

# Capítulo VII

## Tipo Abstrato de Dados Partição

(Union-Find, Disjoint Sets or Partition ADT)

# TAD Partição (com $n$ elementos)

Os elementos dos conjuntos são  $0, 1, 2, \dots, n - 1$ . Cada conjunto é identificado por um dos seus elementos, denominado **o representante** do conjunto.

Domínio =  $\{0, 1, \dots, n - 1\}$

// Cria a partição  $\{\{0\}, \{1\}, \dots, \{n - 1\}\}$ .

Partição **cria**( int  $n$  );

// Devolve o representante do conjunto ao qual  $e$  pertence.

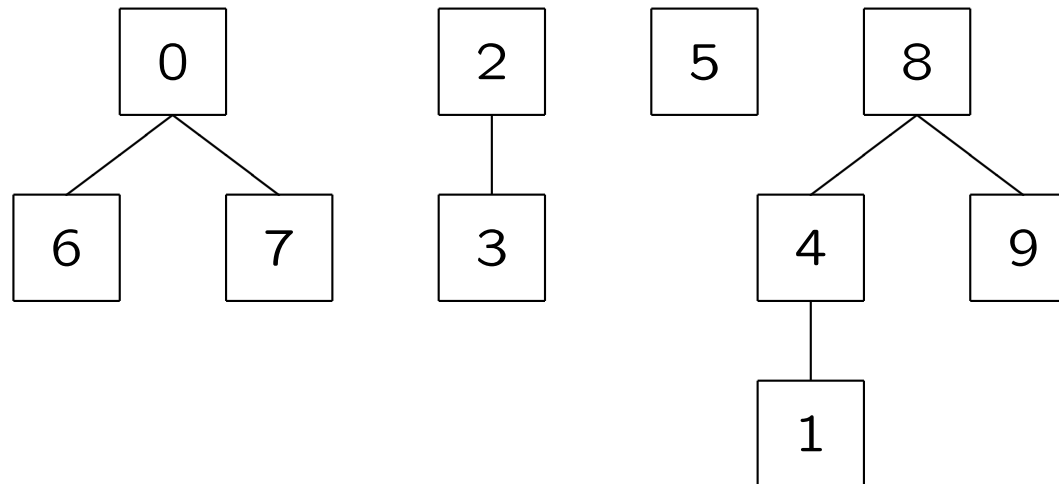
Domínio **representante**( Domínio  $e$  );

// Substitui os conjuntos  $C_e$  e  $C_f$ , cujos representantes são  $e$  e  $f$ ,  
// respetivamente, pelo conjunto  $C_e \cup C_f$ .

// **Pré-condição:**  $e \neq f$  (ou seja,  $C_e \neq C_f$ ).

**void reunião**( Domínio  $e$ , Domínio  $f$  );

# Implementação em Floresta



Complexidade (com  $n$  elementos)

**criação**  $\Theta(?)$

**representante**  $O(?)$

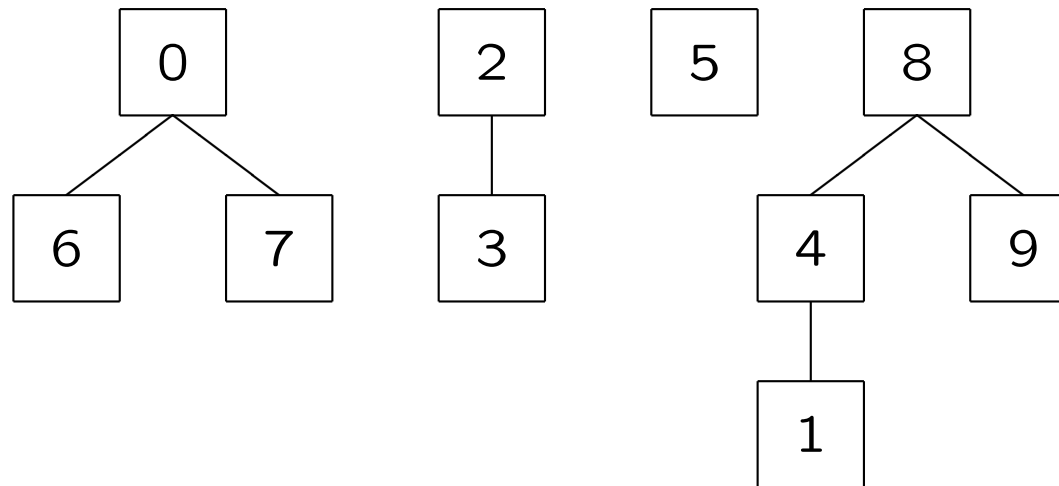
**reunião**  $\Theta(?)$

(no pior caso)

# Implementação em Floresta

(implementada em vetor)

|    |   |    |   |   |    |   |   |    |   |
|----|---|----|---|---|----|---|---|----|---|
| -1 | 4 | -1 | 2 | 8 | -1 | 0 | 0 | -1 | 8 |
| 0  | 1 | 2  | 3 | 4 | 5  | 6 | 7 | 8  | 9 |



