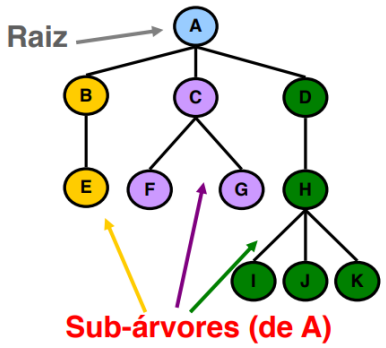
Terminologia Árvores

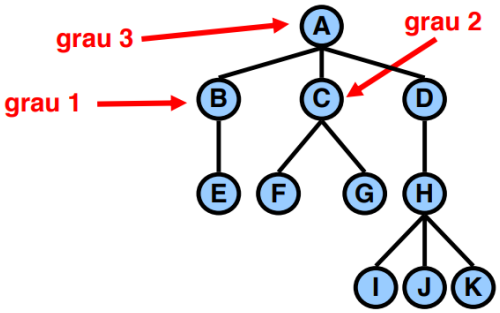
Raíz e Sub-Árvores



- Nº de subárvores = nº de nós-1.

Grau

- Número de subárvores ou número de filhos
- O grau de uma árvore é o máximo entre os graus de seus nodos



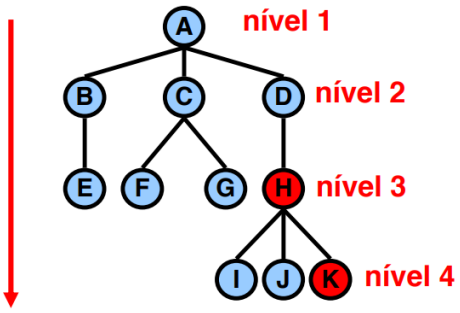
- Nó de derivação tem grau maior que zero.
- Nó folha tem grau 0

Caminho

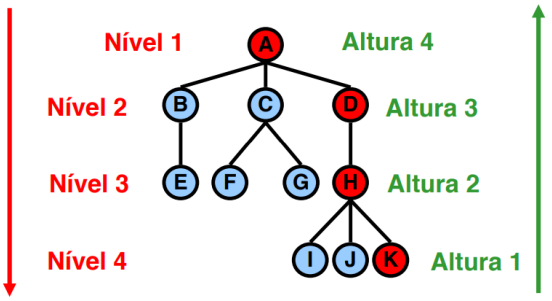
- Sequência de nodos consecutivos distintos entre dois nós
- Comprimento do caminho: Número de níveis entre os 2 nodos (v1, vk) - 1

Nível

- Nº de ligações entre a raíz e o nodo, + uma unidade



Altura



- Altura é a profundidade dos nodos
- N° de ligações entre o nodo e o nodo folha descendente dele de maior nível + 1

Operações sobre árvores:

Criação de árvore
Inserção de nodo (raiz, meio, folha)
Exclusão de nodo
Acesso a um nodo
Destruição da árvore

Na implementação por encadeamento, o acesso é somente pela raíz!

Árvores binárias
- Operação aritmética

Árvores binárias de pesquisa
- ABP, AVL, R-N, Splay

Árvores Binárias

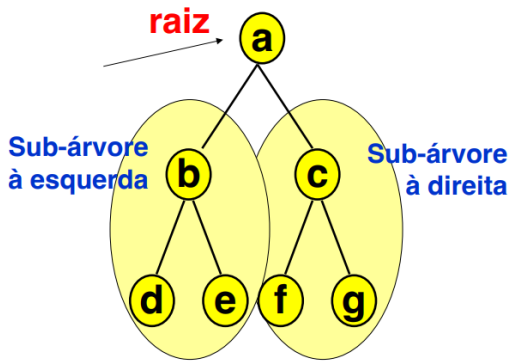
Como transformar árvore n-aria para binária?

