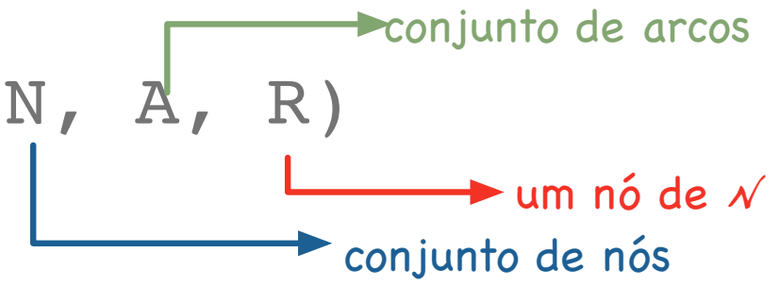
Árvores

Definição de árvores: pesquisa e percursos
Árvores Binarias, percursos em ordem, árvores de expressões
Árvores Binárias de Pesquisa. Definição do TAD e implementação.

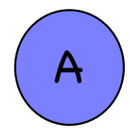
29-Abril-21

Árvores

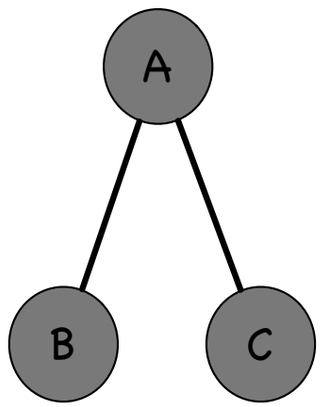
- Definição: Grafo (N, A, R)



({A}, {}, A)

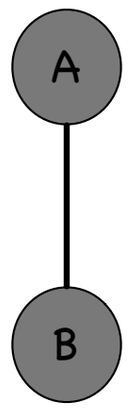


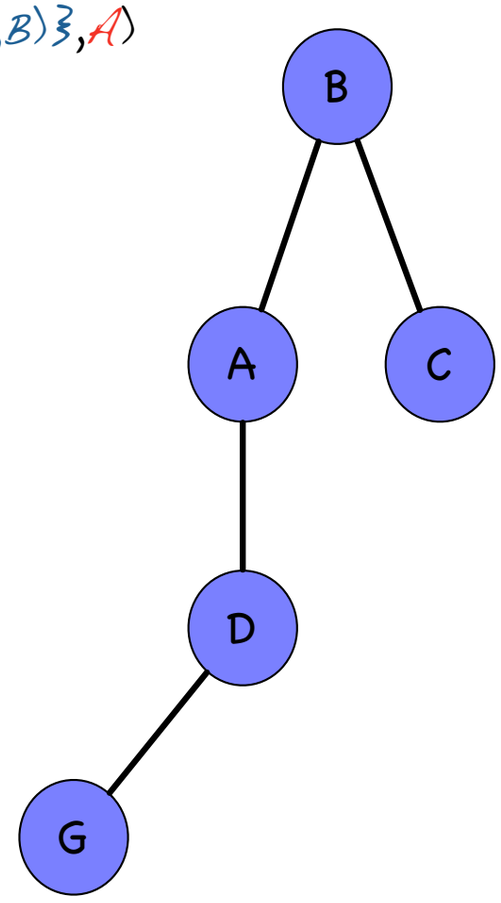
  - Nós especiais
    - raiz
    - folhas
  - Arcos
    - relações de parentalidade entre os nós
      - pai, filho, descendente, antepassado.



({A,B,C}, {(A,B),(A,C)}, A)

({A,B}, {(A,B)}, A)





({A,B,C,D,G}, {(B,A),(B,C),(A,D),(D,G)}, B)

Árvores

- Às vezes o acesso linear das listas torna proibitivo(grandes quantidades de informação) o seu uso

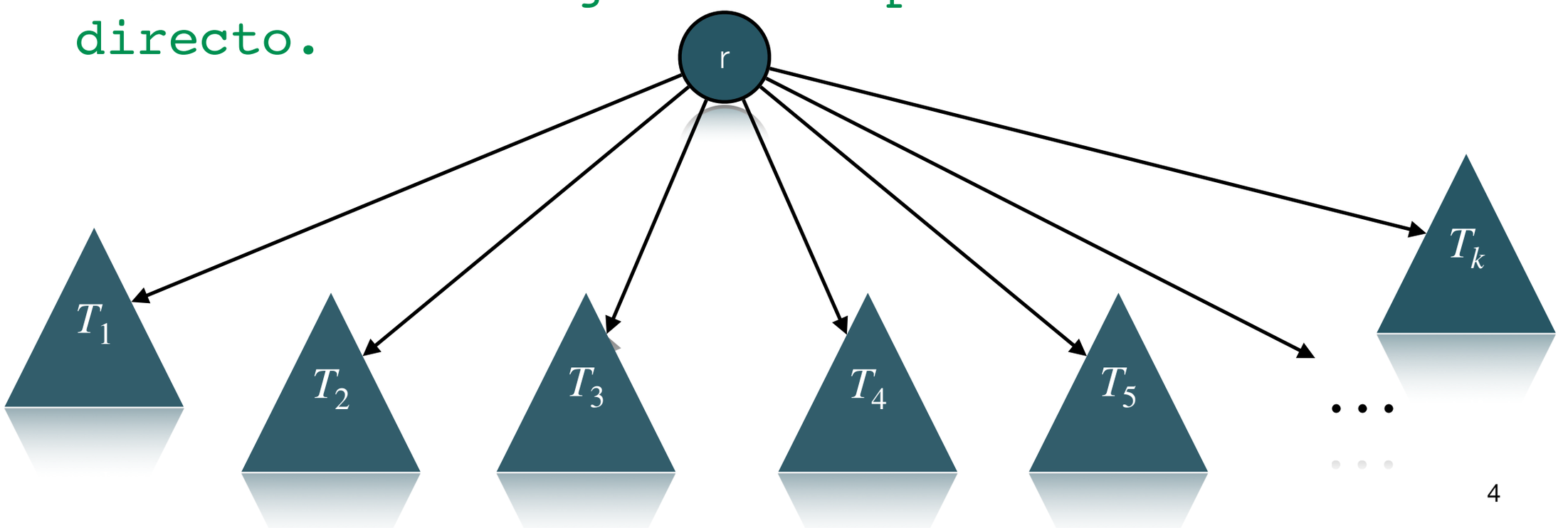
- Existem outras estruturas cujo acesso possa ser melhorado

- Sim, há estruturas que podemos aspirar a uma complexidade da ordem de $log(n)$

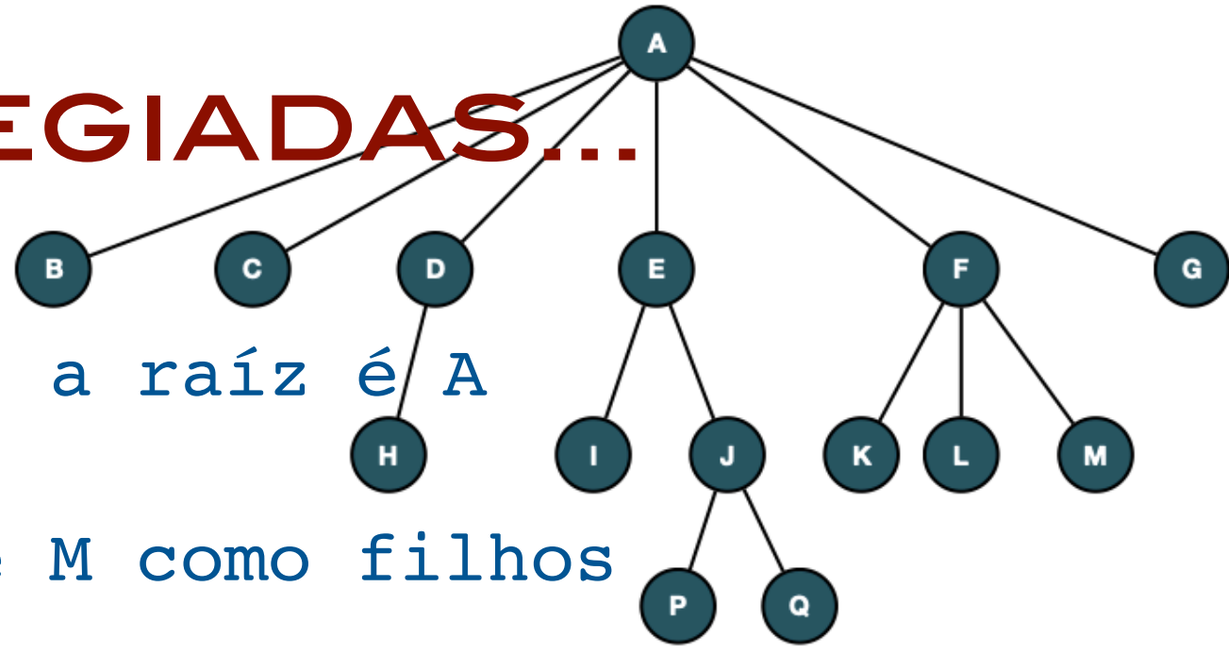
- Essa estrutura é conhecida por “Árvores”

Árvores

- Definição recursiva:
  - Uma colecção de nós:
    - Vazia, é uma árvore
    - Um nó distinguido: r, e k sub-árvores disjuntas, (T1,...,Tk). As raízes destas sub-árvores estão ligadas a r por um arco directo.



Relações privilegiadas...



- Para a árvore apresentada, a raíz é A
- F tem A como pai A e K,L e M como filhos
- O número de filhos dum nó pode ser qq, inclusive 0
- Os nós sem filhos são designados por folhas
  - na árvores da figura as folhas são B;C;H;I;P;Q;K;L;M;G
  - Nós com o mesmo pai são irmãos
  - AS relações de avô e netos, são definidas de igual modo

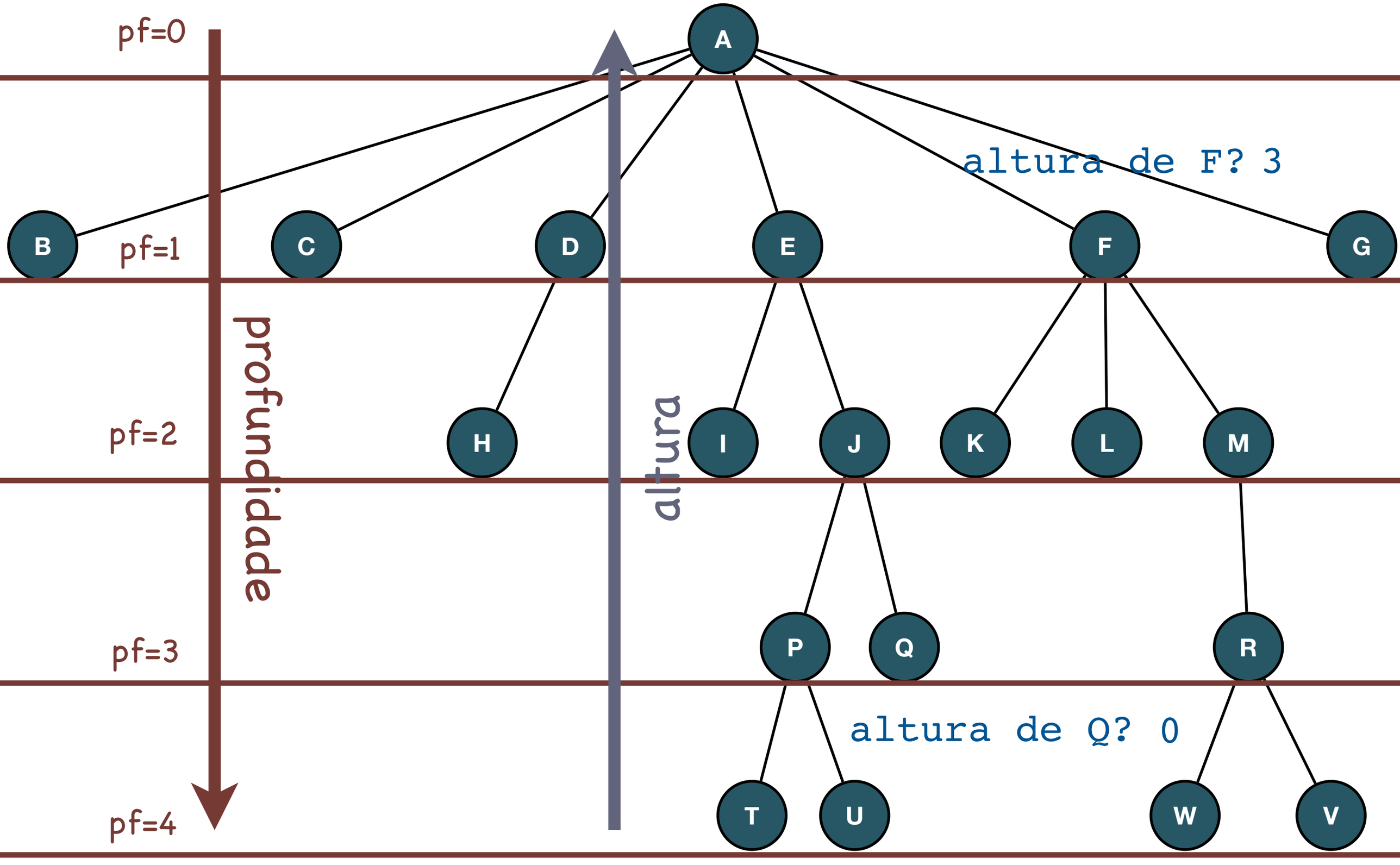
Árvores

- Caminho de n_1 para n_k é definido como a sequência de nós $n_1, n_2, \ldots, n_k$, tais que
  - n_i é pai de n_{i+1}, $1 \leq i < k$
  - comprimento do caminho é igual ao número de arcos do caminho $(k-1)$
  - existe um caminho de comprimento 0 dum nó para si próprio
- Numa árvore existe um e um só caminho da raiz até qualquer dos nós

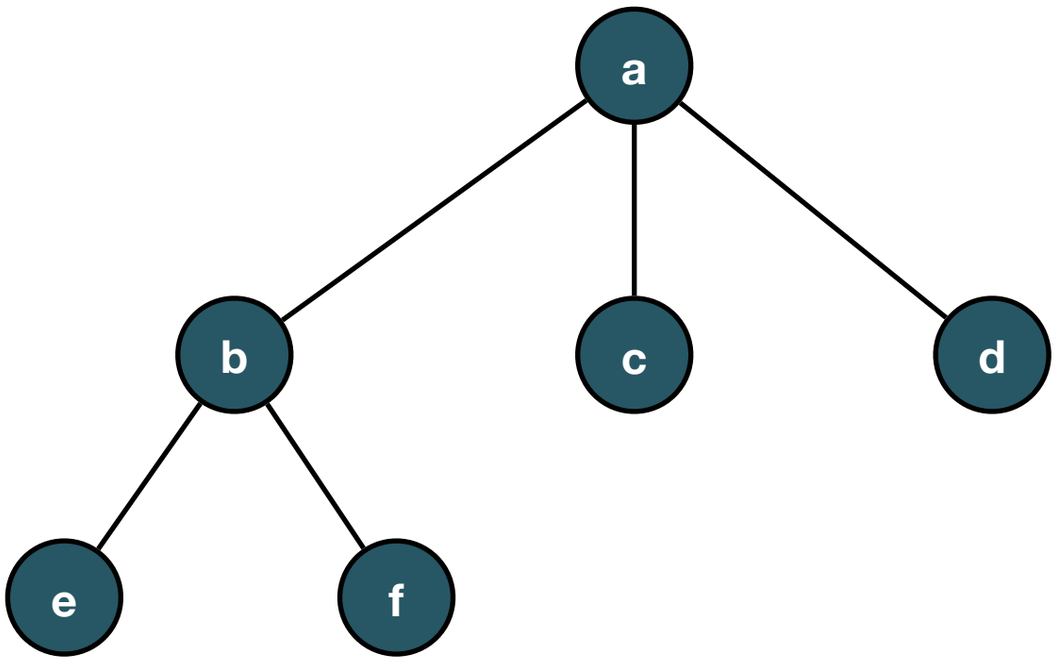
Árvores

- Profundidade dum nó: comprimento do caminho da raiz até ao nó logo
  - a profundidade da raíz é 0
- Altura dum nó: máximo do comprimento dos caminhos do nó até a uma folha, logo
  - altura de qualquer folha é 0
- Altura duma árvore: altura da sua raíz

