

Índice

1. Algoritmos	1
2. Estructuras	2
2.1. Range Minimum Query $\langle n \log n, 1 \rangle$ (get)	2
2.2. Range Minimum Query $\langle n, \log n \rangle$ (get y set)	2
2.3. RMQ con Lazy Update	2
2.4. Suffix Array - Longest Common Prefix	3
3. Geom	3
3.1. Point in Poly	3
3.2. Convex Hull	4
3.3. Circulo mínimo	4
3.4. Máximo rectángulo entre puntos	4
3.5. Centro de masa y area de un polígono	5
3.6. Par de puntos mas cercano	5
3.7. Sweep Line	6
3.8. Intersección de segmentos	6
3.9. Distancia entre segmentos	6
3.10. Cuentitas	6
4. Grafos	8
4.1. Kruskal & Union-Find	8
4.2. Edmond-Karp	9
4.3. Flujo de costo mínimo	9
4.4. Matching perfecto de costo máximo - Hungarian $O(N^3)$	9
4.5. Camino/Circuito Euleriano	10
4.6. Erdős-Gallai	11
4.7. Grafo cactus	11
5. Matemática	11
5.1. Algoritmos de cuentas	11
5.1.1. MCD	11
5.1.2. Teorema Chino del Resto	11
5.1.3. Longitud de los números de 1 a N	12
5.2. Teoremas y propiedades	12
5.2.1. Ecuación de grafo planar	12
5.2.2. Ternas pitagóricas	12
5.2.3. Teorema de Pick	12
5.2.4. Propiedades varias	12
5.3. Tablas y cotas	12
5.3.1. Primos	12
5.3.2. Divisores	12
5.3.3. Factoriales	13
5.4. Solución de Sistemas Lineales	13

5.5. Programación Lineal - Simplex	13
5.6. Factorización QR de Householder	14
5.7. Multiplicación de Karatsuba	15
5.8. Long - Entero largo	16
5.9. Fracción	17
6. Cosas	18
6.1. Morris-Prath	18
6.2. SAT - 2	18
6.3. Rotaciones del cubo	18
7. Extras	19
7.1. Orden total de puntos alrededor de un centro	19
8. El AJI es una fruta	19
8.1. Dinitz	19
8.2. Intersección (y yerbas afines) de circulos en $O(n^3 \lg n)$	20
8.3. Integrador numerico (simpson).	21
8.4. Componentes biconexas, puentes y puntos de articulacion by Juancito	21
8.5. Rotaciones	21
8.6. LIS	21
8.7. Flujo de costo minimo vale multiejes	22
8.8. Dual simple (dual sobre cada componente conexa)	22
8.9. Dual full	23
8.10. LCA	24
8.11. Interseccion semiplano-poligono convexo $O(n)$	24
8.12. Distancia punto-trianguulo en 3D	24
8.13. Algoritmo de Duval	25

1. Algoritmos

#include <algorithm> #include <numeric>

Algo	Params	Funcion
sort, stable_sort	f, l	ordena el intervalo
partial_sort	f, m, l, [cmp]	[f,m] son los m-f menores en orden, [m,l] es el resto en algun orden
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 $\forall i$

Algo	Params	Funcion
find, find_if, find_first_of	f, l, elem / pred / f2, l2	it encuentra $i \in [f, l]$ tq. $i = \text{elem}$, $\text{pred}(i)$, $i \in [f2, l2]$
count, count_if	f, l, elem/pred	cuenta elem, $\text{pred}(i)$
search	f, l, f2, l2	busca $[f2, l2] \in [f, l]$
replace, replace_if	f, l, old / pred, new	cambia old / $\text{pred}(i)$ por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	$\text{pred}(i)$ ad, $!\text{pred}(i)$ atras
min_element, max_element	f, l, [comp]	it min, max de $[f, l]$
lexicographical_compare	f1, l1, f2, l2	bool con $[f1, l1]_i [f2, l2]_j$
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]	$T = \sum / \text{oper de } [f, l]$
inner_product	f1, l1, f2, i	$T = i + [f1, l1] \cdot [f2, \dots]$
partial_sum	f, l, r, [op]	$r+i = \sum / \text{oper de } [f, f+i] \forall i \in [f, l]$
adjacent_difference	f, l, r, [op]	$r[0]=f[0]$, $r[i]=f[i] - f[i-1] \forall i \in [1, l-f]$

2. Estructuras

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

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

```

1  usa: tipo
2  #define LVL 10
3  tipo vec[LVL] [1<<LVL];
4  tipo mn(int i, int j) { // intervalo [i, j)
5      int p = 31-__builtin_clz(j-i);
6      return min(vec[p][i], vec[p][j-(1<<p)]);
7  }
8  void mn_init(int n) {
9      int mp = 31-__builtin_clz(n);
10     forn(p, mp) forn(x, n-(1<<p)) vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)]);
11 }

```

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();
```

```

1  #define MAXN 100000
2  struct rmq {
3      int MAX;
4      tipo vec[4*MAXN];
5      tipo* init(int n) {
6          MAX = 1 << (32-__builtin_clz(n));
7          fill(vec, vec+2*MAX, 0); // 0 = elemento neutro
8          return vec+MAX;
9      }
10     void updall() { dforn(i, MAX) vec[i] = vec[2*i] + vec[2*i+1]; } // + =
        operacion
11     void pset(int i, tipo vl) {
12         vec[i+MAX] = vl;
13         while(i) { i /= 2; vec[i] = vec[2*i] + vec[2*i+1]; } // + = operacion
14     }
15     tipo pget(int i, int j) { return _pget(i+MAX, j+MAX); }
16     tipo _pget(int i, int j) {
17         tipo res = 0; // 0 = elemento neutro
18         if (j-i <= 0) return res;
19         if (i%2) res += vec[i++]; // + = operaci'on
20         res += _pget(i/2, j/2); // + = operaci'on
21         if (j%2) res += vec[--j]; // + = operaci'on
22         return res;
23     }
24 }

```

2.3. RMQ con Lazy Update

Funciona con cualquier operación de asignación “=”, cualquier operación asociativa de combinación “+”, y un elemento nulo de la segunda operación “0”. También es necesaria una función que dado un rango, el valor inicial de ese rango y un cambio pueda decidir el nuevo valor en ese rango. maxn es la máxima cantidad de elementos que tiene la estructura. get(a, b) y set(v, a, b) funcionan en [a, b) Líneas subrayadas no son necesarias si no se usa Lazy Updating; si no se incluyen solo se puede usar set(v, a, a + 1).

```

1  const int maxn = 100010;
2  const int maxpn = 1 << (32-__builtin_clz(maxn));
3
4  tipo V[4 * maxn];
5  tipo L[4 * maxn];
6
7  void update(int n, int a, int b) {
8      if (L[n] == 0) return ; // 0 -> elemento nulo.
9      if (b - a > 1) {
10         L[2 * n] += L[n]; // = -> operador de asignacion.
11         L[2 * n + 1] += L[n]; // = -> operador de asignacion
12     }
13 }

```

```

14 // V[n] = funcion en el rango [a, b) con valor original V[n] y cambio L[n].
15 V[n] += ((tipo) b-a) * L[n];
16 L[n] = 0; // 0 -> elemento nulo
17 }
18
19 void init() {
20     fill(V, V + 2 * maxpn, 0); //0 -> elemento nulo.
21     fill(L, L + 2 * maxpn, 0); //0 -> elemento nulo
22 }
23
24 tipo get(int q, int w, int n = 1, int a = 0, int b = maxpn) {
25     q = max(q, a);
26     w = min(w, b);
27     update(n, a, b);
28
29     if(q >= w) return 0; //0 -> elemento nulo
30     if(q == a && w == b) return V[n];
31     return get(q, w, 2*n, a, (a+b) / 2) + get(q, w, 2 * n + 1, (a+b) / 2, b); // +
        -> Operador asociativo
32 }
33
34 tipo set(tipo v, int q, int w, int n = 1, int a = 0, int b = maxpn) {
35     q = max(q, a);
36     w = min(w, b);
37
38     if (q == a && w == b) {
39         L[n] += v; // = -> operador de asignacion
40     }
41     else if (q < w) V[n] = set(v, q, w, 2 * n, a, (a+b)/2) + set(v, q, w, 2 * n +
        1, (a+b) / 2, b); // + -> operador asociativo.
42
43     update(n, a, b);
44     return V[n];
45 }

```

2.4. Suffix Array - Longest Common Prefix

```

1 typedef unsigned char xchar;
2 #define MAXN 1000000
3
4 int p[MAXN], r[MAXN], t, n;
5
6 bool sacmp(int a, int b) { return p[(a+t)%n] < p[(b+t)%n]; }
7 void bwt(const xchar *s, int nn) {
8     n = nn;
9     int bc[256];
10    memset(bc, 0, sizeof(bc));
11    forn(i, n) ++bc[s[i]];
12    forn(i, 255) bc[i+1] += bc[i];

```

```

13    forn(i, n) r[--bc[s[i]]] = i;
14    forn(i, n) p[i] = bc[s[i]];
15
16    int lnb, nb = 1;
17    for(t = 1; t < n; t *= 2) {
18        lnb = nb; nb = 0;
19        for(int i = 0, j = 1; i < n; i = j++) {
20            /*calcular siguiente bucket*/
21            while(j < n && p[r[j]] == p[r[i]]) ++j;
22            if (j-i > 1) {
23                sort(r+i, r+j, sacmp);
24                int pk, opk = p[(r[i]+t)%n];
25                int q = i, v = i;
26                for(; i < j; i++) {
27                    if (((pk = p[(r[i]+t)%n]) != opk) && !(q <= opk && pk < j)) { opk = pk
                        ; v = i; }
28                    p[r[i]] = v;
29                }
30            }
31            nb++;
32        }
33        if (lnb == nb) break;
34    }
35    // prim = p[0];
36 }
37
38 void lcp(const xchar* s, int* h) { /* h could be over r */
39     int q = 0, j;
40     forn(i, n) if (p[i]) {
41         j = r[p[i]-1];
42         while(q < n && s[(i+q)%n] == s[(j+q)%n]) ++q;
43         h[p[i]-1] = q;
44         if (q > 0) --q;
45     }
46 }

```

3. Geom

3.1. Point in Poly

```

1 usa: algorithm, vector
2 // No se porta bien si le preguntas por un punto que esta justo en la frontera
    del poligono
3 struct pto { tipo x,y; };
4 bool pnpoly(vector<pto>&v, pto p){
5     unsigned i, j, mi, mj, c = 0;
6     for(i=0, j = v.size()-1; i < v.size(); j = i++){
7         if((v[i].y <= p.y && p.y < v[j].y) ||
8             (v[j].y <= p.y && p.y < v[i].y)){
9             mi=i, mj=j; if(v[mi].y > v[mj].y) swap(mi, mj);

```

```

10     if((p.x-v[mi].x) * (v[mj].y-v[mi].y)
11         < (p.y-v[mi].y) * (v[mj].x-v[mi].x)) c^=1;
12     }
13 } return c;
14 }

```

3.2. Convex Hull

```

1  usa: algorithm, vector, sqr
2  tipo pcruz(tipo x1,tipo y1,tipo x2,tipo y2){return x1*y2-x2*y1;}
3  struct pto {
4      tipo x,y;
5      tipo n2(pto &p2) const{
6          return sqr(x-p2.x)+sqr(y-p2.y);
7      }
8  } r;
9  tipo area3(pto a, pto b, pto c){
10     return pcruz(b.x-a.x,b.y-a.y,c.x-a.x,c.y-a.y);
11 }
12 bool men2(const pto &p1, const pto &p2){
13     return (p1.y==p2.y)?(p1.x<p2.x):(p1.y<p2.y);
14 }
15 bool operator<(const pto &p1,const pto &p2){
16     tipo ar = area3(r,p1,p2);
17     return(ar==0)?(p1.n2(r)<p2.n2(r)):ar>0;
18     //< clockwise, >counterclockwise
19 }
20 typedef vector<pto> VP;
21 VP chull(VP &l){
22     VP res = l; if(l.size()<3) return res;
23     r = *(min_element(res.begin(),res.end(),men2));
24     sort(res.begin(),res.end());
25     tint i=0;VP ch;ch.push_back(res[i++]);ch.push_back(res[i++]);
26     while(i<res.size()) // area3 > clockwise, < counterclockwise
27         if(ch.size()>1 && area3(ch[ch.size()-2],ch[ch.size()-1],res[i])<=0)
28             ch.pop_back();
29         else
30             ch.push_back(res[i++]);
31     return ch;
32 }

```

3.3. Circulo mínimo

```

1  usa: algorithm, cmath, vector, pto (con < e ==)
2  usa: sqr, dist2(pto,pto), tint
3  typedef double tipo;
4  typedef vector<pto> VP;
5  struct circ { tipo r; pto c; };
6  #define eq(a,b) (fabs(a-b)<0.000000000000001)
7  circ deIni(VP v){ //l.size()<=3

```