Complexidade (com  $n$  elementos)

criação  $\Theta(?)$

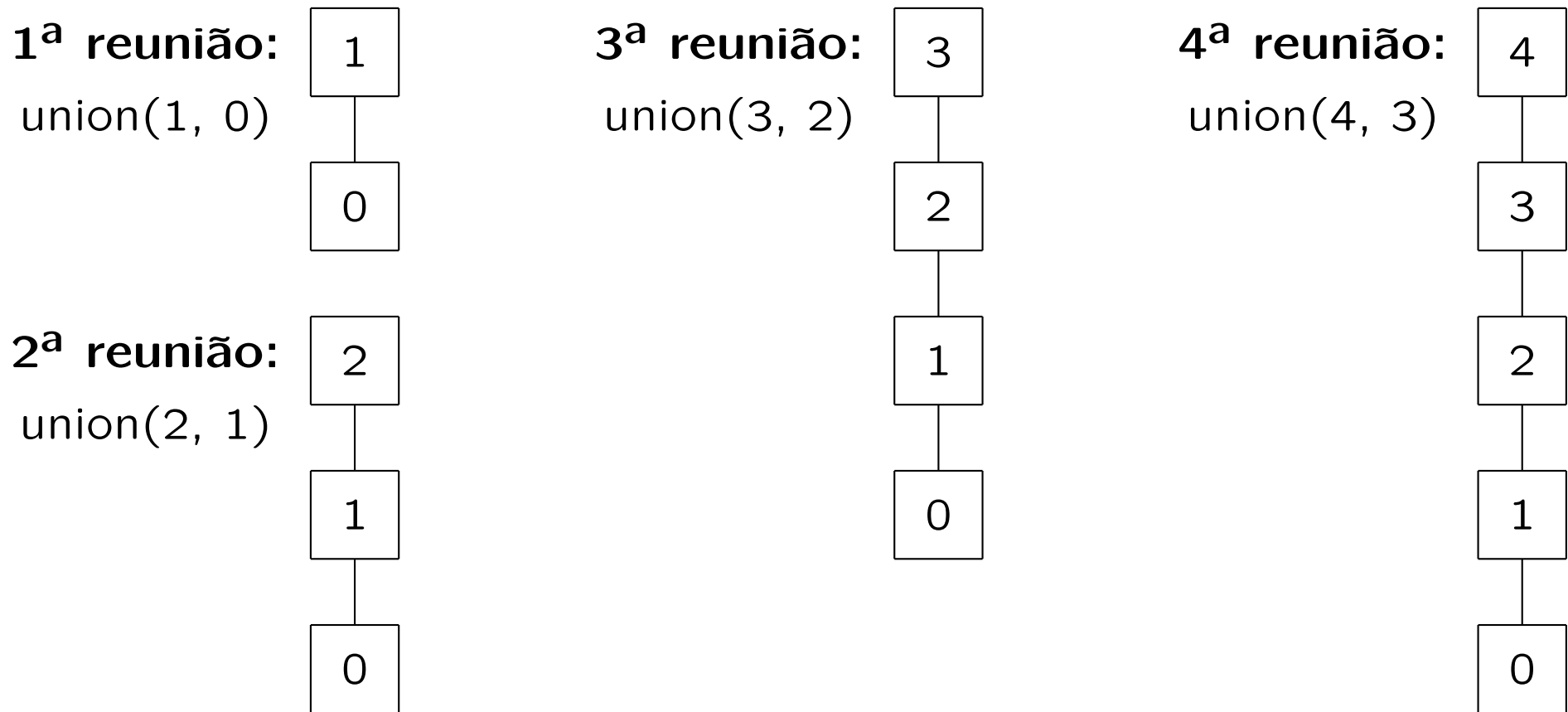
representante  $O(?)$

reunião  $\Theta(?)$

(no pior caso)

# Complexidade do Representante com Reunião **sem Estratégia**

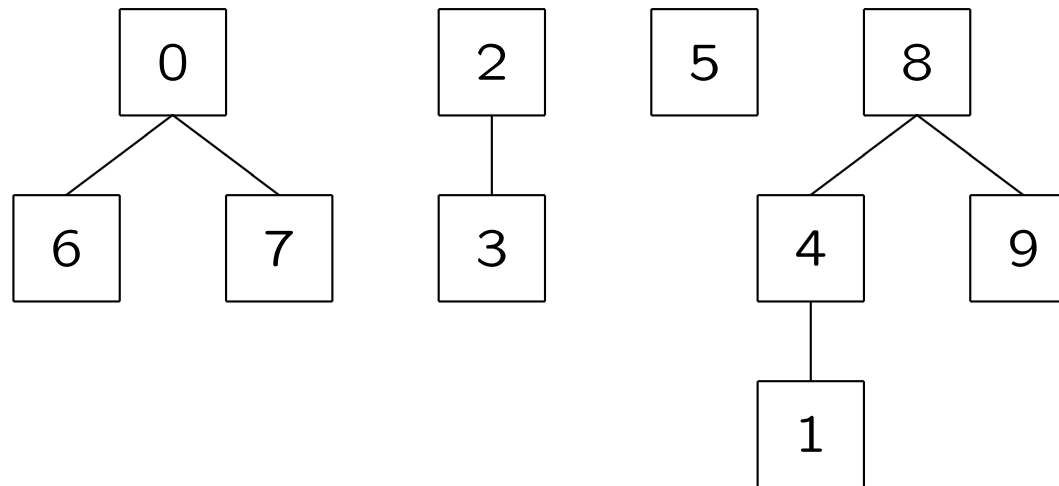
Raiz da nova árvore: primeiro representante



# Implementação em Floresta

(implementada em vetor)

|    |   |    |   |   |    |   |   |    |   |
|----|---|----|---|---|----|---|---|----|---|
| -1 | 4 | -1 | 2 | 8 | -1 | 0 | 0 | -1 | 8 |
| 0  | 1 | 2  | 3 | 4 | 5  | 6 | 7 | 8  | 9 |



**Complexidade (com  $n$  elementos)**

**criação**  $\Theta(n)$

**representante**  $O(n)$

**reunião**  $\Theta(1)$

(no pior caso)

