Aula 12:
Union-Find Disjoint Sets

***Disciplina*: Maratona de Programação 1**

Profs. Edmilson Marmo e Luiz Olmes

edmarmo@unifei.edu.br, olmes@unifei.edu.br

UNIFEI
UNIVERSIDADE FEDERAL DE ITAJUBÁ

Ajuste de Cronograma

Data	Conteúdo
16/10	**9.** Grafos: algoritmos de Dijkstra e Tarjan
23/10	**10.** Warmup 3
30/11	Não houve aula.
06/11	**11.** Upsolving: Warmup 3 + Warmup de Aula
13/11	**12.** Union-Find
20/11	Feriado: não haverá aula.
23/11	**13.** Simulado 2 (Ativ. Extra: sábado)
27/11	**14.** Programação Dinâmica
04/12	**15.** Warmup 4
11/12	**16.** Substitutiva

Nas aulas anteriores...

- **O QUE JÁ ESTUDAMOS?**
  - Introdução à Maratona
  - Problemas ad hoc
  - Standard Template Library (STL)
  - Grafos: DFS e BFS
  - Grafos: algoritmos de Dijkstra e Tarjan
- **OBJETIVOS:**
  - Conectividade dinâmica.
  - Union-Find.
  - Implementação.

Introdução

- Diversas estruturas de dados podem ser empregadas para a resolução de problemas de programação competitiva.
  - Listas
  - Filas
  - Pilhas
  - Árvores
  - Tabelas hash
  - Grafos, etc.

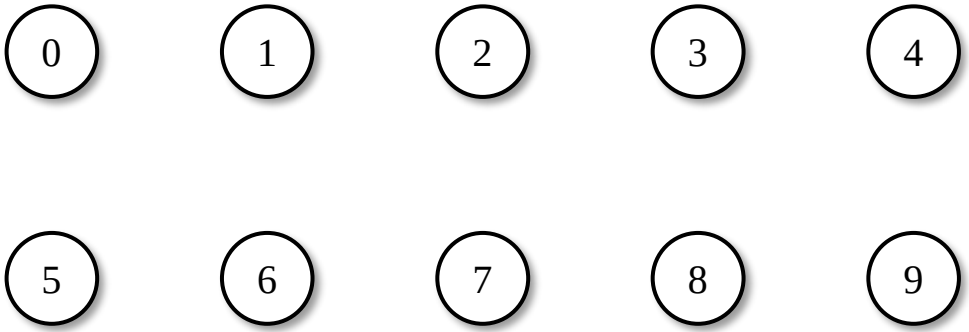
- Uma delas, chamada de Union-Find Disjoint Sets, permite resolver o chamado problema da conectividade dinâmica.

O Problema da Conectividade Dinâmica

- Dado um conjunto com N objetos (sem se importar com a semântica do conjunto), deseja-se suportar duas operações:
  - União (union): conectar dois objetos
  - Consulta (query) da conectividade: existe conexão entre dois objetos?

O Problema da Conectividade Dinâmica

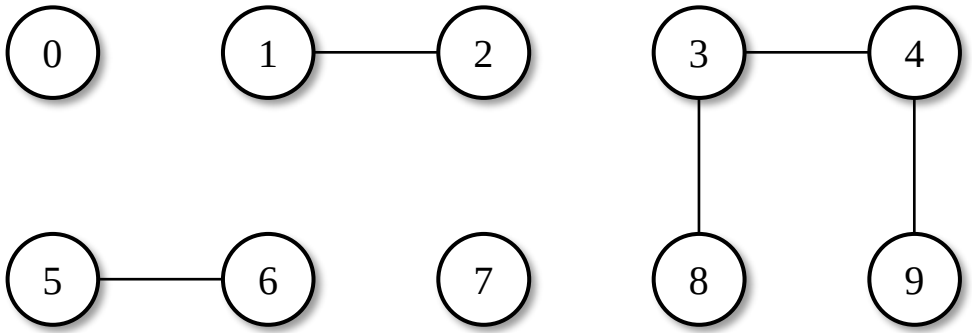
- Dado um conjunto com N objetos (sem se importar com a semântica do conjunto), deseja-se suportar duas operações:
  - União (union): conectar dois objetos
  - Consulta (query) da conectividade: existe conexão entre dois objetos?



O Problema da Conectividade Dinâmica

- Dado um conjunto com N objetos (sem se importar com a semântica do conjunto), deseja-se suportar duas operações:
  - União (union): conectar dois objetos
  - Consulta (query) da conectividade: existe conexão entre dois objetos?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
```

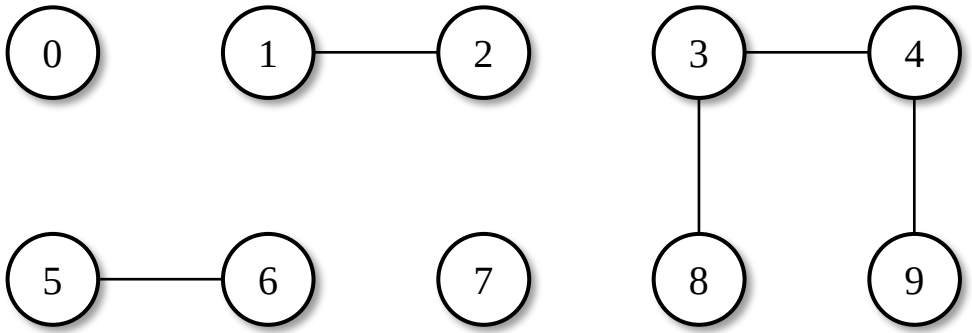


O Problema da Conectividade Dinâmica

- Dado um conjunto com N objetos (sem se importar com a semântica do conjunto), deseja-se suportar duas operações:
  - União (union): conectar dois objetos
  - Consulta (query) da conectividade: existe conexão entre dois objetos?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
query(0, 7)
```

