(k = 0, n1 = rr.size();
dc       k < n1; k++)
2e       rr[k] = in[rr[k]];
dc     }
c1     else //triangle is always convex
00      rr = in;
39     hull.push_back(rr);
||
       // make sure that vertices in
       // all faces are listed ccw
6c     Q.init(pts[rr[0]], pts[rr[1]],
c7      pts[rr[2]]);
9a     if((Q.a - inside) * Q.Z < 0)
d5      reverse(hull.back().begin(),
82       hull.back().end());
||
94     vector<bool> all(N);
6c     for(k = 0; k < in.size(); k++)
6c      all[in[k]] = true;
||
c1     n1 = rr.size();
6c     for(k = 0; k < n1; k++) {
05      faces[rr[k]].push_back(all);
        // add edges
7e      pair<int, int> edge(rr[k],
85       rr[(1 + k) % n1]);
5a      if (done.find(edge)
4e       == done.end())
09       done.insert(edge),
34       edges.push_back(edge);
ac     }
07     break;
c5    }
8b   }
31  }
b4 }
||
61 main(){}  // make it compile in genbook
|| ==================== mst.cc ====================
|| ======== Directed MST  ================
||
31 int n,m,_x,_y,c;
ec vector<pii> v[511],w[511];
66 int nm;
01 bool u[511];
||
48 pii getM(int node){
```

```
d0  pii res(-1,INF);
f4  REP(i,w[node].size()) if(w[node][i].second<res.
||                                          second)
8a   res=w[node][i];
72  return res;
ee }
||
35 bool p[511];
0d bool D[511];
46 pii d[511];
b7 int N[511];
||
17 int go(){
9a  int cost=0;
09  FOR(i,1,n) if(!p[i]) d[i]=getM(i),cost+=d[i].
||                                second,p[i]=true;
42  CL(u,false);
82  vector<vi > cycle;
3d  FOR(i,1,n) if(!D[i] && !u[i]){
b0   int V=i;
5f   vi c;
c6   do{
2f    c.pb(V);
06    u[V]=true;
0b    V=d[V].first;
a2   }while(!u[V]);
3f   while(c.size() && c[0]!=V)
a6    c.erase(c.begin());
63   if(c.size()>1)
20    cycle.pb(c);
c4  }
d3  if(cycle.size()){
16   CL(N,-1);
de   REP(i,cycle.size()) REP(j,cycle[i].size())
00    N[cycle[i][j]]=i,D[cycle[i][j]]=true;
07   FOR(i,1,n){
58    if(N[d[i].first]!=-1) d[i].first=n+N[d[i].
||                                          first];
31    REP(j,w[i].size()){
d4     if(N[w[i][j].first]!=-1) w[i][j].first=n+N[w
||                                   [i][j].first];
1a     if(N[i]!=-1){
67      w[i][j].second-=d[i].second;
b5      if(w[i][j].first!=n+N[i])
27       w[n+N[i]].pb(w[i][j]);
ec     }
92    }
46    if(N[i]!=-1) w[i].clear();
ff   }
d7   n+=cycle.size();
43   return cost+go();
6f  }
ea  return cost;
e2 }
||
08 void dfs(int q){
3e  if(u[q]) return;
76  u[q]=true;nm++;
ff  REP(i,v[q].size()) dfs(v[q][i].first);
c2 }
||
21 Test code:
5a int main() {
33  while(cin>>n>>m,n||m){
bb   REP(i,500) v[i].clear(),w[i].clear();
f0   REP(i,m) {
d4    scanf("%d %d %d",&_x,&_y,&c);
71    _x--,_y--;
7a    v[_x].pb(pii(_y,c));
f9    w[_y].pb(pii(_x,c));
ea   }
78   CL(u,false);
52   CL(p,false);
f5   CL(D,false);
55   nm=0;
e7   if(dfs(0),nm!=n)
e0    puts("impossible");
23   else cout<<go()<<endl;
2c  }
dd  return 0;
9b }
|| ================== pulley.cpp ==================
||  Sample use of "ham.h"  for hamiltonial cycle
||  Reads "colorit.in" test input file and does ham
||                                          . cycle
||
79 #include <stdio.h>
5e #include <string.h>
6e #include <stdlib.h>
aa #include "pulley.h"
||
9b int i,j,k,m,n,T,N,M;
||
34 char name[300][12];
5c int nn,t;
```

```
56 char x[12],y[12];
ce int ix,iy;
||
ab int color[300];
||
2b typedef int (*qsortf)(const void*,const void*);
||
d2 int main() {
d8  scanf("%d",&T);
64  for (t=1;t<=T;t++) {
cc   scanf("%d%d",&N,&M);
cc   if (t != 1) printf("\n");
23   printf("Case #%d size %d %d\n",t,N,M);
c5   fflush(stdout);
86   reset(N);
8c   for (i=0;i<N;i++) scanf(" %s",name[i]);
b5   qsort(name,N,12,(qsortf)strcmp);
eb   for (i=0;i<M;i++) {
0c    scanf(" %s %s", x, y);
b4    ix = ((char*)bsearch(x,name,N,12,(qsortf)
||                                          strcmp)
49     - (char *)name)/12;
15    iy = ((char*)bsearch(y,name,N,12,(qsortf)
||                                          strcmp)
ec     - (char *)name)/12;
84    edge(ix,iy,-1);
02    edge(iy,ix,-1);
25   }
d2   if (ham(nv,-nv) != -nv) printf("no solution\n"
||                                               );
4f   else {
9c    for (i=0;i<=nv;i++) printf("%d ",hamcycle[i])
||                         ;  // repeats start vert
eb    printf("\n");
02   }
85  }
d1  return 0;
d3 }
|| =================== pulley.h ===================
|| Bill Pulleyblank's 3-opt Travelling Salesman
||                                        Heuristic
|| Works pretty well but don't count on it for
||                                     exact answer
||
|| Initialization is a bit weird:  0 is treated as
||                                         infinity
|| it finds the minimum weight tour, so you want to
||                                              use
|| negative weights.  To convert positive weights,
||                                              use
|| weight-BIGNUM where BIGNUM is a lot bigger than
||                                     your biggest
|| weight but not big enough to cause underflow
||
60 #define MAXV 500
8a #define MAXC MAXV
ad #include <string.h>
60 #include <assert.h>
||
a9 int adj[MAXV][MAXV];
||
d6 int nv;
||
0e void reset(int n) {
ea  nv = n;
79  memset(adj,0,sizeof(adj));
56 }
||
04 void edge(int i, int j, int c) {
34  adj[j][i] = adj[i][j] = c;
24 }
||
b3 int hamcycle[MAXV], tmp[MAXV];
87 #define h(a) hamcycle[(a)%nv]
96 #define c(a,b) adj[h(a)][h(b)]
||
a5 int ham(int u, int limit){
2b  int i,j,k,z,in,out,cost,calc,lasti;
50  for (i=0;i<nv;i++) h(i) = i;
93  cost = 0;
e2  for (z=0;z<nv;z++) cost += adj[h(z)][h(z+1)];
||
a8  i = 0;
32  again: for (lasti=i;i<lasti+nv;i++) {
e0   for (j=i+1;j<i+nv-1;j++) {
4f    for (k=j+1;k<i+nv;k++) {
4e     out = c(i,i+1) + c(j,j+1) + c(k,k+1);
9f     in = c(i,j+1) + c(i+1,k) + c(j,k+1);
21     if (in-out < 0) {
76      for (z=0;z<j-i;z++) tmp[z] = h(i+1+z);
29      for (z=0;z<k-j;z++) h(i+1+z) = h(j+1+z);
5c      for (z=0;z<j-i;z++) h(i+1+k-j+z) = tmp[z];
80      cost += (in-out);
a4      if (cost <= limit) return cost;
df      goto again;
```

