1. #Template
    1.1. C++ Template
    1.2. Java Template
    1.3. Python Template
2. Data Structures
    2.1. 2D RMQ
    2.2. Convex Hull Optimization
    2.3. KD Tree
    2.4. Persistent Treap
    2.5. RB Tree
    2.6. Rectangle Union O(n log n)
    2.7. Rectilinear MST O(n log n)
3. Geometry
    3.1. Delaunay Triangulation
    3.2. Minimum Enclosing Disk O(N) expected time
    3.3. Pick Theorem O(n)
    3.4. Primitives
4. Graph
    4.1. Dinic
    4.2. Dominator Tree O((N+M)logN)
    4.3. Heavy Light Decomposition
    4.4. Hopcroftâ□□Karp Bipartite Matching O(Msqrt(N))
    4.5. Hungarian O(N^3)
    4.6. Max Flow Min Cost
    4.7. Minimum Arborescences O(MlogN)
    4.8. Punto de Art. y Bridges O(N)
    4.9. SQRT On Tree
    4.10. Stable Marriage
    4.11. StoerWagner O(N^3)
    4.12. Tree Isomorphism O(NlogN)
5. Number Theory
    5.1. Algoritmo Shanka-Tonelli (x^2 = a(mod p))
    5.2. Extended GCD (ax+by = gcd(a,b))
    5.3. FFT O(NlogN)
    5.4. Fast Modulo Transform O(NlogN)
    5.5. Find a primitive root of a prime number
    5.6. Floyds Cycle-Finding algorithm
    5.7. Gauss O(N^3)
    5.8. Inverso Modular
    5.9. Josephus
    5.10. Linear Recurrence Solver O(N^2logK)
    5.11. Matrix Exponentiation O(N^3log(N))
    5.12. Miller-Rabin is prime (probability test)
    5.13. Modular Equations (ax = b(n))
    5.14. Newton Raphston
    5.15. Newton's Method
    5.16. Parametric Self-Dual Simplex method O(n+m)
    5.17. Pollard Rho O(sqrt(s(n))) expected
    5.18. Shanks' Algorithm O(sqrt(N)) (a^x = b(mod m))
    5.19. Simpson Rule
    5.20. Teorema Chino del Resto
6. String
    6.1. Aho Corasick
    6.2. Lyndon Decomposition O(N)
    6.3. Manacher O(N)
    6.4. Palindrome Tree O(N)
    6.5. Suffix Array O(NlogN)
    6.6. Suffix Automata O(N)
    6.7. Tandems O(NlogN)
    6.8. Z Algorithm O(N)

1. #Template

1.1. C++ Template

```
#define optimizar_io ios_base::sync_with_stdio(0);cin.tie(0);
#include <inttypes.h>
static void main2() {
    char *ppp;
    printf("hello world %p\n", &ppp);
}
static void run_with_stack_size(void (*func)(),size_t stsize){
    char *stack, *send;
    stack=(char *)malloc(stsize);
    send=stack+stsize-16;
    send=(char *)((uintptr_t)send/16*16);
    asm volatile(
      "mov %%rsp, (%0)\n"
      "mov %0, %%rsp\n"
      :
      : "r" (send));
    func();
    asm volatile(
      "mov (%0), %%rsp\n"
      :
      : "r" (send));
    free(stack);
}
int main() {
    run_with_stack_size(main2, 64*1024*1024);
}
```

1.2. Java Template

```
import java.io.IOException;
import java.math.*;
import java.util.*;

public class main {

    public static void main(String[] args)throws IOException{
            //FileReader rd = new FileReader("a.in");
            Scanner cin = new Scanner(System.in);

                  while( cin.hasNext() )
                    int y = cin.nextInt();
            List<Integer> B = new ArrayList<Integer>();

            int [] C = new int[10];

            for( int i = 1; i <= 100; i += 5 ) B.add(i);

            Collections.sort(B);
```

```
            int a = Collections.binarySearch(B, 7);

            B.set(2, 7);
            BigInteger d = cin.nextBigInteger();

            System.out.println(B.get(2));
            System.out.printf("%d", 5);
            cin.close();
    }
}
```

1.3. Python Template

```
import string
import math
import fractions

@memoize
def funcion( s ):
    print s[0:2]
    s = sorted( s )
    return s

arr = []
arr.append( 5 )
arr.append( 1 )
arr = funcion(arr)
print arr[0:5]
for x in range(0, 10):
   arr.append( x )
gg = fractions.gcd(10, 65)

while True:
    try:
        n, c, d  = raw_input().split()
        import math
        print pow( long(c), long(n), long(d) )
    except EOFError:
        break
```

2. Data Structures

2.1. 2D RMQ

```
void build( ){ // O(n*m*log(n)*log(m))
   for(int i = 0; i < n ; i ++){
     for(int j = 0; j < m ; j ++)
           table[0][0][i][j] = Matrix[i][j];
     for(int lj = 1; lj <= log2( m ); lj ++)
           for(int j = 0; j + (1<<(lj-1)) < m; j ++)
            table[0][lj][i][j] =
                  min(table[0][lj-1][i][j],
                        table[0][lj-1][i][j+(1<<(lj-1))]);
   }
   for(int li = 1; li <= log2(n); li ++ )
     for(int i = 0; i < n; i ++ )
           for(int lj = 0; lj <= log2(m); lj++ )
             for(int j = 0; j < m; j ++ )
                   table[li][lj][i][j] =
                       min(table[li-1][lj][i][j],
                             table[li-1][lj][i+(1<<(li-1))][j]);
}
int Query(int x1,int y1,int x2,int y2){
   int lenx=x2-x1+1;
    int kx=log2(lenx);
    int leny=y2-y1+1;
    int ky=log2(leny);
   int min_R1 = min ( table[kx][ky][x1][y1] ,
                   table[kx][ky][x1][y2-(1<<ky) + 1] );
   int min_R2 = min ( table[kx][ky][x2-(1<<kx) + 1][y1],
                   table[kx][ky][x2-(1<<kx)+1][y2-(1<<ky)+1] );
   return min ( min_R1, min_R2 );
}
```

2.2. Convex Hull Optimization

```
//para buscar maximo
typedef complex<ll> point;
typedef vector<point> hull;
ll cross(point a, point b){return imag(conj(a) * b);}
ll dot(point a, point b){ return real(conj(a) * b); }
void add(point a, hull &ch){
    for(int n = (int)ch.size(); n > 1 &&
        cross(ch[n-1]-ch[n-2], a-ch[n-2]) >= 0; n--)
            ch.pop_back();
    ch.push_back(a);
}
ll eval(point a, hull &ch){
    int lo = 0, hi = (int)ch.size()-1;
    while(lo < hi){
        int m = (lo + hi)/2;
        if( dot(ch[m], a) >= dot(ch[m+1], a) ) hi = m;
        else lo = m + 1;
    }
    return dot(ch[lo], a);
}
hull merge(const hull &a, const hull &b){
    int n =(int)a.size(), m =(int)b.size(), x=0, y=0;
    hull c;
    while(x < n && y < m){
        if(real(a[x]) <= real(b[y])) add(a[x++], c);
        else add(b[y++], c);
    }
    while (x < n) add(a[x++], c);
    while (y < m) add(b[y++], c);
    return c;
}
struct dyn{
    vector<hull> H;
    void add(point p){
        hull h; h.push_back(p);
        for (int i = 0; i < (int)H.size(); ++i){
            hull &ch = H[i];
            if (ch.empty()){ ch = h; return; }
```

```
            h = merge(h, ch);
            ch.clear();
        }
        if (!h.empty()) H.push_back(h);
    }
    ll query(point p){
        ll answer = -1ll<<60;
        for (int i = 0; i < (int)H.size(); ++i){
            hull &ch = H[i];
            if(ch.empty()) continue;
            answer = max( answer, eval(p, ch) );
        }
        return answer;
    }
};
```

2.3. KD Tree

```
struct point {
    int x, y;
} P[maxn];
bool cmpx ( const point &a, const point &b ) {
    return a.x < b.x;
}
bool cmpy ( const point &a, const point &b ) {
    return a.y < b.y;
}
inline ll dist ( point a, point b ) {
    return 1ll*(a.x-b.x)*(a.x-b.x)+1ll*(a.y-b.y)*(a.y-b.y);
}
struct kd {
    kd *h1, *h2;
    point p;
}*KD;
void init ( int ini, int fin, kd *nod, int split ) {
    sort ( P+ini, P+1+fin, (!split)?cmpx : cmpy );
    int piv = ( ini+fin )>> 1;
    nod->p = P[piv];
    if ( ini < piv ) {
        nod->h1 = new kd();
        init ( ini, piv-1, nod->h1, split^1 );
    }
    if ( piv+1 <= fin ) {
        nod->h2 = new kd();
        init ( piv+1, fin, nod->h2, split^1 );
    }
}
ll best;
void query ( kd *nod, point p, int split ) {
    best = min ( best, dist ( p, nod->p ) );
    ll tmp = ( !split )? p.x - nod->p.x : p.y - nod->p.y;
    if ( tmp < 0 ) {
        if ( nod->h1 )
            query ( nod->h1, p, split^1 );
        if ( nod->h2 && tmp*tmp < best )
            query ( nod->h2, p, split^1 );
    } else {
        if ( nod->h2 )
            query ( nod->h2, p, split^1 );
        if ( nod->h1 && tmp*tmp < best )
            query ( nod->h1, p, split^1 );
    }
}
```

2.4. Persistent Treap

```
/* Careful with memory and recommended
   to use Garbage Collection  */
typedef struct item* pitem;
struct item {
    int val, sz;
    pitem l, r;
    item ( ) {
        val = 0;
        sz = 1;
        l = r = 0;
    }
};
int sz ( pitem t ) { return (t)? t->sz : 0; }
void upd_sz ( pitem t ) {
    t->sz = sz(t->l) + sz(t->r) + 1;
}
typedef tuple<pitem,pitem> tupla;
tupla split ( pitem v, int k ) {
    if ( !v ) return make_tuple ( v, v );
    pitem l, r, ret;
    ret = new item();//ret = v ( treap )
    ret->val = v->val;
    if ( k >= sz(v->l) + 1 ) {
        tie(l,r) = split ( v->r, k-sz(v->l)-1 );
        ret->l = v->l;//
        ret->r = l;
        upd_sz ( ret );
        return make_tuple ( ret, r );
    } else {
        tie(l,r) = split ( v->l, k );
        ret->r = v->r;//
        ret->l = r;
        upd_sz ( ret );
        return make_tuple( l, ret );
    }
}
pitem merge ( pitem l, pitem r ) {
    if ( !l ) return r;
    if ( !r ) return l;
    pitem clone = new item();//no crear
    int tl = sz(l), tr = sz(r);
    if ( rand() % (tl+tr) < tl ) {
        clone->val = l->val;//clone = l
        clone->l = l->l;
        clone->r = merge ( l->r, r );
    } else {
```

```
        clone->val = r->val;//clone = r
        clone->r = r->r;
        clone->l = merge ( l, r->l );
    }
    upd_sz ( clone );
    return clone;
}
```

