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.

Ideia: Para cada chave, separe as restantes em maiores ou menores.

Relembrando: árvores binárias de busca

Objetivo: minimizar tempo de acesso no pior caso.

Ideia: Para cada chave,separe as restantes em maiores ou menores.

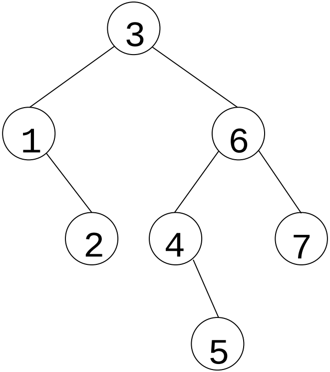
Estrutura hierárquica com divisão binária: uma árvore binária.

Relembrando: árvores binárias de busca

Objetivo: minimizar tempo de acesso no pior caso.

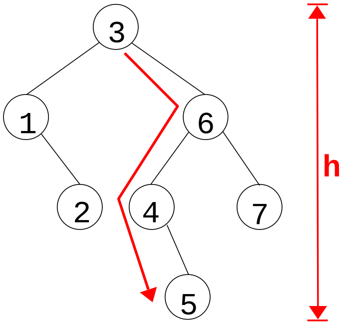
Ideia: Para cada chave,separe as restantes em maiores ou menores.

Estrutura hierárquica com divisão binária: uma árvore binária.



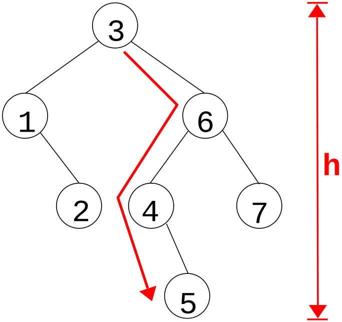
Relembrando: complexidade da busca em árvore binária

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

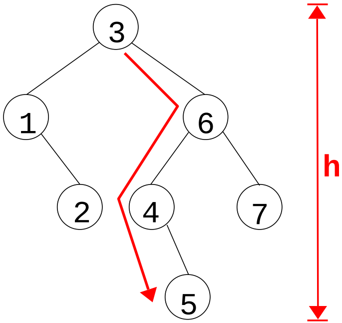
Busca em árvore binária = caminho da raiz até chave desejada (ou até uma folha, caso chave não exista).



Pior caso:

Relembrando: complexidade da busca em árvore binária

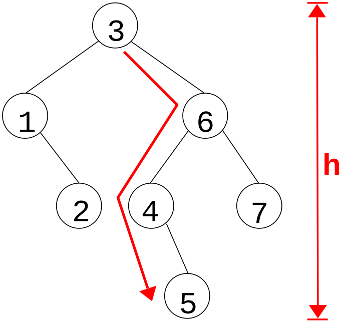
Busca em árvore binária = caminho da raiz até chave desejada (ou até uma folha, caso chave não exista).



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).

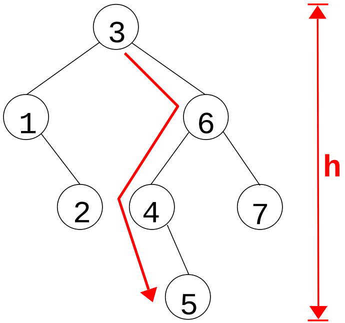


Pior caso: maior caminho da raiz até folha = altura da árvore

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?)

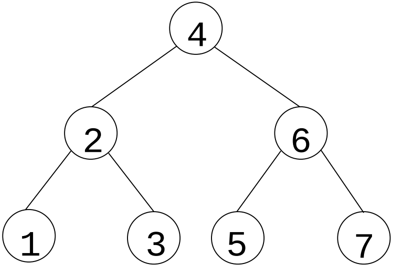
Relembrando: árvore binária de busca ótima

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

Relembrando: árvore binária de busca ótima

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

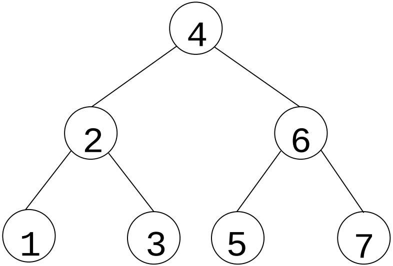
Árvore completa, altura: h = piso(log n) + 1



Relembrando: árvore binária de busca ótima

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

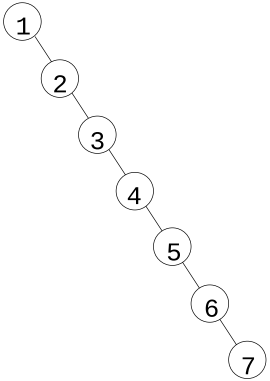
Árvore completa, altura: $h = \log n + 1$



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 $O(\ldots)$

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 $c1 \log n + c2$, complexidade temporal no pior caso $O(\ldots)$

Árvores suficientemente boas

Árvore com altura 2 log n, complexidade temporal no pior caso O(logn)

Árvore com altura c1 logn + c2 , complexidade temporal no pior caso O($log\ n$)

Árvores suficientemente boas

Árvore com altura 2 log n, complexidade temporal no pior caso O(logn)

Árvore com altura c1 logn + c2 , 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 c1 log n + c2 , complexidade temporal no pior caso O($(log\ n)$)

Nem toda árvore com altura O(logn) 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 O(log n)
Árvore com altura c1 log n + c2 , complexidade temporal caso O$(log\ n)$
Nem toda árvore com altura O(logn) é ótima, mas a complexidade assintótica temporal igual à de uma árvore ótima.

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

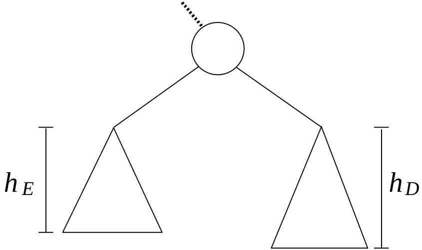
Mais fácil de construir que árvore ótima?
Como garantir que uma árvore binária é balanceada?

