

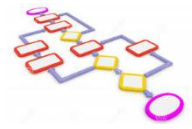


Unidade 7– Heaps e Filas de Prioridade



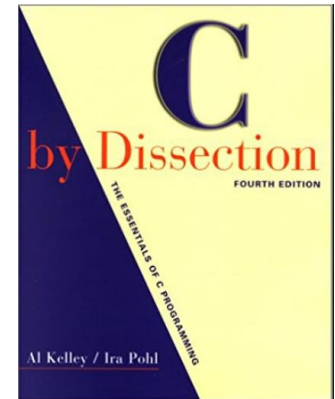
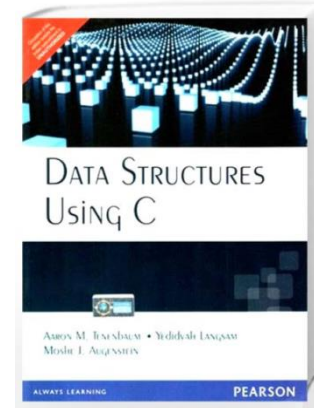
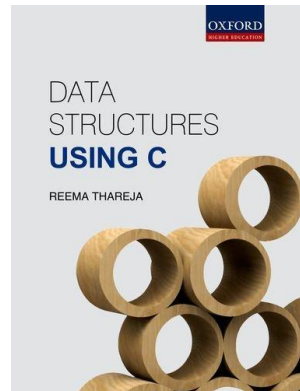
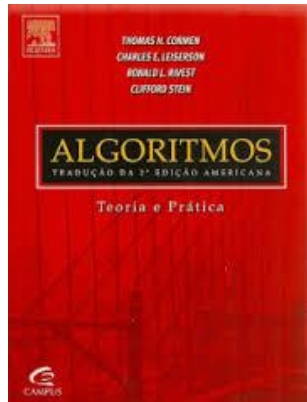
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUVSP
aparecidovfreitas@gmail.com

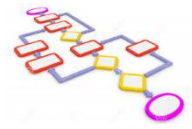




Bibliografia

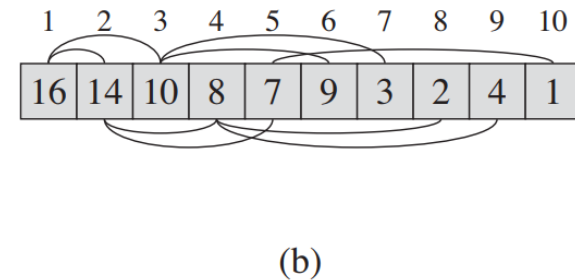
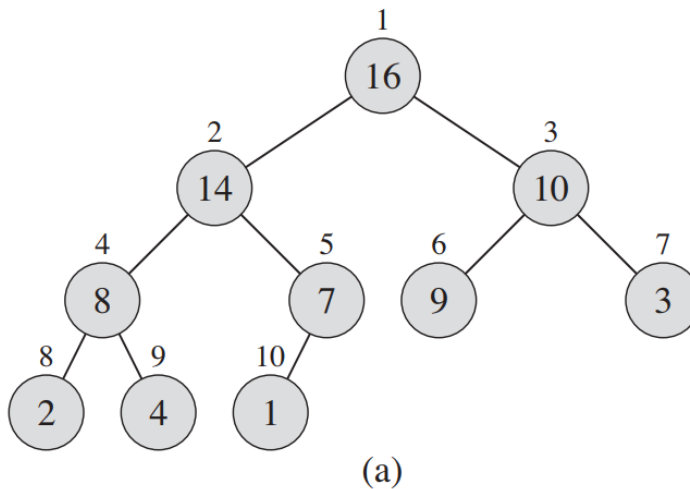
- Algoritmos – Teoria e Prática – **Cormen** – Segunda Edição – Editora Campus, 2002
- Data Structures using C – Oxford University Press – 2014
- Data Structures Using C – A. Tenenbaum, M. Augensem, Y. Langsam, Pearson 1995
- C By Dissection – Kelley, Pohh – Third Edition – Addison Wesley





Heaps

- A estrutura de dados **heap** (binário) pode ser vista como uma **árvore binária praticamente completa**.
- Cada **nó** da árvore corresponde a um elemento do array que armazena o valor do nó;
- A árvore está completamente preenchida em todos os níveis, exceto talvez no nível mais baixo, que é preenchido à esquerda até certo ponto;
- Heaps são usualmente implementados por meio de arrays.

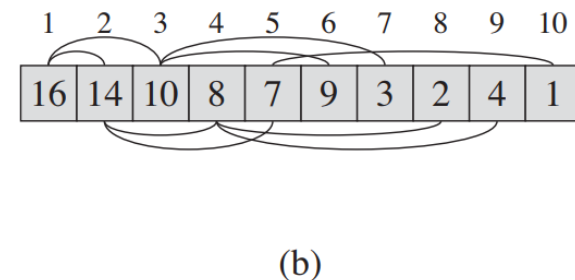
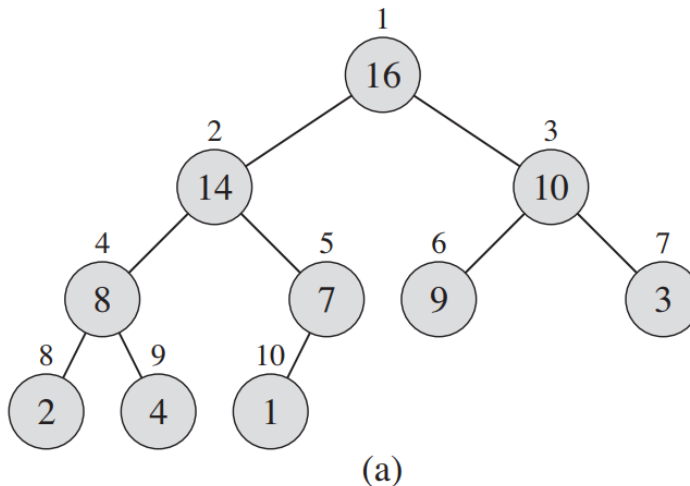


Heaps – Propriedades

É uma estrutura de dados que pode ser visualizada como uma **árvore binária quase completa**.

Cada nó da árvore é ocupado por um elemento e temos as seguintes propriedades:

- A árvore é completa até o penúltimo nível
- No último nível as folhas estão o mais à esquerda possível
- O conteúdo de um nó é **maior ou igual** ao conteúdo dos nós na subárvore enraizada nele (**max-heap**)



- **Observação:** Se o conteúdo de um nó é menor ou igual ao conteúdo dos nós da subárvore enraizada por ele, tem-se um (min-heap).



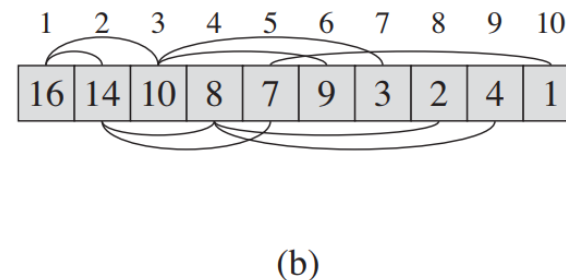
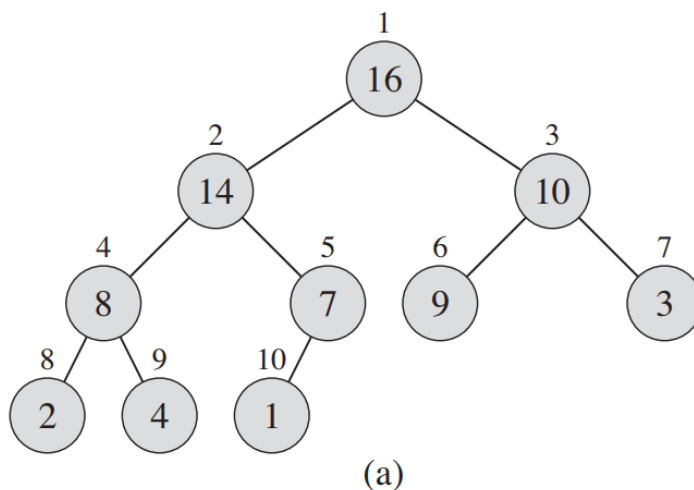
Heaps – Observação

É uma estrutura de dados que pode ser visualizada como uma **árvore binária quase completa**.

Cada nó da árvore é ocupado por um elemento e temos as seguintes propriedades:

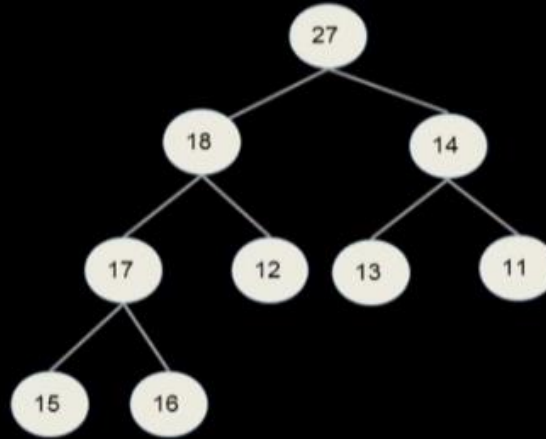
- A árvore é completa até o penúltimo nível
- No último nível as folhas estão o mais à esquerda possível
- O conteúdo de um nó é **maior ou igual** ao conteúdo dos nós na subárvore enraizada nele (**max-heap**)

- **Observação:** Essas propriedades garantem que um heap pode ser implementado em um array $A[1..m]$.





Heap – Exemplo

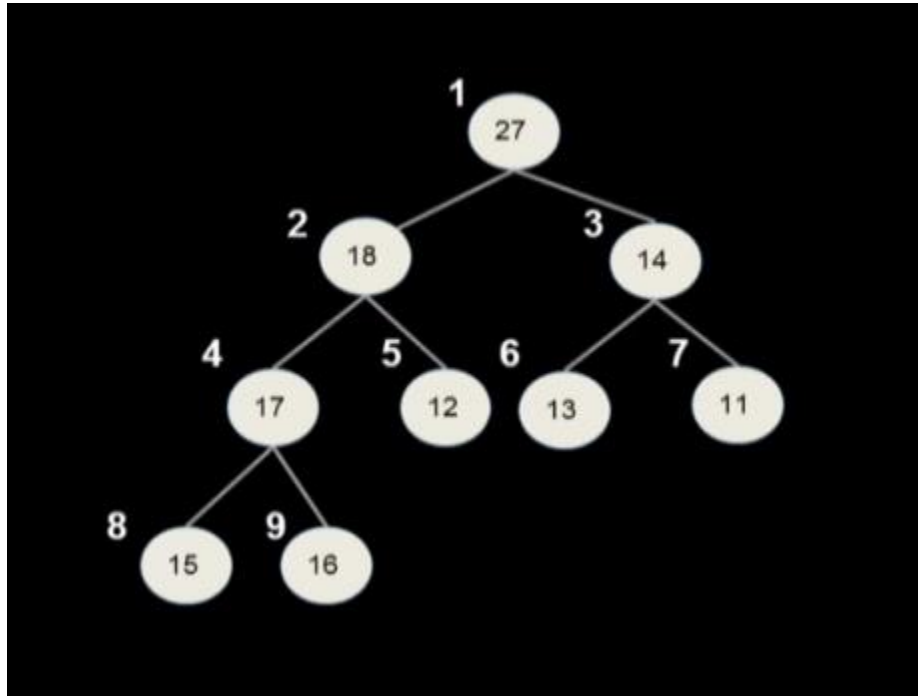


- A árvore é completa até o penúltimo nível ✓
- No último nível as folhas estão o mais à esquerda possível ✓
- O conteúdo de um nó é **maior ou igual** ao conteúdo dos nós na subárvore enraizada nele (**max-heap**) ✓