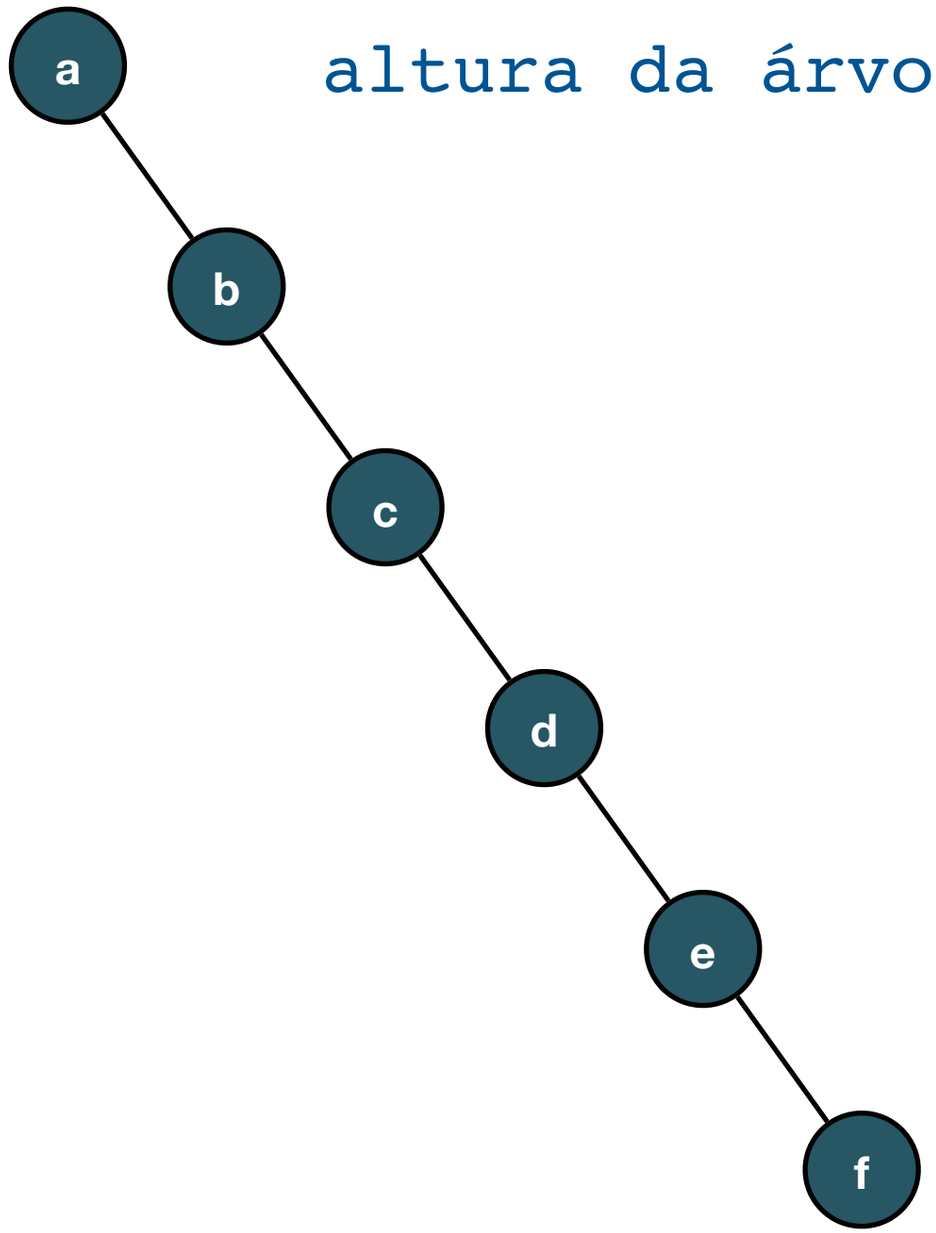
Árvores


altura da árvore ? 4
pf=0
pf=1
pf=2
pf=3
pf=4
profundidade
altura
altura de F? 3
altura de Q? 0
A
B
C
D
E
F
G
H
I
J
K
L
M
P
Q
R
T
U
W
V

Mais exemplos:



altura da árvore? 2

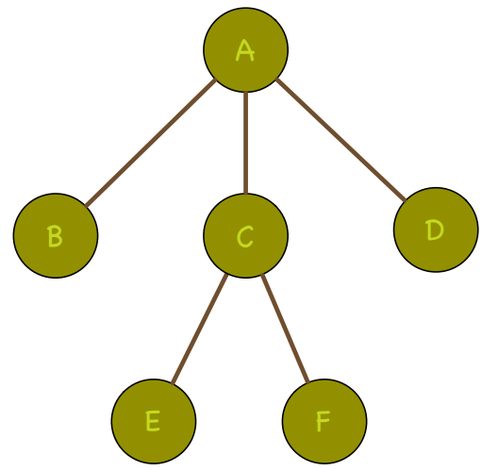


altura da árvore? 5

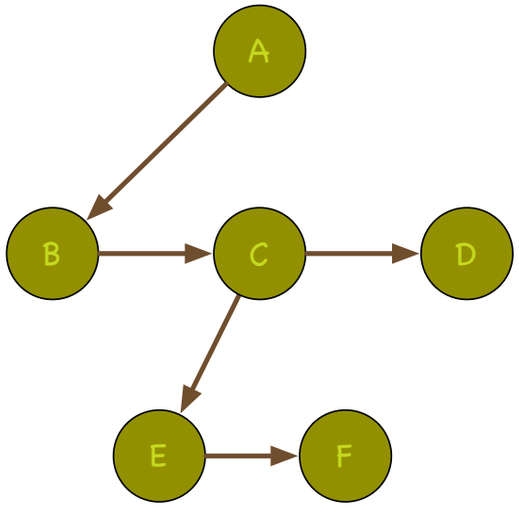
Representação de árvores

- Usando listas/nós:

```
Typedef struct tree_node *tree_ptr;

struct tree_node{
    element_type element;
    tree_ptr first_child;
    tree_ptr next_sibling;

};
```



O que representam os apontadores na “vertical”?



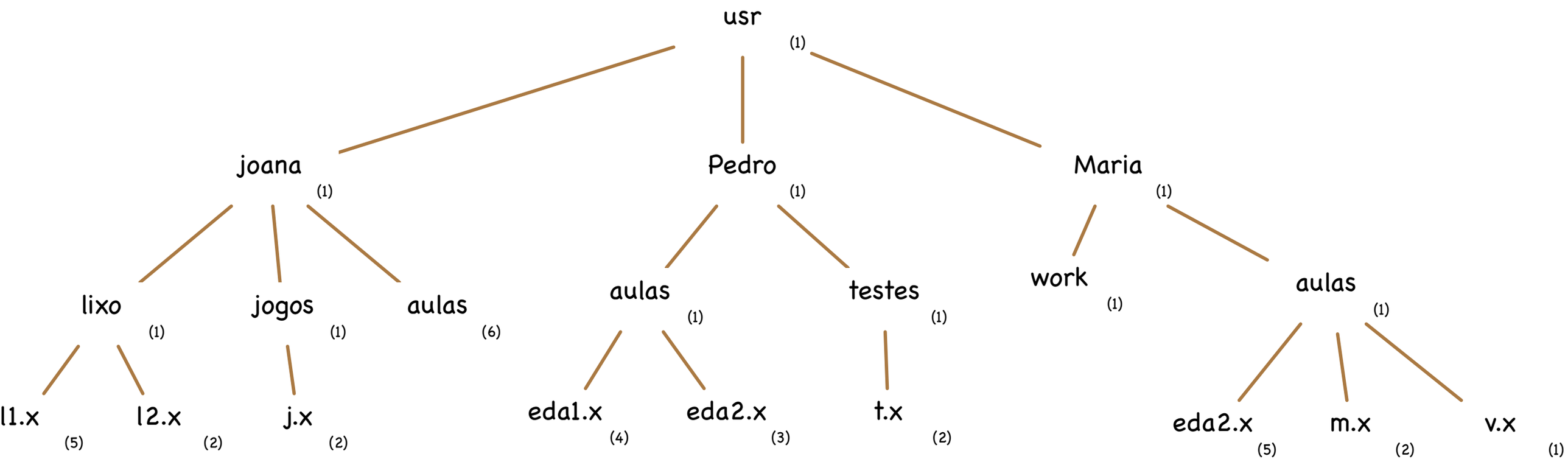
e na “horizontal”?

Operações sobre árvores

- Mais usuais:
  - travessia
  - procura
- Formas (recursivas)
  - Dependem da ordem pela qual se mostram nós e filhos
    - pré-ordem(nó;filhos)
    - post-ordem(filhos;nó)

Exemplo: Árvore de Directorias

- Árvore de diretorias:



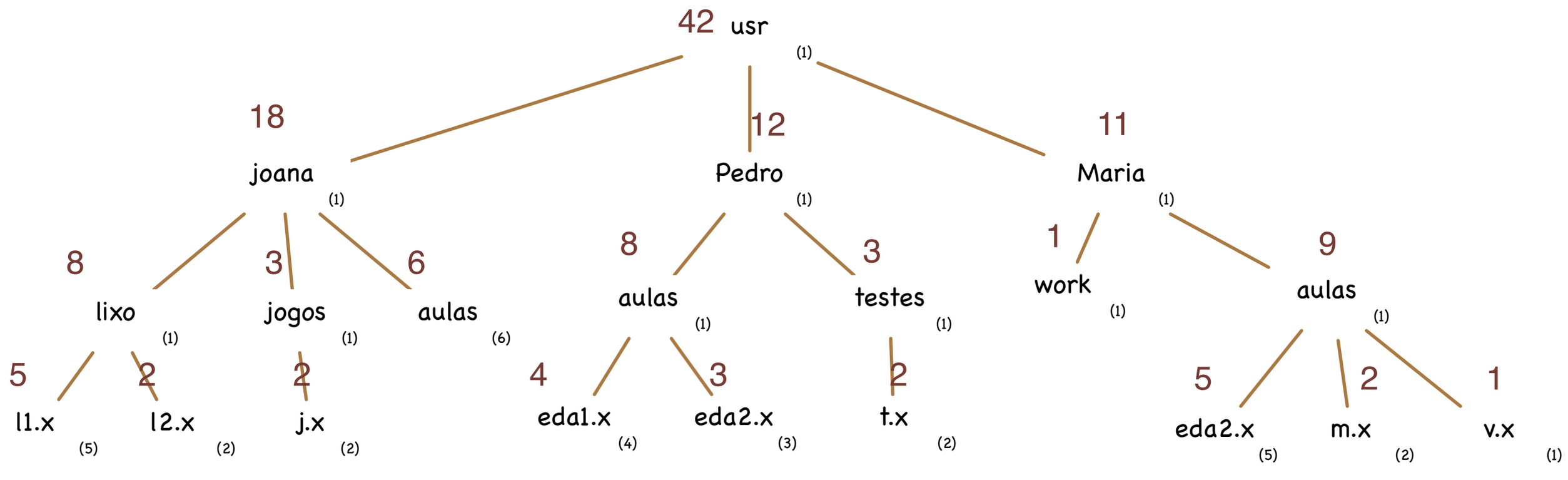
Listagem recursiva

```
void list_dir ( Directory_or_file D, unsigned int depth ){
    if( D is a legitimate entry){
        print_name(depth, D);
        if (D is a directory)
            for each child, c, of D
                list_dir(c, depth+1);
    }

}

void list_directory (Directory_or_file D ){
    list_dir ( D, 0 );
}
```

usr
- joana
  - lixo
    - l1.x
    - l2.x
  - jogos
    - j1.x
  - aulas
- pedro
  - aulas
    - ed1.x
    - ed2.x
  - testes
    - t.x
- Maria
  - work
  - aulas
    - ed2.x
    - m1.x
    - v.x

Determinar o tamanho

```
unsigned int size_directory(Directory_or_file D){
    unsigned int total_size;
    total_size=0;

    if( D is a legitimate entry){
        total_size=file_size(D);
        if (D is a directory)
            for each child, c, of D
                total_size+=size_directory(c));
    }
    return total_size;
}
```



Árvore Binária

- Uma árvore binária é uma árvore, em que:
  - Cada nó tem máximo dois filhos (FE; FD)
- Nova ordem de travessia
  - em-ordem (FE; N; FD)
- Implementação (árvore binária de inteiros..)

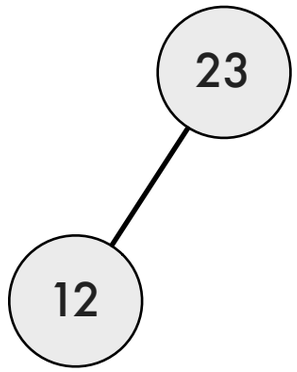
```
struct BNode{
    ElementType Data;
    ABin  Left;
    ABin  Right;
};
```

Exemplo interface

```
….
struct BNode;
typedef struct BNode *Position;
typedef struct BNode *ABin;

ABin MakeEmpty( ABin T );
ABin CreateABin( ElementType d, ABin left,ABin rig );
ElementType Retrieve( Position P );
ABin Left(ABin T);
ABin Right(ABin T);
void PrintOrder(ABin T);
void PrintPreOrder(ABin T);
void PrintPosOrder(ABin T);
….
```

Árvores Binárias

- Criar a árvore da figura:




```
ABin b1=CreateABin(12, NULL, NULL);
ABin b2=CreateABin(23, b1, NULL);
```

- Ver ?

- Ou printEmOrdem(),…

```
PrintOrder(b2)
```

12
23

Exemplo:Árvore de Expressões

- A construção da árvore de expressões correspondente a uma expressão em postfix, faz-se usando o seguinte algoritmo:

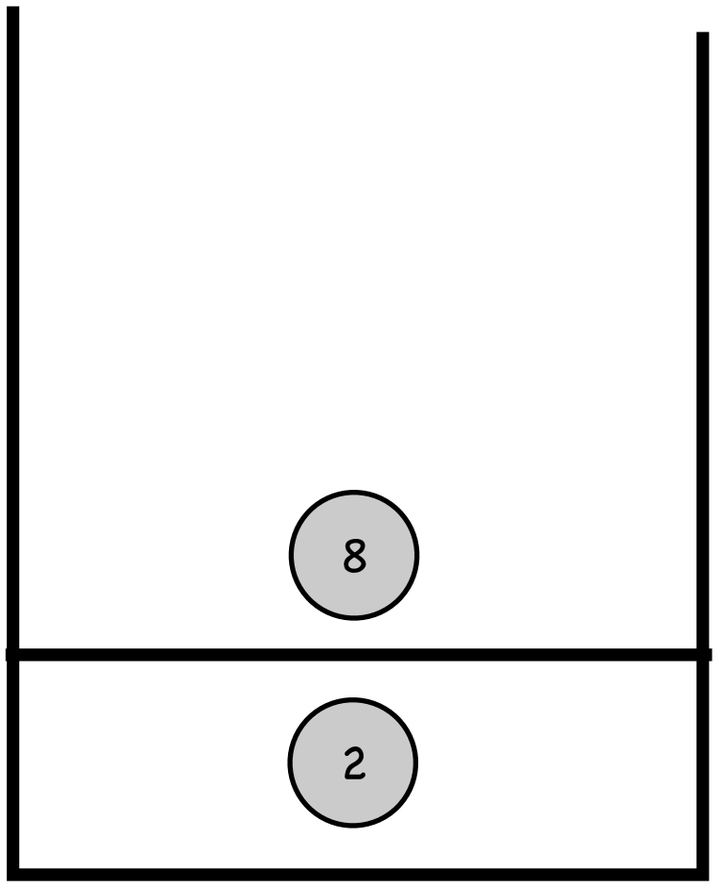
  - Usando uma expressão postfix: lê-se um token de cada vez

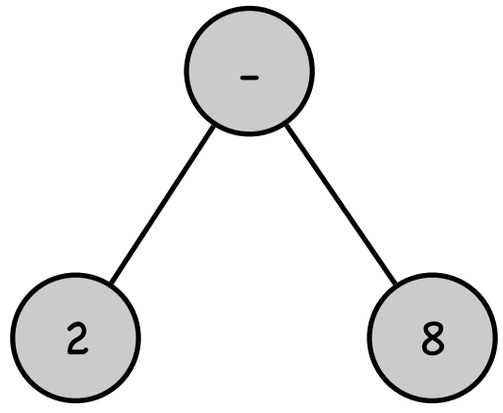
  - se operando:criar uma árvore com esse nó (único) e por numa stack a árvore;

  - se operador:fazer pop de duas árvores T1 e T2 da stack, e construir uma nova árvore cuja raíz é o operador, o filho esquerdo T1 e o filho direito é T2; Fazer push da nova árvore na stack;

Expressão em postfix:

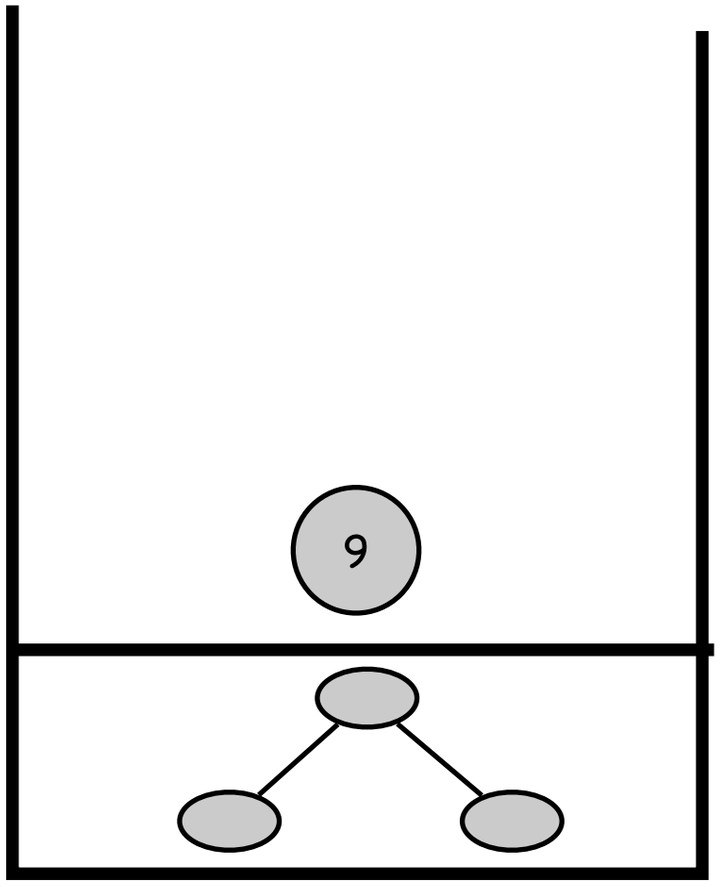
2	8	-	9	/	7	*	67	7	5	*	+	8	7	*	/	-

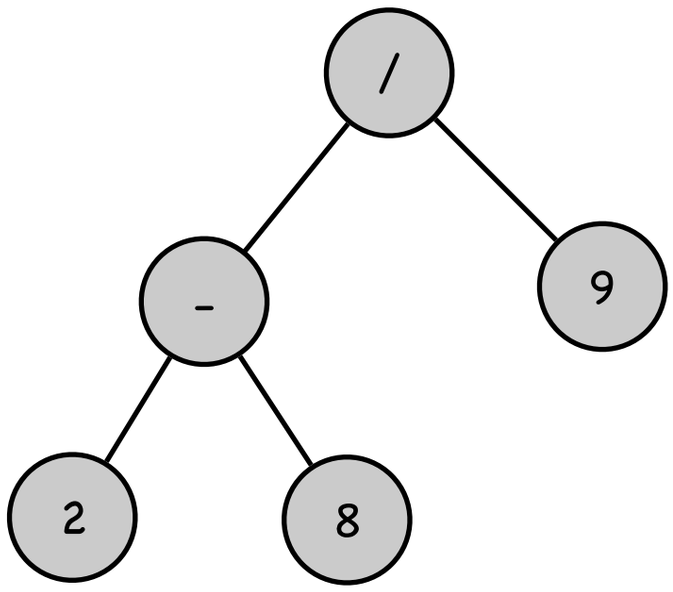




Expressão em postfix:

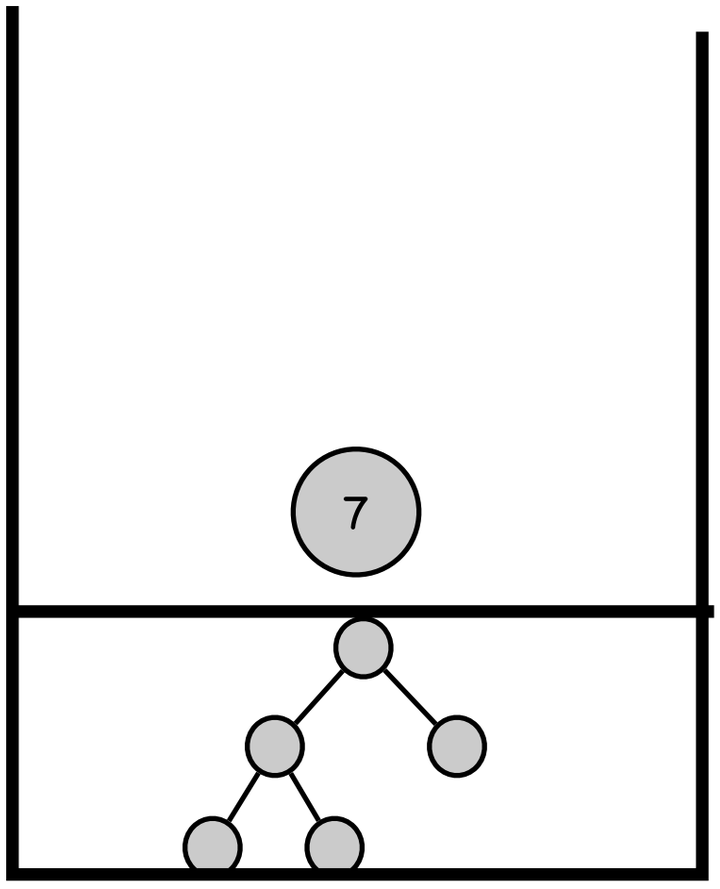
2	8	-	9	/	7	*	67	7	5	*	+	8	7	*	/	-

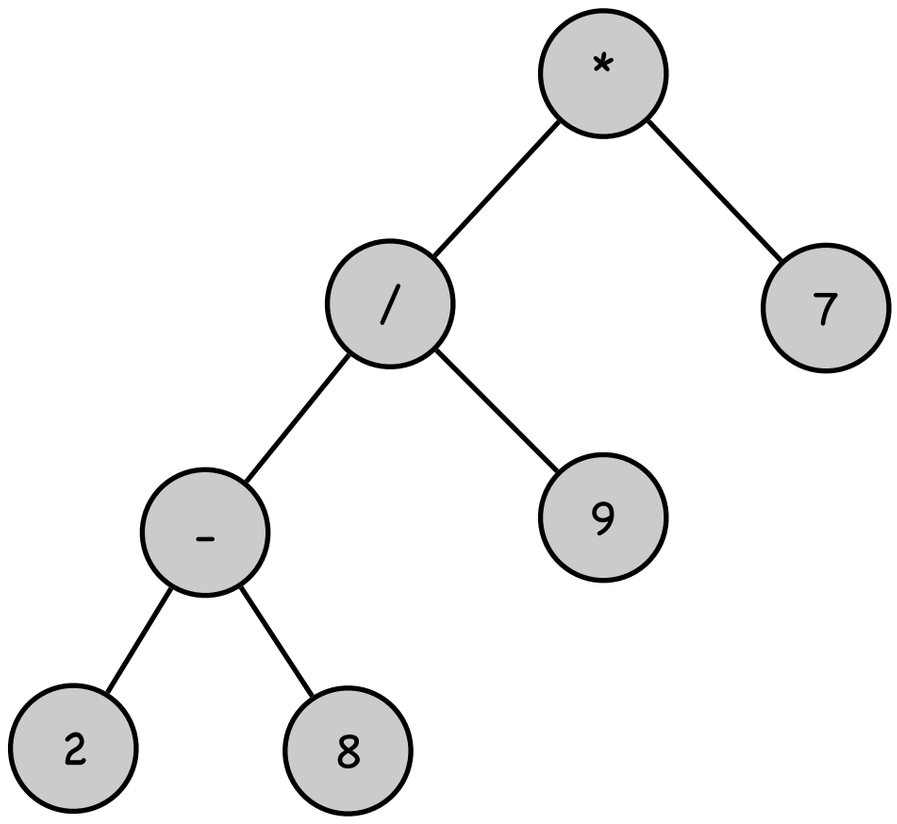




Expressão em postfix:

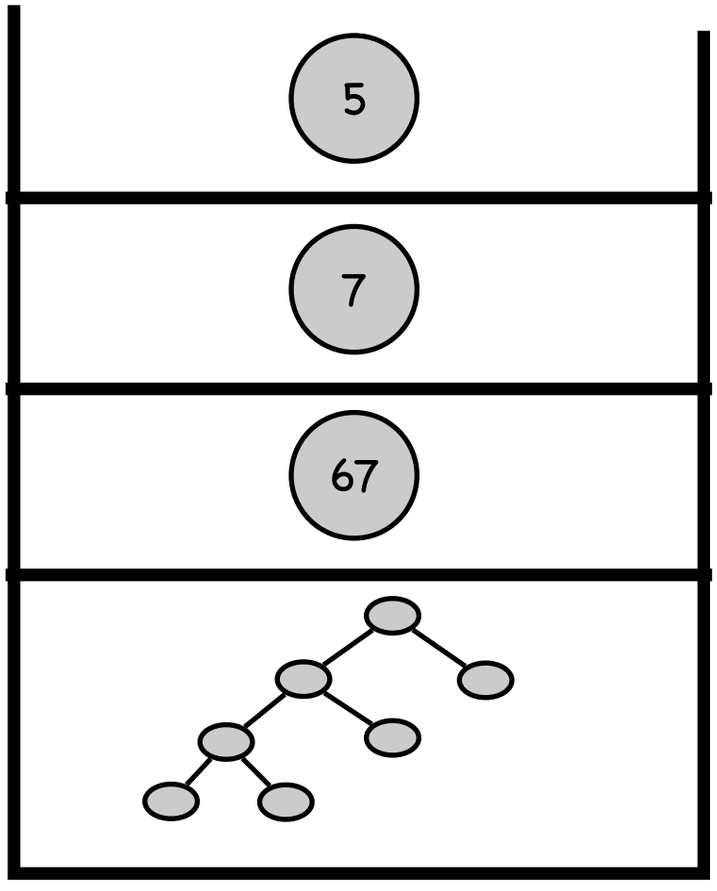
2	8	-	9	/	7	*	67	7	5	*	+	8	7	*	/	-

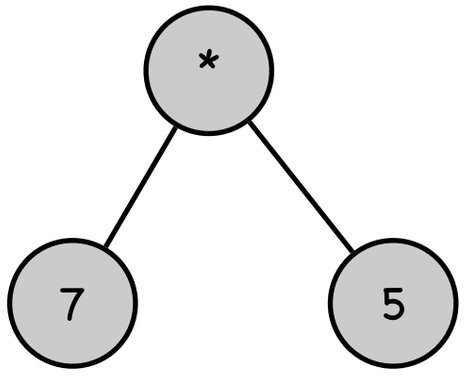




Expressão em postfix:

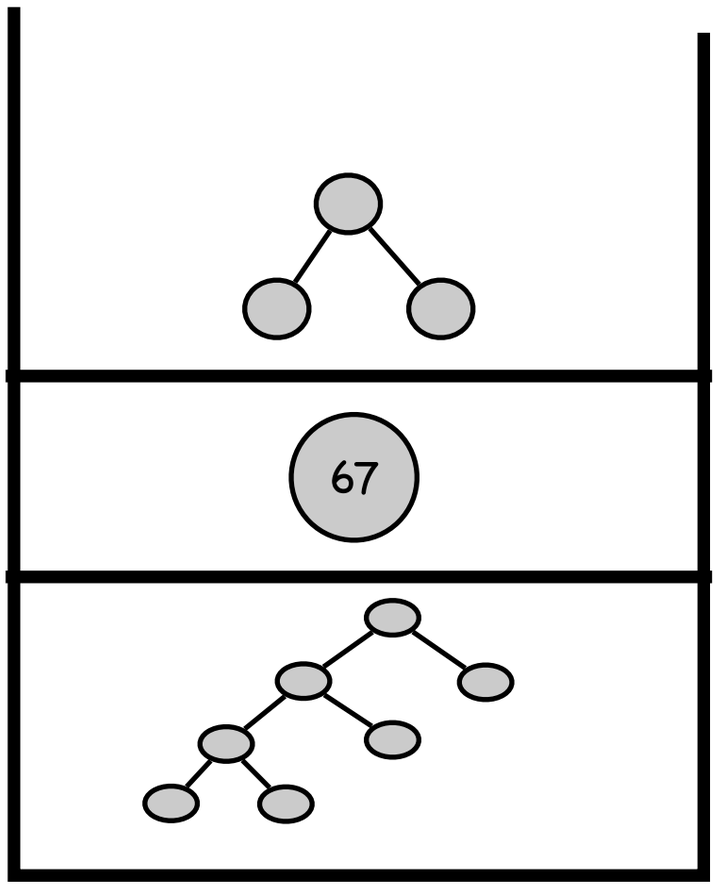
2	8	-	9	/	7	*	67	7	5	*	+	8	7	*	/	-

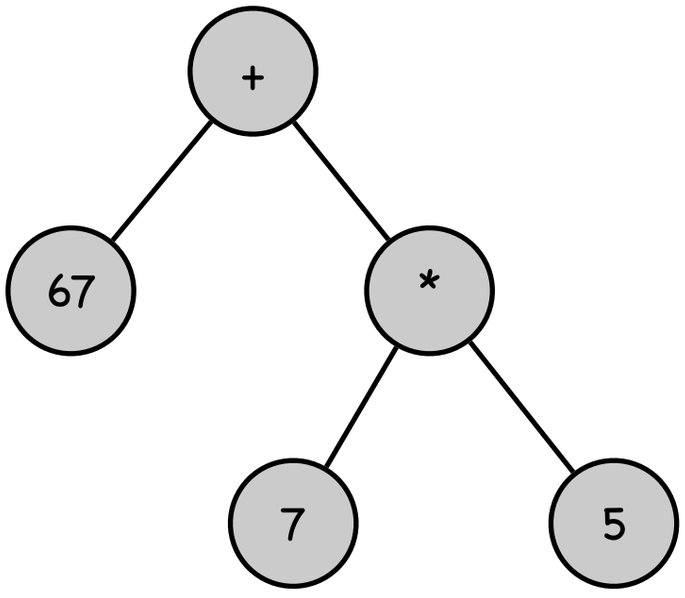




Expressão em postfix:

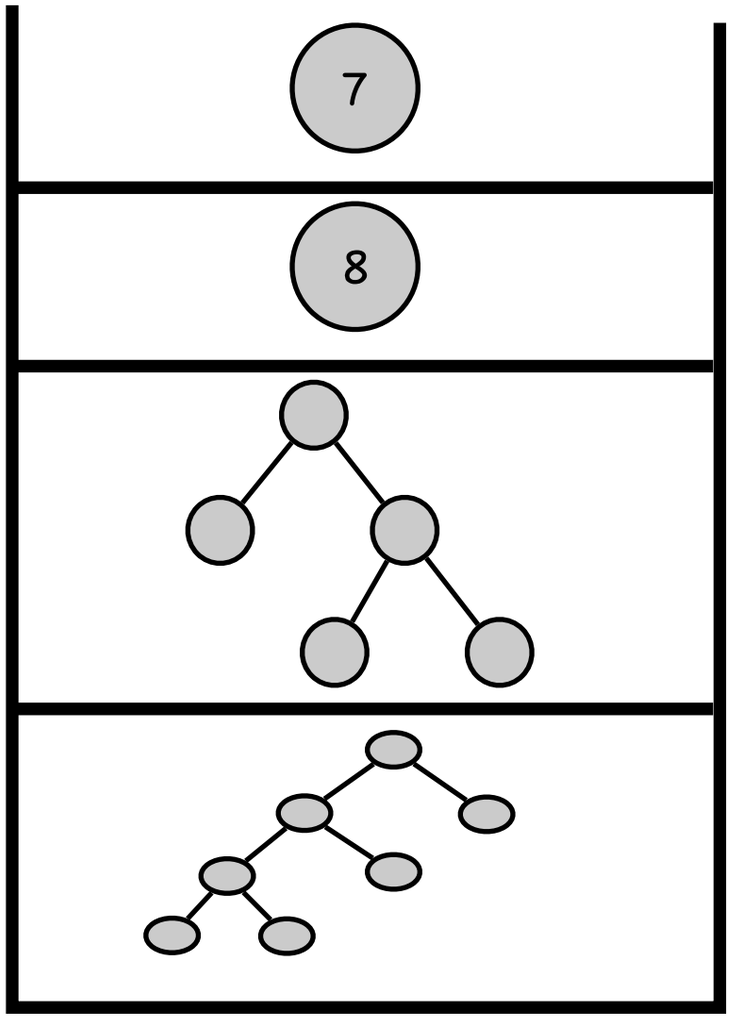
2	8	-	9	/	7	*	67	7	5	*	+	8	7	*	/	-

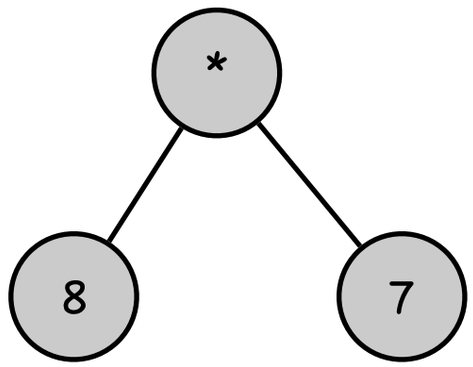




Expressão em postfix:

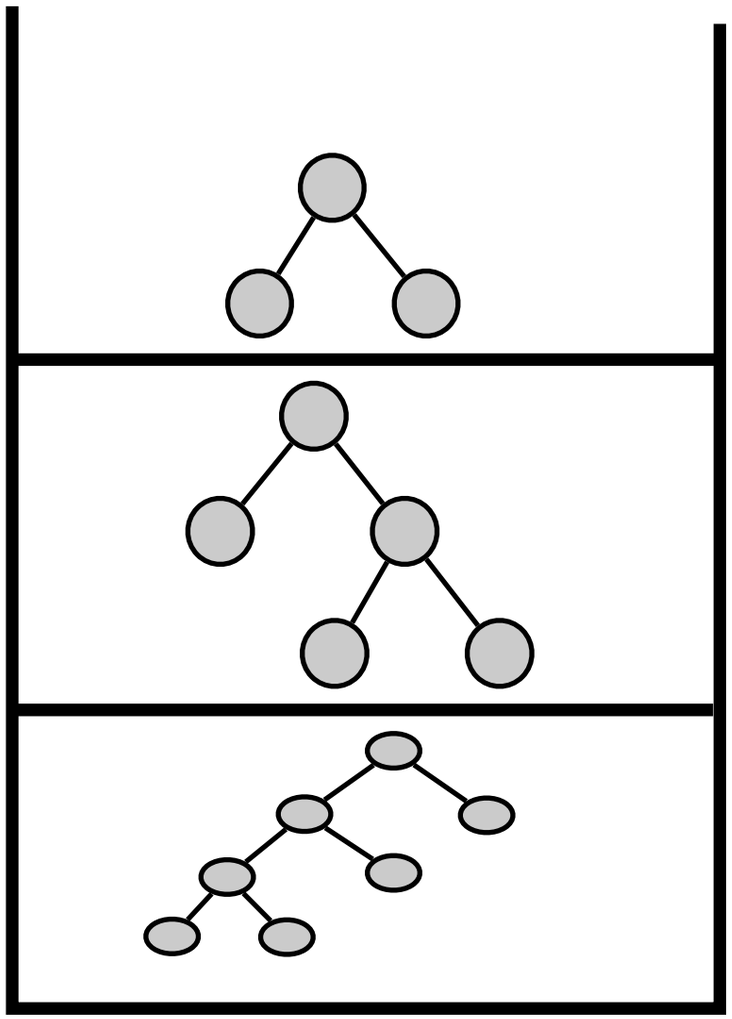
2	8	-	9	/	7	*	67	7	5	*	+	8	7	*	/	-

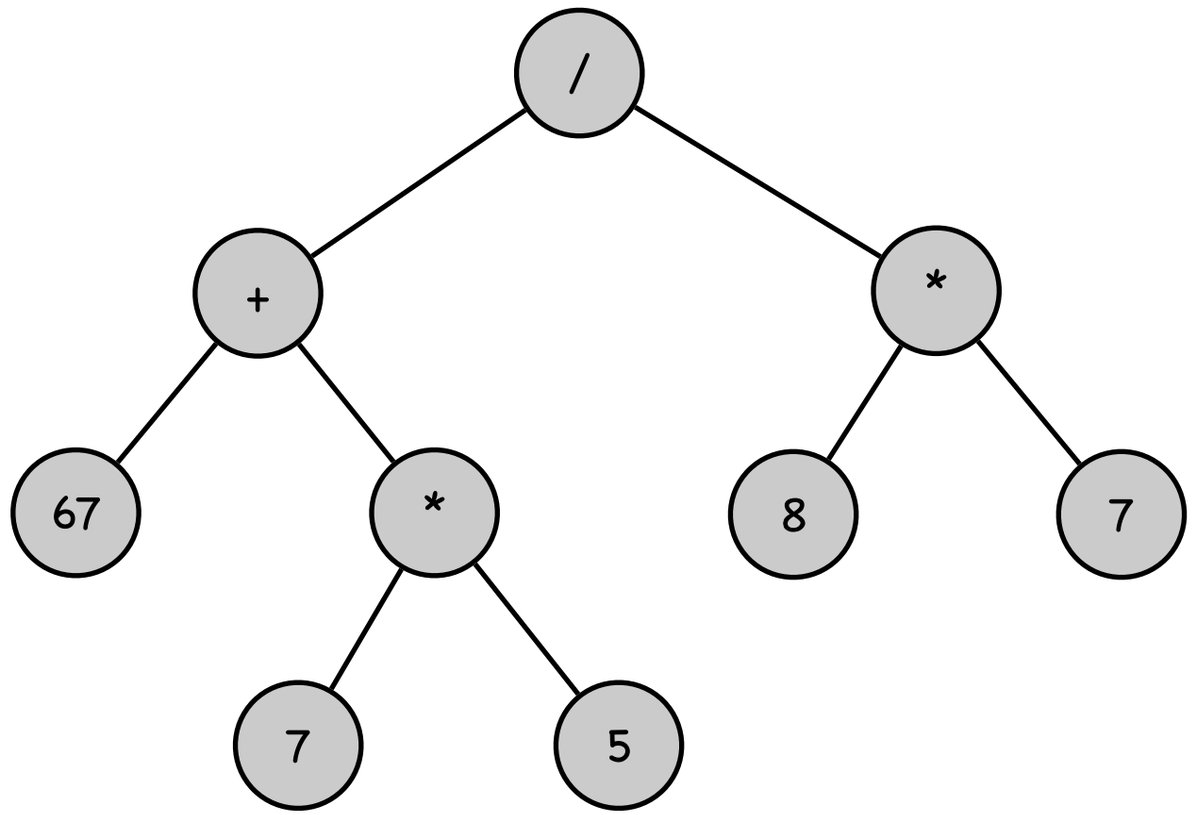




Expressão em postfix:

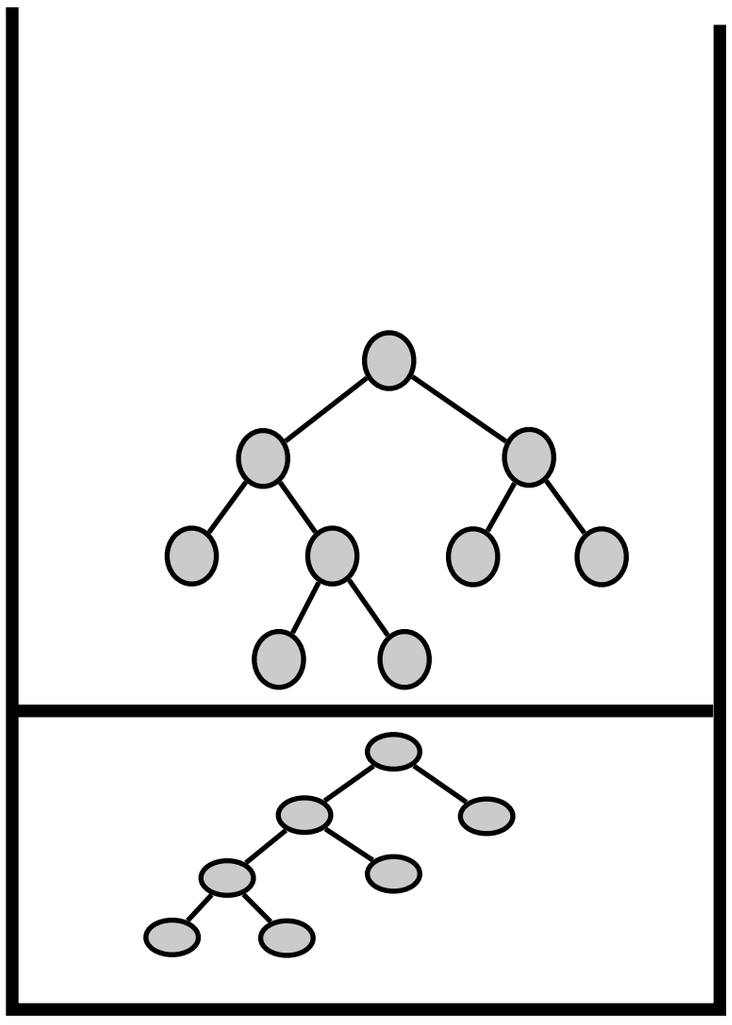
2	8	-	9	/	7	*	67	7	5	*	+	8	7	*	/	-

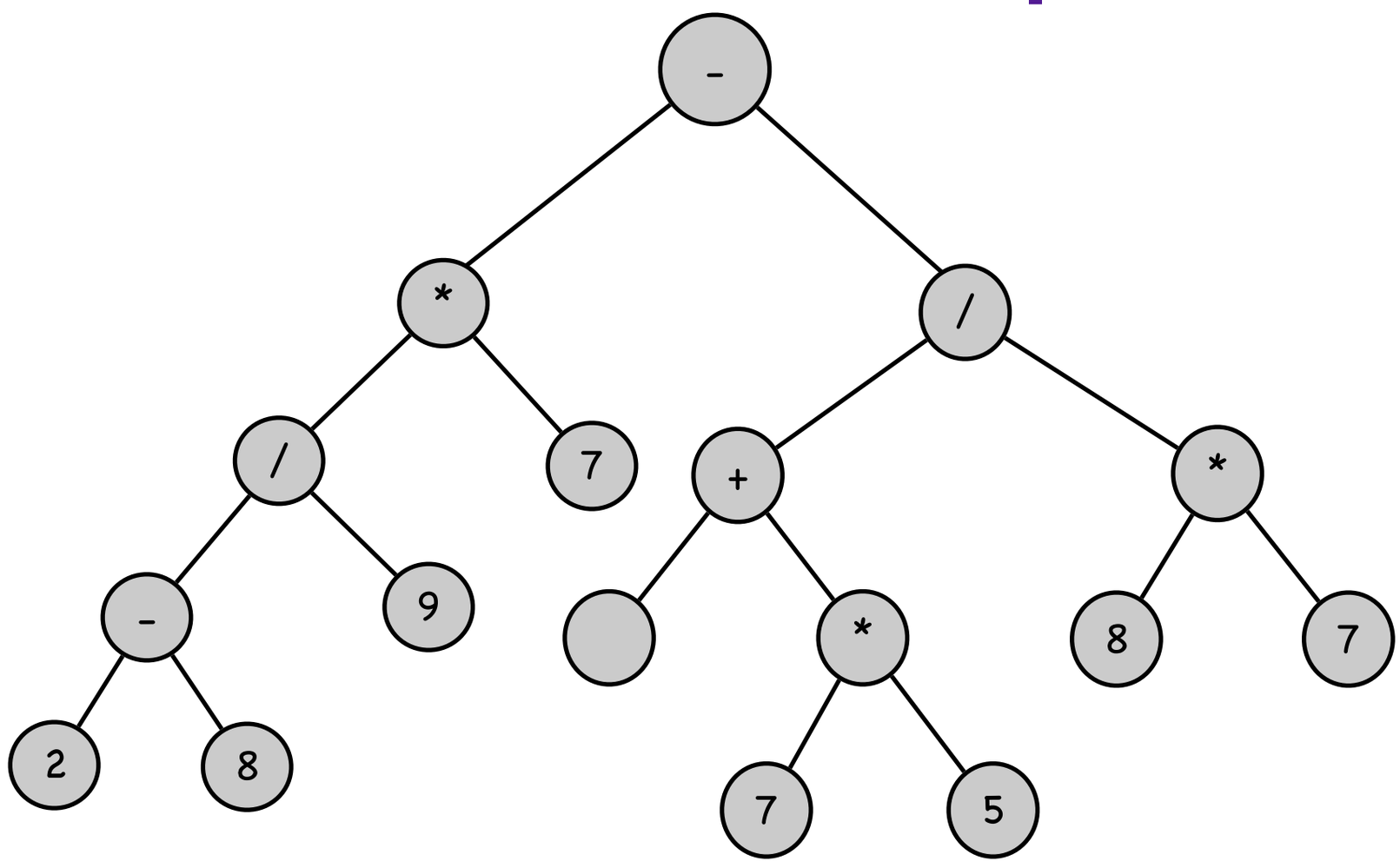




Expressão em postfix:

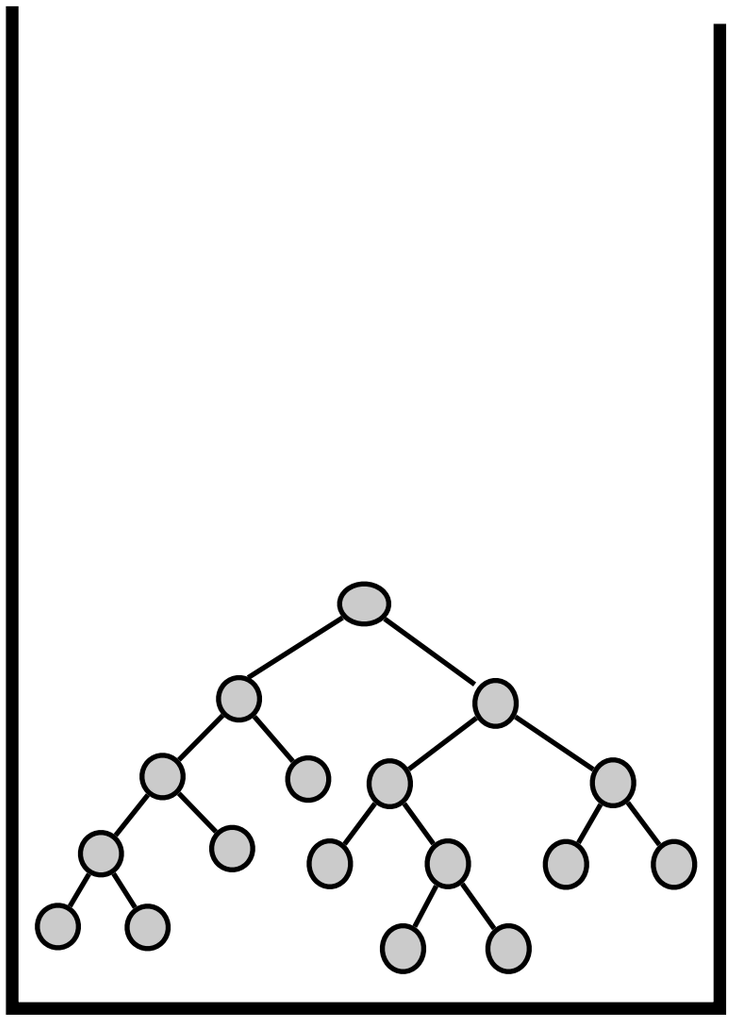
2	8	-	9	/	7	*	67	7	5	*	+	8	7	*	/	-





Expressão em postfix:

2	8	-	9	/	7	*	67	7	5	*	+	8	7	*	/	-



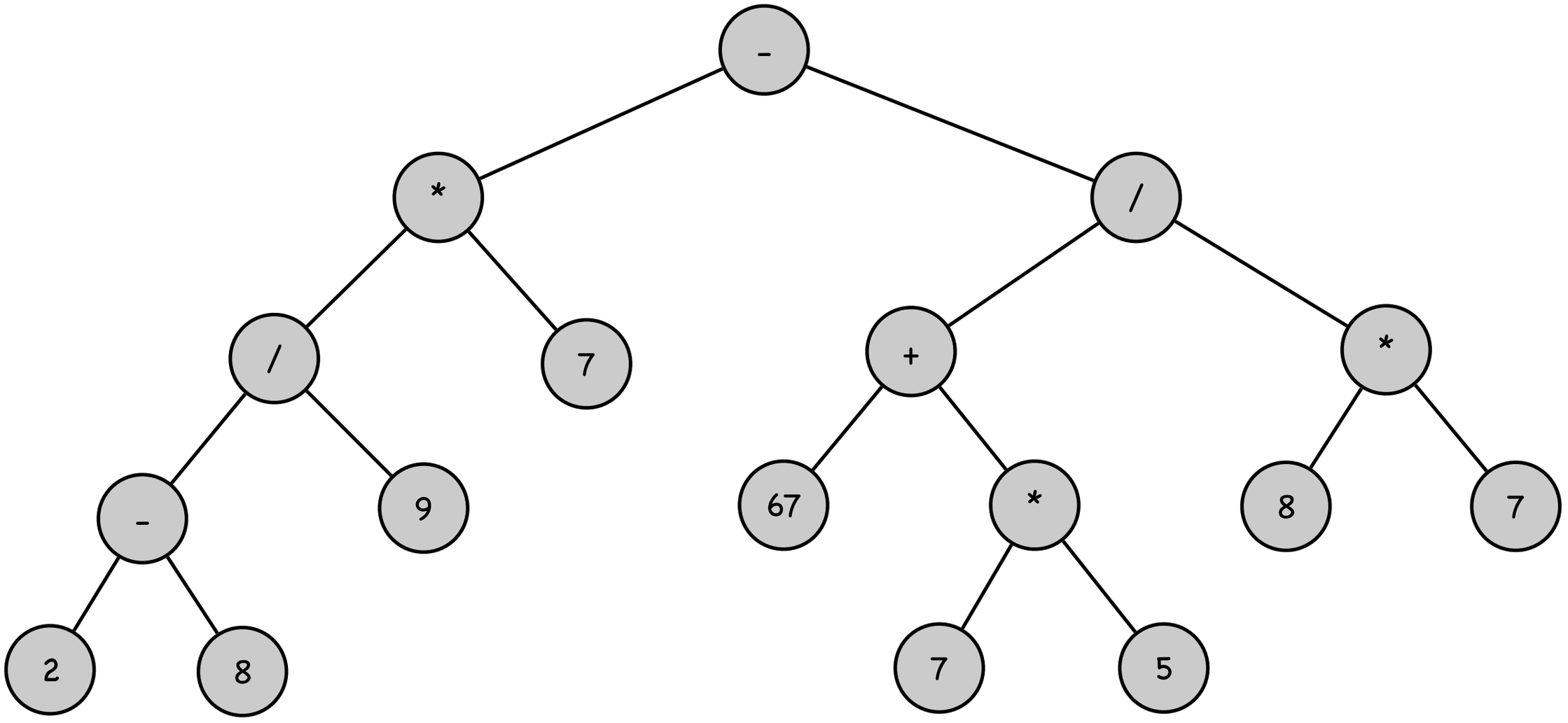
A stack tem um único elemento,

que é a árvore de expressões

pretendida!

