

Estrutura de Dados

Hamilton José Brumatto

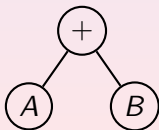
Bacharelado em Ciências da Computação - UESC

6 de junho de 2016

Árvores Binárias

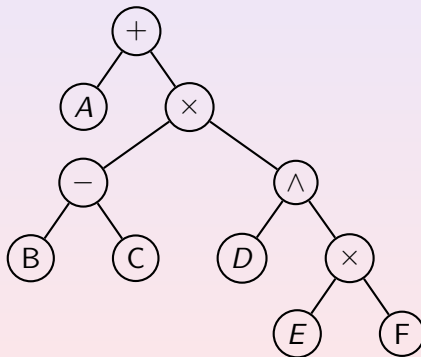
Árvores de Expressão

- Considere a seguinte expressão: $A + B$
- Esta expressão pode ser representada na forma de uma árvore binária:
 - O raiz é a operação.
 - O filho esquerdo é o operando esquerdo.
 - O filho direito é o operando direito.



Árvores de Expressão

- Uma expressão maior: $A + (B - C) \times D \wedge (E \times F)$



Aplicações

- A primeira aplicação direta é que a árvore serve para conversão de formas de notação:
 - Varredura em pré-ordem \rightarrow expressão pré-fixa.
 $+A \times -BC \wedge D \times EF$
 - Varredura em pós-ordem \rightarrow expressão pós-fixa.
 $ABC - DEF \times \wedge \times +$
 - Varredura em ordem simétria (desde que antes de visitar o filho esquerdo imprima-se um “Abre parêntesis”, e após visitar o filho direito imprima-se um “Fecha parêntesis” \rightarrow expressão infixa.
 $(A + ((B - C) \times (D \wedge (E \times F))))$

Aplicações

- A segunda aplicação é o cálculo do valor da expressão

Algoritmo para calcular expressão

Algoritmo CALCULAR(*Arvore a*)

val \leftarrow 0

se *a* \neq vazia **então**

se *info(a)* = *operacao* **então**

op1 = *Calcular(filhoEsq(a))*

op2 = *Calcular(filhoDir(a))*

val \leftarrow *Operacao(op1, info(a), op2)*

senão

val \leftarrow *info(a)*

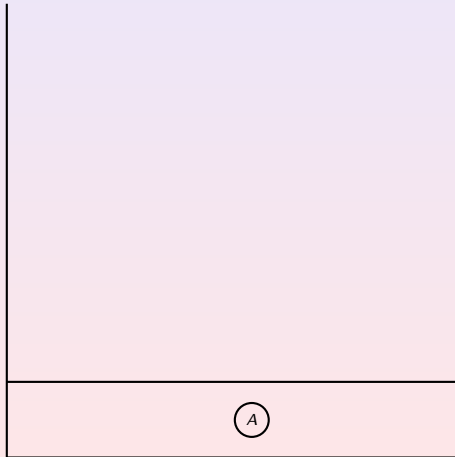
retorne *val*

Construção - Pós Fixa

- A partir de uma expressão pós-fixa →
- Se o item lido é um operando, crie uma árvore folha com o operando como raiz e empilhe.
- Se o item lido é um operador:
 - crie uma árvore binária com o operador como raiz.
 - retire um operando da pilha e insira-o como filho direito.
 - retire um operando da pilha e insira-o como filho esquerdo.
 - empilhe a árvore resultante.
- No final do processo, resta na pilha a árvore de expressão.

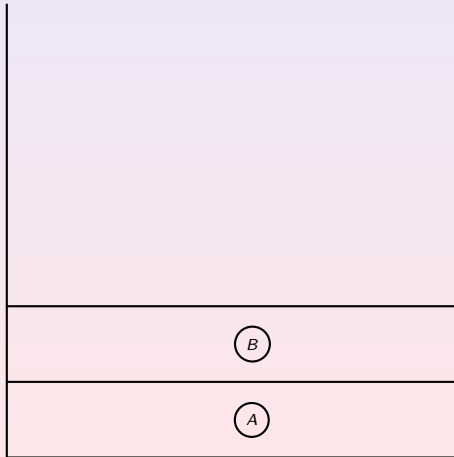
Construção - Pós Fixa

$ABC-DEF \times \wedge \times +$



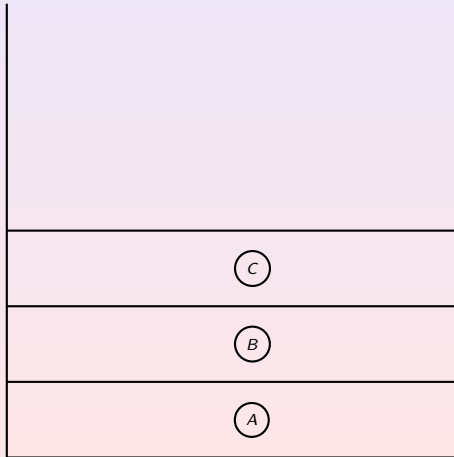
Construção - Pós Fixa

$ABC-DEF \times \wedge \times +$



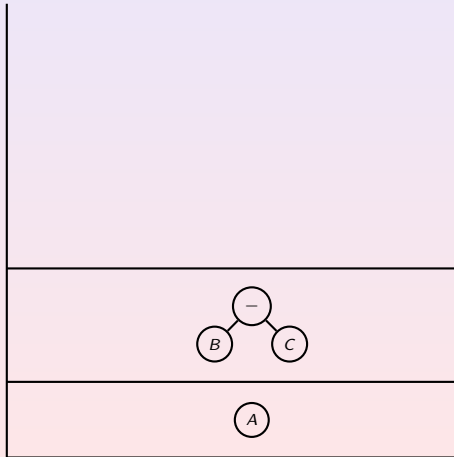
Construção - Pós Fixa

$ABC-DEF \times \wedge \times +$



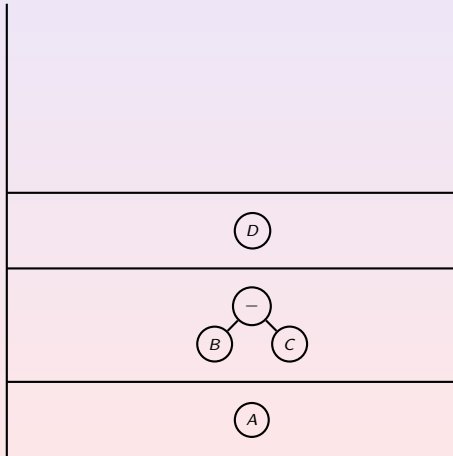
Construção - Pós Fixa

$ABC-DEF\times\wedge\times+$

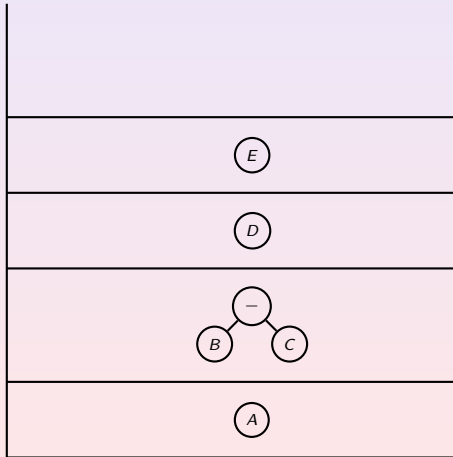


Construção - Pós Fixa

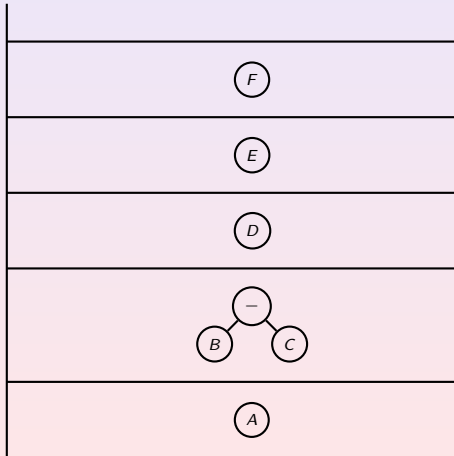
$ABC-DEF \times \wedge \times +$



Construção - Pós Fixa

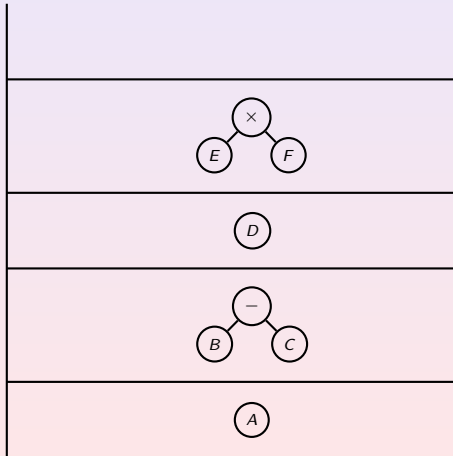
 $ABC-DEF\times\wedge\times+$ 

Construção - Pós Fixa

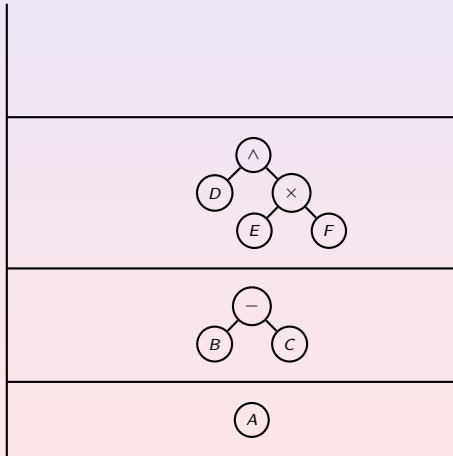
 $ABC-DEF \times \wedge \times +$ 

Construção - Pós Fixa

$ABC-DEF \times \wedge \times +$

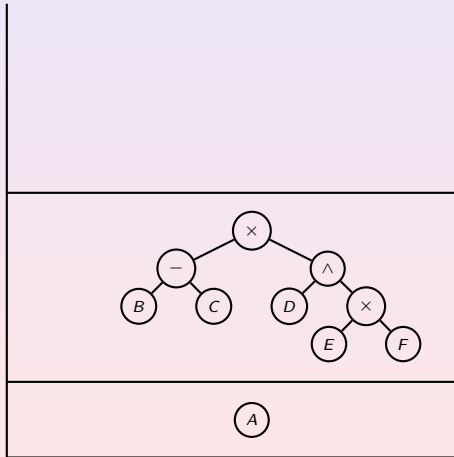


Construção - Pós Fixa
 $ABC-DEF \times \wedge \times +$



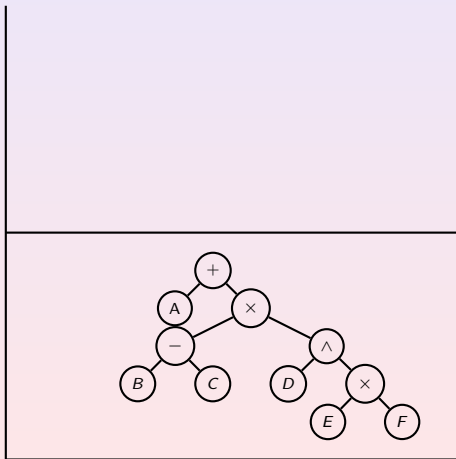
Construção - Pós Fixa

$ABC-DEF \times \wedge \times +$



Construção - Pós Fixa

$ABC-DEF \times \wedge \times +$



Construção - Infixa

- A partir de uma expressão infixa: É necessária uma total parentisação para evitar que a ordem de precedência de operandos seja determinante na ordem das operações:
- A expressão: $A + (B - C) \times D \wedge (E \times F)$ deve ser escrita como:
$$(A + ((B - C) \times (D \wedge (E \times F))))$$
- O algoritmo irá ler: *operando1*, *operacao*, *operando2* e construir uma árvore de expressão com estes três elementos.
- Ao encontrar um abre-parêntesis chama-se recursivamente o algoritmo.
- A última leitura é um fecha-parêntesis, quando retorna a árvore ao chamador.
- Na leitura de um operador, se não houver símbolo, o *operando1* é a árvore de expressão.