2.5. RB Tree

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
typedef tree<
int,
null_type,
less<int>,
rb_tree_tag,
tree_order_statistics_node_update>
ordered_set;
ordered_set X; //declaracion
X.insert(1); // insertar
X.erase( X.find( 2 ) ); //eliminar
cout<<*X.find_by_order(1)<<endl;// k-th menor elemento
cout<<X.order_of_key(-5)<<endl;//lower_bound(cant. de menores hay)
```

2.6. Rectangle Union O(n log n)

```
struct rectangle {
    ll xl, yl, xh, yh;
};
ll rectangle_area(vector<rectangle> &rs) {
    vector<ll> ys; // coordinate compression
    for (auto r : rs) {
            ys.push_back(r.yl);
            ys.push_back(r.yh);
    }
    sort(ys.begin(), ys.end());
    ys.erase(unique(ys.begin(), ys.end()), ys.end());
    int n = ys.size(); // measure tree
    vector<ll> C(8 * n), A(8 * n);
    function<void(int, int, int, int, int, int)> aux =
                    [&](int a, int b, int c, int l, int r, int k) {
                            if ((a = max(a,l)) >= (b = min(b,r)))
                                    return;
                            if (a == l && b == r) C[k] += c;
                            else {
                                    aux(a, b, c, l, (l+r)/2, 2*k+1);
                                    aux(a, b, c, (l+r)/2, r, 2*k+2);
                            }
                            if (C[k]) A[k] = ys[r] - ys[l];
                            else A[k] = A[2*k+1] + A[2*k+2];
                    };
    struct event {
            ll x, l, h, c;
    };
    vector<event> es;
    for (auto r : rs) {
            int l = lower_bound(ys.begin(), ys.end(), r.yl)
                                                    - ys.begin();
            int h = lower_bound(ys.begin(), ys.end(), r.yh)
                                                    - ys.begin();
            es.push_back({ r.xl, l, h, +1 });
            es.push_back({ r.xh, l, h, -1 });
    }
    sort(es.begin(), es.end(), [](event a, event b)
                    {return a.x != b.x ? a.x < b.x : a.c > b.c;});
    ll area = 0, prev = 0;
    for (auto &e : es) {
            area += (e.x - prev) * A[0];
            prev = e.x;
            aux(e.l, e.h, e.c, 0, n, 0);
    }
    return area;
}
```

2.7. Rectilinear MST O(n log n)

```
typedef complex<ll> point;
ll rectilinear_mst(vector<point> ps){
    vector<int> id(ps.size());
    iota(id.begin(), id.end(), 0);
    struct edge{
            int src, dst;
            ll weight;
    };
    vector<edge> edges;
    for (int s = 0; s < 2; ++s){
      for (int t = 0; t < 2; ++t){
        sort(id.begin(), id.end(), [&](int i, int j){
              return real(ps[i] - ps[j]) <
                              imag(ps[j] - ps[i]);
            });
            map<ll, int> sweep;
            for (int i : id){
              for (auto it = sweep.lower_bound(-imag(ps[i]));
                                it != sweep.end(); sweep.erase(it++)){
                    int j = it->second;
                    if (imag(ps[j] - ps[i]) < real(ps[j] - ps[i]))
                      break;
                    ll d = abs(real(ps[i] - ps[j]))
                             + abs(imag(ps[i] - ps[j]));
                            edges.push_back({ i, j, d });
              }
                    sweep[-imag(ps[i])] = i;
            }
              for (auto &p : ps)
                    p = point(imag(p), real(p));
            }
            for (auto &p : ps)
              p = point(-real(p), imag(p));
    }
    ll cost = 0;
```

```
    sort(edges.begin(), edges.end(), [](edge a, edge b){
            return a.weight < b.weight;
    });
    union_find uf(ps.size());
    for (edge e : edges)
            if (uf.join(e.src, e.dst))
                    cost += e.weight;
    return cost;
}
```

3. Geometry

3.1. Delaunay Triangulation

```
/*Incremental Randomized Expected O(NlogN)*/
int n; point P[maxn];
struct edge {
    int t, side;
    edge ( ) { t = -1, side = 0; }
    edge ( int tt, int s ) { t = tt, side = s; }
};
struct triangle {
    point p[3]; edge e[3]; int child[3];
    triangle () {}
    triangle(const point&p0,const point&p1,const point&p2){
        p[0] = p0, p[1] = p1, p[2] = p2;
        child[0] = child[1] = child[2] = 0;
    }
    bool inside(const point &pp) const {
        point a = p[0]-pp, b = p[1]-pp, c = p[2]-pp;
        return cross(a, b) >= 0 &&
         cross(b, c) >= 0 &&
         cross(c, a) >= 0;
    }
};
triangle T[maxn*3]; int ct;
bool is_leaf ( int t ) {
    return !T[t].child[0]&&!T[t].child[1]&&!T[t].child[2];
}
void add_edge ( edge a, edge b ) {
    if ( a.t != -1 ) T[a.t].e[a.side] = b;
    if ( b.t != -1 ) T[b.t].e[b.side] = a;
}
struct Triangulation {
    Triangulation ( ) {
        int M = 1e5 * 3;//multiplicar el maximo valor por 3
        T[0]=triangle(point(-M,-M),point(M,-M),point(0,M));
        ct = 1;
    }
    int find ( int t, const point &p ) {
        while ( !is_leaf(t) ) {
          for ( int i = 0; i < 3; i ++ )
            if (T[t].child[i]&&T[T[t].child[i]].inside(p)){
              t = T[t].child[i]; break;
            }
        } return t;
    }
    void add_point ( const point &p ) {
        int t = find ( 0, p ), tab, tbc, tca;
        tab = ct;
        T[ct++] = triangle ( T[t].p[0], T[t].p[1], p );
        tbc = ct;
        T[ct++] = triangle ( T[t].p[1], T[t].p[2], p );
        tca = ct;
        T[ct++] = triangle ( T[t].p[2], T[t].p[0], p );
        add_edge ( {tab,0}, {tbc,1} );
        add_edge ( {tbc,0}, {tca,1} );
        add_edge ( {tca,0}, {tab,1} );
        add_edge ( {tab,2}, T[t].e[2] );
        add_edge ( {tbc,2}, T[t].e[0] );
        add_edge ( {tca,2}, T[t].e[1] );
        T[t].child[0] = tab; T[t].child[1] = tbc;
        T[t].child[2] = tca;
        flip ( tab, 2 ); flip ( tbc, 2 ); flip ( tca, 2 );
    }
    void flip ( int ti, int pi ) {
        int tj = T[ti].e[pi].t;
        int pj = T[ti].e[pi].side;
        if ( tj == -1 ) return;
           if (!incircle(T[ti].p[0],T[ti].p[1],
                                    T[ti].p[2],T[tj].p[pj])) return;
        int tk = ct;
        T[ct++]=triangle(T[ti].p[(pi+1)%3],
                                   T[tj].p[pj],T[ti].p[pi]);
        int tl = ct;
        T[ct++] = triangle ( T[tj].p[(pj+1)%3],
                                                T[ti].p[pi], T[tj].p[pj]
);
        add_edge ( {tk,0}, {tl,0} );
        add_edge ( {tk,1}, T[ti].e[(pi+2)%3] );
           add_edge ( {tk,2}, T[tj].e[(pj+1)%3] );
           add_edge ( {tl,1}, T[tj].e[(pj+2)%3] );
           add_edge ( {tl,2}, T[ti].e[(pi+1)%3] );
        T[ti].child[0] = tk, T[ti].child[1] = tl,
           T[ti].child[2] = 0;
        T[tj].child[0] = tk, T[tj].child[1] = tl,
           T[tj].child[2] = 0;
        flip ( tk, 1 ); flip ( tk, 2 );
        flip ( tl, 1 ); flip ( tl, 2 );
    }
} delaunay;
void triangulate ( ) {
    delaunay = Triangulation();
    random_shuffle ( P+1, P+1+n );
    for ( int i = 1; i <= n; i ++ )
        delaunay.add_point ( P[i] );
}
```

3.2. Minimum Enclosing Disk O(N) expected time

```
circle circumcircle ( const point &a,
   const point &b, const point &c ) {
```

```
    if ( abs( cross( a - c, b - c ) ) > eps  ) {
            point o = three_point_circle ( a, b, c );
            return { o, abs ( o - a ) };
    }
    point p = min ( { a, b, c } );
    point q = max ( { a, b, c } );
    return circle { (p+q)*0.5, abs(p-q)*0.5 };
}
circle min_enclosing_disk_with_2_points ( vector<point> &p,
                            int n, int a, int b ) {
    circle ret =circle {(p[a]+p[b])*0.5,abs(p[a]-p[b])*0.5};
    for ( int i = 0; i <= n; i ++ ) {
            db d = abs ( ret.p - p[i] );
            if ( d <= ret.r + eps ) continue;
            ret = circumcircle ( p[a], p[b], p[i] );
    }
    return ret;
}
circle min_enclosing_disk_with_1_point ( vector<point> &p,
                            int n, int a ) {
    circle ret = circle { p[a], 0 };
    for ( int i = 0; i <= n; i ++ ) {
            db d = abs ( ret.p - p[i] );
            if ( d <= ret.r + eps ) continue;
            ret =min_enclosing_disk_with_2_points( p, i, a, i );
    }
    return ret;
}
circle min_enclosing_disk ( vector<point> &p ) {
    srand(42);
    random_shuffle ( p.begin(), p.end() );

    int n = p.size() - 1;
    circle ret = circle { p[0], 0 };
    for ( int i = 1; i <= n; i ++ ) {
            db d = abs ( ret.p - p[i] );
            if ( d <= ret.r + eps ) continue;
            ret = min_enclosing_disk_with_1_point ( p, i, i );
    }
    return ret;
}
```

3.3. Pick Theorem O(n)

```
/*A = I + B/2 - 1:
  A = Area of the polygon
  I = Number of integer coordinates points inside
  B = Number of integer coordinates points on the boundary
  Polygon's vertex must have integer coordinates */
ll points_on_segment(const line &s){
    point p = s[0] - s[1];
    return __gcd(abs(p.x), abs(p.y));
}
pair<ll, ll> pick_theorem(polygon &P){
    ll A = area2(P), B = 0, I = 0;
    for (int i = 0, n = P.size(); i < n; ++i)
            B += points_on_segment({P[i], P[NEXT(i)]});
    A = abs(A);
    I = (A - B) / 2 + 1;
    return {I, B};// < points inside, points in boundary>
}
```

3.4. Primitives

