- Árvores Abrangentes de Menor Custo
  - Algoritmo de Kruskal
    - Complexidade & Correcção
- Algoritmo Union-Find
  - Heurística de Compressão de Caminho

Aula 14

Árvores Abrangentes de Menor Custo - Algoritmo Genérico

MST(G)

$A \leftarrow \emptyset$

while (A does not contain all vertices)

  pick $(u,v) \in E$ s.t. (u,v) is safe for A

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

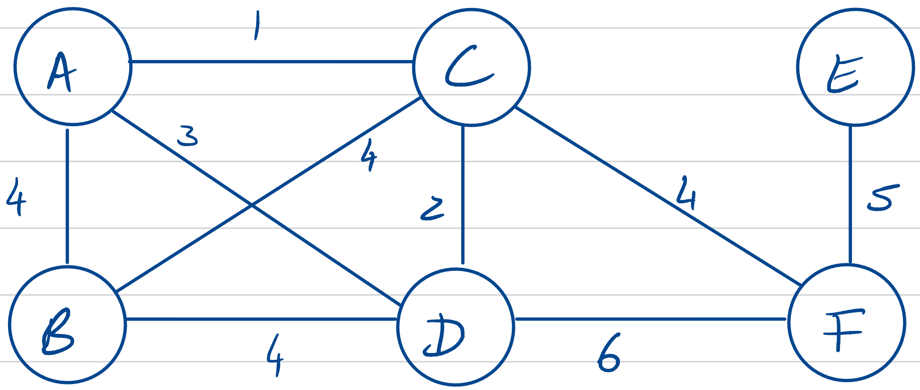
return A

Problema: Como identificar arcos seguros?

Algoritmo de Kruskal

- Associar cada nó ao conjunto de vértices do seu componente na floresta $G_A = (V, A)$

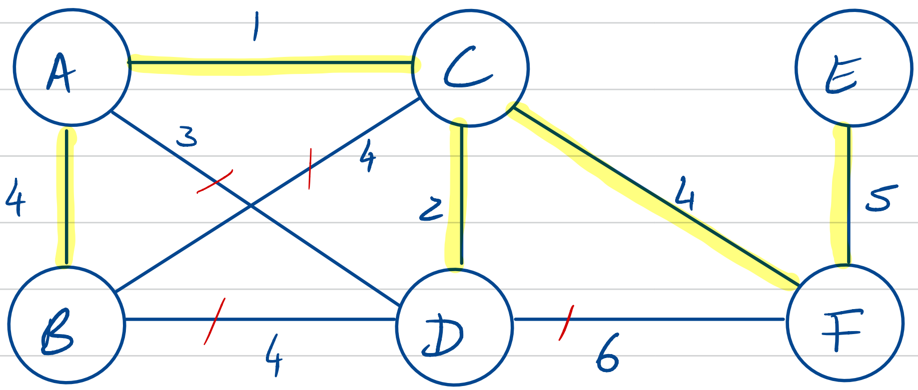
- Percorrer os arcos por ordem crescente de peso e verificar para cada arco se liga dois componentes de $G_A = (V, A)$



Algoritmo de Kruskal

- Associar cada nó ao conjunto de vértices do seu componente na floresta $G_A = (V, A)$

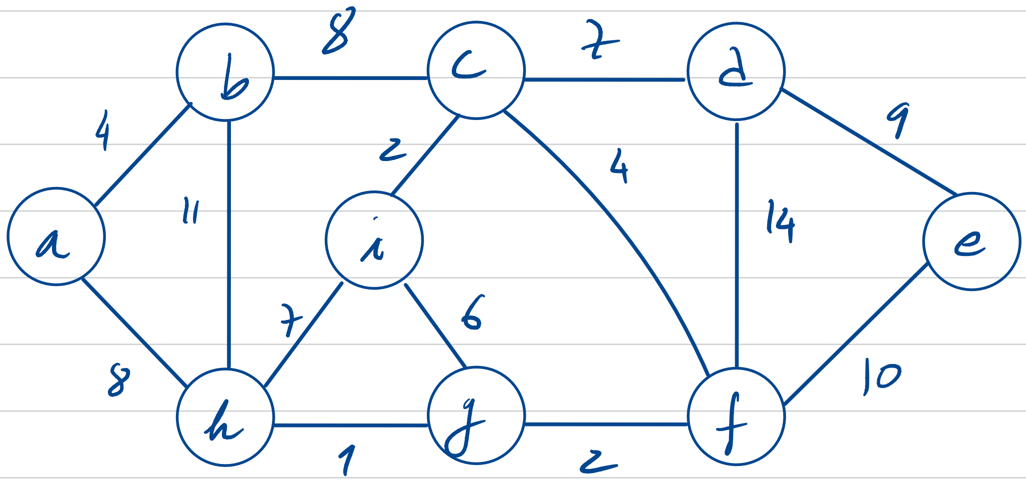
- Percorrer os arcos por ordem crescente de peso e verificar para cada arco se liga dois componentes de $G_A = (V, A)$



