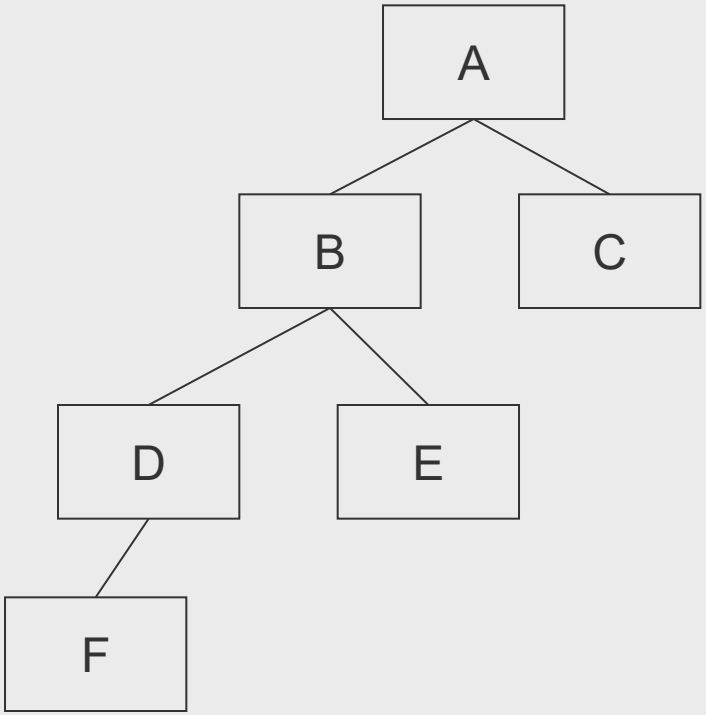
Estruturas de Dados

Árvores Binárias

2023/2024

Árvores Binárias

- Árvore em que cada nodo tem 2 ou menos descendentes.



A,B: 2 descendentes
D: 1 descendente
E: 0 descendentes

Operações com Árvores Binárias

- Operações Básicas:
  - *getNumeroNodos()*
  - *getAltura()*
  - *imprimePreOrdem()*
  - *imprimePosOrdem()*
  - *imprimeEmOrdem()*

Operações com Árvores Binárias

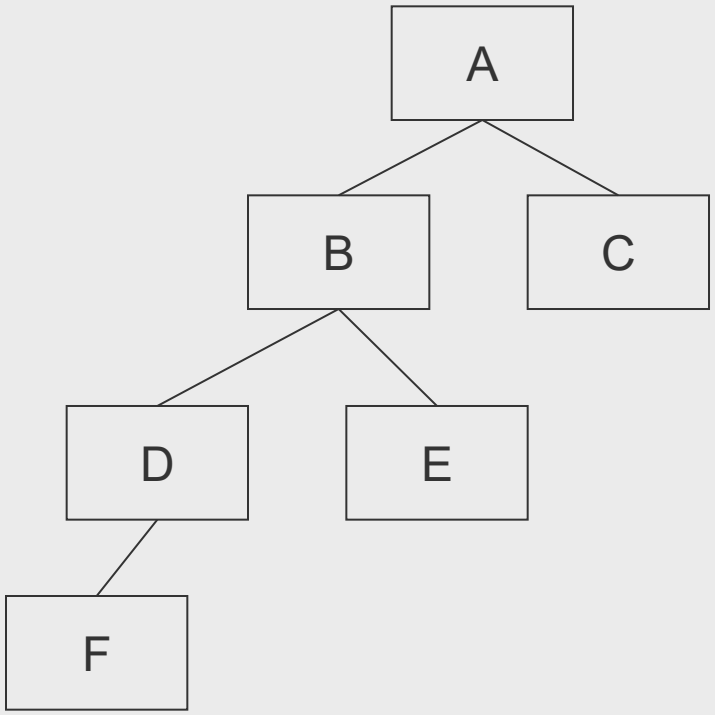
- Como calcular o número de nodos de uma árvore com raiz R?
- Nada mais simples!
  - O número de nodos de uma árvore com raiz *R* é dado pela soma do número de nodos na subárvore esquerda com o número de nodos na subarvore direita mais um (a própria raiz)!
  - E como será o cálculo da altura?

Travessia de Árvore

- Há três formas básicas de percorrer os nodos de uma árvore O(N).
- Em cada nodo da árvore:
  - pré-ordem:
    - o nodo é visitado, sendo depois percorridos o descendente esquerdo e em seguida o direito.
      - Exemplo: duplicação de árvore
  - pós-ordem:
    - É primeiro percorrido o o descendente esquerdo, depois o direito, e depois visitamos o nodo.
      - Exemplo: cálculo de altura ou número e nodos.
  - (em-)ordem:
    - É primeiro percorrido o o descendente esquerdo, visitamos o nodo e depois percorremos o descendente direito.

Travessia de Árvore

- Sequência de travessia



•Qual a sequência de nodos visitados em pré-ordem?

•Qual a sequência de nodos visitados em pós-ordem?

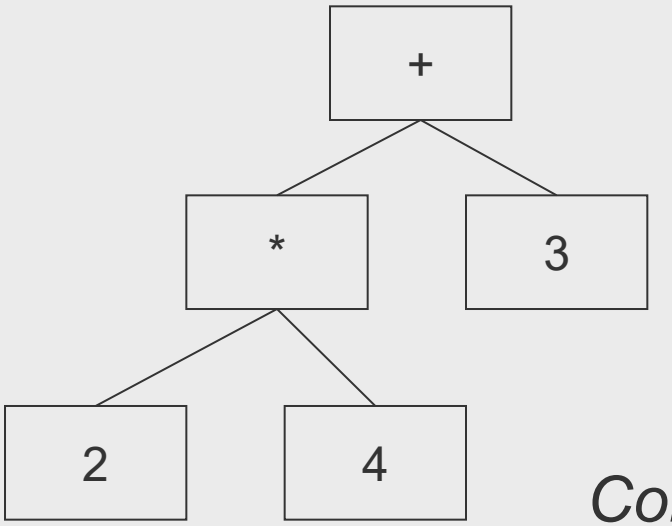
•Qual a sequência de nodos visitados em ordem?

Travessia de Árvore

- É mais simples usar métodos recursivos para implementar a travessia, mas é possível fazê-lo iterativamente com o auxilio de uma pilha!
- A travessia por nível é possível, com o auxílio de uma fila:
  - Como?

Aplicações de Árvores Binárias

- Representação de expressões:



*2*4+3*

Como imprimir a expressão?
Como calcular o valor resultante?

Aplicações de árvores Binárias

Representação de expressões: Impressão (em-ordem)

```
class Nodo{
String valor;
Nodo esquerda, direita;
...
void expressao(){
  if(esquerda!=null) esquerda.expressao();
  System.out.print(valor);
  if(direita!=null) direita.expressao();
  }

}
```

Aplicações de árvores Binárias

Representação de expressões: Cálculo de valor (pós-ordem)

```
class Nodo{
String valor;
Nodo esquerda, direita;
...
Integer CalculaValor(){
  if (esquerda==null)&&(direita==null)
          return Integer.valueOf(valor);
  int valEsq=esquerda.CalculaValor().intValue();
  int valDir=0;
  if(direita!=null)
   direita=direita.CalculaValor().intValue();
  if (valor.equals("+"))
   return valDir+ValEsq;
  if (valor.equals("*"))
   return valDir*ValEsq;
  if (valor.equals("-"))
   return valDir-ValEsq;
   ...
```

Aplicações de árvores Binárias

- Como construir a árvore correspondente a uma dada expressão?
  - Vamos primeiro analisar o caso de a expressão estar um notação *postfix* (o operador encontra-se *depois* dos argumentos) em vez de se encontrar entre eles (notação *infix*)
    - *Exemplo: 4+3*2 (infix) é equivalente a 4 3 2 * + (postfix)*

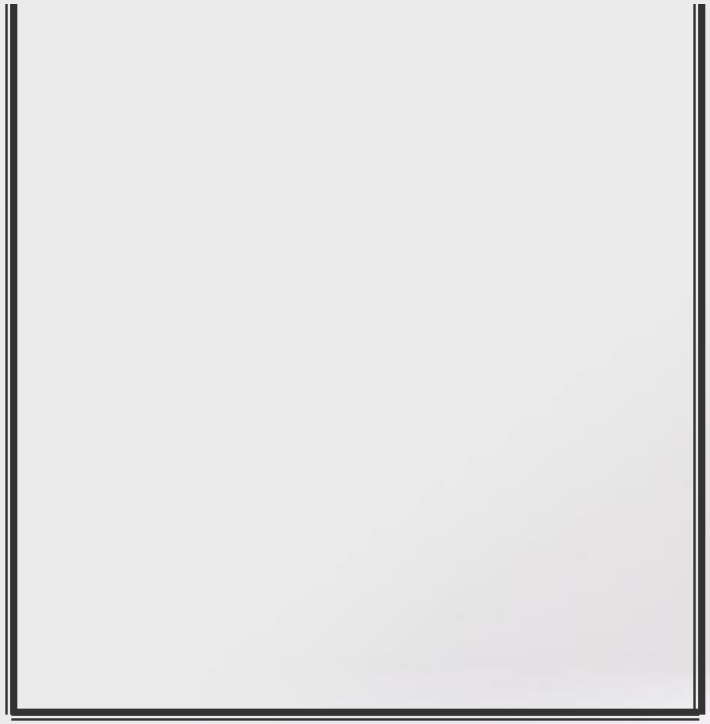
Aplicações de árvores Binárias

- A conversão de *postfix* para árvore é simples:
  - Utiliza-se uma *Pilha* auxiliar e:…
    - Percorrendo os vários símbolos da expressão:
      - Guardam-se na pilha os símbolos que não são operadores…
      - Quando se encontra um operador, é criado um nodo correspondente a essa operação, sendo o descendente direito e esquerdo correspondentes aos dois valores no topo da pilha. Esse nodo é guardado na pilha.

Aplicações de árvores Binárias

- Exemplo (conversão post-fix para árvore):

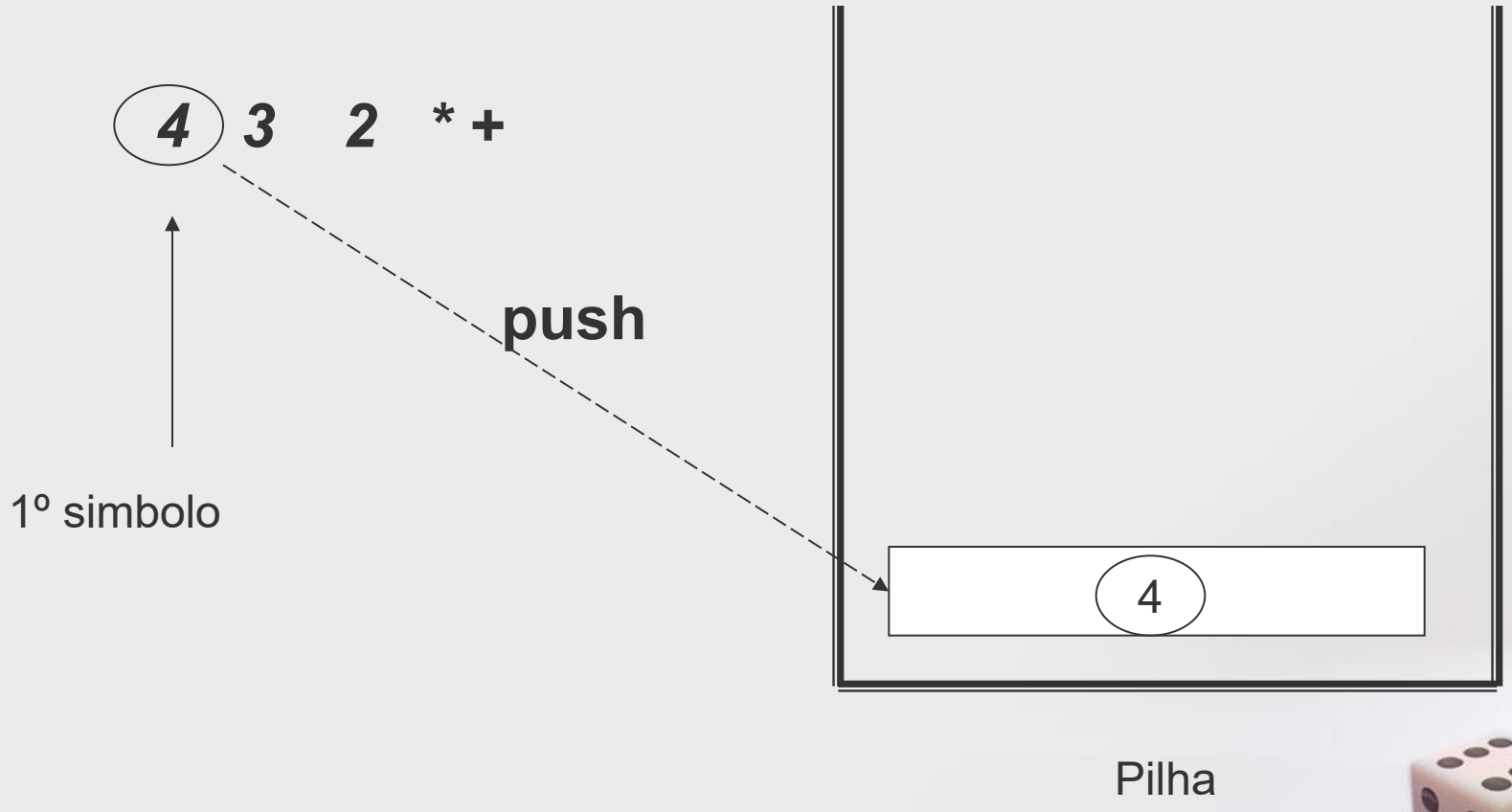
4 3 2 * +



Pilha

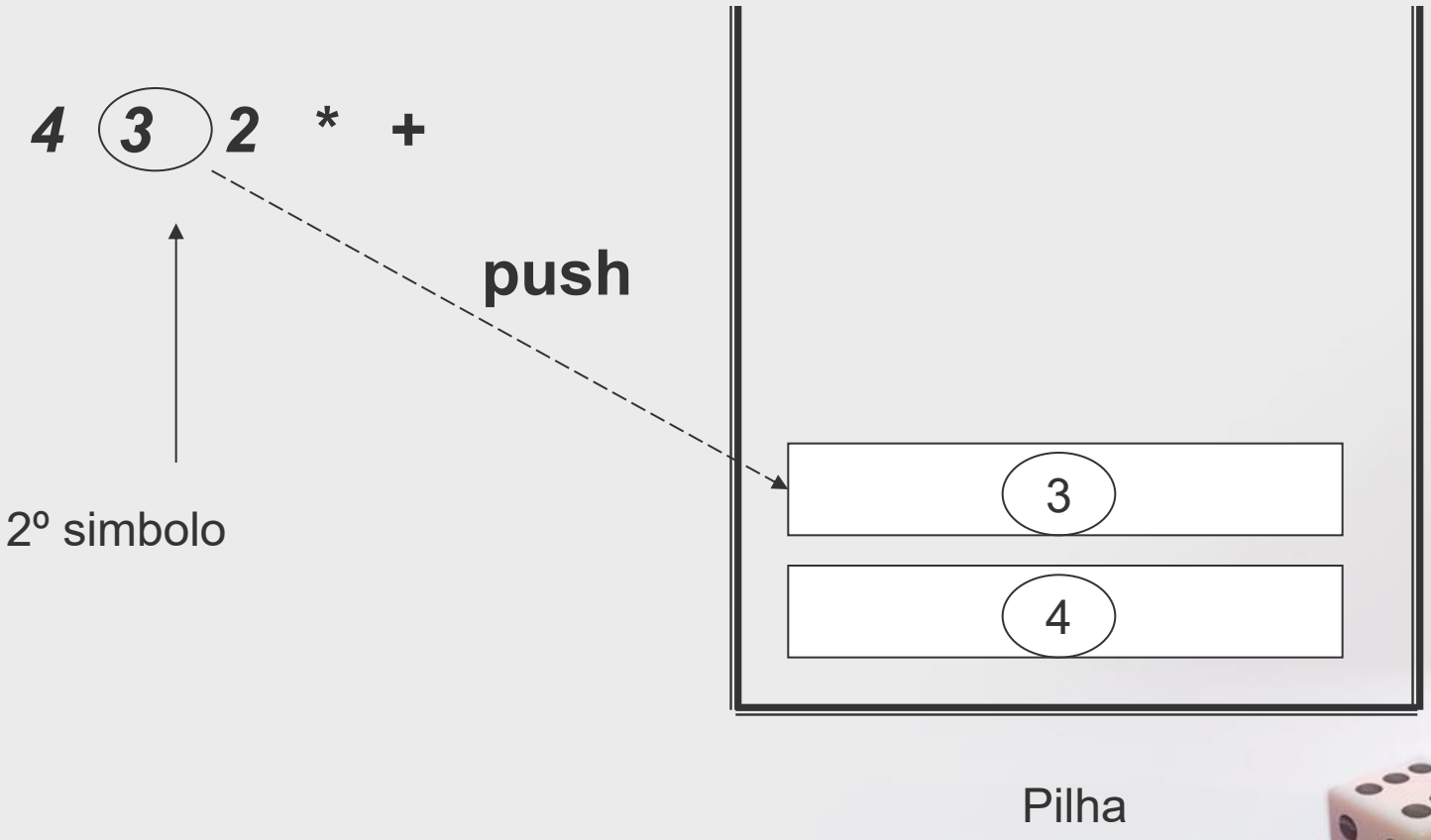
Aplicações de árvores Binárias

- Exemplo (conversão post-fix para árvore):



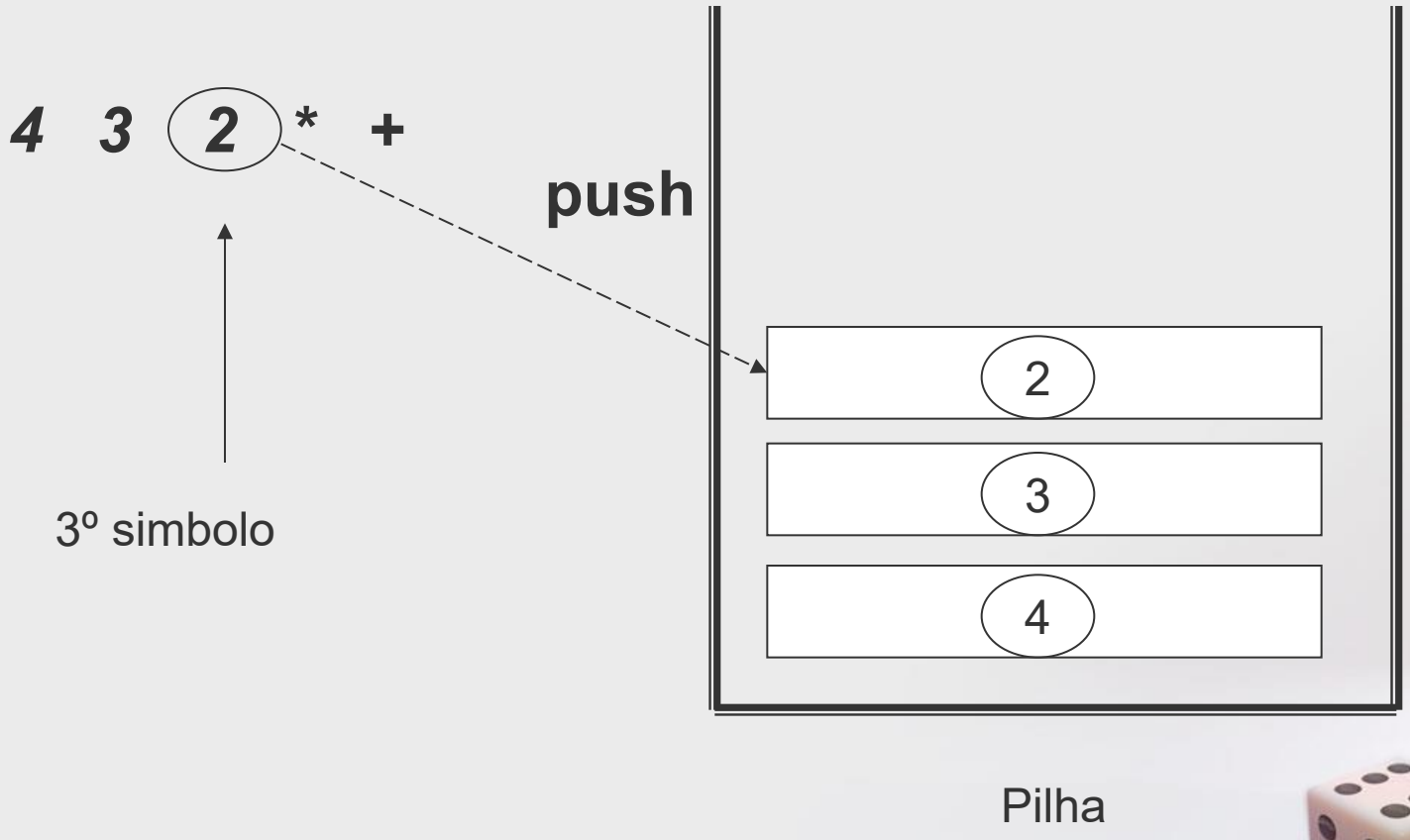
Aplicações de árvores Binárias

- Exemplo (conversão post-fix para árvore):



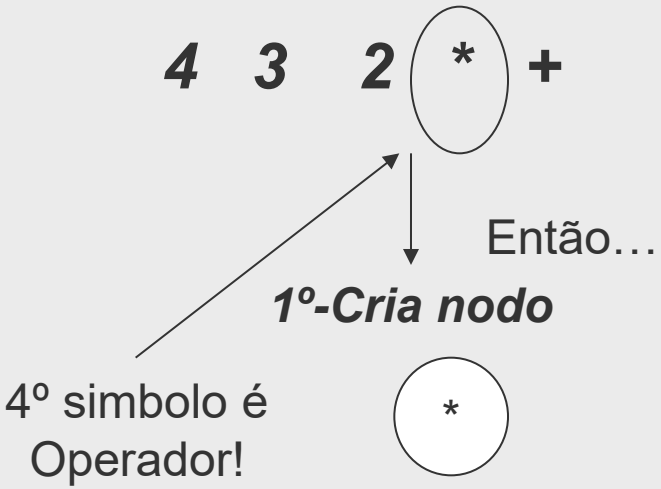
Aplicações de árvores Binárias

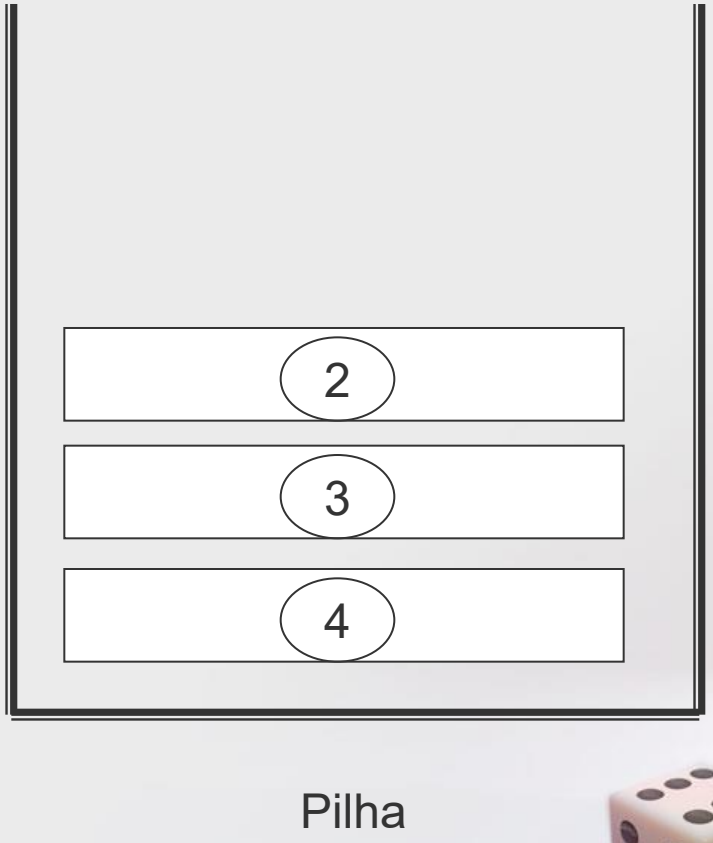
- Exemplo (conversão post-fix para árvore):