```

8     circ r; sort(v.begin(), v.end()); unique(v.begin(), v.end());
9     switch(v.size()) {
10         case 0: r.c.x=r.c.y=0; r.r = -1; break;
11         case 1: r.c=v[0]; r.r=0; break;
12         case 2: r.c.x=(v[0].x+v[1].x)/2.0;
13                 r.c.y=(v[0].y+v[1].y)/2.0;
14                 r.r=dist2(v[0], r.c); break;
15         default: {
16             tipo A = 2.0 * (v[0].x-v[2].x); tipo B = 2.0 * (v[0].y-v[2].y);
17             tipo C = 2.0 * (v[1].x-v[2].x); tipo D = 2.0 * (v[1].y-v[2].y);
18             tipo R = sqr(v[0].x)-sqr(v[2].x)+sqr(v[0].y)-sqr(v[2].y);
19             tipo P = sqr(v[1].x)-sqr(v[2].x)+sqr(v[1].y)-sqr(v[2].y);
20             tipo det = D*A-B*C;
21             if(eq(det, 0)) {swap(v[1],v[2]); v.pop_back(); return deIni(v);}
22             r.c.x = ( D*R-B*P)/det;
23             r.c.y = (-C*R+A*P)/det;
24             r.r = dist2(v[0],r.c);
25         }
26     }
27     return r;
28 }
29 circ minDisc(VP::iterator ini,VP::iterator fin,VP& pIni){
30     VP::iterator ivp;
31     int i,cantP=pIni.size();
32     for(ivp=ini,i=0;i+cantP<2 && ivp!=fin;ivp++,i++) pIni.push_back(*ivp);
33     circ r = deIni(pIni);
34     for(;i>0;i--) pIni.pop_back();
35     for(;ivp!=fin;ivp++) if (dist2(*ivp, r.c) > r.r){
36         pIni.push_back(*ivp);
37         if (cantP<2) r=minDisc(ini,ivp,pIni);
38         else r=deIni(pIni);
39         pIni.pop_back();
40     }
41     return r;
42 }
43 circ minDisc(VP ps){ //ESTA ES LA QUE SE USA
44     random_shuffle(ps.begin(),ps.end()); VP e;
45     circ r = minDisc(ps.begin(),ps.end(),e);
46     r.r=sqrt(r.r); return r;
47 };

```

3.4. Máximo rectángulo entre puntos

```

1  usa: vector, map, algorithm
2  struct pto {
3      tint x,y ;bool operator<(const pto&p2) const{
4          return (x==p2.x)?(y<p2.y):(x<p2.x);
5      }
6  };
7  bool us[10005];

```

```

8 vector<pto> v;
9 tint l,w;
10 tint maxAr(tint x, tint y,tint i){
11     tint marea=0;
12     tint arr=0,aba=w;
13     bool partido = false;
14     for(tint j=i;j<(tint)v.size();j++){
15         if(x>v[j].x)continue;
16         tint dx = (v[j].x-x);
17         if(!partido){
18             tint ar = (aba-arr) * dx;marea>?=ar;
19         } else {
20             tint ar = (aba-y) * dx;marea>?=ar;
21             ar = (y-arr) * dx;marea>?=ar;
22         }
23         if(v[j].y==y)partido=true;
24         if(v[j].y< y)arr>?=v[j].y;
25         if(v[j].y> y)aba<?=v[j].y;
26     }
27     return marea;
28 }
29 tint masacre(){
30     fill(us,us+10002,false);
31     pto c;c.x=0;c.y=0;v.push_back(c);c.x=l;c.y=w;v.push_back(c);
32     tint marea = 0;
33     sort(v.begin(),v.end());
34     for(tint i=0;i<(tint)v.size();i++){
35         us[v[i].y]=true;
36         marea>?=maxAr(v[i].x,v[i].y,i);
37     }
38     for(tint i=0;i<10002;i++)if(us[i])marea>?=maxAr(0,i,0);
39     return marea;
40 }

```

3.5. Centro de masa y area de un polígono

```

1  usa: vector, forn
2  struct pto { tint x,y; };
3  typedef vector<pto> poly;
4  tint pcruz(tint x1, tint y1, tint x2, tint y2) { return x1*y2-x2*y1; }
5  tint area3(const pto& p, const pto& p2, const pto& p3) {
6      return pcruz(p2.x-p.x, p2.y-p.y, p3.x-p.x, p3.y-p.y);
7  }
8  tint areaPor2(const poly& p) {
9      tint a = 0; tint l = p.size()-1;
10     forn(i,l-1) a += area3(p[i], p[i+1], p[l]);
11     return abs(a);
12 }
13 pto bariCentroPor3(const pto& p1, const pto& p2, const pto& p3) {
14     pto r;

```

```

15     r.x = p1.x+p2.x+p3.x; r.y = p1.y+p2.y+p3.y;
16     return r;
17 }
18 struct ptoD { double x,y; };
19 ptoD centro(const poly& p) {
20     tint a = 0; ptoD r; r.x=r.y=0; tint l = p.size()-1;
21     forn(i,l-1) {
22         tint act = area3(p[i], p[i+1], p[l]);
23         pto pact = bariCentroPor3(p[i], p[i+1], p[l]);
24         r.x += act * pact.x; r.y += act * pact.y; a += act;
25     } r.x /= (3 * a); r.y /= (3 * a); return r;
26 }

```

3.6. Par de puntos mas cercano

```

1  usa algorithm, vector, tdbl, tint, tipo, INF, forn, cmath
2  const tint MAX_N = 10010;
3  struct pto { tipo x,y;} r;
4  typedef vector<pto> VP;
5  #define ord(n,a,b) bool n(const pto &p, const pto &q){ \
6      return ((p.a==q.a)?(p.b<q.b):(p.a<q.a));}
7  #define sqr(a) ((a)*(a))
8  ord(mx,x,y);
9  ord(my,y,x);
10 bool vale(const pto &p){return mx(p,r);};
11 tipo dist(pto a,pto b){return sqr(a.x-b.x)+sqr(a.y-b.y);}
12 pto vx[MAX_N];
13 pto vy[MAX_N];
14 tint N;
15 tipo cpair(tint ini, tint fin){
16     if(fin-ini==1)return INF;
17     if(fin-ini==2)return dist(vx[ini], vx[ini+1]);
18     vector<pto> y(fin-ini);
19     copy(vy+ini, vy+fin, y.begin());
20     tint m = (ini+fin)/2;
21     r = vx[m];
22     stable_partition(vy+ini, vy+fin, vale);
23     tipo d = min(cpair(ini, m), cpair(m, fin));
24     vector<pto> w;
25     forn(i, y.size())if(sqr(fabs(y[i].x-vx[m].x))<=d)w.push_back(y[i]);
26     forn(i,w.size()){
27         for(tint j=i+1;(j<(tint)w.size())
28             && sqr(fabs(w[i].y-w[j].y))<d;j++){
29             d<?=dist(w[i],w[j]);
30         }
31     }
32     return d;
33 }
34 tipo closest_pair(){
35     sort(vx, vx+N,mx);

```

```

36 sort(vy, vy+N,my);
37 for(tint i=1;i<N;i++){
38     if(vx[i].x==vx[i-1].x && vx[i].y==vx[i-1].y)return 0;
39 }
40 return sqrt(cpair(0,N));
41 }

```

3.7. Sweep Line

```

1 struct pto { tint x,y; bool operator<(const pto&p2)const{
2     return (y==p2.y)?(x<p2.x):(y<p2.y);
3 };
4 struct slp{ tint x,y,i;bool f; bool operator<(const slp&p2)const{
5     if(y!=p2.y)return y<p2.y;
6     if(x!=p2.x)return x<p2.x;
7     if(f!=p2.f)return f;
8     return i<p2.i;
9 };
10 slp p2slp(pto p,tint i){slp q;q.x=p.x;q.y=p.y;q.i=i;return q;}
11 tint area3(pto a,pto b,pto c){
12     return (b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x);
13 }
14 tint giro(pto a,pto b,pto c){
15     tint a3=area3(a,b,c);
16     if(a3<0) return -1; if(a3>0)return 1;
17     return 0;
18 }
19 bool inter(pair<pto,pto> a, pair<pto,pto> b){
20     pto p=a.first,q=a.second,r=b.first,s=b.second;
21     if(q<p)swap(p,q);if(s<r)swap(r,s);
22     if(r<p){swap(p,r);swap(q,s);}
23     tint a1=giro(p,q,r),a2=giro(p,q,s);
24     if(a1!=0 || a2!=0){
25         return (a1!=a2) && (giro(r,s,p)!=giro(r,s,q));
26     } else {
27         return !(q<r);
28     }
29 }
30 tint cant_intersec(vector<pair<pto,pto> >&v){
31     tint ic=0;
32     set<slp> Q; list<tint> T;
33     for(tint i=0;i<(tint)v.size();i++){
34         slp p1=p2slp(v[i].first,i);slp p2=p2slp(v[i].second,i);
35         if(p2<p1)swap(p1,p2);
36         p1.f=true;p2.f=false;
37         Q.insert(p1);Q.insert(p2);
38     }
39     while(Q.size()>0){
40         slp p = *(Q.begin());Q.erase(p);
41         if(p.f){

```

```

42         for(list<tint>::iterator it=T.begin();it!=T.end();it++)
43             if(inter(v[*it],v[p.i]))ic++;
44         T.push_back(p.i);
45     } else {
46         T.erase(find(T.begin(),T.end(),p.i));
47     }
48 }
49 return ic;
50 }

```

3.8. Intersección de segmentos

```

1 struct pto{tint x,y;};
2 struct seg{pto f,s;};
3 tint sgn(tint a){return (a>0LL) - (a<0LL);}
4 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);}
5 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);}
6 bool inter(seg a, seg b){
7     tint bf = sgn(pc(a.f, a.s, b.f));
8     tint bs = sgn(pc(a.f, a.s, b.s));
9     tint af = sgn(pc(b.f, b.s, a.f));
10    tint as = sgn(pc(b.f, b.s, a.s));
11    if(bf*bs<0 && af*as<0) return true; //cruza sin tocar
12    if((bf==0 && pe(a.f,a.s,b.f) <= 0) || (bs==0 && pe(a.f,a.s,b.s) <= 0))return
        true; //b tiene un vertice en a
13    if((af==0 && pe(b.f,b.s,a.f) <= 0) || (as==0 && pe(b.f,b.s,a.s) <= 0))return
        true; //a tiene un vertice en b
14    return false;
15 }

```

3.9. Distancia entre segmentos

```

1 tdbl dist(pto p, seg s){
2     tdbl a = fabs(tdbl(pc(s.f, s.s, p)));
3     tdbl b = hypot(s.f.x-s.s.x,s.f.y-s.s.y),h=a/b, c = hypot(b, h);
4     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);
5     if(b<1e-10 || c <= d1 || c <= d2)return min(d1, d2); else return h;
6 }
7 tdbl dist(seg a, seg b){
8     return (inter(a, b))?0.0:min(min(dist(a.f, b), dist(a.s, b)), min(dist(b.f, a)
9     , dist(b.s, a)));

```

3.10. Cuentitas

Proyección de v sobre $\lambda w + z$:

$$\frac{\langle v-z, w \rangle w}{\|w\|^2} + z.$$

```

1  usa: cmath, algorithm, tipo
2  struct pto{tipo x,y;};
3  struct lin{tipo a,b,c;};
4  struct circ{pto c; tipo r;};
5  #define sqr(a)((a)*(a))
6  const double PI = (2.0 * acos(0.0));
7  pto punto(tipo x, tipo y){pto r;r.x=x;r.y=y;return r;}
8  const pto cero = punto(0,0);
9  pto suma(pto o, pto s, tipo k){
10     return punto(o.x + s.x * k, o.y + s.y * k);
11 }
12 pto sim(pto p, pto c){return suma(c, suma(p,c,-1), -1);}
13 pto ptoMedio(pto a, pto b){return punto((a.x+b.x)/2.0,(a.y+b.y)/2.0);}
14 tipo pc(pto a, pto b, pto o){
15     return (b.y-o.y)*(a.x-o.x)-(a.y-o.y)*(b.x-o.x);
16 }
17 tipo pe(pto a, pto b, pto o){
18     return (b.x-o.x)*(a.x-o.x)+(b.y-o.y)*(a.y-o.y);
19 }
20 #define sqrd(a,b) (sqr(a.x-b.x)+sqr(a.y-b.y))
21 tipo dist(pto a, pto b){return sqrt(sqrd(a,b));}
22 //define feq(a,b) (fabs((a)-(b))<0.000000000001) para interseccion
23 #define feq(a,b) (fabs((a)-(b))<0.0000000001)
24 tipo zero(tipo t){return feq(t,0.0)?0.0:t;}
25 bool alin(pto a, pto b, pto c){ return feq(0, pc(a,b,c));}
26 bool perp(pto a1, pto a2, pto b1, pto b2){
27     return feq(0, pe(suma(a1, a2, -1.0), suma(b1, b2, -1.0), cero));
28 }
29 bool hayEL(tipo A11, tipo A12, tipo A21, tipo A22){
30     return !feq(0.0, A22*A11-A12*A21);
31 }
32 pto ecLineal(tipo A11, tipo A12, tipo A21, tipo A22, tipo R1, tipo R2){
33     tipo det = A22*A11-A12*A21;
34     return punto((A22*R1-A12*R2)/det,(A11*R2-A21*R1)/det);
35 }
36 lin linea(pto p1, pto p2){
37     lin l;
38     l.b = p2.x-p1.x;
39     l.a = p1.y-p2.y;
40     l.c = p1.x*l.a + p1.y*l.b;
41     return l;
42 }
43 bool estaPL(pto p, lin l){return feq(p.x * l.a + p.y * l.b, l.c);}
44 bool estaPS(pto p, pto a, pto b){
45     return feq(dist(p,a)+dist(p,b),dist(b,a));
46 }
47 lin bisec(pto o, pto a, pto b){
48     tipo da = dist(a,o);
49     return linea(o, suma(a, suma(b,a,-1.0), da / (da+dist(b,o))));
50 }

```