Algoritmo de Kruskal

```
Kruskal(G, w)
  for each v ∈ G.V
      MakeSet(v)
  let E' = sort(G.E, w)
  A = ∅
  for each (u,v) ∈ E'
      if FindSet(u) ≠ FindSet(v)
          A = A ∪ {(u,v)}
          Union(u,v)
  return A
```

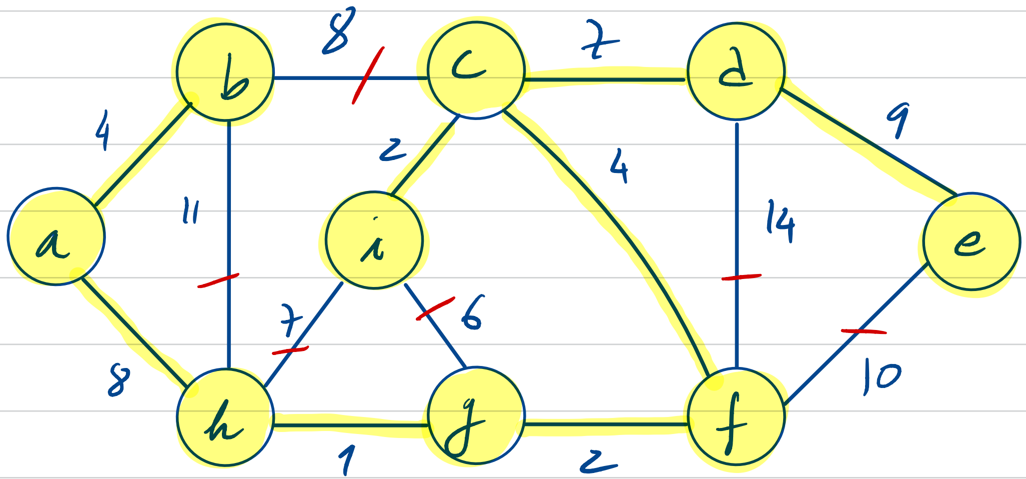
Exemplo:



Algoritmo de Kruskal

```
Kruskal(G, w)
  for each v ∈ G.V
      MakeSet(v)
  let E' = sort(G.E, w)
  A = ∅
  for each (u,v) ∈ E'
    if FindSet(u) ≠ FindSet(v)
       A = A ∪ {(u,v)}
       Union(u,v)
  return A
```

Exemplo:



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

Algoritmo de Kruskal

```
Kruskal(G, w)
  for each v ∈ G.V
      MakeSet(v)
  let E' = sort(G.E, w)
  A = ∅
  for each (u,v) ∈ E'
      if FindSet(u) ≠ FindSet(v)
          A = A ∪ {(u,v)}
          Union(u,v)
  return A
```

Algoritmo Union-Find

- Make-Set(x) - cria o conjunto $\{x\}$
  Complexidade: $O(1)$
- Union(x, y) - faz a união do conjunto de x com o conjunto de y
  Complexidade: $O(\log n)$
- FindSet(x) - retorna o "representante" do conjunto que contém x
  Complexidade: $O(\log n)$
- Nota: Cada conjunto em um representante que o identifica univocamente.

Algoritmo de Kruskal

Análise de Complexidade

```
Kruskal(G, w)
  for each v ∈ G.V
     Makeset(v)
  let E' = sort(G.E, w)
  A = ∅
  for each (u,v) ∈ E'
     if FindSet(u) ≠ FindSet(v)
        A = A ∪ {(u,v)}
        Union(u,v)
  return A
```

Algoritmo de Kruskal

Análise de Complexidade

```
Kruskal(G, w)
  for each v ∈ G.V
      MakeSet(v)
  let E' = sort(G.E, w)
  A = ∅
  for each (u,v) ∈ E'
      if FindSet(u) ≠ FindSet(v)
          A = A ∪ {(u,v)}
          Union(u,v)
  return A
```

- for each / MakeSet: $O(V)$
- sort: $O(V \cdot \log V)$

- O ciclo é executado $|E|$ vezes
  - Custo do FindSet
  - Custo do <u>Union</u>

  $O(\log |V|)$

- Total: $O(|E| \cdot \log |V|)$

Algoritmo de Kruskal - Lema Chave

Lema [Kruskal]

Seja $G = (V, E, w)$ um grafo dirigido pesado, A o subconjunto de de uma MST de G e $C = (V_C, E_C)$ um qualquer componente da floresta $G_A = (V, A)$; então:

- O arco de menor peso que liga C a outro componente componente de G_A é seguro para A.