Aplicações de árvores Binárias

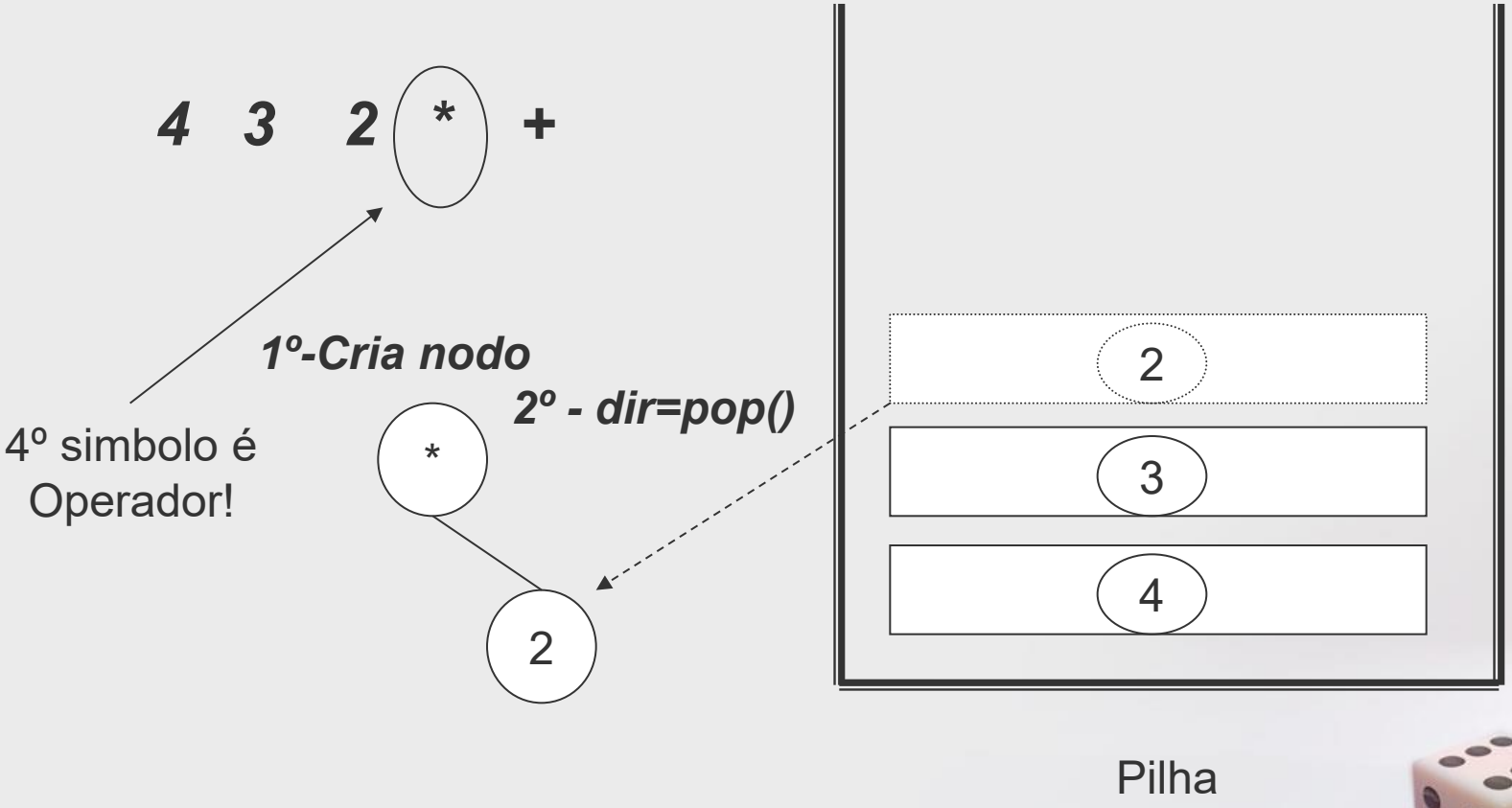
- Exemplo (conversão post-fix para árvore):





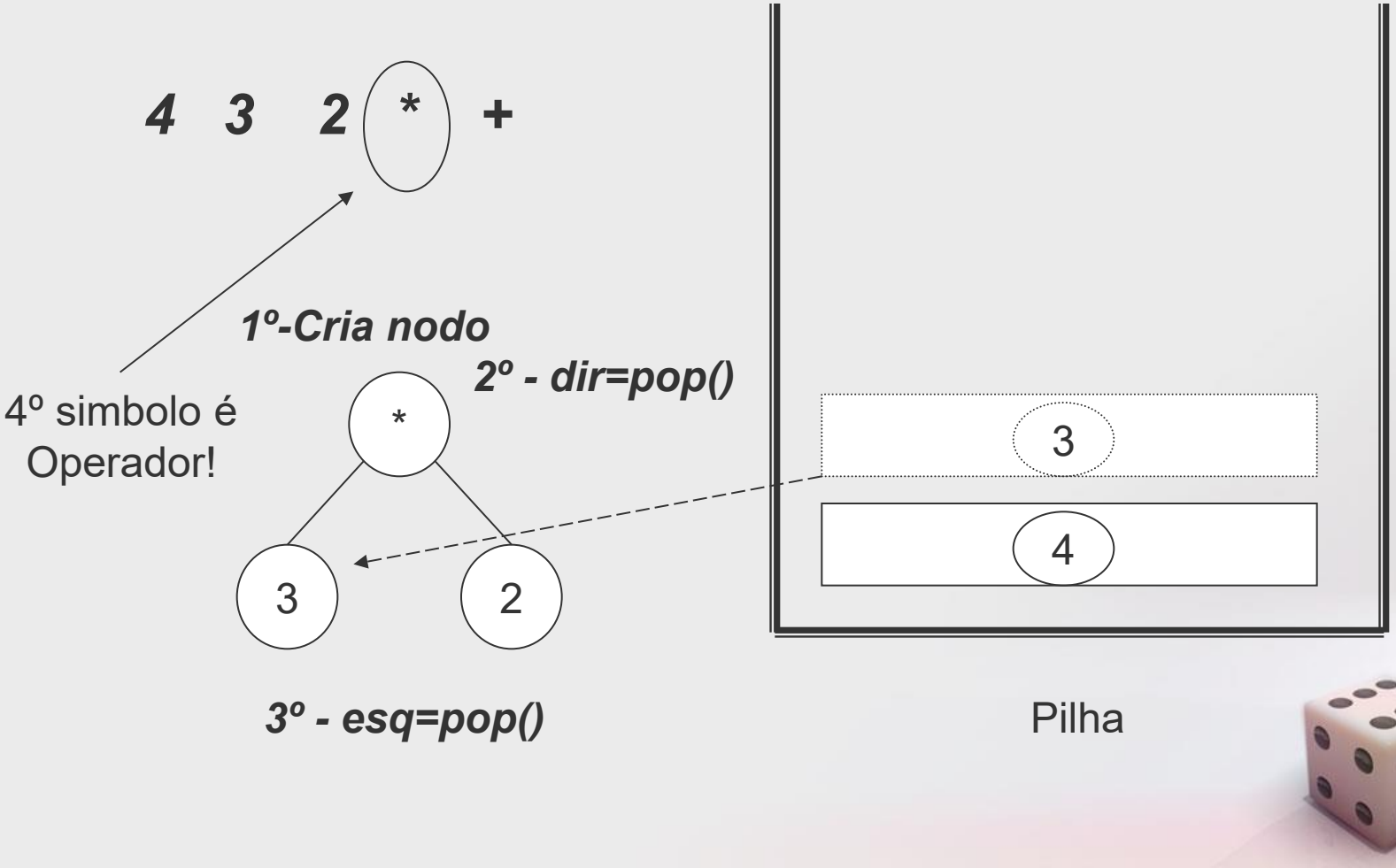
Aplicações de árvores Binárias

- Exemplo (conversão post-fix para árvore):



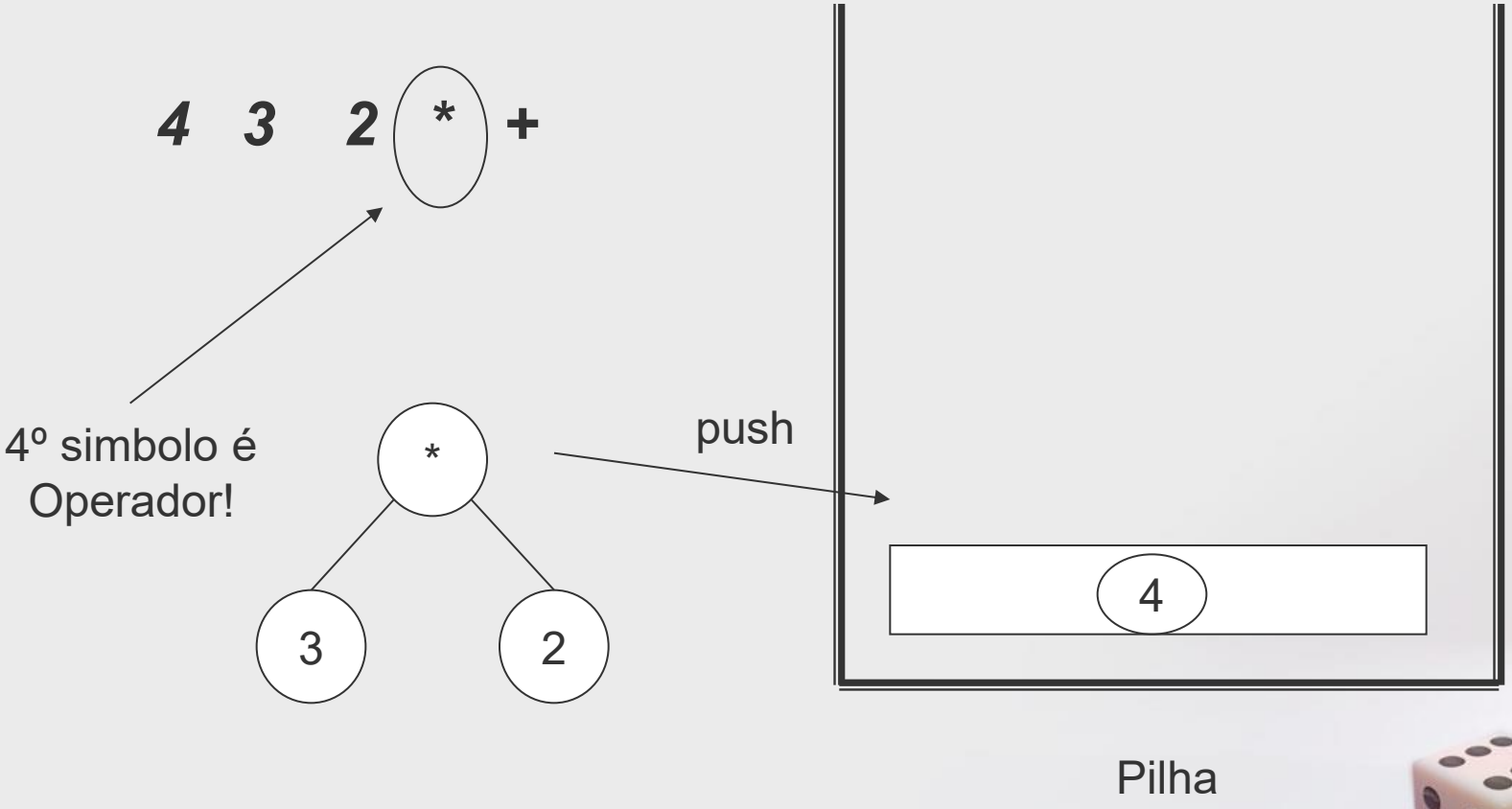
Aplicações de árvores Binárias

- Exemplo (conversão post-fix para árvore):



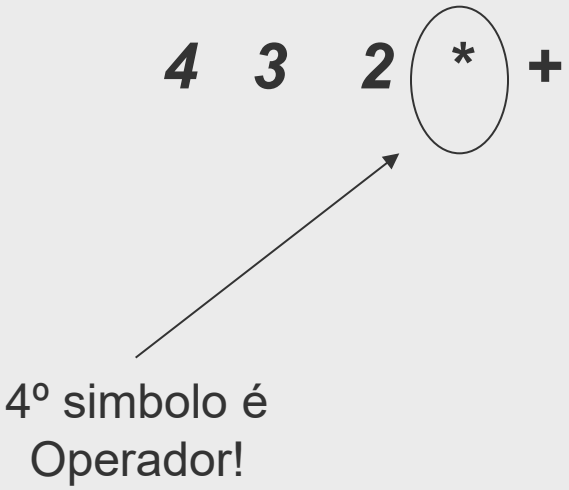
Aplicações de árvores Binárias

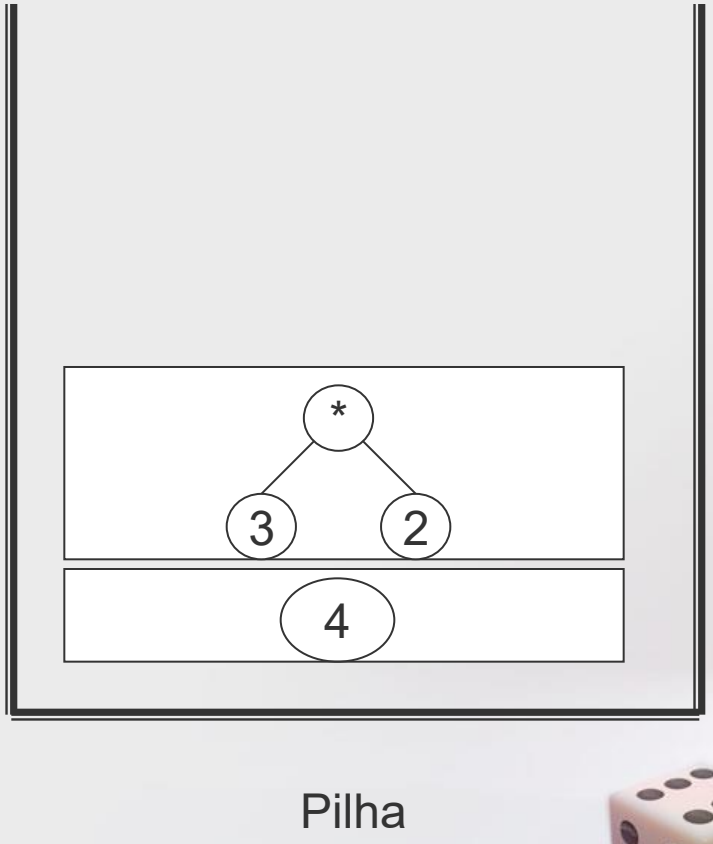
- Exemplo (conversão post-fix para árvore):



Aplicações de árvores Binárias

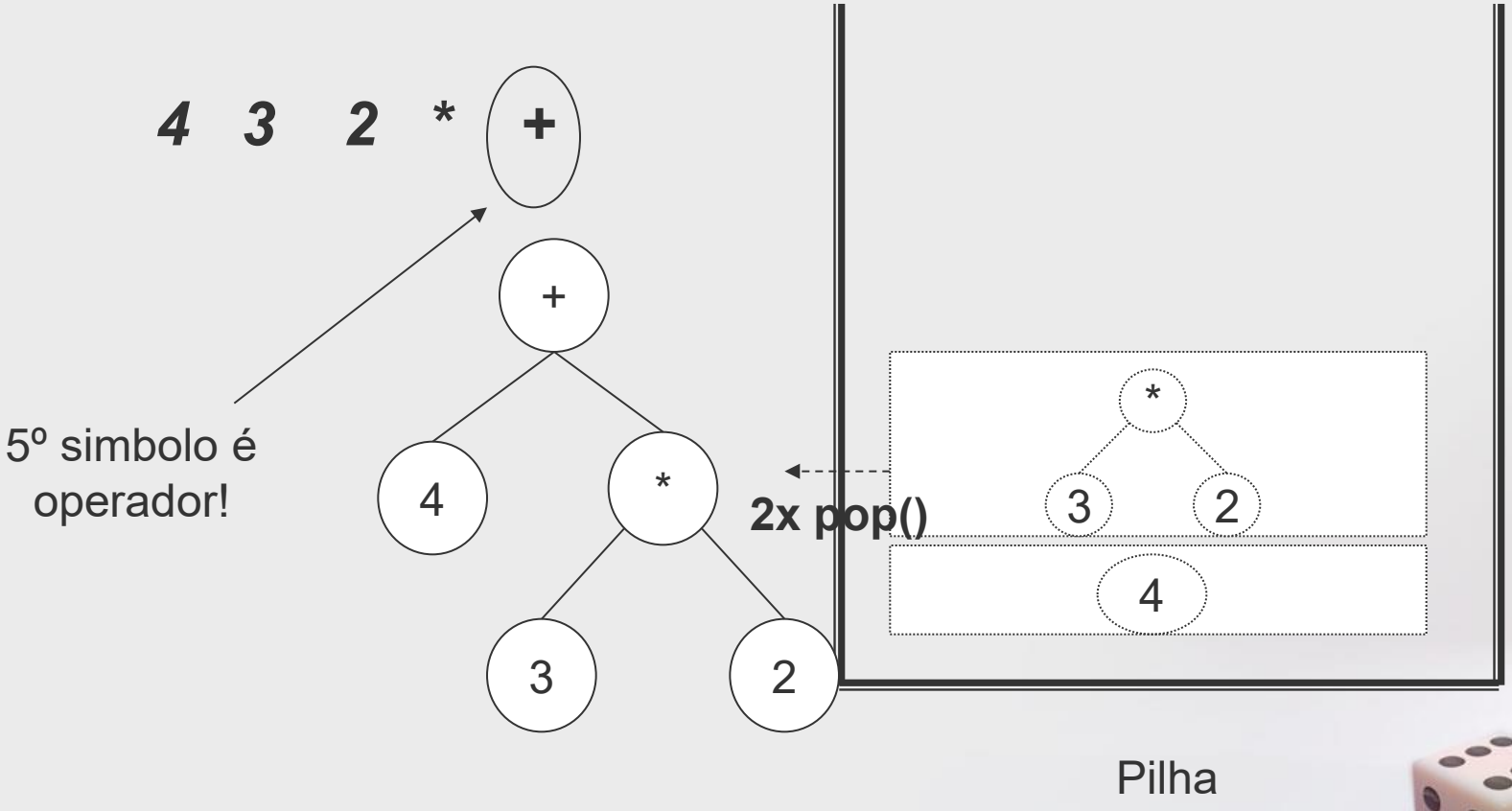
- Exemplo (conversão post-fix para árvore):





Aplicações de árvores Binárias

- Exemplo (conversão post-fix para árvore):

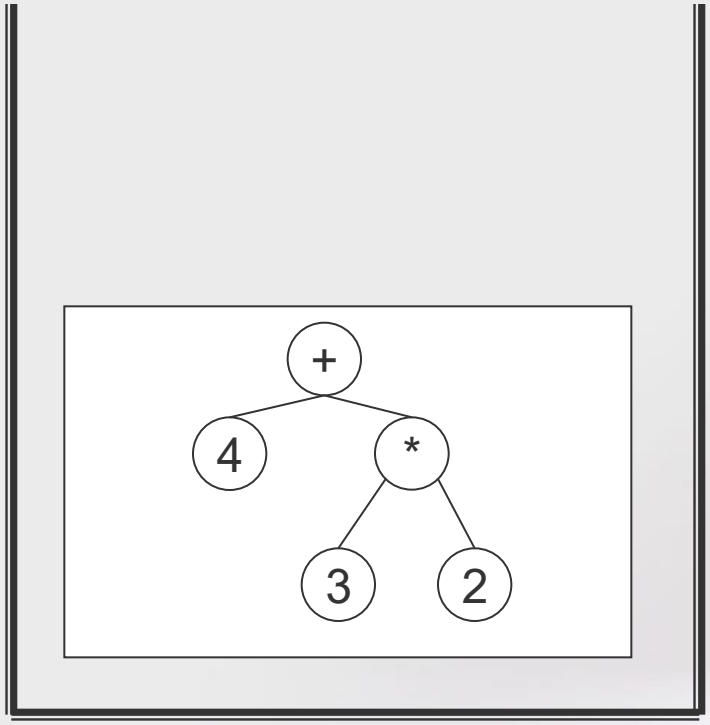


Aplicações de árvores Binárias

- Exemplo (conversão post-fix para árvore):

4 3 2 * +

Acabou!
A árvore está na pilha!



Pilha

Aplicações de árvores Binárias

- E para converter de *infix* para *postfix*??
  - *4+3*2 (infix) é equivalente a 4 3 2 * + (postfix)*
- Note-se que:
  - ***Os operadores aparecem por ordem inversa da sua aplicação*** *(estamos a ignorar as prioridades, vamos depois analisar esse caso)…*
    - *O que sugere de novo que uma pilha pode ser util na conversão*
  - ***Os números aparecem na mesma ordem***

Aplicações de árvores Binárias

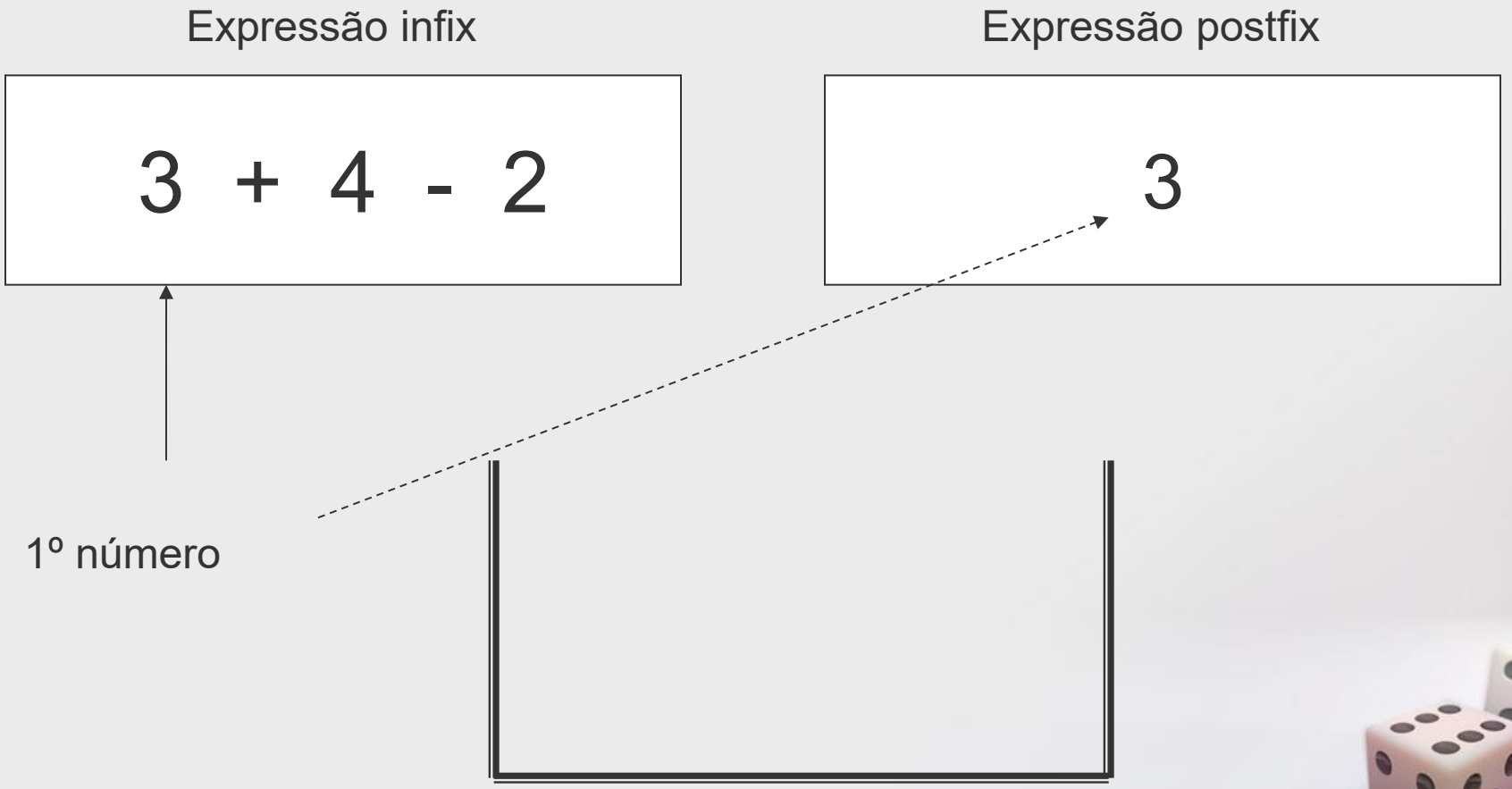
- E para converter de infix para post fix??
  - Tendo em atenção os dois factos apontados, podemos tentar a seguinte versão preliminar:

    - *Percorrendo a expressão:*
      - *Fazer output dos números na ordem que aparecem*
      - *Guardar operadores em pilha*
    - *No final, fazer output dos operadores guardados na pilha.*

  - *Esta versão ainda não está correta: ignora a prioridade dos operadores e regras de associatividade.*

