ANCHETA DE ALGORITMOS PARA MARATONES DE PROGRAMACIÓN
UNIVERSIDAD DE LOS ANDES
BOGOTÁ, COLOMBIA

Última versión: 2008/04/05 08:05 a.m.

AUTORES: ALEJANDRO SOTELO, FEDERICO ARBOLEDA E IVÁN REY.
CON LA GRAN COLABORACIÓN DE AUGUSTO TORRES.
AGRADECIMIENTOS ESPECIALES A RAFAEL GARCÍA POR SU DILIGENCIA Y PACIENCIA PARA COLABORARNOS EN TODO MOMENTO.

POR PROBLEMAS DE ESPACIO NO SE SIGUIÓ NINGUNA REGLA ESTÁNDAR DE INDENTACIÓN.
ALERTA!: JAVA PUEDE IMPRIMIR "-0.0" EN VEZ DE "0.0".

CLASES ÚTILES JDK 6.0	
java.awt	Point,Polygon,Shape
java.awt.geom	Area,GeneralPath,Line2D,Path2D,Rectangle2D
java.lang.math	Math,BigDecimal,BigInteger
java.util	ArrayList<E>,Arrays,Calendar,Collections,Date,LinkedList<E>, StringBuilder,TreeMap<K,V>,TreeSet<E>
java.text	DecimalFormat

ARREGLOS (ORDENAMIENTO, SELECCIÓN, BÚSQUEDAS, PERMUTACIONES, ...)

CASOS DE EJEMPLO CON LONG

☑ **Merge Sort - [John von Neumann (1945)] {O(n*log₂(n))}**

```
void mergeSort(long[] arr, long[] arrTmp, int pi, int pf) {
  if (pf<=pi) return;
  int m=(pi+pf)/2+1,i,j,k; mergeSort(arr,arrTmp,pi,m-1); mergeSort(arr,arrTmp,m,pf);
  for (i=pi,j=m,k=pi; i<=m-1&&j<=pf; k++) arrTmp[k]=arr[i]<=arr[j]?arr[i++]:arr[j++];
  for (; i<=m-1; k++,i++) arrTmp[k]=arr[i];
  for (k=pi; k<j; k++) arr[k]=arrTmp[k];
}
```

☑ **Quick Sort - [C. A. R. Hoare (1960)] {O(n*log₂(n))}**

```
void quickSort(long[] arr, int pi, int pf) {
  if (pi>=pf) return;
  long piv=arr[(pi+pf)/2]; int i=pi,j=pf;
  for ( ; i<=j; i++,j--) {
    while (arr[i]<piv) i++;
    while (arr[j]>piv) j--;
    if (i>j) break; if (i!=j) {long tmp=arr[i]; arr[i]=arr[j]; arr[j]=tmp;}
  }
  quickSort(arr,pi,j); quickSort(arr,i,pf);
}
```

☑ **Insertion Sort {O(n²)}**

```
void insertionSort(long[] arr, int pi, int pf) {
  for (int i=pi+1; i<=pf; i++)
  {int j=i; long v=arr[i]; for ( ; j>pi && arr[j-1]>v; j--) arr[j]=arr[j-1]; arr[j]=v;}
}
```

☑ **Bubble Sort {O(n²)}**

```
void bubbleSort(long[] arr, int pi, int pf) {
  for (int i=pi+1; i<=pf; i++) for (int j=pf; j>=i; j--) if (arr[j-1]>arr[j])
  {long tmp=arr[j-1]; arr[j-1]=arr[j]; arr[j]=tmp;}
}
```

☑ **Binary Search {log₂(n)}**

```
int busquedaBinaria(long[] a, int lI, int lS, long v) {
  while (lI<=lS)
  {int lM=(lI+lS)/2; long p=a[lM]; if (p<v) lI=lM+1; else if (p>v) lS=lM-1; else return lM;}
  return -(lI+1);
}
```

☒ **Búsqueda en silla**

☑ **k-ésimo menor - Quickselect [C. A. R. Hoare (1960)] {expected: O(n); worst: O(n²)}**

```
<T> void swap(T[] arr, int i, int j) {T tmp=arr[i]; arr[i]=arr[j]; arr[j]=tmp;}
<T extends Comparable<T>> T select(T[] arr, int pI, int pF, int k) {
  while (true) {
    int s=pI; swap(arr,(pI+pF)/2,pF); T piv=arr[pF];
    for (int j=pI; j<pF; j++) if (arr[j].compareTo(piv)<0) swap(arr,s++,j);
    swap(arr,s,pF); if (k==s) return arr[k]; if (k<s) pF=s-1; else pI=s+1;
  }
}
```

☑ **Mínimo número de swaps entre elementos adyacentes para ordenar un arreglo sin repetidos (número de inversiones de una permutación) {O(n*log₂(n))}**

En mergeSort, cambiar "arrTmp[k]=arr[i]<=arr[j]?arr[i++]:arr[j++]; " por
"{if (arr[i]<=arr[j]) arrTmp[k]=arr[i++]; else {numSwaps+=j-k; arrTmp[k]=arr[j++];}}"
declarando fuera del método una variable numSwaps que se inicialice cuando corresponda.

☑ **Mínimo número de swaps para ordenar un arreglo sin repetidos {O(n*log₂(n))}**

```
int minSwaps(long[] arr) {
  int n=arr.length,res=0; long m[][]=new long[n][]; boolean used[]=new boolean[n];
  for (int i=0; i<n; i++) m[i]=new long[]{arr[i],i};
  Arrays.sort(m,new Comparator<long[]>() {
    public int compare(long[] a1, long[] a2)
    {return a1[0]!=a2[0]?(a1[0]<a2[0]?-1:1):(a1[1]!=a2[1]?(a1[1]<a2[1]?-1:1):0);}
  });
  for (int i=0,c,j; i<n; i++) if (!used[i])
  {for (j=i,c=0; !used[j]; used[j]=true,j=(int)(m[j][1]),c++); res+=c-1;}
  return res;
}
```

☑ **Rotar matriz a la derecha {O(n*m)}**

```
long[][] rotarMatrizDer(long[][] mat) {
  int n=mat.length,m=n==0?0:mat[0].length; long res[][]=new long[m][n];
  for (int i=0; i<n; i++) for (int j=0; j<m; j++) res[j][i]=mat[n-1-i][j];
  return res;
}
```

☑ **Rotar matriz a la izquierda {O(n*m)}**

```
long[][] rotarMatrizIzq(long[][] mat) {
  int n=mat.length,m=n==0?0:mat[0].length; long res[][]=new long[m][n];
  for (int i=0; i<n; i++) for (int j=0; j<m; j++) res[j][i]=mat[i][m-1-j];
  return res;
}
```

☑ **Permutaciones de una bolsa {O(n!/(n₁!*n₂!*...*nₖ!))}**

```
void permutaciones(char[] arr, int[] frequencySymbols, char[] symbols, int k) {
  if (k==arr.length) {System.out.println(new String(arr)); return;}
  for (int i=0; i<symbols.length; i++) if (frequencySymbols[i]>0) {
    frequencySymbols[i]--; arr[k]=symbols[i];
    permutaciones(arr,frequencySymbols,symbols,k+1);
    frequencySymbols[i]++; arr[k]='\0';
  }
}
```

☑ **k-ésima permutación en orden lexicográfico {O(n²)}**

```
char[] permutation(int k, char[] s) {
  int n=s.length,fact=1,i,j,tj; char ts; s=s.clone();
  for (j=2; j<n; j++) fact*=j;
  for (j=1; j<n; j++) {
    tj=(k/fact)%(n+1-j); ts=s[j+tj-1]; for (i=j+tj; i>j; i--) s[i-1]=s[i-2];
    s[j-1]=ts; fact/=n-j;
  }
  return s;
}
```

ÁLGEBRA LINEAL

☑ **Multiplicación de matrices cuadradas {O(n³)}**

```
void mmult(double[][] a, double[][] b, double[][] answer) { // answer=a*b
  for (int i=0,n=a.length; i<n; i++) for (int j=0; j<n; j++)
  {double r=0d; for (int k=0; k<n; k++) r+=a[i][k]*b[k][j]; answer[i][j]=r;}
}
```

☒ Multiplicación de matrices cuadradas - Strassen [V. Strassen (1969)] {$O(n^{\log_2(7)})$}
☑ Solución sistemas lineales - Eliminación de Gauss-Jordan [Jiuzhang suanshu (179)] {$O(n^3)$}

```
double gaussJordan(double[][] a) { // a puede ser una matriz extendida.
  double det=1d,tmp[],z; int i,u,v,w,n=a.length,m=n==0?0:a[0].length;
  for (i=0; i<n; i++) {
    for (w=i,u=i+1; u<n; u++) if (Math.abs(a[u][i])>Math.abs(a[w][i])) w=u;
    if (Math.abs(a[w][i])<1e-12) return 0d;
    if (w!=i) {tmp=a[w]; a[w]=a[i]; a[i]=tmp; det*=-1d;}
    for (v=i,det*=(z=a[i][i]); v<m; v++) a[i][v]/=z;
    for (u=0; u<n; u++) if (u!=i&&Math.abs(z=a[u][i])>=1e-12)
      for (v=i; v<m; v++) a[u][v]-=z*a[i][v];
  }
  return det;
}
```

☑ Solución sistemas lineales - Descomposición QR mediante reflexiones de Householder {$O(n^3)$}

```
// http://www.docjar.com/html/api/jmat/data/matrixDecompositions/QRDecomposition.java.html
double[] solveQR(double[][] mat, double[] b) { // Least squares solution of mat*X=b
  int m=b.length,i; double nb[][]=new double[m][1],r[]=new double[m];
  for (i=0; i<m; i++) nb[i][0]=b[i]; nb=solveQR(mat,nb); for (i=0; i<m; i++) r[i]=nb[i][0];
  return r;
}
double[][] solveQR(double[][] mat, double[][] b) { // Least squares solution of mat*X=b
  int m=mat.length,n=m==0?0:mat[0].length,i,j,k; // m>=n, b.length==m
  double qr[][]=new double[m][],rdiag[]=new double[n],nrm,s;
  for (i=0; i<m; i++) qr[i]=mat[i].clone();
  for (k=0; k<n; k++) {
    for (nrm=0,i=k; i<m; i++) nrm+=qr[i][k]*qr[i][k];
    if (Math.abs(nrm=Math.sqrt(nrm))>1e-15) {
      if (qr[k][k]<0) nrm*=-1;
      for (i=k; i<m; i++) qr[i][k]/=nrm;
      for (qr[k][k]+=1.0,j=k+1; j<n; j++) {
        for (s=0d,i=k; i<m; i++) s+=(qr[i][k]*qr[i][j]);
        for (s=-s/qr[k][k],i=k; i<m; i++) qr[i][j]+=(s*qr[i][k]);
      }
    }
    if (Math.abs(rdiag[k]=-nrm)<1e-15) return null;
  }
  int nx=m==0?0:b[0].length; double[][] x=new double[m][],res=new double[n][nx];
  for (i=0; i<m; i++) x[i]=b[i].clone();
  for (k=0; k<n; k++) for (j=0; j<nx; j++) {
    for (s=0d,i=k; i<m; i++) s+=(qr[i][k]*x[i][j]);
    for (s=-s/qr[k][k],i=k; i<m; i++) x[i][j]+=(s*qr[i][k]);
  }
  for (k=n-1; k>=0; k--) {
    for (j=0; j<nx; j++) x[k][j]/=rdiag[k];
    for (i=0; i<k; i++) for (j=0; j<nx; j++) x[i][j]-=(x[k][j]*qr[i][k]);
  }
  for (k=0; k<n; k++) for (j=0; j<nx; j++) res[k][j]=x[k][j];
  return x;
}
```

☑ Inversa de una matriz de Vandermonde - Algoritmo de Traub [J. Traub (1966)] {$O(n^2)$}

```
double[][] vandermondeInverse(double[] e) { // vandermondeMatrix[i][j]=e[i]^(j-1)
  int n=e.length,k,i,j;
  double a[]=new double[n+1],b[]=new double[n+1],inv[][]=new double[n][n],p,q,r;
  for (a[0]=-e[0],a[1]=1,k=1; k<n; System.arraycopy(b,0,a,0,k+2),k++)
    for (i=0; i<=k+1; i++) b[i]=(i>0?a[i-1]:0)-e[k]*(i<=k?a[i]:0);
  for (j=0; j<n; j++) {
    for (p=e[j],q=inv[n-1][j]=1,r=a[1],k=1; k<n; r+=a[k+1]*(k+1)*p,k++,p*=e[j])
      q=inv[n-1-k][j]=e[j]*q+a[n-k];
    for (k=0; k<n; k++) inv[k][j]/=r;
  }
  return inv;
}
```

☑ Polynomial interpolation - Algoritmo de Traub [J. Traub (1966)] {$O(n^2)$}

```
double[] polinomialInterpolationTraub(double[] x, double[] y) {
  int n=x.length,i,j; double vi[][]=vandermondeInverse(x),a[]=new double[n];
  for (i=0; i<n; i++) for (j=0; j<n; j++) a[i]+=vi[i][j]*y[j];
  return a; // f(x)=a[0]+a[1]*x+...+a[i]*x^i+...+a[n]*x^n pasa por los puntos dados
}
```

☑ **Polynomial interpolation - Algoritmo de Björck-Pereyra [A. Björck, V. Pereyra (1970)]** {O(n^2)}

```
double[] polinomialInterpolationBjorckPereyra(double[] x, double[] y) {
  int n=x.length,i,k; double[] a=y.clone();
  for (k=1; k<n; k++) for (i=n-1; i>=k; i--) a[i]=(a[i]-a[i-1])/(x[i]-x[i-k]);
  for (k=n-1; k>=1; k--) for (i=k-1; i<n-1; i++) a[i]-=a[i+1]*x[k-1];
  return a; // f(x)=a[0]+a[1]*x+...+a[i]*x^i+...+a[n]*x^n pasa por los puntos dados
}
```

☑ **Regresión lineal - Mínimos cuadrados (lineal)** {O(n)}

```
double[] regresionLineal(double[][] pt) {
  int n=pt.length; double x,y,sX1=0d,sX2=0d,sY1=0d,sY2=0d,sXY=0d;
  for (double[] p:pt) {x=p[0]; y=p[1]; sX1+=x; sX2+=x*x; sY1+=y; sY2+=y*y; sXY+=x*y;}
  double xp=sX1/n,yp=sY1/n,ssxx=sX2-n*xp*xp,ssyy=sY2-n*yp*yp,ssxy=sXY-n*xp*yp;
  double b=ssxy/ssxx,a=yp-b*xp,s=Math.sqrt((ssyy-b*ssxy)/(n-2));
  double seA=s*Math.sqrt(1d/n+xp*xp/ssxx),seB=s/Math.sqrt(ssxx);
  return new double[]{a,b,seA,seB}; // y=a+b*x con errores estándar seA y seB
}
```

