Índice

hute-UBA - Reference

1. STL - Reference

#include <algorithm> #include <numeric>

Algo	**Params**	**Funcion**
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	*void* ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	*void* llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	*it* al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	*bool* esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	*it* encuentra i $\in$[f,l) tq. i=elem, pred(i), i$\in$[f2,l2)
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca [f2,l2) $\in$ [f,l)
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	*it* min, max de [f,l]
lexicographical_compare	f1,l1,f2,l2	*bool* con [f1,l1]¡[f2,l2]
next/prev_permutation	f,l	deja en [f,l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, ...) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f,l), hace un heap de [f,l)
is_heap	f,l	*bool* es [f,l) un heap
accumulate	f,l,i,[op]	$T = \sum$/oper de [f,l)
inner_product	f1, l1, f2, i	T = i + [f1, l1) . [f2, ...)
partial_sum	f, l, r, [op]	r+i = $\sum$/oper de [f,f+i] $\forall i \in$[f,l)
power	e, i, op	$T = e^n$

1.1. HashMap

```
struct hash_string {
  hash<char*> h;
  size_t operator()(const string &s) const { return h(s.c_str()); }
};
void hash_demo() {
  hash_map<string, string, hash_string> foo;
  foo["foo"] = "bar";
  cout << foo["foo"] << endl;
}
```

1.2. Template

```
#include <cassert>
#include <cctype>
#include <climits>
#include <cmath>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#include <ext/numeric>
#include <ext/hash_set>
#include <ext/hash_map>
#include <functional>
#include <iomanip>
#include <iostream>
#include <limits>
#include <list>
#include <map>
#include <numeric>
#include <queue>
#include <set>
#include <sstream>
#include <stack>
#include <string>
#include <vector>
using namespace std;

#define VAR(a,b) typeof(b) a=(b)
#define FOR(i,a,b) for(int i=(a);i<(b);i++)
#define REP(i,n) FOR(i,0,n)
#define FOREACH(it,c) for(VAR(it,(c).begin());it!=(c).end();++it)
#define ALL(c) (c).begin(),(c).end()
#define SIZE(c) ((int)(c).size())

#define DEBUG(x) cerr<<#x<<'='<<x<<endl
template<class T> ostream& operator<<(ostream& o,const vector<T>& c){ FOREACH(p,
    c) o<<*p<<'␣'; return o; }
```

```
template<class T> ostream& operator<<(ostream& o,const vector<vector<T> >& c){
    FOREACH(p,c) o<<*p<<endl; return o; }
template<class U,class V> ostream& operator<<(ostream& o,const pair<U,V>& p){
    return o<<'('<<p.first<<','<<p.second<<')'; }

typedef long long LL;

typedef vector<int> VI;
typedef vector<VI> VVI;
typedef pair<int,int> PII;
typedef vector<LL> VLL;
typedef vector<VLL> VVLL;
typedef vector<string> VS;
```

2. Geom

2.1. Point in Poly

```
struct Point {
  int x,y;
  Point() {}
  Point(int x_,int y_) : x(x_), y(y_) {}
};
Point operator-(const Point& A,const Point& B){ return Point(A.x-B.x,A.y-B.y); }
int det(const Point& A,const Point& B){ return A.x*B.y - A.y*B.x; }
int triarea(const Point& A,const Point& B,const Point& C){ return det(B-A,C-A);
    }

bool point_in_poly(vector<Point>&v,Point p){
  bool ret = false;
  REP(i,SIZE(v)){
    int j = (i+1) % SIZE(v);
    int lo=i, hi=j;
    if ( v[lo].y > v[hi].y ) swap(lo,hi);
    if ( v[lo].y <= p.y&&p.y < v[hi].y ) {
      if ( triarea(v[lo],v[hi],p) > 0 ) ret = !ret;
    }
  } return ret;
}
```

2.2. Poly Area

```
struct Point{
  int x, y;
  Point(int x_,int y_) : x(x_),y(y_) {}
};
int det(const Point& A,const Point& B) { return A.x*B.y - A.y*B.x; }

double area(const vector<Point>& p) {
  int n = p.size(), ret = 0;
  REP(i,n){
    int j = (i+1) % n;
    ret += det(p[i],p[j]);
  }
  return ret / 2.0;
}
```

2.3. Convex Hull O(n*log(n))

```
struct Point{
  int x,y;
  Point(){}
  Point(int x,int y) : x(x),y(y){}
  int lensqr() { return x*x + y*y; }
};
Point operator-(Point a,Point b){ return Point(a.x-b.x,a.y-b.y); }
int det(Point a,Point b){ return a.x*b.y-a.y*b.x; }
int dir(Point a,Point b,Point c){ return det(b-a,c-a); }
bool ltY(const Point& a,const Point& b) { return a.y < b.y || (a.y == b.y && a.x
      < b.x); }

Point REF;
bool ltAngle(const Point& a_,const Point& b_){
  Point a(a_ - REF);
  Point b(b_ - REF);
  return det(a,b) > 0 || (det(a,b) == 0 && a.lensqr() > b.lensqr());
}
vector<Point> convexhull(vector<Point> Q){
  if(Q.size() < 3) return Q;
  vector<Point>::iterator mini = min_element(ALL(Q),ltY);
  REF = *mini;
  Q.erase(mini);
  sort(ALL(Q),ltAngle);
  vector<Point> S;
  S.push_back(REF);
  S.push_back(Q[0]);
  S.push_back(Q[1]);
  FOR(i,2,SIZE(Q)){
    while(dir(S[SIZE(S)-2],S[SIZE(S)-1],Q[i]) < 0){  // (con "<= 0" no deja
        puntos colineales)
      S.pop_back();
      assert(S.size() >= 2);
    }
```

```
    S.push_back(Q[i]);
  }
  return S;
}
```

2.4. Circulo mínimo - PPP

```
usa: algorithm, cmath, vector, pto (con < e ==)
usa: sqr, dist2(pto,pto), tint
typedef double tipo;
typedef vector<pto> VP;
struct circ { tipo r; pto c; };
#define eq(a,b) (fabs(a-b)<0.00000000000001)
circ deIni(VP v){ //l.size()<=3
  circ r;  sort(v.begin(), v.end()); unique(v.begin(), v.end());
  switch(v.size()) {
    case 0: r.c.x=r.c.y=0; r.r = -1; break;
    case 1: r.c=v[0]; r.r=0; break;
    case 2: r.c.x=(v[0].x+v[1].x)/2.0;
        r.c.y=(v[0].y+v[1].y)/2.0;
        r.r=dist2(v[0], r.c); break;
    default: {
      tipo A = 2.0 * (v[0].x-v[2].x);tipo B = 2.0 * (v[0].y-v[2].y);
      tipo C = 2.0 * (v[1].x-v[2].x);tipo D = 2.0 * (v[1].y-v[2].y);
      tipo R = sqr(v[0].x)-sqr(v[2].x)+sqr(v[0].y)-sqr(v[2].y);
      tipo P = sqr(v[1].x)-sqr(v[2].x)+sqr(v[1].y)-sqr(v[2].y);
      tipo det = D*A-B*C;
      if(eq(det, 0)) {swap(v[1],v[2]); v.pop_back(); return deIni(v);}
      r.c.x = ( D*R-B*P)/det;
      r.c.y = (-C*R+A*P)/det;
      r.r = dist2(v[0],r.c);
    }
  }
  return r;
}
circ minDisc(VP::iterator ini,VP::iterator fin,VP& pIni){
  VP::iterator ivp;
  int i,cantP=pIni.size();
  for(ivp=ini,i=0;i+cantP<2 && ivp!=fin;ivp++,i++) pIni.push_back(*ivp);
  circ r = deIni(pIni);
  for(;i>0;i--) pIni.pop_back();
  for(;ivp!=fin;ivp++) if (dist2(*ivp, r.c) > r.r){
    pIni.push_back(*ivp);
    if (cantP<2) r=minDisc(ini,ivp,pIni);
    else r=deIni(pIni);
    pIni.pop_back();
  }
  return r;
}
circ minDisc(VP ps){ //ESTA ES LA QUE SE USA
  random_shuffle(ps.begin(),ps.end()); VP e;
  circ r = minDisc(ps.begin(),ps.end(),e);
  r.r=sqrt(r.r); return r;
};
```

2.5. Máximo rectángulo entre puntos - PPP

```
usa: vector, map, algorithm
struct pto {
  tint x,y ;bool operator<(const pto&p2)const{
    return (x==p2.x)?(y<p2.y):(x<p2.x);
  }
};
bool us[10005];
vector<pto> v;
tint l,w;
tint maxAr(tint x, tint y,tint i){
  tint marea=0;
  tint arr=0,aba=w;
  bool partido = false;
  for(tint j=i;j<(tint)v.size();j++){
    if(x>=v[j].x)continue;
    tint dx = (v[j].x-x);
    if(!partido){
      tint ar = (aba-arr) * dx;marea>?=ar;
    } else {
      tint ar = (aba-y) * dx;marea>?=ar;
      ar = (y-arr) * dx;marea>?=ar;
    }
    if(v[j].y==y)partido=true;
    if(v[j].y< y)arr>?=v[j].y;
    if(v[j].y> y)aba<?=v[j].y;
  }
  return marea;
}
tint masacre(){
  fill(us,us+10002,false);
  pto c;c.x=0;c.y=0;v.push_back(c);c.x=l;c.y=w;v.push_back(c);
  tint marea = 0;
  sort(v.begin(),v.end());
  for(tint i=0;i<(tint)v.size();i++){
    us[v[i].y]=true;
    marea>?=maxAr(v[i].x,v[i].y,i);
  }
```

```
  for(tint i=0;i<10002;i++)if(us[i])marea>?=maxAr(0,i,0);
  return marea;
}
```