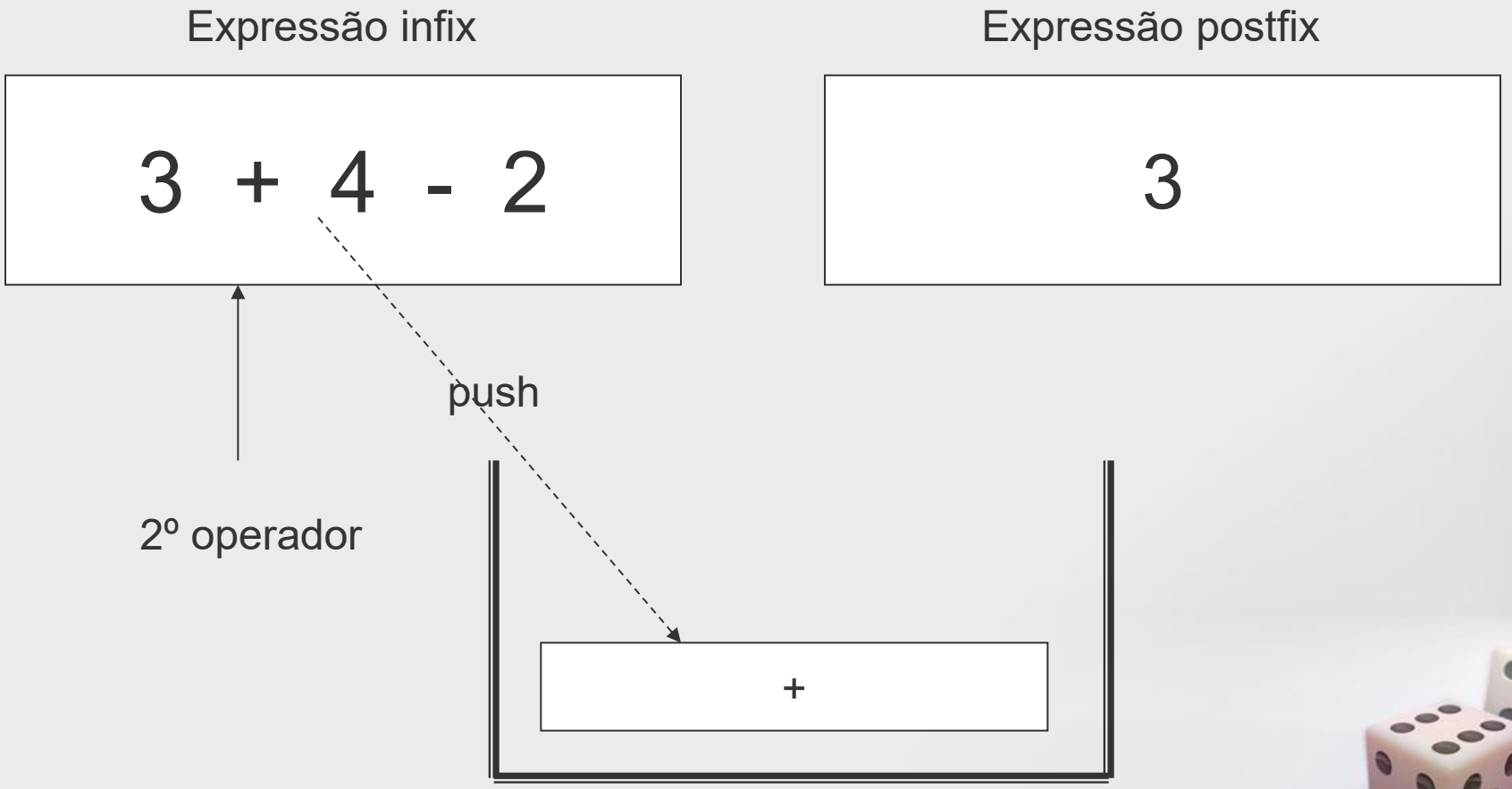
Aplicações de árvores Binárias

- Exemplo 1 (infix para postfix):



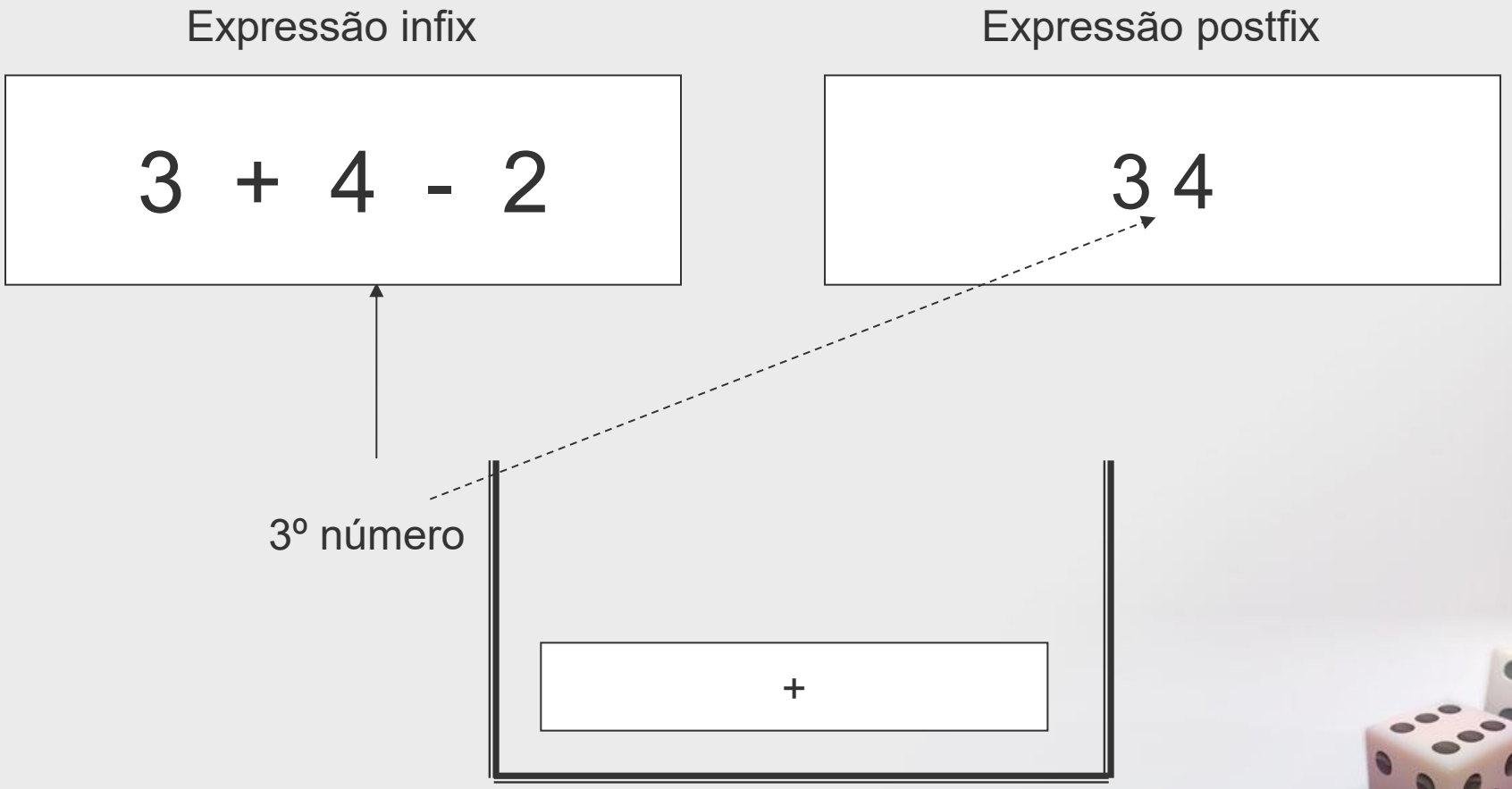
Aplicações de árvores Binárias

- Exemplo 1 (infix para postfix):



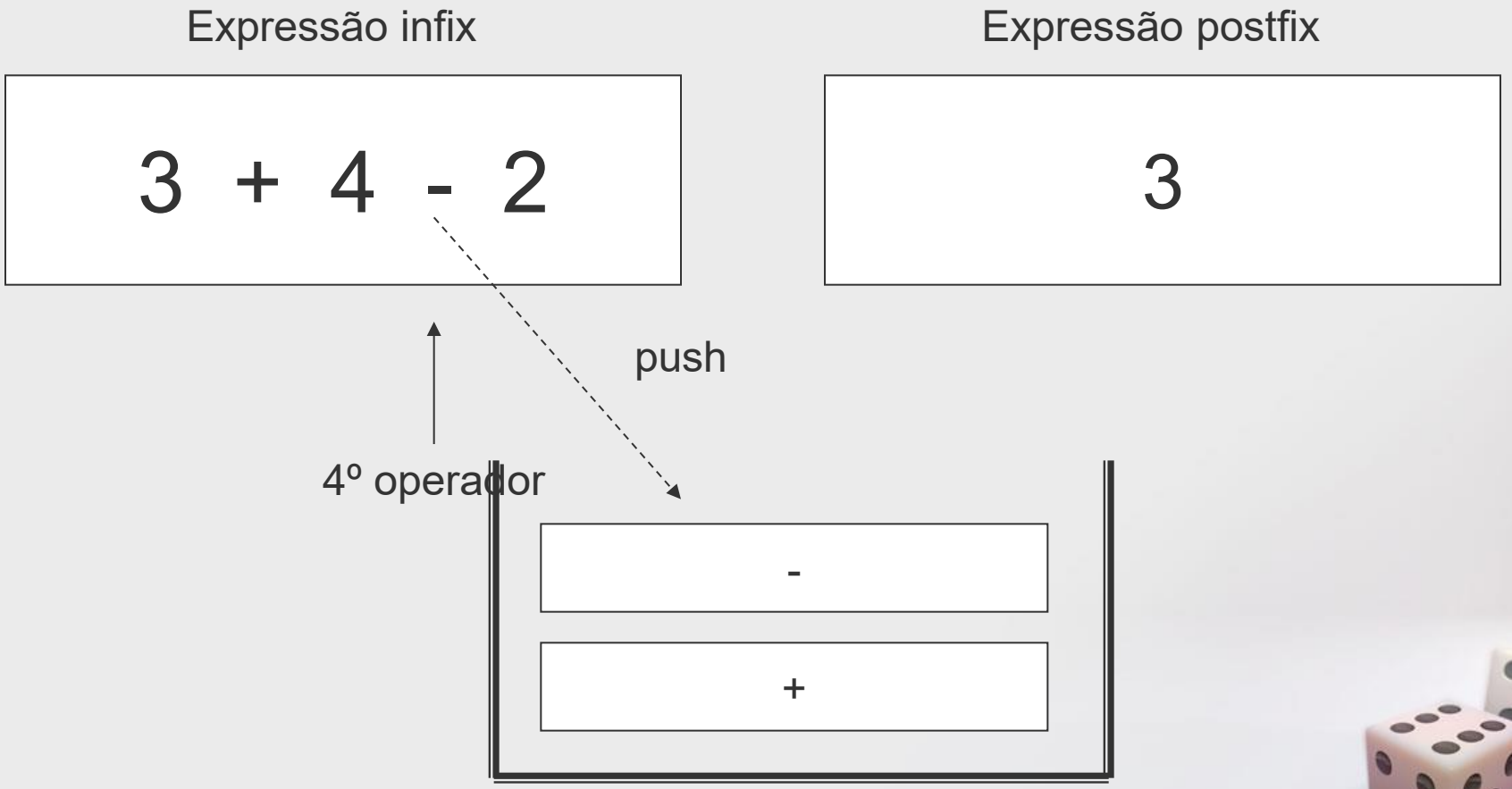
Aplicações de árvores Binárias

- Exemplo 1 (infix para postfix):



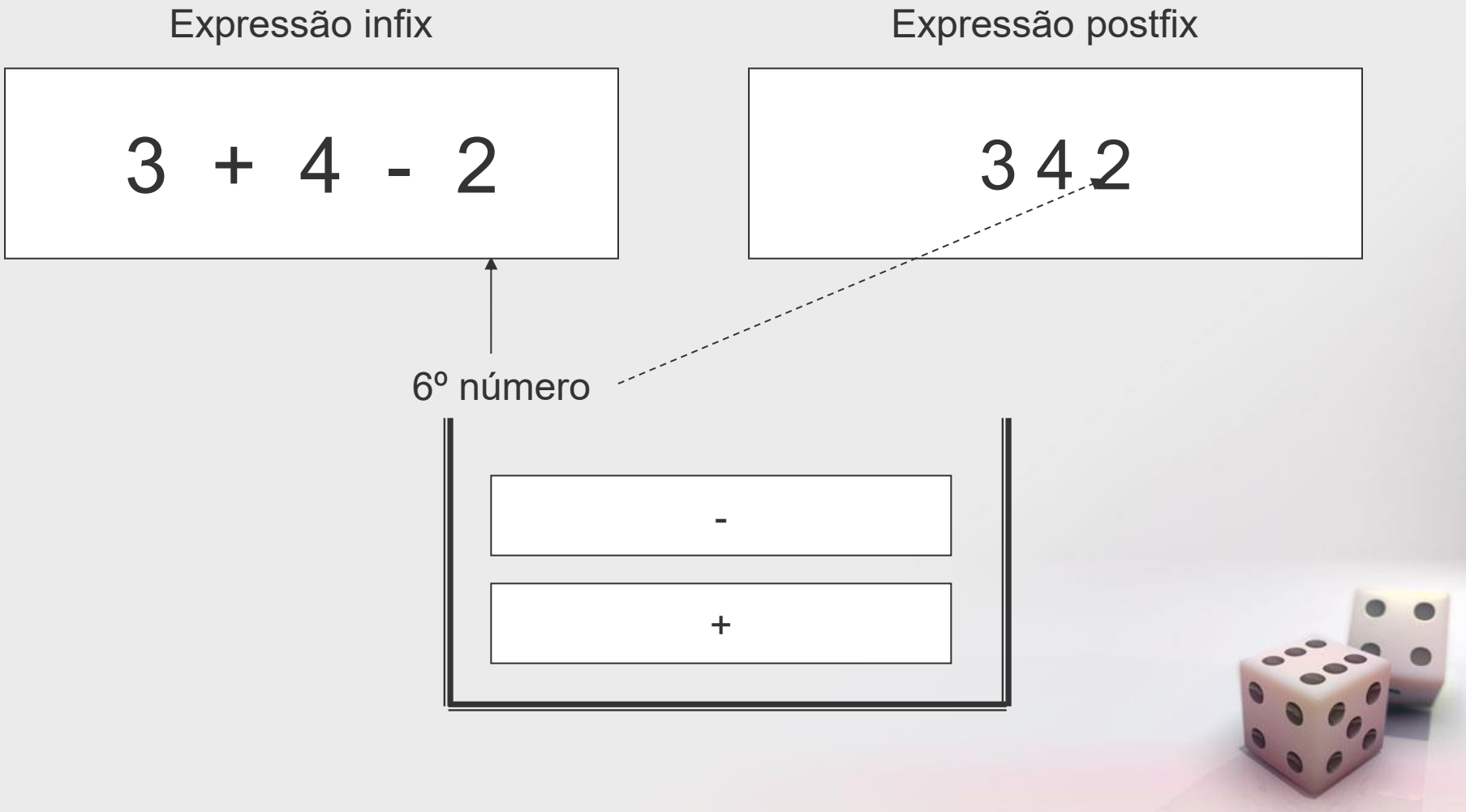
Aplicações de árvores Binárias

- Exemplo 1 (infix para postfix):



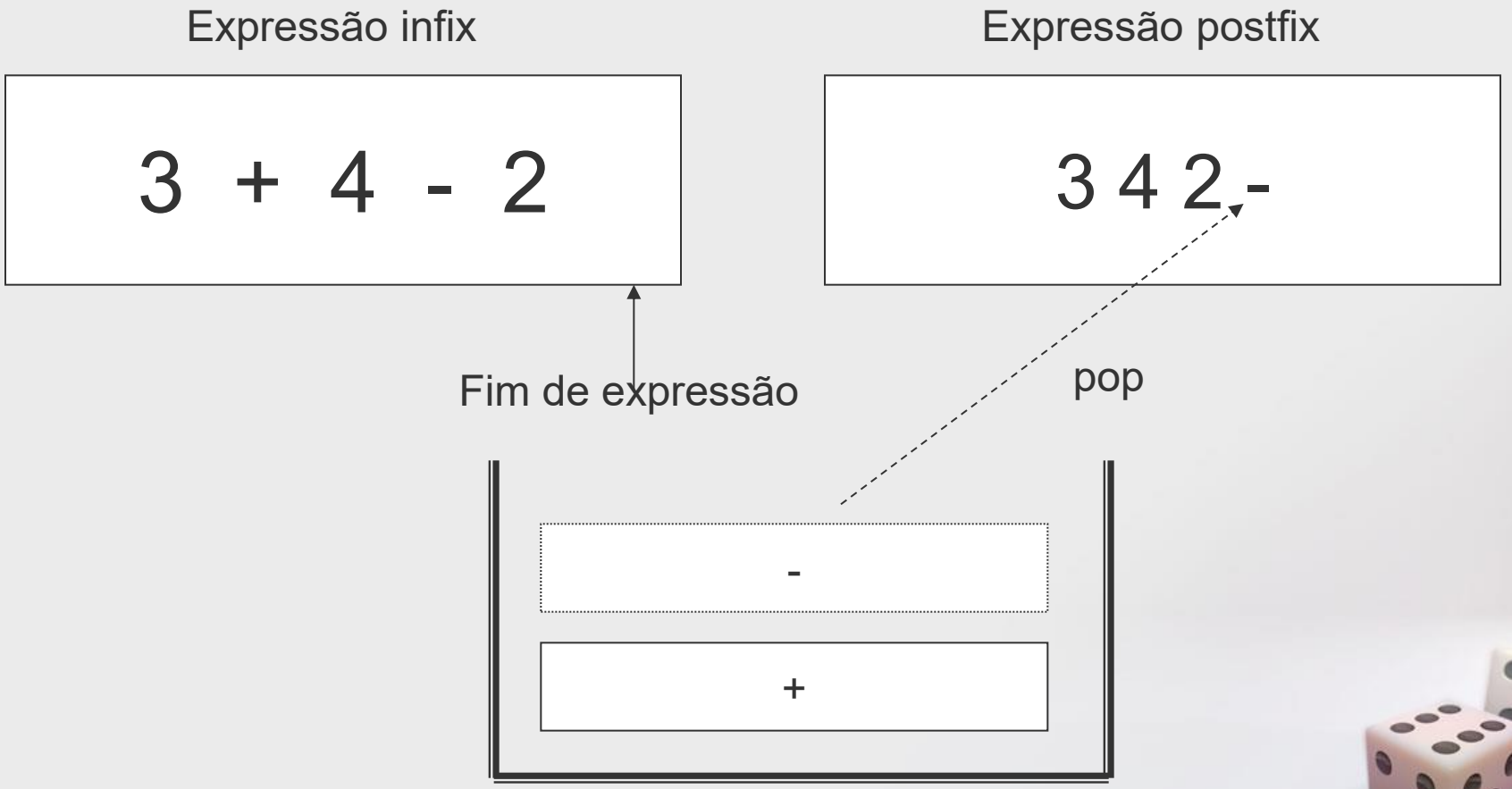
Aplicações de árvores Binárias

- Exemplo 1 (infix para postfix):



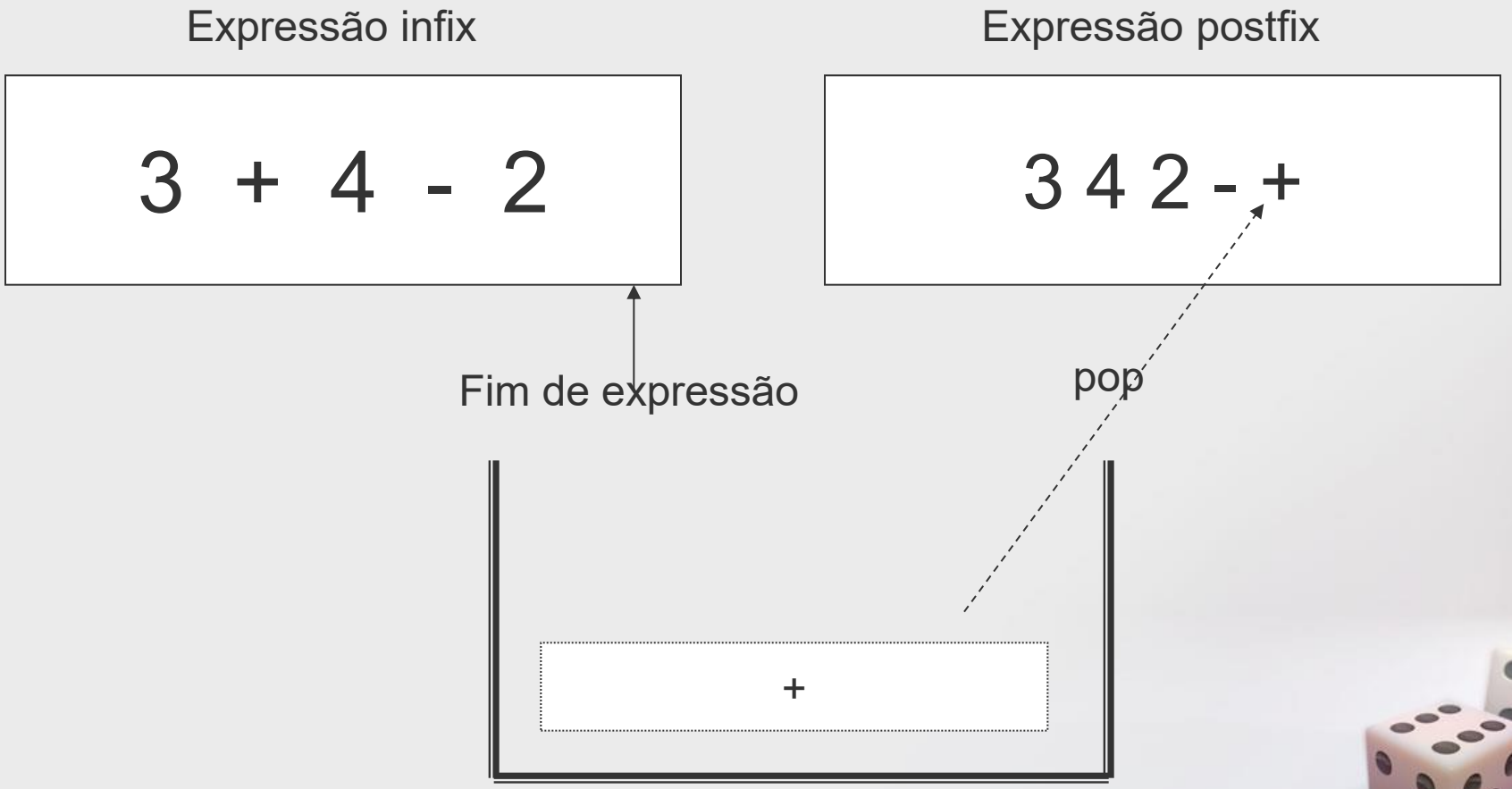
Aplicações de árvores Binárias

- Exemplo 1 (infix para postfix):



Aplicações de árvores Binárias

- Exemplo 1 (infix para postfix):



Aplicações de árvores Binárias

- Conversão de infix para postfix
  - E se os operadores tiverem prioridades diferentes?
  - Vejamos:

2 * 3 + 4 deve ser 2 3 * 4 + e não 2 3 4 + *….

Aplicações de árvores Binárias

- Conversão de infix para postfix
  - Para obter o resultado pretendido, antes de colocar um novo operador na pilha, devemos fazer o *output* dos operadores já na pilha que tenham prioridade superior ao novo operador.