# Complexidade do Algoritmo de Kruskal

## Reunião sem Estratégia

## Representante sem Efeitos Laterais

|                                                 |                     |
|-------------------------------------------------|---------------------|
| criação do heap                                 | $\Theta( A )$       |
| criação da partição                             | $\Theta( V )$       |
| criação do vetor resultado                      | $\Theta(1)$         |
| Ciclo (executado entre $ V  - 1$ e $ A $ vezes) | (ver próximo slide) |
| 1 remoção do mínimo                             | $O(\log  A )$       |
| 2 representante                                 | $O( V )$            |
| Ciclo (executado $ V  - 1$ vezes)               |                     |
| 1 inserção no vetor                             | $\Theta(1)$         |
| 1 reunião                                       | $\Theta(1)$         |
| <b>TOTAL</b>                                    | $O( A  \times  V )$ |

# Complexidade do Algoritmo de Kruskal

Reunião sem Estratégia

Representante sem Efeitos Laterais

## Complexidade do Primeiro Ciclo

$$O( |A| \times \log |A| + |A| \times (2R) )$$

$$O( |A| \times \underbrace{\log |A|}_{\log |A| < 2 \log |V| \text{ porque } |A| < |V|^2} + |A| \times |V| )$$

$$O( |A| \times \log |V| + |A| \times |V| )$$

$$O( |A| \times |V| )$$

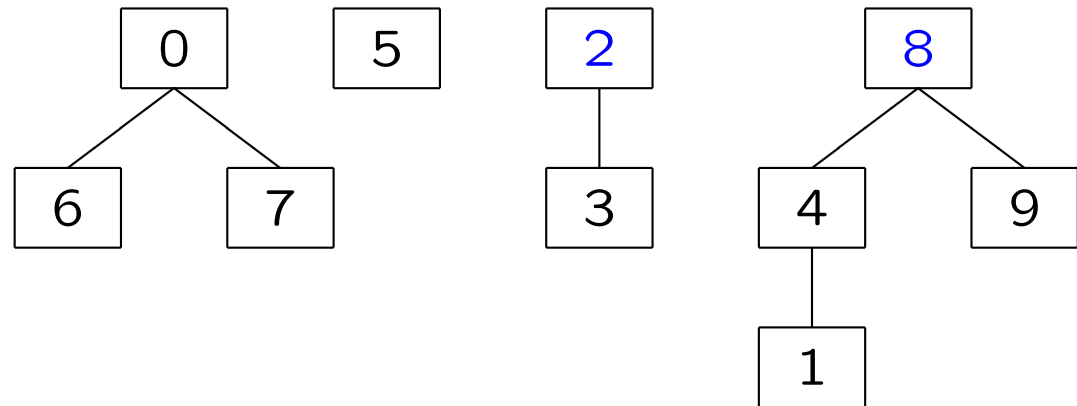
Reunião

por

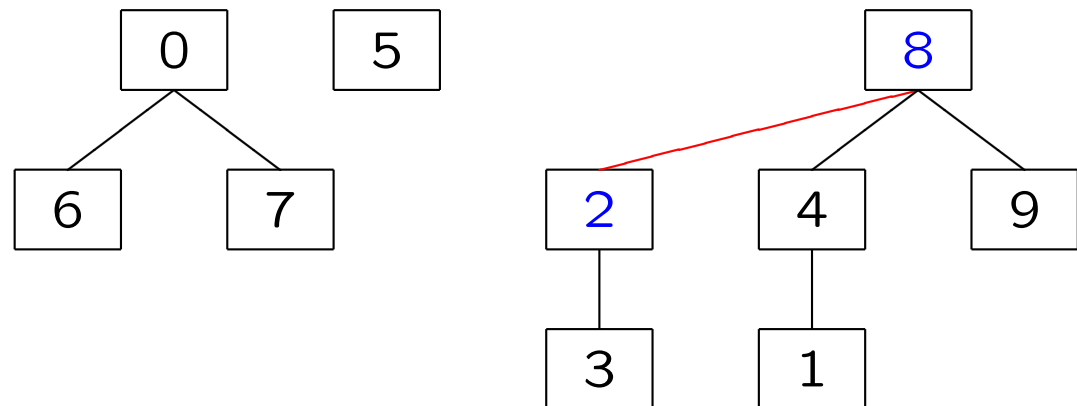
Altura

(2, 8)

|    |   |    |   |   |    |   |   |    |   |
|----|---|----|---|---|----|---|---|----|---|
| -2 | 4 | -2 | 2 | 8 | -1 | 0 | 0 | -3 | 8 |
| 0  | 1 | 2  | 3 | 4 | 5  | 6 | 7 | 8  | 9 |



|    |   |   |   |   |    |   |   |    |   |
|----|---|---|---|---|----|---|---|----|---|
| -2 | 4 | 8 | 2 | 8 | -1 | 0 | 0 | -3 | 8 |
| 0  | 1 | 2 | 3 | 4 | 5  | 6 | 7 | 8  | 9 |



