Conjuntos disjuntos dinâmicos

CLR 22 CLRS 21

Conjuntos disjuntos

Seja $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ uma coleção de conjuntos disjuntos, ou seja,

$$S_i \cap S_j = \emptyset$$

para todo $i \neq j$.

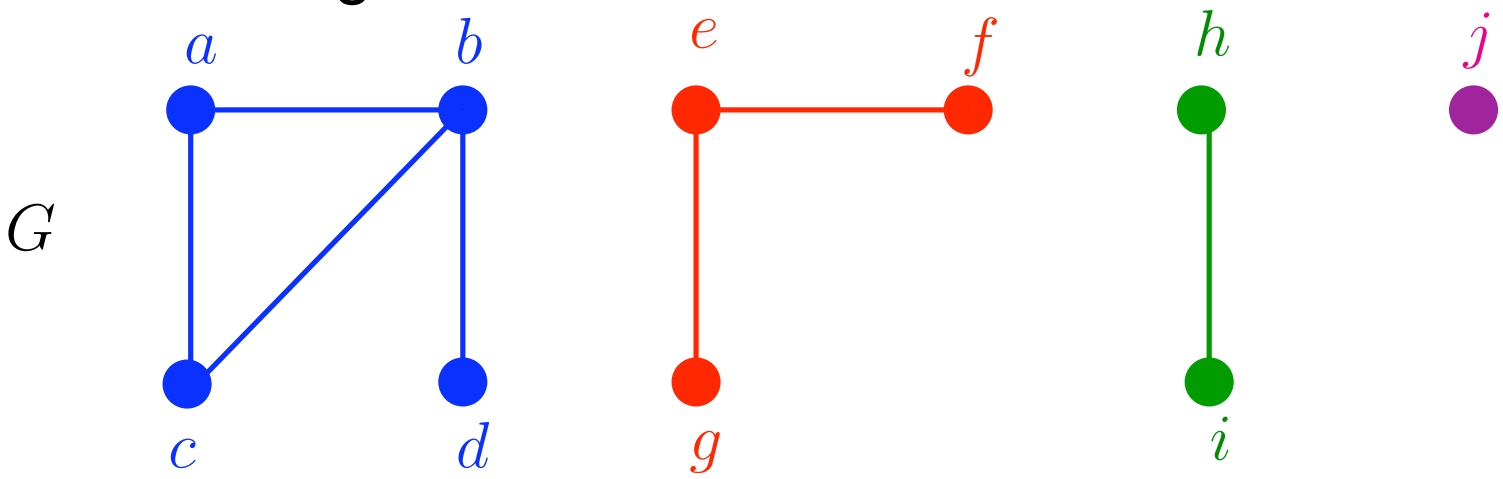
Conjuntos disjuntos

Seja $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ uma coleção de conjuntos disjuntos, ou seja,

$$S_i \cap S_j = \emptyset$$

para todo $i \neq j$.

Exemplo de coleção disjunta de conjuntos: componentes conexos de um grafo



componentes = conjuntos disjuntos de vértices

$\{a, b, c, d\}$ $\{e, f, g\}$ $\{h, i\}$ $\{j\}$

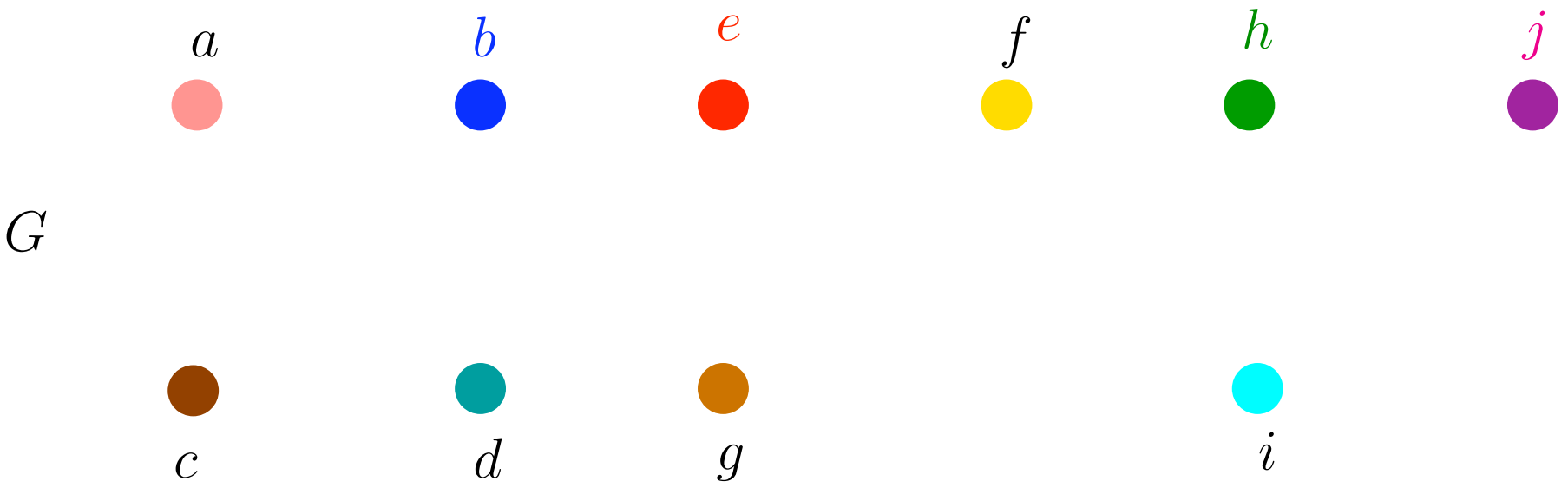
Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

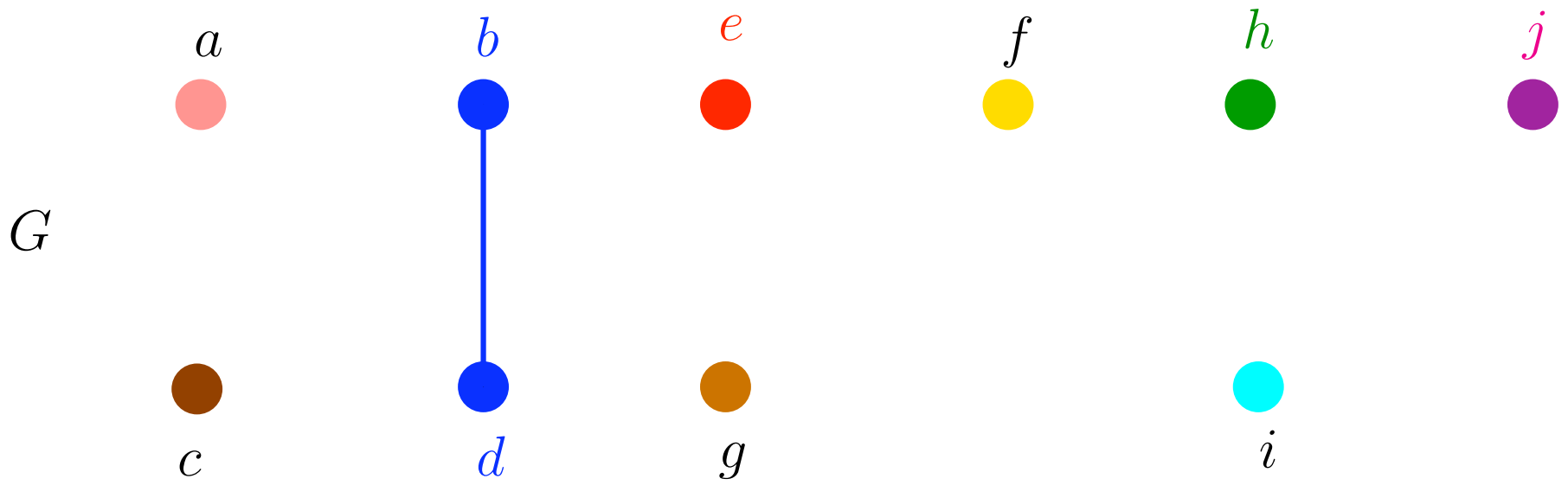


aresta	componentes
	$\{a\}$ $\{b\}$ $\{c\}$ $\{d\}$ $\{e\}$ $\{f\}$ $\{g\}$ $\{h\}$ $\{i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

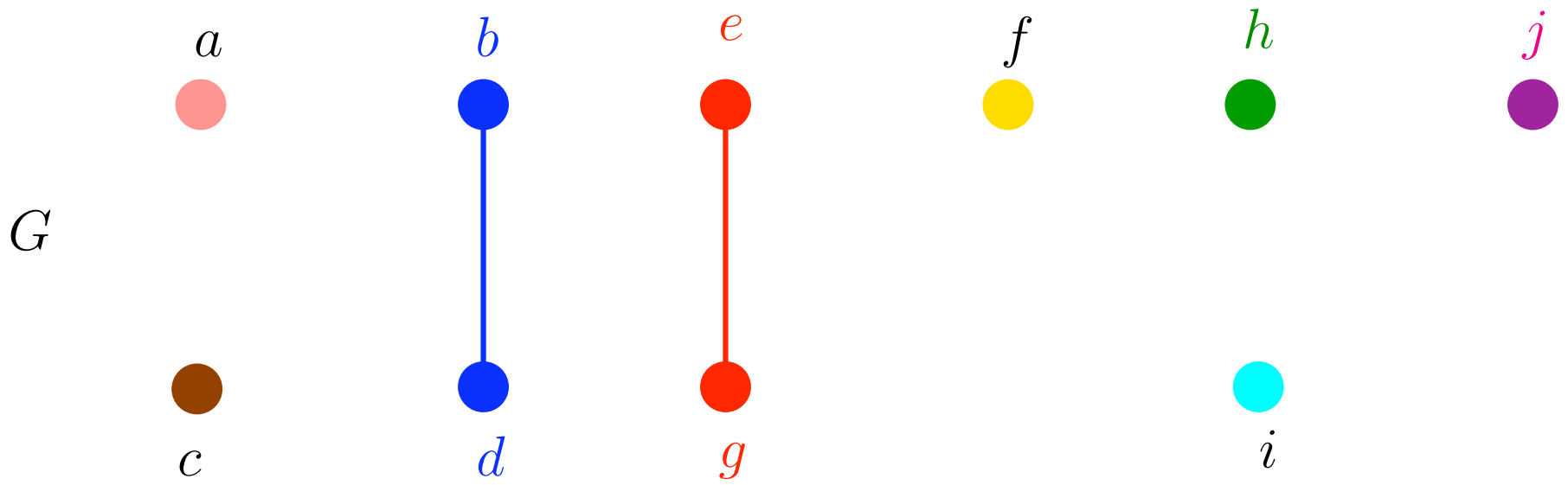


aresta	componentes
(b,d)	$\{a\}\ \{b,d\}\ \{c\}\ \{e\}\ \{f\}\ \{g\}\ \{h\}\ \{i\}\ \{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

