# Índice

# PPP-UBA - Reference

# 1. Algoritmos

**#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 | $T = e^n$ |

# 2. Estructuras

## 2.1. Range Minimum Query $\langle n \log n, 1\rangle$ (get)

**Resrtricción:** $n < 2^{\text{LVL}+1}$**; mn(i, j) incluye i y no incluye j; mn_init O(n log n)**

```
usa: tipo
#define LVL 10
tipo vec[LVL][1<<(LVL+1)];
tipo mn(int i, int j) { // intervalo [i,j)
  int p = 31-__builtin_clz(j-i);
  return min(vec[p][i],vec[p][j-(1<<p)]);
}
void mn_init(int n) {
  int mp = 31-__builtin_clz(n);
  forn(p, mp) forn(x, n-(1<<p)) vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)]);
}
```

## 2.2. Range Minimum Query $\langle n, \log n\rangle$ (get y set)

**Uso:** `MAXN` **es la cantidad máxima de elementos que se banca la estructura. pget(i, j) incluye i y no incluye j. init(n) O(n). Funciona con cualquier operador "+" asociativo y con elemento neutro "0" Se inicializa así:** `cin >> n; tipo* v = rmq.init(n); forn(i, n) cin >> v[i]; rmq.updall();`

```
#define MAXN 100000
struct rmq {
  int MAX;
  tipo vec[3*MAXN];
  tipo* init(int n) {
    MAX = 1 << (32-__builtin_clz(n));
    fill(vec, vec+2*MAX, 0); // 0 = elemento neutro
    return vec+MAX;
  }
  void updall() { dforn(i, MAX) vec[i] = vec[2*i] + vec[2*i+1]; } // + = operacion
  void pset(int i, tipo vl) {
    vec[i+=MAX] = vl;
    while(i) { i /= 2; vec[i] = vec[2*i] + vec[2*i+1]; } // + = operacion
  }
  tipo pget(int i, int j) { return _pget(i+MAX, j+MAX); }
  tipo _pget(int i, int j) {
    tipo res = 0;          // 0 = elemento neutro
    if (j-i <= 0) return res;
    if (i%2) res += vec[i++]; // + = operacin
    res += _pget(i/2, j/2);   // + = operacin
    if (j%2) res += vec[--j]; // + = operacin
```

```
    return res;
  }
};
```

## 2.3. Cantidad de menores o iguales en O(log n)

```
//insersion y consulta de cuantos <= en log n
struct leqset {
   int maxl; vector<int> c;
   int pref(int n, int l) { return (n>>(maxl-l))|(1<<l); }
   void ini(int ml) { maxl=ml; c=vector<int>(1<<(maxl+1)); }
   //inserta c copias de e, si c es negativo saca c copias
   void insert(int e, int q=1) { forn(l,maxl+1) c[pref(e,l)]+=q; }
   int leq(int e) {
      int r=0,a=1;
      forn(i,maxl) {
         a<<=1; int b=(e>>maxl-i-1)&1;
         if (b) r+=c[a]; a|=b;
      } return r + c[a]; //sin el c[a] da los estrictamente menores
   }
   int size() { return c[1]; }
   int count(int e) { return c[e|(1<<maxl)]; }
};
```

## 2.4. Suffix Array - Longuest Common Prefix

```
typedef unsigned char xchar;
#define MAXN 1000000

int p[MAXN], r[MAXN], t, n;

bool sacmp(int a, int b) { return p[(a+t)%n] < p[(b+t)%n]; }
void bwt(const xchar *s, int nn) {
  n = nn;
  int bc[256];
  memset(bc, 0, sizeof(bc));
  forn(i, n) ++bc[s[i]];
  forn(i, 255) bc[i+1]+=bc[i];
  forn(i, n) r[--bc[s[i]]]=i;
  forn(i, n) p[i]=bc[s[i]];

  int lnb,nb = 1;
  for(t = 1; t < n; t*=2) {
    lnb = nb; nb = 0;
    for(int i = 0, j = 1; i < n; i = j++) {
      /*calcular siguiente bucket*/
      while(j < n && p[r[j]] == p[r[i]]) ++j;
      if (j-i > 1) {
        sort(r+i, r+j, sacmp);
        int pk, opk = p[(r[i]+t)%n];
        int q = i, v = i;
        for(; i < j; i++) {
          if (((pk = p[(r[i]+t)%n]) != opk) && !(q <= opk && pk < j)) { opk = pk
               ; v = i; }
          p[r[i]] = v;
        }
      }
      nb++;
    }
    if (lnb == nb) break;
  }
//  prim = p[0];
}

void lcp(const xchar* s, int* h) {
  int q = 0, j;
  forn(i,n) if (p[i]) {
    j = r[p[i]-1];
    while(q < n && s[(i+q)%n] == s[(j+q)%n]) ++q;
    h[p[i]-1] = q;
    if (q > 0) --q;
  }
}
```

# 3. Geom

## 3.1. Point in Poly

```
usa: algorithm, vector
struct pto { tipo x,y; };
bool pnpoly(vector<pto>&v,pto p){
  unsigned i, j, mi, mj, c = 0;
  for(i=0, j = v.size()-1; i< v.size(); j = i++){
    if((v[i].y<=p.y && p.y<v[j].y) ||
       (v[j].y<=p.y && p.y<v[i].y)){
      mi=i,mj=j; if(v[mi].y>v[mj].y)swap(mi,mj);
      if((p.x-v[mi].x) * (v[mj].y-v[mi].y)
       < (p.y-v[mi].y) * (v[mj].x-v[mi].x)) c^=1;
    }
  } return c;
}
```

## 3.2. Convex Hull

```
usa: algorithm, vector, sqr
tipo pcruz(tipo x1,tipo y1,tipo x2,tipo y2){return x1*y2-x2*y1;}
struct pto {
  tipo x,y;
  tipo n2(pto &p2)const{
    return sqr(x-p2.x)+sqr(y-p2.y);
  }
} r;
tipo area3(pto a, pto b, pto c){
  return pcruz(b.x-a.x,b.y-a.y,c.x-a.x,c.y-a.y);
}
bool men2(const pto &p1, const pto &p2){
  return (p1.y==p2.y)?(p1.x<p2.x):(p1.y<p2.y);
}
bool operator<(const pto &p1,const pto &p2){
  tipo ar = area3(r,p1,p2);
  return(ar==0)?(p1.n2(r)<p2.n2(r)):ar>0;
  //< clockwise, >counterclockwise
}
typedef vector<pto> VP;
VP chull(VP & l){
  VP res = l;  if(l.size()<3) return res;
  r = *(min_element(res.begin(),res.end(),men2));
  sort(res.begin(),res.end());
  tint i=0;VP ch;ch.push_back(res[i++]);ch.push_back(res[i++]);
  while(i<res.size())  // area3 > clockwise, < counterclockwise
    if(ch.size()>1 && area3(ch[ch.size()-2],ch[ch.size()-1],res[i])<=0)
      ch.pop_back();
    else
      ch.push_back(res[i++]);
  return ch;
}
```

## 3.3. Circulo mínimo

```
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;
};
```

## 3.4. Máximo rectángulo entre puntos

```
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;
}
```

## 3.5. Máxima cantidad de puntos alineados

```
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;
}
```

## 3.6. Centro de masa y area de un polígono

```
usa: vector, forn
struct pto { tint x,y; };
typedef vector<pto> poly;
tint pcruz(tint x1, tint y1, tint x2, tint y2) { return x1*y2-x2*y1; }
tint area3(const pto& p, const pto& p2, const pto& p3) {
  return pcruz(p2.x-p.x, p2.y-p.y, p3.x-p.x, p3.y-p.y);
}
tint areaPor2(const poly& p) {
  tint a = 0; tint l = p.size()-1;
  forn(i,l-1) a += area3(p[i], p[i+1], p[l]);
  return abs(a);
}
pto bariCentroPor3(const pto& p1, const pto& p2, const pto& p3) {
  pto r;
  r.x = p1.x+p2.x+p3.x; r.y = p1.y+p2.y+p3.y;
  return r;
}
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;
}
```