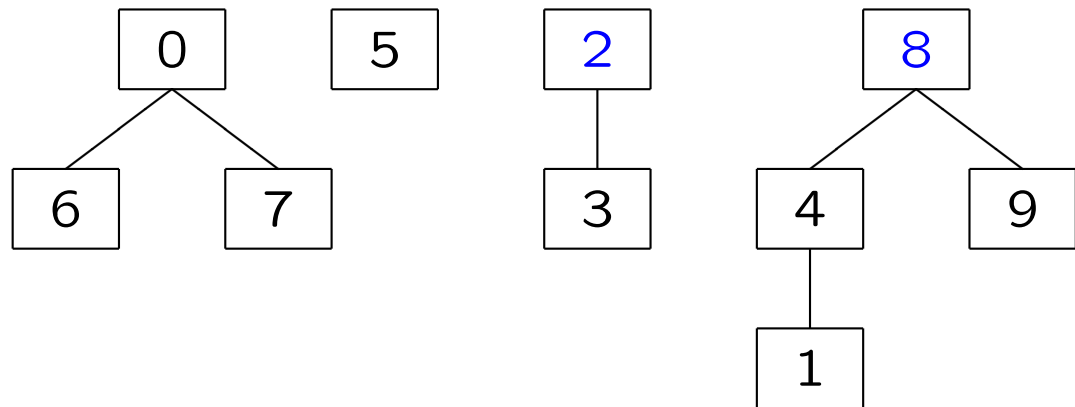
Reunião

por

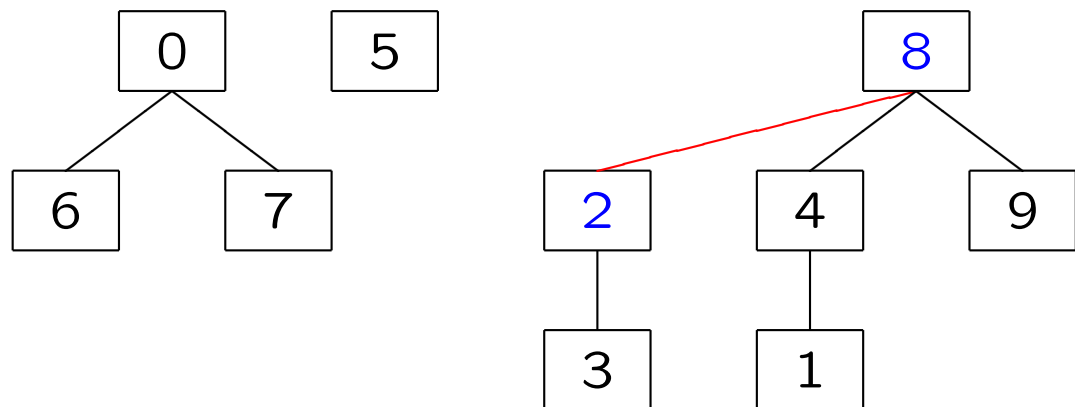
Tamanho

(2, 8)

|    |   |    |   |   |    |   |   |    |   |
|----|---|----|---|---|----|---|---|----|---|
| -3 | 4 | -2 | 2 | 8 | -1 | 0 | 0 | -4 | 8 |
| 0  | 1 | 2  | 3 | 4 | 5  | 6 | 7 | 8  | 9 |



|    |   |   |   |   |    |   |   |    |   |
|----|---|---|---|---|----|---|---|----|---|
| -3 | 4 | 8 | 2 | 8 | -1 | 0 | 0 | -6 | 8 |
| 0  | 1 | 2 | 3 | 4 | 5  | 6 | 7 | 8  | 9 |



# Interface Partição (com $n$ elementos)

```
public interface UnionFind {  
  
    // Creates the partition  $\{\{0\}, \{1\}, \dots, \{\text{domainSize} - 1\}\}$ .  
    // UnionFind( int domainSize );  
  
    // Returns the representative of the set that contains  
    // the specified element.  
    int find( int element ) throws InvalidElementException;  
  
    // Removes the two distinct sets  $S_1$  and  $S_2$  whose representatives  
    // are the specified elements, and inserts the set  $S_1 \cup S_2$ .  
    // The representative of the new set  $S_1 \cup S_2$  can be any of  
    // its members.  
    void union( int representative1, int representative2 ) throws  
        InvalidElementException, NotRepresentativeException,  
        EqualSetsException;  
}
```

# Classe Partição em Vetor

```
public class UnionFindInArray implements UnionFind {  
  
    // The partition is a forest implemented in an array.  
    protected int[] partition;  
  
    // Definition of the range of valid elements.  
    protected String validRangeMsg;
```

# Criar a Partição

```
// Creates the partition {{0},{1},...,{domainSize - 1}}.
public UnionFindInArray( int domainSize ) {
    partition = new int[domainSize];
    for ( int i = 0; i < domainSize; i++ )
        partition[i] = -1;

    int lastElement = domainSize - 1;
    validRangeMsg =
        "Range of valid elements: 0, 1, ..., " + lastElement;
}
```

# Métodos de Validação

```
protected boolean isInTheDomain( int number ) {  
    return ( number >= 0 ) && ( number < partition.length );  
}
```

// **Pre-condition:**  $0 \leq \text{element} < \text{partition.length}$ .

```
protected boolean isRepresentative( int element ) {  
    return partition[element] < 0;  
}
```

# Representante — Recursivo

```
public int find( int element ) throws InvalidElementException {  
    if ( !this.isInTheDomain(element) )  
        throw new InvalidElementException(validRangeMsg);  
  
    return this.findRec(element);  
}
```

// **Pre-condition:**  $0 \leq \text{element} < \text{partition.length}$ .

```
protected int findRec( int element ) {  
    if ( partition[element] < 0 )  
        return element;  
  
    return this.findRec( partition[element] );  
}
```

# Representante — Iterativo

```
public int find( int element ) throws InvalidElementException {  
    if ( !this.isInTheDomain(element) )  
        throw new InvalidElementException(validRangeMsg);  
  
    int node = element;  
    while ( partition[node] >= 0 )  
        node = partition[node];  
    return node;  
}
```

