Buscas em grafos

- **Busca: é um processo sistemático (algoritmo) utilizado para percorrer (visitar) os vértices e arestas de um grafo.**

- **A ideia é explorar o grafo de modo a resolver um problema ou extrair informações de sua estrutura.**

- **Buscas que estudaremos:**
  - **Busca em profundidade (depth-first search)**
  - **Busca em largura (breadth-first search)**

Busca em profundidade

Inicialização

t ← 0 **-- *t é o relógio ou tempo global***

<u>*para*</u> *todo vértice v em V*(*G*) <u>*faça*</u>

PE(*v*) ← 0 **-- *PE(v) é a profundidade de entrada de v***

PS (*v*) ← 0 **-- *PS(v) é a profundidade de saída de v***

pai(*v*) ← *null* **-- *ponteiros que definem a árvore de profundidade***

Chamada Externa

escolher um vértice qualquer v em V(*G*) **-- *este vértice é chamado raiz da busca***

executar P(*v*)

Busca em profundidade

Procedimento recursivo de busca

procedimento $P(v)$

$t \leftarrow t + 1$

$PE(v) \leftarrow t$

para todo vértice w em $N(v)$ *faça*

se $PE(w) = 0$

então
- visitar aresta vw (aresta ***"azul"*** da árvore de profundidade T)
- $pai(w) \leftarrow v$ (v é o pai de w na árvore de profundidade T)
- executar $P(w)$

senão
- *se* $PS(w) = 0$ *e* $w \neq pai(v)$ (se w não saiu da busca e não é pai de v)
- *então* visitar aresta vw (aresta ***"vermelha"*** de retorno)

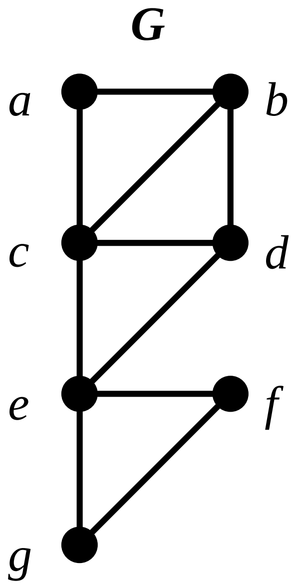
fim-para

$t \leftarrow t + 1$

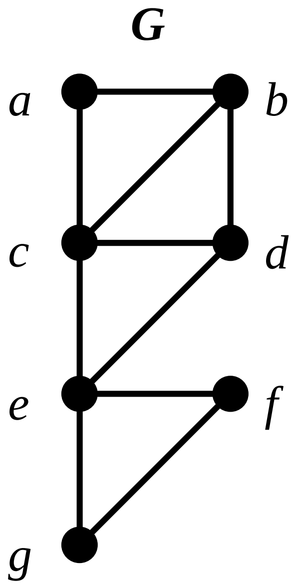
$PS(v) \leftarrow t$

fim-do-procedimento

Busca em profundidade

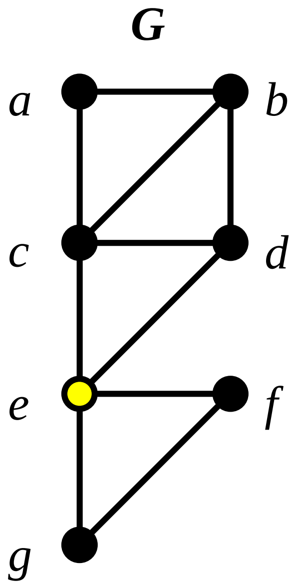


Busca em profundidade



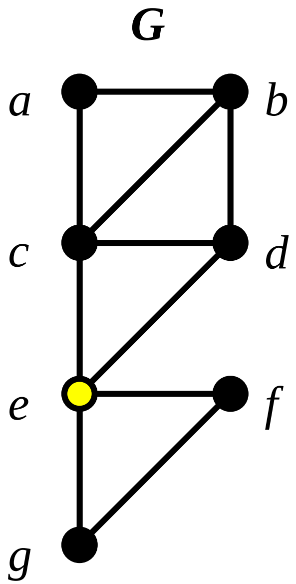
v	a	b	c	d	e	f	g
$PE(v)$	0	0	0	0	0	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade



v	a	b	c	d	e	f	g
$PE(v)$	0	0	0	0	0	0	0
$PS(v)$	0	0	0	0	0	0	0

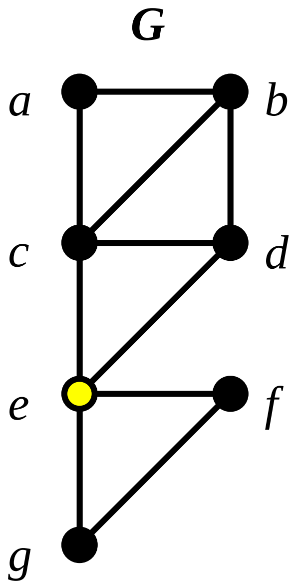
Busca em profundidade



$P(e)$

v	a	b	c	d	e	f	g
$PE(v)$	0	0	0	0	0	0	0
$PS(v)$	0	0	0	0	0	0	0

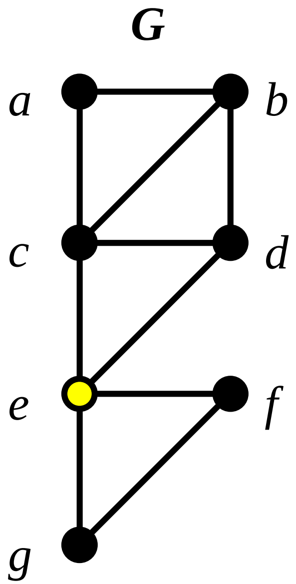
Busca em profundidade

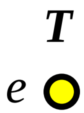


$P(e)$

v	a	b	c	d	e	f	g
$PE(v)$	0	0	0	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

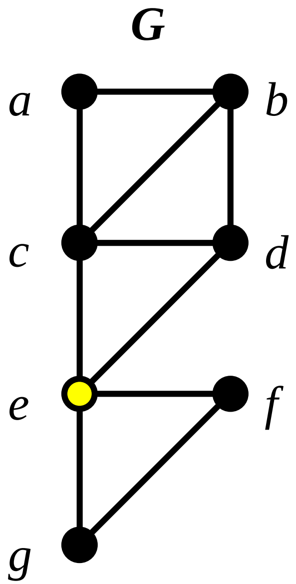


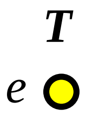


$P(e)$

v	a	b	c	d	e	f	g
$PE(v)$	0	0	0	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

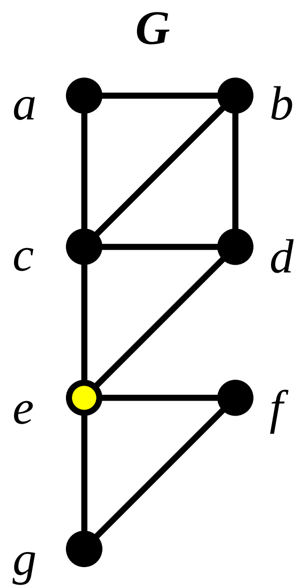


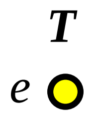


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$\boldsymbol{v}$	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{c}$	$\boldsymbol{d}$	$\boldsymbol{e}$	$\boldsymbol{f}$	$\boldsymbol{g}$
$PE(v)$	0	0	0	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

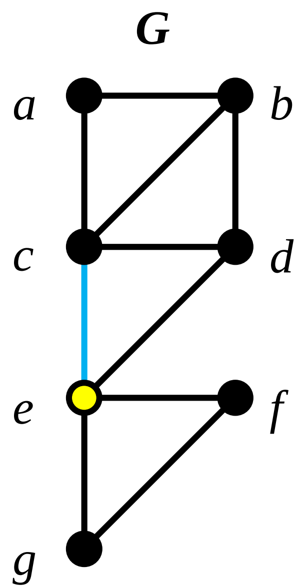


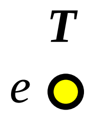


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

v	a	b	c	d	e	f	g
$PE(v)$	0	0	0	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

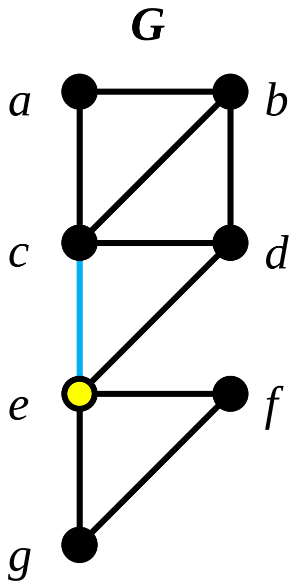


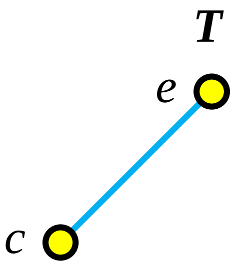


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

v	a	b	c	d	e	f	g
$PE(v)$	0	0	0	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

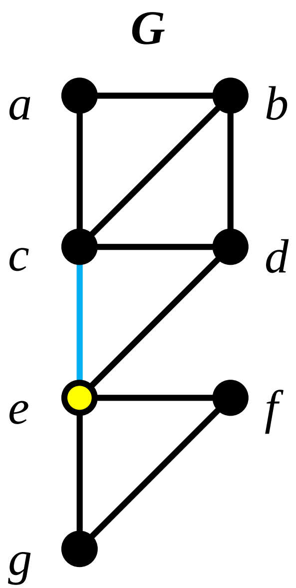


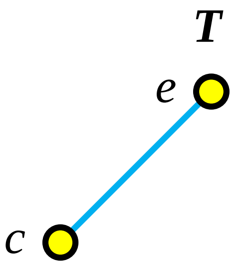


$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$

$\boldsymbol{v}$	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{c}$	$\boldsymbol{d}$	$\boldsymbol{e}$	$\boldsymbol{f}$	$\boldsymbol{g}$
$PE(v)$	0	0	0	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade



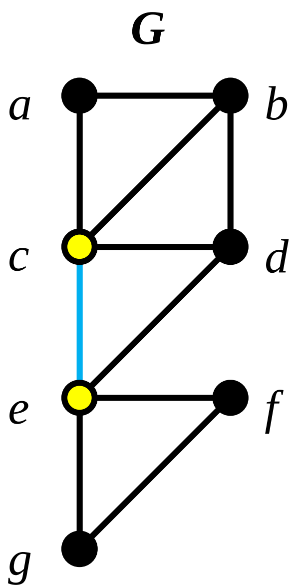


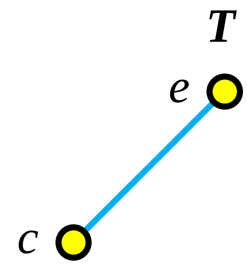
$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c)$

v	a	b	c	d	e	f	g
$PE(v)$	0	0	0	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade



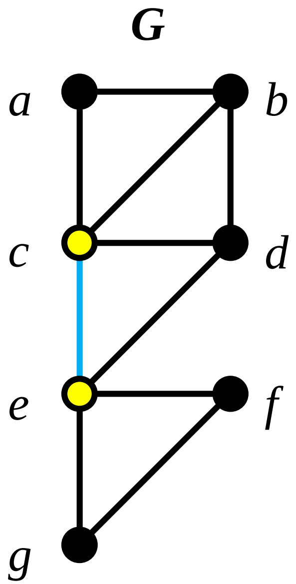


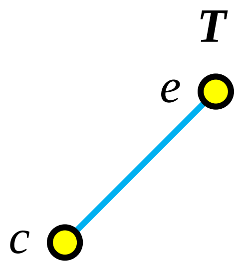
$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c)$

v	a	b	c	d	e	f	g
$PE(v)$	0	0	0	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

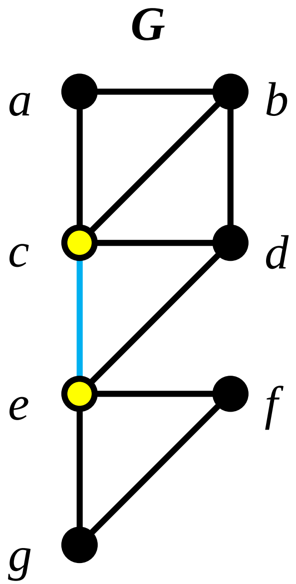


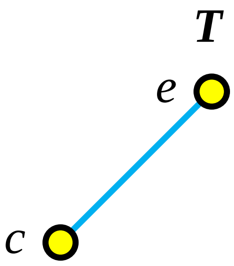


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c)$

v	a	b	c	d	e	f	g
$PE(v)$	0	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade



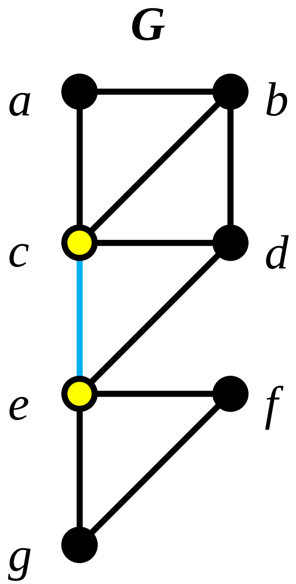


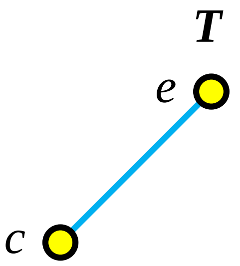
$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*
$PE(v)$	0	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade



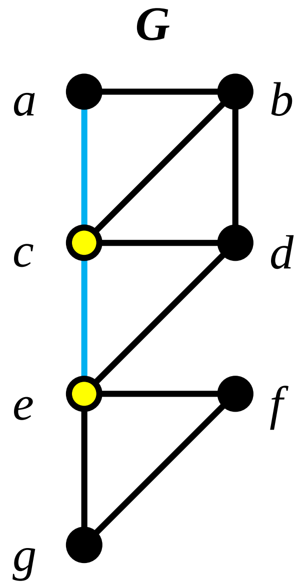


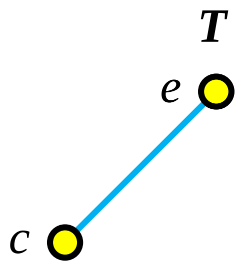
$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*
$PE(v)$	0	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade



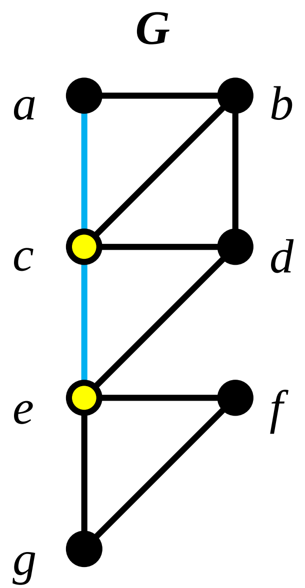


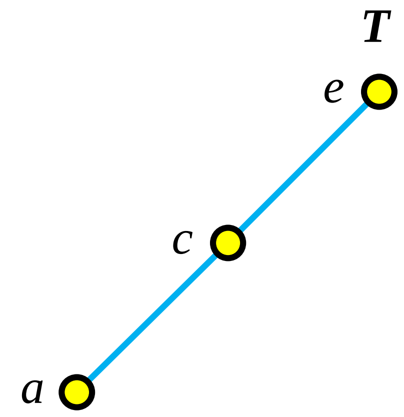
$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$

$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	0	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade



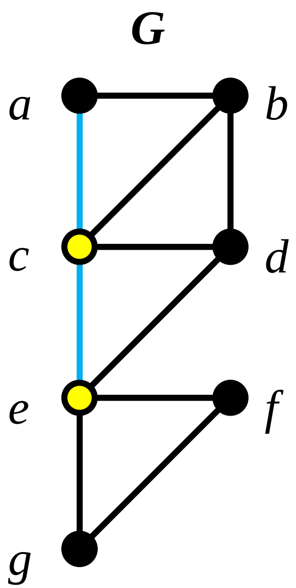


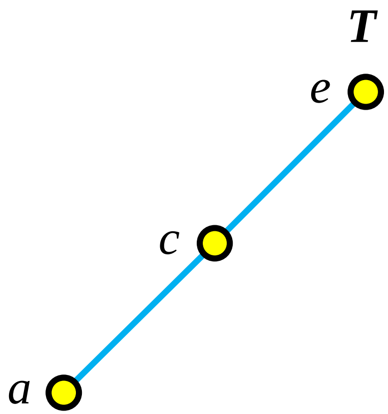
$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$

$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	0	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

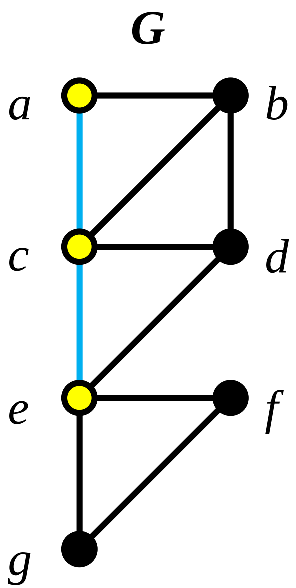


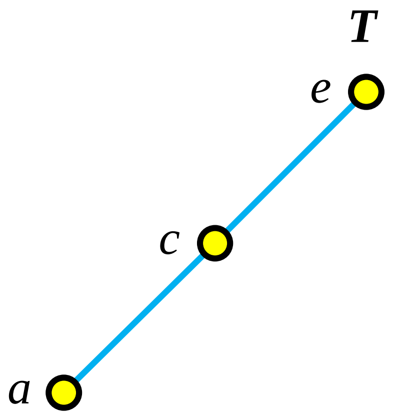


$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$
$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$
$P(a)$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	0	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

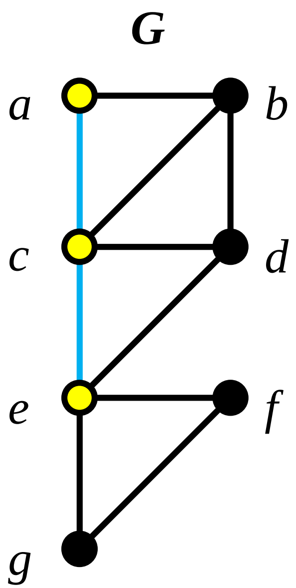


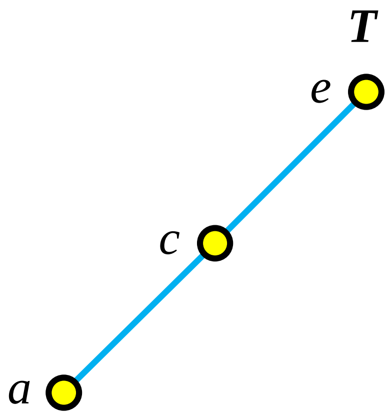


$P(e) \longrightarrow N(e) = \{\boldsymbol{c}, d, f, g\}$
$P(c) \longrightarrow N(c) = \{\boldsymbol{a}, b, d, e\}$
$P(a)$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	0	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

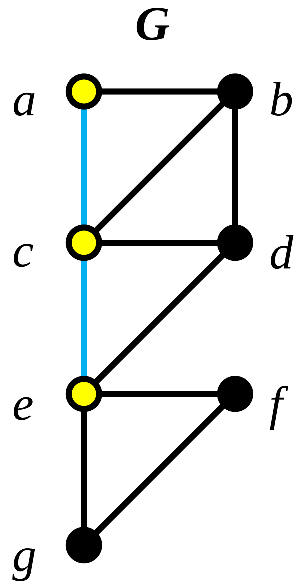


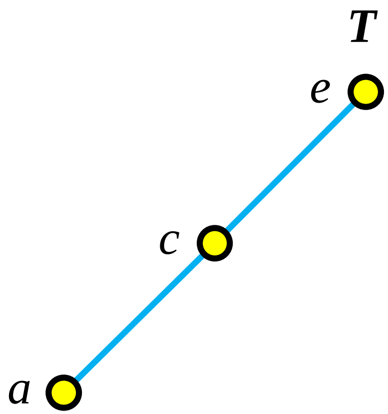


$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$
$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$
$P(a)$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

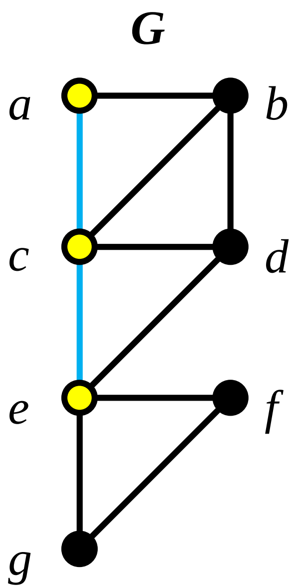


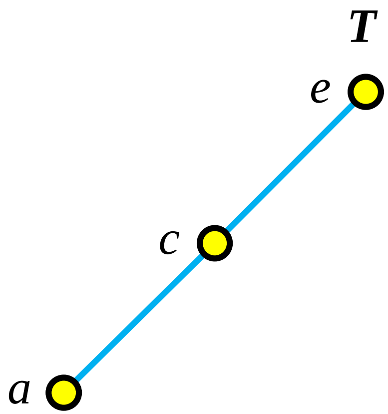


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$

v	**a**	**b**	**c**	**d**	**e**	**f**	**g**
$PE(v)$	3	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade





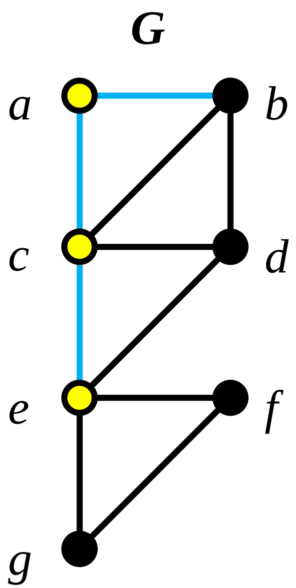
$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$

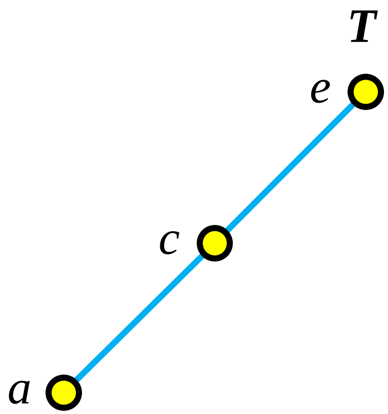
$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$

$P(a) \longrightarrow N(a) = \{\mathbf{b}, c\}$

v	**a**	**b**	**c**	**d**	**e**	**f**	**g**
$PE(v)$	3	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

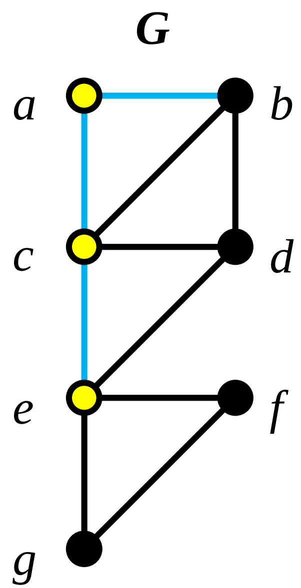


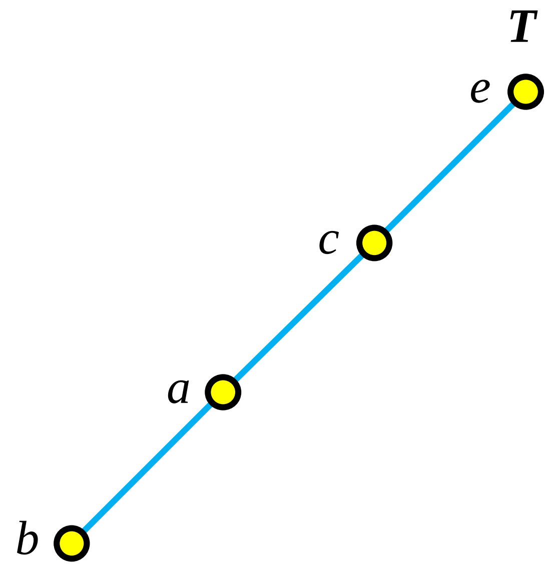


$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$
$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$
$P(a) \longrightarrow N(a) = \{\mathbf{b}, c\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

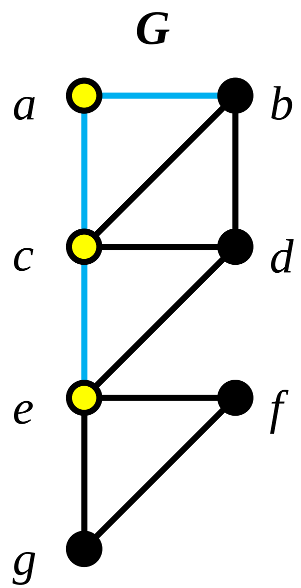


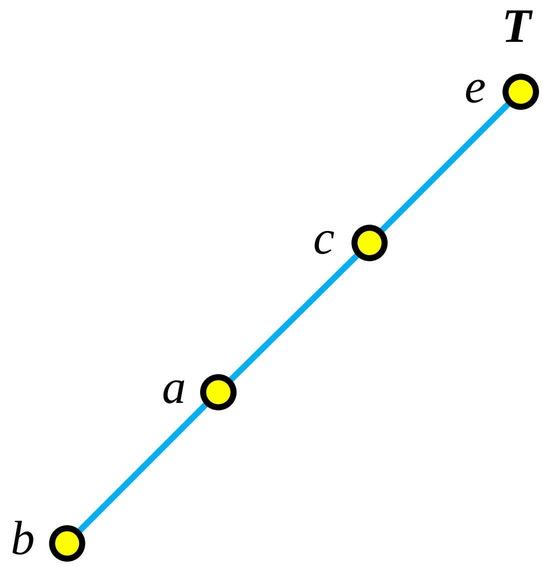


$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$
$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$
$P(a) \longrightarrow N(a) = \{\mathbf{b}, c\}$

v	a	b	c	d	e	f	g
$PE(v)$	3	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

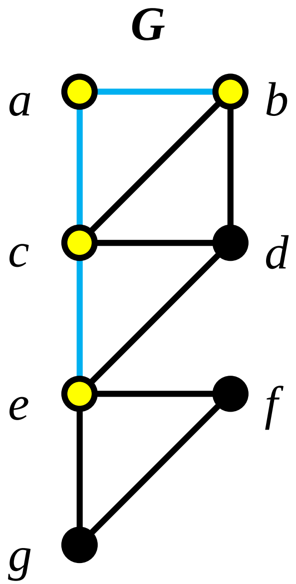


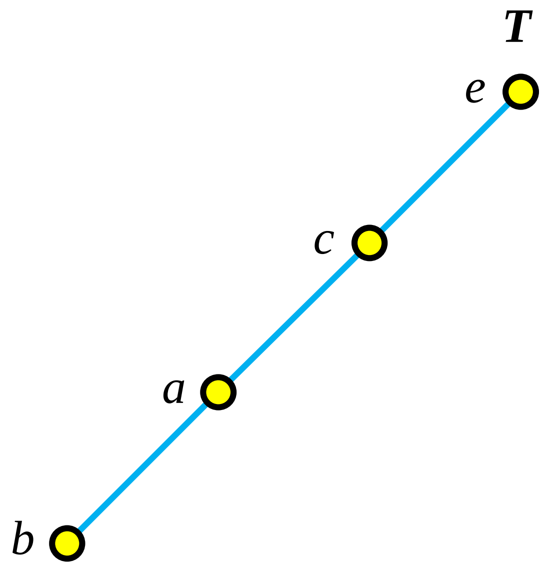


$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$
$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$
$P(a) \longrightarrow N(a) = \{\mathbf{b}, c\}$
$P(b)$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$

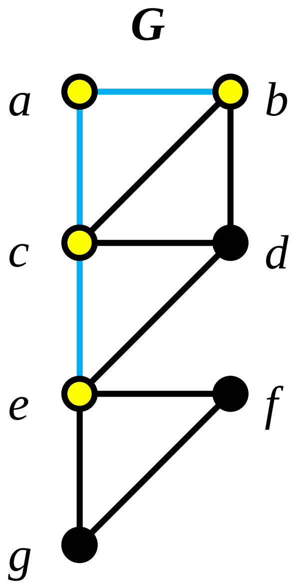
$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$

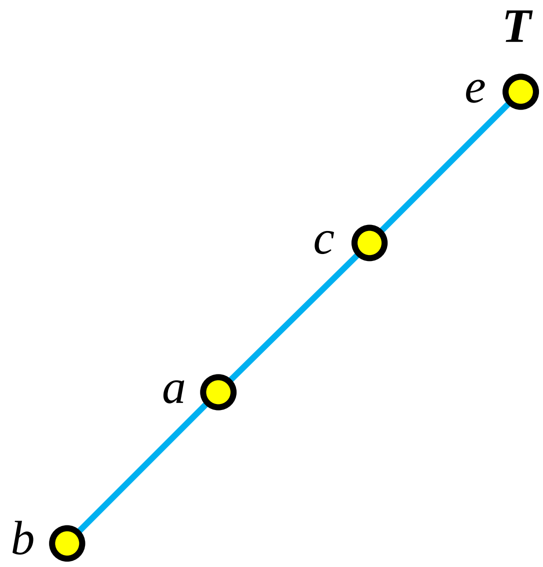
$P(a) \longrightarrow N(a) = \{\mathbf{b}, c\}$

$P(b)$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	0	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$

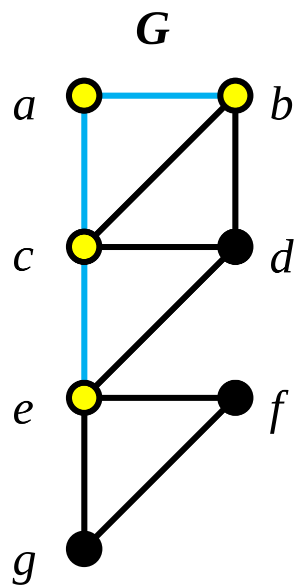
$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$

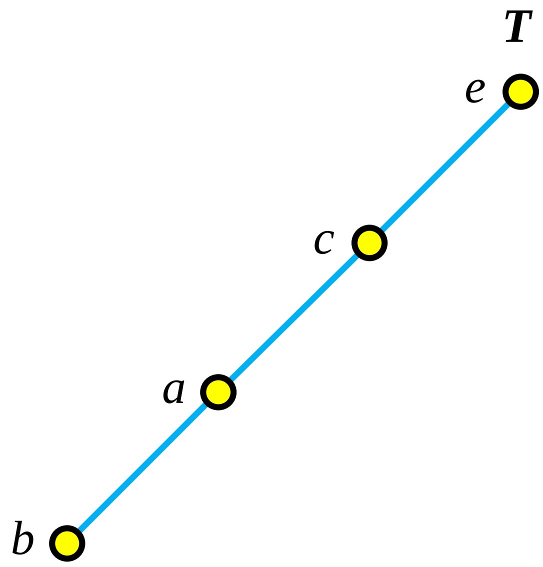
$P(a) \longrightarrow N(a) = \{\mathbf{b}, c\}$

$P(b)$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{\boldsymbol{c}, d, f, g\}$

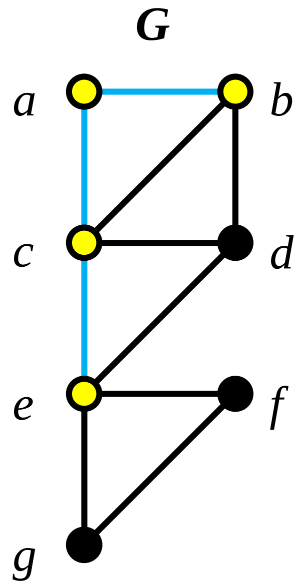
$P(c) \longrightarrow N(c) = \{\boldsymbol{a}, b, d, e\}$

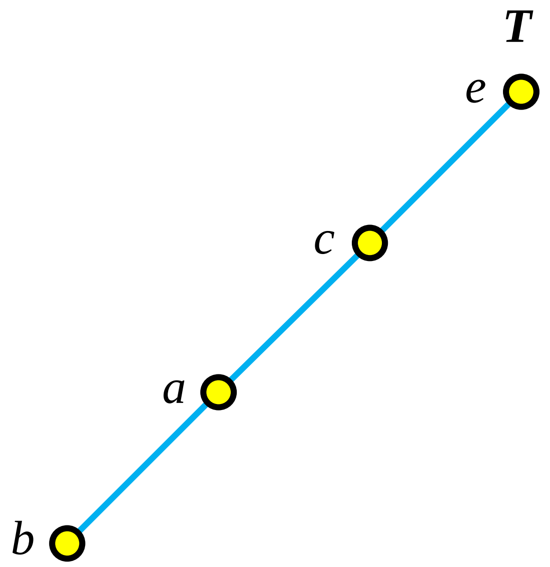
$P(a) \longrightarrow N(a) = \{\boldsymbol{b}, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$

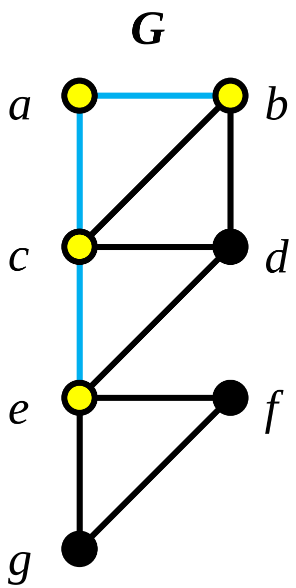
$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$

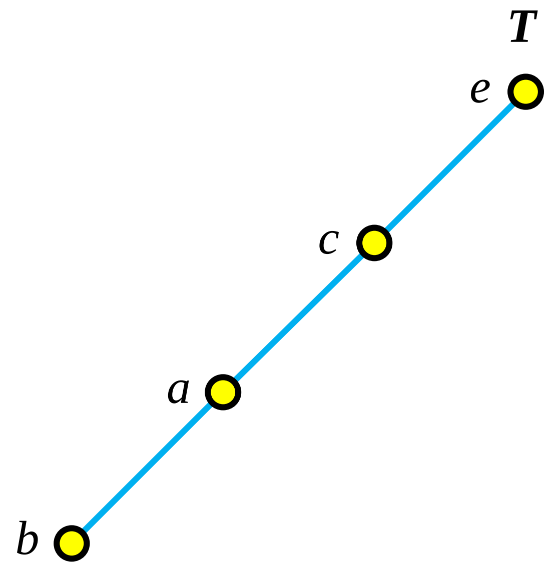
$P(a) \longrightarrow N(a) = \{\mathbf{b}, c\}$

$P(b) \longrightarrow N(b) = \{\mathbf{a}, c, d\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

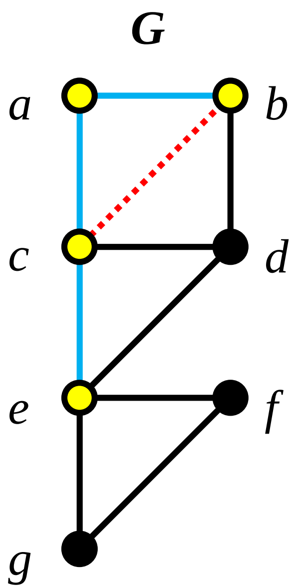
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

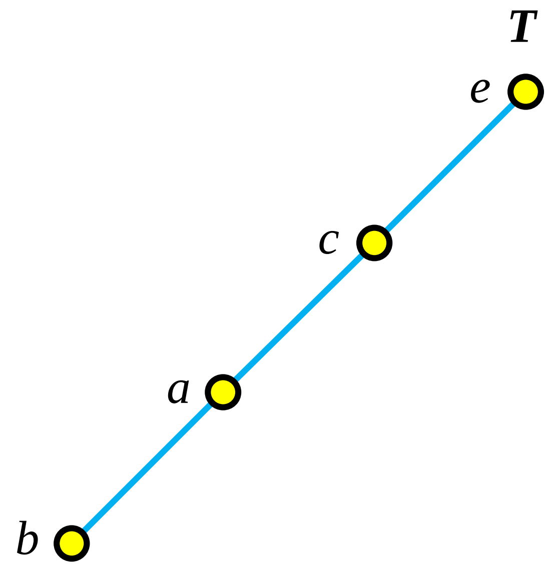
$P(a) \longrightarrow N(a) = \{b, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

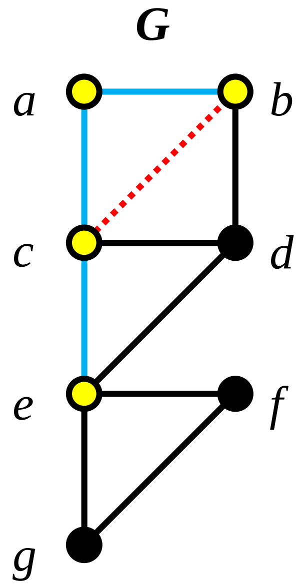
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

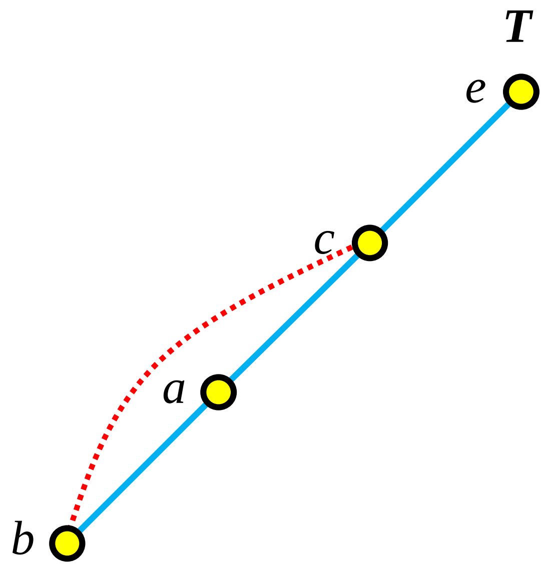
$P(a) \longrightarrow N(a) = \{b, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

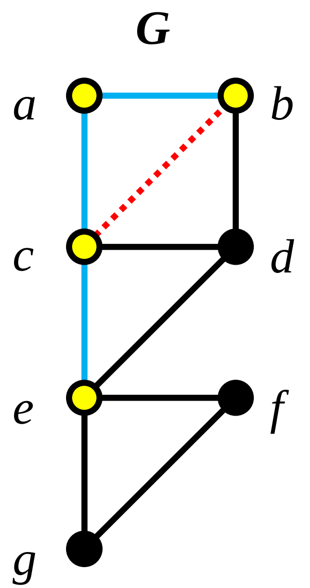


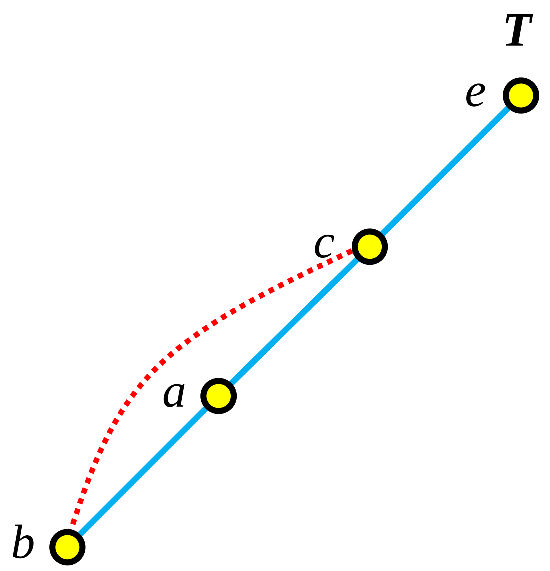


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$
$P(b) \longrightarrow N(b) = \{a, c, d\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

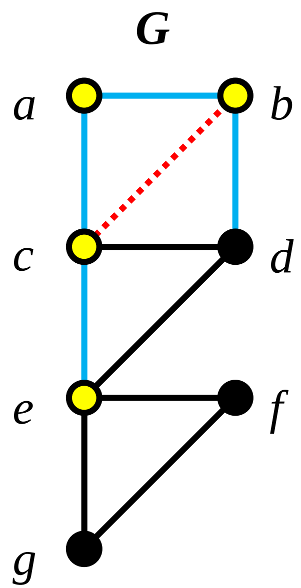


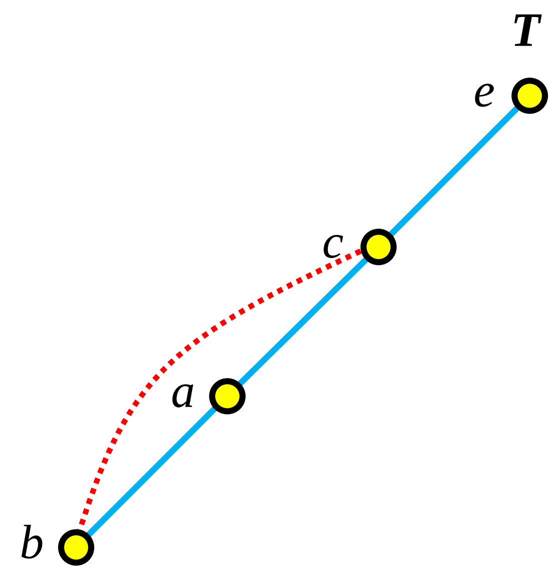


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$
$P(b) \longrightarrow N(b) = \{a, c, d\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

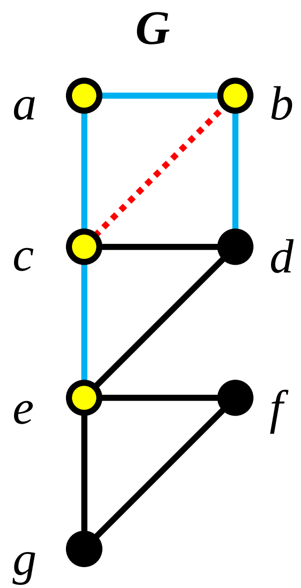
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

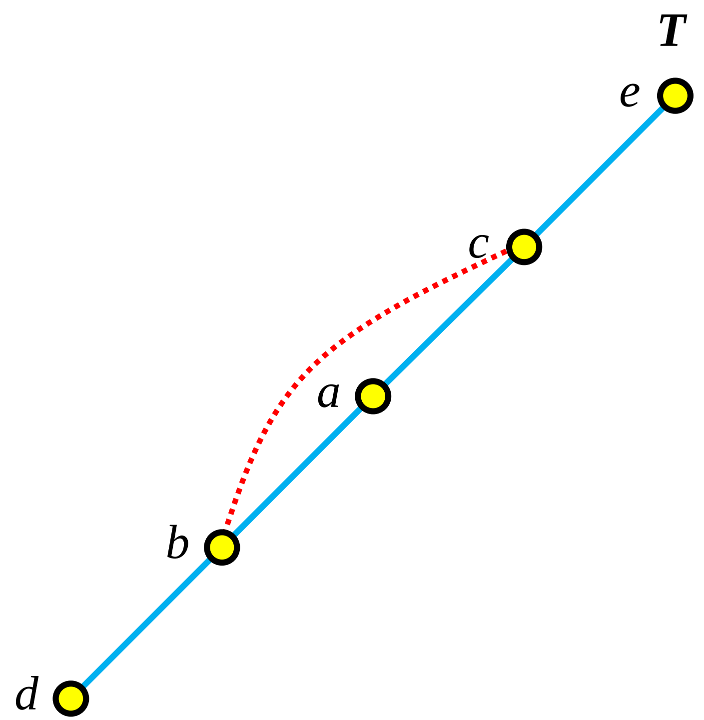
$P(a) \longrightarrow N(a) = \{b, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

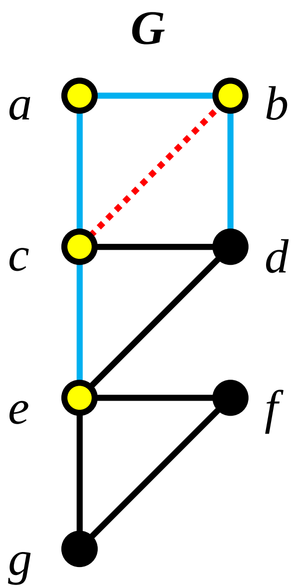


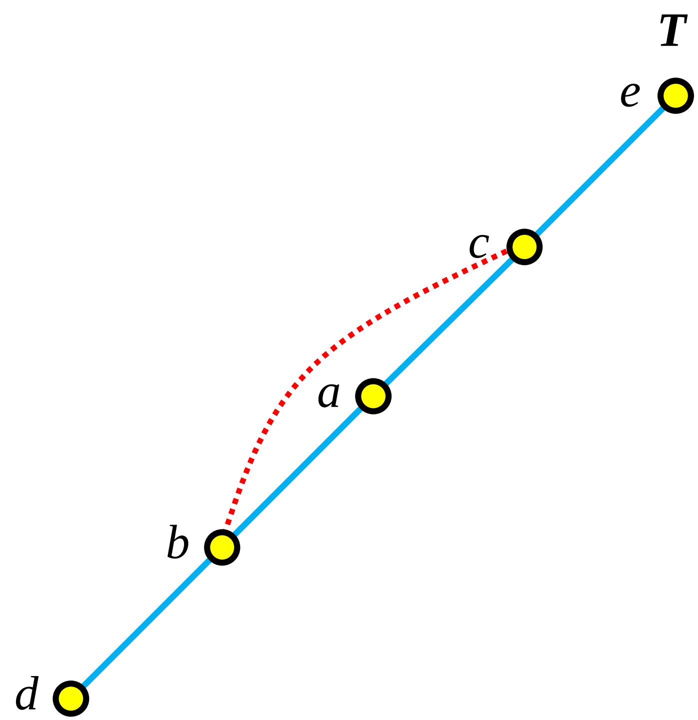


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$
$P(b) \longrightarrow N(b) = \{a, c, d\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
PE(*v*)	3	4	2	0	1	0	0
PS(*v*)	0	0	0	0	0	0	0

Busca em profundidade

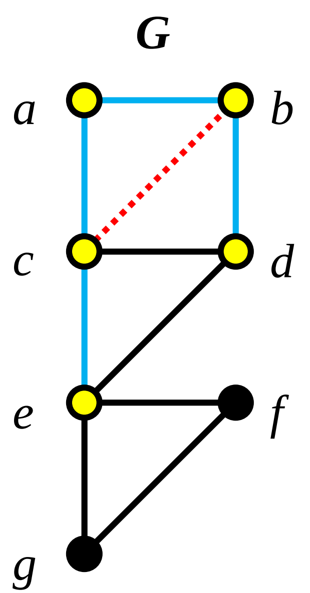


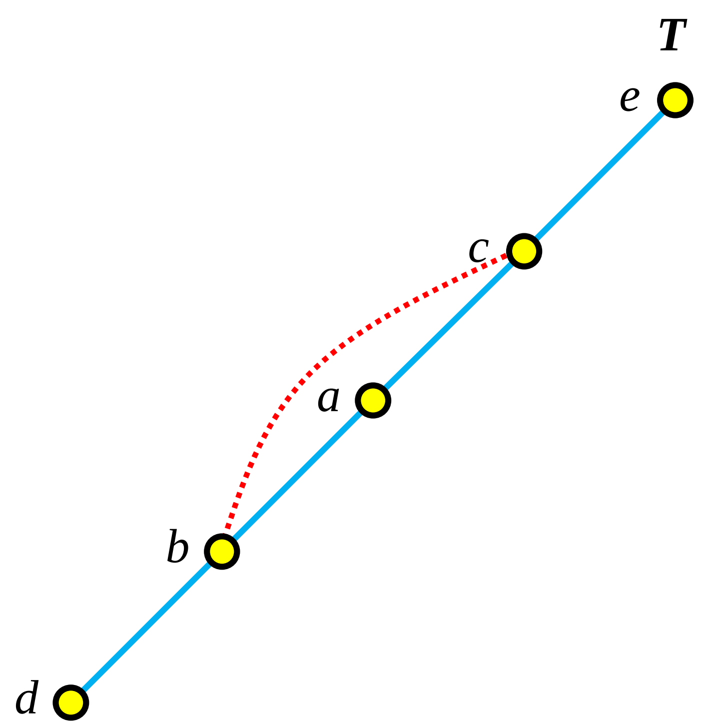


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$
$P(b) \longrightarrow N(b) = \{a, c, d\}$
$P(d)$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
PE(*v*)	3	4	2	0	1	0	0
PS(*v*)	0	0	0	0	0	0	0

Busca em profundidade

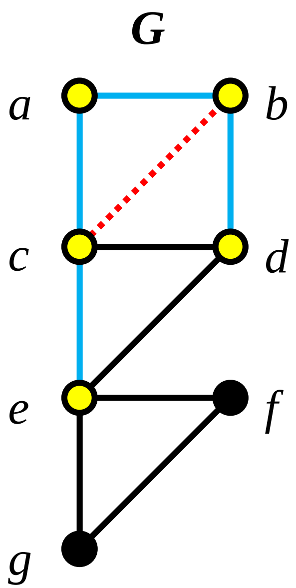


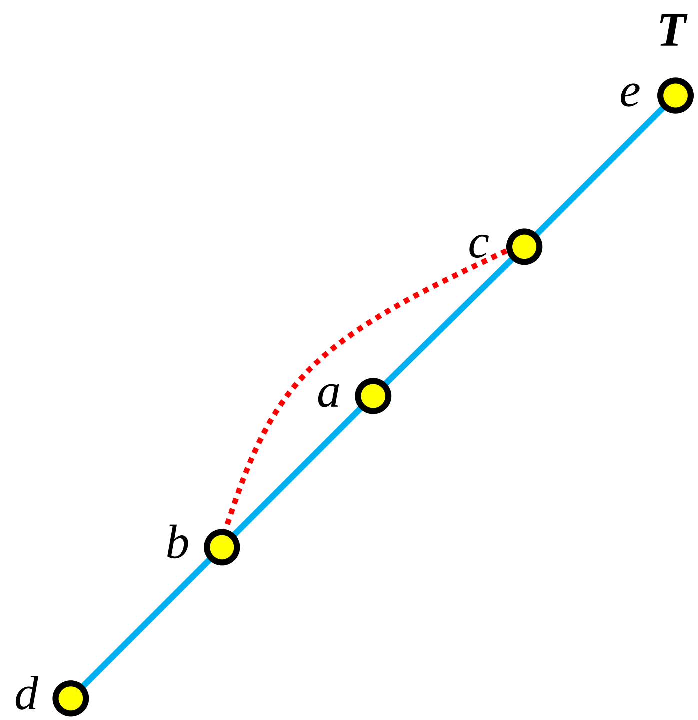


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$
$P(b) \longrightarrow N(b) = \{a, c, d\}$
$P(d)$

v	a	b	c	d	e	f	g
$PE(v)$	3	4	2	0	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$

$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$

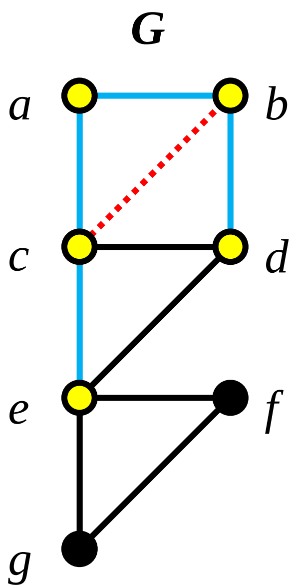
$P(a) \longrightarrow N(a) = \{\mathbf{b}, c\}$

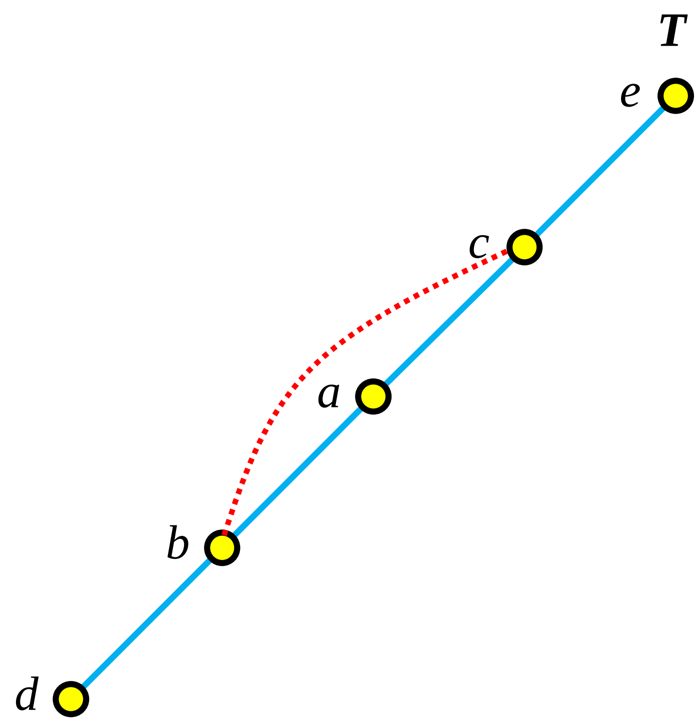
$P(b) \longrightarrow N(b) = \{\mathbf{a}, \mathbf{c}, \mathbf{d}\}$

$P(d)$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
PE(*v*)	3	4	2	5	1	0	0
PS(*v*)	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

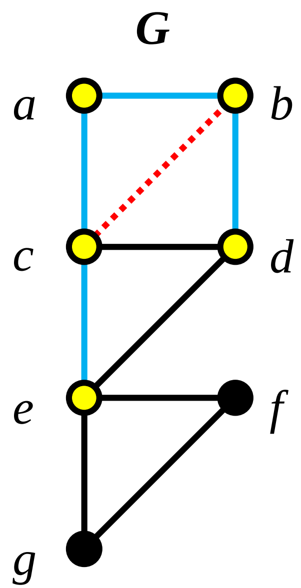
$P(a) \longrightarrow N(a) = \{b, c\}$

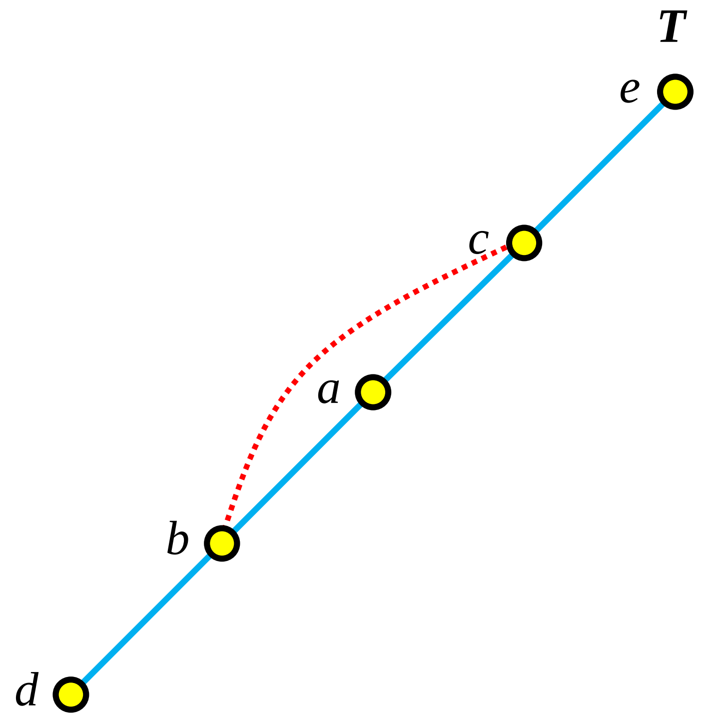
$P(b) \longrightarrow N(b) = \{a, c, d\}$

$P(d) \longrightarrow N(d) = \{b, c, e\}$

v	**a**	**b**	**c**	**d**	**e**	**f**	**g**
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

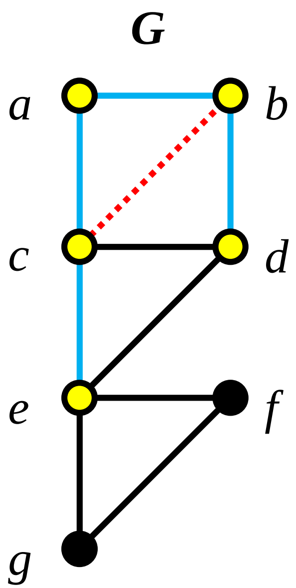
$P(a) \longrightarrow N(a) = \{b, c\}$

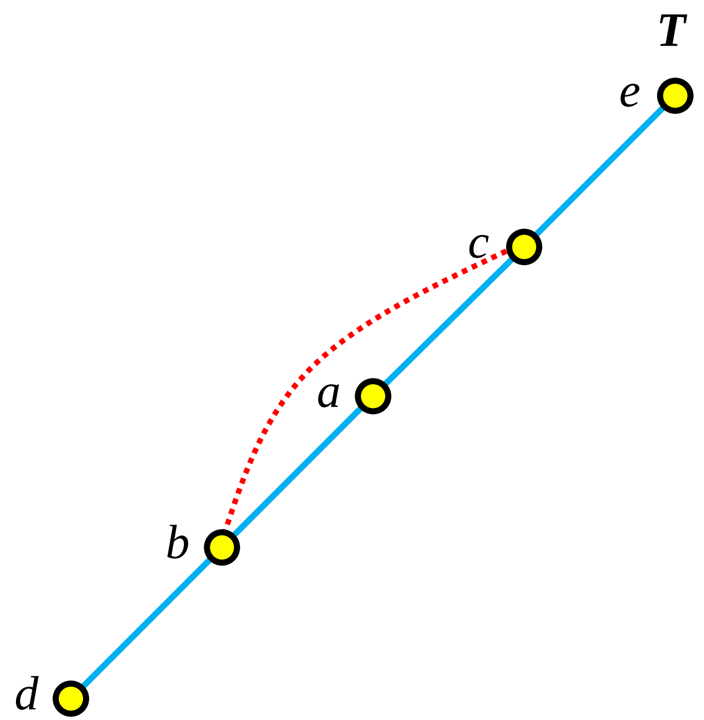
$P(b) \longrightarrow N(b) = \{a, c, d\}$

$P(d) \longrightarrow N(d) = \{b, c, e\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

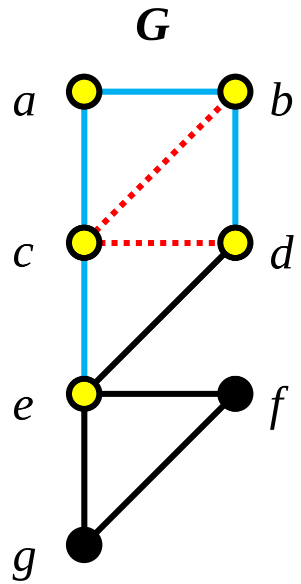


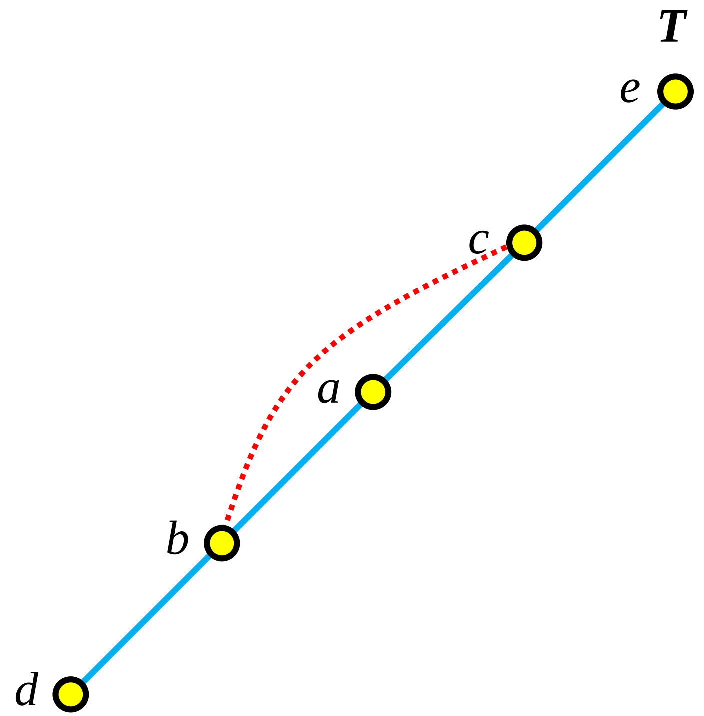


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$
$P(b) \longrightarrow N(b) = \{a, c, d\}$
$P(d) \longrightarrow N(d) = \{b, c, e\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

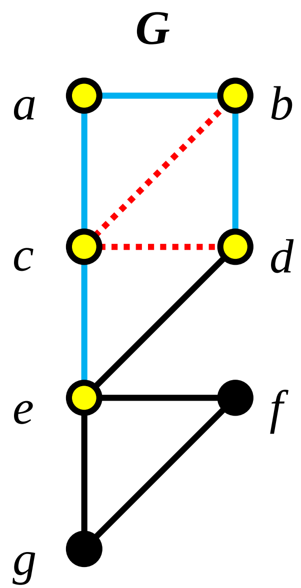
$P(a) \longrightarrow N(a) = \{b, c\}$

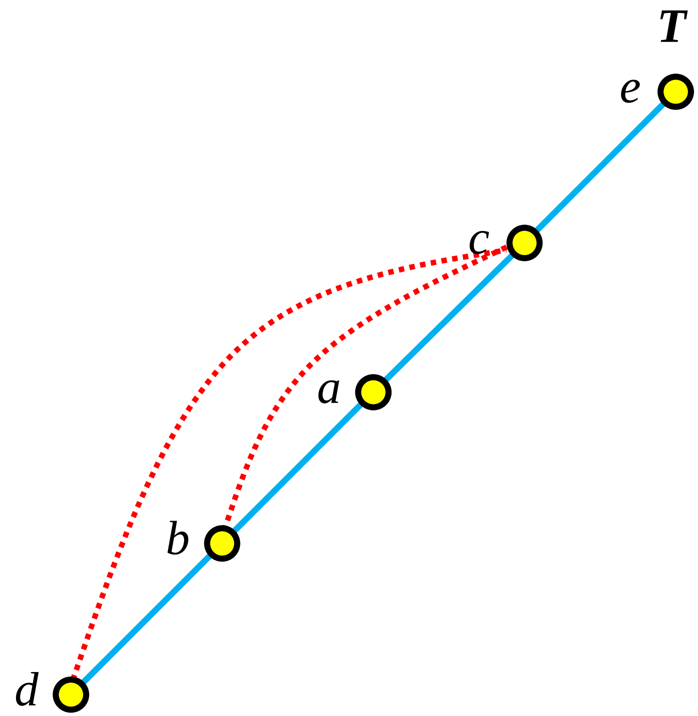
$P(b) \longrightarrow N(b) = \{a, c, d\}$

$P(d) \longrightarrow N(d) = \{b, c, e\}$

v	**a**	**b**	**c**	**d**	**e**	**f**	**g**
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$

$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$

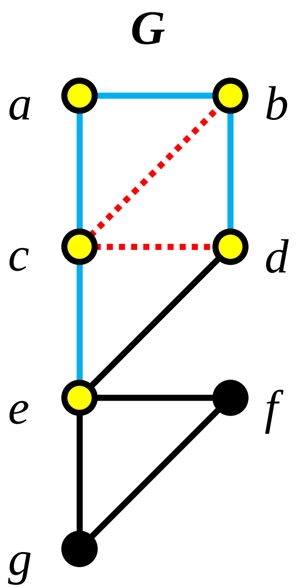
$P(a) \longrightarrow N(a) = \{\mathbf{b}, c\}$

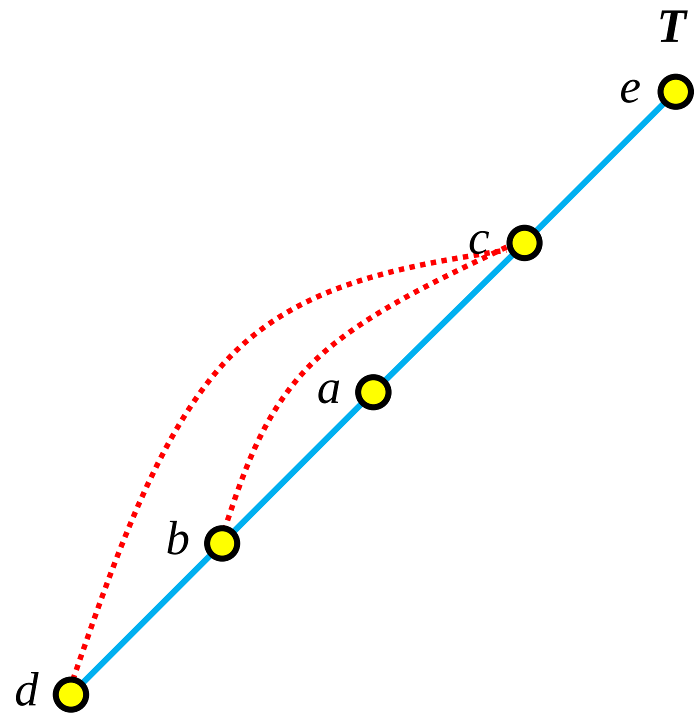
$P(b) \longrightarrow N(b) = \{\mathbf{a}, \mathbf{c}, \mathbf{d}\}$

$P(d) \longrightarrow N(d) = \{\mathbf{b}, \mathbf{c}, e\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade

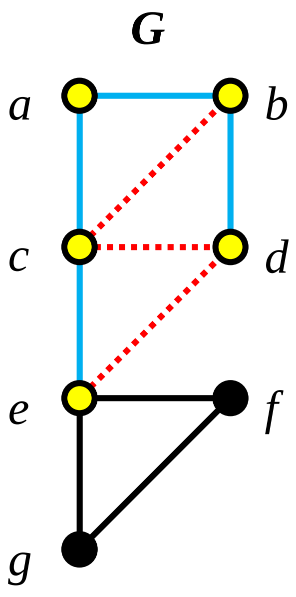


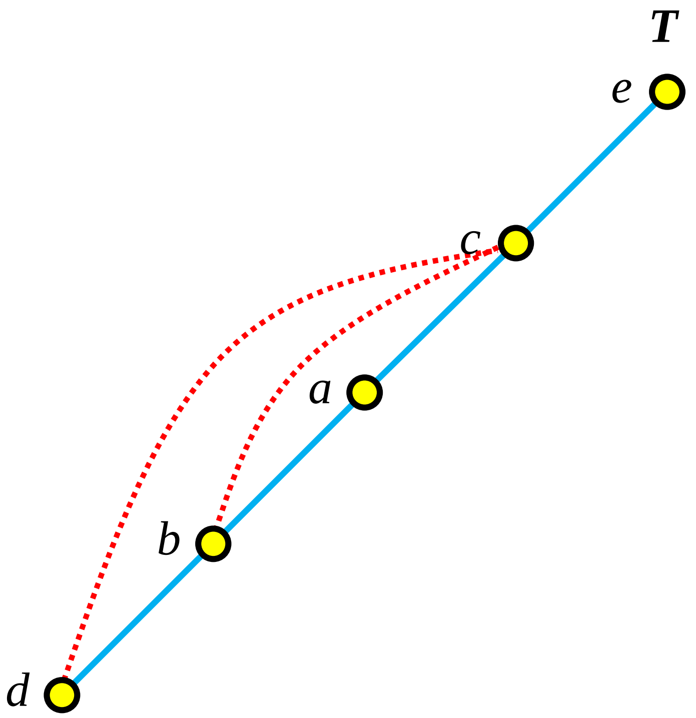


$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$
$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$
$P(a) \longrightarrow N(a) = \{\mathbf{b}, c\}$
$P(b) \longrightarrow N(b) = \{\mathbf{a}, \mathbf{c}, \mathbf{d}\}$
$P(d) \longrightarrow N(d) = \{\mathbf{b}, \mathbf{c}, \mathbf{e}\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
PE(*v*)	3	4	2	5	1	0	0
PS(*v*)	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

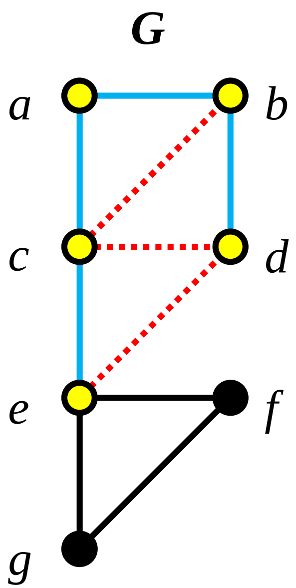
$P(a) \longrightarrow N(a) = \{b, c\}$

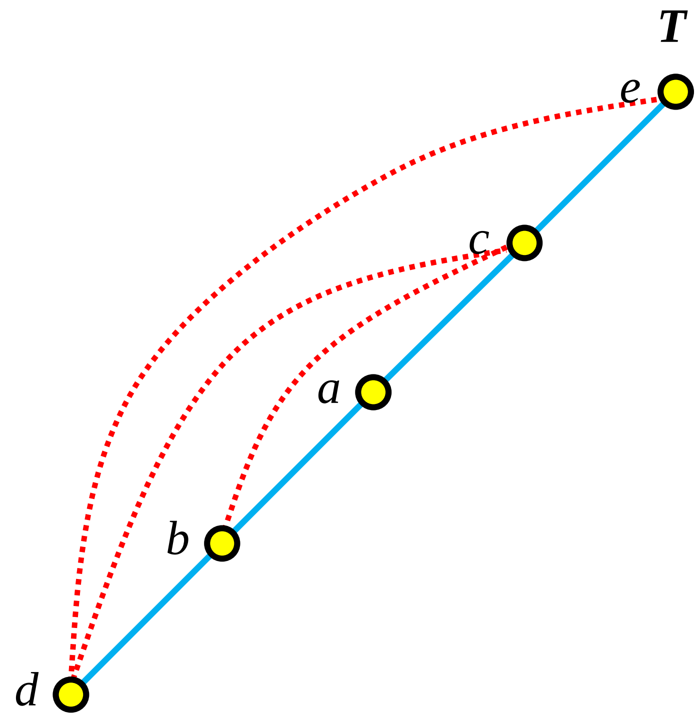
$P(b) \longrightarrow N(b) = \{a, c, d\}$

$P(d) \longrightarrow N(d) = \{b, c, e\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$

$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$

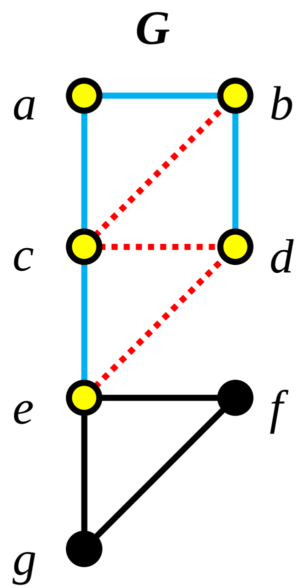
$P(a) \longrightarrow N(a) = \{\mathbf{b}, c\}$

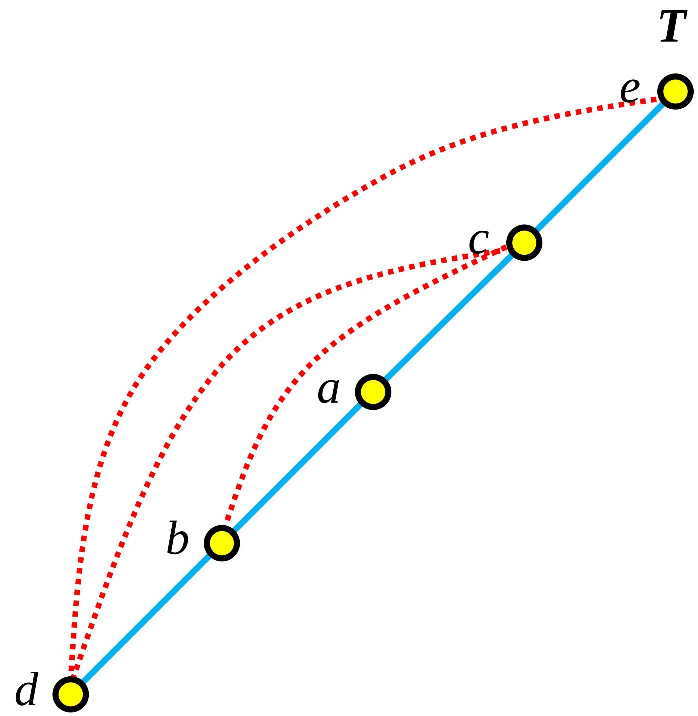
$P(b) \longrightarrow N(b) = \{\mathbf{a}, \mathbf{c}, \mathbf{d}\}$

$P(d) \longrightarrow N(d) = \{\mathbf{b}, \mathbf{c}, \mathbf{e}\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
PE(*v*)	3	4	2	5	1	0	0
PS(*v*)	0	0	0	0	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

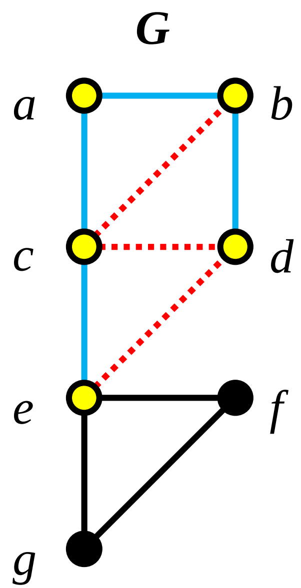
$P(a) \longrightarrow N(a) = \{b, c\}$

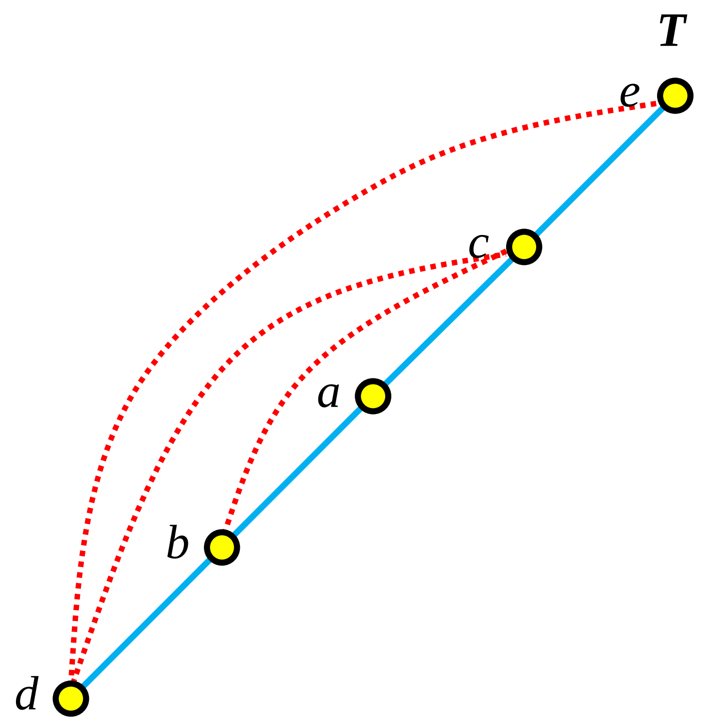
$P(b) \longrightarrow N(b) = \{a, c, d\}$

$P(d) \longrightarrow N(d) = \{b, c, e\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	0	0	0	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{\mathbf{c}, d, f, g\}$

$P(c) \longrightarrow N(c) = \{\mathbf{a}, b, d, e\}$

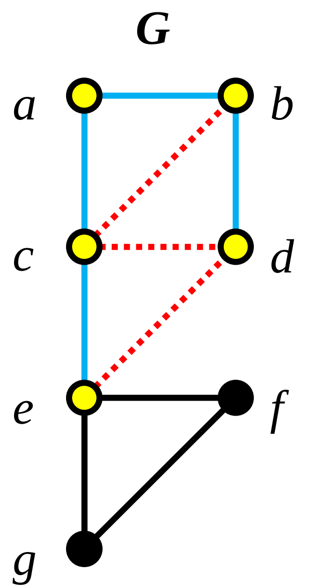
$P(a) \longrightarrow N(a) = \{\mathbf{b}, c\}$

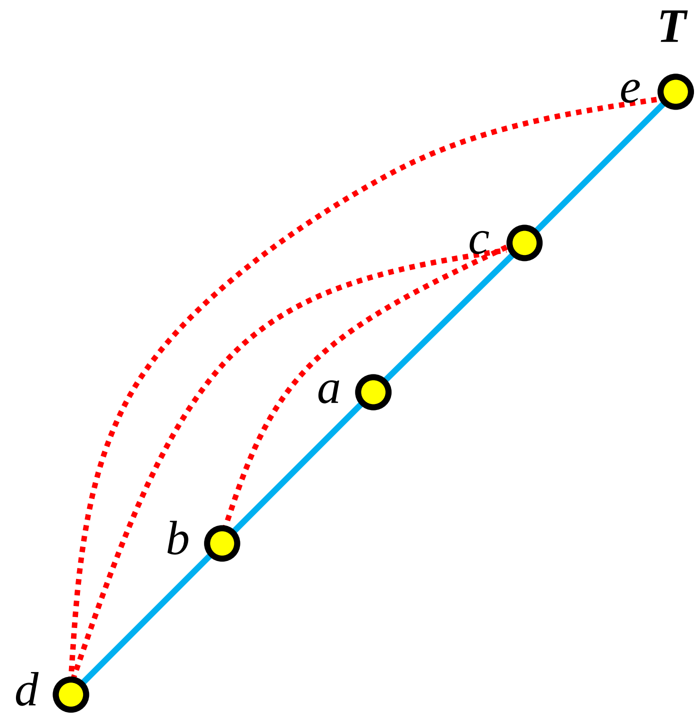
$P(b) \longrightarrow N(b) = \{\mathbf{a}, \mathbf{c}, \mathbf{d}\}$

$P(d) \longrightarrow N(d) = \{\mathbf{b}, \mathbf{c}, \mathbf{e}\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	0	7	0	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

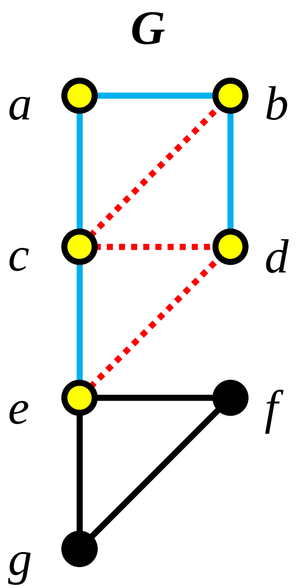
$P(a) \longrightarrow N(a) = \{b, c\}$

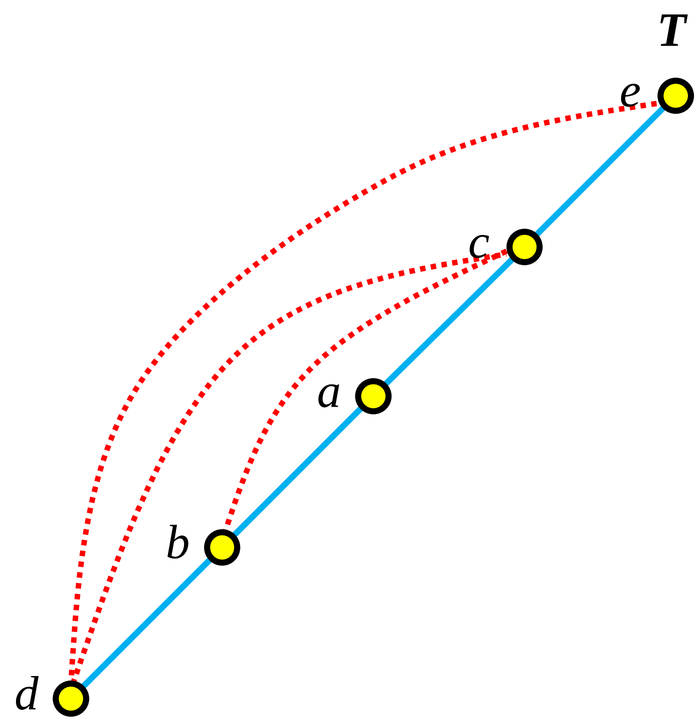
$P(b) \longrightarrow N(b) = \{a, c, d\}$

$P(d) \longrightarrow N(d) = \{b, c, e\}$

v	**a**	**b**	**c**	**d**	**e**	**f**	**g**
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	0	7	0	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

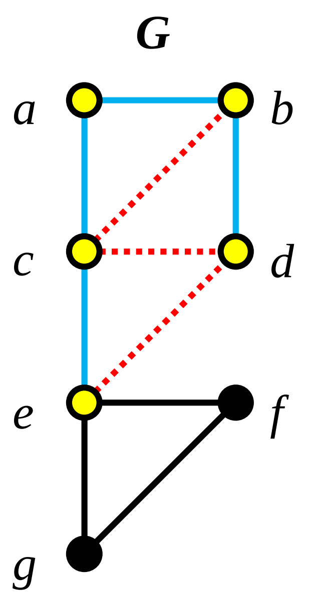
$P(a) \longrightarrow N(a) = \{b, c\}$

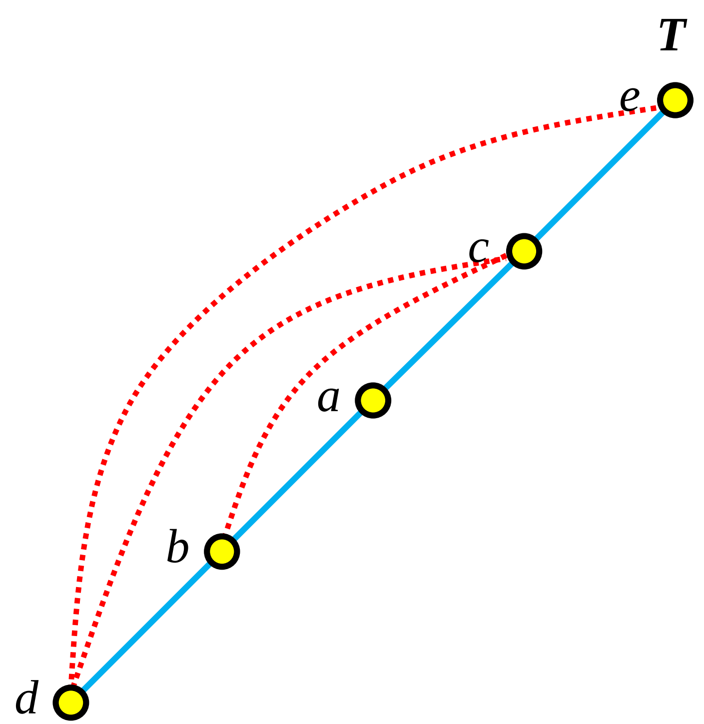
$P(b) \longrightarrow N(b) = \{a, c, d\}$

$P(d) \longrightarrow N(d) = \{b, c, e\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
PE(*v*)	3	4	2	5	1	0	0
PS(*v*)	8	7	0	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, \mathbf{b}, d, e\}$

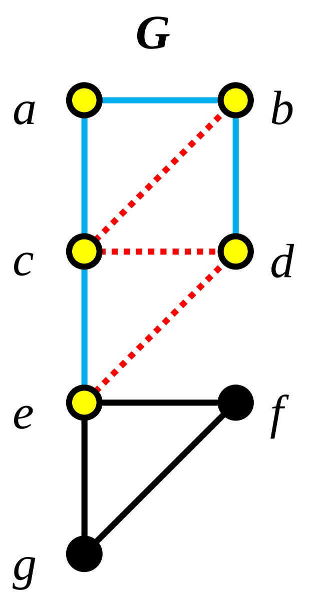
$P(a) \longrightarrow N(a) = \{\mathbf{b}, c\}$

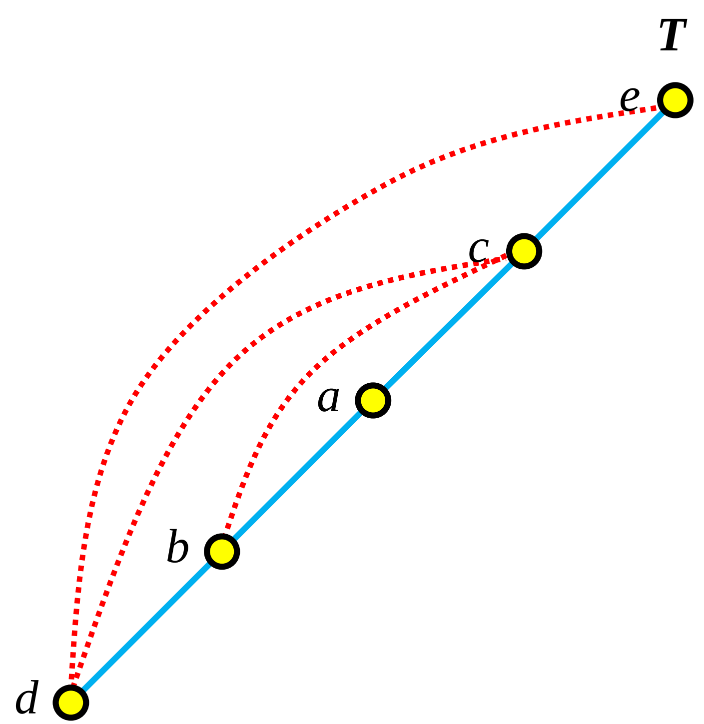
$P(b) \longrightarrow N(b) = \{\mathbf{a}, \mathbf{c}, \mathbf{d}\}$

$P(d) \longrightarrow N(d) = \{\mathbf{b}, \mathbf{c}, \mathbf{e}\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	8	7	0	6	0	0	0

Busca em profundidade

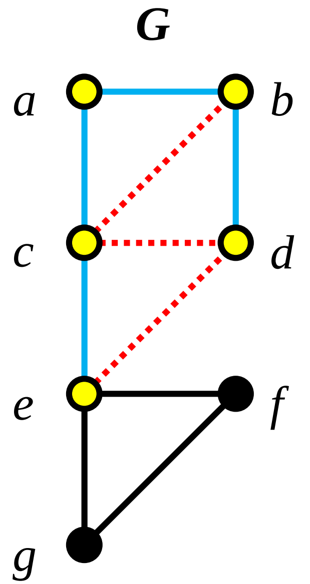


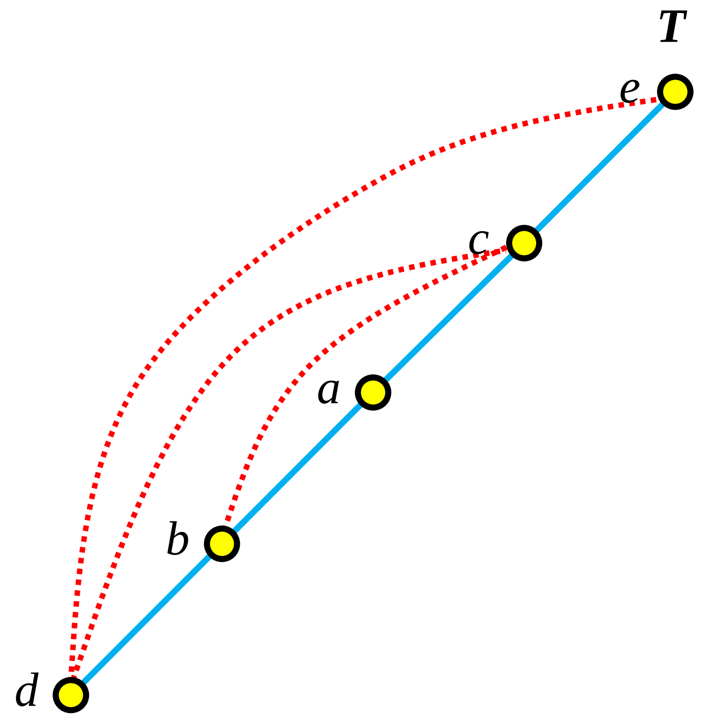


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, \mathbf{b}, \mathbf{d}, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$
$P(b) \longrightarrow N(b) = \{a, c, d\}$
$P(d) \longrightarrow N(d) = \{b, c, e\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	8	7	0	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, \mathbf{b}, \mathbf{d}, e\}$

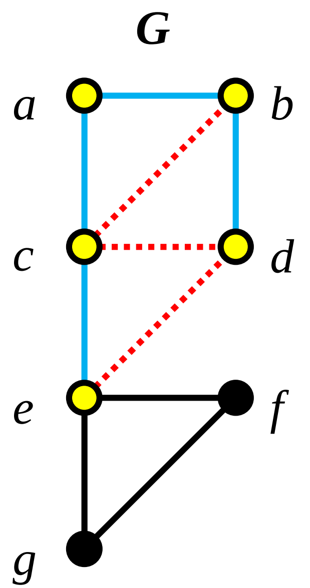
$P(a) \longrightarrow N(a) = \{b, c\}$

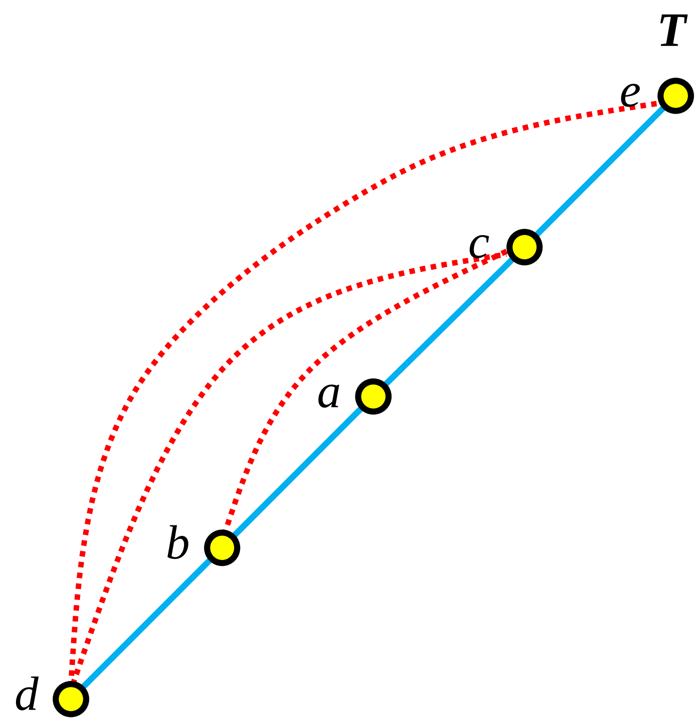
$P(b) \longrightarrow N(b) = \{a, c, d\}$

$P(d) \longrightarrow N(d) = \{b, c, e\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	8	7	0	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, \mathbf{b}, \mathbf{d}, e\}$

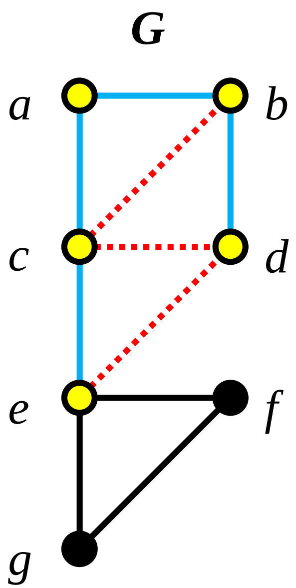
$P(a) \longrightarrow N(a) = \{b, c\}$

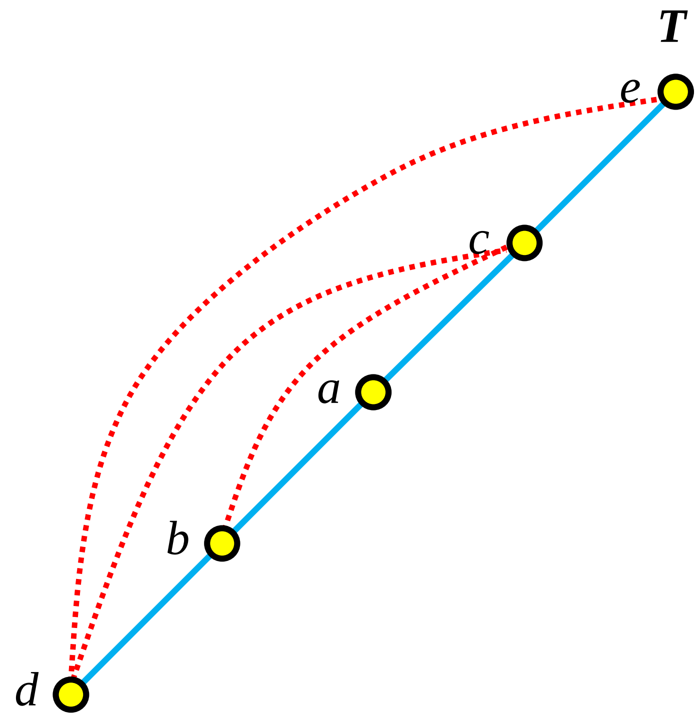
$P(b) \longrightarrow N(b) = \{a, c, d\}$

$P(d) \longrightarrow N(d) = \{b, c, e\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, \mathbf{d}, f, g\}$

$P(c) \longrightarrow N(c) = \{\mathbf{a}, \mathbf{b}, \mathbf{d}, \mathbf{e}\}$

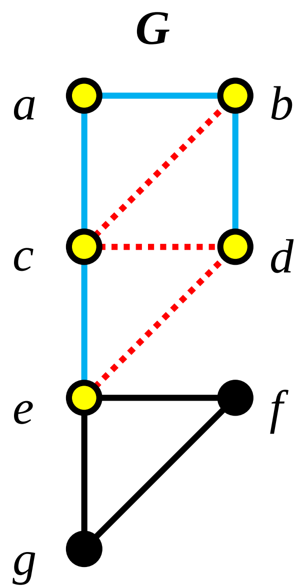
$P(a) \longrightarrow N(a) = \{\mathbf{b}, \mathbf{c}\}$

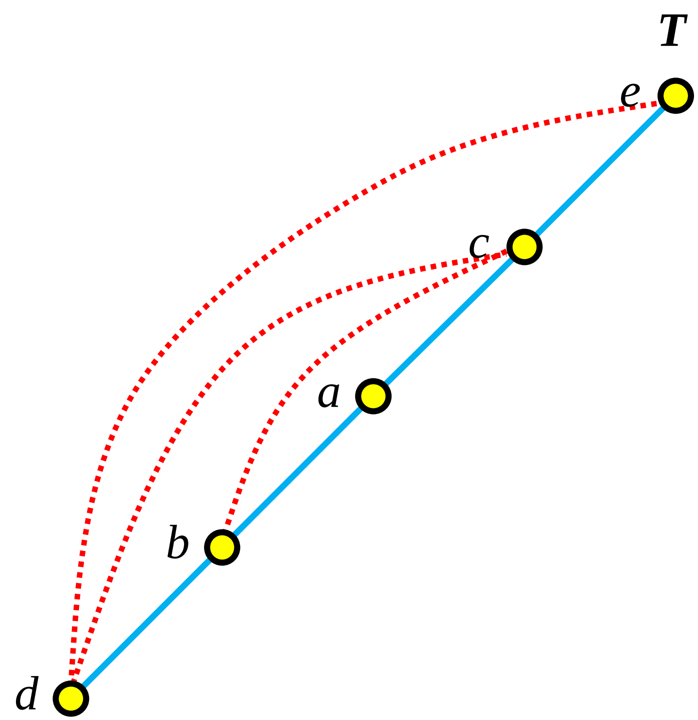
$P(b) \longrightarrow N(b) = \{\mathbf{a}, \mathbf{c}, \mathbf{d}\}$

$P(d) \longrightarrow N(d) = \{\mathbf{b}, \mathbf{c}, \mathbf{e}\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

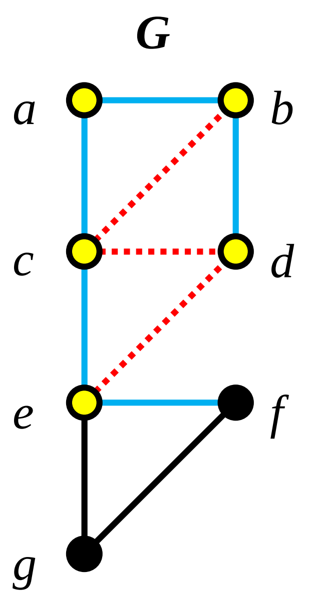
$P(a) \longrightarrow N(a) = \{b, c\}$

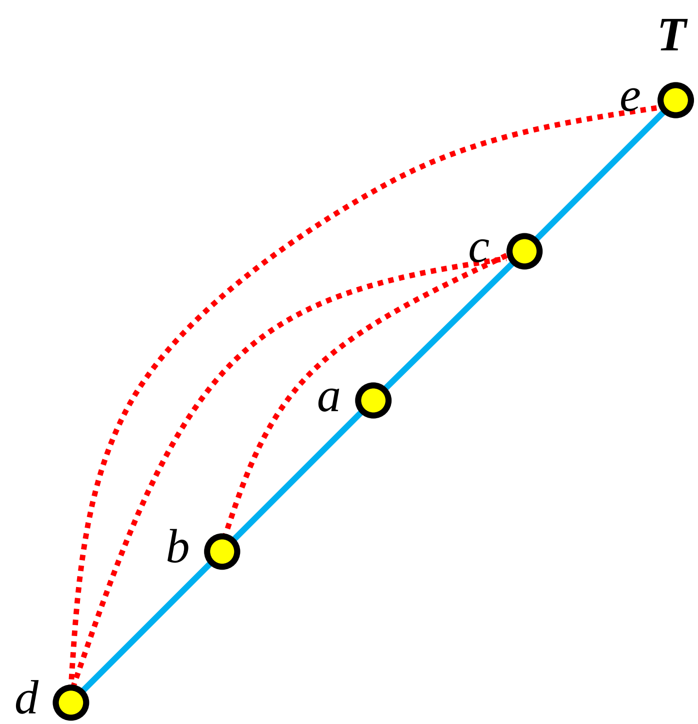
$P(b) \longrightarrow N(b) = \{a, c, d\}$

$P(d) \longrightarrow N(d) = \{b, c, e\}$

v	**a**	**b**	**c**	**d**	**e**	**f**	**g**
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, \mathbf{d}, f, g\}$

$P(c) \longrightarrow N(c) = \{a, \mathbf{b}, \mathbf{d}, e\}$

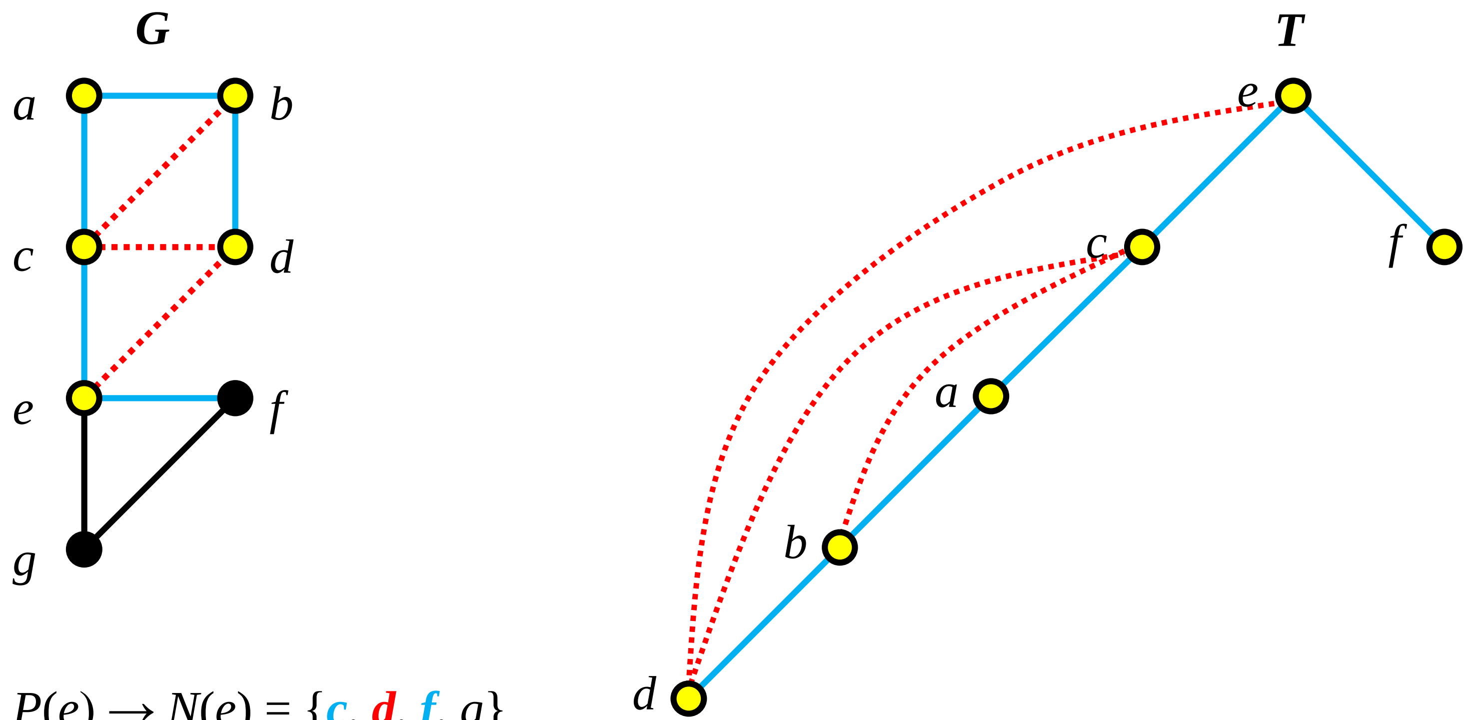
$P(a) \longrightarrow N(a) = \{b, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

$P(d) \longrightarrow N(d) = \{b, c, e\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade



$P(e) \longrightarrow N(e) = \{\mathbf{c}, \mathbf{d}, \mathbf{f}, g\}$

$P(c) \longrightarrow N(c) = \{\mathbf{a}, \mathbf{b}, \mathbf{d}, \mathbf{e}\}$

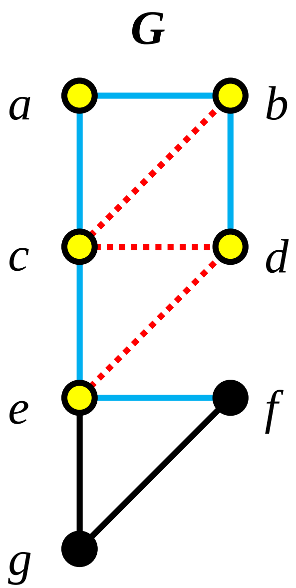
$P(a) \longrightarrow N(a) = \{\mathbf{b}, \mathbf{c}\}$

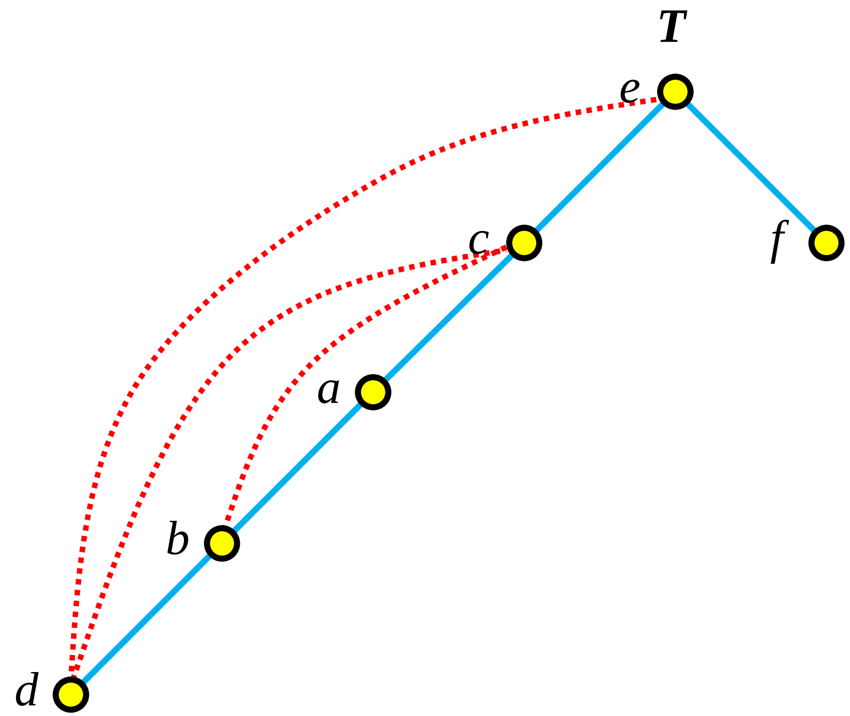
$P(b) \longrightarrow N(b) = \{\mathbf{a}, \mathbf{c}, \mathbf{d}\}$

$P(d) \longrightarrow N(d) = \{\mathbf{b}, \mathbf{c}, \mathbf{e}\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade

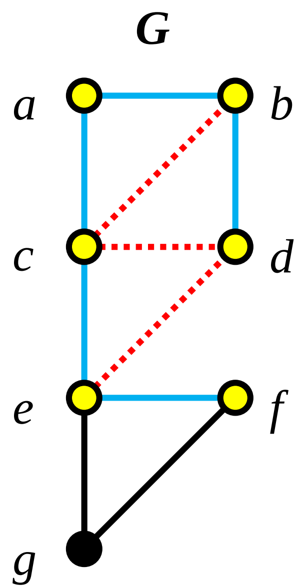


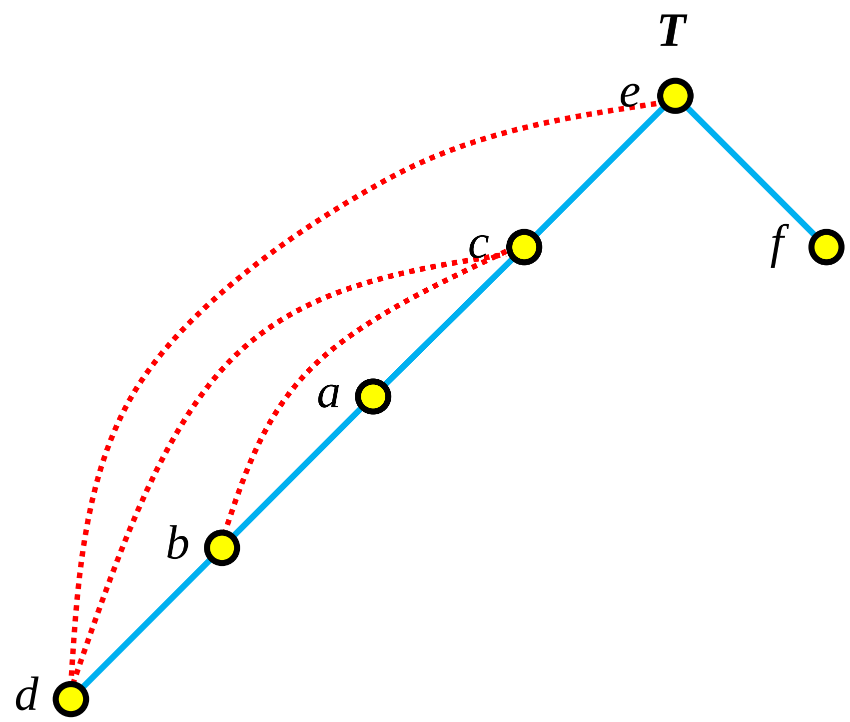


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$
$P(b) \longrightarrow N(b) = \{a, c, d\}$
$P(d) \longrightarrow N(d) = \{b, c, e\}$
$P(f)$

v	**a**	**b**	**c**	**d**	**e**	**f**	**g**
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade

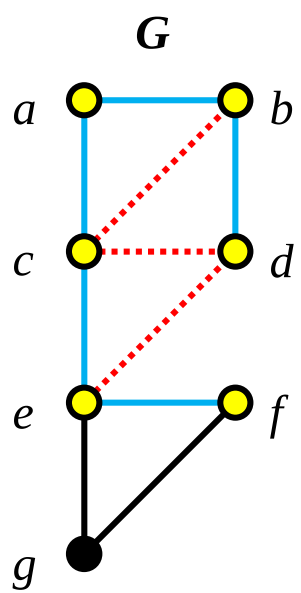


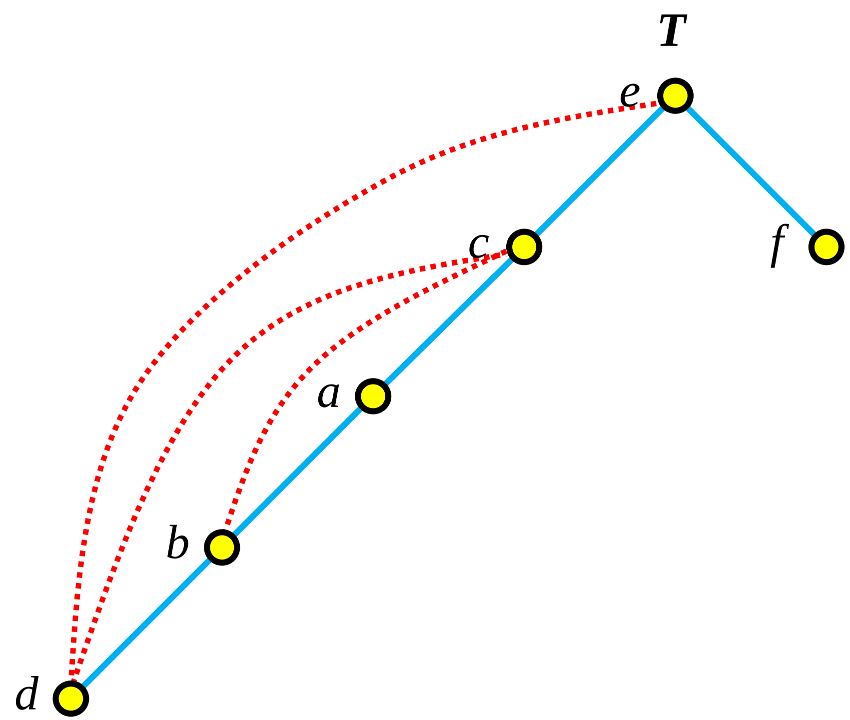


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$
$P(b) \longrightarrow N(b) = \{a, c, d\}$
$P(d) \longrightarrow N(d) = \{b, c, e\}$
$P(f)$

v	**a**	**b**	**c**	**d**	**e**	**f**	**g**
$PE(v)$	3	4	2	5	1	0	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade

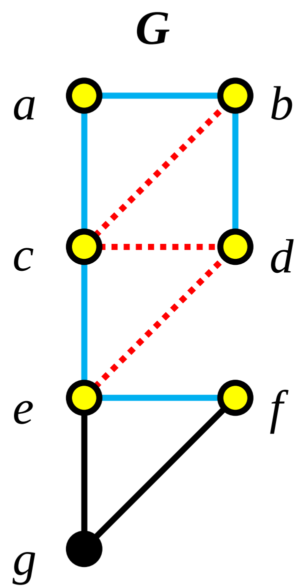


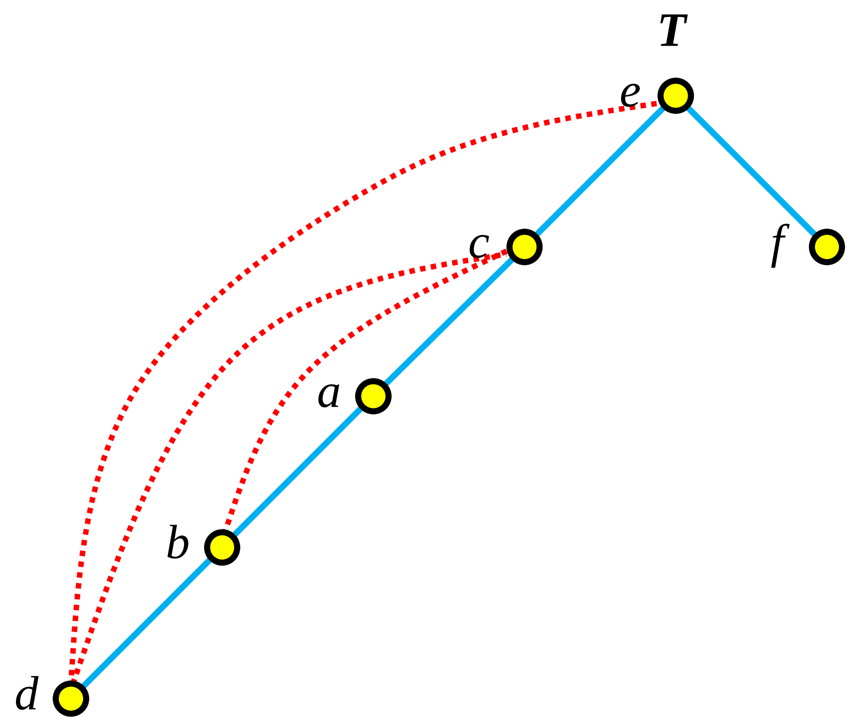


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$
$P(b) \longrightarrow N(b) = \{a, c, d\}$
$P(d) \longrightarrow N(d) = \{b, c, e\}$
$P(f)$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

$P(a) \longrightarrow N(a) = \{b, c\}$

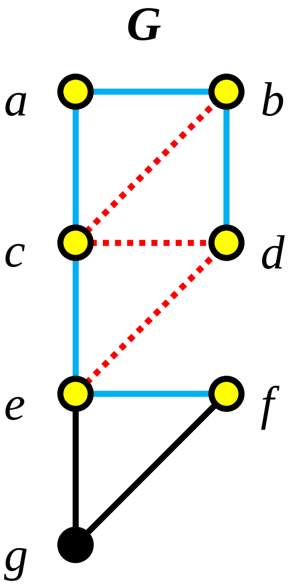
$P(b) \longrightarrow N(b) = \{a, c, d\}$

