Ad-hoc mathematical problems

October 24, 2019

1 Ad hoc mathematical problems

1.1 Simulación matemática (fuerza bruta)

- Llave en general: **acotar** el número de simulaciones que realizar.
- Optimizar el procesamiento de cada simulación.

UVA 00616

The short story titled Coconuts, by Ben Ames Williams, appeared in the Saturday Evening Post on October 9, 1926. The story tells about five men and a monkey who were shipwrecked on an island. They spent the first night gathering coconuts. During the night, one man woke up and decided to take his share of the coconuts. He divided them into five piles. One coconut was left over so he gave it to the monkey, then hid his share and went back to sleep. Soon a second man woke up and did the same thing. After dividing the coconuts into five piles, one coconut was left over which he gave to the monkey. He then hid his share and went back to bed. The third, fourth, and fifth man followed exactly the same procedure. The next morning, after they all woke up, they divided the remaining coconuts into five equal shares. This time no coconuts were left over. An obvious question is "how many coconuts did they originally gather?" There are an infinite number of answers, but the lowest of these is 3,121. But that's not our problem here. Suppose we turn the problem around. If we know the number of coconuts that were gathered, what is the maximum number of persons (and one monkey) that could have been shipwrecked if the same procedure could occur?

Input: The input will consist of a sequence of integers, each representing the number of coconuts gathered by a group of persons (and a monkey) that were shipwrecked. The sequence will be followed by a negative number.

Output: For each number of coconuts, determine the largest number of persons who could have participated in the procedure described above. Display the results similar to the manner shown below, in the Sample Output. There may be no solution for some of the input cases; if so, state that observation.

Sea p el número de personas, n el número de cocos. Para la primera visita, existe k tal que:

$$(n-1) = k_1 p,$$

es decir que p es **divisor de** $n-1$.

Podríamos usar esta propriedad y comprobar todos los divisores de $n-1$, pero da TLE...
Para la visita siguiente, quedan $k_1(p-1)$ cocos y:

$$k_1(p-1) - 1 = k_2 p,$$

con $k_1 > k_2$.
Similarmente, para la visita siguiente

$$k_2(p-1) - 1 = k_3 p.$$

Al restar uno con el otro:

$$(k_1 - k_2)(p-1) = (k_2 - k_3)p.$$

Como $k_1 - k_2 > 0$, y suponiendo $p > 2$ (lo que implica que $p-1$ y p no tienen factores comunes) entonces p divide $k_1 - k_2$.
En particular $k_1 > p$. Como $(n-1) = k_1 p$, entonces $n - 1 > p^2$ y $p < \sqrt{n-1}$.
Entonces nos podemos limitar a buscar p:

- divisor de $n-1$,
- inferior a $n-1$.

Y verificamos luego las propriedades requerida.

In [1]:
```
#include <iostream>
#include <vector>
#include <math.h>
#include <algorithm>

using namespace std;
ios::sync_with_stdio(false);
int n = 1000;
// p must be a divisor of n-1.
int popt=-1;
// For all k determined to be tested
for (int p=std::max((int)sqrt(n-1),2);p>=2;p--)
    if ((n-1)/p>0 && (n-1)%p==0) {
        bool possible = true;
        int s = n;
        // Simulation of the dividing process
        for (int l=1;l<=p;l++) {
            // At each step (person awaking at night),
            // there must remain one coconut when
            // dividing into k
            if (s%p!=1) { possible=false; break;}
            s-= (1+(s-1)/p);
        }
        // At last, there must remain no coconut
        // when dividing them into k
        if (s%p!=0) { continue;}
        if (possible) {
```

```
                popt = p;
                break;
            }
        }
    if (popt<0) {
        cout << n << " coconuts, no solution" << endl;
    } else {
        cout << n << " coconuts, " << popt << " people and 1 monkey" << endl;
    }
```

```
1000 coconuts, no solution
```

Out[1]:

1.2 Encontrar patrón o formula

- Analisis del problema que permita llegar a un patrón o una formula que hace el problema más simple que lo que parece.
- Una implementación ingenua (fuerza bruta) probablemente daría TLE.

