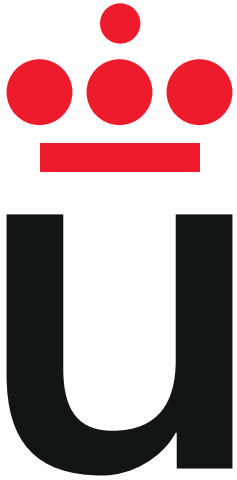


Universidad Rey Juan Carlos

Teamto de Verano

Cristian Perez, Francisco Tortola, Sergio Salazar

Ada Byron 2022

2022

Graph (1)

BFS.java

Description: BFS-DFS
Time: $\mathcal{O}(E+V)$

7aa94f, 17 lines

```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

public class BFS {
    public static void bfs(int vertices, int start, ArrayList<
        Integer>[] graf) {
        boolean[] visitados = new boolean[vertices];
        Queue<Integer> cola = new LinkedList<>();
        cola.add(start);
        while (cola.size() > 0) {
            Integer pop = cola.remove();
            if (graf[pop] == null) continue;
            for (Integer k : graf[pop]) {
                if (!visitados[k]) {
                    cola.add(k);
                    visitados[k] = true;
                } } } } }
```

Tarjan.java

Description: Encontrar los puntos de articulacion, puentes o componentes biconexas de un grafo
Time: $\mathcal{O}(E+V)$

ecc272, 58 lines

```
import java.io.IOException;

public class Main{

    private static final int UNVISITED = 0;

    //Todos los arrays se inicializan a n=numero de vertices
    //HashSet puede ser ArrayList
    private static HashSet<Intpair>[] graf;
    private static int[] dfs_num, dfs_low, dfs_parent,
        articulation_vertex;
    //Inicializar dfs parent con -1.
    //Para i=0 i<n : si dfs_num == UNVISITED lanzar metodo
    private static int dfsNumberCounter, dfsRoot, rootChildren,
        n,puentes;
    //Solo para los puentes
    private static LinkedList<Intpair> lista;
    //Solo Componente biconexas
    //private static LinkedList<HashSet<Integer>> compBiconexas
        = new LinkedList();
    //HashSet<Integer> componenteBiconexa = new HashSet();

    private static void articulationPointAndBridge(int u) {
        //componenteBiconexa.add(u):
        dfs_low[u]= dfsNumberCounter;
        dfs_num[u]= dfsNumberCounter++; // dfs_low[u] <=
            dfs_num[u]
        for (Intpair v_w : graf[u]) {
            if (dfs_num[v_w.x] == UNVISITED) { // a tree edge
                dfs_parent[v_w.x]= u;
                if (u == dfsRoot) ++rootChildren;        //
                    special case, root

                articulationPointAndBridge(v_w.x);

                if (dfs_low[v_w.x] >= dfs_num[u]) // for
                    articulation point
                    articulation_vertex[u]= 1;
                    //compBiconexas.add(componenteBiconexa);
                    //componenteBiconexa=new HashSet<>();
                if (dfs_low[v_w.x] > dfs_num[u]){
                    puentes++;
                    lista.add(new Intpair(Math.min(v_w.x,u),
                        Math.max(v_w.x,u)));
                }
                dfs_low[u]= Math.min(dfs_low[u], dfs_low[v_w.x
                    ]); // update dfs_low[u]
            }
            else if (v_w.x != dfs_parent[u]) // a back edge and
                not direct cycle
                dfs_low[u]= Math.min(dfs_low[u], dfs_num[v_w.x
                    ]); // update dfs_low[u]
        }
        //compBiconexas.add(componenteBiconexa);
        //componenteBiconexa=new HashSet();
    }

    public static void main(String[] args) throws IOException {
        //CONSTRUIR GRAFO
        for (int u = 0; u < n; ++u){
            if (dfs_num[u] == UNVISITED) {
                dfsRoot = u; rootChildren = 0;
                articulationPointAndBridge(u);
                articulation_vertex[dfsRoot]= ((rootChildren >
                    1) ? 1 : 0); // special case
            }
        }
    }
}
```

Dikjstra.java

Description: Shortest Path en un grafo ponderado
Time: $\mathcal{O}(E*log(V))$

2cedb5, 20 lines

```
public class Dikjstra {
    public static void Dikjstra(int nodos, int inicio){
        PriorityQueue<IntPair> pq = new PriorityQueue<>();
        pq.offer(new IntPair(inicio,0)); //offer==add
        int[] dist = new int[nodos];
        Arrays.fill(dist,1000000000);
        dist[inicio]=0;

        while(!pq.isEmpty()){
            IntPair top = pq.poll(); //poll==remove
            int distop=top.d;
            int vtop=top.v;
            if(distop > dist[vtop]) continue;
            for(IntPair aux: graf[vtop]){
                int disaux=aux.d;
                int vaux=aux.v;
                if(dist[vtop]+disaux >= dist[vaux]) continue;
                dist[vaux]=dist[vtop]+disaux;
                pq.offer(new IntPair(vaux,dist[vaux]));
            } } } }
```

FloydWarshall.java

Description: Encontrar la minima distincia entre TODOS los pares de un grafo, el grafo debe estar descrito por su lista de adyacencia graf[][]
Time: $\mathcal{O}(V^3)$

ef3a6f, 11 lines

```
public class FloydWarshall {

    public static int[][] graf;

    public static void FW(int n){
        for(int k=0;k<n;k++)
            for(int i=0;i<n;i++)
                for(int j=0;j<n;j++)
                    graf[i][j] = Math.min(graf[i][j], graf[i][k
                        ]+graf[k][j]);
    }
}
```

TopologicalSort.java

Description: Orden en el que realizar n tareas si 1->2 implica que para hacer 2 hace falta hacer 1
Time: $\mathcal{O}(E+V)$

c778ff, 25 lines

```
public class TopologicalSort {
    public static int n; //vertices
    public static ArrayList<Integer> list;
    public static boolean visitados[];
    public static ArrayList<Integer>[] graf;

    public static void dfs_tps(int u){
        visitados[u]=true;
        for (Integer k : graf[u]) {
            if(!visitados[k]){
                dfs_tps(k);
            }
        }
        list.add(u+1);
    }

    public static void main(String[] args) {
        for(int i=0;i<n;i++){
            if(!visitados[i])
                dfs_tps(i);
        }
        //Recorrido en orden inverso
        for(int i=list.size()-1;i>=0;i--){
            System.out.println(list.get(i));
        } } }
```

MaxFlow.java

Description: Flujo maximo en una red de tuberias
Time: $\mathcal{O}(V*E^2)$

ab6cab, 64 lines

```
public class Max_Flow {

  HashMap<Integer,Integer>[] grafo
    //s=start t=final v=Nvertices
    public static boolean BFS(HashMap<Integer,Integer>[] grafo,
          int s, int t , int parent[], int v){
        boolean[] visited = new boolean[v];
        visited[s]=true;
        LinkedList<Integer> cola = new LinkedList<>();
        cola.addFirst(s);
        parent[s]=-1;
        while(!cola.isEmpty()){
            int aux = cola.remove();
            for(Integer k :  grafo[aux].keySet()){
                if(!visited[k]){
                    if(k==t){
                        parent[t]=aux;
                        return true;
                    }
                    cola.add(k);
                    parent[k]=aux;
                    visited[k]=true;
                }
            }
        }
        return false;
    }
```

```
    public static int fordFulkerson(HashMap<Integer,Integer>[]
         grafo, int s, int t,int v){
        HashMap<Integer,Integer>[] rgrafo = new HashMap[v];
        for(int i=0;i<v;i++){
            rgrafo[i]=new HashMap<>();
            for(Integer k : grafo[i].keySet()){
                rgrafo[i].put(k,grafo[i].get(k));
            }
        }

        int parent[] = new int[v];

        int flujo_maximo=0;

        while(BFS(rgrafo,s,t,parent,v)){
            int flujo=Integer.MAX_VALUE;
            int camino = t;
            while(camino!=s){
                int aux=parent[camino];
                flujo=Math.min(flujo,rgrafo[aux].get(camino));
                camino=parent[camino];
            }
            camino = t;
            while(camino!=s){
                int aux=parent[camino];
                rgrafo[aux].put(camino,rgrafo[aux].get(camino)-
                     flujo);
                if (rgrafo[aux].get(camino) == 0) {
                    rgrafo[aux].remove(camino);
                }
                rgrafo[camino].put(aux,(rgrafo[camino].
                     containsKey(aux) ?  rgrafo[camino].get(aux
                     ) : 0)+flujo);
                camino=parent[camino];
            }
            flujo_maximo+=flujo;
        }
        return flujo_maximo;
    }

}
```

