1 Flow and matching

1.1 Max flow (Dinić)

```
// init(N) where N is no of vertices
// add(u, v, cap) add edge from u to v
// GetMaxFlow(S, T) to get max flow from S to T
// Running time: O(n^3). Very fast in practice
// Solved for N = 5000 in 0.10 seconds on SPOJ
// To get flows, look at edges with non zero flow

const int MAXN = 5000, inf = 2e18;
int src, snk, N, nxt[MAXN+5], dist[MAXN+5];
vector<edge>E[MAXN+5];

struct edge{
  int v, cap, opposite, flow;
};

void init(int _n) { N = _n; }

void add(int u, int v, int cap) {
  E[u].pb({v, cap, len(E[v]), 0});
  E[v].pb({u, 0, len(E[u])-1, 0});
}

bool bfs() {
  reset(dist, -1); dist[src] = 0;
  queue<int> q({src});
  while(!q.empty()) {
    int u = q.front(); q.pop();
    for(int i = 0; i<E[u].size(); i++) {
      if(E[u][i].cap > E[u][i].flow) {
        int v = E[u][i].v;
        if(dist[v] == -1) {
          dist[v] = dist[u] +1;
          q.push(v);
    } } }
  } return dist[snk] != -1;
}

int dfs(int u, int sentFlow) {
  if(u == snk || sentFlow == 0) return sentFlow;
  for(; nxt[u] < E[u].size(); nxt[u]++) {
    int v = E[u][nxt[u]].v, c = E[u][nxt[u]].cap;
    int f = E[u][nxt[u]].flow;
    int opposite = E[u][nxt[u]].opposite;
    if(dist[v] == dist[u]+1 && c > f) {
      int tmp = dfs(v,min(sentFlow,c-f));
      if(tmp != 0) {
        E[u][nxt[u]].flow += tmp;
        E[v][opposite].flow -= tmp;
        return tmp;
    } }
  } return 0;
}

ll GetMaxFlow(int S, int T) {
  src = S; snk = T; ll totalFlow = 0;
  while(bfs()) {
    memset(nxt,0,sizeof(nxt));
    while(int sentFlow = dfs(src,inf))
      totalFlow += sentFlow;
  } return totalFlow;
}
```

1.2 Max bipartite matching

```
// Returns maximum bipartite matching
// The match can be recovered using the vector L
// Works with 0 based too. Fast in practice

struct BipartiteMatcher {
  vector<vll> G; vll L, R, Viz;

  BipartiteMatcher(int n, int m) :
  G(n), L(n, -1), R(m, -1), Viz(n) {}

  void AddEdge(int a, int b) { G[a].pb(b); }

  bool Match(int node) {
    if(Viz[node]) return false;
    Viz[node] = true;

    for(auto vec: G[node]) {
```

```
      if(R[vec] == -1) {
        L[node] = vec; R[vec] = node; return true;
    } }

    for(auto vec: G[node]) {
      if(Match(R[vec])) {
        L[node] = vec; R[vec] = node; return true;
    } } return false; }

  int GetMaximumMatching() {
    int ok = true;
    while (ok--) {
      fill(all(Viz), 0);
      for(int i = 0; i < len(L); ++i)
        if(L[i] == -1) ok |= Match(i);
    }
    int ret = 0;
    for(int i = 0; i < len(L); ++i)
      ret += (L[i] != -1);
    return ret; }
};
```

2 Geometry

2.1 Miscellaneous Geometry

```
ld INF = 1e100, EPS = 1e-12, pi = acos(-1);
#define point PT
#define VP vector<PT>

struct PT{
  ld x, y;
  PT() {}
  PT(ld x, ld y) : x(x), y(y) {}
  PT(const PT &p) : x(p.x), y(p.y) {}
  PT operator + (const PT &p) const {
    return PT(x+p.x, y+p.y); }
  PT operator - (const PT &p) const {
    return PT(x-p.x, y-p.y); }
  PT operator * (ld c) const {return PT(x*c, y*c);}
  PT operator / (ld c) const {return PT(x/c, y/c);}
  bool operator == (const PT &p) const {
    return fabs(x-p.x) < EPS && fabs(y-p.y) < EPS;}
  bool operator != (const PT &p) const {
    return fabs(x-p.x) > EPS || fabs(y-p.y) > EPS;}
};
ld dot(PT p, PT q) { return p.x*q.x+p.y*q.y; }
ld dist2(PT p, PT q) { return dot(p-q,p-q); }
ld dist(PT p, PT q) { return sqrtl(dist2(p, q)); }
ld cross(PT p, PT q) { return p.x*q.y-p.y*q.x; }
ld mag(PT p) { return sqrtl(dot(p, p)); }
ld to_degree(ld angle) { return angle*180.0/pi; }
ld to_radian(ld angle) { return angle*pi/180.0; }
ld angle_in_radian(PT p, PT q){
  return acos(dot(p, q)/(mag(p)*mag(q))); }
ld angle_in_degree(PT p, PT q){
   return to_degree(angle_in_radian(p, q)); }
ostream &operator<<(ostream &os, const PT &p){
    return os<<"("<<p.x<<","<<p.y<< ")"; }
// rotate a pt CCW or CW around the origin
PT RotateCCW90(PT p) { return PT(-p.y,p.x); }
PT RotateCW90(PT p) { return PT(p.y,-p.x); }
PT RotateCCW(PT p, ld t) { return PT(
  p.x*cos(t)-p.y*sin(t), p.x*sin(t)+p.y*cos(t)); }
// project pt c onto line through a & b
// assuming a != b
PT ProjectPointLine(PT a, PT b, PT c) {
  return a+(b-a)*dot(c-a, b-a)/dot(b-a, b-a); }
// project pt c on line segment through a & b
PT ProjectPointSegment(PT a, PT b, PT c) {
  ld r = dot(b-a,b-a);
  if(fabs(r) < EPS) return a;
  r = dot(c-a, b-a)/r;
  if(r < 0) return a; if(r > 1) return b;
  return a +(b-a)*r; }
// compute dis from c to segment between a & b
ld DistancePointSegment(PT a, PT b, PT c) {
  return dist(c, ProjectPointSegment(a, b, c)); }
//get dis between pt(x,y,z) & plane ax+by+cz=d
ld DistancePointPlane(ld x, ld y, ld z,
  ld a, ld b, ld c, ld d)
{ return fabs(a*x+b*y+c*z-d)/sqrtl(a*a+b*b+c*c); }
// find if lines from a, b & c, d are parallel
bool LinesParallel(PT a, PT b, PT c, PT d) {
  return fabs(cross(b-a, c-d)) < EPS;}
bool LinesCollinear(PT a, PT b, PT c, PT d) {
```

```
  return LinesParallel(a, b, c, d) &&
  fabs(cross(a-b, a-c)) < EPS &&
  fabs(cross(c-d, c-a)) < EPS; }
// find if line segment from a, b intersects
// with line segment from c to d
bool SegmentsIntersect(PT a, PT b, PT c, PT d) {
  if(LinesCollinear(a, b, c, d)) {
    if(dist2(a, c) < EPS ||
        dist2(a, d) < EPS || dist2(b, c) < EPS ||
        dist2(b, d) < EPS) return true;
    if(dot(c-a, c-b) > 0 && dot(d-a, d-b) > 0
              && dot(c-b, d-b) > 0) return false;
    return true;
  }
  if(cross(d-a, b-a)*cross(c-a, b-a) > 0)
    return false;
  if(cross(a-c, d-c)*cross(b-c, d-c) > 0)
    return false;
  return true; }
//get intersection of line passing through a,b
//with line passing through c to d, assuming
//unique intersection exists; for segment
//intersection, find if segments intersect first
PT ComputeLineIntersection(PT a, PT b, PT c, PT d){
  b = b-a; d = c-d; c = c-a;
  assert(dot(b, b) > EPS && dot(d, d) > EPS);
  return a + b*cross(c, d)/cross(b, d); }
// compute center of circle given three pts
PT ComputeCircleCenter(PT a, PT b, PT c) {
  b=(a+b)/2; c=(a+c)/2; return
  ComputeLineIntersection(b, b+RotateCW90(a-b),
  c, c+RotateCW90(a-c)); }
// find if pt is in possibly non-convex polygon
// returns 1 for strictly interior pts,
// 0 for strictly exterior pts
bool PointInPolygon(const VP &p, PT q) {
  bool c = 0;
  for(int i = 0; i < len(p); i++) {
    int j = (i+1)%len(p);
    if((p[i].y <= q.y && q.y < p[j].y ||
        p[j].y <= q.y && q.y < p[i].y) &&
        q.x < p[i].x+(p[j].x-p[i].x)*
        (q.y-p[i].y)/(p[j].y-p[i].y)) c = !c;
  } return c; }
// determine if pt is on boundary of a polygon
bool PointOnPolygon(const VP &p, PT q) {
  for(int i = 0; i < len(p); i++)
    if(dist2(ProjectPointSegment(p[i],
      p[(i+1)%len(p)], q), q) < EPS) return true;
  return false; }
// find intersection of line through pts a, b
// with circle centered at c with radius r > 0
VP CircleLineIntersection(PT a, PT b, PT c, ld r) {
  VP ret; b = b-a; a = a-c;
  ld A = dot(b, b), B = dot(a, b);
  ld C = dot(a, a) - r*r, D = B*B - A*C;
  if(D < -EPS) return ret;
  ret.pb(c+a+b*(-B+sqrt(D+EPS))/A);
  if(D>EPS) ret.pb(c+a+b*(-B-sqrt(D))/A);
  return ret; }
// find intersection of circle at center a with
// radii r with circle at center b with radii R
VP CircleCircleIntersection(PT a,PT b,ld r,ld R) {
  VP ret; ld d = sqrt(dist2(a, b));
  if(d>r+R || d+min(r, R) < max(r, R)) return ret;
  ld x =(d*d-R*R+r*r)/(2*d), y = sqrt(r*r-x*x);
  PT v =(b-a)/d; ret.pb(a+v*x+RotateCCW90(v)*y);
  if(y > 0) ret.pb(a+v*x - RotateCCW90(v)*y);
  return ret; }
// This code computes the area or centroid of a
// polygon, assuming that the coordinates are
// listed in clockwise or anticlockwise fashion
// Note that the centroid is often known as
// the "center of gravity" or "center of mass".
ld ComputeSignedArea(const VP &p) {
  ld area = 0;
  for(int i = 0; i < len(p); i++) {
    int j =(i+1) % len(p);
    area += p[i].x*p[j].y - p[j].x*p[i].y;
  } return area / 2.0; }
ld ComputeArea(const VP &p) {
  return fabs(ComputeSignedArea(p)); }
PT ComputeCentroid(const VP &p) {
  PT c(0,0);
  ld scale = 6.0 * ComputeSignedArea(p);
  for(int i = 0; i < len(p); i++) {
```

```
    int j =(i+1) % len(p);
    c += (p[i]+p[j])*(p[i].x*p[j].y-p[j].x*p[i].y);
  } return c / scale; }
// tests whether or not a given polygon
// (in CW or CCW order) is simple
bool IsSimple(const VP &p) {
for(int i = 0; i < len(p); i++)
  for(int k = i+1; k < len(p); k++) {
    int j = (i+1) % len(p), l = (k+1) % len(p);
    if(i == l || j == k) continue;
    if(SegmentsIntersect(p[i], p[j], p[k], p[l]))
      return false;
    } return true; }
```

