

Aula 03: 13/03/2018

Tópico

- *union-find*

Union-Find

Leitura: [Case Study: Union-Find, S&W](#)

Vídeos: [Union-find e Kruskal, Gabriel Russo, canal BCC](#) e [Union-find, Robert Sedgewick](#).

Considere uma coleção de conjuntos disjuntos S_1, S_2, \dots, S_n .

Conjuntos são modificados ao longo do tempo.

Terminologia utiliza metáfora de redes: sítios/*sites*, conexão,...

API

<code>public class</code>	<code>UF</code>	
	<code>UF(int n)</code>	inicialize n sites com nomes inteiros $0, \dots, n-1$
<code>void</code>	<code>union(int p, int q)</code>	acrescenta ligação os sítios p e q
<code>int</code>	<code>find(int p)</code>	retorna nome (=id) do componente contendo p
<code>boolean</code>	<code>connected(int p, int q)</code>	true se p e q estão em um mesmo componente
<code>int</code>	<code>count()</code>	número de componentes

Uso

```
% java Cliente
3
1 2
1 2
2 components
1 0
1 0
1 components
2 0
1 components
% java Cliente < tinyUF.txt
4 3
9 components
3 8
8 components
6 5
7 components
9 4
```

```

6 components
2 1
5 components
8 9
5 components
5 0
4 components
7 2
3 components
6 1
2 components
1 0
2 components
6 7
2 components

```

Estrutura

```
QuickFindUF uf = new QuickFindUF(10);
```

```

uf ----+
      |
      V
+-----+
| id (private)                count: 0 (private) |
| |                             |                 | | | | | | | | | |
| | +---+---+---+---+---+---+---+---+---+---+ |
| +--> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |
|      +---+---+---+---+---+---+---+---+---+ |
|          0  1  2  3  4  5  6  7  8  9         |
| |                             |                 |
| Métodos: cont(), connected(), find(), union() |
| |                             |                 |
+-----+

```

Quick-find

```
UF uf = new UF(10);
```

```

uf ----+
      |
      V
+-----+
| id (private)                count: 10 (private) |
| |                             |                 | | | | | | | | | |
| | +---+---+---+---+---+---+---+---+---+---+ |
| +--> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |
|      +---+---+---+---+---+---+---+---+---+ |

```

```

|         0  1  2  3  4  5  6  7  8  9         |
|
| Métodos: cont(), connected(), find(), union() |
|
+-----+

```

uf.find(3) retorna 3
uf.find(0) retorna 0

```
uf.union(4, 3);
```

```
uf ----+
      |
      V

```

```

+-----+
| id (private)                count:  9 (private) |
| |                            |                    | | | | | | | | |
| | +---+---+---+---+---+---+---+---+---+---+ |
| +--> | 0 | 1 | 2 | 3 | 3 | 5 | 6 | 7 | 8 | 9 |
|       +---+---+---+---+---+---+---+---+---+ |
|         0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union() |
|
+-----+

```

uf.find(3) retorna 3
uf.find(4) retorna 3

```
uf.union(3, 8);
```

```
uf ----+
      |
      V

```

```

+-----+
| id (private)                count:  8 (private) |
| |                            |                    | | | | | | | | |
| | +---+---+---+---+---+---+---+---+---+---+ |
| +--> | 0 | 1 | 2 | 8 | 8 | 5 | 6 | 7 | 8 | 9 |
|       +---+---+---+---+---+---+---+---+---+ |
|         0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union() |
|
+-----+