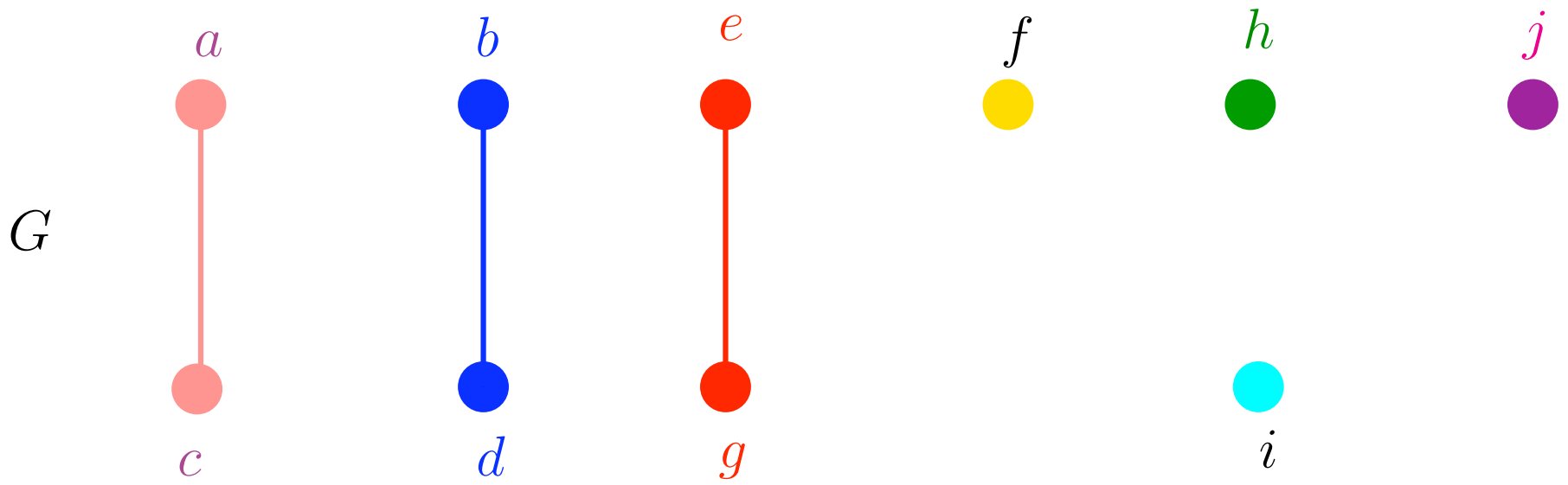


aresta	componentes
(e, g)	$\{a\}\ \{b, d\}\ \{c\}\ \{e, g\}\ \{f\}\ \{h\}\ \{i\}\ \{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

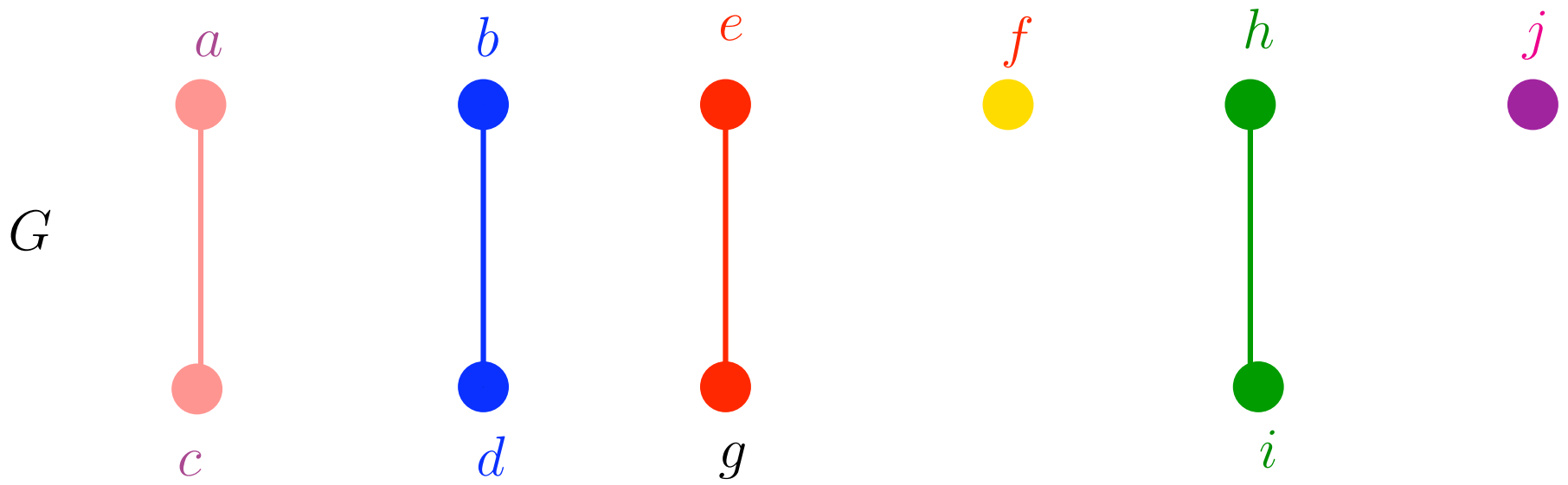


aresta	componentes
(a, c)	$\{a, c\}$ $\{b, d\}$ $\{e, g\}$ $\{f\}$ $\{h\}$ $\{i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

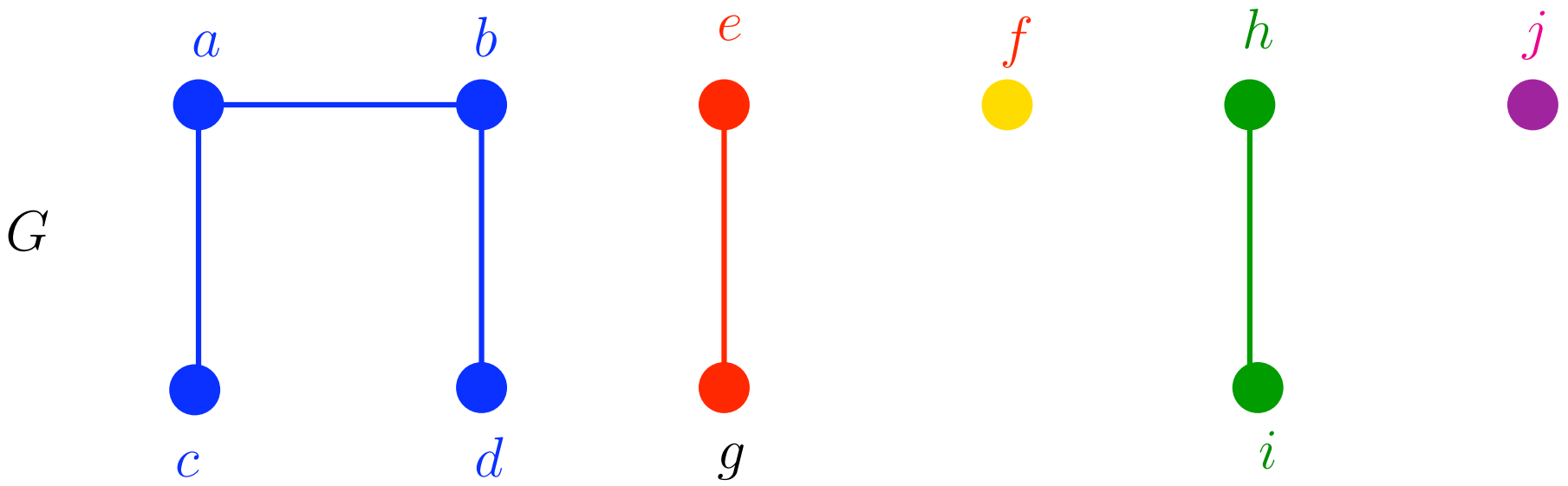


aresta	componentes
(h,i)	$\{a,c\}$ $\{b,d\}$ $\{e,g\}$ $\{f\}$ $\{h,i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico



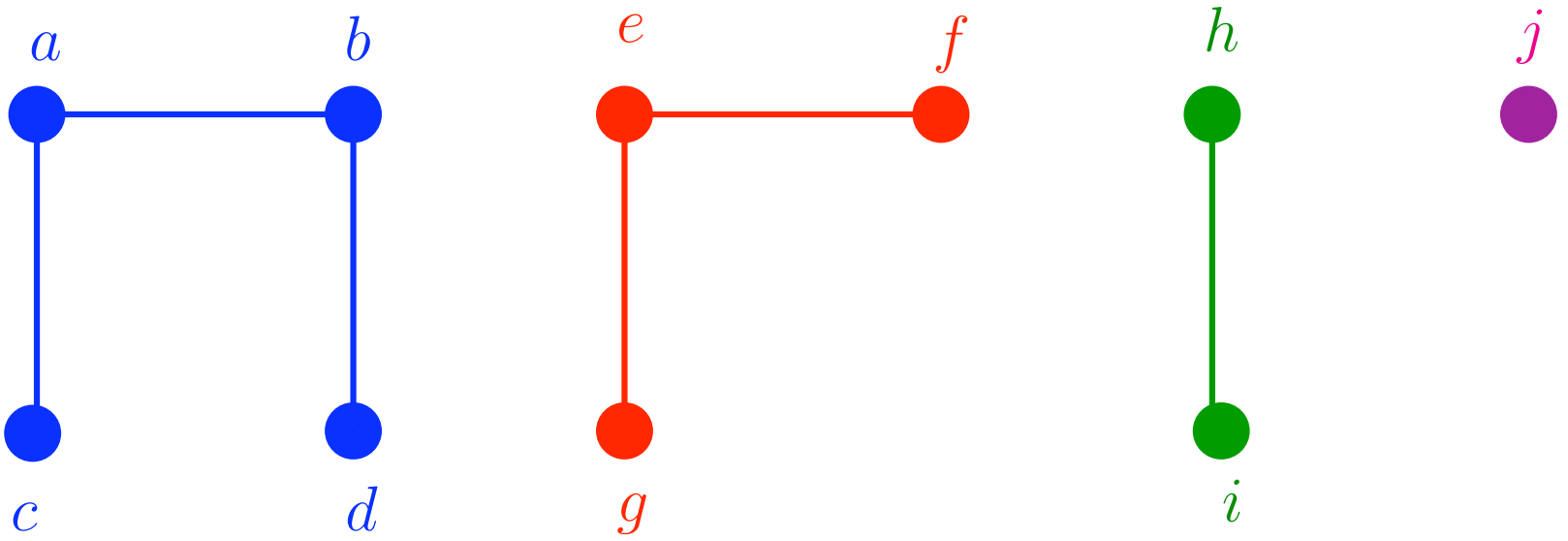
aresta	componentes
(a,b)	$\{a,b,c,d\}$ $\{e,g\}$ $\{f\}$ $\{h,i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

