





## Aula 28: Listas de Prioridades

- ⇒ Conceitos básicos
- ⇒ Definição de heap
- ⇒ Alteração de prioridades

## Listas de Prioridades

### Motivação:

-  Os dados possuem prioridades.
-  A prioridade de um dado pode variar ao longo do tempo.
-  Quando desejado, deve-se selecionar o dado de maior prioridade.

 A situação acima modela uma grande quantidade de problemas.

## Lista de Prioridades

- ➡ É uma tabela, na qual a cada um de seus dados está associada uma prioridade.  
Em geral, a prioridade é um valor numérico.
- ➡ Operações básicas:
  - ▬ seleção do elemento de maior prioridade
  - ▬ inserção de um novo dado
  - ▬ remoção do dado de maior prioridade
- ➡ Também desejada a operação:
  - ▬ alteração de prioridade de um dado
- ➡ Objetivo: Descrever uma estrutura de dados que realize as operações acima, de maneira eficiente.

## Implementação

- ➡ Métodos: por
- ▢ lista não ordenada
  - ▢ lista ordenada
  - ▢ heap
- ➡ Para cada um desses métodos será avaliada a complexidade de cada uma das seguintes operações:
- ▢ seleção (busca)
  - ▢ inserção
  - ▢ remoção
  - ▢ alteração
  - ▢ construção

## Implementação por lista não-ordenada

➡ Os dados formam uma lista não ordenada com  $n$  nós.

➡ Complexidades:

seleção	$O(n)$
inserção	$O(1)$
remoção	$O(n)$
alteração	$O(n)$
construção	$O(n)$

## Implementação por lista ordenada

- ⇒ Os dados formam uma lista ordenada, em ordem decrescente de suas prioridades. As operações de seleção e remoção referem-se sempre ao dado de maior prioridade. Para construir a lista, é necessário ordená-la.
- ⇒ Complexidades:
- |            |               |
|------------|---------------|
| seleção    | $O(1)$        |
| inserção   | $O(n)$        |
| remoção    | $O(1)$        |
| alteração  | $O(n)$        |
| construção | $O(n \log n)$ |

## Implementação por heap

➡ Um heap é uma lista linear composta de elementos com chaves  $s_1, \dots, s_n$ , satisfazendo

$$s_i \leq s_{\lfloor i/2 \rfloor}, 1 \leq i \leq n.$$

➡ A chave representa a prioridade do elemento.

➡ Os heaps formam uma estrutura conveniente para implementar listas de prioridades.

➡ Exemplo

95 60 78 39 28 66 70 33

## Exercício

⇒ Verifique se as seguintes listas constituem heaps.

(i) 33 32 28 31 26 29 25 30 27

(ii) 33 32 28 31 29 26 25 30 27

Tempo: 2 minutos



## Solução

	i	1	2	3	4	5	6	7	8	9
(i)	$s_i$	33	32	28	31	26	29	25	30	27
(ii)	$s_i$	33	32	28	31	29	26	25	30	27

⇒ (i)  $s_6 > s_3$ , não é heap.

⇒ (ii) É heap.