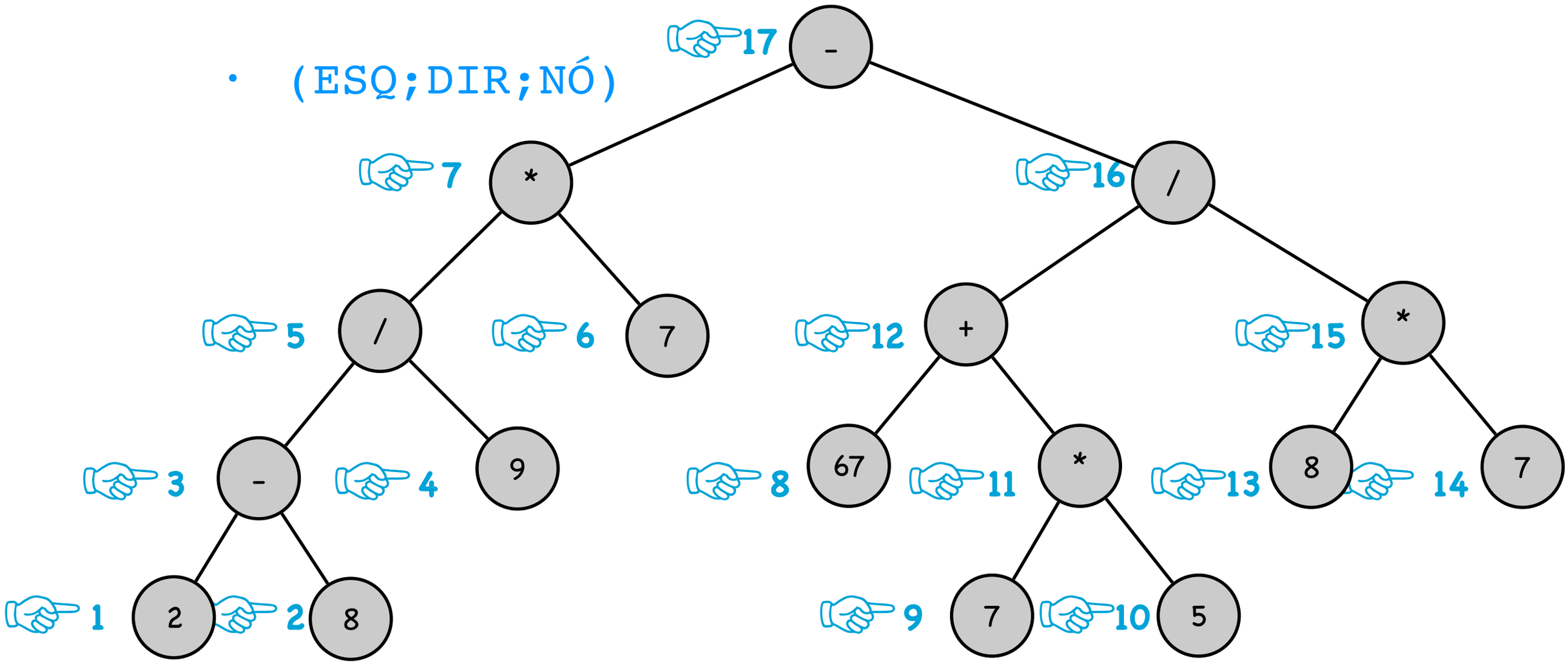
Árvores de Expressões

- Fazendo Pop



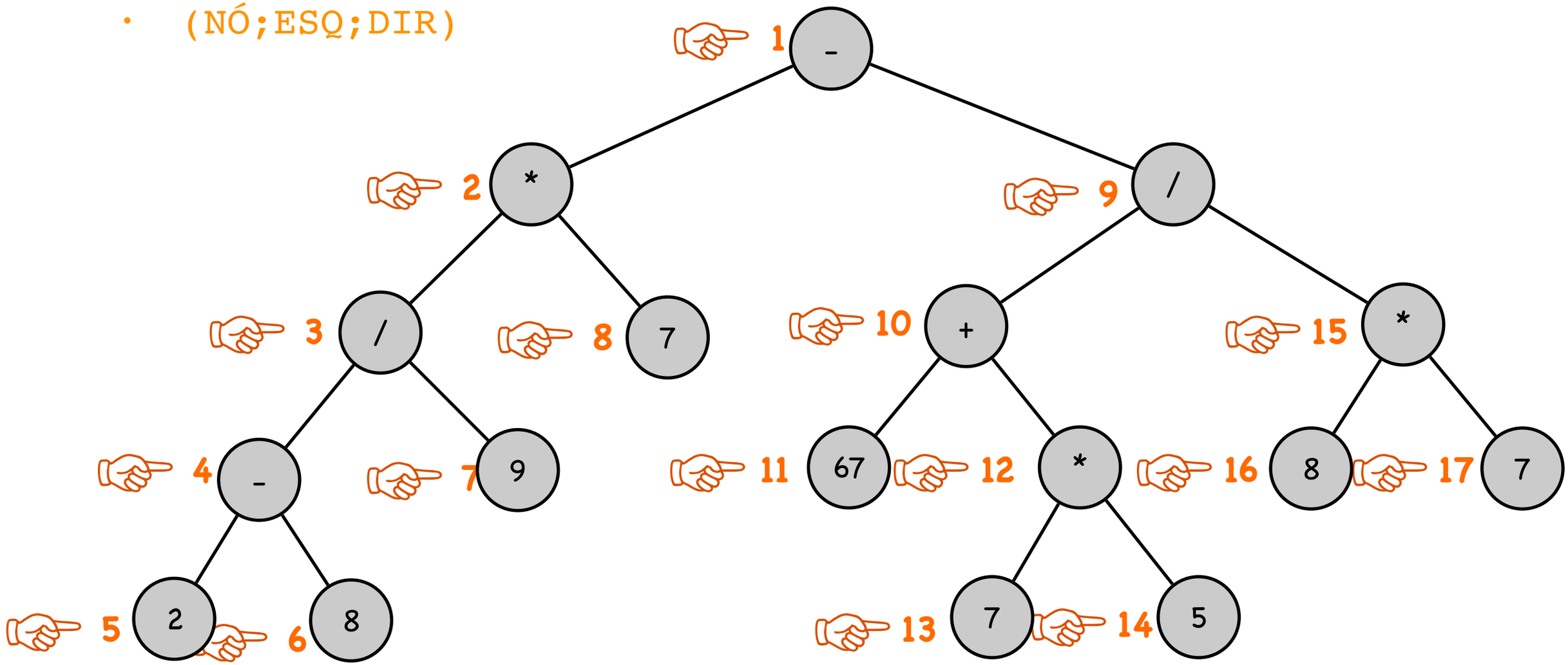
Árvores de expressões

- Percursos?
  - post-ordem?
    - (ESQ;DIR;NÓ)



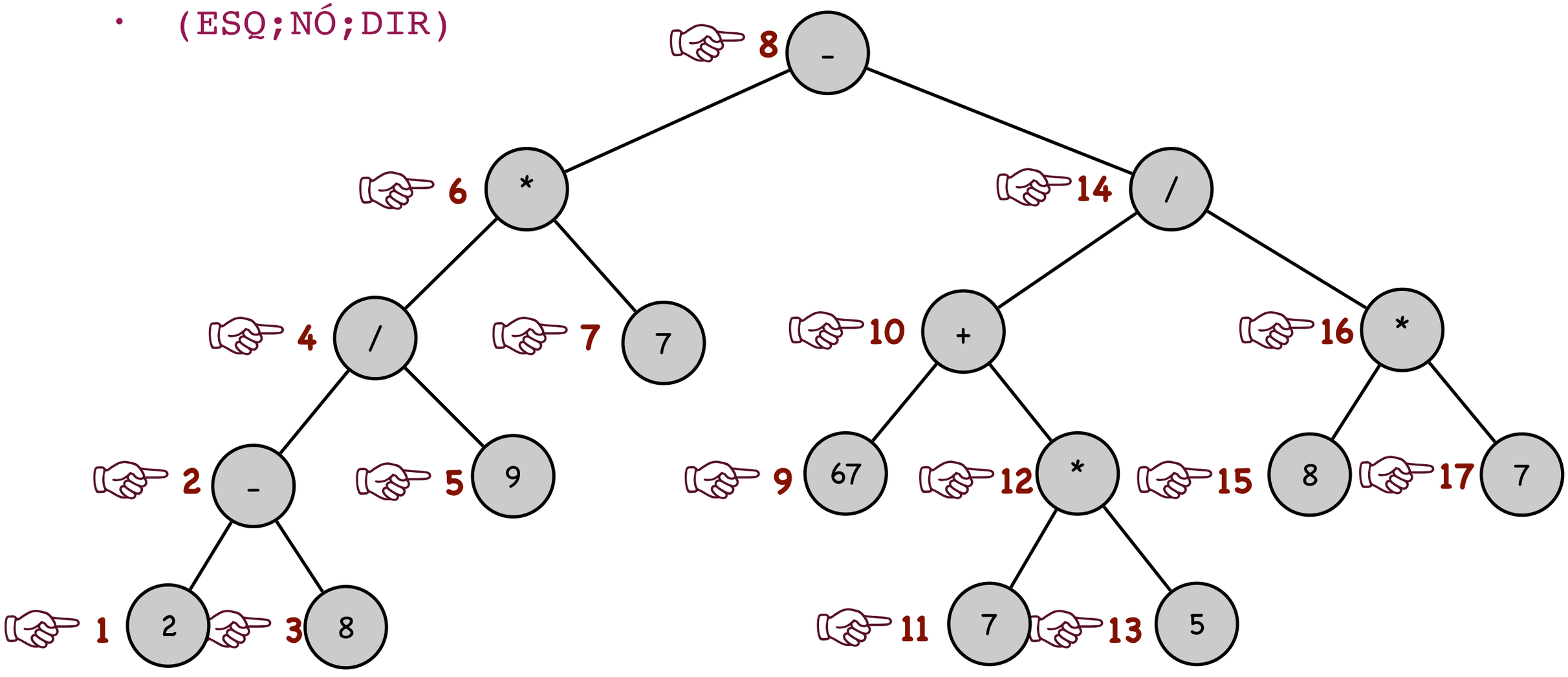
Árvores de expressões

- pre-ordem?
  - (NÓ;ESQ;DIR)

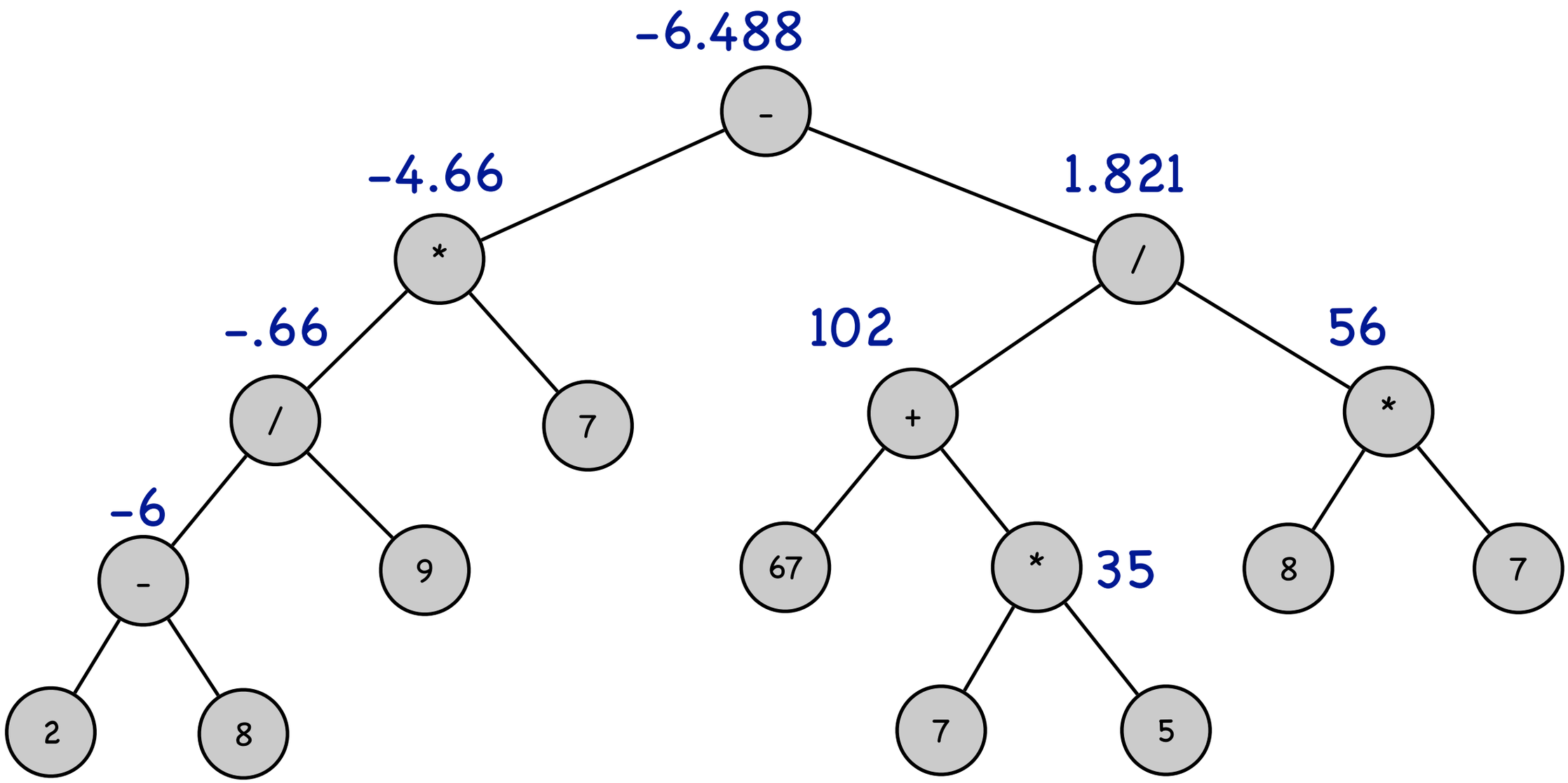


Árvores de expressões

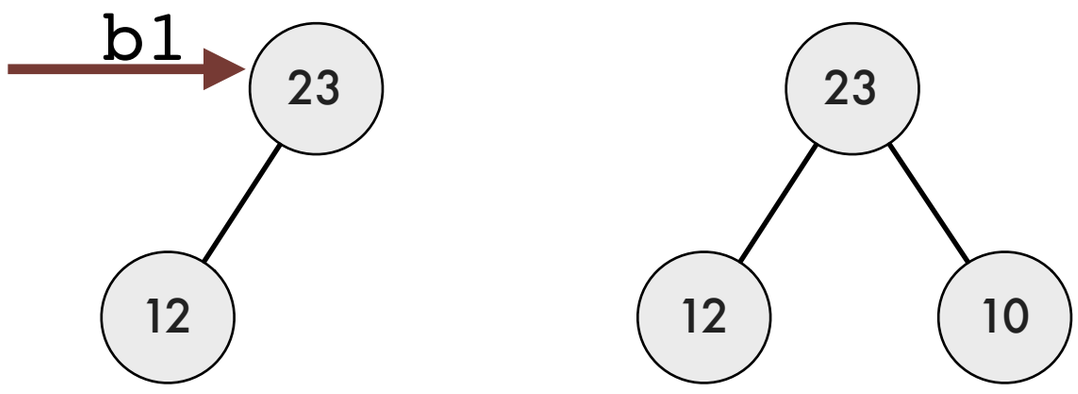
- em-ordem?
  - (ESQ;NÓ;DIR)



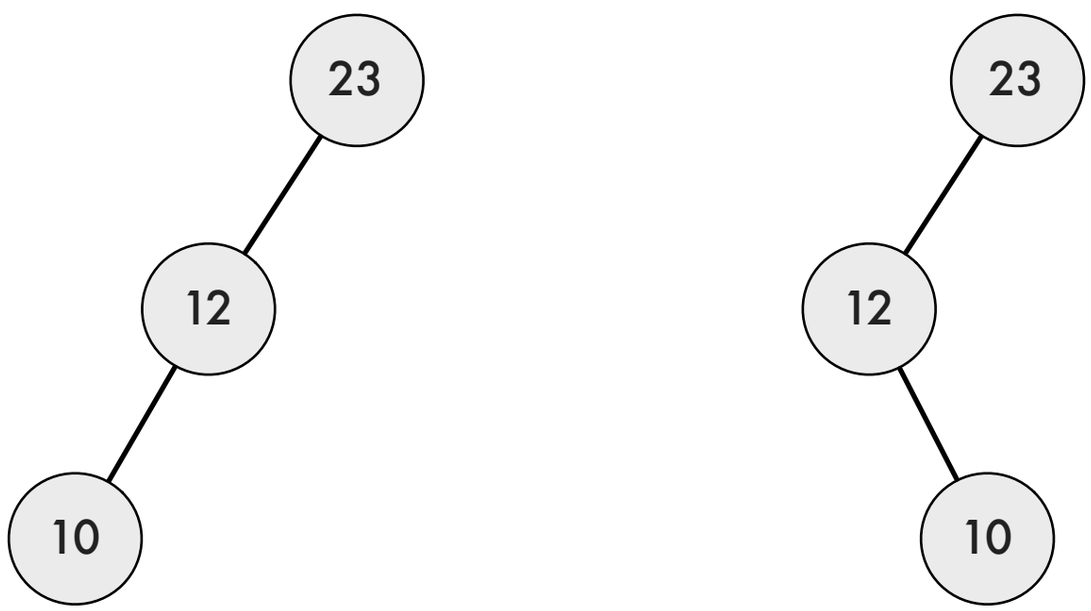
Avaliação duma árvore de expressões?