G



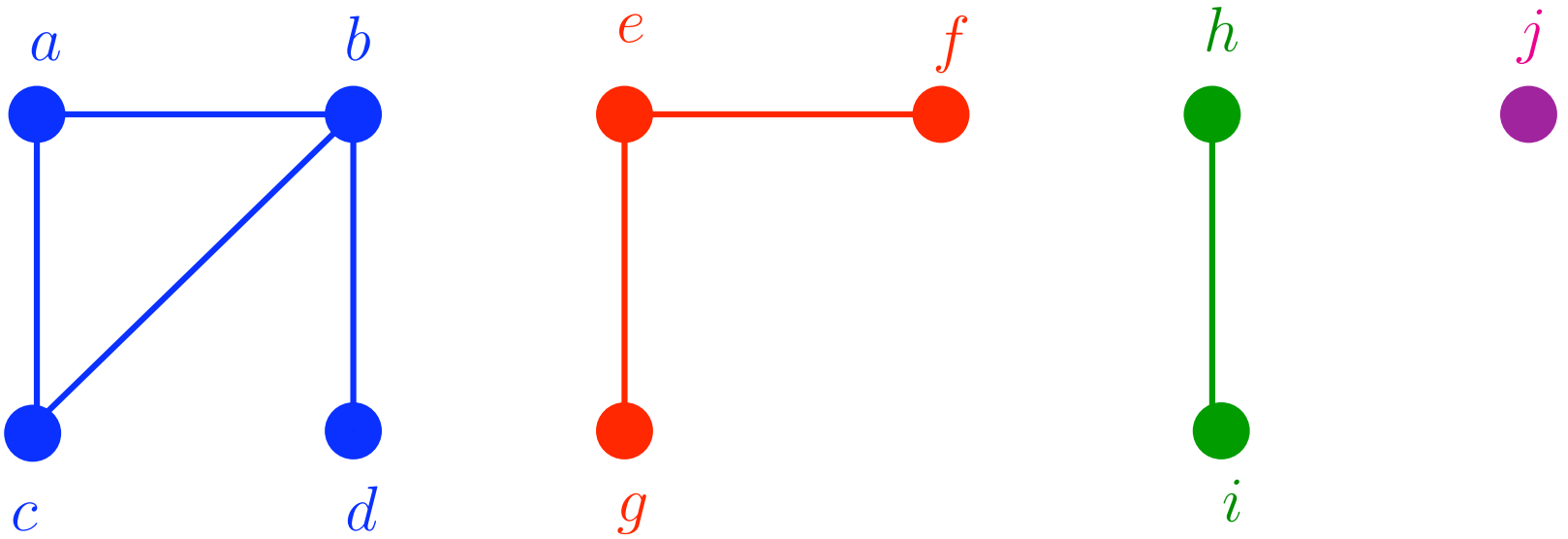
aresta	componentes
(e, f)	$\{a, b, c, d\}$ $\{e, f, g\}$ $\{h, i\}$ $\{j\}$

Coleção disjunta dinâmica

Conjuntos são modificados ao longo do tempo

Exemplo: grafo dinâmico

G



aresta	componentes
(b,c)	$\{a,b,c,d\}$ $\{e,f,g\}$ $\{h,i\}$ $\{j\}$

Operações básicas

$\mathcal{S}$ coleção de conjuntos disjuntos.

Cada conjunto tem um representante.

MAKESET (x): x é elemento novo

$\mathcal{S} \leftarrow \mathcal{S} \cup \{\{x\}\}$

UNION (x, y): x e y em conjuntos diferentes

$\mathcal{S} \leftarrow \mathcal{S} - \{S_x, S_y\} \cup \{S_x \cup S_y\}$

x está em S_x e y está em Sy

FINDSET (x): devolve representante do conjunto que contém x

Connected-Componets

Recebe um grafo G e contrói uma representação dos componentes conexos.

```
CONNECTED-COMPONENTS (G)
1   para cada vértice v de G faça
2       MAKESET (v)
3   para cada aresta (u, v) de G faça
4       se FINDSET (u) ≠ FINDSET (v)
5           então UNION (u, v)
```

Consumo de tempo

$n :=$ número de vértices do grafo
$m :=$ número de arestas do grafo

linha	consumo de todas as execuções da linha
1	$= \Theta(n)$
2	$= n \times$ consumo de tempo MAKESET
3	$= \Theta(m)$
4	$= 2m \times$ consumo de tempo FINDSET
5	$\leq n \times$ consumo de tempo UNION
total	$\leq \Theta(n+m) + n \times$ consumo de tempo MAKESET $+2m \times$ consumo de tempo FINDSET $+n \times$ consumo de tempo UNION

Same-Component

Decide se u e v estão no mesmo componente:

```
SAME-COMPONENT (u, v)
1   se FINDSET (u) = FINDSET (v)
2         então devolva SIM
3         senão devolva NÃO
```

Algoritmo de Kruskal

Encontra uma árvore geradora mínima (CLRS 23).

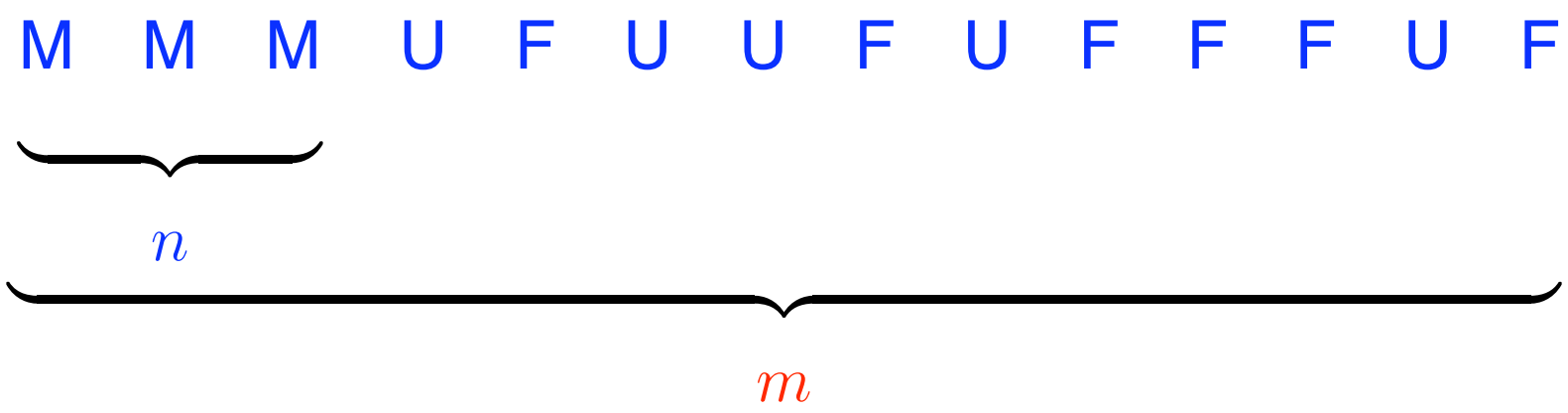
MST-KRUSKAL (G, w) $\quad \triangleright$ G conexo

```
−1  coloque arestas em ordem crescente de w
 0  A ← ∅
 1  para cada vértice v faça
 2        MAKESET (v)
 3  para cada aresta uv em ordem crescente de w faça
 4        se FINDSET (u) ≠ FINDSET (v)
 5              então UNION (u, v)
 8                    A ← A ∪ {uv}
 9  devolva A
```

"Avô" de todos os algoritmos gulosos.

Conjuntos disjuntos dinâmicos

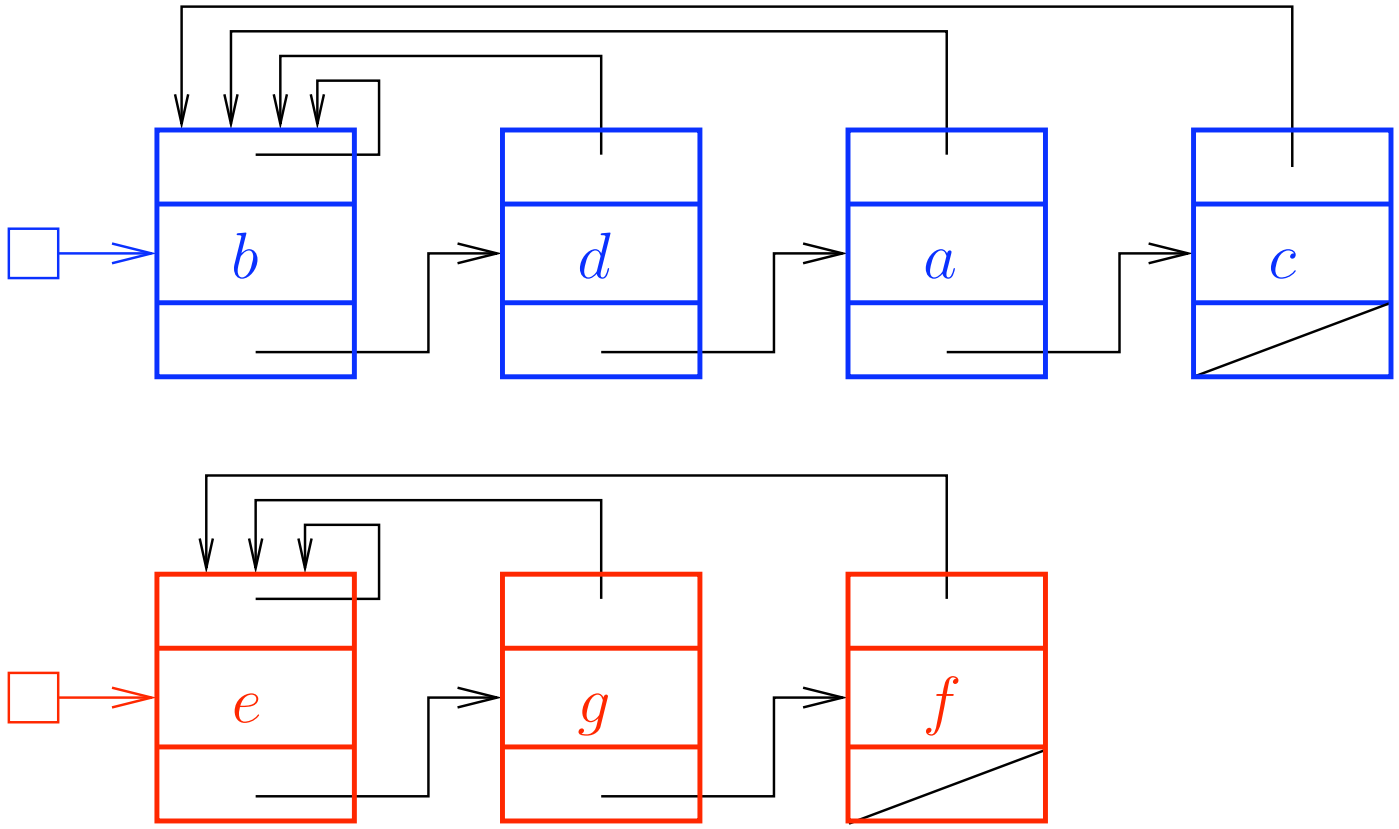
Seqüência de operações MAKESET, UNION, FINDSET



Que estrutura de dados usar?

