ÀRVORES AVL

Árvores AVL: principais definições

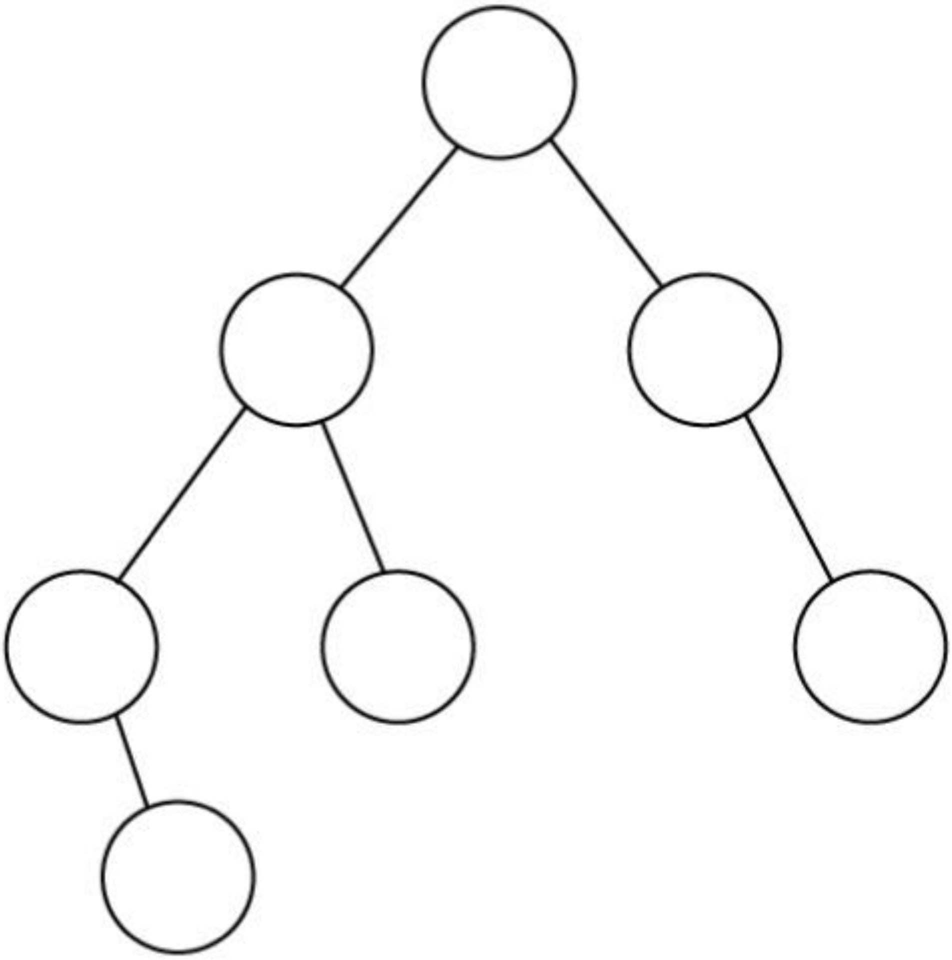
Ao estudar a complexidade dos algoritmos de construção de árvores (binárias) de busca, observamos que a estrutura de dados, apesar de complexa, não tinha desempenho teórico superior às listas desorganizadas. Isso se deve ao fato de que, no pior caso, a complexidade das operações de busca, inserção e remoção é **O**(**n**).

Em seguida, vimos que, para uma redução de complexidade, temos que ter árvores binárias de busca com altura proporcional à **O**(log **n**) e as árvores com essa propriedade são chamadas de árvores balanceadas.

A evolução desse cenário é a obtenção de uma estrutura de dados completamente dinâmica que suporte inserções, remoções e buscas em **O**(log **n**). As árvores AVL fornecem essa funcionalidade. A sigla AVL corresponde às iniciais dos autores Adel'son-Vel'skii e Landis (S7WARCFITFR; MARKFNZON, 1994)

Uma árvore binária de busca **T** é uma árvore AVL se, para qualquer nó **r** de **T**, vale a propriedade: $|h(\mathbf{Ter})-h(\mathbf{Tdr})|\leq 1$.

Observação: h(**Ter**) é a altura da subárvore esquerda de **r** e h(**Tdr**) é a altura da subárvore direita. Dizemos que um nó **r** de uma árvore binária de busca está regulado se $|h(\mathbf{Ter})-h(\mathbf{Tdr})|\leq 1$. Ou seja, em uma árvore AVL, todos os nós estão regulados. A imagem a seguir mostra o exemplo da topologia de uma árvore AVL.



As árvores completas são AVL. Esse fato deriva da própria definição das árvores completas. Além disso, sabemos que as árvores completas com **n** nós têm altura mínima, isto é, não existe topologia com **n** nós com altura inferior à da árvore completa (h=1+⌊log **n**⌋). Ou seja, esse resultado garante que a altura mínima de uma árvore AVL é proporcional a log **n**.

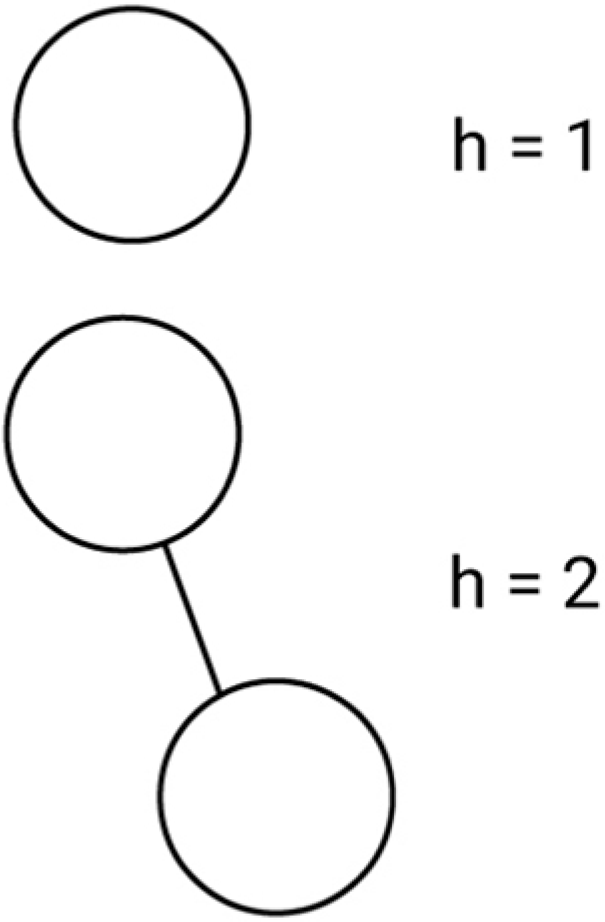
Entretanto, as árvores completas são o melhor caso, isto é, as árvores AVL de altura mínima. Vejamos a árvore AVL da imagem 5. Essa árvore não é completa, uma vez que existe um nó na topologia que tem subárvores vazias e este nó está no antepenúltimo nível da árvore. Assim, o teorema que versa sobre a altura das árvores completas não garante que a árvore da imagem 5 tem altura proporcional a log **n**.

Vamos analisar a família de árvores AVL com pior desempenho, isto é, dada uma quantidade **n** de nós, a maior altura possível. Se essa família de árvores AVL tiver altura proporcional a log **n**, então garantiremos que, no melhor e no pior caso, a altura de uma árvore AVL é proporcional a log **n**.

Vamos construir a família de árvores AVL com altura máxima e menor número de nós possível. A construção será recursiva:

- Para h=1 só existe uma solução, que é a árvore unitária.
- Para h=2, podemos construir a árvore com **n**=2 nós, a raiz e um dos filhos, à esquerda ou à direita, para fins de análise não há diferença.

