Universidade de Brasília
Departamento de Ciência da Computação

Union-find Disjoint Sets
CIC0258 - Tópicos Especiais em Programação Competitiva

Prof. Dr. Vinícius Ruela Pereira Borges

viniciusrpb@unb.br

Brasília-DF, 2022

Informação

- Esses slides foram redigidos e produzidos pelo Prof. Dr. Vinícius R. P. Borges;
- Material didático de referência:

  Halim S., Halim F., *Competitive Programming 3: The New Lower Bound of Programming Contests*, 3ª ed, 2018.
  Laaksonen A., Competitive Programmer's Handbook, disponível online, 2018.

  - Textos do repositório do Grupo UnBalloon [1]
  - Exercícios da plataforma Codeforces

[1] `https://github.com/UnBalloon`

Roteiro

- Contextualização
- Definição
- Implementação Naive
- Implementação Otimizada

Contextualização

Contextualização

- Precisamos modelar o seguinte problema:
- Temos N elementos e, inicialmente, cada elemento representa um conjunto matemático:

$$\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$$

- Aqui, os conjuntos matemáticos possuem um "representante", que é exatamente um único elemento - dentre os N apresentados.

Contextualização

$$\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$$

- Suponha agora que temos Q consultas em que podemos (em qualquer ordem):
  - Verificar se dois elementos estão em um mesmo conjunto;
  - **Unir** dois conjuntos.

Contextualização

$$\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$$

- Para $Q = 4$ consultas, temos:
  1. Unir os conjuntos que possuem os elementos 3 e 4;
  2. Verificar se os elementos 1 e 2 pertencem ao mesmo conjunto;
  3. Unir os conjuntos que possuem os elementos 4 e 5;
  4. Verificar se os elementos 3 e 5 pertencem ao mesmo conjunto.

Contextualização

$$\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$$

1. Unir os conjuntos que possuem os elementos 3 e 4;

Contextualização

$$\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$$

1. Unir os conjuntos que possuem os elementos 3 e 4;

$$\{1\}, \{2\}, \{3, 4\}, \{5\}$$

2. Verificar se os elementos 1 e 2 pertencem ao mesmo conjunto;

Contextualização

$$\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$$

1. Unir os conjuntos que possuem os elementos 3 e 4;

$$\{1\}, \{2\}, \{3, 4\}, \{5\}$$

2. Verificar se os elementos 1 e 2 pertencem ao mesmo conjunto: **Não**.
3. Unir os conjuntos que possuem os elementos 4 e 5;

Contextualização

$$\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$$

1. Unir os conjuntos que possuem os elementos 3 e 4;

$$\{1\}, \{2\}, \{3, 4\}, \{5\}$$

2. Verificar se os elementos 1 e 2 pertencem ao mesmo conjunto: **Não**.
3. Unir os conjuntos que possuem os elementos 4 e 5;

$$\{1\}, \{2\}, \{3, 4, 5\}$$

4. Verificar se os elementos 3 e 5 pertencem ao mesmo conjunto;

$$\{1\},\{2\},\{3\},\{4\},\{5\}$$

1. Unir os conjuntos que possuem os elementos 3 e 4;

$$\{1\},\{2\},\{3,4\},\{5\}$$

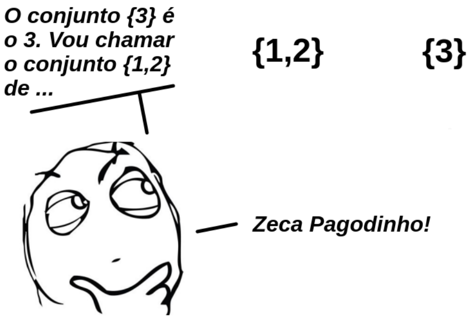
2. Verificar se os elementos 1 e 2 pertencem ao mesmo conjunto: **Não**.
3. Unir os conjuntos que possuem os elementos 4 e 5;

$$\{1\},\{2\},\{3,4,5\}$$

4. Verificar se os elementos 3 e 5 pertencem ao mesmo conjunto: **Sim**.

Contextualização

- Como representar os conjuntos?