Aplicações de árvores Binárias

- Exemplo 2 (infix para postfix c/ prioridade):

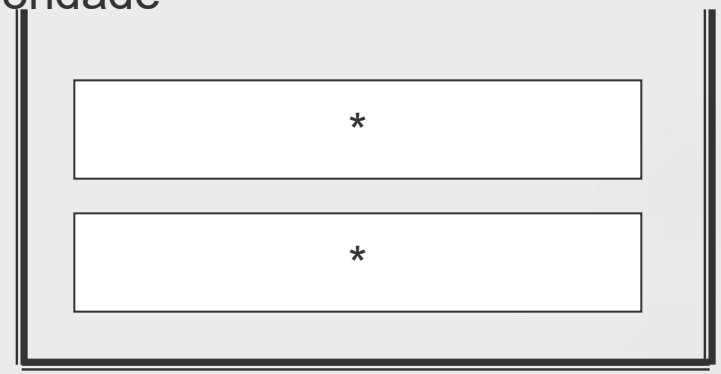
Expressão infix

2 * 3 * 4 - 2

Expressão postfix

2 3 4

Operador de menor prioridade



Aplicações de árvores Binárias

- Exemplo 2 (infix para postfix c/ prioridade):

Expressão infix

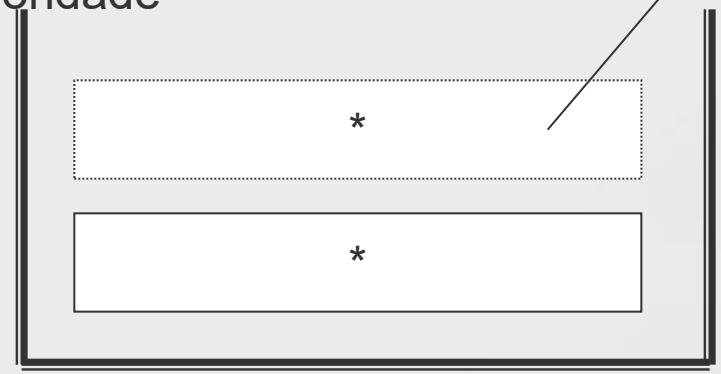
2 * 3 * 4 - 2

Expressão postfix

2 3 4 *

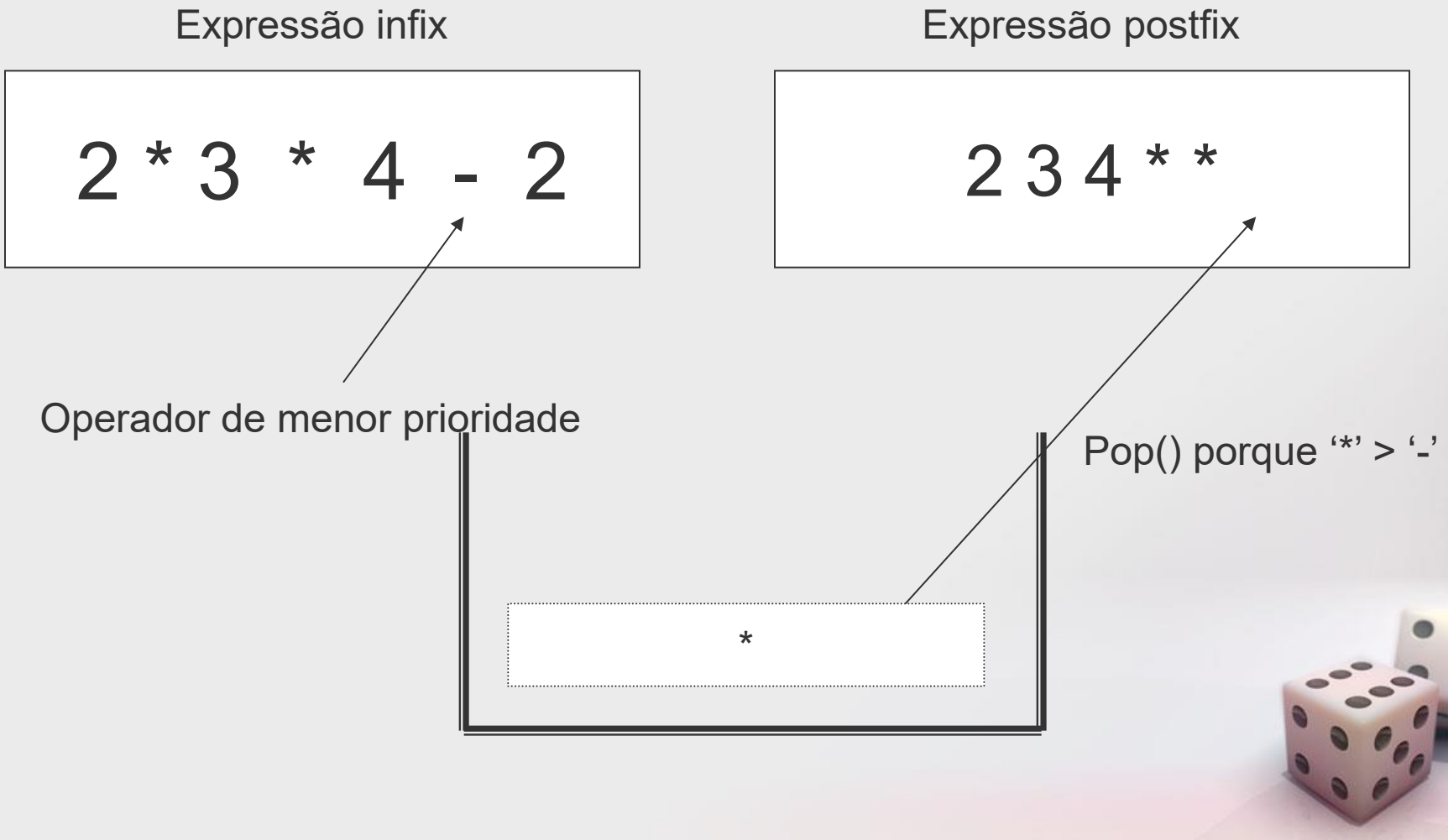
Operador de menor prioridade

Pop() porque '*' > '-'



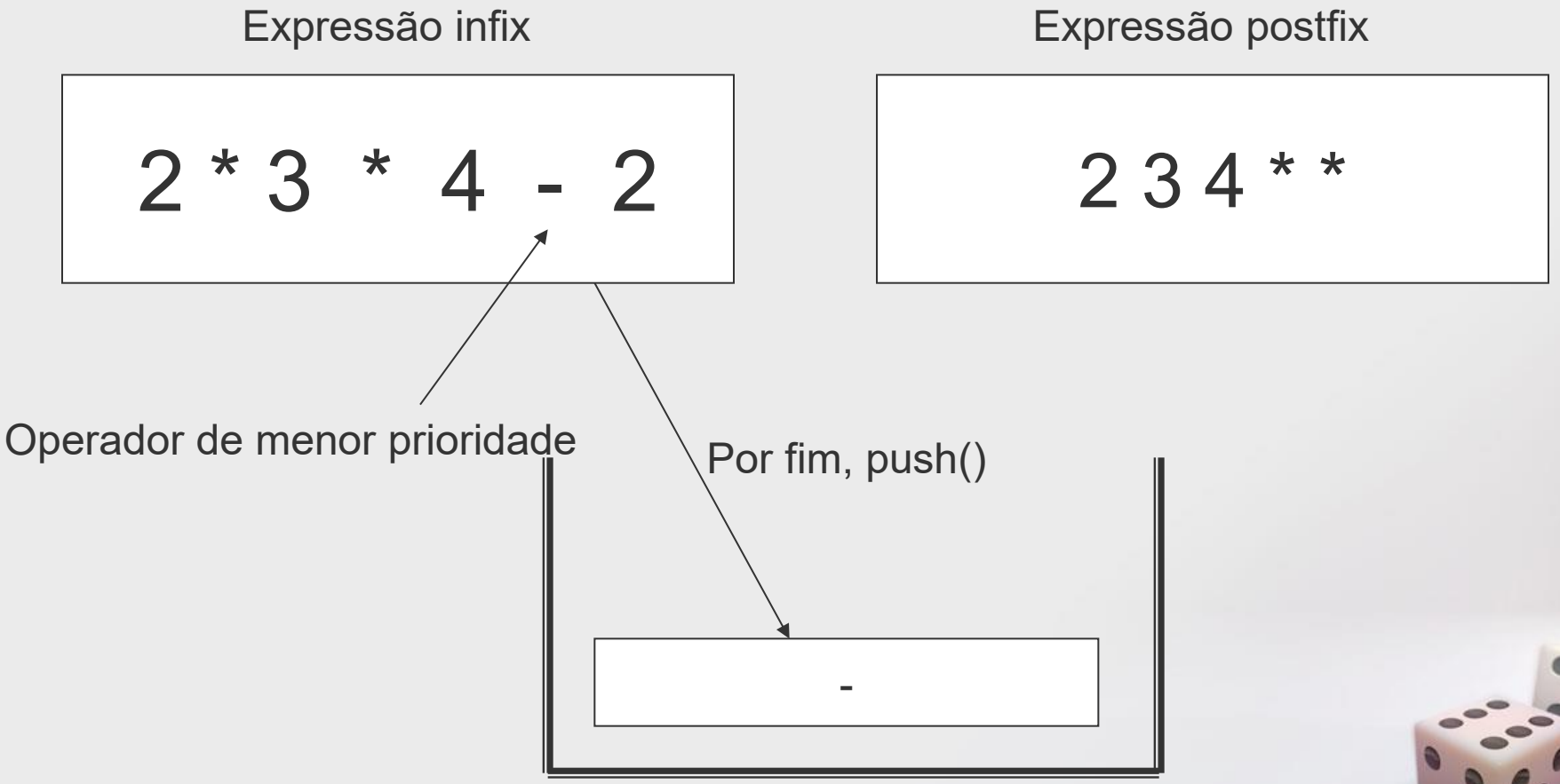
Aplicações de árvores Binárias

- Exemplo 2 (infix para postfix c/ prioridade):



Aplicações de árvores Binárias

- Exemplo 2 (infix para postfix c/ prioridade):



Aplicações de árvores Binárias

- Exemplo 2 (infix para postfix c/ prioridade):

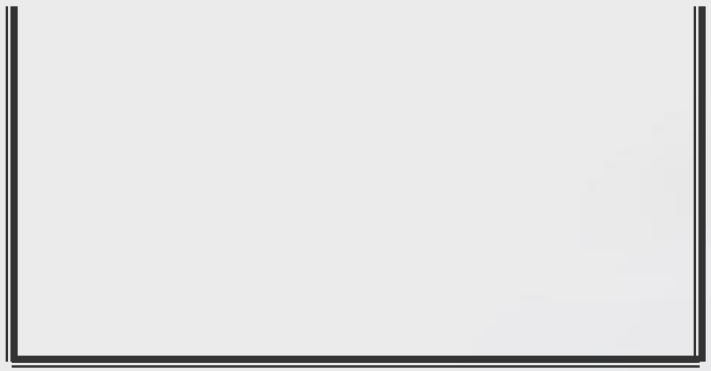
Expressão infix

2 * 3 * 4 - 2

Expressão postfix

2 3 4 * * 2 -

No final



Aplicações de árvores Binárias

- O algoritmo de conversão, incluindo prioridades de operadores, fica:
  - *Percorrendo a expressão:*
    - *Fazer output dos números na ordem que aparecem*
    - *Guardar operadores em pilha*
      - » *Antes de guardar um operador na pilha, fazer output de operadores de prioridade superior que lá estejam.*
  - *No final, fazer output dos operadores guardados na pilha.*

– *Ainda não chega. Esta versão ignora as regras de associatividade. E, já agora, os parentesis…*

3 + 4 + 2 = ((3 + 4) + 2) deve ser 3 4 + 2 + (e não 3 4 2 + +)

Aplicações de árvores Binárias

- **Conversão de infix para postfix**: *Associatividade*
- Para garantir associatividade à esquerda:
  - A prioridade do operador à entrada é considerada menor do que a do mesmo operador na pilha
    - P.Ex: a+b+c=(a+b)+c
      - » Logo 1+2+3 = 1 2 + 3 +
  - Para garantir associatividade à direita:
    - A prioridade do operador à entrada é considerada maior do que a do mesmo operador na pilha:
      - P.Ex: a^b^c=a^(b^c)
        - » Logo 1^2^3=1 2 3 ^ ^

Aplicações de árvores Binárias

- **Conversão de *infix* para *postfix*:** *Parêntesis?*
  - Um parêntesis esquerdo é considerado:
    - um operador de prioridade elevada quando é um símbolo à entrada.
    - Um operador de baixa prioridade quando é um símbolo na pilha.
  - Quando um parêntesis direito é encontrado à entrada, faz-se *output* de todos os operadores na pilha até que seja encontrado o parêntesis esquerdo.

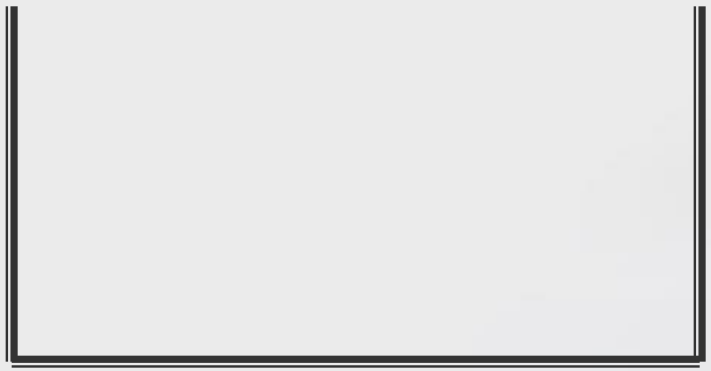
Aplicações de árvores Binárias

- Exemplo Final (*infix* para *postfix*):

Expressão infix

1 + 2 ^ (3 + 4) * 5

Expressão postfix



Aplicações de árvores Binárias

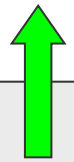
- Exemplo Final (infix para postfix):

Expressão infix

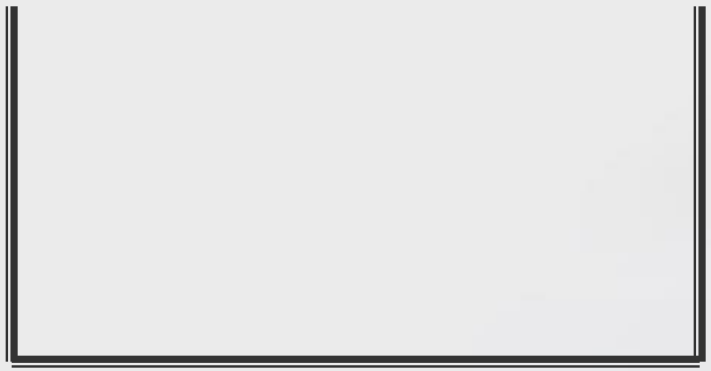
1 + 2 ^ (3 + 4) * 5

Expressão postfix

1



O “1” é enviado para a saída



Aplicações de árvores Binárias

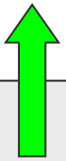
- Exemplo Final (infix para postfix):

Expressão infix

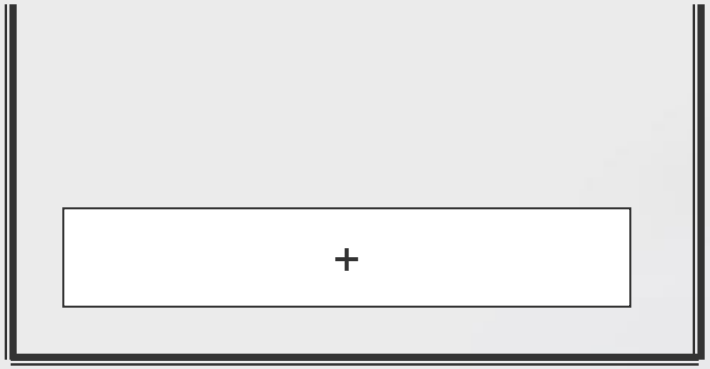
1 + 2 ^ (3 + 4) * 5

Expressão postfix

1



O ‘+’ é enviado para a pilha. Não há lá nenhum outro operador, não é necessário fazer nada.



Aplicações de árvores Binárias

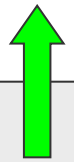
- Exemplo Final (infix para postfix):

Expressão infix

1 + 2 ^ (3 + 4) * 5

Expressão postfix

1 2



O “2” é enviado para a saída

Aplicações de árvores Binárias

- Exemplo Final (infix para postfix):

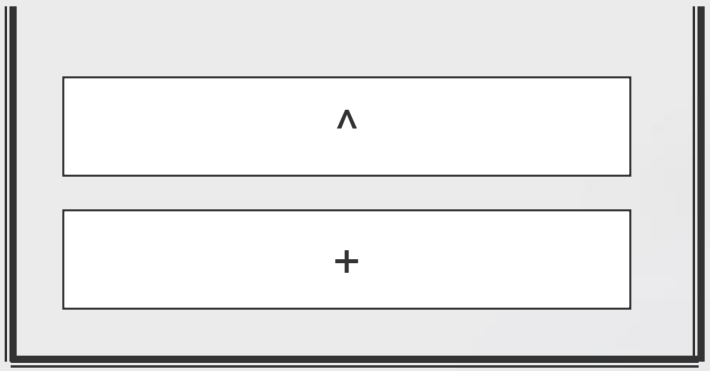
Expressão infix

1 + 2 ^ (3 + 4) * 5

Expressão postfix

1 2

O “>” é colocado na pilha. Como tem prioridade superior ao ‘+’ que já lá está, não acontece nada.

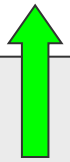


Aplicações de árvores Binárias

- Exemplo Final (infix para postfix):

Expressão infix

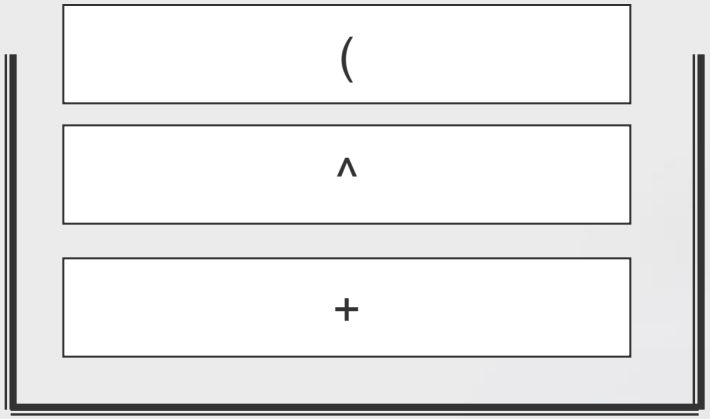
1 + 2 ^ (3 + 4) * 5



Expressão postfix

1 2

O parêntesis esquerdo é colocado na pilha. Quando entra, considera-se sempre que tem mais prioridade do que tudo o que já lá está (por isso nenhum outro operador sai da pilha)



Aplicações de árvores Binárias

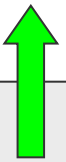
- Exemplo Final (infix para postfix):

Expressão infix

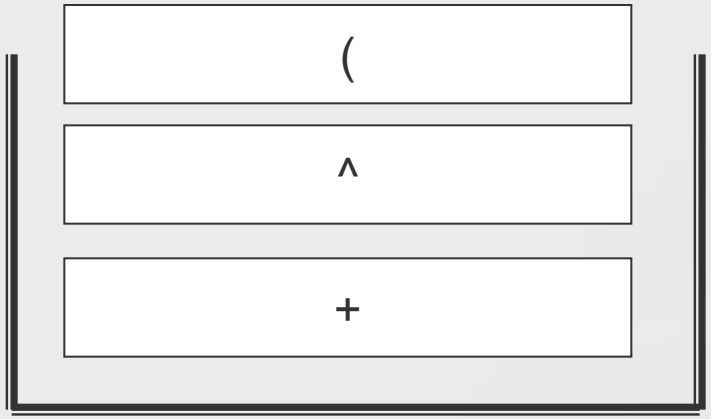
1 + 2 ^ (3 + 4) * 5

Expressão postfix

1 2 3



O “3” é enviado para a saída



Aplicações de árvores Binárias

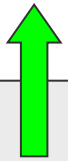
- Exemplo Final (infix para postfix):

Expressão infix

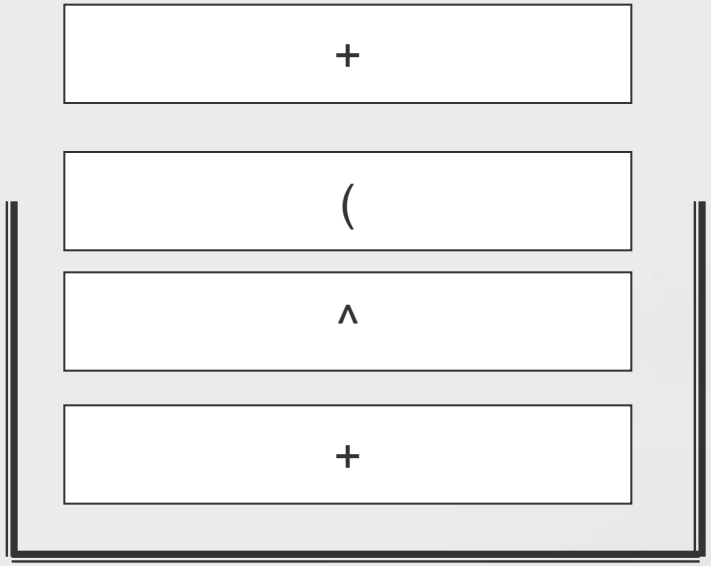
1 + 2 ^ (3 + 4) * 5

Expressão postfix

1 2 3



O “+” é colocado na pilha. Todos os parentesis esquerdos que já se encontram na pilha têm menor prioridade do que qualquer novo operador que lá seja colocado. Por isso nenhum operador é removido da pilha.



Aplicações de árvores Binárias

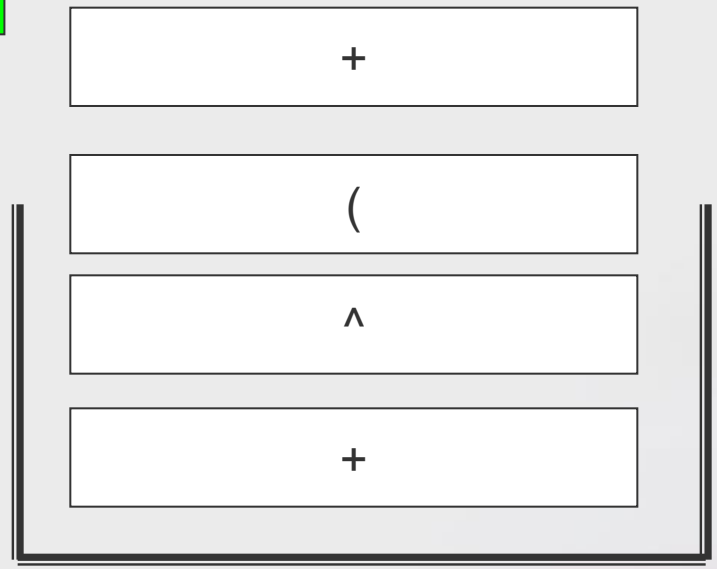
- Exemplo Final (infix para postfix):

Expressão infix

1 + 2 ^ (3 + 4) * 5

Expressão postfix

1 2 3 4



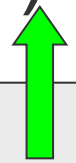
O “4” é enviado para a saída

Aplicações de árvores Binárias

- Exemplo Final (infix para postfix):

Expressão infix

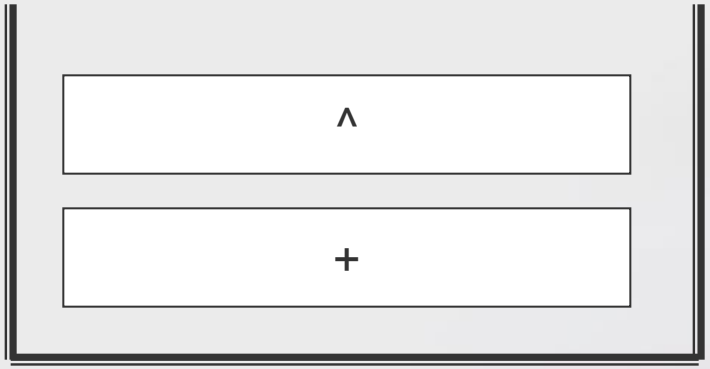
1 + 2 ^ (3 + 4) * 5



Expressão postfix

1 2 3 4 +

Encontrado um parêntesis direito, é feito o *output* de tudo o que está na pilha até ao parêntesis esquerdo.

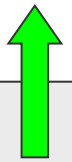


Aplicações de árvores Binárias

- Exemplo Final (infix para postfix):

Expressão infix

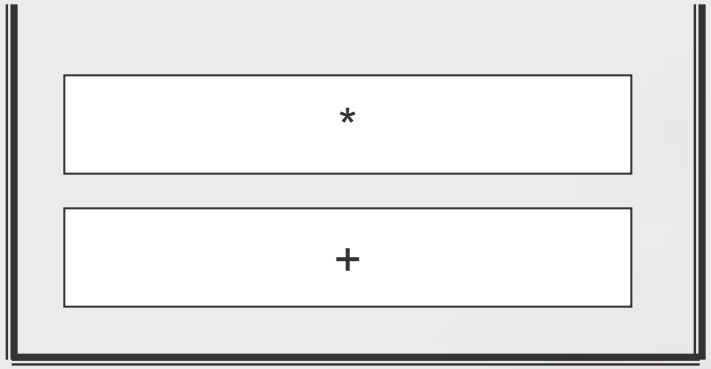
1 + 2 ^ (3 + 4) * 5



Expressão postfix

1 2 3 4 + ^

Antes de colocar o '*' na pilha, o '^' que está no topo, de prioridade superior, é enviado para a saída.

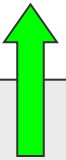


Aplicações de árvores Binárias

- Exemplo Final (infix para postfix):

Expressão infix

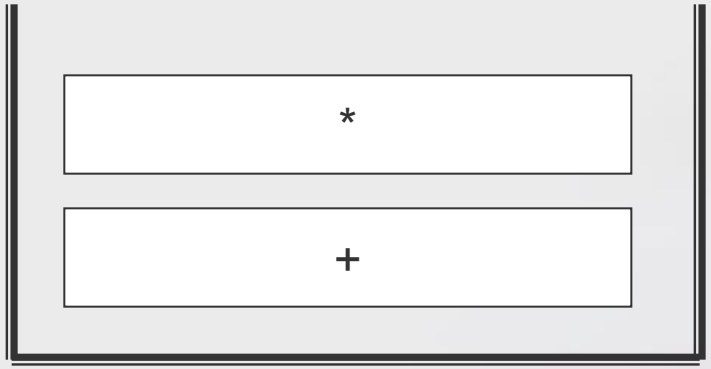
1 + 2 ^ (3 + 4) * 5



Expressão postfix

1 2 3 4 + ^ 5

O “5” é enviado para a saída



Aplicações de árvores Binárias

- Exemplo Final (*infix* para *postfix*):

Expressão *infix*

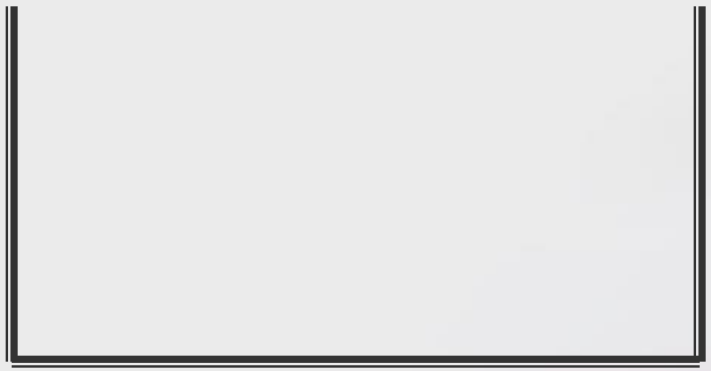
1 + 2 ^ (3 + 4) * 5

Expressão *postfix*

1 2 3 4 + ^ 5 * +

Atingido o final da expressão todo o conteúdo ainda na pilha é enviado para a saída.