## Heaps e árvores binárias

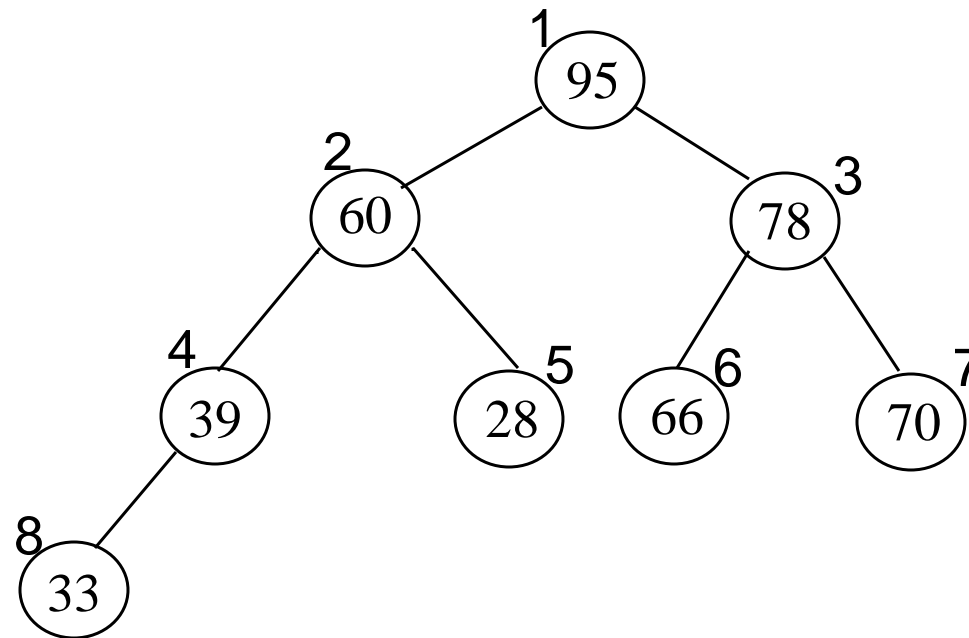
- ➡ Um heap pode ser visualizado através de uma árvore binária completa  $T$ .
- ➡ Os nós de  $T$  são numerados seqüencialmente, da raiz para os níveis mais altos, da esquerda para a direita.
- ➡ Cada nó de  $T$  corresponde a uma chave, sendo o rótulo do nó igual à prioridade da chave.
- ➡ Os nós do último nível de  $T$  são preenchidos da esquerda para a direita.

## Heaps e árvores binárias

- ➡ A propriedade  $s_i \leq s_{\lfloor i/2 \rfloor}$  do heap é equivalente a dizer que cada nó de  $T$  possui rótulo maior ou igual aos rótulos de seus filhos, se existirem.
- ➡ A árvore pode ser representada simplesmente pela lista, requerendo apenas tamanho  $n$ .

## Exemplo

i	1	2	3	4	5	6	7	8
$s_i$	95	60	78	39	28	66	70	33



## Exercício

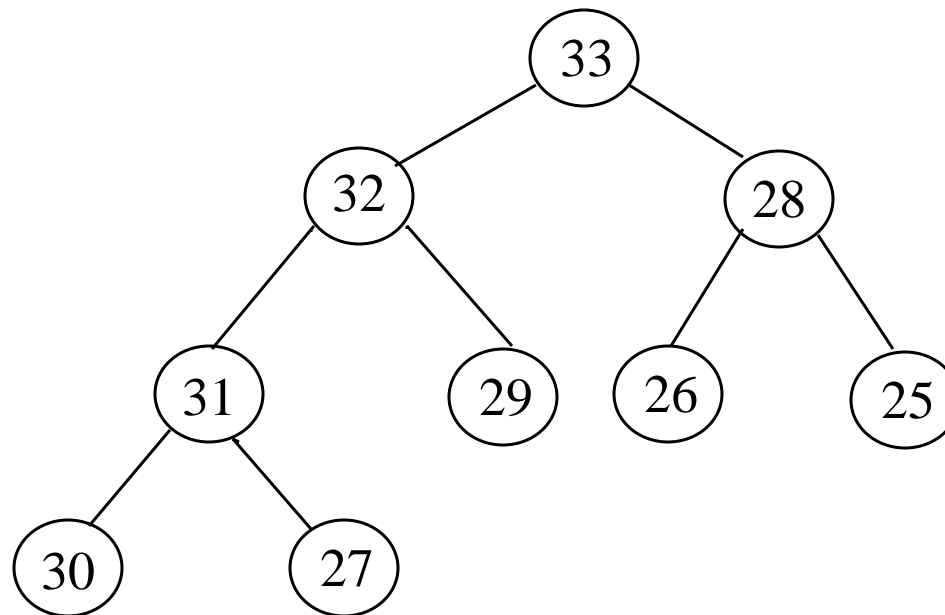
➡ Desenhar a árvore binária correspondente ao heap

33   32   28   31   29   26   25   30   27

Tempo: 1 minuto

## Solução

➡ 33 32 28 31 29 26 25 30 27



## Operações básicas em heaps

### ➡ Complexidades:

Seleção:  $O(1)$

Inserção:  $O(\log n)$

Remoção:  $O(\log n)$

Alteração:  $O(\log n)$

Construção:  $O(n)$

➡ A condição  $s_i \leq s_{\lfloor i/2 \rfloor}$  implica que o elemento de maior prioridade seja sempre o primeiro da ordenação, isto é, a raiz da árvore.

