

## Aula 14

- Algoritmo Union-Find
  - Compressão de Caminho
- Algoritmo de Prim
  - Exemplos
  - Complexidade
  - Conexão



## Algoritmo de Kruskal

Kruskal( $f, v$ )

for each  $v \in f.V$   
    MakeSet( $v$ )

let  $E' = \text{sort}(f.E, w)$

$A = \emptyset$

for each  $(u, v) \in E'$

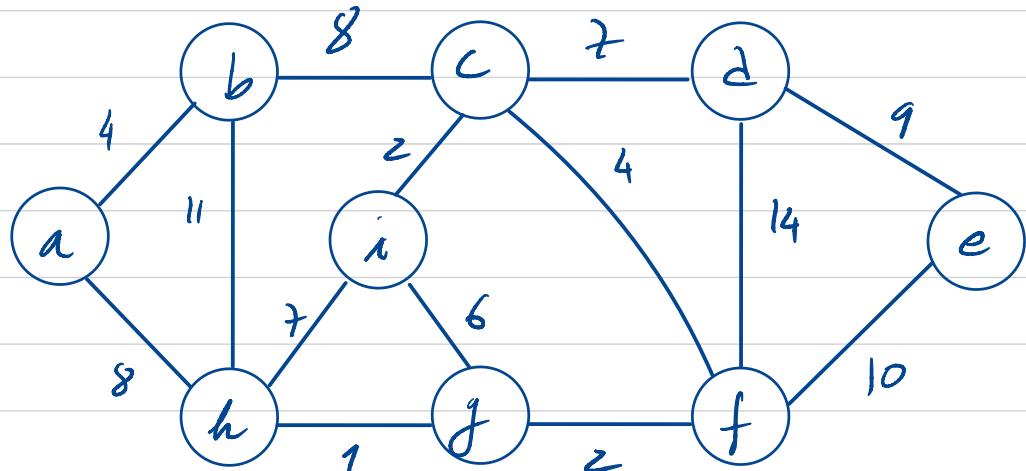
    if FindSet( $u$ )  $\neq$  FindSet( $v$ )

$A = A \cup \{(u, v)\}$

        Union( $u, v$ )

return  $A$

Exemplo:



## Algoritmo de Kruskal

Kruskal( $f, v$ )

for each  $v \in f.V$   
    MakeSet( $v$ )

let  $E' = \text{sort}(f.E, w)$

$A = \emptyset$

for each  $(u, v) \in E'$

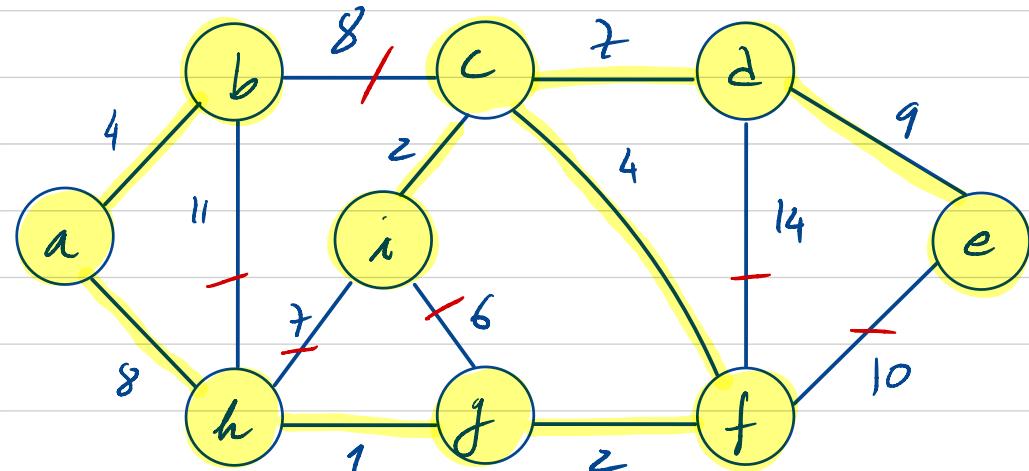
    if FindSet( $u$ )  $\neq$  FindSet( $v$ )

$A = A \cup \{(u, v)\}$

        Union( $u, v$ )

return  $A$

Exemplo:



$$w(T) = 1 + 2 + 2 + 4 + 4 + 8 + 7 + 9 \\ = 37$$

## Algoritmo de Kruskal

Kruskal( $G, V$ )

for each  $v \in G.V$

    MakeSet( $v$ )

let  $E' = \text{sort}(G.E, w)$

$A = \emptyset$

for each  $(u, v) \in E'$

    if FindSet( $u$ )  $\neq$  FindSet( $v$ )

$A = A \cup \{(u, v)\}$

        Union( $u, v$ )

return  $A$

## Algoritmo Union-Find

- Make-Set( $x$ ) - cria o conjunto  $\{x\}$

- Union( $x, y$ ) - faz a união do conjunto de  $x$  com o conjunto de  $y$

- Find-Set( $x$ ) - retorna o "representante" do conjunto que contém  $x$

- Nota: Cada conjunto tem um representante que o identifica univocamente.

## Algoritmo Union-Find - Representação de Conjuntos Disjuntos

Ideia: Conjuntos são representados como árvores n-áreas.

Exemplos:

- Conjunto  $\{B, E\}$

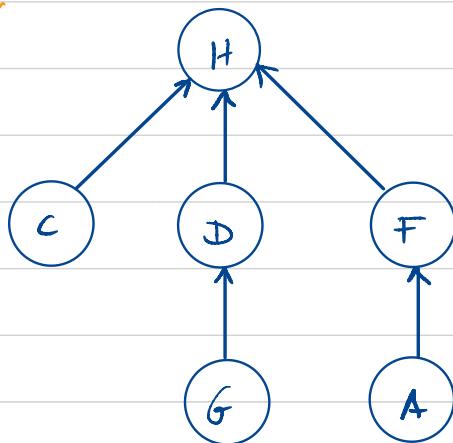


obs: o filho aponta para o pai

Representação Interna: Cada nó tem:

- um pai
- um rank: uma estimativa da "altura" do nó na árvore

- Conjunto  $\{A, C, D, F, G, H\}$

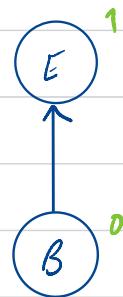


# Algoritmo Union-Find - Representação de Conjuntos Disjuntos

Ideia: Conjuntos são representados como árvores n-árias.