☑ **Regresión polinomial - Mínimos cuadrados (polinomial)** {O((n*$grado^2$)↑$grado^3$)}

```
double[] regresionPolinomial(double[][] pt, int grado) {
  int n=pt.length,m=grado+1,i,j,k; double mat[][]=new double[m][m],b[]=new double[m];
  for (i=0; i<m; i++) for (j=0; j<m; j++) for (k=0; k<n; k++) mat[i][j]+=Math.pow(pt[k][0],i+j);
  for (i=0; i<m; i++) for (k=0; k<n; k++) b[i]+=Math.pow(pt[k][0],i)*pt[k][1];
  double[] a=solveQR(mat,b);
  return a; // f(x)=a[0]+a[1]*x+...+a[i]*x^i+...+a[n]*x^n es el polinomio resultado
}
```

GRAFOS

GRAFOS NO DIRIGIDOS TIENEN SIMÉTRICA SU MATRIZ DE ADYACENCIA
LA DIAGONAL DE LAS MATRICES DE ADYACENCIA DEBE CONTENER CEROS
EN LAS COMPLEJIDADES, V ES EL NÚMERO DE NODOS Y E ES EL NÚMERO DE ARCOS

☑ **Convertidor de matrices de adyacencia a listas de adyacencia** {O(V^2)}

```
int[][] to_lAdy(double[][] mAdy) {
  int n=mAdy.length,lAdy[][]=new int[n][],u,v;
  for (u=0; u<n; u++) {
    List<Integer> p=new ArrayList<Integer>();
    for (v=0; v<n; v++) if (u!=v&&mAdy[u][v]!=Double.POSITIVE_INFINITY) p.add(v);
    lAdy[u]=toArr(p);
  }
  return lAdy;
}
int[] toArr(List<Integer> p) {int r[]=new int[p.size()],i=0; for (int x:p) r[i++]=x; return r;}
```

☑ **BFS (Breadth-First Search)** {O(V+E)}

```
int[] bfs(int[][] lAdy, int v) {
  int n=lAdy.length,res[]=new int[n],queue[]=new int[n],t=0,inf=Integer.MAX_VALUE;
  Arrays.fill(res,inf); res[queue[t++]=v]=0;
  for (int i=0; i<t; i++)
  {v=queue[i]; for (int w:lAdy[v]) if (res[w]==inf) res[queue[t++]=w]=res[v]+1;}
  return res;
}
```

☑ **DFS (Depth-First Search)** {O(V+E)}

Véase: Sort topológico

☑ **TSP (Traveling Salesman Problem)** {O(V^2*2^V)}

```
double tsp(double[][] mAdy, int v) {
  int n=mAdy.length,t=1<<n; double mem[][]=new double[t][n];
  for (double[] arr:mem) Arrays.fill(arr,-1d); return tsp(mAdy,n,v,v,1<<v,mem);
}
double tsp(double[][] mAdy, int n, int v1, int v2, int visitados, double[][] mem) {
  if (mem[visitados][v1]>=0d) return mem[visitados][v1];
  if (visitados==(1<<n)-1) return mem[visitados][v1]=mAdy[v1][v2];
```

```
  double min=Double.POSITIVE_INFINITY,d;
  for (int e=visitados,j=0; j<n; j++,e>>>=1) if ((e&1)==0&&(d=mAdy[v1][j])<min)
    min=Math.min(min,d+tsp(mAdy,n,j,v2,visitados|(1<<j),mem));
  return mem[visitados][v1]=min;
}
```

☑ **Rutas más cortas con caminos de longitud menor o igual que q {O(V³*log₂(q))}**

```
void mmult(double[][] a, double[][] b, double[][] tmp) { // a*=b
  int n=a.length;
  for (int i=0; i<n; i++) for (int j=0; j<n; j++) {
    double r=Double.POSITIVE_INFINITY;
    for (int k=0; k<n; k++) if (a[i][k]<r&&b[k][j]<r) r=Math.min(r,a[i][k]+b[k][j]);
    tmp[i][j]=r;
  }
  for (int i=0; i<n; i++) System.arraycopy(tmp[i],0,a[i],0,n);
}
double[][] costoCaminosQ(double[][] mAdy, int q) {
  int n=mAdy.length,y=q; double[][] x=new double[n][n],z=new double[n][n],u=new double[n][n];
  for (int i=0; i<n; i++)
  {Arrays.fill(z[i],Double.POSITIVE_INFINITY); System.arraycopy(mAdy[i],0,x[i],0,n); z[i][i]=0;}
  while (y!=0) {if ((y&1)==0) {y/=2; mmult(x,x,u);} else {y--; mmult(z,x,u);}}
  return z;
}
```

☑ **Rutas más cortas - Floyd-Warshall {O(V³)}**

```
double[][] floydWarshall(double[][] mAdy) {
  int n=mAdy.length; double x[][]=new double[n][n];
  for (int i=0; i<n; i++) System.arraycopy(mAdy[i],0,x[i],0,n);
  for (int k=0; k<n; k++) for (int i=0; i<n; i++) for (int j=0; j<n; j++)
    x[i][j]=Math.min(x[i][j],x[i][k]+x[k][j]);
  return x;
}
```

☑ **Minimax - Floyd-Warshall {O(V³)}**

Ponga x[i][j]=Math.min(x[i][j],Math.max(x[i][k],x[k][j])) en el algoritmo de Floyd-Warshall.

☑ **Maximin - Floyd-Warshall {O(V³)}**

Ponga x[i][j]=Math.max(x[i][j],Math.min(x[i][k],x[k][j])) en el algoritmo de Floyd-Warshall.

☑ **Safest path - Floyd-Warshall {O(V³)}**

Ponga x[i][j]=Math.max(x[i][j],x[i][k]*x[k][j]) en el algoritmo de Floyd-Warshall (los costos de los arcos representan probabilidades de supervivencia).

☑ **Clausura transitiva - Floyd-Warshall {O(V³)}**

```
boolean[][] floydWarshall(double[][] mAdy) {
  int n=mAdy.length,i,j,k; boolean x[][]=new boolean[n][n];
  for (i=0; i<n; i++) for (j=0; j<n; j++) x[i][j]=i==j||mAdy[i][j]!=Double.POSITIVE_INFINITY;
  for (k=0; k<n; k++) for (i=0; i<n; i++) for (j=0; j<n; j++)
    x[i][j]=x[i][j]||(x[i][k]&&x[k][j]);
  return x;
}
```

☑ **Rutas más cortas desde un nodo - Dijkstra [E. Dijkstra (1959)] {O(log₂(V)*(V+E))}**

```
double[] dijkstra(double[][] mAdy, int[][] lAdy, int v) {
  TreeMap<Double,List<Integer>> map=new TreeMap<Double,List<Integer>>();
  int n=mAdy.length; double mins[]=new double[n]; boolean[] vis=new boolean[n];
  Arrays.fill(mins,Double.POSITIVE_INFINITY);
  mins[v]=0d; map.put(0d,new ArrayList<Integer>(Arrays.asList(v)));
  while (map.size()>0) for (int f:map.pollFirstEntry().getValue()) if (!vis[f]) {
    vis[f]=true;
    for (int g:lAdy[f]) if (!vis[g]&&mins[f]+mAdy[f][g]<mins[g]-1e-11) {
      mins[g]=mins[f]+mAdy[f][g]; List<Integer> p=map.get(mins[g]);
      if (p==null) map.put(mins[g],p=new ArrayList<Integer>()); p.add(g);
    }
  }
  return mins;
}
```

☑ **Rutas más cortas desde un nodo - Dijkstra (Priority Queue) {O(log₂(V)*(V+E))}**

```
double[] dijkstraPriorityQueue(double[][] mAdy, int[][] lAdy, int v) {
  class Nodo implements Comparable<Nodo> {
    int i; double d; Nodo(int pi, double pd) {i=pi; d=pd;}
```

```
    public int compareTo(Nodo e) {return d!=e.d?(d<e.d?-1:1):i-e.i;}
  };
  PriorityQueue<Nodo> pq=new PriorityQueue<Nodo>(lAdy.length);
  int n=mAdy.length,f; double mins[]=new double[n]; boolean[] vis=new boolean[n];
  Arrays.fill(mins,Double.POSITIVE_INFINITY);
  mins[v]=0d; pq.add(new Nodo(v,0d));
  while (!pq.isEmpty()) if (!vis[f=pq.poll().i]) {
    vis[f]=true;
    for (int g:lAdy[f]) if (!vis[g]&&mins[f]+mAdy[f][g]<mins[g]-1e-11)
      pq.add(new Nodo(g,mins[g]=mins[f]+mAdy[f][g]));
  }
  return mins;
}
```

☑ Rutas más cortas desde un nodo (costos negativos) - Bellman-Ford {O(V*E)}

```
double[] bellmanFord(double[][] mAdy, int[][] lAdy, int v) {
  int n=mAdy.length; double[] mins=new double[n];
  Arrays.fill(mins,Double.POSITIVE_INFINITY); mins[v]=0d;
  for (int k=1; k<n; k++) for (int i=0; i<n; i++) for (int j:lAdy[i])
    mins[j]=Math.min(mins[j],mins[i]+mAdy[i][j]);
  for (int i=0; i<n; i++) for (int j:lAdy[i])
    if (mins[j]>mins[i]+mAdy[i][j]+1e-10) return null; // Ciclo de peso negativo
  return mins;
}
```

☑ Rutas más cortas desde un nodo (costos enteros entre 1 y M) - Bucket Priority Queue {O(V*M+E)}

```
int[] dijkstraE(int[][] mAdy, int[][] lAdy, int v, int M) { // Apropiado para 1<=M<=70
  int n=mAdy.length,b[][]=new int[M+1][n],tamB[]=new int[M+1],res[]=new int[n],d,i,j,t,q,sum=1;
  boolean vis[]=new boolean[n]; Arrays.fill(res,Integer.MAX_VALUE);
  for (res[v]=0,b[0][tamB[0]++]=v,d=0; sum>0; d++) {
    for (i=0,t=tamB[0]; i<t; i++) if (!vis[v=b[0][i]]) {
      vis[v]=true;
      for (int w:lAdy[v]) if (d+(q=mAdy[v][w])<res[w]) {res[w]=d+q; b[q][tamB[q]++]=w;}
    }
    int arrTmp[]=b[0];
    for (j=1,sum=0; j<=M; j++) {b[j-1]=b[j]; sum+=(tamB[j-1]=tamB[j]);}
    b[M]=arrTmp; tamB[M]=0;
  }
  return res;
}
```

☑ Árbol de expansión minimal - Kruskal [J. Kruskal (1956)] {O(E*log₂(V))}

```
boolean[][] kruskal(final double[][] mAdy, int[][] lAdy) { // Para grafos conexos no dirigidos
  int n=mAdy.length,t=0; for (int[] a:lAdy) t+=a.length; int edges[][]=new int[t][],k=0;
  for (int i=0; i<n; i++) for (int j:lAdy[i]) if (j>i) edges[k++]=new int[]{i,j,k};
  Arrays.sort(edges,0,k,new Comparator<int[]>() {public int compare(int[] u, int[] v)
  {double r=mAdy[u[0]][u[1]]-mAdy[v[0]][v[1]]; return r!=0?(r<0?-1:1):u[2]-v[2];}});
  DisjointSet forest[]=new DisjointSet[n]; for (int i=0; i<n; i++) forest[i]=new DisjointSet();
  boolean[][] res=new boolean[n][n];
  for (int e[]:Arrays.copyOfRange(edges,0,k))
    if (DisjointSet.find(forest[e[0]])!=DisjointSet.find(forest[e[1]]))
    {res[e[0]][e[1]]=res[e[1]][e[0]]=true; DisjointSet.union(forest[e[0]],forest[e[1]]);}
  return res;
}
```

☑ Árbol de expansión minimal - Prim [V. Jarník (1930), R. C. Prim (1957)] {O(E*V)}

```
boolean[][] prim(double[][] mAdy, int[][] lAdy) { // Para grafos conexos no dirigidos
  int n=mAdy.length,k,i,ie,je; boolean res[][]=new boolean[n][n],vis[]=new boolean[n];
  for (vis[0]=true,k=1; k<n; k++) {
    double me=Double.POSITIVE_INFINITY;
    for (i=0,ie=je=-1; i<n; i++) if (vis[i]) for (int j:lAdy[i]) if (!vis[j]&&mAdy[i][j]<me)
    {ie=i; je=j; me=mAdy[i][j];}
    res[ie][je]=res[je][ie]=vis[je]=true;
  }
  return res;
}
```

☒ Árbol de expansión minimal (grafos dirigidos) - Chu-Liu/Edmonds {O(V²)}

☑ Existencia de un ciclo/camino Euleriano (grafos dirigidos) {O(V+E)}

Un grafo dirigido G=⟨V,E⟩ tiene un ciclo Euleriano si es conexo y todo vértice v cumple que inDegree(v)=outDegree(v). Un grafo dirigido G=⟨V,E⟩ tiene un camino Euleriano si es conexo y todo vértice v cumple que inDegree(v)=outDegree(v), excepto dos vértices v_1 y v_2 para los que se tiene que inDegree(v_1)=outDegree(v_1)-1 y inDegree(v_2)=outDegree(v_2)+1.

☑ **Ciclo/Camino Euleriano partiendo de un nodo (grafos dirigidos) {O(V+E)}**

```
int[] euler(int[][] lAdy, int v) { // Lo encuentra si existe
  List<Integer> r=new ArrayList<Integer>(); euler(lAdy,new int[lAdy.length],v,r);
  int t=r.size(),a[]=new int[t],i; for (i=0; i<t; i++) a[i]=r.get(t-1-i); return a;
}
void euler(int[][] lAdy, int[] tams, int v, List<Integer> r)
{while (tams[v]<lAdy[v].length) euler(lAdy,tams,lAdy[v][tams[v]++],r); r.add(v);}
```

☑ **Existencia de un ciclo/camino Euleriano (grafos no dirigidos) {O(V+E)}**

Un grafo no dirigido G=⟨V,E⟩ tiene un ciclo Euleriano si es conexo y todo vértice v tiene grado par. Un grafo no dirigido G=⟨V,E⟩ tiene un camino Euleriano si es conexo y todo vértice v tiene grado par, excepto dos vértices que tienen grado impar.

