1. TEMPLATE 1
2. NOTAS 1
3. ÁLGEBRA 2
3.1 Simplex 2
3.2 Sistemas lineares 2
3.3 Eliminação Gaussiana 3
4. GEOMETRIA 3
4.1 Intersecção de polígonos 4
4.2 Classificação de reta em relação a um círculo 4
4.3 Par de pontos mais próximos – Line scan sweeping 4
4.4 União de retângulos – Line scan sweeping e árvore de intervalos ... 4
4.5 Convex hull 5
4.6 Intersecção de retas e segmentos 5
4.7 Classificação de ponto em relação a um polígono 6
4.8 Centróide 6
4.9 Círculo definido por 3 pontos 6
4.10 Círculo mínimo que engloba uma lista de pontos 6
4.11 Distância na esfera 6
4.12 Geometria 3D 6
5. CLASSES DE NÚMEROS 7
5.1 Inteiros de Precisão Arbitrária 7
5.2 Frações 7
6. PROGRAMAÇÃO DINÂMICA 8
6.1 LIS 1D 8
6.2 LIS 2D 8
6.3 LCS 8
6.4 Maximum Sum 2D 9
6.5 Mochila 0–1 com conflito e mochila inteira 9
6.6 TSP 9
7. JOGOS COMBINATORIAIS 9
7.1 Posições vencedoras/perdedoras (detecção de ciclos) 9
7.2 Nim 9
7.3 Grundy Numbers 9
8. GRAFOS 10
8.1 Caminho Mínimo 10
8.2 AGM 10
8.3 Diâmetro de uma árvore 10
8.4 Componentes fortemente conectados (Aplicações: 2–SAT) 10
8.5 Pontes, pontos de articulacao e componentes biconectados 11
8.6 Ordenação Topológica 11
8.7 Circuito Euleriano 11
8.8 Fluxo máximo (Vertex cut, Cobertura mínima por caminhos em DAG) ... 11
8.9 Fluxo máximo de custo mínimo 12
8.10 Matching máximo em um grafo bipartido valorado – Hungariano 13
8.11 Grafo de restrições de diferenças 14
8.12 Cobertura mínima de vértices em árvores 14
8.13 Permanente de uma matriz (cobertura por ciclos e matching perfeito) 14
8.14 Stable marriage problem 14
8.15 Matching em grafo bipartido 15
8.16 Corte mínimo global 15
8.17 Matching em grafos quaisquer – Edmonds 15
9. TEORIA DOS NÚMEROS 16
9.1 MDC e MMC 16
9.2 Euclides estendido: ax + by == gcd(a, b) 16
9.3 Equações diofantinas lineares: ax + by == c 16
9.4 Inverso multiplicativo: ax == 1 (mod m) 16
9.5 Menor solução não–negativa de ax == b (mod m) 16
9.6 C(n,k) (mod m) 16
9.7 Exponencial modular: b^e (mod m) 16
9.8 Baby–step Giant–step: menor solução para e em b^e == n (mod m) 16
9.9 Legendre symbol: existe x tal que x^2 == a (mod m) 16
9.10 Fatoração em número primos 16
9.11 MDC(x!, y) 16
9.12 Cálculo dos divisores de um inteiro n 17

UFMG – Universidade Federal de Minas Gerais

9.13 Cálculo **do** número de divisores dos inteiros de 1 a n 17
9.14 Crivo de Eratóstenes 17
9.15 Triângulo de Pascal 17
9.16 Teste de primalidade com Miller–Rabin e Pollard–Rho 17
9.17 Totient 17
9.18 a * b (mod m) 18
9.19 Quantidade de números <= n múltiplos de algum elemento **do** vetor v . 18
10. MATRÓIDES 18
10.1 Intersecção de dois matróides (um gráfico e um de partição) 18
11. ESTRUTURAS DE DADOS 19
11.1 LCA 19
11.2 Árvore de segmentos – RMQ 19
11.3 Árvore de intervalos 21
11.4 Fenwick Tree (BIT) 21
12. MISC 21
12.1 Bitwise operations 21
12.2 Contagem de pontos que não são dominados por outros pontos em 3D .. 22
12.3 Soma de medianas de intervalos de tamanho fixo de um vetor 22
12.4 Algoritmo probabilístico 22
12.5 Distância mínima (cavalos) em um tabuleiro de xadrez 22
12.6 Notação polonesa reversa 22
12.7 Josephu's problem 23
12.8 Índice de uma permutação / permutação de um índice 23
13. STRINGS 23
13.1 Minimum Lexicographic Rotation 23
13.2 Suffix array 23
13.3 String matching – KMP 25
13.4 String matching – Aho–Corasick 25


```
/// ****************************** TEMPLATE ********************************* ///
// Equipe: UFMG SUDO. Competidores: Felipe Menezes Machado, Leonardo Conegundes
// Martinez e Thiago Sonego Goulart. Coach: Itamar Sakae Viana Hata.
// algorithm, bitset, cmath, cstdio, cstdlib, cstring, ctime, deque, stack,
// functional, iostream, list, map, numeric, queue, set, sstream, utility,
// iomanip, string, vector, tr1/tuple, tr1/unordered_map, tr1/unordered_set
using namespace std; using namespace tr1;
#define fori(i, n) for ( int i = 0; i < (n); ++i )
#define forr(i, a, b) for ( int i = (a); i <= (b); ++i )
#define ford(i, a, b) for ( int i = (a); i >= (b); --i )
#define tr(T, i) for (typeof(T.begin()) i = T.begin(); i != T.end(); ++i )
#define sz size()
#define all(x) (x).begin(),(x).end()
#define _sort(x) sort(all(x))
#define pb push_back
#define TRACE(x...) x
#define PRINT(x...) TRACE(printf(x))
#define WATCH(x) TRACE(cout << #x" = " << x << "\n")
const double EPS = 1e-9; const int INF = 0x3F3F3F3F;
int cmpD(double x, double y = 0, double tol = EPS) {
   return ( x <= y + tol ) ? ( x + tol < y ) ? -1 : 0 : 1; }

/// ******************************* NOTAS *********************************** ///
-> PI: 3.14159265358979323846
-> Number or primes: n/ln(n) < f(n) < 1.26*n/ln(n)

-> Modular multiplicative inverse: A^x == 1 (mod M)
If A and M are coprime, then x = phi(m) - 1, where:
phi(n) = (p1-1)*p1^(e1-1) * ... * (pn-1)*pn^(en-1)

-> Polygon number: P(s,n) = ((s-2)*n*n - (s-4)*n) / 2
Nao confundir com binomial coefficients!

-> Catalan Number:
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900,
```

```
2674440, 9694845, 35357670, 129644790, 477638700, 1767263190, 6564120420,
24466267020, 91482563640, 343059613650, 1289904147324, 4861946401452, ...
Cn = C(2n,n) / (n+1) = (2*n)! / ((n+1)!*n!)
Recursive formula => Cn+1 = (4*n + 2) * Cn / (n + 2);

-> Diophantine equations: ax + by = c
if (c % gcd(a,b) != 0) no solution;
Apos dividir a,b,c por gcd(a,b), temos:
x = x0*c + b*t e y = y0*c - a*t,
onde (x0, y0) eh uma solucao qualquer (extended euclid em ax+by=1) e t eh um
inteiro qualquer.

-> Stirling Number of First Kind (Unsigned):
Conta a quantidade de permutações de N elementos com K ciclos disjuntos.
Ex.: 34576182 é uma permutação com 2 ciclos disjuntos (1->3->5->6->1 e
2->4->7->8->2).
S(n,1) = (n-1)!
S(n,k) = (n-1)*S(n-1,k) + S(n-1,k-1)

-> Stirling Number of Second Kind:
Conta a quantidade de modos que um conjunto de N elementos pode ser particionado
em K conjuntos não-vazios.
S(n,1) = 1
S(n,k) = S(n-1,k-1) + k*S(n-1,k)

-> 1st Order Eulerian numbers:
Conta a quantidade de permutações dos números 1..N onde exatamente M elementos
são maiores que o elemento anterior.
Ou seja, v[i-1] < v[i] acontece exatamente M vezes para todo 1 < i <= N.
A(n,0) = A(n,n-1) = 1
A(n,m) = (n-m)*A(n-1,m-1) + (m+1)*A(n-1,m)

-> Pick's theorem:
A = I + B/2 - 1, where: A = area, I = points inside the polygon,
B = points on the boundary,

-> Triangle area = 4/3 median triangle area

-> Sum of two squares:
A number N is expressible as a sum of 2 squares if and only if in the prime
factorization of N, every prime like (4k+3) occurs an even number of times.

/// ****************************** ALGEBRA ****************************** ///
// Simplex
struct simplex {
   // max c * x, s.t: A * x <= b; x >= 0
   simplex( const vector< vector< double > > & A_, const vector< double > & b_,
            const vector< double > & c_ ) : A( A_ ), b( b_ ), c( c_ ) {}
   vector< vector< double > > A; vector< double > b, c, sol;
   vector< bool > N; vector< int > kt; int m, n;
   void pivot( int k, int l, int e ) {
      int x = kt[l]; double p = A[l][e];
      fori(i,k) A[l][i] /= p;
      b[l] /= p; N[e] = false;
      fori(i,m) if (i != l) {b[i] -= A[i][e]*b[l]; A[i][x] = -A[i][e]*A[l][x];}
      fori(j,k) if ( N[j] ) {
         c[j] -= c[e] * A[l][j];
         fori(i,m) if ( i != l ) A[i][j] -= A[i][e] * A[l][j];
      }
      kt[l] = e; N[x] = true; c[x] = -c[e] * A[l][x];
   }
   vector< double > go( int k ) {
      vector< double > res;
      while ( 1 ) {
```

UFMG − Universidade Federal de Minas Gerais

```
         int e = -1, l = -1;
         fori(i,k) if ( N[i] && cmpD( c[i] ) > 0 ) { e = i; break; }
         if ( e == -1 ) break;
         fori(i,m) if ( cmpD(A[i][e]) > 0 && ( l == -1 || cmpD( b[i] / A[i][e],
                        b[l] / A[l][e], 1e-20 ) < 0 ) ) l = i;
         if ( l == -1 ) return vector< double >(); // unbounded
         pivot( k, l, e );
      }
      res.resize( k, 0 );
      fori(i,m) res[kt[i]] = b[i];
      return res;
   }
   vector< double > solve() {
      m = A.sz; n = A[0].sz; int k = m+n+1;
      N = vector< bool >( k, true ); vector< double > c_copy = c;
      c.resize(n+m); kt.resize(m);
      fori(i,m) {
         A[i].resize(k); A[i][n+i] = 1; A[i][k-1] = -1;
         kt[i] = n+i; N[kt[i]] = false;
      }
      int l = min_element(all(b)) - b.begin();
      if ( cmpD( b[l] ) < 0 ) {
         c = vector< double >( k, 0 );
         c[k-1] = -1; pivot(k, l, k-1); sol = go(k);
         if (cmpD(sol[k-1])>0) return vector<double>(); // infeasible
         fori(i,m) if ( kt[i] == k-1 ) {
            fori(j,k-1) if ( N[j] && cmpD( A[i][j] ) != 0 ) {
               pivot( k, i, j ); break;
            }
         }
         c = c_copy; c.resize( k, 0 );
         fori(i,m) fori(j,k) if ( N[j] ) c[j] -= c[kt[i]] * A[i][j];
      }
      sol = go(k-1);
      if ( !sol.empty() ) sol.resize(n);
      return sol;
   }
};
// minimizacao
class Mixture {
public:
   double mix(vector <int> mixture, vector <string> availableMixtures ) {
      string s; int k; int m = mixture.sz; int n = availableMixtures.sz;
      vector< vector< double > > A( m+m, vector< double >( n ) );
      vector< double > c( n ), b( m+m );
      fori(i,m) b[m+i] = -(b[i] = mixture[i]); // vetor b
      fori(j,n) { // matriz A e vetor b
         s = availableMixtures[j];
         istringstream is(s);
         fori(i,m) { is >> k; A[m+i][j] = -(A[i][j] = k); }
         is >> k; c[j] = -k;
      }
      simplex S( A, b, c ); double asw = 0;
      vector< double > sol = S.solve();
      if ( sol.empty() ) asw = 1;
      else fori(i,n) asw += c[i] * sol[i];
      return -asw;
   }
};
// Linear Systems: You should fill the tableau T. Let
// A be an m x n matrix, so the tableau will be (with b[m] and c[n]):
//                           [ -f   |   c[n]   ]
//            T[m+1][n+1] = [-----------------]
//                           [ b[m] | A[m][n]  ]
```

```
// Invert a matrix A[m][m] - store the matrix in the tableau and the identity in
// T[1..m][m+1...2m], make n = 2*m and call solve_linear_system(). Get the inv.
// matrix at T[1..m][m+1...2m]. This code doesn't suppose m == n. After the
// execution, if possible, m is the rank of the matrix. m is the number of lines
// and n the number of columns ( variables ).
int n, m; double x[MAXN+1], T[MAXN+1][MAXN+1]; // tableau
void pivot( int l, int j ) {
   fori(k,n+1) if (k!=j) T[l][k] /= T[l][j]; T[l][j] = 1;
   forr(i,1,m) if ( i != l ) {
      fori(k,n+1) if (k!=j) T[i][k] -= T[l][k] * T[i][j]; T[i][j] = 0;
   }
}
bool solve_linear_system() {
   forr(j,1,n) T[0][j] = j;
   forr(i,1,min(m,n)) {
      int p = i;
      forr(k,i+1,m) if ( cmpD( fabs(T[k][i]), fabs(T[p][i]) ) > 0 ) p = k;
      if (p!=i) fori(j,n+1) swap(T[i][j], T[p][j]);
      if (cmpD(T[i][i])==0) {
         p = i; forr(k,i+1,n) if (cmpD(fabs(T[i][k]),fabs(T[i][p])) > 0) p = k;
         if (p!=i) fori(j,m+1) swap(T[j][i], T[j][p]);
      }
      if ( cmpD(T[i][i])==0 & cmpD(T[i][0])!=0 ) return false;
      else if ( cmpD(T[i][i])!=0 ) pivot(i,i);
      else {
         fori(j,n+1) swap(T[i][j], T[m][j]);
         --i; --m;
      }
   }
   forr(i,(min(m,n))+1,m) if ( cmpD(T[i][0]) != 0 ) return false;
   if (m>n) m = n;
   return true;
}
void get_solution()
{
   forr(i,1,n) x[i] = 0;
   forr(j,1,m) x[(int)T[0][j]] = T[j][0];
}
// eliminacao gaussiana
void gaussian_elimination(double a[MAX][MAX], double b[MAX], int n) {
   int i, j, k, l, maxi; double f, aux;
   for (i=0; i < n; i++) {
      maxi = i;
      for (l=i; l < n; l++) {
         if (fabs(a[l][i]) > fabs(a[maxi][i])) maxi = l;
      }
      for (l=0; l < n; l++) swap(a[i][l],a[maxi][l]);
      aux = b[i], b[i] = b[maxi], b[maxi] = aux;
      for (k=i+1; k < n; k++) {
         f = a[k][i] / a[i][i];
         for (j=i; j < n; j++) a[k][j] -= a[i][j] * f;
         b[k] -= b[i] * f;
      }
   }
   for (i=n-1; i >= 0; i--) {
      b[i] = b[i] / a[i][i]; a[i][i] = 1.0;
      for (j=i-1; j >= 0; j--) { b[j] -= a[j][i] * b[i], a[j][i] = 0.0; }
   }
}

/// ****************************** GEOMETRIA ****************************** ///
struct point {
   double x, y;
   point(double x = 0, double y = 0): x(x), y(y) {}
```

UFMG – Universidade Federal de Minas Gerais

```
   point operator +(point q) { return point(x + q.x, y + q.y); }
   point operator -(point q) { return point(x - q.x, y - q.y); }
   point operator *(double t) { return point(x * t, y * t); }
   point operator /(double t) { return point(x / t, y / t); }
   double operator *(point q) { return x * q.x + y * q.y; }
   double operator %(point q) { return x * q.y - y * q.x; }
   int cmp(point q) const {
      if (int t = ::cmp(x, q.x)) return t; return ::cmp(y, q.y);
   }
   bool operator ==(point q) const { return cmp(q) == 0; }
   bool operator !=(point q) const { return cmp(q) != 0; }
   bool operator < (point q) const { return cmp(q) < 0; }
   friend ostream& operator <<(ostream& o, point p) {
      return o << "(" << p.x << "," << p.y << ")";
   }
   static point pivot;
};
point point::pivot;
double abs(point p) { return hypot(p.x, p.y); }
double arg(point p) { return atan2(p.y, p.x); }
typedef vector<point> polygon;
int ccw(point p, point q, point r) { return cmp((p - r) % (q - r)); }
double angle(point p, point q, point r) {
   point u = p - q, v = r - q; return atan2(u % v, u * v);
}
// Normaliza o vetor para tamanho unitario. Retorna -1 se o vector e' 0
int normalize(point& p) {
   double r = abs(p);
   if( cmp(r) != 0 ) return -1;
   p.x /= r; p.y /= r;
   return 0;
}
// Decide se q esta sobre o segmento fechado pr
bool between(point p, point q, point r) {
   return ccw(p, q, r) == 0 && cmp((p - q) * (r - q)) <= 0;
}
// Decide se os segmentos fechados pq e rs tem pontos em comum.
// s e' o ponto de pq que esta mais proximo de r
bool seg_intersect(point p, point q, point r, point s) {
   point A = q - p, B = s - r, C = r - p, D = s - q;
   int a = cmp(A % C) + 2 * cmp(A % D), b = cmp(B % C) + 2 * cmp(B % D);
   if (a == 3 || a == -3 || b == 3 || b == -3) return false;
   if (a || b || p == r || p == s || q == r || q == s) return true;
   int t = (p < r) + (p < s) + (q < r) + (q < s);
   return t != 0 && t != 4;
}
// Calcula a distancia do ponto r ao segmento pq.
double seg_distance(point p, point q, point r) {
   point A = r - q, B = r - p, C = q - p;
   double a = A * A, b = B * B, c = C * C;
   if (cmp(b, a + c) >= 0) return sqrt(a);
   else if (cmp(a, b + c) >= 0) return sqrt(b);
   else return fabs(A % B) / sqrt(c);
}
// Classifica o ponto p em relacao ao poligono T. Retorna 0, -1 ou 1 dependendo
// se p esta no exterior, na fronteira ou no interior de T, respectivamente.
int in_poly(point p, polygon& T) {
   double a = 0; int N = T.size();
   for (int i = 0; i < N; i++) {
      if (between(T[i], p, T[(i+1) % N])) return -1;
      a += angle(T[i], p, T[(i+1) % N]);
   }
   return cmp(a) != 0;
}
```

```
// Encontra o ponto de intersecao das retas pq e rs.
point line_intersect(point p, point q, point r, point s) {
   point a = q − p, b = s − r, c = point(p % q, r % s);
   return point(point(a.x, b.x) % c, point(a.y, b.y) % c) / (a % b);
}
// Determina o poligono intersecao dos dois poligonos convexos P e Q.
// Tanto P quanto Q devem estar orientados positivamente.
polygon poly_intersect(polygon& P, polygon& Q) {
   int m = Q.size(), n = P.size();
   int a = 0, b = 0, aa = 0, ba = 0, inflag = 0;
   polygon R;
   while ((aa < n || ba < m) && aa < 2*n && ba < 2*m) {
      point p1 = P[a], p2 = P[(a+1) % n], q1 = Q[b], q2 = Q[(b+1) % m];
      point A = p2 − p1, B = q2 − q1;
      int cross = cmp(A % B),ha = ccw(p2, q2, p1), hb = ccw(q2, p2, q1);
      if (cross == 0 && ccw(p1, q1, p2) == 0 && cmp(A * B) < 0) {
         if (between(p1, q1, p2)) R.push_back(q1);
         if (between(p1, q2, p2)) R.push_back(q2);
         if (between(q1, p1, q2)) R.push_back(p1);
         if (between(q1, p2, q2)) R.push_back(p2);
         if (R.size() < 2) return polygon();
         inflag = 1; break;
      } else if (cross != 0 && seg_intersect(p1, p2, q1, q2)) {
         if (inflag == 0) aa = ba = 0;
         R.push_back(line_intersect(p1, p2, q1, q2));
         inflag = (hb > 0) ? 1 : −1;
      }
      if (cross == 0 && hb < 0 && ha < 0) return R;
      bool t = cross == 0 && hb == 0 && ha == 0;
      if (t ? (inflag == 1) : (cross >= 0) ? (ha <= 0) : (hb > 0)) {
         if (inflag == −1) R.push_back(q2);
         ba++; b++; b %= m;
      } else {
         if (inflag == 1) R.push_back(p2);
         aa++; a++; a %= n;
      }
   }
   if (inflag == 0) {
      if (in_poly(P[0], Q)) return P;
      if (in_poly(Q[0], P)) return Q;
   }
   R.erase(unique(all(R)), R.end());
   if (R.size() > 1 && R.front() == R.back()) R.pop_back();
   return R;
}
// Classifica a reta pq em relacao ao circulo C − NAO FOI TESTADA
// Retorna 0 se pq intersecta C em 0 pontos, 1 se pq intersecta C em 1
// ponto, 2 se pq intersecta C em 2 pontos, r e s eh onde ha intersecao
int line_circle(circle C, point p, point q, point& r, point& s) {
   point m; double r0 = seg_distance(p, q, C.first, m);
   if( cmp(r0, C.second) > 0 ) return 0;
   else if( cmp(r0, C.second) == 0 ) {r = s = m; return 1;}
   else {
      double dd = sqrt(C.second*C.second−r0*r0); point v = q−p;
      normalize(v); r = m−v*dd; s = m+v*dd;
      return 2;
   }
}
// line scan sweeping
#define px second
#define py first
typedef pair<double, double> pairdd;
int n, t; // numero de pontos e numero de casos de testes
pairdd pnts [100000]; // conjunto de ponto
```