```
43     }
8f    }
81   }
b9  }
c5  return cost;
83 }
|| ================== rangeop.h ===================
|| RangeOp is a data structure built on
|| an array and an associative operation
|| It takes linear setup time, linear
|| memory, then allows for log-time
|| updates, and log-time calculation of
|| the operation on arbitrary ranges.
|| It also has constant amortized time
|| when you merely increment the front
|| and/or back of ranges.
|| T will be the data type for the
|| operation.
|| op will be the associative (but not
|| necessarily commutative) binary
|| operation.
|| def, the "default value" for an empty
|| range, must be the identity under op.
|| ie, it should satisfy op(x, def) == x
|| == op(def, x).  Be careful to specify
|| it properly depending on your
|| operation.
||
|| Example declarations:
||
|| RangeOp<int, plus<int> > r(v.size());
|| r.set(0, v.begin(), v.end());
||  Creates a RangeOp, using the ad-
||  dition operation, from a known
||  vector v.  Note that the default
||  is assumed to be 0.  Also note
||  that this initialiation is nice
||   and linear in time/memory.
|| x = r.calc(2, 5);
||  x is now equal to v[2]+v[3]+v[4]
||
|| RangeOp<double, multiplies<double> >
||  r (1000, 1);
||   Product operation - note that de-
||   fault value has to be 1.  The de-
||   fault value is ALWAYS used as a
||   basis for range calculations; if
||   it were left as 0, all ranges
||   would return 0.
||
|| int min(const int &a, const int &b)
||  {return min(a,b);}
|| RangeOp<int> r(1000,2147483647,min);
||  The easiest way to specify your
||  own operator.  In this case, min.
||  Note the default value.
||
|| Memory use: RangeOp stores an inter-
|| nal array of T.  Its size is the
|| smallest power of 2 that is at least
|| twice the maximum number of elements.
|| (So, it's never larger than 4 times
|| a normal array size)
||
|| Time complexity: (n is the maximum
||  size of the RangeOp, m is the size
||  of any given range)
|| RangeOp constructor: O(n)
|| Calculating a range: O(log m)
|| Getting a value: O(1)
|| Setting a value: O(log n)
|| Setting a range: O(m log n), and
||  no worse than O(n)
|| Also, computing a series of distinct
|| ranges with endpoints that change
|| monotonically is never worse than
|| O(n).  That is, calculating ranges
|| that are sliding along the array
|| takes amortized CONSTANT time.
||
|| Things to watch for:
|| - Don't call calc with q < p.
|| - Always set the default value
||  properly.
|| - Updating values clears the internal
||  "cache", which clashes with the
||  above amortized-constant-time
||  guarantee.
|| - 2-D RangeOps would require some
||  additional annoying code (you'd
||  need to change RangeOp to support
||  ==, =, and a "lifted" version of
||  op)
||
18 template<class T, class Op = T(*)(const
0d  T&,const T&)> struct RangeOp {
e7  Op op;
d2  T *t, c[32], def;
af  int s, cs[32], ce[32], cv;
||
80  RangeOp(int n, T def=T(), Op op=Op())
06   : cv(0), def(def), op(op) {
c6   for (s=1; s<n; s<<=1);
20   t = new T[s*2];
2c  }
5a  ~RangeOp() {delete[] t;}
||
    // calculate operation on [p,q)
b3  T calc(int p, int q) {
4f   int a=p+=s,b=q+=s,d=0;
4b   for (; a!=b && (cv<=d||a!=cs[d]||
67    b!=ce[d]); d++)
63    a=(cs[d]=a)+1>>1, b=(ce[d]=b)>>1;
b1   if ((cs[d]=a)==(ce[d]=b)) c[d]=def;
5f   if (cv<d) cv=d;
38   while (d--) {
75    c[d]=((a<<=1)-1 << d<p ? c[d+1] :
e9     op(t[--a],c[d+1]));
d5    c[d]=((b<<=1)+1 << d>q ? c[d] :
fe     op(c[d],t[b++]));
92   }
94   return c[0];
f9  }
||
    // set a range of values at once
2f  template<class it> void set(int p,
c6   it i, it j) {
5a   int q=p+=s;
c2   for (; i!=j; ++i) t[q++]=*i;
c7   for (cv=0; q=q+1>>1,p>>=1;)
c7   for (int k=p; k<q; k++)
f4    t[k]=op(t[k*2],t[k*2+1]);
f2  }
||
    // read/write individual values
01  inline T get(int p) { return t[p+s]; }
||
45  void set(int p, T v) {
2f   for (cv=0, p+=s; t[p]=v, p>>=1;)
3d    if ((v=op(t[p*2],t[p*2+1]))==t[p])
40     break;
17  }
f8 };
|| ================ rangeoplars.h =================
|| rangeop.cc - Range query data struc-
|| ture Computes the value of an asso-
|| ciative operator over an arbitrary
|| range of an array in O(logN) time.
|| The tree is constructed bottom-up in
|| O(N) time and accessed or updated in-
|| crementally in O(logN) time. Requires
|| O(N) memory.
|| WARNING: you should initialize all
|| entries before using since the array
|| is padded.
||
54 template<class T, T(*op)(const T&,const
97  T&)> struct RangeOp {
b4  T* A; int nn, i, j;
||
32  RangeOp(int n) {
b1   for(nn=1;nn<n;nn*=2);
7a   A=new T[2*nn];
96   for(i=0;i<2*nn;i++) A[i] = T();
01  }
ac  ~RangeOp() { delete[] A; }
||
b8  template<class It>
6c  void set(int p, It s, It e) {
d3   for(copy(s,e,A+nn+p),i=nn-1; i; i--)
f8    A[i] = op(A[2*i],A[2*i+1]);
32  }
||
45  void set(int p, T v) {
4f   for (A[i=nn+p]=v; i;)
ed    i/=2, A[i]=op(A[2*i],A[2*i+1]);
a2  }
||
9e  T get(int p) { return A[nn+p]; }
||
    // interval is halfopen, i.e. [s,e)
61  T calc( int s, int e, T in ) {
b0   for(i=1; s+i<=e; i*=2) if(s&i)
f7    in = op(in,A[nn/i+s/i]), s+=i;
ee   for(; i; i/=2) if(s+i<=e)
89    in = op(in,A[nn/i+s/i]), s+=i;
b1   return in;
40  }
1b };
|| =================== rat.cc ===================
```

```
79 #include <stdio.h>
8c #include <stdlib.h>
20 #include <string.h>
||
df int gcd(int a,int b) {
9a  if (b == 0) return a>0?a:-a;
f7  return gcd(b,a%b);
4a }
||
9d class rat { public:
2a  int a, b;
14  rat () { a=b=0; };
7b  rat (int aa) { a=aa; b=1; };
a3  rat (int aa, int bb) { a=aa; b=bb; };
d9 };
||
39 rat operator + (rat x, rat y) {
ed  int t = gcd(x.b,y.b);
d1  return rat (y.b/t*x.a + x.b/t*y.a, x.b/t*y.b);
cb }
||
50 rat operator - (rat x) {
9d  return rat(-x.a, x.b);
99 }
||
59 rat operator - (rat x, rat y) {
0f  return x + - y;
fd }
||
29 rat operator * (rat x, rat y) {
e1  int s = gcd(x.a,y.b);
f0  int t = gcd(y.a,x.b);
a1  return rat((x.a/s) * (y.a/t), (x.b/t) * (y.b/s)
||                                                 );
a0 }
||
4a rat recip (rat x) {
5f  if (x.a < 0) return rat (-x.b, -x.a);
42  else return rat(x.b, x.a);
6f }
||
79 rat operator / (rat x, rat y) {
f9  return x * recip(y);
9a }
||
c7 char *pr (rat x) {
aa  char tmp[1000], *t;
ed  if (x.b == 0) {
64   if (x.a < 0) return "-inf";
52   if (x.a > 0) return "inf";
57   return "undef";
df  }
0d  sprintf(tmp,"%d/%d",x.a,x.b);
5f  t = (char *) malloc(strlen(tmp)+1);
96  strcpy(t,tmp);
c3  printf("printing %d %d\n",x.a,x.b);
ce  return t;
e3 }
||
d2 int main() {
72  int i,j,k,l;  char *q;
8a  while (4 == scanf("%d%d%d%d",&i,&j,&k,&l)) {
d1   rat x = rat(i,j);
64   rat y = rat(k,l);
33   printf("x.a %d x.b %d\n",x.a,x.b);
4a   q = pr(x);
b0   printf("x is %s\n",q);
0e   printf("%s %s %s %s %s %s\n",pr(x),pr(y),pr(x+
||                    y),pr(x-y),pr(x*y),pr(x/y));
ed  }
b3  return 0;
90 }
|| ================ redblack.cc =================
|| bugs: erase invalidates the red-black
|| invariant.  code it right if you care
79 #include <stdio.h>
8c #include <stdlib.h>
20 #include <string.h>
10 #include <assert.h>
46 #include <iostream>
ff using namespace std;
||
c5 #define FR(i,a,b) for (int i=a;i<b;i++)
9a #define FOR(i,n) FR(i,0,n)
||
1d template <typename T> struct avl {
a3  typedef avl<T>*A;
4a  T me;
a1  A k[2];
db  bool red;
5e #define Z(i,x) (k[i] ? k[i]->x : 0)
5a #define ZZ(i,j,x) Z(i]->k[j, x)
||
d5  void upd() {}
||
```

```
89  avl(T a) { me=a; k[0]=k[1]=0; red=1; }
91  ~avl() { FOR(i,2) if(k[i])delete k[i];}
||
3a  A rot(int d) {
98   A l = k[d];
6b   k[d] = l->k[!d];
26   l->k[!d] = this;
ee   upd(); l->upd();
52   return l;
8f  }
||
bc  A reb() {
54   if (Z(0,red) && Z(1,red) && !red)
21    red = 1, k[0]->red=k[1]->red=0;
ad   else FOR(i,2) if (Z(i,red)) {
33    if (ZZ(i,i,red)) {
49     k[i]->k[i]->red = 0;
d9     return rot(i);
3b    }
7e    if (ZZ(i,!i,red)) {
f3     k[i]->red = 0;
9e     k[i] = k[i]->rot(!i);
32     return rot(i);
04    }
ed   }
14   return this;
b5  }
||
    //optional, but possibly useful:
||
dc  A insert(T key) {
bd   return this ? k[key>me] = k[key>me]->
ec    insert(key), reb() : new avl(key);
ba  }
||
00  A ins(T key) {
13   if (this) red=0;
61   A x=insert(key);
6e  }
||
49  pair<A,A> erasemax() {
dc   if(!k[1]) return make_pair(this,k[0]);
47   pair<A,A> x = k[1]->erasemax();
fe   k[1] = x.second;
5a   return make_pair(x.first, this);
d7  }
||
5e  A erase(T key) {
2a   if (!this) return 0;
7a   if (me==key) {
8a    FOR(i,2) if (!k[i]) return k[!i];
2f    pair<A,A> xx = k[0]->erasemax();
14    k[0] = xx.second; A x = xx.first;
6a    FOR(i,2) x->k[i] = k[i];
5b    return x->reb();
bf   }
f7   k[key>me] = k[key>me]->erase(key);
87   return reb();
c1  }
||
0b  A find(T key) {
7e   return this ? me == key ? this :
38    k[key>me]->find(key) : 0;
44  }
53 #undef Z
af #undef ZZ
db };
||
4a template <typename T> ostream&
26  operator<<(ostream&o, avl<T>*t) {
21  if (!t) return o;
57  return o<<"("<<t->k[0]<<t->me<<t->k[1]
63   << ")";
56 }
||
d2 int main() {
dd  avl<int> *tree = 0;
2d  while (1) {
6d   int k; char foo[512]; gets(foo);
e4   string f(foo);
64   sscanf(foo, "%*s %i", &k);
0e   if (f=="print") cout << tree << endl;
3a   if (f=="insert") tree = tree->ins(k);
9f   if (f=="erase")  tree =tree->erase(k);
18   if (f=="find")
45    printf("%i\n", !!tree->find(k));
a6   if (f=="die") break;
d8  }
e3 }
|| ================== restore.cc ==================
|| ======= Restore the number given its reminders =
||                                          =======
||
|| Requires the following operations over "bi":
|| 1) bi + bi 2) bi * int 3) bi % int
```

