



Universidad Rey Juan Carlos

# Teamto de Verano

Cristian Perez, Francisco Tortola, Sergio Salazar

SWERC 2022

2022

Graph (1)

BFS.java

Description: BFS-DFS

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

beb0a6, 18 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);
        visitados[start]=true;
        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;
                } } } }
```

ArticulacionPointsAndBridges.java

Description: Encontrar los puntos de articulacion, puentes o componentes biconexas de un grafo

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

59b08a, 70 lines

```
public class Main {

    private static ArrayList<Integer>[] graf;
    //Se inicializan a 0 con tamaño = cantidad de vertices,
    parents a -1;
    private static int[] low, disc, parents,ap;
    //Tamaño = cantidad de vertices
    private static boolean[] visit;
    //Para puentes
    private static LinkedList<IntPair> bridge;

    private static void checkart(int u, int v, int children,
        int t){
        if(parents[u] == -1 && children>1){
            ap[u]=t;
        }
        if(parents[u] != -1 && low[v]>=disc[u]){
            ap[u]=t;
        }
        if(low[v]>disc[u]){
            bridge.add(new IntPair(Math.min(u,v),Math.max(u,v))
                );
        }
    }

    private static void articulation_and_bridge(int u, int t){
        visit[u]=true;
        disc[u]=t;
        low[u]=t++;
        int children=0;
        for(Integer v: graf[u]){
            if(!visit[v]){
                children++;
                parents[v] = u;
                articulation_and_bridge(v,t);
                low[u]=Math.min(low[u], low[v]);
                checkart (u,v,children,t);
            }
        }
    }
}
```

```
        else if(v!=parents[u]){
            low[u]=Math.min(low[u],disc[v]);
        }
    }
}

public static void main(String[] args) throws IOException {
    graf = new ArrayList[n];

    //Construir grafo

    bridge = new LinkedList<>();
    low= new int[n];
    disc= new int[n];
    parents=new int[n];
    for(int i=0;i<n;i++){
        parents[i]=-1;
    }
    ap=new int[n];
    visit= new boolean[n];

    for(int i=0;i<n;i++){
        if(visit[i]==false) articulation_and_bridge(i,0);
    }
    int art_points=0;
    for(int i=0;i<n;i++){
        if(ap[i]!=0) art_points++;
    }
    int puentes=0;
    for(IntPair k : bridge){
        puentes++;
    }
}
}
```

Dijkstra.java

Description: Shortest Path en un grafo ponderado

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

70d905, 16 lines

```
public class Dijkstra {
    public static void Dijkstra(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
            if(top.d > dist[top.v]) continue;
            for(IntPair aux: graf[top.v]){
                if(dist[top.v]+aux.d >= dist[top.v]) continue;
                dist[aux.v]=dist[top.v]+aux.d;
                pq.offer(new IntPair(aux.v,dist[aux.v]));
            } } }
```

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

```
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;

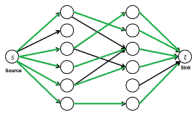
        int size=0;
        for(int i=0;i<graf[u].size();i++){
            int v = graf[u].get(i);
            if(dfs_num[v]==-1){
                size=Math.max(strongConnectedComponents(v),size);
            }
            if(visited[v])
                dfs_low[u]=Math.min(dfs_low[u],dfs_low[v]);
        }
    }
}
```

```
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 con peso 1 (aristas dirigidas), el flujo maximo que llega a end es la cantidad de aristas escogidas.



### 1.2 Math

#### 1.2.1 Number of Spanning Trees

Crea una  $N \times N$  matriz mat, y para cada arista  $a \rightarrow b \in G$ , hacer  $mat[a][b]--$ ,  $mat[b][b]++$  (y  $mat[b][a]--$ ,  $mat[a][a]++$  si  $G$  es no dirigido). Eliminar la  $i$ -esima fila y columna y realizar el determinante; esto entrega el numero de arboles de expansion dirigidos con raiz en  $i$  (if  $G$  es no dirigifo, eliminar cualquier fila/columna).

#### 1.2.2 Teorema de Erdős–Gallai

Un grafo simple con grados  $d_1 \geq \dots \geq d_n$  existe si y solo si  $d_1 + \dots + d_n$  es par y para cada  $k = 1 \dots n$ ,

$$\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k).$$

Written by Anders Sjoqvist and Ulf Lundstrom, 2009 The main sources are: tinyKACTL, Beta and Wikipedia

## 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  $x = -b/2a$ .

$$\begin{matrix} ax + by = e \\ cx + dy = f \end{matrix} \Rightarrow \begin{matrix} x = \frac{ed - bf}{ad - bc} \\ y = \frac{af - ec}{ad - bc} \end{matrix}$$

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

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

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

### 2.3 Trigonometria

$$\begin{aligned} \sin(v + w) &= \sin v \cos w + \cos v \sin w \\ \cos(v + w) &= \cos v \cos w - \sin v \sin w \end{aligned}$$

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

$$\sin v + \sin w = 2 \sin \frac{v + w}{2} \cos \frac{v - w}{2}$$

$$\cos v + \cos w = 2 \cos \frac{v + w}{2} \cos \frac{v - w}{2}$$

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

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

$$\begin{aligned} a \cos x + b \sin x &= r \cos(x - \phi) \\ a \sin x + b \cos x &= r \sin(x + \phi) \end{aligned}$$

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

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

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

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

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:  $\frac{a+b}{a-b} = \frac{\tan \frac{\alpha+\beta}{2}}{\tan \frac{\alpha-\beta}{2}}$

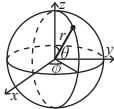
### 2.4.2 Cuadrilateros

Con lados de longitud  $a, b, c, d$ , diagonales  $e, f$ , angulos de la diagonal  $\theta$ , 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  $180^\circ$ ,  $ef = ac + bd$ , y  $A = \sqrt{(p-a)(p-b)(p-c)(p-d)}$ .

### 2.4.3 Coordenadas esfericas



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

## 2.5 Derivadas e Integrales

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

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

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

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

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

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

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

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

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

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

## 2.7 Series

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

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

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

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

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

## 2.8 Probabilidad

Sea  $X$  una variable aleatoria con probabilidad  $p_X(x)$  de tomar el valor  $x$ . Su esperanza es dada por  $\mu = \mathbb{E}(X) = \sum_x x p_X(x)$  y varianza  $\sigma^2 = V(X) = \mathbb{E}(X^2) - (\mathbb{E}(X))^2 = \sum_x (x - \mathbb{E}(X))^2 p_X(x)$  donde  $\sigma$  es la desviacion estandar. Si  $X$  es una varibale continua la funcion de densidad es  $f_X(x)$  y la suma de probabilidades con  $p_X(x)$  es remplazada por la integral con  $f_X(x)$ .

La esperanza es lineal:

$$\mathbb{E}(aX + bY) = a\mathbb{E}(X) + b\mathbb{E}(Y)$$

Para  $X$  e  $Y$  independientes,

$$V(aX + bY) = a^2V(X) + b^2V(Y).$$

### 2.8.1 Distribuciones discretas

#### Binomial distribution

El numero de aciertos  $n$  independientes en experimentos si/no , donde cada acierto tiene una probabilidad de  $p$  es  $\text{Bin}(n, p)$ ,  $n = 1, 2, \dots$ ,  $0 \leq p \leq 1$ .

$$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

$$\mu = np, \sigma^2 = np(1-p)$$

$\text{Bin}(n, p)$  es aproximadamente  $\text{Po}(np)$  para pequeños  $p$ .

#### Distribucion del primer acierto (Geometrica)

El numero de intentos necesarios hasta el primer acierto en experimentos de si/no, donde cada acierto tiene una probabilidad de  $p$  es  $\text{Fs}(p)$ ,  $0 \leq p \leq 1$ .

$$p(k) = p(1-p)^{k-1}, k = 1, 2, \dots$$

$$\mu = \frac{1}{p}, \sigma^2 = \frac{1-p}{p^2}$$

#### Distribucion de Poisson

El numero de eventos ocurridos en un tiempo determinado  $t$  si esos eventos ocurren con una media de  $\kappa$  independientemente del tiempo desde el ultimo suceso es  $\text{Po}(\lambda)$ ,  $\lambda = t\kappa$ .

$$p(k) = e^{-\lambda} \frac{\lambda^k}{k!}, k = 0, 1, 2, \dots$$

$$\mu = \lambda, \sigma^2 = \lambda$$

### 2.8.2 Distribuciones continuas

#### Distribucion uniforme

Si la funcion de densidad es constante entre  $a$  y  $b$  y es 0 fuera de  $U(a, b)$ ,  $a < b$ .

$$f(x) = \begin{cases} \frac{1}{b-a} & a < x < b \\ 0 & \text{otherwise} \end{cases}$$

$$\mu = \frac{a+b}{2}, \sigma^2 = \frac{(b-a)^2}{12}$$

#### Distribucion exponencial

El tiempo entre eventos en un proceso de Poisson es  $\text{Exp}(\lambda)$ ,  $\lambda > 0$ .

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

$$\mu = \frac{1}{\lambda}, \sigma^2 = \frac{1}{\lambda^2}$$

Distribucion normal

Mucho de los sucesos aleatorios reales con media  $\mu$  y varianza  $\sigma^2$  son bien descritos por  $\mathcal{N}(\mu, \sigma^2)$ ,  $\sigma > 0$ .

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

If  $X_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$  and  $X_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$  then

$$aX_1 + bX_2 + c \sim \mathcal{N}(\mu_1 + \mu_2 + c, a^2\sigma_1^2 + b^2\sigma_2^2)$$

2.9 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  $X_1, X_2, \dots$  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 inicial.

Distribucion estacionaria

$\pi$  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  $i$ .  $\pi_j / \pi_i$  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,  $\pi_i$  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).  $\lim_{k \rightarrow \infty} \mathbf{P}^k = \mathbf{1}\pi$ .

