Capítulo VI

Tipo Abstrato de Dados Partição

TAD Partição (com n elementos)

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

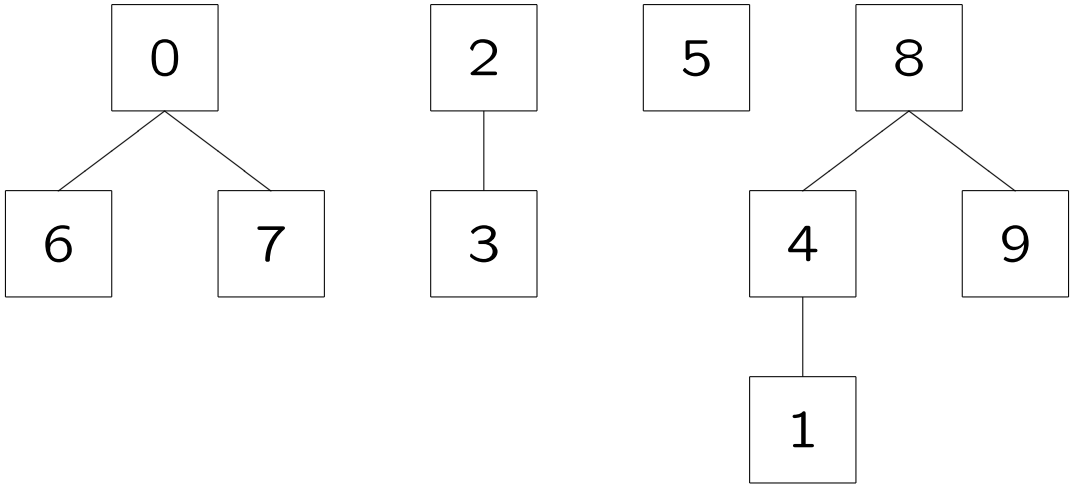
$$\text{Domínio} = \{0, 1, ..., n-1\}$$

// Cria a partição $\{\{0\}, \{1\}, \ldots, \{n-1\}\}$.
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 **união**(Domínio e, Domínio f);

Implementação em Floresta



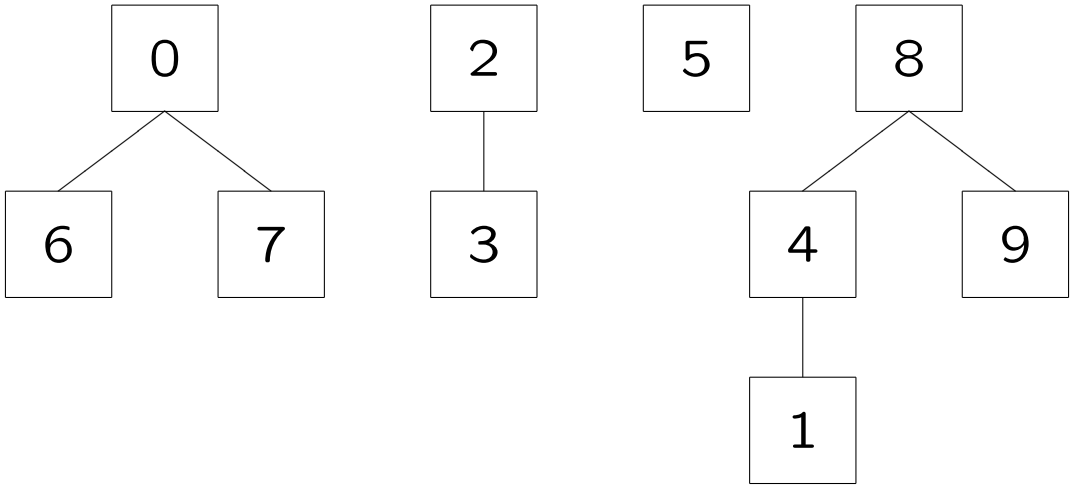
Complexidade (com n elementos)

criação	$\Theta(?)$	**representante**	$O(?)$
união	$\Theta(?)$	(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