Árvores AVL

Para cada nó x, defina:

$h_E(x)$: Altura sub-árvore à esquerda

$h_D(x)$: Altura sub-árvore à direita

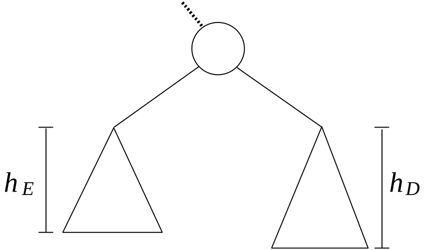


Árvores AVL

Para cada nó x, defina:

$h_E(x)$: Altura sub-árvore à esquerda

$h_D(x)$: Altura sub-árvore à direita



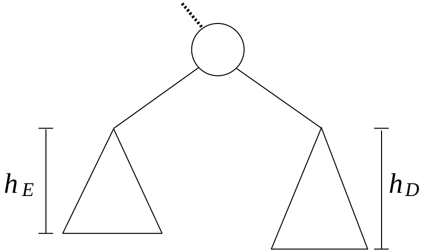
Propriedade AVL: $|h_E(x) - h_D(x)| \leq 1$

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

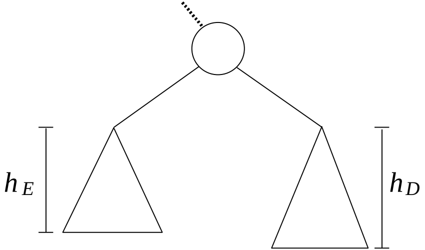
Nó regulado: satisfaz propriedade AVL.

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.

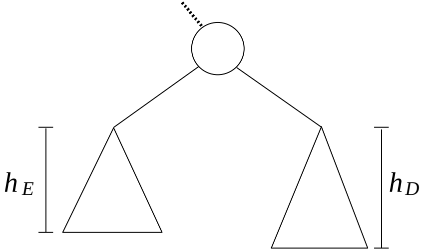
Árvore AVL: todos nós regulados.

Árvores AVL

Para cada nó x, defina:

$h_E(x)$: Altura sub-árvore à esquerda

$h_D(x)$: Altura sub-árvore à direita



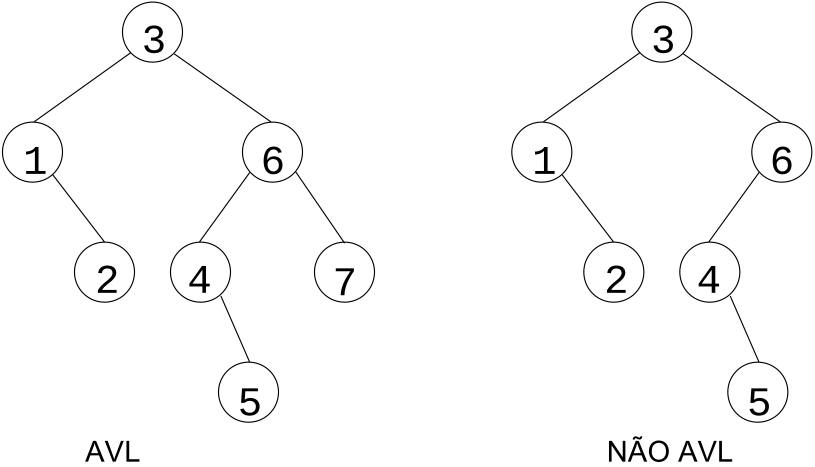
Propriedade AVL: $|h_E(x) - h_D(x)| \leq 1$

Nó regulado: satisfaz propriedade AVL.

Árvore AVL: todos nós regulados.

(Curiosidade: AVL = Adelson-Velskii, G. e Landis, E. M.)

Árvores AVL: exemplo



Árvores AVL são balanceadas

Intuitivamente: diferença pequena de altura entre sub-árvores
$\Rightarrow$ nós se distribuem mais uniformemente.

Provaremos: Toda árvore AVL é balanceada.
(Mas cuidado: nem toda árvore balanceada é AVL.)

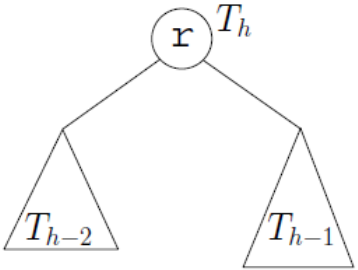
Demonstração: T_h árvore AVL com altura h e mínimo de nós, $|T_h|$ número de nós

- $h = 0$, então $|T_h| = 0$ (árvore vazia)
- $h = 1$, então $|T_h| = 1$ (só a raiz)
- $h = 2$, então $|T_h| = 2$ (raiz mais um nó)
- Generalizar para $h > 1$...

Árvores AVL são balanceadas

(continuação da demonstração)

- Generalizando para $h > 1$.



- **Mínimo** de nós com altura h: raiz $\cup$ T_{h-1} $\cup$ T_{h-2}
- Fórmula geral: $|T_h| = 1 + |T_{h-1}| + |T_{h-2}|$

Árvores AVL são balanceadas

(continuação da demonstração)
Fórmula recursiva para $|T_h|$:

$$|T_h| = \begin{cases} 0 & h = 0 \\ 1 & h = 1 \\ 1 + |T_{h-1}| + |T_{h-2}| & h > 1 \end{cases}$$

Esta fórmula é muito similar a uma sequência muito conhecida.
h-ésimo número de Fibonacci, F_h:

$$F_h = \begin{cases} 0 & h = 0 \\ 1 & h = 1 \\ F_{h-1} + F_{h-2} & h > 1 \end{cases}$$

Veja que: $|T_h| > F_h$.
O que ganhamos com isto?

Árvores AVL são balanceadas

(continuação da demonstração)
Número de nós em árvore AVL: $n \geq |T_h| \geq F_h$.
Existe fórmula fechada para F_h:

$$F_h = \frac{1}{\sqrt{5}} a^h - \frac{1}{\sqrt{5}} b^h$$

Onde $a = \frac{1+\sqrt{5}}{2}$, e $b = \frac{1-\sqrt{5}}{2}$.
Constante b menor que 1: $b = -.2763\ldots$. Então $|b^h| < 1$.
Logo:

$$n \geq F_h \geq \frac{1}{\sqrt{5}} a^h - \mathbf{1}$$

Lembre-se: queremos h em função de n.