UFMG − Universidade Federal de Minas Gerais

```
set< pairdd > box; // armazena todos os pontos a uma distancia maxima em
                   // relacao a coordenada x do ponto atual
double best; // distancia entre os dois pontos mais proximos
pairdd pnt_a, pnt_b; // os dois pontos mais proximos
int compx( pairdd a, pairdd b ) { return cmpD( a.px, b.px ) < 0; }
// armazena em pnt_a e pnt_b o par de pontos mais proximos
// armazena em best a distancia entre pnt_a e pnt_b
void closest_points() {
   sort( pnts, pnts+n, compx );
   best = INF;
   box.insert( pnts[0] );
   int left = 0;
   forr(i,1,n−1) {
      // remove pontos a uma distancia maior que best na coordenada x do pnts[i]
      while ( left < i && cmpD( pnts[i].px − pnts[left].px, best ) > 0 )
         box.erase( pnts[left++] );
      // compara todos os pontos do conjunto box com o pnts[i]
      for ( typeof( box.begin() ) it =
            box.lower_bound(make_pair( pnts[i].py − best, pnts[i].px − best ));
            it != box.end() && cmpD( pnts[i].py + best, it−>py ) >= 0; ++it ) {
         double dx = pnts[i].px − it−>px, dy = pnts[i].py − it−>py;
         double dist = sqrt( dx * dx + dy * dy );
         if (cmpD(dist, best) < 0) {best = dist; pnt_a = pnts[i]; pnt_b = *it;}
      }
      box.insert( pnts[i] );
   }
}
int main ()
{
   scanf( "%d", &t );
   while ( t−− ) {
      scanf( "%d", &n );
      box.clear();
      fori(i,n) scanf( "%lf %lf", &pnts[i].px, &pnts[i].py );
      closest_points();
      printf("%.3lf %.3lf\n", (pnt_a.px + pnt_b.px)/2, (pnt_a.py + pnt_b.py)/2);
      if ( t ) puts("");
  }
  return 0;
}
// union of rectangles
// O(n^2), em que n eh o numero de retangulos
struct event
{
   int ind;   // Index of rectangle in rects
   bool type; // Type of event: 0 = Lower−left ; 1 = Upper−right
   event() {}; event( int ind, int type ) : ind( ind ), type( type ) {};
};
struct point { int x, y; };
int n, e; // n = number of rectangles; e = number of edges
point rects[10010][2]; // Each rectangle consists of 2 points:
                       // [0] = lower−left ; [1] = upper−right
int delta_x, delta_y; // distance between current sweep line and previous
event events_v[20010], events_h[20010]; // Events of vertical/horiz. sweep line
bool compare_x( event a, event b ) {
   return rects[a.ind][a.type].x < rects[b.ind][b.type].x; }
bool compare_y( event a, event b ) {
   return rects[a.ind][a.type].y < rects[b.ind][b.type].y; }
int in_set[10010]; // Boolean array in place of balanced binary tree (set)
long long area;    // The output: Area of the union
int main() {
   while ( scanf( "%d", &n ) == 1 ) { /// x −> v; y −> h
      e = area = 0;
      memset( in_set, 0, sizeof( in_set ) );
```

```
      fori(i,n) {
         scanf("%d %d",&rects[i][0].x,&rects[i][0].y); // Lower-left coordinate
         scanf("%d %d",&rects[i][1].x,&rects[i][1].y); // Upper-right coordinate
         events_v[e] = event( i, 0 ); events_h[e++] = event( i, 0 );
         events_v[e] = event( i, 1 ); events_h[e++] = event( i, 1 );
      }
      sort( events_v, events_v + e, compare_x );
      sort( events_h, events_h + e, compare_y ); // sort set of horizontal edges
      in_set[events_v[0].ind] = 1;
      forr(i,1,e-1) {
         event c = events_v[i]; // Vertical sweep line
         int cnt = 0; // how many rectangles are currently overlapping?
         delta_x = rects[c.ind][c.type].x -
                   rects[events_v[i-1].ind][events_v[i-1].type].x;
         int begin_y = 0;
         //if ( delta_x == 0 ) continue;
         fori(j,e) if ( in_set[events_h[j].ind] == 1 ) {
            if ( events_h[j].type == 0 ) { // Horizontal sweep line
               // Block starts
               if ( cnt == 0 ) begin_y = rects[events_h[j].ind][0].y; ++cnt;
            }
            else {
               --cnt;
               if ( cnt == 0 ) { // Block ends
                  delta_y = (rects[events_h[j].ind][1].y - begin_y);
                  area += delta_x * delta_y;
               }
            }
         }
         in_set[c.ind] = ( c.type == 0 );
      }
      printf("%lld\n", area);
   }
   return 0;
}
// O(n log K) com arvore de intervalos, onde n eh o numero de retangulos
// e K eh o tamanho do maior intervalo (tamanho do eixo y valido)
int main() {
   while ( scanf( "%d", &n ) == 1 ) { /// x -> v; y -> h
      e = area = 0;
      int max_y = 0, min_y = INF;
      fori(i,n) {
         scanf( "%d %d", &rects[i][0].x, &rects[i][0].y );
         scanf( "%d %d", &rects[i][1].x, &rects[i][1].y );
         min_y = min(min_y, rects[i][0].y); max_y = max(max_y, rects[i][1].y);
         events_v[e++] = event( i, 0 ); events_v[e++] = event( i, 1 );
      }
      sort( events_v, events_v + e, compare_x );
      create( min_y, max_y );
      update( rects[events_v[0].ind][0].y, rects[events_v[0].ind][1].y, 1 );
      forr(i,1,e-1) {
         event c = events_v[i]; // vertical sweep line
         delta_x = rects[c.ind][c.type].x -
                   rects[events_v[i-1].ind][events_v[i-1].type].x;
         delta_y = count_ocurrences( min_y, max_y );
         area += delta_x * delta_y;
         if ( c.type == 0 ) update( rects[c.ind][0].y, rects[c.ind][1].y, 1 );
         else update( rects[c.ind][0].y, rects[c.ind][1].y, -1 );
      }
      printf("%lld\n", area);
   }
   return 0;
}
```

UFMG − Universidade Federal de Minas Gerais

```
typedef struct {
   int x, y;
} Point;
typedef Point Polygon[MAX];
double dist_pr(double x, double y, double a, double b, int inf) {
   double tgaux, baux;
   if (inf) return fabs(b-x);
   if (equals(a,0.0)) return fabs(b-y);
   tgaux = -1.0 / a;
   baux = y - tgaux * x;
   return dist_pp(x,y,(baux-b)/(a-tgaux),a*(baux-b)/(a-tgaux)+b);
}
// Convex-hull de um poligono anti-horario.
int xmin, ymax;
int cmp_sort(Point *a, Point *b) {
   if (a->x == xmin && b->x == xmin) return b->y - a->y;
   if (a->x == xmin) return -1;
   if (b->x == xmin) return 1;
   if ((a->y == ymax && b->y == ymax) ||
    prod_vet(a->x-xmin,a->y-ymax,b->x-xmin,b->y-ymax) == 0) return a->x - b->x;
   return -prod_vet(a->x-xmin,a->y-ymax,b->x-xmin,b->y-ymax);
}
void sort(Polygon p, int n) {
   int i, j, min = 0;
   Point aux;
   for (i=1; i < n; i++) {
      if (p[i].x <  p[min].x || (p[i].x == p[min].x && p[i].y > p[min].y))
         min = i;
   }
   if (min != 0) aux = p[0], p[0] = p[min], p[min] = aux;
   xmin = p[0].x, ymax = p[0].y;
   qsort(&p[1],n-1,sizeof(Point),(void*)cmp_sort);
   for (i=n-1; i > 0 && prod_vet(p[i].x-p[0].x,p[i].y-p[0].y,p[i-1].x-p[0].x,
      p[i-1].y-p[0].y) == 0; i--);
   for (j=0; j < (n-i)/2; j++) aux = p[i+j], p[i+j] = p[n-j-1], p[n-j-1] = aux;
}
int convex_hull(Polygon p, int n, Polygon hull) {
   int qtdepontos = 2, i;
   double x1, y1, x2, y2;
   sort(p,n);
   hull[0] = p[0], hull[1] = p[1];
   for (i=2; i <= n; i++) {
      do {
         x1 = p[i%n].x - hull[qtdepontos-1].x;
         y1 = p[i%n].y - hull[qtdepontos-1].y;
         x2 = hull[qtdepontos-2].x - hull[qtdepontos-1].x;
         y2 = hull[qtdepontos-2].y - hull[qtdepontos-1].y;
         if (prod_vet(x1,y1,x2,y2) <= 0 && i != n) qtdepontos--;
         else break;
      }
      while (qtdepontos > 1);
      if (i != n) hull[qtdepontos++] = p[i];
   }
   return qtdepontos;
}
// Interseccao de retas e segmentos.
int intersect(double x0, double y0, double x1, double y1, double x2, double y2,
              double x3, double y3, double *x, double *y) {
   double a1, b1, a2, b2;
   if (equals(x0,x1) && equals(x2,x3)) return 0; // tratar retas verticais?
   if (!equals(x0,x1)) a1 = (y1-y0)/(x1-x0), b1 = y0 - a1*x0;
   if (!equals(x2,x3)) a2 = (y3-y2)/(x3-x2), b2 = y2 - a2*x2;
   if (equals(x0,x1)) *x = x0, *y = a2*x0 + b2;
   else if (equals(x2,x3)) *x = x2, *y = a1*x2 + b1;
```

```
   else {
      if (equals(a1,a2)) return 0; // tratar retas colineares?
      *x = (b2 - b1) / (a1 - a2), *y = a1*(*x) + b1;
   }
   if (!equals(dist_pp(*x,*y,x0,y0)+dist_pp(*x,*y,x1,y1),dist_pp(x0,y0,x1,y1)))
      return 0; // se (p0,p1) eh segmento
   if (!equals(dist_pp(*x,*y,x2,y2)+dist_pp(*x,*y,x3,y3),dist_pp(x2,y2,x3,y3)))
      return 0; // se (p2,p3) eh segmento
   return 1;
}
// Interseccao de segmentos (v2). pv = prod_vet
int segment_intersect(int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4) {
return ((pv(x1-x2,y1-y2,x3-x2,y3-y2) < 0 && pv(x1-x2,y1-y2,x4-x2,y4-y2) > 0) ||
      (pv(x1-x2,y1-y2,x3-x2,y3-y2) > 0 && pv(x1-x2,y1-y2,x4-x2,y4-y2) < 0)) &&
      ((pv(x4-x3,y4-y3,x1-x3,y1-y3) < 0 && pv(x4-x3,y4-y3,x2-x3,y2-y3) > 0) ||
      (pv(x4-x3,y4-y3,x1-x3,y1-y3) > 0 && pv(x4-x3,y4-y3,x2-x3,y2-y3) < 0));
}
// Ponto dentro de um poligono anti-horario qualquer.
int point_in_pol(int x, int y, Polygon p, int n) {
int i, cuts = 0;
double xaux;
for (i=1; i <= n; i++) {
   if ((p[i-1].x == x && p[i-1].y == y) ||
      (prod_vet(p[i-1].x-x,p[i-1].y-y,p[i%n].x-x,p[i%n].y-y) == 0 &&
      equals(dist_pp(x,y,p[i-1].x,p[i-1].y)+dist_pp(x,y,p[i%n].x,p[i%n].y),
      dist_pp(p[i%n].x,p[i%n].y,p[i-1].x,p[i-1].y)))) {
      return 1;
   }
   if (p[i-1].y == y && p[i-1].y == p[i%n].y) {
      if (p[i-1].x > x && p[i%n].x > x && ((p[(n+i-2)%n].y < y &&
      p[(i+1)%n].y > y) || (p[(n+i-2)%n].y > y && p[(i+1)%n].y < y))) cuts++;
      continue;
   }
   if (p[i%n].y == y && p[i%n].x > x && ((p[i-1].y < y && p[(i+1)%n].y > y) ||
      (p[i-1].y > y && p[(i+1)%n].y < y))) { cuts++; continue; }
   if (p[i%n].x == p[i-1].x) xaux = p[i%n].x;
   else xaux = (y - p[i%n].y + (p[i%n].y-p[i-1].y)/(double)(p[i%n].x-p[i-1].x)*
            p[i%n].x) / ((p[i%n].y-p[i-1].y)/(double)(p[i%n].x-p[i-1].x));
   if (xaux+EPS > x && ((y > p[i%n].y && y < p[i-1].y) || (y < p[i%n].y && y >
         p[i-1].y))) cuts++;
}
return cuts & 1;
}
void centroid(Polygon p, int n, Point_d *r) {
int i, x = 0, y = 0; double s = area(p,n);
for (i=1; i <= n; i++) {
   x += (p[i-1].x + p[i].x)*prod_vet(p[i-1].x, p[i-1].y, p[i%n].x, p[i%n].y);
   y += (p[i-1].y + p[i].y)*prod_vet(p[i-1].x, p[i-1].y, p[i%n].x, p[i%n].y);
}
r->x = x/(6.0*s); r->y = y/(6.0*s);
}
// Circulo definido por 3 pontos.
double x, y, r;
void find_circle(Point *p1, Point *p2, Point *p3) {
   double x1, y1, x2, y2, x3, y3, temp;
   x1 = p1->x, y1 = p1->y;
   x2 = p2->x, y2 = p2->y;
   x3 = p3->x, y3 = p3->y;
   if (equals(x1,x2)) swap(y3,y2), swap(x3,x2);
   y = ((x1-x2)*(x1*x1+y1*y1-x3*x3-y3*y3)-(x1-x3)*(x1*x1+y1*y1-x2*x2-y2*y2)) /
      (2*((y2-y1)*(x1-x3)-(y3-y1)*(x1-x2)));
   x = (x1*x1+y1*y1-x2*x2-y2*y2+2*y*(y2-y1)) / (2*(x1-x2));
   r = sqrt((x-x1)*(x-x1)+(y-y1)*(y-y1));
}
```

UFMG – Universidade Federal de Minas Gerais

```
// Menor circulo que engloba um conjunto de pontos.
// -> soh funciona para pontos que facam parte do convex hull do conjunto!!!!!!!
void min_circle(Polygon p, int n) {
   int i = 0, j = 1, k, posmin;
   double PI = acos(-1.0), min, a;
   while (1) {
      min = PI;
      for (k=0; k < n; k++) {
         if (k != i && k != j) {
   a = acos(((p[i].x-p[k].x)*(p[j].x-p[k].x)+(p[i].y-p[k].y)*(p[j].y-p[k].y)) /
      (sqrt((p[i].x-p[k].x)*(p[i].x-p[k].x)+(p[i].y-p[k].y)*(p[i].y-p[k].y))*
      sqrt((p[j].x-p[k].x)*(p[j].x-p[k].x)+(p[j].y-p[k].y)*(p[j].y-p[k].y))));
            if (a < min) min = a, posmin = k;
         }
      }
      if (min+EPS > PI/2) {
         x = (p[i].x + p[j].x) / 2.0; y = (p[i].y + p[j].y) / 2.0;
         r = sqrt((p[i].x-p[j].x)*(p[i].x-p[j].x) +
                 (p[i].y-p[j].y)*(p[i].y-p[j].y)) / 2;
         return;
      }
      if (acos(((p[posmin].x-p[i].x)*(p[j].x-p[i].x)+(p[posmin].y-p[i].y)*
      (p[j].y-p[i].y)) / (sqrt((p[posmin].x-p[i].x)*(p[posmin].x-p[i].x)+
      (p[posmin].y-p[i].y)*(p[posmin].y-p[i].y))*sqrt((p[j].x-p[i].x)*
      (p[j].x-p[i].x)+(p[j].y-p[i].y)*(p[j].y-p[i].y))))-EPS > PI/2)
         i = posmin;
      else if (acos(((p[i].x-p[j].x)*(p[posmin].x-p[j].x)+(p[i].y-p[j].y)*
      (p[posmin].y-p[j].y)) / (sqrt((p[i].x-p[j].x)*(p[i].x-p[j].x)+
      (p[i].y-p[j].y)*(p[i].y-p[j].y))*sqrt((p[posmin].x-p[j].x)*(p[posmin].x
      -p[j].x)+(p[posmin].y-p[j].y)*(p[posmin].y-p[j].y))))-EPS > PI/2)
         j = posmin;
      else { find_circle(&p[i],&p[j],&p[posmin]); return; }
   }
}
// distancia esferica
double spherical_dist(double p_lat, double p_long, double q_lat, double q_long){
return acos(sin(p_lat)*sin(q_lat)+cos(p_lat)*cos(q_lat)*cos(p_long-q_long))*6378
}
// geometria 3D
int cmpD(double a, double b = 0.0) { return a+EPS < b ? -1 : a-EPS > b; }
struct Point {
   double x, y, z;
   Point(double a=0.0,double b=0.0,double c=0.0){x=a,y=b,z=c;}
   Point operator+(const Point &P) const {return Point(x+P.x,y+P.y,z+P.z);}
   Point operator-(const Point &P) const {return Point(x-P.x,y-P.y,z-P.z);}
   Point operator*(double c) const {return Point(x*c,y*c,z*c);}
   Point operator/(double c) const {return Point(x/c,y/c,z/c);}
   double operator!() const {return sqrt(x*x+y*y+z*z);}
};
double dot(Point A, Point B) { return A.x*B.x + A.y*B.y + A.z*B.z; }
Point cross(Point A, Point B) {
   return Point(A.y*B.z-A.z*B.y, A.z*B.x-A.x*B.z, A.x*B.y-A.y*B.x);
}
Point project(Point W, Point V) { return V * dot(W,V) / dot(V,V); }
// do the segments A-B and C-D intersect? (assumes coplanar)
bool seg_intersect(Point A, Point B, Point C, Point D) {
   return cmpD(dot(cross(A-B,C-B),cross(A-B,D-B))) <= 0 &&
   cmpD(dot(cross(C-D,A-D),cross(C-D,B-D))) <= 0;
}
double dist_point_seg(Point P, Point A, Point B) {
   Point PP = A + project(P-A,B-A);
   if (cmpD(!(A-PP)+!(PP-B),!(A-B)) == 0) return !(P-PP);//distance point-line!
   return min(!(P-A),!(P-B));
}
```

```
// segment-segment distance (lines too!)
double dist_seg_seg(Point A, Point B, Point C, Point D) {
   Point E = project(A-D,cross(B-A,D-C)); // distance between lines!
   if (seg_intersect(A,B,C+E,D+E)) return !E;
   return min( min( dist_point_seg(A,C,D), dist_point_seg(B,C,D) ),
         min( dist_point_seg(C,A,B), dist_point_seg(D,A,B) ) );
}
// point-triangle distance. dps = dist_point_seg
double dist_point_tri(Point P, Point A, Point B, Point C) {
   Point N = cross(A-C,B-C);
   Point PP = P + project(C-P,N);
   Point V1 = cross(PP-A,B-A);
   Point V2 = cross(PP-B,C-B);
   Point V3 = cross(PP-C,A-C);
   if (cmpD(dot(V1,V2)) >= 0 && cmpD(dot(V1,V3)) >= 0 && cmpD(dot(V2,V3)) >= 0)
      return !(PP-P); // distance point-plane!
   return min(dps(P,A,B),min(dps(P,A,C),dps(P,B,C)));
}
double dist_tet_tet(Point T1[4], Point T2[4]) {
   double ans = INF;
   for (int i=0; i < 4; i++) // arestas -> arestas
      for (int j=i+1; j < 4; j++)
         for (int ii=0; ii < 4; ii++)
            for (int jj=ii+1; jj < 4; jj++)
               ans = min( ans, dist_seg_seg(T1[i],T1[j],T2[ii],T2[jj]) );
   // pontos -> planos
   for (int i=0; i < 4; i++)
      for (int j=i+1; j < 4; j++)
         for (int k=j+1; k < 4; k++)
            for (int x=0; x < 4; x++)
               ans = min( ans, dist_point_tri(T1[x],T2[i],T2[j],T2[k]) ),
               ans = min( ans, dist_point_tri(T2[x],T1[i],T1[j],T1[k]) );
   return ans;
}
double volume(Point T[4]){return fabs(dot(T[3],cross(T[1]-T[0],T[2]-T[0])))/6.;}

/// ************************ CLASSES DE NÚMEROS ************************** ///
// bigint
const int DIG = 4;
const int BASE = 10000; // BASE**3 < 2**51
const int TAM = 2048;
struct bigint {
   int v[TAM], n;
   bigint(int x = 0): n(1) { memset(v, 0, sizeof(v)); v[n++] = x; fix(); }
   bigint(char *s): n(1) {
      memset(v, 0, sizeof(v));
      int sign = 1;
      while (*s && !isdigit(*s)) if (*s++ == '-') sign *= -1;
      char *t = strdup(s), *p = t + strlen(t);
      while (p > t) {
         *p = 0; p = max(t, p - DIG);
         sscanf(p, "%d", &v[n]);
         v[n++] *= sign;
      }
      free(t); fix();
   }
   bigint& fix(int m = 0) {
      n = max(m, n);
      int sign = 0;
      for (int i = 1, e = 0; i <= n || e && (n = i); i++) {
         v[i] += e; e = v[i] / BASE; v[i] %= BASE;
         if (v[i]) sign = (v[i] > 0) ? 1 : -1;
      }
      for (int i = 1; i < n; i++)
```