Prova

Algoritmo de Kruskal – Lema Chave

Lema [Kruskal]

Seja $G = (V, E, w)$ um grafo dirigido pesado, A o subconjunto de de uma MST de G e $C = (V_C, E_C)$ um qualquer componente da floresta $G_A = (V, A)$; então:

- O arco de menor peso que liga C a outro componente componente de G_A é seguro para A.

Prova

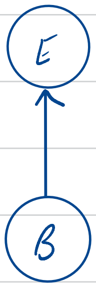
- $(V_C, V \setminus V_C)$ é um corte que respeita A.
- O arco mais leve q liga C a outro componente de G_A é o arco leve que cruza $(V_C, V \setminus V_C)$.
- Aplicando o Teorema Arco Leve ⇒ Arco seguro, concluímos que o arco escolhido é seguro para A.

Algoritmo Union-Find - Representação de Conjuntos Disjuntos

Ideia: Conjuntos são representados como árvores n-árias.

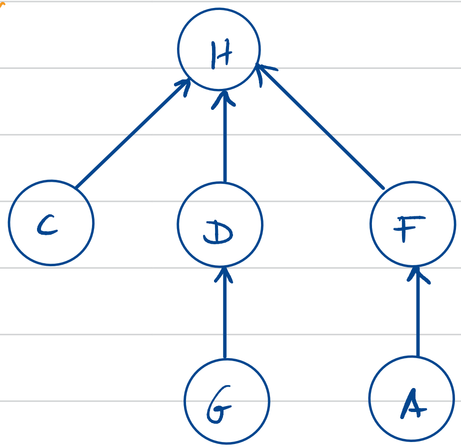
Exemplos:

- Conjunto {B, E}



Obs: o filho aponta p/ o pai

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



Representação Interna: Cada nó tem:

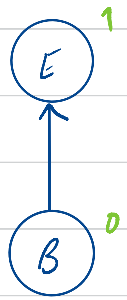
- um pai
- um rank: uma estimativa da "altura" do nó na árvore

Algoritmo Union-Find - Representação de Conjuntos Disjuntos

Ideia: Conjuntos são representados como árvores n-árias.

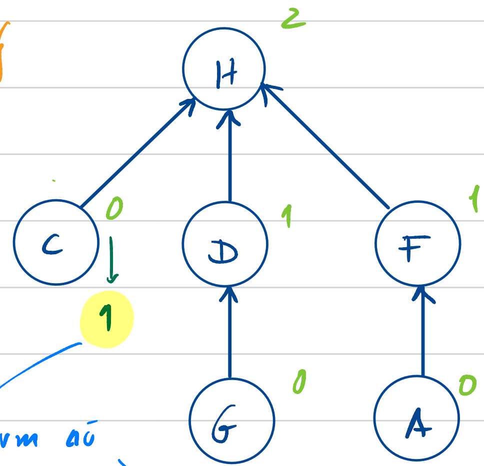
Exemplos:

- Conjunto {B, E}



Obs: o filho aponta p/ o pai

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



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

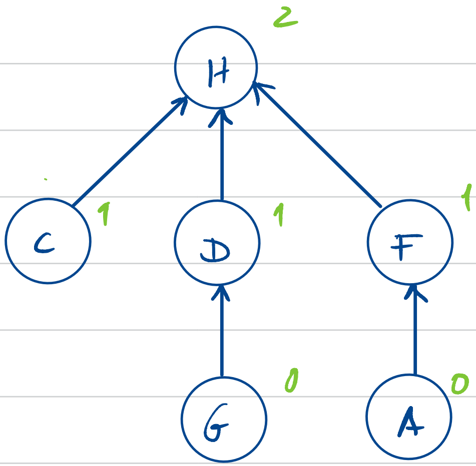
Observações:

- 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 da "altura" do nó na árvore que o contém.

Algoritmo Union-Find

Make-Set (x) ⇒ cria o conjunto com o elemento x

Find Set (x) ⇒ retorna o representante do conjunto que contém x



- Find Set (A) = ?
- Find Set (G) = ?