Logo, a seleção pode ser realizada em  $O(1)$  passos.

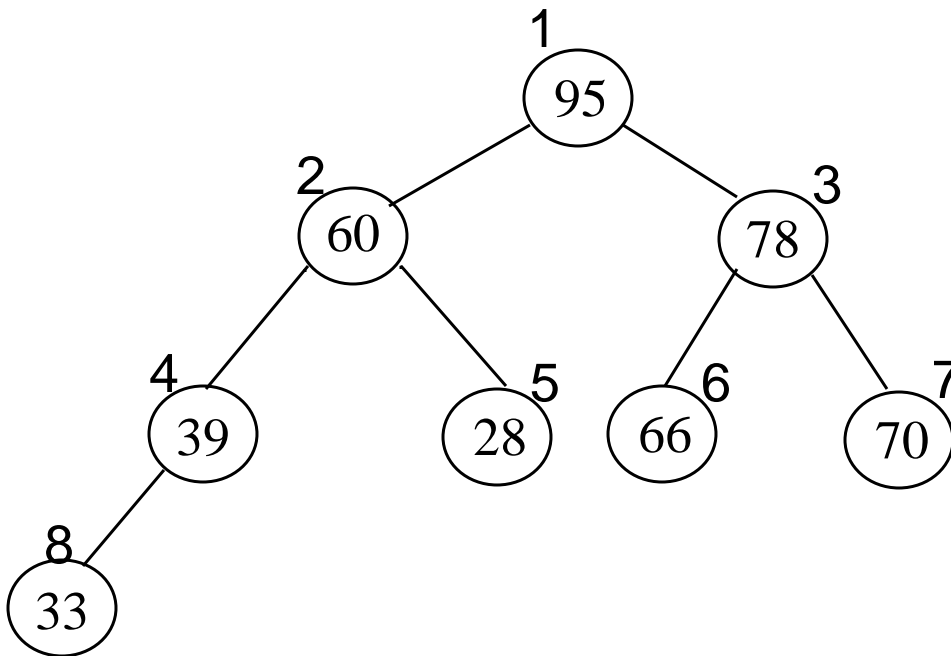
## Alteração de prioridades

- ⇒ Aumento ou diminuição de prioridade de um nó.
- ⇒ O aumento está associado à "subida" do nó, na árvore binária correspondente.
- ⇒ A diminuição está associada à "descida" do nó, na árvore binária correspondente.
- ⇒ A subida e a descida de nós na árvore serão realizadas sempre através de caminhos.



