Pesquisa em memória primária

Pesquisa em memória primária

Recuperar informação a partir de uma grande massa de informação previamente armazenada.

Existem vários métodos de pesquisa, depende de:

- Tamanho (Quantidade de dados)
- Estabilidade (Os dados estarem sujeito a variações - retirado/inclusão)

- Pesquisa Sequencial
- Pesquisa Binária
- Árvores de Pesquisa

Pesquisa Sequencial

- O método de pesquisa mais simples.
  - A partir do primeiro registro, pesquise sequencialmente até encontrar a chave procurada.

Pesquisa Binária

- A pesquisa pode ser mais eficiente se os registros forem mantidos em ordem.
- Para procurar uma chave com registros em ordem podemos:
  1. Compare a chave com o registro que esta na posição do meio da tabela.
  2. Se a chave é menor, o registro procurado deve estar na primeira metade, senão deve estar na segunda metade.
  3. Repita o processo até encontrar a chave.

Pesquisa Binária

A B C D E F G H (PROCURAR G)

A B C **D** E F G H

E **F** G H

G H

Pesquisa binária

- Qual é o problema da pesquisa binária?

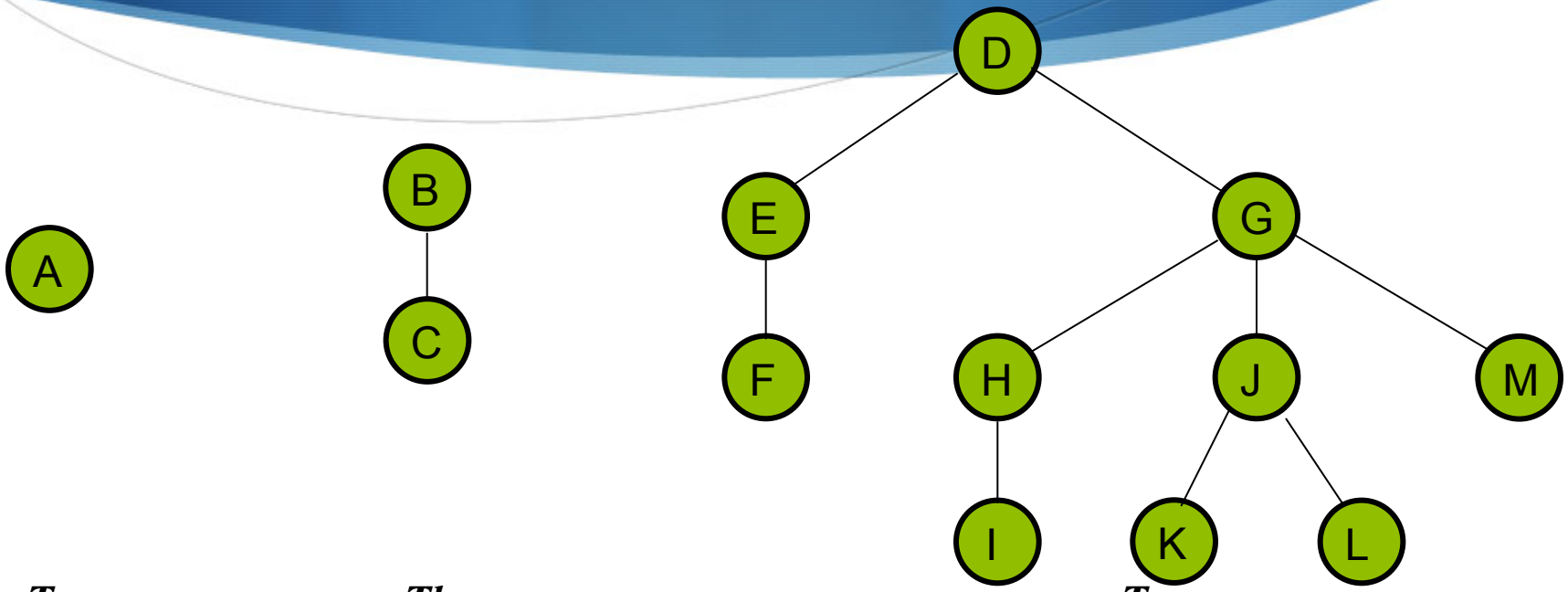
Árvores de pesquisa

- As listas encadeadas usualmente fornecem maior flexibilidade que as matrizes,
  - mas são estruturas lineares e
  - é difícil usá-las para organizar uma representação hierárquica de objetos.
- As pilhas e as filas apresentam hierarquia limitada a uma dimensão.
- As árvores são estruturas próprias para a representação de hierarquia.
- As árvores consistem de:
  - Nós e arcos.
  - A raiz, é um nó que não tem ancestrais tem somente filhos.
  - As folhas não têm descendentes.
  - As árvores são representadas de cima para baixo com a raiz no topo e as folhas na base.

Árvores

- Uma árvore T é um conjunto finito e não-vazio de nós
- T= {r} ∪ T1 ∪ T2 ∪ . . . ∪ Tn, com as seguintes propriedades:
  - um nó especial da árvore, r, é chamado de raiz da árvore; e
  - o restante dos nós é particionado n ≥ 0 subconjuntos, T1 , T2 , …, Tn , cada um dos quais sendo uma sub-árvore.
- A definição de uma árvore é recursiva, uma árvore é definida em termos dela mesmo.
- Não há o problema de recursão infinita porque toda árvore tem um número finito de nós e no caso base uma árvore tem n=0 subárvores.
  - Uma árvore minimal é composta de um único nó, a raiz.

Árvores



Ta ***Tb*** ***Tc***

Ta = {A}, Ta é uma árvore minimal

Tb = {B,{C}}

Tc = {D,{E,{F}},{G,{H,{I}},{J,{K},{L}},{M}}}

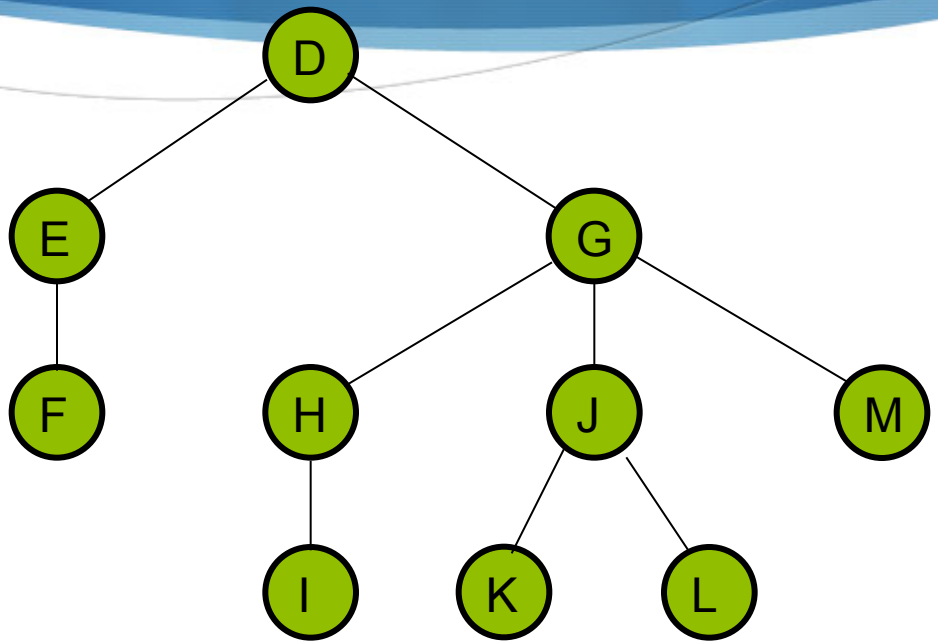
Árvores

- O grau de um nó é o número de sub-árvores relacionadas com aquele nó.
  - Um nó de grau zero não possui sub-árvores.
- Cada raiz **ri** da sub-árvore **Ti** é chamada de filho de **r**.
  - A definição de uma árvore não impõem qualquer condição sobre o número de filhos de um nó.
  - O número de filhos de um nó pode variar de 0 a qualquer inteiro.
- O nó raiz é o **pai** de todas as raízes ri das subárvores Ti, $1 \leq i \leq n$.

Árvores

- Duas raízes ri e rj das subárvores distintas Ti e Tj de Tn são ditas irmãs.
- Dada uma árvore T que contém um conjunto de nós, um caminho em T é definido como uma seqüência não vazia de nós ou arcos
  - P={r1, r2,…, rk}
  - onde ri ∈ R, para 1≤ i ≤ k tal que o i-ésimo nó da seqüência, ri,é o pai do (i+1)-ésimo nó da seqüência ri+1.
- O número de arcos em um caminho é chamado de comprimento do caminho P e é definido por k-1.

Árvores



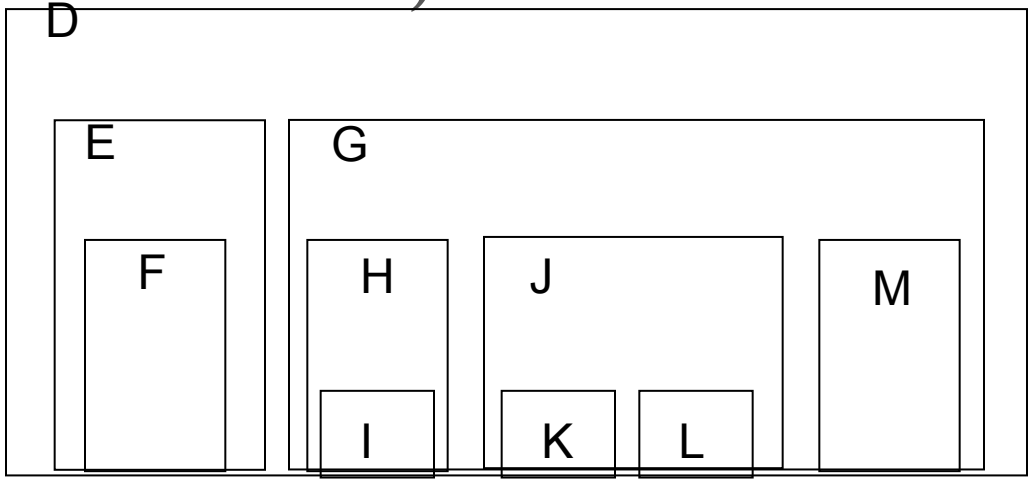
- Na árvore T há 29 caminhos diferentes incluindo os caminhos de comprimento 0; os caminhos de comprimento 1, e os caminhos de comprimento 3.

Árvores

- O nível ou profundidade de um nó ri ∈ R da árvore T é o comprimento do único caminho em T entre a raiz r e o nó ri .
  - A raiz T está no nível 0 e as raízes das subárvores estão no nível 1.
- A altura de um nó ri ∈ R numa árvore T é o comprimento do caminho mais longo do nó ri a uma folha.
  - Folhas têm altura igual a 0.
  - A altura de uma árvore T é a altura do nó raiz r.
  - A árvore vazia é uma árvore legítima de altura 0 (por definição)

Árvores

- Uma árvore pode ser representada como um conjunto de reuniões aninhadas no plano (Diagrama de Venn).



Árvores

- Definições
  - **Nó pai** – nó ao qual um nó está ligado (diretamente).
  - **Nó filho** – cada um dos nós derivados de um nó pai.
  - **Nó ancestral** – todos os nós acima de um dado nó, em direção a raiz.
  - **Nó descendente** – todos os nós abaixo de um dado nó.
  - **Nós irmãos** – nós com o mesmo pai.
  - **Grau** – número de sub-árvores de um nó.
  - **Nó terminal (folha)** – nó sem filho ou com grau zero.
  - **Nível ou profundidade** – número de arcos entre um nó e a raiz.
  - **Altura da árvore** – nível mais alto.
  - **Floresta** – conjunto de árvores disjuntas.

