



Universidade Federal do Rio de Janeiro

Trabalho de Organização de Dados 2

Grupo:

DANILO SILVA DE CARVALHO – 107390588

ROBERTA SANTOS LOPES – 107362886

TAÍSA LOPES MARTINS – 107362828

VINICIUS BASTOS BITTENCOURT – 107362983



Árvore AVL

Motivação:

Por que foi criada essa estrutura?

Vamos analisar uma estrutura de dados muito comum: a Árvore Binária de Busca

Árvore Binária de Busca

Estrutura de dados onde todos os nós são chaves, todos nós à esquerda contêm uma sub-árvore com os valores menores ao nó raiz da sub-árvore e todos os nós da sub-árvore à direita contêm somente valores maiores ao nó raiz desta última sub-árvore.

Árvore Binária de Busca

Tal fato nos permite fazer buscas, inserções e remoções e exibir os resultados de forma ordenada com muita flexibilidade.

Árvore Binária de Busca

Exemplo:

Vamos inserir os números 5, 8, 3, 2, 4, 6, 7, 1, 9, nessa ordem.

Árvore Binária de Busca

Exemplo:

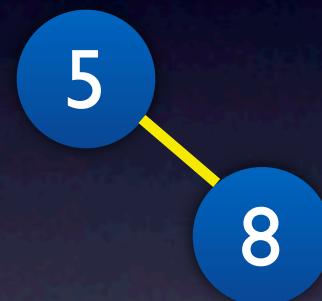
Vamos inserir os números 5, 8, 3, 2, 4, 6, 7, 1, 9, nessa ordem.

5

Árvore Binária de Busca

Exemplo:

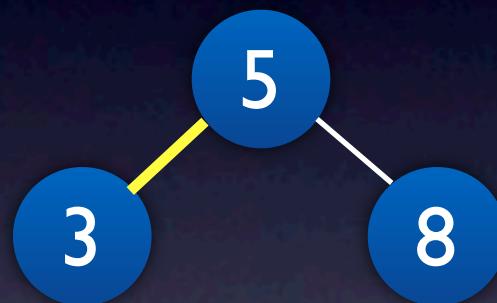
Vamos inserir os números 5, 8, 3, 2, 4, 6, 7, 1, 9, nessa ordem.



Árvore Binária de Busca

Exemplo:

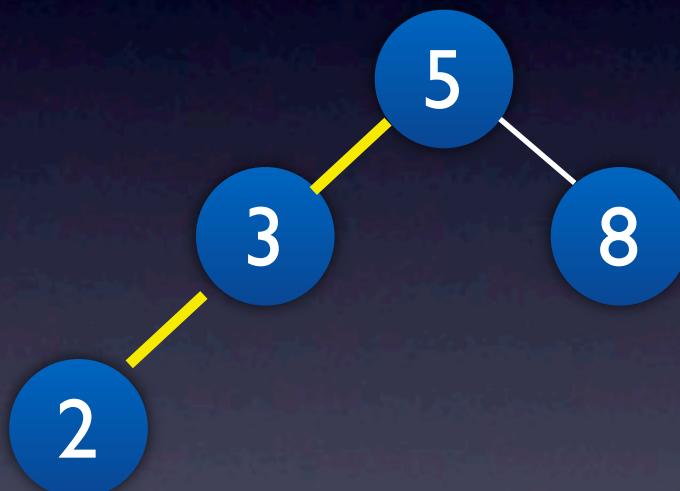
Vamos inserir os números 5, 8, 3, 2, 4, 6, 7, 1, 9, nessa ordem.



Árvore Binária de Busca

Exemplo:

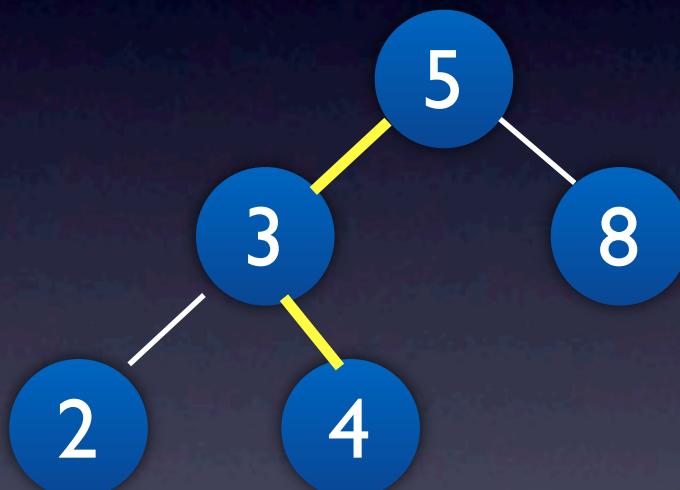
Vamos inserir os números 5, 8, 3, 2, 4, 6, 7, 1, 9, nessa ordem.



Árvore Binária de Busca

Exemplo:

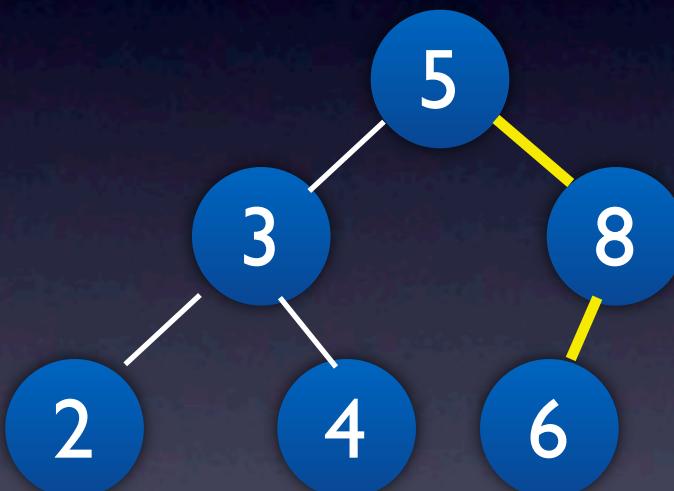
Vamos inserir os números 5, 8, 3, 2, 4, 6, 7, 1, 9, nessa ordem.



Árvore Binária de Busca

Exemplo:

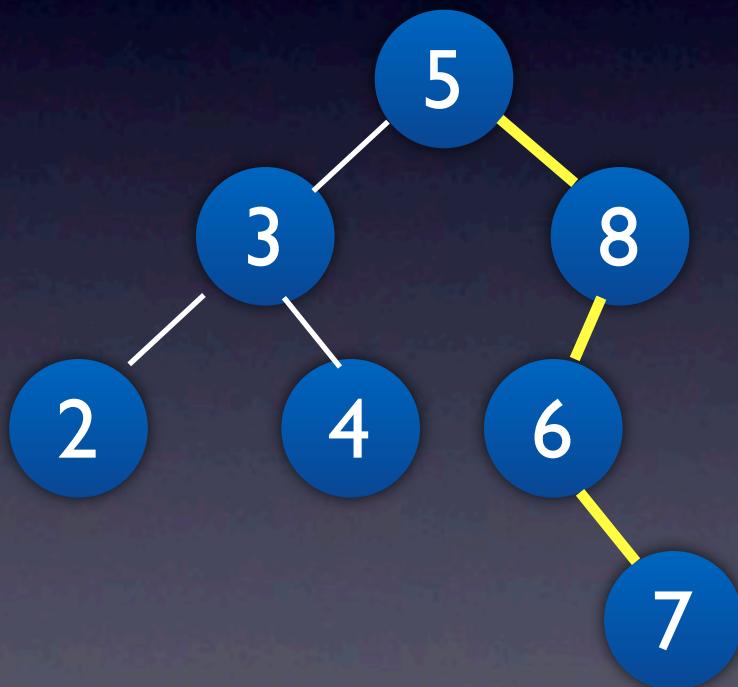
Vamos inserir os números 5, 8, 3, 2, 4, 6, 7, 1, 9, nessa ordem.



Árvore Binária de Busca

Exemplo:

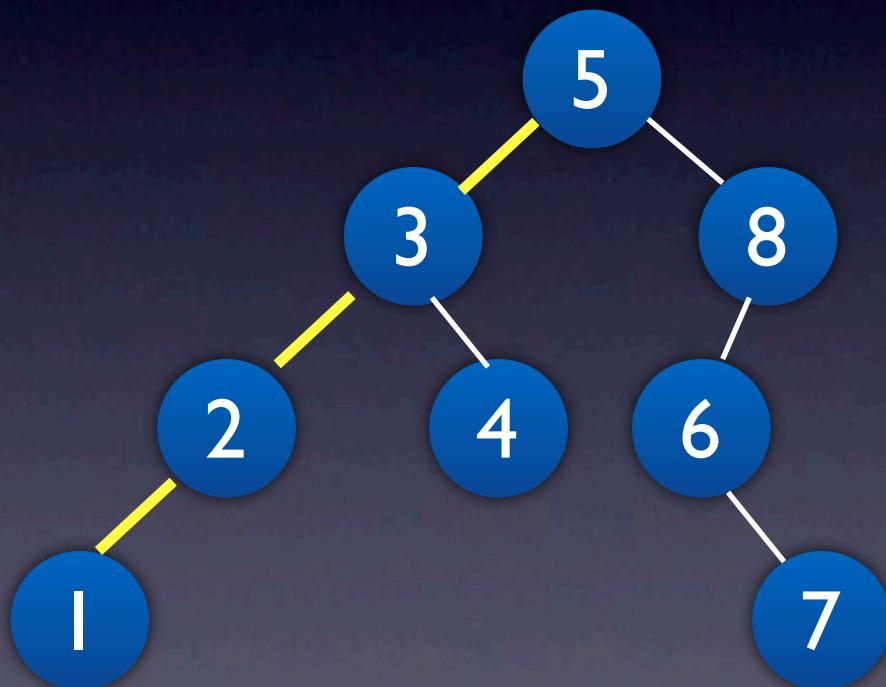
Vamos inserir os números 5, 8, 3, 2, 4, 6, 7, 1, 9, nessa ordem.



Árvore Binária de Busca

Exemplo:

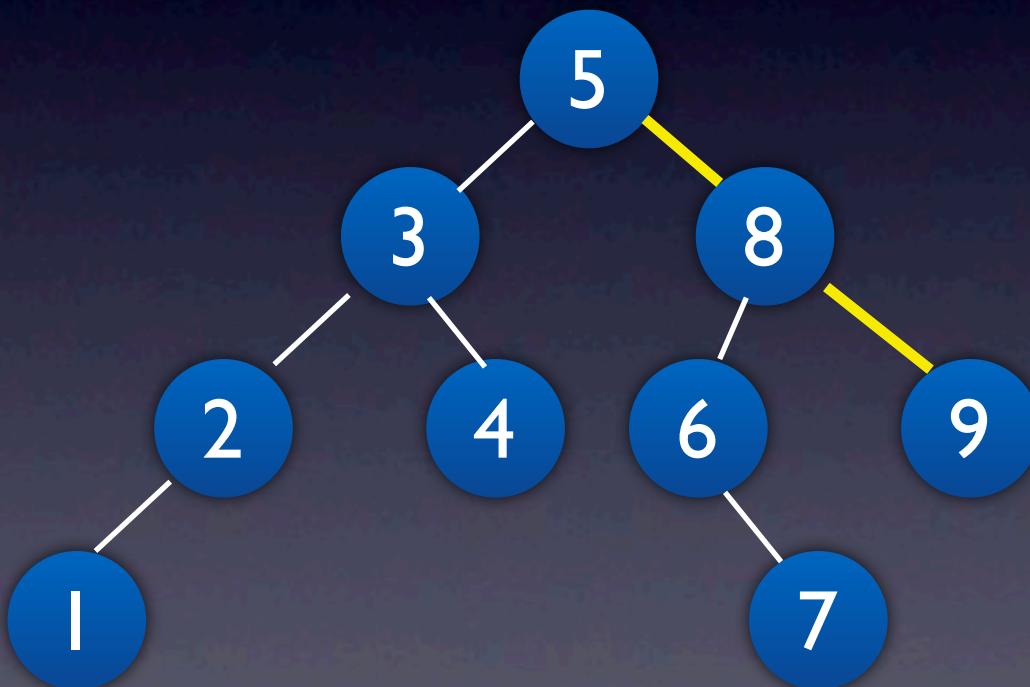
Vamos inserir os números 5, 8, 3, 2, 4, 6, 7, 1, 9, nessa ordem.