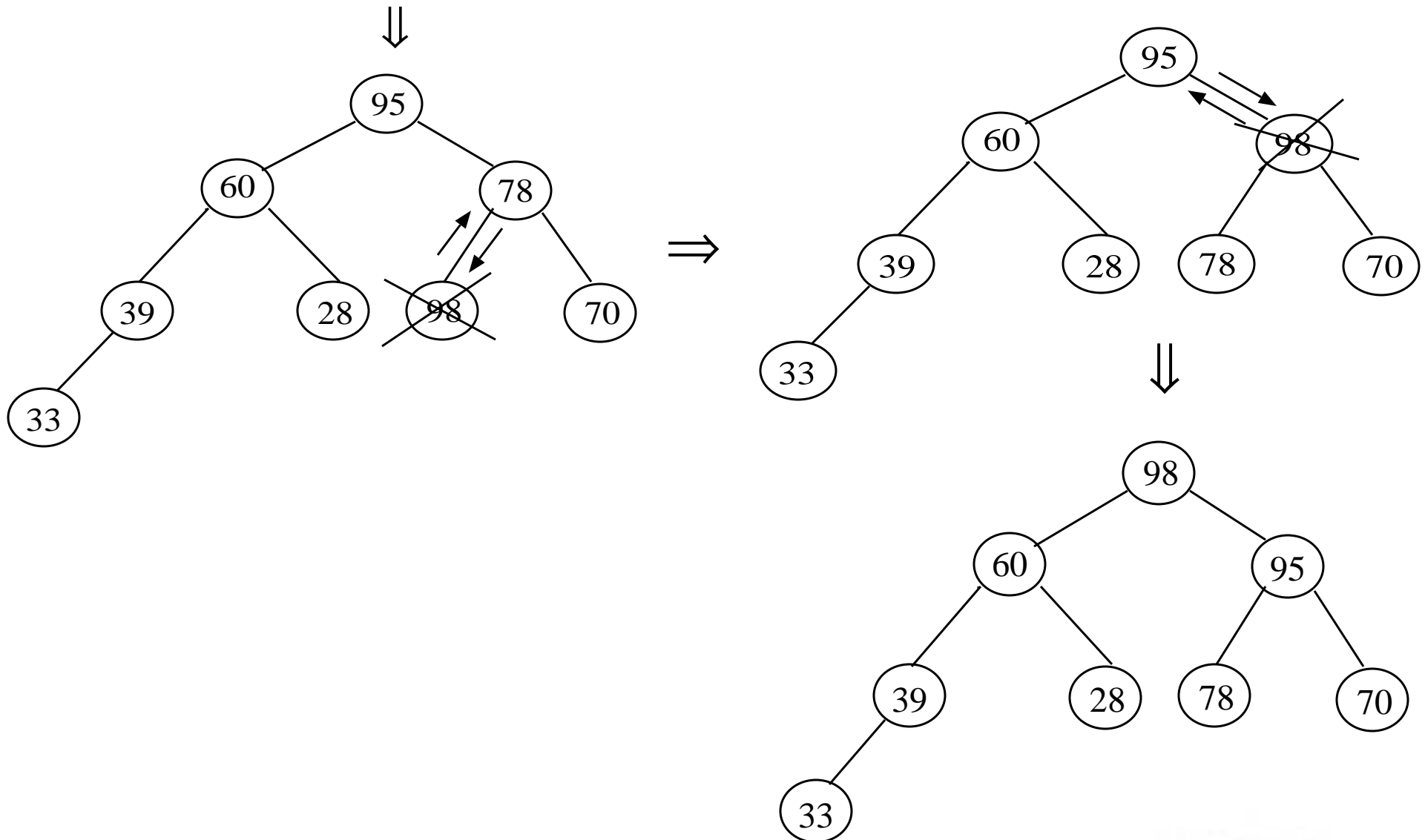
## Aumento da prioridade

⇒ Exemplo:



Aumentar prioridade do nó 6,  
de 66 para 98.

# Exemplo



## Método

- ➡ Seja  $v$  o nó cuja prioridade foi aumentada. Caso a prioridade do pai de  $v$ , se existir, seja menor do que a de  $v$ , trocar de posições  $v$  e o pai de  $v$ . Iterativamente, repetir esta operação, tornando  $v$  igual a seu pai, até que o nó considerado seja a raiz da árvore, ou que sua prioridade seja menor ou igual que a prioridade do seu pai.

## Algoritmo de subida

➡ Algoritmo: subir por um caminho na árvore

```
procedimento subir (i)
  j := ⌊i/2⌋
  se j ≥ 1 então
    se T[i].chave > T[j].chave então
      T[i] ⇔ T[j]
      subir (j)
```

Animar

Voltar

## Algoritmo de subida

- ➡ O heap está armazenado na tabela  $T$ .
- ➡ O parâmetro  $i$  indica a posição do elemento a ser revisto.
- ➡ O campo chave armazena a prioridade do nó.
- ➡ A notação  $T[i] \Leftrightarrow T[j]$  indica a troca de posições em  $T$ , entre os nós  $i$  e  $j$ .
- ➡ Complexidade: Da ordem da altura da árvore.  
Como a árvore é completa, complexidade  $O(\log n)$ .