Algoritmo Union-Find

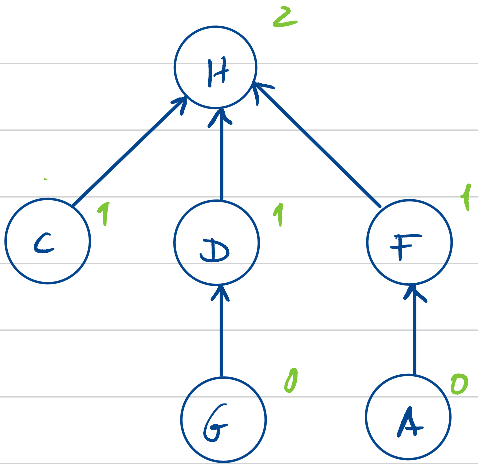
Make-Set(x) ⇒ cria o conjunto com o elemento x

MakeSet(x)
  x.p := x
  x.rank := 0

Complexidade: $O(1)$

FindSet(x) ⇒ retorna o representante do conjunto que contém x

FindSet(x)
  while (x ≠ x.p)
    x := x.p
  return x



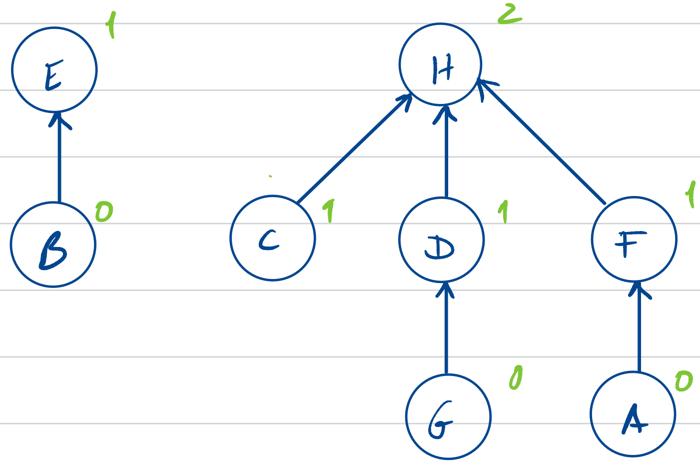
- FindSet(A) = ?
- FindSet(G) = ?

Algoritmo Union-Find

Union (x, y) ⇒ Faz a união dos conjuntos q̄ contém x e y

Union (x, y)

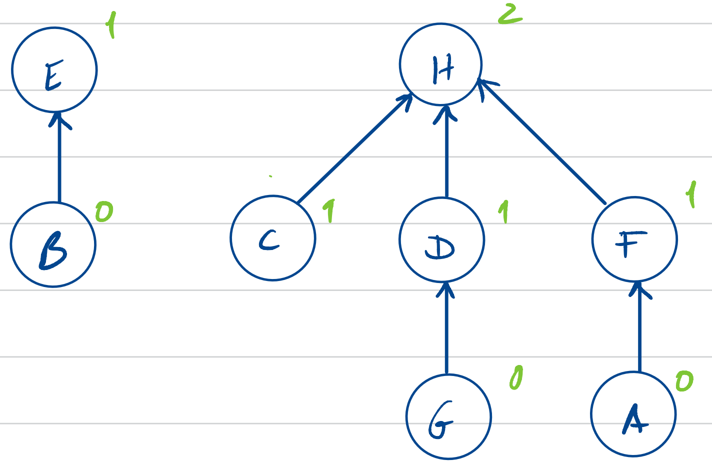
- Union (B, A) ?



Algoritmo Union-Find

Union(x,y) ⇒ Faz a união dos conjuntos q̄ contém x e y

- Union(B, A)?



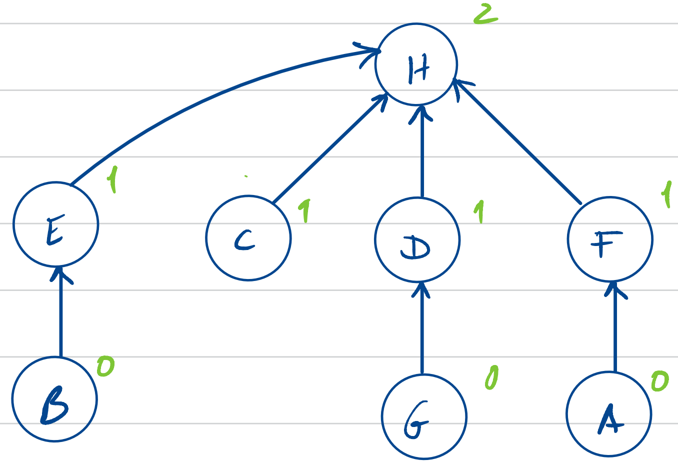

```
Union(x,y)
  let Rx = FindSet(x)
  let Ry = FindSet(y)
  if (Rx == Ry) return
  if (Rx.Rank > Ry.Rank)
     Ry.p := Rx
  else if (Ry.Rank > Rx.Rank)
     Rx.p := Ry
  else
     Ry.p := Rx
     Rx.Rank := Rx.Rank + 1
```

$\}$ Rx.Rank == Ry.Rank

Algoritmo Union - Find

Union (x, y) ⇒ Faz a união dos conjuntos $\bar{q}$ contém x e y

- Union (B, A) ?