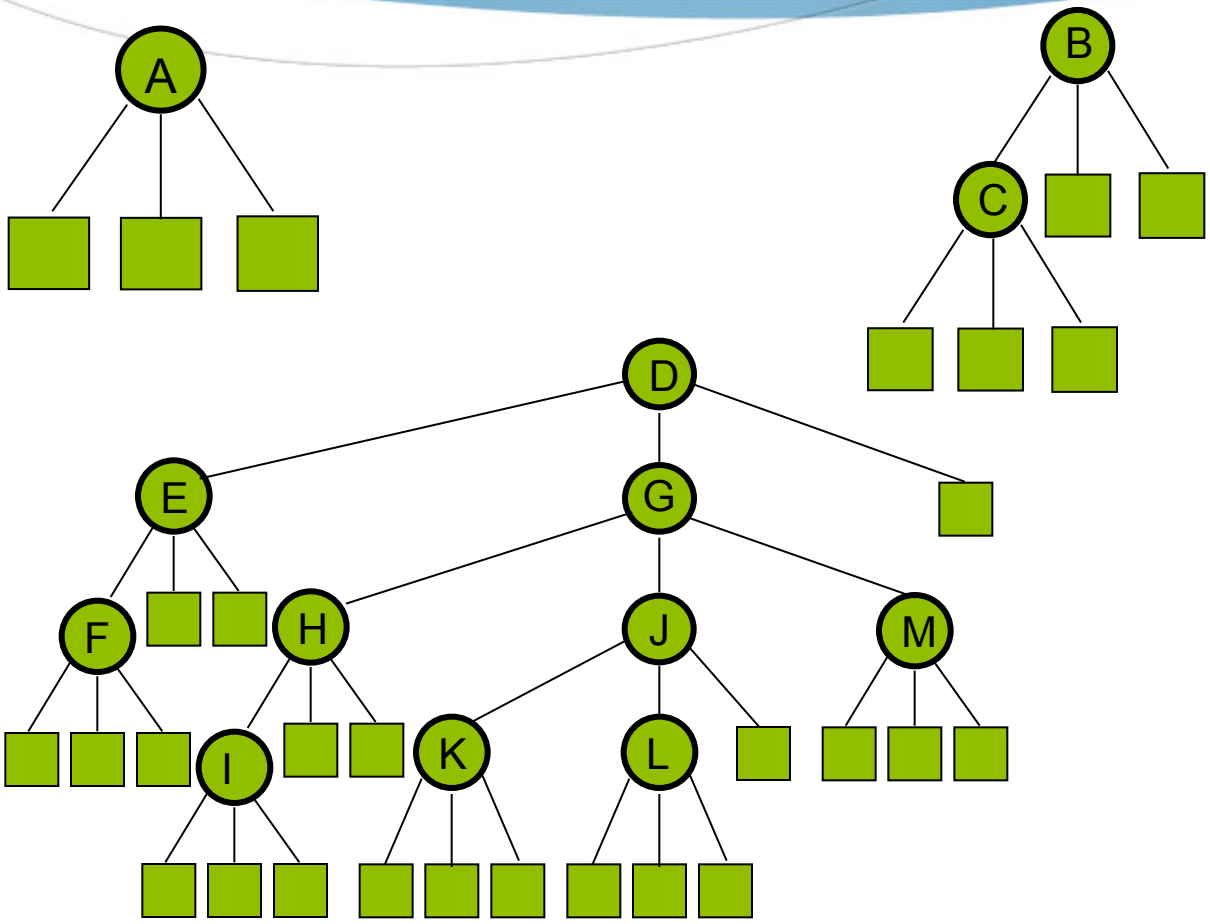
Árvores N-árias

- **Árvore N-ária:**
  - Uma variação de árvores na qual **todos os nós têm exatamente o mesmo grau**.
    - Não é possível construir uma árvore com um número finito de nós, todos com o mesmo grau N, exceto se N=0.
  - É um **conjunto finito** de nós com as seguintes propriedades:
    - O conjunto é vazio, T=∅; ou
    - O conjunto consiste numa raiz, R, e exatamente N árvores N-árias distintas.
    - Os nós remanescentes podem ser particionados em N ≥ 0 subconjuntos, T0, T1,…, TN-1, cada um deles uma árvore N-ária tal que T={R, T0, T1,…, TN-1 }.
  - Os nós de uma árvore N-ária possuem grau **0 ou N**.

Árvores N-árias

- A árvore vazia T=∅ é uma árvore, um objeto do mesmo tipo que a árvore não vazia.
  - Num projeto orientado a objetos, uma árvore vazia deve ser a instância de alguma classe de objetos.
- **Nós externos:** não possuem subárvores, árvores vazias, aparecem nas extremidades das árvores.
- **Nós internos:** árvores não vazias (possuem subárvores).

Árvores ternárias



Árvores N-árias

- Os quadrados representam árvores vazias e os círculos denotam os nós não vazios.
- Árvores ordenadas:
  - árvores cujas sub-árvores estão ordenadas.
- Árvores orientadas:
  - árvores cuja ordenação não é importante.

Árvores Binárias

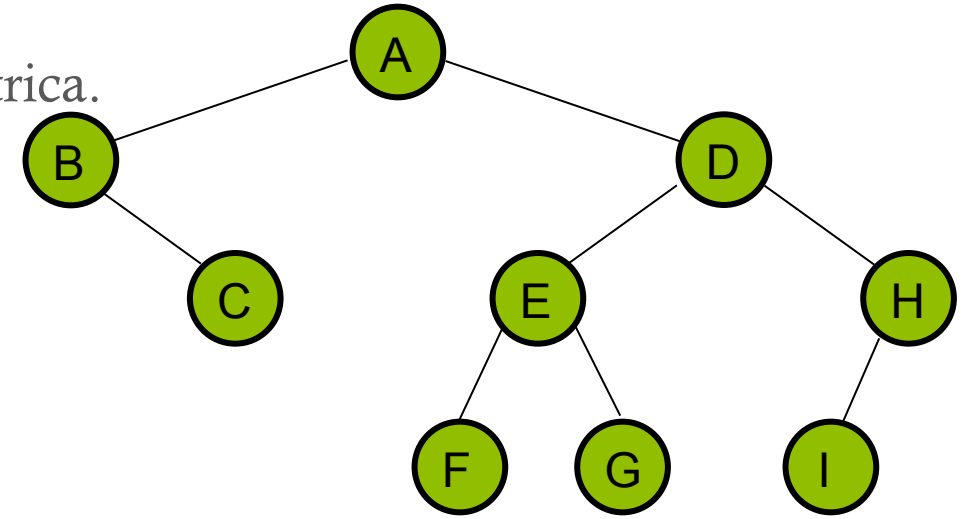
- Uma **árvore binária** é uma árvore **N-ária** para a qual **N=2**.
- Uma árvore binária **T** é um conjunto finito de nós com as seguintes propriedades:
  - O conjunto é vazio, T=∅; ou
  - O conjunto consiste em uma raiz, **r**, e em exatamente duas árvores binárias distintas TL e TR, T={r, TL , TR}.
- A árvore **TL** é dita a subárvore da esquerda e a árvore **TR** é dita a subárvore da direita.
- Uma árvore binária com n ≥ 0 nós internos contém n+1 nós externos,
  - árvore N-ária contém (N-1)n+1 nós externos.

Pesquisa em Árvores

- Algoritmos para a manipulação de árvores têm a característica comum:
  - Visitar sistematicamente todos os nós das árvores.
- O processo de visitação dos nós da árvore é chamado de **pesquisa em árvore**.
- Há duas formas para a realização de um pesquisa:
  - Pesquisa em **profundidade** e
  - Pesquisa em **largura**.

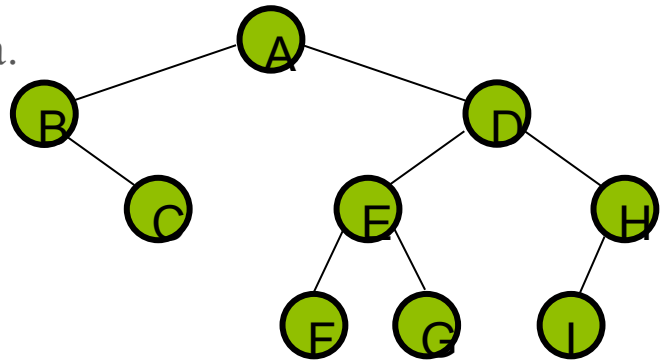
Pesquisa em profundidade

- Alguns percursos em profundidade possuem nomes específicos:
  - Pesquisa em pré-ordem,
  - Pesquisa em pós-ordem e
  - Pesquisa em ordem simétrica.



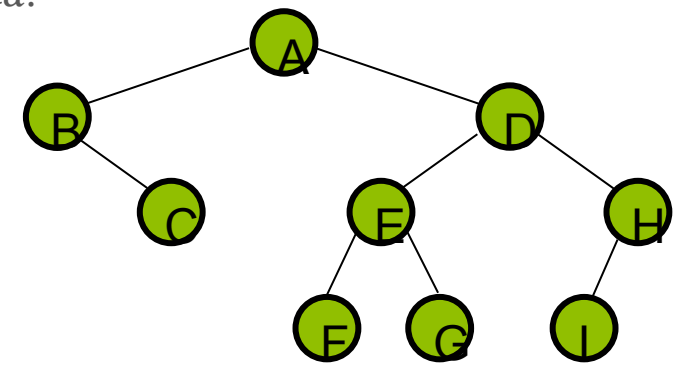
Pesquisa em profundidade

- Pesquisa em **pré-ordem**
  - A raiz é primeiro nó a ser visitado.
  - A pesquisa é definida recursivamente.
    - Visite o nó raiz.
    - Percorra em pré-ordem cada uma das subárvores da raiz na ordem definida.
  - Para uma árvore binária:
    - Visite o nó raiz.
    - Percorra em pré-ordem a subárvore esquerda.
    - Percorra em pré-ordem a subárvore direita.
  - Ex: A, B, C, D, E, F, G, H, I



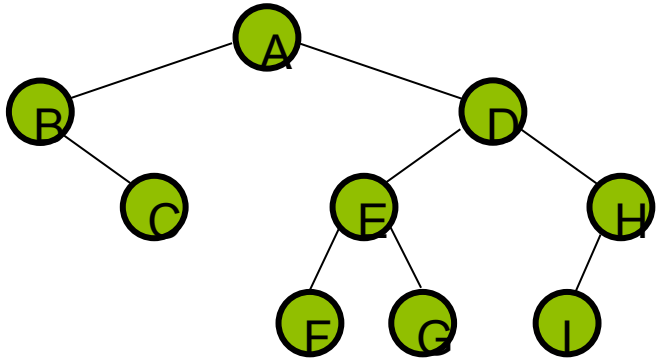
Pesquisa em profundidade

- Pesquisa em **pós-ordem**
  - A raiz é último nó a ser visitado.
  - A Pesquisa é definida recursivamente.
    - Percorra em pós-ordem cada uma das subárvores da raiz na ordem definida.
    - Visite o nó raiz.
  - Para uma árvore binária:
    - Percorra em pós-ordem a subárvore esquerda.
    - Percorra em pós-ordem a subárvore direita.
    - Visite o nó raiz.
  - Ex: C, B, F, G, E, I, H, D, A



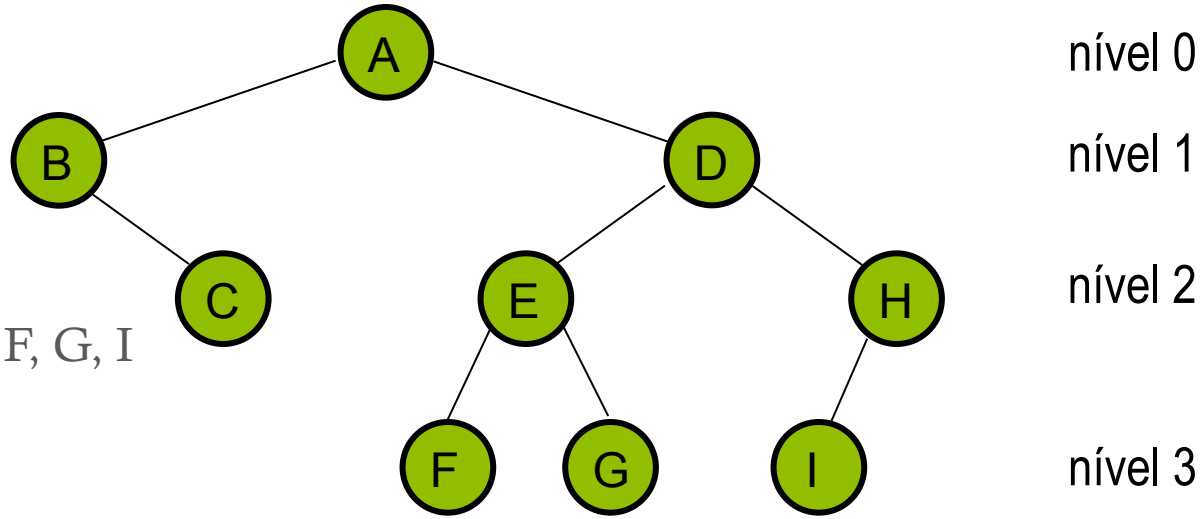
Pesquisa em profundidade

- Pesquisa em **Ordem Simétrica** (in-ordem)
  - O nó raiz é visitado entre as visitas das sub-árvores esquerda e direita.
  - A pesquisa **só faz sentido** em árvores binárias.
    - Percorra a subárvore esquerda.
    - Visite o nó raiz.
    - Percorra a subárvore direita.
  - Ex: B, C, A, F, E, G, D, I, H



Pesquisa em Largura ou Extensão

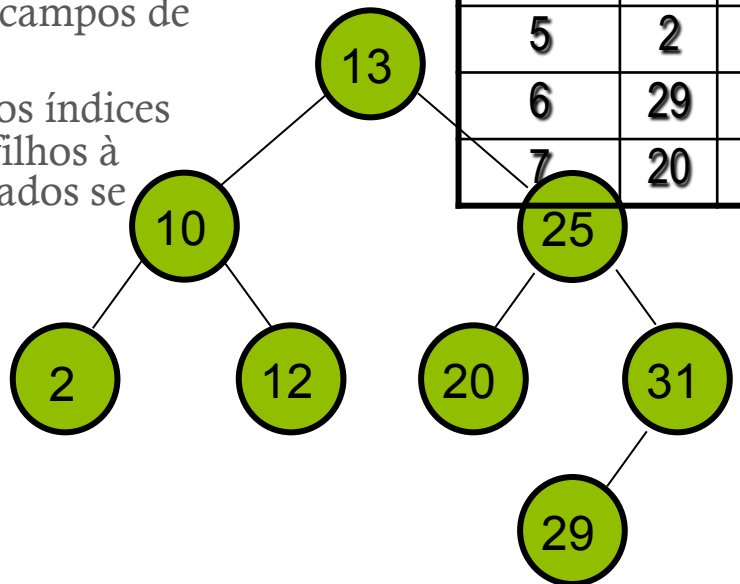
- A pesquisa em extensão visita os nós na ordem dos níveis da árvore de cima para baixo ou de baixo para cima.
- Pode-se visitar os nós em cada nível da esquerda para a direita ou da direita para esquerda.
- Há 4 possibilidades.
- Ex: A, B, D, C, E, H, F, G, I



Implementação de árvores binárias

- As árvores binárias podem ser implementadas:
  - como matrizes e
  - como estruturas encadeadas.
- Para implementar uma árvore binária como uma matriz:
  - um nó é declarado como uma estrutura com um campo de informação e dois campos de ponteiros.
  - Os campos de ponteiros contém os índices das células da matriz em que os filhos à esquerda e à direita são armazenados se houver algum.