1. O primeiro filho de um nodo passa a ser seu filho à esquerda na árvore binária
2. Os demais filhos de um nodo passam a ser filhos à direita do seu irmão imediato à esquerda
3. Executar o mesmo processo para cada nodo da árvore
4. Para converter uma floresta em árvore binária, basta considerar as raízes das árvores como nós irmãos e aplicar a conversão anterior

Nodo de uma árvore binária:

```
typedef struct sNodoA{
   char info;
   struct sNodoA *esq;
   struct sNodoA *dir;
} TNodoA;
```

Caminhamentos:
O acesso é sempre através da raiz, e cada nodo deve ser visitado só uma vez



Pré-Fixado à esquerda:
- Visita a raiz
- Percorre a subárvore esquerda
- n Percorre a subárvore direita
- a-b-d-e-c-f-g

Central à esquerda:
- Percorre a subárvore esquerda
- Visita a raiz
- Percorre a subárvore direita
- Caminhamento para obter os valores em ordem crescente
- d-b-e-a-f-c-g

Pós-fixado à esquerda:
- Percorre a subárvore esquerda
- Percorre a subárvore direita
- Visita a raiz
- d-e-b-f-g-c-a

Pré-fixado à direita:
- Visita a raiz
- Percorre a subárvore direita
- Percorre a subárvore esquerda
- a-c-g-f-b-e-d

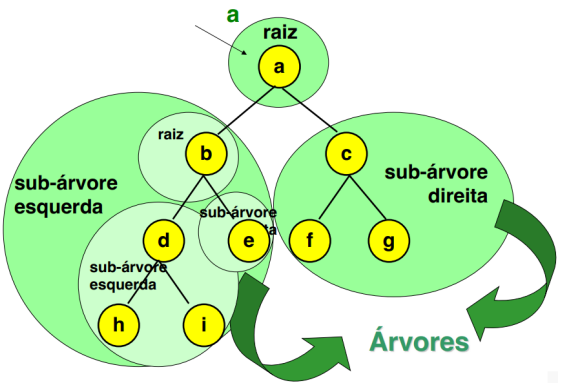
Central à direita:
- Percorre a subárvore direita
- Visita a raiz
- Percorre a subárvore esquerda
- Ordem decrescente
- g-c-f-a-e-b-d

Pós-Fixado à direita
- Percorre a subárvore direita
- Percorre a subárvore esquerda
- Visita a raiz
- g-f-c-e-d-b-a

Operação aritmética: Pré/Pós fixado

Os seis caminhamentos tem o mesmo tempo de processamento

Recursividade:
Funciona como colocar cada nodo em uma pilha, e depois voltar retirando.
É usada para percorrer a árvore, uma vez de cada lado.



Na recursividade…
Se é à esquerda, a recursividade começa pela esquerda.
Se é à direita, a recursividade começa pela direita.
Printf:
- Pré fixado: antes
- Central: no meio
- Pós-fixado: no final

Exemplo: Central Esquerda

```
void central(TNodoA* a)
{
    if (a!= NULL)
    {
        central(a->esq);
        printf("%c\n",a->info);
        central(a->dir);
     }
}
```

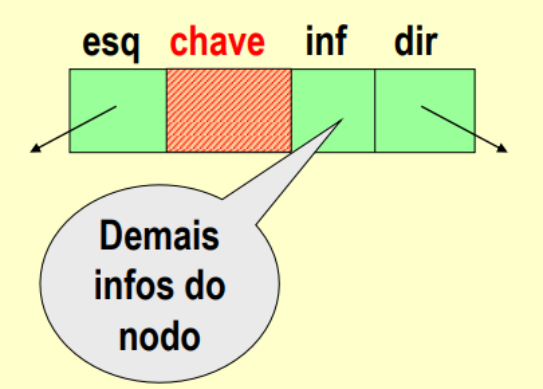
Árvores Binárias de Pesquisa

O que são?

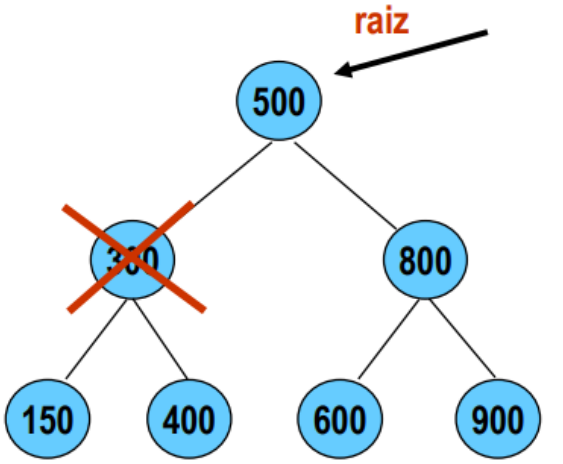
Árvores com uma relação de ordem entre seus nós.

A ordem é definida pela chave!

A ordem deve sempre ser respeitada!



