Árvores balanceadas

Aleardo Manacero Jr.

Árvores Balanceadas

- Para que uma árvore seja, de fato, um mecanismo eficiente, é preciso que os seus elementos estejam distribuídos de forma relativamente homogênea pela estrutura (sub-árvores)
- Como as operações de inserção e remoção são aleatórias, é preciso ter um operador capaz de manter os elementos distribuídos de forma homogênea entre as sub-árvores
- Esse é o operador de **balanceamento**

Árvores Balanceadas

- O balanceamento de uma árvore pode ser feito segundo duas estratégias:

Global – envolvendo toda a árvore

Local – envolvendo apenas uma parte da árvore

Balanceamento Global

- No balanceamento global a árvore balanceada pode ser construída a partir de uma estrutura externa
- Nesse caso o algoritmo é relativamente simples, bastando criar uma lista ordenada com o conteúdo da árvore (antes ou mesmo depois da existência da árvore)
- Dessa lista constrói-se a árvore já balanceada

Balanceamento Global

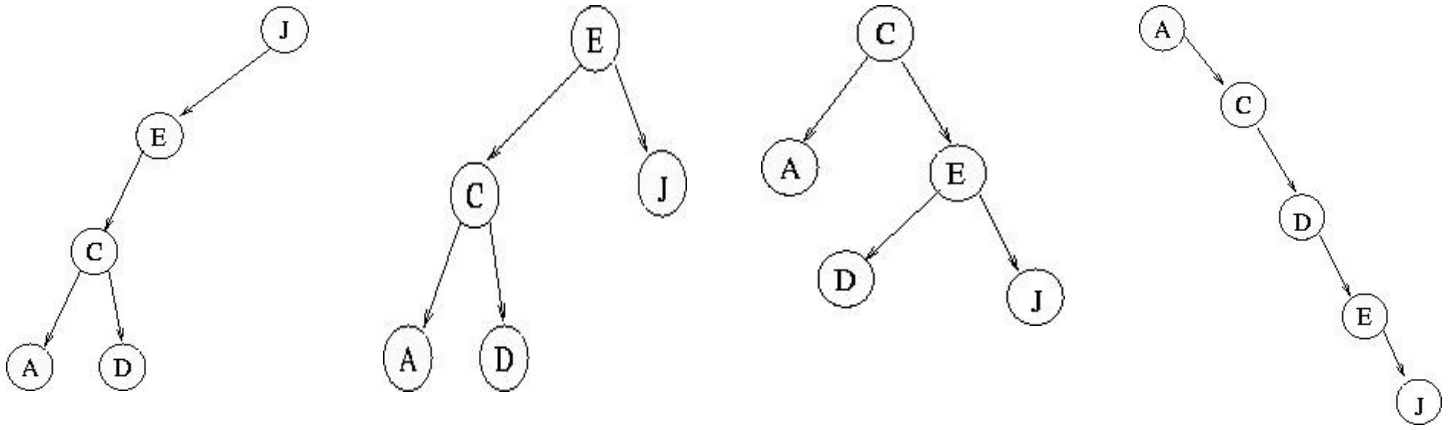
- O problema nessa estratégia é a necessidade de uma estrutura externa à árvore

- Uma solução para esse problema é apresentada através do algoritmo DSW (Colin Day, Quentin Stout e Bette Warren)

- No algoritmo DSW o que se faz é transformar a árvore em uma espinha dorsal (árvore com formato de lista) e depois reconstruir a árvore, sempre com operações de rotação

Criando a espinha dorsal

- Para cada nó *vtx* da árvore faça:
  - Se vtx não é raiz e for filho à esquerda
- Então faça vtx ser filho direito do seu avô
- A subárvore direita de vtx se torna a subárvore esquerda de seu pai
- O nó vtx passa a ter seu pai como filho à esquerda



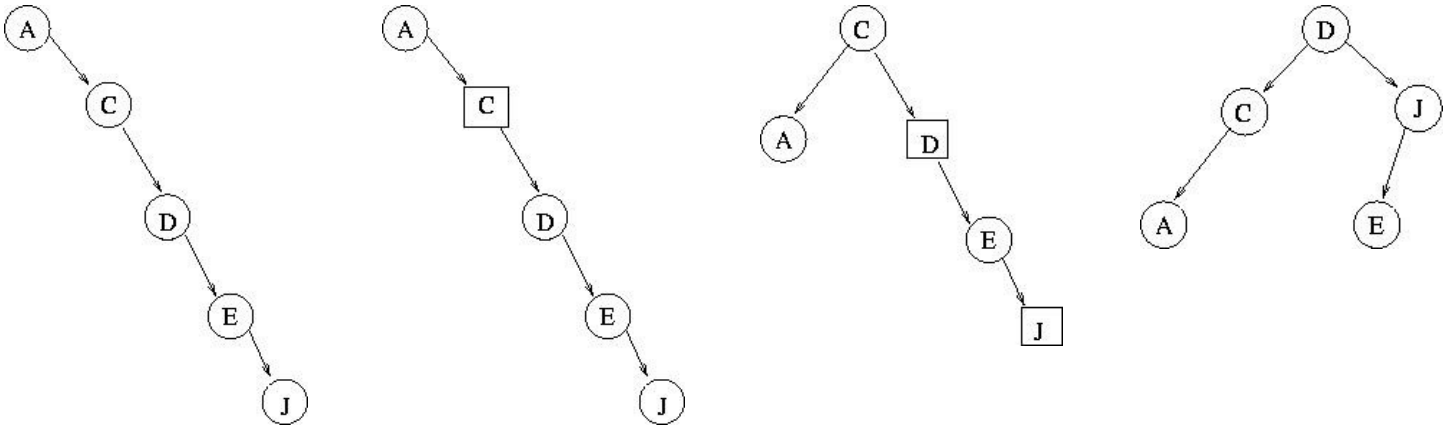
Reconstruíndo a árvore

- Na modificação feita por Stout e Warren, a reconstrução usa o seguinte algoritmo:
  1. Faz $M=2^k-1$, com $M<N$ (nós da árvore)
  2. Faz compressão (raiz, N-M)
  3. Para size=M, enquanto size>1, size/=2
     3.1 Faz compressão (raiz, size/2)

- Compressão é dada por rotações a esquerda dos nós da espinha, a partir da raiz

Reconstruíndo a árvore

- O processo é invertido, porém adotando-se passo 2 na escolha dos vértices que serão promovidos na estrutura da árvore



Balanceamento Local

- O algoritmo DSW trabalha sobre a árvore toda.

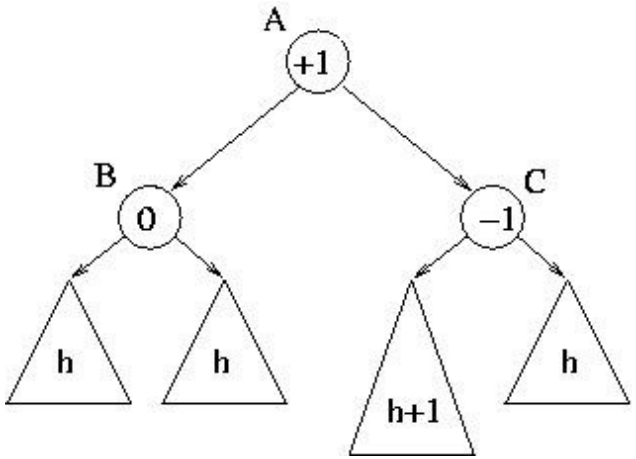
Se houver a necessidade de manter o balanceamento a cada inserção ou remoção, então sua eficiência fica bastante prejudicada

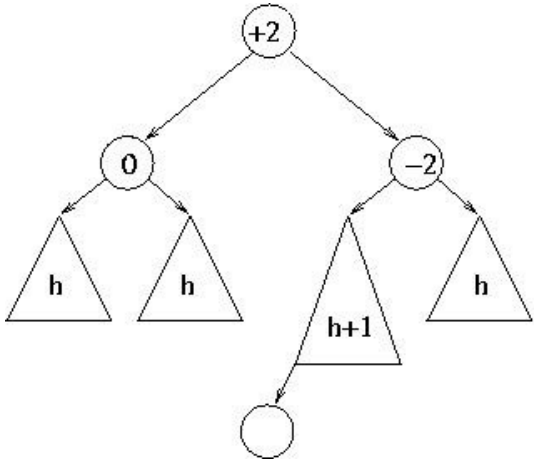
- Uma alternativa para esse problema é fazer uso de algoritmos que trabalhem apenas em parte da árvore, a cada inserção ou remoção

- Algoritmos desse tipo podem ser representados pelas árvores AVL ou árvores Rubronegras

Árvores AVL

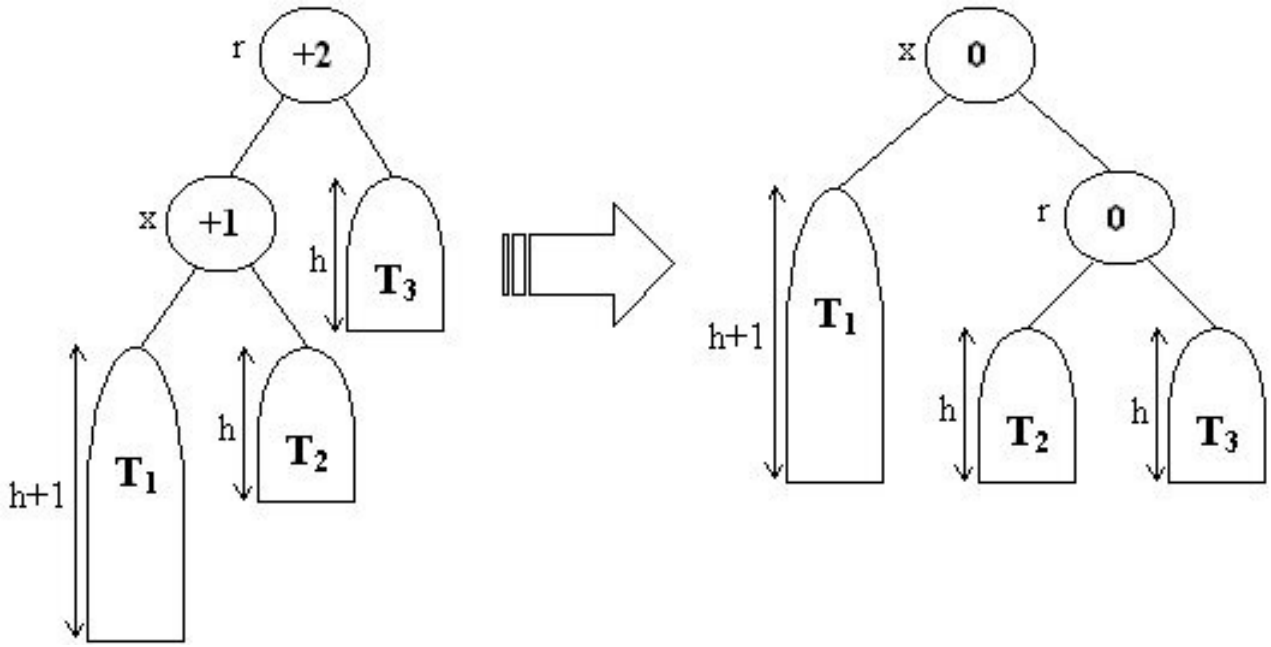
- Recebem esse nome a partir de seus criadores, os matemáticos G.M. Adelson-Velskii e E.M. Landis
- A base do algoritmo está na manutenção dos índices de balanceamento em torno de 1 (-1, 0 ou 1)
- Quando o índice chega a 2 ou –2, deve ser refeito o balanceamento através de rotações





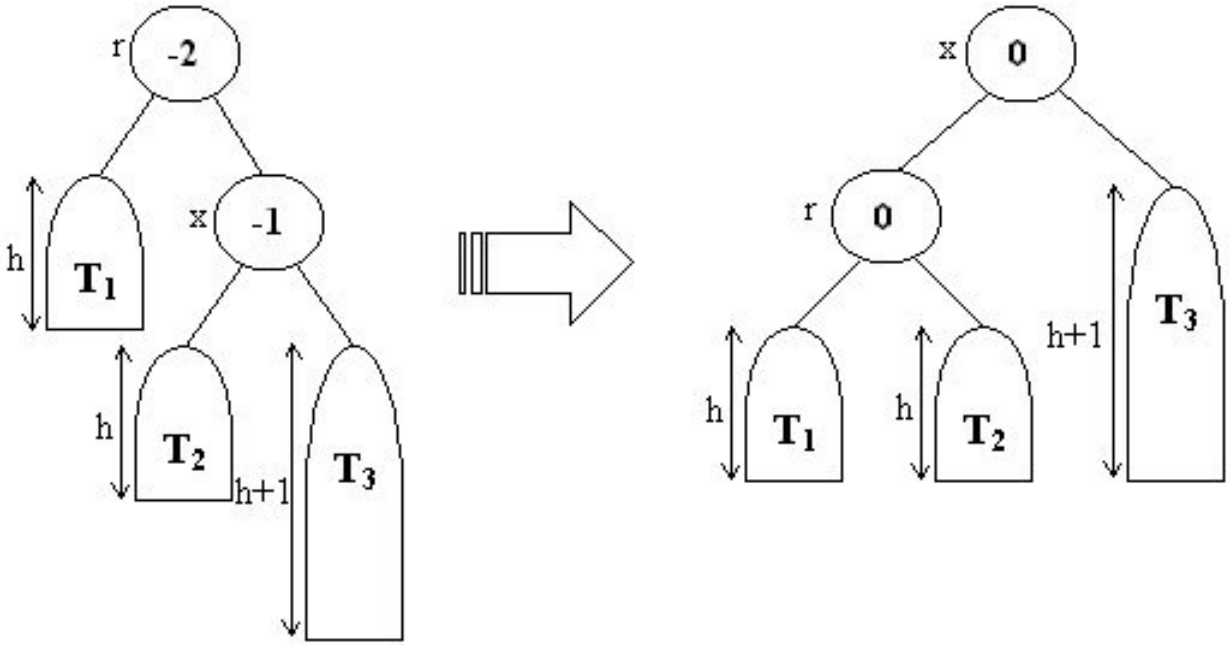
Árvores AVL

- Rotação LL



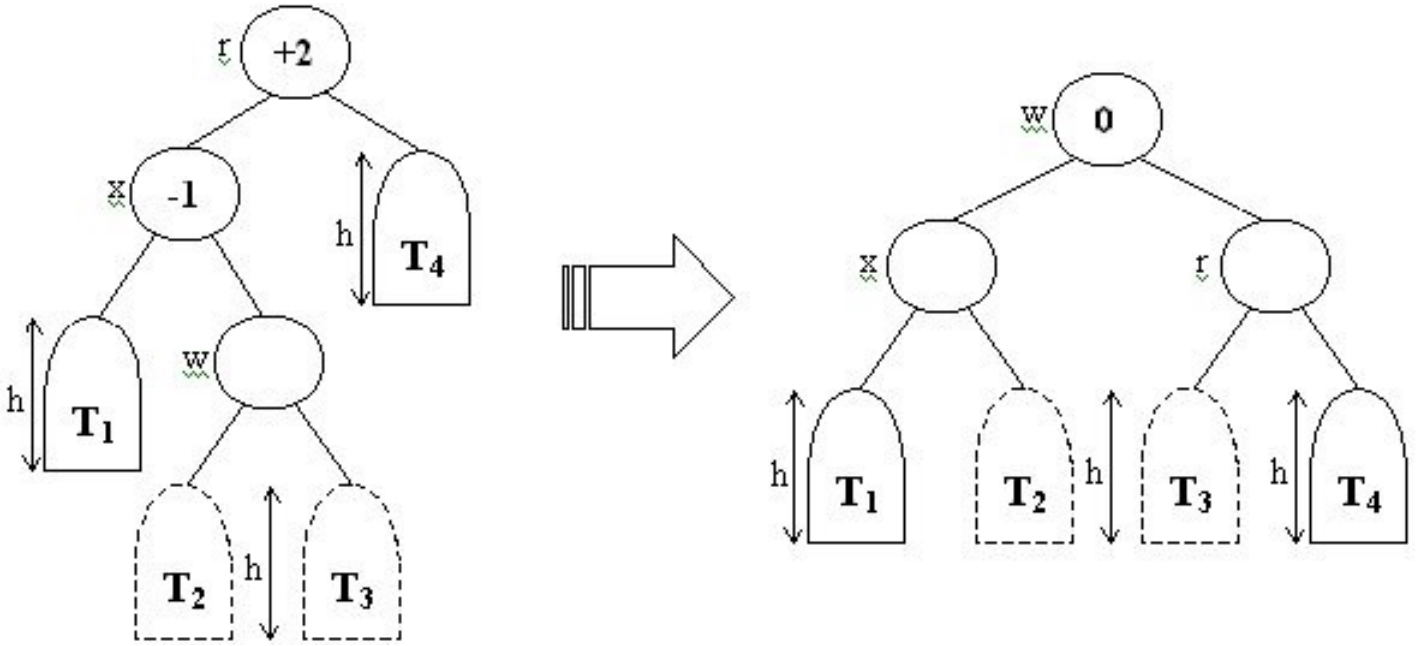
Árvores AVL

- Rotação RR



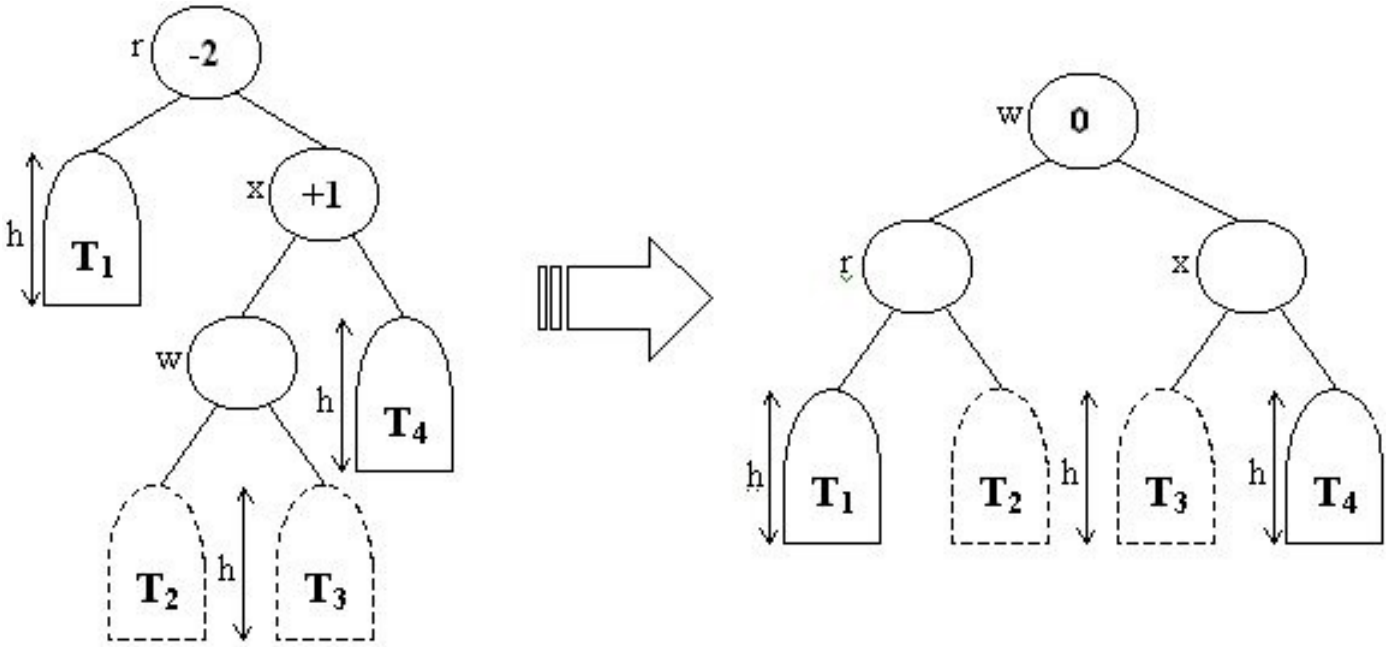
Árvores AVL

- Rotação LR



Árvores AVL

- Rotação RL



Árvores Rubronegras

- São árvores autoajustáveis, que procuram manter o balanceamento de forma automática obrigando-se que sejam atendidas as seguintes restrições:
  - Nós são vermelhos ou negros
  - A raiz é negra
  - Todas as folhas são negras
  - Os dois filhos de um nó vermelho são negros
  - Todos os caminhos de algum nó para suas folhas contêm o mesmo número de nós negros

Árvores Rubronegras

- Essas restrições permitem garantir que o máximo caminho a partir da raiz até uma folha será, no máximo, o dobro do menor caminho

- Isso garante um balanceamento mínimo razoável

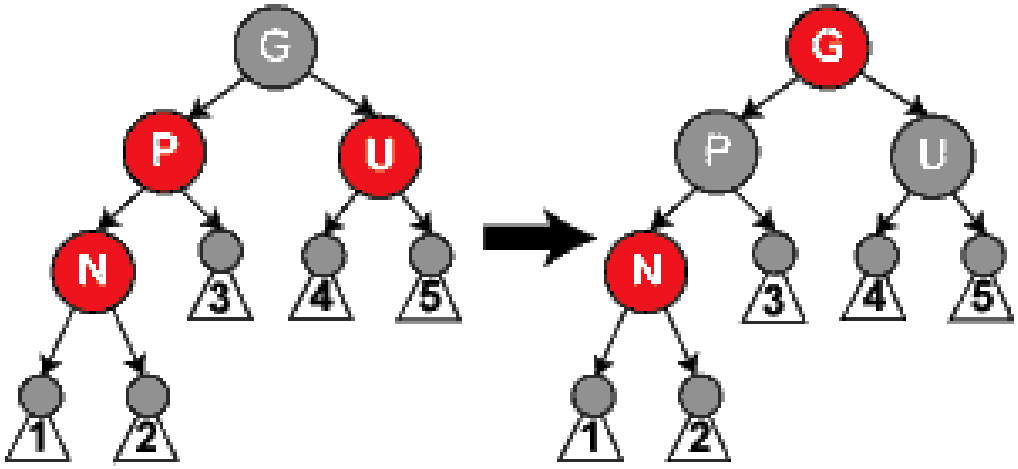
Árvores Rubronegras

- Inserção (nó raiz) – transforma-se o nó para negro

- Inserção (pai negro) – não é preciso nenhuma alteração

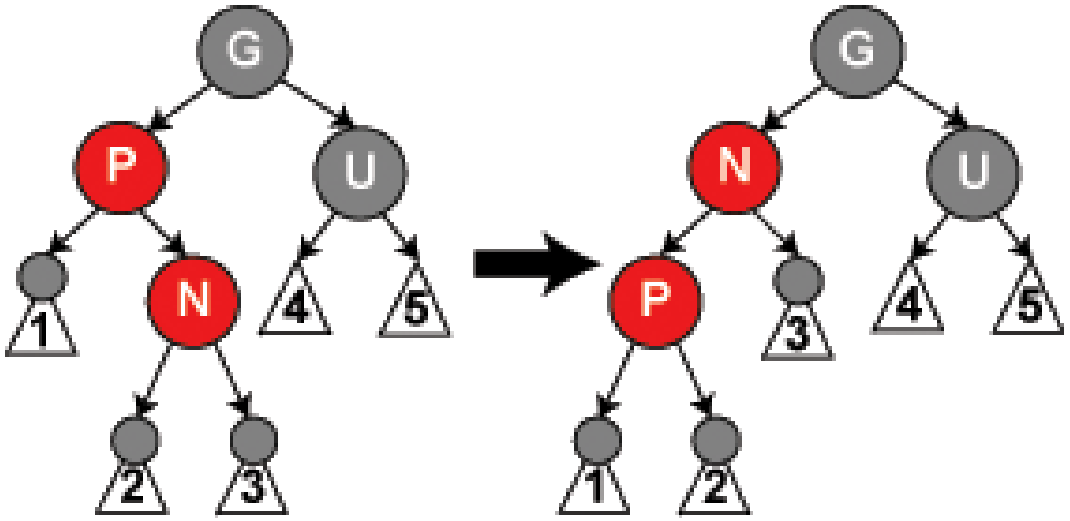
Árvores Rubronegras

- Inserção (pai e tio vermelhos)



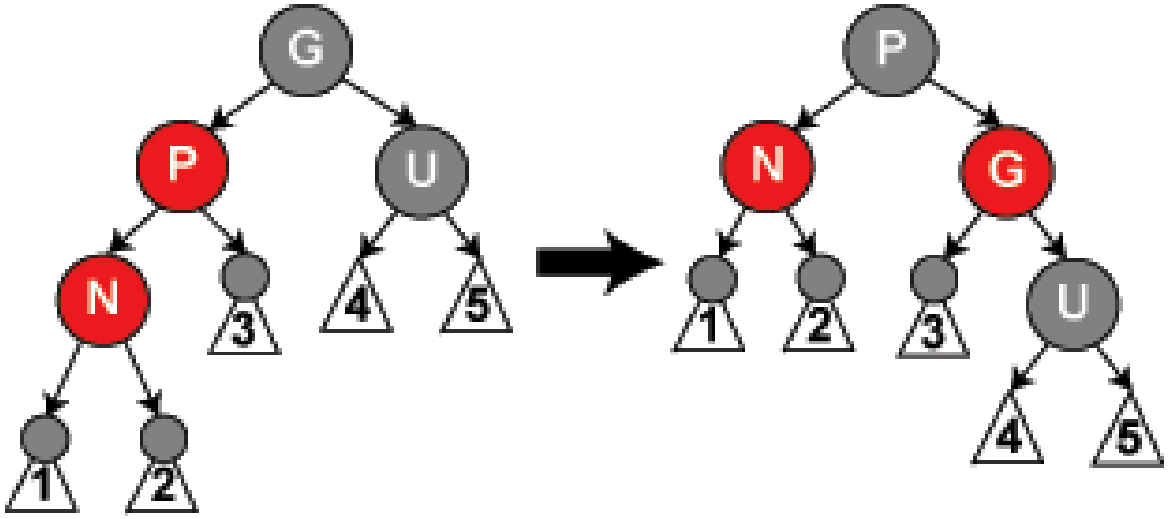
Árvores Rubronegras

- Inserção (pai vermelho, tio negro e nó é filho direito de filho esquerdo)



Árvores Rubronegras

- Inserção (pai vermelho, tio negro e nó é filho esquerdo de filho esquerdo)

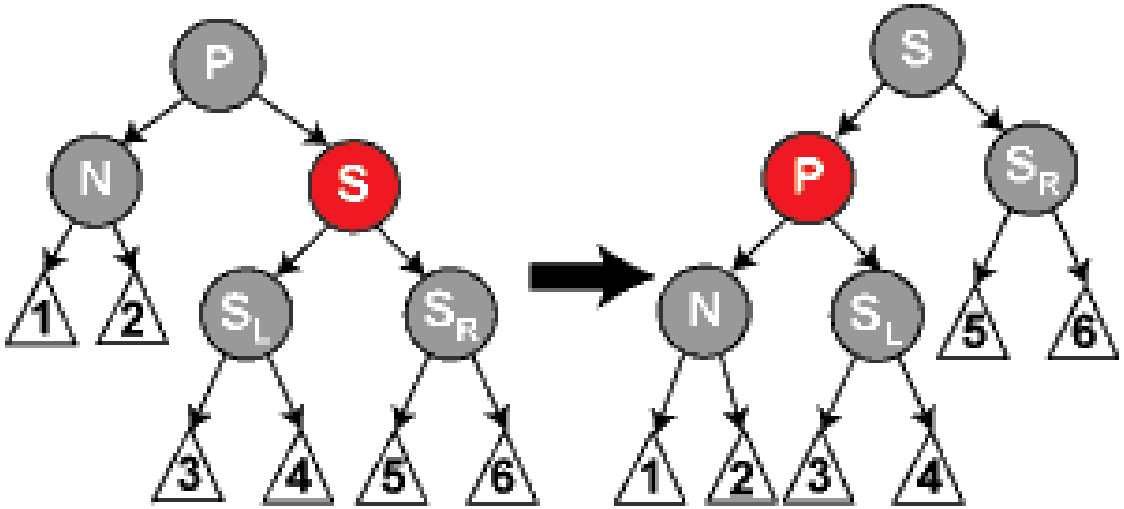


Árvores Rubronegras

- Remoção é feita através da troca do nó a ser removido por nó imediatamente inferior ou superior (como numa árvore binária comum)
- A seguir remove-se a cópia original do nó promovido, trocando-o por seu filho (chamado de N)
- Se N for a nova raiz, não há nada mais a fazer

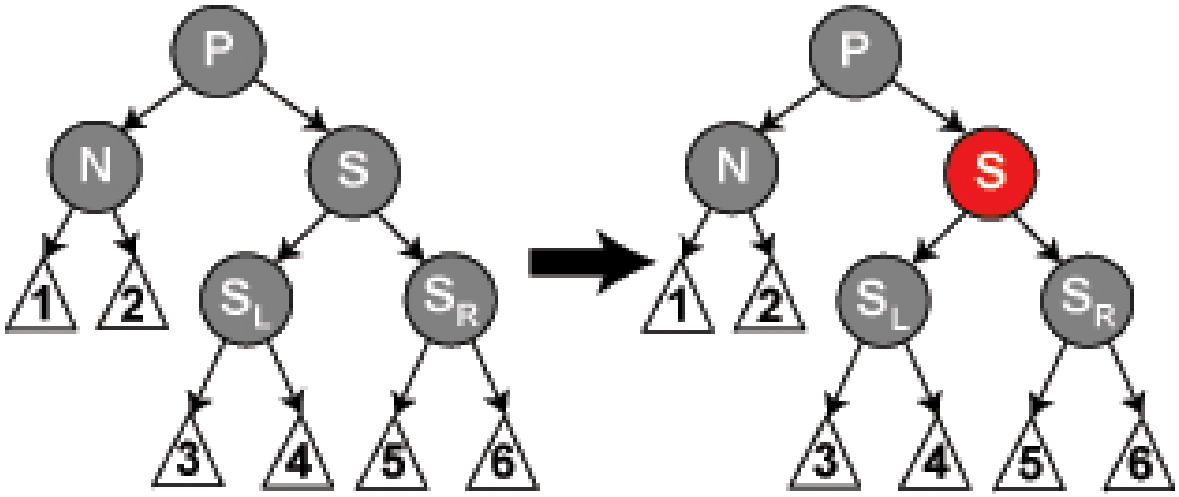
Árvores Rubronegras

- Remoção (irmão é vermelho)



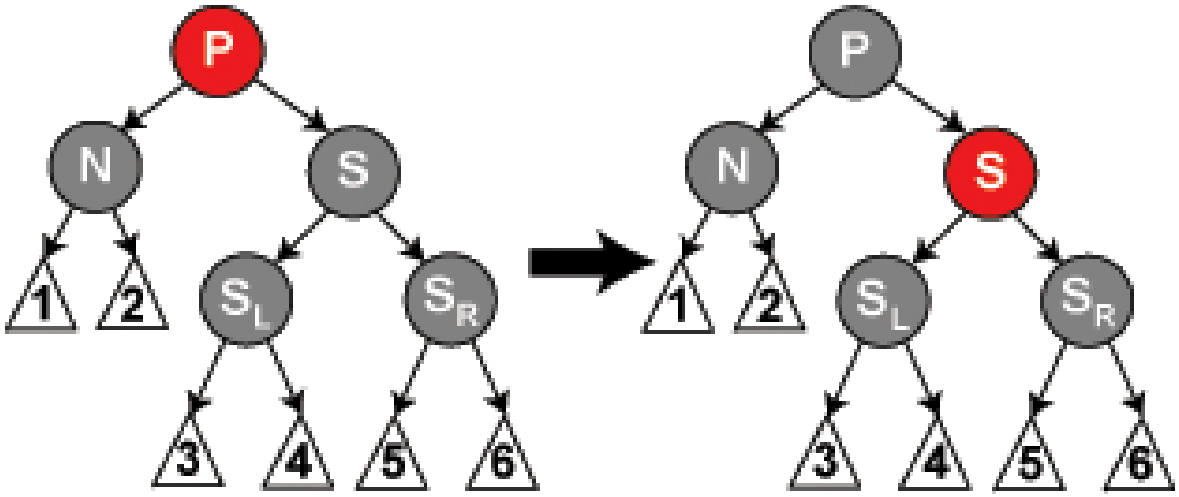
Árvores Rubronegras

- Remoção (pai, irmão e sobrinhos são negros)



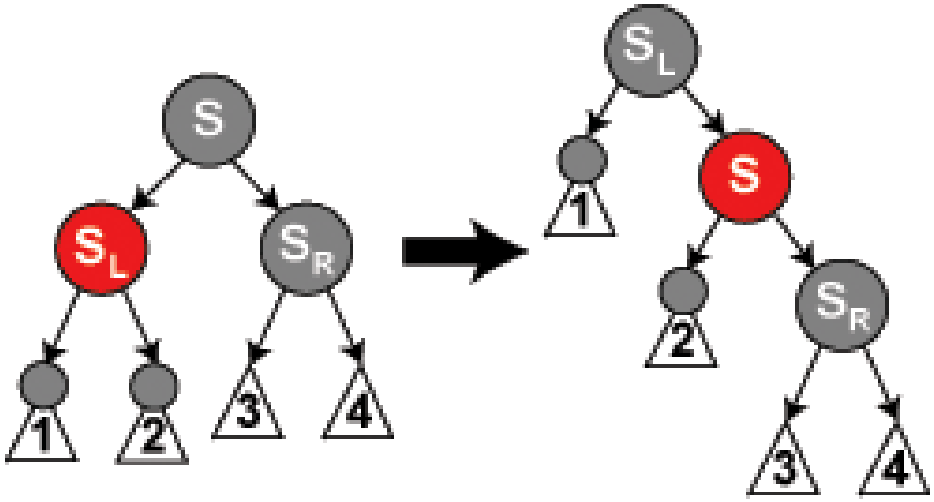
Árvores Rubronegras

- Remoção (pai negro, irmão e sobrinhos são vermelhos)



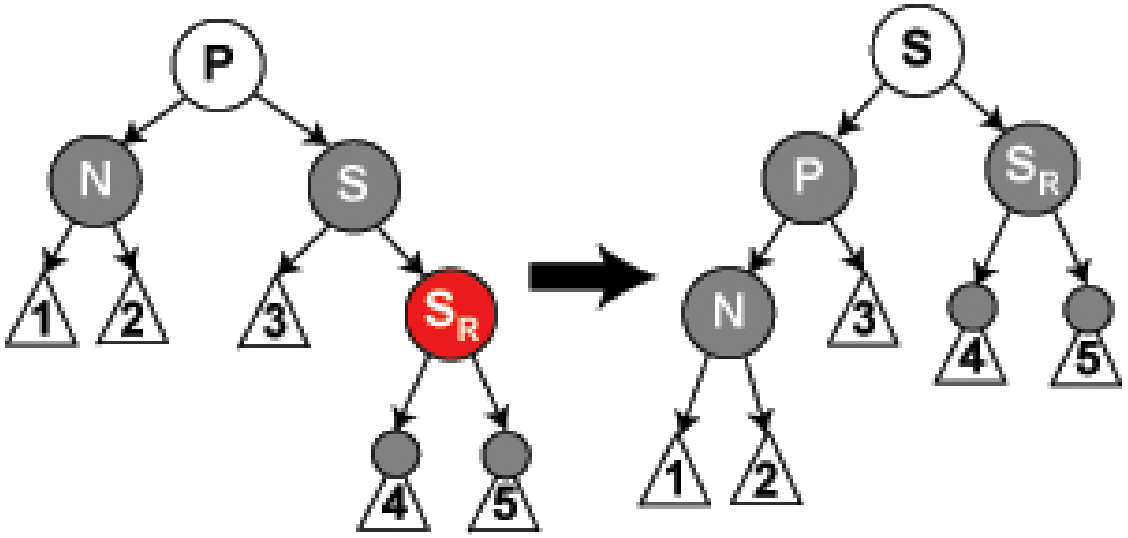
Árvores Rubronegras

- Remoção (irmão negro, sobrinho esquerdo vermelho, sobrinho direito negro e N é filho esquerdo)



Árvores Rubronegras

- Remoção (irmão negro, sobrinho direito vermelho e N é filho esquerdo)



Árvores de afunilamento (splay trees)

- O balanceamento de árvores parte do princípio de que as operações de busca, inserção e remoção ocorrem de forma uniformemente aleatória

- Isso falha na maioria dos sistemas de armazenamento de informações, quando alguns elementos são requisitados numa freqüência bem maior do que outros

Árvores de afunilamento (splay trees)