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 absorbentes ( $p_{ii} = 1$ ), y tdos los estados en **G** acaban en un estado absorbente 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)

UnionFind.java

Description: Conjuntos de union buscar84e122, 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); } } } }
```

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: 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);
    }
};
```

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 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.hc9b7b0, 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: O(n^2 log(1/epsilon))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((1 + 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:  $p(x) = a[0] * x^0 + \dots + a[n-1] * x^{n-1}$ . For numerical precision, pick  $x[k] = c * \cos(k/(n-1) * \pi), k = 0 \dots n-1$ .  
**Time:**  $\mathcal{O}(n^2)$

```
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;
}
```

### BerlekampMassey.h

**Description:** Recovers any  $n$ -order linear recurrence relation from the first  $2n$  terms of the recurrence. Useful for guessing linear recurrences after brute-forcing the first terms. Should work on any field, but numerical stability for floats is not guaranteed. Output will have size  $\leq n$ .  
**Usage:** berlekampMassey({0, 1, 1, 3, 5, 11}) // {1, 2}  
**Time:**  $\mathcal{O}(N^2)$

```
"/..number-theory/ModPow.h"
vector<ll> berlekampMassey(vector<ll> s) {
    int n = sz(s), L = 0, m = 0;
    vector<ll> C(n), B(n), T;
    C[0] = B[0] = 1;

    ll b = 1;
    rep(i,0,n) { ++m;
        ll d = s[i] % mod;
        rep(j,1,L+1) d = (d + C[j] * s[i - j]) % mod;
        if (!d) continue;
        T = C; ll coef = d * modpow(b, mod-2) % mod;
        rep(j,m,n) C[j] = (C[j] - coef * B[j - m]) % mod;
        if (2 * L > i) continue;
        L = i + 1 - L; B = T; b = d; m = 0;
    }

    C.resize(L + 1); C.erase(C.begin());
    for (ll& x : C) x = (mod - x) % mod;
    return C;
}
```

### LinearRecurrence.h

**Description:** Generates the  $k$ 'th term of an  $n$ -order linear recurrence  $S[i] = \sum_j S[i-j-1]tr[j]$ , given  $S[0 \dots \geq n-1]$  and  $tr[0 \dots n-1]$ . Faster than matrix multiplication. Useful together with Berlekamp–Massey.  
**Usage:** linearRec({0, 1}, {1, 1}, k) // k'th Fibonacci number  
**Time:**  $\mathcal{O}(n^2 \log k)$

```
typedef vector<ll> Poly;
ll linearRec(Poly S, Poly tr, ll k) {
    int n = sz(tr);

    auto combine = [&](Poly a, Poly b) {
        Poly res(n * 2 + 1);
```

```
        rep(i,0,n+1) rep(j,0,n+1)
            res[i + j] = (res[i + j] + a[i] * b[j]) % mod;
        for (int i = 2 * n; i > n; --i) rep(j,0,n)
            res[i - 1 - j] = (res[i - 1 - j] + res[i] * tr[j]) % mod;
        res.resize(n + 1);
        return res;
    };
```

```
Poly pol(n + 1), e(pol);
pol[0] = e[1] = 1;

for (++k; k; k /= 2) {
    if (k % 2) pol = combine(pol, e);
    e = combine(e, e);
}

ll res = 0;
rep(i,0,n) res = (res + pol[i + 1] * S[i]) % mod;
return res;
}
```

## 4.2 Optimization

### GoldenSectionSearch.h

**Description:** Finds the argument minimizing the function  $f$  in the interval  $[a, b]$  assuming  $f$  is unimodal on the interval, i.e. has only one local minimum. The maximum error in the result is  $\epsilon$ ps. Works equally well for maximization with a small change in the code. See TernarySearch.h in the Various chapter for a discrete version.  
**Usage:** double func(double x) { return 4+x+.3\*x\*x; }  
double xmin = gss(-1000,1000,func);  
**Time:**  $\mathcal{O}(\log((b-a)/\epsilon))$

```
double gss(double a, double b, double (*f)(double)) {
    double r = (sqrt(5)-1)/2, eps = 1e-7;
    double x1 = b - r*(b-a), x2 = a + r*(b-a);
    double f1 = f(x1), f2 = f(x2);
    while (b-a > eps)
        if (f1 < f2) { //change to > to find maximum
            b = x2; x2 = x1; f2 = f1;
            x1 = b - r*(b-a); f1 = f(x1);
        } else {
            a = x1; x1 = x2; f1 = f2;
            x2 = a + r*(b-a); f2 = f(x2);
        }
    return a;
}
```

### HillClimbing.h

**Description:** Poor man's optimization for unimodal functions

```
typedef array<double, 2> P;

template<class F> pair<double, P> hillClimb(P start, F f) {
    pair<double, P> cur(f(start), start);
    for (double jmp = 1e9; jmp > 1e-20; jmp /= 2) {
        rep(j,0,100) rep(dx,-1,2) rep(dy,-1,2) {
            P p = cur.second;
            p[0] += dx*jmp;
            p[1] += dy*jmp;
            cur = min(cur, make_pair(f(p), p));
        }
    }
    return cur;
}
```

### Integrate.h

**Description:** Simple integration of a function over an interval using Simpson's rule. The error should be proportional to  $h^4$ , although in practice you will want to verify that the result is stable to desired precision when epsilon changes.

```
4756fc, 7 lines
template<class F>
double quad(double a, double b, F f, const int n = 1000) {
    double h = (b - a) / 2 / n, v = f(a) + f(b);
    rep(i,1,n*2)
        v += f(a + i*h) * (i&1 ? 4 : 2);
    return v * h / 3;
}
```

### IntegrateAdaptive.h

**Description:** Fast integration using an adaptive Simpson's rule.  
**Usage:** double sphereVolume = quad(-1, 1, [](double x) { return quad(-1, 1, [&](double y) { return quad(-1, 1, [&](double z) { return x\*x + y\*y + z\*z < 1; });});});

```
92dd79, 15 lines
typedef double d;
#define S(a,b) (f(a) + 4*f((a+b) / 2) + f(b)) * (b-a) / 6
```

```
template <class F>
d rec(F& f, d a, d b, d eps, d S) {
    d c = (a + b) / 2;
    d S1 = S(a, c), S2 = S(c, b), T = S1 + S2;
    if (abs(T - S) <= 15 * eps || b - a < 1e-10)
        return T + (T - S) / 15;
    return rec(f, a, c, eps / 2, S1) + rec(f, c, b, eps / 2, S2);
}

template<class F>
d quad(d a, d b, F f, d eps = 1e-8) {
    return rec(f, a, b, eps, S(a, b));
}
```

### Simplex.h

**Description:** Solves a general linear maximization problem: maximize  $c^T x$  subject to  $Ax \leq b, x \geq 0$ . Returns -inf if there is no solution, inf if there are arbitrarily good solutions, or the maximum value of  $c^T x$  otherwise. The input vector is set to an optimal  $x$  (or in the unbounded case, an arbitrary solution fulfilling the constraints). Numerical stability is not guaranteed. For better performance, define variables such that  $x = 0$  is viable.  
**Usage:** vvd A = {{1,-1}, {-1,1}, {-1,-2}};  
vd b = {1,1,-4}, c = {-1,-1}, x;  
T val = LPSolver(A, b, c).solve(x);  
**Time:**  $\mathcal{O}(NM * \#pivots)$ , where a pivot may be e.g. an edge relaxation.  
 $\mathcal{O}(2^n)$  in the general case.

```
aa8530, 68 lines
typedef double T; // long double, Rational, double + mod<P>...
typedef vector<T> vd;
typedef vector<vd> vvd;

const T eps = 1e-8, inf = 1/.0;
#define MP make_pair
#define ltj(X) if(s == -1 || MP(X[j],N[j]) < MP(X[s],N[s])) s=j

struct LPSolver {
    int m, n;
    vi N, B;
    vvd D;

    LPSolver(const vvd& A, const vd& b, const vd& c) :
        m(sz(b)), n(sz(c)), N(n+1), B(m), D(m+2, vd(n+2)) {
            rep(i,0,m) rep(j,0,n) D[i][j] = A[i][j];
            rep(i,0,m) { B[i] = n+i; D[i][n] = -1; D[i][n+1] = b[i]; }
            rep(j,0,n) { N[j] = j; D[m][j] = -c[j]; }
            N[n] = -1; D[m+1][n] = 1;
        }
```

```
void pivot(int r, int s) {
    T *a = D[r].data(), inv = 1 / a[s];
    rep(i,0,m+2) if (i != r && abs(D[i][s]) > eps) {
        T *b = D[i].data(), inv2 = b[s] * inv;
        rep(j,0,n+2) b[j] -= a[j] * inv2;
        b[s] = a[s] * inv2;
    }
    rep(j,0,n+2) if (j != s) D[r][j] *= inv;
    rep(i,0,m+2) if (i != r) D[i][s] *= -inv;
    D[r][s] = inv;
    swap(B[r], N[s]);
}

bool simplex(int phase) {
    int x = m + phase - 1;
    for (;;) {
        int s = -1;
        rep(j,0,n+1) if (N[j] != -phase) ltj(D[x]);
        if (D[x][s] >= -eps) return true;
        int r = -1;
        rep(i,0,m) {
            if (D[i][s] <= eps) continue;
            if (r == -1 || MP(D[i][n+1] / D[i][s], B[i])
                < MP(D[r][n+1] / D[r][s], B[r])) r = i;
        }
        if (r == -1) return false;
        pivot(r, s);
    }
}

T solve(vd &x) {
    int r = 0;
    rep(i,1,m) if (D[i][n+1] < D[r][n+1]) r = i;
    if (D[r][n+1] < -eps) {
        pivot(r, n);
        if (!simplex(2) || D[m+1][n+1] < -eps) return -inf;
        rep(i,0,m) if (B[i] == -1) {
            int s = 0;
            rep(j,1,n+1) ltj(D[i]);
            pivot(i, s);
        }
    }
    bool ok = simplex(1); x = vd(n);
    rep(i,0,m) if (B[i] < n) x[B[i]] = D[i][n+1];
    return ok ? D[m][n+1] : inf;
}

};

4.3 Matrices
Determinant.h
Description: Calculates determinant of a matrix. Destroys the matrix.
Time: O(N^3)
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;
}
```

```
IntDeterminant.h
Description: Calculates determinant using modular arithmetics. Modulos
can also be removed to get a pure-integer version.
Time: O(N^3)
const ll mod = 12345;
ll det(vector<vector<ll>>& a) {
    int n = sz(a); ll ans = 1;
    rep(i,0,n) {
        rep(j,i+1,n) {
            while (a[j][i] != 0) { // gcd step
                ll t = a[i][i] / a[j][i];
                if (t) rep(k,i,n)
                    a[i][k] = (a[i][k] - a[j][k] * t) % mod;
                swap(a[i], a[j]);
                ans *= -1;
            }
        }
        ans = ans * a[i][i] % mod;
        if (!ans) return 0;
    }
    return (ans + mod) % mod;
}

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: O(n^2 m)
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)
}
```

```
SolveLinear2.h
Description: To get all uniquely determined values of x back from Solve-
Linear, make the following changes:
"SolveLinear.h"
08e495, 7 lines
rep(j,0,n) if (j != i) // instead of rep(j,i+1,n)
// ... then at the end:
x.assign(m, undefined);
rep(i,0,rank) {
    rep(j,rank,m) if (fabs(A[i][j]) > eps) goto fail;
    x[col[i]] = b[i] / A[i][i];
fail; }

SolveLinearBinary.h
Description: Solves Ax = b over F2. If there are multiple solutions, one is
returned arbitrarily. Returns rank, or -1 if no solutions. Destroys A and b.
Time: O(n^2 m)
fa2d7a, 34 lines
typedef bitset<1000> bs;

int solveLinear(vector<bs>& A, vi& b, bs& x, int m) {
    int n = sz(A), rank = 0, br;
    assert(m <= sz(x));
    vi col(m); iota(all(col), 0);
    rep(i,0,n) {
        for (br=i; br<n; ++br) if (A[br].any()) break;
        if (br == n) {
            rep(j,i,n) if(b[j]) return -1;
            break;
        }
        int bc = (int)A[br]._Find_next(i-1);
        swap(A[i], A[br]);
        swap(b[i], b[br]);
        swap(col[i], col[bc]);
        rep(j,0,n) if (A[j][i] != A[j][bc]) {
            A[j].flip(i); A[j].flip(bc);
        }
        rep(j,i+1,n) if (A[j][i]) {
            b[j] ^= b[i];
            A[j] ^= A[i];
        }
        rank++;
    }

    x = bs();
    for (int i = rank; i--;) {
        if (!b[i]) continue;
        x[col[i]] = 1;
        rep(j,0,i) b[j] ^= A[j][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) (mod p^k) where A^-1 starts
as the inverse of A mod p, and k is doubled in each step.
Time: O(n^3)
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;
}
```