Heap – Implementação

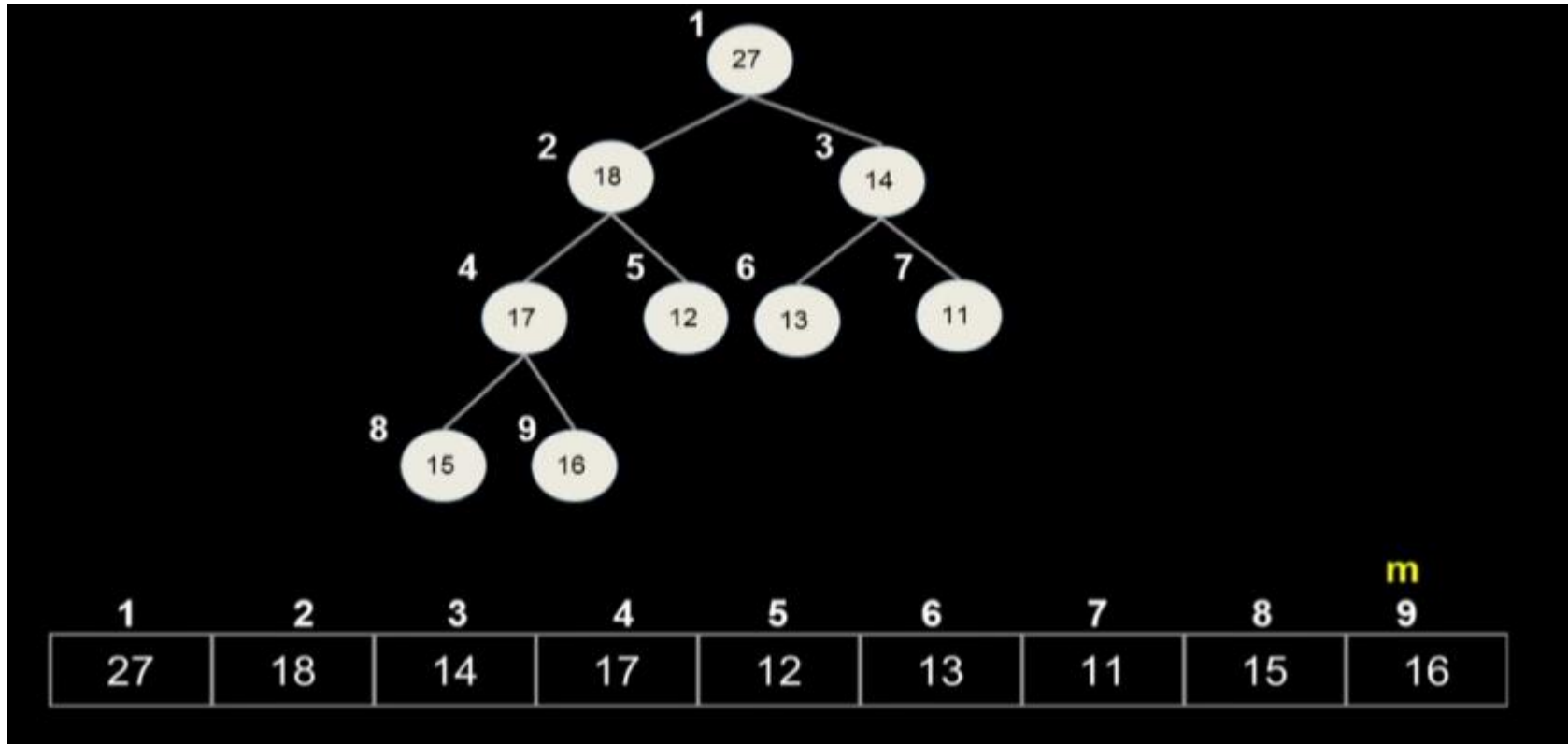
- A implementação do heap, pelas suas propriedades, não exige que se crie uma árvore;
- Pode-se assim, implementar um **heap** por meio de um **array**;





Heap – Implementação

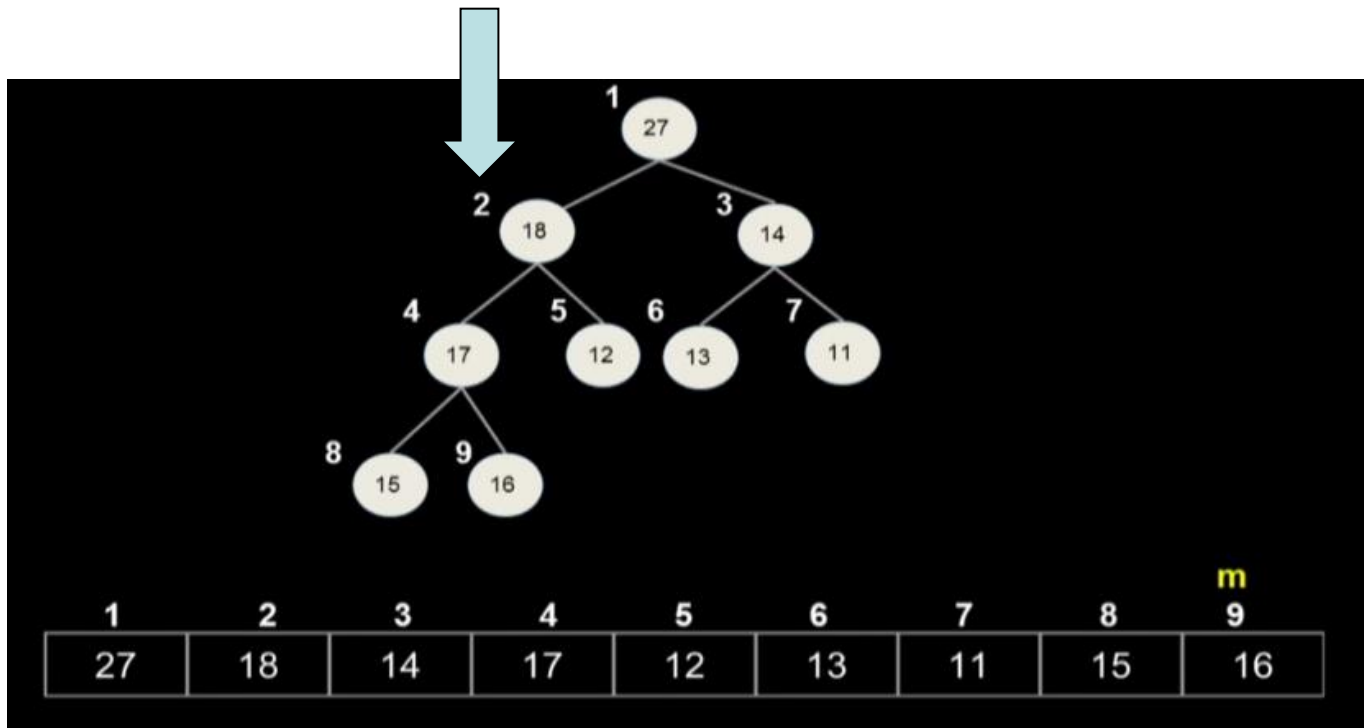
- Assim, a implementação de um heap é feita por meio de um array;





Heap – Implementação

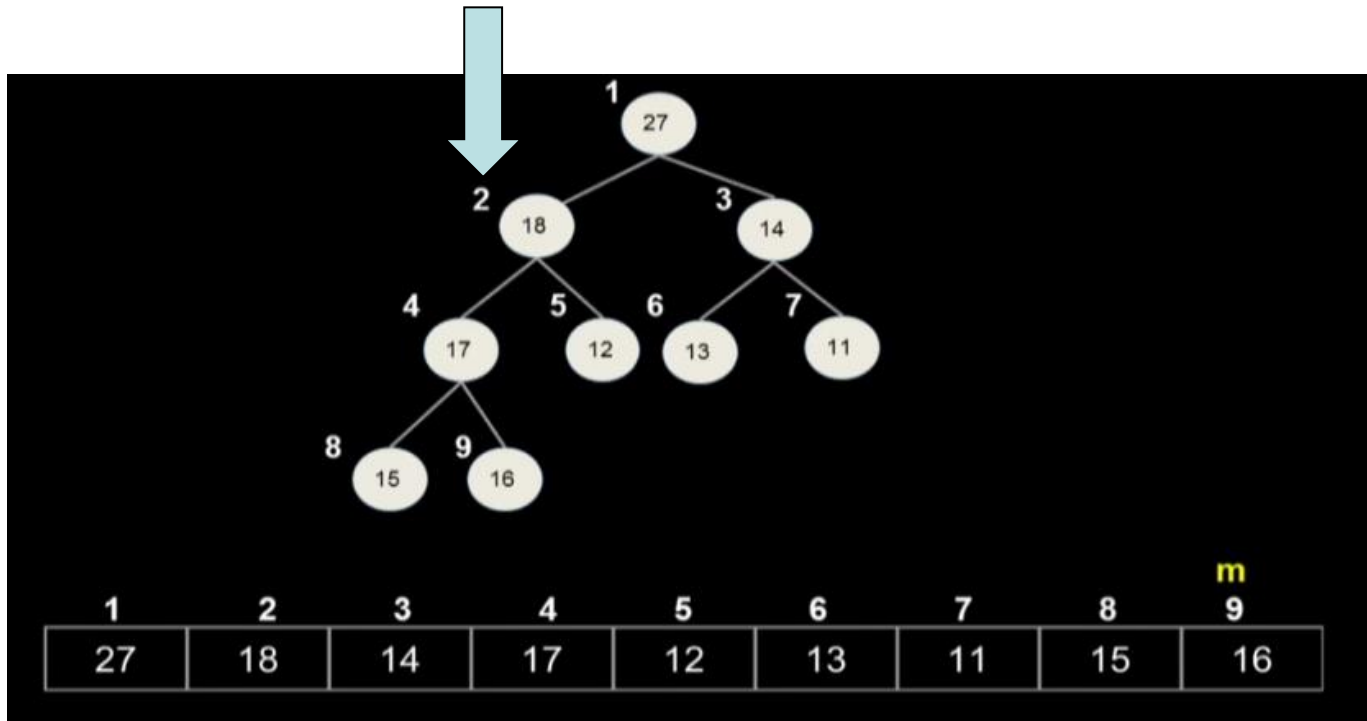
- Dado um elemento do heap, como determinar os valores correspondentes a seus filhos?
- Por exemplo, quem são os filhos do nó de **índice 2**?





Heap – Implementação

- Por exemplo, quem são os filhos do nó de **índice 2**?

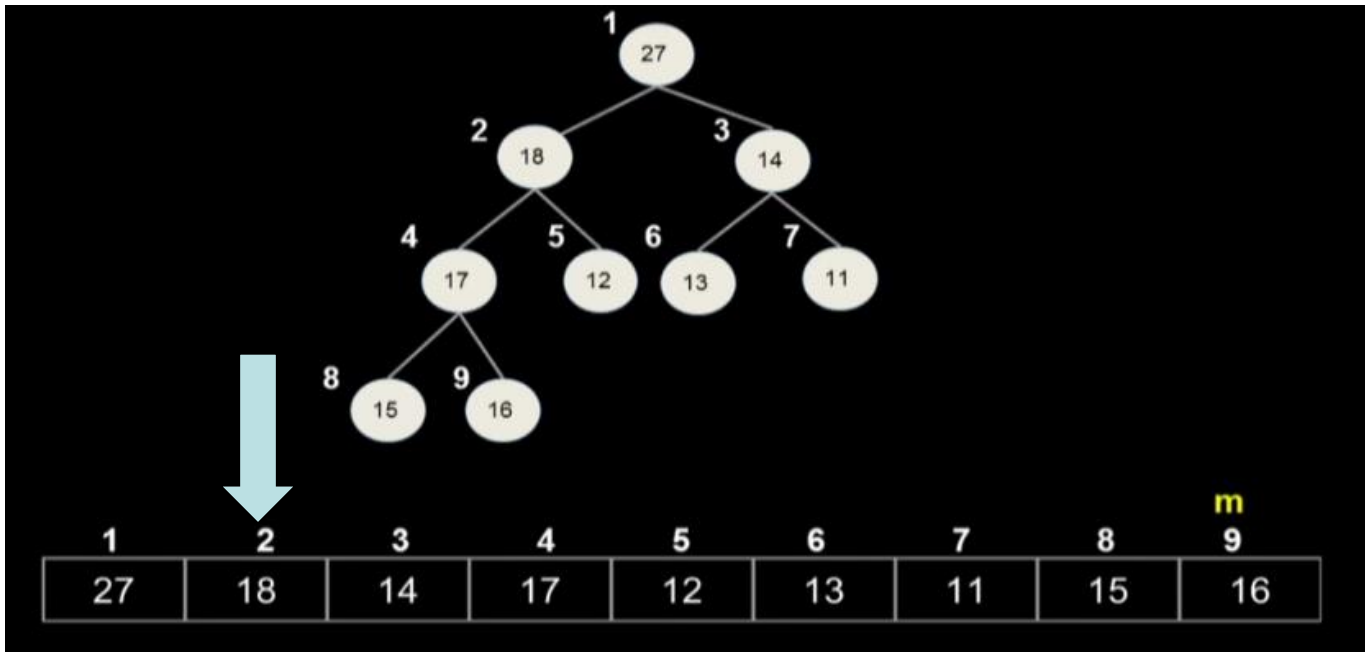


- Observando a árvore correspondente ao heap, é fácil verificar-se que os filhos de 18 são 17 e 12.
- Mas, como determiná-los no array de implementação?



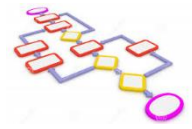
Heap – Implementação

- Por exemplo, quem são os filhos do nó de índice 2?



