

Problem statements

HighOfNode

Tạo một cây nhị phân tìm kiếm T lưu trữ các số nguyên không trùng nhau (giá trị số nguyên < 1000, cây T có thể rỗng). Sau đó tính chiều cao h của một node (biết $h \geq 0$).

INPUT

Số nguyên đầu tiên là node cần tìm chiều cao.

Số nguyên tiếp theo là gốc của cây. Các số nguyên còn lại sẽ được lần lượt thêm vào cây theo thứ tự nhập. Kết thúc khi gặp -1.

OUTPUT

Xuất chiều cao h của node. Nếu node không có trong cây thì xuất "-1". Nếu cây rỗng thì xuất "Empty Tree."

EXAMPLE

Input	Output
60 30 15 50 40 60 55 70 65 80 -1 // Minh họa: <pre> 30 / \ 15 50 / \ 40 60 / \ 55 70 / \ 65 80</pre>	2

15	
30 15 5 20 50 40 60 55 70 -1	
// Minh họa:	
<pre> 30 / \ 15 50 / \ / \ 5 20 40 60 / \ 55 70 </pre>	1
25	
30 15 5 20 50 40 60 55 70 -1	
// Minh họa:	
<pre> 30 / \ 15 50 / \ / \ 5 20 40 60 / \ 55 70 </pre>	-1
8	
-1	Empty Tree.

CountNodeHave1Child

Tạo một cây nhị phân tìm kiếm T lưu trữ các số nguyên (giá trị số nguyên < 1000, cây T có thể rỗng). Sau đó đếm xem trong cây có bao nhiêu node có 1 cây con.

INPUT

Số nguyên đầu tiên là gốc của cây. Các số nguyên sẽ được lần lượt thêm vào cây theo thứ tự nhập. Kết thúc khi gặp -1.

OUTPUT

Xuất số lượng node có 1 cây con trong cây.

EXAMPLE

Input	Output
6 5 7 2 8 1 -1 // Minh họa: <pre> 6 / \ 5 7 / \ 2 8 / 1 </pre>	3 // Node có 1 con: 5, 7, 2
4 8 1 9 -1 // Minh họa: <pre> 4 / \ 1 8 \ 9 </pre>	1 // Node có 1 con: 8
6 -1	0
-1	0

LNR_BST(No_stack_recursive)

Tạo một cây nhị phân tìm kiếm T lưu trữ các số nguyên (giá trị số nguyên < 1000, cây T có thể rỗng). Sau đó in cây theo thứ tự LNR. **Lưu ý dùng stack.**

INPUT

Số nguyên đầu tiên là gốc của cây. Các số nguyên sẽ được lần lượt thêm vào cây theo thứ tự nhập. Kết thúc khi gặp -1.

OUTPUT

Xuất theo LNR. Các giá trị cách nhau một khoảng trắng. Nếu cây rỗng thì in ra "Empty Tree."

EXAMPLE

Input	Output
6 5 7 2 8 1 -1	1 2 5 6 7 8
4 8 1 9 -1	1 4 8 9
-1	Empty Tree.