Árvores Equilibradas

Sumário

- Splay
- Vermelho-Preto
- AA e BB
- Multidimensionais
  - quaternárias
  - k-d
- Pesquisa Lexicográfica
  - tries multivia
  - tries binárias
  - PATRICIA

Árvores Equilibradas

Sumário

- Árvores AVL
- Árvores B
- Splay
- Pesquisa Lexicográfica
  - tries multivia
  - tries binárias

Árvores equilibradas

- **Árvore de pesquisa binária não garante acesso logarítmico**
  - Inserção e eliminação simples podem criar árvores desequilibradas
  - Pior caso é linear: árvore degenera em lista ligada
  - Pior caso ocorre tipicamente para inserções ordenadas
- **Árvores equilibradas**
  - Evitam casos degenerados
  - Garantem O(log N) para operações de inserção, remoção e pesquisa
  - Requerem algoritmos mais elaborados para inserção e remoção
- **Condição adicional na árvore**
  - condição de equilíbrio, garante que nenhum nó está demasiado profundo

Árvores AVL

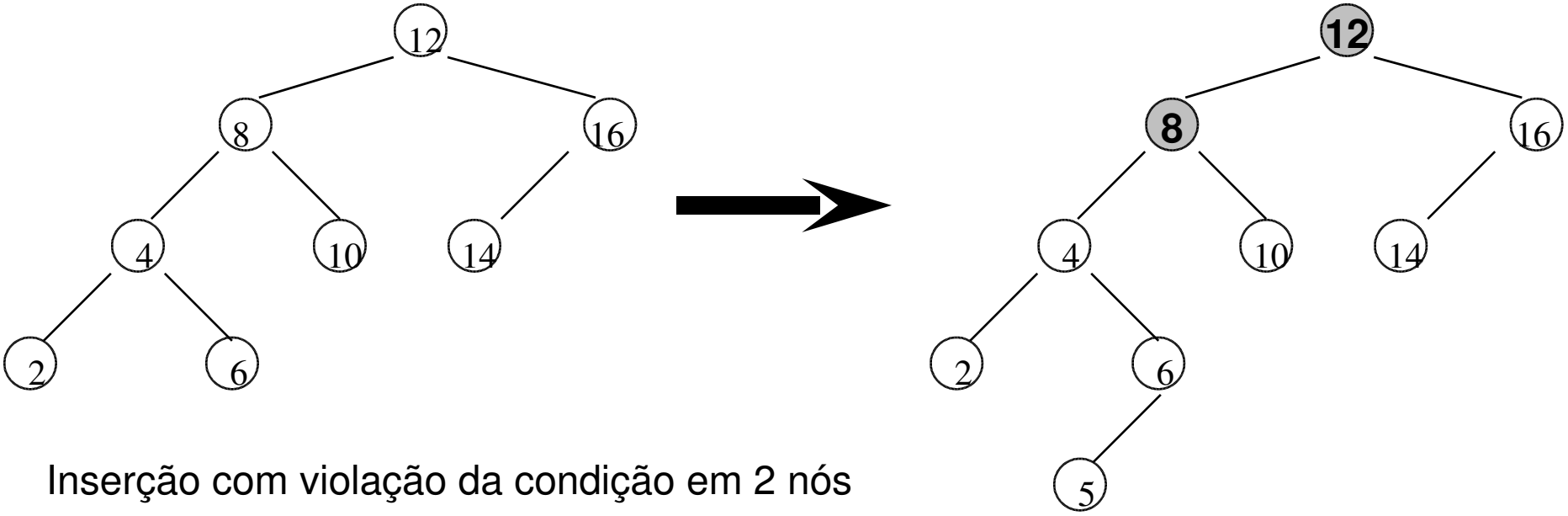
- **Adelson-Velskii e Landis, 1962**
- **Condição de equilíbrio: na altura das sub-árvores de cada nó**
  - diferença de alturas não pode exceder 1
  - garante altura logarítmica para a árvore
  - é simples de manter
- **Definição**
  - Uma árvore AVL é uma árvore de pesquisa binária que respeita a seguinte condição de equilíbrio: para qualquer nó da árvore, as alturas das sub-árvores esquerda e direita diferem no máximo de 1 unidade.
- **Altura de uma árvore**
  - 1 + altura da sua sub-árvore mais alta
  - 0 para árvore só com 1 nó
  - -1 para árvore vazia