Índice	Info	Esq.	Dir.
0	13	4	2
1	31	6	-1
2	25	7	1
3	12	-1	-1
4	10	5	3
5	2	-1	-1
6	29	-1	-1
7	20	-1	-1



Implementação de árvores binárias

- A implementação de árvores baseadas em matrizes pode ser incoveniente em função da alocação estática.
- Essa característica é importante pois pode ser difícil prever quantos nós devem ser criados.
- Normalmente uma estrutura de dados **dinâmica** é o modo mais **eficiente** de representar uma árvore.
- Em uma estrutura de dados dinâmica:
  - um nó é instância de uma classe composta por um membro de informação e dois ponteiros que apontem para as sub-árvores direita e esquerda.

Árvores de Pesquisa

- São árvores que suportam operações eficientes de:
  - busca,
  - inserção e
  - remoção.
- A árvore armazena um número finito de chaves a partir de um conjunto de chaves **K** totalmente ordenado.
- Cada nó na árvore contém pelo menos uma chave e todas as chaves são diferentes.
- Numa árvore de busca há um critério de ordenação de dados.

Pesquisa em Árvores de pesquisa

- A vantagem principal de uma árvore de busca:
  - o critério de **ordenação** dos dados **assegura** que não é necessário **fazer uma pesquisa completa** numa árvore para **encontrar um valor**.

Árvores M-Múltiplas de Pesquisa

- Uma árvore **M-múltipla** de pesquisa **T** é um conjunto finito de chaves onde ou **T** é vazio ou **T** consiste de **n** árvores **M-múltiplas** de busca T0, T1,... ,Tn-1, e n-1 chaves k1, k2,... ,kn-1,

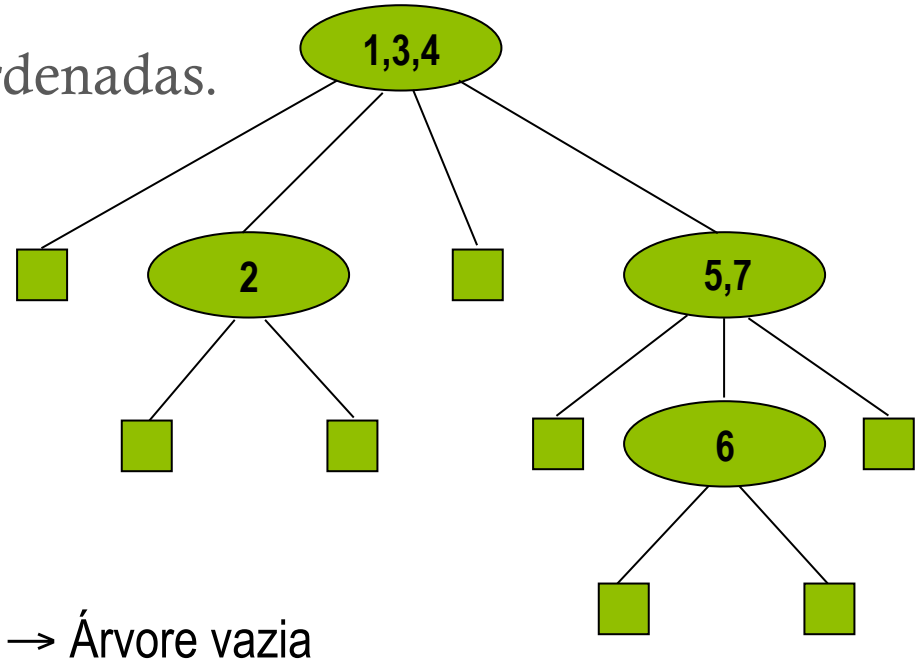
  T={T0, k1, T1, k2, T2, ... , kn-1, Tn-1}

- onde $2 \leq n \leq M$, tal que as chaves e os nós satisfazem as seguintes propriedades de ordenação:
  - As chaves em cada nó são distintas e ordenadas: $ki < ki+1$ para $1 \leq i \leq n-1$.
  - Todas as chaves das subárvores Ti-1 são menores do que ki . A árvore Ti-1 é dita a subárvore da esquerda em relação a ki .
  - Todas as chaves das subárvores Ti+1 são maiores do que ki . A árvore Ti +1 é dita a subárvore da direita em relação a ki .

Árvores M-Múltiplas de Pesquisa

- Cada nó não vazio tem entre 1 e 3 chaves e no máximo 4 subárvores.
- As chaves de cada nó estão ordenadas.

M=4



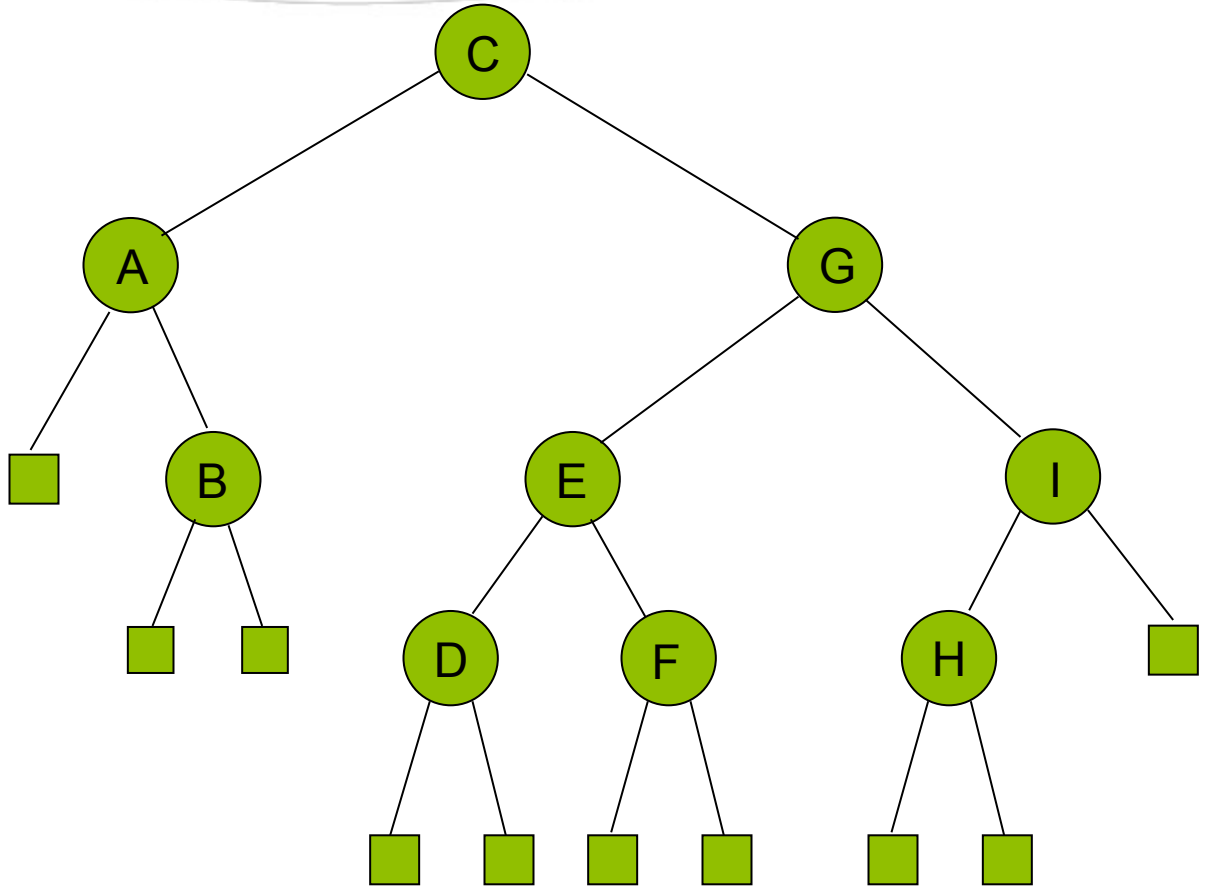
Árvores Binárias de Pesquisa

- Uma árvore binária de pesquisa **T** é um conjunto finito de chaves onde ou **T** é vazio ou **T** consiste em uma raiz e em exatamente 2 árvores binárias de busca **TL** e **TR** ,

  T={r, TL, TR}

- tal que as seguintes propriedades são satisfeitas:
  - **Todas as chaves** da subárvore TL são menores do que **r** .
    - A árvore TL é dita a subárvore da esquerda em relação a **r** .
  - **Todas as chaves** da subárvore TR são maiores do que **r** .
    - A árvore TR é dita a subárvore da direita em relação a **r** .

Árvores Binárias de Pesquisa



Pesquisa em Árvore de pesquisa

- A vantagem principal de uma árvore de pesquisa é:
  - O critério de ordenação dos dados.
- **Não** é necessário fazer um percurso completo numa árvore para encontrar um valor.
- Busca em uma árvore M-múltipla de pesquisa
  - Na busca de um valor **x** primeiramente examina-se as chaves do nó raiz.
  - Se **x** não estiver no nó raiz há 3 possibilidades:
    - **x** é menor que **k1** , neste caso a busca deve prosseguir na subárvore **T0** ;
    - **x** é maior que **kn-1** , neste caso a busca deve prosseguir na subárvore Tn-1 ;
    - Ou existe um **i** tal que $1 \leq i < n-1$ para o qual **ki** < **x** < **ki+1**, caso em que a busca deve prosseguir na árvore **Ti** .

Pesquisa em Árvore de pesquisa

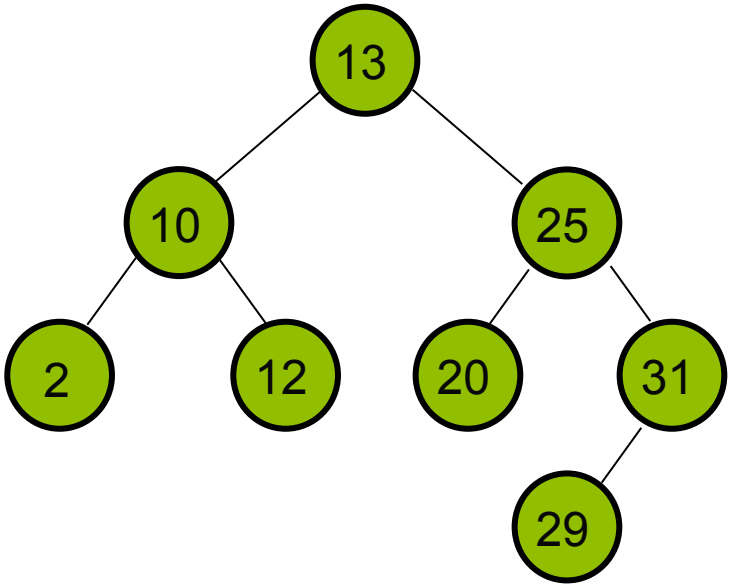
- Para uma árvore M-múltipla de pesquisa quando **x** não é encontrado em um certo nó, a busca **continua apenas em uma das subárvores daquele nó**.
- Uma busca com sucesso começa na raiz e percorre o caminho até o nó no qual a chave se encontra.
- Quando o objeto de busca não está na árvore, o método de busca descrito traça um caminho da raiz **até uma subárvore vazia**.

Pesquisa em Árvore binária de pesquisa

- Algoritmo para localizar um elemento em uma árvore binária.
  1. Para cada nó compare a chave a ser localizada com o valor armazenado no nó correspondente.
  2. Se a chave for menor vá para a sub árvore esquerda.
  3. Se a chave for maior vá para a sub árvore direita.
  4. A busca para quando a chave é igual ao nó.
  5. O algoritmo também deverá parar se o valor não for encontrado.

Pesquisa em Árvore binária de pesquisa

- Para localizar o valor 31 apenas 3 testes são realizados.
- O pior caso é quando se busca o valor 29.