Árvores AVL são balanceadas

(continuação da demonstração)

$$n \geq \frac{1}{\sqrt{5}} a^h - 1$$

$$n + 1 \geq \frac{1}{\sqrt{5}} a^h$$

Use logaritmos:

$$\begin{aligned} \log_a(n+1) &\geq \log_a \left(\frac{1}{\sqrt{5}} a^h \right) \\ &\geq \log_a \frac{1}{\sqrt{5}} + \log_a a^h \\ &\geq \log_a \frac{1}{\sqrt{5}} + h \end{aligned}$$

Árvores AVL são balanceadas

(continuação da demonstração)

Rearranjando $\log_a(n+1) \geq \underbrace{\log_a \frac{1}{\sqrt{5}}}_{c} + h$:

$$
\begin{aligned}
h &\leq \frac{\log_2(n+1)}{\log_2 a} + c \\
h &\in O(\log n)
\end{aligned}
$$

Árvores AVL são balanceadas!

Árvores AVL: inserção

Queremos inserir nova chave:

- mantendo a regulagem de to dos os nós
- em tempo razoável

Nossa estratégia:

Árvores AVL: inserção

Queremos inserir nova chave:

- mantendo a regulagem de to dos os nós
- em tempo razoável

Nossa estratégia:

1. Inserção como árvore binária comum.

Árvores AVL: inserção

Queremos inserir nova chave:

- mantendo a regulagem de to dos 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:

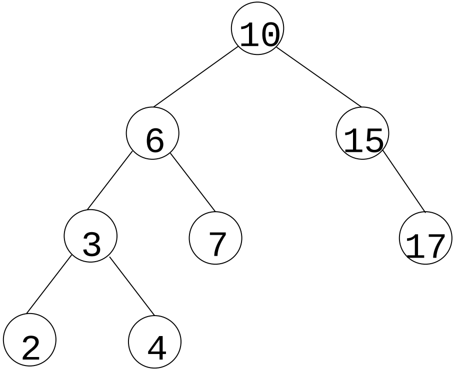
- mantendo a regulagem de to dos os nós
- em tempo razoável

Nossa estratégia:

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

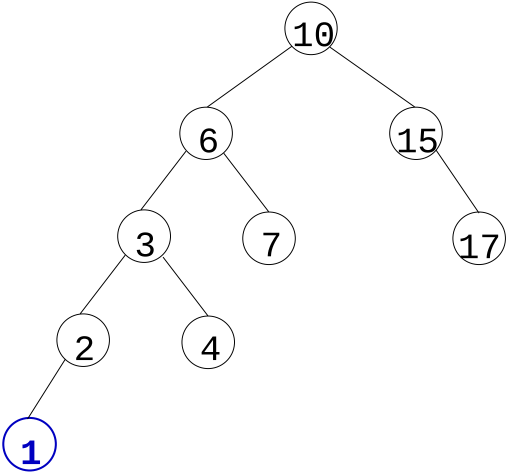
Árvores AVL: exemplo de inserção

Inserir chave com valor 1 na árvore abaixo.



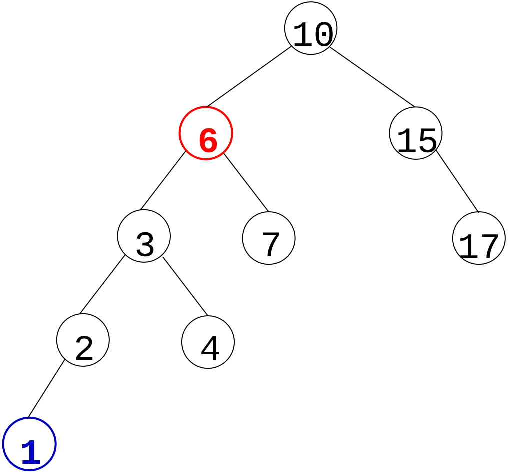
Árvores AVL: exemplo de inserção

Inserir chave com valor 1 na árvore abaixo.



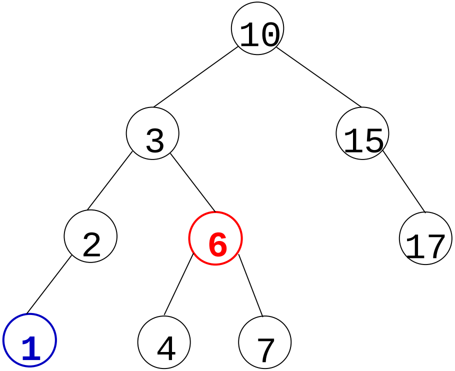
Árvores AVL: exemplo de inserção

Inserir chave com valor 1 na árvore abaixo.



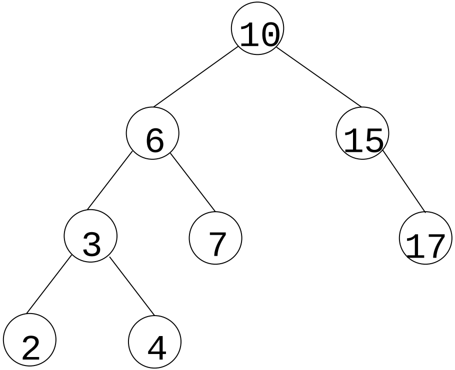
Árvores AVL: exemplo de inserção

Inserir chave com valor 1 na árvore abaixo.



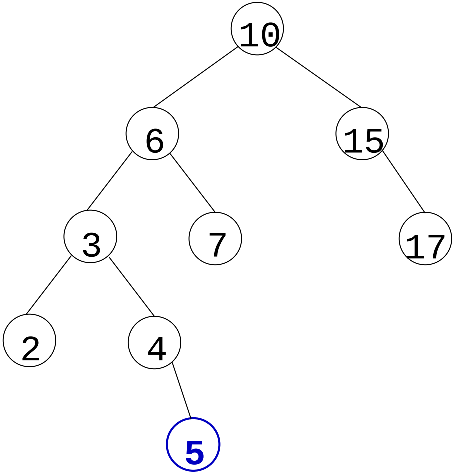
Árvores AVL: exemplo de inserção

Inserir chave com valor 5 na árvore abaixo.