2.2 Convex hull

```
// Compute the 2D convex hull of a set of points
// Eliminate redundant points from the hull if
// REMOVE_REDUNDANT is #defined.
// Running time: O(n log n)
// INPUT: a vector of input points, unordered.
// OUTPUT: a vector of points in the convex hull,
// anticlockwise, starting with bottomleft point
// Returned in the same vector as passed on.

#define REMOVE_REDUNDANT
#define point PT

const ld EPS = 1e-7;

struct PT {
  int x, y;
  PT() {}
  PT(int x, int y) : x(x), y(y) {}
  bool operator < (const PT &p) const {
    return mp(y,x) < mp(p.y,p.x); }
  bool operator == (const PT &p) const {
    return fabs(x-p.x)<EPS && fabs(y-p.y)<EPS; }
};

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

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

void ConvexHull(vector<PT> &pts) {
  sort(all(pts)); onlyunique(pts);
  vector<PT> up, dn;
  for (int i = 0; i < len(pts); i++) {
    while (up.size() > 1 && area2(up[up.size()-2],
           up.back(), pts[i]) >= 0) up.pop_back();
    while (len(dn) > 1 && area2(dn[len(dn)-2],
           dn.back(), pts[i]) <= 0) dn.pop_back();
    up.pb(pts[i]); dn.pb(pts[i]);
  } pts = dn;
  for(int i = len(up)-2; i > 0; i--) pts.pb(up[i]);

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

2.3 Minimum Enclosing Disk (Weizl)

```
// Minimum enclosing circle, Welzl's algorithm
// Expected linear time.
// If there are duplicate points, remove them first

#define point PT
#define circle CR
struct PT { ld x, y; };
```

```
struct CR {
  ld x, y, r;
  CR() {}
  CR(ld x, ld y, ld r): x(x), y(y), r(r) {}
};

CR b_md(vector<PT> R) {
  if(len(R) == 0) return CR(0, 0, -1);
  if(len(R) == 1) return CR(R[0].x, R[0].y, 0);
  if(len(R) == 2) return CR((R[0].x+R[1].x)/2.0,
                            (R[0].y+R[1].y)/2.0,
          hypot(R[0].x-R[1].x, R[0].y-R[1].y)/2.0);

  ld D = (R[0].x-R[2].x)*(R[1].y-R[2].y) -
                  (R[1].x-R[2].x)*(R[0].y-R[2].y);

  ld p0 =(((R[0].x-R[2].x)*(R[0].x+R[2].x) +
           (R[0].y-R[2].y)*(R[0].y+R[2].y))/2 *
           (R[1].y-R[2].y)-((R[1].x-R[2].x) *
           (R[1].x+R[2].x)+(R[1].y-R[2].y) *
           (R[1].y+R[2].y))/2*(R[0].y-R[2].y))/D;

  ld p1 =(((R[1].x-R[2].x)*(R[1].x+R[2].x) +
           (R[1].y-R[2].y)*(R[1].y+R[2].y))/2 *
           (R[0].x-R[2].x)-((R[0].x-R[2].x) *
           (R[0].x+R[2].x)+(R[0].y-R[2].y)*
           (R[0].y+R[2].y))/2*(R[1].x-R[2].x))/D;
  return CR(p0, p1, hypot(R[0].x-p0, R[0].y-p1));
}
CR b_minidisk(vector<PT>& P, int i, vector<PT> R){
  if(i == len(P) || len(R) == 3) return b_md(R);
  CR D = b_minidisk(P, i+1, R);
  if(hypot(P[i].x-D.x, P[i].y-D.y) > D.r) {
    R.push_back(P[i]);
    D = b_minidisk(P, i+1, R);
  } return D;
}

// Call this function.
CR minidisk(vector<PT> P) {
  shuffle(all(P), rng);
  return b_minidisk(P, 0, vector<PT>());
}
```

2.4 Pick's Theorem (Text)

For a polygon with all vertices on lattice points, $A = i + b/2 - 1$, where A is the area, i is the number of lattice points strictly within the polygon, and b is the number of lattice points on the boundary of the polygon. (Note, there is no generalization to higher dimensions)

3 Math Algorithms

3.1 Yarin Sieve

```
// This is the famous "Yarin sieve",
// for use when memory is tight.
#define MAXSIEVE 100000000 // Primes till this
#define MAXSIEVEHALF (MAXSIEVE/2)
#define MAXSQRT 5000 // sqrt(MAXSIEVE)/2
char a[MAXSIEVE/16+2];
#define isprime(n) (a[(n)>>4]&(1<<(((n)>>1)&7)))
// Works when n is odd

int i, j;
memset(a, 255, sizeof(a));
a[0] = 0xFE;
for(i = 1; i < MAXSQRT; i++)
  if(a[i >> 3] && (1 << (i & 7)))
    for(j = 3*i+1; j<MAXSIEVEHALF; j += 2*i+1)
      a[j>>3] &= ~(1 << (j & 7));
```

3.2 Modular arithmetic and linear Diophantine solver

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

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

// returns gcd(a,b); finds x,y s.t. d = ax + by
```

```
int extended_euclid(int a, int b, int &x, int &y) {
  int xx = y = 0; int yy = x = 1;
  while (b) {
    int q = a/b;
    int t = b; b = a%b; a = t;
    t = xx; xx = x-q*xx; x = t;
    t = yy; yy = y-q*yy; y = t;
  } return a;
}

// finds all solutions to ax = b (mod n)
vll modular_equation_solver(int a, int b, int n){
  vll solutions; int x, y;
  int d = extended_euclid(a, n, x, y);
  if (!(b%d)) {
    x = takemod(x*(b/d), n);
    for (int i = 0; i < d; i++)
      solutions.pb(takemod(x + i*(n/d), n));
  } return solutions;
}

// computes a^-1 (mod n), returns -1 on failure
int mod_inverse(int a, int n) {
  int x, y; int d = extended_euclid(a, n, x, y);
  if (d > 1) return -1;
  return takemod(x, n);
}

// Chinese remainder theorem (special case): find z
// such that z % x = a, z % y = b. Here, z is
// unique modulo M = lcm(x,y).
// Return (z,M). On failure, M = -1.
pii CRT(int a1, int n1, int a2, int n2) {
  int x, y; int d = extended_euclid(n1, n2, x, y);
  int l = (n1 * n2) / d, A = (a2-a1);
  if(A % d != 0) return mp(0, -1);
  return mp(takemod(a1+(x*A/d%(n2/d))*n1, l), l);
}

// Chinese remainder theorem: find z such that
// z % x[i] = a[i] for all i. Solution is
// unique mod M = lcm_i (x[i]). Return (z,M). On
// failure, M = -1. Note that we do not require
// the a[i]'s to be relatively prime.
pii CRT(const vll &x, const vll &a) {
  pii ret = mp(a[0], x[0]);
  for (int i = 1; i < x.size(); i++) {
    ret = CRT(ret.ff, ret.ss, a[i], x[i]);
    if (ret.ss == -1) break;
  } return ret;
}

// computes x and y such that ax + by = c;
// on failure returns false. g is gcd(a, b)
// int qa = b / g, qb = b / g;
// X = x + k * qa, Y = y - k * qb are also
// solutions of equation where k is any integer
bool linear_diophantine(int a, int b, int c,
                        int& x0, int& y0, int& g) {
  g = extended_euclid(abs(a), abs(b), x0, y0);
  if(c % g != 0) return false;
  x0 *= c / g; y0 *= c / g;
  if(a < 0) x0 *= -1; if(b < 0) y0 *= -1;
  assert(a * x0 + b * y0 == c);
  return true;
}
```