Árvore Binária de Busca

Exemplo:

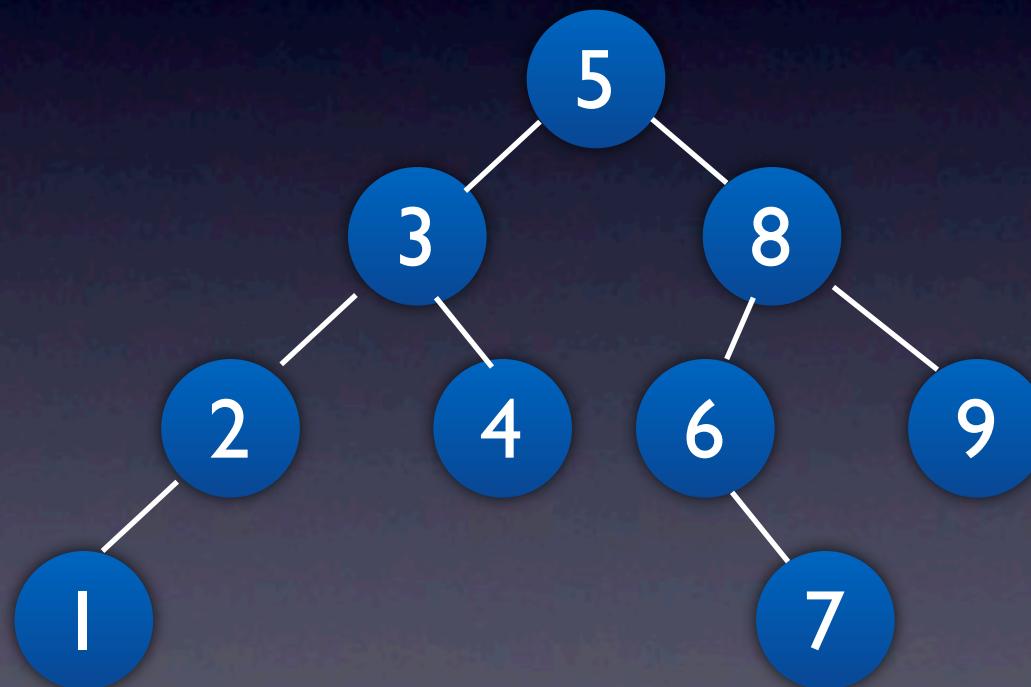
Vamos inserir os números 5, 8, 3, 2, 4, 6, 7, 1, 9, nessa ordem.



Árvore Binária de Busca

Exemplo:

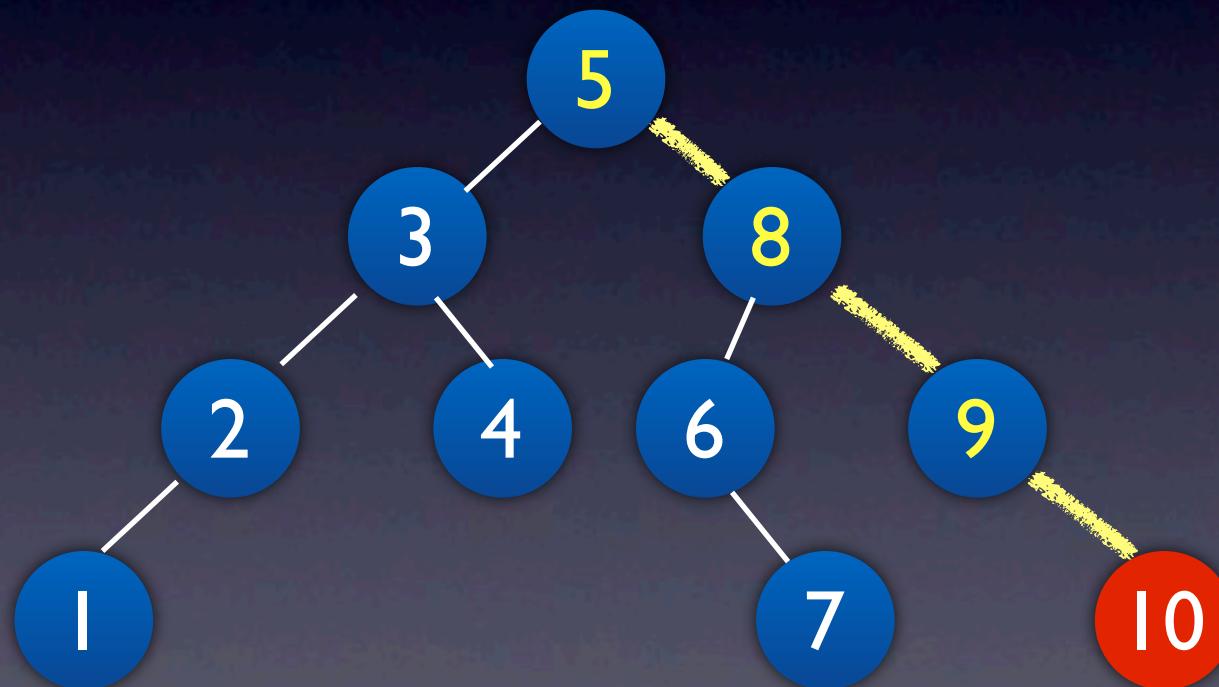
Vamos agora inserir a chave de valor 10.



Árvore Binária de Busca

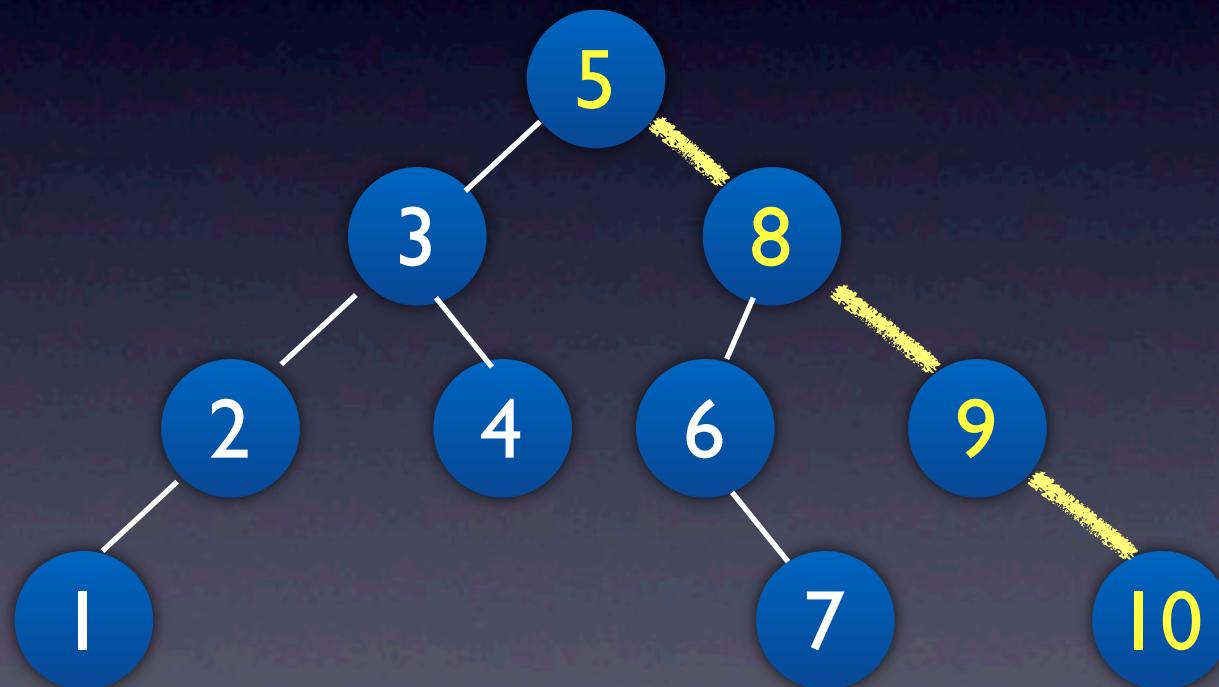
Exemplo:

Vamos agora inserir a chave de valor 10.



Árvore Binária de Busca

Note que tivemos que fazer um pequeno percurso, com poucas comparações e a inserção ocorreu de forma bem rápida.



Árvore Binária de Busca

Agora vamos inserir os mesmos números, porém
em outra ordem: 2,1,3,4,5,6,7,8,9

2



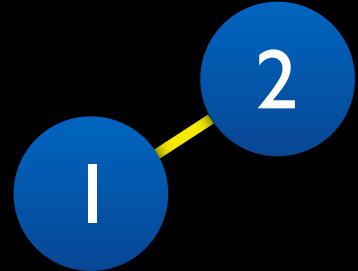


Figure 1: A graph with two nodes. Node 1 is at the bottom left and node 2 is at the top right.

The graph has one edge connecting node 1 and node 2.

The edge is colored yellow.

The nodes are represented by blue circles.

The numbers 1 and 2 are displayed inside their respective nodes.

The overall layout is simple and clear, illustrating a basic graph structure.

The nodes are positioned at opposite corners of the frame, creating a visual balance.

The edge connects the two nodes directly, representing a direct relationship between them.

The colors used (blue for nodes, yellow for edge) provide good contrast against the white background.

The font used for the numbers is a clean, sans-serif style.

The size of the nodes is proportional to the size of the numbers inside them.

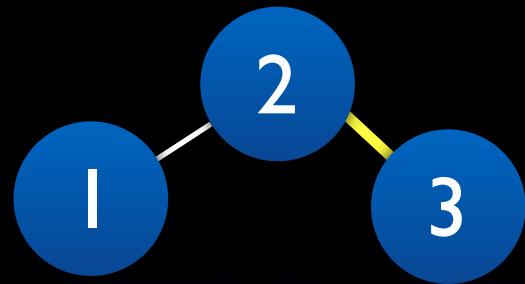
The overall aesthetic is minimalist and modern.

The diagram effectively communicates the basic components of a graph: nodes and edges.

It serves as a clear visual representation for concepts like connectivity and network structures.

The simplicity of the design allows it to be easily understood and reproduced.

It is a well-constructed and informative figure.



• **Path**: A sequence of edges connecting vertices.

• **Simple Path**: A path that does not contain any vertex more than once.

• **Cycle**: A path that starts and ends at the same vertex.

• **Simple Cycle**: A cycle that does not contain any vertex more than once.

• **Length**: The number of edges in a path.

• **Weighted Graph**: A graph where each edge has a numerical value associated with it.

• **Shortest Path**: The path between two vertices with the minimum weight.

• **Spanning Tree**: A tree that contains all the vertices of a graph.

• **Minimum Spanning Tree**: A spanning tree with the minimum weight.

• **Graph Traversal**: A process of visiting every vertex in a graph.

• **Breadth-First Search (BFS)**: A traversal algorithm that explores all the vertices at the same depth before moving to the next depth level.

• **Depth-First Search (DFS)**: A traversal algorithm that explores as far as possible along each branch before backtracking.

