Tinkering With Trees

By: Ricardo & Billy

Deskripsi

AVL tree merupakan sebuah self balancing BST dimana setiap nodenya mempertahankan perbedaan tinggi antara subtree kiri dan subtree kanannya. Terdapat 4 kemungkinan kejadian untuk terjadinya rotasi yaitu left case, left right case, right case, right left case. Tugas anda adalah untuk mencetak detail dari setiap rotasi yang terjadi saat serangkaian angka dimasukkan ke dalam pohon

Format Masukan

Baris pertama berisi sebuah bilangan bulat **N**, yang menandakan jumlah angka yang akan dimasukkan ke dalam tree.
N Baris selanjutnya berisi bilangan bulat $\mathbf{a_i}$ yang akan dimasukkan ke dalam AVL tree secara berurutan

Format Keluaran

Untuk setiap rotasi yang terjadi, cetak detail dalam format berikut:

1. Left Case	*Left Case* *PivotNode: <data_pivot>* *PivotNode->left: <data_anak_kiri>* *PivotNode->left->left: <data_anak_kiri_kiri>* *result:* *NewNode: <data_root_baru>* *NewNode->left: <data_anak_kiri_baru>* *NewNode->right: <data_anak_kanan_baru>*
2. Right Case	*Right Case* *PivotNode: <data_pivot>* *PivotNode->right: <data_anak_kanan>* *PivotNode->right->right: <data_anak_kanan_kanan>* *result:* *NewNode: <data_root_baru>* *NewNode->left: <data_anak_kiri_baru>* *NewNode->right: <data_anak_kanan_baru>*
3. Left Right Case	*Left Right Case* *PivotNode: <data_pivot>* *PivotNode->left: <data_anak_kiri>*

	PivotNode->left->right: <data_anak_kiri_kanan> *result:* *NewNode: <data_root_baru>* *NewNode->left: <data_anak_kiri_baru>* *NewNode->right: <data_anak_kanan_baru>*
4. Right Left Case	*Right Left Case* *PivotNode: <data_pivot>* *PivotNode->right: <data_anak_kanan>* *PivotNode->right->left: <data_anak_kanan_kiri>* *result:* *NewNode: <data_root_baru>* *NewNode->left: <data_anak_kiri_baru>* *NewNode->right: <data_anak_kanan_baru>*

Catatan:

- Untuk rotasi ganda (*left-right* dan *right-left*), hanya **hasil akhir setelah kedua rotasi selesai** yang perlu dicetak.
- **Seluruh test case yang diberikan akan menimbulkan rotasi avl setidaknya satu kali**

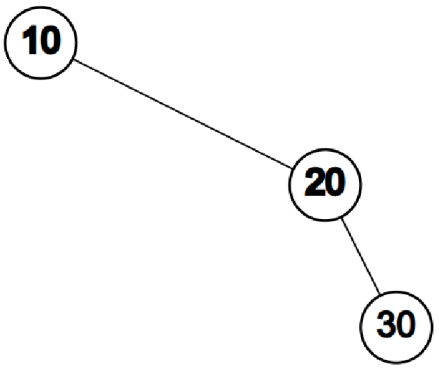
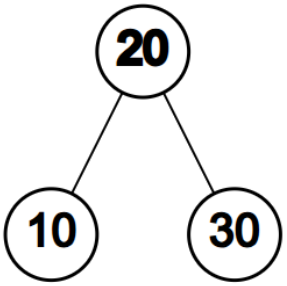
Batasan

$3 \leq N \leq 100$

$0 < a_i \leq 10000$

Tidak ada node duplikat

Contoh

Sample Input 0
3 10 20 30
Sample Output 0
Right Case PivotNode: 10 PivotNode->right: 20 PivotNode->right->right: 30 result: NewNode: 20 NewNode->left: 10 NewNode->right: 30
Penjelasan Sample 0
   Setelah memasukkan node 30, tree akan menjadi right skewed, sehingga dilakukan left rotation sekali hingga tree menjadi seperti gambar di bawah: 