3.3 Gaussian elimination

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

const double EPS = 1e-10;
typedef double ld;
typedef vector<ld> VT;
```

```
typedef vector<VT> VVT;

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

  for(int i = 0; i < n; i++) {
    int pj = -1, pk = -1;
    for(int j = 0; j < n; j++) if(!ipiv[j])
      for(int k = 0; k < n; k++) if(!ipiv[k])
        if(pj==-1 || fabs(a[j][k])>fabs(a[pj][pk]))
        { pj = j; pk = k; }

    if(fabs(a[pj][pk]) < EPS) return 0; // singular
    ipiv[pk]++;
    swap(a[pj], a[pk]); swap(b[pj], b[pk]);

    if(pj != pk) det *= -1;
    irow[i] = pj; icol[i] = pk;

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

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

3.4 Solving linear systems (Text)

To solve a general system of linear equations, put it into matrix form and compute the reduced row echelon form. For example,

$$2x + y = 5$$
$$3x + 2y = 6$$

corresponds to the matrix

$$\left[\begin{array}{cc|c} 2 & 1 & 5 \\ 3 & 2 & 6 \end{array}\right]$$

with RREF

$$\left[\begin{array}{cc|c} 1 & 0 & 4 \\ 0 & 1 & -3 \end{array}\right]$$

After row reduction, if any row has a 1 in the rightmost column and 0 everywhere else, then the system is inconsistent and has no solution. Otherwise, to find a solution, set the variable corresponding to the leftmost 1 in each column equal to the corresponding value in the rightmost column, and set all other variables to 0. Ignore rows consisting entirely of 0. The solution is unique iff the rank of the matrix equals the number of variables.

3.5 Fast Fourier transform (FFT)

```
const ld pi = acos(-1);
auto FFT = [](vector<ld> a, vector<ld> b){
  auto DFT = [](vector<complex<ld>>&a, bool inv){
    int L=31-__builtin_clz(len(a)), n = 1 << L;
    vector<complex<ld>> A(n);
    for(int k = 0, r, i; k < n; A[r] = a[k++])
      for(i = r = 0; i < L; (r<<=1) |= (k>>i++)&1);
    complex<ld> w, wm, t;
    for(int m = 2, j, k; m <= n; m <<= 1) {
      w = {0, 2*pi/m}, wm = exp(inv ? -w : w);
      for(k = 0; k < n; k += m)
        for(j = 0, w = 1; j < m/2; ++j , w *= wm){
          t = w * A[k+j+m/2];
          A[k+j+m/2] = A[k+j]-t; A[k+j] += t;
        }
```

```
    }
    return A;
  };
  int n = 4<<31-__builtin_clz(max(len(a),len(b)));
  vector<complex<ld>> A(n), B(n), CC(n);
  for(int i = 0; i < n; ++i)
    A[i] = i<len(a)?a[i]:0, B[i] = i<len(b)?b[i]:0;
  vector<complex<ld>> AA = DFT(A,0), BB = DFT(B,0);
  for(int i = 0; i < n; ++i) CC[i] = AA[i]*BB[i];
  vector<ld> c;
  for(auto i:DFT(CC,1)) if(len(c)<len(a)+len(b)-1)
    c.pb(i.real()/n+1e-5);
  return c;
};
```

3.6 Pollard rho and (Rabin-Miller)

```
ull mul_mod(ull a, ull b, ull m){
  ull y = (ull)((ld)a * (ld)b / m + 1.0 / 2) * m;
  ull x = a * b, r = x - y;
  if ((ll) r < 0){
    r = r + m; y = y - 1;
  } return r;
}

ull C, a, b;

ull f(ull a, ull b){
  return (mul_mod(a, a, b) + C) % b; }

ull pollard(ull n){
  if(!(n % 2)) return 2;
  C = 0; ull iter = 0;
  while(iter <= 1000){
    ull x, y, d;
    x = y = 2; d = 1;
    while(d == 1){
      x = f(x, n);
      y = f(f(y, n), n);
      ull m = abs(x-y);
      a = m; b = n; d = gcd(a, b);
    }
    if(d != n) return d;
    iter++; C = rand();
  } return 1;
}

// Rabin-Miller primality testing algorithm
bool isPrime(ull n){
  ull d = n-1, s = 0;
  if(n <=3 || n == 5) return true;
  if(!(n % 2)) return false;
  while(!(d % 2)){ s++; d /= 2; }
  for(ull i = 0; i < 32; i++){
    ull a = rand(); a <<= 32;
    a += rand(); a %= (n-3); a += 2;
    ull x = power(a, d, n);   // make power ull
    if(x == 1 || x == n-1) continue;
    for(ull j = 1; j <= s-1; j++){
        x = mul_mod(x, x, n);
        if(x == 1) return false;
        if(x == n-1) break;
    }
    if(x != n-1) return false;
  } return true;
}

map<ull, int> factors;
// Precondition:  factors is an empty map,
//                n is a positive integer
// Postcondition: factors[p] is the exponent of p
//                in prime factorization of n
void fact(ull n){
  if(!isPrime(n)){
    ull fac = pollard(n);
    fact(n/fac); fact(fac);
  } else {
    map<ull,int>::iterator it = factors.find(n);
    if(it != factors.end()) (*it).ss++;
    else factors[n] = 1;
} }
```

3.7 Euler's Totient

```
// This took 0.5s to calculate with MAX = 10^7
#define MAX 10000000
```

```
int phi[MAX]; bool pr[MAX];

void totient(){
  for(int i = 0; i < MAX; i++){
    phi[i] = i; pr[i] = true;
  }
  for(int i = 2; i < MAX; i++)
    if(pr[i]){
      for(int j = i; j < MAX; j += i){
        pr[j] = false;
        phi[j] = phi[j] - (phi[j] / i);
      } pr[i] = true;
  } }
```

4 Graphs

4.1 Strongly connected components

```
struct SCC {
  int V, group_cnt;
  vector<vector<int>> adj, radj;
  vector<int> group_num, vis;
  stack<int> stk;

  // V = number of vertices. 0 based indexing
  SCC(int V): V(V), group_cnt(0), group_num(V),
              vis(V), adj(V), radj(V) {}

  // Call this to add an edge
  void add_edge(int v1, int v2) {
    adj[v1].pb(v2); radj[v2].pb(v1);
  }

  void fill_forward(int x) {
    vis[x] = true;
    for (int v: adj[x]) {
      if(!vis[v]) fill_forward(v);
    } stk.push(x);
  }

  void fill_backward(int x) {
    vis[x] = false;
    group_num[x] = group_cnt;
    for (int v: radj[x]) {
      if(vis[v]) fill_backward(v);
  } }

  // Returns no. of SCCs
  // group_num contains component assignments
  int get_scc() {
    for (int i = 0; i < V; i++) { // 0 based
      if (!vis[i]) fill_forward(i);
    }
    group_cnt = 0;
    while (!stk.empty()) {
      if (vis[stk.top()]) {
        ++group_cnt;
        fill_backward(stk.top());
      } stk.pop();
    } return group_cnt;
  }
};
```

4.2 Bridges

```
const int N = 1e5 + 10;
vector<int> adj[N], cutPts;
vector<pll> bridges; bool visited[N] = {0};

// Use p = -1 for root
void dfs(int u, int p = -1, int d = 0){
  visited[u] = true;
  dep[u] = fup[u] = d;
  bool isCutpt = false;
  for(int &v: adj[u]){
    if(v == p) continue;
    if(!visited[v]){
      dfs(v, u, d+1);
      fup[u] = min(fup[u], fup[v]);
      //Cut-Vertices
      if(fup[v] >= dep[u]){
        //Excluding root
        if(p != -1) isCutpt = true;
        //For root
        else if(adj[u].size() > 1)
          isCutpt = true;
```

```
      }
      //Bridges
      if(fup[v] > dep[u]) bridges.pb(mp(u,v));

    }
    else fup[u] = min(fup[u], dep[v]);
  }
  //Cut - Vertices
  if(isCutpt) cutPts.pb(u);
}
```

4.3 Bellman Ford

```
/*  Returns false if negative cycle detected
    Otherwise function returns true and dist[i]
    is length of shortest path from src to i.
    w[i][j] = weight from i to j.
    prev[i] is used to retrace path from src.
    Running time: O(|V|^3) */
const int N = 100;
int w[N][N], dist[N], prev[N], n;

bool BellmanFord (int src){
  fill(dist, dist+N, inf); reset(prev, -1);
  dist[src] = 0;

  for (int k = 0; k < n; k++)
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        if (dist[j] > dist[i] + w[i][j]){
          if (k == n-1) return false;
          dist[j] = dist[i] + w[i][j];
          prev[j] = i;
        }
  return true;
}
```