## 3.7. Par de puntos mas cercano

```
usa algorithm, vector, tdbl, tint, tipo, INF, forn, cmath
const tint MAX_N = 10010;
struct pto { tipo x,y;} r;
typedef vector<pto> VP;
#define ord(n,a,b) bool n(const pto &p, const pto &q){ \
  return ((p.a==q.a)?(p.b<q.b):(p.a<q.a));}
#define sqr(a) ((a)*(a))
ord(mx,x,y);
ord(my,y,x);
bool vale(const pto &p){return mx(p,r);};
tipo dist(pto a,pto b){return sqr(a.x-b.x)+sqr(a.y-b.y);}
pto vx[MAX_N];
pto vy[MAX_N];
tint N;
tipo cpair(tint ini, tint fin){
  if(fin-ini==1)return INF;
  if(fin-ini==2)return dist(vx[ini], vx[ini+1]);
  vector<pto> y(fin-ini);
  copy(vy+ini, vy+fin, y.begin());
  tint  m = (ini+fin)/2;
  r = vx[m];
  stable_partition(vy+ini, vy+fin, vale);
  tipo d = min(cpair(ini, m), cpair(m, fin));
  vector<pto> w;
  forn(i, y.size())if(sqr(fabs(y[i].x-vx[m].x))<=d)w.push_back(y[i]);
  forn(i,w.size()){
    for(tint j=i+1;(j<(tint)w.size())
      && sqr(fabs(w[i].y-w[j].y))<d;j++){
      d<?=dist(w[i],w[j]);
    }
  }
  return d;
}
tipo closest_pair(){
  sort(vx, vx+N,mx);
  sort(vy, vy+N,my);
  for(tint i=1;i<N;i++){
    if(vx[i].x==vx[i-1].x && vx[i].y==vx[i-1].y)return 0;
  }
  return sqrt(cpair(0,N));
}
```

## 3.8. CCW

```
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;
}
```

## 3.9. Sweep Line

```
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;
}
```

## 3.10. Intersección de segmentos

```
struct pto{tint x,y;};
struct seg{pto f,s;};
tint sgn(tint a){return a;return (a>0)?1:((a<0)?(-1):0);}
tint pc(pto a, pto b, pto o){return (a.x-o.x)*(b.y-o.y)-(a.y-o.y)*(b.x-o.x);}
tint pe(pto a, pto b, pto o){return (a.x-o.x)*(b.x-o.x)+(a.y-o.y)*(b.y-o.y);}
bool inter(seg a, seg b){
  tint ka = sgn(pc(a.f, a.s, b.f))*sgn(pc(a.f, a.s, b.s));
  tint kb = sgn(pc(b.f, b.s, a.f))*sgn(pc(b.f, b.s, a.s));
  if(ka<0 && kb<0)return true; //cruza sin tocar
  if(ka==0 && (pe(a.f,a.s,b.f) <= 0 || pe(a.f,a.s,b.s) <= 0))return true; //b
      tiene un vertice en a
  if(kb==0 && (pe(b.f,b.s,a.f) <= 0 || pe(b.f,b.s,a.s) <= 0))return true; //a
      tiene un vertice en b
  return false;
}
```

## 3.11. 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)));
}
```

## 3.12. Cuentitas

```
usa: cmath, algorithm, tipo
struct pto{tipo x,y;};
struct lin{tipo a,b,c;};
struct circ{pto c; tipo r;};
#define sqr(a)((a)*(a))
const double PI = (2.0 * acos(0.0));
pto punto(tipo x, tipo y){pto r;r.x=x;r.y=y;return r;}
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);}
pto ptoMedio(pto a, pto b){return punto((a.x+b.x)/2.0,(a.y+b.y)/2.0);}
tipo pc(pto a, pto b, pto o){
  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 sqrd(a,b) (sqr(a.x-b.x)+sqr(a.y-b.y))
tipo dist(pto a, pto b){return sqrt(sqrd(a,b));}
//#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);
}
pto interLS(lin l, pto b1, pto b2){
  return interLL(l, linea(b1, b2));
}
pto interSS(pto a1, pto a2, pto b1, pto b2){
  return interLS(linea(a1, a2), b1, b2);
}
bool hayISS(pto a1, pto a2, pto b1, pto b2){
  if (estaPS(a1,b1,b2)||estaPS(a2,b1,b2)) return true;
  if (estaPS(b1,a1,a2)||estaPS(b2,a1,a2)) return true;
  lin a = linea(a1,a2), b = linea(b1, b2);
  if(!hayILL(a,b))return false;
  if(paral(a,b))return false;
  pto i = interLL(a,b);
  //sale(i);sale(a1);sale(a2);sale(b1);sale(b2);cout << endl;
  return estaPS(i,a1, a2) && estaPS(i,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 bariCentro(pto a, pto b, pto c){
  return punto(
    (a.x + b.x + c.x) / 3.0,
    (a.y + b.y + c.y) / 3.0);
}
```

```
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);

}
pair<pto, pto> interCC(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 interCL(c1, l);
}
```

# 4. Grafos

## 4.1. Kruskal & Union-Find

```
usa: vector, utility, forn
typedef pair< tint, pair<tint,tint> > eje;
tint n; vector<eje> ejes; //grafo n=cant nodos
vector<tint> _cl; //empieza con todos en -1
tint cl(tint i) { return (_cl[i] == -1 ? i : _cl[i] = cl(_cl[i])); }
void join(tint i, tint j) { if(cl(i)!=cl(j)) _cl[cl(i)] = cl(j); }
tint krus() {
  sort(ejes.begin(), ejes.end());
  tint u = 0, t = 0;
  cl.clear(); _cl.insert(_cl.begin(), n, -1);
  forn(i,ejes.size()) {
    eje& e = ejes[i];
    if (cl(e.second.first) != cl(e.second.second)) {
      u++; t += e.first; if(u==n-1) return t;
      join(e.second.first, e.second.second);
    }
  } return -1; //-1 es que no es conexo
}
```

## 4.2. Bellman-Ford

```
bool bellmanFord(int n){
  int i,o,d;
  static int dis[2*MAX+2];
  fill(dis,dis+n,INF);
  dis[ORIGEN]=0;
  camino[ORIGEN]=0;
  bool cambios=true;
  for(i=0;i<n && cambios;i++){
    cambios=false;
    forn(o,n)forn(d,n){
      if (dis[d]>dis[o]+ejes[o][d].costo){
        dis[d]=dis[o]+ejes[o][d].costo;
        camino[d]=o;
        cambios=true;
      }
    }
    return dis[DESTINO]<INF;
};
```

## 4.3. Floyd-Warhsall

```
tint n;tint mc[MAXN][MAXN]; //grafo (mat de long de ady)
void floyd(){
  forn(k,n)forn(i,n)forn(j,n) mc[i][j] <?= mc[i][k]+mc[k][j];
}
```

## 4.4. Edmond-Karp