### MatrixInverse-mod.h

**Description:** Invert matrix  $A$  modulo a prime. Returns rank; result is stored in  $A$  unless singular (rank < n). 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.

|                                 |                  |
|---------------------------------|------------------|
| <b>Time:</b> $\mathcal{O}(n^3)$ | a6f68f, 36 lines |
|---------------------------------|------------------|

```
int matInv(vector<vector<ll>>& A) {
    int n = sz(A); vi col(n);
    vector<vector<ll>> tmp(n, vector<ll>(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 (A[j][k]) {
            r = j; c = k; goto found;
        }
        return i;
    }
found:
    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]);
    ll v = modpow(A[i][i], mod - 2);
    rep(j,i+1,n) {
        ll f = A[j][i] * v % mod;
        A[j][i] = 0;
        rep(k,i+1,n) A[j][k] = (A[j][k] - f*A[i][k]) % mod;
        rep(k,0,n) tmp[j][k] = (tmp[j][k] - f*tmp[i][k]) % mod;
    }
    rep(j,i+1,n) A[i][j] = A[i][j] * v % mod;
    rep(j,0,n) tmp[i][j] = tmp[i][j] * v % mod;
    A[i][i] = 1;
}

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

rep(i,0,n) rep(j,0,n)
```

```
    A[col[i]][col[j]] = tmp[i][j] % mod + (tmp[i][j] < 0 ? mod : 0);
    return n;
}
```

### Tridiagonal.h

**Description:**  $x = \text{tridiagonal}(d, p, q, b)$  solves the equation system

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{n-1} \end{pmatrix} = \begin{pmatrix} d_0 & p_0 & 0 & 0 & \cdots & 0 \\ q_0 & d_1 & p_1 & 0 & \cdots & 0 \\ 0 & q_1 & d_2 & p_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & q_{n-3} & d_{n-2} & p_{n-2} \\ 0 & 0 & \cdots & 0 & q_{n-2} & d_{n-1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \end{pmatrix}.$$

This is useful for solving problems on the type

$$a_i = b_i a_{i-1} + c_i a_{i+1} + d_i, 1 \leq i \leq n,$$

where  $a_0, a_{n+1}, b_i, c_i$  and  $d_i$  are known.  $a$  can then be obtained from

$$\{a_i\} = \text{tridiagonal}(\{1, -1, -1, \dots, -1, 1\}, \{0, c_1, c_2, \dots, c_n\}, \{b_1, b_2, \dots, b_n, 0\}, \{a_0, d_1, d_2, \dots, d_n, a_{n+1}\}).$$

Fails if the solution is not unique.  
If  $|d_i| > |p_i| + |q_{i-1}|$  for all  $i$ , or  $|d_i| > |p_{i-1}| + |q_i|$ , or the matrix is positive definite, the algorithm is numerically stable and neither `tr` nor the check for `diag[i] == 0` is needed.

|                               |                  |
|-------------------------------|------------------|
| <b>Time:</b> $\mathcal{O}(N)$ | 8f9fa8, 26 lines |
|-------------------------------|------------------|

```
typedef double T;
vector<T> tridiagonal(vector<T> diag, const vector<T>& super,
    const vector<T>& sub, vector<T> b) {
    int n = sz(b); vi tr(n);
    rep(i,0,n-1) {
        if (abs(diag[i]) < 1e-9 * abs(super[i])) { // diag[i] == 0
            b[i+1] -= b[i] * diag[i+1] / super[i];
            if (i+2 < n) b[i+2] -= b[i] * sub[i+1] / super[i];
            diag[i+1] = sub[i]; tr[++i] = 1;
        } else {
            diag[i+1] -= super[i]*sub[i]/diag[i];
            b[i+1] -= b[i]*sub[i]/diag[i];
        }
    }
    for (int i = n; i--;) {
        if (tr[i]) {
            swap(b[i], b[i-1]);
            diag[i-1] = diag[i];
            b[i] /= super[i-1];
        } else {
            b[i] /= diag[i];
            if (i) b[i-1] -= b[i]*super[i-1];
        }
    }
    return b;
}
```

## 4.4 Fourier transforms

### FastFourierTransform.h

**Description:** `fft(a)` computes  $\hat{f}(k) = \sum_x a[x] \exp(2\pi i \cdot kx/N)$  for all  $k$ .  $N$  must be a power of 2. Useful for convolution: `conv(a, b) = c`, where  $c[x] = \sum a[i]b[x-i]$ . For convolution of complex numbers or more than two vectors: FFT, multiply pointwise, divide by `n`, reverse(`start+1, end`), FFT back. Rounding is safe if  $(\sum a_i^2 + \sum b_i^2) \log_2 N < 9 \cdot 10^{14}$  (in practice  $10^{16}$ ; higher for random inputs). Otherwise, use NTT/FFTMod.  
**Time:**  $\mathcal{O}(N \log N)$  with  $N = |A| + |B|$  ( $\sim 1s$  for  $N = 2^{22}$ )

```
typedef complex<double> C;
typedef vector<double> vd;
void fft(vector<C>& a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
```

```
static vector<complex<long double>> R(2, 1);
static vector<C> rt(2, 1); // (^ 10% faster if double)
for (static int k = 2; k < n; k *= 2) {
    R.resize(n); rt.resize(n);
    auto x = polar(1.0L, acos(-1.0L) / k);
    rep(i,k,2*k) rt[i] = R[i] = i&1 ? R[i/2] * x : R[i/2];
}
vi rev(n);
rep(i,0,n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
rep(i,0,n) if (i < rev[i]) swap(a[i], a[rev[i]]);
for (int k = 1; k < n; k *= 2)
    for (int i = 0; i < n; i += 2 * k) rep(j,0,k) {
        C z = rt[j+k] * a[i+j+k]; // (25% faster if hand-rolled)
        a[i + j + k] = a[i + j] - z;
        a[i + j] += z;
    }
}

vd conv(const vd& a, const vd& b) {
    if (a.empty() || b.empty()) return {};
    vd res(sz(a) + sz(b) - 1);
    int L = 32 - __builtin_clz(sz(res)), n = 1 << L;
    vector<C> in(n), out(n);
    copy(all(a), begin(in));
    rep(i,0,sz(b)) in[i].imag(b[i]);
    fft(in);
    for (C& x : in) x *= x;
    rep(i,0,n) out[i] = in[-i & (n - 1)] - conj(in[i]);
    fft(out);
    rep(i,0,sz(res)) res[i] = imag(out[i]) / (4 * n);
    return res;
}
```

### FastFourierTransformMod.h

**Description:** Higher precision FFT, can be used for convolutions modulo arbitrary integers as long as  $N \log_2 N \cdot \text{mod} < 8.6 \cdot 10^{14}$  (in practice  $10^{16}$  or higher). Inputs must be in  $[0, \text{mod})$ .

|  |                  |
|--|------------------|
| <b>Time:</b> $\mathcal{O}(N \log N)$ , where $N =  A  +  B $ (twice as slow as NTT or FFT) | b82773, 22 lines |
|--|------------------|

```
FastFourierTransform.h
typedef vector<ll> vl;
template<int M> vl convMod(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    vl res(sz(a) + sz(b) - 1);
    int B=32-__builtin_clz(sz(res)), n=1<<B, cut=int(sqrt(M));
    vector<C> L(n), R(n), outs(n), outl(n);
    rep(i,0,sz(a)) L[i] = C((int)a[i] / cut, (int)a[i] % cut);
    rep(i,0,sz(b)) R[i] = C((int)b[i] / cut, (int)b[i] % cut);
    fft(L), fft(R);
    rep(i,0,n) {
        int j = -i & (n - 1);
        outl[j] = (L[i] + conj(L[j])) * R[i] / (2.0 * n);
        outs[j] = (L[i] - conj(L[j])) * R[i] / (2.0 * n) / 1i;
    }
    fft(outl), fft(outs);
    rep(i,0,sz(res)) {
        ll av = ll(real(outl[i])+.5), cv = ll(imag(outs[i])+.5);
        ll bv = ll(imag(outl[i])+.5) + ll(real(outs[i])+.5);
        res[i] = ((av % M * cut + bv) % M * cut + cv) % M;
    }
    return res;
}
```

### NumberTheoreticTransform.h



**Description:**  $\text{ntt}(a)$  computes  $\hat{f}(k) = \sum_x a[x]g^{xk}$  for all  $k$ , where  $g = \text{root}^{(mod-1)/N}$ .  $N$  must be a power of 2. Useful for convolution modulo specific nice primes of the form  $2^a b + 1$ , where the convolution result has size at most  $2^a$ . For arbitrary modulo, see FFTMod.  $\text{conv}(a, b) = c$ , where  $c[x] = \sum a[i]b[x - i]$ . For manual convolution: NTT the inputs, multiply pointwise, divide by  $n$ , reverse(start+1, end), NTT back. Inputs must be in  $[0, \text{mod})$ .

Time:  $\mathcal{O}(N \log N)$

../number-theory/ModPow.hced03d, 33 lines

```
const ll mod = (119 << 23) + 1, root = 62; // = 998244353
// For p < 2^30 there is also e.g. 5 << 25, 7 << 26, 479 << 21
// and 483 << 21 (same root). The last two are > 10^9.
typedef vector<ll> vl;
void ntt(vl &a) {
    int n = sz(a), L = 31 - __builtin_clz(n);
    static vl rt(2, 1);
    for (static int k = 2, s = 2; k < n; k *= 2, s++) {
        rt.resize(n);
        ll z[] = {1, modpow(root, mod >> s)};
        rep(i, k, 2*k) rt[i] = rt[i / 2] * z[i & 1] % mod;
    }
    vi rev(n);
    rep(i, 0, n) rev[i] = (rev[i / 2] | (i & 1) << L) / 2;
    rep(i, 0, n) if (i < rev[i]) swap(a[i], a[rev[i]]);
    for (int k = 1; k < n; k *= 2)
        for (int i = 0; i < n; i += 2 * k) rep(j, 0, k) {
            ll z = rt[j + k] * a[i + j + k] % mod, &ai = a[i + j];
            a[i + j + k] = ai - z + (z > ai ? mod : 0);
            ai += (ai + z >= mod ? z - mod : z);
        }
}
vl conv(const vl &a, const vl &b) {
    if (a.empty() || b.empty()) return {};
    int s = sz(a) + sz(b) - 1, B = 32 - __builtin_clz(s), n = 1
        << B;
    int inv = modpow(n, mod - 2);
    vl L(a), R(b), out(n);
    L.resize(n), R.resize(n);
    ntt(L), ntt(R);
    rep(i, 0, n) out[-i & (n - 1)] = (ll)L[i] * R[i] % mod * inv %
        mod;
    ntt(out);
    return {out.begin(), out.begin() + s};
}
```

FastSubsetTransform.h

**Description:** Transform to a basis with fast convolutions of the form  $c[z] = \sum_{z=x \oplus y} a[x] \cdot b[y]$ , where  $\oplus$  is one of AND, OR, XOR. The size of  $a$  must be a power of two.

