Pontificia Universidad Catolica del Peru

1. Great Circle Distance

```
double greatCircle( double laa, double loa, double lab, double lob )
{
   double PI = acos( -1.0 ), R = 6378.0;
   double u[3] = { cos( laa ) * sin( loa ), cos( laa ) * cos( loa ), sin( laa ) };
   double v[3] = { cos( lab ) * sin( lob ), cos( lab ) * cos( lob ), sin( lab ) };
   double dot = u[0]*v[0] + u[1]*v[1] + u[2]*v[2];
   bool flip = false;
   if( dot < 0.0 )
   {
      flip = true;
      for( int i = 0; i < 3; i++ ) v[i] = -v[i];
   }
   double cr[3] = { u[1]*v[2] - u[2]*v[1], u[2]*v[0] - u[0]*v[2], u[0]*v[1] -
u[1]*v[0] };
   double theta = asin( sqrt( cr[0]*cr[0] + cr[1]*cr[1] + cr[2]*cr[2] ) );
   double len = theta * R;
   if( flip ) len = PI * R - len;
   return len;
}
```

2. Graph facts:

A graph is Hamiltonian if and only if its closure is Hamiltonian.
As complete graphs are Hamiltonian, all graphs whose closure is complete are Hamiltonian, which is the content of the following earlier theorems by Dirac and Ore.
Dirac (1952)
A simple graph with n vertices ($n \geq 3$) is Hamiltonian if each vertex has degree $n/2$ or greater.[1]
Ore (1960)
A graph with n vertices ($n \geq 3$) is Hamiltonian if, for each pair of non-adjacent vertices, the sum of their degrees is n or greater (see Ore's theorem).
The following theorems can be regarded as directed versions:
Ghouila-Houiri (1960)
A strongly connected simple directed graph with n vertices is Hamiltonian or some vertex has a full degree smaller than n.
Meyniel (1973)
A strongly connected simple directed graph with n vertices is Hamiltonian or the sum of full degrees of some two distinct non-adjacent vertices is smaller than $2n - 1$.
The number of vertices must be doubled because each undirected edge corresponds to two directed arcs and thus the degree of a vertex in the directed graph is twice the

degree in the undirected graph.

A graph *G* is 2-edge-connected if and only if it has an orientation that is strongly connected.

Teorema de Euler: para un grafo planar conexo V - E + F = 2
minimum vertex cover <= maximum matching (la igualdad ocurre con grafos bipartitos)
minimum vertex cover <= maximum independent set
teorema de Kirchhoff: el numero de spanning trees de G es igual al determinante de cualquier cofactor de la matriz laplaciana de G
numero de labeled trees de n vertices: n ^ (n - 2)

The number of spanning trees in a complete graph K_n with degrees $d_1, d_2, \ldots, d_n$ is equal to the multinomial coefficient

$$\binom{n-2}{d_1-1, d_2-1, \ldots, d_n-1} = \frac{(n-2)!}{(d_1-1)!(d_2-1)!\cdots(d_n-1)!}.$$

Si G es planar y v >=3, entonces e <= 3v -6
If G is planar and if $v \geq 3$ and there are no cycles of length 3, then $e \leq 2v - 4$.

siempre probar tests con los siguientes grafos:

3. MINIMUM COST ARBORESCENCE

```
#define MAX_V 1000
typedef int edge_cost;
edge_cost INF = INT_MAX;

int V,root,prev[MAX_V];
bool adj[MAX_V][MAX_V];
edge_cost G[MAX_V][MAX_V],MCA;
bool visited[MAX_V],cycle[MAX_V];

void add_edge(int u, int v, edge_cost c){
    if(adj[u][v]) G[u][v] = min(G[u][v],c);
    else G[u][v] = c;
    adj[u][v] = true;
}

void dfs(int v){
    visited[v] = true;

    for(int i = 0;i<V;++i)
        if(!visited[i] && adj[v][i])
            dfs(i);
}

bool check(){
    memset(visited,false,sizeof(visited));
    dfs(root);

    for(int i = 0;i<V;++i)
        if(!visited[i])
            return false;

    return true;
}

int exist_cycle(){
    prev[root] = root;

    for(int i = 0;i<V;++i){
        if(!cycle[i] && i!=root){
            prev[i] = i; G[i][i] = INF;
            for(int j = 0;j<V;++j)
                if(!cycle[j] && adj[j][i] && G[j][i]<G[prev[i]][i])
                    prev[i] = j;
        }
    }
    for(int i = 0,j;i<V;++i){
        if(cycle[i]) continue;
        memset(visited,false,sizeof(visited));
        j = i;
        while(!visited[j]){
            visited[j] = true;
            j = prev[j];
        }
        if(j==root) continue;
        return j;
    }
    return -1;
}

void update(int v){
    MCA += G[prev[v]][v];
    for(int i = prev[v];i!=v;i = prev[i]){
        MCA += G[prev[i]][i];
        cycle[i] = true;
    }
    for(int i = 0;i<V;++i)
        if(!cycle[i] && adj[i][v])
            G[i][v] -= G[prev[v]][v];
    for(int j = prev[v];j!=v;j = prev[j]){
        for(int i = 0;i<V;++i){
            if(cycle[i]) continue;

            if(adj[i][j]){
                if(adj[i][v]) G[i][v] = min(G[i][v],G[i][j]-G[prev[j]][j]);
                else G[i][v] = G[i][j]-G[prev[j]][j];
                adj[i][v] = true;
            }

            if(adj[j][i]){
                if(adj[v][i]) G[v][i] = min(G[v][i],G[j][i]);
                else G[v][i] = G[j][i];
                adj[v][i] = true;
            }
```

```
      }
   }
}

bool min_cost_arborescence(int _root){
   root = _root;
   if(!check()) return false;

   memset(cycle,false,sizeof(cycle));
   MCA = 0;

   int v;

   while((v = exist_cycle())!=-1)
      update(v);

   for(int i = 0;i<V;++i)
      if(i!=root && !cycle[i])
         MCA += G[prev[i]][i];

   return true;
}
```

4. DETERMINE IF A GIVEN POLYNOMIAL HAS NO REPEATED ROOT

Sylvester matrix
Formally, let *p* and *q* be two nonzero polynomials, respectively of degree *m* and *n*. Thus:

$$p(z)=p_0+p_1z+p_2z^2+\cdots+p_mz^m,\; q(z)=q_0+q_1z+q_2z^2+\cdots+q_nz^n.$$

The Sylvester matrix associated to *p* and *q* is then the

$$(n+m)\times(n+m)$$

Thus, if *m*=4 and *n*=3, the matrix

$$S_{p,q}=\begin{pmatrix} p_4 & p_3 & p_2 & p_1 & p_0 & 0 & 0\\ 0 & p_4 & p_3 & p_2 & p_1 & p_0 & 0\\ 0 & 0 & p_4 & p_3 & p_2 & p_1 & p_0\\ q_3 & q_2 & q_1 & q_0 & 0 & 0 & 0\\ 0 & q_3 & q_2 & q_1 & q_0 & 0 & 0\\ 0 & 0 & q_3 & q_2 & q_1 & q_0 & 0\\ 0 & 0 & 0 & q_3 & q_2 & q_1 & q_0 \end{pmatrix}.$$

is:

// p := p mod q, scaled to integer coefficients; return new degree of p

5. Division de polinomios y gcd

```
public static int mod(BigInteger[] p, int dp, BigInteger[] q, int dq) {

  while (dp >= dq) {
    // p := p * q[dq] - q * p[dp] * x^(dp-dq)
    BigInteger f = p[dp];
    for (int i=0 ; i<=dp ; i++)
      p[i] = p[i].multiply(q[dq]);
    for (int i=0 ; i<=dq ; i++)
      p[dp-dq+i] = p[dp-dq+i].subtract(q[i].multiply(f));
    // degree of p has been reduced by at least 1

    while (dp >= 0 && p[dp].signum() == 0)
      --dp;
  }
  // divide coefficients of p by their gcd to keep them small
  BigInteger d = BigInteger.ZERO;
  for (int i=0 ; i<=dp ; i++)
    d = d.gcd(p[i]);
  for (int i=0 ; i<=dp ; i++)
    p[i] = p[i].divide(d);
  return dp;
}

// are the polynomials p and q relatively prime?
public static boolean relprime(BigInteger[] p, int dp, BigInteger[] q, int dq) {
  // dp,dq = degrees of p,q (-1 for zero polynomial)
  while (dp >= 0 && dq >= 0)
    // replace the greater of p,q by its remainder when divided by the other
    if (dp >= dq)
      dp = mod(p, dp, q, dq);
    else
      dq = mod(q, dq, p, dp);
  // if either of p,q is zero, the other is their gcd which must be constant
  return dp <= 0 && dq <= 0;
}

public static void main(String[] arg) throws Exception {
  StreamTokenizer st = new StreamTokenizer(new BufferedReader(new
InputStreamReader(System.in)));
  st.nextToken();
  for (int t = (int) st.nval ; t > 0 ; t--) {
    st.nextToken();
    int n = (int) st.nval;
    // a[i] is coefficient of x^i
    BigInteger[] a = new BigInteger[n+1];
    for (int i=0 ; i<=n ; i++) {
      st.nextToken();
      a[n-i] = BigInteger.valueOf((int) st.nval);
    }
    // d is derivative of polynomial a
    BigInteger[] d = new BigInteger[n];
    for (int i=0 ; i<n ; i++)
      d[i] = a[i+1].multiply(BigInteger.valueOf(i+1));
    // check a and d for common factors
    System.out.println(relprime(a, n, d, n-1) ? "Yes!" : "No!");
  }
}

}
```

6. Stable Marriage

```
* MAIN FUNCTION: stableMarriage()
*     - m:        number of men.
*     - n:        number of women (must be at least as large as m).
*     - L[i][]:   the list of women in order of decreasing preference of man i.
*     - R[j][i]:  the attractiveness of i to j.
* OUTPUTS:
*     - L2R[]:    the mate of man i (always between 0 and n-1)
*     - R2L[]:    the mate of woman j (or -1 if single)
*#define MAXM 1024
#define MAXW 1024

int m, n;
int L[MAXM][MAXW], R[MAXW][MAXM];
int L2R[MAXM], R2L[MAXW];

int p[MAXM];

void stableMarriage()
{
    static int p[128];
    memset( R2L, -1, sizeof( R2L ) );
    memset( p, 0, sizeof( p ) );

    // Each man proposes...
    for( int i = 0; i < m; i++ )
    {
      int man = i;
      while( man >= 0 )
      {
        // to the next woman on his list in order of decreasing preference,
        // until one of them accepts;
        int wom;
        while( 1 )
        {
          wom = L[man][p[man]++];
          if( R2L[wom] < 0 || R[wom][man] > R[wom][R2L[wom]] ) break;
        }

        // Remember the old husband of wom.
        int hubby = R2L[wom];

        // Marry man and wom.
        R2L[L2R[man] = wom] = man;

        // If a guy was dumped in the process, remarry him now.
        man = hubby;
      }
    }
}
```

7. PAPE CONRADT

```
int n, mate[100], inner[100], gf[100];
#define NONE -1
.

for(int i = 0; i < n; i++){
                mate[i] = NONE;
                gf[i] = NONE;
        }
        for(int u = 0; u < n; u++)
                if(mate[u] == NONE)
                        for(int i = 0; i < adj[u].size(); i++){
                                int v = adj[u][i];
                                if(mate[v] == NONE){
                                        mate[u] = v; mate[v] = u;
                                        break;
                                }
                        }
        int w;
        for(int r = 0; r < n; r++)
                if(mate[r] == NONE){
                        memset(inner, 0, sizeof inner);
                        gf[r] = NONE;
                        inner[r] = true;
                        queue<int> q;
                        q.push(r);
                        bool ok = false;

                        while(!q.empty() && !ok){
                                int u = q.front(); q.pop();
                                for(int i = 0; i < adj[u].size(); i++){
                                        int v = adj[u][i];
                                        if(!inner[v]){
                                                if(mate[v] == NONE){
                                                        while(u != NONE){
                                                                w =
mate[u];
                                                                mate[u] =
v; mate[v] = u;
                                                                v = w;
                                                                u = gf[u];
                                                        }
                                                        ok = true;
                                                        break;
                                                }

                                                for(w = u; w != NONE &&
w != v; w = gf[w]);

                                                if(w == NONE){
                                                        inner[v] = true;
                                                        w = mate[v];
                                                        gf[w] = u;
                                                        q.push(w);
                                                }
                                        }
                                }
                        }

                }
```

8. GEOMETRY ALGORITHMS

Formulas for regular tetrahedron

For a regular tetrahedron of edge length *a*:

$$V = \frac{abc}{6}\sqrt{1 + 2\cos\alpha\cos\beta\cos\gamma - \cos^2\alpha - \cos^2\beta - \cos^2\gamma}$$

where α, β, γd. The angle αd to the vertices b and c. The angle βa and c, while γa and b.
Given the distances between the vertices of a tetrahedron the volume can be computed using the Cayley–Menger

Pontificia Universidad Catolica del Peru

$$288 \cdot V^2 = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{12}^2 & d_{13}^2 & d_{14}^2 \\ 1 & d_{12}^2 & 0 & d_{23}^2 & d_{24}^2 \\ 1 & d_{13}^2 & d_{23}^2 & 0 & d_{34}^2 \\ 1 & d_{14}^2 & d_{24}^2 & d_{34}^2 & 0 \end{vmatrix}$$

determinant:
where the subscripts
$i, j \in \{1, 2, 3, 4\}$ $\{a, b, c, d\}$ d_{ij} Tartaglia's formula, is essentially due to the painter Piero della Francesca in the 15th century, as a three dimensional analogue of the 1st century Heron's formula for the area of a triangle.[4]
Distance between the edges
Then another volume formula is given

$$V = \frac{d|(\mathbf{a} \times (\mathbf{b} - \mathbf{c}))|}{6}.$$

by

If OABC forms a generalized tetrahedron with a vertex O as the origin and vectors
$\mathbf{a}, \mathbf{b}\mathbf{c}$ $r = \frac{6V}{|\mathbf{b} \times \mathbf{c}| + |\mathbf{c} \times \mathbf{a}| + |\mathbf{a} \times \mathbf{b}| + |(\mathbf{b} \times \mathbf{c}) + (\mathbf{c} \times \mathbf{a}) + (\mathbf{a} \times \mathbf{b})|}$

and the radius of the circumsphere is given

$$R = \frac{|\mathbf{a^2}(\mathbf{b} \times \mathbf{c}) + \mathbf{b^2}(\mathbf{c} \times \mathbf{a}) + \mathbf{c^2}(\mathbf{a} \times \mathbf{b})|}{12V}$$

by:
which gives the radius of the twelve-point

$$r_T = \frac{|\mathbf{a^2}(\mathbf{b} \times \mathbf{c}) + \mathbf{b^2}(\mathbf{c} \times \mathbf{a}) + \mathbf{c^2}(\mathbf{a} \times \mathbf{b})|}{36V}$$

sphere:
where: $6V = |\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c})|.$
Nota bene, in the formulas throughout this section, the scalar a^2 represents the inner vector product $a \cdot a$, similarly b^2 and c^2 .
The vector position of various centers are given as follows:

The centroid $\mathbf{G} = \frac{\mathbf{a} + \mathbf{b} + \mathbf{c}}{4}.$

$$\mathbf{O} = \frac{\mathbf{a}^2(\mathbf{b}\times\mathbf{c}) + \mathbf{b}^2(\mathbf{c}\times\mathbf{a}) - \mathbf{c}^2(\mathbf{a}\times\mathbf{b})}{2\mathbf{a}\cdot(\mathbf{b}\times\mathbf{c})},$$