Construção - Infixa

Algoritmo CONSTRUIRINFIXA

```
S ← lerSimbolo()
se S = '(' então
    op1 ← ConstruirInfixa()
senão
    op1 ← NovaArvore(S)
S ← lerSimbolo()
se S = ']' então retorne op1
senão
    op ← NovaArvore(S)
    S ← lerSimbolo()
    se S = '(' então
        op2 ← ConstruirInfixa()
    senão
        op2 ← NovaArvore(S)
    op → InsereFilhoEsq(op1)
    op → InsereFilhoDir(op2)
lerSimbolo() retorne op
```

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$S \leftarrow \text{lerSimbolo}() : S = '(' \quad op1 : \quad \quad \quad op : \quad \quad \quad op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

se($S = '('$) \rightarrow então $op1 :$ $op :$ $op2 :$

Construção - Infixa $(A+((B-C)\times(D\wedge(E\times F))))$

$op1 \leftarrow \text{ConstruirInfixa}()$ $op1 :$ $op :$ $op2 :$

Construção - Infixa ($A + ((B - C) \times (D \wedge (E \times F)))$)

$S \leftarrow \text{lerSimbolo}() : S = A$	$op1 :$	$op :$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa ($A + ((B - C) \times (D \wedge (E \times F)))$)

$\text{se}(S = '(') \rightarrow \text{senão}$	$op1 :$	$op :$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa ($A + ((B - C) \times (D \wedge (E \times F))))$)

$op1 \leftarrow \text{NovaArvore}(S)$	$op1 : \textcircled{A}$	$op :$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$S \leftarrow \text{lerSimbolo}() : S = +$	$op1 : (A)$	$op :$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$\text{se}(S = \emptyset) \rightarrow \text{senão}$	$op1 : (A)$	$op :$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$op \leftarrow \text{NovaArvore}(S)$	$op1 : \textcircled{A}$	$op : \textcircled{+}$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A+((B-C)\times(D\wedge(E\times F))))$

$S \leftarrow \text{lerSimbolo}() : S = '('$	$op1 : (A)$	$op : (+)$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A+((B-C)\times(D\wedge(E\times F))))$

se $(S = '(') \rightarrow$ então	$op1 : (A)$	$op : (+)$	$op2 :$
$op1 \leftarrow ConstruirInfixa()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A+((B-C)\times(D\wedge(E\times F))))$

$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : (A)$	$op : (+)$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A+((B-C)\times(D\wedge(E\times F))))$

$S \leftarrow \text{lerSimbolo}() : S = '('$	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : (A)$	$op : (+)$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

se ($S = '('$) → então	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : (A)$	$op : (+)$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : (A)$	$op : (+)$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$S \leftarrow \text{lerSimbolo}() : S = B$	$op1 :$	$op :$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : (A)$	$op : (+)$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

se ($S = '('$) → senão	<i>op1</i> :	<i>op</i> :	<i>op2</i> :
<i>op1</i> \leftarrow ConstruirInfixa()	<i>op1</i> :	<i>op</i> :	<i>op2</i> :
<i>op2</i> \leftarrow ConstruirInfixa()	<i>op1</i> : \textcircled{A}	<i>op</i> : $\textcircled{+}$	<i>op2</i> :
<i>op1</i> \leftarrow ConstruirInfixa()	<i>op1</i> :	<i>op</i> :	<i>op2</i> :

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$op1 \leftarrow \text{NovaArvore}(S)$	$op1 : \textcircled{B}$	$op :$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : \textcircled{A}$	$op : \textcircled{+}$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$S \leftarrow \text{lerSimbolo}() : S = -$	$op1 : \textcircled{B}$	$op :$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : \textcircled{A}$	$op : \textcircled{+}$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

se ($S = \emptyset$) → senão	$op1 : \textcircled{B}$	$op :$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : \textcircled{A}$	$op : \textcircled{+}$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$op \leftarrow \text{NovaArvore}(S)$	$op1 : \textcircled{B}$	$op : \textcircled{-}$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : \textcircled{A}$	$op : \textcircled{+}$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$S \leftarrow \text{lerSimbolo}() : S = C$	$op1 : \textcircled{B}$	$op : \textcircled{-}$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : \textcircled{A}$	$op : \textcircled{+}$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

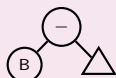
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

se ($S = '('$) \rightarrow senão	$op1 : \textcircled{B}$	$op : \textcircled{-}$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : \textcircled{A}$	$op : \textcircled{+}$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

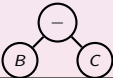
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$op2 \leftarrow \text{NovaArvore}(S)$	$op1 : (B)$	$op : (-)$	$op2 : (C)$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : (A)$	$op : (+)$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

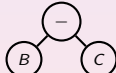
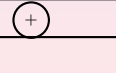
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$op \rightarrow \text{InsereFilhoEsq}(op1)$	$op1 : (B)$		$op2 : (C)$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : (A)$	$op : (+)$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$








Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$op \rightarrow \text{InsereFilhoDir}(op2)$	$op1 : (B)$		$op2 : (C)$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : (A)$	$op : (+)$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

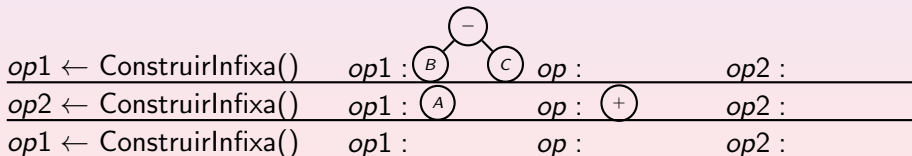
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

lerSimbolo()		
$op1 \leftarrow ConstruirInfixa()$	$op1 :$	$op :$
$op2 \leftarrow ConstruirInfixa()$	$op1 :$	$op :$
$op1 \leftarrow ConstruirInfixa()$	$op1 :$	$op :$

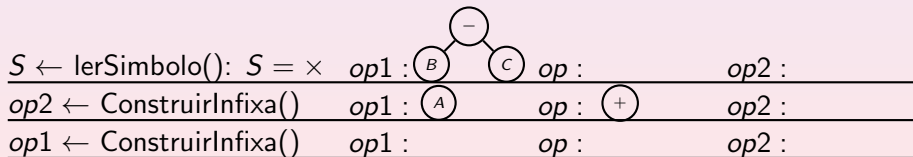
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

retorne <i>op</i>	<i>op1</i> : 	<i>op</i> :   	<i>op2</i> : 
<i>op1</i> \leftarrow ConstruirInfixa()	<i>op1</i> :	<i>op</i> :	<i>op2</i> :
<i>op2</i> \leftarrow ConstruirInfixa()	<i>op1</i> : 	<i>op</i> : 	<i>op2</i> :
<i>op1</i> \leftarrow ConstruirInfixa()	<i>op1</i> :	<i>op</i> :	<i>op2</i> :




Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



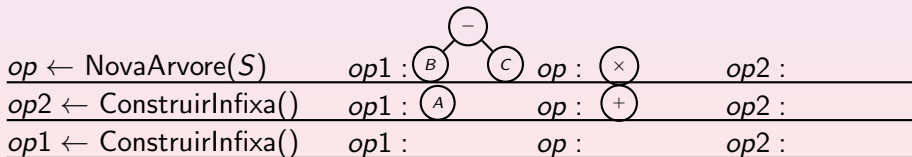
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



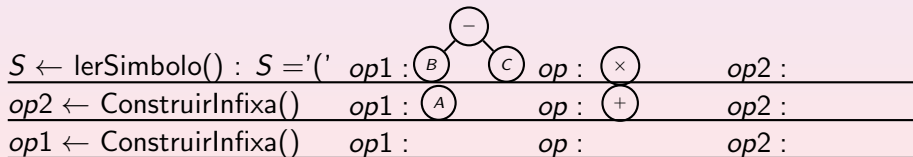
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

se ($S = \emptyset$) \rightarrow senão	$op1 :$		$op :$	$op2 :$	
$op2 \leftarrow$ ConstruirInfixa()	$op1 :$		$op :$		$op2 :$
$op1 \leftarrow$ ConstruirInfixa()	$op1 :$		$op :$	$op2 :$	

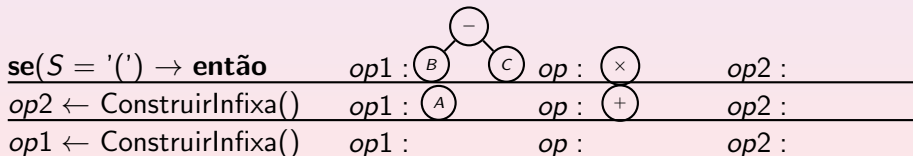
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



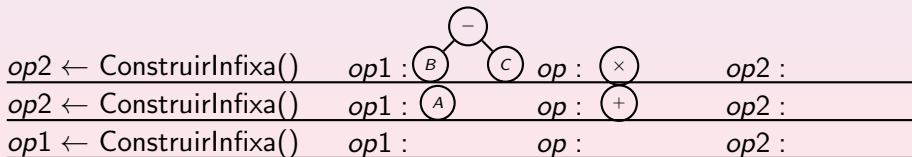
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$








Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$








Construção - Infixa $(A+((B-C)\times(D\wedge(E\times F))))$










Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

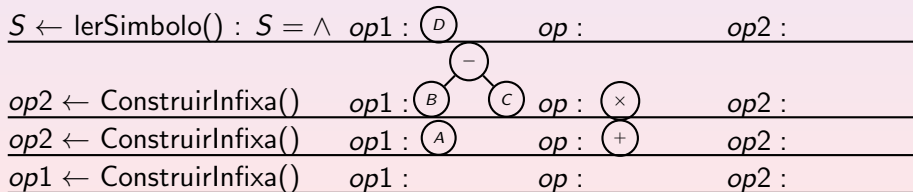
$S \leftarrow \text{lerSimbolo}() : S = D$	$op1 :$	$op :$	$op2 :$
			
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



<u>se($S = '('$) \rightarrow senão</u>	<u>$op1 :$</u>	<u>$op :$</u>	<u>$op2 :$</u>
			
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$









$op1 \leftarrow \text{NovaArvore}(S)$	$op1 :$ 	$op :$	$op2 :$
			
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
			
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

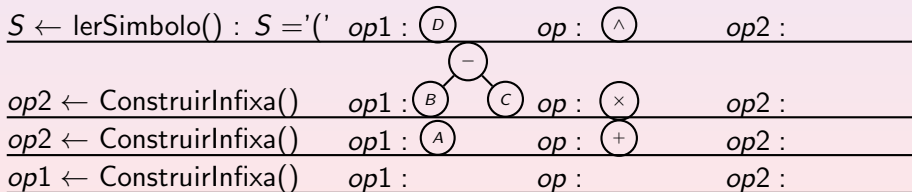
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$ 

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$








$\text{se}(S = \emptyset) \rightarrow \text{senão}$	$op1 :$	(D)	$op :$	$op2 :$
				
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$		$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	(A)	$op :$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$		$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$









$op \leftarrow \text{NovaArvore}(S)$	$op1 :$ 	$op :$ 	$op2 :$
			
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
			
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$ 


Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

se ($S = '('$) → então	$op1 :$ 	$op :$ 	$op2 :$
			
$op2 \leftarrow ConstruirInfixa()$	$op1 :$ 	$op :$ 	$op2 :$
$op2 \leftarrow ConstruirInfixa()$	$op1 :$ 	$op :$ 	$op2 :$
$op1 \leftarrow ConstruirInfixa()$	$op1 :$	$op :$	$op2 :$


Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
			
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
			
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$









Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$S \leftarrow \text{lerSimbolo}() : S = E$	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : (D)$	$op : (\wedge)$	$op2 :$
			
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : (B) \quad (C)$	$op : (\times)$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 : (A)$	$op : (+)$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A+((B-C)\times(D\wedge(E\times F))))$

se ($S = '('$) → senão	$op1 :$	$op :$	$op2 :$
$op2 \leftarrow ConstruirInfixa()$	$op1 : (D)$	$op : (\wedge)$	$op2 :$
			
$op2 \leftarrow ConstruirInfixa()$	$op1 : (B)$	$op : (\times)$	$op2 :$
$op2 \leftarrow ConstruirInfixa()$	$op1 : (A)$	$op : (+)$	$op2 :$
$op1 \leftarrow ConstruirInfixa()$	$op1 :$	$op :$	$op2 :$










Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$op1 \leftarrow \text{NovaArvore}(S)$	$op1 :$ 	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
			
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$










Construção - Infixa $(A+((B-C)\times(D\wedge(E\times F))))$

$S \leftarrow \text{lerSimbolo()} : S = \times$	$op1 :$	(E)	$op :$	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	(D)	$op :$	(\wedge)
		$(-)$		
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	(B)	$op :$	(\times)
		(C)		
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	(A)	$op :$	$(+)$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$		$op :$	$op2 :$


Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$se(S = \emptyset) \rightarrow$ senão	$op1 :$ 	$op :$	$op2 :$
$op2 \leftarrow ConstruirInfixa()$	$op1 :$ 	$op :$ 	$op2 :$
			
$op2 \leftarrow ConstruirInfixa()$	$op1 :$  	$op :$ 	$op2 :$
$op2 \leftarrow ConstruirInfixa()$	$op1 :$ 	$op :$ 	$op2 :$
$op1 \leftarrow ConstruirInfixa()$	$op1 :$	$op :$	$op2 :$










Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$op \leftarrow \text{NovaArvore}(S)$	$op1 :$ 	$op :$ 	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
			
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$












Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

$S \leftarrow \text{lerSimbolo}() : S = F$	$op1 :$	(E)	$op :$	(\times)	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	(D)	$op :$	(\wedge)	$op2 :$
					
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	(B)	$op :$	(\times)	$op2 :$
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	(A)	$op :$	$(+)$	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$		$op :$		$op2 :$

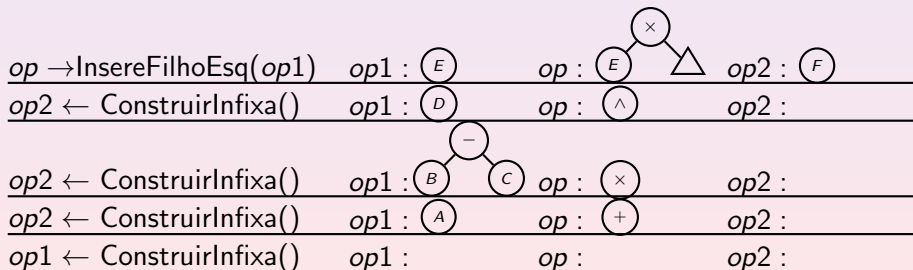
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

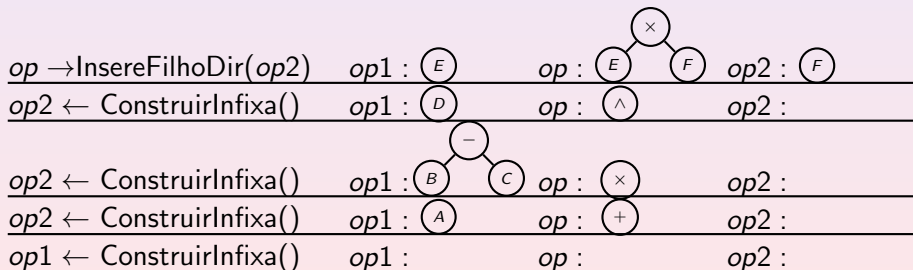
se ($S = '('$) → senão	$op1 :$ 	$op :$ 	$op2 :$
$op2 \leftarrow ConstruirInfixa()$	$op1 :$ 	$op :$ 	$op2 :$
			
$op2 \leftarrow ConstruirInfixa()$	$op1 :$ 	$op :$ 	$op2 :$
$op2 \leftarrow ConstruirInfixa()$	$op1 :$ 	$op :$ 	$op2 :$
$op1 \leftarrow ConstruirInfixa()$	$op1 :$	$op :$	$op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

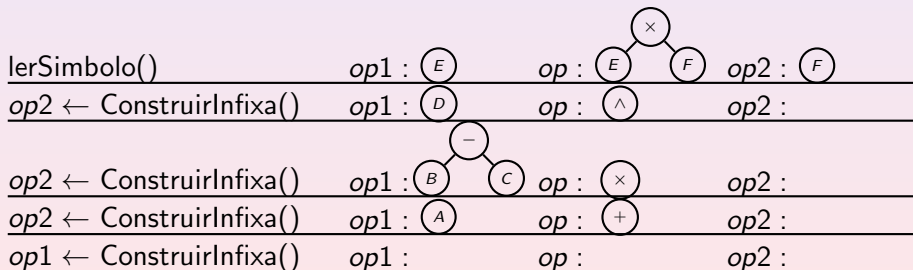
$op2 \leftarrow \text{NovaArvore}(S)$	$op1 :$ 	$op :$ 	$op2 :$ 
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
			
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
			
$op2 \leftarrow \text{ConstruirInfixa}()$	$op1 :$ 	$op :$ 	$op2 :$
$op1 \leftarrow \text{ConstruirInfixa}()$	$op1 :$	$op :$	$op2 :$

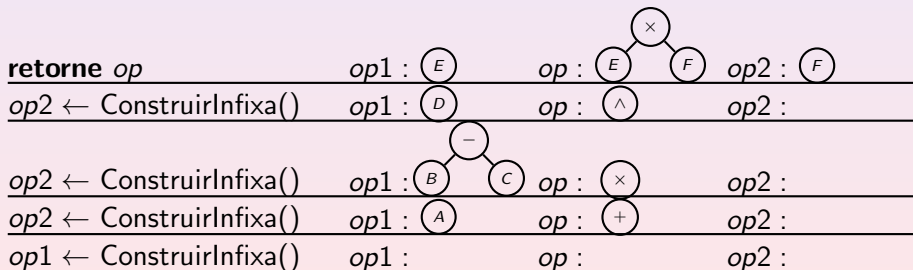
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

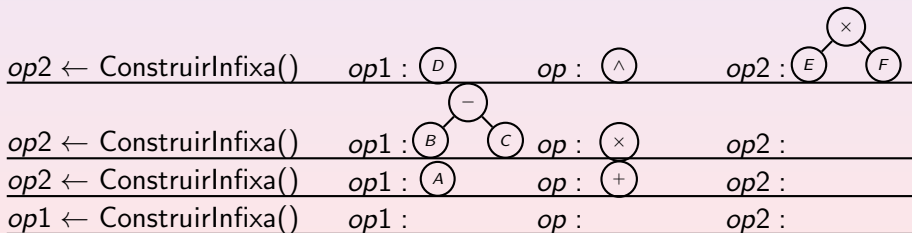


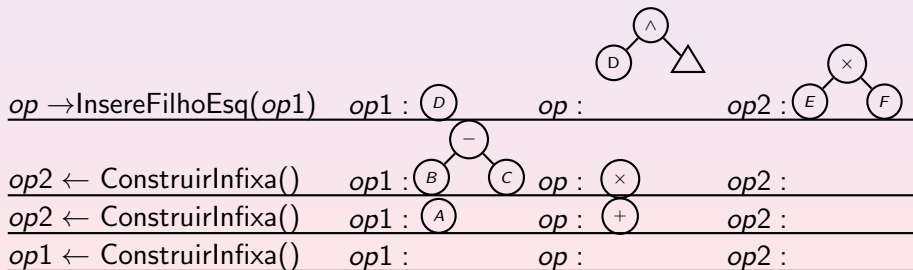
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$ 

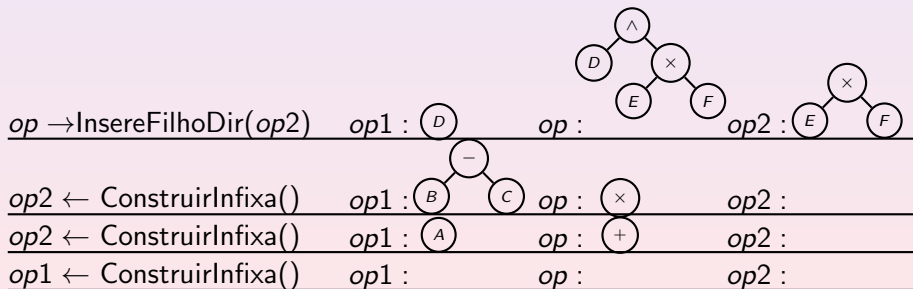
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



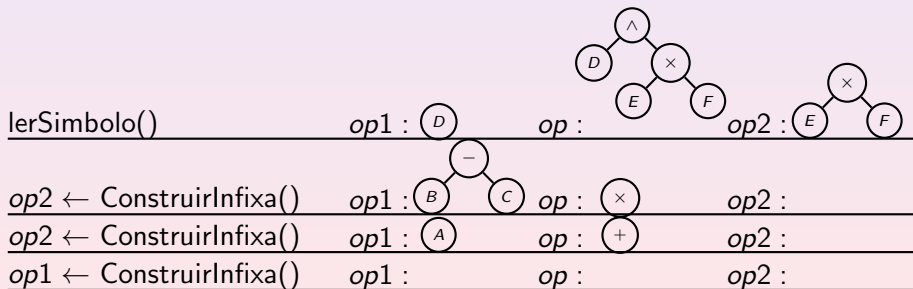
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$ 

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$ 

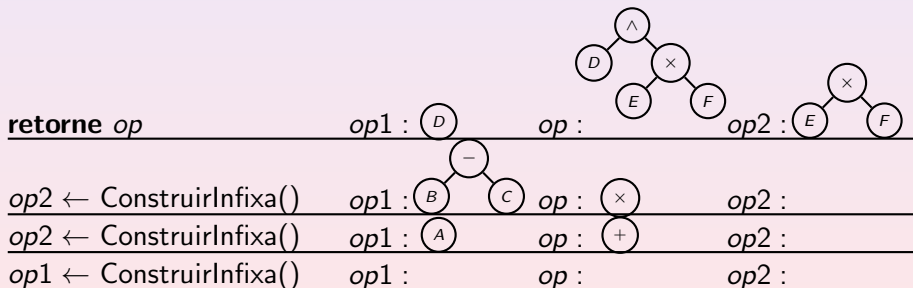
Construção - Infixa $(A+((B-C)\times(D\wedge(E\times F))))$ 

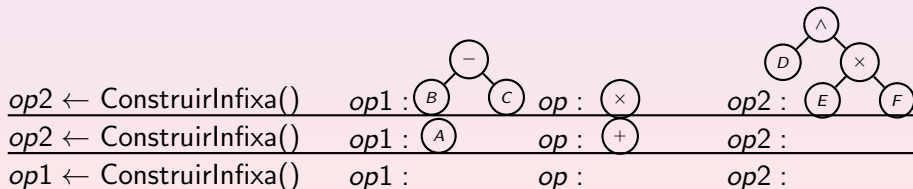
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$ 

Construção - Infixa $(A+((B-C)\times(D\wedge(E\times F))))$

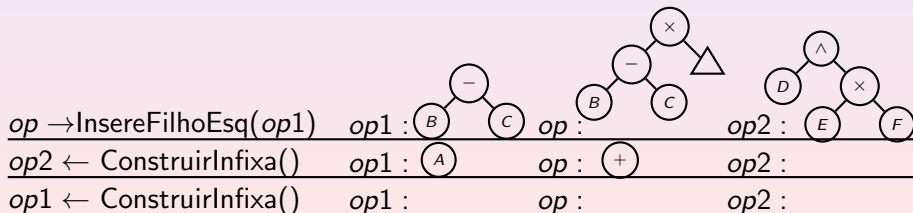


Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

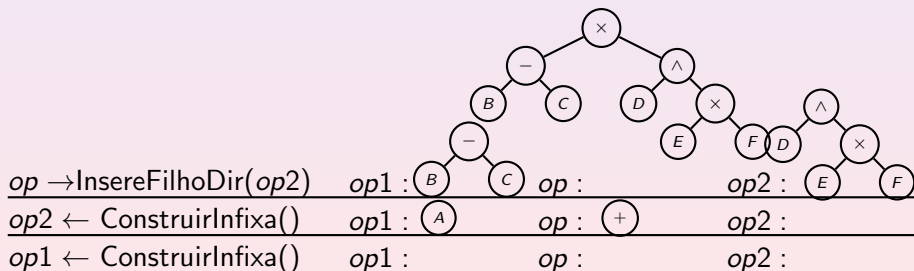


Construção - Infixa $(A+((B-C)\times(D\wedge(E\times F))))$ 

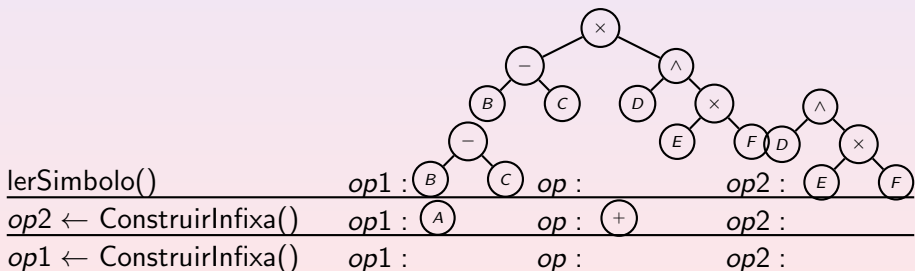
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



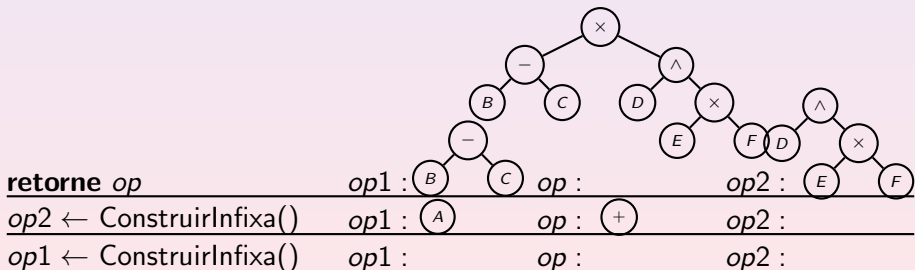
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