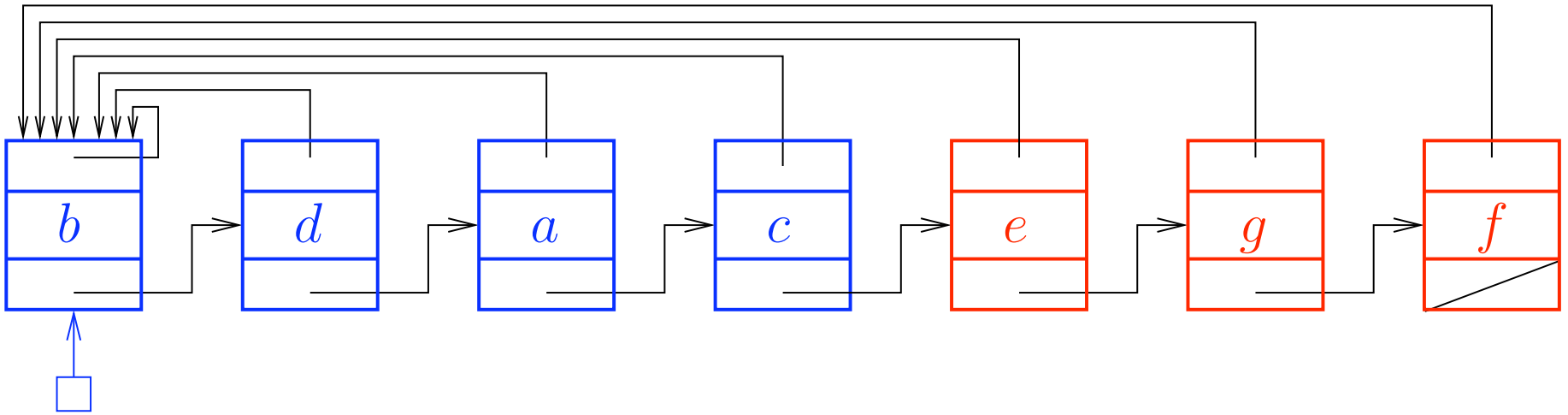
Compromissos (*trade-off*s)

Estrutura de listas ligadas



- cada conjunto tem um representante (início da lista)
- cada nó x tem um campo $repr$
- $repr[x]$ é o representante do conjunto que contém x

Estrutura de listas ligadas



UNION (a, e) : atualiza apontador para o representante.

Consumo de tempo

Operação	número de objetos atualizados
MAKESET(x_1)	1
MAKESET(x_2)	1
$\vdots$	1
MAKESET(x_n)	1
UNION(x_1, x_2)	1
UNION(x_2, x_3)	2
$\vdots$	$\vdots$
UNION(x_{n-1}, x_n)	$n-1$
total	$= \Theta(n^2) = \Theta(m^2)$

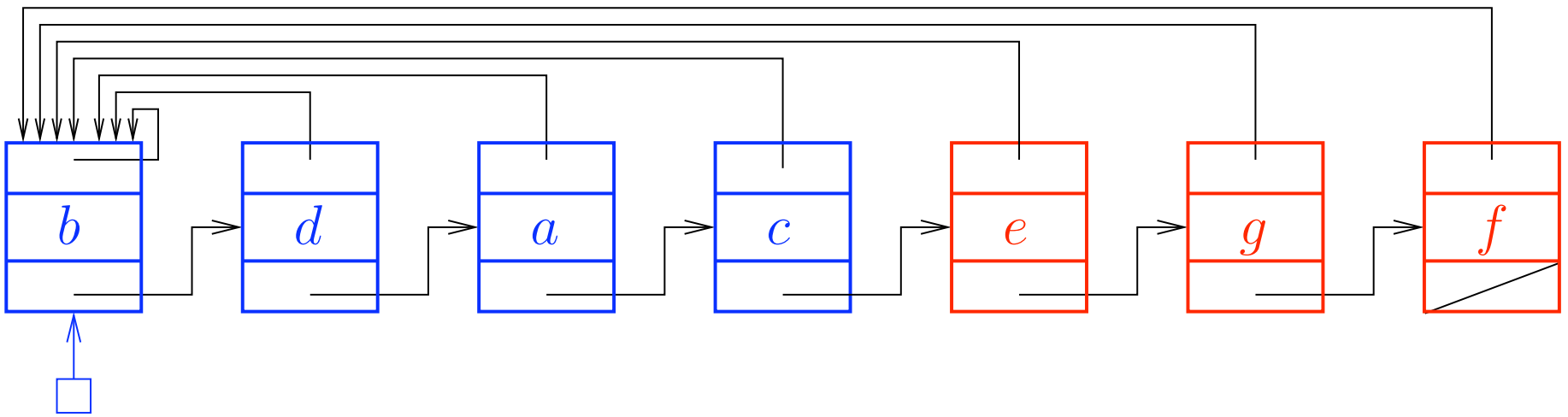
Consumo de tempo

MAKESET	$\Theta(1)$
UNION	$\mathrm{O}(n)$
FINDSET	$\Theta(1)$

Uma seqüência de m operações pode consumir tempo $\Theta(m^2)$ no pior caso.

Consumo de tempo amortizado de cada operação é $\mathrm{O}(m)$.

Melhoramento: *weighted-union*



- cada representante armazena o comprimento da lista
- a lista menor é concatenada com a maior

Cada objeto x é atualizado $\leq \lg n$:

cada vez que x é atualizado o tamanho da lista dobra.

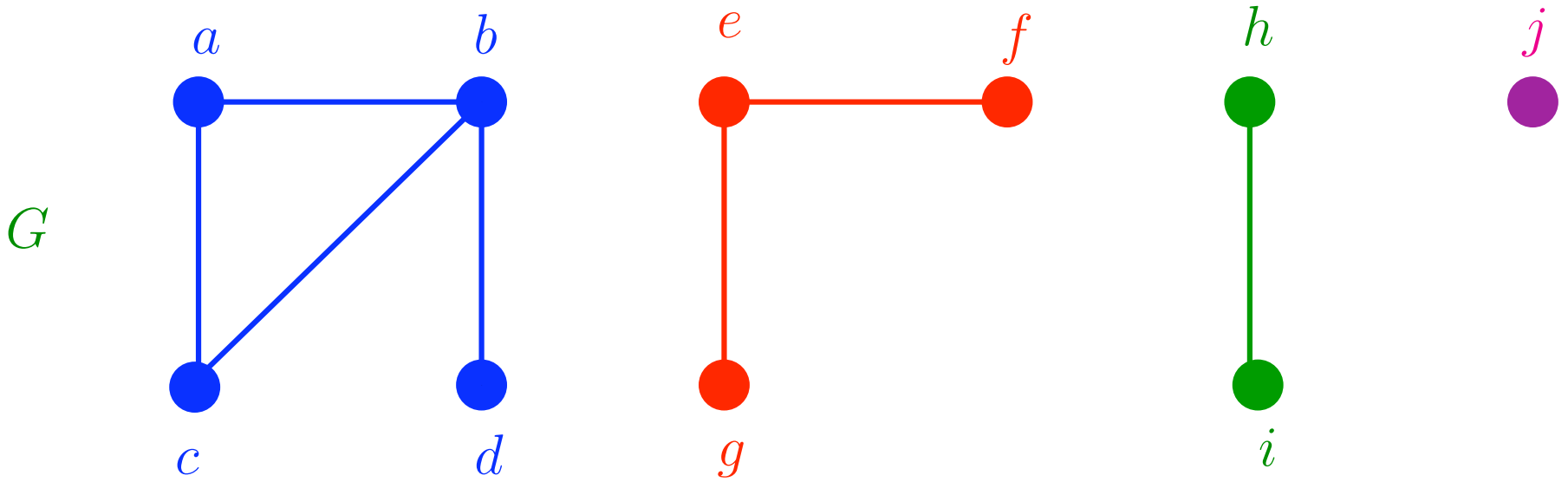
Conclusões

Se conjuntos disjuntos são representados através de listas ligadas e *weighted-union* é utilizada, então uma seqüência de m operações MAKESET, UNION e FINDSET, sendo que n são MAKESET, consome tempo $\mathrm{O}(m + n \lg n)$.

Se conjuntos disjuntos são representados através de listas ligadas e *weighted-union* é utilizada, então o algoritmos CONNECTED-COMPONENTS consome tempo $\mathrm{O}(m + n \lg n)$ e o algoritmo MST-KRUSKAL consome tempo $\mathrm{O}((n + m) \lg n)$.

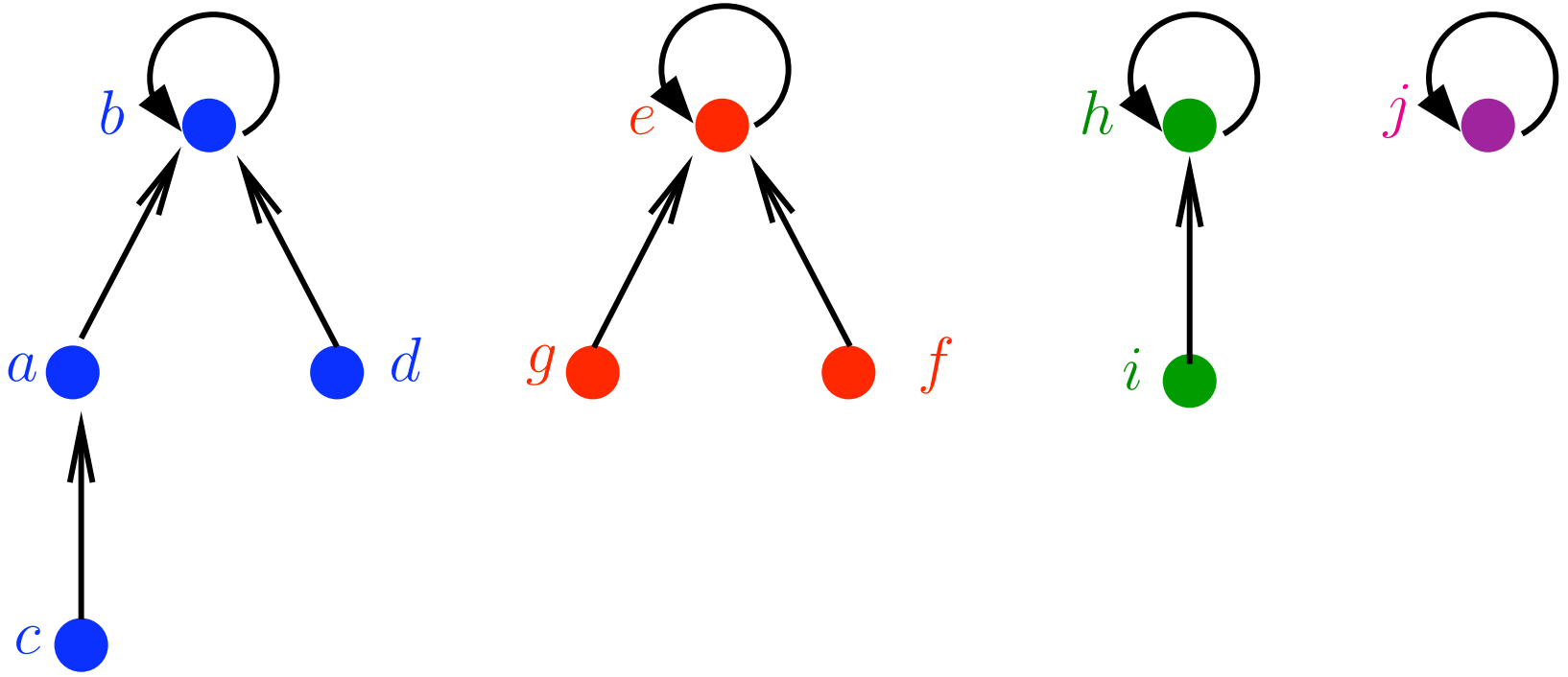
No que se refere ao algoritmo MST-KRUSKAL, estamos supondo que $m = \mathrm{O}(n^2)$.