Número de nós na árvore AVL

- **Uma árvore AVL de altura H tem pelo menos F_{H+3} -1 nós, em que F_i é o número de Fibonacci de ordem i**
  - S_H: tamanho da menor árvore AVL de altura H ($S_0 = 1$, $S_1 = 2$)
  - A árvore mais pequena de altura H tem sub-árvores de alturas H-1 e H-2
  - Cada sub-árvore terá, por sua vez, o número mínimo de nós para a sua altura
  - Então será $S_H = S_{H-1} + S_{H-2} + 1$
  - $S_H = F_{H+3} - 1$, por indução:
    - $S_0 = 1$, é $F_3 - 1$
    - Se $S_{H-1} = F_{H+2} - 1$ e $S_{H-2} = F_{H+1} - 1$,

      então $S_H = S_{H-1} + S_{H-2} + 1 = F_{H+2} - 1 + F_{H+1} - 1 + 1 = F_{H+2} + F_{H+1} - 1 = F_{H+3} - 1$
- **$F_i \approx \phi^i / \sqrt{5}$, com $\phi = (1+\sqrt{5})/2 \approx 1.618$**
  - árvore de altura H tem no mínimo $\phi^{H+3} / \sqrt{5}$ nós
  - $H < 1.44 \log (N+2) - 1.328$ (não mais de 44% acima da mínima)

Árvores AVL



Inserção com violação da condição em 2 nós

- **Inserções e remoções podem destruir o equilíbrio de alguns dos nós da árvore**
  - Necessário verificar condição e reequilibrar se tiver sido destruída

Inserção em Árvores AVL

- **Após uma inserção, só os nós no caminho da raiz ao ponto de inserção podem ter a condição de equilíbrio alterada**
  - condição só depende das alturas das sub-árvores de um nó
- **Para reequilibrar: subir no caminho até à raiz**
  - reequilibrar o nó mais profundo onde surge desequilíbrio
  - toda a árvore resulta equilibrada
- **X: nó a reequilibrar devido a inserção em**

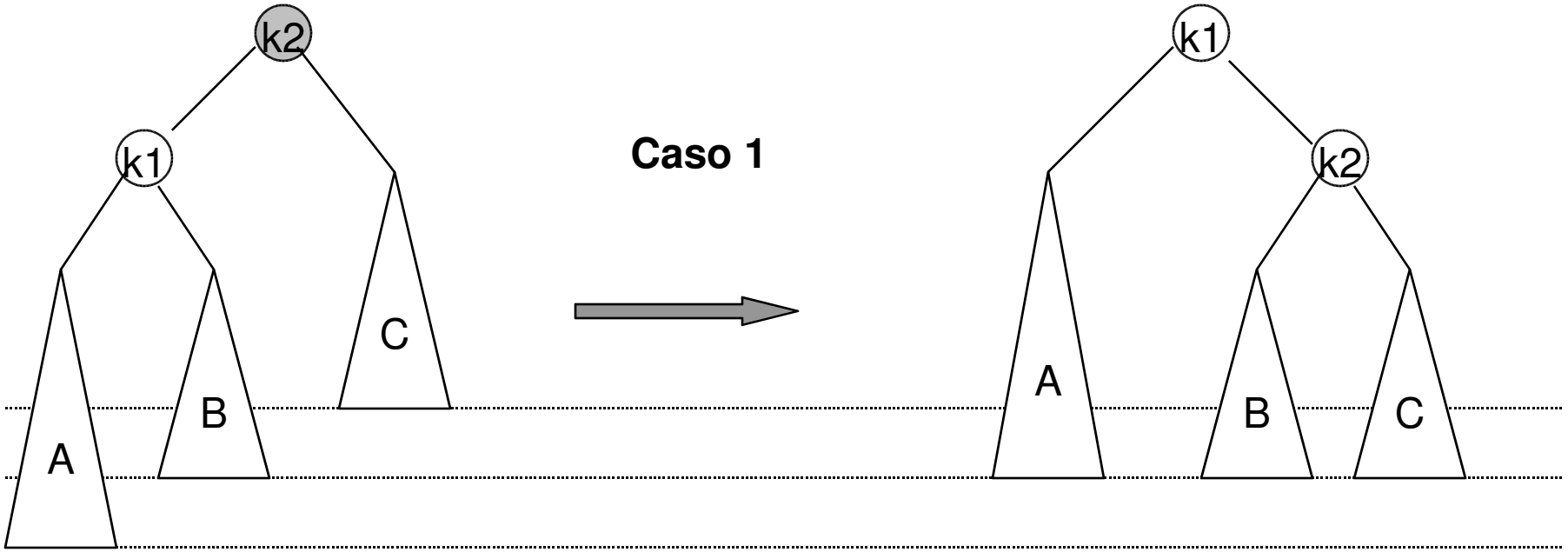
  1- árvore esquerda do filho esquerdo de X