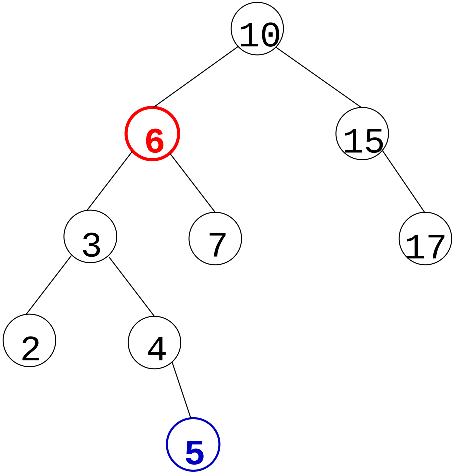
Árvores AVL: exemplo de inserção

Inserir chave com valor 5 na árvore abaixo.



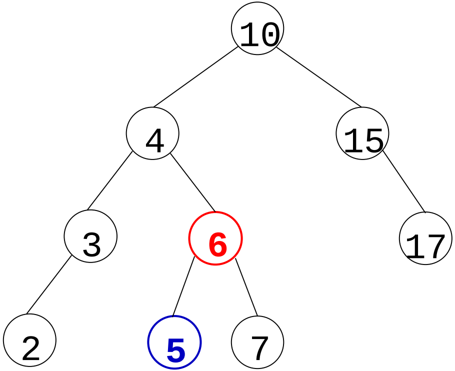
Árvores AVL: exemplo de inserção

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

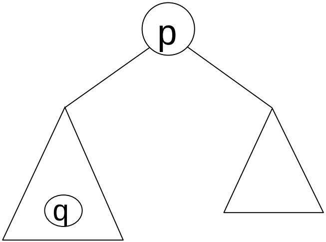
Generalizando os casos anteriores.
Nó inserido:q

- 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

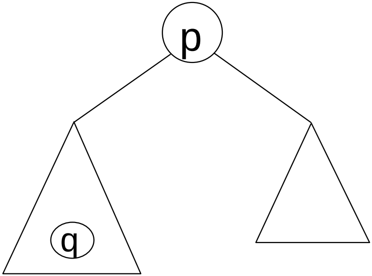
$h_E(p) > h_D(p)$



Árvores AVL: regulagem dos nós

$h_E(p) > h_D(p)$

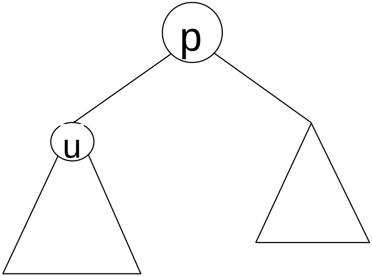
.



Árvores AVL: regulagem dos nós

$h_E(p) > h_D(p)$

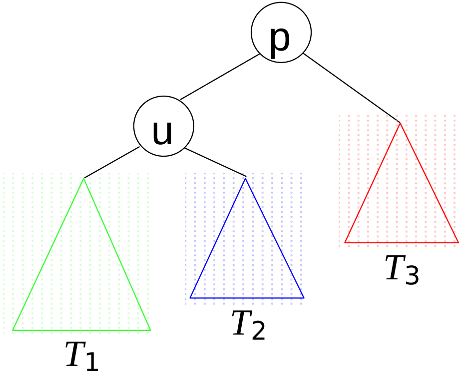
.



