Team Notebook

Temporarily Rebillion - Amirkabir University of Technology

March 25, 2019

Contents

1 BlockCutTree

```
#include<bits/stdc++.h>

using namespace std;

typedef pair<int,int> II;
typedef vector< II >   VII;
typedef vector<int>   VI;
typedef vector< VI >  VVI;
typedef long long int  LL;

#define PB push_back
#define MP make_pair
#define F first
#define S second
#define SZ(a) (int)(a.size())
#define ALL(a) a.begin(),a.end()
#define SET(a,b) memset(a,b,sizeof(a))

#define si(n) scanf("%d",&n)
#define dout(n) printf("%d\n",n)
#define sll(n) scanf("%lld",&n)
#define lldout(n) printf("%lld\n",n)
#define fast_io ios_base::sync_with_stdio(false);cin.tie(
     NULL)

#define TRACE

#ifdef TRACE
#define trace(...) __f(#__VA_ARGS__, __VA_ARGS__)
template <typename Arg1>
void __f(const char* name, Arg1&& arg1){
    cerr << name << " : " << arg1 << std::endl;
}
template <typename Arg1, typename... Args>
void __f(const char* names, Arg1&& arg1, Args&&... args){
    const char* comma = strchr(names + 1, ',');cerr.write(
         names, comma - names) << " : " << arg1<<" | ";__f(
         comma+1, args...);
}
#else
#define trace(...)
#endif

//FILE *fin = freopen("in","r",stdin);
//FILE *fout = freopen("out","w",stdout);
const int N = int(2e5)+1;
const int M = int(2e5)+1;
const int LOGN = 20;
VI g[N],tree[N],st;//graph in edge-list form. N should be 2*
     N
int U[M],V[M],low[N],ord[N],sz[N],depth[N],col[N],C,T,compNo
     [N],extra[N],level[N],DP[LOGN][N];
bool isArtic[N];
int arr[N],dep[N],vis[N];
int adj(int u,int e){
    return u^V[e]^U[e];
}
//everything from [1,n+C] whose extra[i]=0 is part of Block-
     Tree
//1-Based Graph Input.Everything from [1,C] is type B and [C
     ,n+C] is type C.
void dfs(int i){
    low[i]=ord[i]=T++;
    for(int j=0;j<SZ(g[i]);j++){
 int ei=g[i][j],to = adj(i,ei);
 if(ord[to]==-1){
     depth[to]=depth[i]+1;
     st.PB(ei);dfs(to);
     low[i] = min(low[i],low[to]);
     if(ord[i]==0||low[to]>=ord[i]){
  if(ord[i]!=0||j>=1)
      isArtic[i] = true;
  ++C;
  while(!st.empty()){
      int fi=st.back();st.pop_back();
      col[fi]=C;
      if(fi==ei)break;
  }
     }
 }else if(depth[to]<depth[i]-1){
     low[i] = min(low[i],ord[to]);
     st.PB(ei);
 }
    }
}
void run(int n){
    SET(low,-1);SET(depth,-1);
    SET(ord,-1);SET(col,-1);
    SET(isArtic,0);st.clear();C=0;
    for(int i=1;i<=n;++i)
 if(ord[i]==-1){
     T = 0;dfs(i);
 }
}
void buildTree(int n){
    run(n);SET(compNo,-1);
    VI tmpv;SET(extra,-1);
    tmpv.clear();SET(sz,0);
    for(int i=1;i<=n;i++){
 tmpv.clear();
 for(auto e:g[i])
     tmpv.PB(col[e]);
 sort(ALL(tmpv));
 tmpv.erase(unique(ALL(tmpv)), tmpv.end());
 //handle isolated vertics
 if(tmpv.empty()){
     compNo[i]=C+i;extra[C+i]=0;
     sz[C+i]=1;continue;
 }if(SZ(tmpv)==1){//completely in 1 comp.
     compNo[i]=tmpv[0];
     extra[tmpv[0]]=0;
     sz[tmpv[0]]++;
 }else{ //it's an articulation vertex.
     compNo[i]=C+i;
     extra[C+i]=0;sz[C+i]++;
     for(auto j:tmpv){
  extra[j]=0;sz[j]++;
  tree[C+i].push_back(j);
  tree[j].push_back(C+i);
     }
 }
    }
}
int currComp;
void dfs2(int u,int p){
    level[u]=level[p]+1;DP[0][u]=p;
    arr[u]=++T;vis[u]=currComp;
    for(auto w:tree[u])
 if(w!=p)
     dfs2(w,u);
    dep[u]=T++;
}
int lca(int a,int b){
    if(level[a]>level[b])swap(a,b);
    int d = level[b]-level[a];
    for(int i=0;i<LOGN;i++)
 if((1<<i)&d)
     b = DP[i][b];
    if(a==b)return a;
    for(int i=LOGN-1;i>=0;i--)
 if(DP[i][a]!=DP[i][b])
     a=DP[i][a],b=DP[i][b];
    return DP[0][a];
}
bool anc(int p,int u){
    return (arr[u]>=arr[p] && dep[u]<=dep[p]);
}
int main()
```

```
{
    int n,m,q;
    si(n);si(m);si(q);
    for(int i=0;i<m;i++){
 scanf("%d %d",U+i,V+i);
 g[U[i]].PB(i);
 g[V[i]].PB(i);
    }
    buildTree(n);T=0;
    for(int i=1;i<=C+n;i++)
 if(!vis[i] && !extra[i])
     currComp++,dfs2(i,i);
    for(int i=1;i<LOGN;i++)
 for(int j=1;j<=C+n;j++)
     if(!extra[j])
  DP[i][j]=DP[i-1][DP[i-1][j]];
    while(q--){
 int u,v,w;
 si(u);si(v);si(w);
 if(u==v){
     puts(u==w?"Party":"Break-Up");
     continue;
 }
 u=compNo[u];v=compNo[v];w=compNo[w];
 if(!(vis[u]==vis[w] && vis[w]==vis[v])){
     puts("Break-Up");
     continue;
 }
 int LCA = lca(u,v);
 if(level[u]>level[v])swap(u,v);
 if(sz[w]==1 && w!=LCA && w!=DP[0][LCA] && sz[DP[0][w]]>2) w
      = DP[0][w];
 if(sz[u]==1 && u!=LCA && sz[DP[0][w]]>2) u = DP[0][u];
 if(sz[v]==1 && v!=LCA && sz[DP[0][v]]>2) v = DP[0][v];
 bool ok=false;
 ok|=anc(w,u);
 ok|=anc(w,v);
 ok&=anc(LCA,w);
 ok|=(sz[LCA]>2 && w==DP[0][LCA]);
 puts(ok?"Party":"Break-Up");
    }
    return 0;
}
```

2 Centroid

```
#include <bits/stdc++.h>
#define X first
#define Y second
#define pb push_back
using namespace std;
typedef pair<int, int> pii;
typedef pair<pii, int> ppi;
const int maxn = 2e5 + 17, lg = 18;

int n = 1, q, par[maxn][lg], cpar[maxn], h[maxn], sz[maxn];
set<ppi> s[maxn];
vector<int> g[maxn], ch[maxn];
struct Q{
 int t, v, d;
}  qu[maxn];
void prep(int v = 0){
 sz[v] = 1;
 for(auto u : g[v]){
  prep(u);
  sz[v] += sz[u];
 }
}
int get_cent(int root = 0){
 int v = root, size = sz[root];
 bool done = 0;
 while(done ^= 1)
  for(auto &u : g[v])
   if(sz[u] > (size >> 1)){
    v = u, done = 0;
    break;
   }
 int mysz = sz[v];
 for(int u = v; ; u = par[u][0]){
  sz[u] -= mysz;
  if(u == root) break;
 }
 for(auto &u : g[v])
  if(sz[u]){
   int x = get_cent(u);
   //cerr << v << ' ' << x << '\n';
   cpar[x] = v;
   ch[v].pb(x);
  }
 if(v != root){
  int x = get_cent(root);
  //cerr << v << ' ' << x << '\n';
  cpar[x] = v;
  ch[v].pb(x);
 }
 return v;
}
int dis(int v, int u){
    if(h[u] < h[v]) swap(v, u);
    int ans = h[v] + h[u];
    for(int i = 0; i < lg; i++)
 if((h[u] - h[v]) >> i & 1)
     u = par[u][i];
    for(int i = lg - 1; i >= 0; i--)
 if(par[v][i] != par[u][i])
     v = par[v][i], u = par[u][i];
    return v == u ? ans - 2 * h[v] : ans - 2 * (h[v] - 1);
}
void add(int v){
 for(int u = v; u != -1; u = cpar[u]){
  if(v == 6)
   ;//cerr << u << '\n';
  int d = dis(u, v);
  auto it = s[u].lower_bound({{d + 1, -1}, -1});
  if(it != s[u].begin() && prev(it) -> X.Y >= h[v])
   continue;
  it = s[u].insert({{d, h[v]}, v}).X;
  it++;
  while(it != s[u].end() && it -> X.Y <= h[v])
   s[u].erase(prev(++it));
 }
}
int get(int v, int d){
 int ans = -1, cer = -1;
 for(int u = v; u != -1; u = cpar[u]){
  int di = dis(u, v);
  //cerr << u << '\n';
  auto it = s[u].lower_bound({{d - di + 1, -1}, -1});
  if(it != s[u].begin()){
   it--;
   if(it -> X.Y > ans)
    ans = it -> X.Y, cer = it -> Y;
  }
 }
 return cer;
}
```

