Árvores binárias
(IED-001)

Prof. Dr. Silvio do Lago Pereira

Departamento de Tecnologia da Informação

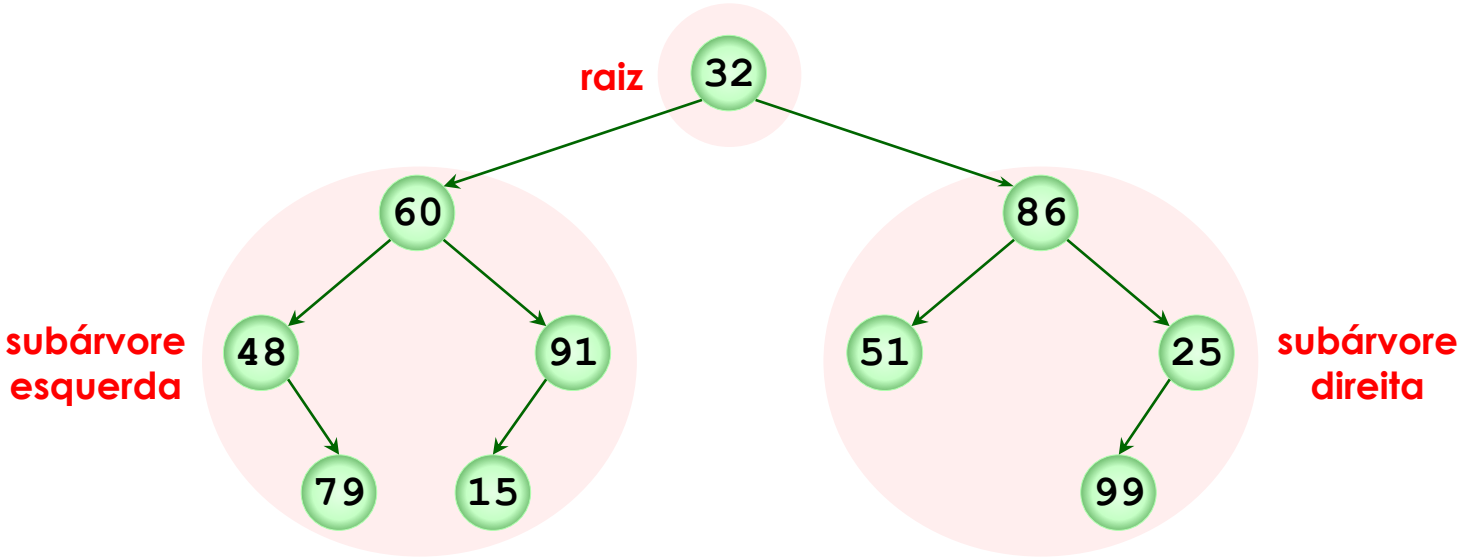
Faculdade de Tecnologia de São Paulo

Árvores binárias

Árvore binária

é uma coleção A de itens organizados **hierarquicamente** tal que, se A não é vazia, então:

- Há um item em A, chamado **raiz**, ao qual os demais itens de A estão subordinados.
- Os demais itens são divididos em **duas** coleções disjuntas, chamadas **subárvores** de A.
- As subárvores de A também são **árvores binárias**.



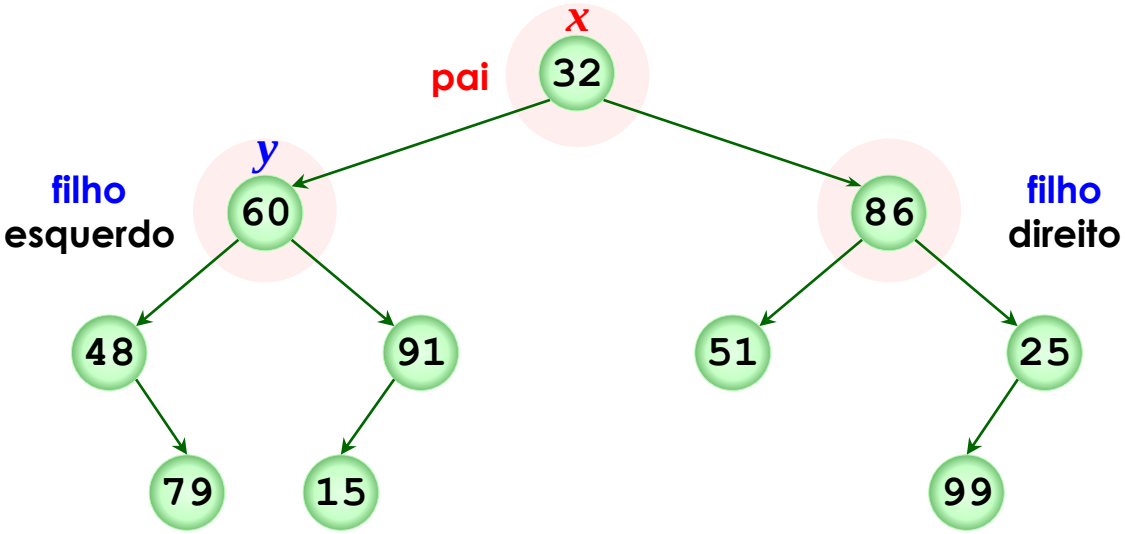
Uma árvore binária é, por definição, uma estrutura recursiva!

Árvores binárias

Pais e filhos

Sejam dois itens $x, y \in A$, tais que y é **diretamente acessível** a partir de x. Então:

- O item x é o **pai** do item y.
- O item y é um **filho** do item x.