UVA 12004

Check the following code which counts the number of swaps of bubble sort.

```
int findSwaps( int n, int a[]) {

int count = 0, i, j, temp, b[100000];
for( i = 0; i < n; i++ ) {
   b[i] = a[i];
}
for( i = 0; i < n; i++ ) {
   for( j = 0; j < n - 1; j++ ) {
       if( b[j] > b[j+1] ) {
          temp = b[j];
          b[j] = b[j+1];
          b[j+1] = temp;
          count++;
       }
   }
}
return count;

}
```

You have to find the average value of 'count' in the given code if we run findSwaps() infinitely many times using constant 'n' and each time some random integers (from 1 to n) are given in array a[]. You can assume that the input integers in array a[] are distinct.

Input: Input starts with an integer T (≤ 1000), denoting the number of test cases. Each test case contains an integer n ($1 \leq n \leq 105$) in a single line.

Output: For each case, print the case number and the desired result. If the result is an integer, print it. Otherwise print it in 'p/q' form, where p and q are relative prime.

- Caso **peor**: datos en orden decreciente, y en eso caso el número de swaps es $K = 1 + 2 + ... + n - 1 = \frac{1}{2}n(n-1)$.
- Caso **mejor**: datos en orden creciente, 0 swap.
- Hay $K + 1$ valores posibles de ese número.
- Ahora todos los casos de número de swaps no son equiprobables: por ejemplo para $n = 3$, tenemos una distribución $1, 2, 2, 1$.
- El valor buscado es:

$$\sum_{i=0}^{K} in_i.$$

- Observar (ese es el patrón) que en esa suma **hay simetría**: $K - i$ y i corresponden a configuraciones simétricas llevando $n_i K$ en la suma; para K par, el número de swaps $K/2$ corresponde a $n_{K/2}$

En promedio, entonces: * para K impar,

$$\frac{1}{N}(n_0 + n_1 + ... + n_{K/2-1})K = \frac{K}{2}$$

- para K par,

$$\frac{1}{N}((n_0 + n_1 + ... + n_{K/2-1})K + n_{K/2}\frac{K}{2}) = \frac{K}{2}$$

```
In [2]: #include <iostream>
        using namespace std;
        unsigned long N=1200;
        unsigned long N2 = N*(N-1)/2;
        cout << "Case " << 1 << ": ";
        if (N2%2==0) {
            cout << N2/2<< endl;
        } else {
            cout << N2 << "/2" << endl;
        }
```

```
Case 1: 359700
```

```
Out[2]:
```

UVA 12149

Richard P. Feynman was a musician, artist, scientist, teacher and Nobel lauriet. He contributed to the development of the atomic bomb, expanded the understanding of quantumelectrodynamics, translated Mayan hieroglyphics, and cut to the heart of the Challenger disaster. But beyond all

of that, Richard Feynman was a unique and multi-faceted individual and he was famous for his unbelievable stories, unusual life style and his popular books and lectures on mathematics and physics.

Once, in Brazil, Feynman got into a kind of a competition with a native to see who could do faster simple aritmethics, Feynman or an abacus (aka an manual calculator machine)! Feynman lost in operations such as addition and multiplication but he won in cubic roots. Given the number 1729.03 he got the result of 12.002 at the end of a few seconds while his oponent got 12.0! The analog procedure to the Square Root is:

$\sqrt{}$n = a + dx $\Rightarrow$ n = (a + dx)^2

$\Rightarrow$ `n = a^2 + 2 * a * dx + (dx)^2`

$\Rightarrow$ `n = a^2 + 2 * a * dx`

$\Rightarrow$ `dx = 1/2 * (n - a^2)/a`

Considering an square of side a, with area a $*$ a, if you do a small increment of dx on each side , you will get a square with area of the square with side a (Light gray) plus the area of the two small rips (Medium Gray) on top plus the area of the small square(dark gray). Since this is only an approximated method, we can ignore this small area ((dx)^2). Then just get value of dx, and substitute in (1).