Exclusão de nodo em uma árvore ordenada:



Três possíveis casos:

- Nó é folha:
  Apenas remover
- Nó não é folha, e tem uma subárvore:
  A raiz da subárvore ocupa o lugar do nodo excluído
- Nó não é folha, e tem duas subárvores
  Árvore precisa ser reestruturada

Como reestruturar recursivamente?

Trocar o valor do nó a ser removido com:

- Valor do nó que tenha a MAIOR CHAVE da sua subárvore à ESQUERDA
  OU
- Valor do nó que tenha a MENOR CHAVE da sua subárvore à direita.

Desse jeito, o valor a ser excluído, quando é trocado, vira nó folha. Assim, basta somente excluir.

É possível repetir o processo se necessário!

Árvores Balanceadas

Em uma simples Árvore Binária de Pesquisa, pode ocorrer o desbalanceamento! Surgem então alternativas para uma distribuição equilibrada de nós, que otimiza a consulta e diminui o número médio de comparações.

Distribuição Uniforme:

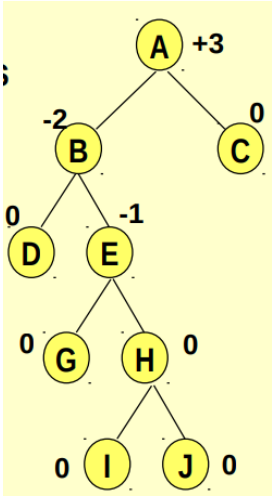
- Balanceamento por altura
  - AVL
  - Rubro-Negras

Distribuição não Uniforme:

- Balanceamento por frequência
  - Splay

AVL

- Altura da subárvore direita difere da altura da subárvore esquerda em no máximo 1
- Cada subárvore do nodo tem a propriedade *fator*



No caso, esse fator é a diferença de altura entre as subárvores da direita e esquerda, para cada nodo.

Positivo: Altura maior na esquerda

Negativo: Altura maior na direita

O fator da árvore é o fator do nodo com maior fator!

Para permanecer AVL após inserção e exclusão, são necessárias rotações!

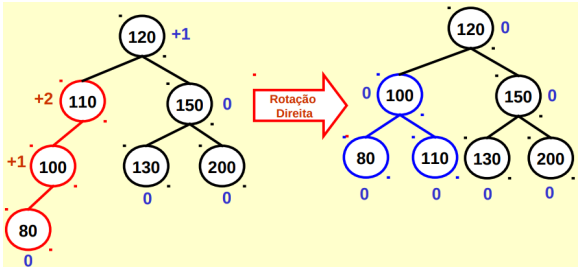
Reestruturação da AVL

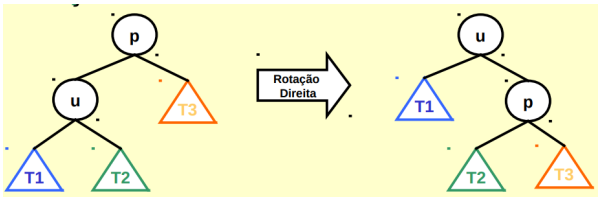
- Preserva a ordem das chaves
- Uma rotação já torna a Árvore AVL novamente

Rotação SIMPLES

À DIREITA:

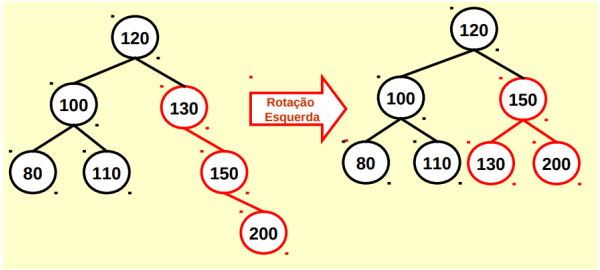
- Árvore com fator positivo e subárvore da esquerda com fator positivo

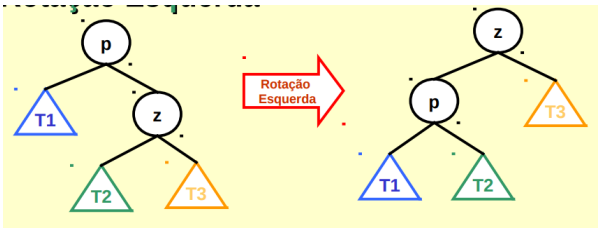




À ESQUERDA:

- Árvore com fator negativo e subárvore da direita com fator negativo

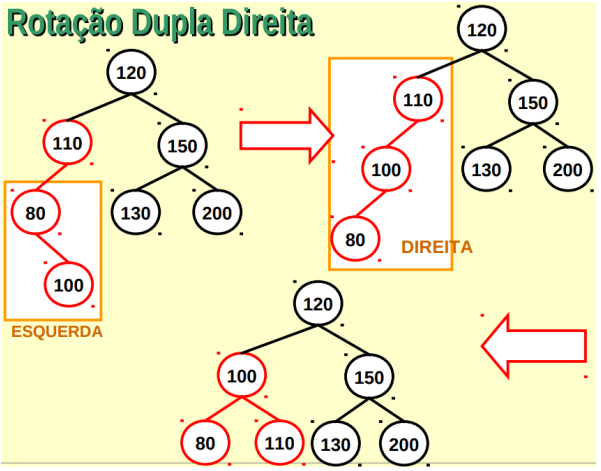


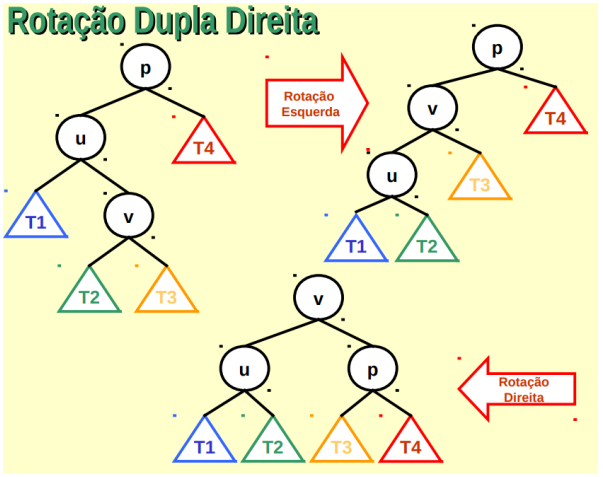


Rotação DUPLA

À DIREITA:

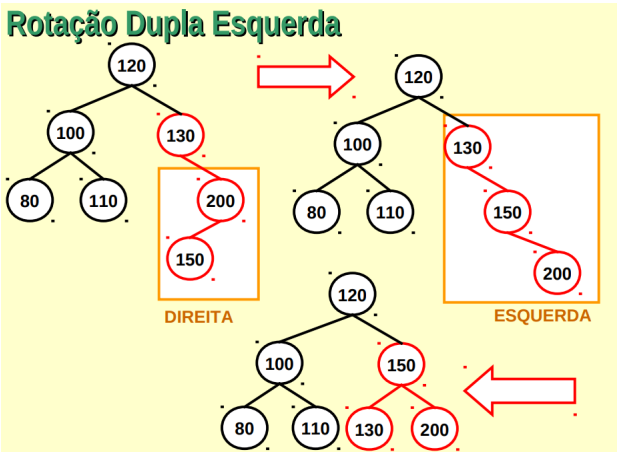
- Árvore com fator positivo e subárvore da esquerda com fator negativo
- Composta por uma rotação simples à esquerda + uma rotação simples à direita

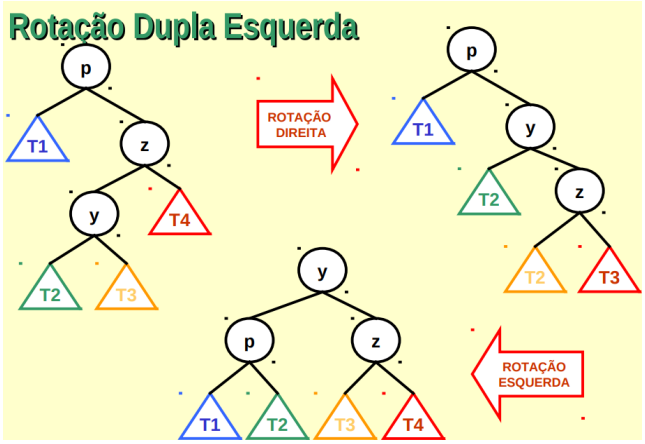




À ESQUERDA

- Árvore com fator negativo e subárvore da direita com fator positivo
- Composta por uma rotação simples à direita + uma rotação simples à esquerda
-





Lembrando que numa inserção, se a chave/nó já existe, a tentativa de inserção é ignorada.
Se a inserção acontece, é preciso verificar se ela vai tornar a árvore desbalanceada (para que ela possa ser balanceada depois)
Também é preciso descobrir QUAL operação de rotação precisa ser executada, e executá-la.