Time:  $\mathcal{O}(N \log N)$

464cf3, 16 lines

```
void FST(vi& a, bool inv) {
    for (int n = sz(a), step = 1; step < n; step *= 2) {
        for (int i = 0; i < n; i += 2 * step) rep(j, i, i+step) {
            int &u = a[j], &v = a[j + step]; tie(u, v) =
                inv ? pii(v - u, u) : pii(v, u + v); // AND
                inv ? pii(v, u - v) : pii(u + v, u); // OR
                pii(u + v, u - v); // XOR
        }
    }
    if (inv) for (int& x : a) x /= sz(a); // XOR only
}
vi conv(vi a, vi b) {
    FST(a, 0); FST(b, 0);
    rep(i, 0, sz(a)) a[i] *= b[i];
    FST(a, 1); return a;
}
```

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

static int BITS=10;

//Si todos los numeros son menores a 2^k BITS=64-k;

static long po = 1 << BITS;

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  $a * x = 1 \bmod m$  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.  $\text{modLog}(a, 1, m)$  can be used to calculate the order of  $a$ .

Time:  $\mathcal{O}(\sqrt{m})$

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.  
 $\text{modsum}(\text{to}, c, k, m) = \sum_{i=0}^{\text{to}-1} (ki + c) \% m$ .  $\text{divsum}$  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:**  $\mathcal{O}(\log^2 p)$  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 |
| <pre>public class SieveOfEratosthenes {     static LinkedList&lt;Integer&gt; sieveOfErastosthenes(int n){         boolean prime[] = new boolean[n+1];         LinkedList&lt;Integer&gt;out = new LinkedList&lt;&gt;();         for(int p=2; p*p&lt;=n; p++)             if(!prime[p]) {                 for (int i = p * p; i &lt;= 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&lt;=n; i++)             if(!prime[i])                 out.add(i);         return out;     } }</pre> |                  |

SieveOfErastosthenesFast.java

**Description:** Generar primos hasta cierto límite

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

|  |                  |
|--|------------------|
|  | be8dd0, 37 lines |
| <pre>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&lt;Boolean&gt;isprime = new Vector&lt;&gt;(MAX_SIZE);     static Vector&lt;Integer&gt;prime = new Vector&lt;&gt;();     static Vector&lt;Integer&gt;SPF = new Vector&lt;&gt;(MAX_SIZE);     // method generate all prime number less then N in O(n)     static void manipulated_seive(int N) {         for (int i = 0; i &lt;= 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&lt;=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 &lt; prime.size() &amp;&amp; i * prime.get(j)                 ) &lt; N &amp;&amp; prime.get(j) &lt;= SPF.get(i); j++) {                 isprime.set(i * prime.get(j), false);                 // put smallest prime factor of i*prime[j]</pre> |                  |

```
        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  $7 \cdot 10^{*18}$

**Usage:** ModMulLL.java

**Time:** 7 veces la complejidad de  $a^b \bmod c$

|   |                  |
|---|------------------|
|   | e3c2be, 21 lines |
| <pre>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 &lt;= 1    n == 4)    return false;     if (n &lt;= 3)    return true;     int d = n - 1;     while (d % 2 == 0) d /= 2;     for (int i = 0; i &lt; k; i++) if (!miillerTest(d, n))         return false;     return true; }</pre> |                  |

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 |
| <pre>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&lt;ull&gt; 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; }</pre> |                  |

## 5.3 Divisibility

Euclid.java

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

|  |                  |
|--|------------------|
|  | 6aba01, 11 lines |
| <pre>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;     }</pre> |                  |

```
    }
    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 |
| <pre>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 &lt; k; i++)             prod *= num[i];         // Initialize result         int result = 0;         // Apply above formula         for (int i = 0; i &lt; k; i++) {             int pp = prod / num[i];             result += rem[i] * ModInverse.modInverse(pp, num[i]                 ) * pp;         }         return result % prod;     } }</pre> |                  |

### 5.3.1 Identidad de Bezut

Para  $a \neq 0, b \neq 0$ , 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  $\phi$  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 |
| <pre>static int phi(int n) {     int result = n;     for (int p = 2; p * p &lt;= n; ++p)     {         if (n % p == 0){             while (n % p == 0) n /= p;             result -= result / p;         }     } }</pre> |                  |

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

### 5.4 Fractions

**ContinuedFractions.h**  
**Description:** Given  $N$  and a real number  $x \geq 0$ , finds the closest rational approximation  $p/q$  with  $p, q \leq N$ . It will obey  $|p/q - x| \leq 1/qN$ .  
For consecutive convergents,  $p_{k+1}q_k - q_{k+1}p_k = (-1)^k$ . ( $p_k/q_k$  alternates between  $> x$  and  $< x$ .) If  $x$  is rational,  $y$  eventually becomes  $\infty$ ; if  $x$  is the root of a degree 2 polynomial the  $a$ 's eventually become cyclic.  
**Time:**  $\mathcal{O}(\log N)$

```
dd6c5e, 21 lines
typedef double d; // for N ~ 1e7; long double for N ~ 1e9
pair<ll, ll> approximate(d x, ll N) {
    ll LP = 0, LQ = 1, P = 1, Q = 0, inf = LLONG_MAX; d y = x;
    for (;;) {
        ll lim = min(P ? (N-LP) / P : inf, Q ? (N-LQ) / Q : inf),
            a = (ll)floor(y), b = min(a, lim),
            NP = b*P + LP, NQ = b*Q + LQ;
        if (a > b) {
            // If b > a/2, we have a semi-convergent that gives us a
            // better approximation; if b = a/2, we *may* have one.
            // Return {P, Q} here for a more canonical approximation.
            return (abs(x - (d)NP / (d)NQ) < abs(x - (d)P / (d)Q)) ?
                make_pair(NP, NQ) : make_pair(P, Q);
        }
        if (abs(y = 1/(y - (d)a)) > 3*N) {
            return {NP, NQ};
        }
        LP = P; P = NP;
        LQ = Q; Q = NQ;
    }
}
```

### FracBinarySearch.h

**Description:** Given  $f$  and  $N$ , finds the smallest fraction  $p/q \in [0, 1]$  such that  $f(p/q)$  is true, and  $p, q \leq N$ . You may want to throw an exception from  $f$  if it finds an exact solution, in which case  $N$  can be removed.  
**Usage:** fracBS([f](Frac f) { return f.p>=3\*f.q; }, 10); // {1,3}  
**Time:**  $\mathcal{O}(\log(N))$

```
27ab3e, 25 lines
struct Frac { ll p, q; };

template<class F>
Frac fracBS(F f, ll N) {
    bool dir = 1, A = 1, B = 1;
    Frac lo{0, 1}, hi{1, 1}; // Set hi to 1/0 to search (0, N]
    if (f(lo)) return lo;
    assert(f(hi));
    while (A || B) {
        ll adv = 0, step = 1; // move hi if dir, else lo
        for (int si = 0; step; (step *= 2) >>= si) {
            adv += step;
            Frac mid{lo.p * adv + hi.p, lo.q * adv + hi.q};
            if (abs(mid.p) > N || mid.q > N || dir == !f(mid)) {
                adv -= step; si = 2;
            }
        }
        hi.p += lo.p * adv;
        hi.q += lo.q * adv;
        dir = !dir;
        swap(lo, hi);
        A = B; B = !!adv;
    }
    return dir ? hi : lo;
}
```

## ContinuedFractions FracBinarySearch IntPerm

### 5.5 Ternas Pitagoricas

Las ternas pitagoricas son generadas de forma unica por

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

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

### 5.6 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  $\mathbb{Z}_2 \times \mathbb{Z}_{2^{a-2}}$ .

### 5.7 Estimates

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

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

### 5.8 Mobius Function

$$\mu(n) = \begin{cases} 0 & n \text{ no tiene factores primos repetidos} \\ 1 & n \text{ tiene un numero par de factores primos} \\ -1 & n \text{ tiene un numero impar de factores primos} \end{cases}$$

Mobius Inversion:

$$g(n) = \sum_{d \mid n} f(d) \Leftrightarrow f(n) = \sum_{d \mid n} \mu(d) g(n/d)$$

Otras formulas utiles:  
La suma sobre todos los divisores positivos de  $n$  de la función de Mobius es cero excepto cuando  $n = 1$ .  $\sum_{d \mid n} \mu(d) = [n = 1]$  (Muy util)

$$g(n) = \sum_{m \mid d} f(d) \Leftrightarrow f(n) = \sum_{m \mid d} \mu(d/n) g(d)$$

$$g(n) = \sum_{1 \leq m \leq n} f(\lfloor \frac{n}{m} \rfloor) \Leftrightarrow f(n) = \sum_{1 \leq m \leq n} \mu(m) g(\lfloor \frac{n}{m} \rfloor)$$

## 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.

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

#### 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

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

### 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.

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

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

| $n$    | 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 | $\sim 2e5$ | $\sim 2e8$ |

#### 6.2.2 Lucas’ Theorem

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

6.2.3 Binomials

multinomial.h

**Description:** Computes  $\binom{k_1 + \dots + k_n}{k_1, k_2, \dots, k_n} = \frac{(\sum k_i)!}{k_1!k_2! \dots k_n!}$ .

---

```
11 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;
}
```

a0a312, 6 lines

6.3 General purpose numbers

6.3.1 Bernoulli numbers

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

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.

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

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

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.  $B(n) = 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, \dots$  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  $n_i$ :  $n_1 n_2 \dots n_k n^{k-2}$   
# with degrees  $d_i$ :  $(n-2)! / ((d_1-1)! \dots (d_n-1)!)$

6.3.7 Catalan numbers

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

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

- 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.)

---

```
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)); }
}
```

47ec0a, 28 lines

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

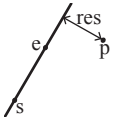
lineDistance.h

**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

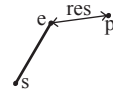
**Description:**  
Returns the shortest distance between point p and the line segment from point s to e.

**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

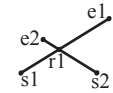
**Description:**  
If a unique intersection point between the line segments going from s1 to e1 and from s2 to e2 exists then it is returned. If no intersection point exists an empty vector is returned. If infinitely many exist a vector with 2 elements is returned, containing the endpoints of the common 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;

---

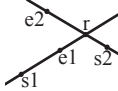
```
"Point.h", "OnSegment.h" 9d57f2, 13 lines

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;



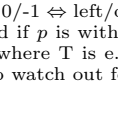
"Point.h"

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

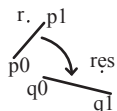
```
template<class P>
int sideOf(P s, P e, P p) { return sgn(s.cross(e, p)); }
```

### 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>.



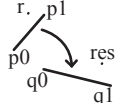
"Point.h"

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.



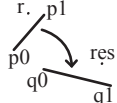
"Point.h"

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();
}
```

### LineProjectionReflection.h

**Description:** Projects point p onto line ab. Set refl=true to get reflection of point p across line ab insted. The wrong point will be returned if P is an integer point and the desired point doesn't have integer coordinates. Products of three coordinates are used in intermediate steps so watch out for overflow.