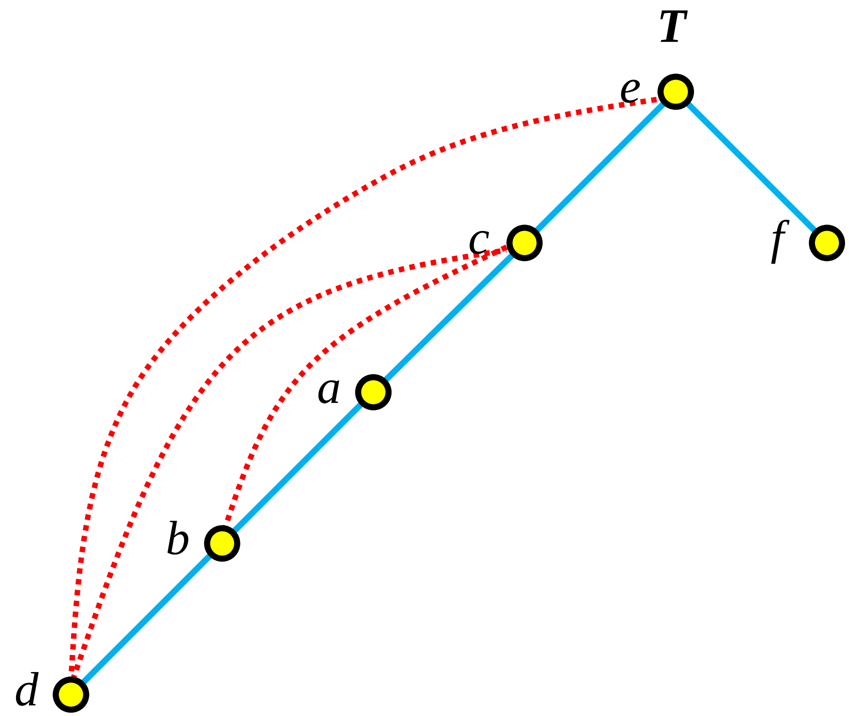
$P(d) \longrightarrow N(d) = \{b, c, e\}$

$P(f) \longrightarrow N(f) = \{e, g\}$

v	a	b	c	d	e	f	g
$PE(v)$	3	4	2	5	1	10	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

$P(a) \longrightarrow N(a) = \{b, c\}$

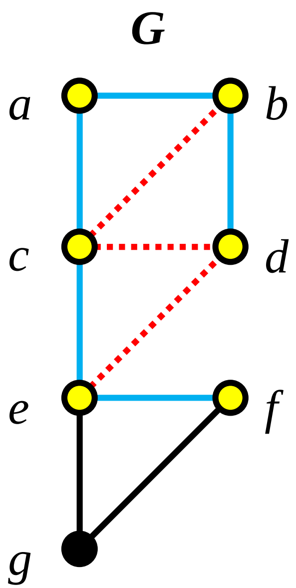
$P(b) \longrightarrow N(b) = \{a, c, d\}$

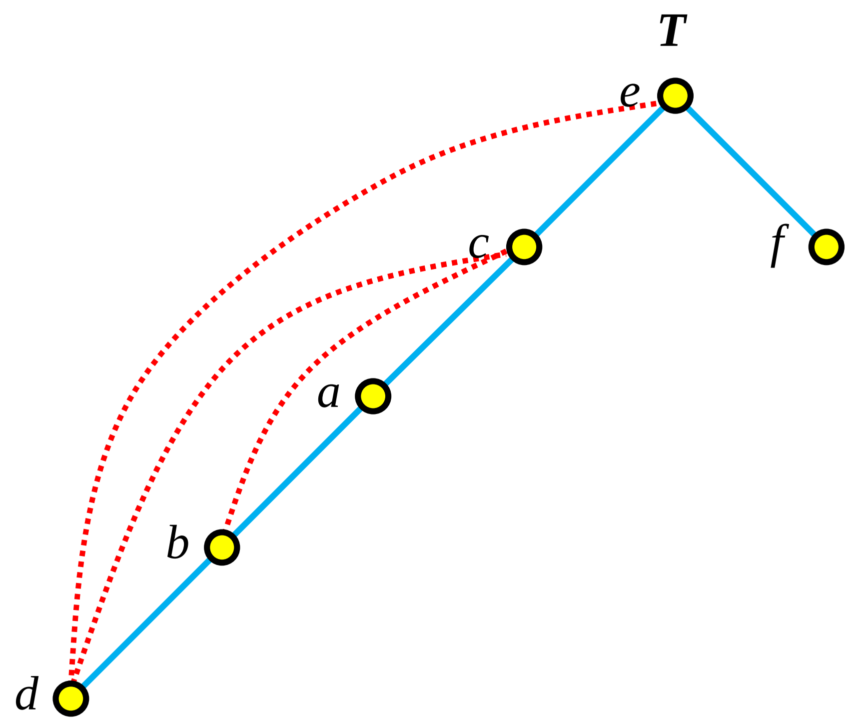
$P(d) \longrightarrow N(d) = \{b, c, e\}$

$P(f) \longrightarrow N(f) = \{e, g\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

$P(a) \longrightarrow N(a) = \{b, c\}$

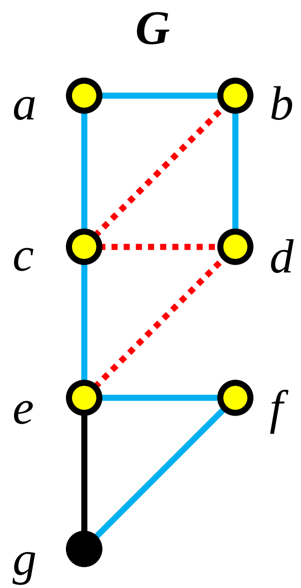
$P(b) \longrightarrow N(b) = \{a, c, d\}$

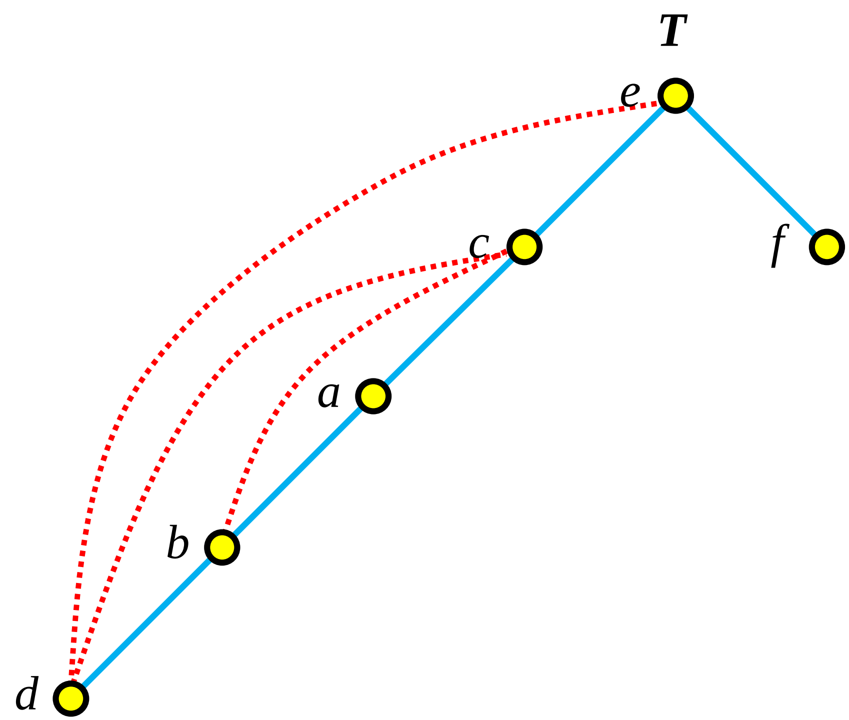
$P(d) \longrightarrow N(d) = \{b, c, e\}$

$P(f) \longrightarrow N(f) = \{e, g\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade

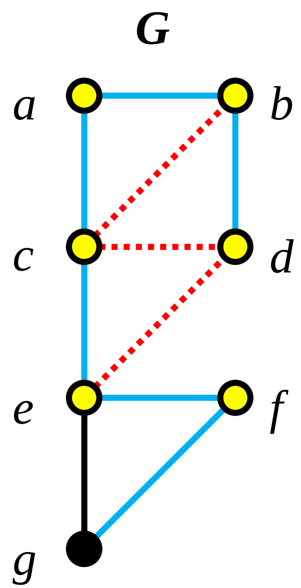


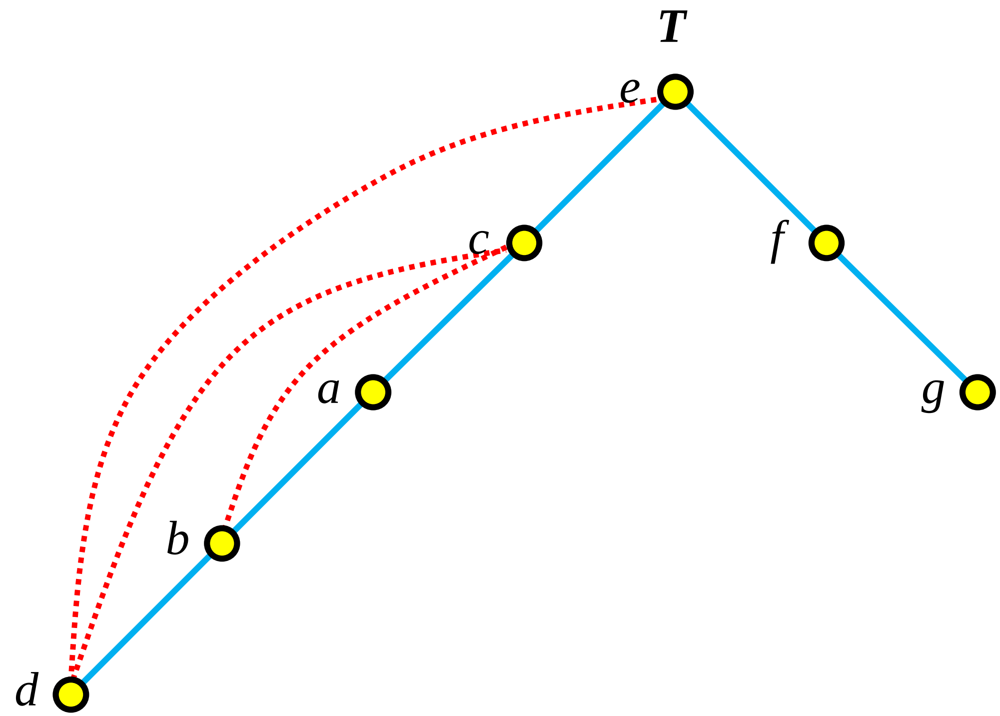


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$
$P(b) \longrightarrow N(b) = \{a, c, d\}$
$P(d) \longrightarrow N(d) = \{b, c, e\}$
$P(f) \longrightarrow N(f) = \{e, g\}$

v	a	b	c	d	e	f	g
$PE(v)$	3	4	2	5	1	10	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

$P(a) \longrightarrow N(a) = \{b, c\}$

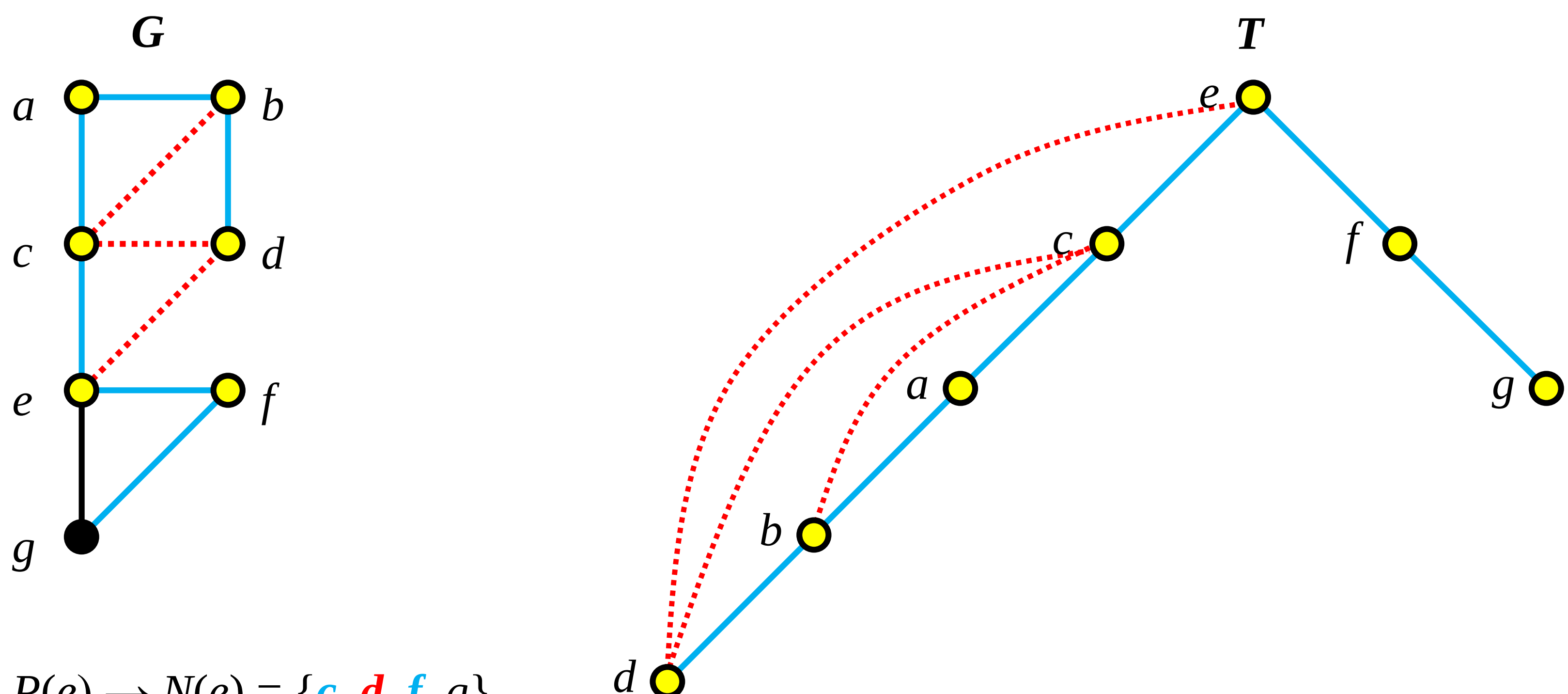
$P(b) \longrightarrow N(b) = \{a, c, d\}$

$P(d) \longrightarrow N(d) = \{b, c, e\}$

$P(f) \longrightarrow N(f) = \{e, g\}$

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*
$PE(v)$	3	4	2	5	1	10	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade



$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

$P(a) \longrightarrow N(a) = \{b, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

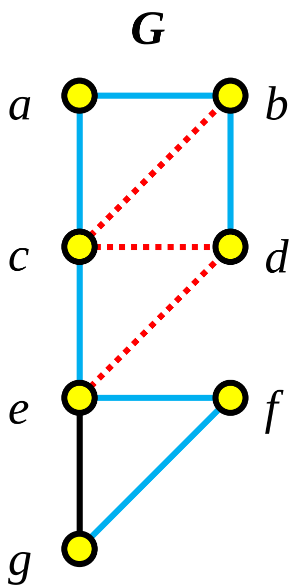
$P(d) \longrightarrow N(d) = \{b, c, e\}$

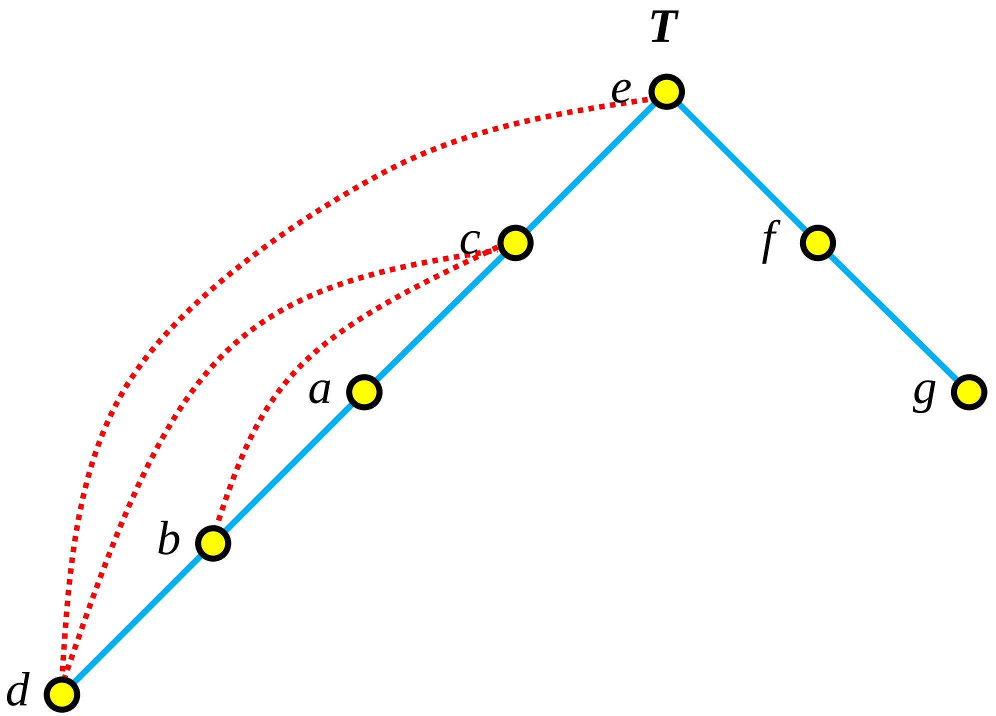
$P(f) \longrightarrow N(f) = \{e, g\}$

$P(g)$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade

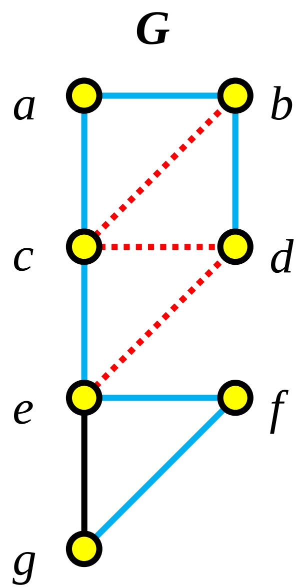


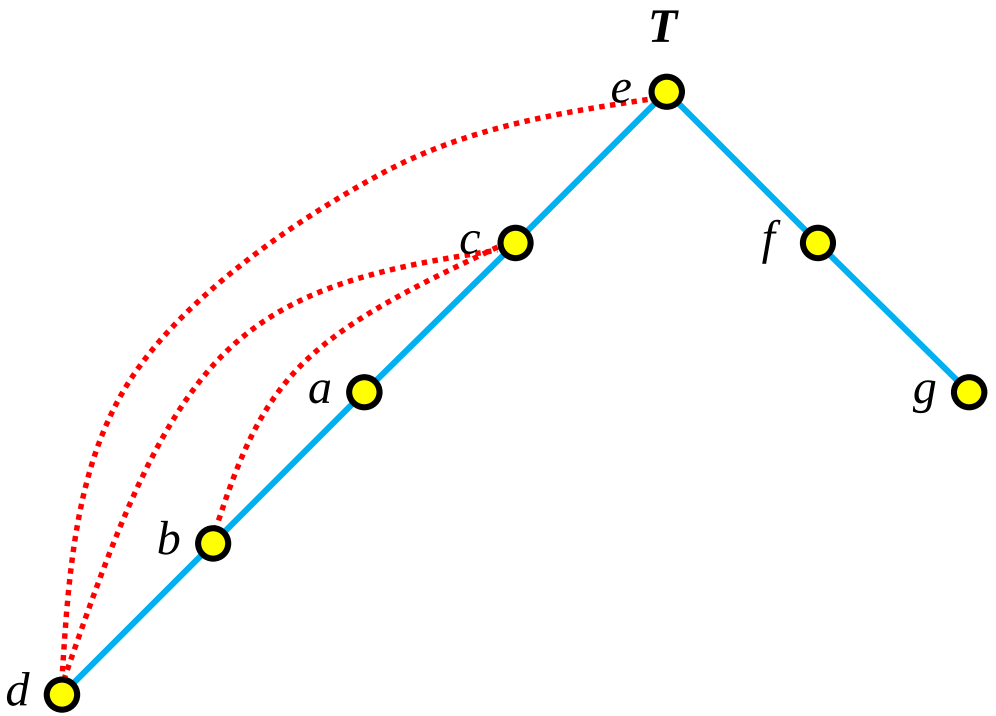


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$
$P(b) \longrightarrow N(b) = \{a, c, d\}$
$P(d) \longrightarrow N(d) = \{b, c, e\}$
$P(f) \longrightarrow N(f) = \{e, g\}$
$P(g)$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	0
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

$P(a) \longrightarrow N(a) = \{b, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

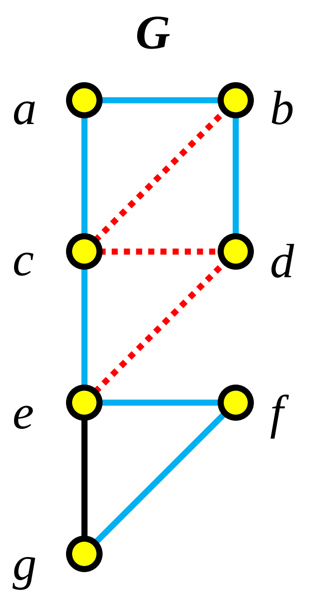
$P(d) \longrightarrow N(d) = \{b, c, e\}$

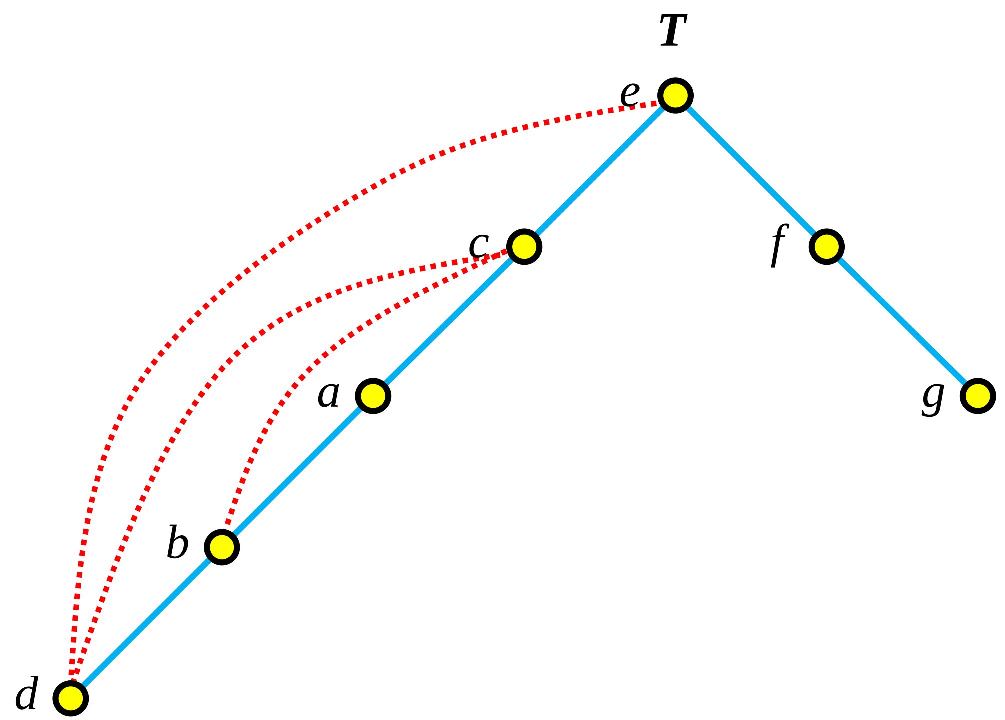
$P(f) \longrightarrow N(f) = \{e, g\}$

$P(g)$

v	a	b	c	d	e	f	g
$PE(v)$	3	4	2	5	1	10	11
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade

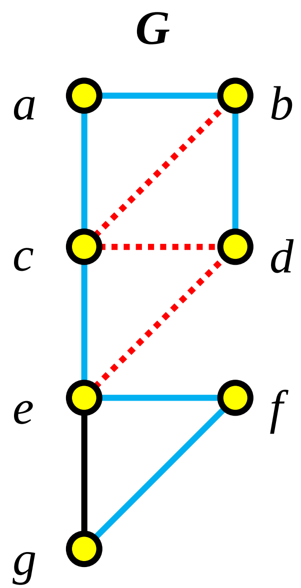


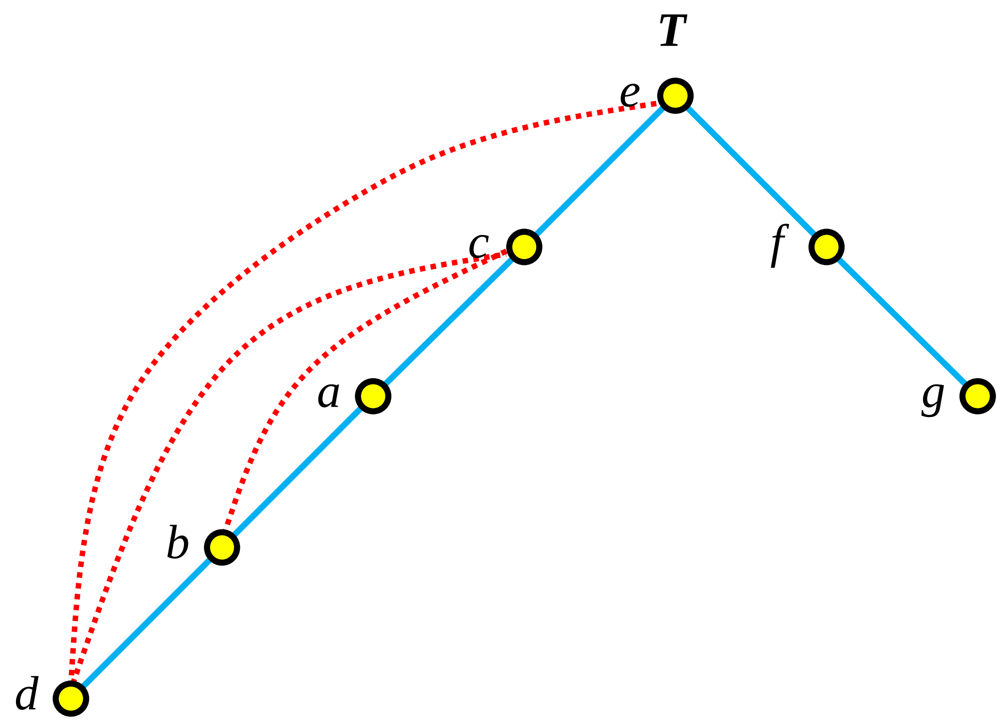


$P(e) \longrightarrow N(e) = \{c, d, f, g\}$
$P(c) \longrightarrow N(c) = \{a, b, d, e\}$
$P(a) \longrightarrow N(a) = \{b, c\}$
$P(b) \longrightarrow N(b) = \{a, c, d\}$
$P(d) \longrightarrow N(d) = \{b, c, e\}$
$P(f) \longrightarrow N(f) = \{e, g\}$
$P(g) \longrightarrow N(g) = \{e, f\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	11
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

$P(a) \longrightarrow N(a) = \{b, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

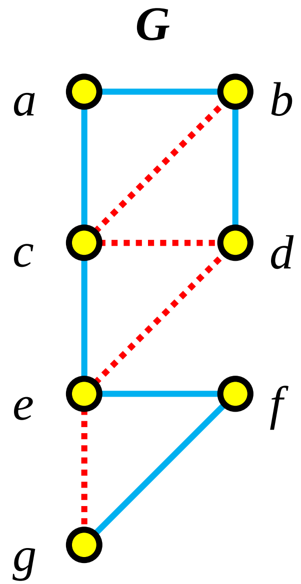
$P(d) \longrightarrow N(d) = \{b, c, e\}$

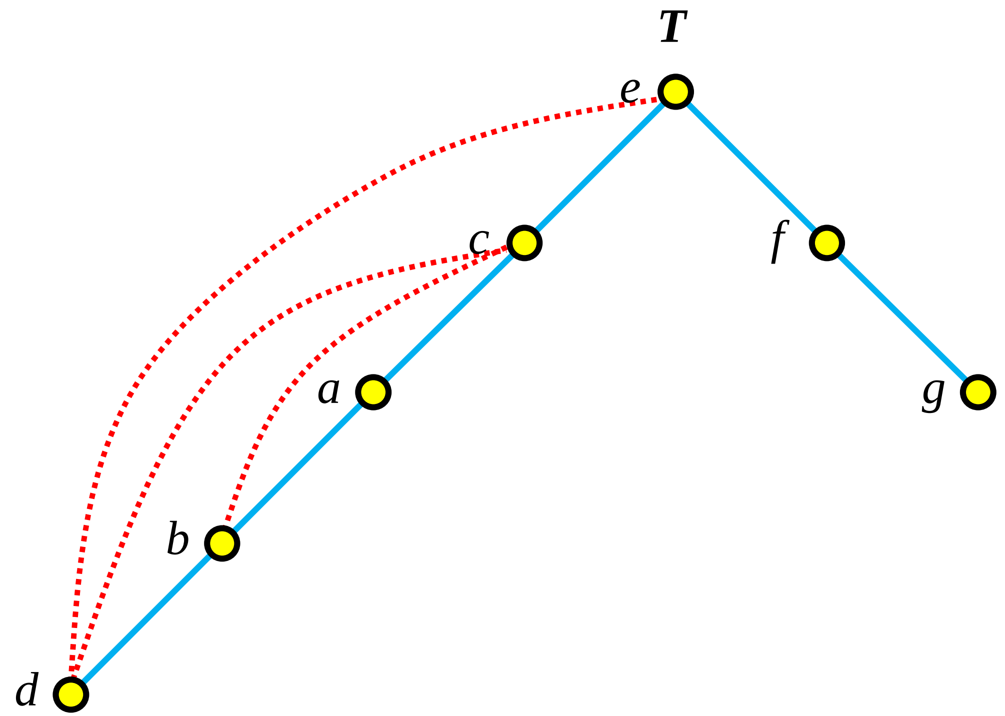
$P(f) \longrightarrow N(f) = \{e, g\}$

$P(g) \longrightarrow N(g) = \{e, f\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	11
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

$P(a) \longrightarrow N(a) = \{b, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

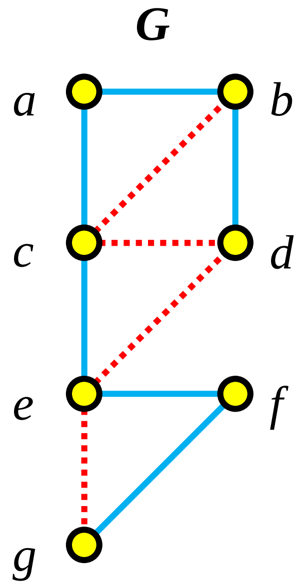
$P(d) \longrightarrow N(d) = \{b, c, e\}$

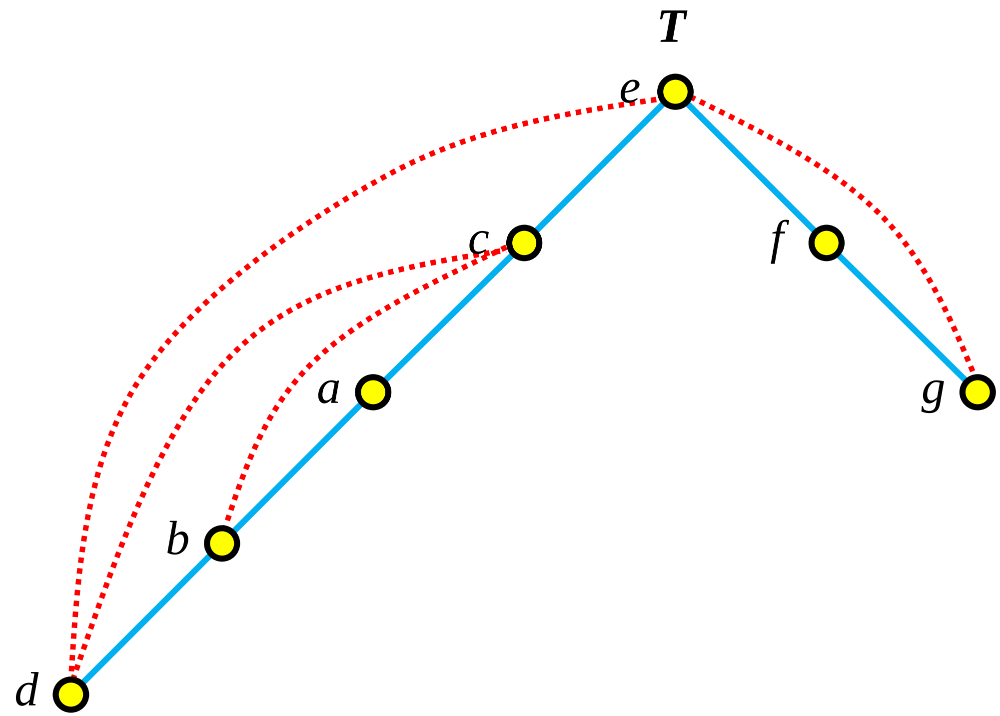
$P(f) \longrightarrow N(f) = \{e, g\}$

$P(g) \longrightarrow N(g) = \{e, f\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	11
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

$P(a) \longrightarrow N(a) = \{b, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

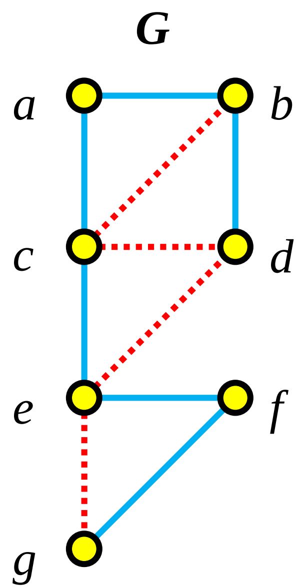
$P(d) \longrightarrow N(d) = \{b, c, e\}$

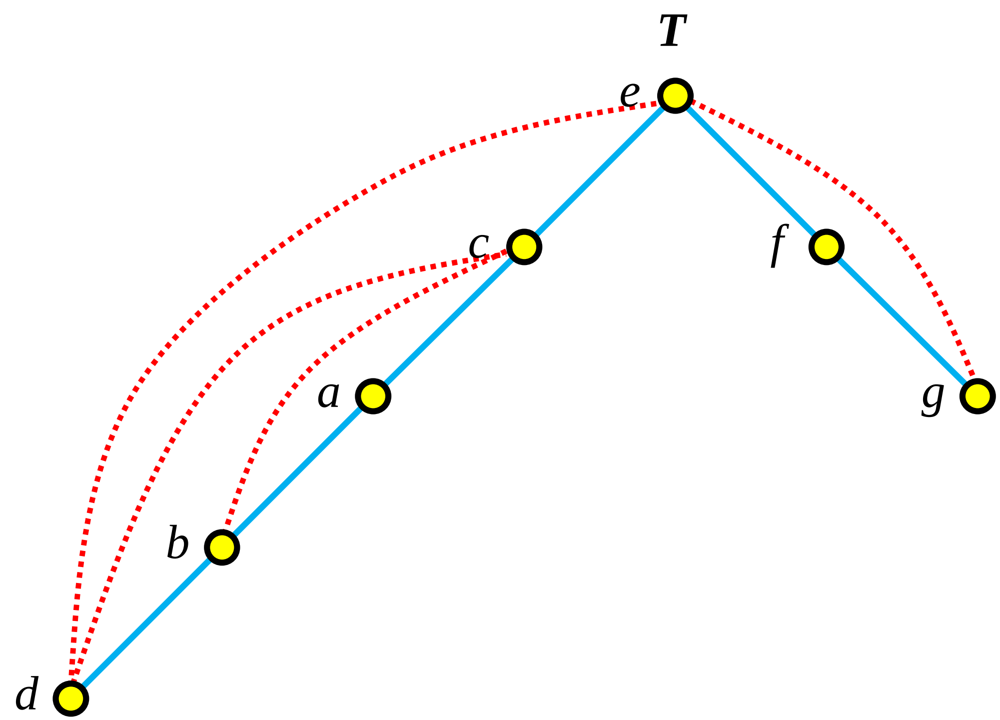
$P(f) \longrightarrow N(f) = \{e, g\}$

$P(g) \longrightarrow N(g) = \{e, f\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	11
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

$P(a) \longrightarrow N(a) = \{b, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

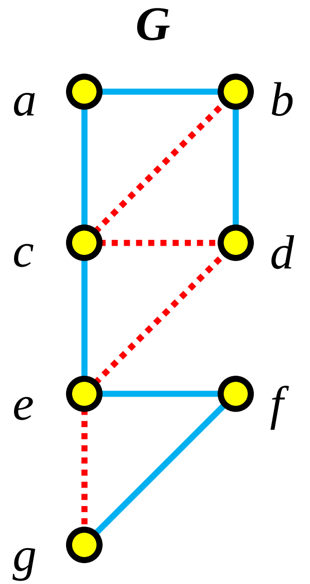
$P(d) \longrightarrow N(d) = \{b, c, e\}$

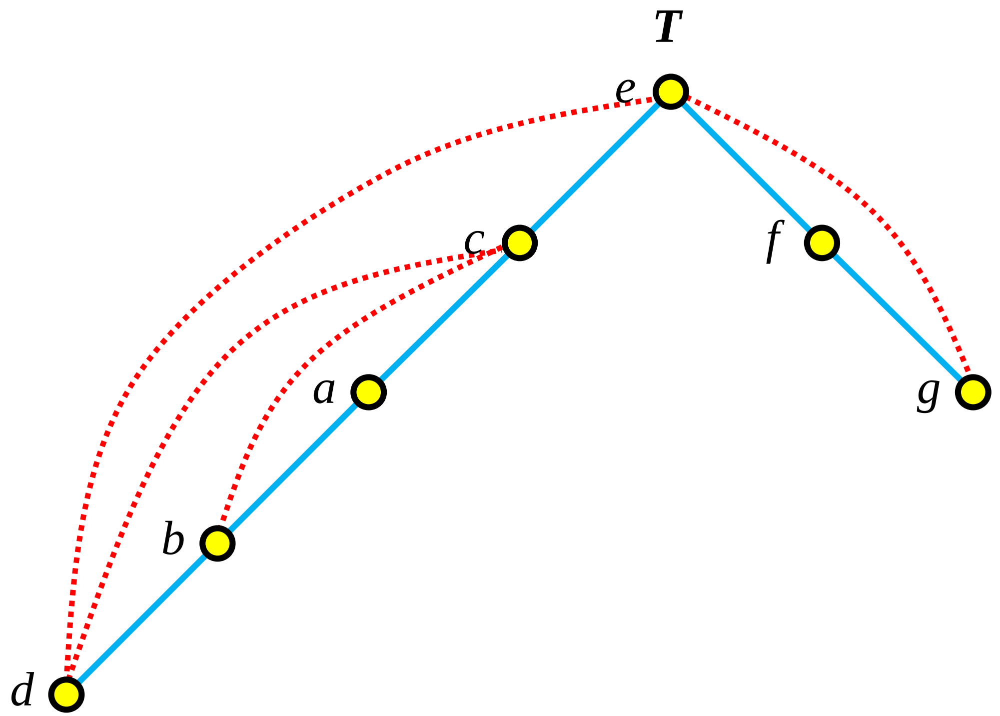
$P(f) \longrightarrow N(f) = \{e, g\}$

$P(g) \longrightarrow N(g) = \{e, f\}$

v	a	b	c	d	e	f	g
$PE(v)$	3	4	2	5	1	10	11
$PS(v)$	8	7	9	6	0	0	0

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

$P(a) \longrightarrow N(a) = \{b, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

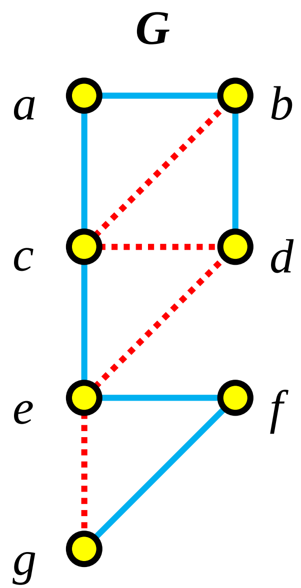
$P(d) \longrightarrow N(d) = \{b, c, e\}$

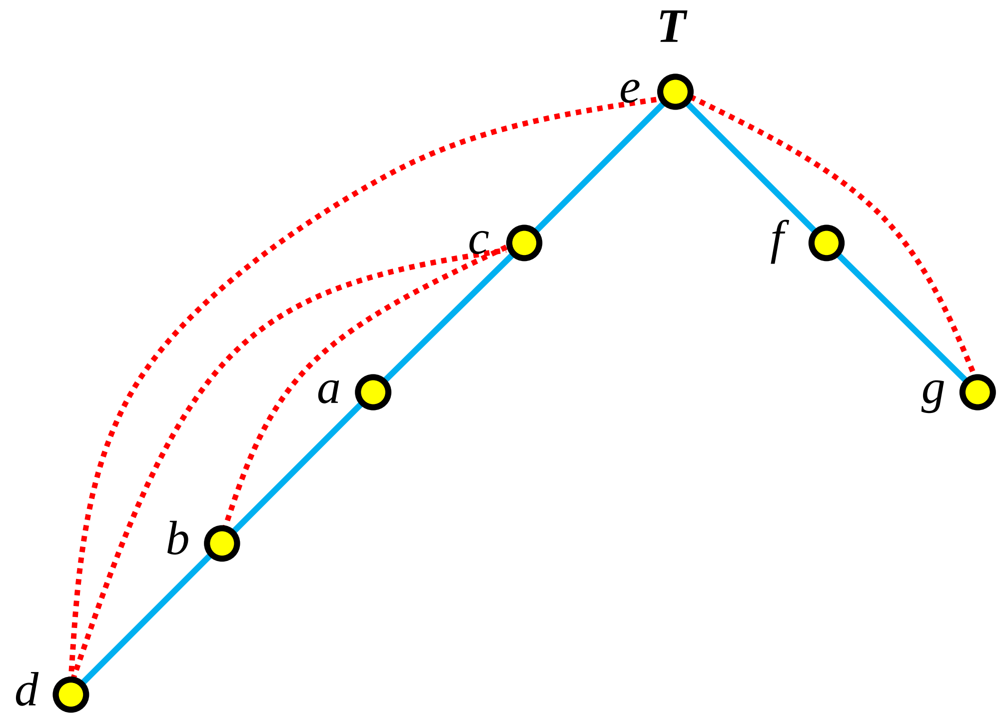
$P(f) \longrightarrow N(f) = \{e, g\}$

$P(g) \longrightarrow N(g) = \{e, f\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	11
$PS(v)$	8	7	9	6	0	0	12

Busca em profundidade





$P(e) \longrightarrow N(e) = \{\mathbf{c}, \mathbf{d}, \mathbf{f}, g\}$

$P(c) \longrightarrow N(c) = \{\mathbf{a}, \mathbf{b}, \mathbf{d}, \mathbf{e}\}$

$P(a) \longrightarrow N(a) = \{\mathbf{b}, \mathbf{c}\}$

$P(b) \longrightarrow N(b) = \{\mathbf{a}, \mathbf{c}, \mathbf{d}\}$

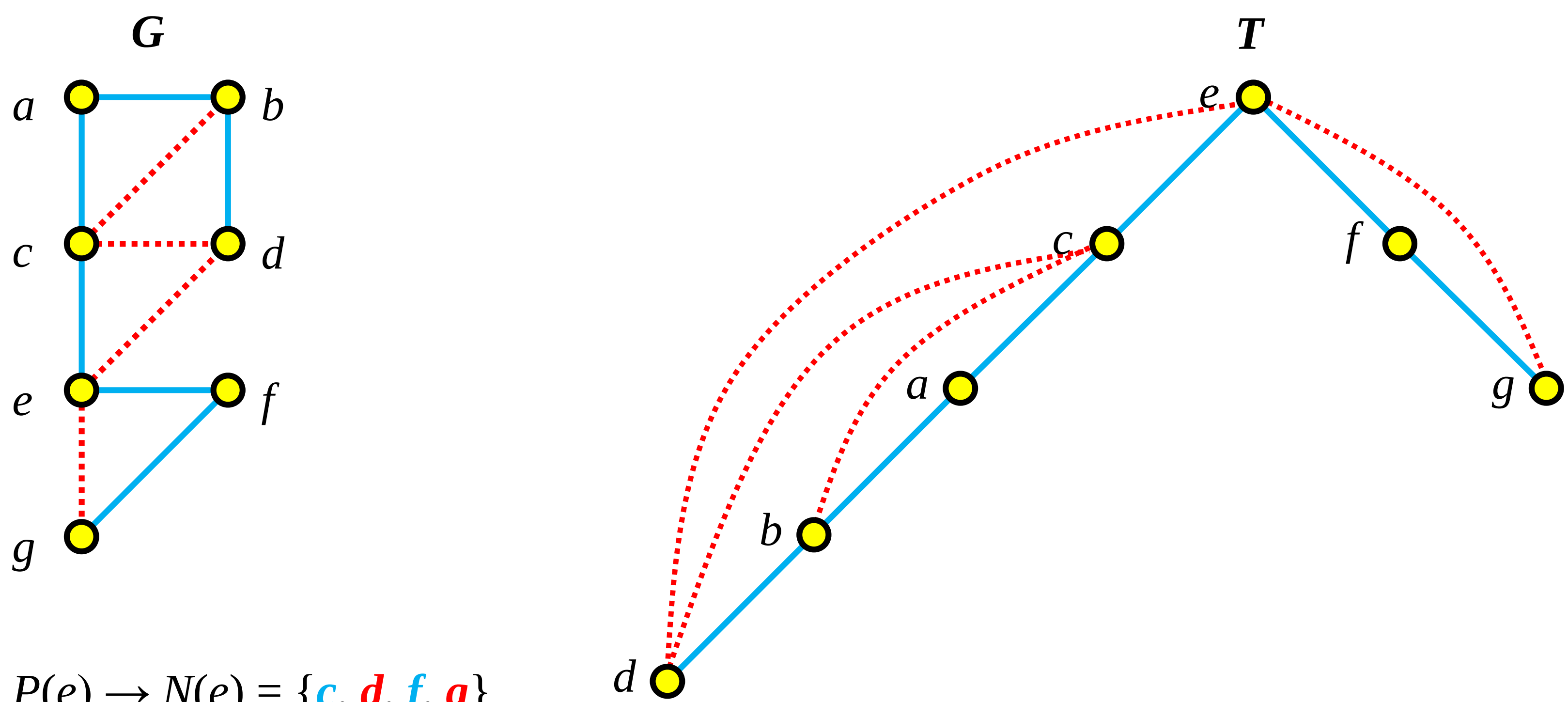
$P(d) \longrightarrow N(d) = \{\mathbf{b}, \mathbf{c}, \mathbf{e}\}$

$P(f) \longrightarrow N(f) = \{\mathbf{e}, \mathbf{g}\}$

$P(g) \longrightarrow N(g) = \{\mathbf{e}, \mathbf{f}\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	11
$PS(v)$	8	7	9	6	0	13	12

Busca em profundidade



$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

$P(a) \longrightarrow N(a) = \{b, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

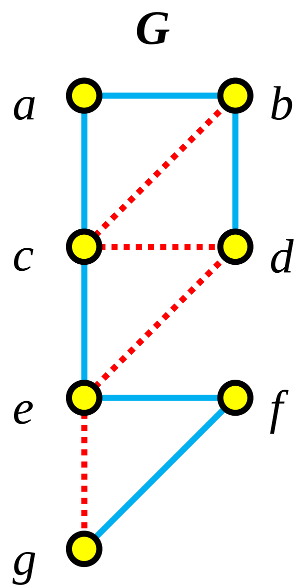
$P(d) \longrightarrow N(d) = \{b, c, e\}$

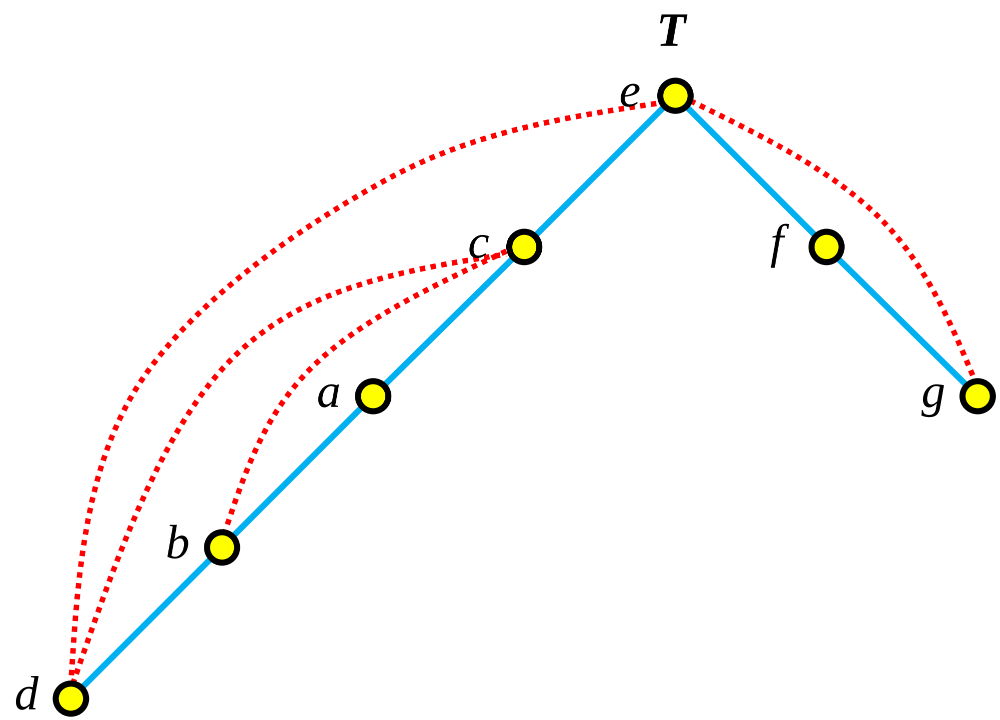
$P(f) \longrightarrow N(f) = \{e, g\}$

$P(g) \longrightarrow N(g) = \{e, f\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	11
$PS(v)$	8	7	9	6	0	13	12

Busca em profundidade





$P(e) \longrightarrow N(e) = \{c, d, f, g\}$

$P(c) \longrightarrow N(c) = \{a, b, d, e\}$

$P(a) \longrightarrow N(a) = \{b, c\}$

$P(b) \longrightarrow N(b) = \{a, c, d\}$

$P(d) \longrightarrow N(d) = \{b, c, e\}$

$P(f) \longrightarrow N(f) = \{e, g\}$

$P(g) \longrightarrow N(g) = \{e, f\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	11
$PS(v)$	8	7	9	6	14	13	12

Busca em profundidade

“Esgotar o filho antes de esgotar o pai”

“A próxima aresta a ser visitada parte sempre do vértice mais recente na busca”

Busca em profundidade

Complexidade da busca em profundidade:

$$O(n + m)$$

(onde $n = |V(G)|$ e $m = |E(G)|$)

Busca em profundidade

Se o grafo for desconexo, a busca alcançará apenas os vértices que estão conectados ao vértice-raiz da busca por caminhos!!

Alternativa:
modificar a chamada externa para alcançar todo o grafo

enquanto existe v em $V(G)$ tal que $PE(v) = 0$ *faça*
executar $P(v)$

Busca em profundidade

- Uma **árvore** é um grafo conexo e acíclico (sem ciclos).
- Se o grafo de entrada *G* é conexo, a árvore de profundidade *T* é uma **árvore geradora de *G*** (isto é, uma árvore que alcança todos os vértices de *G*); neste caso, todos os vértices de *G* são alcançados pela busca e ficam com uma *PE* diferente de zero no final da mesma.
- Somente as arestas **azuis** (ligando pai e filho) pertencem à arvore de profundidade *T*. As arestas **vermelhas** (arestas de retorno) não pertencem a *T*.

Busca em profundidade

Propriedades das arestas de retorno

- Toda aresta de retorno fecha um **ciclo**.
- Toda aresta de retorno liga um vértice v a um de seus **ancestrais** na árvore de profundidade T.

Busca em profundidade

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***
$PE(v)$	3	4	2	5	1	10	11
$PS(v)$	8	7	9	6	14	13	12

$t \rightarrow$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a														
b														
c														
d														
e														
f														
g														

Intervalos de vida dos vértices: *v* é descendente de *w* na árvore de profundidade *T* se e somente se o intervalo de vida de *v* está contido no intervalo de vida de *w*.
Isto é: $PE(v) > PE(w)$ e $PS(v) < PS(w)$ (*v "entra depois" e "sai antes" de w*).

Busca em profundidade

Aplicação 1: Dado um grafo G, verificar se G é conexo.

Solução: Rodar o laço abaixo.

$c \leftarrow 0$

<u>*enquanto*</u> *existe* v *em* $V(G)$ *tal que* $PE(v) = 0$ <u>*faça*</u>

$c \leftarrow c + 1$

executar $P(v)$

<u>*fim-enquanto*</u>

No final da execução, a variável c é igual ao número de componentes conexas de G (subgrafos conexos maximais que compõem G).

Busca em profundidade

Aplicação 2: Dado um grafo G e dois vértices v, w de G, verificar se existe um caminho de v a w em G.

Solução: Basta executar uma única vez o procedimento $P(v)$. No final da execução, se $PE(w) = 0$ então w não foi alcançado pela busca com raiz v, isto é, w está em uma componente conexa diferente da de v, e portanto não existe caminho de v a w em G. Caso contrário, se $PE(w) \neq 0$, então w foi alcançado, e existe um caminho de v a w em G; neste caso, w será descendente de v na árvore de profundidade T, e para determinar o caminho basta rodar o algoritmo a seguir.

Busca em profundidade

Aplicação 2: Dado um grafo G e dois vértices v, w de G, verificar se existe um caminho de v a w em G.

Solução (continuação):

$x \leftarrow w$

$C \leftarrow x$

enquanto $x \neq v$ *faça*

$x \leftarrow pai(x)$

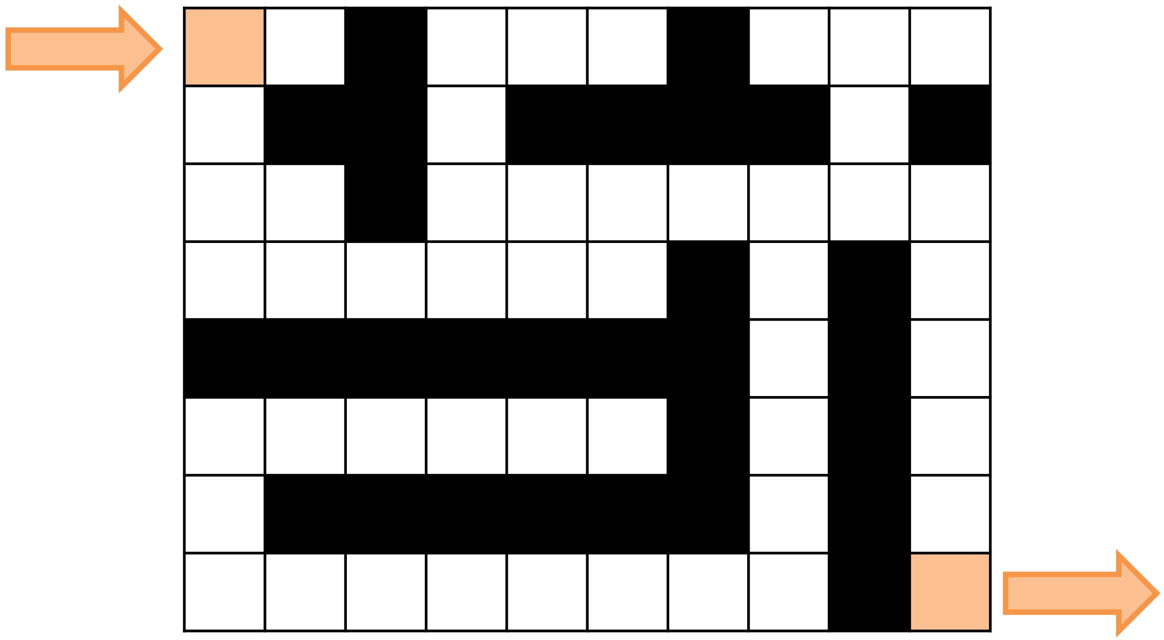
colocar x *à esquerda em* C

fim-enquanto

imprimir C ***Obs: C não é necessariamente o melhor***

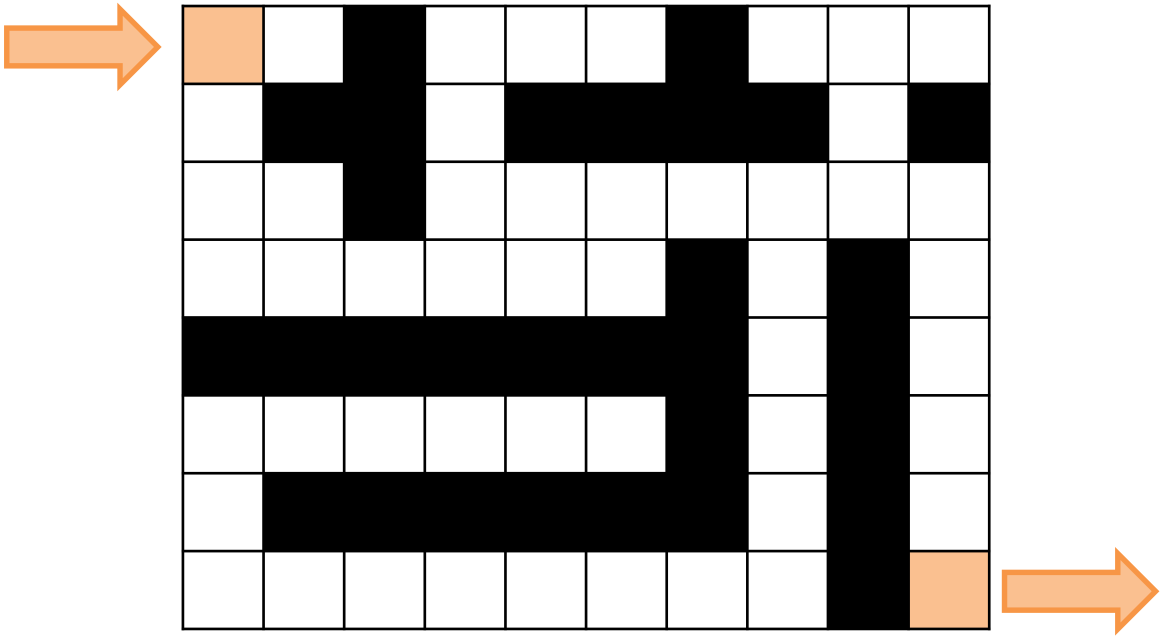
Busca em profundidade

Aplicação 3: Dado um labirinto, determinar um caminho da entrada até a saída (se existir).



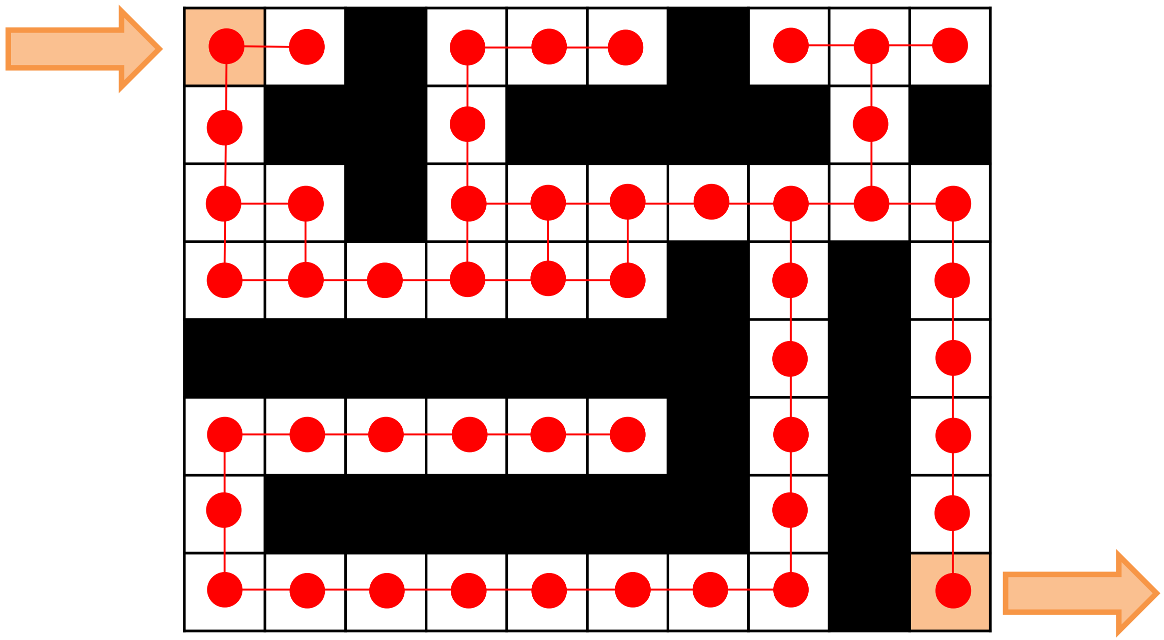
Busca em profundidade

Aplicação 3 - Solução: Montar o grafo correspondente ao labirinto e usar a solução da aplicação anterior.



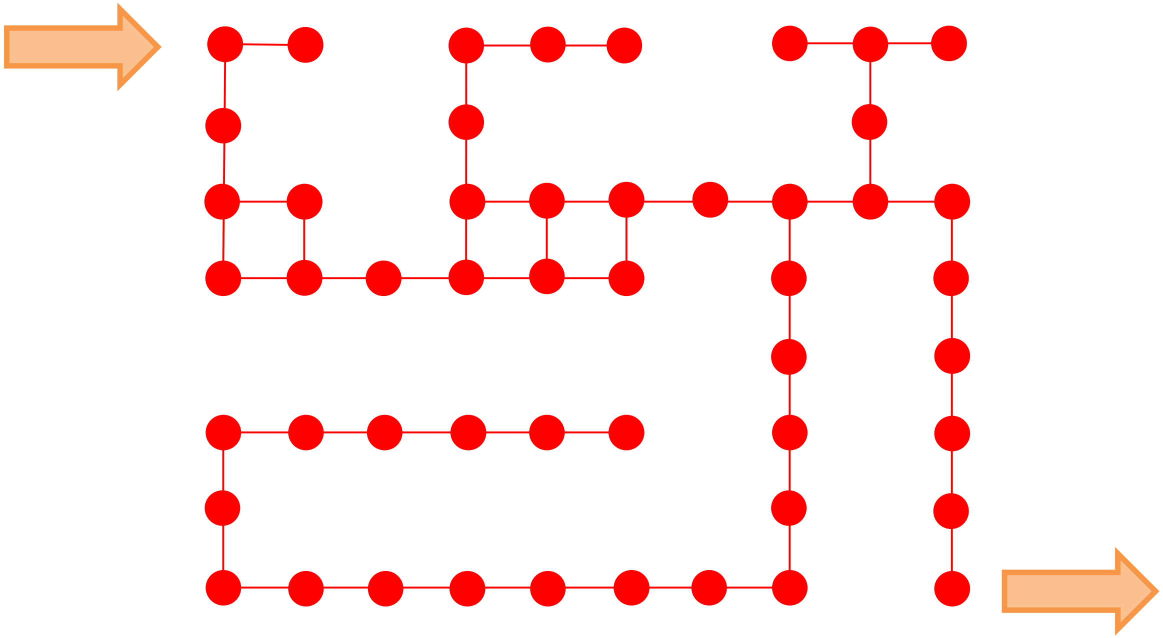
Busca em profundidade

Aplicação 3 - Solução: Montar o grafo correspondente ao labirinto e usar a solução da aplicação anterior.



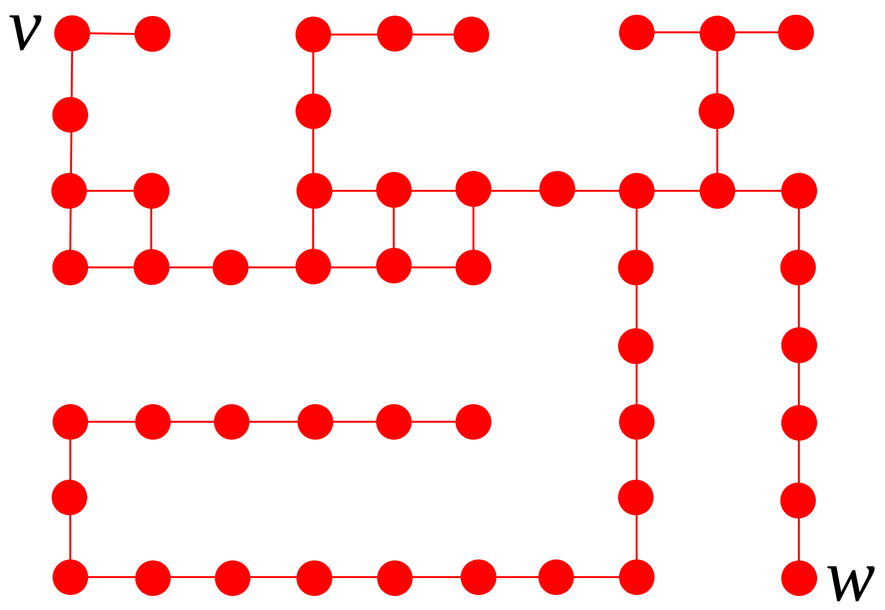
Busca em profundidade

Aplicação 3 - Solução: Montar o grafo correspondente ao labirinto e usar a solução da aplicação anterior.



Busca em profundidade

Aplicação 3 - Solução: Montar o grafo correspondente ao labirinto e usar a solução da aplicação anterior.



Busca em profundidade

Aplicação 4: Dado um grafo G, encontrar um ciclo de G ou concluir que G é acíclico.

Solução: Executar uma busca em profundidade em G. Teremos dois casos:

- A busca não gerou nenhuma aresta de retorno. Então, G é acíclico, e as arestas **azuis** formam uma **floresta geradora** (um conjunto de árvores, uma para cada componente conexa).
- A busca gerou uma aresta de retorno ***vw***. Então, v é descendente de w na árvore, e um ciclo C é formado. Para determinar C, basta rodar o algoritmo a seguir:

Busca em profundidade

Aplicação 4: Dado um grafo G, encontrar um ciclo de G ou concluir que G é acíclico.

Solução (continuação):

- $x \leftarrow v$
  $C \leftarrow x$
  <u>*enquanto*</u> $x \neq w$ <u>*faça*</u>
  $\quad x \leftarrow pai(x)$
  $\quad$ *colocar* x *à direita em* C
  <u>*fim-enquanto*</u>
  colocar v *à direita em* C (*para fechar o ciclo*)
  <u>*imprimir*</u> C

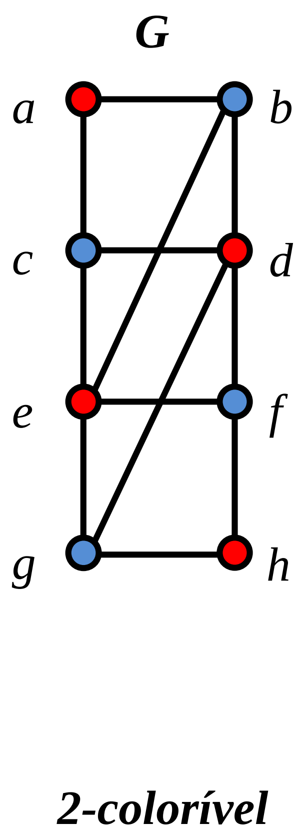
Busca em profundidade

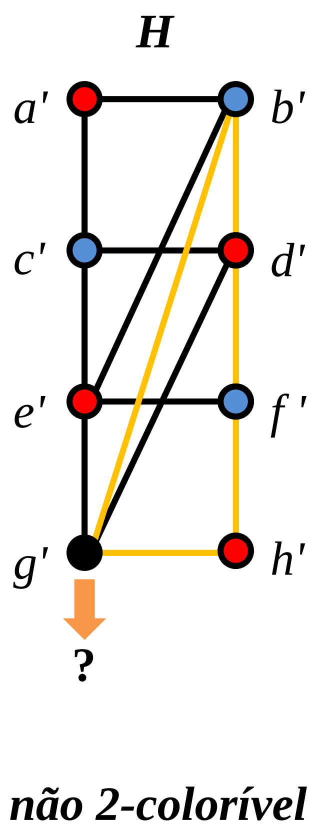
Aplicação 5: Dado um grafo *G*, decidir se *G* é 2-colorível.

- Um grafo *G* é 2-colorível (ou bipartido) se podemos colorir os vértices de *G* com duas cores de modo que vértices vizinhos não recebam a mesma cor.
- Teorema: *G* é 2-colorível se e somente se *G* não contém um ciclo ímpar (ciclo com número ímpar de arestas.

Busca em profundidade

Aplicação 5: Dado um grafo G, decidir se G é 2-colorível.





Busca em profundidade

Aplicação 5: Dado um grafo *G*, decidir se *G* é 2-colorível.

Solução: Atribuir cor "0" à raiz da busca e rodar:

```
procedimento P(v)
  t ← t + 1; PE(v) ← t
  para todo vértice w em N(v) faça
      se PE(w) = 0
         então | visitar vw;
               | pai(w) ← v; cor(w) ← 1 – cor(v);
               | executar P(w)
         senão  se PS(w) = 0 e  w ≠ pai(v)
                    então  se cor(w) ≠ cor(v)
                              então visitar vw
                              senão  pare: ciclo ímpar! (imprimir como na Aplic.4)
  fim-para
  t ← t + 1; PS(v) ← t
```

Busca em profundidade

Exercício: Elaborar um método (algoritmo) para resolver a seguinte questão: Dado um grafo conexo G e uma árvore geradora T de G, decidir se T é uma árvore de profundidade para G. Isto é, decidir se existe uma busca em profundidade em G que produza T como árvore de profundidade.

Busca em profundidade

Aplicação 6: Dado um grafo G, determinar as articulações e os blocos de G.

- Uma articulação de um grafo conexo G é um vértice v cuja remoção desconecta G.
- Se v é articulação então $\omega(G-v) > \omega(G)$, isto é, o número de componentes conexas de $G-v$ é maior do que o número de componentes conexas de G.

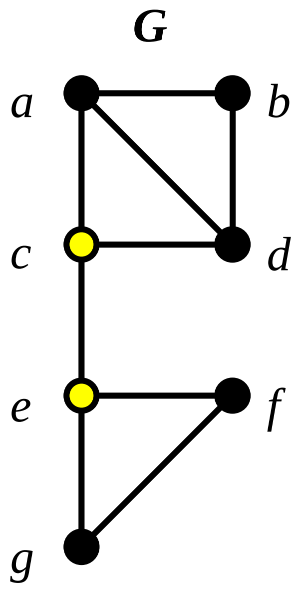
Busca em profundidade

Aplicação 6: Dado um grafo G, determinar as articulações e os blocos de G.

- Uma bloco de um grafo G é um subgrafo maximal H de G com a seguinte propriedade: H (considerado isoladamente) é conexo e não contém articulações.
- Em alguns casos, um bloco pode ser formado por uma única aresta. Esta aresta será chamada ponte.
- Em todo bloco ***que não seja uma ponte***, existem dois caminhos internamente disjuntos entre qualquer par de vértices. Neste caso, o bloco é um subgrafo maximal biconexo.

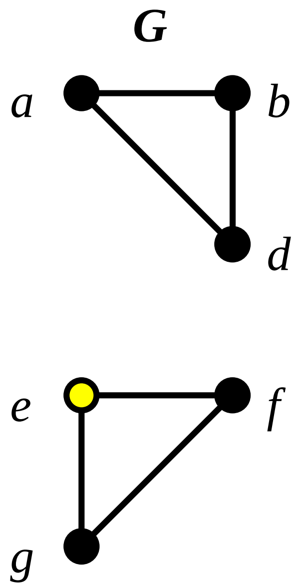
Busca em profundidade

Aplicação 6: Dado um grafo G, determinar as articulações e os blocos de G.



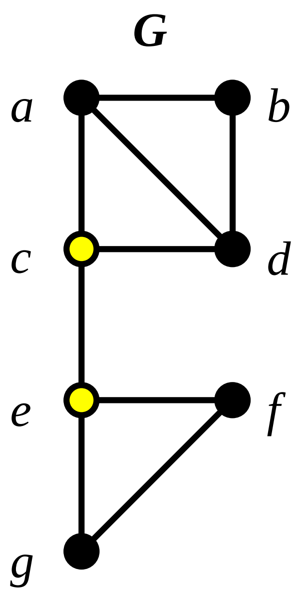
Busca em profundidade

Aplicação 6: Dado um grafo *G*, determinar as articulações e os blocos de *G*.



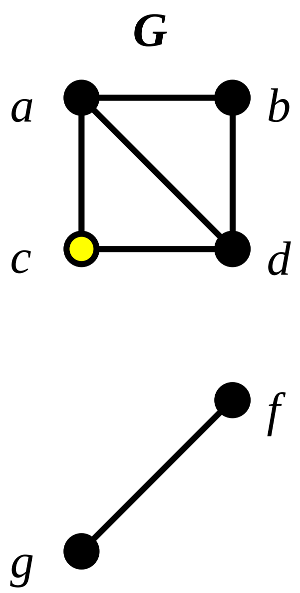
Busca em profundidade

Aplicação 6: Dado um grafo *G*, determinar as articulações e os blocos de *G*.



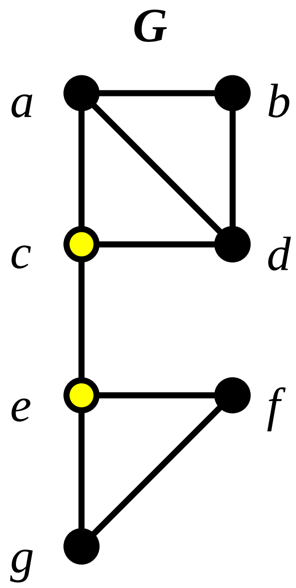
Busca em profundidade

Aplicação 6: Dado um grafo *G*, determinar as articulações e os blocos de *G*.



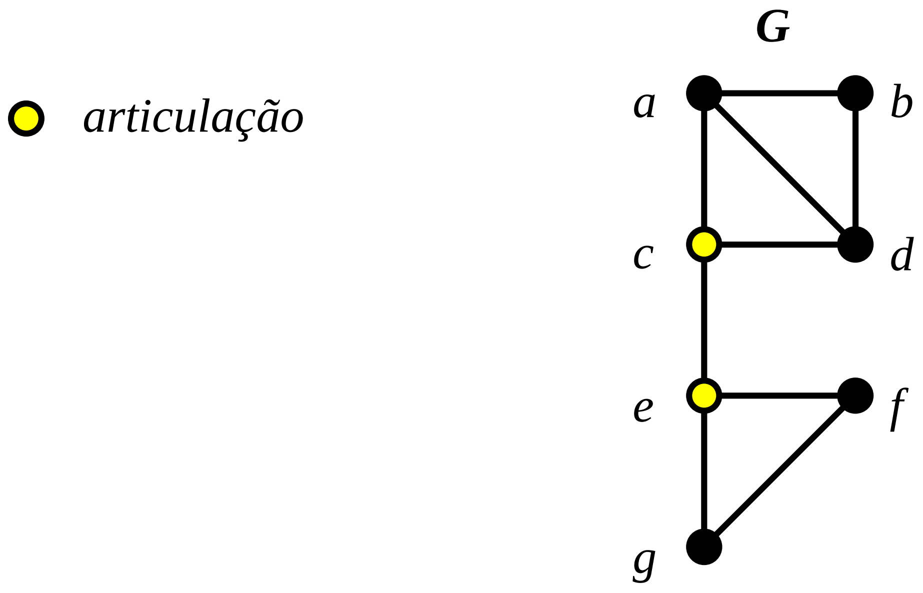
Busca em profundidade

Aplicação 6: Dado um grafo *G*, determinar as articulações e os blocos de *G*.



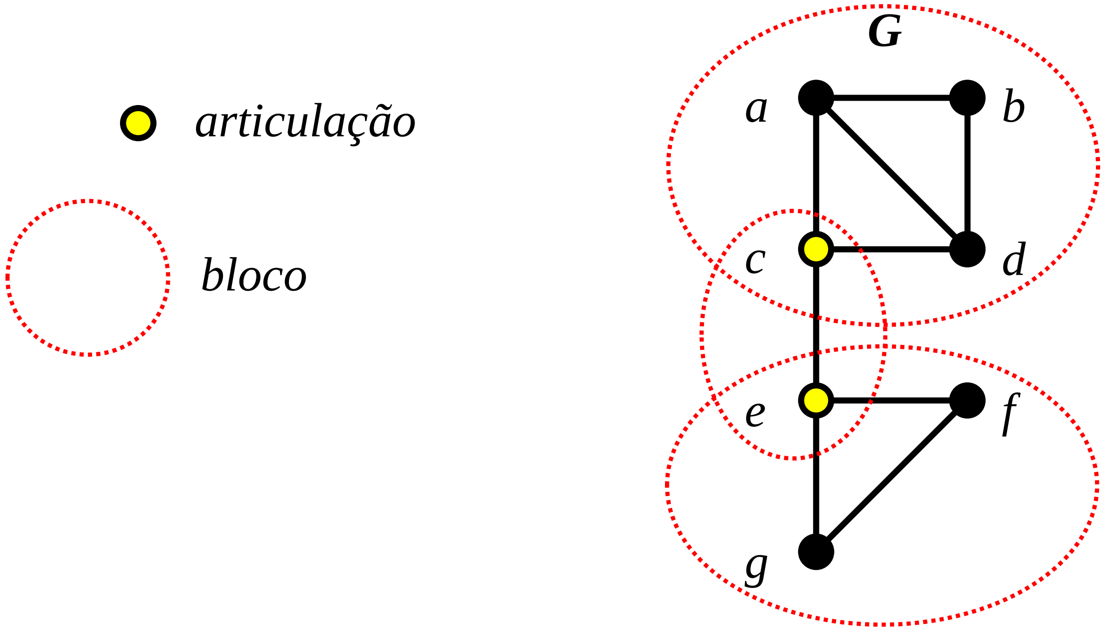
Busca em profundidade

Aplicação 6: Dado um grafo *G*, determinar as articulações e os blocos de *G*.



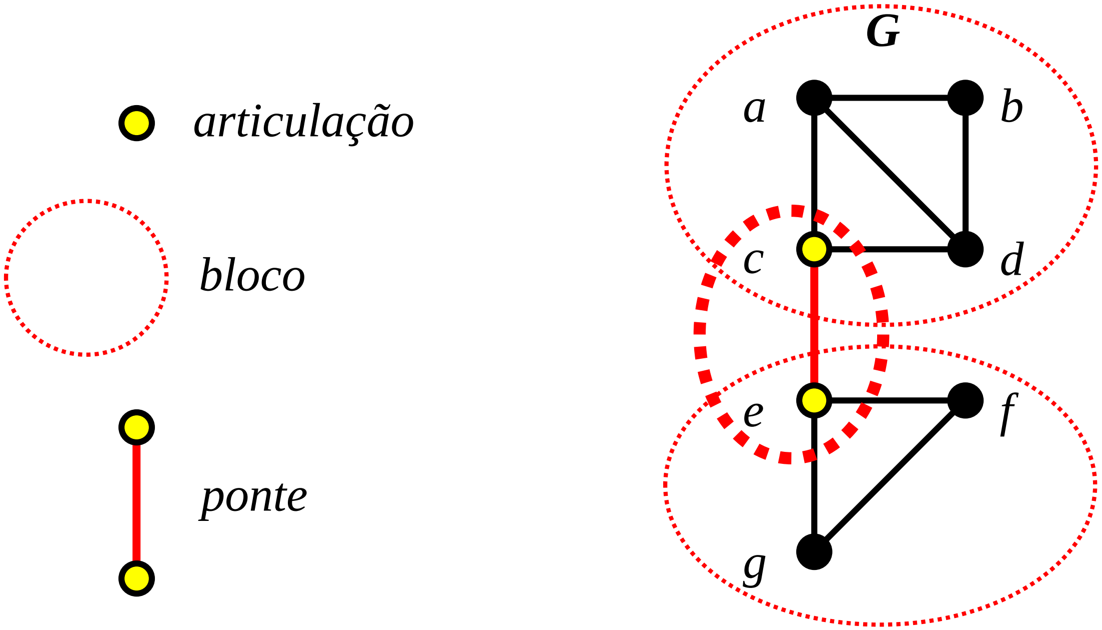
Busca em profundidade

Aplicação 6: Dado um grafo *G*, determinar as articulações e os blocos de *G*.



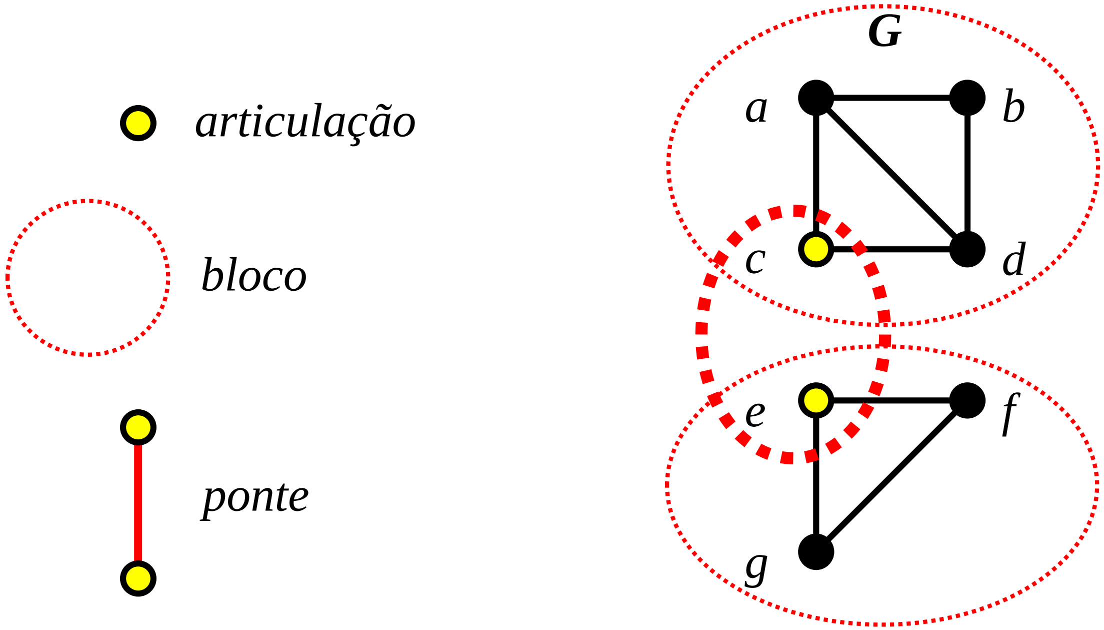
Busca em profundidade

Aplicação 6: Dado um grafo *G*, determinar as articulações e os blocos de *G*.



Busca em profundidade

Aplicação 6: Dado um grafo *G*, determinar as articulações e os blocos de *G*.



A remoção de uma ponte desconecta o grafo!

Busca em profundidade

Aplicação 6: Dado um grafo *G*, determinar as articulações e os blocos de *G*.

- Os blocos de um grafo *G* determinam naturalmente uma *partição* do conjunto de arestas de *G*, isto é, cada aresta pertence a um e apenas um bloco de *G*.
- O mesmo não ocorre em relação aos vértices:
  - se *v* pertence a mais de um bloco então *v* é uma articulação
  - se *v* pertence a um único bloco então *v* ***não*** é uma articulação

Busca em profundidade

Aplicação 6: Dado um grafo *G*, determinar as articulações e os blocos de *G*.

- Considerando o grafo como uma rede …
  - articulações são nós críticos
  - pontes são conexões críticas

Busca em profundidade

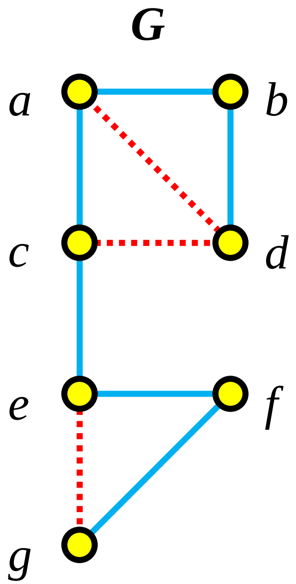
Aplicação 6: Dado um grafo G, determinar as articulações e os blocos de G.

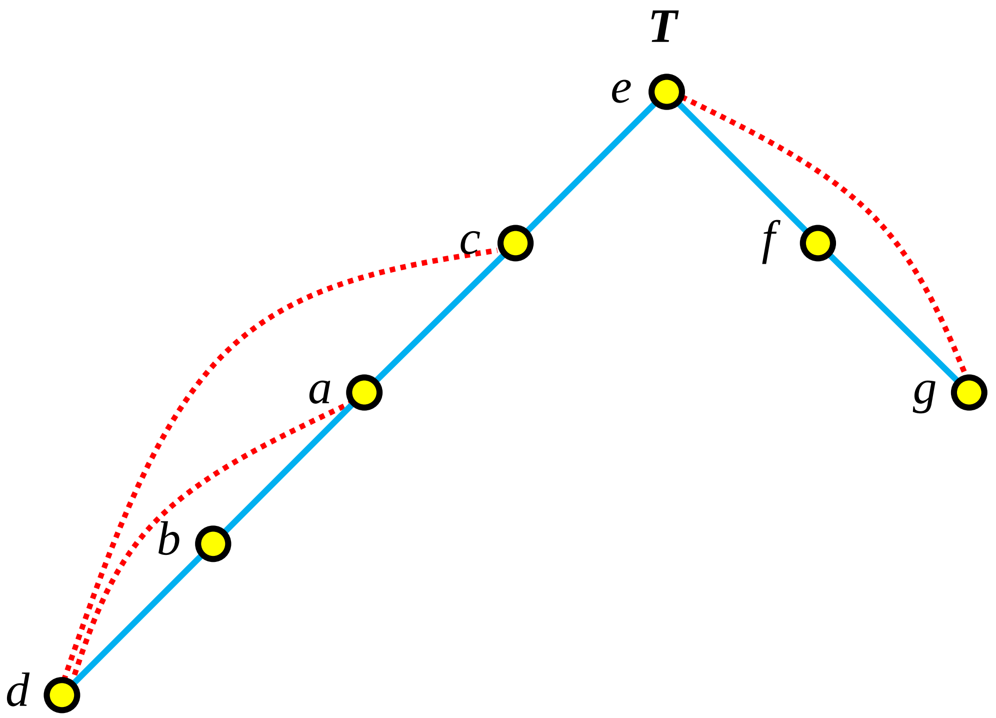
Definição:

Seja T uma árvore de profundidade para o grafo G.

$back(v) = PE$ do vértice w mais próximo da raiz de T que pode ser alcançado a partir de v usando 0 ou mais arestas de T para baixo e, a seguir, no máximo uma aresta de retorno.

Busca em profundidade





v	*a*	*b*	*c*	*d*	*e*	*f*	*g*
PE(*v*)	3	4	2	5	1	10	11
PS(*v*)	8	7	9	6	14	13	12
back(*v*)	2	2	2	2	1	1	1

Busca em profundidade

Aplicação 6: Dado um grafo G, determinar as articulações e os blocos de G.

Como calcular *back(v)*

$$
\begin{aligned}
back(v) = \min\Big(\; & \{ PE(v) \} \cup \\
& \{ back(w) \mid w \text{ é filho de } v \text{ em } T \} \cup \\
& \{ PE(w) \mid vw \text{ é aresta de retorno } \} \; \Big)
\end{aligned}
$$

Busca em profundidade

Aplicação 6: Dado um grafo *G*, determinar as articulações e os blocos de *G*.

<u>procedimento</u> $P(v)$ **-- com cálculo de back(v)**
$t \leftarrow t + 1$; $PE(v) \leftarrow t$; $\boldsymbol{back(v) \leftarrow PE(v)}$; **=> inicialização**
<u>para</u> *todo vértice w em* $N(v)$ <u>faça</u>
    <u>se</u> $PE(w) = 0$
        <u>então</u> | *visitar vw*; $pai(w) \leftarrow v$;
              | *executar* $P(w)$
              | $\boldsymbol{back(v) \leftarrow \min(\, back(v), back(w)\,)}$
        <u>senão</u> <u>se</u> $PS(w) = 0$ <u>e</u> $w \neq pai(v)$
                <u>então</u> | *visitar vw*
                      | $\boldsymbol{back(v) \leftarrow \min(\, back(v), PE(w)\,)}$
<u>fim-para</u>
$t \leftarrow t + 1$; $PS(v) \leftarrow t$

Busca em profundidade

Aplicação 6: Dado um grafo G, determinar as articulações e os blocos de G.

Como usar os valores *back*(v) para determinar as articulações

Teorema: Seja T uma árvore de profundidade para um grafo G. Suponha que os valores $PE(v)$ e $back(v)$ estejam calculados. Seja v um vértice qualquer de G.

- Se ***v*** é a raiz de T então v é articulação sss v possui dois ou mais filhos em T.
- Se ***v não*** é a raiz de T então v é articulação sss existe pelo menos um filho w de v com $back(w) \geq PE(v)$.

Busca em profundidade

Aplicação 6: Dado um grafo G, determinar as articulações e os blocos de G.

<u>procedimento</u> $P(v)$ **-- com cálculo de back(v) e dos blocos (idéia: usar uma pilha)**

$t \leftarrow t + 1$; $PE(v) \leftarrow t$; **back(v) ← PE(v);** **=> inicialização**

<u>para</u> *todo vértice* w *em* $N(v)$ <u>faça</u>

<u>se</u> $PE(w) = 0$

<u>então</u> | *visitar* vw; ***empilhar vw;*** $pai(w) \leftarrow v$

| *executar* $P(w)$

| ***<u>se</u> back(w) ≥ PE(v) <u>então</u> desempilhar e imprimir tudo até vw***

| **back(v) ← min(back(v), back(w))**

<u>senão</u> <u>se</u> $PS(w) = 0$ <u>e</u> $w \neq pai(v)$

<u>então</u> | *visitar* vw; ***empilhar vw***

| **back(v) ← min(back(v), PE(w))**

<u>fim-para</u>

$t \leftarrow t + 1$; $PS(v) \leftarrow t$

Busca em profundidade

Aplicação 6: Dado um grafo *G*, determinar as articulações e os blocos de *G*.

Algumas questões:

- Se a busca em profundidade se iniciar por um vértice que ***não é*** articulação, então a raiz da árvore de profundidade terá apenas um filho! (veja o Teorema)
- **Exercício:** Simular a execução para observar o comportamento da pilha de arestas.
- **Exercício:** Fazer pequenos acréscimos no código para determinar as *articulações* e as *pontes*.

Busca em profundidade

Aplicação 6: Dado um grafo G, determinar as articulações e os blocos de G.

Observação:

- Um filho w de v que satisfaça $back(w) \geq PE(v)$ é chamado de demarcador de v.
- Exercício: Verificar se cada bloco possui o seu próprio demarcador. (Se isto for verdade, então o número de demarcadores é igual ao número de blocos.)

Busca em profundidade p/ digrafos

Inicialização

$t \leftarrow 0$ **-- *t é o relógio ou tempo global***

para todo vértice v em $V(G)$ *faça*

$PE(v) \leftarrow 0$ **-- *PE(v) é a profundidade de entrada de v***

$PS(v) \leftarrow 0$ **-- *PS(v) é a profundidade de saída de v***

$pai(v) \leftarrow null$ **-- *ponteiros que definem a floresta de profundidade T***

Chamada Externa

enquanto existe v em $V(G)$ tal que $PE(v) = 0$ *faça*

executar $P(v)$ **-- *nova raiz da busca***

Busca em profundidade p/ digrafos

Procedimento recursivo de busca p/ digrafos

<u>procedimento</u> $P(v)$

$t \leftarrow t + 1$; $PE(v) \leftarrow t$

<u>para</u> todo vértice w em $N_{out}(v)$ *<u>faça</u>*

<u>se</u> $PE(w) = 0$ *(se w ainda não foi alcançado pela busca)*

<u>então</u>
- *marcar vw como aresta **"azul"** da floresta de profundidade T*
- $pai(w) \leftarrow v$
- *executar* $P(w)$

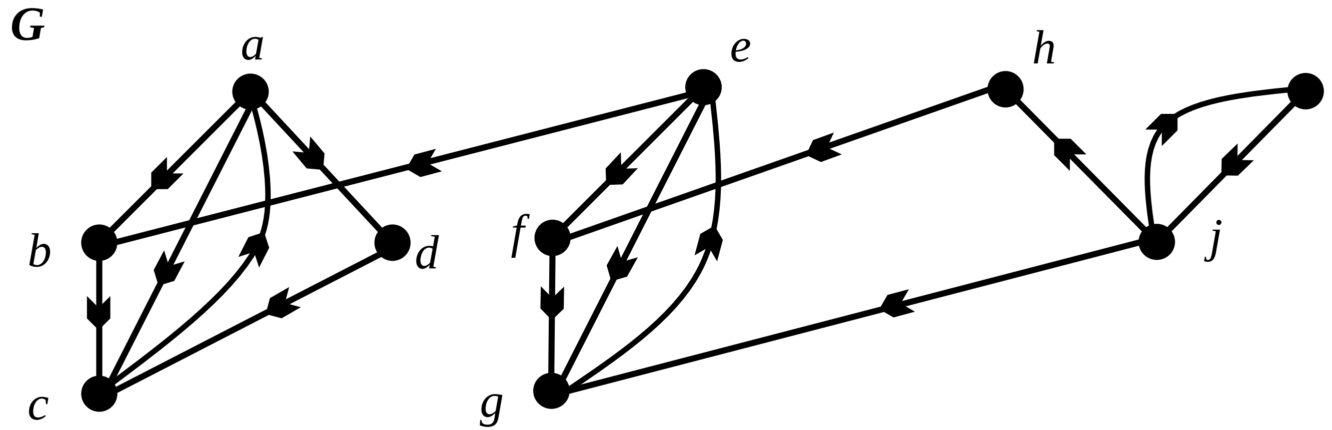
<u>senão</u>
- *<u>se</u>* $PS(w) = 0$ *(se w ainda não saiu da busca)*
  - *<u>então</u> marcar vw como aresta **"vermelha"** de retorno*
  - *<u>senão</u> <u>se</u>* $PE(v) < PE(w)$ ***(se v entrou antes de w na busca)***
    - *<u>então</u> marcar vw como **aresta "amarela" de avanço***
    - *<u>senão</u> marcar vw como **aresta "verde" cruzamento***

<u>fim-para</u>

$t \leftarrow t + 1$; $PS(v) \leftarrow t$

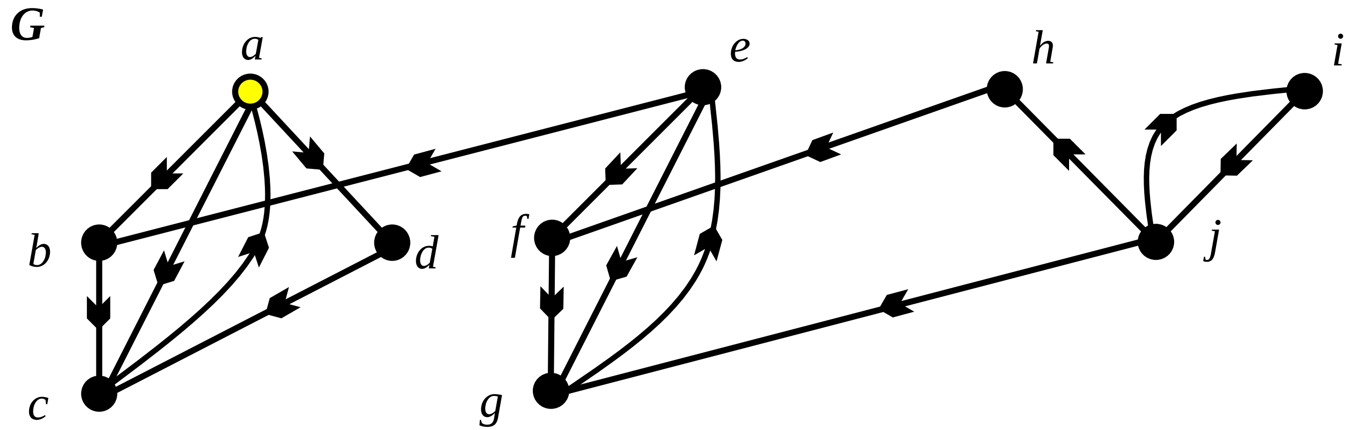
<u>fim-do-procedimento</u>

Busca em profundidade p/ digrafos



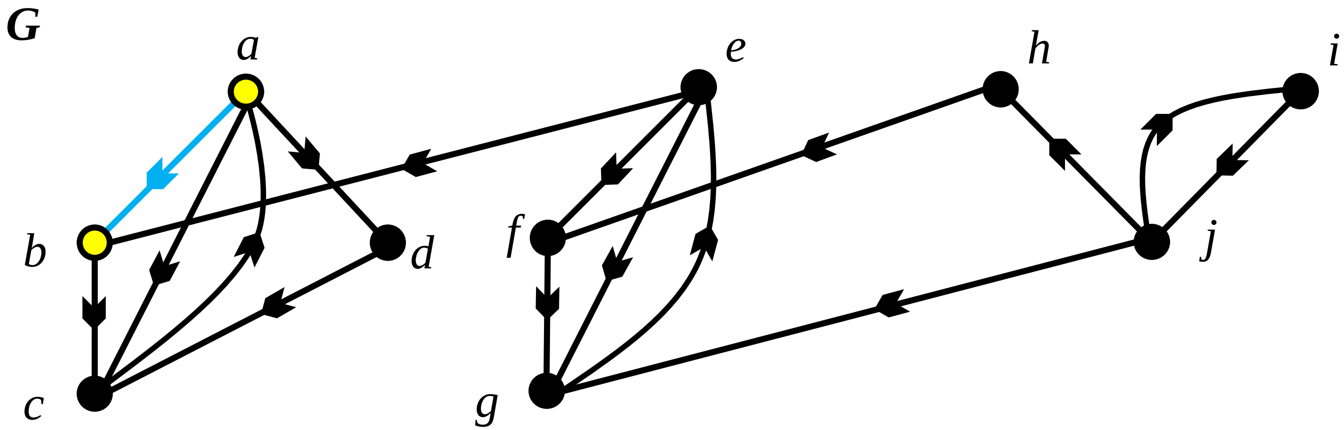
v	a	b	c	d	e	f	g	h	i	j
PE(*v*)	0	0	0	0	0	0	0	0	0	0
PS(*v*)	0	0	0	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos



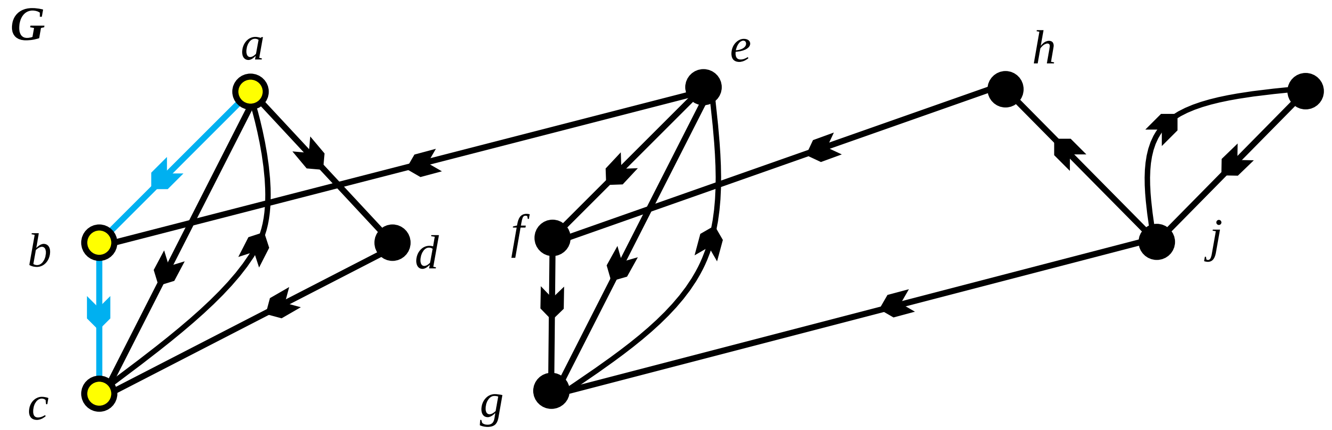
v	a	b	c	d	e	f	g	h	i	j
PE(*v*)	1	0	0	0	0	0	0	0	0	0
PS(*v*)	0	0	0	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos



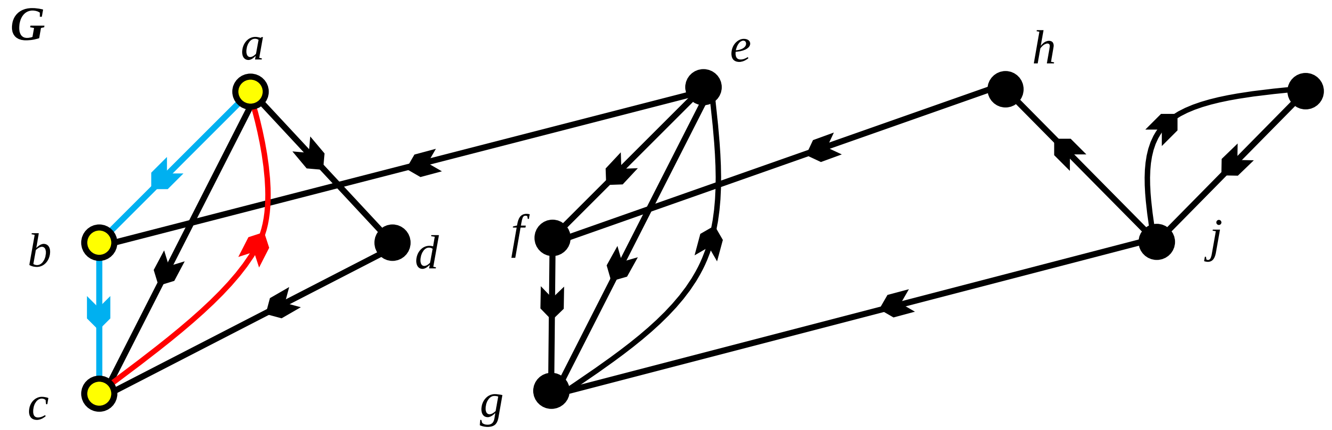
v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	0	0	0	0	0	0	0	0
$PS(v)$	0	0	0	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos



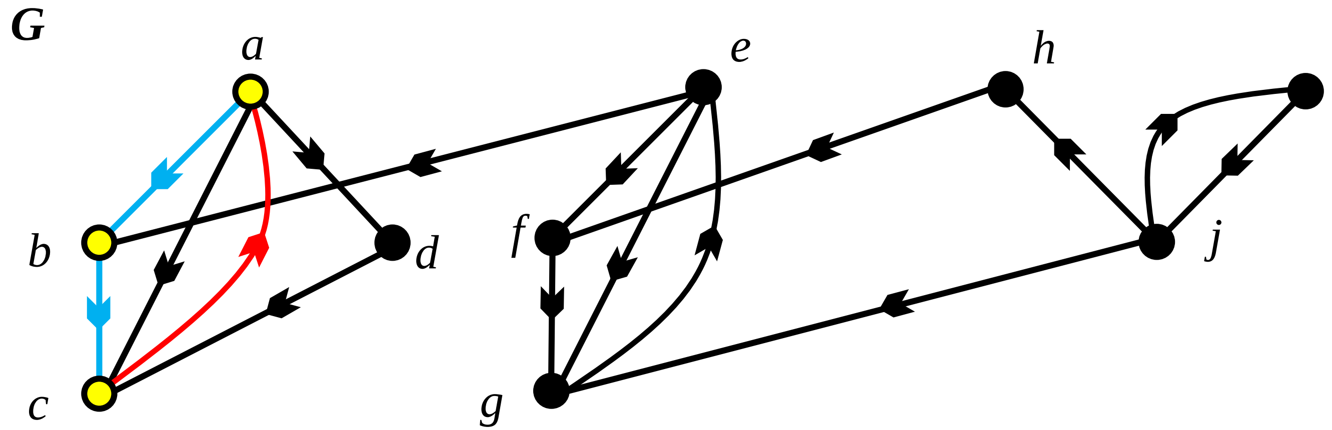
v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	0	0	0	0	0	0	0
$PS(v)$	0	0	0	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos



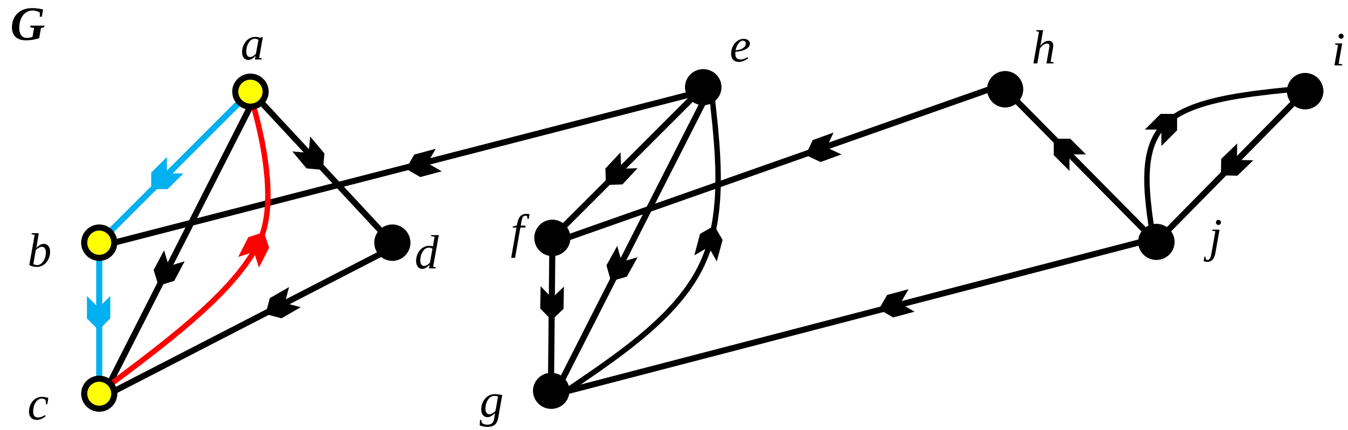
v	a	b	c	d	e	f	g	h	i	j
PE(*v*)	1	2	3	0	0	0	0	0	0	0
PS(*v*)	0	0	0	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos



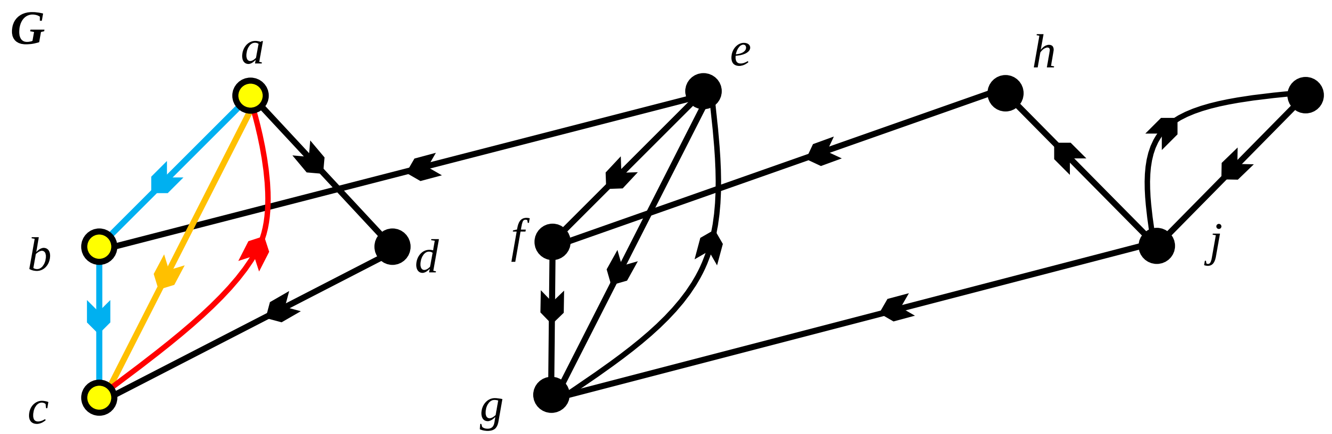
v	a	b	c	d	e	f	g	h	i	j
PE(*v*)	1	2	3	0	0	0	0	0	0	0
PS(*v*)	0	0	4	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos



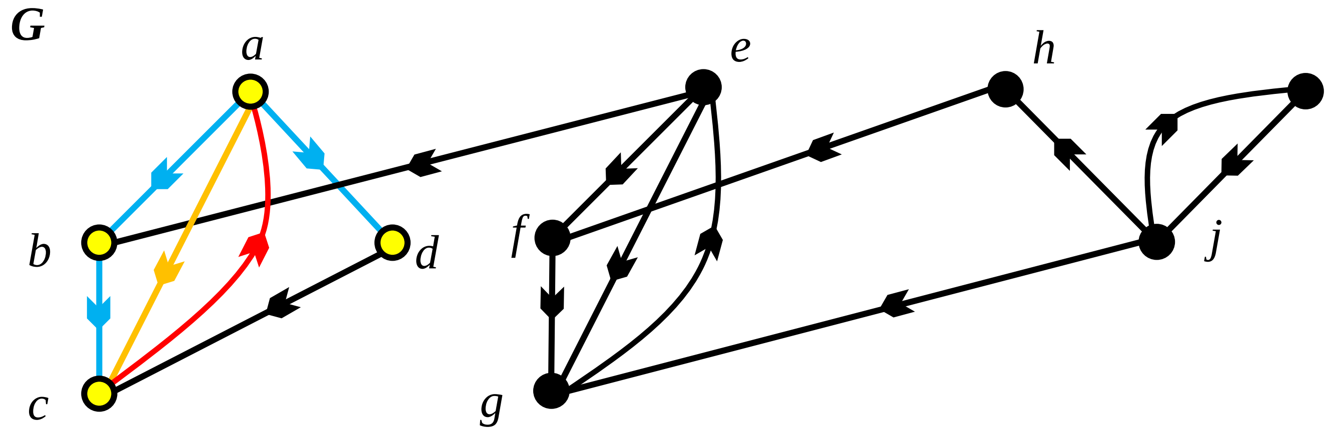
v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	0	0	0	0	0	0	0
$PS(v)$	0	5	4	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos



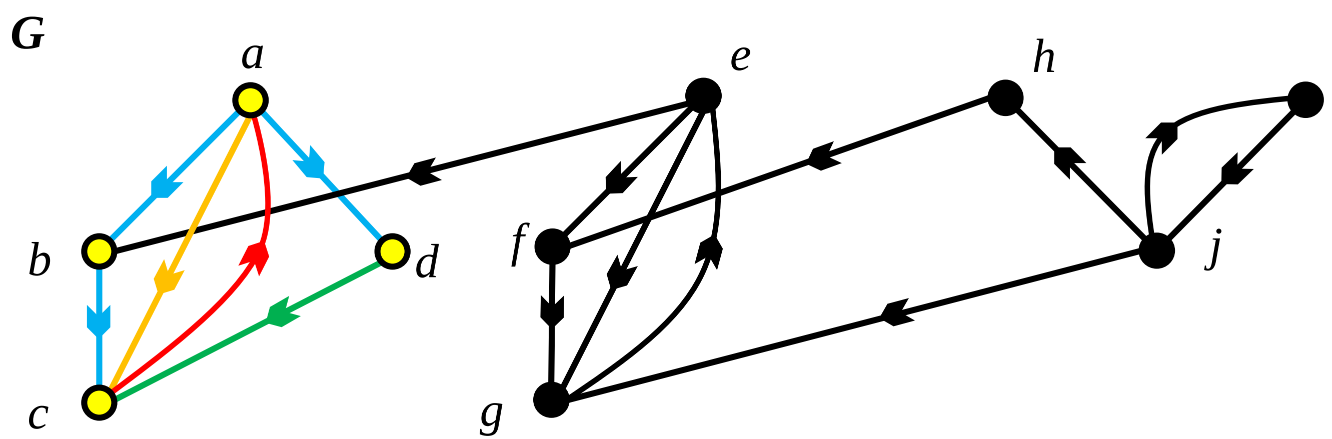
v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	0	0	0	0	0	0	0
$PS(v)$	0	5	4	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos



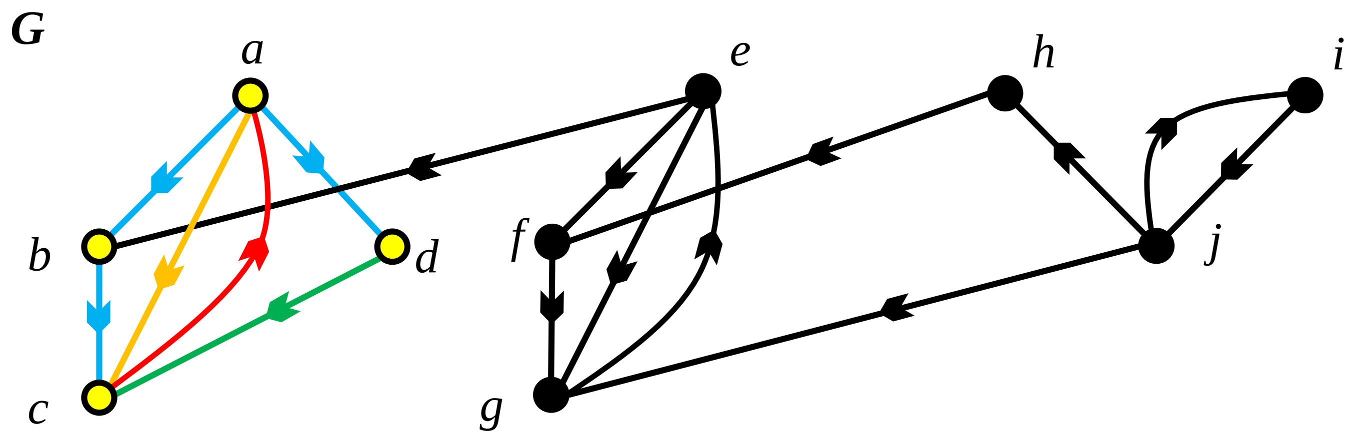
v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	0	0	0	0	0	0
PS(v)	0	5	4	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos



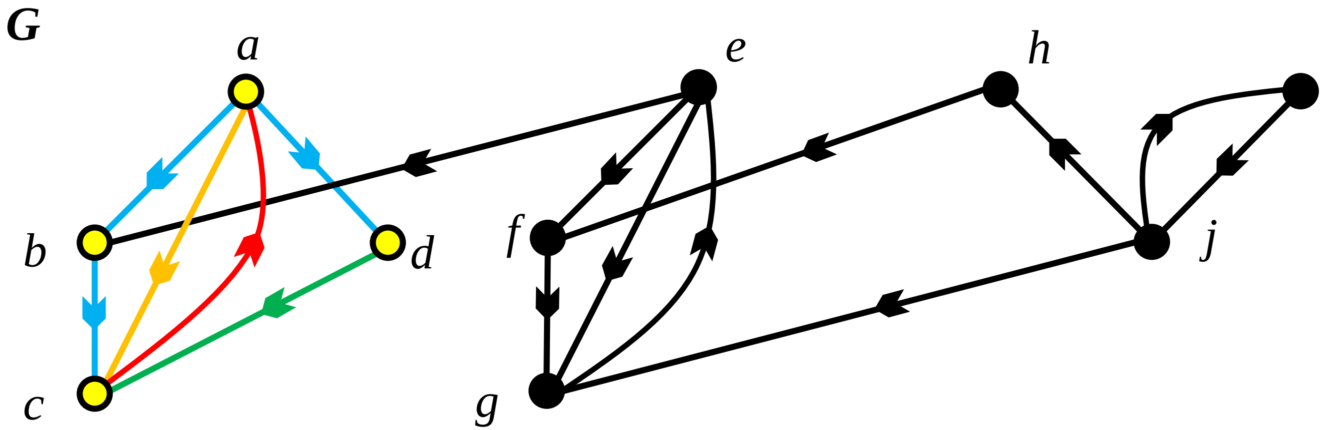
v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	0	0	0	0	0	0
PS(*v*)	0	5	4	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos



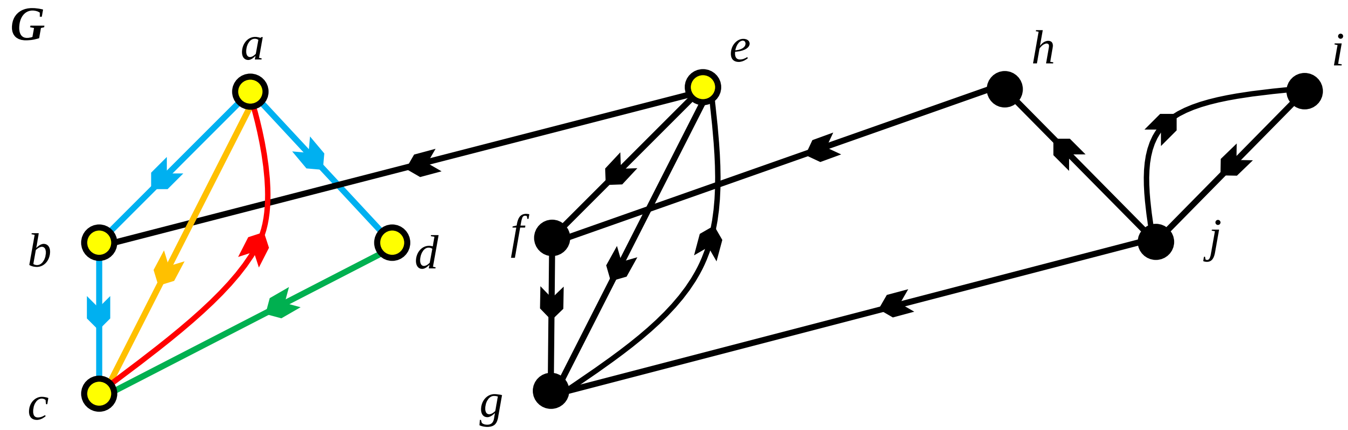
v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	0	0	0	0	0	0
PS(*v*)	0	5	4	7	0	0	0	0	0	0

Busca em profundidade p/ digrafos



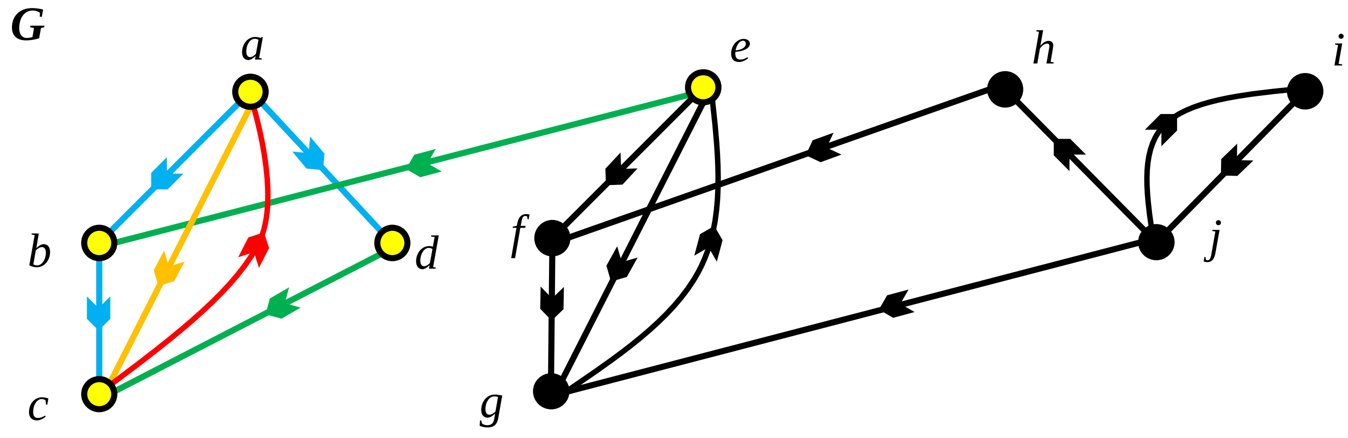
v	a	b	c	d	e	f	g	h	i	j
PE(*v*)	1	2	3	6	0	0	0	0	0	0
PS(*v*)	8	5	4	7	0	0	0	0	0	0

Busca em profundidade p/ digrafos



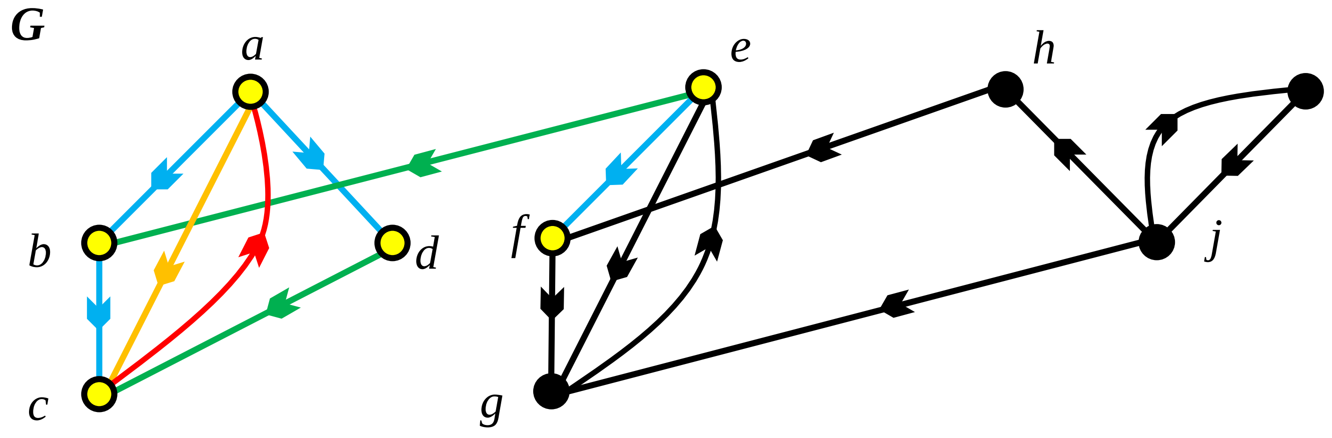
v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	0	0	0	0	0
$PS(v)$	8	5	4	7	0	0	0	0	0	0

Busca em profundidade p/ digrafos



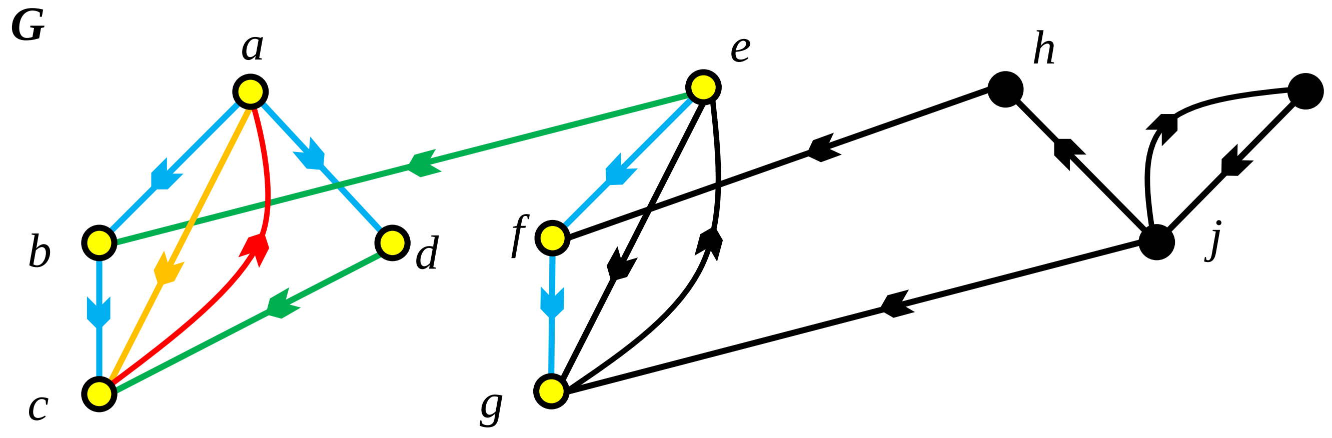
v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
$PE(v)$	1	2	3	6	9	0	0	0	0	0
$PS(v)$	8	5	4	7	0	0	0	0	0	0

Busca em profundidade p/ digrafos



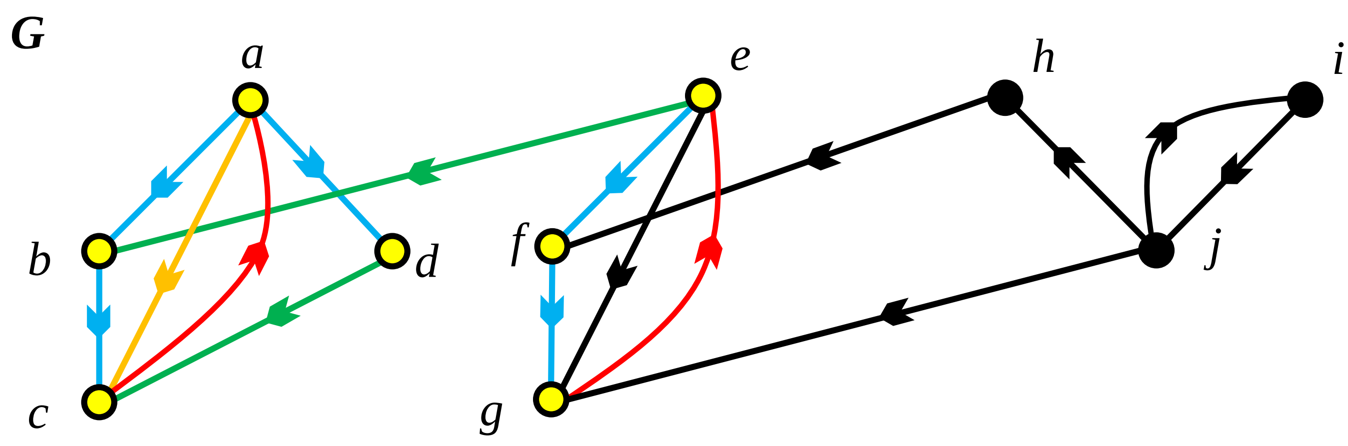
v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	0	0	0	0
$PS(v)$	8	5	4	7	0	0	0	0	0	0

Busca em profundidade p/ digrafos



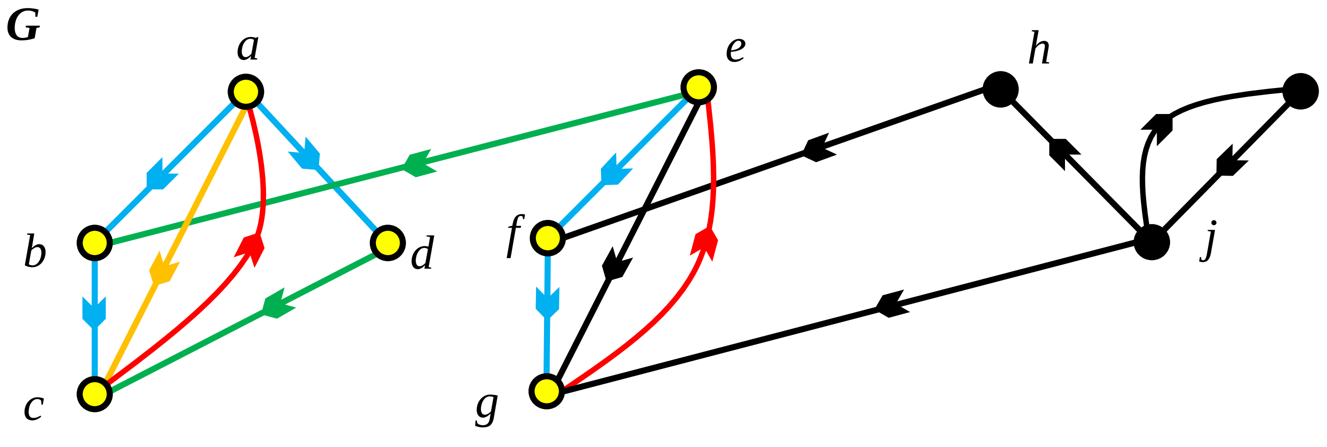
v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
$PE(v)$	1	2	3	6	9	10	11	0	0	0
$PS(v)$	8	5	4	7	0	0	0	0	0	0

Busca em profundidade p/ digrafos



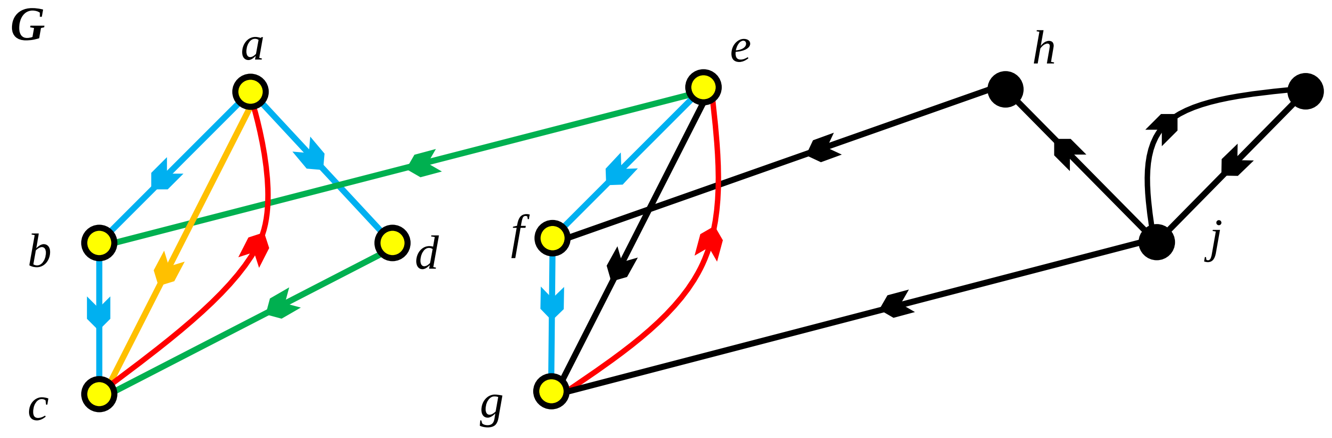
v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	0	0	0
$PS(v)$	8	5	4	7	0	0	0	0	0	0

Busca em profundidade p/ digrafos



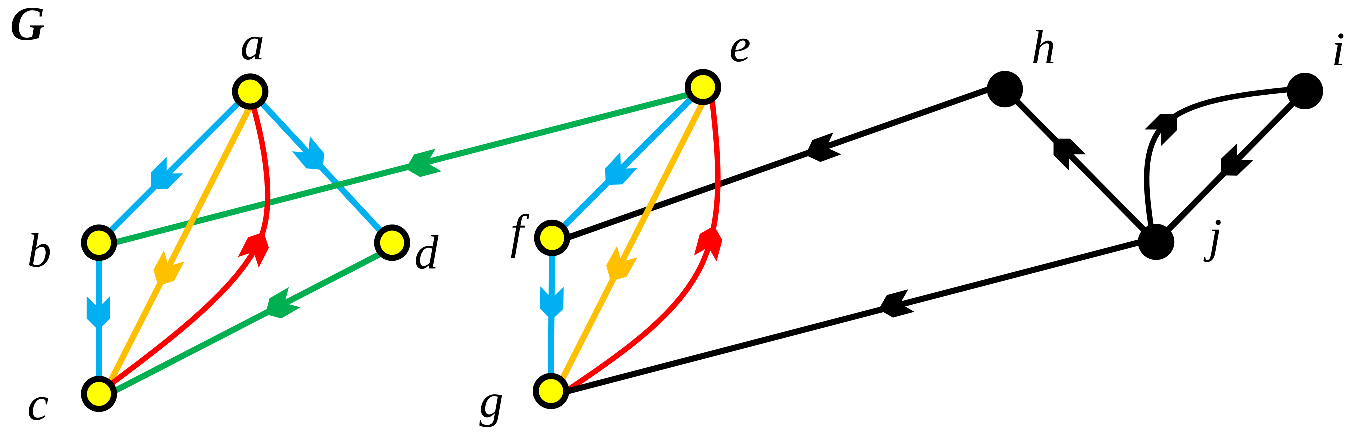
v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	1	2	3	6	9	10	11	0	0	0
PS(*v*)	8	5	4	7	0	0	12	0	0	0

Busca em profundidade p/ digrafos



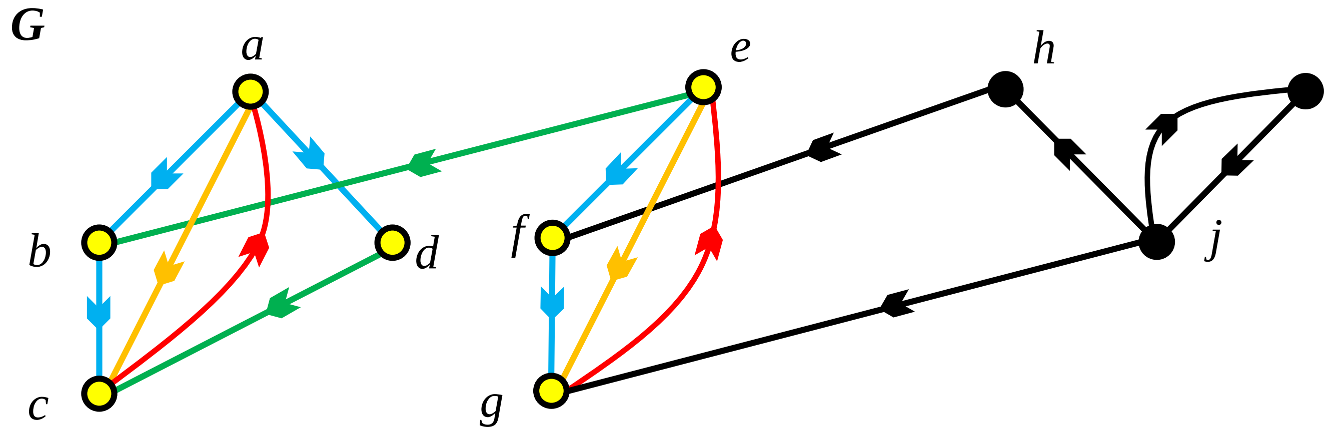
v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	9	10	11	0	0	0
PS(*v*)	8	5	4	7	0	13	12	0	0	0

Busca em profundidade p/ digrafos



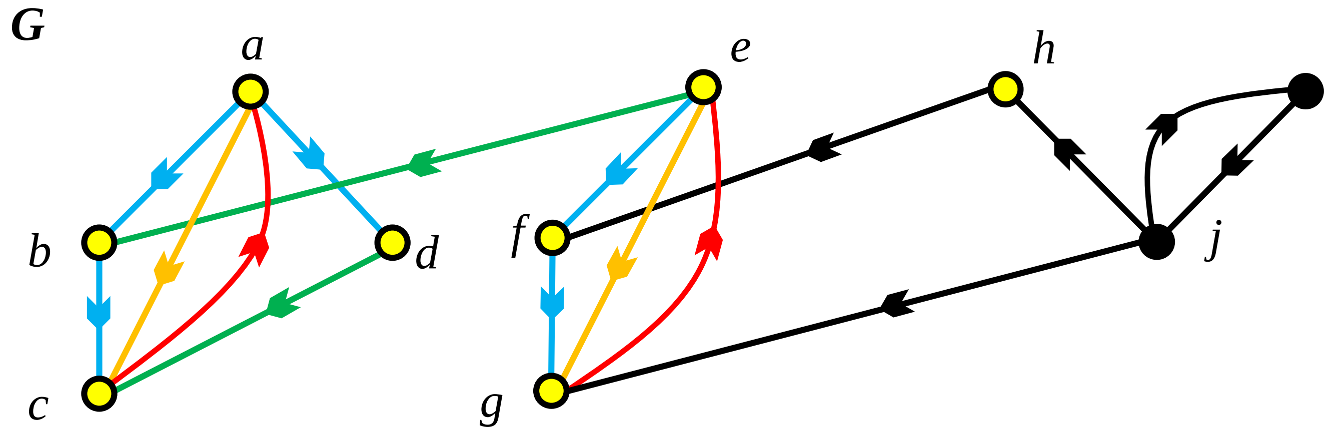
v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	0	0	0
$PS(v)$	8	5	4	7	0	13	12	0	0	0

Busca em profundidade p/ digrafos



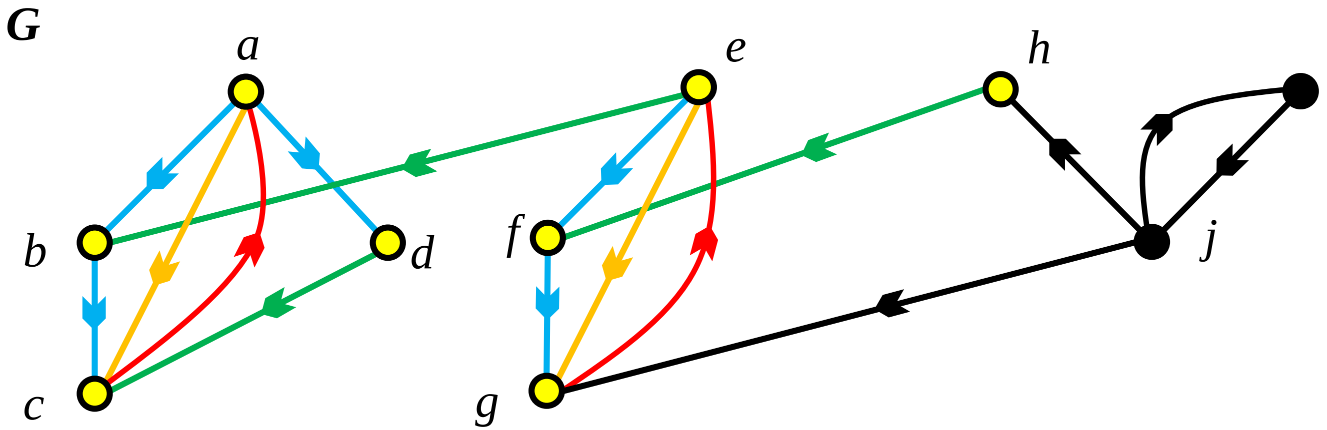
v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	0	0	0
PS(v)	8	5	4	7	14	13	12	0	0	0

Busca em profundidade p/ digrafos



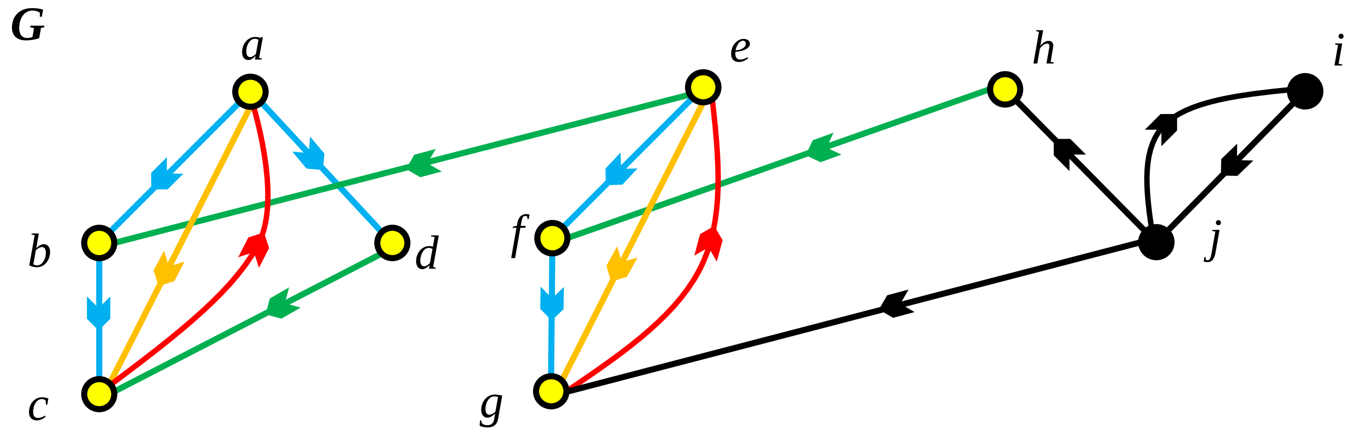
v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	9	10	11	15	0	0
PS(*v*)	8	5	4	7	14	13	12	0	0	0

Busca em profundidade p/ digrafos



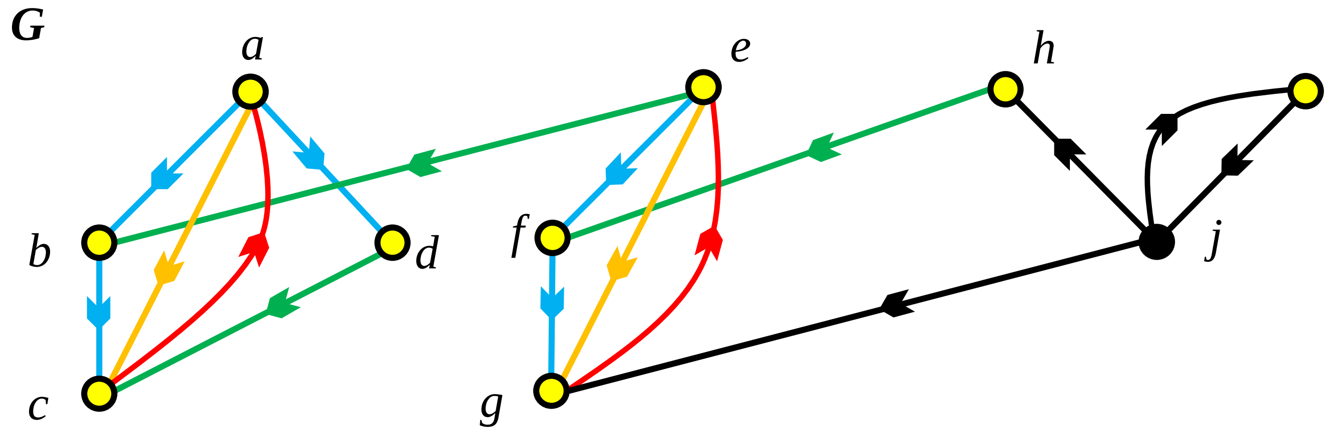
v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	15	0	0
$PS(v)$	8	5	4	7	14	13	12	0	0	0

Busca em profundidade p/ digrafos



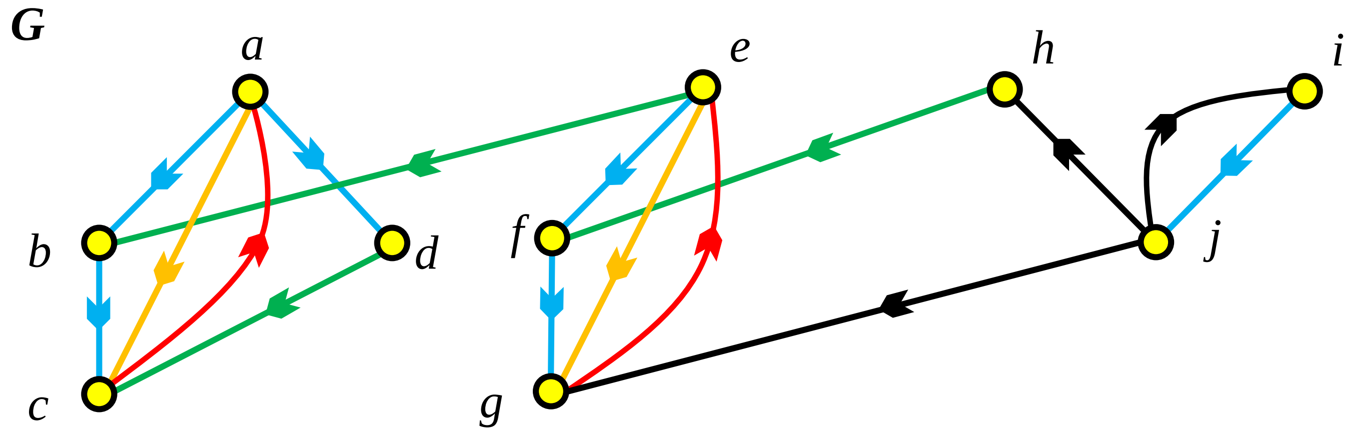
v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	15	0	0
$PS(v)$	8	5	4	7	14	13	12	16	0	0

Busca em profundidade p/ digrafos



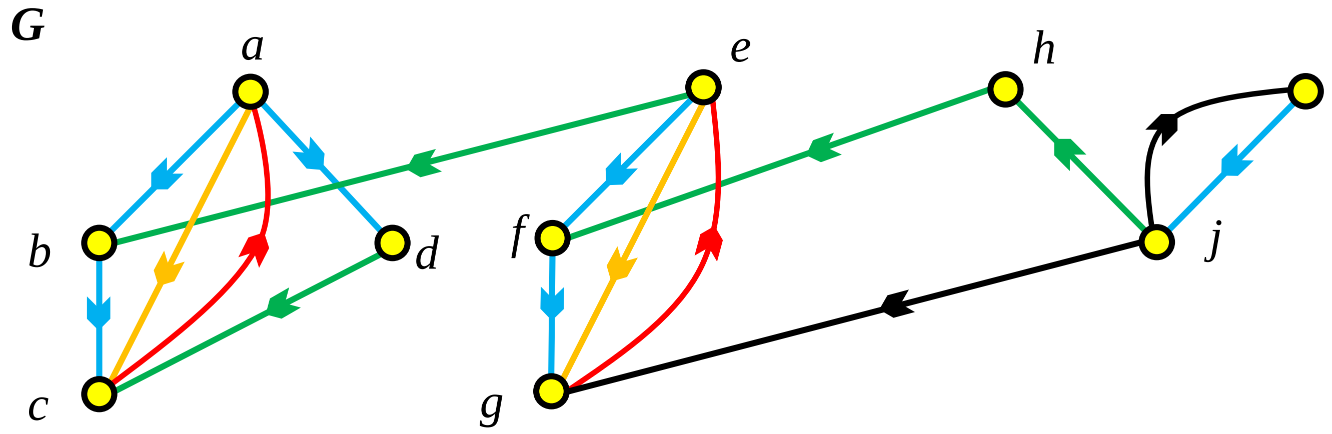
v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	9	10	11	15	17	0
PS(*v*)	8	5	4	7	14	13	12	16	0	0

Busca em profundidade p/ digrafos



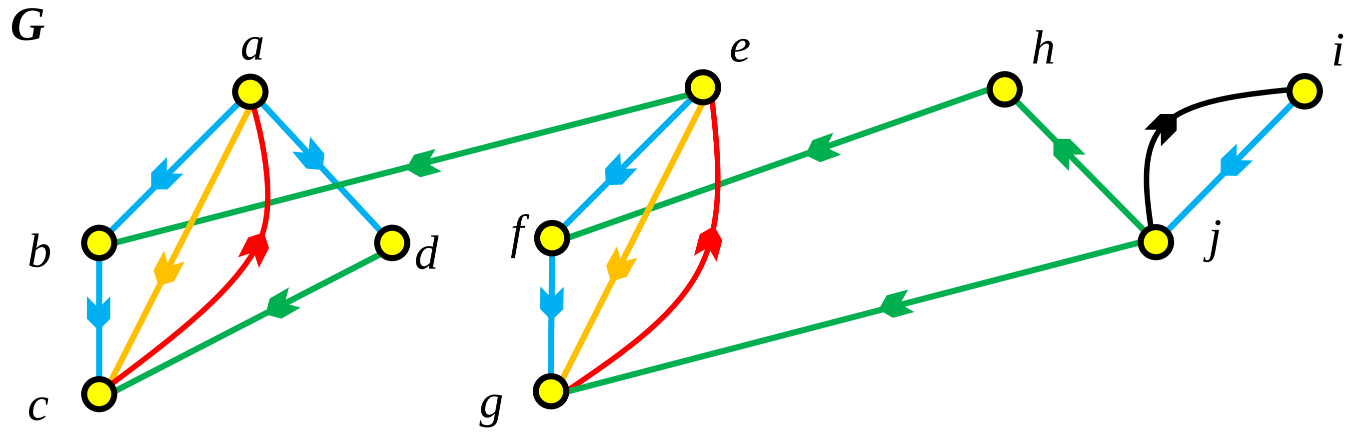
v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	15	17	18
$PS(v)$	8	5	4	7	14	13	12	16	0	0

Busca em profundidade p/ digrafos



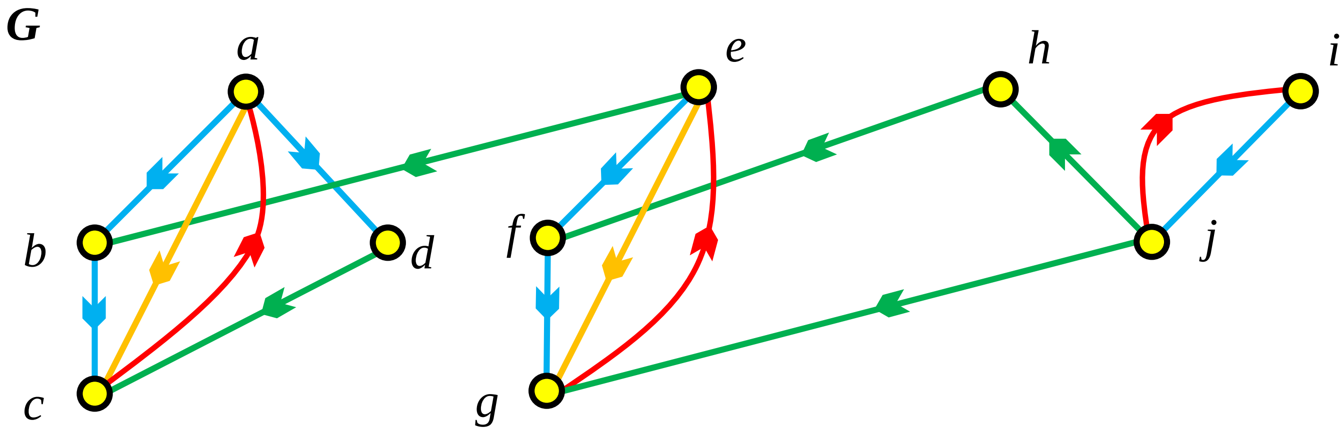
v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	15	17	18
$PS(v)$	8	5	4	7	14	13	12	16	0	0

Busca em profundidade p/ digrafos



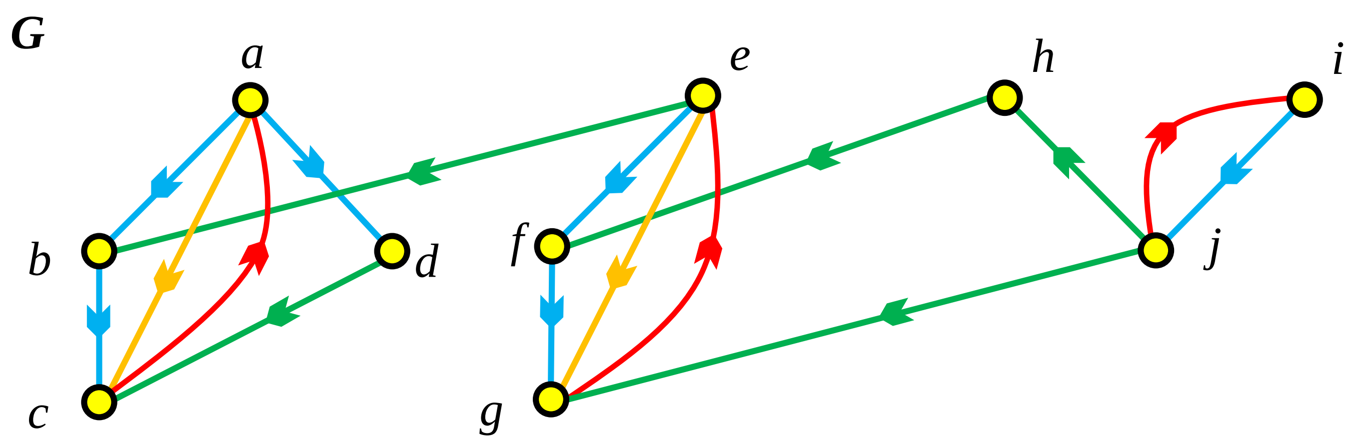
v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	9	10	11	15	17	18
PS(*v*)	8	5	4	7	14	13	12	16	0	0

Busca em profundidade p/ digrafos



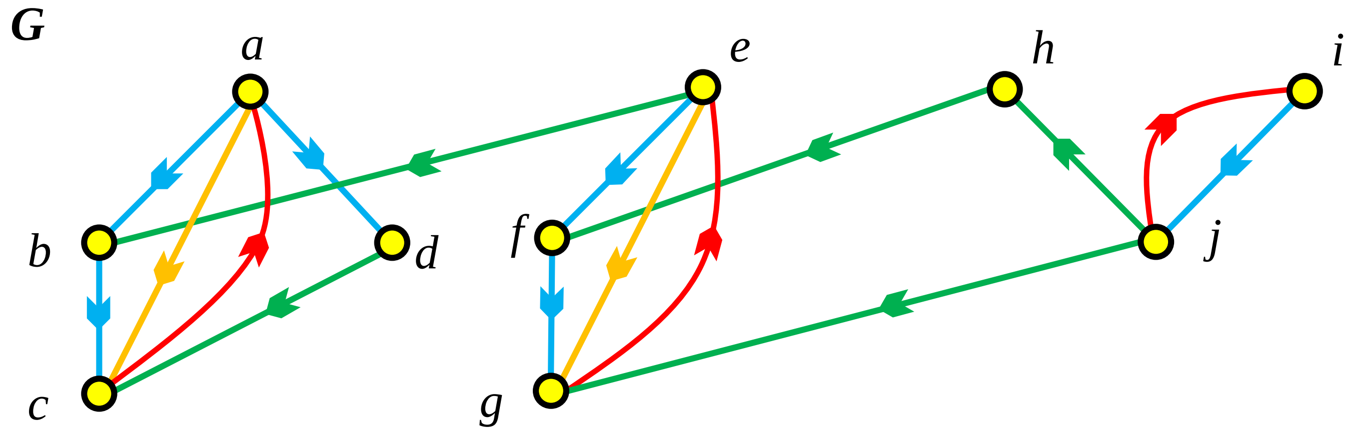
v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	15	17	18
$PS(v)$	8	5	4	7	14	13	12	16	0	0

Busca em profundidade p/ digrafos



v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	9	10	11	15	17	18
PS(*v*)	8	5	4	7	14	13	12	16	0	19

Busca em profundidade p/ digrafos



v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
$PE(v)$	1	2	3	6	9	10	11	15	17	18
$PS(v)$	8	5	4	7	14	13	12	16	20	19

Busca em profundidade p/ digrafos

Complexidade da busca em profundidade para digrafos:

$O(n + m)$

(onde $n = |V(G)|$ e $m = |E(G)|$)

Busca em profundidade p/ digrafos

- A floresta (direcionada) de profundidade *T* é uma **floresta geradora de *G*** (isto é, uma floresta que alcança todos os vértices de *G*); neste caso, todos os vértices de *G* são alcançados pela busca e ficam com uma *PE* diferente de zero no final da mesma.

- Somente as arestas **azuis** (ligando pai e filho) pertencem à floresta de profundidade *T*. As arestas de retorno **(vermelhas)**, de avanço **(amarelas)** e de cruzamento **(verdes)** não pertencem a *T*.

Busca em profundidade p/ digrafos

Propriedades das arestas de retorno

- Toda aresta de retorno fecha um **ciclo direcionado.**

- Toda aresta de retorno liga um vértice *v* a um de seus **ancestrais** na floresta de profundidade *T*.

Busca em profundidade p/ digrafos

Intervalo de vida de um vértice v : $I(v) = [\ PE(v)\ ,\ PS(v)\ ]$

v	***a***	***b***	***c***	***d***	***e***	***F***	***g***	***h***	***i***	***j***
$PE(v)$	1	2	3	6	9	10	11	15	17	18
$PS(v)$	8	5	4	7	14	13	12	16	20	19

$t_{\rightarrow}$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
a																				
b																				
c																				
d																				
e																				
f																				
g																				
h																				
i																				
j																				

Busca em profundidade p/ digrafos

Caracterização das arestas da floresta de profundidade (arestas azuis)

Seja vw aresta de G. Então:

vw é uma aresta da floresta de profundidade
se e somente se
$I(v)$ **contém** $I(w)$ e, no momento da visita, $PE(w) = 0$

Busca em profundidade p/ digrafos

Caracterização das arestas de avanço (arestas amarelas)

Seja vw aresta de G. Então:

vw é uma aresta de avanço
se e somente se
$I(v)$ **contém** $I(w)$ e, no momento da visita, $PE(w) \neq 0$

Busca em profundidade p/ digrafos

Caracterização das arestas de retorno (arestas vermelhas)

Seja *vw* aresta de *G*. Então:

vw é uma aresta de retorno
se e somente se
$I(v)$ **está contido em** $I(w)$

Busca em profundidade p/ digrafos

Caracterização das arestas de cruzamento (arestas verdes)

Seja vw aresta de G. Então:

vw é uma aresta de cruzamento
se e somente se
$I(v)$ **está totalmente à direita de** $I(w)$

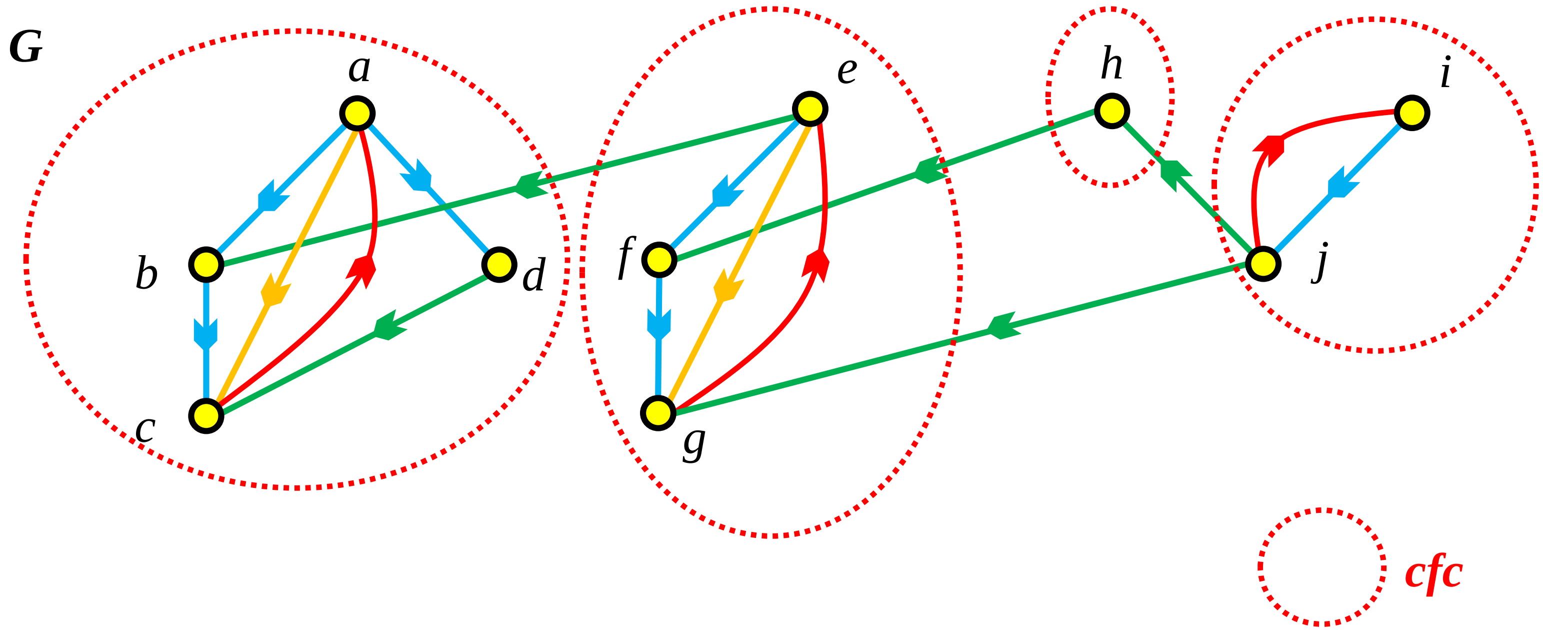
Busca em profundidade p/ digrafos

Aplicação 1: Dado um digrafo G, determinar as *componentes fortemente conexas* (*cfc's*) de G.

- Uma componente fortemente conexa de um digrafo G é um subdigrafo H de G que é maximal com relação à seguinte propriedade:

"Para qualquer par de vértices *v*,*w* de *H*, existe em *H* um caminho direcionado de *v* para *w* e um caminho direcionado de *w* para *v*"

Busca em profundidade p/ digrafos



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	15	17	18
$PS(v)$	8	5	4	7	14	13	12	16	20	19

Busca em profundidade p/ digrafos

Aplicação 1: Dado um digrafo G, determinar as componentes fortemente conexas de G.

- As cfc's de um digrafo G determinam naturalmente uma *partição do conjunto de vértices* de G, isto é, cada vértice pertence a uma única cfc de G.
- O mesmo não ocorre em relação às arestas: algumas arestas não pertencem a nenhuma cfc.
- Se após a busca alguma aresta não pertence a nenhuma cfc, então esta aresta pode ser da **floresta de profundidade**, ou de **avanço**, ou de **cruzamento**.

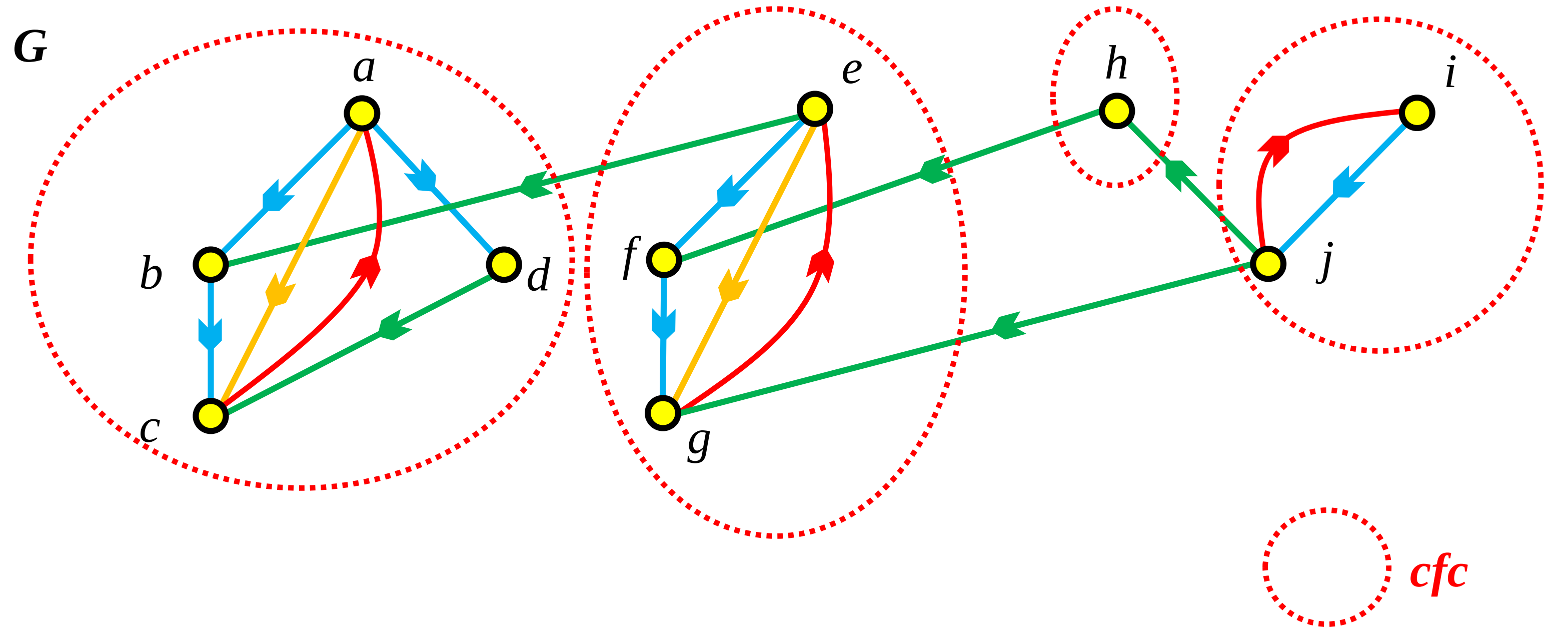
Busca em profundidade p/ digrafos

Aplicação 1: Dado um digrafo G, determinar as componentes fortemente conexas de G.

Definição:
Seja T uma floresta de profundidade para o digrafo G.

$old(v) = PE$ do vértice w mais antigo na busca que esteja *na mesma cfc de v* e que possa ser alcançado a partir de v usando *0 ou mais* arestas **azuis** de T para baixo e, a seguir, *no máximo uma* aresta de **retorno** ou de **cruzamento.**

Busca em profundidade p/ digrafos



v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	9	10	11	15	17	18
PS(*v*)	8	5	4	7	14	13	12	16	20	19
old(*v*)	1	1	1	3	9	9	9	15	17	17

Busca em profundidade p/ digrafos

Aplicação 1: Dado um digrafo G, determinar as componentes fortemente conexas de G.

Como calcular *old*(v)

$$
\begin{aligned}
old(v) = \min\Big(\quad & \{\, PE(v) \,\} \cup \\
& \{\, old(w) \mid w \text{ é filho de } v \text{ em } T \,\} \cup \\
& \{\, PE(w) \mid vw \text{ é aresta de retorno} \,\} \cup \\
& \{\, PE(w) \mid vw \text{ é aresta de cruzamento} \\
& \qquad \text{e } v,w \text{ estão na mesma cfc} \,\} \Big)
\end{aligned}
$$

Busca em profundidade p/ digrafos

<u>*procedimento*</u> *P*(*v*) ***-- com cálculo de old(v) e das cfc's***

t ← *t* + 1; *PE*(*v*) ← *t*; ***old(v) ← PE(v);*** ***=> inicialização de old(v)***

empilhar v em Q

<u>*para*</u> *todo vértice w em* N_{out} (*v*) <u>*faça*</u>

<u>*se*</u> *PE*(*w*) = 0

<u>*então*</u> *marcar vw como aresta da* ***floresta de profundidade***

pai(*w*) ← *v*; *executar P*(*w*)

executar P(*w*); ***old(v) ← min(old(v), old(w))***

<u>*senão*</u> <u>*se*</u> *PS*(*w*) = 0

<u>*então*</u> *marcar vw como aresta de* ***retorno***

old(v) ← min(old(v), PE(w))

<u>*senão*</u> <u>*se*</u> *PE*(*v*) < *PE*(*w*)

<u>*então*</u> *marcar vw como aresta de* ***avanço***

<u>*senão*</u> *marcar vw como aresta de* ***cruzamento***

<u>se</u> w está em Q <u>então</u> old(v)←min(old(v), PE(w))

<u>*fim-para*</u>

t ← *t* + 1; *PS*(*v*) ← *t*

<u>se</u> old(v) = PE(v) <u>então</u> desempilhar todos os vértices até v (inclusive) ***==> formam cfc!***

<u>*fim-do-procedimento*</u>

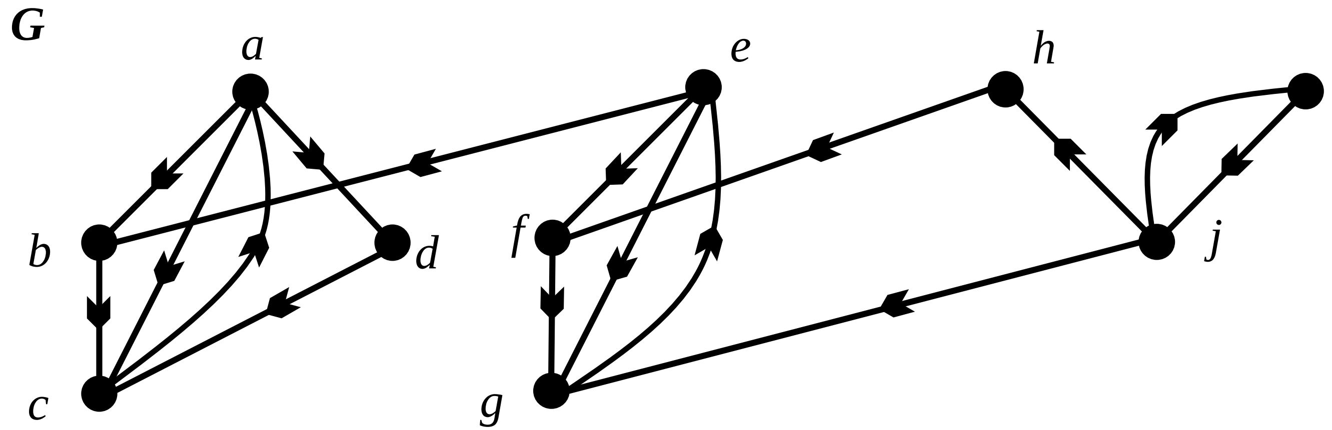
Busca em profundidade p/ digrafos

Aplicação 1: Dado um digrafo *G*, determinar as componentes fortemente conexas de *G*.

Vértices fortes

- Um vértice *v* é **forte** se ***old*(*v*) = *PE*(*v*)** no momento em que *v* sai da busca (última linha do procedimento de busca no slide anterior).
- Ao encontrar um vértice forte *v*, imediatamente desempilhamos todos os vértices até *v*. Os vértices desempilhados formam uma nova cfc.

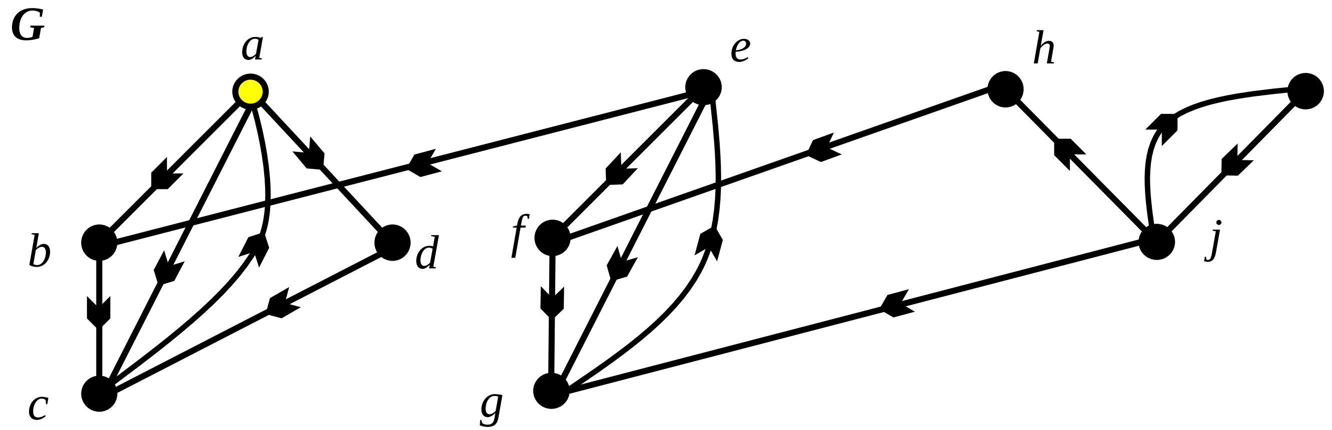
Busca em profundidade p/ digrafos



$Q = \emptyset$

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	0	0	0	0	0	0	0	0	0	0
PS(*v*)	0	0	0	0	0	0	0	0	0	0

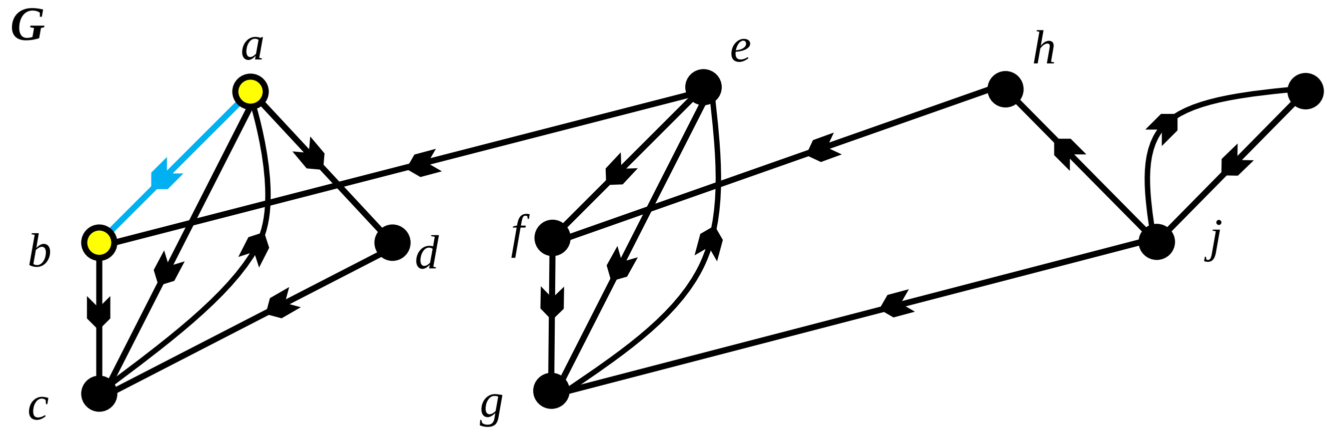
Busca em profundidade p/ digrafos



$Q = \{ a \}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	1	0	0	0	0	0	0	0	0	0
PS(*v*)	0	0	0	0	0	0	0	0	0	0

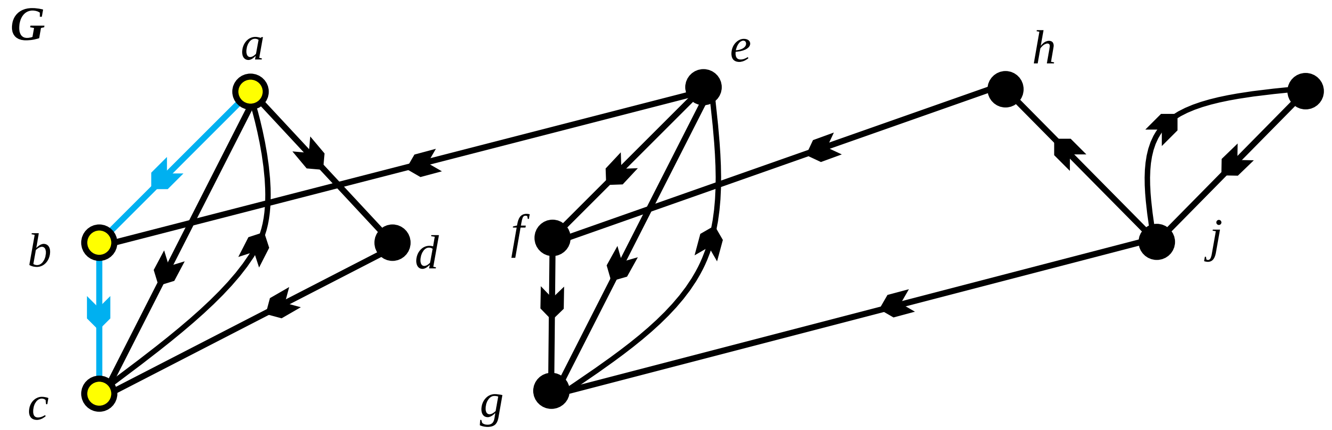
Busca em profundidade p/ digrafos



$Q = \{ a, b \}$

v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	0	0	0	0	0	0	0	0
$PS(v)$	0	0	0	0	0	0	0	0	0	0

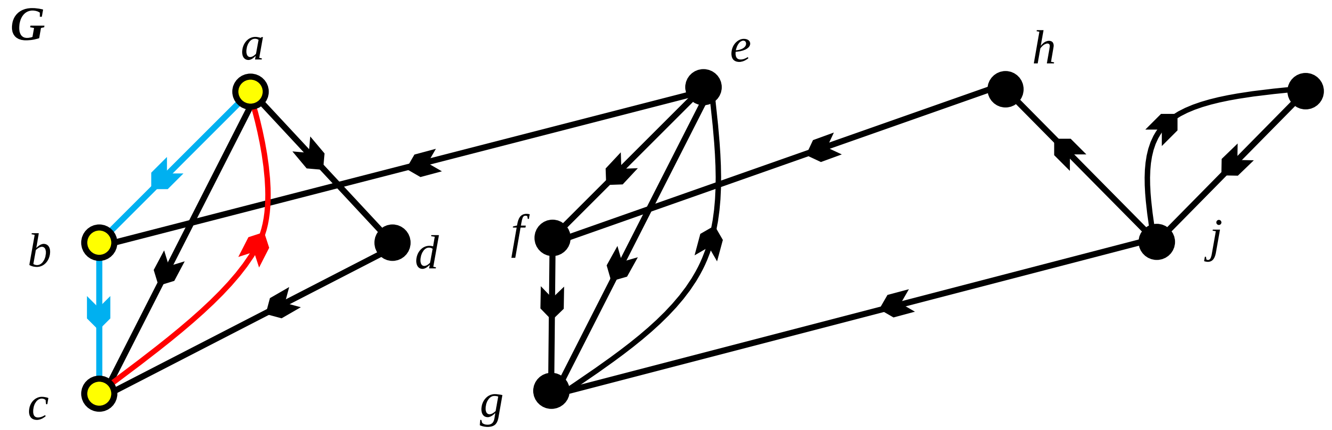
Busca em profundidade p/ digrafos



$Q = \{\, a, b, c \,\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	1	2	3	0	0	0	0	0	0	0
PS(*v*)	0	0	0	0	0	0	0	0	0	0

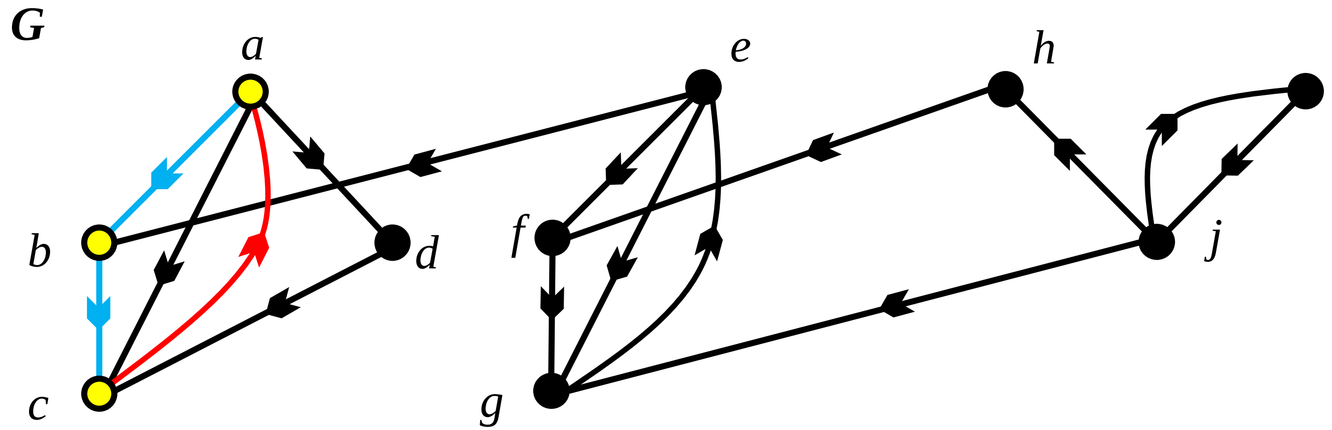
Busca em profundidade p/ digrafos



Q = { a, b, c }

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	1	2	3	0	0	0	0	0	0	0
PS(*v*)	0	0	0	0	0	0	0	0	0	0

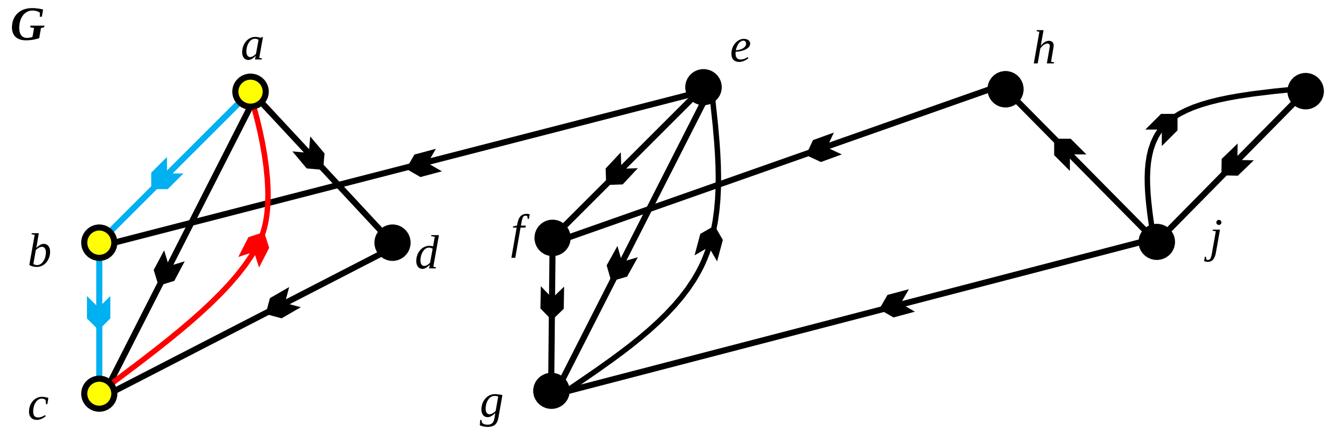
Busca em profundidade p/ digrafos



$Q = \{\, a, b, c \,\}$

v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	0	0	0	0	0	0	0
$PS(v)$	0	0	4	0	0	0	0	0	0	0
$old(v)$			1							

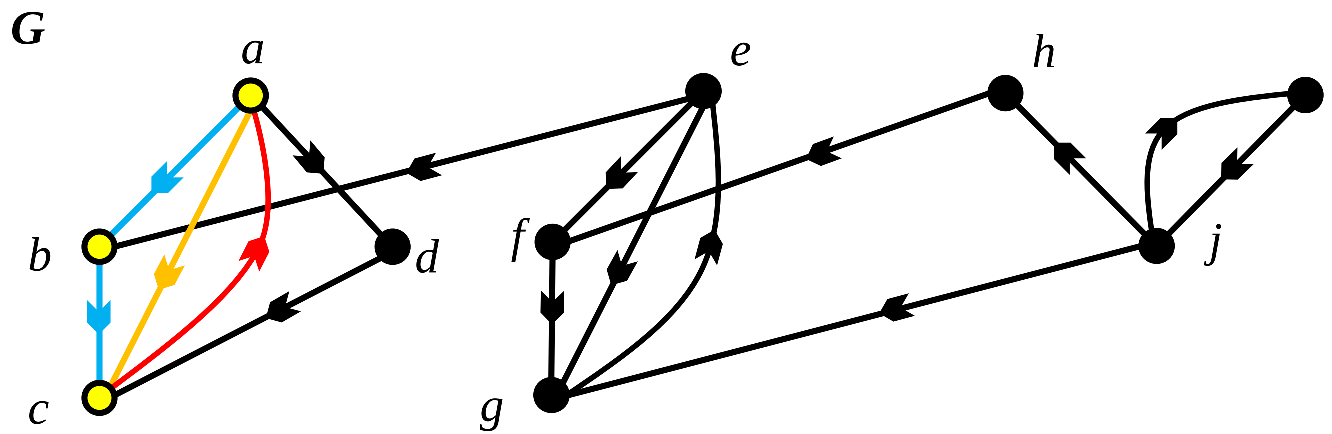
Busca em profundidade p/ digrafos



$Q = \{\ a, b, c\ \}$

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	0	0	0	0	0	0	0
PS(*v*)	0	5	4	0	0	0	0	0	0	0
old(*v*)		1	1							

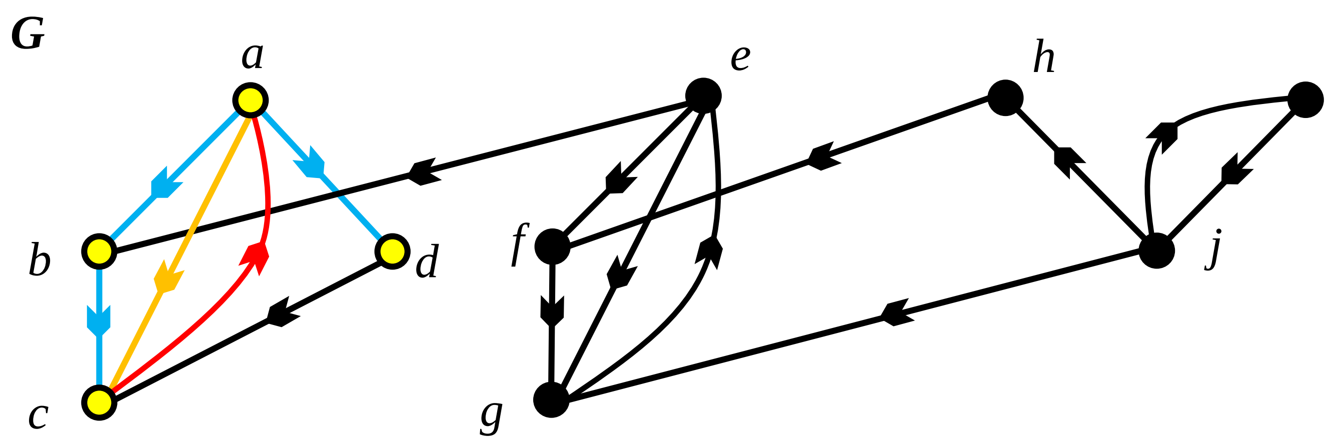
Busca em profundidade p/ digrafos



$$Q = \{\, a, b, c \,\}$$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	1	2	3	0	0	0	0	0	0	0
PS(*v*)	0	5	4	0	0	0	0	0	0	0
old(*v*)		1	1							

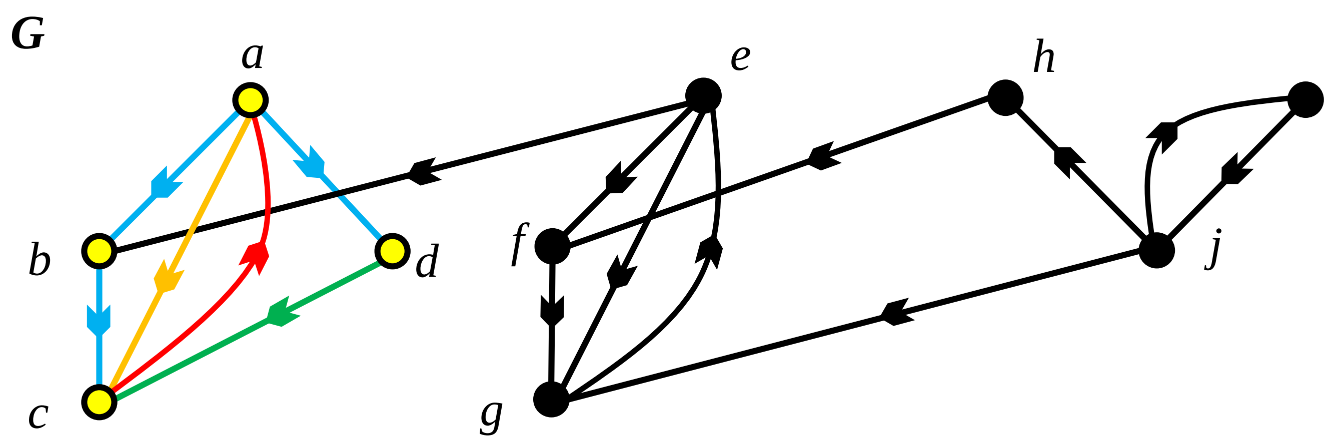
Busca em profundidade p/ digrafos



$Q = \{ a, b, c, d \}$

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	0	0	0	0	0	0
PS(*v*)	0	5	4	0	0	0	0	0	0	0
old(*v*)		1	1							

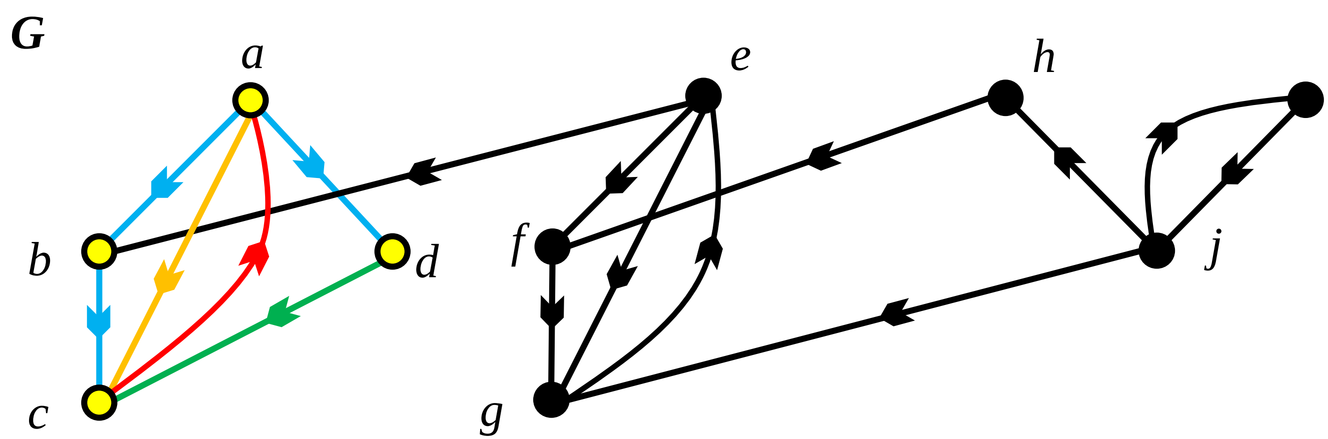
Busca em profundidade p/ digrafos



$Q = \{ a, b, c, d \}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	1	2	3	6	0	0	0	0	0	0
PS(*v*)	0	5	4	0	0	0	0	0	0	0
old(*v*)		1	1							

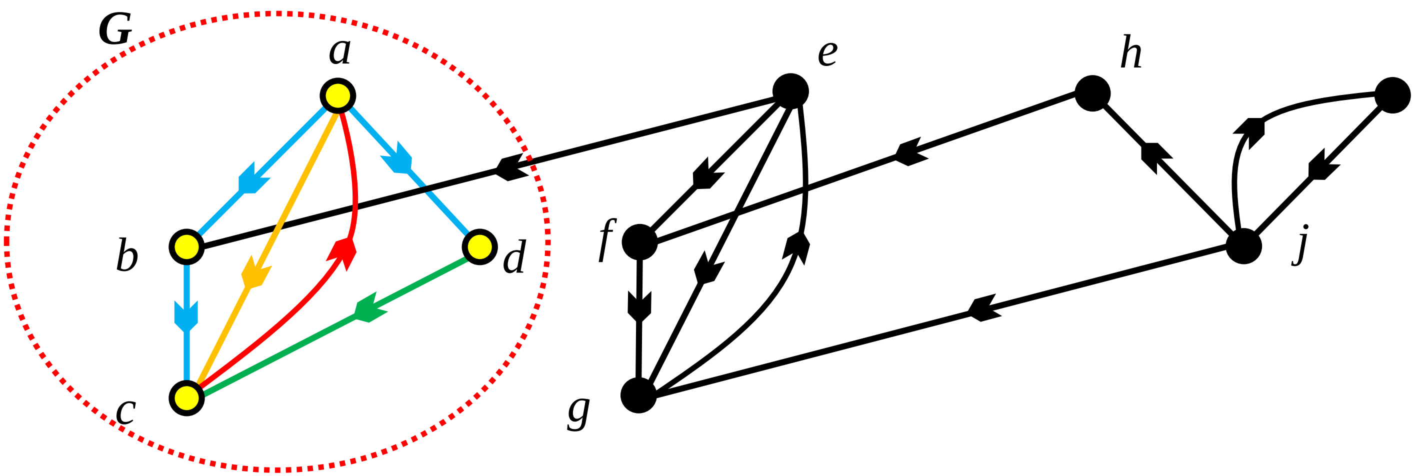
Busca em profundidade p/ digrafos



$Q = \{ a, b, c, d \}$

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	0	0	0	0	0	0
PS(*v*)	0	5	4	7	0	0	0	0	0	0
old(*v*)		1	1	3						

Busca em profundidade p/ digrafos

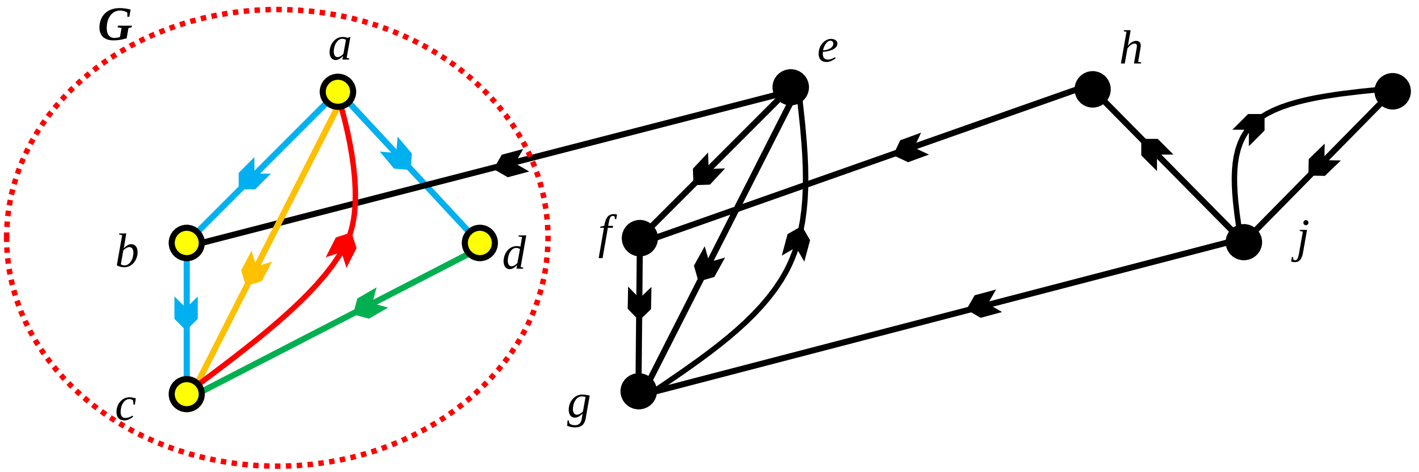


Vértice forte!

$Q = \{\, a, b, c, d \,\}$

v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	0	0	0	0	0	0
$PS(v)$	8	5	4	7	0	0	0	0	0	0
$old(v)$	1	1	1	3						

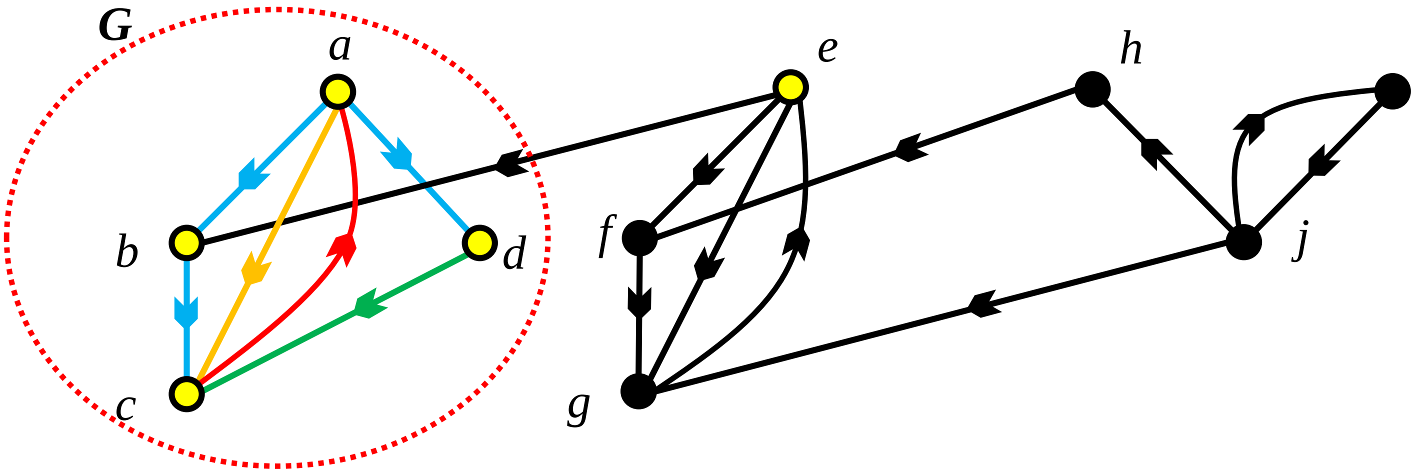
Busca em profundidade p/ digrafos



$Q = Ø$

v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	0	0	0	0	0	0
$PS(v)$	8	5	4	7	0	0	0	0	0	0
$old(v)$	1	1	1	3						

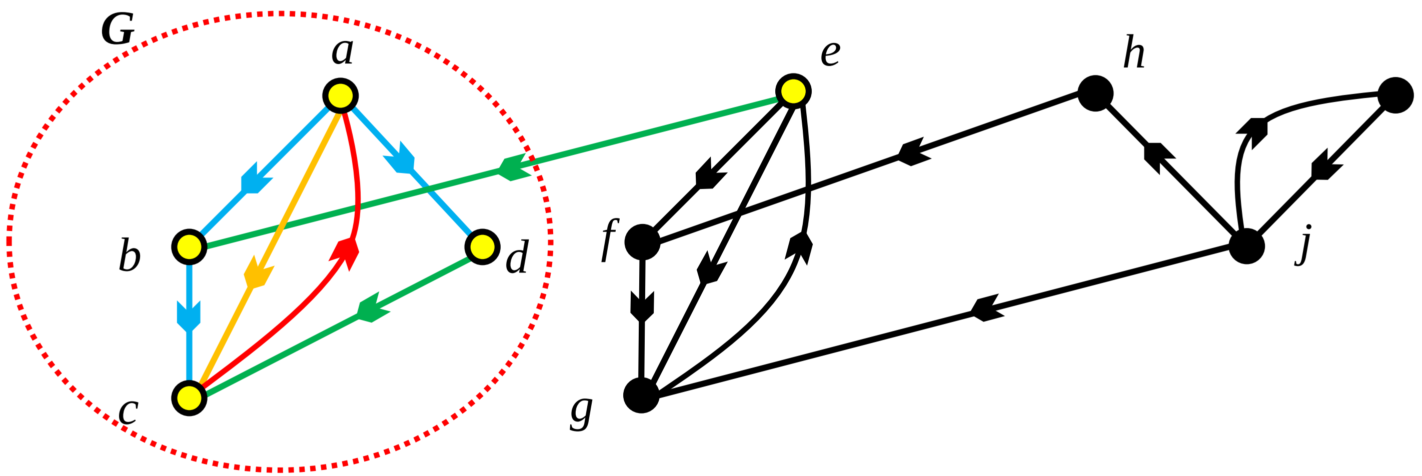
Busca em profundidade p/ digrafos



***Q* = { *e* }**

v	a	b	c	d	e	f	g	h	i	j
PE(*v*)	1	2	3	6	9	0	0	0	0	0
PS(*v*)	8	5	4	7	0	0	0	0	0	0
old(*v*)	1	1	1	3						

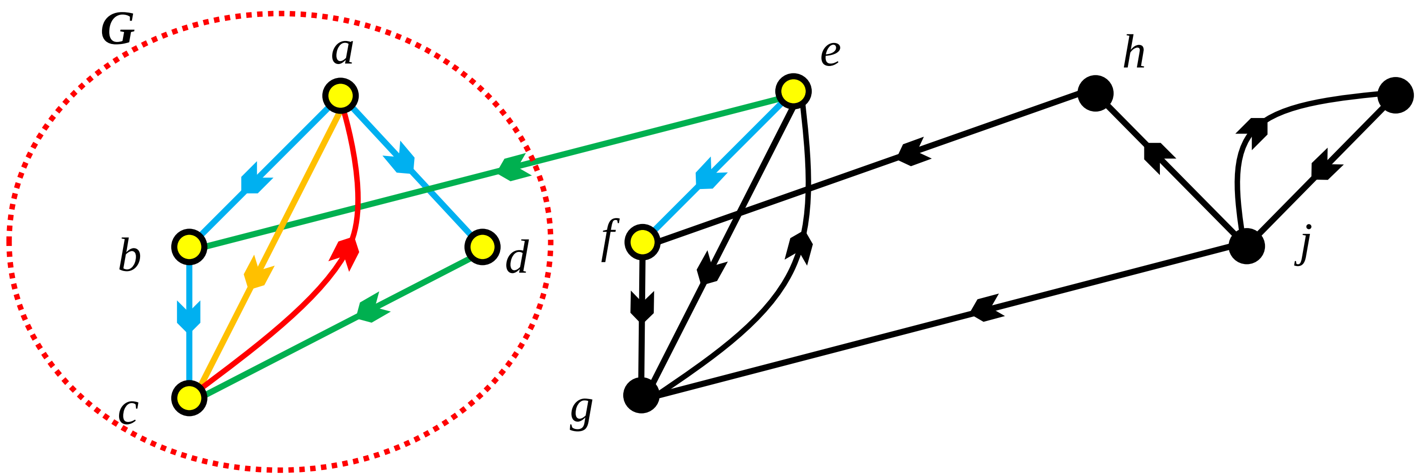
Busca em profundidade p/ digrafos



$Q = \{ e \}$

v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	0	0	0	0	0
$PS(v)$	8	5	4	7	0	0	0	0	0	0
$old(v)$	1	1	1	3						

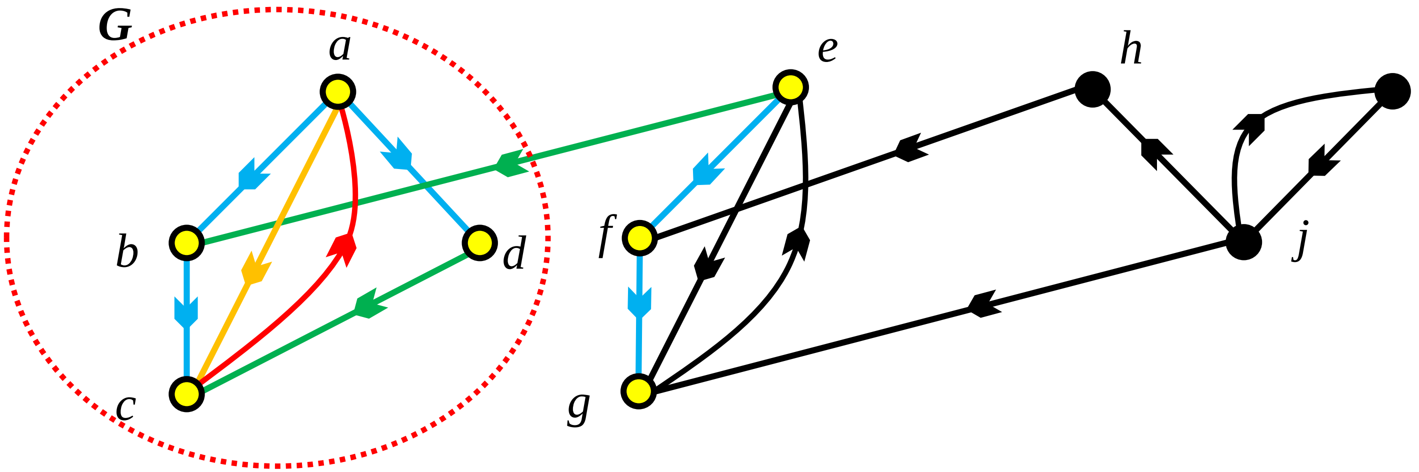
Busca em profundidade p/ digrafos



$Q = \{ e, f \}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	1	2	3	6	9	10	0	0	0	0
PS(*v*)	8	5	4	7	0	0	0	0	0	0
old(*v*)	1	1	1	3						

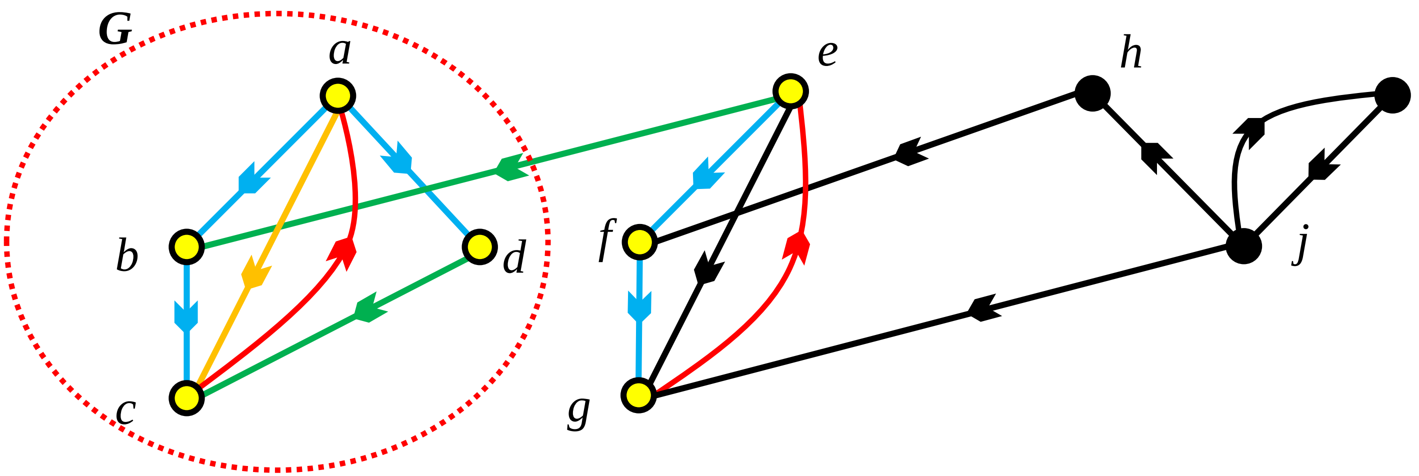
Busca em profundidade p/ digrafos



$Q = \{\, e, f, g \,\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	1	2	3	6	9	10	11	0	0	0
PS(*v*)	8	5	4	7	0	0	0	0	0	0
old(*v*)	1	1	1	3						

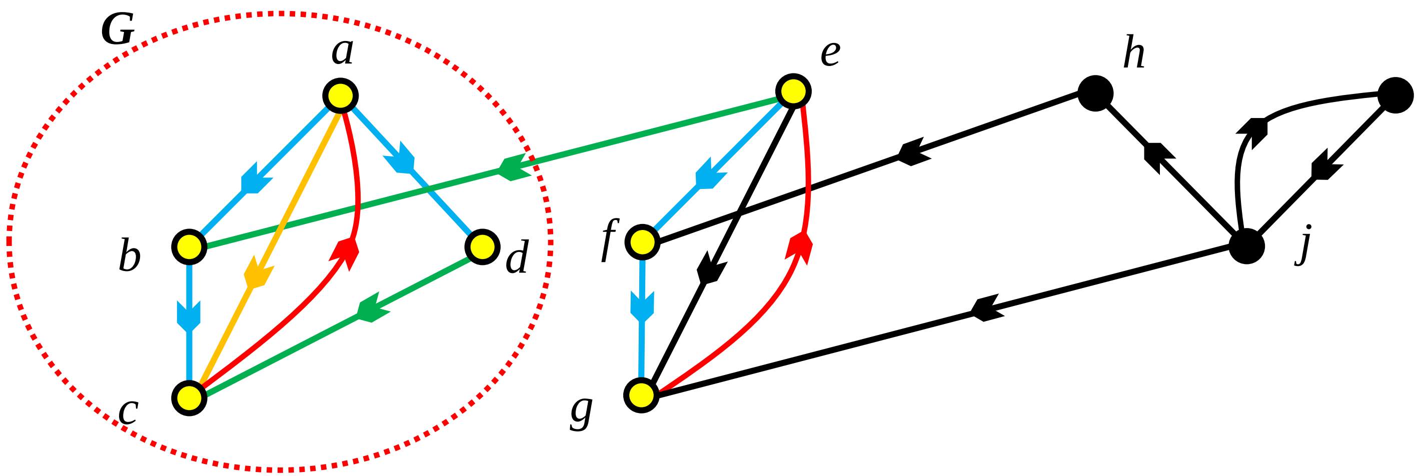
Busca em profundidade p/ digrafos



$$Q = \{\, e, f, g \,\}$$

v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	0	0	0
$PS(v)$	8	5	4	7	0	0	0	0	0	0
$old(v)$	1	1	1	3						

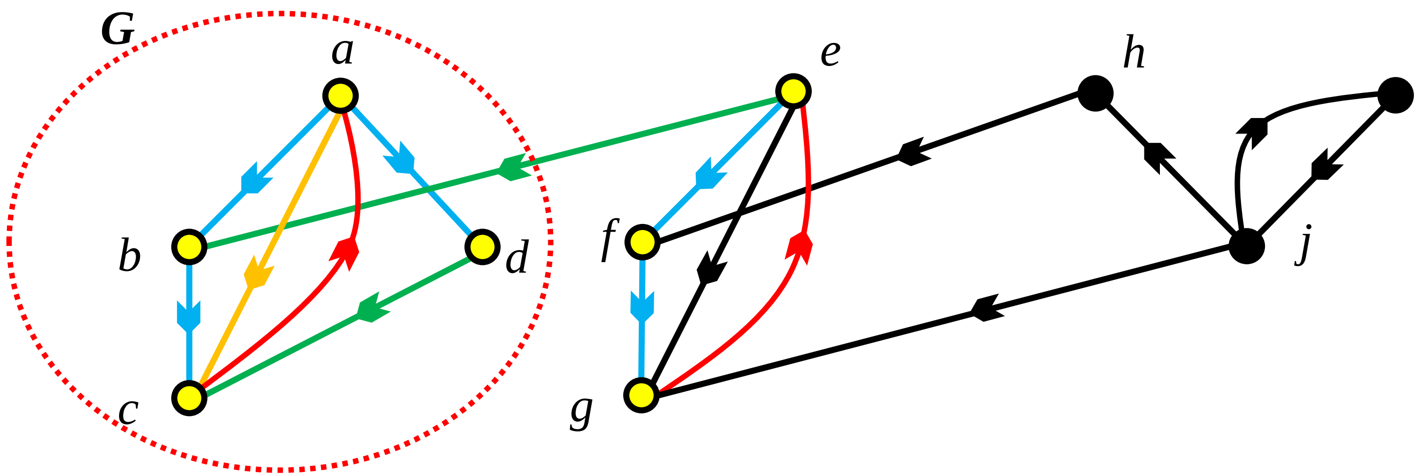
Busca em profundidade p/ digrafos



$Q = \{\, e, f, g \,\}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	1	2	3	6	9	10	11	0	0	0
PS(*v*)	8	5	4	7	0	0	12	0	0	0
old(*v*)	1	1	1	3			9			

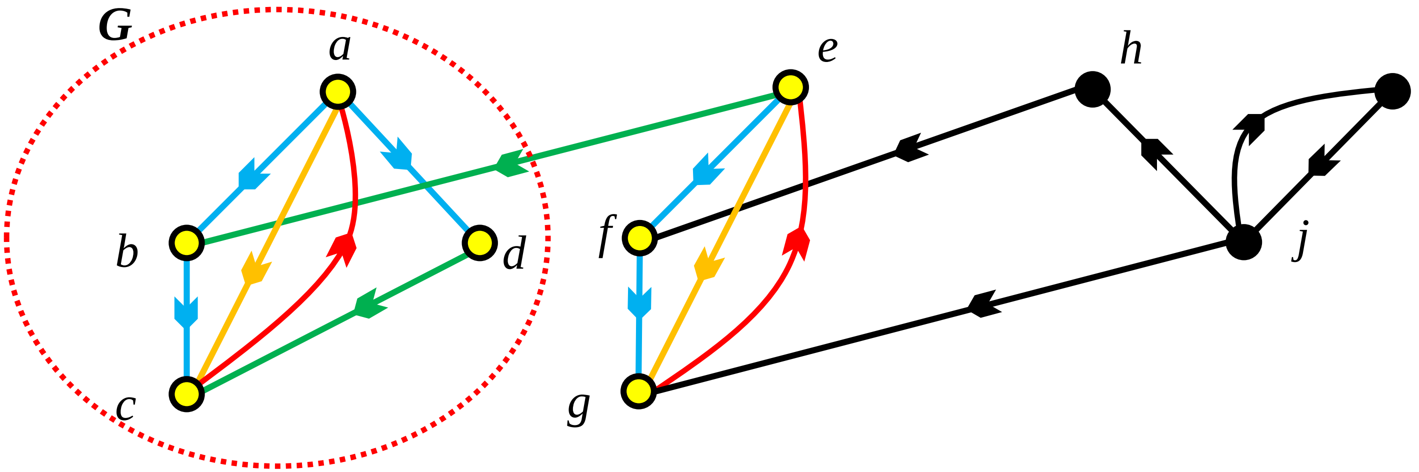
Busca em profundidade p/ digrafos



Q = { e, f, g }

v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	0	0	0
$PS(v)$	8	5	4	7	0	13	12	0	0	0
$old(v)$	1	1	1	3		9	9			

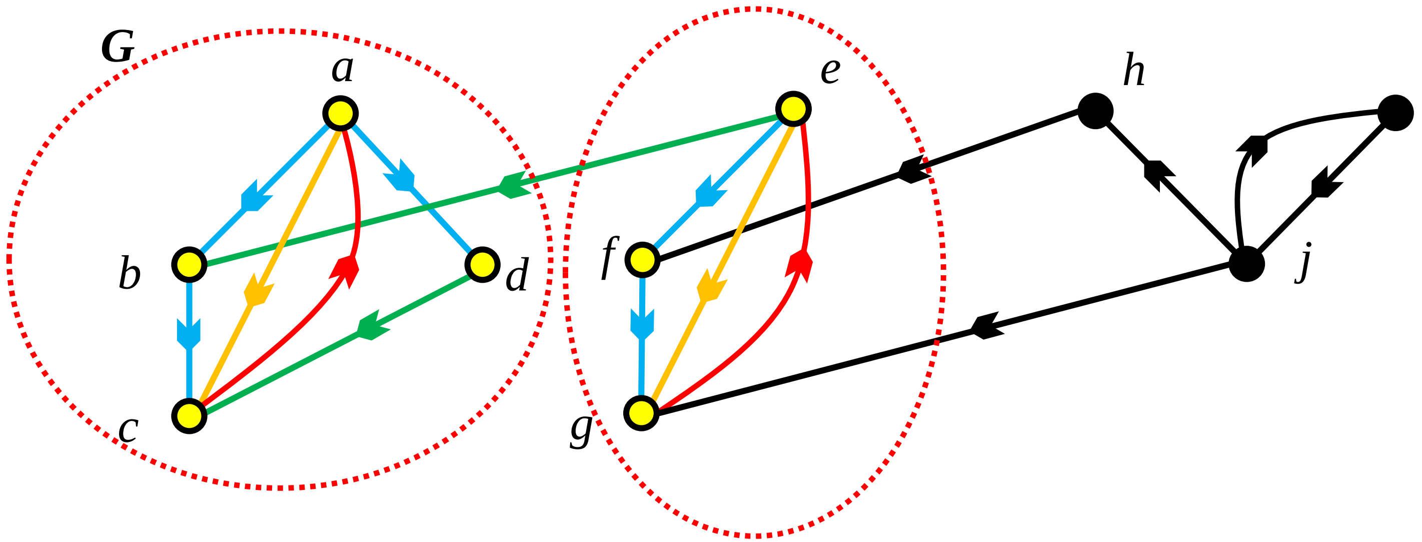
Busca em profundidade p/ digrafos



$Q = \{\ e, f, g\ \}$

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	9	10	11	0	0	0
PS(*v*)	8	5	4	7	0	13	12	0	0	0
old(*v*)	1	1	1	3		9	9			

Busca em profundidade p/ digrafos

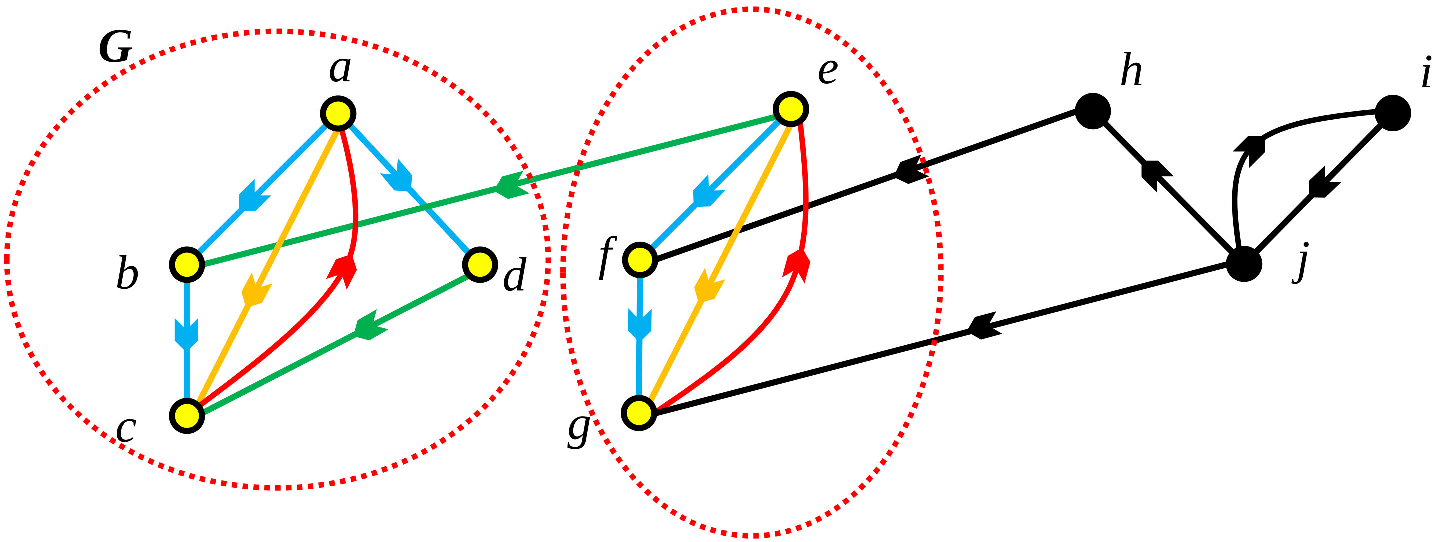


Vértice forte!

$Q = \{\, e, f, g \,\}$

v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	0	0	0
$PS(v)$	8	5	4	7	14	13	12	0	0	0
$old(v)$	1	1	1	3	9	9	9			

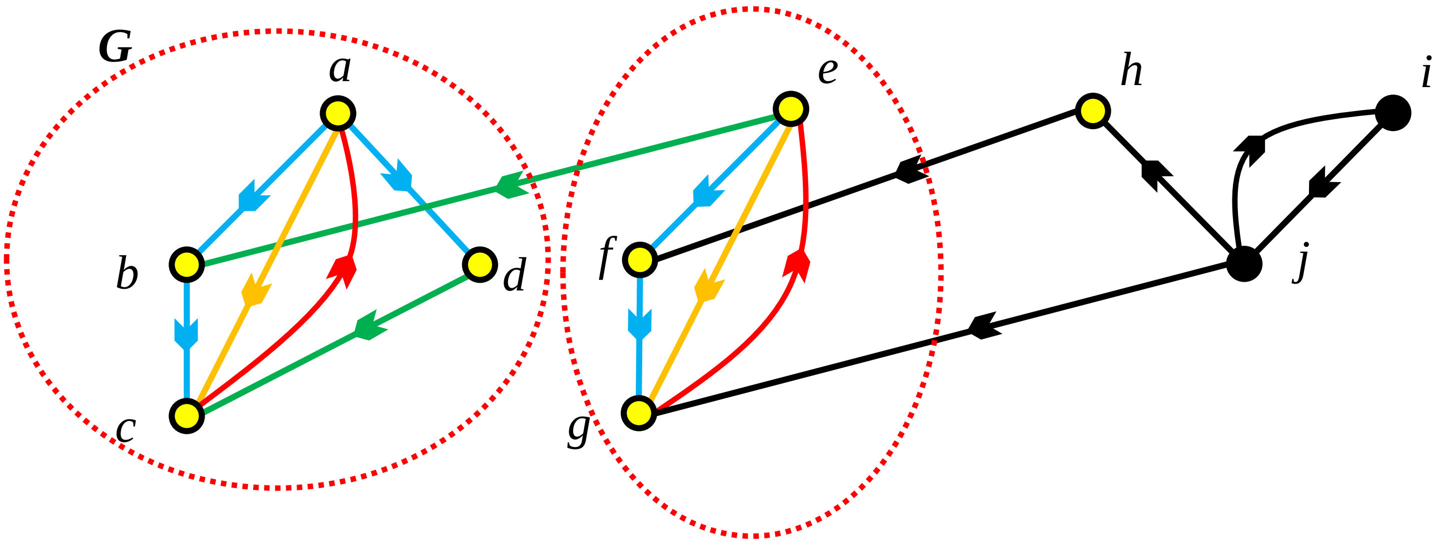
Busca em profundidade p/ digrafos



$Q = \emptyset$

v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	0	0	0
$PS(v)$	8	5	4	7	14	13	12	0	0	0
$old(v)$	1	1	1	3	9	9	9			

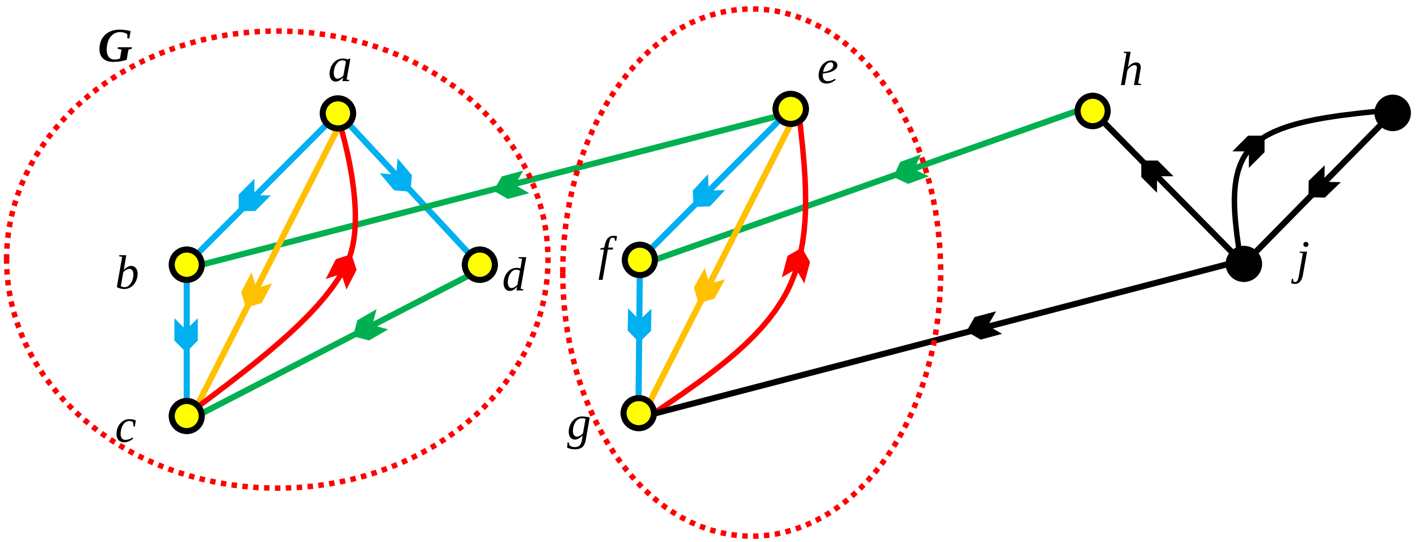
Busca em profundidade p/ digrafos



$Q = \{ h \}$

v	a	b	c	d	e	f	g	h	i	j
PE(*v*)	1	2	3	6	9	10	11	15	0	0
PS(*v*)	8	5	4	7	14	13	12	0	0	0
old(*v*)	1	1	1	3	9	9	9			

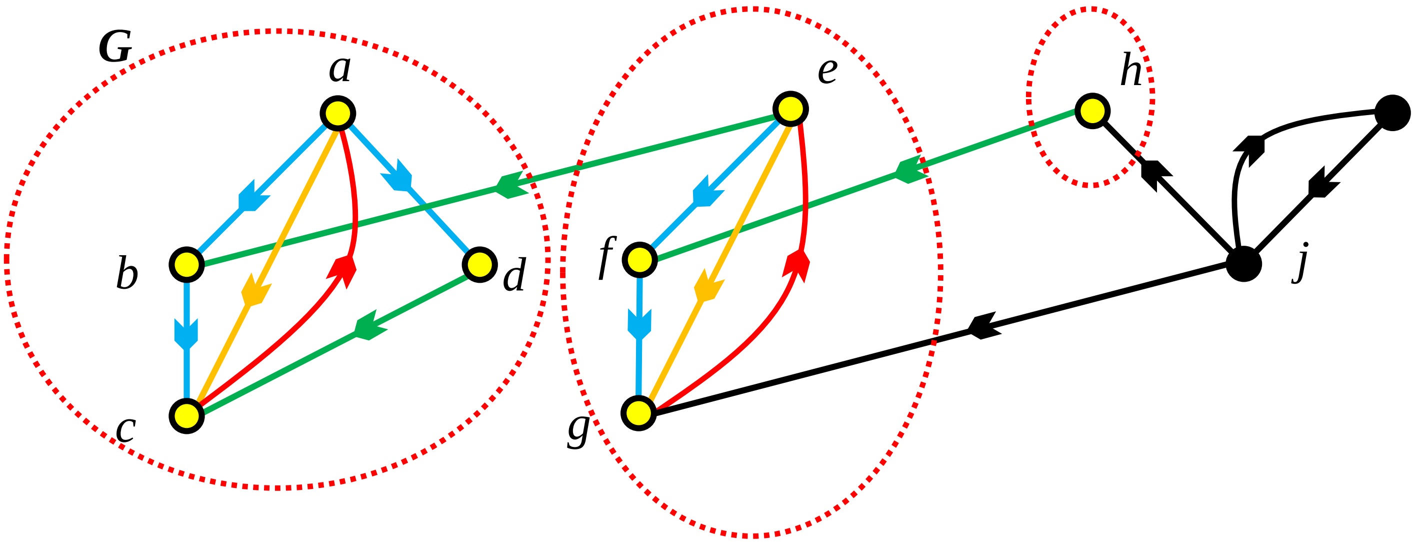
Busca em profundidade p/ digrafos



$Q = \{ h \}$

v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	15	0	0
$PS(v)$	8	5	4	7	14	13	12	0	0	0
$old(v)$	1	1	1	3	9	9	9			

Busca em profundidade p/ digrafos

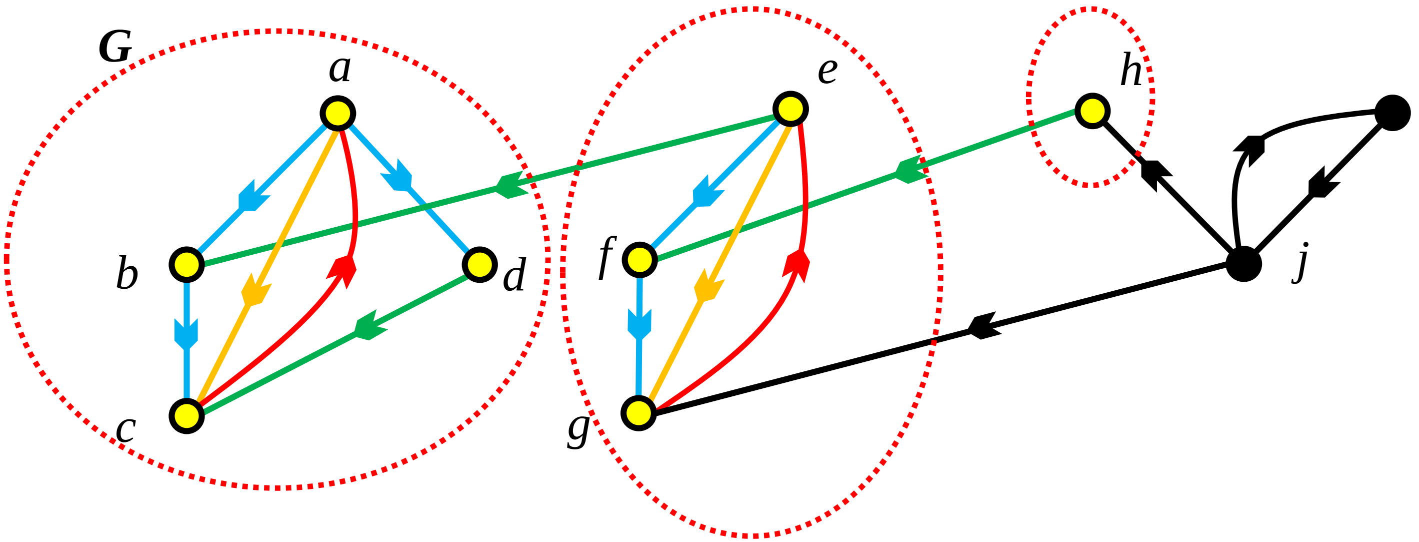


Vértice forte!