Contextualização

- Como representar os conjuntos?
- Qual a melhor estrutura de dados para representar os conjuntos?
- Como consultar e unir os conjuntos?

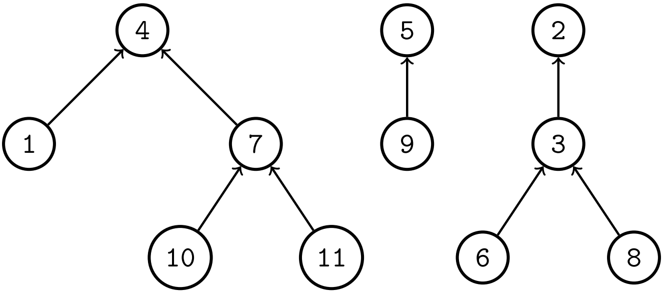
Definição

Definição

- *Disjoint Set Union* (DSU), ou *Union-find disjoint set* (UFDS), é uma estrutura de dados que mantém uma coleção de conjuntos $\{S_1, S_2, \ldots, S_N\}$ disjuntos, isto é, $S_i \cap S_j = \emptyset$ se $i \neq j$;

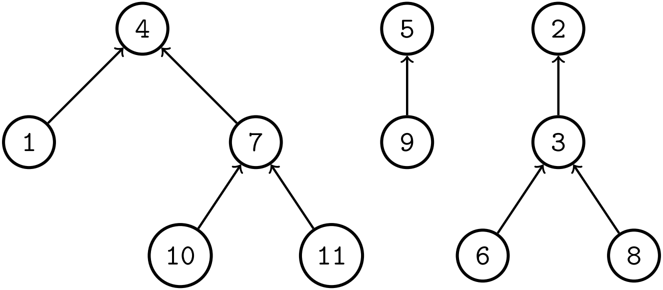
- Há duas operações básicas, com complexidade $O(\log N)$ na versão otimizada:
    - a união entre dois conjuntos disjuntos, e;
    - a identificação do representante da união de conjuntos que um conjunto pertence.

- DSU se baseia em uma floresta de árvores;



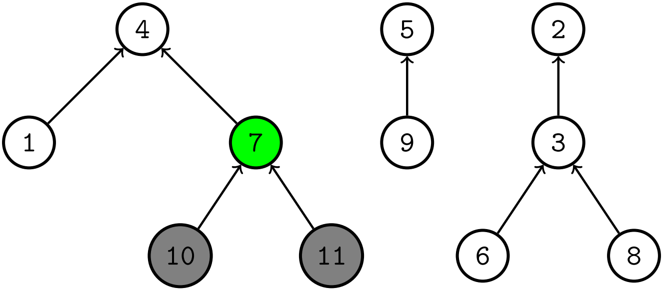
Formalismo

- Cada árvore representa uma união de subconjuntos;



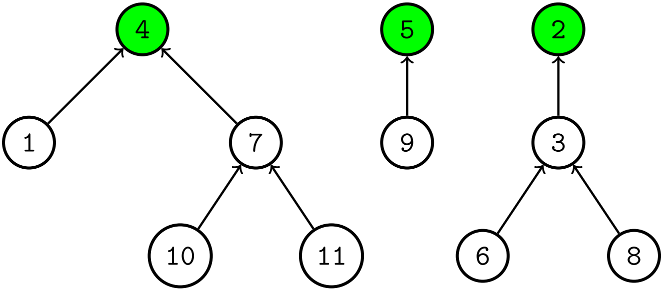
Formalismo

- Cada árvore representa uma união de subconjuntos;
- Por exemplo: $\{10, 11\}$ é um subconjunto de $\{7, 10, 11\}$.