```
12 bi restore(vi primes,vi mods){
c6  bi res=0,p=1;
96  REP(i,primes.size()){
ac   ll temp=(mods[i]-(res%primes[i])+primes[i])%
||                                       primes[i];
37   temp=(temp*inv(p%primes[i],primes[i]))%primes[
||                                              i];
b6   res=res+p*temp;
21   p=p*primes[i];
ac  }
fb  return res;
92 }
|| ================ seg_tree_1.cc =================
|| =============== Segment Tree #1 ================
||                                             ====
|| OPERATIONS: clear, setOne, getLastOne
||
9c struct Node{
0e  bool clear,has;
17  int l,r,add;
b4  Node *lf;
c0  Node *rg;
20  Node(){lf=rg=0;clear=0;has=add=0;}
14 };
||
83 Node *buildTree(int l,int r){
56  Node *curr = new Node();
28  curr->l = l;
cb  curr->r = r;
74  if(l!=r){
e1   curr->lf = buildTree(l,(l+r)/2);
a4   curr->rg = buildTree((l+r)/2+1,r);
e8  }
e3  return curr;
12 }
||
1a void update(Node *curr){
95  curr-> has = !curr->clear &&
b9   (curr->add || curr->lf && !curr->lf->clear &&
||                                   curr->lf->has
ea   || curr->rg && !curr->rg->clear && curr->rg->
||                                            has);
4a }
||
a5 void push(Node *curr){
99  if(curr->clear){
50   curr->has = curr->add = curr->clear=0;
74   if(curr->l != curr->r) curr->lf->clear=curr->
||                                     rg->clear=1;
1e  }
bd  if(curr->add && curr->l != curr->r){
c4   curr->lf->add=1;curr->lf->clear=0;
82   curr->rg->add=1;curr->rg->clear=0;
6f   curr->add=0;
32   update(curr->lf);
bc   update(curr->rg);
6a   update(curr);
1e  }
e5 }
||
df void clear(Node *curr,int from,int to){
a3  push(curr);
6d  if(to < curr->l || curr->r < from) return;
55  if(from <= curr->l && curr->r <= to) curr->
||           clear = 1, curr->add = curr->has = 0;
92  else clear(curr->lf, from, to), clear(curr->rg,
||                                       from, to);
7d  update(curr);
96 }
||
37 void setOne(Node *curr,int from,int to){
42  push(curr);
ae  if(to < curr->l || curr->r < from) return;
1b  if(from <= curr->l && curr->r <= to) curr->add
||                                              ++;
a8  else setOne(curr->lf, from, to), setOne(curr->
||                                   rg, from, to);
60  update(curr);
1e }
||
5e int getLastOne(Node *curr,int from,int to){
ce  push(curr);
64  if(to < curr->l || curr->r < from) return -1;
2f  if(!curr->has) return -1;
eb  if(curr->l == curr->r) return curr->l;
14  int p = getLastOne(curr->rg, from, to);
eb  if(p!=-1) return p;
b3  return getLastOne(curr->lf, from, to);
08 }
|| ================ seg_tree_2.cc =================
|| =============== Segment Tree #2 ================
||                                             ====
|| OPERATIONS:
||  clear  - clears values on a segment
||  addVal - adds a value to every element on the
||                                          segment
||  getSum - gets a sum on the segment
||  getSumSum(a,b) - sum getSum(0,i) where a<=i<=b
||
||
9c struct Node{
7e  bool clear;
0c  int l,r;
5a  ll add,sum,sumsum;
d9  Node *lf;
8d  Node *rg;
f7  Node(){lf=rg=0;clear=0;sumsum=sum=add=0;}
ef };
||
83 Node *buildTree(int l,int r){
56  Node *curr = new Node();
28  curr->l = l;
cb  curr->r = r;
74  if(l!=r){
e1   curr->lf = buildTree(l,(l+r)/2);
a4   curr->rg = buildTree((l+r)/2+1,r);
e8  }
e3  return curr;
12 }
||
43 ll getSum(Node *curr){return curr->clear ? 0 :
||  curr->sum + curr->add * (curr->r - curr->l + 1)
||                                               ;}
||
d5 ll getSumSum(Node *curr){return curr->clear ? 0
||  : curr->sumsum + curr->add * (curr->r - curr->l
||                 + 2)*ll(curr->r - curr->l+1)/2;}
||
1a void update(Node *curr){
fa  curr->sum = 0;
72  if(curr->lf) curr->sum = getSum(curr->lf) +
||                                getSum(curr->rg);
||
7d  curr->sumsum = 0;
fe  if(curr->lf) curr->sumsum = getSumSum(curr->lf)
||    + getSumSum(curr->rg) + getSum(curr->lf) * (c
||                  urr->rg->r - curr->rg->l + 1);
c2 }
||
3e void clear(Node *curr){
02  if(curr->clear){
dd   curr->sum = curr->sumsum = curr->add = curr->
||                                         clear=0;
43   if(curr->l != curr->r) curr->lf->clear=curr->
||                                     rg->clear=1;
cf  }
bb }
2e void push(Node *curr){
ad  clear(curr);
61  if(curr->add && curr->lf){
17   clear(curr->lf);
1d   clear(curr->rg);
d9   update(curr->lf);
2b   update(curr->rg);
85   curr->lf->add+=curr->add;curr->rg->add+=curr->
||                                             add;
1f   curr->add=0;
3b   update(curr);
54  }
8f }
||
df void clear(Node *curr,int from,int to){
a3  push(curr);
6d  if(to < curr->l || curr->r < from) return;
e8  if(from <= curr->l && curr->r <= to) curr->
||           clear = 1, curr->add = curr->sum = 0;
49  else clear(curr->lf, from, to), clear(curr->rg,
||                                       from, to);
10  update(curr);
c5 }
||
55 void addVal(Node *curr,int from,int to, ll val){
4c  push(curr);
b3  if(to < curr->l || curr->r < from) return;
da  if(from <= curr->l && curr->r <= to) curr->add
||                                          += val;
7a  else addVal(curr->lf, from, to, val), addVal(
||                       curr->rg, from, to, val);
89  update(curr);
79 }
||
51 ll getSum(Node *curr,int from,int to){
4e  push(curr);
06  if(to < curr->l || curr->r < from) return 0;
16  if(from <= curr->l && curr->r <= to) return
||                                    getSum(curr);
dc  return getSum(curr->lf, from, to) + getSum(curr
||                                ->rg, from, to);
f3 }
||
```

