Estrutura Abstrata de Dados
Árvore Binária (AB)

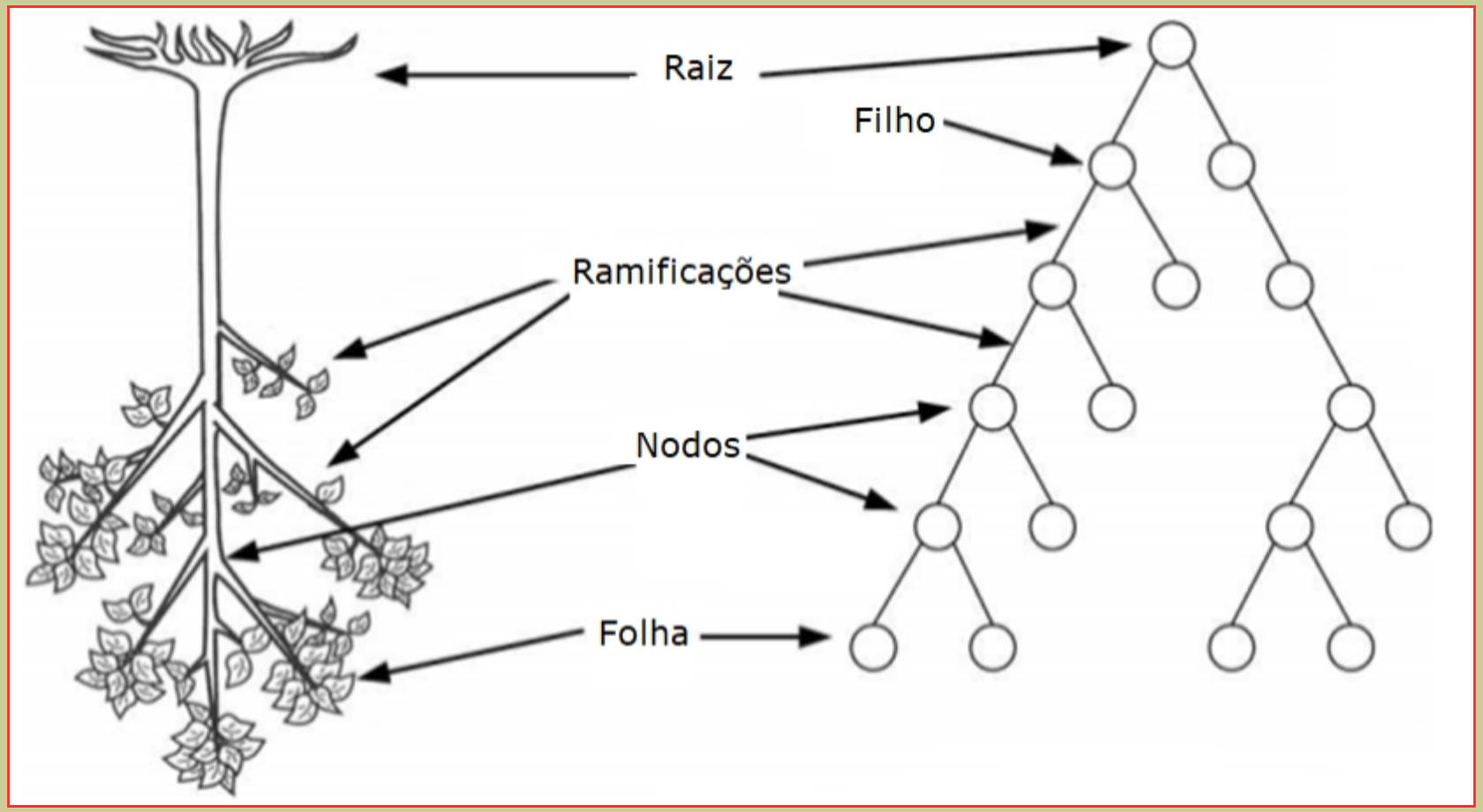
Implementação com ligações simples
(usando ponteiros e memória dinâmica)

Árvore

Definição

- No contexto das ciências da computação, uma **árvore** é uma estrutura de dados em que os dados
  - são todos do mesmo tipo
  - estão dispostos de forma hierárquica (um conjunto de dados está hierarquicamente *subordinado* a outro conjunto de dados)
- Uma **árvore** pode definir-se como um conjunto finito de **nodos** (nós), tal que
  - existe um nodo particular chamado **raiz** (nodo principal)
  - os restantes nodos (denominados de **filhos**) estão divididos em **m** conjuntos disjuntos T_1, T_2, …, T_m ($m \geq 0$), ligados à raiz através de **ramificações**
  - cada conjunto de nodos T_k ($k = 1, \ldots, m$) é uma árvore (**subárvore da raiz**)
  - cada nodo **filho** é também **raiz** de subárvore
  - um nodo sem filhos é denominado de **folha** (ou nodo terminal)