2.6. Máxima cantidad de puntos alineados - PPP

```
usa: algorithm, vector, map, set, forn, forall(typeof)
struct pto {
  tipo x,y;
  bool operator<(const pto &o)const{
    return (x!=o.x)?(x<o.x):(y<o.y);
  }
};
struct lin{
  tipo a,b,c;//ax+by=c
  bool operator<(const lin& l)const{
    return a!=l.a?a<l.a:(b!=l.b?b<l.b:c<l.c);
  }
};
typedef vector<pto> VP;
tint mcd(tint a, tint b){return (b==0)?a:mcd(b, a%b);}
lin linea(tipo x1, tipo y1, tipo x2, tipo y2){
  lin l;
  tint d = mcd(y2-y1, x1-x2);
  l.a = (y2-y1)/d;
  l.b = (x1-x2)/d;
  l.c = x1*l.a + y1*l.b;
  return l;
}
VP v;
typedef map<lin, int> MLI;
MLI cl;
tint maxLin(){
  cl.clear();
  sort(v.begin(), v.end());
  tint m=1, acc=1;
  forn(i, ((tint)v.size())-1){
    acc=(v[i]<v[i+1])?1:(acc+1);
    m>?=acc;
  }
  forall(i, v){
    set<lin> este;
    forall(j, v){
    if(*i<*j||*j<*i)
      este.insert(linea(i->x, i->y, j->x, j->y));
    }
    forall(l, este)cl[*l]++;
  }
  forall(l, cl){
    m>?= l->second;
  }
  return m;
}
```

2.7. Centro de masa y area de un polígono - PPP

```
struct ptoD { double x,y; };
ptoD centro(const poly& p) {
  tint a = 0; ptoD r; r.x=r.y=0; tint l = p.size()-1;
  forn(i,l-1) {
    tint act = area3(p[i], p[i+1], p[l]);
    pto pact = bariCentroPor3(p[i], p[i+1], p[l]);
    r.x += act * pact.x; r.y += act * pact.y; a += act;
  } r.x /= (3 * a); r.y /= (3 * a); return r;
}
```

2.8. Par de puntos mas cercano O(n*log(n))

```
const int INF = 2000000000;

struct Point{ int x,y; };

bool cmpX(const Point& a,const Point& b){ return a.x<b.x || a.x==b.x && a.y<b.y;
     }
bool cmpY(const Point& a,const Point& b){ return a.y<b.y || a.y==b.y && a.x<b.x;
     }

int dist2(int x1,int y1,int x2,int y2){ return (x1-x2)*(x1-x2)+(y1-y2)*(y1-y2);
    }

int go(vector<Point>& A,int l,int h,pair<Point,Point>& r){
  int mindist;
  if(h-l+1<=3){
    sort(&A[l],&A[h+1],cmpY);
    mindist=INF;
    for(int i=l;i<=h;i++)
    for(int j=i+1;j<=h;j++){
      int d=dist2(A[i].x,A[i].y,A[j].x,A[j].y);
      if(mindist>d){ mindist=d; r=make_pair(A[i],A[j]); }
    }
    return mindist;
  }
  int m,d1,d2;
  pair<Point,Point> r1,r2;
```

```
  m=(l+h)/2;
  int Xmid=A[m].x;
  if((d1=go(A,l,m,r1))<(d2=go(A,m+1,h,r2))){
    r=r1;
    mindist=d1;
  }else{
    r=r2;
    mindist=d2;
  }
  inplace_merge(&A[l],&A[m+1],&A[h+1],cmpY);
  vector<int> B;
  for(int i=l;i<=h;i++){
    if((Xmid-A[i].x)*(Xmid-A[i].x)<=mindist) B.push_back(i);
  }
  for(int i=0;i<B.size();i++){
    for(int j=i+1;j<=i+8 && j<B.size();j++){
      d1=dist2(A[B[i]].x,A[B[i]].y,A[B[j]].x,A[B[j]].y);
      if(mindist>d1){
        mindist=d1;
        r=make_pair(A[B[i]],A[B[j]]);
      }
    }
  }
  return mindist;
}
pair<Point,Point> closestPair(vector<Point>& A){
  pair<Point,Point> r;
  if(A.size()>1){
    sort(A.begin(),A.end(),cmpX);
    go(A,0,A.size()-1,r);
  }
  return r;
}
```

2.9. CCW - PPP

```
struct point {tint x, y;};
int ccw(const point &p0, const point &p1, const point &p2){
    tint dx1, dx2, dy1, dy2;
    dx1 = p1.x - p0.x; dy1 = p1.y - p0.y;
    dx2 = p2.x - p0.x; dy2 = p2.y - p0.y;
    if (dx1*dy2 > dy1*dx2) return +1;
    if (dx1*dy2 < dy1*dx2) return -1;
    if ((dx1*dx2 < 0) || (dy1*dy2 < 0)) return -1;
    if ((dx1*dx1+dy1*dy1) < (dx2*dx2+dy2*dy2))return +1;
    return 0;
}
```

2.10. Sweep Line - PPP

```
struct pto { tint x,y; bool operator<(const pto&p2)const{
  return (y==p2.y)?(x<p2.x):(y<p2.y);
}};
struct slp{ tint x,y,i;bool f; bool operator<(const slp&p2)const{
  if(y!=p2.y)return y<p2.y;
  if(x!=p2.x)return x<p2.x;
  if(f!=p2.f)return f;
  return i<p2.i;
}};
slp p2slp(pto p,tint i){slp q;q.x=p.x;q.y=p.y;q.i=i;return q;}
tint area3(pto a,pto b,pto c){
  return (b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x);
}
tint giro(pto a,pto b,pto c){
  tint a3=area3(a,b,c);
  if(a3<0) return -1; if(a3>0)return 1;
  return 0;
}
bool inter(pair<pto,pto> a, pair<pto,pto> b){
  pto p=a.first,q=a.second,r=b.first,s=b.second;
  if(q<p)swap(p,q);if(s<r)swap(r,s);
  if(r<p){swap(p,r);swap(q,s);}
  tint a1=giro(p,q,r),a2=giro(p,q,s);
  if(a1!=0 || a2!=0){
    return (a1!=a2) && (giro(r,s,p)!=giro(r,s,q));
  } else {
    return !(q<r);
  }
}
tint cant_intersec(vector<pair<pto,pto> >&v){
  tint ic=0;
  set<slp> Q; list<tint> T;
  for(tint i=0;i<(tint)v.size();i++){
    slp p1=p2slp(v[i].first,i);slp p2=p2slp(v[i].second,i);
    if(p2<p1)swap(p1,p2);
    p1.f=true;p2.f=false;
    Q.insert(p1);Q.insert(p2);
  }
  while(Q.size()>0){
    slp p = *(Q.begin());Q.erase(p);
    if(p.f){
      for(list<tint>::iterator it=T.begin();it!=T.end();it++)
        if(inter(v[*it],v[p.i]))ic++;
      T.push_back(p.i);
```

```
    } else {
      T.erase(find(T.begin(),T.end(),p.i));
    }
  }
  return ic;
}
```

2.11. Punto de Intersección entre dos rectas y test de intersección entre segmentos

```
typedef int coord;

struct Point{
  coord x, y;
  Point() {}
  Point(coord x_, coord y_) : x(x_),y(y_) {}
  coord lensqr() { return x*x + y*y; }
};

typedef vector<Point> Polygon;

coord det(Point A, Point B) { return A.x*B.y - A.y*B.x; }
Point operator-(Point A, Point B) { return Point(A.x - B.x, A.y - B.y); }
Point operator+(Point A, Point B) { return Point(A.x + B.x, A.y + B.y); }
Point operator*(coord c, Point A) { return Point(c*A.x, c*A.y); }
coord operator*(Point A, Point B) { return A.x*B.x + A.y*B.y; }

bool operator<(const Point& A,const Point &B) {
  return A.x < B.x || (A.x == B.x && A.y < B.y);
}

// signed area
// requires: Point, operator-
coord area(Point A, Point B, Point C) { return det(B-A, C-A); }

// touching points are consider an intersection.
// requires: Point, area;
bool cut(Point p1, Point p2, Point q1, Point q2) {
  // bbox test
  if(max(p1.x, p2.x) < min(q1.x, q2.x)) return false;
  if(max(p1.y, p2.y) < min(q1.y, q2.y)) return false;
  if(min(p1.x, p2.x) > max(q1.x, q2.x)) return false;
  if(min(p1.y, p2.y) > max(q1.y, q2.y)) return false;
  // intersection test
  coord a1 = area(p1, p2, q1);
  coord a2 = area(p1, p2, q2);
  coord b1 = area(q1, q2, p1);
  coord b2 = area(q1, q2, p2);
  if(a1 > 0 && a2 > 0) return false;
  if(a1 < 0 && a2 < 0) return false;
  if(b1 > 0 && b2 > 0) return false;
  if(b1 < 0 && b2 < 0) return false;
  // note: coincident lines return true, and are handled by bbox test.
  return true;
}

// Only use if lines are not parallel, unless they are coincident
// Otherwise it will return an arbitrary endpoint
// You can determine if the lines are parallel using
// cross(p2 - p1, q2 - q1) == 0, and if they are whether they lie in the
// same line with cross(p2 - p1, q1 - p1) == 0
bool cut_point(Point p1, Point p2, Point q1, Point q2, double& rx,double& ry) {
  Point d1 = p2 - p1;
  Point d2 = q2 - q1;
  coord d = det(d1, d2);
  if(d == 0) return false;  // parallel lines
  double t = det(d2, p1 - q1) / (double) d;
  rx = p1.x + t*d1.x;
  ry = p1.y + t*d1.y;
  return true;
}
```