```
db ll getSumSum(Node *curr,int from,int to){
e1  push(curr);
ba  if(to < curr->l) return 0;
99  if(curr->r < from) return getSum(curr)*(to-from
||                                             +1);
8f  if(from <= curr->l && curr->r <= to) return
||   getSumSum(curr) + getSum(curr) * (to - curr->r
||                                               );
de  return getSumSum(curr->lf, from, to) +
||                   getSumSum(curr->rg, from, to);
4b }
|| ================== simplex.cc ==================
|| ===========  Simplex-Method ==================
||
52 const double eps = 1e-3;
f2 const double SM_INF = 1e6; // objective function
||             coefficient for artificial variables
ae typedef vector<double> vd;
77 struct SimplexM{
ef  int startN, N;
f4  vd old_c,c,b,res;
b1  vector<vd> system;
9d  double value;
6f  SimplexM(vd c):N(c.size()),c(c){ } // objective
||                              function (minimize)
71  void add(vd r,double v,int sgn=0){ // v >= 0
7a   for(;c.size()<r.size();c.pb(0));
75   for(;r.size()<c.size();r.pb(0));
14   if(sgn)r.pb(-sgn),c.pb(0);
92   system.pb(r),b.pb(v);
51  }
4f  void refresh_c_func(int pos,int row){
c0   double begin=c[pos];
fe   REP(i,N) c[i]-=begin*system[row][i];
5a  }
48  void gauss(int row,int col){
71   double v=system[row][col];
e1   REP(i,N) system[row][i]/=v;
39   b[row]/=v;
0d   REP(i,system.size()) if(i!=row){
6a    v=system[i][col];
b5    REP(j,N) system[i][j]-=v*system[row][j];
d3    b[i]-=v*b[row];
1b   }
9d  }
    // -2 - NO SOLUTION
    // -1 - minimum is infinity (minimum does not
||                                          exists)
    // 1 - OK
6f  int solve(){
2d   N=c.size(),old_c=c;
ee   int m=system.size();
5a   REP(i,m)system[i].resize(N);
b8   startN=N;
22   vi base(m,-1);
fe   REP(i,N){
1a    int nm=0,pos=-1;
17    REP(j,m) if(fabs(system[j][i])>eps) nm++,pos=
||                                               j;
4b    if(nm==1 && fabs(system[pos][i]-1)<eps &&
||                                   base[pos]==-1)
b0     base[pos]=i;
40   }
de   REP(i,m) if(base[i]==-1){
eb    N++;
d5    c.pb(SM_INF);
20    REP(j,m) system[j].pb(j==i?1:0);
4b    base[i]=N-1;
1f   }
e1   REP(i,m) refresh_c_func(base[i],i);
cc   while(1){
53    int pos=-1,row=-1;
8b    REP(i,N) if(c[i]<-eps && (pos==-1 || c[i]<c[
||                                    pos])) pos=i;
fd    if(pos==-1) break;
9c    double min_teta=0;
28    REP(i,m)
43     if(system[i][pos]>eps &&
af      (row==-1 || min_teta>b[i]/system[i][pos]))
6f       row=i,min_teta=b[i]/system[i][pos];
81    if(row==-1) return -1;
24    base[row]=pos;
46    gauss(row,pos);
da    refresh_c_func(pos,row);
fb   }
b5   REP(i,m) if(base[i]>=startN && fabs(b[i])>eps)
||                                       return -2;
ae   res.resize(startN,0);
ae   REP(i,m) if(base[i]<startN) res[base[i]]=b[i];
15   value=0;
dc   REP(i,startN) value+=res[i]*old_c[i];
20   return 1;
5c  }
38 };
|| ================== splay.cc ===================
```

```
|| ============ Splay tree =====================
||
13 template<class T,void F(T&,const T&,const T&)>
||                               struct splay_tree{
5c  vector<T> a;
c0  vi l,r,p,q;
9a  vector<char> rev;
7e  int root,n;
68  splay_tree(const vector<T>&a):a(a){
7b   n=a.size(),l=r=p=vi(n),q=vi(n,1),q[0]=0;
45   rev.resize(n),root=build(1,--n);
ff  }
5f  void change(int x){ if(x)q[x]=1+q[l[x]]+q[r[x]]
||                      ,F(a[x],a[l[x]],a[r[x]]); }
ab  void upd(int x){
d9   if(x && rev[x])
c7    swap(l[x],r[x]),rev[x]=0,
51    rev[l[x]]^=1,rev[r[x]]^=1;
af  }
4d  int build(int i,int j){
28   int x=i>j?0:i+j>>1;
eb   if(i<j)p[l[x]=build(i,x-1)]=p[r[x]=build(x+1,j
||                                            )]=x;
36   return change(x),x;
64  }
25  void pop(int x){
98   if(!x) return;
cb   bool sw=false;
92   for(int y;y=p[x];){
4c    if(upd(x),x!=l[y])sw^=1,l.swap(r);
bd    (l[p[x]=p[y]]==y?l:r)[p[y]]=x;
1c    change(p[l[y]=r[x]]=y);
91    change(p[r[x]=y]=x);
c1   }
16   root=x; if(sw)l.swap(r);
1d  }
39  int find(int k){
3f   if(k<0 || k>=n) return 0;
65   for(int x=root,y;;)
4d    if(upd(x),k<q[y=l[x]])x=y;
c0    else if((k-=q[y]+1)<0) return x;
a6    else x=r[x];
b0  }
a3  void rv(int i,int j,int x){
f1   if(i<1 && j+2>q[x]) rev[x]^=1; else
7b   if(j>=0 && i<q[x])upd(x),rv(i,j,l[x]),rv(i-q[l
||                        [x]]-1,j-q[l[x]]-1,r[x]);
ee  }
ac  T sum(int i,int j,int x){
c0   if(i<1 && j+2>q[x]) return a[x];
2b   int z=q[upd(x),l[x]];
2f   T res=a[i<=z && z<=j?x:0];
ef   if(j>=0 && i<q[x])F(res,sum(i,j,l[x]),sum(i-z-
||                                  1,j-z-1,r[x]));
eb   return res;
3a  }
da  void change(int x,const T&v){ if(x=find(x))for(
||                    a[x]=v;x;change(x),x=p[x]); }
b9  const T& operator[](int i){ return a[find(i)];
||                                                }
f4  T operator()(int i,int j){ return sum(i,j,root)
||                                              ; }
01  void reverse(int i,int j){ pop(find(i-1)),pop(
||                       find(j+1)),rv(i,j,root); }
70 };
|| ============== stable_marriage.c ===============
||  STABLE MARRIAGES
||  ----------------
||
||  We have n men and n women.  Each person has a
||                           preference list of the
||  folks of the opposite gender. A pair of people
||                              of opposite genders
||  that like each other better than their
||                         respective spouses is an
||  instability.
||  This algorithm finds a match with no
||                                   instabilities.
||
||  INPUT: preferences
||  The arrays (and the men, and the women) are
||                                indexed from 1..n
||  men[i][m] = j means woman m is the jth choice
||                                         of man i
||  wom[i][m] = j means man m is the jth choice of
||                                          woman i
||
||  OUTPUT: match
||  The arrays are indexed from 1..n
||  meng[i] is the ID of the woman engaged to the
||                                          ith man
||  weng[i] is the ID of the man engaged to the ith
||                                            woman
||
||
```

```
79 #include <stdio.h>
69 #define MAXN 300
||
0a int n, men[MAXN][MAXN], wom[MAXN][MAXN], meng[
||                               MAXN], weng[MAXN];
||
6b void stable_marriage() {
f9  int i,j,done,best;
||
0b  for (i=1;i<=n;i++) meng[i] = weng[i] = 0;
6e  do {
4b   done = 1;
bd   for (i=1;i<=n;i++) {
9f    if (meng[i]) continue;
ae    best = 0;
d8    for (j=1;j<=n;j++) {
b6     if ((!best || men[i][j] < men[i][best])
a4      && (!weng[j] || wom[j][i] < wom[j][weng[j]]
||                                     )) best = j;
c8    }
e9    if (weng[best])  meng[weng[best]] = 0;
7a    weng[best] = i;
03    meng[i] = best;
66    done = 0;
7e   }
d0  } while (!done);
ab }
||
|| Example
||
d5 void check() {
13  int i,j;
14  for (i=1;i<=n;i++) for (j=1;j<=n;j++) {
5c   if (men[i][j] < men[i][meng[i]] && wom[j][i] <
||                               wom[j][weng[j]]) {
e5    printf("oops man %d would prefer woman %d\n",
||                                           i,j);
a9    printf("man married to %d; woman to %d\n",
||                                meng[i],weng[j]);
20   }
1c  }
6d }
||
d2 int main() {
a7  int t,i,j,m;
69  scanf("%d",&t);
ab  while (t--) {
64   scanf("%d",&n);
c2   for (i=1;i<=n;i++) meng[i] = weng[i] = 0;
b6   for (i=1;i<=n;i++) for (j=1;j<=n;j++) {
06    scanf("%d",&m);
30    men[i][m] = j;
74   }
da   for (i=1;i<=n;i++) for (j=1;j<=n;j++) {
92    scanf("%d",&m);
64    wom[i][m] = j;
1e   }
||
ff   stable_marriage();
||
53   for (i=1;i<n;i++) printf("%d ",meng[i]);
76   printf("%d\n",meng[n]);
||
c6   check();
73  }
f2  return 0;
62 }
|| ================= strongly.cc ==================
|| Strongly-connected components of a
|| directed graph
||
|| Two nodes a and b are in the same SCC
|| provided there is a path from a to b
|| and a path from b to a. This code
|| assumes the graph is stored in the
|| sparse representation
|| (nv,ne,e,firste). The SCCs are
|| computed. Runtime is linear in V+E.
||
79 #include <stdio.h>
8c #include <stdlib.h>
20 #include <string.h>
37 #include <vector>
bd #include <algorithm>
1e using namespace std;
||
97 #define MAXV 1000000
7a #define MAXE 2000000
||
0a #define FR(i,a,b) \
8f  for(int i=(a);i<(b);i++)
c8 #define FOR(i,n) FR(i,0,n)
6c #define RF(i,a,b) \
ba  for(int i=(a)-1;i>=(b);i--)
01 #define ROF(i,n) RF(i,n,0)
```