Analogia com árvore biológica

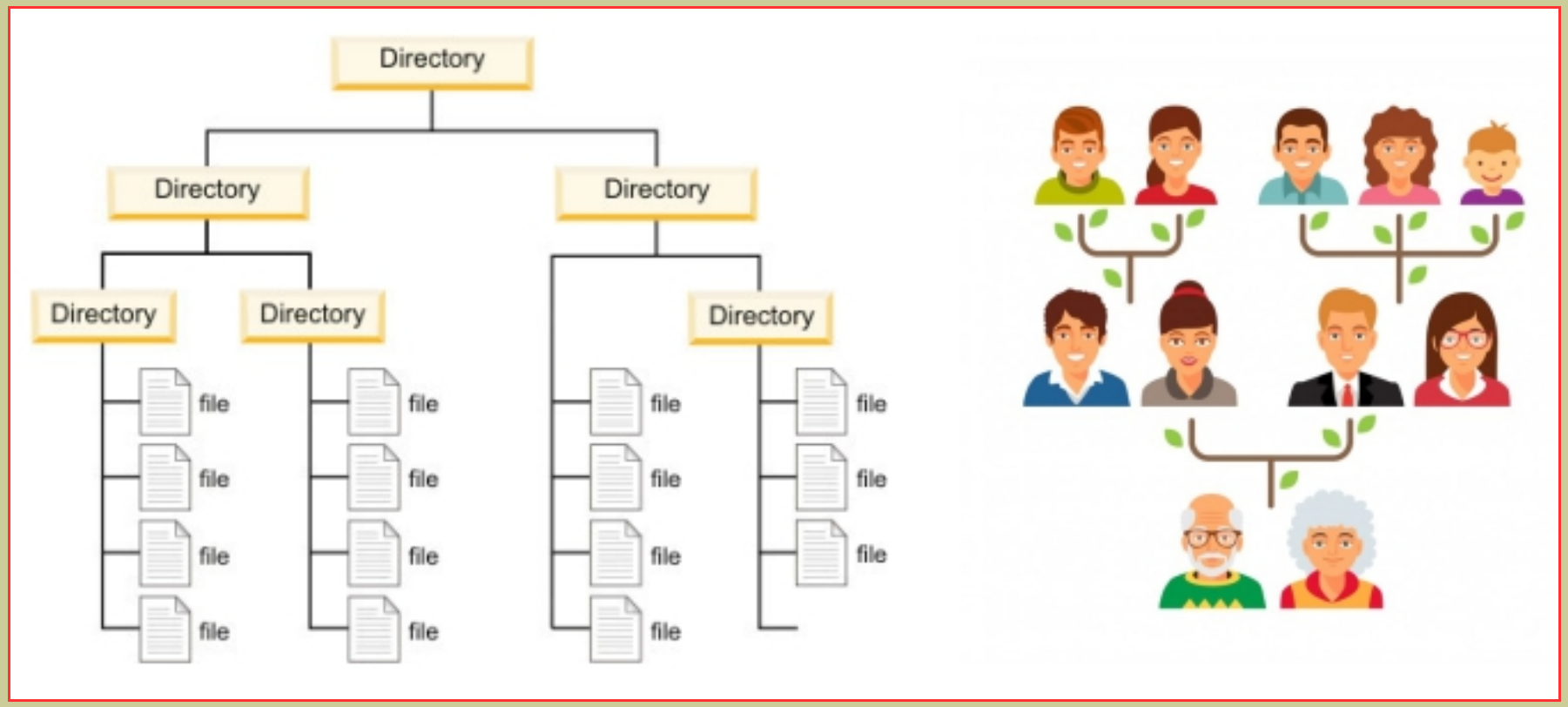


Conceitos gerais

- Existe alguns termos que se podem usar, tais como as que se seguem
  - pai
  - filho
  - irmão
  - antepassado
  - descendente
  - folha
- O **grau** de um **nodo** define-se como o número de subárvores desse nodo
- O **grau** de uma **árvore** é o grau máximo dos seus nodos
- O **nível** de um nodo pode ser definido da seguinte forma
  - o nodo raiz tem nível 0
  - os níveis dos restantes nodos são mais uma unidade do que o nível dos seus pais
- A **altura** (profundidade) de uma **árvore** é o nível máximo dos seus nodos

Exemplos

- A estrutura de pastas (ou diretorias) de um sistema de ficheiros
- Uma árvore genealógica de uma família



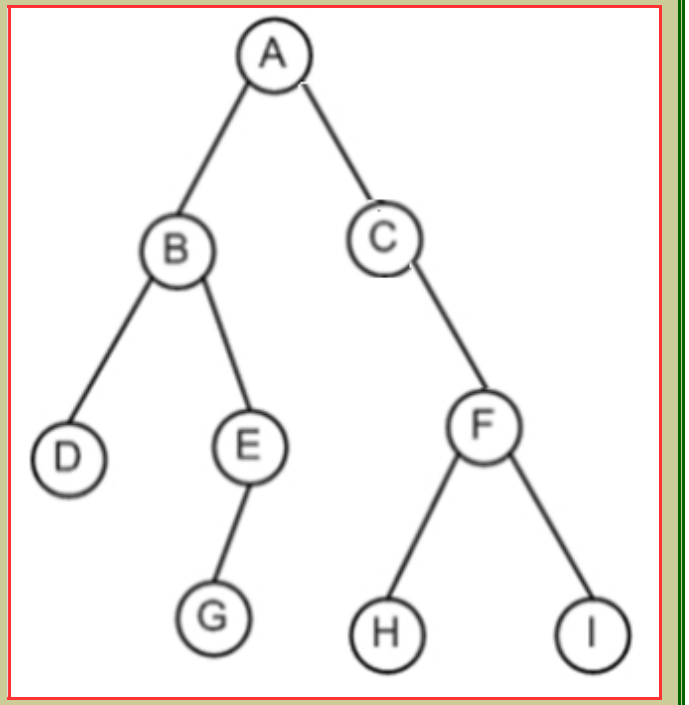
Árvore Binária

Definição

- Uma **árvore binária** é um conjunto finito de elementos (nodos) que pode ser vazio ou particionado em três subconjuntos
  - **raiz** da árvore (elemento inicial, que é único)
  - subárvore **esquerda** (se vista isoladamente forma uma outra árvore)
  - subárvore **direita** (se vista isoladamente forma uma outra árvore)
- Como a definição de árvore é recursiva, muitas operações sobre árvores binárias são definidas de forma recursiva
- Uma árvore onde cada nodo, que não seja folha, tem subárvores esquerda e direita não vazias, é conhecida como **árvore estritamente binária**
- Uma **árvore estritamente binária** com **n** folhas tem **2n-1** nodos
  - o número de folhas é maior, em uma unidade, do que o número dos restantes nodos (que não são folhas)

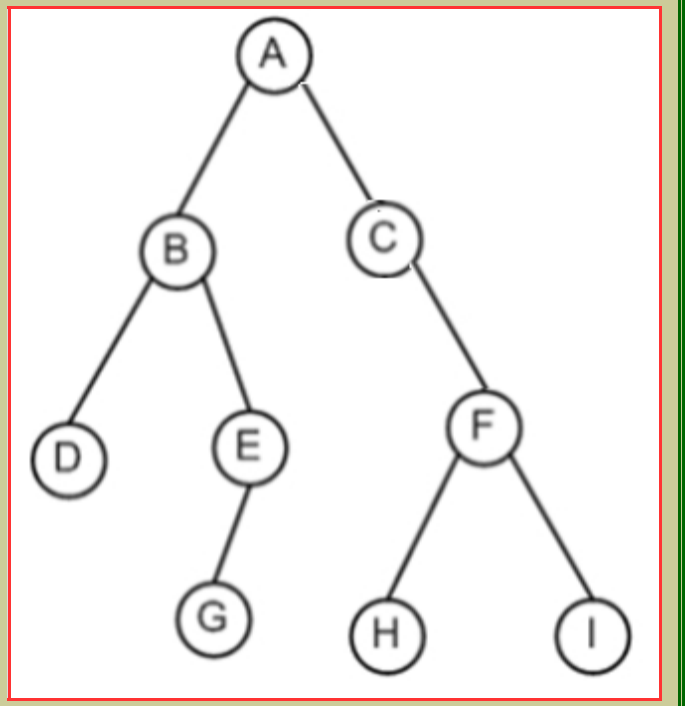
Conceitos gerais

- O nodo **A** é a **raiz** da árvore
- A subárvore **esquerda** tem como **raiz** o nodo **B**
- A subárvore **direita** tem como **raiz** o nodo **C**
- Os nodos **D**, **G**, **H** e **I** são **folhas**
- **A** é **pai** de **B** e **B** é **filho** de **A**
- Os nodos que se encontram no mesmo nível e que têm o mesmo pai, denominam-se de **irmãos**
- **B** e **C** são **irmãos** (e **filhos** de **A**)
- **D** e **E** são **irmãos** (e **filhos** de **B**)
- **H** e **I** são **irmãos** (e **filhos** de **F**)



Conceitos gerais

- $\mathbf{T_A}$ é a **árvore enraizada** no nodo A (toda a árvore)
- $\mathbf{T_F}$ é a **subárvore enraizada** no nodo **F**, a qual contém os nós **F**, **H** e **I**
- Os **graus** do nodo **B** é **2** e do nodo **C** é **1**
- O nodo **A** tem **nível 0**
- Os nodos **B** e **C** têm **nível 1**
- Os nodos **D**, **E** e **F** têm **nível 2**
- Os nodos **G**, **H** e **I** têm **nível 3**
- A **altura** de $\mathbf{T_A}$ é **3**
- A **altura** de $\mathbf{T_C}$ é **2**
- A **altura** de $\mathbf{T_E}$ é **1**
- A **altura** de $\mathbf{T_H}$ é 0