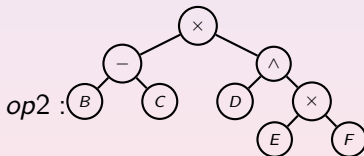
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



$op2 \leftarrow \text{ConstruirInfixa}()$

$op1 : (A)$

$op : (+)$

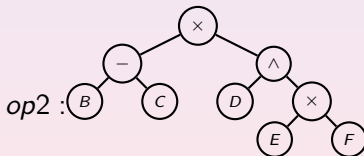
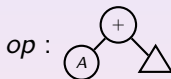
$op1 \leftarrow \text{ConstruirInfixa}()$

$op1 :$

$op :$

$op2 :$

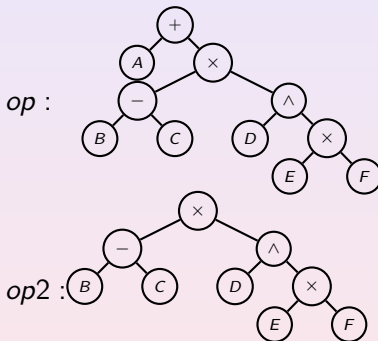
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



$op \rightarrow \text{InsereFilhoEsq}(op1)$ $op1 : (A)$

$op1 \leftarrow \text{ConstruirInfixa}()$ $op1 :$ $op :$ $op2 :$

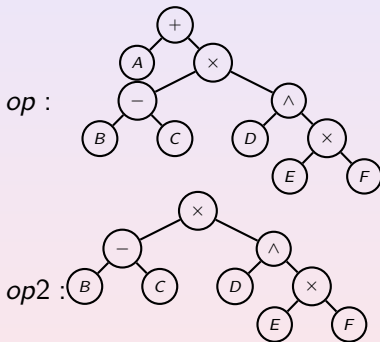
Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



$op \rightarrow \text{InsereFilhoDir}(op2)$ $op1 : (A)$

$op1 \leftarrow \text{ConstruirInfixa}()$ $op1 :$ $op :$ $op2 :$

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



lerSimbolo()

op1 : (A)

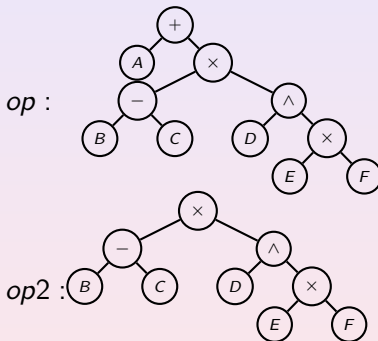
op1 \leftarrow ConstruirInfixa()

op1 :

op :

op2 :

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



retorne *op*

op1 : (A)

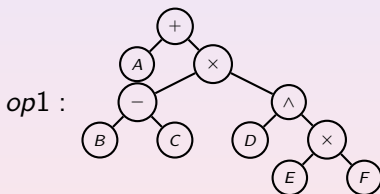
op1 \leftarrow ConstruirInfixa()

op1 :

op :

op2 :

Construção - Infixa $(A+((B-C)\times(D\wedge(E\times F))))$

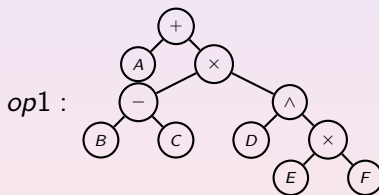


op1 \leftarrow ConstruirInfixa()

op :

op2 :

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

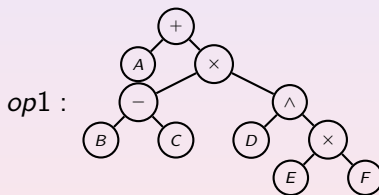


$S \leftarrow \text{lerSimbolo}()$

op :

op2 :

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$

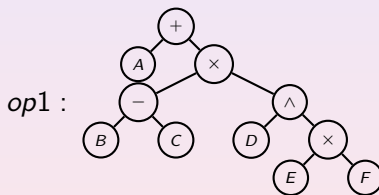


se($S = \emptyset$) \rightarrow então

op :

op2 :

Construção - Infixa $(A + ((B - C) \times (D \wedge (E \times F))))$



retorne *op1*

op :

op2 :

Códigos de Huffman

- Representa uma técnica de compressão de dados que pode atingir valores entre 20 e 90%.
- É aplicado em arquivos de símbolos (texto) onde existe uma distribuição diferenciada na frequência com que cada símbolo aparece.
- Codifica-se os símbolos com tamanhos distintos de bits. Símbolos mais frequentes recebem menos bits, símbolos menos frequentes recebem mais bits.
- Na média o número de bits do arquivo será menor.

Exemplo do código de Huffman

- Considere o alfabeto $C = \{a, b, c, d, e, f\}$.
- Um dado arquivo possui a frequência indicada na tabela abaixo para os caracteres do alfabeto.
- Também na tabela estão indicadas duas possíveis codificações para cada objeto, uma de tamanho fixo e outra de tamanho variável.

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
Frequência (milhares)	45	13	12	16	9	5
Código: <i>Tamanho Fixo</i>	000	001	010	011	100	101
Código: <i>Tamanho Variável</i>	0	101	100	111	1101	1100

- Qual o tamanho do arquivo para cada uma das codificações?

Calculando o custo para cada tipo de codificação

- Codificação com códigos de tamanho fixo:

$$Totaldebits = 3 \times 100.000 = 300.000bits$$

- Codificação com códigos de tamanho variável:

$$\underbrace{1 \times 45}_a + \underbrace{3 \times 13}_b + \underbrace{3 \times 12}_c + \underbrace{3 \times 16}_d + \underbrace{4 \times 9}_e + \underbrace{4 \times 5}_f = 224.000bits$$

- Há um ganho de aproximadamente 25% se utilizarmos a codificação de tamanho variável.

O método

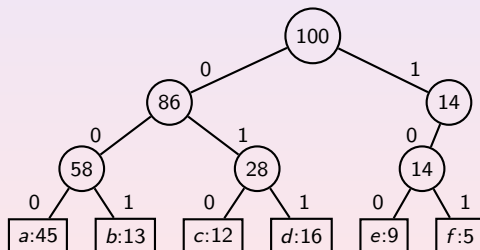
- A solução implica no uso de uma “codificação livre de prefixo”.
- Em uma codificação livre de prefixo, para quaisquer símbolos distintos i e j codificados, a codificação de i não é prefixo da codificação de j .
- No exemplo anterior, usando a codificação variável para a palavra “abc” obtemos: 0101100.
 - O único caracter começado com 0 e que portanto utiliza somente um bit é o 'a';
 - A seqüência 101 define o caracter 'b' e não há qualquer outro caracter que inicie com o código 101;
 - O restante 100 representa o caracter 'c'.

Representando o código

- Precisamos identificar uma estrutura que associe um código ao caracter, de forma que na decodificação encontremos facilmente o símbolo utilizando o código fornecido.
- Uma solução é utilizar uma árvore binária:
 - Um filho esquerdo está associado a um bit 0.
 - Um filho direito a um bit 1.
 - Nas folhas se encontram os símbolos.
 - O código lido 0 ou 1 faz com que na navegação na árvore chegue a um símbolo.
 - Ao achar um símbolo o próximo código é aplicado a partir do raiz.

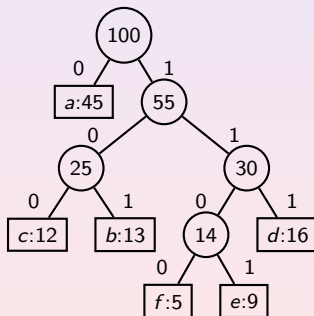
Código de tamanho fixo na forma de árvore

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
Frequência (milhares)	45	13	12	16	9	5
Código	000	001	010	011	100	101



Código de tamanho variável na forma de árvore

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
Frequência (milhares)	45	13	12	16	9	5
Código	0	101	100	111	1101	1100



Propriedades da árvore de código

- Cada código é livre de prefixo: Só há um único caminho para chegar a uma folha, que não passa por outra folha, assim o código de um símbolo não é um prefixo de outro símbolo.
- Uma codificação ótima deve ser representado por uma árvore binária cheia, cada vértice interno tem dois filhos. Seja uma codificação com um vértice interno que só tenha um filho:
 - Se o filho for uma folha. Podíamos colocar esta folha no lugar do vértice e economizaríamos um bit para o código deste símbolo.
 - Se o filho for outro vértice. A partir deste vértice buscamos uma folha, colocamos esta folha como segundo filho do vértice. Economizaríamos no mínimo um bit.
- Buscamos uma árvore binária cheia com $|C|$ folhas (o tamanho do alfabeto) e $|C| - 1$ vértices internos.

Entendendo a proposta de solução

- Começar com $|C|$ árvores folhas isoladas e realizar seqüencialmente $|C - 1|$ operações de agregação, agregando duas árvores a um novo vértice raiz comum. O raiz passa a ter como “peso” a soma dos custos de cada árvore agregada.
- A escolha do par de árvores que serão agregadas dependerá do custo de cada árvore. As duas árvores de menor custo serão escolhidas.
- O raiz de uma árvore carrega como informação o custo da árvore.

O algoritmo de Huffman

Entrada: Conjunto de caracteres de C e a frequências f de cada caracter

Saída: Raiz da árvore binária representando codificação ótima livre de prefixo

Algoritmo HUFFMAN(C)

$n \leftarrow |C|$

$Q \leftarrow C$ \triangleright Q é fila de árvores ordenadas por custo

para $i \leftarrow 1$ **até** $n - 1$ **faça**

$z \leftarrow$ **novo** *Arvore*

$z.esq \leftarrow Extrai_Minimo(Q)$

$z.dir \leftarrow Extrai_Minimo(Q)$

$z.info = z.esq.info + z.dir.info$

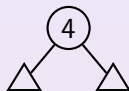
Inserir(Q, z)

retorne *Extrai_Minimo*(Q)

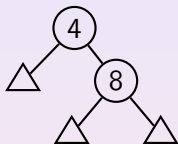
Árvore Binária de Busca

- Árvore Binária de Busca utiliza uma classificação de ordem ao inserir um elemento.
- Dado um raiz da árvore ou subárvore:
 - Todos os descendentes do lado esquerdo tem valores menor ou igual ao valor da raiz.
 - Todos descendentes do lado direito tem valores maior que o da raiz.
- A inserção de um elemento se dá a partir do raiz da árvore.
 - Se o elemento for menor ou igual, tenta-se inserir no filho esquerdo.
 - Se o elemento for maior, tenta-se inserir no filho direito.
- O elemento é inserido quando atinge uma árvore vazia.
- O nome se dá pois para encontrar se um elemento está na árvore a busca é feita de forma semelhante a uma busca binária em uma sequência ordenada.

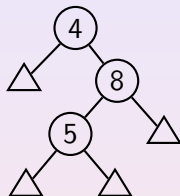
Inserindo Elementos: 4 8 5 0 2 8 1 3 5 9



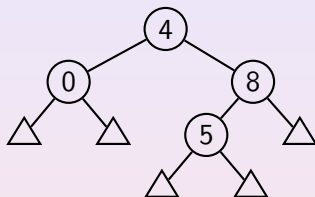
Inserindo Elementos: 4 8 5 0 2 8 1 3 5 9



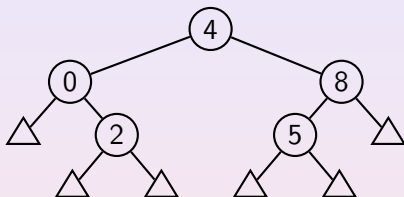
Inserindo Elementos: 4 8 5 0 2 8 1 3 5 9



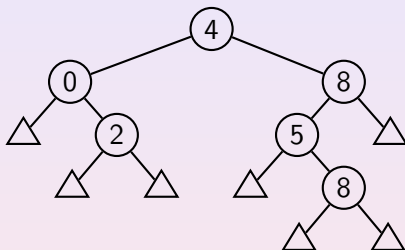
Inserindo Elementos: 4 8 5 0 2 8 1 3 5 9



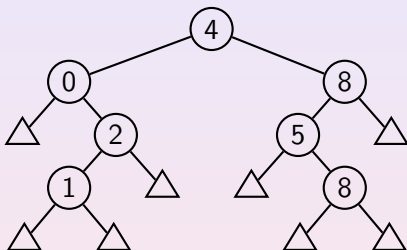
Inserindo Elementos: 4 8 5 0 2 8 1 3 5 9



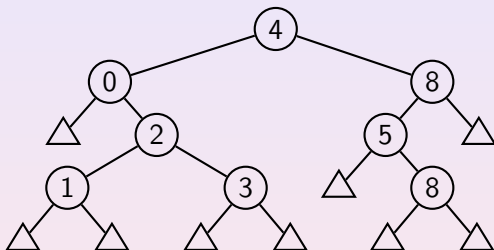
Inserindo Elementos: 4 8 5 0 2 8 1 3 5 9