  2- árvore direita do filho esquerdo de X

  3- árvore esquerda do filho direito de X

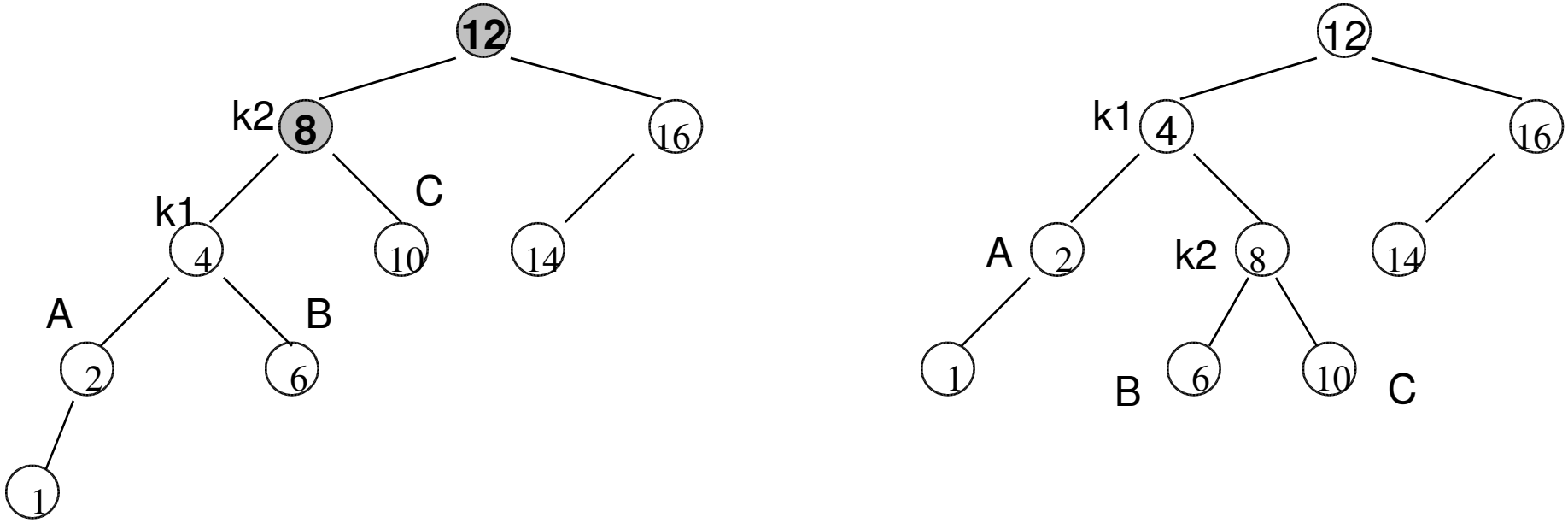
  4- árvore direita do filho direito de X
- **Casos 1 e 4 simétricos; casos 2 e 3 simétricos**