Conceitos gerais

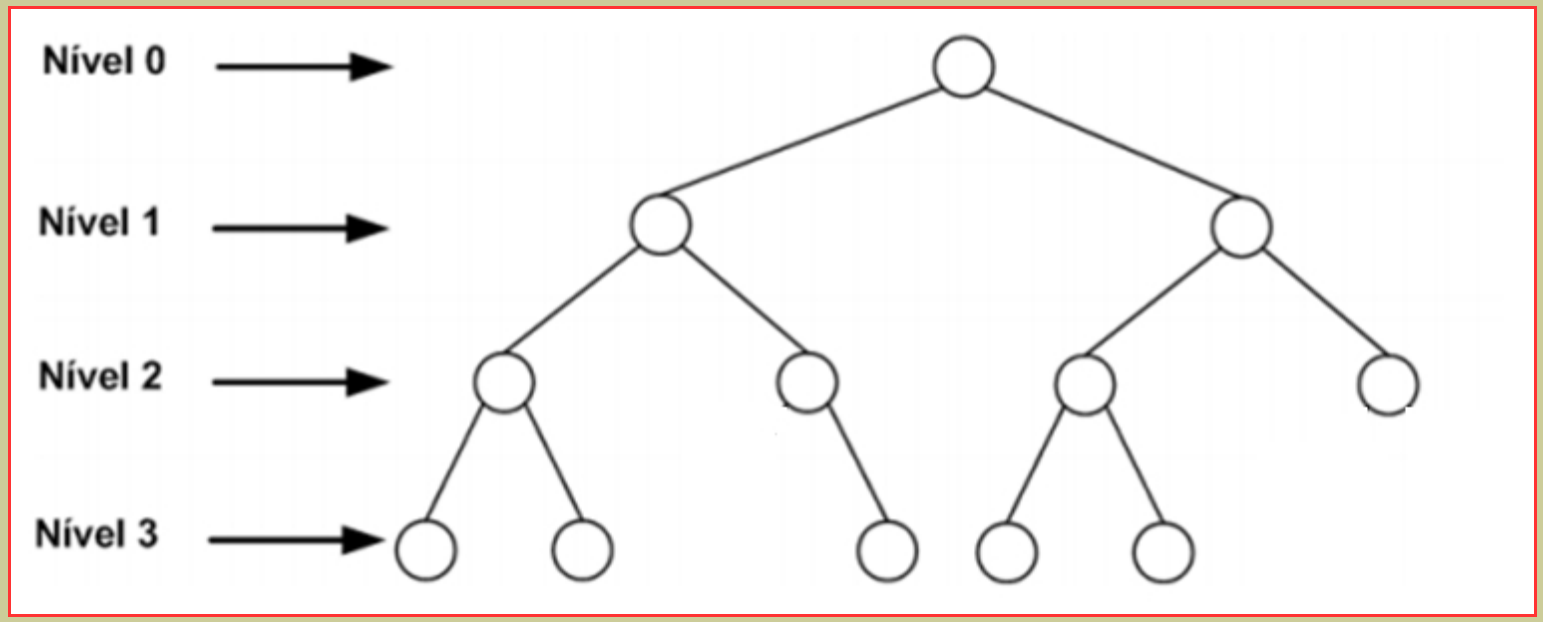
- Um **caminho** da árvore
  - é composto por uma **sequência de nodos consecutivos** (n_1, n_2, n_3, …,n_{k-1}, n_k) tal que existe sempre a relação: n_j é pai de n_{j+1}
  - os k nodos formam um caminho de comprimento k-1
- O **comprimento do caminho** entre o nodo A e o nodo H é 3 (figura anterior)
- A **altura de uma árvore** pode também ser definida como o comprimento do caminho entre a raiz e um dos seus descendentes no maior nível
- **Descendentes** de um nodo são todos os nodos que podem ser alcançados caminhando-se para baixo a partir daquele nodo
- A **altura** de cada uma das folha de uma árvore é **0**

Conceitos gerais

- Uma **árvore binária completa** é uma árvore onde todos os níveis (excepto possivelmente o último) estão completamente preenchidos com nodos
  - ou seja, as folhas só se podem encontrar nos dois últimos níveis
- Uma **árvore binária cheia** é uma árvore **estritamente binária** na qual todas as folhas estão no mesmo nível k, em que **k** é a **altura** da árvore
- Numa *árvore binária cheia*
  - o número total de nodos é $\mathbf{2^{k+1} - 1}$
  - o número total de folhas é $\mathbf{2^{k}}$
- Embora uma *árvore binária cheia* tenha muitos nodos (o máximo para cada nível), a distância da raiz a uma folha qualquer é relativamente pequena

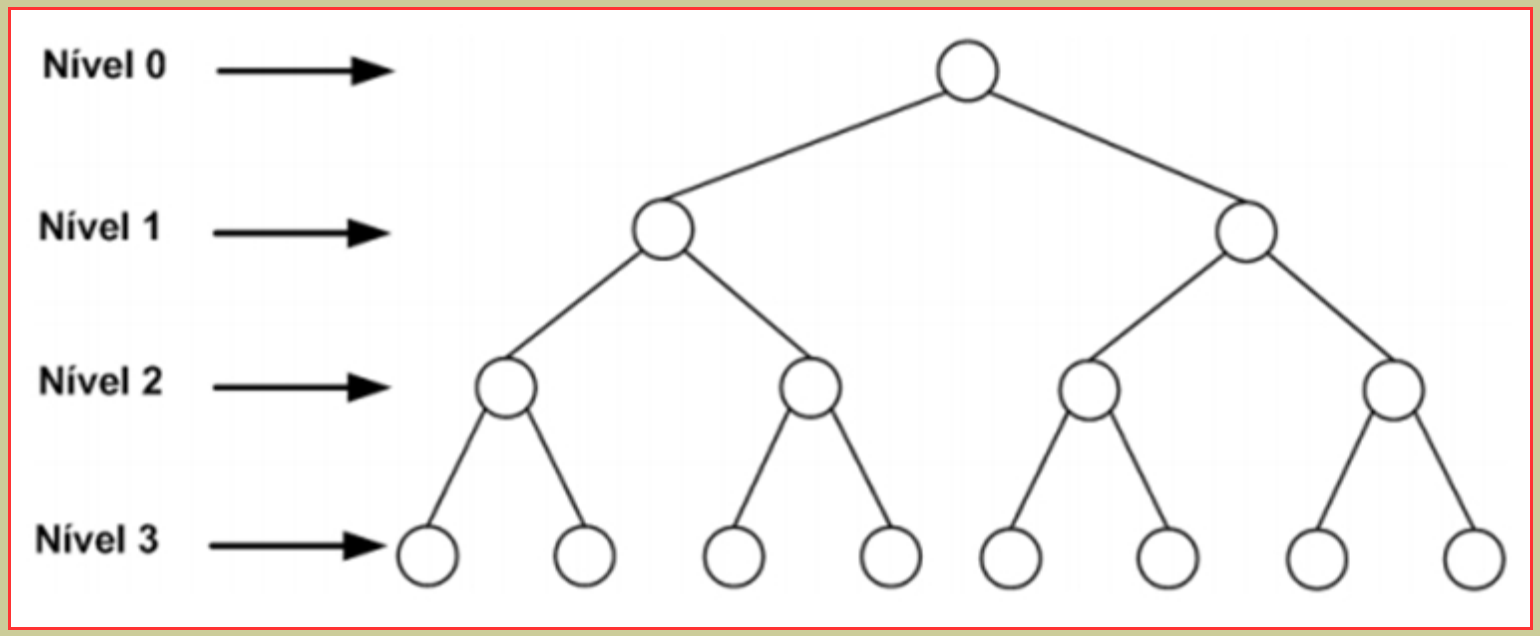
Conceitos gerais

- A figura que se segue
  - representa uma **árvore completa** de **nível 3**
  - tem **altura 3** com **12 nodos** e **6** folhas
  - os níveis 0 (1 nodo), 1 (2 nodos) e 2 (4 nodos) estão completos
  - só há folhas nos níveis 2 (1 folha) e 3 (5 folhas)



