Multiplicación del campesino egipcio

Hace mas de 4000 años, los egipcios ya sabían multiplicar números. El sistema de numeración egipcio era decimal, pero no era posicional como el nuestro, sino que era un sistema aditivo, es decir, para representar un 9 ponen 9 veces el 1, y el 60 son 6 veces el 10. Tenían pictogramas para las 7 primeras potencias de 10. Aun así, el algoritmo de multiplicación que utilizaban puede aplicarse a nuestro sistema decimal.

El sistema se conoce por el papiro de Rhind o de Ahmes. Este documento fue descubierto por Alexander Henry Rhind en 1858 en Luxor (Egipto). El papiro contiene más de 80 problemas con cuestiones aritméticas, geométricas, progresiones, proporciones etc. El algoritmo para multiplicar se utiliza en varios de los problemas (puede consultarse en la página `https://culturacientifica.com/2016/09/21/multiplicar-no-dificil-los-egipcios-los-campesinos-rusos/`). Se puede expresar de forma recursiva como.

$$
\begin{array}{lcll}
prod(a,0) & = & 0. & \\
prod(a,1) & = & a. & \\
prod(a,b) & = & prod(2*a, b/2) & si\ b>1\ y\ b\ es\ par \\
prod(a,b) & = & prod(2*a, b/2)+a & si\ b>1\ y\ b\ es\ impar
\end{array}
$$

Se pide realizar una implementación recursiva de este método de multiplicar.

Entrada

La entrada comienza con una línea con el número de casos de prueba. Cada caso consta de dos números enteros: a y b. Se verifica $0 \leq a \leq 10^9$ y $0 \leq b \leq 10^9$.

Salida

Para cada caso de prueba se muestra en una línea el resultado de la multiplicación de los dos valores de entrada.

Entrada de ejemplo

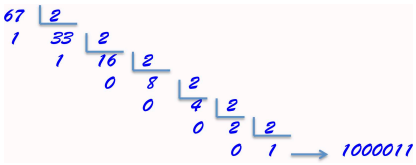
```
5
6 3
10 14
14 10
123 348
0 15
```

Salida de ejemplo

```
18
140
140
42804
0
```

Autor: Isabel Pita.

Transformar un número decimal a binario



Con la revolución digital, el sistema de numeración en base dos o sistema de numeración binario adquiere una gran relevancia al ser el sistema utilizado por los ordenadores y las redes de comunicación. Todo informático necesita en algún momento obtener la representación binaria de un número.

En este problema desarrollaremos un conversor de números expresados en base 10 a números expresados en base 2.

Requisitos de implementación.

El conversor debe realizarse con una función que dado un número entero devuelva en una cadena de caracteres de tipo `std::string` su representación en binario.

Se utilizará una función `resuelveCaso` para leer el dato de entrada, llamar al conversor y escribir la cadena de salida.

Entrada

La entrada comienza con una línea en que se indica el número de casos de prueba. Cada caso consiste en un número entero positivo n ($0 \leq N \leq 2^{31-1}$).

Salida

Para cada caso de prueba se muestra en una línea la representacion en binario del número.

Entrada de ejemplo

```
6
3
8
1
24
156
345
```

Salida de ejemplo

```
11
1000
1
11000
10011100
101011001
```

Autor: Isabel Pita

Invertir los dígitos de un número

Dado un número, definimos su valor invertido como el número obtenido al intercambiar el primer dígito con el último, el segundo con el penúltimo etc. Se pide desarrollar una función que dado un número calcule su valor invertido.

Requisitos de implementación.

El problema debe resolverse utilizando una función recursiva. El prototipo de la función es

```
struct sol {
  int numero;
  int pot;
  }

sol invertir(int n);
```

Donde `pot` es la potencia de 10 necesaria para calcular el número en cada llamada.

No debe utilizarse la función `pow` de la librería STL, ni implementar ninguna función que calcule la potencia de un número.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso se escribe en una línea y consiste en un número entero n tal que $0 \leq n \leq 2^{31} - 1$.

Salida

Para cada caso de prueba se muestra en una línea el valor del número de entrada invertido.

Entrada de ejemplo

```
273645
43270
3
555
0
98543
34
```

Salida de ejemplo

```
546372
7234
3
555
0
34589
43
```

Eliminar los dígitos pares de un número

Para que los niños de su clase aprendan a manejar los números enteros, la profesora ha ideado un juego. Ella dice un número y los alumnos deben transformarlo según la regla que toque ese día. Hoy debemos decir el número que resulta de eliminar los dígitos pares dejando solo los impares. El dígito 0 se considera que es par.