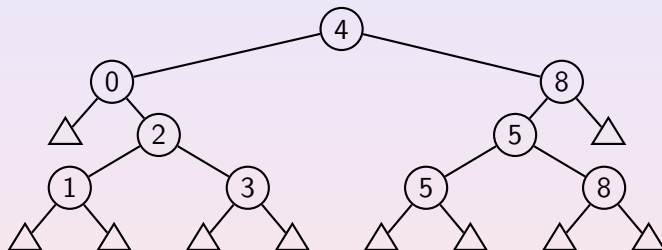
Inserindo Elementos: 4 8 5 0 2 8 1 3 5 9

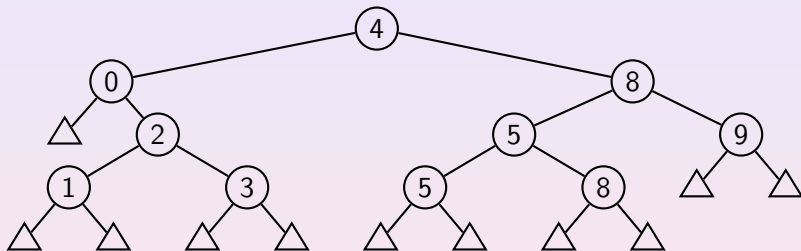


Inserindo Elementos: 4 8 5 0 2 8 1 **3** 5 9



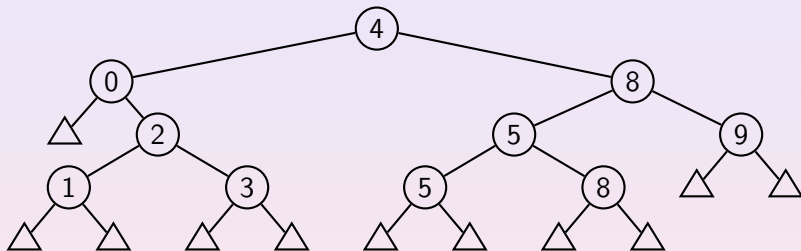
Inserindo Elementos: 4 8 5 0 2 8 1 3 5 9





Como ficaria uma busca em ordem nesta árvore?

Inserindo Elementos: 4 8 5 0 2 8 1 3 5 9



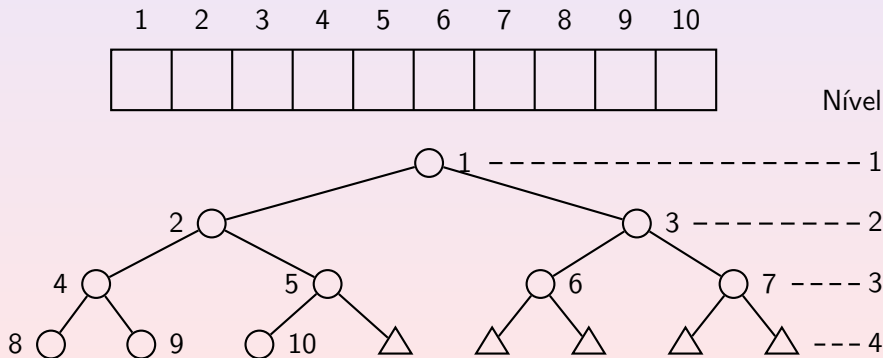
Como ficaria uma busca em ordem nesta árvore: 0 1 2 3 4 5 5 8 8 9

Ordenação por Inserção em árvore Binária

- São n elementos inseridos.
- Cada elemento inserido percorre a altura da árvore.
- No pior caso a altura da árvore é n (sequência já ordenada)
 $\rightarrow O(n^2)$
- No melhor caso a árvore binária é completa, a altura é $\log n \rightarrow O(n \log n)$
- No caso médio a altura da árvore é proporcional a $\log n \rightarrow O(n \log n)$

Heap

- Heap é uma árvore binária quase completa.
- Por ser uma árvore quase completa, o Heap é uma árvore implementada em vetor.
- Veja a ordem dos elementos em um heap:



Definição

- Considere um vetor $A[1, \dots, n]$ representando um Heap.
- Cada posição do vetor corresponde a um nó da árvore do Heap.
- O pai de um nó i é $\lfloor i/2 \rfloor$.
- Um nó i tem $2i$ como filho esquerdo e $2i + 1$ como filho direito.
- Naturalmente, o nó i tem filho esquerdo apenas se $2i \leq n$. e tem filho direito apenas se $2i + 1 \leq n$.
- Um nó i é uma folha se não tem, ou seja, se $2i > n$.
- As folhas são $\lfloor n/2 \rfloor + 1, \dots, n - 1, n$.

Níveis do Heap

- Cada nível p , exceto talvez o último, tem exatamente 2^{p-1} nós, e esses são:
 $2^{p-1}, 2^{p-1} + 1, 2^{p-1} + 2, \dots, 2^p - 1$
- O nó i pertence ao nível $\lfloor \log i \rfloor$.
- O número total de níveis é $1 + \lfloor \log n \rfloor$

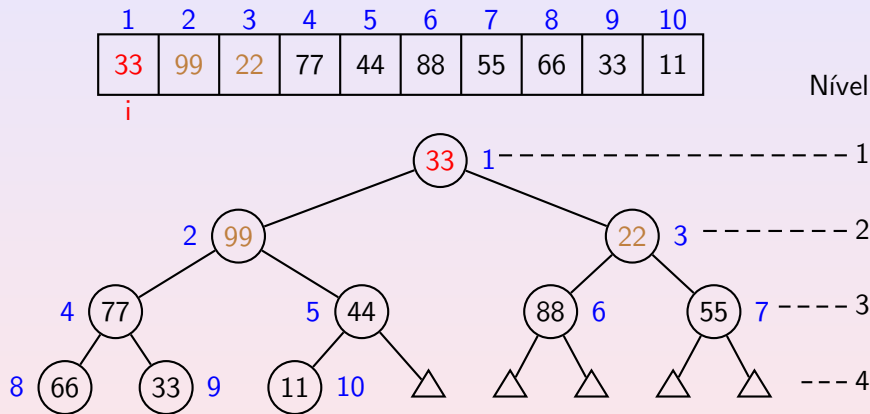
Max-Heap e Min-Heap

- Um nó i satisfaz a propriedade de (max)-heap se:
 - $A[\lfloor i/2 \rfloor] \geq A[i]$ (ou seja, pai \geq filho).
- Uma árvore binária completa é um (max)-heap se todo nó distinto da raiz satisfizer a propriedade de (max)-heap.
- O (máx)imo ou (mai)or elemento de um (max)-heap está na raiz da árvore.
- De forma análoga é definido o Min-Heap (basta trocar max por min).

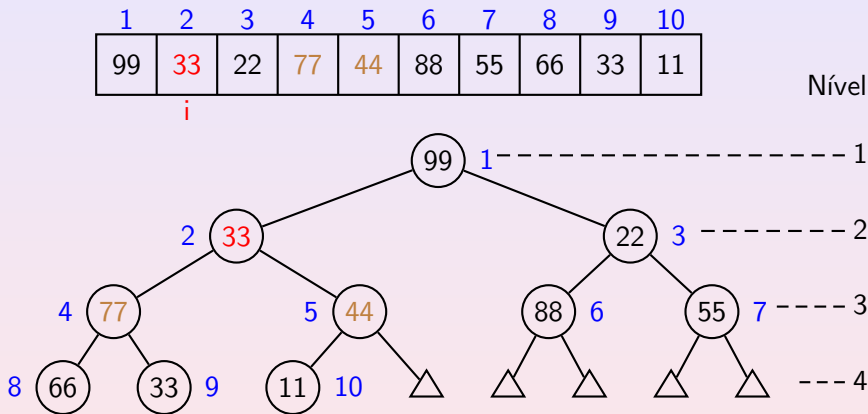
Construindo um Heap Máximo

- São Duas Etapas:
 - 1 Max-Heapfy (Joga um elemento menor para uma folha)
 - Pegamos um elemento na posição i e comparamos com seus filhos: $2i$ e $2i + 1$.
 - Trocamos com o maior.
 - Continuamos recursivamente a partir da nova posição deste elemento.
 - 2 Build-Max-Heap (Construir o Heap-Máximo)
 - A partir do primeiro elemento não folha (de trás para frente): $n/2$
 - Iterativamente aplicamos o algoritmo Max-Heapfy até o elemento na posição 1.
- Garantimos a cada iteração que os pais são sempre os maiores e que os menores elementos sempre são jogados para as folhas.

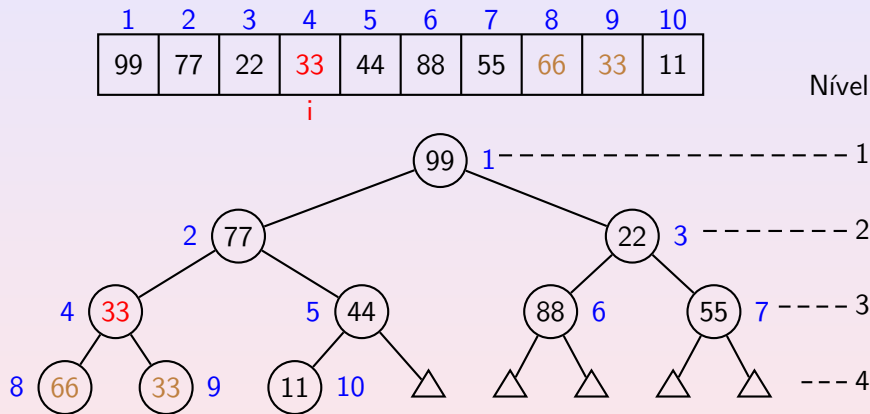
Max-Heapfy



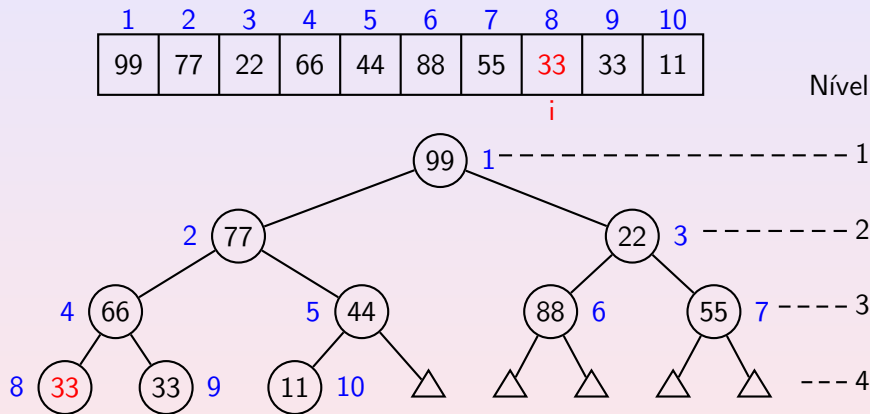
Max-Heapfy



Max-Heapfy



Max-Heapfy



Este Heap não é máximo, mas o elemento “33” não tem descendentes menores que ele.