Árvores Binárias



`Add(10,b1)` ?

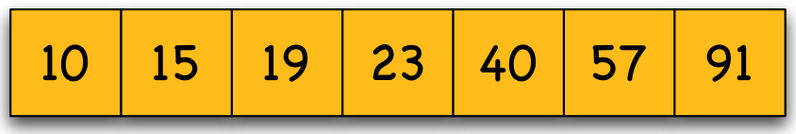


Árvore Binária de Pesquisa(ABP)

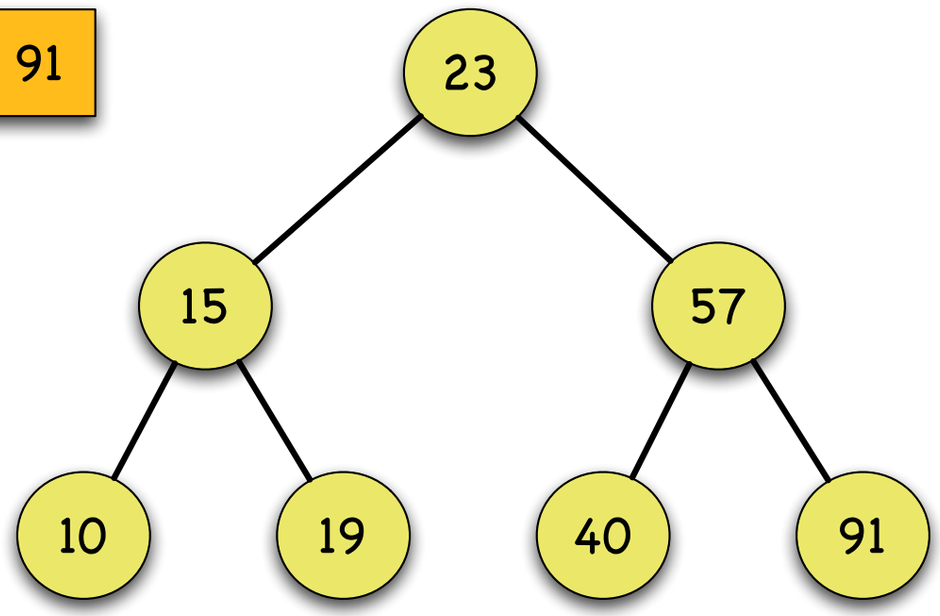
- Árvore binária com convenção FE < N < FD (ABP,BinarySearchTree)

- Por que a pesquisa sobre listas ligadas é lenta para grandes conjuntos de dados, mesmo se estão ordenadas.

- Trabalho de casa, pesquisa binária:
  - Algoritmo



  - Complexidade?



Pesquisa Binária

- Procurar um elemento num array ordenado…

```
int pesquisa(int value, int arr[], int i, int j){
    //pesquisa value em arr, de i a j
    //retorna indice do array ou -1
    int x;
    int y;
    if (i>j)
        return -1;
    else{
        int k=(j-i)/2;
        if (arr[i+k]==value)
            return i+k;
        else {
            if (arr[i+k]>value){
                printf("%d < % d, vai para a esquerda\n", value, arr[i+k]);
                x=i;
                y=i+k-1;
                return pesquisa(value,arr,x,y);
            }
            else{
                x=i+k+1;
                y=j;
                printf("%d > % d, vai para a direita\n", value, arr[i+k]);
                return pesquisa(value,arr,x,y);
            }
        }
    }
```

LISTAGENS

- inorder (FE;N;FD)

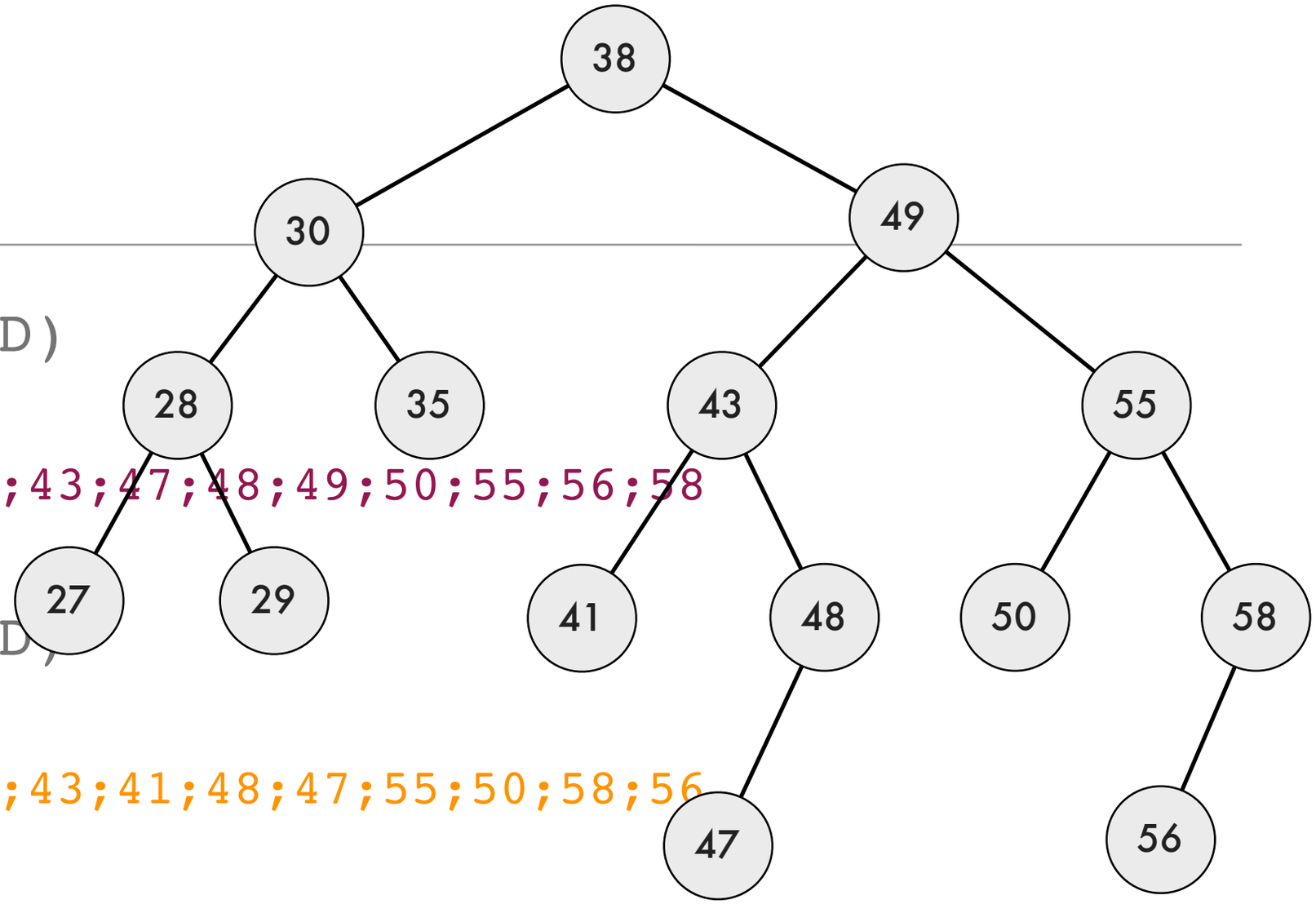
27;28;29;30;35;38;41;43;47;48;49;50;55;56;58

- preorder(N;FE;FD)

38;30;28;27;29;35;49;43;41;48;47;55;50;58;56

- postorder(FE;FD;N)

27;29;28;35;30;41;47;48;43;50;56;58;55;49;38



Arvore Binária de Pesquisa

- Definição do tipo abstracto

```
#ifndef _Tree_H
#define _Tree_H

typedef int ElementType;

struct TreeNode;
typedef struct TreeNode *Position;
typedef struct TreeNode *SearchTree;

SearchTree MakeEmpty( SearchTree T );
Position Find( ElementType X, SearchTree T );
Position FindMin( SearchTree T );
Position FindMax( SearchTree T );
SearchTree Insert( ElementType X, SearchTree T );
SearchTree Delete( ElementType X, SearchTree T );
ElementType Retrieve( Position P );

#endif
```

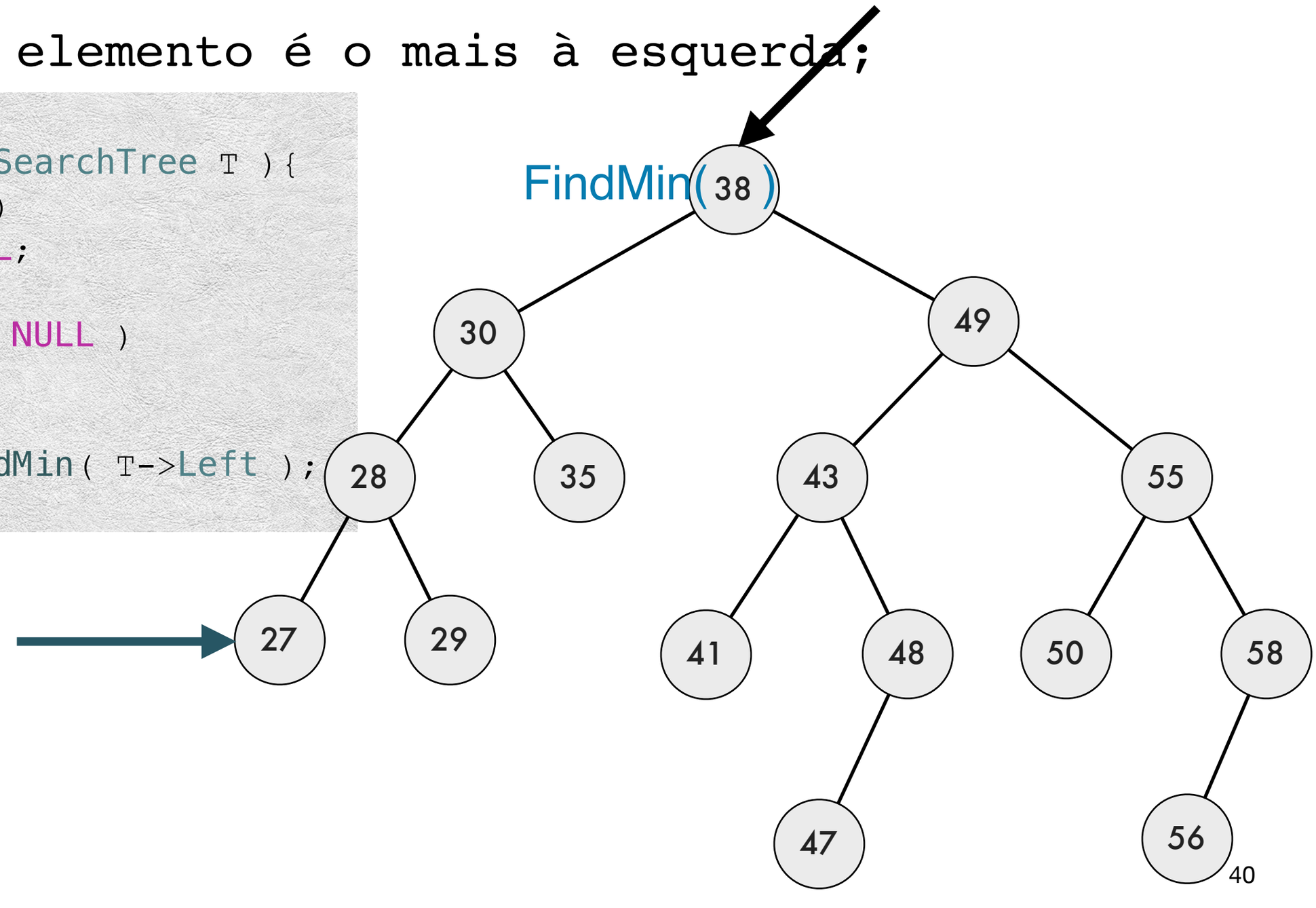
Implementação

```
…
struct TreeNode{
    ElementType Element;
    SearchTree  Left;
    SearchTree  Right;
};


SearchTree MakeEmpty( SearchTree T ){
    if( T != NULL ){
        MakeEmpty( T->Left );
        MakeEmpty( T->Right );
        free( T );
    }
    return NULL;
}

Position Find( ElementType X, SearchTree T ){ /*…}

Position FindMin( SearchTree T ){…}

Position FindMax( SearchTree T ){…}

SearchTree Insert( ElementType X, SearchTree T ){…}

SearchTree Delete( ElementType X, SearchTree T ){…}
```

Pesquisa

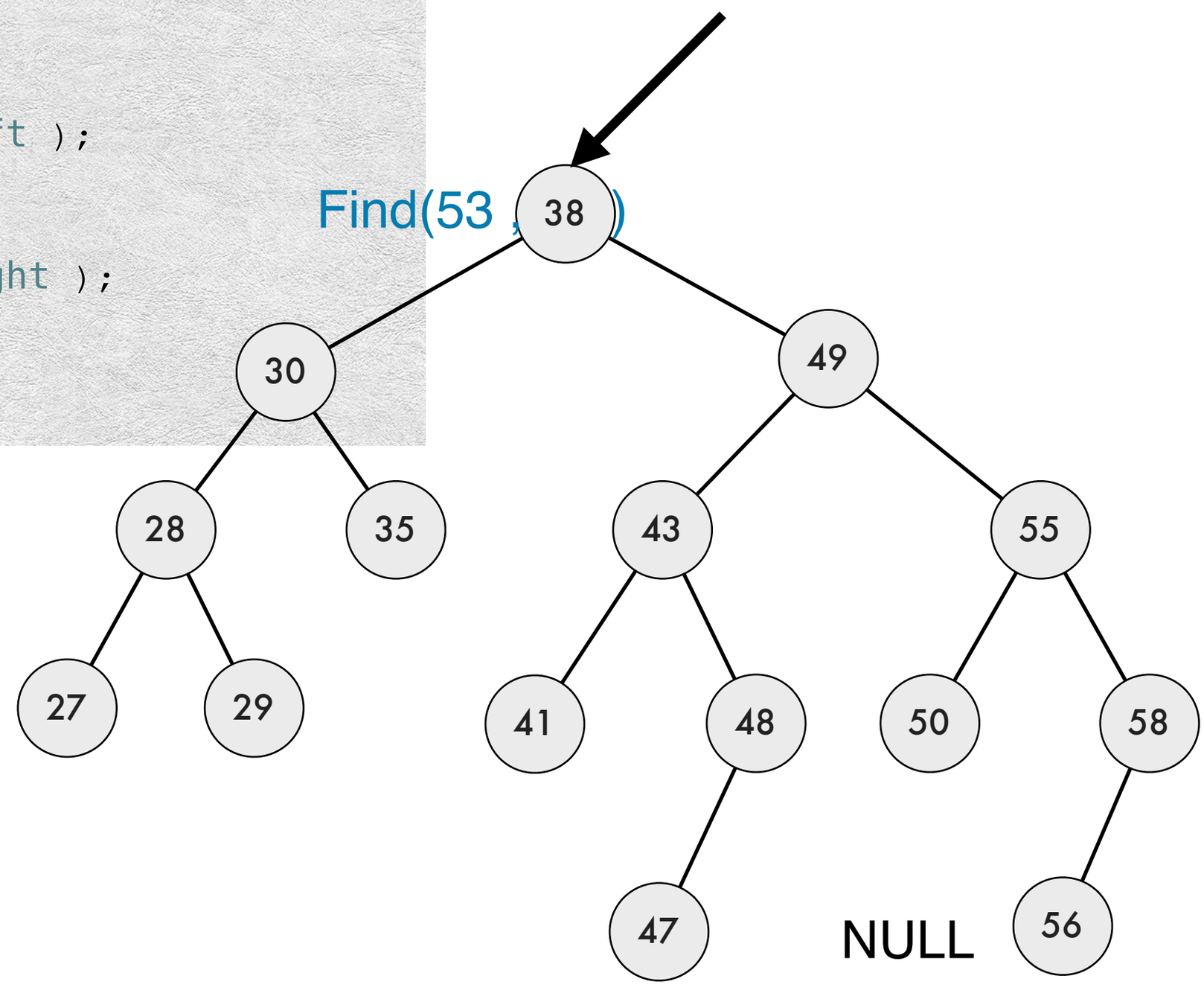
- Atendendo à ordenação:
  - o menor elemento é o mais à esquerda;

```
Position FindMin( SearchTree T ){
    if( T == NULL )
        return NULL;
    else
    if( T->Left == NULL )
        return T;
    else
        return FindMin( T->Left );
}
```