```
59 #define PB push_back
||
90 struct ee {
db  int from, to;
f7 } e[MAXE];
||
65 int firste[MAXV], nv, ne, val[MAXV];
5c int nextid, sp, s[MAXV];
dc vector<vector<int> > sccs;
||
19 int doscc(int me) {
ec  int i,j,lowp,v;
83  val[me] = lowp = nextid++;
f6  s[sp++] = me;
||
cc  for(i=firste[me];e[i].from==me;i++) {
02   v = e[i].to;
21   if(val[v]) j = val[v];
67   else j = doscc(v);
3e   if(j<lowp) lowp=j;
74  }
||
0e  if(lowp==val[me]) {
cf   vector<int> foo;
e6   do {
02    foo.PB(s[−−sp]);
f0    val[s[sp]]=MAXV+1;
85   } while (s[sp]!=me);
d2   sccs.PB(foo);
03  }
||
d2  return lowp;
9b }
||
|| find all SCCs.
17 void scc() {
61  int i;
91  memset(val,0,sizeof(val));
5e  sp = 0;
af  nextid = 1;
db  FOR(i,nv) if(!val[i]) doscc(i);
43 }
||
5b int comp(const ee&a, const ee&b) {
ed  if (a.from==b.from) return a.to<b.to;
3c  return a.from < b.from;
9c }
||
d2 int main() {
    // sample mainline −− reads number of
    // vertices, number of edges, and
    // edgelist; prints out strongly
    // connected components, one per line.
b3  int i,j,k;
8e  while (2==scanf("%d %d ",&nv,&ne)) {
     //read graph
b5   FOR(i,ne)
41    scanf("%d %d",&e[i].from,&e[i].to);
||
28   e[ne].from = e[ne].to = MAXV+1;
fc   sort(e,e+ne,comp);
42   ROF(i,ne) firste[e[i].from] = i;
||
b0   printf("======================\n");
ba   scc();
d0  }
11  return 0;
7d }
||
|| ================== strstr.c ==================
|| linear time strstr()
79 #include <stdio.h>
7d #include <assert.h>
a3 #include <stdlib.h>
c0 #include <string.h>
14 #include <sys/types.h>
||
49 char*fast_strstr(char*hay, char*ndl) {
9f  char *v;
24  size_t i, j = 0, p;
||
c5  while (1) {
d7   for (i = j+1, p = 1; ndl[i]; i++) {
53    if (ndl[i] < ndl[i−p]) p = i−j;
7a    else if (ndl[i] > ndl[i−p]) break;
c6   }
b7   if (ndl[i]) j = i − (i − j) % p;
6a   else break;
a6  }
89  v = ndl + j;
    // v points to the lexicographically
    // largest suffix of ndl
||
b0  size_t ul = v − ndl, vl = strlen(v);
d2  size_t i = ul, j = 0, p = 1, pre = 0;
```

```
81  size_t hl = strlen(hay);
||
    // match v against everything and
    // check for the prefix everywhere
    // that matters.
c5  while (1) {
8e   while (j < vl && v[j] == hay[i+j])
e1    if (++j > p && v[j] − v[j−p]) p=j;
95   if (j == vl && i−pre >= ul) {
2d    if (!memcmp(hay+i−ul, ndl, ul))
ea     return hay+i−ul;
1c    pre = i;
dd   }
e3   else if (!hay[i+j]) goto retnull;
f6   i += p;
e9   if (j >= p+p) j −= p;
72   else j=0, p=1;
16  }
||
4f  retnull:
a5  return NULL;
10 }
||
ff int main() {;}
|| ============== suffix_array.cc ===============
|| ========== Suffix Array ============
||
c8 int n,f[N],p[N],q[N],L[N+N];
fb bool b[N];
74 int lcp(int i,int j){
b4  if(i>j)swap(i,j);
e1  int res=n;
bf  for(i+=n,j+=n−1;i<=j;i=i+1>>1,j=j−1>>1)
61   res=min(res,min(L[i],L[j]));
88  return res;
89 }
cf void setlcp(int i,int v){ for(i+=n;L[i]>v;i/=2)L
||                                         [i]=v; }
ad bool scmp(int x,int y){ return f[x]<f[y]; }
e7 void suff_sort(){
78  REP(i,n)b[i]=0,p[i]=i; b[n]=true;
66  for(int i,j,x,h=0;h<n;){
cb   for(i=j=0;j<=n;++j)if(b[j])sort(p+i,p+j,scmp),
||                                             i=j;
5d   for(i=0,j=1;j<n;++i,++j)b[j]|=f[p[i]]!=f[p[j]]
||                                                ;
75   for(h=h?h*2:1,i=x=0;i<n;f[j]=x+=b[i++])if((j=p
||                                   [i]−h)<0)j+=n;
86  }
11 }
a9 void suff_sort_withlcp(){
f7  REP(i,n)b[i]=0,p[i]=i,L[i+1]=L[i+n]=n; b[n]=
||                                           true;
f1  for(int i,j,x,y,h=0;h<n;){
2f   for(i=j=0;j<=n;++j)if(b[j])sort(p+i,p+j,scmp),
||                                             i=j;
d5   REP(i,n)q[p[i]]=i;
18   for(i=0,j=1;j<n;++i,++j)
90    if(!b[j] && f[p[i]]!=f[p[j]])
e6     b[j]=true,x=p[i]+h,y=p[j]+h,
22     setlcp(i,h?h+lcp(q[x<n?x:x−n],q[y<n?y:y−n]):
||                                              0);
87   for(h=h?h*2:1,i=x=0;i<n;f[j]=x+=b[i++])if((j=p
||                                   [i]−h)<0)j+=n;
b9  }
b0 }
|| ================ suffixarray.cc ===============
a2 #include "suffixarray.h"
||
e3 int main(void)
c4 {
c6  char str[100];
7b  int sarray[100], lcp[100];
58  int i;
||
20  while (scanf("%s", str) == 1) {
07   build_sarray(str, sarray, lcp);
64   for (i = 0; i < strlen(str); i++) {
89    printf("%3d: %2d, %s\n",
6b     i, lcp[i], str+sarray[i]);
2e   }
ac  }
b1  return 0;
98 }
|| ================ suffixarray.h ================
|| Suffix array
|| Author: Howard Cheng
||
|| The build_sarray routine takes in a
|| null−terminated n−character string
|| and constructs two arrays sarray and
|| lcp.  Their properties are:
|| − If p = sarray[i], then the suffix
||  of str starting at p (i.e.
||  str[p..n−1] is the i−th suffix when
```

```
||  all the suffixes are sorted in lex
||  order. NOTE: the empty suffix is
||  not included, so sarray[0] != n.
|| − lcp[i] contains the length of the
||  longest common prefix of the suf−
||  fixes pointed to by sarray[i−1] and
||  sarray[i].  lcp[0] = 0.
|| − To see whether a pattern P occurs
||  in str, you can look for it as the
||  prefix of a suffix. This can be
||  done with a binary search in
||  O(|P| log n) time.
|| The construction of the suffix array
|| takes O(n log n) time.
||
79 #include <stdio.h>
8c #include <stdlib.h>
20 #include <string.h>
65 #include <limits.h>
6d #include <assert.h>
||
0a #define FR(i,a,b) \
8f  for(int i=(a);i<(b);i++)
c8 #define FOR(i,n) FR(i,0,n)
||
44 void build_sarray(char*stir, int*sarray,
43  int*lcp) {
ef  unsigned char*str=(unsigned char*)stir;
||
d9  int n=strlen(stir), d, e, f, h, i, j,
89   l, prm[n], count[n], bucket[256];
83  char bh[n+1];
||
8d  FOR(a,256) bucket[a] = −1;
46  FOR(i,n)
c9   prm[i] = bucket[j = str[i]],
0e   bucket[j] = i;
1a  h = 0;
12  FOR(a,256)
f4   for (i = bucket[a]; i + 1; i = j)
35    j = prm[i], prm[i] = h,
d7    bh[h++] = (i == bucket[a]);
88  bh[n] = 1;
13  FOR(i,n) sarray[prm[i]] = i;
||
a2  l = 0;
46  for (h = 1; h < n; h *= 2) {
d2   FOR(i,n) {
1b    if (bh[i] & 1) count[l=i] = 0;
34    prm[sarray[i]] = l;
14   }
||
a9 #define E(x) e=x,bh[x=e+count[e]++]|=2;
15  E(prm[n−h])
||
93 #define F for (j=i; j==i || !(bh[j]&1)\
f4  && j<n; j++) if ((d=sarray[j]−h) >= 0
||
6e   for (i=0;i<n;i=j) {
19    F) E(prm[d])
0a    F && bh[prm[d]] & 2)
3d     for (f=prm[d]+1;bh[f]==2;f++)
b8      bh[f] &= 1;
88   }
||
e5   FOR(i,n) {
4c    sarray[prm[i]] = i;
25    if (bh[i] == 2) bh[i] = 3;
0c   }
3d  }
||
aa  h = 0;
b1  FOR(i,n) {
27   if ((e = prm[i]) > 0) {
94    j = sarray[e−1];
56    while (str[i+h] == str[j+h]) h++;
39    lcp[e] = h;
4d    if (h > 0) h−−;
ef   }
02  }
9c  lcp[0] = 0;
c1 }
|| ================ suffix_tree.cc ================
|| Lars’ implementation of Ukkonen’s
|| O(n) online algorithm for suffix tree
|| construction.  The map could be
|| replaced with an array if the alpha−
|| bet is small (e.g. binary), but
|| otherwise saving memory is usually
|| more important than the log(alphabet
|| size) slowdown.
||
47 #include <iostream>
2a #include <map>
d2 #include <string>
```