## Reunião por Tamanho (*Union by Size*)

```
public void union( int rep1, int rep2 ) throws  
    InvalidElementException, NotRepresentativeException,  
    EqualSetsException {  
    if ( !this.isInTheDomain(rep1) || !this.isInTheDomain(rep2) )  
        throw new InvalidElementException(validRangeMsg);  
    if ( !this.isRepresentative(rep1) )  
        throw new NotRepresentativeException( "First argument" );  
    if ( !this.isRepresentative(rep2) )  
        throw new NotRepresentativeException( "Second argument" );  
    if ( rep1 == rep2 )  
        throw new EqualSetsException( "The two arguments are equal" );
```

# Reunião por Tamanho

```
if ( partition[rep1] <= partition[rep2] ) {  
    // Size(S1) >= Size(S2).  
    partition[rep1] += partition[rep2];  
    partition[rep2] = rep1;  
}  
else {  
    // Size(S1) < Size(S2).  
    partition[rep2] += partition[rep1];  
    partition[rep1] = rep2;  
}  
}
```

## Reunião por Altura (*Union by Height*)

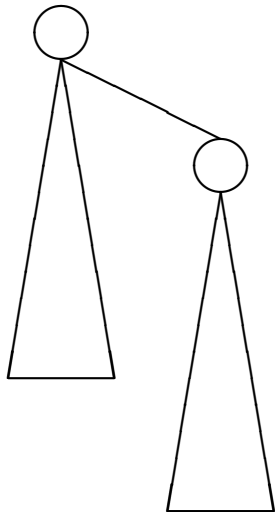
```
public void union( int rep1, int rep2 ) throws  
    InvalidElementException, NotRepresentativeException,  
    EqualSetsException {  
    if ( !this.isInTheDomain(rep1) || !this.isInTheDomain(rep2) )  
        throw new InvalidElementException(validRangeMsg);  
    if ( !this.isRepresentative(rep1) )  
        throw new NotRepresentativeException( "First argument" );  
    if ( !this.isRepresentative(rep2) )  
        throw new NotRepresentativeException( "Second argument" );  
    if ( rep1 == rep2 )  
        throw new EqualSetsException( "The two arguments are equal" );
```

# Reunião por Altura

```
if ( partition[rep1] <= partition[rep2] ) {  
    // Height(S1) >= Height(S2).  
    if ( partition[rep1] == partition[rep2] )  
        partition[rep1]--;  
    partition[rep2] = rep1;  
}  
else  
    // Height(S1) < Height(S2).  
    partition[rep1] = rep2;  
}
```

# Complexidade do Representante com Reunião por **Altura**

Número Mínimo de Nós  
de uma Árvore com Altura  $h$



$$N(1) = 1$$

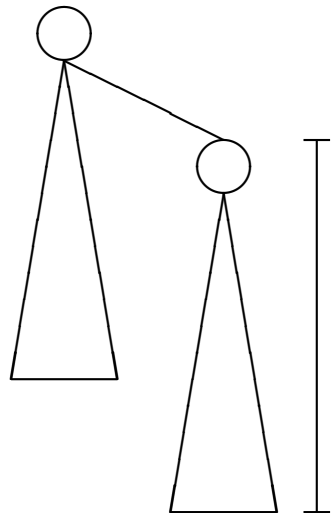
$$N(2) = 2$$

$$N(h) =$$

(para  $h \geq 2$ )

# Complexidade do Representante com Reunião por **Altura**

Número Mínimo de Nós  
de uma Árvore com Altura  $h$



$$N(1) = 1$$

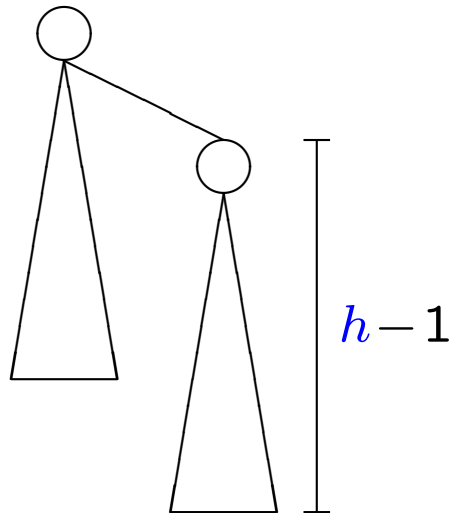
$$N(2) = 2$$

$$N(h) =$$

(para  $h \geq 2$ )

# Complexidade do Representante com Reunião por **Altura**

Número Mínimo de Nós  
de uma Árvore com Altura  $h$



$$N(1) = 1$$

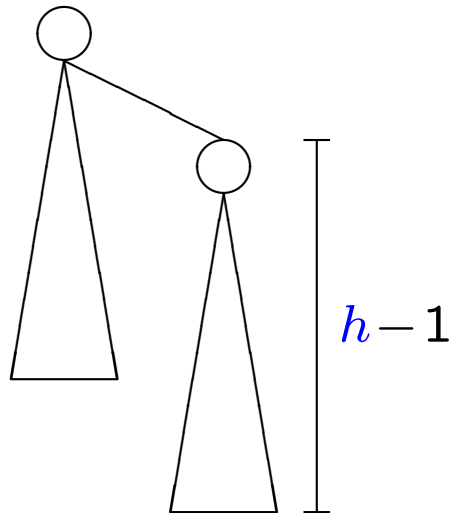
$$N(2) = 2$$

$$N(h) = 2 \times N(h-1)$$

(para  $h \geq 2$ )