"Point.h"

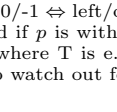
b5562d, 5 lines

```
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
    P v = b - a;
    return p - v.perp()*(1+refl)*v.cross(p-a)/v.dist2();
}
```

```
template<class P>
P lineProj(P a, P b, P p, bool refl=false) {
    P v = b - a;
    return p - v.perp()*(1+refl)*v.cross(p-a)/v.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



"Point.h"

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 * (1l)b.x) <
        make_tuple(b.t, b.half(), a.x * (1l)b.y);
}
```

```
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 * (1l)b.x) <
        make_tuple(b.t, b.half(), a.x * (1l)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 Circles

### CircleIntersection.h

**Description:** Computes the pair of points at which two circles intersect. Returns false in case of no intersection.

"Point.h"

84d6d3, 11 lines

```
typedef Point<double> P;
bool circleInter(P a,P b,double r1,double r2,pair<P, P>* out) {
    if (a == b) { assert(r1 != r2); return false; }
    P vec = b - a;
    double d2 = vec.dist2(), sum = r1+r2, dif = r1-r2,
        p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
    if (sum*sum < d2 || dif*dif > d2) return false;
    P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
    *out = {mid + per, mid - per};
    return true;
}
```

### CircleTangents.h

**Description:** Finds the external tangents of two circles, or internal if r2 is negated. Can return 0, 1, or 2 tangents – 0 if one circle contains the other (or overlaps it, in the internal case, or if the circles are the same); 1 if the circles are tangent to each other (in which case .first = .second and the tangent line is perpendicular to the line between the centers). .first and .second give the tangency points at circle 1 and 2 respectively. To find the tangents of a circle with a point set r2 to 0.

"Point.h"

b0153d, 13 lines

```
template<class P>
vector<pair<P, P>> tangents(P c1, double r1, P c2, double r2) {
    P d = c2 - c1;
    double dr = r1 - r2, d2 = d.dist2(), h2 = d2 - dr * dr;
    if (d2 == 0 || h2 < 0) return {};
    vector<pair<P, P>> out;
    for (double sign : {-1, 1}) {
        P v = (d * dr + d.perp() * sqrt(h2) * sign) / d2;
        out.push_back({c1 + v * r1, c2 + v * r2});
    }
    if (h2 == 0) out.pop_back();
    return out;
}
```

### CircleLine.h

**Description:** Finds the intersection between a circle and a line. Returns a vector of either 0, 1, or 2 intersection points. P is intended to be Point<double>.

"Point.h"

e0cfba, 9 lines

```
template<class P>
vector<P> circleLine(P c, double r, P a, P b) {
    P ab = b - a, p = a + ab * (c-a).dot(ab) / ab.dist2();
    double s = a.cross(b, c), h2 = r*r - s*s / ab.dist2();
    if (h2 < 0) return {};
    if (h2 == 0) return {p};
    P h = ab.unit() * sqrt(h2);
    return {p - h, p + h};
}
```

### CirclePolygonIntersection.h

**Description:** Returns the area of the intersection of a circle with a ccw polygon.  
**Time:**  $\mathcal{O}(n)$

"../content/geometry/Point.h"

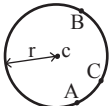
alee63, 19 lines

```
typedef Point<double> P;
#define arg(p, q) atan2(p.cross(q), p.dot(q))
double circlePoly(P c, double r, vector<P> ps) {
    auto tri = [&](P p, P q) {
        auto r2 = r * r / 2;
        P d = q - p;
        auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
        auto det = a * a - b;
        if (det <= 0) return arg(p, q) * r2;
        auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
        if (t < 0 || 1 <= s) return arg(p, q) * r2;
        P u = p + d * s, v = p + d * t;
        return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
    };
    auto sum = 0.0;
    rep(i,0,sz(ps))
        sum += tri(ps[i] - c, ps[(i + 1) % sz(ps)] - c);
    return sum;
}
```

circumcircle.h

**Description:**

The circumcirle of a triangle is the circle intersecting all three vertices. ccRadius returns the radius of the circle going through points A, B and C and ccCenter returns the center of the same circle.



|  |                 |
|--|-----------------|
| "Point.h"  | 1caa3a, 9 lines |
| <pre> <b>typedef</b> Point&lt;<b>double</b>&gt; P; <b>double</b> ccRadius(<b>const</b> P&amp; A, <b>const</b> P&amp; B, <b>const</b> P&amp; C) {     <b>return</b> (B-A).dist()*<b>(C-B).dist()*(A-C).dist()</b> /         abs <b>(</b>(B-A).cross(C-A)<b>)</b> / 2; }  P ccCenter(<b>const</b> P&amp; A, <b>const</b> P&amp; B, <b>const</b> P&amp; C) {     P b = C-A, c = B-A;     <b>return</b> A + (b*c.dist2()<b>-c</b>*b.dist2()<b>)</b>.perp()<b>/b</b>.cross(c) / 2; } </pre> |                 |

MinimumEnclosingCircle.h

**Description:** Computes the minimum circle that encloses a set of points.

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

|   |                  |
|---|------------------|
| "circumcircle.h"  | 09dd0a, 17 lines |
| <pre> pair&lt;P, <b>double</b>&gt; mec(vector&lt;P&gt; ps) {     shuffle(all(ps), mt19937(time(0)));     P o = ps[0];     <b>double</b> r = 0, EPS = 1 + 1e-8;     rep(i,0,sz(ps)) <b>if</b> ((o - ps[i]).dist() &gt; r * EPS) {         o = ps[i], r = 0;         rep(j,0,i) <b>if</b> ((o - ps[j]).dist() &gt; r * EPS) {             o = (ps[i] + ps[j]) / 2;             r = (o - ps[i]).dist();             rep(k,0,j) <b>if</b> ((o - ps[k]).dist() &gt; r * EPS) {                 o = ccCenter(ps[i], ps[j], ps[k]);                 r = (o - ps[i]).dist();             }         }     }     <b>return</b> {o, r}; } </pre> |                  |

7.3 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 |
| <pre> <b>template</b>&lt;<b>class</b> P&gt; <b>bool</b> inPolygon(vector&lt;P&gt; &amp;p, P a, <b>bool</b> strict = <b>true</b>) {     <b>int</b> cnt = 0, n = sz(p);     rep(i,0,n) {         P q = p[(i + 1) % n];         <b>if</b> (onSegment(p[i], q, a)) <b>return</b> !strict;         //or: <i>if (segDist(p[i], q, a) &lt;= eps) return !strict;</i>         cnt ^= ((a.y&lt;p[i].y) - (a.y&lt;q.y)) * a.cross(p[i], q) &gt; 0;     }     <b>return</b> cnt; } </pre> |                  |

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 |
| <pre> <b>template</b>&lt;<b>class</b> T&gt; T polygonArea2(vector&lt;Point&lt;T&gt;&gt;&amp; v) { </pre> |                 |

|   |  |
|---|--|
| <pre>     T a = v.back().cross(v[0]);     rep(i,0,sz(v)-1) a += v[i].cross(v[i+1]);     <b>return</b> a; } </pre> |  |
|---|--|

PolygonCenter.h

**Description:** Returns the center of mass for a polygon.

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

|  |                 |
|--|-----------------|
| "Point.h"  | 9706dc, 9 lines |
| <pre> <b>typedef</b> Point&lt;<b>double</b>&gt; P; P polygonCenter(<b>const</b> vector&lt;P&gt;&amp; v) {     P res(0, 0); <b>double</b> A = 0;     <b>for</b> (<b>int</b> i = 0, j = sz(v) - 1; i &lt; sz(v); j = i++) {         res = res + (v[i] + v[j]) * v[j].cross(v[i]);         A += v[j].cross(v[i]);     }     <b>return</b> res / A / 3; } </pre> |                 |

PolygonCut.h

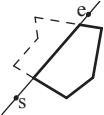
**Description:**

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



PolygonUnion.h

**Description:** Calculates the area of the union of  $n$  polygons (not necessarily convex). The points within each polygon must be given in CCW order. (Epsilon checks may optionally be added to sideOf/sgn, but shouldn't be needed.)

**Time:**  $\mathcal{O}(N^2)$ , where  $N$  is the total number of points

|   |                  |
|---|------------------|
| "Point.h", "sideOf.h"   | 3931c6, 33 lines |
| <pre> <b>typedef</b> Point&lt;<b>double</b>&gt; P; <b>double</b> rat(P a, P b) { <b>return</b> sgn(b.x) ? a.x/b.x : a.y/b.y; } <b>double</b> polyUnion(vector&lt;vector&lt;P&gt;&gt;&amp; poly) {     <b>double</b> ret = 0;     rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {         P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];         vector&lt;pair&lt;<b>double</b>, <b>int</b>&gt;&gt; segs = {{0, 0}, {1, 0}};         rep(j,0,sz(poly)) <b>if</b> (i != j) {             rep(u,0,sz(poly[j])) {                 P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];                 <b>int</b> sc = sideOf(A, B, C), sd = sideOf(A, B, D);                 <b>if</b> (sc != sd) {                     <b>double</b> sa = C.cross(D, A), sb = C.cross(D, B);                     <b>if</b> (min(sc, sd) &lt; 0)                         segs.emplace_back(sa / (sa - sb), sgn(sc - sd));                 } <b>else if</b> (!sc &amp;&amp; !sd &amp;&amp; j&lt;i &amp;&amp; sgn((B-A).dot(D-C))&gt;0){                     segs.emplace_back(rat(C - A, B - A), 1);                     segs.emplace_back(rat(D - A, B - A), -1);                 }             }         }     } } </pre> |                  |

|   |  |
|---|--|
| <pre>     }     sort(all(segs));     <b>for</b> (<b>auto</b>&amp; s : segs) s.first = min(max(s.first, 0.0), 1.0);     <b>double</b> sum = 0;     <b>int</b> cnt = segs[0].second;     rep(j,1,sz(segs)) {         <b>if</b> (!cnt) sum += segs[j].first - segs[j - 1].first;         cnt += segs[j].second;     }     ret += A.cross(B) * sum; } <b>return</b> ret / 2; } </pre> |  |
|---|--|

ConvexHull.h

**Description:**

Returns a vector of the points of the convex hull in counter-clockwise 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 |
| <pre> <b>typedef</b> Point&lt;ll&gt; P; vector&lt;P&gt; convexHull(vector&lt;P&gt; pts) {     <b>if</b> (sz(pts) &lt;= 1) <b>return</b> pts;     sort(all(pts));     vector&lt;P&gt; h(sz(pts)+1);     <b>int</b> s = 0, t = 0;     <b>for</b> (<b>int</b> it = 2; it--; s = --t, reverse(all(pts)))         <b>for</b> (P p : pts) {             <b>while</b> (t &gt;= s + 2 &amp;&amp; h[t-2].cross(h[t-1], p) &lt;= 0) t--;             h[t++] = p;         }     <b>return</b> {h.begin(), h.begin() + t - (t == 2 &amp;&amp; h[0] == h[1])}; } </pre> |                  |

HullDiameter.h

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

|  |                  |
|--|------------------|
| "Point.h"  | c571b8, 12 lines |
| <pre> <b>typedef</b> Point&lt;ll&gt; P; array&lt;P, 2&gt; hullDiameter(vector&lt;P&gt; S) {     <b>int</b> n = sz(S), j = n &lt; 2 ? 0 : 1;     pair&lt;ll, array&lt;P, 2&gt;&gt; res({0, {S[0], S[0]}});     rep(i,0,j)         <b>for</b> (; j = (j + 1) % n) {             res = max(res, {(S[i] - S[j]).dist2(), {S[i], S[j]}});             <b>if</b> ((S[(j + 1) % n] - S[j]).cross(S[i + 1] - S[i]) &gt;= 0)                 <b>break</b>;         }     <b>return</b> res.second; } </pre> |                  |

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 |
| <pre> <b>typedef</b> Point&lt;ll&gt; P;  <b>bool</b> inHull(<b>const</b> vector&lt;P&gt;&amp; l, P p, <b>bool</b> strict = <b>true</b>) {     <b>int</b> a = 1, b = sz(l) - 1, r = !strict;     <b>if</b> (sz(l) &lt; 3) <b>return</b> r &amp;&amp; onSegment(l[0], l.back(), p);     <b>if</b> (sideOf(l[0], l[a], l[b]) &gt; 0) swap(a, b);     <b>if</b> (sideOf(l[0], l[a], p) &gt;= r    sideOf(l[0], l[b], p)&lt;= -r)         <b>return</b> <b>false</b>;     <b>while</b> (abs(a - b) &gt; 1) {         <b>int</b> c = (a + b) / 2; </pre> |                  |