The circumcenter
The Monge

$$\mathbf{M} = \frac{\mathbf{a}\cdot(\mathbf{b}+\mathbf{c})(\mathbf{b}\times\mathbf{c}) + \mathbf{b}\cdot(\mathbf{c}+\mathbf{a})(\mathbf{c}\times\mathbf{a}) + \mathbf{c}\cdot(\mathbf{a}+\mathbf{b})(\mathbf{a}\times\mathbf{b})}{2\mathbf{a}\cdot(\mathbf{b}\times\mathbf{c})}.$$

point
The Euler line relationships

$$\mathbf{G} = \mathbf{M} + \frac{1}{2}(\mathbf{O} - \mathbf{M})$$ $\mathbf{T} = \mathbf{M} + \frac{1}{3}(\mathbf{O} - \mathbf{M})$

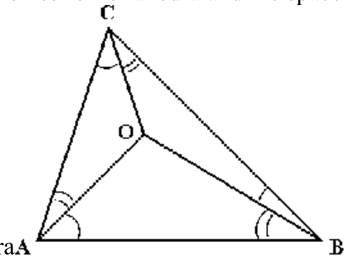
are:
where $\mathbf{T}$

$$\mathbf{a} \cdot \mathbf{O} = \frac{\mathbf{a^2}}{2} \qquad \mathbf{b} \cdot \mathbf{O} = \frac{\mathbf{b^2}}{2} \qquad \mathbf{c} \cdot \mathbf{O} = \frac{\mathbf{c^2}}{2}$$

Also:

$$\mathbf{a} \cdot \mathbf{M} = \frac{\mathbf{a} \cdot (\mathbf{b} + \mathbf{c})}{2} \qquad \mathbf{b} \cdot \mathbf{M} = \frac{\mathbf{b} \cdot (\mathbf{c} + \mathbf{a})}{2} \qquad \mathbf{c} \cdot \mathbf{M} = \frac{\mathbf{c} \cdot (\mathbf{a} + \mathbf{b})}{2}.$$

and:
A law of sines for tetrahedra and the space of all shapes of



tetrahedra

A corollary of the usual law of sines is that in a tetrahedron with vertices *O*, *A*, *B*, *C*, we
have

$$\sin\angle OAB \cdot \sin\angle OBC \cdot \sin\angle OCA = \sin\angle OAC \cdot \sin\angle OCB \cdot \sin\angle OBA.$$

9. Manacher

```
vector<int> d1 (n);
int l=0, r=-1;
```

```
for (int i=0; i<n; ++i) {
    int k = (i>r ? 0 : min (d1[l+r-i], r-i)) + 1;
    while (i+k < n && i-k >= 0 && s[i+k] == s[i-k])  ++k;
    d1[i] = --k;
    if (i+k > r)
            l = i-k,  r = i+k;
}

vector<int> d2 (n);
l=0, r=-1;
for (int i=0; i<n; ++i) {
    int k = (i>r ? 0 : min (d2[l+r-i+1], r-i+1)) + 1;
    while (i+k-1 < n && i-k >= 0 && s[i+k-1] == s[i-k])  ++k;
    d2[i] = --k;
    if (i+k-1 > r)
            l = i-k,  r = i+k-1;
}
```

10. // Number Theoretic Algorithms //

```
/*******
* GCD *
*******
* Euclidean algorithm. Works on non-negative integers.
**/
int gcd( int a, int b ) { return( b == 0 ? a : gcd( b, a % b ) ); }
long long gcd( long long a, long long b ) { return( b == 0 ? a : gcd( b, a % b ) ); }
template< Int > Int gcd( Int a, Int b ) { return( b == 0 ? a : gcd( b, a % b ); }

/************************
* A triple of integers *
************************
* USED BY: egcd, msolve, inverse, ldioph
**/
template< class Int >
struct Triple
{
    Int d, x, y;
    Triple( Int q, Int w, Int e ) : d( q ), x( w ), y( e ) {}
};

/****************
* Extended GCD *
****************
* Given nonnegative a and b, computes d = gcd( a, b )
* along with integers x and y, such that d = ax + by
* and returns the triple (d, x, y).
* WARNING: needs a small modification to work on
*      negative integers (operator% fails).
* REQUIRES: struct Triple
* USED BY: msolve, inverse, ldioph
**/
template< class Int >
Triple< Int > egcd( Int a, Int b )
{
    if( !b ) return Triple< Int >( a, Int( 1 ), Int( 0 ) );
    Triple< Int > q = egcd( b, a % b );
    return Triple< Int >( q.d, q.y, q.x - a / b * q.y );
}

//TEOREMA CHINO
Chino(a1, a2, m1, m2){
 <x, y> egcd(m1, m2)
 return (m1 * x * a2 + m2 * y * a1) % m1 * m2

/**********************************
* Modular Linear Equation Solver *
**********************************
* Given a, b and n, solves the equation ax = b (mod n)
* for x. Returns the vector of solutions, all smaller
* than n and sorted in increasing order. The vector is
* empty if there are no solutions.
* #include <vector>
* REQUIRES: struct Triple, egcd
**/
template< class Int >
vector< Int > msolve( Int a, Int b, Int n )
{
    if( n < 0 ) n = -n;
    Triple< Int > t = egcd( a, n );
    vector< Int > r;
    if( b % t.d ) return r;
    Int x = ( b / t.d * t.x ) % n;
    if( x < Int( 0 ) ) x += n;
    for( Int i = 0; i < t.d; i++ )
      r.push_back( ( x + i * n / t.d ) % n );
    return r;
}


* Linear Diophantine Equation Solver
* Solves integer equations of the form ax + by = c
* for integers x and y. Returns a triple containing
* the answer (in .x and .y) and a flag (in .d).
* If the returned flag is zero, then there are no
* solutions. Otherwise, there is an infinite number
* of solutions of the form
*          x = t.x + k * b / t.d,
*          y = t.y - k * a / t.d;
* where t is the returned triple, and k is any
* integer.
* REQUIRES: struct Triple, egcd
**/
template< class Int >
Triple< Int > ldioph( Int a, Int b, Int c )
{
    Triple< Int > t = egcd( a, b );
    if( c % t.d ) return Triple< Int >( 0, 0, 0 );
    t.x *= c / t.d; t.y *= c / t.d;
    return t;
}

/**********************
```

11. DINIC

```
const int maxnode=20000+5;
const int maxedge=1000000+5;
const int oo=1000000000;

int node,src,dest,nedge;
int head[maxnode],point[maxedge],
next[maxedge],flow[maxedge],capa[maxedge];
int dist[maxnode],Q[maxnode],work[maxnode];

//inicializa el network, con _node igual a numero de nodos, _src como fuente y
como _dest como sink
void init(int _node,int _src,int _dest)
{
 node=_node;
 src=_src-1;
 dest=_dest-1;
 for (int i=0;i<node;i++) head[i]=-1;
 nedge=0;
}
//anhade la arista de u a v con capacidad c1 y la arista de v a u con capacidad c2
void addedge(int u,int v,int c1,int c2)
{
 point[nedge]=v,capa[nedge]=c1,flow[nedge]=0,next[nedge]=head[u],head[u]=(ned
ge++);
 point[nedge]=u,capa[nedge]=c2,flow[nedge]=0,next[nedge]=head[v],head[v]=(ned
ge++);
}
//bfs de dinic
bool dinic_bfs()
{
 memset(dist,255,sizeof(dist));
 dist[src]=0;
 int sizeQ=0;
 Q[sizeQ++]=src;
 for (int cl=0;cl<sizeQ;cl++)
    for (int k=Q[cl],i=head[k];i>=0;i=next[i])
     if (flow[i]<capa[i] && dist[point[i]]<0)
      {
       dist[point[i]]=dist[k]+1;
       Q[sizeQ++]=point[i];
      }
 return dist[dest]>=0;
}
//dfs de dinic
int dinic_dfs(int x,int exp)
{
 if (x==dest) return exp;
 for (int &i=work[x];i>=0;i=next[i])
 {
    int v=point[i],tmp;
    if (flow[i]<capa[i] && dist[v]==dist[x]+1 && (tmp=dinic_dfs(v,min(exp,capa[i]-
flow[i])))>0)
    {
     flow[i]+=tmp;
     flow[i^1]-=tmp;
     return tmp;
    }
 }
 return 0;
}
//flujo
int dinic_flow()
{
 int result=0;
 while (dinic_bfs())
 {
    for (int i=0;i<node;i++) work[i]=head[i];
    while (1)
    {
     int delta=dinic_dfs(src,oo);
     if (delta==0) break;
     result+=delta;
    }
 }
 return result;
}

int main(){

    init(node,src,dest);
    // addedge(u,v,c1,c2);
    int flow=dinic_flow();
    return flow==totalC && flow==totalD;
}
```