## Exercício

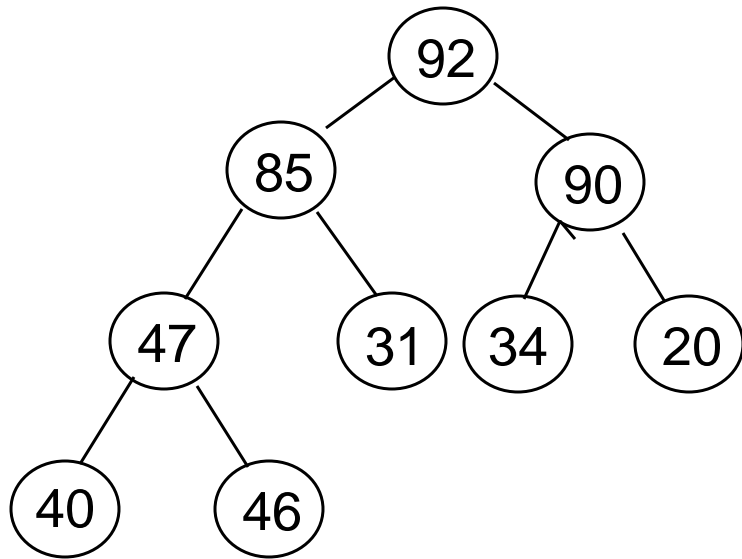
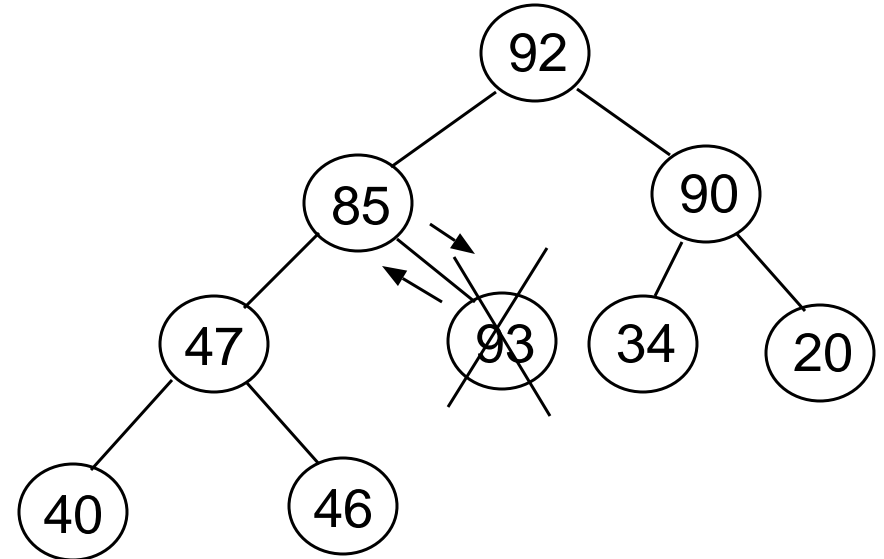
➡ Seja o heap