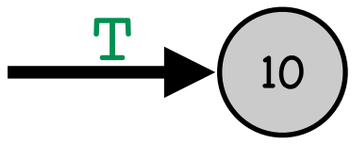
CONTAINS / FIND ?

```
Position Find( ElementType X, SearchTree T ){
    if( T == NULL )
        return NULL;
    if( X < T->Element )
        return Find( X, T->Left );
    else
    if( X > T->Element )
        return Find( X, T->Right );
    else
        return T;
}
```

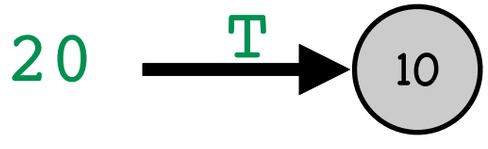


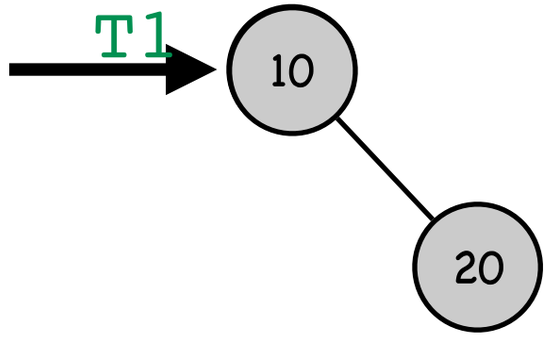
Inserção

- A inserção requer que procuremos o "sitio" certo para o efeito, dado que temos que respeitar a ordem
- Inserir sobre o vazio é fácil
  - T=Insert(10,NULL), basta retornar uma árvore cuja raiz é o nó 10, sem nada à esquerda nem à direita



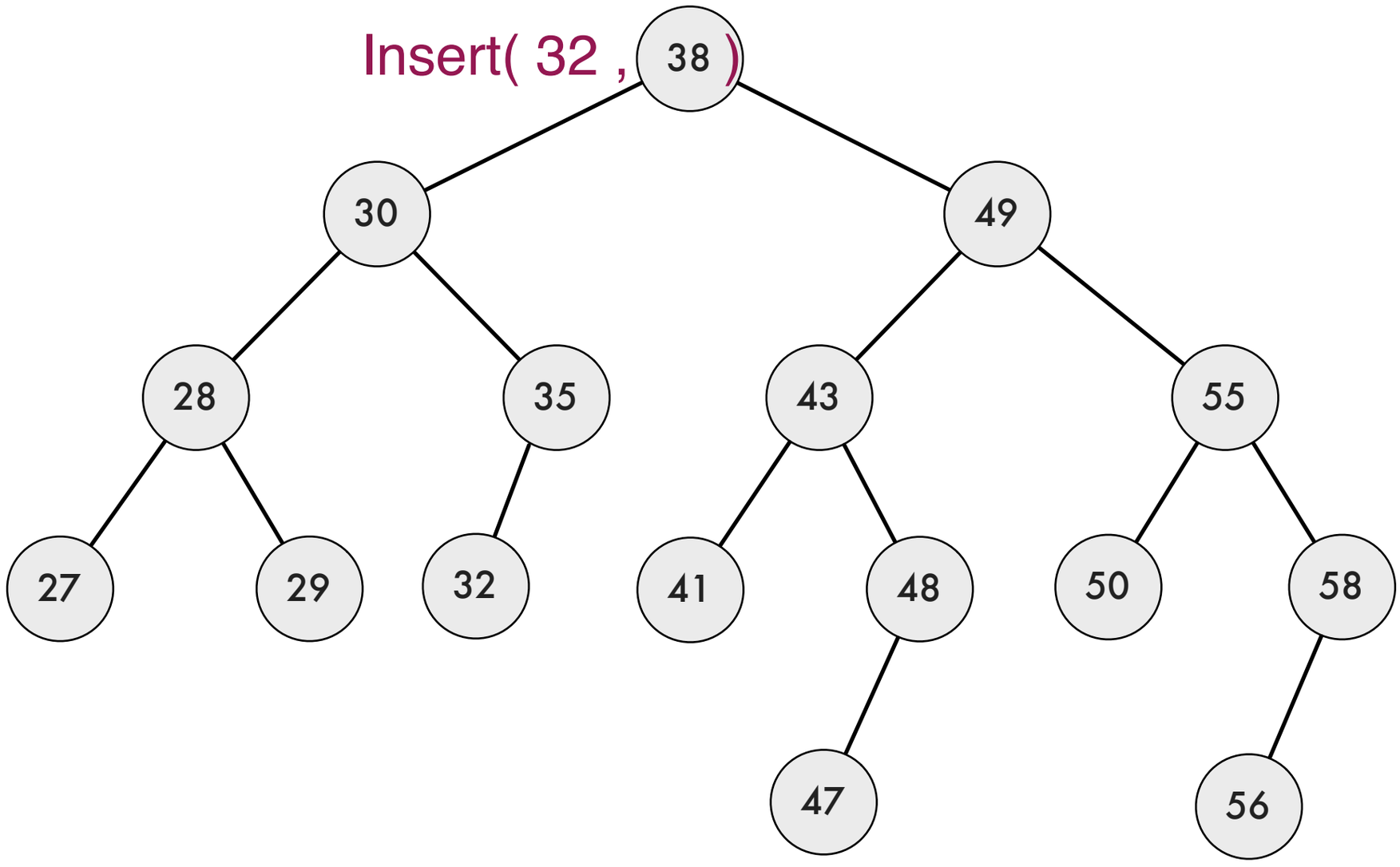
- T1=Insert(20,T)





- A inserção é sempre numa folha …

Exemplo Inserção



Inserção

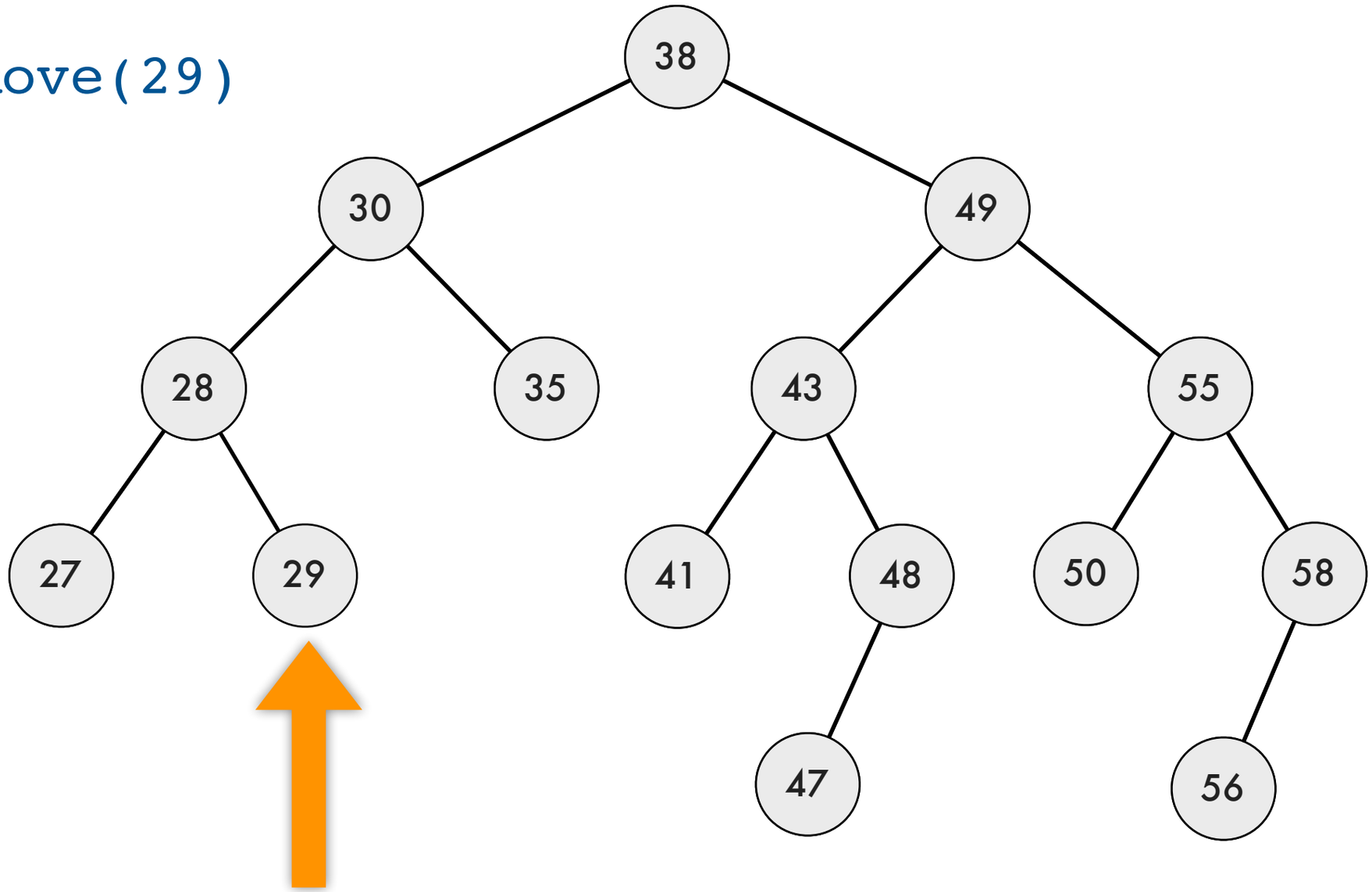
```
SearchTree Insert( ElementType X, SearchTree T ){
     if( T == NULL ){
          T = malloc( sizeof( struct TreeNode ) );
         if( T == NULL )
              FatalError( "Out of space!!!" );
          else{
              T->Element = X;
              T->Left = T->Right = NULL;
          }
     }
     else
          if( X < T->Element )
              T->Left = Insert( X, T->Left );
          else
              if( X > T->Element )
                  T->Right = Insert( X, T->Right );


       return T;  /* Do not forget this line!!*/
}
```

Remoção

- O modo como é efectuada a remoção depende do tipo de nó a remover:
  - Caso 1: Uma folha
  - Caso 2: Um nó com um filho
  - Caso 3: Um nó com 2 filhos

Remoção de um nó

- `remove(29)`



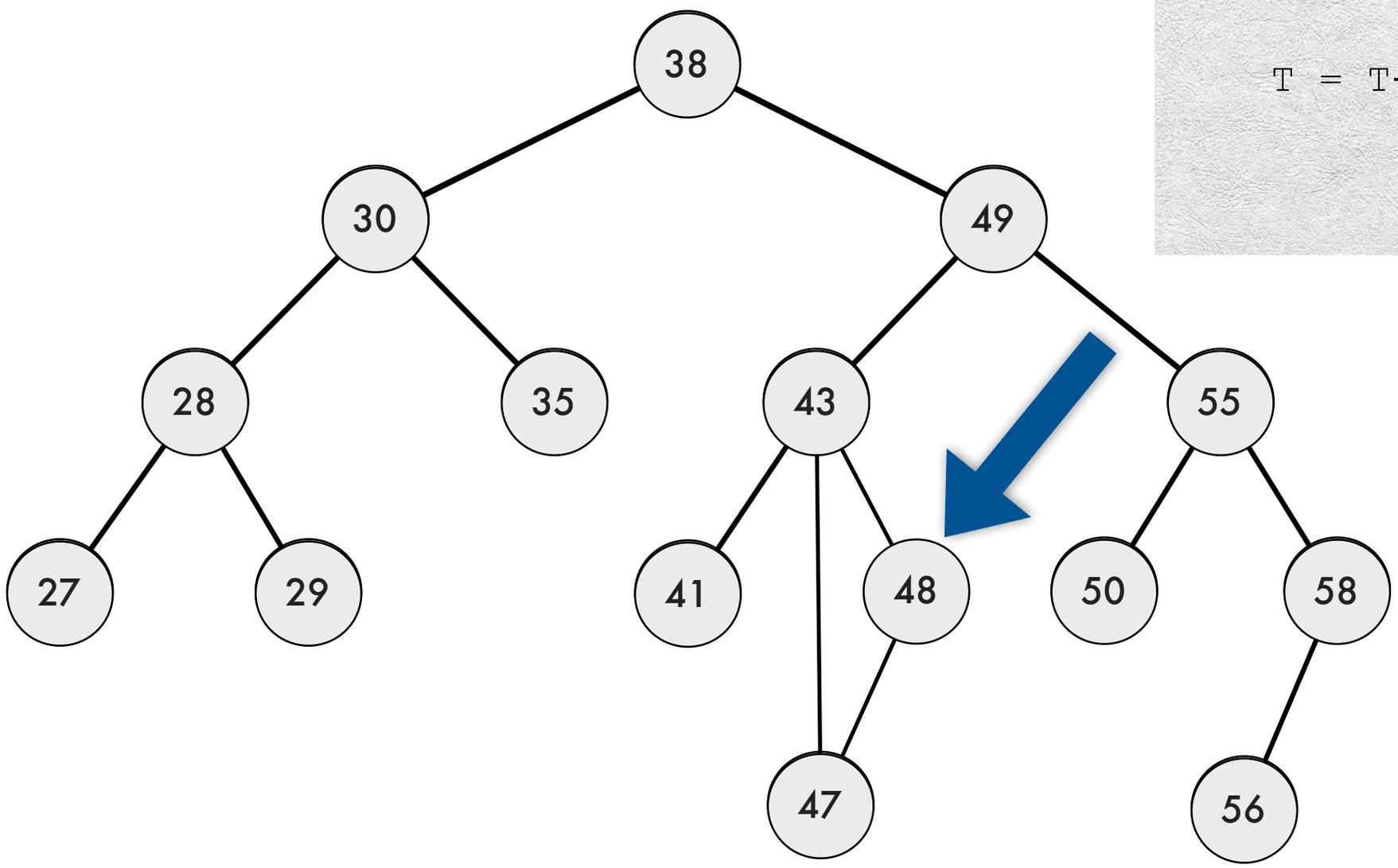
- Para remover uma folha, basta arrancá-la(basta torná-lo NULL, não estraga nada!

- Se o inserirmos a seguir, fica na mesma posição!