Rotação simples



- **k2 é nó mais profundo onde falha o equilíbrio**
  - sub-árvore esquerda está 2 níveis abaixo da direita
    - B não está no mesmo nível de A, ou k2 estaria desequilibrado antes da inserção
    - B não está no mesmo nível que C, ou k1 seria nó desequilibrado mais fundo

Rotação simples



- **Árvore resultante da rotação é AVL**
  - k1 e k2 passam a ter subárvores da mesma altura
  - nova altura da árvore resultante é igual á da árvore anterior à inserção
  - problema fica resolvido com uma só operação

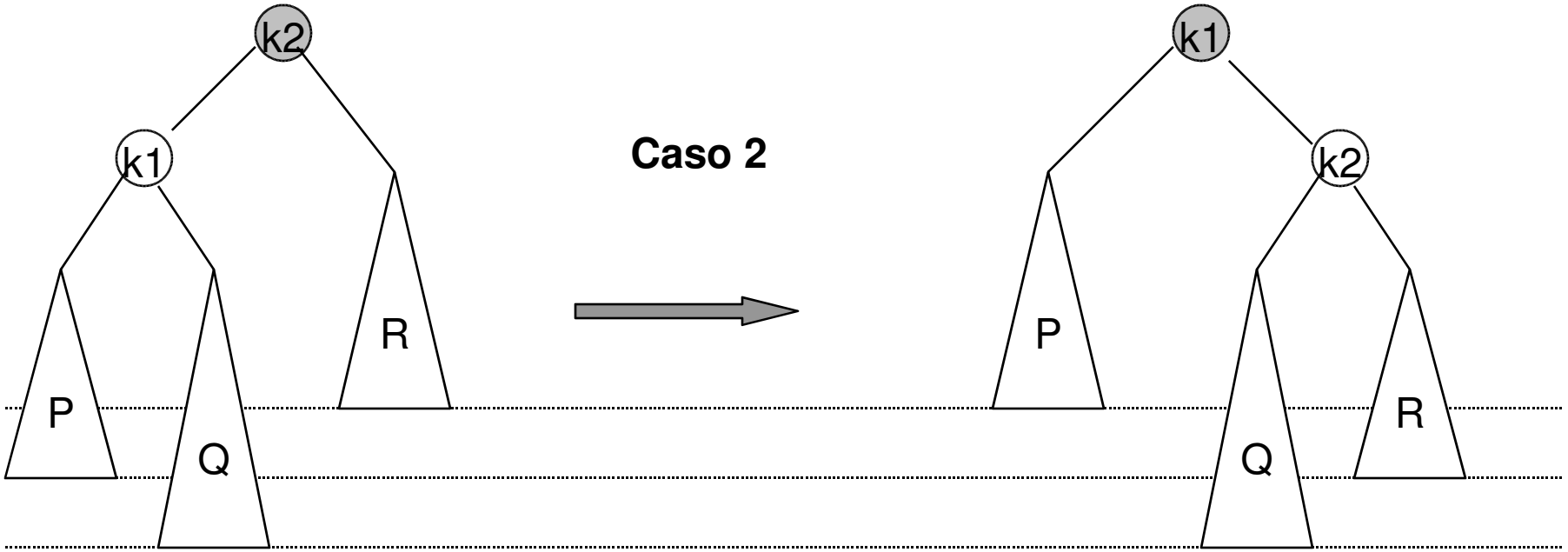
Rotação simples com filho esquerdo

```
/**
     * Rotate binary tree node with left child.
     * For AVL trees, this is a single rotation
     * for case 1.
     */
    static BinaryNode withLeftChild( BinaryNode
    {
        BinaryNode k1 = k2.left;
        k2.left = k1.right;
        k1.right = k2;
        return k1;
    }
```