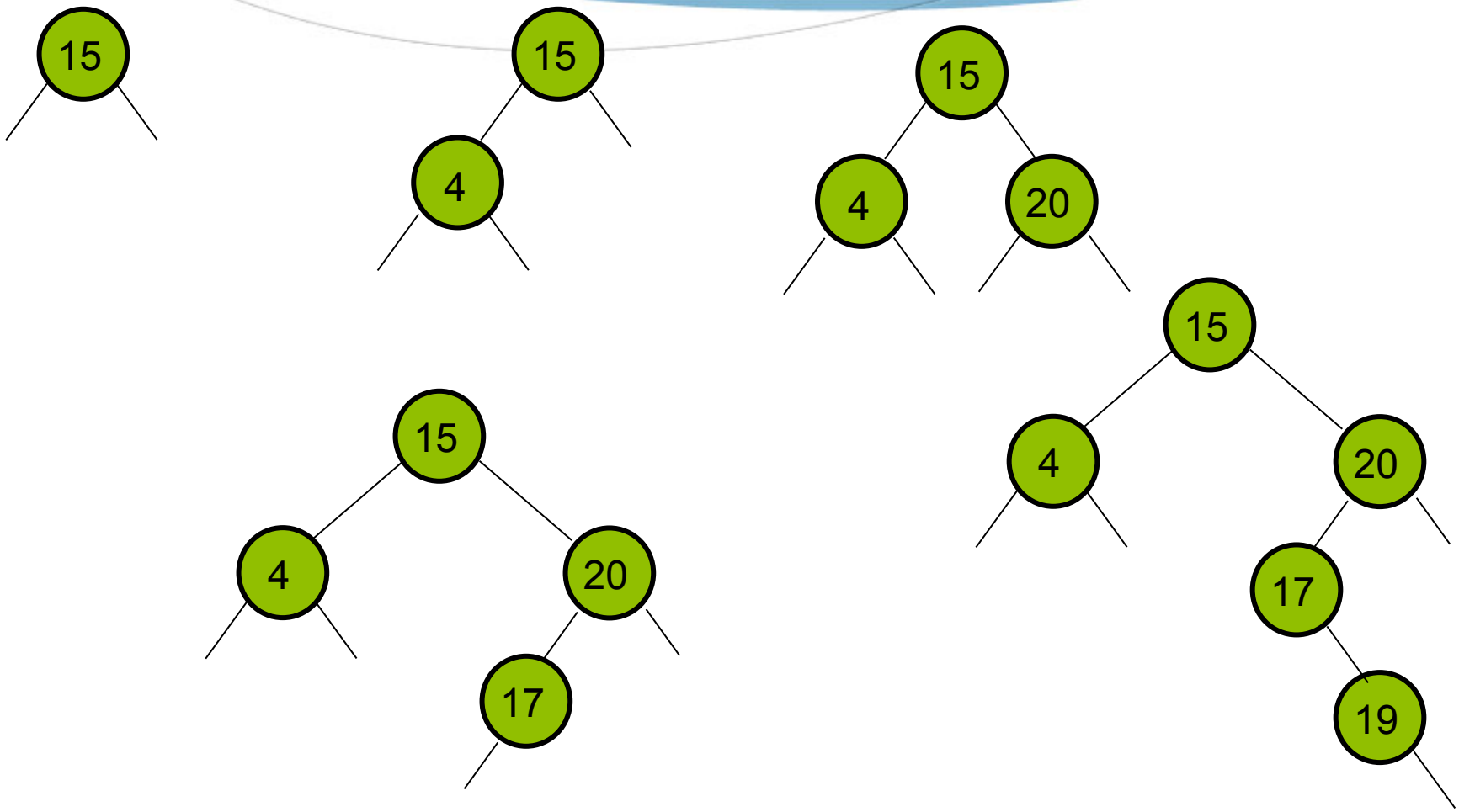


Inserção de um nó em uma árvore binária

- **Inserção de um novo nó** (*n_node*)

  Faz-se *c_node* = raiz
  Enquanto *c_node* <> nulo
  {
  Se *n_node* < *c_node*
  n_node = filho esquerdo de *c_node*
  Se *n_node* > *c_node*
  n_node = filho direito de *c_node*
  }
  Anexar *n_node*
  filho esquerdo de *n_node* = nulo
  filho direito de *n_node*= nulo

Inserção de um nó em uma árvore binária



Inserção de um nó em uma árvore binária

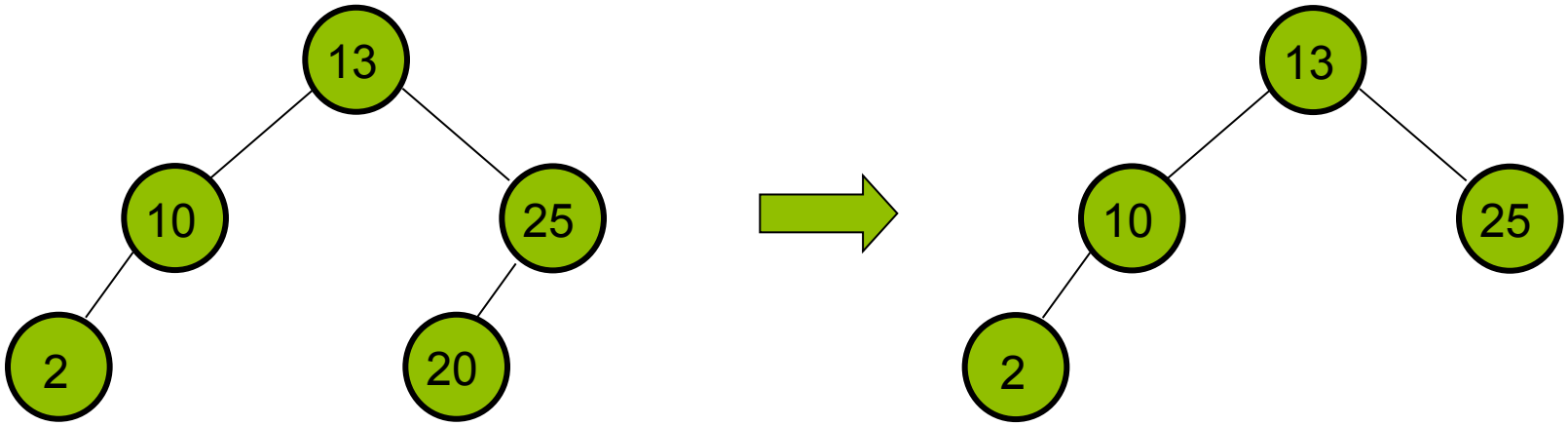
- A inserção **deve evitar** a ocorrência do mesmo elemento mais de uma vez na árvore.
  - Se a inserção de um mesmo elemento for necessária deve-se indicar a decisão a ser tomada em função da aplicação.

Remoção de um nó em uma árvore binária

- A remoção de um nó é **outra operação necessária** para se manter uma árvore binária de busca.
  - O nível de complexidade depende da posição do nó a ser removido da árvore.
- **Existem 3 casos** de remoção de um nó da árvore binária de busca:
  - O nó é uma folha e não tem filhos.
    - O ponteiro de seu ascendente é ajustado para nulo e o nó é removido.
  - O nó tem um filho.
    - O ponteiro de seu ascendente é ajustado para apontar para o filho do nó e o nó é removido.
  - O nó tem dois filhos.
    - Nenhuma operação de uma etapa pode ser realizada pois os ponteiros esquerdo e direito do ascendente não podem apontar para ambos os filhos do nó a ser removido.

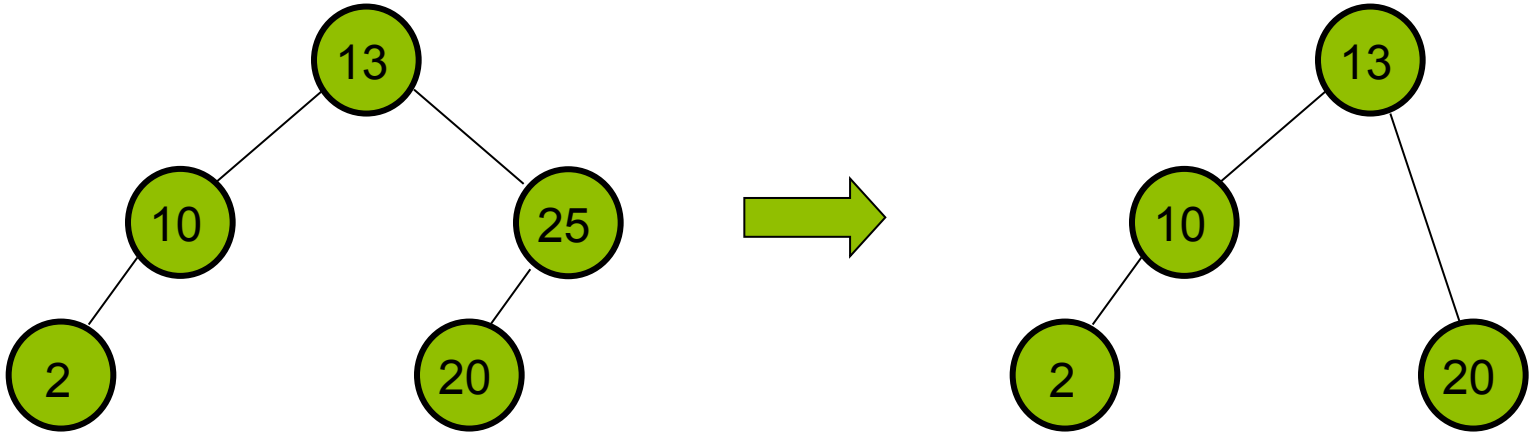
Remoção de um nó em uma árvore binária

- O nó é uma folha e não tem filhos.



Remoção de um nó em uma árvore binária

- O nó tem um filho.



Remoção de um nó em uma árvore binária

- Se o nó tem 2 filhos a remoção pode ser feita de duas formas:
  - por fusão ou
  - por cópia.

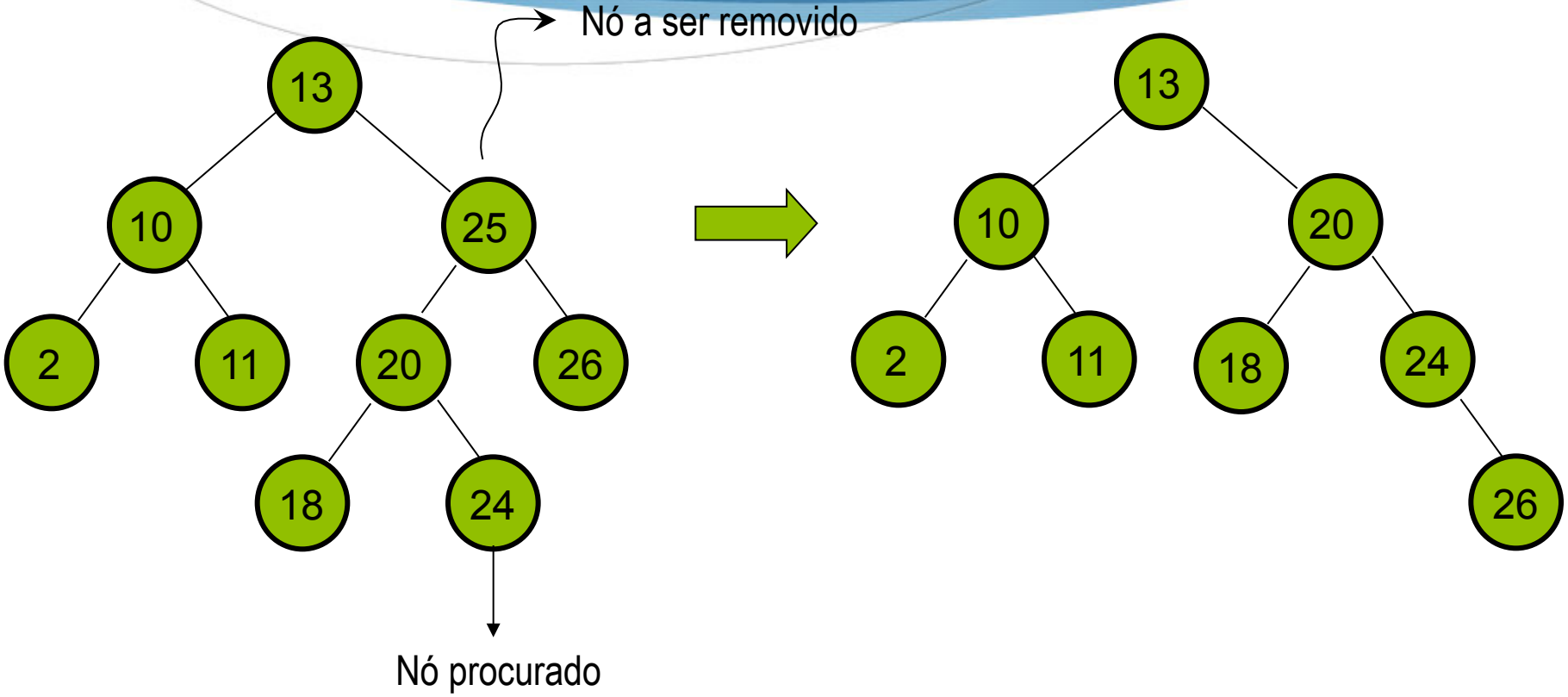
- **Remoção por fusão**
  - Uma árvore é extraída de duas subárvores do nó e esta árvore é anexada ao ascendente do nó.
    - Os valores da subárvore direita são maiores do que os valores da subárvore esquerda.
  - Na subárvore da esquerda o nó com o maior valor deve tornar-se ascendente da subárvore direita.
  - O nó com o maior valor na subárvore da esquerda é o nó mais à direita desta subárvore.
  - Este nó pode ser encontrado movendo-se ao longo da subárvore e tomando-se sempre os ponteiros direitos até encontrar-se o valor nulo.
  - Simetricamente na subárvore da direita o nó com o menor valor deve tornar-se um ascendente da subárvore esquerda.