Final!



Aplicações de árvores Binárias

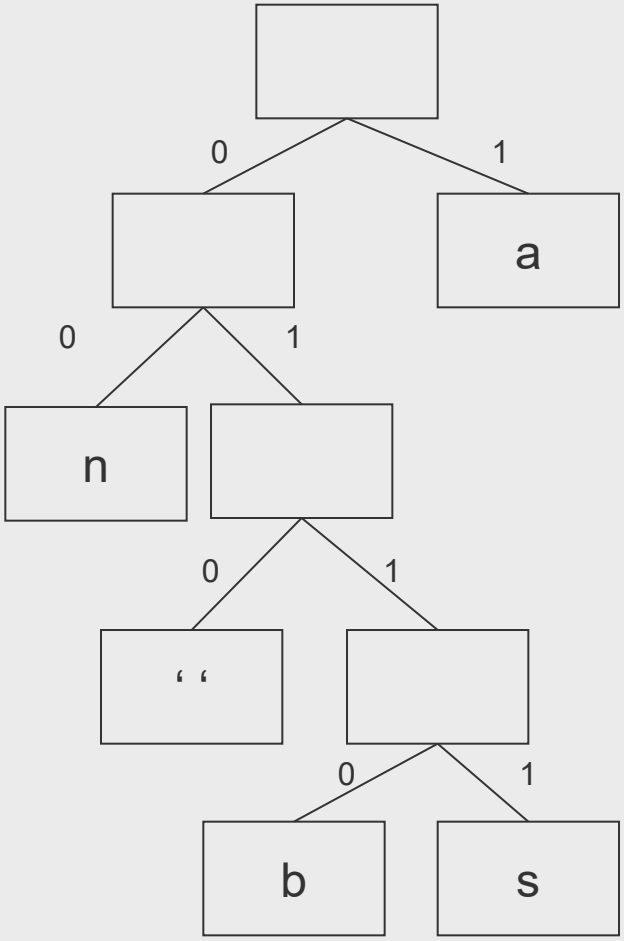
- O algoritmo de conversão, incluindo prioridades de operadores, associatividade e parêntesis, fica:

  - *Percorrendo a expressão:*
    - *Fazer output dos números na ordem que aparecem*
    - *Guardar operadores em pilha*
      - » *Antes de guardar um operador (não um "(") na pilha, fazer output de operadores de prioridade superior que lá estejam. Um parentêsis esquerdo na pilha terá sempre prioridade inferior ao operador novo que lá é colocado.*
      - » *No caso de empate de prioridade entre o novo elemento e o topo da pilha, aumentar ou diminuir prioridade do novo elemento para obter associatividade à direita ou esquerda, respectivamente.*
    - *Guardar sempre parêntesis esquerdos ("(") na pilha.*

  No final, fazer output dos operadores ainda guardados na pilha.

Aplicações de árvores Binárias

- Compressão de dados (árvore de Huffman):



Texto a comprimir:

“banana ananas”

‘a’ (6 ocorrências) =1
‘n’ (4 ocorrências)=00
‘s’ (1 ocorrência) =0111
‘b’ (1 ocorrência) =0110
‘ ‘ (1 ocorrência)=010

Texto recodificado:

“011010010010101001001011 1”

Aplicações de árvores Binárias

- Codificação de Huffman:
  - Como criar uma árvore que minimize o número de bits necessários?
    - Os símbolos mais frequentes devem ter uma representação mais curta.
      - Necessáriamente, os símbolos menos comuns terão uma representação mais longa.

Aplicações de Árvores Binárias

- Algoritmo de Huffman
  - A árvore é construída de baixo para cima, agregando subárvores de acordo com a frequência dos respectivos símbolos.
    - Devem sempre ser agregadas as sub-árvores com menos símbolos
    - Os empates são resolvidos arbitrariamente.

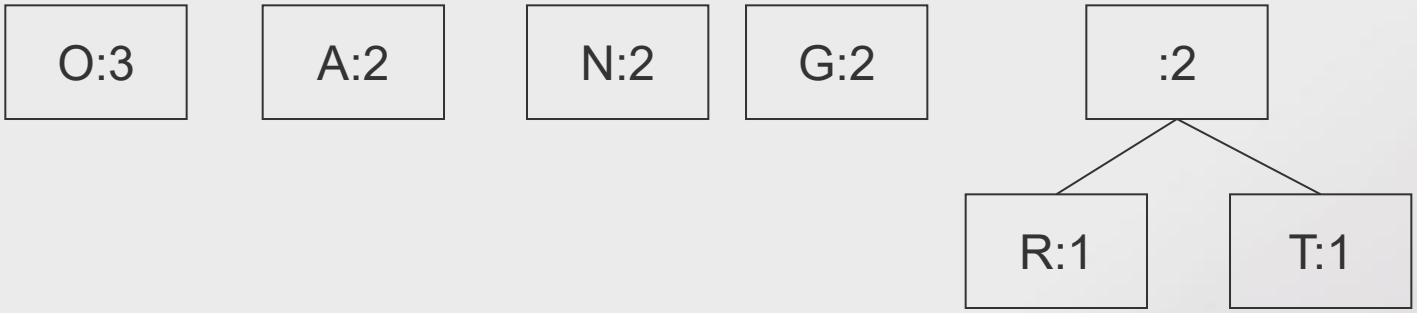
Aplicações de Árvores Binárias

- Exemplo:“ORANGOTANGO”
  - Calcula-se a frequência das letras

O:3	A:2	N:2	G:2	R:1	T:1

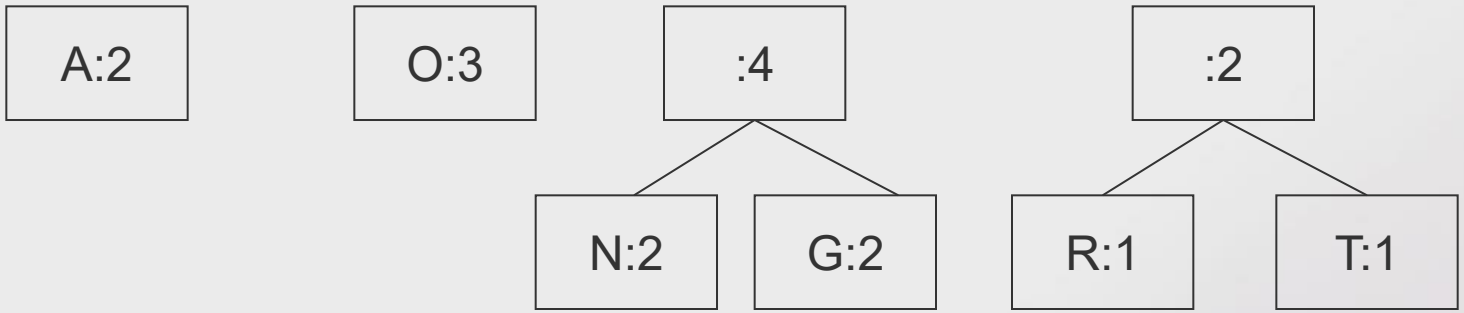
Aplicações de Árvores Binárias

- Exemplo:“ORANGOTANGO”
  - Agrupam-se dois dos elementos com menor frequência numa única árvore



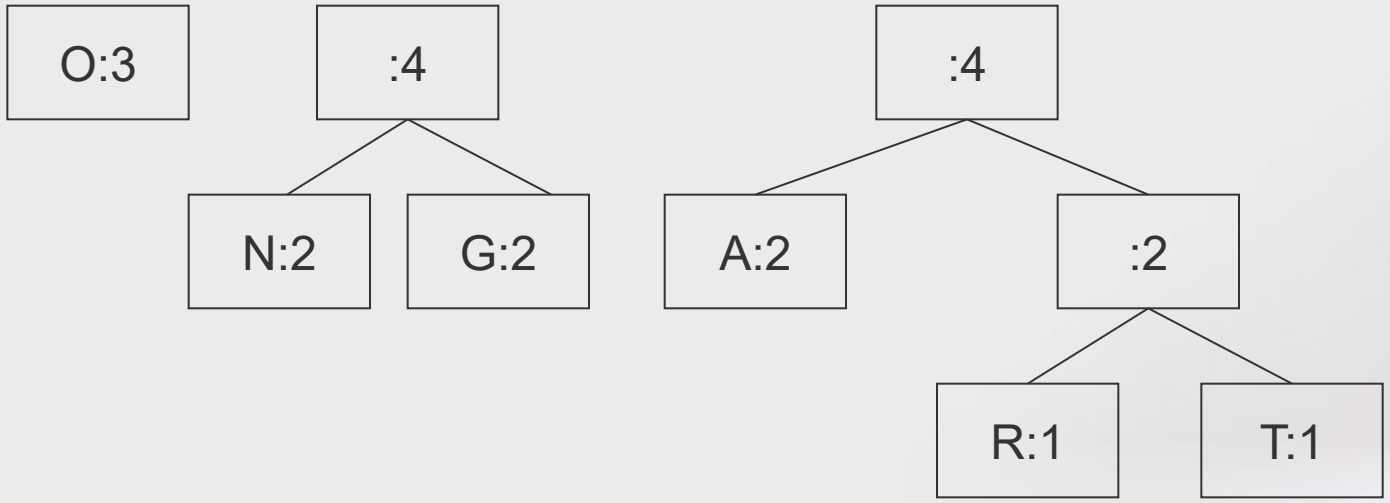
Aplicações de Árvores Binárias

- Exemplo:“ORANGOTANGO”
  - Repete-se o processo. Neste caso, há 4 sub-árvores com apenas dois elementos: {*A*, *N*, *G*, e a sub-árvore que acabámos de construir}
    - Podem ser agrupadas quaisquer duas destas sub-árvores:
    - Vamos escolher, arbitrariamente, *N* e *G*.



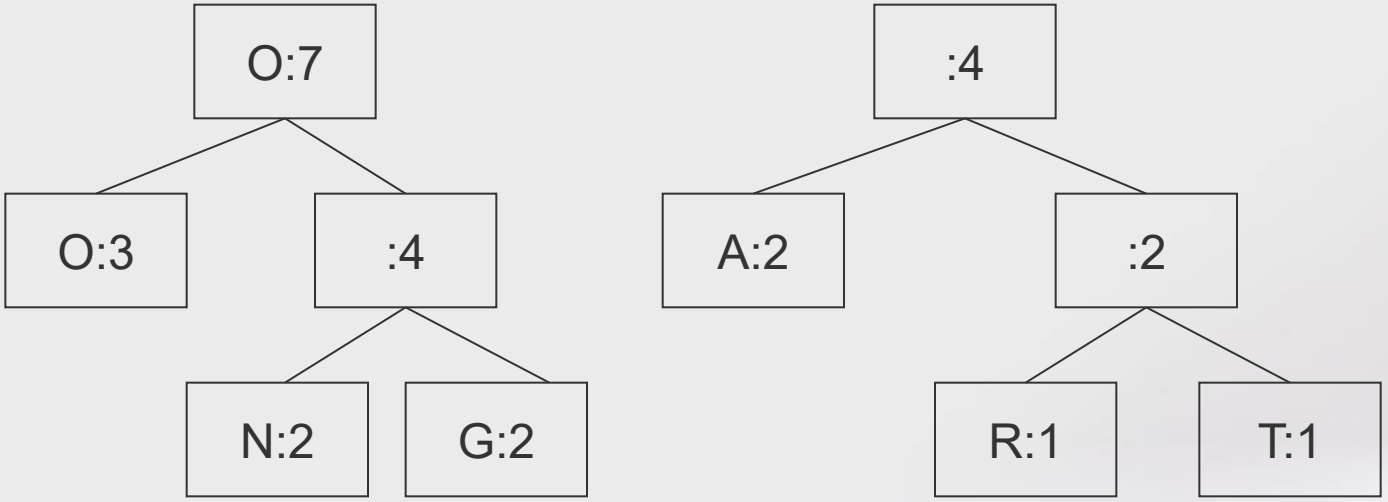
Aplicações de Árvores Binárias

- Exemplo:“ORANGOTANGO”
  - Repete-se o processo. Agora as sub-árvores menores são “A” e a primeira que construimos.



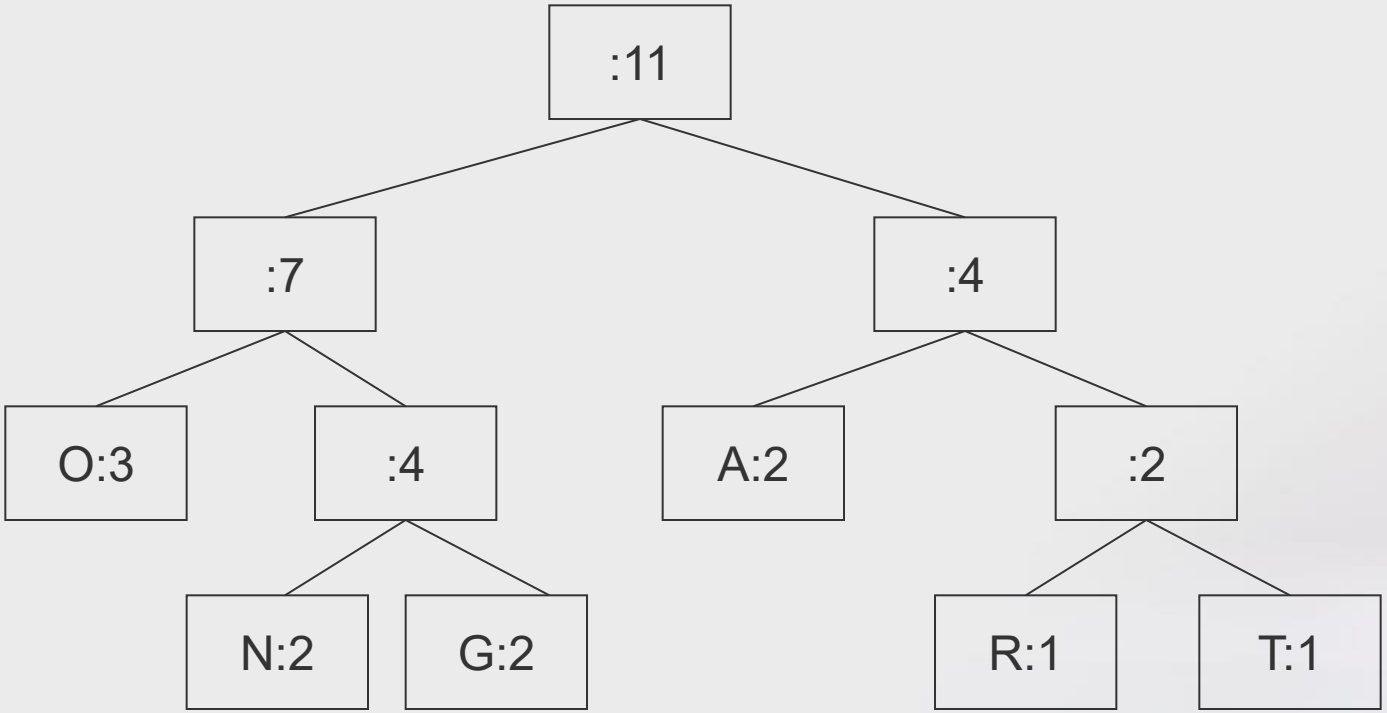
Aplicações de Árvores Binárias

- Exemplo:“ORANGOTANGO”
  - Repete-se o processo…



Aplicações de Árvores Binárias

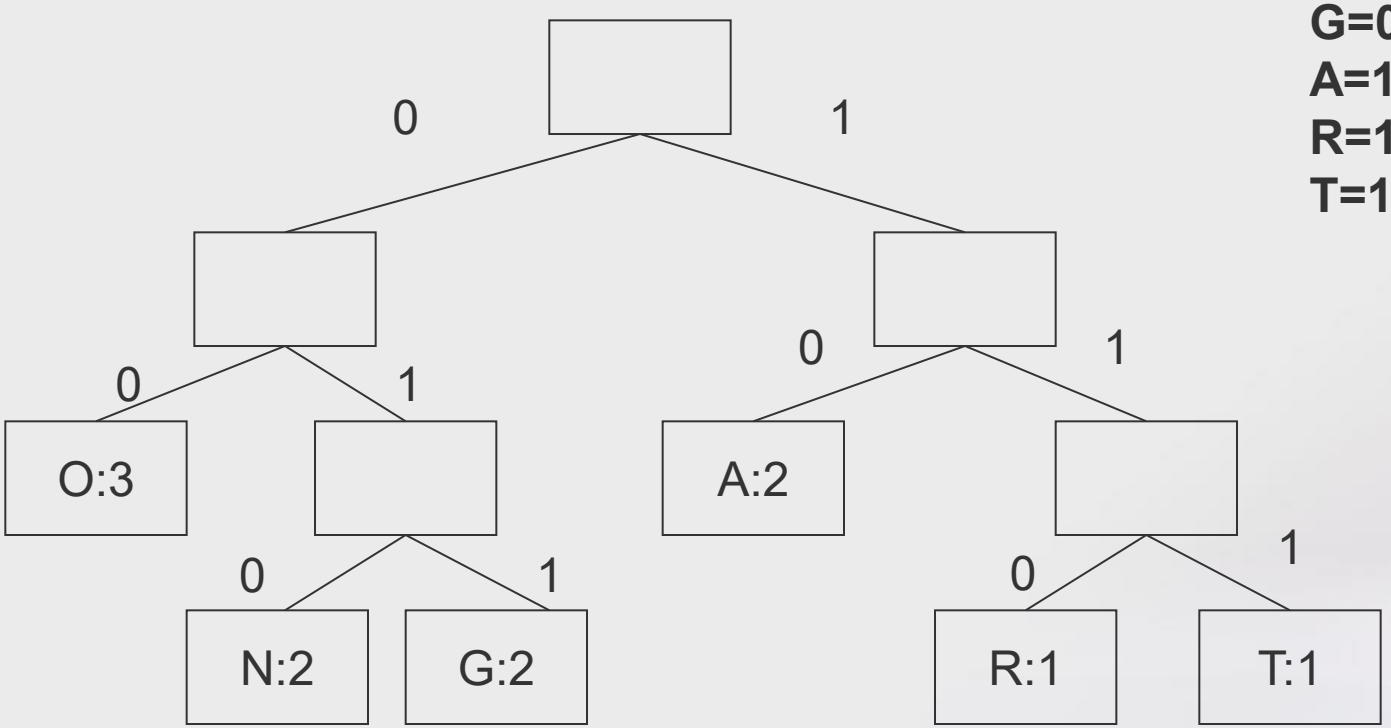
- Exemplo:“ORANGOTANGO”
  - Repete-se o processo…



Aplicações de Árvores Binárias

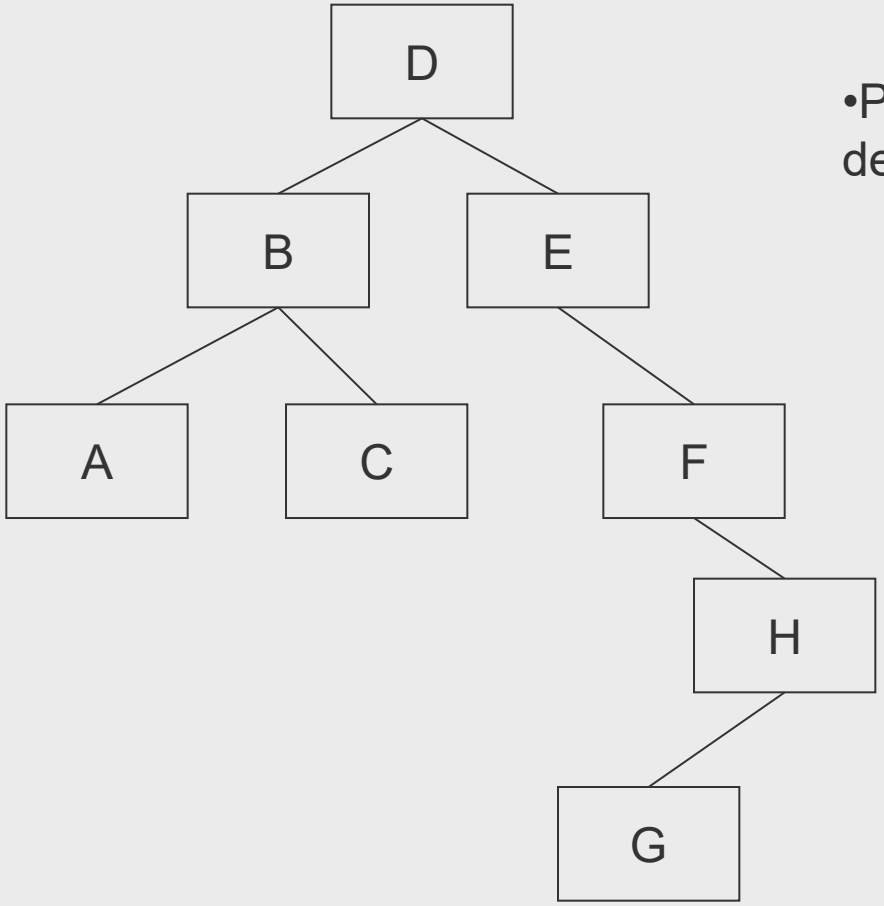
- Exemplo:“ORANGOTANGO”
  - Por fim, obtém-se a codificação

O=00
N=010
G=011
A=10
R=110
T=111



Aplicações de árvores Binárias

- Árvores de Pesquisa Binárias:

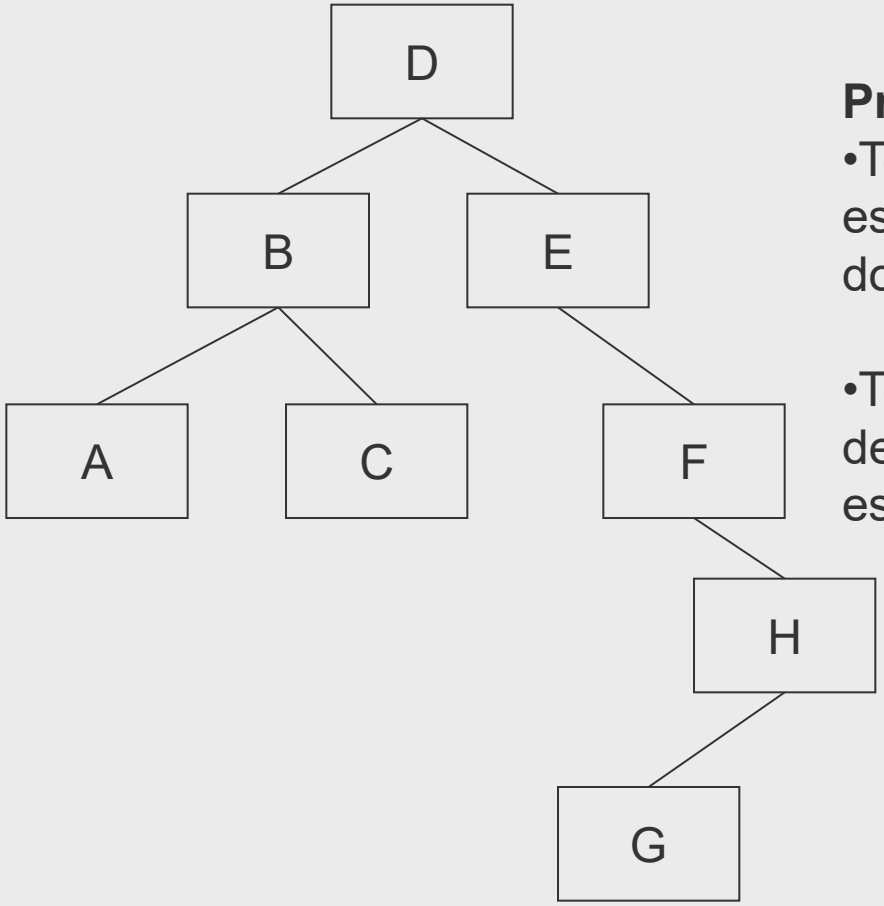


•Podem permitir a inserção e pesquisa de informação em tempo logaritmico:

-Podem ser usadas para implementar diferentes tipos de estruturas de dados,

Aplicações de árvores Binárias

- Árvores de Pesquisa Binárias:



Propriedade de Ordem

•Todos os nodos na sub-árvore esquerda de um nodo raiz são menores do que a essa raiz.