4.4 Dijkstra

```
const int N = 1e5 + 10;
vector<pii> adj[N];
bool vis[N] = {0}; int d[N];
void dijkstra(int v, int n){
  fill(d, d+n, inf); d[v] = 0;
  priority_queue<pii, vector<pii>,
                 greater<pii>> pq;
  pq.push(mp(d[v], v));
  while(!pq.empty()){
    int u = pq.top().ss; pq.pop();
    if(vis[u]) continue; vis[u] = true;
    for(auto &it: adj[u]) {
      if(d[it.ff] > d[u] + it.ss) {
        d[it.ff] = d[u] + it.ss;
        pq.push({d[it.ff], it.ff});
      }
    }
  }
}
```

4.5 Floyd Warshall

```
void floydwarshall(int n){
  for(int i = 0; i < n; i++)
    for(int j = 0; j < n; j++)
      dist[i][j] = graph[i][j];

  for(int k = 0; k < n; k++)
    for(int i = 0; i < n; i++)
      for(int j = 0; j < n; j++)
        dist[i][j] = min(dist[i][j],
                         dist[i][k] + dist[k][j]);
}
```

4.6 Kruskal(MST)

```
/*  graph stores the edges. Kruskal(n) for MST
    spanning_tree will store edges
    mincost will store MST weight */

struct edge {
  int u, v, w;
  edge(int u, int v, int w): u(u), v(v), w(w) {}
};
/* struct cmp {
    bool operator()(const edge& a, const edge& b)
    const { return a.w < b.w; } }
```

```
  set<edge, cmp> s;
  priority_queue<edge, vector<edge>, cmp> q; */
const int N = 1e5 + 10;
vector<edge> graph, spanning_tree;
int mincost = 0, dsu[N], sz[N];

bool edge_sort(const edge &a, const edge &b){
  return a.w < b.w; }

int find(int u){
  return dsu[u] == u ? u : dsu[u] = find(dsu[u]);}

void merge(edge e) {
  int u = e.u; int v = e.v;
  if(u == v) return; //self loops
  if(sz[v] > sz[u]) swap(u, v);
  dsu[v] = u; sz[u] += sz[v]; }

void Kruskal(int n) { //1-based
  sort(all(graph), edge_sort);
  for(int i = 1; i <= n; i++) {
    dsu[i] = i; sz[i] = 1;
  }
  for(edge e : graph) {
    int x = find(e.u); int y = find(e.v);
    if(x == y) continue;
    spanning_tree.pb(e); mincost += e.w; merge(e);
} }
```

4.7 Centroid Decompostion

```
// Decomposed Tree's root is decom[0].
// 1-based tree only. Add both side edges.

const int N = 1e5 + 10;
int n, sz[N];
set<int> graph[N];
vector<int> decom[N];

int get_size(int u, int par){
  sz[u] = 1;
  for(int &v: graph[u])
    if(v != par) sz[u] += get_size(v, u);
  return sz[u];
}

int get_centroid(int u, int par, int total){
  for(int &v: graph[u])
    if(v != par && 2 * sz[v] > total)
      return get_centroid(v, u, total);
  return u;
}

void decompose(int u, int par = 0){
  int total = get_size(u, -1);
  int centroid = get_centroid(u, -1, total);
  decom[par].pb(centroid);

  for(int &v: graph[centroid]) {
    graph[v].erase(centroid);
    decompose(v, centroid);
  }
}
```

4.8 Lowest Common Ancestor

```
const int N = 1e5 + 10, K = 18;
int par[N][K], h[N] = {0}; vector<int> graph[N];

void init(int n) {
  for(int i = 0; i < n + 5; i++)
    for(int j = 0; j < K; j++) par[i][j] = -1;
}

void dfs(int v, int p = -1) {
  par[v][0] = p;
  if(p != -1) h[v] = h[p] + 1;

  for(int i = 1; i < K; i++)
    if(par[v][i-1] != -1)
      par[v][i] = par[par[v][i-1]][i-1];

  for(int i: graph[v]) if(i != p) dfs(i, v);
}

int lca(int v, int u) {
```

```
  if(h[v] < h[u]) swap(v, u);

  for(int i = K-1; i >= 0; i--)
    if(par[v][i] + 1 and h[par[v][i]] >= h[u])
      v = par[v][i];

  if(v == u) return u;

  for(int i = K-1; i >= 0; i--) {
    if(par[v][i] != par[u][i]) {
      v = par[v][i]; u = par[u][i];
  } } return par[v][0]; }
```

5 Strings

5.1 Suffix arrays

```
// Suffix array construction in O(L log^2 L) time.
// Routine for computing the length of longest
// common prefix of two suffixes in O(log L) time.
// INPUT: string s
// OUTPUT: array suffix[] such that
//         suffix[i] = index (from 0 to L-1) of
//         substring s[i...L-1] in list of sorted
//         suffixes. That is, if we take inverse
//         of the permutation suffix[],
//         we get the actual suffix array.

// bobocel is the 0'th suffix in sorted order
//  obocel is the 5'th suffix in sorted order
//   bocel is the 1'st suffix in sorted order
// SuffixArray suffix("bobocel");
// vector<int> v = suffix.GetSuffixArray();
// Expected v: 0 5 1 6 2 3 4
// int lcp = suffix.LongestCommonPrefix(0, 2);
// Expected lcp: 2

struct SuffixArray {
  const int L;
  string s;
  vector<vector<int> > P;
  vector<pair<pair<int,int>,int> > M;

  SuffixArray(const string &s) : L(len(s)), s(s),
                  P(1, vector<int>(L, 0)), M(L){
    for(int i = 0; i < L; i++)
      P[0][i] = L==1 ? i : int(s[i]);

    int skp, lvl;
    for(skp = lvl = 1; skp < L; skp *= 2, lvl++){
      P.pb(vector<int>(L, 0));
      for(int i = 0; i < L; i++)
        M[i] = {{P[lvl-1][i], (i+skp < L ?
                  P[lvl-1][i + skp] : -1000)}, i};
      sort(all(M));
      for (int i = 0; i < L; i++)
      P[lvl][M[i].ss] = ((i && M[i].ff==M[i-1].ff)
                         ? P[lvl][M[i-1].ss] : i);
    }
  }

  vector<int> GetSuffixArray() { return P.back(); }

  // returns the length of longest common prefix
  // of s[i...L-1] and s[j...L-1]
  int LongestCommonPrefix(int i, int j) {
    int len = 0;
    if(i == j) return L - i;
    for(int k = len(P)-1; k>-1 && i<L && j<L; k--){
      if(P[k][i] == P[k][j]) {
        i += 1 << k; j += 1 << k; len += 1 << k;
    } }
    return len; }
};
```

5.2 Palindromic Tree

```
const int MAXN = 105000;

struct node {
    int nxt[26], len, suflink, num;
};

int len; string s; node tree[MAXN];
// node 1: root of len -1, node 2: root of len 0
int num;
```

```
int suff;            // max suffix palindrome
ll ans;

bool addLetter(int pos) {
  int cur = suff, curlen = 0;
  int let = s[pos] - 'a';

  while (true) {
    curlen = tree[cur].len;
    if(pos-curlen > 0 && s[pos-1-curlen] == s[pos])
      break;
    cur = tree[cur].suflink;
  }
  if(tree[cur].nxt[let]) {
    suff = tree[cur].nxt[let]; return false;
  }
  suff = ++num;
  tree[num].len = tree[cur].len + 2;
  tree[cur].nxt[let] = num;

  if(tree[num].len == 1) {
    tree[num].suflink = 2; tree[num].num = 1;
    return true;
  }
  while(true){
    cur = tree[cur].suflink;
    curlen = tree[cur].len;
    if(pos-curlen > 0 && s[pos-1-curlen]==s[pos]){
      tree[num].suflink=tree[cur].nxt[let]; break;
    }
  }
  tree[num].num = 1 + tree[tree[num].suflink].num;
  return true;
}

void initTree() {
  num = 2; suff = 2;
  tree[1].len = -1; tree[1].suflink = 1;
  tree[2].len = 0; tree[2].suflink = 1;
}

int main() {
  cin >> s;
  len = s.size();
  initTree();

  for (int i = 0; i < len; i++)
  { addLetter(i); ans += tree[suff].num; }

  cout << ans << endl;
  return 0;
}
```

5.3 Trie

```
/*  Using instructions:
    struct node *root = makenode();
    insertword(root, s); */
struct node {
  struct node *ptr[26]; char ch; bool fin;
};

struct node *makenode(char c = '#') {
  struct node *fnode = new node;
  fnode->fin = false;
  fnode->ch = c;
  for(int i = 0; i < 26; i++)
    fnode->ptr[i] = NULL;

  return fnode;
};

void insertword(struct node *root, string s) {
  struct node *temp = root;
  for(int i = 0; i < len(s); i++) {
    int id = s[i] - 'a';
    if(temp->ptr[id] == NULL)
      temp->ptr[id] = makenode(s[i]);

    temp = temp->ptr[id];
  } temp->fin = true;
}
```