Rotação simples com filho direito

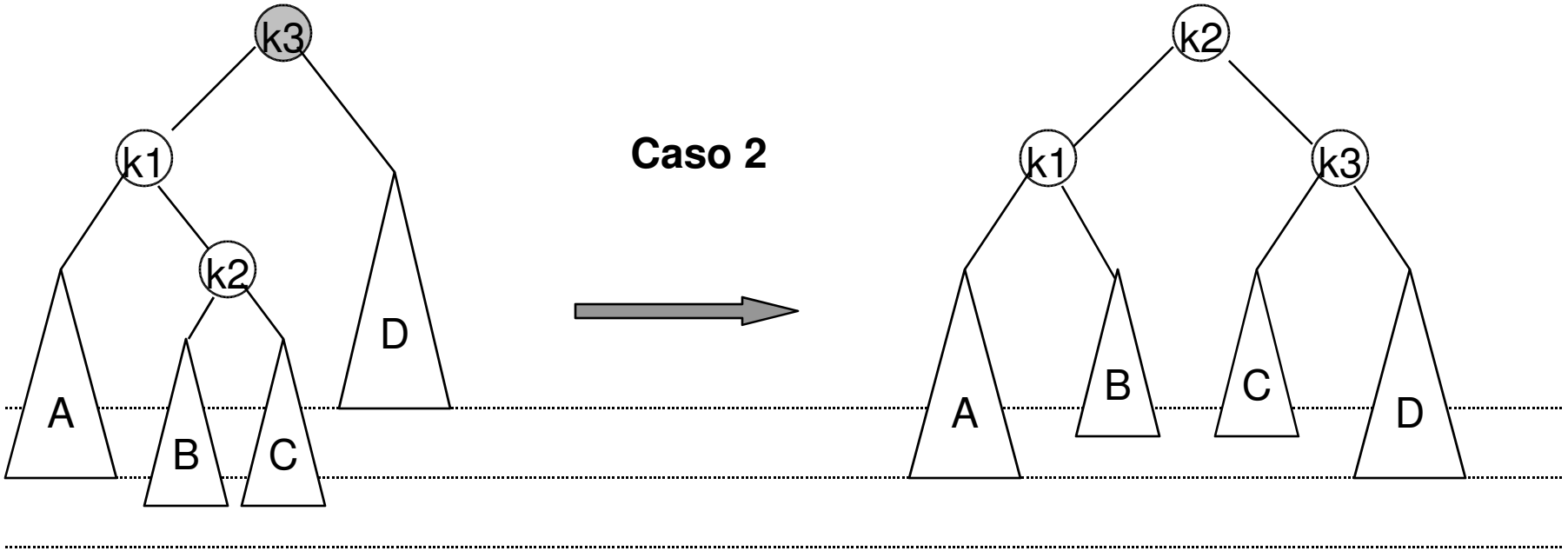
```
/**
  * Rotate binary tree node with right child
  * For AVL trees, this is a single rotation
  * for case 4.
  */
 static BinaryNode withRightChild( BinaryNod
 {
     BinaryNode k2 = k1.right;
     k1.right = k2.left;
     k2.left = k1;
     return k2;
 }
```

Rotação simples no caso 2



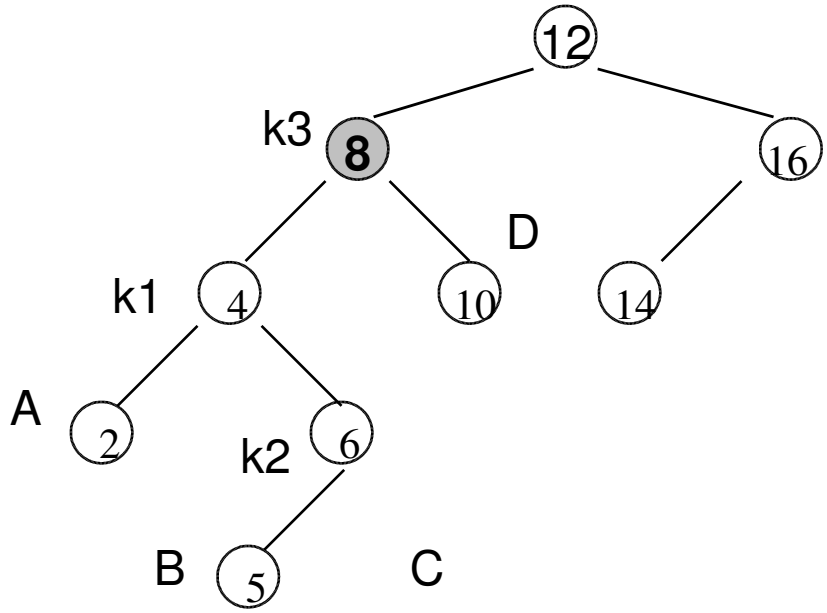
- **Rotação simples não resolve o desequilíbrio!**
  - sub-árvore Q está a 2 níveis de diferença de R
  - sub-árvore Q passa a estar a 2 níveis de diferença de P