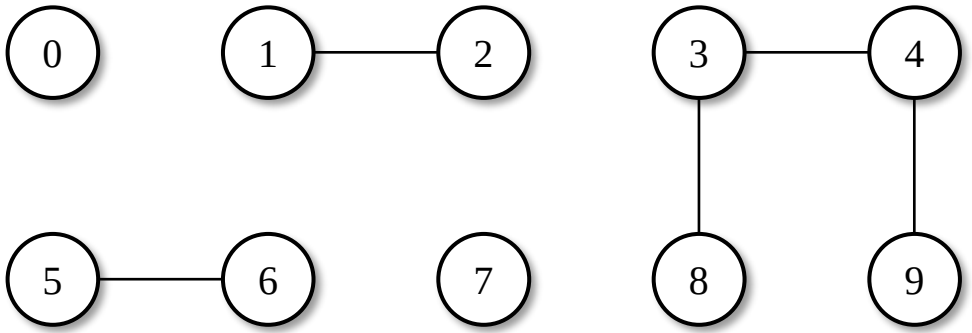
0 e 7 estão conectados?



O Problema da Conectividade Dinâmica

- Dado um conjunto com N objetos (sem se importar com a semântica do conjunto), deseja-se suportar duas operações:
  - União (union): conectar dois objetos
  - Consulta (query) da conectividade: existe conexão entre dois objetos?

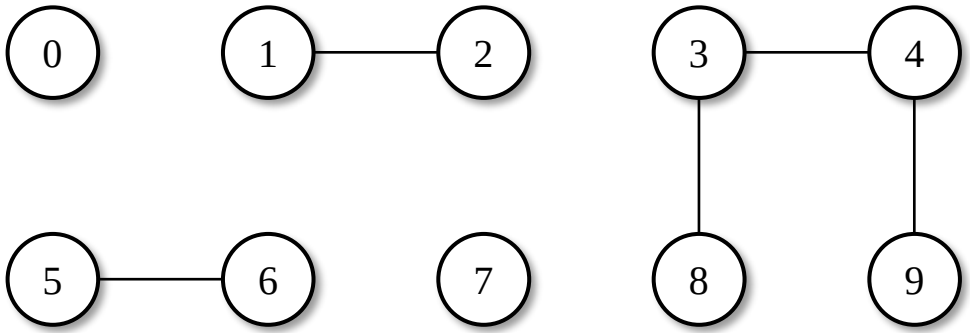
```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
query(0, 7)
```



O Problema da Conectividade Dinâmica

- Dado um conjunto com N objetos (sem se importar com a semântica do conjunto), deseja-se suportar duas operações:
  - União (union): conectar dois objetos
  - Consulta (query) da conectividade: existe conexão entre dois objetos?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
query(0, 7)
query(8, 9)
```

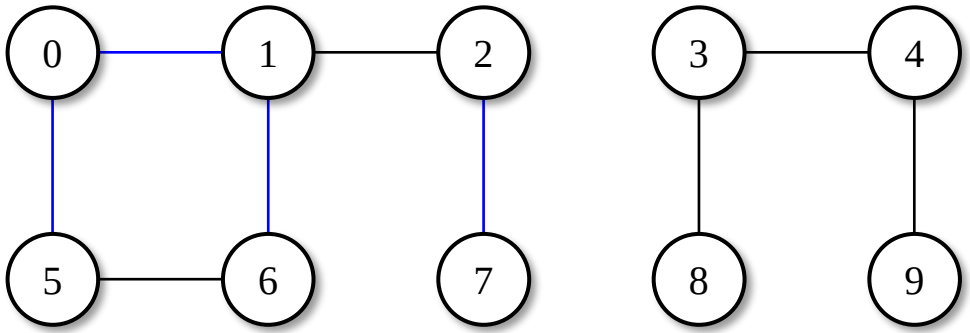


O Problema da Conectividade Dinâmica

- Dado um conjunto com N objetos (sem se importar com a semântica do conjunto), deseja-se suportar duas operações:
  - União (union): conectar dois objetos
  - Consulta (query) da conectividade: existe conexão entre dois objetos?