$Q = \{\ h\ \}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	1	2	3	6	9	10	11	15	0	0
PS(*v*)	8	5	4	7	14	13	12	16	0	0
old(*v*)	1	1	1	3	9	9	9	15		

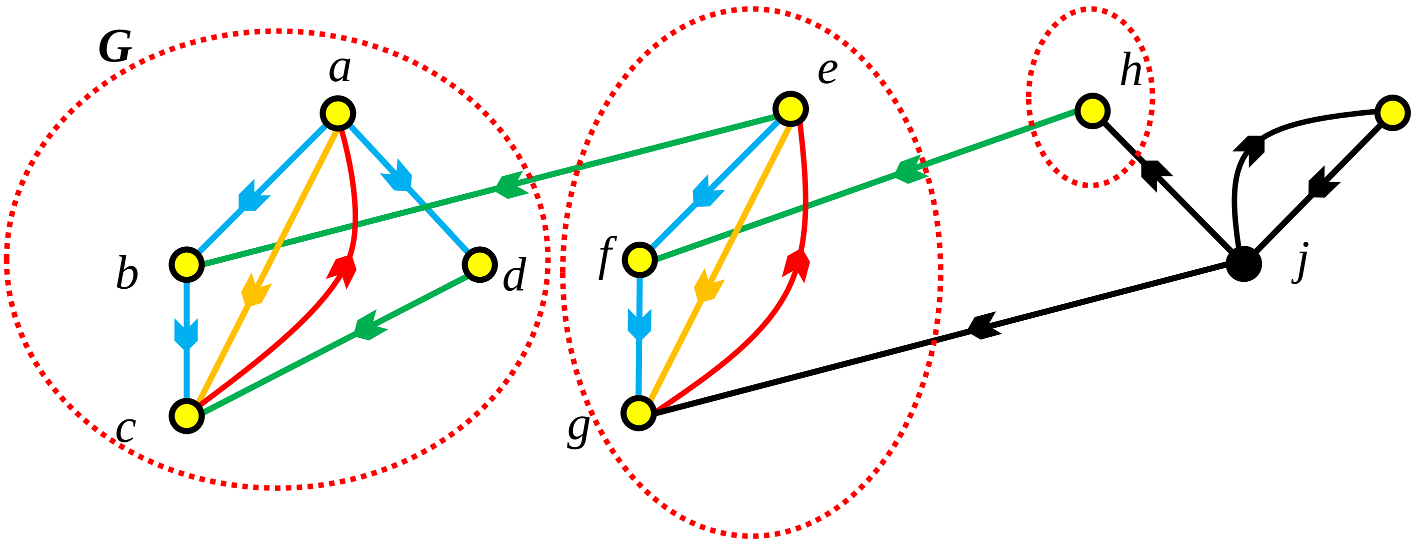
Busca em profundidade p/ digrafos



$Q = \emptyset$

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	9	10	11	15	0	0
PS(*v*)	8	5	4	7	14	13	12	16	0	0
old(*v*)	1	1	1	3	9	9	9	15		

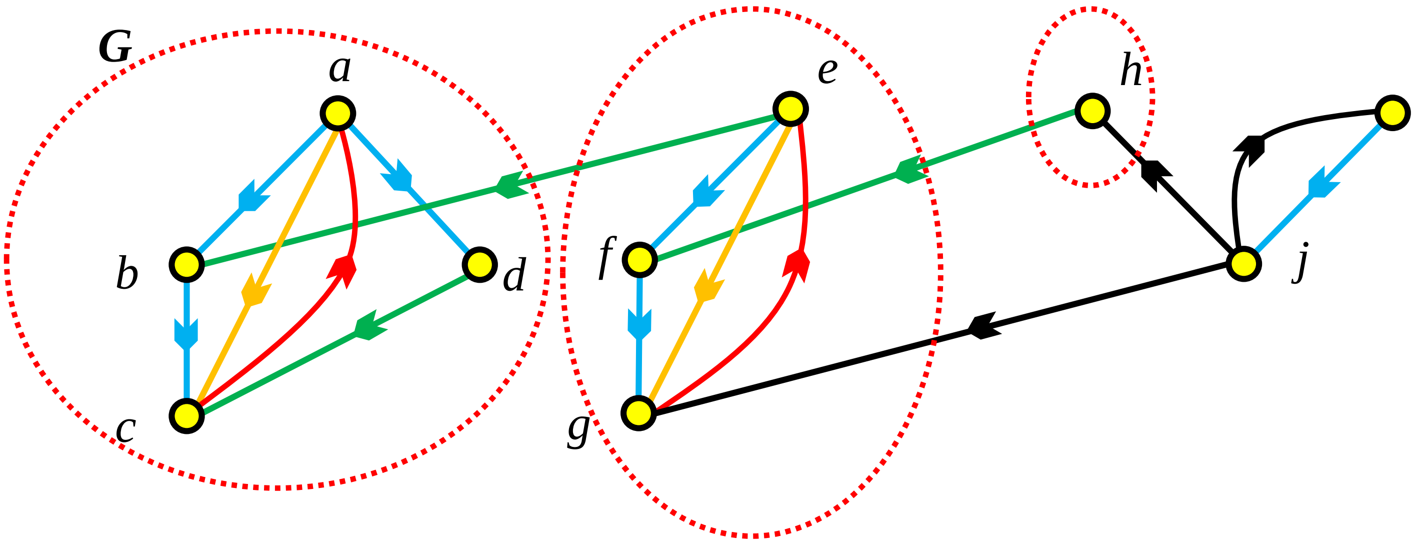
Busca em profundidade p/ digrafos



$Q = \{ i \}$

v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	15	17	0
$PS(v)$	8	5	4	7	14	13	12	16	0	0
$old(v)$	1	1	1	3	9	9	9	15		

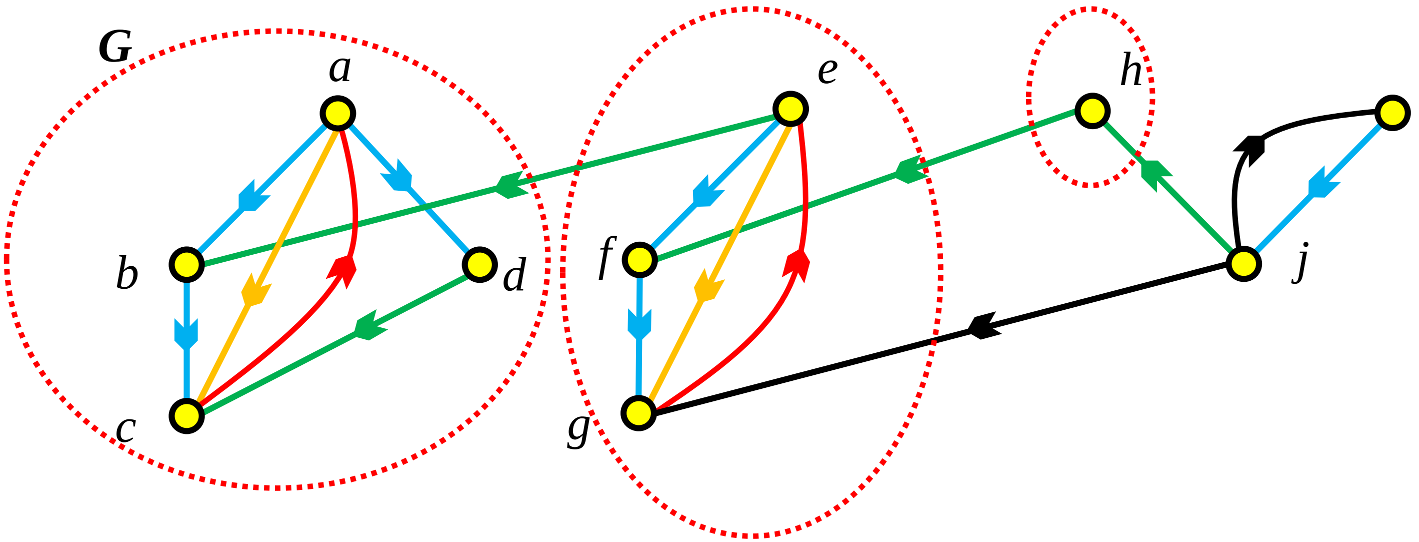
Busca em profundidade p/ digrafos



***Q* = { *i*, *j* }**

v	a	b	c	d	e	f	g	h	i	j
PE(*v*)	1	2	3	6	9	10	11	15	17	18
PS(*v*)	8	5	4	7	14	13	12	16	0	0
old(*v*)	1	1	1	3	9	9	9	15		

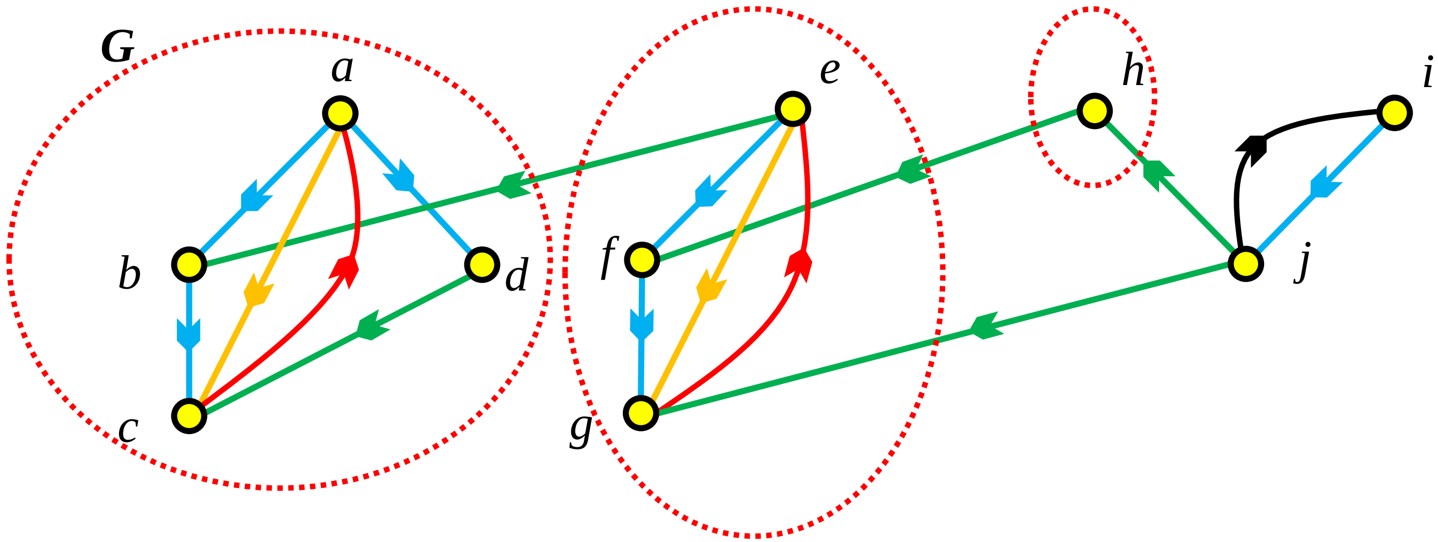
Busca em profundidade p/ digrafos



$Q = \{\, i, j \,\}$

v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	1	2	3	6	9	10	11	15	17	18
$PS(v)$	8	5	4	7	14	13	12	16	0	0
$old(v)$	1	1	1	3	9	9	9	15		

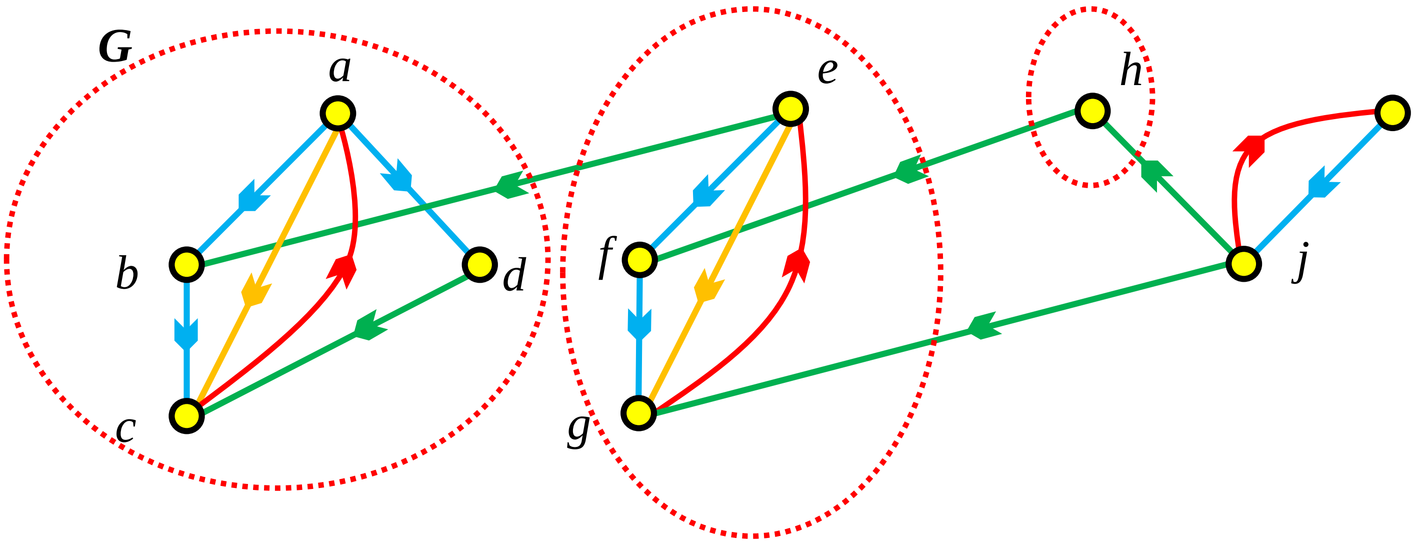
Busca em profundidade p/ digrafos



$$Q = \{\, i, j \,\}$$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	1	2	3	6	9	10	11	15	17	18
PS(*v*)	8	5	4	7	14	13	12	16	0	0
old(*v*)	1	1	1	3	9	9	9	15		

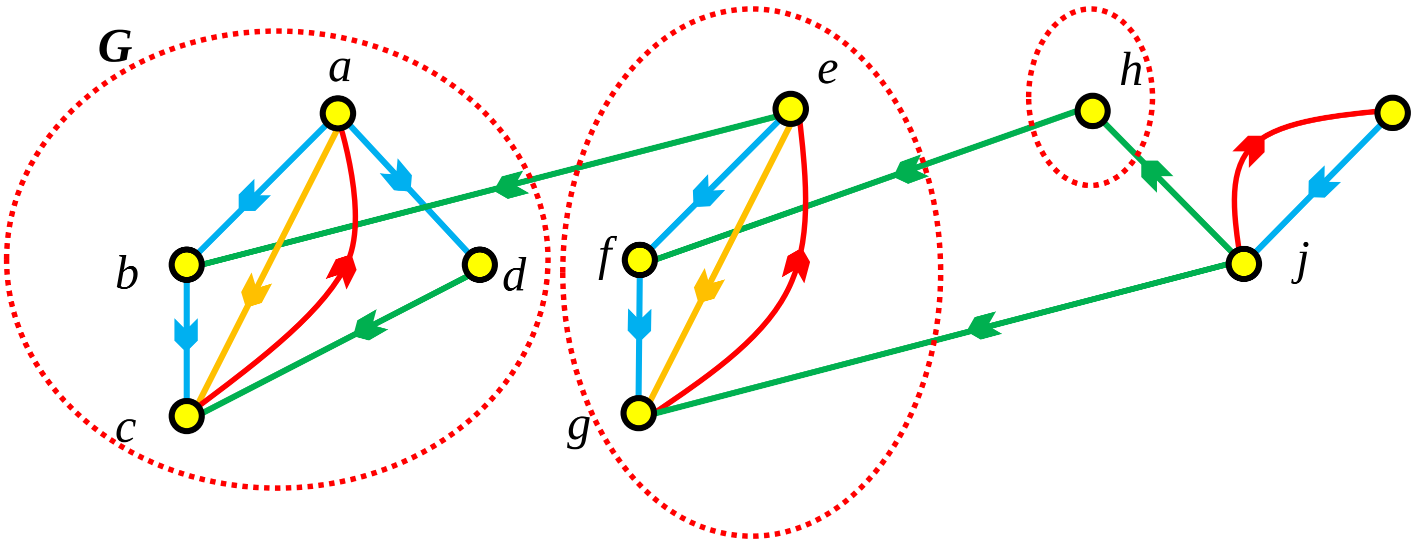
Busca em profundidade p/ digrafos



***Q* = { *i*, *j* }**

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	1	2	3	6	9	10	11	15	17	18
PS(*v*)	8	5	4	7	14	13	12	16	0	0
old(*v*)	1	1	1	3	9	9	9	15		

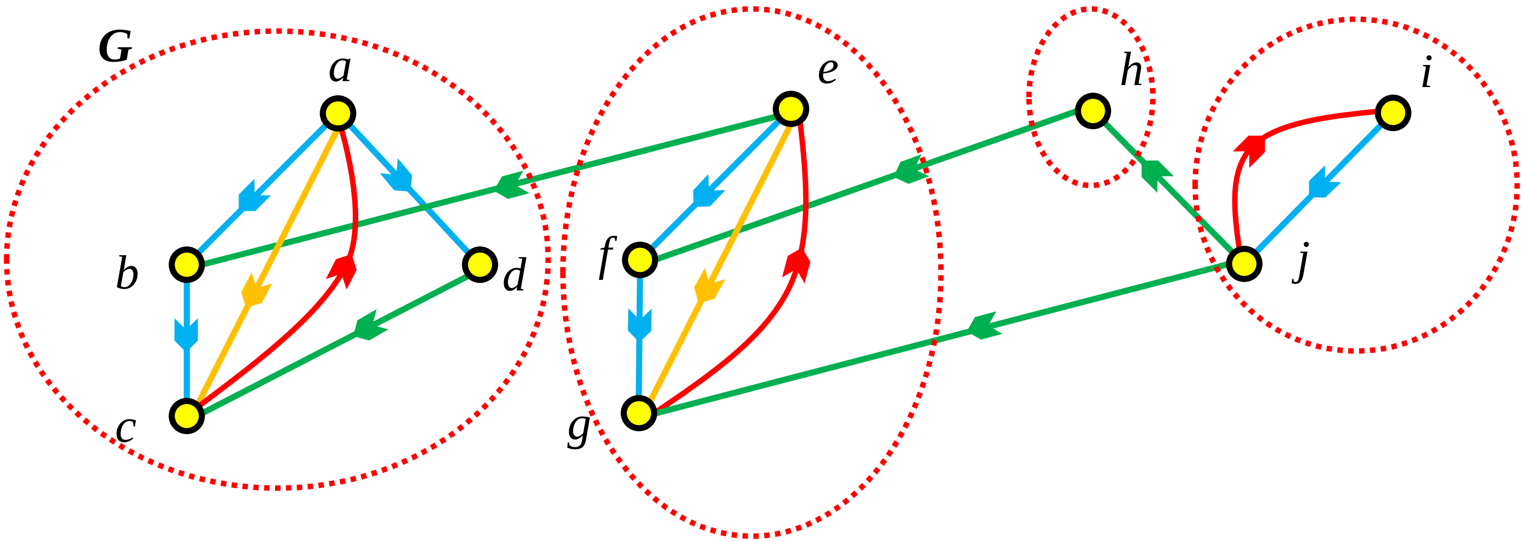
Busca em profundidade p/ digrafos



$$Q = \{\, i, j \,\}$$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	1	2	3	6	9	10	11	15	17	18
PS(*v*)	8	5	4	7	14	13	12	16	0	19
old(*v*)	1	1	1	3	9	9	9	15		17

Busca em profundidade p/ digrafos

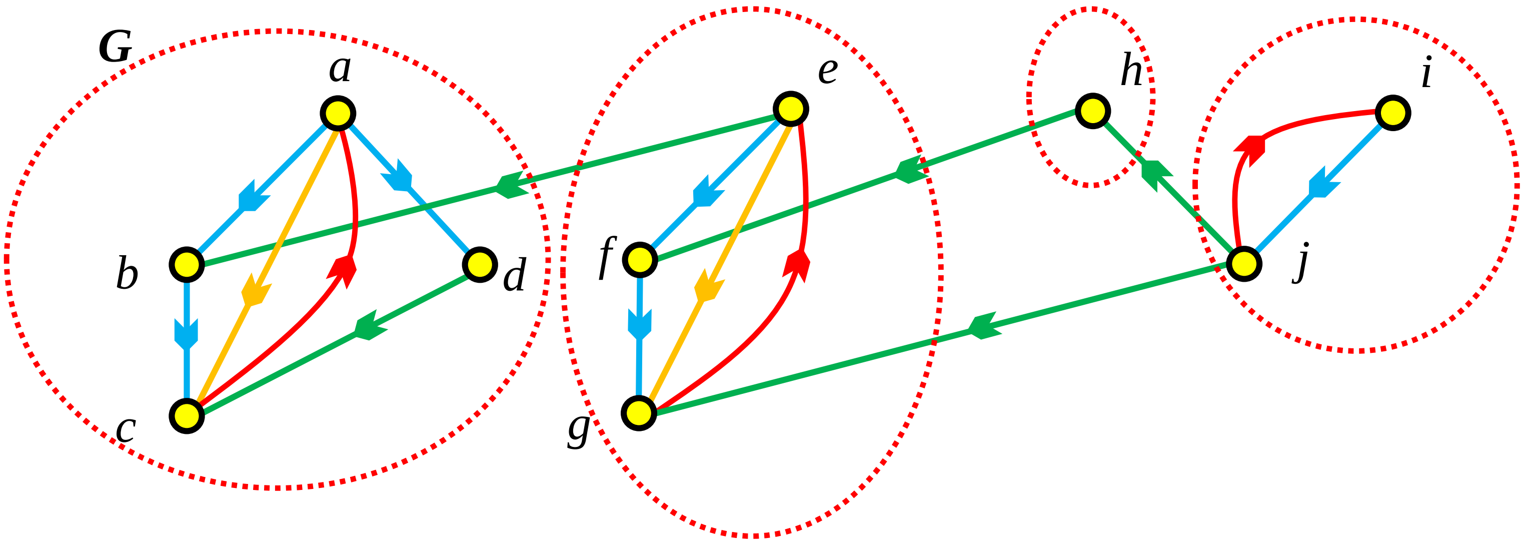


Vértice forte!

$Q = \{\ i, j\ \}$

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	9	10	11	15	17	18
PS(*v*)	8	5	4	7	14	13	12	16	20	19
old(*v*)	1	1	1	3	9	9	9	15	17	17

Busca em profundidade p/ digrafos



***Q* = Ø** **==> FIM!**

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	1	2	3	6	9	10	11	15	17	18
PS(*v*)	8	5	4	7	14	13	12	16	20	19
old(*v*)	1	1	1	3	9	9	9	15	17	17

Busca em profundidade p/ digrafos

Aplicação 1: Dado um digrafo *G*, determinar as componentes fortemente conexas de *G*.

Observações finais

- A determinação das cfc's obviamente produz sempre o mesmo resultado, independentemente da ordem de visita dos vértices/arestas escolhida para a busca!
- Uma aresta *vw* pode desempenhar um papel em uma busca e outro papel em outra. Exemplo: *vw* pode ser aresta de cruzamento em uma busca e ser de avanço em outra, e assim por diante.

Busca em profundidade p/ digrafos

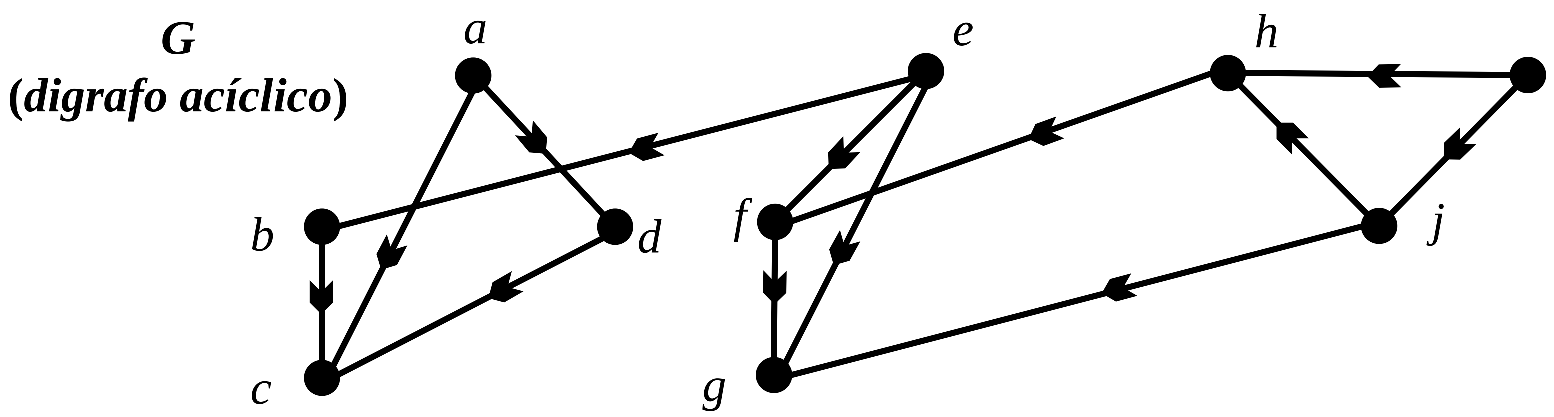
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.

- Um digrafo G é ***acíclico*** se G não contém ciclos direcionados.
- Uma ***ordenação topológica*** de um digrafo acíclico G é uma ordenação $v_1\ v_2\ v_3 \ldots v_{n-1}\ v_n$ dos vértices de G com a seguinte propriedade:

"se existe uma aresta em G de v_i a v_j então $i < j$"

Busca em profundidade p/ digrafos

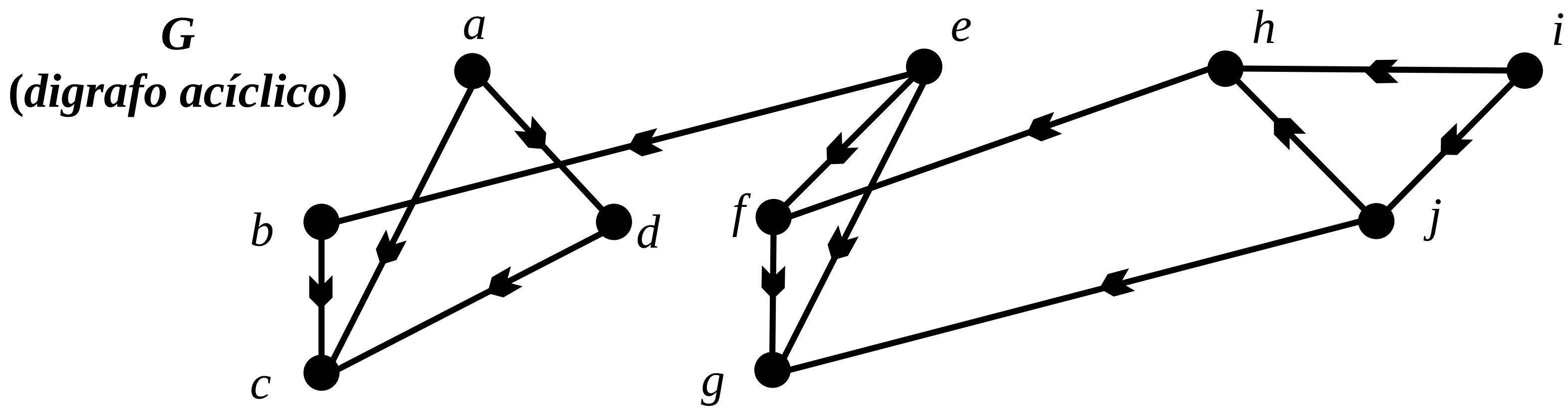
Aplicação 2: Dado um digrafo acíclico *G*, obter uma *ordenação topológica* de *G*.



***Uma ordenação topológica de G:* i j h e f g a b d c**

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



Uma ordenação topológica de G: ***i j h e f g a b d c***

Outra ordenação topológica de G: ***e i j h a f g d b c***

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.

Como decidir se um dado digrafo é acíclico?

Basta rodar uma busca em profundidade sobre G. Se esta busca não produzir nenhuma **aresta de retorno**, então G é acíclico.

Se uma busca sobre G não produz aresta de retorno, então nenhuma outra busca produzirá aresta de retorno!

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.

Aplicações da ordenação topológica

- **Escalonamento de atividades:** cada vértice representa uma atividade, e cada aresta direcionada de ***a*** para ***b*** indica que a atividade ***b*** só pode ser executada **depois** da atividade ***a***. A ordenação topológica fornece portanto uma sequência viável para a realização das atividades.

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.

Aplicações da ordenação topológica

- **Escalonamento de tarefas em um único processador:** cada vértice representa uma **tarefa**, e cada aresta direcionada de ***a*** para ***b*** indica que a tarefa ***a*** deve enviar um dado para a tarefa ***b*** (a tarefa ***b*** só pode iniciar sua execução depois que a tarefa ***a*** enviar o dado). Dado que existe um **único processador** para executar as tarefas, a ordenação topológica fornece uma sequência de execução das tarefas no processador.

Busca em profundidade p/ digrafos

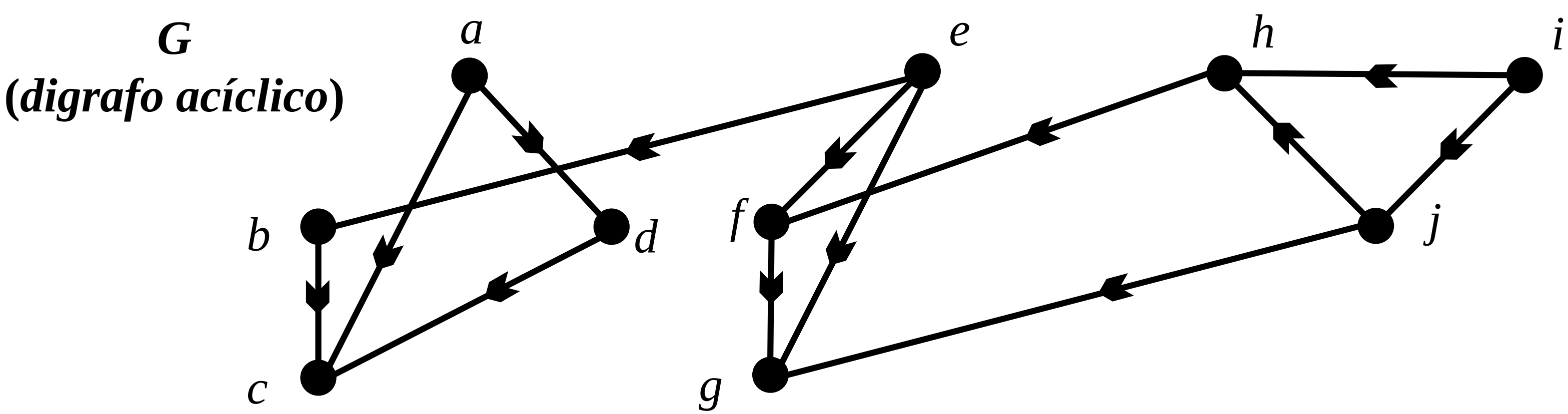
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.

Algoritmo para ordenação topológica de um digrafo G :

1. Rodar uma busca em profundidade sobre G. Se esta busca produziu alguma aresta de retorno, então pare: G não é acíclico. Caso contrário, vá para o passo 2.
2. Ordene os valores das **profundidades de saída** geradas pela busca em **ordem decrescente**.
3. Faça v_1 como o vértice de maior *PS*, v_2 como o vértice de segunda maior *PS*, e assim por diante.
4. A ordenação $v_1\ v_2\ v_3\ \ldots\ v_{n-1}\ v_n$ obtida no passo 3 é uma ordenação topológica!

Busca em profundidade p/ digrafos

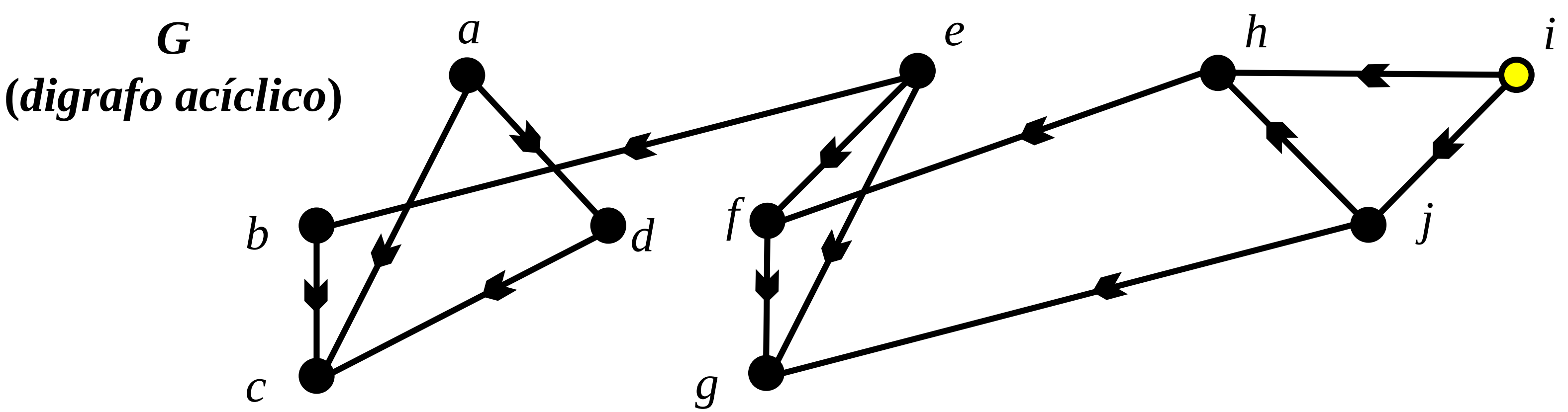
Aplicação 2: Dado um digrafo acíclico *G*, obter uma *ordenação topológica* de *G*.



v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
PE(*v*)	0	0	0	0	0	0	0	0	0	0
PS(*v*)	0	0	0	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos

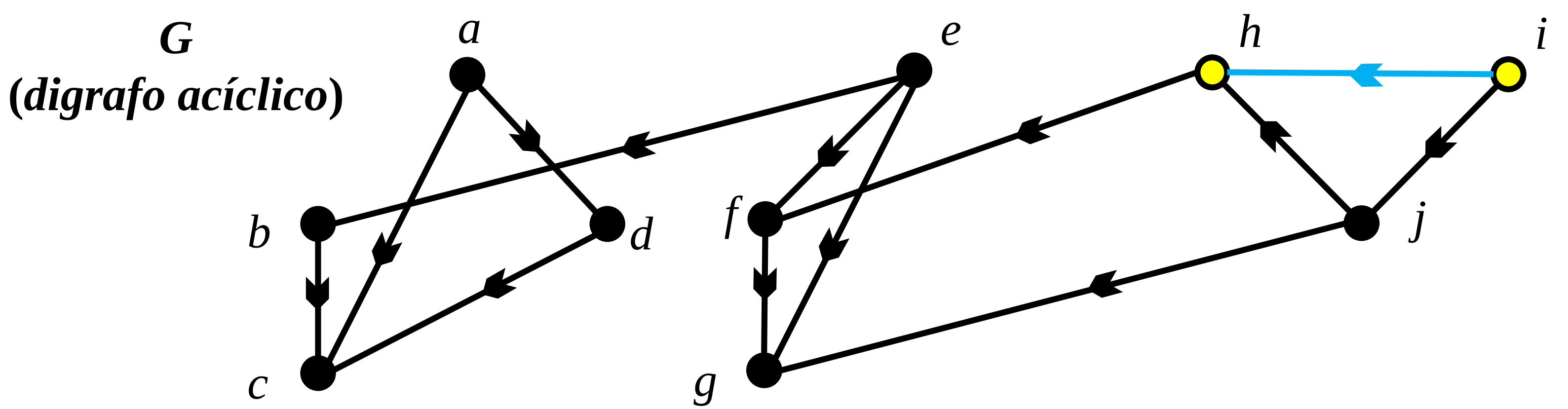
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
$PE(v)$	0	0	0	0	0	0	0	0	1	0
$PS(v)$	0	0	0	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos

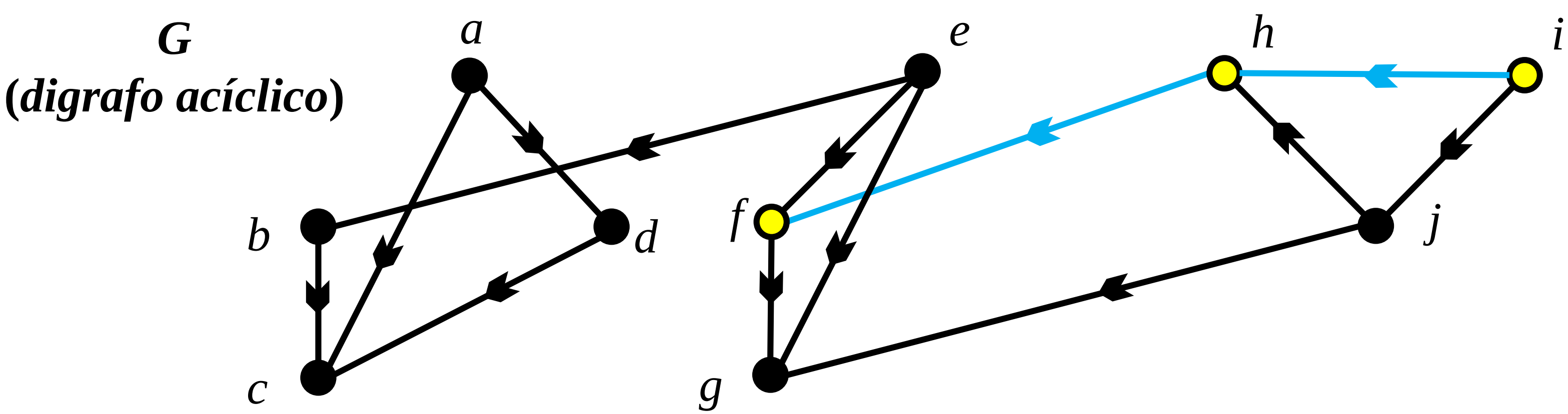
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	0	0	0	0	0	0	2	1	0
$PS(v)$	0	0	0	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos

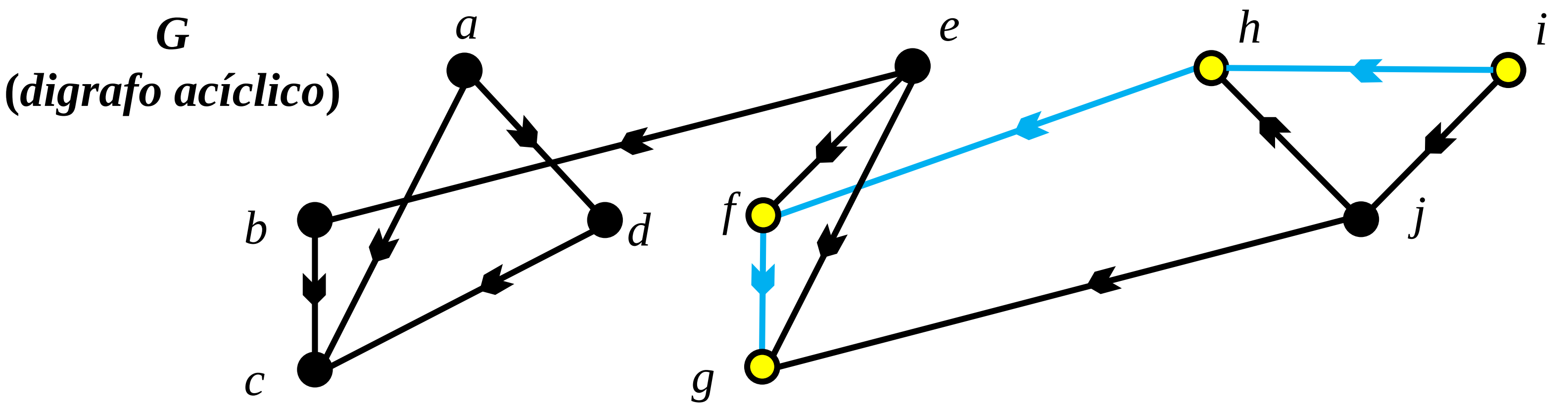
Aplicação 2: Dado um digrafo acíclico *G*, obter uma *ordenação topológica* de *G*.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	0	0	0	0	3	0	2	1	0
$PS(v)$	0	0	0	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos

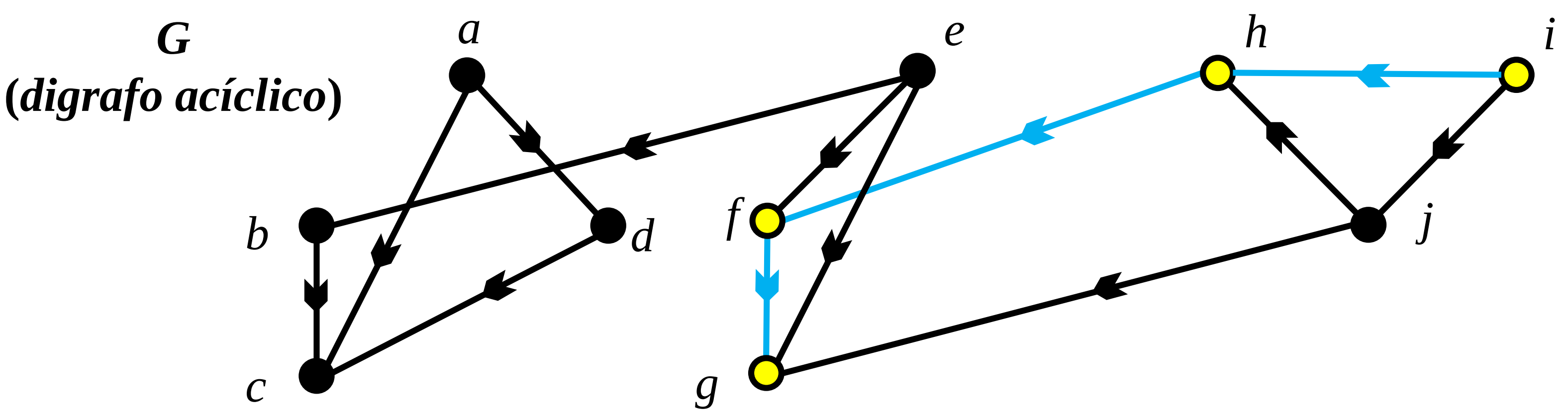
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	0	0	0	0	3	4	2	1	0
$PS(v)$	0	0	0	0	0	0	0	0	0	0

Busca em profundidade p/ digrafos

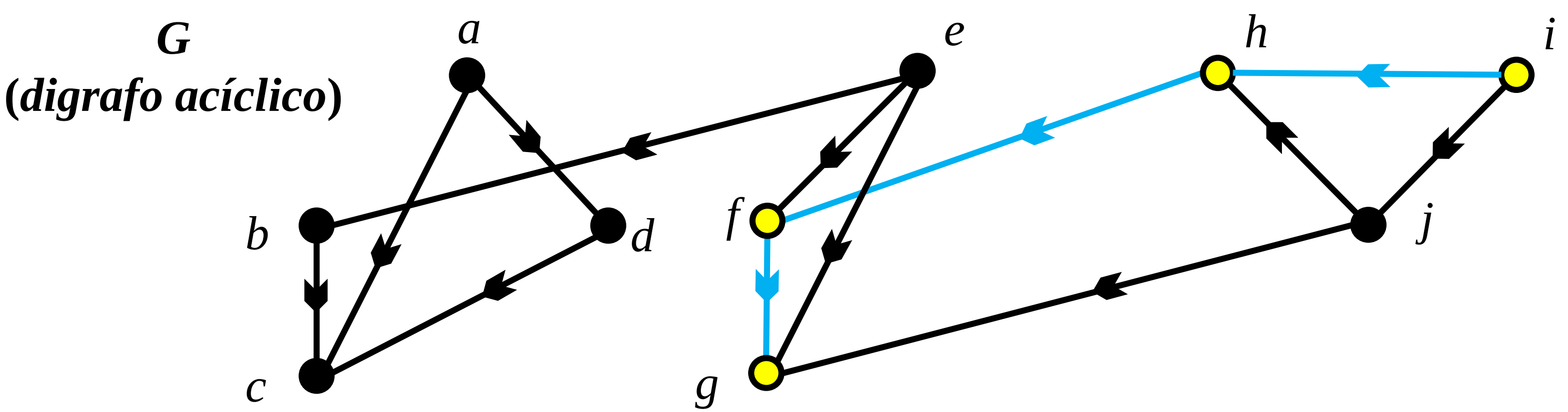
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	0	0	0	0	3	4	2	1	0
$PS(v)$	0	0	0	0	0	0	5	0	0	0

Busca em profundidade p/ digrafos

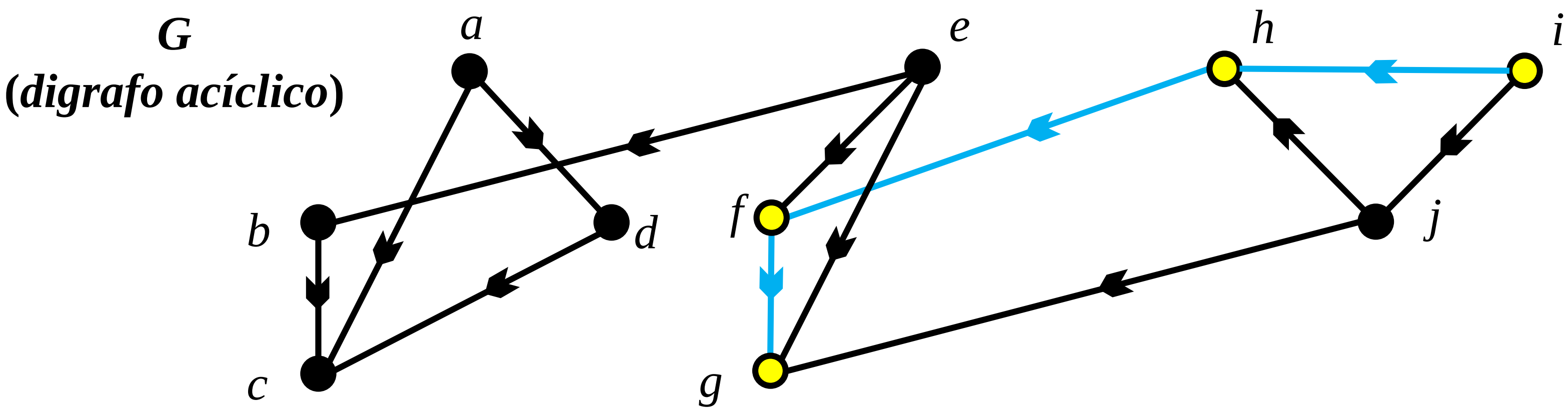
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
$PE(v)$	0	0	0	0	0	3	4	2	1	0
$PS(v)$	0	0	0	0	0	6	5	0	0	0

Busca em profundidade p/ digrafos

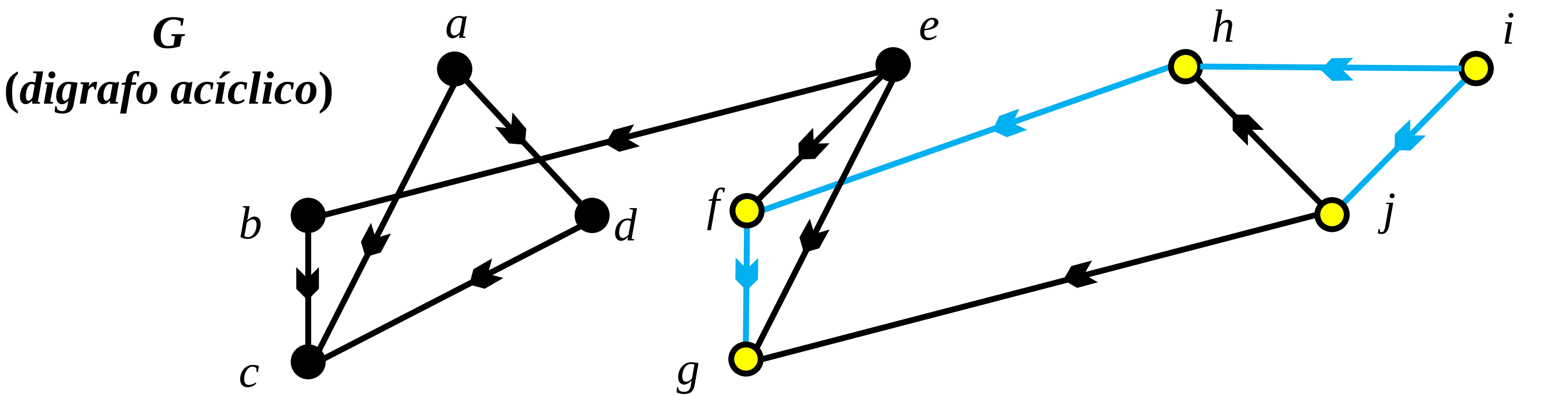
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	0	0	0	0	3	4	2	1	0
$PS(v)$	0	0	0	0	0	6	5	7	0	0

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.

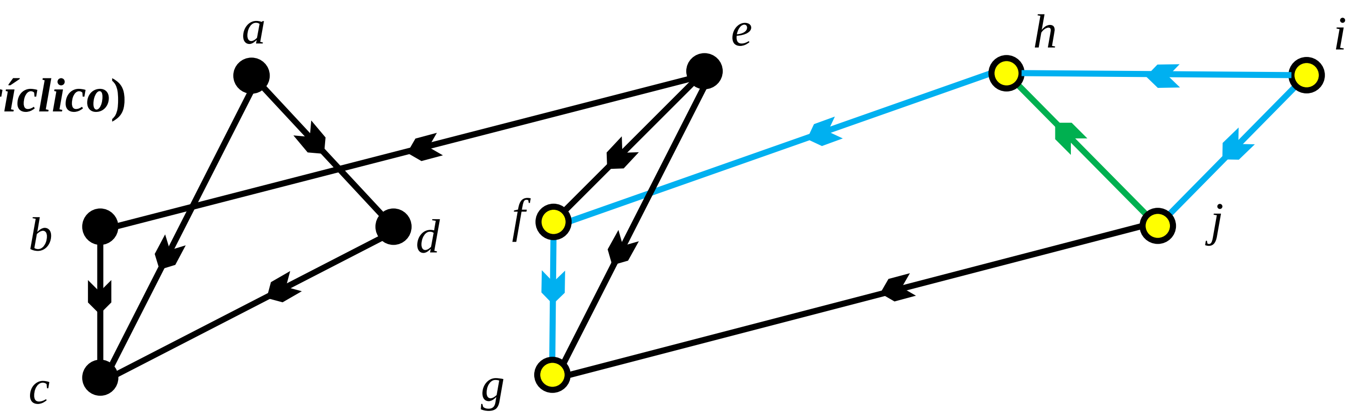


v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*
$PE(v)$	0	0	0	0	0	3	4	2	1	8
$PS(v)$	0	0	0	0	0	6	5	7	0	0

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.

G
(digrafo acíclico)

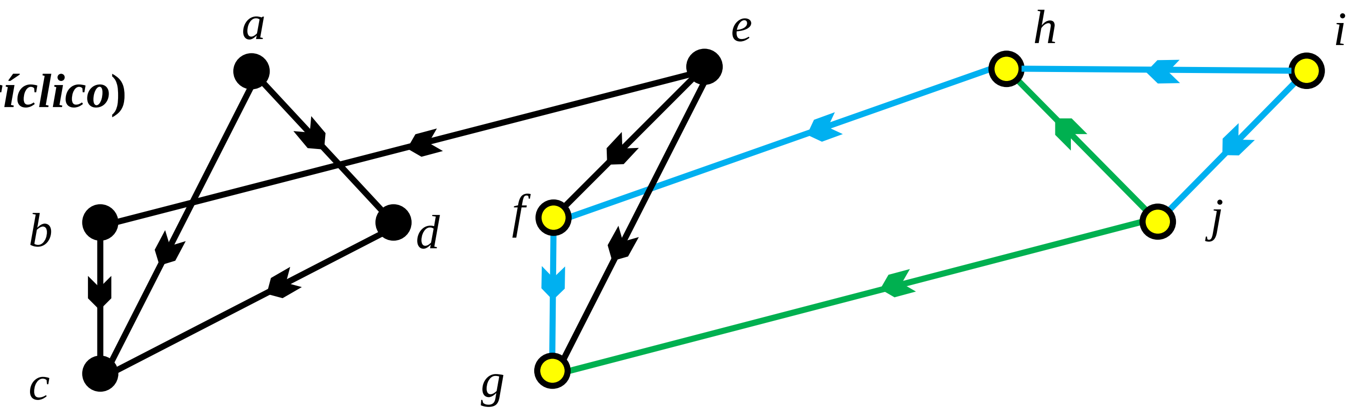


v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
$PE(v)$	0	0	0	0	0	3	4	2	1	8
$PS(v)$	0	0	0	0	0	6	5	7	0	0

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.

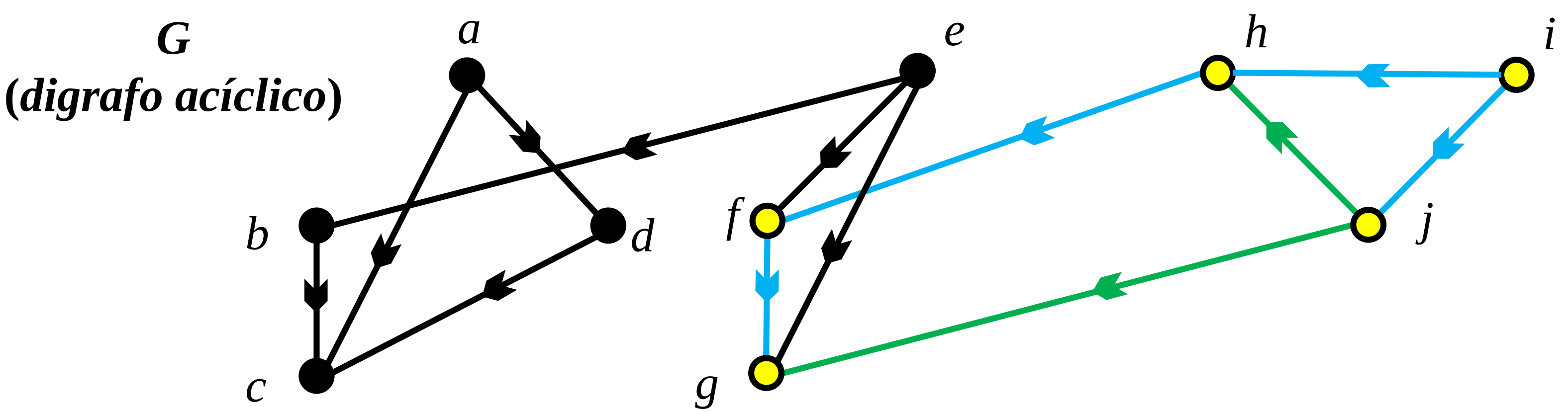
G
(digrafo acíclico)



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	0	0	0	0	3	4	2	1	8
$PS(v)$	0	0	0	0	0	6	5	7	0	0

Busca em profundidade p/ digrafos

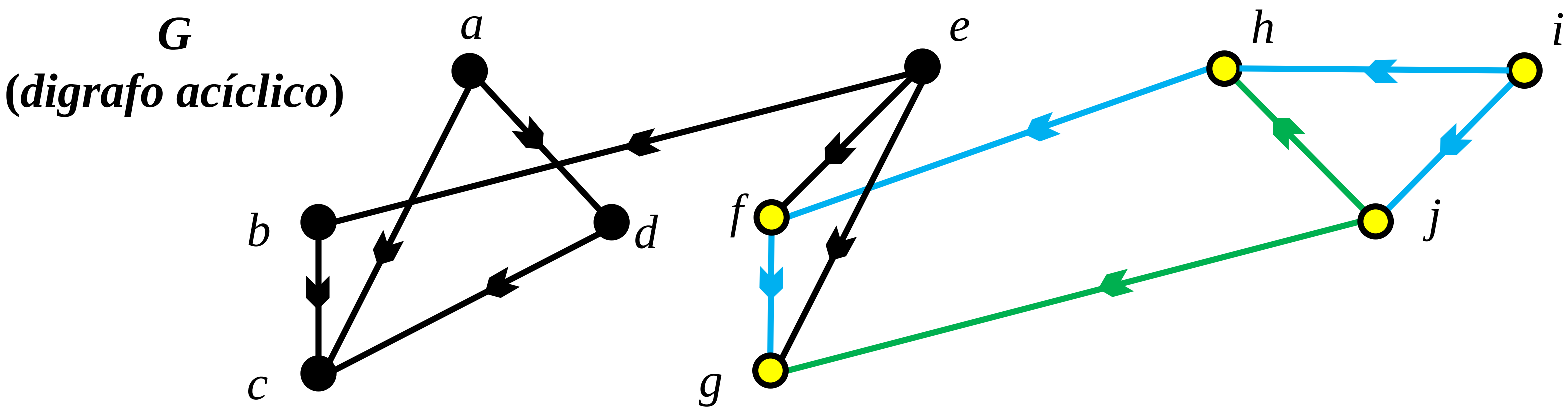
Aplicação 2: Dado um digrafo acíclico *G*, obter uma *ordenação topológica* de *G*.



v	a	b	c	d	e	f	g	h	i	j
PE(*v*)	0	0	0	0	0	3	4	2	1	8
PS(*v*)	0	0	0	0	0	6	5	7	0	9

Busca em profundidade p/ digrafos

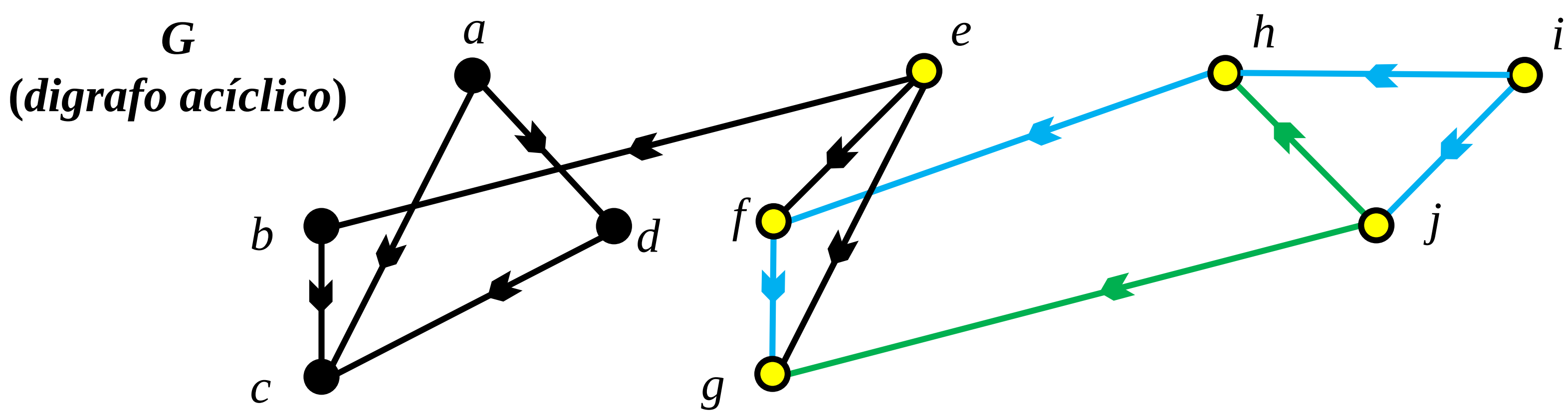
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	0	0	0	0	3	4	2	1	8
$PS(v)$	0	0	0	0	0	6	5	7	10	9

Busca em profundidade p/ digrafos

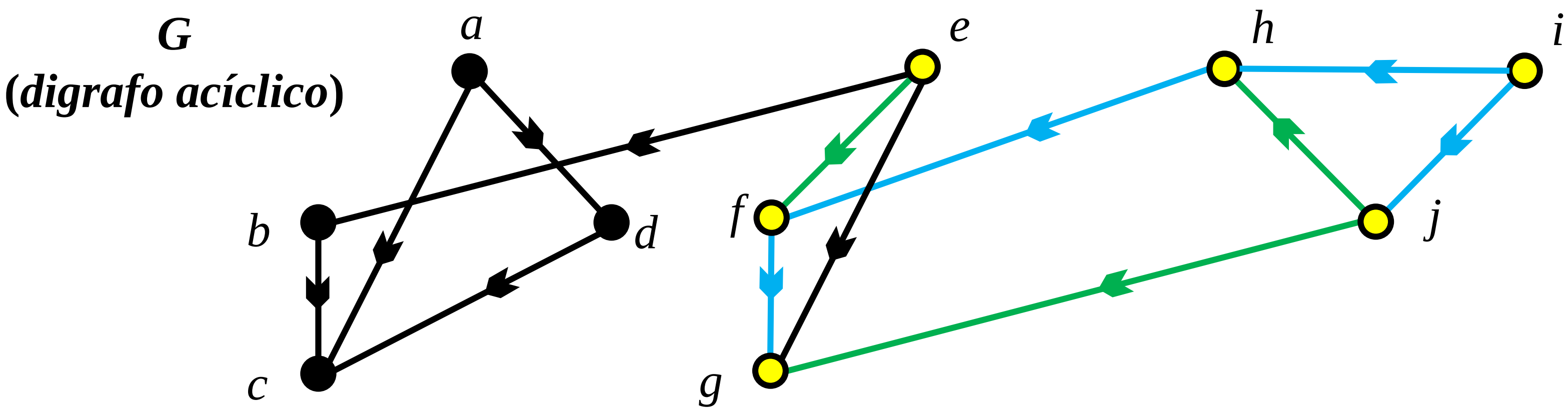
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	0	0	0	11	3	4	2	1	8
$PS(v)$	0	0	0	0	0	6	5	7	10	9

Busca em profundidade p/ digrafos

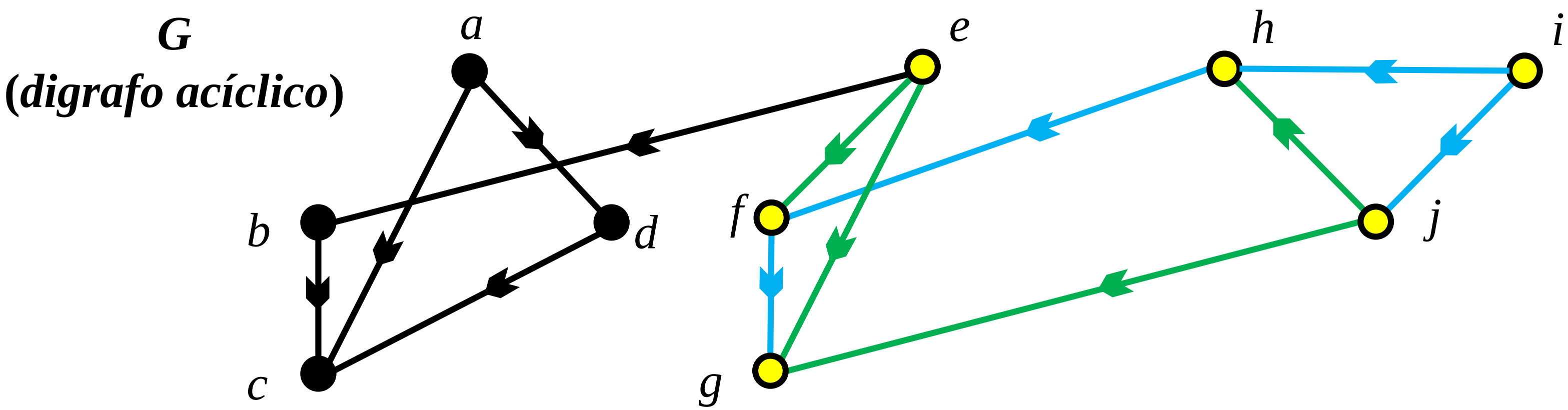
Aplicação 2: Dado um digrafo acíclico *G*, obter uma *ordenação topológica* de *G*.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	0	0	0	11	3	4	2	1	8
$PS(v)$	0	0	0	0	0	6	5	7	10	9

Busca em profundidade p/ digrafos

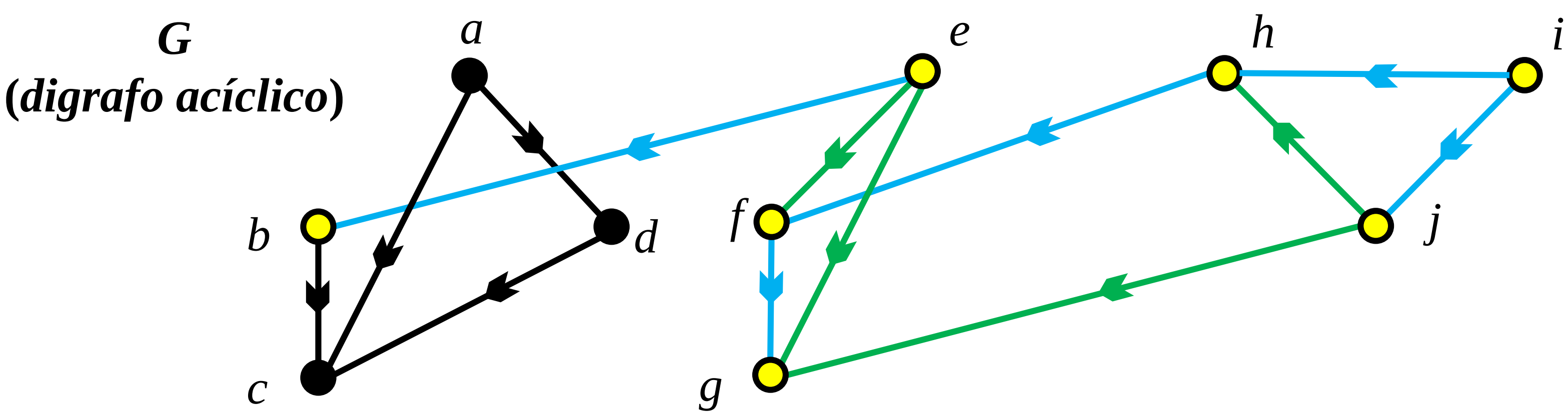
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	0	0	0	11	3	4	2	1	8
$PS(v)$	0	0	0	0	0	6	5	7	10	9

Busca em profundidade p/ digrafos

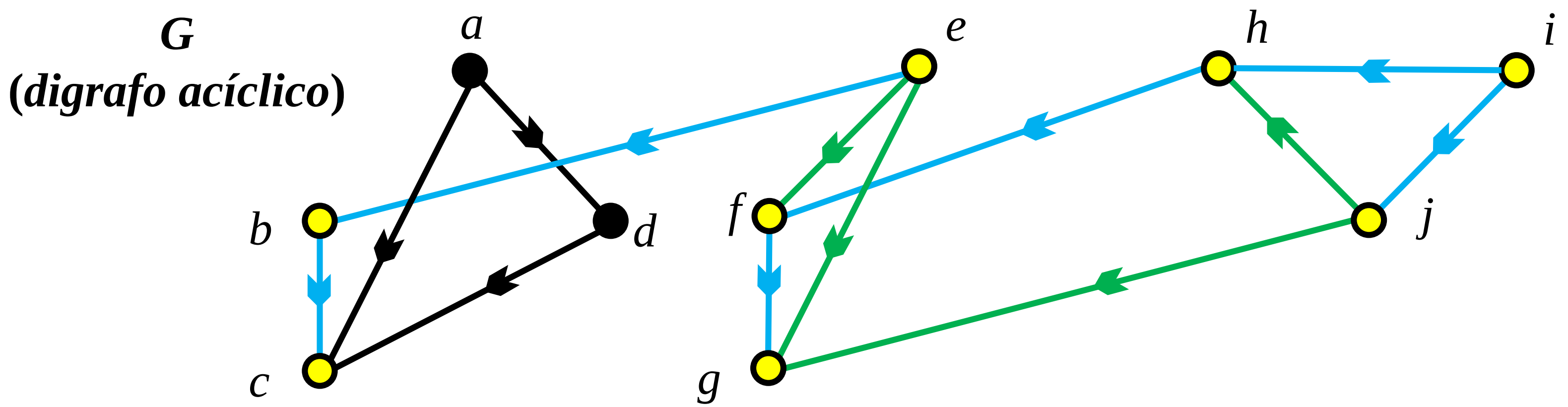
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	12	0	0	11	3	4	2	1	8
$PS(v)$	0	0	0	0	0	6	5	7	10	9

Busca em profundidade p/ digrafos

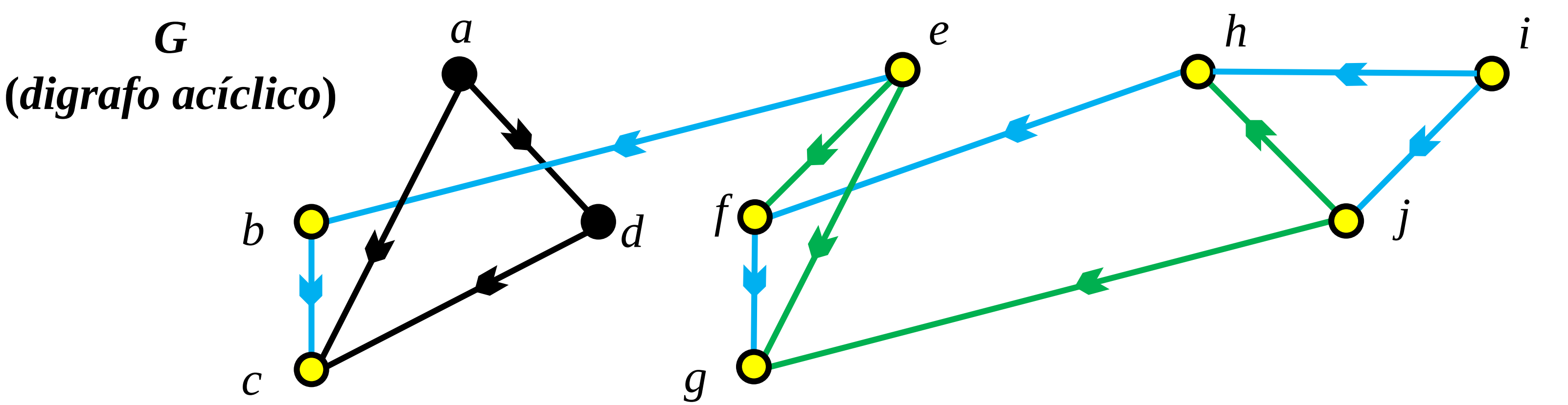
Aplicação 2: Dado um digrafo acíclico *G*, obter uma *ordenação topológica* de *G*.



v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
PE(*v*)	0	12	13	0	11	3	4	2	1	8
PS(*v*)	0	0	0	0	0	6	5	7	10	9

Busca em profundidade p/ digrafos

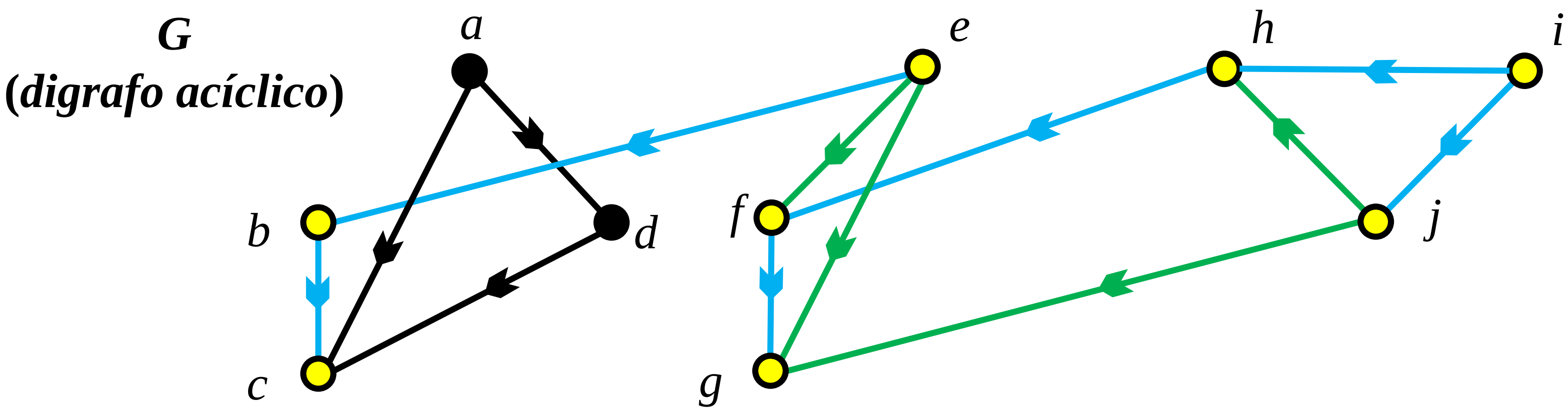
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	12	13	0	11	3	4	2	1	8
$PS(v)$	0	0	14	0	0	6	5	7	10	9

Busca em profundidade p/ digrafos

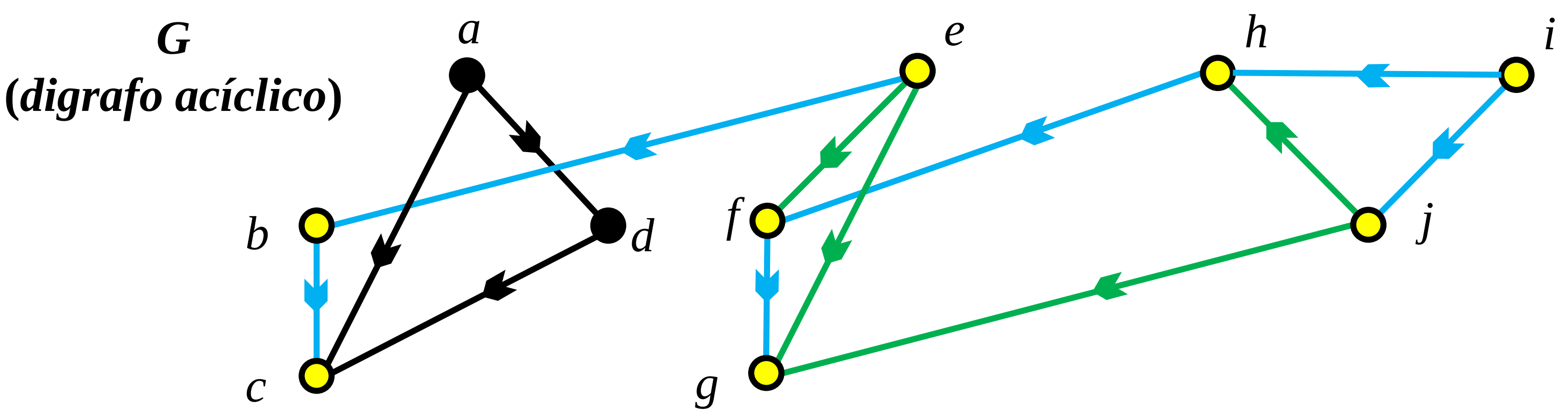
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	12	13	0	11	3	4	2	1	8
$PS(v)$	0	15	14	0	0	6	5	7	10	9

Busca em profundidade p/ digrafos

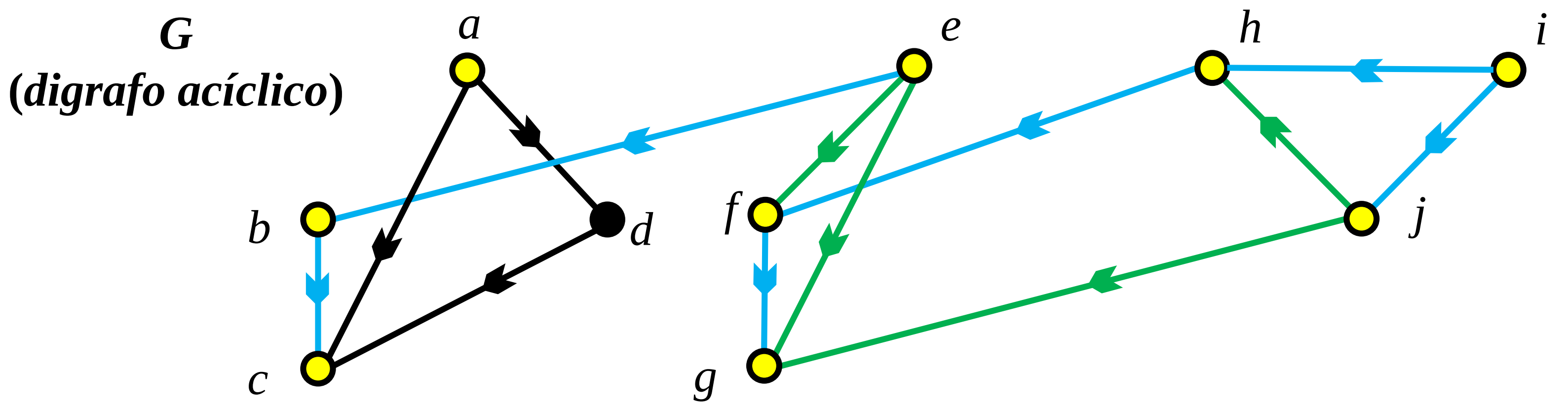
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	0	12	13	0	11	3	4	2	1	8
$PS(v)$	0	15	14	0	16	6	5	7	10	9

Busca em profundidade p/ digrafos

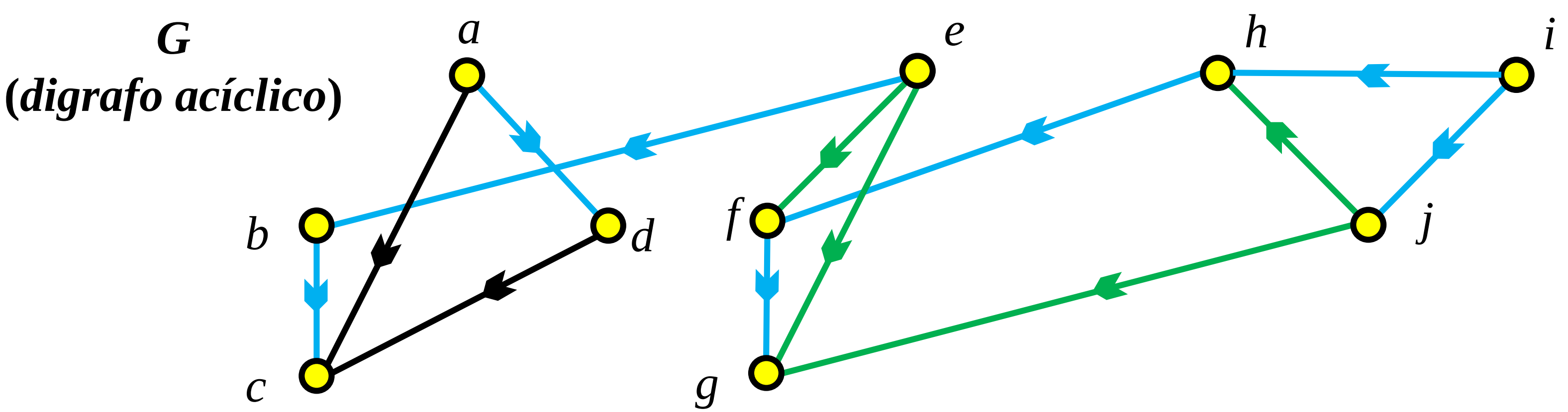
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	17	12	13	0	11	3	4	2	1	8
$PS(v)$	0	15	14	0	16	6	5	7	10	9

Busca em profundidade p/ digrafos

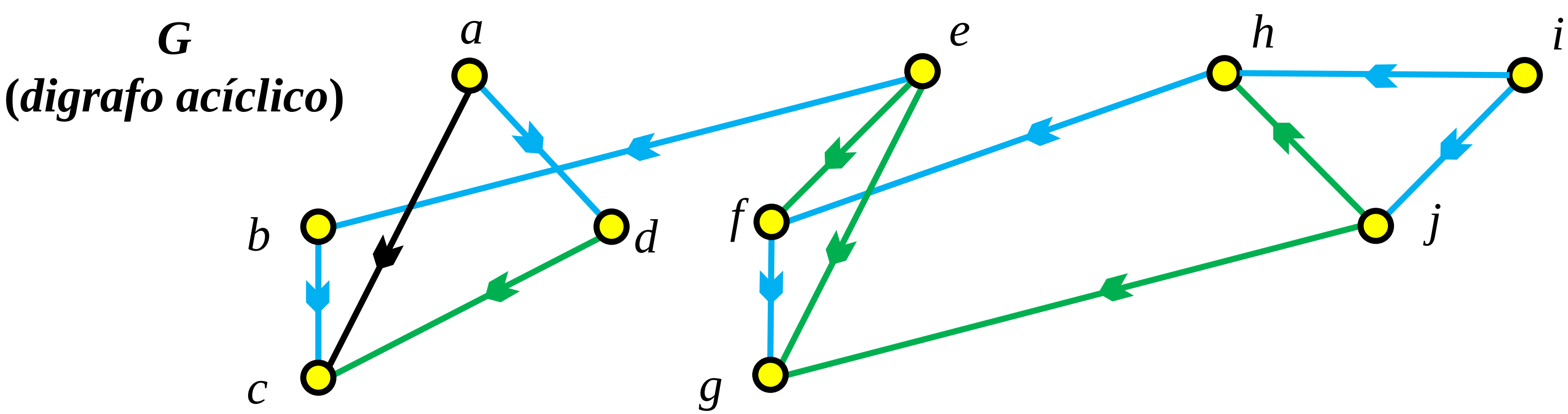
Aplicação 2: Dado um digrafo acíclico *G*, obter uma *ordenação topológica* de *G*.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	17	12	13	18	11	3	4	2	1	8
$PS(v)$	0	15	14	0	16	6	5	7	10	9

Busca em profundidade p/ digrafos

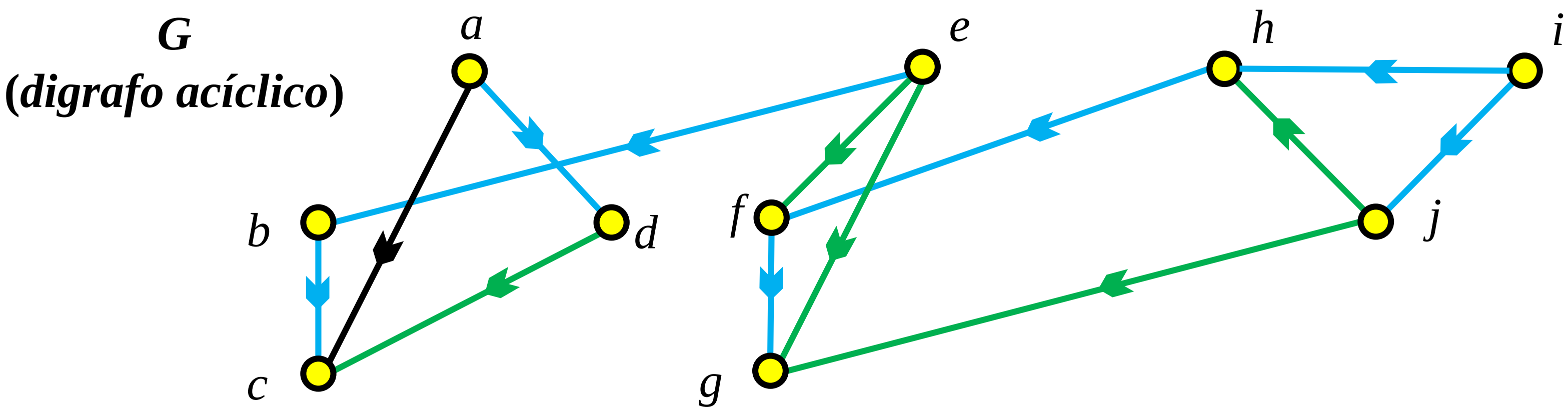
Aplicação 2: Dado um digrafo acíclico *G*, obter uma *ordenação topológica* de *G*.



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	17	12	13	18	11	3	4	2	1	8
$PS(v)$	0	15	14	0	16	6	5	7	10	9

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.

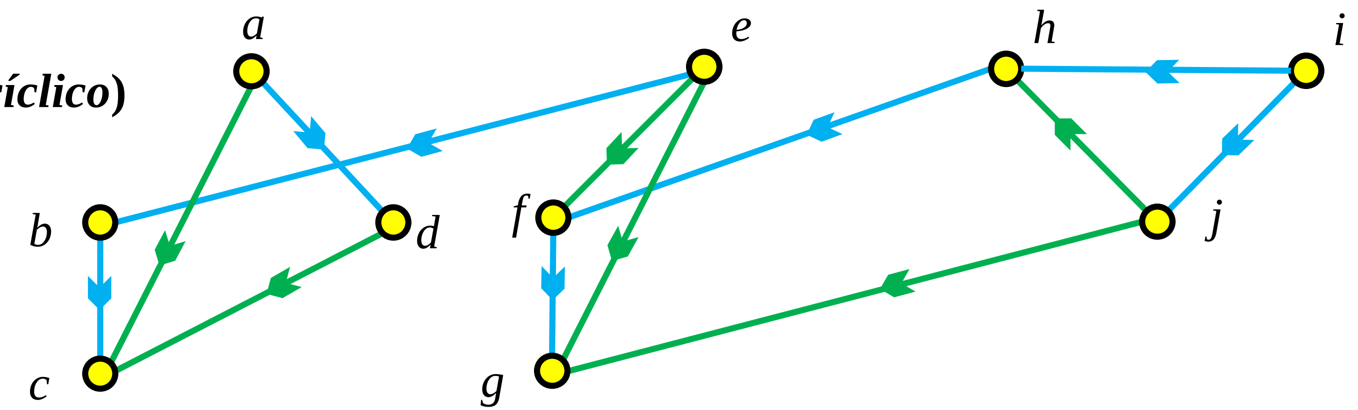


v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	17	12	13	18	11	3	4	2	1	8
$PS(v)$	0	15	14	19	16	6	5	7	10	9

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.

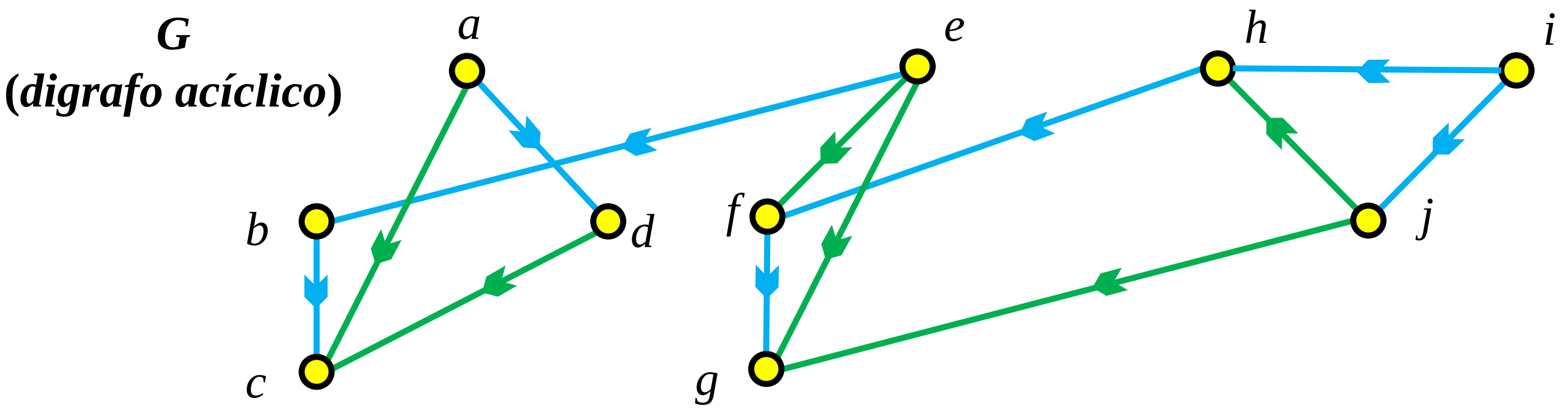
G (digrafo acíclico)



v	a	b	c	d	e	f	g	h	i	j
$PE(v)$	17	12	13	18	11	3	4	2	1	8
$PS(v)$	0	15	14	19	16	6	5	7	10	9

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico *G*, obter uma *ordenação topológica* de *G*.

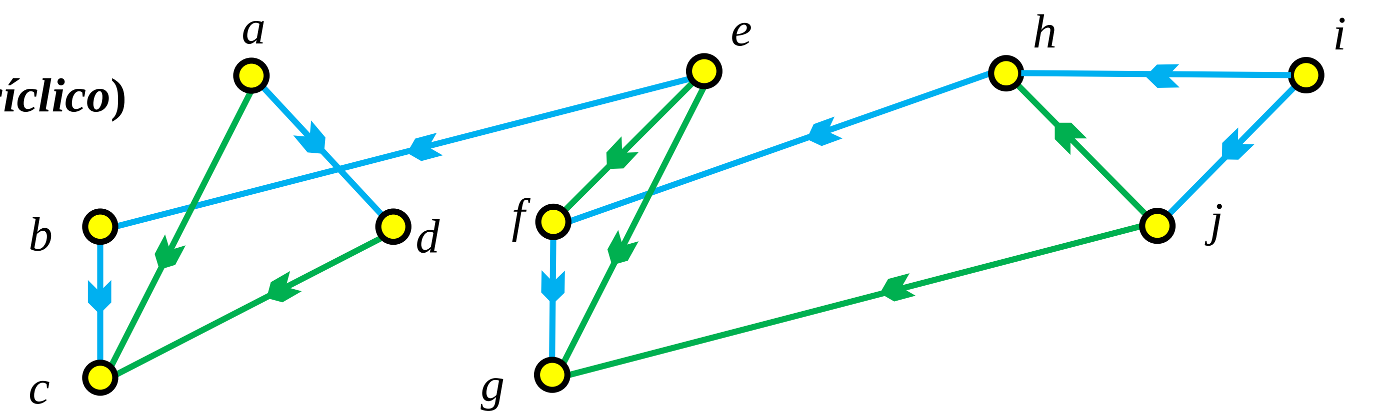


v	a	b	c	d	e	f	g	h	i	j
PE(*v*)	17	12	13	18	11	3	4	2	1	8
PS(*v*)	20	15	14	19	16	6	5	7	10	9

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.

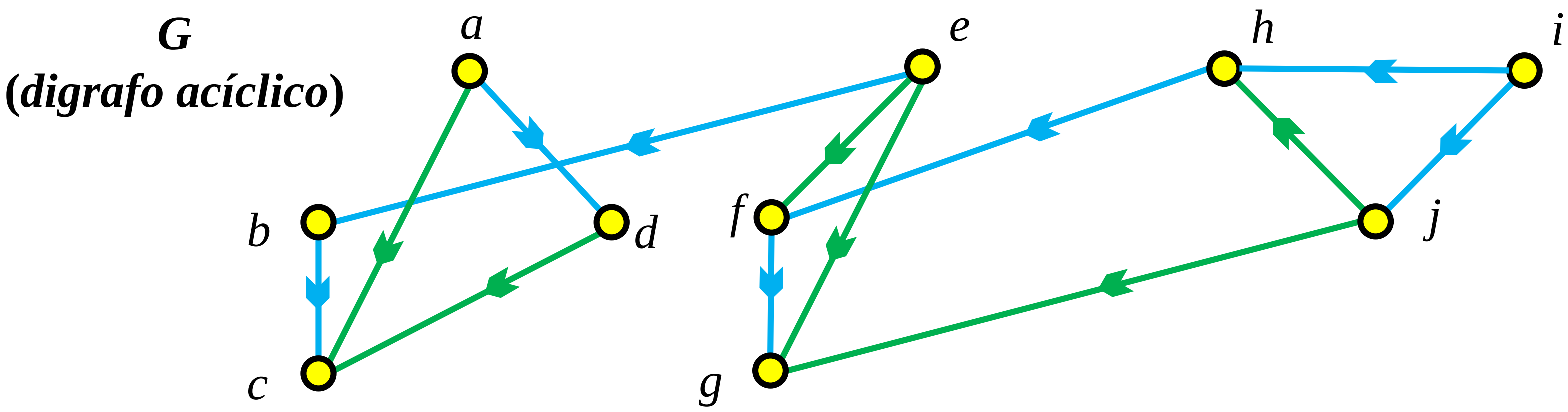
G
(*digrafo acíclico*)



v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***	***i***	***j***
$PS(v)$	20	15	14	19	16	6	5	7	10	9

Busca em profundidade p/ digrafos

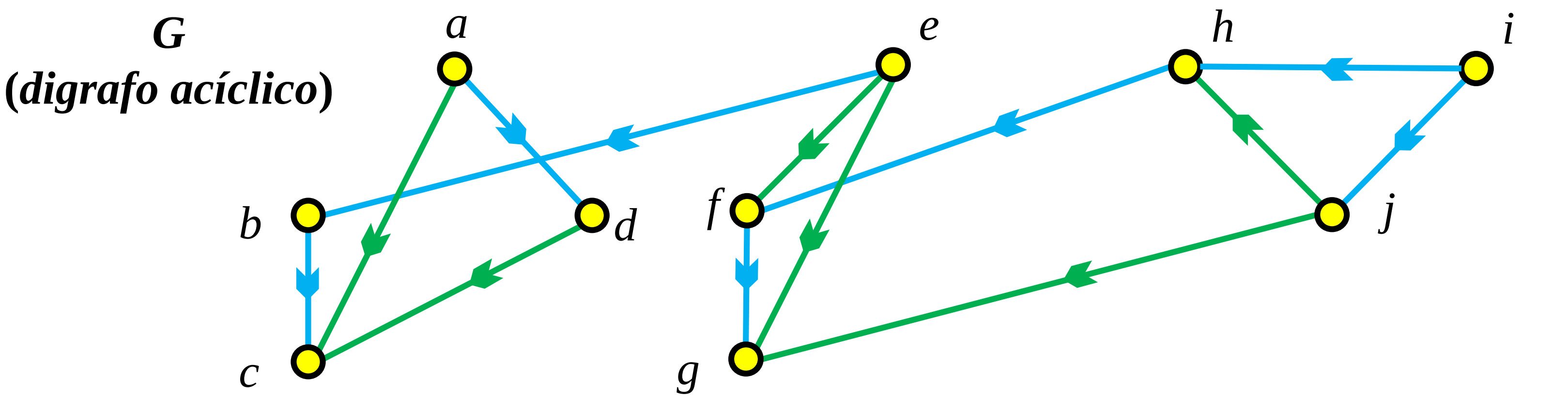
Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



v	a	d	e	b	c	i	j	h	f	g
$PS(v)$	20	19	16	15	14	10	9	7	6	5

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.

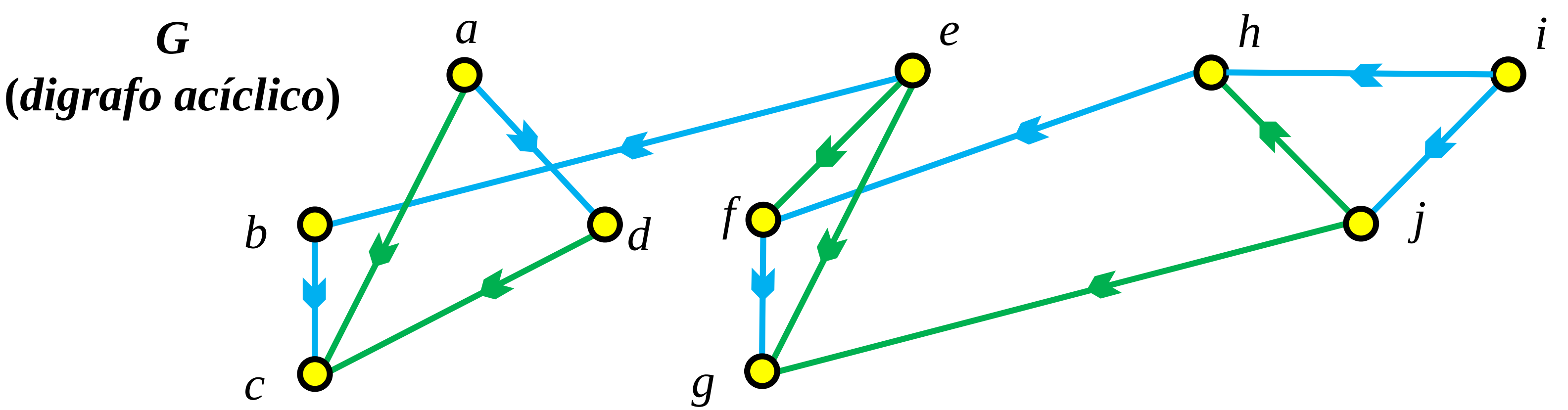


Ordenação topológica

a	***d***	***e***	***b***	***c***	***i***	***j***	***h***	***f***	***g***
v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico *G*, obter uma *ordenação topológica* de *G*.



Ordenação topológica

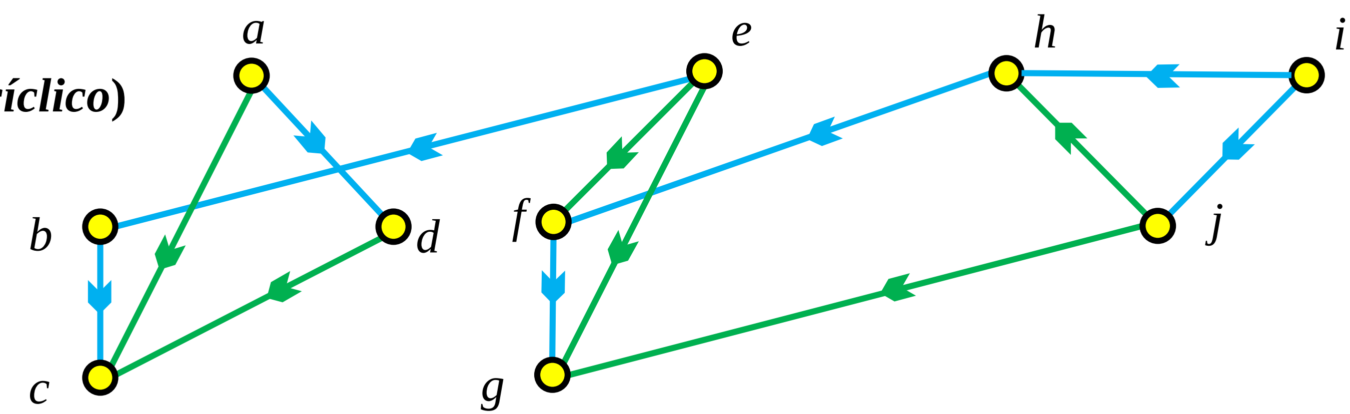
a	*d*	*e*	*b*	*c*	*i*	*j*	*h*	*f*	*g*

Por que este algoritmo funciona corretamente?

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico *G*, obter uma *ordenação topológica* de *G*.

G
(*digrafo acíclico*)



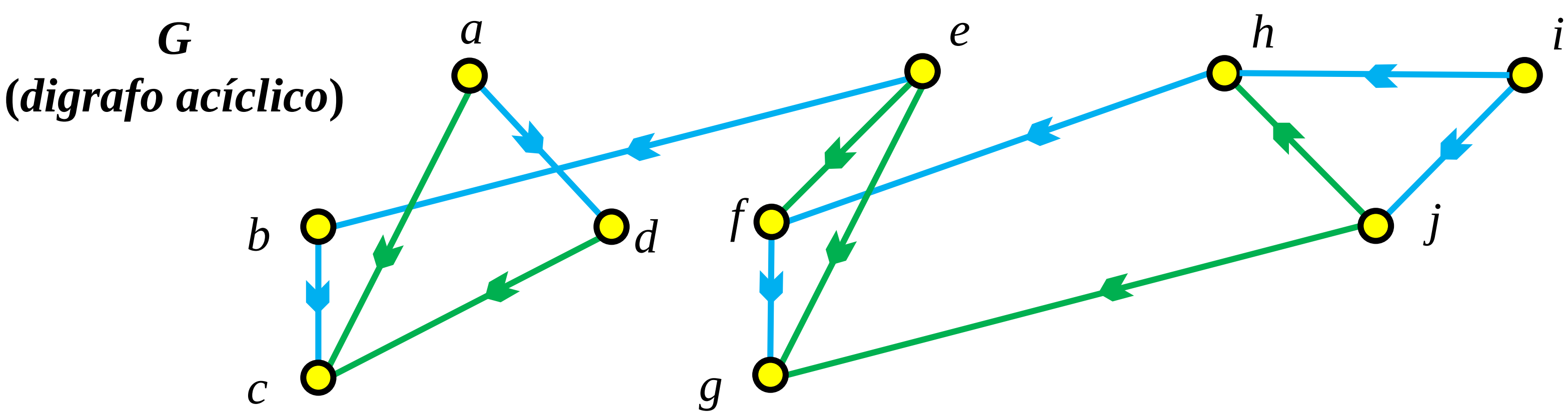
Ordenação topológica

a	***d***	***e***	***b***	***c***	***i***	***j***	***h***	***f***	***g***

Note que o vértice com a menor PS é um sumidouro
(vértice com grau de saída igual a zero)

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico *G*, obter uma *ordenação topológica* de *G*.



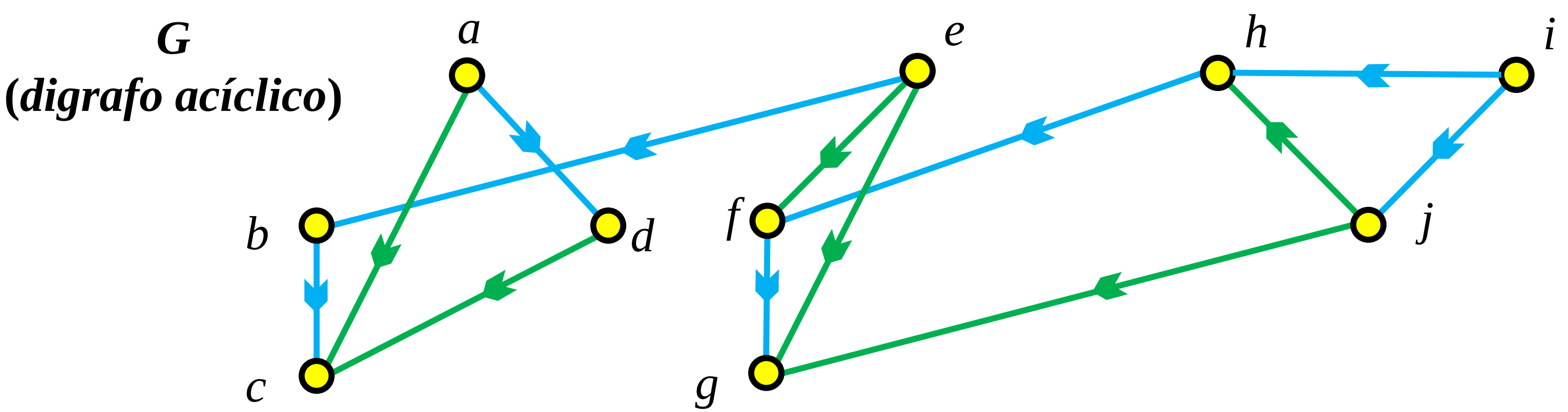
Ordenação topológica

a	*d*	*e*	*b*	*c*	*i*	*j*	*h*	*f*	*g*

Portanto, a posição deste vértice está correta na ordenação topológica, pois dele não partem arestas

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico *G*, obter uma *ordenação topológica* de *G*.



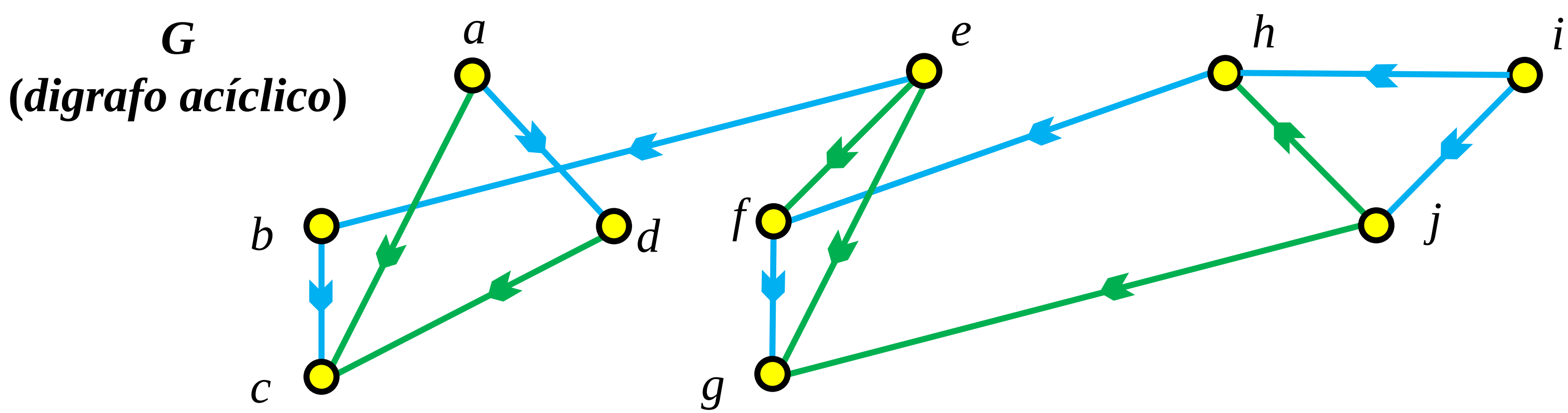
Ordenação topológica

a	***d***	***e***	***b***	***c***	***i***	***j***	***h***	***f***	***g***

A posição de f também está correta, pois f é um sumidouro no digrafo G – g

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.



Ordenação topológica

a	***d***	***e***	***b***	***c***	***i***	***j***	***h***	***f***	***g***
v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}

Aplicando sucessivamente este raciocínio, cada vértice v_i é um sumidouro no digrafo $G - \{v_{i+1}, v_{i+2}, \ldots ,v_n \}$

Busca em profundidade p/ digrafos

Aplicação 2: Dado um digrafo acíclico G, obter uma *ordenação topológica* de G.

Complexidade do algoritmo para ordenação topológica de um digrafo *G*.

- Rodar uma busca em profundidade sobre G tem complexidade $\boldsymbol{O(n + m)}$.
- Cada vez que um vértice sai da busca (recebe uma *PS* diferente de zero), é colocado imediatamente à esquerda da ordenação topológica sendo construída. Isto dispensa o passo de ordenação das *PS'*s.
- Portanto, a complexidade final é $\boldsymbol{O(n + m)}$.

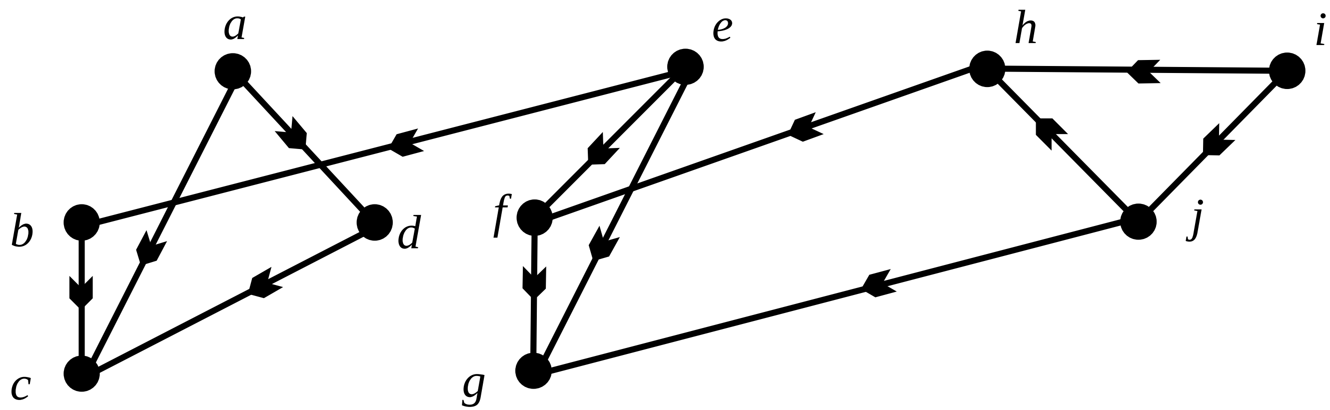
Busca em profundidade p/ digrafos

Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.