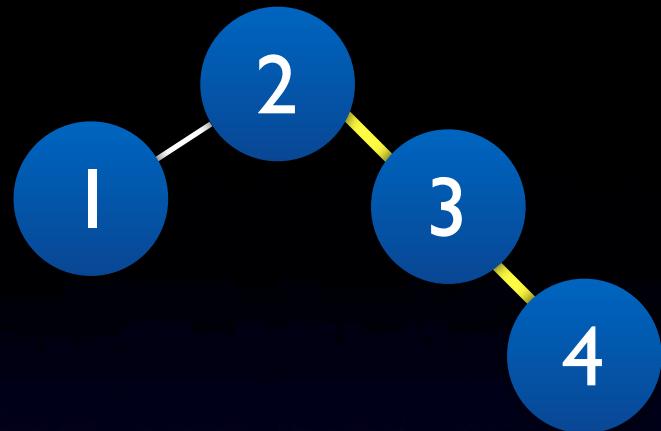
• **Topological Sort**: An ordering of vertices such that for every directed edge $v \rightarrow w$, vertex v appears before vertex w .

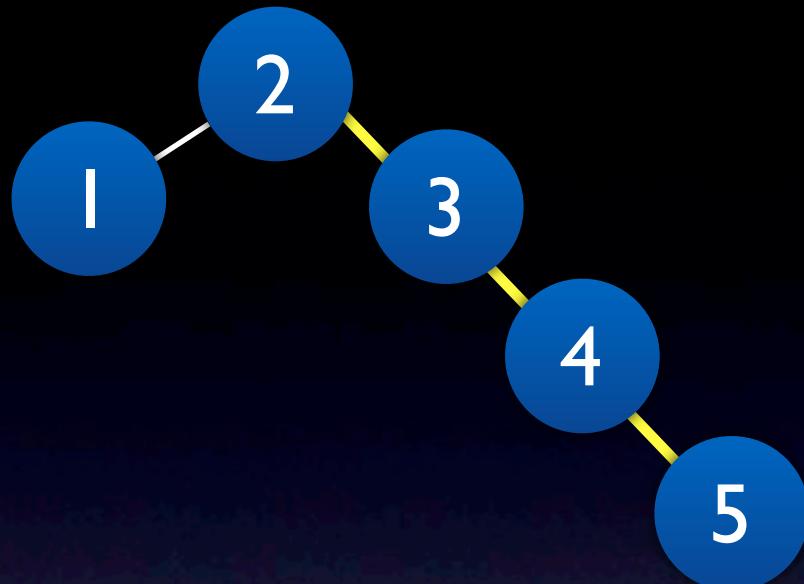
• **Strongly Connected Components (SCCs)**: Subgraphs where every vertex is reachable from every other vertex.

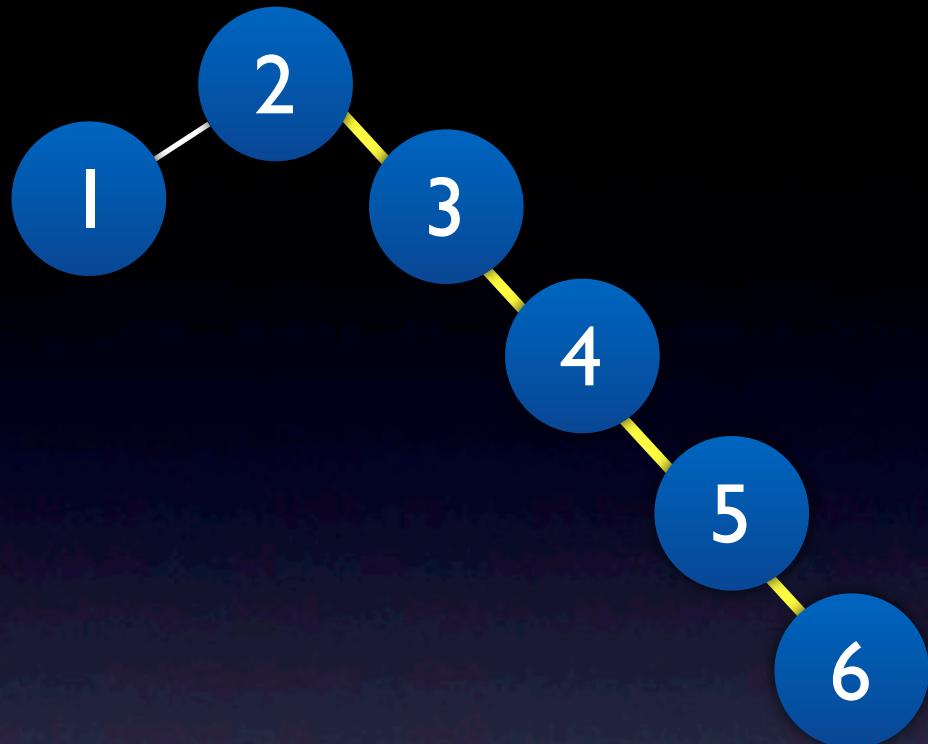
• **Weakly Connected Components (WCCs)**: Subgraphs where every vertex in one component is reachable from every other vertex in the same component.

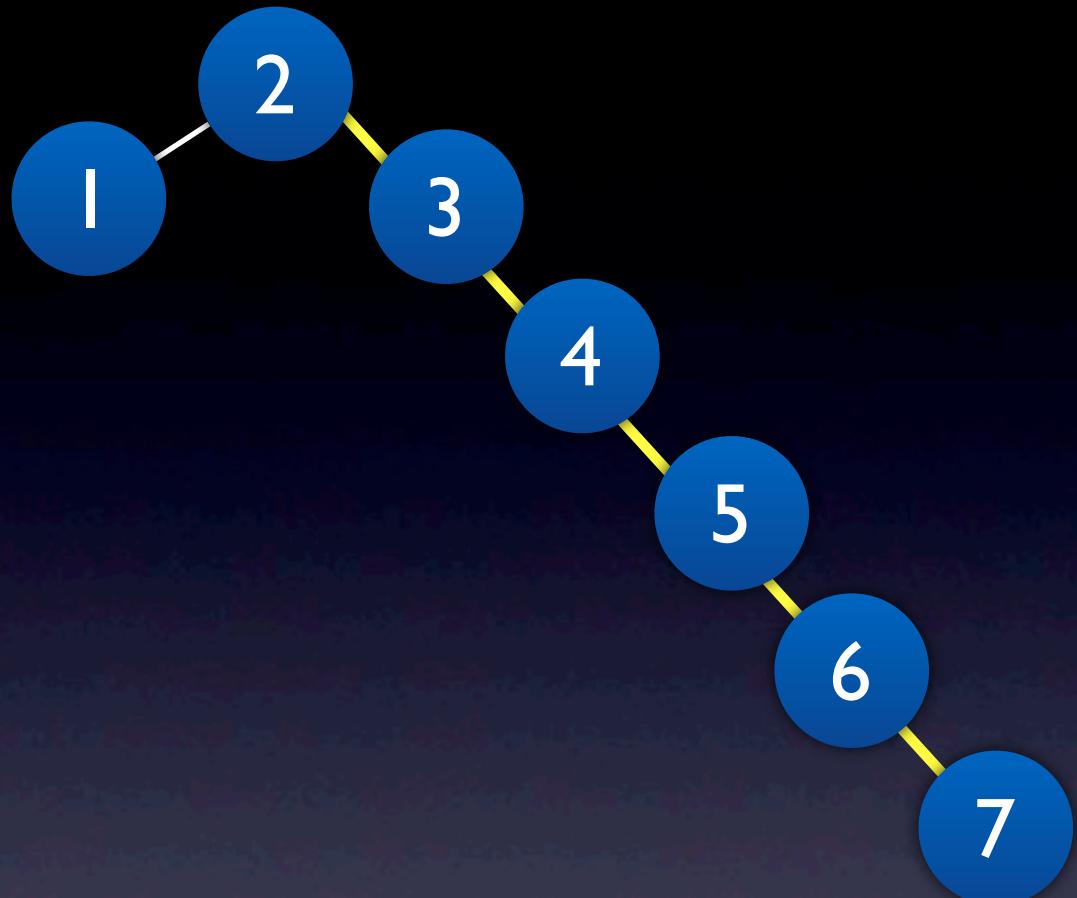
• **Articulation Points**: Vertices whose removal increases the number of connected components.

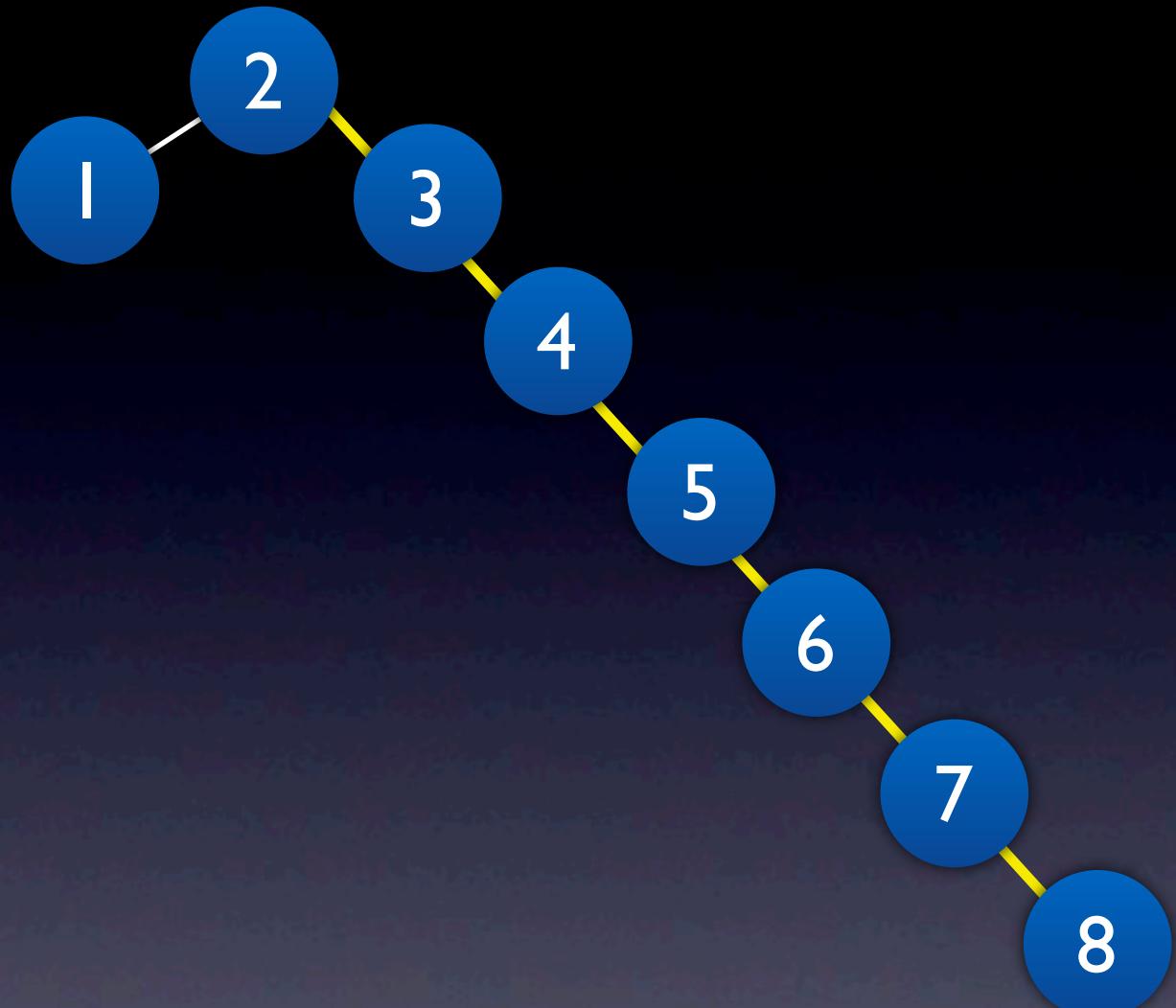
• **Bridge**: An edge whose removal increases the number of connected components.