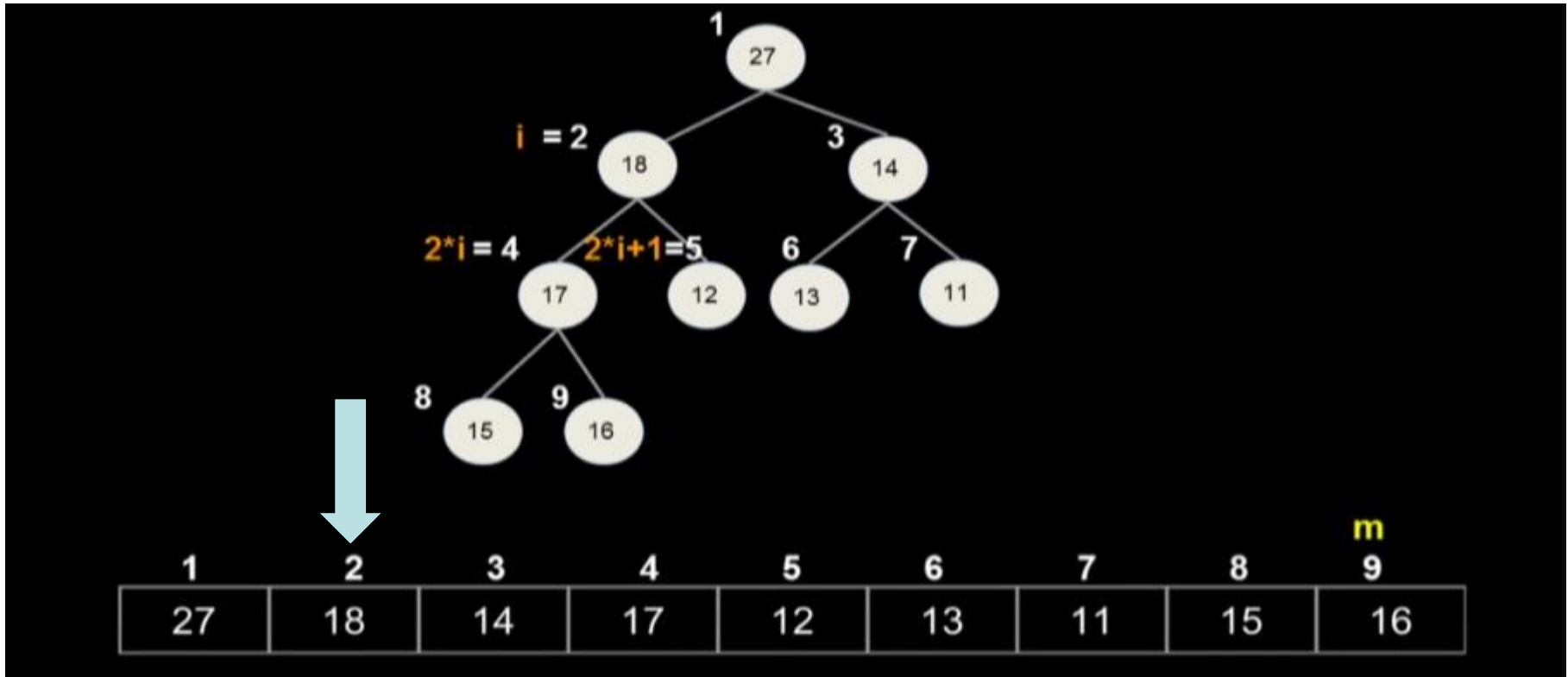
- Mas, como determiná-los no array de implementação?





Heap – Implementação

- Por exemplo, quem são os filhos do nó de índice 2?

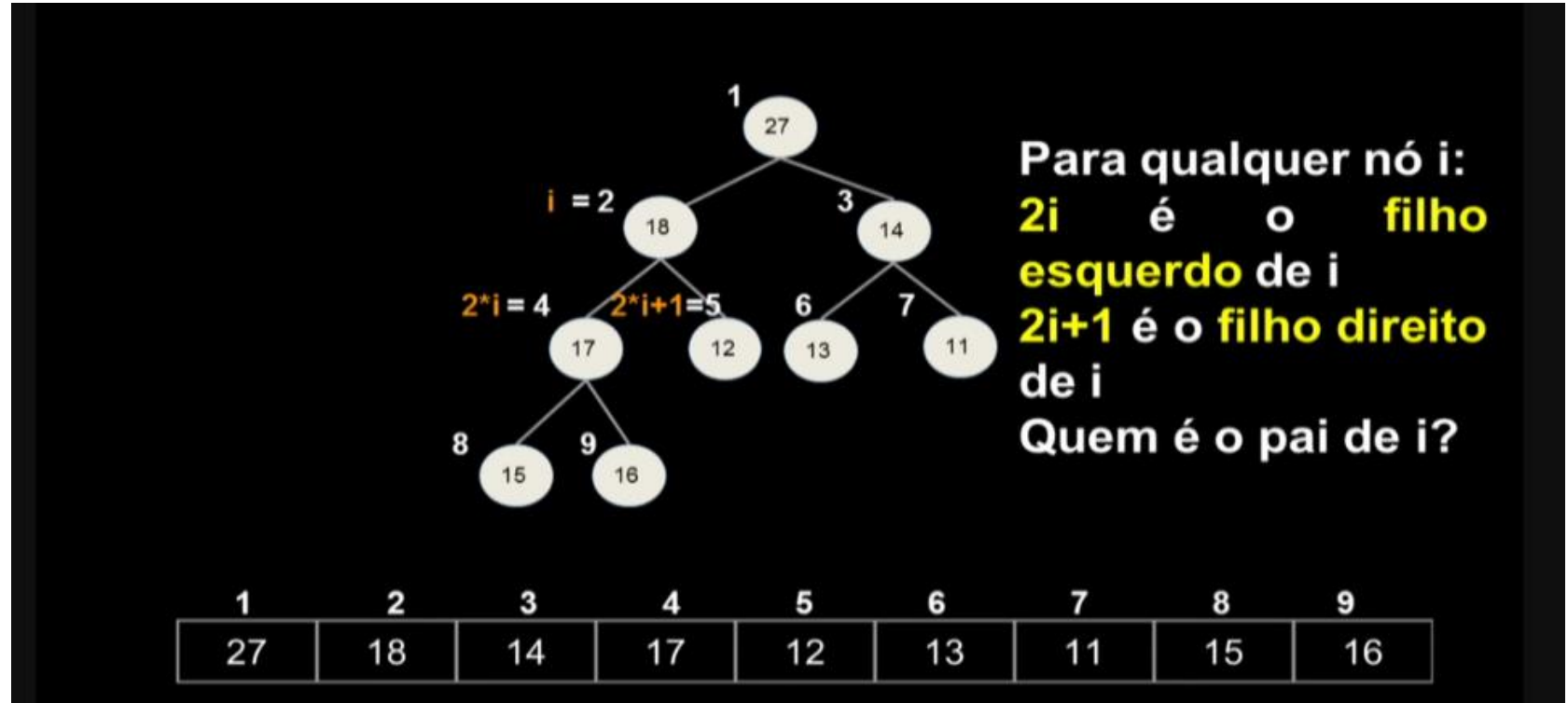


- No array de implementação, o filho a esquerda do nó com endereço i , está na posição $2*i$;
- No array de implementação, o filho a direita do nó com endereço i , está na posição $2*i + 1$.





Heap – Implementação

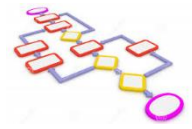




Heap – Implementação

- Dado um nó em um endereço i , qual o endereço de seu pai?





Heap – Implementação

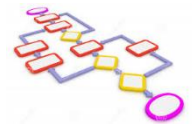
- Dado um nó em um endereço i , qual o endereço de seu pai?



1	2	3	4	5	6	7	8	9
27	18	14	17	12	13	11	15	16

- No array de implementação, o pai de um nó em um endereço i está na posição $\lfloor i/2 \rfloor$.





Heap – Propriedades

- Dado um nó em um endereço i , qual o endereço de seu pai?

PARENT(i)

return $\lfloor i/2 \rfloor$

LEFT(i)

return $2i$

RIGHT(i)

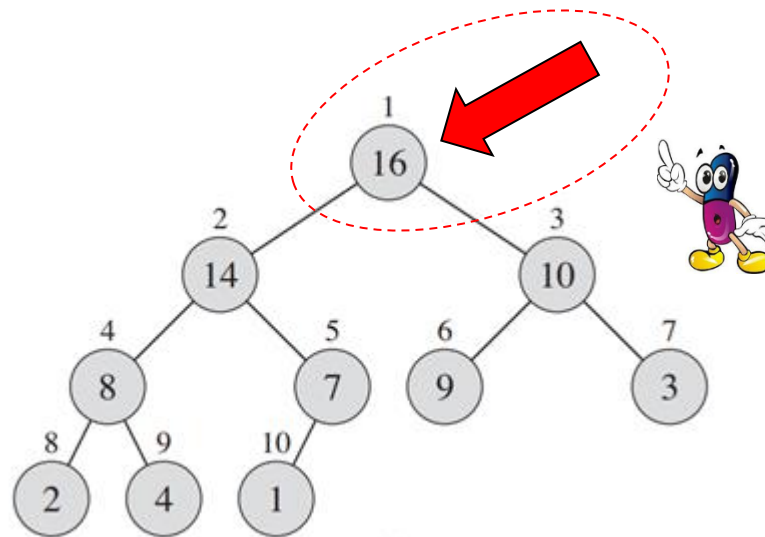
return $2i + 1$





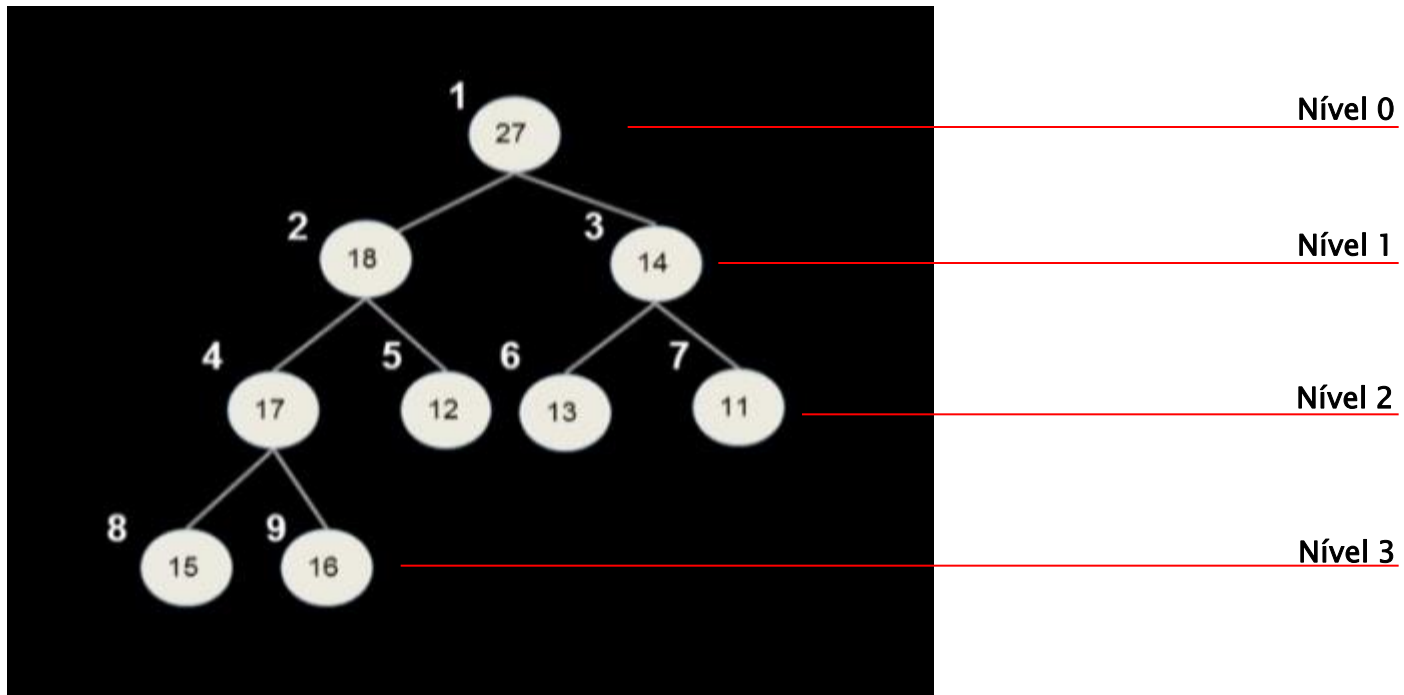
Heap – Observação

- Em um **max-heap**, o maior elemento da árvore, por definição, está na raiz;