Example: To calculate square root of 17, as Feynman has an excelent memory, he knows 'all' perfect squares (as well cubes), he knows that 4 $*$ 4 = 16 then he just use the method above and calculate 4 + 1/8 that equals 4.125 (not very bad as square root of 17 = 4.123...) As Feynman is very lazy, and he doesn't like subtractions at all, he doesn't use negative dx . . . (it's boring..) Your Task is to generalize this procedure to the cubic root, and HELP FEYNMAN! (Just do it, What do you care what other people think?).

Input: The input contains a positive floating-point number per line in the inteval [1 . . . 1000000] (inclusive). The last line of the input file contains a number '0' (zero). This zero should not be processed.

Output: For each line of input print the value of the cubic root approximated by the method explained above. Print the value rounded upto four digits after the decimal point.

- Simple, pero tener cuidado con **redondeos**! Por ejemplo:

  pow(x,1.0/3.0)

  puede fallar para encontrar el 'a'. Usar por ejemplo:

  cbrt(x)

  (ver el caso de prueba 64.0)

1.3 Problemas de rejilla

- Problemas de tipo **rejilla**, en el cual en general lo más dificil es encontrar una manera (rápida) de **indexar** las celdas.

UVA 10233

Dermuba Triangle is the universe-famous flat and triangular region in the L-PAX planet in the Geometria galaxy. Actually nobody knows whether the region is triangular or how it came into existence or how big it is. But the people of Dermuba firmly believe that the region extends to infinity. They live in equilateral triangular fields with sides exactly equal to 1km. Houses are always built at the circumcentres of the triangular fields. Their houses are numbered as shown in the figure on the right. When Dermubian people wishes to visit each other, they follow the shortest path from their house to the destination house. This shortest path is obviously the straight line distance that connects these two houses. Moreover, they also visit all the houses that lie in the straight line they travel. Now, comes your task. You have to write a program which computes the length of the shortest path between two houses given their house numbers.

Input: Input consists of several lines with two non-negative integer values n and m which specify the start and destination house numbers. $0 \leq$ n, m $\leq$ 2147483647. Actually, there are houses beyond this region, but some seventh-sense people in Dermuba say that these houses are left for the dead people.

Output: For each line in the input, print the shortest distance between the given houses in kilometers rounded off to three decimal places.

- El punto es calcular rapidamente un mapeo de indices a coordenadas dentro del triángulo.
- Calcular las sumas $\sum_{i=0}^{L}(2i+1)$ da TLE.
- Usar más bien
$$\sum_{i=0}^{L}(2i+1) = (L+1)^2$$
para determinar el número de linea.
- Determinar luego el número de la celda en la linea.
- Expresar las coordenadas del centro del triangulo equilateral correspondiente con un poco de geometría.

1.4 Secuencias de números

Secuencia matemática dada, con propriedades que explotar.

- Generar algún término de la secuencia.
- En general, se puede aprovechar de cálculos ya hechos.

UVA 10408

A fraction h/k is called a proper fraction if it lies between 0 and 1 and if h and k have no common factors. For any natural number $n \geq 1$, the Farey sequence of order n, F_n, is the sequence

of all proper fractions with denominators which do not exceed n together with the "fraction" 1/1, arranged in increasing order.

So, for example, F_5 is the sequence:

$$\frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{1}{1}.$$

It is not clear exactly who first thought of looking at such sequences. The first to have proved genuine mathematical results about them seems to be Haros, in 1802. Farey stated one of Haros' results without a proof in an article written in 1816, and when Cauchy subsequently saw the article he discoverd a proof of the result and ascribed the concept to Farey, thereby giving rise to the name Farey sequence. Hardy in his A mathematician's apology writes: ...Farey is immortal because he failed to understand a theorem which Haros had proved perfectly fourteen years before...

Surprisingly, certain simple looking claim about Farey sequences is equivalent to the Riemann hypothesis, the single most important unsolved problem in all of mathematics. Ford circles, see picture, provide a method of visualizing the Farey sequence.

But your task is much simpler than this. For a given n, you are to find the k-th fraction in the sequence F_n.

Input: Input consists of a sequence of lines containing two natural numbers n and k, $1 \leq n \leq 1000$ and k sufficiently small such that there is the k-th term in F_n . (The length of F_n is approximately $0.3039635n^2$).