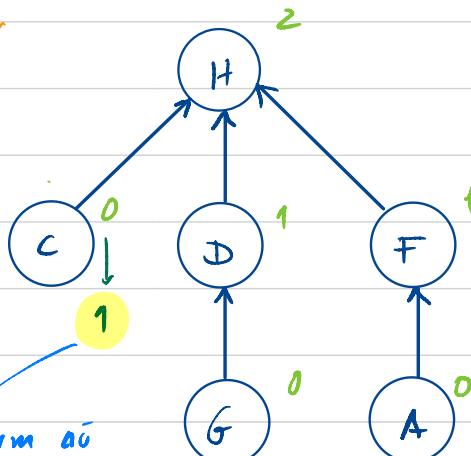
Exemplos:

- Conjunto  $\{B, E\}$



Obs: o filho aponta para o pai

- Conjunto  $\{A, C, D, F, G, H\}$



o rank de um nó  
pode ser superior à  
sua altura efectiva.

Observações:

- Cada conjunto corresponde a uma árvore n-ária
- O representante de um dado conjunto corresponde à raiz da árvore resp.
- Cada nó tem um pai e um rank
- O rank de um nó é uma estimativa de "altura" do nó na árvore que o contém.

## Algoritmo Union - Find

Make-Set( $x$ )  $\Rightarrow$  cria o conjunto com o elemento  $x$

MakeSet( $x$ )

$x.p := x$

$x.rank := 0$

Complexidade:  $O(1)$

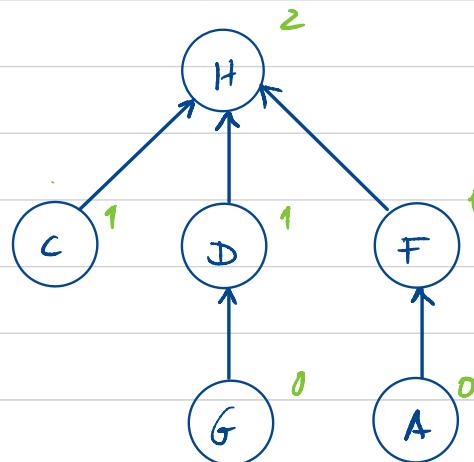
Find Set( $x$ )  $\Rightarrow$  retorna o representante do conjunto que contém  $x$

Find Set( $x$ )

while ( $x \neq x.p$ )

$x := x.p$

return  $x$



• FindSet(A) = ?

• FindSet(G) = ?

## Algoritmo Union - Find

Union ( $x, y$ )  $\Rightarrow$  faz a união dos conjuntos que contém  $x$  e  $y$

• Union ( $B, A$ ) ?

Union ( $x, y$ )

let  $R_x = \text{FindSet}(x)$

let  $R_y = \text{FindSet}(y)$

if ( $R_x == R_y$ ) return

if ( $R_y.\text{rank} > R_x.\text{rank}$ )

$R_y.p := R_x$

else if ( $R_y.\text{rank} > R_x.\text{rank}$ )

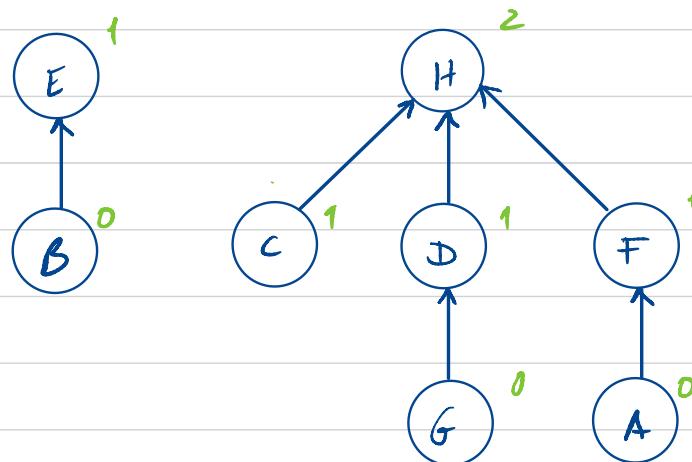
$R_x.p := R_y$

else

$R_y.p := R_x$

$R_x.\text{rank} := R_x.\text{rank} + 1$

$\left. \begin{array}{l} R_x.\text{rank} = R_y.\text{rank} \end{array} \right\}$



## Algoritmo Union - Find

Union ( $x, y$ )  $\Rightarrow$  faz a união dos conjuntos que contém  $x$  e  $y$

• Union (B, A) ?

Union ( $x, y$ )

let  $R_x = \text{FindSet}(x)$

let  $R_y = \text{FindSet}(y)$

if ( $R_x == R_y$ ) return

if ( $R_y.\text{rank} > R_x.\text{rank}$ )

$R_y.p := R_x$

else if ( $R_y.\text{rank} > R_x.\text{rank}$ )

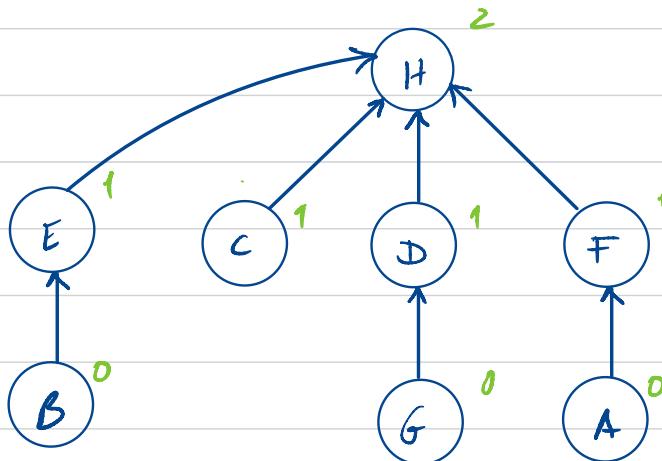
$R_x.p := R_y$

else

$R_y.p := R_x$

$R_x.\text{rank} := R_x.\text{rank} + 1$

$$\left. \begin{array}{l} R_x.\text{rank} = R_y.\text{rank} \\ \end{array} \right\}$$