Árvores AVL: regulagem dos nós

$h_E(p) > h_D(p)$

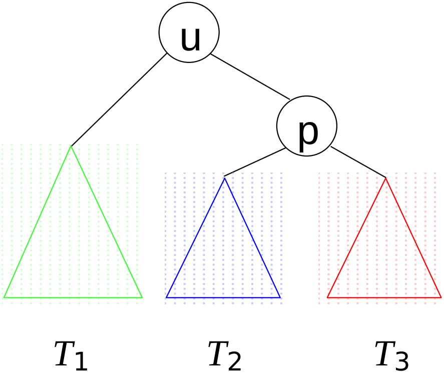
Se $h_E(u) > h_D(u)$



Árvores AVL: regulagem dos nós

$h_E(p) > h_D(p)$

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

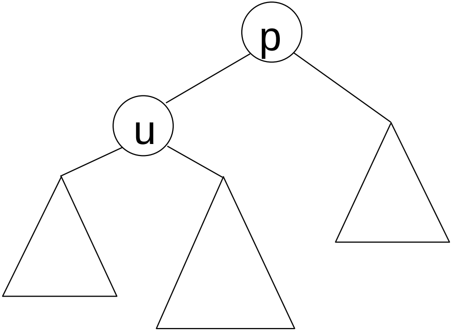


Rotação à direita

Árvores AVL: regulagem dos nós

$h_E(p) > h_D(p)$

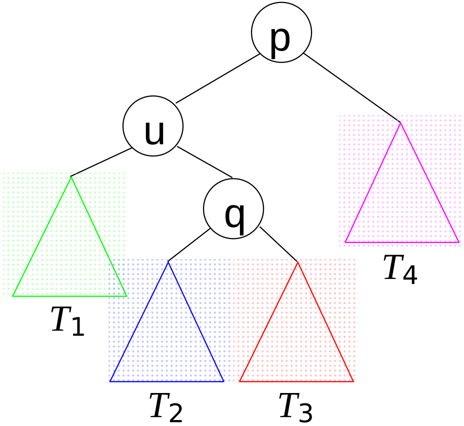
Se $h_E(u) < h_D(u)$



Árvores AVL: regulagem dos nós

$h_E(p) > h_D(p)$

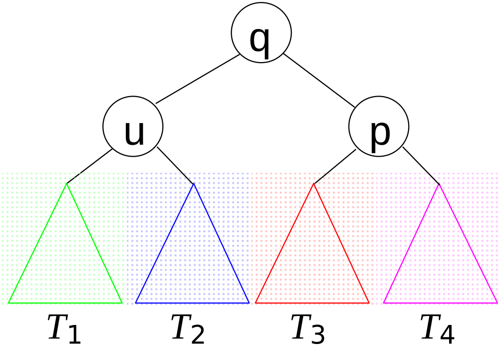
Se $h_E(u) < h_D(u)$



Árvores AVL: regulagem dos nós

$h_E(p) > h_D(p)$

Se $h_E(u) < h_D(u)$



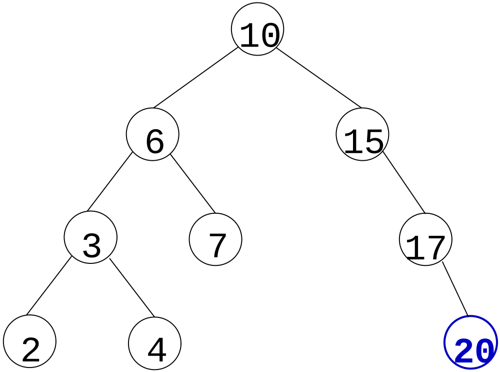
Rotação dupla à direita

Árvores AVL: exemplo de inserção

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

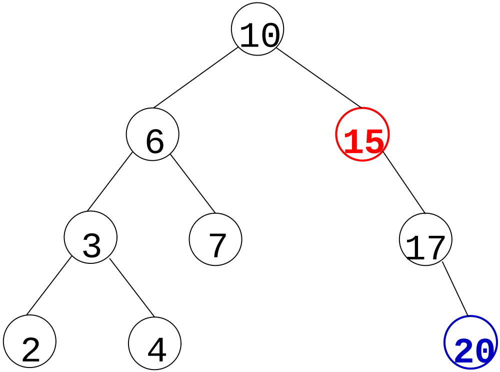
Árvores AVL: exemplo de inserção

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



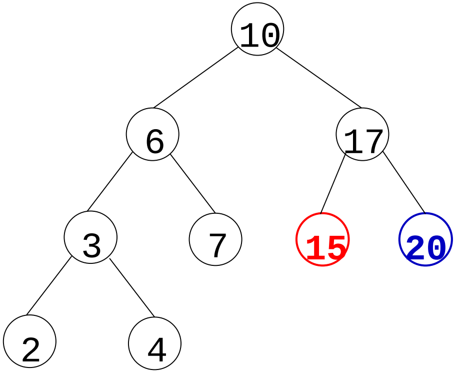
Árvores AVL: exemplo de inserção

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



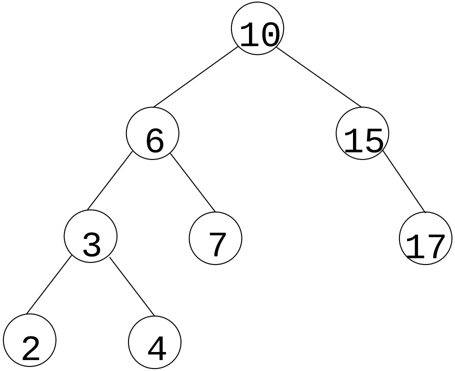
Árvores AVL: exemplo de inserção

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



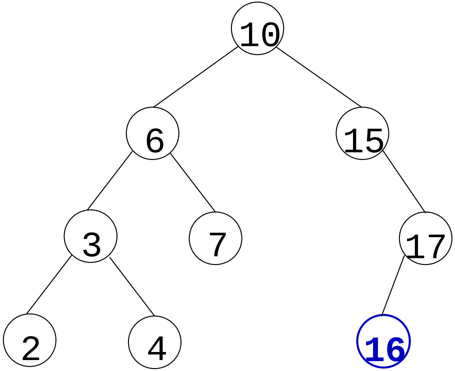
Árvores AVL: exemplo de inserção

Inserir 16 na árvore abaixo:



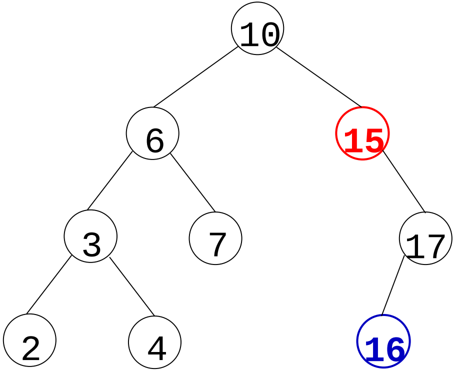
Árvores AVL: exemplo de inserção

Inserir 16 na árvore abaixo:



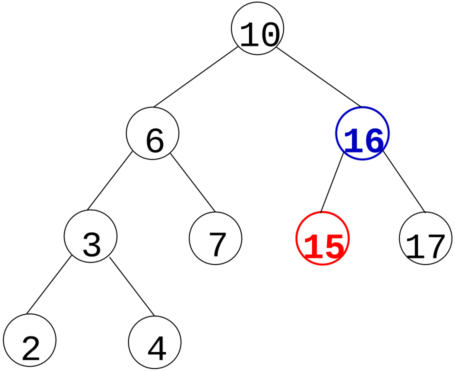
Árvores AVL: exemplo de inserção

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

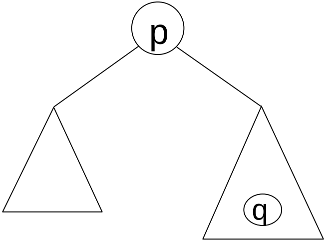
$h_D(p) > h_E(p)$

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

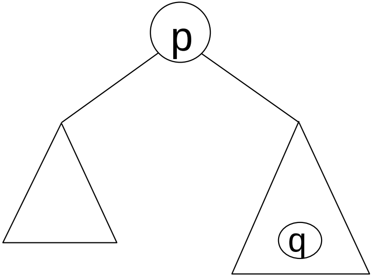
$h_D(p) > h_E(p)$



Árvores AVL: regulagem dos nós

$h_D(p) > h_E(p)$

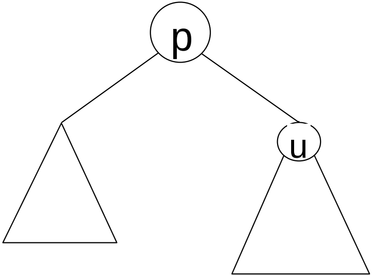
.