```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
query(0, 7)
query(8, 9)
```

```
union(5, 0)
union(7, 2)
union(6, 1)
union(1, 0)
```



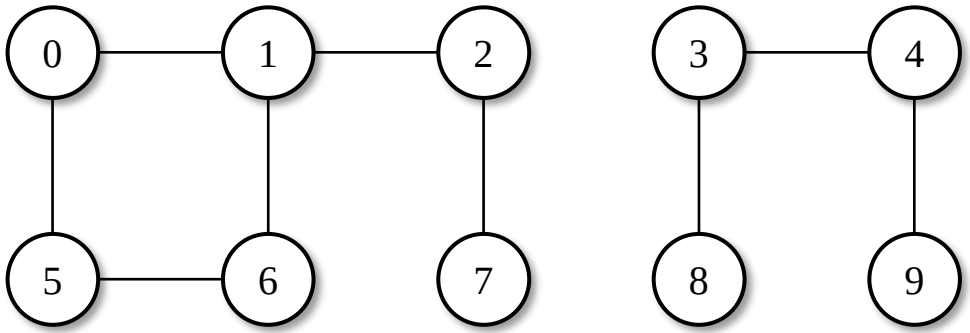
O Problema da Conectividade Dinâmica

- Dado um conjunto com N objetos (sem se importar com a semântica do conjunto), deseja-se suportar duas operações:
  - União (union): conectar dois objetos
  - Consulta (query) da conectividade: existe conexão entre dois objetos?

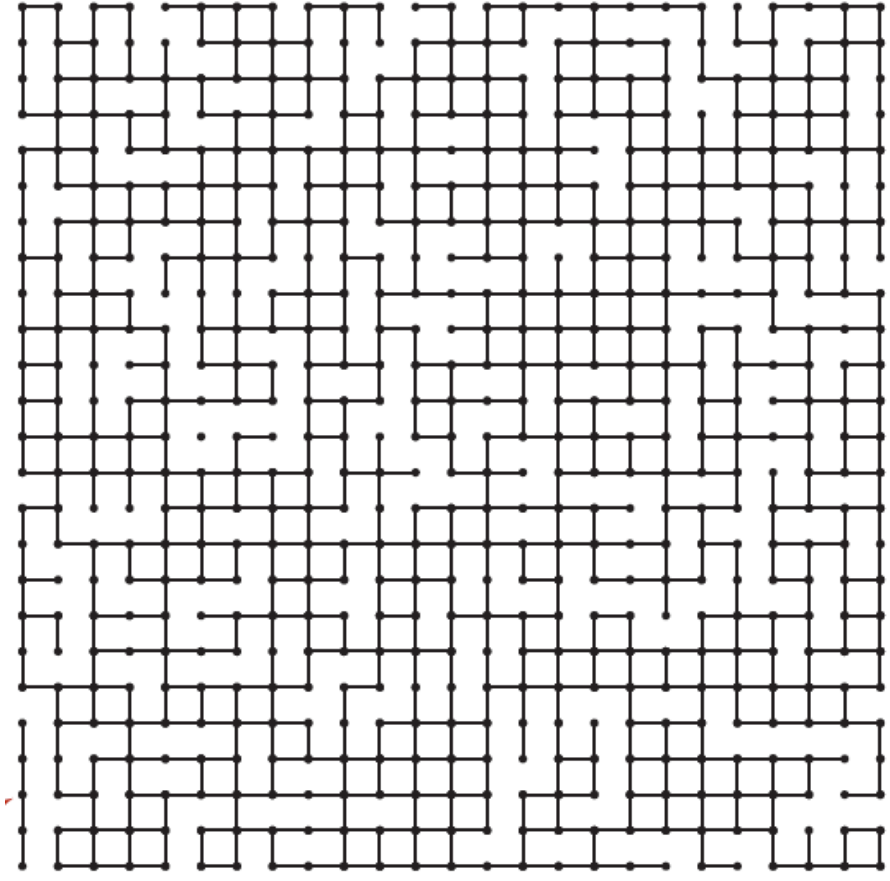
```
union(4, 3)
union(3, 8)
union(6, 5)
union(9, 4)
union(2, 1)
query(0, 7)
query(8, 9)
```

```
union(5, 0)
union(7, 2)
union(6, 1)
union(1, 0)