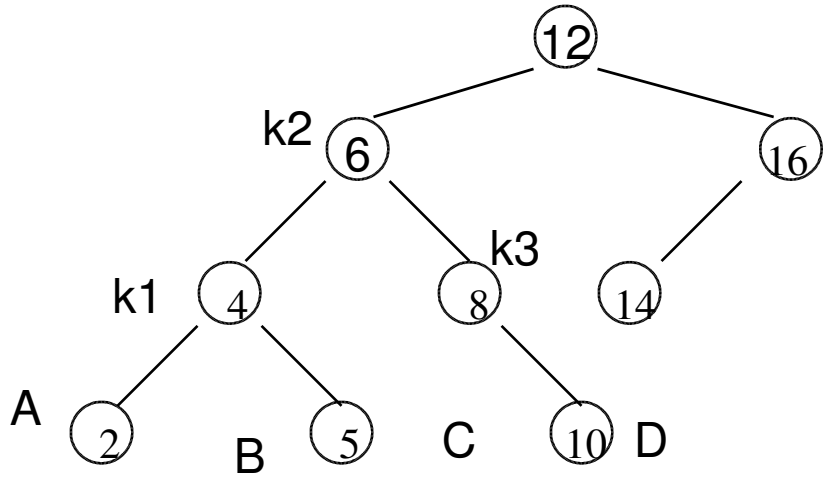
Rotação dupla no caso 2



- **Uma das subárvores B ou C está 2 níveis abaixo de D (e só uma)**
  - k2, a chave intermédia, fica na raiz
  - posições de k1, k3 e subárvores completamente determinadas pela ordenação

Rotação dupla





- **Rotação dupla pode ser vista como sequência de 2 rotações simples**
  - rotação entre o filho e o neto de X
  - rotação entre X e o seu novo filho

Rotação dupla com filho esquerdo

```
/**
 * Double rotate binary tree node: first le
 * with its right child; then node k3 with
 * left child.
 * For AVL trees, this is a double rotation
 * case 2.
 */
static BinaryNode doubleWithLeftChild( Bina
{
    k3.left = withRightChild( k3.left );
    return withLeftChild( k3 );
}
```

Rotação dupla com filho direito

```
    /**
     * Double rotate binary tree node: first ri
     * with its left child; then node k1 with ne
     * right child.
     * For AVL trees, this is a double rotation
     * case 3.
     */
    static BinaryNode doubleWithRightChild(Bina
    {
        k1.right = withLeftChild( k1.right );
        return withRightChild( k1 );
    }
}
```

Inserção em árvore AVL

- **Algoritmo recursivo**
  - Inserir nó com chave X numa árvore A
    - recursivamente, inserir na subárvore conveniente de A, SA
    - se a altura de AS não se modifica: terminar
    - se a altura de AS é modificada: se ocorre desequilíbrio em A, fazer as rotações necessárias para reequilibrar
  - Comparação de alturas
    - requer cálculo repetido de alturas das árvores: preferível manter o resultado da comparação como um factor de equilíbrio
- **Algoritmo iterativo**
  - Especificar paragem logo que uma rotação é realizada