```

```
uf.union(6, 5);
```

```
uf ----+
```

|
V

```
+-----+
| id (private)                count: 7 (private) |
| |                            | | | | | | | | | | |
| | +-----+-----+-----+-----+-----+ |
| +--> | 0 | 1 | 2 | 8 | 8 | 5 | 5 | 7 | 8 | 9 | |
|       +-----+-----+-----+-----+-----+ |
|         0  1  2  3  4  5  6  7  8  9 |
|
| Métodos: cont(), connected(), find(), union() |
|
+-----+
```

```
uf.union(9, 4);
```

```
uf ----+
      |
      V
```

```
+-----+
| id (private)                count: 6 (private) |
| |                            | | | | | | | | | | |
| | +-----+-----+-----+-----+-----+ |
| +--> | 0 | 1 | 2 | 8 | 8 | 5 | 5 | 7 | 8 | 8 | |
|       +-----+-----+-----+-----+-----+ |
|         0  1  2  3  4  5  6  7  8  9 |
|
| Métodos: cont(), connected(), find(), union() |
|
+-----+
```

```
uf.union(2, 1);
```

```
uf ----+
      |
      V
```

```
+-----+
| id (private)                count: 5 (private) |
| |                            | | | | | | | | | | |
| | +-----+-----+-----+-----+-----+ |
| +--> | 0 | 1 | 1 | 8 | 8 | 5 | 5 | 7 | 8 | 8 | |
|       +-----+-----+-----+-----+-----+ |
|         0  1  2  3  4  5  6  7  8  9 |
|
| Métodos: cont(), connected(), find(), union() |
|
+-----+
```

```
uf.union(8, 9);
```

```

uf ----+
  |
  V
+-----+
| id (private)                count: 5 (private) |
| |                            |                    | | | | | | | | | |
| |      +---+---+---+---+---+---+---+---+---+ |
| +--> | 0 | 1 | 1 | 8 | 8 | 5 | 5 | 7 | 8 | 8 | |
|       +---+---+---+---+---+---+---+---+---+ |
|         0  1  2  3  4  5  6  7  8  9          |
|
| Métodos: cont(), connected(), find(), union() |
|
+-----+

```

```
uf.union(5, 0);
```

```

uf ----+
  |
  V
+-----+
| id (private)                count: 5 (private) |
| |                            |                    | | | | | | | | | |
| |      +---+---+---+---+---+---+---+---+---+ |
| +--> | 0 | 1 | 1 | 8 | 8 | 0 | 0 | 7 | 8 | 8 | |
|       +---+---+---+---+---+---+---+---+---+ |
|         0  1  2  3  4  5  6  7  8  9          |
|
| Métodos: cont(), connected(), find(), union() |
|
+-----+

```

```
uf.union(7, 2);
```

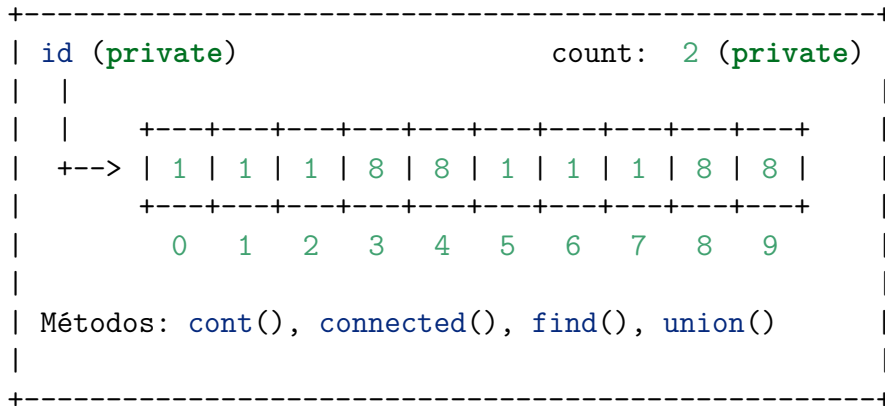
```

uf ----+
  |
  V
+-----+
| id (private)                count: 3 (private) |
| |                            |                    | | | | | | | | | |
| |      +---+---+---+---+---+---+---+---+---+ |
| +--> | 0 | 1 | 1 | 8 | 8 | 0 | 0 | 1 | 8 | 8 | |
|       +---+---+---+---+---+---+---+---+---+ |
|         0  1  2  3  4  5  6  7  8  9          |
|
| Métodos: cont(), connected(), find(), union() |
|
+-----+