- **Idéia: todo digrafo acíclico possui um sumidouro!** (para provar este fato, basta ver que qualquer caminho orientado de comprimento máximo em um digrafo acíclico termina necessariamente em um sumidouro)
- Realizar a seguinte iteração, até que não haja mais vértices: **localizar um sumidouro, retirá-lo do digrafo e colocá-lo à esquerda na ordenação sendo construída.**
- **É possível implementar a idéia acima em tempo linear ? (isto é, $O(n + m)$?)**

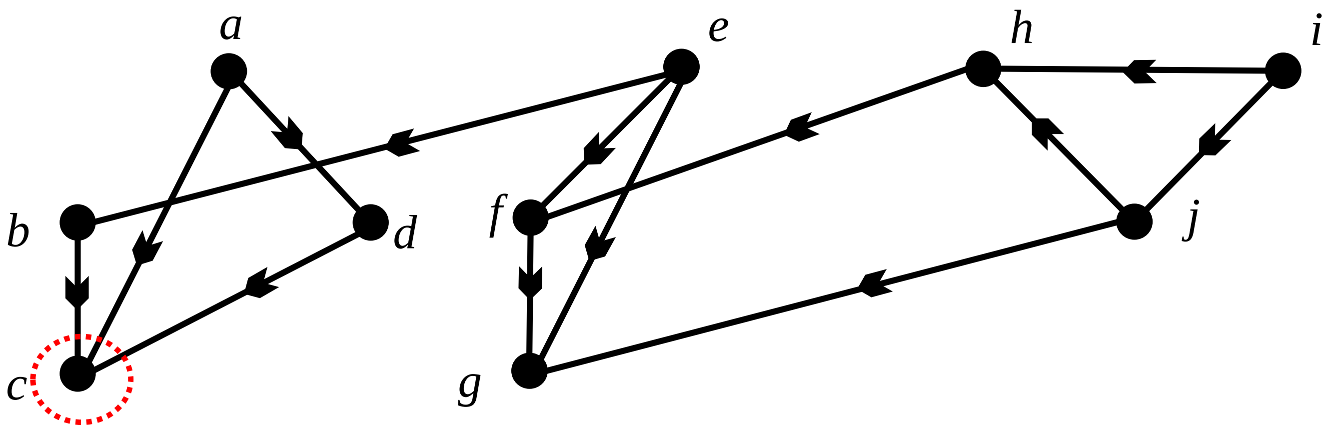
Busca em profundidade p/ digrafos

Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



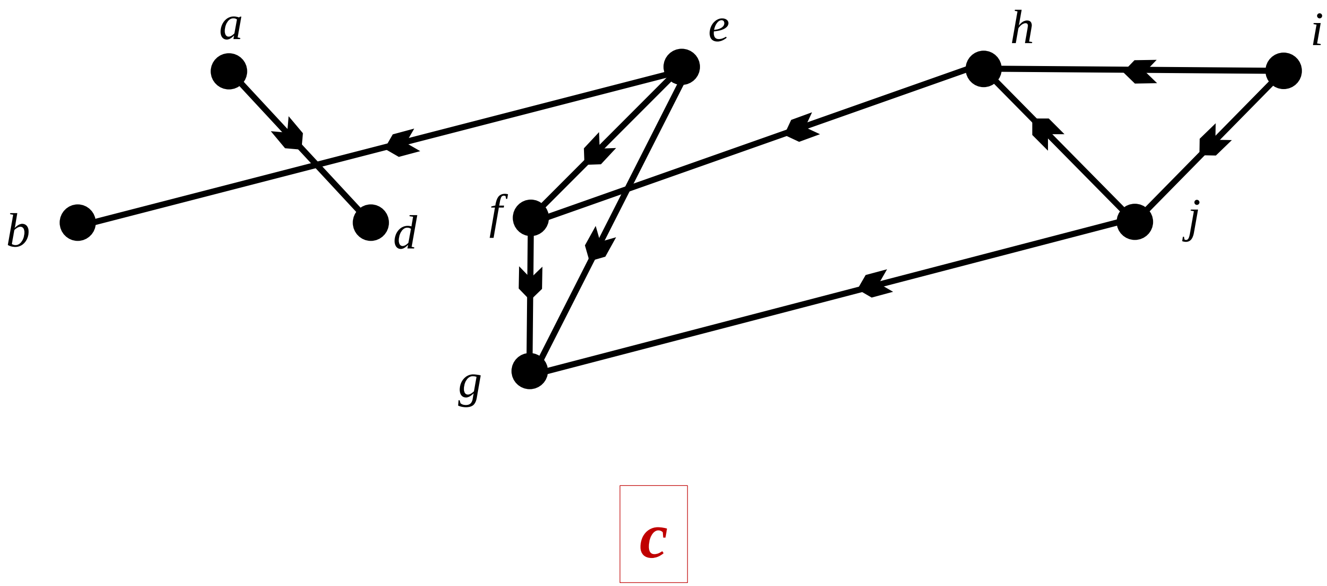
Busca em profundidade p/ digrafos

Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



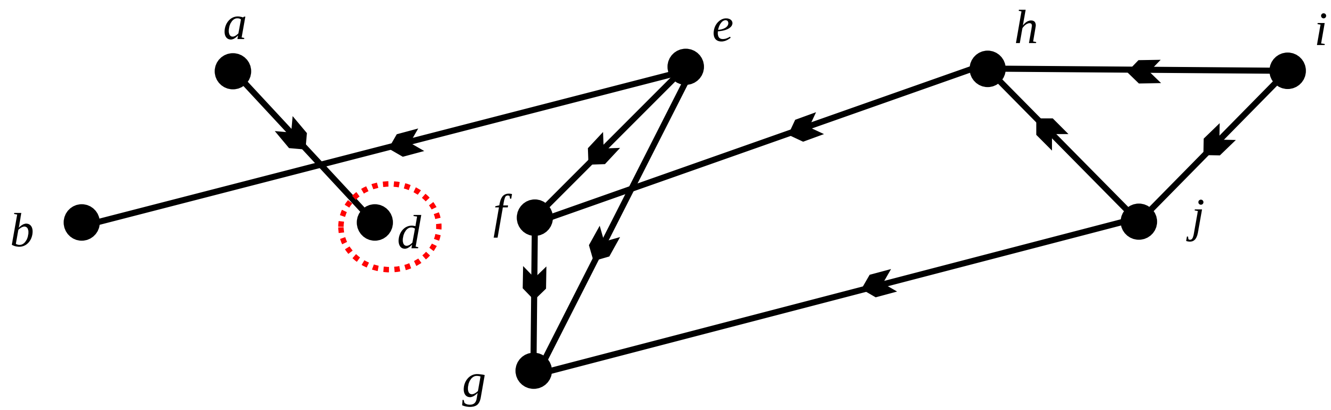
Busca em profundidade p/ digrafos

Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



Busca em profundidade p/ digrafos

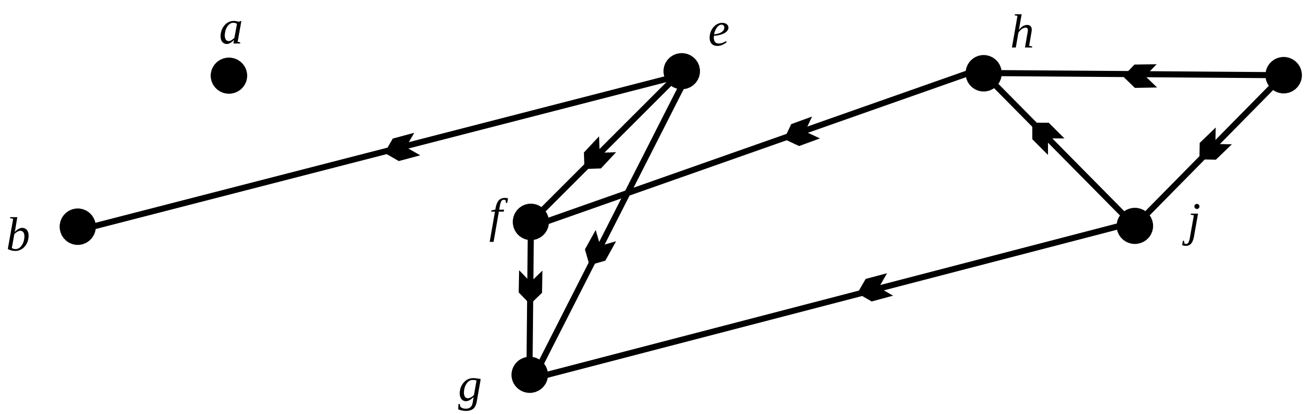
Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



c

Busca em profundidade p/ digrafos

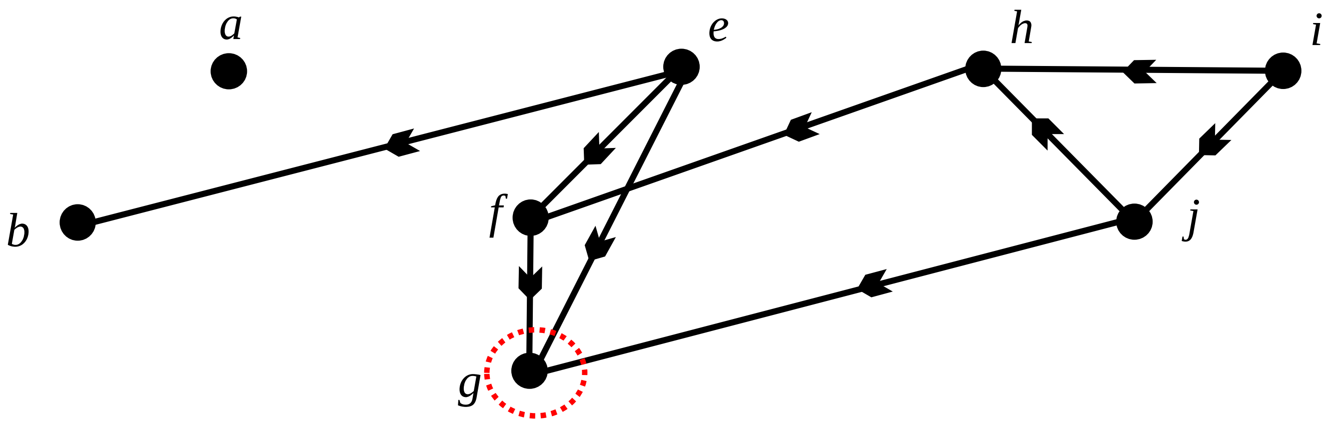
Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



d c

Busca em profundidade p/ digrafos

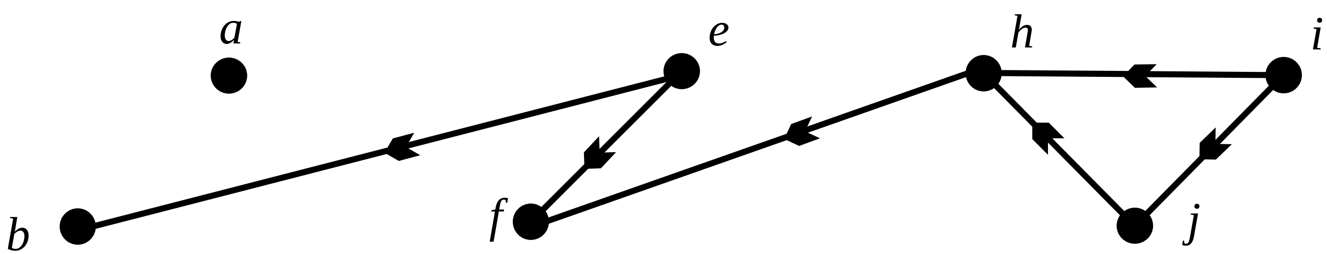
Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



d c

Busca em profundidade p/ digrafos

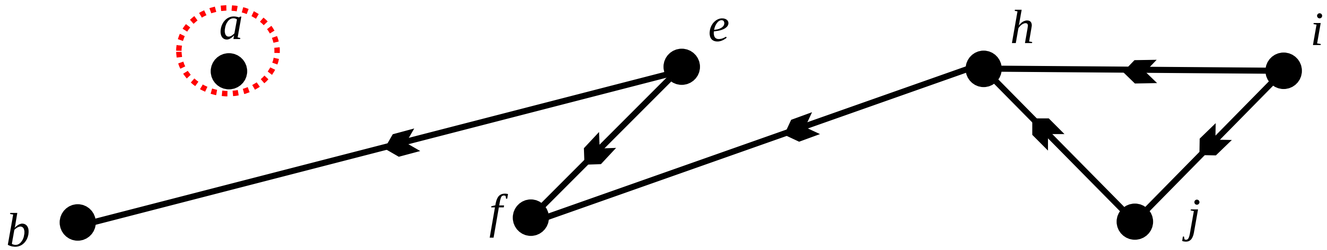
Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



g d c

Busca em profundidade p/ digrafos

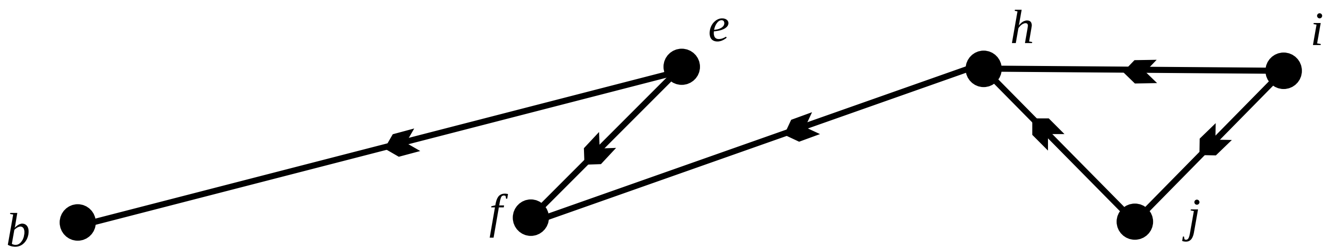
Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



g d c

Busca em profundidade p/ digrafos

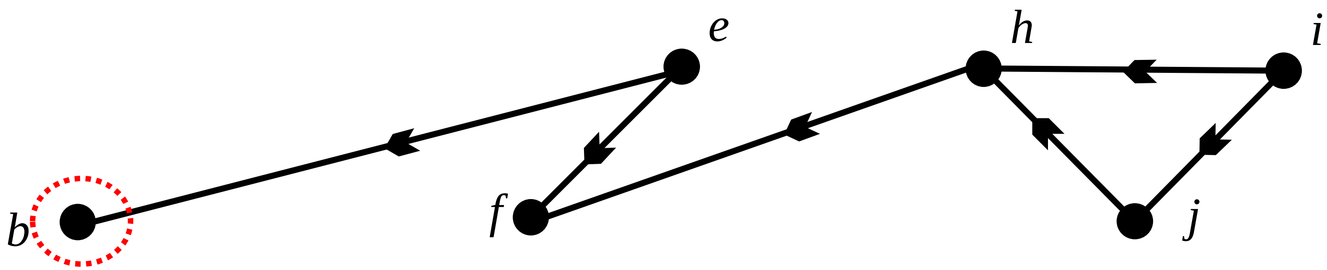
Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



a g d c

Busca em profundidade p/ digrafos

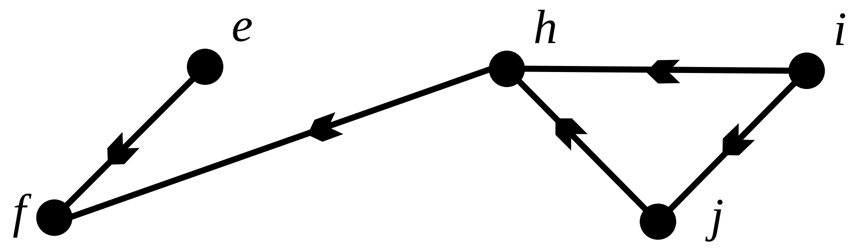
Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



a g d c

Busca em profundidade p/ digrafos

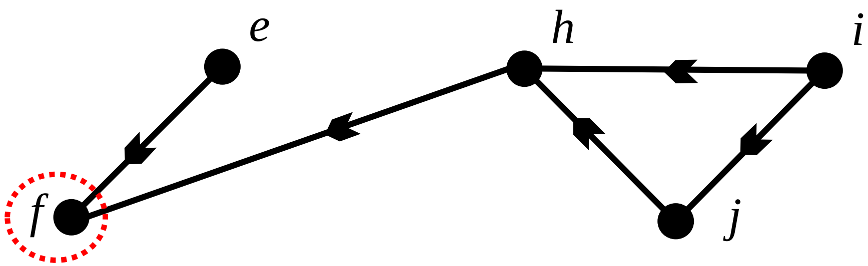
Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



b a g d c

Busca em profundidade p/ digrafos

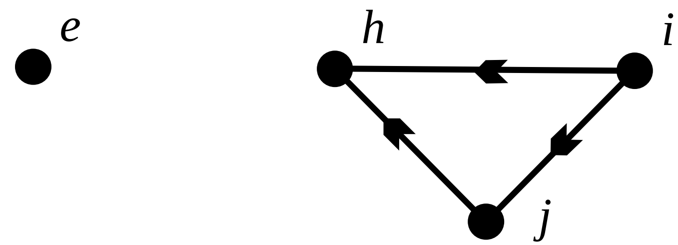
Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



b a g d c

Busca em profundidade p/ digrafos

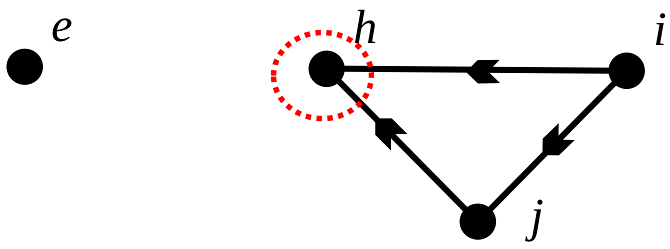
Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



f b a g d c

Busca em profundidade p/ digrafos

Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



f b a g d c

Busca em profundidade p/ digrafos

Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.

h f b a g d c

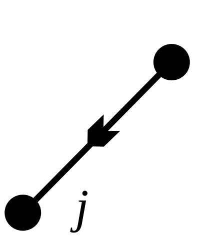
Busca em profundidade p/ digrafos

Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.

h f b a g d c

Busca em profundidade p/ digrafos

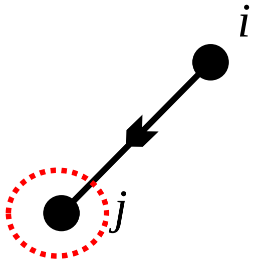
Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



e h f b a g d c

Busca em profundidade p/ digrafos

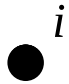
Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



e h f b a g d c

Busca em profundidade p/ digrafos

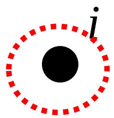
Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



j e h f b a g d c

Busca em profundidade p/ digrafos

Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



j e h f b a g d c

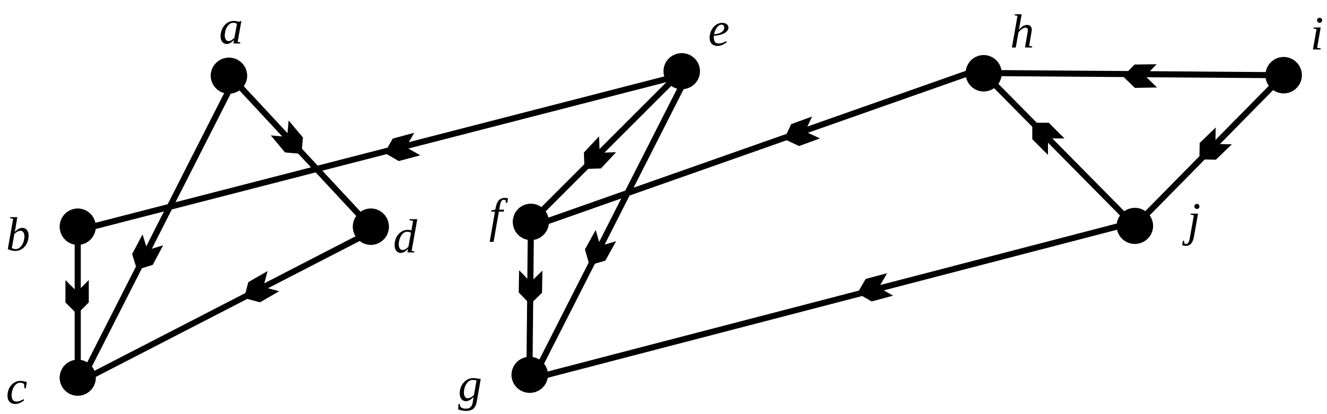
Busca em profundidade p/ digrafos

Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.

i j e h f b a g d c

Busca em profundidade p/ digrafos

Exercício: Desenvolver um algoritmo alternativo para obter uma *ordenação topológica* de um digrafo acíclico *G*.



i j e h f b a g d c

Ordenação topológica!

Busca em largura

Inicialização

$t \leftarrow 0$ ***-- t é o relógio ou tempo global***

$F \leftarrow \varnothing$ ***-- fila auxiliar para a busca em largura***

<u>para</u> todo vértice v em V(G) <u>faça</u>

        $L(v) \leftarrow 0$ ***-- L(v) é o índice de v na busca em largura***

        $pai(v) \leftarrow$ *null*   ***-- ponteiros que definem a floresta de largura***

Para que a busca atinja todos os vértices, no caso de G ser desconexo:

<u>enquanto</u> existe v em V(G) tal que L(v) = 0 <u>faça</u>

            $nivel(v) \leftarrow 0$ ***-- como v é uma nova raiz, seu nível é igual a 0***

            $t \leftarrow t + 1$; $L(v) \leftarrow t$

            colocar v na fila F

            realizar a busca em largura

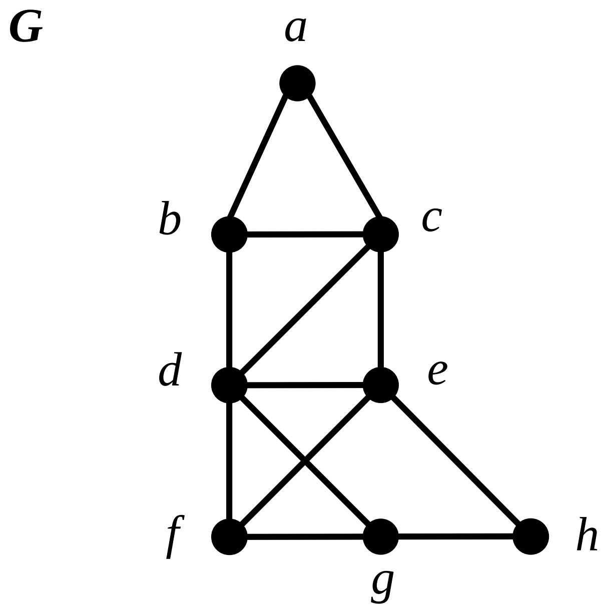
<u>fim-enquanto</u>

Busca em largura

Algoritmo iterativo para a busca em largura:

```
enquanto F ≠ Ø faça
        v ← primeiro elemento de F
        retirar v de F
        para todo vértice w em N(v) faça
            se L(w) = 0
                então  visitar aresta vw          --aresta “pai” da floresta de largura T
                       pai(w) ← v                 --v é o pai de w na floresta de largura T
                       nível(w) ← nível(v)+1;  t ← t + 1;  L(w) ← t
                       colocar w no final da fila F
                senão  se nível(w) = nível(v)
                         então se pai(w) = pai(v)
                                 então vw é aresta “irmão”  } visitar a aresta
                                 senão vw é aresta “primo”  } somente se w
                                                              ainda está em F
                         senão  se nível(w) = nível(v)+1
                                 então vw é aresta “tio”
        fim-para
fim-enquanto
```

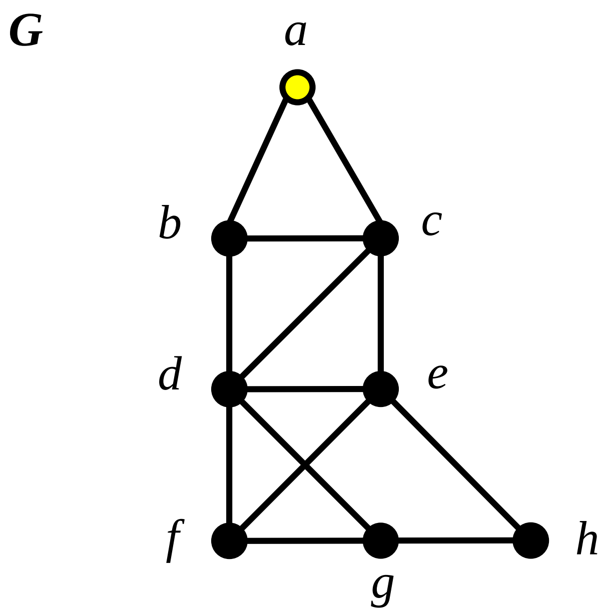
Busca em largura



$F = \emptyset$

v	a	b	c	d	e	f	g	h
$L(v)$	0	0	0	0	0	0	0	0

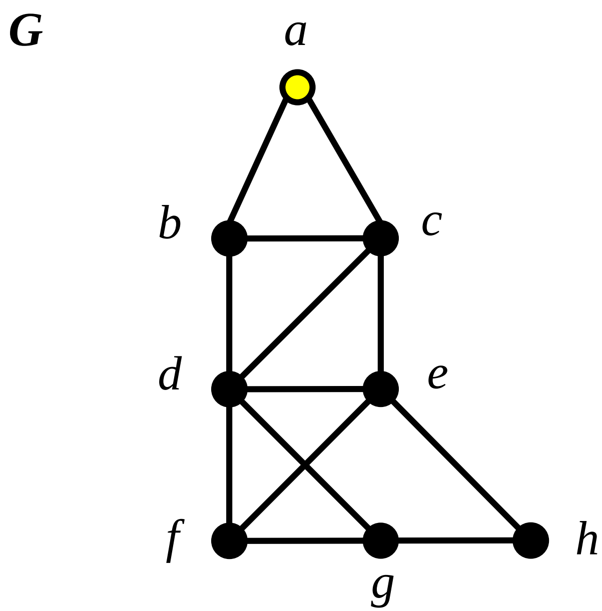
Busca em largura



F* = *a

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***
L(*v*)	1	0	0	0	0	0	0	0

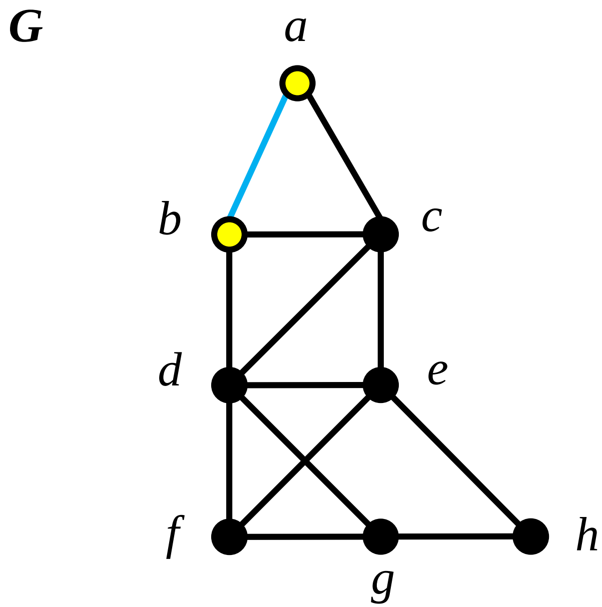
Busca em largura



$F = \cancel{a}$

v	a	b	c	d	e	f	g	h
$L(v)$	1	0	0	0	0	0	0	0

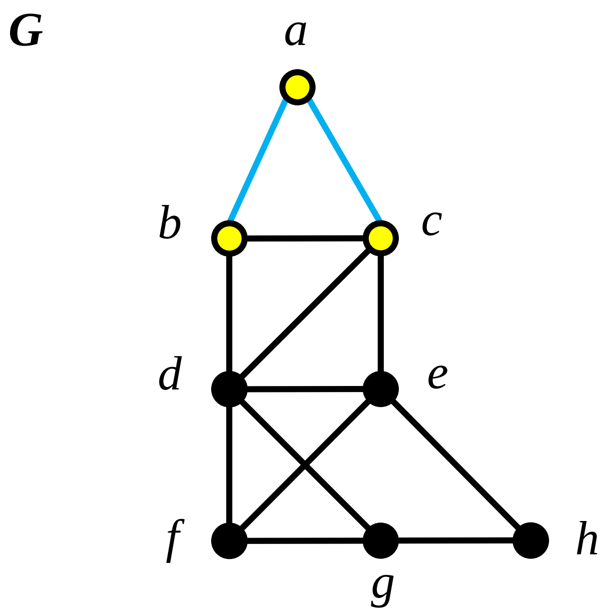
Busca em largura



$F = \cancel{a}\ b$

v	a	b	c	d	e	f	g	h
$L(v)$	1	2	0	0	0	0	0	0

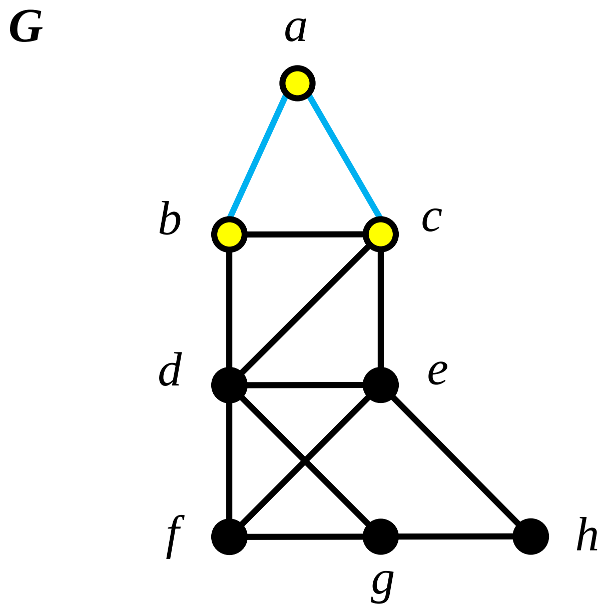
Busca em largura



$F = \cancel{a}\ b\ c$

v	a	b	c	d	e	f	g	h
$L(v)$	1	2	3	0	0	0	0	0

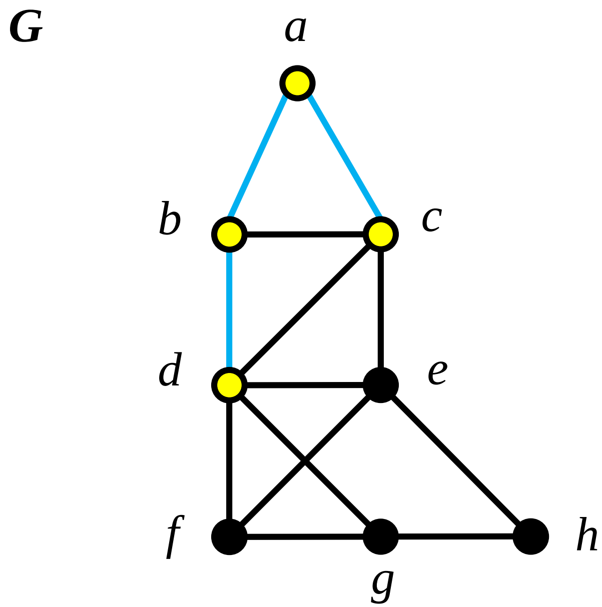
Busca em largura



F* = ~~*a*~~ ~~*b*~~ *c

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***
$L(v)$	1	2	3	0	0	0	0	0

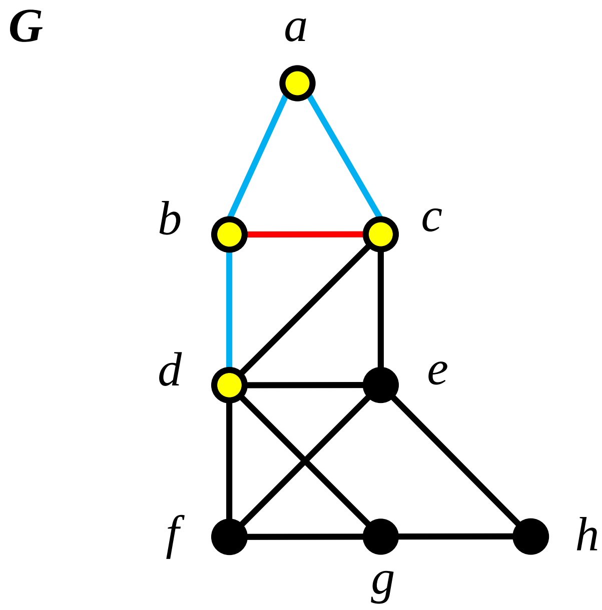
Busca em largura



F* = ~~*a*~~ ~~*b*~~ *c d

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*
$L(v)$	1	2	3	4	0	0	0	0

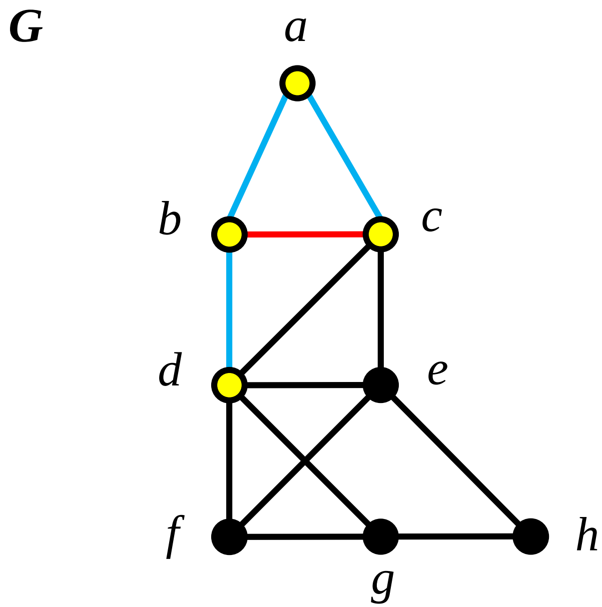
Busca em largura



$F = \cancel{a}\ \cancel{b}\ c\ d$

v	a	b	c	d	e	f	g	h
$L(v)$	1	2	3	4	0	0	0	0

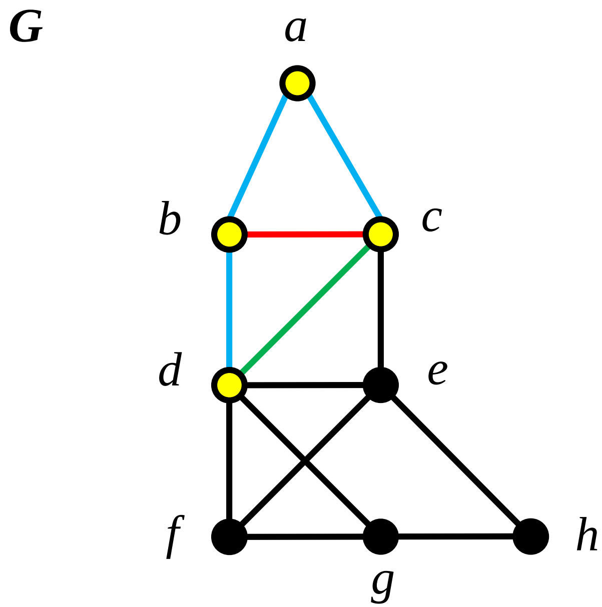
Busca em largura



F = ~~a~~ ~~b~~ ~~c~~ d

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***
$L(v)$	1	2	3	4	0	0	0	0

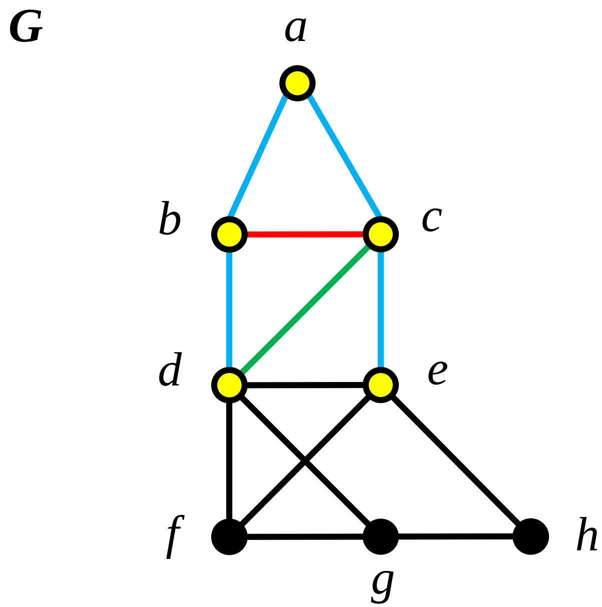
Busca em largura



$F = \cancel{a}\ \cancel{b}\ \cancel{c}\ d$

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*
$L(v)$	1	2	3	4	0	0	0	0

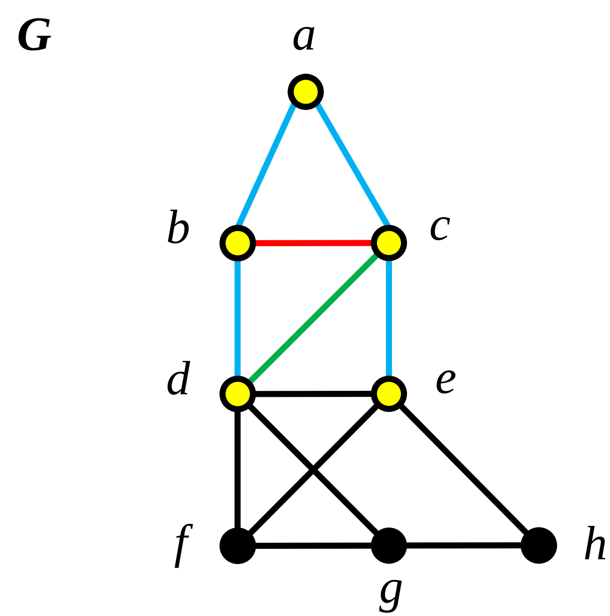
Busca em largura



F* = ~~*a*~~ ~~*b*~~ ~~*c*~~ *d e

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***
$L(v)$	1	2	3	4	5	0	0	0

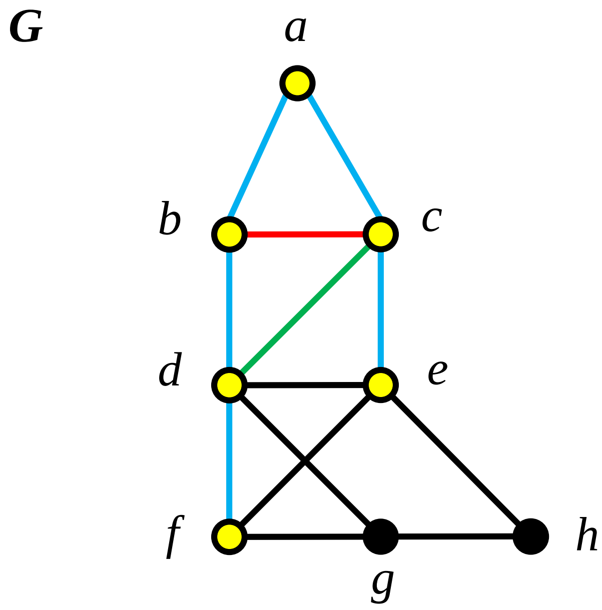
Busca em largura



$F = \cancel{a}\ \cancel{b}\ \cancel{c}\ \cancel{d}\ e$

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*
$L(v)$	1	2	3	4	5	0	0	0

Busca em largura

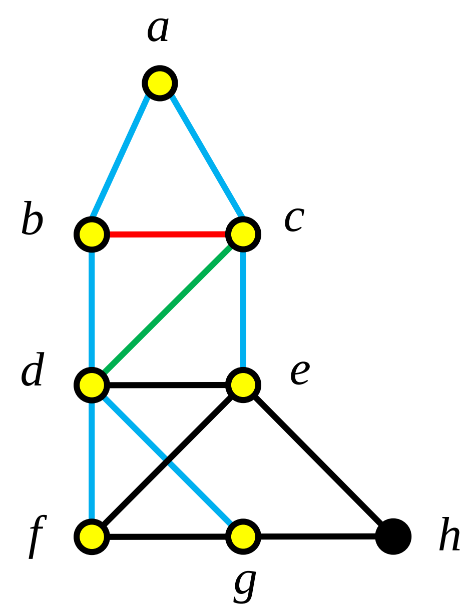


F* = ~~*a*~~ ~~*b*~~ ~~*c*~~ ~~*d*~~ *e f

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***
$L(v)$	1	2	3	4	5	6	0	0

Busca em largura

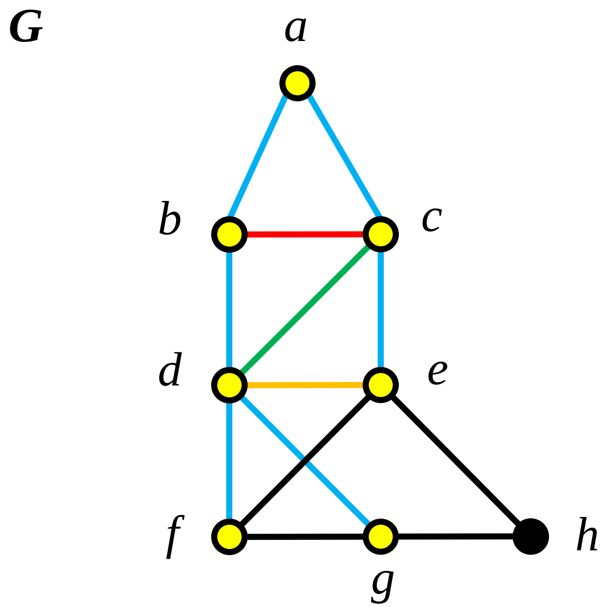
G



$\boldsymbol{F = \cancel{a}\ \cancel{b}\ \cancel{c}\ \cancel{d}\ e\ f\ g}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***
$L(v)$	1	2	3	4	5	6	7	0

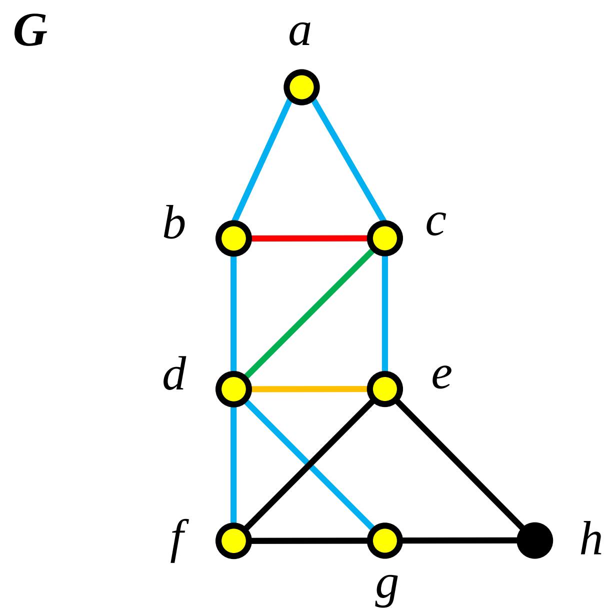
Busca em largura



F **=** ~~***a***~~ ~~***b***~~ ~~***c***~~ ~~***d***~~ ***e f g***

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***
$L(v)$	1	2	3	4	5	6	7	0

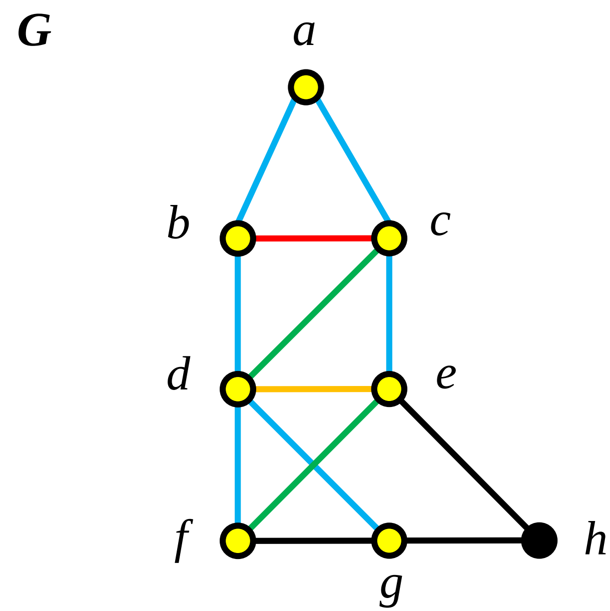
Busca em largura



$F = \cancel{a}\ \cancel{b}\ \cancel{c}\ \cancel{d}\ \cancel{e}\ f\ g$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***
$L(v)$	1	2	3	4	5	6	7	0

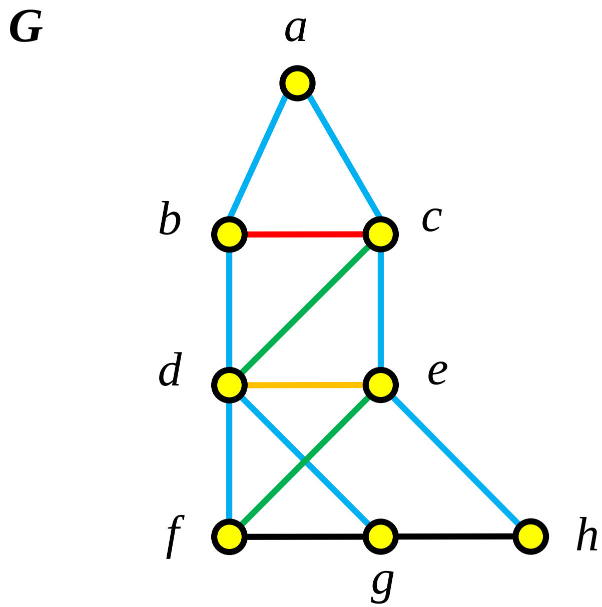
Busca em largura



F* = ~~*a*~~ ~~*b*~~ ~~*c*~~ ~~*d*~~ ~~*e*~~ *f* *g

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*
$L(v)$	1	2	3	4	5	6	7	0

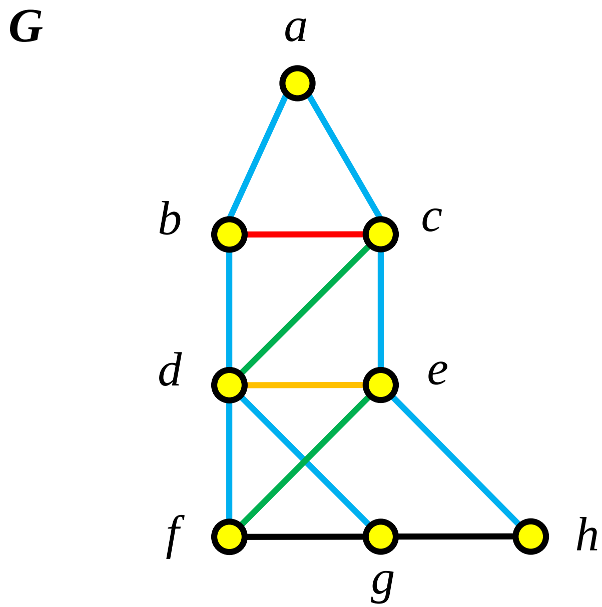
Busca em largura



F = ~~a~~ ~~b~~ ~~c~~ ~~d~~ ~~e~~ f g h

v	a	b	c	d	e	f	g	h
$L(v)$	1	2	3	4	5	6	7	8

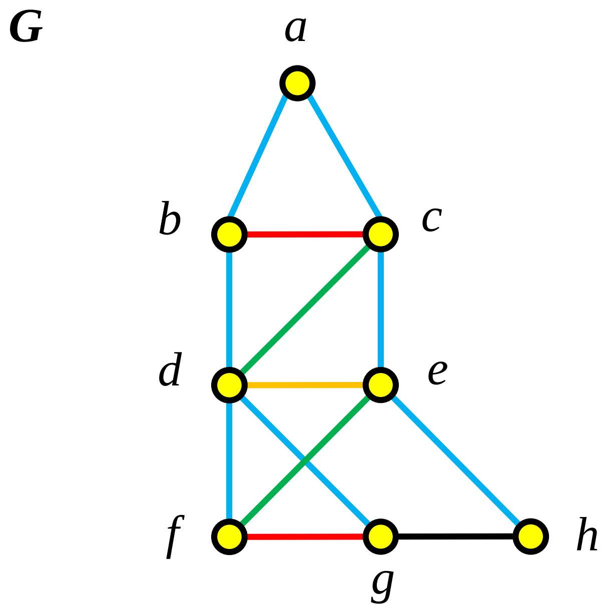
Busca em largura



F* = ~~*a*~~ ~~*b*~~ ~~*c*~~ ~~*d*~~ ~~*e*~~ ~~*f*~~ *g h

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***
$L(v)$	1	2	3	4	5	6	7	8

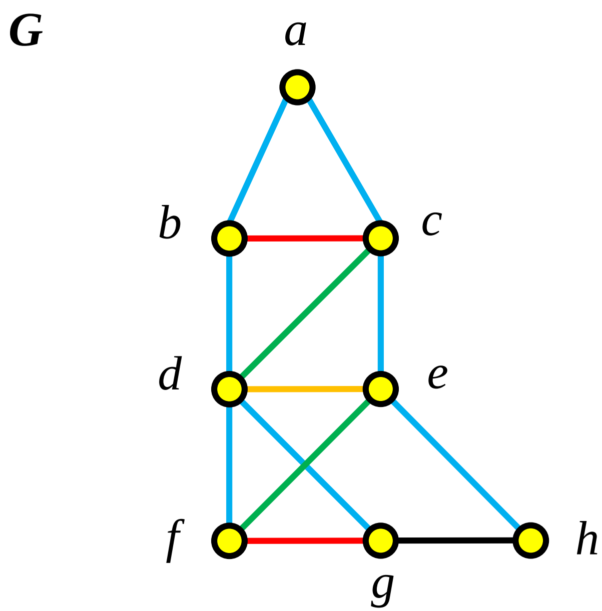
Busca em largura



F* = ~~*a*~~ ~~*b*~~ ~~*c*~~ ~~*d*~~ ~~*e*~~ ~~*f*~~ *g h

v	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*
$L(v)$	1	2	3	4	5	6	7	8

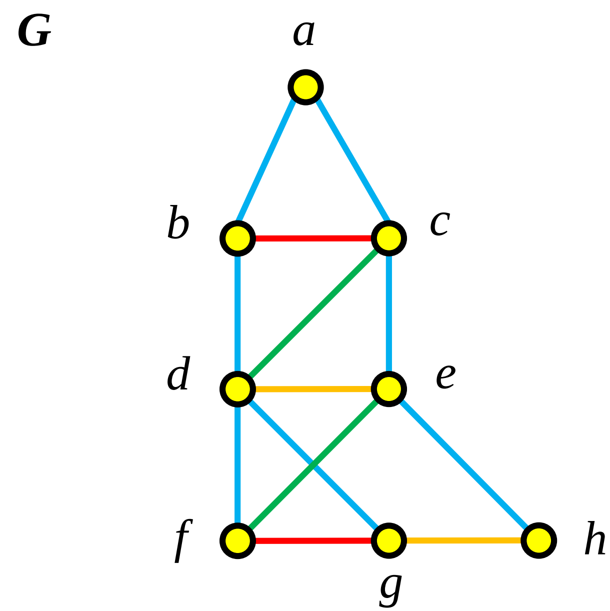
Busca em largura



F* = ~~*a*~~ ~~*b*~~ ~~*c*~~ ~~*d*~~ ~~*e*~~ ~~*f*~~ ~~*g*~~ *h

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***
$L(v)$	1	2	3	4	5	6	7	8

Busca em largura

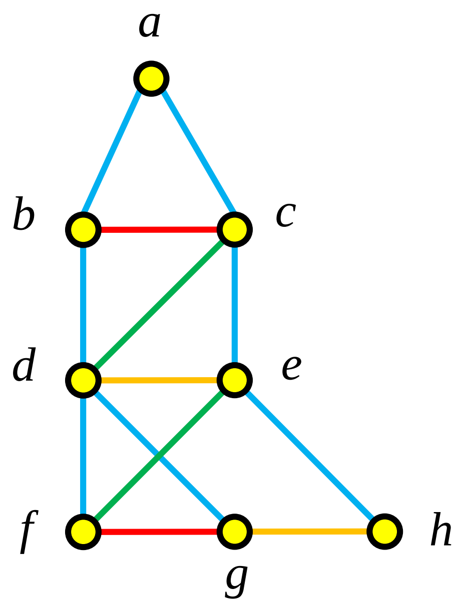


$F = \cancel{a}\ \cancel{b}\ \cancel{c}\ \cancel{d}\ \cancel{e}\ \cancel{f}\ \cancel{g}\ h$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***
$L(v)$	1	2	3	4	5	6	7	8

Busca em largura

G



$F = \cancel{a}\ \cancel{b}\ \cancel{c}\ \cancel{d}\ \cancel{e}\ \cancel{f}\ \cancel{g}\ \cancel{h}$

v	***a***	***b***	***c***	***d***	***e***	***f***	***g***	***h***
$L(v)$	1	2	3	4	5	6	7	8

Busca em largura

“Esgotar o pai antes de processar os filhos”

“A próxima aresta a ser visitada parte sempre do vértice mais antigo na busca”

Busca em largura

Complexidade da busca em largura:

$$O(n + m)$$

(onde $n = |V(G)|$ e $m = |E(G)|$)

Busca em largura

- A floresta de largura *T* é uma **floresta geradora de *G*** (isto é, uma floresta que alcança todos os vértices de *G*); neste caso, todos os vértices de *G* são alcançados pela busca e ficam com um valor $L(v)$ diferente de zero no final da mesma.

- Somente as arestas-***pai*** **(azuis)** (ligando pai e filho) pertencem à floresta de profundidade *T*. As arestas-***irmão*** **(vermelhas)**, ***primo*** **(amarelas)** e ***tio*** **(verdes)** não pertencem a *T*.

Busca em largura

Caracterização das arestas-pai (arestas azuis)

Seja vw aresta de G. Então:

vw é uma aresta-pai (da floresta de largura)
se e somente se
$nível(w) = nível(v) + 1$ e, no momento da visita, $L(w) = 0$

Busca em largura

Caracterização das arestas-tio (arestas verdes)

Seja vw aresta de G. Então:

vw é uma aresta-tio

se e somente se

$nível(w) = nível(v) + 1$ e, no momento da visita, $L(w) \neq 0$

Busca em largura

Caracterização das arestas-irmão (arestas vermelhas)

Seja vw aresta de G. Então:

$\boldsymbol{vw}$ é uma aresta-irmão

se e somente se

$nível(w) = nível(v)$ e $pai(w) = pai(v)$

Busca em largura

Caracterização das arestas-primo (arestas amarelas)

Seja vw aresta de G. Então:

vw é uma aresta-primo

se e somente se

$nível(w) = nível(v)$ e $pai(w) \neq pai(v)$

Busca em largura

Propriedade fundamental da busca em largura

Seja vw uma aresta de um grafo G, e seja T uma floresta de largura de G. Então:

$$\boldsymbol{|\, nível(v) - nível(w) \,| \le 1.}$$

- A demonstração dessa propriedade se reduz ao caso em que vw é uma aresta-tio em relação a T.
- Os vértices que se encontram em F em qualquer momento da busca diferem em no máximo um nível!

Busca em largura

Aplicação 1: Dado um grafo conexo G e um vértice x de G, determinar as *distâncias de* x a todos os demais vértices.

Solução:

- Aplicar uma busca em largura em G com raiz x, obtendo uma árvore de largura T
- Ao final da busca, $dist(x, v) = nível(v)$, para todo v
- Um caminho mínimo de x a v é dado pelo caminho de x a v na árvore T.

Busca em largura

Aplicação 2: Dado um *labirinto* representado por uma matriz, determinar o menor caminho da entrada até a saída do labirinto (se existir).

Solução:

- Modelar a matriz como um grafo, onde a entrada e a saída são representadas por vértices x e y.
- Realizar uma busca em largura em G com uma única raiz x, obtendo uma árvore de largura T.
- Ao final da busca, se $L(y) \neq 0$ então y pode ser alcançado a partir de x.
- O caminho de x a y em T é a solução, e $dist(x, y) = nível(y)$.

Busca em largura

Aplicação 3: Dado um grafo *G*, determinar se *G* é bipartido.

Solução:

- Aplicar uma busca em largura em *G* uma raiz qualquer, obtendo uma floresta de largura *T*.
- Ao final da busca, *G* é bipartido se e somente se *T* não contém ***arestas-irmão*** nem ***arestas-primo.***
- Note que arestas-irmão e arestas-primo fecham ***ciclos ímpares!***
- Para definir uma 2-coloração de *G*: os vértices em níveis pares recebem uma cor, e os vértices em níveis ímpares outra cor.

Busca em largura

Exercício: Elaborar um método (algoritmo) para resolver a seguinte questão: Dado um grafo conexo G e uma árvore geradora T de G, decidir se T é uma árvore de largura para G. Isto é, decidir se existe uma busca em largura em G que produza T como árvore de largura.

Busca em largura

Exercício: Elaborar uma adaptação da busca em largura, para aplicá-la a digrafos.

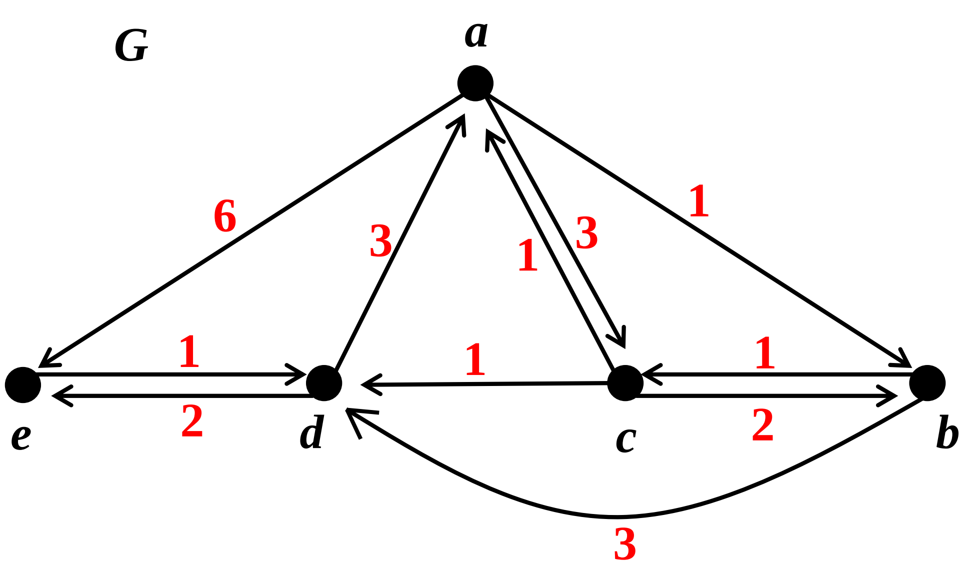
Busca em largura

Exercício: Mostre que o algoritmo a seguir funciona como uma busca em profundidade se a estrutura de dados *D* for uma pilha, e como uma busca em largura se a estrutura de dados *D* for uma fila.

desmarcar todos os vértices
escolher uma raiz v; pai(v) ← null; marcar v; inserir v em D
<u>*enquanto*</u> *D ≠ Ø* <u>*faça*</u>
    seja v o primeiro elemento de D
    <u>*se*</u> *existe w em N(v) que esteja desmarcado*
        <u>*então*</u> *visitar aresta vw*
            pai(w) ← v
            marcar w
            inserir w em D
        <u>*senão*</u> *remover v de D*
<u>*fim-enquanto*</u>

Algoritmo de Dijkstra

Problema: Dado um digrafo *G* com pesos positivos nas arestas, determinar caminhos mínimos de um vértice *v* a todos os demais vértices do digrafo.



matriz de pesos W

	a	***b***	***c***	***d***	***e***
a	0	1	3	∞	6
b	∞	0	1	3	∞
c	1	2	0	1	∞
d	3	∞	∞	0	2
e	∞	∞	∞	1	0

$$W(x, y) = \begin{cases} 0, \text{ se } x = y \\ \text{peso da aresta } xy, \text{ se houver } (x \neq y) \\ \infty, \text{ se não houver aresta } xy \ (x \neq y) \end{cases}$$

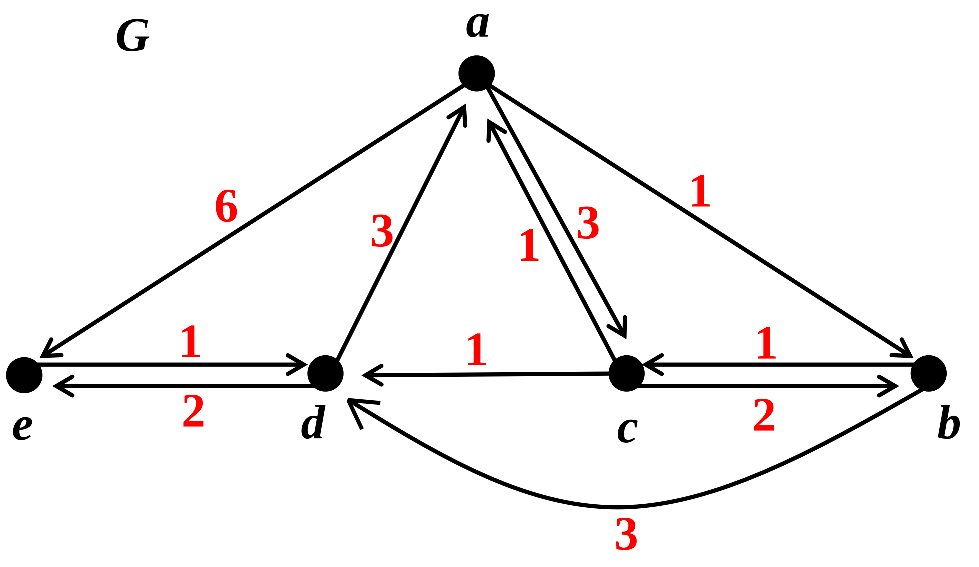
Algoritmo de Dijkstra

Realizando uma adaptação no algoritmo do exercício anterior, e fazendo com que a estrutura de dados *D* seja uma *fila de prioridades* (*heap*), obtemos o **Algoritmo de Dijkstra.**

desmarcar todos os vértices; *escolher uma raiz v*; $L(v) \leftarrow 0$; *pai*$(v) \leftarrow$ *null*; *inserir v em D*
para todo w em $V(G)\backslash\{v\}$ *faça* $L(w) \leftarrow \infty$ *e pai*$(w) \leftarrow$ *null*
enquanto $D \neq \emptyset$ *faça*
    seja v o primeiro elemento de D   **-- *v é elemento de menor valor L(v)***
    remover v de D; marcar v
    para todo vértice w em $N_{out}(v)$ *que esteja desmarcado faça*
        se w não pertence a D então inserir w em D com prioridade $L(w)$
        se $L(v) + W(v, w) < L(w)$
            então $L(w) \leftarrow L(v) + W(v, w)$
                reposicionar w em D **(*de acordo com sua nova prioridade*)**
                pai$(w) \leftarrow v$
    fim-para
fim-enquanto

Algoritmo de Dijkstra

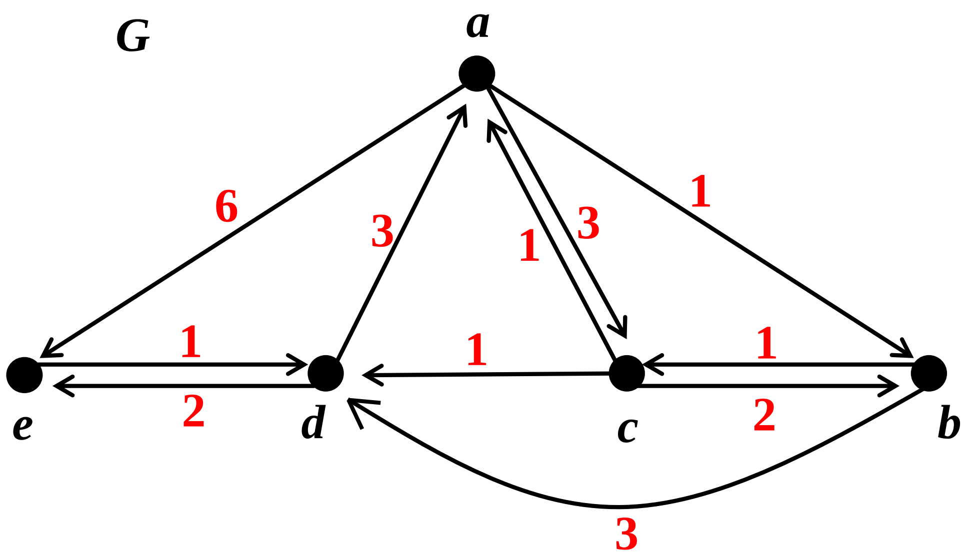
raiz: a



v	*a*	*b*	*c*	*d*	*e*
L(v)	0	∞	∞	∞	∞
pai(v)	*null*	*null*	*null*	*null*	*null*

Algoritmo de Dijkstra

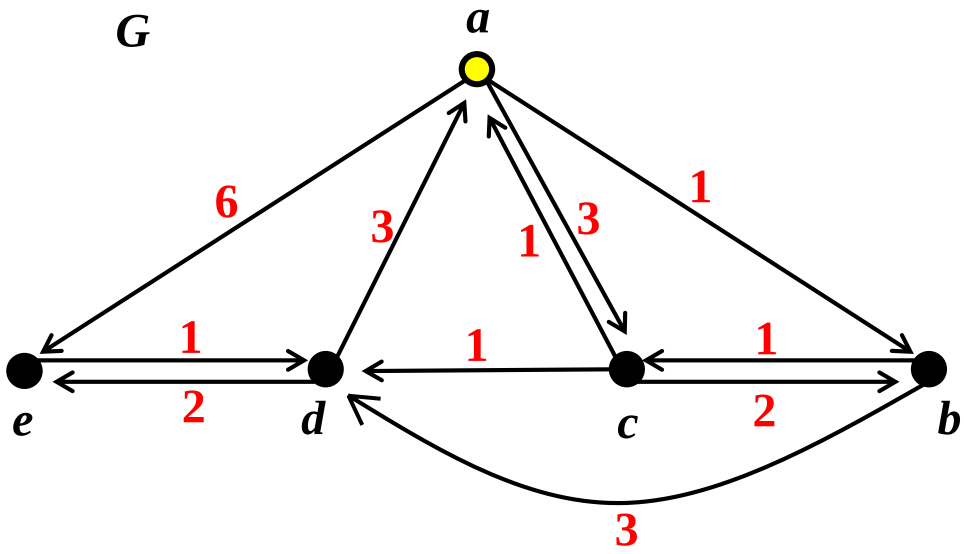
raiz: a



v	***a***	***b***	***c***	***d***	***e***
***L*(*v*)**	0	∞	∞	∞	∞
***pai*(*v*)**	*null*	*null*	*null*	*null*	*null*

Algoritmo de Dijkstra

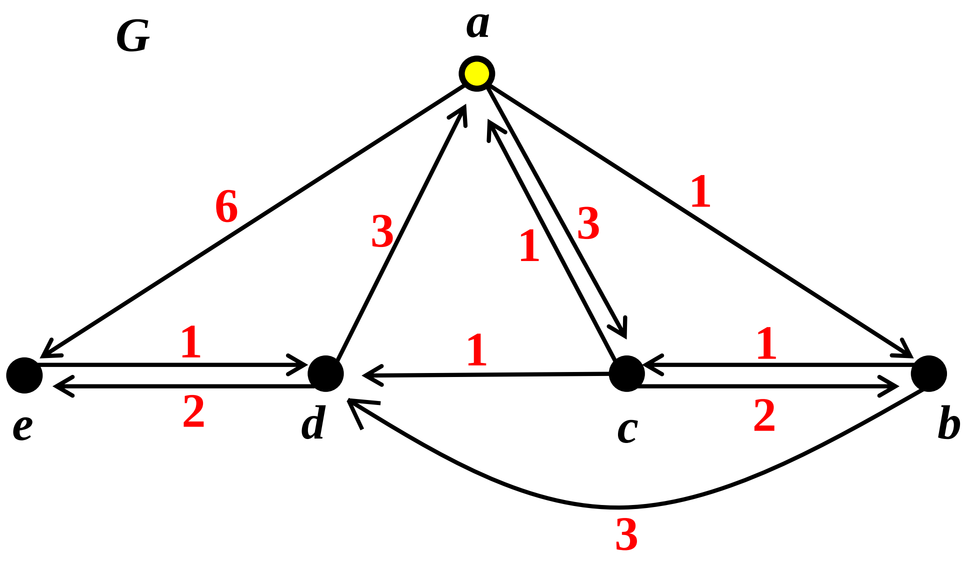
raiz: a



v	*a*	*b*	*c*	*d*	*e*
L(*v*)	0	∞	∞	∞	∞
pai(*v*)	*null*	*null*	*null*	*null*	*null*

Algoritmo de Dijkstra

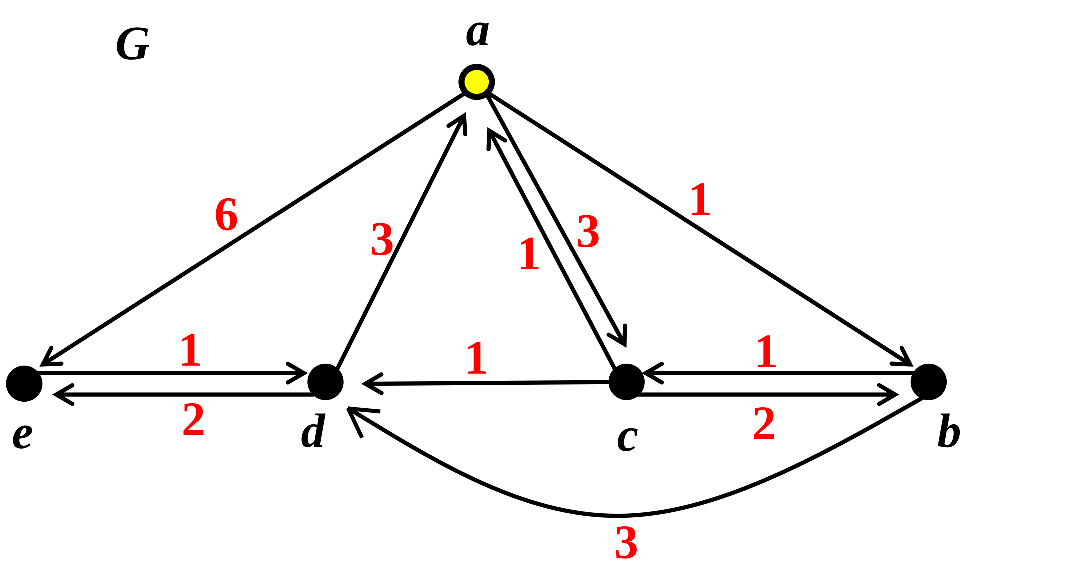
raiz: a



v	*a*	*b*	*c*	*d*	*e*
***L*(*v*)**	0	1	∞	∞	∞
***pai*(*v*)**	*null*	*a*	*null*	*null*	*null*

Algoritmo de Dijkstra

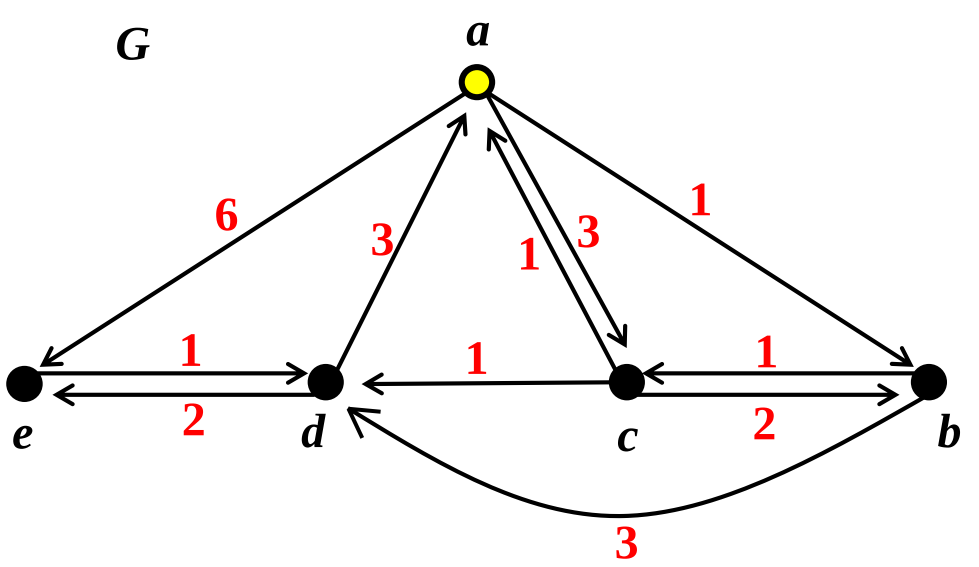
raiz: a



v	***a***	***b***	***c***	***d***	***e***
***L*(*v*)**	0	1	3	∞	∞
***pai*(*v*)**	*null*	*a*	*a*	*null*	*null*

Algoritmo de Dijkstra

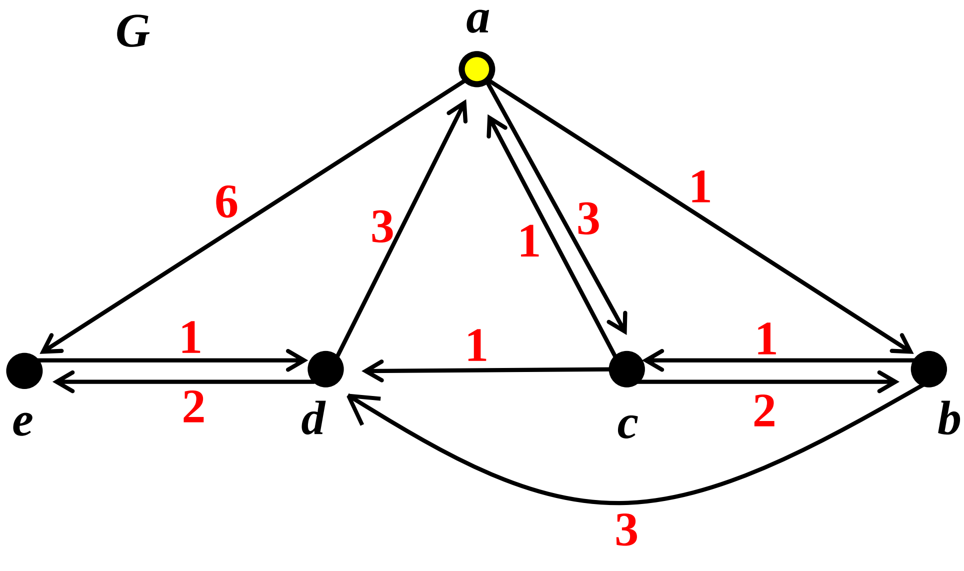
raiz: a



v	a	b	c	d	e
$L(v)$	0	1	3	∞	6
$pai(v)$	*null*	*a*	*a*	*null*	*a*

Algoritmo de Dijkstra

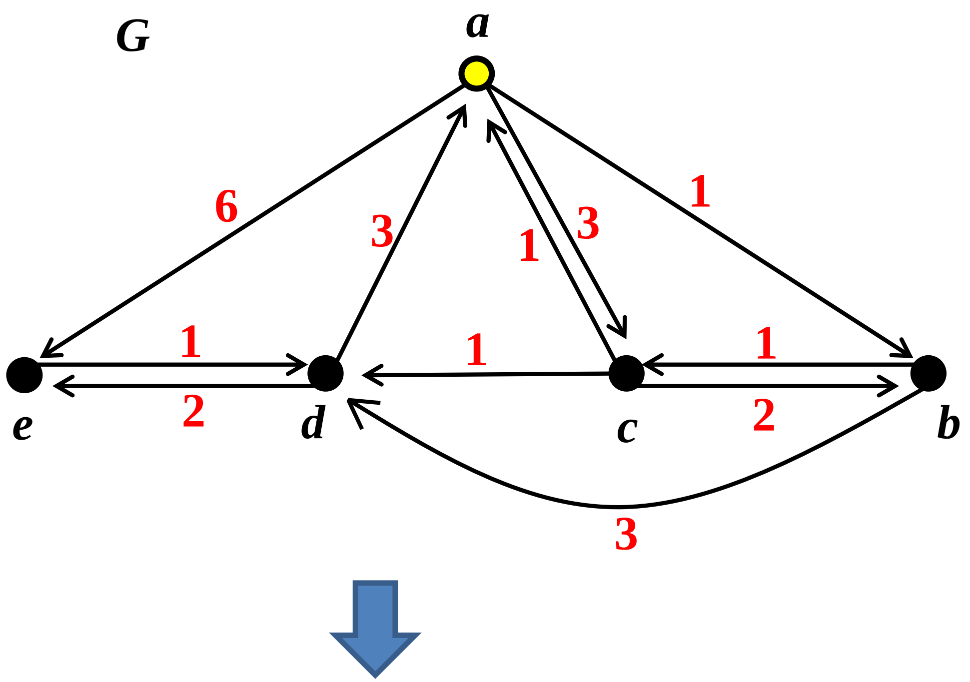
raiz: a



v	*a*	*b*	*c*	*d*	*e*
***L*(*v*)**	0	1	3	∞	6
***pai*(*v*)**	*null*	*a*	*a*	*null*	*a*

Algoritmo de Dijkstra

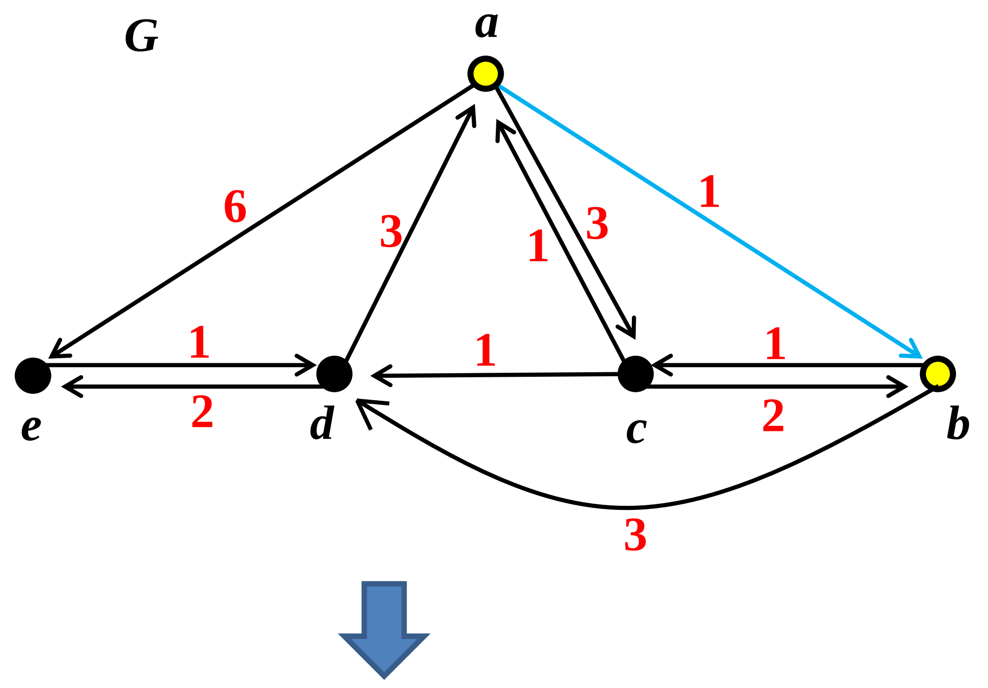
raiz: a



v	***a***	***b***	***c***	***d***	***e***
***L*(*v*)**	0	1	3	∞	6
***pai*(*v*)**	*null*	*a*	*a*	*null*	*a*

Algoritmo de Dijkstra

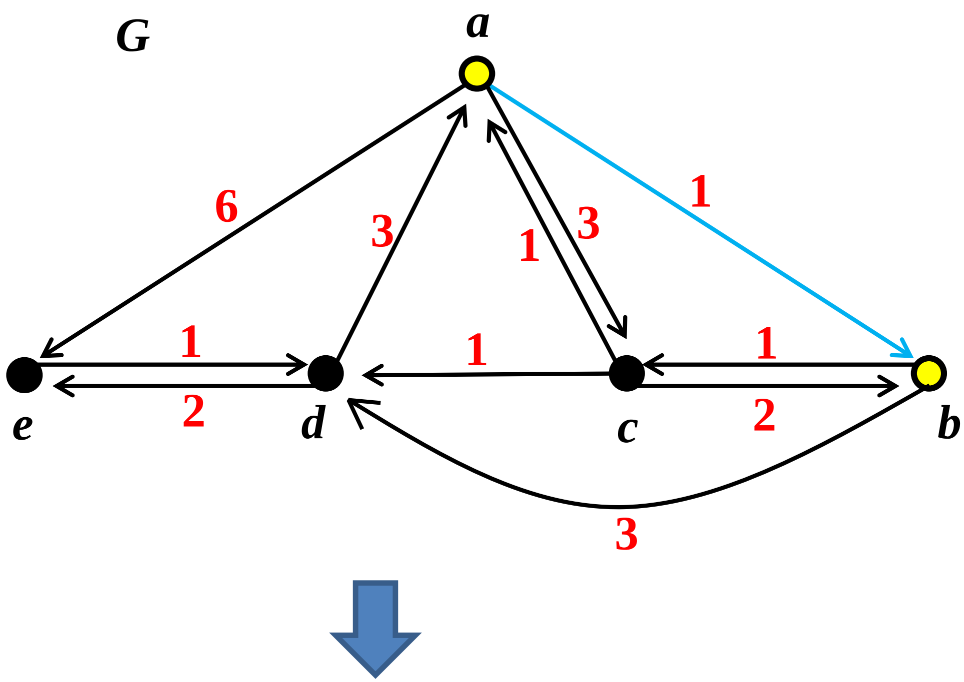
raiz: a



v	*a*	*b*	*c*	*d*	*e*
L(*v*)	0	1	3	∞	6
pai(*v*)	*null*	*a*	*a*	*null*	*a*

Algoritmo de Dijkstra

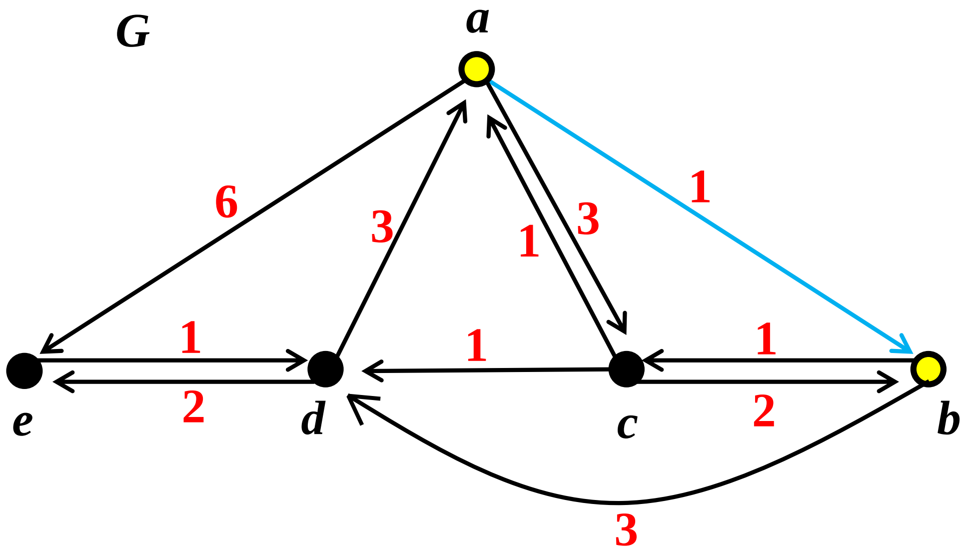
raiz: a



v	*a*	*b*	*c*	*d*	*e*
L(*v*)	0	1	2	∞	6
pai(*v*)	*null*	*a*	*b*	*null*	*a*

Algoritmo de Dijkstra

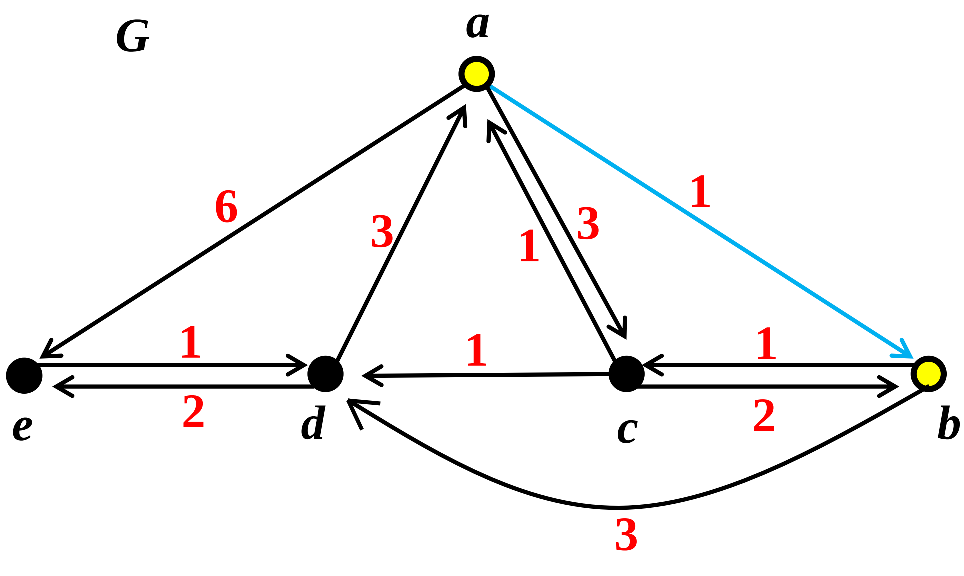
raiz: a



v	*a*	*b*	*c*	*d*	*e*
***L*(*v*)**	0	1	2	4	6
***pai*(*v*)**	*null*	*a*	*b*	*b*	*a*

Algoritmo de Dijkstra

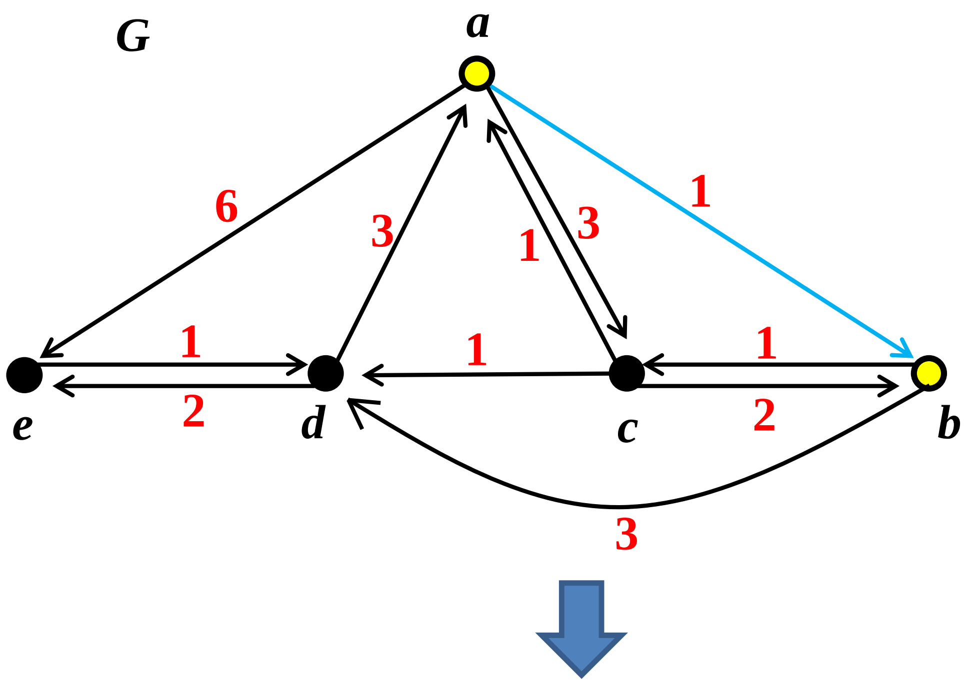
raiz: a



v	*a*	*b*	*c*	*d*	*e*
***L*(*v*)**	0	1	2	4	6
***pai*(*v*)**	*null*	*a*	*b*	*b*	*a*

Algoritmo de Dijkstra

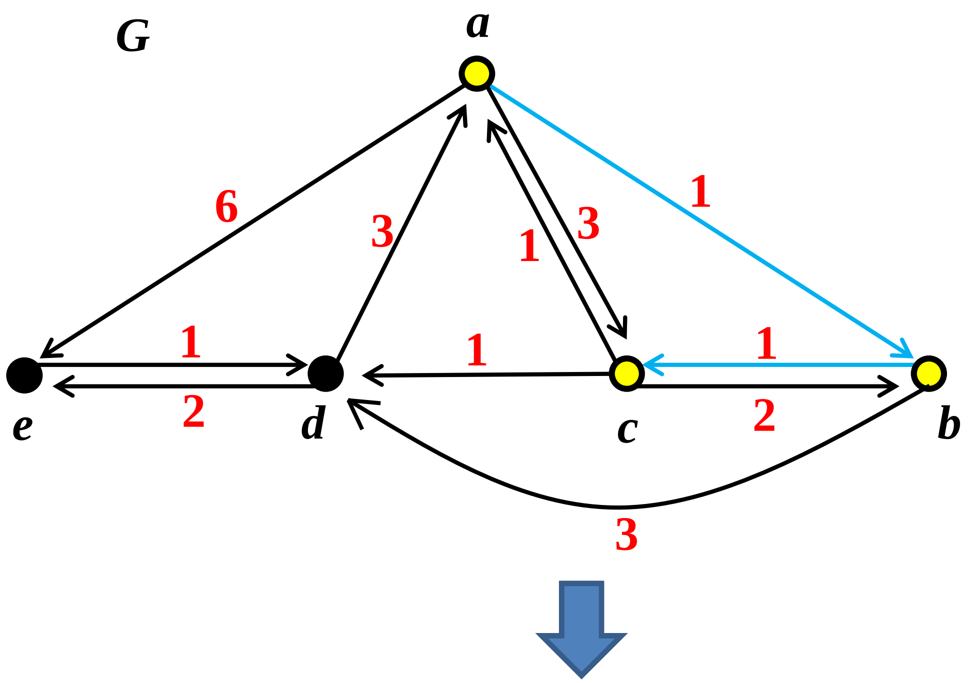
raiz: a



v	*a*	*b*	*c*	*d*	*e*
***L*(*v*)**	0	1	2	4	6
***pai*(*v*)**	*null*	*a*	*b*	*b*	*a*

Algoritmo de Dijkstra

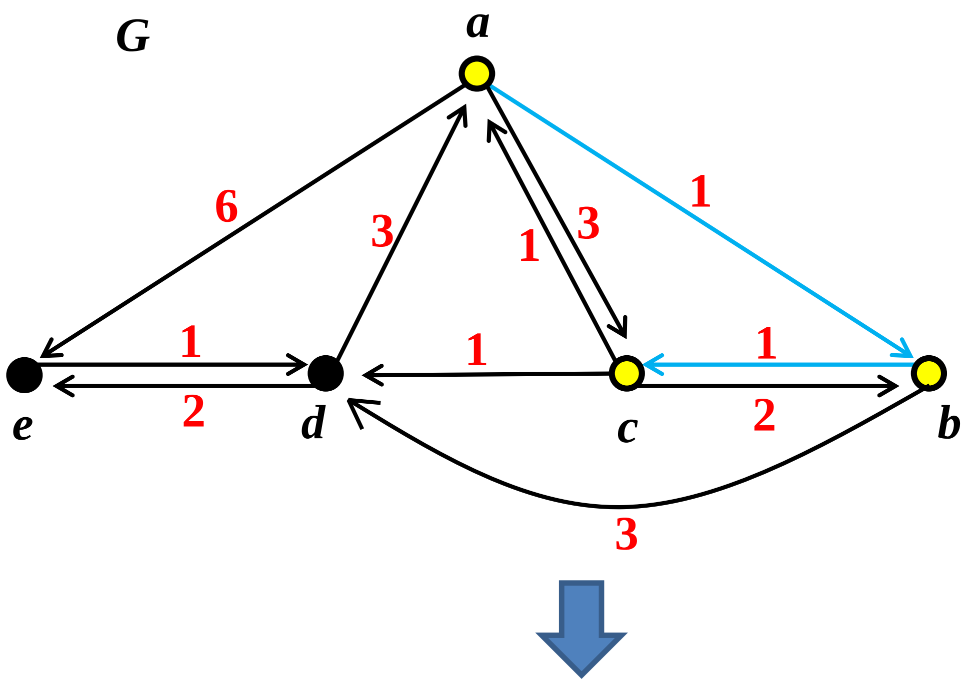
raiz: a



v	*a*	*b*	*c*	*d*	*e*
***L*(*v*)**	0	1	2	4	6
***pai*(*v*)**	*null*	*a*	*b*	*b*	*a*

Algoritmo de Dijkstra

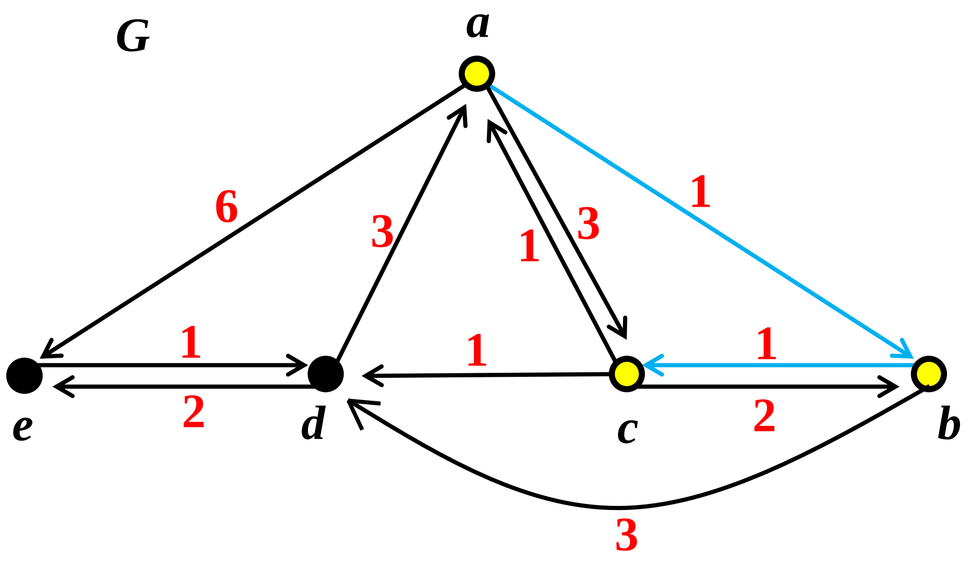
raiz: a



v	*a*	*b*	*c*	*d*	*e*
***L*(*v*)**	0	1	2	3	6
***pai*(*v*)**	*null*	*a*	*b*	*c*	*a*

Algoritmo de Dijkstra

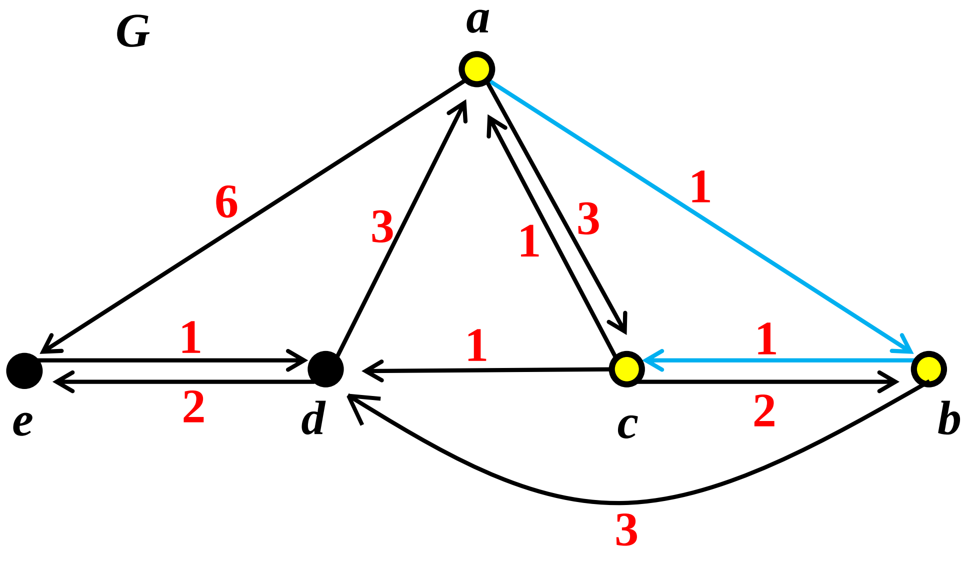
raiz: a



v	***a***	***b***	***c***	***d***	***e***
***L*(*v*)**	0	1	2	3	6
***pai*(*v*)**	*null*	*a*	*b*	*c*	*a*

Algoritmo de Dijkstra

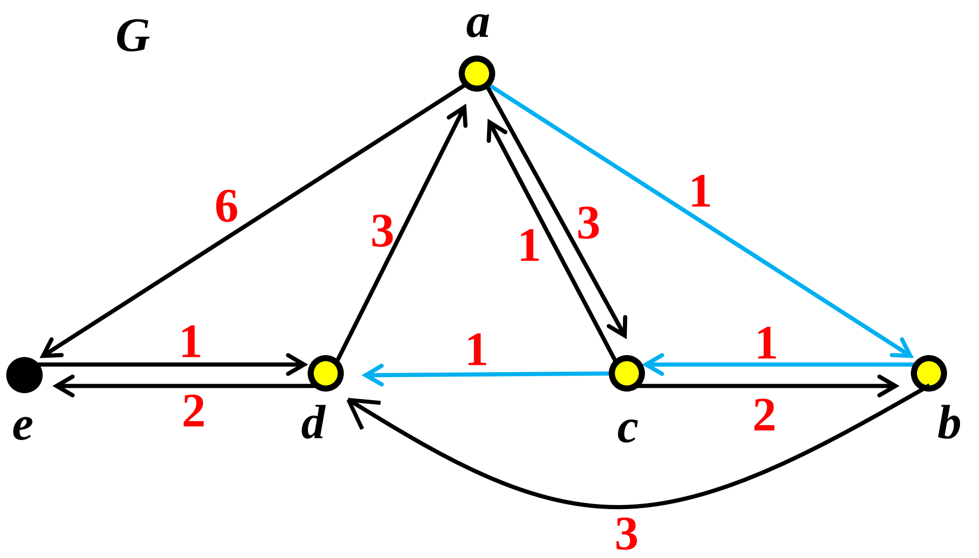
raiz: a



v	*a*	*b*	*c*	*d*	*e*
***L*(*v*)**	0	1	2	3	6
***pai*(*v*)**	*null*	*a*	*b*	*c*	*a*

Algoritmo de Dijkstra

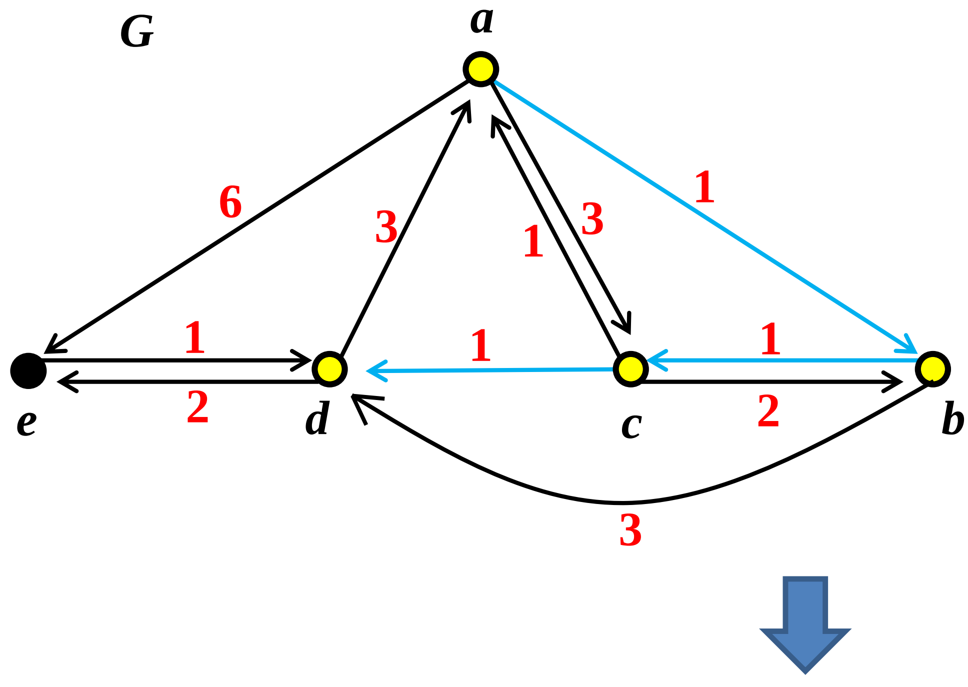
raiz: a



v	*a*	*b*	*c*	*d*	*e*
***L*(*v*)**	0	1	2	3	6
***pai*(*v*)**	*null*	*a*	*b*	*c*	*a*

Algoritmo de Dijkstra

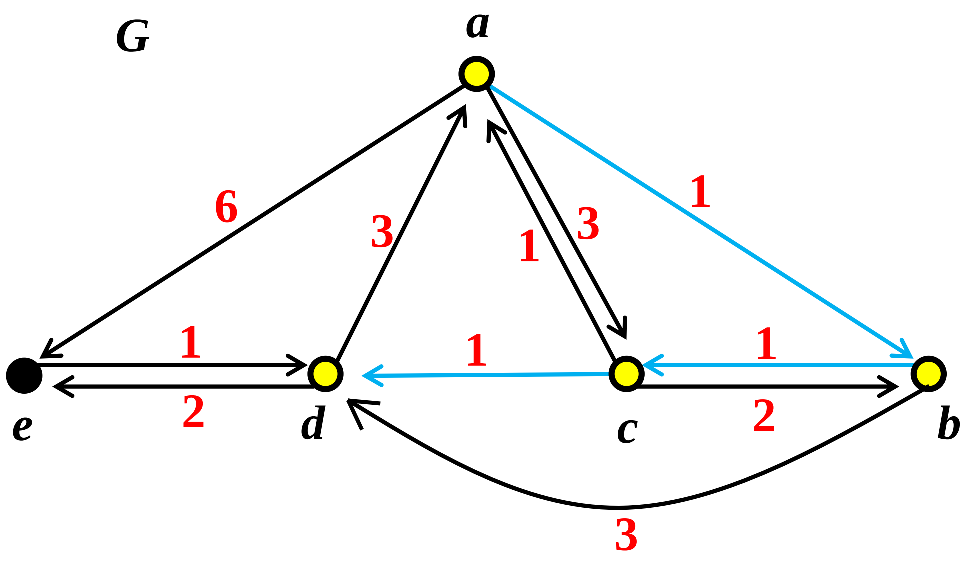
raiz: a



v	*a*	*b*	*c*	*d*	*e*
***L*(*v*)**	0	1	2	3	5
***pai*(*v*)**	*null*	*a*	*b*	*c*	*d*

Algoritmo de Dijkstra

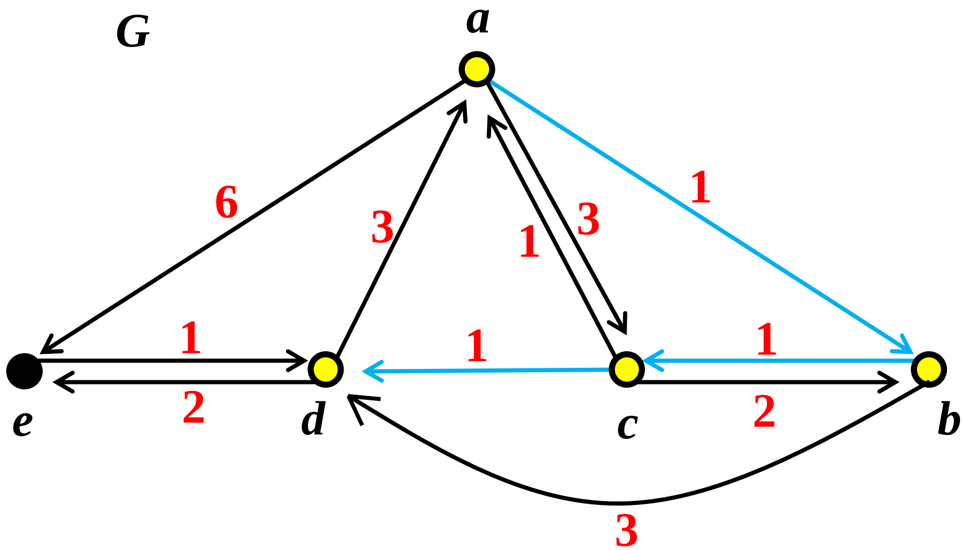
raiz: a



v	*a*	*b*	*c*	*d*	*e*
***L*(*v*)**	0	1	2	3	5
***pai*(*v*)**	*null*	*a*	*b*	*c*	*d*

Algoritmo de Dijkstra

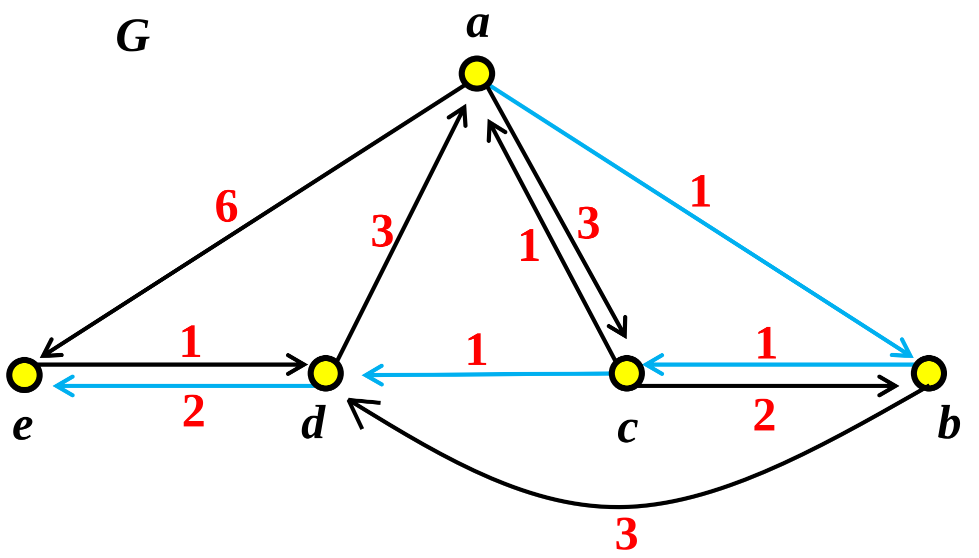
raiz: a



v	***a***	***b***	***c***	***d***	***e***
***L*(*v*)**	0	1	2	3	5
***pai*(*v*)**	*null*	*a*	*b*	*c*	*d*

Algoritmo de Dijkstra

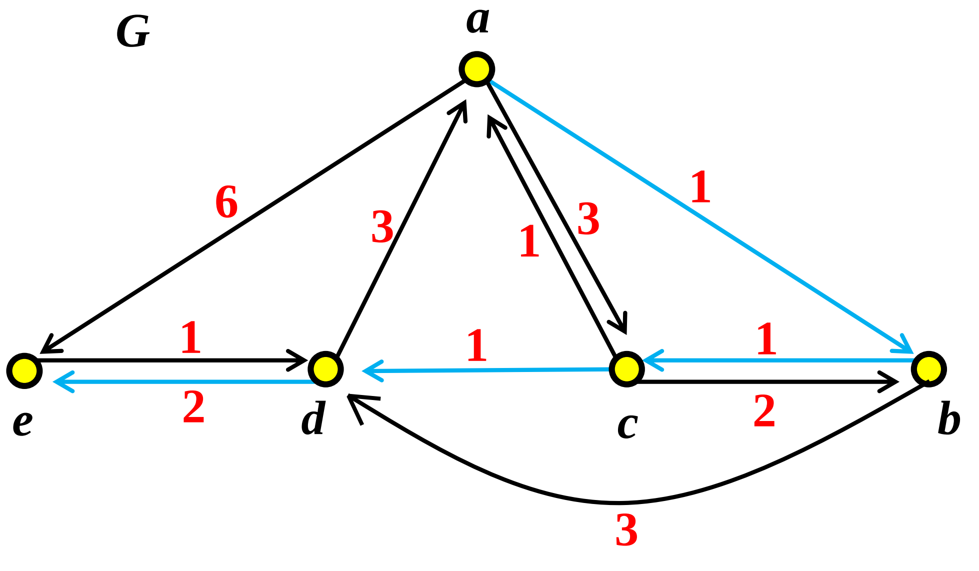
raiz: a



v	*a*	*b*	*c*	*d*	*e*
L(*v*)	0	1	2	3	5
pai(*v*)	*null*	*a*	*b*	*c*	*d*

Algoritmo de Dijkstra

raiz: a



v	***a***	***b***	***c***	***d***	***e***
L(v)	0	1	2	3	5
pai(v)	*null*	*a*	*b*	*c*	*d*

FIM!

Algoritmo de Dijkstra

Complexidade

A complexidade do Algoritmo de Dijkstra é dominada pelas seguintes operações:

- n remoções da estrutura D -- $O(n \log n)$
- n inserções na estrutura D -- $O(n \log n)$
- no máximo m atualizações de prioridade (rótulos $L(v)$) na estrutura D -- $O(m \log n)$
- **TOTAL: $O(\,(n + m) \log n\,)$**

Algoritmo de Dijkstra

Questão: Como determinar os caminhos mínimos?

Solução: Observe que os ponteiros $pai(w)$ determinam naturalmente uma árvore T, chamada de ***árvore de caminhos mínimos*** (a partir da raiz escolhida). Na simulação, esta árvore T está formada pelas arestas azuis.

Para determinar um caminho mínimo da raiz v até um vértice w, basta tomar o *caminho de v a w em T*. O custo deste caminho é precisamente $L(w)$.

Algoritmo de Dijkstra

Questão: Como achar os caminhos mínimos da raiz a todos os demais vértices em um grafo G **não** direcionado?

Solução: Basta transformar G em um digrafo H da seguinte forma: fazer $V(H) = V(G)$ e, para cada aresta não direcionada xy com peso p em G, criar em H duas arestas direcionadas ***xy*** e ***yx***, ambas com peso p. Por outro lado, se não existe aresta xy em G, criar em H duas arestas direcionadas ***xy*** e ***yx*** com peso ∞. Basta então aplicar o algoritmo de Dijkstra a H.

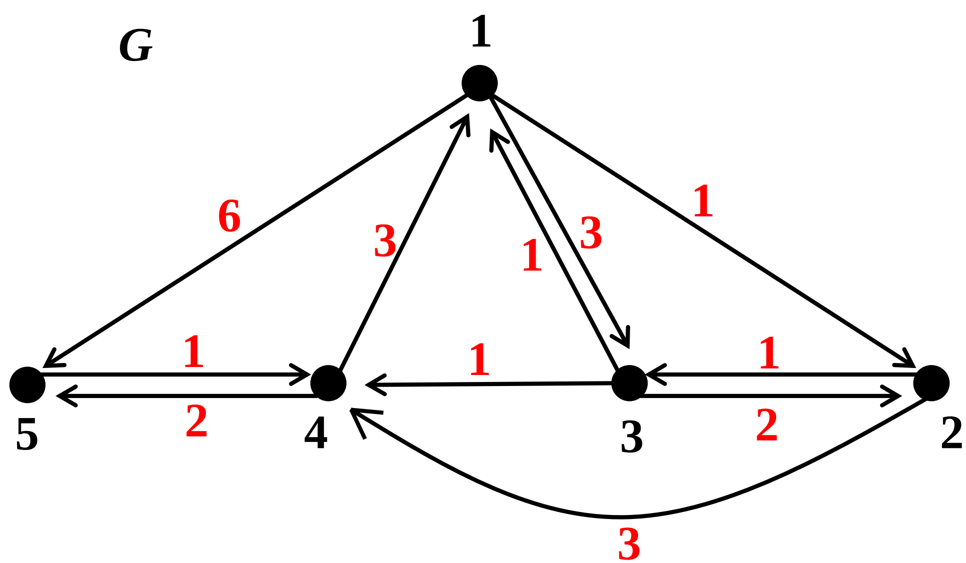
Algoritmo de Dijkstra

Exercício: Mostre que o Algoritmo de Dijkstra se comporta como uma busca em largura se os pesos de todas as arestas são iguais a um. Neste caso, a árvore de caminhos mínimos é uma árvore de largura, e cada rótulo $L(w)$ é igual à distância (em arestas) da raiz até w.

Algoritmo de Floyd-Warshall

Problema: Dado um digrafo *G* com pesos positivos nas arestas, determinar caminhos mínimos entre todos os pares ordenados (*v*, *w*) de vértices do digrafo.

→ A entrada para este problema é a mesma do anterior.



matriz de pesos W

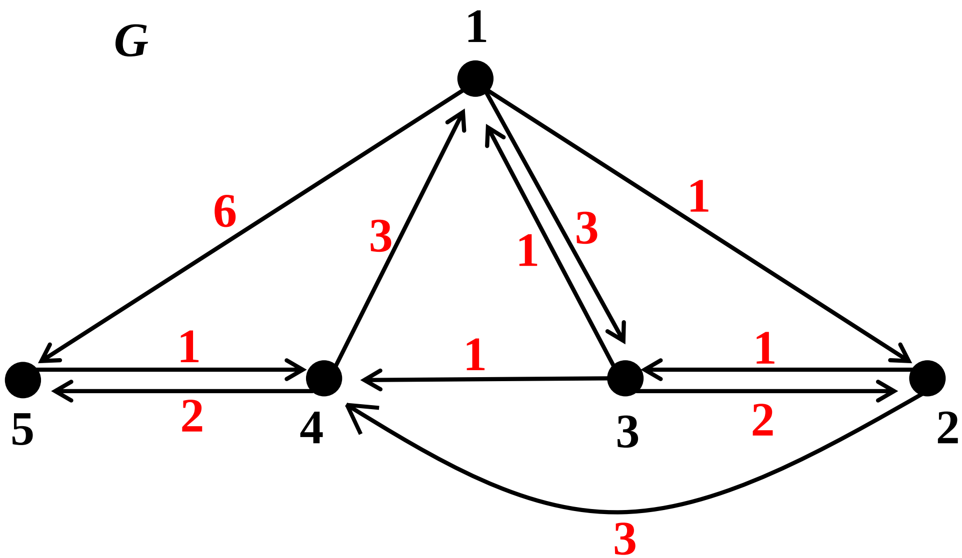
	1	**2**	**3**	**4**	**5**
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

$$W(i, j) = \begin{cases} 0, \text{ se } i = j \\ \text{peso da aresta } ij, \text{ se houver } (i \neq j) \\ \infty, \text{ se não houver aresta } ij \ (i \neq j) \end{cases}$$

Algoritmo de Floyd-Warshall

Problema: Dado um digrafo *G* com pesos positivos nas arestas, determinar caminhos mínimos entre todos os pares ordenados (*v*, *w*) de vértices do digrafo.

→ A entrada para este problema é a mesma do anterior.



Matriz-solução: custos de todos os caminhos mínimos

	1	**2**	**3**	**4**	**5**
1	0	1	2	3	5
2	2	0	1	2	4
3	1	2	0	1	3
4	3	4	5	0	2
5	4	5	6	1	0

Algoritmo de Floyd-Warshall

Ideia do algoritmo: Calcular os valores $D^k(i, j)$

$D^k(i, j)$ = custo do caminho mínimo de ***i*** a ***j*** podendo usar como vértices intermediários os vértices do conjunto $\{1, 2, \ldots, k\} \setminus \{i, j\}$.