```
d8 using namespace std;
60 #define FOR(i,n) for (int i=0;i<n;i++)
||
54 const int INF = 1000000000;
51 const int ALPH_SIZE = 128;
da struct Node;
||
4a struct Edge {
14  int s, e;
0d  Node* node;
5f  Edge() {}
44  Edge(int s, int e, Node* n)
57   : s(s), e(e), node(n) {}
e9 };
||
9c struct Node {
63  Node* link;
93  map<char,Edge> tr;
f9  Node() { link=0; }
4b };
||
|| s and k correspond to the node and
|| position of the active point
04 Node *root, *s;
d2 string text;
c8 int k;
||
|| Push s down as far as possible while
|| remaining above the suffix at text[p]
00 void canonize(Node*& s, int& k, int p) {
de  for(;;) {
b2   if (k > p) break;
73   Edge& e = s−>tr[text[k]];
91   if (e.e − e.s > p − k) break;
f9   k += e.e − e.s + 1;
b8   s = e.node;
a0  }
68 }
||
|| Update tree for suffix at text[p]
5c void update(int p) {
b7  Node* orig = root;
74  for(;;) {
27   Node* r = s;
70   if (k < p) {
7a    Edge& e = s−>tr[text[k]];
1b    int ofs = e.s + p − k;
      // if no need to split
e0    if (text[p] == text[ofs]) break;
36    r = new Node();
95    r−>tr[text[ofs]]
c9     = Edge(ofs, e.e, e.node);
c3    e.node = r;
7a    e.e = ofs−1;
76   }
83   else if (s−>tr.count(text[p]))break;
72   r−>tr[text[p]]
8f    = Edge(p,INF,new Node());
90   if (orig != root) orig−>link = r;
9e   orig = r;
29   s = s−>link;
80   canonize(s,k,p−1);
f9  }
b8  if (orig != root) orig−>link = s;
79  canonize(s,k,p);
c6 }
||
59 void init() {
0a  Node *bot = new Node();
3e  root = s = new Node();
e9  root−>link = bot;
f0  FOR(i,text.size())
77   if(!bot−>tr.count(text[i]))
9b    bot−>tr[text[i]] = Edge(i,i,root);
e5  k = 0;
ba }
||
c8 void dump( Node* n, int lev, int p ) {
6e  if (!n) return;
8f  FOR(i,ALPH_SIZE) {
12   if (!n−>tr.count(i)) break;
6f   FOR(j,lev) cout <<" ";
97   int a=n−>tr[i].s, b=n−>tr[i].e<?p;
27   cout << text.substr(a,b−a+1) <<’\n’;
1c   dump(n−>tr[i].node,lev+1,p);
2d  }
6c }
||
d2 int main() {
0b  text = "MISSISSIPPI";
71  init();
1c  FOR(i,text.size()) {
0b   update(i);
5c   cout << "after " << text[i] << ’\n’;
1f   dump(root,0,i);
dd  }
9b }
||
|| ================ template.java =================
cd ========    Java Template   ===========
||
a0 public class Main
4e {
92  BufferedReader reader;
a9  public Main() {
24   reader = new BufferedReader(new FileReader(ˆ
||                                      gold.in˜));
6b   int n = Integer.parseInt( reader.readLine() );
be  }
b7  public static void main(String[] args) throws
||                                      Exception {
87   new Main();
a1  }
98 }
|| ================= transport.c ==================
||  Transportation problem.
||
||  Example of mincost network flow.
||
||  Input:
||   m - number of suppliers  (on separate line)
||   n - number of consumers  (on separate line)
||   supplier capacities (m integers on a line)
||   consumer demands (n integers on a line)
||   transport costs  (m lines with n integers each
||                                                )
||
||  Output:
||   max flow - total amount of commodity delivered
||   min cost - min transport cost
||   dump - dump of optimal flow network
||
ae #include "flowlite.h"
b5 #include <string.h>
f4 #include <stdio.h>
e0 void dump(int n, int src, int snk, int mx[][SZ],
||                                   int fl[][SZ]){
15  int i,j;
c1  printf("dump:\n");
e1  for (i=0;i<n;i++) for (j=0;j<n;j++) if (mx[i][j
||                                         ] > 0) {
47   printf("from %d to %d min %d max %d flow %d\n"
||                                                ,
3e    i,j,-mx[j][i],mx[i][j],fl[i][j]-fl[j][i]);
b4  }
e0 }
||
9a int Max[SZ][SZ], Cost[SZ][SZ], Flow[SZ][SZ];
||
88 int i,j,k,m,n,t,source=0,sink=1;
||
d2 int main(){
c1  memset(Max,0,sizeof(Max));
cf  memset(Flow,0,sizeof(Flow));
b6  memset(Cost,0,sizeof(Cost));
be  scanf("%d%*[^\n]",&m);
5d  scanf("%d%*[^\n]",&n);
6d  for (i=0;i<m;i++) scanf("%d",&Max[source][2+i])
||                                                ;
d3  scanf("%*[^\n]");
63  for (i=0;i<n;i++) scanf("%d",&Max[2+m+i][sink])
||                                                ;
8b  scanf("%*[^\n]");
3a  for (i=0;i<m;i++) {
2b   for (j=0;j<n;j++) {
f5    scanf("%d",&Cost[2+i][2+m+j]);
51    Max[2+i][2+m+j] = Max[source][2+i];
8f   }
4c   scanf("%*[^\n]");
4f  }
||
c3 #ifdef USEMAXFLOW
a4  i = maxflow(m*n+2,source,sink,Max,Flow);
ea  printf("maxflow %d\n",i);
7d  i = mincost(m*n+2,Max,Cost,Flow);
39 #else
33  Max[sink][source] = 0x7fffffff;
dd  Cost[sink][source] = -10000;     // bigger
||                       magnitude than input costs
86  i = mincost(m*n+2,Max,Cost,Flow);
b2  i -= (Flow[sink][source]-Flow[source][sink]) *
||             -10000;  // always subtract opposing
f4 #endif
3f  printf("min cost %d\n",i);
c9  dump(m*n+2,source,sink,Max,Flow);
eb }
|| =================== trie.cc ===================
|| Simple Trie Code
||
|| Solves "Double Linear Crossword" -
```

```
|| build a string by concatenating words
|| from a dictionary in two different
|| ways such that the two derivations do
|| not share any word boundary.
||
79 #include <stdio.h>
8c #include <stdlib.h>
20 #include <string.h>
||
92 int l, n, nt = 2;
23 char d[1000][21];
97 int trie[22000][26];
3b char leaf[22000], used[22000];
99 char q[1000], loq[1000];
||
f8 int solve(int k1, int k2, int s) {
77  int i,j,k;
22  if (!k1 || !k2) return 0;
38  if (s == l) {
a9   if (k1 != k2 && leaf[k1] && leaf[k2]
8a    && !used[k1] && !used[k2]) {
95    if (!loq[0] || strcmp(q,loq) ==-1)
7f     strcpy(loq,q);
3d    return 1;
44   }
2d   return 0;
dc  }
3a  if (loq[0] && strcmp(q,loq) > 0)
65   return 0;
58  for (i=0; i<26; i++) {
a5   q[s] = i+’a’;
9d   solve(trie[k1][i], trie[k2][i],s+1);
6b   if (leaf[k1] && !used[k1]) {
8e    used[k1] = 1;
67    solve(trie[1][i],trie[k2][i],s+1);
36    used[k1] = 0;
6c   }
6e   if (leaf[k2] && !used[k2] && k2-k1){
9a    used[k2] = 1;
18    solve(trie[k1][i],trie[1][i],s+1);
86    used[k2] = 0;
11   }
81   q[s] = 0;
d1  }
ce  return 0;
73 }
||
d2 int main(){
23  int i,j,k;
4e  scanf("%d%d\n",&l,&n);
c0  for (i=0;i<n;i++) {
e5   gets(d[i]);
12   k = 1;
40   for (j=0; d[i][j]; j++) {
fc    if (!trie[k][d[i][j]-’a’])
19     trie[k][d[i][j]-’a’] = nt++;
2a    k = trie[k][d[i][j]-’a’];
6c   }
29   leaf[k] = 1;
3c  }
d5  q[0] = loq[0] = 0;
51  solve(1,1,0);
49  if (loq[0]) printf("%s\n",loq);
b5  else printf("NO SOLUTION\n");
f0 }
|| ================== turtle2.h ==================
|| 3D turtle graphics
|| Coordinate system:
||  x is forward, y is up, z is right.
94 #include <valarray>
06 #include <assert.h>
f9 using namespace std;
11 #define ld long double
28 #define FOR(i,n) for (int i=0;i<n;i++)
||
90 struct xyz : public valarray<ld> {
fe  xyz() : valarray<ld>() { resize(3); }
fc  xyz(ld a,ld b,ld c):valarray<ld>() {
b7   resize(3); (*this)[0]=a; (*this)[1]=b;
6d   (*this)[2]=c;
4e  }
10  xyz&operator=(const valarray<ld>&a) {
7b   assert(a.size()==3);
2a   *(valarray<ld>*)this = a;
06   return *this;
4d  }
75 };
||
2b ld dot(xyz a, xyz b){return(a*b).sum();}
39 xyz cross(xyz a, xyz b) {
bc  return xyz(a[1]*b[2]-b[1]*a[2],
7e   a[2]*b[0]-b[2]*a[0],
c9   a[0]*b[1]-b[0]*a[1]);
b3 }
71 ld hypot(xyz a){return sqrtl(dot(a,a));}
```