```
/**1- Base element
   2- The traveling direction of the point (ccw)
   3- Intersection
   4- Distance.
   5- Polygon inclusion decision point
   6- Area of a polygon
   7- Scale a polygon
   8- triangulation possible non convex poly O(n^2)
   9-Convex hull (Andrew's Monotone Chain)
   10-Cutting of a convex polygon
   11-Convex polygon inclusion decision point
   12-Incircle
   13-Closest Pair Point
   14-Three Point Circle
   15-Circle_circle_intersect
   16-Tangents Point Circle
   17-Circle-Line-Intersection
   18-Centroid of a (possibly nonconvex) Polygon
   19-Point rotate **/
///----1-Base element----
struct point {
    db x, y;
    point ( db xx = 0, db yy = 0 ): x(xx), y(yy) { }
    point operator + ( const point &a ) const {
            return { x+a.x, y+a.y };
    }
    point operator - ( const point &a ) const {
            return { x-a.x, y-a.y };
    }
    point operator * ( const db &c ) const {
            return { x*c, y*c };
    }
    point operator * ( const point &p ) const {
            return { x*p.x - y*p.y, x*p.y + y*p.x };
    }
    point operator / ( const db &c ) const {
            return { x/c, y/c };
    }
    point operator / ( const point &a ) const {
         return point { x*a.x + y*a.y, y*a.x - x*a.y } /
            /*divide 2 complejos*/( a.x*a.x + a.y*a.y );
     }
    bool operator < ( const point &a ) const {
            if ( abs( x-a.x ) > eps )
                    return x+eps < a.x;
            return y+eps < a.y;
    }
```

```
};
typedef vector<point> polygon;
struct line : public vector<point> {
  line(const point &a, const point &b) {
    push_back(a); push_back(b);
  }
};
struct circle { point p; db r; };
db cross ( const point &a, const point &b ) {
   return a.x*b.y - a.y*b.x;
}
db dot ( const point &a, const point &b ) {
   return a.x*b.x + a.y*b.y;
}
db norm ( const point &p ) {
   return dot ( p, p );
}
db abs ( const point &p ) {
   return sqrt ( norm(p) );
}
db arg ( const point &p ) {
   return atan2 ( p.y, p.x );
}
point conj ( const point &p ) {
    return point { p.x, -p.y };
}
point crosspoint(const line &l, const line &m) {
  db A = cross(l[1] - l[0], m[1] - m[0]);
  db B = cross(l[1] - l[0], l[1] - m[0]);
  if (abs(A)<eps&&abs(B)<eps) return m[0];//same line
  if (abs(A)<eps)assert(false);//PRECONDITION NOT SATISFIED
  return m[0] + (m[1] - m[0])* B / A;
}
///---2-The traveling direction of the point------
int ccw(point a, point b, point c) {
  b = b-a; c = c-a;
  if (cross(b, c) > 0)return +1; // counter clockwise
  if (cross(b, c) < 0)return -1; // clockwise
  if (dot(b, c) < 0)  return +2; // c--a--b on line
  if (norm(b) < norm(c))return -2;// a--b--c on line
  return 0;
}
///----3-Intersection------
bool intersectLL(const line &l, const line &m) {
  return abs(cross(l[1]-l[0],m[1]-m[0]))>eps//non-parallel
         ||abs(cross(l[1]-l[0],m[0]-l[0]))<eps;//same line
}
bool intersectLS(const line &l, const line &s) {
  return cross(l[1]-l[0], s[0]-l[0])* // s[0] is left of l
        cross(l[1]-l[0],s[1]-l[0])<eps;//s[1] is right of l
}
bool intersectLP(const line &l, const point &p) {
  return abs(cross(l[1]-p, l[0]-p)) < eps;
}
bool intersectSS(const line &s, const line &t) {
  return ccw(s[0],s[1],t[0])*ccw(s[0],s[1],t[1]) <= 0 &&
         ccw(t[0],t[1],s[0])*ccw(t[0],t[1],s[1]) <= 0;
}
bool intersectSP(const line &s, const point &p) {
  return abs(s[0]-p)+abs(s[1]-p)-abs(s[1]-s[0])<eps;
}
///---4-Distance----------------
point projection(const line &l, const point &p) {
  db t = dot(p-l[0], l[0]-l[1]) / norm(l[0]-l[1]);
  return l[0] + (l[0]-l[1])*t;
}
point reflection(const line &l, const point &p) {
  return p + point(2,0)*(projection(l, p) - p);
}
double distanceLP(const line &l, const point &p) {
  return abs(p - projection(l, p));
}
double distanceLL(const line &l, const line &m) {
  return intersectLL(l, m) ? 0 : distanceLP(l, m[0]);
}
double distanceLS(const line &l, const line &s) {
  if (intersectLS(l, s)) return 0;
  return min(distanceLP(l, s[0]), distanceLP(l, s[1]));
}
double distanceSP(const line &s, const point &p) {
  const point r = projection(s, p);
  if (intersectSP(s, r)) return abs(r - p);
  return min(abs(s[0] - p), abs(s[1] - p));
}
double distanceSS(const line &s, const line &t) {
  if (intersectSS(s, t)) return 0;
  return min(min(distanceSP(s, t[0]), distanceSP(s, t[1])),
             min(distanceSP(t, s[0]), distanceSP(t, s[1])));
}
///----5-Polygon inclusion decision point----
#define curr(G, i) G[i]
#define next(G, i) G[(i+1)%G.size()]
enum { OUT, ON, IN };
int contains(const polygon &G, const point& p) {
  bool in = false;
  for (int i = 0; i < (int)G.size(); ++i) {
    point a = curr(G,i) - p, b = next(G,i) - p;
    if (a.y > b.y) swap(a, b);
    if (a.y <= 0 && 0 < b.y)
      if (cross(a, b) < 0) in = !in;
    if (cross(a, b) == 0 && dot(a, b) <= 0) return ON;
  }
  return in ? IN : OUT;
}
///----6-Area of a polygon---------------------------
double area2(const polygon& G) {
  double A = 0;
  for (int i = 0; i < (int)G.size(); ++i)
    A += cross(curr(G, i), next(G, i));
  return A;
```

```
158. }
159. ///-----7-Scale a polygon---
160. #define prev(G,i) G[(i-1+G.size())%G.size()]
161. polygon shrink_polygon(const polygon &G, double len) {
162.   polygon res;
163.   for (int i = 0; i < (int)G.size(); ++i) {
164.     point a = prev(G,i), b = curr(G,i), c = next(G,i);
165.     point u = (b - a) / abs(b - a);
166.     double th = arg((c - b)/ u) * 0.5;
167.     res.push_back( b + u * point(-sin(th), cos(th))
168.                                * len / cos(th) );
169.   }
170.   return res;
171. }
172. ///-----8-triangulation possibly non convex poly O(n^2)--
173. polygon make_triangle(const point&a,const point&b,
174.            const point&c){
175.   polygon ret(3);
176.   ret[0] = a; ret[1] = b; ret[2] = c;
177.   return ret;
178. }
179. bool triangle_contains(const polygon&tri,const point&p){
180.   return ccw(tri[0], tri[1], p) >= 0 &&
181.          ccw(tri[1], tri[2], p) >= 0 &&
182.          ccw(tri[2], tri[0], p) >= 0;
183. }
184. bool ear_Q(int i, int j, int k, const polygon& G) {
185.   polygon tri = make_triangle(G[i], G[j], G[k]);
186.   if (ccw(tri[0], tri[1], tri[2]) <= 0) return false;
187.   for (int m = 0; m < (int)G.size(); ++m)
188.     if (m != i && m != j && m != k)
189.       if (triangle_contains(tri, G[m]))
190.         return false;
191.   return true;
192. }
193. void triangulate(const polygon& G, vector<polygon>& t) {
194.   const int n = G.size();
195.   vector<int> l, r;
196.   for (int i = 0; i < n; ++i) {
197.     l.push_back( (i-1+n) % n );
198.     r.push_back( (i+1+n) % n );
199.   }
200.   int i = n-1;
201.   while ((int)t.size() < n-2) {
202.     i = r[i];
203.     if (ear_Q(l[i], i, r[i], G)) {
204.       t.push_back(make_triangle(G[l[i]], G[i], G[r[i]]));
205.       l[ r[i] ] = l[i];
206.       r[ l[i] ] = r[i];
207.     }
208.   }
209. }
210. ///---9-Convex_hull----------------------
211. vector<point> convex_hull(vector<point> ps) {
212.   int n = ps.size(), k = 0;
213.   sort(ps.begin(), ps.end());
214.   vector<point> ch(2*n);
215.   for (int i = 0; i < n; ch[k++] = ps[i++]) // lower-hull
216.     while (k >= 2 && ccw(ch[k-2], ch[k-1], ps[i]) <= 0)--k;
217.  for (int i = n-2,t = k+1;i>=0;ch[k++]=ps[i--])//upper-hull
218.     while (k >= t && ccw(ch[k-2], ch[k-1], ps[i]) <= 0)--k;
219.   ch.resize(k-1);
220.   return ch;
221. }
222. ///---10-Cutting of a convex polygon-----------------
223. polygon convex_cut(const polygon& G, const line& l) {
224.   polygon Q;
225.   for (int i = 0; i < (int)G.size(); ++i) {
226.     point A = curr(G, i), B = next(G, i);
227.     if (ccw(l[0], l[1], A) != -1) Q.push_back(A);
228.     if (ccw(l[0], l[1], A)*ccw(l[0], l[1], B) < 0)
229.       Q.push_back(crosspoint(line(A, B), l));
230.   }
231.   return Q;
232. }
233. ///---11-Convex polygon inclusion decision point-------
234. int convex_contains(const polygon &G, const point &p) {
235.    //G[0] must be the lowest right vertex
236.    int b = 1, e = G.size() - 1;
237.    while(b < e){
238.            int mid = (b + e) / 2;
239.            if(cross( G[0]-p, G[mid]-p) <= eps){
240.                    e = mid;
241.            }
242.            else b = mid + 1;
243.    }
244.    if(cross(G[b]-p,G[b-1]-p)<=eps&&
245.            cross(G[0]-p,G[b]-p)<=eps) if(b > 1
246.            or (G[0].y <= p.y + eps && p.y <= G[1].y + eps))
247.            return IN; // IN or ON
248.    return OUT;
249. }
250. ///--------12-Incircle-------------------------------
251. bool incircle(point a, point b, point c, point p) {
252.   a = a-p; b = b-p; c = c-p;
253.   return norm(a) * cross(b, c)
254.        + norm(b) * cross(c, a)
255.        + norm(c) * cross(a, b) >= 0;
256.       // < : inside, = cocircular, > outside
257. }
258. ///--13-closestPair-------------------------------
259. double closest_pair_points(vector<point> &P) {
260.    auto cmp = [](point a, point b) {
261.            return make_pair(a.y, a.x)
262.                            < make_pair(b.y, b.x);
263.    };
264.    int n = P.size();
265.    sort(P.begin(), P.end());
266.    set<point, decltype(cmp)> S(cmp);
267.    const double oo = 1e9; // adjust
```