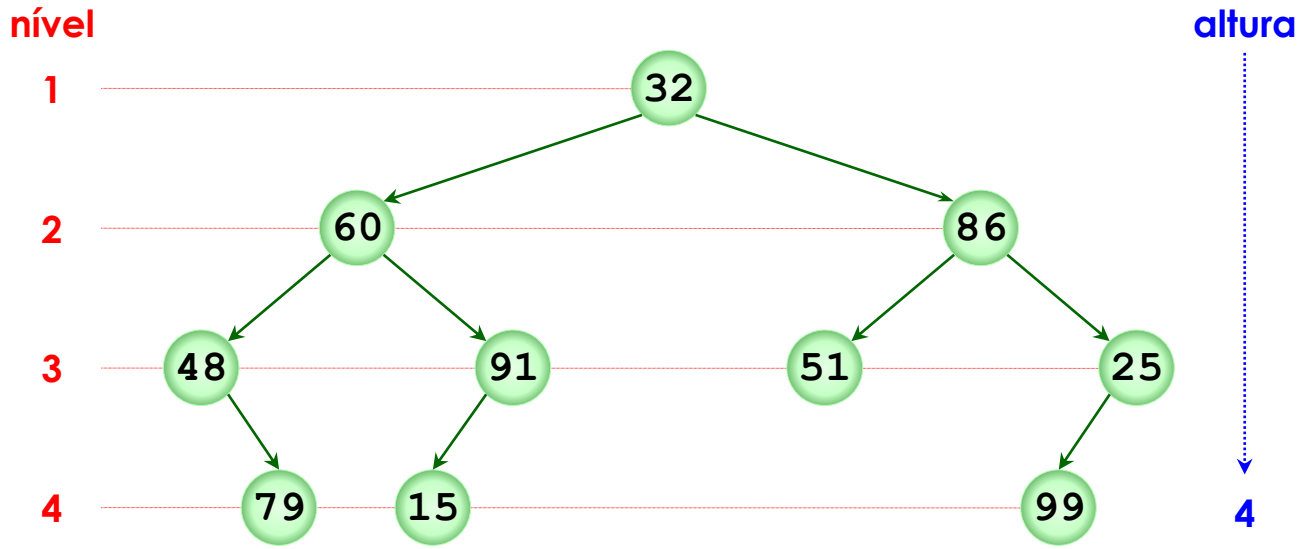


Numa árvore binária, a **raiz** é um item que não tem pai e uma **folha** é um item que não tem filho!

Árvores binárias

Nível e altura

- O **nível** da raiz de uma árvore binária é **1**.
- O **nível** de um item que é filho de um item num nível ***h*** é ***h*+1** .
- A **altura** de uma árvore binária não vazia é o máximo nível de seus itens.

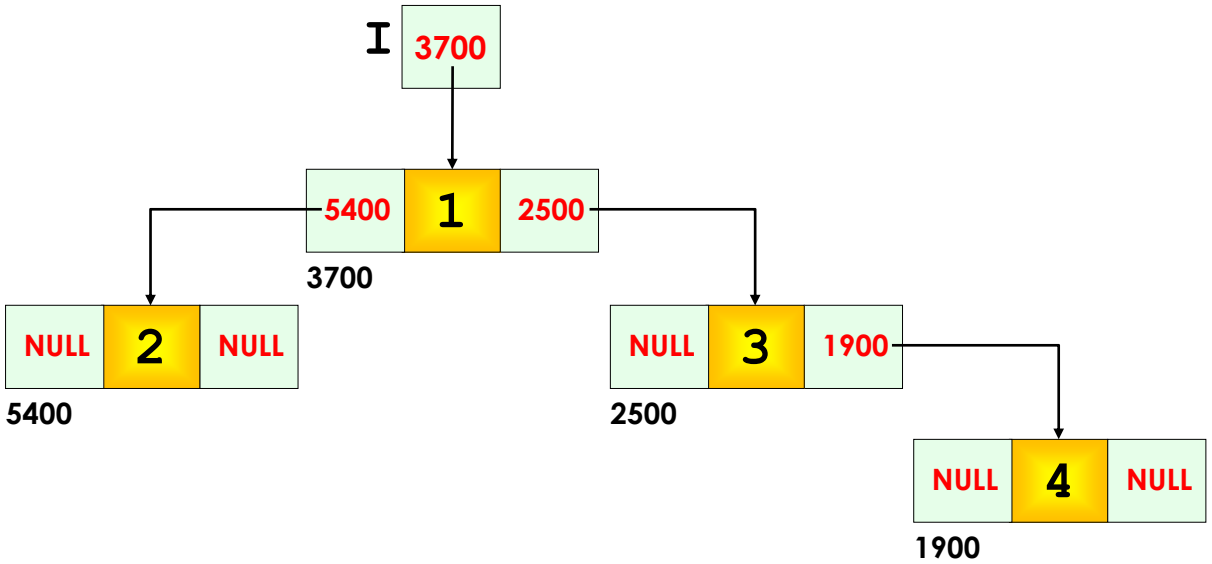


Por definição, uma **árvore binária vazia** tem altura **0**.

Árvores binárias

Nó de árvore binária

Cada nó de uma árvore binária é uma estrutura que guarda um item e dois ponteiros.



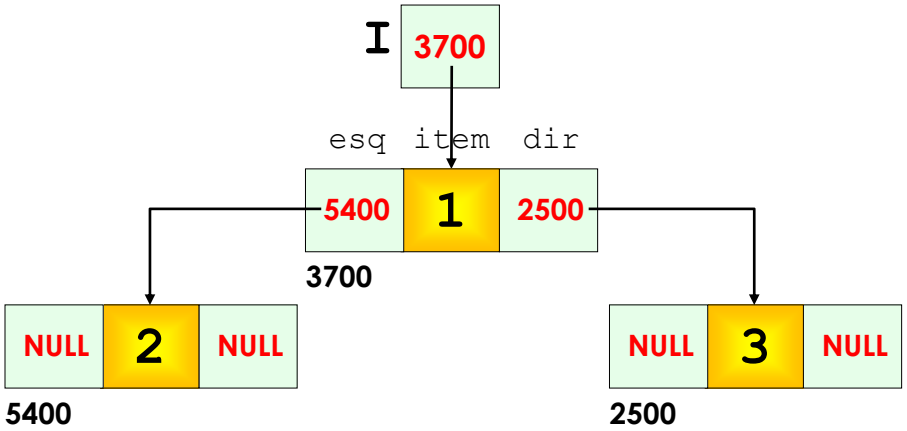
Observações:

- O ponteiro inicial `I` aponta o nó **raiz** da árvore.
- Os ponteiros dentro dos nós aponta seus **filhos** (da esquerda e da direita).
- Os nós com dois ponteiros nulos são denominados **folhas**.