12. DADO

```
struct dice{
            //top, front, left, ri, ba, bo
            int is[6];
            void rollX(){
                        roll(0, 1, 5, 4);
}
void rollY(){
                        roll(0, 3, 5, 2);
}
void rollX(){
                        roll(1, 3, 4, 2);
}
void roll(int a, int b, int c, int d) {
int t = is[d];
is[d] = is[c];
            is[c] = is[b];
is[b] = is[a];
is[a] = t;

}


}
```

13. JOSEPHUS

```
#pragma comment(linker,"/STACK:256000000")

using namespace std;

long long joseph (long long n,long long k) {
    if (n==1LL) return 0LL;
    if (k==1LL) return n-1LL;
    if (k>n) return (joseph(n-1LL,k)+k)%n;
    long long cnt=n/k;
    long long res=joseph(n-cnt,k);
    res-=n%k;
    if (res<0LL) res+=n;
    else res+=res/(k-1LL);
    return res;
}
```

14. MINIMUM SPANNING CIRCLE

```
typedef pair<point, double> circle;


bool in_circle(circle C, point p){
  return cmp((p - C.first).abs(), C.second) <= 0;
}
point circumcenter(point p, point q, point r) {
  point a = p - r, b = q - r, c = point(a * (p + r) / 2, b * (q + r) / 2);
  return point(c % point(a.y, b.y), point(a.x, b.x) % c) / (a % b);
}
point T[100001];
circle spanning_circle(int n) {
  random_shuffle(T, T + n);
  circle C(point(), -INFINITY);
  for (int i = 0; i < n; i++) if (!in_circle(C, T[i])) {
   C = circle(T[i], 0);
            for (int j = 0; j < i; j++) if (!in_circle(C, T[j])) {
            C = circle((T[i] + T[j]) / 2, (T[i] - T[j]).abs() / 2);
              for (int k = 0; k < j; k++) if (!in_circle(C, T[k])) {
                point o = circumcenter(T[i], T[j], T[k]);
                C = circle(o, (o - T[k]).abs());
}
}
  }
  return C;
}
```

15. bridges

```
#define MAX 10000
vector<int> adj[MAX];
vector<pair<int,int> > bridges;
int times[MAX];
int timer;

int dfs(int u, int par){
    int low = times[u] = timer++;
    for(int i = 0; i < adj[u].size(); i++){
            int v = adj[u][i];
            if(times[v] == -1){
                    int lowv = dfs(v, u);
                    if(lowv > times[u])
                            bridges.push_back(make_pair(u, v));
                    else low = min(low, lowv);
            }else if(v != par) low = min(low, times[v]);
    }
    return low;
}
```

16. ARTICULATIONS

```
#define MAX 200
int times[MAX];
int timer;
vector<int> adj[MAX];
set<int> art;

int dfs(int u, int par){
    int low = times[u] = timer++;
    int child = 0;
    for(int i = 0; i < adj[u].size(); i++){
            int v = adj[u][i];
            if(times[v] == -1){
                    child++;
                    int lowv = dfs(v, u);
                    if(lowv >= times[u]) art.insert(u);
                    else low = min(low, lowv);
            }else if(v != par) low = min(low, times[v]);
    }
    if(u ==1 && child <= 1) art.erase(u);
    return low;
}
```

17. CATALAN

1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} \qquad \text{for } n \geq 0.$$

$$C_n = \binom{2n}{n} - \binom{2n}{n+1} \quad \text{for } n \geq 0,$$

1. Cn is the number of Dyck words of length 2n. A Dyck word is a string consisting of n X's and n Y's such that
no initial segment of the string has more Y's than X's
2. Cn counts the number of expressions containing n pairs of parentheses which are correctly matched
3. Cn is the number of di
erent ways n + 1 factors can be completely parenthesized (or the number of ways of associating n applications of a binary operator)
4. Cn is the number of full binary trees with n + 1 leaves.
5. Cn is the number of stack-sortable permutations of 1; :::; n.
6. Cn is the number of permutations of 1; :::; n that avoid the pattern 123 (or any of the other patterns of length
3); that is, the number of permutations with no three-term increasing subsequence

18. TEOREMA DE LUCAS

For non-negative integers *m* and *n* and a prime *p*, the following congruence relation holds:

$$\binom{m}{n} \equiv \prod_{i=0}^{k} \binom{m_i}{n_i} \pmod{p},$$

where $m = m_k p^k + m_{k-1} p^{k-1} + \cdots + m_1 p + m_0,$

and $n = n_k p^k + n_{k-1} p^{k-1} + \cdots + n_1 p + n_0$

are the base *p* expansions of *m* and *n* respectively.

//triangulation
From the Guide: Look at the edge v0 vn−1 , in any triangulation it is contained in exactly one triangle.
Guess what will be the other point of that triangle. Let m[a][b] (a < b) be the value of the best triangulation
of the (part of) polygon va va+1 · · · vb va .
This time, the solution of the longer segments depends on the shorter segments.

19. Best Triangulation

```
m[..][..] = 0;
for(len=3; len<n; len++)
for(a=0; a+len<n; a++){
b=a+len; m[a][b]=1e+10;
for(c=a+1; c<b; c++){
double t=m[a][c]+m[c][b];
if(c>a+1) t+=length(a to c);
if(c<b-1) t+=length(c to b);
m[a][b] <?= t;
}
}
```

20. triangulos 3D

```
bool insideT(punto P,punto A,punto B,punto C,bool tipo){

        punto u1 = B-A, u2 = C-A, N = u1%u2;
if( abs( (P-A)*N )>eps ) return 0;
double c1 = (P-A)*u1, c2 = (P-A)*u2;
double u11 = u1*u1, u12 = u1*u2, u22 = u2*u2;
double delta = u11*u22 - u12*u12;
double s1 = (c1*u22 - c2*u12)/delta;
double s2 = (c2*u11 - c1*u12)/delta;
if( tipo==0 ){
                if( s1+s2>1+eps || s1<-eps || s2<-eps ) return 0;
                return 1;
        }else{
                if( s1>1+eps || s2>1+eps || s1<-eps || s2<-eps ) return 0;
                return 1;
        }
}
```

```
bool corta(punto P,punto Q,punto A,punto B,punto C){
	punto u1 = B-A, u2 = C-A, v = Q-P, N = u1%u2;
	if( abs(v*N)>eps ){
		double den = v*N;
		double t = (A-P)*N/den;
		double s1 = (P-A)*(u2%v)/den, s2 = (P-A)*(v%u1)/den;
		if( t>1+eps || t<-eps || s1<-eps || s2<-eps || s1+s2>1+eps ) return 0;

		return 1;
	}else{
		if( insideT(P-A,A,B,A-v,1) ) return 1;
		if( insideT(P-A,A,C,A-v,1) ) return 1;
		if( insideT(P-B,B,C,B-v,1) ) return 1;
		return 0;
	}
}

int T;
punto A,B,C,P,Q,R;
bool secruzan(){
if( insideT(P,A,B,C,0) || insideT(Q,A,B,C,0) || insideT(R,A,B,C,0) ) return 1;
if( insideT(A,P,Q,R,0) || insideT(B,P,Q,R,0) || insideT(C,P,Q,R,0) ) return 1;
if( corta(P,Q,A,B,C) || corta(P,R,A,B,C) || corta(Q,R,A,B,C) ) return 1;
if( corta(A,B,P,Q,R) || corta(A,C,P,Q,R) || corta(B,C,P,Q,R) ) return 1;
return 0;
}


int main()
{
	cin >> T;
	punto A,B,C,a,b,c;
	f(cases,0,T){
		printf( "Case #%d: ", cases+1);
		A.read(); B.read(); C.read();
		a.read(); b.read(); c.read();
		solve(A,B,C,a,b,c).write();
	}
}
```