☑ **Ciclo/Camino Euleriano partiendo de un nodo (grafos no dirigidos) {O(V+E)}**

```
int[] eulerND(int[][] lAdy, int v) { // Lo encuentra si existe
  int n=lAdy.length; List<Integer> r=new ArrayList<Integer>();
  eulerND(lAdy,new int[n],new boolean[n][n],v,r);
  int t=r.size(),a[]=new int[t],i; for (i=0; i<t; i++) a[i]=r.get(t-1-i); return a;
}
void eulerND(int[][] lAdy, int[] tams, boolean[][] vis, int v, List<Integer> r) {
  for (int x; tams[v]<lAdy[v].length; ) if (!vis[v][x=lAdy[v][tams[v]++]])
  {vis[v][x]=vis[x][v]=true; eulerND(lAdy,tams,vis,x,r);}
  r.add(v);
}
```

☑ **Ciclo Hamiltoniano {O(E^V)}**

```
int[] hamilton(int[][] lAdy) {
  int n=lAdy.length,cam[]=new int[n+1]; if (!hamilton(lAdy,new boolean[n],cam,0,0)) return null;
  return cam;
}
boolean hamilton(int[][] lAdy, boolean[] vis, int[] cam, int t, int v) {
  if (vis[v]&&t<cam.length-1) return false; cam[t++]=v; if (t==cam.length) return v==cam[0];
  vis[v]=true; for (int w:lAdy[v]) if (hamilton(lAdy,vis,cam,t,w)) return true;
  vis[v]=false; return false;
}
```

☑ **Sort topológico (Topological Sort) {O(V+E)}**

```
int[] topologicalSort(int[][] lAdy) { // Retorna null si no recibe un DAG
  int n=lAdy.length,state[]=new int[n]; List<Integer> r=new ArrayList<Integer>();
  for (int v=0; v<n; v++) if (state[v]==0&&!dfsTS(lAdy,v,state,r)) return null;
  int t=r.size(),a[]=new int[t],i; for (i=0; i<t; i++) a[i]=r.get(t-1-i); return a;
}
boolean dfsTS(int[][] lAdy, int v, int[] state, List<Integer> r) {
  state[v]=1;
  for (int w:lAdy[v]) if (state[w]==1||(state[w]==0&&!dfsTS(lAdy,w,state,r))) return false;
  state[v]=2; r.add(v); return true;
}
```

☑ **Decidir si un grafo es un DAG (Directed Acyclic Graph) {O(V+E)}**

```
boolean isDag(int[][] lAdy) {return topologicalSort(lAdy)!=null;}
```

☑ **Caminos más largos en un DAG {O(V+E)}**

```
int[] caminosMasLargosDAG(int[][] lAdy) {
  int n=lAdy.length,res[]=new int[n];
  for (int v:topologicalSort(lAdy)) for (int u:lAdy[v]) res[u]=Math.max(res[u],res[v]+1);
  return res;
}
```

☑ **Rutas más cortas desde un nodo en un DAG {O(V+E)}**

```
double[] rutasMasCortasDAG(double[][] mAdy, int[][] lAdy, int v) {
  int n=lAdy.length; double r[]=new double[n]; Arrays.fill(r,Double.POSITIVE_INFINITY); r[v]=0;
  // Para considerar varias fuentes v1,...,vk inicialice r[v1],...,r[vk] en 0.
  for (int w:topologicalSort(lAdy)) for (int u:lAdy[w]) r[u]=Math.min(r[u],r[w]+mAdy[w][u]);
  return r;
}
```

☒ **Minimum path cover en un DAG**

☑ **Strongly connected components en grafos dirigidos - Kosaraju [R. Kosaraju (1978)] {O(V+E)}**

```
int[][] stronglyConnectedComponents(int[][] lAdy) {
  int n=lAdy.length,lAdyT[][]=new int[n][],tams[]=new int[n]; boolean vis[]=new boolean[n];
  List<Integer> r=new ArrayList<Integer>(),s=new ArrayList<Integer>();
  for (int v=0; v<n; v++) if (!vis[v]) dfsSCC(lAdy,v,vis,r);
  for (int[] arr:lAdy) for (int j:arr) tams[j]++;
  for (int j=0; j<n; tams[j]=0,j++) lAdyT[j]=new int[tams[j]];
  for (int i=0; i<n; i++) for (int j:lAdy[i]) lAdyT[j][tams[j]++]=i;
  List<int[]> res=new ArrayList<int[]>(); Collections.reverse(r); Arrays.fill(vis,false);
  for (int v:r) if (!vis[v]) {
    s.clear(); dfsSCC(lAdyT,v,vis,s);
    int a[]=new int[s.size()],i=0; for (int x:s) a[i++]=x; res.add(a);
  }
  return res.toArray(new int[0][]);
}
void dfsSCC(int[][] lAdy, int v, boolean[] vis, List<Integer> r)
{vis[v]=true; for (int w:lAdy[v]) if (!vis[w]) dfsSCC(lAdy,w,vis,r); r.add(v);}
```

☒ **Strongly connected components en grafos dirigidos - Tarjan [R. Tarjan (1972?)] {O(V+E)}**

☑ **Bridges en grafos dirigidos {O(V+E)}**

```
// Un puente es un arco que al quitarse desconecta al grafo
int cnt,ord[],low[];
int[][] bridges(int[][] lAdy) { // Tomado de [Sed2004]
  int n=lAdy.length; ord=new int[n]; low=new int[n]; List<int[]> res=new ArrayList<int[]>();
  for (int v=0; v<n; v++) if (ord[v]==0) {cnt=1; bridges(v,v,lAdy,res);}
  return res.toArray(new int[0][]);
}
void bridges(int v, int w, int[][] lAdy, List<int[]> res) {
  low[w]=ord[w]=cnt++;
  for (int t:lAdy[w])
    if (ord[t]==0) {
      bridges(w,t,lAdy,res); low[w]=Math.min(low[w],low[t]);
      if (low[t]==ord[t]) res.add(new int[]{w,t});
    }
    else if (t!=v) low[w]=Math.min(low[w],ord[t]);
}
```

☑ **Articulation Points en grafos dirigidos {O(V+E)}**

```
// Un punto de articulación es un vértice que al quitarse desconecta al grafo
int cnt,ord[],low[];
boolean[] articulationPoints(int[][] lAdy) { // Basado en [Sed2004]
  int n=lAdy.length; ord=new int[n]; low=new int[n]; boolean[] res=new boolean[n];
  for (int v=0; v<n; v++) if (ord[v]==0) {cnt=1; articulationPoints(v,v,lAdy,res);}
  return res;
}
void articulationPoints(int v, int w, int[][] lAdy, boolean[] res) {
  low[w]=ord[w]=cnt++;
  for (int t:lAdy[w]) {
    if (ord[t]==0) {
      articulationPoints(w,t,lAdy,res); low[w]=Math.min(low[w],low[t]);
      if ((ord[w]==1&&ord[t]!=2)||(ord[w]!=1&&low[t]>=ord[w])) res[w]=true;
    }
    else if (t!=v) low[w]=Math.min(low[w],ord[t]);
  }
}
```

☑ **Biconnected Components en grafos dirigidos {O(V+E)}**

```
// Descomposición de un grafo en componentes maximales que no tienen puntos de articulación
// (hay mínimo dos caminos disyuntos entre todo par de vértices en cada componente biconectada)
int cnt,ord[],low[],stack[][],tS;
int[][][] biconnectedComponents(int[][] lAdy) { // Basado en [Sed2004]
  int n=lAdy.length; ord=new int[n]; low=new int[n]; stack=new int[n][]; tS=0;
  List<int[][]> res=new ArrayList<int[][]>();
  for (int v=0; v<n; v++) if (ord[v]==0) {cnt=1; tS=0; biconnectedComponents(v,v,lAdy,res);}
  return res.toArray(new int[0][][]);
}
boolean cmp(int[] a, int[] b) {return (a[0]==b[0]&&a[1]==b[1])||(a[0]==b[1]&&a[1]==b[0]);}
void biconnectedComponents(int v, int w, int[][] lAdy, List<int[][]> res) {
```

```
  low[w]=ord[w]=cnt++;
  for (int t:lAdy[w]) if (t!=v) {
    if (ord[t]<ord[w]) stack[tS++]=new int[]{w,t};
    if (ord[t]==0) {
      biconnectedComponents(w,t,lAdy,res); low[w]=Math.min(low[w],low[t]);
      if (low[t]>=ord[w]) {
        int tvS=tS,e[]={w,t};
        while (tS>=1&&!cmp(stack[tS-1],e)) tS--;
        if (tS>=1) tS--; res.add(Arrays.copyOfRange(stack,tS,tvS));
      }
    }
    else low[w]=Math.min(low[w],ord[t]);
  }
}
```

☑ **Determinar si un grafo es bipartito (bicoloreable) - BFS {O(V+E)}**

```
boolean grafoBipartitoBFS(int[][] lAdy) {
  int n=lAdy.length,res[]=new int[n],queue[]=new int[n],i,t,u,v,inf=Integer.MAX_VALUE;
  for (Arrays.fill(res,inf),u=0; u<n; u++) if (res[u]==inf) {
    t=0; res[queue[t++]=u]=0;
    for (i=0; i<t; i++) for (int w:lAdy[v=queue[i]])
      if (res[w]==inf) res[queue[t++]=w]=res[v]+1; else if ((res[w]+res[v])%2==0) return false;
  }
  return true;
}
```

☑ **Determinar si un grafo es bipartito (bicoloreable) - DFS {O(V+E)}**

```
boolean grafoBipartitoDFS(int[][] lAdy) {
  int n=lAdy.length,cs[]=new int[n];
  for (int v=0; v<n; v++) if (cs[v]==0&&!dfsGP(lAdy,v,cs,1)) return false;
  return true;
}
boolean dfsGP(int[][] lAdy, int v, int[] cs, int c) { // c: color
  cs[v]=c; c=3-c;
  for (int w:lAdy[v]) if ((cs[w]!=0&&cs[w]!=c)||(cs[w]==0&&!dfsGP(lAdy,w,cs,c))) return false;
  return true;
}
```

☑ **Teorema de König para grafos bipartitos - [D. König (1914)]**

En cualquier grafo bipartito, el número de arcos en un maximum matching es igual al número de vértices en un minimum vertex cover y el número de vértices en un minimum vertex cover es igual al número total de vértices menos el número de vértices en un maximum independent set (un vertex cover en un grafo no dirigido G=⟨V,E⟩ es un subconjunto de V tal que todo arco en E tiene un extremo en V; un matching en un grafo G=⟨V,E⟩ es un subconjunto de arcos que no comparten extremos dos a dos; un independent set en un grafo G=⟨V,E⟩ es un conjunto de vértices que no son adyacentes).

☑ **Maximum matching en grafos bipartitos no dirigidos**

Divida el conjunto de vértices V del grafo bipartito en dos conjuntos disyuntos V_1 y V_2 tales que todo arco en E conecte un nodo en V_1 con un nodo en V_2. Cree un nodo n_1 conectado a todos los vértices de V_1 y un nodo n_2 conectado a todos los vértices de V_2. El maximum matching está dado por el flujo máximo entre los nodos n_1 y n_2 donde todos los arcos tienen capacidad 1.

☑ **Teorema de Dilworth para Posets (Partially Ordered Sets) - [R. P. Dilworth (1950)]**

En cualquier conjunto parcialmente ordenado, el número de elementos de la anticadena más grande es igual al tamaño de la partición más pequeña del conjunto en una familia de cadenas.

☒ **Maximum independent sets en árboles**

FLUJO EN REDES

☑ **Flujo máximo - Ford-Fulkerson/Edmonds-Karp [J. Edmonds, R. Karp (1972)] {O(V*E²)}**

```
double edmondsKarp(double[][] capacity, int v1, int v2) { // residualCapacity=capacity-flow
  int n=capacity.length,lAdy[][]=new int[n][],ants[]=new int[n],queue[]=new int[n],v,u;
  double f=0d,m,flow[][]=new double[n][n],minCap[]=new double[n]; List h[]=new List[n];
  for (u=0; u<n; u++) h[u]=new ArrayList<Integer>();
  for (u=0; u<n; u++) for (v=0; v<n; v++) if (capacity[u][v]>1e-10) {h[u].add(v); h[v].add(u);}
  for (u=0; u<n; u++) lAdy[u]=toArr(h[u]);
  for (; (m=bfsEK(capacity,flow,lAdy,ants,minCap,queue,v1,v2))>1e-10; f+=m)
    for (v=v2,u=ants[v]; v!=v1; v=u,u=ants[v]) {flow[u][v]+=m; flow[v][u]-=m;}
```

```
  return f;
}
double bfsEK(double[][] capacity, double[][] flow, int[][] lAdy,
             int[] ants, double[] minCap, int[] queue, int v1, int v2) {
  int i,t=0,u; double z; Arrays.fill(ants,-1); ants[v1]=-2; minCap[v1]=Double.POSITIVE_INFINITY;
  for (queue[t++]=v1,i=0; i<t; i++)
    for (int v:lAdy[u=queue[i]]) if ((z=capacity[u][v]-flow[u][v])>1e-10&&ants[v]==-1)
    {ants[v]=u; minCap[v]=Math.min(minCap[u],z); if (v==v2) return minCap[v2]; queue[t++]=v;}
  return 0d;
}
int[] toArr(List<Integer> p) {int r[]=new int[p.size()],i=0; for (int x:p) r[i++]=x; return r;}
```

☑ **Teorema Max-flow/Min-cut - [P.Elias/A.Feinstein/C.E.Shannon/L.R.Ford/D.R.Fulkerson (1956)]**
La máxima cantidad de flujo entre dos nodos v_1 y v_2 es igual al costo de un corte mínimo que separe al nodo v_1 del nodo v_2 (Un corte en un grafo $G=\langle V,E\rangle$ es una partición de V en dos conjuntos A y B. Se dice que todo arco (a,b) en E tal que $a\in A$ y $b\in B$ atraviesa el corte. El costo de un corte es la suma de los costos de los arcos que lo atraviesan. En flujo en redes, el costo de un corte es la suma de los costos de los arcos que atraviesan el corte y que van de la parte de la fuente a la parte del destino.).