Output: For each line of input print one line giving the k-th element of F_n in the format as below.

Observar que si las fracciones buscadas son todas las irreductibles para las cuales el denominador no excede n, entonces, se puede buscarlas entre todas las $\frac{j}{i}$, con $j \leq i$ e $i \leq n$

- o la fracción es irreductible (i y j son primos relativos: eso se puede calcular antes, independientemente de n)
- o es reductible pero entonces su valor será tomado por una fracción irreductible de denominador más chico.

```
In [ ]: #include <iostream>
        #include <map>
        #include <vector>
        #include <algorithm>

        using namespace std;
        std::pair<float,pair<unsigned int,unsigned int> > v[500000];
        bool rprimes[1001][1001];
        inline int gcd(int a,int b) {
            while (b!=0) {
                int t = b;
                b = a%b;
                a = t;
            }
            return a;
```

```
}

int main() {
    ios::sync_with_stdio(false);
    for (unsigned int i = 1; i<=1000; i++) {
        for (unsigned int j = 1; j<=i; j++) {
            rprimes[i][j]=(gcd(i,j)==1);
        }
    }
    unsigned int n,k;
    do {
        cin >> n >> k;
        if (cin.eof()) break;
        int l=0;
        for (unsigned int i = 1; i<=n; i++) {
            for (unsigned int j = 1; j<=i; j++) if (rprimes[i][j]) {
                v[l++]=make_pair((float)j/(float)i,make_pair(j,i));
            }
        }
        std::nth_element(v,v+k-1,v+l);
        cout << v[k-1].second.first << "/" << v[k-1].second.second << endl;
    } while (true);
}
```

1.5 Logarithm, Exponentiation, Power

UVA 11847

A creditor wants a daily payment during n days from a poor miner in debt. Since the miner can not pay his daily obligation, he has negotiated with the creditor an alternative way, convenient for both parties, to pay his debt: the miner will give an equivalent of a $1\mu m(= 0.001mm)$ long piece of a silver bar as a guarantee towards the debt. The silver bar owned by the poor miner is initially n μm units long.

By the end of n days the miner would not have any more silver to give and the creditor would have received an amount of silver equivalent to that of the silver bar initially owned by the miner. By then, the miner expected to have enough money to pay the debt at the next day so that he would have back all his silver.

With this negotiation in mind, the miner has realized that it was not necessary to cut exactly 1μ silver piece from the bar everyday. For instance, at the third day he could give the creditor a 3μ silver piece, taking back the equivalent of a 2μ silver piece which the creditor should already have. Since cutting the bar was rather laborious and time consuming, the miner wanted to minimize the number of cuts he needed to perform on his silver bar in order to make the daily silver deposits during the n days. Could you help him?

Input: Input consists of several cases, each one defined by a line containing a positive integer number n (representing the length in micras of the silver bar and the number of days of the amortization period). You may assume that $0 < n < 20000$. The end of the input is recognized by a line with 0.

Output: For each given case, output one line with a single number: the minimum number of cuts in which to cut a silver bar of length nμ to guarantee the debt during n days.

Cada número entre 1 y n tiene que poder estar representado por una representación binaria sobre los elementos resultando de los cortes.

- Caso fácil: $n = 2^q - 1 = 1 + 2 + ... + 2^{q-1}$ ($q - 1$ cortes, es decir $\log_2(n+1) - 1$). Es óptimo.
- Otros casos:
$$2^q - 1 < n < 2^{q+1} - 1.$$

Con q cortes, tenemos pedazos $1, 2, ..., 2^{q-1}$ y $n - (2^q - 1)$.
Todo el intervalo $[1, 2^{q-1}]$ está cubierto con los primeros q pedazos, y el intervalo $[2^q = n - (2^q - 1) + 2^{q+1} - 1 - n, n = n - (2^q - 1) + (2^q - 1)]$ donde $2^{q+1} - 1 - n < 2^q$ (entonces se puede representar con los primeros q pedazos).