StrongConnectedComponents.java

Description: u-v en la misma scc si existe un camino de u a v y viceversa

Time: $\mathcal{O}(E+V)$

6cf400, 55 lines

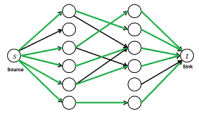
```
public class SCC {
    public static LinkedList<Integer> orden;
    public static ArrayList<Integer>[] graf ;
    public static int[] dfs_num;
    public static int[] dfs_low;
    public static boolean[] visited;
    public static int contador;
    public static int numSCC;

    public static int strongConnectedComponents(int u){
        dfs_low[u]=dfs_num[u]=contador++;
        orden.addLast(u);
        visited[u]=true;
  int size=0;
        for(int i=0;i<graf[u].size();i++){
            int v = graf[u].get(i);
            if(dfs_num[v]==-1){
                size=Math.max(strongConnectedComponents(v),size
                     );
            }
            if(visited[v])
                dfs_low[u]=Math.min(dfs_low[u],dfs_low[v]);
        }
        int auxsize=0;
        if(dfs_low[u]==dfs_num[u]){
            numSCC++;
            System.out.print("SCC "+numSCC+":");
            while(true){
                auxsize++;
                int v = orden.removeLast();
                visited[v]=false;
                System.out.print(v+" ");
                if(u==v) break;
            }
            System.out.println();
        }
  size=Math.max(size,auxsize)
        return size;
    }

    public static void main(String[] args) throws IOException {
        orden = new LinkedList<>();
        dfs_low=new int[h];
        dfs_num=new int[h];
        Arrays.fill(dfs_num,-1);
        Arrays.fill(dfs_low,-1);
        visited=new boolean[h]
        contador=0;
        numSCC=0;
        for(int i=0;i<V;i++){
            if(dfs_num[i]==-1){
                strongConnectedComponents(i);
            }
        }
    }
}
```

1.1 MaximumBipartiteMatching

Reducir el problema al de maxflow, crear un nodo source y otro end. Unimos todos los nodos del conjunto 1 a source y del conjunto 2 a end, el flujo maximo que llega a end es la cantidad de aristas escogidas.



Matematicas (2)

2.1 Ecuaciones

$$ax^2+bx+c=0 \Rightarrow x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

Es extremos es dado por $x=-b/2a$.

$$\left.\begin{array}{l} ax+by=e \\ cx+dy=f \end{array}\right\} \Rightarrow \begin{array}{l} x=\dfrac{ed-bf}{ad-bc} \\ y=\dfrac{af-ec}{ad-bc} \end{array}$$

En general dado un sistema $Ax=b$, la solucion de una variable x_i es dada por

$$x_i=\frac{\det A_i'}{\det A}$$

donde A_i' es A con la i-esima columna remplazada por b.

2.2 Recurrencias

Si $a_n=c_1a_{n-1}+\cdots+c_ka_{n-k}$, y $r_1,\ldots,r_k$ son raices distintas de $x^k+c_1x^{k-1}+\cdots+c_k$, hay $d_1,\ldots,d_k$ tal que

$$a_n=d_1r_1^n+\cdots+d_kr_k^n.$$

Raices diferentes r se convierten en factores polinomiales, e.g. $a_n=(d_1n+d_2)r^n$.

2.3 Trigonometria

$$\sin(v+w)=\sin v\cos w+\cos v\sin w$$
$$\cos(v+w)=\cos v\cos w-\sin v\sin w$$

$$\tan(v+w)=\frac{\tan v+\tan w}{1-\tan v\tan w}$$
$$\sin v+\sin w=2\sin\frac{v+w}{2}\cos\frac{v-w}{2}$$
$$\cos v+\cos w=2\cos\frac{v+w}{2}\cos\frac{v-w}{2}$$

$$(V+W)\tan(v-w)/2=(V-W)\tan(v+w)/2$$

donde V,W son longitudes de angulos de lados opuestos v,w.

$$a\cos x+b\sin x=r\cos(x-\phi)$$
$$a\sin x+b\cos x=r\sin(x+\phi)$$

donde $r=\sqrt{a^2+b^2}$, $\phi=\text{atan2}(b,a)$.

2.4 Geometria

2.4.1 Triangles

Longitudes de los lados: a,b,c

Semiperimetro: $p=\dfrac{a+b+c}{2}$

Area: $A=\sqrt{p(p-a)(p-b)(p-c)}$

Circumradio: $R=\dfrac{abc}{4A}$

Inradio: $r=\dfrac{A}{p}$

Longitud de la mediana (Divide el triangulo en dos triangulos con el mismo area): $m_a=\frac{1}{2}\sqrt{2b^2+2c^2-a^2}$

Longitud de la bisectriz (Divide un angulo en dos):

$$s_a=\sqrt{bc\left[1-\left(\frac{a}{b+c}\right)^2\right]}$$

Teorema del seno: $\frac{\sin\alpha}{a} = \frac{\sin\beta}{b} = \frac{\sin\gamma}{c} = \frac{1}{2R}$

Teorema del coseno: $a^2 = b^2 + c^2 - 2bc\cos\alpha$

Teorema de la tangente: $\frac{a+b}{a-b} = \frac{\tan\frac{\alpha+\beta}{2}}{\tan\frac{\alpha-\beta}{2}}$

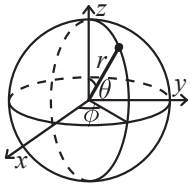
2.4.2 Cuadrilateros

Con lados de longitud a, b, c, d, diagonales e, f, angulos de la diagonal θ, area A y "flujo magico" $F = b^2 + d^2 - a^2 - c^2$:

$$4A = 2ef \cdot \sin\theta = F\tan\theta = \sqrt{4e^2f^2 - F^2}$$

Para cuadrilateros ciclicos la suma de los angulos opuestos es $180°$, $ef = ac + bd$, y $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$.

2.4.3 Coordenadas esfericas



$$\begin{array}{ll} x = r\sin\theta\cos\phi & r = \sqrt{x^2+y^2+z^2} \\ y = r\sin\theta\sin\phi & \theta = \arccos(z/\sqrt{x^2+y^2+z^2}) \\ z = r\cos\theta & \phi = \arctan(y/x) \end{array}$$

2.5 Derivadas e Integrales

$$\frac{d}{dx}\arcsin x = \frac{1}{\sqrt{1-x^2}} \qquad \frac{d}{dx}\arccos x = -\frac{1}{\sqrt{1-x^2}}$$

$$\frac{d}{dx}\tan x = 1 + \tan^2 x \qquad \frac{d}{dx}\arctan x = \frac{1}{1+x^2}$$

$$\int \tan ax = -\frac{\ln|\cos ax|}{a} \qquad \int x\sin ax = \frac{\sin ax - ax\cos ax}{a^2}$$

$$\int e^{-x^2} = \frac{\sqrt{\pi}}{2}\mathrm{erf}(x) \qquad \int xe^{ax}dx = \frac{e^{ax}}{a^2}(ax-1)$$

Integracion por partes:

$$\int_a^b f(x)g(x)dx = [F(x)g(x)]_a^b - \int_a^b F(x)g'(x)dx$$

2.6 Sumas

$$c^a + c^{a+1} + \cdots + c^b = \frac{c^{b+1} - c^a}{c-1}, c \neq 1$$

$$1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(2n+1)(n+1)}{6}$$

$$1^3 + 2^3 + 3^3 + \cdots + n^3 = \frac{n^2(n+1)^2}{4}$$

$$1^4 + 2^4 + 3^4 + \cdots + n^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$$

2.7 Series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots, (-\infty < x < \infty)$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, (-1 < x \leq 1)$$

$$\sqrt{1+x} = 1 + \frac{x}{2} - \frac{x^2}{8} + \frac{2x^3}{32} - \frac{5x^4}{128} + \ldots, (-1 \leq x \leq 1)$$

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots, (-\infty < x < \infty)$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots, (-\infty < x < \infty)$$

2.8 Cadenas de Markov

Una *Cadena de Markov* es un proceso aleatorio discreto con la propiedad de que el siguiente estado depende unicamente del estado actual. Sean $X_1, X_2, \ldots$ unas secuencia de variables aleatorias generadas por un proceso de Markov. Entonces hay una matriz de transicion $\mathbf{P} = (p_{ij})$, con $p_{ij} = \Pr(X_n = i | X_{n-1} = j)$, y $\mathbf{p}^{(n)} = \mathbf{P}^n\mathbf{p}^{(0)}$ es la distribucion de probabilidad para X_n (i.e., $p_i^{(n)} = \Pr(X_n = i)$), donde $\mathbf{p}^{(0)}$ es la distribucion incial.

Distribucion estacionaria

π es una distribucion estacionaria si $\pi = \pi\mathbf{P}$. Si la cadena de Markov es *irreducible* (es posible ir de cualquier estado a otro), entonces $\pi_i = \frac{1}{\mathbb{E}(T_i)}$ donde $\mathbb{E}(T_i)$ es el tiempo esperado entre dos visitas en el estado i. π_j/π_i es el numero experado de visitas en el estado j entre dos visitas del estado i.

Para un grafo conexo, no dirigido and no-bipartito, donde la la probabilidad de transicion es uniforme entre todos los vecinos, π_i es proporcional al grado del nodo i.

Ergodicidad

Una cadena de Markov es *ergodica* if the asymptotic si la distribucion asintotica es independiente del estado inicial de la distribucion. Una cadena de Markov finita es ergodica si es irreducible y *aperiodica* (i.e., el mcd de la longitud de los ciclos es 1). $\lim_{k\to\infty}\mathbf{P}^k = \mathbf{1}\pi$.