Árvores binárias

Exemplo 1. O tipo `Arv`

```
typedef int Item;
typedef struct arv {
    struct arv *esq;
    Item item;
    struct arv *dir;
} *Arv;
```



`Arv` ≡ `struct arv *`

Observações:

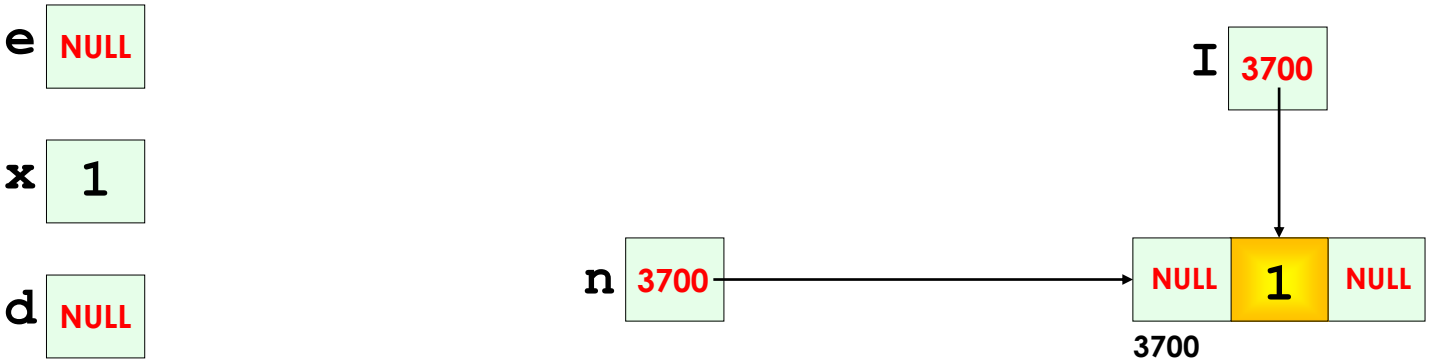
- O tipo `Item` indica o tipo dos itens armazenados na árvore binária.
- O tipo `Arv` é usado para declarar um ponteiro de árvore binária (que aponta a raiz da árvore).
- Se `I` é um ponteiro de árvore binária nulo, então a árvore binária está **vazia**.
- Se `I` é um ponteiro de árvore binária não-nulo, então `I->item` é a **raiz** da árvore, `I->esq` é sua subárvore **esquerda** e `I->dir` é sua subárvore **direita**.

Árvores binárias

Exemplo 2. Criação de nó

```
Arv arv(Arv e, Item x, Arv d) {
    Arv n = malloc(sizeof(struct arv));
    n->esq  = e;
    n->item = x;
    n->dir  = d;
    return n;
  }
```

```
Arv I = arv(NULL,1,NULL);
```

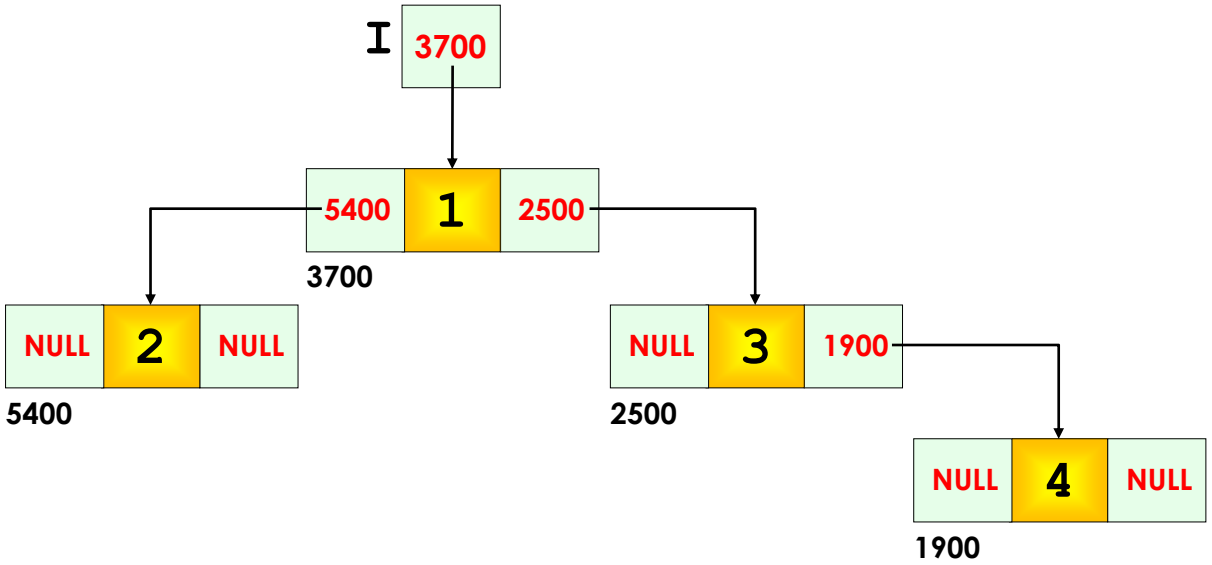


Usando a função **`arv()`** é possível criar qualquer árvore binária desejada!

Árvores binárias

Exemplo 3. Criação de árvore binária com a função `arv()`

```
Arv I = arv(arv(NULL,2,NULL),1,arv(NULL,3,arv(NULL,4,NULL)));
```



Com a função **`arv()`**, a árvore binária é construída de baixo para cima (*bottom-up*)!

Árvores binárias

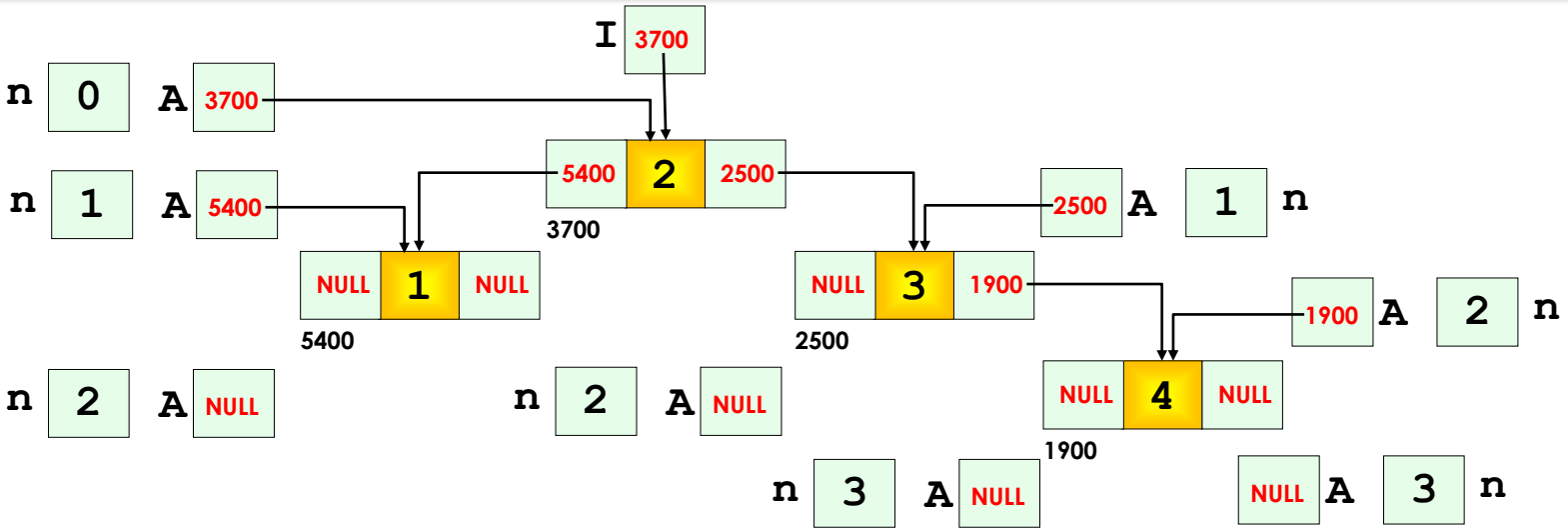
Exemplo 4. Exibição de árvore binária

```
void exibe(Arv A,int n) {
    if( A==NULL ) return;
    exibe(A->dir,n+1);
    printf("%*s%d\n",3*n,"",A->item);
    exibe(A->esq,n+1);
}
```

```
exibe(I,0);
```