21. Numeros de Bell

1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975, ...

The Bell numbers satisfy this recursion formula:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k.$$

And they satisfy "Touchard's congruence": If *p* is any prime number then

$$B_{p+n} \equiv B_n + B_{n+1} \pmod{p}.$$

$$B_{p^m+n} \equiv mB_n + B_{n+1} \pmod{p}.$$

$$B_n = \sum_{k=0}^{n} \left\{ {n \atop k} \right\} = \sum_{k=0}^{n} \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n$$

$$B_{n+m} = \sum_{k=0}^{n} \sum_{j=0}^{m} \left\{ {m \atop j} \right\} \binom{n}{k} j^{n-k} B_k.$$

22. Stirling numbers of the second kind

$$\left\{ {n \atop k} \right\} = \frac{1}{k!} \sum_{j=0}^{k} (-1)^j \binom{k}{j} (k-j)^n.$$

$$\left\{ {n+1 \atop k} \right\} = k \left\{ {n \atop k} \right\} + \left\{ {n \atop k-1} \right\}$$

for *k* > 0 with initial conditions

$$\left\{ {0 \atop 0} \right\} = 1 \quad \text{and} \quad \left\{ {n \atop 0} \right\} = \left\{ {0 \atop n} \right\} = 0$$

22. Stirling numbers of the first kind

The unsigned Stirling numbers of the first kind can be calculated by the recurrence relation

$$\left[{n+1 \atop k}\right] = n\left[{n \atop k}\right] + \left[{n \atop k-1}\right]$$

for $k > 0$, with the initial conditions

$$\left[{0 \atop 0}\right] = 1 \quad \text{and} \quad \left[{n \atop 0}\right] = \left[{0 \atop n}\right] = 0$$

23. SUFFIX ARRAY

```
int n,t; //n es el tamanho de la cadena
int p[MAXN],r[MAXN],h[MAXN];
string s;

bool comp(int i,int j){ return p[i+t] < p[j+t]; }

void suff_arr(){
  int bc[256];
        f(i,0,256) bc[i] = 0;
        f(i,0,n) ++bc[s[i]];
        f(i,1,256) bc[i] += bc[i-1];
        f(i,0,n) r[--bc[s[i]]] = i;
        f(i,0,n) p[i] = bc[s[i]];
        for(t=1; t<n; t *= 2 ){
                for(int i = 0, j = 1; j<n; i = j++ ){
                        while( j<n && p[r[j]] == p[r[i]] ) j++;
                        if( j-i==1 ) continue;
                        int *ini = r+i, *fin = r+j;
                        sort(ini, fin, comp);
                        int pri = p[*ini+t], num = i, pk;
                        for(; ini<fin; p[*ini++] = num ){
                                if(((pk=p[*ini+t]) == pri) || (i<=pri && pk<j)
){}
                                else pri = pk, num = ini-r;
                        }
                }
        }
}
void lcp() {
  int tam = 0, i, j;
  for(i = 0; i < n; i++) if (p[i] > 0) {
        j = r[p[i] - 1];
        while(s[i + tam] == s[j + tam]) ++tam;
        h[p[i] - 1] = tam;
        if (tam > 0) --tam;
  }
  h[n - 1] = 0;
}
```

24. MINIMUM LEX ROT

```
int n = s.size();
s = s + s;
  int mini = 0, p = 1, l = 0;
  while(p < n && mini + l + 1 < n)
    if(s[mini + l] == s[p + l])
      l++;
    else if(s[mini + l] < s[p + l]){
      p = p + l + 1;
      l = 0;
    }else if(s[mini + l] > s[p + l]){
      mini = max(mini + l + 1, p);
      p = mini + 1;
      l = 0;
    }
  s = s.substr(mini, n);
```

25. Z-function

```
void zfunction(char s[], int z[], int sz){
   for(int i = 1, l = 0, r = 0; i < sz; ++i){
        z[i] = 0;
    if (i <= r)
        z[i] = min(z[i - l], r - i + 1);
    while (i + z[i] < sz && s[z[i]] == s[i + z[i]])
        ++z[i];
    if (i + z[i] - 1 > r){
            l = i;
            r = i + z[i] - 1;
      }
    }
    z[0] = sz;
}
```

26. non recursive biconnected components

```
#define N 200001

const int size = 100005;
vector<set<int> > comps;
vector<int> E[N];
pii s[N]; int S = 0; int times[N], timer;

int indice[N], estado[N], par[N], low[N], child[N];
bool es[N];
int ss[N];
int q[N], qb;

void dfs2(int uu){
   int szs = 0; ss[szs++] = uu; timer = 0;
   while(szs){
            int u = ss[szs - 1];
            if(times[u] == -1) low[u] = times[u] = timer++;

            if(indice[u] == E[u].size()) szs--;
            else if(estado[u] == 0){
            int v = E[u][indice[u]];
            if(times[v] != -1){
            if(v != par[u] && times[v] <= times[u]){
            low[u] = min(low[u], times[v]), s[S++] = mp(u, v);
            }
            indice[u]++;
            }else{
            child[u]++;
            par[v] = u;
            s[S++] = mp(u, v);
            estado[u]++;
            ss[szs++] = v;
            }
            }else{
            int v = E[u][indice[u]++];
            if(low[v] >= times[u]){
                        es[u] = true;
            int sz = 0;
            int tam = 0; pii x; pii e(u, v);
            set<int> CC;
            do{
            x = s[--S];
            CC.insert(x.fst); CC.insert(x.snd);
            }while(x != e);
            comps.pb(CC);
            }else low[u] = min(low[u], low[v]);
            estado[u] = 0;
            }
   }
}


f(i, 0, n) times[i] = -1, es[i] = false, estado[i] = 0, indice[i] = 0, par[i] = -1, child[i] =
0;
             comps.clear();
             timer = 0;
            dfs2(0);
            if(child[0] < 2) es[0] = false;
```

27. STOER WAGNER

```
int g[100][100], v[100], w[100], na[100];
bool a[100];

int mincut(int n){
    f(i, 0, n) v[i] = i;
    int best = inf;
    if(n <= 1) return 0;
    while(n > 1){
             a[v[0]] = true;
             f(i, 1, n){
                a[v[i]] = false;
                na[i - 1] = i;
                w[i] = g[v[0]][v[i]];
            }
            int prev = v[0];
            f(i, 1, n){
                int zj = -1;
                f(j, 1 ,n)
                    if(!a[v[j]] && (zj < 0 || w[j] > w[zj]))
                        zj = j;
                a[v[zj]] = true;
                if(i == n - 1){
                    best = min(best, w[zj]);
                    f(j, 0, n)
                        g[v[j]][prev] = g[prev][v[j]] += g[v[zj]][v[j]];
                    v[zj] = v[--n];
                    break;
                }
                prev = v[zj];
                f(j, 1, n) if(!a[v[j]])
                    w[j] += g[v[zj]][v[j]];
            }
    }
    if(best == inf) while(1);
    return best;
}
```

