

CLRS cap 21

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  $x$ .

**FINDSET** ( $x$ ): devolve o identificador do bloco da partição que contém  $x$ .

**UNION** ( $x, y$ ): substitui os blocos da partição que contêm  $x$  e  $y$  pela união deles.

Idéia:

Usar um elemento do conjunto (o *representante*) como *identificador*.

## Como analisar?

Uso típico:

$n - 1$  **UNION**,  $m$  **FINDSET**,  $m \gg n$

em sequência arbitrária.

O custo de uma operação pode variar muito - é **muito** pessimista supor o pior caso em cada uma.

Vale a pena considerar uma sequência completa de operações, em vez de analisar uma a uma.

Isso se chama **Análise Amortizada**.

É um método especialmente útil para analisar estruturas adaptativas, em que uma operação pode preparar caminho para a execução de operações futuras.

## Não é tão novidade assim

Numa busca em grafos:

- 1 para cada  $u \in G.V$
- 2 para cada  $v \in u$ . **Estrela** faça

Análise ingênua:

Como cada estrela tem tamanho  $O(n)$ , os dois laços combinados levam  $O(n^2)$ .

Análise amortizada:

Como na linha 2 cada arco é examinado uma única vez, os dois laços combinados levam  $O(m)$ .

## Implementação 1 do union-find

- ▶ Cada conjunto é uma lista ligada, com cabeçalho apontando início e fim - facilita UNION.
- ▶ Cada elemento da lista tem um apontador nome para o cabeçalho - facilita FINDSET.

MAKESET ( $x$ ) cria uma lista contendo só  $x$ .

FINDSET respondido direto pelo nome.

UNION: juntar as duas listas é fácil, mas é preciso atualizar o nome dos membros de um dos conjuntos.

MAKESET e FINDSET levam  $O(1)$ , mas vários UNION podem gastar  $\Omega(n^2)$  mudanças de nome.

## Melhoria

Na UNION, pendure a lista menor na maior (cabeçalho precisa guardar o tamanho da lista).

Agora, cada vez que um elemento muda de nome, o conjunto em que ele está pelo menos dobra de tamanho. Assim, seu nome muda no máximo  $\lg n$  vezes.

Tempo total de  $n$  UNION:  $O(n \lg n)$ .

Tempo total:  $O(m + n \lg n)$

## Implementação 2 do union-find

Cada conjunto é uma árvore apontando para a raiz.

MAKESET ( $x$ )  
1  $x.pai \leftarrow x$

FINDSET ( $x$ )  
1 se  $x = x.pai$   
2 devolva  $x$   
3 devolva FINDSET ( $x.pai$ )

FINDSET ( $x$ )  
1 enquanto  $x.pai \neq x$  faça  
2  $x \leftarrow x.pai$   
3 devolva  $x$

UNION ( $x, y$ )  $\triangleright x$  e  $y$  representantes distintos  
1  $y.pai \leftarrow x$

Consumo de tempo: do FINDSET pode ser muito ruim...  $\Theta(n)$ .  
Temos que fazer melhor...

## Melhoria 1

Heurística das alturas

MAKESET ( $x$ )  
1  $x.pai \leftarrow x$   
2  $x.rank \leftarrow 0$

FINDSET ( $x$ ): o mesmo de antes

UNION ( $x, y$ )  $\triangleright x$  e  $y$  representantes distintos  
1 se  $x.rank \geq y.rank$   
2 então  $y.pai \leftarrow x$   
3 se  $x.rank = y.rank$   
4 então  $x.rank \leftarrow x.rank + 1$   
5 senão  $x.pai \leftarrow y$

INVARIANTE:  $x.rank \geq$  altura da árvore pendurada em  $x$ .  
Melhorou: FINDSET é  $\Theta(\lg n)$ . Total:  $O(n + m \lg n)$   
Um pouco pior que antes, mas dá para fazer melhor ainda!

## Implementação 3

### Heurística da compressão dos caminhos

```

FINDSET (x)
1  if x.pai ≠ x
2    então x.pai ← FINDSET (x.pai)
3  devolva x.pai

```

### Consumo amortizado de tempo de cada operação:

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

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

Na verdade, é melhor do que isso.

A análise desta ED é vista na disciplina MAC6711.

## O que isso significa

Duas maneiras de entender o  $\lg^*$ :

$$\lg^{(0)}(n) = n$$

$$\lg^{(k)}(n) = \lg(\lg^{(k-1)}(n)), \quad k > 0$$

$$\lg^*(n) = \min\{k \mid \lg^{(k)}(n) \leq 1\}$$

$$b_0 = 1$$

$$b_k = 2^{b_{k-1}}, \quad k > 0$$

$$\lg^*(n) = \min\{k \mid n < b_k\}.$$

## O que isso significa

$$b_n = \underbrace{2^{2^{2^{\dots}}}}_{n \text{ times}}^2$$

$k$	$b_k$
0	1
1	2
2	4
3	16
4	65536
5	$> 10^{19000}$

No mundo real,  $\lg^* n \leq 5$ .