Generar un árbol a partir de sus recorridos en inorden y postorden X27658_es

Preliminares

Recordad que el recorrido en **inorden** de un árbol es la lista de los nodos del árbol ordenada como sigue: en primer lugar, el recorrido en inorden del hijo izquierdo del árbol, después la raíz del árbol, y después el recorrido en inorden del hijo derecho del árbol. En otras palabras:

$$\begin{aligned} inorden(x(t_1, t_2)) &= inorden(t_1) \ \cdot \ x \ \cdot \ inorden(t_2) \\ inorden(()) &= () \end{aligned}$$

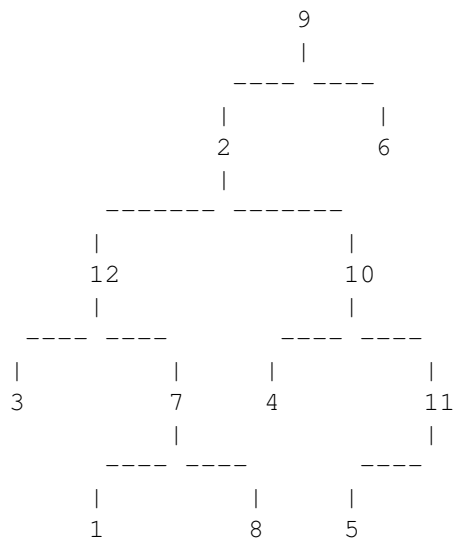
es decir, el *inorden* del árbol vacío es el árbol vacío.

Recordad también que el recorrido en **postorden** de un árbol es la lista de los nodos del árbol ordenada como sigue: en primer lugar, el recorrido en postorden del hijo izquierdo del árbol, después el recorrido en postorden del hijo derecho del árbol, y finalmente la raíz del árbol. En otras palabras:

$$\begin{aligned} postorden(x(t_1, t_2)) &= postorden(t_1) \cdot postorden(t_2) \cdot x \\ postorden(()) &= () \end{aligned}$$

es decir, el postorden del árbol vacío es el árbol vacío.

Por ejemplo, considerad el siguiente árbol, que tiene 12 nodos con valores $1, 2, \ldots, 12$:



Su recorrido en inorden es: 3, 12, 1, 7, 8, 2, 4, 10, 5, 11, 9, 6.
Su recorrido en postorden es: 3, 1, 8, 7, 12, 4, 5, 11, 10, 2, 6, 9.

Ahora bien, sería posible, si sólo conociéramos los recorridos en inorden y postorden, reconstruir el árbol original?

Ejercicio:

Hay que implementar un programa que lea varios casos de entrada. Cada caso empieza con el tamaño n de un árbol binario de enteros desconocido, y viene seguido de los recorridos en

inorden y postorden de este árbol desconocido, donde cada uno de estos recorridos consiste en una lista con todos los valores posibles entre 1 y n. Con esta información, tendréis que construir el árbol y escribirlo por la salida.

Fijaos que el enunciado de este ejercicio ya ofrece el fichero `BinTree.hh`, que tendréis que utilizar para compilar. Sólo hace falta que creéis `main.cc`, poniendo los includes que hagan falta e implementando el programa y las funciones que hagan falta para solucionar el ejercicio. Sólo hace falta que subáis `main.cc` al Jutge.

Entrada

La primera línea de la entrada describe el formato en el que se describen los árboles de salida, o bien INLINEFORMAT o bien VISUALFORMAT. Después viene una línea en blanco. Después vienen un número arbitrario de casos. Cada caso consiste en una primera línea con un número n, una segunda línea con los nodos 1 y n en inorden y separados por espacios en blanco, y una tercera línea con los nodos 1 y n en postorden y separados por espacios en blanco. Después de cada caso hay una línea en blanco.