28. CRIBA LINEAL

```
int A[MAX], P[MAX/10], pc = 0;

void criba(){
        f(i,2,MAX){
                if(!A[i]) P[++pc] = i, A[i] = pc;
                for(int j=1; j<=A[i] && i*P[j]<MAX; j++) A[i*P[j]] = j;
        }
}
```

29. ROMAN TO INT

```
string roman(string &s){
    int cont=0,pos=0;
    string x="";

    while(s[pos]=='M'){
        pos++;
        cont++;
    }

    x+='0'+cont;

    if(pos+1<s.size() && s[pos]=='C' && s[pos+1]=='M'){
        pos+=2;
        x+='9';
    }else if(pos<s.size() && s[pos]=='D'){
        pos++;
        cont=5;
        while(pos<s.size() && s[pos]=='C'){
            pos++;
            cont++;
        }
        x+='0'+cont;
    }else if(pos+1<s.size() && s[pos]=='C' && s[pos+1]=='D'){
        pos+=2;
        x+='4';
    }else{
        cont=0;
        while(pos<s.size() && s[pos]=='C'){
            pos++;
            cont++;
        }
        x+='0'+cont;
    }

    if(pos+1<s.size() && s[pos]=='X' && s[pos+1]=='C'){
        pos+=2;
        x+='9';
    }else if(pos<s.size() && s[pos]=='L'){
        pos++;
        cont=5;
        while(pos<s.size() && s[pos]=='X'){
            pos++;
            cont++;
        }
        x+='0'+cont;
    }else if(pos+1<s.size() && s[pos]=='X' && s[pos+1]=='L'){
        pos+=2;
        x+='4';
    }else{
        cont=0;
        while(pos<s.size() && s[pos]=='X'){
            pos++;
            cont++;
        }
        x+='0'+cont;
    }

    if(pos+1<s.size() && s[pos]=='I' && s[pos+1]=='X'){
        pos+=2;
        x+='9';
    }else if(pos<s.size() && s[pos]=='V'){
        pos++;
        cont=5;
        while(pos<s.size() && s[pos]=='I'){
            pos++;
            cont++;
        }
        x+='0'+cont;
    }else if(pos+1<s.size() && s[pos]=='I' && s[pos+1]=='V'){
        pos+=2;
        x+='4';
    }else{
        cont=0;
        while(pos<s.size() && s[pos]=='I'){
            pos++;
            cont++;
        }
        x+='0'+cont;
    }

    while(x.size()>=1 && x[0]=='0') x.erase(0,1);
    return x;
}
```

30. INT TO ROMAN

```
string conv(int a){
    string romanos="";

    while(a>=1000) romanos+="M", a-=1000;
    if(a>=900) romanos+="CM", a-=900;
    if(a>=500) romanos+="D",  a-=500;
    if(a>=400) romanos+="CD", a-=400;

    while(a>=100) romanos+="C", a-=100;
    if(a>=90)  romanos+="XC", a-=90;
    if(a>=50)  romanos+="L",  a-=50;
    if(a>=40)  romanos+="XL", a-=40;

    while(a>=10) romanos+="X", a-=10;
    if(a==9) romanos+="IX", a-=9;
    if(a>=5) romanos+="V", a-=5;
    if(a==4) romanos+="IV", a-=4;

    while(a>=1) romanos+="I", a--;

    return romanos;
}
```

31. HL Decomposition

```
#define N 100000
int n, par[N], q[N], h[N], head[N], tam[N], color[N];
vector<int> E[N];
struct segment{
  vector<int> has, indice;
  int t, m;

  void update(int pos, int ind, int val, int u, int le, int ri){
    if(le == ri) has[u] = val, indice[u] = ind;
    else{
      int mid = (le + ri) / 2;
      if(pos <= mid) update(pos, ind, val, u * 2 + 1, le, mid);
      else update(pos, ind, val, u * 2 + 2, mid + 1, ri);
      has[u] = has[u * 2 + 1] || has[u * 2 + 2];
    }
  }
  void update(int alt, int ind, int val){
    update(alt - t, ind, val, 0, 0, m - 1);
  }
  int get(int to, int u, int le, int ri){
    if(!has[u] || le > to) return -1;

    if(le == ri) return indice[u];
    int mid = (le + ri) / 2;
    int aux = get(to, u * 2 + 1, le, mid);
    if(aux != -1) return aux;
    return get(to, u * 2 + 2, mid + 1, ri);
  }
  int get(int alt){
    return get(alt - t, 0, 0, m - 1);
  }
  void construct(int a, int b){
    m = h[b] - h[a] + 1;
    has.resize(4 * m);
    indice = has;
    t = h[a];
    for(int i = m - 1, u = b; i >= 0; i--, u = par[u])
      update(i, u + 1, 0, 0, 0, m - 1);
  }
}S[N];

void make(){
  int qb = 0;
  q[qb++] = 0; h[0] = 0;
  f(i, 0, qb){
    int u = q[i];
    FOR(v, E[u]) if(h[*v] == -1)
      par[*v] = u, q[qb++] = *v, h[*v] = h[u] + 1;
  }
  for(int i = n - 1; i >= 0; i--){
    int u = q[i];
    tam[u] = 1;
    FOR(v, E[u]) if(par[*v] == u) tam[u] += tam[*v];
  }
  f(i, 0, n){
    int a = q[i];
    if(head[a] != -1) continue;
    int u = a;
    while(true){
      head[u] = a;
      int next = -1;
      FOR(v, E[u]) if(par[*v] == u && (next == -1 || tam[*v] > tam[next])) next = *v;
      if(next == -1) break;
      u = next;
    }
    S[a].construct(a, u);
  }
}
```

32. Centros de masa

lamina	$\bar{y}$
circular sector	$\frac{4R\sin\left(\frac{1}{2}\theta\right)}{3\theta}$
circular segment	$\frac{4R\sin^3\left(\frac{1}{2}\theta\right)}{3(\theta-\sin\theta)}$
isosceles triangle	$\frac{1}{3}h$
parabolic segment	$\frac{2}{5}h$
semicircle	$\frac{4R}{3\pi}$

33. PAPERWEIGHT

```
struct punto{
      double x,y,z;
```

```
	punto (){}
	punto (double a, double b, double c): x(a), y(b),
z(c){}
	punto operator+ (punto p){ return punto (x+p.x, y+p.y,
z+p.z); }
	punto operator- (punto p){ return punto (x-p.x, y-p.y,
z-p.z); }
	punto operator% (punto p){
		return punto (y*p.z - z*p.y, z*p.x - x*p.z,
x*p.y - y*p.x);
	}
	punto operator* (double t){ return punto (x*t, y*t,
z*t); }
	punto operator/ (double t){ return punto (x/t, y/t,
z/t); }
	double operator* (punto p){ return x*p.x + y*p.y +
z*p.z; }
	double norma (){ return sqrt (cua(x) + cua(y) +
cua(z)); }
	void read (){ scanf ("%lf%lf%lf", &x, &y, &z); }
};
struct lado{
	punto A,B;
	lado (){}
	lado (punto X, punto Y): A(X), B(Y) {}
};
struct cara{
	punto A,B,C;
	cara (){}
	cara (punto X, punto Y, punto Z): A(X), B(Y), C(Z) {}
};

double area (punto A, punto B, punto C){
	return ((B-A)%(C-A)).norma();
}

double d (punto P, cara face){
	punto A = face.A, B = face.B, C = face.C;
	punto N = (B-A) % (C-A);
	N = N/N.norma();
	return (P-A)*N;
}
```

```
punto H[10]; int sz;
bool operator< (punto A, punto B){
	if (A.x != B.x) return A.x < B.x;
	if (A.y != B.y) return A.y < B.y;
	return A.z < B.z;
}
void hull (vector<punto> v){
	punto N = (v[1]-v[0]) % (v[2]-v[0]);
	sz = 0;
	sort (all(v));

	f(i,0,v.size()) {
		while (sz>=2 && N*((H[sz-1] - v[i]) % (H[sz-2] -
v[i])) > -eps) sz--;
		H[sz++] = v[i];
	}
	int t = sz+1;
	fd(i,v.size()-1,0){
		while (sz>=t && N*((H[sz-1] - v[i]) % (H[sz-2] -
v[i])) > -eps) sz--;
		H[sz++] = v[i];
	}
	sz--;
}
bool dentro (punto P, punto A, punto B, punto C){
	double res = area (P,A,B) + area(P,B,C) + area(P,C,A) -
area (A,B,C);
	return abs(res) < eps;
}
bool dentro (punto P){
	double res = 0;
	f(i,1,sz-1){
		if (dentro (P, H[0], H[i], H[i+1])) return 1;
	}
	return 0;
}
double ud (punto P, punto A, punto B){
	return area (P,A,B) / (A-B).norma();
}
```