query(0, 7)
```



O Problema da Conectividade Dinâmica



Para uma grande quantidade de objetos, é necessário um algoritmo eficiente.

O Problema da Conectividade Dinâmica

- Seja a entrada de dados formada por uma sequência de pares de inteiros, na forma `<p, q>`, interpretados como "`p` está conectado a `q`".

- Matematicamente, "estar conectado a" é uma relação de equivalência:
  - Reflexiva: `p` está conectado a `p`.
  - Simétrica: se `p` está conectado a `q`, então `q` está conectado a `p`.
  - Transitiva: se `p` está conectado a `q`, e `q` está conectado a `r`, então `p` está conectado a `r`.

- Uma relação de equivalência divide os objetos em classes de equivalência.

- Dois objetos estão na mesma classe de equivalência se eles estão conectados.

O Problema da Conectividade Dinâmica

- Objetivo do problema: escrever um programa que leia um par `<p, q>` e o imprima somente se `p` e `q` estão conectados.

- Caso os objetos `p` e `q` não estejam conectados, deve-se ignorar este par e realizar a leitura do próximo par disponível na entrada.

- Parece simples, entretanto, este não é um "toy problem".

- É necessário empregar uma estrutura de dados que contenha informações suficientes para determinar se quaisquer dois objetos estão conectados ou não.

Aplicações

- Redes de computadores: os objetos representam os computadores em uma WAN e os pares são as conexões nesta rede.
  - Dados dois computadores p e q, é necessário estabelecer uma conexão direta entre eles ou é possível se comunicar usando conexões existentes?
- Rede elétrica: os objetos representam localidades e os pares as conexões entre elas.
  - Há energia elétrica em todos os locais da rede?
- Redes sociais: os objetos representam pessoas e os pares representam que existe amizade entre elas.
  - Duas pessoas se conhecem? Duas pessoas possuem amigos em comum?

Aplicações

- Equivalência de nomes de variáveis: em algumas linguagens, como Fortran, é possível declarar duas variáveis como sendo equivalentes, isto é, referenciam o mesmo objeto.
  - Após uma sequência de declarações de equivalências, o sistema precisa saber se dois nomes de variáveis são equivalentes.

- Conjuntos matemáticos: os objetos representam elementos de conjuntos individuais.
  - Ao processar um par `<p, q>`, deseja-se saber se eles pertencem ao mesmo conjunto. Se não pertencerem, une-se `p` e `q` em um mesmo conjunto.

- Grafos: pode ser usado para encontrar as componentes conexas de um grafo não direcionado.

Union-Find Disjoint Sets

- A Union-Find Disjoint Sets (UFDS) é uma estrutura de dados que permite modelar uma coleção de conjuntos disjuntos.
  - Disjunto: um mesmo elemento não pertence a dois conjuntos distintos.

- Esta estrutura possui a capacidade de:
  - Determinar se um item pertence a um conjunto.