Salida

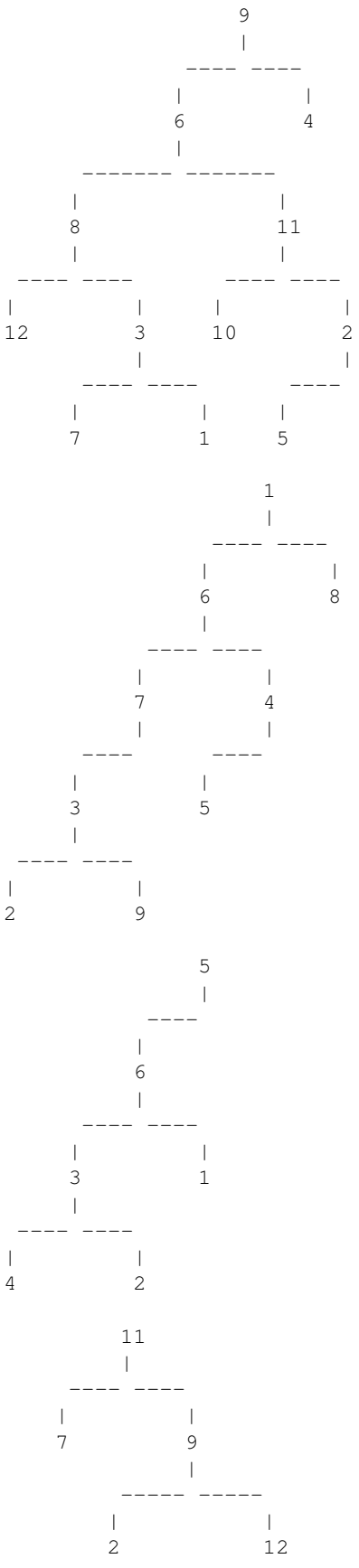
Para cada caso, hay que escribir el árbol binario con n nodos que contienen todos los valores entre 1 y n, y tal que su recorrido en inorden es la primera lista de valores, y su recorrido en postorden es la segunda lista de valores.