3 ConvexHull

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

```
//
//   INPUT:   a vector of input points, unordered.
//   OUTPUT: a vector of points in the convex hull,
    counterclockwise, starting
//           with bottommost/leftmost point

#include <cstdio>
#include <cassert>
#include <vector>
#include <algorithm>
#include <cmath>
// BEGIN CUT
#include <map>
// END CUT

using namespace std;

#define REMOVE_REDUNDANT

typedef double T;
const T EPS = 1e-7;
struct PT {
  T x, y;
  PT() {}
  PT(T x, T y) : x(x), y(y) {}
  bool operator<(const PT &rhs) const { return make_pair(y,x
      ) < make_pair(rhs.y,rhs.x); }
  bool operator==(const PT &rhs) const { return make_pair(y,
      x) == make_pair(rhs.y,rhs.x); }
};

T cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
T area2(PT a, PT b, PT c) { return cross(a,b) + cross(b,c) +
      cross(c,a); }

#ifdef REMOVE_REDUNDANT
bool between(const PT &a, const PT &b, const PT &c) {
  return (fabs(area2(a,b,c)) < EPS && (a.x-b.x)*(c.x-b.x) <=
        0 && (a.y-b.y)*(c.y-b.y) <= 0);
}
#endif

void ConvexHull(vector<PT> &pts) {
  sort(pts.begin(), pts.end());
  pts.erase(unique(pts.begin(), pts.end()), pts.end());
  vector<PT> up, dn;
  for (int i = 0; i < pts.size(); i++) {
    while (up.size() > 1 && area2(up[up.size()-2], up.back(),
          pts[i]) >= 0) up.pop_back();
    while (dn.size() > 1 && area2(dn[dn.size()-2], dn.back(),
          pts[i]) <= 0) dn.pop_back();
    up.push_back(pts[i]);
    dn.push_back(pts[i]);
  }
  pts = dn;
  for (int i = (int) up.size() - 2; i >= 1; i--) pts.
      push_back(up[i]);

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

// BEGIN CUT
// The following code solves SPOJ problem #26: Build the
    Fence (BSHEEP)

int main() {
  int t;
  scanf("%d", &t);
  for (int caseno = 0; caseno < t; caseno++) {
    int n;
    scanf("%d", &n);
    vector<PT> v(n);
    for (int i = 0; i < n; i++) scanf("%lf%lf", &v[i].x, &v[i
        ].y);
    vector<PT> h(v);
    map<PT,int> index;
    for (int i = n-1; i >= 0; i--) index[v[i]] = i+1;
    ConvexHull(h);

    double len = 0;
    for (int i = 0; i < h.size(); i++) {
      double dx = h[i].x - h[(i+1)%h.size()].x;
      double dy = h[i].y - h[(i+1)%h.size()].y;
      len += sqrt(dx*dx+dy*dy);
    }

    if (caseno > 0) printf("\n");
    printf("%.2f\n", len);
    for (int i = 0; i < h.size(); i++) {
      if (i > 0) printf(" ");
      printf("%d", index[h[i]]);
    }
    printf("\n");
  }
}

// END CUT
```

4 ConvexHullTrick

```
typedef long long int64;
typedef long double float128;

const int64 is_query = -(1LL<<62), inf = 1e18;

struct Line {
 int64 m, b;
 mutable function<const Line*()> succ;
 bool operator<(const Line& rhs) const {
  if (rhs.b != is_query) return m < rhs.m;
  const Line* s = succ();
  if (!s) return 0;
  int64 x = rhs.m;
  return b - s->b < (s->m - m) * x;
 }
};

struct HullDynamic : public multiset<Line> { // will
    maintain upper hull for maximum
 bool bad(iterator y) {
  auto z = next(y);
  if (y == begin()) {
   if (z == end()) return 0;
   return y->m == z->m && y->b <= z->b;
  }
  auto x = prev(y);
  if (z == end()) return y->m == x->m && y->b <= x->b;
  return (float128)(x->b - y->b)*(z->m - y->m) >= (float128)
      (y->b - z->b)*(y->m - x->m);
 }
 void insert_line(int64 m, int64 b) {
  auto y = insert({ m, b });
```

```
 y->succ = [=] { return next(y) == end() ? 0 : &*next(y);
     };
 if (bad(y)) { erase(y); return; }
 while (next(y) != end() && bad(next(y))) erase(next(y));
 while (y != begin() && bad(prev(y))) erase(prev(y));
}

int64 eval(int64 x) {
 auto l = *lower_bound((Line) { x, is_query });
 return l.m * x + l.b;
}
};
```

5 Cut

```
stack<int> stak;
inline void add_edge(int v, int u){
    g[v].push_back(u), g[u].push_back(v);
}
int get_cut(int v = 0, int p = -1){
    if(mark[v]) return h[v];
    hi[v] = h[v] = ~p ? h[p] + 1 : 0, mark[v] = 1;
    stak.push(v);
    for(auto u : adj[v])
 smin(hi[v], get_cut(u, v));
    if(hi[v] + 1 == h[v]){
 while(stak.top() != v)
     add_edge(stak.top(), v + n), stak.pop();
 add_edge(v, v + n), stak.pop();
 add_edge(p, v + n);
    }
    return hi[v];
}
```

6 Dates

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

#include <iostream>
#include <string>

using namespace std;

string dayOfWeek[] = {"Mon", "Tue", "Wed", "Thu", "Fri", "
     Sat", "Sun"};

// converts Gregorian date to integer (Julian day number)
int dateToInt (int m, int d, int y){
  return
    1461 * (y + 4800 + (m - 14) / 12) / 4 +
    367 * (m - 2 - (m - 14) / 12 * 12) / 12 -
    3 * ((y + 4900 + (m - 14) / 12) / 100) / 4 +
    d - 32075;
}

// converts integer (Julian day number) to Gregorian date:
     month/day/year
void intToDate (int jd, int &m, int &d, int &y){
  int x, n, i, j;

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

// converts integer (Julian day number) to day of week
string intToDay (int jd){
  return dayOfWeek[jd % 7];
}

int main (int argc, char **argv){
  int jd = dateToInt (3, 24, 2004);
  int m, d, y;
  intToDate (jd, m, d, y);
  string day = intToDay (jd);

  // expected output:
  //    2453089
  //    3/24/2004
  //    Wed
  cout << jd << endl
    << m << "/" << d << "/" << y << endl
    << day << endl;
}
```

7 Dates

```
// Example of using Java's built-in date calculation
     routines

import java.text.SimpleDateFormat;
import java.util.*;

public class Dates {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        SimpleDateFormat sdf = new SimpleDateFormat("M/d/yyyy
             ");
        while (true) {
            int n = s.nextInt();
            if (n == 0) break;
            GregorianCalendar c = new GregorianCalendar(n,
                 Calendar.JANUARY, 1);
            while (c.get(Calendar.DAY_OF_WEEK) != Calendar.
                 SATURDAY)
  c.add(Calendar.DAY_OF_YEAR, 1);
            for (int i = 0; i < 12; i++) {
                System.out.println(sdf.format(c.getTime()));
                while (c.get(Calendar.MONTH) == i) c.add(
                     Calendar.DAY_OF_YEAR, 7);
            }
        }
    }
}
```

8 DecFormat

```
// examples for printing floating point numbers

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

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

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

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

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

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

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

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

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

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

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

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

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

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

9 Delaunay

```
// Slow but simple Delaunay triangulation. Does not handle
// degenerate cases (from O'Rourke, Computational Geometry
     in C)
//
// Running time: O(n^4)
//
// INPUT:   x[] = x-coordinates
//          y[] = y-coordinates
//
// OUTPUT:  triples = a vector containing m triples of
     indices
//                    corresponding to triangle vertices

#include<vector>
using namespace std;

typedef double T;

struct triple {
   int i, j, k;
   triple() {}
   triple(int i, int j, int k) : i(i), j(j), k(k) {}
};

vector<triple> delaunayTriangulation(vector<T>& x, vector<T
     >& y) {
 int n = x.size();
 vector<T> z(n);
 vector<triple> ret;

 for (int i = 0; i < n; i++)
     z[i] = x[i] * x[i] + y[i] * y[i];

 for (int i = 0; i < n-2; i++) {
     for (int j = i+1; j < n; j++) {
  for (int k = i+1; k < n; k++) {
      if (j == k) continue;
      double xn = (y[j]-y[i])*(z[k]-z[i]) - (y[k]-y[i])*(z[j
           ]-z[i]);
      double yn = (x[k]-x[i])*(z[j]-z[i]) - (x[j]-x[i])*(z[k
           ]-z[i]);
      double zn = (x[j]-x[i])*(y[k]-y[i]) - (x[k]-x[i])*(y[j
           ]-y[i]);
      bool flag = zn < 0;
      for (int m = 0; flag && m < n; m++)
   flag = flag && ((x[m]-x[i])*xn +
     (y[m]-y[i])*yn +
     (z[m]-z[i])*zn <= 0);
      if (flag) ret.push_back(triple(i, j, k));
  }
     }
```

```
 }
 return ret;
}

int main()
{
    T xs[]={0, 0, 1, 0.9};
    T ys[]={0, 1, 0, 0.9};
    vector<T> x(&xs[0], &xs[4]), y(&ys[0], &ys[4]);
    vector<triple> tri = delaunayTriangulation(x, y);

    //expected: 0 1 3
    //          0 3 2

    int i;
    for(i = 0; i < tri.size(); i++)
        printf("%d %d %d\n", tri[i].i, tri[i].j, tri[i].k);
    return 0;
}
```

10 Delaunay

```
// Slow but simple Delaunay triangulation. (from O'Rourke,
// Computational Geometry in C)
//
// Running time: O(n^4)
//
// INPUT:   x[] = x-coordinates
//          y[] = y-coordinates
//
// OUTPUT:  ret[][] = an mx3 matrix containing m triples of
     indices
//                  corresponding to triangle vertices

import java.util.*;

public class Delaunay {
    int[][] triangulate(double[] x, double[] y) {
 int n = x.length;
 double z[] = new double[n];
 ArrayList<int[]> ret = new ArrayList<int[]>();

 for (int i = 0; i < n; i++)
     z[i] = x[i] * x[i] + y[i] * y[i];

 for (int i = 0; i < n-2; i++) {
     for (int j = i+1; j < n; j++) {
  for (int k = i+1; k < n; k++) {
      if (j == k) continue;
      double xn = (y[j]-y[i])*(z[k]-z[i]) - (y[k]-y[i])*(z[j
          ]-z[i]);
      double yn = (x[k]-x[i])*(z[j]-z[i]) - (x[j]-x[i])*(z[k
          ]-z[i]);
      double zn = (x[j]-x[i])*(y[k]-y[i]) - (x[k]-x[i])*(y[j
          ]-y[i]);
      boolean flag = zn < 0;
      for (int m = 0; flag && m < n; m++)
   flag = flag && ((x[m]-x[i])*xn +
     (y[m]-y[i])*yn +
     (z[m]-z[i])*zn <= 0);
      if (flag) ret.add(new int[]{i, j, k});
  }
     }
 }
 return ret.toArray(new int[0][0]);
    }
}
```