```

```
uf.union(6, 1);
```

```
uf ----+
      |
      V
```



[...]

```
public class QuickFindUF {
    private int[] id;
    private int count;

    // Initializes an empty union-find data structure with n
    // isolated components 0 through n-1
    public UF(int n) {
        count = n;
        id = new int[n];
        for (int i = 0; i < n; i++) {
            id[i] = i;
        }
    }

    // Return the number of components.
    public int count() {
        return count;
    }

    // Are the two sites p and q in the same component?
    public boolean connected(int p, int q) {
        return find(p) == find(q);
    }

    // Return the component identifier for the component
    // containing site p.
    public int find(int p) {
        return id[p];
    }

    // Merge the component containing site p with the the
    // component containing site q.
```

```

public void union(int p, int q) {
    int pID = find(p);
    int qID = find(q);

    if (pID == qID) return ;

    for (int i = 0; i < id.length; i++) {
        if (id[i] == pID) id[i] = qID;
    }
    count--;
}
}

```

Consumo de tempo

UF()	n
union()	n
find()	1

Experimentos

```

% java Driver < tinyUF.txt
2 components
0.002seg
% java Driver < mediumUF.txt
3 components
0.032seg
% java Driver < largeUF.txt
:-(

```

Hmm. Em `union()` seria razoáveis alterarmos o menor número possível de posições do vetor `id[]`. Para isso precisamos saber qual conjunto tem o menor número de itens.

Quick-union

A ideia é trocar o indicador `id` do componente por um indicador do “pai” do sítio. Por sua vez, se `p` é um sítio, o `pai p` indica quem é avô (`pai[pai[p]]`), que indica quem é o bisavô (`pai[pai[pia[p]]]`), que indica o sítio tataravô,...

O **representante** ou **nome** de um componente será o sítio que é o pai de si mesmo. Hmm. Aqui a metáfora fica meio estranha.

É intuitivo representarmos a estrutura através de um conjunto de árvores disjuntas (= **floresta**) onde as raízes das árvores são os sítios `p` tais que `p == pai[p]`.

Simulação

```
QuickUnionUF uf = new QuickUnion(10);
```

```
uf ----+
      |
      V
```

```
+-----+
| pai (private)                count: 10 (private)|
| |
| | +---+---+---+---+---+---+---+---+---+
| +--> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|       +---+---+---+---+---+---+---+---+---+
|         0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+
```

```
uf.find(3) retorna 3
uf.find(0) retorna 0
```

```
uf.union(4, 3);
```

```
uf ----+
      |
      V
```

```
+-----+
| pai (private)                count:  9 (private)|
| |
| | +---+---+---+---+---+---+---+---+---+
| +--> | 0 | 1 | 2 | 3 | 3 | 5 | 6 | 7 | 8 | 9 |
|       +---+---+---+---+---+---+---+---+---+
|         0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+
```

```
uf.find(3) retorna 3
uf.find(4) retorna 3
```

```
uf.union(3, 8);
```

```
uf ----+
      |
      V
```

```
+-----+
| pai (private)                count:  8 (private)|
| |
```



```

| | +---+---+---+---+---+---+---+---+---+
| +--> | 0 | 1 | 2 | 8 | 3 | 5 | 6 | 7 | 8 | 9 |
|       +---+---+---+---+---+---+---+---+
|         0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

```
uf.union(6, 5);
```

```

uf ----+
    |
    V
+-----+
| pai (private)                count: 7 (private)|
| |
| | +---+---+---+---+---+---+---+---+
| +--> | 0 | 1 | 2 | 8 | 3 | 5 | 5 | 7 | 8 | 9 |
|       +---+---+---+---+---+---+---+---+
|         0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

```
uf.union(9, 4);
```

```

uf ----+
    |
    V
+-----+
| pai (private)                count: 6 (private)|
| |
| | +---+---+---+---+---+---+---+---+
| +--> | 0 | 1 | 2 | 8 | 3 | 5 | 5 | 7 | 8 | 8 |
|       +---+---+---+---+---+---+---+---+
|         0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