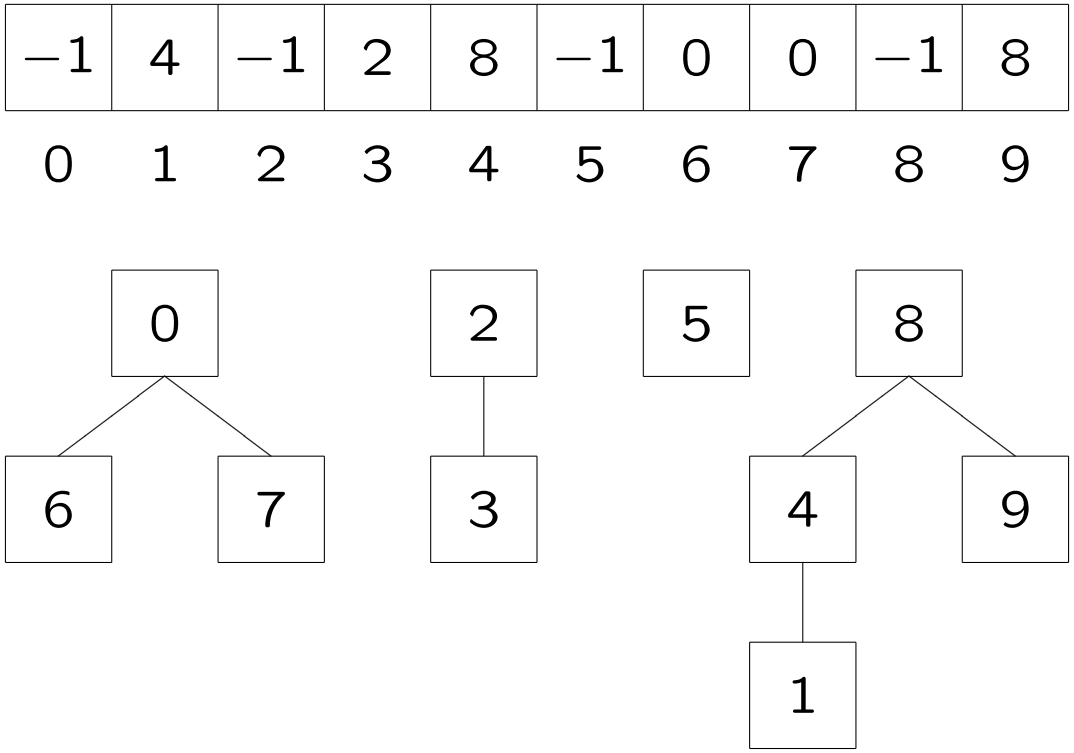
Complexidade (com n elementos)

criação $\Theta(?)$ **representante** $O(?)$

união $\Theta(?)$ (no pior caso)

Implementação em Floresta

(implementada em vetor)



Complexidade (com n elementos)

criação	$\Theta(n)$	**representante**	$O(n)$
união	$\Theta(1)$	(no pior caso)	

Complexidade do Algoritmo de Kruskal

União sem Estratégia

Representante sem Efeitos Laterais

criar heap	$\Theta(\|A\|)$
criar partição	$\Theta(\|V\|)$
criar lista ligada	$\Theta(1)$
Ciclo (executado entre $\|V\|-1$ e $\|A\|$ vezes)	
1 remover mínimo	$O(\log \|A\|)$
2 representante	$O(\|V\|)$
Ciclo (executado $\|V\|-1$ vezes)	
1 inserir à cauda	$\Theta(1)$
1 união	$\Theta(1)$
TOTAL	$O(\|A\| \times \|V\|)$

