Algoritmos e Estrutura de Dados

Fabrício Olivetti de França

02 de Fevereiro de 2019

UFABC

CMCC
Centro de Matemática, Computação e Cognição
Center for Mathematics, Computation and Cognition

Topics

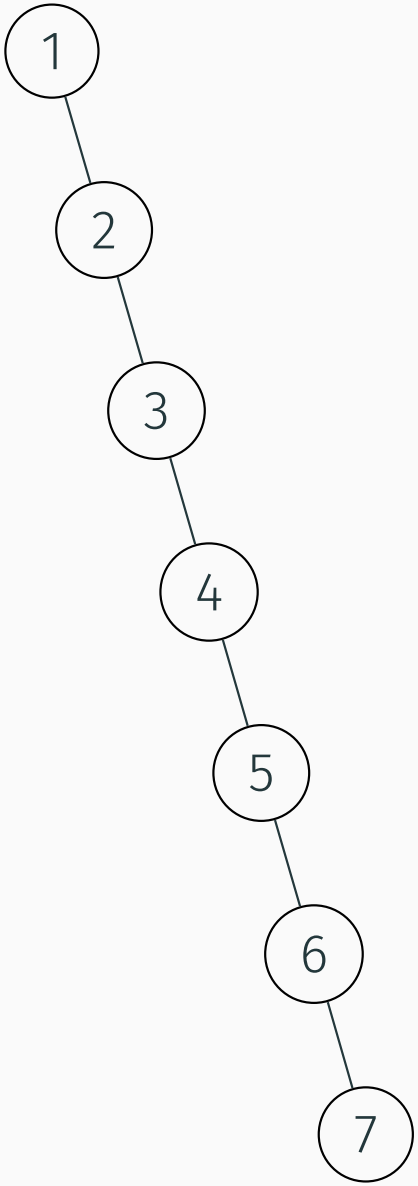
1. Árvores Balanceadas

Árvores Balanceadas

Árvores Binárias Desbalanceadas

Vamos adicionar os elementos 1, 2, 3, 4, 5, 6, 7 nessa ordem em uma árvore binária.

Árvores Binárias Desbalanceadas



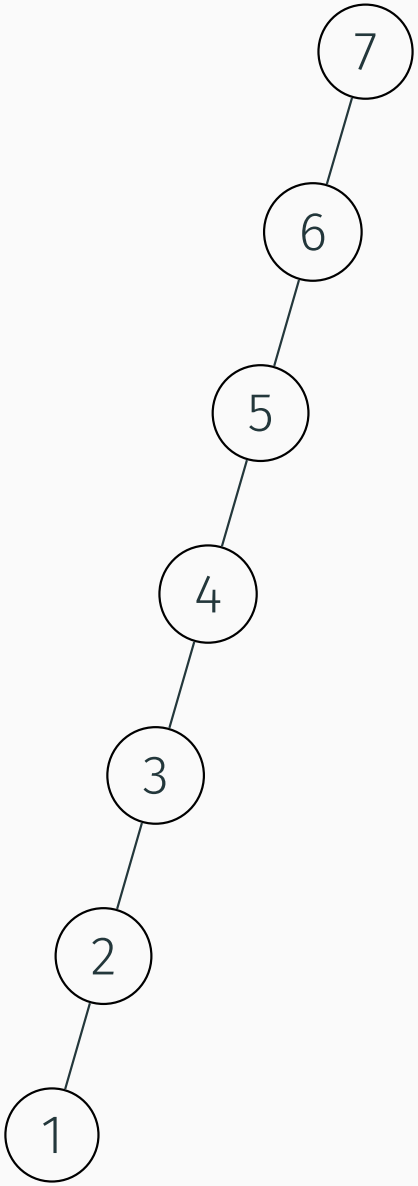
Árvores Binárias Desbalanceadas

A busca nessa árvore terá o mesmo custo assintótico que a busca sequencial em uma lista linear.

Árvores Binárias Desbalanceadas

Da mesma forma, vamos adicionar os elementos na ordem reversa!

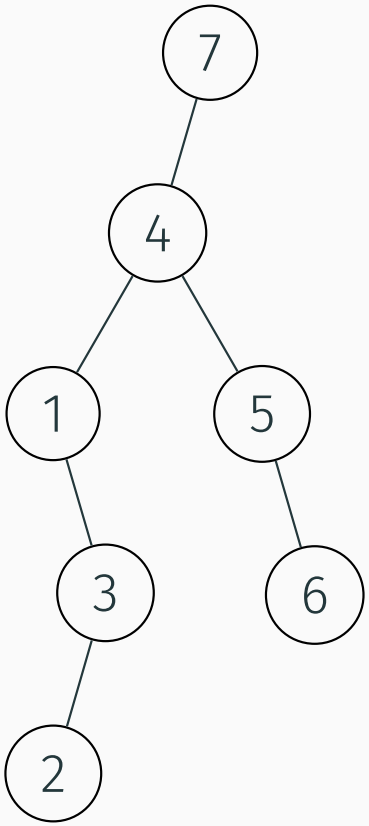
Árvores Binárias Desbalanceadas



Árvores Binárias Desbalanceadas

Finalmente, adicionem na ordem 7, 4, 1, 3, 2, 5, 6.

Árvores Binárias Desbalanceadas



Árvores Binárias Desbalanceadas

Um pouco melhor, mas ainda distante do que gostaríamos!

Árvores Binárias Desbalanceadas

As árvores binárias de busca possuem casos degenerados com altura igual a $n - 1$.

Nesses casos, as operações tem complexidade $O(n)$ no pior caso.

Árvores Binárias Desbalanceadas

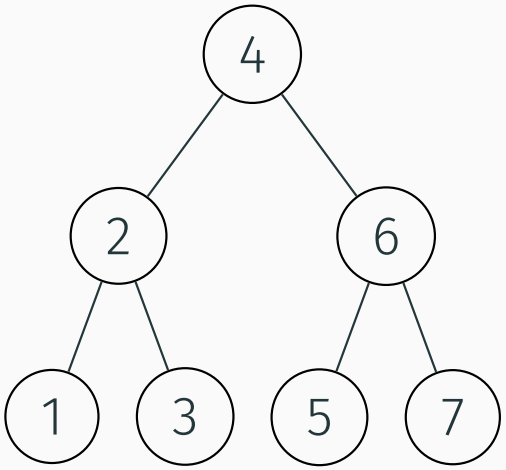
Uma árvore binária ótima é aquela que minimiza sua altura, ou seja, $\lfloor \log n + 1 \rfloor$.

Árvores Balanceadas

Uma **árvore balanceada** é uma árvore que automaticamente ajusta a disposição de seus nós para manter uma altura assintótica de $O(\log n)$.