```
uf.union(2, 1);
```

```

uf ----+
    |
    V
+-----+

```

```

| pai (private)                count: 5 (private)|
| |
| | +---+---+---+---+---+---+---+---+---+---+
| +--> | 0 | 1 | 1 | 8 | 3 | 5 | 5 | 7 | 8 | 8 |
|      +---+---+---+---+---+---+---+---+---+
|          0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

```
uf.union(8, 9);
```

```
uf ----+
      |
      V
```

```

+-----+
| pai (private)                count: 5 (private)|
| |
| | +---+---+---+---+---+---+---+---+---+---+
| +--> | 0 | 1 | 1 | 8 | 3 | 5 | 5 | 7 | 8 | 8 |
|      +---+---+---+---+---+---+---+---+---+
|          0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

```
uf.union(5, 0);
```

```
uf ----+
      |
      V
```

```

+-----+
| pai (private)                count: 4 (private)|
| |
| | +---+---+---+---+---+---+---+---+---+---+
| +--> | 0 | 1 | 1 | 8 | 3 | 0 | 5 | 7 | 3 | 3 |
|      +---+---+---+---+---+---+---+---+---+
|          0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

```
uf.union(7, 2);
```

```
uf ----+
      |
      V
```

```

+-----+
| pai (private)                count:  3 (private) |
| |                             |                 | | | | | | | | | |
| | +---+---+---+---+---+---+---+---+---+---+ |
| +--> | 0 | 1 | 1 | 8 | 3 | 0 | 5 | 1 | 3 | 3 | |
|      +---+---+---+---+---+---+---+---+---+ |
|          0  1  2  3  4  5  6  7  8  9         |
| |                             |                 |
| Métodos: cont(), connected(), find(), union() |
| |                             |                 |
+-----+

```

```
uf.union(6, 1);
```

```
uf -----+
      |
      V

```

```

+-----+
| pai (private)                count:  2 (private) |
| |                             |                 | | | | | | | | | |
| | +---+---+---+---+---+---+---+---+---+---+ |
| +--> | 1 | 1 | 1 | 8 | 3 | 0 | 1 | 1 | 3 | 3 | |
|      +---+---+---+---+---+---+---+---+---+ |
|          0  1  2  3  4  5  6  7  8  9         |
| |                             |                 |
| Métodos: cont(), connected(), find(), union() |
| |                             |                 |
+-----+

```

[...]

Implementação

```

public class QuickUnionUF {
    private int[] pai;
    private int count;

    // Initializes an empty union-find data structure with n
    // isolated components 0 through n-1
    public UF(int n) {
        count = n;
        pai = new int[n];
        for (int i = 0; i < n; i++) {
            pai[i] = i;
        }
    }

    // Return the number of components.
    public int count() {

```

```

    return count;
}

// Are the two sites p and q in the same component?
public boolean connected(int p, int q) {
    return find(p) == find(q);
}

// Return the component paientifier for the component
// containing site p.
public int find(int p) {
    while (p != pai[p]) p = pai[p];
    return p;
}

// Merge the component containing site p with the the
// component containing site q.
public void union(int p, int q) {
    int pRoot = find(p);
    int qRoot = find(q);
    // if (pRoot == qRoot) return ;
    pai[pRoot] = pai[qRoot];
    count--;
}
}