Sample Input 1
5 10 30

20
25
26

Sample Output 1

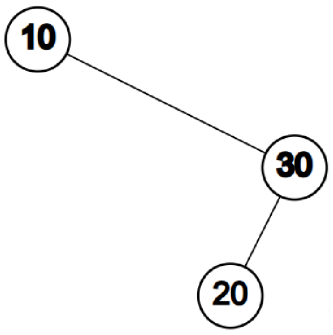
Right Left Case
PivotNode: 10
PivotNode->right: 30
PivotNode->right->left: 20

result:
NewNode: 20
NewNode->left: 10
NewNode->right: 30

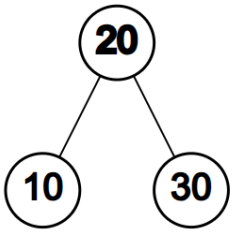
Left Right Case
PivotNode: 30
PivotNode->left: 25
PivotNode->left->right: 26

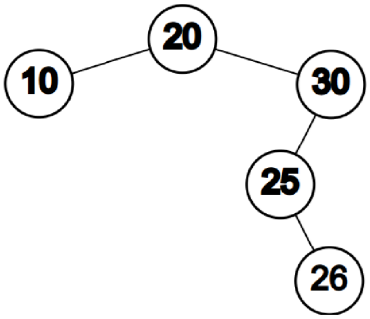
result:
NewNode: 26
NewNode->left: 25
NewNode->right: 30

Penjelasan Sample 1

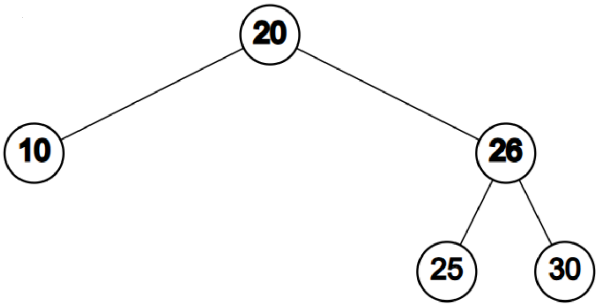


Setelah memasukkan node 20, tree akan menjadi right-left zigzag, sehingga dilakukan right rotation pada node 30 lalu left rotation pada node 10 sehingga tree menjadi seperti gambar di bawah:





Lalu setelah memasukkan 25 dan 26, Tree akan menjadi Left Right zigzag, sehingga dilakukan left rotation pada node 25 lalu right rotation pada node 30 sehingga tree menjadi seperti gambar di bawah:



Tinkering With Trees

By: Ricardo & Billy

Description

AVL tree is a self balancing BST where each node maintains the height difference between the left and right subtrees. There are 4 possibilities that could make the rotations happen, that is left skewed, left right zigzag, right skewed, and right left zigzag. Your task is to print the details of every rotation that happens when a series of integers are inserted to the tree.

Input Format

The first line contains integer **N**, indicating the number of integers to be inserted into the tree.
The next **N** lines each contains an integer $\mathbf{a_i}$ to be inserted into the AVL tree in sequence

Output Format

For each rotation that occurs, print the details corresponding to the identified case. The format is as follows:

1. Left Case	*Left Case* *PivotNode: <pivot_data>* *PivotNode->left: <left_child_data>* *PivotNode->left->left: <left_left_child_data>* *result:* *NewNode: <new_root_data>* *NewNode->left: <new_left_child_data>* *NewNode->right: <new_right_child_data>*
2. Right Case	*Right Case* *PivotNode: <pivot_data>* *PivotNode->right: <right_child_data>* *PivotNode->right->right: <right_right_child_data>* *result:* *NewNode: <new_root_data>* *NewNode->left: <new_left_child_data>* *NewNode->right: <new_right_child_data>*
3. Left Right Case	*Left Right Case* *PivotNode: <pivot_data>* *PivotNode->left: <left_child_data>*