Absorvencia

Una cadena de Markov es una A-cadena si los estados pueden ser particionados en dos conjuntos **A** y **G**, tal que todos los estados en **A** son absorventes ($p_{ii} = 1$), y tdos los estados en **G** acaban en un estado absorvente de**A**. La probabilidad para absorver en un estado $i \in \mathbf{A}$, donde el estado inicial es j, es $a_{ij} = p_{ij} + \sum_{k\in\mathbf{G}} a_{ik}p_{kj}$. El tiempo esperado hasta la absorcion, donde el estado inicial es i, es $t_i = 1 + \sum_{k\in\mathbf{G}} p_{ki}t_k$.

Data structures (3)

SegmentTree.h

Description: Zero-indexed max-tree. Bounds are inclusive to the left and exclusive to the right. Can be changed by modifying T, f and unit.

Time: $\mathcal{O}(\log N)$ 0f4bdb, 18 lines

```
struct Tree {
  typedef int T;
  static constexpr T unit = INT_MIN;
  T f(T a, T b) { return max(a, b); } // (any associative fn)
  vector<T> s; int n;
  Tree(int n = 0, T def = unit) : s(2*n, def), n(n) {}
  void update(int pos, T val) {
    for (s[pos += n] = val; pos /= 2;)
      s[pos] = f(s[pos * 2], s[pos * 2 + 1]);
  }
  T query(int b, int e) { // query [b, e)
    T ra = unit, rb = unit;
    for (b += n, e += n; b < e; b /= 2, e /= 2) {
      if (b % 2) ra = f(ra, s[b++]);
      if (e % 2) rb = f(s[--e], rb);
    }
    return f(ra, rb);
  }};
```

UnionFind.java

Description: Conjuntos de union buscar 84e122, 29 lines

```
static class UnionFind {
        private ArrayList<Integer> p, rank, setSize;
        private int numSets;
        public UnionFind(int N) {
            p = new ArrayList<>(N);
            rank = new ArrayList<>(N);
            setSize = new ArrayList<>(N);
            numSets = N;
            for (int i = 0; i < N; i++) {
                p.add(i);
                rank.add(0);
                setSize.add(1);
            } }

        public int findSet(int i) {
            if (p.get(i) == i) return i;
            else {
                int ret = findSet(p.get(i)); p.set(i, ret);
                return ret; } }

        public Boolean isSameSet(int i, int j) { return findSet
             (i) == findSet(j); }

        public void unionSet(int i, int j) {
            if (!isSameSet(i, j)) { numSets--;
                int x = findSet(i), y = findSet(j);
```

```
    if (rank.get(x) > rank.get(y)) { p.set(y, x);
        setSize.set(x, setSize.get(x) + setSize.
        get(y)); }
    else{
        p.set(x, y); setSize.set(y, setSize.get(y)
            + setSize.get(x));
        if (rank.get(x) == rank.get(y)) rank.set(y,
            rank.get(y) + 1); } } }  }
```

FenwickTree.h

Description: Computes partial sums a[0] + a[1] + ... + a[pos - 1], and updates single elements a[i], taking the difference between the old and new value.

Time: Both operations are $\mathcal{O}(\log N)$.

e62fac, 22 lines

```
struct FT {
  vector<ll> s;
  FT(int n) : s(n) {}
  void update(int pos, ll dif) { // a[pos] += dif
    for (; pos < sz(s); pos |= pos + 1) s[pos] += dif;
  }
  ll query(int pos) { // sum of values in [0, pos)
    ll res = 0;
    for (; pos > 0; pos &= pos - 1) res += s[pos-1];
    return res;
  }
  int lower_bound(ll sum) {// min pos st sum of [0, pos] >= sum
    // Returns n if no sum is >= sum, or -1 if empty sum is.
    if (sum <= 0) return -1;
    int pos = 0;
    for (int pw = 1 << 25; pw; pw >>= 1) {
      if (pos + pw <= sz(s) && s[pos + pw-1] < sum)
        pos += pw, sum -= s[pos-1];
    }
    return pos;
  }
};
```

Numerical (4)

4.1 Polynomials and recurrences

Polynomial.h

c9b7b0, 17 lines

```
struct Poly {
  vector<double> a;
  double operator()(double x) const {
    double val = 0;
    for (int i = sz(a); i--;) (val *= x) += a[i];
    return val;
  }
  void diff() {
    rep(i,1,sz(a)) a[i-1] = i*a[i];
    a.pop_back();
  }
  void divroot(double x0) {
    double b = a.back(), c; a.back() = 0;
    for(int i=sz(a)-1; i--;) c = a[i], a[i] = a[i+1]*x0+b, b=c;
    a.pop_back();
  }
};
```

PolyRoots.h

Description: Finds the real roots to a polynomial.

Usage: `polyRoots({{2,-3,1}},-1e9,1e9) // solve x^2-3x+2 = 0`

Time: $\mathcal{O}\left(n^2 \log(1/\epsilon)\right)$

"Polynomial.h" b00bfe, 23 lines

```
vector<double> polyRoots(Poly p, double xmin, double xmax) {
  if (sz(p.a) == 2) { return {-p.a[0]/p.a[1]}; }
  vector<double> ret;
  Poly der = p;
  der.diff();
  auto dr = polyRoots(der, xmin, xmax);
  dr.push_back(xmin-1);
  dr.push_back(xmax+1);
  sort(all(dr));
  rep(i,0,sz(dr)-1) {
    double l = dr[i], h = dr[i+1];
    bool sign = p(l) > 0;
    if (sign ^ (p(h) > 0)) {
      rep(it,0,60) { // while (h - l > 1e-8)
        double m = (l + h) / 2, f = p(m);
        if ((f <= 0) ^ sign) l = m;
        else h = m;
      }
      ret.push_back((l + h) / 2);
    }
  }
  return ret;
}
```

PolyInterpolate.h

Description: Given n points (x[i], y[i]), computes an n-1-degree polynomial p that passes through them: $p(x) = a[0] * x^0 + ... + a[n-1] * x^{n-1}$. For numerical precision, pick $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \ldots n-1$.

Time: $\mathcal{O}\left(n^2\right)$

08bf48, 13 lines

```
typedef vector<double> vd;
vd interpolate(vd x, vd y, int n) {
  vd res(n), temp(n);
  rep(k,0,n-1) rep(i,k+1,n)
    y[i] = (y[i] - y[k]) / (x[i] - x[k]);
  double last = 0; temp[0] = 1;
  rep(k,0,n) rep(i,0,n) {
    res[i] += y[k] * temp[i];
    swap(last, temp[i]);
    temp[i] -= last * x[k];
  }
  return res;
}
```

4.2 Matrices

Determinant.h

Description: Calculates determinant of a matrix. Destroys the matrix.

Time: $\mathcal{O}\left(N^3\right)$

bd5cec, 15 lines

```
double det(vector<vector<double>>& a) {
  int n = sz(a); double res = 1;
  rep(i,0,n) {
    int b = i;
    rep(j,i+1,n) if (fabs(a[j][i]) > fabs(a[b][i])) b = j;
    if (i != b) swap(a[i], a[b]), res *= -1;
    res *= a[i][i];
    if (res == 0) return 0;
    rep(j,i+1,n) {
      double v = a[j][i] / a[i][i];
      if (v != 0) rep(k,i+1,n) a[j][k] -= v * a[i][k];
    }
  }
  return res;
}
```

SolveLinear.h

Description: Solves $A * x = b$. If there are multiple solutions, an arbitrary one is returned. Returns rank, or -1 if no solutions. Data in A and b is lost.

Time: $\mathcal{O}\left(n^2 m\right)$

44c9ab, 38 lines

```
typedef vector<double> vd;
const double eps = 1e-12;

int solveLinear(vector<vd>& A, vd& b, vd& x) {
  int n = sz(A), m = sz(x), rank = 0, br, bc;
  if (n) assert(sz(A[0]) == m);
  vi col(m); iota(all(col), 0);

  rep(i,0,n) {
    double v, bv = 0;
    rep(r,i,n) rep(c,i,m)
      if ((v = fabs(A[r][c])) > bv)
        br = r, bc = c, bv = v;
    if (bv <= eps) {
      rep(j,i,n) if (fabs(b[j]) > eps) return -1;
      break;
    }
    swap(A[i], A[br]);
    swap(b[i], b[br]);
    swap(col[i], col[bc]);
    rep(j,0,n) swap(A[j][i], A[j][bc]);
    bv = 1/A[i][i];
    rep(j,i+1,n) {
      double fac = A[j][i] * bv;
      b[j] -= fac * b[i];
      rep(k,i+1,m) A[j][k] -= fac*A[i][k];
    }
    rank++;
  }

  x.assign(m, 0);
  for (int i = rank; i--;) {
    b[i] /= A[i][i];
    x[col[i]] = b[i];
    rep(j,0,i) b[j] -= A[j][i] * b[i];
  }
  return rank; // (multiple solutions if rank < m)
}
```

MatrixInverse.h

Description: Invert matrix A. Returns rank; result is stored in A unless singular (rank < n). Can easily be extended to prime moduli; for prime powers, repeatedly set $A^{-1} = A^{-1}(2I - AA^{-1}) \pmod{p^k}$ where A^{-1} starts as the inverse of A mod p, and k is doubled in each step.