```
    double ans = oo;
    for (int i = 0, ptr = 0; i < n; ++i) {
            while (ptr < i && abs(P[i].x - P[ptr].x) >= ans)
                    S.erase(P[ptr++]);
            auto lo = S.lower_bound(point(-oo,P[i].y-ans-eps));
            auto hi = S.upper_bound(point(-oo,P[i].y+ans+eps));
            for (decltype(lo) it = lo; it != hi; ++it)
                    ans = min(ans, abs(P[i] - *it));
            S.insert(P[i]);
    }
    return ans;
}
///----14-Three Point Circle-----------------------
point three_point_circle(const point&a,const point&b,
                                        const point&c){
  point x = (b - a)/norm(b-a), y = (c - a)/norm(c-a);
  return (y-x)/(conj(x)*y - x*conj(y)) + a;
}
///--15-Circle_circle_intersect--------------------
pair<point, point> circle_circle_intersect(const point&c1,
    const double& r1, const point& c2, const double& r2) {
  point A = conj(c2-c1);
  point B = ((c2-c1)*conj(c2-c1))*-1.0 + r2*r2-r1*r1 ;
  point C = (c2-c1)*r1*r1;
  point D = B*B-A*C*4.0;
  complex <db> q ( D.x, D.y );
  q = sqrt(q);
  D = { real(q), imag(q) };
  point z1 = (B*-1.0+D)/(A*2.0)+c1,
            z2 = (B*-1.0-D)/(A*2.0)+c1;
  return pair<point, point>(z1, z2);
}
///--16-Tangents Point Circle----------
vector<point> tangent(point p, circle c) {
    double D = abs(p - c.p);
    if (D + eps < c.r) return {};
    point t = c.p - p;
    double theta = asin( c.r / D );
    double d = cos(theta) * D;
    t = t / abs(t) * d;
    if ( abs(D - c.r) < eps ) return {p + t};
    point rot( cos(theta), sin(theta) );
    return {p + t * rot, p + t * conj(rot)};
}
///-17-Circle-Line-Intersection-----------------------
vector<point> intersectLC( line l, circle c ){
    point u = l[0] - l[1], v = l[0] - c.p;
    double a = dot(u,u), b = dot(u,v),
              cc = dot(v,v) - c.r * c.r;
    double det = b * b - a * cc;
    if ( det < eps ) return { };
    else return { l[0] + u * (-b + sqrt(det)) / a,
                  l[0] + u * (-b - sqrt(det)) / a };
}
///-18--Centroid of a (possibly nonconvex) Polygon
point centroid(const polygon &poly) {
   point c(0, 0);
   double scale = 3.0 * area2(poly);
   for (int i = 0, n = poly.size(); i < n; ++i) {
           int j = (i+1)%n;
           c=c+(poly[i]+poly[j])*(cross(poly[i],poly[j]));
   }
   return c / scale;
}
///-19-Point rotate--------------------------
inline point rotate(point A,double ang){//respect to origin
   double r =  sqrt(dot(A , A));
   double oang = atan2(A.y , A.x);
   return (point){cos(ang + oang),sin(ang + oang)} * r;
}
```

4. Graph

4.1. Dinic

```
int pos, Index[MAXN];///index = -1, pos = 0
int lv[MAXN], Id[MAXN], in, fin, n;
struct edges{ ///N cant de nodos
 int nod, newn, cap, next;
 edges( int a = 0, int b = 0, int c = 0, int e = 0 ){
  nod = a, newn = b, cap = c, next = e;
 }
 int nextn ( int a ){
  return ( nod == a )? newn : nod;
 }
}G[MAXE];
///nod, newn, cap
void insertar( int a, int b, int c ){
 G[pos] = edges( a, b, c, Index[a] );
 Index[a] = pos ++;
 G[pos] = edges( b, a, 0, Index[b] );
 Index[b] = pos ++;
}
queue<int> Q;
bool Bfs( int limt ){
   while( !Q.empty() ) Q.pop();
   fill( lv, lv + n+1, 0);
   lv[in] = 1;
   Q.push( in );
   while( !Q.empty() ) {
           int nod = Q.front();
           Q.pop();
           for( int i = Index[nod]; i != -1; i = G[i].next ){
                   int newn = G[i].newn;
                   if( lv[newn] != 0 || G[i].cap < limt )continue;
                   lv[newn] = lv[nod] + 1;
                   Q.push( newn );
                   if( newn == fin ) return true;
           }
   }
   return false;
```

```
}
bool Dfs( int nod, int limt ){
    if( nod == fin ) return true;
    for( ; Id[nod] != -1; Id[nod] = G[Id[nod]].next ){
            int newn = G[Id[nod]].newn;
            if( lv[nod] + 1 == lv[newn] &&
                    G[Id[nod]].cap >= limt && Dfs( newn, limt ) ){
                    G[Id[nod]].cap -= limt;
                    G[Id[nod]^1].cap += limt;
                    return true;
            }
    }
    return false;
}
int Dinic( ){
    int flow = 0;
    for( int limt = 1024; limt > 0; ){
            if( !Bfs( limt ) ){
             limt >>= 1;
                    continue;
            }
            for( int i = 0; i <= n; i ++ )
                    Id[i] = Index[i];
            while( limt > 0 && Dfs( in, limt ) )
                    flow += limt;
    }
    return flow;
}
```

4.2. Dominator Tree O((N+M)logN)

```
struct graph{
    int n;
    vector<vector<int> > adj, radj, to;
    graph(int n) : n(n), adj(n), radj(n), to(n) {}
    void add_edge(int src, int dst){
            adj[src].push_back(dst);
            radj[dst].push_back(src);
    }
    vector<int> rank, semi, low, anc;
    int eval(int v){
            if (anc[v] < n && anc[anc[v]] < n){
                    int x = eval(anc[v]);
                    if (rank[semi[low[v]]] > rank[semi[x]])
                            low[v] = x;
                    anc[v] = anc[anc[v]];
            }
            return low[v];
    }
    vector<int> prev, ord;
    void dfs(int u){
            rank[u] = ord.size();
            ord.push_back(u);
            for (int i = 0; i < (int) adj[u].size(); ++i){
                    int v = adj[u][i];
                    if (rank[v] < n)
                            continue;
                    dfs(v);
                    prev[v] = u;
            }
    }
    vector<int> idom; // idom[u] is an immediate dominator of u
    void dominator_tree(int r){
            idom.assign(n, n);
            prev = rank = anc = idom;
            semi.resize(n);
            for (int i = 0; i < n; ++i)
                    semi[i] = i;
            low = semi;
            ord.clear();
            dfs(r);
            vector<vector<int> > dom(n);
            for (int x = (int) ord.size() - 1; x >= 1; --x){
                    int w = ord[x];
                    for (int j = 0; j < (int) radj[w].size(); ++j){
                            int v = radj[w][j];
                            int u = eval(v);
                            if (rank[semi[w]] > rank[semi[u]])
                                    semi[w] = semi[u];
                    }
                    dom[semi[w]].push_back(w);
                    anc[w] = prev[w];
                    for (int i=0;i<(int)dom[prev[w]].size();++i){
                            int v = dom[prev[w]][i];
                            int u = eval(v);
                            idom[v] = (rank[prev[w]] > rank[semi[u]]?
                                                                    u : prev[w]);
                    }
                    dom[prev[w]].clear();
            }
            for (int i = 1; i < (int) ord.size(); ++i){
                    int w = ord[i];
                    if (idom[w] != semi[w])
                            idom[w] = idom[idom[w]];
            }
    }
    vector<int> dominators(int u){
            vector<int> S;
            for (; u < n; u = idom[u])
                    S.push_back(u);
            return S;
    }
    void tree(  ){
            for (int i = 0; i < n; ++i){
                    if (idom[i] < n)
                            to[ idom[i] ].push_back( i );
            }
    }
};
```

4.3. Heavy Light Decomposition

```cpp
vector<int> V[MAXN];
int n, sz[MAXN], lv[MAXN], P[MAXN], A[MAXN], B[MAXN], C[MAXN];
// P: padre  A: ult hoja B: pos C:cant
// G[i] = vector<int>( 4*C[i], 0 );
// lv[1] = 1;
void Dfs( int nod = 1, int pad = 0 ){
    int mej = nod;
    A[nod] = nod;
    for( auto i : V[nod] ){
        if( i == pad ) continue;
        lv[i] = lv[nod]+1;
        Dfs( i, nod );
        if( sz[i] > sz[mej] ) mej = i;
        sz[nod] += sz[i];
    }
    mej = A[mej];
    sz[nod] ++;
    P[mej] = pad;
    A[nod] = mej, B[nod] = C[mej];
    C[mej] ++;
}
int sol;
void solve( int a, int b ){
    int a1 = a, b1 = b, dist = 0;
    while( A[a1] != A[b1] ){
       if( lv[ P[ A[a1] ] ] > lv[ P[ A[b1] ] ] )
          dist += lv[a1] - lv[ P[ A[a1] ] ], a1 =  P[ A[a1] ];
        else
          dist += lv[b1] - lv[ P[ A[b1] ] ], b1 =  P[ A[b1] ];
    }
    dist += abs( lv[ a1 ] - lv[ b1 ] );
    int lca = ( lv[a1] > lv[b1] ) ? b1 : a1;

    sol = 0;
    while( A[a] != A[lca] ){
        sol =__gcd(sol,query(A[a],0,C[A[a]]-1,1,B[a],C[A[a]]-1));
        a = P[ A[a] ];
    }

    sol =__gcd(sol, query(A[a], 0, C[A[a]]-1, 1, B[a], B[lca]-1 ));

    while( A[b] != A[lca] ){
        sol =__gcd(sol,query(A[b],0,C[A[b]]-1,1,B[b],C[A[b]]-1));
        b = P[ A[b] ];
    }

    sol =__gcd(sol, query(A[b], 0, C[A[b]]-1, 1, B[b], B[lca] ));
}
```

4.4. HopcroftâKarp Bipartite Matching O(Msqrt(N))

```cpp
const int MAXV = 1001;
const int MAXV1 = 2*MAXV;
vector<int> ady[MAXV];
int D[MAXV1],Mx[MAXV], My[MAXV];
bool BFS(){
    int u, v, i, e;
    queue<int> cola;
    bool f = 0;
    for (i = 0; i < N+M; i++) D[i] = 0;
    for (i = 0; i < N; i++)
        if (Mx[i] == -1) cola.push(i);
    while (!cola.empty()){
        u = cola.front(); cola.pop();
        for (e = ady[u].size()-1; e >= 0; e--) {
            v = ady[u][e];
            if (D[v + N]) continue;
            D[v + N] = D[u] + 1;
            if (My[v] != -1){
                D[My[v]] = D[v + N] + 1;
                cola.push(My[v]);
            }else f = 1;
        }
    }
    return f;
}
int DFS(int u){
    for (int v, e = ady[u].size()-1; e >=0; e--){
        v = ady[u][e];
        if (D[v+N] != D[u]+1) continue;
        D[v+N] = 0;
        if (My[v] == -1 || DFS(My[v])){
            Mx[u] = v;  My[v] = u;  return 1;
        }
    }
    return 0;
}
int Hopcroft_Karp(){
    int i, flow = 0;
    for (i = max(N,M); i >=0; i--) Mx[i] = My[i] = -1;
    while (BFS())
        for (i = 0; i < N; i++)
            if (Mx[i] == -1 && DFS(i))
                ++flow;
     return flow;
}
```

4.5. Hungarian O(N^3)