34. TETRAHEDRON

Base plane area	$A_0 = \frac{\sqrt{3}}{4}a^2$
Surface area[2]	$A = 4A_0 = \sqrt{3}a^2$
Height[3]	$H = \frac{\sqrt{6}}{3}a$
Volume[2]	$V = \frac{1}{3}A_0 h = \frac{\sqrt{2}}{12}a^3$
Angle between an edge and a face	$\arccos\left(\frac{1}{\sqrt{3}}\right) = \arctan(\sqrt{2})$ (approx. 54.7356°)
Angle between two faces[2]	$\arccos\left(\frac{1}{3}\right) = \arctan(2\sqrt{2})$ (approx. 70.5288°)
Angle between the segments joining the center and the vertices,[4] also known as the "tetrahedral angle"	$\arccos\left(\frac{-1}{3}\right) = 2\arctan(\sqrt{2})$ (approx. 109.4712°)
Solid angle at a vertex subtended by a face	$\arccos\left(\frac{23}{27}\right)$ (approx. 0.55129 steradians)
Radius of circumsphere[2]	$R = \sqrt{\frac{3}{8}}a$
Radius of insphere that is tangent to faces[2]	$r = \frac{1}{3}R = \frac{a}{\sqrt{24}}$
Radius of midsphere that is tangent to edges[2]	$r_M = \sqrt{rR} = \frac{a}{\sqrt{8}}$
Radius of exspheres	$r_E = \frac{a}{\sqrt{6}}$
Distance to exsphere center from a vertex	$\sqrt{\frac{3}{2}}a$

35. BCC de chen

```
int n,m,fin;
int orig[MAX], dest[MAX], pila[MAX], top = 0;
vint E[MAX];
int low[MAX], dfsn[MAX], part[MAX], timer;
int ponte[MAX], bicomp[MAX], nbicomp;

int dfsbcc (int u, int p = -1){
	low[u] = dfsn[u] = ++timer;
	int ch = 0;
	FOR(it, E[u]){
		int e = *it, v = VIZ (e, u);
		if (dfsn[v] == 0){
			pila[top++] = e;
			dfsbcc (v, u);
			low[u] = min (low[u], low[v]);
			ch++;
			if (low[v] >= dfsn[u]){
				part[u] = 1;
				nbicomp++;
				do{
					fin = pila[--top];
					bicomp[fin] = nbicomp;
				}while (fin != e);
			}
			if (low[v] == dfsn[v]) ponte[e] = 1;
		}else if (v!=p && dfsn[v] < dfsn[u]){
			pila[top++] = e;
			low[u] = min (low[u], dfsn[v]);
		}
	}
	return ch;
}
void bcc (){
	f(i,0,n) part[i] = dfsn[i] = 0;
	f(i,0,m) ponte[i] = 0;
	nbicomp = timer = 0;
	f(i,0,n) if (dfsn[i] == 0) part[i] = dfsbcc (i) >= 2;
}
```

36. NKMARS

```
struct node{
   int on, area;
   node (int x = 0, int y = 0) : on(x), area(y){}
};
int tam;
struct segmentTree{
   node seg[1<<17];
   int off;
   segmentTree(){
   };
   int area(){ return seg[0].area;}
   void add (int a, int b, int x = 0, int le = 0, int ri = tam){
      if (b<=le || ri<=a) return;
      if (a<=le && ri<=b){ seg[x].on++; seg[x].area = ri - le; return; }
      int me = (le + ri) / 2;
      add (a, b, L(x), le, me);
      add (a, b, R(x), me, ri);
      if(seg[x].on==0) seg[x].area = seg[L(x)].area + seg[R(x)].area;
      else seg[x].area = ri-le;
   }
   void take (int a, int b, int x = 0, int le = 0, int ri = tam){
      if (b<=le || ri<=a) return;
      if (a<=le && ri<=b){
         seg[x].on--;
         if (seg[x].on==0) seg[x].area = seg[L(x)].area + seg[R(x)].area;
         return;
      }
      int me = (le + ri) / 2;
      take (a, b, L(x), le, me);
      take (a, b, R(x), me, ri);
      if(seg[x].on==0) seg[x].area = seg[L(x)].area + seg[R(x)].area;
      else seg[x].area = ri-le;
   }
};
vector<pii> abre[MAX], cierra[MAX];
int main()
{
   int n;
   cin >> n;
   int a1,b1,a2,b2;
   f(i,0,n){
      scanf ("%d%d%d%d", &a1, &b1, &a2, &b2);
      abre[a1].pb (pii(b1,b2));
      cierra[a2].pb (pii(b1,b2));
   }
   segmentTree st;
```

```
    tam = 1<<16;
    int res = 0;
    f(i,0,MAX){
        cout<<i;
        f(j,0,abre[i].size()) st.add (abre[i][j].fst, abre[i][j].snd);//cout <<1111<<"
"<<i<<endl;
        f(j,0,cierra[i].size()) st.take (cierra[i][j].fst, cierra[i][j].snd);// cout <<2222
<<" "<<i<<endl;
        res += st.area();
    }
    cout << res << endl;
```

37. CIRCLE FROM 3 POINTS

```
point center(double x1, double y1, double x2, double y2, double x3, double y3){
   point p;
   double A1 = x1-x2;
   double A2 = x2-x3;
   double B1 = y1-y2;
   double B2 = y2-y3;
   double C1 = (A1*(x1+x2) + B1*(y1+y2))/2;
   double C2 = (A2*(x2+x3) + B2*(y2+y3))/2;
   double d = A1*B2-A2*B1;
   if(fabs(d)<1e-7)return p;
   double y = (A1*C2-A2*C1)/d;
   double x = -(B1*C2-B2*C1)/d;
   p.x = x, p.y = y;
   return p;
}
```

38. CIRCLE LINE INTERSECTION

It is assumed that the circule is located at (0,0), r is the radius of the circule and a, b, c are the coefficients of the line. $ax + by + c = 0$

```
double r, a, b, c; // input

double x0 = -a*c/(a*a+b*b),  y0 = -b*c/(a*a+b*b);
if (c*c > r*r*(a*a+b*b)+EPS)
    puts ("no points");
else if (abs (c*c - r*r*(a*a+b*b)) < EPS) {
    puts ("1 point");
    cout << x0 << ' ' << y0 << '\n';
}
else {
    double d = r*r - c*c/(a*a+b*b);
    double mult = sqrt (d / (a*a+b*b));
    double ax,ay,bx,by;
    ax = x0 + b * mult;
    bx = x0 - b * mult;
    ay = y0 - a * mult;
    by = y0 + a * mult;
    puts ("2 points");
    cout << ax << ' ' << ay << '\n' << bx << ' ' << by << '\n';
}
```