Time: $\mathcal{O}\left(n^3\right)$

ebfff6, 35 lines

```
int matInv(vector<vector<double>>& A) {
  int n = sz(A); vi col(n);
  vector<vector<double>> tmp(n, vector<double>(n));
  rep(i,0,n) tmp[i][i] = 1, col[i] = i;

  rep(i,0,n) {
    int r = i, c = i;
    rep(j,i,n) rep(k,i,n)
      if (fabs(A[j][k]) > fabs(A[r][c]))
        r = j, c = k;
    if (fabs(A[r][c]) < 1e-12) return i;
    A[i].swap(A[r]); tmp[i].swap(tmp[r]);
    rep(j,0,n)
      swap(A[j][i], A[j][c]), swap(tmp[j][i], tmp[j][c]);
    swap(col[i], col[c]);
    double v = A[i][i];
    rep(j,i+1,n) {
      double f = A[j][i] / v;
      A[j][i] = 0;
      rep(k,i+1,n) A[j][k] -= f*A[i][k];
      rep(k,0,n) tmp[j][k] -= f*tmp[i][k];
    }
```

```
    rep(j,i+1,n) A[i][j] /= v;
    rep(j,0,n) tmp[i][j] /= v;
    A[i][i] = 1;
  }

  for (int i = n-1; i > 0; --i) rep(j,0,i) {
    double v = A[j][i];
    rep(k,0,n) tmp[j][k] -= v*tmp[i][k];
  }

  rep(i,0,n) rep(j,0,n) A[col[i]][col[j]] = tmp[i][j];
  return n;
}
```

Number theory (5)

5.1 Modular arithmetic

ModMulLL.java

Description: Calcula $a \cdot b \bmod c$ (or $a^b \bmod c$) para $0 \leq a, b \leq c \leq 7.2 \cdot 10^{18}$.
Time: $\mathcal{O}(1)$ para `modmul`, $\mathcal{O}(\log b)$ para `modpow`

e47d88, 23 lines

```
static int BITS=10;
//Si todos los numeros son menores a 2^k   BITS=64-k;
static long po = 1 << BITS;

static long mod_mul(long a, long b, long mod){
        long x = a * (b & (po-1)) %mod;
        while ((b >>= BITS) > 0) {
            a = (a << BITS) % mod;
            x += (a * (b & (po - 1))) % mod;
        }
        return x % mod;
}
static long mod_pow(long a,long b, long mod){
        long res = 1;
        a = a % mod;
        while (b > 0)
        {
            if ((b & 1) > 0) res = (res * a) % mod;
            b = b >> 1;
            a = (a * a) % mod;
        }
        return res;
}
```

ModInverse.java

Description: Calcula x tal que $a * x = 1 mod m$ a y m son coprimos
Time: $\mathcal{O}(log(m))$

66aa9c, 26 lines

```
public class ModInverse {
    // Returns modulo inverse of a with respect to m using
         extended Euclid
    // Algorithm Assumption: a and m are coprimes, i.e., gcd(a,
         m) = 1
    static int modInverse(int a, int m)
    {
        int m0 = m;
        int y = 0, x = 1;
        if (m == 1) return 0;
        while (a > 1) {
            // q is quotient
            int q = a / m;
            int t = m;
            // m is remainder now, process same as Euclid's
                 algo
            m = a % m;
            a = t;
            t = y;
            // Update x and y
            y = x - q * y;
            x = t;
        }
        // Make x positive
        if (x < 0)
            x += m0;
        return x;
    }
}
```

ModLog.h

Description: Returns the smallest $x > 0$ s.t. $a^x = b \pmod m$, or -1 if no such x exists. modLog(a,1,m) can be used to calculate the order of a.
Time: $\mathcal{O}\left(\sqrt{m}\right)$

c040b8, 11 lines

```
ll modLog(ll a, ll b, ll m) {
  ll n = (ll) sqrt(m) + 1, e = 1, f = 1, j = 1;
  unordered_map<ll, ll> A;
  while (j <= n && (e = f = e * a % m) != b % m)
    A[e * b % m] = j++;
  if (e == b % m) return j;
  if (__gcd(m, e) == __gcd(m, b))
    rep(i,2,n+2) if (A.count(e = e * f % m))
      return n * i - A[e];
  return -1;
}
```

ModSum.h

Description: Sums of mod'ed arithmetic progressions.
`modsum(to, c, k, m)` $= \sum_{i=0}^{to-1} (ki + c) \% m$. `divsum` is similar but for floored division.
Time: $\log(m)$, with a large constant.

5c5bc5, 16 lines

```
typedef unsigned long long ull;
ull sumsq(ull to) { return to / 2 * ((to-1) | 1); }

ull divsum(ull to, ull c, ull k, ull m) {
  ull res = k / m * sumsq(to) + c / m * to;
  k %= m; c %= m;
  if (!k) return res;
  ull to2 = (to * k + c) / m;
  return res + (to - 1) * to2 - divsum(to2, m-1 - c, m, k);
}

ll modsum(ull to, ll c, ll k, ll m) {
  c = ((c % m) + m) % m;
  k = ((k % m) + m) % m;
  return to * c + k * sumsq(to) - m * divsum(to, c, k, m);
}
```

ModSqrt.h

Description: Tonelli-Shanks algorithm for modular square roots. Finds x s.t. $x^2 = a \pmod p$ ($-x$ gives the other solution).
Time: $\mathcal{O}\left(\log^2 p\right)$ worst case, $\mathcal{O}(\log p)$ for most p

"ModPow.h" 19a793, 24 lines

```
ll sqrt(ll a, ll p) {
  a %= p; if (a < 0) a += p;
  if (a == 0) return 0;
  assert(modpow(a, (p-1)/2, p) == 1); // else no solution
  if (p % 4 == 3) return modpow(a, (p+1)/4, p);
  // a^(n+3)/8 or 2^(n+3)/8 * 2^(n-1)/4 works if p % 8 == 5
  ll s = p - 1, n = 2;
  int r = 0, m;
  while (s % 2 == 0)
    ++r, s /= 2;
  while (modpow(n, (p - 1) / 2, p) != p - 1) ++n;
  ll x = modpow(a, (s + 1) / 2, p);
  ll b = modpow(a, s, p), g = modpow(n, s, p);
  for (;; r = m) {
    ll t = b;
    for (m = 0; m < r && t != 1; ++m)
      t = t * t % p;
    if (m == 0) return x;
    ll gs = modpow(g, 1LL << (r - m - 1), p);
    g = gs * gs % p;
    x = x * gs % p;
    b = b * g % p;
  }
}
```

5.2 Primality

SieveOfEratosthenes.java

Description: Generar primos hasta cierto límite
Time: lim=100'000'000 $\approx$ 0.8 s.

90dfae, 16 lines

```
public class SieveOfEratosthenes {
    static LinkedList<Integer> sieveOfErastosthenes(int n){
        boolean prime[] = new boolean[n+1];
        LinkedList<Integer>out = new LinkedList<>();
        for(int p=2; p*p<=n; p++)
            if(!prime[p]) {
                for (int i = p * p; i <= n; i += p)
                    prime[i] = true;
            }
        //Metemos los primos a una lista. prime[p] es falso si
             p es primo.
        for(int i=2; i<=n; i++)
            if(!prime[i])
                out.add(i);
        return out;
    }
}
```

SieveOfErastosthenesFast.java

Description: Generar primos hasta cierto límite
Time: $\mathcal{O}(n)$

be8dd0, 37 lines

```
import java.util.Vector;
class SieveOfErastosthenesFast {
    static final int MAX_SIZE = 1000001;
    // SPF: guarda el factor primo mas pequeno de un numero
    //prime: vector con todos los numeros primos
    static Vector<Boolean>isprime = new Vector<>(MAX_SIZE);
    static Vector<Integer>prime = new Vector<>();
    static Vector<Integer>SPF = new Vector<>(MAX_SIZE);
    // method generate all prime number less then N in O(n)
    static void manipulated_seive(int N) {
        for (int i = 0; i <= N; i++){
            isprime.add(true);
            SPF.add(2);
        }
        // 0 and 1 are not prime
        isprime.set(0, false);
        isprime.set(1, false);
        // Fill rest of the entries
        for (int i=2; i<=N ; i++) {
            // If isPrime[i] == True then i is
            // prime number
            if (isprime.get(i)) {
                // put i into prime[] vector
                prime.add(i);
                // A prime number is its own smallest prime
                     factor
                SPF.set(i, i);
            }
```

```
        // Remove all multiples of i*prime[j] which are not
            prime by making isPrime[i*prime[j]] = false
        // and put smallest prime factor of i*Prime[j] as
            prime[j]
        // [for exp :let i = 5, j = 0, prime[j] = 2 [ i*
            prime[j] = 10]
        // so smallest prime factor of '10' is '2' that is
            prime[j] ]
        // this loop run only one time for number which are
            not prime
        for (int j = 0; j < prime.size() && i * prime.get(j
            ) < N && prime.get(j) <= SPF.get(i); j++) {
            isprime.set(i * prime.get(j), false);
            // put smallest prime factor of i*prime[j]
            SPF.set(i * prime.get(j), prime.get(j));
        }}}}
```

MillerRabin.java

Description: Determina de forma lineal si un numero es primo, funcionalidad grantizada para numeros menores a 7* 10**18

Usage: `ModMulLL.java`

Time: 7 veces la complejidad de $a^b mod c$

e3c2be, 21 lines

```
static boolean miillerTest(int d, int n) {
        int a = 2 + (int)(Math.random() % (n - 4));
        int x = power(a, d, n);
        if (x == 1 || x == n - 1)  return true;
        while (d != n - 1) {
            x = (x * x) % n;
            d *= 2;

            if (x == 1)return false;
            if (x == n - 1) return true;
        }
        return false;
}
static boolean isPrime(int n, int k) {
        if (n <= 1 || n == 4)  return false;
        if (n <= 3) return true;
        int d = n - 1;
        while (d % 2 == 0) d /= 2;
        for (int i = 0; i < k; i++) if (!miillerTest(d, n))
             return false;
        return true;
    }
```

Factor.h

Description: Pollard-rho randomized factorization algorithm. Returns prime factors of a number, in arbitrary order (e.g. 2299 -> {11, 19, 11}).