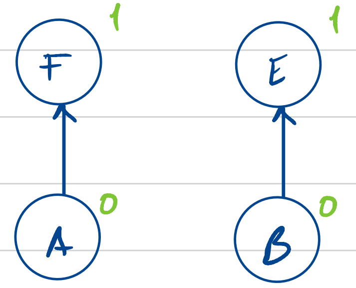

```
Union (x,y)
  let Rx = FindSet(x)
  let Ry = FindSet(y)
  if (Rx == Ry) return
  if (Rx.Rank > Ry.Rank)
     Ry.p := Rx
  else if (Ry.Rank > Rx.Rank)
     Rx.p := Ry
  else
     Ry.p := Rx
     Rx.Rank := Rx.Rank + 1
```

$\}$ $R_x.Rank == R_y.Rank$

Algoritmo Union-Find

Union(x, y) ⇒ faz a união dos conjuntos q contém x e y

- Union(A, E)



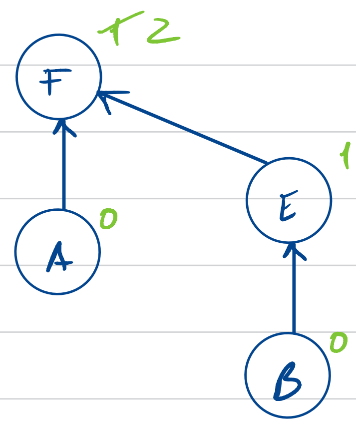

```
Union(x,y)
  let Rx = FindSet(x)
  let Ry = FindSet(y)
  if (Rx == Ry) return
  if (Rx.rank > Ry.rank)
      Ry.p := Rx
  else if (Ry.rank > Rx.rank)
      Rx.p := Ry
  else
      Ry.p := Rx
      Rx.rank := Rx.rank + 1
```

$R_x.rank == R_y.rank$

Algoritmo Union-Find

Union(x, y) ⇒ Faz a união dos conjuntos q̄ contém x e y

- Union(A, E)




```
Union(x,y)
  let Rx = FindSet(x)
  let Ry = FindSet(y)
  if (Rx == Ry) return
  if (Rx.rank > Ry.rank)
      Ry.p := Rx
  else if (Ry.rank > Rx.Rank)
      Rx.p := Ry
  else
      Ry.p := Rx
      Rx.Rank := Rx.Rank + 1
```

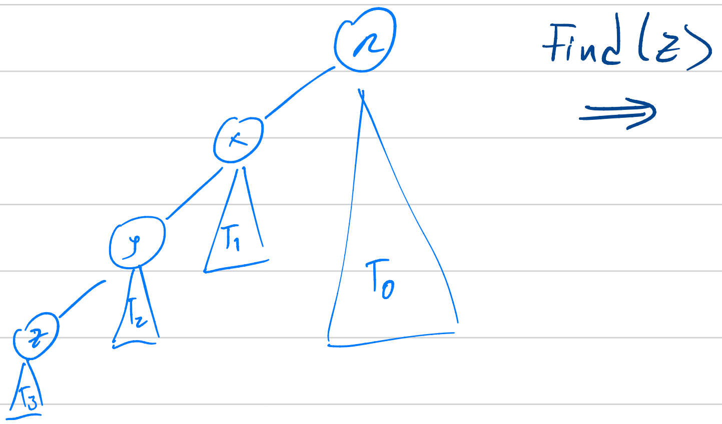
} $R_x.Rank == R_y.Rank$

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 ≠ x.p
    x.p := FindSet(x.p)
  return x.p
```



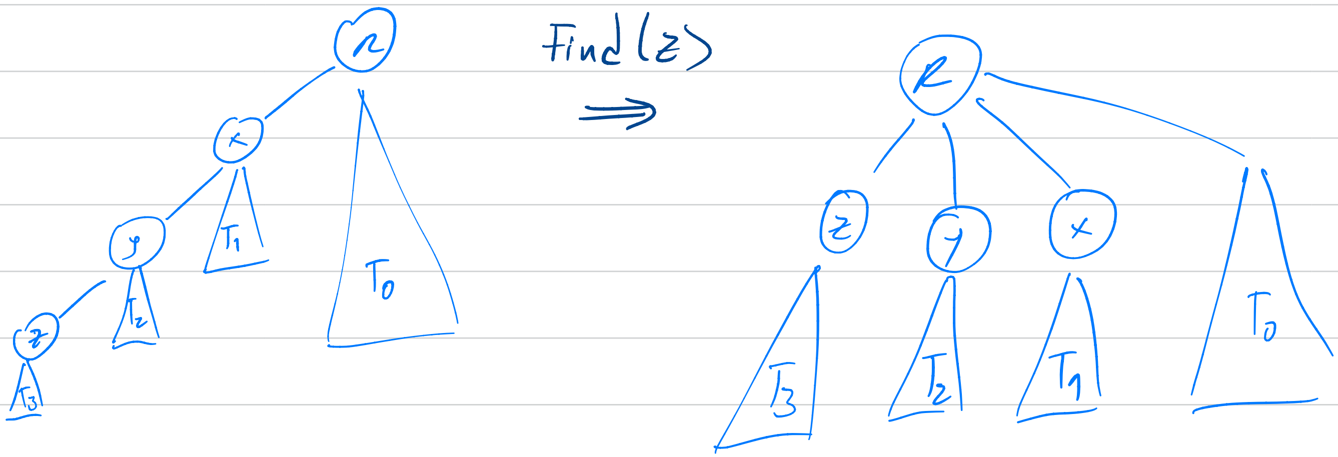
- Operações de Find geram árvores mais achatadas.

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 ≠ x.p
     x.p := FindSet(x.p)
  return x.p
```