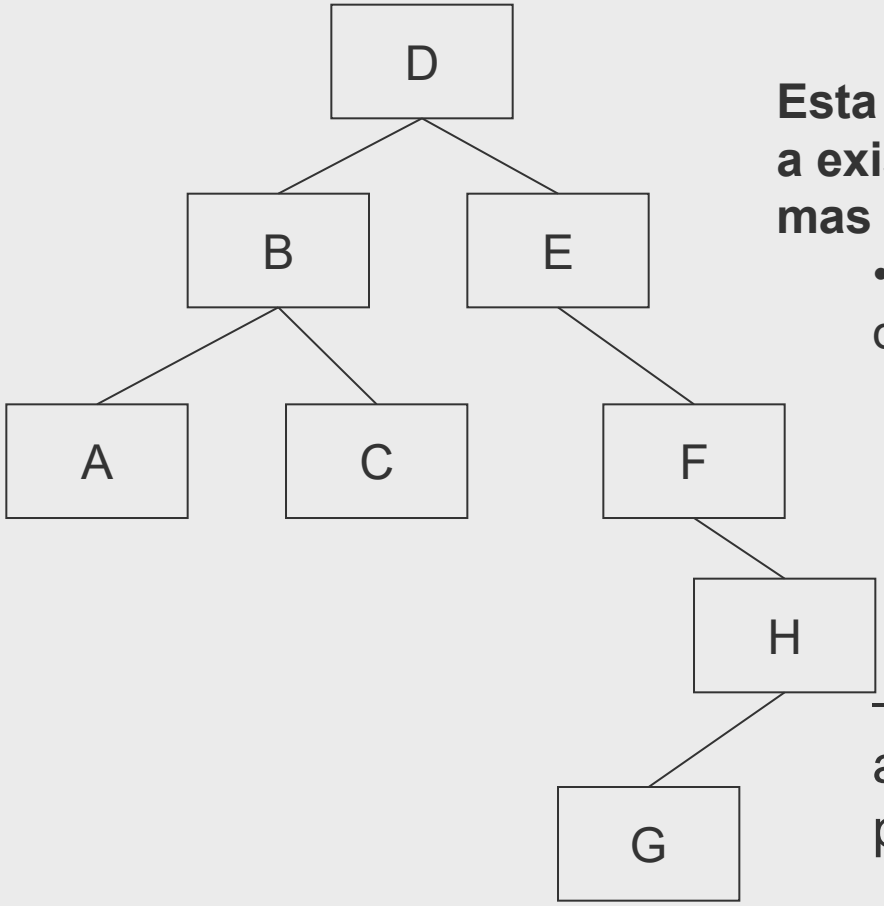
•P.ex: D > A,B e C

•Todos os nodos na sub-árvore direita de um nodo raiz são maiores do que essa raiz.

•P.ex: B < C

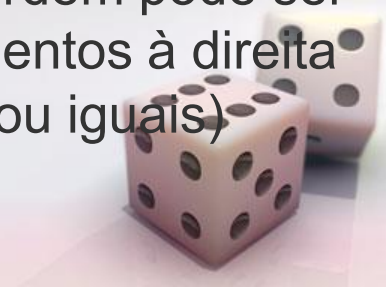
Aplicações de árvores Binárias

- Árvores de Pesquisa Binárias:



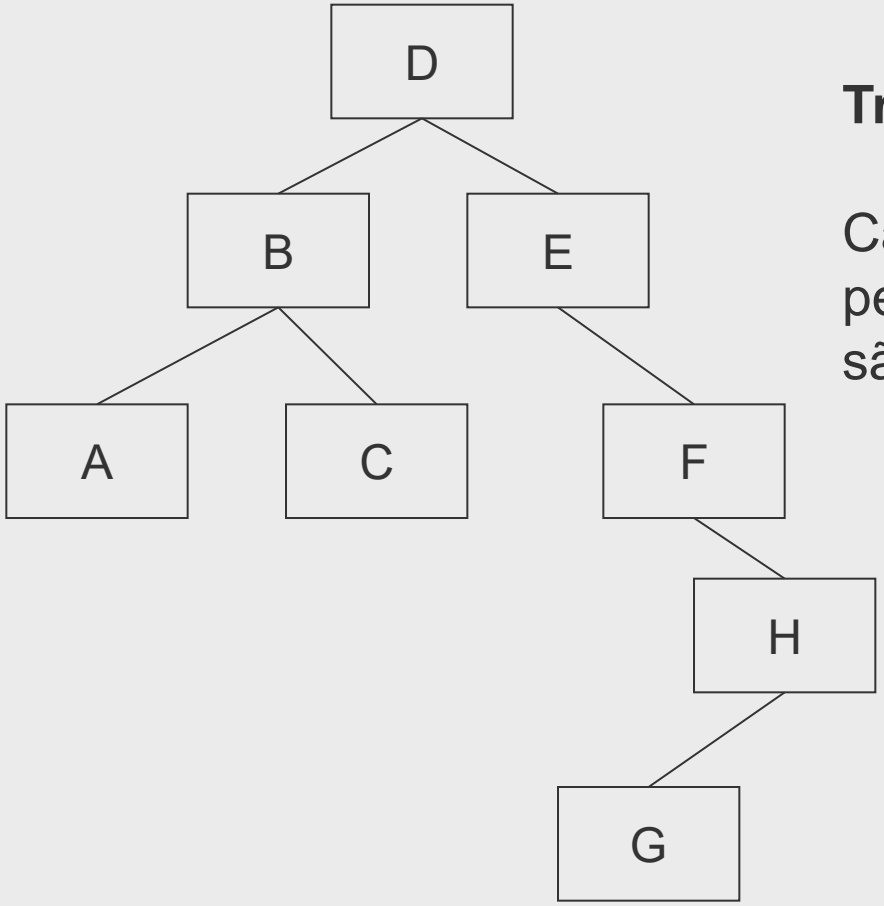
Esta propriedade de ordem não permite a existência de elementos duplicados, mas isso é facilmente ultrapassável:

- Pode ser usada uma estrutura de dados secundária em cada nodo. P.ex:
  - Em cada nodo pode existir uma Lista com os elementos iguais (em termos de ordem)
  - Caso os elementos sejam estritamente iguais, pode ser suficiente armazenar uma contagem.
- A propriedade de ordem pode ser ajustada (p.ex. elementos à direita podem ser maiores ou iguais)



Aplicações de árvores Binárias

- Árvores de Pesquisa Binárias:

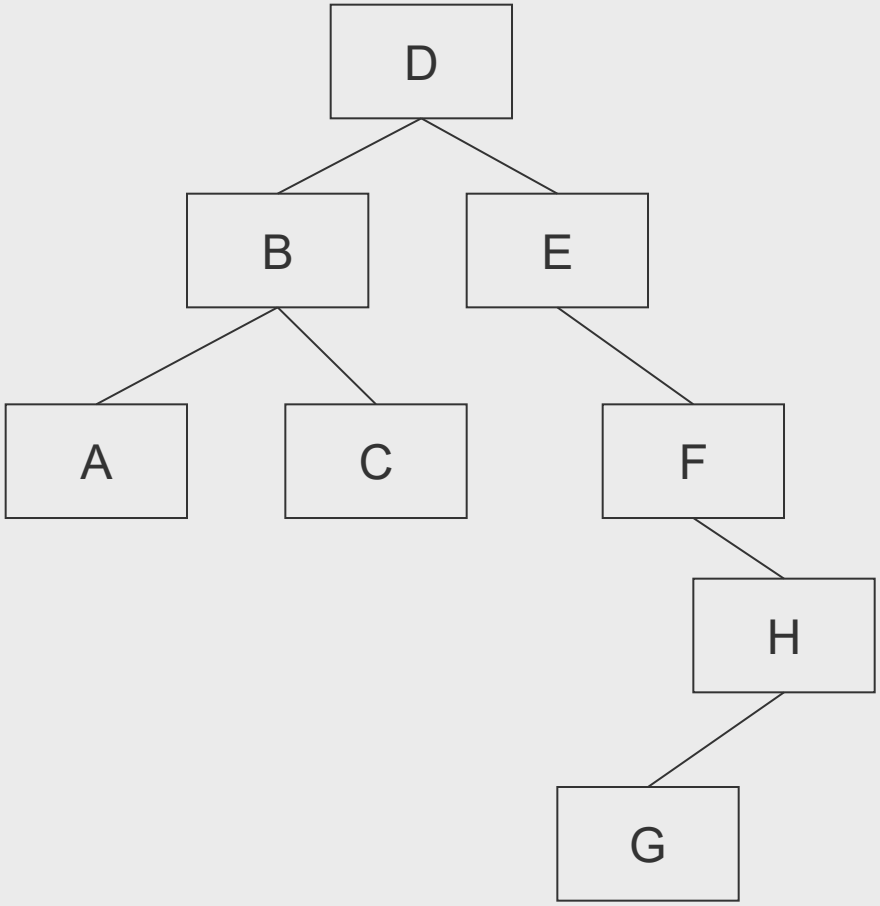


Travessia de árvore de pesquisa

Caso uma árvore de pesquisa seja percorrida em ordem, os elementos são visitados por ordem crescente.

Aplicações de árvores Binárias

- Árvores de Pesquisa Binárias:

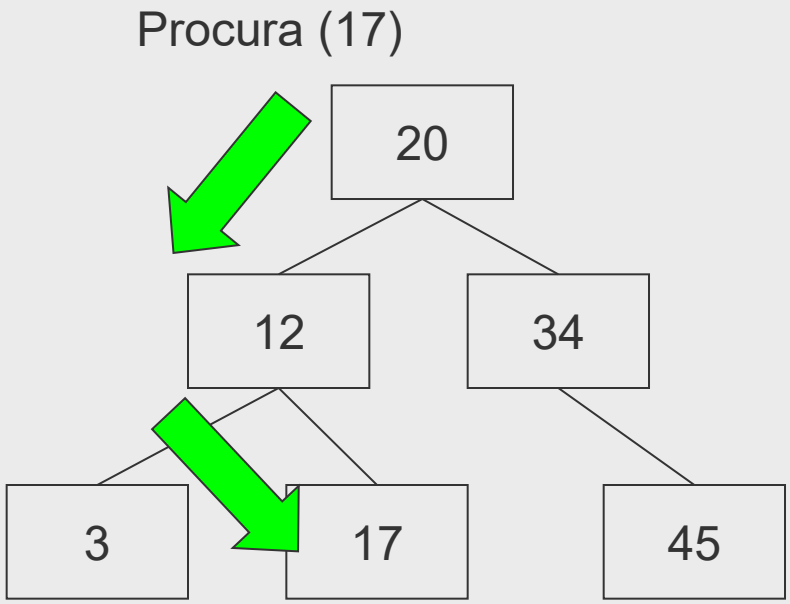


Pesquisa de um valor em árvores binárias de pesquisa

Para procurar o valor *X* numa árvore binária com raiz *R*, verifica-se *R* contém *X*. Caso isso não aconteça, procura-se na sub-árvore esquerda ou direita conforme o elemento a procurar seja menor ou maior, respectivamente, do que o valor da raiz.

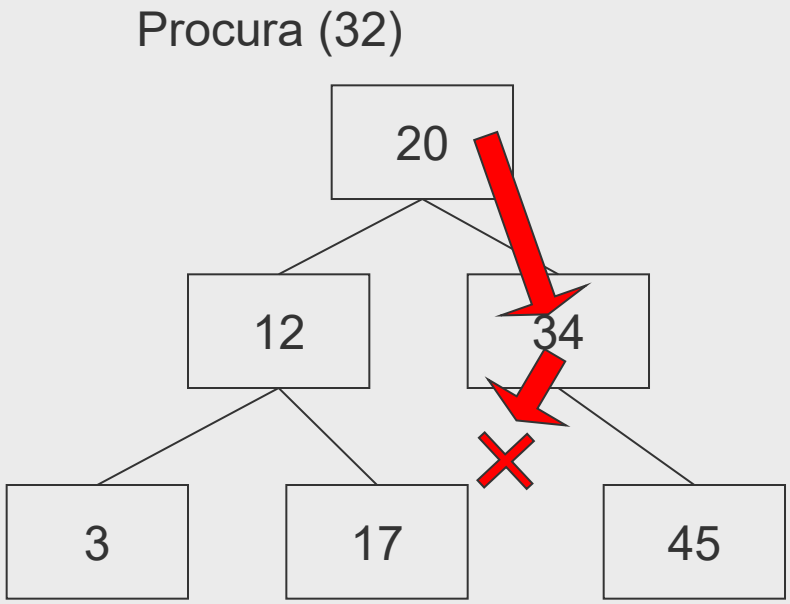
Aplicações de árvores Binárias

- Árvores de Pesquisa:



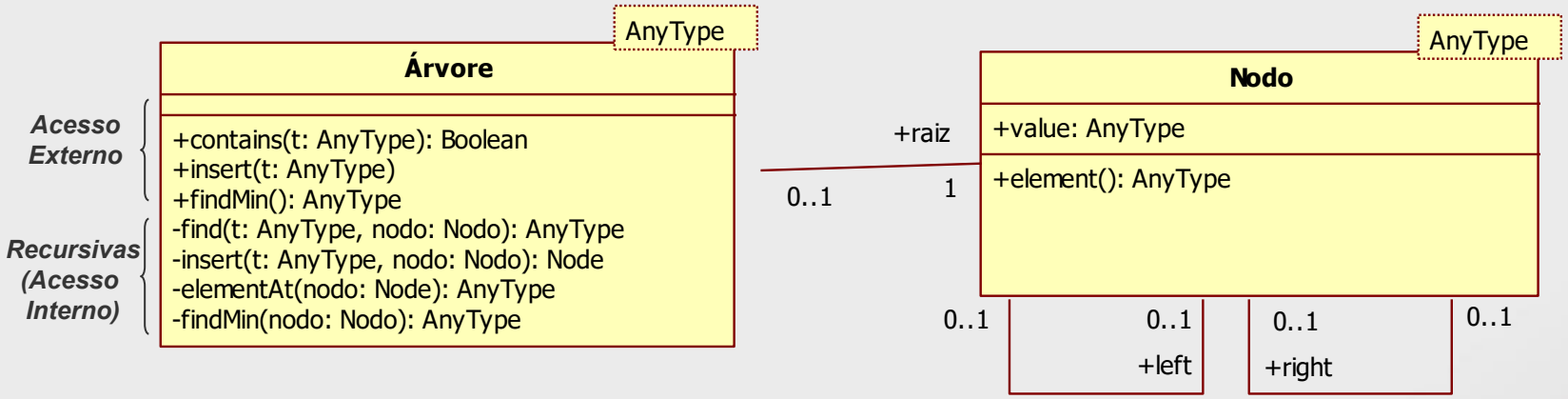
Aplicações de árvores Binárias

- Árvores de Pesquisa:



Aplicações de Árvores Binárias

- Estrutura Típica



- O acesso às funcionalidades é feito através de um método "externo" que se encarrega de chamar o método (usualmente recursivo) interno adequado e
  - *O método externo pode fazer algum processamento adicional que seja necessário (por exemplo, mudar a raiz da àrvore).*
  - *O interface de acesso ao exterior é simplificado.*

Aplicações de árvores Binárias

- **Árvores de Pesquisa Binárias**: Pesquisa do Elemento Mínimo

Versão Iterativa

```
private BinaryNode<AnyType> findMin(BinaryNode<AnyType> t )
{
if( t != null )
    {
    while( t.left != null )
          t = t.left;
    return t; // Encontrado
    }
return null;
}
```

Versão Recursiva

```
private BinaryNode<AnyType> findMin(BinaryNode<AnyType> t )
{
if (t==null)
    return null;
if(t.left==null)
    return t;
else
    return findMin(t.left);
}
```

Aplicações de árvores Binárias

- **Árvores de Pesquisa Binárias**: Pesquisa (versão Iterativa)

```
 private AnyType elementAt( BinaryNode<AnyType> t )
 {
 return t == null ? null : t.element;
 }
 …

private BinaryNode<AnyType> find( AnyType x, BinaryNode<AnyType> t )
{
while( t != null )
    {
    if( x.compareTo( t.element ) < 0 )
         t = t.left;
    else
         if( x.compareTo( t.element ) > 0 )
                  t = t.right;
         else
                  return t;
                  // Encontrado
                  //(Resultado menos provável testado em último lugar)
    }
return null;
}
```

Aplicações de árvores Binárias

- **Árvores de Pesquisa Binárias**: Pesquisa (versão recursiva)

```
private BinaryNode<AnyType> find( AnyType x, BinaryNode<AnyType> t )
{
if( t != null )
    {
    if( x.compareTo( t.element ) < 0 )
         return find(x,t.left);
    else
         if( x.compareTo( t.element ) > 0 )
                   return find(x,t.right);
         else
                  return t;
                  // Encontrado
                  //(Resultado menos provável testado em último lugar)
    }
return null;
}
```

Aplicações de árvores Binárias

- **Árvores de Pesquisa Binárias**: Inserção
- Caso a árvore esteja vazia, cria uma raiz nova.
- Caso contrário:
  - *Se for igual à raiz, gera excepção.*
  - *Se for maior do que a raiz, insere na sub-árvore direita*
  - *Se for menor do que a raiz, insere na sub-árvore esquerda.*

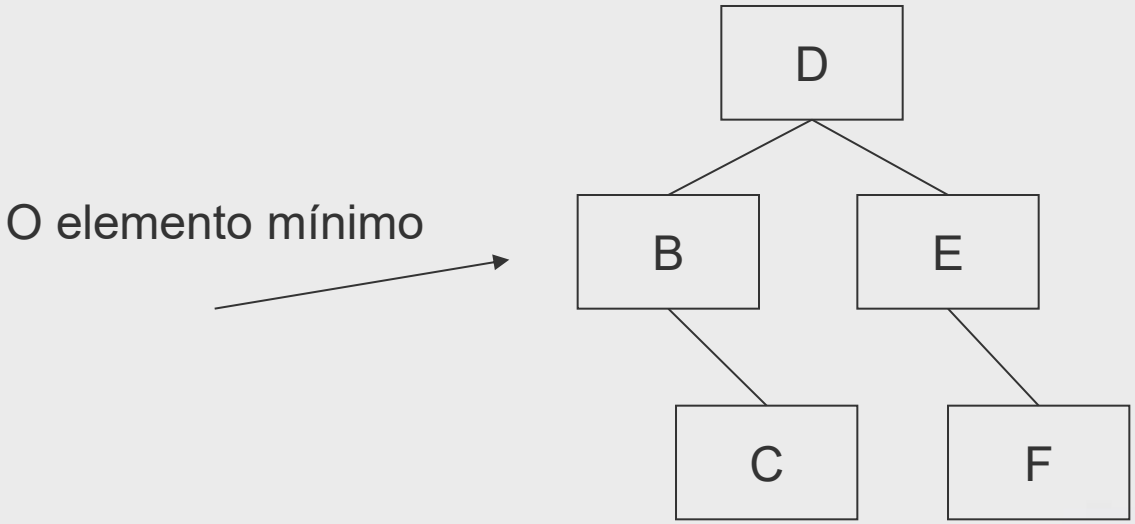
Aplicações de árvores Binárias

- **Árvores de Pesquisa Binárias**: Inserção

```
protected BinaryNode<AnyType> insert
    ( AnyType x, BinaryNode<AnyType> t )
{
    if( t == null )
        t = new BinaryNode<AnyType>( x );
    else
        if( x.compareTo( t.element ) < 0 )
                t.left = insert( x, t.left );
        else if( x.compareTo( t.element ) > 0 )
                t.right = insert( x, t.right );
        else
                throw new DuplicateItemException( x.toString( ) );
        // Duplicado
    return t; //nova raiz desta sub-árvore
}
```

Aplicações de árvores Binárias

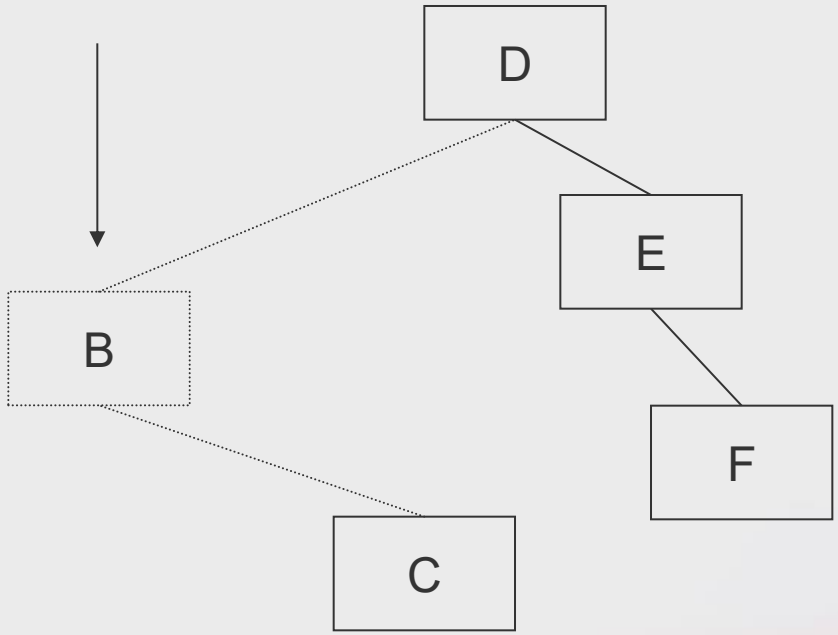
- **Árvores de Pesquisa Binárias**: Remoção
- Caso simples: remoção do elemento mais pequeno de uma subàrvore.



Aplicações de árvores Binárias

- **Árvores de Pesquisa Binárias**: Remoção
- Caso simples: remoção do elemento mais pequeno.

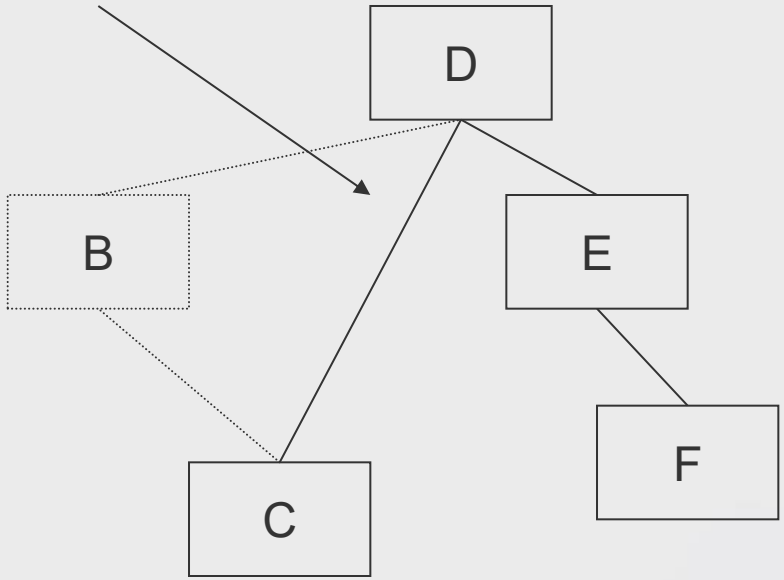
Depois de encontrar B é removido..



Aplicações de árvores Binárias

- **Árvores de Pesquisa Binárias**: Remoção
- Caso simples: remoção do elemento mais pequeno.

Ligando a sub-àrvore esquerda do menor ao antecessor desta forma…



Aplicações de árvores Binárias

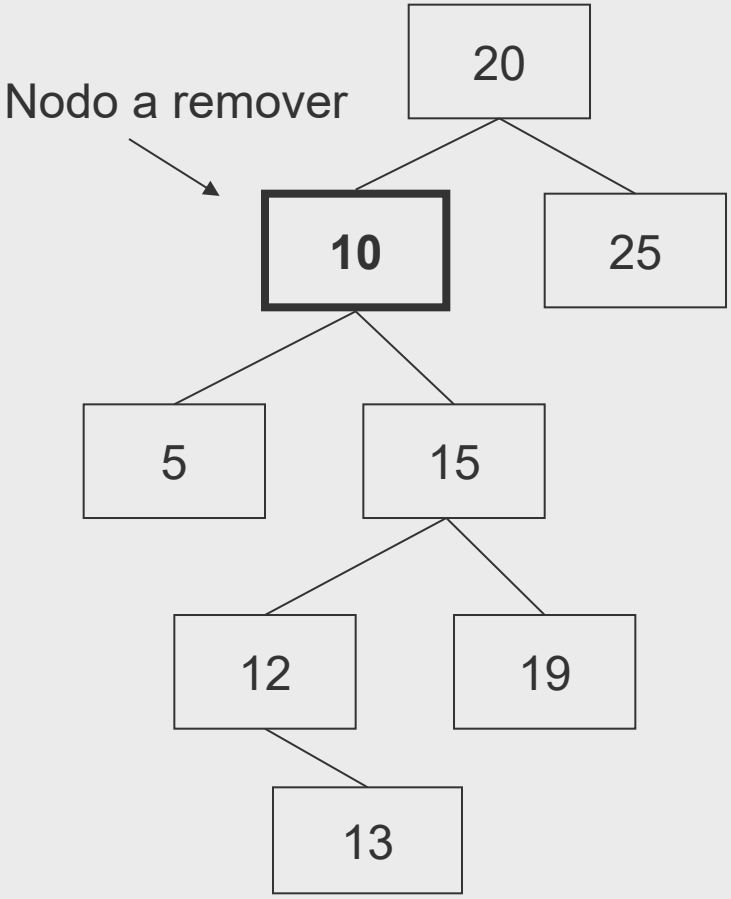
```
public void removeMin()
{
    root=removeMin(root);
}

private BinaryNode<AnyType> removeMin( BinaryNode<AnyType> t )
{
if( t == null )
    throw new ItemNotFoundException( );
else
    if( t.left != null )
          {
          t.left = removeMin( t.left );
          return t;
          }
    else
          return t.right;
}
```

Aplicações de árvores Binárias

- **Árvores de Pesquisa Binárias**: Remoção
- Se a àrvore está vazia
  - Gera excepção
- Caso contário:
  - O método de remoção é chamado adequadamente sobre a subàrvore esquerda ou direita até encontrar o elemento a ser removido. Então…

Aplicações de árvores Binárias

- Remoção de nodo

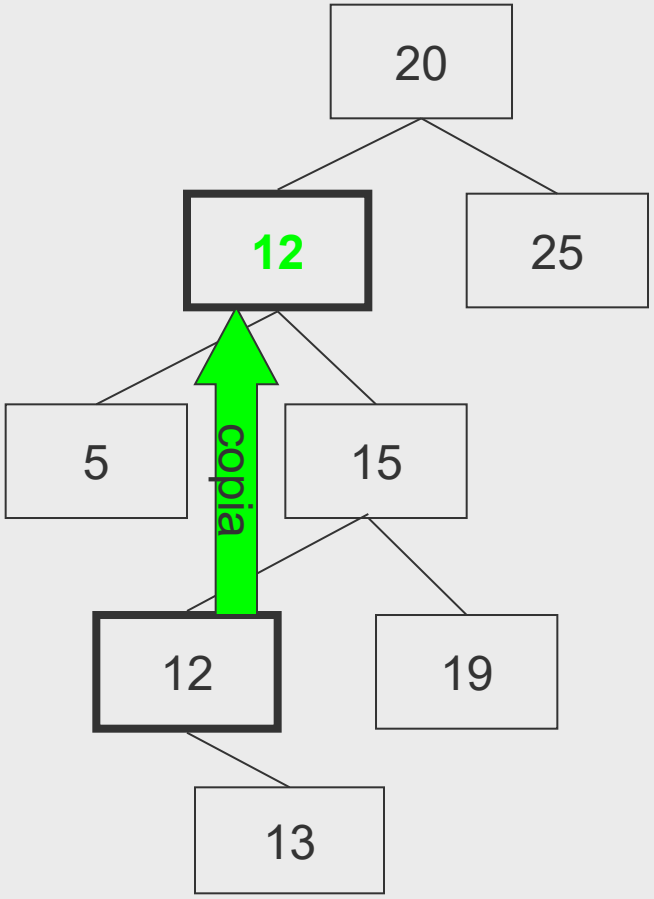


Passos para remoção:

1º Encontrar nodo a remover

Aplicações de árvores Binárias

- Remoção de nodo: remover nodo 10.



