Análise amortizada

CLRS sec 17.4

Tabelas dinâmicas

Vetor que sofre inserções. Cada inserção custa 1.

Inicialmente o vetor tem 0 posições.

Tabelas dinâmicas

Vetor que sofre inserções. Cada inserção custa 1.

Inicialmente o vetor tem 0 posições.

Na primeira inserção, um vetor com uma posição é alocado, e o item em questão é inserido.

Tabelas dinâmicas

Vetor que sofre inserções. Cada inserção custa 1.

Inicialmente o vetor tem 0 posições.

Na primeira inserção, um vetor com uma posição é alocado, e o item em questão é inserido.

A cada inserção em que o vetor está cheio, antes da inserção propriamente dita, um vetor do dobro do tamanho é alocado, o vetor anterior é copiado para o novo vetor e depois é desalocado.

Tabelas dinâmicas

Vetor que sofre inserções. Cada inserção custa 1.

Inicialmente o vetor tem 0 posições.

Na primeira inserção, um vetor com uma posição é alocado, e o item em questão é inserido.

A cada inserção em que o vetor está cheio, antes da inserção propriamente dita, um vetor do dobro do tamanho é alocado, o vetor anterior é copiado para o novo vetor e depois é desalocado.

O custo no pior caso de uma inserção é alto,
pois pode haver uma realocação.

Tabelas dinâmicas

Para $i = 1, 2, \ldots, n$,

$$c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i & \text{se } i-1 \text{ é potência de 2} \end{cases}$$

Tabelas dinâmicas

Para $i = 1, 2, \ldots, n$,

$$c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i & \text{se } i-1 \text{ é potência de 2} \end{cases}$$

Método agregado:

$$\sum_{i=0}^{n-1} c_i = n + (1 + 2 + 2^2 + \cdots + 2^k)$$

onde $k = \lfloor \lg(n-1) \rfloor$.

Tabelas dinâmicas

Para $i = 1, 2, \ldots, n$,

$$c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i & \text{se } i-1 \text{ é potência de 2} \end{cases}$$

Método agregado:

$$\sum_{i=0}^{n-1} c_i = n + (1 + 2 + 2^2 + \cdots + 2^k)$$

onde $k = \lfloor \lg(n-1) \rfloor$.

Logo $\sum_{i=0}^{n-1} c_i = n + 2^{k+1} - 1 < n + 2n - 1 < 3n$.

Tabelas dinâmicas

Para $i = 1, 2, \ldots, n$,

$$c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i & \text{se } i-1 \text{ é potência de 2} \end{cases}$$

Método agregado:

$$\sum_{i=0}^{n-1} c_i = n + (1 + 2 + 2^2 + \cdots + 2^k)$$

onde $k = \lfloor \lg(n-1) \rfloor$.

Logo $\sum_{i=0}^{n-1} c_i = n + 2^{k+1} - 1 < n + 2n - 1 < 3n$.

Custo amortizado por inserção: 3

Análise por créditos

Chame de velho um item que já estava no vetor no momento da última realocação do vetor, e de novos os itens inseridos após a última realocação.

Análise por créditos

Chame de velho um item que já estava no vetor no momento da última realocação do vetor, e de novos os itens inseridos após a última realocação.

Atribuímos 3 créditos por inserção:
- um é usado para pagar pela inserção do item,
- os outros dois são armazenados sobre o item.

Análise por créditos

Chame de velho um item que já estava no vetor no momento da última realocação do vetor, e de novos os itens inseridos após a última realocação.

Atribuímos 3 créditos por inserção:

um é usado para pagar pela inserção do item,
os outros dois são armazenados sobre o item.

Ao ocorrer uma realocação, há 2 créditos sobre cada item novo no vetor, e isso é suficiente para pagar pela cópia de todos os itens do vetor para o novo vetor pois, quando o vetor está cheio, há um item novo para cada item velho.

Análise por créditos

Chame de velho um item que já estava no vetor no momento da última realocação do vetor, e de novos os itens inseridos após a última realocação.