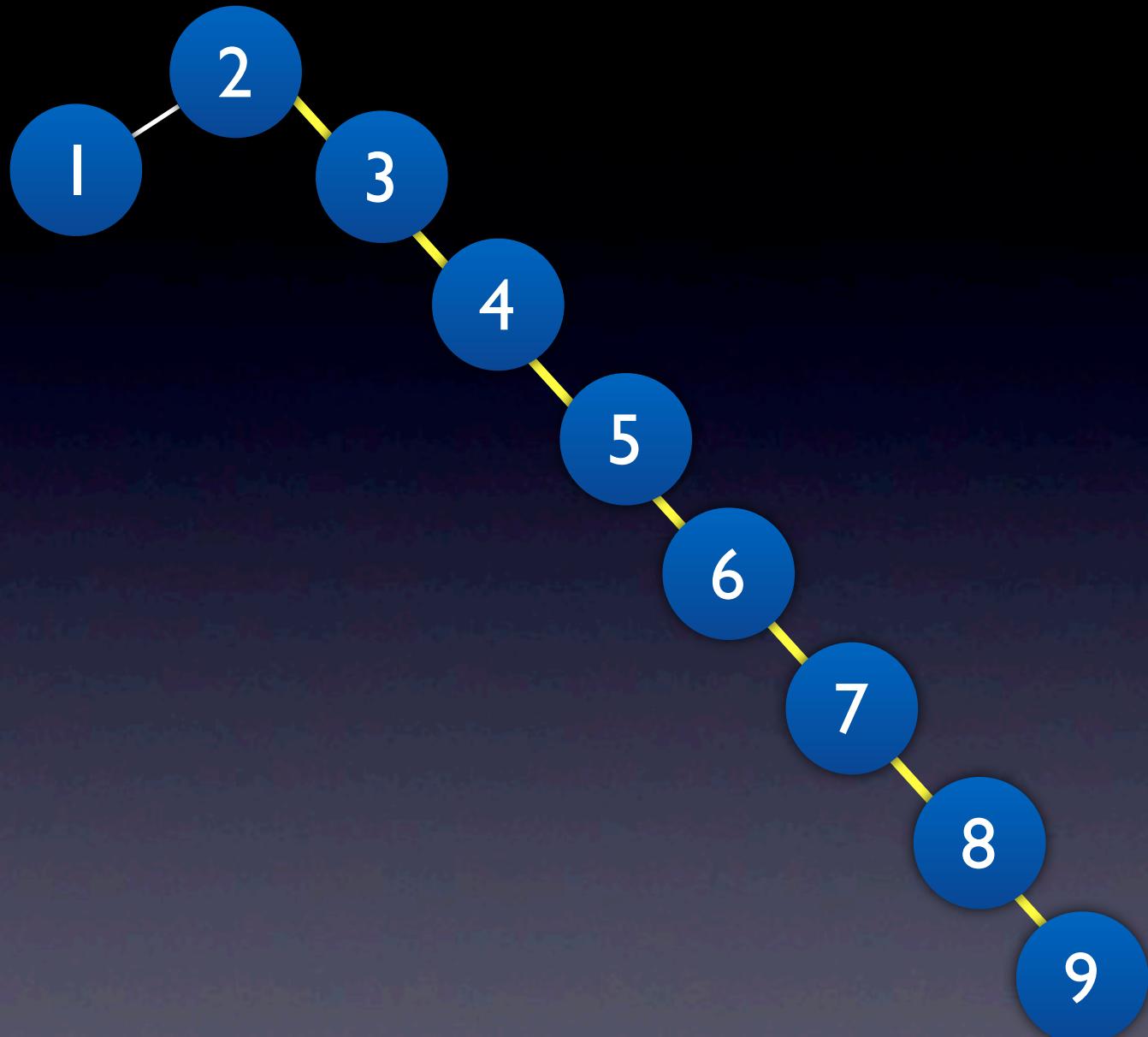


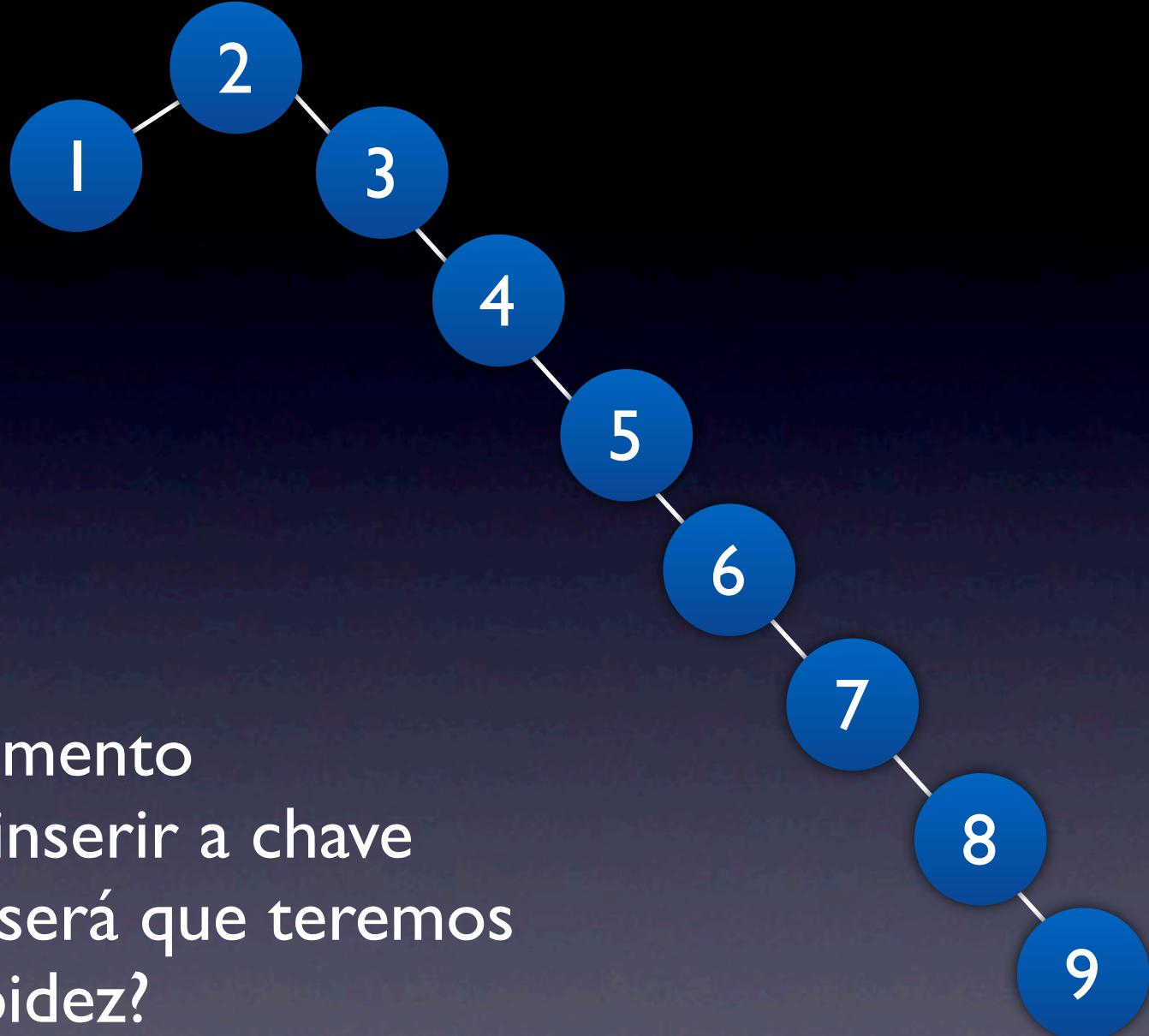




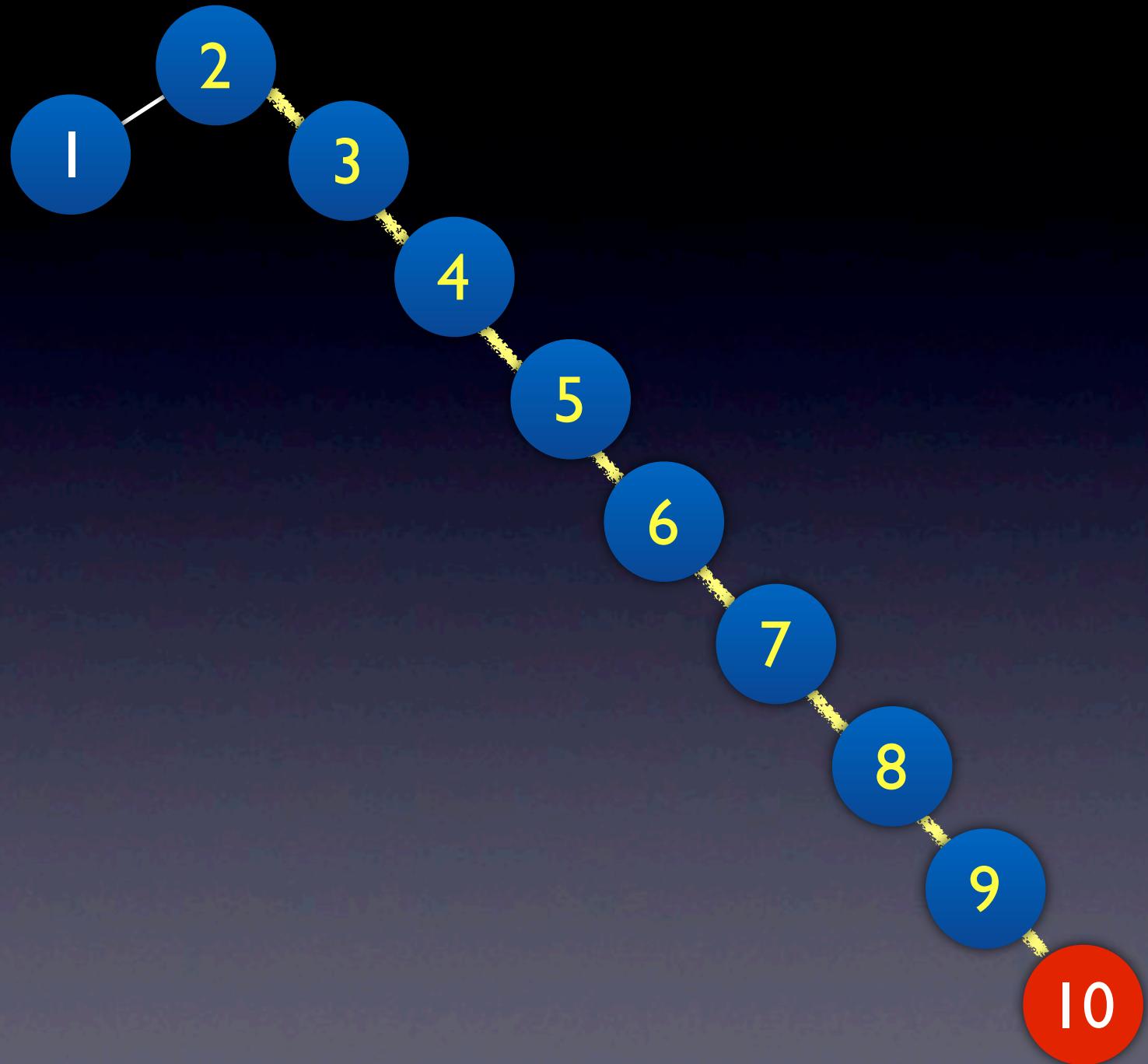


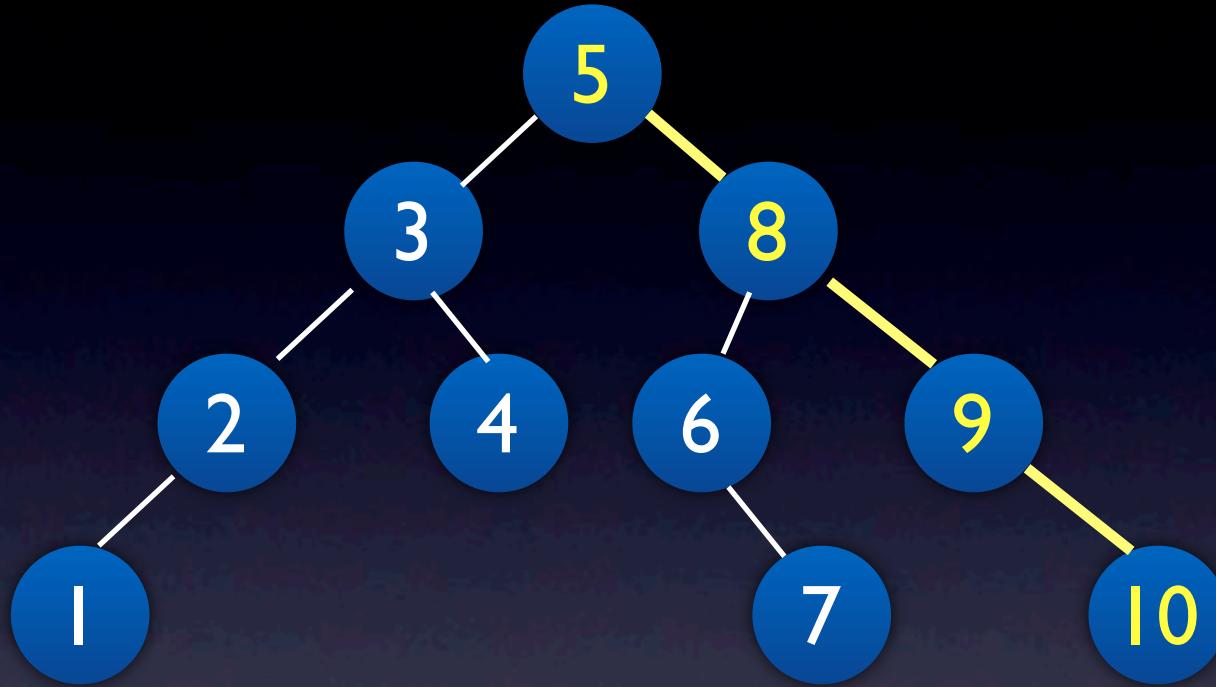






Se nesse momento
desejarmos inserir a chave
de valor 10, será que teremos
a mesma rapidez?





Nesta árvore, que **está balanceada**, tivemos que dar poucos passos para chegar na posição desejada.



Nesta outra, que não está
balanceada, tivemos que fazer
um percurso muito maior.

Conclusão:



Árvores binárias são mais
eficientes quando balanceadas.

Problema:

Como garantir o
balanceamento de uma
árvore?



É aí que surge a nossa
incrível árvore AVL!!!

O termo AVL vem de
seus criadores, Adelson
Velsky e Landis.

Definição

É uma árvore binária de busca auto-balanceada onde as alturas das duas sub-árvore a partir de cada nó difere no máximo em uma unidade.