11 Euclid

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

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

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

// return a % b (positive value)
int mod(int a, int b) {
 return ((a%b) + b) % b;
}

// computes gcd(a,b)
int gcd(int a, int b) {
 while (b) { int t = a%b; a = b; b = t; }
 return a;
}

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

// (a^b) mod m via successive squaring
int powermod(int a, int b, int m)
{
 int ret = 1;
 while (b)
 {
  if (b & 1) ret = mod(ret*a, m);
  a = mod(a*a, m);
  b >>= 1;
 }
 return ret;
}

// returns g = gcd(a, b); finds x, y such that d = ax + by
int extended_euclid(int a, int b, int &x, int &y) {
 int xx = y = 0;
 int yy = x = 1;
 while (b) {
  int q = a / b;
  int t = b; b = a%b; a = t;
  t = xx; xx = x - q*xx; x = t;
  t = yy; yy = y - q*yy; y = t;
 }
 return a;
}

// finds all solutions to ax = b (mod n)
VI modular_linear_equation_solver(int a, int b, int n) {
 int x, y;
 VI ret;
 int g = extended_euclid(a, n, x, y);
 if (!(b%g)) {
  x = mod(x*(b / g), n);
  for (int i = 0; i < g; i++)
   ret.push_back(mod(x + i*(n / g), n));
 }
 return ret;
}

// computes b such that ab = 1 (mod n), returns -1 on
    failure
int mod_inverse(int a, int n) {
 int x, y;
 int g = extended_euclid(a, n, x, y);
 if (g > 1) return -1;
 return mod(x, n);
```

```
}

// Chinese remainder theorem (special case): find z such
    that
// z % m1 = r1, z % m2 = r2. Here, z is unique modulo M =
    lcm(m1, m2).
// Return (z, M). On failure, M = -1.
PII chinese_remainder_theorem(int m1, int r1, int m2, int r2
    ) {
 int s, t;
 int g = extended_euclid(m1, m2, s, t);
 if (r1%g != r2%g) return make_pair(0, -1);
 return make_pair(mod(s*r2*m1 + t*r1*m2, m1*m2) / g, m1*m2 /
      g);
}

// Chinese remainder theorem: find z such that
// z % m[i] = r[i] for all i. Note that the solution is
// unique modulo M = lcm_i (m[i]). Return (z, M). On
// failure, M = -1. Note that we do not require the a[i]'s
// to be relatively prime.
PII chinese_remainder_theorem(const VI &m, const VI &r) {
 PII ret = make_pair(r[0], m[0]);
 for (int i = 1; i < m.size(); i++) {
  ret = chinese_remainder_theorem(ret.second, ret.first, m[i
      ], r[i]);
  if (ret.second == -1) break;
 }
 return ret;
}

// computes x and y such that ax + by = c
// returns whether the solution exists
bool linear_diophantine(int a, int b, int c, int &x, int &y)
     {
 if (!a && !b)
 {
  if (c) return false;
  x = 0; y = 0;
  return true;
 }
 if (!a)
 {
  if (c % b) return false;
  x = 0; y = c / b;
  return true;
 }
 if (!b)
 {
  if (c % a) return false;
  x = c / a; y = 0;
  return true;
 }
 int g = gcd(a, b);
 if (c % g) return false;
 x = c / g * mod_inverse(a / g, b / g);
 y = (c - a*x) / b;
 return true;
}

int main() {
 // expected: 2
 cout << gcd(14, 30) << endl;

 // expected: 2 -2 1
 int x, y;
 int g = extended_euclid(14, 30, x, y);
 cout << g << " " << x << " " << y << endl;

 // expected: 95 451
 VI sols = modular_linear_equation_solver(14, 30, 100);
 for (int i = 0; i < sols.size(); i++) cout << sols[i] << "
     ";
 cout << endl;

 // expected: 8
 cout << mod_inverse(8, 9) << endl;

 // expected: 23 105
 //           11 12
 PII ret = chinese_remainder_theorem(VI({ 3, 5, 7 }), VI({
     2, 3, 2 }));
 cout << ret.first << " " << ret.second << endl;
 ret = chinese_remainder_theorem(VI({ 4, 6 }), VI({ 3, 5 }))
     ;
 cout << ret.first << " " << ret.second << endl;

 // expected: 5 -15
 if (!linear_diophantine(7, 2, 5, x, y)) cout << "ERROR" <<
     endl;
 cout << x << " " << y << endl;
 return 0;
}
```

12 EulerianPath

```
struct Edge;
typedef list<Edge>::iterator iter;

struct Edge
{
 int next_vertex;
 iter reverse_edge;

 Edge(int next_vertex)
  :next_vertex(next_vertex)
  { }
};

const int max_vertices = ;
int num_vertices;
list<Edge> adj[max_vertices];  // adjacency list

vector<int> path;

void find_path(int v)
{
 while(adj[v].size() > 0)
 {
  int vn = adj[v].front().next_vertex;
  adj[vn].erase(adj[v].front().reverse_edge);
  adj[v].pop_front();
  find_path(vn);
 }
 path.push_back(v);
}

void add_edge(int a, int b)
{
 adj[a].push_front(Edge(b));
 iter ita = adj[a].begin();
 adj[b].push_front(Edge(a));
 iter itb = adj[b].begin();
 ita->reverse_edge = itb;
 itb->reverse_edge = ita;
}
```

13 FFT

```
#define REP(i, n) for(int i = 0; i < (n); i++)
typedef int llint;
namespace FFT {
 const int MAX = 1 << 17;

 typedef llint value;
 typedef complex<double> comp;
```

```
int N;
comp omega[MAX];
comp a1[MAX], a2[MAX];
comp z1[MAX], z2[MAX];

void fft(comp *a, comp *z, int m = N) {
 if (m == 1) {
  z[0] = a[0];
 } else {
  int s = N/m;
  m /= 2;

  fft(a, z, m);
  fft(a+s, z+m, m);

  REP(i, m) {
   comp c = omega[s*i] * z[m+i];
   z[m+i] = z[i] - c;
   z[i] += c;
  }
 }
}

void mult(value *a, value *b, value *c, int len) {
 N = 2*len;
 while (N & (N-1)) ++N;
 assert(N <= MAX);

 REP(i, N) a1[i] = 0;
 REP(i, N) a2[i] = 0;
 REP(i, len) a1[i] = a[i];
 REP(i, len) a2[i] = b[i];

 REP(i, N) omega[i] = polar(1.0, 2*M_PI/N*i);
 fft(a1, z1, N);
 fft(a2, z2, N);

 REP(i, N) omega[i] = comp(1, 0) / omega[i];
 REP(i, N) a1[i] = z1[i] * z2[i] / comp(N, 0);
 fft(a1, z1, N);

 REP(i, 2*len) c[i] = round(z1[i].real());
}

void mult_mod(int *a, int *b, int *c, int len, int mod) {
 static llint a0[MAX], a1[MAX];
 static llint b0[MAX], b1[MAX];
 static llint c0[MAX], c1[MAX], c2[MAX];
 REP(i, len) a0[i] = a[i] & 0xFFFF;
 REP(i, len) a1[i] = a[i] >> 16;

 REP(i, len) b0[i] = b[i] & 0xFFFF;
 REP(i, len) b1[i] = b[i] >> 16;

 FFT::mult(a0, b0, c0, len);
 FFT::mult(a1, b1, c2, len);

 REP(i, len) a0[i] += a1[i];
 REP(i, len) b0[i] += b1[i];
 FFT::mult(a0, b0, c1, len);
 REP(i, 2*len) c1[i] -= c0[i] + c2[i];

 REP(i, 2*len) c1[i] %= mod;
 REP(i, 2*len) c2[i] %= mod;
 REP(i, 2*len) c[i] = (c0[i] + ((long long) c1[i] << 16) +
     ((long long) c2[i] << 32)) % mod;
 }
}
#undef REP
```

14 GaussElim

```
int gauss (vector < vector<double> > a, vector<double> & ans
    ) {
 int n = (int) a.size();
 int m = (int) a[0].size() - 1;

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

  for (int i=0; i<n; ++i)
   if (i != row) {
    double c = a[i][col] / a[row][col];
    for (int j=col; j<=m; ++j)
     a[i][j] -= a[row][j] * c;
   }
  ++row;
 }

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

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

15 GaussJordan

```
// Gauss-Jordan elimination with full pivoting.
//
// Uses:
//   (1) solving systems of linear equations (AX=B)
//   (2) inverting matrices (AX=I)
//   (3) computing determinants of square matrices
//
// Running time: O(n^3)
//
// INPUT:   a[][] = an nxn matrix
//          b[][] = an nxm matrix
//
// OUTPUT:  X      = an nxm matrix (stored in b[][])
//          A^{-1} = an nxn matrix (stored in a[][])
//          returns determinant of a[][]

#include <iostream>
#include <vector>
#include <cmath>

using namespace std;

const double EPS = 1e-10;

typedef vector<int> VI;
typedef double T;
```

```
typedef vector<T> VT;
typedef vector<VT> VVT;

T GaussJordan(VVT &a, VVT &b) {
  const int n = a.size();
  const int m = b[0].size();
  VI irow(n), icol(n), ipiv(n);
  T det = 1;

  for (int i = 0; i < n; i++) {
    int pj = -1, pk = -1;
    for (int j = 0; j < n; j++) if (!ipiv[j])
      for (int k = 0; k < n; k++) if (!ipiv[k])
 if (pj == -1 || fabs(a[j][k]) > fabs(a[pj][pk])) { pj = j;
      pk = k; }
    if (fabs(a[pj][pk]) < EPS) { cerr << "Matrix is singular.
        " << endl; exit(0); }
    ipiv[pk]++;
    swap(a[pj], a[pk]);
    swap(b[pj], b[pk]);
    if (pj != pk) det *= -1;
    irow[i] = pj;
    icol[i] = pk;

    T c = 1.0 / a[pk][pk];
    det *= a[pk][pk];
    a[pk][pk] = 1.0;
    for (int p = 0; p < n; p++) a[pk][p] *= c;
    for (int p = 0; p < m; p++) b[pk][p] *= c;
    for (int p = 0; p < n; p++) if (p != pk) {
      c = a[p][pk];
      a[p][pk] = 0;
      for (int q = 0; q < n; q++) a[p][q] -= a[pk][q] * c;
      for (int q = 0; q < m; q++) b[p][q] -= b[pk][q] * c;
    }
  }

  for (int p = n-1; p >= 0; p--) if (irow[p] != icol[p]) {
    for (int k = 0; k < n; k++) swap(a[k][irow[p]], a[k][icol
        [p]]);
  }

  return det;
}

int main() {
  const int n = 4;
  const int m = 2;
  double A[n][n] = { {1,2,3,4},{1,0,1,0},{5,3,2,4},{6,1,4,6}
      };
  double B[n][m] = { {1,2},{4,3},{5,6},{8,7} };
  VVT a(n), b(n);
  for (int i = 0; i < n; i++) {
    a[i] = VT(A[i], A[i] + n);
    b[i] = VT(B[i], B[i] + m);
  }

  double det = GaussJordan(a, b);

  // expected: 60
  cout << "Determinant: " << det << endl;

  // expected: -0.233333 0.166667 0.133333 0.0666667
  //           0.166667 0.166667 0.333333 -0.333333
  //           0.233333 0.833333 -0.133333 -0.0666667
  //           0.05 -0.75 -0.1 0.2
  cout << "Inverse: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)
      cout << a[i][j] << ' ';
    cout << endl;
  }

  // expected: 1.63333 1.3
  //           -0.166667 0.5
  //           2.36667 1.7
  //           -1.85 -1.35
  cout << "Solution: " << endl;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++)
      cout << b[i][j] << ' ';
    cout << endl;
  }
}
```