Complexidade do Algoritmo de Kruskal

União sem Estratégia

Representante sem Efeitos Laterais

Complexidade do Primeiro Ciclo

$$O(\quad |A| \times \log |A| \quad + \quad |A| \times (2\mathbf{R}) \quad)$$

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

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

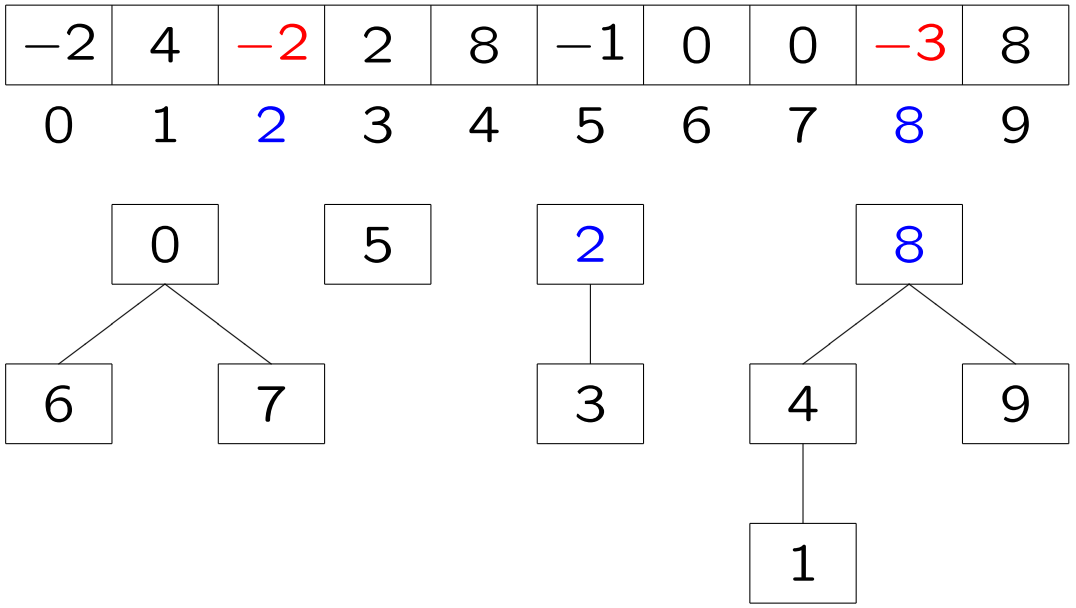
$$O(\quad |A| \times |V| \quad)$$

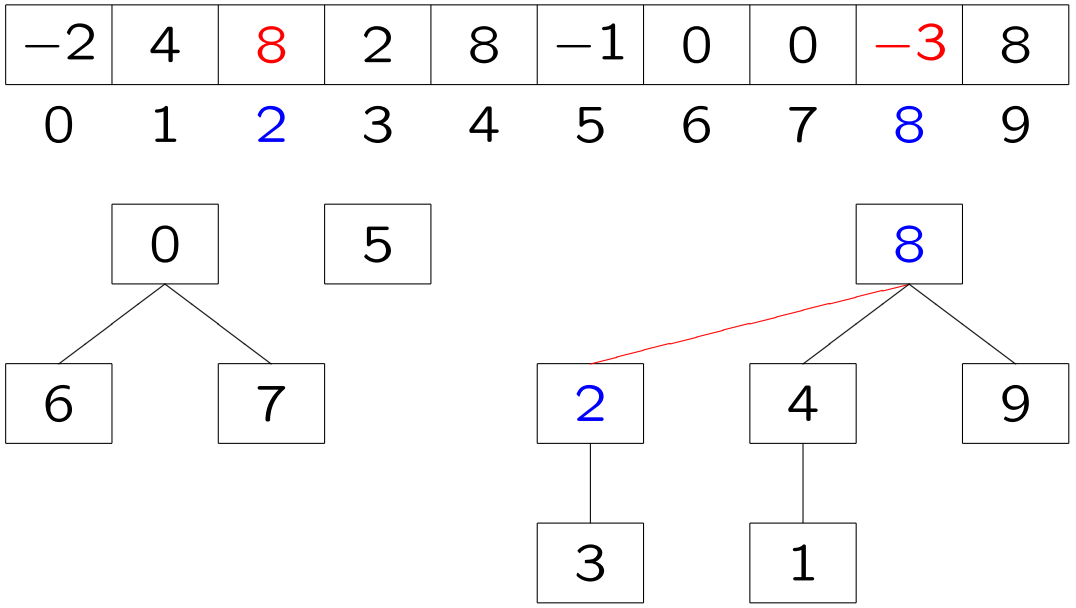
