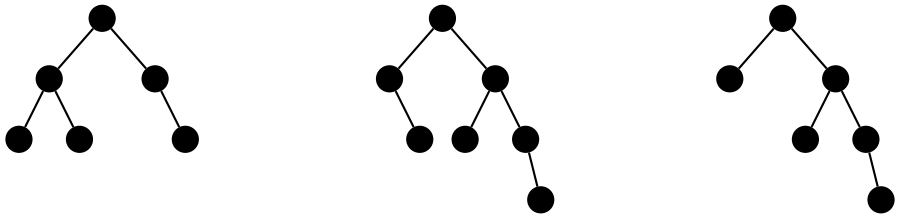
10

¿Está el árbol equilibrado?

Un árbol binario está *equilibrado* si bien es vacío o bien cumple que la diferencia de alturas de sus dos hijos es como mucho 1 y además ambos están equilibrados.

De los siguientes árboles los dos primeros están equilibrados, pero el tercero no lo está.



Dado un árbol binario, queremos averiguar si está equilibrado o no.

Entrada

La entrada comienza con el número de casos que vienen a continuación. Cada caso de prueba consiste en una línea de caracteres con la descripción de un árbol binario: el árbol vacío se representa con un punto (`.`); un árbol no vacío se representa con una `R` (que denota la raíz) seguida de la descripción del hijo izquierdo y del hijo derecho. Los árboles nunca contendrán más de 5.000 nodos.

Salida

Para cada árbol, se escribirá en una línea `SI` si el árbol está equilibrado y `NO` si no lo está.

Entrada de ejemplo

```
3
RRR..R..R.R..
RR.R..RR..R.R..
RR..RR..R.R..
```

Salida de ejemplo

```
SI
SI
NO
```

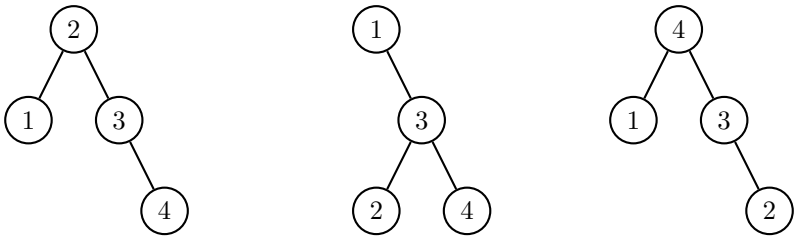
Autor: Alberto Verdejo.

11

¿Es un árbol AVL?

Un árbol AVL (denominado así por las iniciales de los apellidos de sus inventores, Georgii Adelson-Velskii y Yevgeniy Landis) es un árbol binario de búsqueda *equilibrado* en el sentido de que para todo subárbol se cumple que la diferencia entre las alturas de sus dos hijos es como mucho 1. Además, por ser de búsqueda, también se cumple que la clave almacenada en la raíz de cualquier subárbol es estrictamente mayor que todas las claves en su hijo izquierdo y estrictamente menor que todas las claves en su hijo derecho.

De los siguientes árboles (con números enteros como claves, y cuyo valor asociado a cada clave no es relevante para el problema) solamente el de la izquierda es AVL. El del medio no lo es porque no está equilibrado y el de la derecha no lo es porque no se cumple la propiedad del orden entre las claves.



Dado un árbol binario, el problema consiste en decidir si es o no un árbol AVL.

Entrada

La entrada comienza con el número de casos que vienen a continuación. Cada caso de prueba consiste en una línea con la descripción de un árbol binario: primero aparece su raíz (un entero no negativo), y a continuación la descripción del hijo izquierdo y después la del hijo derecho. El número –1 indica el árbol vacío.

Salida

Para cada árbol se escribirá `SI` si el árbol es AVL y `NO` en caso contrario.

Entrada de ejemplo

```
3
2 1 -1 -1 3 -1 4 -1 -1
1 -1 3 2 -1 -1 4 -1 -1
4 1 -1 -1 3 -1 2 -1 -1
```

Salida de ejemplo

```
SI
NO
NO
```

Autor: Alberto Verdejo.

12

Rango de claves en un árbol binario de búsqueda

Dado un árbol binario de búsqueda y dos claves `k1` y `k2`, el problema consiste en producir una lista ordenada con las claves del árbol entre `k1` y `k2`, ambas inclusive.

La implementación debe ser lo más eficiente posible tanto desde el punto de vista del recorrido del árbol (no se exploran partes del árbol en las que sea imposible encontrar claves en el rango) como de generación de la lista resultado.

Entrada

La entrada está formada por diversos casos de prueba. Cada caso ocupa tres líneas. En la primera aparece el número N de claves en el árbol (un número entre 1 y 100.000). A continuación, en una misma línea, aparecen esas claves (números enteros entre 1 y 1.000.000), separadas por espacios, y en el orden en el que tienen que ser insertadas en el árbol. En la siguiente línea aparecen las dos claves, `k1` $\leq$ `k2` (números enteros entre 1 y 1.000.000) que delimitan el rango de claves a buscar.

La entrada termina cuando un árbol no tiene claves (N es 0).

Salida

Para cada caso de prueba se escribirán las claves del árbol entre `k1` y `k2`, ambas inclusive, ordenadas, en una misma línea y separadas por espacios. Si la lista de claves en el rango es vacía, se dejará la línea en blanco.

Entrada de ejemplo

```
6
15 20 25 30 35 40
25 35
6
15 20 25 30 35 40
26 29
6
25 15 30 20 40 35
15 29
0
```

Salida de ejemplo

```
25 30 35

15 20 25
```

Autor: Alberto Verdejo.

13

Encontrar el k-ésimo elemento en un árbol AVL

Extender la implementación de árboles AVL de manera que se mantenga en cada nodo un nuevo atributo `tam_i` que almacene el número de nodos en el hijo izquierdo más uno. Hacer los cambios necesarios para que las funciones que modifican los árboles mantengan coherente el valor de dicho atributo.

Implementar un método para localizar el k-ésimo elemento más pequeño en el árbol en tiempo logarítmico (respecto al tamaño del árbol).

Entrada

La entrada está formada por diversos casos de prueba. Cada caso ocupa cuatro líneas. En la primera aparece el número N de claves (entre 1 y 50.000), no necesariamente distintas, a insertar en el árbol. A continuación, en una misma línea, aparecen esas claves (números enteros entre 1 y 1.000.000), separadas por espacios, en el orden en que deben ser insertadas. (Para este problema el valor asociado a cada clave es indiferente.) En la siguiente línea aparece el número M de elementos que se van a consultar (entre 1 y N). Y en la última línea aparecen, separadas por espacios, las M posiciones (ordinales) de los elementos a consultar (números entre 1 y N).

La entrada termina cuando un árbol no tiene elementos (N es 0).

Salida

Para cada caso de prueba se escribirán, en líneas separadas, los elementos consultados, si existen. Si no hay elemento en la posición consultada, se escribirá `??` en su lugar. Tras procesar cada caso se escribirá una línea más con `---`.

Entrada de ejemplo

```
4
15 20 25 30
2
1 3
5
16 8 4 4 32
3
2 4 5
0
```

Salida de ejemplo

```
15
25
---
8
32
??
---
```

Autor: Alberto Verdejo.