16 Geom3D

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

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

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

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

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

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

17 Geometry

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

#include <iostream>
#include <vector>
#include <cmath>
#include <cassert>

using namespace std;

double INF = 1e100;
double EPS = 1e-12;

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

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

// rotate a point CCW or CW around the origin
PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
PT RotateCW90(PT p)  { return PT(p.y,-p.x); }
PT RotateCCW(PT p, double t) {
  return PT(p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t));
}

// project point c onto line through a and b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a + (b-a)*dot(c-a, b-a)/dot(b-a, b-a);
}

// project point c onto line segment through a and b
PT ProjectPointSegment(PT a, PT b, PT c) {
  double r = dot(b-a,b-a);
  if (fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if (r < 0) return a;
  if (r > 1) return b;
  return a + (b-a)*r;
}

// compute distance from c to segment between a and b
double DistancePointSegment(PT a, PT b, PT c) {
  return sqrt(dist2(c, ProjectPointSegment(a, b, c)));
}

// compute distance between point (x,y,z) and plane ax+by+cz
    =d
double DistancePointPlane(double x, double y, double z,
                          double a, double b, double c, double
                              d)
{
  return fabs(a*x+b*y+c*z-d)/sqrt(a*a+b*b+c*c);
}

// determine if lines from a to b and c to d are parallel or
     collinear
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;
}

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

// determine if line segment from a to b intersects with
// line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if (LinesCollinear(a, b, c, d)) {
    if (dist2(a, c) < EPS || dist2(a, d) < EPS ||
      dist2(b, c) < EPS || dist2(b, d) < EPS) return true;
    if (dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0 && dot(c-b, d-
        b) > 0)
      return false;
    return true;
  }
  if (cross(d-a, b-a) * cross(c-a, b-a) > 0) return false;
  if (cross(a-c, d-c) * cross(b-c, d-c) > 0) return false;
  return true;
}

// compute intersection of line passing through a and b
// with line passing through c and d, assuming that unique
// intersection exists; for segment intersection, check if
// segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d) {
  b=b-a; d=c-d; c=c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d);
}

// compute center of circle given three points
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2;
  c=(a+c)/2;
  return ComputeLineIntersection(b, b+RotateCW90(a-b), c, c+
      RotateCW90(a-c));
}

// determine if point is in a possibly non-convex polygon (
    by William
// Randolph Franklin); returns 1 for strictly interior
    points, 0 for
// strictly exterior points, and 0 or 1 for the remaining
    points.
// Note that it is possible to convert this into an *exact*
    test using
// integer arithmetic by taking care of the division
    appropriately
// (making sure to deal with signs properly) and then by
    writing exact
// tests for checking point on polygon boundary
bool PointInPolygon(const vector<PT> &p, PT q) {
  bool c = 0;
  for (int i = 0; i < p.size(); i++){
    int j = (i+1)%p.size();
    if ((p[i].y <= q.y && q.y < p[j].y ||
      p[j].y <= q.y && q.y < p[i].y) &&
```

```
    q.x < p[i].x + (p[j].x - p[i].x) * (q.y - p[i].y) / (p[
        j].y - p[i].y))
    c = !c;
  }
  return c;
}

// determine if point is on the boundary of a polygon
bool PointOnPolygon(const vector<PT> &p, PT q) {
  for (int i = 0; i < p.size(); i++)
    if (dist2(ProjectPointSegment(p[i], p[(i+1)%p.size()], q)
        , q) < EPS)
      return true;
    return false;
}

// compute intersection of line through points a and b with
// circle centered at c with radius r > 0
vector<PT> CircleLineIntersection(PT a, PT b, PT c, double r
    ) {
  vector<PT> ret;
  b = b-a;
  a = a-c;
  double A = dot(b, b);
  double B = dot(a, b);
  double C = dot(a, a) - r*r;
  double D = B*B - A*C;
  if (D < -EPS) return ret;
  ret.push_back(c+a+b*(-B+sqrt(D+EPS))/A);
  if (D > EPS)
    ret.push_back(c+a+b*(-B-sqrt(D))/A);
  return ret;
}

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

```
// This code computes the area or centroid of a (possibly
    nonconvex)
// polygon, assuming that the coordinates are listed in a
    clockwise or
// counterclockwise fashion. Note that the centroid is often
    known as
// the "center of gravity" or "center of mass".
double ComputeSignedArea(const vector<PT> &p) {
  double area = 0;
  for(int i = 0; i < p.size(); i++) {
    int j = (i+1) % p.size();
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  }
  return area / 2.0;
}

double ComputeArea(const vector<PT> &p) {
  return fabs(ComputeSignedArea(p));
}

PT ComputeCentroid(const vector<PT> &p) {
  PT c(0,0);
  double scale = 6.0 * ComputeSignedArea(p);
  for (int i = 0; i < p.size(); i++){
    int j = (i+1) % p.size();
    c = c + (p[i]+p[j])*(p[i].x*p[j].y - p[j].x*p[i].y);
  }
  return c / scale;
}

// tests whether or not a given polygon (in CW or CCW order)
     is simple
bool IsSimple(const vector<PT> &p) {
  for (int i = 0; i < p.size(); i++) {
    for (int k = i+1; k < p.size(); k++) {
      int j = (i+1) % p.size();
      int l = (k+1) % p.size();
      if (i == l || j == k) continue;
      if (SegmentsIntersect(p[i], p[j], p[k], p[l]))
        return false;
    }
  }
  return true;
}

int main() {

  // expected: (-5,2)
  cerr << RotateCCW90(PT(2,5)) << endl;
```

```
  // expected: (5,-2)
  cerr << RotateCW90(PT(2,5)) << endl;

  // expected: (-5,2)
  cerr << RotateCCW(PT(2,5),M_PI/2) << endl;

  // expected: (5,2)
  cerr << ProjectPointLine(PT(-5,-2), PT(10,4), PT(3,7)) <<
       endl;

  // expected: (5,2) (7.5,3) (2.5,1)
  cerr << ProjectPointSegment(PT(-5,-2), PT(10,4), PT(3,7))
       << " "
       << ProjectPointSegment(PT(7.5,3), PT(10,4), PT(3,7))
           << " "
       << ProjectPointSegment(PT(-5,-2), PT(2.5,1), PT(3,7))
           << endl;

  // expected: 6.78903
  cerr << DistancePointPlane(4,-4,3,2,-2,5,-8) << endl;

  // expected: 1 0 1
  cerr << LinesParallel(PT(1,1), PT(3,5), PT(2,1), PT(4,5))
       << " "
       << LinesParallel(PT(1,1), PT(3,5), PT(2,0), PT(4,5))
           << " "
       << LinesParallel(PT(1,1), PT(3,5), PT(5,9), PT(7,13))
           << endl;

  // expected: 0 0 1
  cerr << LinesCollinear(PT(1,1), PT(3,5), PT(2,1), PT(4,5))
        << " "
       << LinesCollinear(PT(1,1), PT(3,5), PT(2,0), PT(4,5))
           << " "
       << LinesCollinear(PT(1,1), PT(3,5), PT(5,9), PT(7,13))
            << endl;

  // expected: 1 1 1 0
  cerr << SegmentsIntersect(PT(0,0), PT(2,4), PT(3,1), PT
       (-1,3)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(4,3), PT
           (0,5)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(2,-1), PT
           (-2,1)) << " "
       << SegmentsIntersect(PT(0,0), PT(2,4), PT(5,5), PT
           (1,7)) << endl;

  // expected: (1,2)
```

```
  cerr << ComputeLineIntersection(PT(0,0), PT(2,4), PT(3,1),
       PT(-1,3)) << endl;

  // expected: (1,1)
  cerr << ComputeCircleCenter(PT(-3,4), PT(6,1), PT(4,5)) <<
       endl;

  vector<PT> v;
  v.push_back(PT(0,0));
  v.push_back(PT(5,0));
  v.push_back(PT(5,5));
  v.push_back(PT(0,5));

  // expected: 1 1 1 0 0
  cerr << PointInPolygon(v, PT(2,2)) << " "
      << PointInPolygon(v, PT(2,0)) << " "
      << PointInPolygon(v, PT(0,2)) << " "
      << PointInPolygon(v, PT(5,2)) << " "
      << PointInPolygon(v, PT(2,5)) << endl;

  // expected: 0 1 1 1 1
  cerr << PointOnPolygon(v, PT(2,2)) << " "
      << PointOnPolygon(v, PT(2,0)) << " "
      << PointOnPolygon(v, PT(0,2)) << " "
      << PointOnPolygon(v, PT(5,2)) << " "
      << PointOnPolygon(v, PT(2,5)) << endl;

  // expected: (1,6)
  //           (5,4) (4,5)
  //           blank line
  //           (4,5) (5,4)
  //           blank line
  //           (4,5) (5,4)
  vector<PT> u = CircleLineIntersection(PT(0,6), PT(2,6), PT
      (1,1), 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " ";
      cerr << endl;
  u = CircleLineIntersection(PT(0,9), PT(9,0), PT(1,1), 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " ";
      cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(10,10), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " ";
      cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(8,8), 5, 5);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " ";
      cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 10,
      sqrt(2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " ";
      cerr << endl;
  u = CircleCircleIntersection(PT(1,1), PT(4.5,4.5), 5, sqrt
      (2.0)/2.0);
  for (int i = 0; i < u.size(); i++) cerr << u[i] << " ";
      cerr << endl;

  // area should be 5.0
  // centroid should be (1.1666666, 1.166666)
  PT pa[] = { PT(0,0), PT(5,0), PT(1,1), PT(0,5) };
  vector<PT> p(pa, pa+4);
  PT c = ComputeCentroid(p);
  cerr << "Area: " << ComputeArea(p) << endl;
  cerr << "Centroid: " << c << endl;

  return 0;
}
```