As operações de busca, inserção e eliminação de elementos possuem complexidade $O(\log n)$.

Definição

Ao inserir ou eliminar elementos, a árvore terá de ser verificada para que, caso necessário, seja feito um rebalanceamento, que ocorre por meio de rotações.

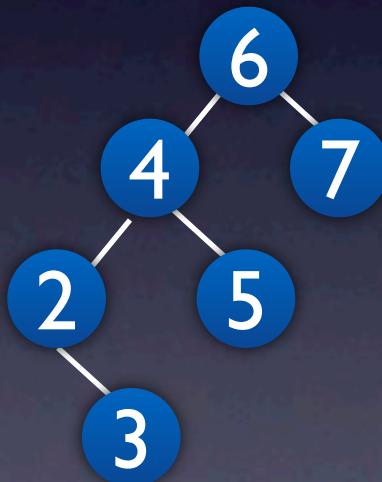


Vamos ver um pouco da
implementação em
Java...



Entendendo as rotações

Vejamos essa árvore:



Entendendo as rotações

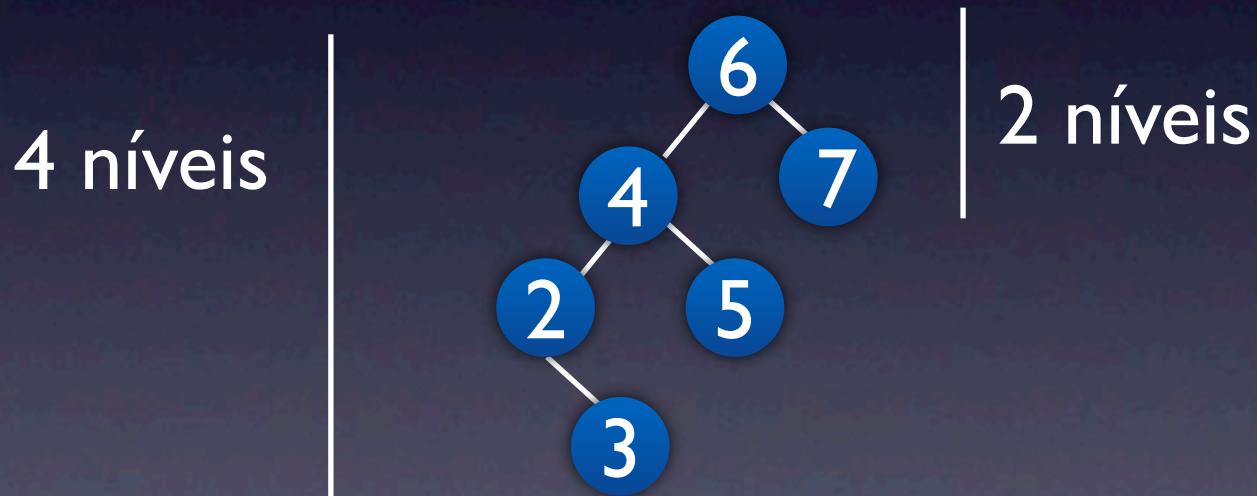
Rotação Simples

A altura da sub-árvore de raiz 4 difere de mais de 1 unidade da de chave 7.



Entendendo as rotações

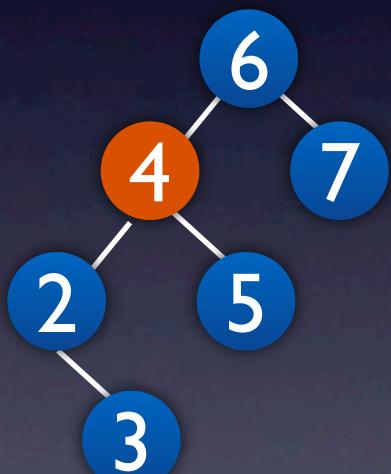
Como a altura da sub-árvore à esquerda de 6 difere de 2 em relação à sua sub-árvore direita, rotaciona-se em torno do 4.