```

51 bool paral(lin l1, lin l2){return !hayEL(l1.a, l1.b, l2.a, l2.b);}
52 bool hayILL(lin l1, lin l2){ //!paralelas || misma
53     return !paral(l1,l2)|| !hayEL(l1.a, l1.c, l2.a, l2.c);
54 }
55 pto interLL(lin l1, lin l2){//li==l2->pincha
56     return ecLineal(l1.a, l1.b, l2.a, l2.b, l1.c, l2.c);
57 }
58 bool hayILS(lin l, pto b1, pto b2){
59     lin b = linea(b1,b2);
60     if(!hayILL(l,b))return false;
61     if(estaPL(b1,l))return true;
62     return estaPS(interLL(l,b), b1,b2);
63 }
64 pto interLS(lin l, pto b1, pto b2){
65     return interLL(l, linea(b1, b2));
66 }
67 pto interSS(pto a1, pto a2, pto b1, pto b2){
68     return interLS(linea(a1, a2), b1, b2);
69 }
70 bool hayISS(pto a1, pto a2, pto b1, pto b2){
71     if (estaPS(a1,b1,b2)||estaPS(a2,b1,b2)) return true;
72     if (estaPS(b1,a1,a2)||estaPS(b2,a1,a2)) return true;
73     lin a = linea(a1,a2), b = linea(b1, b2);
74     if(!hayILL(a,b))return false;
75     if(paral(a,b))return false;
76     pto i = interLL(a,b);
77     //sale(i);sale(a1);sale(a2);sale(b1);sale(b2);cout << endl;
78     return estaPS(i,a1, a2) && estaPS(i,b1,b2);
79 }
80 tipo distPL(pto p, lin l){
81     return fabs((l.a * p.x + l.b * p.y - l.c)/sqrt(sqr(l.a)+sqr(l.b)));
82 }
83 tipo distPS(pto p, pto a1, pto a2){
84     tipo aa = sqrd(a1, a2);
85     tipo d = distPL(p, linea(a1, a2));
86     tipo xx = aa+sqr(d);
87     tipo a1a1 = sqrd(a1, p);
88     tipo a2a2 = sqrd(a2, p);
89     if(max(a1a1, a2a2) > xx){
90         return sqrt(min(a1a1, a2a2));
91     }else{
92         return d;
93     }
94 }
95 //
96 pto bariCentro(pto a, pto b, pto c){
97     return punto(
98         (a.x + b.x + c.x) / 3.0,
99         (a.y + b.y + c.y) / 3.0);
100 }

```

```

101 pto circunCentro(pto a, pto b, pto c){
102     tipo A = 2.0 * (a.x-c.x); tipo B = 2.0 * (a.y-c.y);
103     tipo C = 2.0 * (b.x-c.x); tipo D = 2.0 * (b.y-c.y);
104     tipo R = sqrt(a.x)-sqrt(c.x)+sqrt(a.y)-sqrt(c.y);
105     tipo P = sqrt(b.x)-sqrt(c.x)+sqrt(b.y)-sqrt(c.y);
106     return ecLineal(A,B,C,D,R,P);
107 }
108 pto ortoCentro(pto a, pto b, pto c){
109     pto A = sim(a, ptoMedio(b,c));
110     pto B = sim(b, ptoMedio(a,c));
111     pto C = sim(c, ptoMedio(b,a));
112     return circunCentro(A,B,C);
113 }
114 pto inCentro(pto a, pto b, pto c){
115     return interLL(bisec(a, b, c), bisec(b, a, c));
116 }
117 pto rotar(pto p, pto o, tipo s, tipo c){
118     //gira cw un angulo de sin=s, cos=c
119     return punto(
120         o.x + (p.x - o.x) * c + (p.y - o.y) * s,
121         o.y + (p.x - o.x) * -s + (p.y - o.y) * c
122     );
123 }
124 bool hayEcCuad(tipo a, tipo b, tipo c){//a*x*x+b*x+c=0 tiene sol real?
125     if(feq(a,0.0))return false;
126     return zero((b*b-4.0*a*c)) >= 0.0;
127 }
128 pair<tipo, tipo> ecCuad(tipo a, tipo b, tipo c){//a*x*x+b*x+c=0
129     tipo dx = sqrt(zero(b*b-4.0*a*c));
130     return make_pair((-b + dx)/(2.0*a), (-b - dx)/(2.0*a));
131 }
132 bool adentroCC(circ g, circ c){//c adentro de g sin tocar?
133     return g.r > dist(g.c, c.c) + c.r || !feq(g.r, dist(g.c, c.c) + c.r);
134 }
135 bool hayICL(circ c, lin l){
136     if(feq(0,l.b)){
137         swap(l.a, l.b);
138         swap(c.c.x, c.c.y);
139     }
140     if(feq(0,l.b))return false;
141     return hayEcCuad(
142         sqrt(l.a)+sqrt(l.b),
143         2.0*l.a*l.b*c.c.y-2.0*(sqrt(l.b)*c.c.x+l.c*l.a),
144         sqrt(l.b)*(sqrt(c.c.x)+sqrt(c.c.y)-sqrt(c.r))+sqrt(l.c)-2.0*l.c*l.b*c.c.y
145     );
146 }
147 pair<pto, pto> interCL(circ c, lin l){
148     bool sw=false;
149     if(sw=feq(0,l.b)){
150         swap(l.a, l.b);

```

```

151         swap(c.c.x, c.c.y);
152     }
153     pair<tipo, tipo> rc = ecCuad(
154         sqrt(l.a)+sqrt(l.b),
155         2.0*l.a*l.b*c.c.y-2.0*(sqrt(l.b)*c.c.x+l.c*l.a),
156         sqrt(l.b)*(sqrt(c.c.x)+sqrt(c.c.y)-sqrt(c.r))+sqrt(l.c)-2.0*l.c*l.b*c.c.y
157     );
158     pair<pto, pto> p(
159         punto(rc.first, (l.c - l.a * rc.first) / l.b),
160         punto(rc.second, (l.c - l.a * rc.second) / l.b)
161     );
162     if(sw){
163         swap(p.first.x, p.first.y);
164         swap(p.second.x, p.second.y);
165     }
166     return p;
167 }
168 bool hayICC(circ c1, circ c2){
169     lin l;
170     l.a = c1.c.x-c2.c.x;
171     l.b = c1.c.y-c2.c.y;
172     l.c = (sqrt(c2.r)-sqrt(c1.r)+sqrt(c1.c.x)-sqrt(c2.c.x)+sqrt(c1.c.y)
173         -sqrt(c2.c.y))/2.0;
174     return hayICL(c1, l);
175 }
176 }
177 pair<pto, pto> interCC(circ c1, circ c2){
178     lin l;
179     l.a = c1.c.x-c2.c.x;
180     l.b = c1.c.y-c2.c.y;
181     l.c = (sqrt(c2.r)-sqrt(c1.r)+sqrt(c1.c.x)-sqrt(c2.c.x)+sqrt(c1.c.y)
182         -sqrt(c2.c.y))/2.0;
183     return interCL(c1, l);
184 }

```

4. Grafos

4.1. Kruskal & Union-Find

```

1  usa: vector, utility, forn
2  typedef pair< tint, pair<int,int> > eje;
3  int n; vector<eje> ejes; //grafo n=cant nodos
4  #define MAXN 100000
5  int _cl[MAXN]; //empieza con todos en -1
6  int cl(int i) { return (_cl[i] == -1 ? i : _cl[i] = cl(_cl[i])); }
7  void join(int i, int j) { if(cl(i)!=cl(j)) _cl[cl(i)] = cl(j); }
8  tint krus() {
9      if (n==1) return 0;

```



```

10 sort(ejes.begin(), ejes.end());
11 int u = 0; tint t = 0;
12 memset(_cl,-1,sizeof(_cl));
13 forn(i,ejes.size()) {
14     eje& e = ejes[i];
15     if (cl(e.second.first) != cl(e.second.second)) {
16         u++; t += e.first; if(u==n-1) return t;
17         join(e.second.first, e.second.second);
18     }
19 } return -1; // -1 es que no es conexo
20 }

```

4.2. Edmond-Karp

```

1  usa: map,algorithm,queue
2  struct Eje{ long f,m; long d(){return m-f;}};
3  typedef map <int, Eje> MIE; MIE red[MAX_N];
4  int N,F,D;
5  void iniG(int n, int f, int d){N=n; F=f; D=d;fill(red, red+N, MIE());}
6  void aEje(int d, int h, int m){
7      red[d][h].m=m;red[d][h].f=red[h][d].m=red[h][d].f=0;
8  }
9  #define DIF_F(i,j) (red[i][j].d())
10 #define DIF_FI(i) (i->second.d())
11 int v[MAX_N];
12 long camAu(){
13     fill(v, v+N,-1);
14     queue<int> c;
15     c.push(F);
16     while(!c.empty() && v[D]==-1){
17         int n = c.front(); c.pop();
18         for(MIE::iterator i = red[n].begin(); i!=red[n].end(); i++){
19             if(v[i->first]==-1 && DIF_FI(i) > 0){
20                 v[i->first]=n;
21                 c.push(i->first);
22             }
23         }
24     }
25     if(v[D]==-1)return 0;
26     int n = D;
27     long f = DIF_F(v[n], n);
28     while(n!=F){
29         f<?DIF_F(v[n], n);
30         n=v[n];
31     }
32     n = D;
33     while(n!=F){
34         red[n][v[n]].f=- (red[v[n]][n].f+f);
35         n=v[n];
36     }

```

```

37     return f;
38 }
39 long flujo(){long tot=0, c;do{tot+=(c=camAu());}while(c>0); return tot;}

```

4.3. Flujo de costo mínimo

```

1  #define MAXN 100
2  const int INF = 1<<30;
3  struct Eje{
4      int f, m, p;
5      int d(){return m-f;}
6  };
7  Eje red[MAXN][MAXN];
8  int adyc[MAXN], ady[MAXN][MAXN];
9  int N,F,D;
10 void iniG(int n, int f, int d){ // n, fuente, destino
11     N=n;F=f;D=d;
12     fill(red[0], red[N], (Eje){0,0,0});
13     fill(ady, ady+N, 0);
14 }
15 void aEje(int d, int h, int m, int p){
16     red[h][d].p = -(red[d][h].p = p);
17     red[d][h].m = m; //poner [h][d] en m tambien para hacer eje bidireccional
18     ady[d][ady[d]++] = h; ady[h][ady[h]++] = d;
19 }
20 int md[MAXN], vd[MAXN];
21 int camAu(int &v){
22     fill(vd, vd+N, -1);
23     vd[F]=F; md[F]=0;
24     forn(rep, N)forn(i, N)if(vd[i]!=-1)forn(jj, ady[i]){
25         int j = ady[i][jj], nd = md[i]+red[i][j].p;
26         if(red[i][j].d()>0)if(vd[j]==-1 || md[j] > nd)md[j]=nd,vd[j]=i;
27     }
28     v=0;
29     if(vd[D]==-1)return 0;
30     int f = INF;
31     for(int n=D;n!=F;n=vd[n]) f <?= red[vd[n]][n].d();
32     for(int n=D;n!=F;n=vd[n]){
33         red[n][vd[n]].f=- (red[vd[n]][n].f+f);
34         v += red[vd[n]][n].p * f;
35     }
36     return f;
37 }
38 int flujo(int &r){ // r = costo, return = flujo
39     r=0; int v,f=0, c;
40     while((c = camAu(v)))r += v,f += c;
41     return f;
42 }

```

4.4. Matching perfecto de costo máximo - Hungarian $O(N^3)$