Estrutura *disjoint-set forest*



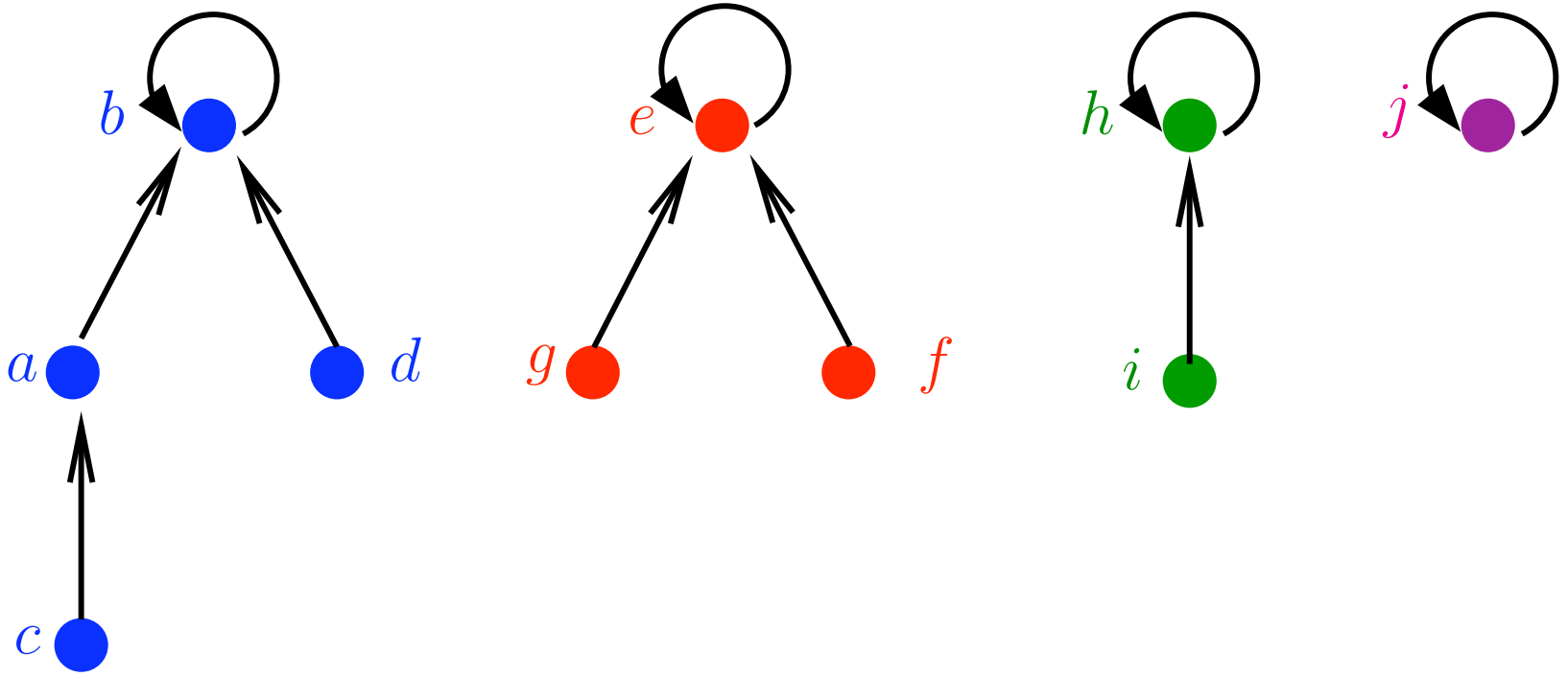
- cada conjunto tem uma $raiz$, que é o seu representate
- cada nó x tem um pai
- $pai[x] = x$ se e só se x é uma raiz

Estrutura *disjoint-set forest*



- cada conjunto tem uma $raiz$
- cada nó x tem um pai
- $pai[x] = x$ se e só se x é uma raiz

MakeSet$_0$ e FindSet$_0$



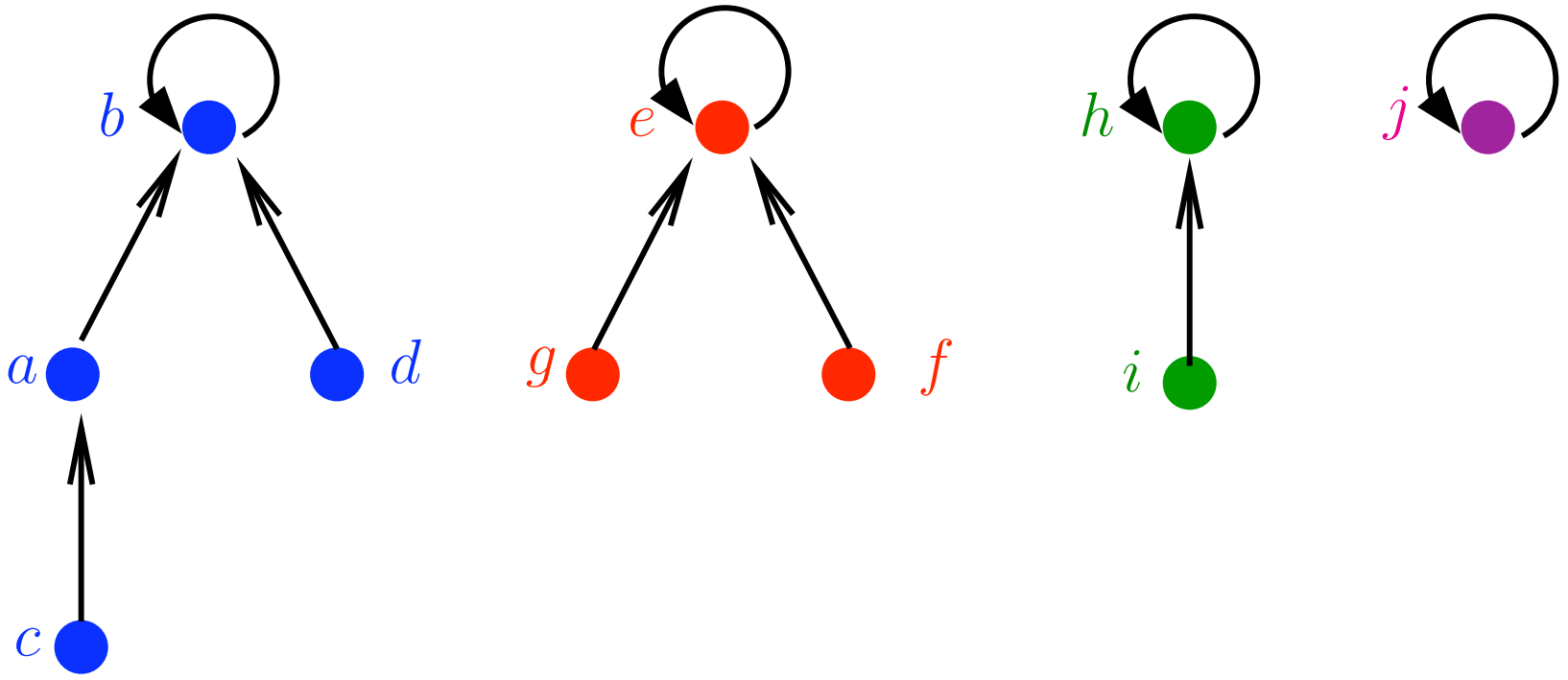
MAKESET$_0$ (x)

```
1    pai[x] ← x
```

FINDSET$_0$ (x)

```
1    enquanto pai[x] ≠ x faça
2        x ← pai[x]
3    devolva x
```

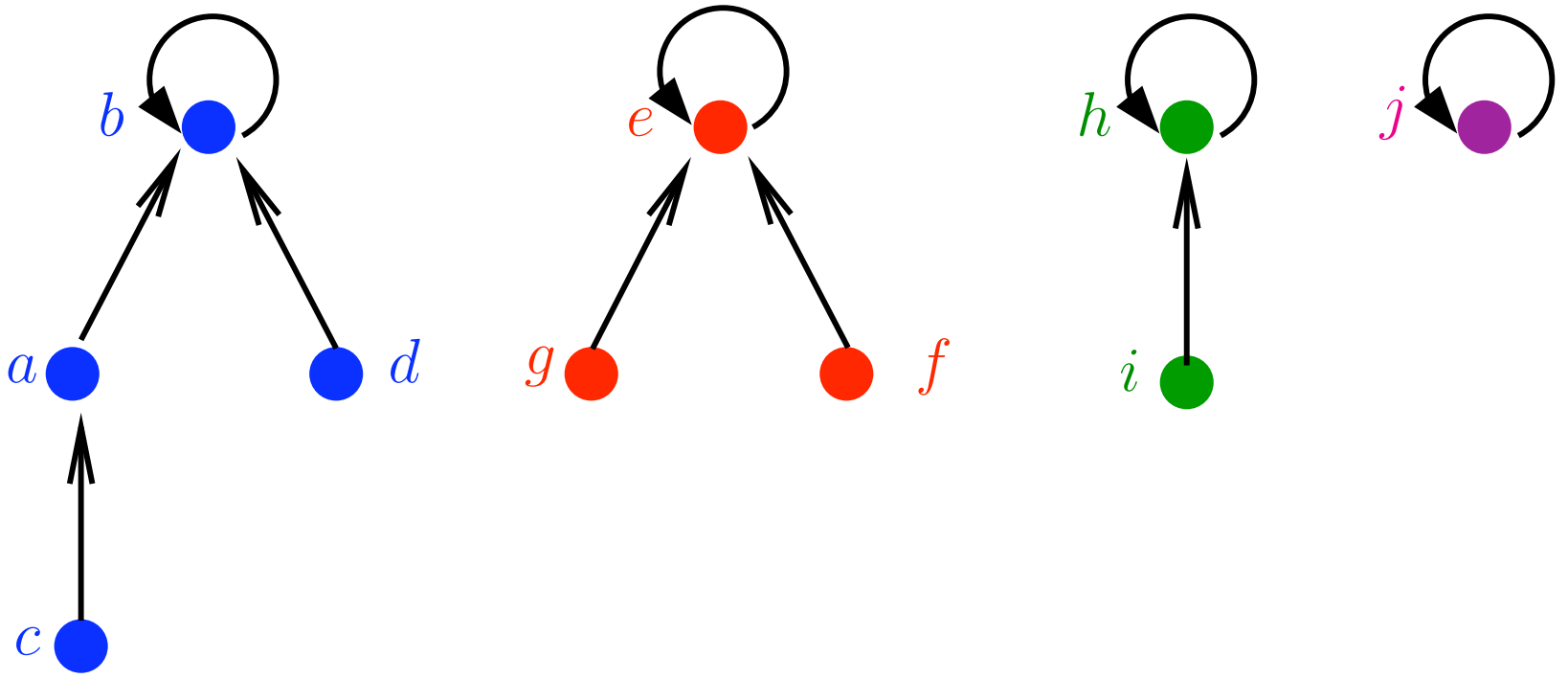
FindSet$_1$



FINDSET$_1$ (x)

```
1    se pai[x] = x
2          então devolva x
3          senão devolva FINDSET_1 (pai[x])
```