União

por

Altura

$(2, 8)$



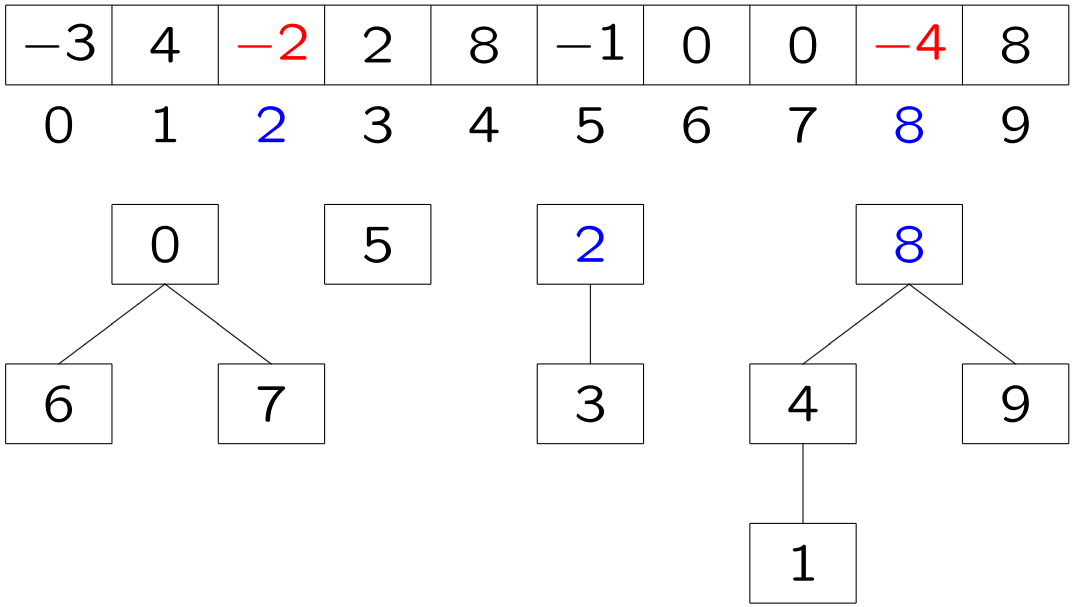


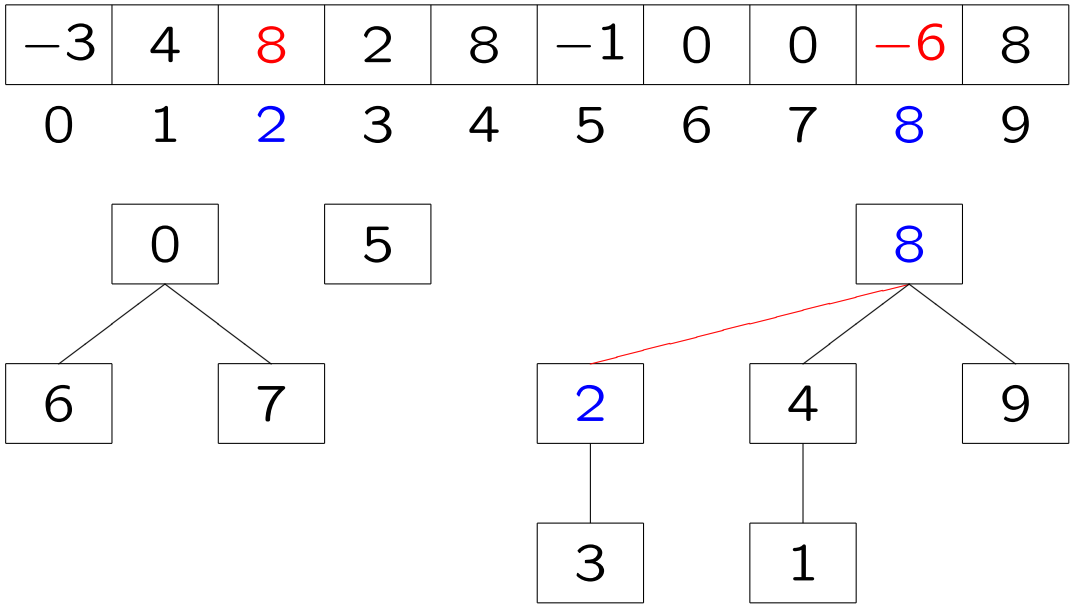
União

por

Tamanho

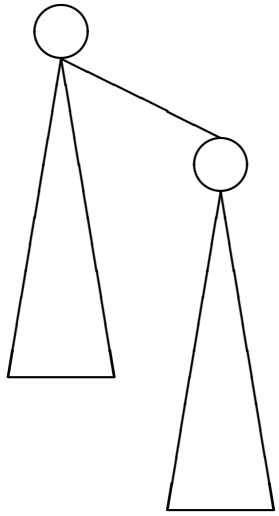
$(2, 8)$





Complexidade do Representante com União por **Altura**

Número Mínimo de Nós de uma Árvore com Altura h



$N(1) = 1$

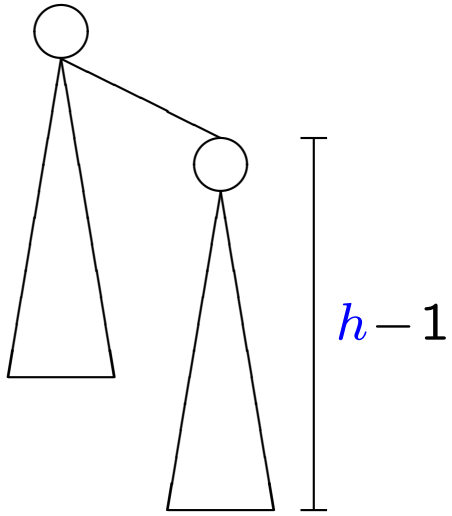
$N(2) = 2$

$N(h) =$

(para $h \geq 2$)

Complexidade do Representante com União por **Altura**

Número Mínimo de Nós de uma Árvore com Altura h



$N(1) = 1$

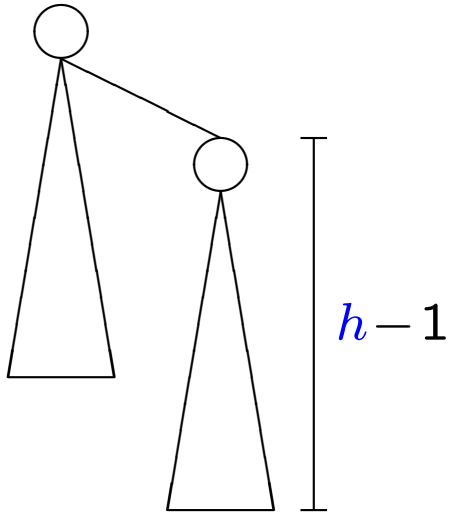
$N(2) = 2$

$N(h) =$

(para $h \geq 2$)

Complexidade do Representante com União por **Altura**

Número Mínimo de Nós de uma Árvore com Altura h



$$N(1) = 1$$

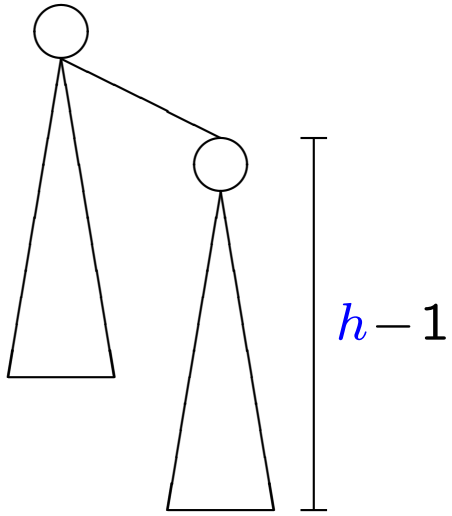
$$N(2) = 2$$

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

(para $h \geq 2$)

Complexidade do Representante com União por **Altura**

Número Mínimo de Nós de uma Árvore com Altura h



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

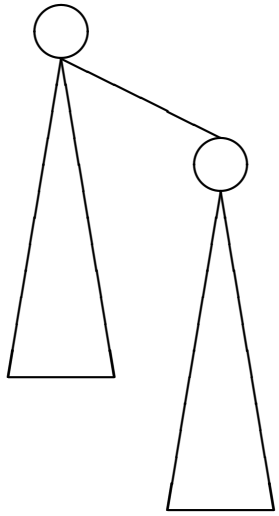
$$N(2) = 2 \qquad = 2^1$$

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

(para $h \geq 2$)

Complexidade do Representante com União por **Tamanho**

Número Mínimo de Nós de uma Árvore com Altura h



$N(1) = 1$

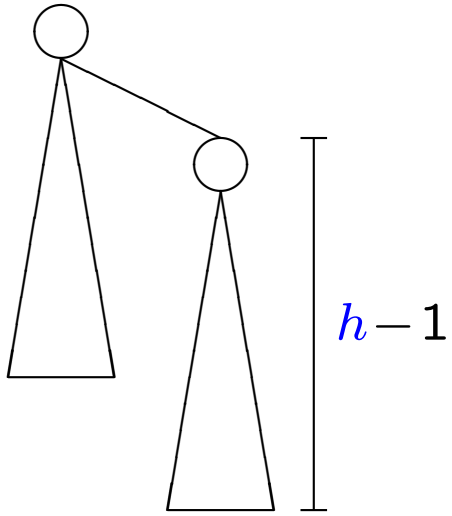
$N(2) = 2$

$N(h) =$

(para $h \geq 2$)

Complexidade do Representante com União por **Tamanho**

Número Mínimo de Nós de uma Árvore com Altura h



$N(1) = 1$

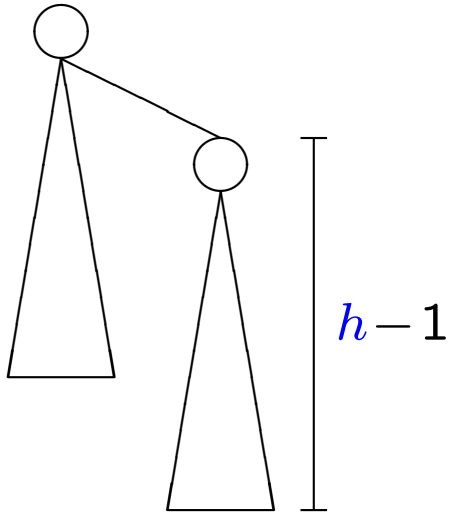
$N(2) = 2$

$N(h) =$

(para $h \geq 2$)

Complexidade do Representante com União por **Tamanho**

Número Mínimo de Nós de uma Árvore com Altura h



$$N(1) = 1$$

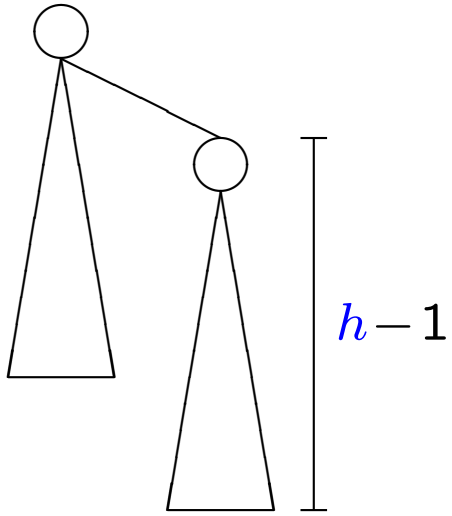
$$N(2) = 2$$

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

(para $h \geq 2$)

Complexidade do Representante com União por **Tamanho**

Número Mínimo de Nós de uma Árvore com Altura h



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

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

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

(para $h \geq 2$)