vídeo

```
        4
    3
2
    1
```



Note que essa função exibe a árvore de modo que possamos ver a hierarquia entre os itens!

Árvores binárias

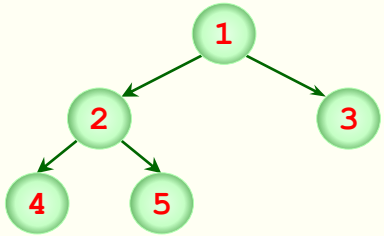
Exercício 1. Programa para criação e exibição de árvore binária

Complete e execute o programa a seguir.

```
#include <stdio.h>
#include <stdlib.h>
...
int main(void) {
    Arv I = arv(arv(NULL,2,NULL),1,arv(NULL,3,arv(NULL,4,NULL)));
    exibe(I,0);
    return 0;
}
```

Exercício 2. Teste com outra árvore

Altere o programa anterior para criar e exibir a árvore a seguir.



Árvores binárias

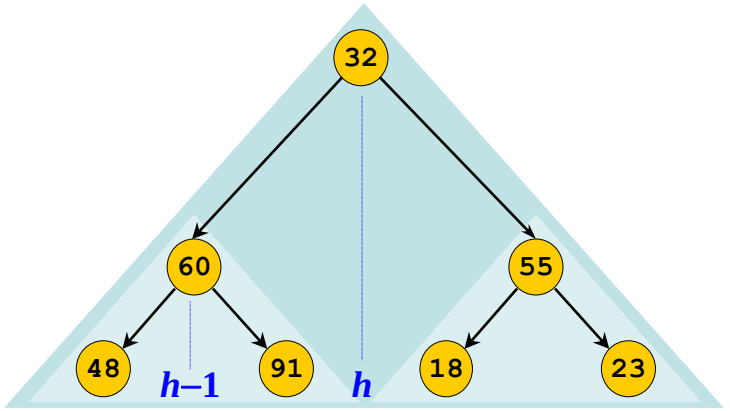
Exemplo 5. Criação de árvore binária completa aleatória, com altura *h*

```
Arv completa(int h) {
    if( h==0 ) return NULL;
    return arv(completa(h-1), rand()%100, completa(h-1));
}
```

Exercício 3. Teste da função `completa()`

Complete e execute o programa a seguir.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
...
int main(void) {
    srand(time(NULL));
    Arv A = completa(3);
    exibe(A,0);
    return 0;
}
```



árvore binária completa, com altura h=3

Uma **árvore binária completa** é uma árvore binária em que todos os níveis estão completos!

Árvores binárias

Exercício 4. Árvore binária balanceada, com *n* itens aleatórios

Uma árvore binária é **balanceada** se, para cada nó, a diferença entre o número de descendentes à esquerda e o número de descendentes à direita é no máximo 1. Crie a função **`balanceada(n)`**, que devolve uma árvore balanceada com **`n`** itens aleatórios, e execute o programa a seguir.

```
int main(void) {
    srand(time(NULL));
    exibe(balanceada(9),0);
    return 0;
}
```

Exercício 5. Árvore binária aleatória, com *n* itens aleatórios

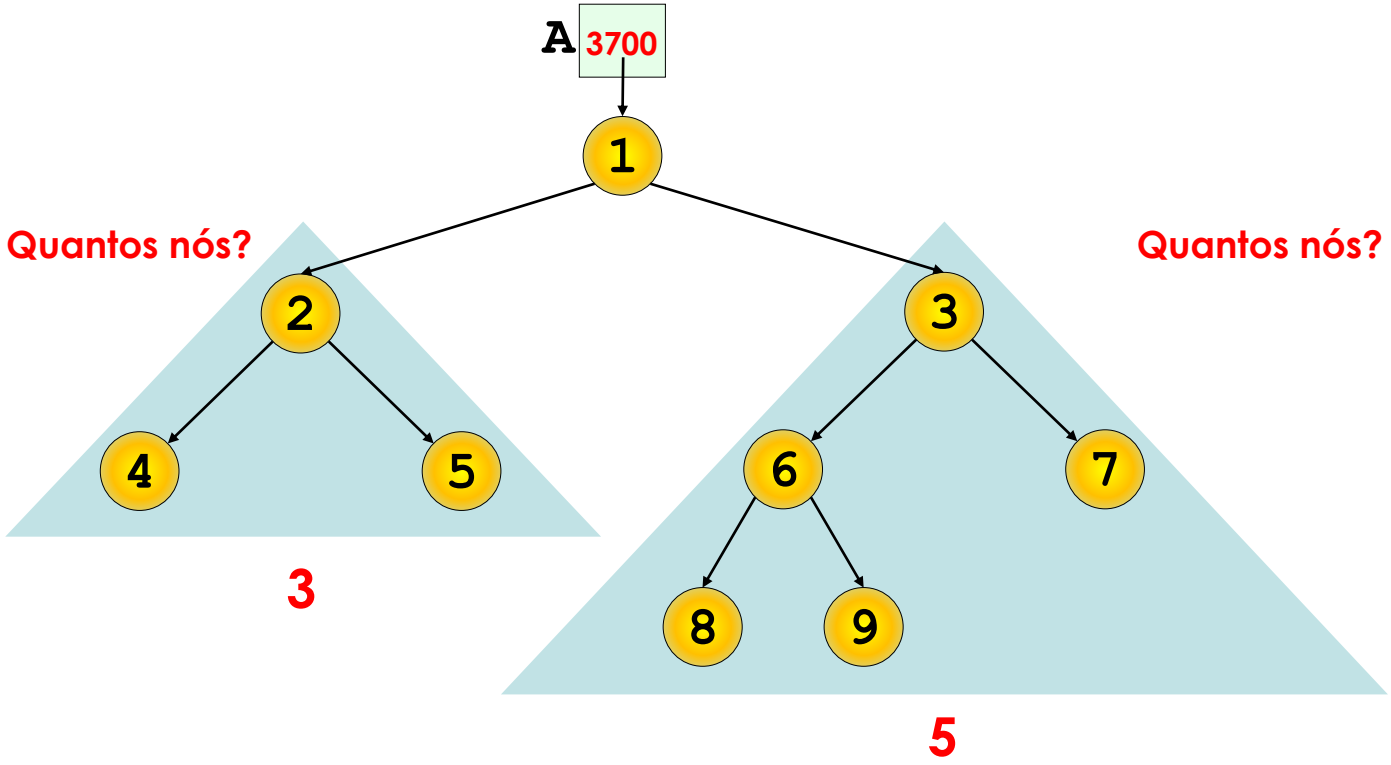
Uma árvore binária **aleatória** é uma árvore binária qualquer. Crie a função **`aleatoria(n)`**, que devolve uma árvore binária aleatória, com **`n`** itens aleatórios, e execute o programa a seguir.