$\textbf{Union}_0$



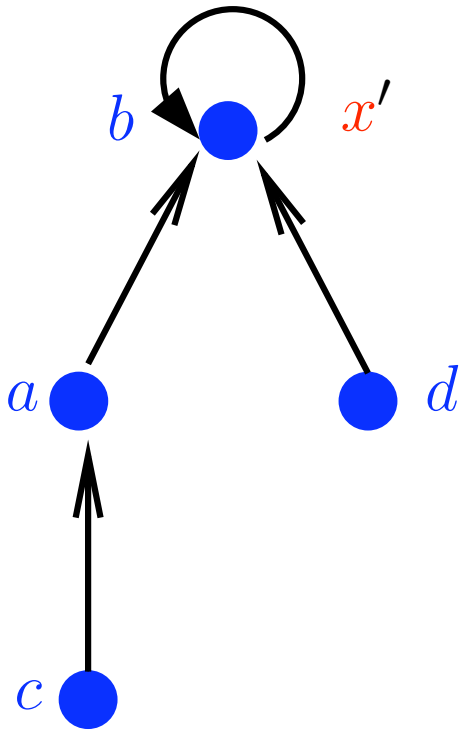
$\text{UNION}_0\ (x, y)$

1 $\quad x' \leftarrow \text{FINDSET}_0\ (x)$

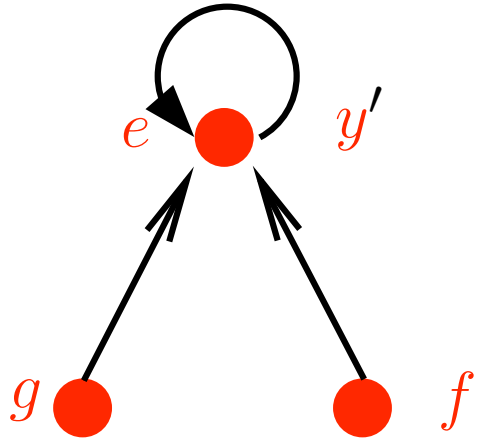
2 $\quad y' \leftarrow \text{FINDSET}_0\ (y)$

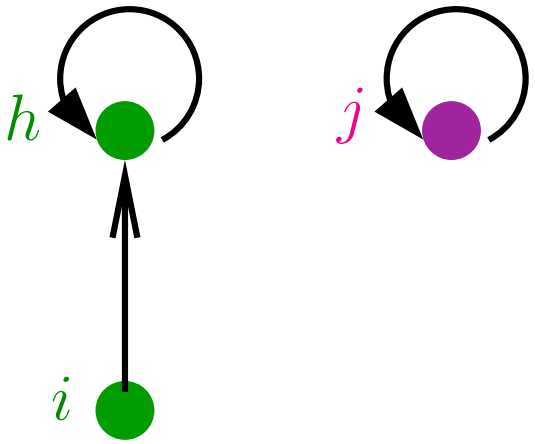
3 $\quad pai[y'] \leftarrow x'$

Union_0



$\text{UNION}_0\ (a, f)$





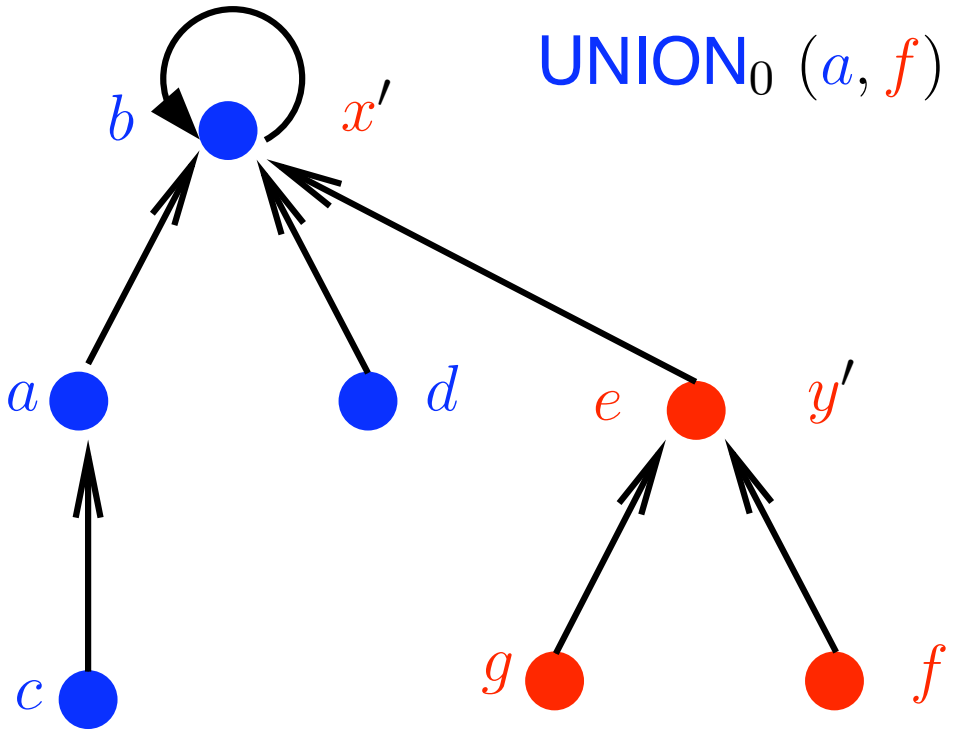
$\text{UNION}_0\ (x, y)$

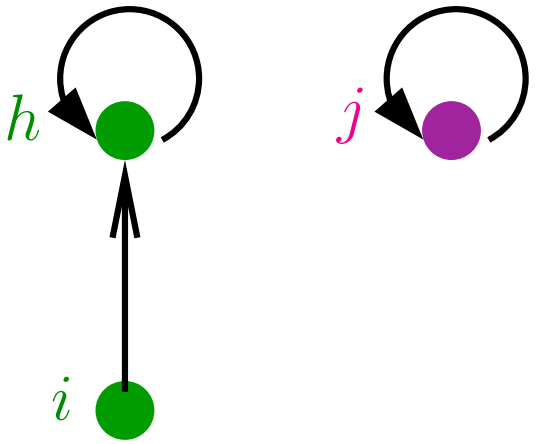
1 $\quad x' \leftarrow \text{FINDSET}_0\ (x)$

2 $\quad y' \leftarrow \text{FINDSET}_0\ (y)$

3 $\quad pai[y'] \leftarrow x'$

Union_0





UNION$_0$ (x, y)

1 $x' \leftarrow$ FINDSET$_0$ (x)

2 $y' \leftarrow$ FINDSET$_0$ (y)

3 $pai[y'] \leftarrow x'$

MakeSet_0, Union_0 e FindSet_1

$\text{MAKESET}_0\ (x)$

1 $\quad pai[x] \leftarrow x$

$\text{UNION}_0\ (x, y)$

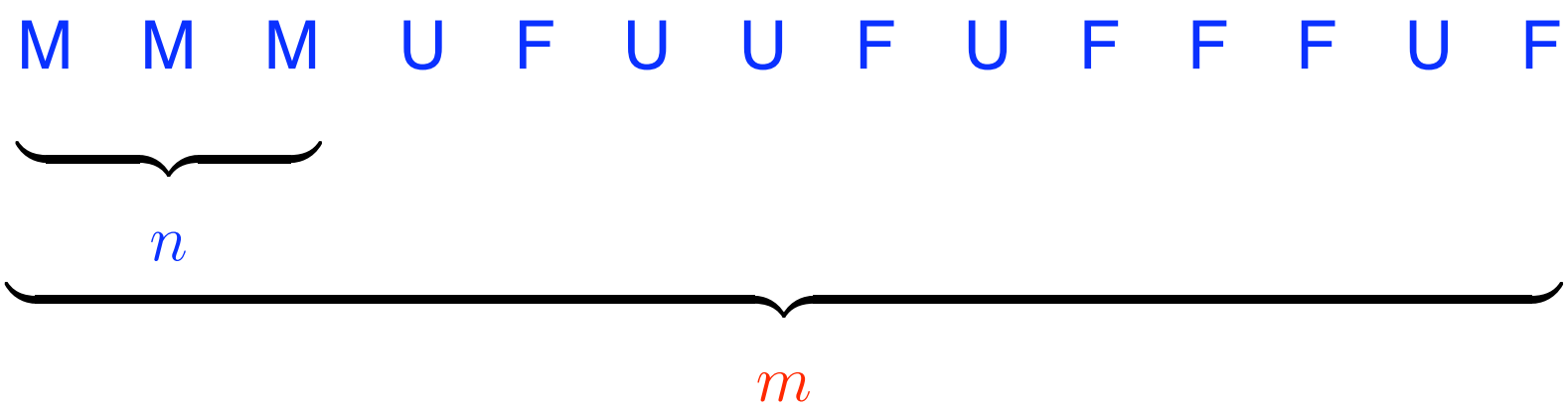
1 $\quad x' \leftarrow \text{FINDSET}_0\ (x)$
2 $\quad y' \leftarrow \text{FINDSET}_0\ (y)$
3 $\quad pai[y'] \leftarrow x'$

$\text{FINDSET}_1\ (x)$

1 $\quad$ **se** $pai[x] = x$
2 $\qquad$ **então devolva** x
3 $\qquad$ **senão devolva** $\text{FINDSET}_1\ (pai[x])$