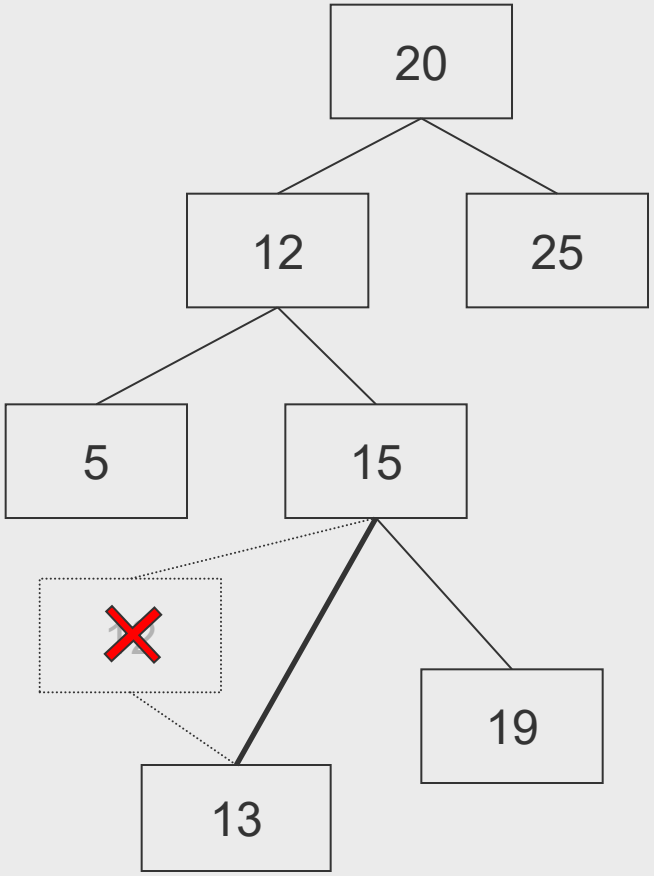
Passos para remoção:

1º Encontrar nodo a remover

2º Copiar nodo menor da sub-árvore direita.

Aplicações de árvores Binárias

- Remoção de nodo: remover nodo 10.



Passos para remoção:

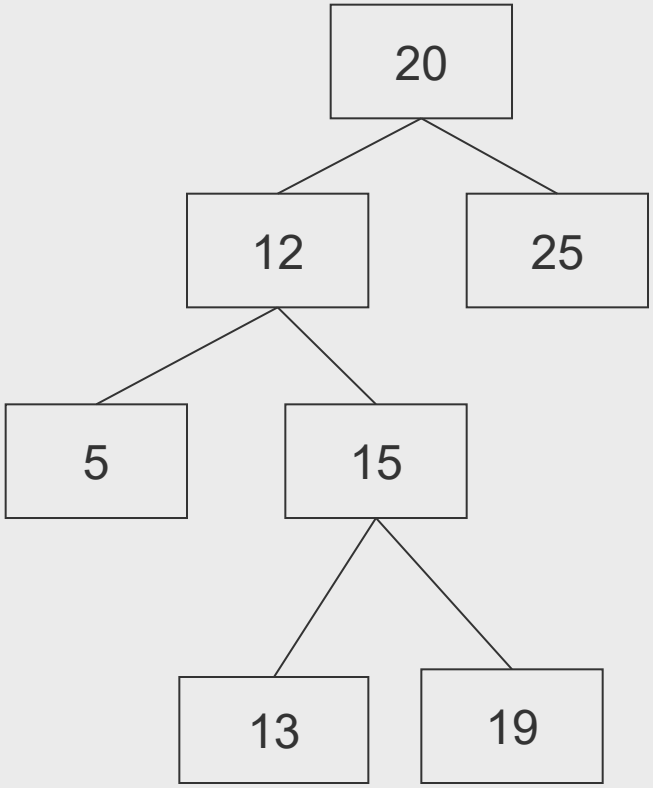
1º Encontrar nodo a remover

2º Copiar nodo menor da sub-árvore direita.

3º Remover menor nodo da sub-árvore direita.

Aplicações de árvores Binárias

» Remoção de nodo: remover nodo 10.



Passos para remoção:

1º Encontrar nodo a remover

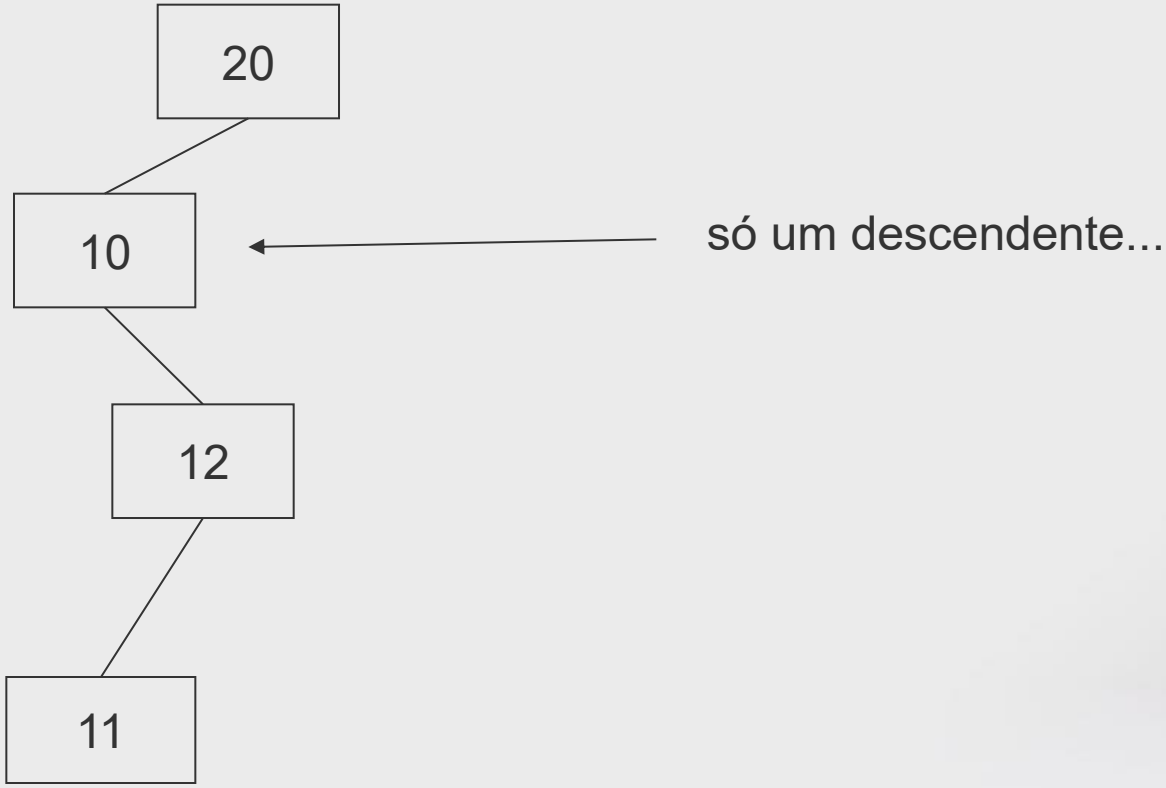
2º Copiar nodo menor da sub-árvore direita.

3º Remover menor nodo da sub-árvore direita.

4º A remoção está concluída

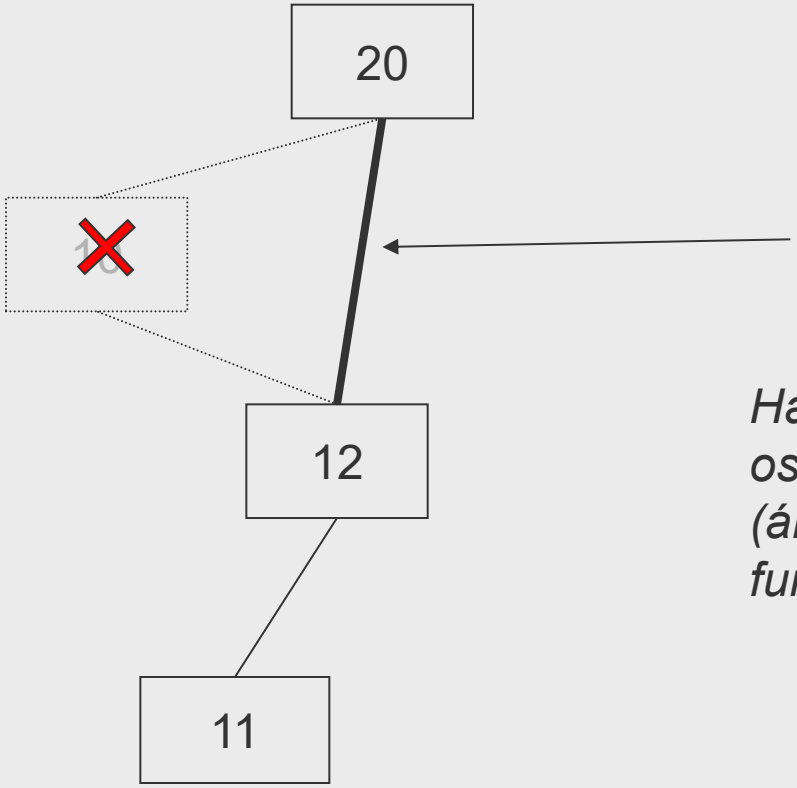
Aplicações de árvores Binárias

- Remoção de nodo: Casos especiais



Aplicações de árvores Binárias

- Remoção de nodo: Casos especiais



Basta remover nodo e ligar...

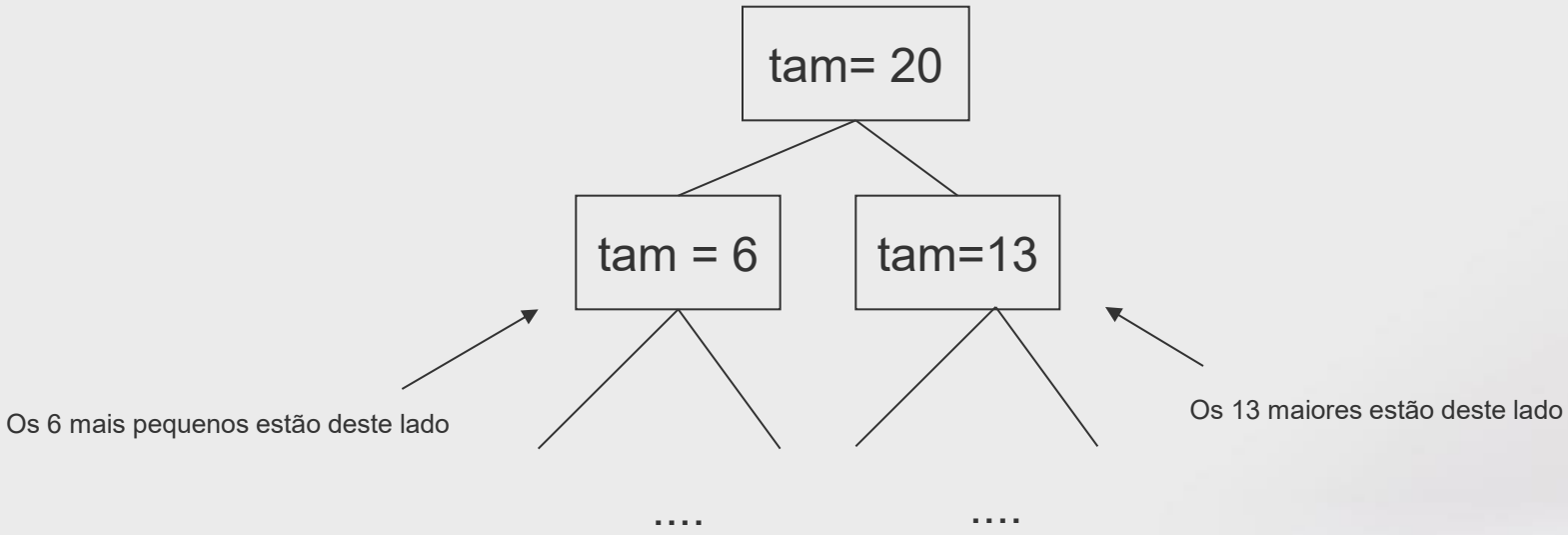
Há também que ter em atenção se os casos degenerados usuais (árvore vazia ou de um elemento) funcionam bem

Aplicações de árvores Binárias

```
protected BinaryNode<AnyType> remove( AnyType x, BinaryNode<AnyType> t )
 {
    if( t == null )
          throw new ItemNotFoundException( x.toString( ) );
    if( x.compareTo( t.element ) < 0 )
          t.left = remove( x, t.left );
    else if( x.compareTo( t.element ) > 0 )
          t.right = remove( x, t.right );
    else
          if( t.left != null && t.right != null ) // Dois descendentes
          {
             t.element = findMin( t.right ).element;
             t.right = removeMin( t.right );
          }
    else
          t = ( t.left != null ) ? t.left : t.right; // Um só descendente
return t;
}
```

Aplicações de árvores Binárias

- Procura do K-ésimo maior (menor) elemento.
  - O *tamanho* de um nodo é igual ao número dos seus descendentes (incluindo ele próprio).

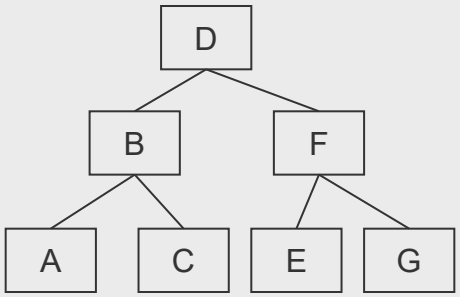


Árvores Binárias de Pesquisa

- Complexidade das operações
  - O custo das operações é proporcional ao número de nodos acedidos.
    - O custo de acesso a um nodo é proporcional a *profundidade+1*
      - *Profundidade=número de arcos no percurso até raiz*

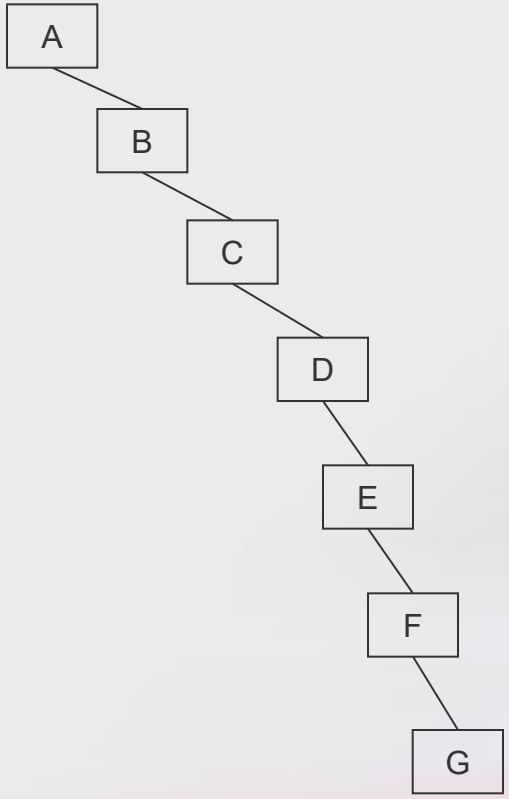
Árvores Binárias de Pesquisa

- Complexidade das operações:
  - Conforme a estrutura da árvore:



Árvore equilibrada: *O(log N)* no pior caso

Árvore desequilibrada: *O(N)* no pior caso



Árvores Binárias de Pesquisa

- Complexidade das operações:
  - **A complexidade varia entre *O(log N)* e *O(N)*!**
  - **E no caso "médio"?**
    - No caso de uma árvore criada através da inserção de elementos aleatórios distribuidos de forma uniforme, o desempenho é 38% pior do que no caso ideal.
  - **A remoção de nodos pode ser ligeiramente problemática (em teoria) dado que tende a desequilibrar a árvore para a esquerda...**
    - A profundidade de uma àrvore sujeita a *N* inserções/remoções tende a convergir para $O(\sqrt{N})$.
    - Mas em termos práticos o efeito não costuma ser observável.
  - **O principal problema não é a complexidade média ou o desequilirio causado pela remoção:**
    - O principal problema reside no facto de a introdução de sequências ordenadas (ou semi-ordenadas) é uma acção relativamente frequente, resultando directamente na redução ao pior caso possível.

Árvores Binárias Balanceadas

- Para além da propriedade de ordem, pode ser também adicionada uma propriedade adicional que assegura o equilíbrio da àrvore.
  - Nenhum nodo poderá atingir uma profundidade demasiado elevada em relação aos outros.
  - Há diferentes variantes da propriedade de equilibrio.
  - As operações de inserção e remoção tornam-se mais complicadas, mas a operação de pesquisa fica mais eficiente.

Árvores Equilibradas

- Abordagem simplificada
  - A propriedade de equilibrio mais simples é requerer que as sub-árvores direita e esquerda tenham a mesma profundidade.
    - A propriedade de equilibrio aplica-se, tal como a propriedade de ordem, a todos os nodos da árvore.
  - Esta abordagem é no entanto demasiado custosa em termos de *inserção* e *remoção* de nodos.
    - É necessário encontrar alternativas...

Árvore AVL

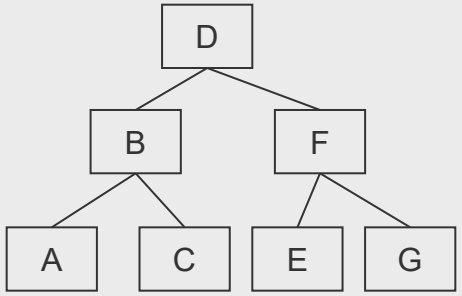
- Descoberta por **A**delson-**V**elskii e **L**andis

- **Propriedade de equilibrio AVL**

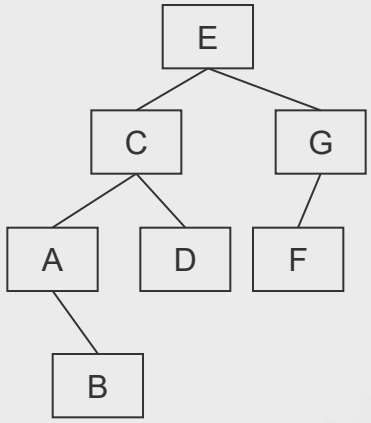
  Para qualquer nodo na árvore, a profundidade das sub-árvores direita e esquerda só pode diferir em 1 (a profundidade de uma sub-árvore vazia é -1).

Árvore AVL

Árvore “perfeitamente” equilibrada



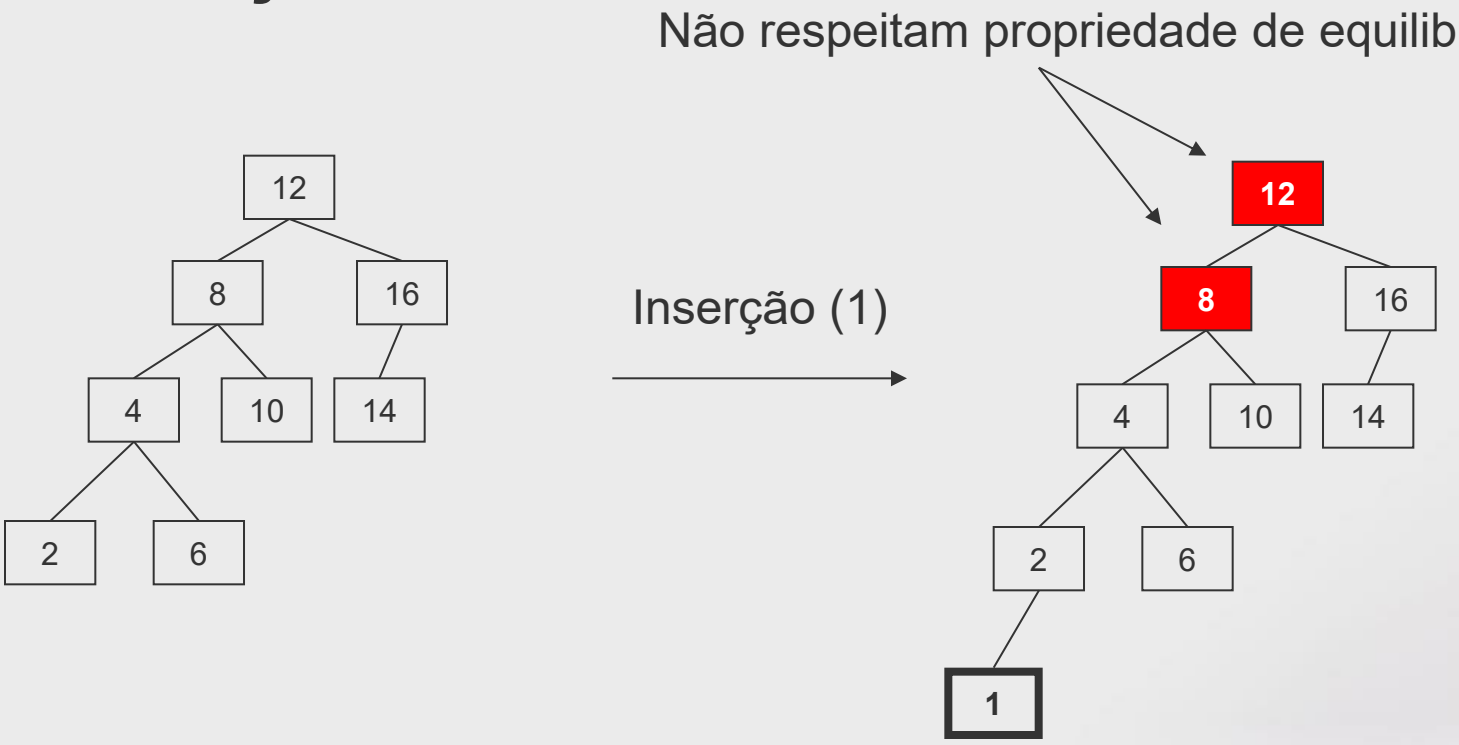
Árvore AVL



A profundidade de uma árvore AVL é, *no pior caso*, 1.44 vezes a profundidade de uma àrvore ideal.