```
||
ac struct frame {
    // M[0] - fwd, M[1] - up, M[2] - right.
a0  xyz M[3];
    // take basis, make orthonormal basis.
6f  void fixit() {
e9   M[0] /= hypot(M[0]);
be   M[1] -= M[0]*dot(M[0],M[1]);
9d   M[1] /= hypot(M[1]);
a5   M[2] -= M[0]*dot(M[0],M[2]);
2b   M[2] -= M[1]*dot(M[1],M[2]);
a7   M[2] /= hypot(M[2]);
48  }
    // init with basis.
a2  frame(xyz fwd, xyz up, xyz ri) {
3d   M[0]=fwd; M[1]=up; M[2]=ri;
91   fixit();
e9  }
    // x * identity
a4  frame(ld x) {
1f   FOR(i,3)FOR(j,3) M[i][j] = x*(i==j);
8d  }
    // transform point
ab  xyz operator*(xyz a) const {
3f   return xyz(dot(M[0],a),
2b    dot(M[1],a), dot(M[2],a));
9c  }
    // compose maps
64  frame operator*(frame b) const {
7d   frame f(0);
ba   FOR(i,3)FOR(j,3)FOR(k,3)
0a    f[i][j] += M[i][k]*b[k][j];
42   return f;
2f  }
    // matrix inverse (= transpose)
08  frame inv() const {
66   frame f(0);
2f   FOR(i,3) FOR(j,3) f[i][j] = M[j][i];
da   return f;
03  }
    // compose with inverse
64  frame operator/(frame b) const {
59   return (*this)*b.inv();
77  }
    // indexing
60  xyz operator[](int k) const {
64   return M[k]; }
75  xyz&operator[](int k) { return M[k]; }
50 };
|| roll clockwise d radians
a8 frame roll(ld d) {
93  return frame(xyz( 1,       0,      0),
c8   xyz( 0, cosl(d),sinl(d)),
c3   xyz( 0,-sinl(d),cosl(d)));
6b }
|| pitch backward d radians
6d frame pitch(ld d) {
85  return frame(xyz( cosl(d),sinl(d), 0),
47   xyz(-sinl(d),cosl(d), 0),
16   xyz(       0,      0, 1));
55 }
|| yaw right d radians
44 frame yaw(ld d) {
27  return frame(xyz( cosl(d), 0,sinl(d)),
3f   xyz(       0, 1,      0),
81   xyz(-sinl(d), 0,cosl(d)));
e9 }
|| ================= turtle.cpp ==================
||  Partial solution to Finals 99 Problem F - Robot
||
||  Needs line segment intersection in 3-space to
||                                      be complete
||
||
79 #include <stdio.h>
7b #include <math.h>
e6 #include <string.h>
d6 #define M_PI acos(-1.0)
fb #include "turtle.h"
||
a8 triple joint[100];
89 double len[100],angle[100];
||
63 int D,N,i,j,k,nj;
||
d2 int main() {
a7  while (scanf("%d",&N) && N != -1) {
4f   printf("Case %d: ",++D);
81   frame f(triple(0,0,1),triple(0,1,0),triple(1,0
||                                            ,0));
     //print("initframe",f);
95   for (i=0;i<N;i++) scanf("%lf",&len[i]);
e3   for (i=0;i<N;i++) scanf("%lf",&angle[i]);
7e   joint[0] = triple(0,0,0);
e9   for (i=0;i<N;i++) {
```

```
      //printf("joint %lg %lg %lg\n",joint[i].x,
||                          joint[i].y,joint[i].z);
be    if (i%2 == 0) f = yaw(angle[i]) * f;
32    else f = pitch(-angle[i]) * f;
      //if (i%2 == 0) printf("yaw %lg\n",angle[i]);
      //else printf("pitch %lg\n",-angle[i]);
      //print("frame",f);
46    joint[i+1] = joint[i] + len[i] * ahead(f);
5a    if (joint[i+1].z < 0) {
ce     printf("servo %d attempts to move arm below
||                                   floor\n",i+1);
22     goto nextcase;
9c    }
e1   }
d8   printf("robot’s hand is at (%0.3f,%0.3f,%0.3f)
||                                  \n",joint[N].x,
ab    joint[N].y,joint[N].z);
df   nextcase:;
33  }
39  return 0;
64 }
|| ================== turtle.h ===================
||  3-D Turtle Graphics
|| Datatypes:
||  triple - an (x,y,z) triple representing a point
||                                      or a vector
||  frame - an orthonormal frame of reference
||  T - the element type (usually double)
|| Procedures:
||  T dist(triple a, triple b)   -  distance
||                               between 2 points
||   - dist(0,x) is the magnitude of x
||  T dot(triple a, triple b)    - dot product of 2
||                                             vecs
||
||  +, -, *  - element-by-element operations on
||                                          triples
||  triple normalize(x)  - unit length, same
||                                        direction
||  T cos(triple a, triple b) - cos of angle
||                                between 2 vectors
||  T sin(triple a, triple b) - sin of angle
||                                between 2 vectors
||  triple cross(triple a, triple b) - cross
||                                          product
||  dsin(), dcos() - helper routines use degrees
||   - exact for multiples of 90
||  frame(triple forward, triple up, triple right)
||   - creates orthonormal frame of reference
||   - x axis is straight ahead
||   - y axis is up
||   - z axis is to the right
||  triple ahead(frame f) - gives the x axis of f
||                                      in standard
||   frame of reference
||  *, /   - composition of frames of reference
||  frame yaw(double angle) - creates a frame that
||                               turns angle to the
||   right when multiplied to left of another frame
||  frame pitch(double angle) - creates a frame
||                           that inclines angle up
||  frame roll(double angle) - creates a frame that
||   rotates angle clockwise
||  frame yawto(frame f, triple from, triple to) -
||                             creates a frame that
||   turns right/left to pass directly over/under/
||                            through "to" position
||
||  frame pitchto(frame f, triple from, triple to)
||                                - creates a frame
||   that inclines up/down to pass object directly
||                                to the right/left
||   of or through "to" position
||  Note:  pitchto() and yawto() have singularities
||                                 if from == to or
||   if to is directly behind from.  from == to is
||                                      resolved by
||   returning no directional change.  to behind
||                                 from is resolved
||   by 180 degree yaw or pitch.
||  Note:  any pair of yaw/pitch/roll is sufficient
||                             for navigation.  You
||   can create rollto() if you really need it.
||
56 #define T double
||
c4 struct triple {
00  T x,y,z;
af  triple() { x=y=z=0; }
b4  triple(T a) {
22   x = y = z = a;
84  }
b6  triple(T a, T b, T c) {
13   x = a; y = b; z = c;
55  }
```

```
5e };
||
98 T dist(triple a, triple b) {
15  return sqrt((a.x-b.x)*(a.x-b.x)
23   + (a.y-b.y)*(a.y-b.y) + (a.z-b.z)*(a.z-b.z));}
||
f0 T dot(triple a, triple b) {
bc  return a.x*b.x + a.y*b.y + a.z * b.z;
28 }
||
d6 triple operator + (triple a, triple b) {
2d  return triple(a.x+b.x, a.y+b.y, a.z+b.z);
74 }
||
56 triple operator - (triple a, triple b) {
2e  return triple(a.x-b.x, a.y-b.y, a.z-b.z);
44 }
||
96 triple operator * (triple a, triple b) {
2d  return triple(a.x*b.x, a.y*b.y, a.z*b.z);
7c }
||
95 triple normalize (triple a) {
96  T norm = dist(a,0);
2b  return triple(a.x/norm, a.y/norm, a.z/norm);
64 }
||
30 T cos(triple a, triple b) {
61  return dot(normalize(a), normalize(b));
76 }
||
52 triple cross(triple a, triple b) {
9f  return triple(a.y*b.z-a.z*b.y, a.z*b.x-a.x*b.z,
||                               a.x*b.y-a.y*b.x);
c1 }
||
7d T sin(triple a, triple b) {
0b  return dist(cross(normalize(a),normalize(b)),0)
||                                                ;
dd }
||
da T dsin(T d) {
ef  T dd = fmod(d+360,360);
d8  if (dd == 0) return 0;
d9  if (dd == 180) return 0;
81  if (dd == 90) return 1;
18  if (dd == 270) return -1;
c6  return sin(dd*M_PI/180);
b3 }
||
40 T dcos(T d) {
06  T dd = fmod(d+360,360);
df  if (dd == 0) return 1;
6d  if (dd == 180) return -1;
b1  if (dd == 90) return 0;
09  if (dd == 270) return 0;
b4  return cos(dd*M_PI/180);
c0 }
||
ac struct frame {
cd  T m[3][3];
ab  frame(T x) {
7c   int i,j;
30   for (i=0;i<3;i++) for (j=0;j<3;j++) m[i][j] =
||                                       x*(i==j);
7c  }
af  frame(triple a, triple b, triple c) {
21   triple aa = normalize(a);
e2   triple bb = normalize(cross(cross(a,b),a));
86   triple cc = normalize(cross(a,b));
7a   if (dot(c,cc) < 0) cc = 0 - cc;
07   m[0][0] = aa.x;
39   m[0][1] = aa.y;
a6   m[0][2] = aa.z;
6f   m[1][0] = bb.x;
0b   m[1][1] = bb.y;
39   m[1][2] = bb.z;
22   m[2][0] = cc.x;
2f   m[2][1] = cc.y;
a9   m[2][2] = cc.z;
e8  }
||
4c  T * operator [] (int i) { return m[i]; }
32 };
||
b8 void print(char *s, frame f) {
3e  int i,j;
da  printf("fram %s\n",s);
88  for (i=0;i<3;i++) {
2a   for (j=0;j<3;j++) {
9f    printf("%8.2f ",f[i][j]);
29   }
b1   printf("\n");
a8  }
20 }
```