```cpp
#define MAXN 300
int N,A[MAXN+1][MAXN+1],p,q, oo = 1 <<30;
int fx[MAXN+1],fy[MAXN+1],x[MAXN+1],y[MAXN+1];
int hungarian(){
    memset(fx,0,sizeof(fx));
    memset(fy,0,sizeof(fy));
    memset(x,-1,sizeof(x));
    memset(y,-1,sizeof(y));
    for(int i = 0; i < N; ++i)
        for(int j = 0; j < N; ++j) fx[i] = max(fx[i],A[i][j]);
    for(int i = 0; i < N; ){
        vector<int> t(N,-1), s(N+1,i);
        for(p = q = 0; p <= q && x[i]<0; ++p)
```

```
            for(int k = s[p], j = 0; j < N && x[i]<0; ++j)
                if (fx[k]+fy[j]==A[k][j] && t[j]<0)
                {
                    s[++q]=y[j];
                    t[j]=k;
                    if(s[q]<0)
                        for(p=j; p>=0; j=p)
                            y[j]=k=t[j], p=x[k], x[k]=j;
                }
        if (x[i]<0){
            int d = oo;
            for(int k = 0; k < q+1; ++k)
                for(int j = 0; j < N; ++j)
                    if(t[j]<0) d=min(d,fx[s[k]]+fy[j]-A[s[k]][j]);
            for(int j = 0; j < N; ++j) fy[j]+=(t[j]<0?0:d);
            for(int k = 0; k < q+1; ++k) fx[s[k]]-=d;
        }
        else ++i;
    }
    int ret = 0;
    for(int i = 0; i < N; ++i) ret += A[i][x[i]];
    return ret;
}
```

4.6. Max Flow Min Cost

```
namespace MaxFlowMinCost{
    #define MAXE 1000005
    #define MAXN 100010
   #define oo 1e9
    int pos, Index[MAXN], In, Fin, NN;///index = -1
    typedef int type_cost;
    typedef pair<type_cost, int> par;
    type_cost Phi[MAXN];
    struct edges{
        int nod, newn, cap, next;
        type_cost cost;
        edges( int a=0,int b=0,int c=0,type_cost d=0,int e=0 ){
            nod = a, newn = b, cap = c, cost = d, next = e;
        }
    }G[MAXE];
    void initialize( int cnod, int source, int sink ){
        In = source, Fin = sink, NN = cnod;
        memset( Index, -1, sizeof(Index) );
        pos = 0;
    }
    ///nod, newn, cap, cost
    void insertar( int a, int b, int c, type_cost d ){
        G[pos] = edges( a, b, c, d, Index[a] );
        Index[a] = pos ++;
        G[pos] = edges( b, a, 0, -d, Index[b] );
        Index[b] = pos ++;
    }
    priority_queue<par, vector<par>, greater<par> >Qp;
    int F[MAXN], parent[MAXN];
    type_cost dist[MAXN];
    par Max_Flow_Min_Cost( ){
        int FlowF = 0;
        type_cost CostF = 0;
        intnod, newn, flow;
        type_cost newc, cost;
           memset( Phi, 0, sizeof(Phi) );
        for( ; ; ){
            fill( F, F + 1 + NN, 0 );
            fill( dist, dist + 1 + NN, oo );
            F[In] = oo, dist[In] = 0;
            Qp.push( par( 0, In ) );
            while( !Qp.empty() ){
                nod = Qp.top().second, cost = Qp.top().first;
                Qp.pop();
                flow = F[nod];
                for( int i = Index[nod]; i != -1; i = G[i].next ){
                    newn = G[i].newn;
                    newc = cost + G[i].cost + Phi[nod] - Phi[newn];
                    if( G[i].cap > 0 && dist[newn] > newc ){
                        dist[newn] = newc;
                        F[newn] = min( flow, G[i].cap );
                        parent[newn] = i;
                        Qp.push( par( newc, newn ) );
                    }
                }
            }
            if( F[Fin] <= 0 ) break;
            CostF += (( dist[Fin] + Phi[Fin] ) * F[Fin] );
            FlowF += F[Fin];
            for( int i = In; i <= Fin; i ++ )
                if( F[i] ) Phi[i] += dist[i];
            nod = Fin;
            while( nod != In ){
                G[parent[nod]].cap -= F[Fin];
                G[parent[nod]^1].cap += F[Fin];
                nod = G[parent[nod]].nod;
            }
        }
        return par( CostF, FlowF );
    }
}
```

4.7. Minimum Arborescences O(MlogN)

```
template<typename T>
struct minimum_aborescense{
    struct edge{
            int src, dst;
            T weight;
    };
    vector<edge> edges;
    void add_edge(int u, int v, T w){
            edges.push_back({ u, v, w });
    }
    T solve(int r){
            int n = 0;
```

```
            for (auto e : edges)
                    n = max(n, max(e.src, e.dst) + 1);
            int N = n;
            if( N == 0 ) return 0;
            for (T res = 0;;){
             vector<edge> in(N,{-1,-1,numeric_limits<T>::max()});
                    vector<int> C(N, -1);
                    for (auto e : edges)
                            if (in[e.dst].weight > e.weight)
                                    in[e.dst] = e;
                    in[r] = {r, r, 0};
                    for (int u = 0; u < N; ++u){
                            if (in[u].src < 0)
                                    return numeric_limits<T>::max();
                            res += in[u].weight;
                    }
                    vector<int> mark(N, -1);
                    int index = 0;
                    for (int i = 0; i < N; ++i)     {
                            if (mark[i] != -1)      continue;
                            int u = i;
                            while (mark[u] == -1){
                                    mark[u] = i;
                                    u = in[u].src;
                            }
                            if (mark[u] != i || u == r)
                                    continue;
                            for(int v=in[u].src;u!=v;v=in[v].src)
                                    C[v] = index;
                            C[u] = index++;
                    }
                    if (index == 0) return res;
                    for (int i = 0; i < N; ++i)
                            if (C[i] == -1) C[i] = index++;
                    vector<edge> next;
                    for (auto &e : edges)
                        if(C[e.src]!=C[e.dst]&&C[e.dst]!=C[r])
                                    next.push_back({C[e.src], C[e.dst],
                                            e.weight-in[e.dst].weight});
                    edges.swap(next);
                    N = index;
                    r = C[r];
            }
    }
};
```

4.8. Punto de Art. y Bridges O(N)

```
void bridges_PtoArt ( int nod ){
    Td[nod] = low[nod] = ++ k;
    for( auto num : V[nod] ){
        int newn = G[num].nextn( nod );
        if( G[num].band ) continue;
        G[num].band = true;
        if( Td[newn] ){
            low[nod] = min( low[nod], Td[newn] );
            continue;
        }
        bridges_PtoArt( newn );
        low[nod] = min( low[nod], low[newn] );
        if(Td[nod] < low[newn])
            puente.push(par( nod, newn ));
        if( (Td[nod] == 1 && Td[newn] > 2 ) ||
            ( Td[nod] != 1 && Td[nod] <= low[newn] ) )
            Punto_art[nod] = true;
    }
}
```

4.9. SQRT On Tree

```
void Dfs( int nod, int pad ){
   P[nod] = pad;
   if( lv[nod] % 2 ) G[nod] = ++k;
   for( auto i : V[nod] ){
           if( pad == i ) continue;
           lv[i] = lv[nod]+1;
           Dfs( i, nod );
   }
   if( lv[nod] % 2 == 0 ) G[nod] = ++k;
}
struct r{ int f, s, id; } Q[MAXA]; // f <= s
int R, kk;
bool comp ( const r s1, const r s2 ){
   if( G[s1.f] / R != G[s2.f] / R )
           return G[s1.f] / R < G[s2.f] / R;
   return G[s1.s] < G[s2.s];
}
void mov( int x, int y ){
    int p, cant = 0;
     while( x != y ){
        kk ++;
        if( lv[x] >= lv[y]  ){
            p = P[x];
            if( mark[p] )
               mark[x] = false, remover( A[x] );
            else
               mark[p] = true, add( A[p] );
            x = p;
        }else{
            tmp[++cant] = y;
            y = P[y];
        }
    }
    for( int i = cant; i >= 1; i -- ){
        p = tmp[i];
        if( mark[p] )
            mark[x] = false, remover( A[x] );
        else
            mark[p] = true, add( A[p] );
        x = p;
    }
}
```

4.10. Stable Marriage

```
typedef vector<int> vi; typedef vector<vi> vvi;
#define rep(i,a,b) for ( __typeof(a) i=(a); i<(b); ++i)
vi stable_marriage(int n, int **m, int **w){
    queue<int> q;
    vi at(n, 0), eng(n, -1), res(n, -1); vvi inv(n, vi(n));
    rep(i,0,n) rep(j,0,n) inv[i][w[i][j]] = j;
    rep(i,0,n) q.push(i);
    while (!q.empty()) {
        int curm = q.front(); q.pop();
        for (int &i = at[curm]; i < n; i++) {
            int curw = m[curm][i];
            if (eng[curw] == -1) { }
            else if (inv[curw][curm] < inv[curw][eng[curw]])
                q.push(eng[curw]);
            else continue;
            res[eng[curw] = curm] = curw, ++i; break;
        }
    }
    return res;
}
```

4.11. StoerWagner O(N^3)

```
//maximo flujo seleccionando la mejor fuente y mejor sumidero
int G[MAXN][MAXN], w[MAXN], N;
bool A[MAXN], merged[MAXN];
int StoerWagner(int n){
    int best = 1e8;
    for(int i=1;i<n;++i) merged[i] = 0;
    merged[0] = 1;
    for(int phase=1;phase<n;++phase){
        A[0] = 1;
        for(int i=1;i<n;++i){
            if(merged[i]) continue;
            A[i] = 0;
            w[i] = G[0][i];
        }
        int prev = 0,next;
        for(int i=n-1-phase;i>=0;--i){
            // hallar siguiente vertice que no esta en A
            next = -1;
            for(int j=1;j<n;++j)
                if(!A[j] && (next==-1 || w[j]>w[next]))
                    next = j;
            A[next] = true;
            if(i>0){
                prev = next;
                // actualiza los pesos
                for(int j=1;j<n;++j) if(!A[j])
                   w[j] += G[next][j];
            }
        }
        if(best>w[next]) best = w[next];
        for(int i=0;i<n;++i){// mezcla s y t
            G[i][prev] += G[next][i];
            G[prev][i] += G[next][i];
        }
        merged[next] = true;
    }
    return best;
}
```

4.12. Tree Isomorphism O(NlogN)