Time: $\mathcal{O}\left(n^{1/4}\right)$, less for numbers with small factors.

"ModMulLL.h", "MillerRabin.h" a33cf6, 18 lines

```
ull pollard(ull n) {
  auto f = [n](ull x) { return modmul(x, x, n) + 1; };
  ull x = 0, y = 0, t = 30, prd = 2, i = 1, q;
  while (t++ % 40 || __gcd(prd, n) == 1) {
    if (x == y) x = ++i, y = f(x);
    if ((q = modmul(prd, max(x,y) - min(x,y), n))) prd = q;
    x = f(x), y = f(f(y));
  }
  return __gcd(prd, n);
}
vector<ull> factor(ull n) {
  if (n == 1) return {};
  if (isPrime(n)) return {n};
  ull x = pollard(n);
  auto l = factor(x), r = factor(n / x);
  l.insert(l.end(), all(r));
  return l;
}
```

5.3 Divisibility

Euclid.java

Description: Finds {x, y, d} s.t. ax + by = d = gcd(a, b).

6aba01, 11 lines

```
static BigInteger[] euclid(BigInteger a, BigInteger b) {
  BigInteger x = BigInteger.ONE, yy = x;
  BigInteger y = BigInteger.ZERO, xx = y;
  while (b.signum() != 0) {
    BigInteger q = a.divide(b), t = b;
    b = a.mod(b); a = t;
    t = xx; xx = x.subtract(q.multiply(xx)); x = t;
    t = yy; yy = y.subtract(q.multiply(yy)); y = t;
  }
  return new BigInteger[]{x, y, a};
}
```

CRT.java

Description: Teorema chino de los restos. Usa ModInverse

Time: $\mathcal{O}(n * log(n))$

10128a, 23 lines

```
public class CRT {
    // k es el tamanyo de num[] y rem[].
    // Returns el numero minimo.
    // x tal que:
    // x % num[0] = rem[0],
    // x % num[1] = rem[1],
    // ..................
    // x % num[k-2] = rem[k-1]
    // Asumimos que: Los numeros en num[] son coprimos dos a
        dos (mcd de cada par es 1)
    static int findMinX(int num[], int rem[], int k) {
        // Compute product of all numbers
        int prod = 1;
        for (int i = 0; i < k; i++)
            prod *= num[i];
        // Initialize result
        int result = 0;
        // Apply above formula
        for (int i = 0; i < k; i++) {
            int pp = prod / num[i];
            result += rem[i] * ModInverse.modInverse(pp, num[i
                ]) * pp;
        }
        return result % prod;
    }}
```

5.3.1 Identidad de Bezut

Para $a \neq, b \neq 0$, entonces $d = gcd(a, b)$ es el menor entero positivo para el que hay soluciones enteras de

$$ax + by = d$$

SI (x, y) es una solucion, entonces todas las soluciones enteras vienen dadas por:

$$\left(x + \frac{kb}{\gcd(a,b)}, y - \frac{ka}{\gcd(a,b)}\right), \quad k \in \mathbb{Z}$$

phiFunction.java

Description: *Euler's* ϕ function is defined as $\phi(n) :=$ # of positive integers $\leq n$ that are coprime with n. $\phi(1) = 1$, p prime $\Rightarrow \phi(p^k) = (p-1)p^{k-1}$, m, n coprime $\Rightarrow \phi(mn) = \phi(m)\phi(n)$. If $n = p_1^{k_1} p_2^{k_2} ... p_r^{k_r}$ then $\phi(n) = (p_1 - 1)p_1^{k_1 - 1} ... (p_r - 1)p_r^{k_r - 1}$. $\phi(n) = n \cdot \prod_{p|n}(1 - 1/p)$.

$\sum_{d|n} \phi(d) = n$, $\sum_{1 \leq k \leq n, \gcd(k,n)=1} k = n\phi(n)/2, n > 1$

Euler's thm: a, n coprime $\Rightarrow a^{\phi(n)} \equiv 1 \pmod{n}$.

Fermat's little thm: p prime $\Rightarrow a^{p-1} \equiv 1 \pmod{p}$ $\forall a$.

0e1c3c, 13 lines

```
static int phi(int n)
    int result = n;
    for (int p = 2; p * p <= n; ++p)
    {
        if (n % p == 0){
            while (n % p == 0) n /= p;
            result -= result / p;
        }
    }

    if (n > 1) result -= result / n;
    return result;
}
```

5.4 Ternas Pitagoricas

Las ternas pitagoricas son generadas de forma unica por

$$a = k \cdot (m^2 - n^2), \;\; b = k \cdot (2mn), \;\; c = k \cdot (m^2 + n^2),$$

con $m > n > 0$, $k > 0$, $m \perp n$, ni m o n par.

5.5 Primos

$p = 962592769$ es $2^{21} \mid p - 1$, puede ser util. Para hashing usar 970592641 (31-bit number), 31443539979727 (45-bit), 3006703054056749 (52-bit). Hay 78498 primos menores que 1 000 000.

Raices primitivas existen modulo cualquier potencia prima p^a, excepto para $p = 2, a > 2$, y hay muchos $\phi(\phi(p^a))$. Para $p = 2, a > 2$, el grupo $\mathbb{Z}_{2^a}^{\times}$ es isomorfo a $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$.

5.6 Estimates

$\sum_{d|n} d = O(n \log \log n)$.

EL numero de divisores de n es cercano a 100 para $n < 5e4$, 500 para $n < 1e7$, 2000 para $n < 1e10$, 200 000 para $n < 1e19$.

Combinatorial (6)

6.1 Permutations

6.1.1 Factorial

n	1	2	3	4	5	6	7	8	9	10
$n!$	1	2	6	24	120	720	5040	40320	362880	3628800
n	11	12	13	14	15	16	17			
$n!$	4.0e7	4.8e8	6.2e9	8.7e10	1.3e12	2.1e13	3.6e14			
n	20	25	30	40	50	100	150	171		
$n!$	2e18	2e25	3e32	8e47	3e64	9e157	6e262	>DBL_MAX		

IntPerm.h

Description: Permutation -> integer conversion. (Not order preserving.) Integer -> permutation can use a lookup table.

Time: $\mathcal{O}(n)$

044568, 6 lines

```
int permToInt(vi& v) {
  int use = 0, i = 0, r = 0;
  for(int x:v) r = r * ++i + __builtin_popcount(use & -(1<<x)),
```

```
    use |= 1 << x;                          // (note: minus, not ~!)
  return r;
}
```

6.1.2 Cycles

Let $g_S(n)$ be the number of n-permutations whose cycle lengths all belong to the set S. Then

$$\sum_{n=0}^{\infty} g_S(n)\frac{x^n}{n!} = \exp\left(\sum_{n\in S}\frac{x^n}{n}\right)$$

6.1.3 Derangements

Permutations of a set such that none of the elements appear in their original position.

$$D(n) = (n-1)(D(n-1)+D(n-2)) = nD(n-1)+(-1)^n = \left\lfloor \frac{n!}{e} \right\rceil$$

6.1.4 Burnside's lemma

Given a group G of symmetries and a set X, the number of elements of X *up to symmetry* equals

$$\frac{1}{|G|}\sum_{g\in G}|X^g|,$$

where X^g are the elements fixed by g ($g.x = x$).

If $f(n)$ counts "configurations" (of some sort) of length n, we can ignore rotational symmetry using $G = \mathbb{Z}_n$ to get

$$g(n) = \frac{1}{n}\sum_{k=0}^{n-1} f(\gcd(n,k)) = \frac{1}{n}\sum_{k|n} f(k)\phi(n/k).$$

6.2 Partitions and subsets

6.2.1 Partition function

Number of ways of writing n as a sum of positive integers, disregarding the order of the summands.

$$p(0) = 1,\ p(n) = \sum_{k\in\mathbb{Z}\setminus\{0\}} (-1)^{k+1} p(n-k(3k-1)/2)$$

$$p(n) \sim 0.145/n \cdot \exp(2.56\sqrt{n})$$

n	0	1	2	3	4	5	6	7	8	9	20	50	100
$p(n)$	1	1	2	3	5	7	11	15	22	30	627	~2e5	~2e8

6.2.2 Lucas' Theorem

Let n, m be non-negative integers and p a prime. Write $n = n_k p^k + ... + n_1 p + n_0$ and $m = m_k p^k + ... + m_1 p + m_0$. Then $\binom{n}{m} \equiv \prod_{i=0}^{k}\binom{n_i}{m_i} \pmod p$.