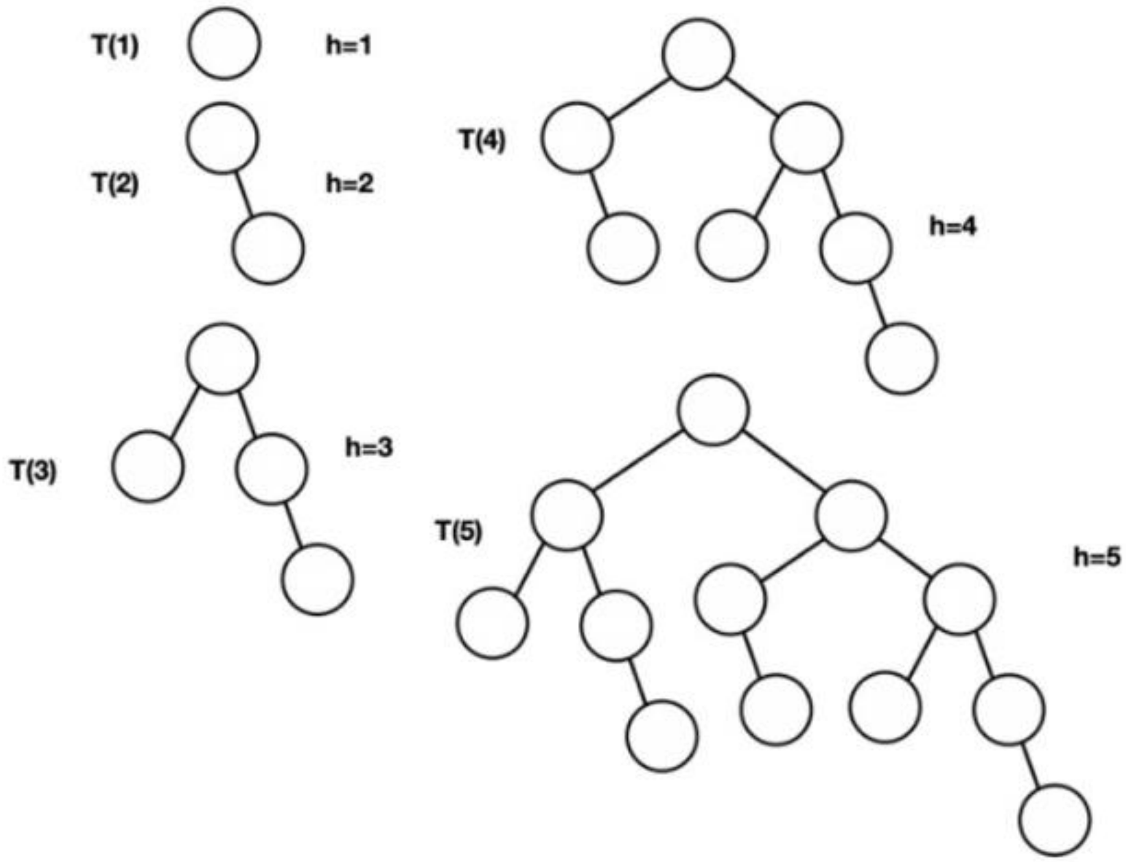
A imagem a seguir mostra essas topologias:



Para construir as árvores com altura h e número mínimo de nós, comporemos as árvores da seguinte forma recursiva:

1. Arbitra-se uma nova raiz.
2. Agregam-se a essa raiz como subárvore esquerda e direita as árvores de altura $\boldsymbol{h-1}$ e $\boldsymbol{h-2}$.

Assim, teremos as estruturas montadas da imagem a seguir:



Provamos que a árvore tem altura h e número mínimo de nós com a seguinte argumentação: em qualquer uma da árvores **T**(1) a **T**(5), qualquer folha que seja removida causa a destruição da propriedade AVL ou a redução da altura da árvore, isto é, ou a árvore deixa de ser AVL ou perde a altura proposta. Sendo assim, as topologias da imagem 7 representam as árvores AVL com altura h e número mínimo de nós. Observe que essas estruturas não são únicas. Dependendo de como fazemos a composição, alteramos a topologia, mas não o número de nós.

Observe também que podemos encontrar o número mínimo de nós em função da altura com a seguinte soma recursiva $|Th|=|Th-1|+|Th-1|+1$. Isto é: a cardinalidade da árvore com altura h é igual à soma da cardinalidade da árvore de altura $h-1$ com a cardinalidade da árvore de altura $h-2$ mais 1.

A tabela a seguir apresenta a evolução dessa série.

Árvore	h	$\lvert T_h \rvert$	F_h
T_1	1	1	1
T_2	2	2	1
T_3	3	4	2
T_4	4	7	3
T_5	5	12	5
T_6	6	20	8
T_7	7	33	13
T_8	8	54	21

Tabela 1 - Comparação da sequência **T**h com **F**h (sequência de Fibonacci).

Da imagem, vemos que, para $h>2$, $\mathbf{|T_h| = F_{h+2} - 1}$. Ou seja, o valor mínimo de **n** para que possamos construir uma árvore AVL com altura h é $\mathbf{n = F_{h+2} - 1}$. O termo geral da série de Fibonacci é:

$$\mathbf{F_n = 1/\sqrt{5}\ [(1+\sqrt{5}/2)^n - (1-\sqrt{5}/2)^n]}$$

Sendo assim, $|\mathbf{T}_h| = n$ é:

$$\mathbf{n = |T_h| = 1/\sqrt{5}\ [(1+\sqrt{5}/2)^{h+2} - (1-\sqrt{5}/2)^{h+2}] - 1}$$

$$\mathbf{n = 1/\sqrt{5}\ (1+\sqrt{5}/2)^{h+2} - 1/\sqrt{5}\ (1-\sqrt{5}/2)^{h+2} - 1}$$

$$\mathbf{n > 1/\sqrt{5}\ (1+\sqrt{5}/2)^{h+2}}$$

$$\log_{1+\sqrt{5}/2} > h+2- \log_{1+\sqrt{5}/2} \sqrt{5}$$

$$h < \log_{1+\sqrt{5}/2} n + \log_{1+\sqrt{5}/2} \sqrt{5} - 2$$

Isto é, a altura da árvore AVL é proporcional a log $\mathbf{n}$. Ou seja, as árvores AVL são **balanceadas**. A família de árvores AVL com essa propriedade é chamada de árvores de Fibonacci.

A conclusão desse estudo mostra que as melhores árvores AVL, isto é, as que têm altura mínima são as árvores completas e têm altura proporcional a log $\mathbf{n}$. As piores árvores AVL, isto é, que têm altura máxima e número mínimo de nós, que são as árvores de Fibonacci, também têm altura proporcional a log $\mathbf{n}$.

De fato, não determinamos a função matemática para calcular a altura de uma árvore AVL com **n** nós. Entretanto, sabemos que essa função é limitada superiormente pela altura das árvores de Fibonacci e inferiormente pela altura das árvores completas, ambas proporcionais a log $\mathbf{n}$.

Sendo assim, a função que calcula a altura de uma árvore AVL com **n** nós é "sanduichada" por duas funções logarítmicas com bases diferentes, sendo assim, o comportamento da altura de uma árvore AVL qualquer é proporcional a log **n**, garantindo que as árvores AVL são balanceadas.

Árvores AVL: busca

As operações básicas em uma árvore AVL geralmente envolvem os mesmos algoritmos de uma árvore de busca binária desbalanceada. No entanto, para manter o balanceamento, operações extra são necessárias e conhecidas como rotações de nós. Uma rotação simples ocorre quando um nó está desbalanceado e seu filho estiver no mesmo sentido da inclinação.

Busca em árvores AVL

Uma árvore AVL é uma árvore binária de busca, portanto, não existe diferença entre o algoritmo da busca em uma árvore binária de busca e o da busca em uma árvore AVL. O algoritmo é rigorosamente o mesmo.

Quando estudamos a busca nas árvores binárias de busca, concluímos que a complexidade era **O**(**n**). Isso acontece uma vez que o pior caso, isto é, as árvores com altura máxima e menor quantidade de nós são as árvores zig-zag, que têm altura **n**. Nas árvores AVL, as piores árvores, isto é, as árvores com altura máxima e menor quantidade de nós, são as árvores de Fibonacci que têm altura proporcional a log **n**.