1.6 Polinomios

Manipulación de polinomios. * A través de un contenedor sobre los coefficientes. * Evaluación eficiente con el método de Horner

$$a_0x^n + a_1x^{n-1} + \ + a_{n-1}x + a_n = (((...((0.x + a_0)x + a_1)x + ...)x + a_{n-1})x + a_n)$$

Variación de complejidad en función del grado n?

UVA 10268

Looking throw the "Online Judge's Problem Set Archive" I found a very interesting problem number 498, titled "Polly the Polynomial". Frankly speaking, I did not solve it, but I derived from it this problem. Everything in this problem is a derivative of something from 498. In particular, 498 was "...designed to help you remember...basic algebra skills, make the world a better place, etc., etc.". This problem is designed to help you remember basic derivation algebra skills, increase the speed in which world becomes a better place, etc., etc. In 498 you had to evaluate the values of polynomial:

$$a_0x^n + a_1x^{n-1} + \ + a_{n-1}x + a_n.$$

In this problem you should evaluate its derivative. Remember that derivative is defined as

$$a_0nx^{n-1} + a_1(n-1)x^{n-2} + ... + a_{n-1}.$$

All the input and output data will fit into integer, i.e. its absolute value will be less than 2^{31}.

```
In [ ]: #include <iostream>
        #include <sstream>
        #include <string>
        using namespace std;
```

```
int coeffs[1000000];
int main() {
    ios::sync_with_stdio(false);
    int x;
    string s;
    while (cin >> x) {
        getline(cin,s);
        getline(cin,s);
        stringstream ss(s);
        int n=0;
        while (ss>>coeffs[n++]) {
        };
        n-=2;
        int val = coeffs[0]*n;
        for (int i=1;i<n;i++) {
            val*=x;
            val+=coeffs[i]*(n-i);
        }
        cout << val << endl;
}
```

2 Problemas Matemáticos: BigInteger

Límites de los tipos básicos:

Tipo	Bits	Rango	Magnitud
char	8	$[-128, 127]$	1×10^2
unsigned char	8	$[0, 255]$	2×10^2
short	16	$[-32768, 32767]$	3×10^4
unsigned short	16	$[0, 65535]$	6×10^4
int	32	$[-2147483648, 2147483647]$	2×10^9
unsigned int	32	$[0, 4294967295]$	4×10^9
long long	64	$[-9223372036854775808, 9223372036854775807]$	9×10^{18}
unsigned long long	64	$[0, 18446744073709551615]$	2×10^{19}

A veces, en función de como está formulado el problema, se puede usar algún truco para no llegar a saturación...

- Artimética de modulo.
- Decomposición en factores primos.

Ejemplo: Calcular el último digito no-nulo de 25!
12! = 479001600 (ok en 32 bits)
13! = 6227020800 (no ok en 32 bits)
20!= 2432902008176640000 (ok en 64 bits)
21!= 51090942171709440000 (no ok en 64 bits)

```
In [ ]: #include <iostream>
        using namespace std;
        unsigned long long product = 1;
        int n2 = 0;
        int n5 = 0;
        int n  = 25;
        for (int i=2;i<=n;i++) {
            unsigned long long f = i;
            // Check if i multiple of 2
            // If so, divide i by 2 before multiplying to the factorial
            while (f%2==0) {
                n2++;
                f=f>>1;
            }
            // Check if i multiple of 5
            // If so, divide i by 5 before multiplying to the factorial
            while (f%5==0) {
                n5++;
                f/=5;
            }
            product *= f;
        }
        product = <<(n2-n5);
        cout << product%10 << endl;
```

En los casos de que las operaciones que hacer entre los números son simples (adiciones, mutiplicaciones...): encodificarlos en cadenas de caracteres.

- Operaciones **dígito por dígito**...
- En una implementación casera:

1. Cuidado a los **acarreos**.
2. Cuidado al **orden** de los dígitos en la cadena (recorrido desde el final o invertir la cadena).
3. Cuidado al **tamaño** total de la cadena (si importa o no traer ceros como digitos de peso fuerte).