6.2.3 Binomials

multinomial.h

Description: Computes $\binom{k_1+\cdots+k_n}{k_1,k_2,\ldots,k_n} = \frac{(\sum k_i)!}{k_1!k_2!\ldots k_n!}$. a0a312, 6 lines

```
ll multinomial(vi& v) {
  ll c = 1, m = v.empty() ? 1 : v[0];
  rep(i,1,sz(v)) rep(j,0,v[i])
    c = c * ++m / (j+1);
  return c;
}
```

6.3 General purpose numbers

6.3.1 Bernoulli numbers

EGF of Bernoulli numbers is $B(t) = \frac{t}{e^t-1}$ (FFT-able).
$B[0,\ldots] = [1, -\frac{1}{2}, \frac{1}{6}, 0, -\frac{1}{30}, 0, \frac{1}{42}, \ldots]$

Sums of powers:

$$\sum_{i=1}^{n} n^m = \frac{1}{m+1}\sum_{k=0}^{m}\binom{m+1}{k}B_k \cdot (n+1)^{m+1-k}$$

Euler-Maclaurin formula for infinite sums:

$$\sum_{i=m}^{\infty} f(i) = \int_m^\infty f(x)dx - \sum_{k=1}^{\infty}\frac{B_k}{k!}f^{(k-1)}(m)$$
$$\approx \int_m^\infty f(x)dx + \frac{f(m)}{2} - \frac{f'(m)}{12} + \frac{f'''(m)}{720} + O(f^{(5)}(m))$$

6.3.2 Stirling numbers of the first kind

Number of permutations on n items with k cycles.

$$c(n,k) = c(n-1,k-1) + (n-1)c(n-1,k),\ c(0,0) = 1$$
$$\textstyle\sum_{k=0}^{n} c(n,k)x^k = x(x+1)\ldots(x+n-1)$$

$c(8,k) = 8, 0, 5040, 13068, 13132, 6769, 1960, 322, 28, 1$
$c(n,2) = 0, 0, 1, 3, 11, 50, 274, 1764, 13068, 109584, \ldots$

6.3.3 Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k j:s s.t. $\pi(j) > \pi(j+1)$, $k+1$ j:s s.t. $\pi(j) \geq j$, k j:s s.t. $\pi(j) > j$.

$$E(n,k) = (n-k)E(n-1,k-1) + (k+1)E(n-1,k)$$
$$E(n,0) = E(n,n-1) = 1$$
$$E(n,k) = \sum_{j=0}^{k}(-1)^j\binom{n+1}{j}(k+1-j)^n$$

6.3.4 Stirling numbers of the second kind

Partitions of n distinct elements into exactly k groups.

$$S(n,k) = S(n-1,k-1) + kS(n-1,k)$$
$$S(n,1) = S(n,n) = 1$$
$$S(n,k) = \frac{1}{k!}\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}j^n$$

6.3.5 Bell numbers

Total number of partitions of n distinct elements. $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \ldots$. For p prime,

$$B(p^m + n) \equiv mB(n) + B(n+1) \pmod p$$

6.3.6 Labeled unrooted trees

on n vertices: n^{n-2}
on k existing trees of size n_i: $n_1 n_2 \cdots n_k n^{k-2}$
with degrees d_i: $(n-2)!/((d_1-1)!\cdots(d_n-1)!)$

6.3.7 Catalan numbers

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \binom{2n}{n} - \binom{2n}{n+1} = \frac{(2n)!}{(n+1)!n!}$$

$$C_0 = 1,\ C_{n+1} = \frac{2(2n+1)}{n+2}C_n,\ C_{n+1} = \sum C_i C_{n-i}$$

$C_n = 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, \ldots$

- sub-diagonal monotone paths in an $n \times n$ grid.
- strings with n pairs of parenthesis, correctly nested.
- binary trees with with $n+1$ leaves (0 or 2 children).
- ordered trees with $n+1$ vertices.
- ways a convex polygon with $n+2$ sides can be cut into triangles by connecting vertices with straight lines.
- permutations of $[n]$ with no 3-term increasing subseq.

Geometry (7)

7.1 Geometric primitives

Point.h

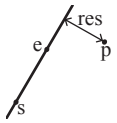
Description: Class to handle points in the plane. T can be e.g. double or long long. (Avoid int.) 47ec0a, 28 lines

```
template <class T> int sgn(T x) { return (x > 0) - (x < 0); }
template<class T>
struct Point {
  typedef Point P;
  T x, y;
  explicit Point(T x=0, T y=0) : x(x), y(y) {}
  bool operator<(P p) const { return tie(x,y) < tie(p.x,p.y); }
  bool operator==(P p) const { return tie(x,y)==tie(p.x,p.y); }
  P operator+(P p) const { return P(x+p.x, y+p.y); }
  P operator-(P p) const { return P(x-p.x, y-p.y); }
  P operator*(T d) const { return P(x*d, y*d); }
  P operator/(T d) const { return P(x/d, y/d); }
  T dot(P p) const { return x*p.x + y*p.y; }
  T cross(P p) const { return x*p.y - y*p.x; }
  T cross(P a, P b) const { return (a-*this).cross(b-*this); }
  T dist2() const { return x*x + y*y; }
  double dist() const { return sqrt((double)dist2()); }
  // angle to x-axis in interval [-pi, pi]
  double angle() const { return atan2(y, x); }
  P unit() const { return *this/dist(); } // makes dist()=1
  P perp() const { return P(-y, x); } // rotates +90 degrees
  P normal() const { return perp().unit(); }
  // returns point rotated 'a' radians ccw around the origin
  P rotate(double a) const {
    return P(x*cos(a)-y*sin(a),x*sin(a)+y*cos(a)); }
```

```
  friend ostream& operator<<(ostream& os, P p) {
    return os << "(" << p.x << "," << p.y << ")"; }
};
```

lineDistance.h

Description:
Returns the signed distance between point p and the line containing points a and b. Positive value on left side and negative on right as seen from a towards b. a==b gives nan. P is supposed to be Point<T> or Point3D<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long. Using Point3D will always give a non-negative distance. For Point3D, call .dist on the result of the cross product.

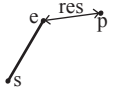


"Point.h" f6bf6b, 4 lines

```
template<class P>
double lineDist(const P& a, const P& b, const P& p) {
  return (double)(b-a).cross(p-a)/(b-a).dist();
}
```

SegmentDistance.h

Description:
Returns the shortest distance between point p and the line segment from point s to e.



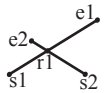
Usage:
```
Point<double> a, b(2,2), p(1,1);
bool onSegment = segDist(a,b,p) < 1e-10;
```

"Point.h" 5c88f4, 6 lines

```
typedef Point<double> P;
double segDist(P& s, P& e, P& p) {
  if (s==e) return (p-s).dist();
  auto d = (e-s).dist2(), t = min(d,max(.0,(p-s).dot(e-s)));
  return ((p-s)*d-(e-s)*t).dist()/d;
}
```

SegmentIntersection.h

Description:
If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common line segment. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or long long.



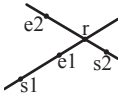
Usage:
```
vector<P> inter = segInter(s1,e1,s2,e2);
if (sz(inter)==1)
cout << "segments intersect at " << inter[0] << endl;
```

"Point.h", "OnSegment.h" 9d57f2, 13 lines

```
template<class P> vector<P> segInter(P a, P b, P c, P d) {
  auto oa = c.cross(d, a), ob = c.cross(d, b),
       oc = a.cross(b, c), od = a.cross(b, d);
  // Checks if intersection is single non-endpoint point.
  if (sgn(oa) * sgn(ob) < 0 && sgn(oc) * sgn(od) < 0)
    return {(a * ob - b * oa) / (ob - oa)};
  set<P> s;
  if (onSegment(c, d, a)) s.insert(a);
  if (onSegment(c, d, b)) s.insert(b);
  if (onSegment(a, b, c)) s.insert(c);
  if (onSegment(a, b, d)) s.insert(d);
  return {all(s)};
}
```

lineIntersection.h

Description:
If a unique intersection point of the lines going through s1,e1 and s2,e2 exists {1, point} is returned. If no intersection point exists {0, (0,0)} is returned and if infinitely many exists {-1, (0,0)} is returned. The wrong position will be returned if P is Point<ll> and the intersection point does not have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow if using int or ll.



Usage:
```
auto res = lineInter(s1,e1,s2,e2);
if (res.first == 1)
cout << "intersection point at " << res.second << endl;
```

"Point.h" a01f81, 8 lines

```
template<class P>
pair<int, P> lineInter(P s1, P e1, P s2, P e2) {
  auto d = (e1 - s1).cross(e2 - s2);
  if (d == 0) // if parallel
    return {-(s1.cross(e1, s2) == 0), P(0, 0)};
  auto p = s2.cross(e1, e2), q = s2.cross(e2, s1);
  return {1, (s1 * p + e1 * q) / d};
}
```

sideOf.h

Description: Returns where p is as seen from s towards e. 1/0/-1 $\Leftrightarrow$ left/on line/right. If the optional argument *eps* is given 0 is returned if p is within distance *eps* from the line. P is supposed to be Point<T> where T is e.g. double or long long. It uses products in intermediate steps so watch out for overflow if using int or long long.