UFMG − Universidade Federal de Minas Gerais

```
         if (v[i] * sign < 0) { v[i] += sign * BASE; v[i+1] -= sign; }
      while (n && !v[n]) n--;
      return *this;
   }
   int cmp(const bigint& x = 0) const {
      int i = max(n, x.n), t = 0;
      while (1) if ((t = ::cmp(v[i], x.v[i])) || i-- == 0) return t;
   }
   bool operator <(const bigint& x) const { return cmp(x) < 0; }
   bool operator ==(const bigint& x) const { return cmp(x) == 0; }
   bool operator !=(const bigint& x) const { return cmp(x) != 0; }
   operator string() const {
      ostringstream s; s << v[n];
      for (int i = n-1; i > 0; i--) {s.width(DIG); s.fill('0'); s << abs(v[i]);}
      return s.str();
   }
   friend ostream& operator<<(ostream& o, const bigint& x){return o<<(string)x;}
   bigint& operator +=(const bigint& x) {
      for (int i = 1; i <= x.n; i++) v[i] += x.v[i];
      return fix(x.n);
   }
   bigint operator +(const bigint& x) { return bigint(*this) += x; }
   bigint& operator -=(const bigint& x) {
      for (int i = 1; i <= x.n; i++) v[i] -= x.v[i];
      return fix(x.n);
   }
   bigint operator -(const bigint& x) { return bigint(*this) -= x; }
   bigint operator -() { bigint r = 0; return r -= *this; }
   void ams(const bigint& x, int m, int b) { // *this += (x * m) << b;
      for (int i = 1, e = 0; (i <= x.n || e) && (n = i + b); i++) {
         v[i+b] += x.v[i] * m + e; e = v[i+b] / BASE; v[i+b] %= BASE;
      }
   }
   bigint operator *(const bigint& x) const {
      bigint r;
      for (int i = 1; i <= n; i++) r.ams(x, v[i], i-1);
      return r;
   }
   bigint& operator *=(const bigint& x) { return *this = *this * x; }
   // cmp(x / y) == cmp(x) * cmp(y); cmp(x % y) == cmp(x);
   bigint div(const bigint& x) {
      if (x == 0) return 0;
      bigint q; q.n = max(n - x.n + 1, 0);
      int d = x.v[x.n] * BASE + x.v[x.n-1];
      for (int i = q.n; i > 0; i--) {
         int j = x.n + i - 1;
         q.v[i] = int((v[j] * double(BASE) + v[j-1]) / d);
         ams(x, -q.v[i], i-1);
         if (i == 1 || j == 1) break;
         v[j-1] += BASE * v[j]; v[j] = 0;
      }
      fix(x.n); return q.fix();
   }
   bigint& operator /=(const bigint& x) { return *this = div(x); }
   bigint& operator %=(const bigint& x) { div(x); return *this; }
   bigint operator /(const bigint& x) { return bigint(*this).div(x); }
   bigint operator %(const bigint& x) { return bigint(*this) %= x; }
   bigint pow(int x) {
      if (x < 0) return (*this == 1 || *this == -1) ? pow(-x) : 0;
      bigint r = 1;
      for (int i = 0; i < x; i++) r *= *this;
      return r;
   }
   bigint root(int x) {
```

```cpp
      if (cmp() == 0 || cmp() < 0 && x % 2 == 0) return 0;
      if (*this == 1 || x == 1) return *this;
      if (cmp() < 0) return -(-*this).root(x);
      bigint a = 1, d = *this;
      while (d != 1) {
         bigint b = a + (d /= 2);
         if (cmp(b.pow(x)) >= 0) { d += 1; a = b; }
      }
      return a;
   }
   friend bigint gcd(bigint x, bigint y){return (y.cmp()!=0 ? gcd(y,x%y) : x);}
};
// fracoes
typedef long long number;
number gcd(number a, number b) { return b ? gcd(b,a%b) : a; }
struct Frac {
   number n, d; bool ok;
   Frac(number a, number b, bool v = true) {
      ok = v && b;
      if (ok) { n = a/gcd(a,b), d = b/gcd(a,b); if (d < 0) n *= -1, d *= -1; } }
   Frac operator+(const Frac &f) const{return Frac(n*f.d+f.n*d,d*f.d,ok&&f.ok);}
   Frac operator-(const Frac &f) const{return Frac(n*f.d-f.n*d,d*f.d,ok&&f.ok);}
   Frac operator*(const Frac &f) const{return Frac(n*f.n,d*f.d,ok && f.ok);}
   Frac operator/(const Frac &f) const{return Frac(n*f.d,d*f.n,ok && f.ok);}
   void print() const { if (!ok) puts("INVALID");
      else if (d == 1) printf("%lld\n",n);else printf("%lld|%lld\n",n,d); }
};
Frac eval(char s[], int from, int to) {
   for (int k=0; k < 3; k++) for (int i=to,d=0; i >= from; i--) {
      if (s[i] == ')') d++; else if (s[i] == '(') d--;
      if (d > 0) continue; assert(d == 0);
      if (k == 0 && s[i] == '+') return eval(s,from,i-1) + eval(s,i+1,to);
      if (k == 0 && s[i] == '-') return eval(s,from,i-1) - eval(s,i+1,to);
      if (k == 1 && s[i] == '*') return eval(s,from,i-1) * eval(s,i+1,to);
      if (k == 1 && s[i] == '/') return eval(s,from,i-1) / eval(s,i+1,to);
      if (k == 2 && s[i] == '|') return eval(s,from,i-1) / eval(s,i+1,to);
   }
   if (s[from] == '(' && s[to] == ')') return eval(s,from+1,to-1);
   number n = 0;
   while (from <= to) {
      assert(s[from] >= '0' && s[from] <= '9'); n = n*10 + s[from++] - '0'; }
   return Frac(n,1);
}

/// ********************************** DP ******************************** ///
// Longest Increasing Subsequence - LIS O(n log n)
void lis( const vector< int > & v, vector< int > & asw ) {
   vector<int> pd(v.sz,0), pd_index(v.sz), pred(v.sz);
   int maxi = 0, x, j, ind;
   fori(i,v.sz) {
      x = v[i];
      j = lower_bound( pd.begin(), pd.begin() + maxi, x ) - pd.begin();
      pd[j] = x; pd_index[j] = i;
      if( j == maxi ) { maxi++; ind = i; }
      pred[i] = j ? pd_index[j-1] : -1;
   } // return maxi;
   int pos = maxi-1, k = v[ind];
   asw.resize( maxi );
   while ( pos >= 0 ) {
      asw[pos--] = k;
      ind = pred[ind];
      k = v[ind];
   }
}
```

UFMG − Universidade Federal de Minas Gerais

```cpp
// LIS 2D - O(n log n log m), em que m é o tamanho da resposta
struct point {
   int x, y;
   bool operator<(const point& o) const {if(x!=o.x) return x<o.x; return y<o.y;}
   bool is_dominated( const point & o ) { return o.x <= x && o.y <= y; }
};
const int MAXN = 100010;
point points[MAXN]; set< point > level[MAXN]; set< point >::iterator aux[MAXN];
int n;
int lis2d() {
   int tam = 1, cnt_pnts, cnt_levels = 0; point p, p2;
   set< point > S; set< point >::iterator it;
   level[cnt_levels].clear(); // o primeiro ponto eh inserido no primeiro nivel
   level[cnt_levels++].insert(points[0]);
   forr(i,1,n-1) {
      p = points[i]; // proximo ponto
      // determina nivel do prox. ponto: o proximo ponto sera colocado no maior
      // nivel i tal que exista algum ponto no nivel i-1 que tenha coordenadas x
      // e y menores ou iguais as coordenadas do novo ponto: busca binaria
      int lo = 0, hi = tam, m, lev = 0;
      while ( lo < hi ) {
         m = lo+(hi-lo)/2;
         // p/ saber se no nivel m existe algum pto que domina o ponto p, testar
         // a dominancia do ponto imediatamente anterior ao primeiro ponto >= p
         it = level[m].lower_bound( p );
         if ( it != level[m].begin() ) --it;
         if ( p.is_dominated( *it ) ) { lo = m+1; lev = m+1; } else hi = m;
      }
      if ( lev == tam ) { // cria-se novo nivel
         level[cnt_levels].clear(); level[cnt_levels++].insert(p); tam++;
      }
      else { // remove ptos r de level[lev] tais que p.x <= r.x e p.y <= r.y.
         // Assim, em qualquer instante, os pontos de level[lev] terao
         // coordenadas x crescentes e coordenadas y decrescentes
         cnt_pnts = 0;
         for (it=level[lev].lower_bound(p);it!=level[lev].end();++it) {
            if ( p.y <= it->y ) aux[cnt_pnts++] = it;
            else break; // ja que a coordenada y esta diminuindo...
         }
         ford(j,cnt_pnts-1,0) level[lev].erase( aux[j] ); level[lev].insert(p);
      }
   }
   return cnt_levels; // o numero de niveis eh igual ao tamanho da LIS
}
// Longest Common Subsequence: LCS O(nm) tempo e espaco
typedef pair<int, int> ii;
string lcs( vector<string> s1, vector<string> s2 ) {
   int M = s1.sz, N = s2.sz;
   vector< vector<int> > m(M+1, vector<int>(N+1, 0));
   vector< vector<ii > > p(M+1, vector<ii>(N+1, ii()));
   ford(i,M-1,0) ford(j,N-1,0) {
      if(s1[i] == s2[j]) {m[i][j] = 1 + m[i+1][j+1]; p[i][j] = make_pair(1, 1);}
      else {
         if (m[i][j+1] > m[i+1][j]) {m[i][j]=m[i][j+1]; p[i][j]=make_pair(0,1)};
         else { m[i][j] = m[i+1][j]; p[i][j] = make_pair(1, 0) };
      }
   }
   string asw = "";
   int len = m[0][0], i = 0, j = 0, b1, b2;
   while ( len ) {
      if (p[i][j]==make_pair(1,1)) {asw+=s1[i]; len--; if (len != 0) asw+=" ";}
      b1 = p[i][j].first; b2 = p[i][j].second;
      i += b1; j += b2;
   }
```

```
   return asw;
}
// Maximum Sum 2D: O(n^3)
int maxSum2D( const vector< vector<int> >& M ) {
   int N = M.sz;
   // s[i][j][k] = soma dos nros. da k-esima coluna entre as linhas [i,j]
   vector<vector<vector<int> > > s(N,vector< vector<int> >(N,vector<int>(N,0)));
   fori(k,N) fori(i,N) forr(j,i,N-1)
   if ( i == j ) s[i][j][k] = M[i][k];
   else s[i][j][k] = s[i][j-1][k] + M[j][k];
   int MAX = -INF; fori(i,N) fori(j,N) MAX = max( MAX, maxSum1D( s[i][j] ) );
   return MAX;
}
// Knapsack - O(nW) - conflito: estabelecer ordem de precedencia entre itens.
// pred[i] = ind. do ult. item que pode ser colocado na mochila antes do item i
vector<int> knapsack( const vector<int>& w, const vector<int>& u, int W
                  /*, const vector<int>& pred */ ) {
   int N = w.sz, aux;
   // M[i][j] -> max. utilidade de uma mochila de cap. max. j com itens de 0 a i
   vector< vector<int> > M( N+1, vector<int>(W+1, 0) );
   vector< vector<int> > s(N+1, vector<int>(W+1, 0)); // quais itens na mochila
   forr(i,1,N) forr(j,1,W) {
      if ( w[i-1] > j ) M[i][j] = M[i-1][j]; // caso 1: item i n cabe na mochila
      else { // caso 2: item i cabe na mochila de capacidade j
         aux = M[i-1][j-w[i-1]] + u[i-1];
         // aux = M[pred[i-1]+1][j-w[i-1]] + u[i-1];
         // >= se quiser maximizar a quantidade de itens na mochila
         if( aux > M[i-1][j] ) { M[i][j] = aux; s[i][j] = 1; }
         else M[i][j] = M[i-1][j];
      }
   }
   // M[N][W] - utilidade maxima da mochila
   vector<int> sol; // subconjunto de itens que devem ser colocados na mochila
   int j = W;
   ford(i,N,1) {
      if( s[i][j] == 1 ) {
         sol.pb(w[i-1]); j -= w[i-1];
         // i = pred[i-1]+1; // adicionar se for mochila com conflito
      }
   }
   reverse(all(sol));
   return sol;
}
// inteira (com repeticao) - O(nW)
int knapsack( const vector<int>& w, const vector<int>& u, int W ) {
   // M[i] -> maxima utilidade de uma mochila de capacidade maxima i
   int N = w.sz, aux;
   vector<int> M( W+1, 0 );
   forr(i,1,W) {
      aux = -INF;
      fori(j,N) if (w[j] <= i && (u[j] + M[i-w[j]] > aux)) aux=u[j] + M[i-w[j]];
      M[i] = max( aux, M[i-1] );
   }
   return M[W];
}
// TSP: dado um subconjunto S dos vertices com 0 \in S, seja C(S,j) o menor
// percurso comecando em 0 que visita todos os vertices de S e termina em j.
// Se |S| = 2, entao C(S,j) = d[0][j], \forall j = 2, ..., n. Se |S| > 2, entao
// C(S,j) = min_{k \neq j}( C(S\{j},k) + d[k][j] ). dp[i][j] = custo minimo de
// visitar os vertices do estado i comecando em 0 e terminando em j - O(2^n n^2)
int tsp( const vector< vector<int> >& g ) {
   int n = g.sz, x, current_cost;
   vector< vector< int > > dp( 1 << n, vector< int >( n, INF ) );
   fori(j,n) dp[0][j] = 0, dp[1<<j][j] = g[0][j];
```