```
#define all(c) (c).begin(), (c).end()
struct tree{
    int n;
    vector<vector<int>> adj;
    tree(int n) : n(n), adj(n) {}
    void add_edge(int src, int dst){
            adj[src].push_back(dst);
            adj[dst].push_back(src);
    }
    vector<int> centers(){
            vector<int> prev;
            int u = 0;
            for (int k = 0; k < 2; ++k)     {
                    queue<int> q;
                    prev.assign(n, -1);
                    for (q.push(prev[u] = u); !q.empty(); q.pop()){
                            u = q.front();
                            for (auto v : adj[u]){
                                    if (prev[v] >= 0) continue;
                                    q.push(v);
                                    prev[v] = u;
                            }
                    }
            }
            vector<int> path = { u };
            while (u != prev[u])
                    path.push_back(u = prev[u]);
            int m = path.size();
            if (m % 2 == 0)
                    return {path[m/2-1], path[m/2]};
            else
                    return {path[m/2]};
    }
    vector<vector<int>> layer;
    vector<int> prev;
    int levelize(int r){
            prev.assign(n, -1);
            prev[r] = n;
            layer = {{r}};
            while (1){
                    vector<int> next;
                    for (int u : layer.back())
                            for (int v : adj[u]){
                                    if (prev[v] >= 0)
                                            continue;
                                    prev[v] = u;
```

```
                                    next.push_back(v);
                                }
                    if (next.empty()) break;
                    layer.push_back(next);
            }
            return layer.size();
    }
};
bool isomorphic(tree S, int s, tree T, int t){
    if (S.n != T.n) return false;
    if (S.levelize(s) != T.levelize(t))     return false;

    vector<vector<int>> longcodeS(S.n + 1), longcodeT(T.n + 1);
    vector<int> codeS(S.n), codeT(T.n);
    for (int h = (int) S.layer.size() - 1; h >= 0; --h)     {
            map<vector<int>, int> bucket;
            for (int u : S.layer[h]){
                    sort(all(longcodeS[u]));
                    bucket[longcodeS[u]] = 0;
            }
            for (int u : T.layer[h]){
                    sort(all(longcodeT[u]));
                    bucket[longcodeT[u]] = 0;
            }
            int id = 0;
            for (auto &p : bucket) p.second = id++;
            for (int u : S.layer[h]){
                    codeS[u] = bucket[longcodeS[u]];
                    longcodeS[S.prev[u]].push_back(codeS[u]);
            }
            for (int u : T.layer[h]){
                    codeT[u] = bucket[longcodeT[u]];
                    longcodeT[T.prev[u]].push_back(codeT[u]);
            }
    }
    return codeS[s] == codeT[t];
}
bool isomorphic(tree S, tree T){
    auto x = S.centers(), y = T.centers();
    if (x.size() != y.size()) return false;
    if (isomorphic(S, x[0], T, y[0])) return true;
    return x.size() > 1 && isomorphic(S, x[1], T, y[0]);
}
```

5. Number Theory

5.1. Algoritmo Shanka-Tonelli (x^2 = a(mod p))

```
//devuelve x (mod p) tal que x^2 = a (mod p)
long long solve_quadratic( long long a, int p ){
    if( a == 0 ) return 0;
    if( p == 2 ) return a;
    if( powMod(a,(p-1)/2, p) != 1 ) return -1;
    int phi = p-1, n = 0, k = 0, q = 0;
    while( phi%2 == 0 ) phi/=2, n ++;
    k = phi;
    for( int j = 2; j < p; j ++ )
        if( powMod( j, (p-1)/2, p ) == p-1 ){
          q = j; break;
        }
    long long t = powMod( a, (k+1)/2, p );
    long long r = powMod( a, k, p );
    while( r != 1 ){
        int i = 0, v = 1;
        while( powMod( r, v, p ) != 1 ) v *= 2, i ++;
        long long e = powMod( 2, n-i-1, p );
        long long u = powMod( q, k*e, p );
        t = (t*u)%p;
        r = (r*u*u)%p;
    }
    return t;
}
```

5.2. Extended GCD (ax+by = gcd(a,b))

```
//devuelve x,y tal que ax+by = gcd(a,b)
int64 extended_euclid( int64 a, int64 b, int64& x, int64& y ) {
  int64 g = a;
  x = 1, y = 0;
  if ( b != 0 ) {
    g = extended_euclid( b, a % b, y, x );
    y -= ( a / b ) * x;
  }
  return g;
}
```

5.3. FFT O(NlogN)

```
#define PI acos(-1)
typedef complex<double> base;
void fft (vector<base> & a, int invert){
    int n = (int) a.size();
    for (int i = 1, j = 0; i < n-1; ++i){
            for (int k = n >> 1; (j ^= k) < k; k >>= 1);
            if (i < j) swap(a[i], a[j]);
    }
    for (int len=2; len <= n; len<<=1) {
            double ang = 2*PI/len * invert;
            base wlen(cos(ang), sin(ang)), w(1);
            for (int i=0; i < n; i += len, w = base(1) )
                    for (int j=0; j<len/2; ++j, w *= wlen ){
                            base u = a[i+j], v = a[i+j+len/2] * w;
                            a[i+j] = u + v;
                            a[i+j+len/2] = u - v;
    }               }
    if (invert == -1){ for (int i=0; i<n; ++i) a[i] /= n; }
} //a la hora de conv. de complex a int real + o - 0.5
```

5.4. Fast Modulo Transform O(NlogN)

```
const int mod = 167772161;
// so the algorithm works until n = 2 ^17 = 131072
const int G = 3; // primitive root
//const int MOD = 1073872897 = 2  ^ 30 + 2 ^ 17 + 1, g = 7
// another good choice is MOD = 167772161 = 2^27+2^25+1, g = 3
```

```
// a bigger choice would be MOD = 3221225473 = 2^31+2^30+1, g = 5
// but it requires unsigned long long for multiplications
// n must be a power of two
// sign = 1
// sign = -1
// fast modulo transform
//   (1) n = 2^k < 2^23
//   (2) only predetermined mod can be used
//   (3) Inverso Modular */
void fmt(vector<ll> &x, int sign = +1){
   int n = x.size();
   for (int i = 0, j = 1; j < n - 1; ++j){
           for (int k = n >> 1; k > (i ^= k); k >>= 1);
           if (j < i) swap(x[i], x[j]);
   }
   ll h = pow(G, (mod - 1) / n, mod);
   if (sign < 0) h = inv(h, mod);
   for (int m = 1; m < n; m *= 2){
           ll w = 1, wk = pow(h, n / (2 * m), mod);
           for (int i = 0; i < m; ++i){
                   for (int j = i; j < n; j += 2 * m){
                           ll u = x[j], d = ( x[j + m] * w ) % mod;
                           x[j] = (u + d)%mod;
                           x[j + m] = (u - d + mod)%mod;
                   }
                   w = w * wk % mod;
           }
   }
   if (sign < 0){
           ll n_inv = inv(n, mod);
           for (auto &a : x) a = (a * n_inv) % mod;
   }
}
```

5.5. Find a primitive root of a prime number

```
// Assuming the Riemnan Hypothesis it runs in O(log^6(p)*sqrt(p))
int generator (int p){
    vector<int> fact;
    int phi = p-1, n = phi;
    for (int i=2; i*i<=n; ++i)
        if (n % i == 0){
            fact.push_back (i);
            while (n % i == 0)
                n /= i;
        }
    if (n > 1) fact.push_back (n);
    for (int res=2; res<=p; ++res){
        bool ok = true;
        for (size_t i=0; i<fact.size() && ok; ++i)
            ok &= powmod (res, phi / fact[i], p) != 1;
        if (ok) return res;
    }
    return -1;
}
```

5.6. Floyds Cycle-Finding algorithm

```
par find_cycle() {
    int t = f(x0), h = f(t), mu = 0, lam = 1;
    while (t != h) t = f(t), h = f(f(h));
    h = x0;
    while (t != h) t = f(t), h = f(h), mu++;
    h = f(t);
    while (t != h) h = f(h), lam++;
    return par(mu, lam);
}
```

5.7. Gauss O(N^3)

```
const int oo = 0x3f3f3f3f;
const double eps = 1e-9;
int gauss(vector<vector<double>> a, vector<double> &ans){
   int n = (int) a.size();
   int m = (int) a[0].size() - 1;
   vector<int> where(m, -1);
   for (int col = 0, row = 0; col < m && row < n; ++col){
           int sel = row;
           for (int i = row; i < n; ++i)
                   if (abs(a[i][col]) > abs(a[sel][col])) sel = i;
           if (abs(a[sel][col]) < eps) continue;
           for (int i = col; i <= m; ++i)
                   swap(a[sel][i], a[row][i]);
           where[col] = row;
           for (int i = 0; i < n; ++i)
                   if (i != row){
                           double c = a[i][col] / a[row][col];
                           for (int j = col; j <= m; ++j)
                                   a[i][j] -= a[row][j] * c;
                   }
           ++row;
   }
   ans.assign(m, 0);
   for (int i = 0; i < m; ++i)
           if (where[i] != -1)
                   ans[i] = a[where[i]][m] / a[where[i]][i];
   for (int i = 0; i < n; ++i)     {
           double sum = 0;
           for (int j = 0; j < m; ++j)
                   sum += ans[j] * a[i][j];
           if (abs(sum - a[i][m]) > eps)
                   return 0;
   }
   for (int i = 0; i < m; ++i)
           if (where[i] == -1)     return oo;
   return 1;
}
```

5.8. Inverso Modular

```
ll inv(ll b, ll M){ //mcd(b,m)==1
   ll u = 1, x = 0, s = b, t = M;
   while( s ){
           ll q = t / s;
```

```
            swap(x -= u * q, u);
            swap(t -= s * q, s);
    }
    return (x %= M) >= 0 ? x : x + M;
}
```

5.9. Josephus

```
// n-cantidad de personas, m es la longitud del salto.
// comienza en la k-esima persona.
ll josephus(ll n, ll m, ll k) {
    ll x = -1;
    for (ll i = n - k + 1; i <= n; ++i) x = (x + m) % i;
    return x;
}
ll josephus_inv(ll n, ll m, ll x){
    for (ll i = n;; i--){
            if (x == i) return n - i;
            x = (x - m % i + i) % i;
    }
    return -1;
}
```

5.10. Linear Recurrence Solver O(N^2logK)