Complexidade do Representante com União por **Altura** ou por **Tamanho**

Altura Máxima de uma Árvore com n nós

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

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

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

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

$$\boxed{h \leq 1 + \log(n)}$$

Complexidade do Algoritmo de Kruskal

União por Altura ou por Tamanho

Representante sem Efeitos Laterais

criar heap	$\Theta(\|A\|)$
criar partição	$\Theta(\|V\|)$
criar lista ligada	$\Theta(1)$
Ciclo (executado entre $\|V\|-1$ e $\|A\|$ vezes)	
1 remover mínimo	$O(\log \|A\|)$
2 representante	$O(\log \|V\|)$
Ciclo (executado $\|V\|-1$ vezes)	
1 inserir à cauda	$\Theta(1)$
1 união	$\Theta(1)$
TOTAL	$O(\|A\| \times \log \|V\|)$

Complexidade do Algoritmo de Kruskal

União por Altura ou por Tamanho

Representante sem Efeitos Laterais

Complexidade do Primeiro Ciclo

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

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

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

Interface Partição (com n elementos)

```
public interface UnionFind
{
    // Creates the partition {{0},{1},...,{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 S1 and S2 whose representatives
    // are the specified elements, and inserts the set S1 ∪ S2.
    // The representative of the new set S1 ∪ S2 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 <= element < 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 <= element < partition.length.
protected int findRec( int element )
{
    if ( partition[element] < 0 )
        return element;
    else
        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;
}
```

União por Tamanho (1)

```
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");
```

União por Tamanho (2)

```
    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;
    }
}
```

União por Altura (1)

```
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");
```

União por Altura (2)

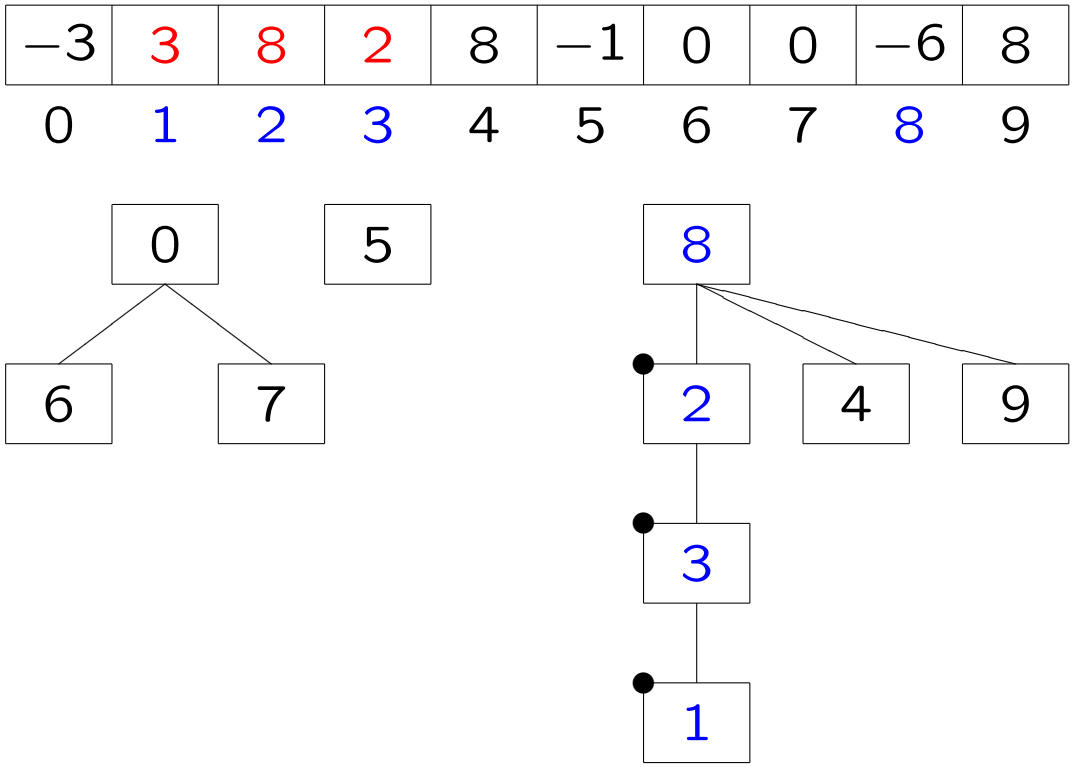
```
        if ( partition[rep1] <= partition[rep2] )
        {
            // Height(S1) >= Height(S2).
            if ( partition[rep1] == partition[rep2] )
                partition[rep1]--;
            partition[rep2] = rep1;
        }
        else
            // Height(S1) < Height(S2).
       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

}   // End of Class UnionFindInArray.
```