```

1  #define MAXN 256
2  #define INFTO 0x7f7f7f7f
3  int n;
4  int mt[MAXN][MAXN]; // Matriz de costos (X * Y)
5  int xy[MAXN], yx[MAXN]; // Matching resultante (X->Y, Y->X)
6
7  int lx[MAXN], ly[MAXN], slk[MAXN], slkx[MAXN], prv[MAXN];
8  char S[MAXN], T[MAXN];
9  void updtree(int x) {
10     forn(y, n) if (lx[x] + ly[y] - mt[x][y] < slk[y]) {
11         slk[y] = lx[x] + ly[y] - mt[x][y];
12         slkx[y] = x;
13     } }
14 int hungar() {
15     forn(i, n) {
16         ly[i] = 0;
17         lx[i] = *max_element(mt[i], mt[i]+n);
18     }
19     memset(xy, -1, sizeof(xy));
20     memset(yx, -1, sizeof(yx));
21     forn(m, n) {
22         memset(S, 0, sizeof(S));
23         memset(T, 0, sizeof(T));
24         memset(prv, -1, sizeof(prv));
25         memset(slk, 0x7f, sizeof(slk));
26         queue<int> q;
27         #define bpone(e, p) { q.push(e); prv[e] = p; S[e] = 1; updtree(e); }
28         forn(i, n) if (xy[i] == -1) { bpone(i, -2); break; }
29         int x=0, y=-1;
30         while (y== -1) {
31             while (!q.empty() && y== -1) {
32                 x = q.front(); q.pop();
33                 forn(j, n) if (mt[x][j] == lx[x] + ly[j] && !T[j]) {
34                     if (yx[j] == -1) { y = j; break; }
35                     T[j] = 1;
36                     bpone(yx[j], x);
37                 }
38             }
39             if (y!= -1) break;
40             int dlt = INFTO;
41             forn(j, n) if (!T[j]) dlt = min(dlt, slk[j]);
42             forn(k, n) {
43                 if (S[k]) lx[k] -= dlt;
44                 if (T[k]) ly[k] += dlt;
45                 if (!T[k]) slk[k] -= dlt;
46             }
47             forn(j, n) if (!T[j] && !slk[j]) {
48                 if (yx[j] == -1) {
49                     x = slkx[j]; y = j; break;
50                 } else {

```

```

51         T[j] = 1;
52         if (!S[yx[j]]) bpone(yx[j], slkx[j]);
53     }
54 }
55 }
56 if (y!= -1) {
57     for(int p = x; p != -2; p = prv[p]) {
58         yx[p] = y;
59         int ty = xy[p]; xy[p] = y; y = ty;
60     }
61 } else break;
62 }
63 int res = 0;
64 forn(i, n) res += mt[i][xy[i]];
65 return res;
66 }

```

4.5. Camino/Circuito Euleriano

```

1  usa: algorithm, vector, list, forn
2  typedef string ejeVal;
3  #define MENORATODOS ""
4  typedef pair<ejeVal, tint> eje;
5  tint n; vector<eje> ady[MAXN]; tint g[MAXN];
6  //grafo (inG = in grado o grado si es no dir)
7  tint aux[MAXN];
8  tint pinta(tint f) {
9     if (aux[f]) return 0;
10    tint r = 1; aux[f] = 1;
11    forn(i, ady[f].size()) r+=pinta(ady[f][i].second);
12    return r;
13 }
14 tint compCon() { fill(aux, aux+n, 0); tint r=0; forn(i, n) if (!aux[i]) { r++;
15     pinta(r); } return r; }
16 bool isEuler(bool path, bool dir) {
17     if (compCon() > 1) return false; tint c = (path ? 2 : 0);
18     forn(i, n) if (!dir ? ady[i].size() % 2 : g[i] != 0) {
19         if (dir && abs(g[i]) > 1) return false;
20         c--; if(c<0) return false; }
21     return true;
22 }
23 bool findCycle(tint f, tint t, list<tint>& r) {
24     if (aux[f] >= ady[f].size()) return false;
25     tint va = ady[f][aux[f]++].second;
26     r.push_back(va);
27     return (va != t ? findCycle(va, t, r) : true);
28 }
29 list<tint> findEuler(bool path) { //always directed, no repeated values
30     if (!isEuler(path, true)) return list<tint>();
31     bool agrego = false;

```

```

31 if (path) {
32     tint i = max_element(g, g + n)-g;
33     tint j = min_element(g, g + n)-g;
34     if (g[i] != 0) { ady[i].push_back( eje(MENORATODOS, j) ); agrego = true; }
35 }
36 tint x = -1;
37 forn(i,n) {sort(ady[i].begin(), ady[i].end()); if (x<0 || ady[i][0] < ady[x
    ][0]) x=i;}
38 fill(aux, aux+n, 0);
39 list<tint> r; findCycle(x,x,r); if (!agrego) r.push_front(r.back());
40 list<tint> aux; bool find=false;
41 list<tint>::iterator it = r.end();
42 do{ if (!find) --it;
43     for(find=findCycle(*it, *it, aux);!aux.empty();aux.pop_front()) it = r.
        insert(++it, aux.front());
44 } while (it != r.begin());
45 return r;
46 }

```

4.6. Erdős-Gallai

```

1 includes: algorithm, functional, numeric, forn
2 tint n; tint d[MAXL]; //grafo
3 tint sd[MAXL]; //auxiliar
4 bool graphical() {
5     if (accumulate(d, d+n, 0) % 2 == 1) return false;
6     sort(d, d+n, greater<tint>()); copy(d, d+n, sd);
7     forn(i,n) sd[i+1] += sd[i];
8     forn(i,n) {
9         if (d[i] < 0) return false;
10        tint j = lower_bound(d+i+1, d+n, i+1, greater<tint>()) - d;
11        if (sd[i] > i*(i+1) + sd[n-1] - sd[j-1] + (j-i-1)*(i+1))
12            return false;
13    } return true;
14 }

```

4.7. Grafo cactus

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

```

1 struct eje { int t,i; };
2 typedef vector<eje> cycle;
3 int n,m,us[MAXM],pa[MAXN],epa[MAXN],tr[MAXM];
4 vector<eje> ady[MAXN];
5 void iniG(int nn) { n=nn; m=0; fill(ady,ady+n,vector<eje>());
6 fill(pa,pa+n,-1); }
7 //f:from t:to d:0 si no es dirigido y 1 si es dirigido
8 void addE(int f, int t, int d) {
9     ady[f].push_back((eje){t,m});
10    if (!d) ady[t].push_back((eje){f,m}), tr[m]=0;
11    us[m++] = 0;

```

```

12 }
13 //devuelve false si algun eje esta en mas de un ciclo
14 bool cycles(vector<cycle>& vr,int f=0,int a=-2,int ai=-2) {
15     int t; pa[f]=a; epa[f]=ai;
16     forn(i,ady[f].size()) if (!tr[ady[f][i].i]++) if (pa[t=ady[f][i].t] != -1) {
17         cycle c(1,ady[f][i]); int ef=f;
18         do {
19             if (!ef) return 0;
20             eje e=ady[pa[ef]][epa[ef]];
21             if (us[e.i]++) return 0;
22             c.push_back(e);
23         } while ((ef=pa[ef])!=t);
24         vr.push_back(c);
25     } else if (!cycles(vr,t,f,i)) return 0;
26     return 1;
27 };

```

5. Matemática

5.1. Algoritmos de cuentas

5.1.1. MCD

```

1 tint mcd(tint a, tint b){ return (a==0)?b:mcd(b%a, a);}
2 struct dxy {tint d,x,y;};
3 dxy mcde(tint a, tint b) {
4     dxy r, t;
5     if (b == 0) {
6         r.d = a; r.x = 1; r.y = 0;
7     } else {
8         t = mcde(b,a%b);
9         r.d = t.d; r.x = t.y;
10        r.y = t.x - a/b*t.y;
11    }
12    return r;
13 }

```

5.1.2. Teorema Chino del Resto

```

1 usa: mcde
2 #define modq(x) (((x)%q+q)%q)
3 tint tcr(tint* r, tint* m, int n) { // x ≡ r_i (m_i) i ∈ [0..n)
4     tint p=0, q=1;
5     forn(i, n) {
6         p = modq(p-r[i]);
7         dxy w = mcde(m[i], q);
8         if (p%w.d) return -1; // sistema incompatible
9         q = q / w.d * m[i];
10        p = modq(r[i] + m[i] * p / w.d * w.x);

```

```

11 }
12 return p; // x \equiv p (q)
13 }

```

5.1.3. Longitud de los números de 1 a N

```

1 tint sumDig(tint n, tint m){ // resultado modulo m
2     tint b=10, d=1, r=0;
3     while(b<=n){
4         r = (r + (b-b/10LL)*(d++))%m;
5         b*=10LL;
6     }
7     return (r + (n-b/10LL+1LL)*d)%m;
8 }

```

5.2. Teoremas y propiedades

5.2.1. Ecuación de grafo planar

$regiones = ejes - nodos + componentes$ $Conexas + 1$

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: $(m+in)^2 = a+ib$

5.2.3. Teorema de Pick

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

5.2.4. Propiedades varias

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

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

$$\sum_{i=1}^n \binom{n-1}{i-1} = 2^{n-1} ; \sum_{i=1}^n i \cdot \binom{n-1}{i-1} = n \cdot 2^{n-1}$$

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 (σ_0) para *algunos* $n/\neg \exists n' < n, \sigma_0(n') \geq \sigma_0(n)$
 $\sigma_0(60) = 12$; $\sigma_0(120) = 16$; $\sigma_0(180) = 18$; $\sigma_0(240) = 20$; $\sigma_0(360) = 24$
 $\sigma_0(720) = 30$; $\sigma_0(840) = 32$; $\sigma_0(1260) = 36$; $\sigma_0(1680) = 40$; $\sigma_0(10080) = 72$
 $\sigma_0(15120) = 80$; $\sigma_0(50400) = 108$; $\sigma_0(83160) = 128$; $\sigma_0(110880) = 144$
 $\sigma_0(498960) = 200$; $\sigma_0(554400) = 216$; $\sigma_0(1081080) = 256$; $\sigma_0(1441440) = 288$
 $\sigma_0(4324320) = 384$; $\sigma_0(8648640) = 448$

Suma de divisores (σ_1) para *algunos* $n/\neg \exists n' < n, \sigma_1(n') \geq \sigma_1(n)$
 $\sigma_1(96) = 252$; $\sigma_1(108) = 280$; $\sigma_1(120) = 360$; $\sigma_1(144) = 403$; $\sigma_1(168) = 480$
 $\sigma_1(960) = 3048$; $\sigma_1(1008) = 3224$; $\sigma_1(1080) = 3600$; $\sigma_1(1200) = 3844$
 $\sigma_1(4620) = 16128$; $\sigma_1(4680) = 16380$; $\sigma_1(5040) = 19344$; $\sigma_1(5760) = 19890$
 $\sigma_1(8820) = 31122$; $\sigma_1(9240) = 34560$; $\sigma_1(10080) = 39312$; $\sigma_1(10920) = 40320$
 $\sigma_1(32760) = 131040$; $\sigma_1(35280) = 137826$; $\sigma_1(36960) = 145152$; $\sigma_1(37800) = 148800$
 $\sigma_1(60480) = 243840$; $\sigma_1(64680) = 246240$; $\sigma_1(65520) = 270816$; $\sigma_1(70560) = 280098$
 $\sigma_1(95760) = 386880$; $\sigma_1(98280) = 403200$; $\sigma_1(100800) = 409448$
 $\sigma_1(491400) = 2083200$; $\sigma_1(498960) = 2160576$; $\sigma_1(514080) = 2177280$
 $\sigma_1(982800) = 4305280$; $\sigma_1(997920) = 4390848$; $\sigma_1(1048320) = 4464096$
 $\sigma_1(4979520) = 22189440$; $\sigma_1(4989600) = 22686048$; $\sigma_1(5045040) = 23154768$
 $\sigma_1(9896040) = 44323200$; $\sigma_1(9959040) = 44553600$; $\sigma_1(9979200) = 45732192$

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