Remoção de um nó em uma árvore binária por fusão

- Algoritmo

```
Se o nó não tem filhos à direita
    Anexe seu filho da esquerda à seu ascendente
Senão
      Se o nó não tem filhos à esquerda
        Anexe seu filho da direita à seu ascendente
    Senão {
          Mova-se para a esquerda
        Mova-se para a direita até encontrar o valor nulo
        O nó encontrado será ascendente da subárvore direita
     }
    Remova o nó
```

Remoção de um nó em uma árvore binária por fusão



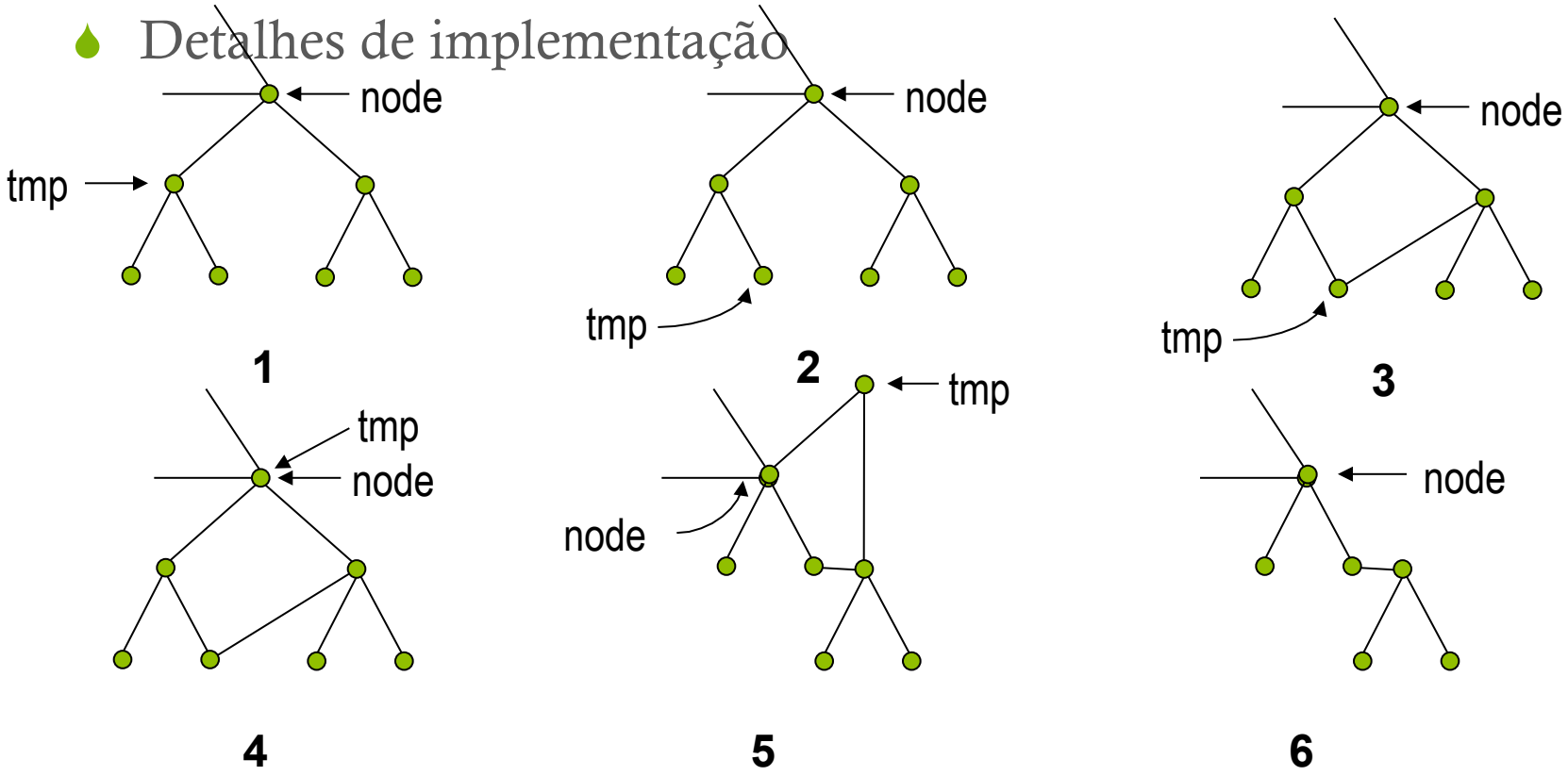
Remoção de um nó em uma árvore binária por fusão

Implementação

```
if (node !=0) {
    if (node->right==0) node=node->left;
    else
       if (node->left==0) node=node->right;
       else {
          tmp=node->left;                                  1
          while (tmp->right != 0) tmp=tmp->right;          2
          tmp->right = node->right;                        3
          tmp=node;                                        4
          node = node->left;                               5
       }
     delete tmp;                                           6
 }
```

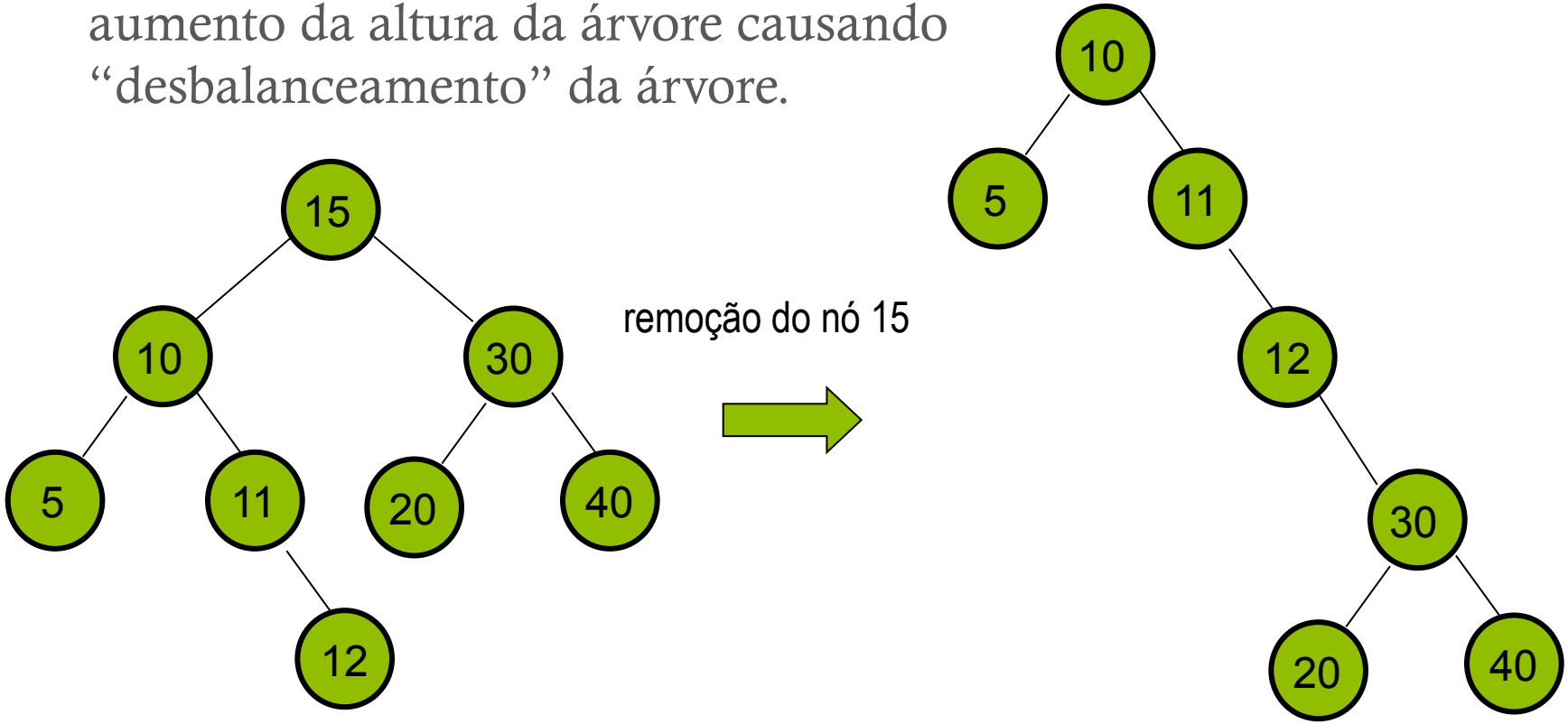
Remoção de um nó em uma árvore binária por fusão

- Detalhes de implementação



Remoção de um nó em uma árvore binária por fusão

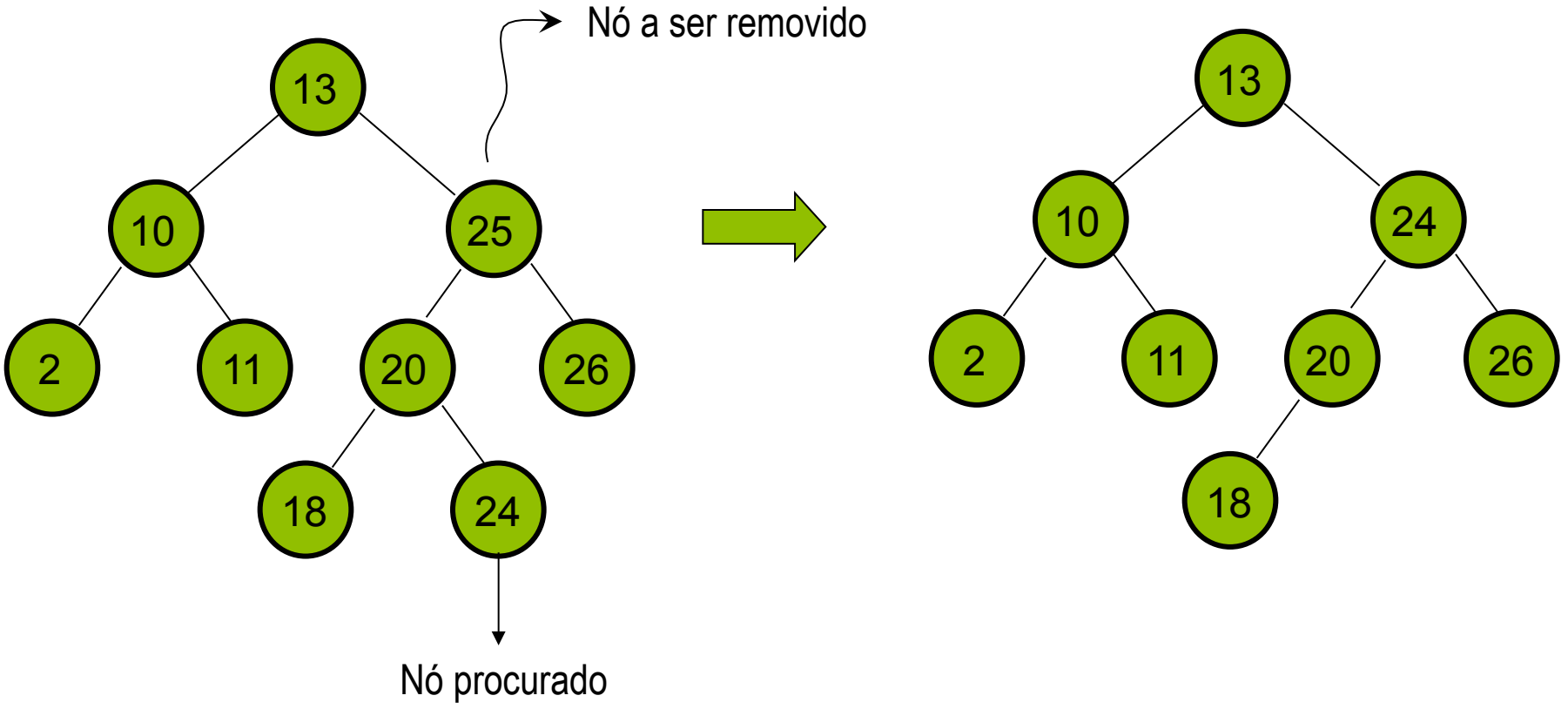
- O algoritmo de remoção por fusão pode resultar no aumento da altura da árvore causando "desbalanceamento" da árvore.



Remoção de um nó em uma árvore binária por cópia

- A remoção por cópia remove uma chave k_1 sobreescrevendo-a por uma outra chave k_2 e então removendo o nó que contém k_2. e então removendo o nó que contém
- Se o nó tem 2 filhos ele pode ser reduzido a um dos dois casos simples:
  - o nó é uma folha ou
  - o nó tem somente um filho não-vazio.
  - Pode-se substituir a chave que está sendo removida pelo seu predecessor (ou sucessor) imediato.
  - O predecessor de chave é a chave do nó mais à direita na subárvore da esquerda.
  - Analogamente, seu sucessor imediato é a chave do nó mais à esquerda na subárvore da direita.
  - Localiza-se o predecessor a partir da raiz da subárvore esquerda e movendo-se para direita o tanto quanto possível até encontrar o valor nulo.
  - Substitui-se a chave do predecessor pelo nó a ser removido.
  - Dessa forma o nó se transforma em uma folha ou antecessor de apenas um filho não vazio.