Remoção de um nó

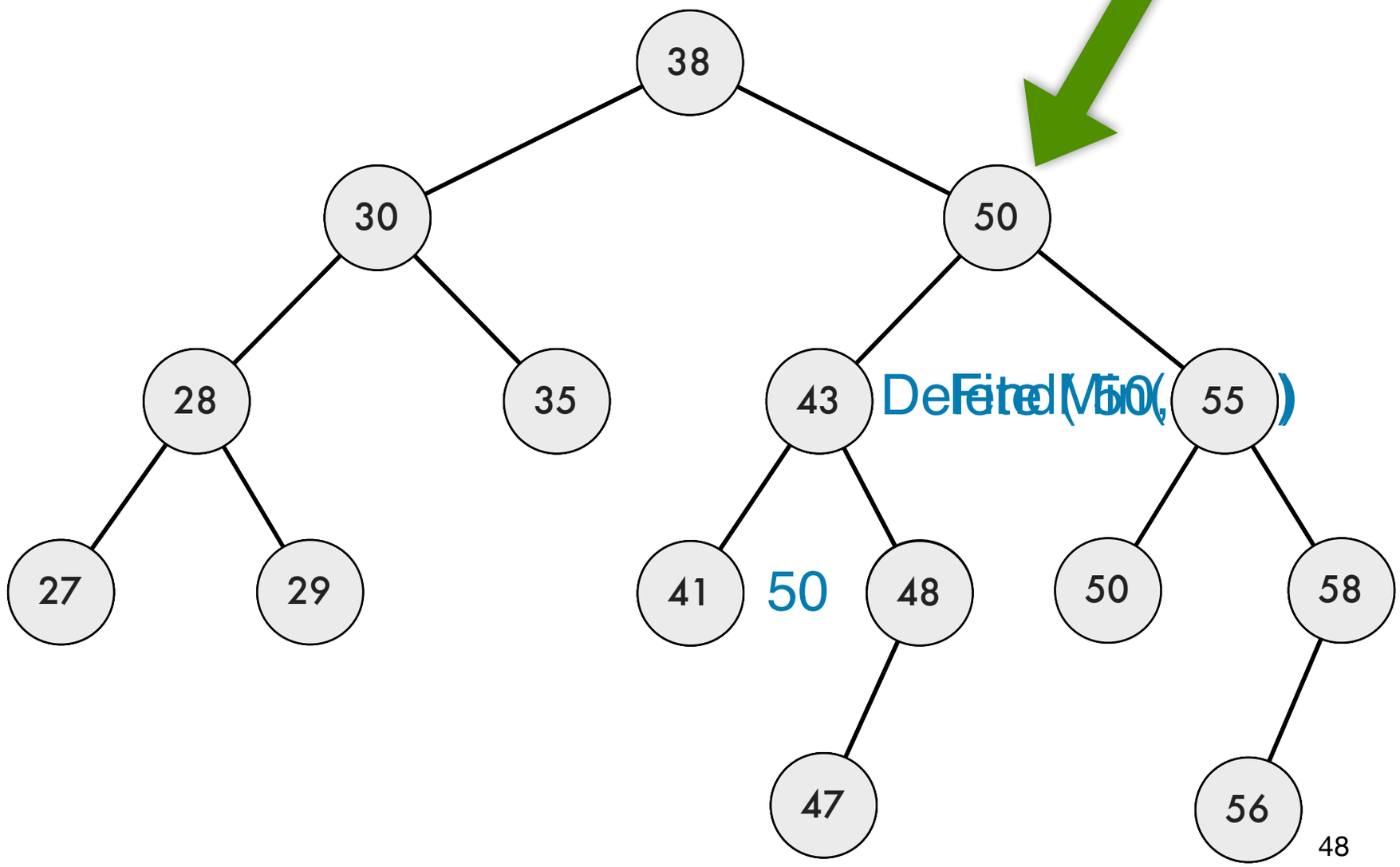
- `remove(48)`

```
if( T->Right == NULL )

    T = T->Left;
```



Remoção dum nó com 2 filhos

```
if( (T->Left!=NULL) && (T->Right!=NULL) ) { /* Two children */
        /* Replace with smallest in right subtree */
        TmpCell = FindMin( T->Right );
        T->Element = TmpCell->Element;
        T->Right = Delete( T->Element, T->Right );
 }
```

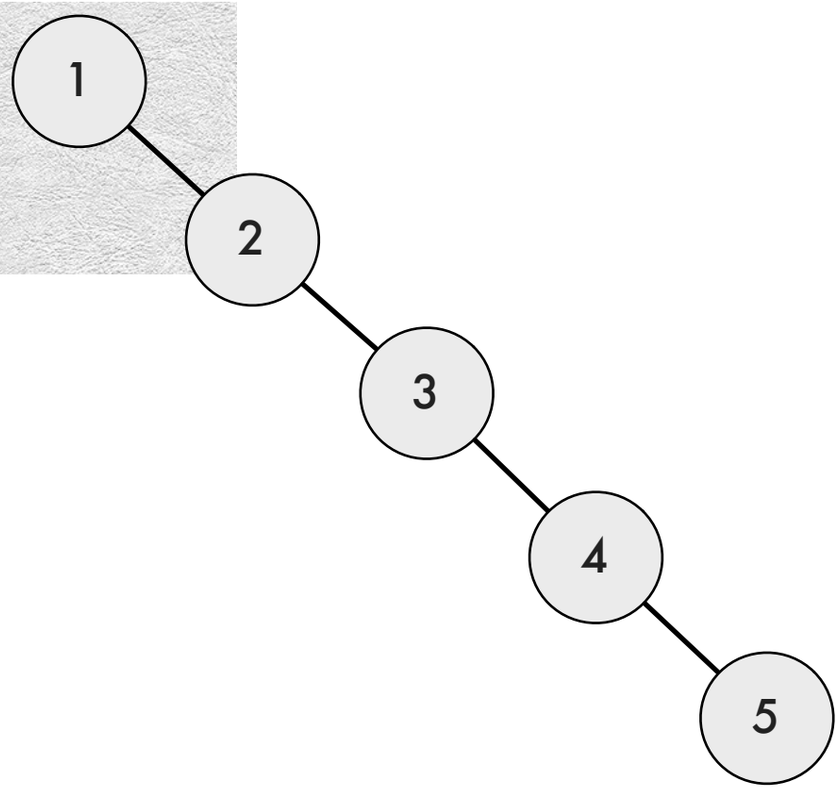


```
SearchTree Delete( ElementType X, SearchTree T ){
    Position TmpCell;

    if( T == NULL )
        Error( "Element not found" );
    else
        if( X < T->Element )  /* Go left */
            T->Left = Delete( X, T->Left );
        else
            if( X > T->Element )  /* Go right */
                T->Right = Delete( X, T->Right );
            else  /* Found element to be deleted */
                if( (T->Left!=NULL) && (T->Right!=NULL) ) { /* Two children */
                    /* Replace with smallest in right subtree */
                    TmpCell = FindMin( T->Right );
                    T->Element = TmpCell->Element;
                    T->Right = Delete( T->Element, T->Right );
                }
                else{  /* One or zero children */
                    TmpCell = T;
                    if( T->Left == NULL ) /* Also handles 0 children */
                        T = T->Right;
                    else if( T->Right == NULL )
                            T = T->Left;
                }
            free( TmpCell );

    return T;
}
```

ABP

- Criar uma ABP, inserindo todos os números de 1 até 50.

```
SearchTree T2=NULL;
   for (int i=1;i<=50;i++)
        T2=Insert(i,T2);
```



Altura da árvore?

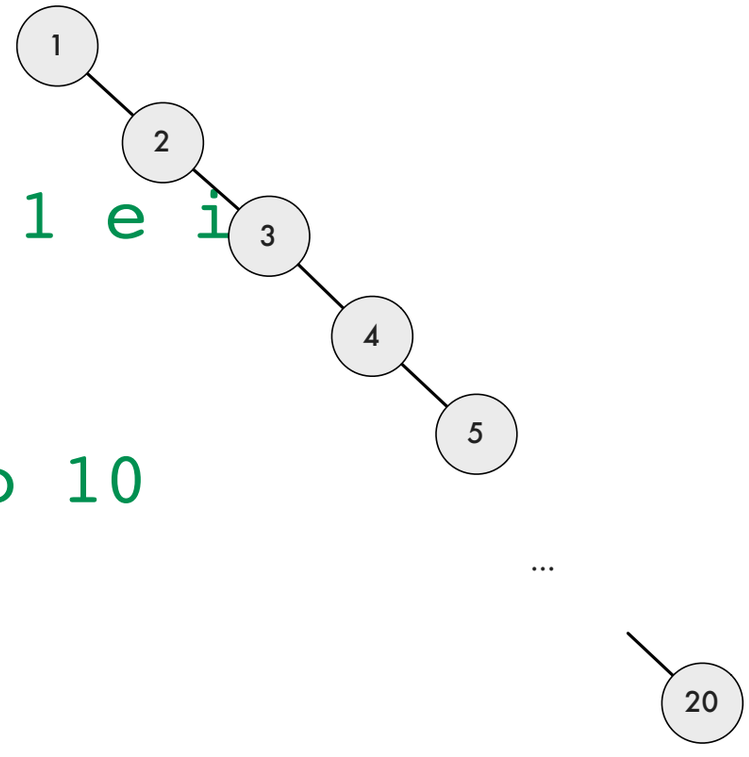
ADIVINHA QUE NÚMERO É?

- Alguém pensa num número entre 1 e 20. Outra pessoa tem de adivinhar qual o número pensado podendo perguntar se é maior ou menor que o número que o número sugerido.

- Quantas tentativas em média são necessárias para adivinhar o número?

  - Depende da estratégia:Começar no 1 e ir andando..Ou acerto ou é maior…

  - Pior caso? Melhor caso?Neste caso 10 tentativas(em média);



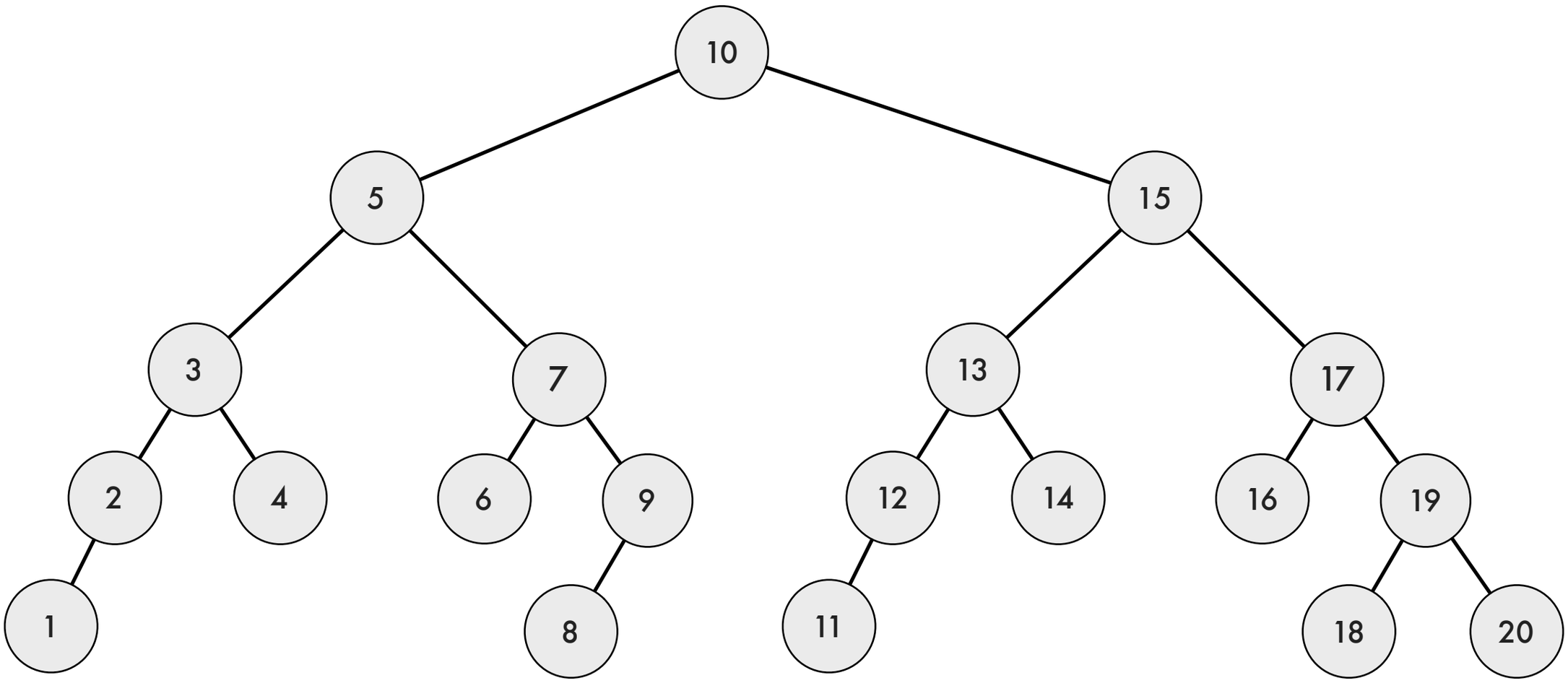
Adivinha que número é?

- Uma estratégia mais inteligente:
  - Escolher o número do meio (10 no início) se for maior dizer 15, se for menor dizer 5,...
  - Neste caso quantas tentativas em média?

- Aposto que no máximo em 4 tentativas adivinho qual o número em que pensaram….
- Na melhor das hipóteses adivinho à primeira.
- Porquê 4?

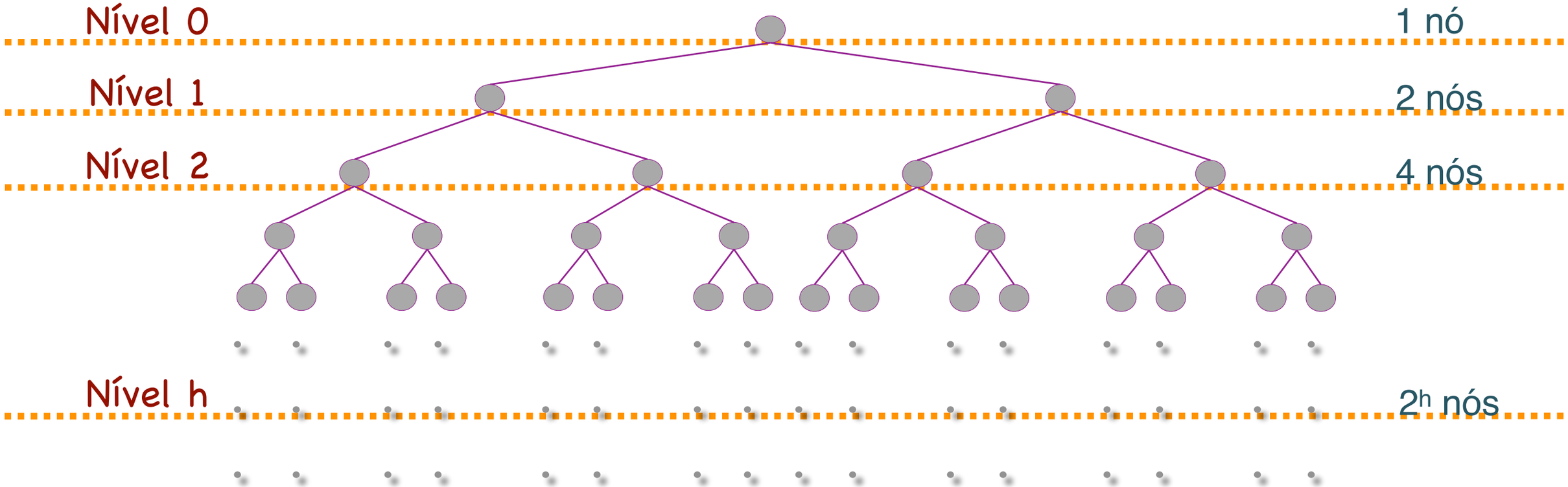
Adivinha que número é?

Árvore binária de Pesquisa, com os 20 números, inseridos pela ordem 10,5,15, ….



Vamos contar

- Qual a altura duma árvore binária com N nós e completa em cada nível?



Relação entre altura e nº de nós?

- Seja N o número total de nós da árvore binária completa de altura h

$$N = 1 + 2 + \cdots 2^h \Leftrightarrow$$

$$\Leftrightarrow N = \sum_{i=0}^{h} 2^i \Leftrightarrow$$

$$\sum_{n=0}^{N} r^n = \frac{1 - r^{N+1}}{1 - r}$$

Progressão geométrica de razão r

$$\Leftrightarrow N = \frac{2^{h+1} - 1}{2 - 1} \Leftrightarrow$$

$$\Leftrightarrow N + 1 = 2^{h+1} \Leftrightarrow$$

$$\Leftrightarrow h = \log(N + 1) - 1$$

$$\therefore h \text{ é } O(\log(N))$$

Complexidade das operações sobre ABPs?

```
….
SearchTree MakeEmpty( SearchTree T );
Position Find( ElementType X, SearchTree T );
Position FindMin( SearchTree T );
Position FindMax( SearchTree T );
SearchTree Insert( ElementType X, SearchTree T );
SearchTree Delete( ElementType X, SearchTree T );
…
```

Exercícios

- Mostre o resultado de inserir:
  - 45;34;3;6;19;32;77;8 numa inicialmente vazia ABP;
  - Qual a ABP resultante de:
    - remover(19);
    - remover(3);
    - remover(32);
- Listagem pre/pos/em ordem da ABP resultante?