Observe que:

- $D^0(i, j) = W(i, j)$
  (valor na posição (i, j) da matriz de pesos W)
- $D^n(i, j)$ = ***custo do caminho mínimo de i a j***
  (valor na posição (i, j) da matriz-solução)

Algoritmo de Floyd-Warshall

Ideia do algoritmo: Calcular os valores $D^k(i, j)$

***Definindo as matrizes do algoritmo*:**

Para cada $k = 0, 1, 2, \ldots, n$, defina D^k como a matriz contendo todos os valores $D^k(i, j)$

Observe que:

- $D^0 = W$ (matriz de pesos da entrada)
- $D^n =$ ***matriz-solução***

Algoritmo de Floyd-Warshall

Ideia do algoritmo: Calcular os valores $\boldsymbol{D^k(i, j)}$

Esboço do algoritmo:

- *inicializar* $\boldsymbol{D^0} \leftarrow \boldsymbol{W}$
- *para cada* $k = 1, 2, \ldots, n$ *faça*

  calcular a matriz $\boldsymbol{D^k}$

Algoritmo de Floyd-Warshall

Ideia do algoritmo: Calcular os valores $\boldsymbol{D^k(i, j)}$

Esboço do algoritmo:

- *inicializar* $\boldsymbol{D^0} \leftarrow \boldsymbol{W}$
- *para cada* $k = 1, 2, \ldots, n$ *faça*

  calcular a matriz $\boldsymbol{D^k}$

Cálculo da matriz $\boldsymbol{D^k}$ **:**

para cada $i = 1, 2, \ldots, n$ *faça*

para cada $j = 1, 2, \ldots, n$ *faça*

calcular $\boldsymbol{D^k(i, j)}$

Algoritmo de Floyd-Warshall

Ideia do algoritmo: Calcular os valores $\boldsymbol{D^k(i, j)}$

Esboço do algoritmo:

- *inicializar* $\boldsymbol{D^0} \leftarrow \boldsymbol{W}$
- *para cada* $k = 1, 2, \ldots, n$ *faça*

  para cada $i = 1, 2, \ldots, n$ *faça*

  para cada $j = 1, 2, \ldots, n$ *faça*

  calcular $\boldsymbol{D^k(i, j)}$

Algoritmo de Floyd-Warshall

Ideia do algoritmo: Calcular os valores $D^k(i, j)$

Fórmula para calcular $D^k(i, j)$:

$$D^k(i, j) = \min \{ D^{k-1}(i, j) \, , \; D^{k-1}(i, k) + D^{k-1}(k, j) \}$$

Algoritmo de Floyd-Warshall

Ideia do algoritmo: Calcular os valores $D^k(i, j)$

Fórmula para calcular $D^k(i, j)$:

$$D^k(i, j) = \min \{ D^{k-1}(i, j),\ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$

custo do caminho mínimo de i a j
sem considerar k
como vértice intermediário
(valor da iteração anterior)

Algoritmo de Floyd-Warshall

Ideia do algoritmo: Calcular os valores $D^k(i, j)$

Fórmula para calcular $D^k(i, j)$:

$$D^k(i, j) = \min \{ D^{k-1}(i, j), \; D^{k-1}(i, k) + D^{k-1}(k, j) \}$$

custo do caminho mínimo de i a j
sem considerar k
como vértice intermediário
(valor da iteração anterior)

custo do caminho mínimo de i a j
considerando k
como vértice intermediário
(nova possibilidade)

Algoritmo de Floyd-Warshall

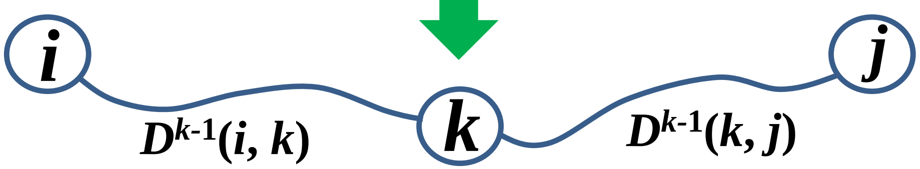
Ideia do algoritmo: Calcular os valores $D^k(i, j)$

Fórmula para calcular $D^k(i, j)$:

$$D^k(i, j) = \min \{ D^{k-1}(i, j), D^{k-1}(i, k) + D^{k-1}(k, j) \}$$

custo do caminho mínimo de i a j
sem considerar k
como vértice intermediário
(valor da iteração anterior)

custo do caminho mínimo de i a j
considerando k
como vértice intermediário
(nova possibilidade)



Algoritmo de Floyd-Warshall

Ideia do algoritmo: Calcular os valores $\boldsymbol{D^k(i, j)}$

Esboço do algoritmo:

- *inicializar* $\boldsymbol{D^0} \leftarrow \boldsymbol{W}$
- *para cada* $k = 1, 2, \ldots, n$ *faça*

  para cada $i = 1, 2, \ldots, n$ *faça*

  para cada $j = 1, 2, \ldots, n$ *faça*

  calcular $\boldsymbol{D^k(i, j)}$

Algoritmo de Floyd-Warshall

Ideia do algoritmo: Calcular os valores $\boldsymbol{D^k(i, j)}$

Esboço do algoritmo:

- *inicializar* $\boldsymbol{D^0} \leftarrow \boldsymbol{W}$
- *para cada* $k = 1, 2, \ldots, n$ *faça*

    para cada $i = 1, 2, \ldots, n$ *faça*

        para cada $j = 1, 2, \ldots, n$ *faça*

$$\boldsymbol{D^k(i, j) \leftarrow \min \{ D^{k-1}(i, j) \,,\; D^{k-1}(i, k) + D^{k-1}(k, j) \}}$$

Algoritmo de Floyd-Warshall

Ideia do algoritmo: Calcular os valores $\boldsymbol{D^k(i, j)}$

Esboço do algoritmo:

- *inicializar* $\mathbf{D^0} \leftarrow \boldsymbol{W}$
- *para cada* $k = 1, 2, \ldots, n$ *faça*

  para cada $i = 1, 2, \ldots, n$ *faça*

  para cada $j = 1, 2, \ldots, n$ *faça*

  $$\boldsymbol{D^k(i, j) \leftarrow \min \{ D^{k-1}(i, j) , \; D^{k-1}(i, k) + D^{k-1}(k, j) \}}$$

Note que $\boldsymbol{D^k}$ é calculada apenas a partir de $\boldsymbol{D^{k-1}}$ **!**

Algoritmo de Floyd-Warshall

Ideia do algoritmo: Calcular os valores $\boldsymbol{D^k(i, j)}$

Esboço do algoritmo:

- *inicializar* $\boldsymbol{D^0 \leftarrow W}$
- *para cada* k = 1, 2, …, n *faça*

$\mathrm{O}(n^3)$

para cada i = 1, 2, …, n *faça*

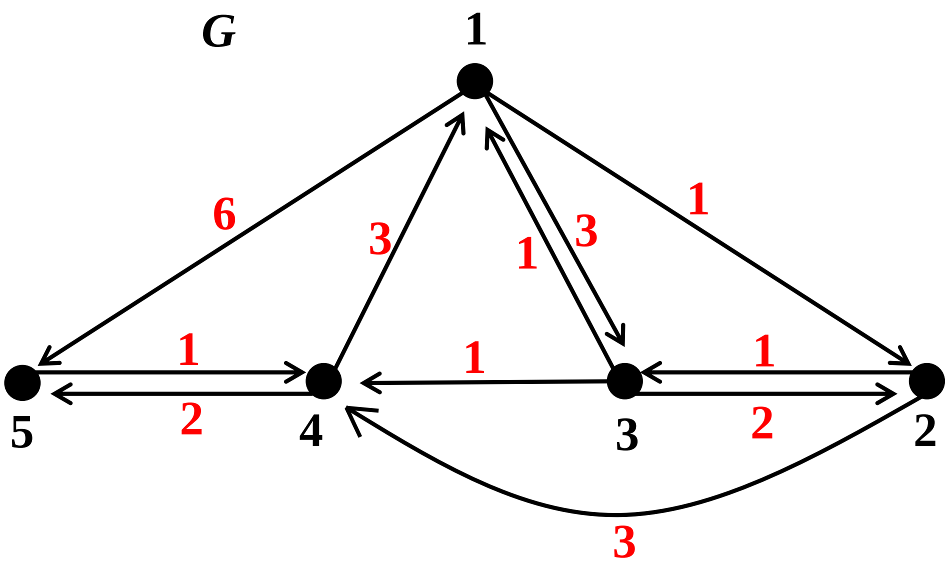
para cada j = 1, 2, …, n *faça*

$$\boldsymbol{D^k(i, j) \leftarrow \min \{ D^{k-1}(i, j) \,,\; D^{k-1}(i, k) + D^{k-1}(k, j) \}}$$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) \,,\; D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

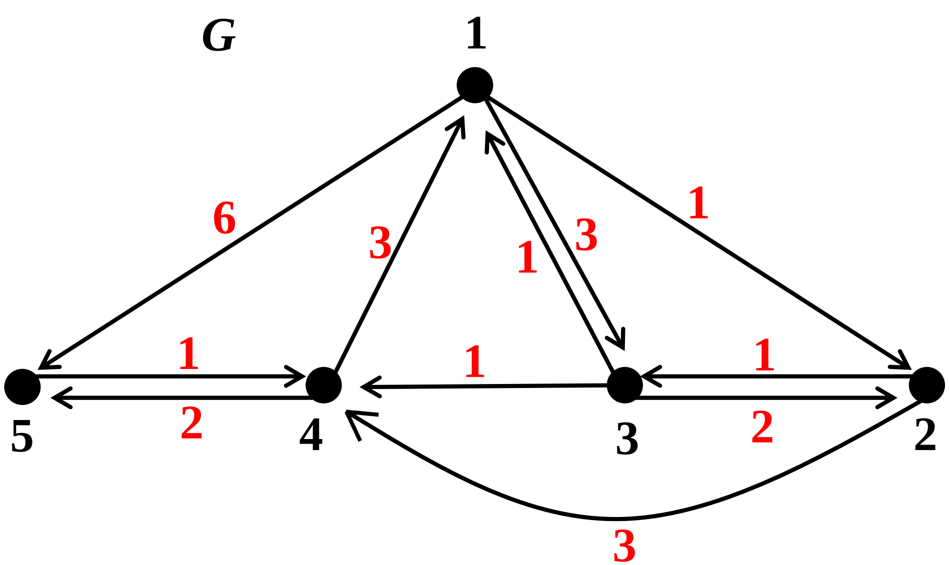
D^1

	1	2	3	4	5
1					
2					
3					
4					
5					

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i,j) = \min \{ D^{k-1}(i,j) \, , \; D^{k-1}(i,k) + D^{k-1}(k,j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0				
2					
3					
4					
5					

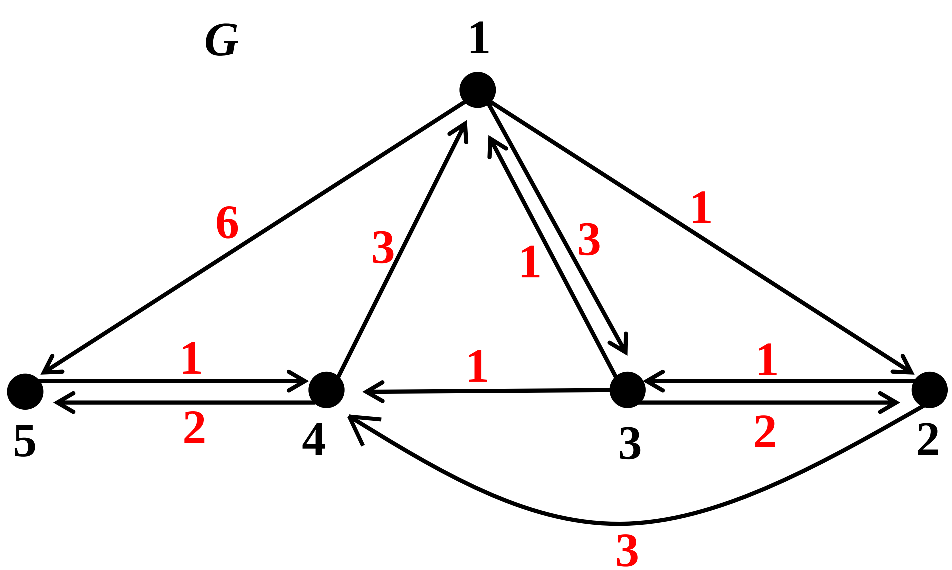
$$D^1(1,1) = \min \{ D^0(1,1) \, , \; D^0(1,1) + D^0(1,1) \}$$

$\boxed{0}$ $\boxed{0}$ $\boxed{0}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de $\boldsymbol{D^k}$ a partir de $\boldsymbol{D^{k-1}}$

$$D^k(i, j) = \min \{ D^{k-1}(i, j) , \ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1			
2					
3					
4					
5					

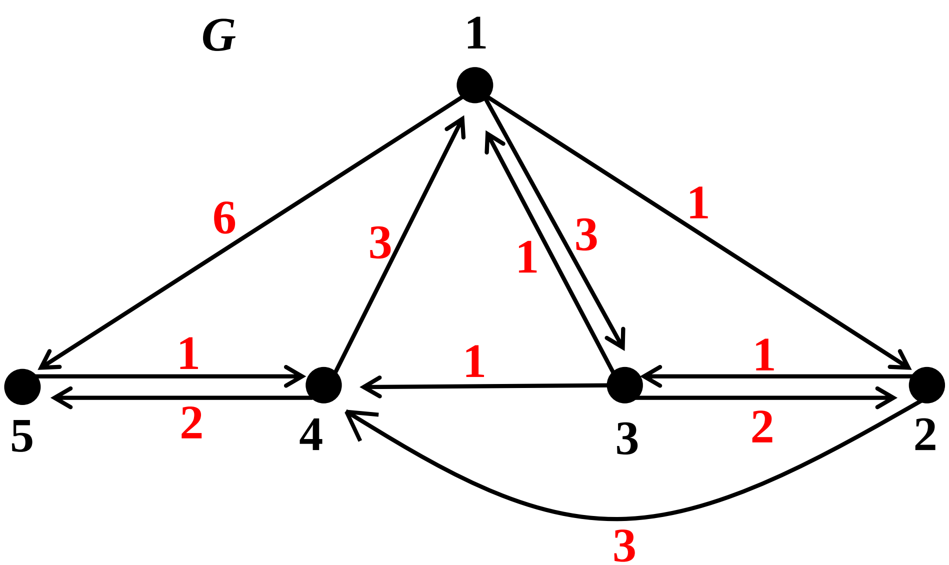
$$D^1(1, 2) = \min \{ D^0(1, 2) , \ D^0(1, 1) + D^0(1, 2) \}$$

$\boxed{1}$ $\boxed{0}$ $\boxed{1}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) ,\ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3		
2					
3					
4					
5					

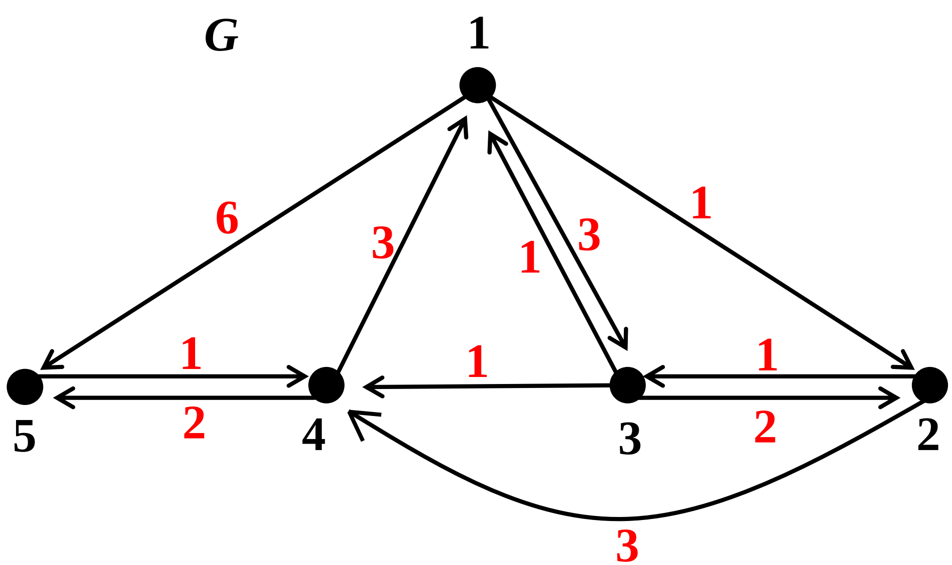
$$D^1(1, 3) = \min \{ D^0(1, 3) ,\ D^0(1, 1) + D^0(1, 3) \}$$

$\boxed{3}$ $\boxed{0}$ $\boxed{3}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de $\boldsymbol{D^k}$ a partir de $\boldsymbol{D^{k\text{-}1}}$

$$D^k(i, j) = \min \{ D^{k-1}(i, j) ,\ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	
2					
3					
4					
5					

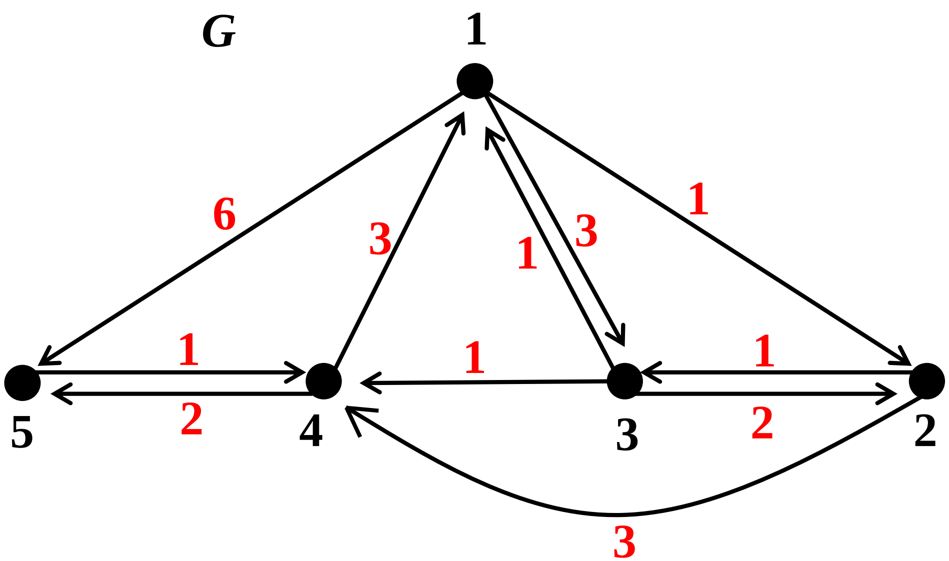
$$D^1(1, 4) = \min \{ D^0(1, 4) ,\ D^0(1, 1) + D^0(1, 4) \}$$

$\boxed{\infty}$ $\boxed{0}$ $\boxed{\infty}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) , D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

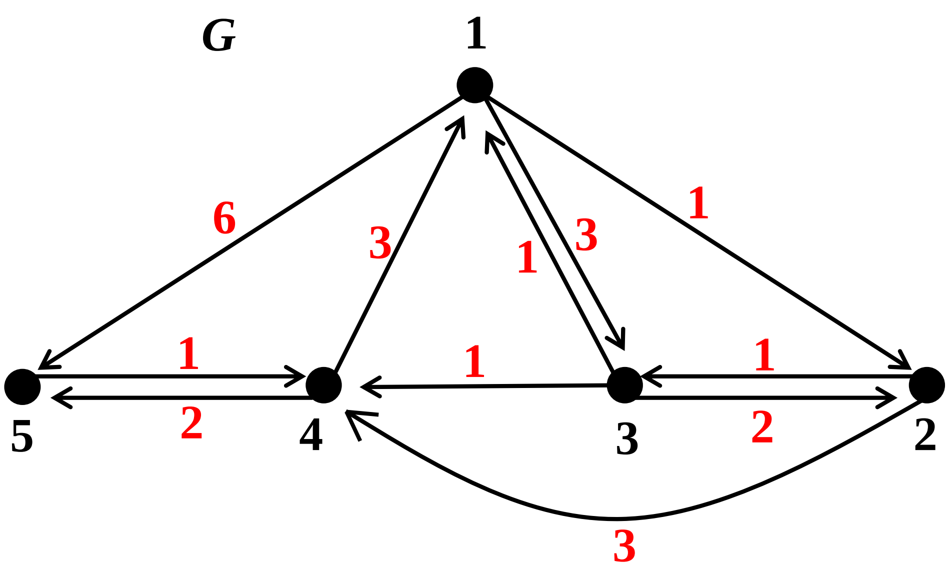
	1	2	3	4	5
1	0	1	3	∞	6
2					
3					
4					
5					

$$D^1(1, 5) = \min \{ \underbrace{D^0(1, 5)}_{\boxed{6}} , \underbrace{D^0(1, 1)}_{\boxed{0}} + \underbrace{D^0(1, 5)}_{\boxed{6}} \}$$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) ,\ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞				
3					
4					
5					

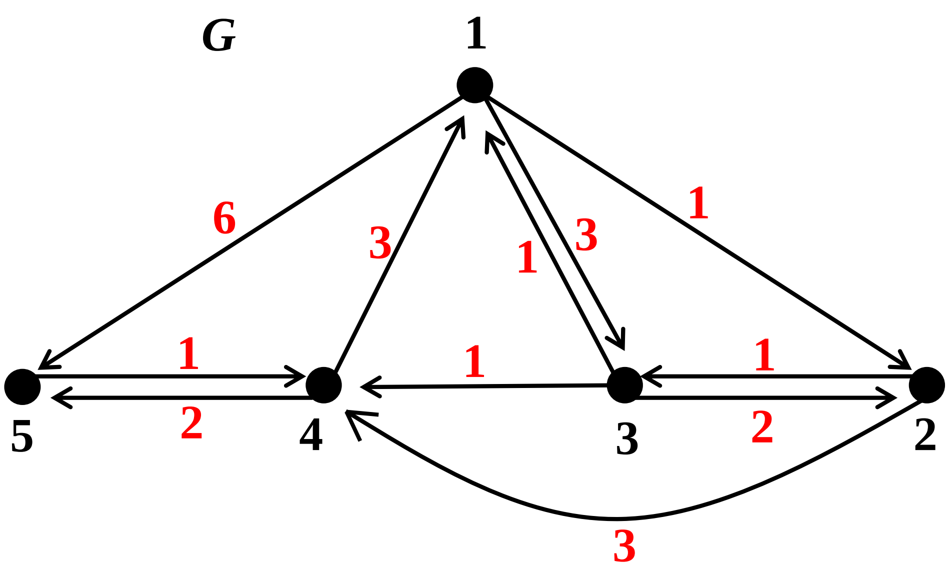
$$D^1(2, 1) = \min \{ D^0(2, 1) ,\ D^0(2, 1) + D^0(1, 1) \}$$

$\boxed{\infty}$ $\boxed{\infty}$ $\boxed{0}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i,j) = \min \{ D^{k-1}(i,j) \,,\; D^{k-1}(i,k) + D^{k-1}(k,j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0			
3					
4					
5					

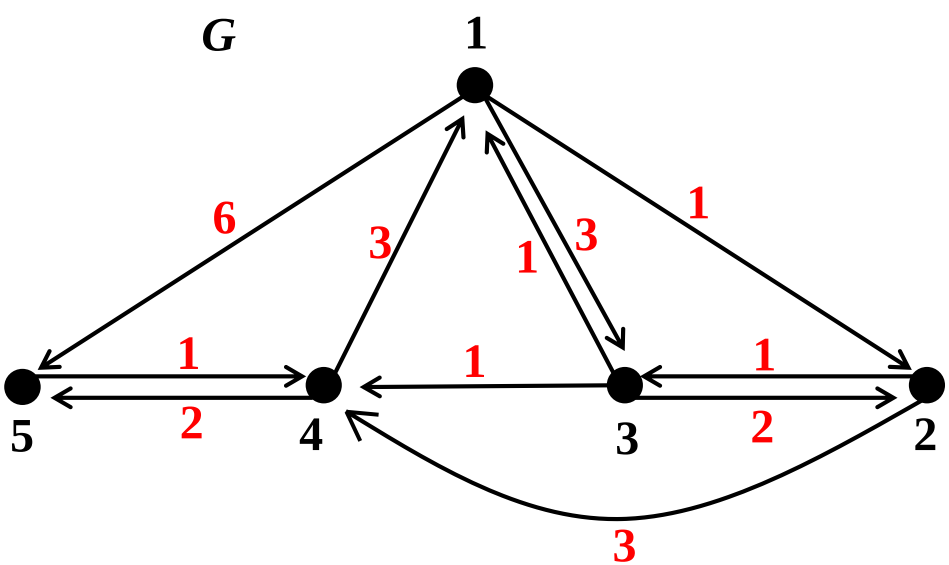
$$D^1(2,2) = \min \{ D^0(2,2) \,,\; D^0(2,1) + D^0(1,2) \}$$

$\boxed{0}$ $\boxed{\infty}$ $\boxed{1}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de $\boldsymbol{D^k}$ a partir de $\boldsymbol{D^{k-1}}$

$$D^k(i, j) = \min \{ D^{k-1}(i, j) , \; D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1		
3					
4					
5					

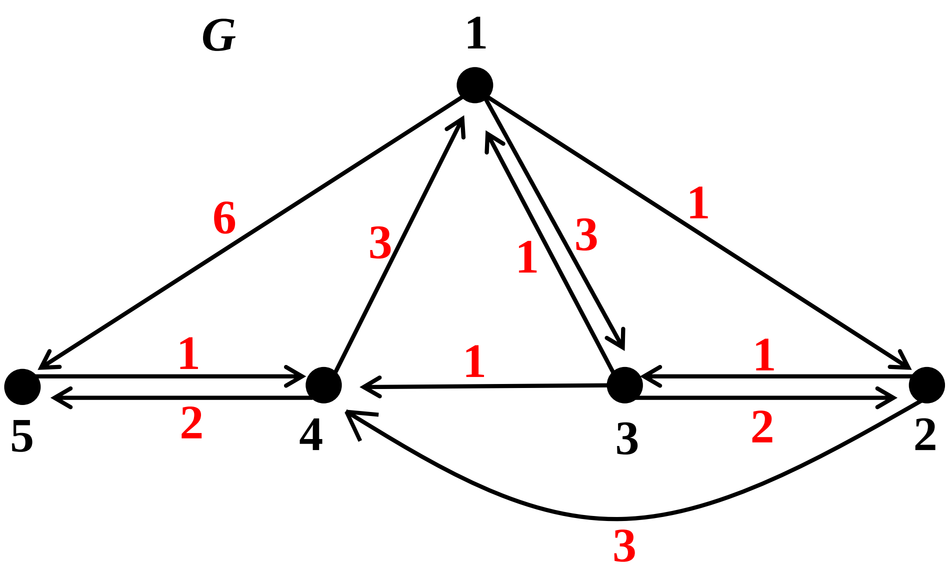
$$D^1(2, 3) = \min \{ D^0(2, 3) , \; D^0(2, 1) + D^0(1, 3) \}$$

$\boxed{1}$ $\boxed{\infty}$ $\boxed{3}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) \,,\ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	
3					
4					
5					

$$D^1(2, 4) = \min \{ D^0(2, 4) \,,\ D^0(2, 1) + D^0(1, 4) \}$$

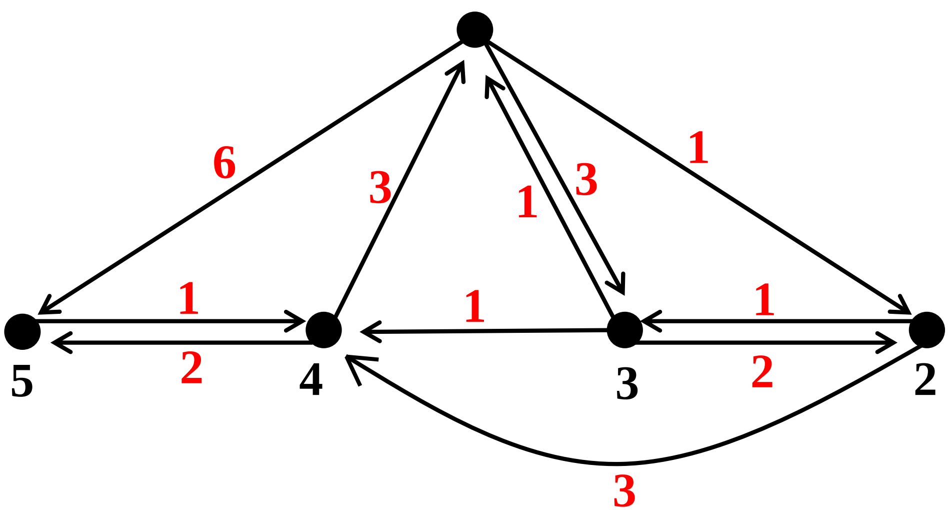
$\boxed{3}$ $\boxed{\infty}$ $\boxed{\infty}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de $\boldsymbol{D^k}$ a partir de $\boldsymbol{D^{k-1}}$

$$D^k(i,j) = \min\{\, D^{k-1}(i,j)\,,\; D^{k-1}(i,k) + D^{k-1}(k,j)\,\}$$

G



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3					
4					
5					

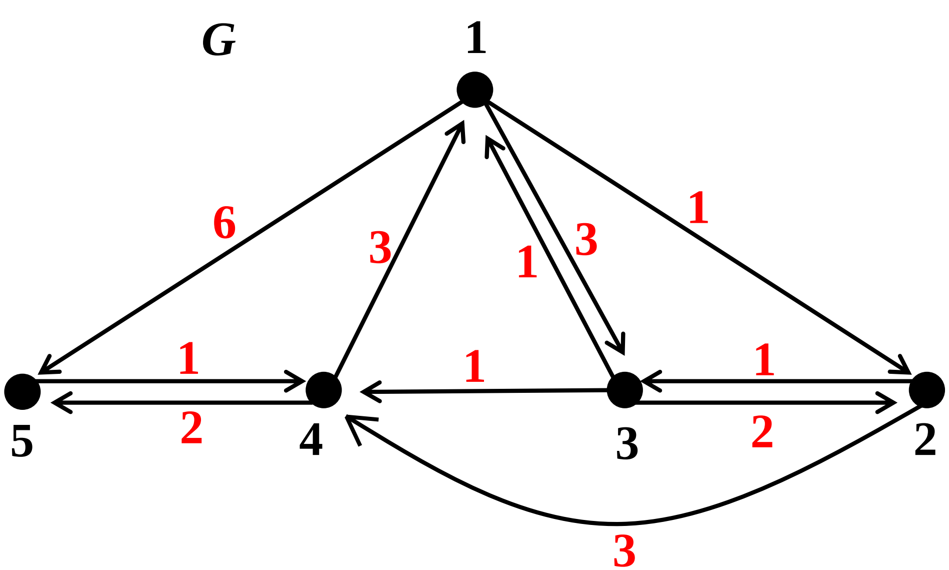
$$D^1(2,5) = \min\{\, D^0(2,5)\,,\; D^0(2,1) + D^0(1,5)\,\}$$

$\boxed{\infty}$ $\boxed{\infty}$ $\boxed{6}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) \,,\; D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1				
4					
5					

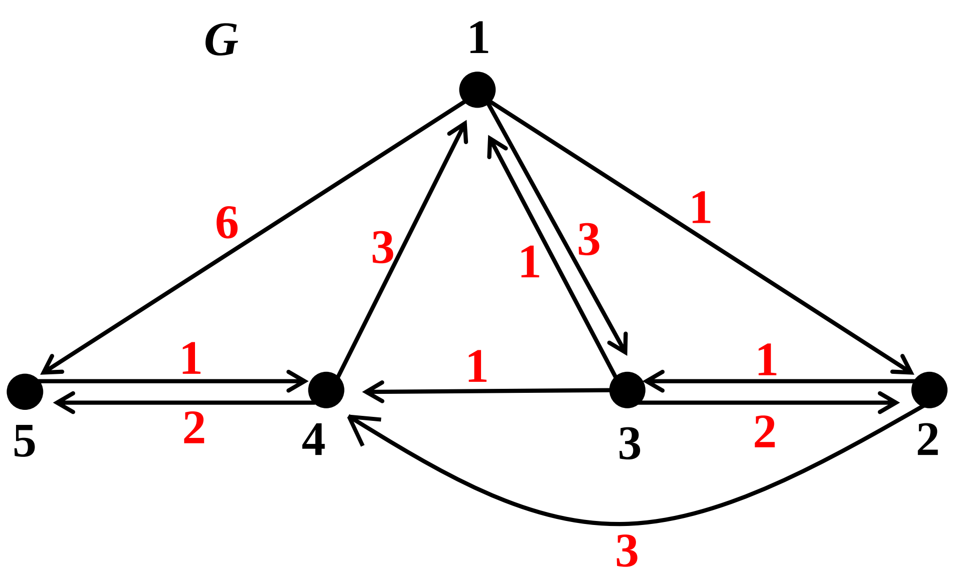
$$D^1(3, 1) = \min \{ D^0(3, 1) \,,\; D^0(3, 1) + D^0(1, 1) \}$$

$\boxed{1}$ $\boxed{1}$ $\boxed{0}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) , D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

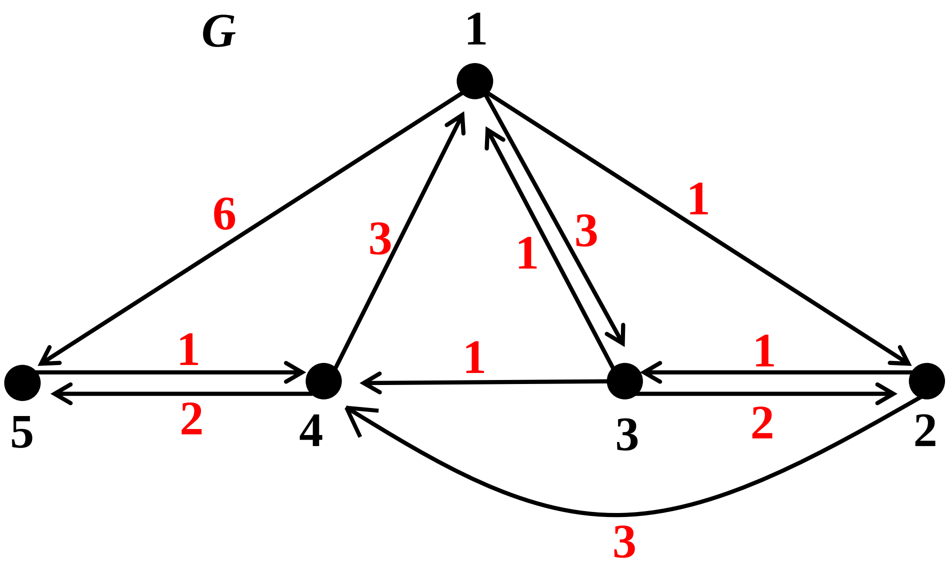
	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2			
4					
5					

$$D^1(3, 2) = \min \{ \underbrace{D^0(3, 2)}_{\boxed{2}} , \underbrace{D^0(3, 1)}_{\boxed{1}} + \underbrace{D^0(1, 2)}_{\boxed{1}} \}$$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) , \; D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0		
4					
5					

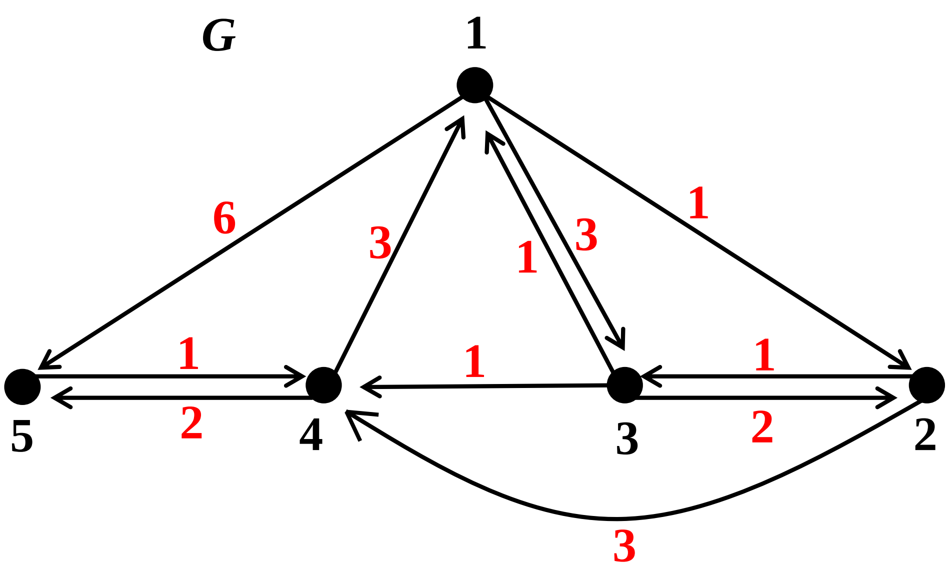
$$D^1(3, 3) = \min \{ D^0(3, 3) , \; D^0(3, 1) + D^0(1, 3) \}$$

$\boxed{0}$ $\boxed{1}$ $\boxed{3}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) \, , \; D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	
4					
5					

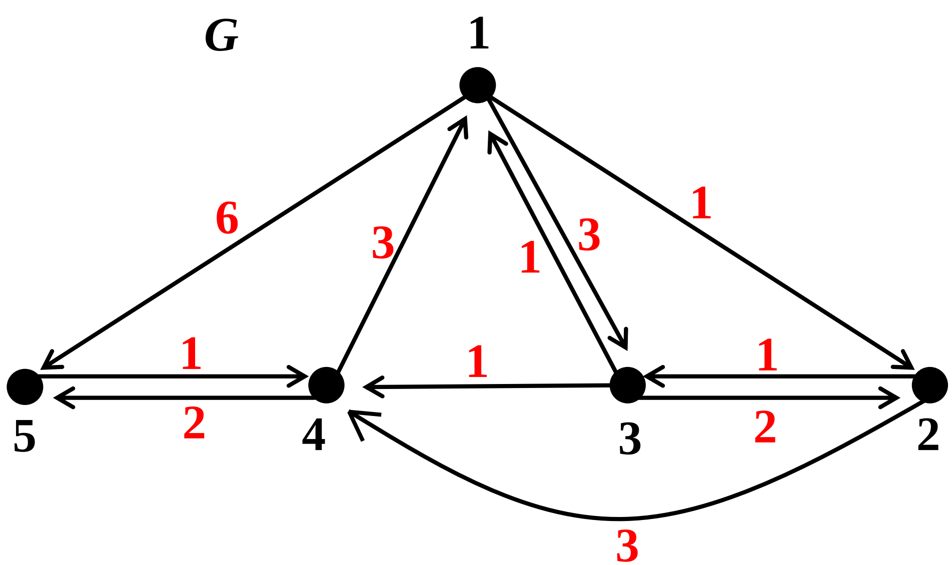
$$D^1(3, 4) = \min \{ D^0(3, 4) \, , \; D^0(3, 1) + D^0(1, 4) \}$$

$\boxed{1}$ $\boxed{1}$ $\boxed{\infty}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) ,\ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

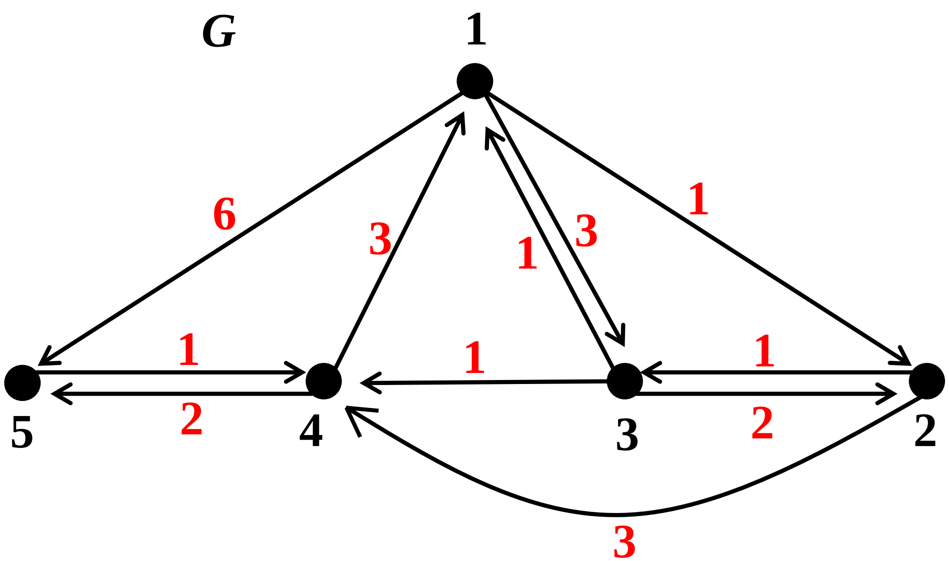
	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4					
5					

$$D^1(3, 5) = \min \{ \underbrace{D^0(3, 5)}_{\boxed{\infty}} ,\ \underbrace{D^0(3, 1)}_{\boxed{1}} + \underbrace{D^0(1, 5)}_{\boxed{6}} \}$$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) \,,\; D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

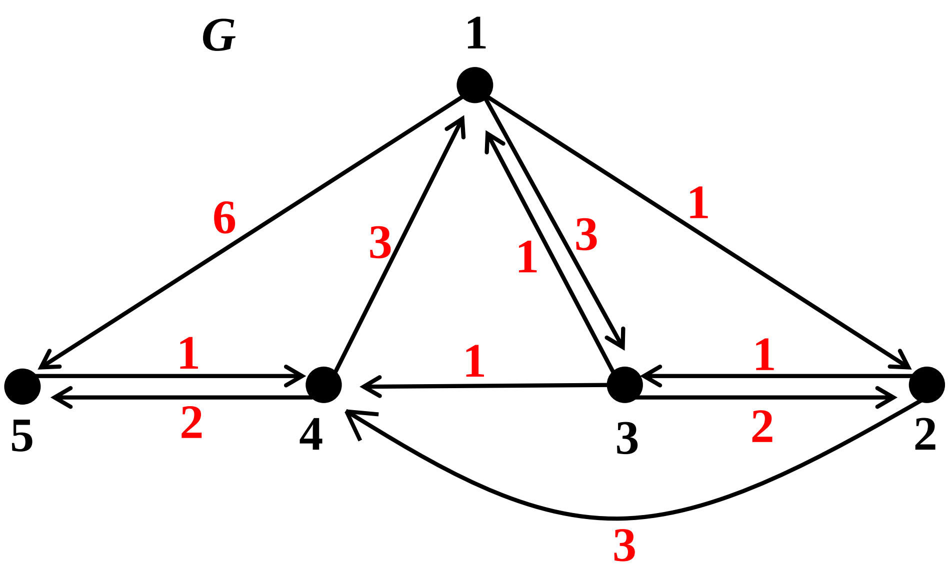
	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3				
5					

$$D^1(4, 1) = \min \{ \underbrace{D^0(4, 1)}_{\boxed{3}} \,,\; \underbrace{D^0(4, 1)}_{\boxed{3}} + \underbrace{D^0(1, 1)}_{\boxed{0}} \}$$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) ,\ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3	4			
5					

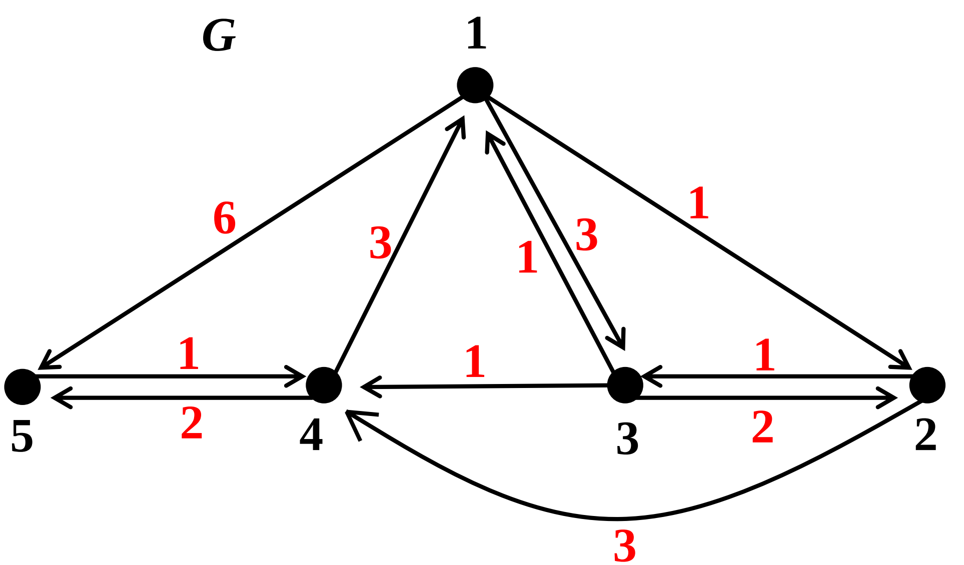
$$D^1(4, 2) = \min \{ D^0(4, 2) ,\ D^0(4, 1) + D^0(1, 2) \}$$

$\boxed{\infty}$ $\boxed{3}$ $\boxed{1}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de $\boldsymbol{D^k}$ a partir de $\boldsymbol{D^{k-1}}$

$$D^k(i, j) = \min \{\, D^{k-1}(i, j)\,,\; D^{k-1}(i, k) + D^{k-1}(k, j) \,\}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3	4	6		
5					

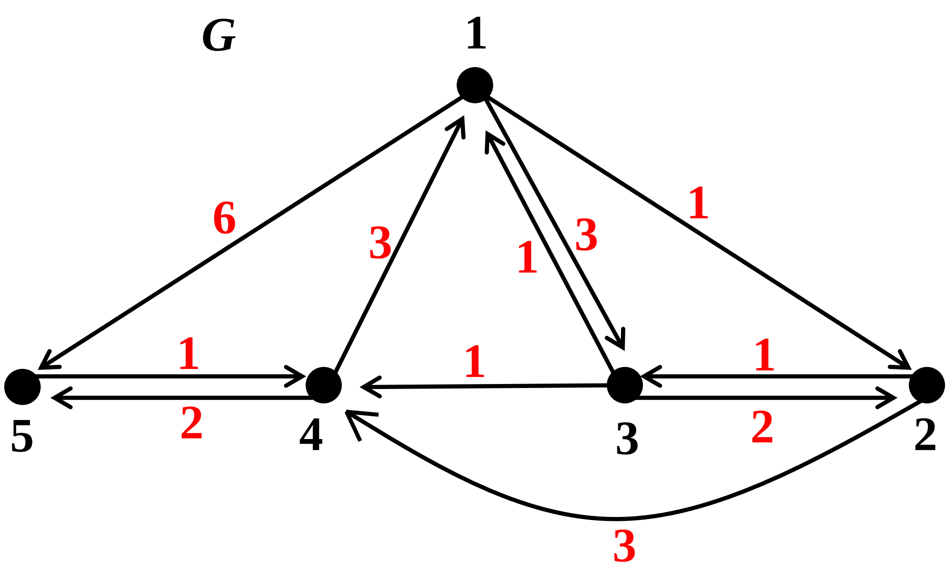
$$D^1(4, 3) = \min \{\, D^0(4, 3)\,,\; D^0(4, 1) + D^0(1, 3) \,\}$$

$\boxed{\infty}$ $\boxed{3}$ $\boxed{3}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) ,\ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3	4	6	0	
5					

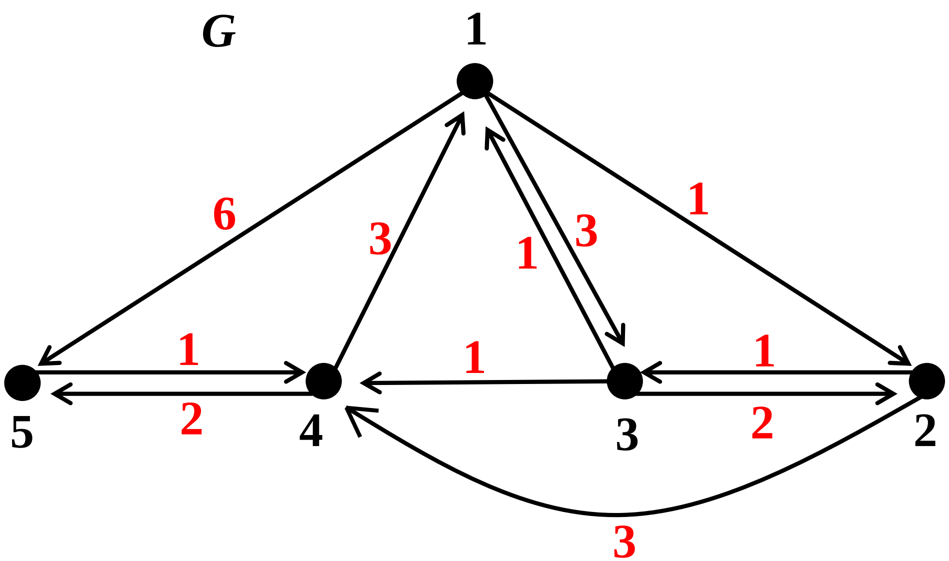
$$D^1(4, 4) = \min \{ D^0(4, 4) ,\ D^0(4, 1) + D^0(1, 4) \}$$

0 3 ∞

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) , \ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3	4	6	0	2
5					

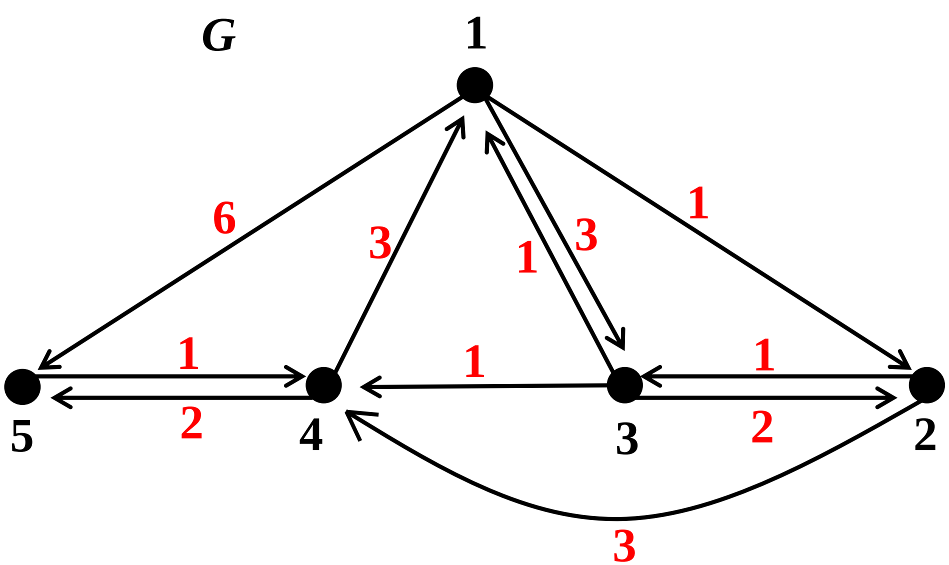
$$D^1(4, 5) = \min \{ D^0(4, 5) , \ D^0(4, 1) + D^0(1, 5) \}$$

$\boxed{2}$ $\boxed{3}$ $\boxed{6}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) ,\ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3	4	6	0	2
5	∞				

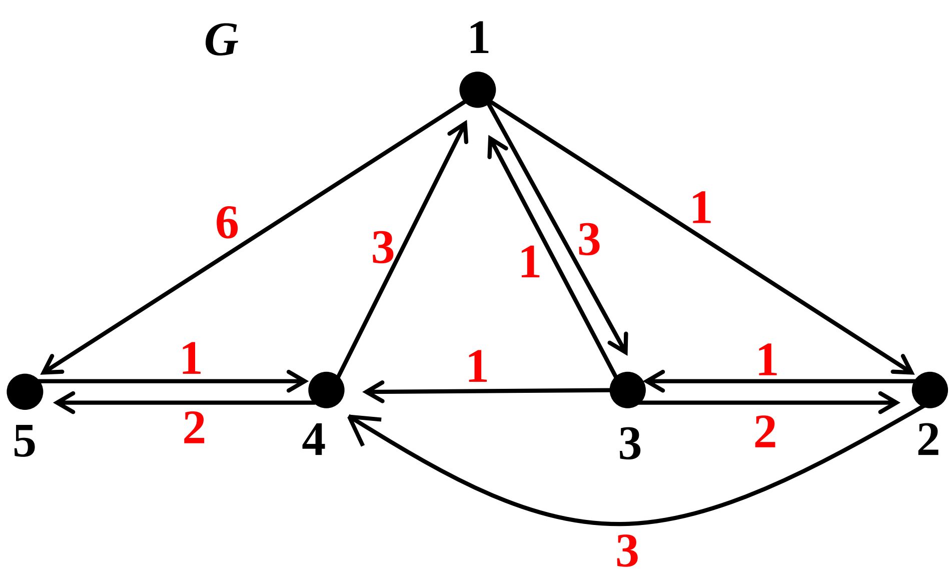
$$D^1(5, 1) = \min \{ D^0(5, 1) ,\ D^0(5, 1) + D^0(1, 1) \}$$

$\boxed{\infty}$ $\boxed{\infty}$ $\boxed{0}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{\, D^{k-1}(i, j)\ ,\ D^{k-1}(i, k) + D^{k-1}(k, j) \,\}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3	4	6	0	2
5	∞	∞			

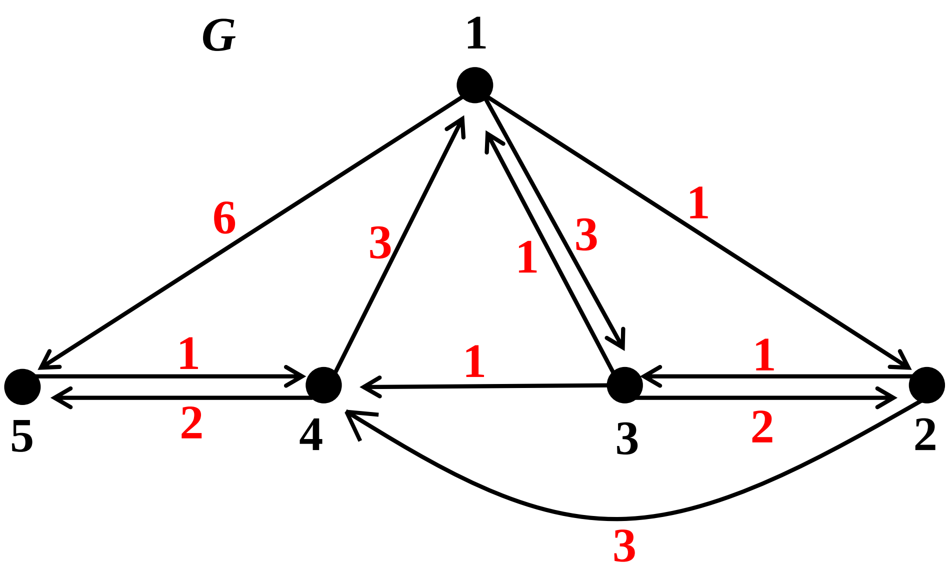
$$D^1(5, 2) = \min \{\, D^0(5, 2)\ ,\ D^0(5, 1) + D^0(1, 2) \,\}$$

$\boxed{\infty}$ $\boxed{\infty}$ $\boxed{1}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i,j) = \min\{\, D^{k-1}(i,j)\,,\ D^{k-1}(i,k) + D^{k-1}(k,j)\,\}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3	4	6	0	2
5	∞	∞	∞	1	0

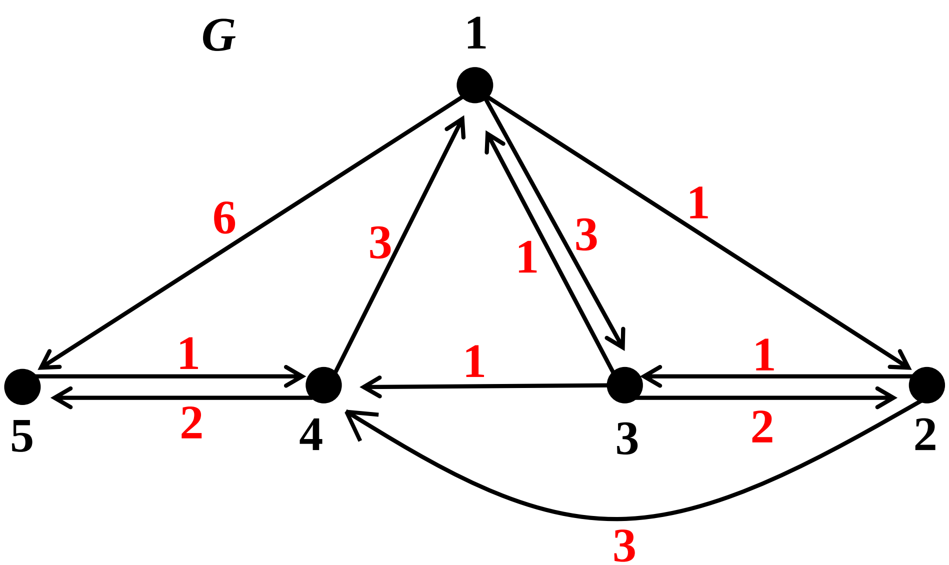
$$D^1(5,3) = \min\{\, D^0(5,3)\,,\ D^0(5,1) + D^0(1,3)\,\}$$

$\boxed{\infty}$ $\boxed{\infty}$ $\boxed{3}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) ,\ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3	4	6	0	2
5	∞	∞	∞	1	

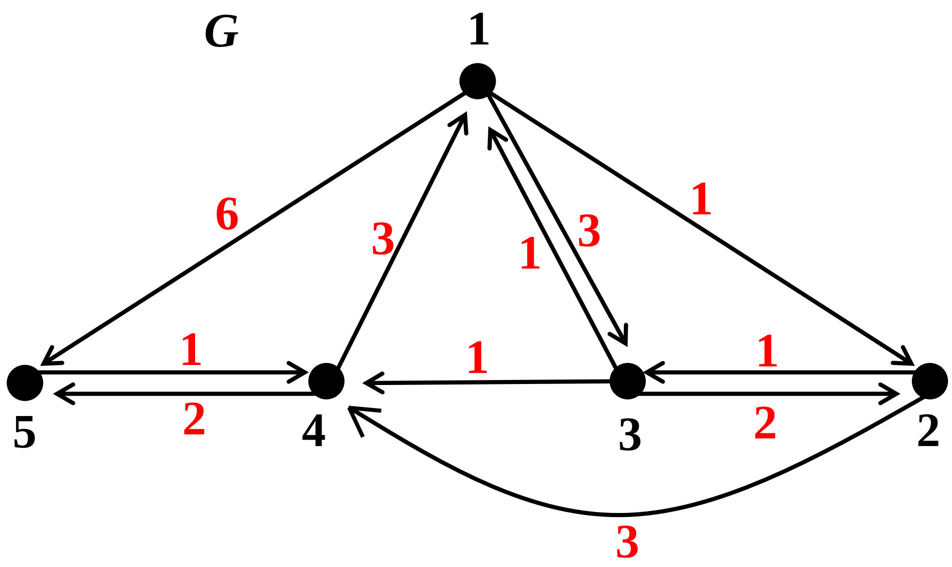
$$D^1(5, 4) = \min \{ D^0(5, 4) ,\ D^0(5, 1) + D^0(1, 4) \}$$

$\boxed{1}$ $\boxed{\infty}$ $\boxed{\infty}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i, j) = \min \{ D^{k-1}(i, j) ,\ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3	4	6	0	2
5	∞	∞	∞	1	0

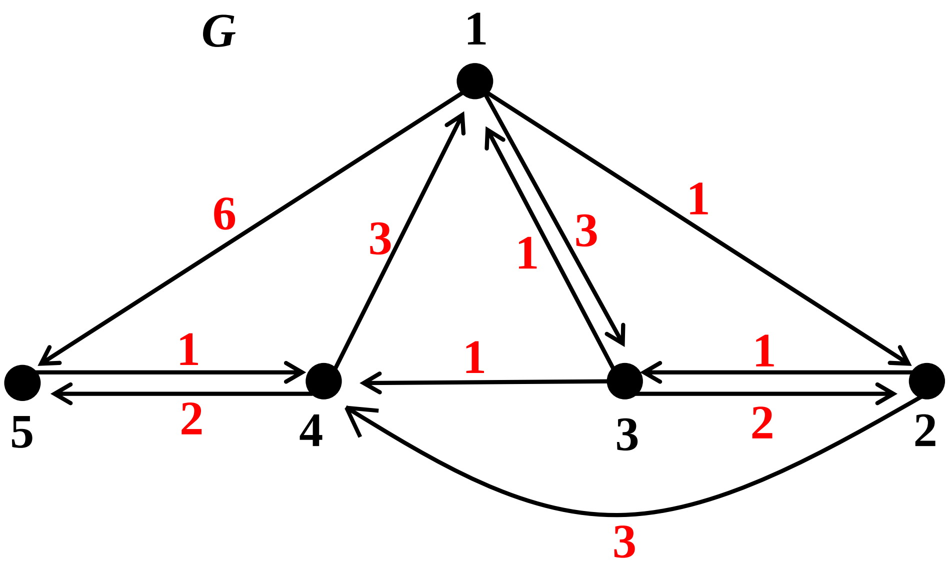
$$D^1(5, 5) = \min \{ D^0(5, 5) ,\ D^0(5, 1) + D^0(1, 5) \}$$

$\boxed{0}$ $\boxed{\infty}$ $\boxed{6}$

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de $\boldsymbol{D^k}$ a partir de $\boldsymbol{D^{k-1}}$

$$D^k(i, j) = \min \{\, D^{k-1}(i, j)\ ,\ D^{k-1}(i, k) + D^{k-1}(k, j) \,\}$$



D^0

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	∞
4	3	∞	∞	0	2
5	∞	∞	∞	1	0

D^1

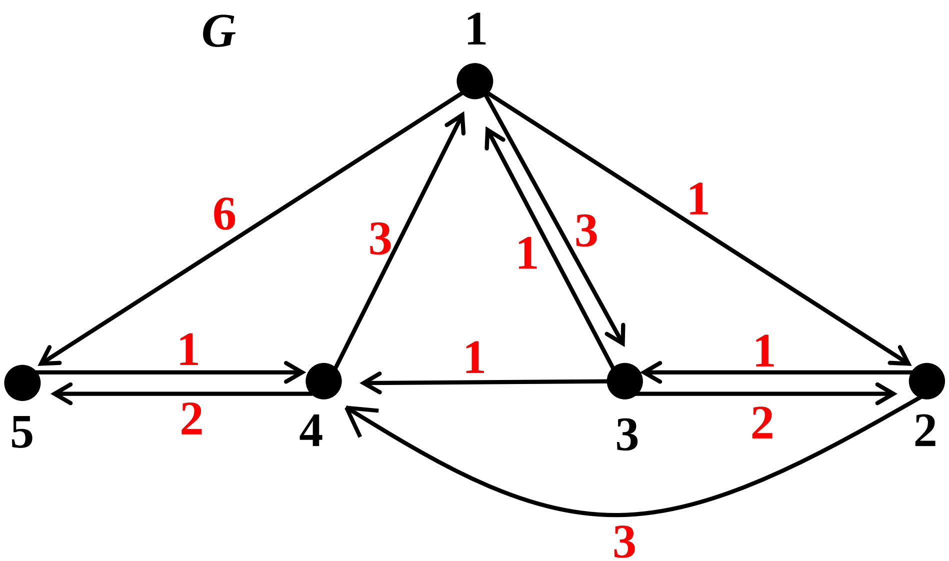
	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3	4	6	0	2
5	∞	∞	∞	1	0

valores que “melhoraram”
considerando 1
como vértice intermediário

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i,j) = \min \{ D^{k-1}(i,j) \, , \; D^{k-1}(i,k) + D^{k-1}(k,j) \}$$



D^1

	1	2	3	4	5
1	0	1	3	∞	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3	4	6	0	2
5	∞	∞	∞	1	0

D^2

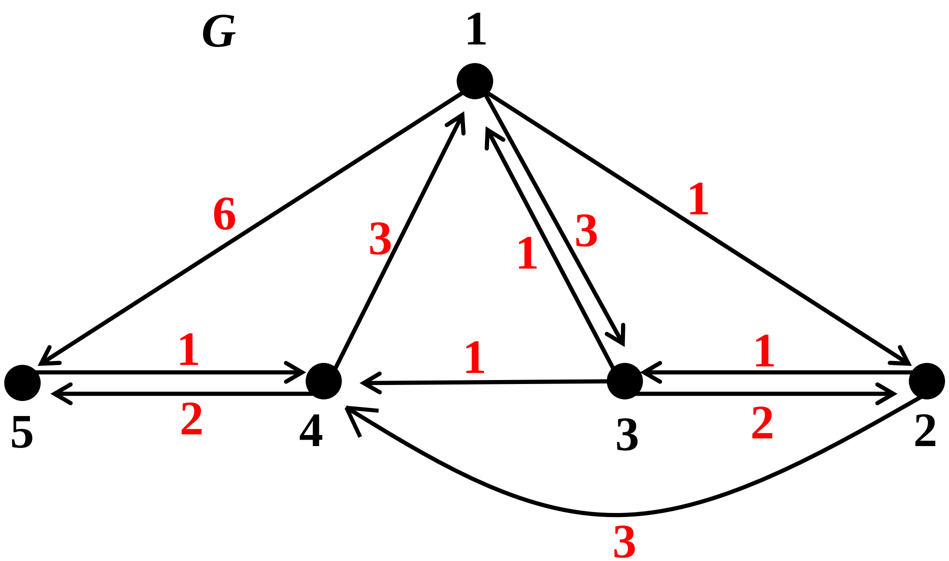
	1	2	3	4	5
1	0	1	2	4	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3	4	5	0	2
5	∞	∞	∞	1	0

valores que “melhoraram”
considerando 1 e 2
como vértices intermediários

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i,j) = \min\{\, D^{k-1}(i,j)\,,\ D^{k-1}(i,k) + D^{k-1}(k,j) \,\}$$



D^2

	1	2	3	4	5
1	0	1	2	4	6
2	∞	0	1	3	∞
3	1	2	0	1	7
4	3	4	5	0	2
5	∞	∞	∞	1	0

D^3

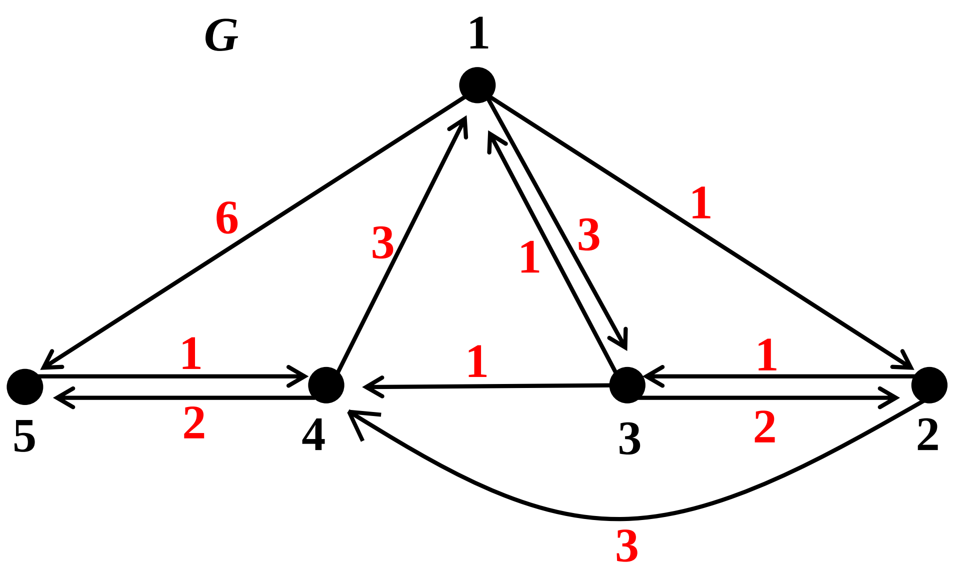
	1	2	3	4	5
1	0	1	2	3	6
2	2	0	1	2	∞
3	1	2	0	1	7
4	3	4	5	0	2
5	∞	∞	∞	1	0

valores que "melhoraram" considerando 1, 2, 3 como vértices intermediários

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i,j) = \min\{\, D^{k-1}(i,j)\,,\; D^{k-1}(i,k) + D^{k-1}(k,j) \,\}$$



D^3

	1	2	3	4	5
1	0	1	2	3	6
2	2	0	1	2	∞
3	1	2	0	1	7
4	3	4	5	0	2
5	∞	∞	∞	1	0

D^4

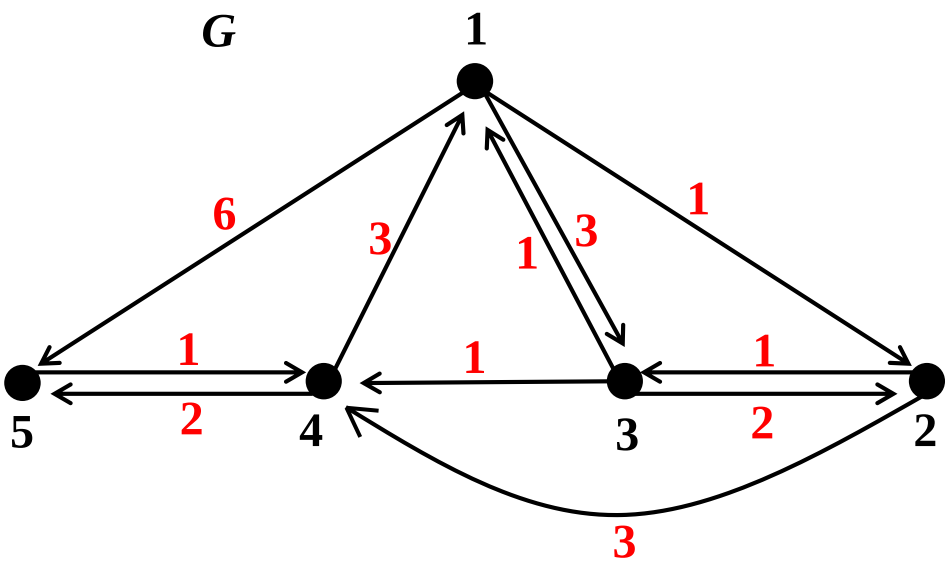
	1	2	3	4	5
1	0	1	2	3	5
2	2	0	1	2	4
3	1	2	0	1	3
4	3	4	5	0	2
5	4	5	6	1	0

valores que "melhoraram" considerando 1, 2, 3, 4 como vértices intermediários

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de D^k a partir de D^{k-1}

$$D^k(i,j) = \min \{ D^{k-1}(i,j) ,\ D^{k-1}(i,k) + D^{k-1}(k,j) \}$$



D^4

	1	2	3	4	5
1	0	1	2	3	5
2	2	0	1	2	4
3	1	2	0	1	3
4	3	4	5	0	2
5	4	5	6	1	0

nenhum valor melhorou
considerando 1, 2, 3, 4, 5
como vértices intermediários

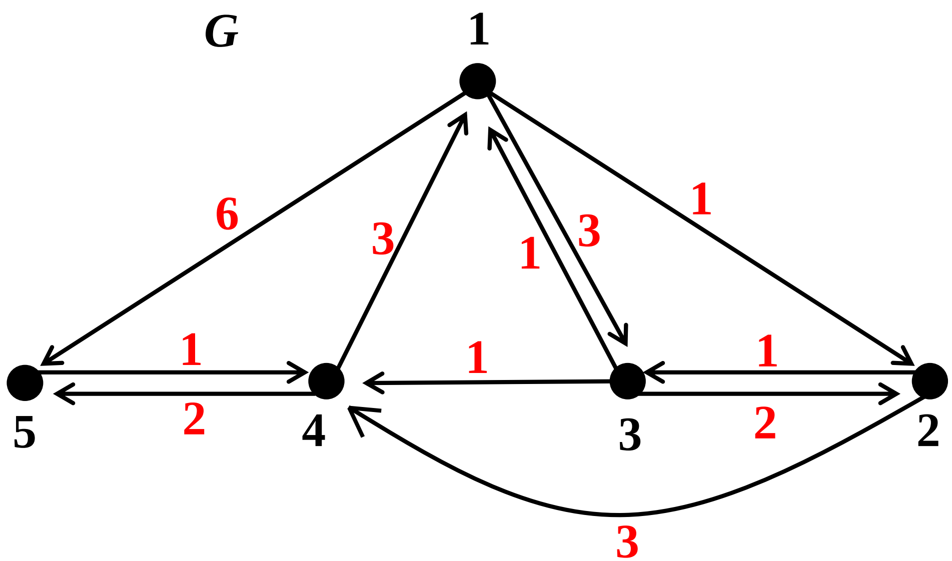
D^5

	1	2	3	4	5
1	0	1	2	3	5
2	2	0	1	2	4
3	1	2	0	1	3
4	3	4	5	0	2
5	4	5	6	1	0

Algoritmo de Floyd-Warshall

Simulação: visualizar o cálculo de $\boldsymbol{D^k}$ a partir de $\boldsymbol{D^{k-1}}$

$$D^k(i, j) = \min \{ D^{k-1}(i, j) ,\ D^{k-1}(i, k) + D^{k-1}(k, j) \}$$



D^4

	1	2	3	4	5
1	0	1	2	3	5
2	2	0	1	2	4
3	1	2	0	1	3
4	3	4	5	0	2
5	4	5	6	1	0

D^5

	1	2	3	4	5
1	0	1	2	3	5
2	2	0	1	2	4
3	1	2	0	1	3
4	3	4	5	0	2
5	4	5	6	1	0

MATRIZ FINAL DE CUSTOS DOS CAMINHOS MÍNIMOS

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos ...)

Solução: Determinar os valores $P^k(i, j)$

$P^k(i, j)$ = penúltimo vértice do caminho mínimo de i a j cujos vértices intermediários estão no conjunto $\{1, 2,\ldots, k\} \setminus \{i, j\}$.

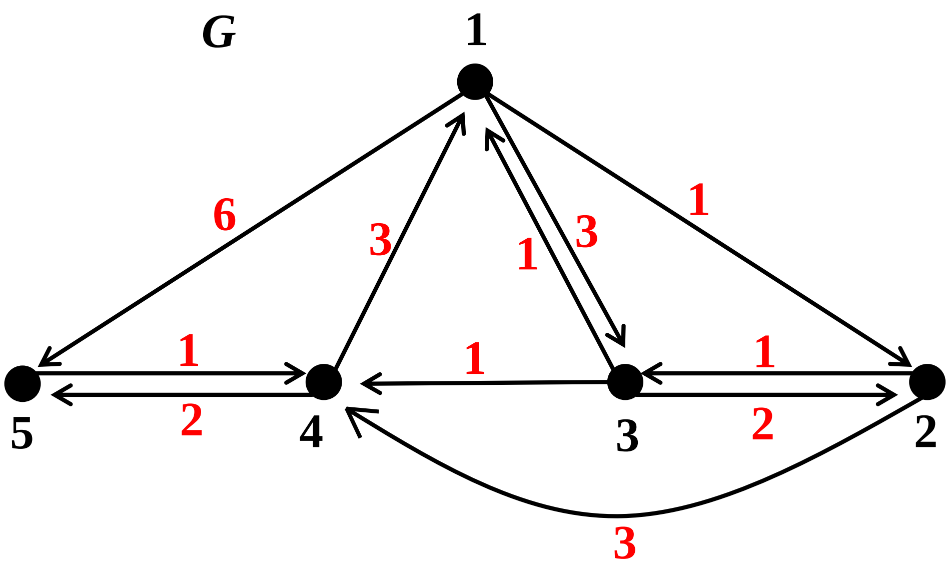
Observe que:

$$P^0(i, j) = \begin{cases} i, & \text{se existe aresta direcionada de } \mathbf{i} \text{ para } \mathbf{j} \\ \textbf{null}, & \text{caso contrário} \end{cases}$$

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

***Definindo as matrizes do auxiliares do algoritmo*:**
Para cada $k = 0, 1, 2, \ldots, n$, defina $\boldsymbol{P^k}$ como a matriz contendo todos os valores $\boldsymbol{P^k(i, j)}$



P^0

	1	2	3	4	5
1	*null*	1	1	*null*	1
2	*null*	*null*	2	2	*null*
3	3	3	*null*	3	*null*
4	4	*null*	*null*	*null*	4
5	*null*	*null*	*null*	5	*null*

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

Como calcular $P^k(i, j)$?

- Sabemos que:

$$D^k(i, j) = \min \{ D^{k-1}(i, j) \, , \; D^{k-1}(i, k) + D^{k-1}(k, j) \}$$

- Se $D^{k-1}(i, j) \leq D^{k-1}(i, k) + D^{k-1}(k, j)$

  então $P^k(i, j) = P^{k-1}(i, j)$ ($\longrightarrow$ mesmo vértice da iteração anterior)

  senão $P^k(i, j) = P^{k-1}(k, j)$ ($\longrightarrow$ nova possibilidade de caminho)

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

Como calcular $P^k(i, j)$?

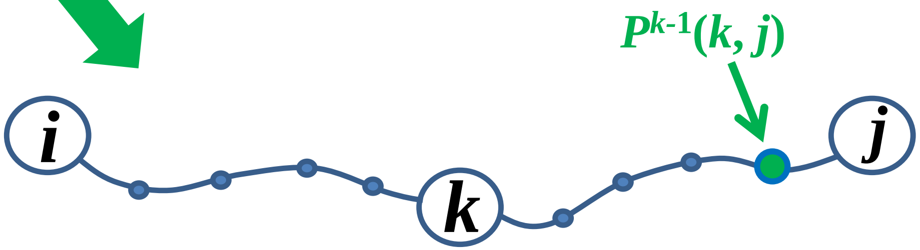
- Sabemos que:

$$\boxed{D^k(i, j) = \min \{ D^{k-1}(i, j) \ , \ D^{k-1}(i, k) + D^{k-1}(k, j) \}}$$

- Se $D^{k-1}(i, j) \leq D^{k-1}(i, k) + D^{k-1}(k, j)$

  então $P^k(i, j) = P^{k-1}(i, j)$

  senão $P^k(i, j) = P^{k-1}(k, j)$



Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

Novo algoritmo:

- *inicializar* $\boldsymbol{D^0}$ *e* $\boldsymbol{P^0}$
- *para cada* $k = 1, 2, \ldots, n$ *faça*

  para cada $i = 1, 2, \ldots, n$ *faça*

  para cada $j = 1, 2, \ldots, n$ *faça*

  se $\boldsymbol{D^{k-1}(i, j)} \leq \boldsymbol{D^{k-1}(i, k) + D^{k-1}(k, j)}$

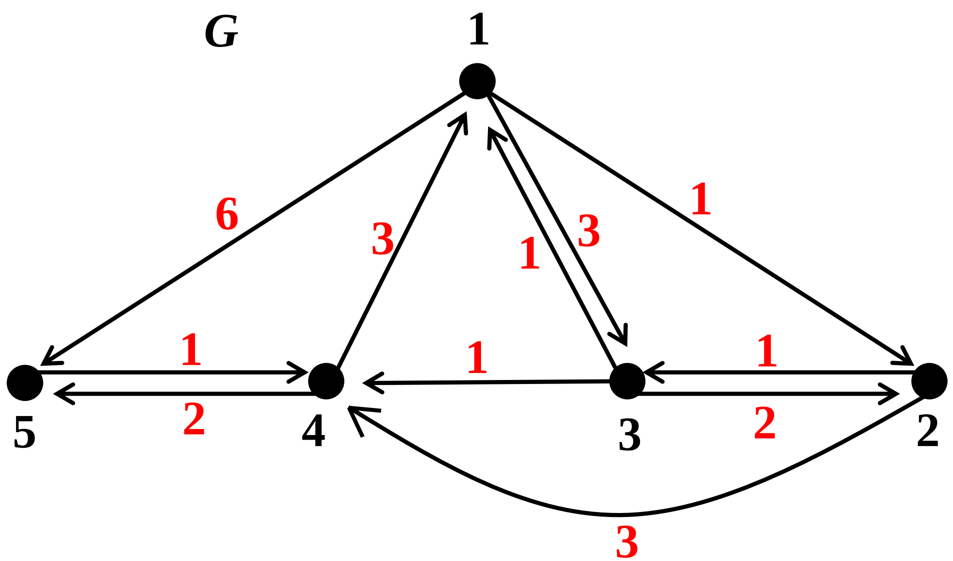
  então $\boldsymbol{D^k(i, j)} = \boldsymbol{D^{k-1}(i, j)}$ *e* $\boldsymbol{P^k(i, j)} = \boldsymbol{P^{k-1}(i, j)}$

  senão $\boldsymbol{D^k(i, j)} = \boldsymbol{D^{k-1}(i, k) + D^{k-1}(k, j)}$ *e*

  $\boldsymbol{P^k(i, j)} = \boldsymbol{P^{k-1}(k, j)}$

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)



***D*^{5}**

	1	2	3	4	5
1	0	1	2	3	5
2	2	0	1	2	4
3	1	2	0	1	3
4	3	4	5	0	2
5	4	5	6	1	0

***P*^{5}**

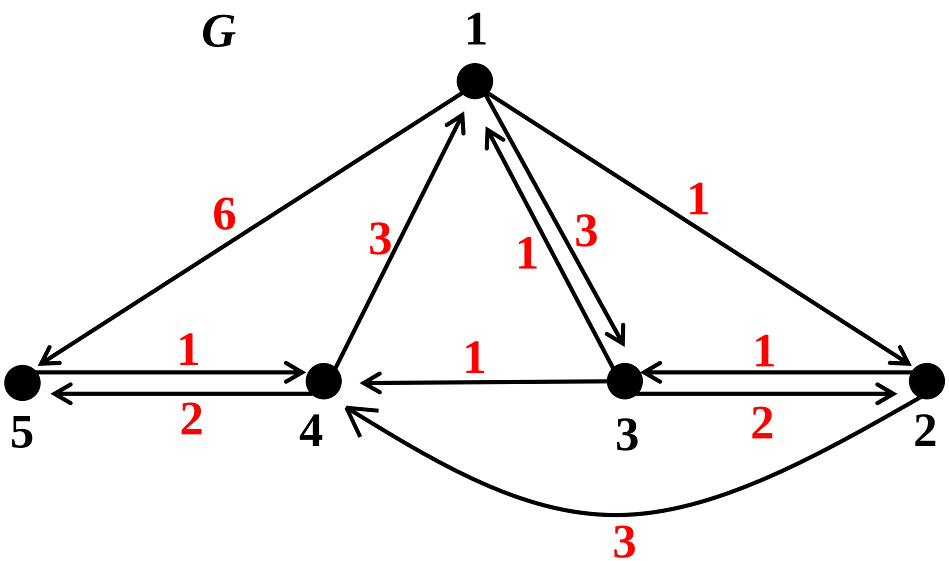
	1	2	3	4	5
1	*null*	1	2	3	4
2	3	*null*	2	3	4
3	3	3	*null*	3	4
4	4	1	2	*null*	4
5	4	1	2	5	*null*

MATRIZES FINAIS

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

Exemplo: calcular o caminho mínimo de 5 a 3



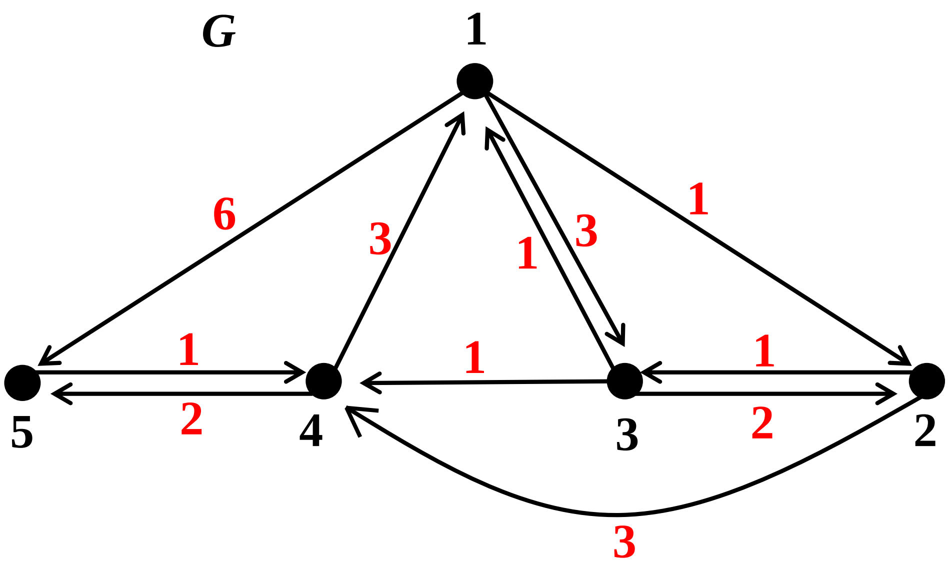
P^5

	1	2	3	4	5
1	*null*	1	2	3	4
2	3	*null*	2	3	4
3	3	3	*null*	3	4
4	4	1	2	*null*	4
5	4	1	2	5	*null*

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

Exemplo: calcular o caminho mínimo de 5 a 3



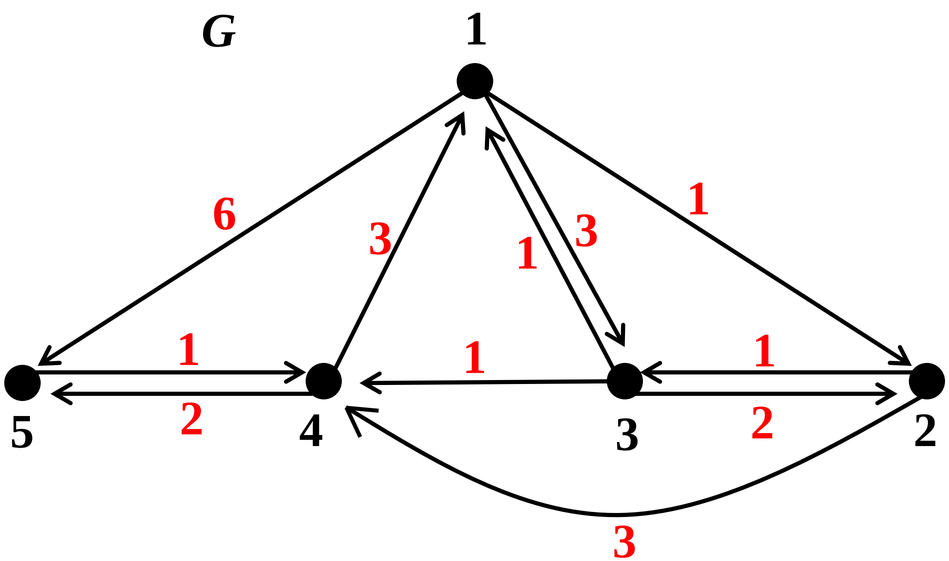
P^5

	1	2	3	4	5
1	*null*	1	2	3	4
2	3	*null*	2	3	4
3	3	3	*null*	3	4
4	4	1	2	*null*	4
5	4	1	2	5	*null*

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

Exemplo: calcular o caminho mínimo de 5 a 3



5

3

$$P^5(5, 3) = 2$$

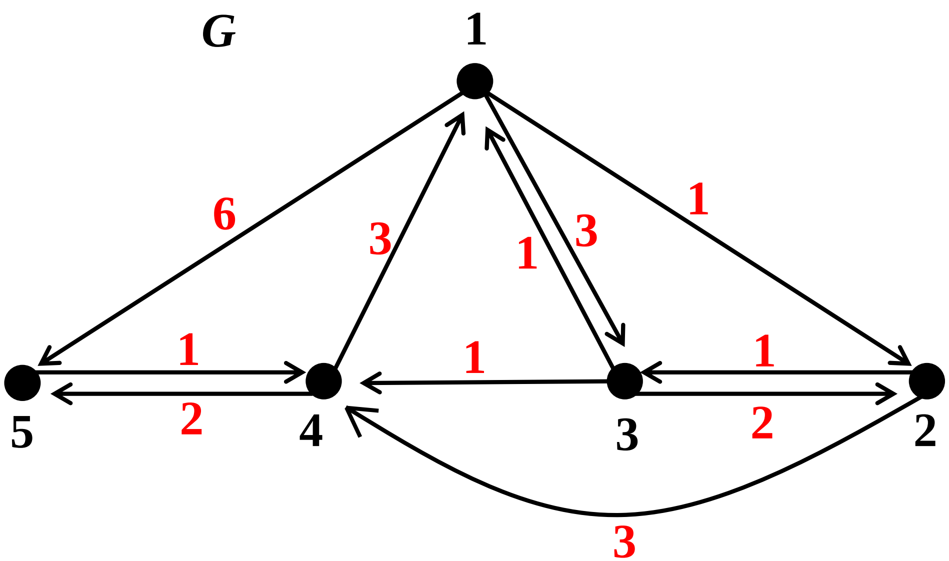
P^5

	1	**2**	**3**	**4**	**5**
1	*null*	1	2	3	4
2	3	*null*	2	3	4
3	3	3	*null*	3	4
4	4	1	2	*null*	4
5	4	1	2	5	*null*

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

Exemplo: calcular o caminho mínimo de 5 a 3



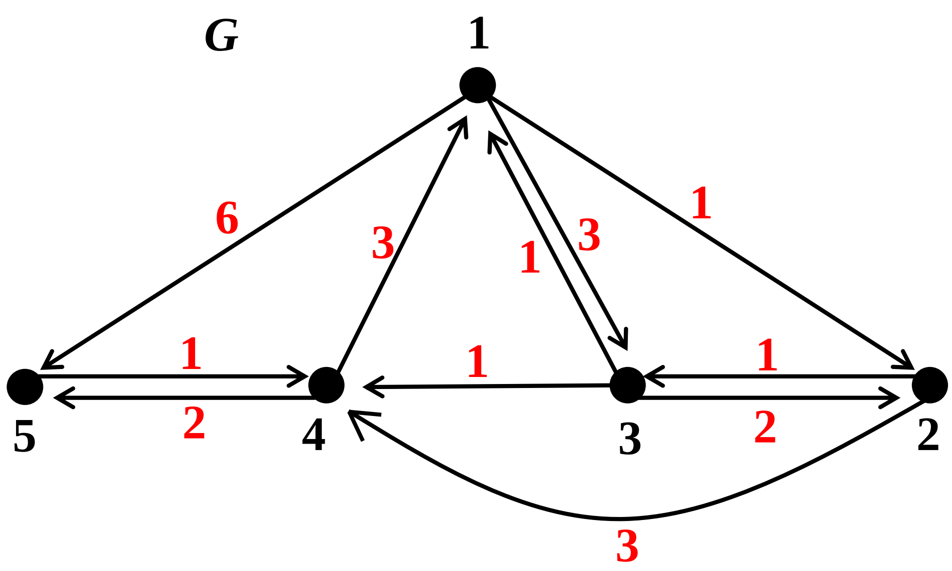
P^5

	1	2	3	4	5
1	*null*	1	2	3	4
2	3	*null*	2	3	4
3	3	3	*null*	3	4
4	4	1	2	*null*	4
5	4	1	2	5	*null*

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

Exemplo: calcular o caminho mínimo de 5 a 3



$$P^5(5, 2) = 1$$

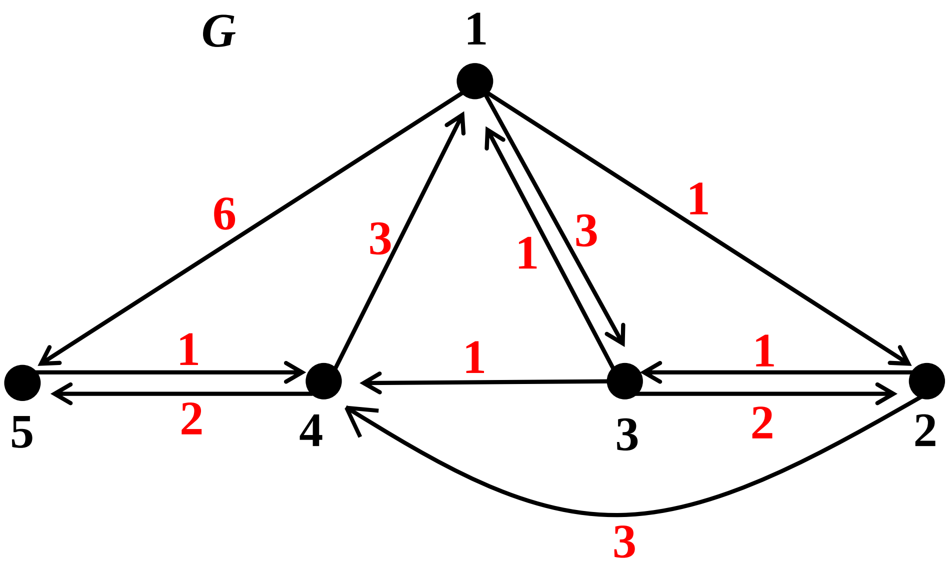
P^5

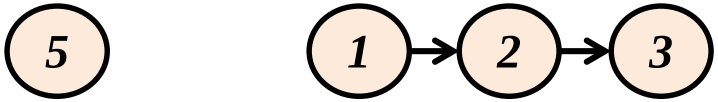
	1	2	3	4	5
1	*null*	1	2	3	4
2	3	*null*	2	3	4
3	3	3	*null*	3	4
4	4	1	2	*null*	4
5	4	1	2	5	*null*

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

Exemplo: calcular o caminho mínimo de 5 a 3





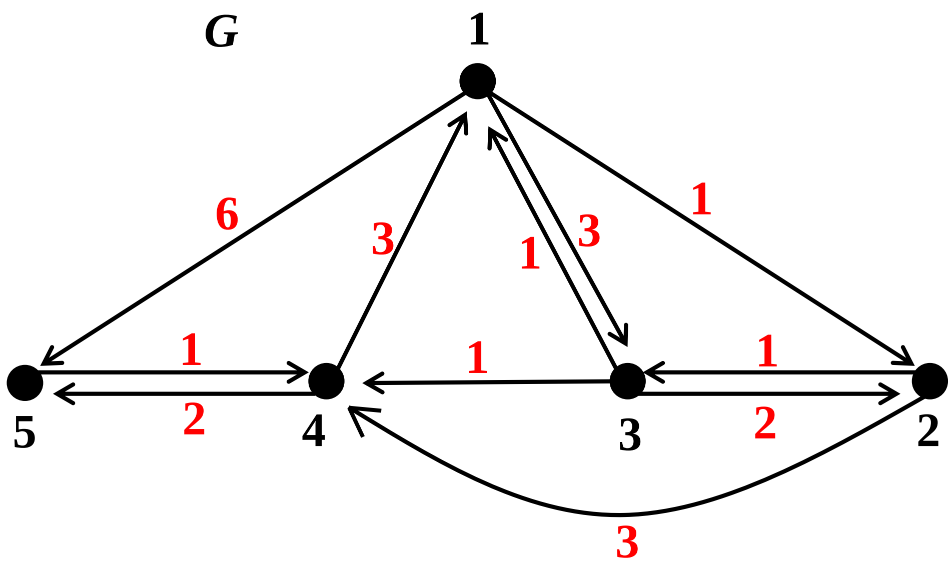
P^5

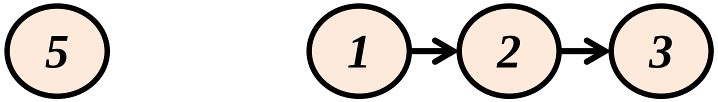
	1	2	3	4	5
1	*null*	1	2	3	4
2	3	*null*	2	3	4
3	3	3	*null*	3	4
4	4	1	2	*null*	4
5	4	1	2	5	*null*

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

Exemplo: calcular o caminho mínimo de 5 a 3





$$P^5(5, 1) = 4$$

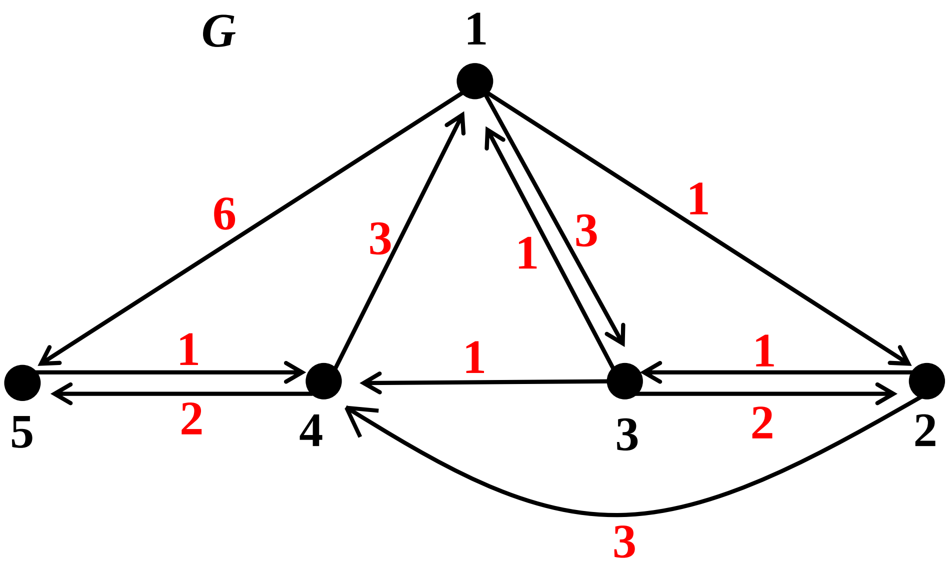
P^5

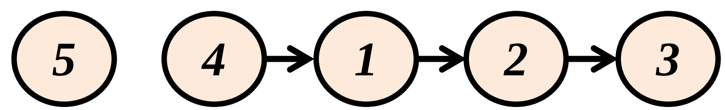
	1	2	3	4	5
1	*null*	1	2	3	4
2	3	*null*	2	3	4
3	3	3	*null*	3	4
4	4	1	2	*null*	4
5	4	1	2	5	*null*

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

Exemplo: calcular o caminho mínimo de 5 a 3





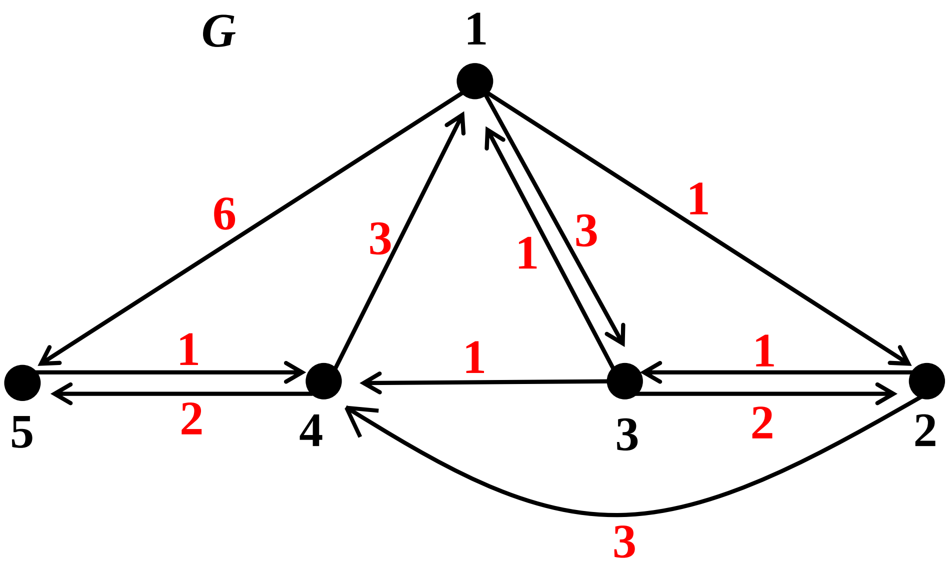
P^5

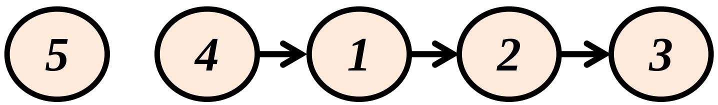
	1	2	3	4	5
1	*null*	1	2	3	4
2	3	*null*	2	3	4
3	3	3	*null*	3	4
4	4	1	2	*null*	4
5	4	1	2	5	*null*

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

Exemplo: calcular o caminho mínimo de 5 a 3





$P^5(5, 4) = 5$

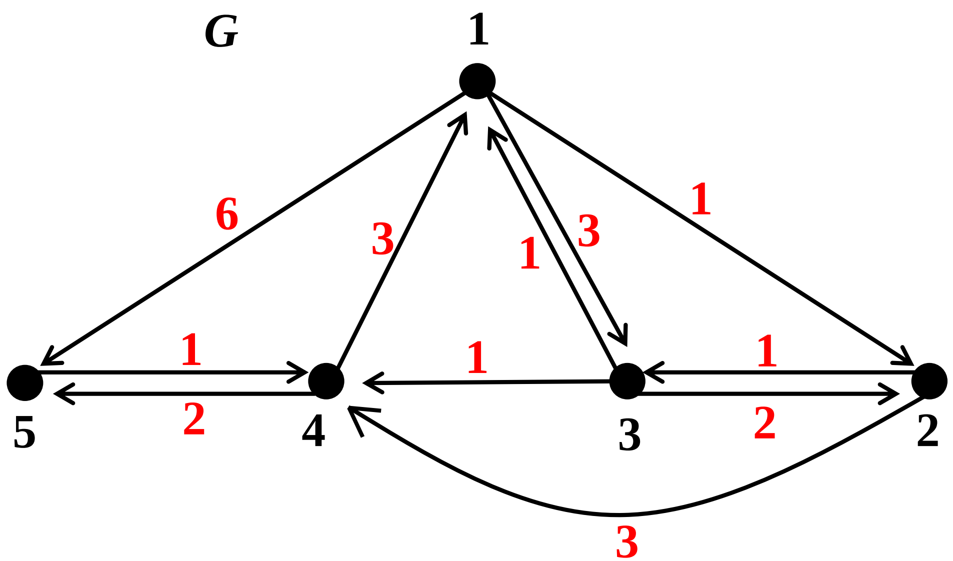
P^5

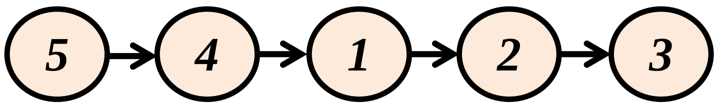
	1	2	3	4	5
1	*null*	1	2	3	4
2	3	*null*	2	3	4
3	3	3	*null*	3	4
4	4	1	2	*null*	4
5	4	1	2	5	*null*

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos …)

Exemplo: calcular o caminho mínimo de 5 a 3





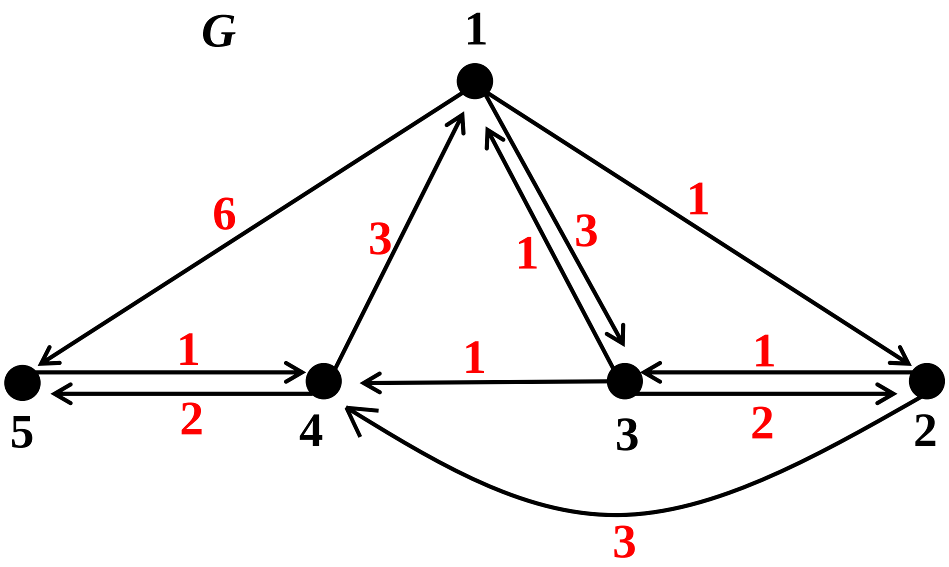
P^5

	1	2	3	4	5
1	*null*	1	2	3	4
2	3	*null*	2	3	4
3	3	3	*null*	3	4
4	4	1	2	*null*	4
5	4	1	2	5	*null*

Algoritmo de Floyd-Warshall

Questão: Como determinar os caminhos mínimos? (e não só seus custos ...)

Exemplo: calcular o caminho mínimo de 5 a 3



5 → *4* → *1* → *2* → *3*

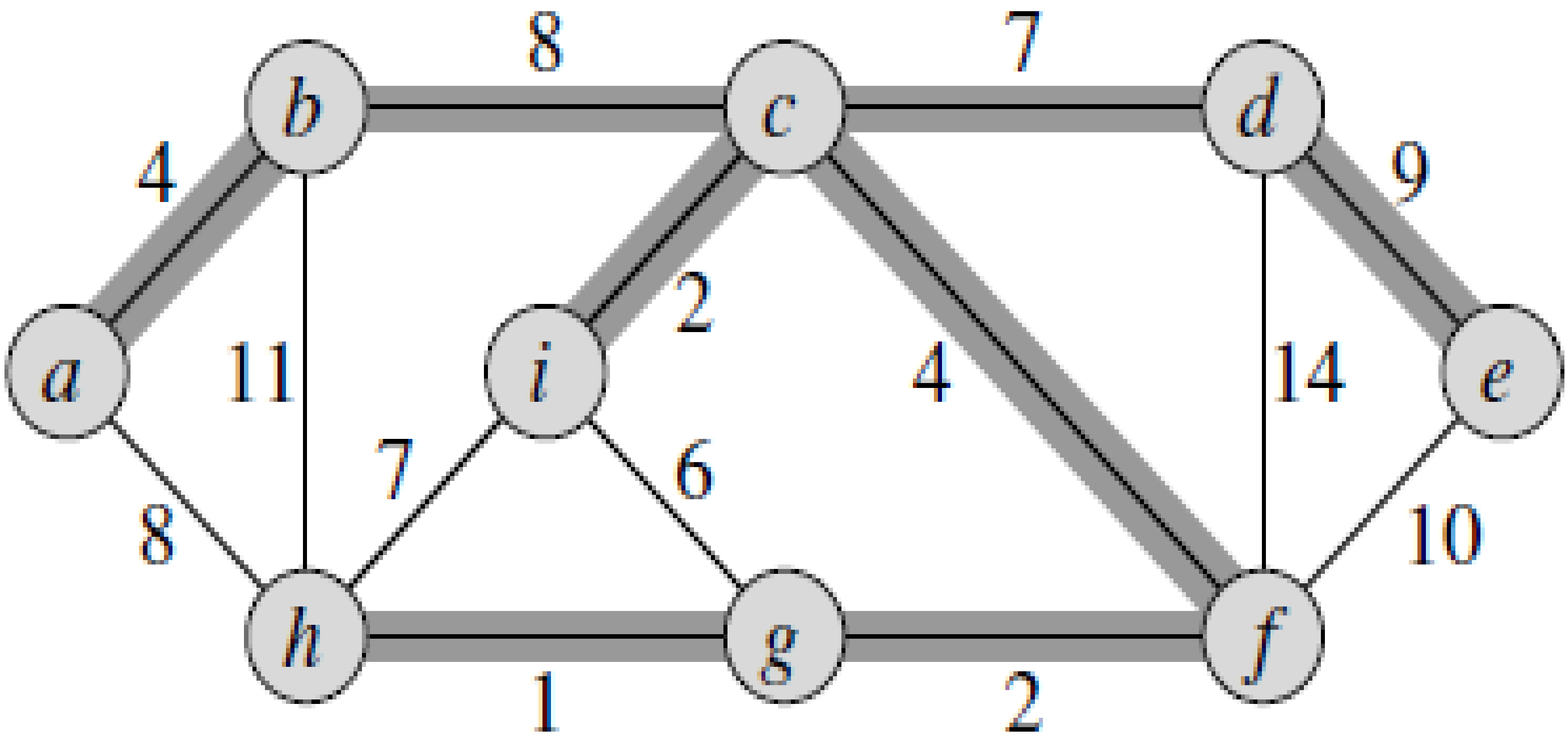
FIM!

P^5

	1	2	3	4	5
1	*null*	1	2	3	4
2	3	*null*	2	3	4
3	3	3	*null*	3	4
4	4	1	2	*null*	4
5	4	1	2	5	*null*

Algoritmo de Kruskal

Problema: Dado um grafo conexo *G* com pesos positivos nas arestas, determinar uma árvore geradora de *G* com custo mínimo.



Algoritmo de Kruskal

O Algoritmo de Kruskal utiliza uma técnica de programação conhecida como **Método Guloso**, cuja estratégia geral é: **"a cada nova iteração, acrescente à solução parcial a parte mais *apetitosa***".