Remoção de um nó em uma árvore binária por cópia



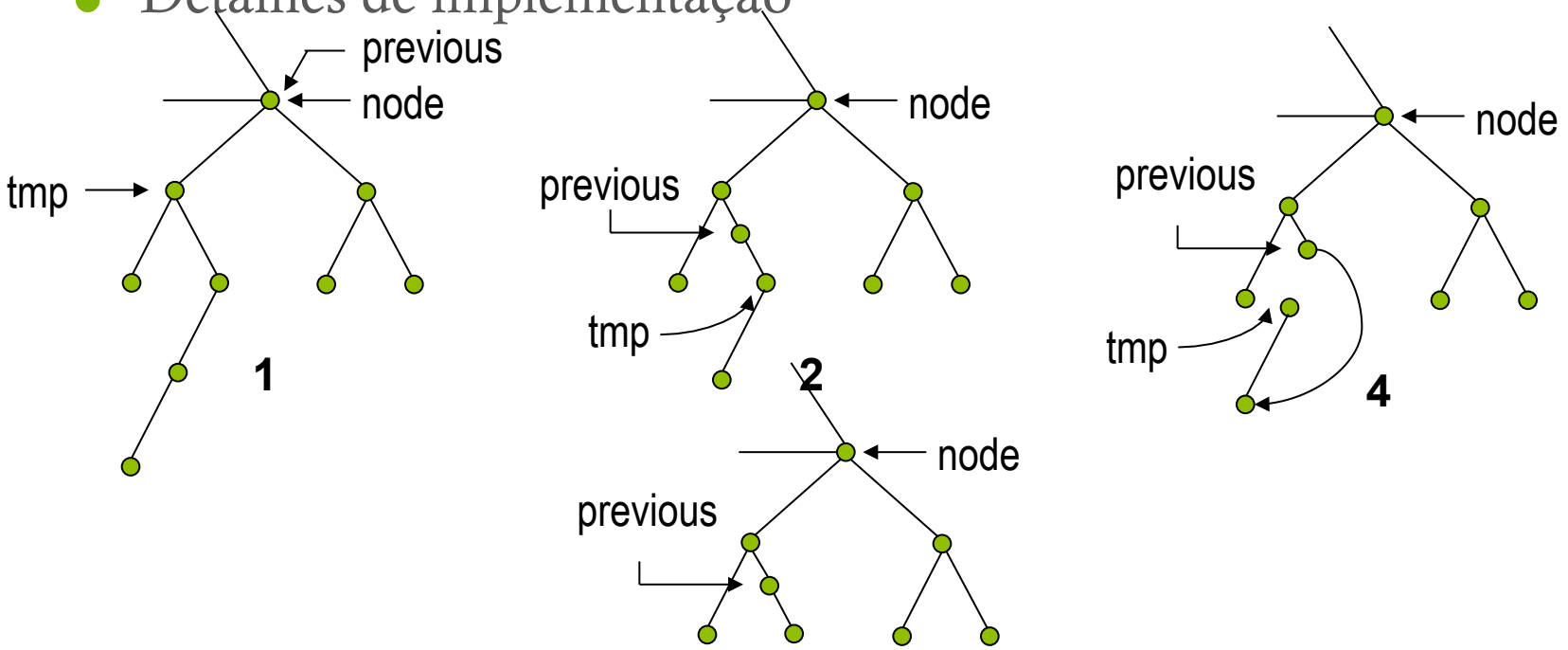
Remoção de um nó em uma árvore binária por cópia

- Implementação

```
if (node->right==0) node=node->left;
else
  if (node->left==0) node=node->right;
  else {
     tmp=node->left;
      prev = node;
       while (tmp->right != 0){                    1
         prev=tmp;                                 2
         tmp=tmp->right;
     }
     node->key = tmp->key;                         3
     if (prev->node==0) prev->left = tmp->left;    4
     else prev->right = tmp->left;                 5
   }
  delete tmp;
```

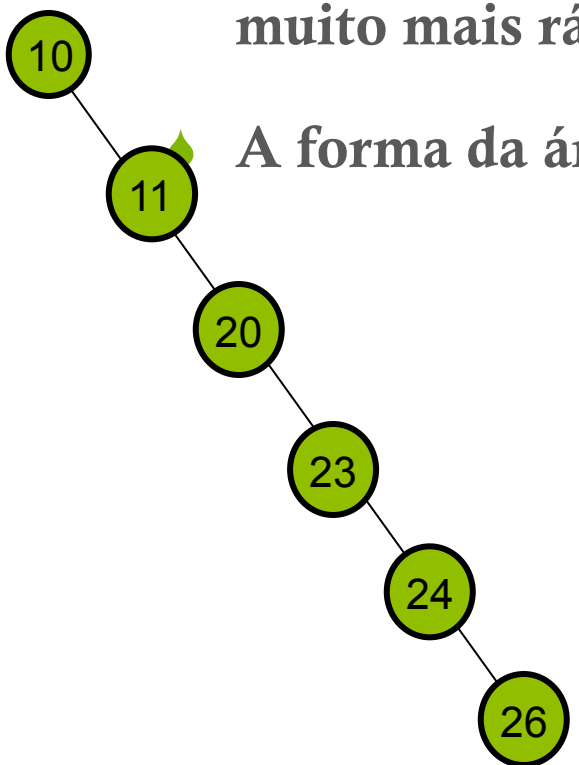
Remoção de um nó em uma árvore binária por cópia

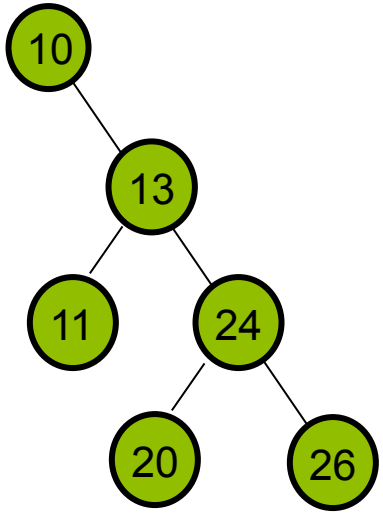
- Detalhes de implementação

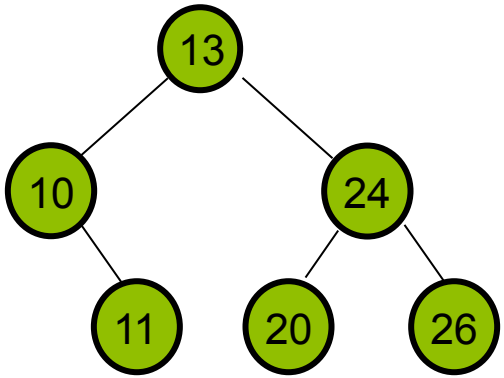


Balanceamento de árvores

- **As árvores podem permitir** que o processo de pesquisa seja **muito mais rápido** do que pesquisa em listas encadeadas.
- **A forma da árvore** influi na eficiência do processo de pesquisa.







Balanceamento de árvores

- Uma árvore binária é balanceada se a diferença na altura de ambas as sub-árvores de qualquer nó na árvore é zero ou um.
- Uma árvore é **perfeitamente** balanceada se a árvore é balanceada **e todas as folhas** se encontram em um nível ou em dois níveis.
- Em uma árvore balanceada o número de nós **n** que uma árvore balanceada de altura **h** pode armazenar é dado por:

  n= 2h – 1

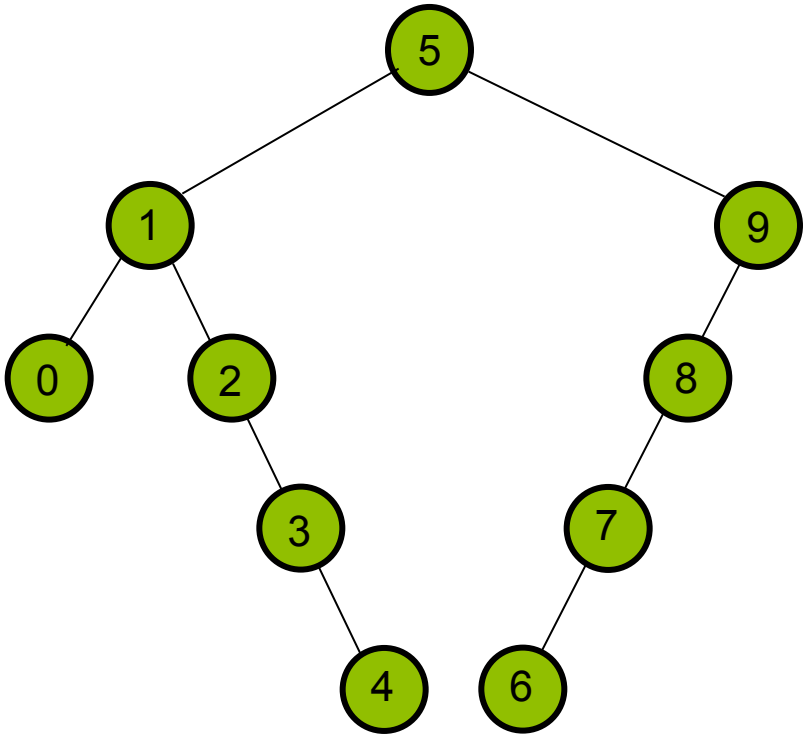
  - Logo, a altura **h** necessária para armazenar **n** nós em uma árvore balanceada é dada por:

$$h = \log_2 (n+1)$$

Balanceamento de árvores

- Procedimento para balancear uma árvore
  - Armazenar os dados ou chaves em uma matriz.
  - Ordenar a matriz.
  - Colocar na raiz o elemento central da matriz ordenada.
  - O filho da esquerda da raiz será o elemento central entre os elementos da matriz ordenada que são menores que a raiz.
  - O filho da direita da raiz será o elemento central entre os elementos da matriz ordenada que são maiores que a raiz.
  - O processo se repete até que todos os elementos da matriz sejam colocados na árvore.

Balanceamento de árvores



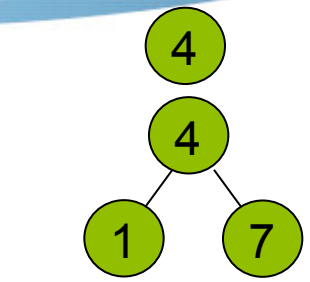
Balanceamento de árvores

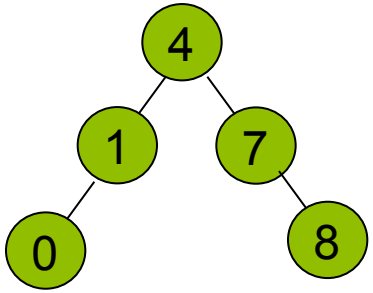
(a) 0 1 2 3 (4) 5 6 7 8 9

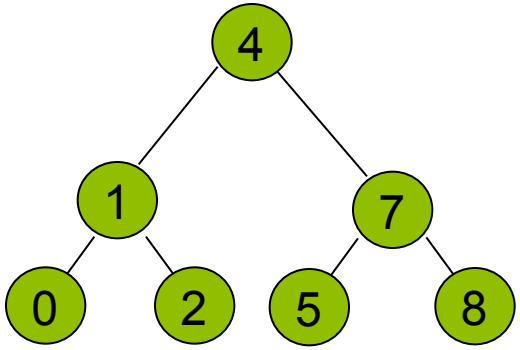
(b) 0 (1) 2 3 [4] 5 6 (7) 8 9

(c) (0) [1] 2 3 [4] 5 6 [7] (8) 9

(d) [0] [1] (2) 3 [4] (5) 6 [7] [8] 9

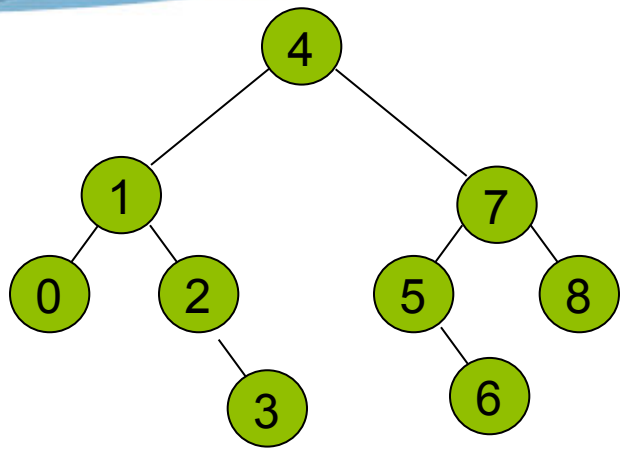




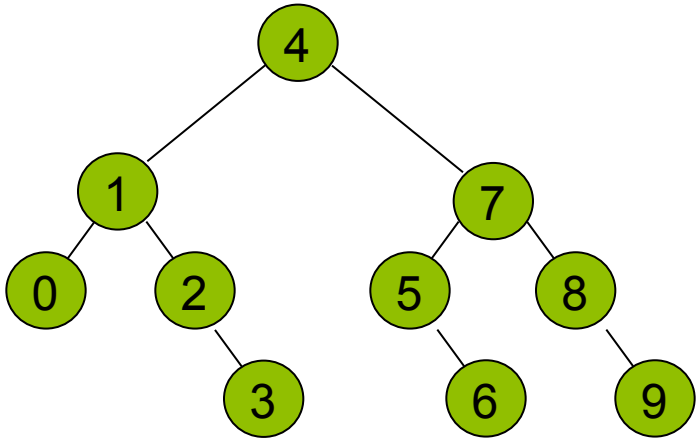


Balanceamento de árvores

(e) 0 1 2 3 4 5 6 7 8 9



(f) 0 1 2 3 4 5 6 7 8 9



Balanceamento de árvores

- O algoritmo **DSW** (Day, Stout, Warren) utiliza procedimentos sem ordenação.
  - A vantagem do algoritmo **DSW** é não precisar de uma matriz de dados adicional.
  - O essencial para as transformações de árvore nesse algoritmo é a rotação.
    - Há dois tipos de rotação: à esquerda e à direita.

1. O algoritmo DSW transforma uma árvore binária desbalanceada em uma árvore com estrutura semelhante à uma lista encadeada chamada "**espinha dorsal**".
2. A árvore do tipo espinha dorsal é então transformada em uma árvore balanceada girando repetidamente cada segundo nó ao redor do seu ascendente.