  - Determinar se dois itens pertencem ao mesmo conjunto.
  - Unir dois conjuntos em um único conjunto.

- Todas as operações são realizadas em tempo quase constante.
  - A complexidade computacional destas operações é ≈ O(1).

Union-Find Disjoint Sets

- A ideia desta estrutura de dados é escolher um elemento representativo para cada conjunto.

- Ao garantir que um conjunto é representado unicamente por um elemento, a tarefa de determinar se dois objetos pertencem ao mesmo conjunto torna-se simples.
  - O elemento representativo (ou pai) pode ser usado como identificador do conjunto.

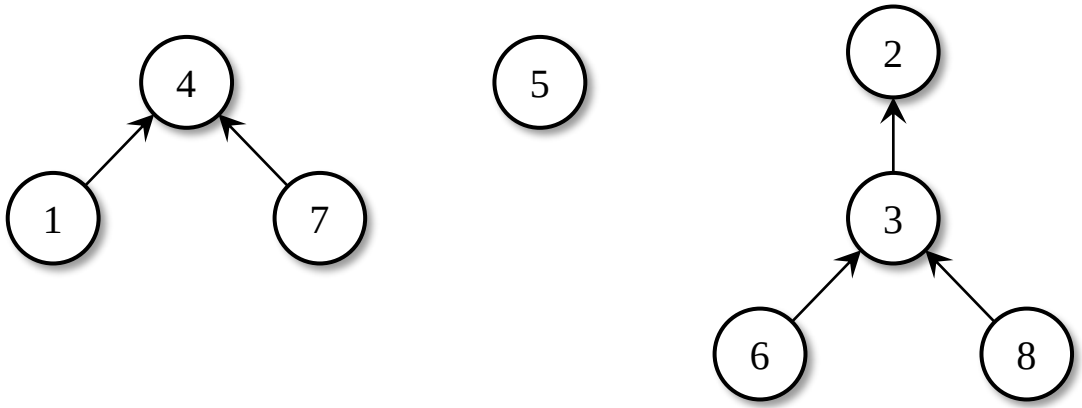
- Para garantir este comportamento, uma Union-Find emprega um modelo conceitual de árvore, onde cada árvore forma um conjunto disjunto.
  - Apenas conceitualmente, pois a implementação pode ser realizada com arrays.

Union-Find Disjoint Sets

- Neste modelo, a raiz de cada árvore é o elemento representativo daquele conjunto.
- Como toda árvore possui uma única raiz, este valor é usado como identificador único do conjunto.
- Assim, para cada elemento que se deseja obter o identificador de seu conjunto, percorre-se a estrutura da árvore até a sua raiz e verifica-se o seu valor representativo.
- Para realizar as operações de forma eficiente, para cada conjunto armazena-se o índice do elemento representativo e a altura de sua árvore.

Union-Find Disjoint Sets

- Exemplo: sejam três conjuntos {1, 4, 7}, {5} e {2, 3, 6, 8}.

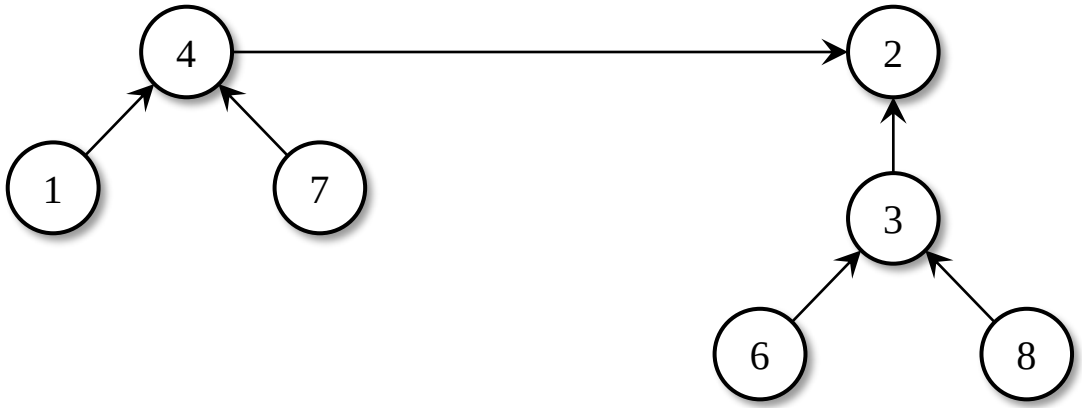


- O elemento 6 tem como representativo o elemento 2, pois pode-se seguir o caminho $6 \rightarrow 3 \rightarrow 2$.
  - Dois elementos pertencem ao mesmo conjunto quando eles possuem o mesmo elemento representativo.

Union-Find Disjoint Sets

- Exemplo: sejam três conjuntos {1, 4, 7}, {5} e {2, 3, 6, 8}.

- Para se juntar dois conjuntos, o representativo de um é conectado ao representativo de outro.



Union-Find Disjoint Sets

- Exemplo: sejam três conjuntos {1, 4, 7}, {5} e {2, 3, 6, 8}.

- Para se juntar dois conjuntos, o representativo de um é conectado ao representativo de outro.

- A eficiência da estrutura Union-Find depende da forma na qual os conjuntos são unidos. Deve-se sempre conectar o representativo do menor conjunto ao representativo do maior conjunto.
  - Esta estratégia garante que o comprimento de qualquer caminho seja proporcional a $O(\log n)$, o que permite encontrar um representativo de forma eficiente percorrendo o caminho.

Union-Find Disjoint Sets

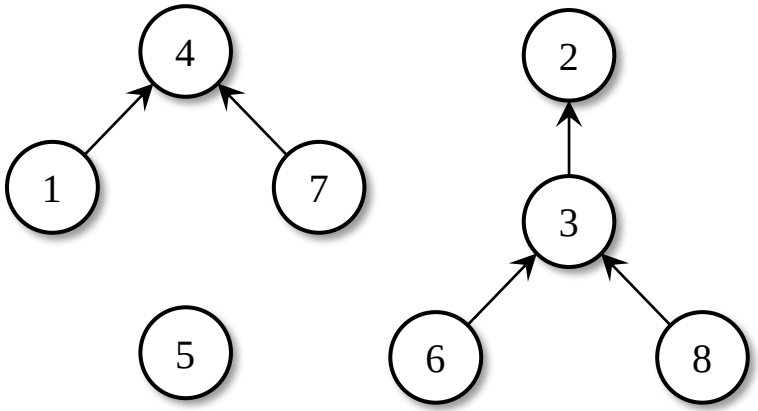
- A estrutura Union-Find pode ser implementada usando apenas arrays.

- O primeiro array, denotado por `link`, indica, para cada elemento, o próximo elemento do seu caminho até o representativo.
  - Ou o próprio elemento, caso ele seja o representativo.

Union-Find Disjoint Sets

- A estrutura Union-Find pode ser implementada usando apenas arrays.

- O primeiro array, denotado por `link`, indica, para cada elemento, o próximo elemento do seu caminho até o representativo.
  - Ou o próprio elemento, caso ele seja o representativo.



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
	4	2	2	4	5	3	4	3

Union-Find Disjoint Sets

- A estrutura Union-Find pode ser implementada usando apenas arrays.

- O primeiro array, denotado por `link`, indica, para cada elemento, o próximo elemento do seu caminho até o representativo.
  - Ou o próprio elemento, caso ele seja o representativo.

- O segundo array, denotado por `size`, indica, para cada representativo, o tamanho do conjunto correspondente.

Union-Find Disjoint Sets

- Exemplo:

	link	size
[0]		
[1]		
[2]		
[3]		
[4]		
[5]		
[6]		
[7]		
[8]		

No início, temos 8 elementos. Cada elemento isolado é um conjunto.

Union-Find Disjoint Sets

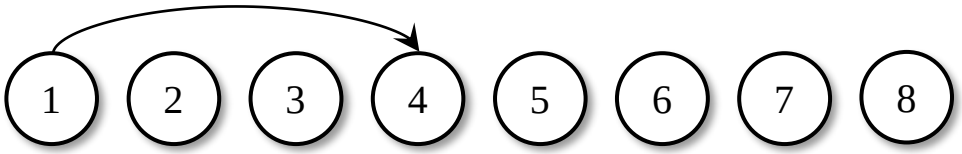
- Exemplo:

	link	size
[0]		
[1]	1	1
[2]	2	1
[3]	3	1
[4]	4	1
[5]	5	1
[6]	6	1
[7]	7	1
[8]	8	1

No início, temos 8 elementos. Cada elemento isolado é um conjunto.
Cada elemento é o seu próprio representativo e cada conjunto tem tamanho 1.

Union-Find Disjoint Sets

- Exemplo:

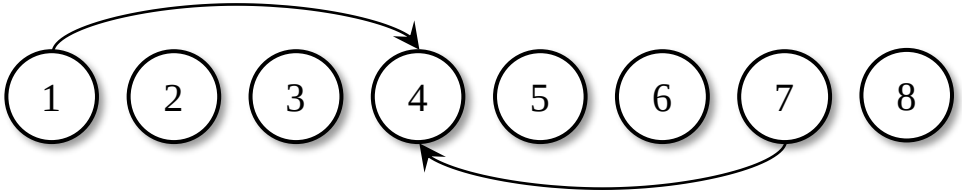


	link	size
[0]		
[1]	4	1
[2]	2	1
[3]	3	1
[4]	4	2
[5]	5	1
[6]	6	1
[7]	7	1
[8]	8	1

União de 4 e 1

Union-Find Disjoint Sets

- Exemplo:

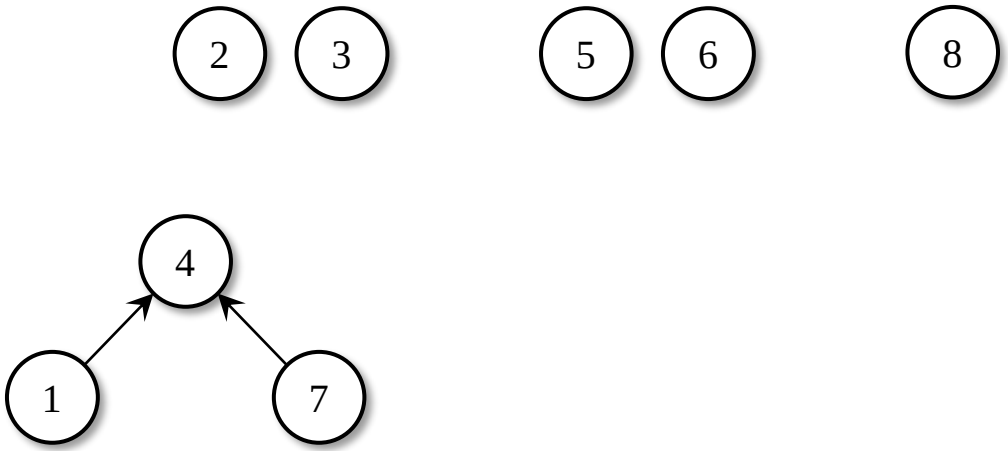


	link	size
[0]		
[1]	4	1
[2]	2	1
[3]	3	1
[4]	4	3
[5]	5	1
[6]	6	1
[7]	4	1
[8]	8	1

União de 7 e 4

Union-Find Disjoint Sets

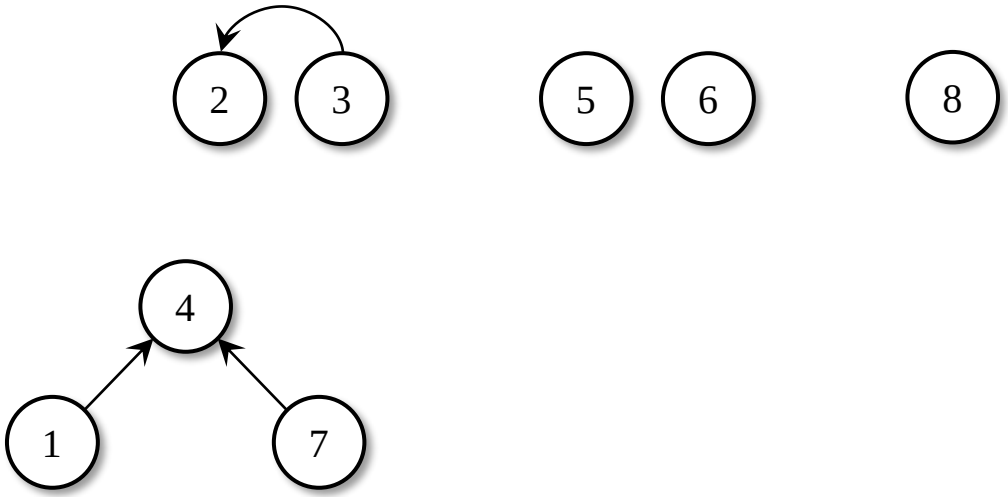
- Exemplo:



	link	size
[0]		
[1]	4	1
[2]	2	1
[3]	3	1
[4]	4	3
[5]	5	1
[6]	6	1
[7]	4	1
[8]	8	1

Union-Find Disjoint Sets

- Exemplo:

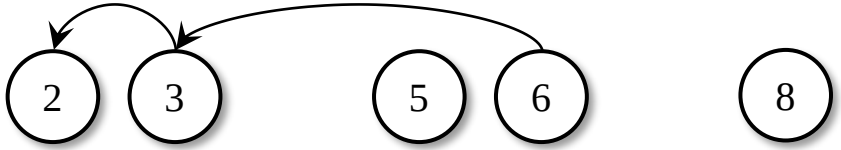


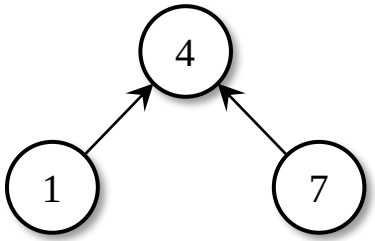
União de 2 e 3

	link	size
[0]		
[1]	4	1
[2]	2	2
[3]	2	1
[4]	4	3
[5]	5	1
[6]	6	1
[7]	4	1
[8]	8	1

Union-Find Disjoint Sets

- Exemplo:



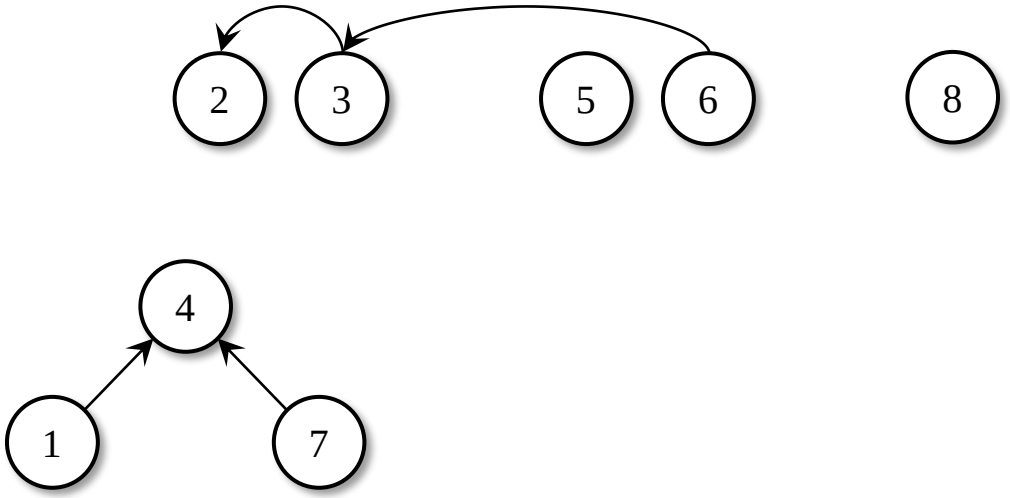


União de 6 e 3

	link	size
[0]		
[1]	4	1
[2]	2	3
[3]	2	1
[4]	4	3
[5]	5	1
[6]	3	1
[7]	4	1
[8]	8	1

Union-Find Disjoint Sets

- Exemplo:



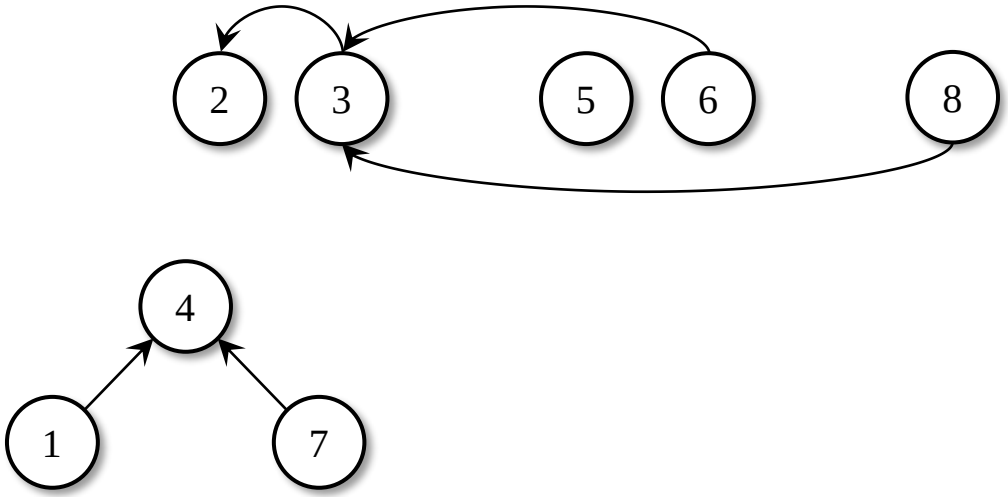
	link	size
[0]		
[1]	4	1
[2]	2	3
[3]	2	1
[4]	4	3
[5]	5	1
[6]	3	1
[7]	4	1
[8]	8	1

União de 6 e 3

3 está em um conjunto maior. O size[2] que aumenta, pois o 2 é o representativo.

Union-Find Disjoint Sets

- Exemplo:



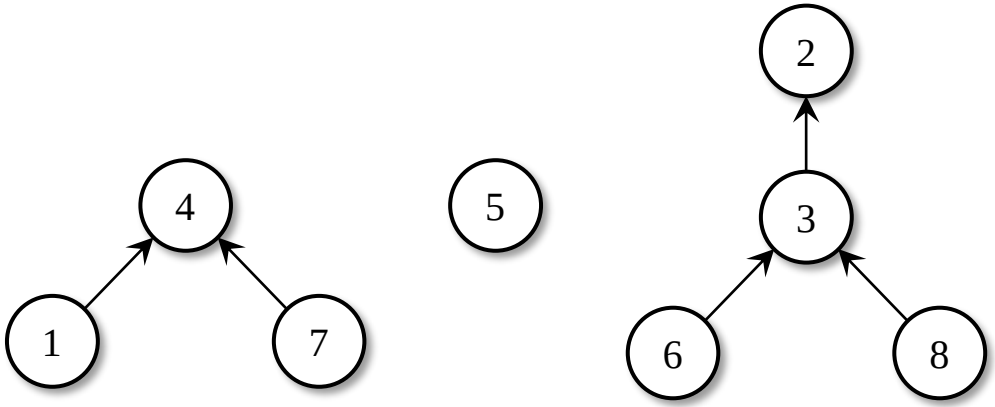
	link	size
[0]		
[1]	4	1
[2]	2	4
[3]	2	1
[4]	4	3
[5]	5	1
[6]	3	1
[7]	4	1
[8]	3	1

União de 8 e 3

3 está em um conjunto maior. O size[2] que aumenta, pois o 2 é o representativo.

Union-Find Disjoint Sets

- Exemplo:



	link	size
[0]		
[1]	4	1
[2]	2	4
[3]	2	1
[4]	4	3
[5]	5	1
[6]	3	1
[7]	4	1
[8]	3	1

Implementação em C++