## Algoritmo Union-Find

Union( $x, y$ )  $\Rightarrow$  faz a união dos conjuntos que contém  $x$  e  $y$

Union( $x, y$ )

let  $R_x = \text{FindSet}(x)$

let  $R_y = \text{FindSet}(y)$

if ( $R_x == R_y$ ) return

if ( $R_y.\text{rank} > R_x.\text{rank}$ )

$R_y.p := R_x$

else if ( $R_y.\text{rank} > R_x.\text{rank}$ )

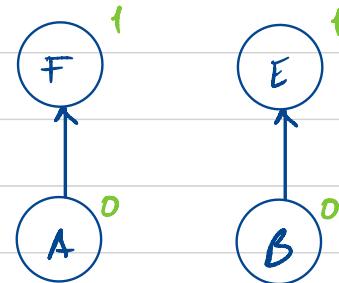
$R_x.p := R_y$

else

$R_y.p := R_x$

$R_x.\text{rank} := R_x.\text{rank} + 1$

• Union( $A, E$ )



## Algoritmo Union-Find

Union( $x, y$ )  $\Rightarrow$  faz a união dos conjuntos que contém  $x$  e  $y$

Union( $x, y$ )

let  $R_x = \text{FindSet}(x)$

let  $R_y = \text{FindSet}(y)$

if ( $R_x == R_y$ ) return

if ( $R_y.\text{rank} > R_x.\text{rank}$ )

$R_y.p := R_x$

else if ( $R_y.\text{rank} > R_x.\text{rank}$ )

$R_x.p := R_y$

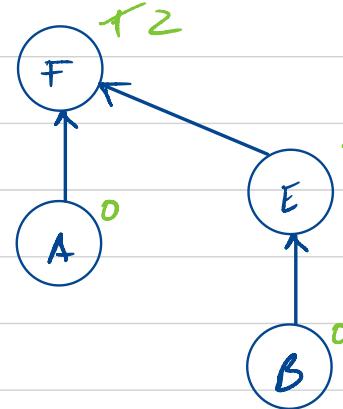
else

$R_y.p := R_x$

$R_x.\text{rank} := R_x.\text{rank} + 1$

$$\left. \begin{array}{l} R_x.\text{rank} = R_y.\text{rank} \\ \end{array} \right\}$$

• Union( $A, E$ )



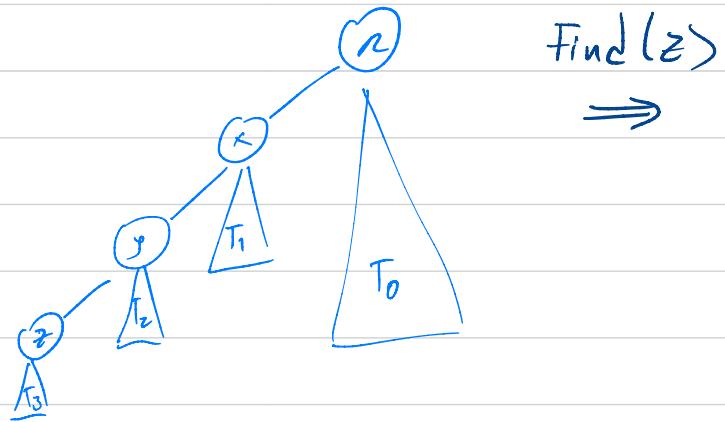
## Algoritmo Union-Find: Compressão de Caminho

FindSet( $x$ ): retorna o representante do conjunto que contém  $x$

⇒ A árvore que contém  $x$  é achatada durante a operação de FindSet

FindSet( $x$ )

```
if  $x \neq x.p$ 
     $x.p := \text{FindSet}(x.p)$ 
return  $x.p$ 
```



- Operações de Find geram árvores mais achatadas.

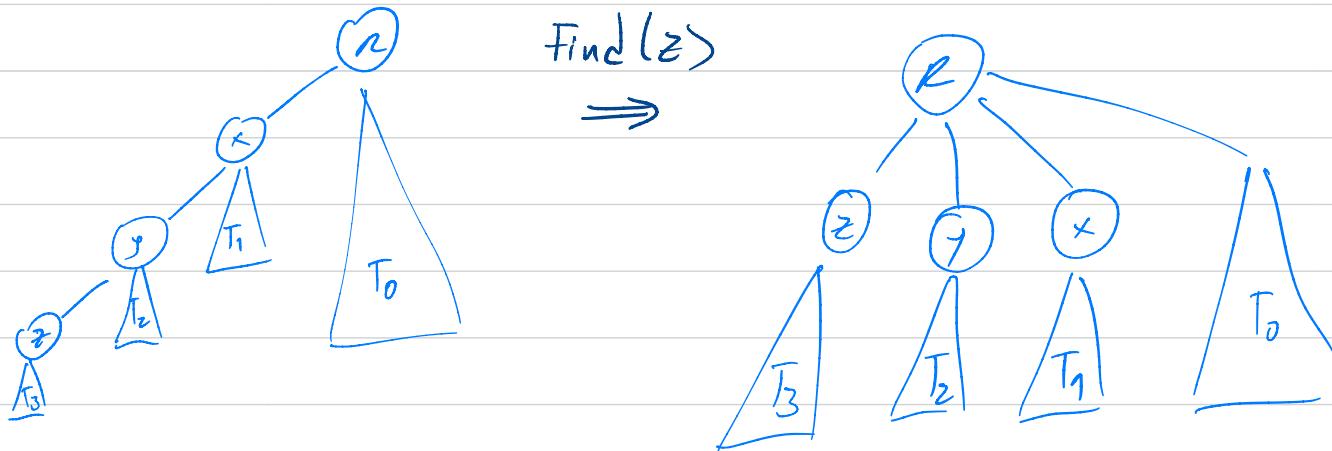
## Algoritmo Union-Find: Compressão de Caminho

FindSet( $x$ ): retorna o representante do conjunto que contém  $x$

⇒ A árvore que contém  $x$  é achatada durante a operação de FindSet

FindSet( $x$ )

```
if  $x \neq x.p$ 
     $x.p := \text{FindSet}(x.p)$ 
return  $x.p$ 
```



- Operações de Find geram árvores mais achatadas.

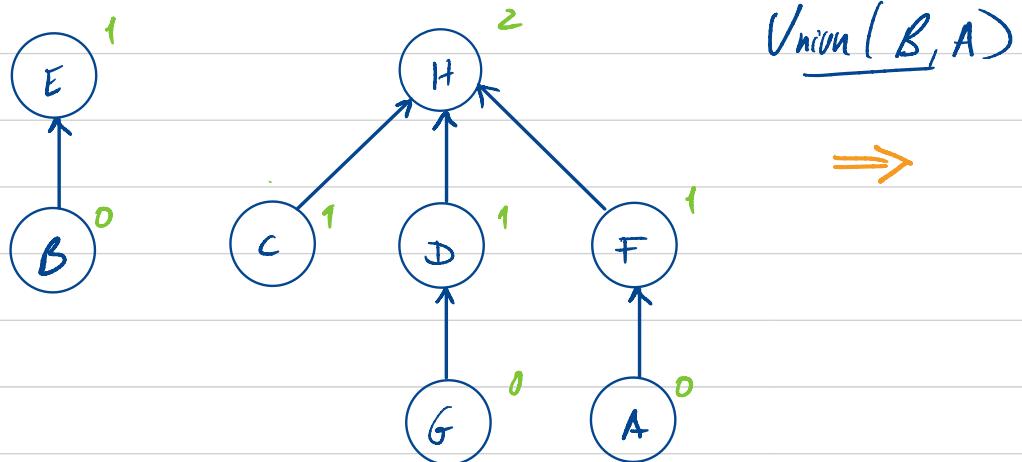
## Algoritmo Union-Find: Compressão de Caminho

FindSet( $x$ ): retorna o representante do conjunto que contém  $x$

$\Rightarrow$  A árvore que contém  $x$  é achatada durante a operação de FindSet

FindSet( $x$ )

```
if  $x \neq x.p$ 
     $x.p := \text{FindSet}(x.p)$ 
return  $x.p$ 
```



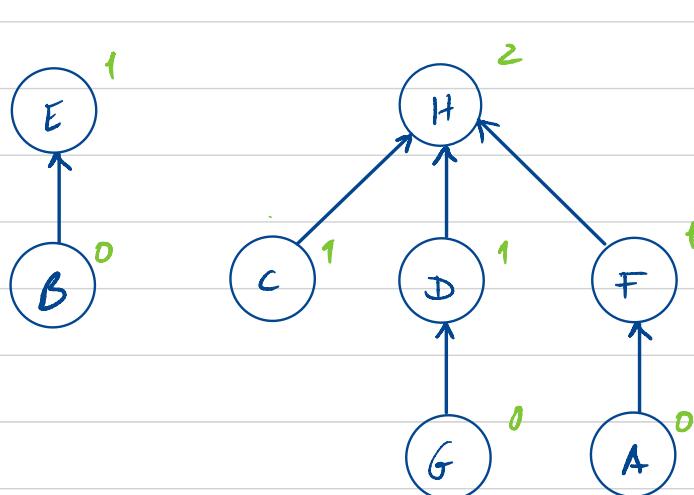
## Algoritmo Union-Find: Compressão de Caminho

FindSet( $x$ ): retorna o representante do conjunto que contém  $x$

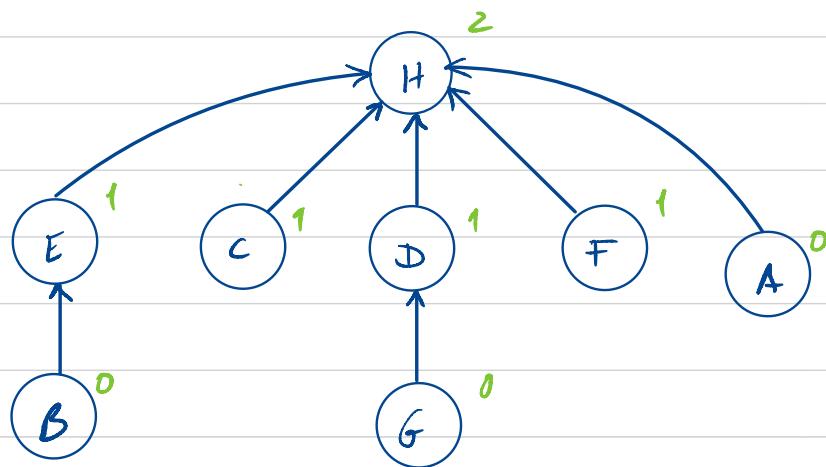
⇒ A árvore que contém  $x$  é achatada durante a operação de FindSet

FindSet( $x$ )

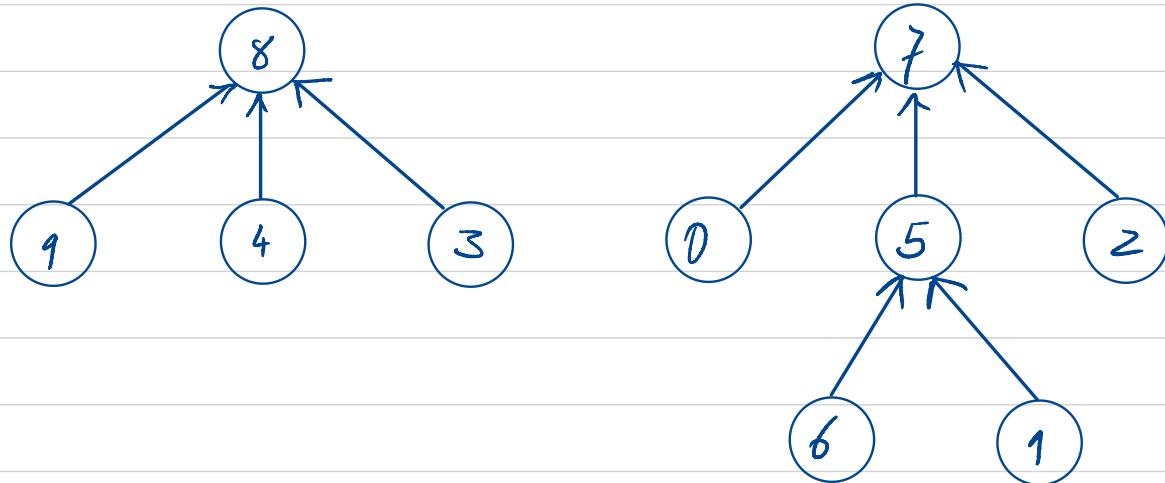
```
if  $x \neq x.p$ 
     $x.p := \text{FindSet}(x.p)$ 
return  $x.p$ 
```



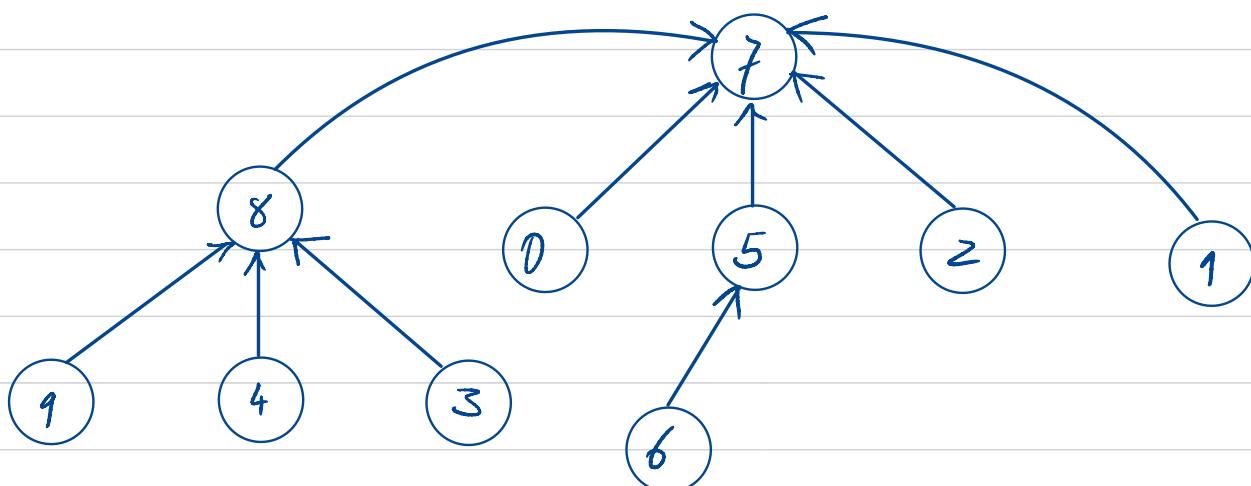
Union( $B, A$ )



## Algoritmo Union-Find : Compressão de Caminho



Union (1,3) :



## Algoritmo Union Find - Análise Teórica

Objectivo: Provar que a complexidade de  $\text{FindSet}(n)$  e  $\text{Union}(n, j)$  é  $O(\lg n)$ , onde  $n$  é o nº de elementos considerados.

Propriedade [Nós raiz]

Quando um nó deixa de ser raiz

Propriedade [Variação do Rank-1]

O rank de um nó x só pode

Propriedade [Variação do Rank-z]

Só os ranks é que podem aumentar

Propriedade [Variação do Rank-3]

Os ranks ao longo dos caminhos a ligar nós folha a nós raiz.

## Algoritmo Union Find - Análise Teórica

Objectivo: Provar que a complexidade de  $\text{FindSet}(n)$  e  $\text{Union}(n, j)$  é  $O(\lg n)$ , onde  $n$  é o nº de elementos considerados.

### Propriedade [Nós raiz]

Quando um nó deixa de ser raiz nunca poderá voltar a sê-lo.

### Propriedade [Variação do Rank-1]

O rank de um nó x só pode crescer com o tempo.

### Propriedade [Variação do Rank-z]

Só os ranks de nós raiz é que podem aumentar

↳ Qd um nó deixa de ser raiz, o seu rank não é mais alterado.

### Propriedade [Variação do Rank-3]

Os ranks crescem estreitamente ao longo dos caminhos a ligar nós folha a nós raiz.

## Algoritmo Union Find - Análise Teórica

Lema dos Ranks

O nº de nós com rank  $R$  é no máximo  $n/z^R$

Corolário Altura máxima de um nó é:  $\log_2 n$

Prova

- Os todos os elementos fazem adicionados ao mesmo conjunto, teremos apenas um único elemento com rank máximo.

$$n/z^R = 1 \Leftrightarrow z^R = n \Leftrightarrow R = \log_2 n$$

## Algoritmo Union Find - Análise Teórica

### Lema dos Ranks

O nº de nós com rank  $R$  é no máximo  $n/z^R$

Corolário Altura máxima de um nó é:  $\log_2 n$

### Prova

- Os todos os elementos fazem adicionados ao mesmo conjunto, teremos apenas um único elemento com rank máximo.

$$n/z^R = 1 \Leftrightarrow z^R = n \Leftrightarrow R = \log_2 n$$

# Algoritmo Union Find - Análise Teórica

## Lema dos Ranks

O nº de nós com rank  $R$  é no máximo  $n/z^R$

## Lema Auxiliar

Um nó com rank  $R$  é raiz de uma árvore com pelo menos  $z^R$  elementos.

### Prova

- A prova faz-se por indução no nº de uniões,  $n$ .

$n=0$  No inicio todos os elementos são raízes de árvores com rank 0.

Cada nó tem exatamente 1 elemento ( $1 = z^0$ ). ✓

$n > 0$  Queremos provar que a proposição é verdadeira depois de  $\text{Union}(x, y)$ .

Há 2 casos a considerar:

$$\textcircled{I} \quad R_x = R_y$$

$$\textcircled{II} \quad R_x > R_y \text{ ou } R_y > R_x$$

# Algoritmo Union Find - Análise Teórica

## Lema dos Ranks

O nº de nós com rank  $R$  é no máximo  $n/z^R$

## Lema Avançado

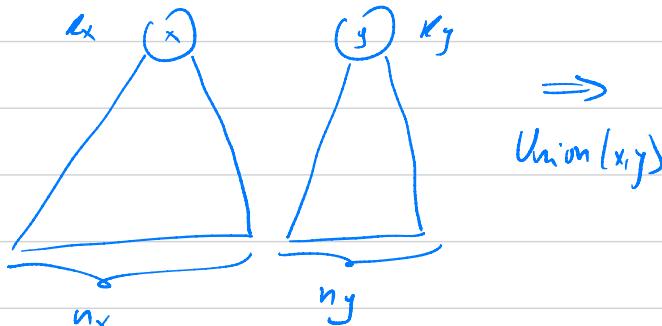
Um nó com rank  $R$  é raiz de uma árvore com pelo menos  $z^R$  elementos.

### Prova

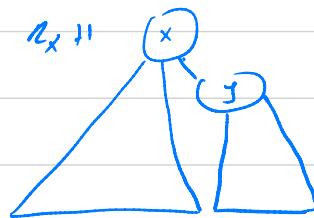
①  $r_x = r_y$

$$r'_x = r_x + 1$$

→ Queremos provar que o nº de elementos da nova árvore é  $\geq z^{r_x+1}$



$\Rightarrow$   
Union( $x, y$ )



• Da HJ:  $n_x \geq z^{r_x}$  e  $n_y \geq z^{r_y} = z^{r_x}$

$$\begin{aligned} n_x + n_y &\geq z^{r_x} + z^{r_y} \\ &= z \cdot z^{r_x} \\ &= z^{r_x+1} \quad \checkmark \end{aligned}$$

# Algoritmo Union Find - Análise Teórica

## Lema dos Ranks

O nº de nós com rank  $R$  é no máximo  $n/z^R$

## Lema Avançado

Um nó com rank  $R$  é raiz de uma árvore com pelo menos  $z^R$  elementos.

### Prova

II)  $r_y < r_x$  ( $r_x < r_y$  é simétrico)



$$\begin{aligned} n_x + n_y &\geq z^{r_x} \\ n_x &\geq z^{r_x} \end{aligned}$$

$\underbrace{\quad}_{\text{IIH}} \quad \checkmark$

## Algoritmo Union-Find: Complexidade

- $m$  operações Union-Find numa estrutura com  $n$  nós tem complexidade:

- $O(m \cdot \log n)$   $\Rightarrow$  Sem compressão de caminho

- $O(m \cdot \alpha(n))$   $\Rightarrow$  Com compressão de caminho

↳ Inversa da função de Ackermann

## Algoritmo de Prim

Prim( $G, w, r$ )

for each  $v \in G.V$

$v.key = \infty; \pi.v = \text{nil}$

$r.key := 0;$

let  $Q$  be a min-priority queue with content  $G.V$

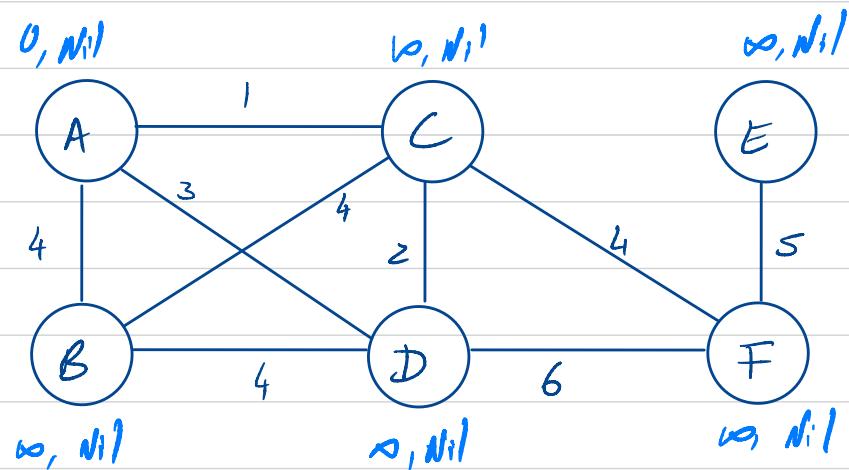
while  $Q \neq \emptyset$

let  $m = \text{ExtractMin}(Q)$

for each  $v \in G.Adj[m]$

if ( $v.key > w(m, v)$ )  $\& v \in Q$

$v.key := w(m, v); \pi.v := m$



## Algoritmo de Prim

Prim( $G, w, r$ )

for each  $v \in G.V$

$v.key = \infty; \pi.v = \text{nil}$

$r.key := 0;$

let  $Q$  be a min-priority queue with content  $G.V$

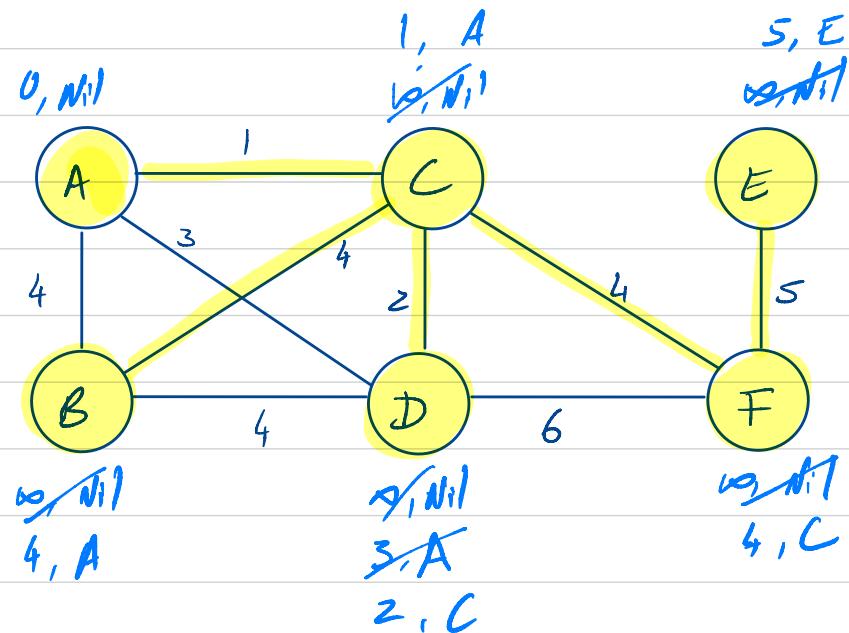
while  $Q \neq \emptyset$

let  $m = \text{ExtractMin}(Q)$

for each  $v \in G.Adj[m]$

if ( $v.key > w(m, v)$ )  $\& v \in Q$

$v.key := w(m, v); \pi.v := m$



## Algoritmo de Prim

Prim( $G, w, r$ )

for each  $v \in G.V$

$v.key = \infty; v.\pi = \text{Nil}$

$r.key := 0;$

let  $Q$  be a min-priority queue with content  $G.V$

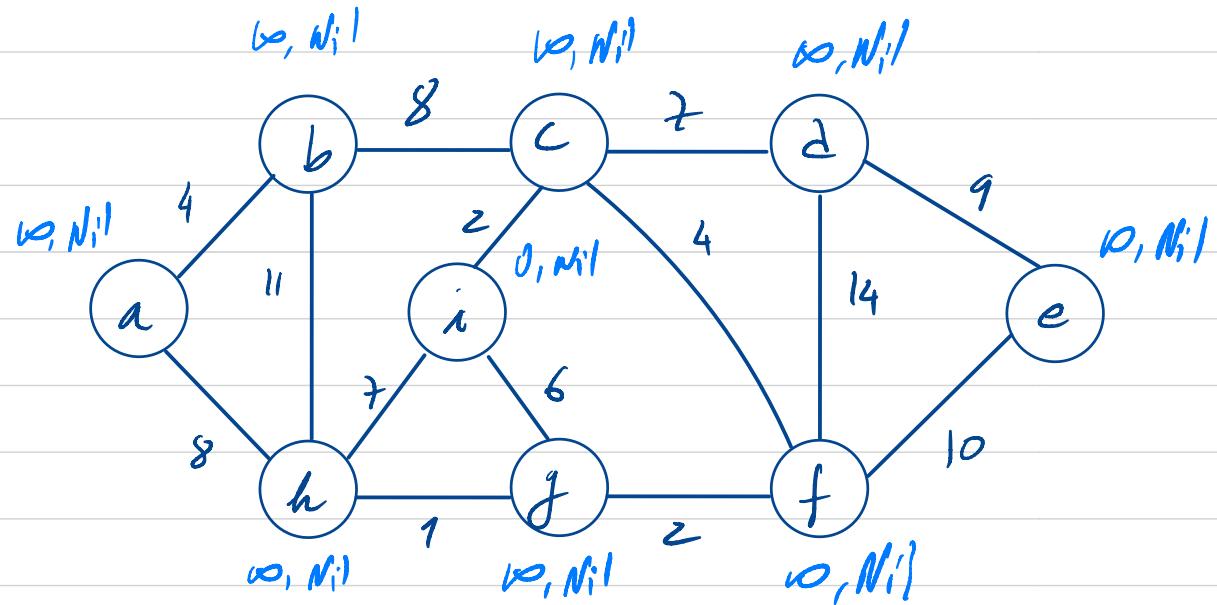
while  $Q \neq \emptyset$

let  $m = \text{ExtractMin}(Q)$

for each  $v \in G.Adj[m]$

if ( $v.key > w(m, v)$ )  $\& v \in Q$

$v.key := w(m, v); v.\pi := m$



## Algoritmo de Prim

Prim( $G, w, r$ )

for each  $v \in G.V$

$v.key = \infty; v.\pi = \text{nil}$

$r.key := 0;$

let  $Q$  be a min-priority queue with content  $G.V$

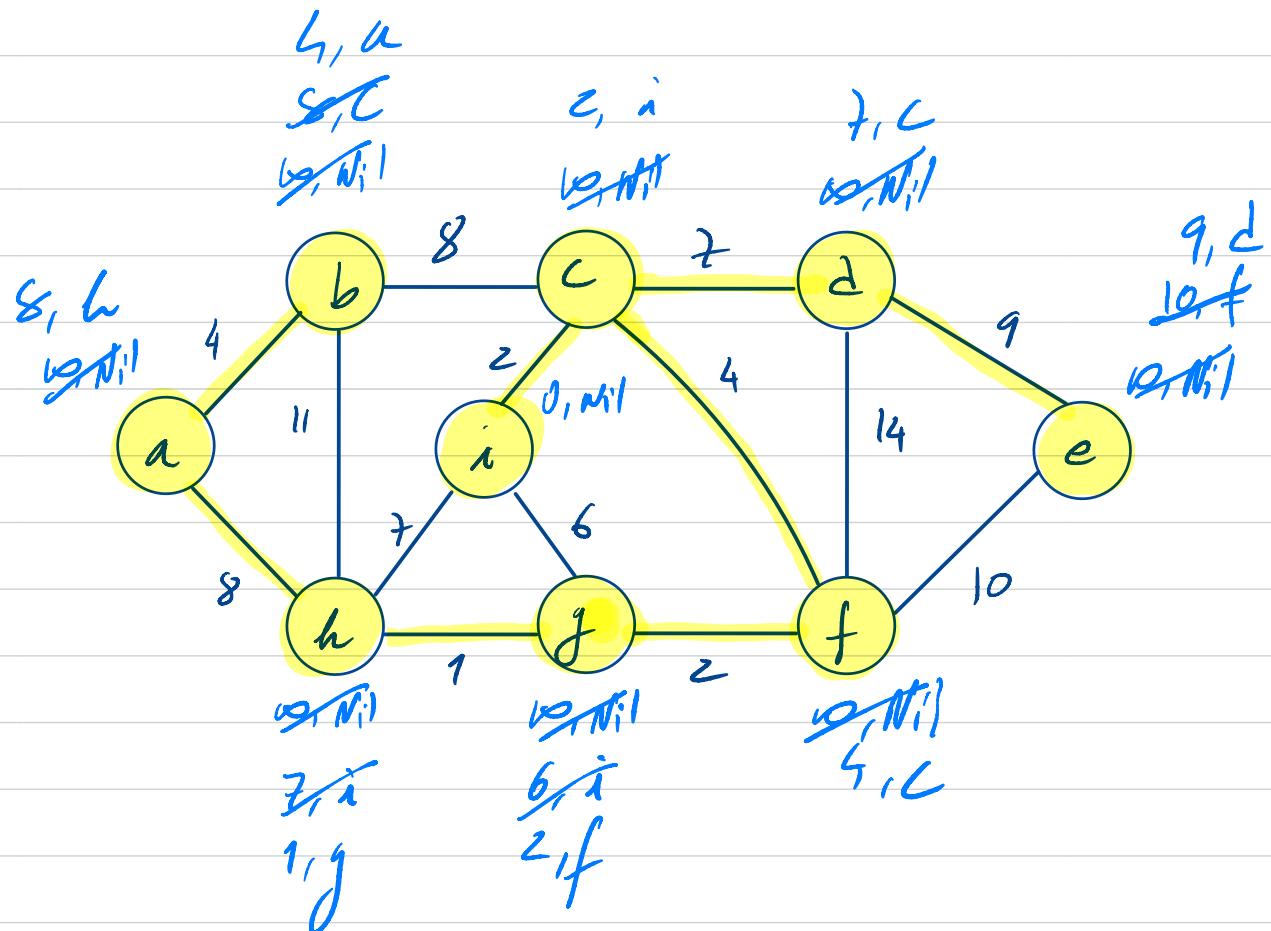
while  $Q \neq \emptyset$

let  $m = \text{ExtractMin}(Q)$

for each  $v \in G.Adj[m]$

if ( $v.key > w(m, v)$ )  $\& v \in Q$

$v.key := w(m, v); v.\pi := m$



## Algoritmo de Prim

Prim( $G, w, r$ )

for each  $v \in G.V$

$v.key = \infty; v.\pi = \text{nil}$

$r.key := 0;$

let  $Q$  be a min-priority queue with content  $G.V$

while  $Q \neq \emptyset$

let  $m = \text{ExtractMin}(Q)$

for each  $v \in G.Adj[m]$

if ( $v.key > w(m, v)$ )  $\&\& v \in Q$

$v.key := w(m, v); v.\pi := m$

## Análise de Complexidade

## Algoritmo de Prim

Prim( $G, w, r$ )

for each  $v \in G.V$

$v.key = \infty; v.\pi = \text{nil}$

)  $O(|V|)$

$r.key := 0;$

let  $Q$  be a min-priority queue with content  $G.V$   $O(|V|)$

while  $Q \neq \emptyset$

→ nº de iterações  $O(|V|)$

let  $m = \text{ExtractMin}(Q)$

for each  $v \in G.\text{Adj}[m]$  → nº TOTAL de iterações:  $O(|E|)$

if ( $v.key > w(m, v)$ )  $\&$   $v \in Q$

$v.key := w(m, v); v.\pi := m$

↪  $O(\lg |V|)$

Análise de Complexidade

(custo total do circ for:  $O(|E| \cdot \lg |V|)$ )

Custo total:  $O(|E| \cdot \lg |V|)$

## Algoritmo de Prim

Prim( $G, w, r$ )

for each  $v \in G.V$

$v.key = \infty; v.\pi = \text{nil}$

$r.key := 0;$

$A := \emptyset$

let  $Q$  be a min-priority queue with content  $G.V$

while  $Q \neq \emptyset$

let  $m = \text{ExtractMin}(Q)$

if ( $m \notin Q$ )  $A := A \cup \{(m, \pi, m)\}$

for each  $v \in G.\text{Adj}[m]$

if ( $v.key > w(m, v)$ )  $\& v \in Q$

$v.key := w(m, v); v.\pi := m$

## Análise da Convergência

(I1)  $A = \{(v, \pi, v) \mid v \notin Q \& v.\pi \neq \text{nil}\}$   
é um subconjunto de uma MST

(I2)  $\forall v \in Q$ .  
 $m.key = \min \{w(m, v) \mid v \in V \setminus Q\}$

(I3)  $\forall v \in V$ .  
 $v.\pi \neq \text{nil} \Rightarrow w(\pi, \pi, v) = v.key$

## Invariante do Algoritmo de Prim

(I1)  $A = \{(v, \pi, v) \mid v \notin Q \wedge v, \pi \neq \text{Nil}\}$   
é um subconjunto de uma MST

(I2)  $\forall u \in Q$ .  
 $u.\text{key} = \min \{ w(u, v) \mid v \in V \setminus Q \}$

(I3)  $\forall v \in V$ .  
 $v, \pi \neq \text{Nil} \Rightarrow w(v, \pi, v) = v.\text{key}$

## Inicialização (fim da primeira iteração)

(I1)  $A = \emptyset$  é subconjunto de uma MST ✓

(I2)  $V \setminus Q = \{R\}$

$\forall v \in N(R) \cdot v.\text{key} = w(v, R)$   
 $\forall v \notin N(R) \cdot v.\text{key} = \infty$  ✓

(I3)

$v, \pi \neq \text{Nil} \Leftrightarrow v, \pi = R$   
 $\Leftrightarrow v.\text{key} = w(R, v)$  ✓

# Invariante do Algoritmo de Prim

Maintém-se

$$(I_1) \quad A = \{ (\pi, \tau, v) \mid v \notin Q \wedge \pi.\tau \neq \text{Nil} \}$$

é um subconjunto de uma MST

$$(I_1) \quad A' = A \cup \{ (m, \tau, m) \}$$

$$(I_2) \quad \forall v \in Q.$$

$$m.\text{key} = \min \{ w(m, v) \mid v \in V \setminus Q \}$$

$$(I_3) \quad \forall \tau \in V.$$

$$\pi.\tau \neq \text{Nil} \Rightarrow w(\pi, \tau) = \tau.\text{key}$$

- Há que provar que  $(m, \tau, m)$  é segundo para  $A$

- Temos de encontrar um corte  $(S, V \setminus S)$  que respeite  $A$  e para o qual  $(m, \tau, m)$  seja ponte.

- $(V \setminus Q, Q)$  respeita  $A$