Consumo de tempo

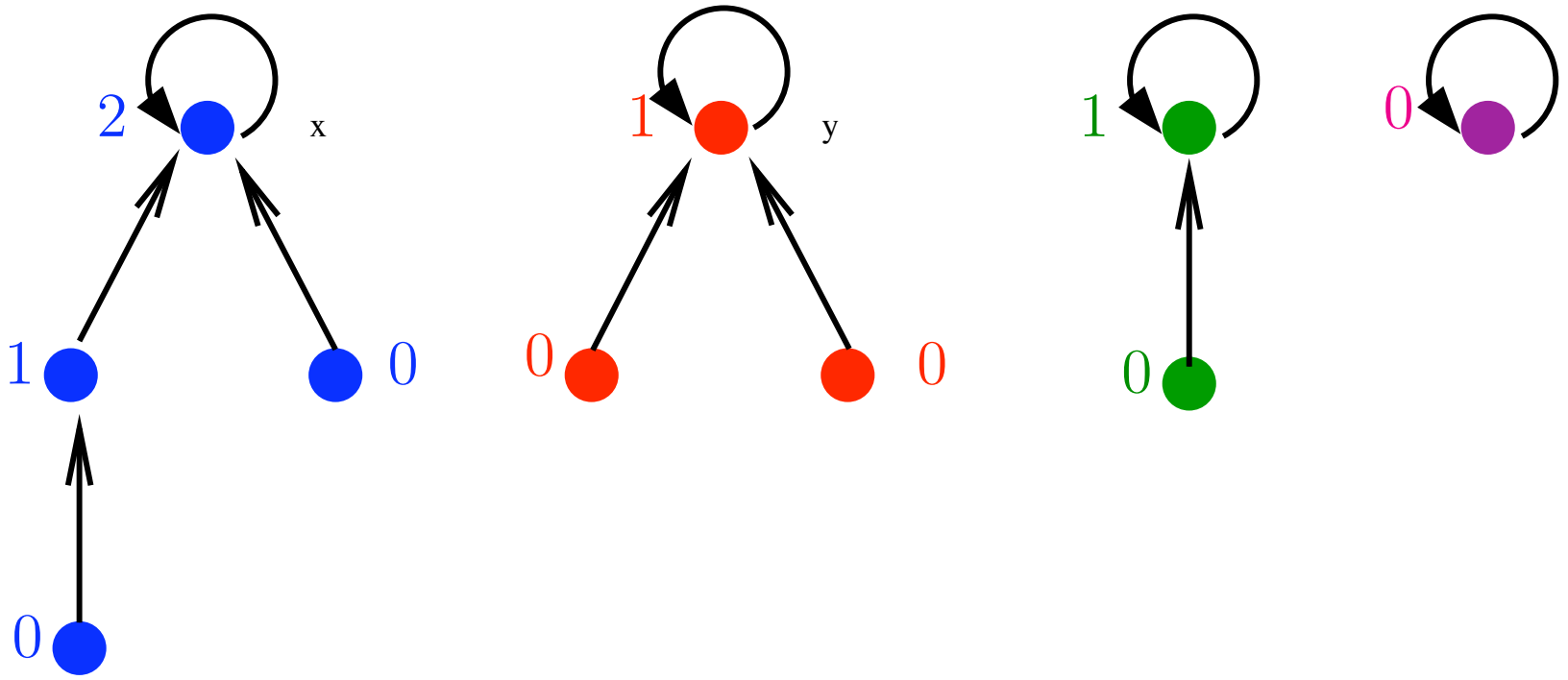
MAKESET$_0$	$\Theta(1)$
UNION$_0$	$\mathrm{O}(n)$
FINDSET$_0$	$\mathrm{O}(n)$



Custo total da seqüência:

$$n\,\Theta(1) + n\,\mathrm{O}(n) + m\,\mathrm{O}(n) = \mathrm{O}(mn)$$

Melhoramento 1: *union by rank*



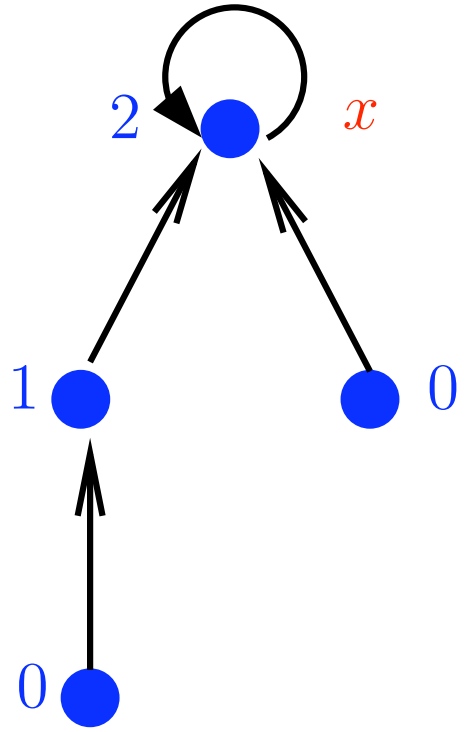
$rank[x]$ = posto do nó x

MAKESET (x)

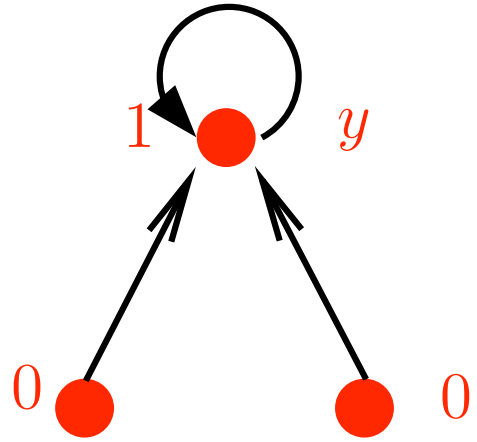
1 $pai[x] \leftarrow x$

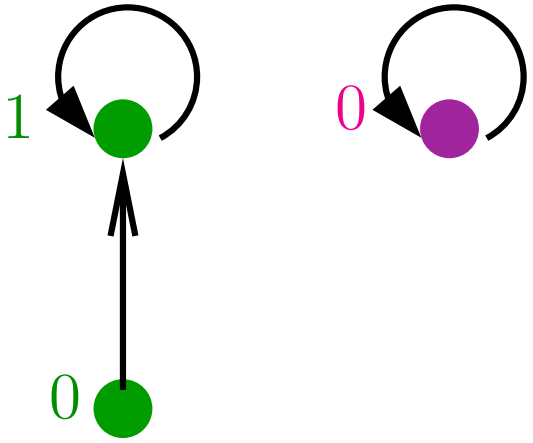
2 $rank[x] \leftarrow 0$

Melhoramento 1: *union by rank*



UNION (x, y)





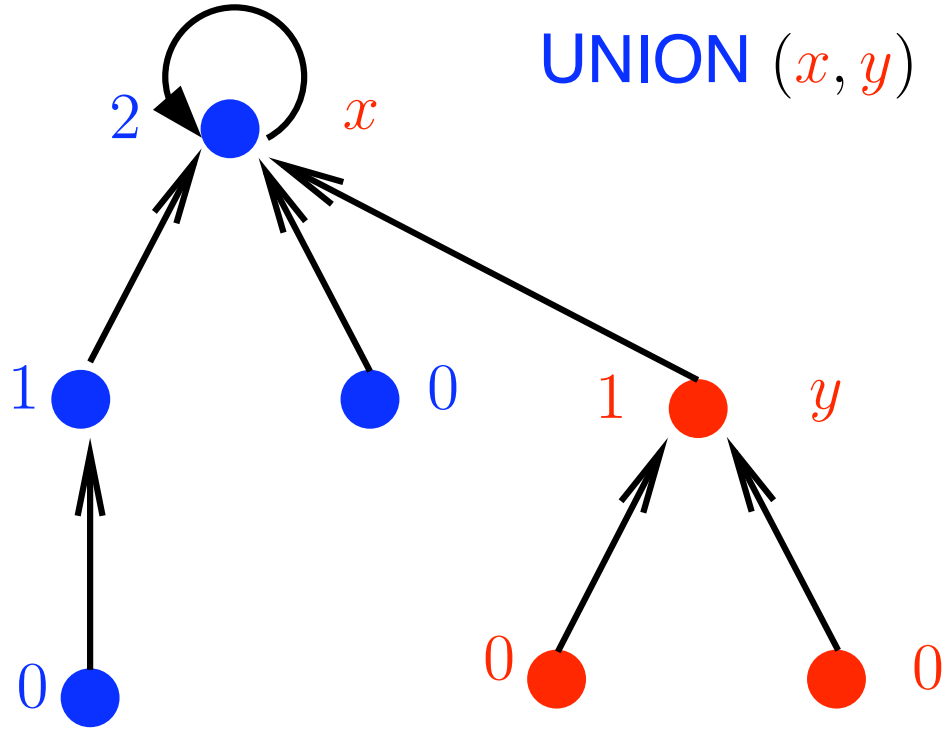
$rank[x]$ = posto do nó x

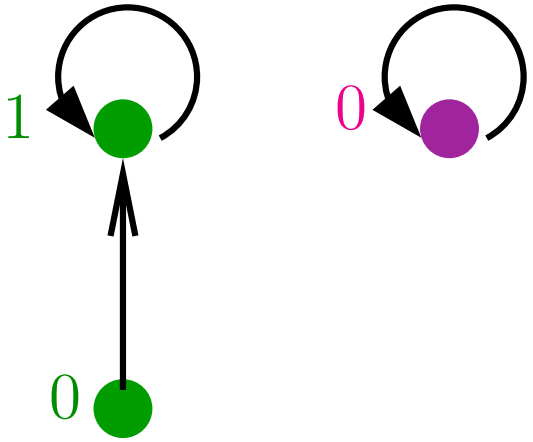
MAKESET (x)