```

Consumo de tempo

UF()	<i>n</i>
union()	<i>n</i>
find()	<i>n</i>

Experimentos

```

% java Driver < ../testes/tinyUF.txt
2 components
0.002seg
% java Driver < ../testes/mediumUF.txt
3 components
0.025seg
% java Driver < ../testes/largeUF.txt
:-(

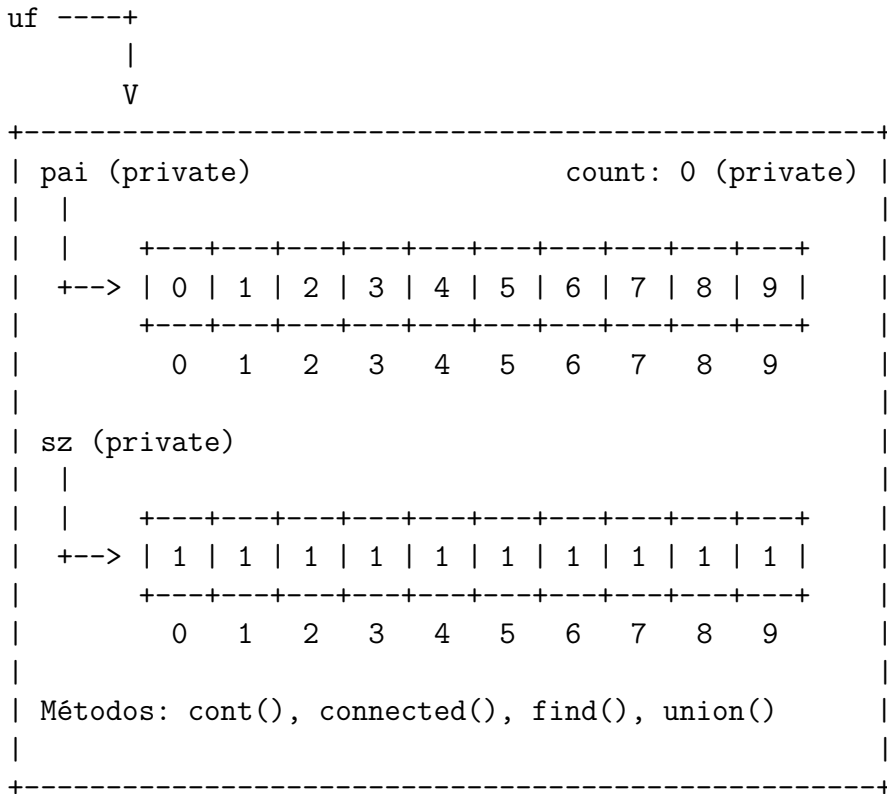
```

Weighted quick-union

Ideia, ligar a raiz da árvore com menos sítios na raiz da árvore com mais sítios. Isso seria a política natural para tornarmos o quick-find mais eficiente.