Árvores AVL: regulagem dos nós

$h_D(p) > h_E(p)$

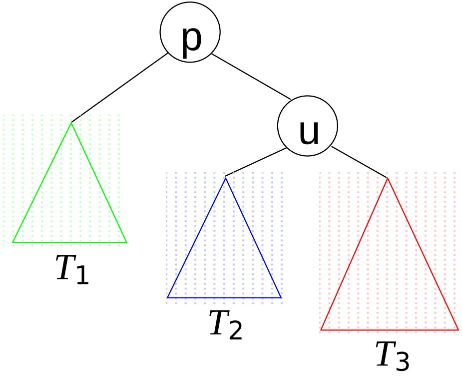
.



Árvores AVL: regulagem dos nós

$h_D(p) > h_E(p)$

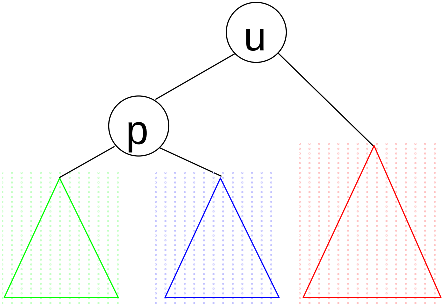
Se $h_E(u) < h_D(u)$



Árvores AVL: regulagem dos nós

$h_D(p) > h_E(p)$

Se $h_E(u) < h_D(u)$



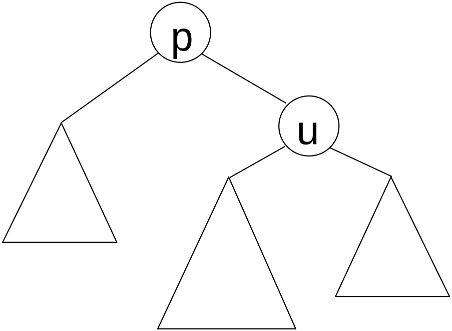
T_1 T_2 T_3

Rotação à esquerda

Árvores AVL: regulagem dos nós

$h_D(p) > h_E(p)$

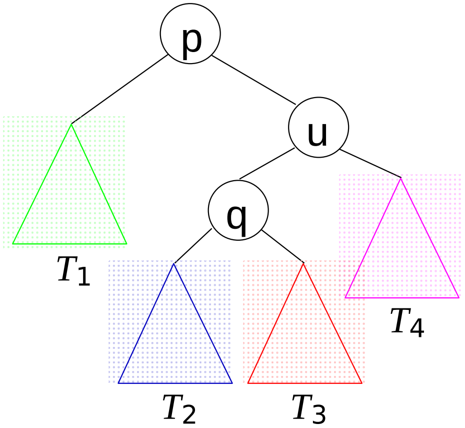
Se $h_E(u) > h_D(u)$



Árvores AVL: regulagem dos nós

$h_D(p) > h_E(p)$

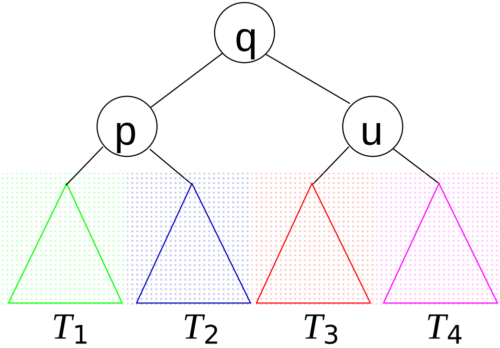
Se $h_E(u) > h_D(u)$



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 2.2: $h_E(u) > h_D(u) \Rightarrow$ 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 2.2: $h_E(u) > h_D(u) \Rightarrow$ 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 2.2: $h_E(u) > h_D(u) \Rightarrow$ 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 2.2: $h_E(u) > h_D(u) \Rightarrow$ 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. Porquê?

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 ↑ .chave then
  if nó ↑ .ptesq ≠ λ then
    InsereAVL(nó ↑ .ptesq, chave)
    if h_E(nó) > h_D(nó) + 1 then {Caso 1}
      u ← nó ↑ .ptesq
      if h_E(u) > h_D(u) + 1 then {Caso 1.1}
        RotaçãoDireita(nó, u)
      else {Caso 1.2}
        v ← u ↑ .ptdir
        RotaçãoDuplaDireita(nó, u, v)
      ...
  else
    nó ↑ .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: $bal := h_D - h_E$.

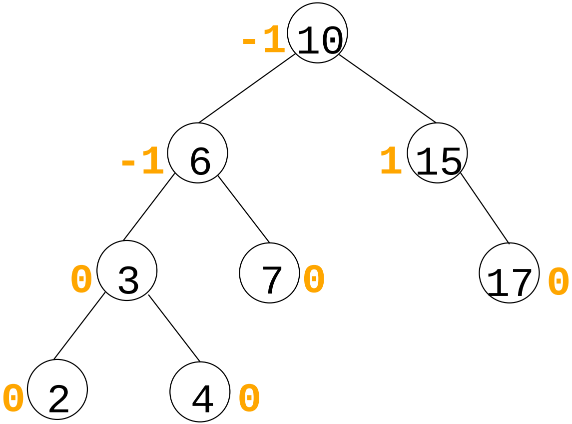
Árvores AVL: Detalhes de implementação

- 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.