Níveis de um Heap

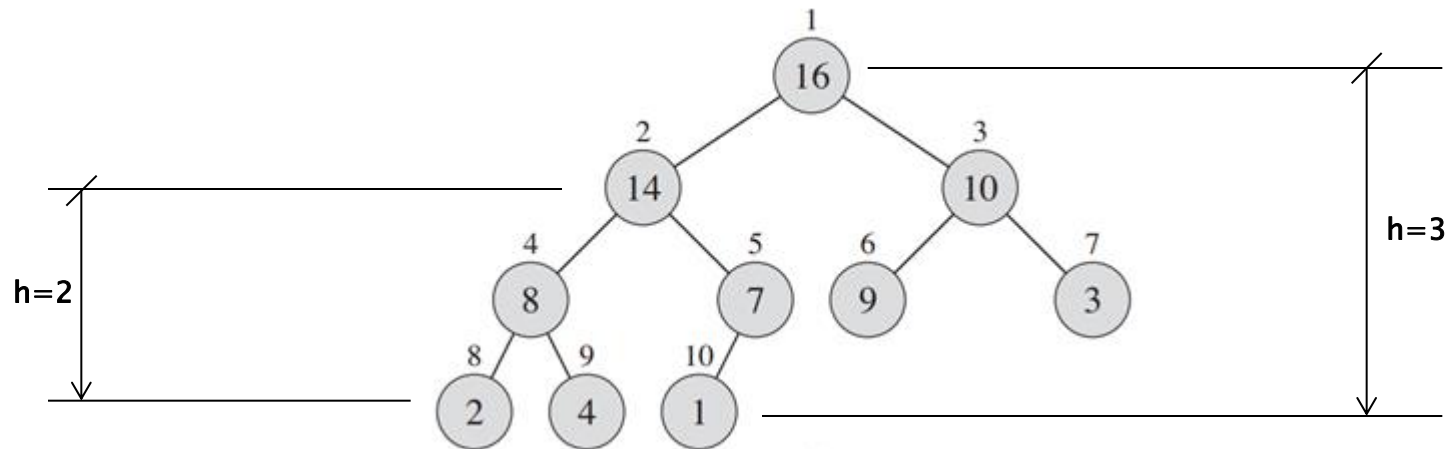
- Cada nível p , tem exatamente 2^p nós, exceto talvez no último nível;
- Por exemplo, no heap abaixo, o nível 2 tem exatamente 4 nós.

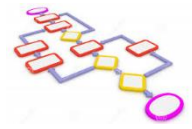




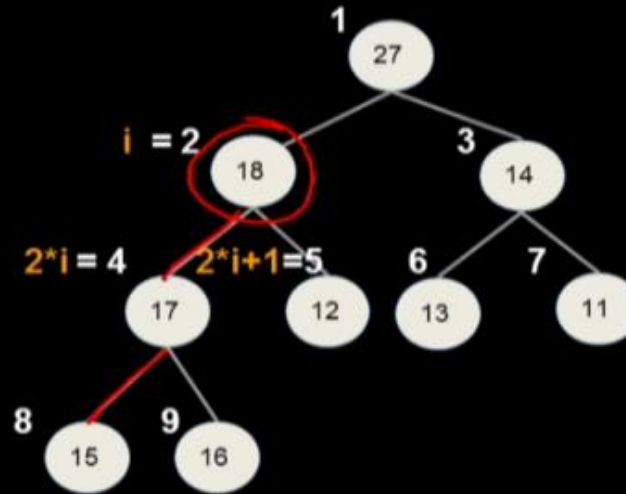
Altura de nós do Heap

A altura de um nó i é o maior comprimento de um caminho de i até uma folha, isto é, o número de arestas no caminho mais longo desde i até uma folha.





Altura de nós do Heap



As folhas têm
altura 0
Qual a altura do
nó número 2?
Resposta: 2

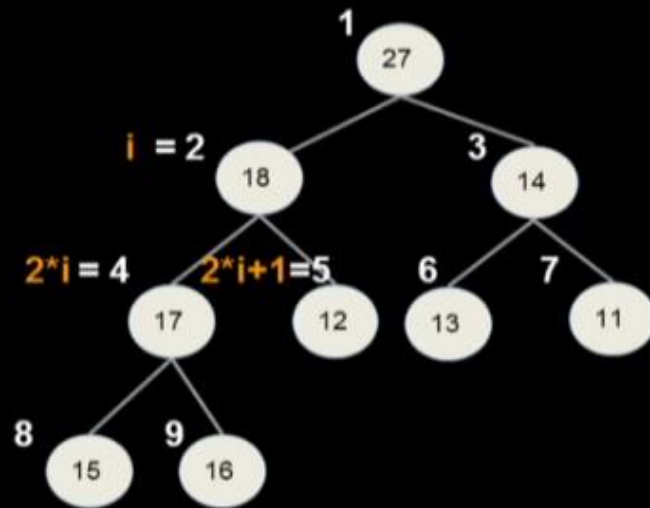
1	2	3
27	18	14

A altura de um nó i é o maior comprimento de um caminho de i a uma folha, isto é, o número de arestas no caminho mais longo desde i até uma folha.





Altura de nós do Heap



Pode ser demonstrado que a altura de um nó i é:

$$h = \lfloor \lg(m/i) \rfloor$$

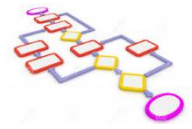
A altura da raiz é:

$$h = \lfloor \lg(m) \rfloor$$

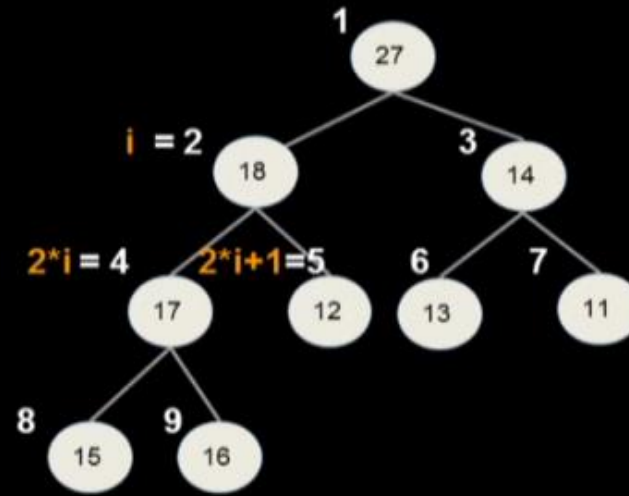
1	2	3
27	18	14

A altura de um nó i é o maior comprimento de um caminho de i a uma folha, isto é, o número de arestas no caminho mais longo desde i até uma folha.





Altura de uma árvore



Pode ser demonstrado que a altura de um nó i é:
 $h = \lfloor \lg(m/i) \rfloor$
A altura da árvore é:
 $h = \lfloor \lg(m) \rfloor$

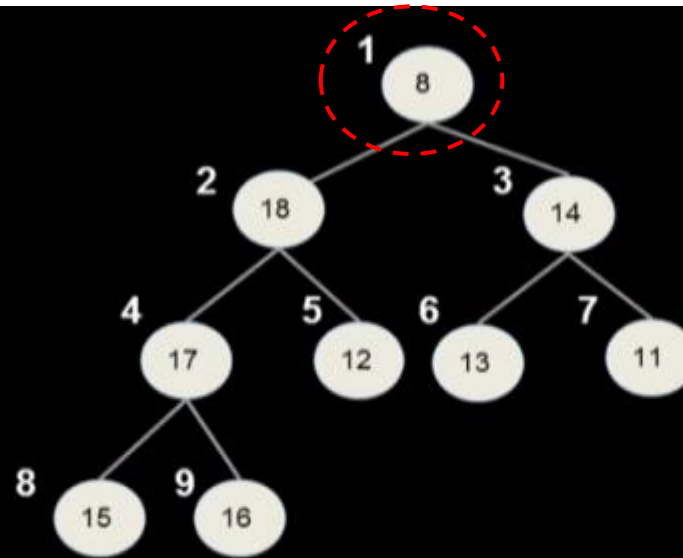
1	2	3
27	18	14

A altura de um nó i é o maior comprimento de um caminho de i a uma folha, isto é, o número de arestas no caminho mais longo desde i até uma folha.





Manutenção do Heap



É quase um heap.

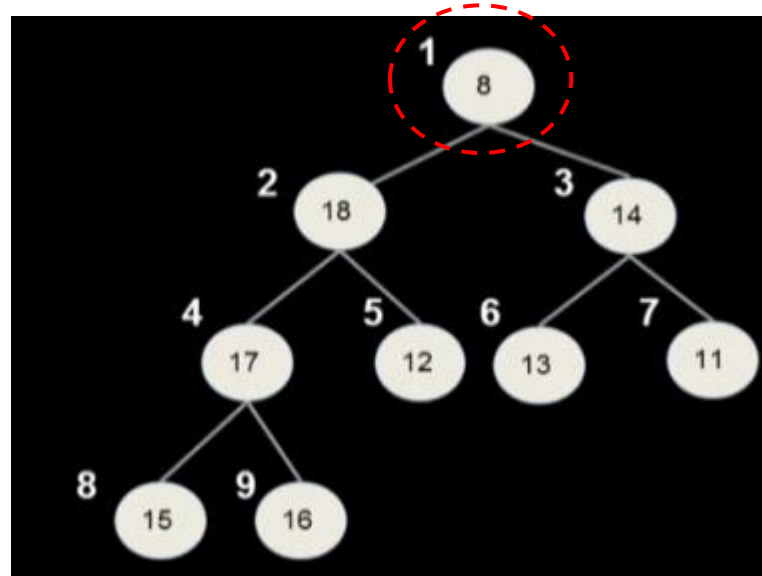
Apenas o elemento da raiz não satisfaz a propriedade de heap.

1	2	3	4	5	6	7	8	9
8	18	14	17	12	13	11	15	16





Manutenção do Heap

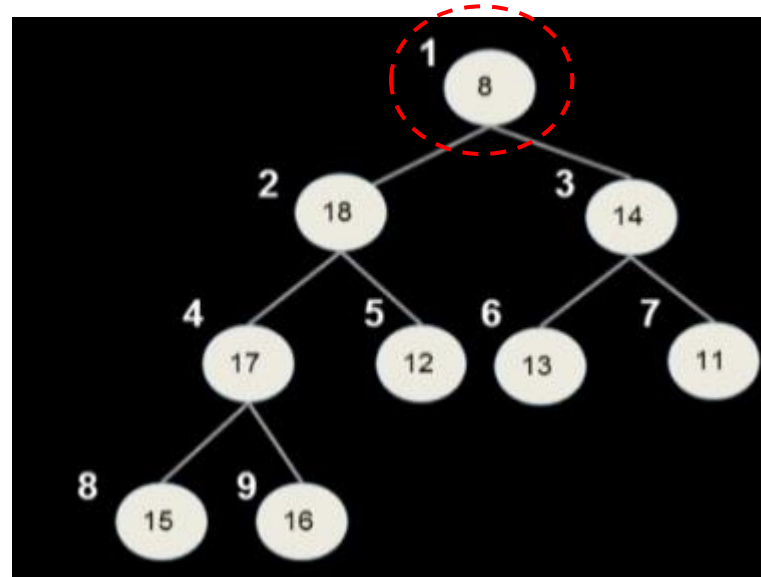


O que se deve fazer para que a árvore se torne um heap?