UFMG – Universidade Federal de Minas Gerais

```
   forr(i,1,(1<<n)-1) fori(j,n) if( !( i & (1<<j) ) ) {
      x = i | (1<<j), current_cost = 0;
      fori(k,n) if( i & (1<<k) ) dp[x][j] = min( dp[x][j], dp[i][k] + g[k][j] );
   }
   return dp[(1<<n)-1][0];
}

/// ******************************* JOGOS ******************************* ///
// Dadas N moedas, a cada jogada um jogador pode retirar moves[i] moedas.
// Ganha quem retirar a ultima moeda. (1) Todas as posicoes terminais sao
// perdedoras. (2) Se eh possivel mover a partir da posicao i para uma posicao
// perdedora, entao a posicao i eh vencedora. (3) Se eh possivel mover a partir
// da posicao i apenas para posicoes vencedoras, entao a posicao i eh perdedora.
// conta o numero de posicoes perdedoras do jogador 1 entre 0 e N
int MAX_COINS = 22; // no. maximo de moedas retiradas em uma jogada
int number_of_loosing_positions( int N, vector< int > moves ) {
   // ultimas MAX_COINS posicoes em relacao ao jogador 1
   int mask = ( 1 << MAX_COINS ) - 1; // 1 -> vencedora e 0 -> perdedora
   int res = -1; // numero de posicoes perdedoras do jogador 1 de 0 a N
   map< int, int > last; // last[i]=ult. ind. em que a mascara i ocorreu
   vector<int> r; // r[i]=numero de posicoes perdedoras do jogador 1 de 0 ate i
   forr(i,0,N) {
      // assume inicialmente que a posicao i eh perdedora para o jogador 1
      // (ultimo bit de mask igual a 0)
      mask <<= 1; ++res;
      tr(moves,it) { // decide se posicao i eh vencedora para o jogador 1
         if ( !( mask & (1 << *it) ) ) { // eh possivel ir p/ uma posicao perd.?
            ++mask; --res; break; // a posicao i eh vencedora para o jogador 1
         }
      }
      mask &= (1 << MAX_COINS)-1; // considera apenas as ult. MAX_COINS posicoes
      if ( last.find( mask ) != last.end() ) { // detecta repeticoes
         int cycle_size = i - last[mask]; // tamanho do ciclo
         int num_cycles = ( N - i ) / cycle_size;  //ciclos de i ate N
         int loosing_positions_in_cycle = res - r[ last[mask] ];
         res += num_cycles * loosing_positions_in_cycle;
         i += num_cycles * cycle_size;
      }
      last[mask] = i; // armazena mascara atual
      r.pb(res);
   }
   return res;
}
// Jogo de Nim: N pilhas com moedas. A cada jogada um jogador retira uma
// quantidade positiva de moedas de uma das pilhas. Ganha quem retirar moedas
// pela ultima vez. Seja n_1, n_2, ..., n_N o numero de moedas nas N pilhas.
// Esta eh uma posicao perdedora se e somente se n_1 xor ... xor n_N = 0
int is_winning_nim( vector< int > piles ) {
   int res = 0; fori(i,piles.sz) res ^= piles[i]; return res != 0;
}
// Grundy numbers - Cada posicao na matriz representa o menor inteiro nao
// negativo que nao pode ser alcancado a partir daquela posicao
// OBS: Inicializar matrix com -1
const int K = 1, ROWS = 8, COLUMNS = 8;
int matrix[K][ROWS][COLUMNS], NUM_LEGAL_MOVES = 4;
int dx[4] = {-2, -2, -1, +1}, dy[4] = {+1, -1, -2, -2};
#define valid(x,y) (x) >= 0 && (x) < COLUMNS && (y) >= 0 && (y) < ROWS
int grundy_number( int ind, int x, int y ) {
   if ( matrix[ind][y][x] != -1 ) return matrix[ind][y][x];
   int nx, ny, aux;
   vector< int > alcancavel( NUM_LEGAL_MOVES + 1, 0 );
   fori(i,NUM_LEGAL_MOVES) {
      nx = x + dx[i]; ny = y + dy[i];
      if( valid( nx, ny ) ) alcancavel[grundy_number( ind, nx, ny )]=1;
```

```
   }
   fori(i,NUM_LEGAL_MOVES+1) if ( !alcancavel[i] ) return matrix[ind][y][x] = i;
   return matrix[ind][y][x] = NUM_LEGAL_MOVES + 1;
}

/// ******************************** GRAFOS ******************************** ///
// Dijkstra - O(m log n) - lista de adjacencia. arco: (cost, W)
typedef pair<int, int> ii;
void dijkstra( const vector< vector<ii> >& g, int v, vector<int>& dist ) {
   int d, cost, w; set<ii> Q;
   dist[v] = 0; Q.insert( ii(0, v) );
   while( !Q.empty() ) {
      ii top = *Q.begin();
      Q.erase( Q.begin() ); v = top.second; d = top.first;
      fori(i,g[v].sz) {
         w = g[v][i].second; cost = g[v][i].first;
         if ( dist[v] + cost < dist[w] ) {
            if ( dist[w] != INF ) Q.erase( Q.find( ii( dist[w], w ) ) );
            dist[w] = dist[v] + cost; Q.insert( ii( dist[w], w ) );
         }
      }
   }
}
// Bellman Ford - O(mn) - Atualiza dist, retorna se ha ciclo de custo negativo
struct edge { int s, t, w; }; // origem, destino e custo
bool bellman_ford( const vector<edge>& edges, int N, int v, vector<int>& dist ){
   bool stop; int m = edges.sz;
   dist.assign( N, INF ); dist[v] = 0;
   fori(i,N) {
      stop = true;
      fori(j,m) if( dist[edges[j].s] + edges[j].w < dist[edges[j].t] ) {
         dist[edges[j].t] = dist[edges[j].s] + edges[j].w;
         stop = false;
      }
      if ( stop ) break;
   }
   // detecta ciclos negativos
   fori(i,m) if( dist[edges[i].s] + edges[i].w < dist[edges[i].t] ) return true;
   return false;
}
// Prim - O(n^2)
double mst_prim( const vector< vector<double> >& g ) {
   int N = g.sz, v = 0; double weight, distance, result = 0;
   vector<double> dist(N, INF); vector<bool> intree(N, false);
   dist[v] = 0;
   while ( !intree[v] ) {
      intree[v] = true;
      fori(i,N) if(g[v][i] != INF && !intree[i]) dist[i] = min(dist[i],g[v][i]);
      v = 0; distance = INF;
      forr(i,1,N-1) if ( !intree[i] && dist[i] < distance ) {
         distance = dist[i]; v = i;
      }
      if ( distance != INF ) result += distance;
   }
   return result;
}
// Kruskal - O(m log m)
struct edge {
   int s, t; double w; // origem, destino, custo
   bool operator<( const edge& e ) const { return cmp(w, e.w) < 0; }
};
double mst_kruskal( int N, vector< edge > & edges ) {
   int u, v, k; double result = 0; edge e;
   vector<int> pa(N), comp_sz(N, 1);
```

UFMG – Universidade Federal de Minas Gerais

```
   fori(i,N) pa[i] = i;
   sort(all(edges));
   k = 0; // numero de arestas da floresta em construcao
   for ( int i = 0; i < edges.sz && k < N - 1; ++i ) {
      e = edges[i];
      for ( u = e.s; u != pa[u]; u = pa[u] ); // pa[u] = pa[pa[u]];
      for ( v = e.t; v != pa[v]; v = pa[v] ); // pa[v] = pa[pa[v]];
      if ( u == v ) continue;
      if ( comp_sz[u] < comp_sz[v] ) { pa[u] = v; comp_sz[v] += comp_sz[u]; }
      else { pa[v] = u; comp_sz[u] += comp_sz[v]; }
      result += e.w; k++;
   }
   return result;
}
// Diametro de uma arvore - O(n). o caminho [fartest1, ..., fartest2] forma o
// diametro. central_node eh um vertice central da arvore
const int MAXN = 101;
int dist[MAXN], parent[MAXN], visited[MAXN], tree[MAXN][MAXN], cnt_edges[MAXN]
int n, fartest1, fartest2;
void dfs( int v, int p, int & fartest ) {
   visited[v] = 1; parent[v] = p;
   if ( p != -1 ) dist[v] = dist[p] + 1;
   if ( dist[v] > dist[fartest] ) fartest = v;
   fori(i,cnt_edges[v]) if (!visited[tree[v][i]]) dfs(tree[v][i], v, fartest);
}
int calc_diameter() {
   int half_diameter, central_node;
   fartest1 = fartest2 = 0;
   memset(parent,-1,n*sizeof(int)); memset(visited,0,n*sizeof(int));
   dist[0] = 0; dfs( 0, -1, fartest1 );
   memset(parent,-1,n*sizeof(int)); memset(visited,0,n*sizeof(int));
   dist[fartest1] = 0; dfs( fartest1, -1, fartest2 );
   half_diameter = dist[fartest2] / 2; central_node = fartest2;
   while( half_diameter-- ) central_node = parent[central_node];
   return central_node;
}
// Componentes fortemente conectados - Tarjan - O(n+m)
// ssc = nro. de componentes fortemente conectados do grafo direcionado g.
// components[i] = id do componente ao qual pertence o vertice i
const int MAXN = 3001;
vector< int > g[MAXN], S;
int indices[MAXN], lowlinks[MAXN], component[MAXN], indice, scc;
bool in_stack[MAXN];
void do_tarjan( int v ) {
   indices[v] = indice; lowlinks[v] = indice;
   ++indice; S.pb(v); in_stack[v] = true;
   fori(i,g[v].sz) {
      int w = g[v][i];
      if ( indices[w] == -1 ) {
         do_tarjan(w); lowlinks[v] = min( lowlinks[v], lowlinks[w] ); }
      else if ( in_stack[w] ) lowlinks[v]=min(lowlinks[v], indices[w]);
   }
   if ( lowlinks[v] == indices[v] ) {
      int w = S[S.sz-1]; S.pop_back();
      while ( w != v ) {
         in_stack[w] = false; component[w] = scc; w = S[S.sz-1]; S.pop_back(); }
      component[v] = scc++; in_stack[v] = false;
   }
}
void scc_tarjan( int n ) {
   fori(i,n) { indices[i] = lowlinks[i] = component[i] = -1; in_stack[i] = 0; }
   S.clear(); indice = scc = 0;
   fori(i,n) if( component[i] == -1 ) do_tarjan(i);
}
```

```
// 2−SAT − Cria−se um grafo de implicacao da seguinte forma: Vertices: um para
// cada variavel e sua negacao. Arestas: p/ uma disjuncao da forma (x0 v ~x3),
// cria−se duas arestas: (~x0 −> ~x3) e ( x3 −> x0 ). OBS: indexar vertices a
// partir de 1. Todos os vertices de um mesmo componente fortemente conectado ou
// sao todos verdadeiros ou todos falsos.
int N;
#define INDEX(i) i > 0 ? i : (2 * N + 1 + i)
while ( cin >> N >> M ) {
   vector< vector<int> > g(2*N+1);
   vector<int> component(2*N+1);
   fori(i,2*N+1) component[i] = −1;
   while ( M−− ) {
      cin >> u >> v;
      g[INDEX(−u)].pb(INDEX(v)); g[INDEX(−v)].pb(INDEX(u));
   }
   scc_tarjan( g, component, cnt_vertex );
   int sat = 1;
   forr(i,1,cnt_vertex)
      if ( component[INDEX(i)] == component[INDEX(−i)] ) { sat = 0; break; }
   cout << sat << endl;
}
// Pontes, pontos de articulacao e componentes biconectados
// ord_cnt = contador do numero de vertices visitados na arvore de DFS
// root_children_cnt = contador do numero de filhos da raiz
// ord[v] = ordem em que o vertice v foi visitado na DFS−tree em pre−ordem
// low[v] = menor numero em preordem do vertice da DFS−tree que pode ser
//          alcancado por uma sequencia de tree edges e uma back edge
// parent[v] = antecessor do vertice v na arvore de DFS
// bridges = lista com as pontes, art_pts = booleanos para pontos de articulacao
// biconnected_component[i] = arestas do componente biconectado i
// edges = pilha com as arestas exploradas (util para descobrir CB)
int ord_cnt, root_children_cnt;
vector< int > ord, low, parent, art_pts;
vector< pair< int, int > > bridges;
vector< vector< pair< int, int > > > biconnected_component;
stack< pair< int, int > > edges;
void dfs_bridges_art_pts( const vector< vector<int> >& g, int v ) {
   ord[v] = ord_cnt++; low[v] = ord[v];
   pair< int, int > null_edge = make_pair(v, v);
   fori(i,g[v].sz) {
      edges.push( null_edge );
      int w = g[v][i];
      if ( parent[v] != w ) {
         if ( ord[w] == −1 ) { // (v, w) e uma tree edge
            edges.push( make_pair(v, w) );
            parent[w] = v;
            dfs_bridges_art_pts(g, w);
            if ( parent[v] == v && parent[w] == v ) {
               vector< pair< int, int > > bc_edges;
               pair< int, int > e;
               while ( (e = edges.top()) != null_edge ) {
                  if ( e.first != e.second ) bc_edges.pb(e);
                  edges.pop();
               }
               biconnected_component.pb(bc_edges);
               root_children_cnt++;
               if ( root_children_cnt == 2 ) art_pts[v] = 1;
            }
            else if ( low[w] >= ord[v] ) {
               art_pts[v] = 1;
               vector< pair< int, int > > bc_edges;
               pair< int, int > e;
               while ( (e = edges.top()) != null_edge ) {
                  if ( e.first != e.second ) bc_edges.pb(e);
```

UFMG − Universidade Federal de Minas Gerais

```
                  edges.pop();
               }
               biconnected_component.pb(bc_edges);
            }
            low[v] = min( low[v], low[w] );
            if ( low[w] == ord[w] ) bridges.push_back( make_pair(v, w) );

         }
         else { // (v, w) e uma back edge (back link ou down link)
            low[v] = min( low[v], ord[w] );
            if ( ord[v] > ord[w] ) edges.push( make_pair(v, w) );
         }
      }
   }
}
void all_bridges_art_points( const vector< vector<int> >& g ) {
   int N = g.sz;
   ord.assign(N,−1); art_pts.assign(N,0); low.resize(N); parent.resize(N);
   bridges.clear(); biconnected_component.clear();
   ord_cnt = 0;
   fori(i,g.sz) if ( ord[i] == −1 ) {
      while ( !edges.empty() ) edges.pop();
      parent[i] = i; root_children_cnt = 0;
      dfs_bridges_art_pts(g, i);
   }
}
// Ordenacao topologica de um DAG − O(m+n)
vector<int> topsort(const vector< vector<int> >& g, vector<int> indegree) {
   int N = g.sz, v; vector<int> sorted; queue<int> zeroin;
   fori(i,N) if ( indegree[i] == 0 ) zeroin.push(i);
   while ( !zeroin.empty() ) {
      v = zeroin.front(); zeroin.pop(); sorted.pb(v);
      fori(i,N) if ( g[v][i] != INF ) {
         indegree[i]−−; if ( indegree[i] == 0 ) zeroin.push(i);
      }
   }
   return sorted;
}
// Circuito Euleriano − O(m+n)
void dfs2( vector< vector<int> >& g, int v, vector<int>& dfs_path ) {
   fori(i,g.sz) if(g[v][i] != INF) { g[v][i] = INF; dfs2(g, i, dfs_path); }
   dfs_path.pb(v);
}
void print_eulerian_circuit( const vector< vector<int> >& g ) {
   vector< vector< int > > g2 = g; vector< int > dfs_path;
   dfs2( g2, 0, dfs_path );
   fori(i,dfs_path.sz−1) cout << dfs_path[i] << ' ' << dfs_path[i+1] << endl;
}
// Maxflow − Ford−Fulkerson − O(mf) − Retorna o valor do fluxo maximo
// prev[v] − pai do vertice v no caminho encontrado do source ao sink
// res[] − copia da capacidade (eh alterada em cada chamada)
// g[i] − lista de adjacencias do vertice i (adicionar aresta nos dois sentidos)
// OBS: memset( cap, 0, sizeof( cap ) ); fori(i,n) g[i].clear();
const int TAM = 110;
int cap[TAM][TAM], res[TAM][TAM], prev[TAM];
vector<int> g[TAM];
int maxflow( int n, int source, int sink ) {
   int ans = 0, i;
   memcpy( res, cap, sizeof( res ) );
   while( true ) {
      memset( prev,−1, sizeof prev );
      prev[source] = −2;
      queue<int> q; q.push( source );
      while( !q.empty() && prev[sink] == −1 ) {
```

```
         int u = q.front(); q.pop();
         ford(i,g[u].sz-1,0) {
            int v = g[u][i];
            if( prev[v] == -1 && res[u][v] > 0 ) { prev[v] = u; q.push(v); }
         }
      }
      if( prev[sink] == -1 ) break;
      int bot = INF, len;
      for(i = sink; prev[i] >= 0; i = prev[i]) bot = min(bot, res[prev[i]][i]);
      for(i = sink; prev[i] >= 0; i = prev[i]) {
         res[prev[i]][i] -= bot; res[i][prev[i]] += bot;
      }
      ans += bot;
   }
   return ans;
}
// Vertex Cut (num. minimo de vertices que se removidos desconectam o grafo)
memset( cap, 0, sizeof( cap ) ); fori(i,2*n) g[i].clear();
// cria aresta do vertice original i para a sua copia e da copia para o original
fori(i,n) { cap[2*i][2*i+1] = 1; g[2*i].pb(2*i+1); g[2*i+1].pb(2*i); }
// cria arestas: (a,b'), (b',a), (a',b), (b,a')
fori(i,m) {
   cap[2*a+1][2*b] = INF; cap[2*b+1][2*a] = INF;
   g[2*a].pb(2*b+1); g[2*b+1].pb(2*a); g[2*a+1].pb(2*b); g[2*b].pb(2*a+1);
}
int mincut, asw = INF, s = 0;
forr(t,1,n-1) { min_cut = maxflow(2*n, 2*s+1, 2*t); asw = min(asw, mincut); }
if ( asw >= INF ) asw = n; // se o corte minimo for INF
// Cobertura minima por caminhos: dado um DAG, determinar o  menor numero de
// caminhos disjuntos em vertices que cobrem todos os vertices do digrafo
// Ideia: criar grafo bipartido e resolver o matching maximo nesse grafo. Para
// cada arco (a,b) da entrada, criar arco com capacidade 1 no grafo bipartido.
int min_cover_by_directed_paths() {
   // cria grafo
   int asw = maxflow(n, source, sink);
   return n - asw;
}
// Maxflow - Improved Shortest Augmenting Path Algorithm, O(n^2 m)
#define MAX 203
int n, m, source, sink; // numero de vertices, arestas, origem e destino
int G[MAX][MAX], F[MAX][MAX], g[MAX][MAX]; // capacidade, fluxo e grafo
int pi[MAX];  // predecessores
int CurrentNode[MAX]; // aresta atual para cada vertice
int queue[MAX];  // fila para a reverse BFS
int d[MAX];  // distancia
int numbs[MAX];  // numbs[k] eh o numero de vertices i com d[i]==k
int rev_BFS() {
   int i, j, head(0), tail(0);
   fori(i,n) numbs[ d[i] = n ]++;
   numbs[n]--; d[sink] = 0; numbs[0]++;
   queue[++tail] = sink;
   while( head != tail ) {
      i = queue[++head];
      fori(j,n) {
         if( d[j] < n || G[j][i] == 0 )  continue;
         queue[ ++tail ] = j; numbs[n]--; d[j] = d[i] + 1; numbs[d[j]]++;
      }
   }
   return 0;
}
int Augment() {
   int i, j, tmp, width(INF);
   for( i = sink, j = pi[i]; i != source; i = j, j = pi[j] ) {
      tmp = G[j][i]; if(tmp < width)  width = tmp;
```