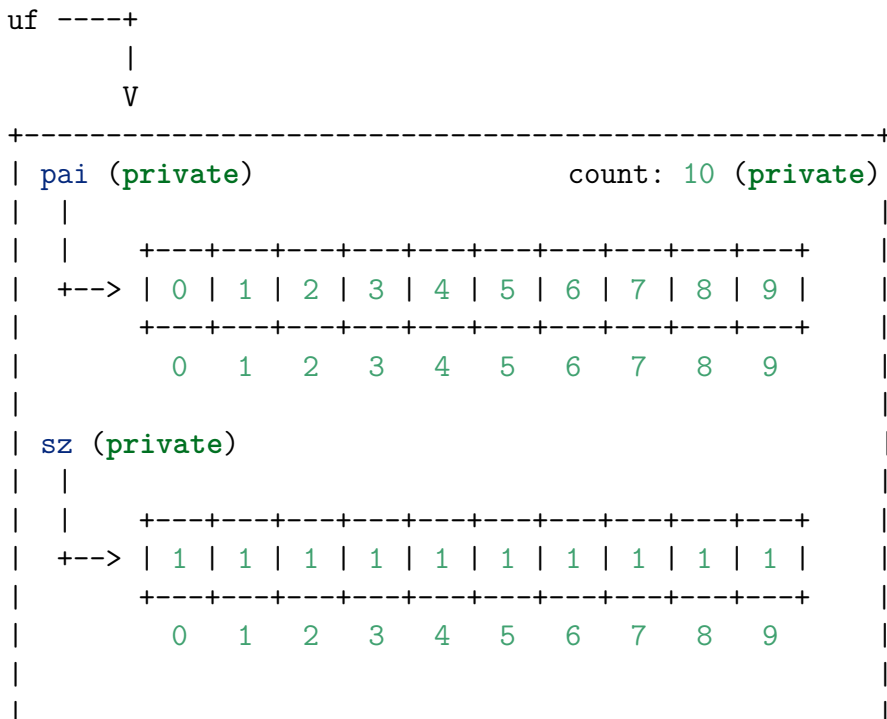
Estrutura

```
UF uf = new UF(10);
```



Simulação

```
WeightedQuickUnionUF uf = new WeightedQuickUnionUF(10);
```



```

| Métodos: cont(), connected(), find(), union() |
| | |
+-----+
uf.find(3) retorna 3
uf.find(0) retorna 0

uf.union(4, 3);

uf ----+
  |
  V
+-----+
| pai (private)                count: 9 (private)|
| | | | | | | | | | | | |
| | +---+---+---+---+---+---+---+---+---+---+ |
| +--> | 0 | 1 | 2 | 4 | 4 | 5 | 6 | 7 | 8 | 9 | |
| | +---+---+---+---+---+---+---+---+---+---+ |
| | 0 1 2 3 4 5 6 7 8 9 | |
| | |
| sz (private) |
| | | | | | | | | | | | |
| | +---+---+---+---+---+---+---+---+---+---+ |
| +--> | 1 | 1 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | |
| | +---+---+---+---+---+---+---+---+---+---+ |
| | 0 1 2 3 4 5 6 7 8 9 | |
| | |
| Métodos: cont(), connected(), find(), union() |
| | |
+-----+

```

```

uf.find(3) retorna 3
uf.find(4) retorna 3

uf.union(3, 8);

uf ----+
  |
  V
+-----+
| pai (private)                count: 8 (private)|
| | | | | | | | | | | | |
| | +---+---+---+---+---+---+---+---+---+---+ |
| +--> | 0 | 1 | 2 | 4 | 4 | 5 | 6 | 7 | 4 | 9 | |
| | +---+---+---+---+---+---+---+---+---+---+ |
| | 0 1 2 3 4 5 6 7 8 9 | |
| | |
| sz (private) |
| | |
+-----+

```

```

| | +-----+
| +--> | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 1 |
|       +-----+
|         0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

```
uf.union(6, 5);
```

```
uf ----+
      |
      V

```

```

+-----+
| pai (private)                count: 7 (private) |
| |
| | +-----+
| +--> | 0 | 1 | 2 | 4 | 4 | 6 | 6 | 7 | 4 | 9 |
|       +-----+
|         0  1  2  3  4  5  6  7  8  9
|
| sz (private)
| |
| | +-----+
| +--> | 1 | 1 | 1 | 1 | 3 | 1 | 2 | 1 | 1 | 1 |
|       +-----+
|         0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

```
uf.union(9, 4);
```

```
uf ----+
      |
      V

```

```

+-----+
| pai (private)                count: 6 (private) |
| |
| | +-----+
| +--> | 0 | 1 | 2 | 4 | 4 | 6 | 6 | 7 | 8 | 4 |
|       +-----+
|         0  1  2  3  4  5  6  7  8  9
|
| sz (private)

```

```

| |
| | +-----+
| +--> | 1 | 1 | 1 | 1 | 4 | 1 | 2 | 1 | 1 | 1 |
|       +-----+
|         0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