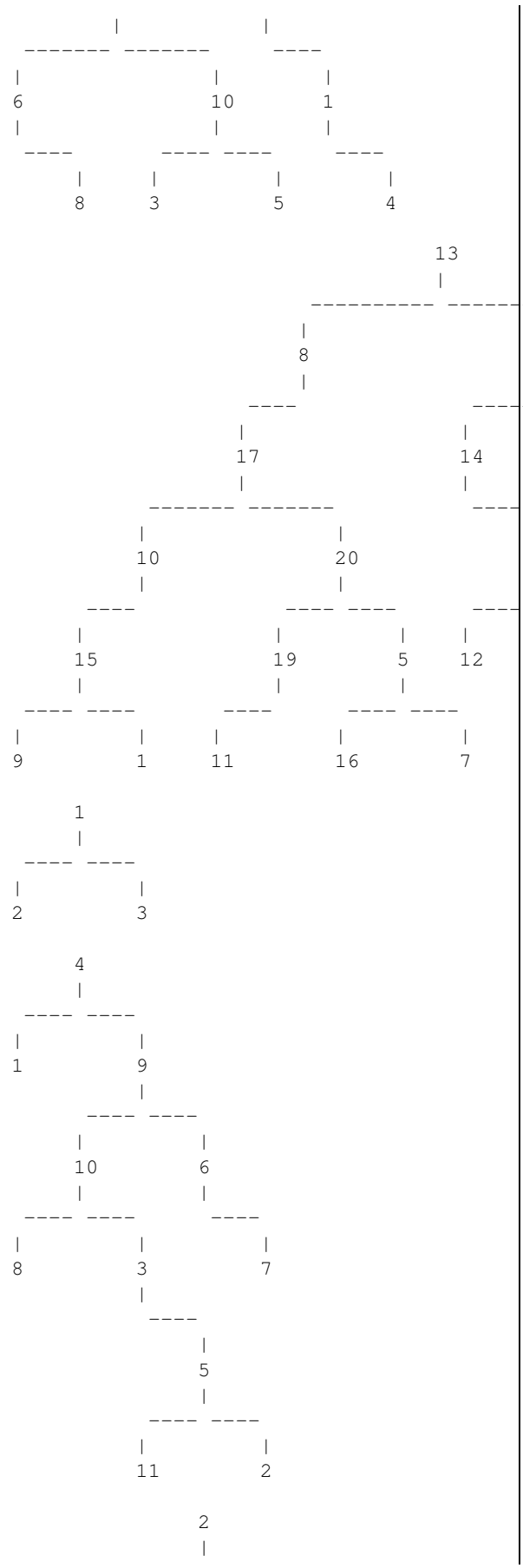
Ejemplo de entrada 1

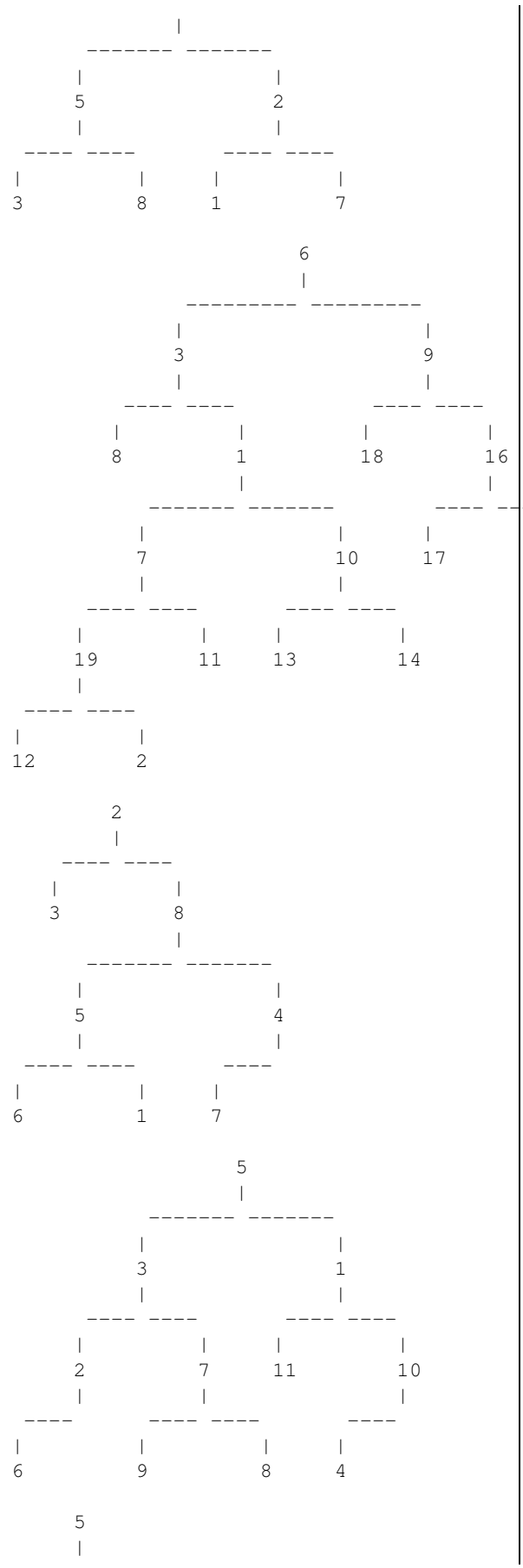
```
VISUALFORMAT

12
12 8 7 3 1 6 10 11 5 2 9 4
12 7 1 3 8 10 5 2 11 6 4 9

9
2 3 9 7 6 5 4 1 8
2 9 3 7 5 4 6 8 1

6
4 3 2 6 1 5
4 2 3 1 6 5

12
7 11 6 8 2 3 10 5 9 12 1 4
7 8 6 3 5 10 2 4 1 12 9 11

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

3
2 1 3
2 3 1

11
1 4 8 10 3 11 5 2 9 6 7
1 8 11 2 5 3 10 7 6 9 4

6
6 5 3 1 4 2
6 1 3 5 4 2

16
10 2 16 1 9 11 12 13 3 14 5 6 7 4 8 15
10 16 2 9 12 11 13 14 7 6 5 3 1 8 15 4

21
15 17 3 6 8 10 21 16 20 13 12 11 19 14 9 7 1 5 2 18 4
15 3 17 6 21 10 20 13 16 8 19 11 7 9 14 5 18 2 4 1 12

8
4 3 5 8 6 1 2 7
3 8 5 1 7 2 6 4

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

8
3 2 6 5 1 8 7 4
3 6 1 5 7 4 8 2

11
6 2 3 9 7 8 5 11 1 4 10
6 2 9 8 7 3 11 4 10 1 5

6
5 6 4 1 3 2
6 4 3 2 1 5

7
```

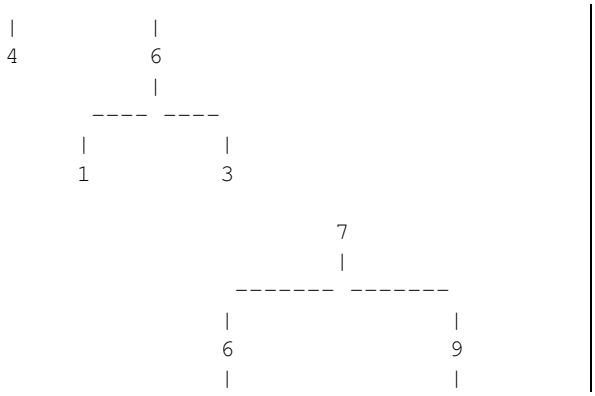
```
5 1 6 7 2 3 4
5 7 6 2 1 4 3

3
2 3 1
1 3 2

8
1 7 3 8 5 4 6 2
1 3 7 8 4 2 6 5

6
5 4 2 1 6 3
4 1 3 6 2 5

11
10 8 5 11 6 2 7 4 1 9 3
10 8 11 5 2 6 1 4 3 9 7
```