2.12. Distancia entre segmentos

```
tdbl dist(pto p, seg s){
  tdbl a = fabs(tdbl(pc(s.f, s.s, p)));
  tdbl b = hypot(s.f.x-s.s.x,s.f.y-s.s.y),h=a/b, c = hypot(b, h);
  tdbl d1 = hypot(s.f.x-p.x,s.f.y-p.y), d2 = hypot(s.s.x-p.x,s.s.y-p.y);
  if(b<1e-10 || c <= d1 || c <= d2)return min(d1, d2); else return h;
}
tdbl dist(seg a, seg b){
  return (inter(a, b))?0.0:min(min(dist(a.f, b), dist(a.s, b)), min(dist(b.f, a)
      , dist(b.s, a)));
}
```

2.13. Line

```
const double EPS = 1e-10;

bool isEqual(double x,double y){ return fabs(x-y)<EPS; }

struct Point{
  double x,y;
```

```
  Point(){}
  Point(double x,double y):x(x),y(y){}
};

// Ecuacion: ax + by + c = 0
// y = m*x + b -> a=-m, b=1 y c=-b
struct Line {
  double a,b,c;
  Line(){}
  Line(Point p1,Point p2){
    if(p1.x==p2.x){
      a=1; b=0; c=-p1.x;
    }else{
      a=-(p1.y-p2.y)/(p1.x-p2.x); b=1; c=-a*p1.x-p1.y;
    }
  }
  Line(Point p,double m){ a=-m; b=1; c=m*p.x-p.y; }
};

bool parallel(Line l1,Line l2){
  return isEqual(l1.a,l2.a) && isEqual(l1.b,l2.b);
}
bool sameLine(Line l1,Line l2){
  return parallel(l1,l2) && isEqual(l1.c,l2.c);
}
bool intersection(Line l1,Line l2,Point& p){
  if(parallel(l1,l2)) return false;
  if(isEqual(l1.b,0)){
    p.x=-l1.c;
    p.y=-l2.a*p.x-l2.c;
  }else{
    if(isEqual(l2.b,0)){
      p.x=-l2.c;
    }else{
      p.x=(l1.c-l2.c)/(l2.a-l1.a);
    }
    p.y=-l1.a*p.x-l1.c;
  }
  return true;
}
Point closestPoint(Point p, Line l) {
  Point r;
  // vertical
  if(isEqual(l.b, 0.0)) { r.x = -l.c; r.y = p.y; return r; }
  // horizontal
  if(isEqual(l.a, 0.0)) { r.x = p.x; r.y = -l.c; return r; }

  Line per(p, 1/l.a);
  intersection(l, per, r);
  return r;
}
```

2.14. Cuentitas

```
const pto cero = punto(0,0);
pto suma(pto o, pto s, tipo k){
  return punto(o.x + s.x * k, o.y + s.y * k);
}
pto sim(pto p, pto c){return suma(c, suma(p,c,-1), -1);}
tipo pc(pto a, pto b, pto o){ // o es origen
  return (b.y-o.y)*(a.x-o.x)-(a.y-o.y)*(b.x-o.x);
}
tipo pe(pto a, pto b, pto o){
  return (b.x-o.x)*(a.x-o.x)+(b.y-o.y)*(a.y-o.y);
}
//#define feq(a,b) (fabs((a)-(b))<0.000000000001) para interseccion
#define feq(a,b) (fabs((a)-(b))<0.000000001)
tipo zero(tipo t){return feq(t,0.0)?0.0:t;}
bool alin(pto a, pto b, pto c){  return feq(0, pc(a,b,c));}
bool perp(pto a1, pto a2, pto b1, pto b2){
  return feq(0, pe(suma(a1, a2, -1.0), suma(b1, b2, -1.0), cero));
}
bool hayEL(tipo A11, tipo A12, tipo A21, tipo A22){
  return !feq(0.0, A22*A11-A12*A21);
}
pto ecLineal(tipo A11, tipo A12, tipo A21, tipo A22, tipo R1, tipo R2){
  tipo det = A22*A11-A12*A21;
  return punto((A22*R1-A12*R2)/det,(A11*R2-A21*R1)/det);
}
lin linea(pto p1, pto p2){
  lin l;
  l.b = p2.x-p1.x;
  l.a = p1.y-p2.y;
  l.c = p1.x*l.a + p1.y*l.b;
  return l;
}
bool estaPL(pto p, lin l){return feq(p.x * l.a + p.y * l.b, l.c);}
bool estaPS(pto p, pto a, pto b){
  return feq(dist(p,a)+dist(p,b),dist(b,a));
}
lin bisec(pto o, pto a, pto b){
  tipo da = dist(a,o);
  return linea(o, suma(a, suma(b,a,-1.0), da / (da+dist(b,o))));
```

```
}
bool paral(lin l1, lin l2){return !hayEL(l1.a, l1.b, l2.a, l2.b);}
bool hayILL(lin l1, lin l2){ //!paralelas || misma
  return !paral(l1,l2)|| !hayEL(l1.a, l1.c, l2.a, l2.c);
}
pto interLL(lin l1, lin l2){//li==l2->pincha
  return ecLineal(l1.a, l1.b, l2.a, l2.b, l1.c, l2.c);
}
bool hayILS(lin l, pto b1, pto b2){
  lin b = linea(b1,b2);
  if(!hayILL(l,b))return false;
  if(estaPL(b1,l))return true;
  return estaPS(interLL(l,b), b1,b2);
}
tipo distPL(pto p, lin l){
  return fabs((l.a * p.x + l.b * p.y - l.c)/sqrt(sqr(l.a)+sqr(l.b)));
}
tipo distPS(pto p, pto a1, pto a2){
  tipo aa = sqrd(a1, a2);
  tipo d = distPL(p, linea(a1, a2));
  tipo xx = aa+sqr(d);
  tipo a1a1 = sqrd(a1, p);
  tipo a2a2 = sqrd(a2, p);
  if(max(a1a1, a2a2) > xx){
    return sqrt(min(a1a1, a2a2));
  }else{
    return d;
  }
}
pto circunCentro(pto a, pto b, pto c){
  tipo A = 2.0 * (a.x-c.x);tipo B = 2.0 * (a.y-c.y);
  tipo C = 2.0 * (b.x-c.x);tipo D = 2.0 * (b.y-c.y);
  tipo R = sqr(a.x)-sqr(c.x)+sqr(a.y)-sqr(c.y);
  tipo P = sqr(b.x)-sqr(c.x)+sqr(b.y)-sqr(c.y);
  return ecLineal(A,B,C,D,R,P);
}
pto ortoCentro(pto a, pto b, pto c){
  pto A = sim(a, ptoMedio(b,c));
  pto B = sim(b, ptoMedio(a,c));
  pto C = sim(c, ptoMedio(b,a));
  return circunCentro(A,B,C);
}
pto inCentro(pto a, pto b, pto c){
  return interLL(bisec(a, b, c), bisec(b, a, c));
}
pto rotar(pto p, pto o, tipo s, tipo c){
  //gira cw un angulo de sin=s, cos=c
  return punto(
    o.x + (p.x - o.x) * c + (p.y - o.y) * s,
    o.y + (p.x - o.x) * -s + (p.y - o.y) * c
  );
}
bool hayEcCuad(tipo a, tipo b, tipo c){//a*x*x+b*x+c=0 tiene sol real?
  if(feq(a,0.0))return false;
  return zero((b*b-4.0*a*c)) >= 0.0;
}
pair<tipo, tipo> ecCuad(tipo a, tipo b, tipo c){//a*x*x+b*x+c=0
  tipo dx = sqrt(zero(b*b-4.0*a*c));
  return make_pair((-b + dx)/(2.0*a),(-b - dx)/(2.0*a));
}
bool adentroCC(circ g, circ c){//c adentro de g sin tocar?
  return g.r > dist(g.c, c.c) + c.r || !feq(g.r, dist(g.c, c.c) + c.r);
}
bool hayICL(circ c, lin l){
  if(feq(0,l.b)){
    swap(l.a, l.b);
    swap(c.c.x, c.c.y);
  }
  if(feq(0,l.b))return false;
  return hayEcCuad(
    sqr(l.a)+sqr(l.b),
    2.0*l.a*l.b*c.c.y-2.0*(sqr(l.b)*c.c.x+l.c*l.a),
    sqr(l.b)*(sqr(c.c.x)+sqr(c.c.y)-sqr(c.r))+sqr(l.c)-2.0*l.c*l.b*c.c.y
  );
}
pair<pto, pto> interCL(circ c, lin l){
  bool sw=false;
  if(sw=feq(0,l.b)){
    swap(l.a, l.b);
    swap(c.c.x, c.c.y);
  }
  pair<tipo, tipo> rc = ecCuad(
    sqr(l.a)+sqr(l.b),
    2.0*l.a*l.b*c.c.y-2.0*(sqr(l.b)*c.c.x+l.c*l.a),
    sqr(l.b)*(sqr(c.c.x)+sqr(c.c.y)-sqr(c.r))+sqr(l.c)-2.0*l.c*l.b*c.c.y
  );
  pair<pto, pto> p(
    punto(rc.first, (l.c - l.a * rc.first) / l.b),
    punto(rc.second, (l.c - l.a * rc.second) / l.b)
  );
  if(sw){
    swap(p.first.x, p.first.y);
```

```
    swap(p.second.x, p.second.y);
  }
  return p;
}
bool hayICC(circ c1, circ c2){
  lin l;
  l.a = c1.c.x-c2.c.x;
  l.b = c1.c.y-c2.c.y;
  l.c = (sqr(c2.r)-sqr(c1.r)+sqr(c1.c.x)-sqr(c2.c.x)+sqr(c1.c.y)
    -sqr(c2.c.y))/2.0;
  return hayICL(c1, l);

}
```