```
usa: map,algorithm,queue
struct Eje{ long f,m; long d(){return m-f;}};
typedef map <int, Eje> MIE; MIE red[MAX_N];
int N,F,D;
void iniG(int n, int f, int d){N=n; F=f; D=d;fill(red, red+N, MIE());}
void aEje(int d, int h, int m){
  red[d][h].m=m;red[d][h].f=red[h][d].m=red[h][d].f=0;
}
#define DIF_F(i,j) (red[i][j].d())
#define DIF_FI(i)  (i->second.d())
int v[MAX_N];
long camAu(){
  fill(v, v+N,-1);
  queue<int> c;
  c.push(F);
  while(!(c.empty()) && v[D]==-1){
    int n = c.front(); c.pop();
    for(MIE::iterator i = red[n].begin(); i!=red[n].end(); i++){
      if(v[i->first]==-1 && DIF_FI(i) > 0){
        v[i->first]=n;
        c.push(i->first);
      }
    }
  }
  if(v[D]==-1)return 0;
  int n = D;
  long f = DIF_F(v[n], n);
  while(n!=F){
    f<?=DIF_F(v[n], n);
    n=v[n];
  }
  n = D;
  while(n!=F){
    red[n][v[n]].f=-(red[v[n]][n].f+=f);
    n=v[n];
  }
  return f;
}
long flujo(){long tot=0, c;do{tot+=(c=camAu());}while(c>0); return tot;}
```

## 4.5. Preflow-push

```
usa: algorithm, list, forn
#define MAX_N 200
typedef list<tint> lint;
typedef lint::iterator lintIt;
//usadas para el flujo
tint f[MAX_N][MAX_N]; //flujo
tint e[MAX_N]; //exceso
tint h[MAX_N]; //altura
lintIt cur[MAX_N];

//esto representa el grafo que hay que armar
lint ady[MAX_N]; //lista de adyacencias (para los dos lados)
tint c[MAX_N][MAX_N]; //capacidad (para los dos lados)
tint n; //cant de nodos

tint cf(tint i, tint j) { return c[i][j] - f[i][j]; }

void push(tint i, tint j) {
  tint p = min(e[i], cf(i,j));
  f[j][i] = -(f[i][j] += p);
  e[i] -= p;
  e[j] += p;
}
void lift(tint i) {
  tint hMin = n*n;
  for(lintIt it = ady[i].begin() ; it != ady[i].end() ; ++it) {
    if (cf(i, *it) > 0) hMin = min(hMin, h[*it]);
  }
  h[i] = hMin + 1;
}
void iniF(tint desde)
{
  forn(i,n) {
    h[i] = e[i] = 0;
    forn(j,n) f[i][j] = 0;
    cur[i] = ady[i].begin();
  }
  h[desde] = n;
  for(lintIt it = ady[desde].begin() ; it != ady[desde].end() ; ++it)
  {
    f[*it][desde] = -(f[desde][*it] = e[*it] = c[desde][*it]);
  }
}
void disch(tint i) {
  while(e[i] > 0)  {
    lintIt& it = cur[i];
    if (it == ady[i].end()) {lift(i); it = ady[i].begin();}
    else if (cf(i,*it) > 0 && h[i] == h[*it] + 1) push(i,*it);
    else ++it;
  }
}
tint calcF(tint desde, tint hasta) {
  iniF(desde);
  lint l;
  forn(i,n) {if (i != desde && i != hasta) l.push_back(i);}
  for(lintIt it = l.begin() ; it != l.end() ; ++it) {
    tint antH = h[*it];
    disch(*it);
    if (h[*it] > antH) { //move to front
      l.push_front(*it);
      l.erase(it);
      it = l.begin();
    }
  } return e[hasta];
}
void addEje(tint a, tint b, tint ca) {
  //requiere reiniciar las capacidades
  if (c[a][b] == 0) {//soporta muchos ejes mismo par de nodos
    ady[a].push_back(b);
    ady[b].push_back(a);
  }
  c[b][a] = c[a][b] += ca;
}
void iniGrafo(tint nn) {  //requiere n ya leido
  n=nn;
  forn(i,n) {
    forn(j,n) c[i][j] = 0;
    //solo si se usa la version de addeje con soporte multieje
    ady[i].clear();
  }
}
```

## 4.6. Flujo de costo mínimo

```
#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, fuente, destino
  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 = costo, return = flujo
  r=0; int v,f=0, c;
  while((c = camAu(v)))r += v,f += c;
  return f;
}
```

## 4.7. Matching perfecto de costo máximo - Hungarian O(Nˆ 3)

```
#define MAXN 256
#define INFTO 0x7f7f7f7f
int n;
int mt[MAXN][MAXN]; // Matriz de costos (X * Y)
int xy[MAXN], yx[MAXN]; // Matching resultante (X->Y, Y->X)

int lx[MAXN], ly[MAXN], slk[MAXN], slkx[MAXN], prv[MAXN];
char S[MAXN], T[MAXN];
void updtree(int x) {
  forn(y, n) if (lx[x] + ly[y] - mt[x][y] < slk[y]) {
    slk[y] = lx[x] + ly[y] - mt[x][y];
    slkx[y] = x;
} }
int hungar() {
  forn(i, n) {
    ly[i] = 0;
    lx[i] = *max_element(mt[i], mt[i]+n);
  }
  memset(xy, -1, sizeof(xy));
  memset(yx, -1, sizeof(yx));
  forn(m, n) {
    memset(S, 0, sizeof(S));
    memset(T, 0, sizeof(T));
    memset(prv, -1, sizeof(prv));
    memset(slk, 0x7f, sizeof(slk));
    queue<int> q;
    #define bpone(e, p) { q.push(e); prv[e] = p; S[e] = 1; updtree(e); }
    forn(i, n) if (xy[i] == -1) { bpone(i, -2); break; }
    int x=0, y=-1;
    while (y==-1) {
      while (!q.empty() && y==-1) {
        x = q.front(); q.pop();
        forn(j, n) if (mt[x][j] == lx[x] + ly[j] && !T[j]) {
          if (yx[j] == -1) { y = j; break; }
          T[j] = 1;
          bpone(yx[j], x);
        }
      }
      if (y!=-1) break;
      int dlt = INFTO;
      forn(j, n) if (!T[j]) dlt = min(dlt, slk[j]);
      forn(k, n) {
        if (S[k]) lx[k] -= dlt;
        if (T[k]) ly[k] += dlt;
        if (!T[k]) slk[k] -= dlt;
      }
      forn(j, n) if (!T[j] && !slk[j]) {
        if (yx[j] == -1) {
          x = slkx[j]; y = j; break;
        } else {
          T[j] = 1;
          if (!S[yx[j]]) bpone(yx[j], slkx[j]);
        }
      }
    }
    if (y!=-1) {
```

```
      for(int p = x; p != -2; p = prv[p]) {
        yx[y] = p;
        int ty = xy[p]; xy[p] = y; y = ty;
      }
    } else break;
  }
  int res = 0;
  forn(i, n) res += mt[i][xy[i]];
  return res;
}
```

## 4.8. Componentes biconexas

```
#define MAXN 1000
list<int> g[MAXN], gt[MAXN];
vector<vector<int> > comBi; // OUTPUT del algoritmo
int usd[MAXW];
int tN;

void initGrafo(int n) {
  fill_n(g ,n,list<int>());
  fill_n(gt,n,list<int>());
  tN = n;
}
void aEje(int s, int d) {
  g[s].push_front(d);
  gt[d].push_front(s);
}
void dfs(int nodo, list<int> grafo[], vector<int> &pila) {
  if (usd[nodo]) return;
  usd[nodo] = 1;
  forall(i,grafo[nodo]) dfs(*i,grafo,pila);
  pila.push_back(nodo);
}
void calcularBico() {
  vector<int> orden;
  memset(usd,0,sizeof(usd));
  forn(i,tN) if (!usd[i]) dfs(i,g,orden);
  memset(usd,0,sizeof(usd));
  comBi.clear();
  while (!orden.empty()) {
    int actual = orden.back(); orden.pop_back();
    if (!usd[actual]) {
      comBi.push_back(vector<int>());
      dfs(actual,gt,comBi.back());
    }
  }
}
```

## 4.9. Camino/Circuito Euleriano