```
||
e0 triple ahead(frame x) {
9b  return triple(x.m[0][0],x.m[0][1],x.m[0][2]);
59 }
||
fe frame operator * (frame x, frame y) {
9a  frame r(0);
67  int i,j,k;
d6  for (i=0;i<3;i++) for (j=0;j<3;j++) for (k=0;k<
||                                             3;k++)
7c   r.m[i][k] += x.m[i][j] * y.m[j][k];
23  return r;
9c }
||
fb frame operator / (frame x, frame y) {
9b  frame r(0);
6d  int i,j,k;
dc  for (i=0;i<3;i++) for (j=0;j<3;j++) for (k=0;k<
||                                             3;k++)
fc   r.m[i][k] += x.m[i][j] * y.m[k][j];
22  return r;
9c }
||
4d frame roll(T d) {  // roll clockwise d degrees
b6  frame r(1);
55  r[1][1] = r[2][2] = dcos(d);
14  r[1][2] = dsin(d);
1f  r[2][1] = -dsin(d);
29  return r;
84 }
||
cb frame yaw(T d) {  // yaw right d degrees
d7  frame r(1);
2a  r[0][0] = r[2][2] = dcos(d);
6b  r[0][2] = dsin(d);
14  r[2][0] = -dsin(d);
d4  return r;
db }
||
e3 frame pitch(T d) {  // pitch up d degrees
9d  frame r(1);
2a  r[0][0] = r[1][1] = dcos(d);
6b  r[0][1] = dsin(d);
44  r[1][0] = -dsin(d);
de  return r;
5b }
||
fc triple intoframe (frame f, triple a) {
7b  return triple(
af   f[0][0]*a.x + f[0][1]*a.y + f[0][2]*a.z,
4a   f[1][0]*a.x + f[1][1]*a.y + f[1][2]*a.z,
91   f[2][0]*a.x + f[2][1]*a.y + f[2][2]*a.z);
11 }
||
6c frame pitchto(frame f, triple from, triple to) {
82  triple inmyframe = intoframe(f,to-from);
08  inmyframe.z = 0;
f4  if (dist(inmyframe,0) < 1e-10) return 1;
7e  return pitch(180/M_PI*atan2(inmyframe.y,
||                                  inmyframe.x));
b6 }
||
26 frame yawto(frame f, triple from, triple to) {
82  triple inmyframe = intoframe(f,to-from);
9a  inmyframe.y = 0;
d0  if (dist(inmyframe,0) < 1e-10) return 1;
20  return yaw(180/M_PI*atan2(inmyframe.z,inmyframe
||                                             .x));
5c }
|| ==================  val2.cpp ===================
|| Quick and dirty bignum /and/ polynomial code
|| You should only need a couple of these methods
|| Ask Ralph how to use it.
|| Current div will not work for polys, if I recall
||                                         correctly
||
94 #include <valarray>
67 #include <iostream>
97 using namespace std;
||
1c #define BIGINT
|| #define POLY
||
22 #ifdef BIGINT
c4 typedef int T;
25 int BASE=10;
ab #endif
||
39 #ifdef POLY
aa typedef double T;
51 int BASE=0;
78 #endif
||
14 typedef valarray<T> val;
fb int SZ=10;
```

```
||
10 val newVal(T x=0) {
26  val r(SZ);
8d  r[0]=x;
ff  return r;
12 }
||
a1 bool operator<(const val& a, const val& b) {
68  for(int i=SZ-1; i>0; i--)
d5   if(a[i]!=b[i])
5d    return a[i]<b[i];
9e  return a[0]<b[0];
41 }
||
69 ostream& operator<<(ostream& s, const val& v) {
24  for(int i=SZ-1; i>=0; i--) s << v[i] << " ";
31  return s;
1c }
||
15 val operator*(const val& a, const val& b) {
97  val c = newVal();
e3  for(int i=0; i<SZ; i++)
aa   for(int j=0; i+j<SZ; j++)
5c    c[i+j] += a[i]*b[j];
ff  return c;
da }
||
f6 val& wrap(val v) {
4e #ifdef BIGINT
45  for(int i=0; i<SZ-1; i++) {
e6   v[i+1]+=v[i]/BASE;
66   v[i]%=BASE;
c7   if(v[i]<0) {
d2    v[i+1]--;
02    v[i]+=BASE;
15   }
c5  }
9a  for(int i=SZ-1; i>0 && v[i-1]==BASE-1; i--) {
4d   v[i]=0;
df   v[i-1]=-1;
0b  }
3a  return v;
d5 #endif
9b }
||
c9 val div(val& a, const val& b) {
b0  int i,j;
a6  val ret=newVal();
5a  for(j=SZ-1; j>0; j--)
5b   if(b[j]) break;
a6  for(i=SZ-1; i>0; i--)
2c   if(a[i]) break;
8b  for(; i>=j; i--) {
c1   ret[i-j] = a[i]/b[j];
23   a[slice(i-j,j+1,1)] -= a[i]/b[j]*b[slice(0,j+1
||                                              ,1)];
     // You may remove the following when using
||                                       polynomials
ec   if(i) {
79    a[i-1] += BASE*a[i];
4d    a[i]=0;
f0    a=wrap(a);
b2   }
     //cout << a << " + " << b << " * " << ret <<
||                                             endl;
7d  }
fe  val zero=newVal();
d4  while(wrap(a)<zero) {
3e   ret[0]--;
41   a+=b;
ff  }
bb  while(!(wrap(a)<wrap(b))) {
c6   ret[0]++;
2c   a-=b;
1c  }
17  return ret;
8d }
||
e5 val operator/(val a, val& b) {
95  return div(a,b);
21 }
31 val& operator%(val a, val& b) {
b9  div(a,b);
c1  return a;
73 }
||
||
||
c4 int main(void) {
3e  val x=wrap(newVal(1000000));
fe  val y=wrap(newVal(19));
    //cout << wrap(-x+y);
64  cout << wrap(x/y) << " " << wrap(x%y) << endl;
37  return 0;
dc }
```

```
|| ================  watchman.cpp =================
|| Given a concave simple polygon (no repeating
||                                            edges)
|| Calculate a triangulation, and solve the
||                                 watchman problem
|| Almost worked in practice.  I made a fix, but do
||                                          not have
|| the test cases to check with.  beware.
||
79 #include "geometry.h"
||
e3 int N;
29 poly p;
46 int col[200];
8f int good[200][200];
||
23 void doit(int i, int j, int a, int b) {
09  if((i+1)==j) {
b7   col[i]=a;
43   col[j]=b;
b5   return;
50  }
09  for(int k=i+1; k!=j; k++) if(good[i][k]&&good[k
||                                          ][j]) {
     // Insert edge ij
     // recurse on halfpolys i..k and k..j
1c   doit(i,k,a,3-a-b);
43   doit(k,j,3-a-b,b);
35   return;
14  }
25 }
||
b6 void triangulate() {
99  fu(i,N) fu(j,N) if(i!=j) good[i][j]=1;
b0  fu(i,N) fu(j,N) if(pointInsidePolygon((p[i]+p[j
||                     ])*0.5, p)!=1) good[i][j]=0;
2f  fu(i,N) good[(i+1)%N][i]=good[i][(i+1)%N]=1;
94  fu(i,N) fu(j,i) fu(k,N) if(k!=i && k!=j) { int
||                       l=(k+1)%N; if(l!=i && l!=j)
d2   if(doesIntersect(p[i],p[j],p[k],p[l])) good[i]
||                               [j]=good[j][i]=0;
fb  }
82  doit(0,N-1,0,1);
40  int cnt[3]={0,0,0};
43  fu(i,N) cnt[col[i]]++;
f3  fu(i,3) if(cnt[i]<=N/3) {
22   cout << cnt[i] << endl;
48   fu(j,N) if(col[j]==i) printf("%.3lf %.3lf\n",p
||                         [j].real(),p[j].imag());
d4   return;
6b  }
7e }
||
d2 int main() {
15  cin >> N;
97  fu(i,N) {
8d   double x,y;
8d   cin >> x >> y;
4c   p.push_back(point(x,y));
cf  }
bc  srand(time(0));
c7  if(rand()%2) reverse(p.begin(),p.end());
36  rotate(&p[0],&p[rand()%N],&p[N]);
5b  triangulate();
af }
|| ==================  zalgo.cc ===================
|| ========= Z-Algo ===========
||
c5 vi ZAlgo(string vec) {
1c  int N = vec.sz;
34  vi Z(N);
e9  int L = 0, R = 0;
78  FOR(i, 1, N) {
df   if (i <= R)
f4    Z[i] = min(Z[i-L], R-i+1);
c9   while ((i + Z[i] < N) && (vec[i+Z[i]] == vec[Z
||                                           [i]]))
f7    Z[i]++;
8f   if (R < i + Z[i] - 1) {
b8    L = i;
a2    R = i + Z[i] - 1;
89   };
79  }
8a  return Z;
b1 }
```