2.15. Circle

```
struct Circle {
  Point c;
  double r;
  Circle() {}
  Circle(Point c, double r) : c(c), r(r) {}
};

bool isInside(Circle& c1, Circle& c2) {
  return isLess( len(c1.c - c2.c) + c1.r, c2.r );
}

bool isOutside(Circle& c1, Circle& c2) {
  return isLess( c1.r + c2.r, len(c1.c - c2.c) );
}

Point rotate(Point a, Point b) {
  double d = len(a);
  a.x /= d;
  a.y /= d;
  return Point(a.x*b.x - a.y*b.y, b.x*a.y + a.x*b.y);
}

vector<Point> intersection(Circle& c1, Circle& c2) {
  vector<Point> r;
  if(isInside(c1, c2) || isInside(c2, c1) || isOutside(c1, c2)) {
    return r;
  }
  double d = len(c1.c - c2.c);
  double alfa = acos((c1.r*c1.r + d*d - c2.r*c2.r)/(2*c1.r*d));
  double dy = c1.r*sin(alfa);
  double dx = c1.r*cos(alfa);
  Point v(fabs(c1.c.x-c2.c.x), fabs(c1.c.y-c2.c.y));
  r.push_back(c1.c + rotate(v, Point(dx, dy)));
  if(isLess(0.0, dy)) {
    r.push_back(c1.c + rotate(v, Point(dx, -dy)));
  }
  return r;
}
```

3. Grafos

3.1. DFS(Orden topologico)

```
// dfs for directed graphs
void dfs_visit(VVI& adj,int u,VI& color,VI& pred,VI& ts){
  color[u] = 1;
  REP(i,SIZE(adj[u])){
    int v = adj[u][i];
    if( color[v] == 0 ){  // (u,v): tree edge
      pred[v] = u;
      dfs_visit(adj,v,color,pred,ts);
    }else if( color[v] == 1 ) {
      // (u,v): back-edge; cycle found (might be self-loop)
    }else{
      assert( color[v] == 2 );
      // (u,v): forward or cross edge
    }
  }
  color[u] = 2;
  ts.push_back(u);
}

VI dfs(VVI& adj,VI& pred){
  int n = adj.size();
  pred = VI(n,-1);
  VI color(n,0);
  VI ts;
  REP(u,n) if( color[u] == 0 ) dfs_visit(adj,u,color,pred,ts);
  reverse(ALL(ts));
  return ts;
}
```

3.2. Dijkstra PQ (se puede convertir en Prim) O(e*log(e))

```
const int INF = 2000000000;

VI dijkstra(vector<vector<PII> >& adj,int ini){
  int N = adj.size();
  priority_queue<PII,vector<PII>,greater<PII> > Q;
  VI dist(N,INF);
  VI pred(N,-1);  //C
  Q.push(PII(0,ini));
  dist[ini] = 0;
  pred[ini] = ini;  //C
  vector<bool> done(N,false);
  while( Q.size() ){
    int u = Q.top().second; Q.pop();
    if( done[u] ) continue;
    done[u] = true;
    REP(i,SIZE(adj[u])){
      int v = adj[u][i].first;
      int d = dist[u] + adj[u][i].second; // prim: int d=adj[u][i].second;
      if( dist[v] > d ){
        dist[v] = d;
        pred[v] = u;  //C
        Q.push(PII(d,v));
      }
    }
  }
  return dist;
}
```

3.3. Dijkstra Set O(e*log(v))

```
const int INF = 2000000000;

VI dijkstra(vector<vector<PII> >& adj,int ini){
  int N = adj.size();
  set<PII> Q;
  VI dist(N,INF);
  VI pred(N,-1);  //C
  Q.insert(PII(0,ini));
  dist[ini] = 0;
  pred[ini] = ini;  //C
  while( Q.size() ){
    int u = Q.begin()->second;
    Q.erase(Q.begin());
    REP(i,SIZE(adj[u])){
      int v = adj[u][i].first;
      int d = dist[u] + adj[u][i].second;
      if( dist[v] > d ){
        Q.erase(PII(dist[v],v));
        dist[v] = d;
        pred[v] = u;  //C
        Q.insert(PII(dist[v],v));
      }
    }
  }
  return dist;
}
```

3.4. Strongly connected components O(v+e)

```
// Strongly connected components
VVI scc(VVI& adj){
  int n = adj.size();

  // first dfs
  VI color(n,0), pred(n,-1), order;
  REP(u,n) if( color[u] == 0 ) dfs_visit(adj,u,color,pred,order);
  assert( SIZE(order) == n );

  // compute G^{T}
  VVI adjT(n);
  REP(u,n) REP(i,SIZE(adj[u])){
    int v = adj[u][i];
    adjT[v].push_back(u);
  }

  // second dfs (in order of decreasing finishing time)
  fill(ALL(color),0);
  fill(ALL(pred),-1);
  VVI ret;
  for( int i = SIZE(order)-1; i>=0; i-- ) {
    int u = order[i];
    if ( color[u] == 0 ){
      // new strongly connected component
      ret.push_back(VI());
      dfs_visit(adjT,u,color,pred,ret.back());
    }
  }
  return ret;
}
```

3.5. Kruskal O(e*log(e)) & Union-Find

```
struct UnionFind{
  UnionFind(int N) : boss(N){ REP(i,N) boss[i]=i; }
  int root(int u){
    if(u!=boss[u]) boss[u]=root(boss[u]);
    return boss[u];
  }
  bool join(int u,int v){
    int a=root(u),b=root(v);
    if(rand()%2) boss[a]=b; else boss[b]=a;
    return a!=b;
  }
  VI boss;
};
struct Edge{
  int u,v,w;
  Edge(int u,int v,int w):u(u),v(v),w(w){}
  bool operator<(const Edge& e) const{ return w<e.w; }
};
PII kruskal(int N, vector<Edge>& E){
  sort(E.begin(),E.end());
  UnionFind B(N);
  int totalw = 0, ngroups = N;
  REP(i,SIZE(E)){
    if(B.join(E[i].u,E[i].v)){
      // adding edge to the MST
      totalw += E[i].w;
      ngroups--;
    }
  }
  return PII(totalw,ngroups);
}
```

3.6. Bellman-Ford O(v*e)

```
const int INF = 1000000000;
VI dist,pred;

bool bellman_ford(vector<vector<PII> >& adj,int ini){
  int N = adj.size();
  dist = VI(N,INF);
  pred = VI(N,-1);
  dist[ini] = 0;
  REP(k,N){
    REP(u,N) REP(i,SIZE(adj[u])){
      int v = adj[u][i].first;
      int d = dist[u] + adj[u][i].second;
      if( dist[u] != INF && dist[v] > d ){
        dist[v] = d;
        pred[v] = u;
      }
    }
  }
  REP(u,N) REP(i,SIZE(adj[u])){
    int v = adj[u][i].first;
    int d = dist[u] + adj[u][i].second;
    if( dist[v] > d ) return false;
  }
  return true;
}
```

3.7. MaxFlow O($v * e^2$)

```
const int INF = 1000000000;
const int MAXN = 100;
const int MAXK = 2*MAXN + 2;
int cap[MAXK][MAXK],flow[MAXK][MAXK];

bool augment(int s,int t,vector<int>& pred){
  int K=pred.size();
  queue<int> Q;
  Q.push(s);
  fill(pred.begin(),pred.end(),-1);
  pred[s]=s;
  while(Q.size()){
    int u=Q.front();
    Q.pop();
    if(u==t) return true;
    FOR(v,0,K){
      if(flow[u][v] < cap[u][v] && pred[v]==-1){
        pred[v]=u;
        Q.push(v);
      }
    }
  }
  return false;
}
int maxflow(int s,int t,int K){
  FOR(i,0,K)FOR(j,0,K)flow[i][j]=0;
  vector<int> pred(K);
  while(augment(s,t,pred)){
    int add=INF;
    for(int i=t;i!=s;i=pred[i]) add = min(add, cap[pred[i]][i] - flow[pred[i]][i
        ]);
    for(int i=t;i!=s;i=pred[i]){
      flow[pred[i]][i] += add;
      flow[i][pred[i]] -= add;
    }
  }
  int ret=0;
  FOR(i,0,K-2) ret += flow[s][i];
  return ret;
}
```