- Se vuelve difícil si potencias o divisiones. . .

```
In [ ]: bignum operator+(const bignum &b) {
            int siz = std::max(s.size(),b.s.size())+1;
            string sum(siz,'0');
            bignum nsum(sum);
            int acareo = 0;
            for (int k=1;k<=std::max(s.size(),b.s.size());k++) {
                int suma = 0;
                if (s.size()>=k) suma+=(*this)[k]-'0';
                if (b.s.size()>=k) suma+=b[k]-'0';
                suma+=acareo;
                if (suma>=10)
```

```
                acareo = 1;
            else
                acareo = 0;
            nsum[k]=suma%10+'0';
        }
        if (acareo) nsum[siz-1]=acareo+'0';
        return nsum;
    };
```

UVA 763

The standard interpretation of the binary number 1010 is $8+2=10$. An alternate way to view the sequence 1010 is to use Fibonacci numbers as bases instead of powers of two. For this problem, the terms of the Fibonacci sequence are:

$$1;2;3;5;8;13;21;...$$

where each term is the sum of the two preceding terms (note that there is only one 1 in the sequence as defined here). Using this scheme, the sequence 1010 could be interpreted as $1\times 5+0\times 3+1\times 2+0\times 1=7$. This representation is called a Fibinary number.

Note that there is not always a unique Fibinary representation of every number. For example the number 10 could be represented as either $8+2$ (10010) or as $5+3+2$ (1110). To make the Fibinary representations unique, larger Fibonacci terms must always be used whenever possible (i.e. disallow 2 adjacent 1's). Applying this rule to the number 10, means that 10 would be represented as $8+2$ (10010).

Input and Output: Write a program that takes two valid Fibinary numbers and prints the sum in Fibinary form. These numbers will have at most 100 digits. In case that two or more test cases had to be solved, it must be a blank line between two consecutive, both in input and output files.

Al agregar dos dígitos:

$$2f_n = f_n + f_{n-1} + f_{n-2} = f_{n+1} + f_{n-2}$$

- Implementar la adición clásica con acarreo pero guardado el "acarreo hacia atrás".
- Utilizar un ciclo while para agregar el acarreo hacia atrás.
- Post-proceso para evitar tener dos 1 consecutivos

En Java, soporte para este tipo de estructuras: BigInteger.
Soporta la operaciones:

- Adición y resta: add, subtract,
- Multiplicación y división: multiply, divide,
- División entera: remainder, mod, divideAndRemainder
- Potencia: power

UVA 10925

Viktor lives in a far and cold country named Krakovia that is passing by difficult times. Viktor works in a factory and after a work day he often goes with some friends to a bar to drink some beers and to dream with better times. Due to some economical problems the inflation is very high in Krakovia and a beer costs about 5,400,000,000 Krakovian dollars. Because this, is hard to check the value of the bill and to divide its value equally by Viktor and his friends. As you have a good heart you have decided to help them to solve this problem.

Input: There will be several test cases, each test case stats with two numbers $1 \leq N \leq 1000$ that is the number of items in the bill and $1 \leq F \leq 20$ that represents how many friends are in the bar and should pay the bill. Then there will be N lines, each line represents the value of a item. The value of a item is indicated by an integer $1 \leq V \leq 1020$. Input is terminated by a bill where $N = F = 0$.

Output: For each test case you should print the message: 'Bill #N costs S: each friend should pay P', where N represents the number of the bill, starting from 1; S indicates the sum of the items of the bill; and P is how much money each friend should pay, you should calculate this value by dividing the total value of the bill by the number of friends in the bar, if the result is not an integer value just print the integer part of the number, see the sample input/output. After each test case, you should print a blank line.