Sendo assim, a complexidade do algoritmo da busca, quando aplicado em árvores AVL, é **O**(log **n**).

Atenção!

O que define a complexidade da busca é a altura da árvore, não o algoritmo propriamente dito. Isso se deve ao fato de que o algoritmo de busca tem como operação fundamental a comparação, e que esse algoritmo executa uma comparação por nível na árvore e, ainda, no pior caso, a chave encontrada está no último nível da árvore, que é igual à altura da árvore.

Inserção em árvores AVL

Seja **T** uma árvore AVL na qual vamos inserir uma nova chave. Como uma árvore AVL é uma árvore binária de busca, então o funcionamento do algoritmo de inserção, isto é, onde a nova chave deve ser inserida, é definido da mesma forma que já estudamos. A busca determina a posição, que é a subárvore vazia onde a busca deveria prosseguir para encontrar a chave que desejamos inserir.

Agora, vamos nos concentrar em verificar o que pode acontecer com a propriedade fundamental das árvores AVL, isto é, a regulagem dos nós. Vamos recordar que, em uma árvore AVL, deve valer para todos os nós $|h(\mathbf{Ter})-h(\mathbf{Tdr})|\leq \mathbf{1}$. Quando isso ocorre, dizemos que o nó está regulado. Para fins de análise, vamos considerar a árvore da imagem:

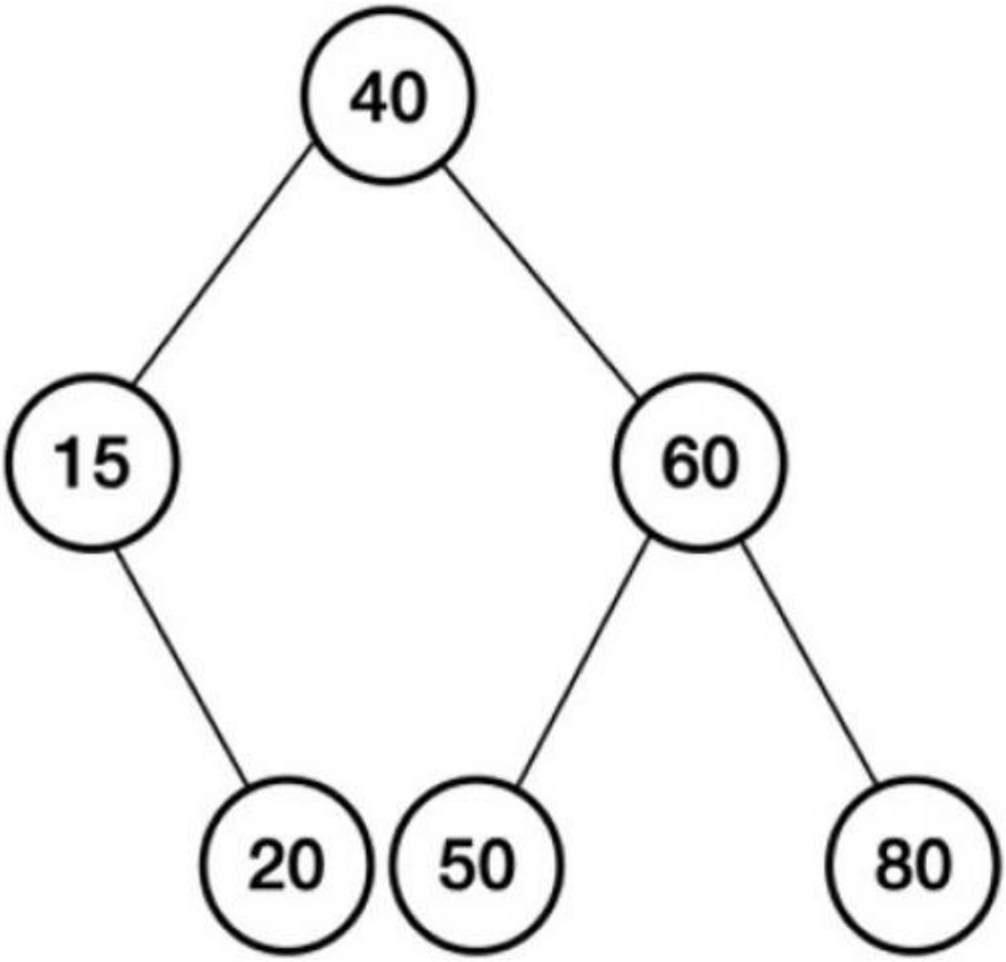


Imagem 8 - Árvore AVL.

Na árvore da imagem 8, se inserirmos uma nova chave, a chave “10”, por exemplo, não haverá a perda da regulagem dos nós da árvore. A árvore resultante continuará a ser uma árvore AVL (imagem 9). Veja:

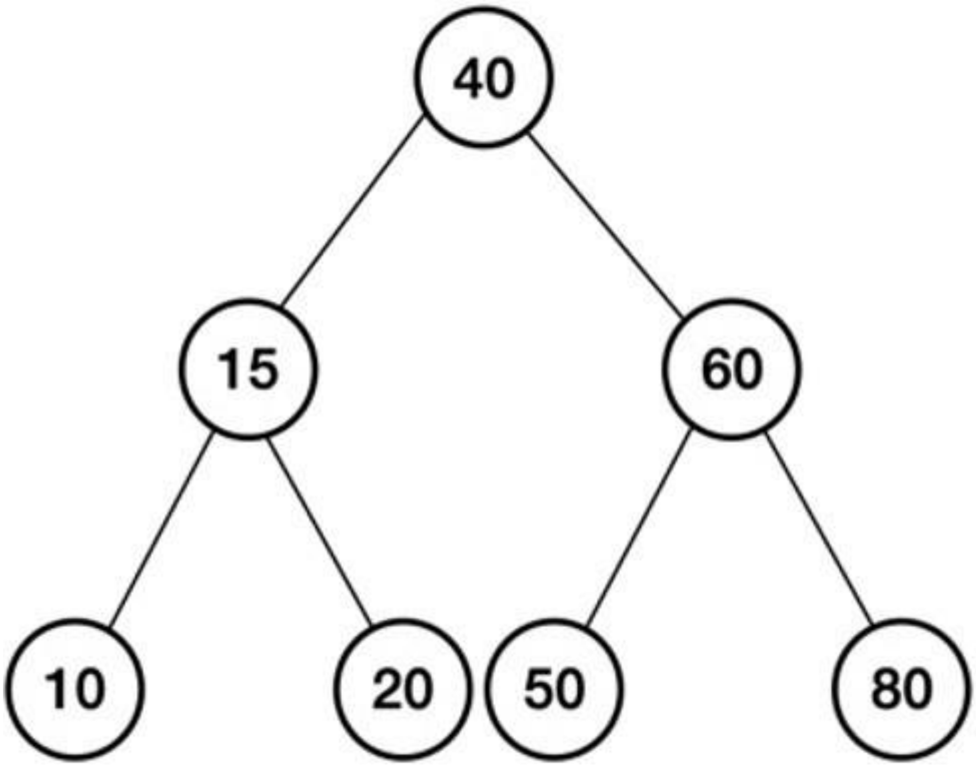


Imagem 9 - Árvore resultante após inserção da chave “10”.

Contudo, se, ao invés de inserirmos a chave “10”, inserirmos a chave “30”, a árvore resultante deixa de ser AVL. Na imagem a seguir, temos a árvore resultante e, nessa árvore, temos que o nó “20” está regulado, porém o nó “15” está desregulado. A altura de sua subárvore esquerda é 0 e sua subárvore direita tem altura 2 (imagem 10).

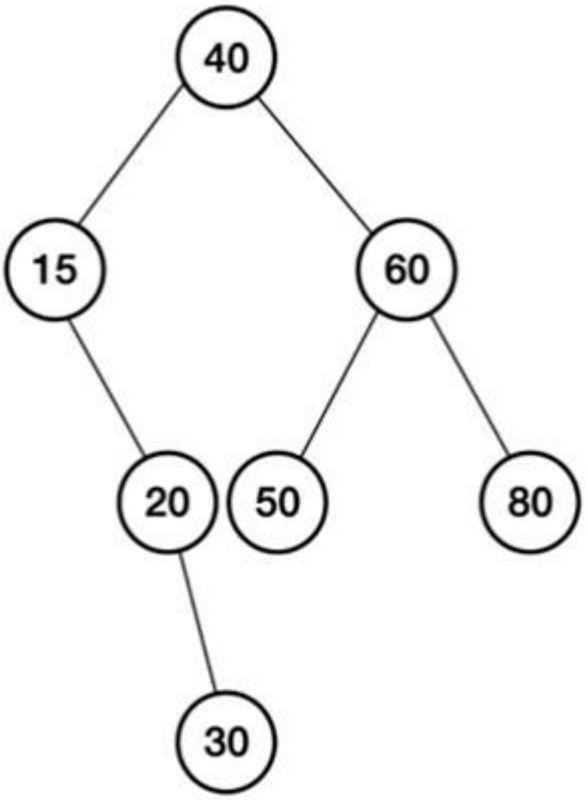


Imagem 10 - Árvore resultante após a inserção da chave “30” – não é AVL.

Observe que, na árvore da imagem 10, o nó “15” é o nó mais profundo desregulado. O nó mais profundo é aquele de maior nível, ou seja, “15” é o nó de maior nível na árvore desregulado. Vamos concentrar a análise nesse nó, seja “u” esse nó. A situação que vamos analisar é a da imagem 11. Veja:

Imagem 11 - Inserção de uma chave **x**>**u** na árvore AVL.

Destacando a raiz de T2, temos as seguintes possíveis situações (imagem 12) antes da inserção da chave **x**.

Observe que o único cenário viável é o da imagem 12 (A). Ao inserir **x** na árvore de A, T2 ou T3 podem crescer e esse crescimento não desregula o nó " **v**". Nas árvores de B e C, a inserção de " **x**" na árvore desregularia " **v** ", o que contraria nossa premissa de que " **u** " é o nó mais profundo desregulado, ou não desregula nó algum, o que também contraria nossa premissa.

Análise semelhante deve ser feita para o caso de inserirmos uma chave **y**<**u**. Nesse caso, pode-se concluir que o único cenário de estudo é o da imagem 13. Chamaremos de caso 1 o correspondente à imagem 12 (A) e caso 2 o da imagem 13.

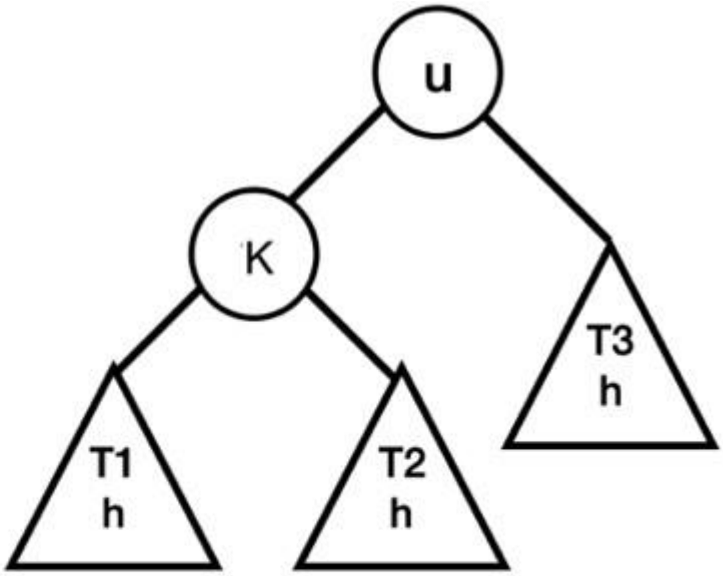


Imagem 13 - Caso 2 - Inserção de uma nova chave menor que **u**.

Voltando à análise do caso 1, veremos que esse caso pode ser dividido em dois subcasos:

- O subcaso 1.a, que corresponde à inserção de uma chave **x**>**u** e **x**>v.
- O subcaso 1.b, que corresponde à inserção de uma chave **x** tal que **u**<**x**<**v**.

Na imagem 14, temos a ilustração do subcaso 1a. Suponhamos que, ao inserir a chave " **x** ", o ramo T3 da árvore cresça. Definindo balanço de **x** como **bal(x)=h(Tdr)−h(Ter)**, temos que o nó " **v** " continua regulado **(bal(v)=1),** conforme nossa hipótese de análise, entretanto " **u** " está desregulado **(bal(u)=2).** Após a inserção, a subárvore esquerda de **u** tem altura h e a subárvore direita tem altura h**+2**.

A operação que reajusta a estrutura é a rotação para a esquerda. Trata-se de transformar o nó " **u** " como raiz da subárvore, fazer " **u** " seu filho esquerdo (**u**<**v**, lembre-se de que se trata de uma árvore binária de busca) e fazer **T1** subárvore esquerda de " **u** " (os nós **k** de T1 satisfazem a propriedade **k**<**u**), T2 subárvore direita de "**u**" (os nós **k** de T2 satisfazem a propriedade **u**<**k**<**v**), e T3 subárvore direita de **v** (os nós **k** de T3 satisfazem a propriedade **k**>**v**).

Atenção!

Após a rotação, a árvore resultante tem mesma altura que a árvore original. Isso garante que, para os ascendentes de u**, se houver, e porventura existirem e tiverem se tornado desregulados após a inserção, a aplicação da rotação iria ajustá-los automaticamente.**

Vejamos essa situação na árvore da imagem 15:

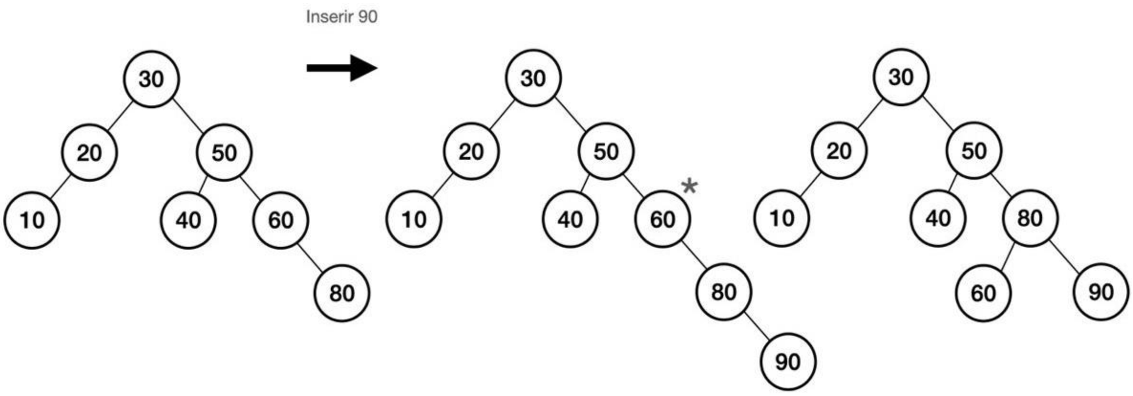


Imagem 15 - Inserção da chave “90”.

Na imagem 15, inserimos a chave “90” na estrutura. O resultado da inserção é que o nó “60” é o nó mais profundo desregulado. Entretanto, os nós “50” e “30” também estão desregulados. Conforme vimos, aplicamos a rotação para a esquerda no nó “60”, que resulta na estrutura com as chaves “60”, “80” e “90” com altura 2. A rotação regula a estrutura e, como a subárvore resultante tem altura 2, assim como o ramo original, os nós “50” e “30” voltam a estar regulados automaticamente.

O caso 2a é análogo ao caso 1a. Após a inserção de uma chave **y**<**k** na árvore da imagem 16 (A), podemos ter como resultado a árvore da imagem 16 (B) e, após a aplicação da rotação para direita (imagem 16(C)), a árvore volta a estar regulada.

Todas as observações feitas para o caso 1a são análogas ao caso 2a, veja:

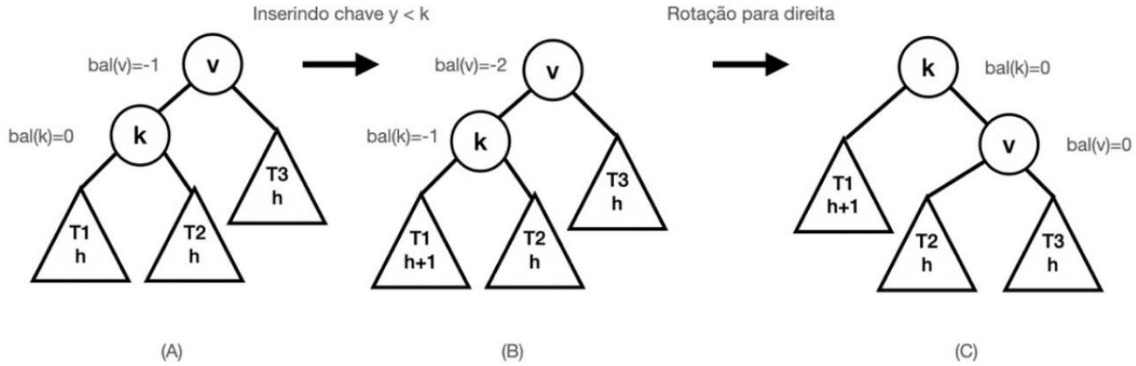


Imagem 16 - Caso 2a, inserção de chave **y**<**k**.

Vejamos agora a análise do caso 1.b, isto é, a inserção de uma chave **x** tal que **u**<**x**<**v**. A imagem 17 ilustra o caso 1.b:

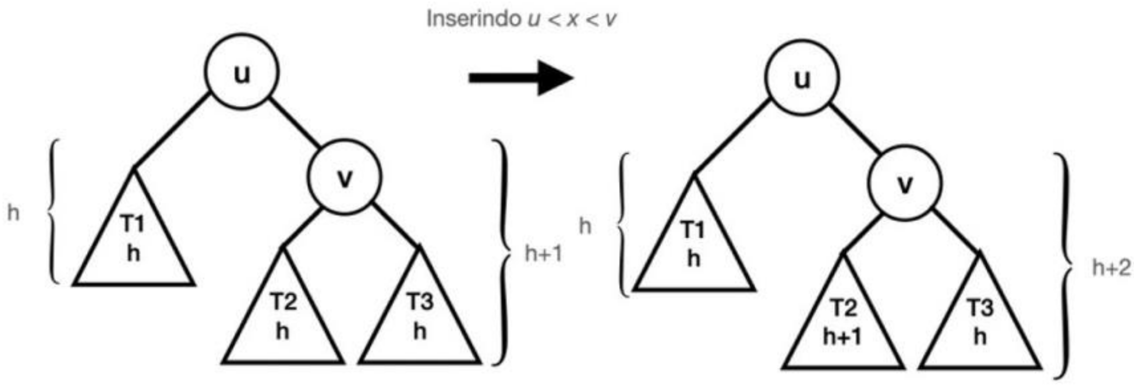


Imagem 17 - Inserindo **u**<**x**<**v**.

Destacando a raiz de T2 antes da inserção da nova chave, podemos ter as configurações da imagem 18. Veja as implicações e possibilidades dessas configurações:

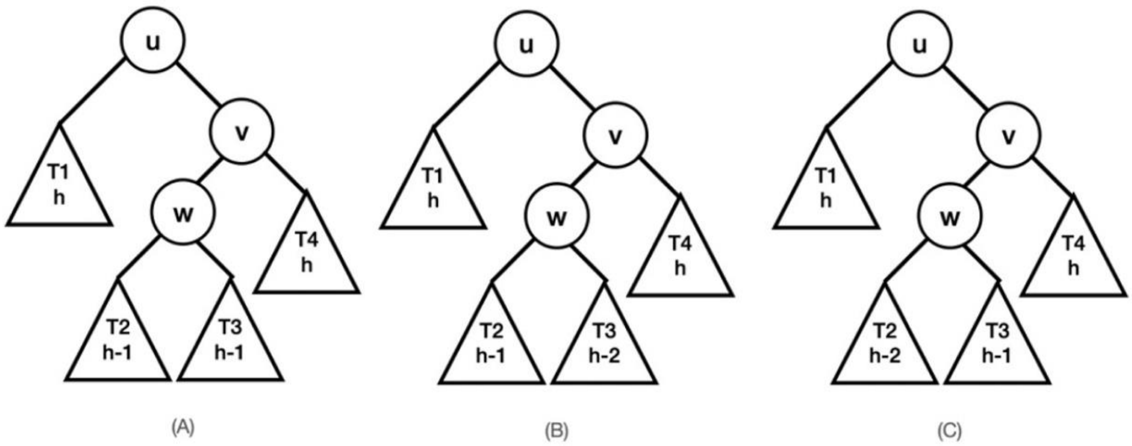


Imagem 18 - Análise do subcaso 1.b

No caso da imagem 18 (A), caso a nova chave seja maior ou menor que **w**, pode implicar no crescimento do ramo direito ou esquerdo da árvore com raiz **w**. No caso da imagem 18 (A), T2 ou T3. Caso essas árvores não cresçam, isto é, mantenham sua altura original, nada precisamos fazer. Contudo, se T2 ou T3 crescer devido à inclusão, teremos **w** regulado, **v** regulado e **u** desregulado, coerente com o que assumimos inicialmente. Então, o cenário da figura 18(A) é um dos focos do nosso estudo.

No caso da imagem 18 (B) e (C), observe que os casos são semelhantes, divergem somente no ramo de maior altura esquerdo (imagem 18 (B)) e direito (imagem 18(C)). Analisando a árvore de B, se inserirmos uma nova chave no ramo direito (T3), poderá ocorrer o crescimento de T3, contudo, se T3 crescer, a árvore de raiz **w** tem balanço zero, não repercutindo nos nós **v** e **u**. Entretanto, se T2 crescer, isto é, passar a ter altura h, o nó **w** passará a estar desregulado, o que fere o que supomos inicialmente, isto é, que **u** é o nó mais profundo desregulado. Assim, o único cenário plausível para análise é o de A.

Vejamos então o que pode ocorrer quando inserimos uma nova chave na subárvore de raiz **w** da imagem 18 (A).

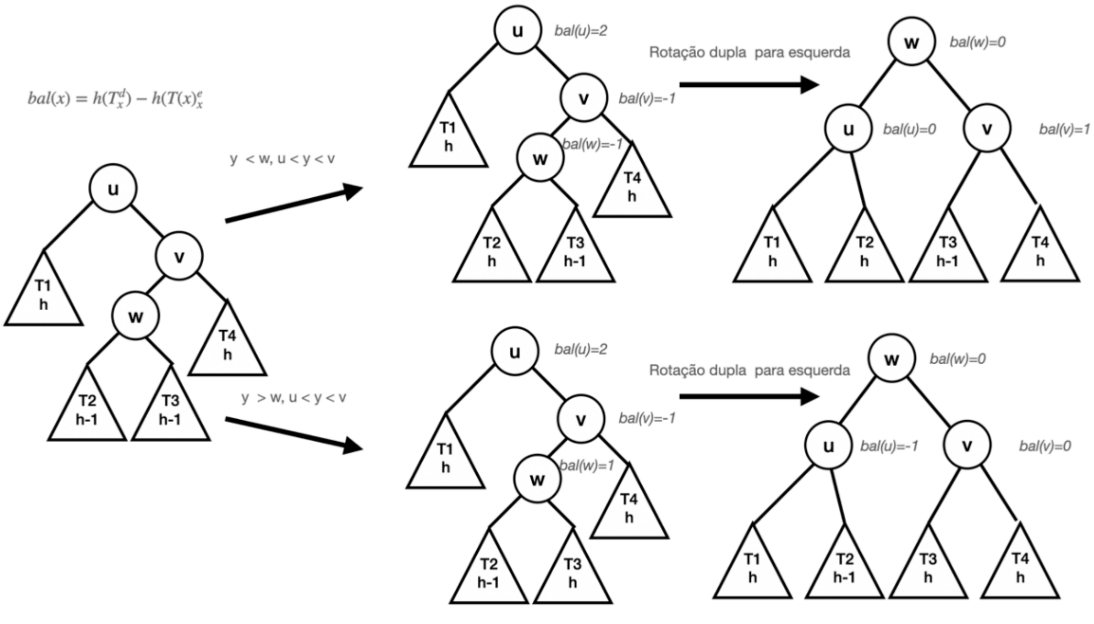


Imagem 19 - Rotação dupla para a esquerda.

A imagem 19 ilustra a situação, observe que **u** é o nó desregulado. A rotação dupla para a esquerda resolve a regulagem de **u**. Nessa rotação, **w** é posto como raiz da subárvore. Como **u** é menor que **w**,**u** é inserido como filho esquerdo de **w** e **v** como filho direito de **w**(**v** > **w**). Observe que:

- Os nós de T1 são menores que **u**.
- Os nós **z** de T2 obedecem à propriedade **u**<**z**<**w**.
- Os nós **z** de T3 obedecem à propriedade **w**<**z**<**v**.
- Os nós **z** de T4 obedecem **z**>**v**.

Vejamos um exemplo:

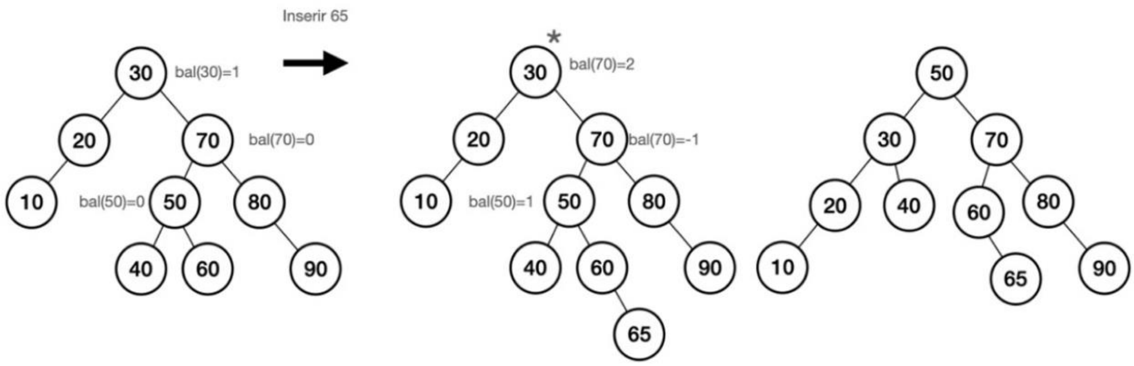


Imagem 20 - Inserção da chave 65.

Observe que, na árvore do exemplo, após a inserção da chave 65, os nós 50 e 70 permanecem regulados e 30 desregulado. Após a inserção dupla para a esquerda, todos os nós ficam regulados.

Após a aplicação da rotação dupla para a esquerda, a árvore resultante tem a mesma altura que a árvore original, antes da inserção da nova chave. Essa propriedade garante que, regulando o nó **u**, todos os ancestrais de **u** ficam automaticamente regulados, como ocorreu no Caso 1a.

A análise do caso 2b é análoga à do caso 1b. A imagem 21 mostra como identificar e aplicar a rotação dupla para a direita.

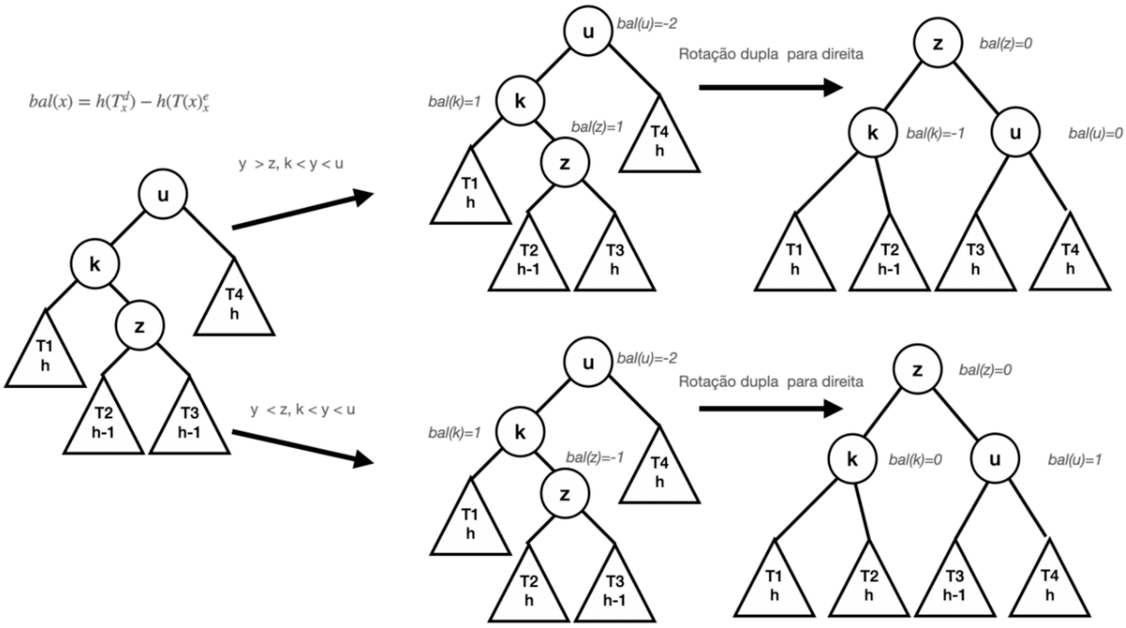


Imagem 21 - Caso 2b - rotação dupla para direita

Para concluir a análise do algoritmo de inclusão de nós na árvore AVL, resta determinar como é feita a identificação das rotações pelo algoritmo. Observe que, na análise, nos referimos à diferença de altura entre o ramo direito e esquerdo. Essa diferença foi chamada de balanço do nó. Contudo, se calcularmos a altura das subárvores esquerda e direita de um nó para determinar o balanço, não será possível manter a complexidade de **O**(log **n**) da operação de inserção, uma vez que, para calcular a altura de uma árvore, é necessário executar um algoritmo com complexidade **O**(**n**).

A solução para esse problema é armazenar o balanço do nó junto com a chave. Conforme definido, **bal(x)=h(Tdx)−h(Tex)**. O crescimento do módulo do balanço de uma chave indica que o ramo cresceu, sendo necessário atualizar o balanço incrementando-o, caso o ramo que tenha crescido seja o direito ou decrementando-o, caso o ramo que tenha crescido seja o esquerdo.

A identificação do caso ocorre quando um nó tem balanço 2 indicando o caso 1 ou balanço -2 indicando o caso 2. O subcaso é identificado pelo balanço do filho direito:

- No caso 1, se o balanço do filho direito for 1, temos o subcaso 1a.
- Se o balanço do filho direito for -1, temos o subcaso 1b.

Analogamente, o balanço do filho esquerdo identifica o subcaso do caso 2.

- Caso o balanço do filho esquerdo seja -1, temos o caso 2a.
- Caso seja 1, temos o subcaso 2b.

O pseudocódigo do algoritmo será omitido, entretanto, pode ser facilmente encontrado na internet.

Remoção em árvores AVL

A análise da remoção é muito semelhante à da inserção de novos nós na árvore. Toda a análise da inserção baseia-se no estudo do crescimento de um ramo da árvore devido ao novo nó contendo a nova chave.