# Complexidade do Representante com Reunião por **Altura**

Número Mínimo de Nós  
de uma Árvore com Altura  $h$



$$N(1) = 1 = 2^0$$

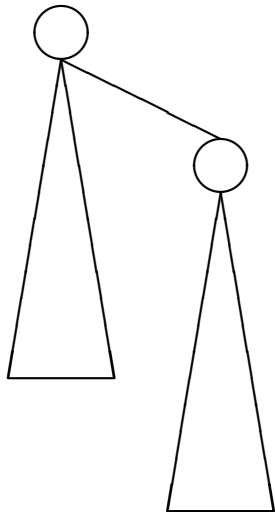
$$N(2) = 2 = 2^1$$

$$N(h) = 2 \times N(h-1) \stackrel{H.I.}{=} 2 \times 2^{h-2} = 2^{h-1}$$

(para  $h \geq 2$ )

# Complexidade do Representante com Reunião por **Tamanho**

Número Mínimo de Nós  
de uma Árvore com Altura  $h$



$$N(1) = 1$$

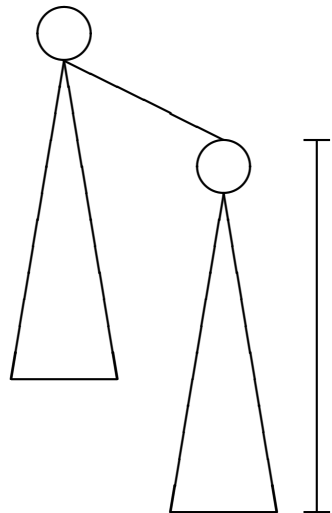
$$N(2) = 2$$

$$N(h) =$$

(para  $h \geq 2$ )

# Complexidade do Representante com Reunião por **Tamanho**

Número Mínimo de Nós  
de uma Árvore com Altura  $h$



$$N(1) = 1$$

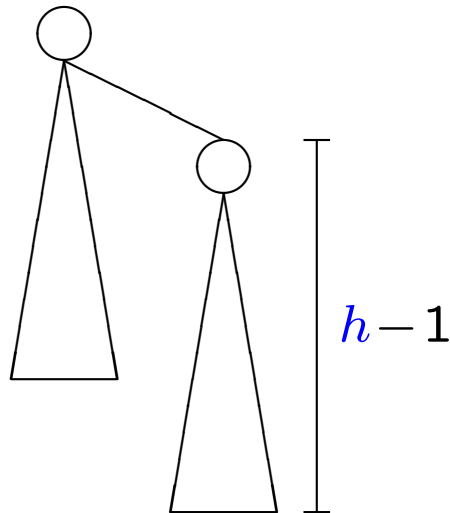
$$N(2) = 2$$

$$N(h) =$$

(para  $h \geq 2$ )

# Complexidade do Representante com Reunião por **Tamanho**

Número Mínimo de Nós  
de uma Árvore com Altura  $h$



$$N(1) = 1$$

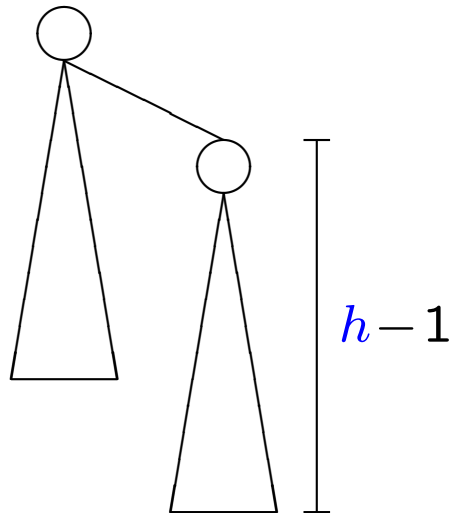
$$N(2) = 2$$

$$N(h) = 2 \times N(h-1)$$

(para  $h \geq 2$ )

# Complexidade do Representante com Reunião por **Tamanho**

Número Mínimo de Nós  
de uma Árvore com Altura  $h$



$$N(1) = 1 = 2^0$$

$$N(2) = 2 = 2^1$$

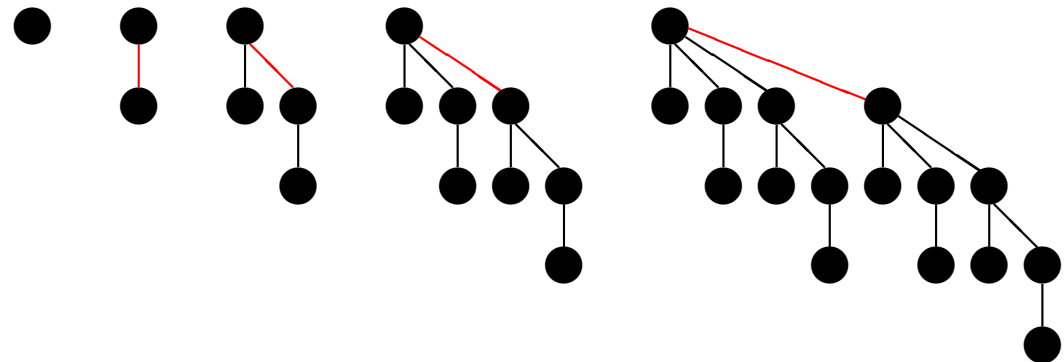
$$N(h) = 2 \times N(h-1) \stackrel{H.I.}{=} 2 \times 2^{h-2} = 2^{h-1}$$

(para  $h \geq 2$ )