lado esquerdo desbalanceado -> rotação para direita

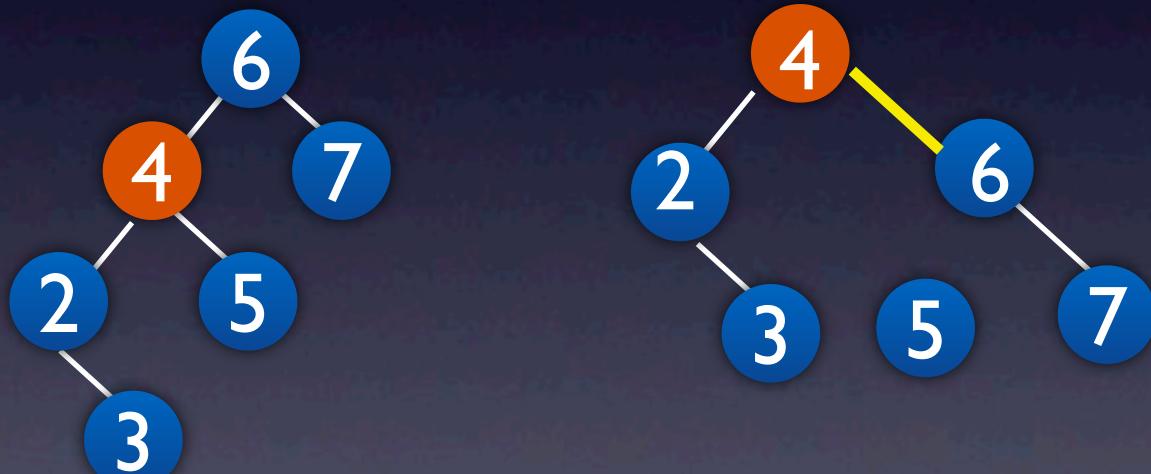
Entendendo as rotações

4 passa a ser a nova raiz



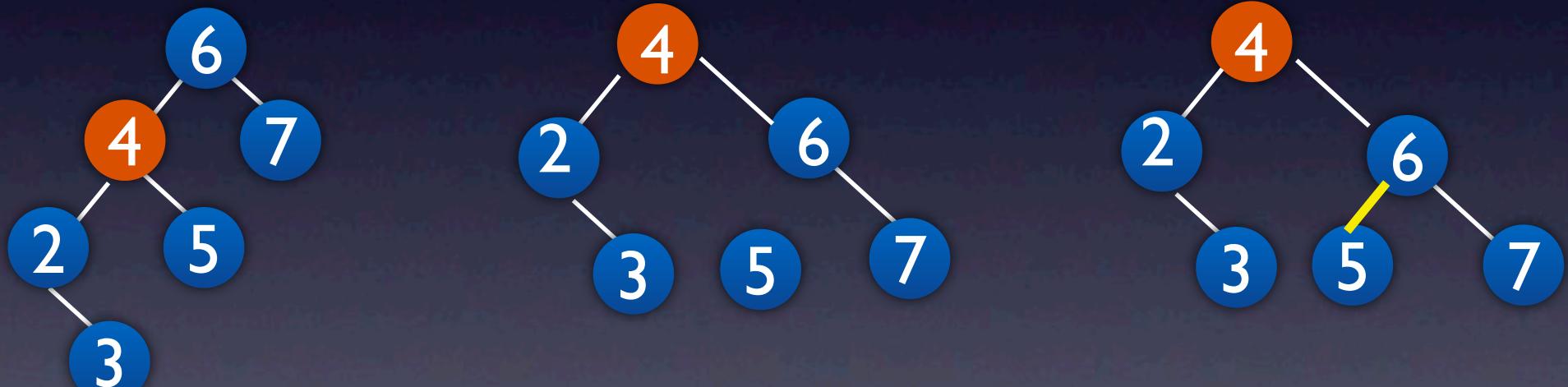
Entendendo as rotações

sub-árvore enraizada em 6 vira filho direito de 4



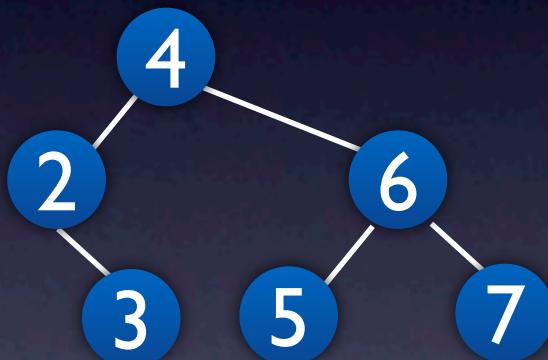
Entendendo as rotações

5 passa a ser filho esquerdo de 6



Entendendo as rotações

Resultado Final



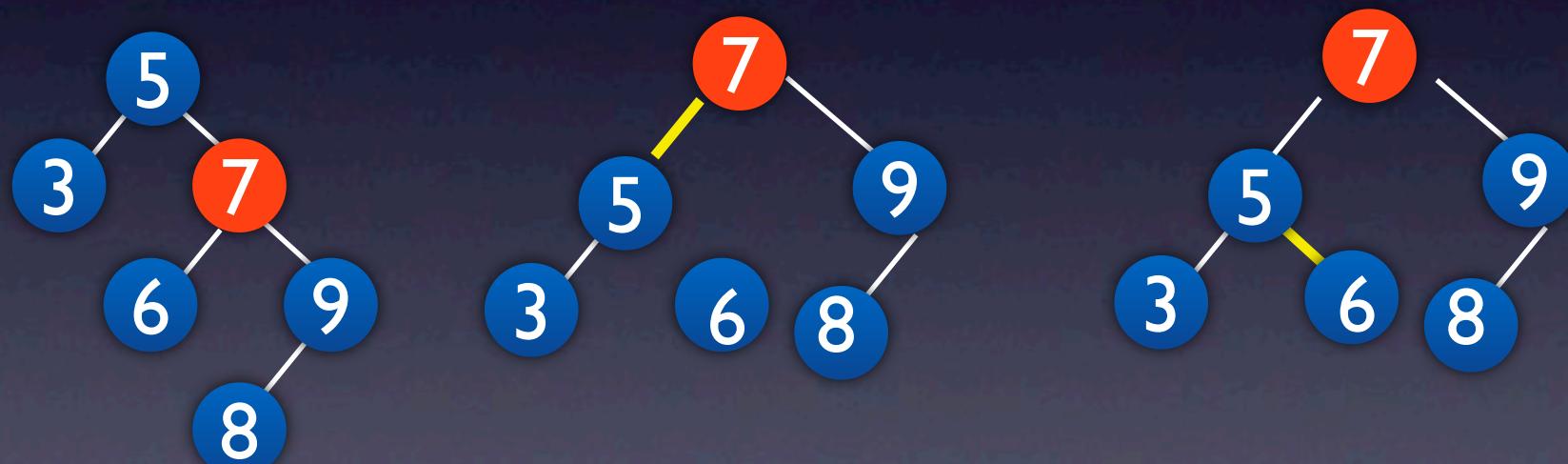
Balanceada!

Entendendo as rotações

Uma rotação à esquerda segue o mesmo algoritmo
porém invertendo direita com esquerda.

Entendendo as rotações

A altura da sub-árvore de raiz 7 difere de mais de 1 unidade da de chave 3.



Entendendo as rotações

Generalizando...

Rotação à Direita

Seja Y o filho à esquerda de X

Torne X o filho à direita de Y

Torne o filho à direita de Y o filho à esquerda de X.

Rotação à Esquerda

Seja Y o filho à direita de X

Torne X filho à esquerda de Y

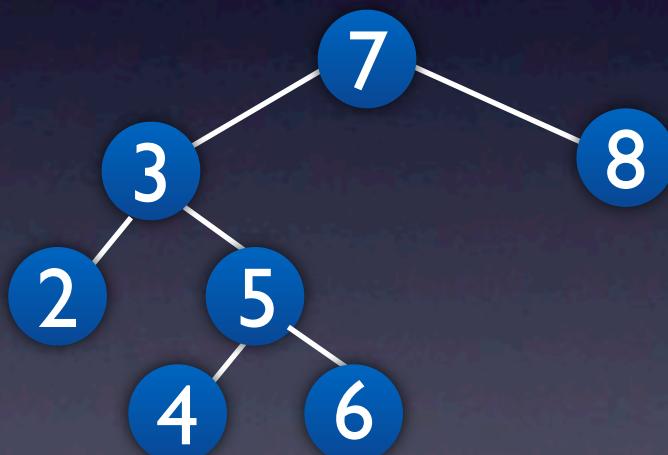
Torne o filho à esquerda de Y o filho à direita de X.

Entendendo as rotações

Mas nem sempre apenas uma rotação é suficiente para balancear a árvore...

Entendendo as rotações

Como balancear essa árvore ?



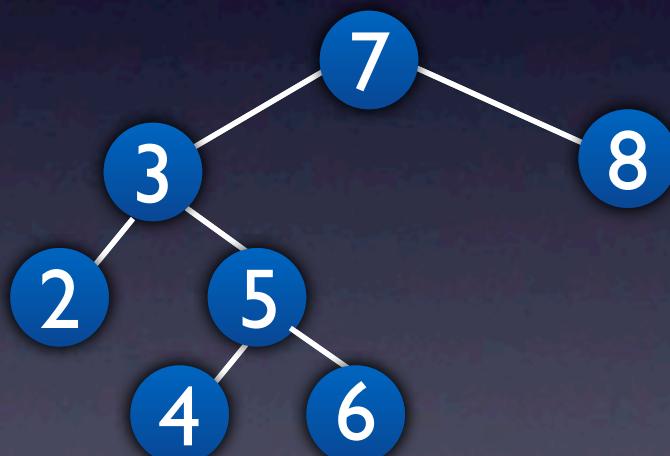
Entendendo as rotações

Rotação Dupla

Se um nó estiver desbalanceado e seu filho estiver inclinado no sentido inverso ao pai, teremos a ocorrência de uma rotação dupla.

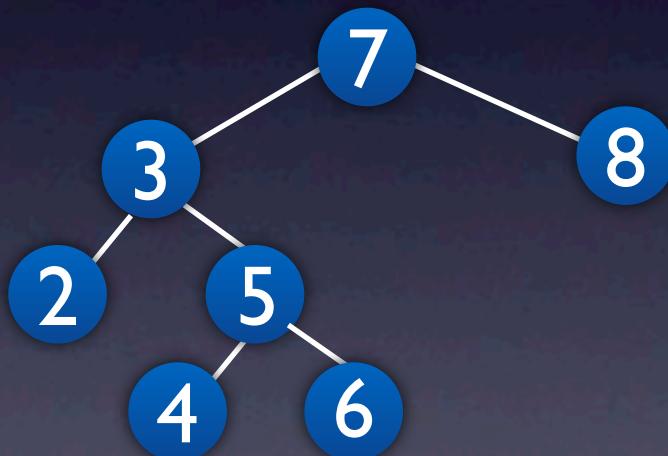
Na verdade, trata-se apenas de duas rotações simples seguidas.

Entendendo as rotações



Entendendo as rotações

Se um nó estiver desbalanceado e seu filho estiver inclinado no sentido inverso ao pai, teremos a ocorrência de uma rotação dupla.



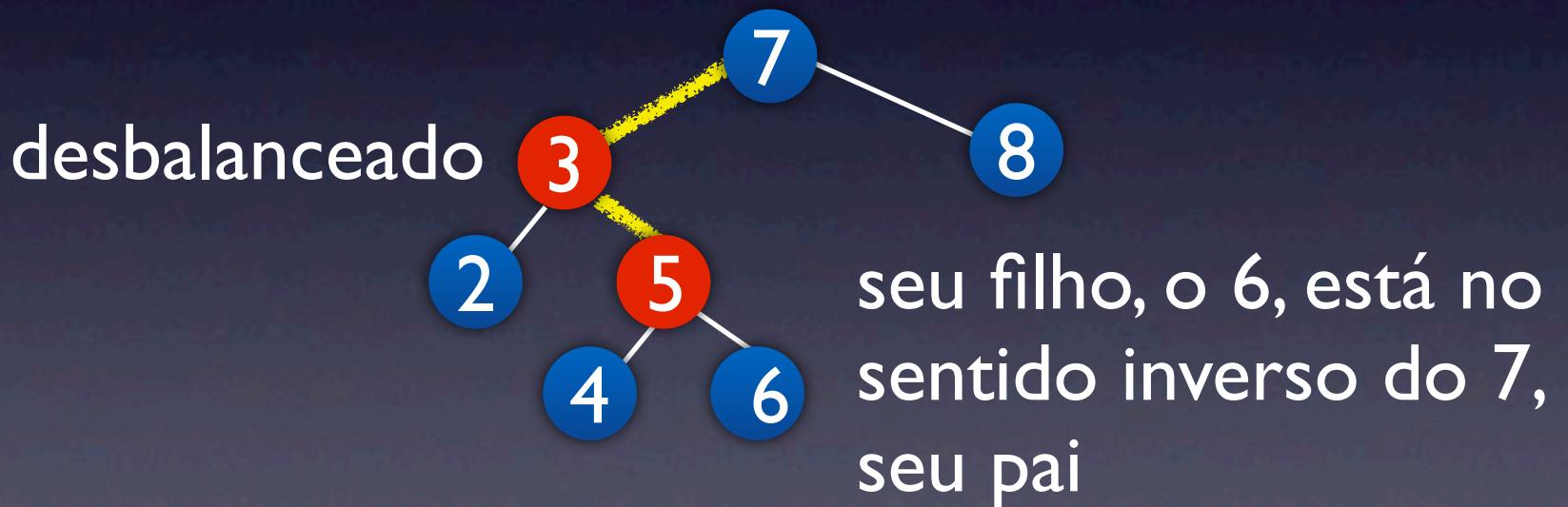
Entendendo as rotações

Se um nó estiver desbalanceado e seu filho estiver inclinado no sentido inverso ao pai, teremos a ocorrência de uma rotação dupla.

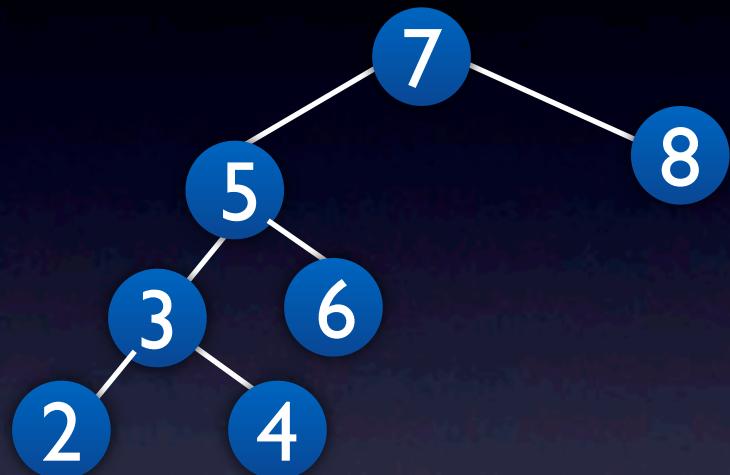


Entendendo as rotações

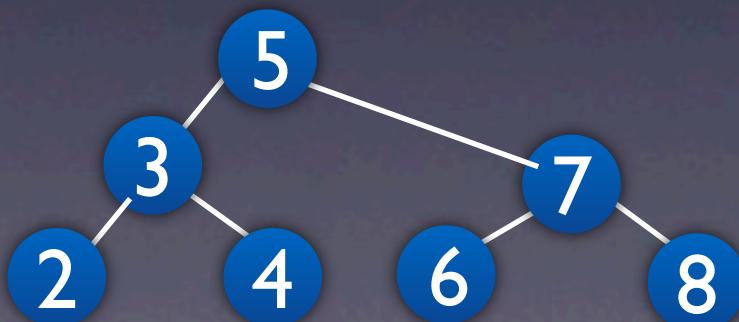
Se um nó estiver desbalanceado e seu filho estiver inclinado no sentido inverso ao pai, teremos a ocorrência de uma rotação dupla.