Manutenção do Heap



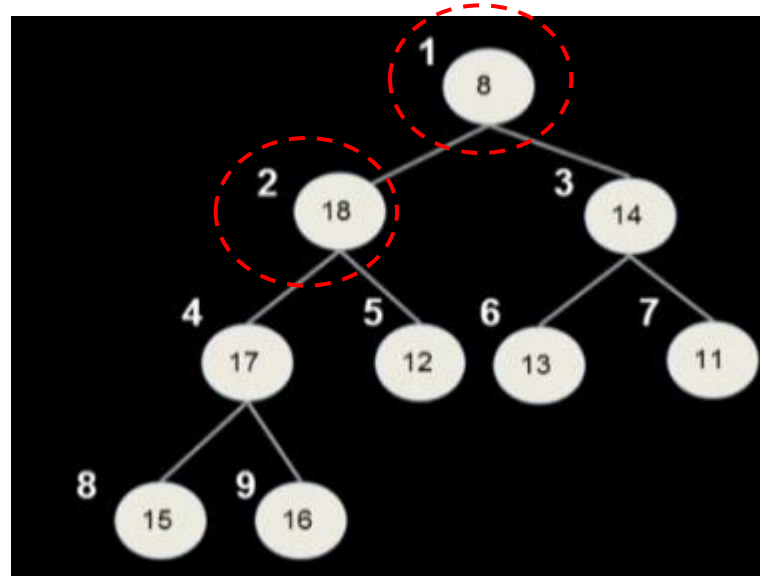
- Uma ideia seria descer o nó com o valor **8** para baixo, até deixá-lo numa posição conveniente que atenda a propriedade do heap;

1	2	3	4	5	6	7	8	9
8	18	14	17	12	13	11	15	16





Manutenção do Heap



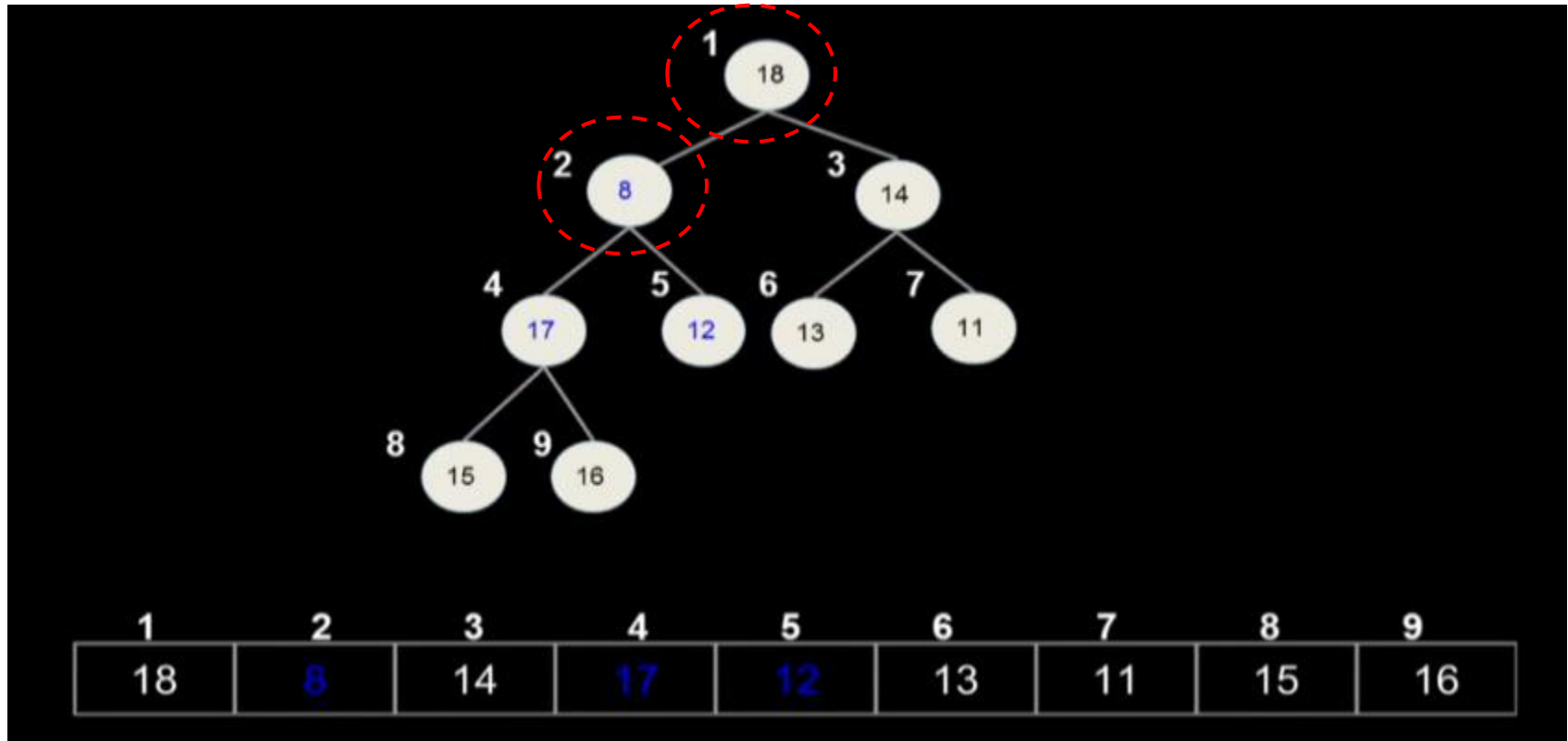
- Primeiramente, vamos comparar o 8 com os seus filhos;
- Vamos trabalhar com o maior dos filhos, no caso o nó com o valor 18;
- Proceda-se à troca do 8 com o 18;

1	2	3	4	5	6	7	8	9
8	18	14	17	12	13	11	15	16



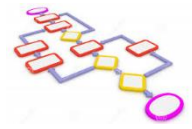


Manutenção do Heap

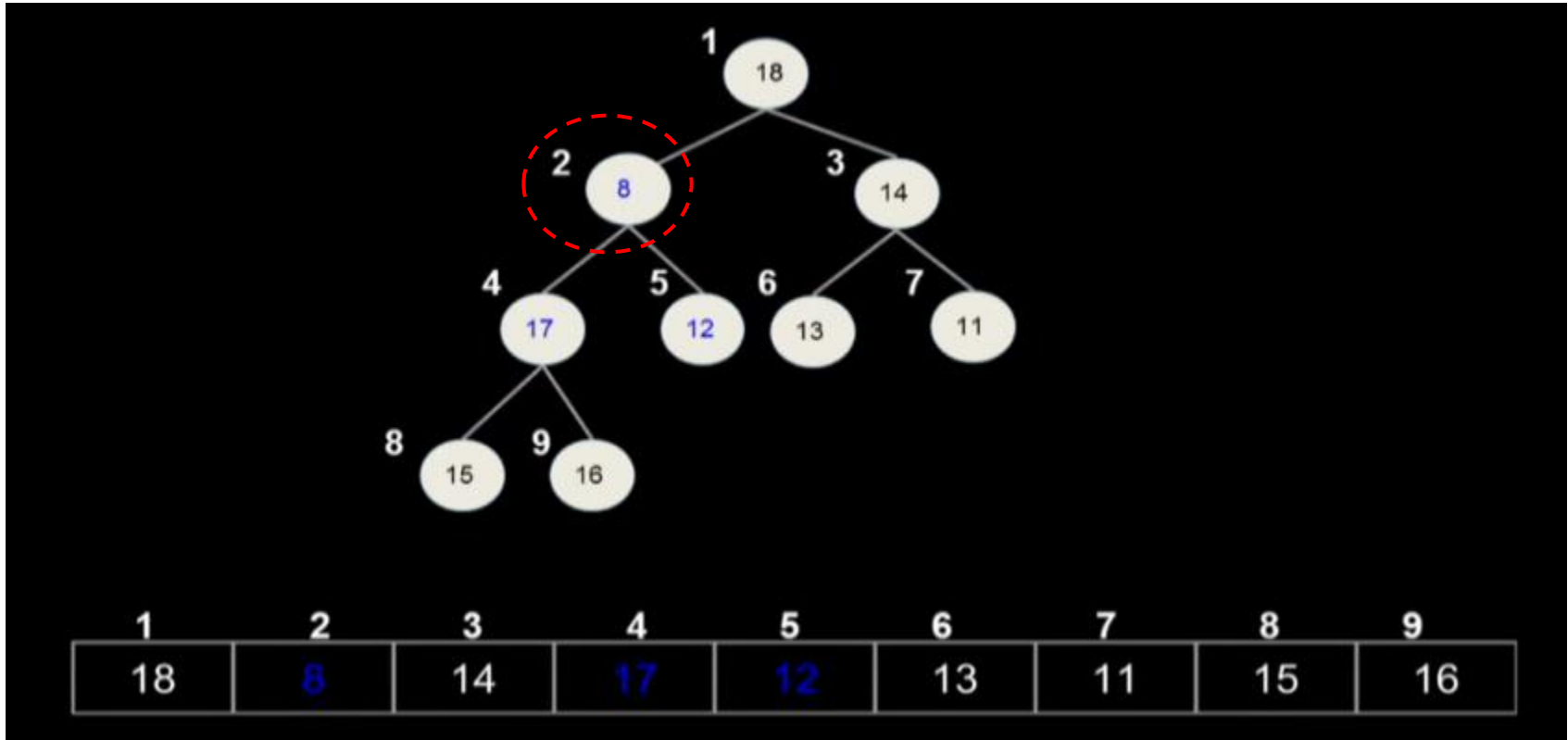


■ XXXX





Manutenção do Heap

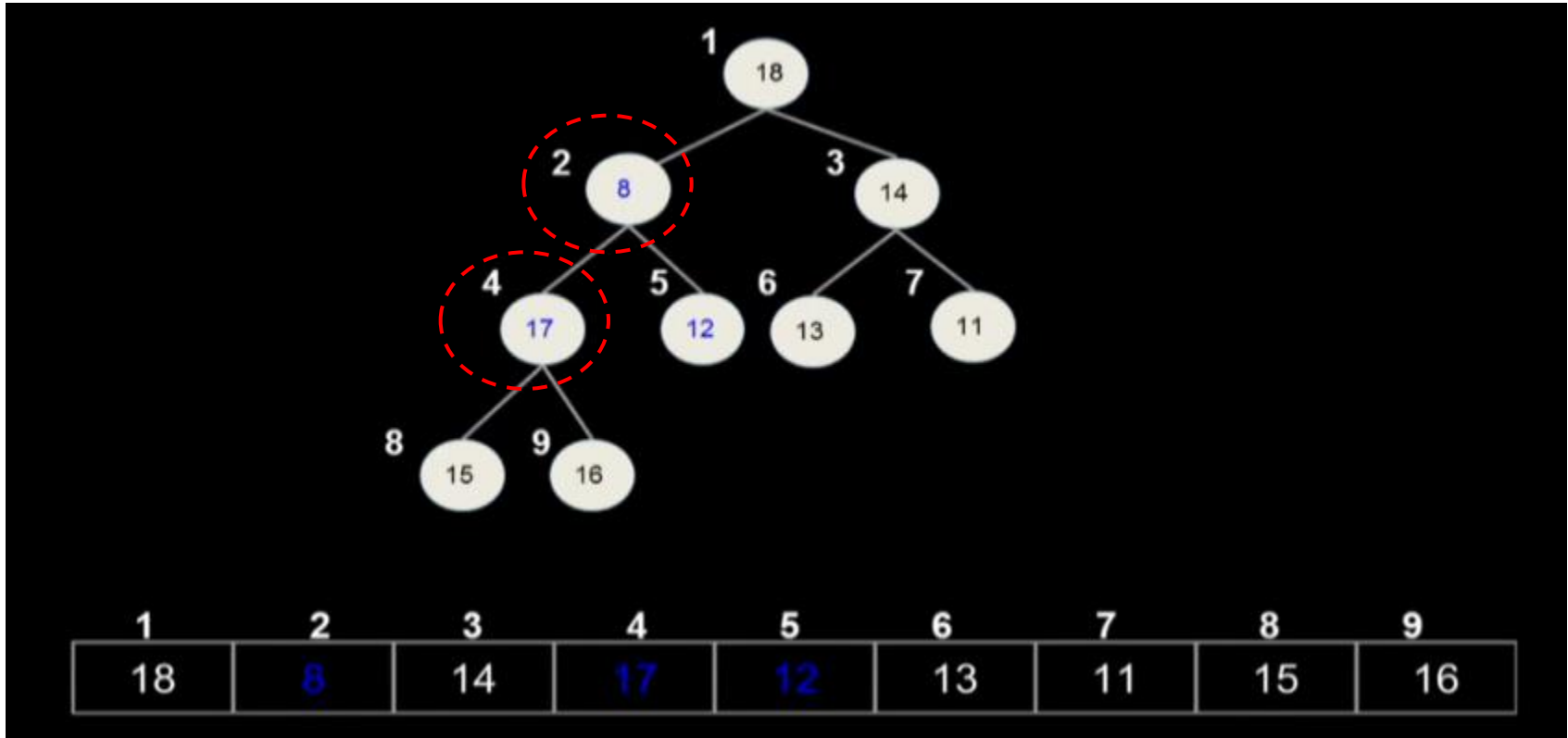


- Fazemos agora o mesmo com os filhos do nó 8;
- Compara-se o nó 8 com os seus filhos 17 e 12