```
1.  // variaveis
2.  int link[N];
3.  int size[N];
4.
5.  // inicializacao
6.  for(int i = 0; i < N; i++) link[i] = i; // i
7.  for(int i = 0; i < N; i++) size[i] = 1; // um!
8.
9.  // A funcao find() recebe um elemento X e retorna o representativo de X.
10. int find(int x)
11. {
12.     while(x != link[x]) x = link[x];
13.     return x;
14. }
15.
```

Implementação em C++

```
16. // A funcao same() verifica se os elementos A e B pertencem ao mesmo conjunto.
17. int same(int a, int b)
18. {
19.     return find(a) == find(b);
20. }
21.
22. // A funcao unite() mescla os conjuntos que contem os elementos A e B
23. void unite(int a, int b)
24. {
25.     a = find(a);
26.     b = find(b);
27.
28.     if(a == b) return; // A e B ja estao no mesmo conjunto
29.
30.     if(size[a] < size[b]) std::swap(a, b);
31.
32.     size[a] += size[b];
33.     link[b] = a;
34. }
```

Implementação em C++

- A função `find()` pode ser mais eficientemente implementada da seguinte forma:

```
1.  // A funcao find() recebe um elemento X e retorna o representativo de X.
2.  int find(int x)
3.  {
4.      return (x == link[x]) ? x : (link[x] = find(link[x]));
5.  }
```

Implementação em C++

- A função `find()` pode ser mais eficientemente implementada da seguinte forma:

```
1.  // A funcao find() recebe um elemento X e retorna o representativo de X.
2.  int find(int x)
3.  {
4.      return (x == link[x]) ? x : (link[x] = find(link[x]));
5.  }
```

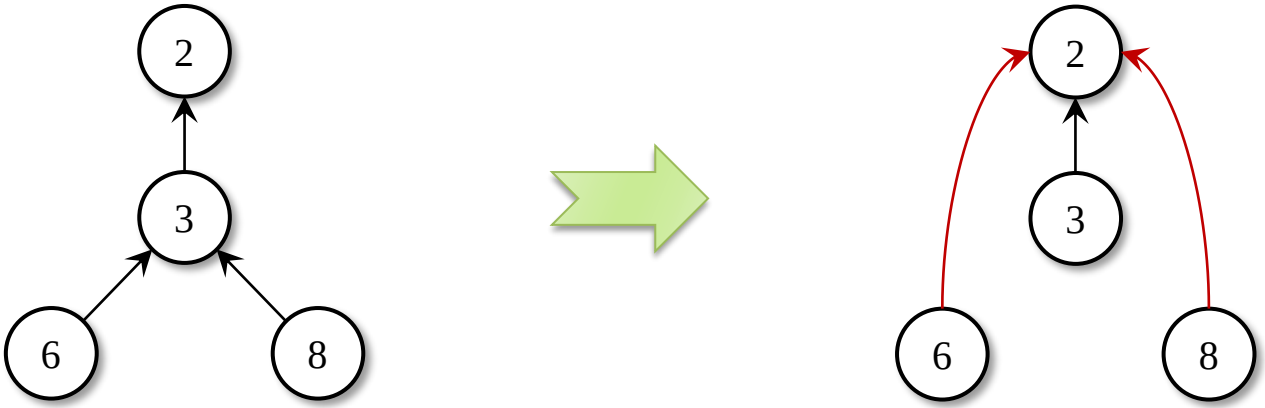
- Nesta modificação, cada elemento passa a apontar diretamente para o seu representativo.

- Essa modificação é conhecida como path compression e faz com que o tempo de execução torne-se próximo de O(1).

Implementação em C++

- A função `find()` pode ser mais eficientemente implementada da seguinte forma:

```
1.  // A funcao find() recebe um elemento X e retorna o representativo de X.
2.  int find(int x)
3.  {
4.      return (x == link[x]) ? x : (link[x] = find(link[x]));
5.  }
```



Dúvidas?

Aula 12:
Union-Find Disjoint Sets

***Disciplina*: Maratona de Programação 1**

Profs. Edmilson Marmo e Luiz Olmes

edmarmo@unifei.edu.br, olmes@unifei.edu.br

UNIFEI
UNIVERSIDADE FEDERAL DE ITAJUBÁ