```
uf.union(2, 1);
```

```

uf ----+
  |
  V
+-----+
| pai (private)                count: 5 (private)|
| |
| | +-----+
| +--> | 0 | 1 | 1 | 4 | 4 | 6 | 6 | 7 | 8 | 4 |
|       +-----+
|         0  1  2  3  4  5  6  7  8  9
|
| sz (private)
| |
| | +-----+
| +--> | 1 | 2 | 1 | 1 | 3 | 1 | 2 | 1 | 1 | 1 |
|       +-----+
|         0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

```
uf.union(8, 9);
```

```

uf ----+
  |
  V
+-----+
| pai (private)                count: 4 (private)|
| |
| | +-----+
| +--> | 0 | 1 | 1 | 4 | 4 | 6 | 6 | 7 | 4 | 4 |
|       +-----+
|         0  1  2  3  4  5  6  7  8  9
|
| sz (private)

```



```

| |
| | +-----+
| |--> | 1 | 2 | 1 | 1 | 4 | 1 | 2 | 1 | 1 | 1 |
| | +-----+
| | 0 1 2 3 4 5 6 7 8 9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

```
uf.union(5, 0);
```

```

uf ----+
  |
  V
+-----+
| pai (private)                count: 3 (private) |
| |
| | +-----+
| |--> | 6 | 1 | 1 | 4 | 4 | 6 | 6 | 7 | 4 | 4 |
| | +-----+
| | 0 1 2 3 4 5 6 7 8 9
|
| sz (private)
| |
| | +-----+
| |--> | 1 | 2 | 1 | 1 | 4 | 1 | 3 | 1 | 1 | 1 |
| | +-----+
| | 0 1 2 3 4 5 6 7 8 9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

```
uf.union(7, 2);
```

```

uf ----+
  |
  V
+-----+
| pai (private)                count: 2 (private) |
| |
| | +-----+
| |--> | 6 | 1 | 1 | 4 | 4 | 6 | 6 | 1 | 4 | 4 |
| | +-----+
| | 0 1 2 3 4 5 6 7 8 9
|
| sz (private)
| |

```

```

| |
| | +---+---+---+---+---+---+---+---+---+
| +--> | 1 | 3 | 1 | 1 | 4 | 1 | 3 | 1 | 1 | 1 |
|       +---+---+---+---+---+---+---+---+
|         0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

```
uf.union(6, 1);
```

```

uf ----+
      |
      V
+-----+
| pai (private)                count:  2 (private) |
| |
| | +---+---+---+---+---+---+---+---+---+
| +--> | 6 | 6 | 1 | 4 | 4 | 6 | 6 | 1 | 4 | 4 |
|       +---+---+---+---+---+---+---+---+
|         0  1  2  3  4  5  6  7  8  9
|
| sz (private)
| |
| | +---+---+---+---+---+---+---+---+---+
| +--> | 1 | 3 | 1 | 1 | 5 | 1 | 6 | 1 | 1 | 1 |
|       +---+---+---+---+---+---+---+---+
|         0  1  2  3  4  5  6  7  8  9
|
| Métodos: cont(), connected(), find(), union()
|
+-----+

```

Implementação

```

public class WeightedQuickUnionUF {
    private int[] pai;
    private int[] sz;
    private int count;

    // Initializes an empty union-find data structure with n
    // isolated components 0 through n-1
    public UF(int n) {
        count = n;
        pai = new int[n];
        sz = new int[n];
    }
}

```

```

    for (int i = 0; i < n; i++) {
        pai[i] = i;
        sz[i] = 1;
    }
}

// Return the number of components.
public int count() {
    return count;
}

// Are the two sites p and q in the same component?
public boolean connected(int p, int q) {
    return find(p) == find(q);
}

// Return the component identifier for the component
// containing site p.
public int find(int p) {
    while (p != pai[p]) {
        pai[p] = pai[pai[p]]; // diminui o tamanho do caminho a metade
        p = pai[p];
    }
    return p;
}