Ejemplo de salida 1









Ejemplo de entrada 2

```
INLINEFORMAT

12
12 8 7 3 1 6 10 11 5 2 9 4
12 7 1 3 8 10 5 2 11 6 4 9

9
2 3 9 7 6 5 4 1 8
2 9 3 7 5 4 6 8 1

6
4 3 2 6 1 5
4 2 3 1 6 5

12
7 11 6 8 2 3 10 5 9 12 1 4
7 8 6 3 5 10 2 4 1 12 9 11

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

3
2 1 3
2 3 1

11
1 4 8 10 3 11 5 2 9 6 7
1 8 11 2 5 3 10 7 6 9 4

6
6 5 3 1 4 2
6 1 3 5 4 2

16
10 2 16 1 9 11 12 13 3 14 5 6 7 4 8 15
10 16 2 9 12 11 13 14 7 6 5 3 1 8 15 4

21
15 17 3 6 8 10 21 16 20 13 12 11 19 14 9 7 1 5 2 18 4
15 3 17 6 21 10 20 13 16 8 19 11 7 9 14 5 18 2 4 1 12

8
4 3 5 8 6 1 2 7
3 8 5 1 7 2 6 4

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

8
3 2 6 5 1 8 7 4
3 6 1 5 7 4 8 2

11
6 2 3 9 7 8 5 11 1 4 10
6 2 9 8 7 3 11 4 10 1 5

6
5 6 4 1 3 2
6 4 3 2 1 5

7
5 1 6 7 2 3 4
5 7 6 2 1 4 3

3
2 3 1
1 3 2

8
1 7 3 8 5 4 6 2
1 3 7 8 4 2 6 5

6
5 4 2 1 6 3
4 1 3 6 2 5

11
10 8 5 11 6 2 7 4 1 9 3
10 8 11 5 2 6 1 4 3 9 7
```

Ejemplo de salida 2

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

```
12(8(6(17(15,3),),16(10(,21),13(20,))),1(14(11(,19),9(
4(,6(5(3,8),2(1,7)))
6(3(8,1(7(19(12,2),11),10(13,14))),9(18,16(17,15(,4(5,
2(3,8(5(6,1),4(7,)))
5(3(2(6,),7(9,8)),1(11,10(4,)))
5(,1(4(6,),2(3,)))
3(1(5,2(6,7))),4)
2(,3(,1))
5(8(7(1,3),),6(4,2))
5(,2(4,6(1,3)))
7(6(5(8(10,),11),2),9(4(,1),3))
```

Observación

AVISO: Este problema requiere un análisis a fondo antes de empezar a programar, porque hay que pensar detenidamente la estrategia y sobre todo los detalles. La solución no es larga pero no llegaréis por "ensayo y error", es imperativo usar papel y lápiz primero. Los conceptos que hay que tener muy presentes mentalmente para poder llegar a la solución son:

- la definición recursiva del inorden y el postorden, y la posición de la raíz en los dos casos,
- cómo hacer una *inmersión*, es decir, utilizar parámetros en las llamadas recursivas para establecer el contexto en el que se ejecuta cada llamada.

En particular, hay que notar que si tanto inorden `I` como postorden `P` se almacenan en vectores, los inordenes y postordenes de los subárboles son subsecuencias de los vectores `I` y `P`, con posiciones iniciales y finales que son índices en `I` y `P`.

Vuestra solución ha de trabajar con `BinTree`, aunque podéis utilizar otras estructuras de datos presentadas en el curso como elemento de soporte.
Para tratar la entrada y la salida, podéis utilizar este esquema:

```
#include <iostream>
using namespace std;
#include "BinTree.hh"
/* Puedes añadir más includes */

typedef BinTree<int> BT;

/* Las funciones que quieras */

int main() {
    string format;
    cin >> format;
    int n;
    while (cin >> n) {
        /* Leer dos secuencias de n enteros cada una. */

        BinTree t;
        /* Calcular t a partir de las dos secuencias. */
```

```
        t.setOutputFormat(format=="INLINEFORMAT"?  BT::INLINEFORMAT : BT::VISUA
        cout << t << endl;
    }
}
```

Evalución sobre 10 puntos:

- Solución lenta: 5 puntos.
- Solución rápida: 10 puntos.

Entendemos como solución rápida una que es correcta, de coste lineal y capaz de superar los juegos de pruebas públicos y privados. Entendemos como solución lenta una que no es rápida, pero es correcta y capaz de superar los juegos de pruebas públicos.

Información del problema

Autor : PRO2
Generación : 2024-10-10 12:11:55