```
int main(void) {
    srand(time(NULL));
    exibe(aleatoria(9),0);
    return 0;
}
```

Árvores binárias

Exercício 6. Quantidade de nós numa árvore binária

Crie a função recursiva **`nos(A)`**, que devolve a quantidade de nós existentes na árvore binária **`A`**. Em seguida, faça um programa para testar o funcionamento da função.

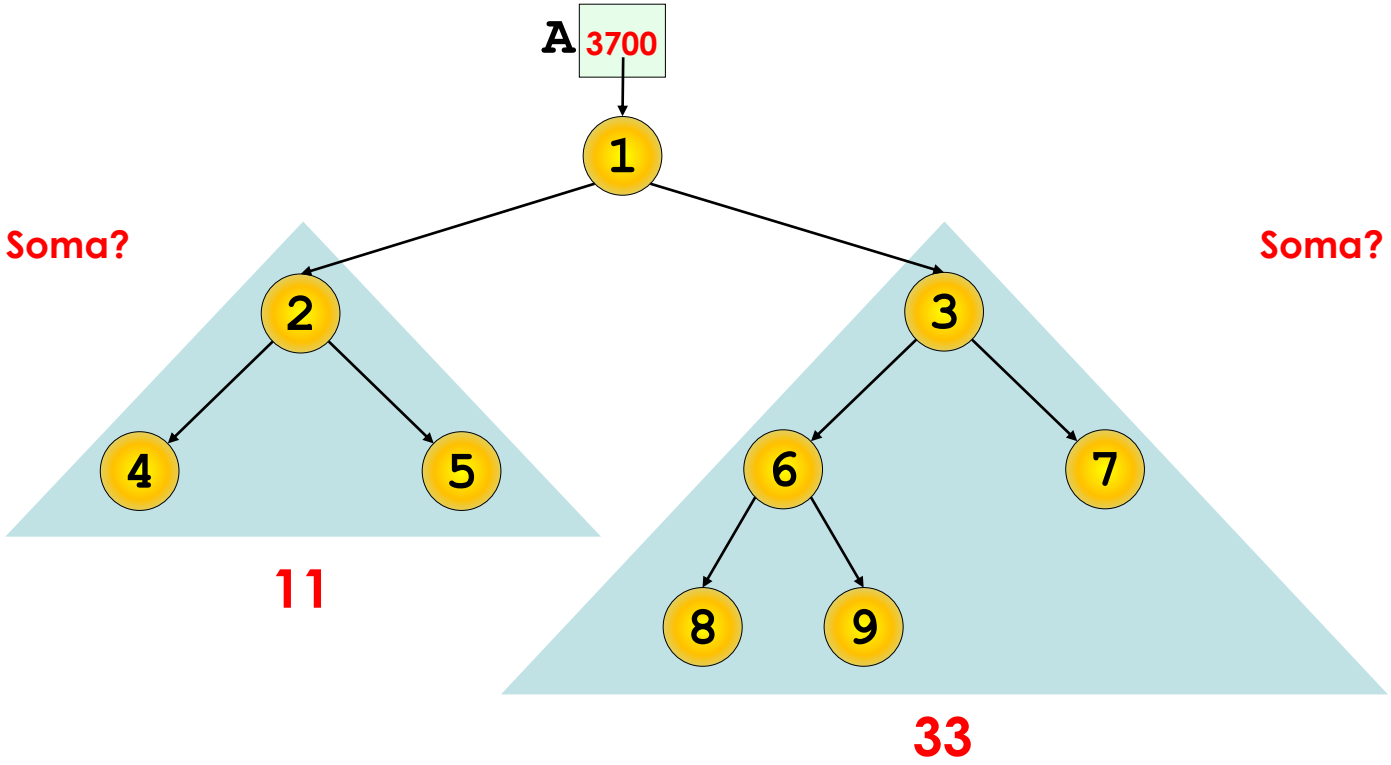


No caso geral, a quantidade de nós é igual a 1 + quantidade da esquerda + quantidade da direita!

Árvores binárias

Exercício 7. Soma dos itens numa árvore binária

Crie a função recursiva **`soma(A)`**, que devolve a soma de todos os itens armazenados na árvore binária **`A`**. Em seguida, faça um programa para testar o funcionamento da função.

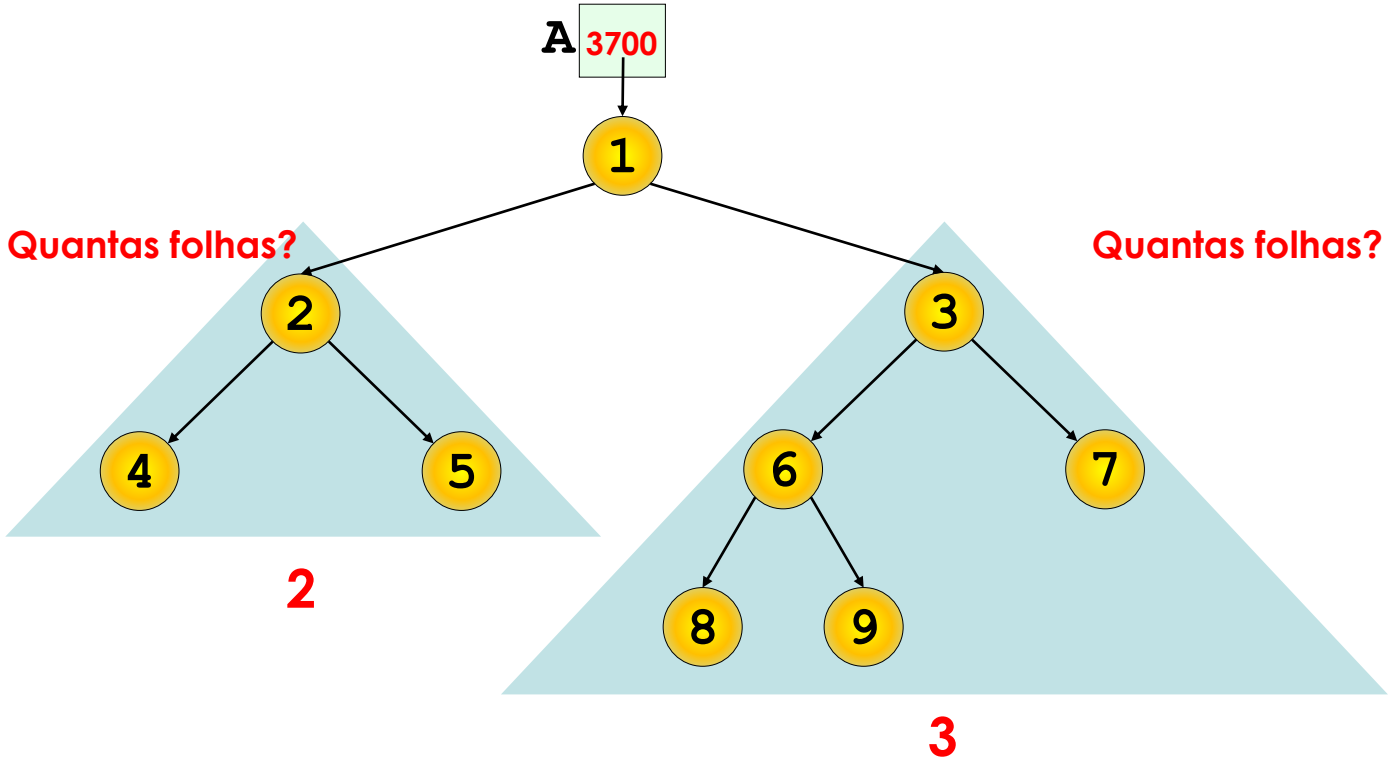


No caso geral, a soma total dos itens é igual a raiz + soma da esquerda + soma da direita!

Árvores binárias

Exercício 8. Quantidade de folhas numa árvore binária

Crie a função recursiva **`folhas(A)`**, que devolve a quantidade de folhas existentes na árvore binária **`A`**. Depois, faça um programa para testar o funcionamento da função.

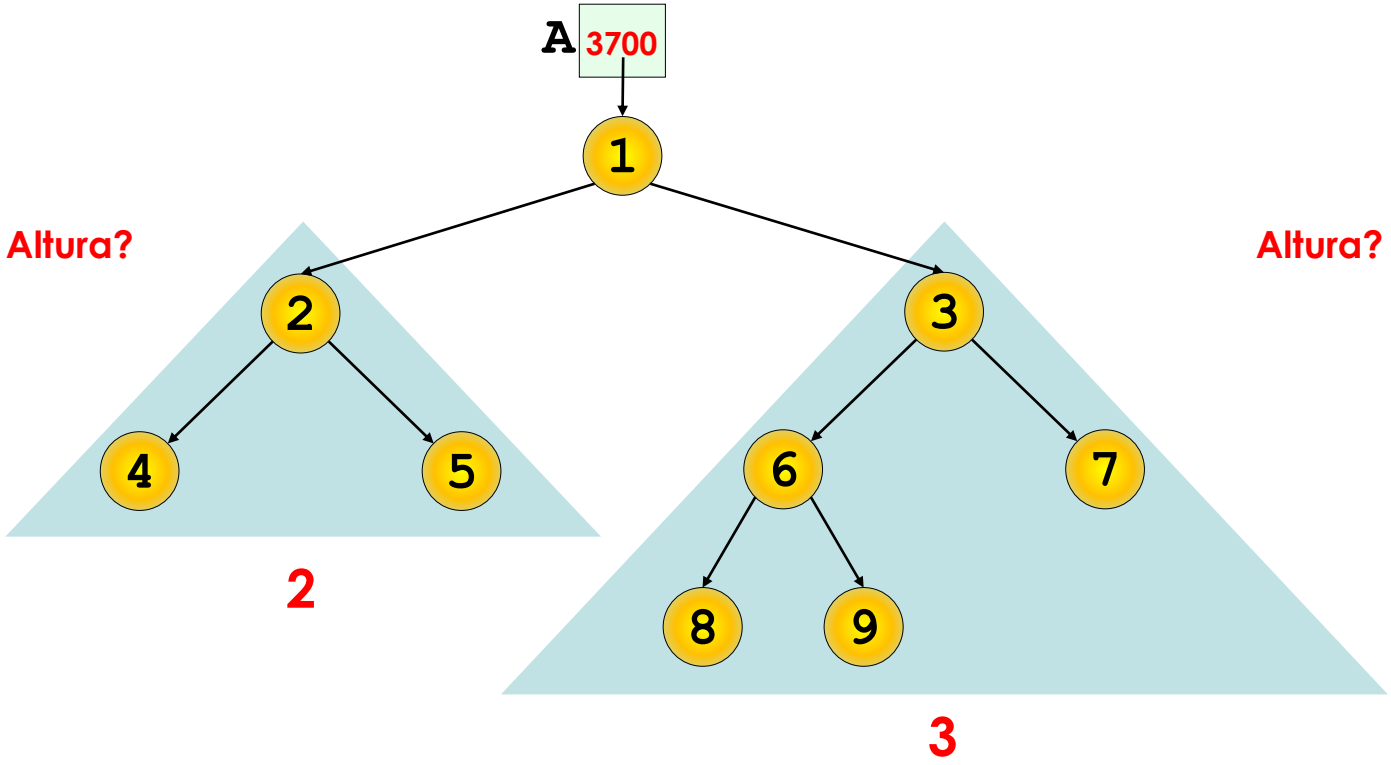