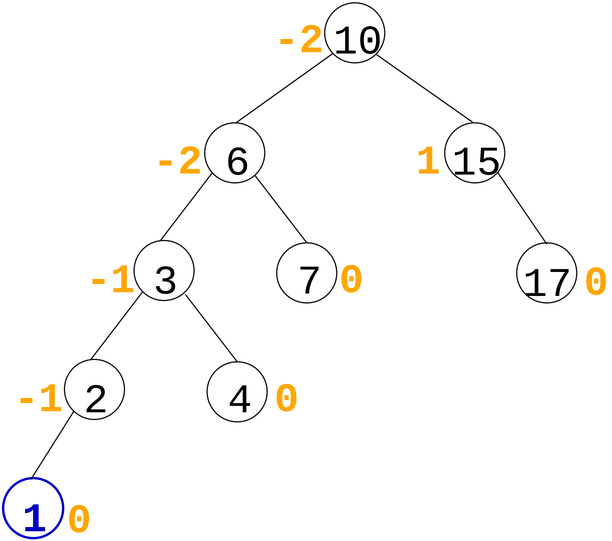
Árvores AVL: Detalhes de implementação

- 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.

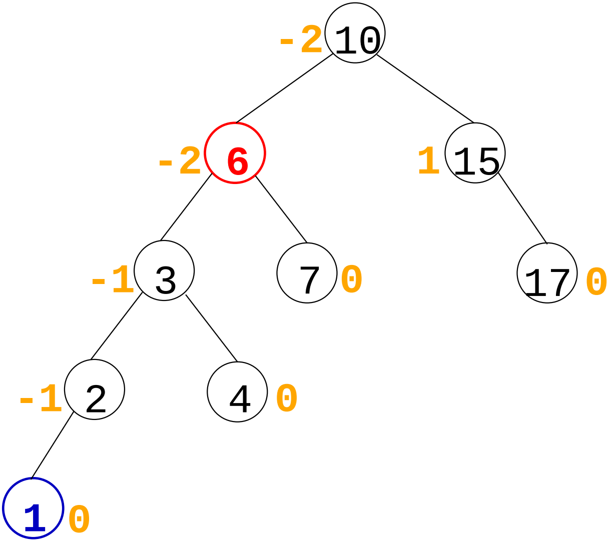
Árvores AVL: Detalhes de implementação



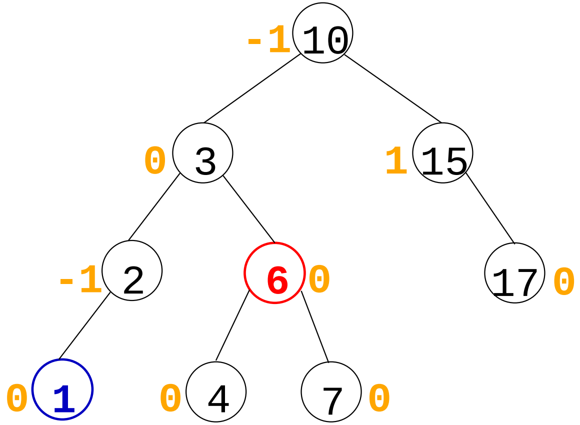
Árvores AVL: Detalhes de implementação



Árvores AVL: Detalhes de implementação



Árvores AVL: Detalhes de implementação



Algoritmos de inserção e balanceamento em AVL

Função inicio-no

$\texttt{procedimento}\ \textit{inicio-no}(pt)$

$\quad ocupar(pt)$

$\quad pt \uparrow .esq := \lambda; \quad pt \uparrow .dir := \lambda$

$\quad pt \uparrow .chave := x; \quad pt \uparrow .bal := 0$

Algoritmos de inserção e balanceamento em AVL

Função caso1

procedimento $caso1(pt, h)$

$ptu := pt \uparrow .esq$

se $ptu \uparrow .bal = -1$ então

$pt \uparrow .esq := ptu \uparrow .dir; \quad ptu \uparrow .dir := pt$

$pt \uparrow .bal := 0; \quad pt := ptu$

senão $ptv := ptu \uparrow .dir$

$ptu \uparrow .dir := ptv \uparrow .esq; \quad ptv \uparrow .esq := ptu$

$pt \uparrow .esq := ptv \uparrow .dir; \quad ptv \uparrow .dir := pt$

se $ptv \uparrow .bal = -1$ então $pt \uparrow .bal := 1$ senão $pt \uparrow .bal := 0$

se $ptv \uparrow .bal = 1$ então $ptu \uparrow .bal := -1$ senão $ptu \uparrow .bal := 0$

$pt := ptv$

$pt \uparrow .bal := 0; \quad h := F$

Algoritmos de inserção e balanceamento em AVL

Função caso2

```
procedimento caso2(pt, h)
    ptu := pt↑.dir
    se ptu↑.bal = 1 então
        pt↑.dir := ptu↑.esq;   ptu↑.esq := pt
        pt↑.bal := 0;   pt := ptu
    senão ptv := ptu↑.esq
        ptu↑.esq := ptv↑.dir;   ptv↑.dir := ptu
        pt↑.dir := ptv↑.esq;   ptv↑.esq := pt
        se ptv↑.bal = 1 então pt↑.bal := −1 senão pt↑.bal := 0
        se ptv↑.bal = −1 então ptu↑.bal := 1 senão ptu↑.bal := 0
        pt := ptv
    pt↑.bal := 0;   h := F
```

Algoritmos de inserção e balanceamento em AVL

Função ins-AVL

```
procedimento ins-AVL(x, pt, h)
    se pt = λ então
        inicio-no(pt)
        h := V
    senão se x = pt↑.chave então pare
        se x < pt↑.chave então
            ins-AVL(x, pt↑.esq, h)
            se h então
                caso pt↑.bal seja
                 1:    pt↑.bal := 0;   h := F
                 0:    pt↑.bal := -1
                -1:    caso1(pt, h)            % rebalanceamento
        senão ins-AVL(x, pt↑.dir, h)
            se h então
                caso pt↑.bal seja
                -1:    pt↑.bal := 0;   h := F
                 0:    pt↑.bal := 1
                 1:    caso2(pt, h)            % rebalanceamento
```

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 apropriadas (p o de ser mais de uma): $O(\log n)$