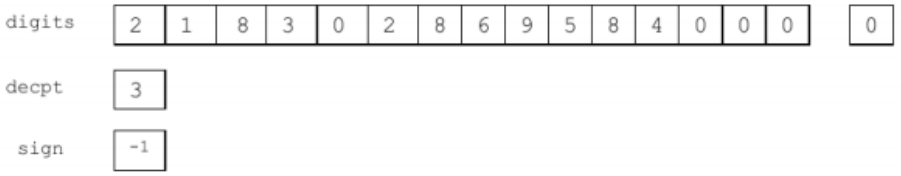
```
In [ ]: import java.util.Scanner;
        import java.math.BigInteger;

        class Solution {
            public static void main(String[] args) {
                Scanner reader = new Scanner(System.in);
                int nbCases = 1;
                while (true) {
                    int nBills = reader.nextInt();
                    int nFriends= reader.nextInt();
                    if (nBills == 0 && nFriends == 0) break;
                    BigInteger total = BigInteger.ZERO;
                    for (int i = 0; i < nBills; i++) {
                        BigInteger individual = reader.nextBigInteger();
                        total = total.add(individual);
                    }
                    BigInteger mean = total.divide(nFriends);
                    System.out.println("Bill #" + (caseNo++) + " costs " + sum + ": each friend
                    System.out.println();
                }
            }
        }
```

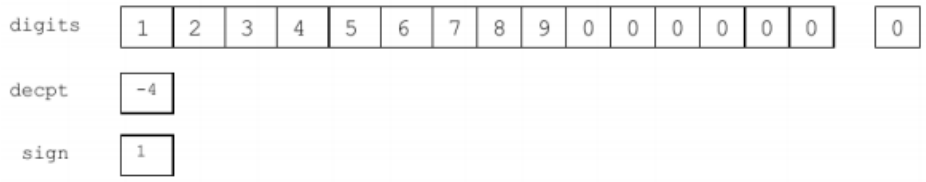
Otros features que valen la pena:

public BigInteger[] divideAndRemainder(BigInteger val)

public BigInteger pow(int exponent)



alt text



alt text

public BigInteger gcd(BigInteger val)

public BigInteger modPow(BigInteger exponent,BigInteger m)

public boolean isProbablePrime(int certainty)

public String toString(int radix)

public String toString()

UVA 11821

A number with 30 decimal digits of precision can be represented by a structure type as shown in the examples below. It includes a 30-element integer array (digits), a single integer (decpt) to represent the position of the decimal point and an integer (or character) to represent the sign (+/-).

For example, the value -218.302869584 might be stored as

The value 0.0000123456789 might be represented as follows.

Your task is to write a program to calculate the sum of high-precision numbers

```
In [ ]: import java.util.Scanner;
        import java.math.BigInteger;
        import java.util.ArrayList;
        class Main {
            public static void main(String[] args) {
                Scanner reader = new Scanner(System.in);
                int n = reader.nextInt();
                for (int i=0;i<n;i++) {
                    ArrayList<BigInteger> numbers = new ArrayList<BigInteger>();
                    ArrayList<Integer> poses = new ArrayList<Integer>();
                    while (true) {
```

```
                String b = reader.next();
                if (b.equals("0"))
                    break;
                int pointpos = b.indexOf('.');
                if (pointpos>=0) {
                    b = b.substring(0, pointpos) + b.substring(pointpos+1)
                    pointpos = b.length()-pointpos;
                } else {
                    pointpos = 0;
                }
                BigInteger big = new BigInteger(b);
                numbers.add(big);
                poses.add(pointpos);
            }
            int posmin = 0;
            for (int j=0;j<numbers.size();j++) {
                if (poses.get(j)>posmin)
                    posmin = poses.get(j);
            }
            BigInteger sum = BigInteger.ZERO;
            // Multiply all the big integers by their corresponding power o
            // and sum them
            for (int j=0;j<numbers.size();j++) {
                BigInteger a = numbers.get(j);
                for (int k=0;k<posmin-poses.get(j);k++) {
                    a = a.multiply(BigInteger.TEN);
                }
                sum=sum.add(a);
            }
            String ss = sum.toString();
            if (ss.length()-posmin>0)
                ss =ss.substring(0,ss.length()-posmin) + "." + ss.substrin
            else {
                while (ss.length()<=posmin) {
                    ss = "0"+ss;
                }
                ss =ss.substring(0,ss.length()-posmin) + "." + ss.substrin
            }
            // Remove trailing zeros
            while (ss.charAt(ss.length()-1)=='0')
                ss =ss.substring(0,ss.length()-1);
            // Remove trailing .
            if (ss.charAt(ss.length()-1)=='.')
                ss =ss.substring(0,ss.length()-1);
            System.out.println(ss);
        }
    }
}
```