Algoritmo Max-Heapfy

Algoritmo MAX_HEAPFY(A, n, i)

$e \leftarrow 2 * i$

$d \leftarrow 2 * i + 1$

se $e \leq n$ **e** $A[e] > A[i]$ **então**

$max \leftarrow e$

senão

$max \leftarrow i$

se $d \leq n$ **e** $A[d] > A[max]$ **então**

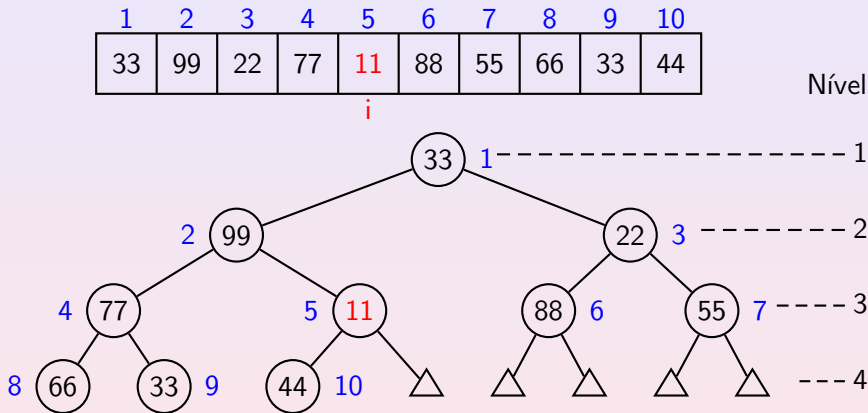
$max \leftarrow d$

se $max \neq i$ **então**

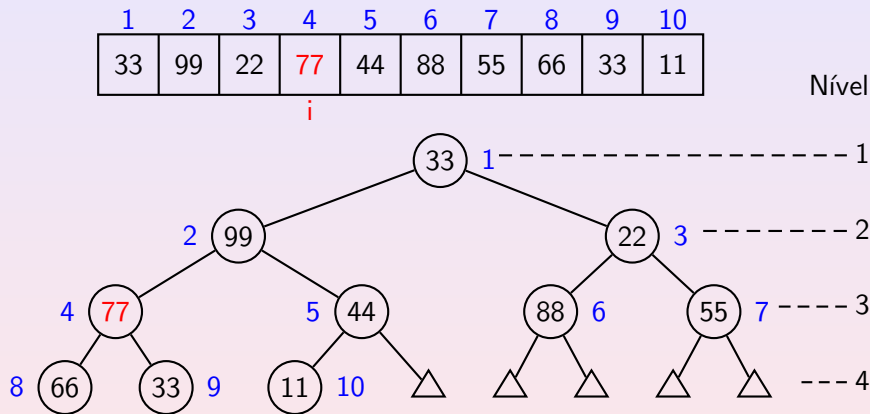
$A[i] \leftrightarrow A[max]$

Max_Heapfy(A, n, max)

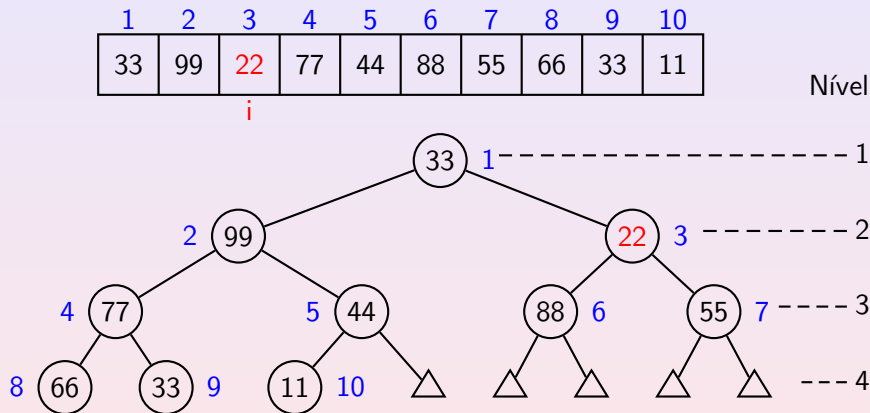
Build-Max-Heap



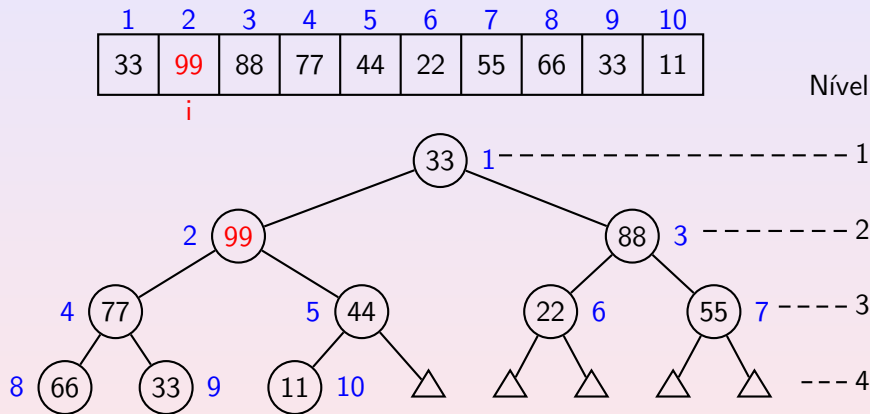
Build-Max-Heap



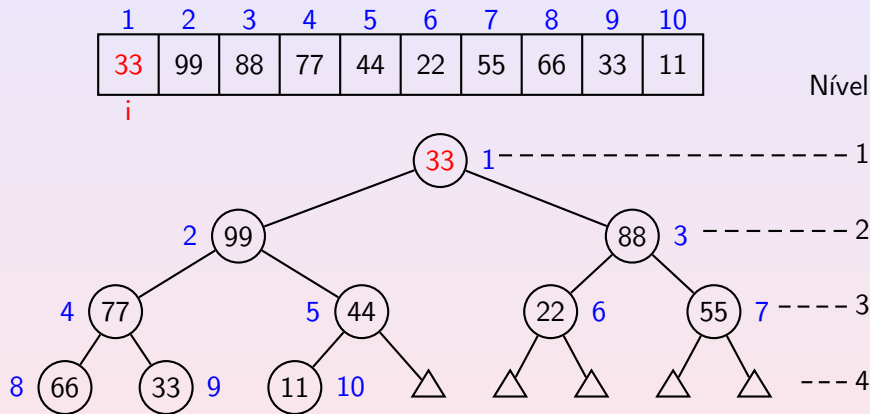
Build-Max-Heap



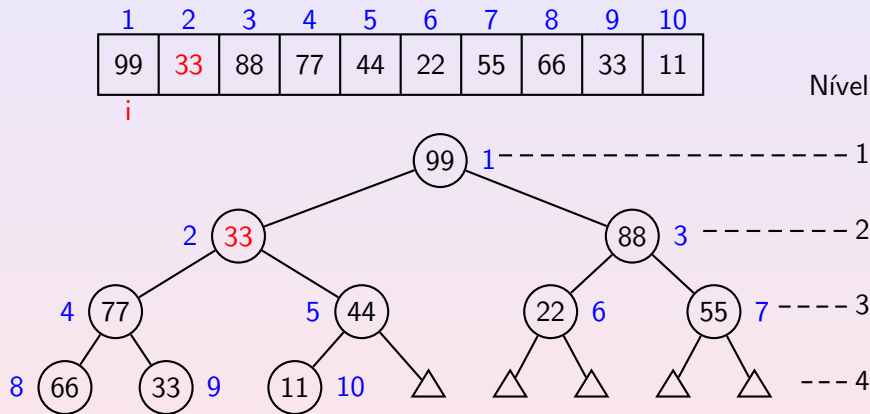
Build-Max-Heap



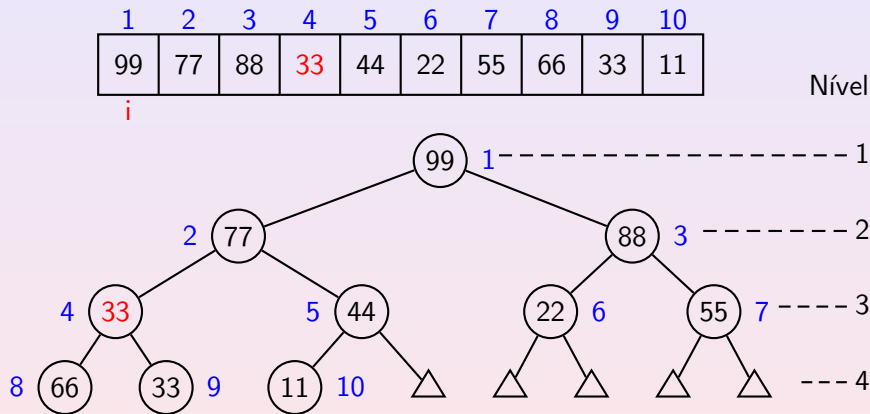
Build-Max-Heap



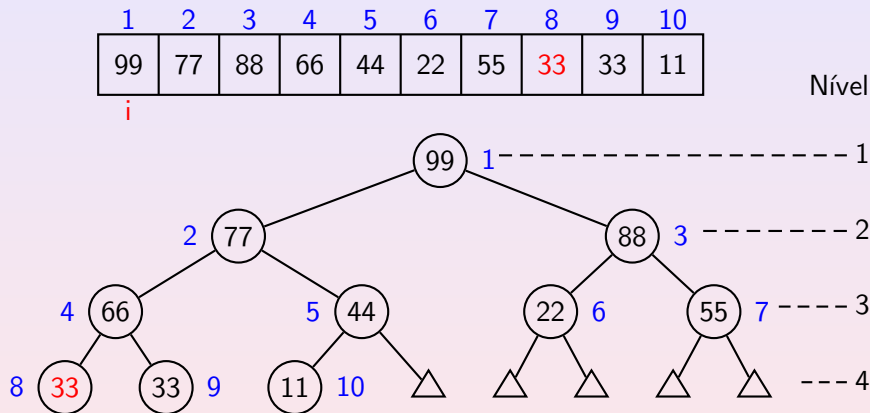
Build-Max-Heap



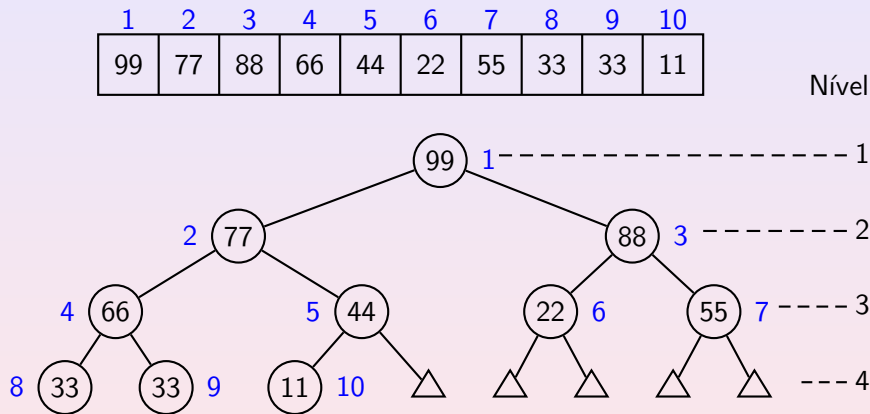
Build-Max-Heap



Build-Max-Heap



Build-Max-Heap



Heap Máximo atingido

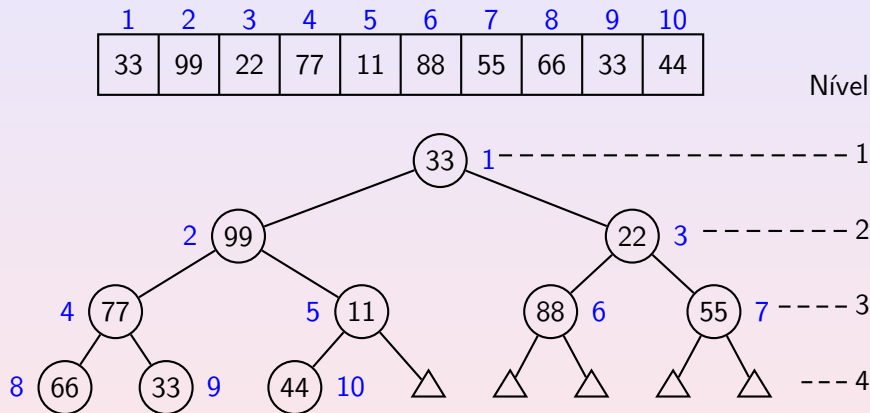
Algoritmo Build_Max_Heap

```
Algoritmo BUILD_MAX_HEAP( $A, n$ )  
  para  $i \leftarrow \lfloor n/2 \rfloor$  até 1 faça  
    Max_Heapfy( $A, n, i$ )
```

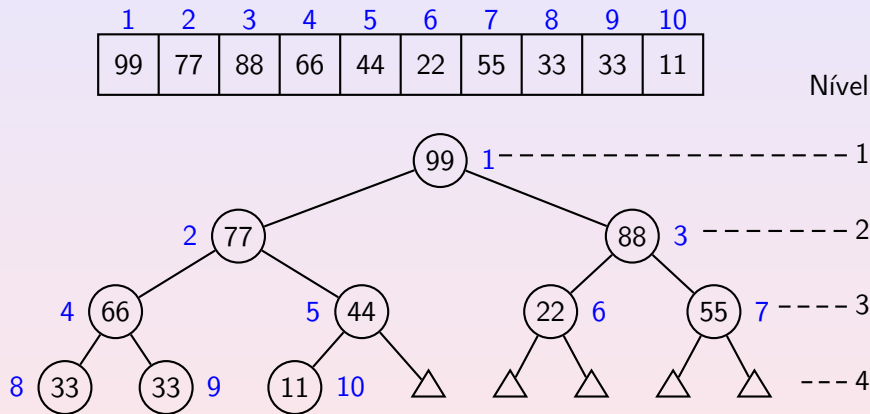
Heap Sort

- O algoritmo de ordenação por Seleção em estrutura Heap segue as seguintes etapas:
 - 1 Primeiro construímos um Heap Máximo.
 - 2 O primeiro elemento é o maior da sequência. Trocamos este com o último, o último já está na posição. Vamos considerar agora a sequência sem este elemento.
 - 3 Aplicamos um Max-Heapfy nos $n - 1$ elementos restantes e o elemento pequeno que ficou na raiz é jogado para uma folha.
 - 4 Repete-se o processo até que sobre somente um elemento na sequência considerada.