Rotação Dupla à direita



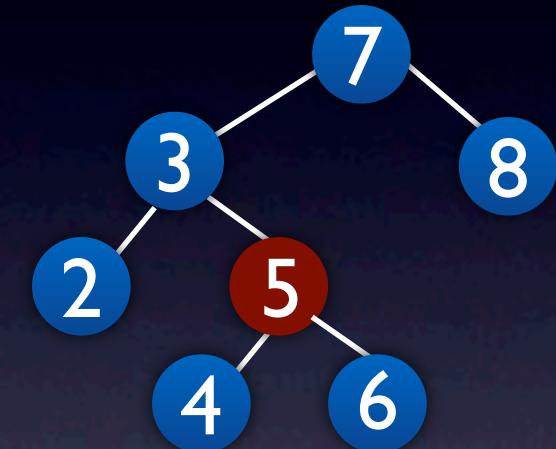
Rotação Simples à esquerda



Rotação Simples à direita

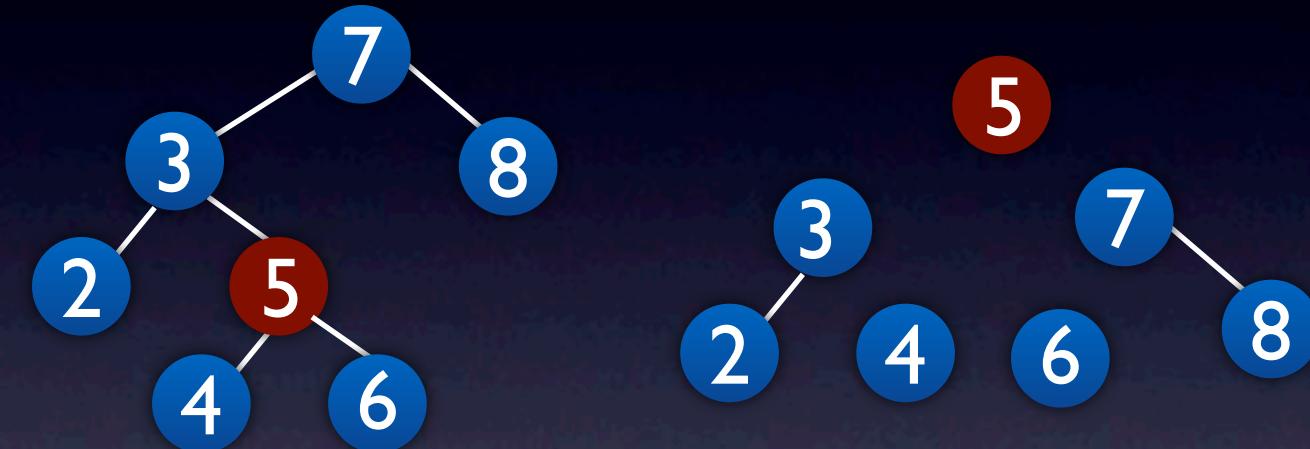
Rotação Dupla à direita

5 será a nova raiz da sub-árvore enraizada em 7



Rotação Dupla à direita

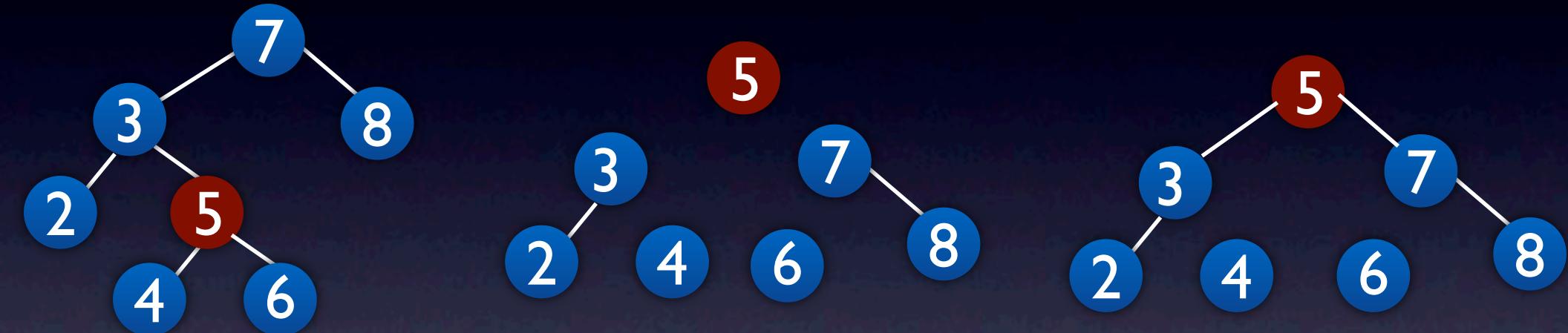
Como a nova raiz passa a ser o 5, o nó 3 passa a não ser mais filho de 7



Além disso, as arestas de 4 e 6 com 5 são desfeitas

Rotação Dupla à direita

As sub-árvores de raizes 3 e 7 se ligam a 5



Rotação Dupla à direita

4 se liga a 3 e 6 se liga a 7



Vamos ver agora as
rotações em Java...



Inserção

A inserção em uma árvore AVL ocorre da mesma forma que a inserção na árvore binária, porém, levando em conta o balanceamento da árvore, ou seja, realizando as operações de rotação quando necessárias.

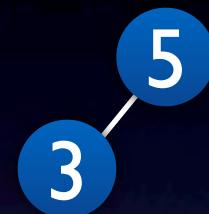
Exemplos

Insere 5

5

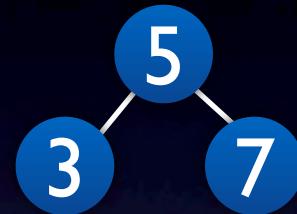
Exemplos

Insere 3 - balanço correto - difere I



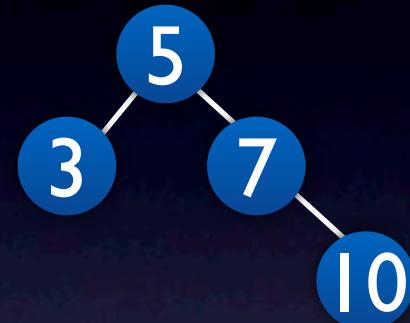
Exemplos

Insere 7 - balanço correto - difere 0



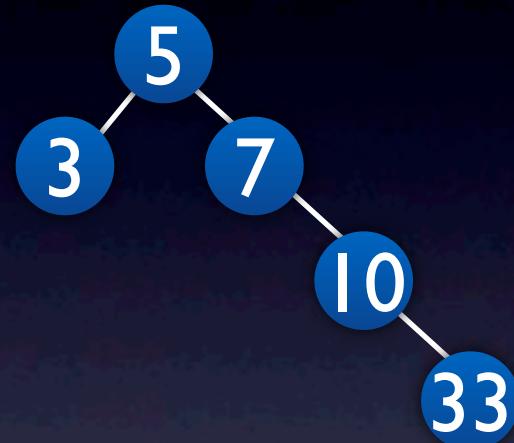
Exemplos

Insere 10 - balanço correto - difere I



Exemplos

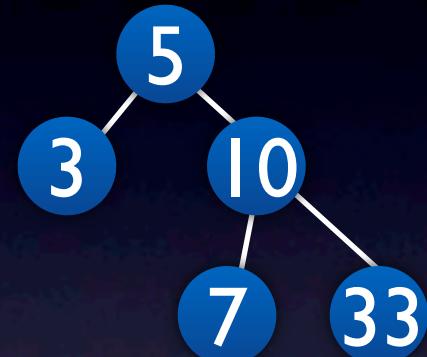
Insere 33 - balanço errado - difere 2



Neste caso, realizar rotação simples à esquerda

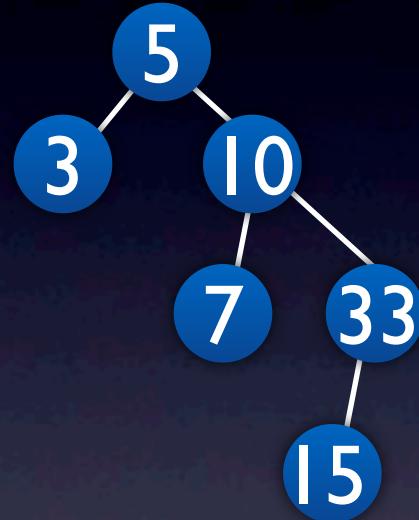
Exemplos

balanço correto - difere I



Exemplos

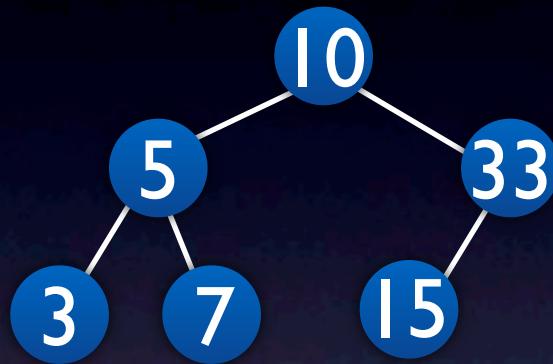
Insere 15 - balanço incorreto - difere 2



Neste caso, realizar rotação simples à esquerda

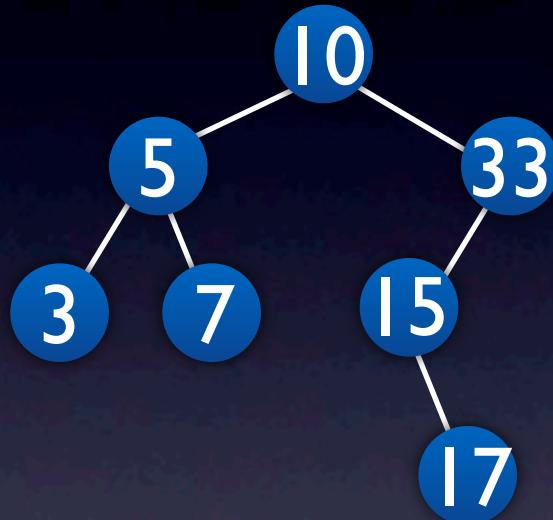
Exemplos

Agora o balanço está correto - difere I.



Exemplos

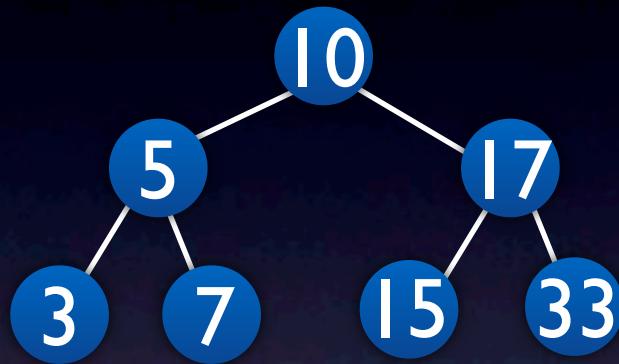
Insere 17 - balanço incorreto - difere 2



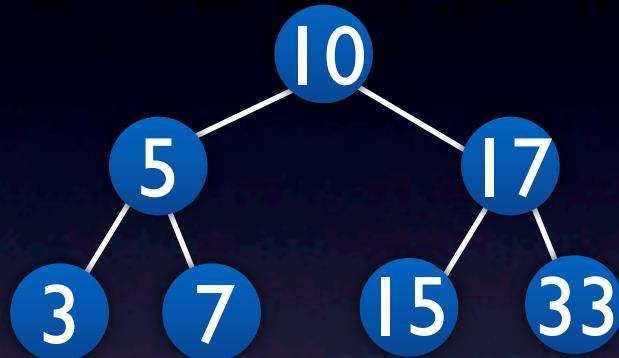
Neste caso, realizar rotação dupla à direita

Exemplos

balanceada - diferença de zero nas alturas



Exemplos

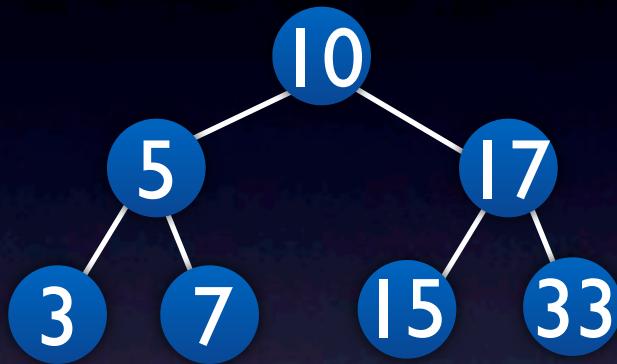


Isto é uma AVL balanceada.

Vamos ver o algoritmo
de inserção em nosso
programa em Java...