1 $pai[x] \leftarrow x$

2 $rank[x] \leftarrow 0$

Melhoramento 1: *union by rank*





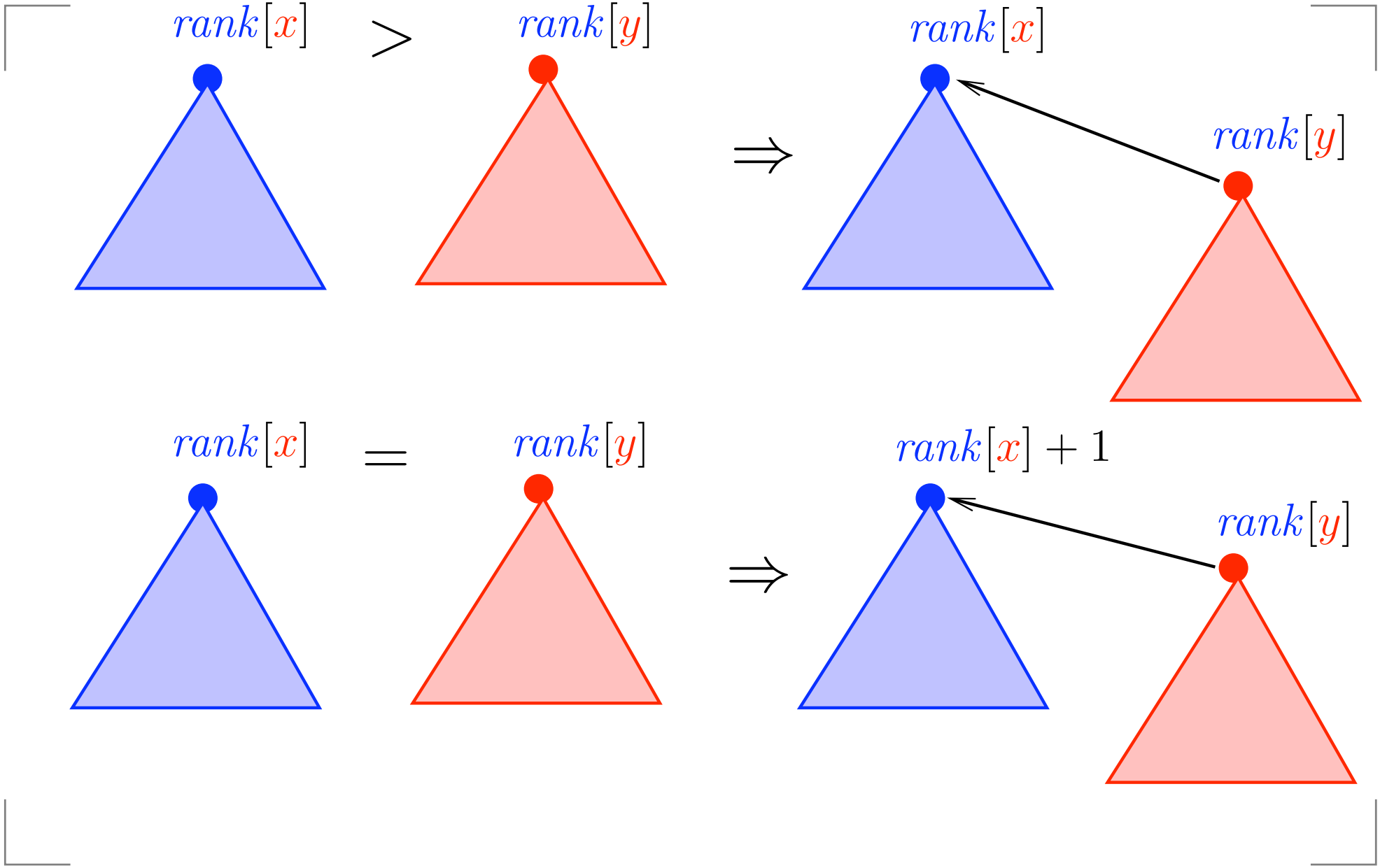
$rank[x]$ = posto do nó x

MAKESET (x)

1 $\quad pai[x] \leftarrow x$

2 $\quad rank[x] \leftarrow 0$

Melhoramento 1: *union by rank*



Melhoramento 1: *union by rank*

```
UNION (x, y)  ▷ com "union by rank"
1    x' ← FINDSET (x)
2    y' ← FINDSET (y)     ▷ supõe que x' ≠ y'
3    se rank[x'] > rank[y']
4         então pai[y'] ← x'
5         senão pai[x'] ← y'
6               se rank[x'] = rank[y']
7                    então rank[y'] ← rank[y'] + 1
```

Melhoramento 1: estrutura

- $rank[x] \leq rank[pai[x]]$ para cada nó x
- $rank[x] = rank[pai[x]]$ se e só se x é raiz
- $rank[pai[x]]$ é uma função não-descrente do tempo
- número de nós de uma árvore de raiz x é $\geq 2^{rank[x]}$.
- número de nós de posto k é $\leq n/2^k$.
- $altura(x) = rank[x] \leq \lg n$ para cada nó x

$altura(x) :=$ comprimento do mais longo caminho
que vai de x até uma folha

Melhoramento 1: custo

Seqüência de operações MAKESET, UNION, FINDSET

M M M U F U U F U F F F U F

$\underbrace{}_{n}$

$\underbrace{}_{m}$

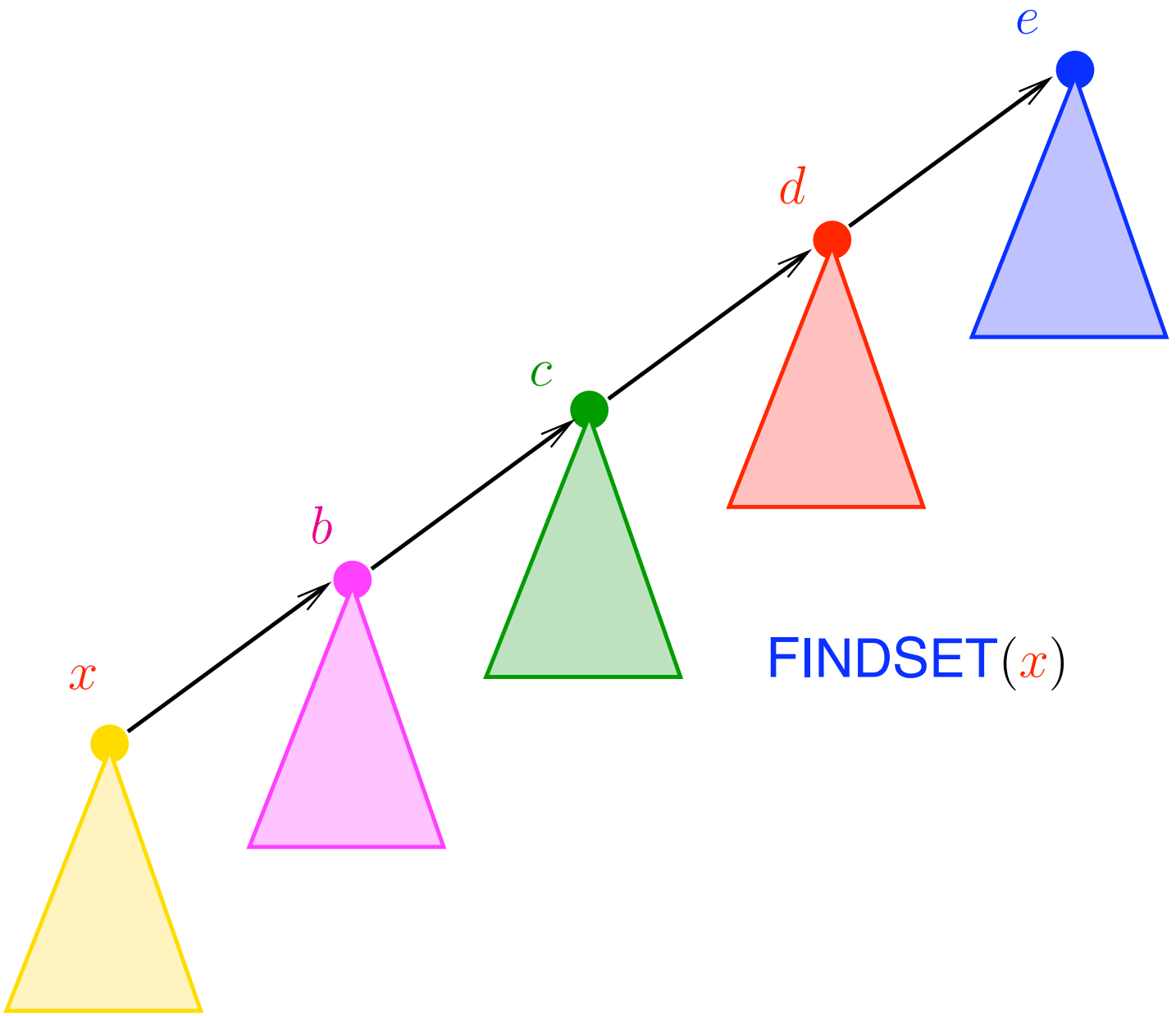
$altura(x) \leq \lg n$ para cada nó x

Consumos de tempo:

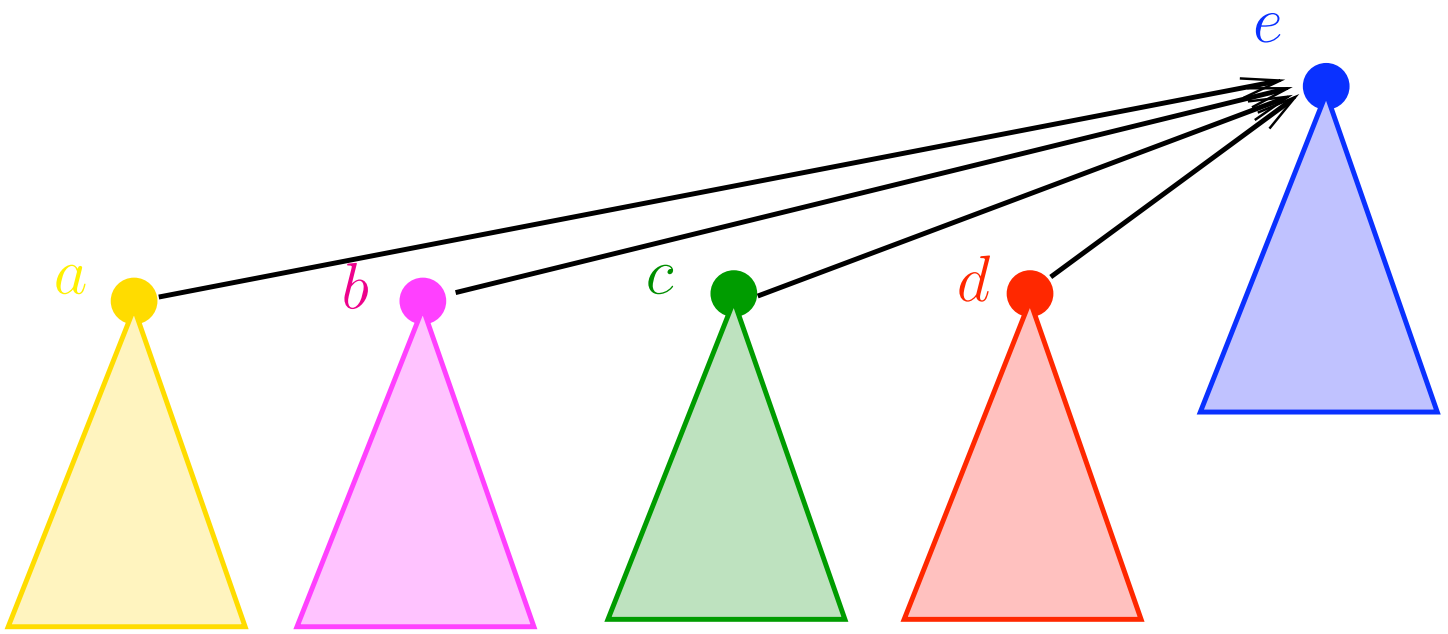
MAKESET	$\Theta(1)$
UNION	$\mathrm{O}(\lg n)$
FINDSET	$\mathrm{O}(\lg n)$

Consumo de tempo total da seqüência: $\mathrm{O}(m \lg n)$

Melhoramento 2: *path compression*



Melhoramento 2: *path compression*



FINDSET(x)

Melhoramento 2: *path compression*

```
FINDSET (x)  ▷ com "path compression"
1    se x ≠ pai[x]
2         então pai[x] ← FINDSET (pai[x])
3    devolva pai[x]
```

- $rank[x] \leq rank[pai[x]]$ para cada nó x
- $rank[x] = rank[pai[x]]$ se e só se x é raiz
- $rank[pai[x]]$ é uma função não-descrente do tempo
- número de nós de uma árvore de raiz x é $\geq 2^{rank[x]}$.
- número de nós de posto k é $\leq n/2^k$.
- $altura(x) \leq rank[x] \leq \lg n$ para cada nó x