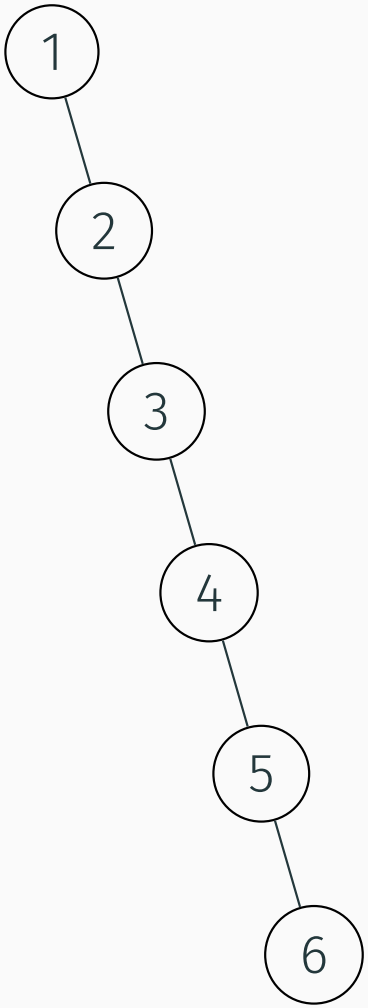
Exercício

A seguinte árvore está balanceada?



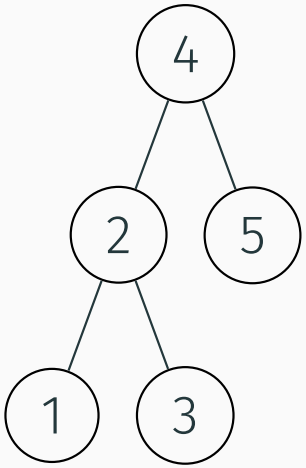
Exercício

A seguinte árvore está balanceada?



Exercício

A seguinte árvore está balanceada?



Árvores Balanceadas

- **Árvores AVL** (Adelson-Velskii e Landis).
- **Árvores Rubro-Negra**.
- **Árvores Splay**.
- **Árvores-B**.

Árvores AVL

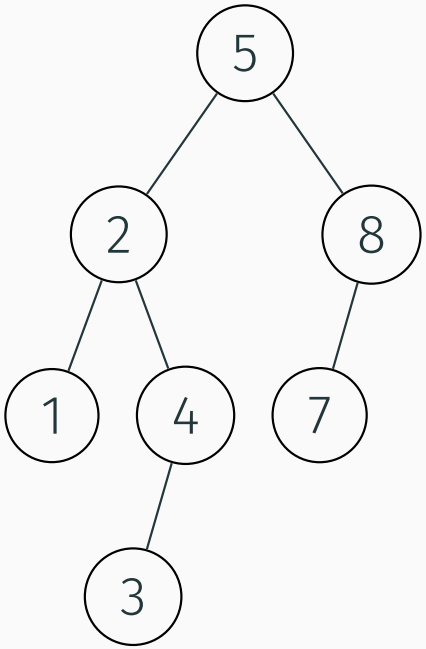
Árvore Balanceada dinâmica (se torna balanceada no momento da inserção/remoção).

Árvores AVL

1. Condição de balanceamento é de que a altura das sub-árvores de cada nó pode diferir em, no máximo, 1 unidade.
2. Toda sub-árvore é uma árvore AVL.

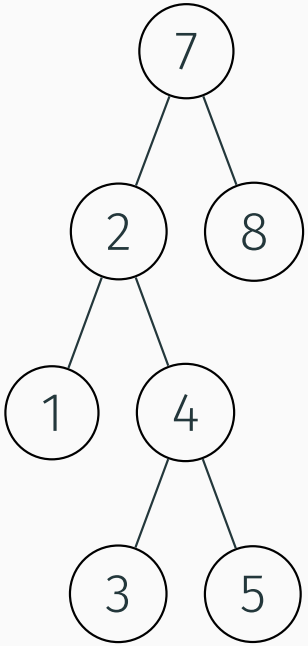
Exercício

Essa é uma árvore AVL?



Exercício

Essa é uma árvore AVL?



Altura da AVL

Assumindo $n(h)$ como o número mínimo de nós internos em uma árvore AVL de altura h.

Altura da AVL

Temos que $n(0) = 1, n(1) = 2$

Altura da AVL

Para $h > 1$ temos o nós raíz acrescido dos nós da árvore com altura $h - 1$ e $h - 2$ (estamos verificando o mínimo de nós).

$$n(h) = 1 + n(h-1) + n(h-2)$$

Altura da AVL

Sabemos que $n(h-1) > n(h-2)$ e, portanto, $n(h) > 2n(h-2)$, com isso temos que:

$$n(h) > 2n(h-2) > 4n(h-4) > 8n(h-6) > \ldots > 2^i n(h-2i)$$

Altura da AVL

Fazendo $2i = h$ temos:

$n(h) > 2^{h/2} n(0) = 2^{h/2}$

Altura da AVL

Resolvendo temos:

$h < 2 \log n(h) = O(logn)$

Struct da árvore AVL

Vamos alterar um pouco nossa estrutura de árvore para conter informação da altura do nó:

“‘C typedef struct tree tree;

struct tree { TREE_TYPE x; int height; struct tree * left; struct tree * right; };

Inserção na AVL

A inserção ocorre de forma similar a árvore binária de busca:

```
tree * insert (tree * t, tree * node) {

  if (t==NULL) return node;
  if (node->x < t->x)
      t->left  = insert(t->left , node);
  else if (node->x > t->x)
      t->right = insert(t->right, node);
  t->height = MAX( height(t->left), height(t->right) )
                + 1;

  return t;
}
```

Inserção na AVL

O rebalanceamento ocorre logo após uma chamada recursiva de insert, levando a 4 casos (na verdade dois casos, e os outros dois simétricos):

1. **Esquerda-esquerda:** o nó foi inserido a esquerda do filho esquerdo do nó atual.
2. **Direita-direita:** o nó foi inserido a direita do filho direito do nó atual.
3. **Esquerda-direita:** o nó foi inserido a direita do filho esquerdo do nó atual.
4. **Direita-esquerda:** o nó foi inserido a esquerda do filho direito do nó atual.

Esquerda-esquerda

Basta fazer uma rotação para a direita utilizando o filho da esquerda como pivô:

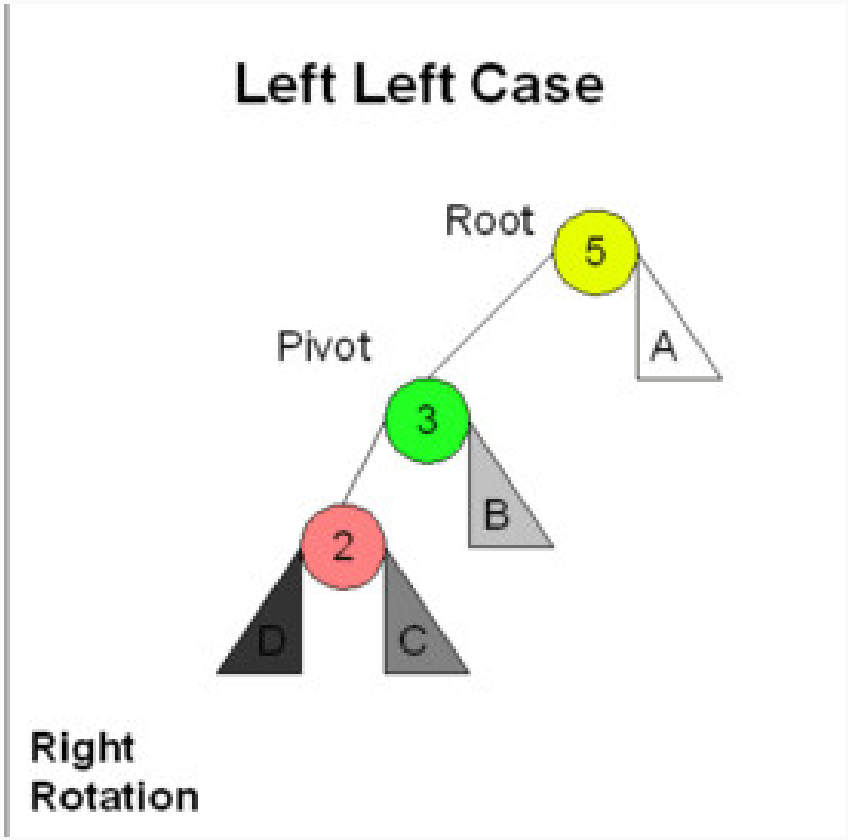


Figura 1: FONTE: https://medium.com/@randerson112358/avl-trees-a7b4f1fa2d1a

Esquerda-esquerda

Basta fazer uma rotação para a direita utilizando o filho da esquerda como pivô:

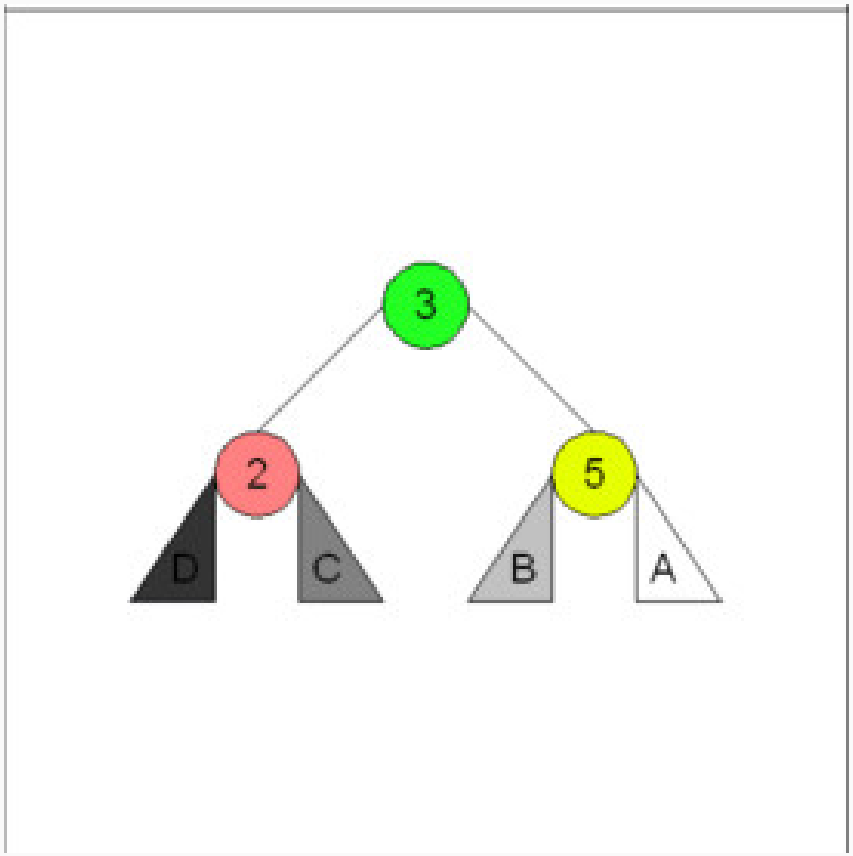


Figura 2: FONTE: https://medium.com/@randerson112358/avl-trees-a7b4f1fa2d1a

Direita-direita

Basta fazer uma rotação para a esquerda utilizando o filho da direita como pivô:

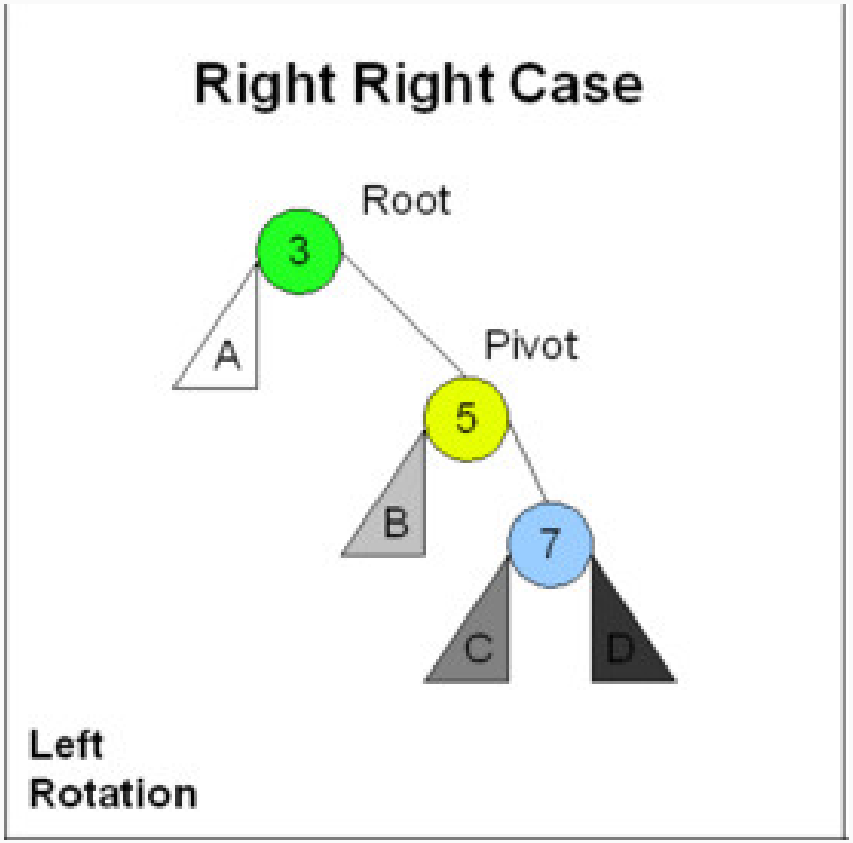


Figura 3: FONTE: https://medium.com/@randerson112358/avl-trees-a7b4f1fa2d1a

Direita-direita

Basta fazer uma rotação para a esquerda utilizando o filho da direita como pivô:

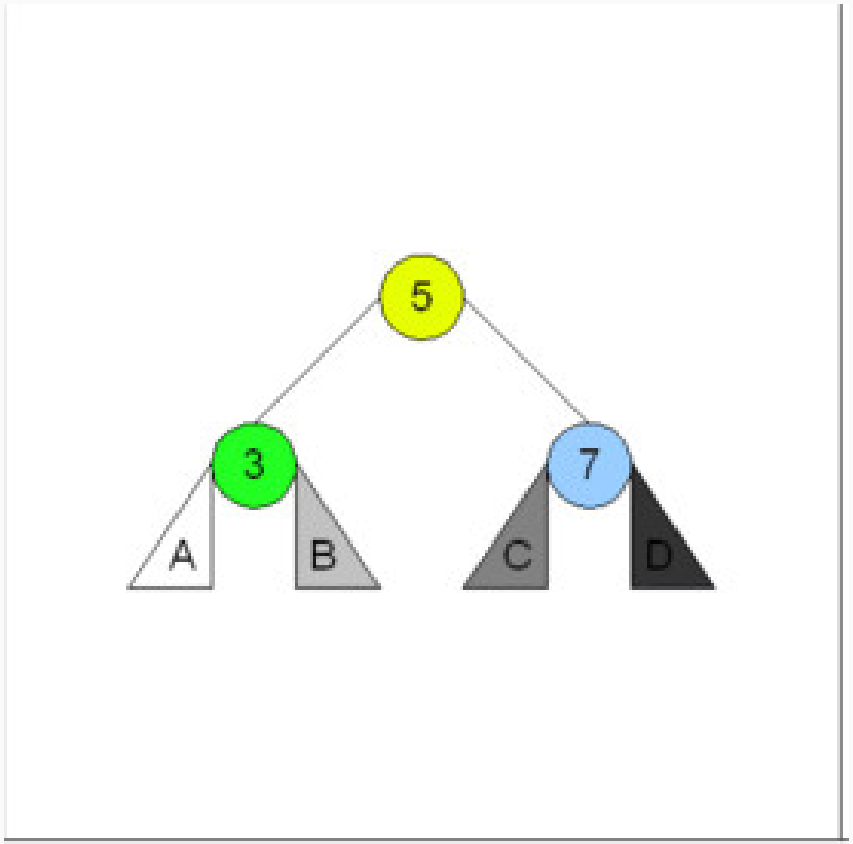


Figura 4: FONTE: https://medium.com/@randerson112358/avl-trees-a7b4f1fa2d1a

Esquerda-direita

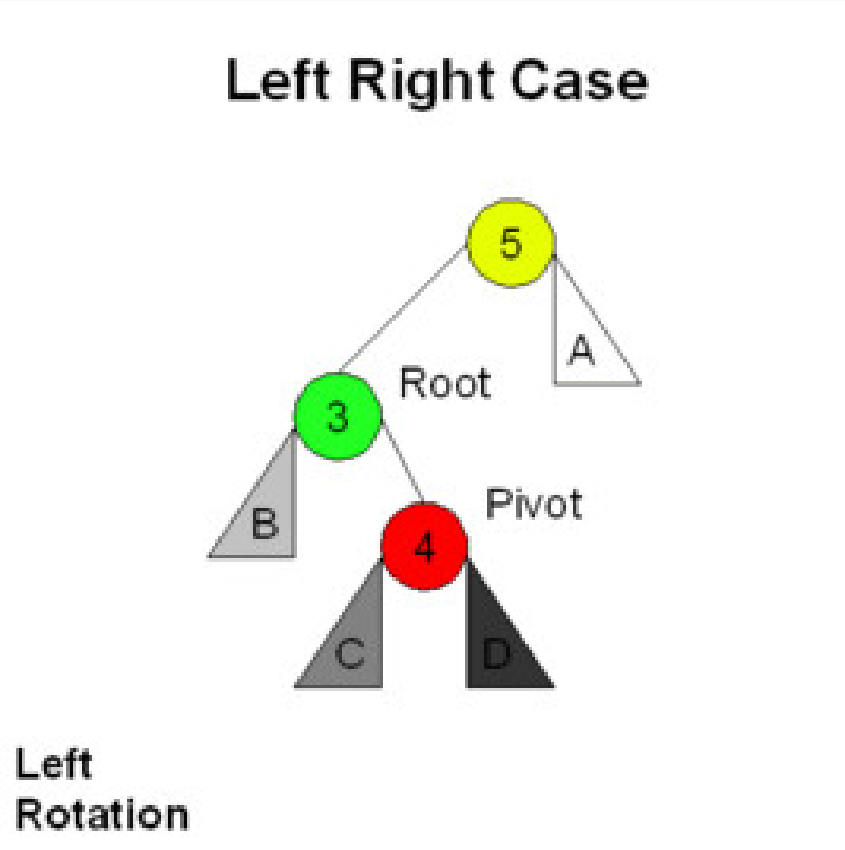


Figura 5: FONTE:

https://medium.com/@randerson112358/avl-trees-a7b4f1fa2d1a

Esquerda-direita

Primeiro fazemos uma rotação para a esquerda:

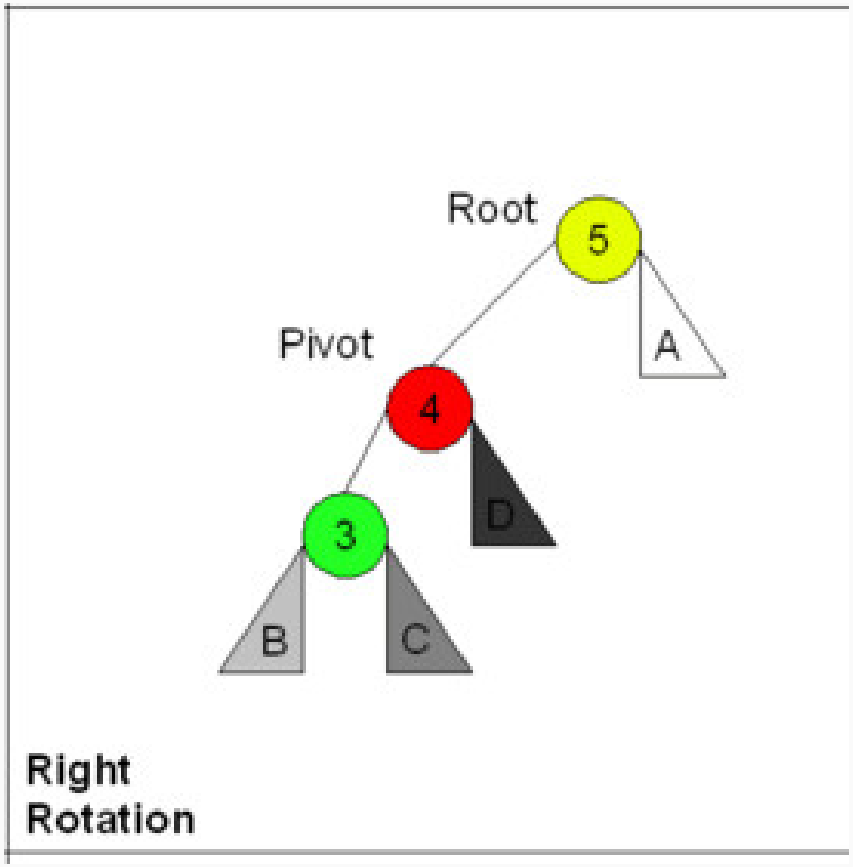


Figura 6: FONTE:

https://medium.com/@randerson112358/avl-trees-a7b4f1fa2d1a

Esquerda-direita

Em seguida uma rotação para a direita:

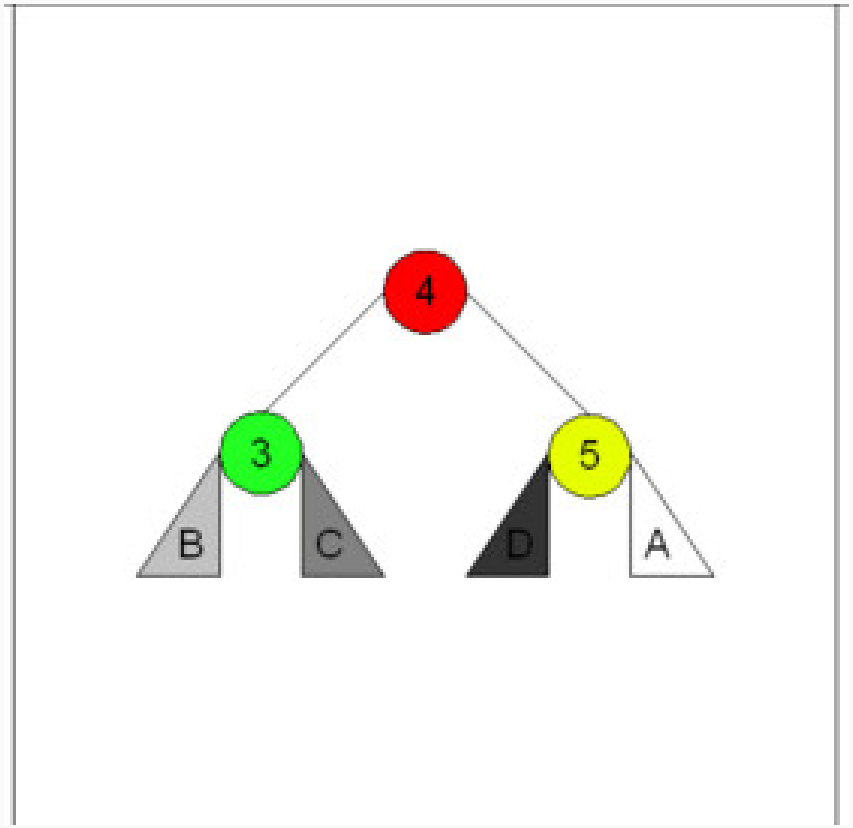


Figura 7: FONTE:

https://medium.com/@randerson112358/avl-trees-a7b4f1fa2d1a

Direita-esquerda

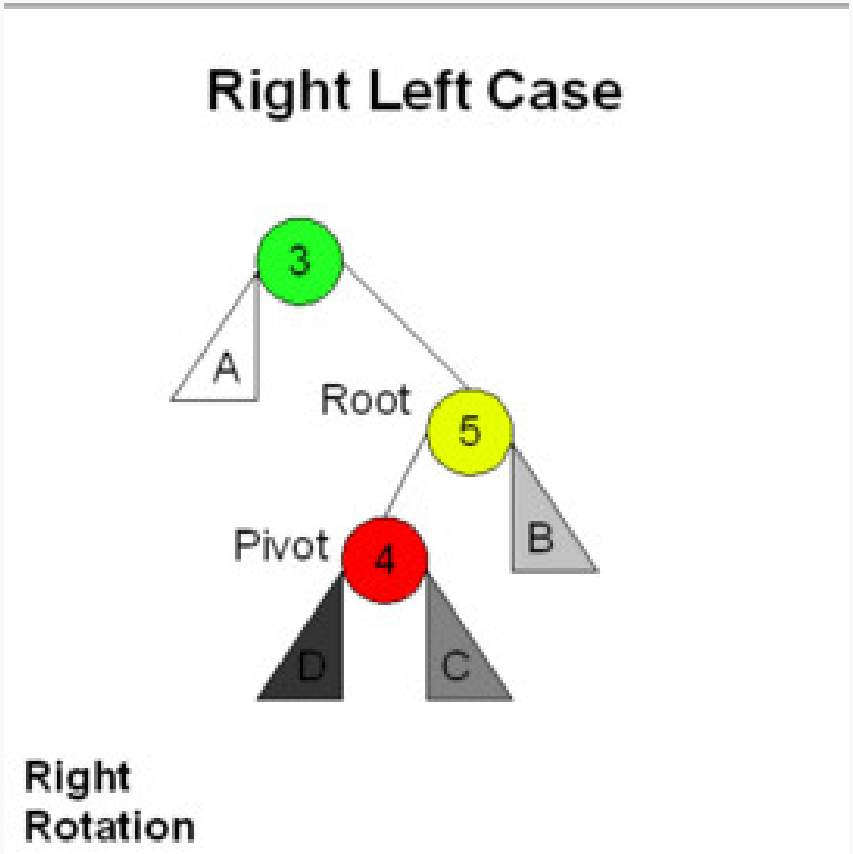


Figura 8: FONTE:

https://medium.com/@randerson112358/avl-trees-a7b4f1fa2d1a

Direita-esquerda

Primeiro fazemos uma rotação para a direita:

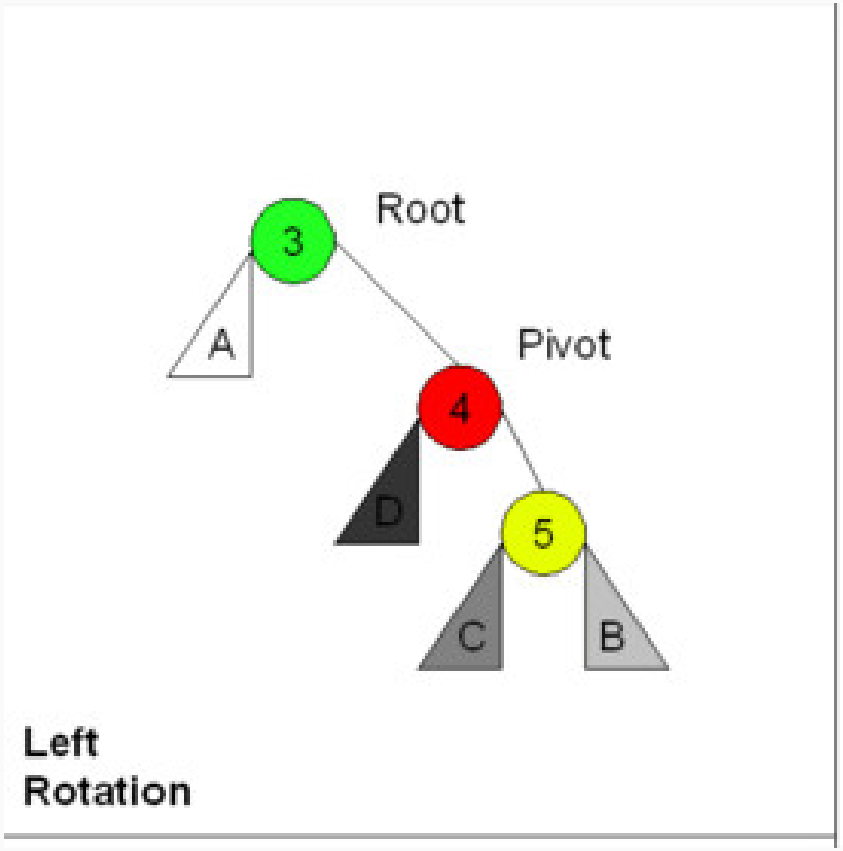


Figura 9: FONTE:

https://medium.com/@randerson112358/avl-trees-a7b4f1fa2d1a

Direita-esquerda

Em seguida uma rotação para a esquerda:

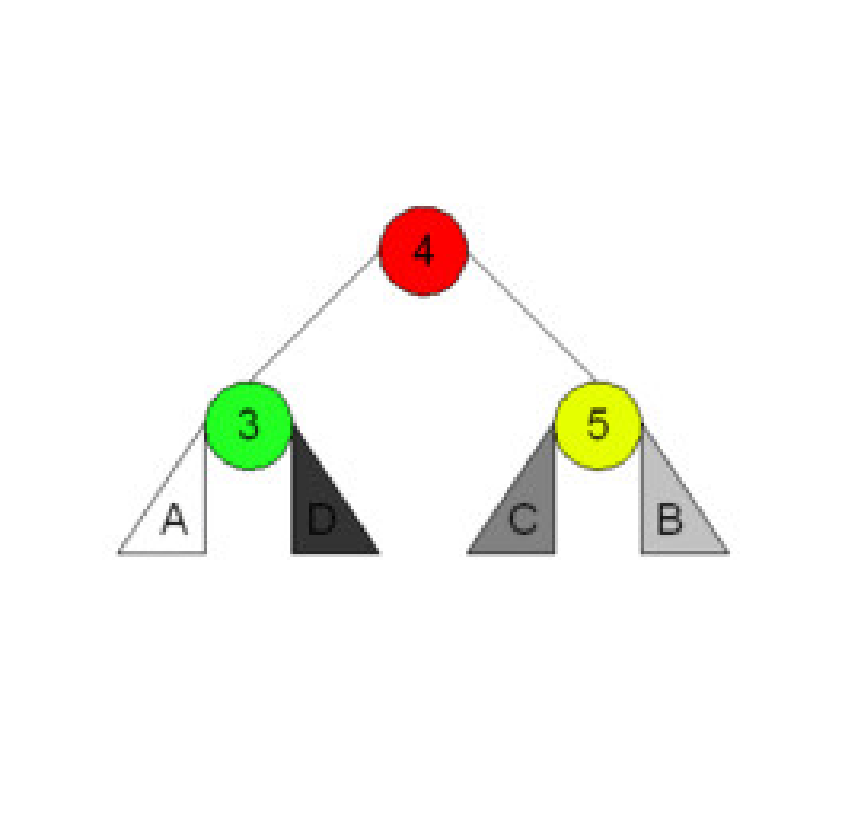


Figura 10: FONTE:

https://medium.com/@randerson112358/avl-trees-a7b4f1fa2d1a

Inserção na AVL

Dessa forma temos que modificar a inserção para realizar as rotações necessárias.

Inserção na AVL

```
tree * insert (tree * t, tree * node) {

  if (t==NULL) return node;

  if (node->x < t->x)
  {
    t->left  = insert(t->left , node);
    if (height(t->left) - height(t->right) == 2)
    {
      if (node->x < t->left->x) t = left_left(t);
      else                      t = left_right(t);
    }
  }
```

Inserção na AVL

```
  else if (node->x > t->x) {
    t->right = insert(t->right, node);
    if (height(t->right) - height(t->left) == 2)
    {
      if (node->x > t->right->x) t = right_right(t);
      else                       t = right_left(t);
    }
  }

  t->height = calc_height(t);

  return t;
}
```

Rotação para a Direita

```
tree * left_left( tree * root ) {
    tree * pivot = root->left;
    root->left = pivot->right;
    pivot->right = root;

    root->height = calc_height(root);
    pivot->height = calc_height(pivot);

    return pivot;
}
```

Rotação para a Esquerda

```
tree * right_right( tree * root ) {
    tree * pivot = root->right;
    root->right = pivot->left;
    pivot->left = root;

    root->height = calc_height(root);
    pivot->height = calc_height(pivot);

    return pivot;
}
```

Rotação Esquerda-Direita

```
tree * left_right( tree * t ) {
    t->left = right_right( t->left );
    return left_left( t );
}
```

Rotação Direita-Esquerda

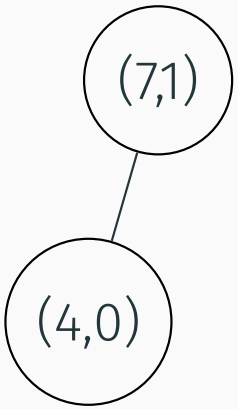
```
tree * right_left( tree * t ) {
    t->right = left_left( t->right );
    return right_right( t );
}
```

Passo a passo

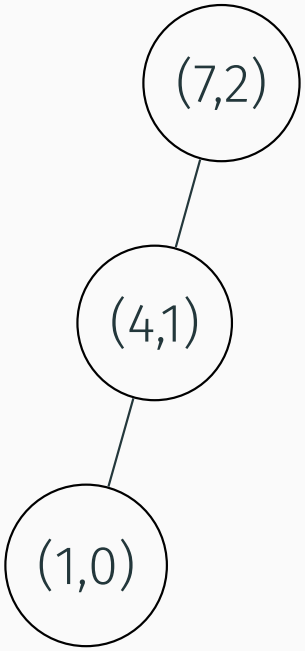
Vamos adicionar nós em uma AVL na ordem 7, 4, 1, 3, 2, 5, 6.