Balanceamento
Como descobrir uma árvore desbalanceada:

- Subtrair valor de alturas de cada subárvore
  OU
- Armazenar variável balanço em cada nó
  balanço: altura(esq) - altura(dir)

Calculando o balanceamento:
Quando um nodo 'q' é inserido. Considerando o nodo já existente 'v'.

- 'q' pertence à sub-árvore esquerda de 'v', e a inclusão aumenta a subárvore
  balanço(v)++
- 'q' pertence à subárvore direita de 'v' e a inclusão aumenta a subárvore
- balanço(v)--

Se o balanço é dois, a árvore está desbalanceada.

Calculando o aumento na altura:
Depende do fator(v) antes da inclusão.

Inserção à DIREITA:

- fator(v) era 1 -> balanço(v) será 0
- fator(v) era 0 -> balanço(v) será -1
- fator(v) era -1 -> balanço(v) será -2

Inserção à ESQUERDA:

- fator(v) era -1 -> balanço(v) será 0
- fator(v) era 0 -> balanço(v) será 1
- fator(v) era 1 -> balanço(v) será 2

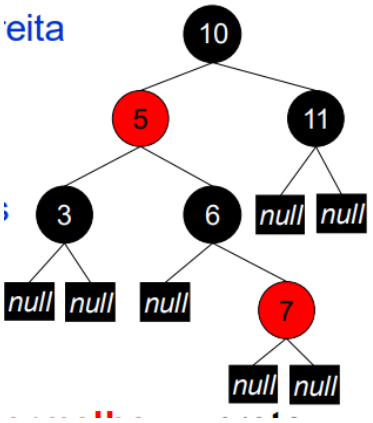
A remoção de nodos é semelhante à inclusão. Nem sempre é possível resolver com só uma rotação.

Árvores Rubro-Negras

- É uma árvore binária de pesquisa com nodos coloridos de vermelho e preto.
- Em qualquer caminho da raíz até as folhas, nenhum caminho será maior que duas vezes o comprimento de qualquer outro (aproximadamente balanceada)
- Menos rotações que a AVL

Campos:
– Chave
– Ponteiro para subárvores esquerda
– Ponteiro para subárvores direita
– Cor (todo nodo é vermelho ou preto)

Nodos null sao tratados como folhas!



- A raíz é PRETA
- Se um nodo é VERMELHO, seus filhos são PRETOS.
- Para cada nodo, TODOS os caminhos de um nodo até as folhas descendentes têm o mesmo número de nodos pretos (Black-Height).

Quando a estrutura da árvore é modificado e alguma propriedade é violada, é necessário:

- Alterar as cores de nodos
- Fazer rotações

Todo nodo é inserido VERMELHO.

- Se o pai é PRETO, não há promblema
- Se o pai é VERMELHO, uma propriedade é violoda

Inserção com pai VERMELHO

- Avô é PRETO
- Cor do TIO
  - Vermelho
  - Preto

Tio VERMELHO:

Alterar cores pai, tio e avô.

- Pai e Tio -> Virarão PRETOS
- Avô -> Virará VERMELHO

Caso o avô seja a RAIZ:

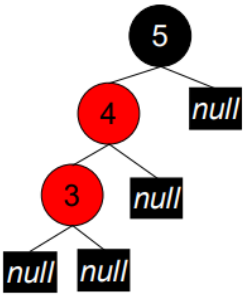
- Depois das mudanças, altera também Avô-> Volta a ser PRETO

Tio PRETO:

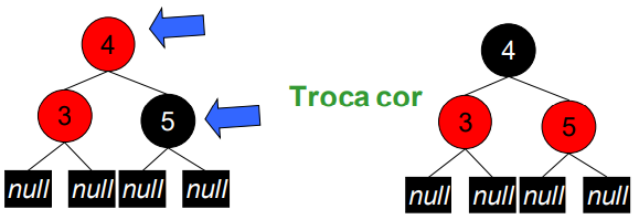
- Analisar situações !