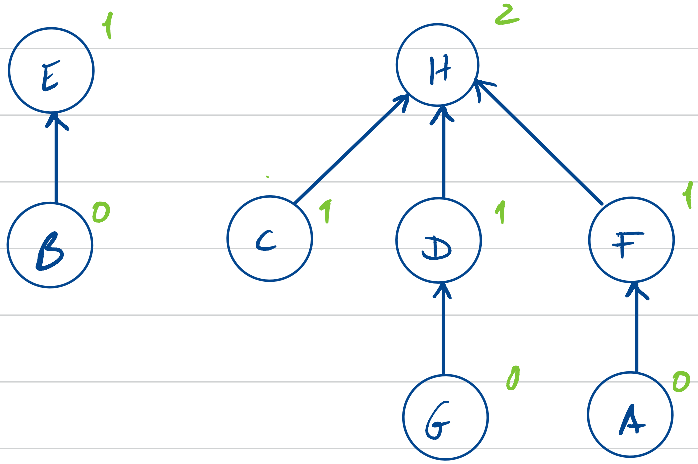
- Operações de Find geram árvores mais achatadas.

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 ≠ x.p
      x.p := FindSet(x.p)
   return x.p
```



Union(B, A)

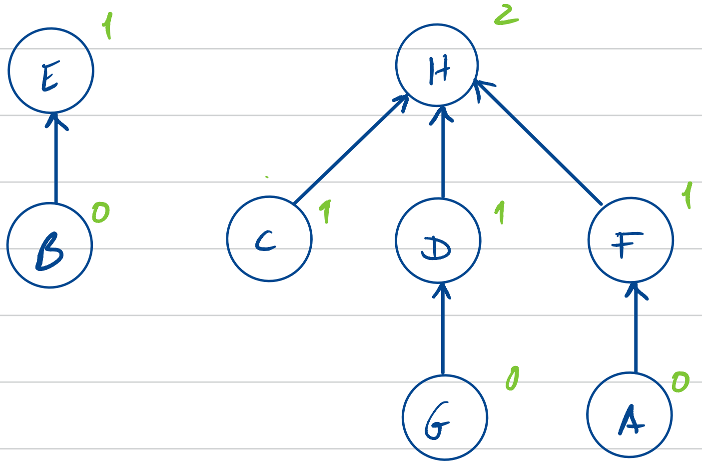
⇒

Algoritmo Union-Find: Compressão de Caminho

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

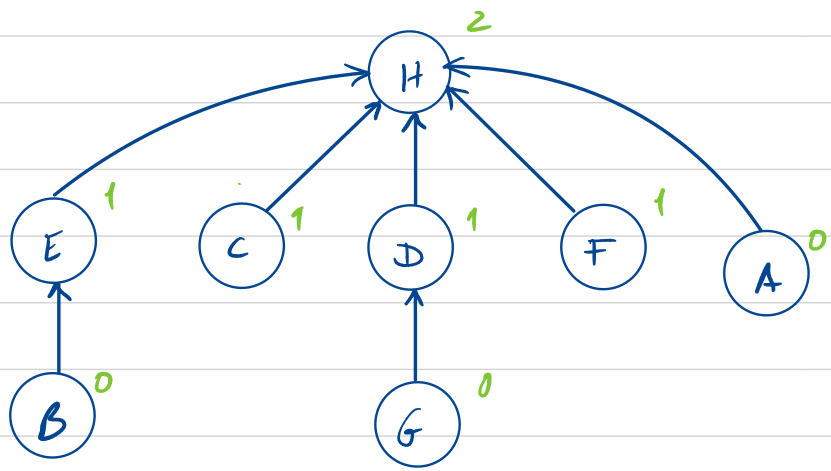
⇒ A árvore que contém x é achatada durante a operação de FindSet