Passo a passo

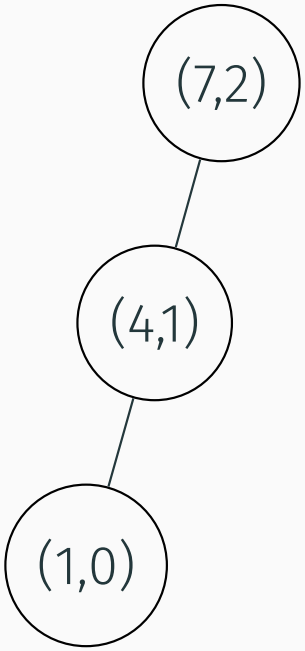
Passo a passo



Passo a passo

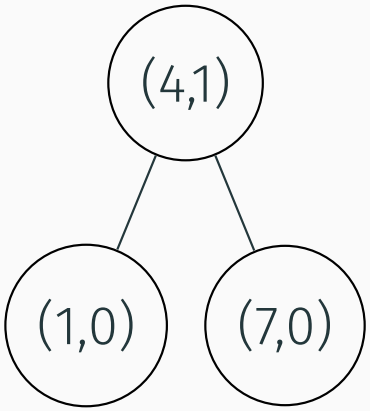


Passo a passo

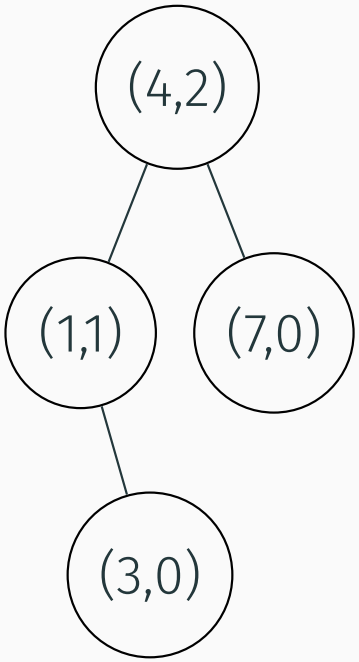


Passo a passo

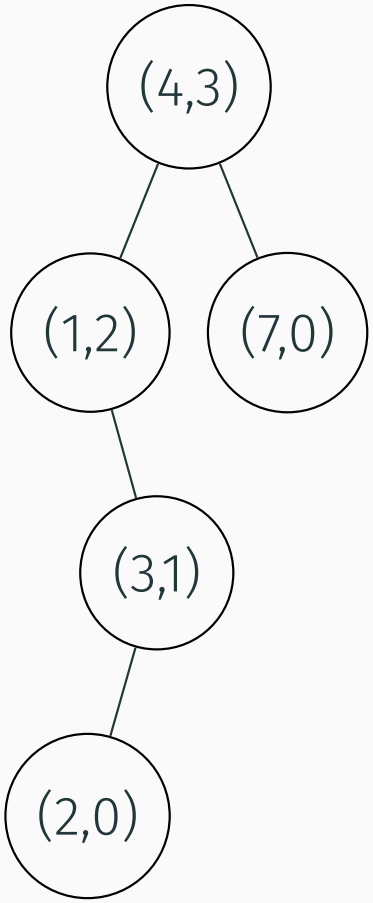
Rotação para direita:



Passo a passo

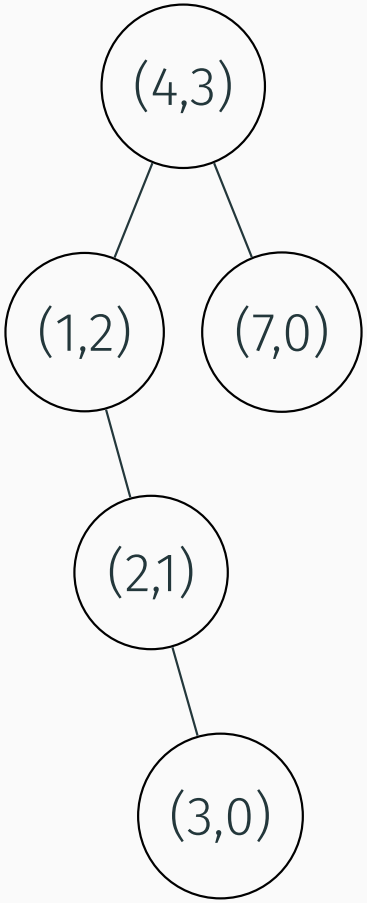


Passo a passo



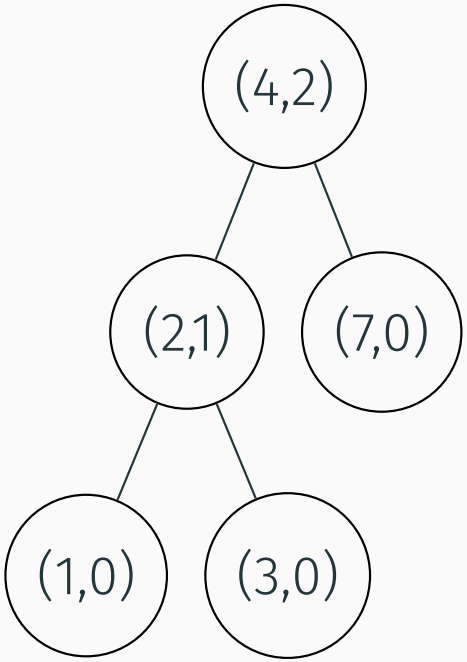
Passo a passo

Rotaciona filho direita para direita:

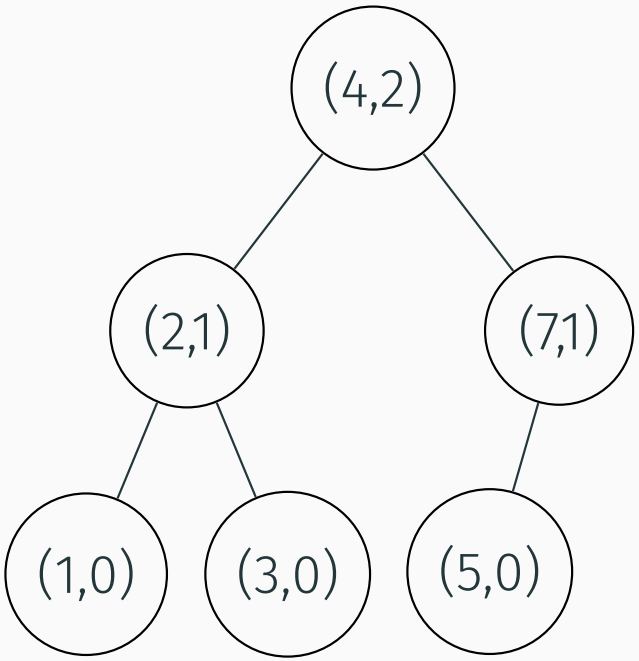


Passo a passo

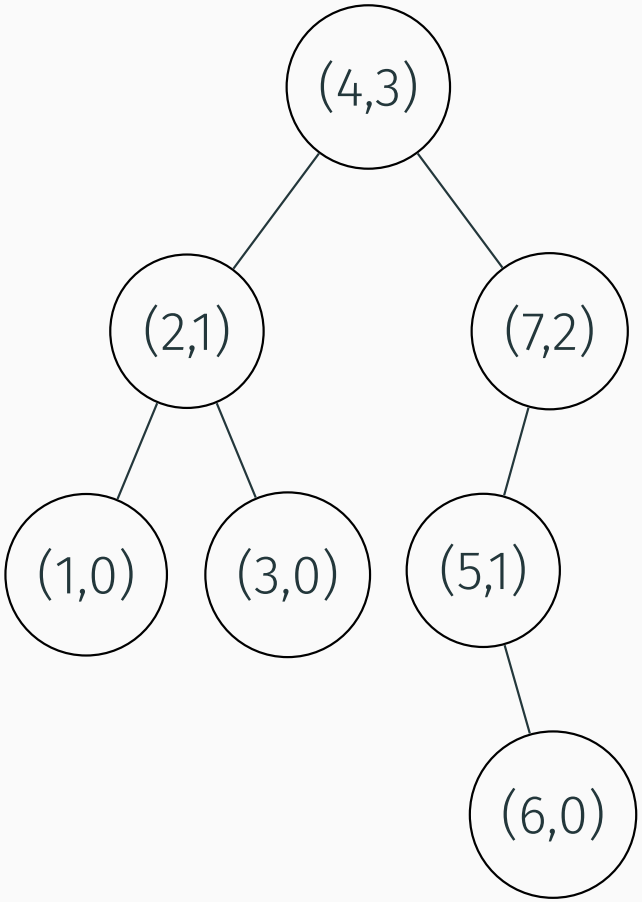
Rotaciona para esquerda:



Passo a passo

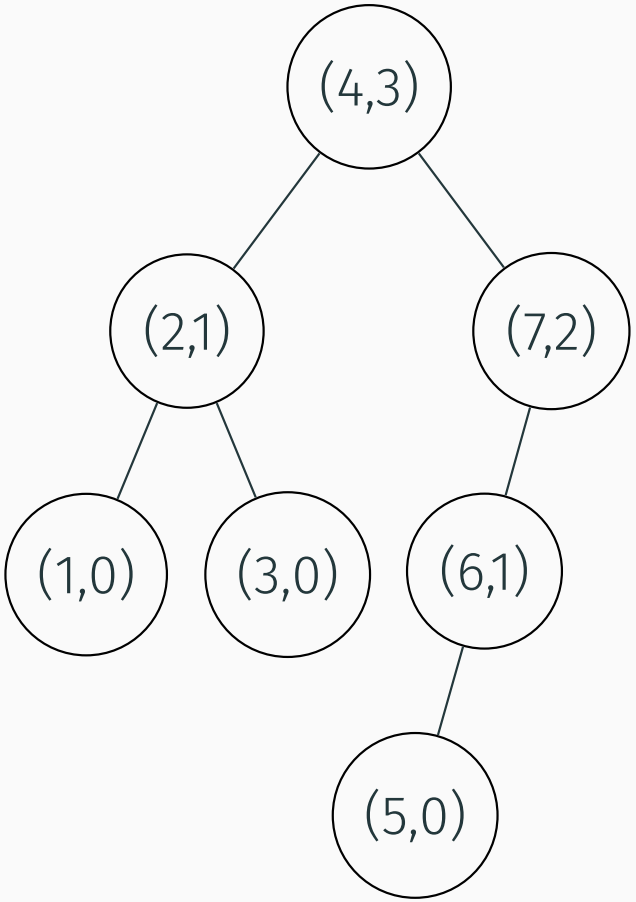


Passo a passo



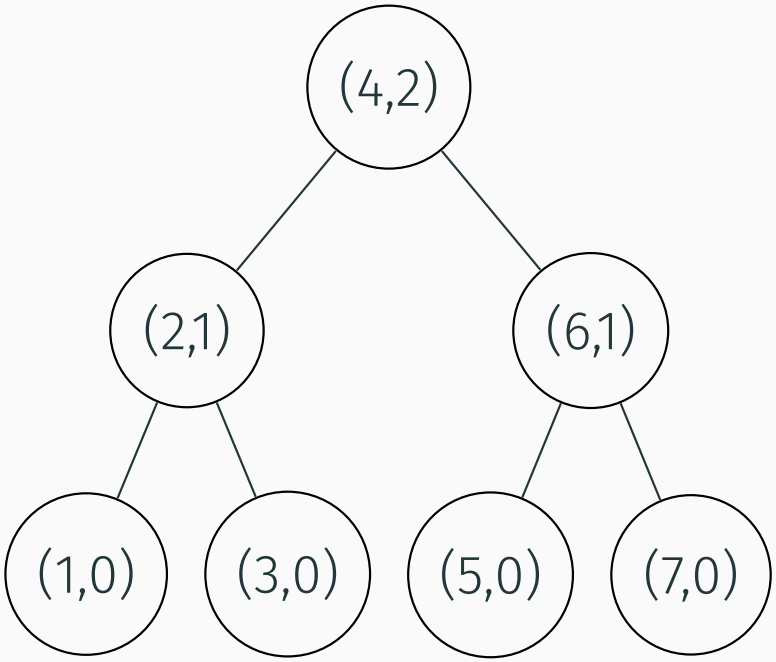
Passo a passo

Rotaciona filho da esquerda para esquerda:



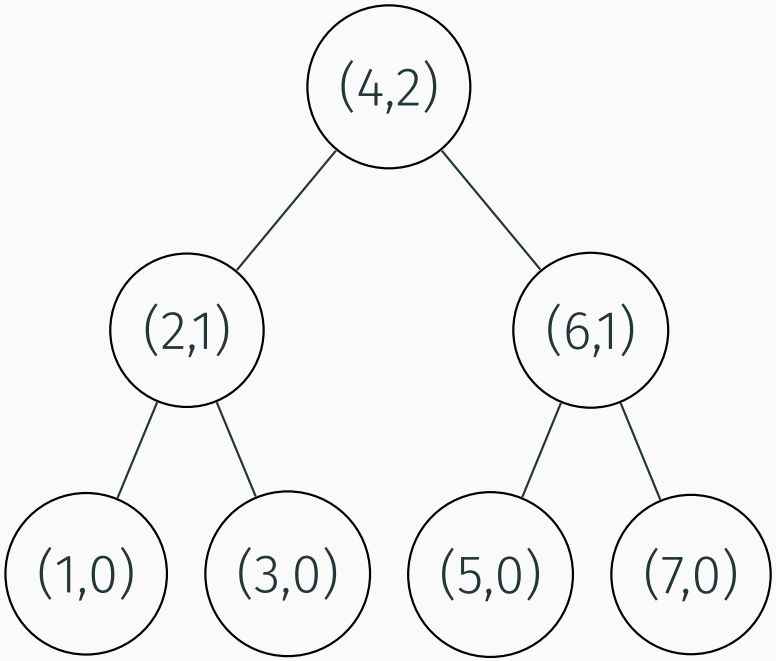
Passo a passo

Rotaciona para direita:



Passo a passo

Árvore de 7 nós com altura 2.



Vantagens da AVL

- Busca é sempre $O(\log n)$ pois a árvore está sempre balanceada.
- Inserções e Remoções são sempre $O(\log n)$.
- O balanceamento adiciona um fator constante nos custos das operações.

Desvantagens da AVL

- Operações com custo assintótico baixo, mas ainda assim que custam tempo computacional.
- O fator constante da remoção pode ser grande demais.

Outras árvores balanceadas

- **Árvore Rubro-Negra:** necessita apenas uma rotação por operação, mas não tão balanceada do que a AVL.
- **Árvores-B:** geralmente utilizada para grandes quantidades de dados ou quando esses residem em disco.
- **Splay:** atualiza o balanceamento da árvore durante o acesso ao elemento, operações amortizadas $O(\log n)$.

Próxima Aula

Aprenderemos sobre algoritmos de **ordenação** para estruturas lineares.