Requisitos de implementación.
La implementación debe realizarse utilizando una función recursiva que reciba el número y devuelva el número calculado como resultado de la función (no como parámetro). La entrada y salida de datos debe realizarse en la función `resuelveCaso`. No se pueden utilizar estructuras auxiliares (vectores...).

Entrada

La entrada consta de una serie de casos de prueba. Cada caso se escribe en una línea y consiste en un número entero positivo mayor que cero n tal que $0 \leq n \leq 2^{63} - 1$.

Salida

Para cada caso de prueba se muestra en una línea el número obtenido.

Entrada de ejemplo

```
273645
4327
3
555
0
98543
34
500
400
```

Salida de ejemplo

```
735
37
3
555
0
953
3
5
0
```

Dígitos complementarios

Diremos que un dígito es complementario de otro si la suma de ambos es 9. Así, el 0 y el 9 son dígitos complementarios, así como el 1 y el 8, el 2 y el 7 etc.

En este problema nos dan un número y debemos calcular el número formado por sus dígitos complementarios, en su mismo orden y en orden inverso.

Requisitos de implementación.

Implementar una función recursiva que reciba un número entero y devuelva el número formado por los dígitos complementarios.

Implementar otra función recursiva que reciba un número entero y devuelva el inverso del número formado por los dígitos complementarios.

Ambas funciones deben tratar cada dígito del número de entrada una sola vez. Se pueden definir más parámetros o valores de salida de la función si se considera necesario. No se puede utilizar el tipo *std::string* para resolver el ejercicio.

No olvides poner tu nombre, el usuario del juez que has utilizado hoy y el comentario con el coste de la función.

Entrada

La entrada comienza con el número de casos de prueba. A continuación cada caso se escribe en una línea y consiste en un número 0 <= N <= 1.000.000.000.

Salida

Para cada caso de prueba se muestra en una línea el número obtenido sustituyendo cada dígito por su complementario, seguido del inverso del número obtenido al sustituir cada número por su inverso.

Entrada de ejemplo

```
7
45637
555
2
90
3050217
0
99
```

Salida de ejemplo

```
54362 26345
444 444
7 7
9 90
6949782 2879496
9 9
0 0
```

La sucesión de Fibonacci.

La sucesión de Fibonacci fue descrita por Leonardo da Vinci en el siglo XIII. Es famosa por sus numerosas aplicaciones, y porque aparece en muchas formaciones de la naturaleza.

La sucesión puede definirse de forma recursiva como sigue:

$$fib(n) = \begin{cases} 0 & n == 0 \\ 1 & n == 1 \\ fib(n-1) + fib(n-2) & n > 1 \end{cases}$$

Requisitos de implementación.

Realizar una implementación con recursión múltiple y comprobar que el tiempo de ejecución es inaceptable para valores de entrada mayores que 50.

Realizar una implementación con recursión simple utilizando parámetros acumuladores. Estudiar el coste de esta solución.

Entrada

La entrada consta de una serie de casos. Cada caso se escribe en una línea y consiste en un número N ($0 <= N <= 89$).

Salida

Para cada valor de entrada mostrar en una línea el elemento de la sucesión de Fibonacci en la posición N. La posición del primer elemento de la sucesión es cero.

Entrada de ejemplo

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

Salida de ejemplo

```
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
```

Números combinatorios

Calcular el número combinatorio $\binom{a}{b}$
Sabiendo que

$$\binom{a}{b} = \begin{cases} 1 & b == 0 \\ 1 & a == b \\ \binom{a-1}{b-1} + \binom{a-1}{b} & b > 0 \wedge a \neq b \end{cases}$$

Requisitos de implementación.

Para mejorar el coste se generalizará la función recursiva con un nuevo parámetro de salida. El nuevo parámetro es una matriz de a filas y b columnas inicializada en la función resuelveCaso con el valor -1. En el elemento i,j de la matriz se guarda el número combinatorio $\binom{i}{j}$. Antes de realizar una llamada recursiva se comprueba si el valor que se quiere calcular ya está guardado en la matriz. En caso afirmativo se utiliza este valor, en caso negativo se realiza la llamada recursiva. Al terminar una llamada recursiva se debe actualizar el valor en la matriz para que pueda ser utilizado por otras llamadas.

Entrada

La entrada consiste en una serie de casos de prueba. Cada caso son dos números enteros positivos tales que: $0 \leq b \leq a$.

Salida

Para cada caso de prueba se escribe en una línea el número combinatorio $\binom{a}{b}$.