Atribuímos 3 créditos por inserção:
 um é usado para pagar pela inserção do item,
 os outros dois são armazenados sobre o item.

Ao ocorrer uma realocação, há 2 créditos sobre cada item novo no vetor, e isso é suficiente para pagar pela cópia de todos os itens do vetor para o novo vetor pois, quando o vetor está cheio, há um item novo para cada item velho.

Em outras palavras, o segundo crédito paga a cópia do item na primeira realocação que acontecer após a sua inserção, e o terceiro crédito paga a cópia de um item velho nesta mesma realocação.

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i s_i: tamanho da tabela T_i.

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i $\quad$ s_i: tamanho da tabela T_i.

Seja $\phi(T_i) = 2n_i - s_i$.

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i s_i: tamanho da tabela T_i.

Seja $\phi(T_i) = 2n_i - s_i$.

Como T_0 é vazia, $n_0 = s_0 = 0$, e portanto $\phi(T_0) = 0$.

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i s_i: tamanho da tabela T_i.

Seja $\phi(T_i) = 2n_i - s_i$.

Como T_0 é vazia, $n_0 = s_0 = 0$, e portanto $\phi(T_0) = 0$.

Como não há remoção, $n_i \geq s_i/2$. Logo $\phi(T_i) \geq 0$.

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i $\quad s_i$: tamanho da tabela T_i.

Seja $\phi(T_i) = 2n_i - s_i$.

Como T_0 é vazia, $n_0 = s_0 = 0$, e portanto $\phi(T_0) = 0$.

Como não há remoção, $n_i \geq s_i/2$. Logo $\phi(T_i) \geq 0$.

Lembre-se que $c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i & \text{se } i-1 \text{ é potência de 2} \end{cases}$

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i s_i: tamanho da tabela T_i.

Seja $\phi(T_i) = 2n_i - s_i$.

Como T_0 é vazia, $n_0 = s_0 = 0$, e portanto $\phi(T_0) = 0$.

Como não há remoção, $n_i \geq s_i/2$. Logo $\phi(T_i) \geq 0$.

Lembre-se que $c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i & \text{se } i-1 \text{ é potência de 2} \end{cases}$

Custo amortizado: $\hat{c}_i = c_i + \phi(T_i) - \phi(T_{i-1})$

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i s_i: tamanho da tabela T_i.

Seja $\phi(T_i) = 2n_i - s_i$.

Como T_0 é vazia, $n_0 = s_0 = 0$, e portanto $\phi(T_0) = 0$.

Como não há remoção, $n_i \geq s_i/2$. Logo $\phi(T_i) \geq 0$.

Lembre-se que $c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i & \text{se } i-1 \text{ é potência de 2} \end{cases}$

Custo amortizado: $\hat{c}_i = c_i + \phi(T_i) - \phi(T_{i-1})$

Note que $n_i = n_{i-1} + 1$.

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i s_i: tamanho da tabela T_i.

Seja $\phi(T_i) = 2n_i - s_i$.

Como T_0 é vazia, $n_0 = s_0 = 0$, e portanto $\phi(T_0) = 0$.

Como não há remoção, $n_i \geq s_i/2$. Logo $\phi(T_i) \geq 0$.

Lembre-se que $c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i & \text{se } i-1 \text{ é potência de 2} \end{cases}$

Custo amortizado: $\hat{c}_i = c_i + \phi(T_i) - \phi(T_{i-1})$

Note que $n_i = n_{i-1} + 1$.

Se $i-1$ não é potência de 2, então $c_i = 1$ e $s_i = s_{i-1}$.

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i s_i: tamanho da tabela T_i.

Seja $\phi(T_i) = 2n_i - s_i$.

Como T_0 é vazia, $n_0 = s_0 = 0$, e portanto $\phi(T_0) = 0$.

Como não há remoção, $n_i \geq s_i/2$. Logo $\phi(T_i) \geq 0$.