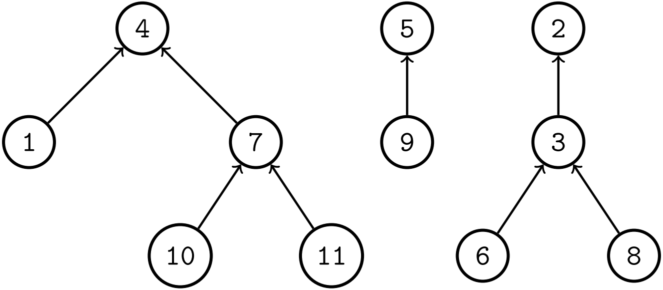
Formalismo

- A raiz de cada árvore é o representante da união de subconjuntos;



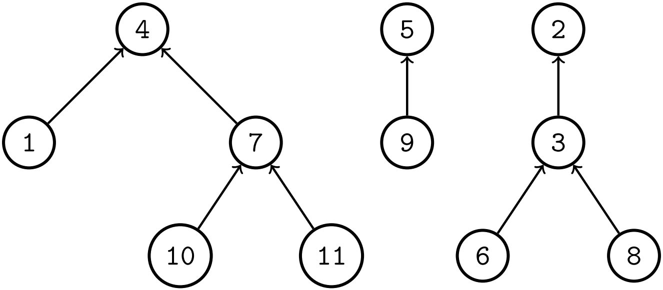
Formalismo

- Cada nó da árvore representa um dos conjuntos que compõem a união;



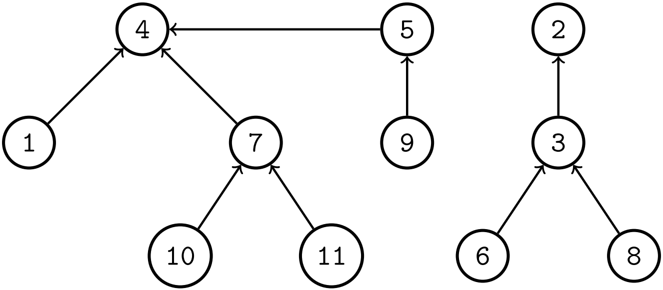
Formalismo

- Conjuntos representados por árvores distintas são disjuntos;
- Assim temos: $\{4, 1, 7, 10, 11\}$, $\{5, 9\}$ e $\{2, 3, 6, 8\}$.



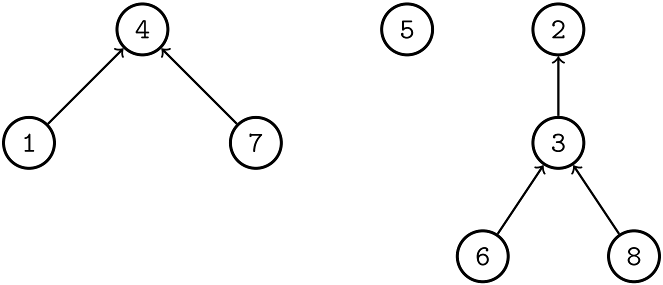
Formalismo

- Duas árvores podem ser unidas tornando a raiz de uma delas filha da raiz da outra;



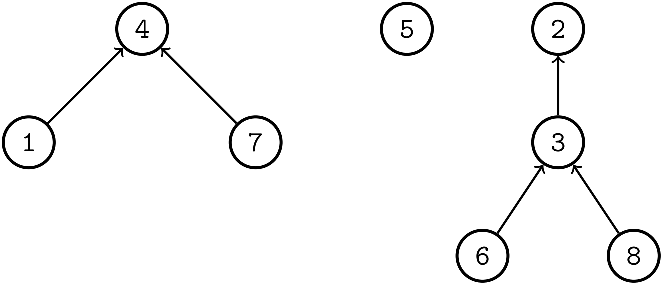
Exemplo

- Sejam os conjuntos $\{1, 4, 7\}$, $\{5\}$ e $\{2, 3, 6, 8\}$:



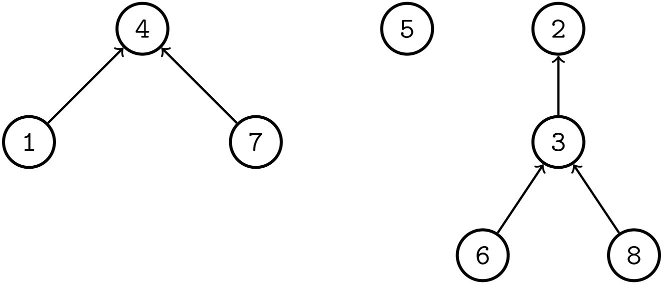
Exemplo

- Os elementos $\{4\}$, $\{5\}$ e $\{2\}$ são os representantes dos três conjuntos:



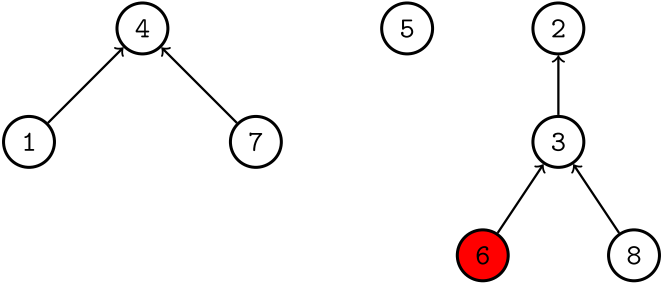
Exemplo

- Podemos encontrar os conjuntos e seus representantes para cada elemento seguindo uma cadeia que começa no elemento representante do conjunto:



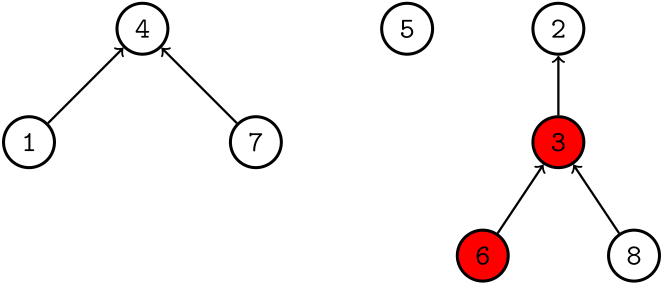
Exemplo

- 2 é o representante do elemento 6 conforme a cadeia 6 $\rightarrow$...



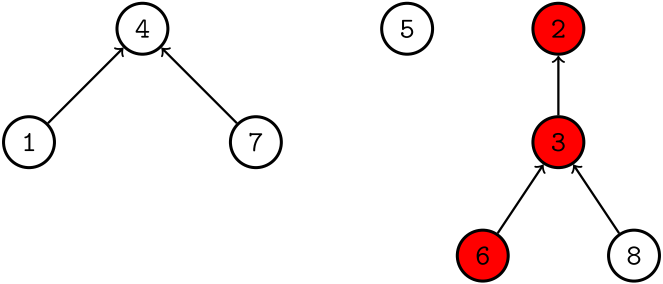
Exemplo

- 2 é o representante do elemento 6 conforme a cadeia $6 \rightarrow 3 \rightarrow$...



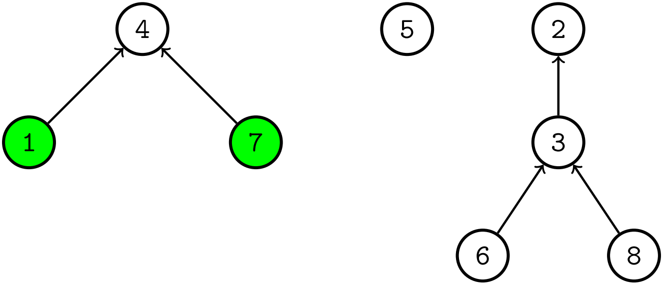
Exemplo

- 2 é o representante do elemento 6 conforme a cadeia $6 \to 3 \to 2$.