	PivotNode->left->right: <left_right_child_data> *result:* *NewNode: <new_root_data>* *NewNode->left: <new_left_child_data>* *NewNode->right: <new_right_child_data>*
4. Right Left Case	*Right Left Case* *PivotNode: <pivot_data>* *PivotNode->right: <right_child_data>* *PivotNode->right->left: <right_left_child_data>* *result:* *NewNode: <new_root_data>* *NewNode->left: <new_left_child_data>* *NewNode->right: <new_right_child_data>*

Notes:

- For double rotations, you only need to print the final state of the tree **after both rotations are completed**
- **Every test case is guaranteed to trigger at least one rotation**

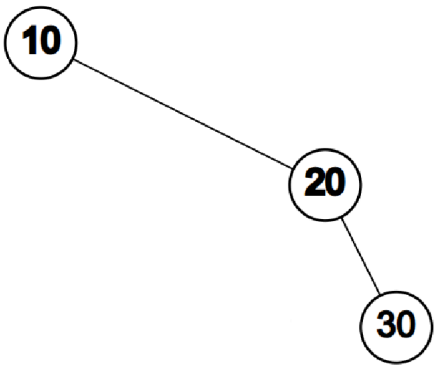
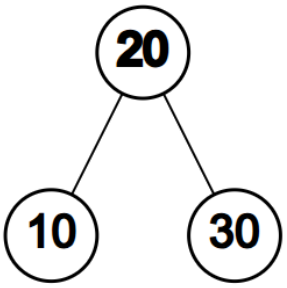
Constraints

$3 \leq N \leq 100$

$0 < a_i \leq 10000$

No duplicate numbers will be inserted

Examples

Sample Input 0
3 10 20 30
Sample Output 0
Right Case PivotNode: 10 PivotNode->right: 20 PivotNode->right->right: 30 result: NewNode: 20 NewNode->left: 10 NewNode->right: 30
Sample 0 Explanation
   As visualized above, after inserting 30, the tree will become right skewed. To balance the tree, left rotation is done on node 10. The result of this rotation is visualized on the image below: 

Sample Input 1
5 10

30
20
25
26

Sample Output 1

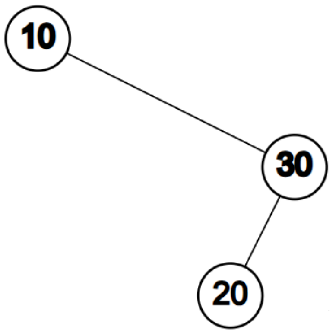
Right Left Case
PivotNode: 10
PivotNode->right: 30
PivotNode->right->left: 20

result:
NewNode: 20
NewNode->left: 10
NewNode->right: 30

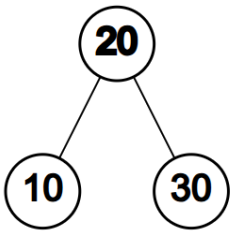
Left Right Case
PivotNode: 30
PivotNode->left: 25
PivotNode->left->right: 26

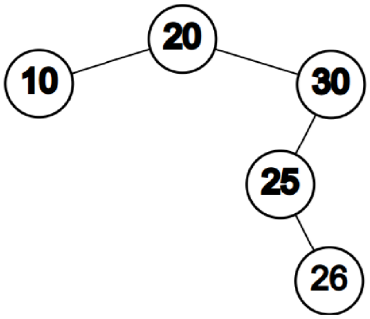
result:
NewNode: 26
NewNode->left: 25
NewNode->right: 30

Sample 1 Explanation



As visualized above, after entering node 20, the tree becomes right-left zigzag. To balance the tree, right rotation is done on node 30 and then left rotation is done on node 10. The result of this rotation is visualized on the image below:





As visualized above, after entering 25 and 26, the tree will become left-right zigzag. To balance the tree, left rotation is done on node 25 and then right rotation is done on node 30. The result of this rotation is visualized on the image below: