Árvores AVL

Letícia Rodrigues Bueno

2 de agosto de 2013

Relembrando: árvores binárias de busca

- Objetivo: minimizar tempo de acesso no pior caso.

Relembrando: árvores binárias de busca

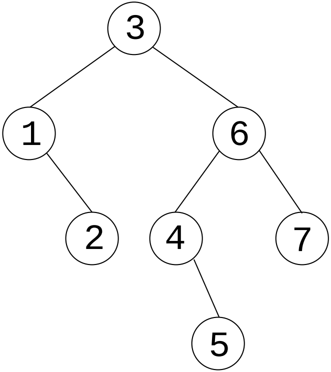
- Objetivo: minimizar tempo de acesso no pior caso.
- Idéia: Para cada chave, separe as restantes em *maiores* ou *menores*.

Relembrando: árvores binárias de busca

- Objetivo: minimizar tempo de acesso no pior caso.
- Idéia: Para cada chave, separe as restantes em *maiores* ou *menores*.
- Estrutura hierárquica com divisão binária: uma *árvore binária*.

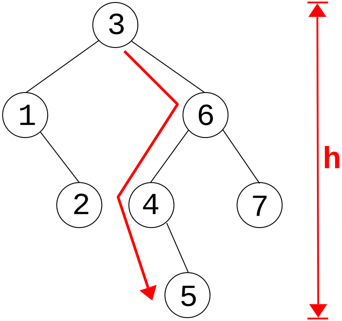
Relembrando: árvores binárias de busca

- Objetivo: minimizar tempo de acesso no pior caso.
- Idéia: Para cada chave, separe as restantes em *maiores* ou *menores*.
- Estrutura hierárquica com divisão binária: uma *árvore binária*.



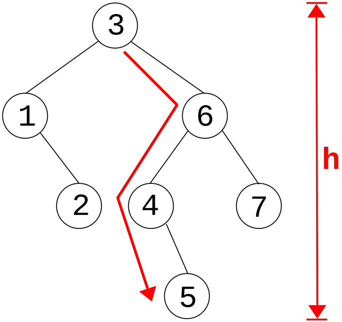
Relembrando: complexidade da busca em árvore binária

- Busca em árvore binária = caminho da raiz até chave desejada (ou até uma folha, caso chave não exista).



Relembrando: complexidade da busca em árvore binária

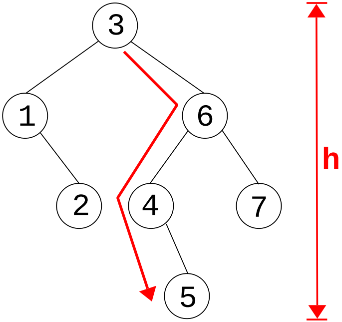
- Busca em árvore binária = caminho da raiz até chave desejada (ou até uma folha, caso chave não exista).



- Pior caso: maior caminho da raiz até folha

Relembrando: complexidade da busca em árvore binária

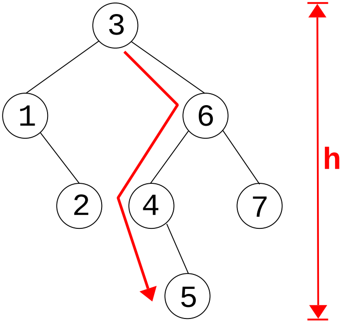
- Busca em árvore binária = caminho da raiz até chave desejada (ou até uma folha, caso chave não exista).



- Pior caso: maior caminho da raiz até folha = altura da árvore

Relembrando: complexidade da busca em árvore binária

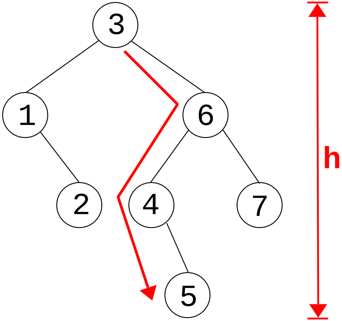
- Busca em árvore binária = caminho da raiz até chave desejada (ou até uma folha, caso chave não exista).



- Pior caso: maior caminho da raiz até folha = altura da árvore
- Complexidade pior caso: $O(h)$

Relembrando: complexidade da busca em árvore binária

- Busca em árvore binária = caminho da raiz até chave desejada (ou até uma folha, caso chave não exista).



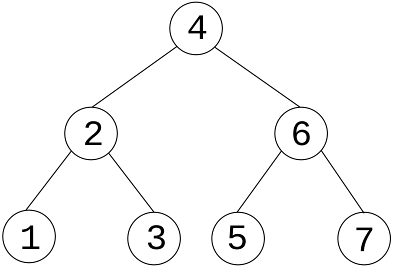
- Pior caso: maior caminho da raiz até folha = altura da árvore
- Complexidade pior caso: $O(h)$ (como otimizar pior caso?)

Relembrando: árvore binária de busca ótima

- Árvore ótima: minimiza tempo de busca (no pior caso)

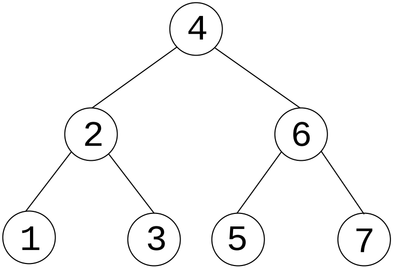
Relembrando: árvore binária de busca ótima

- Árvore ótima: minimiza tempo de busca (no pior caso)
- Árvore completa, altura: $h = \lfloor \log n \rfloor + 1$



Relembrando: árvore binária de busca ótima

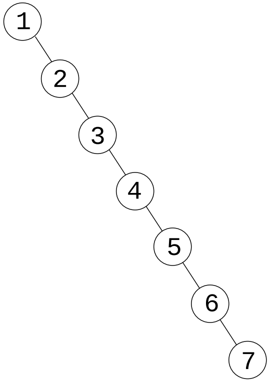
- Árvore ótima: minimiza tempo de busca (no pior caso)
- Árvore completa, altura: $h = \lfloor \log n \rfloor + 1$



- complexidade temporal no pior caso: $O(\log n)$

Relembrando: construção de árvore ótima

- após inserções, árvore binária de busca pode degenerar em uma lista



- tempo de busca pior caso: $O(n)$

Relembrando: construção de árvore ótima

- Estrutura fixa: chaves pré-determinadas.
- Dado um conjunto com n chaves, é possível construir a árvore ótima em tempo $O(n^3)$ (ou $O(n^2)$ se usarmos um algoritmo mais elaborado).
- Para manter a árvore ótima, deveríamos executar o algoritmo a cada inserção: impraticável!
- Podemos manter complexidade de pior caso da inserção em $O(\log n)$?

Árvores "suficientemente boas"

- Árvore com altura $2 \log n$, complexidade temporal no pior caso $O(\dots)$

Árvores "suficientemente boas"

- Árvore com altura $2 \log n$, complexidade temporal no pior caso $O(\log n)$

Árvores "suficientemente boas"

- Árvore com altura $2 \log n$, complexidade temporal no pior caso $O(\log n)$
- Árvore com altura $c_1 \log n + c_2$, complexidade temporal no pior caso $O(\ldots)$

Árvores "suficientemente boas"

- Árvore com altura $2 \log n$, complexidade temporal no pior caso $O(\log n)$
- Árvore com altura $c_1 \log n + c_2$, complexidade temporal no pior caso $O(\log n)$

Árvores "suficientemente boas"

- Árvore com altura $2 \log n$, complexidade temporal no pior caso $O(\log n)$
- Árvore com altura $c_1 \log n + c_2$, complexidade temporal no pior caso $O(\log n)$
- Nem toda árvore com altura $O(\log n)$ é ótima, mas a complexidade assintótica temporal de pior caso para a busca é igual à de uma árvore ótima.

Árvores "suficientemente boas"

- Árvore com altura $2 \log n$, complexidade temporal no pior caso $O(\log n)$
- Árvore com altura $c_1 \log n + c_2$, complexidade temporal no pior caso $O(\log n)$
- Nem toda árvore com altura $O(\log n)$ é ótima, mas a complexidade assintótica temporal de pior caso para a busca é igual à de uma árvore ótima.

Definição: *Árvore binária balanceada* é aquela com altura $O(\log n)$

Árvores "suficientemente boas"

- Árvore com altura $2 \log n$, complexidade temporal no pior caso $O(\log n)$
- Árvore com altura $c_1 \log n + c_2$, complexidade temporal no pior caso $O(\log n)$
- Nem toda árvore com altura $O(\log n)$ é ótima, mas a complexidade assintótica temporal de pior caso para a busca é igual à de uma árvore ótima.

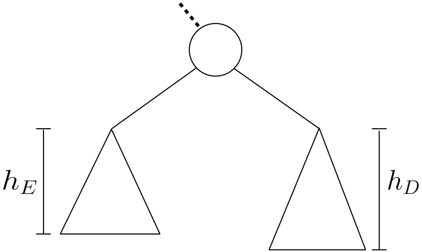
Definição: *Árvore binária balanceada* é aquela com altura $O(\log n)$

Mais fácil de construir que árvore ótima?
Como garantir que uma árvore binária é balanceada?

Árvores AVL

Para cada nó x, defina:

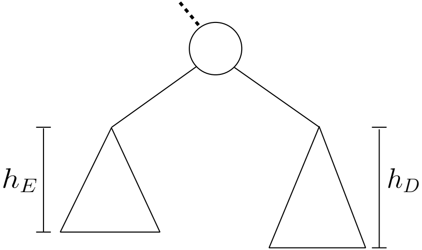
- $h_E(x)$: altura sub-árvore à esquerda
- $h_D(x)$: altura sub-árvore à direita



Árvores AVL

Para cada nó x, defina:

- $h_E(x)$: altura sub-árvore à esquerda
- $h_D(x)$: altura sub-árvore à direita

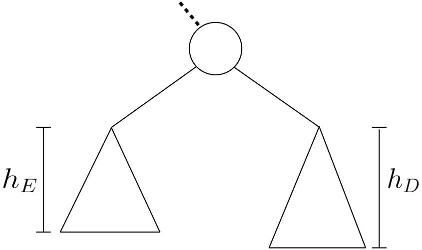


- **Propriedade AVL**: $|h_E(x) - h_D(x)| \leq 1$

Árvores AVL

Para cada nó x, defina:

- $h_E(x)$: altura sub-árvore à esquerda
- $h_D(x)$: altura sub-árvore à direita

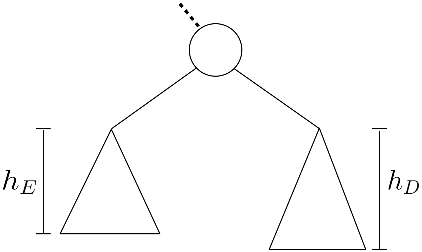


- **Propriedade AVL**: $|h_E(x) - h_D(x)| \leq 1$
- Nó **regulado**: satisfaz propriedade AVL.

Árvores AVL

Para cada nó x, defina:

- $h_E(x)$: altura sub-árvore à esquerda
- $h_D(x)$: altura sub-árvore à direita

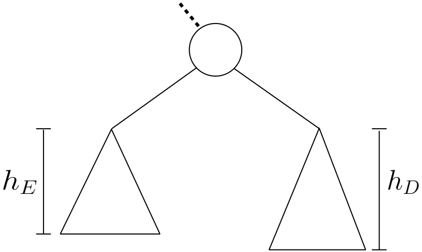


- **Propriedade AVL**: $|h_E(x) - h_D(x)| \leq 1$
- Nó **regulado**: satisfaz propriedade AVL.
- **Árvore AVL**: todos nós regulados.

Árvores AVL

Para cada nó x, defina:

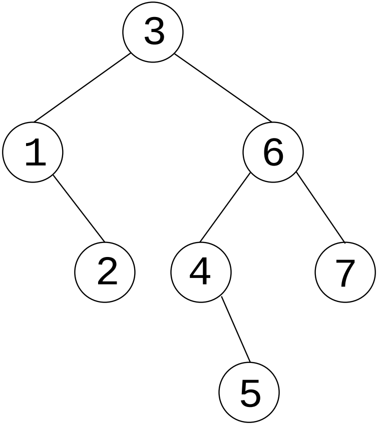
- $h_E(x)$: altura sub-árvore à esquerda
- $h_D(x)$: altura sub-árvore à direita



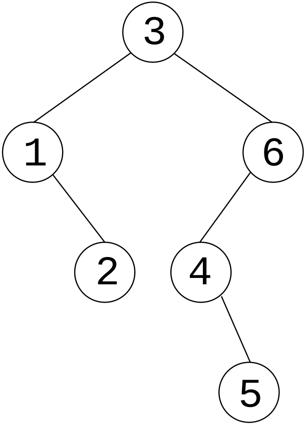
- **Propriedade AVL**: $|h_E(x) - h_D(x)| \leq 1$
- Nó **regulado**: satisfaz propriedade AVL.
- **Árvore AVL**: todos nós regulados.

(Curiosidade: AVL = **A**delson-**V**elskii, G. e **L**andis, E. M.)

Árvores AVL: exemplo



AVL



NÃO AVL

Árvores AVL: inserção

Queremos inserir nova chave:

- mantendo a regulagem de todos os nós
- em tempo razoável

Nossa estratégia:

Árvores AVL: inserção

Queremos inserir nova chave:

- mantendo a regulagem de todos os nós
- em tempo razoável

Nossa estratégia:

1. Inserção como árvore binária comum.

Árvores AVL: inserção

Queremos inserir nova chave:

- mantendo a regulagem de todos os nós
- em tempo razoável

Nossa estratégia:

1. Inserção como árvore binária comum.
2. verificar se existem nós desregulados.

Árvores AVL: inserção

Queremos inserir nova chave:

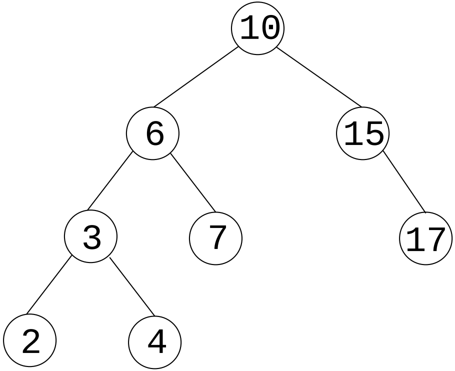
- mantendo a regulagem de todos os nós
- em tempo razoável

Nossa estratégia:

1. Inserção como árvore binária comum.
2. verificar se existem nós desregulados.
3. se existem, tornar os nós regulados.

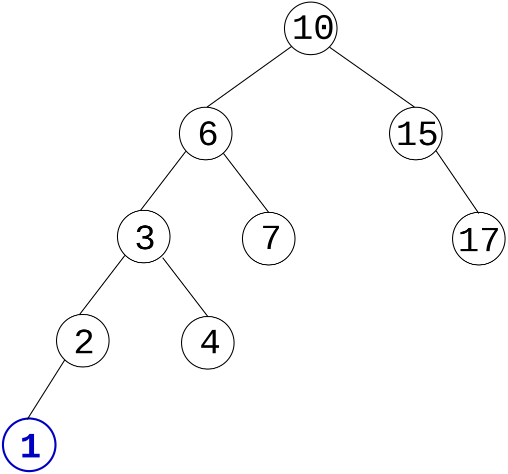
Árvores AVL: exemplo de inserção

Inserir chave com valor 1 na árvore abaixo.



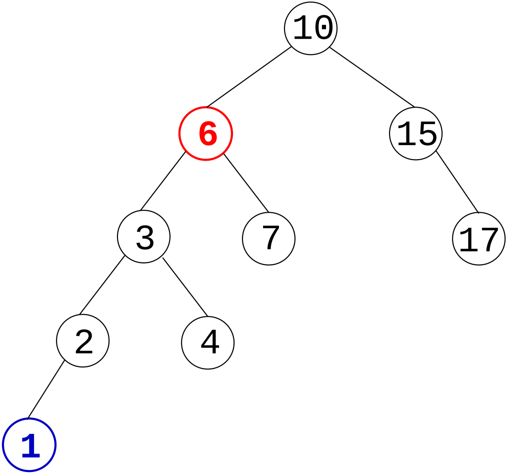
Árvores AVL: exemplo de inserção

Inserir chave com valor 1 na árvore abaixo.



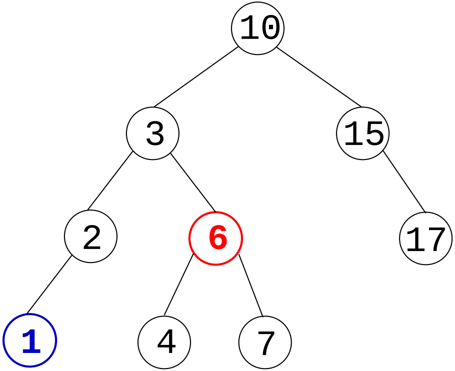
Árvores AVL: exemplo de inserção

Inserir chave com valor 1 na árvore abaixo.



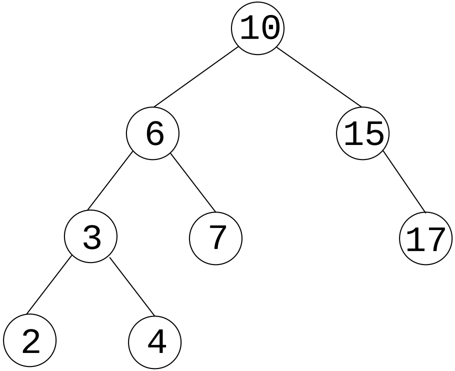
Árvores AVL: exemplo de inserção

Inserir chave com valor 1 na árvore abaixo.



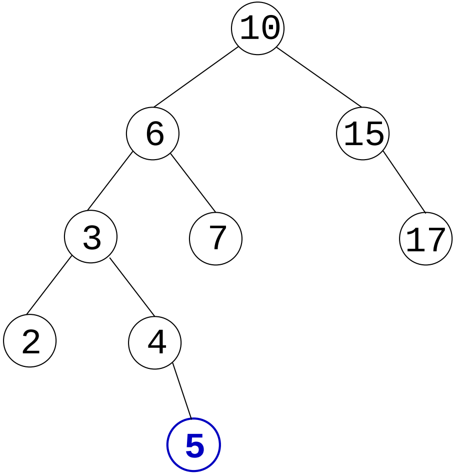
Árvores AVL: exemplo de inserção

Inserir chave com valor 5 na árvore abaixo.



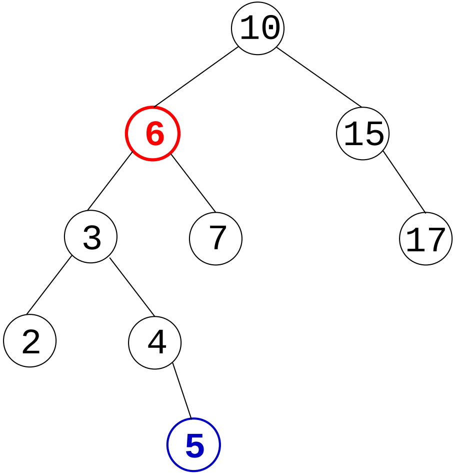
Árvores AVL: exemplo de inserção

Inserir chave com valor 5 na árvore abaixo.



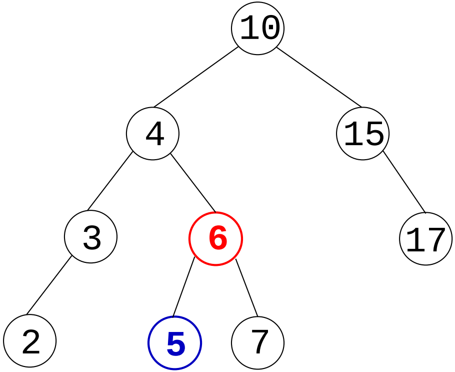
Árvores AVL: exemplo de inserção

Inserir chave com valor 5 na árvore abaixo.



Árvores AVL: exemplo de inserção

Inserir chave com valor 5 na árvore abaixo.



Árvores AVL: regulagem dos nós

Generalizando os casos anteriores.

Árvores AVL: regulagem dos nós

Generalizando os casos anteriores.
Nó inserido: q

- somente sub-árvores que contém q aumentaram altura

Árvores AVL: regulagem dos nós

Generalizando os casos anteriores.
Nó inserido: q

- somente sub-árvores que contém q aumentaram altura
- os nós que ficaram desregulados são todos ancestrais de q

Árvores AVL: regulagem dos nós

Generalizando os casos anteriores.
Nó inserido: q

- somente sub-árvores que contém q aumentaram altura
- os nós que ficaram desregulados são todos ancestrais de q

Seja p o ancestral de q desregulado mais próximo de q.

- q foi inserido na sub-árvore esquerda de p

Árvores AVL: regulagem dos nós

Generalizando os casos anteriores.
Nó inserido: q

- somente sub-árvores que contém q aumentaram altura
- os nós que ficaram desregulados são todos ancestrais de q

Seja p o ancestral de q desregulado mais próximo de q.

- q foi inserido na sub-árvore esquerda de p
- $h_E(p) > h_D(p)$

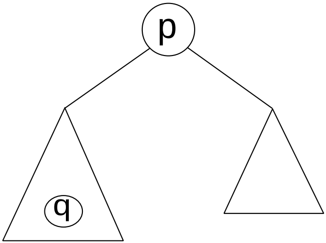
Árvores AVL: regulagem dos nós

Generalizando os casos anteriores.
Nó inserido: q

- somente sub-árvores que contém q aumentaram altura
- os nós que ficaram desregulados são todos ancestrais de q

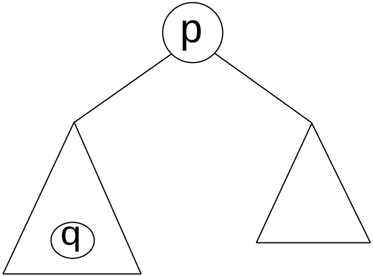
Seja p o ancestral de q desregulado mais próximo de q.

- q foi inserido na sub-árvore esquerda de p
- $h_E(p) > h_D(p)$



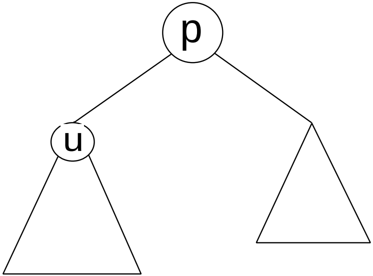
Árvores AVL: regulagem dos nós

- $h_E(p) > h_D(p)$
- .



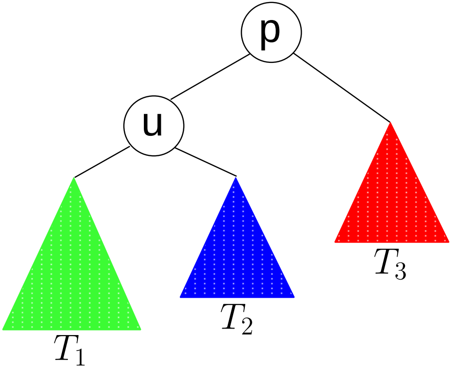
Árvores AVL: regulagem dos nós

- $h_E(p) > h_D(p)$
- .



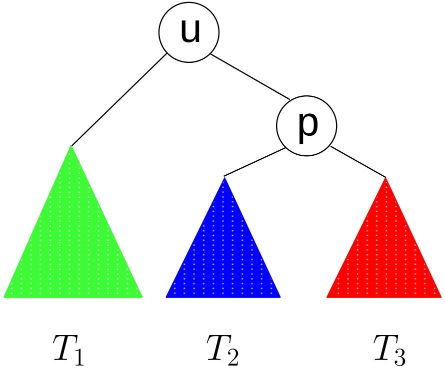
Árvores AVL: regulagem dos nós

- $h_E(p) > h_D(p)$
- Se $h_E(u) > h_D(u)$



Árvores AVL: regulagem dos nós

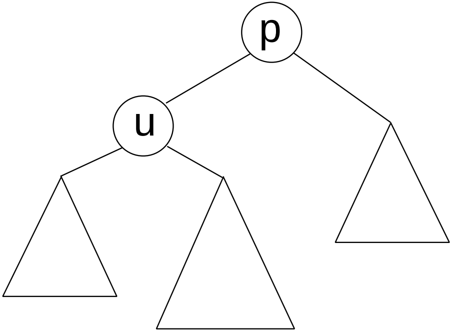
- $h_E(p) > h_D(p)$
- Se $h_E(u) > h_D(u)$



Rotação à direita

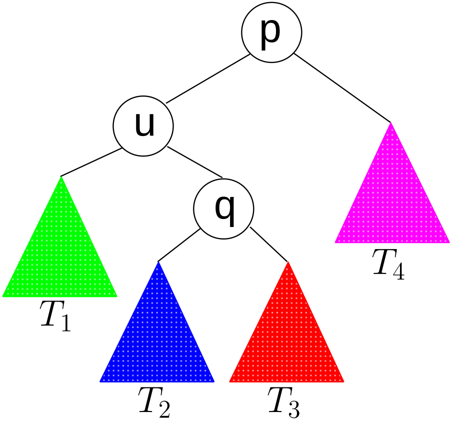
Árvores AVL: regulagem dos nós

- $h_E(p) > h_D(p)$
- Se $h_E(u) < h_D(u)$



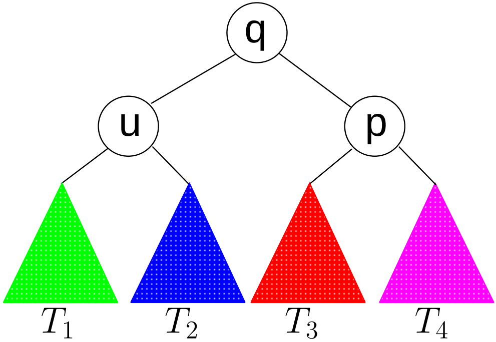
Árvores AVL: regulagem dos nós

- $h_E(p) > h_D(p)$
- Se $h_E(u) < h_D(u)$



Árvores AVL: regulagem dos nós

- $h_E(p) > h_D(p)$
- Se $h_E(u) < h_D(u)$



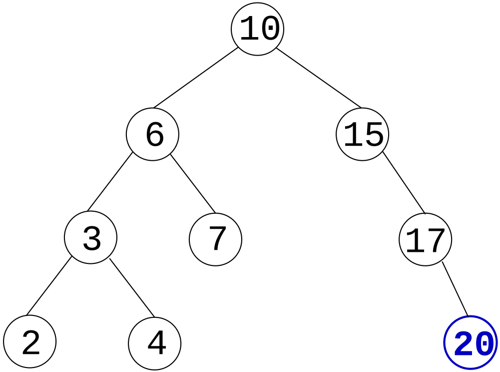
Rotação dupla à direita

Árvores AVL: exemplo de inserção

E se o nó inserido estiver à direita do nó desregulado?

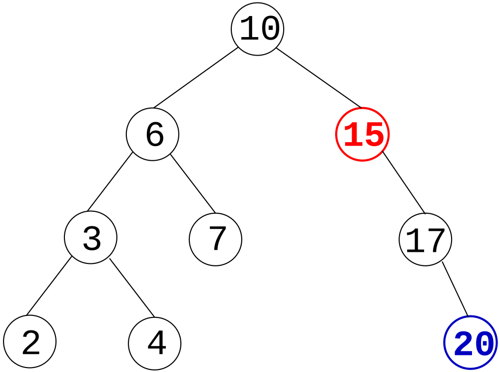
Árvores AVL: exemplo de inserção

E se o nó inserido estiver à direita do nó desregulado?
Inserir 20 na árvore abaixo:



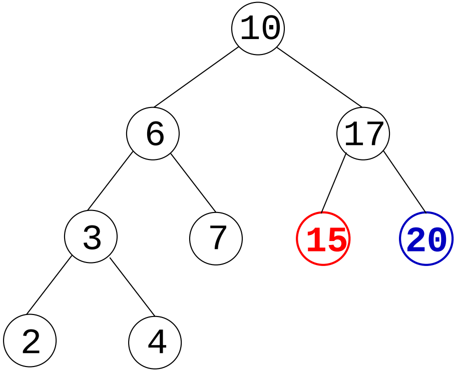
Árvores AVL: exemplo de inserção

E se o nó inserido estiver à direita do nó desregulado?
Inserir 20 na árvore abaixo:



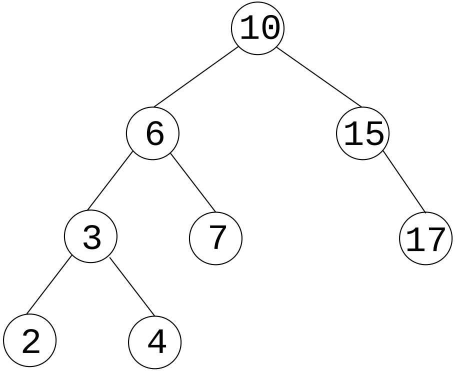
Árvores AVL: exemplo de inserção

E se o nó inserido estiver à direita do nó desregulado?
Inserir 20 na árvore abaixo:



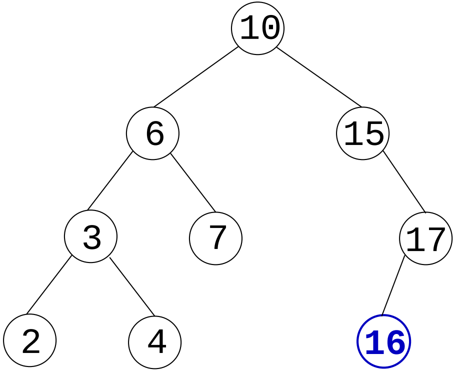
Árvores AVL: exemplo de inserção

Inserir 16 na árvore abaixo:



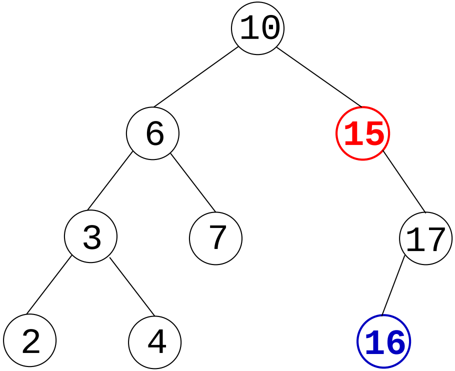
Árvores AVL: exemplo de inserção

Inserir 16 na árvore abaixo:



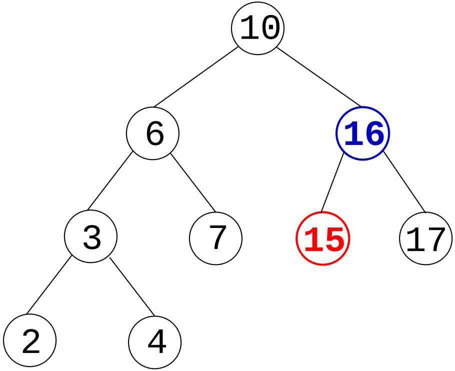
Árvores AVL: exemplo de inserção

Inserir 16 na árvore abaixo:



Árvores AVL: exemplo de inserção

Inserir 16 na árvore abaixo:



Árvores AVL: regulagem de nós

Generalizando os dois casos anteriores.
Nó inserido q.
Seja p o ancestral de q desregulado mais próximo de q.

- q foi inserido na sub-árvore direita de p

Árvores AVL: regulagem de nós

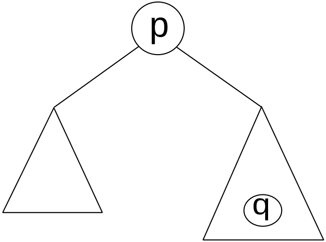
Generalizando os dois casos anteriores.
Nó inserido q.
Seja p o ancestral de q desregulado mais próximo de q.

- q foi inserido na sub-árvore direita de p
- $h_D(p) > h_E(p)$

Árvores AVL: regulagem de nós

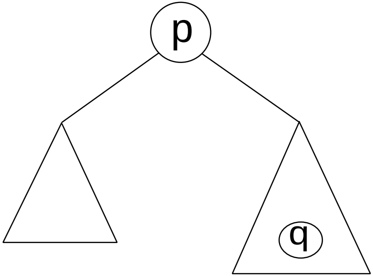
Generalizando os dois casos anteriores.
Nó inserido q.
Seja p o ancestral de q desregulado mais próximo de q.

- q foi inserido na sub-árvore direita de p
- $h_D(p) > h_E(p)$



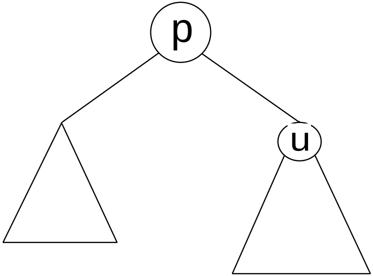
Árvores AVL: regulagem dos nós

- $h_D(p) > h_E(p)$
- .



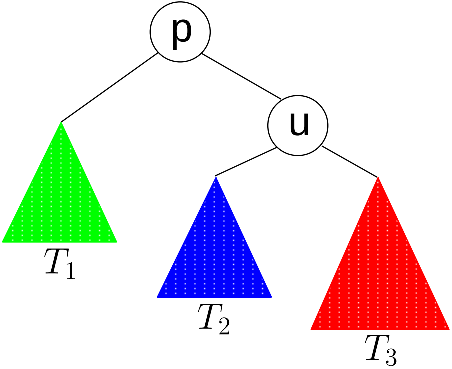
Árvores AVL: regulagem dos nós

- $h_D(p) > h_E(p)$
- .



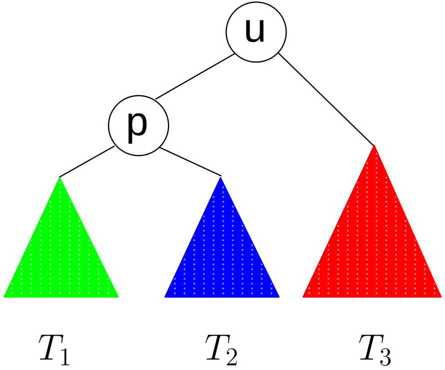
Árvores AVL: regulagem dos nós

- $h_D(p) > h_E(p)$
- Se $h_E(u) < h_D(u)$



Árvores AVL: regulagem dos nós

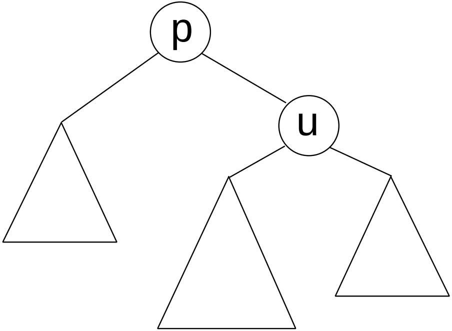
- $h_D(p) > h_E(p)$
- Se $h_E(u) < h_D(u)$



Rotação à esquerda

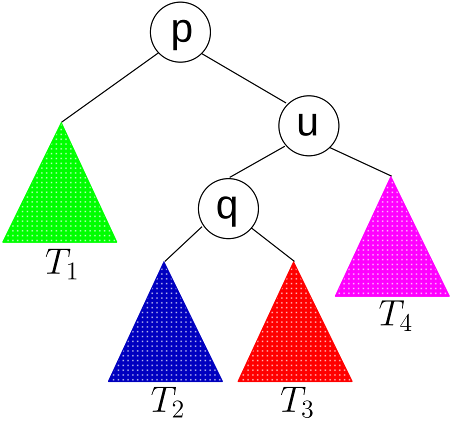
Árvores AVL: regulagem dos nós

- $h_D(p) > h_E(p)$
- Se $h_E(u) > h_D(u)$



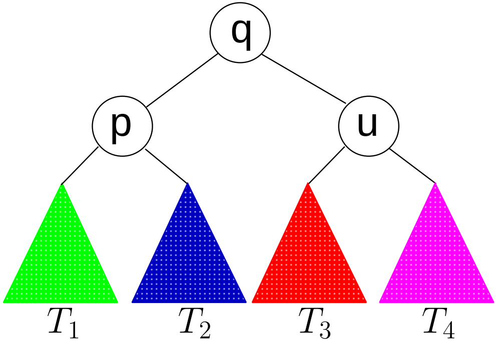
Árvores AVL: regulagem dos nós

- $h_D(p) > h_E(p)$
- Se $h_E(u) > h_D(u)$



Árvores AVL: regulagem dos nós

- $h_D(p) > h_E(p)$
- Se $h_E(u) > h_D(u)$



Rotação dupla à esquerda

Árvores AVL: regulagem de nós

q nó inserido, p ancestral desregulado, u filho de p na mesma subárvore de q.

Resumo dos casos:

- **Caso 1**: $h_E(p) > h_D(p)$

Árvores AVL: regulagem de nós

q nó inserido, p ancestral desregulado, u filho de p na mesma subárvore de q.

Resumo dos casos:

- **Caso 1**: $h_E(p) > h_D(p)$
  - **Caso 1.1**: $h_E(u) > h_D(u) \Rightarrow$ rotação direita

Árvores AVL: regulagem de nós

q nó inserido, p ancestral desregulado, u filho de p na mesma subárvore de q.
Resumo dos casos:

- **Caso 1**: $h_E(p) > h_D(p)$
  - **Caso 1.1**: $h_E(u) > h_D(u) \Rightarrow$ rotação direita
  - **Caso 1.2**: $h_E(u) < h_D(u) \Rightarrow$ rotação dupla direita

Árvores AVL: regulagem de nós

q nó inserido, p ancestral desregulado, u filho de p na mesma subárvore de q.

Resumo dos casos:

- **Caso 1**: $h_E(p) > h_D(p)$
  - **Caso 1.1**: $h_E(u) > h_D(u) \Rightarrow$ rotação direita
  - **Caso 1.2**: $h_E(u) < h_D(u) \Rightarrow$ rotação dupla direita
- **Caso 2**: $h_E(p) < h_D(p)$

Árvores AVL: regulagem de nós

q nó inserido, p ancestral desregulado, u filho de p na mesma subárvore de q.

Resumo dos casos:

- **Caso 1**: $h_E(p) > h_D(p)$
  - **Caso 1.1**: $h_E(u) > h_D(u) \Rightarrow$ rotação direita
  - **Caso 1.2**: $h_E(u) < h_D(u) \Rightarrow$ rotação dupla direita
- **Caso 2**: $h_E(p) < h_D(p)$
  - **Caso 2.1**: $h_E(u) < h_D(u) \Rightarrow$ rotação esquerda

Árvores AVL: regulagem de nós

q nó inserido, p ancestral desregulado, u filho de p na mesma subárvore de q.
Resumo dos casos:

- **Caso 1**: $h_E(p) > h_D(p)$
  - **Caso 1.1**: $h_E(u) > h_D(u) \Rightarrow$ rotação direita
  - **Caso 1.2**: $h_E(u) < h_D(u) \Rightarrow$ rotação dupla direita
- **Caso 2**: $h_E(p) < h_D(p)$
  - **Caso 2.1**: $h_E(u) < h_D(u) \Rightarrow$ rotação esquerda
  - **Caso 1.2**: $h_E(u) > h_D(u) \Rightarrow$ rotação dupla esquerda

Árvores AVL: regulagem de nós

q nó inserido, p ancestral desregulado, u filho de p na mesma subárvore de q.

Resumo dos casos:

- **Caso 1**: $h_E(p) > h_D(p)$
  - **Caso 1.1**: $h_E(u) > h_D(u) \Rightarrow$ rotação direita
  - **Caso 1.2**: $h_E(u) < h_D(u) \Rightarrow$ rotação dupla direita
- **Caso 2**: $h_E(p) < h_D(p)$
  - **Caso 2.1**: $h_E(u) < h_D(u) \Rightarrow$ rotação esquerda
  - **Caso 1.2**: $h_E(u) > h_D(u) \Rightarrow$ rotação dupla esquerda

Operação de rotação: $O(1)$ (ajustar ponteiros $p.esq$, $p.dir$, $u.esq$, $u.dir$).

Árvores AVL: regulagem de nós

q nó inserido, p ancestral desregulado, u filho de p na mesma subárvore de q.
Resumo dos casos:

- **Caso 1**: $h_E(p) > h_D(p)$
  - **Caso 1.1**: $h_E(u) > h_D(u) \Rightarrow$ rotação direita
  - **Caso 1.2**: $h_E(u) < h_D(u) \Rightarrow$ rotação dupla direita
- **Caso 2**: $h_E(p) < h_D(p)$
  - **Caso 2.1**: $h_E(u) < h_D(u) \Rightarrow$ rotação esquerda
  - **Caso 1.2**: $h_E(u) > h_D(u) \Rightarrow$ rotação dupla esquerda

Operação de rotação: $O(1)$ (ajustar ponteiros $p.esq$, $p.dir$, $u.esq$, $u.dir$).
Encontrar p: $O(\log n)$ (caminhar de q na direção da raiz)

Árvores AVL: regulagem de nós

q nó inserido, p ancestral desregulado, u filho de p na mesma subárvore de q.
Resumo dos casos:

- **Caso 1**: $h_E(p) > h_D(p)$
  - **Caso 1.1**: $h_E(u) > h_D(u) \Rightarrow$ rotação direita
  - **Caso 1.2**: $h_E(u) < h_D(u) \Rightarrow$ rotação dupla direita
- **Caso 2**: $h_E(p) < h_D(p)$
  - **Caso 2.1**: $h_E(u) < h_D(u) \Rightarrow$ rotação esquerda
  - **Caso 1.2**: $h_E(u) > h_D(u) \Rightarrow$ rotação dupla esquerda

Operação de rotação: $O(1)$ (ajustar ponteiros $p.esq$, $p.dir$, $u.esq$, $u.dir$).
Encontrar p: $O(\log n)$ (caminhar de q na direção da raiz)
Regular p torna a árvore AVL. Por quê?

Árvores AVL: algoritmo para inserção

InsereAVL($nó, chave$)

Árvores AVL: algoritmo para inserção

InsereAVL($nó, chave$)

if $chave < nó \uparrow .chave$ **then**

Árvores AVL: algoritmo para inserção

InsereAVL($nó, chave$)
if $chave < nó \uparrow .chave$ **then**
  if $nó \uparrow .ptesq \neq \lambda$ **then**
    InsereAVL($nó \uparrow .ptesq, chave$)
  else
    $nó \uparrow .ptesq =$ NovoNó($chave$)

Árvores AVL: algoritmo para inserção

InsereAVL($nó, chave$)
if $chave < nó \uparrow .chave$ **then**
  if $nó \uparrow .ptesq \neq \lambda$ **then**
    InsereAVL($nó \uparrow .ptesq, chave$)
  else
    $nó \uparrow .ptesq =$ NovoNó($chave$)
else $\{chave > nó \uparrow .chave\}$
  ...

Árvores AVL: algoritmo para inserção

InsereAVL($nó, chave$)
if $chave < nó \uparrow .chave$ **then**
  if $nó \uparrow .ptesq \neq \lambda$ **then**
    InsereAVL($nó \uparrow .ptesq, chave$)
    if $h_E(nó) > h_D(nó) + 1$ **then** {Caso 1}
      $u \leftarrow nó \uparrow .ptesq$
      if $h_E(u) > h_D(u) + 1$ **then** {Caso 1.1}
        RotaçãoDireita($nó, u$)
      else {Caso 1.2}
        $v \leftarrow u \uparrow .ptdir$
        RotaçãoDuplaDireita($nó, u, v$)
      ...
  else
    $nó \uparrow .ptesq =$ NovoNó($chave$)

Árvores AVL: Detalhes de implementação

- O cálculo de h_E e h_D demanda tempo $O(\log n)$.

Árvores AVL: Detalhes de implementação

- O cálculo de h_E e h_D demanda tempo $O(\log n)$.
- Esse cálculo é feito uma vez para cada nó entre q e a raiz.

Árvores AVL: Detalhes de implementação

- O cálculo de h_E e h_D demanda tempo $O(\log n)$.
- Esse cálculo é feito uma vez para cada nó entre q e a raiz.
- Número de operações: $O(\log n \log n)$ excede o desejado!

Árvores AVL: Detalhes de implementação

- O cálculo de h_E e h_D demanda tempo $O(\log n)$.
- Esse cálculo é feito uma vez para cada nó entre q e a raiz.
- Número de operações: $O(\log n \log n)$ excede o desejado!
- Na verdade, não preciso saber exatamente h_E e h_D, mas apenas se um nó está desregulado e qual subárvore é a maior.

Árvores AVL: Detalhes de implementação

- O cálculo de h_E e h_D demanda tempo $O(\log n)$.
- Esse cálculo é feito uma vez para cada nó entre q e a raiz.
- Número de operações: $O(\log n \log n)$ excede o desejado!
- Na verdade, não preciso saber exatamente h_E e h_D, mas apenas se um nó está desregulado e qual subárvore é a maior.
- Uso de um campo *bal* $\in \{-1, 0, 1\}$ para armazenar o *balanço* de cada nó: $bal := h_D - h_E$.

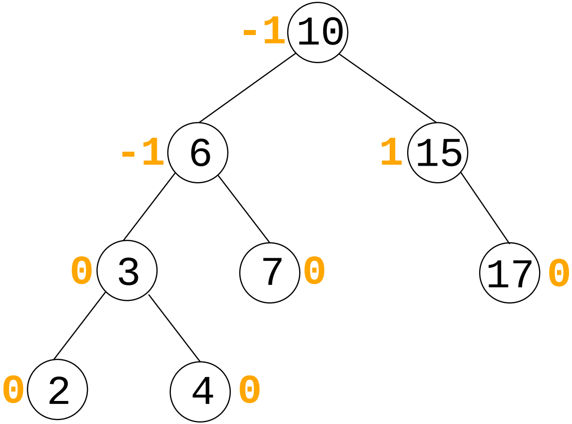
Árvores AVL: Detalhes de implementação

- O cálculo de h_E e h_D demanda tempo $O(\log n)$.
- Esse cálculo é feito uma vez para cada nó entre q e a raiz.
- Número de operações: $O(\log n \log n)$ excede o desejado!
- Na verdade, não preciso saber exatamente h_E e h_D, mas apenas se um nó está desregulado e qual subárvore é a maior.
- Uso de um campo $bal \in \{-1, 0, 1\}$ para armazenar o *balanço* de cada nó: $bal := h_D - h_E$.
- A cada inserção, atualizo o balanço dos ancestrais de q. Se algum balanço tornar-se -2 ou 2, faço as rotações apropriadas.

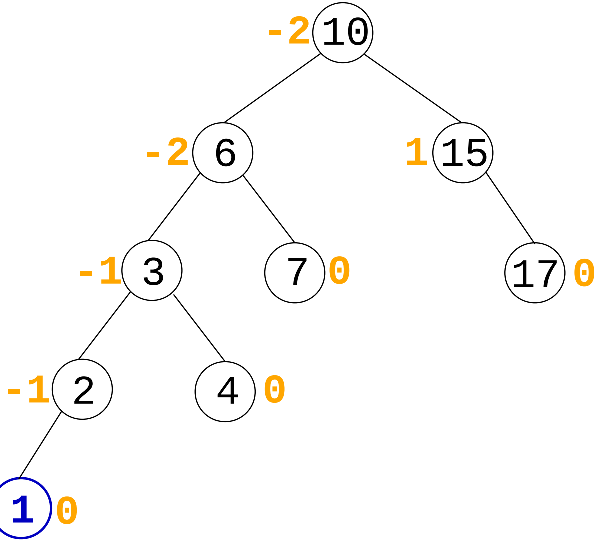
Árvores AVL: Detalhes de implementação

- O cálculo de h_E e h_D demanda tempo $O(\log n)$.
- Esse cálculo é feito uma vez para cada nó entre q e a raiz.
- Número de operações: $O(\log n \log n)$ excede o desejado!
- Na verdade, não preciso saber exatamente h_E e h_D, mas apenas se um nó está desregulado e qual subárvore é a maior.
- Uso de um campo $bal \in \{-1, 0, 1\}$ para armazenar o *balanço* de cada nó: $bal := h_D - h_E$.
- A cada inserção, atualizo o balanço dos ancestrais de q. Se algum balanço tornar-se -2 ou 2, faço as rotações apropriadas.
- Um novo nó tem balanço 0.

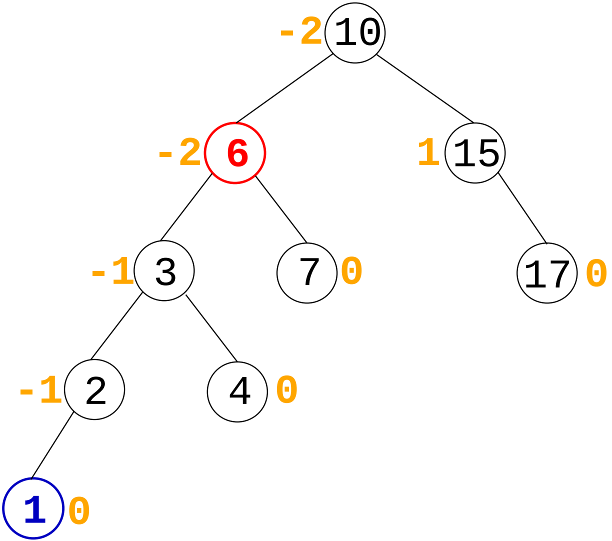
Árvores AVL: Detalhes de implementação



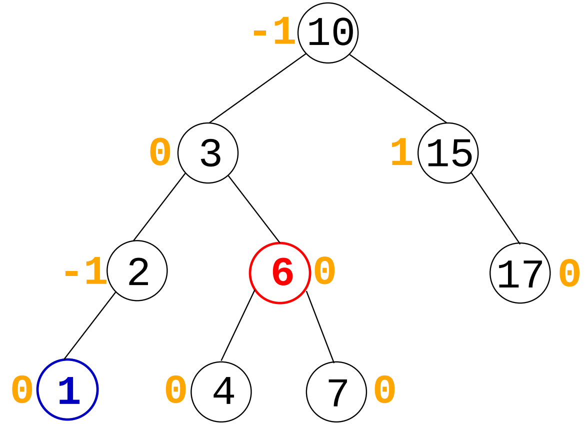
Árvores AVL: Detalhes de implementação



Árvores AVL: Detalhes de implementação



Árvores AVL: Detalhes de implementação



Árvores AVL: remoção

1. faça a remoção como na árvore de busca binária: $O(\log n)$

Árvores AVL: remoção

1. faça a remoção como na árvore de busca binária: $O(\log n)$
2. se outro nó ocupar o lugar do nó removido, atualize os balanços desde o pai desse nó até a posição atual do nó: $O(\log n)$

Árvores AVL: remoção

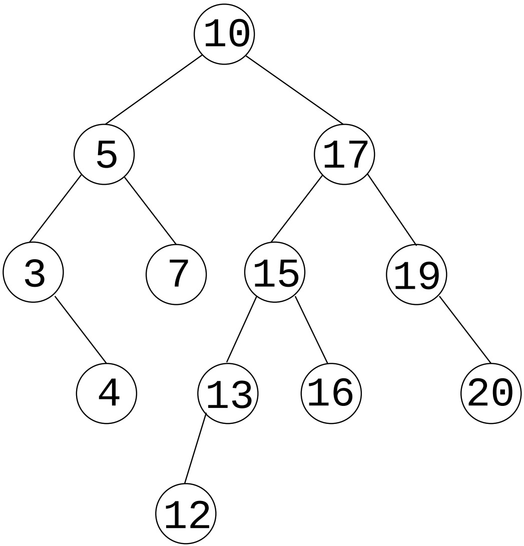
1. faça a remoção como na árvore de busca binária: $O(\log n)$
2. se outro nó ocupar o lugar do nó removido, atualize os balanços desde o pai desse nó até a posição atual do nó: $O(\log n)$
3. percorra o caminho desde o pai do nó removido até a raiz, fazendo as operações de rotação aprorpiadas (pode ser mais de uma): $O(\log n)$