Além disso, a árvore AVL é uma árvore binária de busca, a remoção recai nos três subcasos estudados. Como exemplo, vejamos a remoção de um nó de T3 que resulta no encurtamento de T3 (imagem 22).

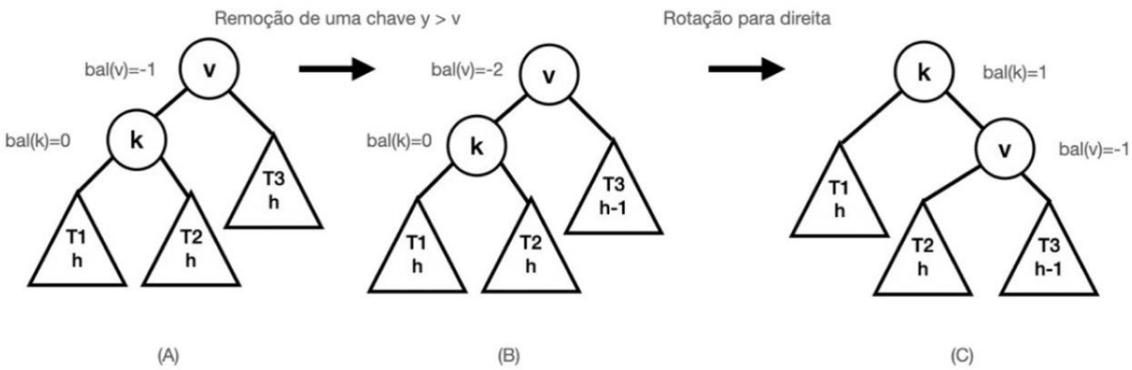


Imagem 22 - Remoção de uma chave **Y**>**V**.

O ajuste aplicado na estrutura foi o do caso 2a. Porém, observe que há uma sutil diferença em relação ao balanço de **k**, filho esquerdo de **V**. No exemplo, **k** tem balanço 0 e resultou na rotação para a direita. Na inclusão, a rotação para a direita ocorria quando o balanço de **K** era igual a -1.

Sendo assim, a aplicação da rotação para a direita ocorre quando **bal(v)=−2** e o balanço do filho esquerdo de **v**, no caso do exemplo **K**, é 0 ou −2, isto é, **bal(k)=0 ou bal(k)= −1** (imagem 23).

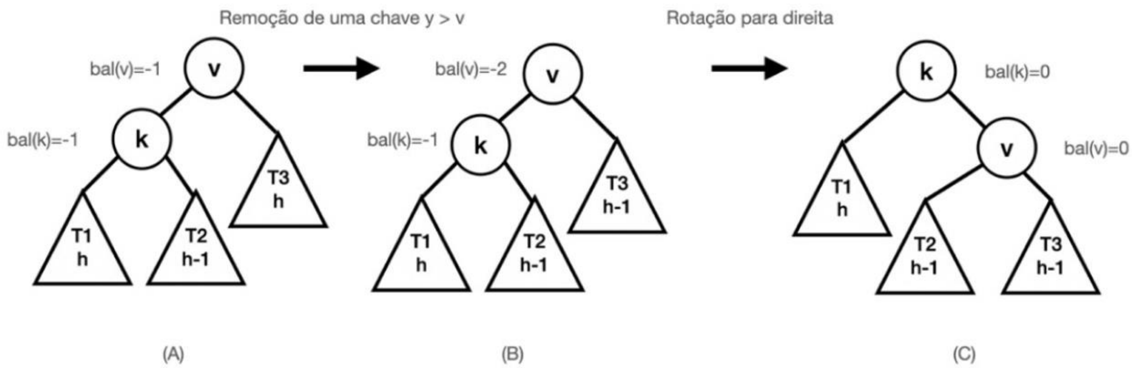


Imagem 23 - Aplicação da rotação do Caso 2a, possibilidade 2.

A imagem 23 mostra que, na aplicação da rotação na remoção, existe a possibilidade da estrutura resultante ter seu tamanho reduzido de uma unidade. Nesse caso, caso haja ancestrais de **v** na árvore original, isto é, antes da remoção de **y**, o efeito da remoção pode propagar-se para os ancestrais de **v**.

Essa situação ocorre, por exemplo, na árvore de Fibonacci. A imagem 24 mostra um exemplo:

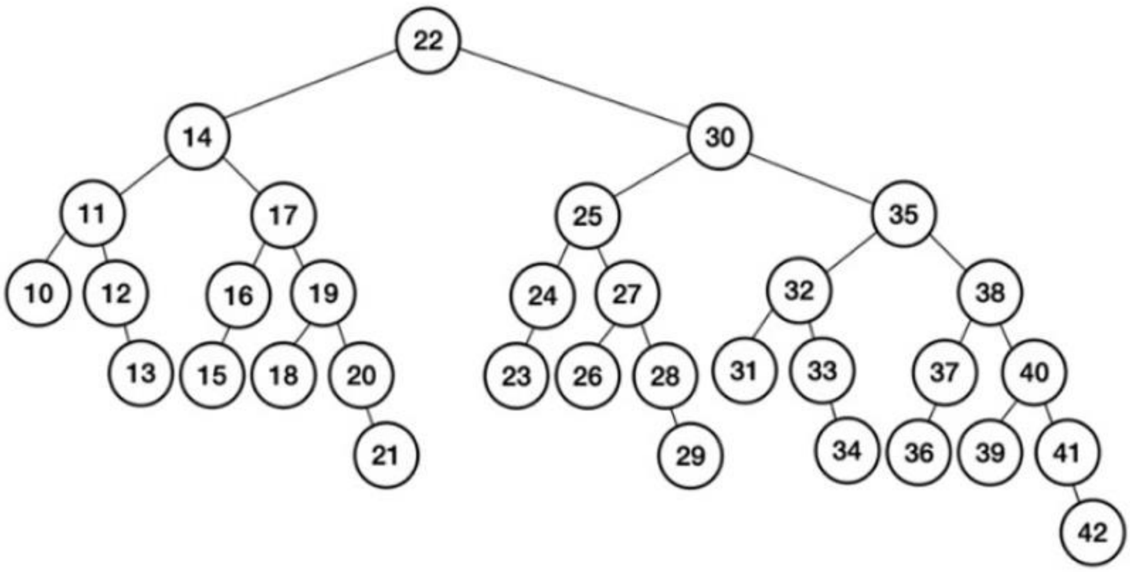


Imagem 24 - Árvore de Fibonacci.

Ao removermos o nó 10, temos que o **bal(11)=2 e bal(12)=1**, que corresponde ao caso 1a. Aplicando a rotação para a esquerda, temos a configuração da imagem 25:

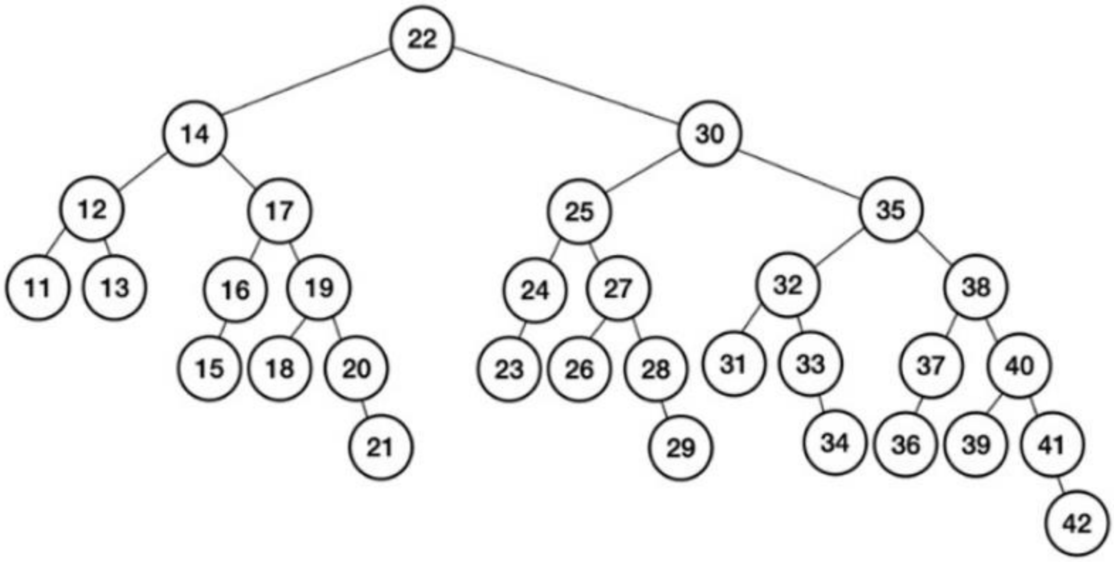


Imagem 25 - Resultado da aplicação da rotação para a esquerda no nó 11.