Ordenação: Build-Max-Heap

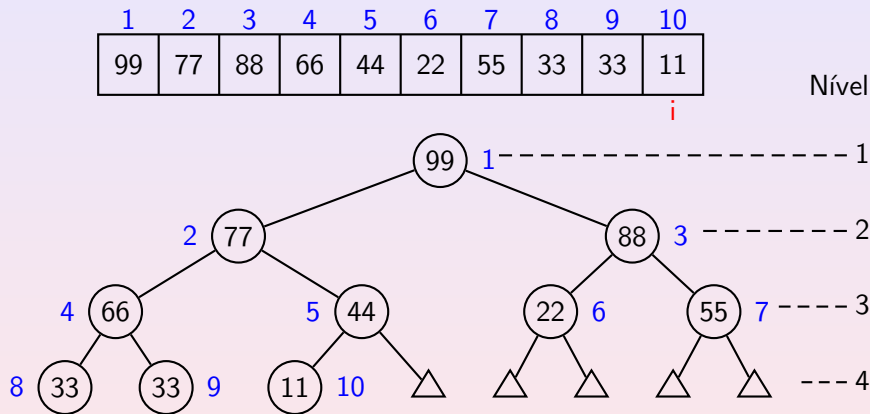


Ordenação: Build-Max-Heap

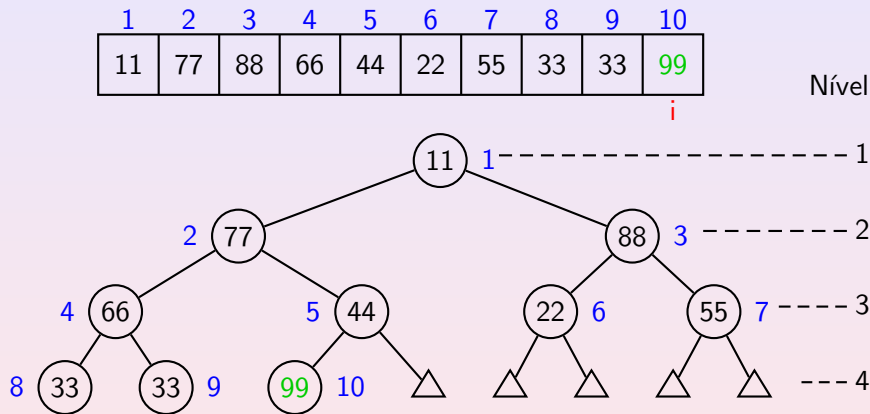


Heap Máximo atingido

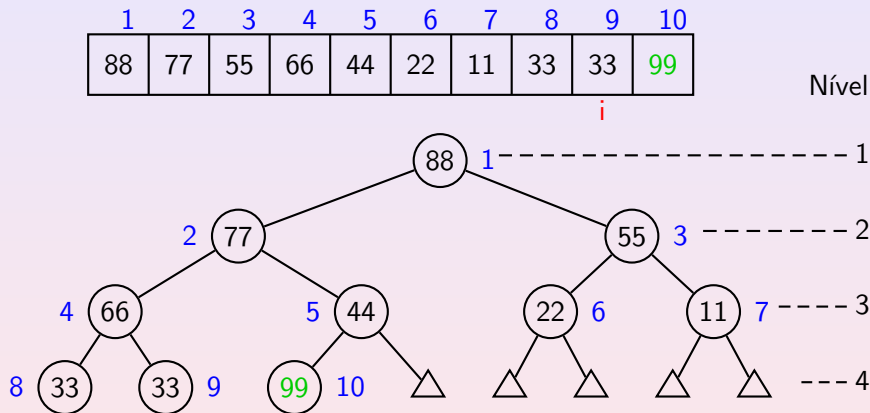
Ordenação: Troca de Elementos e Max-Heapfy



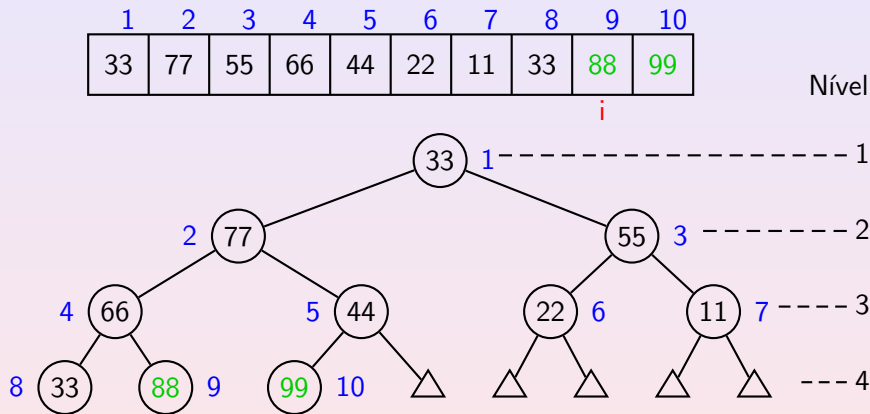
Ordenação: Troca de Elementos e Max-Heapfy



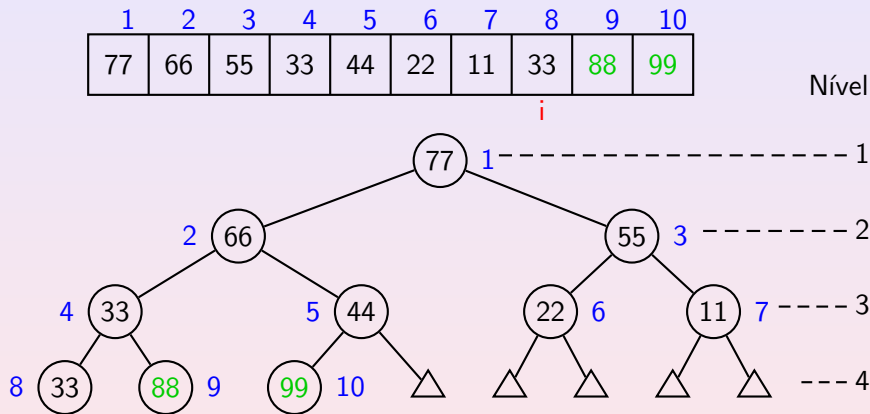
Ordenação: Troca de Elementos e Max-Heapfy



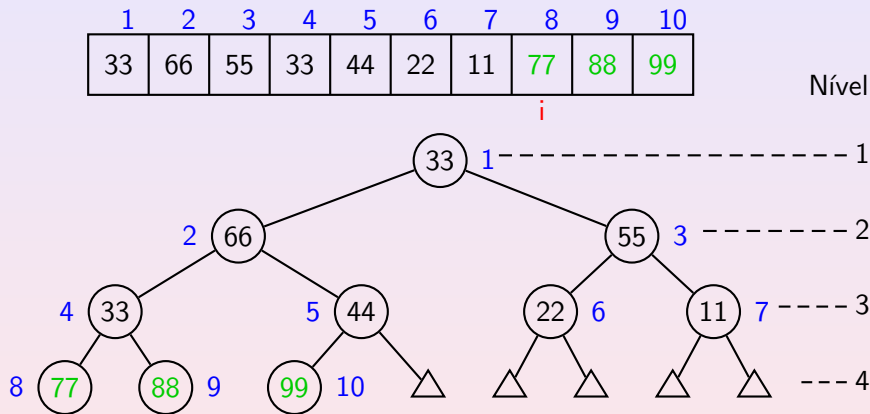
Ordenação: Troca de Elementos e Max-Heapfy



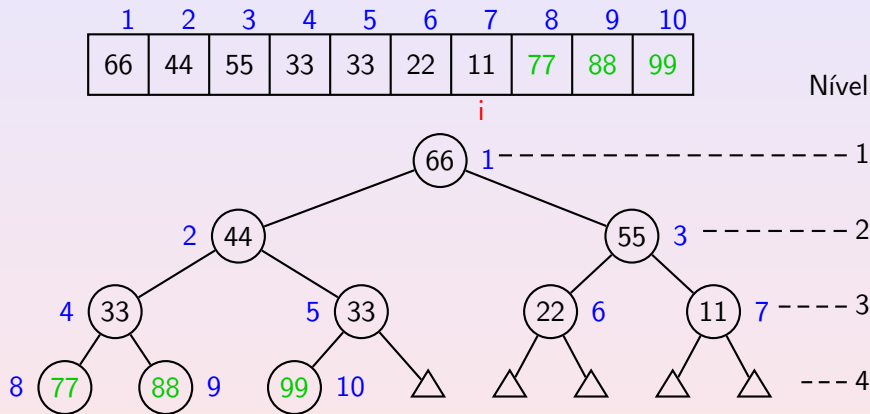
Ordenação: Troca de Elementos e Max-Heapfy



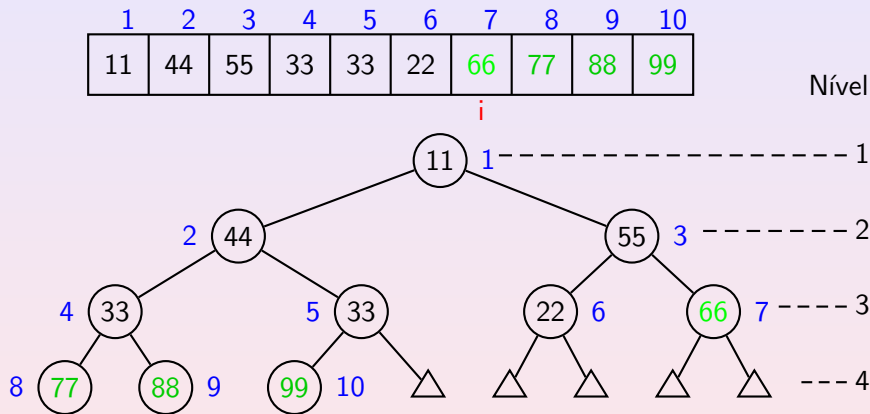
Ordenação: Troca de Elementos e Max-Heapfy



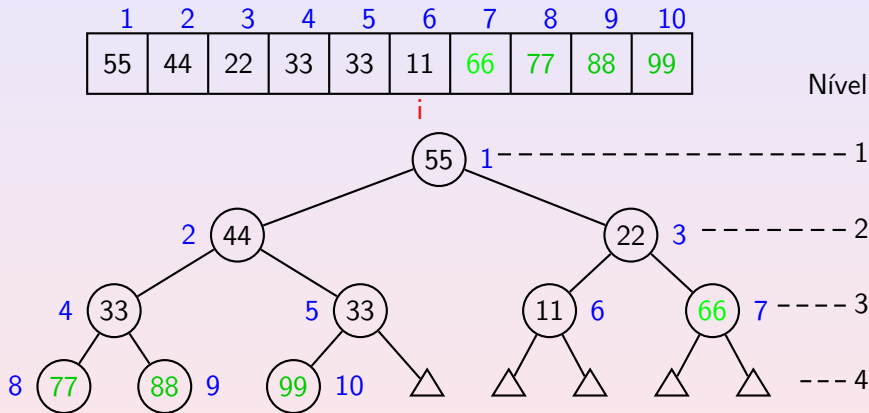
Ordenação: Troca de Elementos e Max-Heapfy



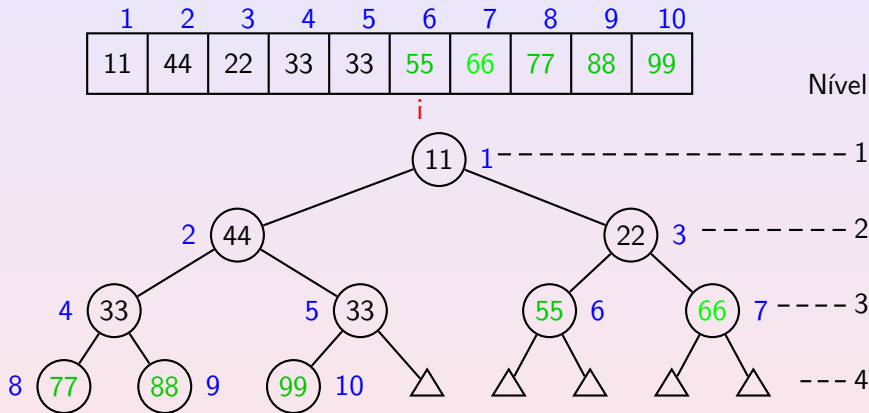
Ordenação: Troca de Elementos e Max-Heapfy