```
usa: algorithm, vector, list, forn
typedef string ejeVal;
#define MENORATODOS ""
typedef pair<ejeVal, tint> eje;
tint n; vector<eje> ady[MAXN]; tint g[MAXN];
  //grafo (inG = in grado o grado si es no dir)
tint aux[MAXN];
tint pinta(tint f) {
  if (aux[f]) return 0;
  tint r = 1; aux[f] = 1;
  forn(i,ady[f].size()) r+=pinta(ady[f][i].second);
  return r;
}
tint compCon() { fill(aux, aux+n, 0); tint r=0; forn(i,n) if (!aux[i]) { r++;
    pinta(r); } return r; }
bool isEuler(bool path, bool dir) {
  if (compCon() > 1) return false; tint c = (path ? 2 : 0);
  forn(i,n) if(!dir ? ady[i].size() % 2 : g[i] != 0) {
    if (dir && abs(g[i]) > 1) return false;
    c--; if(c<0) return false; }
  return true;
}
bool findCycle(tint f, tint t, list<tint>& r) {
  if (aux[f] >= ady[f].size()) return false;
  tint va = ady[f][aux[f]++].second;
  r.push_back(va);
  return (va != t ? findCycle(va, t, r) : true);
}
list<tint> findEuler(bool path) { //always directed, no repeated values
  if (!isEuler(path, true)) return list<tint>();
  bool agrego = false;
  if (path) {
    tint i = max_element(g, g + n)-g;
    tint j = min_element(g, g + n)-g;
    if (g[i] != 0) { ady[i].push_back( eje(MENORATODOS, j) ); agrego = true; }
  }
  tint x = -1;
  forn(i,n) {sort(ady[i].begin(), ady[i].end()); if (x<0 || ady[i][0] < ady[x
      ][0]) x=i;}
  fill(aux, aux+n, 0);
  list<tint> r; findCycle(x,x,r); if (!agrego) r.push_front(r.back());
  list<tint> aux; bool find=false;
  list<tint>::iterator it = r.end();
```

```
  do{ if (!find) --it;
    for(find=findCycle(*it, *it, aux);!aux.empty();aux.pop_front()) it = r.insert(++it, aux.front());
  } while (it != r.begin());
  return r;
}
```

## 4.10. Erdös-Gallai

```
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;
}
```

## 4.11. Puntos de articulación

```
usa: vector, forn
typedef vector< vector<tint> > adyList;
adyList g; //EL GRAFO
vector<bool> artR; tint artT;
vector<tint> artB,artD;
void dfs(tint ant, tint v) {
  artB[v] = artT; artD[v] = artT++;
  forn(i, g[v].size()) if (artD[g[v][i]] == -1) {
      if (!v && i) artR[0]=true;
      tint w = g[v][i]; dfs(v, w);
      if (artB[w] < artD[v]) artB[v] <?= artB[w];
      else if (artB[w] >= artD[v] && v) artR[v]=true;
  } else if (g[v][i] != ant) {
      artB[v] <?= artD[g[v][i]];
  }
}
vector<bool> artPoints() {
  //dice true en los que son ptos de articulacion
  artR.clear(); artR.insert(artR.begin(), g.size(), false);
  if (!g.empty()) {
      artD.clear(); artD.insert(artD.begin(), g.size(), -1);
      artB.resize(g.size()); artT = 0; dfs(-1, 0);
  }
  return artR;
}
```

## 4.12. Grafo cactus

**Def:** Un grafo es cactus *sii* todo eje está en a lo sumo un ciclo.

```
struct eje { int t,i; };
typedef vector<eje> cycle;
int n,m,us[MAXM],pa[MAXN],epa[MAXN],tr[MAXM];
vector<eje> ady[MAXN];
void iniG(int nn) { n=nn; m=0; fill(ady,ady+n,vector<eje>());
fill(pa,pa+n,-1); }
//f:from t:to d:0 si no es dirigido y 1 si es dirigido
void addE(int f, int t, int d) {
  ady[f].push_back((eje){t,m});
  if (!d) ady[t].push_back((eje){f,m}), tr[m]=0;
  us[m++]=0;
}
//devuelve false si algun eje esta en mas de un ciclo
bool cycles(vector<cycle>& vr,int f=0,int a=-2,int ai=-2) {
  int t; pa[f]=a; epa[f]=ai;
  forn(i,ady[f].size()) if (!tr[ady[f][i].i]++) if (pa[t=ady[f][i].t]!=-1) {
    cycle c(1,ady[f][i]); int ef=f;
    do {
      if (!ef) return 0;
      eje e=ady[pa[ef]][epa[ef]];
      if (us[e.i]++) return 0;
      c.push_back(e);
    } while ((ef=pa[ef])!=t);
    vr.push_back(c);
  } else if (!cycles(vr,t,f,i)) return 0;
  return 1;
};
```

# 5. Matemática

## 5.1. Algoritmos de cuentas

### 5.1.1. MCD

```
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;
}
```

### 5.1.2. Número combinatorio

```
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;
}
// Bolas en Cajas
tint bolEnCaj(tint b, tint c) {return comb(c+b-1,b); }
```

### 5.1.3. Teorema Chino del Resto

```
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)
}
```

### 5.1.4. Potenciación en O(log(e))

```
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;
}
```

### 5.1.5. Longitud de los números de 1 a N

```
tint sumDig(tint n, tint m){ // resultado modulo m
    tint b=10, d=1, r=0;
    while(b<=n){
        r = (r + (b-b/10LL)*(d++))%m;
        b*=10LL;
    }
    return (r + (n-b/10LL+1LL)*d)%m;
}
```

## 5.2. Teoremas y propiedades

### 5.2.1. Ecuación de grafo planar

$regiones = ejes - nodos + componentesConexas + 1$

### 5.2.2. 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: $(\mathrm{m}+i\mathrm{n})^2 = \mathrm{a}+i\mathrm{b}$

### 5.2.3. Teorema de Pick

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

### 5.2.4. Propiedadas varias

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

$\sum_{i=1}^{n} i^4 = \frac{n\cdot(n+1)\cdot(2n+1)\cdot(3n^2+3n-1)}{12}$ ; $\sum_{i=1}^{n} i^5 = \left(\frac{n\cdot(n+1)}{2}\right)^2 \cdot \frac{2n^2+2n-1}{3}$

$\sum_{i=1}^{n} \left(\begin{array}{c} n-1 \\ i-1 \end{array}\right) = 2^{n-1}$ ; $\sum_{i=1}^{n} i \cdot \left(\begin{array}{c} n-1 \\ i-1 \end{array}\right) = n \cdot 2^{n-1}$

## 5.3. Tablas y cotas

### 5.3.1. 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** $10^n$
$\pi(10^1) = 4$ ; $\pi(10^2) = 25$ ; $\pi(10^3) = 168$ ; $\pi(10^4) = 1229$ ; $\pi(10^5) = 9592$
$\pi(10^6) = 78.498$ ; $\pi(10^7) = 664.579$ ; $\pi(10^8) = 5.761.455$ ; $\pi(10^9) = 50.847.534$
$\pi(10^{10}) = 455.052{,}511$ ; $\pi(10^{11}) = 4.118.054.813$ ; $\pi(10^{12}) = 37.607.912.018$

### 5.3.2. Divisores

Cantidad de divisores ($\sigma_0$) 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 ($\sigma_1$) para *algunos* $n/\neg\exists n' < n, \sigma_1(n') \geqslant \sigma_1(n)$
$\sigma_1(96) = 252$ ; $\sigma_1(108) = 280$ ; $\sigma_1(120) = 360$ ; $\sigma_1(144) = 403$ ; $\sigma_1(168) = 480$
$\sigma_1(960) = 3048$ ; $\sigma_1(1008) = 3224$ ; $\sigma_1(1080) = 3600$ ; $\sigma_1(1200) = 3844$
$\sigma_1(4620) = 16128$ ; $\sigma_1(4680) = 16380$ ; $\sigma_1(5040) = 19344$ ; $\sigma_1(5760) = 19890$
$\sigma_1(8820) = 31122$ ; $\sigma_1(9240) = 34560$ ; $\sigma_1(10080) = 39312$ ; $\sigma_1(10920) = 40320$
$\sigma_1(32760) = 131040$ ; $\sigma_1(35280) = 137826$ ; $\sigma_1(36960) = 145152$ ; $\sigma_1(37800) = 148800$
$\sigma_1(60480) = 243840$ ; $\sigma_1(64680) = 246240$ ; $\sigma_1(65520) = 270816$ ; $\sigma_1(70560) = 280098$
$\sigma_1(95760) = 386880$ ; $\sigma_1(98280) = 403200$ ; $\sigma_1(100800) = 409448$
$\sigma_1(491400) = 2083200$ ; $\sigma_1(498960) = 2160576$ ; $\sigma_1(514080) = 2177280$
$\sigma_1(982800) = 4305280$ ; $\sigma_1(997920) = 4390848$ ; $\sigma_1(1048320) = 4464096$
$\sigma_1(4979520) = 22189440$ ; $\sigma_1(4989600) = 22686048$ ; $\sigma_1(5045040) = 23154768$
$\sigma_1(9896040) = 44323200$ ; $\sigma_1(9959040) = 44553600$ ; $\sigma_1(9979200) = 45732192$

### 5.3.3. Factoriales

| | |
|---|---|
| 0! = 1 | 11! = 39.916.800 |
| 1! = 1 | 12! = 479.001.600 (∈ `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 (∈ `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

## 5.4. Solución de Sistemas Lineales