No caso geral, a quantidade de folhas é igual a folhas da esquerda + folhas da direita!

Árvores binárias

Exercício 9. Altura de uma árvore binária

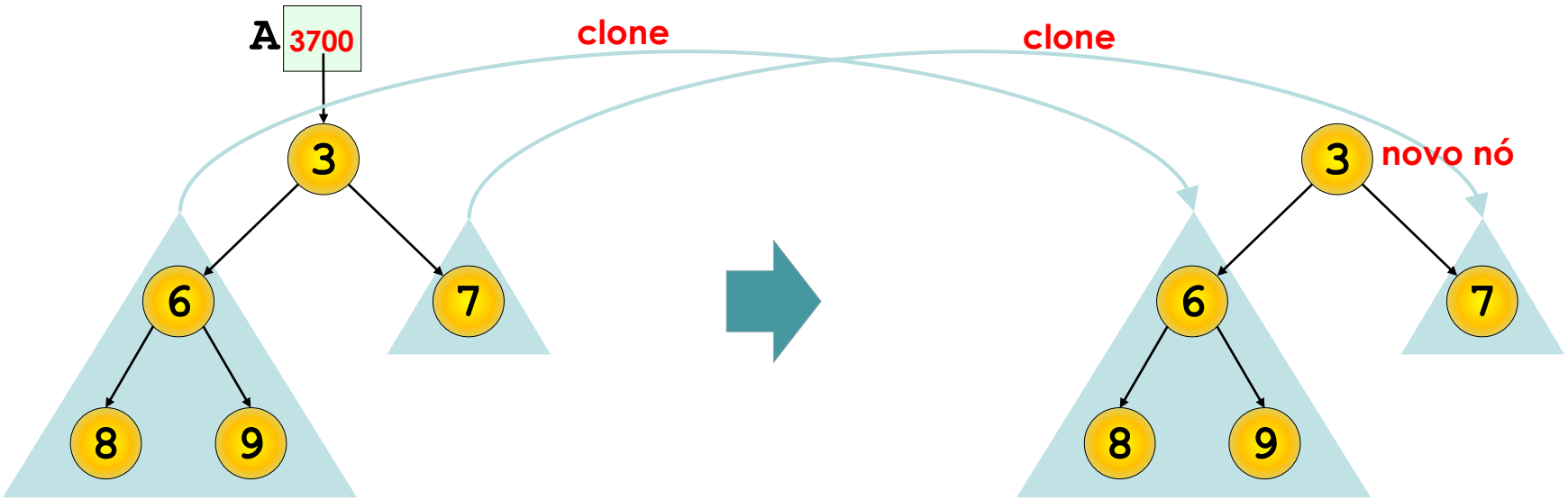
Crie a função recursiva **`altura(A)`**, que devolve a altura de uma árvore binária **`A`**, supondo que árvore vazia tem altura **`0`**. Depois, faça um programa para testar o funcionamento da função.



No caso geral, a altura da árvore é igual a 1 + máximo entre as alturas de suas subárvores!

Exercício 10. Clone de uma árvore binária

Crie a função recursiva **`clone(A)`**, que devolve um clone (cópia) da árvore binária **`A`**. Em seguida, faça um programa para testar o funcionamento da função.

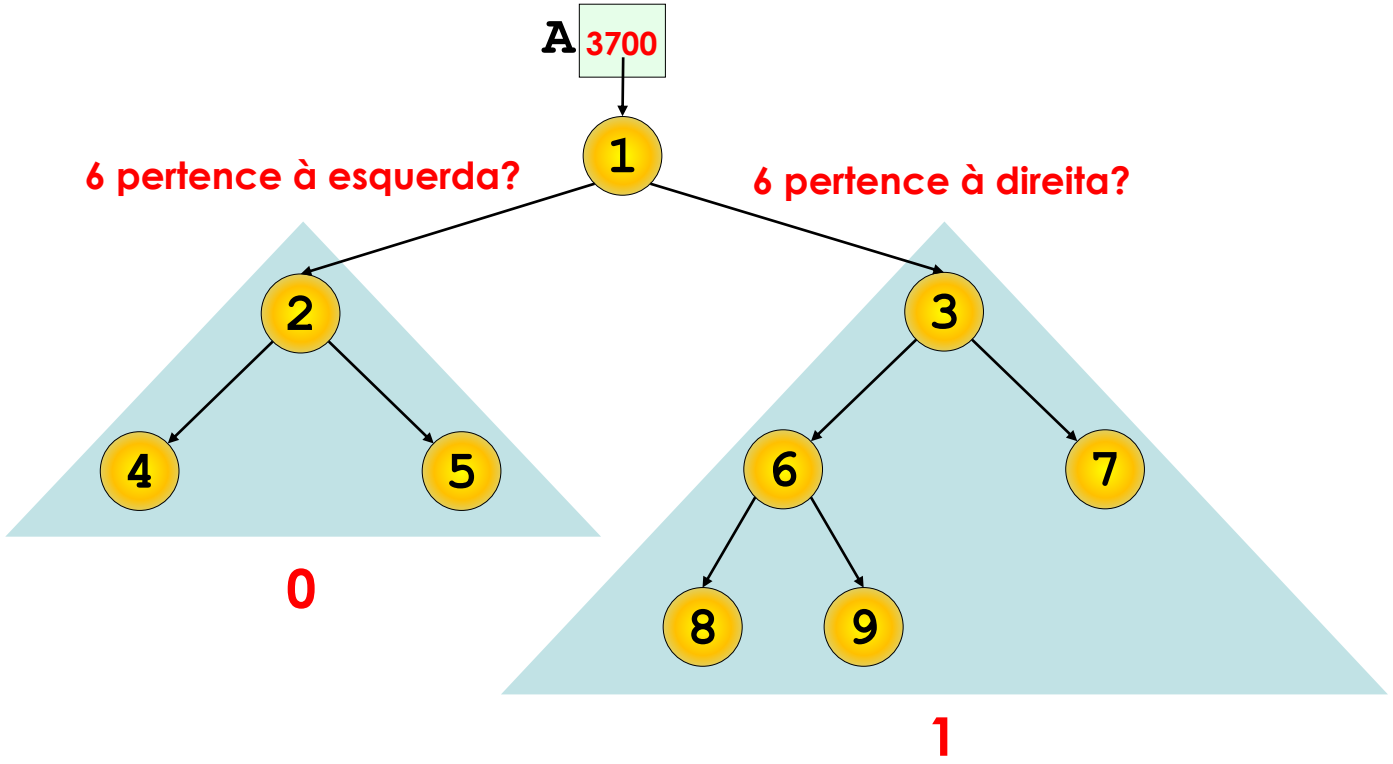


No caso geral, clone da árvore é novo nó com a raiz, cujos filhos são os clones das subárvores!

Árvores binárias

Exercício 11. Pertinência em árvore binária

Crie a função recursiva **`pertence(x,A)`**, que determina se o item **`x`** pertence à árvore binária **`A`**. Em seguida, faça um programa para testar o funcionamento da função.

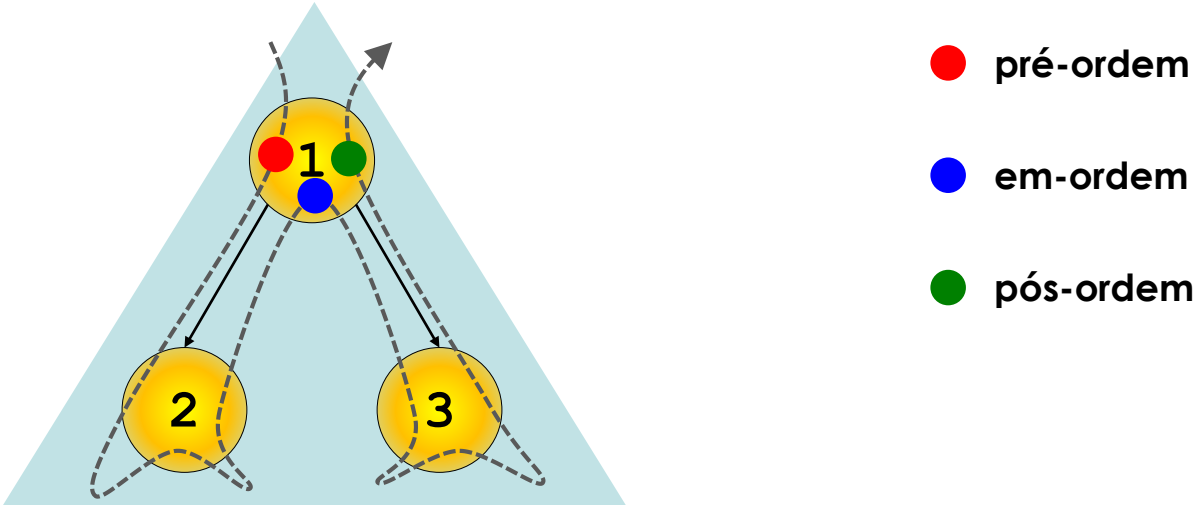


No caso geral, um item pertence a uma árvore se ele pertence à sua esquerda ou à sua direita!