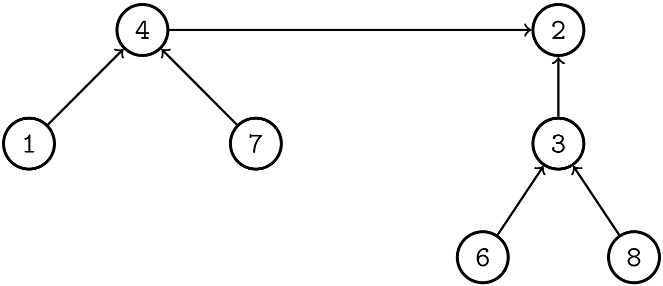
Exemplo

- Dois elementos pertencem exatamente ao mesmo conjunto quando seus representantes são os mesmos.



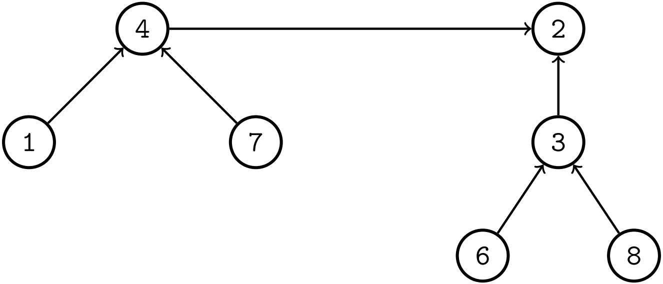
Exemplo

- Dois conjuntos A e B podem ser unidos ao conectar o representante do conjunto A ao representante do conjunto B.



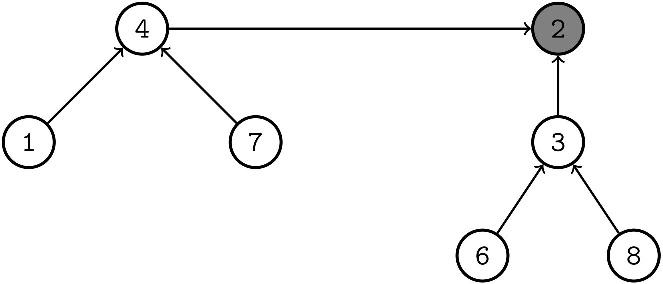
Exemplo

- Por exemplo, os conjuntos $\{1, 4, 7\}$ and $\{2, 3, 6, 8\}$ podem ser unidos como:



Exemplo

- Assim, o conjunto resultante contém os elementos $\{1, 2, 3, 4, 6, 7, 8\}$;
- O elemento 2 é o representante de todo o conjunto e o ex-representante 4 passa a ter o 2 como representante.



Definição

- A eficiência da DSU depende de como os conjuntos são unidos;
- Isso nos leva a adotar uma estratégia simples:
  - conectar o representante do menor conjunto ao representante do maior conjunto;
- Veja que essa cadeia pode ser vista como uma árvore e, para percorrê-la, podemos fazer isso em tempo $O(\log N)$.

Implementação Tradicional

pc programação competitiva

Implementação Tradicional

- Essa implementação considera o uso de vetores para permitir o caminhamento desde os nodos folhas até a raiz da árvore;
- No vetor, cada conjunto é representado por um inteiro de 1 a N;
- **Vetor `parent`**: para cada elemento, armazena o próximo elemento na cadeia ou ele mesmo, caso represente um conjunto.

Implementação Tradicional

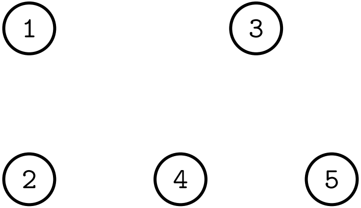
- Inicialmente, cada elemento pertence a um único conjunto:

```
vector<int> parent(n+1);

/* ... */

for (int i = 1; i <= n; i++)
    parent[i] = i;

/* ... */
```

- Para facilitar a implementação, o representante do conjunto é o próprio número.