UFMG – Universidade Federal de Minas Gerais

```
   }
   for( i = sink, j = pi[i]; i != source; i = j, j = pi[j] ) {
      G[j][i] -= width; F[j][i] += width;
      G[i][j] += width; F[i][j] -= width;
   }
   return width;
}
int Retreat( int &i ) {
   int tmp, j, mind(n-1);
   fori(j,n) if( G[i][j] > 0 && d[j] < mind ) mind = d[j];
   tmp = d[i]; numbs[d[i]]--; d[i] = 1 + mind; numbs[d[i]]++;
   if( i != source ) i = pi[i];
   return numbs[tmp];
}
int maxflow() {
   int flow(0), i = source, j;
   rev_BFS();
   fori(k,n) CurrentNode[k] = 0;
   while( d[source] < n ) {
      for(j = CurrentNode[i]; j < n; ++j) if(G[i][j] > 0 && d[i]==d[j]+1) break;
      if( j < n ) {
         CurrentNode[i] = j; pi[j] = i; i = j;
         if( i == sink ) { flow += Augment(); i = source; }
      }
      else {
         CurrentNode[i] = 0;
         if( Retreat(i) == 0 ) break;
      }
   }
   return flow;
}
// Min-cost Maxflow - Edmonds-Karp relabelling + Dijkstra
// Retorna: o fluxo, o custo na variavel fcost
// fnet armazena o fluxo da rede. fnet[u][v] e fnet[v][u] podem ser
//                                  positivos. neste caso, pegar a diferenca.
// NAO ESQUECER DO CLR(cap, 0); CLR(cost, INF); criar grafo a partir desses dois
#define NV 105
#define Pot(u,v) (d[u] + pi[u] - pi[v])
#define CLR(a, x) memset( a, x, sizeof( a ) )
int cap[NV][NV], cost[NV][NV]; // usados para criar a rede
int adj[NV][NV], fnet[NV][NV], deg[NV], par[NV], d[NV], pi[NV];
int minflow( int n, int s, int t, int &fcost ) {
   CLR( deg, 0 ); CLR( fnet, 0 ); CLR( pi, 0 );
   fori(i,n) fori(j,n)  if( cap[i][j] || cap[j][i] ) adj[i][deg[i]++] = j;
   int flow = fcost = 0;
   while( dijkstra( n, s, t ) ) {
      int bot = INF;
      for( int v = t, u = par[v]; v != s; u = par[v = u] )
         bot = min( bot, fnet[v][u] ? fnet[v][u] : ( cap[u][v]-fnet[u][v] ) );
      for( int v = t, u = par[v]; v != s; u = par[v = u] )
         if( fnet[v][u] ) { fnet[v][u] -= bot; fcost -= bot * cost[v][u]; }
      else { fnet[u][v] += bot; fcost += bot * cost[u][v]; }
      flow += bot;
   }
   return flow;
}
// para grafos DENSOS
bool dijkstra( int n, int s, int t ) {
   fori(i,n) d[i] = INF, par[i] = -1;
   d[s] = 0; par[s] = -n - 1;
   while( 1 ) {
      int u = -1, bestD = INF;
      fori(i,n) if( par[i] < 0 && d[i] < bestD ) bestD = d[u = i];
      if( bestD == INF ) break;
```

```
      par[u] = −par[u] − 1;
      fori(i,deg[u]) {
         int v = adj[u][i];
         if( par[v] >= 0 ) continue;
         if( fnet[v][u] && d[v] > Pot(u,v) − cost[v][u] ) {
            d[v] = Pot( u, v ) − cost[v][u]; par[v] = −u − 1;
         }
         if( fnet[u][v] < cap[u][v] && d[v] > Pot(u,v) + cost[u][v] ) {
            d[v] = Pot( u, v ) + cost[u][v]; par[v] = −u − 1;
         }
      }
   }
   fori(i,n) if( pi[i] < INF ) pi[i] += d[i];
   return par[t] >= 0;
}
// para grafos ESPARSOS
#define BUBL { \
    t = q[i]; q[i] = q[j]; q[j] = t; \
    t = inq[q[i]]; inq[q[i]] = inq[q[j]]; inq[q[j]] = t; }
int q[NV], inq[NV], qs;
bool dijkstra( int n, int s, int t ) {
   CLR( d, 0x3F ); CLR( par, −1 ); CLR( inq, −1 );
   d[s] = qs = 0; inq[q[qs++] = s] = 0; par[s] = n;
   while( qs ) {
      int u = q[0]; inq[u] = −1;
      q[0] = q[−−qs];
      if( qs ) inq[q[0]] = 0;
      for( int i = 0, j = 2*i + 1, t; j < qs; i = j, j = 2*i + 1 ) {
         if( j + 1 < qs && d[q[j + 1]] < d[q[j]] ) j++;
         if( d[q[j]] >= d[q[i]] ) break;
         BUBL;
      }
      for( int k = 0, v = adj[u][k]; k < deg[u]; v = adj[u][++k] ) {
         if( fnet[v][u] && d[v] > Pot(u,v) − cost[v][u] )
            d[v] = Pot(u,v) − cost[v][par[v] = u];
         if( fnet[u][v] < cap[u][v] && d[v] > Pot(u,v) + cost[u][v] )
            d[v] = Pot(u,v) + cost[par[v] = u][v];
         if( par[v] == u ) {
            if( inq[v] < 0 ) { inq[q[qs] = v] = qs; qs++; }
            for( int i = inq[v], j = ( i − 1 )/2, t;
                 d[q[i]] < d[q[j]]; i = j, j = ( i − 1 )/2 ) BUBL;
         }
      }
   }
   fori(i,n) if( pi[i] < INF ) pi[i] += d[i];
   return par[t] >= 0;
}
// Encontrar K caminhos disjuntos em um DAG que passem pelo maior no. de vert.
// Ideia: criar um source de capacidade K e duplicar cada vertice original.
cin >> n >> m;
CLR(cap, 0); CLR(cost, INF); CLR(adj, 0);
fori(i,m)
{
   cin >> a >> b; a−−; b−−;
   cap[2*a][2*a+1] = 1; cost[2*a][2*a+1] = 0;
   cap[2*b][2*b+1] = 1; cost[2*b][2*b+1] = 0;
   cap[2*a+1][2*b] = 1; cost[2*a+1][2*b] = −1;
}
int pre_src = 2*n, source = 2*n+1, sink = 2*n+2, asw = 0, maxflow;
cap[pre_src][source] = K; cost[pre_src][source] = 0;
fori(i,n) {
   cap[source][2*i] = 1; cost[source][2*i] = −1;
   cap[2*i+1][sink] = 1; cost[2*i+1][sink] = 0;
}
```

UFMG − Universidade Federal de Minas Gerais

```
maxflow = minflow( 2*n+3, pre_src, sink, asw );
cout << −asw << endl;
// Algoritmo Hungariano − O(n^3): matching maximo em um grafo bipartido valorado
// Na funcao main(): forr(i,1,N) forr(j,1,N) scanf( "%d",&w[i][j] );
// hungarian_algorithm(); int asw = 0; forr(i,1,N) asw += lx[i] + ly[i];
// p/ matching min, inicializar matriz w com custos neg. e imprimir −asw
const int MAXN = 501;
int N; // numero de vertices em cada particao (|X| = |Y| = N)
int lx[MAXN], ly[MAXN], w[MAXN][MAXN]; // lx[i] + ly[j] >= w[i][j]
int mark_T[MAXN], vizinhos_X[MAXN][MAXN], num_vizinhos_X[MAXN];
int saturadoX[MAXN], saturadoY[MAXN], paiX[MAXN], paiY[MAXN], match[MAXN];
vector <int> S, T;
int not_saturated() {
   forr(i,1,N) if( saturadoX[i] == 0 ) return i;
   return 0;
}
int choose( int *x_pai ) {
   fori(i,S.sz) fori(j,num_vizinhos_X[ S[i] ]) {
      if( !mark_T[ vizinhos_X[ S[i] ][j] ] ) {
         *x_pai = S[i];
         return vizinhos_X[ S[i] ][j];
      }
    }
    return −1;
}
int clean_and_create() {
   CLR( num_vizinhos_X, 0 ); CLR( mark_T, 0 ); CLR( saturadoX, 0 );
   CLR( saturadoY, 0 ); CLR( match, −1 ); CLR( paiX, −1 ); CLR( paiY, −1 );
   forr(i,1,N) forr(j,1,N) if( lx[i] + ly[j] == w[i][j] )
      vizinhos_X[i][ num_vizinhos_X[i]++ ] = j;
}
void start() {
   forr(i,1,N) {
      int k = −1;
      forr(j,1,N) if( w[i][j] > k ) k = w[i][j];
      ly[i] = 0; lx[i] = k; // maior valor da linha i
   }
   clean_and_create();
}
void compute() {
   int best = INF;
   fori(i,S.sz) forr(j,1,N) if( !mark_T[ j ] ) {
      if(lx[S[i]]+ly[j]−w[S[i]][j] < best) best = lx[S[i]] + ly[j] − w[S[i]][j];
   }
   fori(i,S.sz) lx[ S[i] ] −= best;
   fori(i,T.sz) ly[ T[i] ] += best;
   clean_and_create();
}
void hungarian_algorithm() {
   int i, u, y, z, x_pai;
   start();
   while( u = not_saturated() ) {
      S.clear(); T.clear(); S.push_back(u);
      CLR( mark_T, 0 );
      while(1) {
         // pega vertice em Y (viz. de alguem que esteja em S) q n pertenca a T
         y = choose( &x_pai );
         if( y == −1 ) { compute(); break; } // nao encontrou matching perfeito
         else {
            if( saturadoY[y] ) {
               S.push_back( match[y] ); T.push_back(y);
               mark_T[ y ] = 1; paiX[y] = x_pai; paiY[ match[y] ] = y;
            }
            else { // caminho de aumento
```

```
               paiX[y] = x_pai;
               int atual = y;
               while( atual != -1 ) {
                  match[ atual ] = paiX[atual]; saturadoY[atual] = 1;
                  saturadoX[ paiX[atual] ] = 1; atual = paiY[ paiX[atual] ] ;
               }
               break;
            }
         }
      }
   }
}
// Difference Constraints - Problema: Resolver um sistema linear com
// desigualdades da forma: x_i - x_j <= b_k, 1 <= i, j <= n. Solucao: Criar um
// grafo direcionado apropriado (grafo de restricoes) e encontrar o menor
// caminho de um vertice artificial v_{n+1} p/ todos os demais vertices.
// 1. Construcao do digrafo G:
// 1.1. x_i - x_j <= b_k -> arco (v_j, v_i) com custo b_k, 1 <= i, j <= n
// 1.2. introduzir vertice v{n+1} com arco (v_{n+1}, v_{i})=0, 1 <= i <= n
// 2. Achar caminho minimo a partir do vertice n+1 para todos os outros
//    vertices, por exemplo, com algoritmo de bellman-ford
// 3. Se G nao tem ciclo com custo negativo, entao x_{i} = dist[i]
//    Se G tem ciclo com custo negativo, entao o sistema eh inviavel
// Resolve também problemas com restricoes das formas:
// (1) x_a + x_{a+1} + ... + x_{b} <= A1, (2) x_a + x_{a+1} + ... + x_{b} >= A2
// (3) -x_a - x_{a+1} - ... - x_{b} >= A3, (4) -x_a - x_{a+1} - ...- x_{b} <= A3
// Seja S(k) = x_0 + ... + x_k; x_0 eh uma variavel artificial tal que x_0 = 0
// Com isso, x_a + x_{a+1} + ... + x_{b} = S(b) - S(a-1), e entao
//          -x_a - x_{a+1} - ... - x_{b} = S(a-1) - S(b)
// Portanto, eh possivel colocar (1), (2), (3) e (4) da forma s_i - s_j <= b_k.
// Se existe solucao, entao S(i) = dist[i] - dist[i-1]
vector<int> dist(n+2, INF); vector<edge> edges; edge e;
fori(i,m) {
   cin >> a >> b >> c >> d;
   if ( c == "lt" ) { e.s = a-1; e.t = a+b; e.w = d-1; }
   else { e.s = a+b; e.t = a-1; e.w = -(d+1); }
   edges.pb(e);
}
forr(i,0,n) { e.s = n+1; e.t = i; e.w = 0; edges.pb(e); }
bool negative_cycle = bellman_ford(edges, n+2, n+1, dist);
if ( negative_cycle ) printf("-1");
else forr(i,1,n) printf("%d", dist[i]-dist[i-1] );
// Min Vertex Cover em arvores - Encontrar o tamanho do menor subconjunto de
// vertices que cobre todos os outros vertices de uma arvore.
const int MAXN = 1510;
int visited[MAXN], p[MAXN], num_child[MAXN], leaves[MAXN], marks[MAXN], f, n;
int min_vertex_cover_in_trees() {
   CLR( visited, 0 );
   int k;
   fori(i,f) if( p[leaves[i]] != -1 && !visited[ p[leaves[i]] ] ) {
      visited[ p[leaves[i]] ] = 1;
      if( p[ p[leaves[i] ] ] != -1) num_child[ p[ p[leaves[i]] ] ]--;
      if( p[ p[leaves[i]] ] != -1 && !visited[p[ p[leaves[i]] ]] &&
          num_child[ p[ p[leaves[i]]]] == 0 && !marks[ p[ p[leaves[i]] ] ] ) {
            marks[ p[ p[leaves[i]] ] ] = 1;
            leaves[ f++ ] = p[ p[leaves[i]] ];
      }
   }
   int asw = 0;
   fori(i,n) if( visited[i] ) asw++;
   return asw;
}
int main() {
   // Entrada: n - numero de vertices
```

UFMG – Universidade Federal de Minas Gerais

```
   // id:(q) v_1 ... v_q - id do vertice, numero de vizinhos e lista de vizinhos
   while( scanf( "%d",&n ) == 1 ) {
      int k, child, tam;
      CLR( p, -1 ); CLR( num_child, 0 ); CLR( marks, 0 );
      f = 0;
      fori(i,n) {
         scanf( "%d:(%d)", &k, &tam );
         if( tam == 0 ) { leaves[f++] = k; marks[k] = 1; }
         num_child[k] = tam;
         fori(j,tam) { scanf( "%d", &child ); p[child] = k; }
      }
      int asw = min_vertex_cover_in_trees();
      printf( "%d\n", asw );
   }
   return 0;
}
// Permanente da matriz quadrada n x n: perm(M) = (-1)^n *
// \sum_{S \subseteq {1,...,n}} (-1)^|S| \prod_{i = 1}^n \sum_{j \in S} a_{ij}
// Aplicações: (1) cycle covers em digrafos valorados - permanente igual a soma
// dos pesos de todos as coberturas por ciclo do drigrafo. (2) perfect matching
// em grafos bipartidos valorados - permanente igual a soma dos pesos de todos
// os matchings perfeitos do grafo. Em grafos não valorados, o permanente é
// igual ao número de cycle covers ou de matchings perfeitos.
// Complexidade: O(2^n * n) - usando codigo de gray.
int graycode[1 << MAXN]; // Gera o codigo gray
void gen_gray_code( int nbits ) {
   int toggled = 0; graycode[0] = 0;
   for( int n = 1; n < (1 << nbits); ++n ) {
      toggled ^= n & ~(n-1);
      graycode[n] = toggled;
   }
}
const int MAXN = 20;
// line_sum[i]: soma dos elementos da i-esima linha da matrix
int N, matrix[MAXN][MAXN], line_sum[MAXN];
long long calc_permanent() {
   int set, last_set, ub = 1 << N, tam, diff, index;
   long long product, sum, permanent = 0;
   fori(i,MAXN) line_sum[i] = 0;
   forr(j,1,ub-1) {
      last_set = graycode[j-1]; set = graycode[j]; tam= __builtin_popcount(set);
      diff = set ^ last_set; // diferenca entre set e last_set
      index = __builtin_ctz( diff ); // indice do bit alterado
      product = 1;
      fori(i,N) {
         sum = line_sum[i];
         if ( set & ( 1 << index ) ) sum += matrix[i][index];
         else sum -= matrix[i][index];
         line_sum[i] = sum; product *= sum;
      }
      permanent += ( tam % 2 ) ? -product : product;
   }
   return ( N % 2 ) ? -permanent : permanent;
}
// Stable marriage problem
int men[MAX][MAX], women[MAX][MAX];
int order[MAX][MAX];
void mry(int men[][],int women[][],int n,vector<int> &ans,vector<int> &ans_rev){
   vector <int> proposals(n,0);
   stack <int> singles;
   int i, j, k;
   for (i=0; i < n; i++) for (j=0; j < n; j++) order[i][women[i][j]] = j;
   ans = vector <int>(n,-1), ans_rev = vector <int>(n,-1);
   for (i=0; i < n; i++) singles.push(i);
```

```
   while (!singles.empty()) {
      i=singles.top(); singles.pop(); j=men[i][proposals[i]++]; k=ans_rev[j];
      if (k == -1) ans[i] = j, ans_rev[j] = i;
      else if (order[j][i] < order[j][k]) {
         ans[k] = -1; ans[i] =  j; ans_rev[j] = i; singles.push(k);
      } else singles.push(i);
   }
}
// Matching em grafo bipartido -> Inicializar nl e nr.
int nr, nl, mate[MAX];
char visited[MAX];
int dfs(int i) {
   int j; if (visited[i]) return 0; visited[i] = 1;
   for (j=0; j < nr; j++) {
      if (g[i][j] && (mate[j] == -1 || dfs(mate[j]))) {mate[j] = i; return 1;}
   }
   return 0;
}
int match() {
   int i, ans = 0;
   memset(mate,-1,sizeof(mate));
   for (i=0; i < nl; i++) memset(visited,0,sizeof(visited)), ans += dfs(i);
   return ans;
}
// Min-cut entre quaisquer vertices
int stoer_wagner(Graph g, int n) {
   int order[MAX], used[MAX], merged[MAX], i, j, k, p, cut, ans = INF;
   Graph aux;
   memset(merged,0,sizeof(merged));
   for (p=1; p < n; p++) {
      memset(used,0,sizeof(used));
      memcpy(aux,g,sizeof(Graph));
      used[order[0] = 0] = 1;
      for (k=1; k < n; k++) {
         for (i=-1,j=1; j < n; j++) {
            if(!used[j] && !merged[j] && (i == -1 || aux[0][j] > aux[0][i]))
               i = j;
         }
         if (i == -1) break;
         used[order[k] = i] = 1;
         for (j=1; j < n; j++) {
            aux[0][j] += aux[i][j]; aux[j][0] += aux[i][j];
            aux[i][j] = aux[j][i] = 0;
         }
      }
      for (i=cut=0; i < n; i++) {
         if (order[k-2] != i) {
            g[order[k-2]][i] += g[order[k-1]][i];
            g[i][order[k-2]] += g[order[k-1]][i];
         }
         cut += g[order[k-1]][i];
         g[order[k-1]][i] = g[i][order[k-1]] = 0;
      }
      merged[order[k-1]] = 1;
      if (cut < ans) ans = cut;
   }
   return ans;
}
// Matching em grafos quaisquer - Edmonds
// IMPORTANTE: na main inicializar n e graph (1-based)
const int MAXN = 256;
int n, start, finish, new_base, match[MAXN], father[MAXN], base[MAXN];
int graph[MAXN][MAXN], in_queue[MAXN], in_path[MAXN], in_blossom[MAXN];
queue< int > q;
```