5.4 Longest palindromic substring(Manacher)

```
// Maximal palindrome lengths centered around each
```

```
// position in a string(incl. positions between
// characters) and returns them in left-to-right
// order of centres. Linear time.
// Ex: "opposes" ->
// [0, 1, 0, 1, 4, 1, 0, 1, 0, 1, 0, 3, 0, 1, 0]
vector<int> fastLongestPalindromes(string str) {
  int i = 0, j, d, s, e, lLen, palLen=0;
  vector<int> res;
  while(i < len(str)) {
    if(i > palLen && str[i-palLen-1] == str[i]) {
      palLen += 2; i++; continue;
    }
    res.pb(palLen); s = len(res) - 2, e = s-palLen;
    bool b = true;
    for(j = s; j > e; j--) {
      d = j - e - 1;
      if(res[j] == d){palLen = d; b = 0; break;}
      res.pb(min(d, res[j]));
    }
    if(b){ palLen = 1; i++; }
  }
  res.pb(palLen); lLen = len(res);
  s = lLen-2; e = s-(2 * len(str) + 1 - lLen);
  for(i = s; i > e; i--) {
    d = i-e-1; res.pb(min(d, res[i])); }
  return res;}
```

5.5 Z & Prefix Function, Knuth Morris Pratt (KMP)

```
const int N = 1e5 + 10;
int z[N], p[N];

void zfunction(string s) {
  int l = 0, r = 0, n = len(s); z[0] = 0;
  for(int i = 1; i < n; i++) {
    if(i <= r) z[i] = min(z[i-l], r - i + 1);

    while(z[i]+i < n && s[z[i]] == s[z[i]+i])
      z[i]++;

    if(z[i] + i - 1 > r) {
      l = i; r = z[i] + i - 1;
} } }
```

```
void prefixfunction(string s) {
  p[0] = 0; int n = len(s);
  for(int i = 1; i < n; i++) {
    int j = p[i-1];
    while(j > 0 && s[j] != s[i]) j = p[j-1];
    if(s[j] == s[i]) p[i] = j+1;
} }

// Returns all positions where match is found
vector<int> kmpsearch(string text, string pat){
  int n = len(text), m = len(pat), i = 0, j = 0;
  prefixfunction(pat); vector<int> res;
  while(i < n) {
    if(pat[j] == text[i]){
      j++; i++;
    } if(j == m) {
      res.pb(i-j);
      j = p[j-1];
    } else if (i < n && pat[j] != text[i]){
      if(j!= 0) j = p[j-1];
      else i++;
    }
  } return res;
}
```

6 Data Structures

6.1 Binary Indexed Tree (BIT)

```
struct BIT {
  vector<int> bit;         // 1 based elements
  int n, k;
  BIT(int n) : n(n), bit(n+1), k(ceil(log2(n))) {}

  void add(int i, int val) { // update a[i] += val
    while(i <= n) {
      bit[i] += val; i += (i & -i);
  } }
  // For Range sum Update
  // add(bit1, a, val);
  // add(bit1, b+1, -val);
  // add(bit2, a, val*(a-1));
```

```
  // add(bit2, b+1, -val*b);

  int get(int i) {       // sum of a[1] .... a[i]
    int sum = 0;
    while(i > 0) {
      sum += bit[i]; i -= (i & -i);
    } return sum;
  }
  // For range sum query a[1] ... a[i]
  // return get(bit1, i) * i - get(bit2, i);

  // sum of a[l] .... a[r] (both inclusive)
  int get(int l, int r){ return get(r)-get(l-1); }

  void build(vector<int> &a, int n) {
    for(int i = 1; i <= n; i++) add(i, a[i]);
  }

  int lower_bound(int v) {
    int sum = 0, pos = 0;
    for(int i = k; i >= 0; i--) {
      int pw = (1 << i);
      // bit[pos + pw] <= v for upper_bound
      if(pos + pw <= n && sum + bit[pos + pw] < v)
        sum += bit[pos += pw];
    } return pos + 1;
  }
};
```

6.2 Binary Indexed Tree (2D BIT)

```
const int N = 1000, M = 1000;
int BIT[N+1][M+1], a[N+1][M+1];
int n, m;   // 1 <= x <= n, 1 <= y <= m

// Add val to cell (x, y)
void update(int x, int y, int val){
  for(int x1 = x; x1 <= n; x1 += (x1 & -x1))
    for(int y1 = y; y1 <= m; y1 += (y1 & -y1))
      BIT[x1][y1] += val;
}

// Get sum of (1, 1) to (x, y)
int sum(int x, int y){
  int sum = 0;
  for(int x1 = x; x1 > 0; x1 -= (x1 & -x1))
    for(int y1 = y; y1 > 0; y1 -= (y1 & -y1))
      sum += BIT[x1][y1];
  return sum;
}

// Get sum of rectangle with lower left cell
// (x1, y1) to upper right cell (x2, y2)
int sum(int x1, int y1, int x2, int y2){
  return sum(x2, y2) - sum(x2, y1-1) -
         sum(x1-1, y2) + sum(x1-1, y1-1);
}

// Build BIT. (1 based rows and columns)
void build(){
  for(int i = 1; i <= n; i++)
    for(int j = 1; j <= m; j++)
      update(i, j, a[i][j]);
}
```

6.3 Segment Tree with Lazy Propagation

```
const int N = 1e5 + 10;
int seg[4*N], lz[4*N], a[N], n;

#define lc 2*cur
#define rc 2*cur+1
#define mid (s+e)/2

inline int merge(int x, int y) { return x + y; }
inline int cnt(int l, int r) { return r - l + 1; }
// return 1 for RMQ

void build(int cur, int s, int e) {
  if(s == e) {
    seg[cur] = a[s];
    return;
  }
  build(lc, s, mid); build(rc, mid+1, e);
  seg[cur] = merge(seg[lc], seg[rc]);
}
```

```
void update(int cur, int s, int e,
                            int l, int r, int val){
  if(s != e){lz[lc] += lz[cur]; lz[rc] += lz[cur];}
  seg[cur] += cnt(s, e) * lz[cur]; lz[cur] = 0;
  if(e < l || s > r) return;
  if(l <= s && e <= r) {
    seg[cur] += cnt(s, e) * val;
    if(s != e) { lz[lc] += val; lz[rc] += val; }
    return;
  }
  update(lc, s, mid, l, r, val);
  update(rc, mid+1, e, l, r, val);
  seg[cur] = merge(seg[lc], seg[rc]);
}

int query(int cur, int s, int e, int l, int r) {
  if(s != e){lz[lc] += lz[cur]; lz[rc] += lz[cur];}
  seg[cur] += cnt(s, e) * lz[cur]; lz[cur] = 0;
  if(e < l || s > r) return 0; // -inf for RMQ
  if(l <= s && e <= r) return seg[cur];

  int q1 = query(lc, s, mid, l, r);
  int q2 = query(rc, mid+1, e, l, r);
  return merge(q1, q2);
}

void build() { build(1, 1, n); }
void update(int pos, int val)
{ update(1, 1, n, pos, pos, val); }
void update(int l, int r, int val)
{ update(1, 1, n, l, r, val); }
int query(int l, int r)
{ return query(1, 1, n, l, r); }

// For merge sort tree in complete overlap in query
// Replace lower and upper to remove equality sign
/*    Number of elements <= k in [l, r]
  upper_bound(all(seg[cur]), k)-seg[cur].begin();

      Number of elements >= k in [l, r]
  seg[cur].end()-lower_bound(all(seg[cur]), k);
*/
```

```
// merge(all(seg[lc]), all(seg[rc]),
//                          back_inserter(seg[cur]));
```

6.4 Sparse Table RMQ

```
const int N = 1e5 + 10, K = 25;    // log2(N)
int st[N][K+1], n, a[N];          // 0 based
void build() {
  for(int i = 0; i < n; i++)
    st[i][0] = a[i];

  for(int j = 1; j <= K; j++)
    for(int i = 0; i + (1 << j) - 1 < n; i++)
      st[i][j] = min(st[i][j-1],
                     st[i+(1<<(j-1))][j-1]);
}

int query(int l, int r) {    // RMQ, GCD
  int j = log2(r-l+1);
  return min(st[l][j], st[r-(1<<j)+1][j]);
}
```

7 Miscellaneous

7.1 Stable Marriage

```
vector<queue<int>> M; vector<vector<int>> F; int n;

vector<pii> SMP() {
  vector<int> hus(n+1, -1); queue<int> q;
  for(int i = 1; i <= n; i++) q.push(i);
  while(!q.empty()) {
    int husb = q.front(); q.pop();
    int wife = M[husb].front(); M[husb].pop();
    if(hus[wife] == -1) {
      hus[wife] = husb;
    } else if(F[wife][hus[wife]] > F[wife][husb]) {
      swap(hus[wife], husb); q.push(husb);
    } else q.push(husb);
  }
  vector<pii> res;
  for(int i = 1; i <= n; i++) {
```

```
    res.pb(mp(hus[i], i));
  } return res;
}

int main() {
  // n is number of marriages
  cin >> n; M.resize(n+1); F.resize(n+1);
  for(int i = 0; i < n; i++) {
    // First line is the woman in question
    int r; cin >> r;
    vector<int> cur(n+1);
    // Next n elements is husbands listed in rank
    // Earlier is better
    for(int j = 0; j < n; j++) {
      int x; cin >> x;
      cur[x] = j;
    } F[r] = cur;
  }
  // Same for men as women
  for(int i = 0; i < n; i++) {
    int r; cin >> r;
    queue<int> cur;
    for(int j = 0; j < n; j++) {
      int x; cin >> x;
      cur.push(x);
    } M[r] = cur;
  }
  auto ans = SMP();
  cout << ans << endl;
}
```