```
    (sideOf(l[0], l[c], p) > 0 ? b : a) = c;
}
return sgn(l[a].cross(l[b], p)) < r;
}
```

LineHullIntersection.h

**Description:** Line-convex polygon intersection. The polygon must be ccw and have no collinear points. lineHull(line, poly) returns a pair describing the intersection of a line with the polygon:  $\bullet(-1, -1)$  if no collision,  $\bullet(i, -1)$  if touching the corner  $i$ ,  $\bullet(i, i)$  if along side  $(i, i + 1)$ ,  $\bullet(i, j)$  if crossing sides  $(i, i + 1)$  and  $(j, j + 1)$ . In the last case, if a corner  $i$  is crossed, this is treated as happening on side  $(i, i + 1)$ . The points are returned in the same order as the line hits the polygon. extrVertex returns the point of a hull with the max projection onto a line.  
**Time:**  $\mathcal{O}(\log n)$

"Point.h"7cf45b, 39 lines

```
#define cmp(i, j)  sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i)  cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
template <class P> int extrVertex(vector<P>& poly, P dir) {
    int n = sz(poly), lo = 0, hi = n;
    if (extr(0)) return 0;
    while (lo + 1 < hi) {
        int m = (lo + hi) / 2;
        if (extr(m)) return m;
        int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
        (ls < ms || (ls == ms && ls == cmp(lo, m)) ? hi : lo) = m;
    }
    return lo;
}

#define cmpL(i)  sgn(a.cross(poly[i], b))
template <class P>
array<int, 2> lineHull(P a, P b, vector<P>& poly) {
    int endA = extrVertex(poly, (a - b).perp());
    int endB = extrVertex(poly, (b - a).perp());
    if (cmpL(endA) < 0 || cmpL(endB) > 0)
        return {-1, -1};
    array<int, 2> res;
    rep(i, 0, 2) {
        int lo = endB, hi = endA, n = sz(poly);
        while ((lo + 1) % n != hi) {
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
            (cmpL(m) == cmpL(endB) ? lo : hi) = m;
        }
        res[i] = (lo + !cmpL(hi)) % n;
        swap(endA, endB);
    }
    if (res[0] == res[1]) return {res[0], -1};
    if (!cmpL(res[0]) && !cmpL(res[1]))
        switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
            case 0: return {res[0], res[0]};
            case 2: return {res[1], res[1]};
        }
    return res;
}
```

7.4 Misc. Point Set Problems

ClosestPair.h

**Description:** Finds the closest pair of points.  
**Time:**  $\mathcal{O}(n \log n)$

"Point.h"ac41a6, 17 lines

```
typedef Point<ll> P;
pair<P, P> closest(vector<P> v) {
    assert(sz(v) > 1);
    set<P> S;
    sort(all(v), [](P a, P b) { return a.y < b.y; });
    pair<ll, pair<P, P>> ret{LLONG_MAX, {P(), P()}};
    int j = 0;
```

```
    for (P p : v) {
        P d{1 + (ll)sqrt(ret.first), 0};
        while (v[j].y <= p.y - d.x) S.erase(v[j++]);
        auto lo = S.lower_bound(p - d), hi = S.upper_bound(p + d);
        for (; lo != hi; ++lo)
            ret = min(ret, {(p - *lo).dist2(), {p, *lo}});
        S.insert(p);
    }
    return ret.second;
}
```

ManhattanMST.h

**Description:** Given  $N$  points, returns up to  $4 \cdot N$  edges, which are guaranteed to contain a minimum spanning tree for the graph with edge weights  $w(p, q) = -p.x - q.x + -p.y - q.y$ . Edges are in the form (distance, src, dst). Use a standard MST algorithm on the result to find the final MST.  
**Time:**  $\mathcal{O}(N \log N)$

"Point.h"df6f59, 23 lines

```
typedef Point<int> P;
vector<array<int, 3>> manhattanMST(vector<P> ps) {
    vi id(sz(ps));
    iota(all(id), 0);
    vector<array<int, 3>> edges;
    rep(k, 0, 4) {
        sort(all(id), [&](int i, int j) {
            return (ps[i]-ps[j]).x < (ps[j]-ps[i]).y;});
        map<int, int> sweep;
        for (int i : id) {
            for (auto it = sweep.lower_bound(-ps[i].y);
                 it != sweep.end(); sweep.erase(it++)) {
                int j = it->second;
                P d = ps[i] - ps[j];
                if (d.y > d.x) break;
                edges.push_back({d.y + d.x, i, j});
            }
            sweep[-ps[i].y] = i;
        }
        for (P& p : ps) if (k & 1) p.x = -p.x; else swap(p.x, p.y);
        return edges;
    }
}
```

kdTree.h

**Description:** KD-tree (2d, can be extended to 3d)

"Point.h"bac5b0, 63 lines

```
typedef long long T;
typedef Point<T> P;
const T INF = numeric_limits<T>::max();

bool on_x(const P& a, const P& b) { return a.x < b.x; }
bool on_y(const P& a, const P& b) { return a.y < b.y; }

struct Node {
    P pt; // if this is a leaf, the single point in it
    T x0 = INF, x1 = -INF, y0 = INF, y1 = -INF; // bounds
    Node *first = 0, *second = 0;

    T distance(const P& p) { // min squared distance to a point
        T x = (p.x < x0 ? x0 : p.x > x1 ? x1 : p.x);
        T y = (p.y < y0 ? y0 : p.y > y1 ? y1 : p.y);
        return (P(x, y) - p).dist2();
    }

    Node(vector<P>&& vp) : pt(vp[0]) {
        for (P p : vp) {
            x0 = min(x0, p.x); x1 = max(x1, p.x);
            y0 = min(y0, p.y); y1 = max(y1, p.y);
        }
    }
}
```

```
    if (vp.size() > 1) {
        // split on x if width >= height (not ideal...)
        sort(all(vp), x1 - x0 >= y1 - y0 ? on_x : on_y);
        // divide by taking half the array for each child (not
        // best performance with many duplicates in the middle)
        int half = sz(vp)/2;
        first = new Node({vp.begin(), vp.begin() + half});
        second = new Node({vp.begin() + half, vp.end()});
    }
};

struct KDTree {
    Node* root;
    KDTree(const vector<P>& vp) : root(new Node({all(vp)})) {}

    pair<T, P> search(Node *node, const P& p) {
        if (!node->first) {
            // uncomment if we should not find the point itself:
            // if (p == node->pt) return {INF, P()};
            return make_pair((p - node->pt).dist2(), node->pt);
        }

        Node *f = node->first, *s = node->second;
        T bfirst = f->distance(p), bsec = s->distance(p);
        if (bfirst > bsec) swap(bsec, bfirst), swap(f, s);

        // search closest side first, other side if needed
        auto best = search(f, p);
        if (bsec < best.first)
            best = min(best, search(s, p));
        return best;
    }

    // find nearest point to a point, and its squared distance
    // (requires an arbitrary operator< for Point)
    pair<T, P> nearest(const P& p) {
        return search(root, p);
    }
};
```

DelaunayTriangulation.h

**Description:** Computes the Delaunay triangulation of a set of points. Each circumcircle contains none of the input points. If any three points are collinear or any four are on the same circle, behavior is undefined.  
**Time:**  $\mathcal{O}(n^2)$

"Point.h", "3dHull.h"c0e7bc, 10 lines

```
template<class P, class F>
void delaunay(vector<P>& ps, F trifun) {
    if (sz(ps) == 3) { int d = (ps[0].cross(ps[1], ps[2]) < 0);
        trifun(0, 1, d, 2 - d); }
    vector<P> p3;
    for (P p : ps) p3.emplace_back(p.x, p.y, p.dist2());
    if (sz(ps) > 3) for(auto t:hull3d(p3)) if ((p3[t.b]-p3[t.a]).
        cross(p3[t.c]-p3[t.a]).dot(P3(0, 0, 1)) < 0)
        trifun(t.a, t.c, t.b);
}
```

FastDelaunay.h

**Description:** Fast Delaunay triangulation. Each circumcircle contains none of the input points. There must be no duplicate points. If all points are on a line, no triangles will be returned. Should work for doubles as well, though there may be precision issues in 'circ'. Returns triangles in order  $\{t[0][0], t[0][1], t[0][2], t[1][0], \dots\}$ , all counter-clockwise.  
**Time:**  $\mathcal{O}(n \log n)$

"Point.h"eefdf5, 88 lines

```
typedef Point<ll> P;
typedef struct Quad* Q;
```

```
typedef __int128_t lll; // (can be ll if coords are < 2e4)
P arb(LLONG_MAX,LLONG_MAX); // not equal to any other point

struct Quad {
    Q rot, o; P p = arb; bool mark;
    P& F() { return r()->p; }
    Q& r() { return rot->rot; }
    Q prev() { return rot->o->rot; }
    Q next() { return r()->prev(); }
} *H;

bool circ(P p, P a, P b, P c) { // is p in the circumcircle?
    lll p2 = p.dist2(), A = a.dist2()-p2,
        B = b.dist2()-p2, C = c.dist2()-p2;
    return p.cross(a,b)*C + p.cross(b,c)*A + p.cross(c,a)*B > 0;
}

Q makeEdge(P orig, P dest) {
    Q r = H ? H : new Quad{new Quad{new Quad{0}}}};
    H = r->o; r->r()->r() = r;
    rep(i,0,4) r = r->rot, r->p = arb, r->o = i & 1 ? r : r->r();
    r->p = orig; r->F() = dest;
    return r;
}

void splice(Q a, Q b) {
    swap(a->o->rot->o, b->o->rot->o); swap(a->o, b->o);
}

