CONCEITOS DE BALANCEAMENTO

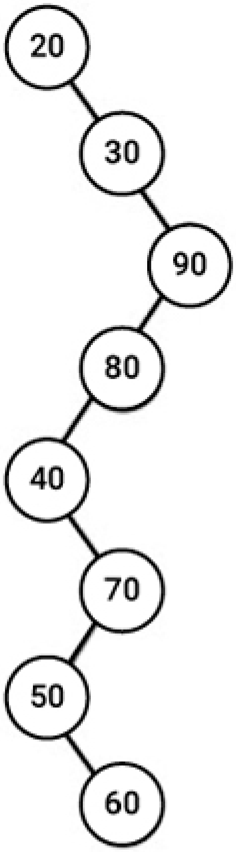
Balanceamento em árvores

Árvores binárias de busca são estruturas de dados dinâmicas, isto é, capazes de realizar busca, inserção e remoção de chaves, uma a uma, sem perder suas propriedades nem necessitar de processamento suplementar para manter suas características. Entretanto, a estrutura de dados é complexa e não apresenta ganho em relação a uma massa de dados desorganizada, isto é, ambas, no pior caso, necessitam de **O**(**n**) operações para busca, o que implica **O**(**n**) para inserção e remoção. Ou seja, a árvore binária de busca não é melhor em termos de performance computacional que uma lista sem organização alguma.

Por que estudar uma estrutura de dados mais complexa que uma lista, se sua complexidade computacional é a mesma?

A resposta para esta pergunta requer uma análise do estudo da complexidade computacional dos algoritmos de busca, inserção e remoção em árvores binárias de busca.

A complexidade computacional da busca é proporcional à altura da árvore, e que, no pior caso, uma árvore tem altura **n**. A família de árvores com altura **n** é chamada de árvores zig-zag. A imagem a seguir mostra uma árvore zig-zag de altura 8.



Isto é, as árvores zig-zag são a família de árvores binárias de busca com pior desempenho possivel, uma vez que a complexidade da busca e, consequentemente, da inserção e da remoção é **O**(**n**) e não é possível construir uma árvore binária de busca com altura maior que **n**.

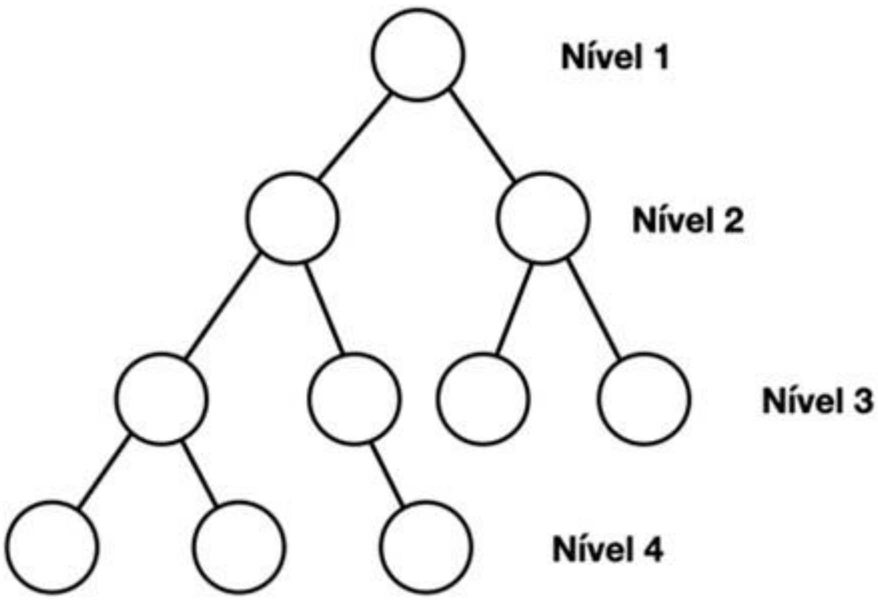
Se as zig-zag são as piores árvores (piores no sentido que a complexidade e a manutenção da estrutura são as mesmas de uma lista desorganizada), qual a família com melhor desempenho?