Usage:
```
bool left = sideOf(p1,p2,q)==1;
```

"Point.h" 3af81c, 9 lines

```
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }

template<class P>
int sideOf(const P& s, const P& e, const P& p, double eps) {
  auto a = (e-s).cross(p-s);
  double l = (e-s).dist()*eps;
  return (a > l) - (a < -l);
}
```

OnSegment.h

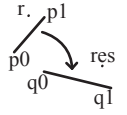
Description: Returns true iff p lies on the line segment from s to e. Use `(segDist(s,e,p)<=epsilon)` instead when using Point<double>.

"Point.h" c597e8, 3 lines

```
template<class P> bool onSegment(P s, P e, P p) {
  return p.cross(s, e) == 0 && (s - p).dot(e - p) <= 0;
}
```

linearTransformation.h

Description:
Apply the linear transformation (translation, rotation and scaling) which takes line p0-p1 to line q0-q1 to point r.



"Point.h" 03a306, 6 lines

```
typedef Point<double> P;
P linearTransformation(const P& p0, const P& p1,
    const P& q0, const P& q1, const P& r) {
  P dp = p1-p0, dq = q1-q0, num(dp.cross(dq), dp.dot(dq));
  return q0 + P((r-p0).cross(num), (r-p0).dot(num))/dp.dist2();
}
```

Angle.h

Description: A class for ordering angles (as represented by int points and a number of rotations around the origin). Useful for rotational sweeping. Sometimes also represents points or vectors.

Usage:
```
vector<Angle> v = {w[0], w[0].t360() ...}; // sorted
int j = 0; rep(i,0,n) { while (v[j] < v[i].t180()) ++j; }
// sweeps j such that (j-i) represents the number of positively
oriented triangles with vertices at 0 and i
```

0f0602, 35 lines

```
struct Angle {
  int x, y;
  int t;
  Angle(int x, int y, int t=0) : x(x), y(y), t(t) {}
  Angle operator-(Angle b) const { return {x-b.x, y-b.y, t}; }
  int half() const {
    assert(x || y);
    return y < 0 || (y == 0 && x < 0);
  }
  Angle t90() const { return {-y, x, t + (half() && x >= 0)}; }
  Angle t180() const { return {-x, -y, t + half()}; }
  Angle t360() const { return {x, y, t + 1}; }
};
bool operator<(Angle a, Angle b) {
  // add a.dist2() and b.dist2() to also compare distances
  return make_tuple(a.t, a.half(), a.y * (ll)b.x) <
         make_tuple(b.t, b.half(), a.x * (ll)b.y);
}

// Given two points, this calculates the smallest angle between
// them, i.e., the angle that covers the defined line segment.
pair<Angle, Angle> segmentAngles(Angle a, Angle b) {
  if (b < a) swap(a, b);
  return (b < a.t180() ?
          make_pair(a, b) : make_pair(b, a.t360()));
}
Angle operator+(Angle a, Angle b) { // point a + vector b
  Angle r(a.x + b.x, a.y + b.y, a.t);
  if (a.t180() < r) r.t--;
  return r.t180() < a ? r.t360() : r;
}
Angle angleDiff(Angle a, Angle b) { // angle b - angle a
  int tu = b.t - a.t; a.t = b.t;
  return {a.x*b.x + a.y*b.y, a.x*b.y - a.y*b.x, tu - (b < a)};
}
```

7.2 Polygons

InsidePolygon.h

Description: Returns true if p lies within the polygon. If strict is true, it returns false for points on the boundary. The algorithm uses products in intermediate steps so watch out for overflow.

Usage:
```
vector<P> v = {P{4,4}, P{1,2}, P{2,1}};
bool in = inPolygon(v, P{3, 3}, false);
```

Time: $\mathcal{O}(n)$

"Point.h", "OnSegment.h", "SegmentDistance.h" 2bf504, 11 lines

```
template<class P>
bool inPolygon(vector<P> &p, P a, bool strict = true) {
  int cnt = 0, n = sz(p);
  rep(i,0,n) {
    P q = p[(i + 1) % n];
    if (onSegment(p[i], q, a)) return !strict;
    //or: if (segDist(p[i], q, a) <= eps) return !strict;
    cnt ^= ((a.y<p[i].y) - (a.y<q.y)) * a.cross(p[i], q) > 0;
  }
  return cnt;
}
```

PolygonArea.h

Description: Returns twice the signed area of a polygon. Clockwise enumeration gives negative area. Watch out for overflow if using int as T!

"Point.h" f12300, 6 lines

```
template<class T>
T polygonArea2(vector<Point<T>>& v) {
```

```
  T a = v.back().cross(v[0]);
  rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);
  return a;
}
```

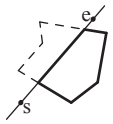
PolygonCenter.h

Description: Returns the center of mass for a polygon.
Time: $\mathcal{O}(n)$

"Point.h" 9706dc, 9 lines

```
typedef Point<double> P;
P polygonCenter(const vector<P>& v) {
  P res(0, 0); double A = 0;
  for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
  }
  return res / A / 3;
}
```

PolygonCut.h



Description:
Returns a vector with the vertices of a polygon with everything to the left of the line going from s to e cut away.
Usage: `vector<P> p = ...;`
`p = polygonCut(p, P(0,0), P(1,0));`

"Point.h", "lineIntersection.h" f2b7d4, 13 lines

```
typedef Point<double> P;
vector<P> polygonCut(const vector<P>& poly, P s, P e) {
  vector<P> res;
  rep(i,0,sz(poly)) {
    P cur = poly[i], prev = i ? poly[i-1] : poly.back();
    bool side = s.cross(e, cur) < 0;
    if (side != (s.cross(e, prev) < 0))
      res.push_back(lineInter(s, e, cur, prev).second);
    if (side)
      res.push_back(cur);
  }
  return res;
}
```

ConvexHull.h

Description:
Returns a vector of the points of the convex hull in counterclockwise order. Points on the edge of the hull between two other points are not considered part of the hull.
Time: $\mathcal{O}(n \log n)$

"Point.h" 310954, 13 lines

```
typedef Point<ll> P;
vector<P> convexHull(vector<P> pts) {
  if (sz(pts) <= 1) return pts;
  sort(all(pts));
  vector<P> h(sz(pts)+1);
  int s = 0, t = 0;
  for (int it = 2; it--; s = --t, reverse(all(pts)))
    for (P p : pts) {
      while (t >= s + 2 && h[t-2].cross(h[t-1], p) <= 0) t--;
      h[t++] = p;
    }
  return {h.begin(), h.begin() + t - (t == 2 && h[0] == h[1])};
}
```

HullDiameter.h

Description: Returns the two points with max distance on a convex hull (ccw, no duplicate/collinear points).

"Point.h" c571b8, 12 lines

```
typedef Point<ll> P;
array<P, 2> hullDiameter(vector<P> S) {
  int n = sz(S), j = n < 2 ? 0 : 1;
  pair<ll, array<P, 2>> res({0, {S[0], S[0]}});
  rep(i,0,j)
    for (;; j = (j + 1) % n) {
      res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});
      if ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) >= 0)
        break;
    }
  return res.second;
}
```

PointInsideHull.h

Description: Determine whether a point t lies inside a convex hull (CCW order, with no collinear points). Returns true if point lies within the hull. If strict is true, points on the boundary aren't included.
Time: $\mathcal{O}(\log N)$

"Point.h", "sideOf.h", "OnSegment.h" 71446b, 14 lines

```
typedef Point<ll> P;

bool inHull(const vector<P>& l, P p, bool strict = true) {
  int a = 1, b = sz(l) - 1, r = !strict;
  if (sz(l) < 3) return r && onSegment(l[0], l.back(), p);
  if (sideOf(l[0], l[a], l[b]) > 0) swap(a, b);
  if (sideOf(l[0], l[a], p) >= r || sideOf(l[0], l[b], p)<= -r)
    return false;
  while (abs(a - b) > 1) {
    int c = (a + b) / 2;
    (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
  }
  return sgn(l[a].cross(l[b], p)) < r;
}
```

Strings (8)

StringMatching.java

Description: Comprueba todas las apariciones de P en T.
Time: $\mathcal{O}(n+m)$

debdbc, 21 lines

```
public class StringMatching {
    char[] T, P; // T = text, P = pattern
    int n, m; // n = length of T, m = length of P
    int [] b; // b = back table0
    void kmpPreprocess() { // call this before calling
        kmpSearch()
        int i = 0, j = -1; b[0] = -1; // starting values
        while (i < m) { // pre-process the pattern string P
            while (j >= 0 && P[i] != P[j]) j = b[j]; // if
                different, reset j using b
            i++; j++; // if same, advance both pointers
            b[i] = j;
        }
    }
    void kmpSearch() { // this is similar as kmpPreprocess(),
        but on string T
        int i = 0, j = 0; // starting values
        while (i < n) { // search through string T
            while (j >= 0 && T[i] != P[j]) j = b[j]; // if
                different, reset j using b
            i++; j++; // if same, advance both pointers
            if (j == m) { // a match found when j == m
                System.out.printf("P is found at index %d in T\
                    n", i - j);
                j = b[j]; // prepare j for the next possible
                    match
            }}}}
```