92 85 90 47 31 34 20 40 46

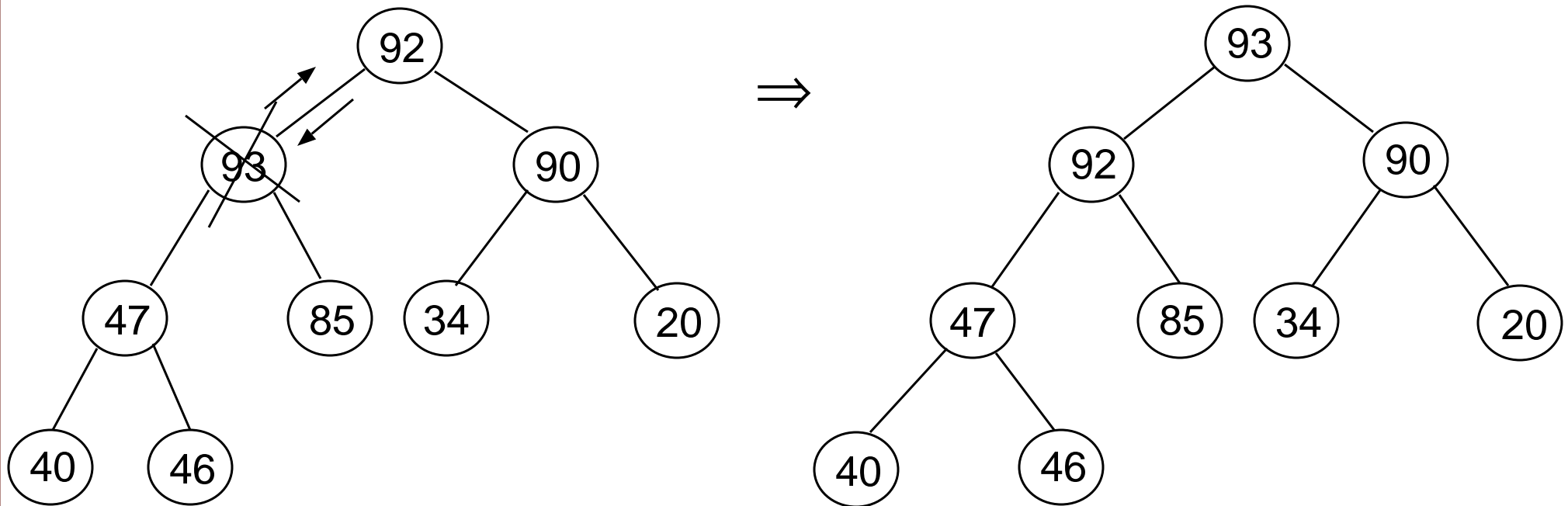
➡ Aplicando o algoritmo de subida, determinar o heap resultante da alteração de prioridade do 5o. nó, de 31 por 93.