Lembre-se que $c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i & \text{se } i-1 \text{ é potência de 2} \end{cases}$

Custo amortizado: $\hat{c}_i = c_i + \phi(T_i) - \phi(T_{i-1})$

Note que $n_i = n_{i-1} + 1$.

Se $i-1$ não é potência de 2, então $c_i = 1$ e $s_i = s_{i-1}$.
Assim $\hat{c}_i = 1 + (2n_i - s_i) - (2n_{i-1} - s_{i-1}) = 1 + 2 = 3$.

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i s_i: tamanho da tabela T_i.

$\phi(T_i) = 2n_i - s_i$.

Lembre-se que $c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i+1 & \text{se } i-1 \text{ é potência de 2} \end{cases}$

Custo amortizado: $\hat{c}_i = c_i + \phi(T_i) - \phi(T_{i-1})$

Lembre-se que $n_i = n_{i-1} + 1$.

Se i é potência de 2, então $\ldots$

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i s_i: tamanho da tabela T_i.

$\phi(T_i) = 2n_i - s_i$.

Lembre-se que $c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i+1 & \text{se } i-1 \text{ é potência de 2} \end{cases}$

Custo amortizado: $\hat{c}_i = c_i + \phi(T_i) - \phi(T_{i-1})$

Lembre-se que $n_i = n_{i-1} + 1$.

Se i é potência de 2, então $\ldots$

$c_i = i$, $s_i = 2s_{i-1}$ e $s_{i-1} = n_{i-1} = i - 1$.

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i s_i: tamanho da tabela T_i.

$\phi(T_i) = 2n_i - s_i$.

Lembre-se que $c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i+1 & \text{se } i-1 \text{ é potência de 2} \end{cases}$

Custo amortizado: $\hat{c}_i = c_i + \phi(T_i) - \phi(T_{i-1})$

Lembre-se que $n_i = n_{i-1} + 1$.

Se i é potência de 2, então $\ldots$

$c_i = i$, $s_i = 2s_{i-1}$ e $s_{i-1} = n_{i-1} = i - 1$.

Assim

$$\hat{c}_i = i + (2n_i - s_i) - (2n_{i-1} - s_{i-1})$$

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i s_i: tamanho da tabela T_i.

$\phi(T_i) = 2n_i - s_i$.

Lembre-se que $c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i+1 & \text{se } i-1 \text{ é potência de 2} \end{cases}$

Custo amortizado: $\hat{c}_i = c_i + \phi(T_i) - \phi(T_{i-1})$

Lembre-se que $n_i = n_{i-1} + 1$.

Se i é potência de 2, então $\ldots$

$c_i = i$, $s_i = 2s_{i-1}$ e $s_{i-1} = n_{i-1} = i - 1$.

Assim

$$\begin{aligned} \hat{c}_i &= i + (2n_i - s_i) - (2n_{i-1} - s_{i-1}) \\ &= i + (2(n_{i-1}+1) - 2s_{i-1}) - n_{i-1} \end{aligned}$$

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i $\quad s_i$: tamanho da tabela T_i.

$\phi(T_i) = 2n_i - s_i$.

Lembre-se que $c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i+1 & \text{se } i-1 \text{ é potência de 2} \end{cases}$

Custo amortizado: $\hat{c}_i = c_i + \phi(T_i) - \phi(T_{i-1})$

Lembre-se que $n_i = n_{i-1} + 1$.

Se i é potência de 2, então $\ldots$

$c_i = i$, $\quad s_i = 2s_{i-1} \quad$ e $\quad s_{i-1} = n_{i-1} = i - 1$.

Assim

$$\begin{aligned} \hat{c}_i &= i + (2n_i - s_i) - (2n_{i-1} - s_{i-1}) \\ &= i + (2(n_{i-1}+1) - 2s_{i-1}) - n_{i-1} \\ &= i + (2 + n_{i-1} - 2s_{i-1}) \end{aligned}$$

Método do potencial

T_i: tabela antes da inserção i

n_i: número de itens na tabela T_i s_i: tamanho da tabela T_i.