☑ **Flujo máximo de costo mínimo (minimum cost maximum flow (mincost maxflow))**
Encuentre un flujo máximo (Edmonds-Karp). Mientras hayan ciclos de costo negativo (usar Bellman-Ford) en la red residual calibrada con los costos, aumente el flujo a lo largo del ciclo.

☑ **Máximo número de caminos disyuntos entre dos vértices**
Corresponde al máximo flujo entre los dos vértices donde todo arco tiene capacidad 1.

PROGRAMACIÓN DINÁMICA

☑ **Sufijo común más largo (Longest common suffix) {O(n*m)}**
Sufijo común más largo de dos cadenas a[0..n-1],b[0..m-1]:

$$\begin{aligned}&\text{answer}=\text{sufcml}(n,m)\\&\text{sufcml}(i,j)=\text{sufcml}(i-1,j-1)+1 && \text{si } i>0\wedge j>0\wedge a[i-1]=b[j-1]\\&\qquad=0 && \text{de lo contrario}\end{aligned}$$

☑ **Subcadena común más larga (Longest common substring) {O(n*m)}**
Subcadena común más larga de dos cadenas a[0..n-1],b[0..m-1]:

$$\text{answer}=(\uparrow i,j\,|\,1\le i\le n\wedge 1\le j\le m:\text{sufcml}(i,j))$$

☒ **Supercadena común más corta entre k cadenas (Shortest common superstring)**

☑ **Subsecuencia común más larga (Longest common subsequence) {O(n*m)}**
Subsecuencia común más larga de dos cadenas a[0..n-1],b[0..m-1]:

$$\begin{aligned}&\text{answer}=\text{subcml}(n,m)\\&\text{subcml}(i,j)=0 && \text{si } i=0\vee j=0\\&\qquad=\text{subcml}(i-1,j-1)+1 && \text{si } i>0\wedge j>0\wedge a[i-1]=b[j-1]\\&\qquad=\text{subcml}(i,j-1)\uparrow\text{subcml}(i-1,j) && \text{si } i>0\wedge j>0\wedge a[i-1]\neq b[j-1]\end{aligned}$$

☑ **Supersecuencia común más corta (Shortest common supersequence) {O(n*m)}**
Supersecuencia común más corta de dos cadenas a[0..n-1],b[0..m-1]:

$$\begin{aligned}&\text{answer}=\text{subcmc}(n,m)=n+m-\text{subcml}(n,m)\\&\text{subcmc}(i,j)=i\uparrow j && \text{si } i=0\vee j=0\\&\qquad=\text{subcmc}(i-1,j-1)+1 && \text{si } i>0\wedge j>0\wedge a[i-1]=b[j-1]\\&\qquad=1+(\text{subcmc}(i,j-1)\downarrow\text{subcmc}(i-1,j)) && \text{si } i>0\wedge j>0\wedge a[i-1]\neq b[j-1]\end{aligned}$$

☑ **Distancia de Levenshtein {O(n*m)}**
Distancia de edición entre dos cadenas a[0..n-1],b[0..m-1] (mínimo número de modificaciones, inserciones y eliminaciones para transformar una cadena en otra):

$$\begin{aligned}&\text{answer}=\text{dl}(n,m)\\&\text{dl}(i,j)=i\uparrow j && \text{si } i=0\vee j=0\\&\qquad=(\text{dl}(i,j-1)+1)\downarrow(\text{dl}(i-1,j)+1)\downarrow\text{dl}(i-1,j-1) && \text{si } i>0\wedge j>0\wedge a[i-1]=b[j-1]\\&\qquad=(\text{dl}(i,j-1)+1)\downarrow(\text{dl}(i-1,j)+1)\downarrow(\text{dl}(i-1,j-1)+1) && \text{si } i>0\wedge j>0\wedge a[i-1]\neq b[j-1]\end{aligned}$$

☑ **Problema de la pavimentación {O(n*m)}**
Número de maneras de pavimentar un camino de dimensiones $1\times n$ con losas de dimensiones $1\times[1..m]$:

$$\begin{aligned}&\text{answer}=\text{nmpc}(n)\\&\text{nmpc}(i)=1 && \text{si } i=0\\&\qquad=(\sum k\,|\,1\le k\le m\downarrow i:\text{nmpc}(i-k)) && \text{si } i>0\end{aligned}$$

☑ **Mínimo número de monedas para retornar un cambio (Coin change) {O(x*n)}**
Mínimo número de monedas con denominaciones d[0],d[1],...,d[n-1] para devolver una cantidad x:

$$\begin{aligned}&\text{answer}=\text{mnmc}(x)\\&\text{mnmc}(c)=0 && \text{si } c=0\end{aligned}$$

```
        = (↓k|0≤k<n∧d[k]≤c:mnmc(c-d[k]))+1   si c>0  donde la identidad de ↓ es ∞
```

☑ **Número de maneras para retornar un cambio con monedas (Coin counting) {O(x*n)}**

Número de maneras para dar una cantidad x con monedas de denominaciones d[0],d[1],...,d[n-1]:

```
  ① answer=nmrc(x)
  nmrc(c) = 1                                  si c=0
          = (∑k|0≤k<n∧d[k]≤c:nmrc(c-d[k]))     si c>0  donde la identidad de ∑ es 0
  ② answer=nmrc(x,n)
  nmrc(c,i) = 1                                si c=0
            = 0                                si c>0∧i=0
            = nmrc(c,i-1)                      si c>0∧i>0∧d[i-1]>c
            = nmrc(c,i-1)+nmrc(c-d[i-1],i)     si c>0∧i>0∧d[i-1]≤c
```

☑ **Decidir si algunos números de una lista de naturales suman un valor {O(x*n)}**

Decidir si algunos elementos de un arreglo de naturales d[0..n-1] suman una cantidad natural x:

```
  answer=aean(x,n)
  aean(c,i) = true                             si c=0
            = false                            si c>0∧i=0
            = aean(c,i-1)                      si c>0∧i>0∧d[i-1]>c
            = aean(c,i-1)∨aean(c-d[i-1],i-1)   si c>0∧i>0∧d[i-1]≤c
```

☑ **Problema del morral {O(m*n)}**

Se tienen n>0 objetos con pesos p[0],p[1],…,p[n-1] y utilidades u[0],u[1],…,u[n-1] y un morral con capacidad m. Máxima utilidad conseguible:

```
  answer=mr(m,n)
  mr(c,i) = 0                                          si i=0
          = mr(c,i-1)                                  si i>0∧p[i-1]>c
          = mr(c,i-1)↑(mr(c-p[i-1],i-1)+u[i-1])        si i>0∧p[i-1]≤c
```

☑ **Problema del CD {O(m*n)}**

Se tiene un CD con n>0 pistas con duraciones d[0],d[1],…,d[n-1] y un casette con capacidad m. Máximo tiempo de grabación: Véase el problema del morral con p=d y u=d.

☑ **Matrix chain multiplication {O(n³)}**

Dada una lista de n matrices con dimensiones d[0]×d[1],d[1]×d[2],...,d[n-1]×d[n], dar el mínimo número de multiplicaciones necesarias para multiplicarlas en secuencia.

```
  answer=mcm(0,n)
  mcm(i,j) = 0                                          si i≥j-1
           = (↓k|i+1≤k≤j-1:mcm(i,k)+mcm(k,j)+d[i]*d[k]*d[j])  si i<j-1
```

☑ **Mayor suma alcanzable por subarreglos - Algoritmo de Kadane {O(n)}**

```
long kadane(long[] arr) {
  long act=0,r=0;
  for (long v:arr) {act+=v; if (act>r) r=act; if (act<0) act=0;}
  return r;
}
```

☑ **Mayor suma alcanzable por submatrices - Algoritmo de Kadane {O(n³)}**

```
long kadane2D(long[][] mat) {
  int m=mat.length,n=m==0?0:mat[0].length; long r=0,sumas[][]=new long[m][n],arr[]=new long[n];
  for (int i=0; i<m; i++) for (int j=0; j<n; j++) sumas[i][j]=mat[i][j]+(i>0?sumas[i-1][j]:0);
  for (int i1=0; i1<m; i1++) for (int i2=i1; i2<m; i2++) {
    for (int j=0; j<n; j++) arr[j]=sumas[i2][j]-(i1>0?sumas[i1-1][j]:0);
    r=Math.max(r,kadane(arr));
  }
  return r;
}
```

☑ **Subsecuencia creciente más larga (Longest increasing subsequence) {O(n²)}**

```
int lis1(long[] arr) {
  int n=arr.length,maxs[]=new int[n],res=0,i,j;
  for (i=0; i<n; res=Math.max(res,maxs[i]),i++)
    for (maxs[i]=1,j=0; j<i; j++) if (arr[i]>arr[j]) maxs[i]=Math.max(maxs[i],maxs[j]+1);
  return res;
}
```

☑ **Subsecuencia creciente más larga (Longest increasing subsequence) {O(n*log₂(n))}**

```
int lis2(long[] arr) {
  int n=arr.length,res=0; long val[]=new long[n+1]; val[0]=Long.MIN_VALUE; //inds[0]=0
  for (long v:arr) { // Sea i el índice de v en arr
    int j=Arrays.binarySearch(val,0,res+1,v); j=(j<0?-j-1:j)-1; //ants[i]=inds[j];
    if (j==res||v<val[j+1]) {val[j+1]=v; res=Math.max(res,j+1);} //inds[j+1]=i;
```

```
  }
  return res;
}
```

GEOMETRÍA COMPUTACIONAL

LOS PUNTOS DE LOS POLÍGONOS SE ENUMERAN EN SENTIDO CONTRARIO AL DE LAS MANECILLAS DEL RELOJ
EL PRIMER PUNTO DE LOS POLÍGONOS NO SE REPITE AL FINAL

☑ double[][] to Shape {O(n)}

```
Path2D.Double getShape(double[][] pt) {
  Path2D.Double r=new Path2D.Double(Path2D.WIND_EVEN_ODD); r.moveTo(pt[0][0],pt[0][1]);
  for (int i=1; i<pt.length; i++) r.lineTo(pt[i][0],pt[i][1]);
  r.closePath(); return r;
}
```

☑ Shape to List<double[][]> {O(n)}

```
java.util.List<double[][]> getPolygons(Shape sh) {
  java.util.List<double[][]> r=new ArrayList<double[][]>();
  java.util.List<double[]> z=new ArrayList<double[]>();
  double[] c=new double[6];
  for (PathIterator ph=sh.getPathIterator(null); !ph.isDone(); ph.next()) {
    if (ph.currentSegment(c)!=PathIterator.SEG_CLOSE) z.add(new double[]{c[0],c[1]});
    else {r.add(z.toArray(new double[0][])); z.clear();}
  }
  return r;
}
```

☑ Distancia al cuadrado entre dos puntos {O(1)}

```
double ds(double[] a, double[] b) {return (b[0]-a[0])*(b[0]-a[0])+(b[1]-a[1])*(b[1]-a[1]);}
```

☑ Distancia de punto a línea {O(1)}

```
double distPL(double[] p1, double[] p2, double[] p) {
  return Math.abs((p2[0]-p1[0])*(p1[1]-p[1])-(p2[1]-p1[1])*(p1[0]-p[0]))/Math.sqrt(ds(p1,p2));
}
```

☑ Distancia de punto a segmento {O(1)}

```
double distPS(double[] p1, double[] p2, double[] p) {
  double dP=ds(p1,p2),d1=ds(p1,p),d2=ds(p2,p);
  return (d2+dP<d1||d1+dP<d2)?Math.sqrt(Math.min(d1,d2)):distPL(p1,p2,p);
}
```

☑ Line Intersection 2D {O(1)}

```
double[] intLineas(double x1, double y1, double x2, double y2,
                   double x3, double y3, double x4, double y4) {
  double xa=x2-x1,xb=x4-x3,xc=x1-x3,ya=y2-y1,yb=y4-y3,yc=y1-y3,d=yb*xa-xb*ya,n=xb*yc-yb*xc;
  return Math.abs(d)<1e-11?null:new double[]{x1+xa*n/d,y1+ya*n/d};
}
```

☑ Segment Intersection 2D {O(1)}

```
double[] intSegmentos(double x1, double y1, double x2, double y2,
                      double x3, double y3, double x4, double y4) { // No se sobrelapan
  double[] p1={x1,y1},p2={x2,y2},p3={x3,y3},p4={x4,y4},p12[]={p1,p2},p34[]={p3,p4};
  for (double[] p:p12) if (distPS(p3,p4,p)<1e-11) return p;
  for (double[] p:p34) if (distPS(p1,p2,p)<1e-11) return p;
  double[] p=intLineas(x1,y1,x2,y2,x3,y3,x4,y4);
  return p!=null&&distPS(p1,p2,p)<1e-11&&distPS(p3,p4,p)<1e-11?p:null;
}
```

☑ Circle Intersection 2D {O(1)}

```
double[][] intCirculos(double[] c1, double r1, double[] c2, double r2) {
  if (r2<r1) return intCirculos(c2,r2,c1,r1);
  double d=Math.sqrt(ds(c1,c2));
  if (d<1e-11||d<r2-r1-1e-11||d>r1+r2+1e-11) return null;
  double u=(d*d-r1*r1+r2*r2)/(d*2),v=Math.sqrt(Math.max(r2*r2-u*u,0));
  double dx=(c1[0]-c2[0])/d,dy=(c1[1]-c2[1])/d;
  return new double[][]{{c2[0]+dx*u-dy*v,c2[1]+dy*u+dx*v},{c2[0]+dx*u+dy*v,c2[1]+dy*u-dx*v}};
}
```

☒ Sphere Intersection 3D

☑ Perímetro de un polígono {O(n)}

```
double perimetro(double[][] pt) {
  double r=0d;
```

```
  for (int i=0,t=pt.length; i<t; i++) r+=Math.sqrt(ds(pt[i],pt[i+1==t?0:i+1]));
  return r;
}
```

☑ **Área de un polígono {O(n)}**

```
double area(double[][] pt) {
  double r=0d; int t=pt.length;
  for (int i=0,j=1; i<t; i++,j=j+1==t?0:j+1) r+=pt[i][0]*pt[j][1]-pt[i][1]*pt[j][0];
  return Math.abs(r)/2;
}
```

☑ **Área de un polígono con puntos coplanares en 3D {O(n)}**

Utilice la fórmula en 2D considerando las coordenadas en z de los puntos en el cálculo de la magnitud de los productos cruz.