Balanceamento de árvores

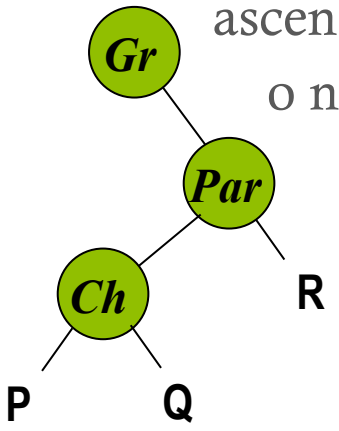
- A rotação à direita do nó **Ch** com relação ao seu ascendente Par é realizada de acordo com o seguinte algoritmo:
  - RotateRight (Gr, Par, Ch)

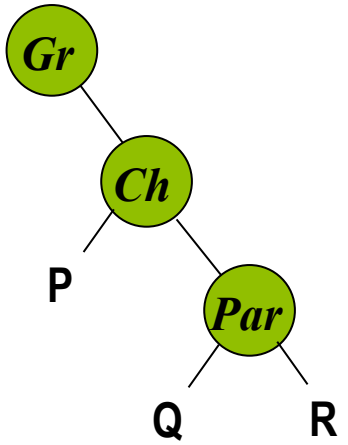
    SE **Par** não é a raiz da árvore (se Gr não é nulo)

    o avô **Gr** do filho **Ch** se torna o ascendente de **Ch** substituindo-se **Par**;

    a subárvore direita de **Ch** se torna a árvore esquerda do ascendente **Par** de Ch;

    o nó **Ch** recebe **Par** como seu filho direito;





Balanceamento de árvores

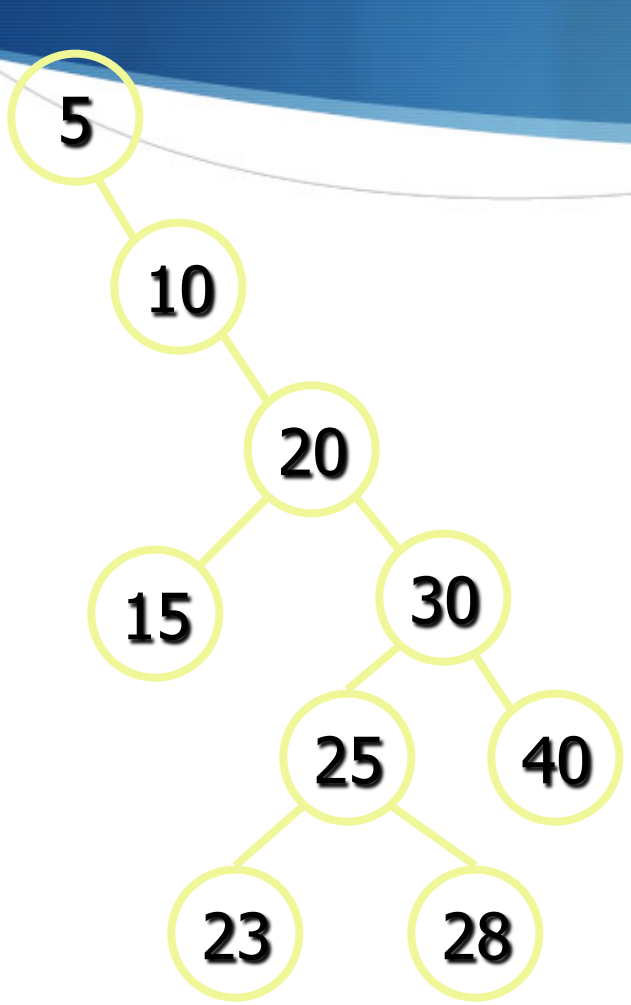
- Operações básicas do algoritmo **DSW**:
  - **Criação** de uma espinha dorsal.
  - **Rotações** à direita e à esquerda.
  - **Transformação** da espinha dorsal em uma árvore balanceada.

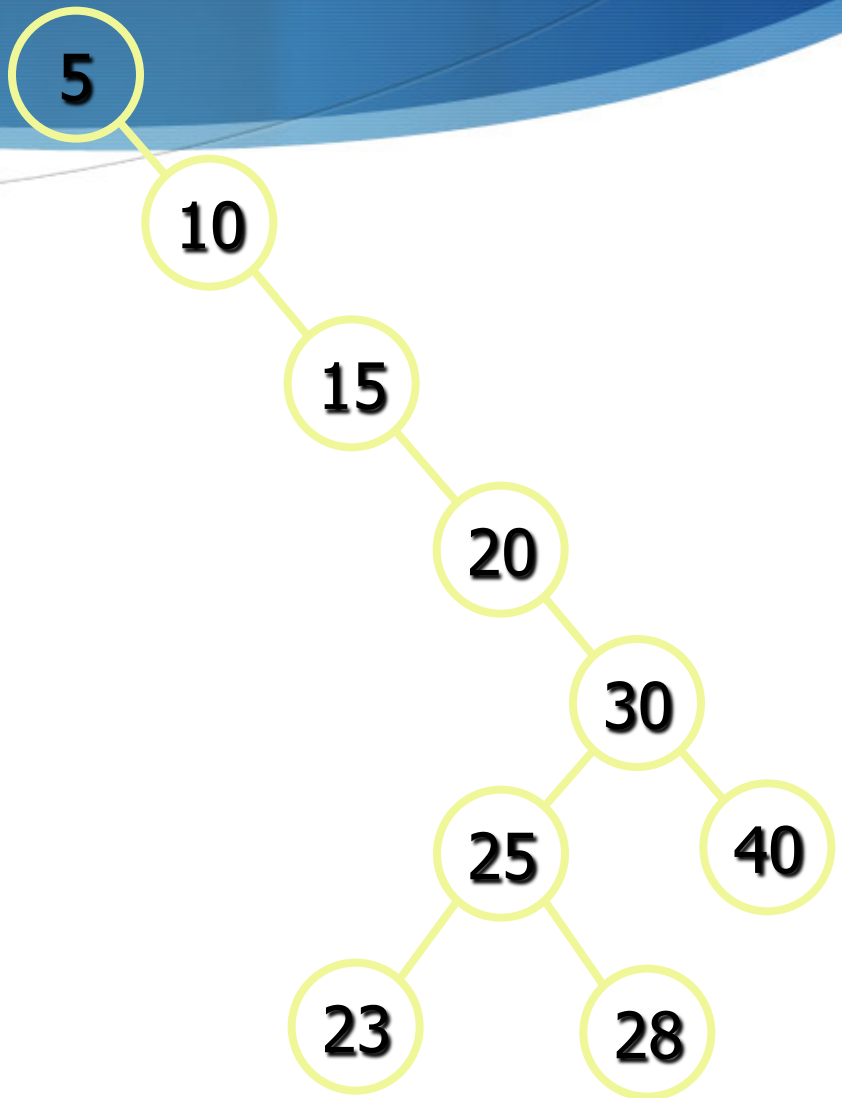
Balanceamento de árvores

- Na 1ªfase cria-se uma espinha dorsal:

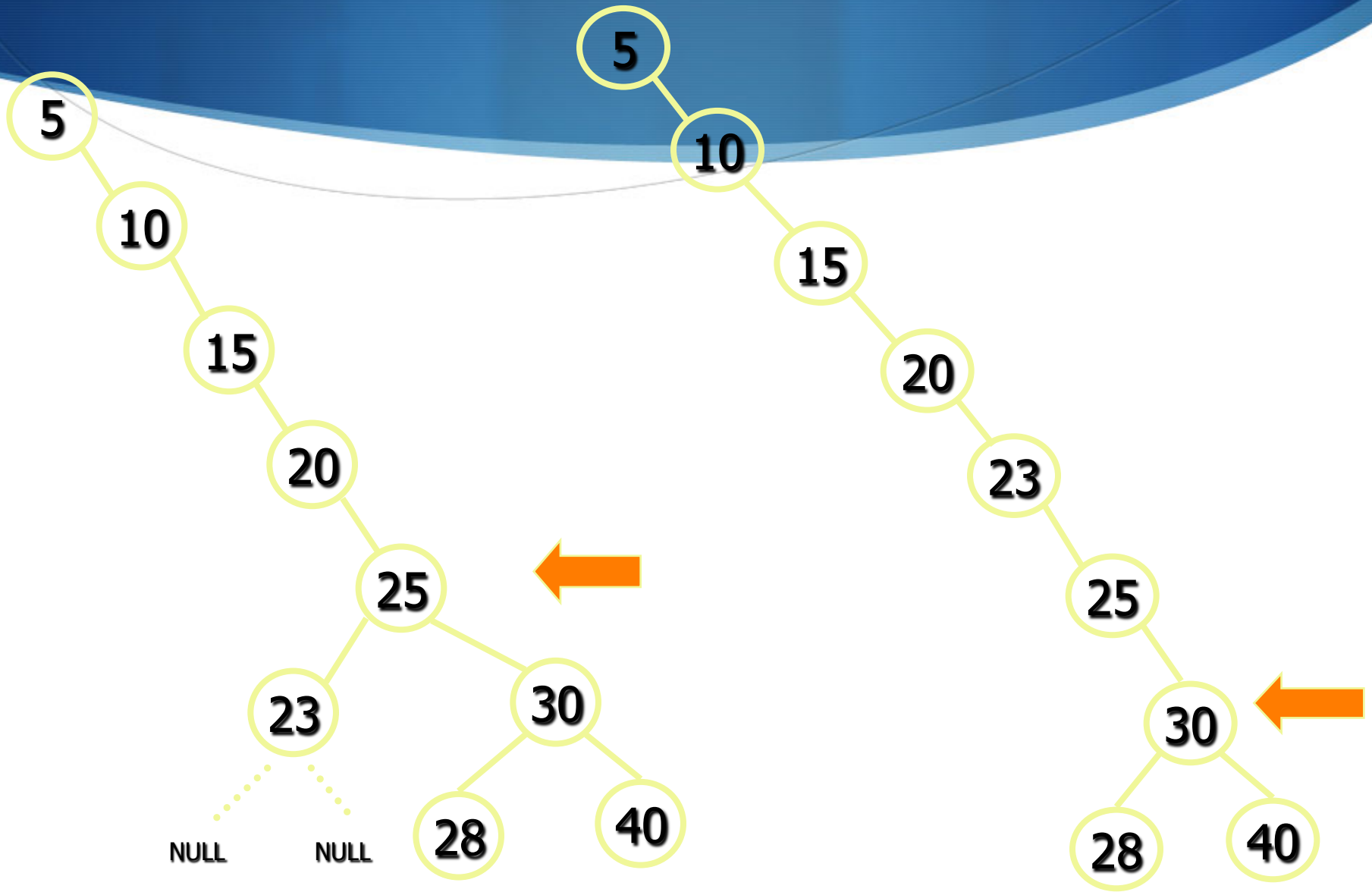
```
CreateBackbone(root,n)
tmp = root;
while ( tmp != 0 )
 if tmp tem um filho à esquerda
      Rotacione este filho em torno de tmp
      Ajuste tmp ao filho que acabou de se tornar pai
 else ajuste tmp ao seu filho direito
```

Balanceamento de árvores





Balanceamento de árvores



Balanceamento de árvores

Balanceamento de árvores

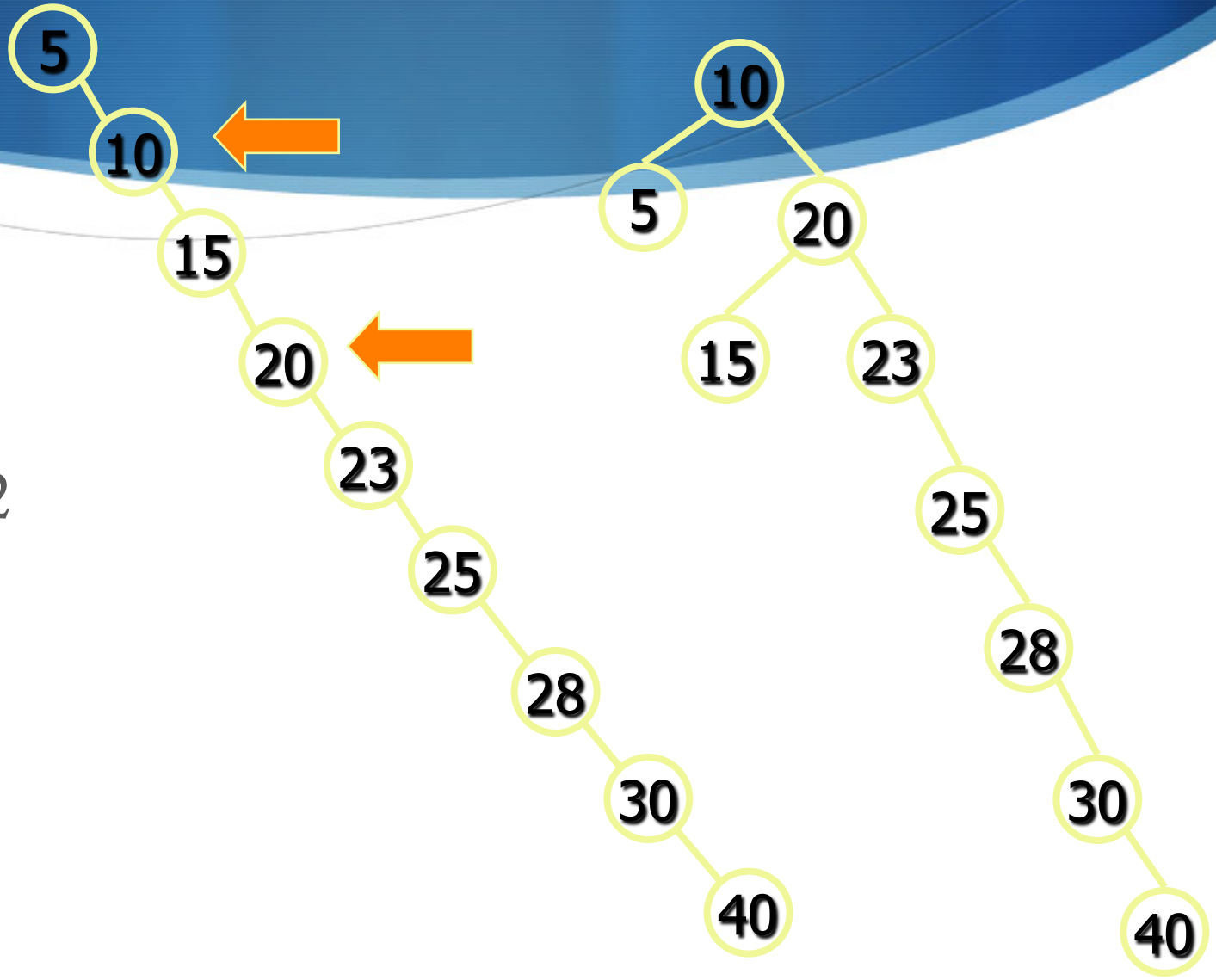
- Uma rotação exige o conhecimento sobre o ascendente de tmp de forma que outro ponteiro tem que ser mantido quando é implementado o algoritmo.
- Na **2ª fase** a espinha dorsal **é transformada** em uma árvore balanceada.
  - Em cada passe para baixo cada segundo nó até um ponto determinado é rotacionado ao redor de seu ascendente.