```
/* x[i+n] = a[0] x[i] + a[1] x[i+1] + ... + a[n-1] x[i+n-1]
   with initial solution x[0], x[1], ..., x[n-1]
   Complexity: O(n^2 log k) time, O(n log k) space */
ll linear_recurrence(vector<ll> a, vector<ll> x, ll k){
    int n = a.size();
    vector<ll> t(2 * n + 1);
    function<vector<ll>(ll)> rec = [&](ll k){
            vector<ll> c(n);
            if (k < n) c[k] = 1;
            else{
                    vector<ll> b = rec(k / 2);
                    fill(t.begin(), t.end(), 0);
                    for (int i = 0; i < n; ++i)
                            for (int j = 0; j < n; ++j){
                                    t[i+j+(k&1)] += (b[i]*b[j])%mod;
                                    t[i+j+(k&1)] %= mod;
                            }
                    for (int i = 2*n-1; i >= n; --i)
                            for (int j = 0; j < n; ++j){
                                    t[i-n+j] += (a[j]*t[i])%mod;
                                    t[i-n+j] %= mod;
                            }
                    for (int i = 0; i < n; ++i)
                            c[i] = t[i];
            }
            return c;
    };
    vector<ll> c = rec(k);
    ll ans = 0;
    for (int i = 0; i < x.size(); ++i){
            ans += (c[i] * x[i])%mod;
            ans %= mod;
    }
    return ans;
}
```

5.11. Matrix Exponentiation O(N^3log(N))

```
typedef vector <ll> vect;
typedef vector < vect > matrix;
matrix identity (int n) {
    matrix A(n, vect(n));
    for (int i = 0; i <n; i++) A[i][i] = 1;
    return A;
 }
matrix mul(const matrix &A, const matrix &B) {
  matrix C(A.size(), vect(B[0].size()));
  for (int i = 0; i < C.size(); i++)
    for (int j = 0; j < C[i].size(); j++)
      for (int k = 0; k < A[i].size(); k++){
        C[i][j] += (A[i][k] * B[k][j])%mod;
          C[i][j] %= mod;
      }
  return C;
}
matrix powm(const matrix &A, ll e) {
  return ( e == 0 ) ? identity(A.size()) :
         ( e % 2 == 0 ) ? powm(mul(A, A), e/2) :
                                          mul(A, powm(A, e-1));
}
```

5.12. Miller-Rabin is prime (probability test)

```
bool suspect(ll a, int s, ll d, ll n) {
    ll x = powMod(a, d, n);
    if (x == 1)     return true;
    for (int r = 0; r < s; ++r) {
            if (x == n - 1) return true;
            x = mulmod(x, x, n);
    }
    return false;
}
// {2,7,61,0}                is for n < 4759123141 (= 2^32)
// {2,3,5,7,11,13,17,19,23,0} is for n < 10^16 (at least)
unsigned test[] = { 2, 3, 5, 7, 11, 13, 17, 19, 23, 0 };
bool miller_rabin(ll n) {
    if (n <= 1 || (n > 2 && n % 2 == 0)) return false;
    ll d = n - 1; int s = 0;
    while (d % 2 == 0) ++s, d /= 2;
    for (int i = 0; test[i] < n && test[i] != 0; i++)
            if (!suspect(test[i], s, d, n))
                    return false;
    return true;
}
```

5.13. Modular Equations (ax = b(n))

```
/* Modular Linear Equation Solver O(log(n))
 * Given a, b and n, solves the equation ax = b(n)
 * for x. Returns the vector of solutions, all smaller
```

```
 * than n and sorted in increasing order. */
vector< int > msolve( int a, int b, int n ){
    if( n < 0 ) n = -n;
   int d, x, y;
    d = extended_euclid( a, n, x, y );
    vector< int > r;
    if( b % d ) return r;
    int x0 = ( b / d * x ) % n;
    if( x0 < 0 ) x0 += n;
   x0 = x0 % (n / d);
    for( int i = 0; i < d; i++ )
        r.push_back( ( x0 + i * n / d ) % n );
    return r;
}
```

5.14. Newton Raphston

```
double eval(double P[],int n,  double x){
   double r = 0;
   for(int i = n - 1; i >=0; i--){
           r*=x;
           r+=P[i];
   }
   return r;
}
int main() {
    int test = 1, n;
    while(scanf("%d", &n) && n) {
        double a[10] = {};
        for(int i = n; i >= 0; i--) scanf("%lf", &a[i]);
        double ret[10];
        int m = n;
        for(int i = 0; i < m; i++) {
            double b[10] = {};   // f'(x)
            for(int j = 0; j <= n; j++)
                b[j] = a[j+1]*(j+1);
            double x = 25, tx;  //max_value
            if(i)   x = ret[i-1];
            while(true) {
                double fx =eval(a,n+1,x),ffx =eval(b,n,x);
                tx = x - fx/ffx;
                if(fabs(fx) < 1e-8)
                    break;
                x = tx;
            }
            ret[i] = x;
            for(int j = n; j >= 0; j--)
                a[j] = a[j] + a[j+1]*x;
            for(int j = 0; j <= n; j++)
                a[j] = a[j+1];
            n--;
        }
        printf("Equation %d:", test++);
        n = m;
        sort(ret, ret+n);
        for(int i = 0; i < n; i++) printf(" %.4lf", ret[i]);
        printf("\n");
    }
}
```

5.15. Newton's Method

```
template<class F, class G>
double find_root(F f, G df, double x){
   for (int iter = 0; iter < 100; ++iter){
           double fx = f(x), dfx = df(x);
           x -= fx / dfx;
           if (fabs(fx) < 1e-12)
                   break;
   }
   return x;
}
```

5.16. Parametric Self-Dual Simplex method O(n+m)

```
/* - Solve a canonical LP:
                    min. c x
           s.t. A x <= b
                    x >= 0     */
const double eps = 1e-9, oo = numeric_limits<double>::infinity();
typedef vector<double> vec;
typedef vector<vec> mat;
double simplexMethodPD(mat &A, vec &b, vec &c){
   int n = c.size(), m = b.size();
   mat T(m + 1, vec(n + m + 1));
   vector<int> base(n + m), row(m);
   for(int j = 0; j < m; ++j){
           for (int i = 0; i < n; ++i)
                   T[j][i] = A[j][i];
           T[j][n + j] = 1;
           base[row[j] = n + j] = 1;
           T[j][n + m] = b[j];
   }
   for (int i = 0; i < n; ++i) T[m][i] = c[i];
   while (1){
           int p = 0, q = 0;
           for (int i = 0; i < n + m; ++i)
                   if (T[m][i] <= T[m][p]) p = i;
           for (int j = 0; j < m; ++j)
                   if (T[j][n + m] <= T[q][n + m]) q = j;
           double t = min(T[m][p], T[q][n + m]);
           if (t >= -eps)  {
                   vec x(n);
                   for (int i = 0; i < m; ++i)
                           if (row[i] < n) x[row[i]] = T[i][n + m];
                   // x is the solution
                   return -T[m][n + m]; // optimal
           }
           if (t < T[q][n + m]){
                   // tight on c -> primal update
                   for (int j = 0; j < m; ++j)
                           if (T[j][p] >= eps)
```

```
                                if (T[j][p] * (T[q][n + m] - t) >=
                                        T[q][p] * (T[j][n + m] - t))
                                        q = j;

                        if (T[q][p] <= eps)
                                return oo; // primal infeasible
                }else{
                        // tight on b -> dual update
                        for (int i = 0; i < n + m + 1; ++i)
                                T[q][i] = -T[q][i];
                        for (int i = 0; i < n + m; ++i)
                                if (T[q][i] >= eps)
                                        if (T[q][i] * (T[m][p] - t) >=
                                                T[q][p] * (T[m][i] - t))
                                                p = i;
                        if (T[q][p] <= eps)
                                return -oo; // dual infeasible
                }
                for (int i = 0; i < m + n + 1; ++i)
                        if (i != p) T[q][i] /= T[q][p];
                T[q][p] = 1; // pivot(q, p)
                base[p] = 1;
                base[row[q]] = 0;
                row[q] = p;
                for (int j = 0; j < m + 1; ++j)
                        if (j != q){
                                double alpha = T[j][p];
                                for (int i = 0; i < n + m + 1; ++i)
                                        T[j][i] -= T[q][i] * alpha;
                        }
        }
        return oo;
}
```

5.17. Pollard Rho O(sqrt(s(n))) expected

```
#define func(x)(mulmod(x, x+B, n)+ A )
ll pollard_rho(ll n) {
  if( n == 1 ) return 1;
  if( miller_rabin( n ) )
     return n;
  ll d = n;
  while( d == n ){
    ll A = 1 + rand()%(n-1), B = 1 + rand()%(n-1);
    ll x = 2, y = 2;
    d = -1;
    while( d == 1 || d == -1 ){
        x = func(x), y = func(func(y));
        d = __gcd( x-y, n );
    }
  }
  return abs(d);
}
```

5.18. Shanks' Algorithm O(sqrt(N)) (a^x = b(mod m))

```
//return x such that a^x = b(mod m)
int solve ( int a, int b, int m ){
    int n = (int)sqrt( m + .0 )+1, an = 1;
    for ( int i = 0; i < n; i++ )
        an = (an * a)%m;
    map<int, int>vals;
    for ( int i = 1, cur = an; i <= n; ++ i ){
        if ( ! vals. count ( cur ) )
            vals [ cur ] = i ;
        cur = (cur * an)%m;
    }
    for ( int i = 0, cur = b; i <= n; ++ i ){
        if ( vals. count ( cur ) ){
            int ans = vals [ cur ] * n - i ;
            if ( ans < m )return ans;
        }
        cur = (cur * a)%m;
    }
    return -1;
}
```

5.19. Simpson Rule

```
// Error = O( (delta x)^4 )
const int ITR = 1e4; //must be an even number
double Simpson(double a,double b, double f(double)){
   double s = f(a) + f(b), h = (b - a) / ITR;
   for (int i = 1; i < ITR; ++i) {
           double x = a + h * i;
           s += f(x)*( i&1 ? 4 : 2);
   }
   return s * h/3;
}
```

5.20. Teorema Chino del Resto

```
int resto_chino (vector<int> x, vector<int> m, int k){
    int i, tmp, MOD = 1, RES = 0;
    for (i=0; i <k ; i++) MOD *= m[i];
    for (i =0; i <k ; i++){
        tmp = MOD/m[i];
        tmp *= inverso_mod(tmp, m[i]);
        RES += (tmp*x[i]) % MOD;
    }
    return RES % MOD;
}
```

6. String

6.1. Aho Corasick

```
int tree[MAXN][26], fail[MAXN];
int termina[MAXN], size = 1;
void addWord( string pal ){
    int p = 0;
    for(char c : pal){
        if( !tree[p][c-'a'] )
            tree[p][c-'a'] = size++;
        p = tree[p][c-'a'];
    }
```