Conceitos gerais

- A figura que se segue
  - representa uma **árvore cheia** de **nível 3** (com 4 níveis completos)
  - tem **altura 3** com **15 nodos** (2^4 - 1) e **8** folhas (2^3)



Operações básicas sobre uma Árvore Binária

Criar uma AB (vazia)

Libertar/destruir uma AB

Verificar se uma AB está vazia

Mostrar os elementos de uma AB: pré-ordem, em-ordem e pós-ordem

Determinar a altura de uma AB

Pesquisar um elemento numa AB

Atualizar um elemento de uma AB

Inserir um elemento numa AB

Remover um elemento de uma AB

Nodos e ligações

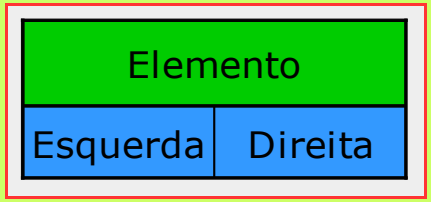
Estrutura

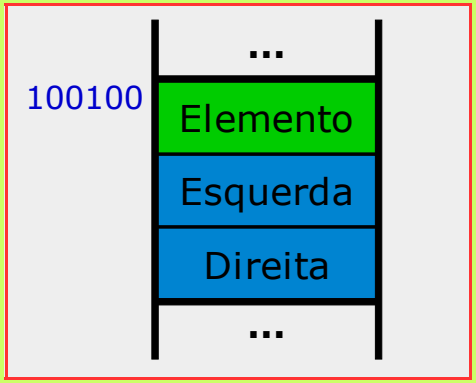
- Um **nodo** pode ser representado por uma estrutura com os seguintes dados
  - a **informação** propriamente dita que se pretende guardar e tratar
  - as **ligações** a outros dois nodos, correspondentes
    - à raiz da **subárvore esquerda**
    - à raiz da **subárvore direita**
- As **ligações** são feitas através de **apontadores**/**ponteiros**, os quais indicam onde
  - está a raiz da subárvore **esquerda**
  - está a raiz da subárvore **direita**

Definição de um nodo

- Em linguagem C

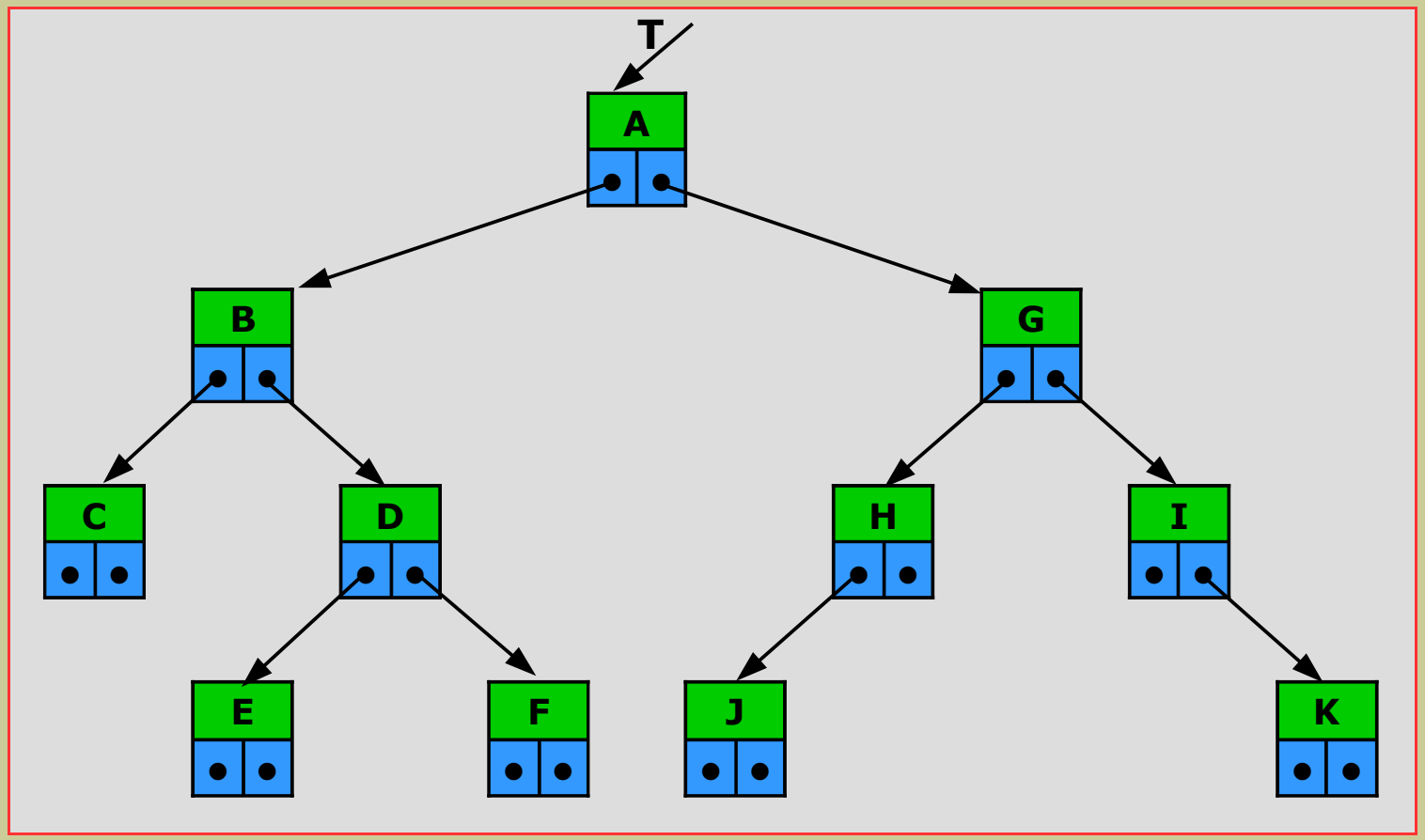
```
struct  NodoAB {
    INFOAB  Elemento;
    struct NodoAB  *Esquerda;
    struct NodoAB  *Direita;
};
typedef  struct NodoAB  *PNodoAB;
```





- definição implementada através de uma estrutura composta por três campos:
  - **Elemento**: INFOAB é o tipo de dados que contém a informação a guardar e tratar
  - **Esquerda**: ligação à raiz da subárvore **esquerda**
  - **Direita**: ligação à raiz da subárvore **direita**
- definição perante um problema de circularidade, pois os campos **Esquerda** e **Direita**
  - são apontadores para elementos do tipo **NodoAB**
  - fazem parte da estrutura **NodoAB**

Representação gráfica



- O identificador **T** é um **ponteiro** para a **raiz** da **Árvore**
- É a partir deste identificador que se chega a todos os nodos da Árvore