3.8. Flujo de costo mínimo O(v^3) - PPP

```
#define MAXN 100
const int INF = 1<<30;
struct Eje{
  int f, m, p;
  int d(){return m-f;}
};
Eje red[MAXN][MAXN];
int adyc[MAXN], ady[MAXN][MAXN];
int N,F,D;
void iniG(int n, int f, int d){
  N=n;F=f;D=d;
  fill(red[0], red[N], (Eje){0,0,0});
  fill(adyc, adyc+N, 0);
}
void aEje(int d, int h, int m, int p){
  red[h][d].p = -(red[d][h].p = p);
  red[d][h].m = m; //poner [h][d] en m tambien para hacer eje bidireccional
  ady[d][adyc[d]++]=h; ady[h][adyc[h]++]=d;
}
int md[MAXN],vd[MAXN];
int camAu(int &v){
  fill(vd, vd+N, -1);
  vd[F]=F; md[F]=0;
  forn(rep, N)forn(i, N)if(vd[i]!=-1)forn(jj, adyc[i]){
    int j = ady[i][jj], nd = md[i]+red[i][j].p;
    if(red[i][j].d()>0)if(vd[j]==-1 || md[j] > nd)md[j]=nd,vd[j]=i;
  }
  v=0;
  if(vd[D]==-1)return 0;
  int f = INF;
  for(int n=D;n!=F;n=vd[n]) f <?= red[vd[n]][n].d();
  for(int n=D;n!=F;n=vd[n]){
    red[n][vd[n]].f=-(red[vd[n]][n].f+=f);
    v += red[vd[n]][n].p * f;
  }
  return f;
}
int flujo(int &r){
  r=0; int v,f=0, c;
  while((c = camAu(v)))r += v,f += c;
  return f;
}
```

3.9. Erdös-Gallai - PPP

```
includes: algorithm, functional, numeric, forn
tint n;tint d[MAXL]; //grafo
tint sd[MAXL]; //auxiliar
bool graphical() {
```

```
  if (accumulate(d, d+n, 0) % 2 == 1) return false;
  sort(d, d+n, greater<tint>()); copy(d, d+n, sd);
  forn(i,n) sd[i+1]+=sd[i];
  forn(i,n) {
    if (d[i] < 0) return false;
    tint j = lower_bound(d+i+1, d+n, i+1, greater<tint>()) - d;
    if (sd[i] > i*(i+1) + sd[n-1] - sd[j-1] + (j-i-1)*(i+1))
      return false;
  } return true;
}
```

3.10. Puntos de articulación O(e+v)

```
// num[u]: number of components left when u is removed.
// dis[u]: discovery time of u
// low[u]: min{ dis[v], { dis[w] : (u,w) is a back edge from some descendant u
    of v } }
VVI adj;
VI num,dis,low,pred;
int id = 0;

void dfs(int u,bool root=true){
  low[u] = dis[u] = id++;
  REP(i,SIZE(adj[u])){
    int v = adj[u][i];
    if( dis[v] == -1 ){
      pred[v] = u;
      dfs(v,false);
      if( root || low[v] >= dis[u] ) num[u]++;
      low[u] <?= low[v];
    }else if( pred[u] != v ){
      low[u] <?= dis[v];
    }
  }
}
```

3.11. Construccion de un grafo grilla

```
const int INF = 2000000000;

#define LEFT 1
#define UP 2
#define DOWN 4
#define RIGHT 8

// left, up, down, right
int dr[]={0,-1,1,0};
int dc[]={-1,0,0,1};

int main(){
  int n;
  int tc = 1;
  for(cin>>n;n!=0;cin>>n){
    vector<PII> A(n),B(n);
    REP(i,n){
      cin >> A[i].first >> A[i].second >> B[i].first >> B[i].second;
      if(A[i].first > B[i].first || A[i].second > B[i].second) swap(A[i],B[i]);
    }
    int sx,sy,ex,ey;
    cin >> sx >> sy >> ex >> ey;
    VI X,Y;
    REP(i,n){
      X.push_back(A[i].first);
      X.push_back(B[i].first);
      Y.push_back(A[i].second);
      Y.push_back(B[i].second);
    }
    X.push_back(-INF);
    Y.push_back(-INF);
    sort(ALL(X));
    sort(ALL(Y));
    X.erase(unique(ALL(X)),X.end());
    Y.erase(unique(ALL(Y)),Y.end());
    int R = SIZE(Y);
    int C = SIZE(X);
    VVI wall(R,VI(C,0));
    FOR(i,0,n){
      int c1 = lower_bound(ALL(X),A[i].first) - X.begin();
      int r1 = lower_bound(ALL(Y),A[i].second) - Y.begin();
      int c2 = lower_bound(ALL(X),B[i].first) - X.begin();
      int r2 = lower_bound(ALL(Y),B[i].second) - Y.begin();
      if(r1==r2){
        FOR(c,c1,c2) wall[r1][c] |= UP;
        FOR(c,c1,c2) wall[r1-1][c] |= DOWN;
      }else{
        assert(c1==c2);
        FOR(r,r1,r2) wall[r][c1] |= LEFT;
        FOR(r,r1,r2) wall[r][c1-1] |= RIGHT;
      }
    }
    int sc = upper_bound(ALL(X),sx) - X.begin() - 1;
    int sr = upper_bound(ALL(Y),sy) - Y.begin() - 1;
    int ec = upper_bound(ALL(X),ex) - X.begin() - 1;
```

```
    int er = upper_bound(ALL(Y),ey) - Y.begin() - 1;

    cout << "City "<< tc++ << endl;
    cout << "Peter has to cross " << bfs(wall,sr,sc,er,ec) << " streets" << endl
        ;
  }
  return 0;
}
```

4. Matemática

4.1. Algoritmos de cuentas

4.1.1. Numbers - combinatorio, catalan y stirling

```
LL choose2(int n, int k) {
  LL ret=1;
  REP(i,k) ret = ret*(n-i)/(i+1);
  return ret;
}

// Binomial coefficient choose[n][k] is the number of ways of picking k
    unordered outcomes from n possibilities.
const int UB=50;
LL choose[UB][UB];
void fill_choose(int M){
  REP(n,M){
    choose[n][0] = 1;
    FOR(k,1,M) choose[n][k] = choose[n-1][k] + choose[n-1][k-1];
  }
}

// Catalan numbers
// Interpretations of the nth Catalan number include:
// 1. The number of ways to arrange n pairs of matching parentheses.
// 2. The number of ways a convex polygon of n+2 sides can be split into n
    triangles.
// 3. The number of rooted binary trees with exactly n+1 leaves.
// NOTE: C[36] overflows long long
const int UBC = 36;
LL C[UBC];
void fill_C(int U){
  assert(U <= UBC);
  C[0] = 1;
  FOR(i,1,U) FOR(j,0,i) C[i] += C[j]*C[i-1-j];
}

// Stirling numbers of the 1st kind
// s(n,k) counts the number of permutations of n objects with exactly k cycles.
const int UBS = 20;
LL S1[UBS][UBS];
void fill_S1(int M){
  assert(M<=UBS);
  S1[0][0] = 1;
  FOR(n,1,M){
```

```
    S1[n][0] = 0;
    FOR(k,1,n+1) S1[n][k] = S1[n-1][k-1] + (n-1)*S1[n-1][k];
  }
}

// Stirling numbers of the 2nd kind
// S(n,k) is the number of ways to partition a set of n objects into k groups.
LL S2[UBS][UBS];
void fill_S2(int M){
  assert(M<=UBS);
  S2[0][0] = 1;
  FOR(n,1,M){
    S2[n][0] = 0;
    FOR(k,1,n+1) S2[n][k] = S2[n-1][k-1] + k*S2[n-1][k];
  }
}
```

4.1.2. MCD - Luciano

```
// x*a + y*b = gcd(a,b)
LL gcd_ext(LL a,LL b,LL& x,LL& y){
  if(b==0){ x=1; y=0; return a; }
  LL x2,y2;
  LL r = gcd_ext(b,a%b,x2,y2);
  x = y2;
  y = x2 - y2*(a/b);
  return r;
}
```

4.1.3. MCD - PPP

```
tint mcd(tint a, tint b){ return (a==0)?b:mcd(b%a, a);}
struct dxy {tint d,x,y;};
dxy mcde(tint a, tint b) {
  dxy r, t;
  if (b == 0) {
    r.d = a; r.x = 1; r.y = 0;
  } else {
    t = mcde(b,a%b);
    r.d = t.d; r.x = t.y;
    r.y = t.x - a/b*t.y;
  }
  return r;
}
```

4.1.4. Número combinatorio - PPP

```
tint _comb[MAXMEM][MAXMEM];
tint comb(tint n, tint m) {
  if (m<0||m>n)return 0;if(m==0||m==n)return 1;
  if (n >= MAXMEM) return comb(n-1,m-1)+comb(n-1,m);
  tint& r = _comb[n][m];
  if (r == -1) r = comb(n-1,m-1)+comb(n-1,m);
  return r;
}
```

4.1.5. Teorema Chino del Resto - PPP

```
usa: mcde
#define modq(x) (((x)%q+q)%q)
tint tcr(tint* r, tint* m, int n) { // x \equiv r_i (m_i) i \in [0..n)
  tint p=0, q=1;
  forn(i, n) {
    p = modq(p-r[i]);
    dxy w = mcde(m[i], q);
    if (p%w.d) return -1; // sistema incompaible
    q = q / w.d * m[i];
    p = modq(r[i] + m[i] * p / w.d * w.x);
  }
  return p; // x \equiv p (q)
}
```

4.1.6. Potenciación en O(log(e)) - PPP

```
tint potLog(tint b, tint e, tint m) {
    if (!e) return 1LL;
    tint r=potLog(b, e>>1, m);
    r=(r*r)%m;
    return (e&1)?(r*b)%m:r;
}
```

