Camino más largo en un árbol binario — V22704_es

Implementad una función *recursiva* que, dado un árbol binario de enteros, devuelva la lista de valores que se encuentran desde la raíz siguiendo el camino más largo del árbol. En caso de que haya varios caminos máximos, se deberá escoger el camino que va lo más a la izquierda posible. Esta es la cabecera:

```
/**
 * @brief Devuelve los valores del camino más largo de un árbol binario.
 *
 * Un camino va desde la raíz del árbol hasta una hoja (un nodo sin hijos).
 * Si hay más de un camino máximo, hay que devolver el que va por las ramas de
 * a la izquierda posible.
 *
 * @param t El árbol binario.
 * @returns Los valores de los nodos del camino más largo de `t`.
 */
vector<int> longest_leftmost_path(BinTree<int> t);
```

Un ejemplo de árbol y su salida correspondiente sería:

```
longest_leftmost_path( 3            ) => [3,3,2,1]
                       |-- 1
                       |   |-- #
                       |   '-- 5
                       '-- 3
                           |-- 2
                           |   |-- 1
                           |   '-- 7
                           '-- #
```

Los ficheros públicos (icono del gatito) contienen:

`bintree.hh`	la clase `BinTree`
`bintree-io.hh`	la entrada/salida de `BinTree`
`bintree-inline.hh`	la e/s en formato ”inline”
`vector-io.hh`	la función `print_vector`
`main.cc`	el programa principal

También hay un `Makefile` y el directorio `.vscode` que tiene la configuración para compilar y depurar con VSCode.

Hay que implementar `longest_leftmost_path` en un **fichero `.cc` nuevo**, compilar (está preparado para poder compilar y depurar con VSCode), y finalmente **enviar solo el fichero con la función**.

Entrada

La primera línea de la entrada describe el formato en el que se describen los árboles, ”inline” o ”visual”. Después vienen un número arbitrario de casos. Cada caso consiste en la descripción de un árbol binario de enteros.

Salida

Para cada caso, la salida contiene el correspondiente camino más largo y más a la izquierda del árbol, en un formato del que ya se encarga el programa principal.

Observación

Vuestra función y subfunciones que creéis deben trabajar solo con árboles. Debéis encontrar una solución *recursiva* del problema.

Os aconsejamos hacer este problema en dos fases: 1) hacer una solución correcta a pesar de que no sea eficiente; y 2) buscar una solución más eficiente utilizando inmersión.

Ejemplo de entrada

```
visual
7
|-- 1
'-- 2
    |-- 3
    |   |-- 5
    |   '-- 4
    '-- 5

6
|-- 7
|   |-- 8
|   '-- 7
'-- 8
    |-- 4
    '-- 6

2
|-- 2
|   |-- 7
|   |   |-- #
|   |   '-- 2
|   |       |-- #
|   |       '-- 7
|   '-- 8
'-- 4
    |-- #
    '-- 7
        |-- 3
        '-- 5

3
|-- 3
|   |-- 4
|   '-- 5
'-- 7
    |-- 1
    '-- 5

7
|-- 4
'-- 3

6
|-- 5
|   |-- 2
|   '-- 7
'-- #

2

4
|-- #
'-- 6
    |-- 3
    '-- 1

4
|-- 8
|   |-- 4
|   |   |-- #
|   |   '-- 7
|   '-- 8
|       |-- 5
|       '-- 1
'-- #

4
```

Ejemplo de salida

```
[7,2,3,5]
[6,7,8]
[2,2,7,2,7]
[3,3,4]
```

```
[7,4]
[6,5,2]
[2]
[4,6,3]
[4,8,4,7]
[4]
```

Información del problema

Autoría: Pau Fernández
Traducción: Pau Fernández

Generación: 2026-04-02T17:11:19.109Z