☑ **Centroide (centro de masa) de un polígono {O(n)}**

```
double[] centroide(double[][] pt) {
  double p[]={0d,0d},d=area(pt)*6;
  for (int i=0,j=1,t=pt.length; i<t; i++,j=j+1==t?0:j+1)
    for (int k=0; k<2; k++) p[k]+=(pt[i][k]+pt[j][k])*(pt[i][0]*pt[j][1]-pt[j][0]*pt[i][1]);
  return new double[]{p[0]/d,p[1]/d};
}
```

☑ **Punto dentro de polígono? {O(n)}**

```
boolean dentroPoligono(double[][] pt, double[] p, boolean bd) { // bd: con borde?
  boolean b=false;
  for (int i=0,j=1,t=pt.length; i<t; i++,j=j+1==t?0:j+1) {
    if (distPS(pt[i],pt[j],p)<1e-11) return bd;
    if (((pt[j][1]<=p[1]&&p[1]<pt[i][1])||(pt[i][1]<=p[1]&&p[1]<pt[j][1])) &&
       (p[0]-pt[j][0]<(p[1]-pt[j][1])*(pt[i][0]-pt[j][0])/(pt[i][1]-pt[j][1]))) b=!b;
  }
  return b;
}
```

☑ **Punto dentro de polígono? - Java {¿O(n)?}**

```
boolean dentroPoligono(double[][] pt, double[] p, boolean bd) { // bd: con borde?
  for (int i=0,j=1,t=pt.length; i<t; i++,j=j+1==t?0:j+1)
    if (Line2D.ptSegDist(pt[i][0],pt[i][1],pt[j][0],pt[j][1],p[0],p[1])<1e-11) return bd;
  return getShape(pt).contains(p[0],p[1]);
}
```

☑ **Círculo dados tres o dos puntos {O(1)}**

```
double[] circulo3P(double[] p1, double[] p2, double[] p3)
{return circulo3P(p1[0],p1[1],p2[0],p2[1],p3[0],p3[1]);}
double[] circulo3P(double x1, double y1, double x2, double y2, double x3, double y3) {
  double x4=(x1+x2)/2,y4=(y1+y2)/2,x5=(x3+x2)/2,y5=(y3+y2)/2;
  double c[]=intLineas(x4,y4,x4+y2-y1,y4+x1-x2,x5,y5,x5+y2-y3,y5+x3-x2),cx=c[0],cy=c[1];
  return new double[]{cx,cy,Math.sqrt((cx-x1)*(cx-x1)+(cy-y1)*(cy-y1))};
}
double[] circulo2P(double[] p1, double[] p2)
{return circulo2P(p1[0],p1[1],p2[0],p2[1]);}
double[] circulo2P(double x1, double y1, double x2, double y2) {
  return new double[]{(x1+x2)/2,(y1+y2)/2,Math.sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1))/2};
}
```

☑ **Closest Pair {O(n*log$_2$(n))}**

```
Comparator<double[]> c=new Comparator<double[]>() {
  public int compare(double[] p, double[] q)
  {return Math.abs(p[0]-q[0])>=9.9e-12?(p[0]<q[0]?-1:1):(p[1]!=q[1]?(p[1]<q[1]?-1:1):0);}
};
int f(int n) {return n>=0?n:-n-1;}
double closestPair(double[][] pt) {
  Arrays.sort(pt,c); int k=pt.length-1,z[]=new int[k+1];
  for (int i=k; i>=0; i--) z[i]=i==k||Math.abs(pt[i][0]-pt[i+1][0])>=9.9e-12?i:z[i+1];
  for (int i=0; i<k; i++) if (c.compare(pt[i],pt[i+1])==0) return 0;
  return Math.sqrt(closestPair(pt,0,k,z));
}
double closestPair(double[][] pt, int lI, int lS, int[] z) {
  if (lI>=lS) return Double.POSITIVE_INFINITY;
  int lM=(lI+lS)/2; double inf=Double.NEGATIVE_INFINITY,x=pt[lM][0];
  double r=Math.min(closestPair(pt,lI,lM,z),closestPair(pt,lM+1,lS,z)),d=Math.sqrt(r);
```

```
  for (int i=f(Arrays.binarySearch(pt,lI,lM+1,new double[]{x-d+1e-11,inf},c)); i<=lM; i++)
    for (int j1=lM+1,j2,j; j1<=lS&&pt[j1][0]<pt[i][0]+d; j1=j2+1) {
      j=f(Arrays.binarySearch(pt,j1,(j2=z[j1])+1,new double[]{pt[j1][0],pt[i][1]},c));
      for (int k=0; k<2; k++) if (j1<=j-k&&j-k<=j2) r=Math.min(r,ds(pt[i],pt[j-k]));
    }
  return r;
}
```

☑ **Convex Hull - Graham Scan [Ronald Graham (1972)] {O(n*log₂(n))}**

```
int sgn(double x) {return Math.abs(x)>9.9e-12?(x<0?-1:1):0;}
double cruz(double[] a, double[] b) {return a[0]*b[1]-a[1]*b[0];}
double cruz(double[] a, double[] b, double[] c) {return cruz(a,b)+cruz(b,c)+cruz(c,a);}
double[][] convexHullGS(double[][] pt, boolean bd) { // bd: con borde?
  int h=pt.length,i=h,t=0; double v[]=null,w[],r[][]=new double[h][];
  for (double[] p:pt) if (v==null||p[1]<v[1]||(p[1]==v[1]&&p[0]>v[0])) v=p; v=v.clone();
  for (double[] p:pt) {p[0]-=v[0]; p[1]-=v[1];}
  Arrays.sort(pt,new Comparator<double[]>() {public int compare(double[] a, double[] b)
  {double cz=cruz(b,a); return sgn(sgn(cz)!=0?cz:a[0]*a[0]+a[1]*a[1]-b[0]*b[0]-b[1]*b[1]);}});
  if (bd) while (i-1>=0&&cruz(pt[h-1],pt[i-1])==0) i--;
  if (bd) for (int k=i; k<(i+h)/2; k++) {w=pt[k]; pt[k]=pt[h-1-k+i]; pt[h-1-k+i]=w;}
  for (double[] p:pt) {while (t>=2&&sgn(cruz(r[t-1],p,r[t-2]))<(bd?0:1)) t--; r[t++]=p;}
  for (double[] p:pt) {p[0]+=v[0]; p[1]+=v[1];}
  return Arrays.copyOfRange(r,0,t);
}
```

☑ **Minimal (~Smallest) Enclosing Circle (~Disc) - {expected: O(n); worst: O(n³)}**

```
double[] mecR(double[][] pt) { // Randomized algorithm
  int t=pt.length,i,j,k; double[] mc; if (t==1) return circulo2P(pt[0],pt[0]);
  for (mc=circulo2P(pt[0],pt[1]),i=2; i<t; i++) if (Math.sqrt(ds(pt[i],mc))>mc[2]+1e-11)
    for (mc=circulo2P(pt[0],pt[i]),j=1; j<i; j++) if (Math.sqrt(ds(pt[j],mc))>mc[2]+1e-11)
      for (mc=circulo2P(pt[i],pt[j]),k=0; k<j; k++) if (Math.sqrt(ds(pt[k],mc))>mc[2]+1e-11)
        mc=circulo3P(pt[i],pt[j],pt[k]);
  return mc;
}
```

☑ **Minimal (~Smallest) Enclosing Circle (~Disc) - [Pr. Chrystal (1885)] {O(n²)}**

```
double[] mec(double[][] pt) {
  pt=convexHullGS(pt,false); int t=pt.length; boolean g1=false,g2=false;
  if (t==1) return new double[]{pt[0][0],pt[0][1],0};
  for (int u=0,v=1,w=-1; true; v=g1?w:v,u=g2?w:u,w=-1) {
    double max=-2,aw=0,bw=0,cw=0;
    for (int i=0; i<t; i++) if (i!=u&&i!=v) {
      double a=ds(pt[i],pt[u]),b=ds(pt[i],pt[v]),c=ds(pt[u],pt[v]),d=(a+b-c)/Math.sqrt(a*b);
      if (d>max) {max=d; w=i; aw=a; bw=b; cw=c;}
    }
    if (max<=0) return circulo2P(pt[u],pt[v]);
    g1=bw+cw<aw; g2=cw+aw<bw; if (!g1&&!g2) return circulo3P(pt[u],pt[v],pt[w]);
  }
}
```

☑ **Anchura mínima de un conjunto de puntos {O(n*log₂(n))}**

```
double anchura(double[][] pt) {
  pt=convexHullGS(pt,false); int t=pt.length,k=2; double r=Double.POSITIVE_INFINITY;
  if (t>2) for (int i=0,j=1; i<t; i++,j=j+1==t?0:j+1) {
    while (distPL(pt[i],pt[j],pt[k+1==t?0:k+1])>distPL(pt[i],pt[j],pt[k])) k=k+1==t?0:k+1;
    r=Math.min(r,distPL(pt[i],pt[j],pt[k]));
  }
  return t<=2?0:r;
}
```

☑ **Diámetro de un conjunto de puntos {O(n*log₂(n))}**

```
double diametro(double[][] pt) {
  pt=convexHullGS(pt,false); int t=pt.length,k=1; double r=0;
  if (t>1) for (int i=0; i<t; i++) {
    while (ds(pt[i],pt[k+1==t?0:k+1])>ds(pt[i],pt[k])) k=k+1==t?0:k+1;
    r=Math.max(r,ds(pt[i],pt[k]));
  }
  return Math.sqrt(r);
}
```

☑ **Line Segment Intersection - Plane Sweep Algorithm [Bentley-Ottmann (1979)]** {O((n+k)*log₂(n))}

```
// INESTABLE CON LA PRECISIÓN
int sgn(double v) {return Math.abs(v)>1e-11?(v<0?-1:1):0;}
double getM(double[] a, double[] b) {return (b[1]-a[1])/(b[0]-a[0]);}
double getY(double[] a, double[] b, double x) {return a[1]+getM(a,b)*(x-a[0]);}
double[] rot(double[] a, int f) {
  double ang=Math.atan2(a[1],a[0])+Math.E*f,d=Math.sqrt(a[0]*a[0]+a[1]*a[1]);
  return new double[]{d*Math.cos(ang),d*Math.sin(ang)};
}
TreeMap<Double,Set<Integer>> events=new TreeMap<Double,Set<Integer>>(new Comparator<Double>()
{public int compare(Double x1, Double x2) {return sgn(x1-x2);}});
void add(double x, int i)
{Set<Integer> s=events.get(x); if (s==null) events.put(x,s=new TreeSet<Integer>()); s.add(i);}
void notify(double[][][] sgs, double x, int i, int j) {
  if (i<0||j<0) return;
  double[] p1=sgs[i][0],p2=sgs[i][1],p3=sgs[j][0],p4=sgs[j][1];
  double[] p=intSegmentos(p1[0],p1[1],p2[0],p2[1],p3[0],p3[1],p4[0],p4[1]);
  if (p!=null&&sgn(p[0]-x)>=0) {add(p[0],i); add(p[0],j);}
}
double[][] bentleyOttmann(double[][][] segments) { // No debe haber dos segmentos sobrelapados
  class ComparatorSg implements Comparator<double[][]> {
    double x=0d;
    public int compare(double[][] sg1, double[][] sg2) {
      int cY=sgn(getY(sg1[0],sg1[1],x)-getY(sg2[0],sg2[1],x));
      int cM=sgn(getM(sg1[0],sg1[1])-getM(sg2[0],sg2[1]));
      return cY!=0?cY:(cM!=0?cM:sgn(sg1[0][0]-sg2[0][0]));
    }
  }; ComparatorSg c=new ComparatorSg();
  TreeMap<double[][],Integer> sweepLine=new TreeMap<double[][],Integer>(c);
  int n=segments.length; double sgs[][][]=new double[n][][],ult[]=null; events.clear();
  for (int i=0; i<n; i++) {
    double[][] sgA={rot(segments[i][0],1),rot(segments[i][1],1)},sgB={sgA[1],sgA[0]};
    for (double[] p:(sgs[i]=sgA[0][0]>sgA[1][0]?sgB:sgA)) add(p[0],i);
  }
  java.util.List<double[]> res=new ArrayList<double[]>();
  while (!events.isEmpty()) {
    Map.Entry<Double,Set<Integer>> e=events.pollFirstEntry(); Set<Integer> s=e.getValue();
    double sgI[][]=sgs[s.iterator().next()],x=e.getKey(),y=getY(sgI[0],sgI[1],x),p[]={x,y};
    double q[]=rot(p,-1); for (int id:s) sweepLine.remove(sgs[id]); c.x=x;
    if (s.size()>1&&(ult==null||sgn(Math.sqrt(ds(p,ult)))!=0)) {res.add(q); ult=p;}
    Map.Entry<double[][],Integer> eS=sweepLine.ceilingEntry(sgI),eI=sweepLine.lowerEntry(sgI);
    int j=eS==null?-1:eS.getValue(),i=eI==null?-1:eI.getValue();
    for (int k:s) if (sgn(sgs[k][1][0]-x)>0) {
      notify(sgs,x,j,k); notify(sgs,x,i,k); sweepLine.put(sgs[k],k); p=null;
    }
    if (p!=null) notify(sgs,x,j,i);
  }
  return res.toArray(new double[0][]);
}
```

☑ **Determinar si un rectángulo cabe dentro de otro - Carver [W. Carver]** {O(1)}

```
boolean cabe(long p, long q, long a, long b) { // El rectángulo pXq cabe en el rectángulo aXb?
  long x,y,z,w; if (p<q) {x=p; p=q; q=x;} if (a<b) {x=a; a=b; b=x;}
  if (p<=a&&q<=b) return true;
  if (p==q) return b>=q;
  x=2*p*q*a; y=p*p-q*q; z=p*p+q*q; w=z-a*a;
  return p>a&&1d*b*z>=x+y*Math.sqrt(w)-1e-10;
}
```

☑ **Máximo número de segmentos disyuntos dos a dos en un conjunto (1D)** {O(n*log₂(n))}

```
int sgn(double x) {return x!=0?(x<0?-1:1):0;}
int maxSegmentosDisyuntos(double[][] sgs) {
  Arrays.sort(sgs,new Comparator<double[]>() {
    public int compare(double[] a, double[] b)
    {return -sgn(sgn(a[0]-b[0])!=0?a[0]-b[0]:a[1]-b[1]);}
  });
```

```
  double ult=Double.POSITIVE_INFINITY; int res=0;
  for (double[] sg:sgs) if (sg[1]<ult-1e-15) {ult=sg[0]; res++;}
  return res;
}
```

☒ Voronoi Diagram
☒ Delaunay Triangulation
☒ Minimum Width Annuli
☒ Largest Empty Circle
☒ Euclidean Minimum Spanning Tree
☒ Árbol de Steiner
☒ Shortest length triangulation
☑ Teorema de Pick - [Georg Alexander Pick (1899)]

Dado un polígono simple cuyos puntos están sobre coordenadas enteras, se tiene que A=i+b/2-1 donde A es el área del polígono, i es el número de puntos con coordenadas enteras dentro del polígono y b es el número de puntos con coordenadas enteras en el borde del polígono.

☒ Fórmulas varias

θ=arccos(u●v/(|u|*|v|))
[Lados: a,b,c][Ángulos: α,β,γ][Alturas: x,y,z]...
p=a+b+c s=p/2 A=sqrt(s(s-a)(s-b)(s-c))=ax/2 a/sen(α)=b/sen(β)=c/sen(γ) $a^2=b^2+c^2-2bc*\cos(\alpha)$
$a=b*\cos(\gamma)+c*\cos(\beta)$ A=bc*sen(α)/2 r=A/s R=abc/(4a)
$\text{sen}(\alpha/2)=\sqrt{((s-b)(s-c)/(bc))}$ $\cos(\alpha/2)=\sqrt{(s(s-a)/(bc))}$ $\tan(\alpha/2)=\sqrt{((s-b)(s-c)/(s(s-a)))}$
$A=x^2y^2z^2/\sqrt{((yz+xz+xy)(-yz+xz+xy)(yz-xz+xy)(yz+xz-xy))}$
Coordenadas rectangulares a polares: $x=r*\cos(\varphi)*\cos(\theta)$ $y=r*\cos(\varphi)*\sin(\theta)$ $z=r*\sin(\varphi)$
Coordenadas polares a rectangulares: $\tan(\theta)=y/x$ $\tan(\varphi)=z/\sqrt{(x^2+y^2)}$ $r=\sqrt{(x^2+y^2+z^2)}$

NÚMEROS (MCD, BINOMIAL, PRIMOS, FÓRMULAS, ...)

π(n)=PrimeCountingFuntion(n)=#primes([2..n])
$\xi(n)=e_1+e_2+...+e_k$ donde $n=p_1^{e1}*p_2^{e2}*...*p_k^{ek}$ con $p_1,p_2,...,p_k$ primos
$\zeta(n)=k$ donde $n=p_1^{e1}*p_2^{e2}*...*p_k^{ek}$ con $p_1,p_2,...,p_k$ primos

☑ Máximo común divisor - Algoritmo de Euclides [Euclides (300AC)] {$O(\log_2(a\uparrow b))$}

```
long gcd(long a, long b) {long t; while (b!=0) {t=b; b=a%b; a=t;} return a;}
```

☑ Máximo común divisor precalculado {$O(M^2)$}

```
int[][] gcd(int M) { // [0..M][0..M]
  int r[][]=new int[M+1][M+1],i,j;
  for (i=0; i<=M; i++) for (j=0; j<=M; j++) r[i][j]=i==0?j:(j==0?i:(i>j?r[i%j][j]:r[j%i][i]));
  return r;
}
```

☑ Máximo común divisor - Algoritmo extendido de Euclides [Euclides (300AC)] {$O(\log_2(a\uparrow b))$}

```
long[] gcdExtendido(long a, long b) { // answer[0]=gcd(a,b), answer[1]*a+answer[2]*b=gcd(a,b)
  boolean bs=a<b; long xAnt=1,yAnt=0,x=0,y=1;
  if (bs) {long tmp=a; a=b; b=tmp;}
  while (b!=0) {
    long q=a/b,r=a%b,xTmp=xAnt-q*x,yTmp=yAnt-q*y; a=b; b=r; xAnt=x; yAnt=y; x=xTmp; y=yTmp;
  }
  return new long[]{a,bs?yAnt:xAnt,bs?xAnt:yAnt};
}
```

☑ Mínimo común múltiplo {$O(\log_2(a\uparrow b))$}

```
long lcm(long a, long b) {return a*(b/gcd(a,b));}
```

☑ Primos relativos (coprimos) {$O(\log_2(a\uparrow b))$}

```
boolean primosRelativos(long a, long b) {return gcd(a,b)==1L;}
```

☑ Coeficiente binomial {$O(r\downarrow(n-r))$}

```
BigInteger binomial(int n, int r) { // [n][r], 0<=r<=n
  BigInteger res=BigInteger.valueOf(1);
  for (int i=1,t=r>n-r?n-r:r; i<=t; i++)
    res=res.multiply(BigInteger.valueOf(n-i+1)).divide(BigInteger.valueOf(i));
  return res;
}
```

☑ Coeficiente binomial precalculado (fila) {$O(n)$}

```
BigInteger[] binomialF(int n) { // [n][0..n]
  BigInteger arr[]=new BigInteger[n+1]; arr[0]=arr[n]=BigInteger.ONE;
  for (int i=1,t=n>>>1; i<=t; i++)
    arr[i]=arr[n-i]=arr[i-1].multiply(BigInteger.valueOf(n-i+1)).divide(BigInteger.valueOf(i));
  return arr;
```

```
}
```
☑ **Coeficiente binomial precalculado (matriz) {O(N²)}**
```
BigInteger[][] binomialM(int N) { // [0..N][0..N]
  BigInteger mat[][]=new BigInteger[N+1][N+1];
  for (int n=0; n<=N; n++) {
    mat[n][0]=mat[n][n]=BigInteger.ONE;
    for (int i=1,t=n>>>1; i<=t; i++) mat[n][i]=mat[n][n-i]=mat[n-1][i-1].add(mat[n-1][i]);
  }
  return mat;
}
```
☑ **Primos - Criba de Eratóstenes [Eratóstenes] {O(M*log₂(M)*log₂(log₂(M)))}**
```
long[] primos(int M) { // [0..M-1], M>=2
  boolean b[]=new boolean[M]; int i,j,k,c=2;
  for (i=2; (k=i*i)<M; i++) if (!b[i]) for (j=k; j<M; j+=i) if (!b[j]) {b[j]=true; c++;}
  long r[]=new long[M-c]; for (i=2,j=0; i<M; i++) if (!b[i]) r[j++]=i;
  return r;
}
```
☑ **Primos - Criba segmentada de Eratóstenes**
```
long[] primos(long x1, long x2, long[] primos) { // [x1..x2], 2<=x1<=x2<M^2, M=max(primos)
  int T=(int)(x2-x1+1),c=0,i,n=primos.length; long R=(long)(Math.sqrt(x2)+1+1e-5),p,q;
  boolean b[]=new boolean[T];
  for (i=0; i<n&&(p=primos[i])<=R; i++) for (q=Math.max((x1+p-1)/p,2)*p; q<=x2; q+=p)
    if (!b[(int)(q-x1)]) {b[(int)(q-x1)]=true; c++;}
  for (p=x1; p<=R; p++) if (!b[(int)(p-x1)]) for (q=p*p; q<=x2; q+=p)
    if (!b[(int)(q-x1)]) {b[(int)(q-x1)]=true; c++;}
  long r[]=new long[T-c]; for (p=x1,i=0; p<=x2; p++) if (!b[(int)(p-x1)]) r[i++]=p;
  return r;
}
```
☑ **Test de primalidad {O(π(sqrt(x)))}**
```
boolean esPrimo(long x, long[] primos) { // 2<=x<M^2, M=max(primos)
  if ((x&1L)==0L) return x==2L;
  int i,n=primos.length; double R=(long)(Math.sqrt(x)+1+1e-5); long p;
  if (x<=primos[n-1]) return Arrays.binarySearch(primos,x)>=0;
  for (i=0; i<n&&(p=primos[i])<=R; i++) if (x%p==0) return false;
  return true;
}
```
☑ **Descomponer un número en factores primos {O(π(sqrt(x))+log₂(x))}**
```
long[][] factoresPrimos(long x, long[] primos) { // 2<=x<M^2, M=max(primos). INMUTABLES.
  List<long[]> res=new ArrayList<long[]>(); double R=(long)(Math.sqrt(x)+1+1e-5); long p,c;
  for (int i=0,n=primos.length; i<n&&(p=primos[i])<=R; i++) {
    for (c=0; x%p==0; x/=p,c++); if (c>0) res.add(new long[]{p,c});
  }
  if (x>1) res.add(new long[]{x,1});
  return res.toArray(new long[0][]);
}
```
☑ **Multiplicar y dividir números en descomposición prima {O(ζ(x)+ζ(y))}**
```
long[][] mulPowFP(long[][] fP1, long[][] fP2, long exp) { // answer=num(fP1)*num(fP2)^(exp)
  if (exp==0) return fP1;
  int i=0,j=0,n=fP1.length,m=fP2.length; List<long[]> res=new ArrayList<long[]>(n+m);
  while (true) {
    while ((i<n||j<m)&&!(i<n&&j<m&&fP1[i][0]==fP2[j][0])) {
      boolean b=i<n&&(j==m||fP1[i][0]<fP2[j][0]); long[] f=b?fP1[i++]:fP2[j++];
      res.add(b||exp==1?f:new long[]{f[0],f[1]*exp});
    }
    if (i==n&&j==m) break;
    long p=fP1[i][0],e=fP1[i++][1]+exp*fP2[j++][1]; if (e!=0) res.add(new long[]{p,e});
  }
  return res.toArray(new long[0][]);
}
```
☑ **Cantidad de divisores de un número en descomposición prima {O(ζ(x))}**
```
long divisoresFE(long[][] fp) { // fp sin factores primos con exponentes negativos
  long r=1L; for (long[] f:fp) r*=(f[1]+1); return r;
}
```
☑ **Cantidad de factores primos en la descomposición prima {O(M*log₂(M)*log₂(log₂(M)))}**

```
int[] factoresPrimos(int M) { // [0..M-1], M>=2
  int b[]=new int[M]; int i,j,k;
  for (i=2; (k=i*i)<M; i++) if (b[i]==0) for (j=k; j<M; j+=i) if (b[j]==0) b[j]=i;
  for (i=2; i<M; i++) b[i]=b[i]==0?1:b[i/b[i]]+1;
  return b;
}
```

☑ **Factorizaciones de un número**

```
List<String> factorizaciones(int n) {return iterarFc(n,new int[100],0,new ArrayList<String>());}
List<String> iterarFc(int n, int[] divs, int k, List<String> res) {
  for (int d=(k>0?divs[k-1]:2),dT=(int)(Math.sqrt(n)+1e-5); d<=dT; d++) if (n%d==0)
    iterarFc(n/(divs[k]=d),divs,k+1,res);
  if (k>=0) {String s=""; for (int i=0; i<k; i++) s+=divs[i]+"X"; s+=n; res.add(s);}
  return res;
}
```

☑ **Aproximación de Stirling - [Abraham de Moivre, James Stirling]**

$n!\approx raiz(2\pi n)(n/e)^n$

☑ **Teorema pequeño de Fermat**

$a^{p-1}\equiv 1 \pmod p$ donde p es primo y a no es múltiplo de p

☑ **Euler's totient function**

Sea p primo, $p_1,...,p_m$ primos distintos y $n=p_1^{k1}*...*p_m^{km}$.

$\varphi(1)=1 \quad \varphi(p^k)=(p-1)*p^{k-1} \quad \varphi(n)=(p_1-1)*p_1^{k1-1}*...*(p_m-1)*p_m^{km-1}=n*(1-1/p_1)*...*(1-1/p_m)$

$\varphi(a*b)=\varphi(a)*\varphi(b)$ donde gcd(a,b)=1

☑ **Teorema de Euler**

$a^{\varphi(n)}\equiv 1 \pmod n$ donde n es un entero positivo y gcd(a,n)=1

$a^b\equiv a^{b \bmod \varphi(n)} \pmod n$ donde n es un entero positivo y gcd(a,n)=1

☑ **Teorema chino del residuo**

Sean $n_1,...,n_k$ enteros primos relativos de a pares. Entonces para todos $a_1,...,a_k$ enteros, existe un x entero que soluciona el sistema de congruencias simultáneas $x\equiv a_1 \pmod{n_1},...,x\equiv a_k \pmod{n_k}$.

```
long solTeoremaChinoResiduo(long a[], long[] n) { // n tiene coprimos de a pares
  long N=1,x=0; for (long y:n) N*=y;
  for (int i=0; i<a.length; i++) x=(x+a[i]*gcdExtendido(n[i],N/n[i])[2]*N/n[i])%N;
  return (x+N)%N;
}
```

☑ **Regla de Simpson**

$$\int_a^b f(x)dx \approx \frac{(b-a)}{6}*\left[f(a)+4*f\left(\frac{a+b}{2}\right)+f(b)\right]$$

☒ **Método de Newton**

☒ **Método de la bisección**

☑ **Combinaciones, Permutaciones**

binomialCoefficient(n,r)=n!/(r!*(n-r)!)

r_permutations(n,r)=n!/(n-r)!

☑ **Números de Catalán (Catalan Numbers)**

catalan(n)=binomialCoefficient(n*2,n)/(n+1)=(n*2)!/((n+1)!*n!)

Usos:

* Cantidad de árboles de búsqueda binaria que pueden construirse con un conjunto de n números de tal manera que a cada elemento del conjunto se le asocie exactamente un nodo del árbol.
* Número de cadenas de longitud n*2 construidas con n ceros y n unos tales que ningún prefijo tenga más ceros que unos.
* Número de expresiones que contienen n pares de paréntesis bien anidados.
* Número de maneras de asociar n+1 factores en la aplicación de un operador binario asociativo.
* Número de árboles binarios de n+1 hojas donde cada nodo tiene cero o dos hijos.
* Número de caminos monótonos en una malla n×n entre esquinas opuestas, sin cruzar la diagonal.
* Número de maneras de dividir un polígono convexo de n+2 lados en triángulos con líneas de vértice a vértice que no se crucen.

☑ **Factorial Cuádruple (Quadruple Factorial)**

Corresponde a variantes etiquetadas de los problemas asociados a los números de Catalán.

quadrupleFact(n)=catalan(n)*(n+1)!=(n*2)!/n!

Usos:

* Número de árboles binarios que pueden construirse usando n elementos: quadrupleFact(n)/(n+1)

BIGINTEGER

☑ **Raíz cuadrada BigInteger {O($\log_2(n)$)}**

```
BigInteger integerSqrt(BigInteger n) { // n^(1/2)
  for (BigInteger x=n,y; ; x=y)
  {y=x.add(n.divide(x)).shiftRight(1); if (y.compareTo(x)>=0) return x;}
}
```

☑ **Raíz p-ésima BigInteger {O(log₂(n))}**

```
BigInteger integerRoot(BigInteger n, int p) { // n^(1/p)
  BigInteger p0=BigInteger.valueOf(p),p1=BigInteger.valueOf(p-1);
  for (BigInteger x=n,y; ; x=y) {
    y=x.multiply(p1).add(n.divide(x.pow(p-1))).divide(p0);
    if (y.compareTo(x)>=0) return x;
  }
}
```

☑ **Multiplicación de números grandes - Karatsuba [A. A. Karatsuba (1960)] {O(n^log2(3))}**

```
BigInteger karatsuba(BigInteger x, BigInteger y) { // Código de Robert Sedgewick y Kevin Wayne
  int n=Math.max(x.bitLength(),y.bitLength()); if (n<=2000) return x.multiply(y);
  n=(n/2)+(n%2);
  BigInteger b=x.shiftRight(n),a=x.subtract(b.shiftLeft(n));
  BigInteger d=y.shiftRight(n),c=y.subtract(d.shiftLeft(n));
  BigInteger ac=karatsuba(a,c),bd=karatsuba(b,d),abcd=karatsuba(a.add(b),c.add(d));
  return ac.add(abcd.subtract(ac).subtract(bd).shiftLeft(n)).add(bd.shiftLeft(n*2));
}
```

☑ **Fibonacci {O(log₂(n))}**

```
BigInteger fib(int n) {
  BigInteger i=BigInteger.ONE,h=i,j=BigInteger.ZERO,k=j,t;
  for (; n>0; n/=2) {
    if (n%2==1)
    {j=i.multiply(h).add(j.multiply(k)).add(t=j.multiply(h)); i=i.multiply(k).add(t);}
    t=h.pow(2); h=k.multiply(h).multiply(BigInteger.valueOf(2)).add(t); k=k.pow(2).add(t);
  }
  return j;
}
```

☑ **Suma, multiplicación, división y potenciación de números grandes**

Véase la clase java.math.BigInteger.

STRINGOLOGY

☑ **Bordes de cadenas - Knuth-Morris-Pratt [D. Knuth, V. Pratt, J. H. Morris (1977)] {O(|W|)}**

```
int[] getBorderArray(char[] W) {
  int[] T=new int[W.length+1]; T[0]=-1; T[1]=0;
  for (int i=2,j=0; i<=W.length; )
  {if (W[i-1]==W[j]) T[i++]=++j; else if (j>0) j=T[j]; else T[i++]=0;}
  return T;
}
```

☑ **String Search - Knuth-Morris-Pratt [D. Knuth, V. Pratt, J. H. Morris (1977)] {O(|W|)}**

```
int indexOf(char[] S, char[] W, int[] T) { // Índice donde ocurre S en W. T=getBorderArray(W).
  if (S.length==0) return 0;
  for (int m=0,i=0; m+i<W.length; )
  {if (S[i]==W[m+i]) {if (++i==S.length) return m;} else {m+=i-T[i]; if (i>0) i=T[i];}}
  return -1;
}
```

☑ **Período de una cadena {O(|W|)}**

```
int periodo(char[] W) {int t=W.length,k=t-getBorderArray(W)[t]; return t%k==0?t/k:1;}
```

☑ **Búsqueda de patrones en una cadena - Aho-Corasick [A. Aho, M. Corasick (1975)] {O(n+m+z)}**

```
// Busca todas las ocurrencias de los patrones en el objetivo
// La respuesta es una lista de parejas de la forma <pos,ind> donde pos es una posición
// donde ocurre el ind-ésimo patrón.
// k=|patterns|, n=|patterns[0]|+|patterns[1]|+...+|patterns[k-1]|, m=|target|, z=|answer|
List<int[]> search(char[][] patterns, char[] target) {
  List<int[]> res=new ArrayList<int[]>(); Trie root=new Trie(),q=root,p;
  for (int k=0; k<patterns.length; k++) add(root,patterns[k],k);
  prepare(root);
  for (int j=0; j<target.length; j++) {
    while ((p=q.gotoF(root,target[j]))==null) q=q.failure;
    q=p; if (q.out!=null) for (int id:q.out) res.add(new int[]{j-patterns[id].length+1,id});
```

```
  }
  return res;
}
void prepare(Trie root) {
  Queue<Trie> queue=new LinkedList<Trie>(); Trie r,v,p;
  if (root.nodes!=null) for (Trie t:root.nodes) {t.failure=root; queue.add(t);}
  while (!queue.isEmpty()) if ((r=queue.poll()).nodes!=null) for (Trie u:r.nodes) {
    queue.add(u);
    for (v=r.failure; (p=v.gotoF(root,u.ch))==null; ) v=v.failure;
    u.failure=p; if (p.out!=null) u.addOutputs(p.out);
  }
}
void add(Trie root, char[] a, int idPattern) {
  Trie u=root,t;
  for (char c:a)
    if ((t=u.get(c))==null) {
      u.nodes=u.nodes==null?new Trie[1]:Arrays.copyOf(u.nodes,u.nodes.length+1);
      (u=u.nodes[u.nodes.length-1]=new Trie()).ch=c;
    }
    else u=t;
  u.addOutputs(idPattern);
}
static class Trie { // Keyword tree
  char ch=0; Trie[] nodes=null; int[] out=null; Trie failure=null; Trie() {}
  void addOutputs(int...arr) {
    if (out==null) {out=arr; return;}
    int g=out.length,f=arr.length; System.arraycopy(arr,0,out=Arrays.copyOf(out,g+f),g,f);
  }
  Trie get(char c) {if (nodes!=null) for (Trie t:nodes) if (t.ch==c) return t; return null;}
  Trie gotoF(Trie root, char c) {Trie t=get(c); return t!=null?t:(this==root?root:null);}
}
```

PARSING

☑ **Parser de expresiones aritméticas que incluyen operadores +,-,*,/, paréntesis y constantes flotantes, que concuerda con el parser de Java para evaluar expresiones de este tipo {O(|s|)}**

```
double parse(String s) {return new SimpleParser().parse(new StringTokenizer(s,"+-/*()",true));}
static class SimpleParser { // El menos unario - se detecta y se convierte en ~
  Stack<String> pO=new Stack<String>(); Stack<Double> pV=new Stack<Double>(); String opA=null;
  double parse(StringTokenizer st) {
    for (shift(st); opA!=null; )
      if (pO.isEmpty()||(f(pO.peek(),"+-")&&f(opA,"/*"))||f(opA,"~")) shift(st); else reduce();
    while (!pO.isEmpty()) reduce();
    return pV.pop();
  }
  void shift(StringTokenizer st) {
    if (opA!=null) pO.push(opA);
    int i=0; for (opA=null; st.hasMoreTokens(); i++) {
      String s=st.nextToken();
      if (f(s,"+-/*")) opA=i>0?s:"~";
      else if (f(s,"(")) pV.push(new SimpleParser().parse(st));
      else if (!f(s,")")) pV.push(Double.parseDouble(s));
      if (f(s,"+-/*)")) break;
    }
  }
  void reduce() {
    char c=pO.pop().charAt(0);
    if (c=='~') {pV.push(-pV.pop()); return;}
    double b=pV.pop(),a=pV.pop();
    pV.push(c=='+'?a+b:(c=='-'?a-b:(c=='*'?a*b:a/b)));
  }
  boolean f(String s, String f) {return f.indexOf(s)>=0;}
}
```

☒ **Gramáticas libres de contexto - Algoritmo CYK [Cocke, Younger, Kasami] {O(n^3)}**

TEORIA DE JUEGOS

☒ **Juegos de suma cero**
☑ **Nim**

http://en.wikipedia.org/wiki/Nim
From Wikipedia, the free encyclopedia
Nim is a two-player mathematical game of strategy in which players take turns removing objects from distinct heaps. On each turn, a player must remove at least one object, and may remove any number of objects provided they all come from the same heap.
Variants of Nim have been played since ancient times. The game is said to have originated in China (it closely resembles the Chinese game of "Jianshizi", or "picking stones"), but the origin is uncertain; the earliest European references to Nim are from the beginning of the 16th century. Its current name was coined by Charles L. Bouton of Harvard University, who also developed the complete theory of the game in 1901, but the origins of the name were never fully explained. The name is probably derived from German nimm! meaning "take!", or the obsolete English verb nim of the same meaning. It should also be noted that turning the word NIM upside-down and backwards results in WIN (see Ambigram).
Nim is usually played as a misère game, in which the player to take the last object loses. Nim can also be played as a normal play game, which means that the person who makes the last move (i.e., who takes the last object) wins. This is called normal play because most games follow this convention, even though Nim usually does not.
Normal play Nim (or more precisely the system of nimbers) is fundamental to the Sprague-Grundy theorem, which essentially says that in normal play every impartial game is equivalent to a Nim heap that yields the same outcome when played in parallel with other normal play impartial games (see disjunctive sum).
A version of Nim is played – and has symbolic importance– in the French New Wave film Last Year at Marienbad (1961).
Illustration
A normal play game may start with heaps of 3, 4 and 5 objects:
In order to win always leave an even number of 1's, 2's, and 4's.

Sizes of heaps			Moves
A	B	C	
3	4	5	I take 2 from A
1	4	5	You take 3 from C
1	4	2	I take 1 from B
1	3	2	You take 1 from B
1	2	2	I take entire A heap leaving two 2's.
0	2	2	You take 1 from B
0	1	2	I take 1 from C leaving two 1's. (In misère play I would take 2 from C leaving (0,1,0).)
0	1	1	You take 1 from B
0	0	1	I take entire C heap and win.

Mathematical theory
Nim has been mathematically solved for any number of initial heaps and objects; that is, there is an easily-calculated way to determine which player will win and what winning moves are open to that player. In a game that starts with heaps of 3, 4, and 5, the first player will win with optimal play, whether the misère or normal play convention is followed.
The key to the theory of the game is the binary digital sum of the heap sizes, that is, the sum (in binary) neglecting all carries from one digit to another. This operation is also known as exclusive or (xor) or vector addition over GF(2). Within combinatorial game theory it is usually called the nim-sum, as will be done here. The nim-sum of x and y is written $x \oplus y$ to distinguish it from the ordinary sum, $x + y$. An example of the calculation with heaps of size 3, 4, and 5 is as follows:

Binary	Decimal	
011_2	3_{10}	Heap A
100_2	4_{10}	Heap B
101_2	5_{10}	Heap C

010_2	2_{10}	The nim-sum of heaps A, B, and C, $3 \oplus 4 \oplus 5 = 2$

An equivalent procedure, which is often easier to perform mentally, is to express the heap sizes as sums of distinct powers of 2, cancel pairs of equal powers, and then add what's left:

3 = 0 + 2 + 1 =	2	1	Heap A
4 = 4 + 0 + 0 = 4			Heap B
5 = 4 + 0 + 1 = 4		1	Heap C

---

2 = 2 What's left after cancelling 1s and 4s

In normal play, the winning strategy is to finish every move with a Nim-sum of 0, which is always possible if the Nim-sum is not zero before the move. If the Nim-sum is zero, then the next player will lose if the other player does not make a mistake. To find out which move to make, let X be the Nim-sum of all the heap sizes. Take the Nim-sum of each of the heap sizes with X, and find a heap whose size decreases. The winning strategy is to play in such a heap, reducing that heap to the Nim-sum of its original size with X. In the example above, taking the Nim-sum of the sizes is X = 3 ⊕ 4 ⊕ 5 = 2. The Nim-sums of the heap sizes A=3, B=4, and C=5 with X=2 are

A ⊕ X = 3 ⊕ 2 = 1 B ⊕ X = 4 ⊕ 2 = 6 C ⊕ X = 5 ⊕ 2 = 7

The only heap that is reduced is heap A, so the winning move is to reduce the size of heap A to 1 (by removing two objects).

As a particular simple case, if there are only two heaps left, the strategy is to reduce the number of objects in the bigger heap to make the heaps equal. After that, no matter what move your opponent makes, you can make the same move on the other heap, guaranteeing that you take the last object.

When played as a misère game, Nim strategy is different only when the normal play move would leave no heap of size 2 or larger. In that case, the correct move is to leave an odd number of heaps of size 1 (in normal play, the correct move would be to leave an even number of such heaps).

In a misère game with heaps of sizes 3, 4 and 5, the strategy would be applied like this:

A	B	C	Nim-sum	
3	4	5	$011_2=2_{10}$	I take 2 from A, leaving a sum of 000, so I will win.
1	4	5	$000_2=0_{10}$	You take 2 from C
1	4	3	$110_2=6_{10}$	I take 2 from B
1	2	3	$000_2=0_{10}$	You take 1 from C
1	2	2	$001_2=1_{10}$	I take 1 from A
0	2	2	$000_2=0_{10}$	You take 1 from C
0	2	1	$011_2=3_{10}$	The normal play strategy would be to take 1 from B, leaving an even number (2) heaps of size 1. For misère play, I take the entire B heap, to leave an odd number (1) of heaps of size 1.
0	0	1	$001_2=1_{10}$	You take 1 from C, and lose.

Proof of the winning formula

The soundness of the optimal strategy described above was demonstrated by C. Bouton.

Theorem. In a normal Nim game, the first player has a winning strategy if and only if the nim-sum of the sizes of the heaps is nonzero. Otherwise, the second player has a winning strategy.

Proof: Notice that the nim-sum (⊕) obeys the usual associative and commutative laws of addition (+), and also satisfies an additional property, x ⊕ x = 0 (technically speaking, the nonnegative integers under ⊕ form an Abelian group of exponent 2).

Let x1, ..., xn be the sizes of the heaps before a move, and y1, ..., yn the corresponding sizes after a move. Let s = x1 ⊕ ... ⊕ xn and t = y1 ⊕ ... ⊕ yn. If the move was in heap k, we have xi = yi for all i ≠ k, and xk > yk. By the properties of ⊕ mentioned above, we have

t = 0 ⊕ t = s ⊕ s ⊕ t = s ⊕ (x1 ⊕ ... ⊕ xn) ⊕ (y1 ⊕ ... ⊕ yn)
= s ⊕ (x1 ⊕ y1) ⊕ ... ⊕ (xn ⊕ yn) = s ⊕ 0 ⊕ ... ⊕ 0 ⊕ (xk ⊕ yk) ⊕ 0 ⊕ ... ⊕ 0
= s ⊕ xk ⊕ yk

(*) t = s ⊕ xk ⊕ yk.

The theorem follows by induction on the length of the game from these two lemmata.