4.2. Teoremas y propiedades - PPP

4.2.1. Ecuación de grafo planar

$regiones = ejes - nodos + componentesConexas + 1$

4.2.2. Ternas pitagóricas

Hay ternas pitagóricas de la forma: $(a, b, c) = (m^2 - n^2, 2 \cdot m \cdot n, m^2 + n^2) \forall m > n > 0$ y son primitivas *sii* $(2|m \cdot n) \wedge (mcd(m, n) = 1)$
(Todas las primitivas (con (a, b) no ordenado) son de esa forma.) Obs: $(\mathrm{m}+i\mathrm{n})^2 = \mathrm{a}+i\mathrm{b}$

4.2.3. Teorema de Pick

$A = I + \frac{B}{2} - 1$, donde I = interior y B = borde

4.2.4. Propiedadas varias

$\sum_{i=0}^{n} r^i = \frac{r^{n+1}-1}{r-1}$; $\sum_{i=1}^{n} i^2 = \frac{n\cdot(n+1)\cdot(2n+1)}{6}$; $\sum_{i=1}^{n} i^3 = \left(\frac{n\cdot(n+1)}{2}\right)^2$

$\sum_{i=1}^{n} i^4 = \frac{n\cdot(n+1)\cdot(2n+1)\cdot(3n^2+3n-1)}{12}$

$\sum_{i=1}^{n} \begin{pmatrix} n-1 \\ i-1 \end{pmatrix} = 2^n$; $\sum_{i=1}^{n} i \cdot \begin{pmatrix} n-1 \\ i-1 \end{pmatrix} = n \cdot 2^{n-1}$

4.3. Tablas y cotas

4.3.1. Primos

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997 1009 1013 1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 1097 1103 1109 1117 1123 1129 1151 1153 1163 1171 1181 1187 1193 1201 1213 1217 1223 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291 1297 1301 1303 1307 1319 1321 1327 1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471 1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597 1601 1607 1609 1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697 1699 1709 1721 1723 1733 1741 1747 1753 1759 1777 1783 1787 1789 1801 1811 1823 1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949 1951 1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063 2069 2081

Primos cercanos a 10^n

9941 9949 9967 9973 10007 10009 10037 10039 10061 10067 10069 10079
99961 99971 99989 99991 100003 100019 100043 100049 100057 100069
999959 999961 999979 999983 1000003 1000033 1000037 1000039
9999943 9999971 9999973 9999991 10000019 10000079 10000103 10000121
99999941 99999959 99999971 99999989 100000007 100000037 100000039 100000049
999999893 999999929 999999937 1000000007 1000000009 1000000021 1000000033

Cantidad de primos menores que 10^n

$\pi(10^1) = 4$; $\pi(10^2) = 25$; $\pi(10^3) = 168$; $\pi(10^4) = 1229$; $\pi(10^5) = 9592$
$\pi(10^6) = 78.498$; $\pi(10^7) = 664.579$; $\pi(10^8) = 5.761.455$; $\pi(10^9) = 50.847.534$
$\pi(10^{10}) = 455.052{,}511$; $\pi(10^{11}) = 4.118.054.813$; $\pi(10^{12}) = 37.607.912.018$

4.3.2. Divisores

Cantidad de divisores (σ_0) para *algunos* $n/\neg\exists n' < n, \sigma_0(n') \geqslant \sigma_0(n)$
$\sigma_0(60) = 12$; $\sigma_0(120) = 16$; $\sigma_0(180) = 18$; $\sigma_0(240) = 20$; $\sigma_0(360) = 24$
$\sigma_0(720) = 30$; $\sigma_0(840) = 32$; $\sigma_0(1260) = 36$; $\sigma_0(1680) = 40$; $\sigma_0(10080) = 72$
$\sigma_0(15120) = 80$; $\sigma_0(50400) = 108$; $\sigma_0(83160) = 128$; $\sigma_0(110880) = 144$
$\sigma_0(498960) = 200$; $\sigma_0(554400) = 216$; $\sigma_0(1081080) = 256$; $\sigma_0(1441440) = 288$
$\sigma_0(4324320) = 384$; $\sigma_0(8648640) = 448$
Suma de divisores (σ_1) para *algunos* $n/\neg\exists n' < n, \sigma_1(n') \geqslant \sigma_1(n)$
$\sigma_1(96) = 252$; $\sigma_1(108) = 280$; $\sigma_1(120) = 360$; $\sigma_1(144) = 403$; $\sigma_1(168) = 480$
$\sigma_1(960) = 3048$; $\sigma_1(1008) = 3224$; $\sigma_1(1080) = 3600$; $\sigma_1(1200) = 3844$
$\sigma_1(4620) = 16128$; $\sigma_1(4680) = 16380$; $\sigma_1(5040) = 19344$; $\sigma_1(5760) = 19890$
$\sigma_1(8820) = 31122$; $\sigma_1(9240) = 34560$; $\sigma_1(10080) = 39312$; $\sigma_1(10920) = 40320$
$\sigma_1(32760) = 131040$; $\sigma_1(35280) = 137826$; $\sigma_1(36960) = 145152$; $\sigma_1(37800) = 148800$
$\sigma_1(60480) = 243840$; $\sigma_1(64680) = 246240$; $\sigma_1(65520) = 270816$; $\sigma_1(70560) = 280098$
$\sigma_1(95760) = 386880$; $\sigma_1(98280) = 403200$; $\sigma_1(100800) = 409448$
$\sigma_1(491400) = 2083200$; $\sigma_1(498960) = 2160576$; $\sigma_1(514080) = 2177280$
$\sigma_1(982800) = 4305280$; $\sigma_1(997920) = 4390848$; $\sigma_1(1048320) = 4464096$
$\sigma_1(4979520) = 22189440$; $\sigma_1(4989600) = 22686048$; $\sigma_1(5045040) = 23154768$
$\sigma_1(9896040) = 44323200$; $\sigma_1(9959040) = 44553600$; $\sigma_1(9979200) = 45732192$

4.3.3. Factoriales

$0! = 1$	$11! = 39.916.800$
$1! = 1$	$12! = 479.001.600$ ($\in$ `int`)
$2! = 2$	$13! = 6.227.020.800$
$3! = 6$	$14! = 87.178.291.200$
$4! = 24$	$15! = 1.307.674.368.000$
$5! = 120$	$16! = 20.922.789.888.000$
$6! = 720$	$17! = 355.687.428.096.000$
$7! = 5.040$	$18! = 6.402.373.705.728.000$
$8! = 40.320$	$19! = 121.645.100.408.832.000$
$9! = 362.880$	$20! = 2.432.902.008.176.640.000$ ($\in$ `tint`)
$10! = 3.628.800$	$21! = 51.090.942.171.709.400.000$

max signed tint = 9.223.372.036.854.775.807
max unsigned tint = 18.446.744.073.709.551.615

4.4. Solución de Sistemas Lineales - PPP

```cpp
typedef vector<tipo> Vec;
typedef vector<Vec> Mat;
#define eps 1e-10
#define feq(a, b) (fabs(a-b)<eps)
bool resolver_ev(Mat a, Vec y, Vec &x, Mat &ev){
  int n = a.size(), m = n?a[0].size():0, rw = min(n, m);
  vector<int> p; forn(i,m) p.push_back(i);
  forn(i, rw){
    int uc=i, uf=i;
    forsn(f, i, n) forsn(c, i, m) if(fabs(a[f][c])>fabs(a[uf][uc])) {uf=f;uc=c;}
    if (feq(a[uf][uc], 0)) { rw = i; break; }
    forn(j, n) swap(a[j][i], a[j][uc]);
    swap(a[i], a[uf]); swap(y[i], y[uf]); swap(p[i], p[uc]);
    tipo inv = 1 / a[i][i]; //aca divide
    forsn(j, i+1, n) {
      tipo v = a[j][i] * inv;
      forsn(k, i, m) a[j][k]-=v * a[i][k];
      y[j] -= v*y[i];
    }
  } // rw = rango(a), aca la matriz esta triangulada
  forsn(i, rw, n) if (!feq(y[i],0)) return false; // checkeo de compatibilidad
  x = vector<tipo>(m, 0);
  dforn(i, rw){
    tipo s = y[i];
    forsn(j, i+1, rw) s -= a[i][j]*x[p[j]];
    x[p[i]] = s / a[i][i]; //aca divide
  }
  ev = Mat(m-rw, Vec(m, 0)); // Esta parte va SOLO si se necesita el ev
  forn(k, m-rw) {
    ev[k][p[k+rw]] = 1;
    dforn(i, rw){
      tipo s = -a[i][k+rw];
      forsn(j, i+1, rw) s -= a[i][j]*ev[k][p[j]];
      ev[k][p[i]] = s / a[i][i]; //aca divide
    }
  }
  return true;
}

bool diagonalizar(Mat &a){
  // PRE: a.cols > a.filas
  // PRE: las primeras (a.filas) columnas de a son l.i.
  int n = a.size(), m = a[0].size();
  forn(i, n){
    int uf = i;
    forsn(k, i, n) if (fabs(a[k][i]) > fabs(a[uf][i])) uf = k;
    if (feq(a[uf][i], 0)) return false;
    swap(a[i], a[uf]);
    tipo inv = 1 / a[i][i]; // aca divide
    forn(j, n) if (j != i) {
      tipo v = a[j][i] * inv;
      forsn(k, i, m) a[j][k] -= v * a[i][k];
    }
    forsn(k, i, m) a[i][k] *= inv;
  }
  return true;
}
```