Algoritmo de Kruskal

Dado o grafo G, o algoritmo constrói uma árvore geradora T de G com custo mínimo

$V(T) \leftarrow V(G)$; $E(T) \leftarrow \emptyset$; ***-- inicialização de T***

seja $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_m$ *uma ordenação das arestas de G onde* $peso(\mathbf{e}_j) \le peso(\mathbf{e}_{j+1})$, $j = 1,\ldots, m-1$

para $j = 1,\ldots, m$ *faça*

    se $\mathbf{e}_j$ *não forma ciclo com as arestas em* $E(T)$

        então *acrescente* $\mathbf{e}_j$ *a* $E(T)$

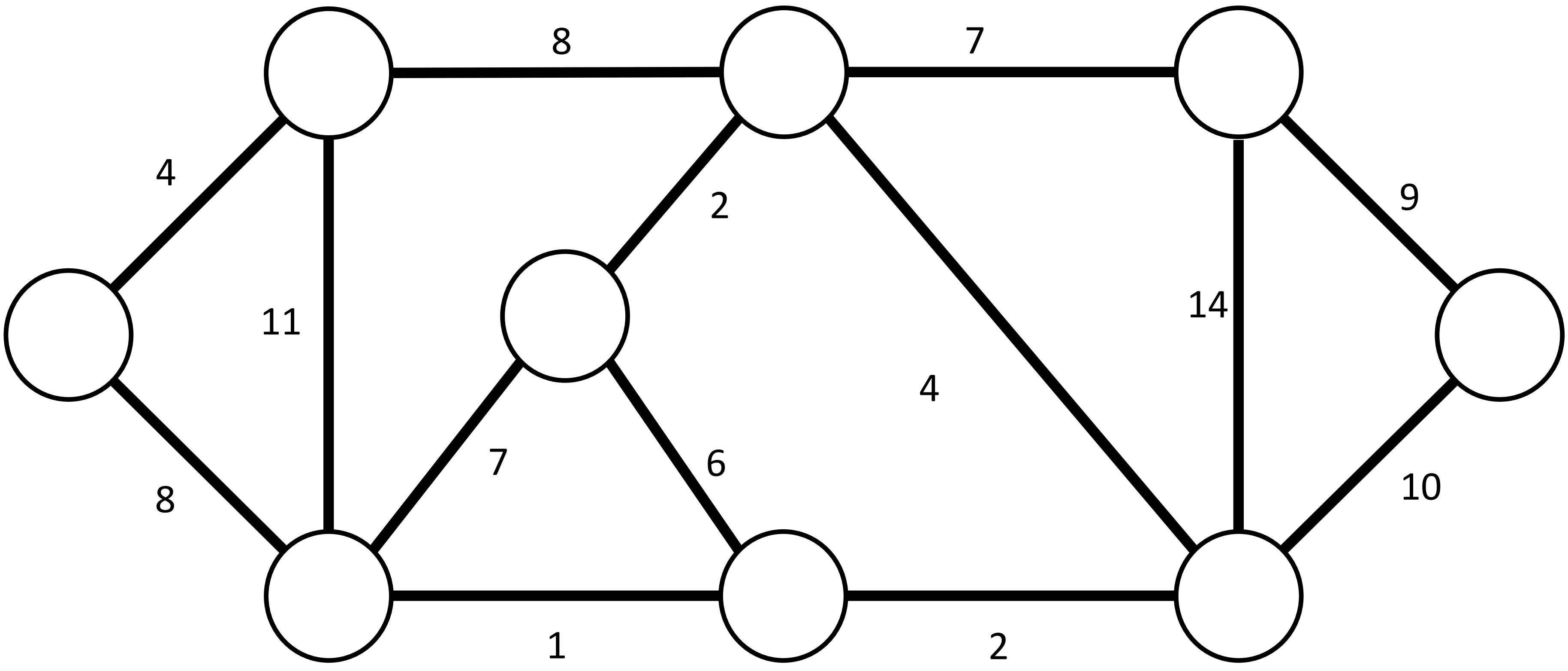
    fim-se

fim-para

retorne T

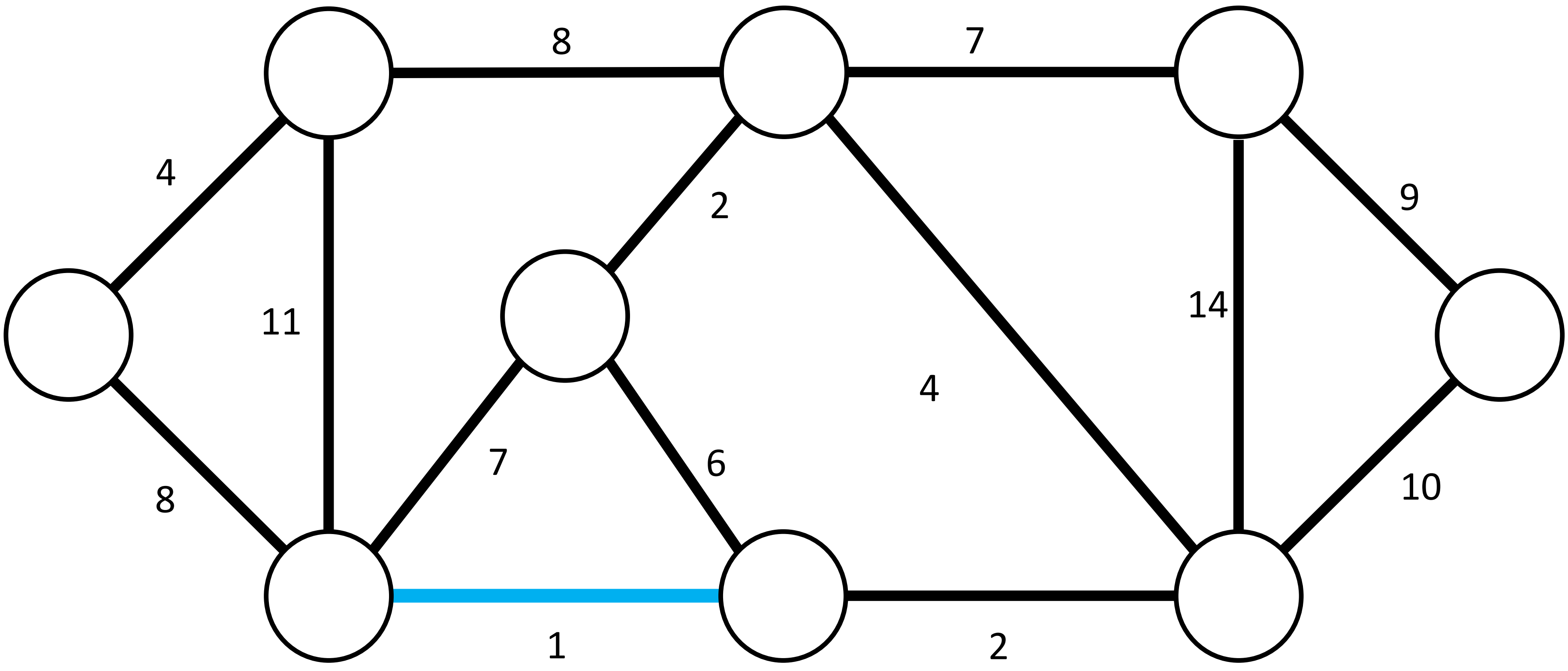
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



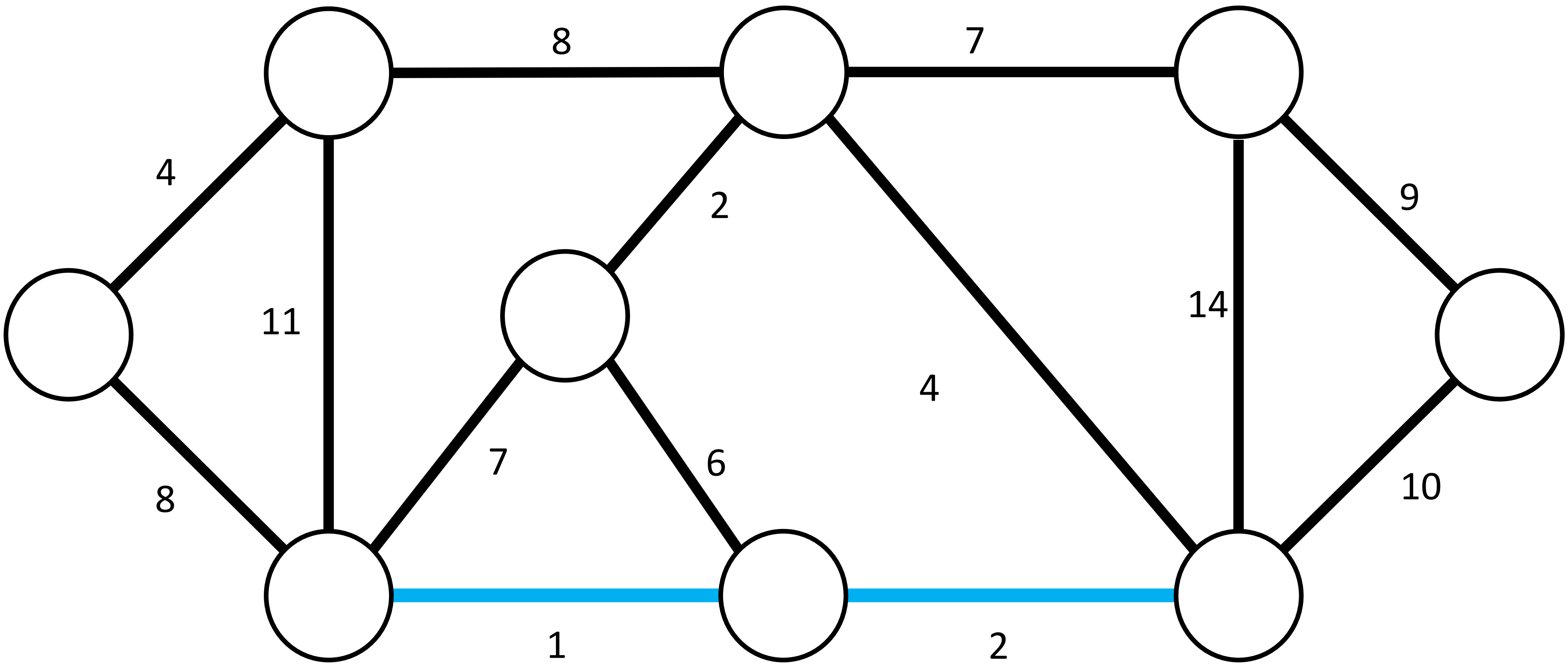
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



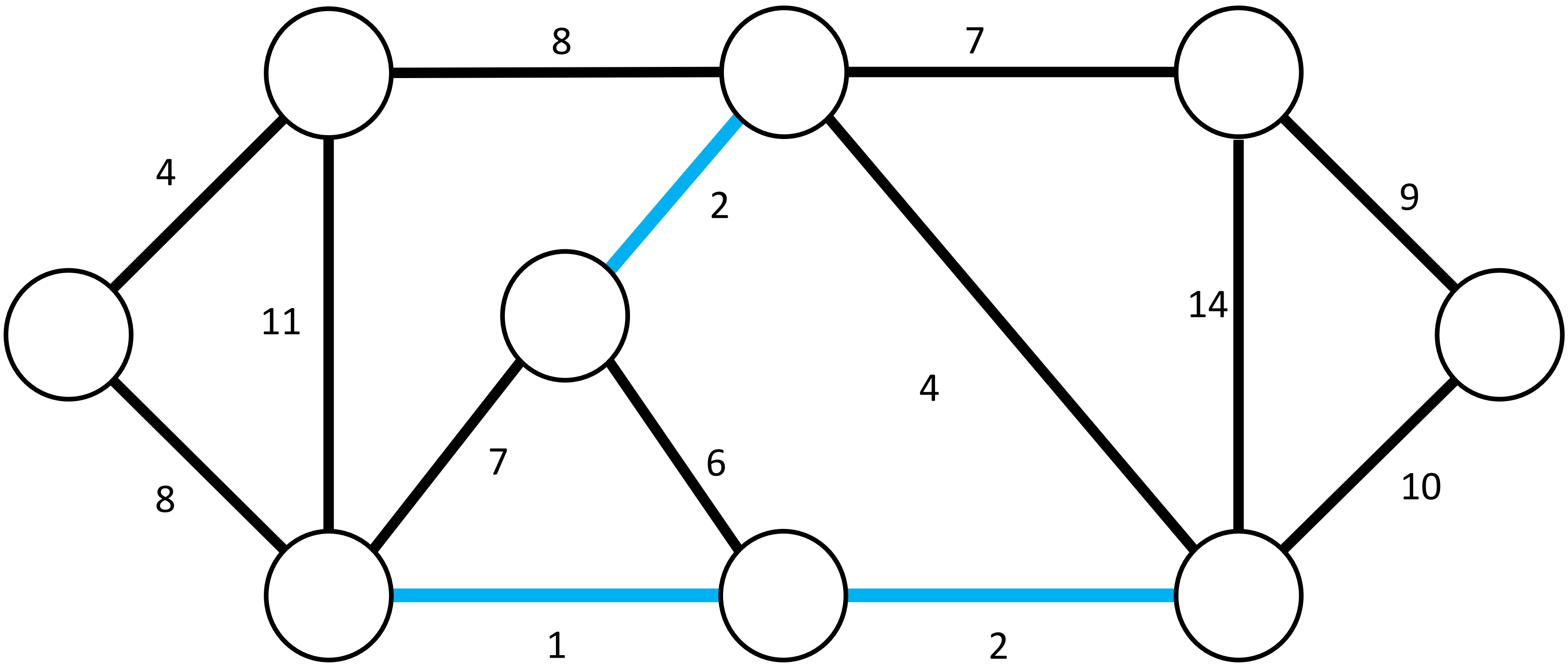
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



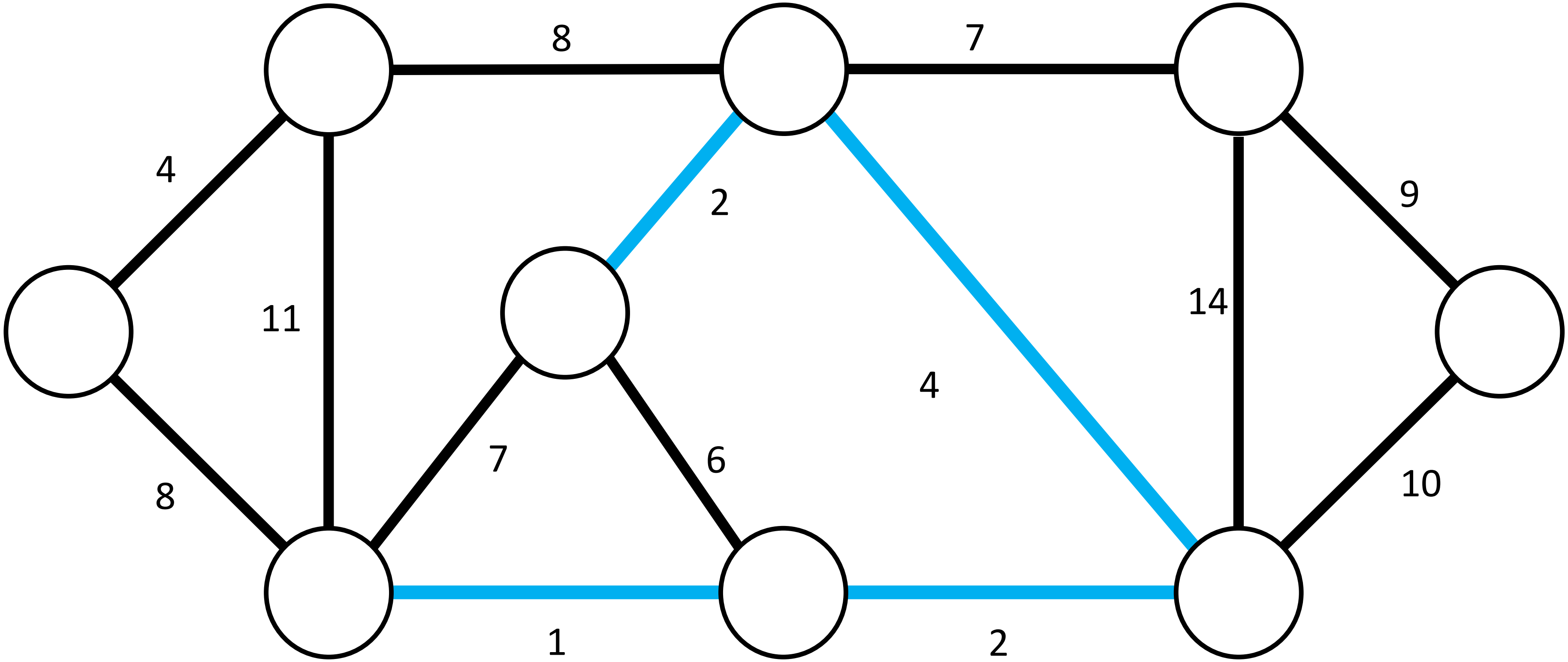
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



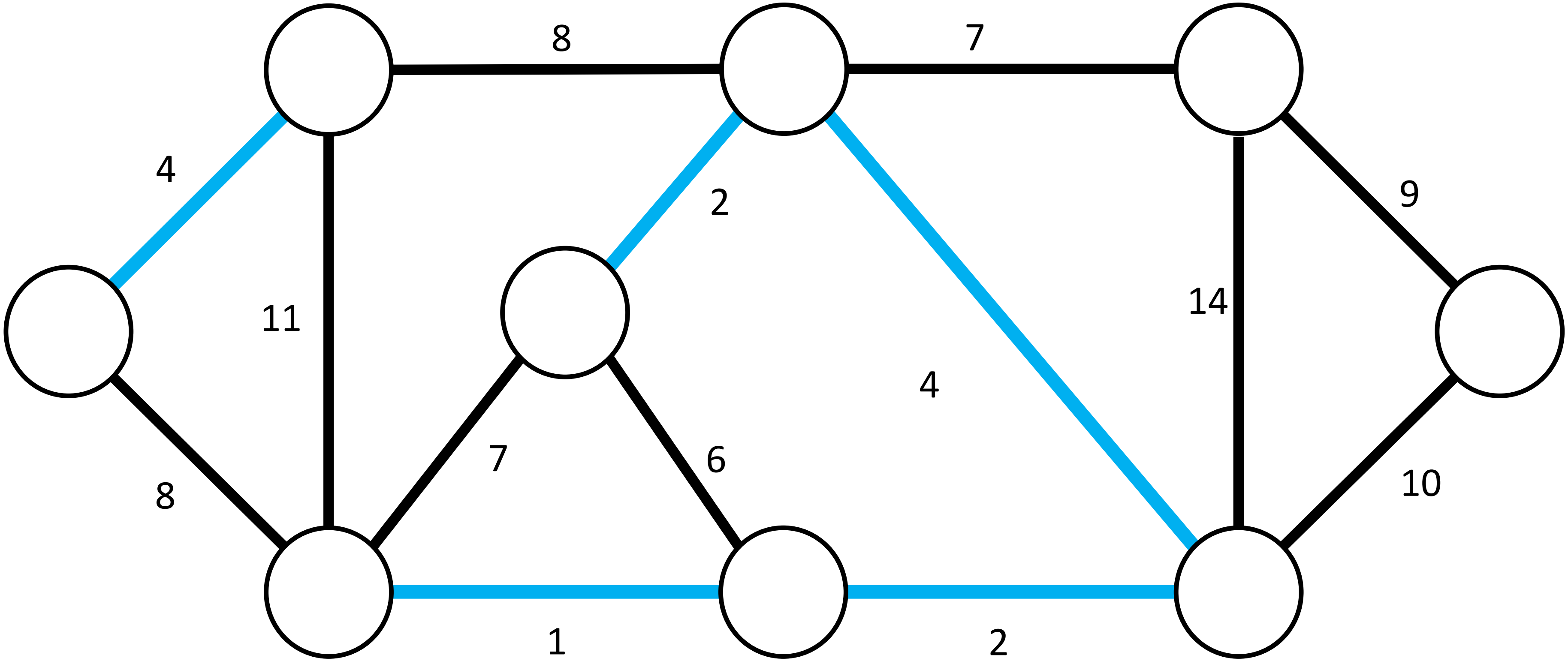
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



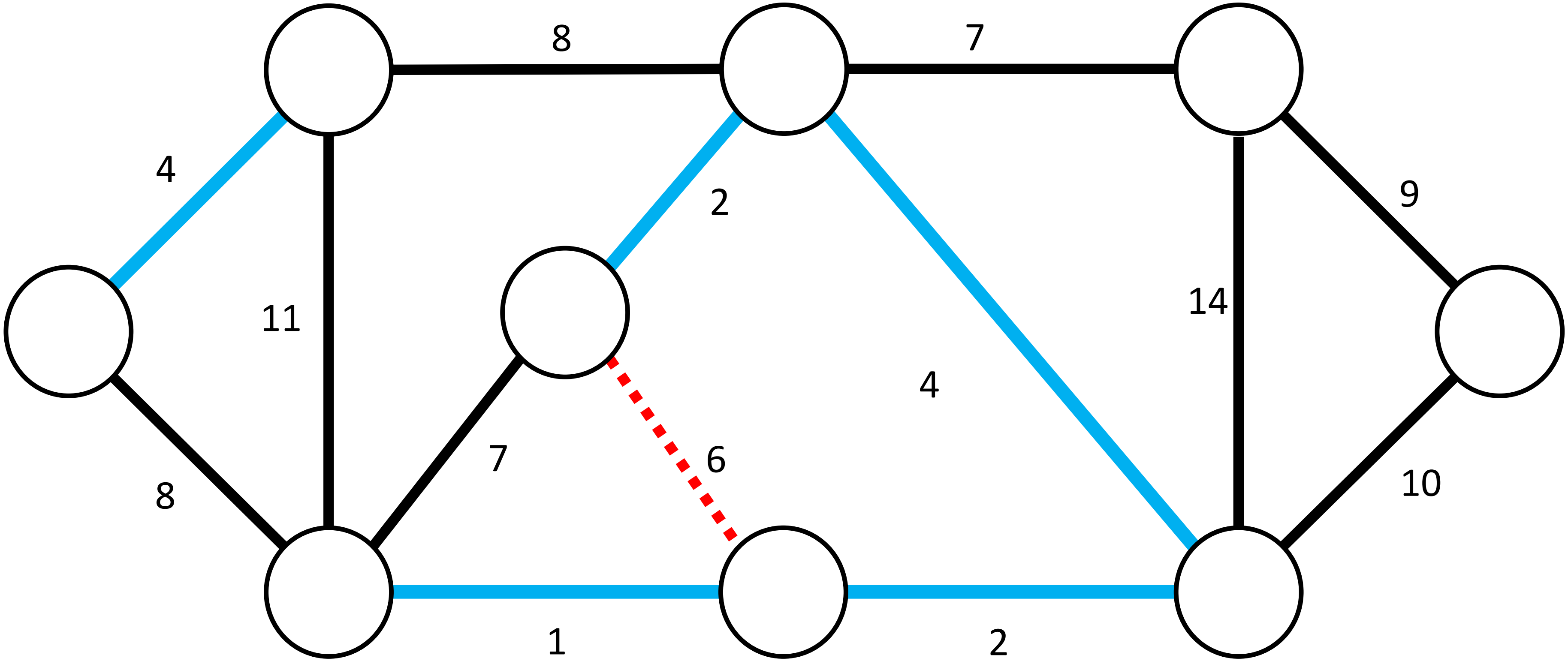
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



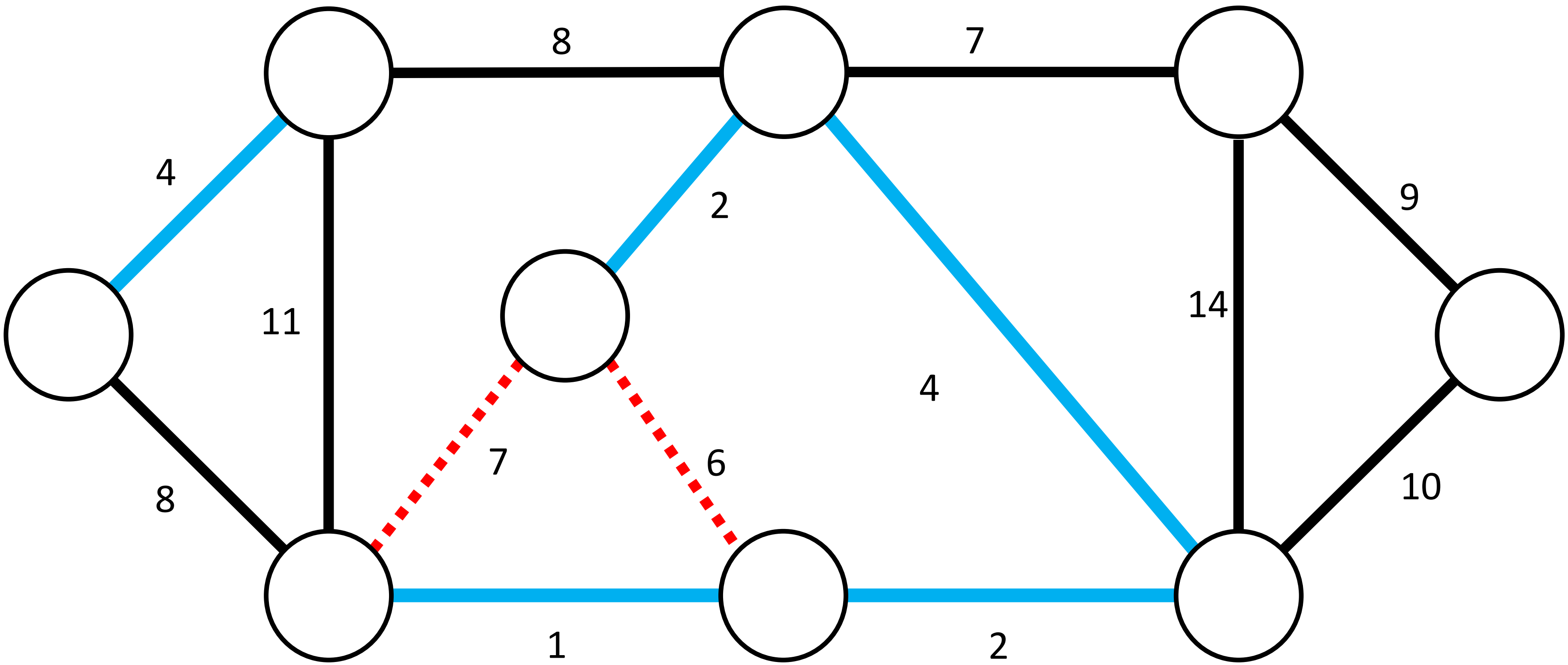
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



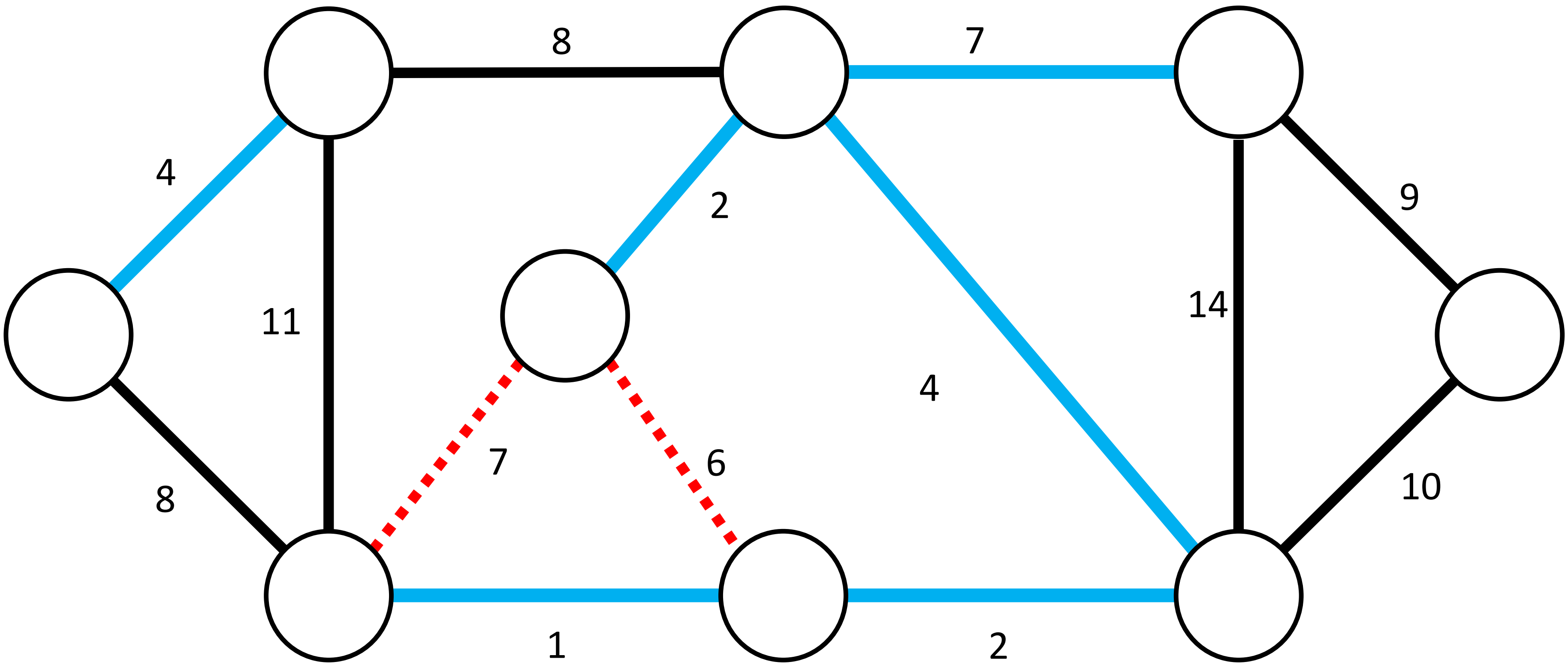
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



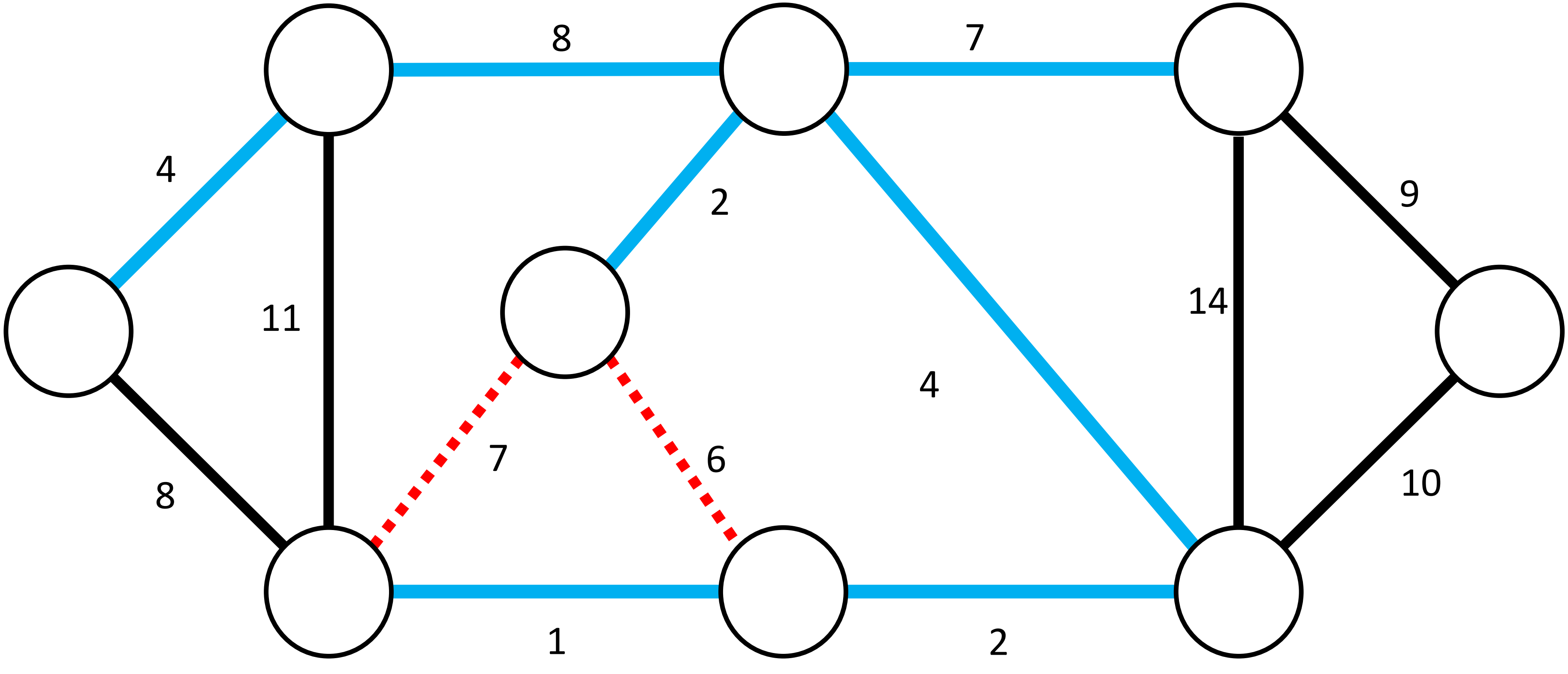
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



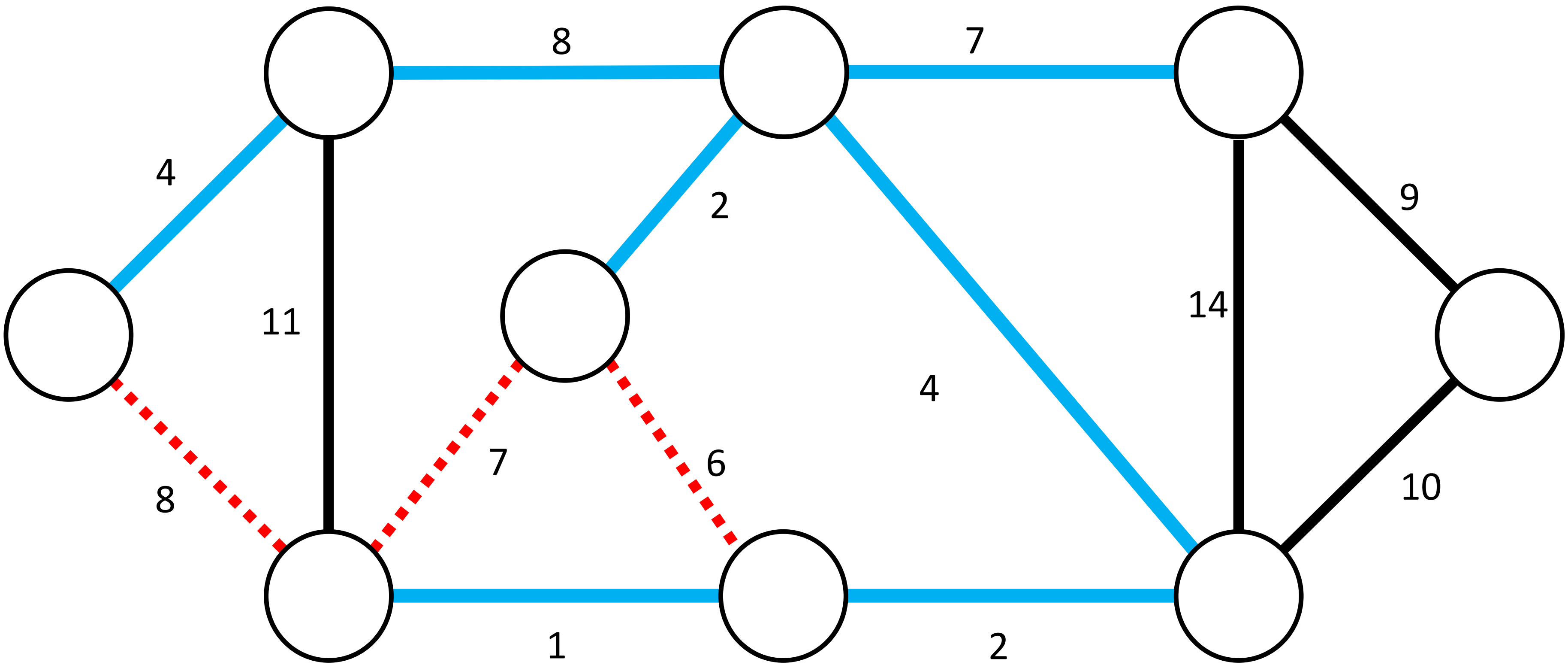
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



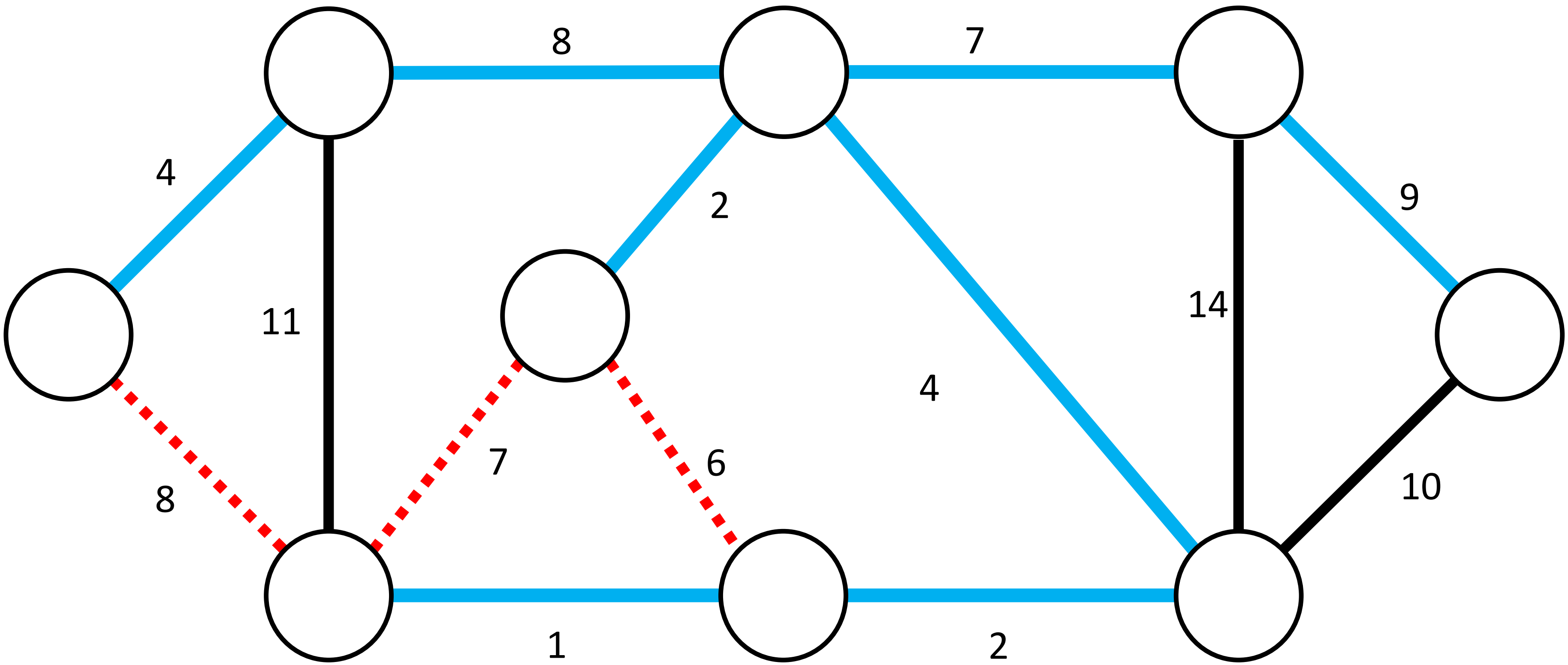
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



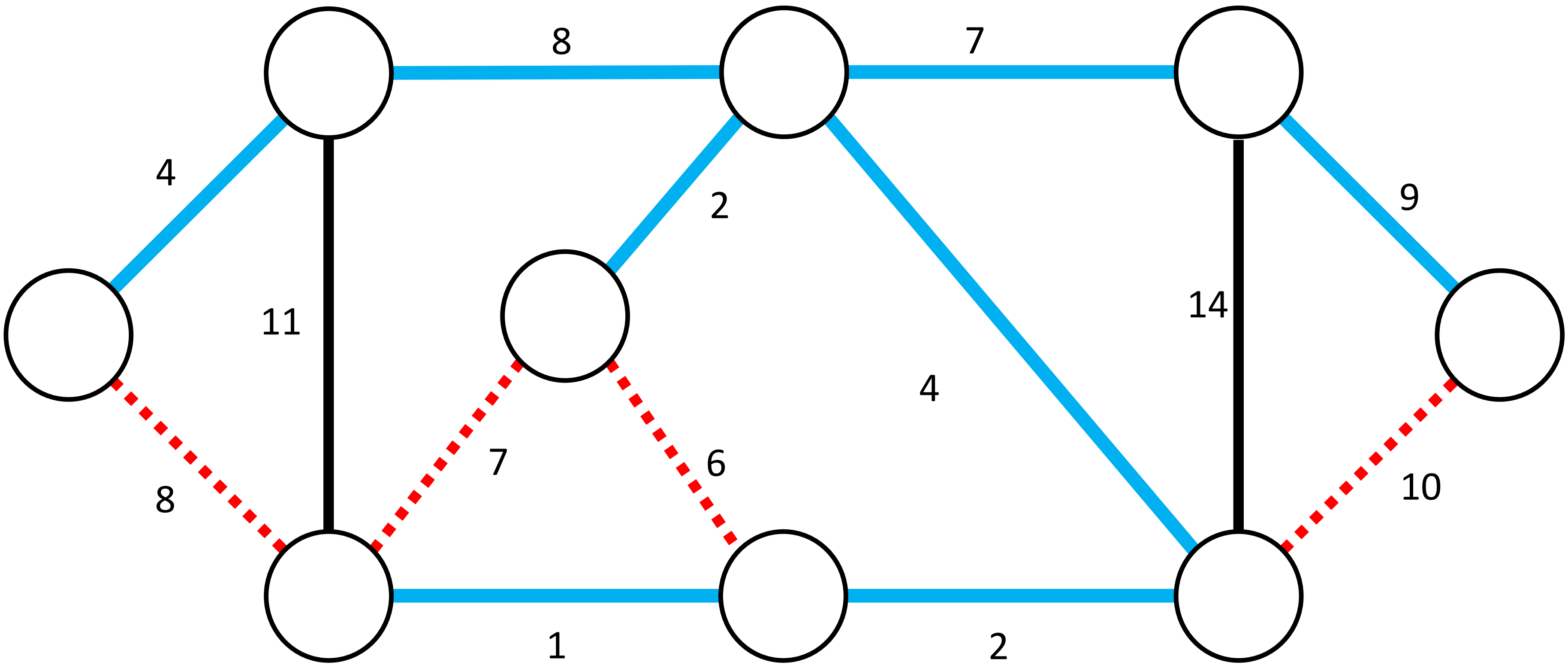
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



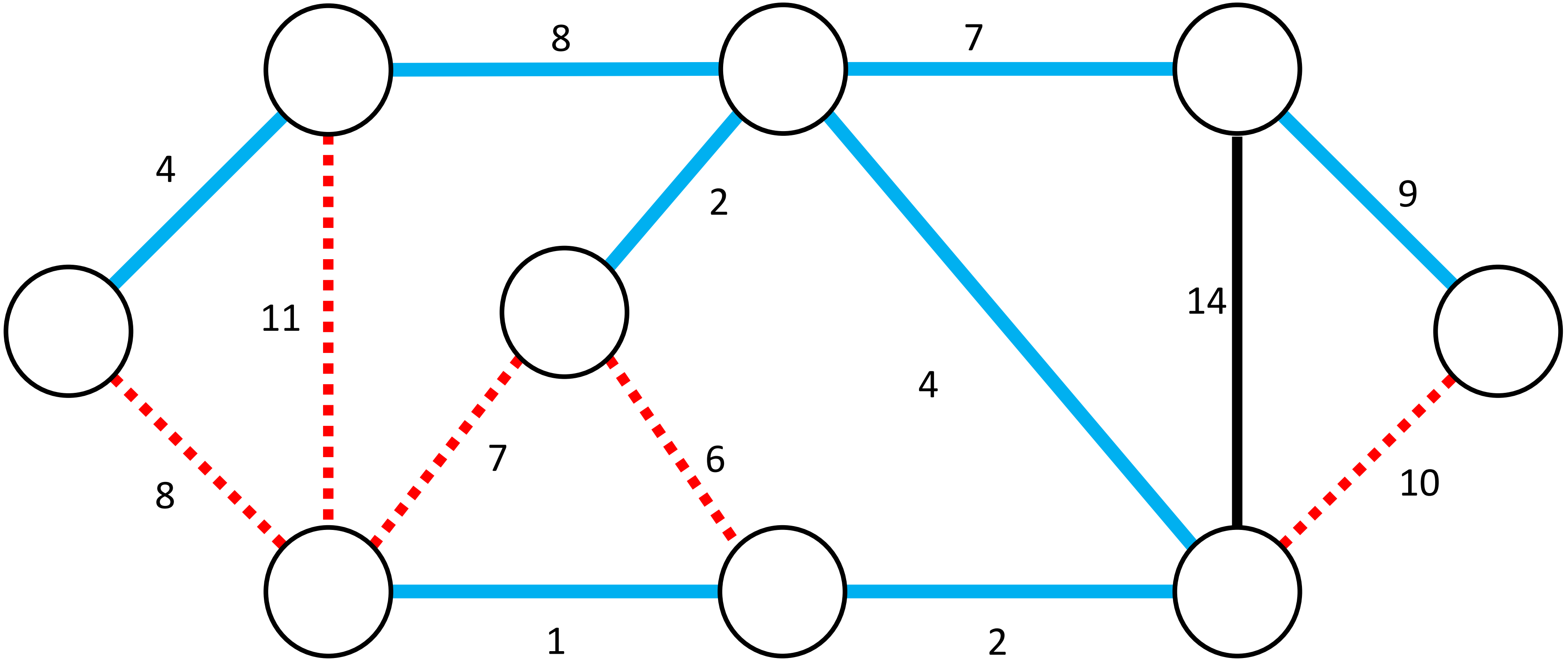
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



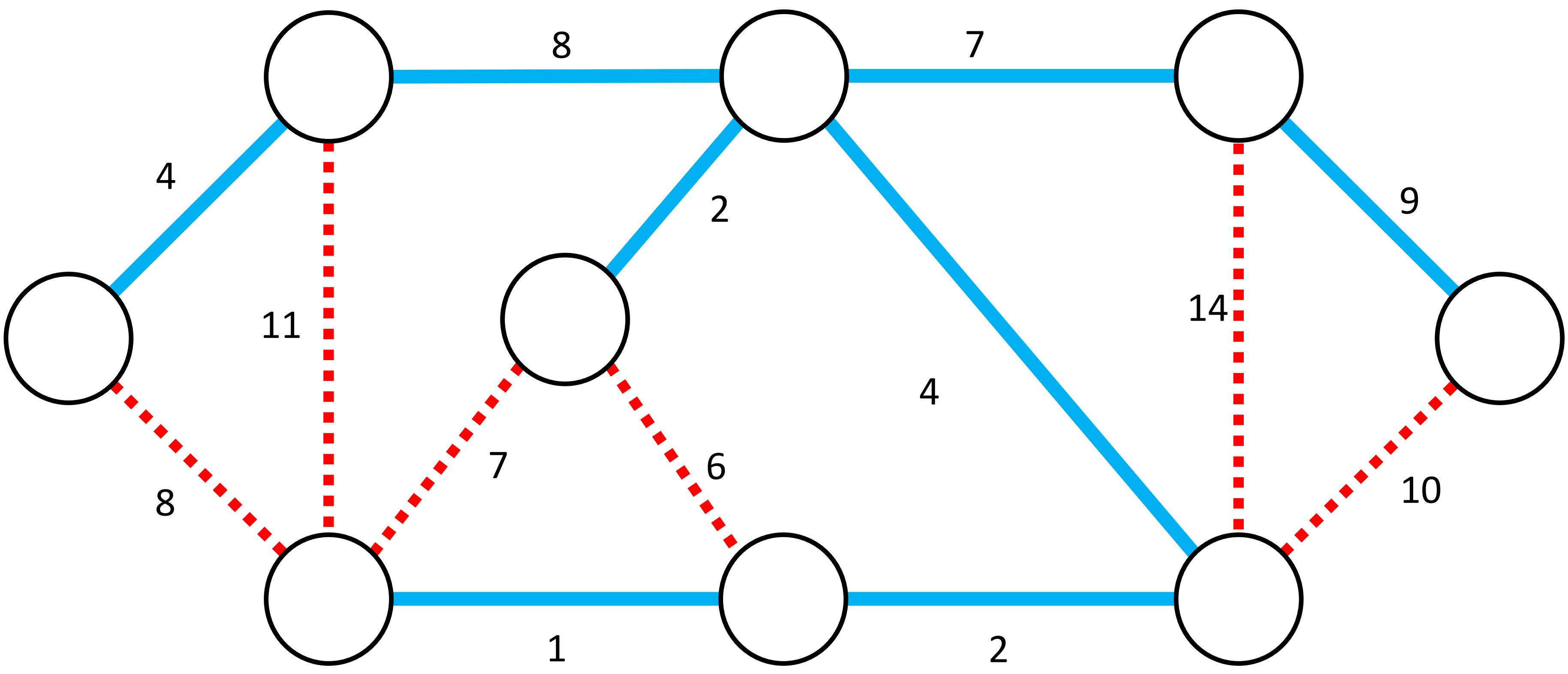
Algoritmo de Kruskal

Exemplo: Simulação da execução do algoritmo.



Algoritmo de Kruskal

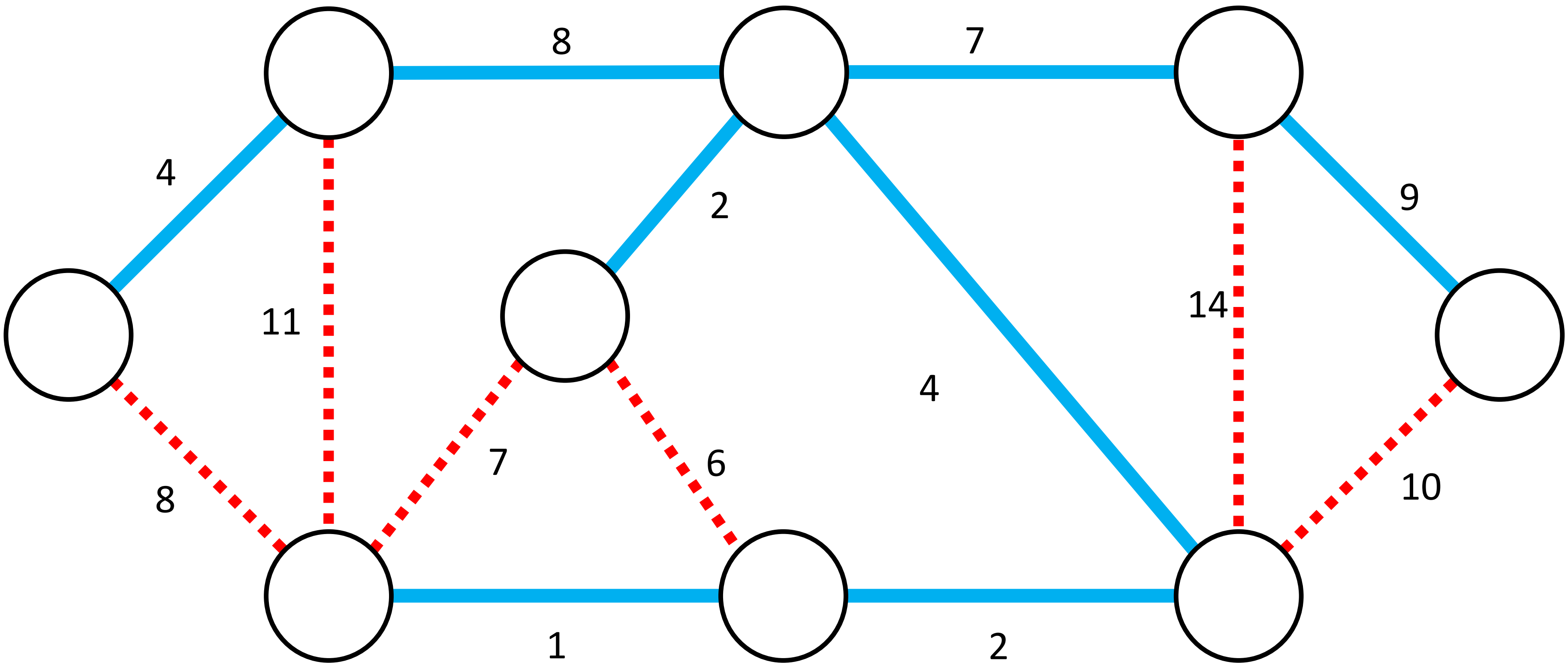
Exemplo: Simulação da execução do algoritmo.



Algoritmo de Kruskal

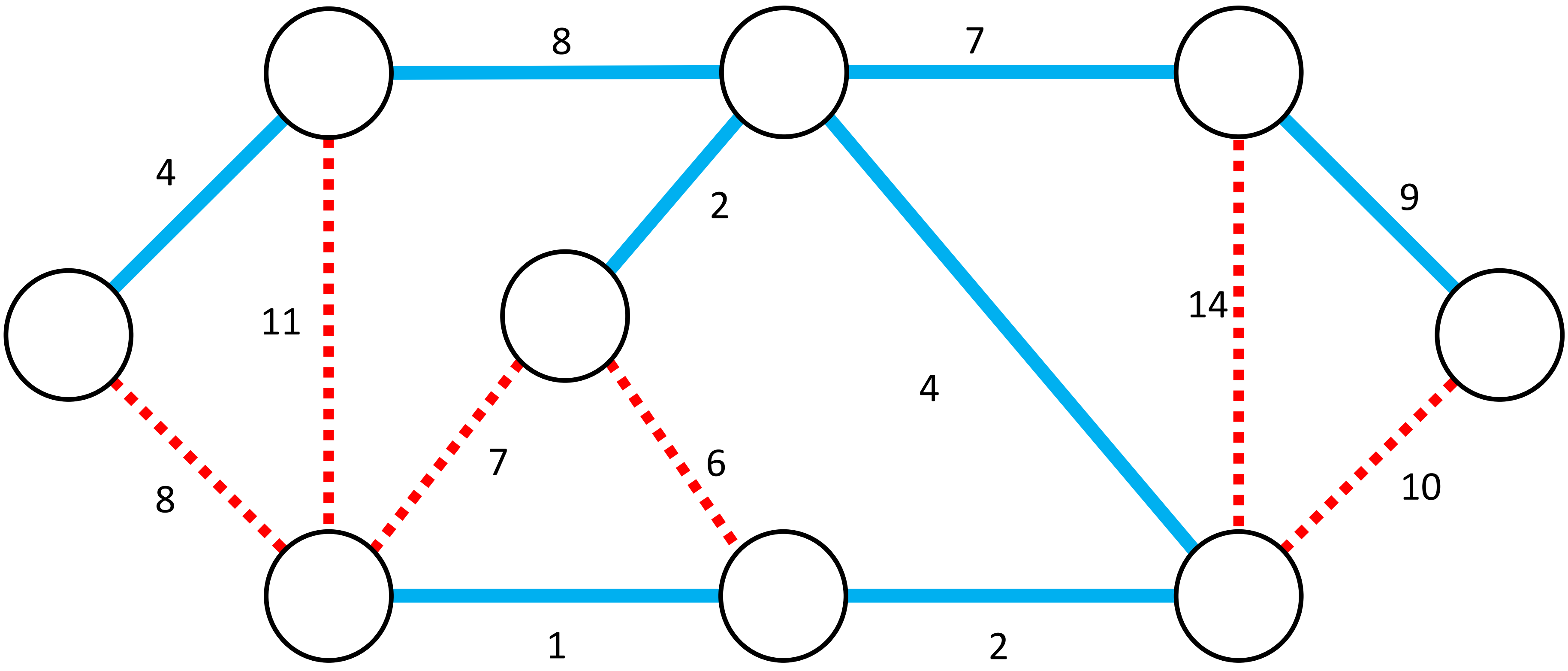
Exemplo: Simulação da execução do algoritmo.

custo final da árvore geradora: 37



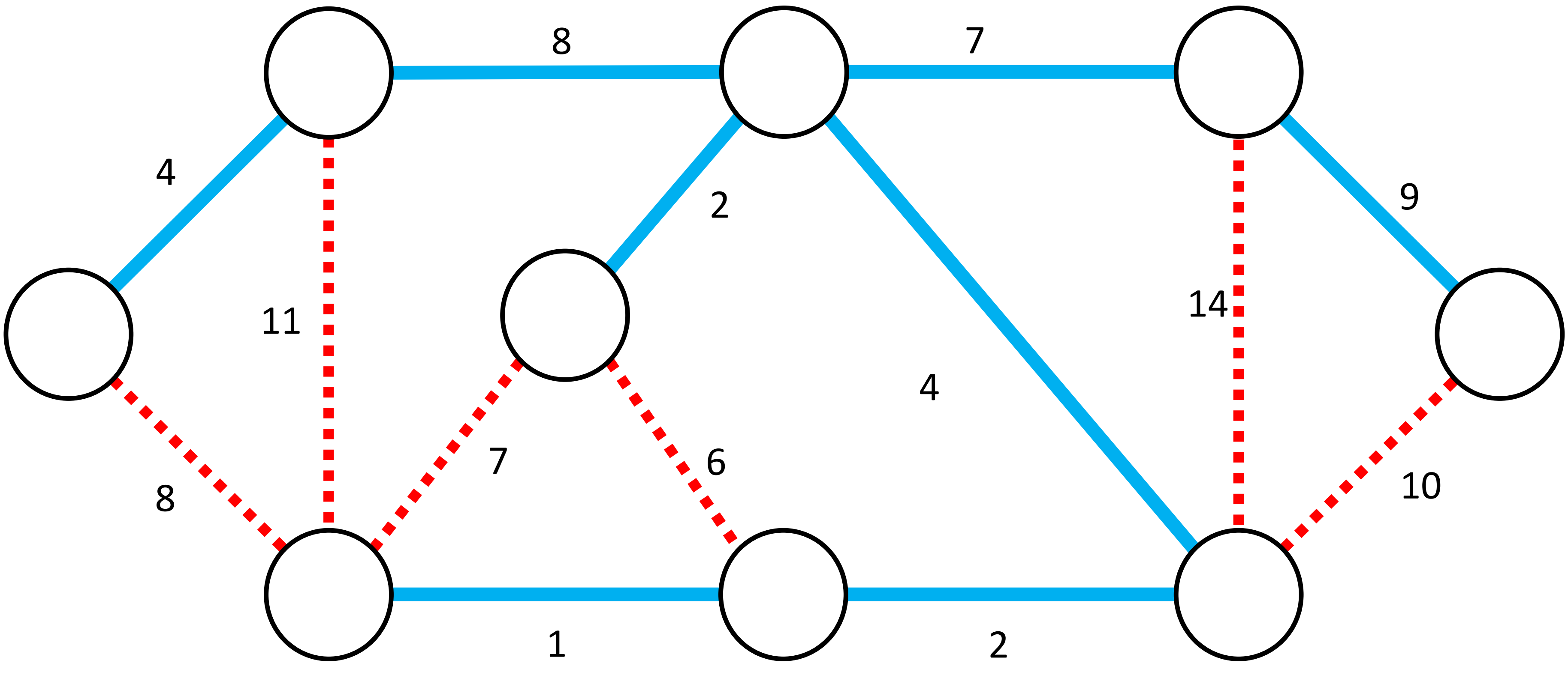
Algoritmo de Kruskal

Questão: Como saber de modo eficiente se a aresta em consideração forma ciclo com as azuis já escolhidas?



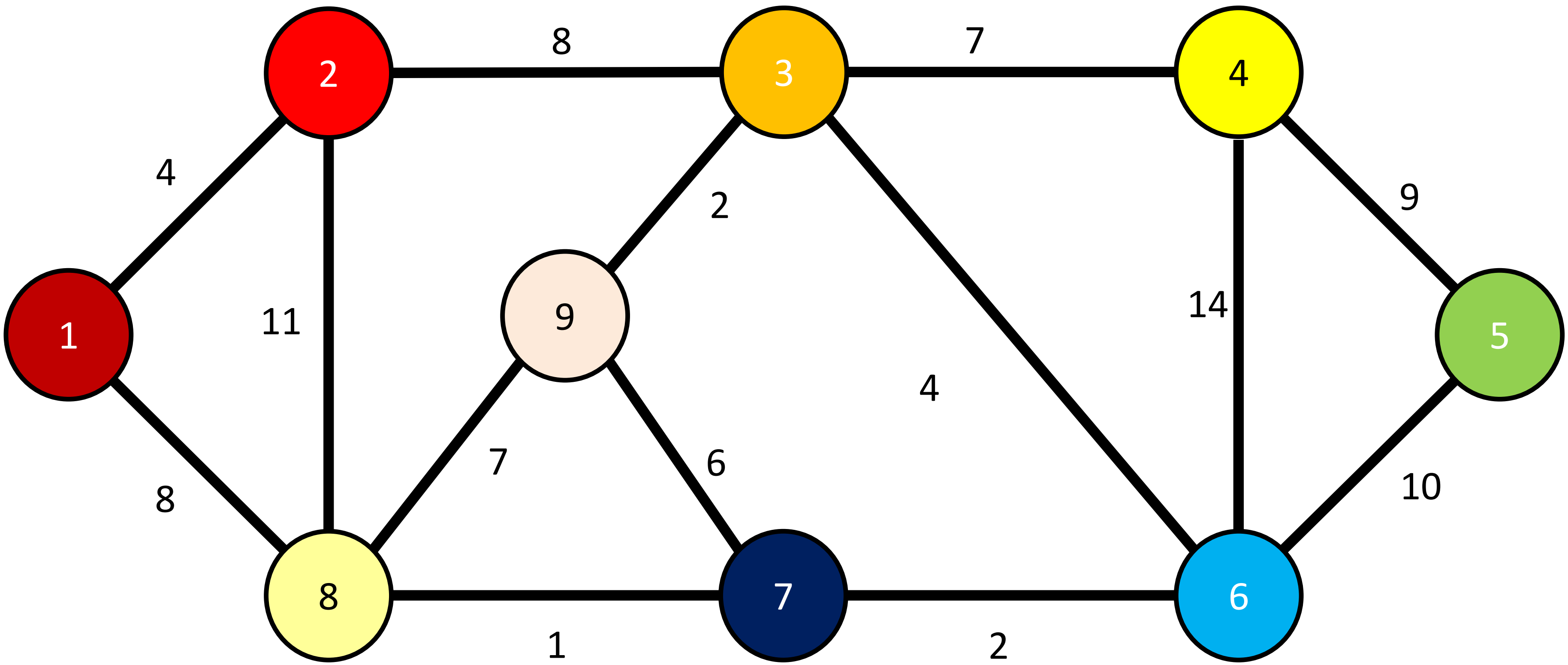
Algoritmo de Kruskal

Estratégia: Uma nova aresta pode ser incorporada à solução (floresta) parcial se ela une duas árvores distintas!



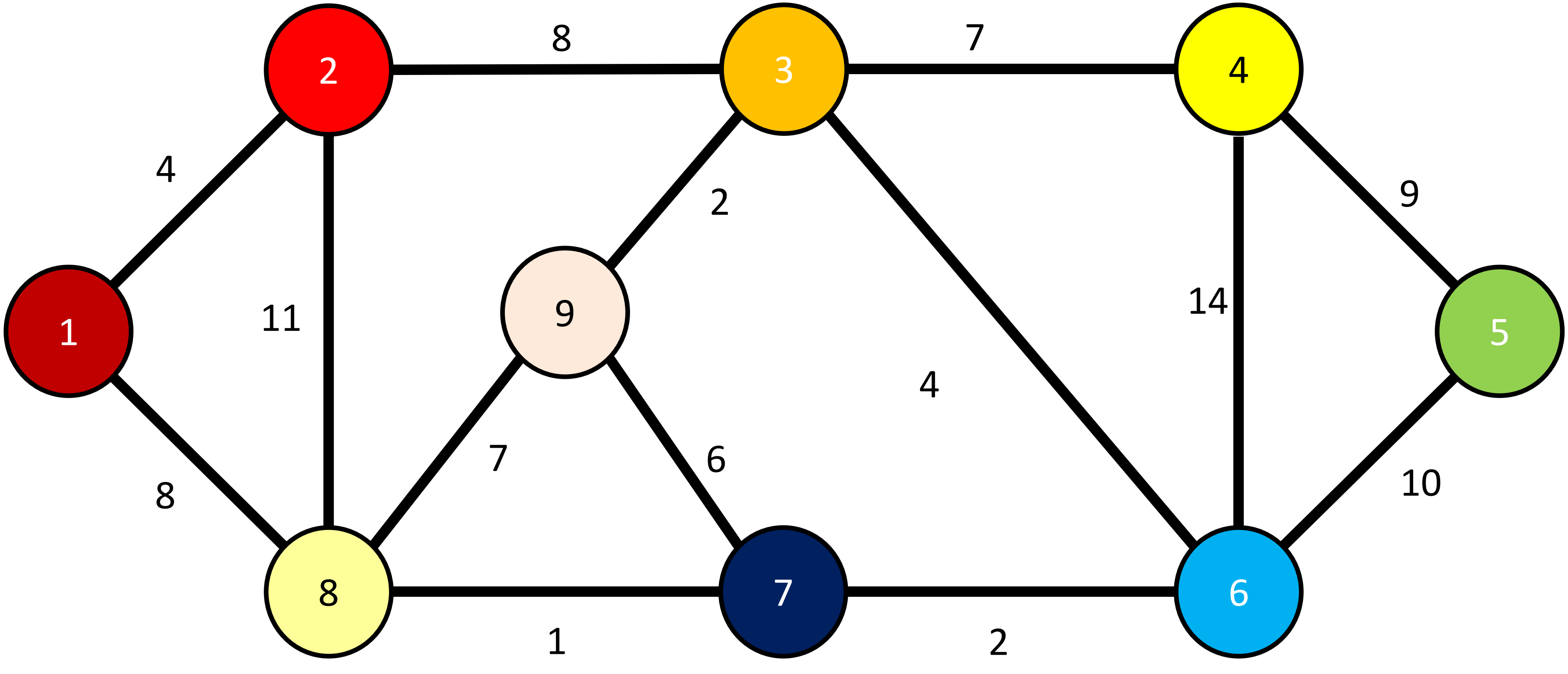
Algoritmo de Kruskal

Início: Os *n* vértices formam *n* árvores triviais distintas (cada uma com sua cor). Nenhuma aresta foi escolhida.



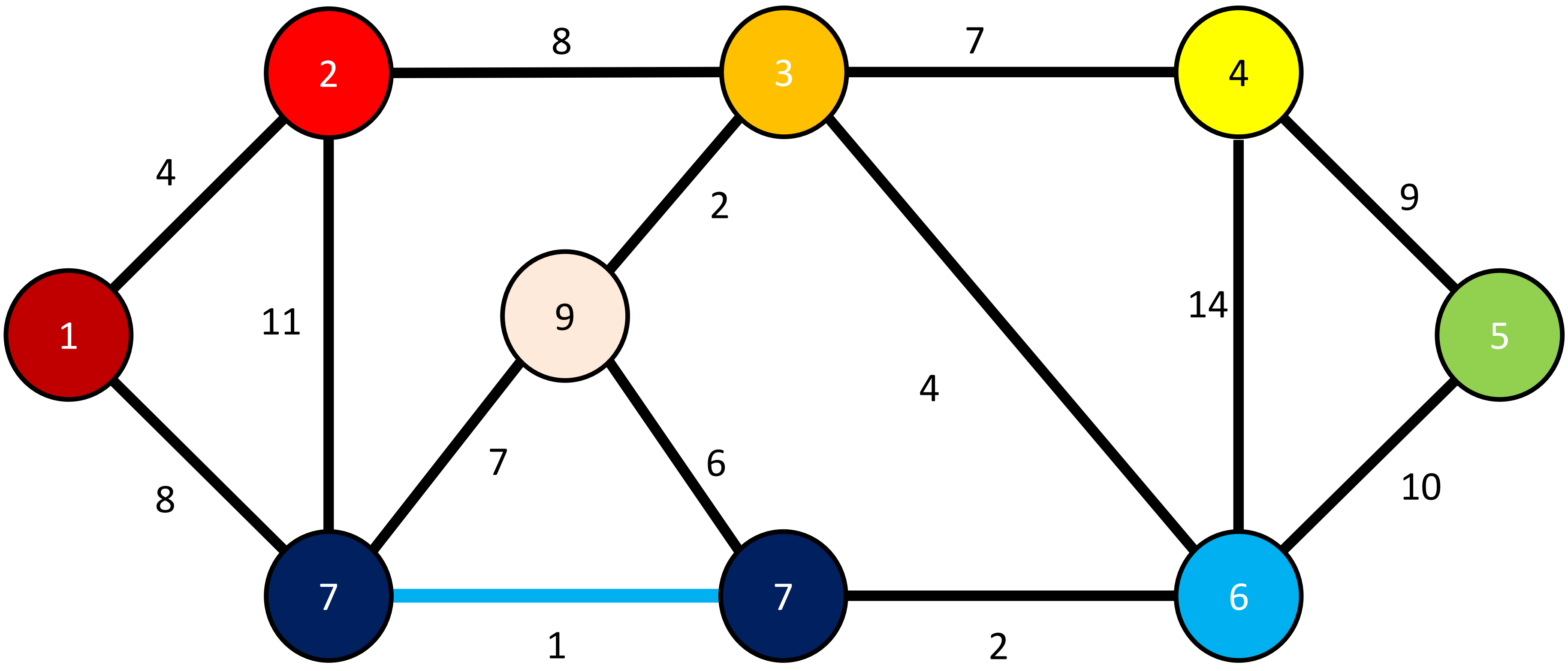
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



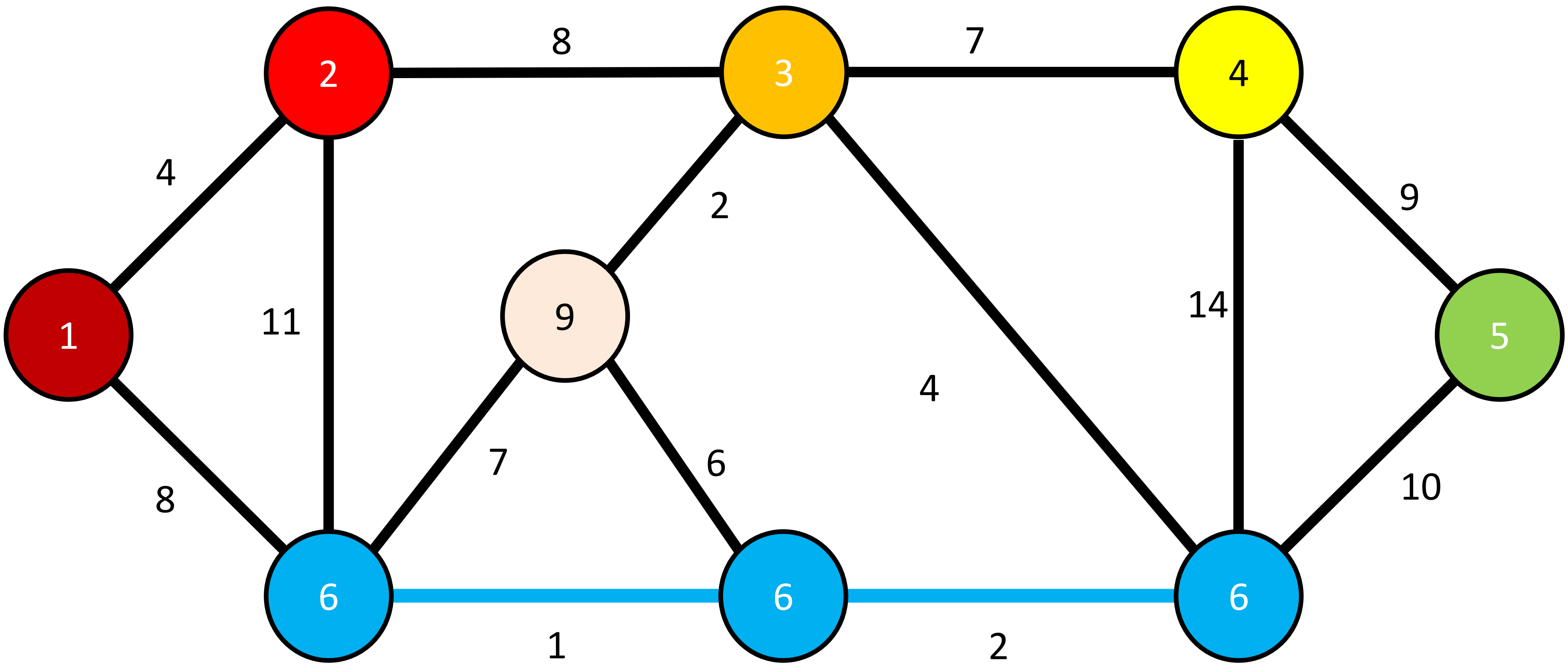
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



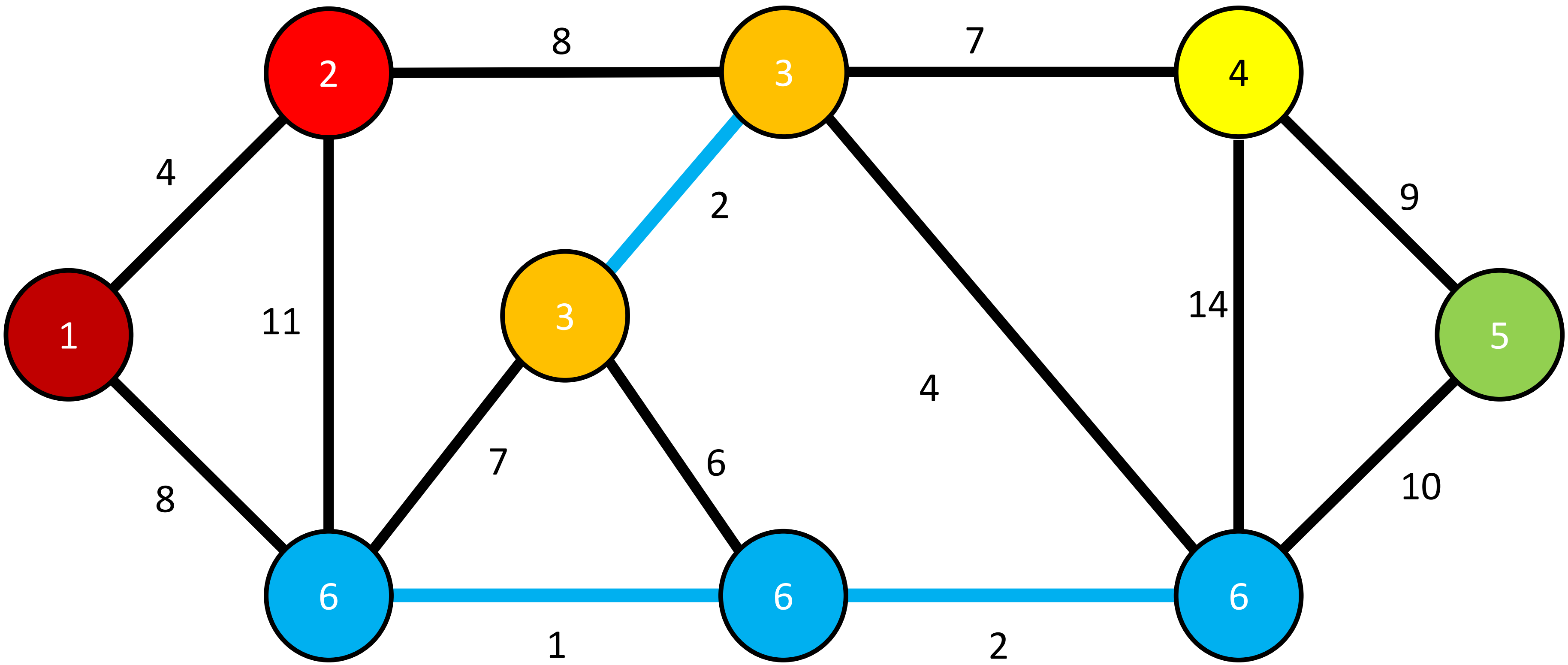
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



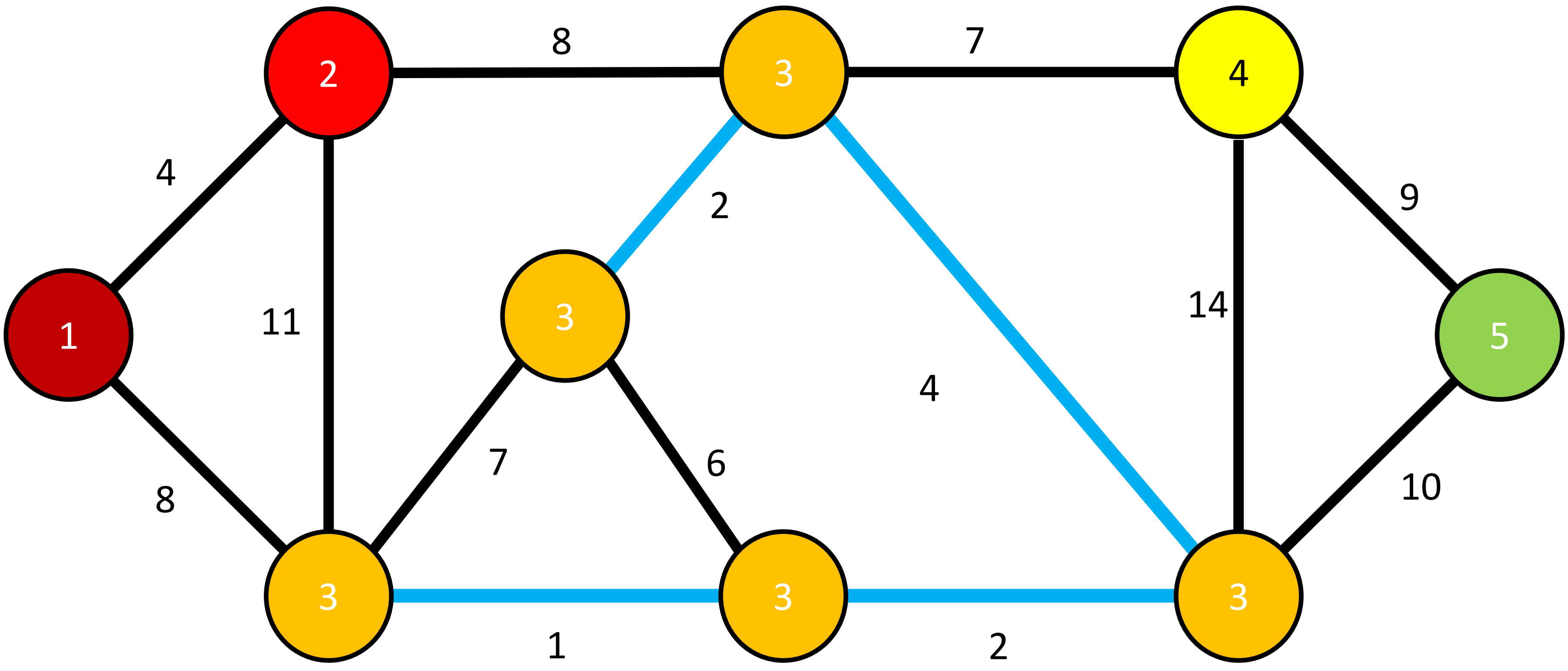
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



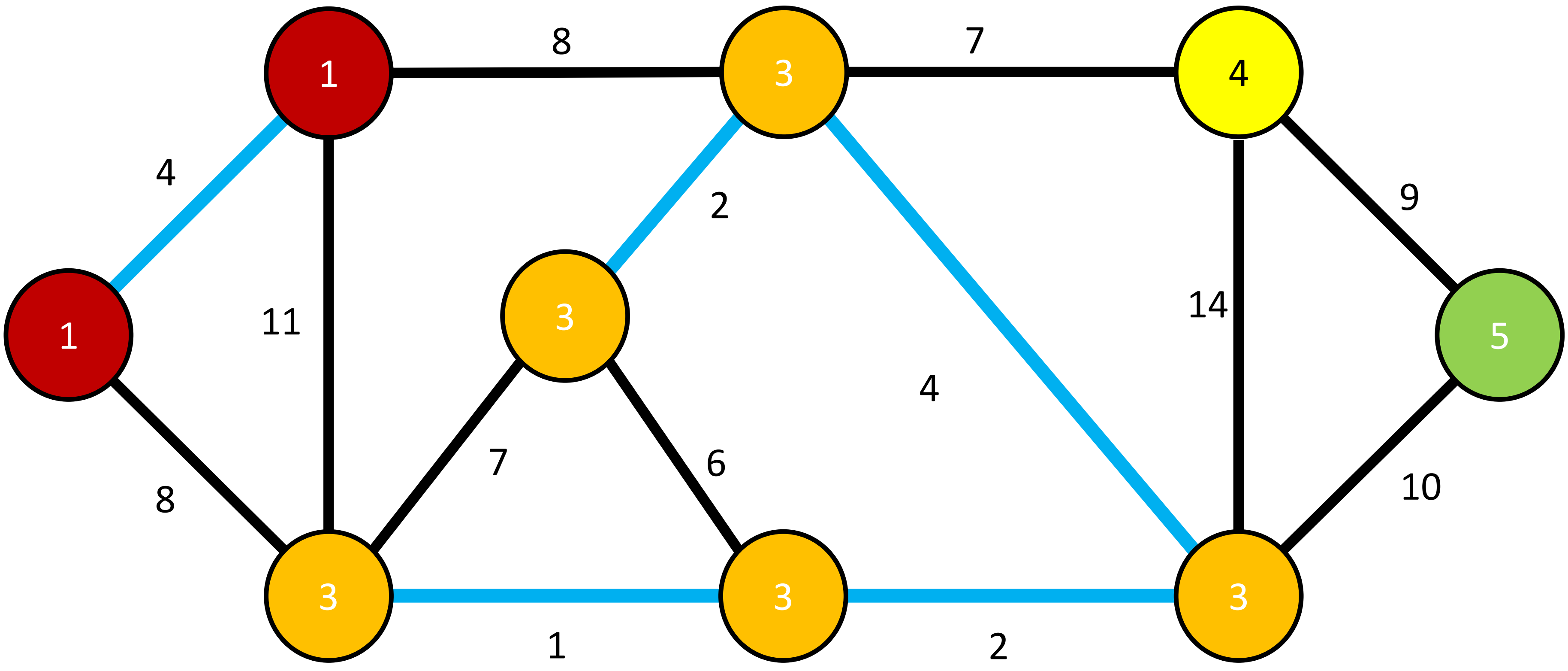
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



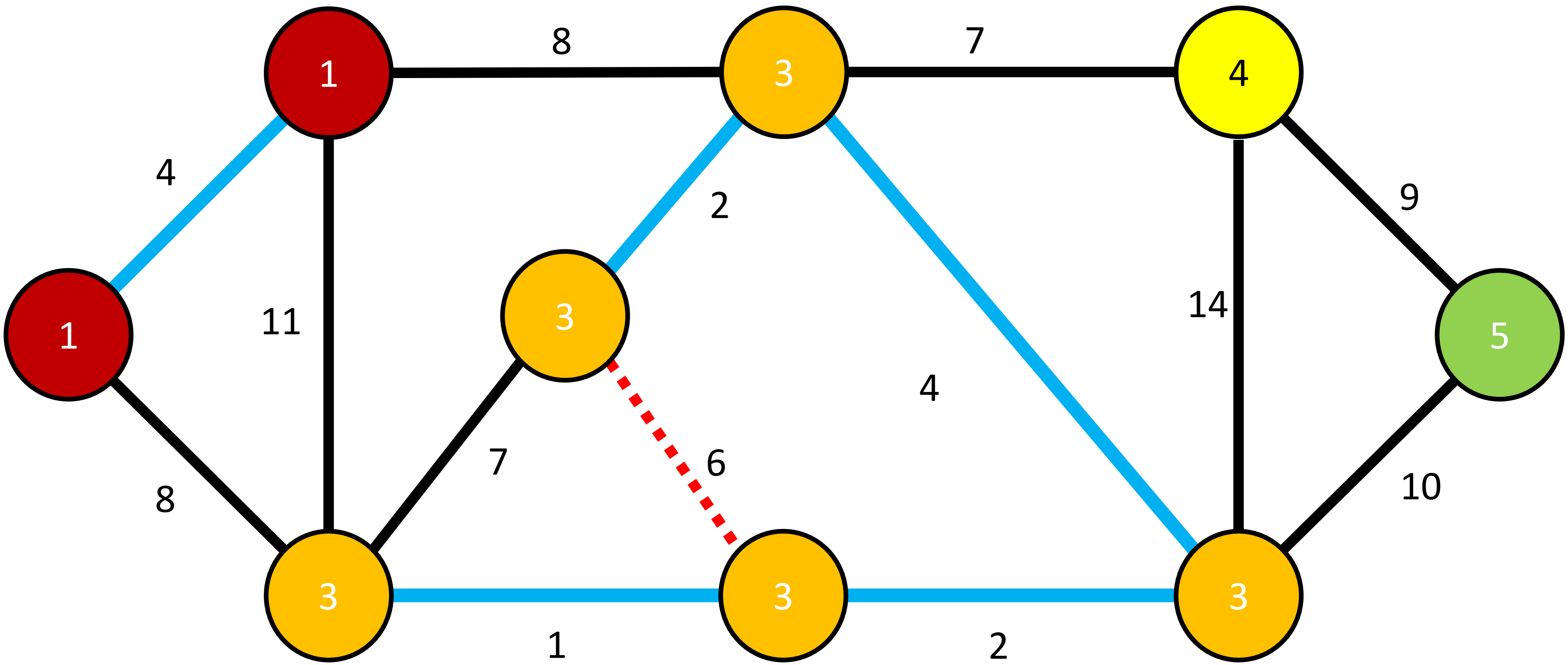
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



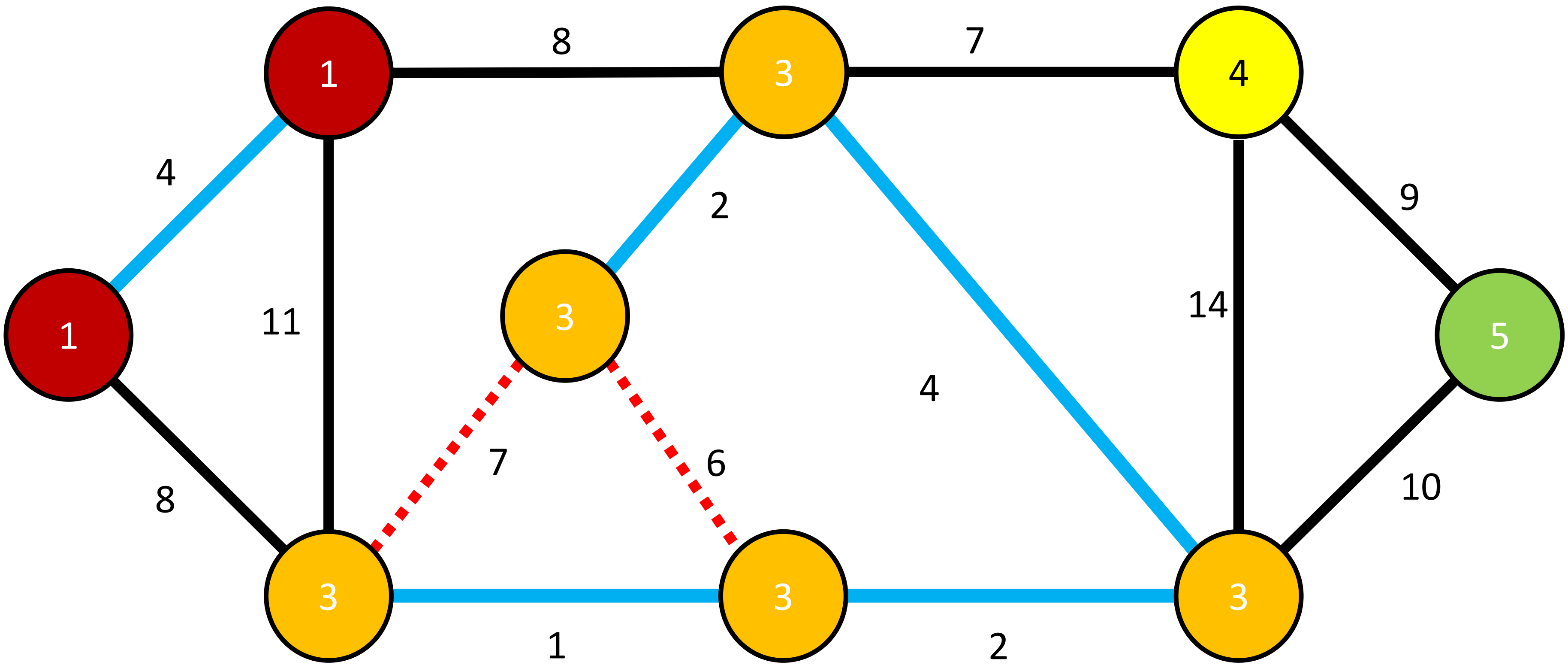
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



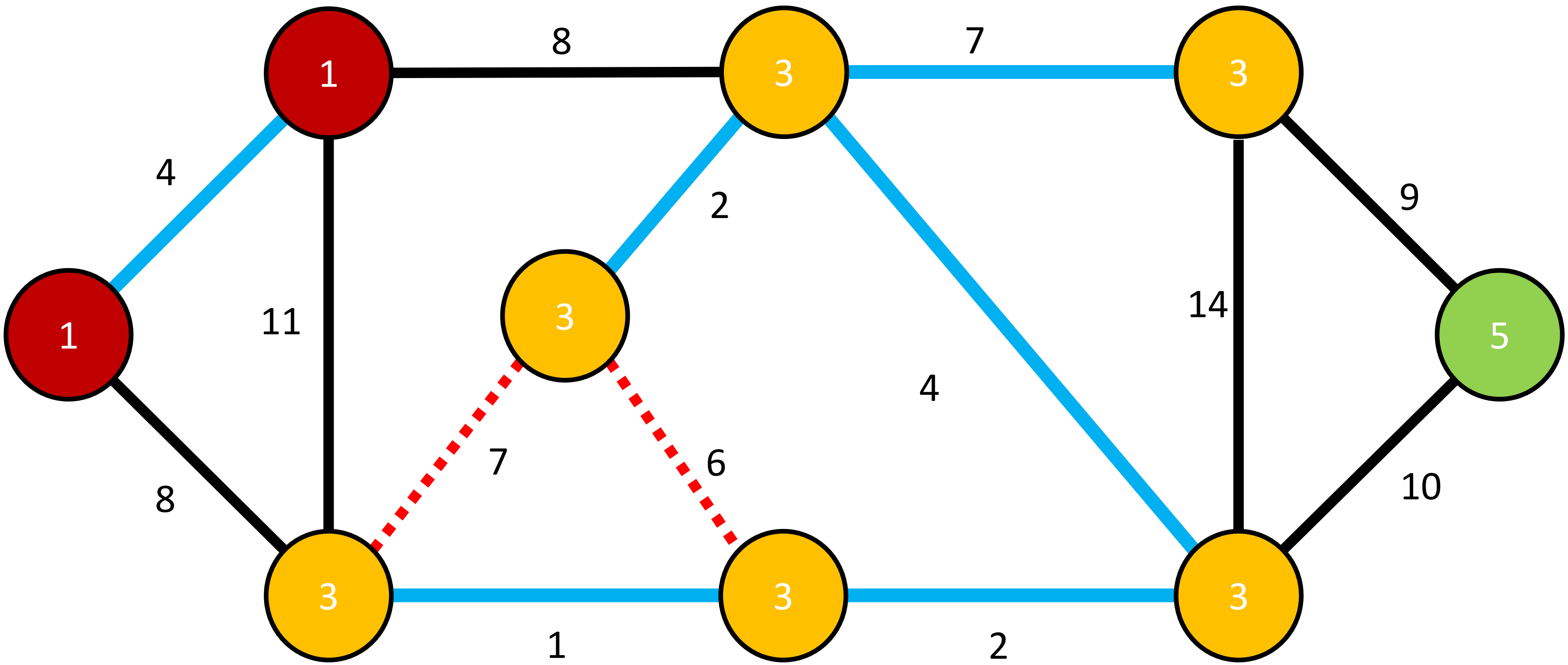
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



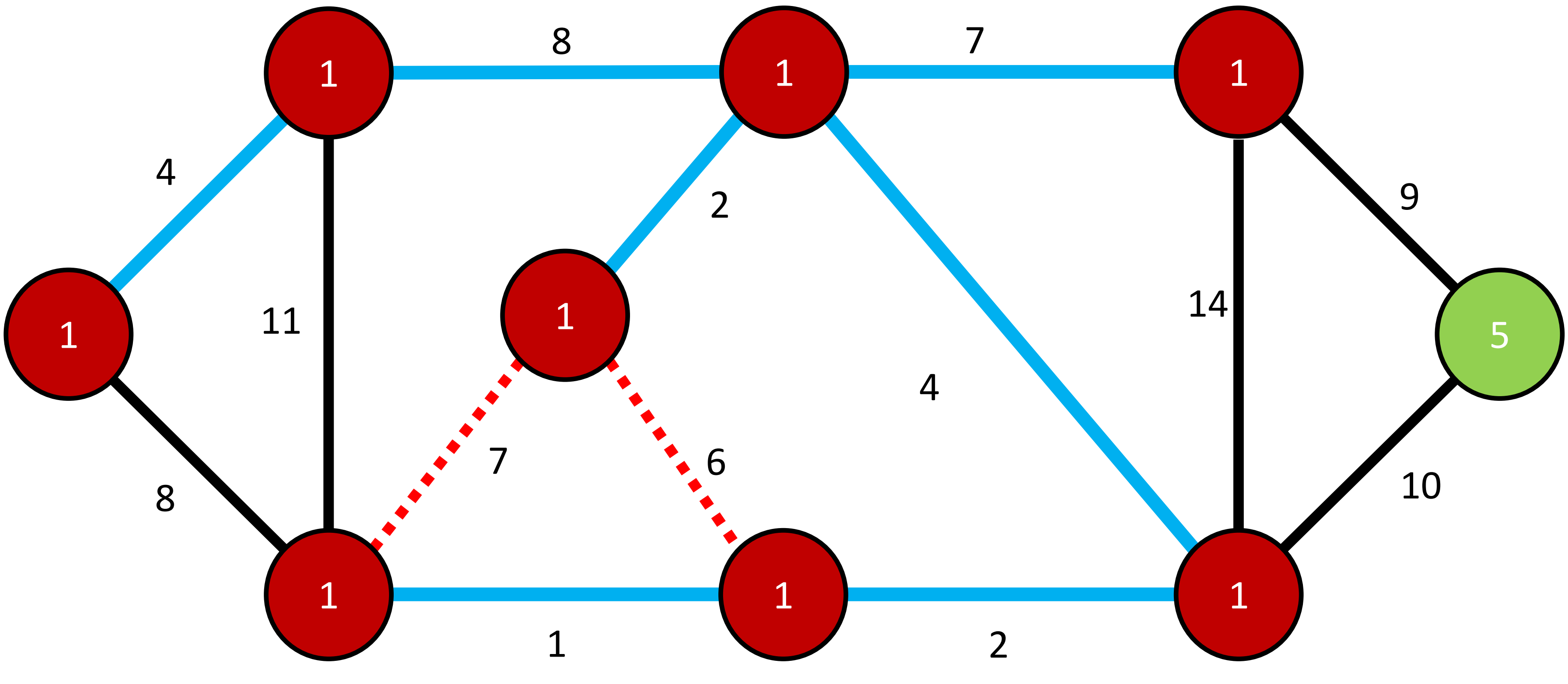
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



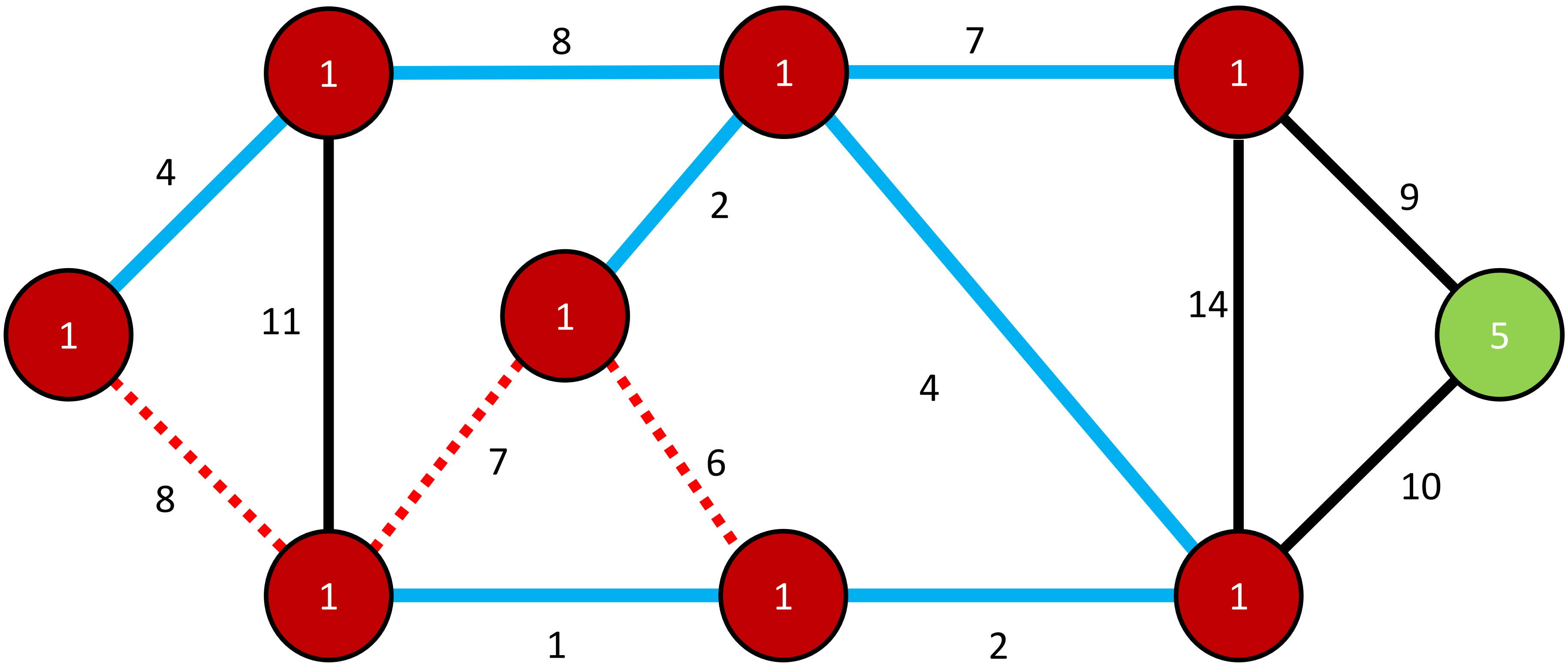
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



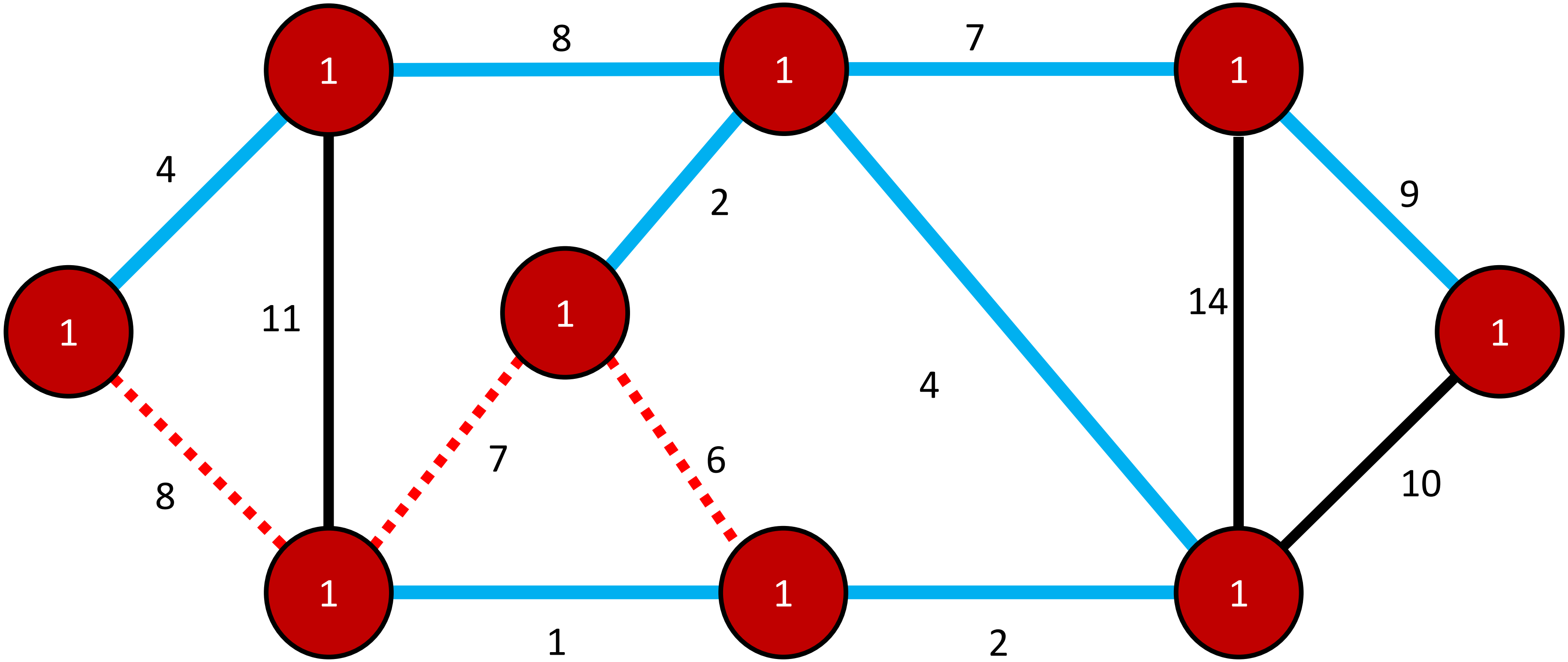
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



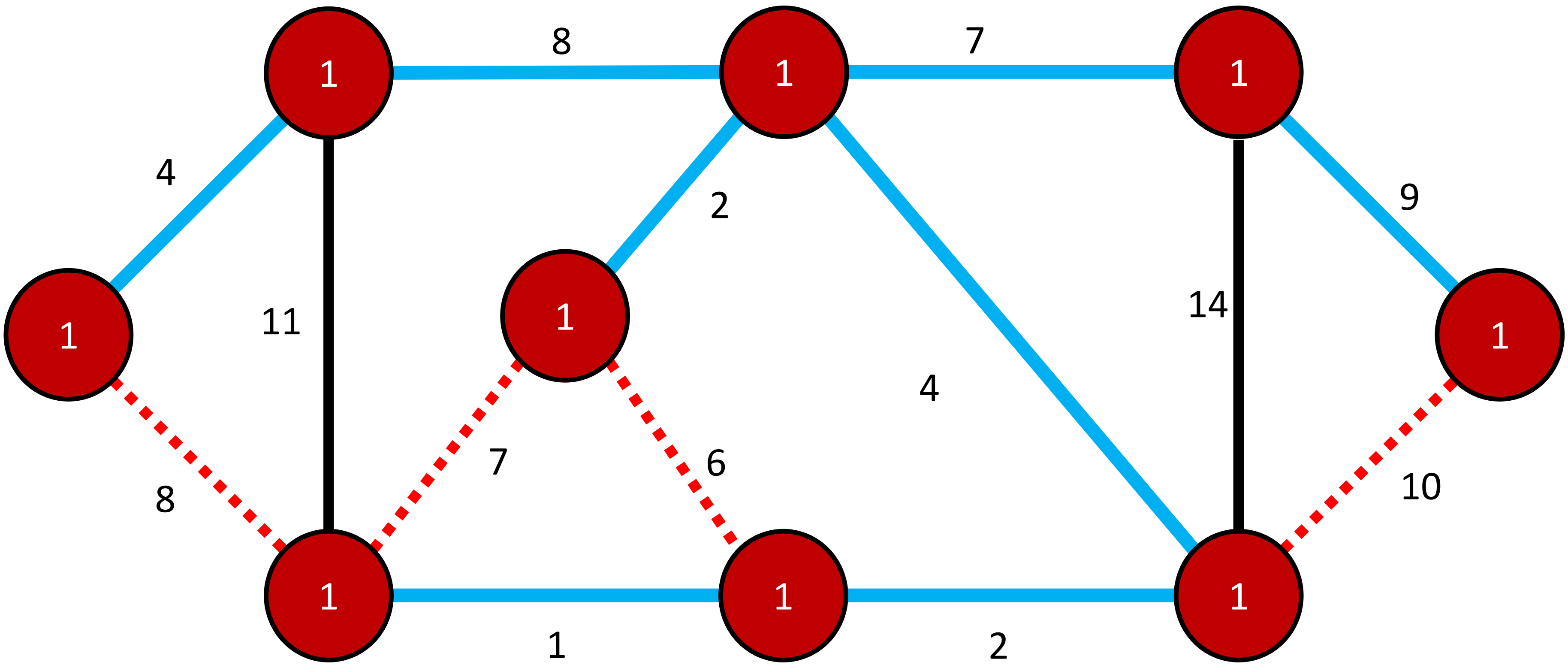
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



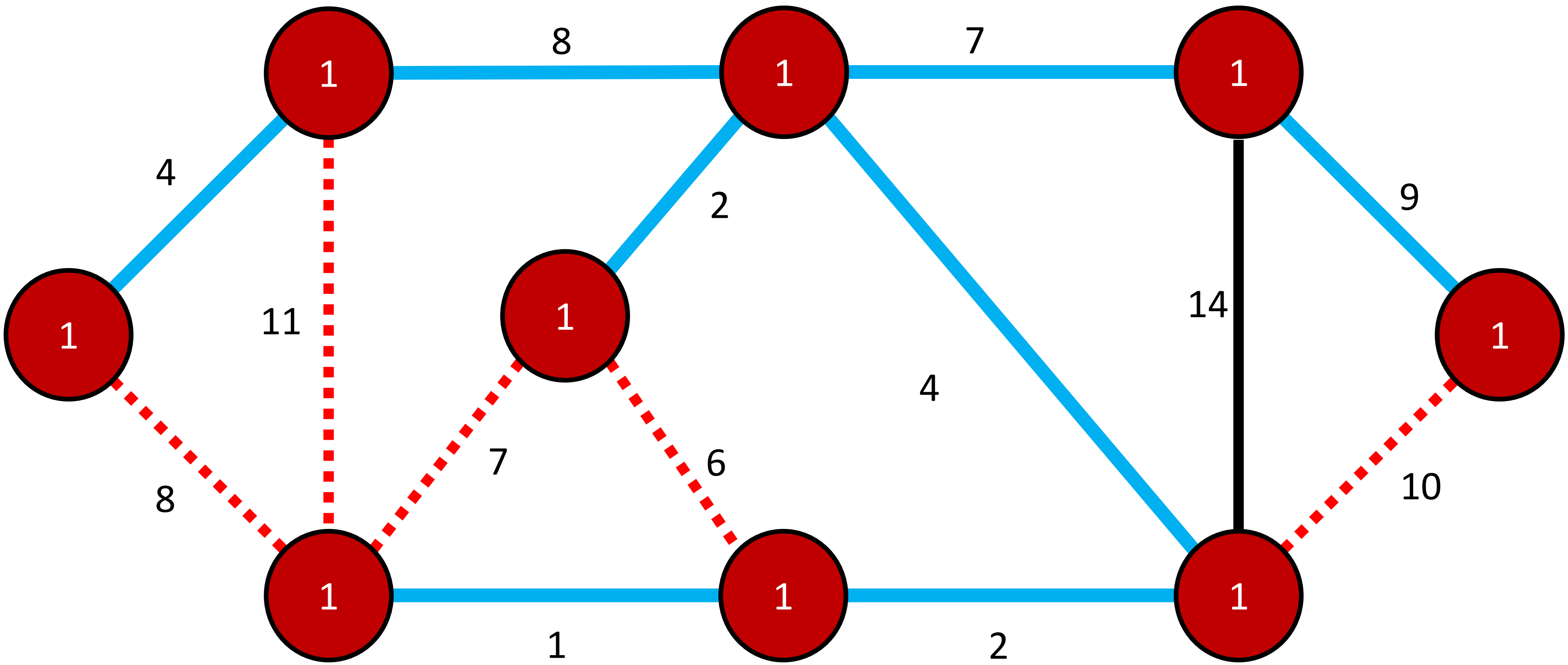
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



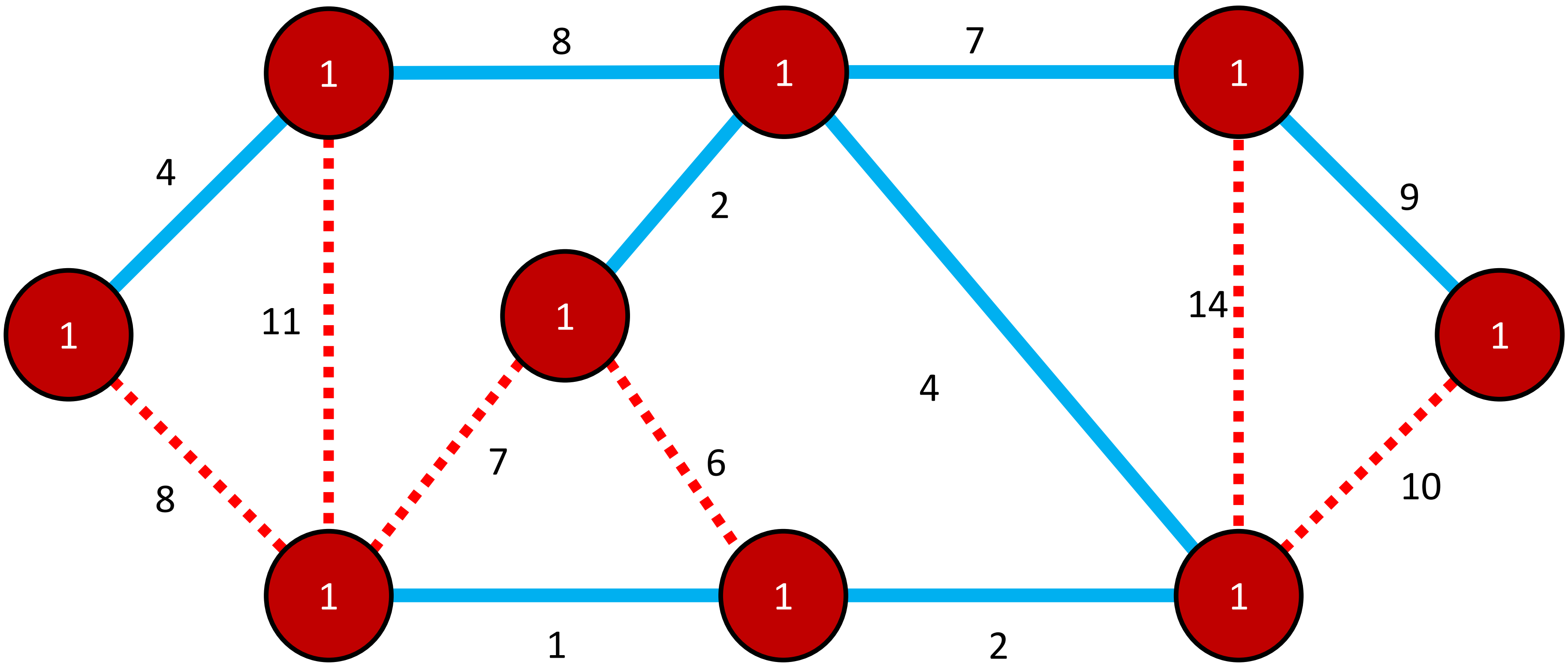
Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



Algoritmo de Kruskal

Evolução do algoritmo: À medida que as arestas são escolhidas, realizamos a fusão das árvores.



Algoritmo de Kruskal

Como implementar a verificação de ciclos?

Algoritmo de Kruskal – nova versão

Entrada: grafo G com $V(G) = \{1,2,\ldots, n\}$

Saída: árvore geradora T de G com custo mínimo

$V(T) \leftarrow V(G)$; $E(T) \leftarrow \varnothing$; ***-- inicialização de T***

para $j = 1,\ldots, n$ *faça*

$\quad\quad \boldsymbol{S_j} \leftarrow \{ j\}$; ***-- inicialização dos n conjuntos disjuntos de vértices das n árvores triviais***

seja $\boldsymbol{e_1, e_2, \ldots, e_m}$ *uma ordenação das arestas de G onde* $peso(\boldsymbol{e_j}) \leq peso(\boldsymbol{e_{j+1}})$, $j = 1,\ldots, m-1$

para $j = 1,\ldots, m$ *faça*

$\quad$ *sejam* $\boldsymbol{S_p}$ *e* $\boldsymbol{S_q}$ *os conjuntos associados aos extremos da aresta* $\boldsymbol{e_j}$ (*onde* $p \leq q$)

$\quad$ *se* $p \neq q$ *então*

$\quad\quad\quad$ *acrescente* $\boldsymbol{e_j}$ *a* $E(T)$

$\quad\quad\quad$ $\boldsymbol{S_p} \leftarrow \boldsymbol{S_p} \cup \boldsymbol{S_q}$

$\quad$ *fim-se*

fim-para

retorne T

Algoritmo de Kruskal

Cálculo da Complexidade

Algoritmo de Kruskal – nova versão

Entrada: grafo G com $V(G) = \{1,2,\ldots, n\}$

Saída: árvore geradora T de G com custo mínimo

$V(T) \leftarrow V(G);\ E(T) \leftarrow \varnothing;$ ***-- inicialização de T***

para $j = 1,\ldots, n$ *faça*

$\boldsymbol{S_j} \leftarrow \{ j\};$ ***-- inicialização dos n conjuntos disjuntos de vértices das n árvores triviais***

seja $\boldsymbol{e_1, e_2, \ldots ,e_m}$ *uma ordenação das arestas de G onde* $peso(\boldsymbol{e_j}) \leq peso(\boldsymbol{e_{j+1}}),\ j = 1,\ldots, m-1$

para $j = 1,\ldots, m$ *faça*

sejam $\boldsymbol{S_p}$ *e* $\boldsymbol{S_q}$ *os conjuntos associados aos extremos da aresta* $\boldsymbol{e_j}$ *(onde* $p \leq q$*)*

se $p \neq q$ *então*

acrescente $\boldsymbol{e_j}$ *a* $E(T)$

$\boldsymbol{S_p} \leftarrow \boldsymbol{S_p} \cup \boldsymbol{S_q}$

fim-se

fim-para

retorne T

Algoritmo de Kruskal

Cálculo da Complexidade

Algoritmo de Kruskal – nova versão

Entrada: grafo G com $V(G) = \{1,2,\ldots, n\}$

Saída: árvore geradora T de G com custo mínimo

$V(T) \leftarrow V(G)$; $E(T) \leftarrow \emptyset$; $\boxed{O(n)}$

para $j = 1,\ldots, n$ *faça*

  $\boldsymbol{S_j} \leftarrow \{ j\}$; ***-- inicialização dos n conjuntos disjuntos de vértices das n árvores triviais***

seja $\boldsymbol{e_1, e_2, \ldots, e_m}$ *uma ordenação das arestas de G onde peso*$(\boldsymbol{e_j}) \leq$ *peso*$(\boldsymbol{e_{j+1}})$, $j = 1,\ldots, m-1$

para $j = 1,\ldots, m$ *faça*

  sejam $\boldsymbol{S_p}$ *e* $\boldsymbol{S_q}$ *os conjuntos associados aos extremos da aresta* $\boldsymbol{e_j}$ (*onde* $p \leq q$)

  se $p \neq q$ *então*

    acrescente $\boldsymbol{e_j}$ *a* $E(T)$

    $\boldsymbol{S_p} \leftarrow \boldsymbol{S_p} \cup \boldsymbol{S_q}$

  fim-se

fim-para

retorne T

Algoritmo de Kruskal

Cálculo da Complexidade

Algoritmo de Kruskal – nova versão

Entrada: grafo G com $V(G) = \{1,2,\ldots, n\}$

Saída: árvore geradora T de G com custo mínimo

$V(T) \leftarrow V(G)$; $E(T) \leftarrow \varnothing$; $O(n)$

para $j = 1,\ldots, n$ *faça*

  $\boldsymbol{S_j} \leftarrow \{ j\}$; $O(n)$

seja $\boldsymbol{e_1, e_2, \ldots ,e_m}$ *uma ordenação das arestas de G onde peso*$(\boldsymbol{e_j}) \leq$ *peso*$(\boldsymbol{e_{j+1}})$, $j = 1,\ldots, m-1$

para $j = 1,\ldots, m$ *faça*

  sejam $\boldsymbol{S_p}$ *e* $\boldsymbol{S_q}$ *os conjuntos associados aos extremos da aresta* $\boldsymbol{e_j}$ (*onde* $p \leq q$)

  se $p \neq q$ *então*

    acrescente $\boldsymbol{e_j}$ *a* $E(T)$

    $\boldsymbol{S_p} \leftarrow \boldsymbol{S_p} \cup \boldsymbol{S_q}$

  fim-se

fim-para

retorne T

Algoritmo de Kruskal

Cálculo da Complexidade

Algoritmo de Kruskal – nova versão

Entrada: grafo G com $V(G) = \{1,2,\ldots, n\}$

Saída: árvore geradora T de G com custo mínimo

$V(T) \leftarrow V(G);\ E(T) \leftarrow \varnothing;$ $\boxed{O(n)}$

para $j = 1,\ldots, n$ *faça*

  $\boldsymbol{S_j} \leftarrow \{\,j\};$ $\boxed{O(n)}$

seja $\boldsymbol{e_1, e_2, \ldots, e_m}$ *uma ordenação das arestas de G onde peso*$(\boldsymbol{e_j}) \leq$ *peso*$(\boldsymbol{e_{j+1}})$ $\boxed{O(m \log m)}$

para $j = 1,\ldots, m$ *faça*

  sejam $\boldsymbol{S_p}$ *e* $\boldsymbol{S_q}$ *os conjuntos associados aos extremos da aresta* $\boldsymbol{e_j}$ (*onde* $p \leq q$)

  se $p \neq q$ *então*

    acrescente $\boldsymbol{e_j}$ *a* $E(T)$

    $\boldsymbol{S_p} \leftarrow \boldsymbol{S_p} \cup \boldsymbol{S_q}$

  fim-se

fim-para

retorne T

Algoritmo de Kruskal

Cálculo da Complexidade

Algoritmo de Kruskal – nova versão

Entrada: grafo G com $V(G) = \{1,2,\ldots, n\}$

Saída: árvore geradora T de G com custo mínimo

$V(T) \leftarrow V(G)$; $E(T) \leftarrow \emptyset$; $\boxed{O(n)}$

para $j = 1,\ldots, n$ *faça* $\boxed{O(n)}$

  $\boldsymbol{S_j} \leftarrow \{ j\}$;

seja $\boldsymbol{e_1, e_2, \ldots, e_m}$ *uma ordenação das arestas de G onde peso*$(\boldsymbol{e_j}) \leq$ *peso*$(\boldsymbol{e_{j+1}})$ $\boxed{O(m \log m)}$

para $j = 1,\ldots, m$ *faça*

  sejam $\boldsymbol{S_p}$ *e* $\boldsymbol{S_q}$ *os conjuntos associados aos extremos da aresta* $\boldsymbol{e_j}$ (*onde* $p \leq q$)

  se $p \neq q$ *então*

    acrescente $\boldsymbol{e_j}$ *a* $E(T)$ $\boxed{O(1)}$

    $\boldsymbol{S_p} \leftarrow \boldsymbol{S_p} \cup \boldsymbol{S_q}$

  fim-se

fim-para

retorne T

Algoritmo de Kruskal

Cálculo da Complexidade

Algoritmo de Kruskal – nova versão

Entrada: grafo G com $V(G) = \{1,2,\ldots, n\}$

Saída: árvore geradora T de G com custo mínimo

$V(T) \leftarrow V(G)$; $E(T) \leftarrow \varnothing$; $\boxed{O(n)}$

para $j = 1,\ldots, n$ *faça* $\boxed{O(n)}$

$\quad \boldsymbol{S_j} \leftarrow \{ j\}$;

seja $\boldsymbol{e_1, e_2, \ldots, e_m}$ *uma ordenação das arestas de G onde peso(*$\boldsymbol{e_j}$*)* $\leq$ *peso(*$\boldsymbol{e_{j+1}}$*)* $\boxed{O(m \log m)}$

para $j = 1,\ldots, m$ *faça*

$\quad$ *sejam* $\boldsymbol{S_p}$ *e* $\boldsymbol{S_q}$ *os conjuntos associados aos extremos da aresta* $\boldsymbol{e_j}$ (*onde* $p \leq q$)

$\quad$ *se* $p \neq q$ *então*

$\quad\quad$ *acrescente* $\boldsymbol{e_j}$ *a* $E(T)$ $\boxed{O(1)}$

$\quad\quad$ $\boldsymbol{S_p} \leftarrow \boldsymbol{S_p} \cup \boldsymbol{S_q}$

$\quad$ *fim-se*

fim-para

retorne T

$\boxed{\text{Total do “para”: } O(m + n \log n)}$

Algoritmo de Kruskal

Cálculo da Complexidade

Algoritmo de Kruskal – nova versão

Entrada: grafo G com $V(G) = \{1,2,\ldots, n\}$

Saída: árvore geradora T de G com custo mínimo

$V(T) \leftarrow V(G)$; $E(T) \leftarrow \varnothing$; $\boxed{O(n)}$

para $j = 1,\ldots, n$ *faça*

  $\boldsymbol{S_j} \leftarrow \{ j\}$; $\boxed{O(n)}$

seja $\boldsymbol{e_1, e_2, \ldots, e_m}$ *uma ordenação das arestas de G onde* $peso(\boldsymbol{e_j}) \leq peso(\boldsymbol{e_{j+1}})$ $\boxed{O(m \log m)}$

para $j = 1,\ldots, m$ *faça*

  sejam $\boldsymbol{S_p}$ *e* $\boldsymbol{S_q}$ *os conjuntos associados aos extremos da aresta* $\boldsymbol{e_j}$ (*onde* $p \leq q$)

  se $p \neq q$ *então*

    acrescente $\boldsymbol{e_j}$ *a* $E(T)$ $\boxed{O(1)}$

    $\boldsymbol{S_p} \leftarrow \boldsymbol{S_p} \cup \boldsymbol{S_q}$

  fim-se

fim-para

retorne T

$\boxed{\textit{Total do “para”: } O(m + n \log n)}$

$\boxed{\textit{Total do algoritmo: } O(m \log m) = O(m \log n)}$

Algoritmo de Prim

O Algoritmo de Prim também utiliza o método guloso, mas tem um princípio de funcionamento diferente:

- no algoritmo de Kruskal, a solução parcial é uma **floresta**;
- já no algoritmo de Prim, é uma **árvore** (grafo conexo).