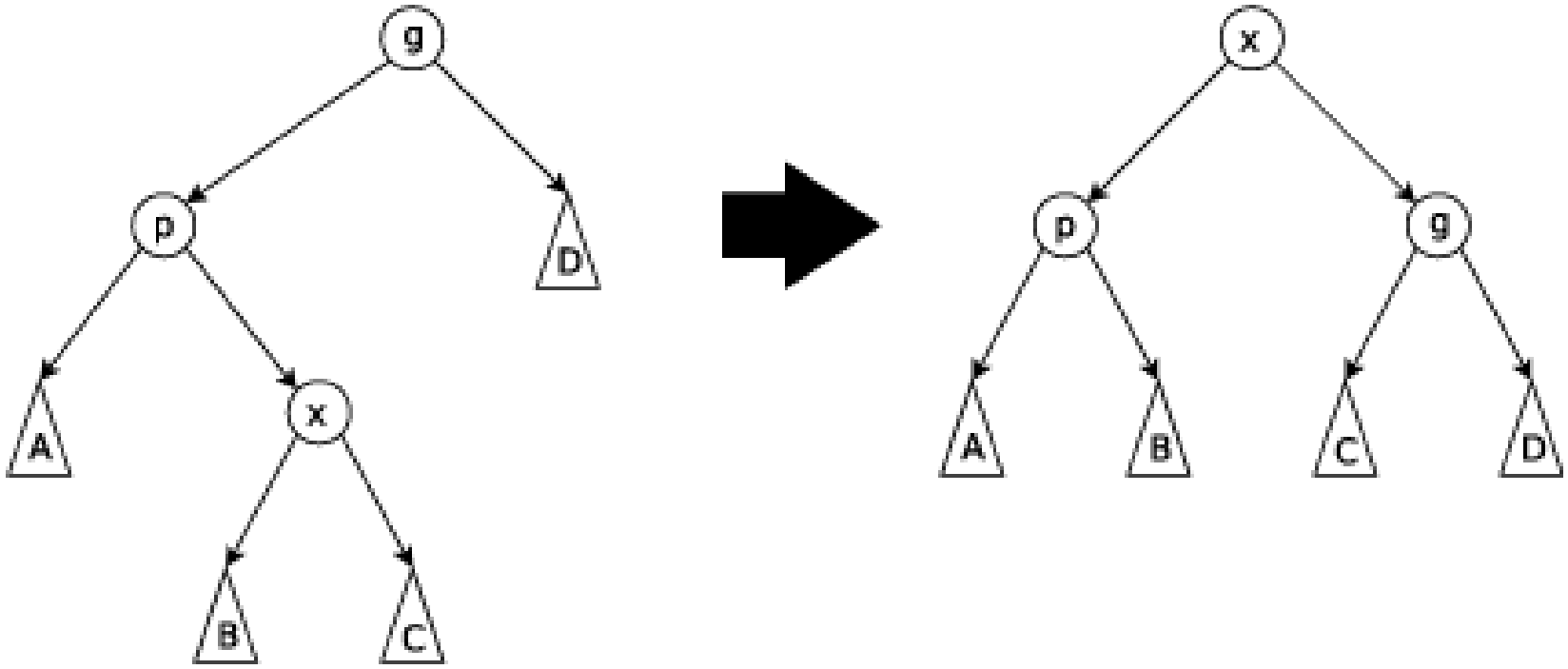
- Nesses casos é interessante que a árvore tenha os elementos mais freqüentes mais próximos da raiz

- Isso é obtido com árvores de afunilamento, ou splay trees

- Nelas os nós acessados são movidos para cima através de rotações

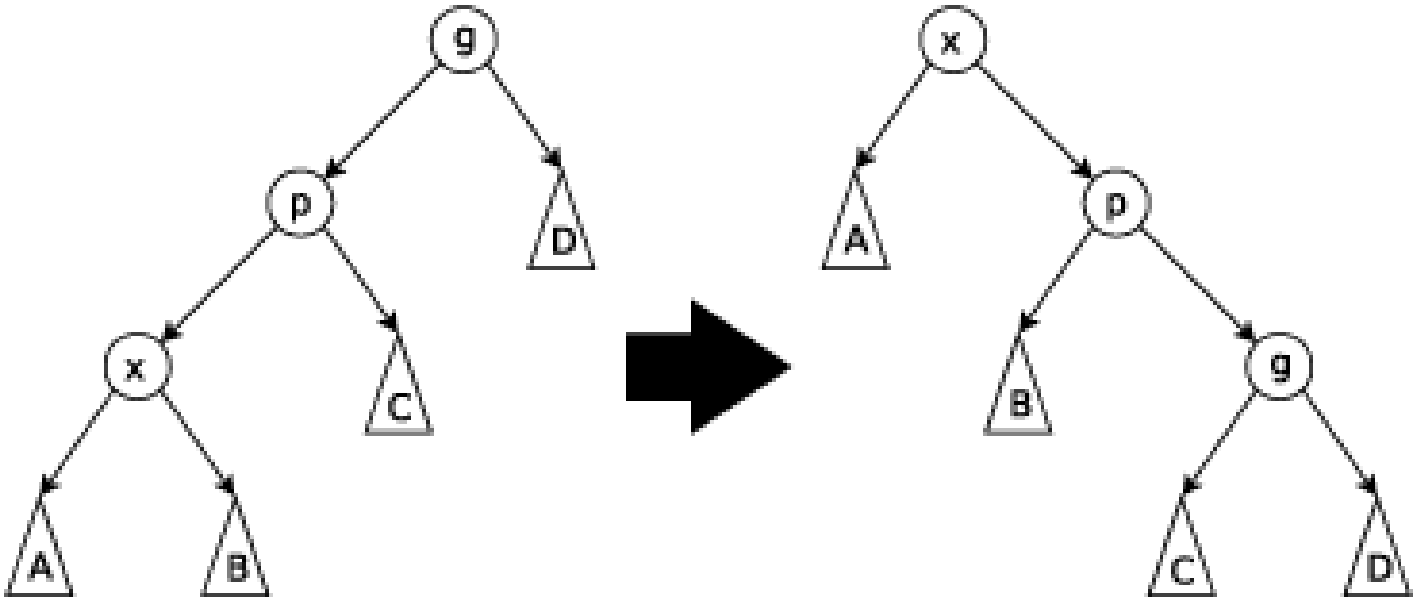
Rotações em Splay Trees

- Nó acessado é filho direito de filho esquerdo



Rotações em Splay Trees

- Nó acessado é filho esquerdo de filho esquerdo



Rotações em Splay Trees

- Nó acessado não tem avô