Manutenção do Heap

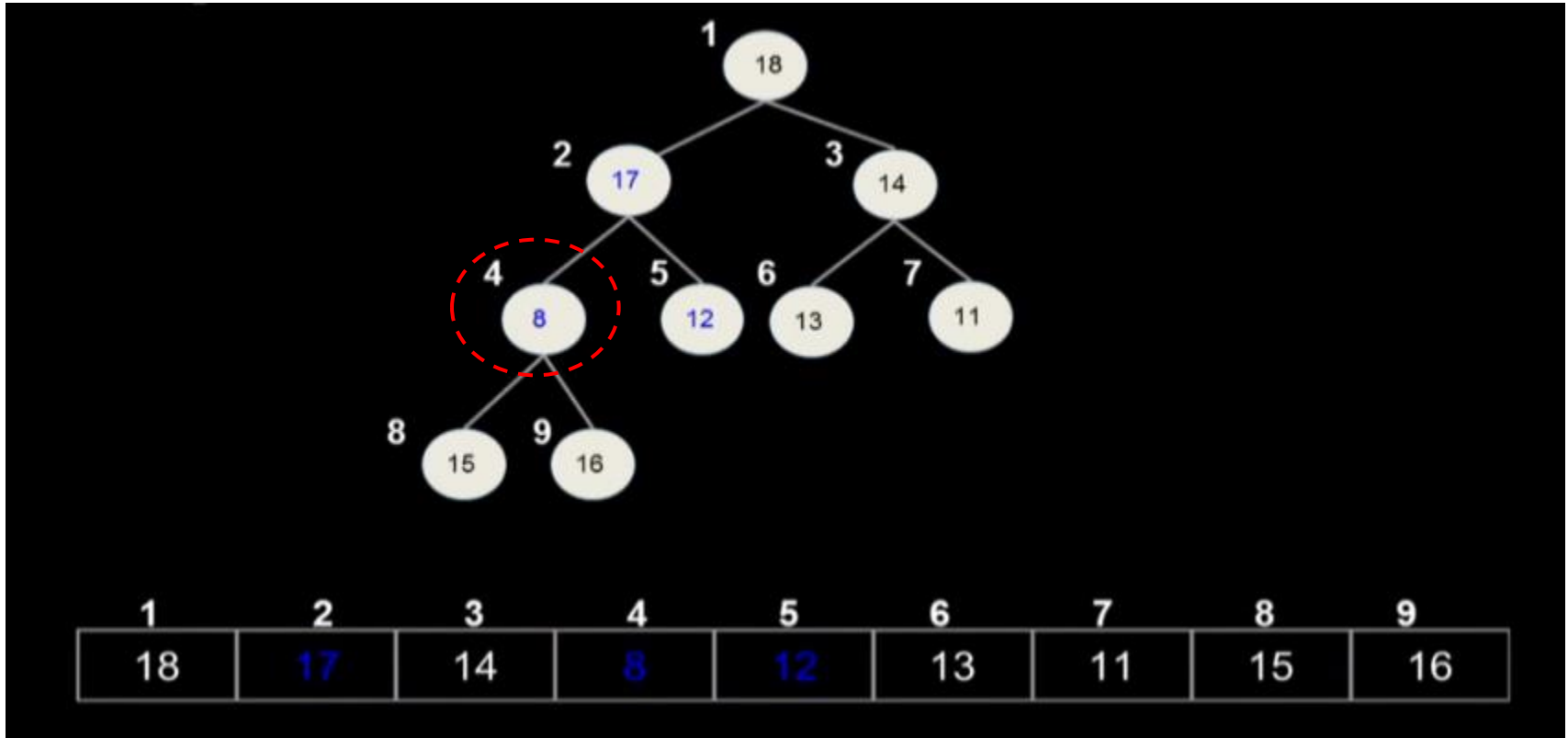


- Proceda-se à troca do nó 8 com o maior dos filhos que é o 17





Manutenção do Heap

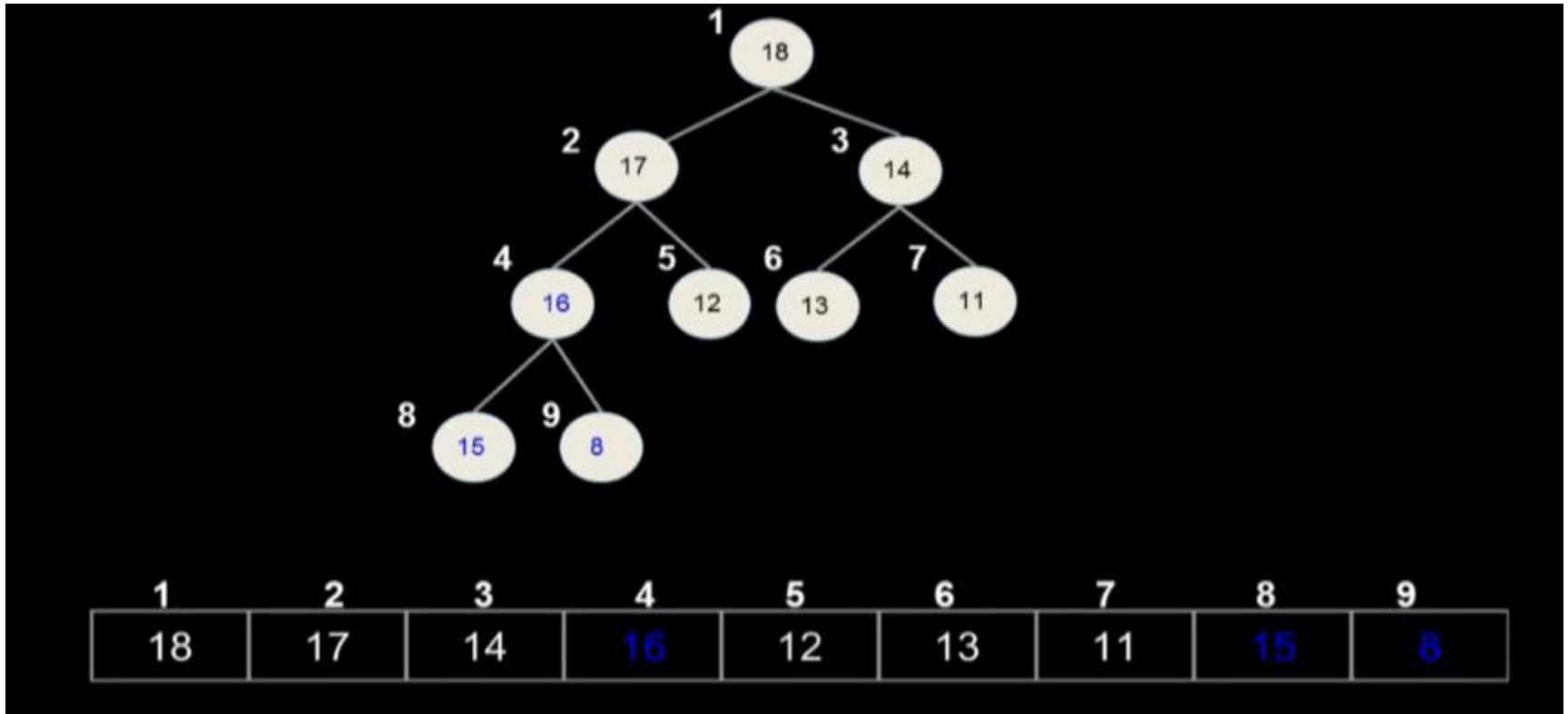


- O processo continua até a árvore se tornar um heap (heapfy);





Manutenção do Heap



■ A árvore agora é um **heap**;





Algoritmo Max-Heapfy

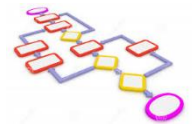
Manutenção da propriedade de heap

MAX-HEAPIFY (A, m, i)

```
1 e ← 2*i
2 d ← 2*i + 1
3 se e ≤ m e A[e] > A[i]
4   então maior ← e
5   senão maior ← i
6 se d ≤ m e A[d] > A[maior]
7   então maior ← d
8 se maior ≠ i
9   então A[i] ↔ A[maior]
10   MAX-HEAPIFY (A, m, maior)
```

Recebe o vetor A [1 ...m] e o índice i, tal que as árvores com raízes nos filhos esquerdo e direito do nó i são max-heaps.





Algoritmo Max-Heapfy

Manutenção da propriedade de heap

MAX-HEAPIFY (A, m, i)

```
1 e ← 2*i
2 d ← 2*i + 1
3 se e ≤ m e A[e] > A[i]
4   então maior ← e
5   senão maior ← i
6 se d ≤ m e A[d] > A[maior]
7   então maior ← d
8 se maior ≠ i
9   então A[i] ↔ A[maior]
10   MAX-HEAPIFY (A, m, maior)
```

$$T(h) \leq T(h-1) + \Theta(1)$$

$$T(h) = O(h)$$

Como: a altura de um nó i é:

$$h = \lfloor \lg(m/i) \rfloor$$

$$\begin{aligned} T(h) &= O(h) = O(\lfloor \lg(m/i) \rfloor) \\ &= O(\lg(m/i)) = O(\lg(m)) \end{aligned}$$





Construção de um Max Heap

- Dado um array de valores, como construir um **max heap** a partir deste array?

1	2	3	4	5	6	7	8	9
40	20	15	35	80	71	16	21	70

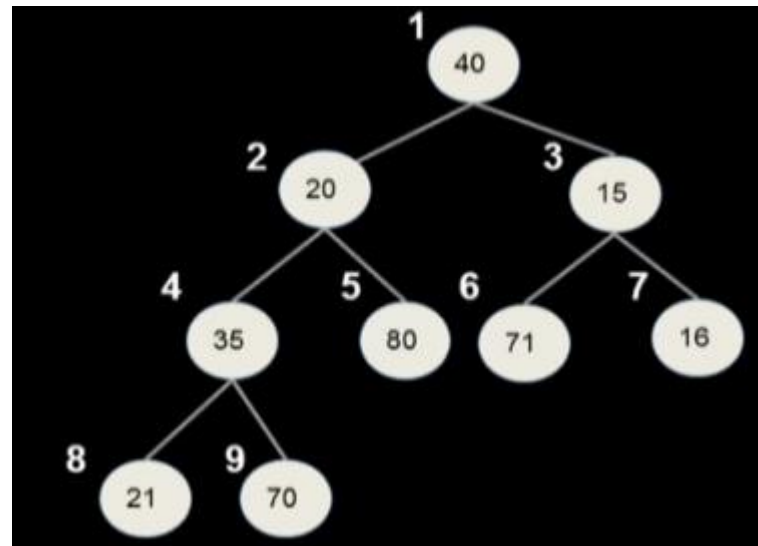




Construção de um Max Heap

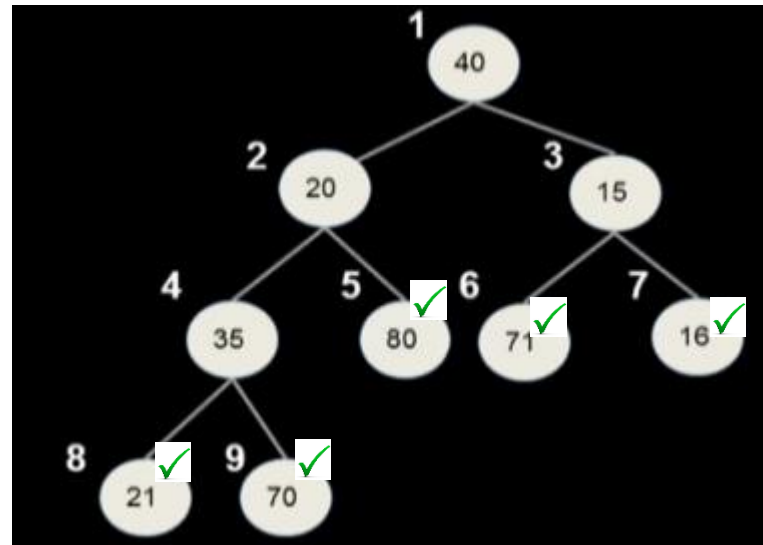
1	2	3	4	5	6	7	8	9
40	20	15	35	80	71	16	21	70

■ Árvore Inicial



Construção de um Max Heap

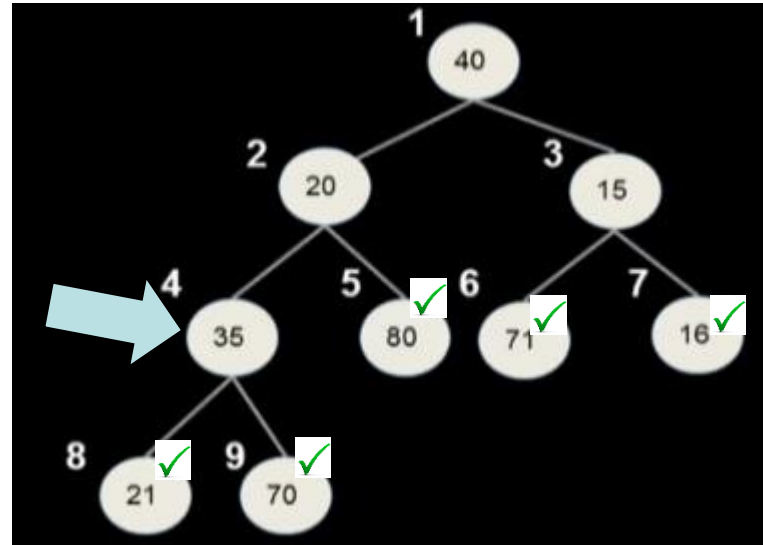
- Os nós folhas cumprem a propriedade de heap pois eles não têm filhos!