Observe que, após a rotação, o nó “14” tem balanço 2 e o nó “17” tem balanço 1, também o caso 1a. Aplicando a rotação, temos a árvore da imagem 26.

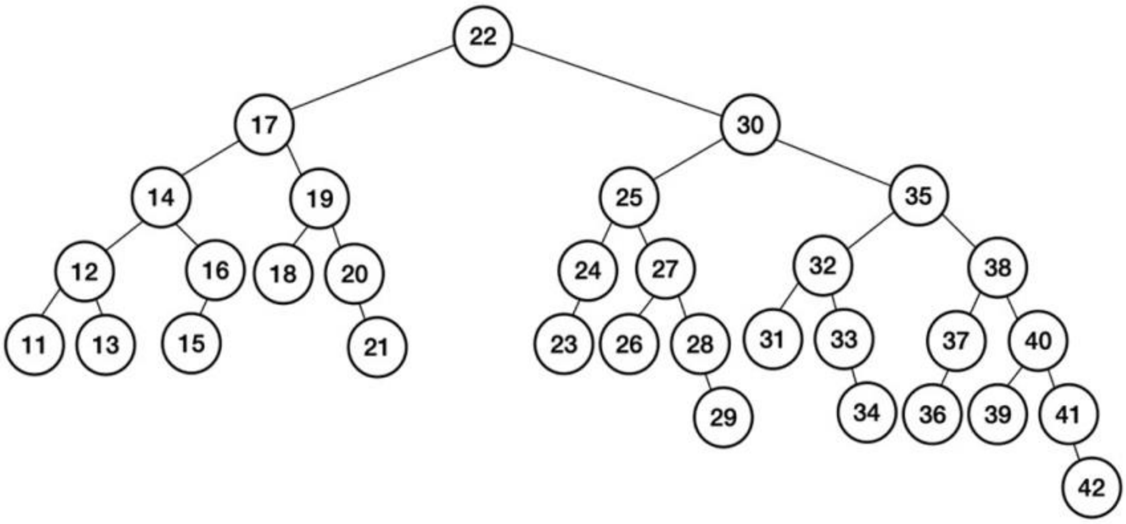


Imagem 26 - Resultado da aplicação da rotação para a esquerda no nó “14”.

Após a rotação, o nó “22” tem balanço 2 e o nó “30” tem balanço 1, também caso 1a. Aplicando a rotação para a esquerda, no nó “22” temos a árvore da imagem 27:

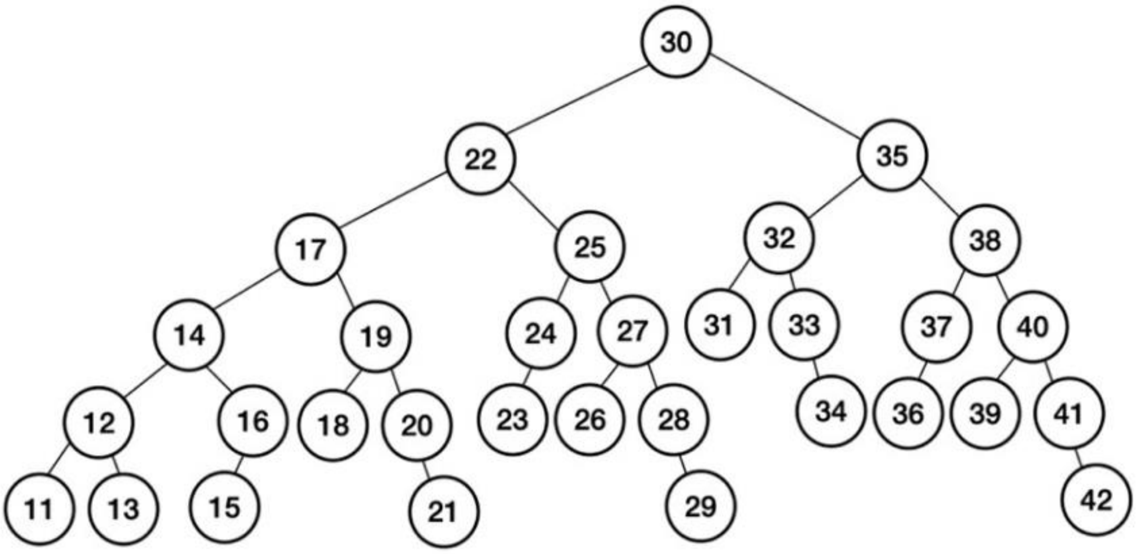


Imagem 27 - Resultado da aplicação da rotação para a esquerda no nó “22”.

Observe que as rotações em sequência reestruturam a propriedade AVL após a remoção. A árvore de Fibonacci é o pior caso da remoção. Conforme visto, nas árvores de Fibonacci é necessário realizar todas as rotações do pai do nó removido; no caso do exemplo da imagem anterior, o nó “11” até a raiz.

Cada rotação tem complexidade da **O(1),** assim, a aplicação da rotação no caminho da folha removida até a raiz não tem impacto na complexidade da remoção que é **O**(log **n**).

Concluindo, o algoritmo de remoção recai nos subcasos estudados e, após a aplicação desses subcasos, o balanço do nó pai do nó removido deve ser atualizado e aplicadas as rotações conforme os casos estudados na inserção caso 1, subcaso 1a e 1b ou caso 2, subcaso 2a e 2b.

Na remoção, os subcasos 1a e 2a ocorrem quando bal(v) =2 e do filho direito é 1 ou 0 ou bal(v) = -2 ou o balanço do filho esquerdo é -1 ou 0, respectivamente. Os subcasos 1b e 2b são identificados da mesma forma que na inclusão.

Estudo da complexidade da árvore AVL

Vimos que as árvores AVL são balanceadas, isto é, toda árvore AVL tem altura proporcional a log **n**. Desse fato ocorre que a complexidade da busca é **O**(log **n**). Foi visto que a complexidade da busca é proporcional à altura da árvore, no caso da árvore log **n**, e que toda árvore AVL é uma árvore binária de busca. Sendo assim, aplica-se o mesmo algoritmo estudado.

A inclusão de uma nova chave na árvore AVL também tem complexidade **O**(log **n**), isso decorre do fato de que a inclusão é consequência direta da busca e a busca executa em **O**(log **n**). No pior caso, a inclusão de uma nova chave pode resultar em uma rotação, que é uma operação simples, que executa em **O**(1). Não é impactante na complexidade, mas vimos que, após a aplicação da rotação na inclusão, o ramo alterado pela inclusão preserva sua altura original, isto é, antes da inserção da nova chave, fazendo com que novas rotações nos ancestrais sejam desnecessárias.

Finalmente, o pior caso da remoção é a remoção da folha menos profunda de uma árvore de Fibonacci. Nesse caso, realizamos todas as rotações da folha removida até a raiz. Contudo, cada rotação tem complexidade da **O**(1) e são realizadas **O**(log **n**) rotações até regular toda a árvore. Assim, as árvores AVL fornecem algoritmos totalmente dinâmicos capazes de realizar busca, inserção e remoção em **O**(log **n**).