// Merge the component containing site p with the the
// component containing site q.
public void union(int p, int q) {
    int pRoot = find(p);
    int qRoot = find(q);
    if (pRoot == qRoot) return ;

    if (sz[pRoot] < sz[qRoot]) {
        pai[pRoot] = pai[qRoot];
        sz[qRoot] += sz[pRoot];
    }
    else {
        pai[qRoot] = pai[pRoot];
        sz[pRoot] += sz[qRoot];
    }
    count--;
}
}

```

Consumo de tempo

<hr/>	
UF()	n
union()	$\lg n$
find()	$\lg n$
<hr/>	

Para demonstrar que o consumo de tempo de `union()` e `find()` é não superior a $\lg n$, basta verificar que

Na floresta de árvores disjuntas produzida durante uma sequência de operações `union()`, toda árvore com altura h tem pelo menos 2^h nós.

A demonstração é por indução no número de operações `union()` realizadas.

Inicialmente nenhuma operação `union()` foi realizada e toda árvore tem altura zero e possui um nó. Logo vale a afirmação.

Sejam p e q sítios e considere a operação `union(p, q)`. Se p e q estão em uma mesma árvore não há o que demonstrar. Portanto, podemos supor que a árvore T_p que contém p e árvore T_q que contém q são distintas.

Sejam h_p e n_p a altura e número de nós de T_p e h_q e n_q a altura e número de nós de T_q . Pela hipótese de indução $n_p \geq 2^{h_p}$ e $n_q \geq 2^{h_q}$.

Seja T a árvore de altura h resultante da operação `union(p, q)`. Se $h \leq \max\{h_p, h_q\}$, não há o que demonstrar. Assim, podemos supor que, digamos, $n_p \leq n_q$ e $h = h_q + 1$. Logo,

$$n = n_p + n_q \geq n_q + n_q \geq 2^{h_q} + 2^{h_q} = 2^{h_q+1} = 2^h.$$

O que encerra este rascunho de demonstração.

Experimentos

```
% java Driver < ../testes/tinyUF.txt
2 components
0.003seg
% java Driver < ../testes/mediumUF.txt
3 components
0.027seg
% java Driver < ../testes/largeUF.txt
6 components
4.079seg

em find().

> java Driver < ../testes/tinyUF.txt
2 components
0.003seg
% java Driver < ../testes/mediumUF.txt
3 components
0.025seg
% java Driver < ../testes/largeUF.txt
6 components
3.923seg
```

Weighted quick-union with path compression

Consumo de tempo

UF()	n
union()	$\lg^* n$ amortizado , praticamente constante
find()	$\lg^* n$, amortizado , praticamente constante

Implementação

```
public class UF {
    private int[] pai;
    private int[] sz;
    private int count;

    // Initializes an empty union-find data structure with n
    // isolated components 0 through n-1
    public UF(int n) {
        count = n;
        pai = new int[n];
        sz = new int[n];
        for (int i = 0; i < n; i++) {
            pai[i] = i;
            sz[i] = 1;
        }
    }

    // Return the number of components.
    public int count() {
        return count;
    }

    // Are the two sites p and q in the same component?
    public boolean connected(int p, int q) {
        return find(p) == find(q);
    }

    // Return the component identifier for the component
    // containing site p.
    // Versão recursiva
    public int find(int p) {
        if (p != pai[p]) {
            pai[p] = find(pai[p]);
        }
        return pai[p];
    }

    // Merge the component containing site p with the the
    // component containing site q.
```

```

public void union(int p, int q) {
    int pRoot = find(p);
    int qRoot = find(q);
    if (pRoot == qRoot) return ;

    if (sz[pRoot] < sz[qRoot]) {
        pai[pRoot] = pai[qRoot];
        sz[pRoot] += sz[qRoot];
    }
    else {
        pai[qRoot] = pai[pRoot];
        sz[qRoot] += sz[pRoot];
    }
    count--;
}
}

```

Resumo

	UF()	union()	find()
quick-find	n	n	1
quick-union	n	n	n
weighted quick-union	n	$\lg n$	$\lg n$
com path-compression	n	$\lg^* n$	$\lg^* n$

n	$\lg^* n$
1	0
2	1
4	2
16	3
65536	4
2^{65536}	5

Parece que na prática weighted quick-union e weighted quick-union com path-compression não são muito diferentes.