# Complexidade do Representante com Reunião por **Altura** ou por **Tamanho**

Número Mínimo de Nós  
de uma Árvore com Altura  $h$

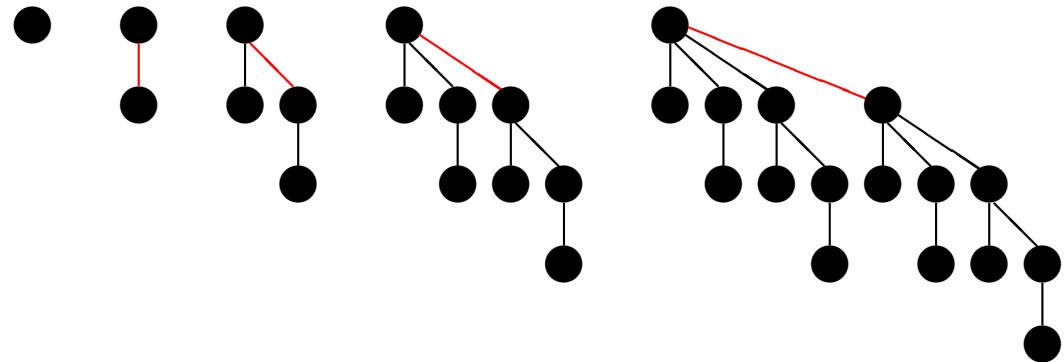
| Altura $h$ | Nº mín. nós |
|------------|-------------|
| 1          | $1 = 2^0$   |
| 2          | $2 = 2^1$   |
| 3          | $4 = 2^2$   |
| 4          | $8 = 2^3$   |
| 5          | $16 = 2^4$  |
| 6          | $32 = 2^5$  |
| 7          | $64 = 2^6$  |



# Complexidade do Representante com Reunião por **Altura** ou por **Tamanho**

Altura Máxima de uma Árvore com  $n$  nós

| Altura $h$ | Nº mín. nós |
|------------|-------------|
| 1          | $1 = 2^0$   |
| 2          | $2 = 2^1$   |
| 3          | $4 = 2^2$   |
| 4          | $8 = 2^3$   |
| 5          | $16 = 2^4$  |
| 6          | $32 = 2^5$  |
| 7          | $64 = 2^6$  |



Qual é a maior altura de uma árvore, se o domínio tiver  $n = 20$  nós?

# Complexidade do Representante com Reunião por **Altura** ou por **Tamanho**

Altura Máxima de uma Árvore com  $n$  nós

Dado  $n$  (o número total de nós), existe  $h$  tal que:

$$\underbrace{2^{h-1}}_{\substack{h \text{ é a maior altura} \\ \text{com } n \text{ nós}}} \leq n < \underbrace{2^h}_{\substack{\text{número mínimo de nós} \\ \text{para altura } h + 1}}$$

$$2^{h-1} \leq n$$

$$h - 1 \leq \log(n)$$

$$h \leq 1 + \log(n)$$

# Complexidade do Algoritmo de Kruskal

Reunião por Altura ou por Tamanho

Representante sem Efeitos Laterais

|                                                 |                          |
|-------------------------------------------------|--------------------------|
| criação do heap                                 | $\Theta( A )$            |
| criação da partição                             | $\Theta( V )$            |
| criação do vetor resultado                      | $\Theta(1)$              |
| Ciclo (executado entre $ V  - 1$ e $ A $ vezes) | (ver próximo slide)      |
| 1 remoção do mínimo                             | $O(\log  A )$            |
| 2 representante                                 | $O(\log  V )$            |
| Ciclo (executado $ V  - 1$ vezes)               |                          |
| 1 inserção no vetor                             | $\Theta(1)$              |
| 1 reunião                                       | $\Theta(1)$              |
| <b>TOTAL</b>                                    | $O( A  \times \log  V )$ |

# Complexidade do Algoritmo de Kruskal

Reunião por Altura ou por Tamanho

Representante sem Efeitos Laterais

## Complexidade do Primeiro Ciclo

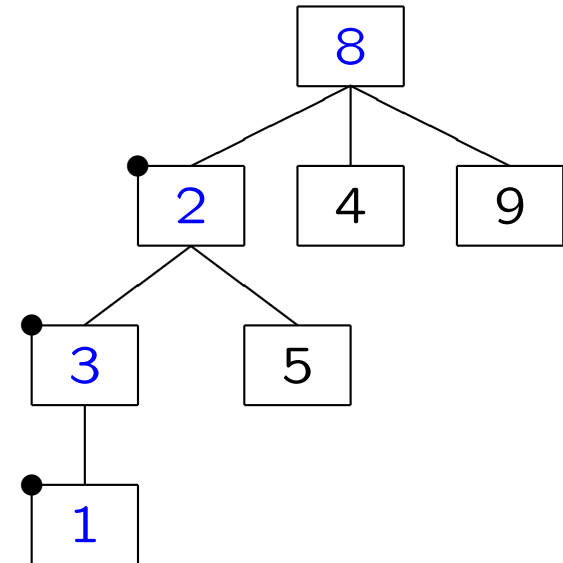
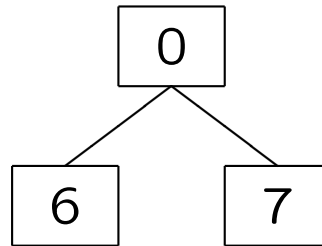
$$O(|A| \times \log |A| + |A| \times (2R))$$