```
    //termina[p].push_back( pal_id );
    termina[p] = pal.size();
}
void buildersuffix(){
    queue<int> Q;
    for(int i = 0; i < 26; i++)
        if( tree[0][i] ) Q.push(tree[0][i]);
    while( !Q.empty() ){
        int u, v = Q.front(); Q.pop();
           //for( auto i : termina[fail[v]] )
           //      termina[v].push_back( i );
        termina[v] = max(termina[v], termina[fail[v]]);
        for( int i = 0; i < 26; i++ )
            if(u = tree[v][i]){
                fail[u] = tree[fail[v]][i];
                Q.push( u );
            }else
                tree[v][i] = tree[fail[v]][i];
    }
}
```

6.2. Lyndon Decomposition O(N)

```
/*s = w1w2w3..wk, w1 >= w2 >=...>= wk.
  > Menor RotaciÃ³n LexicogrÃ¡fica:Es el mayor valor
  de i, tal que i < n, en la descomposicion de lyndon
  de la cadena s+s, n = |s|  */
void lyndon( string s ){
    int n = (int)s.length(), i = 0;
    while( i < n ){
        int j = i+1, k = i;
        while( j < n && s[k] <= s[j] ){
            if( s[k] < s[j] ) k = i;
            else k ++;
            j ++;
        }
        while( i <= k ){
            cout << s.substr( i, j-k )<<endl;
            i += j-k;
}    }    }
```

6.3. Manacher O(N)

```
int rad[ 2 * MAXLEN ], n;
char s[MAXLEN];
void manacher( ){ /// i%2!=0 par, i%2==0 impar
  int i, j, k;    /// i -> 2*i o 2*i+1
  for ( i = 0, j = 0; i < 2 * n - 1; i += k ) {
    while ( i - j >= 0 && i + j + 1 < 2 * n &&
            s[(i - j)/2] == s[(i + j + 1)/2] )
              j++;
   rad[i] = j;
    for(k = 1;k <= rad[i] && rad[i-k] != rad[i]-k;k++ )
      rad[ i + k ] = min( rad[ i - k ], rad[i] - k );
   j = max( j - k, 0 );
} }
```

6.4. Palindrome Tree O(N)

```
struct PalindromicTree{
    int tree[MAXN][30], link[MAXN], length[MAXN], sz, ult;
    int diff[MAXN], slink[MAXN], ans[MAXN], sans[MAXN];
    string s;
    void ini(  ){
        memset( tree, 0, sizeof(tree) );
        memset( link, 0, sizeof(link) );
        memset( length, 0, sizeof(length) );
        length[1] = -1, link[1] = 1;
        length[2] = 0, link[2] = 1;
        sz = ult = 2, s.clear();
    }
    int find_x( int suff, int p ){
           int len = length[suff];
           while( p - len < 1 || s[p] != s[p-len-1] )
             suff = link[suff], len = length[suff];
           return suff;
   }
    void insertar( char c ){
        int p = s.size();
        s.push_back( c );
        int suff = find_x( ult, p );
        if( tree[suff][c-'a'] == 0 ){
                    tree[suff][c-'a'] = ++sz;
                    length[sz] = length[suff] + 2;
                    link[sz] = ( length[sz] == 1 )? 2 :
                            tree[find_x( link[suff], p )][c-'a'];
            diff[sz] = length[sz]-length[link[sz]];
            slink[sz] = ( diff[sz]!=diff[link[sz]] )?
                         link[sz] : slink[link[sz]];
        }
           ult = tree[suff][c-'a'];
    }
    void descomponer( int i ){
        ans[i] = 1 << 30;
        for(int v = ult; length[v]>0; v = slink[v]){
                    sans[v]= ans[i -(length[slink[v]] + diff[v])];
                    if(diff[v] == diff[link[v]])
                            sans[v] = min(sans[v], sans[link[v]]);
                    ans[i] = min(ans[i], sans[v] + 1);
           }
    }
}palin;
```

6.5. Suffix Array O(NlogN)

```
int n, _sa[LEN], _b[LEN],  top[LEN], _tmp[LEN];
int LCP[LEN], *SA = _sa, *B = _b, *tmp = _tmp;
char s[LEN];
void build_lcp (){
    for(int i = 0, k = 0; i < n; ++i){
        if(B[i] == n - 1)
            continue;
        for(int j = SA[B[i] + 1]; i + k < n &&
                            j + k < n && s[i+k] == s[j + k]; k++);
```

```
        LCP[B[i]] = k;
        if( k ) k--;
    }
}
void build_sa (){
   //memset 0 -> _sa, _b, _tmp, top, LCP
    s[n] = '\0', n ++;
    int na = (n < 256 ? 256 : n);
    for (int i = 0; i < n ; i++)
        top[B[i] = s[i]]++;
    for (int i = 1; i < na; i++)
        top[i] += top[i - 1];
    for (int i = 0; i < n ; i++)
        SA[--top[B[i]]] = i;
    for (int ok = 1,j = 0;ok < n && j < n-1;ok <<= 1){
        for (int i = 0; i < n; i++){
            j = SA[i] - ok;
            if (j < 0)
                j += n;
            tmp[top[B[j]]++] = j;
        }
        SA[tmp[top[0] = 0]] = j = 0;
        for (int i = 1; i < n; i++){
            if (B[tmp[i]] != B[tmp[i - 1]] ||
                            B[tmp[i]+ok] != B[tmp[i-1] + ok])
                top[++j] = i;
            SA[tmp[i]] = j;
        }
        swap(B, SA), swap(SA, tmp);
    }
    build_lcp();
    n --, s[n] = '\0';
}
```

6.6. Suffix Automata O(N)

```
// Construct:
// Automaton sa; for(char c : s) sa.extend(c);
// 1. Number of distinct substr O( N ):
//    - Find number of different paths --> DFS on SA
//    - f[u] = 1 + sum( f[v] for v in s[u].next
// 2. Number of occurrences of a substr O( N ):
//    - Initially, in extend: s[cur].cnt = 1; s[clone].cnt = 0;
//    - for( auto it = base.rbegin(); it != base.rend(); it ++ ){
//        int p = st[it->second].link;
//        cnt[p] += cnt[it->second];  }
// 3. Find total length of different substrings O( N ):
//    - We have f[u] = number of strings starting from node u
//    - ans[u] = sum(ans[v] + d[v] for v in next[u])
// 4. Lexicographically k-th substring O(N)
//    - Based on number of different substring
// 5. Find first occurrence O(N)
//    - firstpos[cur] = len[cur] - 1, firstpos[clone] = firstpos[q]
// 6. Longest common substring of two strings s, t O(N).
struct state {
    int len, link;
    int fpos;///
    map<char,int>next;
    state( ){
        len = 0, link = -1, fpos = 0;
        next.clear();
    }
};
const int MAXLEN = 100002;
state st[MAXLEN*2];
int sz, last;
set<pair<int,int>> base ;///
int cnt[MAXLEN*2];///
void sa_init() {
    sz = last = 0;
    st[0] = state();
   cnt[0] = 0;
    sz++;
    base.clear();
}
void sa_extend (char c) {
    int cur = sz++;
    st[cur] = state();
    st[cur].len = st[last].len + 1;
    st[cur].fpos = st[cur].len - 1;///
    cnt[cur] = 1 ; ///
    base.insert(make_pair(st[cur].len, cur));///
    int p;
    for (p=last; p!=-1 && !st[p].next.count(c); p=st[p].link)
        st[p].next[c] = cur;
    if (p == -1)
        st[cur].link = 0;
    else {
        int q = st[p].next[c];
        if (st[p].len + 1 == st[q].len)
            st[cur].link = q;
        else {
            int clone = sz++;
            st[clone] = state();
            st[clone].len = st[p].len + 1;
            st[clone].next = st[q].next;
            st[clone].link = st[q].link;
            st[clone].fpos = st[q].fpos;///
            cnt[clone]=0;///
            base.insert(make_pair(st[clone].len,clone)); ///
            for (; p!=-1 && st[p].next[c]==q; p=st[p].link)
                st[p].next[c] = clone;
            st[q].link = st[cur].link = clone;
        }
    }
    last = cur;
}
//6. Longest common substring of two strings s, t.
string lcs (string s, string t) {
    sa_init();
    for (int i=0; i<(int)s.length(); i++)
```

```
        sa_extend (s[i]);
    int v = 0,  l = 0, best = 0,  bestpos = 0;
    for (int i=0; i<(int)t.length(); i++) {
        while (v && !st[v].next.count(t[i])) {
            v = st[v].link;
            l = st[v].len;
        }
        if (st[v].next.count(t[i])) {
            v = st[v].next[t[i]];
            l++;
        }
        if (l > best) best = l,  bestpos = i;
    }
    return t.substr (bestpos-best+1, best);
}
```

6.7. Tandems O(NlogN)

```
void output_tandem (const string & s, int shift,
           bool left, int cntr, int l, int l1, int l2){
   int pos;
   if (left) pos = cntr-l1;
   else pos = cntr-l1-l2-l1+1;
   cout << "[" << shift + pos << ".."; // ini
   cout << shift + pos+2*l-1 << "] = "; // fin
   cout << s.substr (pos, 2*l) << endl;
}
void output_tandems (const string & s, int shift,
           bool left, int cntr, int l, int k1, int k2){
   for (int l1=1; l1<=l; l1++) {
           if (left && l1 == l)  break;
           if (l1 <= k1 && l-l1 <= k2)
              output_tandem(s,shift,left,cntr, l, l1, l-l1);
   }
}
inline int get_z (const vector<int> & z, int i) {
   return 0<=i && i<(int)z.size() ? z[i] : 0;
}
void find_tandems (string s, int shift = 0) {
   int n = (int) s.length();
   if (n == 1)  return;
   int nu = n/2,  nv = n-nu;
   string u = s.substr (0, nu),
           v = s.substr (nu);
   string ru = string (u.rbegin(), u.rend()),
           rv = string (v.rbegin(), v.rend());
   find_tandems (u, shift);
   find_tandems (v, shift + nu);
   vector<int> z1 = z_function (ru),
           z2 = z_function (v + '#' + u),
           z3 = z_function (ru + '#' + rv),
           z4 = z_function (v);
   for (int cntr=0; cntr<n; cntr++) {
           int l, k1, k2;
           if (cntr < nu) {
                   l = nu - cntr;
                   k1 = get_z (z1, nu-cntr);
                   k2 = get_z (z2, nv+1+cntr);
           }else {
                   l = cntr - nu + 1;
                   k1 = get_z (z3, nu+1 + nv-1-(cntr-nu));
                   k2 = get_z (z4, (cntr-nu)+1);
           }
           if(k1 + k2 >= l) // longitud 2*l
              output_tandems(s,shift,cntr<nu,cntr,l,k1,k2);
   }
}
```

6.8. Z Algorithm O(N)

```
vector<int> z_function (const string & s){
   int n = (int) s.length();
   vector<int> z (n);
   for (int i=1, l=0, r=0; i<n; i++) {
           if (i <= r) z[i] = min (r-i+1, z[i-l]);
           while (i+z[i] < n && s[z[i]] == s[i+z[i]])
                   z[i]++;
           if (i+z[i]-1 > r) l = i,  r = i+z[i]-1;
   }
   return z;
}
```