UFMG – Universidade Federal de Minas Gerais

```
int find_common_ancestor(int u, int v) {
    memset(in_path, 0, sizeof(in_path));
    while(1) {u=base[u]; in_path[u]=1; if (u==start) break; u=father[match[u]];}
    while(1) {v=base[v]; if (in_path[v]) break; v = father[match[v]]; }
    return v;
}
void reset_trace(int u) {
   while (base[u] != new_base) {
      int v = match[u]; in_blossom[base[u]] = 1; in_blossom[base[v]] = 1;
      u = father[v];
      if (base[u] != new_base) father[u] = v;
   }
}
void contract(int u, int v) {
   new_base = find_common_ancestor(u, v);
   memset(in_blossom, 0, sizeof(in_blossom));
   reset_trace(u); reset_trace(v);
   if (base[u] != new_base) father[u] = v;
   if (base[v] != new_base) father[v] = u;
   forr(i,1,n) if (in_blossom[base[i]]) {
      base[i] = new_base; if (!in_queue[i]) q.push(i);
   }
}
void find_augmenting_path() {
   memset(in_queue, 0, sizeof(in_queue));
   memset(father, 0, sizeof(father));
   forr(i,1,n) base[i] = i;
   finish = 0;
   while( !q.empty() ) q.pop(); q.push(start); in_queue[start] = 1;
   while ( !q.empty() ) {
      int u = q.front(); q.pop();
      forr(v,1,n) if ((graph[u][v]) && (base[u] != base[v]) && (match[u] != v)){
         if ((v==start)||((match[v]>0)&&(father[match[v]]>0))) contract(u,v);
         else if (father[v] == 0) {
            father[v] = u;
            if (match[v] > 0) q.push(match[v]);
            else { finish = v; return; }
         }
      }
   }
}
void augment_path() {
    int u = finish, v, w;
    while (u > 0){v=father[u]; w = match[v]; match[v] = u; match[u] = v; u = w;}
}
int edmonds() {
   memset(match,0,sizeof(match));
   forr(i,1,n) if (match[i] == 0) {
      start = i; find_augmenting_path();
      if (finish > 0) augment_path();
   }
   int r = 0;
   forr(i,1,n) if ( match[i] > 0 ) r++;
   return r / 2;
}
void print() { forr(i,1,n) if ( i < match[i] ) printf("%d %d\n", i, match[i]); }

/// *********************** TEORIA DOS NÚMEROS ************************** ///
// mdc(x,y)
int gcd(int x, int y) { return y ? gcd(y, x % y) : abs(x); }
// mmc(x,y) = mmc(x,y) = (x / mdc(x,y)) * y
long long lcm(int x, int y) {
    if (x && y) return abs(x) / gcd(x, y) * (long long) abs(y);
    else return (long long) abs(x | y);
```

```
}
// Determina x e y tais que ax + by == gcd(a, b)
typedef pair<int, int> bezout;
bezout find_bezout(int a, int b) {
    if (b == 0) return bezout(1, 0);
    bezout u = find_bezout(b, a % b);
    return bezout(u.second, u.first − (a/b) * u.second);
}
// Determina x e y tais que ax + by == c. (x, y) eh uma solucao particular para
// o problema. As demais solucoes sao da forma: (x + bk, y − ak), para todo
// inteiro k, positivo ou negativo. Retorna (INF,INF) se nao tiver solucao.
bezout solve_linear_diophantine(int a, int b, int c) {
   int d = gcd(a, b);
   if ( c % d ) return bezout(INF, INF);
   int new_c = c / d;
   bezout asw = find_bezout( a / d, b / d );
   asw.first *= new_c; asw.second *= new_c;
   return asw;
}
// Determina x tal que a * x conguente 1 (mod m) pelo teorema de Fermat
long long inv_mult( long long n, long long mod ) {
   return mod_exp( n, mod−2, mod );
}
// Acha a menor solucao nao−negativa de a * x = b (mod m)
// Retorna −1 se a congruencia for impossivel.
int mod(int x, int m) { return ( x % m ) + ( (x < 0) ? m : 0 ); }
int solve_mod(int a, int b, int m) {
    if (m < 0) return solve_mod(a, b, −m);
    if (a < 0 || a >= m || b < 0 || b >= m)
      return solve_mod(mod(a, m), mod(b, m), m);
    bezout t = find_bezout(a, m);
    int d = t.first * a + t.second * m;
    if (b % d) return −1;
    else return mod(t.first * (b / d), m / gcd(a, m));
    // while ( ... ) x += m / gcd(a, m); // outras solucoes da equacao
}
// Calcula o valor de x em a * x = b mod c. Nao eh necessario que c seja primo.
// Testar a condicao a seguir para saber se a equacao tem solucao:
// if ( ( (a*x) % c + c ) % c != ( b % c + c ) % c )
// outras solucoes: while ( ... ) x += c / gcd(a, c);
long long solve_mod(long long a, long long b, long long c) {
   return a ? (solve_mod(c % a, (a − b % a) % a, a) * c + b) / a : 0;
}
// C(n,k) modulo m. Necessario que gcd(fat[n−k], m) = gcd(fat[k], m) = 1
typedef long long int lli;
int fatorial[MAXN+1]; // pre−calcular fatorial[i] mod m
int calc_combination( int n, int k, int m ) {
   int a = fatorial[n];
   int b = inv_mult(fatorial[n−k], m), c = inv_mult(fatorial[k], m);
   return ( ( ( (lli)a * (lli)b ) % m ) * (lli)c ) % m;
}
// C(n,k) modulo m
long long binom( int n, int k, int m ) {
   int mdc, num[k], den[k]; long long resp = 1;
   fori(i,k) { num[i] = n − i; den[i] = i + 1; }
   fori(i,k) fori(j,k) {
      mdc = gcd( den[i], num[j] );
      den[i] /= mdc; num[j] /= mdc;
   }
   fori(i,k) { resp *= num[i]; resp %= m; }
   return resp;
}
// b^e % m
long long mod_exp(int b, int e, int m) {
   long long res = 1;
   while ( e > 0 ) {
      if ( e & 1 ) res = (res * b) % m; e >>= 1; b = ( (long long) b * b) % m; }
   return res;
}
// Calcula o menor valor de e na expressao b^e = n % p
// Baby−step Giant−step algorithm. Retorna −1 se eh impossivel.
long long discrete_logarithm( long long b, long long n, long long p ) {
   if ( n == 1 ) return 0;
   map < long long, int > table;
   long long m = sqrt(p) + 1, pot = 1, pot2 = 1;
   fori(j,m) {
      if ( pot == n ) return j;
      table[( n * inv_mult( pot, p ) ) % p] = j;
      pot = ( pot * b ) % p;
   }
   fori(i,m) {
      if ( table.find( pot2 ) != table.end() ) return i * m + table[pot2];
      pot2 = ( pot * pot2 ) % p;
   }
   return −1;
}
// Calcula o legendre symbol (a|p) − p eh um primo impar
// (a|p) =  0 se a = 0 (mod p)
// (a|p) = +1 se a \neq 0 (mod p) e existe x, tal que x^2 = a (mod p)
// (a|p) = −1 se a \neq 0 (mod p) e nao existe x, tal que x^2 = a (mod p)
// (a|p) = a^{(p−1)/2} (mod p)
int legendre_symbol( long long a, long long p ) {
   long long signal = 1, asw;
   if ( a < 0 ) { signal = ( p % 4 == 1 ) ? 1 : −1; a = −a; }
   a %= p; asw = mod_exp(a,(p−1)/2,p);
   if ( asw == p−1 ) asw = −1;
   return signal * asw;
}
// Retorna a fatoracao em numero primos de abs(n)
typedef map<int, int> prime_map;
void divide(prime_map & M, int & n, int p) { while(n%p==0) { M[p]++; n /= p; } }
prime_map factoring(int n) {
    prime_map M;
    if (n < 0) return factoring(−n);
    if (n < 2) return M;
    divide(M, n, 2); divide(M, n, 3);
    int maxP = (int) sqrt(n) + 2;
    for ( int p = 5; p < maxP; p += 6 ) { divide(M, n, p); divide(M, n, p+2); }
    if (n > 1) M[n]++;
    return M;
}
// Decide se o inteiro n eh primo
bool is_prime(int n)
{
    if (n < 0) return is_prime(−n);
    if (n < 5 || n % 2 == 0 || n % 3 == 0) return (n == 2 || n == 3);
    int maxP = sqrt(n) + 2;
    for (int p = 5; p < maxP; p += 6)
        if (n % p == 0 || n % (p+2) == 0) return false;
    return true;
}
// mdc(x!,y)
int fact_gcd(int x, int y) {
   if ( y <= x ) return y;
   int asw = 1;
   prime_map M = factoring(y);
   for ( prime_map::iterator it = M.begin(); it != M.end(); ++it ) {
      int p = it−>first, n = x;
```

```
      int d1 = it->second; // d1 = qtas vezes p aparece como fator de y
      int d2 = 0; // d2 = qtas vezes p aparece como fator de x!
      while( n > 0 ) d2 += (n /= p);
      for ( int i = 0; i < min(d1, d2); i++ ) asw *= p;
   }
   return asw;
}
// Retorna os divisores do inteiro n (divisores pode conter numeros repetidos)
vector<int> divisors( int n ) {
   int maxP = (int) sqrt(n) + 2;
   vector<int> divisores;
   forr(i,1,maxP) if ( n % i == 0 ) { divisores.pb( i ); divisores.pb( n/i ); }
   return divisores;
}
// Retorna o nro. de divisores de cada inteiro no intervalo [1..abs(maxv)]
vector<int> number_of_divisors( int maxn ) {
    if ( maxn < 0 ) return divisors(-maxn);
    vector<int> v(maxn+1);
    forr(i,1,maxn) for ( int k = i; k <= maxn; k += i ) v[k]++;
    return v;
}
// Crivo otimizado
const int MAXN = 10000001, SIZE = MAXN/16+1;
const int MAX_PRIMES = 685000; // 1.26 * MAXN / log(MAXN);
char mark[SIZE]; // ( mark[n>>4]&(1<<((n>>1)&7)) ) == 0 se 2*n+1 eh primo
char is_prime[MAXN];
int primes[MAX_PRIMES], cnt_primes;
void sieve( int N ) {
   int i, j;
   for ( i = 3; i*i <= N; i += 2 ) if ( ( mark[i>>4] & (1<<((i>>1)&7)) ) == 0 )
      for ( j = i*i; j <= N; j += i<<1 ) mark[j>>4] |= ( 1<<((j>>1)&7) );
   cnt_primes = 0;
   primes[cnt_primes++] = 2;
   for ( i = 3; i <= N; i += 2 ) if ( (mark[i>>4]&(1<<((i>>1)&7))) == 0 )
      primes[cnt_primes++] = i;
   fori(i,cnt_primes) is_prime[primes[i]] = 1;
}
// Triangulo de Pascal
const int TAM = 50;
long long C[TAM][TAM];
void calc_pascal() {
    memset( C, 0, sizeof( C ) );
    fori(i,TAM) {
        C[i][0] = C[i][i] = 1;
        forr(j,1,i-1) C[i][j] = C[i-1][j-1] + C[i-1][j];
    }
}
// Teste de primalidade com Miller-Rabin Pollard-Rho
typedef long long int Int;
#define MAX ( (Int) 1 << 63 )-1
#define gcc 10007
Int p[10] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
Int Gcd(Int x, Int y) { return y ? Gcd(y, x % y) : x; }
inline Int produc_mod(Int a, Int b, Int mod) {
   Int sum = 0;
   while(b) { if(b & 1) sum = (sum + a) % mod; a = (a + a) % mod; b /= 2; }
   return sum;
}
inline Int power_mod(Int a, Int b, Int mod) {
   Int sum = 1;
   while(b) {
      if(b & 1) sum = produc_mod(sum, a, mod);
      a = produc_mod(a, a, mod); b /= 2;
   }
```

UFMG – Universidade Federal de Minas Gerais

```
   return sum;
}
bool rabin_miller(Int n) {
   int i, j, k = 0; Int u, m, buf;
   if(n == 2) return true;
   if(n < 2 || !(n & 1)) return false;
   m = n-1;
   while(!(m & 1)) { k++; m /= 2; }
   for(i = 0; i < 9; i++) {
      if(p[i] >= n) return true;
      u = power_mod(p[i], m, n);
      if(u == 1) continue;
      for(j = 0; j < k; j++) {
         buf = produc_mod(u, u, n);
         if(buf == 1 && u != 1 && u != n-1) return false;
         u = buf;
      }
      if(u-1) return false;
   }
   return true;
}
Int pollard_rho(Int n) {
   while(1) {
      int i = 1;
      Int x = rand() % (n-1) + 1, y = x, k = 2, d;
      do {
         i++; d = Gcd(n + y - x, n);
         if(d > 1 && d < n) return d;
         if(i == k) y = x, k *= 2;
         x = ( produc_mod(x, x, n) + n - gcc ) % n;
      } while(y != x);
   }
}
Int prime[10000];
int top;
void prime_divisor(Int key) {
   if( rabin_miller(key) ) prime[top++] = key;
   else {
      Int buf = pollard_rho(key);
      if( rabin_miller(buf) ) prime[top++] = buf;
      else prime_divisor(buf);
      if( rabin_miller(key / buf) ) prime[top++] = key / buf;
      else prime_divisor(key / buf);
   }
}
// Determina se n pode ser escrito como a soma de quadrados perfeitos
int main() {
    int t, i;
    Int n;
    scanf("%d", &t);
    while(t--) {
        scanf("%lld", &n);
        if( n < 0 ) printf("NO\n");
        else if ( n == 0 || n == 1 ) printf("YES\n");
        else {
            top = 0; prime_divisor(n);
            for(i = 0; i < top; ++i) if(prime[i] != 2 && (prime[i]-1)%4) break;
            if(i < top) printf("NO\n"); else printf("YES\n");
        }
    }
}
// phi
int phi[MAX+1];
void totient() {
```

```
   int i, j;
   for (i=1; i <= MAX; i++) phi[i] = i;
   for (i=2; i <= MAX; i+=2) phi[i] >>= 1;
   for (j=3; j <= MAX; j+=2) {
      if (phi[j] == j) {
         phi[j]--; for (i=2*j; i <= MAX; i+=j) phi[i] = phi[i] / j * (j - 1); }
   }
}
// (a * b) % MOD sem overflow. ATENCAO: mod < 2^62
uint64 mult_mod(uint64 a, uint64 b, uint64 MOD) {
   uint64 y = (float64)a*(float64)b/MOD + (float64)1/2;
   y *= MOD; uint64 x = a * b; uint64 r = x - y;
   if ((long long)r < 0) r += MOD, y--;
   return r;
}
// qtd. de nros. <= n que sao multiplos de algum elemento de v. V1: genérica.
// v = vetor, index = tamanho do vetor, n = valor a ser testado, LCM = 1
long long count_v1(vector <long long> &v, int index,long long n,long long LCM) {
   long long req, ans = 0;
   for (int i=0; i < index; ++i) {
      req = lcm(LCM,v[i]); if (req <= n) ans += n / req - cnt(v,i,n,req);
   }
   return ans;
}
// V2: mais rápida, só funciona quando gcd(v[i],v[j]) == 1, para qualquer i != j
int count_v2(int v[], int index, int n) {
   int ans = 0, i;
   for (i=0; i < index && v[i] <= n; i++) ans += n / v[i] - count(v,i,n/v[i]);
   return ans;
}

/// ***************************** MATRÓIDES ****************************** ///
const int N = 10010; // numero de elementos do conjunto base (arestas)
struct UnionFind { // estrutura union find para matroid grafico (florestas)
   int p[N], comp_sz[N];
   void clear( int m ){
      memset( p, -1, m * sizeof( int ) );
      memset( comp_sz, 0, m * sizeof( int ) );
   }
   int find( int x ) {
      while( p[x] != -1 ) x = p[x];
      return x;
   }
   void union_set( int x, int y ) {
      int u = find(x), v = find(y);
      if( u != v ) {
         if ( comp_sz[u] < comp_sz[v] ) { p[u] = v; comp_sz[v] += comp_sz[u]; }
         else { p[v] = u; comp_sz[u] += comp_sz[v]; }
      }
   }
};
UnionFind ufs;
// M1 = (E,I1): matroide grafico (arvores geradoras)
// M2 = (E,I2): matroide de particao - cores (cor i pode aparecer no maximo
//              maxcor[i] vezes)
// Algoritmo: Intersecao de dois matroides para encontrar o subconjunto
//            independente maximal de I1 \cap I2
int n, m, K; // vertices, arestas, cores
int x[N], y[N]; // i-esima aresta = (x[i],y[i])
int maxcor[N]; // maxcor[i] - numero de vezes que a cor i pode ainda ser
               // utilizada
int cor[N]; // cor[i] - cor da i-esima aresta
vector<int> g[N]; // grafo auxiliar
bool base[N]; // base[i] - se a i-esima aresta pertence ao conjunto
```

UFMG – Universidade Federal de Minas Gerais

```
              // dependente maximal de I1 \cap I2
bool X1[N], X2[N]; // X1 = {x \in S \ I : {I + x} \in I1},
                   // X2 = {x \in S \ I : {I + x} \in I2}
int prev[N]; // para bfs que encontra caminho minimo em g de X1 para X2
bool visited[N];
// no caso de achar a intersecao de matroides valorados, trocar bfs por
// caminho de custo minimo, e no caso de empate, escolher aquele que
// visita o menor numero de vertices
int bfs() {
   fori(i,m) if( X1[i] && X2[i] ) return i;
   memset( prev, -1, m * sizeof( int ) );
   memset( visited, false, m * sizeof( bool ) );
   queue<int> Q;
   fori(i,m) if( X1[i] ) { Q.push(i); visited[i] = true; }
   while( !Q.empty() ) {
      int v = Q.front(); Q.pop();
      fori(j,g[v].sz) {
         int u = g[v][j];
         if( visited[u] ) continue;
         prev[u] = v;
         if( X2[u] ) return u;
         Q.push(u); visited[u] = true;
      }
   }
   return -1;
}
int matroid_intersection() {
   memset( base, false, m * sizeof( bool ) );
   int res;
   for( res = 0; true; res++ ) {
      fori(i,m) g[i].clear(); // monta grafo auxiliar
      fori(i,1) if( base[i] ) { // fori(i,m), i -> y e j -> x
         ufs.clear(n); // arcos do matroide I1
         fori(j,m) if( base[j] && i != j ) ufs.union_set( x[j], y[j] );
         fori(j,m) if(!base[j] && ufs.find(x[j]) != ufs.find(y[j])) g[i].pb(j);
         fori(j,m) if( !base[j] ) { // arcos do matroide I2
            maxcor[cor[i]]++;
            if( maxcor[cor[j]] >= 1 ) g[j].pb(i);
            maxcor[cor[i]]--;
         }
      }
      // inicializa estrutura union find (vertices de arestas da base devem
      // pertencer ao mesmo componente)
      ufs.clear(n);
      fori(i,m) if( base[i] ) ufs.union_set( x[i], y[i] );
      fori(i,m) { // inicializa X1 e X2
         X1[i] = X2[i] = false;
         if( !base[i] ) {
            // se a aresta i ainda pode ser utilizada (nao forma ciclo com
            // as arestas da base), entra em X1
            if( ufs.find(x[i]) != ufs.find(y[i]) ) X1[i] = true;
            // se a cor[i] ainda pode ser utilizada, entra em X2
            if( maxcor[cor[i]] >= 1 ) X2[i] = true;
         }
      }
      // menor caminho no grafo g de X1 para X2
      int v = bfs(); // v eh o vertice de X2 que foi alcancado primeiro
      if( v == -1 ) break; // encontrou subc. independente maximal de I1 \cap I2
      while ( true ) { // atualiza elementos da base
         maxcor[cor[v]] += ( base[v] ? 1 : -1 ); base[v] = !base[v];
         if( X1[v] ) break; v = prev[v];
      }
   }
   return res;
```

```
}
int main() {
   int cas = 1;
   while ( scanf( "%d %d %d", &n, &m, &K ) == 3 ) {
      fori(i,K) maxcor[i] = 1; // cada empresa so pode ser respons. por 1 aresta
      fori(i,m) {
         scanf( "%d %d %d", &x[i], &y[i], &cor[i] );
         x[i]--; y[i]--; cor[i]--;
      }
      int res = matroid_intersection();
      printf( "Instancia %d\n%s\n\n",cas++,res == n-1 ? "sim" : "nao");
   }
   return 0;
}

/// ************************ ESTRUTURAS DE DADOS ************************ ///
// LCA: lowest common ancestor
const int MAXN = 1010; // numero maximo de vertices
int N; // numero de vertices
int parent[MAXN], level[MAXN];
vector< vector< int > > graph; vector< int > in_degree, visited;
void determine_level( int v, int lev ) { // O(N)
   visited[v] = true;
   level[v] = lev;
   fori(i,graph[v].sz) if ( !visited[graph[v][i]] )
      determine_level( graph[v][i], lev + 1 );
}
// implementacao < O(N log N), O(log N) >
const int LOGMAXN = 12;
int P[MAXN][LOGMAXN]; // P[i][j] - (2^j)-esimo ancestral de i
void pre_process() { // O(N log N)
   fori(i,N) fori(j,(1<<j<N)) P[i][j] = -1;
   fori(i,N) P[i][0] = parent[i];
   for ( int j = 1; 1 << j < N; ++j ) fori(i,N) if ( P[i][j - 1] != -1 )
      P[i][j] = P[P[i][j - 1]][j - 1];
}
int query( int p, int q ) { // O(log N)
   int tmp, log;
   if ( level[p] < level[q] ) tmp = p, p = q, q = tmp;
   for ( log = 1; 1 << log <= level[p]; log++ );
   log--;
   ford(i,log,0) if ( level[p] - (1 << i) >= level[q] ) p = P[p][i];
   if ( p == q ) return p;
   ford(i,log,0) if ( P[p][i] != -1 && P[p][i] != P[q][i] )
      p = P[p][i], q = P[q][i];
   return parent[p];
}
int main() {
   // ... computa in_degree, graph e parent
   int root;
   fori(i,N) if( in_degree[i] == 0 ) { root = i; break; }
   determine_level( root, 0 );
   visited.assign( N, 0 );
   pre_process();
}
// implementacao < O(N), O(sqrt(N)) >
int P[MAXN]; // P[i] - pai do v. i q esta situado no ult. niv. da secao anterior
void dfs( int v, int nr ) { // O(N)
   visited[v] = 1;
   if ( level[v] < nr ) P[v] = 1;
   else {
      if( !( level[v] % nr ) ) P[v] = parent[v];
      else P[v] = P[parent[v]];
   }
```



```
   fori(i,graph[v].sz) if (!visited[graph[v][i]]) dfs(graph[v][i], nr);
}
int query( int x, int y ) { // O(sqrt(N))
   while ( P[x] != P[y] ) {
      if ( level[x] > level[y] ) x = P[x];
      else y = P[y];
   }
   while ( x != y ) {
      if ( level[x] > level[y] ) x = parent[x];
      else y = parent[y];
   }
   return x;
}
int main() {
   // ... computa in_degree, graph e parent
   int root, height = -1;
   fori(i,N) if( in_degree[i] == 0 ) { root = i; break; }
   determine_level( root, 0 );
   fori(i,N) height = max( height, level[i] );
   visited.assign( N, 0 );
   dfs( root, height );
}
// RMQ: Range Minimum Query - Arvore de segmentos - <O(N),O(log N)>
const int MAXN = 100010; // nro. maximo de elementos da sequencia original
const int MAXIND = 300000; // 2 * 2^[logN + 1] + 1
int N; // numero de elementos do vetor A
int M[MAXIND];// M[i]-ind. em A do menor valor do intervalo assoc. ao no i
int A[MAXN]; // A[i] - i-esimo elemento da sequencia original
// pair<int, int> intervals[MAXIND]; // intervalo associado a cada no
void initialize( int node, int b, int e ) { // O(N) - node = 1, b = 0, e = N-1
   //intervals[node] = make_pair( b, e );
   if ( b == e ) M[node] = b;
   else {
      int m = ( b + e ) / 2;
      initialize( 2 * node, b, m );
      initialize( 2 * node + 1, m + 1, e );
      if ( A[M[2 * node]] <= A[M[2 * node + 1]] ) M[node] = M[2 * node];
      else M[node] = M[2 * node + 1];
   }
}
// O(log N) - node = 1, b = 0, e = N-1
int query( int node, int b, int e, int i, int j ) {
   if ( i > e || j < b ) return -1;
   if ( b >= i && e <= j ) return M[node];
   int m = ( b + e ) / 2;
   int p1 = query( 2 * node, b, m, i, j );
   int p2 = query( 2 * node + 1, m + 1, e, i, j );
   if ( p1 == -1 ) return p2; if ( p2 == -1 ) return p1;
   if ( A[p1] < A[p2] ) return p1;
   return p2;
}
// O(log N) - node = 1, b = 0, e = N-1, k = posicao que foi alterada
// OBS: alterar valor de A[k] fora dessa funcao
void update( int node, int b, int e, int k ) {
   if ( k > e || k < b ) return; // faz a funcao ser O(log N)
   if ( b == e ) M[node] = k;
   else {
      int m = ( b + e ) / 2;
      update( 2 * node, b, m, k );
      update( 2 * node + 1, m + 1, e, k );
      if ( A[M[2 * node]] <= A[M[2 * node + 1]] ) M[node] = M[2 * node];
      else M[node] = M[2 * node + 1];
   }
}
```

```
int main() {
   int Q, a, b;
   scanf( "%d %d", &N, &Q );
   memset( M, -1, sizeof( M ) );
   fori(i,N) scanf( "%d", &A[i] );
   initialize( 1, 0, N-1 );
   fori(i,Q) {
      scanf( "%d %d", &a, &b );
      a--; b--;
      printf( "%d\n", A[query( 1, 0, N-1, a, b )] );
   }
   return 0;
}
// Aplicacoes de RMQ:
// 1 - Determinar a maior area dada pelo produto do tamanho do segmento entre i
// e j e o menor valor contido nesse segmento. Solucao: divisao e conquista
long long greater_area( int i, int j )
{
   if ( i > j ) return -1;
   if ( i == j ) return A[i];
   int index = query( 1, 0, N-1, i, j );
   long long total_area = (long long) A[index] * ( j - i + 1 );
   long long left_area = greater_area( i, index - 1 );
   long long right_area = greater_area( index + 1, j );
   return max( total_area, max( left_area, right_area ) );
}
// 2 - Determinar o valor mais frequente de um intervalo ordenado.
// Solucao: transformar o problema em RMQ (maximum).
int v[MAXN]; // entrada do problema
int inicio[MAXN]; // inicio[i] - indice em que inicia o valor v[i]
int fim[MAXN]; // fim[i] - indice em que termina o valor v[i]
// transforma o problema de determinar o valor mais frequente do vetor ordenado
// v em um problema de RMQ no vetor A
// Ex: v = [-1 -1 3 3 3 4 4 5 10 10]
//     A = [ 2  2 3 3 3 2 2 1  2  2] - frequencia de cada valor
int preprocess_most_frequent_value() {
   int freq = 0, last = INF, first_index = 0;
   fori(i,N) {
      if ( v[i] != last ) {
         forr(j,first_index,i-1){A[j]=-freq; inicio[j]=first_index; fim[j]=i-1;}
         freq = 1; first_index = i;
      }
      else freq++;
      last = v[i];
   }
   forr(j,first_index,N-1){A[j]=-freq; inicio[j]=first_index; fim[j]=N-1;}
   memset( M, -1, sizeof( M ) );
   initialize( 1, 0, N-1 );
}
// retorna o indice do valor mais frequente. trata o primeiro e o ultimo valores
// do intervalo como casos especiais
int query_most_frequent_value( int b, int e ) {
   int freq1, freq2, freq3 = -INF, index, MAX;
   freq1 = min( fim[b], e ) - b + 1; freq2 = e - max( inicio[e], b ) + 1;
   if ( b + freq1 < e - freq2 ) {
      index = query( 1, 0, N-1, b + freq1, e - freq2 );
      freq3 = -A[index];
   }
   MAX = max( freq1, max( freq2, freq3 ) );
   return ( MAX == freq1 ) ? ( b ) : ( MAX == freq2 ? e : index );
}
// 3 - determinar o segmento de soma maxima.
struct interval { int mr_n, mr, ml_n, ml, m_n, m_center, full, n; };
interval M[MAXIND]; // unica alteracao nos dados globais
```