Balanceamento de árvores

- Na 2ª fase a espinha dorsal é transformada em uma árvore balanceada.

```
CreatePerfectTree(n)
   M = 2^floor[lg(n+1)]-1;   (lg=>base 2)
   Faça n-M rotações iniciando a partir do topo
      da espinha dorsal;
   While ( M > 1 ){
      M = M/2;
       Faça M rotações iniciando a partir do topo
      da espinha dorsal;}
```

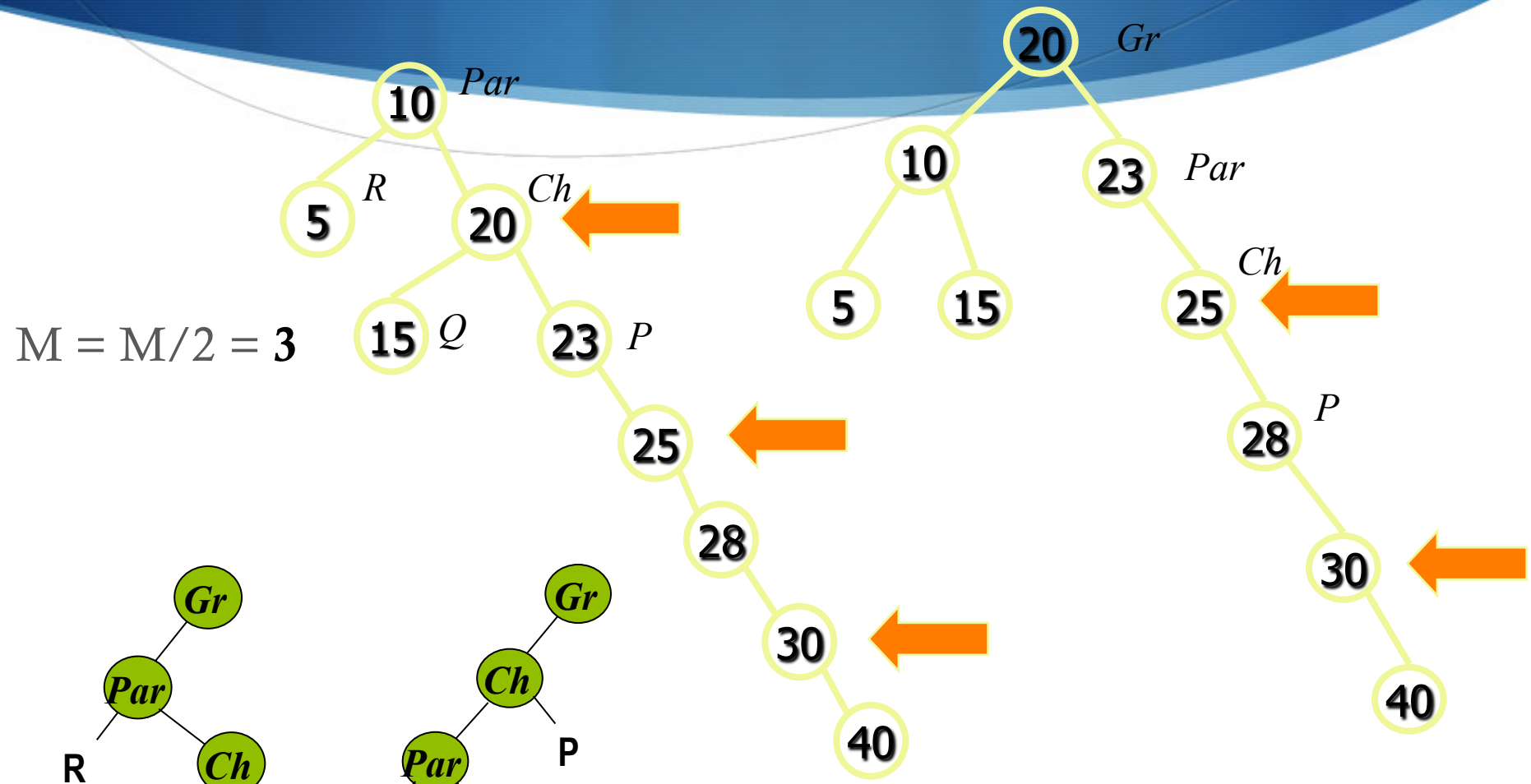
Balanceamento de árvores



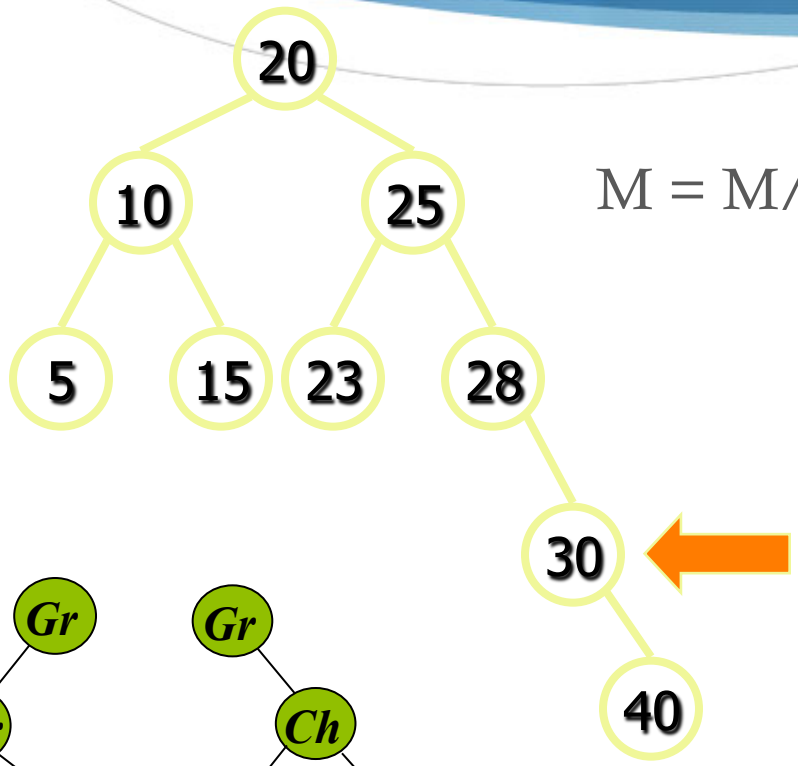
M = 7

n – M = 2

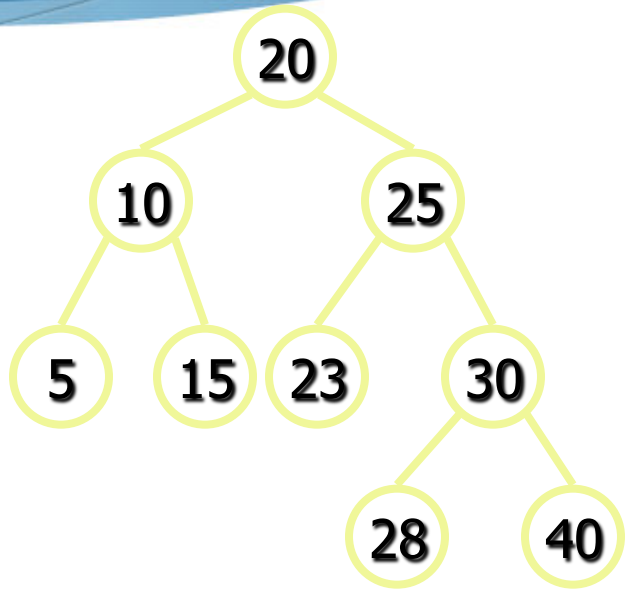
Balanceamento de árvores

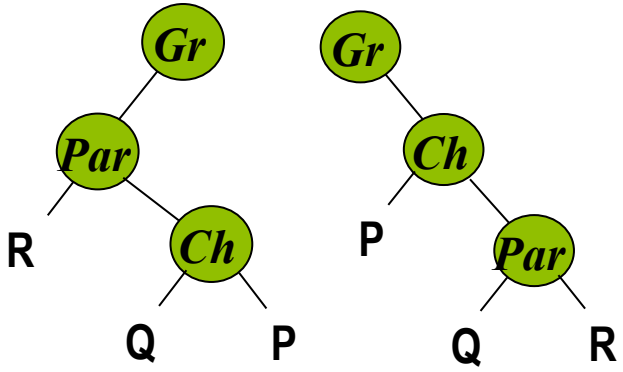


Balanceamento de árvores

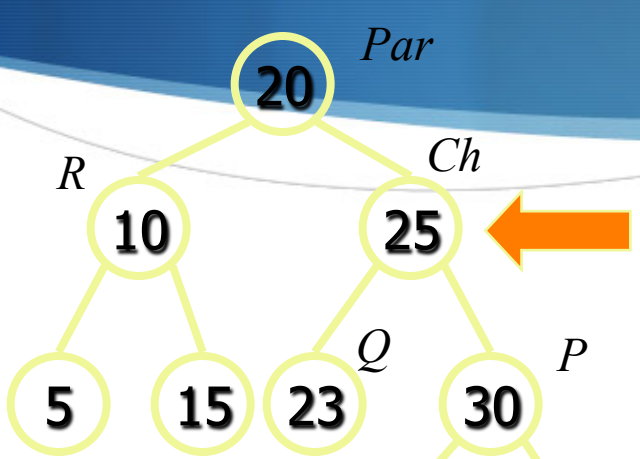


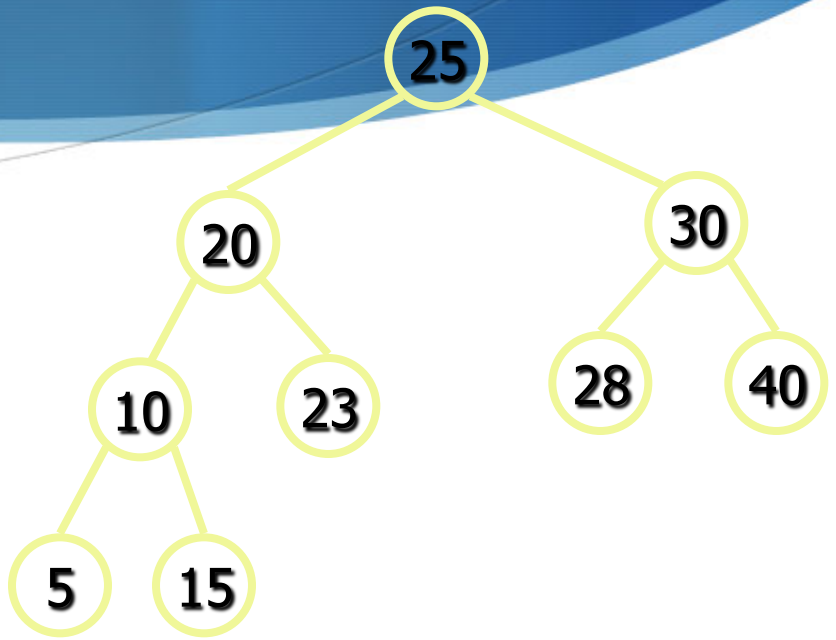
M = M/2 = 3



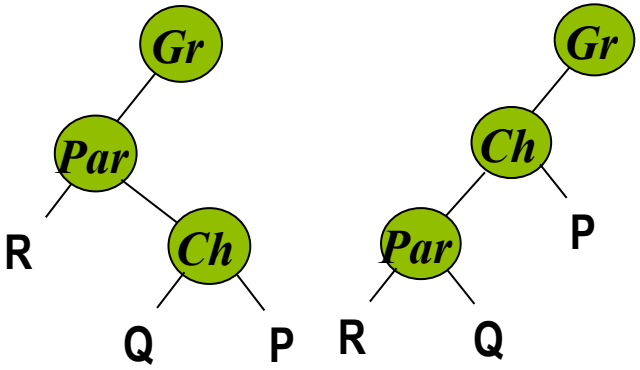


Balanceamento de árvores





M = M/2 = 1



Árvore balanceada

Exercício

- Faça um programa que crie uma arvore binária com as seguintes funções:
  - Inserção de um nó na arvore.
  - Remoção por fusão.
  - Remoção por copia.
- Implemente o Algoritmo DSW de balanceamento de árvores.
  - Fase 1
  - Fase 2