Q connect(Q a, Q b) {
    Q q = makeEdge(a->F(), b->p);
    splice(q, a->next());
    splice(q->r(), b);
    return q;
}

pair<Q,Q> rec(const vector<P>& s) {
    if (sz(s) <= 3) {
        Q a = makeEdge(s[0], s[1]), b = makeEdge(s[1], s.back());
        if (sz(s) == 2) return { a, a->r() };
        splice(a->r(), b);
        auto side = s[0].cross(s[1], s[2]);
        Q c = side ? connect(b, a) : 0;
        return {side < 0 ? c->r() : a, side < 0 ? c : b->r() };
    }

#define H(e) e->F(), e->p
#define valid(e) (e->F().cross(H(base)) > 0)
    Q A, B, ra, rb;
    int half = sz(s) / 2;
    tie(ra, A) = rec({all(s) - half});
    tie(B, rb) = rec({sz(s) - half + all(s)});
    while ((B->p.cross(H(A)) < 0 && (A = A->next())) ||
        (A->p.cross(H(B)) > 0 && (B = B->r()->o)));
    Q base = connect(B->r(), A);
    if (A->p == ra->p) ra = base->r();
    if (B->p == rb->p) rb = base;

#define DEL(e, init, dir) Q e = init->dir; if (valid(e)) \
    while (circ(e->dir->F(), H(base), e->F())) { \
        Q t = e->dir; \
        splice(e, e->prev()); \
        splice(e->r(), e->r()->prev()); \
        e->o = H; H = e; e = t; \
    }
    for (;;) {
        DEL(LC, base->r(), o); DEL(RC, base, prev());
        if (!valid(LC) && !valid(RC)) break;
        if (!valid(LC) || (valid(RC) && circ(H(RC), H(LC))))
            base = connect(RC, base->r());
        else
            base = connect(base->r(), LC->r());
    }
}
```

```
    }
    return { ra, rb };
}

vector<P> triangulate(vector<P> pts) {
    sort(all(pts)); assert(unique(all(pts)) == pts.end());
    if (sz(pts) < 2) return {};
    Q e = rec(pts).first;
    vector<Q> q = {e};
    int qi = 0;
    while (e->o->F().cross(e->F(), e->p) < 0) e = e->o;
#define ADD { Q c = e; do { c->mark = 1; pts.push_back(c->p); \
    q.push_back(c->r()); c = c->next(); } while (c != e); }
    ADD; pts.clear();
    while (qi < sz(q)) if (!(e = q[qi++])->mark) ADD;
    return pts;
}

7.5 3D
PolyhedronVolume.h
Description: Magic formula for the volume of a polyhedron. Faces should
point outwards.
3058c3, 6 lines

template<class V, class L>
double signedPolyVolume(const V& p, const L& trilst) {
    double v = 0;
    for (auto i : trilst) v += p[i.a].cross(p[i.b]).dot(p[i.c]);
    return v / 6;
}

Point3D.h
Description: Class to handle points in 3D space. T can be e.g. double or
long long.
8058ae, 32 lines

template<class T> struct Point3D {
    typedef Point3D P;
    typedef const P& R;
    T x, y, z;
    explicit Point3D(T x=0, T y=0, T z=0) : x(x), y(y), z(z) {}
    bool operator<(R p) const {
        return tie(x, y, z) < tie(p.x, p.y, p.z); }
    bool operator==(R p) const {
        return tie(x, y, z) == tie(p.x, p.y, p.z); }
    P operator+(R p) const { return P(x+p.x, y+p.y, z+p.z); }
    P operator-(R p) const { return P(x-p.x, y-p.y, z-p.z); }
    P operator*(T d) const { return P(x*d, y*d, z*d); }
    P operator/(T d) const { return P(x/d, y/d, z/d); }
    T dot(R p) const { return x*p.x + y*p.y + z*p.z; }
    P cross(R p) const {
        return P(y*p.z - z*p.y, z*p.x - x*p.z, x*p.y - y*p.x);
    }
    T dist2() const { return x*x + y*y + z*z; }
    double dist() const { return sqrt((double)dist2()); }
    //Azimuthal angle (longitude) to x-axis in interval [-pi, pi]
    double phi() const { return atan2(y, x); }
    //Zenith angle (latitude) to the z-axis in interval [0, pi]
    double theta() const { return atan2(sqrt(x*x+y*y),z); }
    P unit() const { return *this/(T)dist(); } //makes dist()==1
    //returns unit vector normal to *this and p
    P normal(P p) const { return cross(p).unit(); }
    //returns point rotated 'angle' radians ccw around axis
    P rotate(double angle, P axis) const {
        double s = sin(angle), c = cos(angle); P u = axis.unit();
        return u*dot(u)*(1-c) + (*this)*c - cross(u)*s;
    }
};
```

```
3dHull.h
Description: Computes all faces of the 3-dimension hull of a point set. *No
four points must be coplanar*, or else random results will be returned. All
faces will point outwards.
Time: O(n^2)
"Point3D.h"
5b45fc, 49 lines

typedef Point3D<double> P3;

struct PR {
    void ins(int x) { (a == -1 ? a : b) = x; }
    void rem(int x) { (a == x ? a : b) = -1; }
    int cnt() { return (a != -1) + (b != -1); }
    int a, b;
};

struct F { P3 q; int a, b, c; };

vector<F> hull3d(const vector<P3>& A) {
    assert(sz(A) >= 4);
    vector<vector<PR>> E(sz(A), vector<PR>(sz(A), {-1, -1}));
#define E(x,y) E[f.x][f.y]
    vector<F> FS;
    auto mf = [&](int i, int j, int k, int l) {
        P3 q = (A[j] - A[i]).cross((A[k] - A[i]));
        if (q.dot(A[l]) > q.dot(A[i]))
            q = q * -1;
        F f{q, i, j, k};
        E(a,b).ins(k); E(a,c).ins(j); E(b,c).ins(i);
        FS.push_back(f);
    };
    rep(i,0,4) rep(j,i+1,4) rep(k,j+1,4)
        mf(i, j, k, 6 - i - j - k);

    rep(i,4,sz(A)) {
        rep(j,0,sz(FS)) {
            F f = FS[j];
            if(f.q.dot(A[i]) > f.q.dot(A[f.a])) {
                E(a,b).rem(f.c);
                E(a,c).rem(f.b);
                E(b,c).rem(f.a);
                swap(FS[j--], FS.back());
                FS.pop_back();
            }
            int nw = sz(FS);
            rep(j,0,nw) {
                F f = FS[j];
#define C(a, b, c) if (E(a,b).cnt() != 2) mf(f.a, f.b, i, f.c);
                C(a, b, c); C(a, c, b); C(b, c, a);
            }
            for (F& it : FS) if ((A[it.b] - A[it.a]).cross(
                A[it.c] - A[it.a]).dot(it.q) <= 0) swap(it.c, it.b);
            return FS;
        };
    };

sphericalDistance.h
Description: Returns the shortest distance on the sphere with radius r
between the points with azimuthal angles (longitude) f1 (φ1) and f2 (φ2)
from x axis and zenith angles (latitude) t1 (θ1) and t2 (θ2) from z axis
(0 = north pole). All angles measured in radians. The algorithm starts by
converting the spherical coordinates to cartesian coordinates so if that is
what you have you can use only the two last rows. dx*radius is then the
difference between the two points in the x direction and d*radius is the
total distance between the points.
611f07, 8 lines

double sphericalDistance(double f1, double t1,
    double f2, double t2, double radius) {
    double dx = sin(t2)*cos(f2) - sin(t1)*cos(f1);
```



```
double dy = sin(t2)*sin(f2) - sin(t1)*sin(f1);
double dz = cos(t2) - cos(t1);
double d = sqrt(dx*dx + dy*dy + dz*dz);
return radius*2*asin(d/2);
}
```

## Strings (8)

### StringMatching.java

**Description:** Comprueba todas las aparicones 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);
}
```

### Zfunc.h

**Description:** z[x] computes the length of the longest common prefix of s[i:] and s, except z[0] = 0. (abacaba -> 0010301)

**Time:**  $\mathcal{O}(n)$  3ae526, 12 lines

```
vi Z(string S) {
    vi z(sz(S));
    int l = -1, r = -1;
    rep(i,l,sz(S)) {
        z[i] = i >= r ? 0 : min(r - i, z[i - l]);
        while (i + z[i] < sz(S) && S[i + z[i]] == S[z[i]])
            z[i]++;
        if (i + z[i] > r)
            l = i, r = i + z[i];
    }
    return z;
}
```

### Manacher.h

**Description:** For each position in a string, computes p[0][i] = half length of longest even palindrome around pos i, p[1][i] = longest odd (half rounded down).

**Time:**  $\mathcal{O}(N)$  e7ad79, 13 lines

```
array<vi, 2> manacher(const string& s) {
    int n = sz(s);
    array<vi,2> p = {vi(n+1), vi(n)};
    rep(z,0,2) for (int i=0,l=0,r=0; i < n; i++) {
        int t = r-i+!z;
        if (i<r) p[z][i] = min(t, p[z][l+t]);
        int L = i-p[z][i], R = i+p[z][i]-!z;
        while (L>=1 && R+1<n && s[L-1] == s[R+1])
            p[z][i]++, L--, R++;
        if (R>r) l=L, r=R;
    }
    return p;
}
```

### MinRotation.h

**Description:** Finds the lexicographically smallest rotation of a string.

**Usage:** rotate(v.begin(), v.begin()+minRotation(v), v.end());

**Time:**  $\mathcal{O}(N)$  d07a42, 8 lines

```
int minRotation(string s) {
    int a=0, N=sz(s); s += s;
    rep(b,0,N) rep(k,0,N) {
        if (a+k == b || s[a+k] < s[b+k]) {b += max(0, k-1); break;}
        if (s[a+k] > s[b+k]) { a = b; break; }
    }
    return a;
}
```

### SuffixArray.h

**Description:** Builds suffix array for a string. sa[i] is the starting index of the suffix which is i'th in the sorted suffix array. The returned vector is of size  $n + 1$ , and sa[0] = n. The lcp array contains longest common prefixes for neighbouring strings in the suffix array: lcp[i] = lcp(sa[i], sa[i-1]), lcp[0] = 0. The input string must not contain any zero bytes.

**Time:**  $\mathcal{O}(n \log n)$  38db9f, 23 lines

```
struct SuffixArray {
    vi sa, lcp;
    SuffixArray(string& s, int lim=256) { // or basic_string<int>
        int n = sz(s) + 1, k = 0, a, b;
        vi x(all(s)+1, y(n), ws(max(n, lim)), rank(n);
        sa = lcp = y, iota(all(sa), 0);
        for (int j = 0, p = 0; p < n; j = max(1, j * 2), lim = p) {
            p = j, iota(all(y), n - j);
            rep(i,0,n) if (sa[i] >= j) y[p++] = sa[i] - j;
            fill(all(ws), 0);
            rep(i,0,n) ws[x[i]]++;
            rep(i,1,lim) ws[i] += ws[i - 1];
            for (int i = n; i--;) sa[--ws[x[y[i]]]] = y[i];
            swap(x, y), p = 1, x[sa[0]] = 0;
            rep(i,1,n) a = sa[i - 1], b = sa[i], x[b] =
                (y[a] == y[b] && y[a + j] == y[b + j]) ? p - 1 : p++;
        }
        rep(i,1,n) rank[sa[i]] = i;
        for (int i = 0, j; i < n - 1; lcp[rank[i++]] = k)
            for (k && k--, j = sa[rank[i] - 1];
                s[i + k] == s[j + k]; k++);
    }
};
```

### SuffixTree.h

**Description:** Ukkonen's algorithm for online suffix tree construction. Each node contains indices [l, r] into the string, and a list of child nodes. Suffixes are given by traversals of this tree, joining [l, r] substrings. The root is 0 (has l = -1, r = 0), non-existent children are -1. To get a complete tree, append a dummy symbol - otherwise it may contain an incomplete path (still useful for substring matching, though).