Exemplo

- Sejam os conjuntos $\{1\}, \{2\}, \{3\}, \{4\}, \{5\}$:



elem	1	2	3	4	5
parent	1	2	3	4	5

Implementação Tradicional

- A função `find_set` retorna o representante do conjunto que contém o elemento x;
- Tal conjunto pode ser encontrado percorrendo-se a cadeia que começa em x até encontrar o representante que é ele mesmo.

```
int find_set(int x)
{
    while(x != parent[x])
        x = parent[x];

    return x;
}
```

Implementação Tradicional

- Versão recursiva da função `find_set`...

```
int find_set(int x)
{
    if(parent[x] == x)
        return x;

    return find_set(parent[x]);
}
```

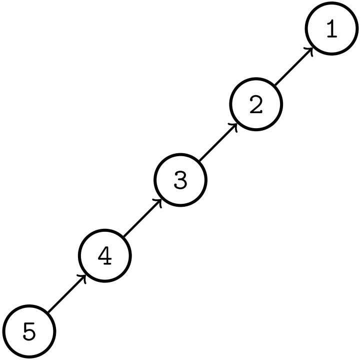
Implementação Tradicional

- A função `same_set` verifica se os elementos a e b pertencem ao mesmo conjunto:

```
bool same_set(int a, int b)
{
    return find_set(a) == find_set(b);
}
```

Implementação Tradicional

- Problema: o pior caso!



elem	1	2	3	4	5
parent	1	1	2	3	4

Implementação Tradicional: complexidade

- No pior caso, quando o conjunto é formado por uma árvore longa, temos que percorrer todas as posições do vetor `parent`;
- Assim, a complexidade da função `find_set` é $O(N)$;
- Consequentemente, as funções `same_set` e `join_sets` também possuirão complexidade $O(N)$.

Implementação Otimizada

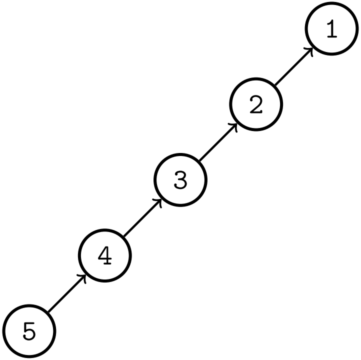
pc programação competitiva

Implementação Otimizada

- A eficiência da estrutura *union-find* depende de como os conjuntos são unidos;
- Assim, a complexidade da função `find_set` é $O(N)$;
- Consequentemente, as funções `same_set` e `join_sets` também possuirão complexidade $O(N)$.

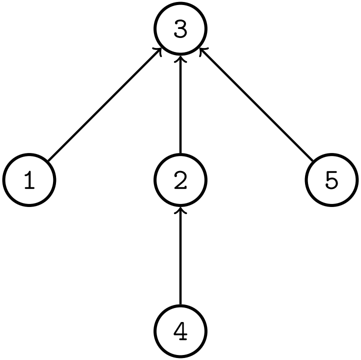
Exemplo: Implementação Otimizada

- Revendo o pior cenário...



Exemplo: Implementação Otimizada

- Com a árvore na organização dessa maneira, menos profunda e mais esparsada, não é necessário passar por todos os nodos ao percorrer da folha até o nodo representante.



Implementação Otimizada

- Como obter essa representação?
- Podemos seguir uma simples estratégia: sempre conectar o representante do menor conjunto com o representante do maior conjunto;
  - caso os conjuntos possuam o mesmo tamanho, pode-se fazer uma escolha arbitrária.
- Por meio dessa estratégia, o comprimento máximo do ramo mais profundo da árvore será $O(\log N)$.

Implementação Otimizada

- Para sabermos o tamanho de cada conjunto, dado pela quantidade de nodos de cada árvore, vamos considerar um novo vetor `card`:

```
vector<int> card(n+1);

/* ... */

for (int i = 1; i <= n; i++)
    card[i] = 1;

/* ... */
```

- Inicialmente, cada conjunto é composto por um único elemento.