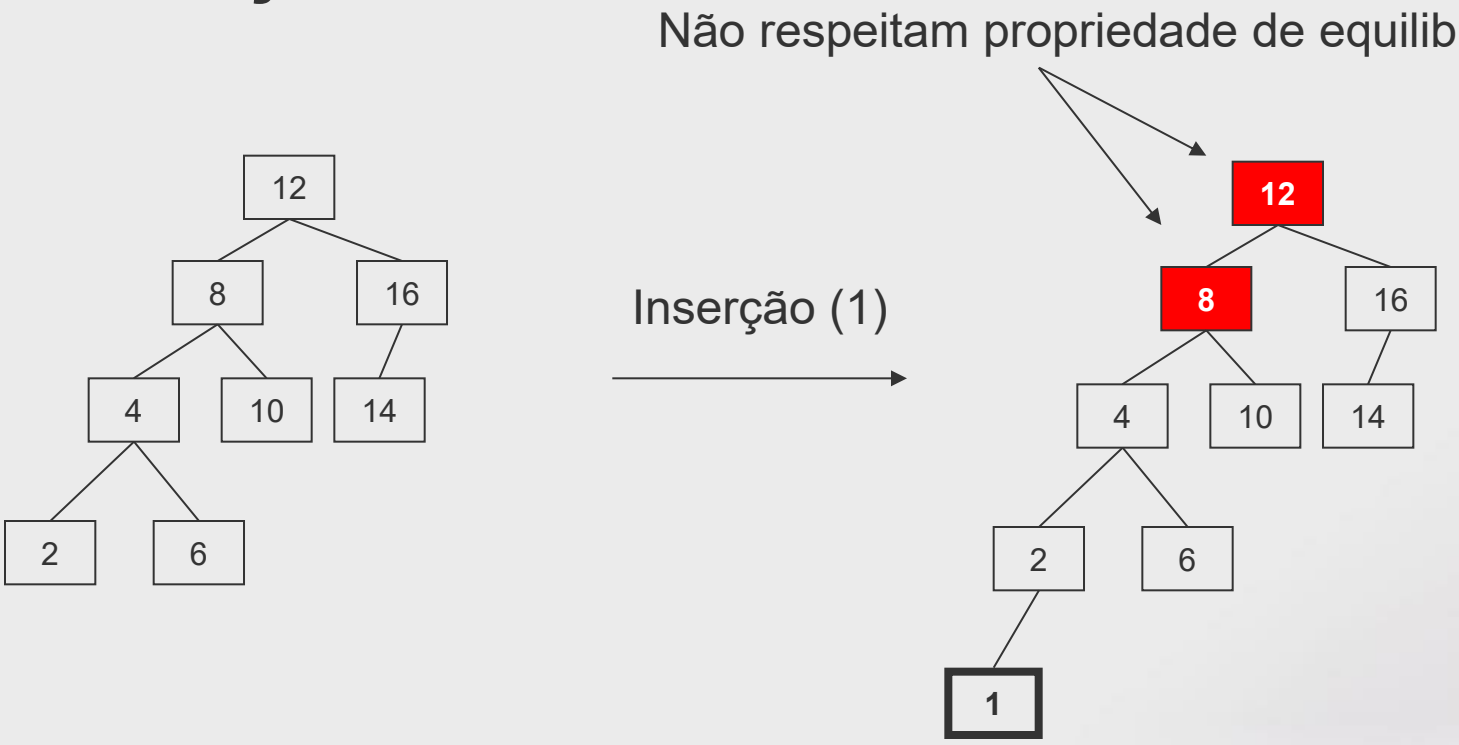
Árvore AVL

- Inserção



Árvore AVL

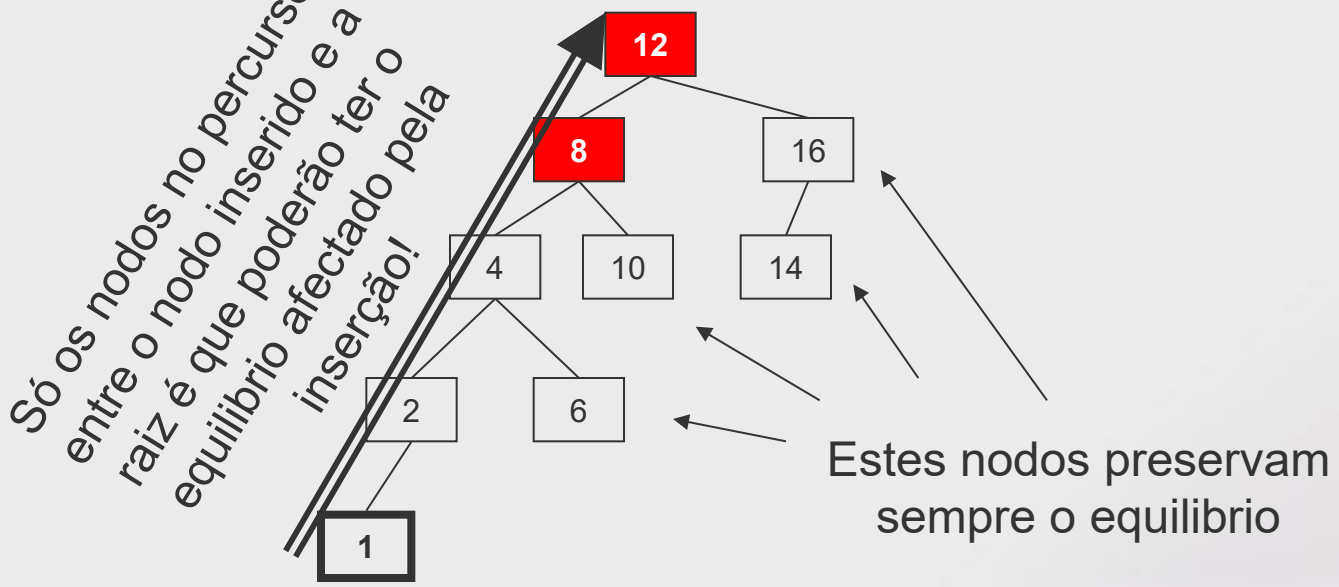
- Inserção



Depois da inserção, torna-se necessário efectuar uma operação adicional que assegura a preservação equilíbrio da árvore.

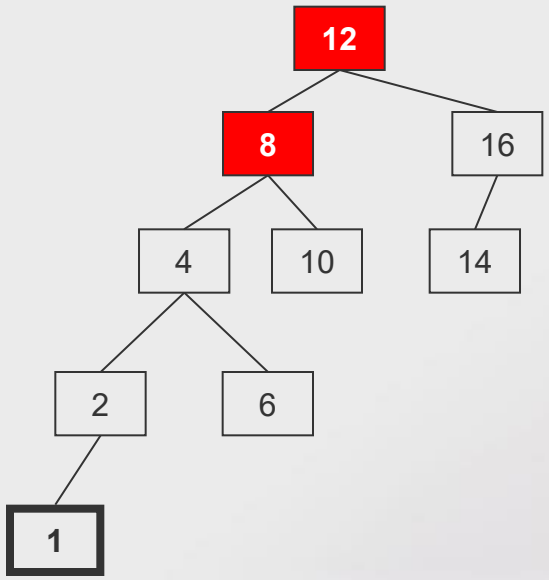
Árvore AVL

- Inserção
  - Observação



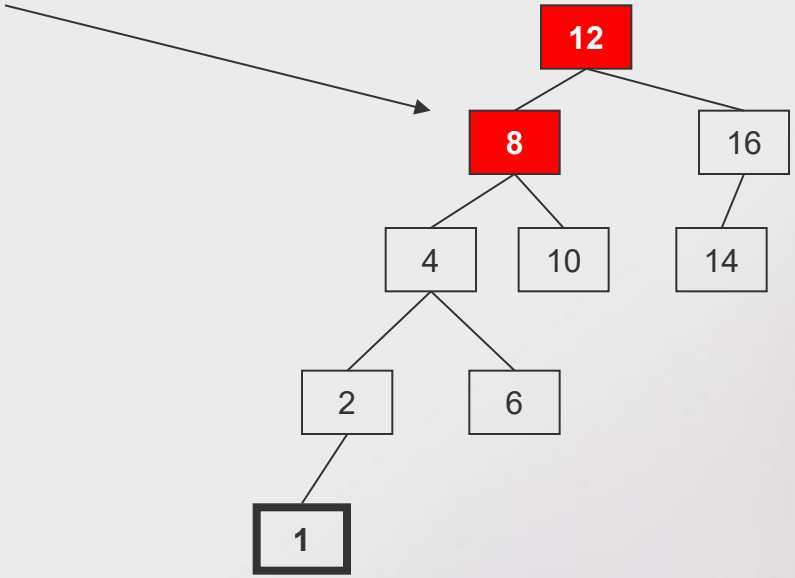
Árvore AVL

- Reequilibrar a árvore
  - Vamos percorrer a àrvore desde o nodo inserido até à raiz, reequilibrando a àrvore em cada nodo conforme necessário.
    - Será necessário reequilibrar apenas o primeiro nodo que viola a condição de equilibrio



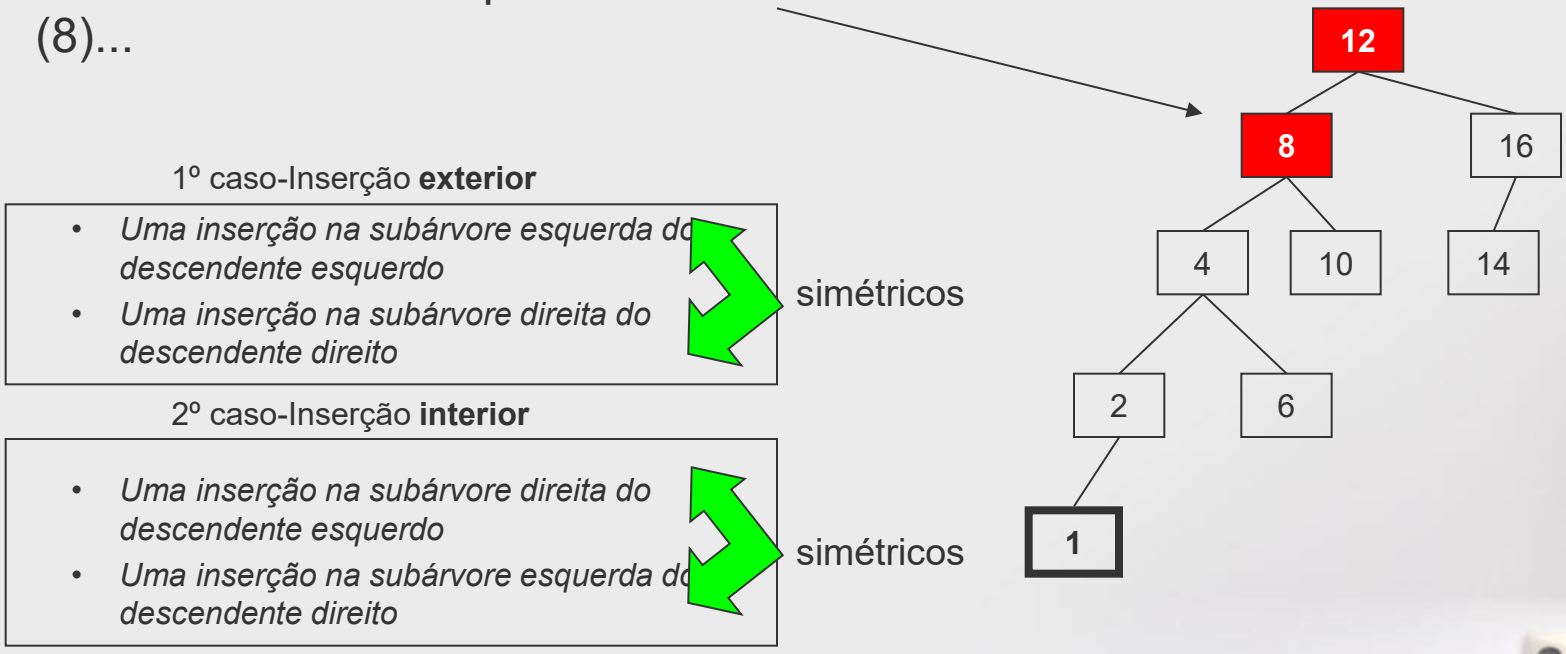
Árvore AVL

- Reequilibrar a árvore
  - Identificamos o nodo que necessita de ser reequilibrado (8)...
  - O desequilibrio pode acontecer por causa de um dos seguintes cenários:
    - *Uma inserção na subárvore esquerda do descendente esquerdo*
    - *Uma inserção na subárvore direita do descendente direito*
    - *Uma inserção na subárvore direita do descendente esquerdo*
    - *Uma inserção na subárvore esquerda do descendente direito*



Árvore AVL

- Reequilibrar a árvore
  - Identificamos o nodo que necessita de ser reequilibrado (8)...

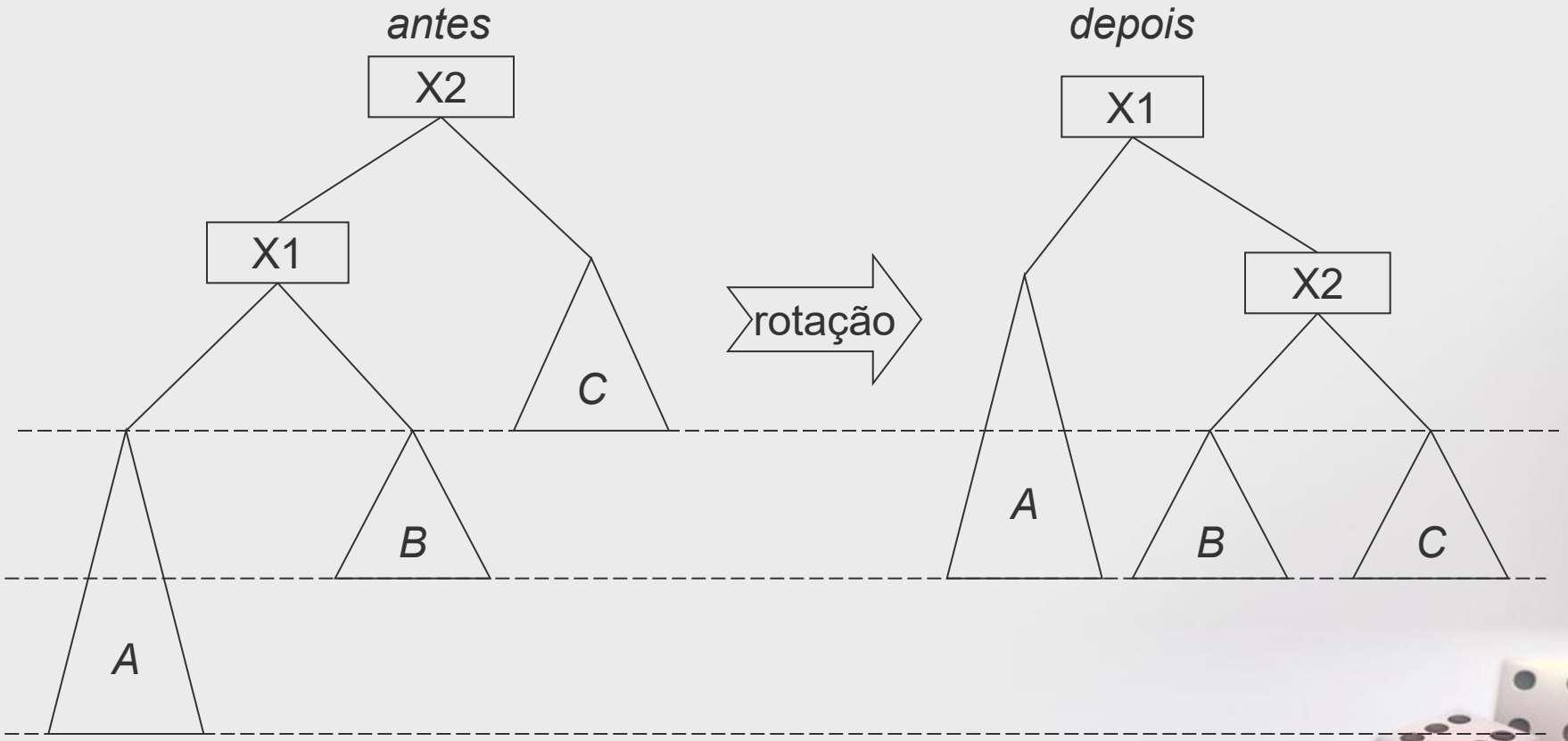


Árvore AVL

- Rotações
  - O resultado de uma inserção exterior pode ser equilibrado com uma única rotação da árvore.
  - O resultado de uma inserção interior pode ser equilibrado com duas rotações da árvore.

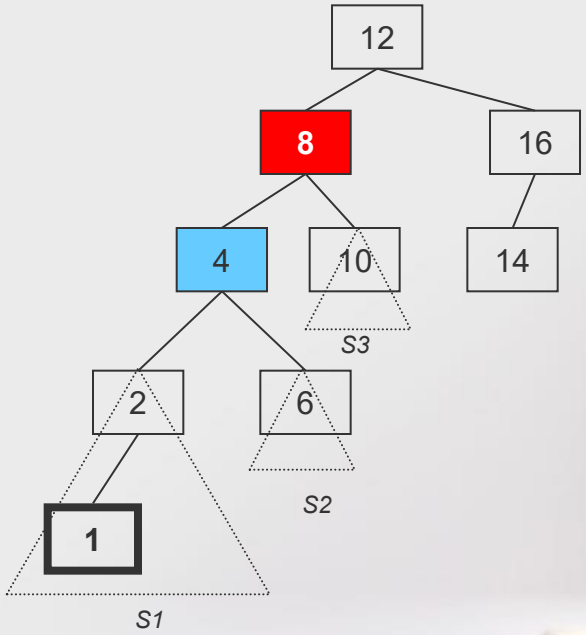
Árvore AVL

- Rotação



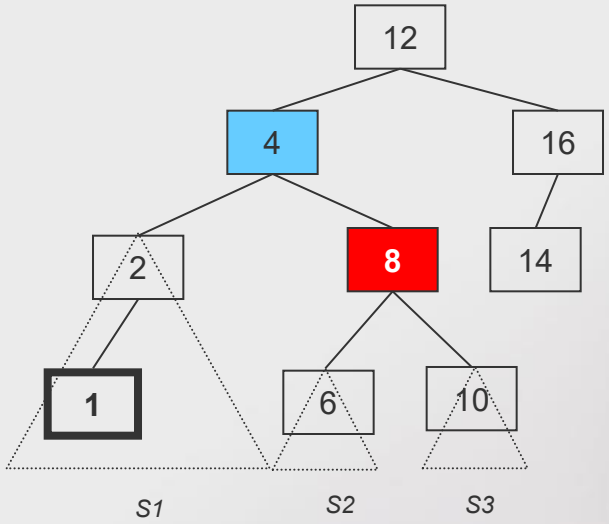
Árvore AVL

- Reequilibrar a árvore
  - Identificamos o nodo que necessita de ser reequilibrado (8)...
  - No caso de uma inserção exterior:
    - *Efectuamos uma rotação no nodo 8*



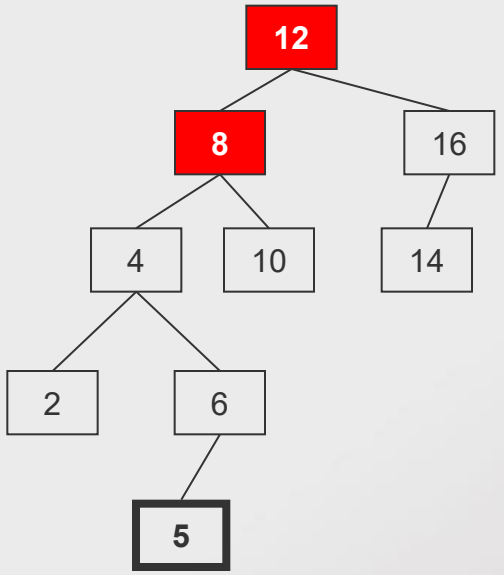
Árvore AVL

- Reequilibrar a árvore
  - Identificamos o nodo que necessita de ser reequilibrado (8)...
  - No caso de uma inserção exterior:
    - *Efectuamos uma rotação no nodo 8*
  - *A árvore está equilibrada!*



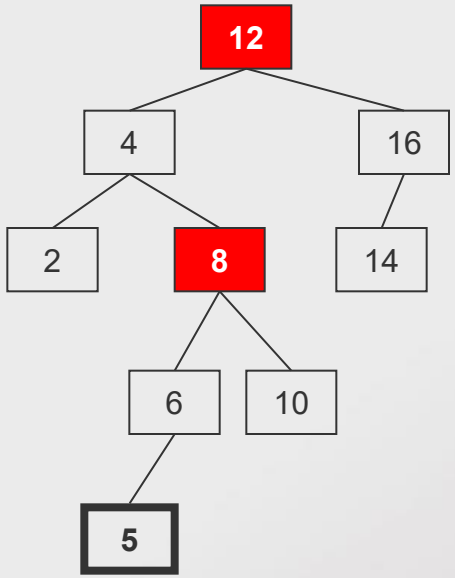
Árvore AVL

- Reequilibrar a árvore
  - E no caso de uma inserção interior?



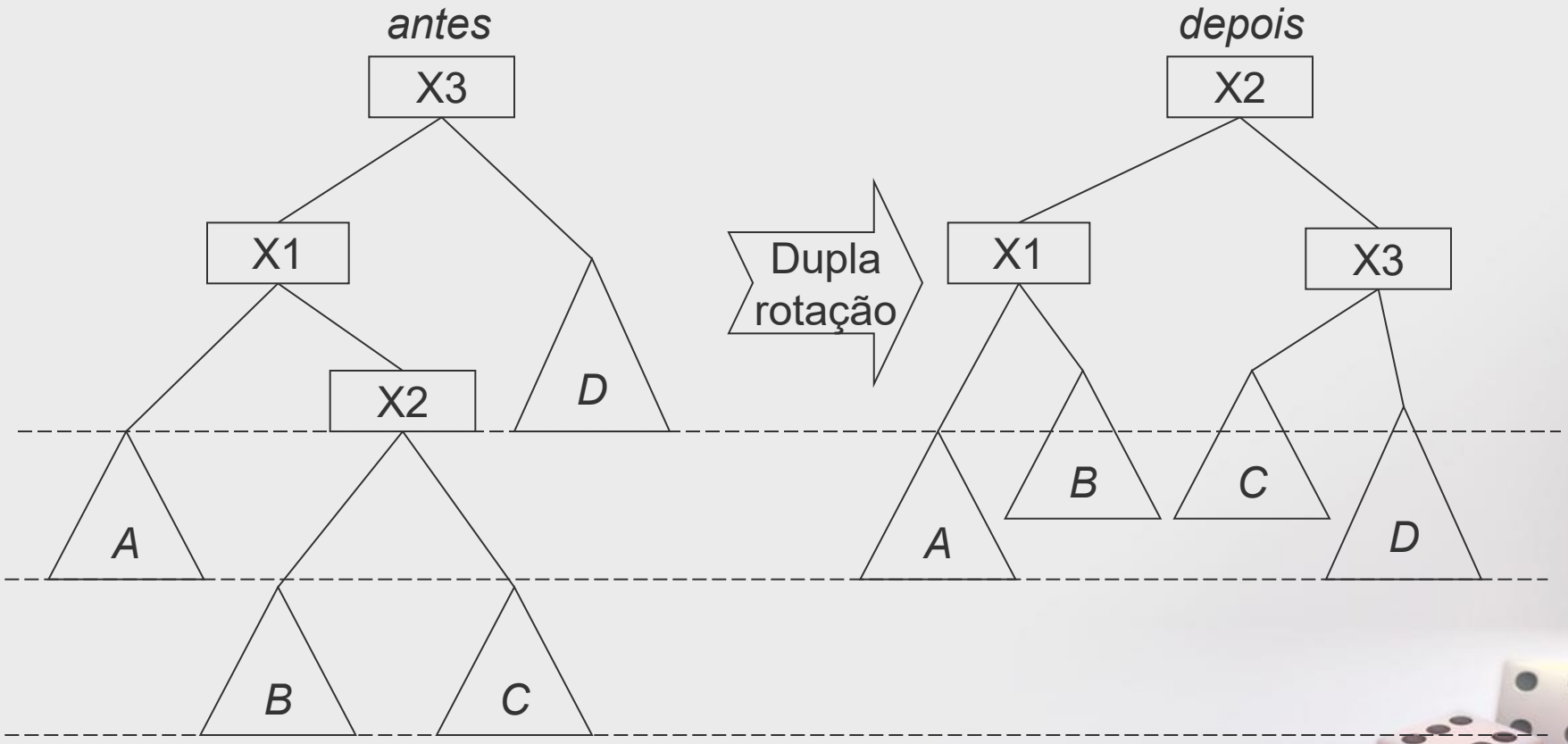
Árvore AVL

- Reequilibrar a árvore
  - E no caso de uma inserção interior?
  - A mesma rotação já não resolve o desequilibrio...



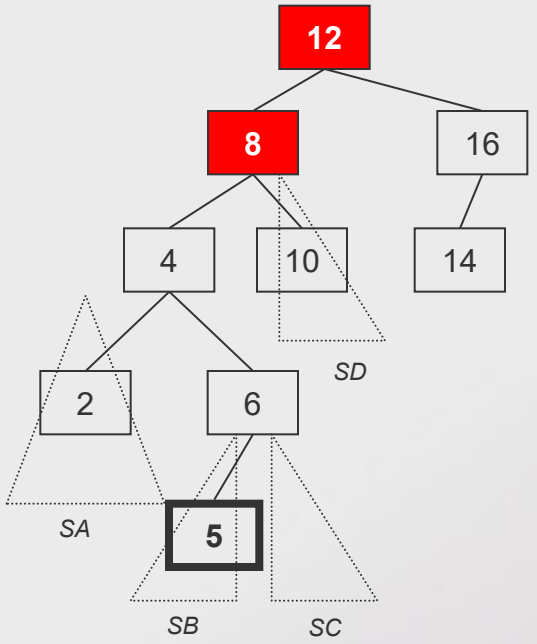
Árvore AVL

- Dupla Rotação



Árvore AVL

- Reequilibrar a árvore
  - E no caso de uma inserção interior?
  - Vamos subdividir o a subárvore onde a inserção é feita em duas...
  - E fazer duas rotações



Árvore AVL

- Reequilibrar a árvore
  - E no caso de uma inserção interior?
  - Vamos subdividir o a subárvore onde a inserção é feita em duas...
  - E fazer duas rotações