$\phi(T_i) = 2n_i - s_i$.

Lembre-se que $c_i = \begin{cases} 1 & \text{se } i-1 \text{ não é potência de 2} \\ i+1 & \text{se } i-1 \text{ é potência de 2} \end{cases}$

Custo amortizado: $\hat{c}_i = c_i + \phi(T_i) - \phi(T_{i-1})$

Lembre-se que $n_i = n_{i-1} + 1$.

Se i é potência de 2, então $\ldots$

$c_i = i$, $s_i = 2s_{i-1}$ e $s_{i-1} = n_{i-1} = i - 1$.

Assim

$$\begin{aligned} \hat{c}_i &= i + (2n_i - s_i) - (2n_{i-1} - s_{i-1}) \\ &= i + (2(n_{i-1}+1) - 2s_{i-1}) - n_{i-1} \\ &= i + (2 + n_{i-1} - 2s_{i-1}) \\ &= i + (2 - (i-1)) \;=\; 3. \end{aligned}$$

Union-Find

CLRS cap 21

Coleção de conjuntos disjuntos

Queremos uma ED boa para representar uma partição de um conjunto, e as seguintes operações sobre a partição:

Coleção de conjuntos disjuntos

Queremos uma ED boa para representar uma partição de um conjunto, e as seguintes operações sobre a partição:

- MAKESET(x): cria um conjunto unitário com o elemento x;
- FIND(x): devolve o identificador do conjunto da partição que contém x;
- UNION(x, y): substitui os conjuntos da partição que contêm x e y pela união deles.

Coleção de conjuntos disjuntos

Queremos uma ED boa para representar uma partição de um conjunto, e as seguintes operações sobre a partição:

- MAKESET(x): cria um conjunto unitário com o elemento x;
- FIND(x): devolve o identificador do conjunto da partição que contém x;
- UNION(x, y): substitui os conjuntos da partição que contêm x e y pela união deles.

O identificador de um conjunto é um elemento do conjunto: o seu representante.

Coleção de conjuntos disjuntos

Queremos uma ED boa para representar uma partição de um conjunto, e as seguintes operações sobre a partição:

- MAKESET(x): cria um conjunto unitário com o elemento x;
- FIND(x): devolve o identificador do conjunto da partição que contém x;
- UNION(x, y): substitui os conjuntos da partição que contêm x e y pela união deles.

O identificador de um conjunto é um elemento do conjunto: o seu representante.

Como podemos armazenar cada conjunto da partição?

Coleção de conjuntos disjuntos

Possibilidade 1:

Coleção de conjuntos disjuntos

Possibilidade 1: os elementos de cada conjunto da partição são armazenados em uma lista (duplamente) ligada.

Coleção de conjuntos disjuntos

Possibilidade 1: os elementos de cada conjunto da partição são armazenados em uma lista (duplamente) ligada.

Qual é o custo de cada operação?

representante: o primeiro elemento da lista ligada.
n: número de elementos na união dos conjuntos.

Coleção de conjuntos disjuntos

Possibilidade 1: os elementos de cada conjunto da partição são armazenados em uma lista (duplamente) ligada.

Qual é o custo de cada operação?

representante: o primeiro elemento da lista ligada.
n: número de elementos na união dos conjuntos.

- MAKESET(x): cria uma lista ligada com uma única célula, contendo x. Consumo de tempo no pior caso: $\Theta(1)$.

Coleção de conjuntos disjuntos

Possibilidade 1: os elementos de cada conjunto da partição são armazenados em uma lista (duplamente) ligada.

Qual é o custo de cada operação?

representante: o primeiro elemento da lista ligada.
n: número de elementos na união dos conjuntos.

- MAKESET(x): cria uma lista ligada com uma única célula, contendo x. Consumo de tempo no pior caso: $\Theta(1)$.
- FIND(x): recebe um apontador x e devolve o apontador do representante do conjunto que contém x. Consumo de tempo no pior caso: $\Theta(n)$.