```
FindSet(x)
  if x ≠ x.p
    x.p := FindSet(x.p)
  return x.p
```

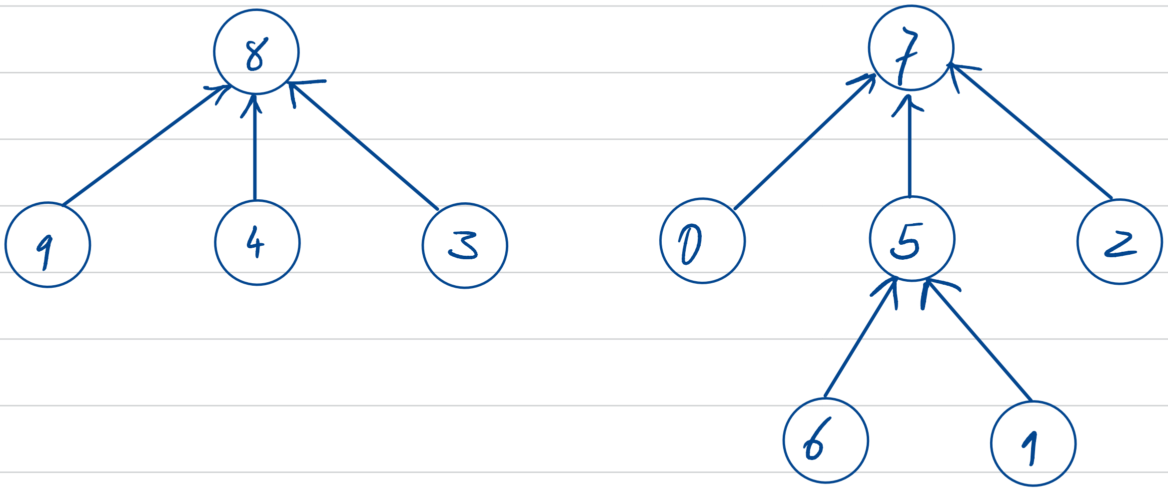


Union(B, A)

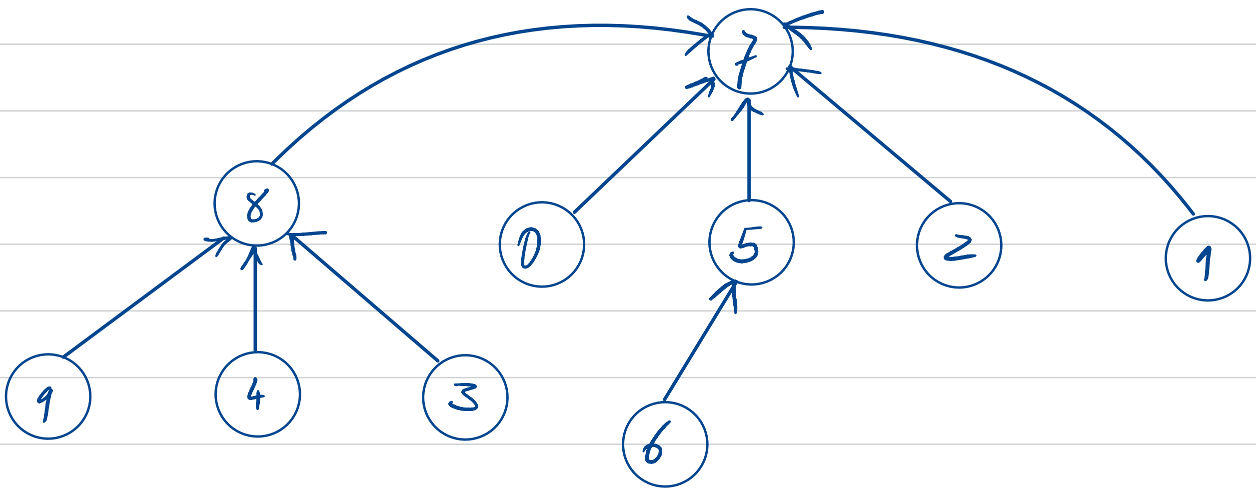
⇒



Algoritmo Union-Find: Compressão de Caminho



Union (1, 3):



Algoritmo Union Find - Análise Teórica

Objectivo: Provar que a complexidade de FindSet(x) e Union(x, y) 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-2]

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 FindSet(x) e Union(x, y) 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-2]

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 estritamente 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 $n/2^R$

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

Prova

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

$$n/2^R = 1 \iff 2^R = n \iff R = \log_2 n$$

Algoritmo Union Find - Análise Teórica

Lema dos Ranks

O nº de nós com rank R é no máximo $n/2^R$

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

Prova

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

$$n/2^R = 1 \iff 2^R = n \iff R = \log_2 n$$

Algoritmo Union Find - Análise Teórica

Lema dos Ranks

O nº de nós com rank R é no máximo $n/2^R$

Lema Auxiliar

Um nó com Rank R é raiz de uma árvore com pelo menos 2^R elementos.

Prova

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