A) Rotação direita

Exemplo: Adicionando o 3

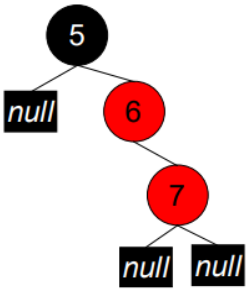


- Faz a ROTAÇÃO Direita
- TROCA a cor de pai e filho.

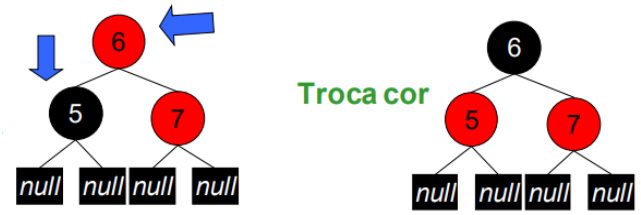


B) Rotação esquerda

Exemplo: Adicionando o 7

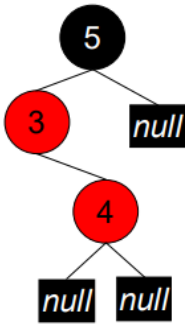


- Faz a ROTAÇÃO esquerda
- TROCA a cor de pai e avô

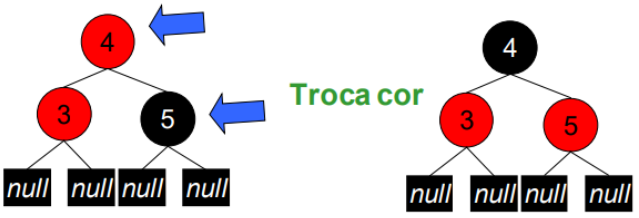


C) Rotação dupla direita

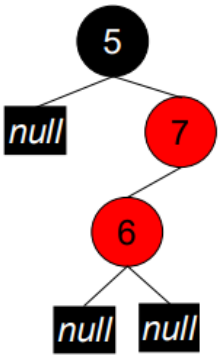
Exemplo: Adicionando o 4



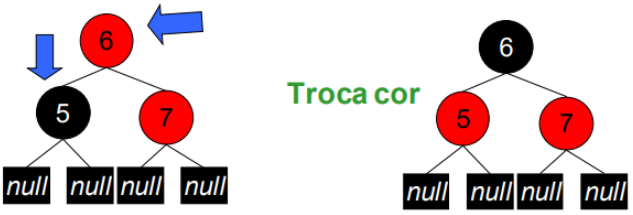
- Faz a ROTAÇÃO DUPLA direita
- TROCA a cor do nó e avô (depois da rotação dupla)



D) Rotação dupla esquerda



- Faz a ROTAÇÃO DUPLA esquerda
- Troca cor de nó e avô (depois da rotação dupla)



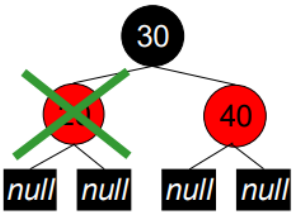
Remoção
Similar à exclusão padrão de ABP.

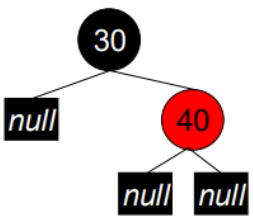
Remoção de nodo intermediário: não há problema porque as cores permanecem iguais, só troca de valores.

Remoção de nodos folha:

Remoção de Nodo Folha Vermelho:

- Não altera o balanceamento da árvore
- Nodo deletado não é raiz
- Nodos vermelhos não se tornam adjacentes



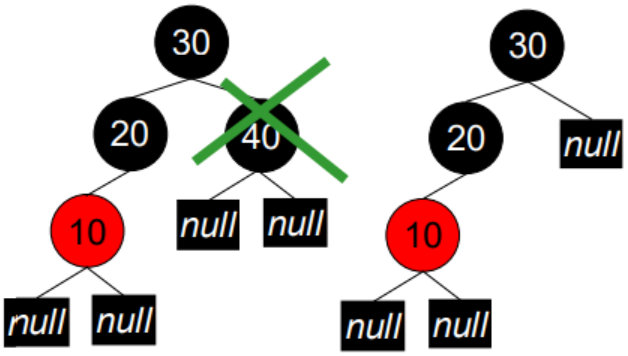


Remoção de Nodo Folha Preto:

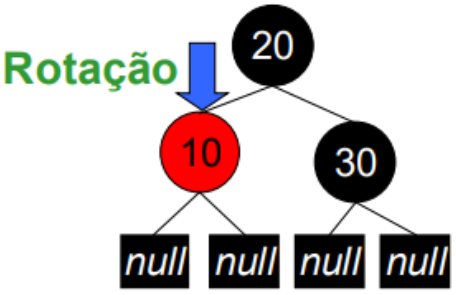
- Gera problema! Altera o balanceamento
- Se inclui um nodo preto adicional nos caminhos que passam pelo nodo que foi excluído

CASO IRMÃO PRETO - FILHO VERMELHO

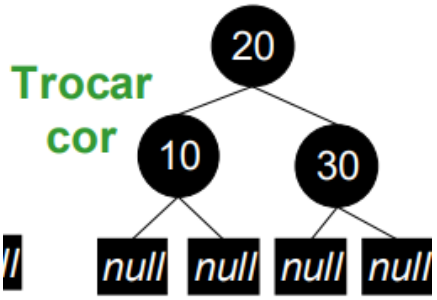
1. ao remover, a árvore fica desbalanceada



2. por isso, preciso fazer a rotação correspondente



3. depois disso, é necessário trocar a cor



Outros exemplos estão no moodle

Árvores Splay

- Não usa regras de altura ou balanço.
- Usa operação de mover para a raiz (SPLAYING).

As operações de rotação são usadas para mover o nodo acessado para a raiz! Sempre há rotação

- É balanceada por frequência! Os nodos recentemente inseridos/acessados são localizados mais rápidos.
- Com acesso uniforme aos nodos, o desempenho é pior que outras ABPS!

- PIOR CASO: Nodos acessados sequencialmente em ordem

Splaying

- Quando pesquisamos pela chave k e ela é encontrada em um nodo x, o nodo é movido para raiz.
- Quando k não é o encontrado, o pai do nodo externo onde se termina a busca é movido para a raiz.
- Quando uma chave é removida, seu pai é movido pela raiz.

Passos para o SPLAY

- Em cada passo, x vai ser movido para mais perto da raiz.

Se o nodo tem um avô, cada passado depende de

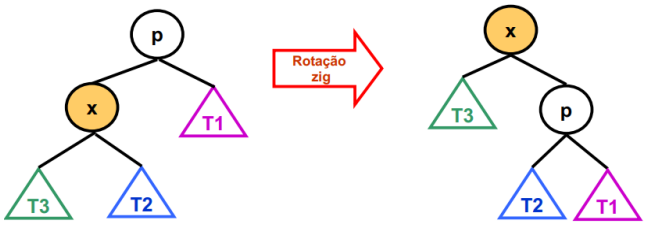
o nodo x é
- filho esquerdo de p
- filho direito de p

o nodo pai (p) é
- filho esquerdo do avô v
- filho direito do avô v

Rotações

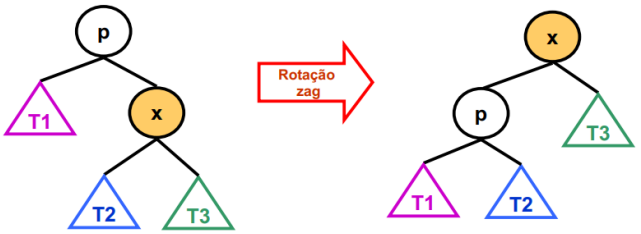
ZIG

Quando x é filho da raiz e pai > filho (x)