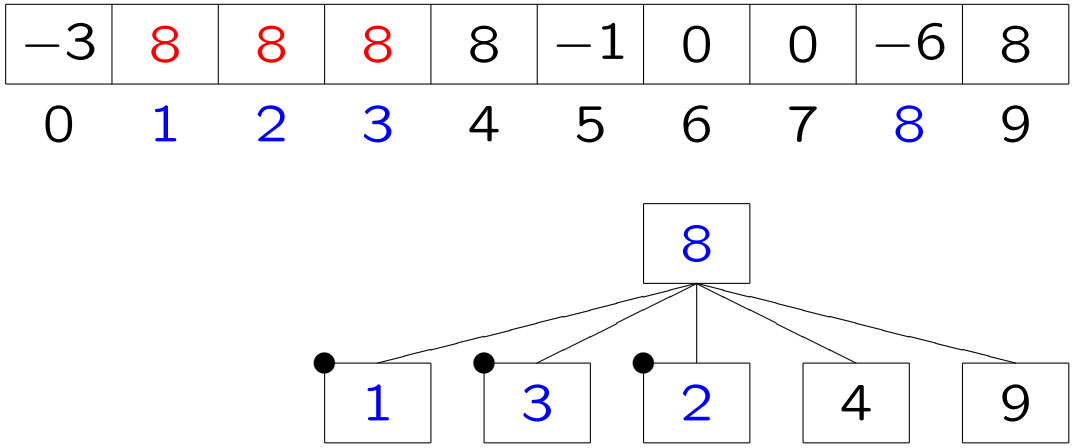
Compressão

do

Caminho

find(1)





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;
    else
    {   partition[element] = this.findPathCompr( partition[element] );
        return partition[element];
    }
}
```

Complexidade no PIOR CASO

de U operações de **união** e R operações de **representante**

(com n elementos)

União sem Estratégia	$\Theta(1)$	$O(U + R\,n)$
Representante sem Efeitos Laterais	$O(n)$	
União por Altura ou por Tamanho	$\Theta(1)$	$O(U + R \log n)$
Representante sem Efeitos Laterais	$O(\log n)$	
União por **Nível** ou por 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 $\alpha(k,n)$ $\quad (k \geq n \geq 1)$

2^2		4
2^{2^2}	2^4	16
$2^{2^{2^2}}$	2^{16}	65536
$2^{2^{2^{2^2}}\}4}$	2^{65536}	$\approx 20\,000$ algarismos

Na prática,

$$\boxed{\alpha(k,n) \leq 4}$$

porque

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

Complexidade do Algoritmo de Kruskal

União por Nível ou por Tamanho

Representante com Compressão do Caminho

criar heap	$\Theta(\|A\|)$
criar partição	$\Theta(\|V\|)$
criar lista ligada	$\Theta(1)$
Ciclo (executado entre $\|V\|-1$ e $\|A\|$ vezes)	
1 remover mínimo	$O(\log \|A\|)$
2 representante	$O(\log \|V\|)$
Ciclo (executado $\|V\|-1$ vezes)	
1 inserir à cauda	$\Theta(1)$
1 união	$\Theta(1)$
TOTAL	$O(\|A\| \times \log \|V\|)$

Complexidade do Algoritmo de Kruskal

União por Nível ou por Tamanho

Representante com Compressão do Caminho

Complexidade do Primeiro Ciclo

$$
\begin{aligned}
O(\ & |A| \times \log |A| \ + \ \underbrace{|A| \times (2\mathbf{R})}_{2|A| \geq 2(|V|-1) \geq |V|} \ ) \\
O(\ & |A| \times \log |V| \ + \ (2|A|)\, \alpha(2|A|, |V|) \ ) \\
O(\ & |A| \times \log |V| \ + \ |A| \ ) \\
O(\ & |A| \times \log |V| \ )
\end{aligned}
$$