$\underline{n=0}$ No início todos os elementos são raízes de árvores com rank 0.
Cada nó tem exactamente 1 elemento $(1 = 2^0)$. ✓

$\underline{n>0}$ Queremos provar que a proposição é verdadeira depois de Union(x,y).
Há 2 casos a considerar:

I $R_x = R_y$

II $R_x > R_y$ ou $R_y > R_x$

Algoritmo Union Find - Análise Teórica

Lema dos Ranks

O nº de nós com rank R é no máximo $n/2^R$

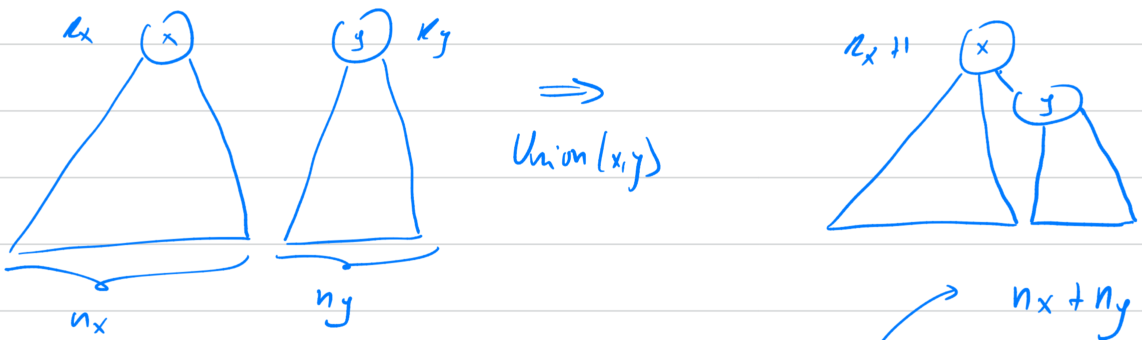
Lema Auxiliar

Um nó com rank R é raiz de uma árvore com pelo menos 2^R elementos.

Prova

(I) $R_x = R_y$

$R_x' = R_x + 1$ Queremos provar que o nº de elementos da nova árvore é $\geq 2^{R_x+1}$



• Da HI: $n_x \geq 2^{R_x}$ e $n_y \geq 2^{R_y} = 2^{R_x}$

$$n_x + n_y \geq 2^{R_x} + 2^{R_y} = 2 \cdot 2^{R_x} = 2^{R_x+1}$$ ✓

Algoritmo Union Find - Análise Teórica

Lema dos Ranks

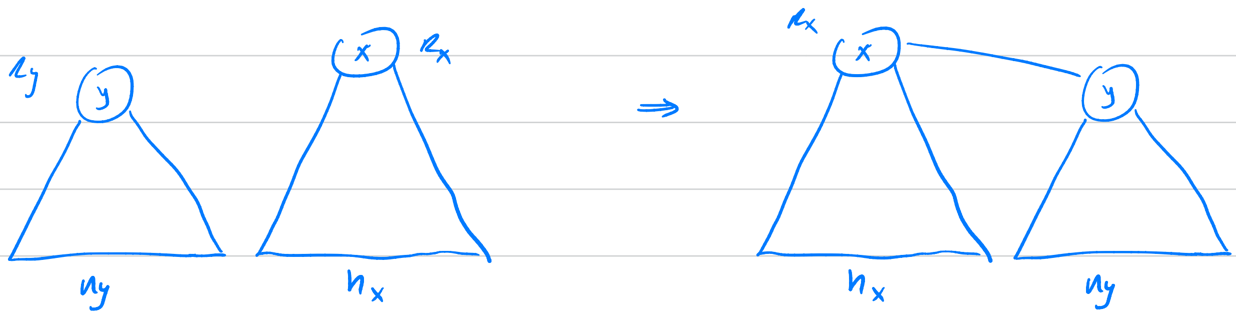
O nº de nós com rank R é no máximo $n/2^R$

Lema Auxiliar

Um nó com Rank R é raiz de uma árvore com pelo menos 2^R elementos.

Prova

I $R_y < R_x$ ($R_x < R_y$ é simétrico)



$$n_x + n_y \geq 2^{R_x}$$

$$\underbrace{n_x \geq 2^{R_x}}_{IH} \quad \checkmark$$

Algoritmo Union-Find: Complexidade

- m operações Union-Find numa estrutura com n nós têm complexidade:
  - $O(m.\log n) \Rightarrow$ sem compressão de caminho
  - $O(m.\alpha(n)) \Rightarrow$ com compressão de caminho

    $\alpha(n)$ → Inversa da função de Ackermann