18 GraphCutInference

```
// Special-purpose {0,1} combinatorial optimization solver
    for
// problems of the following by a reduction to graph cuts:
//
//        minimize        sum_i psi_i(x[i])
//  x[1]...x[n] in {0,1}    + sum_{i < j} phi_{ij}(x[i], x[j
    ])
//
// where
//      psi_i : {0, 1} --> R
//   phi_{ij} : {0, 1} x {0, 1} --> R
//
// such that
//   phi_{ij}(0,0) + phi_{ij}(1,1) <= phi_{ij}(0,1) + phi_{ij
    }(1,0) (*)
//
// This can also be used to solve maximization problems
    where the
// direction of the inequality in (*) is reversed.
//
// INPUT: phi -- a matrix such that phi[i][j][u][v] = phi_{
    ij}(u, v)
//        psi -- a matrix such that psi[i][u] = psi_i(u)
//        x -- a vector where the optimal solution will be
    stored
//
// OUTPUT: value of the optimal solution
//
// To use this code, create a GraphCutInference object, and
    call the
// DoInference() method. To perform maximization instead of
    minimization,
// ensure that #define MAXIMIZATION is enabled.

#include <vector>
#include <iostream>

using namespace std;

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

const int INF = 1000000000;

// comment out following line for minimization
#define MAXIMIZATION

struct GraphCutInference {
  int N;
  VVI cap, flow;
  VI reached;

  int Augment(int s, int t, int a) {
    reached[s] = 1;
    if (s == t) return a;
    for (int k = 0; k < N; k++) {
      if (reached[k]) continue;
      if (int aa = min(a, cap[s][k] - flow[s][k])) {
 if (int b = Augment(k, t, aa)) {
   flow[s][k] += b;
   flow[k][s] -= b;
   return b;
 }
      }
    }
    return 0;
  }

  int GetMaxFlow(int s, int t) {
    N = cap.size();
    flow = VVI(N, VI(N));
    reached = VI(N);

    int totflow = 0;
    while (int amt = Augment(s, t, INF)) {
      totflow += amt;
      fill(reached.begin(), reached.end(), 0);
    }
```

```
    return totflow;
  }

  int DoInference(const VVVVI &phi, const VVI &psi, VI &x) {
    int M = phi.size();
    cap = VVI(M+2, VI(M+2));
    VI b(M);
    int c = 0;

    for (int i = 0; i < M; i++) {
      b[i] += psi[i][1] - psi[i][0];
      c += psi[i][0];
      for (int j = 0; j < i; j++)
 b[i] += phi[i][j][1][1] - phi[i][j][0][1];
      for (int j = i+1; j < M; j++) {
 cap[i][j] = phi[i][j][0][1] + phi[i][j][1][0] - phi[i][j
     ][0][0] - phi[i][j][1][1];
 b[i] += phi[i][j][1][0] - phi[i][j][0][0];
 c += phi[i][j][0][0];
      }
    }

#ifdef MAXIMIZATION
    for (int i = 0; i < M; i++) {
      for (int j = i+1; j < M; j++)
 cap[i][j] *= -1;
      b[i] *= -1;
    }
    c *= -1;
#endif

    for (int i = 0; i < M; i++) {
      if (b[i] >= 0) {
 cap[M][i] = b[i];
      } else {
 cap[i][M+1] = -b[i];
 c += b[i];
      }
    }

    int score = GetMaxFlow(M, M+1);
    fill(reached.begin(), reached.end(), 0);
    Augment(M, M+1, INF);
    x = VI(M);
    for (int i = 0; i < M; i++) x[i] = reached[i] ? 0 : 1;
    score += c;
#ifdef MAXIMIZATION
    score *= -1;
#endif

    return score;
  }

};

int main() {

  // solver for "Cat vs. Dog" from NWERC 2008

  int numcases;
  cin >> numcases;
  for (int caseno = 0; caseno < numcases; caseno++) {
    int c, d, v;
    cin >> c >> d >> v;

    VVVVI phi(c+d, VVVI(c+d, VVI(2, VI(2))));
    VVI psi(c+d, VI(2));
    for (int i = 0; i < v; i++) {
      char p, q;
      int u, v;
      cin >> p >> u >> q >> v;
      u--; v--;
      if (p == 'C') {
 phi[u][c+v][0][0]++;
 phi[c+v][u][0][0]++;
      } else {
 phi[v][c+u][1][1]++;
 phi[c+u][v][1][1]++;
      }
    }

    GraphCutInference graph;
    VI x;
    cout << graph.DoInference(phi, psi, x) << endl;
  }

  return 0;
}
```

19 HLD

```
const int maxn = 1e5 + 17, lg = 17;
int n, q, col[maxn], head[maxn], par[lg][maxn], h[maxn], st[
     maxn], ft[maxn], iman[maxn << 2], sina[maxn << 2];
vector<int> g[maxn];
pair<int, int> qu[maxn];
int prep(int v = 0, int p = -1){
 if(g[v].empty() || g[v].size() == 1 && g[v][0] == p){
  col[v] = head[v] = v;
  return 1;
 }
 int sz = 1, big, mx = 0;
 for(int i = 0; i < g[v].size(); i++){
  int u = g[v][i];
  if(u == p) continue;
  par[0][u] = v;
  h[u] = h[v] + 1;
  int s = prep(u, v);
  sz += s;
  if(s > mx)
   mx = s, big = i;
 }
 col[v] = col[ g[v][big] ];
 head[ col[v] ] = v;
 swap(g[v][0], g[v][big]);
 return sz;
}
void get_st(int v = 0){
 static int time = 0;
 st[v] = time++;
 for(auto u : g[v])
  if(u != par[0][v])
   get_st(u);
 ft[v] = time;
}
int lca(int v, int u){
 if(h[u] < h[v])
  swap(v, u);
 for(int i = 0; i < lg; i++)
  if(h[u] - h[v] >> i & 1)
   u = par[i][u];
 for(int i = lg - 1; i >= 0; i--)
  if(par[i][v] != par[i][u])
   v = par[i][v], u = par[i][u];
 return v == u ? v : par[0][v];
}
int dis(int v, int u){
 return h[v] + h[u] - 2 * h[lca(v, u)];
}
void sadra(int id){
 if(sina[id] == -1)
  return;
 iman[id << 1] = iman[id << 1 | 1] = sina[id << 1] = sina[id
      << 1 | 1] = sina[id];
 sina[id] = -1;
}
void majid(int s, int e, int x, int l = 0, int r = n, int id
      = 1){
```

```
 if(s <= l && r <= e){
  iman[id] = sina[id] = x;
  return ;
 }
 if(e <= l || r <= s) return ;
 sadra(id);
 int mid = l + r >> 1;
 majid(s, e, x, l, mid, id << 1);
 majid(s, e, x, mid, r, id << 1 | 1);
 iman[id] = max(iman[id << 1], iman[id << 1 | 1]);
}
int hamid(int s, int e, int l = 0, int r = n, int id = 1){
 if(s <= l && r <= e) return iman[id];
 if(e <= l || r <= s) return 0;
 sadra(id);
 int mid = l + r >> 1;
 return max(hamid(s, e, l, mid, id << 1), hamid(s, e, mid, r
     , id << 1 | 1));
}
void change(int v, int u, int x){
 //cerr << "changeing " << v << ' ' << u << ' ' << x << '\n
     ';
 if(col[v] == col[u]){
  majid(st[u], st[v] + 1, x);
  return ;
 }
 if(col[v] != col[ par[0][v] ]){
  majid(st[v], st[v] + 1, x);
  change(par[0][v], u, x);
  return ;
 }
 majid(st[ head[ col[v] ] ], st[v] + 1, x);
 change(par[0][ head[ col[v] ] ], u, x);
}
void Change(int v, int u, int x){
 int p = lca(v, u);
 change(v, p, x);
 change(u, p, x);
}
int get_max(int v, int u){
 if(col[v] == col[u])
  return hamid(st[u], st[v] + 1);
 if(col[v] != col[ par[0][v] ])
  return max(hamid(st[v], st[v] + 1), get_max(par[0][v], u))
     ;
 return max(hamid(st[ head[ col[v] ] ], st[v] + 1), get_max(
     par[0][ head[ col[v] ] ], u));
}
int Get_max(int v, int u){
 int p = lca(v, u);
 return max(get_max(v, p), get_max(u, p));
}
int main(){
 ios::sync_with_stdio(0), cin.tie(0);
 memset(sina, -1, sizeof sina);
 cin >> n >> q;
 for(int i = 1, v, u; i < n; i++){
  cin >> v >> u;
  v--, u--;
  g[v].push_back(u);
  g[u].push_back(v);
 }
 prep();
}
```