Coleção de conjuntos disjuntos

Possibilidade 1: os elementos de cada conjunto da partição são armazenados em uma lista (duplamente) ligada.

Qual é o custo de cada operação?

representante: o primeiro elemento da lista ligada.
n: número de elementos na união dos conjuntos.

- MAKESET(x): cria uma lista ligada com uma única célula, contendo x. Consumo de tempo no pior caso: $\Theta(1)$.
- FIND(x): recebe um apontador x e devolve o apontador do representante do conjunto que contém x. Consumo de tempo no pior caso: $\Theta(n)$.
- UNION(x, y): recebe os representantes de dois conjuntos distintos e substitui esses conjuntos pela união deles. Consumo de tempo no pior caso: $\Theta(n)$.

Coleção de conjuntos disjuntos

Possibilidade 2:

Coleção de conjuntos disjuntos

Possibilidade 2: os elementos de cada conjunto da partição são armazenados em uma lista ligada, em que todos os elementos apontam para o primeiro da lista (o representante).

Coleção de conjuntos disjuntos

Possibilidade 2: os elementos de cada conjunto da partição são armazenados em uma lista ligada, em que todos os elementos apontam para o primeiro da lista (o representante).

Qual é o custo de cada operação?

Coleção de conjuntos disjuntos

Possibilidade 2: os elementos de cada conjunto da partição são armazenados em uma lista ligada, em que todos os elementos apontam para o primeiro da lista (o representante).

Qual é o custo de cada operação?

- MAKESET(x): cria uma lista ligada com uma única célula x. Consumo de tempo no pior caso: $\Theta(1)$.

Coleção de conjuntos disjuntos

Possibilidade 2: os elementos de cada conjunto da partição são armazenados em uma lista ligada, em que todos os elementos apontam para o primeiro da lista (o representante).

Qual é o custo de cada operação?

- MAKESET(x): cria uma lista ligada com uma única célula x. Consumo de tempo no pior caso: $\Theta(1)$.
- FIND(x): recebe um apontador x e devolve o apontador do representante do conjunto que contém x. Consumo de tempo no pior caso: $\Theta(1)$.

Coleção de conjuntos disjuntos

Possibilidade 2: os elementos de cada conjunto da partição são armazenados em uma lista ligada, em que todos os elementos apontam para o primeiro da lista (o representante).

Qual é o custo de cada operação?

- MAKESET(x): cria uma lista ligada com uma única célula x. Consumo de tempo no pior caso: $\Theta(1)$.
- FIND(x): recebe um apontador x e devolve o apontador do representante do conjunto que contém x. Consumo de tempo no pior caso: $\Theta(1)$.
- UNION(x, y): recebe os representantes de dois conjuntos distintos e substitui esses conjuntos pela união deles. Consumo de tempo no pior caso: $\Theta(n)$ (pois temos que atualizar o representante de um dos conjuntos).

Coleção de conjuntos disjuntos

Possibilidade 3: florestas disjuntas.

Coleção de conjuntos disjuntos

Possibilidade 3: florestas disjuntas.

Primeiro: esqueça os apontadores da lista ligada.

Coleção de conjuntos disjuntos

Possibilidade 3: florestas disjuntas.

Primeiro: esqueça os apontadores da lista ligada.

Segundo: atualize o representante apenas do representante do segundo conjunto.

Coleção de conjuntos disjuntos

Possibilidade 3: florestas disjuntas.

Primeiro: esqueça os apontadores da lista ligada.

Segundo: atualize o representante apenas do representante do segundo conjunto.

Exemplo: na aula...