- A função **`join_sets`** pode ser então alterada como se segue:

```
void join_sets(int a, int b)
{
    a = find_set(a);
    b = find_set(b);

    /* ... */

}
```

Implementação Otimizada

- Suponha que o conjunto representado por a seja o maior em tamanho;
- Assim, a ideia é unir os dois conjuntos, em que a será o representante.

```
void join_sets(int a, int b)
{
    a = find_set(a);
    b = find_set(b);

    /* ... */

    card[a] += card[b];
}
```

Implementação Otimizada

- Assim, basta incrementar a cardinalidade do conjunto representado por a com a cardinalidade do conjunto representado por b.

```
void join_sets(int a, int b)
{
    a = find_set(a);
    b = find_set(b);

    /* ... */

    card[a] += card[b];
}
```

Implementação Otimizada

- Por fim, o representante do conjunto que contém b é agora o representante do conjunto dado por a.

```
void join_sets(int a, int b)
{
    a = find_set(a);
    b = find_set(b);

    /* ... */

    card[a] += card[b];
    parent[b] = a;
}
```

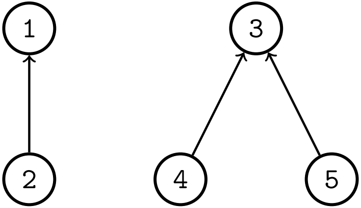
Implementação Otimizada

- Agora repare na padronização para fazer com que o conjunto a seja sempre o maior;

```
void join_sets(int a, int b)
{
    a = find_set(a);
    b = find_set(b);

    if(card[a] < card[b])
        swap(a,b);

    card[a] += card[b];
    parent[b] = a;
}
```

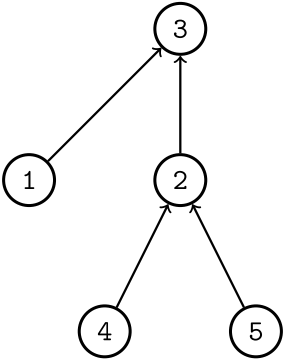
Exemplo

- Para os conjuntos $\{1, 2\}$ e $\{3, 4, 5\}$:



..elem	1	2	3	4	5
parent	1	1	3	3	3
..card	2	1	2	1	1

Exemplo 2



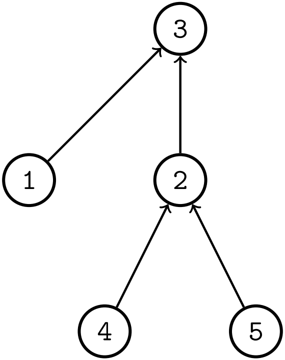
..elem	1	2	3	4	5
parent	3	3	3	2	2
..card	1	5	3	1	1

Implementação Otimizada

- A função **`find_set`** é atualizada para obtermos o representante do conjunto que contém cada elemento apenas na primeira passada;

```
int find_set(int x)
{
    if(x == parent[x])
        return x;

    return parent[x] = find_set(parent[x]);
}
```

Exemplo 2: atualizando...



..elem	1	2	3	4	5
parent	3	3	3	3	3
..card	1	5	3	1	1

Implementação Otimizada: complexidade

- A complexidade da função `find_set` é $O(\log N)$, assumindo que o comprimento de cada cadeia na árvore é $O(\log N)$;

- Assim, as funções `same_set` e `join_sets` também possuem complexidade $O(\log N)$;

- A função `join_sets` garante que o comprimento de cada cadeia é $O(\log N)$ por sempre conectar o menor conjunto ao maior conjunto.

Universidade de Brasília
Departamento de Ciência da Computação

Union-find Disjoint Sets
CIC0258 - Tópicos Especiais em Programação Competitiva

Prof. Dr. Vinícius Ruela Pereira Borges

viniciusrpb@unb.br

Brasília-DF, 2022