20 Hungarian

```
typedef long long ll;
const ll INFL = (1 << 60);
using Weight = ll;
const Weight InfWeight = INFL;

Weight hungarianMin(const vector <vector<Weight>> &A) {
 if (A.empty()) return 0;
 int h = A.size(), n = A[0].size();
 if (h > n) return InfWeight;
 vector <Weight> fx(h), fy(n);
 vector<int> x(h, -1), y(n, -1);
 vector<int> t(n), s(h + 1);
 for (int i = 0; i < h;) {
  fill(t.begin(), t.end(), -1);
  s[0] = i;
  int q = 0;
  for (int p = 0; p <= q; ++p) {
   for (int k = s[p], j = 0; j < n; ++j) {
    if (fx[k] + fy[j] == A[k][j] && t[j] < 0) {
     s[++q] = y[j];
     t[j] = k;
     if (s[q] < 0) {
      for (p = j; p >= 0; j = p) {
       y[j] = k = t[j];
       p = x[k];
       x[k] = j;
      }
      ++i;
      goto continue_;
     }
    }
   }
  }
  if (0) {
  continue_:;
  } else {
   Weight d = InfWeight;
   for (int j = 0; j < n; j++)
    if (t[j] < 0) {
     for (int k = 0; k <= q; ++k)
      if (A[s[k]][j] != InfWeight)
       d = min(d, A[s[k]][j] - fx[s[k]] - fy[j]);
    }
   if (d == InfWeight)
    return InfWeight;
   for (int j = 0; j < n; ++j) {
    if (t[j] >= 0)
     fy[j] -= d;
   }
   for (int k = 0; k <= q; ++k)
    fx[s[k]] += d;
  }
 }
 Weight res = 0;
 for (int i = 0; i < h; ++i)
  res += A[i][x[i]];
 return res;
}
```

21 JavaGeometry

```
// In this example, we read an input file containing three
    lines, each
// containing an even number of doubles, separated by commas
    . The first two
// lines represent the coordinates of two polygons, given in
     counterclockwise
// (or clockwise) order, which we will call "A" and "B". The
     last line
// contains a list of points, p[1], p[2], ...
//
// Our goal is to determine:
//   (1) whether B - A is a single closed shape (as opposed
    to multiple shapes)
//   (2) the area of B - A
//   (3) whether each p[i] is in the interior of B - A
//
// INPUT:
//   0 0 10 0 0 10
```

```
//   0 0 10 10 10 0
//   8 6
//   5 1
//
// OUTPUT:
//   The area is singular.
//   The area is 25.0
//   Point belongs to the area.
//   Point does not belong to the area.

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

public class JavaGeometry {

    // make an array of doubles from a string
    static double[] readPoints(String s) {
        String[] arr = s.trim().split("\\s++");
        double[] ret = new double[arr.length];
        for (int i = 0; i < arr.length; i++) ret[i] = Double.
             parseDouble(arr[i]);
        return ret;
    }

    // make an Area object from the coordinates of a polygon
    static Area makeArea(double[] pts) {
        Path2D.Double p = new Path2D.Double();
        p.moveTo(pts[0], pts[1]);
        for (int i = 2; i < pts.length; i += 2) p.lineTo(pts[
             i], pts[i+1]);
        p.closePath();
        return new Area(p);
    }

    // compute area of polygon
    static double computePolygonArea(ArrayList<Point2D.Double
         > points) {
        Point2D.Double[] pts = points.toArray(new Point2D.
             Double[points.size()]);
        double area = 0;
        for (int i = 0; i < pts.length; i++){
            int j = (i+1) % pts.length;
            area += pts[i].x * pts[j].y - pts[j].x * pts[i].y
                 ;
        }
        return Math.abs(area)/2;
    }

// compute the area of an Area object containing several
     disjoint polygons
static double computeArea(Area area) {
    double totArea = 0;
    PathIterator iter = area.getPathIterator(null);
    ArrayList<Point2D.Double> points = new ArrayList<
         Point2D.Double>();

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

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

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

    double[] pointsA = readPoints(scanner.nextLine());
    double[] pointsB = readPoints(scanner.nextLine());
    Area areaA = makeArea(pointsA);
    Area areaB = makeArea(pointsB);
    areaB.subtract(areaA);
    // also,
    //   areaB.exclusiveOr (areaA);
    //   areaB.add (areaA);
    //   areaB.intersect (areaA);

    // (1) determine whether B - A is a single closed
         shape (as
    //     opposed to multiple shapes)
    boolean isSingle = areaB.isSingular();
    // also,
    //   areaB.isEmpty();

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

    // (2) compute the area of B - A
    System.out.println("The area is " + computeArea(areaB
         ) + ".");

    // (3) determine whether each p[i] is in the interior
         of B - A
    while (scanner.hasNextDouble()) {
        double x = scanner.nextDouble();
        assert(scanner.hasNextDouble());
        double y = scanner.nextDouble();

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

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

```
        // Each of these can be embedded in an Area object (e
            .g., new Area (rect)).

    }
}
```

22 KDTree

```
//
    ----------------------------------------------------------------------

// A straightforward, but probably sub-optimal KD-tree
    implmentation
// that's probably good enough for most things (current it's
     a
// 2D-tree)
//
//  - constructs from n points in O(n lg^2 n) time
//  - handles nearest-neighbor query in O(lg n) if points are
     well
//    distributed
//  - worst case for nearest-neighbor may be linear in
    pathological
//    case
//
// Sonny Chan, Stanford University, April 2009
//
    ----------------------------------------------------------------------


#include <iostream>
#include <vector>
#include <limits>
#include <cstdlib>

using namespace std;

// number type for coordinates, and its maximum value
typedef long long ntype;
const ntype sentry = numeric_limits<ntype>::max();

// point structure for 2D-tree, can be extended to 3D
struct point {
    ntype x, y;
    point(ntype xx = 0, ntype yy = 0) : x(xx), y(yy) {}
};

bool operator==(const point &a, const point &b)
{
    return a.x == b.x && a.y == b.y;
}

// sorts points on x-coordinate
bool on_x(const point &a, const point &b)
{
    return a.x < b.x;
}

// sorts points on y-coordinate
bool on_y(const point &a, const point &b)
{
    return a.y < b.y;
}

// squared distance between points
ntype pdist2(const point &a, const point &b)
{
    ntype dx = a.x-b.x, dy = a.y-b.y;
    return dx*dx + dy*dy;
}

// bounding box for a set of points
struct bbox
{
    ntype x0, x1, y0, y1;

    bbox() : x0(sentry), x1(-sentry), y0(sentry), y1(-sentry)
        {}

    // computes bounding box from a bunch of points
    void compute(const vector<point> &v) {
        for (int i = 0; i < v.size(); ++i) {
            x0 = min(x0, v[i].x); x1 = max(x1, v[i].x);
            y0 = min(y0, v[i].y); y1 = max(y1, v[i].y);
        }
    }

    // squared distance between a point and this bbox, 0 if
        inside
    ntype distance(const point &p) {
        if (p.x < x0) {
            if (p.y < y0)      return pdist2(point(x0, y0), p)
                ;
            else if (p.y > y1) return pdist2(point(x0, y1), p
                );
            else               return pdist2(point(x0, p.y), p
                );
        }
        else if (p.x > x1) {
            if (p.y < y0)      return pdist2(point(x1, y0), p)
                ;
            else if (p.y > y1) return pdist2(point(x1, y1), p
                );
            else               return pdist2(point(x1, p.y), p
                );
        }
        else {
            if (p.y < y0)      return pdist2(point(p.x, y0), p
                );
            else if (p.y > y1) return pdist2(point(p.x, y1),
                p);
            else               return 0;
        }
    }
};

// stores a single node of the kd-tree, either internal or
    leaf
struct kdnode
{
    bool leaf;      // true if this is a leaf node (has one
        point)
    point pt;       // the single point of this is a leaf
    bbox bound;     // bounding box for set of points in
        children

    kdnode *first, *second; // two children of this kd-node

    kdnode() : leaf(false), first(0), second(0) {}
    ~kdnode() { if (first) delete first; if (second) delete
        second; }

    // intersect a point with this node (returns squared
        distance)
    ntype intersect(const point &p) {
        return bound.distance(p);
    }

    // recursively builds a kd-tree from a given cloud of
        points
    void construct(vector<point> &vp)
    {
        // compute bounding box for points at this node
        bound.compute(vp);

        // if we're down to one point, then we're a leaf node
        if (vp.size() == 1) {
            leaf = true;
```

```
            pt = vp[0];
        }
        else {
            // split on x if the bbox is wider than high (not
                best heuristic...)
            if (bound.x1-bound.x0 >= bound.y1-bound.y0)
                sort(vp.begin(), vp.end(), on_x);
            // otherwise split on y-coordinate
            else
                sort(vp.begin(), vp.end(), on_y);

            // divide by taking half the array for each child
            // (not best performance if many duplicates in
               the middle)
            int half = vp.size()/2;
            vector<point> vl(vp.begin(), vp.begin()+half);
            vector<point> vr(vp.begin()+half, vp.end());
            first = new kdnode(); first->construct(vl);
            second = new kdnode(); second->construct(vr);
        }
    }
};

// simple kd-tree class to hold the tree and handle queries
struct kdtree
{
    kdnode *root;

    // constructs a kd-tree from a points (copied here, as it
        sorts them)
    kdtree(const vector<point> &vp) {
        vector<point> v(vp.begin(), vp.end());
        root = new kdnode();
        root->construct(v);
    }
    ~kdtree() { delete root; }

    // recursive search method returns squared distance to
        nearest point
    ntype search(kdnode *node, const point &p)
    {
        if (node->leaf) {
            // commented special case tells a point not to
               find itself
//            if (p == node->pt) return sentry;
//            else
                return pdist2(p, node->pt);
        }

        ntype bfirst = node->first->intersect(p);
        ntype bsecond = node->second->intersect(p);

        // choose the side with the closest bounding box to
            search first
        // (note that the other side is also searched if
            needed)
        if (bfirst < bsecond) {
            ntype best = search(node->first, p);
            if (bsecond < best)
                best = min(best, search(node->second, p));
            return best;
        }
        else {
            ntype best = search(node->second, p);
            if (bfirst < best)
                best = min(best, search(node->first, p));
            return best;
        }
    }

    // squared distance to the nearest
    ntype nearest(const point &p) {
        return search(root, p);
    }
};

//
    ----------------------------------------------------------------

// some basic test code here

int main()
{
    // generate some random points for a kd-tree
    vector<point> vp;
    for (int i = 0; i < 100000; ++i) {
        vp.push_back(point(rand()%100000, rand()%100000));
    }
    kdtree tree(vp);

    // query some points
    for (int i = 0; i < 10; ++i) {
        point q(rand()%100000, rand()%100000);
        cout << "Closest squared distance to (" << q.x << ",
            " << q.y << ")"
             << " is " << tree.nearest(q) << endl;
    }

    return 0;
}

//
    ----------------------------------------------------------------
```

23 LogLan

```
// Code which demonstrates the use of Java's regular
    expression libraries.
// This is a solution for
//
//   Loglan: a logical language
//   http://acm.uva.es/p/v1/134.html
//
// In this problem, we are given a regular language, whose
    rules can be
// inferred directly from the code. For each sentence in the
    input, we must
// determine whether the sentence matches the regular
    expression or not. The
// code consists of (1) building the regular expression (
    which is fairly
// complex) and (2) using the regex to match sentences.