Tempo: 3 minutos

## Solução

 $\Rightarrow$ 

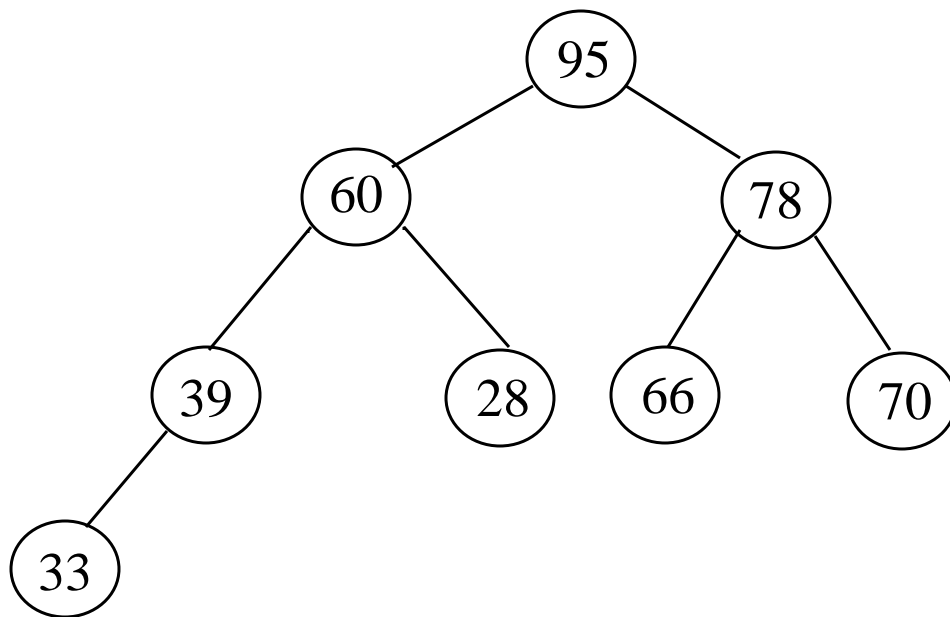
## Solução





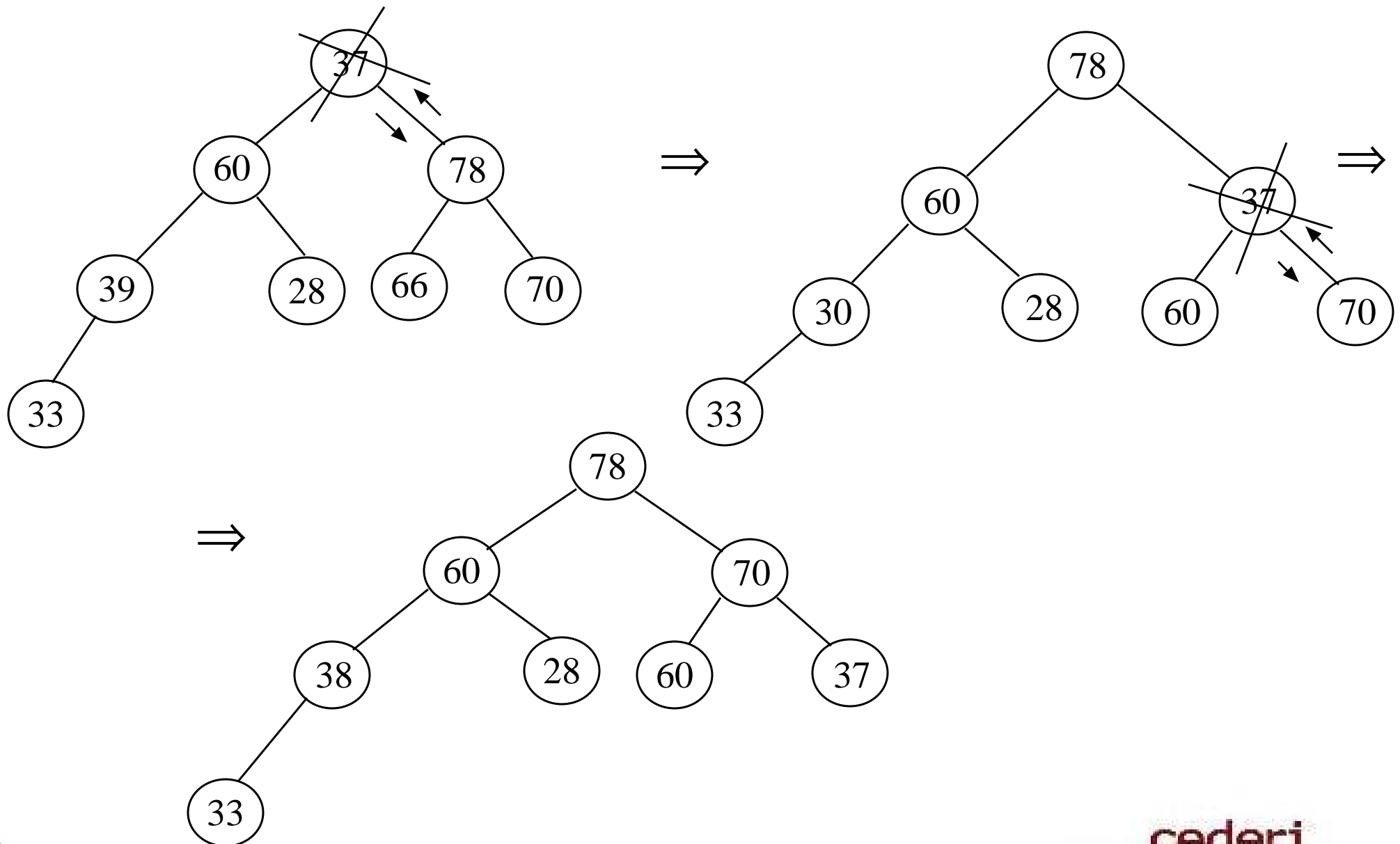
## Diminuição de prioridade

➡ Exemplo



Diminuir a prioridade do nó 1, de 95 por 37.