UFMG – Universidade Federal de Minas Gerais

```
interval join( const interval &i1, const interval &i2 ) {
   int max_sum = i1.mr + i2.ml, max_n = i1.mr_n + i2.ml_n;
   interval i;
   if( i1.mr + i2.full >= i2.mr ) {
      i.mr = i1.mr+i2.full; i.mr_n = i1.mr_n + i2.n;
      if( i.mr > max_sum ) { max_sum = i.mr; max_n = i.mr_n; }
      else if ( i.mr == max_sum && i.mr_n > max_n ) max_n = i.mr_n;
   }
   else { i.mr = i2.mr; i.mr_n = i2.mr_n; }
   if( i2.ml + i1.full >= i1.ml ) {
      i.ml = i2.ml+i1.full; i.ml_n = i2.ml_n + i1.n;
      if( i.ml > max_sum ) { max_sum = i.ml; max_n = i.ml_n; }
      else if( i.ml == max_sum && i.ml_n > max_n ) max_n = i.ml_n;
   }
   else { i.ml = i1.ml;  i.ml_n = i1.ml_n; }
   if( i1.m_center > max_sum ) { max_sum = i1.m_center; max_n = i1.m_n; }
   else if( i1.m_center == max_sum && i1.m_n > max_n ) max_n = i1.m_n;
   if( i2.m_center > max_sum ) { max_sum = i2.m_center; max_n = i2.m_n; }
   else if( i2.m_center == max_sum && i2.m_n > max_n ) max_n = i2.m_n;
   i.m_center = max_sum; i.m_n = max_n;
   i.full = i1.full + i2.full; i.n = i1.n + i2.n;
   return i;
}
void initialize( int node, int b, int e ) {
   if ( b == e ) {
      M[node].mr_n = M[node].ml_n = M[node].m_n = M[node].n        = 1;
      M[node].full = M[node].mr = M[node].ml  = M[node].m_center = A[b];
   }
   else {
      initialize( 2 * node, b, (b + e) / 2 );
      initialize( 2 * node + 1, (b + e) / 2 + 1, e );
      M[node] = join( M[2*node], M[2*node + 1] );
   }
}
interval query( int node, int b, int e, int i, int j ) {
    interval p1;
    if ( i > e || j < b ) { p1.n = -1; return p1; }
    if ( b >= i && e <= j ) return M[node];
    p1 = query( 2 * node, b, (b + e) / 2, i, j );
    interval p2 = query( 2 * node + 1, (b + e) / 2 + 1, e, i, j );
    if ( p1.n == -1 ) return p2; if ( p2.n == -1 ) return p1;
    return join( p1, p2 );
}
// 4 - construir uma treap (Binary Search Heap). Cada no tem uma prioridade
// (usada para montar o heap) e um label (usado para montar a arvore binaria).
// Solucao: Divisao e conquista com arvore de segmentos, em que cada consulta
// retorna o proximo no da treap
typedef pair< string, int > treap_node; // label, prioridade
treap_node A[MAXN]; // A[i] - i-esimo elemento da sequencia original
void build_treap( int i, int j ) {
   if ( i > j ) return;
   if ( i == j ) {printf("(%s/%d)", A[i].first.c_str(), -A[i].second); return;}
   int index = query( 1, 0, N-1, i, j );
   printf( "(" ); build_treap( i, index - 1 );
   printf( "%s/%d", A[index].first.c_str(), -A[index].second );
   build_treap( index + 1, j ); printf( ")" );
}
int main() {
   string s, label; int a;
   while ( cin >> N && N ) {
      memset( M, -1, sizeof( M ) );
      fori(i,N) {
         cin >> s;
         int separator = s.find('/');
```

```
        label = s.substr( 0, separator );
        sscanf( s.substr( separator + 1 ).c_str(), "%d", &a );
        A[i] = make_pair( label, −a );
     }
     sort( A, A + N ); // ordena de acordo com os labels
     initialize( 1, 0, N−1 );
     build_treap( 0, N−1 );
     printf( "\n" );
  }
  return 0;
}
// 5 − determinar a soma dos dois elementos de maior valor em [i, j].
int max_sum( int node, int b, int e, int i, int j ) {
  int index, left, right;
  index = query( node, b, e, i, j );
  left = query( node, b, e, i, index − 1 );
  right = query( node, b, e, index + 1, j );
  return −A[index] + max( −A[left], −A[right] );
}
// 6 − determinar qual o no mais a esquerda na arvore tem valor
//     maior que K, e atualizar os dados apropriadamente.
void update( int node, int l, int r, int K )
{
  if ( l == r ) { A[l] −= K; M[node] = l; return; }
  int m = ( l + r ) >> 1;
  if ( A[M[2 * node]] >= K ) update( 2 * node, l, m, K );
  else if ( A[M[2 * node + 1]] >= K ) update(2 * node + 1, m + 1, r, K);
  if ( A[M[2 * node]] >= A[M[2 * node + 1]] ) M[node] = M[2 * node];
  else M[node] = M[2 * node + 1];
}
// Árvore de intervalos
#define MAX 30010
struct Node { int l, r, x, count, cl, cr; };
Node tree[4*MAX];
void create(int l, int r, int i = 1) {
  tree[i].l = l; tree[i].r = r; tree[i].x = (l + r) / 2;
  tree[i].count = tree[i].cl = tree[i].cr = 0;
  if (r−l == 1) return;
  if (l < tree[i].x) create(l,tree[i].x,2*i);
  if (r > tree[i].x) create(tree[i].x,r,2*i+1);
}
int update(int l, int r, int w = 1, int i = 1) {
  if (l <= tree[i].l && r >= tree[i].r) tree[i].count += w;
  else if (tree[i].r−tree[i].l > 1) {
     if (l < tree[i].x) tree[i].cl = update(l,r,w,i*2);
     if (r > tree[i].x) tree[i].cr = update(l,r,w,i*2+1);
  }
  return tree[i].count > 0 ? tree[i].r−tree[i].l:tree[i].cl+tree[i].cr;
}
int query(int x, int i = 1) {
  int ans = tree[i].count;
  if (tree[i].r−tree[i].l == 1) return ans;
  if (x < tree[i].x) ans += query(x,2*i);
  if (x >= tree[i].x) ans += query(x,2*i+1);
  return ans;
}
int count_ocurrences(int l, int r, int i = 1) {
  int ans = 0;
  if (tree[i].count > 0) return r − l;
  if (tree[i].r − tree[i].l == 1) return 0;
  if (l == tree[i].l && r >= tree[i].x) ans += tree[i].cl;
  else if (l < tree[i].x && tree[i].cl)
     ans += count_ocurrences(l,min(r,tree[i].x),2*i);
  if (r == tree[i].r && l < tree[i].x) ans += tree[i].cr;
```

UFMG − Universidade Federal de Minas Gerais

```
  else if (r > tree[i].x && tree[i].cr)
     ans += count_ocurrences(max(l,tree[i].x),r,2*i+1);
  return ans;
}
// Fenwick Tree (BIT) 1D
int tree[MAX];
void create(int t[], int n) { memset(t,0,n*sizeof(int)); }
int query(int tree[], int from, int to) {
  int sum;
  if (from != 0) return query(tree, 0, to) − query(tree, 0, from−1);
  for (sum=0; to >= 0; to = (to & (to + 1)) − 1) sum += tree[to];
  return sum;
}
void update(int tree[], int k, int inc, int n) {
  for ( ; k < n; k |= k + 1) tree[k] += inc;
}
// Fenwick Tree (BIT) 2D
int tree[MAX1][MAX2];
void create(int n1, int n2) {
  for (int i=0; i < n1; i++) memset(tree[i],0,n2*sizeof(int));
}
int query2(int x, int y) {
  int sum; for (sum=0; y >= 0; y=(y&(y+1))−1) sum += tree[x][y];
  return sum;
}
int query(int x1, int y1, int x2, int y2) {
  int sum;
  if (x1 != 0) return query(0,y1,x2,y2) − query(0,y1,x1−1,y2);
  for (sum=0; x2 >= 0; x2=(x2&(x2+1))−1) {
     sum += query2(x2,y2);
     if (y1) sum −= query2(x2,y1−1);
  }
  return sum;
}
void update(int x, int y, int inc, int n1, int n2) {
  int y2; for (;x < n1;x|=x+1) for (y2=y; y2 < n2; y2|=y2+1) tree[x][y2]+=inc;
}

/// ********************************* MISC ********************************* ///
// Bitwise operations
A | B;                            // set union
A & B;                            // set intersection
A & ~B;                           // set subtraction
((1 << N) − 1) & ~A               // set negation
A |= 1 << bit;                    // set bit
A &= ~(1 << bit);                 // clear bit
(A & 1 << bit) != 0;              // test whether element bit is in A
(A > 0) && (A & (A − 1)) == 0 // whether A has exactly 1 element (2^x)
A = A & (A − 1)                   // remove the smallest element from A
A = A & ~(A − 1)                  // remove all but the small. elem. from A
A & (A << 1)                      // check if there are adjacent bits
__builtin_ctz(A)                  // count trailing zeros
__builtin_clz(A)                  // count leading zeros,do not use when A=0
__builtin_popcount(A)             // number of 1 bits
for( int x = 0; x < (1 << n); ++x )       // all subsets of {0,...,n−1}
for( int x = s; x != 0; x = (x − 1) & s ) // all non−empty subsets of s
for( int x = 0; x = x − s & s; )  // all non−empty subsets of s in increa. order
// all subsets of {0,...,n−1} in gray code order
set = 0;
forr(j,1,(1<<n)−1) {
  last_set = set; set ^= j & ~(j−1);
  diff = set ^ last_set; index = __builtin_ctz( diff );
}
// iterate through all k−element subsets of {0,...,n−1} in increase order
```

```
int x = (1 << k) - 1; // do not use when k=0
while ( !(x & 1 << n) ) {
   int lo = x & ~(x - 1), lz = (x + lo) & ~x;
   x |= lz; x &= ~(lz - 1); x |= (lz / lo / 2) - 1;
}
// Dados n pontos distintos 3D, p1<p2 se e somente se: p1.x<p2.x AND p1.y < p2.y
// AND p1.z<p2.z. Um ponto pi eh excelente se nao existe j tal que pj < pi.
// Problema: determinar quantos pontos sao excelentes. Solucao: ordenar pontos
// pela coordenada z, e varrer os pontos nesta ordem decidindo quais sao
// excelentes. Quando o ponto pi eh avaliado, todos os pontos que podem ser
// melhores do que ele ja terao sido avaliados, uma vez que estes possuem
// coordenada z menor que pi.z. A ideia eh manter um conjunto S de pontos
// ordenados (em relacao a x) que sao excelentes e nao sao dominados por nenhum
// outro ponto excelente. Complexidade: O(n log n).
struct point { int x, y, z;
bool operator<( const point & o ) const { return x < o.x; } };
bool comp_z( const point & a, const point & b ) { return a.z < b.z; }
point v[100010]; set< point > S; // ordenado em relacao a x
vector< point > V; // auxiliar
// v[i] eh um ponto excelente se tiver a menor coordenada x ou se sua coordenada
// y for menor q a coordenada y do ponto anterior a ele em relacao a coord. x
bool is_excellent( int i ) {
   set< point >::iterator j = S.lower_bound( v[i] );
   if ( j == S.begin() ) return true; --j;
   return v[i].y < j->y;
}
int main() {
   int T, N;
   scanf( "%d", &T );
   while ( T-- )
   {
      scanf( "%d", &N );
      fori(i,N) scanf( "%d %d %d", &v[i].x, &v[i].y, &v[i].z );
      sort( v, v+N, comp_z );
      int asw = 1; // o 1o. ponto eh excelente
      S.clear(); S.insert( v[0] );
      forr(i,1,N-1) if ( is_excellent( i ) ) {
         asw++; // ponto i eh excelente
         // remove pontos excelentes que sao dominados por v[i]
         // dominado: ponto p2 eh dominado pelo ponto p1 se p1.x < p2.x
         // e p1.y < p2.y; ainda assim p2 eh um ponto excelente
         V.clear();
         for(set<point>::iterator it = S.lower_bound(v[i]); it!=S.end(); ++it){
            if ( v[i].y < it->y ) V.push_back( *it );
            else break; // ja que a coordenada y esta diminuindo
         }
         fori(j,V.sz) S.erase( V[j] );
         S.insert( v[i] );
      } printf("%d\n", asw);
   }
}
// Dados N valores inteiros em um vetor v, calcula a soma das medianas de cada
// intervalo de tamanho K. Complexidade: O(n log n)
const int LOGMAXN = 17, MAXN = 65536; // valores de v no intervalo [0, MAXN]
int tree[LOGMAXN][MAXN];
void insert(int x) { fori(i,LOGMAXN) { tree[i][x]++; x/=2; } }
void erase(int x) { fori(i,LOGMAXN) { tree[i][x]--; x/=2; } }
int kth_element(int k) {
   int a = 0, b = LOGMAXN - 1;
   while (b--) { a*=2; if (tree[b][a]<k) k -= tree[b][a++]; }
   return a;
}
long long sum_of_medians( const vector< int > & v, int N, int K ) {
   long long result = 0;
```