Operações sobre a estrutura INFOAB

Mostrar os dados de um elemento do tipo INFOAB

void **mostrarElementoAB** (**INFOAB** X)

- operação que mostra os dados/informação do elemento X do tipo INFOAB

Criar um elemento do tipo INFOAB

INFOAB **criarElementoAB** ()

- operação para criar um elemento do tipo INFOAB, com dados vindos de fonte adequada

Comparar dois elementos do tipo INFOAB

int **compararElementosAB** (**INFOAB** X, **INFOAB** Y)

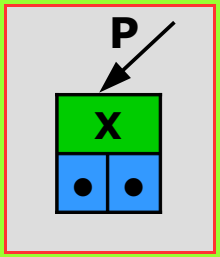
- operação que compara dois elementos do tipo INFOAB (segundo o campo que é a chave)
- devolve: **-1** (X < Y), **0** (X = Y) ou **1** (X > Y)

Operações básicas sobre um nodo

Criar um nodo de uma Árvore Binária

Entrada: informação/dados que se pretende guardar na AB
Saída: um ponteiro para um nodo (informação + ligações)

```
PNodoAB criarNodoAB (INFOAB X)
{
  PNodoAB  P;
  P = (PNodoAB) malloc(sizeof(struct NodoAB));
  if (P == NULL)
     return NULL;
  P→Elemento = X;
  P→Esquerda = NULL;
  P→Direita = NULL;
  return P;
}
```



Libertar/destruir um nodo de uma Árvore Binária

Entrada: um ponteiro para o nodo que se pretende destruir
Saída: o ponteiro a apontar para NULL

```
PNodoAB libertarNodoAB (PNodoAB P)
{
  P→Esquerda = NULL;
  P→Direita = NULL;
  free(P);
  P = NULL;
  return  P;
}
```

Operações básicas sobre uma Árvore Binária

Criar uma Árvore Binária

Entrada: --
Saída: devolve um Árvore Binária vazia (sem elementos)

```
PNodoAB criarAB ()
{
  PNodoAB T;
  T = NULL;
  return T;
}
```

Libertar/destruir uma Árvore Binária

- Operação para libertar todos os nodos de uma Árvore Binária

Entrada: uma Árvore Binária T (a que se pretende libertar/destruir)
Saída: a Árvore Binária T vazia (T a apontar para NULL)

```
PNodoAB destruirAB (PNodoAB T)
{
  if (T == NULL)
      return T;
  T→Esquerda = destruirAB(T→Esquerda);
  T→Direita = destruirAB(T→Direita);
  T = libertarNodoAB(T);
  return T;
}
```

Verificar se uma Árvore Binária está vazia

Entrada: uma Árvore Binária T

Saída: 1 (se T está vazia) ou 0 (se T não está vazia)

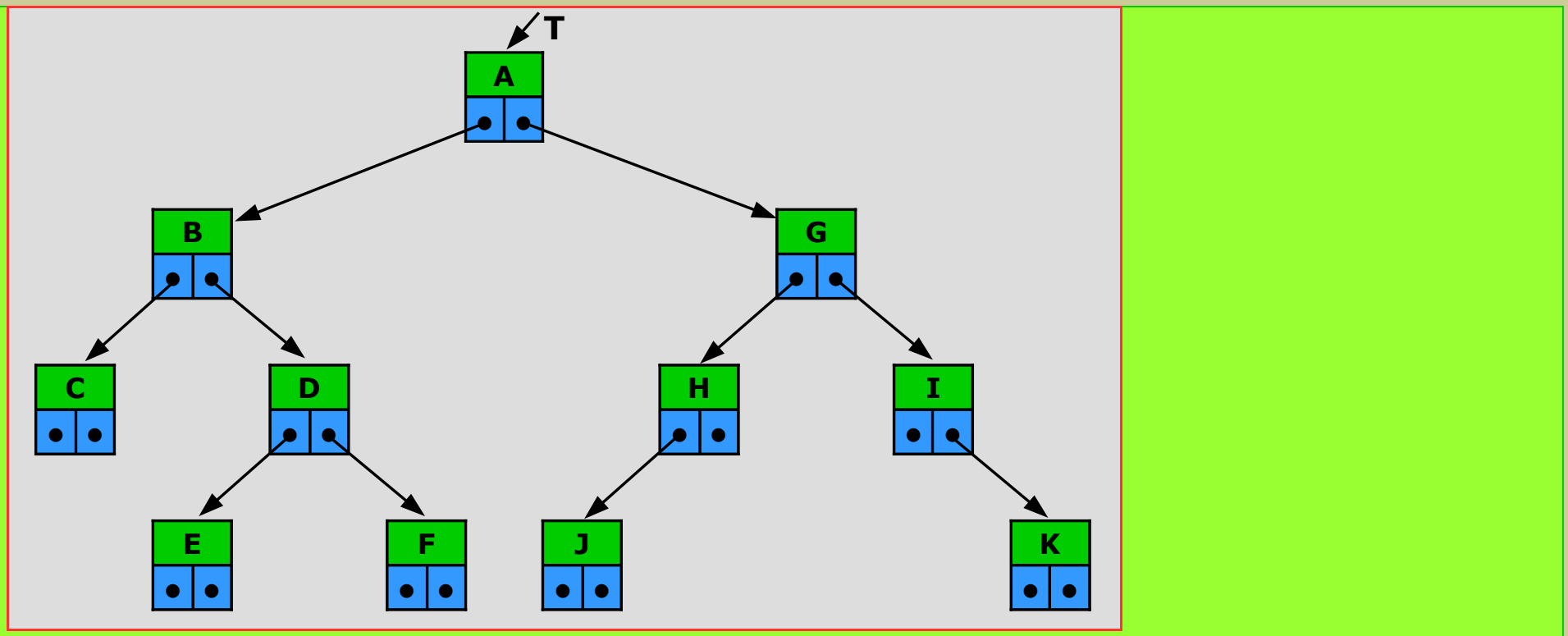
```
int ABVazia (PNodoAB T)
{
  if (T == NULL)
     return 1;
  else
     return 0;
}
```

Mostrar os elementos de uma Árvore Binária

- Para se mostrar os elementos de uma Árvore Binária, deve-se percorrâ-la de uma forma organizada
- Uma Árvore Binária pode ser percorrida de duas formas:
  - em profundidade:
    - travessia em *em-ordem*: subárvore esquerda, raíz e subárvore direita
    - travessia em *pré-ordem*: raíz, subárvore esquerda e subárvore direita.
    - travessia em *pós-ordem*: subárvore esquerda, subárvore direita e raíz.
  - em largura:
    - travessia por níveis: nível 0 (raiz), nivel 1, nivel 2, ...