Lemma 1. If s = 0, then t ≠ 0 no matter what move is made.

Proof: If there is no possible move, then the lemma is vacuously true (and the first player loses the normal play game by definition). Otherwise, any move in heap k will produce t = xk ⊕ yk from (*). This number is nonzero, since xk ≠ yk.

Lemma 2. If s ≠ 0, it is possible to make a move so that t = 0.

Proof: Let d be the position of the leftmost (most significant) nonzero bit in the binary representation of s, and choose k such that the dth bit of xk is also nonzero. (Such a k must exist, since otherwise the dth bit of s would be 0.) Then letting yk = s ⊕ xk, we claim that yk < xk: all bits to the left of d are the same in xk and yk, bit d decreases from 1 to 0 (decreasing the value by 2^k), and any change in the remaining bits will amount to at most 2^k-1. The first player can thus make a move by taking xk − yk objects from heap k, then

t = s ⊕ xk ⊕ yk (by (*)) = s ⊕ xk ⊕ (s ⊕ xk) = 0.

The modification for misère play is demonstrated by noting that the modification first arises in a position that has only one heap of size 2 or more. The normal play strategy is for the player to reduce this to size 0 or 1, leaving an even number of heaps with size 1, and the misère strategy is to do the opposite. From that point on, all moves are forced.

Other variations of the game

In another game which is commonly known as Nim (but is better called the subtraction game S(1,2,...,k)), an upper bound is imposed on the number of stones that can be removed in a turn. Instead of removing arbitrarily many stones, a player can only remove 1 or 2 or ... or k at a time. This game is commonly played in practice with only one heap (for instance with k = 3 in the game Thai 21 on Survivor: Thailand, where it appeared as an Immunity Challenge).

Bouton's analysis carries over easily to the general multiple-heap version of this game. The only difference is that as a first step, before computing the Nim-sums, we must reduce the sizes of the heaps modulo k + 1. If this makes all the heaps of size zero (in misère play), the winning move is to take k objects from one of the heaps. In particular, in a play from a single heap of n stones, the second player can win iff

$n \equiv 0 \pmod{k+1}$ (in normal play), or $n \equiv 1 \pmod{k+1}$ (in misère play).

VARIOS

☒ Series

$(\Sigma i|1\le i\le n:i)=n(n+1)/2$ $(\Sigma i|1\le i\le n:i^2)=n(n+1)(2n+1)/6$

$(\Sigma i|1\le i\le n:i^3)=n^2(n+1)^2/4$ $(\Sigma i|1\le i\le n:i^4)=n(n+1)(2n+1)(3n^2+3n-1)/30$

$(\Sigma i|1\le i\le n:i^5)=n^2(n+1)^2(2n^2+2n-1)/12$ $(\Sigma i|1\le i\le n:i^6)=n(n+1)(2n+1)(3n^4+6n^3-3n+1)/42$

$(\Sigma i|1\le i\le n:i^7)=n^2(n+1)^2(3n^4+6n^3-n^2-4n+2)/24$ $(\Sigma i|1\le i\le n:i^8)=n(n+1)(2n+1)(5n^6+15n^5+5n^4-15n^3-n^2+9n-3)/90$

$(\Sigma i|0\le i\le n:A^i)=(A^{n+1}-1)/(A-1)$ $(\Sigma i|1\le i\le n:iA^i)=(n(A^{n+2}-A^{n+1})-A^{n+1}+A)/(A-1)^2$

$(\Sigma i|0\le i\le \infty:i*A^i)=A/(1-A)^2$

$(\Sigma i|0\le i\le n:\text{binomial}(n,r)x^iy^{n-i})=(x+y)^n$ $(\Sigma i|0\le i\le n:\text{binomial}(n,r))=2^n$

☑ Decimal a Romano

```
String[] r1={"I","X","C","M"},r2={"V","L","D"};
String intToRomano(int n) {
  if (n<=0 || n>=4000) return null; String s="";
  for (int i=0; n>0; i++, n/=10) {
    int d=(n%10),u=d%5;
    if (u==4) s=r1[i]+(d==4?r2[i]:r1[i+1])+s;
    else {for (int k=0; k<u; k++) s=r1[i]+s; if (d>4) s=r2[i]+s;}
  }
  return s;
}
```

☑ Romano a Decimal

```
Map<String,Integer> mapRomanos=new TreeMap<String,Integer>();
int romanoToInt(String s) {
  if (mapRomanos.isEmpty()) for (int i=1; i<4000; i++) mapRomanos.put(intToRomano(i),i);
  return mapRomanos.containsKey(s)?mapRomanos.get(s):-1;
}
```

☑ Postorden dado preorden e inorden

```
String postOr(String preOr, int i1, int t1, String inOr, int i2, int t2) {
  if (t1==0) return "";
  char r=preOr.charAt(i1); int k=inOr.indexOf(r,i2)-i2;
  return postOr(preOr,i1+1,k,inOr,i2,k)+postOr(preOr,i1+k+1,t1-k-1,inOr,i2+k+1,t2-k-1)+r;
}
```

☑ Disjoint-set data structure {union,find: O(α(n))=O(ackermann^{-1}(n,n))}

```
static class DisjointSet {
  DisjointSet parent=this; int rank=0;
  DisjointSet() {}
  static void union(DisjointSet x, DisjointSet y) {
    DisjointSet xr=find(x),yr=find(y);
    if (xr.rank>yr.rank)      yr.parent=xr;
    else if (xr.rank<yr.rank) xr.parent=yr;
    else if (xr!=yr) {yr.parent=xr; xr.rank++;}
  }
  static DisjointSet find(DisjointSet x) {return x.parent==x?x:(x.parent=find(x.parent));}
}
```

☑ Sat-2 (2-satisfiability)

```
//-------------------------------------------------------------------------------------------
// Determinar si una expresión booleana en forma normal conjuntiva es satisfacible.
//   Sean p[1],...,p[n] variables booleanas.
//   Cada pareja <i,j> en 'clausulas' (1<=|i|<=n,1<=|j|<=n) representa la cláusula:
//       p[|i|]v p[|j|]  si i>0,j>0       ¬p[|i|]v p[|j|]  si i<0,j>0
```

```
//       p[|i|]v¬p[|j|]  si i>0,j<0       ¬p[|i|]v¬p[|j|]  si i<0,j<0
// Si la fórmula es insatisfacible, retorna null. Si la fórmula es satisfacible, retorna un
// estado que la satisface, donde estado[k-1] es el valor de la variable p[k] (k=1..n).
boolean[] sat2(int[][] clausulas, int n) {
  int t=clausulas.length,estado[]=new int[n]; boolean res[]=new boolean[n];
  if (!sat2(clausulas,estado,0)) return null;
  for (int i=0; i<n; i++) res[i]=estado[i]==2; return res;
}
boolean or(int a, int b) {return a==2||b==2;} // 0: sin valor, 1: false, 2: true
boolean sat2(int[][] m, int[] estado, int k) {
  if (k==m.length) return true;
  int i1=m[k][0],i2=m[k][1],a1=Math.abs(i1)-1,a2=Math.abs(i2)-1,e1=estado[a1],e2=estado[a2];
  for (int v1=1; v1<=2; v1++) for (int v2=1; v2<=2; v2++)
    if ((e1==0||(e1==v1))&&(e2==0||(e2==v2))&&or(i1<0?3-v1:v1,i2<0?3-v2:v2)&&(a1!=a2||v1==v2))
    {estado[a1]=v1; estado[a2]=v2; if (sat2(m,estado,k+1)) return true;}
  estado[a1]=e1; estado[a2]=e2; return false;
}
```

☑ Teorema maestro

(Tomado de la referencia [Cor2001])
Sean $a \geq 1$ y $b > 1$ constantes, $f(n)$ una función y $T(n)=a*T(n/b)+f(n)$ una ecuación de recurrencia definida en los enteros no negativos. $T(n)$ puede ser acotado asintóticamente así:
1. Si $f(n)=O(n^{\log_b(a)-\epsilon})$ para alguna constante $\epsilon>0$, entonces $T(n)=\Theta(n^{\log_b(a)})$.
2. Si $f(n)=\Theta(n^{\log_b(a)})$, entonces $T(n)=\Theta(n^{\log_b(a)}*\log(n))$.
3. Si $f(n)=\Omega(n^{\log_b(a)+\epsilon})$ para alguna constante $\epsilon>0$ y $a*f(n/b)\leq c*f(n)$ para alguna constante $c<1$ y n suficientemente grande, entonces $T(n)=\Theta(f(n))$.

☑ Código gray

```
long gray(long n) {return n^(n>>1);}
long inverseGray(long n) {
  for (long ish=1,ans=n,idiv; true; ish<<=1)
  {ans^=(idiv=ans>>ish); if (idiv<=1||ish==32) return ans;}
}
```

☑ Polyominoes

Free: 1,1,1,2,5,12,35,108,369,1285,4655,17073,63600,238591,901971,3426576,13079255,50107909,192622052,742624232,2870671950,11123060678,43191857688,168047007728,654999700403,2557227044764,9999088822075,39153010938487,153511100594603
One-Sided: 1,1,2,7,18,60,196,704,2500,9189,33896,126759,476270,1802312,6849777,26152418,100203194,385221143,1485200848,5741256764,22245940545,86383382827,336093325058,1309998125640
Fixed: 1,2,6,19,63,216,760,2725,9910,36446,135268,505861,1903890,7204874,27394666,104592937,400795844,1540820542,5940738676,22964779660,88983512783,345532572678,1344372335524,5239988770268,20457802016011,79992676367108,313224032098244,1228088671826973
With holes: 0,0,0,0,0,0,1,6,37,195,979,4663,21474,96496,425449,1849252,7946380,33840946,143060339,601165888,2513617990,10466220315,43425174374

BIBLOGRAFÍA, RECURSOS

Textos

[Ber2008] "Computational Geometry: Algorithms and Applications" de Mark de Berg.
[Cor2001] "Introduction to Algorithms" de Thomas Cormen et. al.
[Car1993] "ALGUNOS ALGORITMOS LOGARÍTMICOS" de Rodrigo Cardoso.
[Gro2004] "Handbook of Graph Theory" de Jonathan L. Gross y Jay Yellen.
[Sed2004] "Algorithms in Java. Part 5: Graph Algorithms" de Robert Sedgewick.
[Ski1998] "The Algorithm Design Manual" de Steven S. Skiena.
[Ski2003] "Programming Challenges" de Steven S. Skiena y Miguel A. Revilla.
[Vil1996] "Diseño y Manejo de Estructuras de Datos en C" de Jorge A. Villalobos C.

Enlaces

A Tutorial on Dynamic Programming
http://mat.gsia.cmu.edu/classes/dynamic/dynamic.html
ACM Solver
http://www.acmsolver.org/
ACMBEGINNER : ACM Programming Tutorial Site for Valladolid Online Judge
http://www.acmbeginner.tk/
Algorithm Tutorials
http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=alg_index
Algorithmist.com

http://www.algorithmist.com/

Computing intersections in a set of line segments: the Bentley-Ottmann algorithm
http://www.scs.carleton.ca/~michiel/lecturenotes/ALGGEOM/bentley-ottmann.pdf

Elementary Graph Algorithms
http://www.cse.ohio-state.edu/~lai/780/9.graph.pdf

Fibonacci number(fast method)
http://wiki.cs.cityu.edu.hk/acm/

Ford-Fulkerson's Algorithm
http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/maxflow/Maxflow.shtml.en

Geometry Algorithms
http://geometryalgorithms.com/

Geometry Algorithm Archive
http://www.geometryalgorithms.com/algorithm_archive.htm

Implementatations of Algorithms For Line-Segment Intersection
http://www.dike.de/~skanthak/university/cpsc516.pdf
http://www.dike.de/~skanthak/university/lineseg.tar.gz

Intersection point of two lines (2 dimensions)
http://local.wasp.uwa.edu.au/~pbourke/geometry/lineline2d/

Karatsuba.java
http://www.cs.princeton.edu/introcs/78crypto/Karatsuba.java.html

Least Squares Fitting--Polynomial
http://mathworld.wolfram.com/LeastSquaresFittingPolynomial.html

Mathworld
http://mathworld.wolfram.com/Circle-CircleIntersection.html
http://mathworld.wolfram.com/LeastSquaresFitting.html
http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html
http://mathworld.wolfram.com/Triangle.html

Minimal Enclosing Circle - Modern Solutions
http://www.cs.mcgill.ca/~cs507/projects/1998/jacob/solutions.html

Parker-Traub algorithm for inversion of Vandermonde and related matrices
http://www.math.uconn.edu/~olshevsky/papers/m5.pdf

PNPOLY - Point Inclusion in Polygon Test
http://www.ecse.rpi.edu/Homepages/wrf/Research/Short_Notes/pnpoly.html

QRDecomposition.java
http://www.docjar.com/html/api/jmat/data/matrixDecompositions/QRDecomposition.java.html

Set Matching and Aho-Corasick Algorithm
http://www.cs.uku.fi/~kilpelai/BSA05/lectures/slides04.pdf

The Convex Hull of a 2D Point Set or Polygon
http://geometryalgorithms.com/Archive/algorithm_0109/algorithm_0109.htm

The Traveling Salesperson Problem
http://mat.gsia.cmu.edu/classes/dynamic/node8.html

Topcoder
http://www.topcoder.com
http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=alg_index

Topological sort: implementation
http://www.brpreiss.com/books/opus5/html/page556.html

Voronoi Diagram using Divide-and-Conquer Paradigm
http://www.personal.kent.edu/~rmuhamma/Compgeometry/MyCG/Voronoi/DivConqVor/divConqVor.htm

Voronoi Diagrams and a Day at the Beach
http://www.ams.org/featurecolumn/archive/voronoi.html

Wikipedia (http://en.wikipedia.org/wiki/)

Aho-Corasick_algorithm
Breadth-first_search
Chinese_remainder_theorem
Disjoint-set_data_structure
Edmonds-Karp_algorithm
Eulerian_path
Euler%27s_totient_function
Extended_Euclidean_algorithm
Floyd-Warshall_algorithm
Gray_code
Hopcroft-Karp_algorithm
Knuth-Morris-Pratt_algorithm
Kruskal%27s_algorithm
List_of_algorithms
Longest_common_substring_problem
Longest_increasing_subsequence
http://es.wikipedia.org/wiki/N%C3%BAmeros_de_Catalan
Permutation
Prim%27s_algorithm
Selection_algorithm
Simpson's_rule
Vandermonde_matrix

---------- FIN DEL DOCUMENTO ----------