4.5. Fracción

```cpp
int gcd(int a, int b) { return b != 0 ? gcd(b, a%b) : a; }
int lcm(int a, int b) { return a!=0 || b!=0 ? a / gcd(a,b) * b : 0; }

class Fraction {
  void norm() { int g = gcd(n,d); n/=g; d/=g; if( d<0 ) { n=-n; d=-d; } }
  int n, d;

public:
  Fraction() : n(0), d(1) {}
  Fraction(int n_, int d_) : n(n_), d(d_) {
    assert(d != 0);
    norm();
  }

  Fraction operator+(const Fraction& f) const {
    int m = lcm(d, f.d);
    return Fraction(m/d*n + m/f.d*f.n, m);
  }
  Fraction operator-() const { return Fraction(-n,d); }
  Fraction operator-(const Fraction& f) const { return *this + (-f); }
  Fraction operator*(const Fraction& f) const { return Fraction(n*f.n, d*f.d); }
  Fraction operator/(const Fraction& f) const { return Fraction(n*f.d, d*f.n); }

  bool Fraction::operator<(const Fraction& f) const { return n*f.d < f.n*d; }

  friend std::ostream& operator<<(std::ostream&, const Fraction&);
};

ostream& operator<<(ostream& os, const Fraction& f) {
  os << f.n;
  if(f.d != 1 && f.n != 0) os << '/' << f.d;
  return os;
}
```

5. MISC

5.1. SAT- 2 - PPP

```
usa: vector, set, map, list
typedef pair<bool,tint> term;
typedef term nodo;
typedef pair<term,term> eje;
typedef pair<term,term> ecu;
typedef map< nodo, set<nodo> > grafo;
#define esta(e,c) ((c).find(e)!=(c).end())
#define forall(i,c,t) for(t::iterator i=(c).begin();i!=(c).end();++i)
term nega(term t) { return term(!t.first, t.second); }
void addEje(eje e, grafo& g, grafo& gt) {
  g[e.first].insert(e.second);
  gt[e.second].insert(e.first);
}
void addEcu(ecu e, grafo& g, grafo& gt) {
  addEje(eje(nega(e.first), e.second), g, gt);
  addEje(eje(nega(e.second), e.first), g, gt);
}
list<nodo> fs;
set<nodo> noVs;
grafo gr,grt;
void dfs(nodo ini, grafo& g, set<nodo>& mEn) {
  mEn.insert(ini);
  noVs.erase(ini);
  forall(i, g[ini], set<nodo>) {
    if (esta(*i, noVs)) dfs(*i, g, mEn);
  }
  fs.push_front(ini);
}
void iniNoVs() {
  noVs.clear();
  forall(i, gr, grafo) noVs.insert(i->first);
}
void calcFs() {
  set<nodo> dummy; fs.clear();
  iniNoVs();
  while(!noVs.empty()){
    dfs(*noVs.begin(), gr, dummy);
  }
}
list< set<nodo> > comps;
void compCon() {
  iniNoVs(); comps.clear();
  forall(f, fs, list<nodo>) {
    if (esta(*f, noVs)) {
      set<nodo> comp;
      dfs(*f, grt, comp);
      comps.push_back(comp);
    }
  }
}
bool satisf() {
  forall(c, comps, list< set<nodo> >) {
    forall(t, *c, set<nodo>) {
      if (esta(nega(*t), *c)) return false;
    }
  } return true;
}
bool resolver() {
  calcFs();
  compCon();
  return satisf();
}
```

5.2. LIS - Longest Increasing Subsequence O(n*log(n))

```
int lis(VI& v){
  VI ret;
  REP(i,SIZE(v)){
    VI::iterator it = lower_bound(ALL(ret), v[i]);  // upper_bound deja
        elementos iguales
    if( it == ret.end() ) ret.push_back(v[i]); else *it = v[i];
  }
  return ret.size();
}
```

5.3. Merge Sort

```
// cnt_inv = number of inversions in A
LL cnt_inv = 0;

void merge(VLL& A,int a,int b,int c){
  VLL B(A.begin()+a, A.begin()+b);
  VLL C(A.begin()+b, A.begin()+c);
  int i=0,j=0,k=a;
  while(i<B.size() && j<C.size()){
    if(B[i] <= C[j]){
      A[k++] = B[i++];
    }else{
      cnt_inv += B.size()-i;
```

```
      A[k++] = C[j++];
    }
  }
  while(i<B.size()) A[k++] = B[i++];
  while(j<C.size()) A[k++] = C[j++];
}
void mergesort(VLL& A,int l,int u){
  if(u-l > 1){
    int m = (l+u) / 2;
    mergesort(A,l,m);
    mergesort(A,m,u);
    merge(A,l,m,u);
  }
}
```

5.4. To Roman

```
// Roman numerals
// Converts an integer in the range [1, 4000) to a lower case Roman numeral
string fill(char ch,int n) { return string(n,ch); }
string toRoman( int n ) {
    if( n < 4 ) return fill( 'i', n );
    if( n < 6 ) return fill( 'i', 5 - n ) + "v";
    if( n < 9 ) return string( "v" ) + fill( 'i', n - 5 );
    if( n < 11 ) return fill( 'i', 10 - n ) + "x";
    if( n < 40 ) return fill( 'x', n / 10 ) + toRoman( n % 10 );
    if( n < 60 ) return fill( 'x', 5 - n / 10 ) + 'l' + toRoman( n % 10 );
    if( n < 90 ) return string( "l" ) + fill( 'x', n / 10 - 5 ) + toRoman( n %
        10 );
    if( n < 110 ) return fill( 'x', 10 - n / 10 ) + "c" + toRoman( n % 10 );
    if( n < 400 ) return fill( 'c', n / 100 ) + toRoman( n % 100 );
    if( n < 600 ) return fill( 'c', 5 - n / 100 ) + 'd' + toRoman( n % 100 );
    if( n < 900 ) return string( "d" ) + fill( 'c', n / 100 - 5 ) + toRoman( n %
        100 );
    if( n < 1100 ) return fill( 'c', 10 - n / 100 ) + "m" + toRoman( n % 100 );
    if( n < 4000 ) return fill( 'm', n / 1000 ) + toRoman( n % 1000 );
    return "?";
}
```

5.5. Tokenize

```
VS tokenize(string str,string del){
  str += del[0];
  VS ret;
  string w;
  REP(i,SIZE(str)){
    if(del.find(str[i])==string::npos){
      w += str[i];
    }else if(w!=""){
      ret.push_back(w),w="";
    }
  }
  return ret;
}
```

5.6. Fibonacci

```
// includes: ext/numeric
// usa: operator* de VVLL
LL fib(LL n) {
  VVLL mat(2,VLL(2,1));
  mat[1][1]=0;
  return __gnu_cxx::power(mat,n)[1][0];
}
```

5.7. Matrix

```
// Matrix multiplication
// usa: REP
template<class T>
vector<vector<T> > operator*(const vector<vector<T> >& A,const vector<vector<T>
    >& B){
  assert(A.size() > 0);
  assert(B.size() > 0);
  assert(A[0].size() == B.size());
  int n = A.size();
  int m = A[0].size();
  int l = B[0].size();
  vector<vector<T> > ret(n,vector<T>(l,T(0)));
  REP(i,n) REP(j,l) REP(k,m){
    ret[i][j] += A[i][k] * B[k][j];  // % MOD; ret[i][j] %= MOD;
  }
  return ret;
}
```

5.8. Sieve

```
// pf[n] = smallest prime factor of n
// usa: REP
const int U=1000;
int pf[U];
void init(){
  REP(i,U) pf[i] = i;
  for(int p=2; p*p<U; p++) if( pf[p] == p ){
```

```
    for(int m=p*p; m<U; m+=p) pf[m] <?= p;
  }
}
```

6. JAVA

6.1. FunctionRoot

```
// Method to carry out the secant search.
static double secant(int n, double del, double x, double dx) {
  int k = 0; double x1 = x+dx;
  while ((Math.abs(dx)>del) && (k<n)) {
    double d = f(x1)-f(x);
    double x2 = x1-f(x1)*(x1-x)/d;
    x = x1;
    x1 = x2;
    dx = x1-x;
    k++;
  }
  if (k==n)
    System.out.println("No convergence in " + n + " iterations");
  return x1;
}
static double f(double x) { ... }
```