Mostrar os elementos de uma Árvore Binária

- Exemplo de como percorrer os elementos de uma Árvore Binária



em-ordem: C ⇒ B ⇒ E ⇒ D ⇒ F ⇒ A ⇒ J ⇒ H ⇒ G ⇒ I ⇒ K

pré-ordem: A ⇒ B ⇒ C ⇒ D ⇒ E ⇒ F ⇒ G ⇒ H ⇒ J ⇒ I ⇒ K

pós-ordem: C ⇒ E ⇒ F ⇒ D ⇒ B ⇒ J ⇒ H ⇒ K ⇒ I ⇒ G ⇒ A

por níveis: 0: A # 1: B ⇒ G # 2: C ⇒ D ⇒ H ⇒ I # 3: E ⇒ F ⇒ J ⇒ K

Mostrar os elementos de uma Árvore Binária

- Operação para fazer a travessia em **em-ordem** (esquerda, raíz e direita)

Entrada: uma Árvore Binária T
Saída: -- (mostra os elementos da árvores pela ordem indicada)

```
void mostrarEmOrdemAB (PNodoAB T)
{
  if (T != NULL) {
     mostrarEmOrdemAB(T→Esquerda);
     mostrarElementoAB(T→Elemento);
     mostrarEmOrdemAB(T→Direita);
  }
}
```

Mostrar os elementos de uma Árvore Binária

- Operação para fazer a travessia em **pré-ordem** (raíz, esquerda e direita)

Entrada: uma Árvore Binária T
Saída: -- (mostra os elementos da árvores pela ordem indicada)

```
void mostrarPreOrdemAB (PNodoAB T)
{
  if (T != NULL) {
     mostrarElementoAB(T→Elemento);
     mostrarPreOrdemAB(T→Esquerda);
     mostrarPreOrdemAB(T→Direita);
  }
}
```

Mostrar os elementos de uma Árvore Binária

- Operação para fazer a travessia em **pós-ordem** (esquerda, direita e raiz)

Entrada: uma Árvore Binária T
Saída: -- (mostra os elementos da árvores pela ordem indicada)

```
void mostrarPosOrdemAB (PNodoAB T)
{
  if (T != NULL) {
     mostrarPosOrdemAB(T→Esquerda);
     mostrarPosOrdemAB(T→Direita);
     mostrarElementoAB(T→Elemento);
  }
}
```

Determinar a altura de uma Árvore Binária

Entrada: uma Árvore Binária T

Saída: a altura da Árvore Binária T (-1 se é vazia, 0 se só tem um elemento, ...)

```
int alturaAB (PNodoAB T)
{
  int  altEsq, altDir
  if (T == NULL)
     return  -1;   // por definição, altura de uma árvore vazia é -1
  altEsq = alturaAB(T→Esquerda);
  altDir = alturaAB(T→Direita);
  if (altEsq > altDir)
     return  altEsq + 1;
  else
     return  altDir + 1;
}
```

Pesquisar um elemento numa Árvore Binária

Entrada: o elemento a pesquisar e uma Árvore Binária T
Saída: ponteiro para o nodo com o elemento a pesquisar (existe); NULL (não existe)

```
PNodoAB pesquisarAB (INFOAB X, PNodoAB T)
{
  PNodoAB  P;
  if (T == NULL)
     return NULL;   // o elemento X não existe em T
  if (compararElementosAB(X, T→Elemento) == 0)
     return T;    // foi encontrado um nodo com o elemento igual a X
  P = pesquisarAB(X, T→Esquerda);
  if (P != NULL)
     return P;    // existe um nodo com elemento igual a X na subárvore esquerda
  // procurar um nodo com elemento igual a X na subárvore direita
  return pesquisarAB(X, T→Direita);
}
```

Atualizar um elemento de uma Árvore Binária

Entrada: a informação atual (chave), a nova informação e uma Árvore Binária T
Saída: --

```
void atualizarElementoAB (INFOAB X, INFOAB Novo, PNodoAB T)
{
  PNodoAB  P;
  P = pesquisarAB(X, T);
  if (P != NULL)
     P→Elemento = Novo;
}
```

Inserir um elemento numa Árvore Binária

- O algoritmo de inserção de um elemento numa Árvore Binária
  - começa com uma pesquisa deste elemento na árvore (posição de inserção)
  - o nodo com o novo elemento é sempre inserido como folha da árvore
- Necessidade da pesquisa obedecer a algum critério
  - para existir uniformidade na distribuição dos nodos pela árvore
  - de forma a que árvore não fique totalmente desequilibrada
- Um critério possível
  - a altura dos filhos (subárvores esquerda e direita) de cada nodo da Árvore Binária

Inserir um elemento numa Árvore Binária

Entrada: elemento a inserir e uma Árvore Binária T
Saída: a Árvore Binária T atualizada (a raiz de T pode ter sofrido alteração)

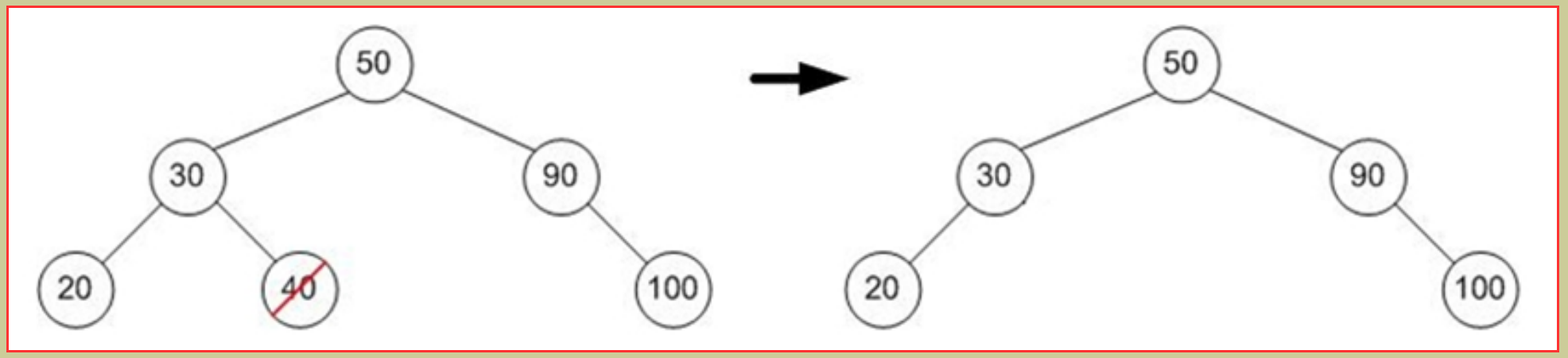
```
PNodoAB inserirPorAlturaAB (INFOAB X, PNodoAB T)
{
  if (T == NULL){
     T = criarNodoAB(X);
     return  T;
  }
  if (alturaAB(T→Esquerda) > alturaAB(T→Direita))
     T→Direita = inserirPorAlturaAB(X, T→Direita);
  else
     T→Esquerda = inserirPorAlturaAB(X, T→Esquerda);
  return  T;
}
```

Remover um elemento de uma Árvore Binária

- Considerar três casos distintos, segundo as caracteristicas do nodo a remover:
  - se é um nodo sem filhos (folha)
  - se é um nodo com um único filho
  - se é um nodo com dois filhos.