# Exemplo



## Método

- ➡ Seja  $v$  o nó cuja prioridade foi diminuída. Caso a prioridade de algum filho de  $v$ , se existir, seja maior do que a de  $v$ , trocar as posições de  $v$  e seu filho de maior prioridade. Iterativamente, repetir esta operação, tornando  $v$  igual a seu filho de maior prioridade, até que o nó considerado seja uma folha, ou que sua prioridade seja maior ou igual do que a de seus filhos.

## Algoritmo de descida

⇒ Algoritmo: Descer por um caminho na árvore

⇒ Procedimento: descer (i, n)

```
j := 2 x i
se j ≤ n então
    se j < n então
        se T [j+1].chave > T [j].chave então
            j := j+1
    se T [i].chave < T [j].chave então
        T [i] ⇔ T [j]
        descer (j, n)
```

## Algoritmo de descida

- ➡ O heap está armazenado na tabela T.
- ➡ O parâmetro i indica a posição do elemento a ser revisto.
- ➡ O campo chave armazena a prioridade do nó.
- ➡ A notação  $T[i] \Leftrightarrow T[j]$  indica a troca de posição em T, entre os nós i e j.

## Exercício

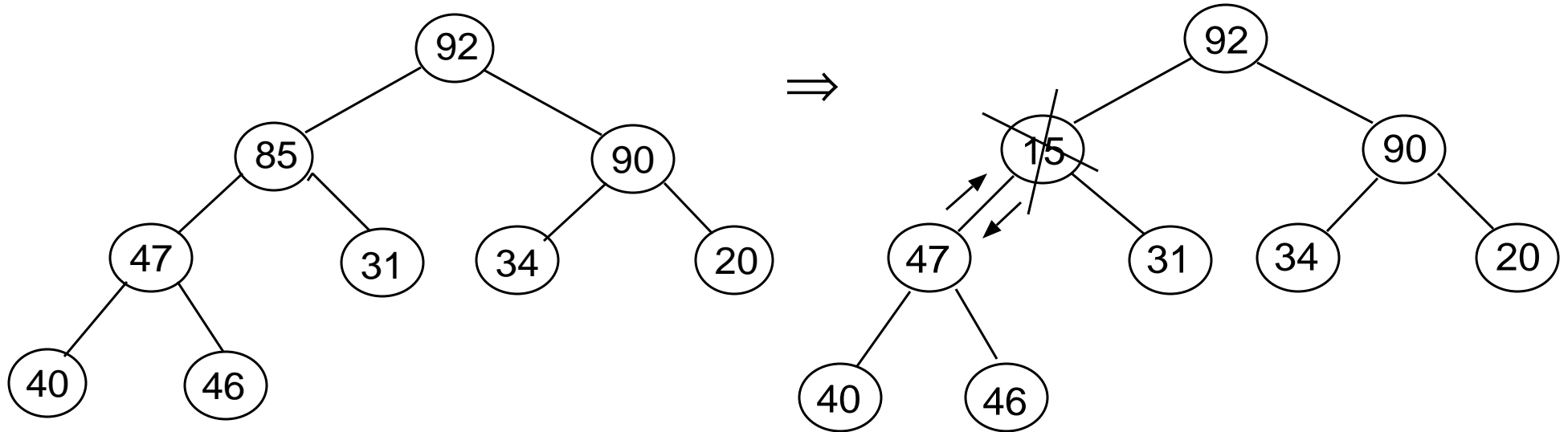
➡ Seja o heap

92   85   90   47   31   34   20   40   46

➡ Aplicando o algoritmo de descida, determine o heap resultante da alteração de prioridade do 2º nó, de 85 por 15.

Tempo: 3 minutos

## Solução



## Solução