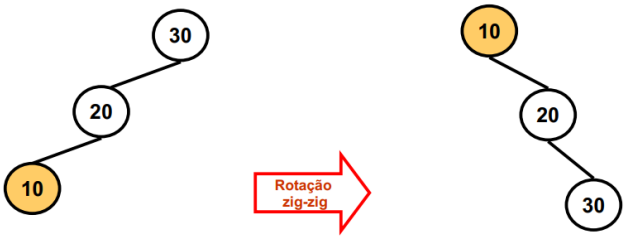
ZAG

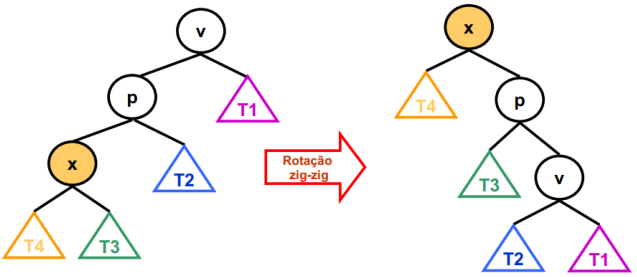
Quando x é filho da raiz e filho>pai



ZIG-ZIG

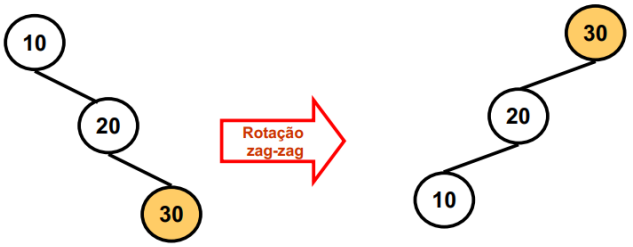
filho(x) < pai < avô

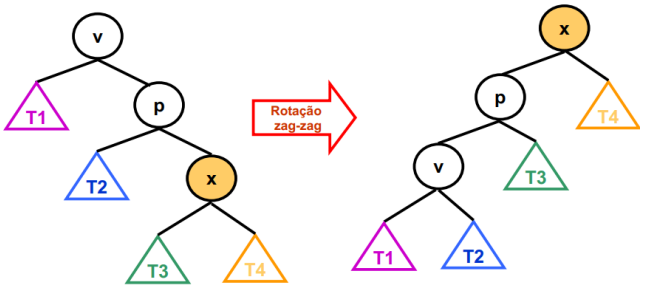




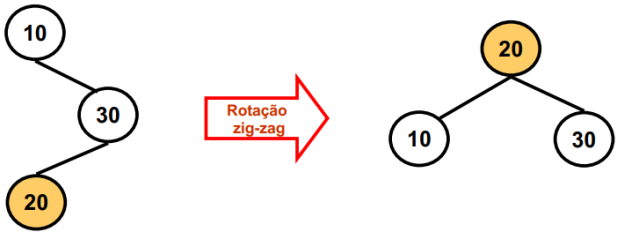
ZAG-ZAG

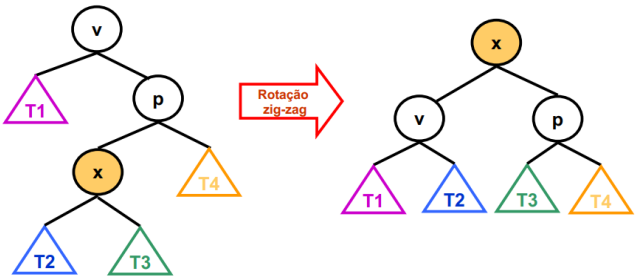
filho(x) > pai > avô



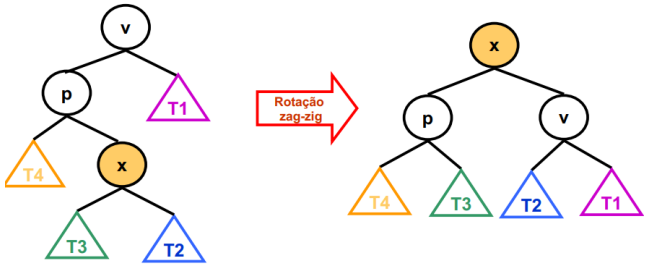


ZIG-ZAG
pai > filho(x) >avô

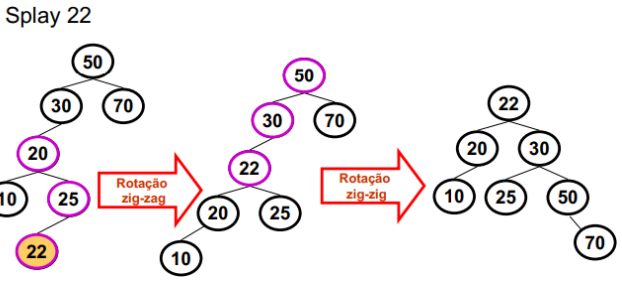




ZAG-ZIG
avô > filho(x) > pai



O split se faz combinando-se as rotações:



Para isso, a cada rotação, é preciso verificar qual deve ser feita de acordo com os critérios.

Nesse exemplo, para a primeira rotação, é possível verificar que
pai > filho(x) > avô

Depois, ainda verifica-se que para a segunda rotação
avô > pai > filho(x)

Por fim, quando verifico que 22 já está acima, posso parar de fazer as rotações;

altura maxima e minima
(lista é a máxima, completamente balanceada é a mínima)