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

public class LogLan {

    public static String BuildRegex (){
 String space = " +";

 String A = "([aeiou])";
 String C = "([a-z&&[^aeiou]])";
 String MOD = "(g" + A + ")";
 String BA = "(b" + A + ")";
 String DA = "(d" + A + ")";
 String LA = "(l" + A + ")";
 String NAM = "([a-z]*" + C + ")";
 String PREDA = "(" + C + C + A + C + A + "|" + C + A + C +
      C + A + ")";

 String predstring = "(" + PREDA + "(" + space + PREDA + ")
      *)";
 String predname = "(" + LA + space + predstring + "|" + NAM
       + ")";
 String preds = "(" + predstring + "(" + space + A + space +
       predstring + ")*)";
```

```
String predclaim = "(" + predname + space + BA + space +
     preds + "|" + DA + space +
          preds + ")";
String verbpred = "(" + MOD + space + predstring + ")";
String statement = "(" + predname + space + verbpred +
     space + predname + "|" +
          predname + space + verbpred + ")";
String sentence = "(" + statement + "|" + predclaim + ")";

return "^" + sentence + "$";
   }

   public static void main (String args[]){

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

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

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

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

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

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

24 MaxFlow

```
//be careful about memset(h,-1,sizeof h);
const int maxn = 2e3 + 17, maxm = maxn * maxn + 17, inf = 1
     e9 + 17;
void add(int v, int u, int vu, int uv = 0) {
 to[ecnt] = u, prv[ecnt] = head[v], cap[ecnt] = vu, head[v]
      = ecnt++;
 to[ecnt] = v, prv[ecnt] = head[u], cap[ecnt] = uv, head[u]
      = ecnt++;
}
int dfs(int v, int flow = inf) {
 if (v == sink || flow == 0) return f;
 if (mark[v]) return 0;
 mark[v] = 1;
 for (int e = head[v]; e != -1; e = prv[e])
  if (cap[e]) {
   int x = dfs(to[e], min(flow, cap[e]));
   if (x)
    return cap[e] -= x, cap[e ^ 1] += x, x;
  }
 return 0;
}
int maxflow() {
 int ans = 0;
 for (int tmp; (tmp = dfs(so)); ans += tmp)
  memset(mark, 0, sizeof mark);
 return ans;
}

int head[maxn], to[maxm], prv[maxm], cap[maxm], cost[maxm],
     ecnt;
void add(int v, int u, int cst, int vu, int uv = 0) {
 prv[ecnt] = head[v], to[ecnt] = u, cap[ecnt] = vu, cost[
      ecnt] = cst, head[v] = ecnt++;
 prv[ecnt] = head[u], to[ecnt] = v, cap[ecnt] = uv, cost[
      ecnt] = -cst, head[u] = ecnt++;
}
int d[maxn], par[maxn];
bool mark[maxn];
bool spfa() {
 memset(d, 63, sizeof d);
 d[so] = 0;
 int h = 0, t = 0;
 q[t++] = so, par[so] = -1;
 while (h < t) {
  int v = q[h++];
  mark[v] = 0;
  for (int e = head[v]; ~e; e = prv[e])
   if (!mark[to[e]] && cap[e] && d[to[e]] > d[v] + cost[e])
    mark[to[e]] = 1, d[to[e]] = d[v] + cost[e], q[t++] = to[e
         ], par[to[e]] = e;
 }
 return d[sink] < 1e9;
}
int mincost() {
 int ans = 0;
 while (spfa())
  for (int e = par[sink]; ~e; e = par[to[e ^ 1]])
   cap[e]--, cap[e ^ 1]++, ans += cost[e];
 return ans;
}

//dinic!

const int maxn = 2e3 + 17, maxm = 5e4 + 17, inf = 1e9;
int ptr[maxn], head[maxn], prv[maxm], to[maxm], cap[maxm], d
     [maxn], q[maxn], dis[maxn], so = maxn - 1, sink = maxn
     - 2, ecnt;
void init(){
```

```
 memset(head, -1, sizeof head);
 ecnt = 0;
}
void add(int v, int u, int vu, int uv = 0){
 to[ecnt] = u, prv[ecnt] = head[v], cap[ecnt] = vu, head[v]
      = ecnt++;
 to[ecnt] = v, prv[ecnt] = head[u], cap[ecnt] = uv, head[u]
      = ecnt++;
}
bool bfs(){
 memset(dis, 63, sizeof dis);
 dis[so] = 0;
 int h = 0, t = 0;
 q[t++] = so;
 while(h < t){
  int v = q[h++];
  for(int e = head[v]; e >= 0; e = prv[e])
   if(cap[e] && dis[ to[e] ] > dis[v] + 1){
    dis[ to[e] ] = dis[v] + 1, q[t++] = to[e];
    if(to[e] == sink)
     return 1;
   }
 }
 return 0;
}
int dfs(int v, int f = inf){
 if(v == sink || f == 0)
  return flow;
 int ret = 0;
 for(int &e = ptr[v]; e >= 0; e = prv[e])
  if(dis[v] == dis[ to[e] ] - 1){
   int x = dfs(to[e], min(f, cap[e]));
   f -= x, ret += x;
   cap[e] -= x, cap[e ^ 1] += x;
   if(!f)
    break;
  }
 return ret;
}
int mf(){
 int ans = 0;
 while(bfs()){
  memcpy(ptr, head, sizeof ptr);
  ans += dfs(so);
 }
 return ans;
}
```

25 MaxIndependentSet

```
bool dfs(int v){
    if(mark[v]) return 0;
    mark[v] = 1;
    for(auto u : adj[v][0])
 if(mat[u][1] == -1 || dfs(mat[u][1]))
     return mat[v][0] = u, mat[u][1] = v, 1;
    return 0;
}
void dfs(int v, int part){
    seen[v][part] = 1;
    for(auto u : adj[v][part])
 if(!seen[u][!part]){
     bad[u] = 1;
     seen[u][!part] = 1;
     dfs(mat[u][!part], part);
 }
}
void maximum_independent_set(){
    memset(mat, -1, sizeof mat);
    bool br = 0;
    int ans = n;
    while(br ^= 1){
 memset(mark, 0, sizeof mark);
 for(int i = 0; i < n; i++)
     if(mat[i][0] == -1 && dfs(i))
  ans--, br = 0;
    }
    for(int i = 0; i < n; i++)
 for(int j = 0; j < 2; j++)
     if(seen[i][j] == 0 && mat[i][j] == -1)
  dfs(i, j);
    cout << ans << '\n';
    for(int i = 0; i < n; i++)
 if(bad[i] == 0 && seen[i][0] == 1)
     cout << i + 1 << ' ';
    cout << '\n';
}
```

26 OrderedSet

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
tree<int, null_type, less<int>, rb_tree_tag,
     tree_order_statistics_node_update> os;
```

27 Primes

```
// O(sqrt(x)) Exhaustive Primality Test
#include <cmath>
#define EPS 1e-7
typedef long long LL;
bool IsPrimeSlow (LL x)
{
  if(x<=1) return false;
  if(x<=3) return true;
  if (!(x%2) || !(x%3)) return false;
  LL s=(LL)(sqrt((double)(x))+EPS);
  for(LL i=5;i<=s;i+=6)
  {
    if (!(x%i) || !(x%(i+2))) return false;
  }
  return true;
}
// Primes less than 1000:
//      2     3     5     7    11    13    17    19    23    29
     31    37
//     41    43    47    53    59    61    67    71    73    79
     83    89
//     97   101   103   107   109   113   127   131   137   139
    149   151
//    157   163   167   173   179   181   191   193   197   199
    211   223
//    227   229   233   239   241   251   257   263   269   271
    277   281
//    283   293   307   311   313   317   331   337   347   349
    353   359
//    367   373   379   383   389   397   401   409   419   421
    431   433
//    439   443   449   457   461   463   467   479   487   491
    499   503
//    509   521   523   541   547   557   563   569   571   577
    587   593
//    599   601   607   613   617   619   631   641   643   647
    653   659
//    661   673   677   683   691   701   709   719   727   733
    739   743
//    751   757   761   769   773   787   797   809   811   821
    823   827
//    829   839   853   857   859   863   877   881   883   887
    907   911
//    919   929   937   941   947   953   967   971   977   983
    991   997

// Other primes:
//    The largest prime smaller than 10 is 7.
```

```
//   The largest prime smaller than 100 is 97.
//   The largest prime smaller than 1000 is 997.
//   The largest prime smaller than 10000 is 9973.
//   The largest prime smaller than 100000 is 99991.
//   The largest prime smaller than 1000000 is 999983.
//   The largest prime smaller than 10000000 is 9999991.
//   The largest prime smaller than 100000000 is 99999989.
//   The largest prime smaller than 1000000000 is 999999937.
//   The largest prime smaller than 10000000000 is
    9999999967.
//   The largest prime smaller than 100000000000 is
    99999999977.
//   The largest prime smaller than 1000000000000 is
    999999999989.
//   The largest prime smaller than 10000000000000 is
    9999999999971.
//   The largest prime smaller than 100000000000000 is
    99999999999973.
//   The largest prime smaller than 1000000000000000 is
    999999999999989.
//   The largest prime smaller than 10000000000000000 is
    9999999999999937.
//   The largest prime smaller than 100000000000000000 is
    99999999999999997.
//   The largest prime smaller than 1000000000000000000 is
    999999999999999989.
```

28 SCC

```
bool mark[maxn], in_comp[maxn];
vector<int> g[maxn], rg[maxn];
void dfs(int v, vector<int> *g, vector<int> &vec){
    mark[v] = 1;
    for(auto u : g[v])
 if(!mark[u])
     dfs(u, g, vec);
    vec.push_back(v);
}
bool mark[maxn], in_comp[maxn];
int main(){
    vector<int> all;
    for(int i = 0; i < n; i++)
 if(!mark[i])
     dfs(i, g, all);
    memset(mark, 0, sizeof mark);
    reverse(all.begin(), all.end());
    for(auto v : all){
 if(mark[v]) continue;
 vector<int> comp;
 dfs(v, rg, comp);
 for(auto u : comp) in_comp[u] = 1;
 for(auto u : comp) in_comp[u] = 0;
    }
}
```

29 SegmentPointer

```
struct Node{
    Node *L, *R;
    ll iman;
    int sina;
    Node(){
 iman = sina = 0;
    }
    void arpa(){
 if(L) return ;
 L = new Node();
 R = new Node();
    }
    void majid(int s, int e, int x, int l = 0, int r = tb){
 if(s <= l && r <= e){
     sina += x;
     return ;
 }
 if(e <= l || r <= s)
     return ;
 arpa();
 int mid = l + r >> 1;
 L -> majid(s, e, x, l, mid);
 R -> majid(s, e, x, mid, r);
 iman = L -> iman + L -> sina * (ll) (mid - l) + R -> iman +
      R -> sina * (ll) (r - mid);
    }
    ll hamid(int s, int e, int l = 0, int r = tb){
 if(s <= l && r <= e){
     return iman + sina * (ll) (r - l);
 }
 if(e <= l || r <= s) return 0;
 arpa();
 int mid = l + r >> 1;
 return L -> hamid(s, e, l, mid) + R -> hamid(s, e, mid, r)
     + sina * (ll) (min(r, e) - max(l, s));
    }
}  root;
```

30 SuffixArray

```
#define REP(i, n) for (int i = 0; i < (int)(n); ++i)

namespace SuffixArray
{
    const int MAXN = 1 << 21;
    char * S;
    int N, gap;
    int sa[MAXN], pos[MAXN], tmp[MAXN], lcp[MAXN];

    bool sufCmp(int i, int j)
    {
 if (pos[i] != pos[j])
     return pos[i] < pos[j];
 i += gap;
 j += gap;
 return (i < N && j < N) ? pos[i] < pos[j] : i > j;
    }

    void buildSA()
    {
 N = strlen(S);
 REP(i, N) sa[i] = i, pos[i] = S[i];
 for (gap = 1;; gap *= 2)
     {
  sort(sa, sa + N, sufCmp);
  REP(i, N - 1) tmp[i + 1] = tmp[i] + sufCmp(sa[i], sa[i +
       1]);
  REP(i, N) pos[sa[i]] = tmp[i];
  if (tmp[N - 1] == N - 1) break;
     }
    }

    void buildLCP()
    {
 for (int i = 0, k = 0; i < N; ++i) if (pos[i] != N - 1)
     {
         for (int j = sa[pos[i] + 1]; S[i + k] == S[j + k];)
      ++k;
         lcp[pos[i]] = k;
         if (k)--k;
     }
    }
} // end namespace SuffixArray
```

31 aho

```
const int maxn=1e5*52+12;
int f[maxn],nxt[maxn][52],mark[maxn],co[maxn],sz=1,q
     [100012],qu,haqani[1012];
int insert(string &s){
 int v=0;
 for(int i=0;i<s.size();i++){
  if(!nxt[v][s[i]])
   nxt[v][s[i]]=sz++;
  v=nxt[v][s[i]];
 }
 return v;
}
void aho(){
 int h=0,t=0;
 for(int i=0;i<52;i++)
  if(nxt[0][i])
   q[t++]=nxt[0][i];
 while(h<t){
  int v=q[h];
  for(int i=0;i<52;i++)
   if(nxt[v][i])
    f[ nxt[v][i] ] = nxt[ f[v] ][i],q[t++]=nxt[v][i];
   else
    nxt[v][i] = nxt[ f[v] ][i];
  h++;
 }
}
```

32 and-convolution

```
void transform(int *from, int *to)
{
    if(to - from == 1)
        return;
    int *mid = from + (to - from) / 2;
    transform(from, mid);
    transform(mid, to);
    for(int i = 0; i < mid - from; i++)
    {
        int a = *(from + i);
        int b = *(mid + i);
        *(from + i) = b;
        *(mid + i) = a + b;
    }
}
void inverse(int *from, int *to)
{
    if(to - from == 1)
        return;
    int *mid = from + (to - from) / 2;
    inverse(from, mid);
    inverse(mid, to);
    for(int i = 0; i < mid - from; i++)
    {
        int a = *(from + i);
        int b = *(mid + i);
        *(from + i) = -a + b;
        *(mid + i) = a;
    }
}
```

33 kmp

```
const int maxn = 5e6 + 17;
string s, p;
int f[maxn];
int main(){
 ios::sync_with_stdio(0),cin.tie(0);
 cin >> s >> p;
 int k = 0;
 for(int i = 1; i < p.size(); i++){
  while(k && p[k] != p[i]) k = f[k];
  if(p[k] == p[i]) k++;
  f[i + 1] = k;
 }
 k = 0;
 for(int i = 0; i < s.size(); i++){
  while(k && p[k] != s[i]) k = f[k];
  if(p[k] == s[i]) k++;
  if(k == p.size()){
   cerr << "A match occurred on " << i << '\n';
   k = f[k];
  }
 }
 return 0;
}
```

34 or-convolution

```
void transform(int *from, int *to)
{
    if(to - from == 1)
        return;
    int *mid = from + (to - from) / 2;
    transform(from, mid);
    transform(mid, to);
    for(int i = 0; i < mid - from; i++)
        *(mid + i) += *(from + i);
}

void inverse(int *from, int *to)
{
    if(to - from == 1)
        return;
    int *mid = from + (to - from) / 2;
    inverse(from, mid);
    inverse(mid, to);
    for(int i = 0; i < mid - from; i++)
        *(mid + i) -= *(from + i);
}
```

35 polard

```
#define MAXL (50000>>5)+1
#define GET(x) (mark[x>>5]>>(x&31)&1)
#define SET(x) (mark[x>>5] |= 1<<(x&31))
int mark[MAXL];
int P[50000], Pt = 0;
void sieve() {
    register int i, j, k;
    SET(1);
    int n = 46340;
    for (i = 2; i <= n; i++) {
        if (!GET(i)) {
            for (k = n/i, j = i*k; k >= i; k--, j -= i)
                SET(j);
            P[Pt++] = i;
        }
    }
}
long long mul(unsigned long long a, unsigned long long b,
     unsigned long long mod) {
    long long ret = 0;
    for (a %= mod, b %= mod; b != 0; b >>= 1, a <<= 1, a = a
         >= mod ? a - mod : a) {
        if (b&1) {
            ret += a;
```

```
            if (ret >= mod) ret -= mod;
        }
    }
    return ret;
}
void exgcd(long long x, long long y, long long &g, long long
      &a, long long &b) {
    if (y == 0)
        g = x, a = 1, b = 0;
    else
        exgcd(y, x%y, g, b, a), b -= (x/y) * a;
}
long long llgcd(long long x, long long y) {
    if (x < 0)   x = -x;
    if (y < 0)   y = -y;
    if (!x || !y)  return x + y;
    long long t;
    while (x%y)
        t = x, x = y, y = t%y;
    return y;
}
long long inverse(long long x, long long p) {
    long long g, b, r;
    exgcd(x, p, g, r, b);
    if (g < 0) r = -r;
    return (r%p + p)%p;
}
long long mpow(long long x, long long y, long long mod) { //
      mod < 2^32
    long long ret = 1;
    while (y) {
        if (y&1)
            ret = (ret * x)%mod;
        y >>= 1, x = (x * x)%mod;
    }
    return ret % mod;
}
long long mpow2(long long x, long long y, long long mod) {
    long long ret = 1;
    while (y) {
        if (y&1)
            ret = mul(ret, x, mod);
        y >>= 1, x = mul(x, x, mod);
    }
    return ret % mod;
}
int isPrime(long long p) { // implements by miller-babin
    if (p < 2 || !(p&1)) return 0;
    if (p == 2)    return 1;
    long long q = p-1, a, t;
    int k = 0, b = 0;
    while (!(q&1)) q >>= 1, k++;
    for (int it = 0; it < 2; it++) {
        a = rand()%(p-4) + 2;
        t = mpow2(a, q, p);
        b = (t == 1) || (t == p-1);
        for (int i = 1; i < k && !b; i++) {
            t = mul(t, t, p);
            if (t == p-1)
                b = 1;
        }
        if (b == 0)
            return 0;
    }
    return 1;
}
long long pollard_rho(long long n, long long c) {
    long long x = 2, y = 2, i = 1, k = 2, d;
    while (true) {
        x = (mul(x, x, n) + c);
        if (x >= n) x -= n;
        d = llgcd(x - y, n);
        if (d > 1) return d;
        if (++i == k) y = x, k <<= 1;
    }
    return n;
}
void factorize(int n, vector<long long> &f) {
    for (int i = 0; i < Pt && P[i]*P[i] <= n; i++) {
     if (n%P[i] == 0) {
      while (n%P[i] == 0)
       f.push_back(P[i]), n /= P[i];
     }
    }
    if (n != 1) f.push_back(n);
}
void llfactorize(long long n, vector<long long> &f) {
    if (n == 1)
        return ;
    if (n < 1e+9) {
        factorize(n, f);
        return ;
    }
    if (isPrime(n)) {
        f.push_back(n);
        return ;
    }
    long long d = n;
    for (int i = 2; d == n; i++)
        d = pollard_rho(n, i);
    llfactorize(d, f);
    llfactorize(n/d, f);
}
```

36 xor-convolution

```
void transform(int *from, int *to)
{
    if(to - from == 1)
        return;
    int *mid = from + (to - from) / 2;
    transform(from, mid);
    transform(mid, to);
    for(int i = 0; i < mid - from; i++)
    {
        int a = *(from + i);
        int b = *(mid + i);
        *(from + i) = a + b;
        *(mid + i) = a - b;
    }
}

void inverse(int *from, int *to) {
    transform(from, to);
    for (int *i = from; i < to; i++) (*i) /= (to-from);
}
```

37 z-function

```
vector<int> z_function(string s) {
    int n = (int) s.length();
    vector<int> z(n);
    for (int i = 1, l = 0, r = 0; i < n; ++i) {
  if (i <= r)
     z[i] = min (r - i + 1, z[i - l]);
  while (i + z[i] < n && s[z[i]] == s[i + z[i]])
     ++z[i];
  if (i + z[i] - 1 > r)
     l = i, r = i + z[i] - 1;
    }
    return z;
}
```