Trie.java

Description: Trie

be7c4c, 51 lines

```
// Java implementation of search and insert operations
// on Trie
public class Trie {
    // Alphabet size (# of symbols)
    static final int ALPHABET_SIZE = 26;
    // trie node
    static class TrieNode {
        TrieNode[] children = new TrieNode[ALPHABET_SIZE];
        // isEndOfWord is true if the node represents
        // end of a word
        boolean isEndOfWord;
        TrieNode() {
            isEndOfWord = false;
            for (int i = 0; i < ALPHABET_SIZE; i++)
                children[i] = null;
        }
    }
    static TrieNode root;
    // If not present, inserts key into trie
    // If the key is prefix of trie node,
    // just marks leaf node
    static void insert(String key) {
        int level;
        int length = key.length();
        int index;
        TrieNode pCrawl = root;
        for (level = 0; level < length; level++) {
            index = key.charAt(level) - 'a';
            if (pCrawl.children[index] == null)
                pCrawl.children[index] = new TrieNode();

            pCrawl = pCrawl.children[index];
        }
        // mark last node as leaf
        pCrawl.isEndOfWord = true;
    }
    // Returns true if key presents in trie, else false
    static boolean search(String key) {
        int level;
        int length = key.length();
        int index;
        TrieNode pCrawl = root;
        for (level = 0; level < length; level++) {
            index = key.charAt(level) - 'a';
            if (pCrawl.children[index] == null)
                return false;
            pCrawl = pCrawl.children[index];
        }
        return (pCrawl != null && pCrawl.isEndOfWord);
    }
}
```

Others (9)

MergeIntervals.java

Description: Union de Intervalos
Time: $\mathcal{O}(nLog(n))$

1c2569, 28 lines

```
public class MergeIntervals {
    public static void main(String[] args) throws IOException {
        ArrayList<IntPair> al = new ArrayList<>();
        for(int i=0;i<q;i++){ //Extremos de los intervalos
            al.add(new IntPair(Integer.parseInt(st.nextToken())
                ,Integer.parseInt(st.nextToken())));
        } //Ordenar de menor a mayor por el incio del intervalo
        Collections.sort(al);
```

```
        Stack<IntPair> stack = new Stack<>();
        stack.push(al.get(0));
        for(int i=1;i<al.size();i++){
            IntPair top = stack.peek();
            if(top.fini<al.get(i).ini){
                stack.push(al.get(i));
            }
            else if(top.fini<al.get(i).fini){
                top.fini=al.get(i).fini;
                stack.pop();
                stack.push(top);
            }
        }
        int total=0;
        while(!stack.isEmpty()){
            IntPair t= stack.pop();
            total+=(t.fini-t.ini+1);
        }
        System.out.println(total);
    }
}
```

Skyline.java

Description: Dado n edficios encontrar la forma/area del skyline, se devuelven los vertices superior izquierdo hasta el ultimo, que es inferior derecho
Time: $\mathcal{O}(nLog(n))$

6439c5, 91 lines

```
public class Skyline {
    public static List<IntPair> getSkyline(long[][] buildings)
         {
        int n = buildings.length;
        List<IntPair> salida = new ArrayList<IntPair>();
        if (n == 0) return salida;
        if (n == 1) {
            long xStart = buildings[0][0];
            long xEnd = buildings[0][1];
            long y = buildings[0][2];
            salida.add(new IntPair(xStart,y));
            salida.add(new IntPair(xEnd,0));
            return salida;
        }
        List<IntPair> leftSkyline, rightSkyline;
        leftSkyline = getSkyline(Arrays.copyOfRange(buildings,
             0, n / 2));
        rightSkyline = getSkyline(Arrays.copyOfRange(buildings,
              n / 2, n));
        return mergeSkylines(leftSkyline, rightSkyline);
    }

    public static List<IntPair> mergeSkylines(List<IntPair>
         left, List<IntPair> right) {
        long nL = left.size(), nR = right.size();
        int pL = 0, pR = 0;
        long currY = 0, leftY = 0, rightY = 0;
        long x, maxY;
        ArrayList<IntPair> salida = new ArrayList<IntPair>();
        while ((pL < nL) && (pR < nR)) {
            IntPair pointL = left.get(pL);
            IntPair pointR = right.get(pR);
            if (pointL.ini < pointR.ini) {
                x = pointL.ini;
                leftY = pointL.alt;
                pL++;
            }
            else {
                x = pointR.ini;
                rightY = pointR.alt;
                pR++;
            }
            maxY = Math.max(leftY, rightY);
            if (currY != maxY) {
                updateOutput(salida, x, maxY);
                currY = maxY;
            }
        }
        appendSkyline(salida, left, pL, nL, currY);
        appendSkyline(salida, right, pR, nR, currY);
        return salida;
    }

    public static void updateOutput(List<IntPair> output, long
         x, long y) {
        if (output.isEmpty() || output.get(output.size() - 1).
             ini != x)
            output.add(new IntPair(x,y));
        else {
            output.get(output.size() - 1).setAlt(y);
        } }

    public static void appendSkyline(List<IntPair> output, List
         <IntPair> skyline, int p, long n, long currY) {
        while (p < n) {
            IntPair point = skyline.get(p);
            long x = point.ini;
            long y = point.alt;
            p++;
            if (currY != y) {
                updateOutput(output, x, y);
                currY = y;
            }  }  }

    public static void main(String[] args) {
            long [][] skyline = new long[q][3];
            //0 ->Inicio 1->Final 2->Ancho
            List<IntPair> sl = getSkyline(skyline);
            long area=0;
            for(int j=0;j<sl.size()-1;j++){
                long a = sl.get(j).ini;
                long alt = sl.get(j).alt;
                long b = sl.get(j + 1).ini;
                area+=((b-a)*alt);
            }
            System.out.println(area);
        }  }

    public static class IntPair implements Comparable{
        long ini; long alt;

        public IntPair(long i, long a){ ini=i; alt=a; }
        public void setAlt(long alt) { this.alt = alt; }

        @Override
        public int compareTo(Object o) {
            IntPair i = (IntPair) o;
            return (int) (this.ini-i.ini);  } } }
```

Y si no ac? (10)

troubleshoot.txt

62 lines

```
Pre-envio:
Escribe algunos casos de prueba simples, si la muestra no es
     suficiente.
Sabes los limites? Genera los casos maximos.
Esta bien el uso de la memoria?
Podria haber overflow?
Asegurese de enviar el archivo correcto.

Respuesta incorrecta: WA
USA LONG
Imprime tu solucion! Imprime tambien la salida de debug.
Estas limpiando todas las estructuras de datos entre casos de
     prueba?
Puede tu algoritmo manejar todo el rango de entrada?
Vuelve a leer el enunciado completo del problema.
Manejas todos los casos limite correctamente?
Has entendido correctamente el problema?
Codigo copiado incorrecto?
Alguna variable no inicializada?
Algun desbordamiento?
Variables con el mismo nombre?
Recursividad correcta?
Confundir N y M, i y j, etc.?
Estas seguro de que tu algoritmo funciona?
En que casos especiales no has pensado?
Estas seguro de que las funciones STL (Libreria estandar) que
     usas funcionan como crees?
Agrega algunas respuestas a las preguntas, tal vez vuelva a
     enviar.
Crea algunos casos de prueba para ejecutar su algoritmo.
Ve a traves del algoritmo para un caso simple.
Revisa esta lista nuevamente.
Explique su algoritmo a un compadre de equipo.
Pidele al compadre de equipo que mire tu codigo.
Salga a dar un paseillo, p. al posadero.
Es correcto su formato de salida? (incluyendo espacios en
     blanco)
Vuelva a escribir su solucion desde el principio o deje que un
     compadre de equipo lo haga.

Error de ejecucion (RTE):
USA LONG
Has probado todos los casos limites localmente?
Alguna variable no inicializada?
Esta leyendo o escribiendo fuera del rango de cualquier vector?
Alguna llamada que pueda fallar?
Alguna posible division por 0? (modulo 0 por ejemplo)
Alguna recursion infinita posible?
Iteradores invalidos?
Estas usando demasiada memoria?
Depuracion con reenvios (por ejemplo, seniales reasignadas,
     consulte Varios).

Tiempo limite (TLE):
Tienes posibles bucles infinitos?
Cual es la complejidad de su algoritmo?
Estas copiando muchos datos innecesarios? (Referencias)
Que tan grande es la entrada y la salida? (considere buffered
     reader)
Evite el vector, el mapa. (use arrays)
Usas vectores? Cambiar a array.
Que piensan tus compadres de equipo sobre tu algoritmo?

Memoria limite excedida (MLE):
Cual es la cantidad maxima de memoria que su algoritmo deberia
     necesitar?
Esta limpiando todas las estructuras de datos entre casos de
     prueba?

Vida antes que muerte. Fuerza antes que debilidad. Viaje antes
     que destino.
```