Problema número 803

Early adopters

Tiempo máximo: 1,000 s Memoria máxima: 4096 KiB

http://www.aceptaelreto.com/problem/statement.php?id=803

Los *early adopters* (en español *consumidores pioneros*) de un producto son aquellos que están muy atentos del mercado y de sus últimas novedades y las adquieren antes de que sean adoptadas por el gran público.

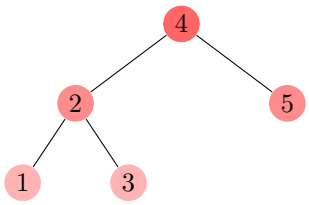
Aunque este tipo de comportamiento se da en muchas áreas, lo habitual es que aparezca sobre todo en el mundo de la tecnología y la compra de *hardware*, con mucha gente esperando el último lanzamiento de teléfono móvil o videoconsola. Ocurre también en la compra de *software*. Los más viejos del lugar recordarán, quizá, la locura que desató la venta de Windows 95 (que hizo que incluso gente sin ordenador se hiciera con un sistema operativo...) y los más jóvenes conocerán seguro a gente esperando con ansia la salida del nuevo juego de su saga favorita.

En el mundo del *desarrollo* software ocurre lo mismo. Muchos programadores están ávidos de probar el último *framework* de programación web, o la "build" nocturna del motor de juegos que utilizan para su desarrollo *indie*. El problema en este caso es que estos programadores suelen sufrir todos los errores o deficiencias de esas primeras versiones poco depuradas que desarrolladores menos "impulsivos" no se encuentran cuando deciden cambiar de versión.

Un ejemplo muy conocido de esto les ocurrió a los primeros programadores de Buggy++, un lenguaje de programación de cierto éxito de finales del siglo pasado. Venía con una potente biblioteca en la que se incluían unos árboles de búsqueda con una implementación defectuosa.

Los árboles de búsqueda son una estructura de datos recursiva que almacena valores en un árbol binario. Cuando se añade un nuevo valor al árbol, se inserta de tal forma que el recorrido en inorden del árbol (hijo izquierdo, raíz, hijo derecho) quede ordenado. Además, durante esa inserción las posiciones de los elementos se ajustan para que la altura del árbol binario se mantenga dentro de unos límites y la búsqueda de los elementos sea logarítimica.

Lamentablemente, en aquellos primeros días de Buggy++ la implementación de la inserción era muy simple y no mantenía la altura dentro de unos límites. En concreto la operación de inserción no equilibraba el árbol: si el elemento que se insertaba era menor que el elemento de la raiz, llamaba recursivamente para insertarlo en el hijo izquierdo; si no se hacía en el hijo derecho. Las llamadas terminaban cuando el hijo al que había que ir era vacío; en ese caso se insertaba ahí. A modo de ejemplo, el árbol siguiente es el que se conseguía llamando a la inserción con el orden [4, 2, 3, 5, 1] o [4, 5, 2, 1, 3], entre otros.



Con esa implementación, la inserción de los números 1...N de forma consecutiva daba lugar a un árbol degenerado en el que la raiz almacenaba el 1 y todos los nodos excepto la única hoja tenían un hijo derecho con el elemento siguiente al suyo.

La pregunta que nos hacemos es en qué orden debía insertar uno de aquellos heróicos *early adopters* de Buggy++ los números entre el 1 y el N si quería limitar la altura del árbol a un cierto valor K.

Entrada

La primera línea de la entrada contiene el número de casos de prueba que vendrán a continuación.

Cada caso de prueba está compuesto de dos números. El primero, $N \leq 5.000$, indica que se deben insertar en el árbol todos los números entre 1 y N. El segundo, $K \leq 30$, indica la altura máxima que debe tener el árbol de búsqueda construido.

Recuerda que la altura de un árbol es cero si el árbol está vacío y el máximo de la altura de los dos hijos más uno si no lo es.

Se garantiza que en cada ejecución de la solución la suma de los N de todos los casos de prueba no superará 100.000.

Salida

Por cada caso de prueba se escribirá el orden en que hay que insertar los números entre el 1 y el N para que el árbol tenga una altura máxima de K. Si no es posible construir semejante árbol se escribirá `IMPOSIBLE`. Si hay más de una forma, se elegirá aquella que tenga antes los números más pequeños.

Los números de la lista deben ir separados por espacios.

Entrada de ejemplo

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

Salida de ejemplo

```
IMPOSIBLE
2 1 3
1 2 3
2 1 4 3 5
```

Autor: Marco Antonio Gómez Martín.

Revisor: Pedro Pablo Gómez Martín.