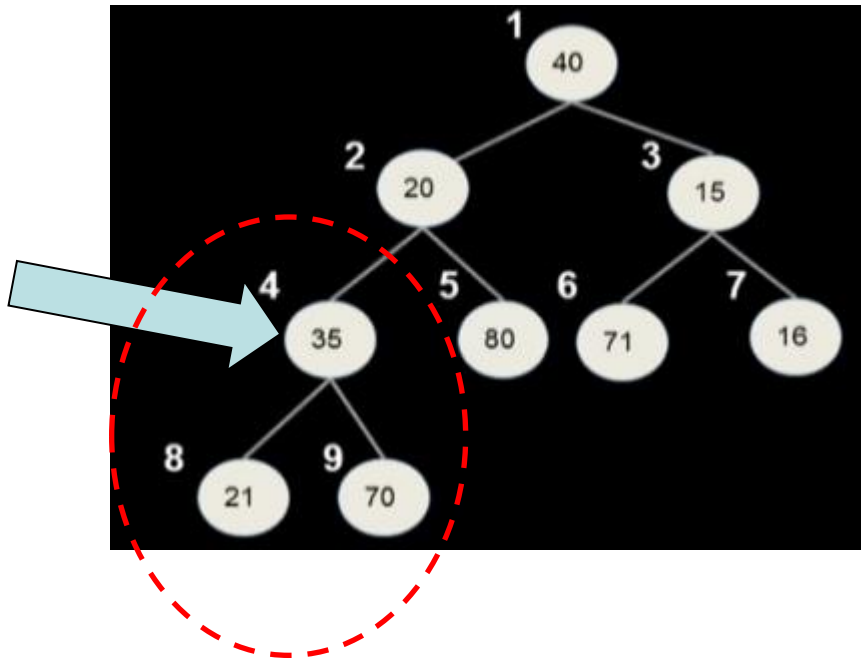
Construção de um Max Heap

- O nó **35** não cumpre a propriedade de heap;



Construção de um Max Heap

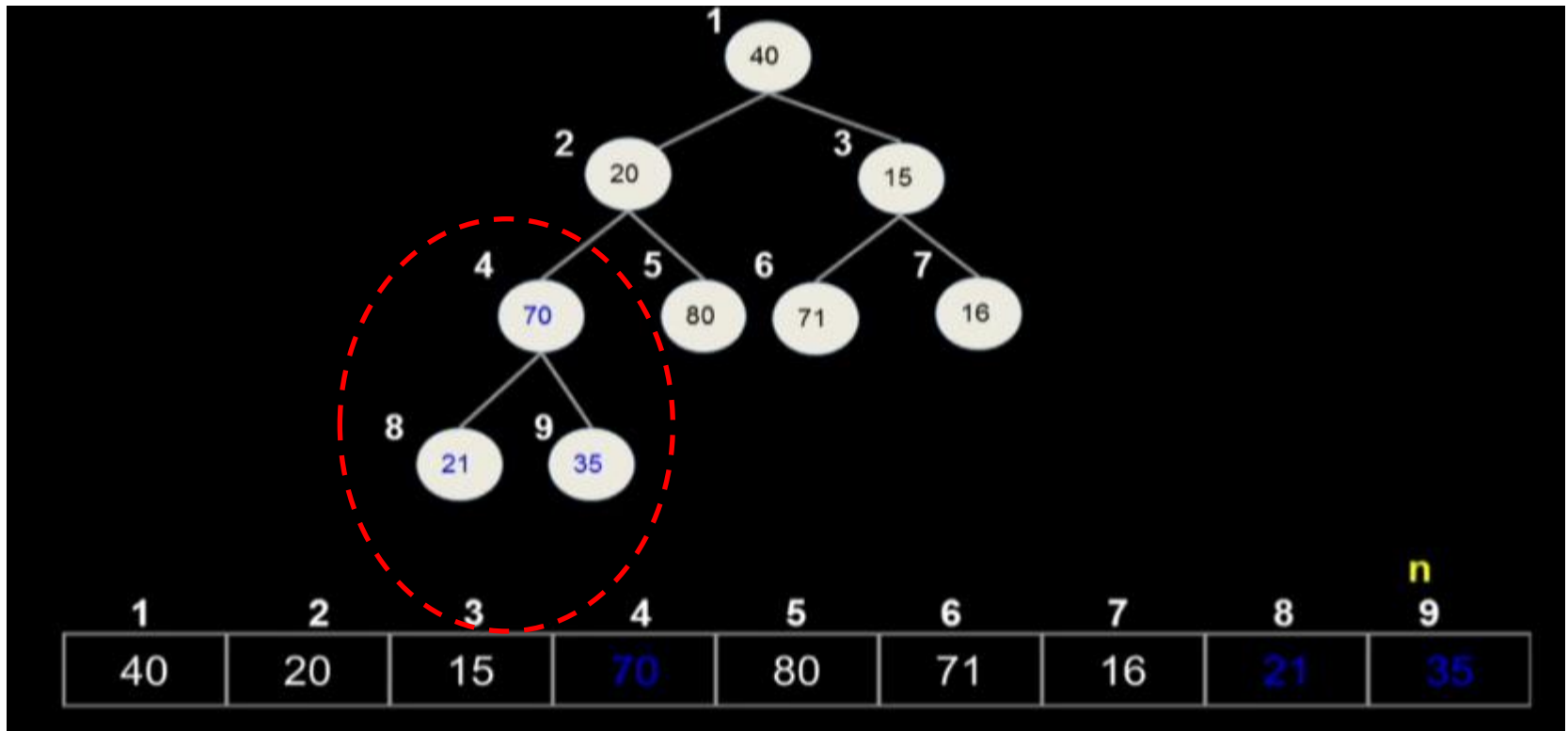
- O nó **35** não cumpre a propriedade de heap;
- Vamos trocá-lo com o nó 70 que é o maior dos filhos;





Construção de um Max Heap

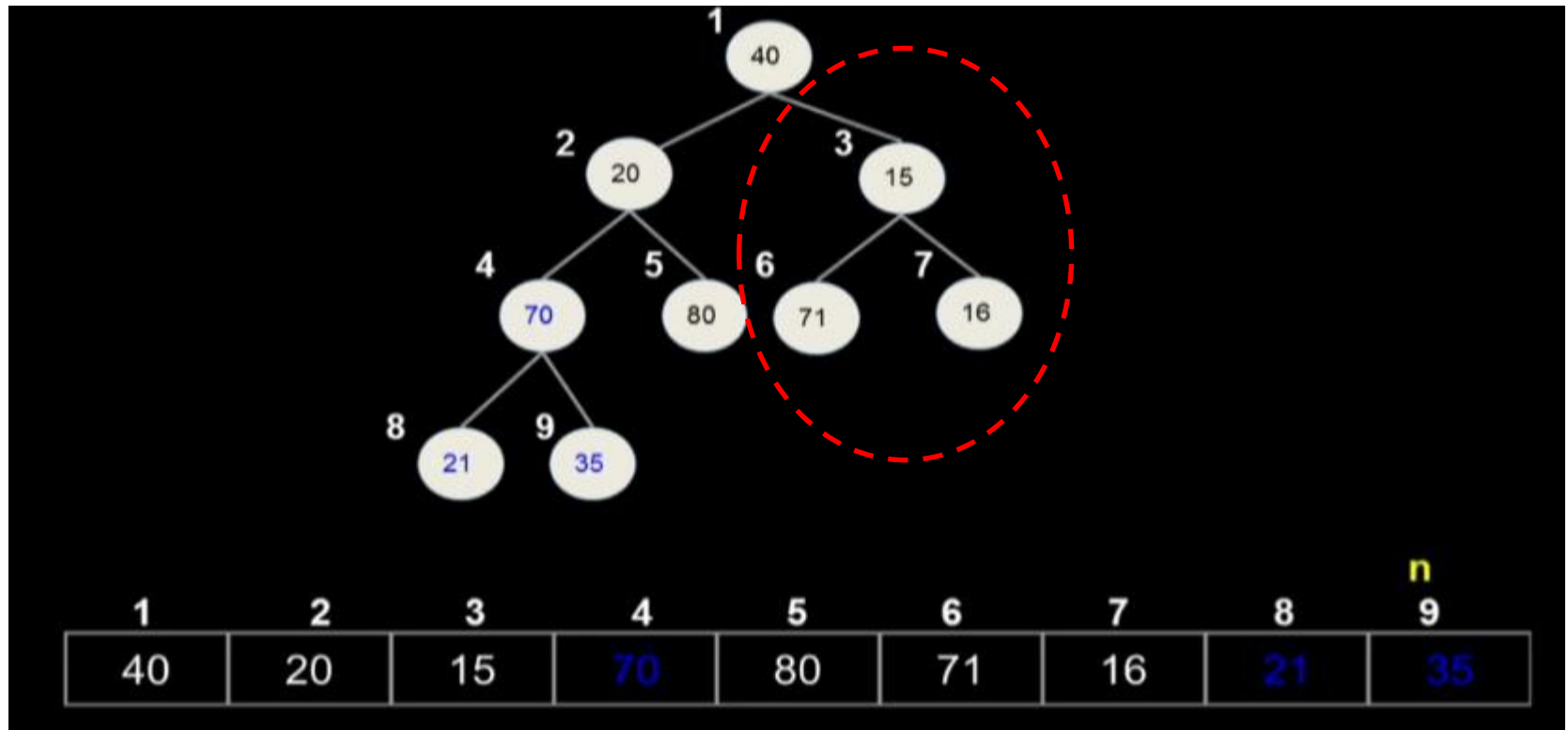
- O nó **35** não cumpre a propriedade de heap;
- Vamos trocá-lo com o nó 70 que é o maior dos filhos;





Construção de um Max Heap

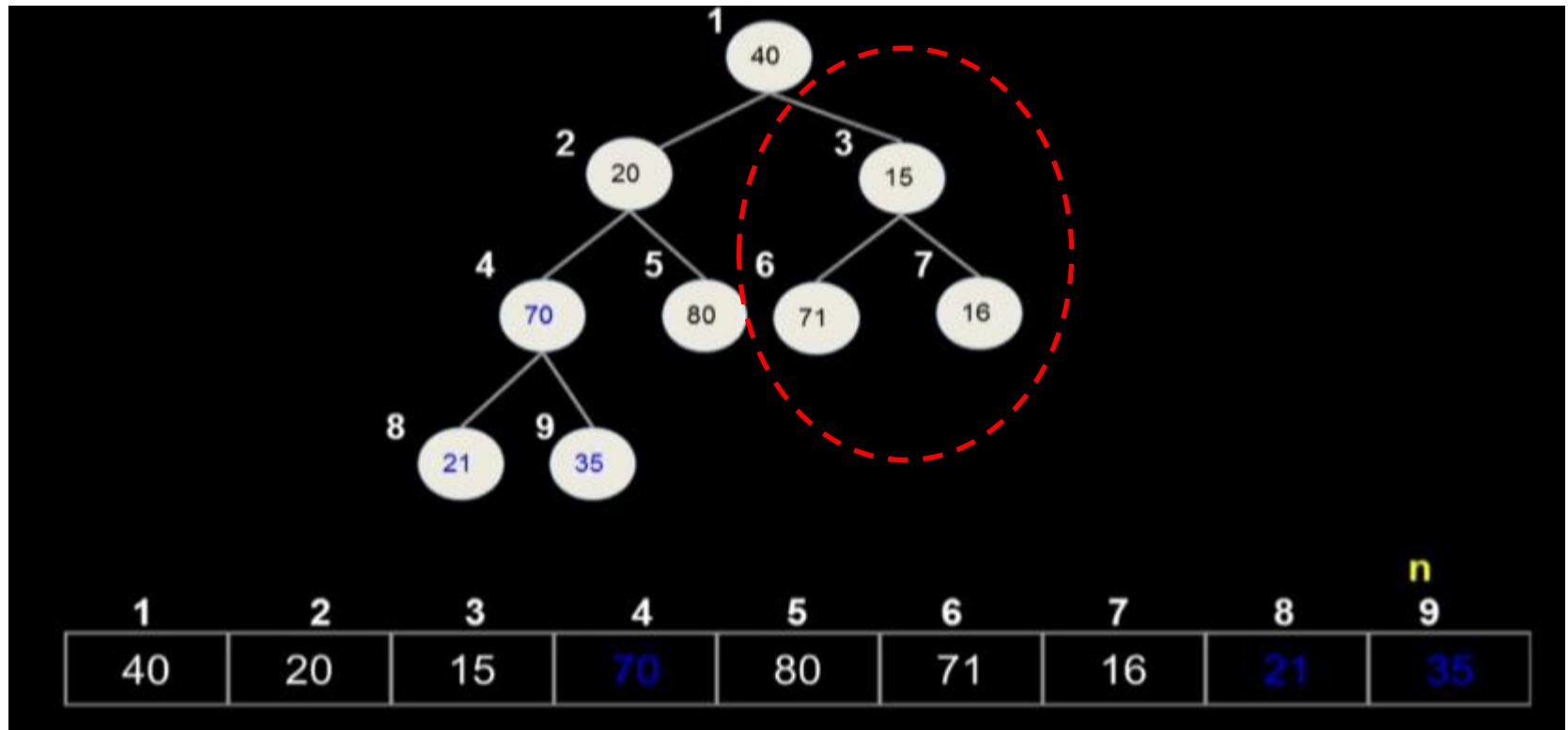
- O processo continua com o nó 15;