$$O(|A| \times \log |V| + |A| \times \log |V|)$$

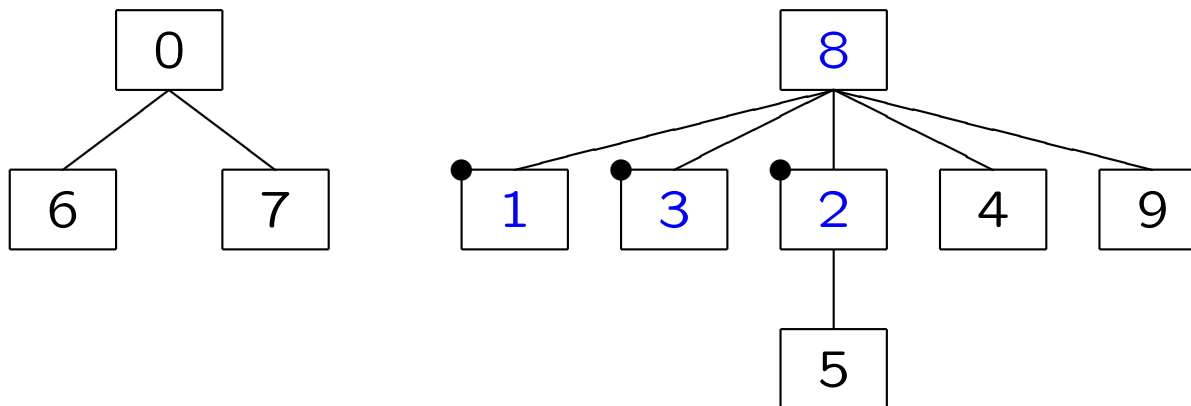
$$O(|A| \times \log |V|)$$

# Compressão do Caminho **find(1)**

|    |   |   |   |   |   |   |   |    |   |
|----|---|---|---|---|---|---|---|----|---|
| -3 | 3 | 8 | 2 | 8 | 2 | 0 | 0 | -7 | 8 |
| 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8  | 9 |



|    |   |   |   |   |   |   |   |    |   |
|----|---|---|---|---|---|---|---|----|---|
| -3 | 8 | 8 | 8 | 8 | 2 | 0 | 0 | -7 | 8 |
| 0  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8  | 9 |



# Representante com Compressão do Caminho

```
public int find( int element ) throws InvalidElementException {  
    if ( !this.isInTheDomain(element) )  
        throw new InvalidElementException(validRangeMsg);  
  
    return this.findPathCompr(element);  
}
```

```
protected int findPathCompr( int element ) {  
    if ( partition[element] < 0 )  
        return element;  
  
    int root = this.findPathCompr( partition[element] );  
    partition[element] = root;  
  
    return root;  
}
```

Complexidade no PIOR CASO de  
 $U$  operações de **reunião** e  $R$  operações de **representante**  
 (com  $n$  elementos)

|                                    |             |               |
|------------------------------------|-------------|---------------|
| Reunião sem Estratégia             | $\Theta(1)$ | } $O(U + Rn)$ |
| Representante sem Efeitos Laterais | $O(n)$      |               |

|                                    |             |                     |
|------------------------------------|-------------|---------------------|
| Reunião por Altura ou Tamanho      | $\Theta(1)$ | } $O(U + R \log n)$ |
| Representante sem Efeitos Laterais | $O(\log n)$ |                     |

|                                                     |             |                       |
|-----------------------------------------------------|-------------|-----------------------|
| Reunião por <b>Nível</b> (código Altura) ou Tamanho | $\Theta(1)$ | } $O(k \alpha(k, n))$ |
| Representante com Compressão do Caminho             | $O(\log n)$ |                       |

se  $k = U + R \geq n$  [Tarjan 75].

Valor de  $\alpha(k, n)$  ( $k \geq n \geq 1$ )

$$2^2$$

$$4$$

$$2^{2^2}$$

$$2^4$$

$$16$$

$$2^{2^{2^2}}$$

$$2^{16}$$

$$65536$$

$$2^{2^{2^{2^2}}}\}^4$$

$$2^{65536}$$

$\approx 20\,000$  algoritmos

**Na prática,**

$$\alpha(k, n) \leq 4$$

porque

$$2^{2^{\dots^2}}\}^{16} > \log n.$$

# Complexidade do Algoritmo de Kruskal

Reunião por Nível ou por Tamanho

Representante com Compressão do Caminho

|                                                 |                          |
|-------------------------------------------------|--------------------------|
| criação do heap                                 | $\Theta( A )$            |
| criação da partição                             | $\Theta( V )$            |
| criação do vetor resultado                      | $\Theta(1)$              |
| Ciclo (executado entre $ V  - 1$ e $ A $ vezes) | (ver próximo slide)      |
| 1 remoção do mínimo                             | $O(\log  A )$            |
| 2 representante                                 | $O(\log  V )$            |
| Ciclo (executado $ V  - 1$ vezes)               |                          |
| 1 inserção no vetor                             | $\Theta(1)$              |
| 1 reunião                                       | $\Theta(1)$              |
| <b>TOTAL</b>                                    | $O( A  \times \log  V )$ |

# Complexidade do Algoritmo de Kruskal

Reunião por Nível ou por Tamanho

Representante com Compressão do Caminho

## Complexidade do Primeiro Ciclo

$$O( |A| \times \log |A| + \underbrace{|A| \times (2R)}_{2|A| \geq 2(|V|-1) \geq |V|} )$$

$$O( |A| \times \log |V| + (2|A|) \underbrace{\alpha(2|A|, |V|)}_{\leq 4} )$$

$$O( |A| \times \log |V| + |A| )$$

$$O( |A| \times \log |V| )$$