Árvores AVL: remoção

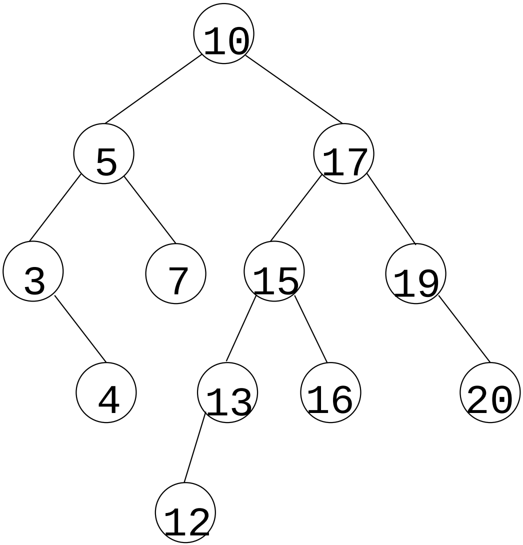
1. faça a remoção como na árvore de busca binária: $O(\log n)$

2. se outro nó ocupar o lugar do nó removido, atualize os balanços desde o pai desse nó até a posição atual do nó: $O(\log n)$

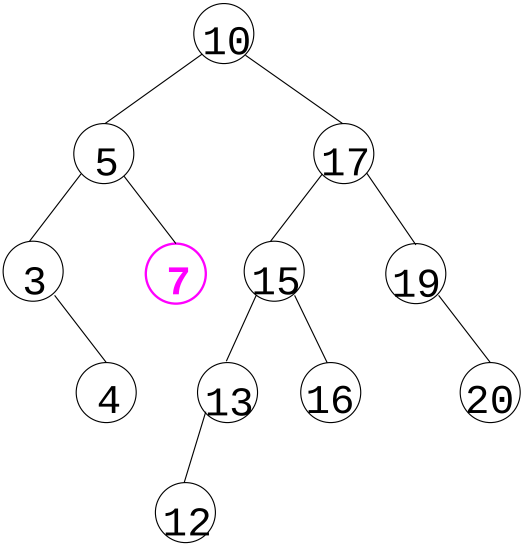
3. percorra o caminho desde o pai do nó removido até a raiz, fazendo as operações de rotação apropriadas (p o de ser mais de uma): $O(\log n)$

Total: $O(\log n)$.

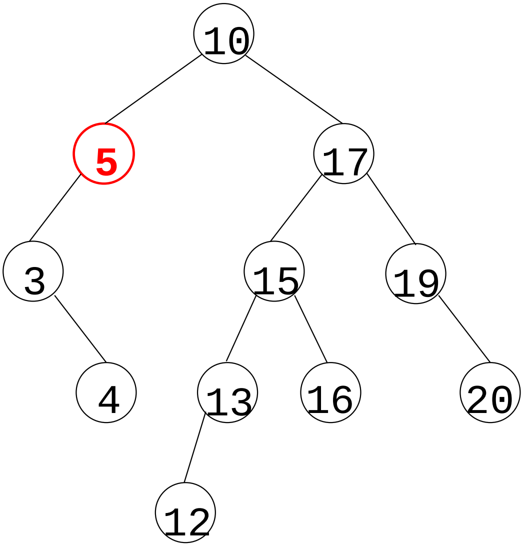
Árvores AVL: remoção



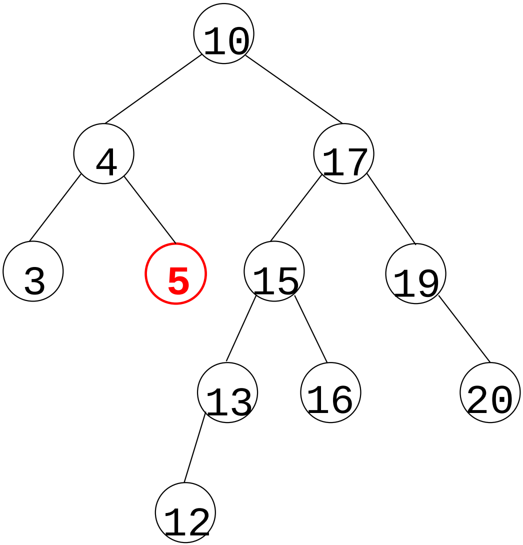
Árvores AVL: remoção



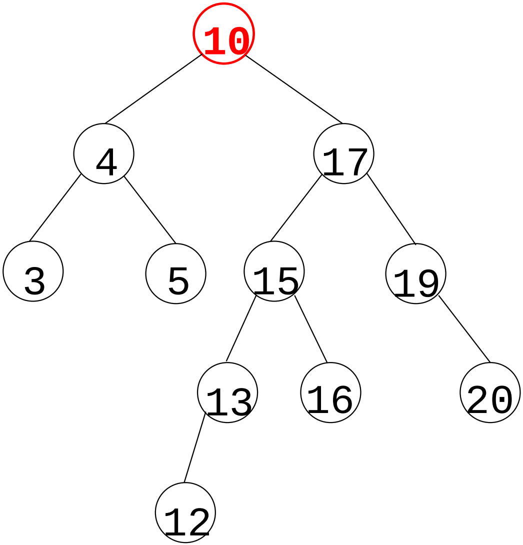
Árvores AVL: remoção



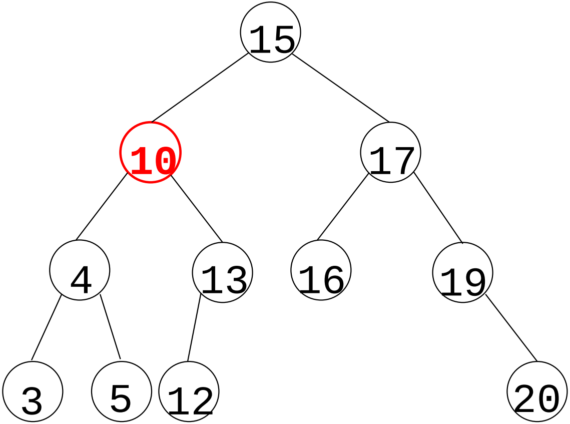
Árvores AVL: remoção



Árvores AVL: remoção



Árvores AVL: remoção



Conclusão

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

Árvore binária balanceada: alternativa suficientemente boa.

Árvore AVL: tipo de árvore balanceada.
Busca, inserção, remoção em árvore AVL: $O(\log n)$.

Bibliografia Utilizada

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