Para responder a essa pergunta, recordaremos alguns conceitos de árvore binárias.

Árvores binárias

Diz-se que uma árvore binária **T** é completa se os nós de **T** com subárvores vazias estão no último ou no penúltimo nivel.

A árvore da imagem a seguir é uma árvore completa. Os nós que possuem subárvores vazias estão no nível 3 e no nível 4, respectivamente, no penúltimo e no último nível da árvore.



Um resultado importante sobre as árvores binárias completas é o que se segue:

Seja **T** uma árvore binária completa com **n**>0 nós. Então **T** possui altura mínima e $h=1+\lfloor \log \mathbf{n} \rfloor$.

Esse resultado é muito importante, mostra que não existe árvore binária com **n** nós com altura inferior à $h=1+\lfloor \log \mathbf{n} \rfloor$. Ou seja, a melhor árvore binária que podemos construir tem altura proporcional a log **n**.

Se limitarmos a construção das árvores binárias de busca às árvores binárias completas, temos os algoritmos de busca, inserção e remoção funcionando em log **n**.

Esse é o objetivo, melhorar a complexidade computacional da busca, inserção e remoção.

De forma ampla, dizemos que árvores binárias cujas alturas são proporcionais a log n são árvores balanceadas. Ou seja, árvores binárias completas são balanceadas, mas será que toda árvore balanceada é completa?

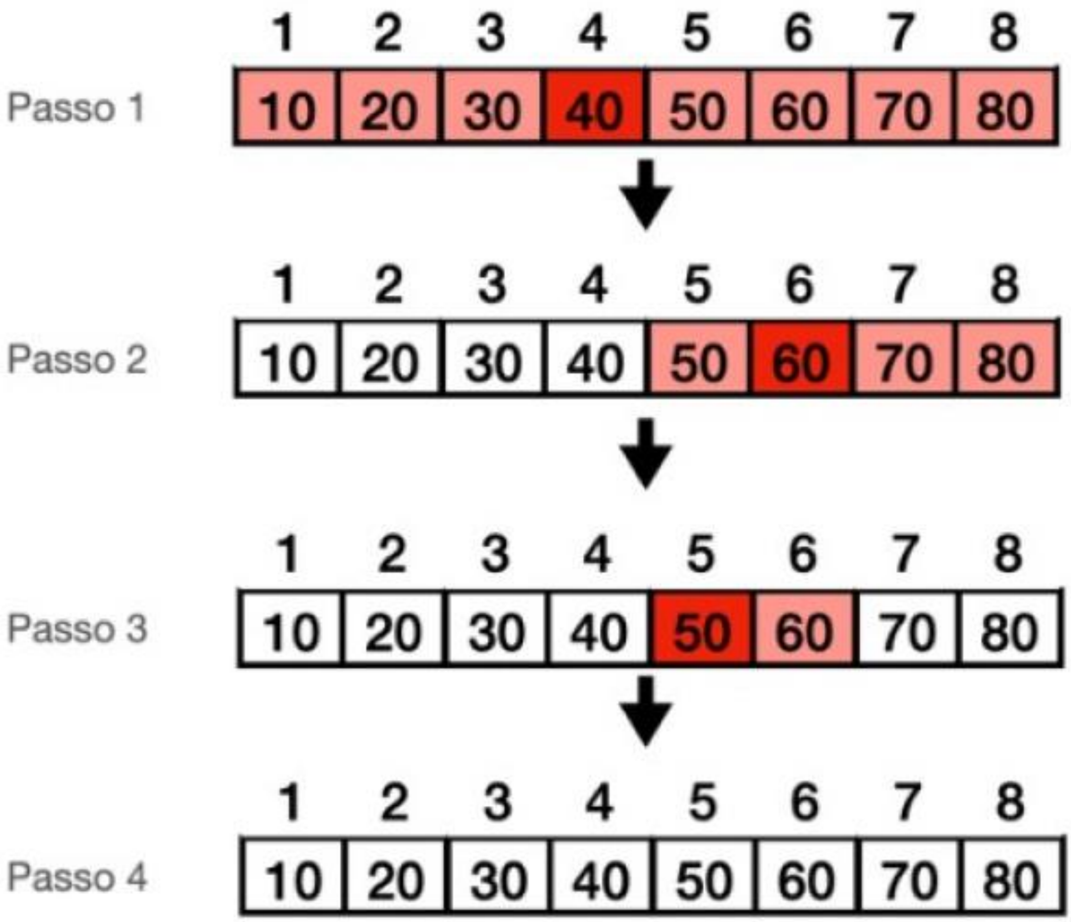
Não, existem outras famílias de árvores binárias, neste caso especificamente, de busca, que são balanceadas, porém não são completas, por exemplo, as árvores AVL e as árvores rubro-negras.