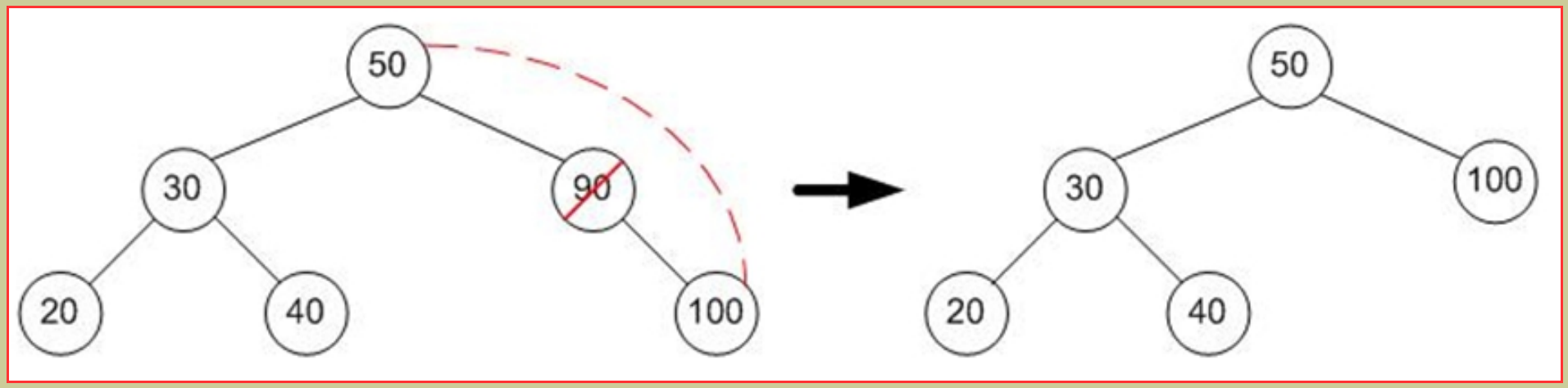
Remover um elemento de uma Árvore Binária

- Remoção de um nodo sem filhos (folha)
  - exemplo: remover o nodo com **40** da árvore em baixo



Remover um elemento de uma Árvore Binária

- Remoção de um nodo com um único filho
  - exemplo: remover o nodo com **90** da árvore em baixo



Remover um elemento de uma Árvore Binária

- Remoção de um nodo com dois filhos
  - o processo implica substituir este nodo por uma das folhas sua descendente:

    1º) escolher uma das folhas descendente do nodo a remover que o irá substituir

    2º) substituir a informação (Elemento) do nodo a remover pela da folha escolhida

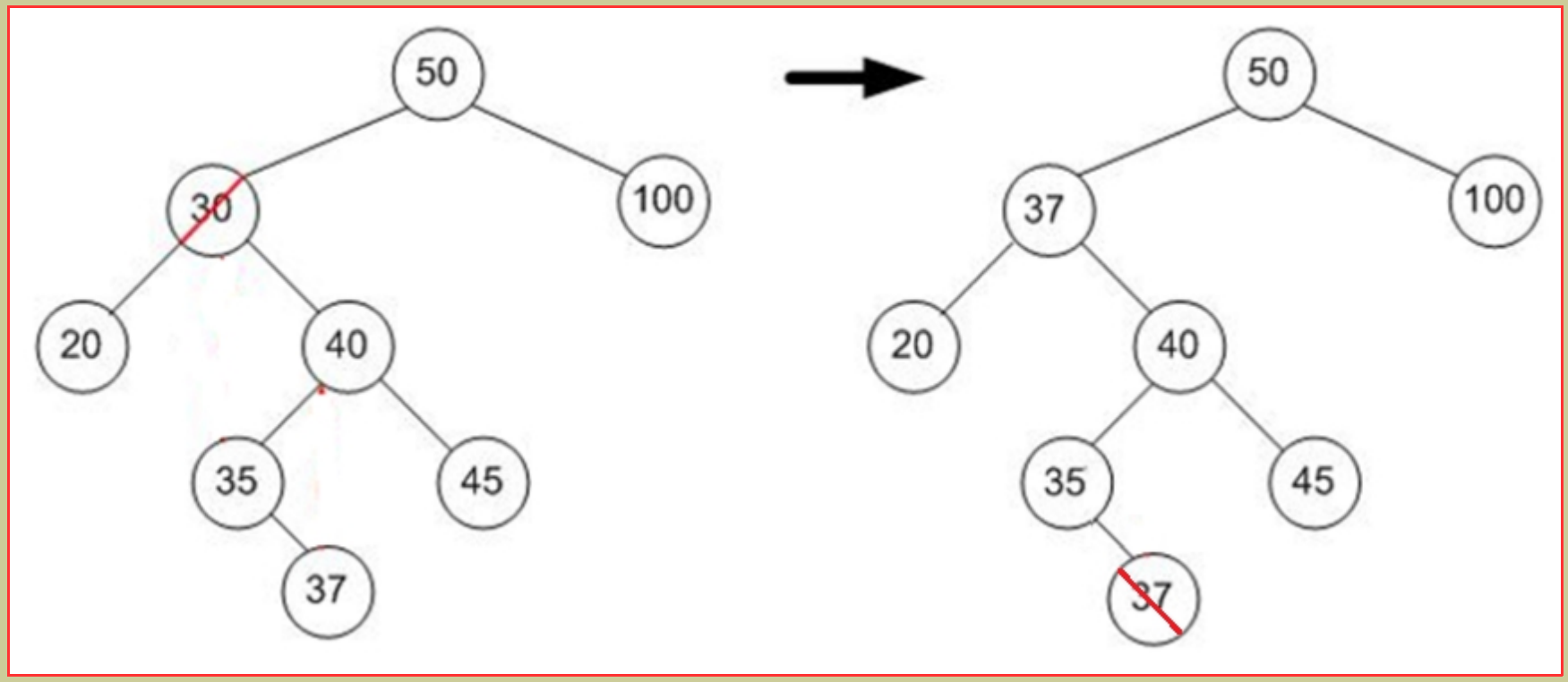
    3º) remover a folha escolhida

Remover um elemento de uma Árvore Binária

- Remoção de um nodo com dois filhos
  - exemplo: remover o nodo com **30** da árvore em baixo

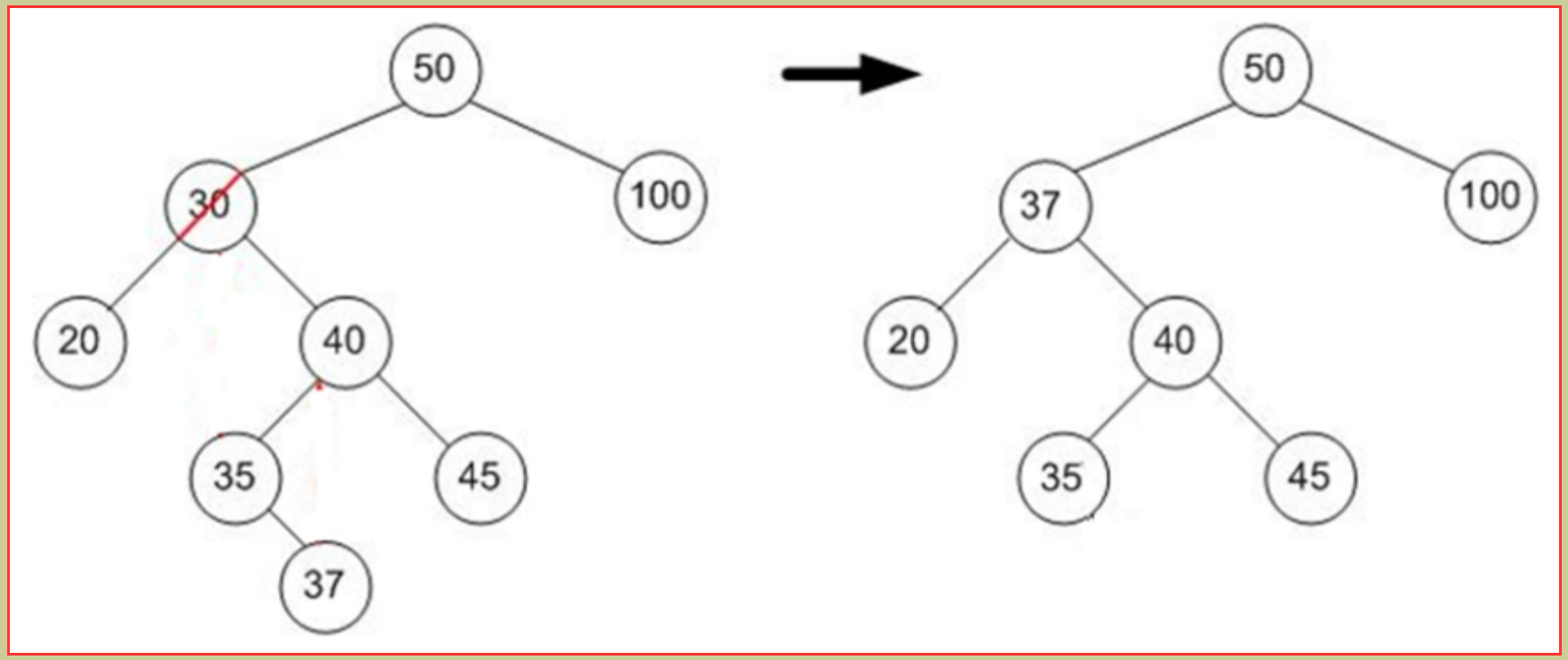
    1º) escolher a folha com **37** (descendente mais longe) - podia ser a com 20 ou 45