Árvores AVL: remoção

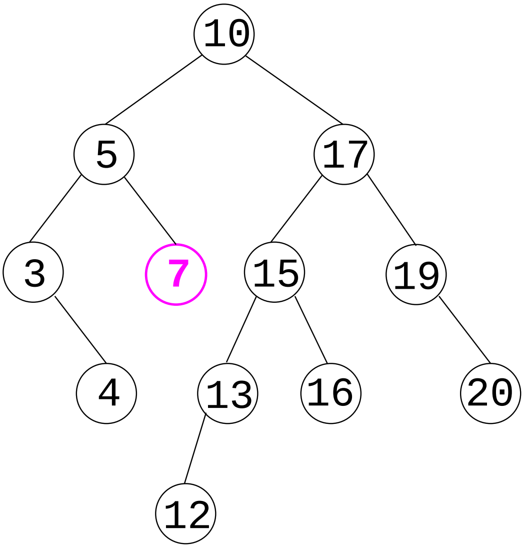
1. faça a remoção como na árvore de busca binária: $O(\log n)$
2. se outro nó ocupar o lugar do nó removido, atualize os balanços desde o pai desse nó até a posição atual do nó: $O(\log n)$
3. percorra o caminho desde o pai do nó removido até a raiz, fazendo as operações de rotação aprorpiadas (pode ser mais de uma): $O(\log n)$

Total: $O(\log n)$.

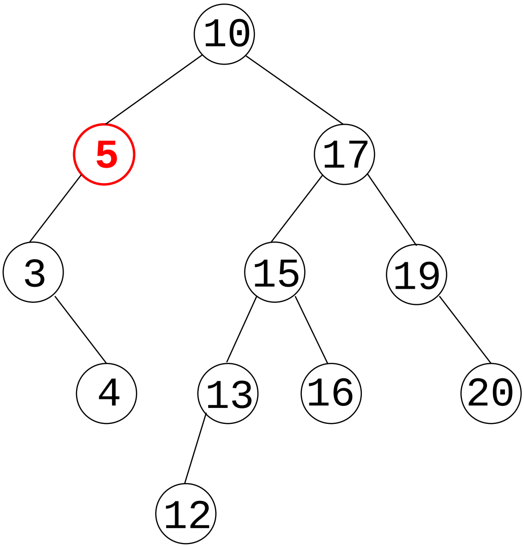
Árvores AVL: remoção



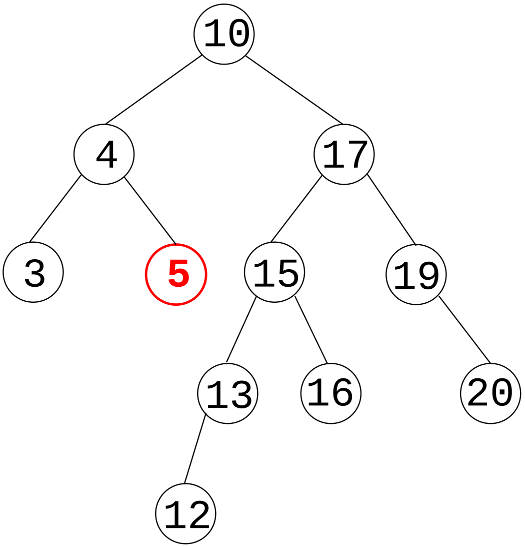
Árvores AVL: remoção



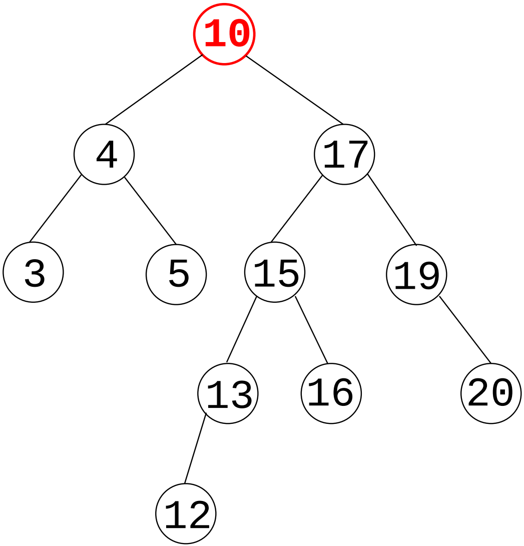
Árvores AVL: remoção



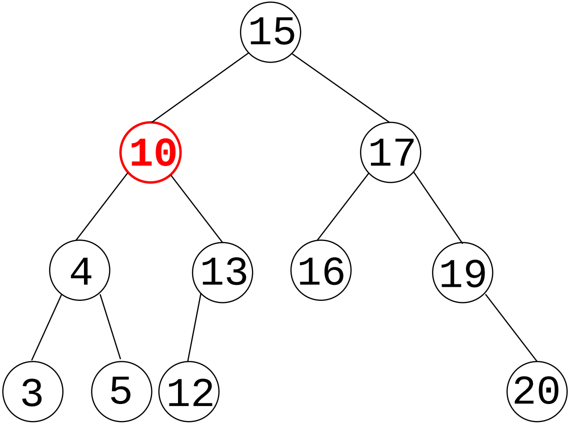
Árvores AVL: remoção



Árvores AVL: remoção



Árvores AVL: remoção



Conclusão

- Árvore binária de busca ótima: inserções/remoções não são viáveis.

Conclusão

- Árvore binária de busca ótima: inserções/remoções não são viáveis.
- Árvore binária balanceada: alternativa *suficientemente boa*.

Conclusão

- Árvore binária de busca ótima: inserções/remoções não são viáveis.
- Árvore binária balanceada: alternativa *suficientemente boa*.
- Árvores balanceadas são muito usadas na vida real: ex. `TreeMap` e `TreeSet` do Java.util. Em C++, `Map` e `Set` do STL.

Conclusão

- Árvore binária de busca ótima: inserções/remoções não são viáveis.
- Árvore binária balanceada: alternativa *suficientemente boa*.
- Árvores balanceadas são muito usadas na vida real: ex. `TreeMap` e `TreeSet` do Java.util. Em C++, `Map` e `Set` do STL.
- Árvore AVL: tipo de árvore balanceada.

Conclusão

- Árvore binária de busca ótima: inserções/remoções não são viáveis.
- Árvore binária balanceada: alternativa *suficientemente boa*.
- Árvores balanceadas são muito usadas na vida real: ex. `TreeMap` e `TreeSet` do Java.util. Em C++, `Map` e `Set` do STL.
- Árvore AVL: tipo de árvore balanceada.
- Busca, inserção, remoção em árvore AVL: $O(\log n)$.

Conclusão

- Árvore binária de busca ótima: inserções/remoções não são viáveis.
- Árvore binária balanceada: alternativa *suficientemente boa*.
- Árvores balanceadas são muito usadas na vida real: ex. `TreeMap` e `TreeSet` do Java.util. Em C++, `Map` e `Set` do STL.
- Árvore AVL: tipo de árvore balanceada.
- Busca, inserção, remoção em árvore AVL: $O(\log n)$.

Próxima aula: mais árvores balanceadas – Árvores vermelho-pretas (*red-black trees*).

Bibliografia Utilizada

SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.