Árvores binárias

Percurso em árvore binária

é uma forma sistemática de visitar e processar os nós de uma árvore.

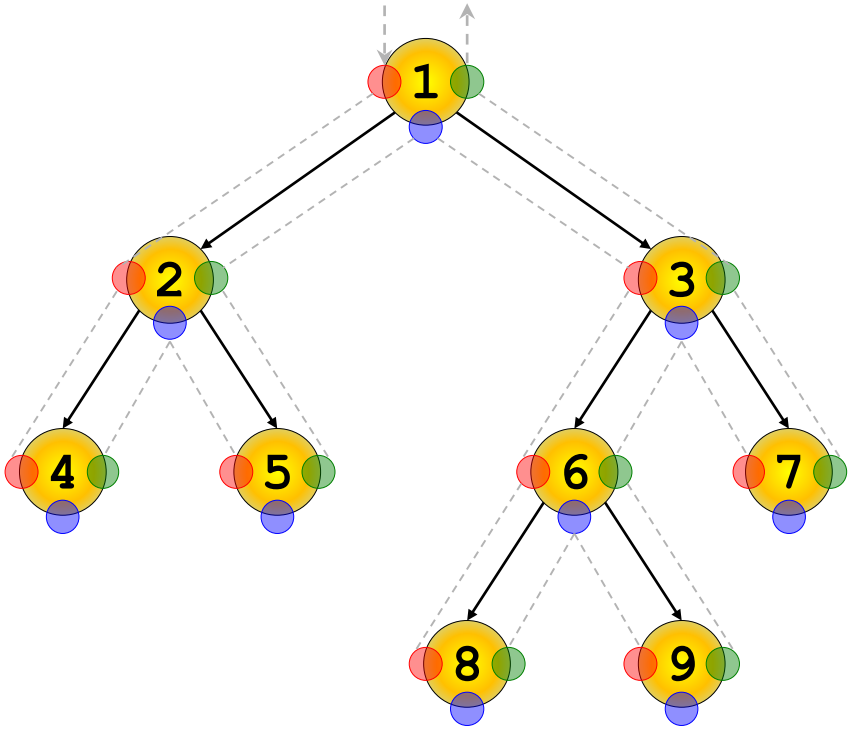


Percursos (diretos) em profundidade:

- **Pré-ordem**: processa a **raiz**, percorre a **esquerda** e, finalmente, percorre a **direita** (**R-E-D**).
- **Em-ordem**: percorre a **esquerda**, processa a **raiz** e, finalmente, percorre a **direita** (**E-R-D**).
- **Pós-ordem**: percorre a **esquerda**, percorre a **direita** e, finalmente, processa a **raiz** (**E-D-R**).

Exercício 12. Percursos

Para cada tipo de percurso, informe em que ordem os itens da árvore a seguir são processados.



Além dos percursos **diretos** (da esquerda para direita), há também três tipos de percursos **inversos**!

Árvores binárias

Exercício 13. Pré-ordem

Crie a função recursiva **`preordem(A)`**, que exibe a sequência pré-ordem da árvore binária **`A`**.

Exercício 14. Em-ordem

Crie a função recursiva **`emordem(A)`**, que exibe a sequência em-ordem da árvore binária **`A`**.

Exercício 15. Pós-ordem

Crie a função recursiva **`posordem(A)`**, que exibe a sequência pós-ordem da árvore binária **`A`**.