Continuando...



Agora vamos brincar com a remoção...

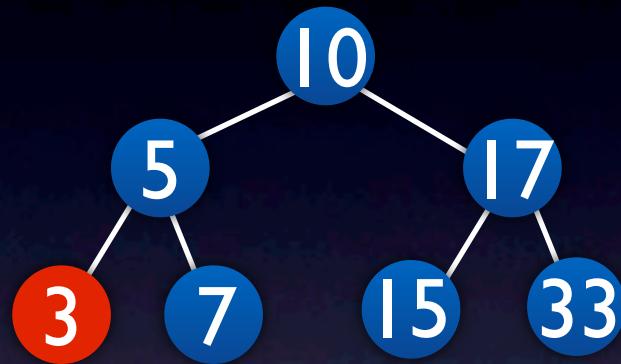
Remoção

Remover um nó de uma árvore AVL consiste em remover diretamente o nó caso ele seja uma folha e verificar o balanceamento realizando as rotações quando necessárias.

Caso o nó não seja uma folha troca-se ele pelo maior nó menor que ele (uma folha) e então prossegue como explicado acima.

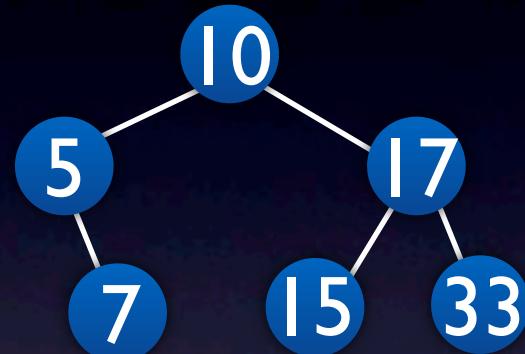
Exemplos

Remove 3



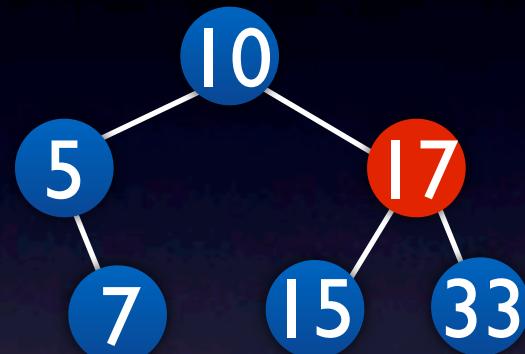
Exemplos

OK, balanceada (fácil)



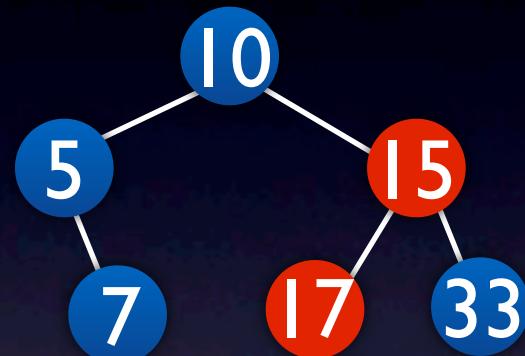
Exemplos

Agora vamos remover o 17...



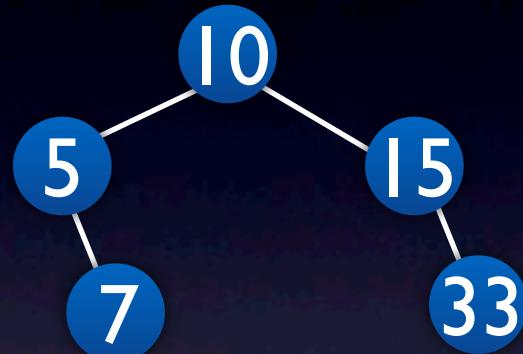
Exemplos

Agora vamos remover o 17...



Exemplos

Foi trocado pela chave mais à direita do seu filho esquerdo.

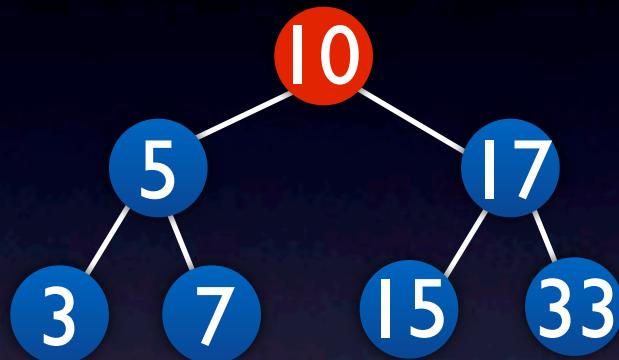


Vamos complicar um pouco...

Voltando a árvore balanceada antes
das remoções...

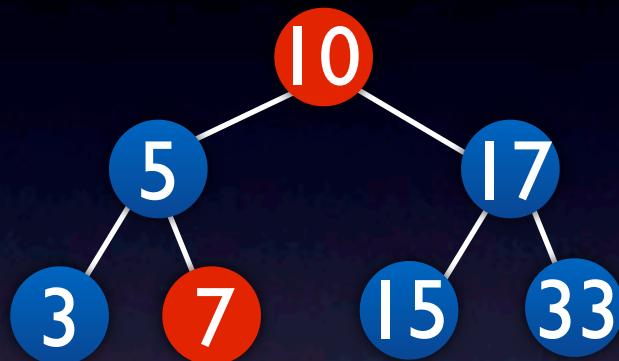
Exemplos

Remover o 10!



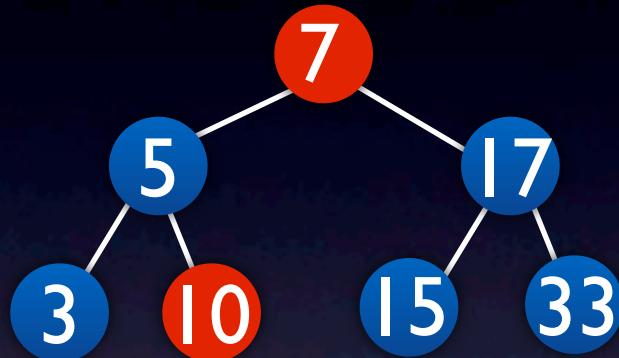
Exemplos

7 é o nó mais à direita do filho esquerdo de 10



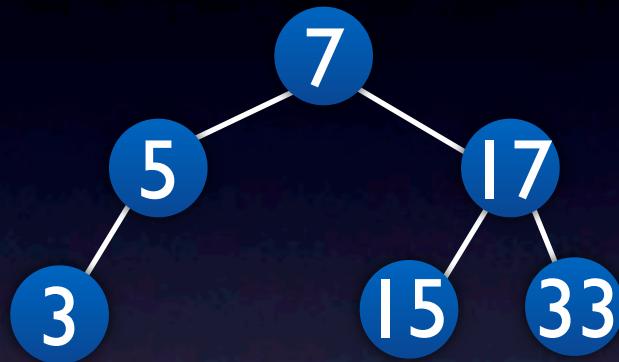
Exemplos

Invertemos as posições e excluimos a folha 10



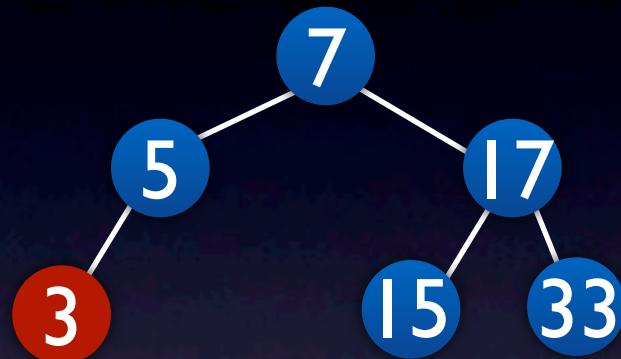
Exemplos

7 vira raiz e a árvore está balanceada.



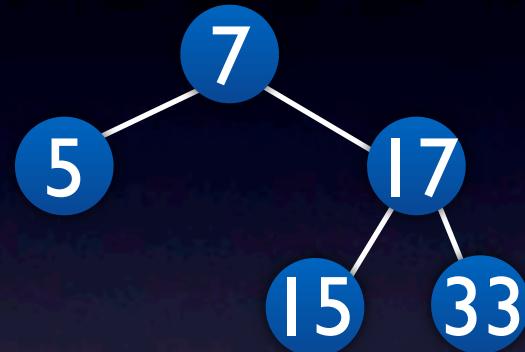
Exemplos

Agora vamos remover o 3



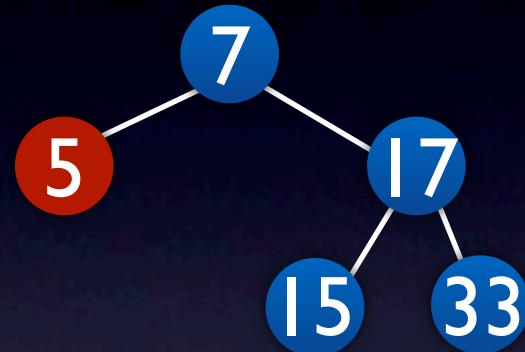
Exemplos

Balanceada - difere de I



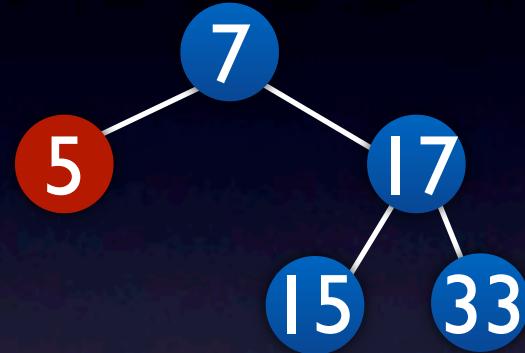
Exemplos

Agora vamos remover o 5



Exemplos

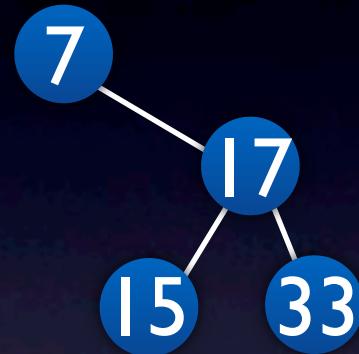
Agora vamos remover o 5



Note que agora teremos que fazer rotações, pois ficará desbalanceada.

Exemplos

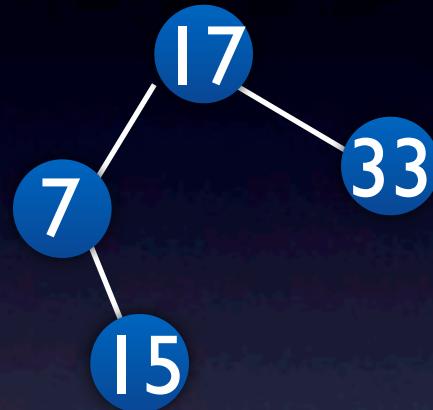
Balanceamento errado - difere 2



Faremos uma rotação simples à esquerda.

Exemplos

Balanceamento correto - difere |



Agora está novamente balanceada.

E por fim, o algoritmo
de remoção...



O desempenho das operações
numa árvore pode ser medido
pela sua altura.

Árvore	Altura Máxima
A.Bin.Busca	n
Rubro-Negra	$2 * \log_2(n)$
AVL	$1.44 * \log_2(n)$

Árvore	Complexidade
A.Bin.Busca	$O(n)$
Rubro-Negra	$O(\log_2(n))$
AVL	$O(\log_2(n))$

Vantagens

Árvores AVL são mais rápidas nas operações de busca.

Desvantagens

Perde-se tempo quando necessitamos fazer um rebalanceamento. Isso pode ocorrer nas inserções e remoções.

Conclusão

Árvores AVL são bastante eficientes.

Vamos agora ver mais exemplos no programa que desenvolvemos...