7.2 2-SAT

```
/*  init() initializes 2-SAT arrays.
    solve() checks if a valid solution exists
    mark[u] stores the bool value of the node u
    Use mark[] to recover the final solution
    Notes on Indexing Nodes :
    n = Number of Nodes(excluding negation)
    u = 2k, !u = 2k + 1
    Nodes are 0-indexed. [0, NUM_VERTICES) */

const int N = (int) 5e5 + 10;
int NUM, id, n, arr[N*2];
vector<int> adj[N*2];
bool mark[N*2];

inline bool dfs(int node){
  if (mark[node^1]) return false;
  if(mark[node]) return true;
  mark[node] = true;
  arr[id++] = node;
  for(int i: adj[node])
    if(!dfs(i)) return false;
  return true;
}

inline void init(){
  NUM = 2*n; reset(mark, 0);
  for(int i = 0; i < NUM; i++) adj[i].clear();
}

// Adds the clause (u or v)
inline void addOr(int u, int v){
  adj[u^1].push_back(v); adj[v^1].push_back(u);
}

// Adds the clause (u == v)
inline void addEquivalent(int u, int v){
  adj[u].push_back(v); adj[v].push_back(u);
  adj[u^1].push_back(v^1);
  adj[v^1].push_back(u^1);
}

// Adds the clause (u xor v)
inline void addXor(int u, int v){
  addOr(u, v); addOr(u^1, v^1);
}

// Forces variable (u) to be true
inline void forceTrue(int u){
  adj[u^1].push_back(u);
}

// Forces variable (u) to be false
inline void forceFalse(int u){forceTrue(u^1);}
```

```
// Adds the clause (u and v)
inline void addAnd(int u, int v){
  forceTrue(u); forceTrue(v);
}

inline void addImplication(int u, int v){
  adj[u].push_back(v);
}

// Returns true if a solution exists.
inline bool solve(){
  for(int i = 0; i < NUM; i++){
    sort(all(adj[i])); onlyunique(adj[i]);
  }
  for(int i = 0; i < NUM; i += 2){
    if((!mark[i]) && (!mark[i+1])){
      id = 0;
      if(!dfs(i)){
        while(id > 0) mark[arr[--id]] = false;
        if(!dfs(i+1)) return false;
      }
    }
  }
  return true;
}
```

7.3 LCS

```
/*  Calculates the lCS of two vectors
    Backtracks to find a single subsequence. */
const int N = 1000 + 10;
int A[N], B[N], dp[N][N] = {0}, n, m;
vector<int> res;

void backtrack(int i, int j) {
  if(!i || !j) return;
  if(A[i-1] == B[j-1]) {
    res.push_back(A[i-1]); backtrack(i-1, j-1);
  } else {
    if(dp[i][j-1] >= dp[i-1][j]) backtrack(i, j-1);
    else backtrack(i-1, j);
} }

void LCS() {
  for(int i=1; i<=n; i++)
    for(int j=1; j<=m; j++) {
      if(A[i-1]==B[j-1]) dp[i][j]=dp[i-1][j-1]+1;
      else dp[i][j] = max(dp[i-1][j], dp[i][j-1]);
    }
  backtrack(n, m); reverse(all(res)); }
```

7.4 LIS

```
// Given a list of numbers of length n, this
// returns the longest increasing subsequence.
//
// Running time: O(n log n)
//
//   INPUT: a vector of integers
//   OUTPUT: a vector containing the LIS

#define STRICTLY_INCREASNG

vll LongestIncreasingSubsequence(vll v) {
  vpll best; vll dad(v.size(), -1);

  for (int i = 0; i < v.size(); i++) {
#ifdef STRICTLY_INCREASNG
    pll item = mp(v[i], 0);
    auto it = lower_bound(all(best), item);
    item.ss = i;
#else
    pll item = mp(v[i], i);
    auto it = upper_bound(all(best), item);
#endif
    if (it == best.end()) {
      dad[i] = best.empty() ? -1 : best.back().ss;
      best.pb(item);
    } else {
      dad[i] = it==best.begin() ? -1:prev(it)->ss;
      *it = item;
    }
  }

  vll ret;
```

```
  for (int i = best.back().ss; i >= 0; i = dad[i])
    ret.pb(v[i]);
  reverse(all(ret)); return ret;
}
```

7.5 Ternary Search

```
// f is bitonic with a single global maximum
int lo = -1, hi = n;
while (hi - lo > 1) {
  int mid = (hi + lo) / 2;
  if (f(mid) > f(mid + 1)) hi = mid;
  else lo = mid;
} //lo + 1 is the answer
```

7.6 C ++ Template

```
#include <bits/stdc++.h>
#include <ext/pb_ds/detail/standard_policies.hpp>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef long long ll;
typedef long double ld;
template <typename T>
using ordered_set = tree<T, null_type, less<T>,
rb_tree_tag, tree_order_statistics_node_update>;
// order_of_key (k): No. of items < k (strict)
// find_by_order(k): K-th element(counting from 0)

#define pb emplace_back
#define mp make_pair
#define ff first
#define ss second
#define all(v) v.begin(), v.end()
#define len(a) int(a.size())
#define sqrt sqrtl
#define gcd __gcd
#define reset(a, val) memset(a, val, sizeof(a))
#define dbg trace
#define fnu for(int i = 0; i < n; i++)
#define fup(i, s, n) for(int i = s; i < n; i++)
#define initialise reset
#define rev(s) reverse(all(s))
// onlyunique(v) v.erase(unique(all(v)),v.end())
#define vll vector<int>
#define pll pair<int, int>
#define pii pll
#define fast_exp power
#define dbl ld
#define int ll
// copy(a, a+n, ostream_iterator<int>(cout, " "))
mt19937_64 rng(chrono::steady_clock::now()
                    .time_since_epoch().count());

#ifdef ONLINE_JUDGE
  #define endl '\n'
  #pragma comment(linker, "/stack:200000000")
  #pragma GCC optimize ("O3")
  #pragma GCC optimize ("O2")
  #pragma GCC optimize("Ofast")
  #pragma GCC optimize ("unroll-loops")
  #pragma GCC target("sse,sse2,sse3,ssse3,sse4,
                popcnt,abm,mmx,avx,tune=native")
  #define _CRT_SECURE_NO_WARNINGS
#endif

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

```
template<typename T> // replace set with vector
ostream& operator<< (ostream& os,const set<T>& v)
{ for (T u: v) os << u << " "; return os; }
template<typename T, typename S>
ostream& operator<<(ostream& os,const pair<T,S>& v)
{ os << v.ff << " " << v.ss; return os; }

struct pair_hash {
  inline size_t operator()(const pii & v) const {
     return v.ff * 31 + v.ss; }
};

const int mod = 1e9 + 7, inf = 2e18, ninf = -2e18;

int takemod(int a, int mod = mod){
  return ((a % mod) + mod) % mod; }

int power(int x, int y, int mod = mod) {
  int ans = 1;
  x = takemod(x, mod);
  while(y) {
    if(y % 2) ans = (x * ans) % mod;
    x = (x * x) % mod;
    y /= 1;
  } return ans;
}

int modinv(int a, int mod = mod){
  return takemod(power(a, mod-2, mod)); }

signed main() {
  ios_base::sync_with_stdio(false);
  cin.tie(0); cout.tie(0);
  time_t t1 = clock(), t2;
  #ifndef ONLINE_JUDGE
    freopen("input.txt", "r", stdin);
    freopen("output.txt", "w", stdout);
    freopen("error.txt", "w", stderr);
  #endif

  t2 = clock();
  #ifndef ONLINE_JUDGE
    cerr << "\ntime taken: " << t2-t1 << '\n';
  #endif // ONLINE_JUDGE
  return 0; }
```

7.7 C ++ Build System

```
{
  "cmd": ["bash", "-c",
          "g++ -std=c++14 '${file}' -o x && x" ],
  "file_regex":
        "^(..[^:]*):([0-9]+):?([0-9]+)?:? (.*)$",
  "working_dir": "${file_path}",
  "selector": "source.c, source.c++",
  "variants":
  [
    {
      "name": "Run",
      "cmd": ["bash", "-c",
            "g++ -std=c++14 '${file}' -o x && x"]
    }
  ]
}
```

7.8 Java BigInt

```
// Initialisation
BigInteger A, B, C;
A = BigInteger.valueOf(10);
B = new BigInteger("10");