Exercício 16. Em-ordem inversa

Crie a função recursiva **`eoi(A)`**, que exibe a sequência em-ordem inversa da árvore binária **`A`**.

Exercício 17. Poda de árvore binária

Crie a função recursiva **`poda(&A)`**, que remove todas, e só, as folhas de uma árvore binária **`A`**.

Exercício 18. Destruição de árvore binária

Crie a função recursiva **`destroi(&A)`**, que destrói uma árvore binária **`A`**.

Árvores binárias

Exercício 19. Contagem

Crie a função recursiva **`conta(x,A)`**, que informa quantas vezes o item **`x`** ocorre na árvore **`A`**.

Exercício 20. Igualdade

Crie a função recursiva **`iguais(A,B)`**, que informa se as árvores binárias **`A`** e **`B`** são iguais.

Exercício 21. Espelho I

Crie a função recursiva **`espelho(A)`**, que devolve uma cópia espelhada da árvore binária **`A`**.

Exercício 22. Espelho II

Crie a função recursiva **`espelho(A,B)`**, que informa se a árvore **`A`** é espelho da árvore **`B`**.

Exercício 23. Balanceada II

Crie a função recursiva **`balanceada(v,p,u)`**, que devolve uma árvore binária balanceada com os itens do vetor **`v`**, cujo primeiro item está na posição **`u`** e cujo último item está na posição **`u`**.

Fim