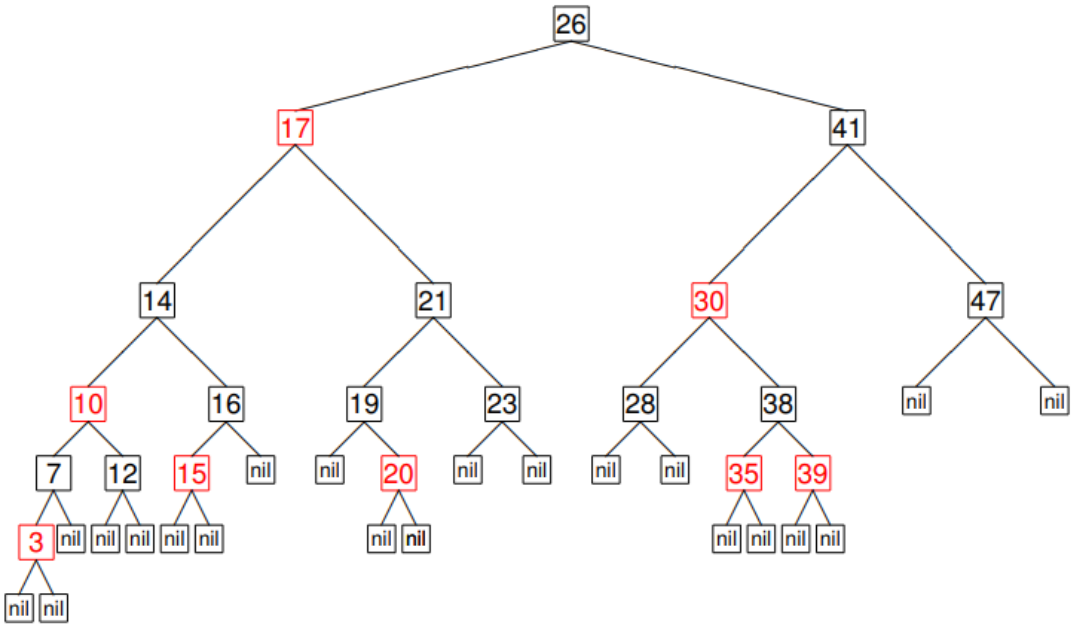
Árvore Rubro-Negra

Definição:

Uma árvore rubro-negra é uma árvore binária de busca auto balanceada, projetada para busca de dados na memória principal (RAM), em que cada nó possui os atributos abaixo:

- cor (1 bit): pode ser vermelho ou preto.
- key (e.g. inteiro): indica o valor de uma chave.
- left, right: ponteiros que apontam para a subárvore esquerda e direita, resp.
- pai: ponteiro que aponta para o nó pai. O campo pai do nó raiz aponta para nil.

Exemplo:



Uma árvore rubro-negra é uma árvore binária de busca, com algumas propriedades adicionais. Quando um nó não possui um filho (esquerdo ou direito), então vamos supor que ao invés de apontar para nil, ele

aponta para um nó fictício, que será uma folha da árvore. Assim, todos os nós internos contêm chaves e todas as folhas são nós fictícios.
As propriedades da árvore rubro-negra são:

1. Todo nó da árvore ou é vermelho ou é preto.
2. A raiz é **sempre** preta;
3. Toda folha (nil - leaf) é preta;
4. Se um nó é vermelho, então ambos os filhos são pretos.
5. Para todo nó, todos os caminhos do nó até as folhas descendentes contêm o mesmo número de nós pretos;
6. É uma BST auto balanceada.

Características:

A altura h de uma árvore rubro-negra de n chaves ou nós internos é no máximo 2 log(n + 1).

Complexidade:

Complexities in big O notation		
	Space complexity	
Space	$O(n)$	
	Time complexity	
Function	Amortized	Worst Case
Search	$O(\log n)$[1]	$O(\log n)$[1]
Insert	$O(1)$[2]	$O(\log n)$[1]
Delete	$O(1)$[2]	$O(\log n)$[1]

Diferenças entre RB e AVL

- AVL requer mais rotações para ser balanceada;
- Rubro negra: máximo de duas rotações para cada balanceamento;
- A busca é mais rápida em uma AVL, já que é estritamente balanceada;
- Inserção e deleção são mais rápidas na rubro negra, já que requer menos rotações.