Entrada de ejemplo

```
5 2
5 3
5 4
7 4
```

Salida de ejemplo

```
10
10
5
35
```

Autor: Isabel Pita.

Calcula el número de formas de obtener un valor con unos ciertos números realizando solo sumas

Nos dan un vector de números enteros positivos todos ellos diferentes y un valor entero S. Nos piden el número de formas distintas que hay de obtener la cantidad S utilizando únicamente sumas y los valores del vector. Sólo se puede utilizar cada valor una vez.

Requisitos de implementación.
Definir la estrategia recursiva que se va a utilizar.
Utilizar una matriz como parámetro acumulador para evitar repetir llamadas recursivas.

Entrada

La entrada consiste en una serie de casos de prueba. Cada caso se define en dos líneas. En la primera aparece el número de elementos del vector y la cantidad S que se quiere obtener. En la segunda línea aparecen los valores del vector separados por un blanco.

Salida

Para cada caso de prueba se escribe en una línea el número de formas de obtener la cantidad S.

Entrada de ejemplo

```
3 4
1 2 3
4 3
2 1 3 4
5 6
4 2 3 5 1
5 10
4 2 6 3 5
```

Salida de ejemplo

```
1
2
3
2
```

Autor: Isabel Pita.

Torneo de tenis

Hemos organizado un torneo de tenis entre todos los vecinos de la urbanización. El torneo es a un solo partido, es decir, de cada partido sale un ganador que pasa a la siguiente ronda donde se enfrenta con el ganador de otro partido.

Ayer empezamos los partidos, pero como hacía frio muchos jugadores no se presentaron y sus partidos quedaron sin jugar. Por decisión de los organizadores, los jugadores que no se presentan pierden el partido y pasa a la siguiente ronda el jugador que si se presentó. Si no se presenta ninguno de los dos jugadores, no se jugará tampoco el partido de la fase siguiente, clasificándose el otro jugador si lo hubiera. En la primera ronda se juega el primer partido de cada jugador, en la segunda ronda se enfrentan los ganadores de la primera ronda y asi sucesivamente. En cada ronda se enfrentan el ganador de la mitad izquierda con el ganador de la mitad derecha.

Sobre la lista que teníamos con los partidos que debían jugarse, hemos marcado con "NP" aquellos jugadores que no se han presentado. Ahora queremos saber cuantos partidos se han jugado realmente hasta la presente ronda.

Requisitos de implementación.

Implementar una función recursiva que reciba un vector con los datos de los partidos y el número de ronda que nos piden y devuelva el número de partidos que se han jugado realmente, es decir que se presentaron los dos jugadores, hasta la ronda que se pide incluida, un booleano indicando si se presentó alguno de los jugadores y la ronda a la que corresponden esos datos.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba tiene 2 líneas. En la primera se indica el número de elementos del vector y la ronda hasta la que se desea obtener información. En la siguiente aparece el nombre de los jugadores o NP si el jugador no se presentó. En la primera ronda los jugadores de las posiciones pares juegan contra el jugador en la posición impar siguiente, es decir el jugador de la posición cero contra el de la posición 1, el de la posición 2 contra el de la posición 3 etc.

Los nombres de los jugadores son caracteres alfanuméricos y no contienen blancos.El número de elementos del vector es una potencia de dos y es siempre mayor que 1, de forma que los partidos puedan organizarse correctamente. Se garantiza que el número de ronda es mayor o igual que 1 y menor o igual que el número total de rondas posibles.

Salida

Para cada caso de prueba se escribe en una línea el número de partidos que se han jugado realmente, es decir aquellos en los que se presentaron los dos jugadores, hasta la ronda que se pide incluida.

Entrada de ejemplo

```
4 2
Jug1 NP Jug2 NP
4 2
NP NP NP NP
4 2
NP NP Jug1 Jug2
8 3
NP Jug1 NP NP Jug2 Jug3 Jug4 NP
8 2
Jug1 Jug2 NP NP Jug3 NP Jug4 NP
8 1
Jug1 Jug2 Jug3 Jug4 Jug5 Jug6 Jug7 Jug8
8 2
Jug1 Jug2 Jug3 Jug4 Jug5 Jug6 Jug7 Jug8
8 3
Jug1 Jug2 Jug3 Jug4 Jug5 Jug6 Jug7 Jug8
```

Salida de ejemplo

```
1
0
1
3
2
4
6
7
```

Autor: Isabel Pita. (Idea obtenida de un problema del concurso de la UVA).