    2º) substituir a informação (Elemento) do nodo com **30** pela do nodo com **37**



Remover um elemento de uma Árvore Binária

- Remoção de um nodo com dois filhos
  - exemplo: remover o nodo com **30** da árvore em baixo
    1º) escolher a folha com **37** (descendente mais longe) - podia ser a com 20 ou 45
    2º) substituir a informação (Elemento) do nodo com **30** pela do nodo com **37**
    3º) remover a folha escolhida (também com **37**)



Remover um elemento de uma Árvore Binária

Entrada: o elemento X a remover e uma Árvore Binária T
Saída: a Árvore Binária T atualizada (a raiz de T pode ter sofrido alteração)

```
PNodoAB removerAB (INFOAB X, PNodoAB T)
{
  PNodoAB  P;
  if (compararElementosAB(T→Elemento, X) == 0) {   // T→Elemento = X
     T = removerNodoAB(T);
     return T;
  }
  P = pesquisarAB(X, T→Esquerda);
  if (P != NULL)
     T→Esquerda = removerAB(X, T→Esquerda);
  else
     T→Direita = removerAB(X, T→Direita);
  return T;
}
```

Remover um elemento de uma Árvore Binária – operação auxiliar

Entrada: ponteiro para o nodo que se pretende remover (subárvore T não vazia)
Saída: ponteiro para a raíz da subárvore sem aquele nodo (pode ter sido alterada)

```
PNodoAB removerNodoAB (PNodoAB T) {
  PNodoAB P;
  INFOAB X;
  if (T→Esquerda == NULL  &&  T→Direita == NULL) {   // T é uma folha
     T = libertarNodoAB(T);
     return T;    // T = NULL
  }
  if (T→Esquerda == NULL) {      // T só tem um filho: o direito
     P = T;
     T = T→Direita;
     P = libertarNodoAB(P);
     return T;
  }
...
```

Remover um elemento de uma Árvore Binária – operação auxiliar

```
...
  if (T→Direita == NULL){      //  T só tem um filho: o esquerdo
      P = T;
      T = T→Esquerda;
      P = libertarNodoAB(P);
      return T;
  }
  // T tem dois filhos: remover a folha escolhida e copiar a sua informação
  T = procurarFolhaAB(T, &X);
  T→Elemento = X;
  return T;
}
```

Remover um elemento de uma Árvore Binária – operação auxiliar

Operação (recursiva): procurar uma folha descendente do nodo a remover
Entrada: ponteiro para o nodo que se pretende remover (subárvore T não vazia)
Saída: ponteiro para a raíz da subárvore sem aquele nodo (pode ter sido alterada)

```
PNodoAB procurarFolhaAB (PNodoAB T, INFOAB *X){
  PNodoAB P;
  if (T→Esquerda == NULL  &&  T→Direita == NULL) {
     *X = T→Elemento;
     T = libertarNodoAB(T);
     return NULL;  //  ou: return T;
  }
  if (alturaAB(T→Esquerda) > alturaAB(T→ Direita))
     T→Esquerda = procurarFolhaAB(T→Esquerda, X);
  else
     T→Direita = procurarFolhaAB(T→ Direita, X);
  return T;
}
```

Travessias não recursivas de uma Árvore Binária

- Necessário utilizar uma memória para guardar os nodos já visitados e/ou por visitar, de forma a poder-se regressar posteriormente a estes nodos
- Como o caminho de regresso é normalmente o caminho percorrido no sentido contrário, devemos usar uma EAD Pilha ou uma EAD Fila para tal
- Nas travessias mais comuns (em pré-ordem, em-ordem e pós-ordem)
  - utiliza-se uma EAD Pilha para armazenar os nodos já visitados
  - definição do tipo de dados INFOPilha e da EAD Pilha

```
typedef PNodoAB INFOPilha;
struct NodoPilha {
    INFOPilha Elemento;
    struct NodoPilha *Ant;
};
typedef struct NodoPilha *PNodoPilha;
```

Travessias não recursivas de uma Árvore Binária

Operação: travessia em **pré-ordem** (raíz, esquerda e direita)

```
void mostrarPreOrdemAB (PNodoAB T) {
  PNodoPilha  S = criarPilha();
  PNodoAB  P;
  if (T == NULL)
     return;
  S = push(T, S);
  while (pilhaVazia(S) == 0) {
     P = topo(S);
     S = pop(S);
     mostrarElementoPilha(P→Elemento);
     if (P→Direita != NULL)
        S = push(P→Direita, S);
     if (P→Esquerda != NULL)
        S = push(P→Esquerda, S);
  }
}
```

Travessias não recursivas de uma Árvore Binária

- Como se pode verificar, as versões recursivas dos algoritmos de travessias estudadas são muito mais fáceis de implementar do que as correspondentes versões iterativas
- No entanto, num outro tipo de travessia isso não acontece, como é o caso da **travessia por níveis**
- Na travessia por níveis
  - utiliza-se uma EAD Fila para armazenar os nodos já visitados
  - definição do tipo de dados INFOFila e da EAD Fila

```
typedef  PNodoAB  INFOFila;
struct NodoFila {
   INFOFila  Elemento;
   struct NodoFila *Seg;
};
typedef struct NodoFila *PNodoFila;
```

Travessias não recursivas de uma Árvore Binária

Operação: travessia **por níveis**

```
void mostrarPorNiveisAB (PNodoAB T) {
  PNodoFila Q = criarFila();
  PNodoAB  P;
  if (T == NULL)
     return;
  Q = juntar(T, Q);
  while (filaVazia(Q) == 0) {
     P = frente(Q);
     mostrarElementoFila(P→Elemento);
     Q = remover(Q);
     if (P→Esquerda != NULL)
        Q = juntar(P→Esquerda, Q);
     if (P→Direita != NULL)
        Q = juntar(P→Direita, Q);
  }
}
```