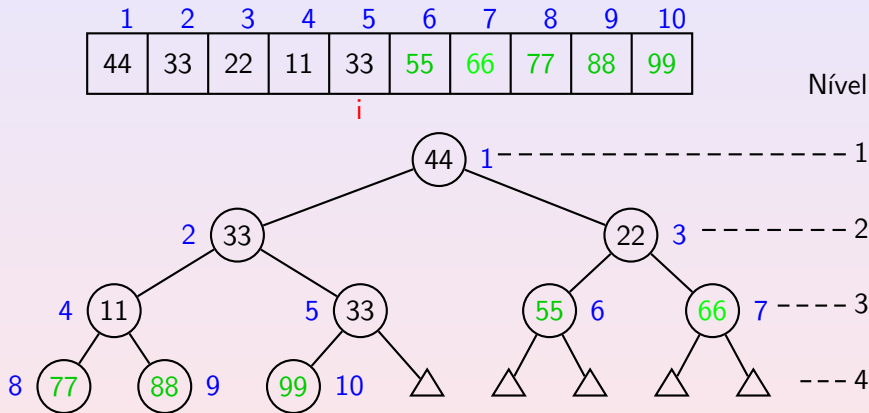
Ordenação: Troca de Elementos e Max-Heapfy



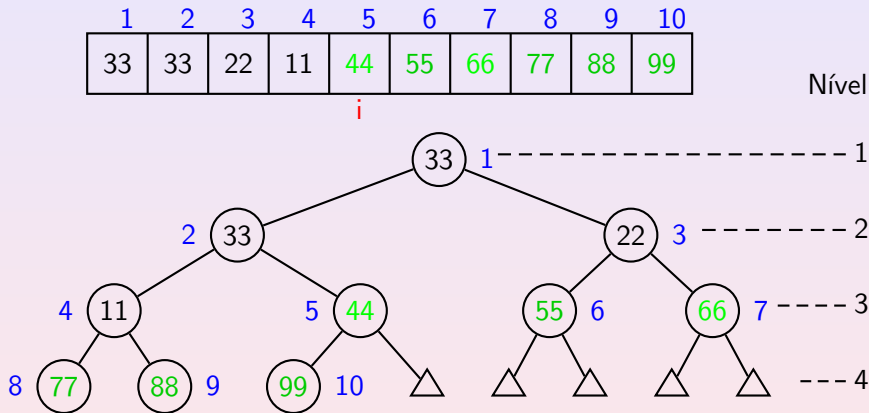
Ordenação: Troca de Elementos e Max-Heapfy



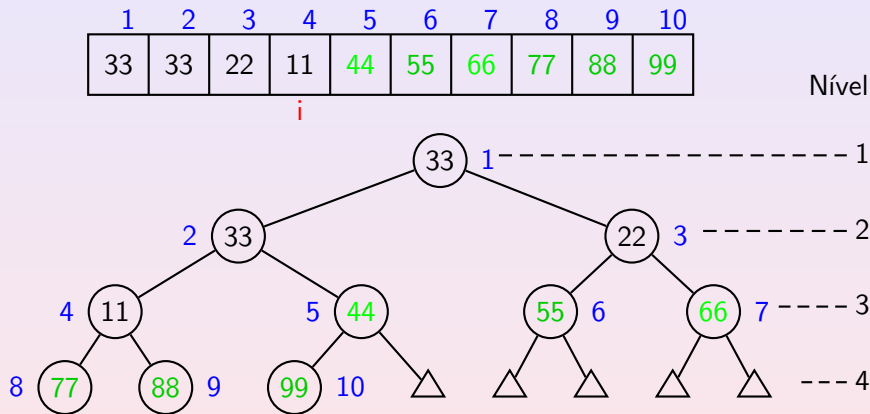
Ordenação: Troca de Elementos e Max-Heapfy



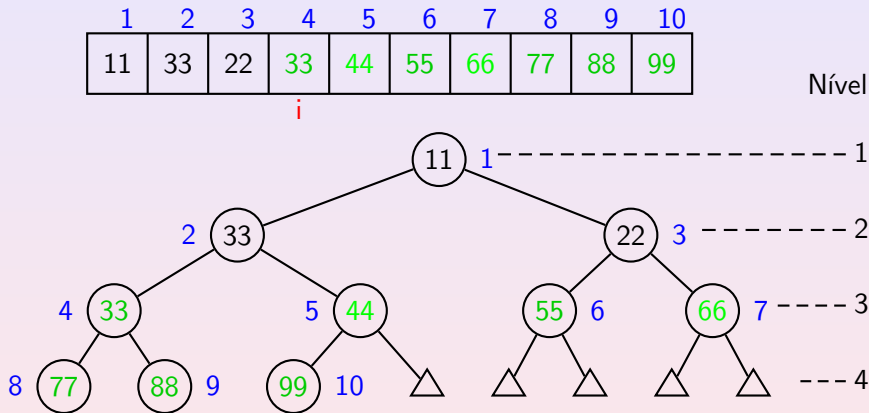
Ordenação: Troca de Elementos e Max-Heapfy



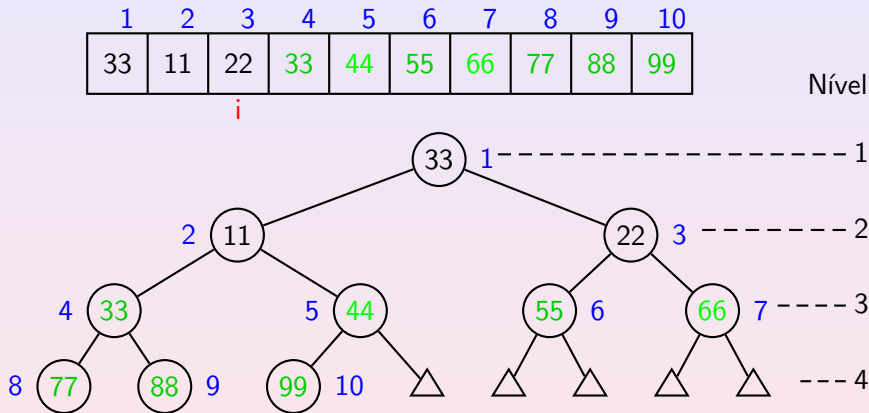
Ordenação: Troca de Elementos e Max-Heapfy



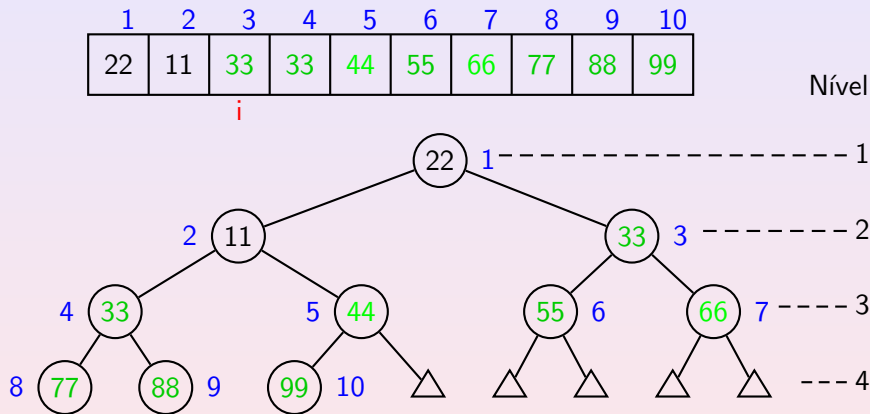
Ordenação: Troca de Elementos e Max-Heapfy



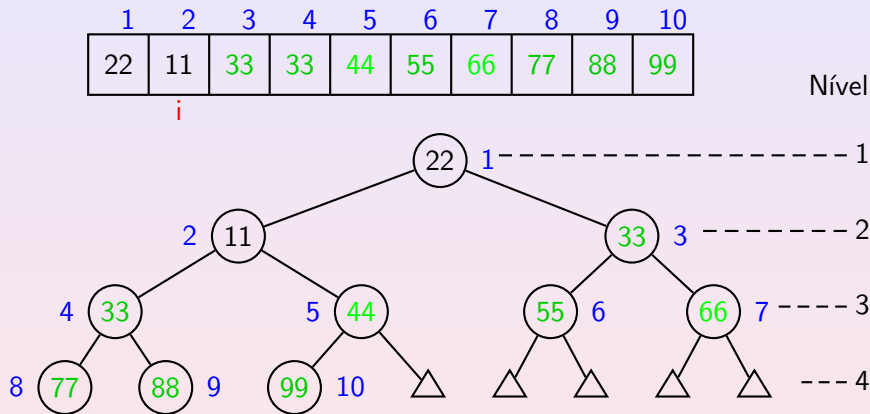
Ordenação: Troca de Elementos e Max-Heapfy



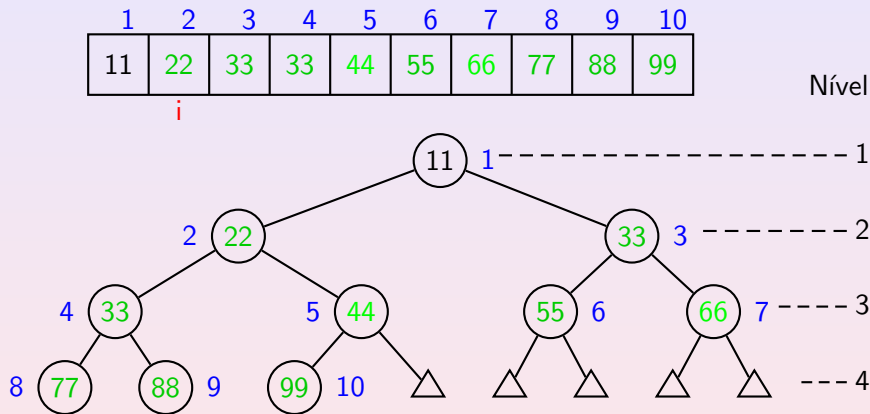
Ordenação: Troca de Elementos e Max-Heapfy



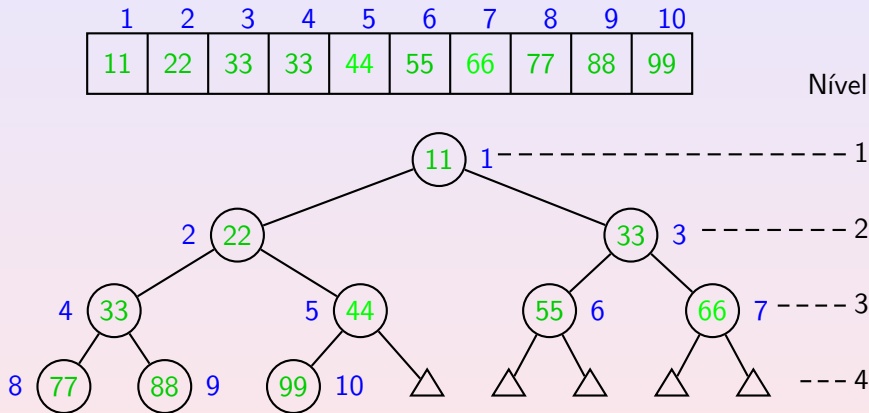
Ordenação: Troca de Elementos e Max-Heapfy



Ordenação: Troca de Elementos e Max-Heapfy



Ordenação: Troca de Elementos e Max-Heapfy



Sequência classificada em ordem crescente

Algoritmo Heap-Sort

```
Algoritmo HEAP_SORT( $A, n$ )  
  Build_Max_Heap( $A, n$ )  
  para  $i \leftarrow n$  até 2 faça  
     $A[1] \leftrightarrow A[i]$   
    Max_Heapfy( $A, i - 1, 1$ )
```

Algoritmo Heap-Sort - Custo

- O algoritmo Max-Heapfy executa em $O(\log n)$, que é a altura da árvore.
- O algoritmo Build-Max-Heap possui um cálculo mais complexo de complexidade, mas é $O(n)$.
- O Algoritmo Heap-Sort aplica n vezes o Max-Heapfy, portanto é $O(n \log n)$.

Considerações finais sobre a estrutura Heap

- É uma estrutura de árvore binária que pode ser mantida em vetores.
- Não é simples de manutenção em alguma lista dinâmica. Se quiser uma estrutura dinâmica, então que seja árvore, mesmo.
- O Heap Máximo, por manter sempre o maior elemento no início, é eficaz para representação de filas com prioridades.
 - A fila inicialmente deve ser um Heap-Máximo.
 - Insere-se o elemento no final da fila.
 - Ele vai trocando de posição com o pai, enquanto for maior que o mesmo.
 - No final, continuaremos com um Heap-Máximo.
- A inserção de elemento na fila é $O(\log n)$ e a retirada é $O(1)$ (com o custo de andar a fila no vetor, a não ser que trate de forma circular).

Realizar a 8ª lista de exercícios