6.2. Gauss

```
static double determinant(double a[][]) {
  int n = a.length; int index[] = new int[n];
  gaussian(a, index); // Transform the matrix into an upper triangle
  double d = 1; // Take the product of the diagonal elements
  for (int i=0; i<n; ++i) d = d*a[index[i]][i];
  int sgn = 1; // Find the sign of the determinant
  for (int i=0; i<n; ++i) {
    if (i != index[i]) {
      sgn = -sgn;
      int j = index[i];
      index[i] = index[j];
      index[j] = j;
    }
  }
  return sgn*d;
}

// Method to solve the equation a[][] x[] = b[] with
// the partial-pivoting Gaussian elimination.
static double[] solve(double a[][], double b[], int index[]) {
  int n = b.length; double x[] = new double[n];
  gaussian(a, index); // Transform the matrix into an upper triangle
  for(int i=0; i<n-1; ++i) { // Update the array b[i] with the ratios stored
```

```
    for(int j =i+1; j<n; ++j) b[index[j]] -= a[index[j]][i]*b[index[i]];
  }
  // Perform backward substitutions
  x[n-1] = b[index[n-1]]/a[index[n-1]][n-1];
  for (int i=n-2; i>=0; --i) {
    x[i] = b[index[i]];
    for (int j=i+1; j<n; ++j) x[i] -= a[index[i]][j]*x[j];
    x[i] /= a[index[i]][i];
  }
  return x;
}

public static double[][] invert(double a[][]) {
  int n = a.length; int index[] = new int[n];
  double x[][] = new double[n][n]; double b[][] = new double[n][n];
  for (int i=0; i<n; ++i) b[i][i] = 1;
  gaussian(a, index); // Transform the matrix into an upper triangle
  // Update the matrix b[i][j] with the ratios stored
  for (int i=0; i<n-1; ++i) for (int j=i+1; j<n; ++j) for (int k=0; k<n; ++k)
    b[index[j]][k] -= a[index[j]][i]*b[index[i]][k];
  // Perform backward substitutions
  for (int i=0; i<n; ++i) {
    x[n-1][i] = b[index[n-1]][i]/a[index[n-1]][n-1];
    for (int j=n-2; j>=0; --j) {
      x[j][i] = b[index[j]][i];
      for (int k=j+1; k<n; ++k) x[j][i] -= a[index[j]][k]*x[k][i];
      x[j][i] /= a[index[j]][j];
    }
  }
  return x;
}

// Method to carry out the partial-pivoting Gaussian elimination.
// Here index[] stores pivoting order.
static void gaussian(double a[][], int index[]) {
  int n = index.length; double c[] = new double[n];
  for (int i=0; i<n; ++i) index[i] = i;
  for (int i=0; i<n; ++i) { // Find the rescaling factors, one from each row
    double c1 = 0;
    for (int j=0; j<n; ++j) {
      double c0 = Math.abs(a[i][j]);
      if (c0 > c1) c1 = c0;
    }
    c[i] = c1;
  }
  int k = 0; // Search the pivoting element from each column
  for (int j=0; j<n-1; ++j) {
    double pi1 = 0;
    for (int i=j; i<n; ++i) {
      double pi0 = Math.abs(a[index[i]][j]);
      pi0 /= c[index[i]];
      if (pi0 > pi1) {
        pi1 = pi0;
        k = i;
      }
    }
    // Interchange rows according to the pivoting order
    int itmp = index[j]; index[j] = index[k]; index[k] = itmp;
    for (int i=j+1; i<n; ++i) {
      double pj = a[index[i]][j]/a[index[j]][j];
      a[index[i]][j] = pj; // Record pivoting ratios below the diagonal
      for (int l=j+1; l<n; ++l) // Modify other elements accordingly
        a[index[i]][l] -= pj*a[index[j]][l];
    }
  }
}
```

6.3. Integrate

```
static double simpson2(double a, double b, double del, int step, int maxstep) {
  double h = b-a;
  double c = (b+a)/2;
  double fa = f(a);
  double fc = f(c);
  double fb = f(b);
  double s0 = h*(fa+4*fc+fb)/6;
  double s1 = h*(fa+4*f(a+h/4)+2*fc + 4*f(a+3*h/4)+fb)/12;
  step++;
  if (step >= maxstep) {
    System.out.println ("Not converged after " + step + " recursions");
    return s1;
  } else {
    if (Math.abs(s1-s0) < 15*del)
      return s1;
    else
      return simpson2(a, c, del/2, step, maxstep) +
             simpson2(c, b, del/2, step, maxstep);
  }
}
static double monte(int steps) {
  Random r = new Random(); double s0 = 0; double ds = 0;
  for (int i=0; i<steps; ++i) {
    double x = r.nextDouble();
```

```
    s0 += f(x); ds += f(x)*f(x);
  }
  s0 /= steps; ds /= steps;
  ds = Math.sqrt(Math.abs(ds-s0*s0)/steps); // error
  return s0;
}
static double f(double x) { ... }
```

6.4. LIS

```
static int[] elems;
static int lis() {
  TreeSet<Integer> lis = new TreeSet<Integer>();
  for (int i = 0; i < elems.length; i++) {
    lis.add(elems[i]);
    Integer higher = lis.higher(elems[i]);
    if (higher != null) { // estricto: && !higher.equals(elems[i])
      lis.remove(higher);
    }
  }
  return lis.size();
}
```

6.5. Max Flow

```
static int INF = Integer.MAX_VALUE/2;
static int n;
static Map<Integer, Integer>[] capacity; // capacity adjacency list
static Map<Integer, Integer>[] cost; // cost adjacency list - min cost
static Map<Integer, Integer>[] flow;  // flow matrix
static int[] pred;    // augment path predecesors

static int maxFlow(int source, int sink) {
  initFlow(); int totalFlow = 0;
  while (bfs(source,sink)) {  // search augment path
    int pathInc = INF;
    for (int u = sink; pred[u] >= 0; u = pred[u]) {
      pathInc = min(pathInc, room(pred[u], u));
    }
    for (int u = sink; pred[u] >= 0; u = pred[u]) {  // udpate flow
    int v = pred[u];
      setFlow(v, u, flow(v, u) + pathInc);
      setFlow(u, v, flow(u, v) - pathInc)
    }
    totalFlow += pathInc;
  }
  return totalFlow;
}

static int[] maxFlowMinCost(int source, int sink) {
  initFlow(); int totalCost = 0, totalFlow = 0;
  while (bellmanFord(source,sink)) {  // search augment path
    int pathInc = INF;
    for (int u = sink; pred[u] >= 0; u = pred[u]) {
      pathInc = min(pathInc, room(pred[u], u));
    }
    //System.err.print("inc:"+pathInc + " path:"+sink);
    for (int u = sink; pred[u] >= 0; u = pred[u]) {  // update flow and cost
    int v = pred[u];
      //System.err.print(" " + v);
      totalCost += pathInc * cost(pred[u], u);
      setFlow(v, u, flow(v, u) + pathInc);
      setFlow(u, v, flow(u, v) - pathInc)
    }
    //System.err.println(" cost:"+totalCost);
    totalFlow += pathInc;
  }
  return new int[] {totalFlow, totalCost};
}

static void init() {
  capacity = new Map[n];
  cost = new Map[n]; // min cost
  for(int i = 0; i < n; i++) {
    capacity[i] = new HashMap();
    cost[i] = new HashMap(); // min cost
  }
}
static int flow(int u, int v) { return flow[u][v]; }
static int room(int u, int v) {
  return (flow(u, v) < 0) ? -flow(u, v) : cap(u, v) - flow(u,v);
}
static int cap(int u, int v) {
  Integer c = capacity[u].get(v);
  return c == null ? 0 : c;
}
static void setCap(int u, int v, int c) { capacity[u].put(v, c); }
static void setFlow(int u, int v, int c) { flow[u].put(v, c); }
static Set<Integer> adys(int u) { return capacity[u].keySet(); }

/****** regular ******/
static boolean bfs(int source, int sink) {
  boolean[] visited = new boolean[n];
```

```
  LinkedList<Integer> q = new LinkedList<Integer>();
  q.add(source);
  visited[source] = true;
  pred[source] = -1;
  while (!q.isEmpty()) {
    int u = q.poll();
    if (u == sink) return true;
    for (int v = 0; v < n; v++) {
      if (!visited[v] && room(u, v) > 0) {
        visited[v] = true;
        q.add(v);
        pred[v] = u;
      }
    }
  }
  return false;
}

static boolean dfs(int from, int to) {
  if (from == to) return true;
  boolean[] visited = new boolean[n];
  for (int u = 0; u < n; u++) {
    if (!visited[u] && room(from, u) > 0 && dfs(u, to)) {
      pred[u] = from;
      return true;
    }
  }
  return false;
}

/****** min cost ******/
static int cost(int u, int v) {
  boolean back = flow(u,v) < 0;
  Integer c = back ? cost[v].get(u) : cost[u].get(v);
  return c == null ? 0 : (back ? -1 : 1) * c;
}
static void setCost(int u, int v, int c) { cost[u].put(v, c); }
static void set(int u, int v, int k, int t) { setCap(u,v,k); setCost(u,v,t); }

// bellman ford
static boolean bellmanFord(int source, int sink) {
  int[] d = new int[n];
  Arrays.fill(d, INF);
  d[source] = 0;
  pred[source] = -1;
  boolean changed = true;
  for (int u = 0; u < n && changed; u++) {
    changed = false;
    for (int o = 0; o < n; o++) {
      for (int v = 0; v < n; v++) {
        int alt = d[o] + cost(o, v);
        if (d[v] > alt && room(o, v) > 0) {
          d[v] = alt;
          pred[v] = o;
          changed = true;
        }
      }
    }
  }
  return d[sink] < INF;
}
```

6.6. Next Permutation

```
static void swap(TYPE[] a, int i, int j) {
  TYPE aux = a[i];  a[i] = a[j]; a[j] = aux;
}
static void reverse(TYPE[] a) { reverse(a, 0); }
static void reverse(TYPE[] a, int from) {
  int last = a.length - 1;
  for(int i = (a.length - from) / 2 - 1; i >= 0; i--) {
    TYPE aux = a[i + from]; a[i + from] = a[last - i]; a[last - i] = aux;
  }
}
static boolean nextPermutation(TYPE[] a) {
  return nextPermutation(a, 0, a.length);
}
static boolean nextPermutation(TYPE[] a, int begin, int end) {
  if(begin + 1 >= end) return false;
  int i = end - 1;
  while(true) {
    int j = i--;
    if(a[i] < a[j]) {
      int n = end; while(a[i] >= a[--n]);
      swap(a, i, n); reverse(a, j); return true;
    }
    if(i == begin) { reverse(a); return false; }
  }
}
```