Busca

1. Começando pelo nó raiz da árvore, verificar se o valor do nó a ser buscado é maior ou menor do que o nó atual
2. Se for menor, o nó atual passa a ser o filho esquerdo do nó antigo (nesse caso, o raiz)
3. Se for maior, o nó atual passa a ser o filho direito do nó antigo
4. Realizar esse procedimento até encontrar (ou não) o nó desejado
5. Retornar o valor booleano da busca

Código:

```
 private boolean searchNode(RedBlackNode node, int value) {
        boolean check = false;
        while ((node != nullNode) && check != true)
        {
            int nodeValue = node.element;
            if (value < nodeValue)
                node = node.leftChild;
            else if (value > nodeValue)
                node = node.rightChild;
            else
            {
                check = true;
                break;
            }
            check = searchNode(node, value);
        }
        return check;
    }
```

Inserção

1. Seja x e y nós raiz e folha da árvore, respectivamente
2. Checar se o nó raiz está vazio ou não. Se sim, o nó inserido será adicionado como nó raiz de cor preta
3. Se não, comparar o nó raiz com o novo nó. Se o novo nó for maior que a raiz, percorrer pela subárvore direita. Se não, pela esquerda
4. Repetir passo 3 até chegar na folha
5. Fazer o pai do nó raiz também ser pai do novo nó
6. Se o valor do nó folha for menor que o do novo nó, novo nó será filho esquerdo. Se for maior, filho direito
7. Fazer filhos do novo nó como sendo nulos
8. Novo nó será vermelho
9. Restaurar as propriedades da árvore performando rotações ou mudando as cores dos nós.

Algoritmo para manutenção das propriedades da árvore

1. Performar os passos até que o pai p do nó inserido seja vermelho
2. Se p for filho esquerdo do nó avô do nó inserido
   2.1. Caso 1
   - Quando a cor do filho direito do nó avô do nó inserido for vermelho, transforme a cor de ambos os filhos do avô para preto e faça avô ficar vermelho
   - Designar o nó avô para o nó inserido

   2.2. Caso 2
   - Se não, se o nó inserido for o filho direito do pai, designar p para o nó inserido e realizar rotação esquerda

   2.3. Caso 3
   - Transformar pai para preto e avô para vermelho e performar rotação direita para o nó inserido
3. Se p não for filho esquerdo do nó avô
   3.1. Se o filho esquerdo do avô for vermelho, transformar ambos os filhos do avô em preto e avô em vermelho

3.2. Designar avô para o nó inserido
3.3. Se não, se o nó inserido for filho esquerdo do pai, designar nó pai para o filho e performar rotação direita
3.4. Transformar nó pai para preto e avô para vermelho
3.5. Realizar rotação esquerda para nó avô
4. Fazer nó raiz preto

Remoção

Algoritmo para remoção de nó

1. Realizar a deleção padrão de árvore binária de busca. Fazendo isso, sempre é deletado um nó que ou é folha ou só tem um filho (se for interno, copia-se o sucessor e recursivamente chama a remoção para o sucessor, sendo o sucessor sempre folha ou nó com um filho). Então só é necessário cuidar de casos em que o nó é folha ou só tem um filho. Seja v o nó a ser deletado e u o filho que o substitui (u será nulo quando v for folha, e nulo é sempre preto)
2. Caso simples - Se ou u ou v são vermelho, deleta-se v e marca-se u como preto
3. Se ambos u e v forem preto

   3.1. Colorir u como double black. Agora deve-se converter double black em apenas black

   3.2. Fazer os seguintes passos enquanto u for double black e não for raiz. Seja s o irmão de u

   3.2.1 Se s for preto e um de seus filhos for vermelho, realizar rotação. Seja r o filho vermelho de s. Existem quatro possíveis alternativas dependendo das posições

   a) Caso esquerda esquerda - s é filho esquerdo de seu pai e r é filho direito de s ou ambos os filhos de s são vermelhos.

Aqui, o pai p substitui o nó deletado, nó irmão s se torna novo pai e o filho esquerdo r substitui s

b) Caso esquerda direita - s é filho esquerdo de seu pai e r é filho direito de s. Nessa situação, filho r substitui s e s se torna filho esquerdo de r. Após isso, r se torna pai de p e s.

c) Caso direita direita - s é filho direito de seu pai e r é filho direito de s ou ambos os filhos de s são vermelhos. Nesse caso, o pai p substitui o nó deletado, nó irmão s se torna novo pai e o filho direito r substitui s

d) Caso direita esquerda - s é filho direito de seu pai e r é filho esquerdo de s. Nesse caso, filho r substitui s e s se torna filho direito de r. Após isso, r se torna pai de p e s.

3.2.2 Se s for preto e ambos seus filhos são preto, recolorir e recorrer ao pai se ele for preto

3.2.3 Se s for vermelho, performar rotação para levantar o antigo irmão, recolorir ele e o pai. O novo irmão é sempre preto. Existem duas alternativas para esse caso

a) Caso esquerda - s é filho esquerdo do pai. Rotacionar para direita o pai p

b) Caso direita - s é filho direito do pai. Rotacionar para esquerda o pai p

3.3. Se u for raiz, colorí-lo para apenas black e retornar (altura de pretos da árvore reduz em 1).