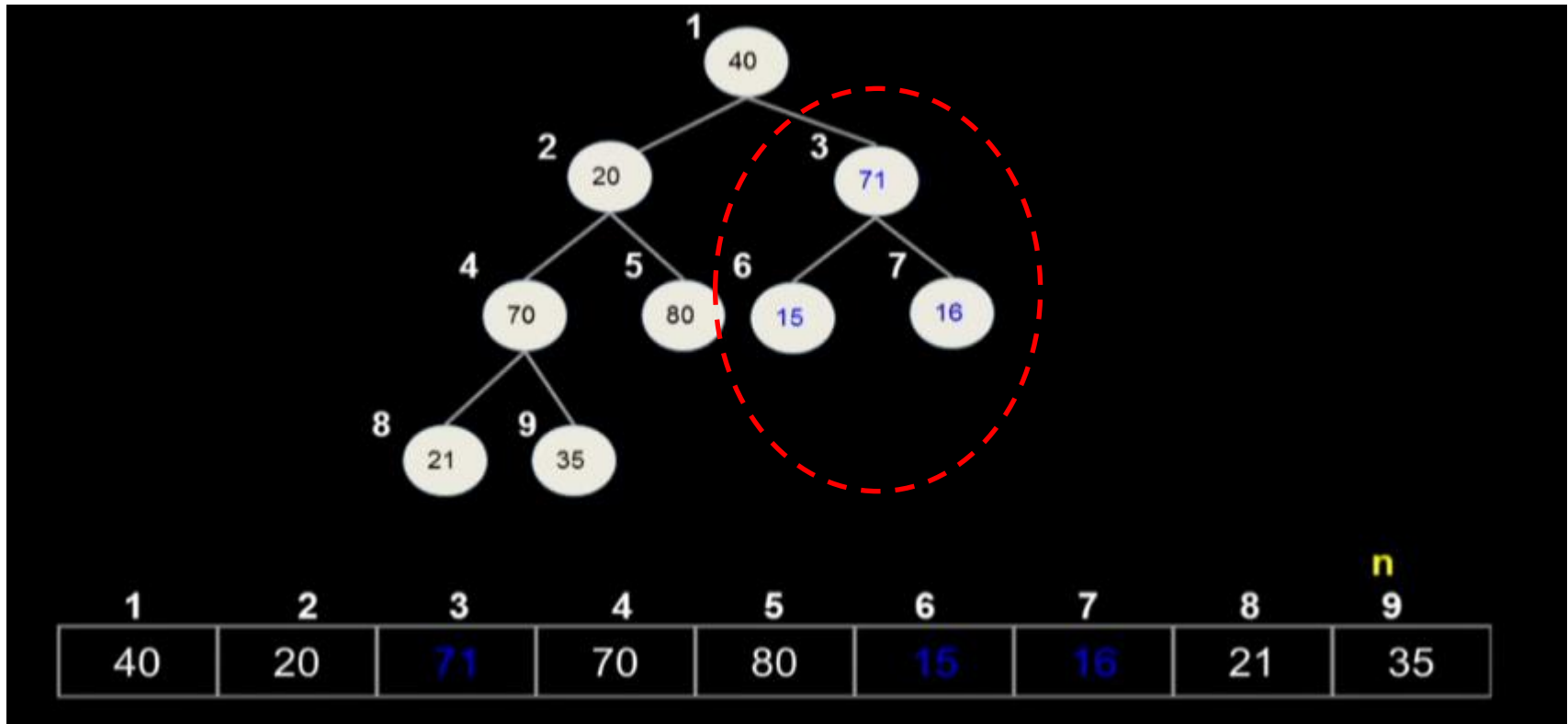
Construção de um Max Heap

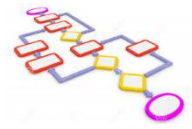
- O processo continua com o nó 15;
- O nó com valor 15 é trocado com o nó 71



Construção de um Max Heap

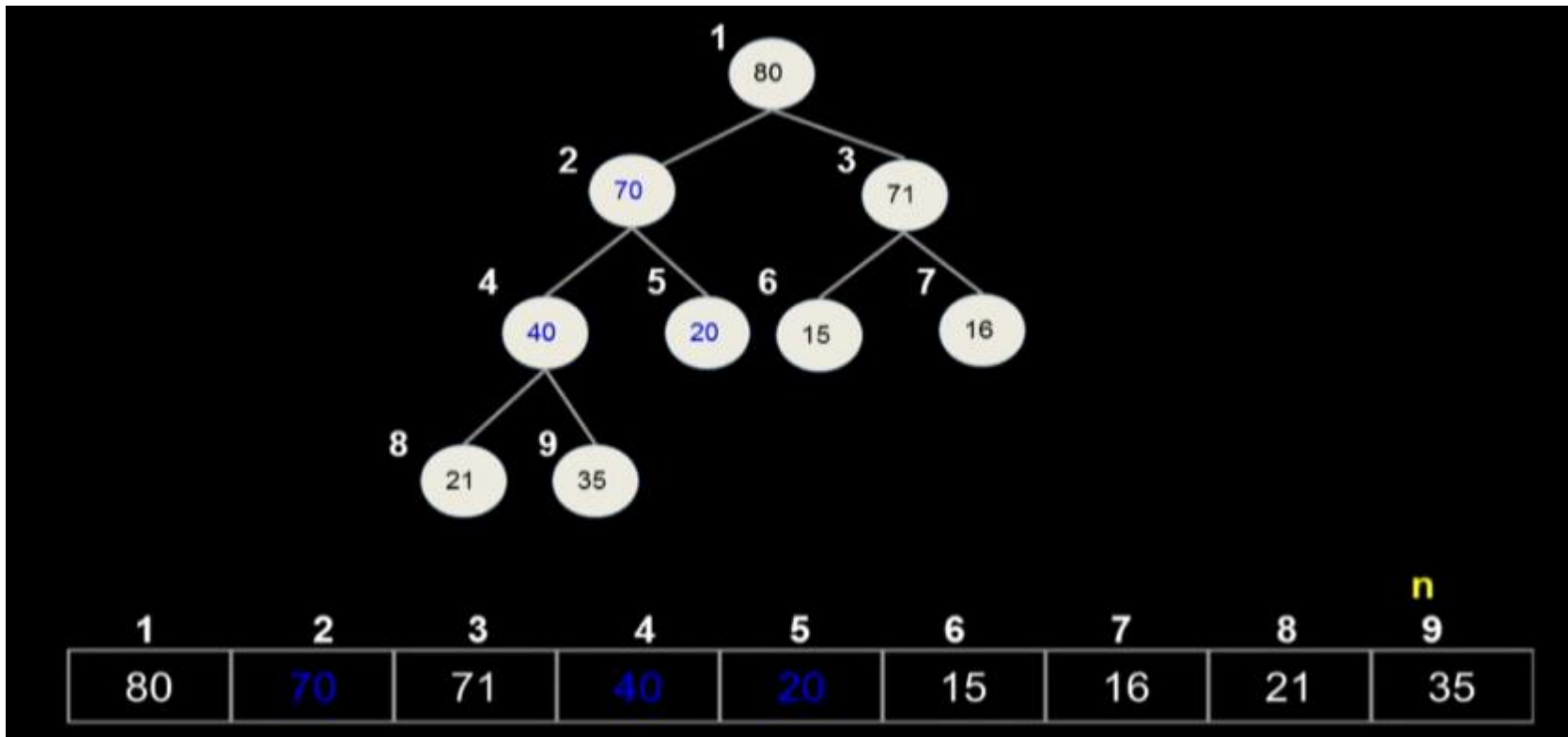
- O processo continua com o nó 15;
- O nó com valor 15 é trocado com o nó 71

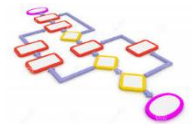




Construção de um Max Heap

- O processo continua;
- Na verdade, o que se está fazendo é para cada nó não folha, aplica-se o algoritmo max-heapfy, até finalizar-se a árvore com a estrutura de **heap**.





Construção de um Max Heap

BUILD-MAX-HEAP (A, n)

1. para $i \leftarrow \lfloor n/2 \rfloor$ decrescendo até 1 faça
2. **MAX-HEAPIFY (A, n, i)**

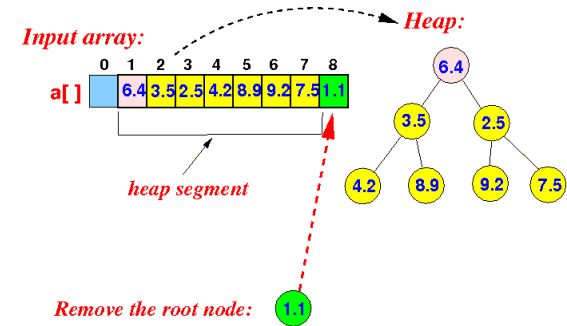
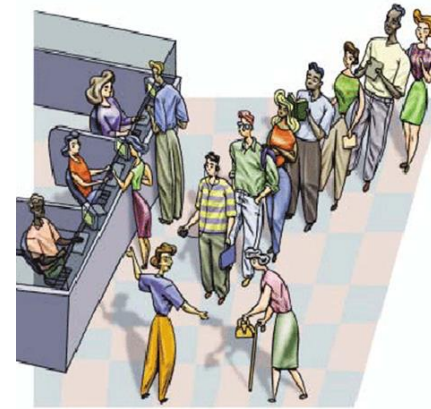
- Mostra-se que a complexidade do algoritmo BUILD-MAX-HEAP é $O(n)$





Aplicações de Heap

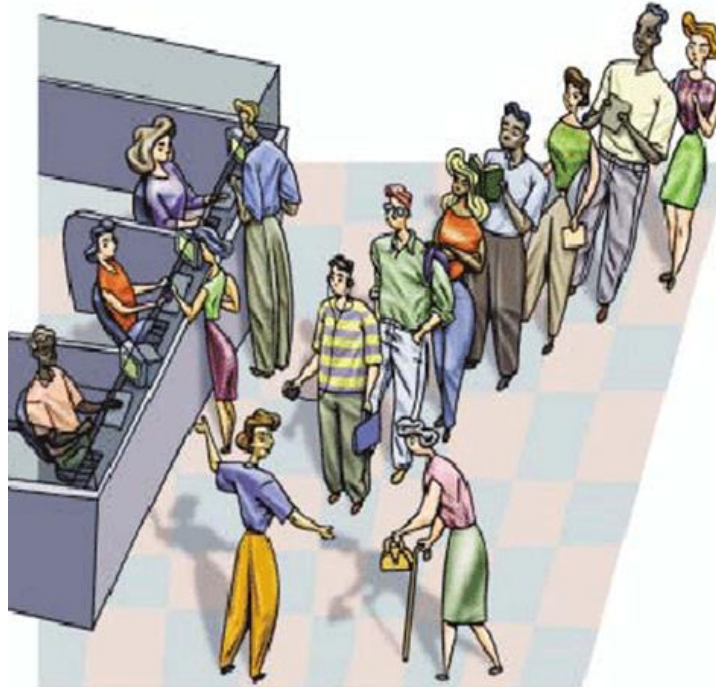
- Fila de Prioridade
- Heapsort

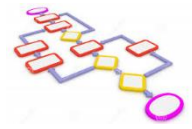




Fila de Prioridade

- É uma fila que permite que elementos sejam adicionados associados a uma **prioridade**;
- Cada elemento na fila deve possuir um dado adicional que representa sua **prioridade** de atendimento;
- Uma regra explícita define que o elemento de maior **prioridade** (o que tem o maior número associado) deve ser o primeiro a ser removido da fila, quando uma remoção for requerida;





Fila de Prioridade

- São empregadas em diversas aplicações;
- Por exemplo, em filas de bancos geralmente há um esquema de prioridade em que clientes preferenciais, idosos ou mulheres grávidas possuem maior prioridade de atendimento quando comparados aos demais clientes;





Fila de Prioridade – Operações

- **Inserir** com prioridade;
- **Remover** do elemento com mais alta prioridade;
- **Alterar** a prioridade de um determinado elemento;
- **Retornar** o número de elementos existentes na fila de prioridade;
- **Testar** a existência de elementos de mesma prioridade.