Construindo árvores balanceadas

Antes de estudarmos as estruturas de dados completamente dinâmicas que fornecem árvores binárias de busca balanceadas, isto é, com altura de **O**(log **n**), vamos estudar o algoritmo que transforma uma árvore binária de busca em uma árvore binária de busca completa.

Existem várias formas de se construir uma árvore binária de busca completa. A mais intuitiva é derivada da pesquisa binária, que é um método de busca em um vetor ordenado que se baseia na estratégia "dividir para conquistar". A ideia é simples, compara-se a chave que está se buscando com a chave armazenada no elemento central do vetor, isto é, se o vetor tem tamanho **n**, o elemento **n**/2. Caso a chave buscada seja menor que o elemento armazenado na posição **n**/2, aplica-se o método recursivamente na primeira metade do vetor, caso contrário, na segunda metade.

Esse método de busca é eficiente, uma vez que é capaz de realizar a busca em **O**(log **n**). Um exemplo de aplicação desse método pode ser visto na imagem 3, que ilustra a busca da chave "55".

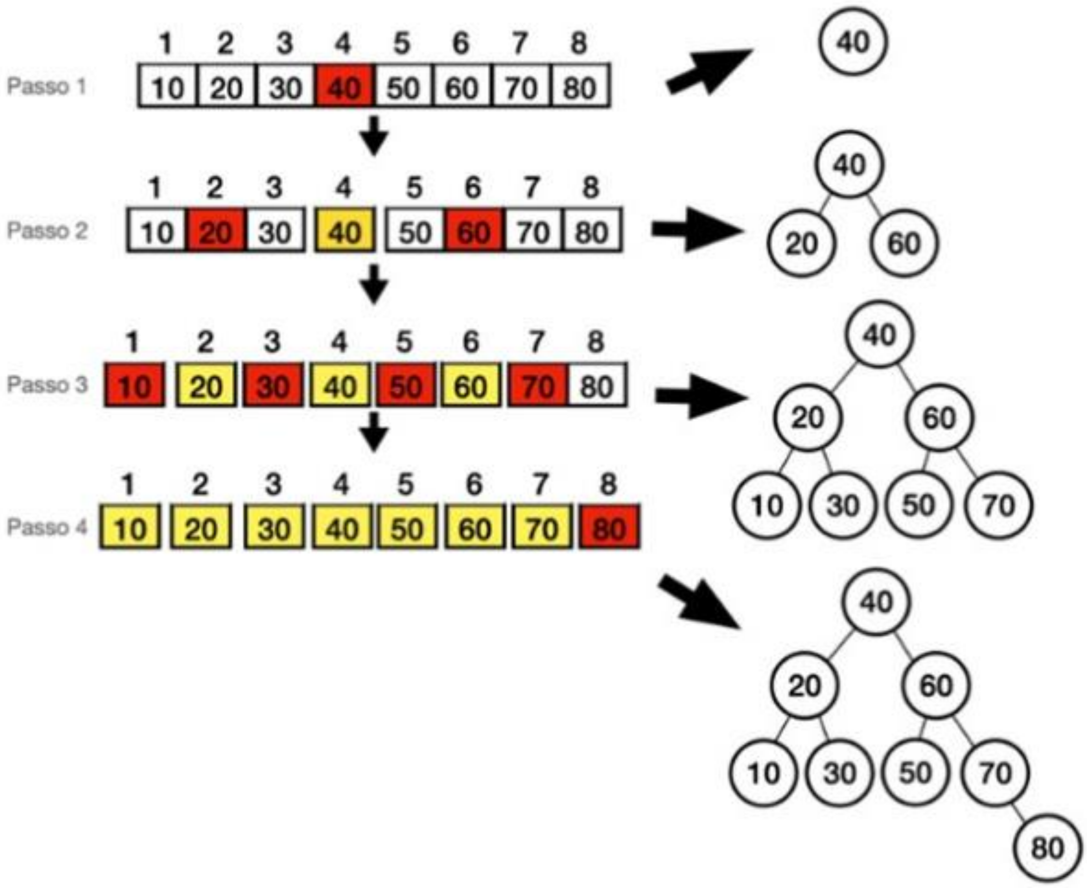


1. Compara-se a chave central 8/2 = 4, que armazena a chave “40”, com a chave “55”. Assim, se “55” estiver na estrutura de dados, estará na segunda metade.

2. Compara-se a chave central (5+8)/2=6, que armazena a chave “60”, com a chave “55”. Assim, se “55” estiver na estrutura, estará na primeira metade.

3. Compara-se a chave central (5+6)/2=5, que armazena a chave “50”, com a chave “55”. Assim, se “55” estiver no vetor, estará na segunda metade, que é o elemento 6 do vetor, que não armazena a chave “55”, terminando, assim, a busca binária.

Como realizamos **O**(log **n**) comparações no pior caso, que é não encontrar a chave buscada, se transformarmos as comparações possíveis em uma árvore, teremos uma árvore com altura proporcional à log **n**, ou seja, uma árvore balanceada. A imagem a seguir ilustra o processo para o mesmo vetor:



A cada passo, elegemos uma raiz, elemento central, e aplicamos recursivamente o método nas metades esquerda e direita, que nos fornece as raízes das subárvores esquerda e direita recursivamente.

O método mostrado constrói uma árvore balanceada, uma vez que o número de níveis da árvore deriva do número de comparações na pesquisa binária, e esse número é log **n**. Na verdade, podemos ver que construímos uma árvore binária de busca completa, que também é balanceada.

Esse método de construção, apesar de intuitivo, possui diversas desvantagens. Precisamos de um vetor auxiliar e a sequência de chaves ordenadas, o que, sem dúvida, aumenta a necessidade de alocação de memória. O ideal é aplicar um algoritmo que resolva o problema de construir uma árvore binária de busca sem utilizar nenhuma estrutura de dados auxiliar.

Aplicações das árvores balanceadas

Árvores binárias balanceadas possuem diversas aplicações teóricas e práticas, embora elas sejam amplamente utilizadas na teoria e em melhoramento de performance algorítmica. Vamos conferir!

1. Árvores binárias de busca balanceadas podem ser usadas para construir e manter listas ordenadas, tais como filas de prioridade. Podem também ser usadas para implementar qualquer algoritmo que requeira listas ordenadas, para alcançar o melhor desempenho assintótico.

2. Alguns algoritmos de geometria computacional exploram variações de árvores de busca balanceadas para resolver diversos problemas, tais como a interseção entre segmentos de reta e o problema de localização do ponto de forma eficiente.

3. Árvores AVL podem ser usadas para formar um dicionário de uma linguagem ou de programas, como os opcodes de um assembler ou um interpretador.

4. Árvores B são exemplos de árvores amplamente utilizadas para organizar dados em banco de dados e HD.