```
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;
}
```

## 5.5. Programación Lineal - Simplex

**Teorema de dualidad (fuerte):** Dado un problema lineal $\Pi_1$: minimizar $c^t \cdot X$, sujeto a $A \cdot X \leqslant b, X \geqslant 0$ se define el problema lineal *dual standard* $\Pi_2$ como: minimizar $-b^t \cdot Y$, sujeto a $A^t \cdot Y \leqslant c$. Si $\Pi_1$ es satisfacible entonces $\Pi_2$ es satisfacible y $c^t \cdot X = b^t \cdot Y$. Si $\Pi_1$ es insatisfacible o no acotado entonces $\Pi_2$ es insatisfacible o no acotado (Obs: no pueden ser ambos no acotados).

Dados `cfun`, `rmat` y `bvec`; Minimiza $\texttt{cfun}^t \cdot \texttt{xvar}$ sujeto a las condiciones $\texttt{rmat} \cdot \texttt{xvar} \leq \texttt{bvec}$. Los valores de `bvec` pueden ser negativos para representar desigualdades de $\geq$ (por ejemplo: $-x \leq -5$).

Es sensible a errores numéricos; se recomiendan valores de `eps=1e-16` y `epsval=1e-14`. El orden de magnitud de `epsval` debe ser del orden de la relación entre los valores más grandes de `rmat`.

```
usa: resolver,

#define MAXVAR 64
#define MAXRES 128
tipo rmat[MAXRES][MAXVAR+MAXRES*2];
tipo bvec[MAXRES];
tipo cfun[MAXVAR+MAXRES*2];
tipo xvar[MAXVAR];

#define HAYSOL  0
#define NOSOL -1
#define NOCOTA -2

int simplex(int m, int n) { // cant restric; cant vars
  int base[MAXVAR+MAXRES], esab[MAXVAR+MAXRES];
  int nn = n+m; // Variables (originales) + holgura
  tipo res = 0;

  forn(i, m) forn(j, m) rmat[i][n+j] = (i==j);
  forn(i, m) cfun[n+i] = 0;

  forn(i, n) esab[i] = -1;
  forn(i, m) { base[i] = n+i; esab[n+i] = i; }

  // Agrega las artificiales; si todos los bvec[] son positivos se puede omitir esto
  int arts[MAXRES];
  int bmin = 0;
  forn(i, m) if (bvec[i] < bvec[bmin]) bmin = i;
  int art = bvec[bmin] < -eps;
  forn(i, m) arts[i] = 2*(bvec[i] >= -eps) - 1;
  if (art) {
    forn(i, m) rmat[i][nn] = -(bvec[i] < -eps);
```

```
    esab[n+bmin] = -1; esab[nn] = bmin; base[bmin] = nn;
    nn++;
  }

  Mat B(m, Vec(m, 0));
  Vec y(m, 0), c(m, 0), d(m, 0);
  int j0 = 0;
  do {
    forn(i, m) forn(j, m) B[i][j] = arts[j] * rmat[j][base[i]];
    forn(i, m) c[i] = art?base[i]>=m+n:cfun[base[i]];
    resolver(B, c, y);

    for(; j0 < nn; ++j0) if (esab[j0] == -1) {
      res = art?j0>=m+n:cfun[j0];
      forn(i, m) res -= y[i] * arts[i] * rmat[i][j0];
      if (j0 < m+n && res < epsval) break;
    }

    forn(i, m) forn(j, m) B[i][j] = rmat[i][base[j]];
    forn(i, m) c[i] = rmat[i][j0];
    resolver(B, c, d);
    forn(i, m) c[i] = bvec[i];
    resolver(B, c, y);

    if (j0 == nn) if (art) {
      if (esab[m+n] != -1 && y[esab[m+n]] > epsval) return NOSOL;
      for(int i = m+n-1; i >= 0; i--) if (esab[i] == -1) { esab[i] = esab[m+n];
          base[esab[i]] = i; break; }
      art = 0; nn = m+n; j0 = 0; continue;
    } else break; // Optimo

    bool bl = true;
    forn(i, m) bl = bl && (d[i] <= eps);

    if (bl) return NOCOTA; // Problema no acotado

    int j1 = 0;
    forn(i, m) if (d[i] > 0) {
      tipo mlt = y[i] / d[i];
      if (!bl || (feq(mlt, res) && (base[i] < j1)) || (mlt < res)) {
        res = mlt;
        j1 = base[i];
        bl = true;
      }
    }
    if (res < eps && ++j0) continue;
    if (art && j1 == m+n) nn--, art--;

    int w = esab[j1];      // variable de salida
    base[w] = j0;           // Entra j0
    esab[j0] = w;
    esab[j1] = -1;          // j1 es no basica ahora.
    j0 = 0;
  } while(1);

  forn(i, m) forn(j, m) B[i][j] = rmat[i][base[j]];
  forn(i, m) c[i] = bvec[i];
  resolver(B, c, y);

  forn(i, n) xvar[i] = (esab[i] == -1)?0:y[esab[i]];

  return HAYSOL;
}
```

## 5.6. Factorización QR de Householder

Descompone $A = Q \cdot R$. Observación: $|det(A)| = |det(R)|$.

```
typedef vector<vector<tipo> > Mat;
typedef vector<tipo> Vec;
tipo sqr(tipo x) {return x*x;}

void show(Mat &a);

void qr(const Mat &a, Mat &q, Mat &r) {
  int n = a.size();
  r = a;
  q = Mat(n, Vec(n, 0));
  forn(i, n) forn(j, n) q[i][j] = (i==j);

  forn(k, n-1) {
    tipo beta = 0;
    forsn(i, k, n) beta += sqr(r[i][k]);
    tipo alph = sqrt(beta);
    if (alph * r[k][k] >= 0) alph = -alph;

    Vec v(n, 0);
    forsn(i, k, n) v[i] = r[i][k]; v[k] -= alph;
    beta += sqr(v[k]) - sqr(r[k][k]);

    #define QRmult(X) \
    forn(i, n) { tipo w = 0; \
      forsn(j, k, n) w += X * v[j]; w /= beta/2; \
```

```
      forsn(j, k, n) X -= w * v[j]; }

    // Q := Q * (I - 2 v * v^t) = Q - 2 * ((Q * v) * v^t)
    QRmult(q[i][j]);
    // A := Qj * A; \equiv A^t := A^t * Qj;
    QRmult(r[j][i]);

    forsn(i, k+1, n) r[i][k] = 0;
  }
}

// QR para calcular autvalores (no estoy seguro de para qu matrices sirve)
Mat operator* (const Mat &ml, const Mat &mr) {
  int a = ml.size(), b = mr.size(), c = mr[0].size();
  Mat res(a, Vec(c, 0));
  forn(i, a) forn(j, c) forn(k, b) res[i][j] += ml[i][k] * mr[k][j];
  return res;
}

#define iterac ???
void autoval(Mat &a) {
  int n = a.size();
  Mat q(n, Vec(n, 0));
  forn(i, iterac) {
    qr(a, q, a);
    a = a * q;
  }
  // Los autovalores convergen en la diagonal de "a"
}
```

## 5.7. Multiplicación de Karatsuba

`BASE` y `BASEEXP` deben ser tales que $\texttt{BASE} = 10^{\texttt{BASEEXP}}$ y además, $\texttt{BASE}^2 \cdot \mathit{largo}$ entre en un `int` o `tint`, según el caso.
Los números se representan en base `BASE` con la parte menos significativa en los índices más bajos.

```
#define BASE 1000000
#define BASEXP 6

typedef tint tipo; // o int

tipo* ini(int l){
  tipo *r = new tipo[l];
  fill(r, r+l, 0);
  return r;
}
#define add(l,s,d,k)forn(i, l)(d)[i]+=(s)[i]*k
void mulFast(int l, tipo *n1, tipo *n2, tipo *nr){
  if(l<=0)return;
  if(l<35){
    forn(i, l)forn(j, l)nr[i+j]+=n1[i]*n2[j];
  }else{
    int lac = l/2, lbd = l - (l/2);
    tipo *a = n1, *b=n1+lac, *c=n2, *d=n2+lac;
    tipo *ab = ini(lbd+1), *cd = ini(lbd+1);
    tipo *ac = ini(lac+lac), *bd = ini(lbd+lbd);
    add(lac, a, ab, 1);
    add(lbd, b, ab, 1);
    add(lac, c, cd, 1);
    add(lbd, d, cd, 1);
    mulFast(lac, a, c, ac);
    mulFast(lbd, b, d, bd);
    add(lac+lac, ac, nr+lac,-1);
    add(lbd+lbd, bd, nr+lac,-1);
    add(lac+lac, ac, nr,1);
    add(lbd+lbd, bd, nr+lac+lac,1);
    mulFast(lbd+1, ab, cd, nr+lac);
    free(ab); free(cd); free(ac); free(bd);
  }
}
void mulFast(int l1, tipo *n1, int l2, tipo *n2, int &lr, tipo *nr){
  while(l1<l2) n1[l1++]=0;
  while(l2<l1) n2[l2++]=0;
  lr=l1+l2+3;
  fill(nr, nr+lr, 0);
  mulFast(l1, n1, n2, nr);

  tipo r = 0;
  forn(i, lr){
    tipo q = r+nr[i];
    nr[i] = q%BASE,r = q/BASE;
  }
  while(lr>1 && nr[lr-1]==0)lr--;
}

// Cosas extra (convierten entre base 10 y 10^n)
void base10ton(int &l, tipo* n) {
  tipo p10[BASEXP]; p10[0] = 1;
  forn(i, BASEXP-1) p10[i+1] = p10[i] * 10;

  int nl = (l+BASEXP-1)/BASEXP;
```

```
  forsn(i, l, nl*BASEXP) n[i] = 0;
  forn(i, nl) {
    tint s = 0;
    forn(j, BASEXP) s+= n[i*BASEXP+j]*p10[j];
    n[i] = s;
  }
  l = nl;
}

void baseNto10(int &l, tipo* n) {
  for(int i = l-1; i>=0; --i) {
    tipo v = n[i];
    forn(j, BASEXP) {
      n[i*BASEXP+j] = v % 10; v /= 10;
    }
  }
  l = l*BASEXP;
  while (!n[l-1] && l > 1) l--;
}
```

## 5.8. Long - Entero largo

```
typedef tint tipo;
#define BASEXP 6
#define BASE 1000000
#define LMAX 1000

struct Long {
  int l;
  tipo n[LMAX];
  Long(tipo x) {  l = 0; forn(i, LMAX) { n[i]=x%BASE; l+=!!x||!i; x/=BASE;} }
  Long(){*this = Long(0);}
  Long(string x) {
    l=(x.size()-1)/BASEXP+1;
    fill(n, n+LMAX, 0);
    tipo r=1;
    forn(i,x.size()){
      n[i / BASEXP] += r * (x[x.size()-1-i]-'0');
      r*=10; if(r==BASE)r=1;
    }
  }
};

void out(Long& a) {
  char msg[BASEXP+1];
  cout << a.n[a.l-1];
  dforn(i,a.l-1) {
    sprintf(msg, "%6.6llu", a.n[i]); cout << msg; // 6 = BASEXP !
  }
  cout << endl;
}
void invar(Long &a) {
  fill(a.n+a.l, a.n+LMAX, 0);
  while(a.l>1 && !a.n[a.l-1]) a.l--;
}

void lsuma(const Long&a, const Long&b, Long&c) { // c = a + b
  c.l = max(a.l, b.l);
  tipo q = 0;
  forn(i, c.l) q += a.n[i]+b.n[i], c.n[i]=q%BASE, q/=BASE;
  if(q) c.n[c.l++] = q;
  invar(c);
}
Long& operator+= (Long&a, const Long&b) { lsuma(a, b, a); return a; }
Long operator+ (const Long&a, const Long&b) { Long c; lsuma(a, b, c); return c; }

bool lresta(const Long&a, const Long&b, Long&c) { // c = a - b
  c.l = max(a.l, b.l);
  tipo q = 0;
  forn(i, c.l) q += a.n[i]-b.n[i], c.n[i]=(q+BASE)%BASE, q=(q+BASE)/BASE-1;
  invar(c);
  return !q;
}
Long& operator-= (Long&a, const Long&b) { lresta(a, b, a); return a; }
Long operator- (const Long&a, const Long&b) {Long c; lresta(a, b, c); return c;}

bool operator< (const Long&a, const Long&b) { Long c; return !lresta(a, b, c); }
bool operator<= (const Long&a, const Long&b) { Long c; return lresta(b, a, c); }
bool operator== (const Long&a, const Long&b) { return a <= b && b <= a; }

void lmul(const Long&a, const Long&b, Long&c) { // c = a * b
  c.l = a.l+b.l;
  fill(c.n, c.n+b.l, 0);
  forn(i, a.l) {
    tipo q = 0;
    forn(j, b.l) q += a.n[i]*b.n[j]+c.n[i+j], c.n[i+j] = q%BASE, q/=BASE;
    c.n[i+b.l] = q;
  }
  invar(c);
}

```

```
Long& operator*= (Long&a, const Long&b) { Long c; lmul(a, b, c); return a=c; }
Long operator* (const Long&a, const Long&b) { Long c; lmul(a, b, c); return c; }

void lmul(const Long&a, int b, Long&c) { // c = a * b
  int q = 0;
  forn(i, a.l) q += a.n[i]*b, c.n[i] = q%BASE, q/=BASE;
  c.l = a.l;
  while(q) c.n[c.l++] = q%BASE, q/=BASE;
}

Long& operator*= (Long&a, int b) { lmul(a, b, a); return a; }
Long operator* (const Long&a, int b) { Long c = a; c*=b; return c; }

void ldiv(const Long& a, tipo b, Long& c, tipo& rm) { // c = a / b ; rm = a % b
  rm = 0;
  dforn(i, a.l) {
    rm = rm * BASE + a.n[i];
    c.n[i] = rm / b; rm %= b;
  }
  c.l = a.l;
  invar(c);
}

void ldiv(const Long& a, const Long& b, Long& c, Long& rm) { // c = a / b ; rm =
     a % b
  rm = 0;
  dforn(i, a.l) {
    dforn(j, rm.l) rm.n[j+1] = rm.n[j];
    rm.n[0] = a.n[i]; rm.l++;
    tipo q = rm.n[b.l] * BASE + rm.n[b.l-1];
    tipo u = q / (b.n[b.l-1] + 1);
    tipo v = q /  b.n[b.l-1] + 1;
    while (u < v-1) {
      tipo m = (u+v)/2;
      if (b*m <= rm) u = m; else v = m;
    }
    c.n[i] = u;
    rm -= b*u;
  }
  c.l = a.l;
  invar(c);
}
```

## 5.9. Fracción

```
usa: algorithm, tint, mcd
struct frac {
  tint p,q;
  frac(tint num=0, tint den=1):p(num),q(den) { norm(); }
  frac& operator+=(const frac& o){
    tint a = mcd(q,o.q);
    p=p*(o.q/a)+o.p*(q/a);
    q*=(o.q/a);
    norm();
    return *this;
  }
  frac& operator-=(const frac& o){
    tint a = mcd(q,o.q);
    p=p*(o.q/a)-o.p*(q/a);
    q*=(o.q/a);
    norm();
    return *this;
  }
  frac& operator*=(frac o){
    tint a = mcd(q,o.p);
    tint b = mcd(o.q,p);
    p=(p/b)*(o.p/a);
    q=(q/a)*(o.q/b);
    return *this;
  }
  frac& operator/=(frac o){
    tint a = mcd(q,o.q);
    tint b = mcd(o.p,p);
    p=(p/b)*(o.q/a);
    q=(q/a)*(o.p/b);
    norm();
    return *this;
  }

  void norm(){
    tint aux = mcd(p,q);
    if (aux){ p/=aux; q/=aux; }
    else { q=1; }
    if (q<0) { q=-q; p=-p; }
  }
};
```

# 6. Cosas

## 6.1. Morris-Prath

```
tint pmp[MAXL];
void preMp(string& x){
  tint i=0, j = pmp[0] = -1;
  while(i<(tint)x.size()){
    while(j>-1 && x[i] != x[j]) j = pmp[j];
      pmp[++i] = ++j;
  }
}
void mp(string& b, string& g){
  preMp(b);
  tint i=0,j=0;
  while(j<(tint)g.size()){
    while(i>-1 && b[i] != g[j]){i = pmp[i];}
    i++; j++;
    if (i>=(tint)b.size()){
      OUTPUT(j - i);
      i=pmp[i];
    }
  }
}
```

## 6.2. Subsecuencia común más larga

```
tint lcs(vector<tint> a, vector<tint> b) { // Longest Common Subsequence
  vector< vector<tint> > m(2, vector<tint>(b.size()+1));
  forn(i,a.size())forn(j,b.size())
    m[1-i%2][j+1]=(a[i]==b[j]?m[i%2][j]+1:max(m[i%2][j+1],m[1-i%2][j]));
  return m[a.size()%2][b.size()];
}
```

## 6.3. SAT - 2

```
usa: stack
#define MAXN 1024
#define MAXEQ 1024000

int fch[2*MAXN], nch[2*MAXEQ], dst[2*MAXEQ], eqs; // Grafo
#define addeje(s,d) { nch[eqs]=fch[s]; dst[fch[s]=eqs++]=d; }
#define neg(X) (2*MAXN-1-(X))
void init() {
  memset(fch, 0xff, sizeof(fch));
  eqs=0;
}
void addEqu(int a, int b) {
  addeje(neg(a), b);
  addeje(neg(b), a);
}
int us[2*MAXN], lw[2*MAXN], id[2*MAXN];
stack<int> q; int qv, cp;
void tjn(int i) {
  lw[i] = us[i] = ++qv;
  id[i]=-2; q.push(i);
  for(int j = fch[i]; j!=-1; j=nch[j]) { int x = dst[j];
    if (!us[x] || id[x] == -2) {
      if (!us[x]) tjn(x);
      lw[i] = min(lw[i], lw[x]);
    }
  }
  if (lw[i] == us[i]) {
    int x; do { x = q.top(); q.pop(); id[x]=cp; } while (x!=i);
    cp++;
  }
}
void compCon(int n) { // Tarjan algorithm
  memset(us, 0, sizeof(us));
  memset(id, -1, sizeof(id));
  q=stack<int>(); qv = cp = 0;
  forn(i, n) {
    if (!us[i]) tjn(i);
    if (!us[neg(i)]) tjn(neg(i));
  }
}
bool satisf(int n) {
  compCon(n);
  forn(i, n) if (id[i] == id[neg(i)]) return false;
  return true;
}
```

## 6.4. Male-optimal stable marriage problem O($N^2$)

gv[i][j] es la $j$-esima mujer en orden de preferencia en la lista del varon $i$.
om[i][j] es la posición que ocupa el hombre $j$ en la lista de la mujer $i$.

```
#define MAXN 1000
int gv[MAXN][MAXN],om[MAXN][MAXN]; // Inpu del algoritmo
int pv[MAXN],pm[MAXN];             // Oupu del algoritmo
int pun[MAXN];                     // Auxiliar

void stableMarriage(int n) {
  fill_n(pv,n,-1); fill_n(pm,n,-1); fill_n(pun,n,0);
  int s = n, i = n-1;
```

```
  #define engage pm[j] = i; pv[i] = j;
  while (s) {
    while (pv[i] == -1) {
      int j = gv[i][pun[i]++];
      if (pm[j] == -1) {
        s--; engage;
      }
      else if (om[j][i] < om[j][pm[j]]) {
        int loser = pm[j];
        pv[loser] = -1;
        engage;
        i = loser;
      }
    }
    i--; if (i < 0) i = n-1;
} }
```

## 6.5. Rotaciones del cubo

```
#define _ALTA {forn(h, 6) rot[p][h] = d[h]; p++;}
#define _DER forn(h, 6) d[h] = _der[d[h]];
#define _UP forn(h, 6) d[h] = _up[d[h]];

int rot[24][6];
const int _der[6] = {0, 2, 4, 1, 3, 5};
const int _up[6] = {1, 5, 2, 3, 0, 4};

void rotaciones() {
  int d[6];
  int p = 0;
  forn(i, 6) d[i] = i;                  / \
  forn(i, 2) {                  4 --> /   \ <-- 3
    forn(j, 3) {                     |\ 0 /|
      _ALTA; _DER;                   | \ / |
      _ALTA; _DER;                   |2 | 1|
      _ALTA; _DER;                    \ | /
      _ALTA; _UP;                      \|/
    }                                   ^
    _DER; _UP; _UP;                     5
  }
  return;
}
```

## 6.6. Poker

```
usa: list, vector, map, string, algorithm, forn, tint, pint
#define STRAIGHT_VALUE 14
#define FLUSH_VALUE 15
typedef pair<int,int> pint;
typedef vector< pint > hand;
typedef vector< int > puntaje;

int cantPairs(hand& m) {
    int pares=0;
    forn(i,m.size()) forn(j,i) if (m[i].first == m[j].first)
    pares++;
    return pares;
}

int isStraight(hand& m) {
    sort(m.begin(), m.end());
    int ls=4;
    if (m[4].first==14 && m[0].first==2) ls=3; //esta linea acepta escaleras
        desde el A
    forn(i, ls) if (m[i].first != m[i+1].first - 1) return 0;
    return STRAIGHT_VALUE;
}

int isFlush(hand& m) {
    forn(i, m.size()-1) if (m[i].second != m[i+1].second) return 0;
    return FLUSH_VALUE;
}

int gamePoints(hand& m) {
    int f=isFlush(m),s=isStraight(m),p=cantPairs(m) * 4;
    return max(f+s,p); //esto esta para aceptar cartas duplicadas
}

puntaje points(hand& m) {
    puntaje r;
    r.push_back(gamePoints(m));
    map<int, int> c;
    int i;
    forn(i,m.size()) c[m[i].first]++;
    vector<pint> cants;
    map<int, int>::iterator it;
    for(it = c.begin() ; it != c.end() ; ++it) {
        cants.push_back( pint( it->second, it->first ) );
    }
    sort(cants.begin(), cants.end());
    forn(i, cants.size()) {
        r.push_back(cants[cants.size()-1-i].second);
```

```
    }
    //esta linea que sigue arregla la comparacion con escaleras que empiezan
        desde A
    if ((r[0]==FLUSH_VALUE || r[0]==FLUSH_VALUE+STRAIGHT_VALUE) && r[1]==14 && r
        [2]!=13) r[1]=1;
    return r;
}
tint comp(hand& m1, hand& m2) {
  puntaje n1 = points(m1); puntaje n2 = points(m2);
  return (n1 > n2 ? 1 : n1 == n2 ? 0 : -1);
}
tint convN(char c) {
  switch(c) {
  case 'A': return 14; case 'K': return 13; case 'Q': return 12;
  case 'J': return 11; case 'T': return 10;
  } return c - '0';
}
pint readCard() {
  string s; cin >> s;
  return (s == "" ? pint(-1,-1) : pint(convN(s[0]), s[1]));
}
hand readHand() { hand r;
  forn(i,5) {
    pint c = readCard();
    if (c == pint(-1,-1)) return hand();
    r.push_back(c);
  } return r;
}
```

# 7. Extras

## 7.1. Convex Hull en 3D

Le das un mar de puntos y un triangulito inicial en una cara de la convex hull.

```
usa: cstdio, vector, queue, iostream, fstream, cmath

const double KETO = 1e-9;
typedef long double tdbl;
inline tint sqr(tint a){return a*a; }
struct pto{tint x,y,z;};
pto point(tint x, tint y, tint z){pto r; r.x=x; r.y=y;r.z=z; return r;}

pto operator - (pto a, pto b) { return point(a.x-b.x, a.y-b.y, a.z-b.z); }
pto operator ^ (pto a, pto b) { return point(a.y*b.z-a.z*b.y,
a.z*b.x-a.x*b.z, a.x*b.y-a.y*b.x); }
tint operator * (pto a, pto b) { return a.x*b.x + a.y*b.y + a.z*b.z; }
bool operator == (pto a, pto b) { return a.x==b.x && a.y==b.y && a.z==b.z; }
tdbl len (pto a) { return sqrt(1.0*(a*a)); }
tint len2(pto a) { return a*a; }
ifstream in("d.in");
ifstream out("d.out");
#define FS first
#define SD second
#define MP make_pair
bool ok[1700][1700];
int main () {
  int runs; in >> runs;
  while (runs--) {
    vector<pto> p;
      int x1,y1,x2,y2;
      in >> x1 >> y1 >> x2 >> y2;
      p.push_back(point(x1,y1,0)); p.push_back(point(x2,y1,0));
      p.push_back(point(x2,y2,0)); p.push_back(point(x1,y2,0));
    int N; in >> N;
    tdbl area=-abs(x2-x1)*abs(y2-y1), area2;
    if(N){
      forn(i, N){
          int h; in >> x1 >> y1 >> x2 >> y2 >> h;
          p.push_back(point(x1,y1,h)); p.push_back(point(x2,y1,h));
          p.push_back(point(x2,y2,h)); p.push_back(point(x1,y2,h));
      }
      fill(ok[0], ok[p.size()], false);
      queue<pair<pair<int, int>, int> > q;
      q.push(MP(MP(0,1),2));
      while (!q.empty()) {
        int A = q.front().FS.FS;
        int B = q.front().FS.SD;
        int x = q.front().SD; q.pop();
        if (ok[A][B]) continue;
        tdbl Ccos3D = -1e100;
        tdbl Ccos2D = -1e100;
        tdbl Cdist  = -1e100;
          int C = -1;
        pto n = (p[x]-p[B]) ^ (p[x]-p[A]);
        forn(i, p.size()){
          if (ok[B][i] || ok[i][A]) continue;
          pto mi = (p[i]-p[A]) ^ (p[i]-p[B]);
          if (mi.x==0&&mi.y==0&&mi.z==0) continue;

          tdbl icos3D = tdbl (mi*n) / len(mi) / len(n);
          tdbl icos2D = tdbl ((p[i]-p[B])*(p[B]-p[A])) / len(p[i]-p[B]) / len(p[
              B]-p[A]);
```

```
        tdbl idist  = len(mi);

        if ((icos3D>Ccos3D+KETO) ||
            (icos3D>Ccos3D-KETO && icos2D>Ccos2D+KETO) ||
            (icos3D>Ccos3D-KETO && icos2D>Ccos2D-KETO && Cdist<idist)) {
          C = i;
          Ccos3D = icos3D;
          Ccos2D = icos2D;
          Cdist  = idist;
        }
      }
      ok[A][B]=ok[B][C]=ok[C][A]=true;
      q.push(MP(MP(C,B), A));
      q.push(MP(MP(A,C), B));
      area += 0.5 * len((p[C]-p[A]) ^ (p[C]-p[B]));
    }
  }else{
    area = -area;
  }
  out >> area2;
  if(abs(area2-area)>1e-4){
    cout << "MAL" << endl;
  }
}
cout << "FIN" << endl;
return 0;
}
```

## 7.2. Componentes conexas en un subgrafo grilla

```
int dx[4]={0,0,-1,1}, dy[4]={-1,1,0,0};
struct Cas{int p[4];};
const int MAXN = 105;
Cas c[MAXN*2][MAXN*2];
int px, py;
void put(int x, int y, int d, int l, int t){
  forn(i, l){
    if(d==0)c[x+i][y].p[0] = c[x+i][y-1].p[1] = t;
    if(d==1)c[x][y+i].p[2] = c[x-1][y+i].p[3] = t;
  }
}
void init(){
  Cas cc; fill(cc.p, cc.p+4, 0);
  fill(c[0], c[MAXN], cc);
}
```

## 7.3. Orden total de puntos alrededor de un centro

```
struct Cmp{
  pto r;
  Cmp(pto _r){r = _r;}
  int cuad(const pto &a) const{
    if(a.x > 0 && a.y >= 0)return 0;
    if(a.x <= 0 && a.y > 0)return 1;
    if(a.x < 0 && a.y <= 0)return 2;
    if(a.x >= 0 && a.y < 0)return 3;
    assert(a.x ==0 && a.y==0);
    return -1;
  }
  bool cmp(const pto&p1, const pto&p2)const{
    int c1 = cuad(p1), c2 = cuad(p2);
    if(c1==c2){
      return p1.y*p2.x<p1.x*p2.y;
    }else{
      return c1 < c2;
    }
  }
  bool operator()(const pto&p1, const pto&p2) const{
    return cmp(pto(p1.x-r.x,p1.y-r.y),pto(p2.x-r.x,p2.y-r.y));
  }
};
```