Implementação 1

Make-Set (x)

pai$[x] \leftarrow x$

Implementação 1

Make-Set (x)

pai$[x] \leftarrow x$

Find (x)

$r \leftarrow x$
enquanto pai$[r] \neq r$ **faça**
    $r \leftarrow$ pai$[r]$
devolva r

Implementação 1

Make-Set (x)

  pai$[x] \leftarrow x$

Find (x)

  $r \leftarrow x$

  enquanto pai$[r] \neq r$ **faça**

    $r \leftarrow$ pai$[r]$

  devolva r

Union (x, y) $\triangleright$ x e y representantes distintos

  pai$[y] \leftarrow x$

Implementação 1

Make-Set (x)

  pai$[x] \leftarrow x$

Find (x)

  $r \leftarrow x$

  enquanto pai$[r] \neq r$ **faça**

    $r \leftarrow$ pai$[r]$

  devolva r

Union (x, y) $\triangleright$ x e y representantes distintos

  pai$[y] \leftarrow x$

Consumo de tempo: do Find pode ser muito ruim... $\Theta(n)$.

Temos que fazer melhor...

Implementação 2

Heurística dos tamanhos

Make-Set (x)
1 pai$[x] \leftarrow x$
2 rank$[x] \leftarrow 0$

Implementação 2

Heurística dos tamanhos

Make-Set (x)

1 pai$[x] \leftarrow x$

2 rank$[x] \leftarrow 0$

Find (x): o mesmo de antes

Implementação 2

Heurística dos tamanhos

```
Make-Set (x)
    pai[x] ← x
    rank[x] ← 0
```

Find (x): o mesmo de antes

```
Union (x, y)        ▷ x e y representantes distintos
    se rank[x] ≥ rank[y]
        então pai[y] ← x
              se rank[x] = rank[y]
                  então rank[x] ← rank[x] + 1
        senão pai[x] ← y
```

Implementação 2

Heurística dos tamanhos

Make-Set (x)

  pai$[x] \leftarrow x$
  rank$[x] \leftarrow 0$

Find (x): o mesmo de antes

Union (x, y) $\triangleright$ x e y representantes distintos

  se rank$[x] \geq$ rank$[y]$
    então pai$[y] \leftarrow x$
      se rank$[x] =$ rank$[y]$
        então rank$[x] \leftarrow$ rank$[x] + 1$
    senão pai$[x] \leftarrow y$

Consumo de tempo: melhor... Análise feita na aula...
Dá para fazer melhor ainda!

Implementação 3

Heurística da compressão dos caminhos

Find (x)
if pai$[x] \neq x$
  então pai$[x] \leftarrow$ Find (pai$[x]$)
devolva pai$[x]$

Implementação 3

Heurística da compressão dos caminhos

Find (x)
1 if pai$[x] \neq x$
2 **então** pai$[x] \leftarrow$ Find (pai$[x]$)
3 **devolva** pai$[x]$

Consumo *amortizado* de tempo de cada operação:

$$O(\log^* n),$$

onde $\log^* n$ é o número de vezes que temos que aplicar o $\log$ até atingir um número menor ou igual a 1.

Implementação 3

Heurística da compressão dos caminhos

Find (x)
if pai$[x] \neq x$
    então pai$[x] \leftarrow$ Find (pai$[x]$)
devolva pai$[x]$

Consumo *amortizado* de tempo de cada operação:

$$O(\log^* n),$$

onde $\log^* n$ é o número de vezes que temos que aplicar o $\log$ até atingir um número menor ou igual a 1.

Na verdade, é melhor do que isso,
e há uma análise justa, conforme discutido em aula.

Union-Find

Make-Set (x)

pai$[x] \leftarrow x$
rank$[x] \leftarrow 0$

Find (x)

if pai$[x] \neq x$
    então pai$[x] \leftarrow$ Find (pai$[x]$)
devolva pai$[x]$

Union (x, y) $\quad \triangleright$ x e y representantes distintos

se rank$[x] \geq$ rank$[y]$
    então pai$[y] \leftarrow x$
        se rank$[x] =$ rank$[y]$
            então rank$[x] \leftarrow$ rank$[x] + 1$
    senão pai$[x] \leftarrow y$

Aplicação: você vai ver na aula de grafos na segunda!