39. PAIRWISE SUM RECONSTRUCTION

```
bool pairsums( int *ans, multiset< int > &seq )
{
    int N = seq.size();
    if( N < 3 ) return false;
    __typeof( seq.end() ) it = seq.begin();
    int a = *it++, b = *it++, i = 2;

    for( ; i * ( i - 1 ) < 2 * N && it != seq.end(); i++, ++it )
    {
        // assume seq[i] = ans[1] + ans[2]
        ans[0] = a + b - *it;
        if( ans[0] & 1 ) continue;
        ans[0] >>= 1;

        // try ans[0] as a possible least element
        multiset< int > seq2 = seq;
        int j = 1;
        while( seq2.size() )
        {
            ans[j] = *seq2.begin() - ans[0];
            for( int k = 0; k < j; k++ )
            {
                __typeof( seq2.end() ) jt = seq2.find( ans[k] + ans[j] );
                if( jt == seq2.end() ) goto hell;
                seq2.erase( jt );
            }
            j++;
        }
        hell:;
        if( j * ( j - 1 ) < 2 * N ) continue;
        return true;
    }
    return false;
}
```

40. Digitos mas significativos

Como imprimir los N digitos mas significativos?... los 5 es asi: (notar q no es tan trivial como suena)
print n/d rounded to 5 sig digits

```
void PrintNumber(double x){
  double y;int p, left, moder;
  if(x>=1){
    left=0; while(x>=1){ x /= 10; left++; }
    // move x left 5 digits
    x *= 100000; p=int (x);
    y = x-p; if(y >= 0.5 - 0.000001) p++;
    //p has the 5 sig digits, so print
    if(left>=5){
      cout<<p; for(int i=5; i<left;i++) cout<<0;
    }
    else { // need a decimal
      moder=10000;
      for(int i=0;i<left;i++){
        cout<< p/moder; p %= moder; moder /=10;
      }
      cout<<'.';
      for(int i=left;i<5;i++){
        cout<<p/moder; p %= moder; moder /= 10;
      }
    } //else
  }
  else { // x<1
    cout<<"0.";
    // find how many 0's after decimal
    left=-1; while(x<1) {x *= 10; left++; }
    x *= 10000; p = int (x);
    y = x-p; if(y>=0.5) p++;  //p now holds 5 sig digits
    // print 0's
    for(int i=0;i<left;i++) cout<<0; cout<<p;
  }
```

41. TREAP

```
ll mod = 1000000000;
struct node{
        node *L, *R;
        int y, x, rev, cnt, k;
        ll sum;
        node(int value, int key){
                L = R = NULL;
                sum = k = value;
      x = key;
                y = rand();
                rev = 0; cnt = 1;
        }
};

void refresh(node* T){
        if(!T) return;
        if(T->rev){
                if(T->L) T->L->rev ^= 1;
                if(T->R) T->R->rev ^= 1;
      swap(T->L, T->R);
                T->rev = 0;
        }
}
void update(node *T){
        if(!T) return;
        T->sum = T->k; T->cnt = 1;
        if(T->L) T->sum += T->L->sum,
                 T->cnt += T->L->cnt;
        if(T->R) T->sum += T->R->sum,
                 T->cnt += T->R->cnt;
}
void split(node *T, int x, node* &L, node* &R){
        if(!T) L = R = NULL;
        else if(T->x > x){
                split(T->L, x, L, T->L);
                R = T;
        }else{
                split(T->R, x, T->R, R);
                L = T;
        }
        update(R); update(L);
}
void split2(node *T, int cnt, node* &L, node* &R){
   refresh(T);
        if(!T) L = R = NULL;
        else if(cnt <= (T->L ? T->L->cnt : 0)){
                split2(T->L, cnt, L, T->L);
                R = T;
        }else{
                split2(T->R, cnt - (T->L ? T->L->cnt : 0)
- 1, T->R, R);
                L = T;
        }
        update(R); update(L);
}

node* merge(node*L, node*R){
   if(!L) return R;
   if(!R) return L;
   refresh(L); refresh(R);
   node *T;
   if(L->y > R->y)
      L->R = merge(L->R, R), T = L;
   else
      R->L = merge(L, R->L), T = R;
   update(T);
   return T;
}

node* add(node *T, node *N){
        if(!T || T->y < N->y){
                node* R; split(T, N->x, N->L, N->R);
                T = N;
        }else if(N->x < T->x)
                T->L = add(T->L, N);
        else
                T->R = add(T->R, N);
        update(T);
   return T;
}
int main(){
        freopen("reverse.in", "r", stdin);
        freopen("reverse.out", "w", stdout);
        ios::sync_with_stdio(false);
        int n, m;
   node *root = NULL;
   cin >> n >> m;
   f(i, 0, n){
      int x; cin >> x;
      root = add(root, new node(x, i));
   }
   while(m--){
      int q, l, r; cin >> q >> l >> r;
      node *A, *B, *C, *D;
      split2(root, r, A, B);
      split2(A, l - 1, C, D);
      if(q == 0)
         cout << D->sum << endl;
      else{
         D->rev ^= 1;
         refresh(D);
      }
      root = merge(merge(C, D), B);
   }
}
```

42. CLOSEST PAIR OF POINTS

```
#define px second
#define py first
typedef pair<long long, long long> pairll;
int n;
pairll pnts [100000];
set<pairll> box;
double best;
int compx(pairll a, pairll b) { return a.px<b.px; }
int main () {
scanf("%d", &n);
for (int i=0;i<n;++i) scanf("%lld %lld", &pnts[i].px, &pnts[i].py);
sort(pnts, pnts+n, compx);
best = 1500000000; // INF
box.insert(pnts[0]);
int left = 0;
for (int i=1;i<n;++i) {
while (left<i && pnts[i].px-pnts[left].px > best) box.erase(pnts[left++]);
for (typeof(box.begin()) it=box.lower_bound(make_pair(pnts[i].py-best, pnts[i].px-
best));
it!=box.end() && pnts[i].py+best>=it->py; it++)
best = min(best, sqrt(pow(pnts[i].py - it->py, 2.0)+pow(pnts[i].px - it->px, 2.0)));
box.insert(pnts[i]);
printf("%.2f\n", best);
}
return 0;
}
```

43. PICK

$$A = i + \frac{b}{2} - 1.$$

44. KIRCHHOFF para bipartitos

If *G* is the complete bipartite graph with vertices 1 to n_1 in one partition and vertices $n_1 + 1$ to n in the other partition, the number of labeled spanning trees of *G* is $n_1^{n_2-1} n_2^{n_1-1}$, where $n_2 = n - n_1$.

46. LIS2

```
set<par> A[MAX];
set<par>::iterator it,iit;
int n, x[MAX], y[MAX];
bool ok(int k,int i){
    it = A[k].lower_bound (par (x[i], -oo));
    if( it==A[k].begin() ) return 0;
    it--;
    return y[i] > it->second;
}

void update(int k, int i){
    it = A[k].lower_bound (par (x[i], y[i]));
    while( it!=A[k].end() && y[i] <= it->second ){
        iit = it;
        iit++;
        A[k].erase(it);
        it = iit;
    }
    if( (it==A[k].end() || x[i]!=it->first) && (it==A[k].begin() || y[i]!=(--it)->second)
)
            A[k].insert( par (x[i], y[i]));
}

int main()
{
    cin >> n;
    f(i,0,n) scanf("%d%d", x+i, y+i);// y[i] = -y[i];
    int tam = 0;
    f(i,0,n){
        int lo = 0, hi = tam;
        while( lo < hi ){
            int me = (lo+hi)/2;
            if( ok(me, i) ) lo = me+1;
            else hi = me;
        }
        if( lo==tam ) tam++;
        update (lo, i);
    }
    cout << tam << endl;
```