Balanceamento da árvore

Árvores binárias de busca são estruturas de dados relativamente eficientes, contanto que estejam balanceadas. Uma árvore desbalanceada pode ter pior caso $O(n)$ para todas as operações, o que pode ser tão ruim quanto uma lista ligada ordenada.
Entretanto, dada uma árvore potencialmente desbalanceada, podemos reconstruí-la de modo que os elementos estejam dispostos na altura mais baixa possível. Para isso, basta primeiro fazer um percurso in-ordem na árvore original e inserir os elementos em uma lista. Como resultado teremos uma lista $L=(a_0,a_1,a_2,...,a_n)$ na qual todos os elementos originais da árvore estão ordenados.
Para criar uma árvore balanceada a partir dessa lista, precisamos selecionar o elemento central de L, $a_{n/2}$. Ele é o mais indicado para ser a raiz, pois aproximadamente metade dos elementos têm valores menores que o dele e aproximadamente metade têm valores maiores. O filho esquerdo dele será o elemento central da sub-lista esquerda $(a_1,a_1,a_2,...,a_{n//2-1})$ e o seu filho direito será o elemento central da sub-lista
direita $(a_{n//2+1},a_{n//2+2},...,a_n)$. Se continuarmos processando essas sub-listas recursivamente, inserindo os elementos na árvore de maneira sistemática, o resultado será uma árvore tão balanceada quanto possível.
Neste exercício você deverá criar um programa que lê uma árvore desbalanceada e, usando o procedimento explicado acima, cria uma nova árvor balanceada. Tenha cuidado na hora de selecionar o elemento central de cada lista e sub-lista, de modo que a sua árvore será exatamente aquela esperada no exercício.

Após construir a nova árvore balanceada, os resultados dos percursos pré-ordem e pós-ordem.

Entrada

A entrada será uma lista em notação Python contendo os elementos na ordem em que eles devem ser inseridos na árvore desbalanceada (inicialmente vazia).

Saída

Imprima a palavra `pre:`, incluindo os dois-pontos, seguido do percurso pré-ordem da árvore reconstruída.

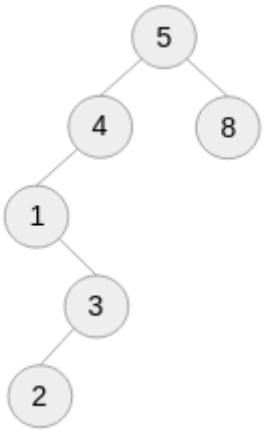
Na linha seguinte, imprima a palavra `pos:`, incluindo o dois-pontos, seguido do percurso pós-ordem da árvore reconstruída.

Observações

Dada a sub-lista $(a_i,a_{i+1},a_{i+2},...,a_j)(\text{??},\text{??}+1,\text{??}+2,\ldots,\text{??})$, na qual $a_{\text{??}}$ é o elemento mais à esquerda da sub-lista e $a_{\text{??}}$ é o elemento mais a direita, calcule o índice do elemento central como sendo `c = (i + j) // 2`.

Exemplo ilustrado 1

A entrada do primeiro exemplo é `[5,4,8,1,3,2]`. Então a árvore desbalanceada desse caso de teste é a seguinte:



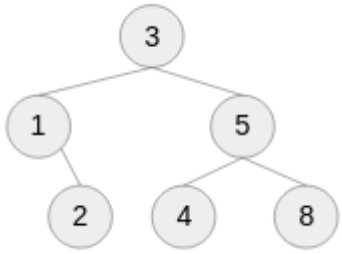
O percurso in-ordem dessa árvore produz a lista que contém todos os elementos da árvore em ordem, isto é, $L_1=(1,2,3,4,5,8)\text{?}1=(1,2,3,4,5,8)$. O elemento central dessa lista é o 33. Ele será a raiz da nova árvore.

A escolha desse elemento como central produz duas sub-listas, que são a sub-lista esquerda $L_2=(1,2)\text{?}2=(1,2)$ e a sub-lista direita $L_3=(4,5,8)\text{?}3=(4,5,8)$. O elemento central de $L_2\text{?}2$ é 11, então esse é o próximo elemento a ser inserido na árvore.

A escolha de 11 como elemento central de $L_2\text{?}2$ produz duas novas sub-listas. Uma delas é vazia e a outra contém apenas o elemento 22. Então esse é o próximo a ser inserido na árvore.

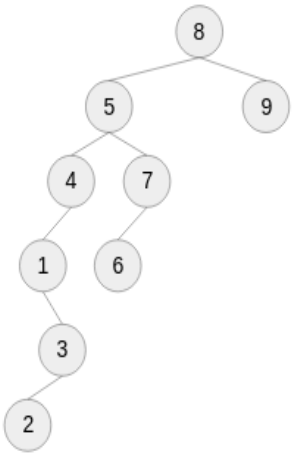
Agora voltamos a olhar para a sub-lista $L_3=(4,5,8)\text{?}3=(4,5,8)$. O elemento central é o 55 e é ele quem vamos inserir agora na árvore. Depois vamos inserir os elementos 44 e 88.

A ordem na qual inserimos os elementos na nova árvore foi a seguinte: $3,1,2,5,4,83,1,2,5,4,8$. A árvore balanceada resultante é a mostrada na figura abaixo.

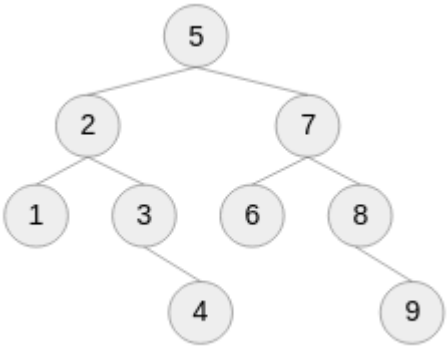


Exemplo ilustrado 2

A entrada do segundo exemplo é `[8,5,9,4,7,1,6,3,2]`. A árvore inicial é a seguinte:



O percurso in-ordem produz a lista
inicial $L_1=(1,2,3,4,5,6,7,8,9)$ $1=(1,2,3,4,5,6,7,8,9)$.
A inserção dos elementos na nova árvore será na ordem
seguinte: $5,2,1,3,4,7,6,8,95,2,1,3,4,7,6,8,9$.
A árvore balancada resultante será como na figura abaixo.



Dicas

A implementação ideal dos caminhamentos pré-ordem, in-ordem e pós-ordem é por meio de duas funções recursivas. Você pode imprimir os nós visitados na própria função.

Por padrão, a função `print()` imprime uma quebra de linha após imprimir as expressões que ela recebe. Para imprimir todo o percurso em uma só linha, passe o argumento `end=' '`. Desse modo a impressão será finalizada com um espaço.

Por exemplo, os dois comandos a seguir:

```
print('Hello', end=' ')
print('world!', end=' ')
print()
```

Imprimem a mensagem `"Hello world! "` em uma única linha, seguidos de uma quebra de linha.

Exemplos de Entrada e Saída

Entrada	`[5,4,8,1,3,2]`
Saída	`pre: 3 1 2 5 4 8` `pos: 2 1 4 8 5 3`
Entrada	`[8,5,9,4,7,1,6,3,2]`
Saída	`pre: 5 2 1 3 4 7 6 8 9` `pos: 1 4 3 2 6 9 8 7 5`