```

1 typedef vector<tipo> Vec;
2 typedef vector<Vec> Mat;
3 #define eps 1e-10
4 #define feq(a, b) (fabs(a-b)<eps)
5 bool resolver_ev(Mat a, Vec y, Vec &x, Mat &ev){
6     int n = a.size(), m = n?a[0].size():0, rw = min(n, m);
7     vector<int> p; forn(i,m) p.push_back(i);
8     forn(i, rw){
9         int uc=i, uf=i;
10        // aca pivotea. lo unico importante es que a[i][i] sea no nulo
11        forsn(f, i, n) forsn(c, i, m) if(fabs(a[f][c])>fabs(a[uf][uc])) {uf=f;uc=c;}
12        if (feq(a[uf][uc], 0)) { rw = i; break; }
13        forn(j, n) swap(a[j][i], a[j][uc]);
14        swap(a[i], a[uf]); swap(y[i], y[uf]); swap(p[i], p[uc]);
15        // fin pivoteo
16        tipo inv = 1 / a[i][i]; //aca divide
17        forsn(j, i+1, n) {
18            tipo v = a[j][i] * inv;
19            forsn(k, i, m) a[j][k]-=v * a[i][k];
20            y[j] -= v*y[i];
21        }
22    } // rw = rango(a), aca la matriz esta triangulada
23    forsn(i, rw, n) if (!feq(y[i],0)) return false; // chequeo de compatibilidad
24    x = vector<tipo>(m, 0);
25    dforn(i, rw){
26        tipo s = y[i];
27        forsn(j, i+1, rw) s -= a[i][j]*x[p[j]];
28        x[p[i]] = s / a[i][i]; //aca divide
29    }
30    ev = Mat(m-rw, Vec(m, 0)); // Esta parte va SOLO si se necesita el ev
31    forn(k, m-rw) {

```

```

32        ev[k][p[k+rw]] = 1;
33        dforn(i, rw){
34            tipo s = -a[i][k+rw];
35            forsn(j, i+1, rw) s -= a[i][j]*ev[k][p[j]];
36            ev[k][p[i]] = s / a[i][i]; //aca divide
37        }
38    }
39    return true;
40 }
41
42 bool diagonalizar(Mat &a){
43     // PRE: a.cols > a.filas
44     // PRE: las primeras (a.filas) columnas de a son l.i.
45     int n = a.size(), m = a[0].size();
46     forn(i, n){
47         int uf = i;
48         forsn(k, i, n) if (fabs(a[k][i]) > fabs(a[uf][i])) uf = k;
49         if (feq(a[uf][i], 0)) return false;
50         swap(a[i], a[uf]);
51         tipo inv = 1 / a[i][i]; // aca divide
52         forn(j, n) if (j != i) {
53             tipo v = a[j][i] * inv;
54             forsn(k, i, m) a[j][k] -= v * a[i][k];
55         }
56         forsn(k, i, m) a[i][k] *= inv;
57     }
58     return true;
59 }

```

5.5. Programación Lineal - Simplex

Teorema de dualidad (fuerte): Dado un problema lineal Π_1 : minimizar $c^t \cdot X$, sujeto a $A \cdot X \leq b, X \geq 0$ se define el problema lineal *dual standard* Π_2 como: minimizar $-b^t \cdot Y$, sujeto a $A^t \cdot Y \geq c$. Si Π_1 es satisfacible entonces Π_2 es satisfacible y $c^t \cdot X = b^t \cdot Y$. Si Π_1 es insatisfacible o no acotado entonces Π_2 es insatisfacible o no acotado (Obs: no pueden ser ambos no acotados).

Dados `cfun`, `rmat` y `bvec`; Minimiza $cfun^t \cdot xvar$ sujeto a las condiciones $rmat \cdot xvar \leq bvec$. 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`.

```

1 usa: resolver,
2
3 #define MAXVAR 64
4 #define MAXRES 128
5 tipo rmat[MAXRES][MAXVAR+MAXRES*2];
6 tipo bvec[MAXRES];
7 tipo cfun[MAXVAR+MAXRES*2];

```

```

8  tipo xvar[MAXVAR];
9
10 #define HAYSOL 0
11 #define NOSOL -1
12 #define NOCOTA -2
13
14 int simplex(int m, int n) { // cant restric; cant vars
15     int base[MAXVAR+MAXRES], esab[MAXVAR+MAXRES];
16     int nn = n+m; // Variables (originales) + holgura
17     tipo res = 0;
18
19     for(i, m) for(j, m) rmat[i][n+j] = (i==j);
20     for(i, m) cfun[n+i] = 0;
21
22     for(i, n) esab[i] = -1;
23     for(i, m) { base[i] = n+i; esab[n+i] = i; }
24
25     // Agrega las artificiales; si todos los bvec[] son positivos se puede omitir
26     // esto
27     int arts[MAXRES];
28     int bmin = 0;
29     for(i, m) if (bvec[i] < bvec[bmin]) bmin = i;
30     int art = bvec[bmin] < -eps;
31     for(i, m) arts[i] = 2*(bvec[i] >= -eps) - 1;
32     if (art) {
33         for(i, m) rmat[i][nn] = -(bvec[i] < -eps);
34         esab[n+bmin] = -1; esab[nn] = bmin; base[bmin] = nn;
35         nn++;
36     }
37
38     Mat B(m, Vec(m, 0));
39     Vec y(m, 0), c(m, 0), d(m, 0);
40     int j0 = 0;
41     do {
42         for(i, m) for(j, m) B[i][j] = arts[j] * rmat[j][base[i]];
43         for(i, m) c[i] = art?base[i]>=m+n:cfun[base[i]];
44         resolver(B, c, y);
45
46         for(; j0 < nn; ++j0) if (esab[j0] == -1) {
47             res = art?j0>=m+n:cfun[j0];
48             for(i, m) res -= y[i] * arts[i] * rmat[i][j0];
49             if (j0 < m+n && res < epsval) break;
50         }
51
52         for(i, m) for(j, m) B[i][j] = rmat[i][base[j]];
53         for(i, m) c[i] = rmat[i][j0];
54         resolver(B, c, d);
55         for(i, m) c[i] = bvec[i];
56         resolver(B, c, y);

```

```

57     if (j0 == nn) if (art) {
58         if (esab[m+n] != -1 && y[esab[m+n]] > epsval) return NOSOL;
59         for(i = m+n-1; i >= 0; i--) if (esab[i] == -1) { esab[i] = esab[m+n];
60             base[esab[i]] = i; break; }
61         art = 0; nn = m+n; j0 = 0; continue;
62     } else break; // Optimo
63
64     bool bl = true;
65     for(i, m) bl = bl && (d[i] <= eps);
66
67     if (bl) return NOCOTA; // Problema no acotado
68
69     int j1 = 0;
70     for(i, m) if (d[i] > 0) {
71         tipo mlt = y[i] / d[i];
72         if (!bl || (feq(mlt, res) && (base[i] < j1)) || (mlt < res)) {
73             res = mlt;
74             j1 = base[i];
75             bl = true;
76         }
77     }
78     if (res < eps && ++j0) continue;
79     if (art && j1 == m+n) nn--, art--;
80
81     int w = esab[j1]; // variable de salida
82     base[w] = j0; // Entra j0
83     esab[j0] = w;
84     esab[j1] = -1; // j1 es no basica ahora.
85     j0 = 0;
86     while(1);
87
88     for(i, m) for(j, m) B[i][j] = rmat[i][base[j]];
89     for(i, m) c[i] = bvec[i];
90     resolver(B, c, y);
91
92     for(i, n) xvar[i] = (esab[i] == -1)?0:y[esab[i]];
93
94     return HAYSOL;
95 }

```

5.6. Factorización QR de Householder

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

```

1  typedef vector<vector<tipo>> > Mat;
2  typedef vector<tipo> Vec;
3  tipo sqr(tipo x) {return x*x;}
4
5  void show(Mat &a);
6

```

```

7 void qr(const Mat &a, Mat &q, Mat &r) {
8     int n = a.size();
9     r = a;
10    q = Mat(n, Vec(n, 0));
11    forn(i, n) forn(j, n) q[i][j] = (i==j);
12
13    forn(k, n-1) {
14        tipo beta = 0;
15        forsn(i, k, n) beta += sqr(r[i][k]);
16        tipo alph = sqrt(beta);
17        if (alph * r[k][k] >= 0) alph = -alph;
18
19        Vec v(n, 0);
20        forsn(i, k, n) v[i] = r[i][k]; v[k] -= alph;
21        beta += sqr(v[k]) - sqr(r[k][k]);
22
23        #define QRmult(X) \
24        forn(i, n) { tipo w = 0; \
25            forsn(j, k, n) w += X * v[j]; w /= beta/2; \
26            forsn(j, k, n) X -= w * v[j]; }
27
28        // Q := Q * (I - 2 v * v^t) = Q - 2 * ((Q * v) * v^t)
29        QRmult(q[i][j]);
30        // A := Qj * A; \equiv A^t := A^t * Qj;
31        QRmult(r[j][i]);
32
33        forsn(i, k+1, n) r[i][k] = 0;
34    }
35 }
36
37 // QR para calcular autvalores (no estoy seguro de para qu'e matrices sirve)
38 Mat operator* (const Mat &ml, const Mat &mr) {
39     int a = ml.size(), b = mr.size(), c = mr[0].size();
40     Mat res(a, Vec(c, 0));
41     forn(i, a) forn(j, c) forn(k, b) res[i][j] += ml[i][k] * mr[k][j];
42     return res;
43 }
44
45 #define iterac ???
46 void autoval(Mat &a) {
47     int n = a.size();
48     Mat q(n, Vec(n, 0));
49     forn(i, iterac) {
50         qr(a, q, a);
51         a = a * q;
52     }
53     // Los autovalores convergen en la diagonal de "a"
54 }

```

5.7. Multiplicación de Karatsuba

BASE y BASEXP deben ser tales que $BASE = 10^{BASEXP}$ y además, $BASE^2 \cdot largo$ 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.

```

1  #define BASE 1000000
2  #define BASEXP 6
3
4  typedef tint tipo; // o int
5
6  tipo* ini(int l){
7      tipo *r = new tipo[l];
8      fill(r, r+l, 0);
9      return r;
10 }
11 #define add(l,s,d,k)forn(i, l)(d)[i]+=(s)[i]*k
12 void mulFast(int l, tipo *n1, tipo *n2, tipo *nr){
13     if(l<=0)return;
14     if(l<35){
15         forn(i, l)forn(j, l)nr[i+j]+=n1[i]*n2[j];
16     }else{
17         int lac = l/2, lbd = l - (l/2);
18         tipo *a = n1, *b=n1+lac, *c=n2, *d=n2+lac;
19         tipo *ab = ini(lbd+1), *cd = ini(lbd+1);
20         tipo *ac = ini(lac+lac), *bd = ini(lbd+lbd);
21         add(lac, a, ab, 1);
22         add(lbd, b, ab, 1);
23         add(lac, c, cd, 1);
24         add(lbd, d, cd, 1);
25         mulFast(lac, a, c, ac);
26         mulFast(lbd, b, d, bd);
27         add(lac+lac, ac, nr+lac,-1);
28         add(lbd+lbd, bd, nr+lac,-1);
29         add(lac+lac, ac, nr,1);
30         add(lbd+lbd, bd, nr+lac+lac,1);
31         mulFast(lbd+1, ab, cd, nr+lac);
32         free(ab); free(cd); free(ac); free(bd);
33     }
34 }
35 void mulFast(int l1, tipo *n1, int l2, tipo *n2, int &lr, tipo *nr){
36     while(l1<l2) n1[l1++]=0;
37     while(l2<l1) n2[l2++]=0;
38     lr=l1+l2+3;
39     fill(nr, nr+lr, 0);
40     mulFast(l1, n1, n2, nr);
41
42     tipo r = 0;
43     forn(i, lr){
44         tipo q = r+nr[i];

```

```

45     nr[i] = q%BASE, r = q/BASE;
46 }
47 while(lr>1 && nr[lr-1]==0)lr--;
48 }
49
50 // Cosas extra (convierten entre base 10 y 10^n)
51 void base10ton(int &l, tipo* n) {
52     tipo p10[BASEXP]; p10[0] = 1;
53     forn(i, BASEXP-1) p10[i+1] = p10[i] * 10;
54
55     int nl = (1+BASEXP-1)/BASEXP;
56     forsn(i, 1, nl*BASEXP) n[i] = 0;
57     forn(i, nl) {
58         tint s = 0;
59         forn(j, BASEXP) s+= n[i*BASEXP+j]*p10[j];
60         n[i] = s;
61     }
62     l = nl;
63 }
64
65 void baseNto10(int &l, tipo* n) {
66     for(int i = l-1; i>=0; --i) {
67         tipo v = n[i];
68         forn(j, BASEXP) {
69             n[i*BASEXP+j] = v % 10; v /= 10;
70         }
71     }
72     l = l*BASEXP;
73     while (!n[l-1] && l > 1) l--;
74 }

```

5.8. Long - Entero largo

```

1  typedef tint tipo;
2  #define BASEXP 6
3  #define BASE 1000000
4  #define LMAX 1000
5
6  struct Long {
7      int l;
8      tipo n[LMAX];
9      Long(tipo x) { l = 0; forn(i, LMAX) { n[i]=x%BASE; l+=!!x||!i; x/=BASE;} }
10     Long(){*this = Long(0);}
11     Long(string x) {
12         l=(x.size()-1)/BASEXP+1;
13         fill(n, n+LMAX, 0);
14         tipo r=1;
15         forn(i,x.size()){
16             n[i / BASEXP] += r * (x[x.size()-1-i]-'0');

```

```

17         r*=10; if(r==BASE)r=1;
18     }
19 }
20 };
21
22 void out(Long& a) {
23     char msg[BASEXP+1];
24     cout << a.n[a.l-1];
25     dform(i,a.l-1) {
26         sprintf(msg, "%6.6llu", a.n[i]); cout << msg; // 6 = BASEXP !
27     }
28     cout << endl;
29 }
30 void invar(Long &a) {
31     fill(a.n+a.l, a.n+LMAX, 0);
32     while(a.l>1 && !a.n[a.l-1]) a.l--;
33 }
34
35 void lsuma(const Long&a, const Long&b, Long&c) { // c = a + b
36     c.l = max(a.l, b.l);
37     tipo q = 0;
38     forn(i, c.l) q += a.n[i]+b.n[i], c.n[i]=q%BASE, q/=BASE;
39     if(q) c.n[c.l++] = q;
40     invar(c);
41 }
42 Long& operator+=( Long&a, const Long&b) { lsuma(a, b, a); return a; }
43 Long operator+ (const Long&a, const Long&b) { Long c; lsuma(a, b, c); return c; }
44
45 bool lresta(const Long&a, const Long&b, Long&c) { // c = a - b
46     c.l = max(a.l, b.l);
47     tipo q = 0;
48     forn(i, c.l) q += a.n[i]-b.n[i], c.n[i]=(q+BASE)%BASE, q=(q+BASE)/BASE-1;
49     invar(c);
50     return !q;
51 }
52 Long& operator-= (Long&a, const Long&b) { lresta(a, b, a); return a; }
53 Long operator- (const Long&a, const Long&b) { Long c; lresta(a, b, c); return c; }
54
55 bool operator< (const Long&a, const Long&b) { Long c; return !lresta(a, b, c); }
56 bool operator<= (const Long&a, const Long&b) { Long c; return lresta(b, a, c); }
57 bool operator== (const Long&a, const Long&b) { return a <= b && b <= a; }
58
59 void lmul(const Long&a, const Long&b, Long&c) { // c = a * b
60     c.l = a.l+b.l;
61     fill(c.n, c.n+b.l, 0);
62     forn(i, a.l) {
63         tipo q = 0;
64         forn(j, b.l) q += a.n[i]*b.n[j]+c.n[i+j], c.n[i+j] = q%BASE, q/=BASE;
65         c.n[i+b.l] = q;

```



```

66     }
67     invar(c);
68 }
69
70 Long& operator*= (Long&a, const Long&b) { Long c; lmul(a, b, c); return a=c; }
71 Long operator* (const Long&a, const Long&b) { Long c; lmul(a, b, c); return c; }
72
73 void lmul(const Long&a, tipo b, Long&c) { // c = a * b
74     tipo q = 0;
75     forn(i, a.l) q += a.n[i]*b, c.n[i] = q%BASE, q/=BASE;
76     c.l = a.l;
77     while(q) c.n[c.l++] = q%BASE, q/=BASE;
78     invar(c);
79 }
80
81 Long& operator*= (Long&a, tipo b) { lmul(a, b, a); return a; }
82 Long operator* (const Long&a, tipo b) { Long c = a; c*=b; return c; }
83
84 void ldiv(const Long& a, tipo b, Long& c, tipo& rm) { // c = a / b ; rm = a % b
85     rm = 0;
86     dforn(i, a.l) {
87         rm = rm * BASE + a.n[i];
88         c.n[i] = rm / b; rm %= b;
89     }
90     c.l = a.l;
91     invar(c);
92 }
93
94 Long operator/ (const Long&a, tipo b) { Long c; tipo r; ldiv(a, b, c, r);
95     return c; }
96
97 tipo operator% (const Long&a, tipo b) { Long c; tipo r; ldiv(a, b, c, r);
98     return r; }
99
100 void ldiv(const Long& a, const Long& b, Long& c, Long& rm) { // c = a / b ; rm =
101     a % b
102     rm = 0;
103     dforn(i, a.l) {
104         if (rm.l == 1 && rm.n[0] == 0)
105             rm.n[0] = a.n[i];
106         else {
107             dforn(j, rm.l) rm.n[j+1] = rm.n[j];
108             rm.n[0] = a.n[i]; rm.l++;
109         }
110         tipo q = rm.n[b.l] * BASE + rm.n[b.l-1];
111         tipo u = q / (b.n[b.l-1] + 1);
112         tipo v = q / b.n[b.l-1] + 1;
113         while (u < v-1) {
114             tipo m = (u+v)/2;
115             if (b*m <= rm) u = m; else v = m;
116         }
117     }

```

```

113     c.n[i] = u;
114     rm -= b*u;
115 }
116 c.l = a.l;
117 invar(c);
118 }
119
120 Long operator/ (const Long&a, const Long & b) { Long c,r; ldiv(a, b, c, r);
121     return c; }
122
123 Long operator% (const Long&a, const Long & b) { Long c,r; ldiv(a, b, c, r);
124     return r; }

```

5.9. Fracción

```

1  usa: algorithm, tint, mcd
2  struct frac {
3      tint p,q;
4      frac(tint num=0, tint den=1):p(num),q(den) { norm(); }
5      frac& operator+=(const frac& o){
6          tint a = mcd(q,o.q);
7          p=p*(o.q/a)+o.p*(q/a);
8          q*=(o.q/a);
9          norm();
10         return *this;
11     }
12     frac& operator-=(const frac& o){
13         tint a = mcd(q,o.q);
14         p=p*(o.q/a)-o.p*(q/a);
15         q*=(o.q/a);
16         norm();
17         return *this;
18     }
19     frac& operator*=(frac o){
20         tint a = mcd(q,o.p);
21         tint b = mcd(o.q,p);
22         p=(p/b)*(o.p/a);
23         q=(q/a)*(o.q/b);
24         return *this;
25     }
26     frac& operator/=(frac o){
27         tint a = mcd(q,o.q);
28         tint b = mcd(o.p,p);
29         p=(p/b)*(o.q/a);
30         q=(q/a)*(o.p/b);
31         norm();
32         return *this;
33     }
34
35     void norm(){
36         tint aux = mcd(p,q);

```

```

37     if (aux){ p/=aux; q/=aux; }
38     else { q=1; }
39     if (q<0) { q=-q; p=-p; }
40 }
41 };

```

6. Cosas

6.1. Morris-Prath

premp[$i + 1$] da el maximo k en $[0, i)$ tal que $s[0, k) = s[i - k, i)$

```

1  tint pmp[MAXL];
2  void preMp(string& x){
3      tint i=0, j = pmp[0] = -1;
4      while(i<(tint)x.size()){
5          while(j>-1 && x[i] != x[j]) j = pmp[j];
6          pmp[++i] = ++j;
7      }
8  }
9  void mp(string& b, string& g){
10     preMp(b);
11     tint i=0, j=0;
12     while(j<(tint)g.size()){
13         while(i>-1 && b[i] != g[j]){i = pmp[i];}
14         i++; j++;
15         if (i>=(tint)b.size()){
16             OUTPUT(j - i);
17             i=pmp[i];
18         }
19     }
20 }

```

6.2. SAT - 2

```

1  usa: stack
2  #define MAXN 1024
3  #define MAXEQ 1024000
4
5  int fch[2*MAXN], nch[2*MAXEQ], dst[2*MAXEQ], eqs; // Grafo
6  #define addeje(s,d) { nch[eqs]=fch[s]; dst[fch[s]=eqs++]=d; }
7  #define neg(X) (2*MAXN-1-(X))
8  void init() {
9      memset(fch, 0xff, sizeof(fch));
10     eqs=0;
11 }
12 void addEqu(int a, int b) {
13     addeje(neg(a), b);
14     addeje(neg(b), a);

```

```

15 }
16 char verdad[2*MAXN]; // Solo si interesa el valor de verdad
17 int us[2*MAXN], lw[2*MAXN], id[2*MAXN];
18 stack<int> q; int qv, cp;
19 void tjn(int i) {
20     lw[i] = us[i] = ++qv;
21     id[i]=-2; q.push(i);
22     for(int j = fch[i]; j!=-1; j=nch[j]) { int x = dst[j];
23         if (!us[x] || id[x] == -2) {
24             if (!us[x]) tjn(x);
25             lw[i] = min(lw[i], lw[x]);
26         }
27     }
28     if (lw[i] == us[i]) {
29         int x; do { x = q.top(); q.pop(); id[x]=cp; } while (x!=i);
30         verdad[cp] = (id[neg(i)] < 0); // Valor de verdad de variable i es
31         verdad[id[i]]
32         cp++;
33     }
34 void compCon(int n) { // Tarjan algorithm
35     memset(us, 0, sizeof(us));
36     memset(id, -1, sizeof(id));
37     q=stack<int>(); qv = cp = 0;
38     forn(i, n) {
39         if (!us[i]) tjn(i);
40         if (!us[neg(i)]) tjn(neg(i));
41     }
42 }
43 bool satisf(int n) {
44     compCon(n);
45     forn(i, n) if (id[i] == id[neg(i)]) return false;
46     return true;
47 }

```

6.3. Rotaciones del cubo

```

1  #define _ALTA {forn(h, 6) rot[p][h] = d[h]; p++;}
2  #define _DER forn(h, 6) d[h] = _der[d[h]];
3  #define _UP forn(h, 6) d[h] = _up[d[h]];
4
5  int rot[24][6];
6  const int _der[6] = {0, 2, 4, 1, 3, 5};
7  const int _up[6] = {1, 5, 2, 3, 0, 4};
8
9  void rotaciones() {
10     int d[6];
11     int p = 0;
12     forn(i, 6) d[i] = i;
13     forn(i, 2) {

```

4 --> / \ <-- 3

```

14   forn(j, 3) {
15       _ALTA; _DER;
16       _ALTA; _DER;
17       _ALTA; _DER;
18       _ALTA; _UP;
19   }
20   _DER; _UP; _UP;
21 }
22 return;
23 }

```

7. Extras

7.1. Orden total de puntos alrededor de un centro

```

1 struct Cmp{
2     pto r;
3     Cmp(pto _r){r = _r;}
4     int cuad(const pto &a) const{
5         if(a.x > 0 && a.y >= 0)return 0;
6         if(a.x <= 0 && a.y > 0)return 1;
7         if(a.x < 0 && a.y <= 0)return 2;
8         if(a.x >= 0 && a.y < 0)return 3;
9         assert(a.x ==0 && a.y==0);
10        return -1;
11    }
12    bool cmp(const pto&p1, const pto&p2)const{
13        int c1 = cuad(p1), c2 = cuad(p2);
14        if(c1==c2){
15            return p1.y*p2.x<p1.x*p2.y;
16        }else{
17            return c1 < c2;
18        }
19    }
20    bool operator()(const pto&p1, const pto&p2) const{
21        return cmp(pto(p1.x-r.x,p1.y-r.y),pto(p2.x-r.x,p2.y-r.y));
22    }
23 };

```

8. El AJI es una fruta

8.1. Dinitz

```

1 #define INF 1000000000 // Infinito de FLUJO
2 #define MAX_M 1000000 // Maximo de aristas
3 #define MAX_N 45000 // Maximo de nodos
4 int v[2*MAX_M], l[2*MAX_M]; // Vecino, link. link te tira el indice de la arista
5 // "al reves" asociada en la lista del vecino.
6 long c[2*MAX_M]; // Capacidad

```

```

6 int sz[MAX_N], po[MAX_N], r[MAX_N], n, S, T;
7 typedef map<int,long> Mii;
8 Mii CAP[MAX_N];
9 void iniG() {
10     n = 0;
11     memset(sz,0,sizeof(sz));
12     forn(i,MAX_N) CAP[i].clear();
13 }
14 void aEje(int d,int h,long cap) {
15     if (d == h) return; // Ignoramos completamente autoejes, obvio :D
16     n = max(n,max(d,h));
17     pair<Mii::iterator,bool> par = CAP[d].insert(make_pair(h,0));
18     if (par.second) {
19         CAP[h][d] = 0;
20         sz[d]++;
21         sz[h]++;
22     }
23     par.first->second += max(cap,(long)0);
24 }
25 void _aEje(int d,int h,long capDH, long capHD) {
26     #define ASIG(d,h,cap) {v[po[d]] = h; c[po[d]] = cap; l[po[d]] = po[h];}
27     ASIG(d,h,capDH);
28     ASIG(h,d,capHD);
29     po[d]++; po[h]++;
30 }
31 void _iniG() {
32     po[0] = 0;
33     forn(i,n-1) po[i+1] = po[i] + sz[i];
34     forn(u,n) forall(v,CAP[u])
35         if (u < v->first) _aEje(u,v->first,v->second,CAP[v->first][u]);
36 }
37 long aumentar() {
38     // bfs
39     forn(i,n) r[i] = -1;
40     r[S] = 0;
41     queue<int> q;
42     q.push(S);
43     while (!q.empty()) {
44         int x = q.front(); q.pop();
45         int d = r[x] + 1, b = po[x];
46         if (r[T] >= 0 && d > r[T]) break;
47         forsn(j,b,b+sz[x])
48             if (c[j]>0 && r[v[j]] < 0) {
49                 r[v[j]] = d;
50                 q.push(v[j]);
51             }
52     }
53     // dfs que hace la magia :P
54     long res = 0;
55     static int path[MAX_N]; path[0] = S;

```

```

56 static int p[MAX_N], ind[MAX_N];
57 memset(p, -1, sizeof(p));
58 int pp = 0; // Path pointer, es la longitud
59 while (pp >= 0) {
60     int x = path[pp];
61     if (x == T) {
62         long f = INF;
63         int pri = 0;
64         forn(i, pp) if (c[ind[i]] <= f) f = c[ind[i]], pri = i;
65         forn(i, pp) c[ind[i]] -= f, c[l[ind[i]]] += f;
66         res += f;
67         pp = pri;
68     }
69     else if (++p[x] < sz[x]) {
70         int j = po[x] + p[x];
71         if (p[v[j]] < 0 && c[j] > 0 && r[v[j]] == 1 + r[x])
72             ind[pp] = j, path[++pp] = v[j];
73     }
74     else pp--;
75 }
76 return res;
77 }
78 long flujo(int ss, int tt) {
79     S = ss; T = tt;
80     n = max(n, max(S, T)) + 1; // Aca, n ya tiene el valor posta
81     _iniG();
82     forn(i, n) po[i] -= sz[i];
83
84     long res = 0, c;
85     do {res += (c = aumentar());} while (c > 0);
86     return res;
87 }

```

8.2. Intersección (y yerbas afines) de círculos en $O(n^3 \lg n)$

```

1  typedef double real;
2  struct pto {
3      real x, y;
4      pto() : x(0), y(0) {}
5      pto(real xx, real yy) : x(xx), y(yy) {}
6      pto operator +(const pto &o) const { return pto(x+o.x, y+o.y); }
7      pto operator -(const pto &o) const { return pto(x-o.x, y-o.y); }
8      pto operator *(real k) const { return pto(k*x, k*y); }
9      real norma() const { return hypot(x, y); }
10     pto rotar(real alfa) const { return pto(x * cos(alfa) - y * sin(alfa), x * sin(
        alfa) + y * cos(alfa)); }
11 };
12
13 struct circ { pto c; real r; };
14 #define sqr(x) ((x)*(x))

```

```

15 struct event {
16     real x; int t;
17     event(real xx, int tt) : x(xx), t(tt) {}
18     bool operator <(const event &o) const { return x < o.x; }
19 };
20
21 typedef vector<circ> VC;
22 typedef vector<event> VE;
23
24 real cuenta(VE &v, real A, real B) {
25     sort(all(v));
26     real res = 0.0, lx = ((v.empty())?0.0:v[0].x);
27     int contador = 0;
28     forn(i, v.size()) {
29         // ESTA ES LA LINEA MAGICA QUE SE CAMBIA
30         // Cambiando trivialmente el if, hacemos que compute:
31         // interseccion de todos (contador == n), union de todos (contador > 0),
32         // conjunto de puntos cubierto por exactamente k círculos (contador == k
            ), etc
33         if (contador == n) res += v[i].x - lx;
34         contador += v[i].t;
35         lx = v[i].x;
36     }
37     return res;
38 }
39
40 const real PI = 4.0 * atan(1.0);
41
42 // Primitiva de sqrt(r*r - x*x) como funcion real de una variable x.
43 inline real primitiva(real x, real r) {
44     if (x >= r) return r*r*PI/4.0;
45     if (x <= -r) return -r*r*PI/4.0;
46     real raiz = sqrt(r*r - x*x);
47     return 0.5 * (x * raiz + r*r*atan(x/raiz));
48 }
49
50 real interCirc(const VC &v) {
51     vector<real> p; p.reserve(v.size() * (v.size() + 2));
52     forn(i, v.size()) {
53         p.push_back(v[i].c.x + v[i].r);
54         p.push_back(v[i].c.x - v[i].r);
55     }
56     forn(i, v.size())
57     forn(j, i) {
58         const circ &a = v[i], &b = v[j];
59         real d = (a.c - b.c).norma();
60         if (fabs(a.r - b.r) < d && d < a.r + b.r) {
61             real alfa = acos((sqr(a.r) + sqr(d) - sqr(b.r)) / (2.0 * d * a.r));
62             pto vec = (b.c - a.c) * (a.r / d);
63             p.push_back((a.c + vec.rotar(alfa)).x);

```

```

64     p.push_back((a.c + vec.rotar(-alfa)).x);
65 }
66 }
67 sort(all(p));
68 real res = 0.0;
69 forn(i,p.size()-1) {
70     const real A = p[i], B = p[i+1];
71     VE ve; ve.reserve(2 * v.size());
72     forn(j,v.size()) {
73         const circ &c = v[j];
74         real arco = primitiva(B-c.c.x,c.r) - primitiva(A-c.c.x,c.r);
75         real base = c.c.y * (B-A);
76         ve.push_back(event(base + arco,-1));
77         ve.push_back(event(base - arco, 1));
78     }
79     res += cuenta(ve,A,B);
80 }
81 return res;
82 }

```

8.3. Integrador numerico (simpson).

```

1  typedef double Funcion(double);
2  double integrar(Funcion *f, double a,double b, int n) {
3      double h = (b-a)/(double)(n);
4      double res = 0.0;
5      double x0 = a;
6      double fx0 = f(x0);
7      const double h2 = h/2.0;
8      forn(i,n) {
9          double fx0h = f(x0+h);
10         res += fx0 + fx0h + 4.0 * f(x0+h2);
11         x0 += h;
12         fx0 = fx0h;
13     }
14     return res * h / 6.0;
15 }

```

8.4. Componentes biconexas, puentes y puntos de articulacion by Juancito

```

1  // g es la lista de adyacencia de un grafo en forma vector<int>, N es cantidad
2  // de vertices,
3  // MAXN es una cota superior tanto para cantidad de vertices como cantidad de
4  // aristas
5
6  int D[MAXN], L[MAXN], J[MAXN], I[MAXN]; char E[MAXN];
7  int P[2 * MAXN], R, S[2 * MAXN], K, A[MAXN], T;
8  void component() {
9      int r = P[--R]; COMPO_START(); COMPO_V(r);

```

```

8      for (int u = P[R - 1]; u != r; u = P[--R - 1]) {
9          COMPO_V(u);
10         if (D[P[R - 2]] < D[u]) forn(i, J[u]) COMPO_EDGE(u, g[u][i]);
11     } COMPO_END();
12 }
13 void dfs (int r) {
14     E[r] = K = T = R = 0; A[S[K++]] = r] = -1;
15     while (K) { int u = S[--K], v;
16         switch (E[u]) {
17             case 0: L[u] = D[u] = T++; J[u] = I[u] = 0; P[R++] = u;
18             case 1: c1: if (I[u] == (int)g[u].size()) break;
19                 if (D[v = g[u][I[u]]] == -1) {
20                     //if(u == r and I[u]) ARTICULATION(u);
21                     E[A[v] = S[K++] = u] = 2, E[S[K++] = v] = 0;
22                 } else { if (v != A[u] && D[v] < L[u]) L[u] = D[v];
23                     if (D[v] < D[u]) swap(g[u][J[u]++], g[u][I[u]]); //COMP
24                     I[u]++; E[S[K++] = u] = 1;
25                 } break;
26             case 2: v = g[u][I[u]], P[R++] = u;
27                 if (L[v] < L[u]) L[u] = L[v];
28                 //if (L[v] >= D[u] && u != r) ARTICULATION(u);
29                 //if (L[v] >= D[v]) BRIDGE(u, v);
30                 if (L[v] >= D[u]) component(); //COMP
31                 I[u]++; goto c1;
32             }
33         }
34     }
35 void BC()
36 {
37     forn(i, N) D[i] = -1;
38     forn(i, N) if(D[i] == -1) dfs(i);
39 }

```

8.5. Rotaciones

Rodrigues rotation formula (rota \mathbf{v} alrededor de \mathbf{z} vector unitario, segun un angulo θ :

$$\mathbf{v}_{\text{rot}} = \mathbf{v} \cos \theta + (\mathbf{z} \times \mathbf{v}) \sin \theta + \mathbf{z}(\mathbf{z} \cdot \mathbf{v})(1 - \cos \theta)$$

8.6. LIS

```

1  // Las lineas marcadas con // Camino no son necesarias si no se desea
2  // reconstruir el camino.
3
4  #define MAXN 1000000
5
6  int v[MAXN]; // INPU del algoritmo.
7  int mv[MAXN];
8  int mi[MAXN], p[MAXN]; // Camino
9  int l[MAXN]; // Aca apareceria la maxima subsecuencia creciente

```

```

9
10 int lis(int n)
11 {
12     forn(i,n) mv[i] = INF;
13     forn(i,n) mi[i] = -1; // Camino
14     forn(i,n) p[i] = -1; // Camino
15     mv[0] = -INF;
16     int res = 0;
17     forn(i,n)
18     {
19         // Con upper_bound es m\'axima subsecuencia no decreciente.
20         // Con lower_bound es m\'axima subsecuencia creciente.
21         int me = upper_bound(mv,mv+n,v[i]) - mv;
22         p[i] = mi[me-1]; // Camino
23         mv[me] = v[i];
24         mi[me] = i; // Camino
25         if (me > res) res = me;
26     }
27     for(int a = mi[res], i = res - 1; a != -1; a = p[a], i--) // Camino
28         l[i] = a; // Indices: poniendo l[i] = v[a] quedan los valores.
29     return res;
30 }

```

8.7. Flujo de costo minimo vale multiejes

```

1 // Se asumen costos y capacidades no negativos.
2 #define MAXN 100
3 #define MAXM 10000
4 int S,T,N,M;
5 Cost co[MAXM];
6 Cap ca[MAXM], f[MAXM];
7 int g1[MAXM], g2[MAXM];
8
9 void iniG(int n,int s,int t) { N = n; S = s; T = t; M = 0; }
10 void aEje(int d,int h,Cap cap, Cost cost) {
11     f[M] = 0;
12     ca[M] = cap;
13     co[M] = cost;
14     g1[M] = d;
15     g2[M] = h;
16     M++;
17 }
18
19 const Cost INF = 1000000000000000000LL;
20 int p[MAXN];
21 Cost dist[MAXN];
22 inline void foo(int d,int h, Cost cost, int j, Cap mf) {
23     if (mf > 0) {
24         Cost c = dist[d] + cost;
25         if (c < dist[h]) { dist[h] = c; p[h] = j; }

```

```

26     }
27 }
28
29 Cost camAu(Cap &x) {
30     // Bellman ford.
31     forn(i,N) {dist[i] = INF; p[i] = -1;}
32     dist[S] = 0;
33     forn(i,N) forn(j,M) {
34         int d = g1[j], h = g2[j];
35         foo(d,h,co[j],j, ca[j] - f[j]);
36         foo(h,d,-co[j],j, f[j]);
37     } // aca ya tenemos computado el camino optimo para aumentar, si hay.
38     int ac = T;
39     Cap mF = x;
40     while (p[ac] != -1) {
41         int j = p[ac];
42         if (g1[j] == ac) { ac = g2[j]; mF = min(mF,f[j]); }
43         else { ac = g1[j]; mF = min(mF,ca[j] - f[j]); }
44     }
45     if (ac != S) return 0; // No hay camino.
46     ac = T;
47     while (p[ac] != -1) {
48         int j = p[ac];
49         if (g1[j] == ac) { ac = g2[j]; f[j] -= mF; }
50         else { ac = g1[j]; f[j] += mF; }
51     }
52     x -= mF;
53     return mF * dist[T];
54 }
55
56 // flujo recibe la cantidad de flujo deseada (+inf para usar el flujo maximo).
57 // al final f queda con la cantidad de flujo efectivamente pasada
58 // Devuelve el costo del flujo en cuestion.
59 Cost flujo(Cap &f) {
60     Cap f0 = f, lf = f;
61     Cost res = 0;
62     while (1) {
63         res += camAu(f);
64         if (f == lf || f == 0) break;
65         lf = f;
66     }
67     f = f0 - f;
68     return res;
69 }

```

8.8. Dual simple (dual sobre cada componente conexa)

```

1 // Usa : pto (resta), Orden total de puntos alrededor de un centro.
2 #define MAXN 1100
3 #define MAXM 6500

```

```

4
5 int m; // Cantidad de ejes
6 pto nodos[MAXN]; // nodos[i] es la coordenada del nodo i. INPU!!!!!!!!!!!!
7 int ea[2*MAXM], eb[2*MAXM]; // Cada eje va de ea[i] a eb[i]
8
9 void ginit() { m = 0; } // Funciones de entrada
10 void aEje(int a, int b) { ea[m] = a; eb[m++] = b;
11                          ea[m] = b; eb[m++] = a; }
12
13 int indi[2*MAXM]; // Indice del eje en la lista de adyacencia de a
14 vint g[MAXN]; // g : listas de adyacencia
15 int compo[MAXN]; // Componente conexa de cada vertice.
16
17 int reg[2*MAXM]; // Region que toca cada arista OUPU!!!!!!!!!!!!
18 vint dejes[MAXM]; // Lista de ejes de una region
19                      // (caminando con mano izquierda en la pared) OUPU
20
21 Cmp micompa = Cmp(pto());
22 bool micmp(int e1, int e2) { return micompa(nodos[eb[e1]], nodos[eb[e2]]); }
23 bool mimen(pto a, pto b) { if (a.y != b.y) return a.y < b.y; return a.x < b.x; }
24 int tReg;
25 void workReg(int i) {
26     dejes[tReg].clear();
27     int ej = i;
28     do { dejes[tReg].push_back(ej);
29          reg[ej] = tReg;
30          ej = g[eb[ej]][(indi[ej~1]+1) % g[eb[ej]].size()]; }
31     while (ej != i);
32     tReg++;
33 }
34 // Le pasas la cantidad de nodos y una variable donde te deja cant de comp.
   // conexas
35 // Devuelve la cantidad de vertices del dual (regiones en el grafo original).
36 // En el grafo construido, las regiones 0..c-1 son las regiones exteriores de
   // cada componente conexa.
37 int buildDual(int n, int &c) {
38     forn(i, n) g[i].clear();
39     forn(i, m) g[ea[i]].push_back(i);
40     forn(i, n) {
41         micompa.r = nodos[i];
42         sort(all(g[i]), micmp);
43     }
44     forn(i, n) forn(j, g[i].size()) indi[g[i][j]] = j;
45     forn(i, m) reg[i] = -1;
46     forn(i, n) compo[i] = -1;
47     tReg = 0; c = 0;
48     forn(i, n)
49     if (compo[i] == -1) {
50         int menor = i;
51         compo[i] = c;

```

```

52     queue<int> q;
53     q.push(i);
54     while (!q.empty()) {
55         int x = q.front(); q.pop();
56         if (mimen(nodos[x], nodos[menor])) menor = x;
57         forall(ej, g[x]) {
58             int y = eb[*ej];
59             if (compo[y] == -1) {
60                 compo[y] = c;
61                 q.push(y);
62             } } }
63     c++;
64     if (g[menor].empty()) dejes[tReg++].clear();
65     else workReg(*g[menor].begin());
66 }
67 forn(i, m) if (reg[i] == -1) workReg(i);
68 return tReg;
69 }
70 // EJEMPLO DE RECORRIDO DE LAS ARISTAS EN EL DUAL
71 // form(i, tReg) forall(ej, dejes[i]) ARISTA_EN_EL_DUAL(i, reg[*ej~1]);

```

8.9. Dual full

```

1 // usa: point in poly, area de un poligono
2 VP regPol[MAXM]; // Poligono que define el borde EXTERNO de una region (
   // indefinido en la cara externa final)
3 // Luego de llamar a buildDual, llamas a este con el c que te dieron
4 // Te devuelve un N: El nuevo dual tiene vertices [c, N]. [0, c) son basura.
5 // La unica cara externa tiene numero N-1
6 int buildFullDual(int c) {
7     forn(i, tReg) { // Construcción de los poligonos
8         regPol[i] = VP(dejes[i].size());
9         forn(j, dejes[i].size())
10             regPol[i][j] = nodos[ea[dejes[i][j]]];
11     } // Ahora si el full dual
12     dejes[tReg].clear();
13     forn(i, c)
14     if (!dejes[i].empty()) {
15         int mejor = tReg;
16         tint mar = AREA_INF;
17         pto p = nodos[ea[dejes[i][0]]];
18         forsn(j, c, tReg)
19         if (compo[ea[dejes[j][0]]] != i) {
20             if (pnpoly(regPol[j], p)) {
21                 tint ar = areaPor2(regPol[j]);
22                 if (ar < mar) {
23                     mejor = j;
24                     mar = ar;
25                 } } }
26     // Enchufar la region i a la mejor

```

```

27     forall(ej,dejes[i]) {
28         reg[*ej] = mejor;
29         dejes[mejor].push_back(*ej);
30     }
31 }
32 return ++tReg;
33 }

```

8.10. LCA

```

1 // La cantidad maxima de vertices n debe ser menor que 2^(LVL-1).
2 #define LVL 18
3 #define MAXN (1<<LVL)
4 int d[MAXN];
5 bool fcmp(int a,int b) { return d[a] < d[b];}
6 int vec[LVL][MAXN];
7 int mn(int i, int j) { // intervalo [i,j]
8     int p = 31-__builtin_clz(j-i);
9     return min(vec[p][i],vec[p][j-(1<<p)],fcmp);
10 }
11 void mn_init(int n) {
12     int mp = 31-__builtin_clz(n);
13     forn(p, mp) forn(x, n-(1<<p)) vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)],
14         fcmp);
15 }
16 vector<int> t[MAXN]; // Tree
17 int pos[MAXN];
18 int id, pp;
19 void dfs(int x) {
20     d[x] = id++;
21     vec[0][pos[x] = pp++] = x;
22     forall(y,t[x])
23         if (d[*y] == -1) {
24             dfs(*y);
25             vec[0][pp++] = x;
26         }
27 }
28 void lcaInit(int n,int raiz) {
29     id = pp = 0;
30     memset(d,-1,sizeof(d));
31     dfs(raiz);
32     mn_init(2*n-1);
33 }
34 int lca(int i,int j) {
35     int a = pos[i], b = pos[j];
36     if (a > b) swap(a,b);
37     return mn(a,b+1);
38 }

```

8.11. Interseccion semiplano-poligono convexo $O(n)$

```

1 // Usa: pto (con doubles) (+ - , producto cruz ^, producto por un escalar *, ==)
2 const double EPS = 1e-9;
3 pto irs(pto a,pto b,pto p0, pto p1) {
4     #define onr(p) (fabs((b-a)^(p-a)) < EPS)
5     if (onr(p1)) return p1;
6     if (onr(p0)) return p0;
7     return p0 + (p1-p0) * (((a-p0)^(b-a)) / ((p1-p0)^(b-a)));
8 }
9 // Parado en a, mirando hacia b, interseca el semiplano de la izquierda con el
10 // poligono convexo p
11 // Un VP vacio representa el conjunto vacio.
12 VP cortar(const VP &p, pto a, pto b) {
13     #define inside(p) (((b-a)^(p-a)) >= -EPS)
14     int n = p.size();
15     VP res; if (n==0) return p;
16     int in = inside(p[n-1]);
17     for (int i=0,j=n-1;i<n;j=i++) {
18         int nin = inside(p[i]);
19         if (nin != in) {
20             in = nin;
21             res.push_back(irs(a,b,p[j],p[i]));
22         }
23         if (in) res.push_back(p[i]);
24     }
25     res.resize(unique(all(res)) - res.begin());
26     while (res.size() > 1 && res.back() == res.front()) res.pop_back();
27     return res;
28 }

```

8.12. Distancia punto-triángulo en 3D

```

1 // usa pto con operaciones, dis2 (distancia al cuadrado), norma2 (norma al
2 // cuadrado)
3 tipo dis2puntossegmento(pto a, pto b, pto c) {
4     pto ba = b - a, ca = c - a, bc = b - c;
5     tipo t = (ca * ba) / (ba * ba);
6     if(0 <= t and t <= 1) {
7         pto proy = ba * t;
8         pto normal = ca - proy;
9         return normal.norma2();
10     }
11     else return min(dis2(a, c), dis2(b, c));
12 }
13 tipo dis2rectarecta(pto a, pto b, pto c, pto d) {
14     tipo res = dis2puntossegmento(a, b, c);
15     tipo a11 = ba * ba, a12 = -(dc * ba), a21 = ba * dc, a22 = -(dc * dc);
16     tipo det = a11 * a22 - a12 * a21;
17     if(zero(det)) return res;
18     swap(a11, a22); a12 = -a12; a21 = -a21;

```



```

19  tipo t1 = (a11 * (ca * ba) + a12 * (ca * dc)) / det;
20  tipo t2 = (a21 * (ca * ba) + a22 * (ca * dc)) / det;
21  ba = ba * t1, dc = dc * t2;
22  return dis2(ba, ca + dc);
23 }
24
25 tipo dis2segseg(pto a, pto b, pto c, pto d) {
26  tipo res = INF;
27  res = min(res, dis2puntosegmento(a, b, c));
28  res = min(res, dis2puntosegmento(a, b, d));
29  res = min(res, dis2puntosegmento(c, d, a));
30  res = min(res, dis2puntosegmento(c, d, b));
31
32  pto ba = b - a, dc = d - c, ca = c - a;
33  tipo a11 = ba * ba, a12 = -(dc * ba), a21 = ba * dc, a22 = -(dc * dc);
34  tipo det = a11 * a22 - a12 * a21;
35  if(zero(det)) return res;
36  else {
37    swap(a11, a22); a12 = -a12; a21 = -a21;
38    tipo t1 = (a11 * (ca * ba) + a12 * (ca * dc)) / det;
39    tipo t2 = (a21 * (ca * ba) + a22 * (ca * dc)) / det;
40    if(0 <= t1 and t1 <= 1 and 0 <= t2 and t2 <= 1) {
41      ba = ba * t1, dc = dc * t2;
42      return min(res, dis2(ba, ca + dc));
43    }
44    else return res;
45  }
46 }
47
48 tipo mindis2puntotrialgulo(pto a, pto b, pto c, pto x) {
49  pto ba = b - a, ca = c - a, xa = x - a, caba = ca - ba;
50  tipo a11 = ba * ba, a12 = ba * caba, a21 = ba * caba, a22 = caba * caba;
51  tipo det = a11 * a22 - a12 * a21;
52  if(zero(det)) return INF;
53  else {
54    swap(a11, a22); a12 = -a12; a21 = -a21;
55    tipo t1 = (a11 * (xa * ba) + a12 * (xa * caba)) / det;
56    tipo t2 = (a21 * (xa * ba) + a22 * (xa * caba)) / det;
57    if(0 <= t2 and t2 <= t1 and t1 <= 1) {
58      ba = ba * t1; caba = caba * t2;
59      return dis2(xa, ba + caba);
60    }
61    else return INF;
62  }
63 }

```

8.13. Algoritmo de Duval

Dada una string s devuelve la Lyndon decomposition en tiempo lineal usando el algoritmo de Duval. Factoriza s como $s_1 s_2 \dots s_k$ con $s_1 \geq s_2 \geq \dots \geq s_k$ y tal que s_i es

Lyndon, esto es, es su menor rotación.

```

1 void duval(char* s) {
2   int i = 0, n = strlen(s), j, k;
3   while (i < n) {
4     j = i + 1, k = i;
5     while (j < n and s[k] <= s[j]) {
6       if(s[k] < s[j]) k = i; else k++; j++; }
7     while (i <= k) {
8       LYNDON(i, i + j - k); i += j - k; }}

```

Obtener la mínima rotación de s : en la descomposición de Lyndon de s^2 es el último $i < |s|$ con el que empieza una Lyndon.

Dada una string s devuelve un array $m[0:n]$ tal que $m[i]$ contiene el mínimo sufijo de $s[0:i+1]$.

```

1 void minimumSuffixArray (char* s, int* res) {
2   int i = 0, n = strlen(s), j, k;
3   while (i < n) {
4     j = i + 1; k = i; res[i] = i;
5     while (j < n and s[k] <= s[j]) {
6       if (s[k] < s[j]) res[j] = k = i;
7       else res[j] = j - k + res[k], k++; j++; }
8     while (i <= k) i += j - k; }}

```

Dada una string s devuelve un array $m[0:n]$ tal que $m[i]$ contine el máximo sufijo de $s[0:i+1]$.

```

1 void maximumSuffixArray (char* s, int* res) {
2   int i = 0, n = strlen(s), j, k; forn(1, n) res[1] = -1;
3   while (i < n) {
4     j = i + 1; k = i;
5     if (res[i] == -1) res[i] = i;
6     while (j < n and s[k] >= s[j]) {
7       if (s[k] > s[j]) k = i; else k++;
8       if (res[j] == -1) res[j] = i; j++; }
9     while (i <= k) i += j - k; }}

```