// cer = 2 or 3
boolean isprime = A.isProbablePrime(cer);
// C is the next prime after A
C = A.nextProbablePrime();

// BigDecimal (init same as BigInt)
// C = A + B. MathContext is optional.
// Same for subtract and divide
C = A.add(B, new MathContext(6));
// C = A / B. Scale is no. of digits after decimal
C = A.divide(B, scale, new MathContext(6));
C = A.divide(B, scale, RoundingMode.CEILING);
// C = int(A / B)
```

```
C = A.divideToIntegralValue(B);
C = A.setScale(6);
// Precision of 4 places(including before decimal)
C = A.round(new MathContext(4));
```

7.9 Java Template

```
import java.util.*; import java.lang.*;
import java.io.*;

public class Main{
  public static void solve(PrintWriter out)
                            throws Exception{}

  public static void main(String[] args)
                            throws Exception{
    PrintWriter out = new PrintWriter(System.out);
    solve(out); out.close();
  }

  static class in{
   static BufferedReader b = new BufferedReader
        (new InputStreamReader(System.in));
   static StringTokenizer t =
                      new StringTokenizer("");
    static String next() throws Exception{
      while (!t.hasMoreTokens())
        t = new StringTokenizer(b.readLine());
      return t.nextToken();
    } static int nextInt() throws Exception{
      return Integer.parseInt(next());
    } // Same for Long and Double
  }
}
```

7.10 Python Fast I/O

```
from sys import stdin, stdout, setrecursionlimit

stdin = open('input.txt', 'r')
stdout = open('output.txt', 'w')

inp = int(stdin.readline())    # Similar to input()
stdout.write(str(inp))      # Use only with strings

setrecursionlimit(1000000)
```

7.11 Bit tricks

Clearing the lowest 1 bit: `x & (x - 1)`, all trailing 1's: `x & (x + 1)`
Setting the lowest 0 bit: `x | (x + 1)`
Enumerating subsets of a bitmask m:
`x=0; do { ...; x=(x+1+~m)&m; } while (x!=0);`
`__builtin_ctz/__builtin_clz` returns the number of trailing/leading zero bits.
`__builtin_popcount(unsigned x)` counts 1-bits (slower than table lookups). For 64-bit unsigned integer type, use the suffix '`ll`', i.e. `__builtin_popcountll`.

8 Combinatorics

8.1 Sums

$\sum_{k=0}^{n} k = n(n+1)/2$
$\sum_{k=a}^{b} k = (a+b)(b-a+1)/2$
$\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$
$\sum_{k=0}^{n} k^3 = n^2(n+1)^2/4$
$\sum_{k=0}^{n} k^4 = (6n^5 + 15n^4 + 10n^3 - n)/30$
$\sum_{k=0}^{n} k^5 = (2n^6 + 6n^5 + 5n^4 - n^2)/12$
$\sum_{k=0}^{n} x^k = (x^{n+1} - 1)/(x-1)$
$\sum_{k=0}^{n} kx^k = (x - (n+1)x^{n+1} + nx^{n+2})/(x-1)^2$
$1 + x + x^2 + \cdots = 1/(1-x)$
$(x+a)^{-n} = \sum_{k=0}^{\infty} \binom{-n}{k} x^k a^{-n-k}$

$\binom{n}{k} = \frac{n!}{(n-k)!k!}$
$\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$
$\binom{n+1}{k} = \frac{n+1}{n-k+1}\binom{n}{k}$
$\binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$
$\binom{n}{k} = \frac{n}{n-k}\binom{n-1}{k}$
$\binom{n}{k} = \frac{n-k+1}{k}\binom{n}{k-1}$
$12! \approx 2^{28.8}$
$20! \approx 2^{61.1}$

8.2 Binomial coefficients

Number of ways to pick a multiset of size k from n elements: $\binom{n+k-1}{k}$
Number of n-tuples of non-negative integers with sum s: $\binom{s+n-1}{n-1}$, at most s: $\binom{s+n}{n}$

Number of n-tuples of positive integers with sum s: $\binom{s-1}{n-1}$
Number of lattice paths from $(0,0)$ to (a,b), restricted to east and north steps: $\binom{a+b}{a}$

8.3 Multinomial theorem

$(a_1 + \cdots + a_k)^n = \sum \binom{n}{n_1,\ldots,n_k} a_1^{n_1} \ldots a_k^{n_k}$, where $n_i \geq 0$ and $\sum n_i = n$.

$$\binom{n}{n_1, \ldots, n_k} = M(n_1, \ldots, n_k) = \frac{n!}{n_1! \ldots n_k!}$$

$$M(a, \ldots, b, c, \ldots) = M(a + \cdots + b, c, \ldots) M(a, \ldots, b)$$

8.4 Catalan numbers

$C_n = \frac{1}{n+1}\binom{2n}{n}$. $C_0 = 1$, $C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i}$. $C_{n+1} = C_n \frac{4n+2}{n+2}$.
$C_0, C_1, \ldots = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900,$
C_n is the number of: properly nested sequences of n pairs of parentheses; rooted ordered binary trees with $n+1$ leaves; triangulations of a convex $(n+2)$-gon.

8.5 Derangements

Number of permutations of $n = 0, 1, 2, \ldots$ elements without fixed points is $1, 0, 1, 2, 9, 44, 265, 1854, 14833, \ldots$ Recurrence: $D_n = (n-1)(D_{n-1} + D_{n-2}) = nD_{n-1} + (-1)^n$. Corollary: number of permutations with exactly k fixed points is $\binom{n}{k} D_{n-k}$.

8.6 Stirling numbers of 1^{st} kind

$s_{n,k}$ is $(-1)^{n-k}$ times the number of permutations of n elements with exactly k permutation cycles. $|s_{n,k}| = |s_{n-1,k-1}| + (n-1)|s_{n-1,k}|$. $\sum_{k=0}^{n} s_{n,k} x^k = x^{\underline{n}}$
Number of permutations of n elements with exactly k cycles, all of length $\geq r$: $d_r(n,k) = (n-1)d_r(n-1,k) + (n-1)^{\underline{r-1}}\, d_r(n-r, k-1)$, $d_r(n,k) = 0$ for $n \leq kr - 1$, $d_r(n,1) = (n-1)!$.

8.7 Stirling numbers of 2^{nd} kind

$S_{n,k}$ is the number of ways to partition a set of n elements into exactly k non-empty subsets. $S_{n,k} = S_{n-1,k-1} + kS_{n-1,k}$. $S_{n,1} = S_{n,n} = 1$. $x^n = \sum_{k=0}^{n} S_{n,k}\, x^{\underline{k}}$

8.8 Bell numbers

B_n is the number of partitions of n elements. $B_0, \ldots = 1, 1, 2, 5, 15, 52, 203, \ldots$ $B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k = \sum_{k=1}^{n} S_{n,k}$. Bell triangle: $B_r = a_{r,1} = a_{r-1,r-1}$, $a_{r,c} = a_{r-1,c-1} + a_{r,c-1}$.

8.9 Eulerian numbers

$E(n,k)$ is the number of permutations with exactly k descents ($i : \pi_i < \pi_{i+1}$) / ascents ($\pi_i > \pi_{i+1}$) / excedances ($\pi_i > i$) / $k+1$ weak excedances ($\pi_i \geq i$). Formula: $E(n,k) = (k+1)E(n-1,k) + (n-k)E(n-1,k-1)$. $x^n = \sum_{k=0}^{n-1} E(n,k)\binom{x+k}{n}$.

9 Number Theory

9.1 Prime-counting function

$\pi(n) = |\{p \leq n : p \text{ is prime}\}|$. $n/\ln(n) < \pi(n) < 1.3n/\ln(n)$. $\pi(1000) = 168$, $\pi(10^6) = 78498$, $\pi(10^9) = 50\ 847\ 534$. n-th prime $\approx n \ln n$.

9.2 Fermat primes

A Fermat prime is a prime of form $2^{2^n} + 1$. The only known Fermat primes are 3, 5, 17, 257, 65537. A number of form $2^n + 1$ is prime only if it is a Fermat prime.

9.3 Mersenne primes

A Mersenne prime is a prime of form $2^n - 1$. Known Mersenne primes correspond to (necessarily prime) indexes 2, 3, 5, 7, 13, 17, 19, 31, 61, 89, 107,

127, 521, 607, 1279, 2203, 2281, 3217, 4253, 4423, 9689, 9941, 11213, 19937, 21701, 23209, 44497, 86243, 110503, 132049, 216091, 756839, 859433, 1257787, 1398269, 2976221, 3021377, 6972593, 13466917.

9.4 Perfect numbers

$n > 1$ is called perfect if it equals sum of its proper divisors and 1. Even n is perfect iff $n = 2^{p-1}(2^p - 1)$ and $2^p - 1$ is prime (Mersenne's). No odd perfect numbers are yet found.

9.5 Carmichael numbers

A positive composite n is a Carmichael number ($a^{n-1} \equiv 1 \pmod{n}$ for all a: $\gcd(a, n) = 1$), iff n is square-free, and for all prime divisors p of n, $p - 1$ divides $n - 1$.

9.6 Number/sum of divisors

$\tau(p_1^{a_1} \dots p_k^{a_k}) = \prod_{j=1}^{k}(a_j + 1)$. $\sigma(p_1^{a_1} \dots p_k^{a_k}) = \prod_{j=1}^{k} \frac{p_j^{a_j+1}-1}{p_j-1}$. Maximum number of divisors: 4, 12, 32, 64, 128, 240, 448, 768, 1344, 2304, 4032, 6720, 10752, 17280, 26880, 41472, 64512, 103680

9.7 Euler's phi function

$\phi(n) = |\{m \in N, m \le n, \gcd(m, n) = 1\}|$.
$\phi(mn) = \frac{\phi(m)\phi(n)\gcd(m,n)}{\phi(\gcd(m,n))}$. $\phi(p^a) = p^{a-1}(p - 1)$. $\sum_{d|n} \phi(d) = \sum_{d|n} \phi(\frac{n}{d}) = n$.

9.8 Fermat's theorem

$a^p \equiv a \pmod{p}$ if p is prime.
For any polynomial $f(x)$ with integer coefficients and prime p, $f(x)^{p^n} \equiv f(x^{p^n}) \pmod{p}$

9.9 Euler's theorem

$a^{\phi(n)} \equiv 1 \pmod{n}$, if $\gcd(a, n) = 1$.

9.10 Wilson's theorem

p is prime iff $(p - 1)! \equiv -1 \pmod{p}$.

9.11 Mobius function

$\mu(1) = 1$. $\mu(n) = 0$, if n is not squarefree. $\mu(n) = (-1)^s$, if n is the product of s distinct primes. Let f, F be functions on positive integers. If for all $n \in N$, $F(n) = \sum_{d|n} f(d)$, then $f(n) = \sum_{d|n} \mu(d)F(\frac{n}{d})$, and vice versa. $\phi(n) = \sum_{d|n} \mu(d)\frac{n}{d}$. $\sum_{d|n} \mu(d) = 1$.
If f is multiplicative, then $\sum_{d|n} \mu(d)f(d) = \prod_{p|n}(1 - f(p))$, $\sum_{d|n} \mu(d)^2 f(d) = \prod_{p|n}(1 + f(p))$.

9.12 Discrete logarithm problem

Find x from $a^x \equiv b \pmod{m}$. Can be solved in $O(\sqrt{m})$ time and space with a meet-in-the-middle trick. Let $n = \lceil\sqrt{m}\rceil$, and $x = ny - z$. Equation becomes $a^{ny} \equiv ba^z \pmod{m}$. Precompute all values that the RHS can take for $z = 0, 1, \dots, n - 1$, and brute force y on the LHS, each time checking whether there's a corresponding value for RHS.

9.13 Pythagorean triples

Integer solutions of $x^2 + y^2 = z^2$ All relatively prime triples are given by: $x = 2mn, y = m^2 - n^2, z = m^2 + n^2$ where $m > n, \gcd(m, n) = 1$ and $m \not\equiv n \pmod{2}$. All other triples are multiples of these. Equation $x^2 + y^2 = 2z^2$ is equivalent to $(\frac{x+y}{2})^2 + (\frac{x-y}{2})^2 = z^2$.

9.14 Postage stamps/McNuggets problem

Let a, b be relatively-prime integers. There are exactly $\frac{1}{2}(a-1)(b-1)$ numbers *not* of form $ax+by$ $(x, y \geq 0)$, and the largest is $(a-1)(b-1)-1 = ab-a-b$.

9.15 Fermat's two-squares theorem

Odd prime p can be represented as a sum of two squares iff $p \equiv 1 \pmod 4$. A product of two sums of two squares is a sum of two squares. Thus, n is a sum of two squares iff every prime of form $p = 4k+3$ occurs an even number of times in n's factorization.

10 Graph Theory

10.1 Euler's theorem

For any planar graph, $V - E + F = 1 + C$, where V is the number of graph's vertices, E is the number of edges, F is the number of faces in graph's planar drawing, and C is the number of connected components. Corollary: $V - E + F = 2$ for a 3D polyhedron.

10.2 Vertex covers and independent sets

Let M, C, I be a max matching, a min vertex cover, and a max independent set. Then $|M| \leq |C| = N - |I|$, with equality for bipartite graphs. Complement of an MVC is always a MIS, and vice versa. Given a bipartite graph with partitions (A, B), build a network: connect source to A, and B to sink with edges of capacities, equal to the corresponding nodes' weights, or 1 in the unweighted case. Set capacities of the original graph's edges to the infinity. Let (S, T) be a minimum s-t cut. Then a maximum(-weighted) independent set is $I = (A \cap S) \cup (B \cap T)$, and a minimum(-weighted) vertex cover is $C = (A \cap T) \cup (B \cap S)$.

10.3 Matrix-tree theorem

Let matrix $T = [t_{ij}]$, where t_{ij} is the number of multiedges between i and j, for $i \neq j$, and $t_{ii} = -\deg_i$. Number of spanning trees of a graph is equal to the determinant of a matrix obtained by deleting any k-th row and k-th column from T. If G is a multigraph and e is an edge of G, then the number $\tau(G)$ of spanning trees of G satisfies recurrence $\tau(G) = \tau(G - e) + \tau(G/e)$, when $G - e$ is the multigraph obtained by deleting e, and G/e is the contraction of G by e (multiple edges arising from the contraction are preserved.)

10.4 Euler tours

Euler tour in an undirected graph exists iff the graph is connected and each vertex has an even degree. Euler tour in a directed graph exists iff in-degree of each vertex equals its out-degree, and underlying undirected graph is connected. Construction:

```
doit(u):
  for each edge e = (u, v) in E, do: erase e, doit(v)
  prepend u to the list of vertices in the tour
```

10.5 Stable marriages problem

While there is a free man m: let w be the most-preferred woman to whom he has not yet proposed, and propose m to w. If w is free, or is engaged to someone whom she prefers less than m, match m with w, else deny proposal.

10.6 Stoer-Wagner's min-cut algorithm

Start from a set A containing an arbitrary vertex. While $A \neq V$, add to A the most tightly connected vertex ($z \notin A$ such that $\sum_{x \in A} w(x, z)$ is maximized.) Store cut-of-the-phase (the cut between the last added vertex and rest of the graph), and merge the two vertices added last. Repeat until the graph is contracted to a single vertex. Minimum cut is one of the cuts-of-the-phase.

10.7 Prufer code of a tree

Label vertices with integers 1 to n. Repeatedly remove the leaf with the smallest label, and output its only neighbor's label, until only one edge remains. The sequence has length $n-2$. Two isomorphic trees have the same sequence, and every sequence of integers from 1 and n corresponds to a tree. Corollary: the number of labelled trees with n vertices is n^{n-2}.

10.8 Erdos-Gallai theorem

A sequence of integers $\{d_1, d_2, \ldots, d_n\}$, with $n-1 \geq d_1 \geq d_2 \geq \cdots \geq d_n \geq 0$ is a degree sequence of some undirected simple graph iff $\sum d_i$ is even and $d_1 + \cdots + d_k \leq k(k-1) + \sum_{i=k+1}^{n} \min(k, d_i)$ for all $k = 1, 2, \ldots, n-1$.

11 Games

11.1 Grundy numbers

For a two-player, normal-play (last to move wins) game on a graph (V, E): $G(x) = \text{mex}(\{G(y) : (x, y) \in E\})$, where $\text{mex}(S) = \min\{n \geq 0 : n \notin S\}$. x is losing iff $G(x) = 0$.

11.2 Sums of games

- *Player chooses a game and makes a move in it* Grundy number of a position is xor of grundy numbers of positions in summed games.
- *Player chooses a non-empty subset of games (possibly, all) and makes moves in all of them* A position is losing iff each game is in a losing position.
- *Player chooses a proper subset of games (not empty and not all), and makes moves in all chosen ones.* A position is losing iff grundy numbers of all games are equal.
- *Player must move in all games, and loses if can't move in some game* A position is losing if any of the games is in a losing position.

11.3 Misère Nim

A position with pile sizes $a_1, a_2, \ldots, a_n \geq 1$, not all equal to 1, is losing iff $a_1 \oplus a_2 \oplus \cdots \oplus a_n = 0$ (like in normal nim.) A position with n piles of size 1 is losing iff n is *odd*

12 Math

12.1 Taylor series

$f(x) = f(a) + \frac{x-a}{1!}f'(a) + \frac{(x-a)^2}{2!}f^{(2)}(a) + \cdots + \frac{(x-a)^n}{n!}f^{(n)}(a) + \ldots$.
$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$
$\ln x = 2(a + \frac{a^3}{3} + \frac{a^5}{5} + \ldots)$, where $a = \frac{x-1}{x+1}$. $\ln x^2 = 2\ln x$.
$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots$, $\arctan x = \arctan c + \arctan \frac{x-c}{1+xc}$ (e.g c=.2)
$\pi = 4\arctan 1$, $\pi = 6\arcsin \frac{1}{2}$

12.2 List of Primes

1e5	3	19	43	49	57	69	103	109	129	151	153
1e6	33	37	39	81	99	117	121	133	171	183	
1e7	19	79	103	121	139	141	169	189	223	229	
1e8	7	39	49	73	81	123	127	183	213		