UFMG – Universidade Federal de Minas Gerais

```
   memset( tree, 0, sizeof(tree) );
   fori(i,N) {
      insert( v[i] );
      if (i>=K) erase( v[i-K] );
      if (i>=K-1) result += kth_element( (K+1)/2 );
   }
   return result;
}
// Algoritmo probabilistico: determinar se existe um nro q aparece + que
// n/2 vezes em um intervalo de tamanho n. Prob. de acertar: 1-(1/2)^ITER
#define MAX 300000
#define ITER 18
int n, Q, C, lo, hi, m, iter, x, cnt, a[MAX]; pair<int, int> b[MAX];
int main() {
    while ( scanf( "%d%d", &n, &C ) == 2 ) {
      fori(i,n) { scanf( "%d", &a[i] ); b[i] = make_pair( a[i], i ); }
      sort( b, b+n );
      scanf( "%d", &Q );
      fori(qq,Q) {
         scanf( "%d%d", &lo, &hi ); --lo; --hi;
         m = hi-lo+1;
         fori(iter,ITER) {
            x = a[lo+rand()%m];
            cnt = upper_bound( b, b+n, make_pair(x, hi) ) -
                  lower_bound( b, b+n, make_pair(x, lo) );
            if( cnt*2 > m ) break;
         }
         if( iter == ITER ) printf( "no\n" );
         else printf( "yes %d\n", x );
      }
   }
   return 0;
}
// Distancia minima entre dois cavalos em um tabuleiro de xadrez - O(1)
long long dist(long long x1, long long y1, long long x2, long long y2) {
    long long dx = abs(x2-x1), dy = abs(y2-y1), lb = (dx+1)/2;
    if ( abs(dx) == 1 && dy == 0 || abs(dy) == 1 && dx == 0 ) return 3;
    if ( abs(dx) == 2 && abs(dy) == 2 ) return 4;
    lb = max( lb, (dy+1)/2 ); lb = max( lb, (dx+dy+2)/3 );
    while ( (lb%2) != (dx+dy)%2 ) lb++;
    return lb;
}
// Notacao convencional p/ notacao polonesa reversa. Operandos: a, b, ..., z.
// Operadores em ordem de precedencia: +-*/^, expressoes com parentizacao.
void in_fix_2_reverse_polish( const string & s ) {
   int n = s.sz, pilha[MAXN], topo = 0; string res, operators = "+-*/^";
   fori(i,n) {
      if ( isalpha( s[i] ) ) printf("%c",s[i]);
      else if ( s[i] == '(' ) pilha[topo++] = '(';
      else if ( s[i] == ')' ) {
         while ( topo != -1 ) {
            if ( pilha[topo-1] == '(' ) { topo--; break; }
            printf("%c",pilha[--topo]);
         }
      }
      else if ( operators.find(s[i]) != string::npos ) {
         while ( topo && operators.find(pilha[topo-1])!=string::npos &&
                 operators.find(s[i]) <= operators.find(pilha[topo-1]) )
                     printf("%c",pilha[--topo]);
         pilha[topo++] = s[i];
      }
   }
   while ( topo != 0 ) printf("%c",pilha[--topo]); puts("");
}
```

```
// josephus
long long josephus(long long n, long long d) {
   long long K = 1;
   while (K <= (d-1)*n) K = (d * K + d - 2) / (d - 1);
   return d * n + 1 - K;
}
// returns the index (0-based!!) of s among all the possible anagrams
int freq[256];
long long count;
long long find_index(char *s, int n) {
   long long ans; int i;
   if (n < 2) { freq[*s-'a']++; count = 1; return 0; }
   ans = find_index(s+1,n-1);
   count = (count * n) / ++freq[*s-'a'];
   for (i=0; i < *s-'a'; i++) ans += freq[i] * count / n;
   return ans;
}
// returns the index k-th (1-based!!) anagram of s
string getAnagram(string s, int k) {
   int n = s.length();
   vector <int> freq(26,0), index(26);
   long long acc = 0, count;
   if (!n) return "";
   sort(s.begin(),s.end());
   for (int i=0; i < n; i++) { freq[s[i]-'a']++; index[s[i]-'a'] = i; }
   count = fat(n);
   for (int i=0; i < 26; i++) count /= fat(freq[i]);
   for (int i=0; i < 26; i++) {
      if (freq[i] && acc+count*freq[i]/n >= k) {
         s[index[i]] = s[0];
         return (char)('a'+i) + getAnagram(s.substr(1),k-acc);
      }
      acc += count * freq[i] / n;
   }
   return "";
}

/// ****************************** STRINGS ********************************* ///
// Minimum Lexicographic Rotation - O(n lg n)
int min_index( const string& s ) {
   int n = s.sz; vector<int> v(n);
   string ss, s1, s2; ss = s + s;
   fori(i,v.sz) v[i] = i;
   while ( v.sz != 1 ) {
      vector<int> vv;
      for ( int i = 0; i < v.sz; i += 2 ) {
         if ( i < v.sz - 1 ) {
            s1 = ss.substr( v[i], v[i+1] - v[i] );
            s2 = ss.substr( v[i+1], v[i+1] - v[i] );
            if ( s1 <= s2 ) vv.pb(v[i]);
            else vv.pb(v[i+1]);
         }
         else vv.pb(v[i]);
      }
      v = vv;
   }
   return v[0];
}
// Suffix array
// ASCII: 33 a 47 -> !"#$%&'()*+,-./
const int MAXN = 200010; // numero maximo de chars na string
int N, v[MAXN]; // tamanho da string atual e representacao em inteiros da string
int SA[MAXN], SAR[MAXN]; // suffix array e suffix array "reverso"
int LCP[MAXN]; // longest common prefix
```

UFMG – Universidade Federal de Minas Gerais

```
int CUR_ALF; // tamanho do alfabeto da string atual
const char FIRST_CHAR = ','; // primeiro char do alfabeto (tabela ASCII)
const char LAST_CHAR = 'z'; // ultimo char do alfabeto (tabela ASCII)
const int ALF = LAST_CHAR - FIRST_CHAR + 1; // tamanho do alfabeto
vector< int > occ[ALF]; // occ[i] - posicoes em que o char i aparece
#define GetI() ( SA12[t] < n0 ? SA12[t] * 3 + 1 : (SA12[t] - n0) * 3 + 2 )
inline bool leq( int a1, int a2, int b1, int b2 ) {
   return( a1 < b1 || a1 == b1 && a2 <= b2 ); }
inline bool leq( int a1, int a2, int a3, int b1, int b2, int b3 ) {
   return( a1 < b1 || a1 == b1 && leq( a2, a3, b2, b3 ) ); }
static void radix_pass( int* a, int* b, int* r, int n, int K ) {
   int* c = new int[K + 1];
   forr(i,0,K) c[i] = 0;
   fori(i,n) c[r[a[i]]]++;
   int sum = 0;
   forr(i,0,K) { int t = c[i]; c[i] = sum; sum += t; }
   fori(i,n) b[c[r[a[i]]]++] = a[i];
   delete [] c;
}
void suffix_array( int* s, int* SA, int n, int K )
{
   int i, j, n0 = (n+2)/3, n1 = (n+1)/3, n2 = n/3, n02 = n0+n2;
   int* s12 = new int[n02 + 3]; int* SA12 = new int[n02 + 3];
   int* s0   = new int[n0]; int* SA0 = new int[n0];
   int name = 0, c0 = -1, c1 = -1, c2 = -1;

   for(i = 0, j = 0; i < n+(n0-n1); i++) if ( i % 3 != 0 ) s12[j++] = i;
   s12[n02] = s12[n02+1] = s12[n02+2] = SA12[n02] = SA12[n02+1] = SA12[n02+2]=0;

   radix_pass( s12, SA12, s+2, n02, K ); radix_pass( SA12, s12, s+1, n02, K );
   radix_pass( s12, SA12, s, n02, K );

   fori(i,n02) {
      if (s[SA12[i]] != c0 || s[SA12[i]+1] != c1 || s[SA12[i]+2] != c2) {
         name++; c0 = s[SA12[i]]; c1 = s[SA12[i]+1];  c2 = s[SA12[i]+2]; }
      if ( SA12[i] % 3 == 1 ) s12[SA12[i]/3] = name;
      else s12[SA12[i]/3 + n0] = name;
   }

   if ( name < n02 ) {
      suffix_array( s12, SA12, n02, name ); fori(i,n02) s12[SA12[i]] = i + 1; }
   else fori(i,n02) SA12[s12[i] - 1] = i;

   j = 0;
   fori(i,n02) if ( SA12[i] < n0 ) s0[j++] = 3 * SA12[i];

   radix_pass(s0, SA0, s, n0, K);

   for ( int p = 0, t = n0-n1, k = 0; k < n; k++ ) {
      int i = GetI();
      int j = SA0[p];
      if ( SA12[t] < n0 ?
           leq( s[i], s12[SA12[t] + n0], s[j], s12[j/3] ) :
           leq( s[i], s[i+1], s12[SA12[t]-n0+1], s[j], s[j+1], s12[j/3+n0] ) ) {
             SA[k] = i; t++;
             if ( t == n02 ) for ( k++; p < n0; p++, k++ ) SA[k] = SA0[p];
      }
      else {
         SA[k] = j; p++;
         if ( p == n0 ) for ( k++; t < n02; t++, k++ ) SA[k] = GetI();
      }
   }
   delete [] s12; delete [] SA12; delete [] SA0; delete [] s0;
}
```

```
// Transforma alfabeto qualquer em alfabeto inteiro
// Ex: "aebeg" para "1 3 2 3 4". Complexidade: O(N + ALF)
void initialize( const string & s ) {
   N = s.sz;
   memset( v, 0, sizeof( v ) );
   int cnt = 1;
   fori(i,ALF) occ[i].clear();
   fori(i,N) occ[s[i] − FIRST_CHAR].pb( i );
   fori(i,ALF) if (!occ[i].empty()) {fori(j,occ[i].sz) v[occ[i][j]]=cnt; ++cnt;}
   CUR_ALF = cnt−1;
}
// Constroi o vetor LCP. Complexidade: O(N).
// LCP[i] = longest common prefix dos sufixos i e i+1
void lcp( const string & s ) {
   fori(i,N) SAR[SA[i]] = i;
   int h = 0, r;
   fori(p,N) if ( SAR[p]+1 < N ) {
      r = SA[SAR[p]+1];
      while( r+h < N && p+h < N && s[r+h] == s[p+h] ) h++;
      LCP[SAR[p]] = h;
      if( h > 0 ) h−−;
   }
   // apenas se for necessario calcular lcp em O(log n) − usando segtree
   memset( M, −1, sizeof( M ) );
   initialize( 1, 0, N−2 );
}
// Retorna o LCP dos sufixos i e j. Complexidade: O(n)
// Propriedade: lcp(i, j) = min_{i <= k < j} LCP[k]
int lcp_linear( int i, int j ) {
   int MIN = INF;
   forr(k,i,j−1) if ( LCP[k] < MIN ) MIN = LCP[k];
   return MIN;
}
// Segtree para queries de qualquer intervalo em lcp em O(log n)
// Overhead da segtree pode nao compensar
const int MAXIND = 525000;
int M[MAXIND];
void initialize( int node, int b, int e ) { ... } // LCP no lugar de A
int query( int node, int b, int e, int i, int j ) { ... } // LCP no lugar de A
// Retorna o LCP dos sufixos i e j. Complexidade O(log n)
// Necessario criar a segtree para fazer RMQ
int lcp_log( int i, int j ) { return LCP[query( 1, 0, N−2, i, j−1 )]; }
// Aplicacoes de suffix array:
// 1 − Dada uma string s de tamanho N, retorna a maior substring de s que
// aparece pelo menos M vezes. Em caso de empate retorna a menor lexicografica.
// Solucao: construir o suffix array de s e iterar com o contador i da posicao
// 0 a N−M computando o lcp dos sufixos i e i+K−1.
string longest_substring( const string & s, int M ) {
   if ( M == 1 ) return s; // apenas com arvore de segmentos
   N = s.sz;
   initialize( s ); suffix_array( v, SA, N, CUR_ALF ); lcp( s );
   int MAX = 0, pos = 0, aux;
   fori(i,N−M+1) if ((aux = lcp_log( i, i+M−1 )) > MAX) { MAX = aux; pos = i; }
   if ( MAX == 0 ) return "NAO EXISTE";
   return s.substr( SA[pos], MAX );
}
// 2 − Dada uma string s de tamanho N, retorna a maior substring de s que
// aparece pelo menos 2 vezes e quantas vezes ela aparece. Em caso de empate
// retorna a menor lexicografica. Solucao: construir o suffix array de s e
// iterar com o contador i da posicao 0 a N−2 computando o lcp dos sufixos i e
// i+1 e computando o numero de ocorrencias (estarao em posicoes contiguas do
// suffix array)
pair< string, int > longest_substring( const string & s ) {
   N = s.sz;
```

UFMG − Universidade Federal de Minas Gerais

```
   initialize( s ); suffix_array( v, SA, N, CUR_ALF ); lcp( s );
   int MAX = 0, pos = 0, cnt = 0;
   fori(i,N−1) {
      if ( LCP[i] > MAX ) { MAX = LCP[i]; pos = i; cnt = 2; }
      else if ( LCP[i] == MAX )
      {
         int j = 0;
         for( ; j < MAX; ++j ) if ( s[SA[i]+j] != s[SA[pos]+j] ) break;
         if ( j == MAX ) ++cnt;
      }
   }
   if ( MAX == 0 ) return make_pair( "", 0 );
   return make_pair( s.substr( SA[pos], MAX ), cnt );
}
// 3 − Dada uma string s, retorna o numero de substrings distintas em s
// Solucao: contar o numero de nos com excecao da raiz na suffix tree. Isso pode
// ser feito atraves da representacao em suffix array
int number_of_substrings( const string & s ) {
   int asw = 0;
   initialize( s ); suffix_array( v, SA, N, CUR_ALF ); lcp( s );
   asw += N − SA[0];
   fori(i,N−1) asw += ( N − SA[i+1] ) − LCP[i];
   return asw;
}
// 4 − Dada uma string s, retorna o numero de substrings distintas em s que
// aparecem pelo menos duas vezes
int number_of_substrings_twice( const string & s ) {
   int asw = 0, last = 0;
   initialize( s ); suffix_array( v, SA, N, CUR_ALF ); lcp( s );
   fori(i,N−1) { if ( LCP[i] > last ) asw += LCP[i] − last; last = LCP[i]; }
   return asw;
}
// 5 − Dado um vetor de strings vs, retorna a maior substring comum de todas as
// strings armazenadas em vs
int longest_common_substring( const vector< string > & vs ) {
   int asw = 0, cnt = 0, num_words = vs.sz, end = 0, sum = 0, aux = 0;
   char c = '!'; // se vs.sz > 15, gerar tab. ASCII e verificar quais chars usar
   string s = "";
   vector< int > init, occur( num_words, 0 );;
   fori(i,num_words) {
      s += vs[i]; s.append( 1, c );
      fori(j,vs[i].sz) init.pb( cnt );
      init.pb( cnt );
      cnt++; c++;
   }
   initialize( s );
   suffix_array( v, SA, N, CUR_ALF );
   lcp( s );
   // o intervalo [start,end] deve conter pelo menos um sufixo
   // de cada uma das strings de vs
   fori(start,N−1) {
      if ( start != 0 ) {
         occur[init[SA[start−1]]]−−;
         if ( occur[init[SA[start−1]]] == 0 ) sum−−;
      }
      while ( sum != num_words && end < N ) {
         if ( occur[init[SA[end]]] == 0 ) { occur[init[SA[end]]] = 1; sum++; }
         else occur[init[SA[end]]]++;
         end++;
      }
      if (sum == num_words && ( aux = lcp( start, end − 1 ) ) > asw) asw = aux;
   }
   return asw;
}
```

```
// String matching - Algoritmo KMP - O(n+m)
// F[i] - size of the largest prefix of pattern[0..i] that is also a
// suffix of pattern[1..i]. Ex: pattern = {a,b,a,c,a,b}, F = {0,0,1,0,1,2}
int F[64];
void build_failure_function( const string & pattern ) {
   int m = pattern.sz;
   F[0] = -1;
   fori(i,m) {
      F[i+1] = F[i] + 1;
      while ( F[i+1] > 0 && pattern[i] != pattern[ F[i+1]-1 ] )
         F[i+1] = F[ F[i+1]-1 ] + 1;
   }
}
// retorna a posicao inicial de cada ocorrencia de pattern em text
vector<int> KMP( const string & text, const string & pattern ) {
   build_failure_function( pattern );
   vector<int> start_positions;
   int j = 0, m = pattern.sz, n = text.sz;
   fori(i,n) while ( true ) {
      if ( text[i] == pattern[j] ) {
         if ( ++j == m ) { start_positions.pb( i - m + 1 ); j = F[j]; } break;
      }
      if ( j == 0 ) break;
      j = F[j];
   }
   return start_positions;
}
// String matching - Algoritmo aho-corasick
const int NULO = -1, MAX_NO = 100010, MAX_PAD = 10010;
typedef map<char, int> mapach;
typedef map<string, int> mapastr;
struct automato {
   mapach trans[MAX_NO];
   mapastr pad;
   list<int> pos[MAX_PAD];
   int falha[MAX_NO], final[MAX_NO], tam[MAX_PAD], numNos;
   automato(): numNos(0) {}
   // Funcao de inicializacao. 1 chamada por instancia, antes das outras funcoes
   void inic() {
      memset(falha, NULO, sizeof(falha));
      memset(final, NULO,  sizeof(final));
      fori(i,numNos) trans[i].clear();
      pad.clear(); numNos = 1;
   }
   // Funcao que adiciona um padrao ao automato. Uma chamada por padrao, depois
   // da inicializacao. Retorna o ind. de acesso a variavel global pos.
   int adiciona_padrao(const char* s) {
      pair<mapach::iterator, bool> pch;
      int i, no = 0, numPad = pad.size();
      if (pad.count(s)) return pad[s];
      else pad.insert(make_pair(s, numPad));
      for (i = 0; s[i]; i++) {
         if ((pch = trans[no].insert(make_pair(s[i], numNos))).second) numNos++;
         no = pch.first->second; }
      tam[numPad] = i ? i : 1;
      return final[no] = numPad;
   }
   // Funcao que gera o tratamento de falhas.
   // Uma chamada por instancia, depois da adicao de todos os padroes.
   void gera_falhas() {
      queue<int> fila;
      int filho;
      tr(it, trans[0].begin(), trans[0].end()) {
         falha[filho = it->second] = 0; fila.push(filho); }
```

UFMG – Universidade Federal de Minas Gerais

```
      while (!fila.empty()) {
         int atual = fila.front(); fila.pop();
         tr(it, trans[atual].begin(), trans[atual].end()) {
            char c = it->first; filho = it->second;
            int ret = falha[atual];
            while (ret != NULO && !trans[ret].count(c)) ret = falha[ret];
            if (ret != NULO) {
               falha[filho] = trans[ret][c];
               if (final[filho]==NULO && final[falha[filho]]!=NULO)
               final[filho] = final[falha[filho]];
            }
            else if (trans[0].count(c)) falha[filho] = trans[0][c];
            fila.push(filho);
         }
      }
   }
   // Funcao que busca os padroes em uma texto de consulta.
   // Uma chamada por consulta,depois da geracao do tratamento de falhas.
   // Preenche a variavel global pos com posicoes iniciais d cada padrao.
   void consulta(const char* s) {
      int ret, atual = 0, i = 0;
      int N = pad.size();
      for (int j = 0; j < N; j++) pos[j].clear();
      do {
         while(atual != NULO && !trans[atual].count(s[i])) atual = falha[atual];
         atual = (atual == NULO) ? 0 : trans[atual][s[i]];
         for (ret = atual; ret != NULO && final[ret] != NULO; ret = falha[ret]){
            pos[final[ret]].push_back(i - tam[final[ret]] + 1);
            while (falha[ret]!=NULO && final[falha[ret]]==final[ret])
            ret = falha[ret]; }
      } while (s[i++]);
   }
};
// funcao main para algoritmo aho-corasick
// IMPORTANTE: usar "aut.pad[patterns[j]]" para indexar padrao j
// chamar, nasta ordem: inic, adiciona_padrao, gera_falhas, consulta
int main() {
   int k, q;
   string text, pattern;
   cin >> k;   // numero de instancias
   fori(i,k) {
      cin >> text >> q; // texto, numero de padroes
      vector<string> patterns( q ); // padroes a serem pesquisados
      automato aut; aut.inic(); // cria e inicia automato
      fori(j,q) {
         cin >> pattern;
         patterns[j] = pattern;
         aut.adiciona_padrao(pattern.c_str()); } // adiciona padrao
      aut.gera_falhas(); // tratamento de falhas
      aut.consulta(text.c_str()); // realiza pesquisa em todos
      fori(j,q) {
         // imprime se o j-esimo padrao existe ou nao no texto
         if( aut.pos[ aut.pad[patterns[j]] ].empty() ) puts("n");
         else cout << "y" << endl; }
      //// imprime o primeiro caractere das ocorrencias dos padroes
      //fori(j,q) {
         //cout << patterns[j] << ": ";
         //tr(it, aut.pos[ aut.pad[patterns[j]] ].begin(),
            //aut.pos[ aut.pad[patterns[j]] ].end()) cout << *it << " ";
         //cout << endl;
      //} cout << endl;
   }
   return 0;
}
```