**Time:**  $\mathcal{O}(26N)$  aae0b8, 50 lines

```
struct SuffixTree {
    enum { N = 200010, ALPHA = 26 }; // N ~ 2*maxlen+10
    int toi(char c) { return c - 'a'; }
    string a; // v = cur node, q = cur position
    int t[N][ALPHA], l[N], r[N], p[N], s[N], v=0, q=0, m=2;

    void ukkadd(int i, int c) { suff:
        if (r[v]<=q) {
            if (t[v][c]==-1) { t[v][c]=m; l[m]=i;
                p[m++]=v; v=s[v]; q=r[v]; goto suff; }
            v=t[v][c]; q=l[v];
        }
        if (q==-1 || c==toi(a[q])) q++; else {
            l[m+1]=i; p[m+1]=m; l[m]=l[v]; r[m]=q;
            p[m]=p[v]; t[m][c]=m+1; t[m][toi(a[q])]=v;
            l[v]=q; p[v]=m; t[p[m]][toi(a[l[m]])]=m;
            v=s[p[m]]; q=l[m];
            while (q<r[m]) { v=t[v][toi(a[q])]; q+=r[v]-l[v]; }
            if (q==r[m]) s[m]=v; else s[m]=m+2;
        }
    }
};
```

```
        q=r[v]-(q-r[m]);  m+=2;  goto suff;
    }
}

SuffixTree(string a) : a(a) {
    fill(r,r+N,sz(a));
    memset(s, 0, sizeof s);
    memset(t, -1, sizeof t);
    fill(t[1],t[1]+ALPHA,0);
    s[0] = 1; l[0] = l[1] = -1; r[0] = r[1] = p[0] = p[1] = 0;
    rep(i,0,sz(a)) ukkadd(i, toi(a[i]));
}

// example: find longest common substring (uses ALPHA = 28)
pii best;
int lcs(int node, int i1, int i2, int olen) {
    if (l[node] <= i1 && i1 < r[node]) return 1;
    if (l[node] <= i2 && i2 < r[node]) return 2;
    int mask = 0, len = node ? olen + (r[node] - l[node]) : 0;
    rep(c,0,ALPHA) if (t[node][c] != -1)
        mask |= lcs(t[node][c], i1, i2, len);
    if (mask == 3)
        best = max(best, {len, r[node] - len});
    return mask;
}

static pii LCS(string s, string t) {
    SuffixTree st(s + (char)('z' + 1) + t + (char)('z' + 2));
    st.lcs(0, sz(s), sz(s) + 1 + sz(t), 0);
    return st.best;
}
};
```

### Hashing.h

**Description:** Self-explanatory methods for string hashing. 3f02d8, 44 lines

```
// Arithmetic mod 2^64-1. 2x slower than mod 2^64 and more
// code, but works on evil test data (e.g. Thue-Morse, where
// ABBA... and BAAB... of length 2^10 hash the same mod 2^64).
// "typedef ull H;" instead if you think test data is random,
// or work mod 10^9+7 if the Birthday paradox is not a problem.
struct H {
    typedef uint64_t ull;
    ull x; H(ull x=0) : x(x) {}
#define OP(O,A,B) H operator O(H o) { ull r = x; asm \
(A "addq %%rdx, %0\n adcq $0,%0" : "+a"(r) : B); return r; }
OP(+,,"d"(o.x)) OP(*,"mul %l\n", "r"(o.x) : "rdx")
H operator-(H o) { return *this + ~o.x; }
ull get() const { return x + !~x; }
bool operator==(H o) const { return get() == o.get(); }
bool operator<(H o) const { return get() < o.get(); }
};

static const H C = (11)1e11+3; // (order ~ 3e9; random also ok)

struct HashInterval {
    vector<H> ha, pw;
    HashInterval(string& str) : ha(sz(str)+1), pw(ha) {
        pw[0] = 1;
        rep(i,0,sz(str))
            ha[i+1] = ha[i] * C + str[i],
            pw[i+1] = pw[i] * C;
    }
    H hashInterval(int a, int b) { // hash [a, b)
        return ha[b] - ha[a] * pw[b - a];
    }
};

vector<H> getHashes(string& str, int length) {
    if (sz(str) < length) return {};
    H h = 0, pw = 1;
```

### Hashing AhoCorasick MergeIntervals Skyline

```
    rep(i,0,length)
        h = h * C + str[i], pw = pw * C;
    vector<H> ret = {h};
    rep(i,length,sz(str)) {
        ret.push_back(h = h * C + str[i] - pw * str[i-length]);
    }
    return ret;
}
```

H hashString(string& s){H h{}; for(char c:s) h=h\*C+c;return h;}

### AhoCorasick.h

**Description:** AhoCorasick automaton, used for multiple pattern matching. Initialize with AhoCorasick ac(patterns); the automaton start node will be at index 0. find(word) returns for each position the index of the longest word that ends there, or -1 if none. findAll(−, word) finds all words (up to  $N\sqrt{N}$  many if no duplicate patterns) that start at each position (shortest first). Duplicate patterns are allowed; empty patterns are not. To find the longest words that start at each position, reverse all input. For large alphabets, split each symbol into chunks, with sentinel bits for symbol boundaries. **Time:** construction takes  $\mathcal{O}(26N)$ , where  $N$  = sum of length of patterns. find(x) is  $\mathcal{O}(N)$ , where  $N$  = length of x. findAll is  $\mathcal{O}(NM)$ . f35677, 66 lines

```
struct AhoCorasick {
    enum {alpha = 26, first = 'A'}; // change this!
    struct Node {
        // (nmatches is optional)
        int back, next[alpha], start = -1, end = -1, nmatches = 0;
        Node(int v) { memset(next, v, sizeof(next)); }
    };
    vector<Node> N;
    vi backp;
    void insert(string& s, int j) {
        assert(!s.empty());
        int n = 0;
        for (char c : s) {
            int& m = N[n].next[c - first];
            if (m == -1) { n = m = sz(N); N.emplace_back(-1); }
            else n = m;
        }
        if (N[n].end == -1) N[n].start = j;
        backp.push_back(N[n].end);
        N[n].end = j;
        N[n].nmatches++;
    }
    AhoCorasick(vector<string>& pat) : N(1, -1) {
        rep(i,0,sz(pat)) insert(pat[i], i);
        N[0].back = sz(N);
        N.emplace_back(0);

        queue<int> q;
        for (q.push(0); !q.empty(); q.pop()) {
            int n = q.front(), prev = N[n].back;
            rep(i,0,alpha) {
                int &ed = N[n].next[i], y = N[prev].next[i];
                if (ed == -1) ed = y;
                else {
                    N[ed].back = y;
                    (N[ed].end == -1 ? N[ed].end : backp[N[ed].start])
                        = N[y].end;
                    N[ed].nmatches += N[y].nmatches;
                    q.push(ed);
                }
            }
        }
    }
    vi find(string word) {
        int n = 0;
        vi res; // ll count = 0;
```

```
        for (char c : word) {
            n = N[n].next[c - first];
            res.push_back(N[n].end);
            // count += N[n].nmatches;
        }
        return res;
    }
    vector<vi> findAll(vector<string>& pat, string word) {
        vi r = find(word);
        vector<vi> res(sz(word));
        rep(i,0,sz(word)) {
            int ind = r[i];
            while (ind != -1) {
                res[i - sz(pat[ind]) + 1].push_back(ind);
                ind = backp[ind];
            }
        }
        return res;
    }
};
```

## Others (9)

### MergeIntervals.java

**Description:** Union de Intervalos 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 edificios encontrar la forma/area del skyline, se devuelven los vertices superior izquierdo hasta el ultimo, que es inferior derecho **Time:**  $\mathcal{O}(n\log(n))$  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:<br>Escribe algunos casos de prueba simples, si la muestra no es suficiente.<br>Sabes los limites? Genera los casos maximos.<br>Esta bien el uso de la memoria?<br>Podria haber overflow?<br>Asegurese de enviar el archivo correcto.   |          |
| Respuesta incorrecta: WA<br>USA LONG<br>Imprime tu solucion! Imprime tambien la salida de debug.<br>Estas limpiando todas las estructuras de datos entre casos de prueba?<br>Puede tu algoritmo manejar todo el rango de entrada?<br>Vuelve a leer el enunciado completo del problema.<br>Manejas todos los casos limite correctamente?<br>Has entendido correctamente el problema?<br>Codigo copiado incorrecto?<br>Alguna variable no inicializada?<br>Algun desbordamiento?<br>Variables con el mismo nombre?<br>Recursividad correcta?<br>Confundir N y M, i y j, etc.?<br>Estas seguro de que tu algoritmo funciona?<br>En que casos especiales no has pensado?<br>Estas seguro de que las funciones STL (Libreria estandar) que usas funcionan como crees?<br>Agrega algunas respuestas a las preguntas, tal vez vuelva a enviar.<br>Crea algunos casos de prueba para ejecutar su algoritmo.<br>Ve a traves del algoritmo para un caso simple.<br>Revisa esta lista nuevamente.<br>Explique su algoritmo a un compadre de equipo.<br>Pidele al compadre de equipo que mire tu codigo.<br>Salga a dar un paseillo, p. al posadero.<br>Es correcto su formato de salida? (incluyendo espacios en blanco)<br>Vuelva a escribir su solucion desde el principio o deje que un compadre de equipo lo haga. |          |

|   |  |
|---|--|
| Error de ejecucion (RTE):<br>USA LONG<br>Has probado todos los casos limites localmente?<br>Alguna variable no inicializada?<br>Esta leyendo o escribiendo fuera del rango de cualquier vector?<br>Alguna llamada que pueda fallar?<br>Alguna posible division por 0? (modulo 0 por ejemplo)<br>Alguna recursion infinita posible?<br>Iteradores invalidos?<br>Estas usando demasiada memoria?<br>Depuracion con reenvios (por ejemplo, seniales reasignadas, consulte Varios). |  |
| Tiempo limite (TLE):<br>Tienes posibles bucles infinitos?<br>Cual es la complejidad de su algoritmo?<br>Estas copiando muchos datos innecesarios? (Referencias)<br>Que tan grande es la entrada y la salida? (considere buffered reader)<br>Evite el vector, el mapa. (use arrays)<br>Usas vectores? Cambiar a array.<br>Que piensan tus compadres de equipo sobre tu algoritmo?  |  |
| Memoria limite excedida (MLE):<br>Cual es la cantidad maxima de memoria que su algoritmo deberia necesitar?<br>Esta limpiando todas las estructuras de datos entre casos de prueba?   |  |
| Vida antes que muerte. Fuerza antes que debilidad. Viaje antes que destino.   |  |