Algoritmo de Prim

Dado o grafo G, o algoritmo constrói uma árvore geradora T de G com custo mínimo

escolher um vértice qualquer **v** *em* $V(G)$; ***-- vértice escolhido para ser a raiz de T***

$V(T) \leftarrow \{v\}$; $E(T) \leftarrow \emptyset$; ***-- inicialização de T***

<u>*para*</u> $j = 1, \ldots, n-1$ <u>*faça*</u>

  seja $\mathbf{e = xy}$ *a aresta de peso mínimo com um extremo* **x** *em* $V(T)$ *e outro* **y** *em* $V(G) \setminus V(T)$

  acrescente **e** *a* $E(T)$

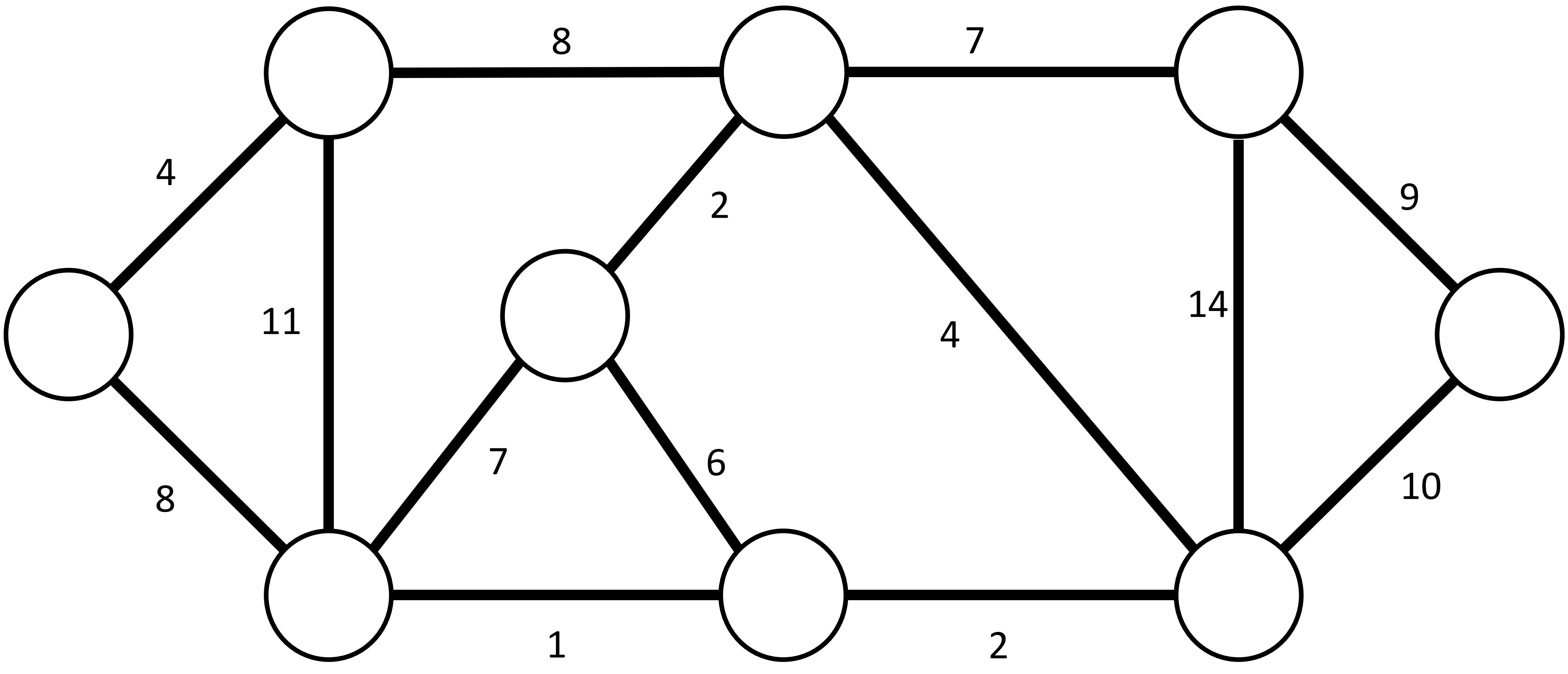
  acrescente **y** *a* $V(T)$

<u>*fim-para*</u>

<u>*retorne*</u> T

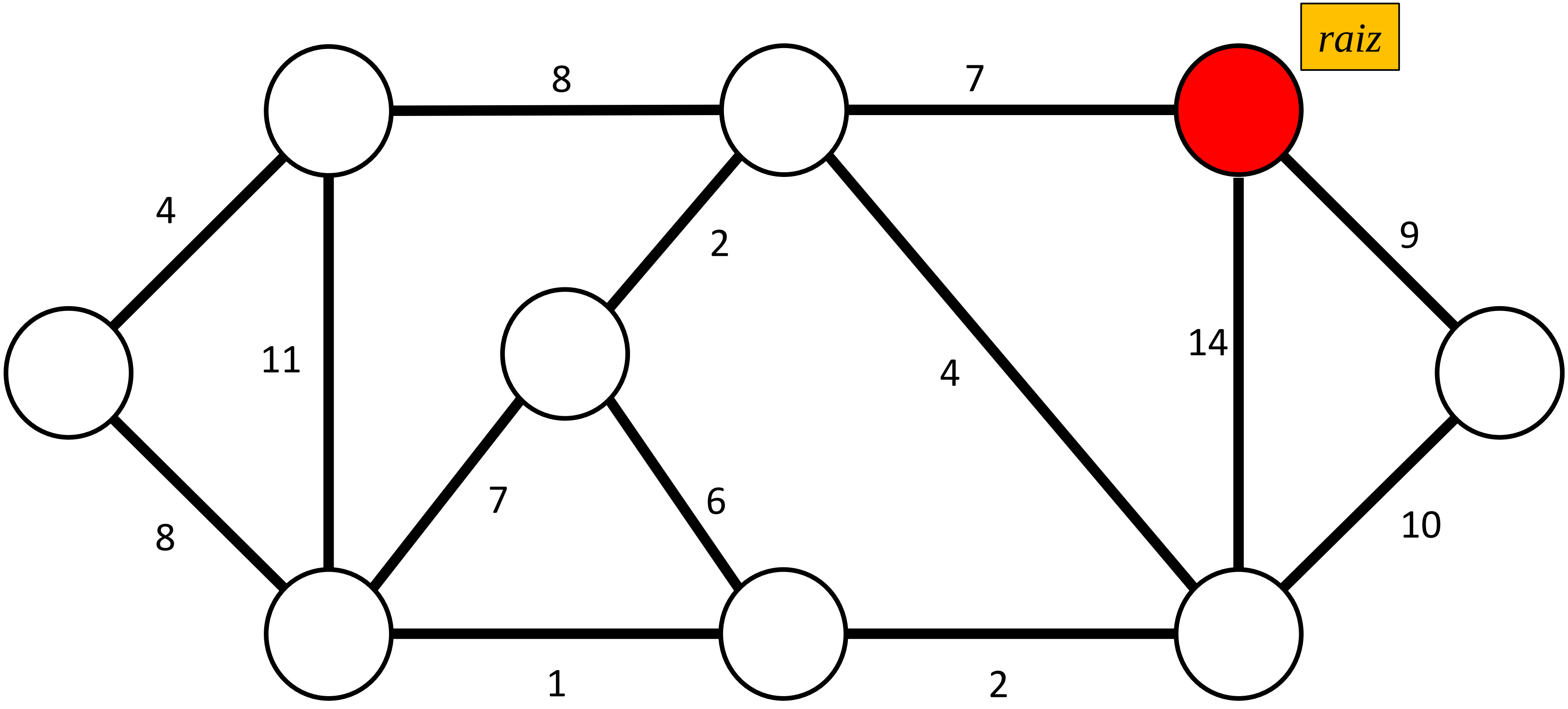
Algoritmo de Prim

Exemplo: Simulação da execução do algoritmo.



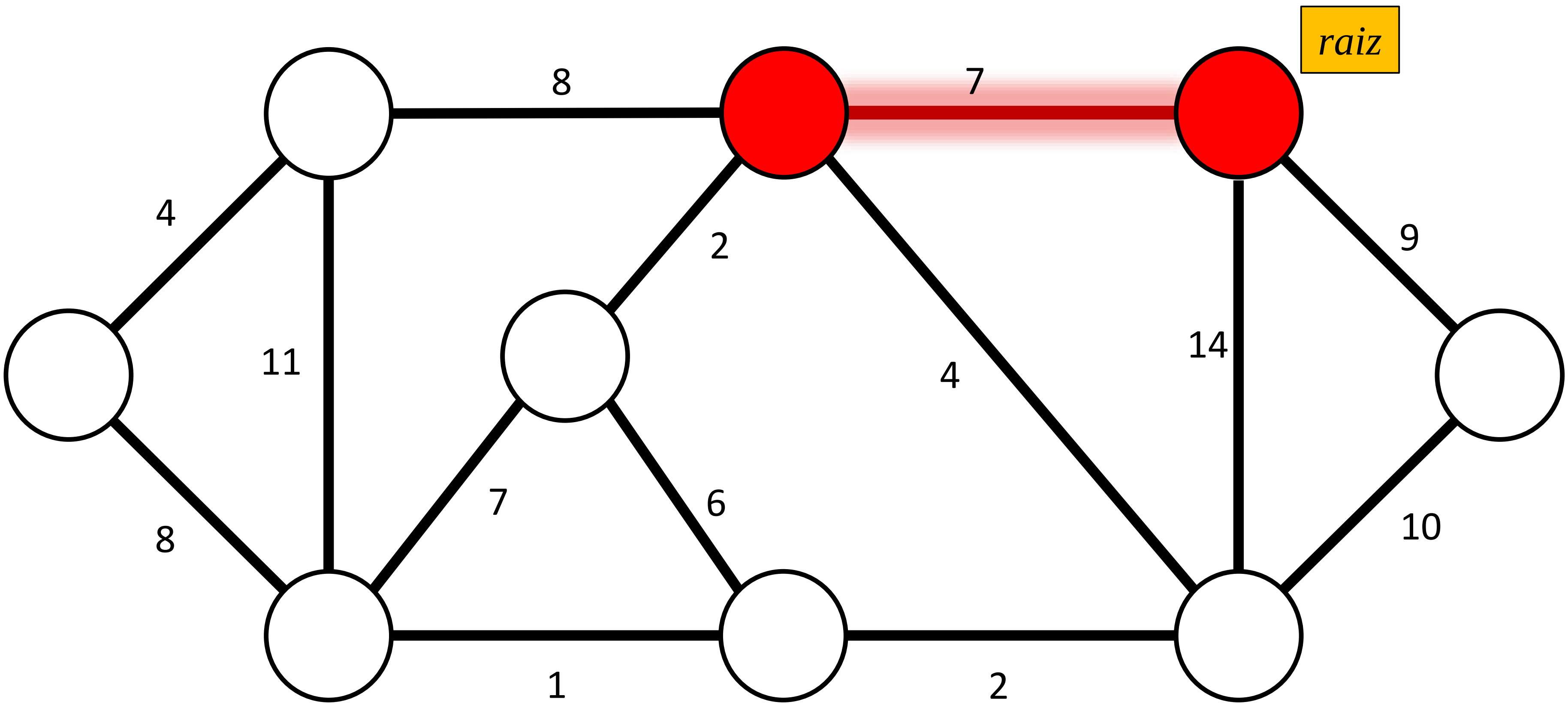
Algoritmo de Prim

Exemplo: Simulação da execução do algoritmo.



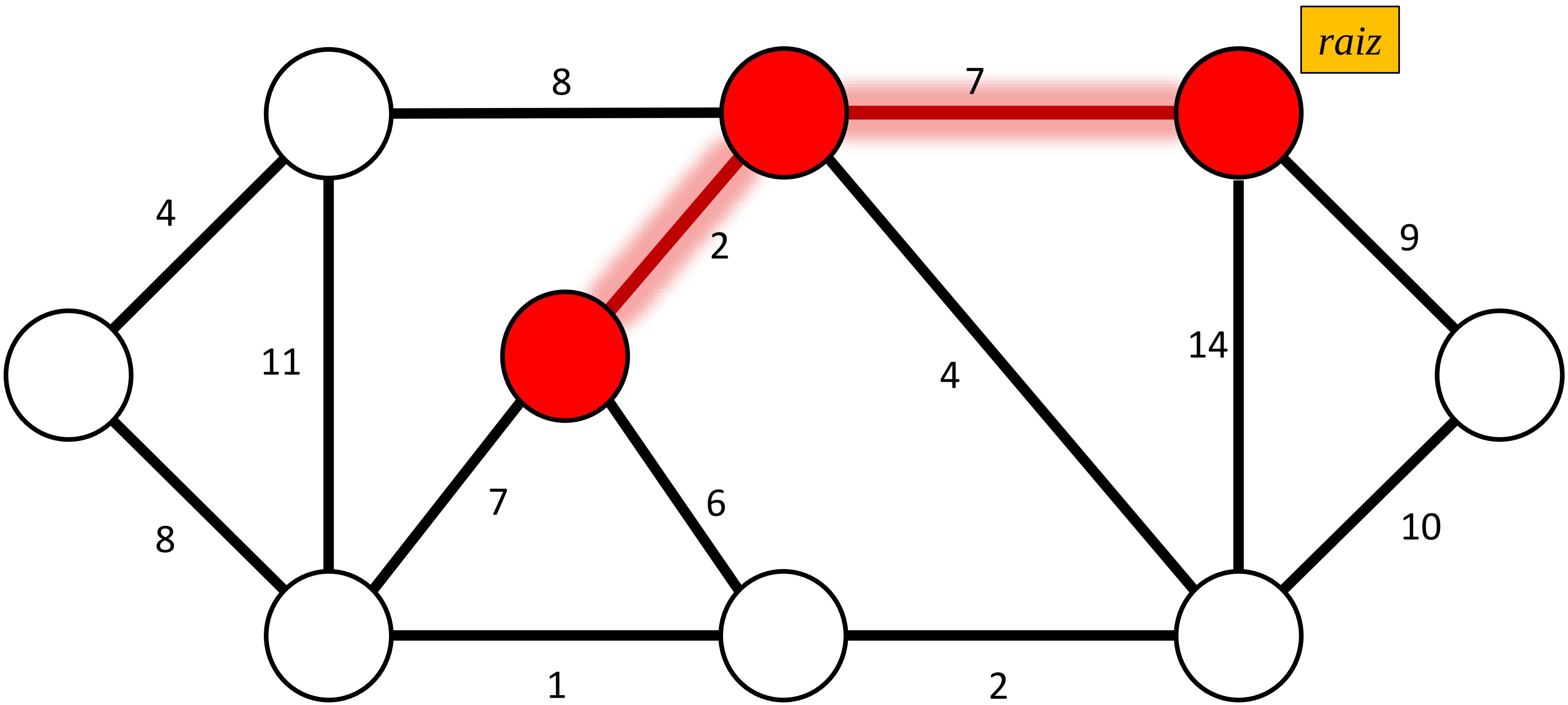
Algoritmo de Prim

Exemplo: Simulação da execução do algoritmo.



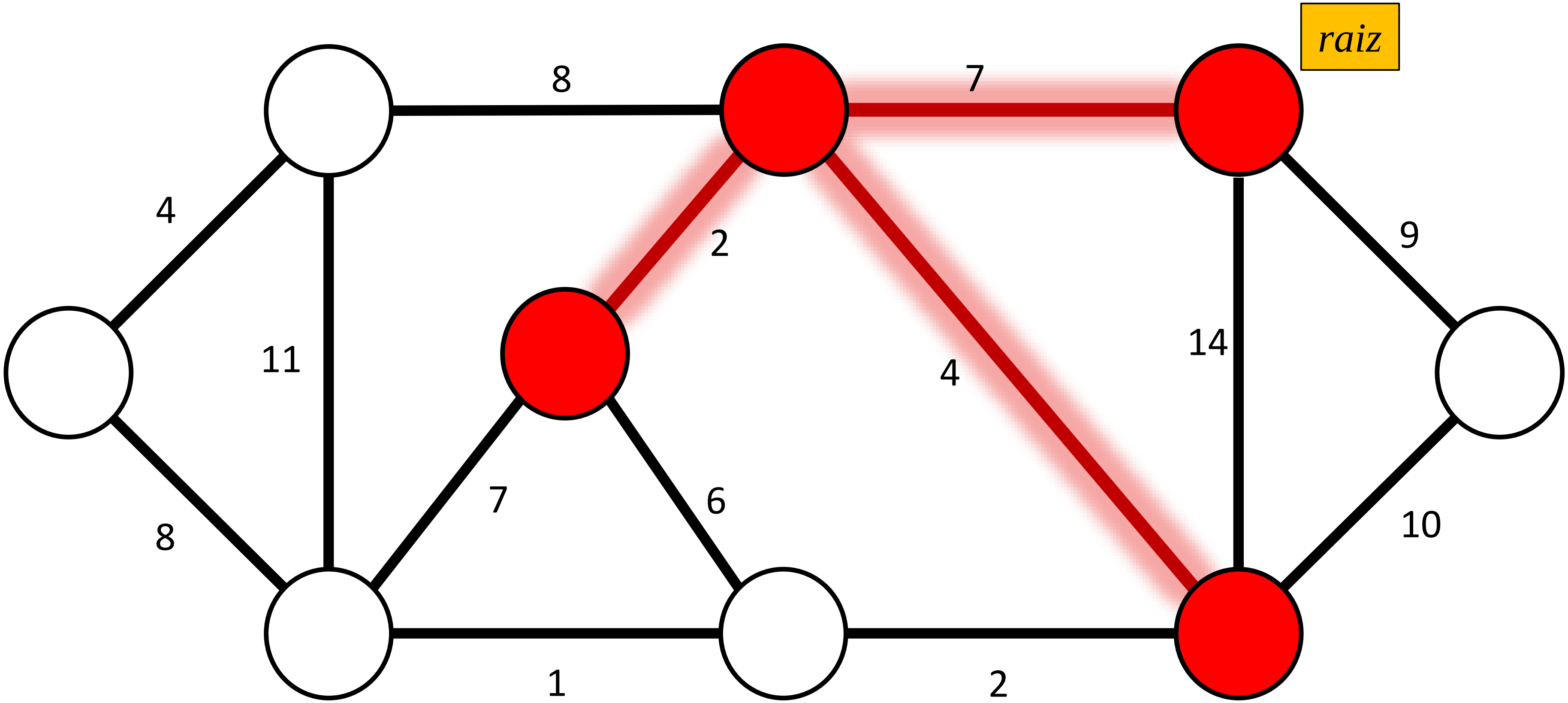
Algoritmo de Prim

Exemplo: Simulação da execução do algoritmo.



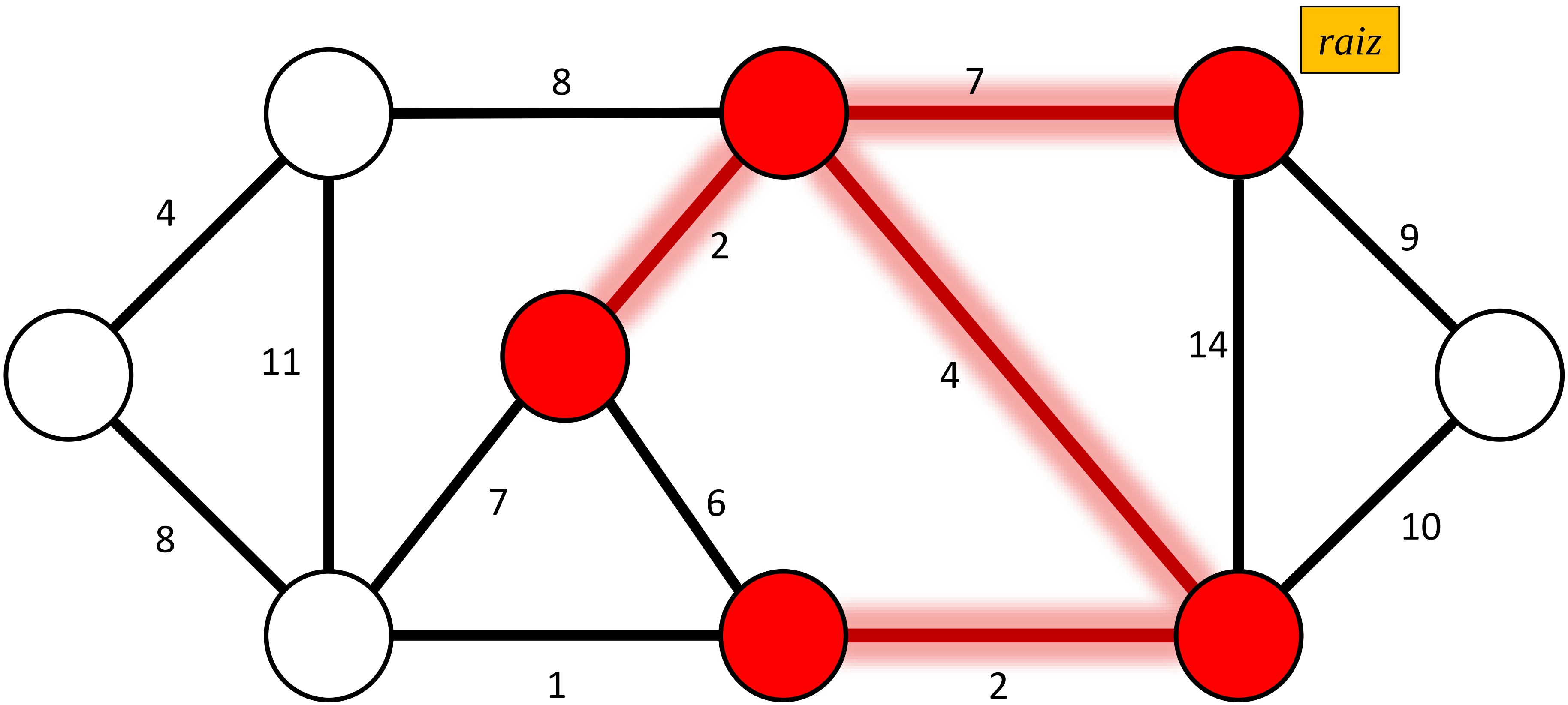
Algoritmo de Prim

Exemplo: Simulação da execução do algoritmo.



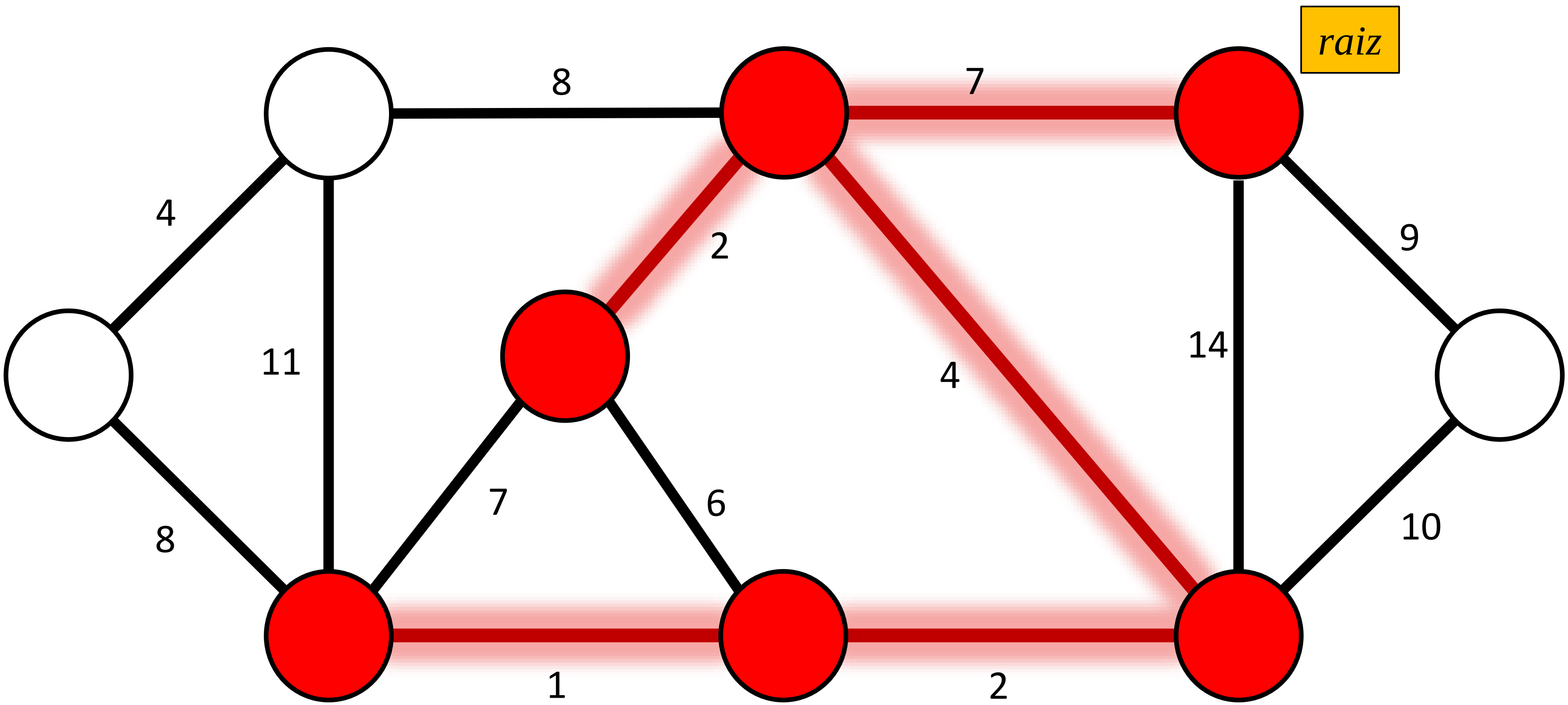
Algoritmo de Prim

Exemplo: Simulação da execução do algoritmo.



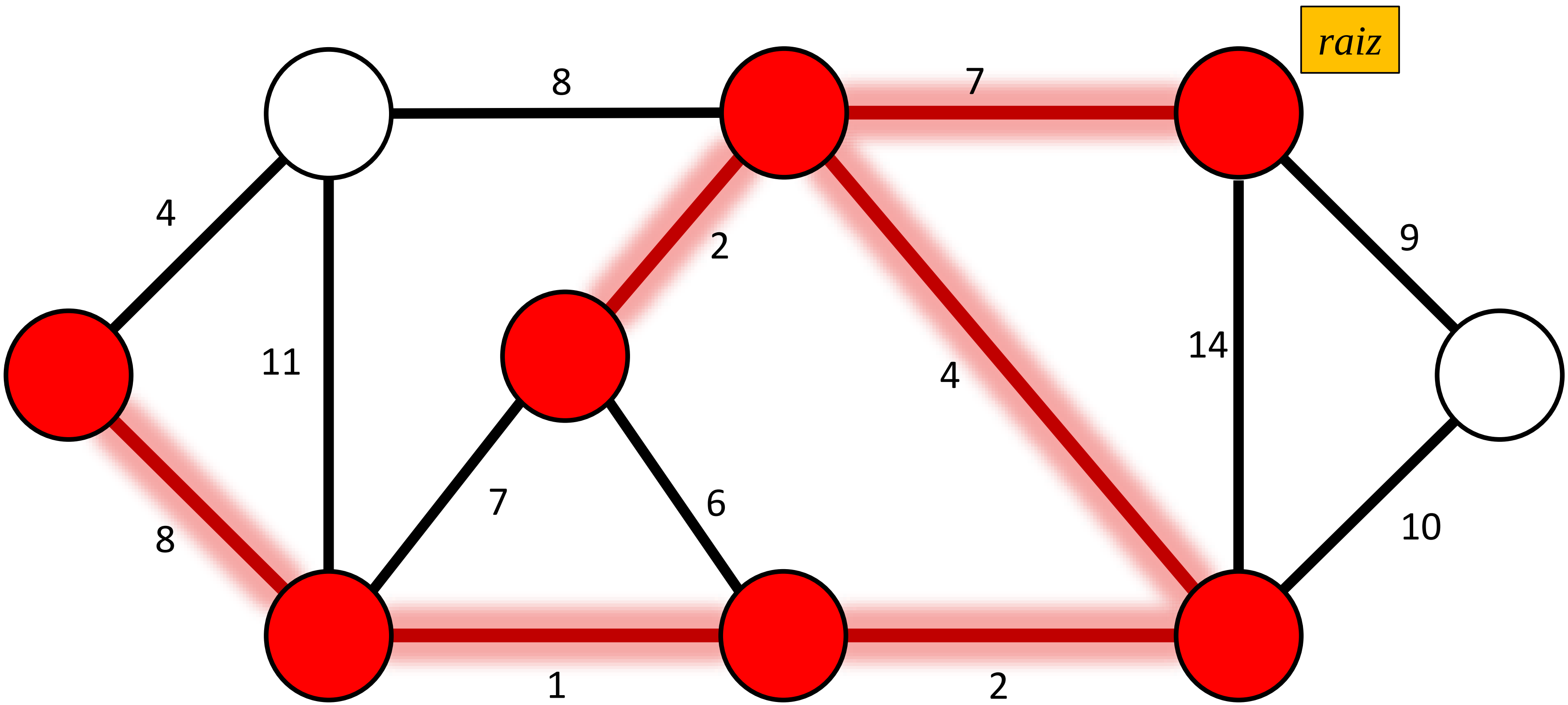
Algoritmo de Prim

Exemplo: Simulação da execução do algoritmo.



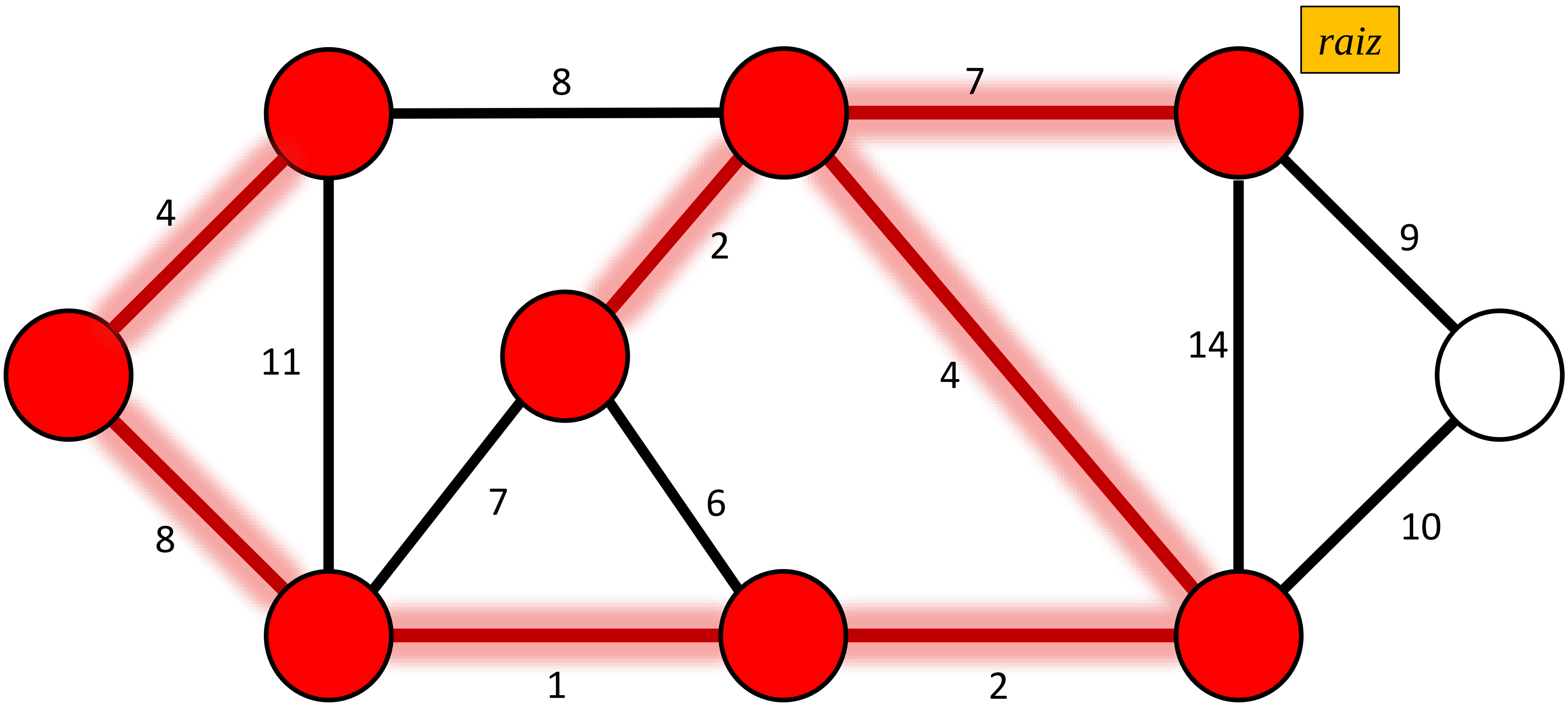
Algoritmo de Prim

Exemplo: Simulação da execução do algoritmo.



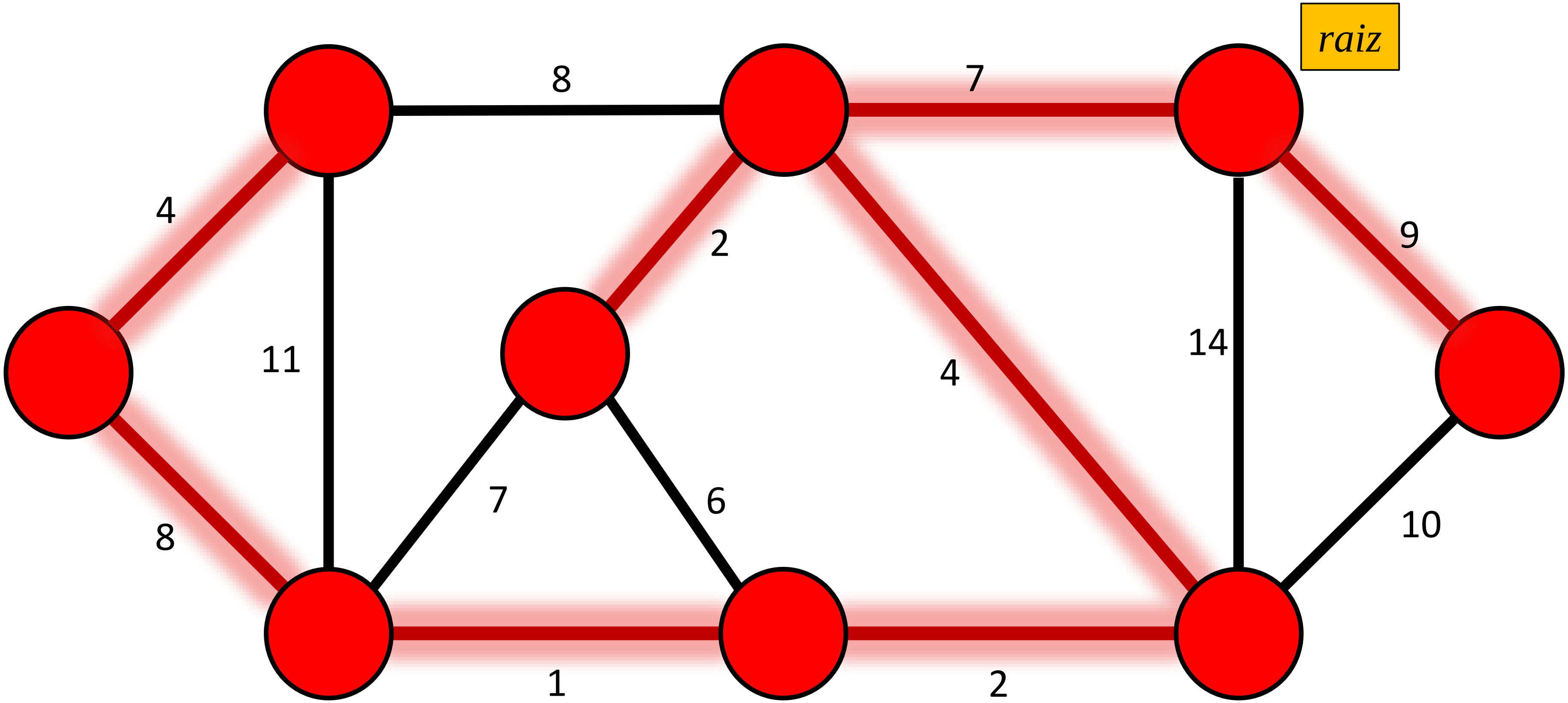
Algoritmo de Prim

Exemplo: Simulação da execução do algoritmo.



Algoritmo de Prim

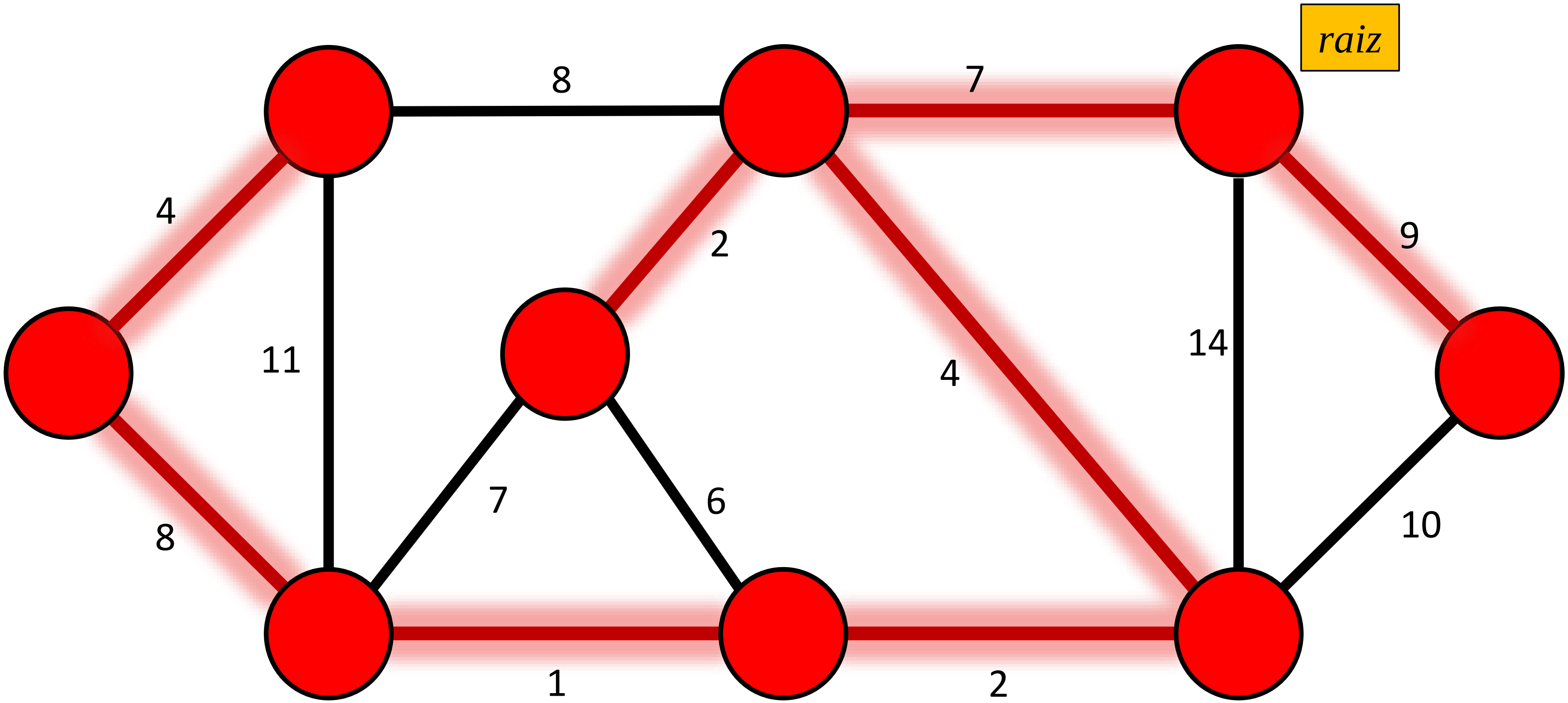
Exemplo: Simulação da execução do algoritmo.



Algoritmo de Prim

Exemplo: Simulação da execução do algoritmo.

FINAL: custo da árvore geradora é 37

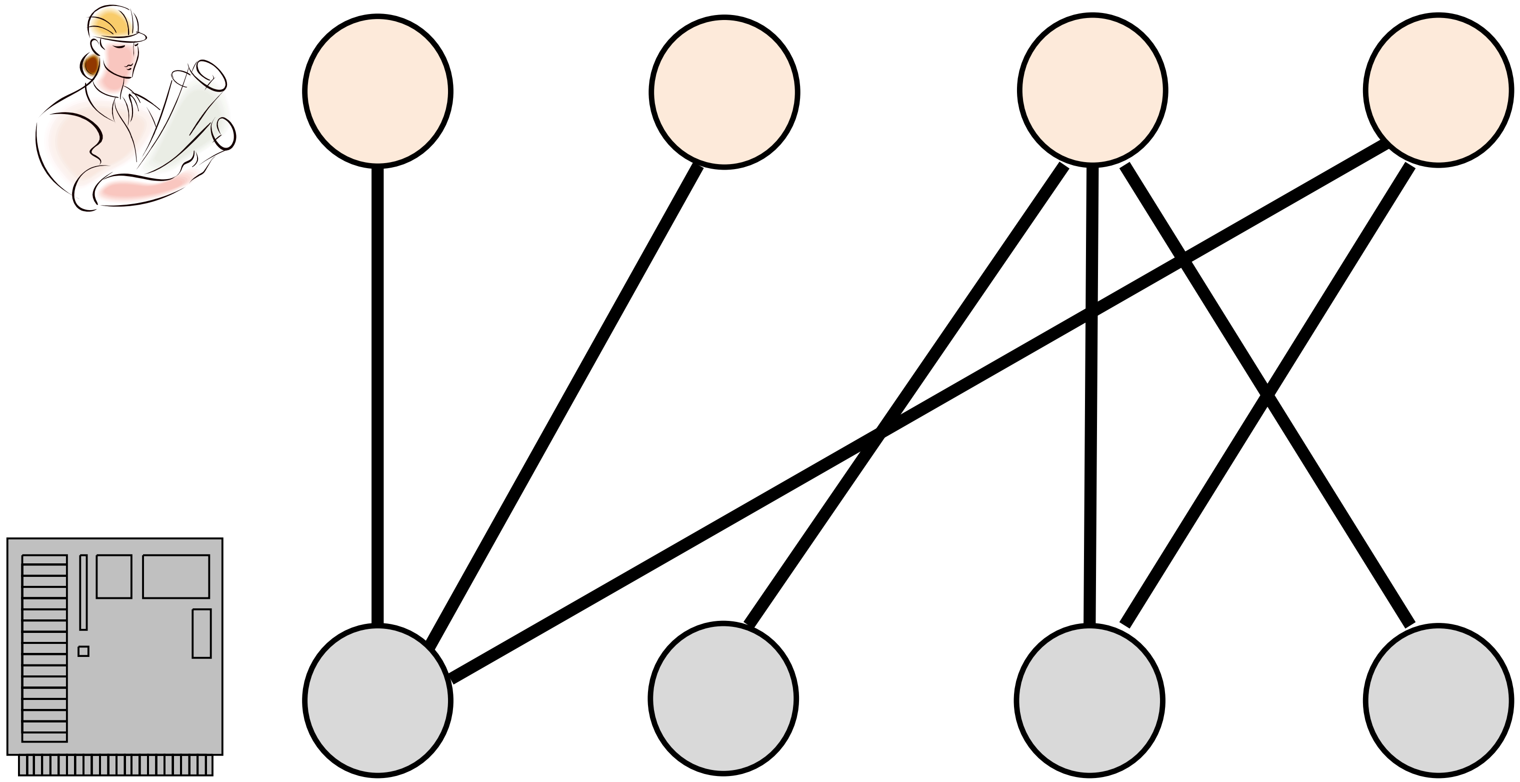


Emparelhamentos

Problema 1: Para cada operário de uma fábrica existe uma lista das máquinas que ele está habilitado a operar. Encontre uma alocação que maximize o número de pares da forma "operário ***i*** opera a máquina ***j***".

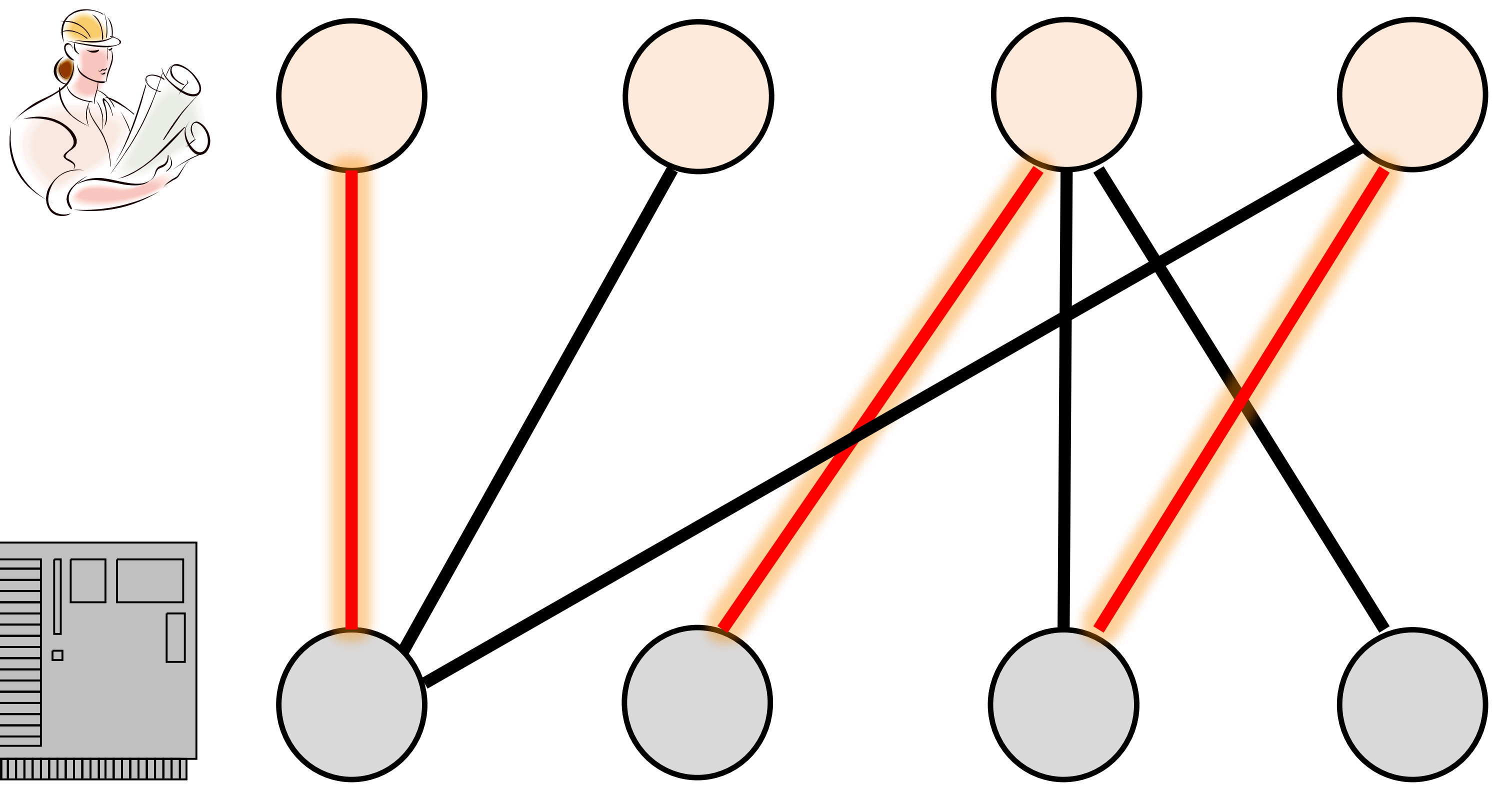
Emparelhamentos

Problema 1: Para cada operário de uma fábrica existe uma lista das máquinas que ele está habilitado a operar. Encontre uma alocação que maximize o número de pares da forma "operário ***i*** opera a máquina ***j***".



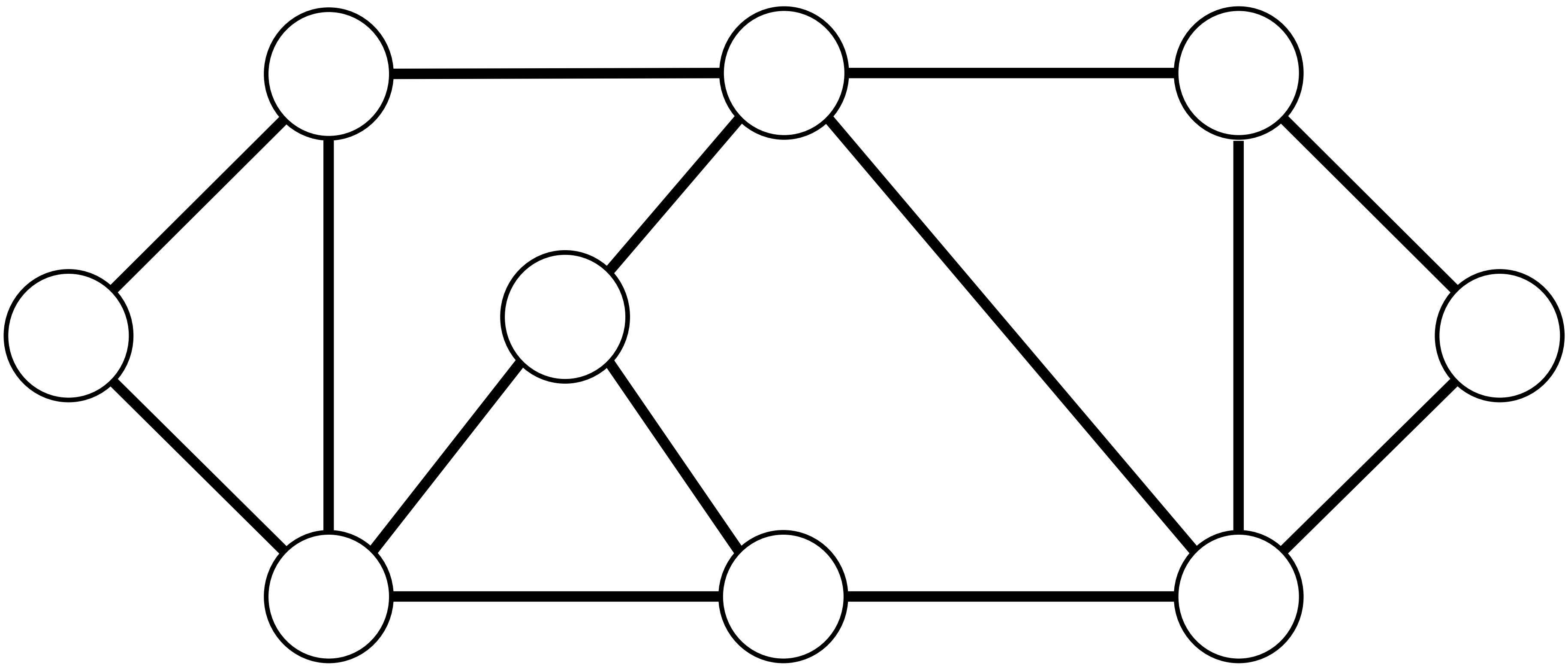
Emparelhamentos

Problema 1: Para cada operário de uma fábrica existe uma lista das máquinas que ele está habilitado a operar. Encontre uma alocação que maximize o número de pares da forma "operário ***i*** opera a máquina ***j***".



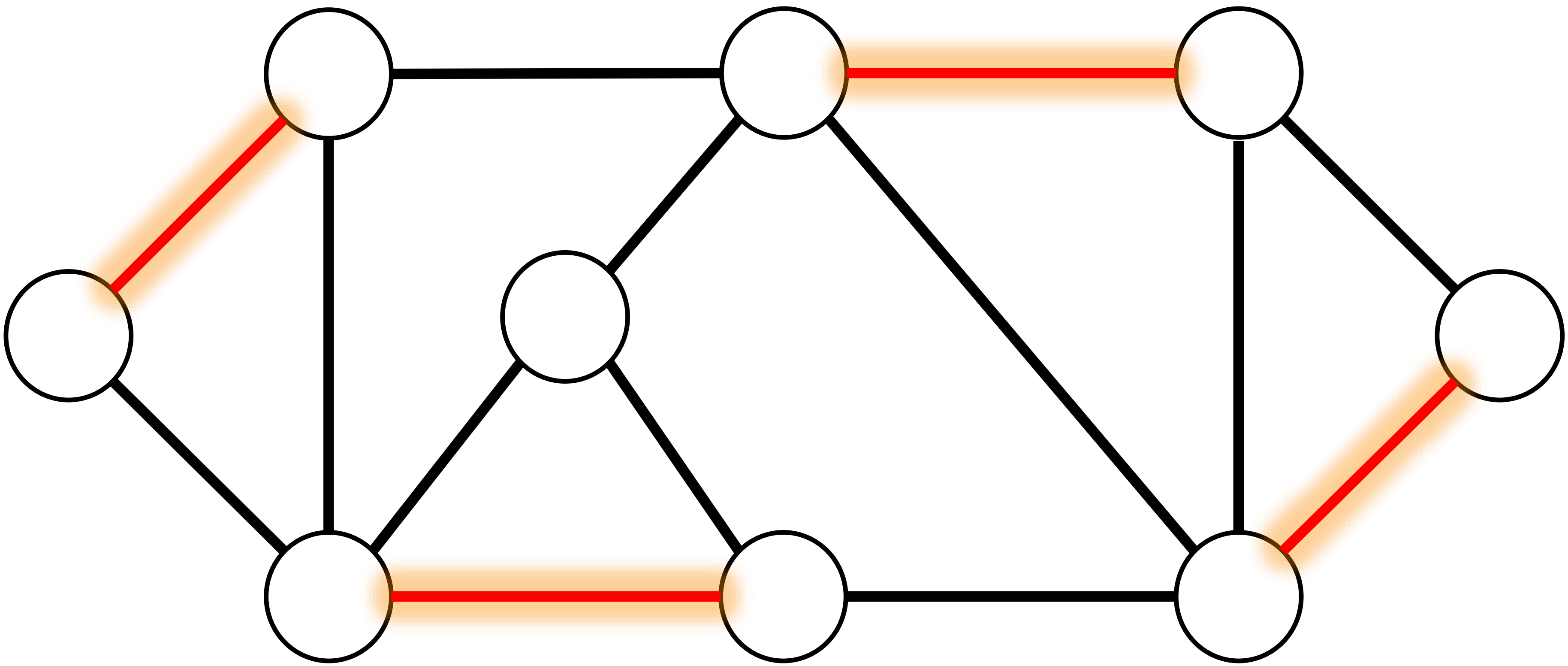
Emparelhamentos

Problema 2: Deseja-se estabelecer um pareamento de processadores em uma rede. Encontre o maior número de pares de processadores "parceiros".



Emparelhamentos

Problema 2: Deseja-se estabelecer um pareamento de processadores em uma rede. Encontre o maior número de pares de processadores "parceiros".

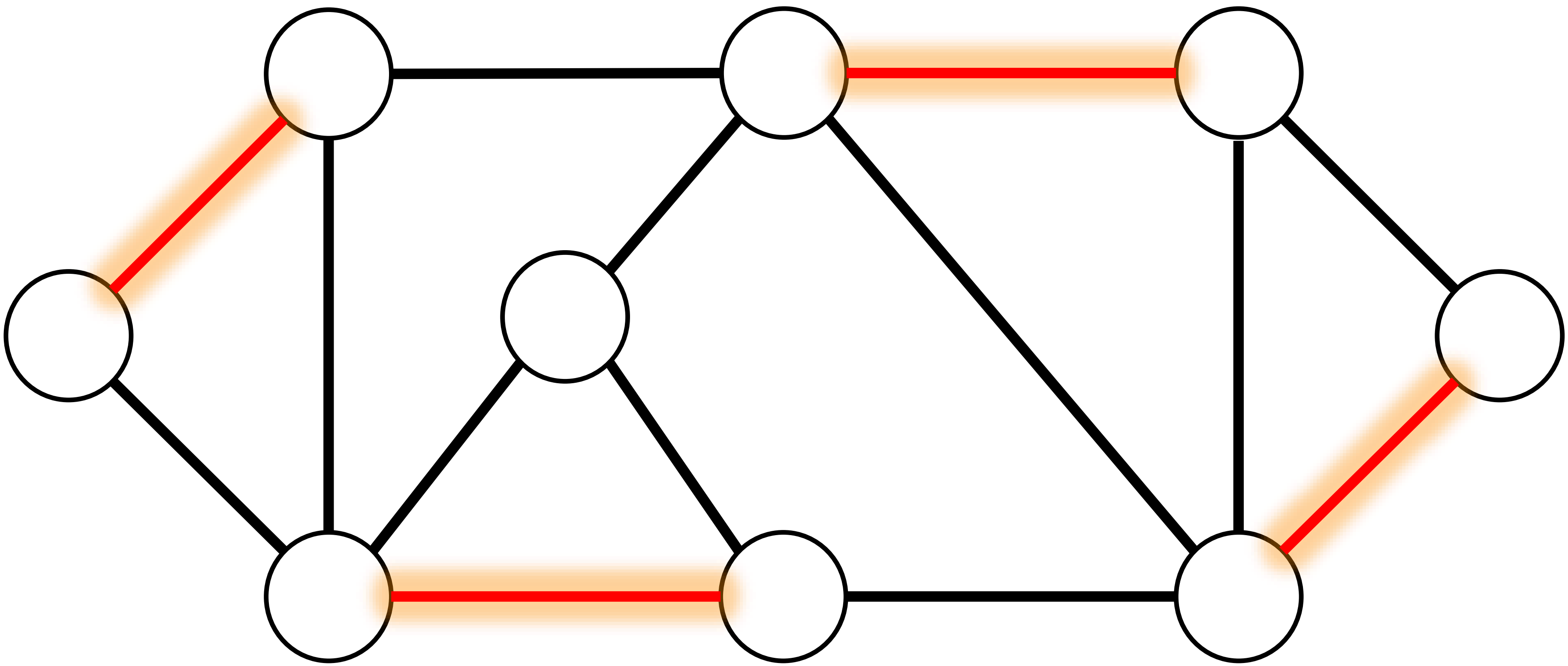


Emparelhamentos

Definição: Um ***emparelhamento*** em um grafo G é um conjunto de arestas que não têm extremos em comum.

Emparelhamentos

Definição: Um ***emparelhamento*** em um grafo *G* é um conjunto de arestas que não têm extremos em comum.

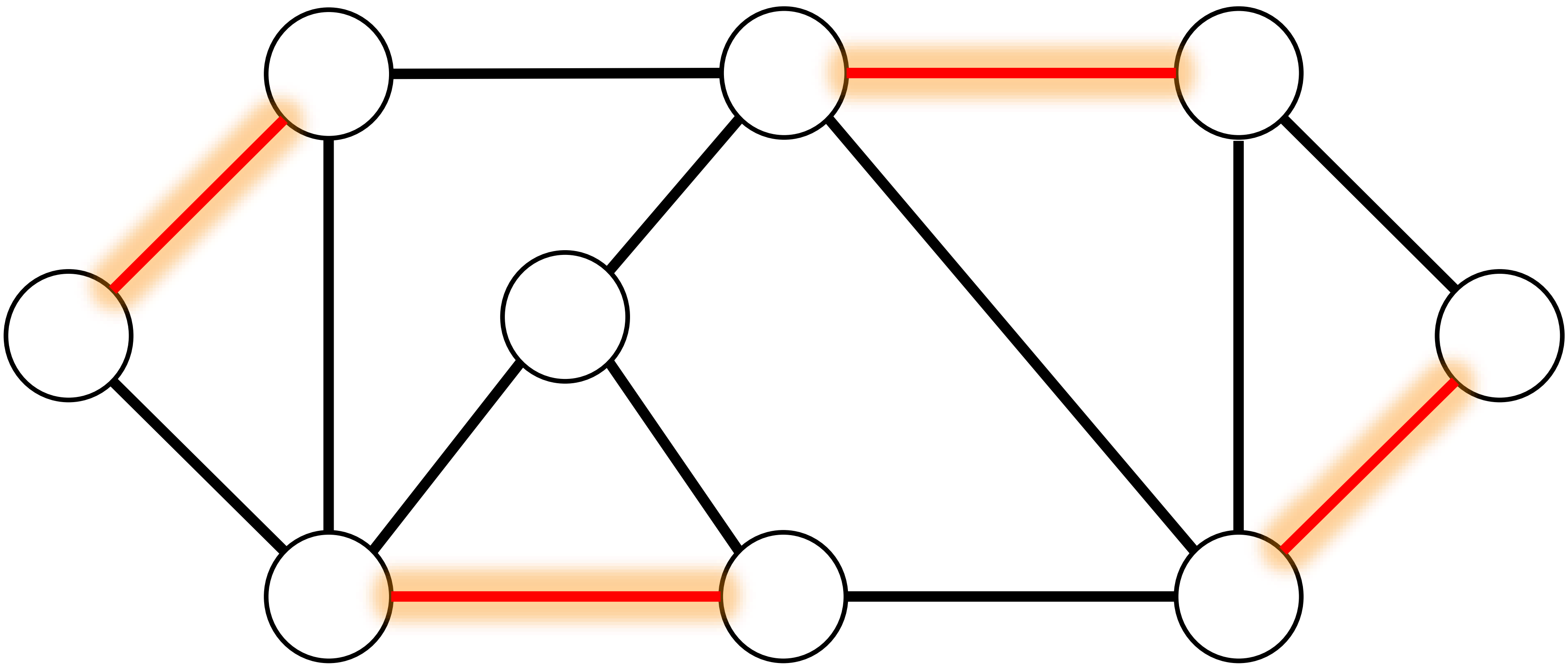


Emparelhamentos

Definição: Um emparelhamento em um grafo G é ***máximo*** se possui o maior número possível de arestas. (Obs: pode haver um número exponencial de emparelhamentos máximos!)

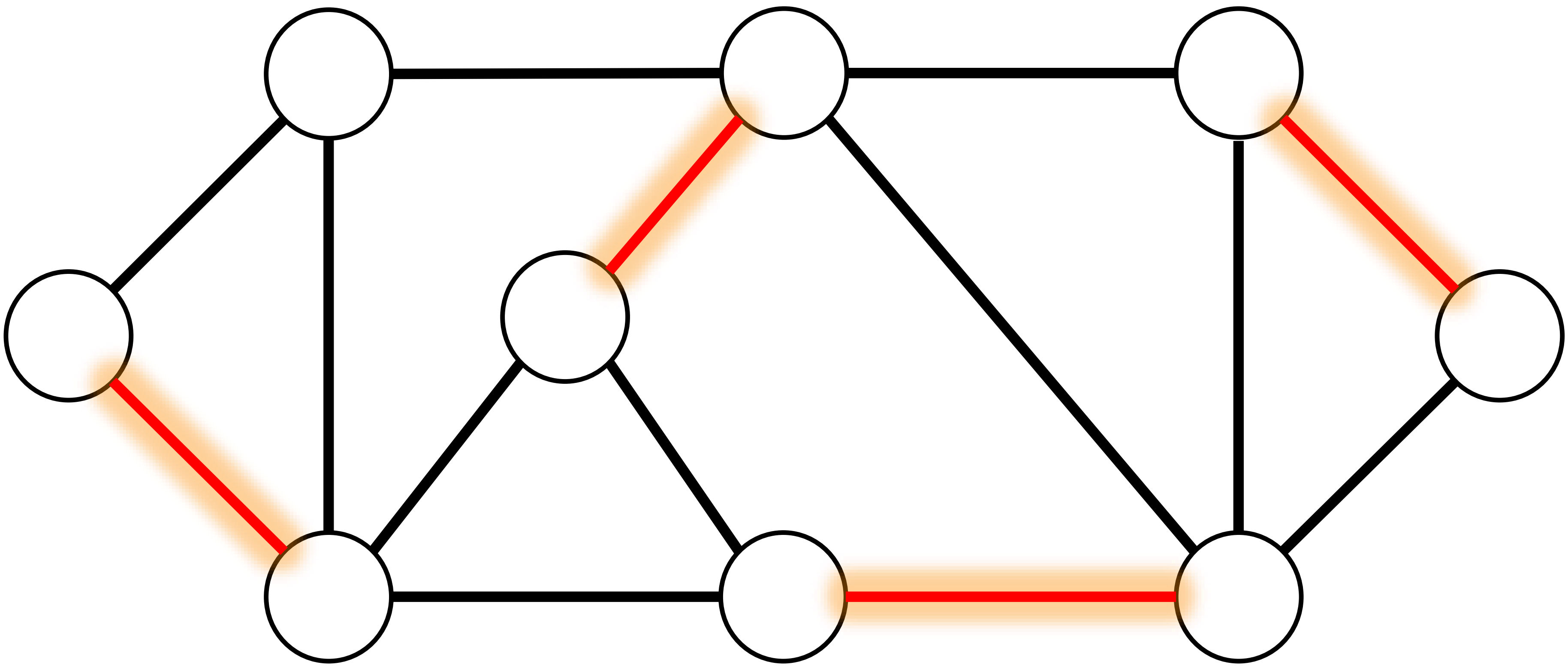
Emparelhamentos

Definição: Um emparelhamento em um grafo G é ***máximo*** se possui o maior número possível de arestas. (Obs: pode haver um número exponencial de emparelhamentos máximos!)



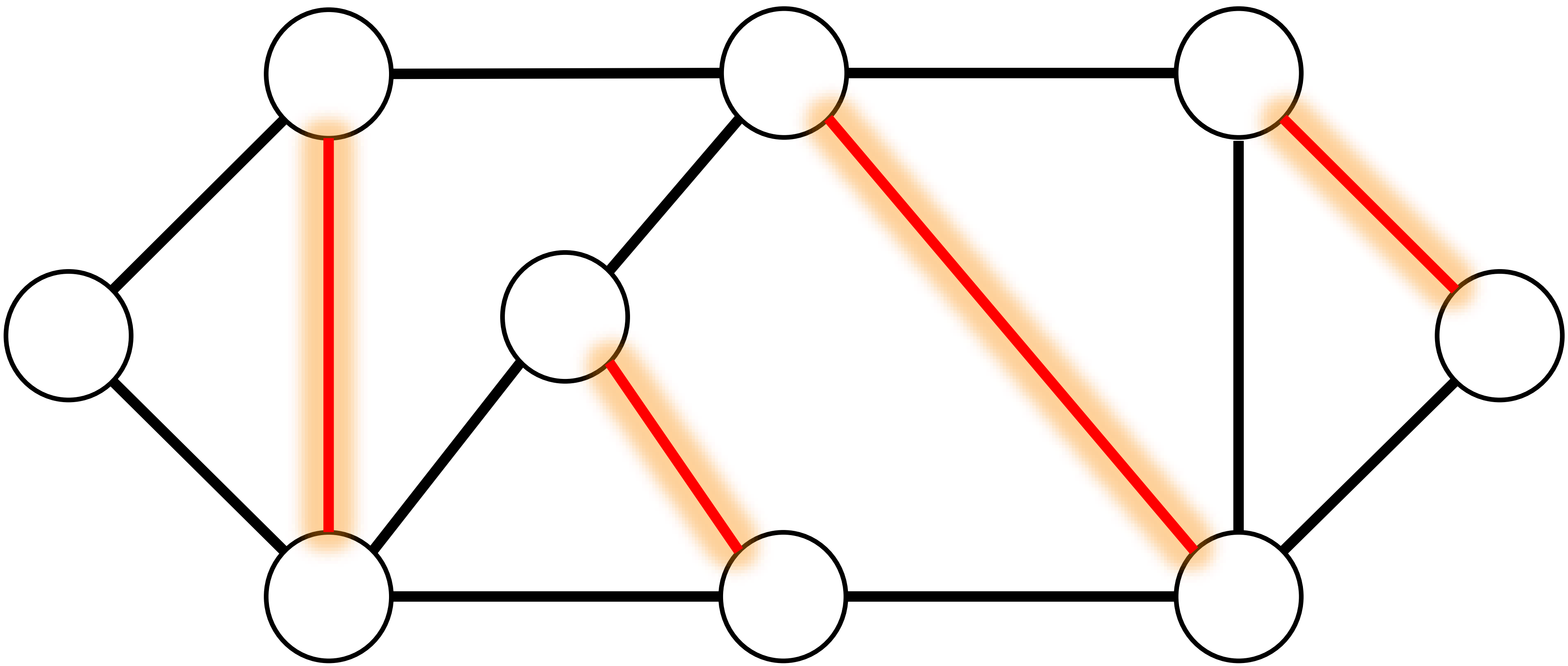
Emparelhamentos

Definição: Um emparelhamento em um grafo G é ***máximo*** se possui o maior número possível de arestas. (Obs: pode haver um número exponencial de emparelhamentos máximos!)



Emparelhamentos

Definição: Um emparelhamento em um grafo *G* é ***máximo*** se possui o maior número possível de arestas. (Obs: pode haver um número exponencial de emparelhamentos máximos!)

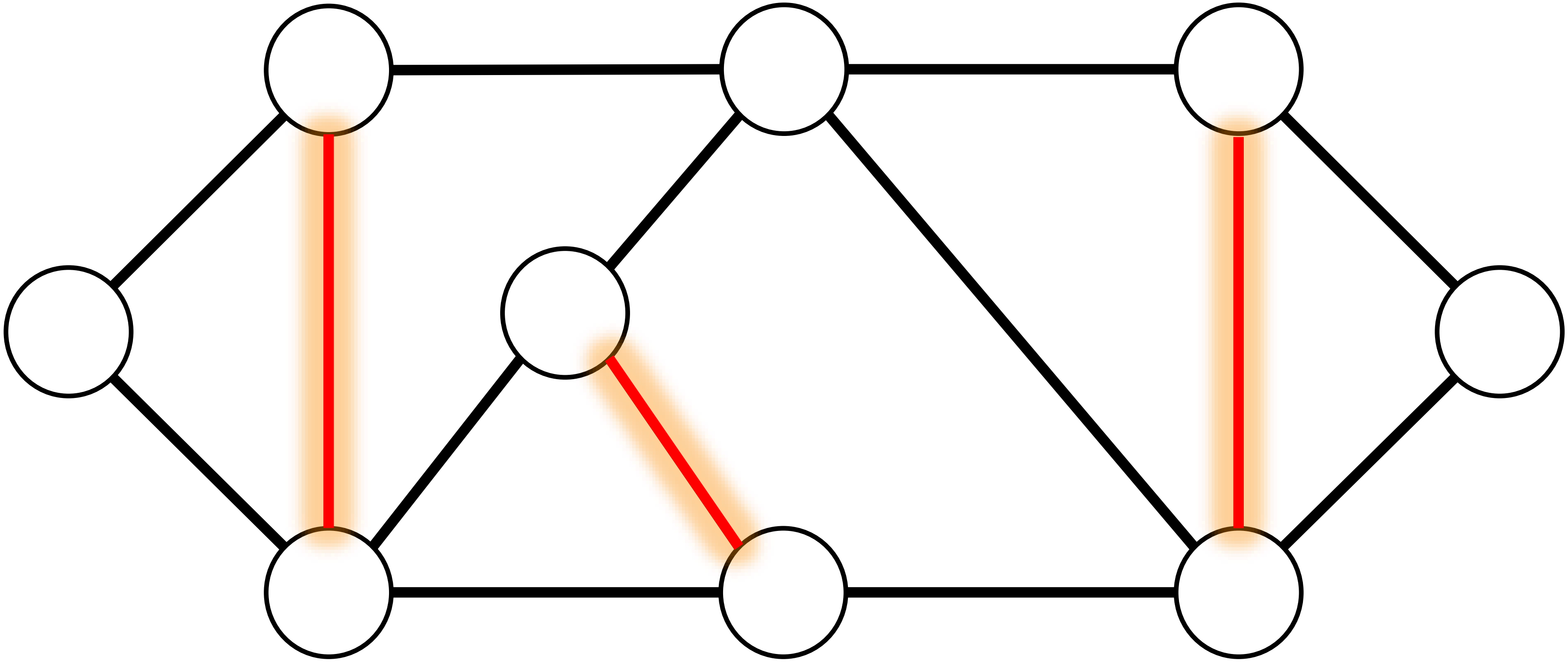


Emparelhamentos

Definição: Um emparelhamento em um grafo G é ***maximal*** se não está contido em nenhum emparelhamento maior. (Obs: todo emparelhamento *máximo* é *maximal*.)

Emparelhamentos

Definição: Um emparelhamento em um grafo G é ***maximal*** se não está contido em nenhum emparelhamento maior. (Obs: todo emparelhamento *máximo* é *maximal*.)



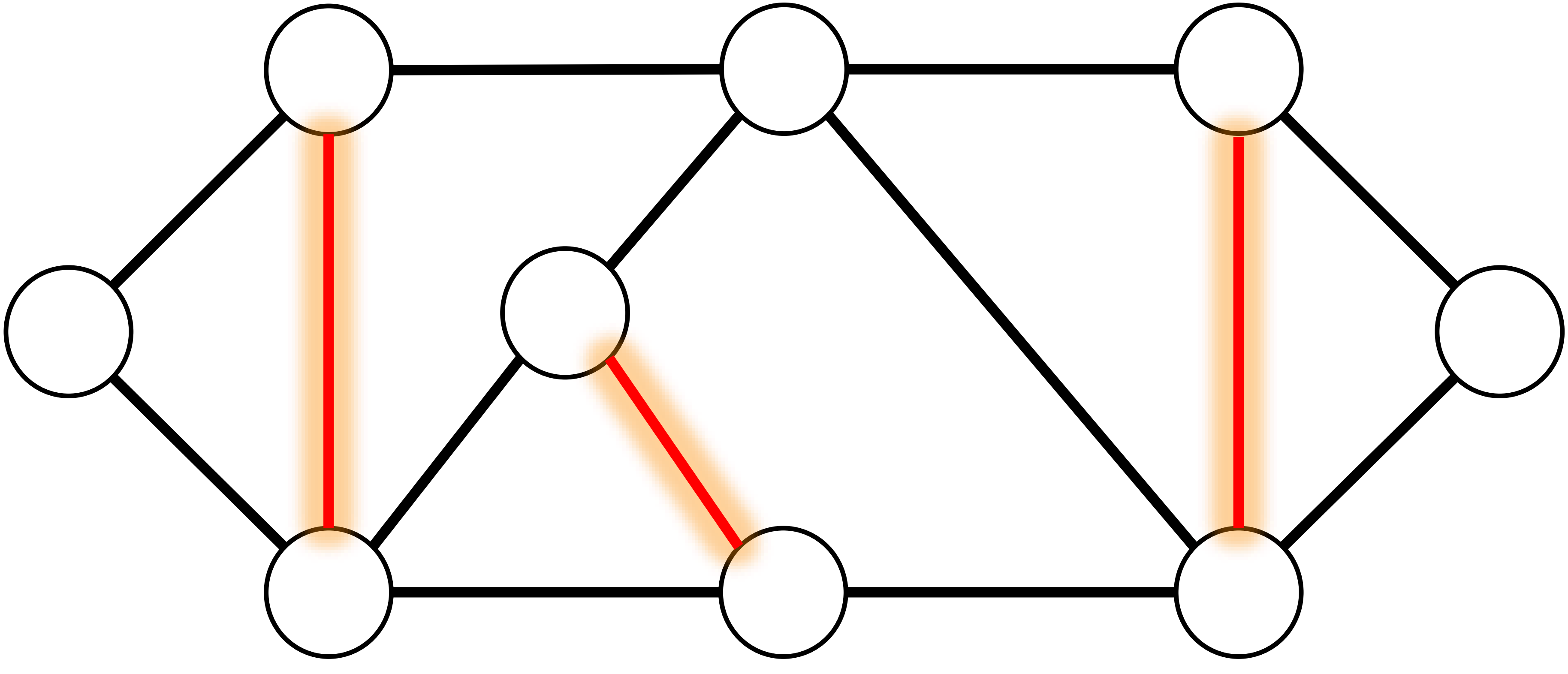
exemplo de emparelhamento maximal mas não máximo

Emparelhamentos

Definição: Seja M um emparelhamento em um grafo G. Um vértice v é ***M-saturado*** se alguma aresta de M incide sobre v ; caso contrário, v é ***M-insaturado.***

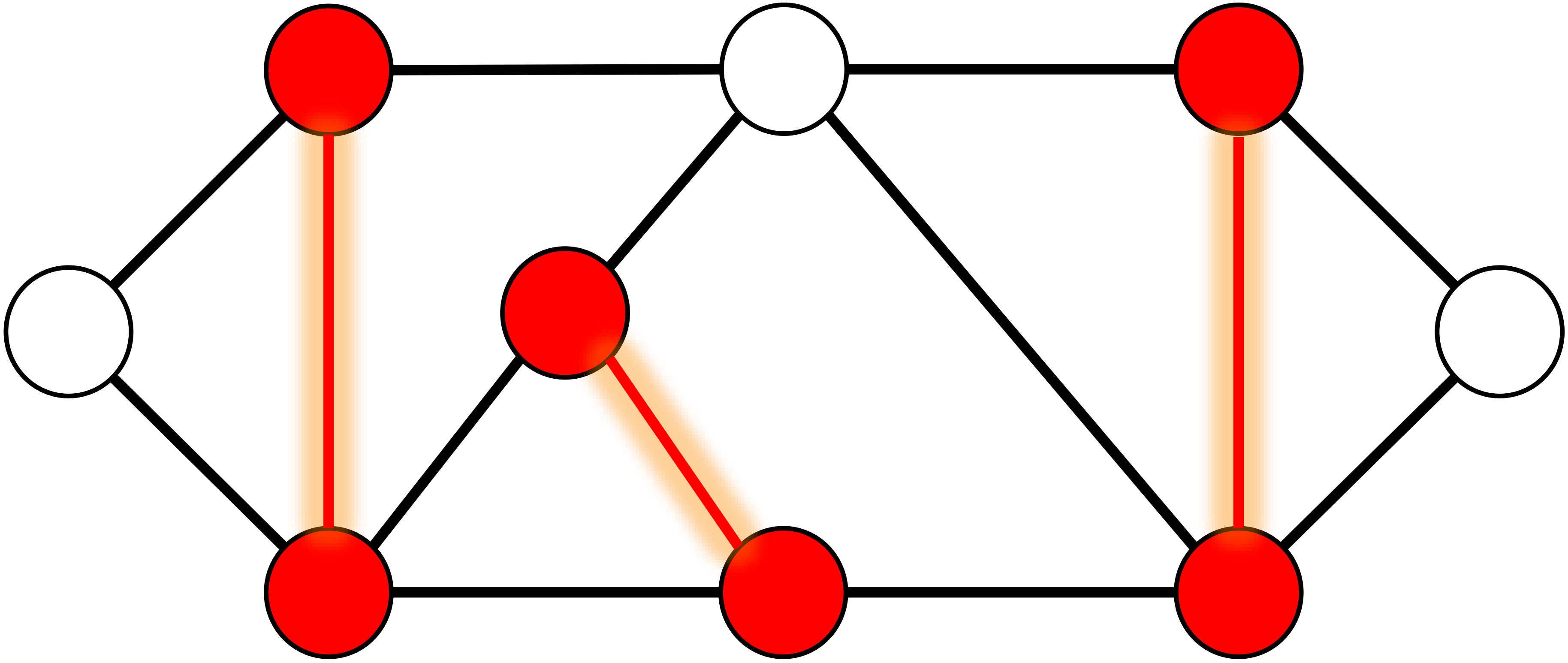
Emparelhamentos

Definição: Seja *M* um emparelhamento em um grafo *G*. Um vértice *v* é ***M-saturado*** se alguma aresta de *M* incide sobre *v* ; caso contrário, *v* é ***M-insaturado.***



Emparelhamentos

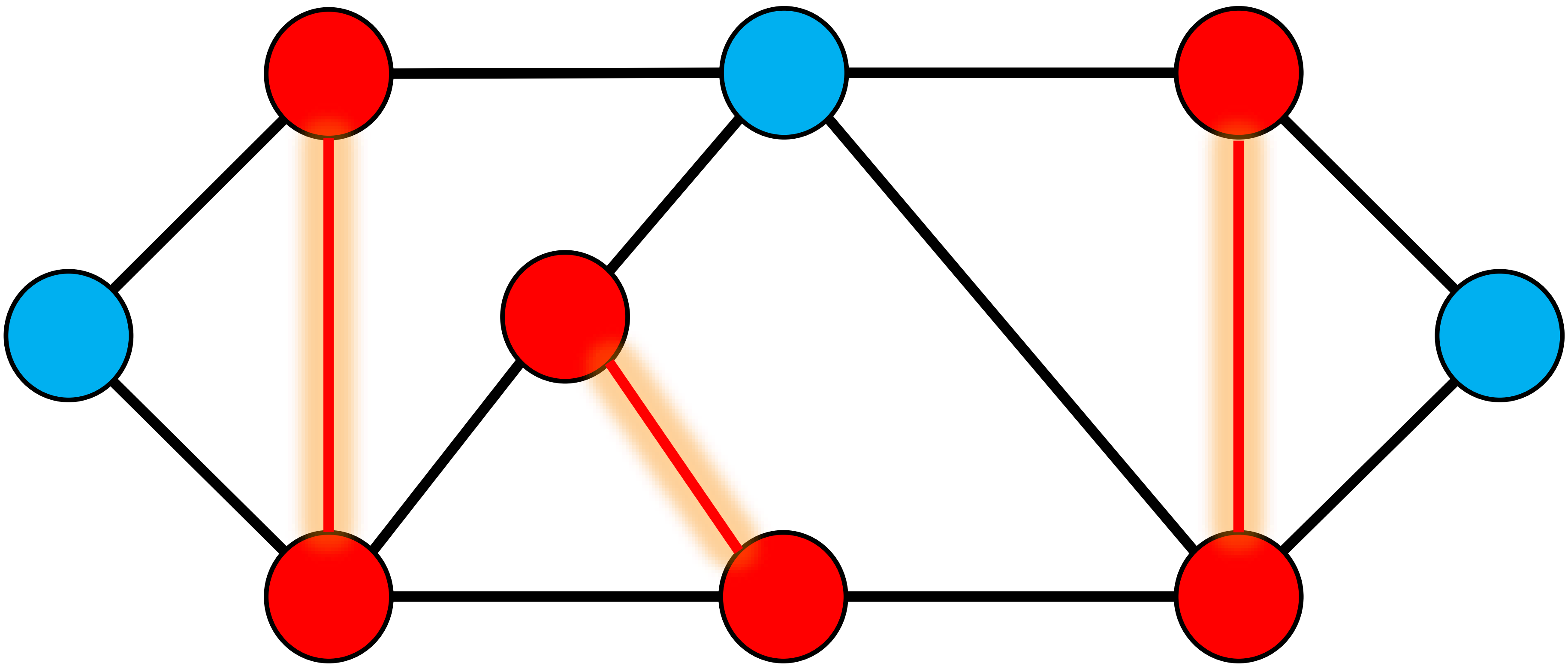
Definição: Seja *M* um emparelhamento em um grafo *G*. Um vértice *v* é ***M-saturado*** se alguma aresta de *M* incide sobre *v* ; caso contrário, *v* é ***M-insaturado.***



vértices saturados em vermelho

Emparelhamentos

Definição: Seja *M* um emparelhamento em um grafo *G*. Um vértice *v* é ***M-saturado*** se alguma aresta de *M* incide sobre *v*; caso contrário, *v* é ***M-insaturado.***



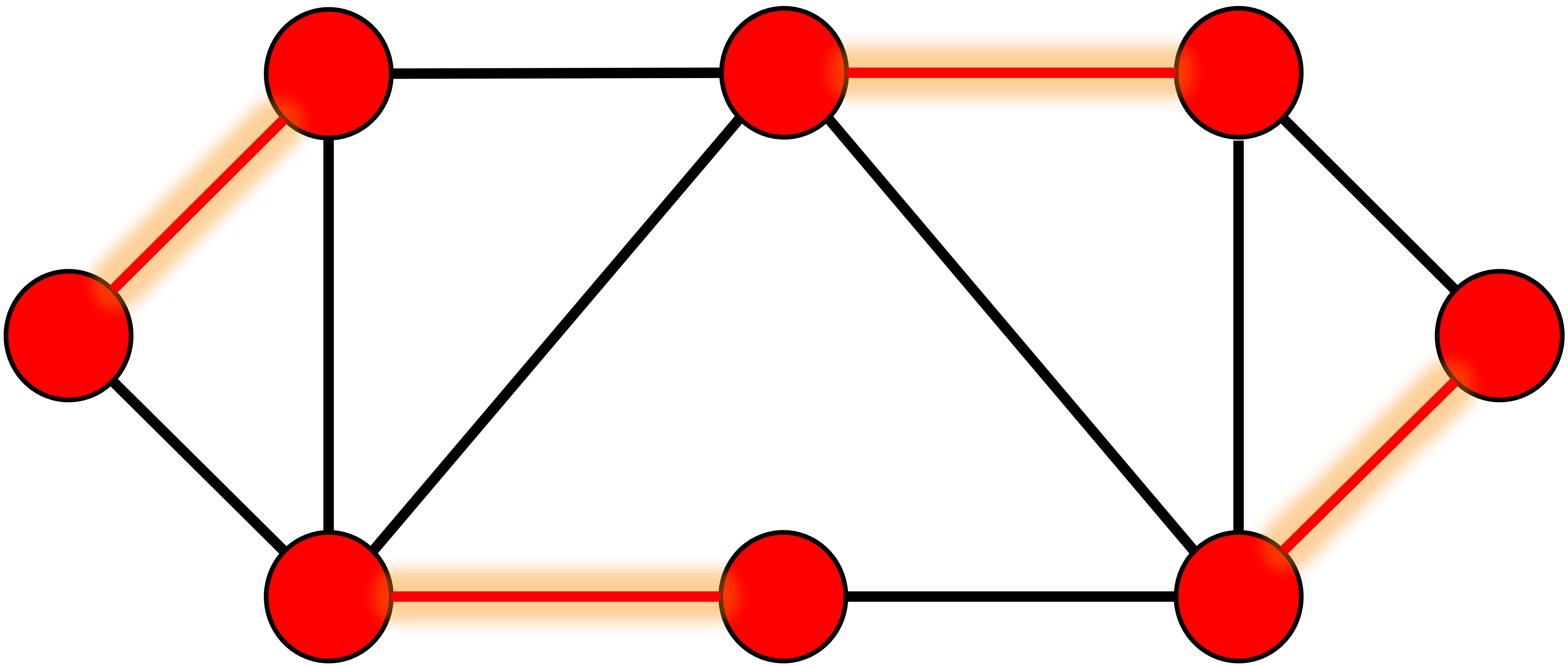
vértices saturados em vermelho ***vértices insaturados em azul***

Emparelhamentos

Definição: Um emparelhamento *M* em um grafo *G* é chamado ***perfeito*** se todo vértice de *G* é ***M-saturado.*** (Obs: todo emparelhamento perfeito é máximo.)

Emparelhamentos

Definição: Um emparelhamento M em um grafo G é chamado ***perfeito*** se todo vértice de G é ***M-saturado.*** (Obs: todo emparelhamento perfeito é máximo.)



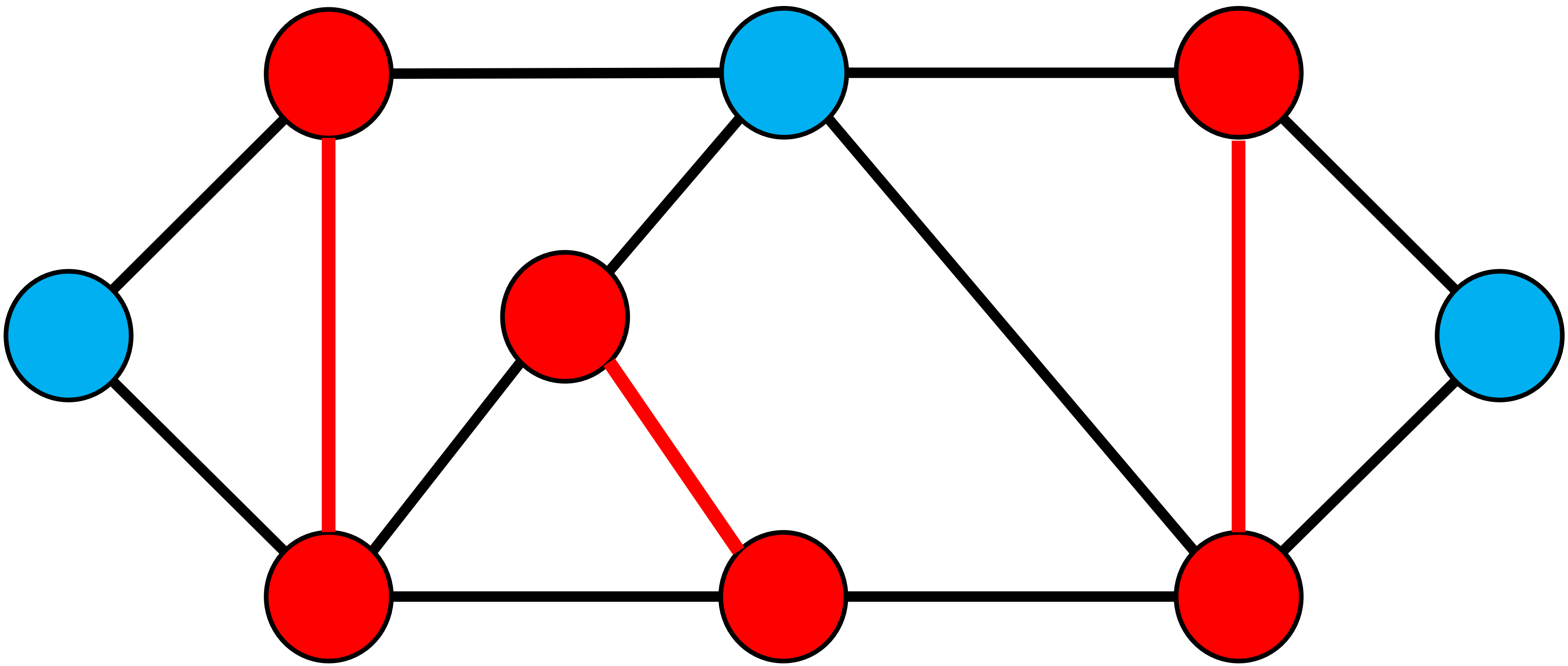
vértices saturados em vermelho ***não há vértices insaturados!***

Emparelhamentos

Def.: Seja M um emparelhamento. Um ***caminho M-aumentante*** é aquele que inicia e termina em vértices M-insaturados, e alterna arestas de M com arestas de $E(G)\backslash M$.

Emparelhamentos

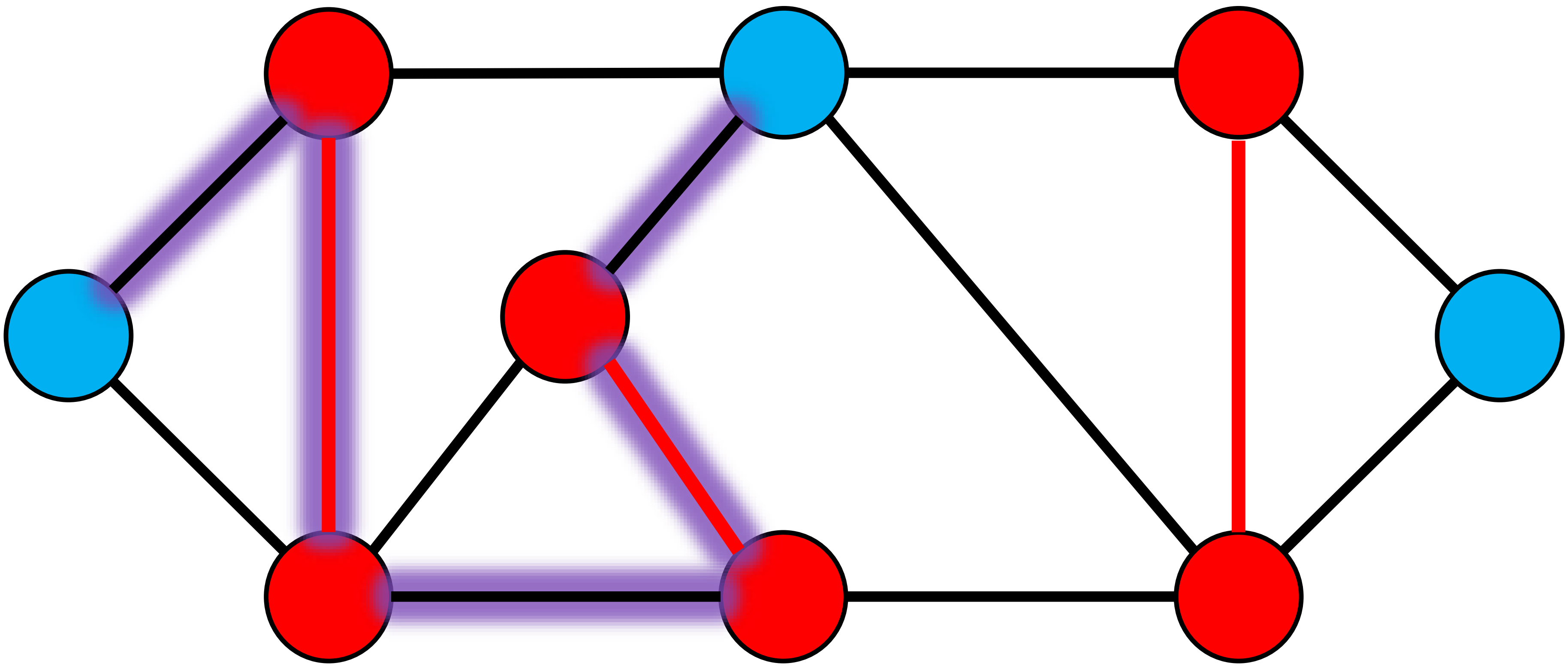
Def.: Seja M um emparelhamento. Um ***caminho M-aumentante*** é aquele que inicia e termina em vértices M-insaturados, e alterna arestas de M com arestas de $E(G)\setminus M$.



vértices saturados em vermelho ***vértices insaturados em azul***

Emparelhamentos

Def.: Seja M um emparelhamento. Um ***caminho M-aumentante*** é aquele que inicia e termina em vértices M-insaturados, e alterna arestas de M com arestas de $E(G)\backslash M$.



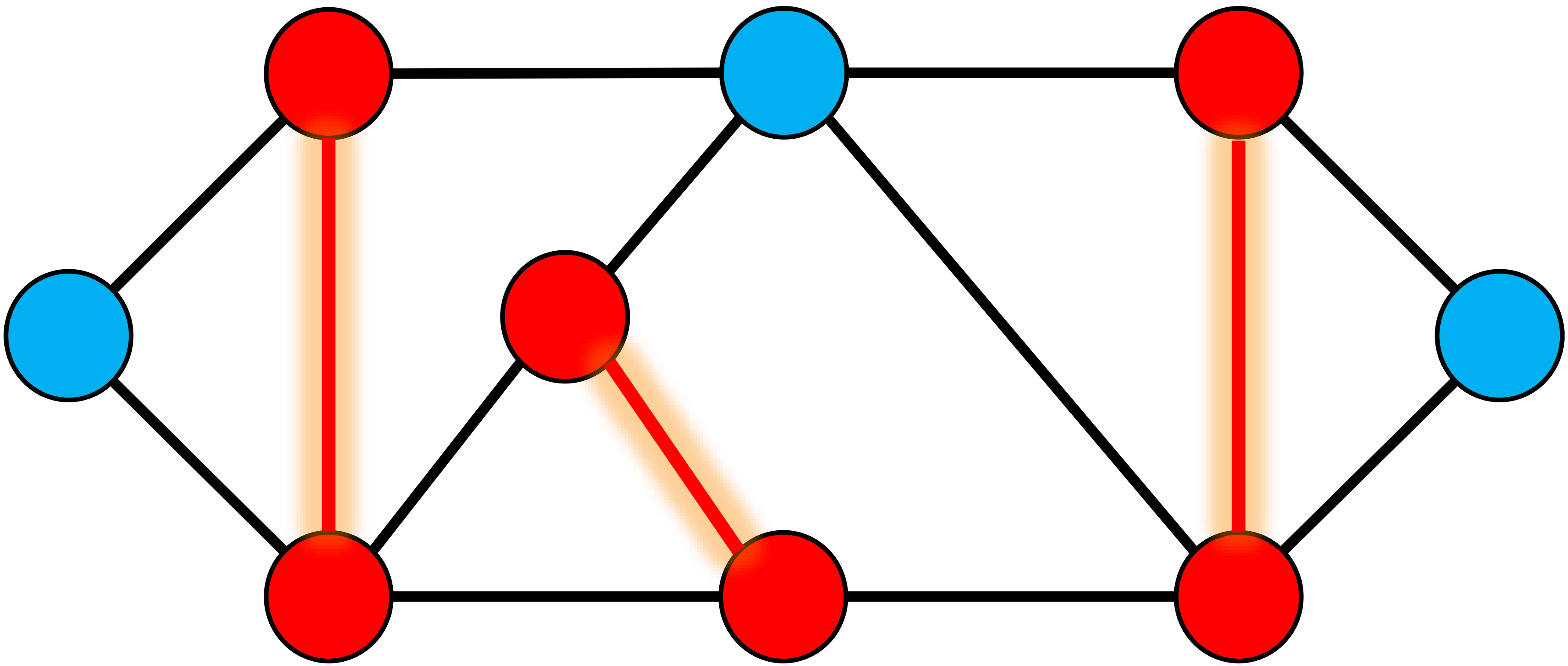
caminho M-aumentante

Emparelhamentos

Importante! Um caminho M-aumentante ***P*** fornece um modo de aumentar o tamanho do emparelhamento M em uma unidade.

Emparelhamentos

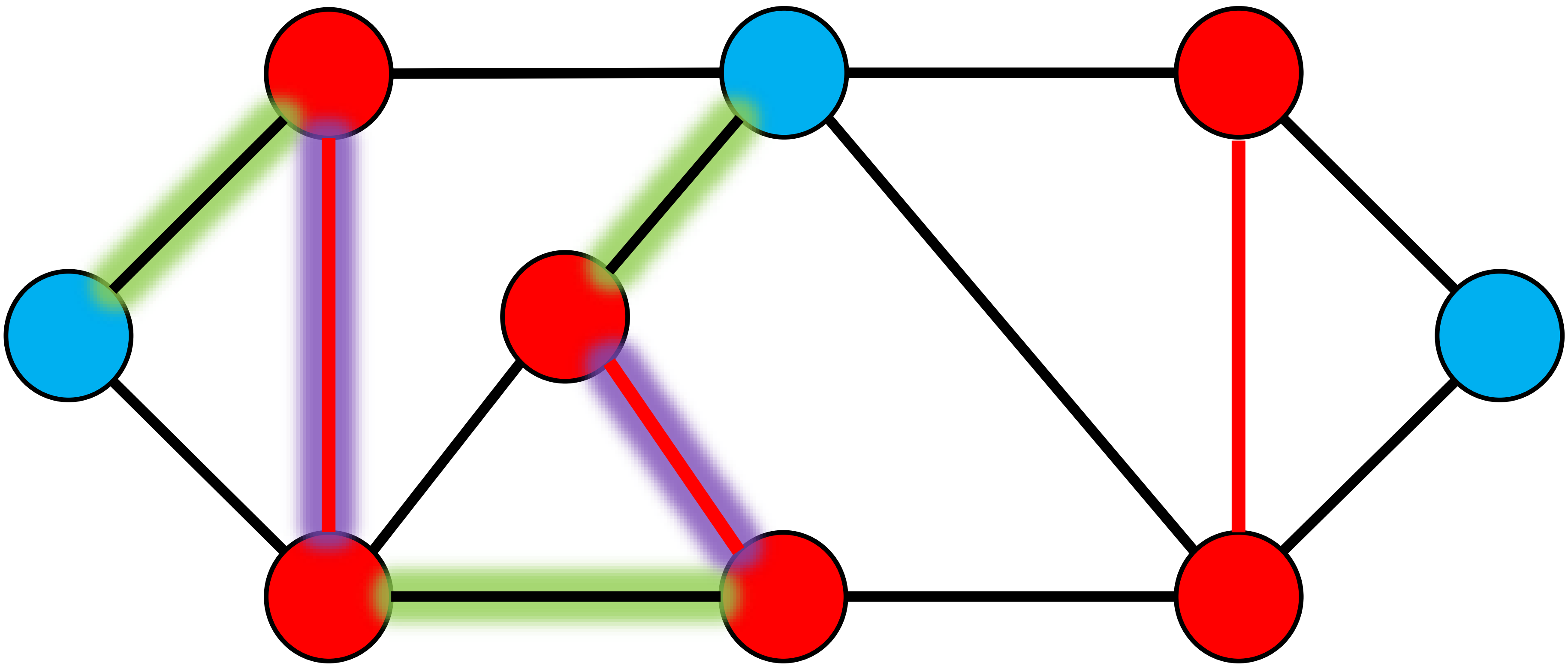
Importante! Um caminho *M*-aumentante ***P*** fornece um modo de aumentar o tamanho do emparelhamento *M* em uma unidade.



emparelhamento M

Emparelhamentos

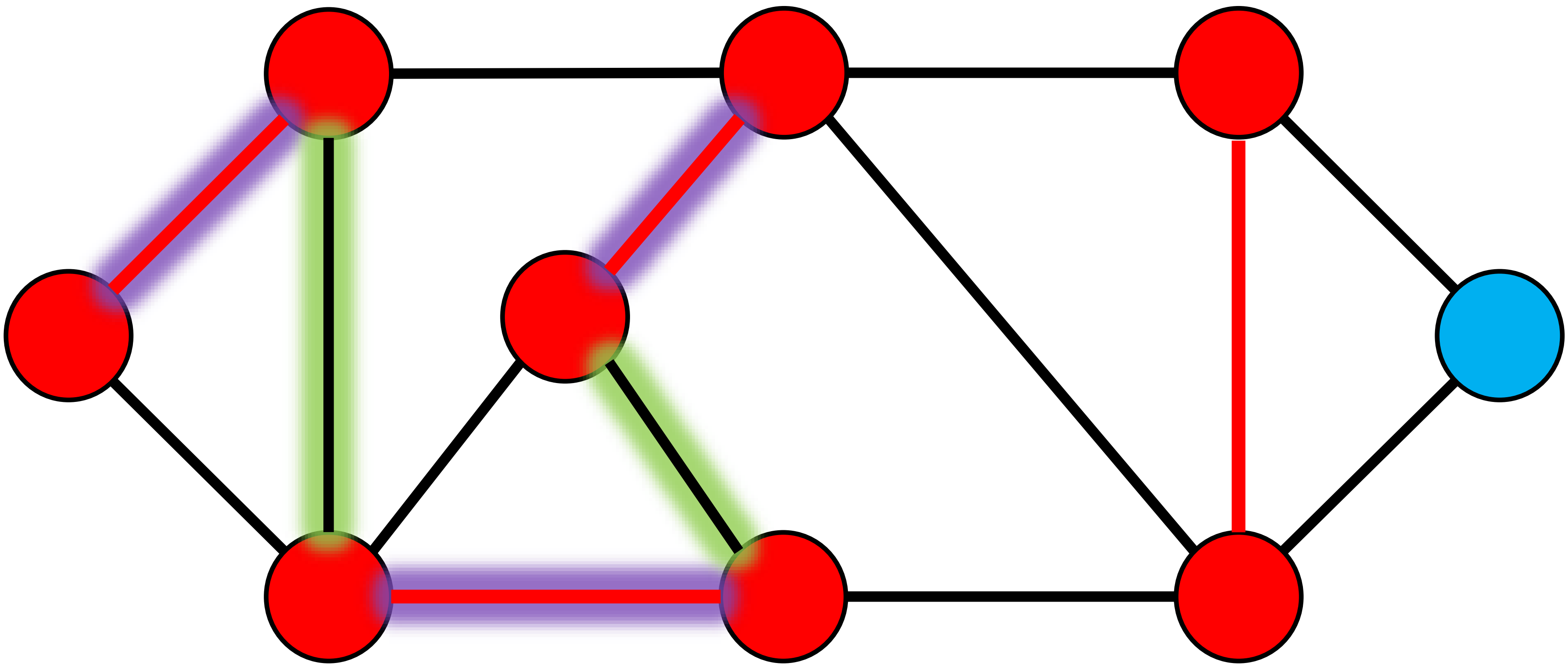
Importante! Um caminho *M*-aumentante ***P*** fornece um modo de aumentar o tamanho do emparelhamento *M* em uma unidade.



caminho M-aumentante P

Emparelhamentos

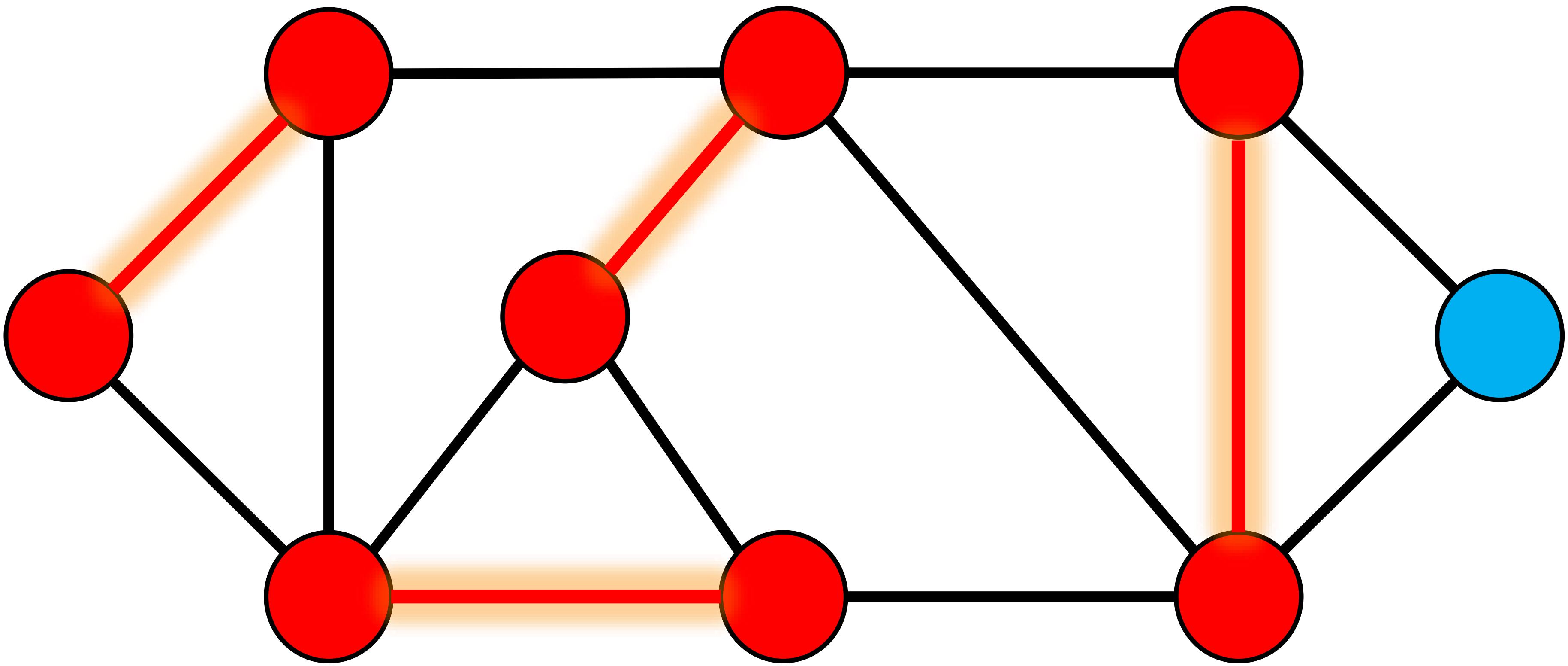
Importante! Um caminho *M*-aumentante ***P*** fornece um modo de aumentar o tamanho do emparelhamento *M* em uma unidade.



em P, trocar arestas de M por arestas de E(G)\M, e vice-versa

Emparelhamentos

Importante! Um caminho *M*-aumentante ***P*** fornece um modo de aumentar o tamanho do emparelhamento *M* em uma unidade.



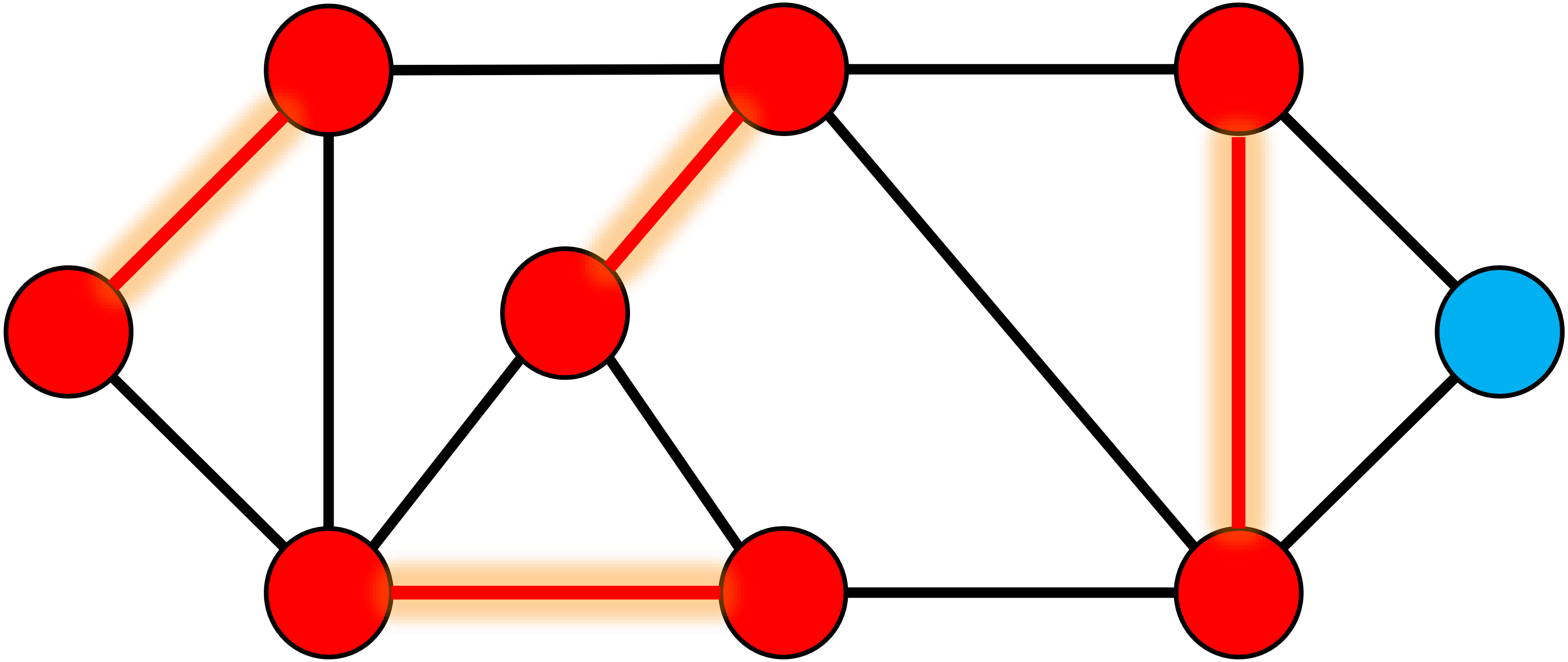
novo emparelhamento M'

Emparelhamentos

A operação de aumento do emparelhamento é dada por

$$M' \leftarrow M \,\Delta\, E(P)$$

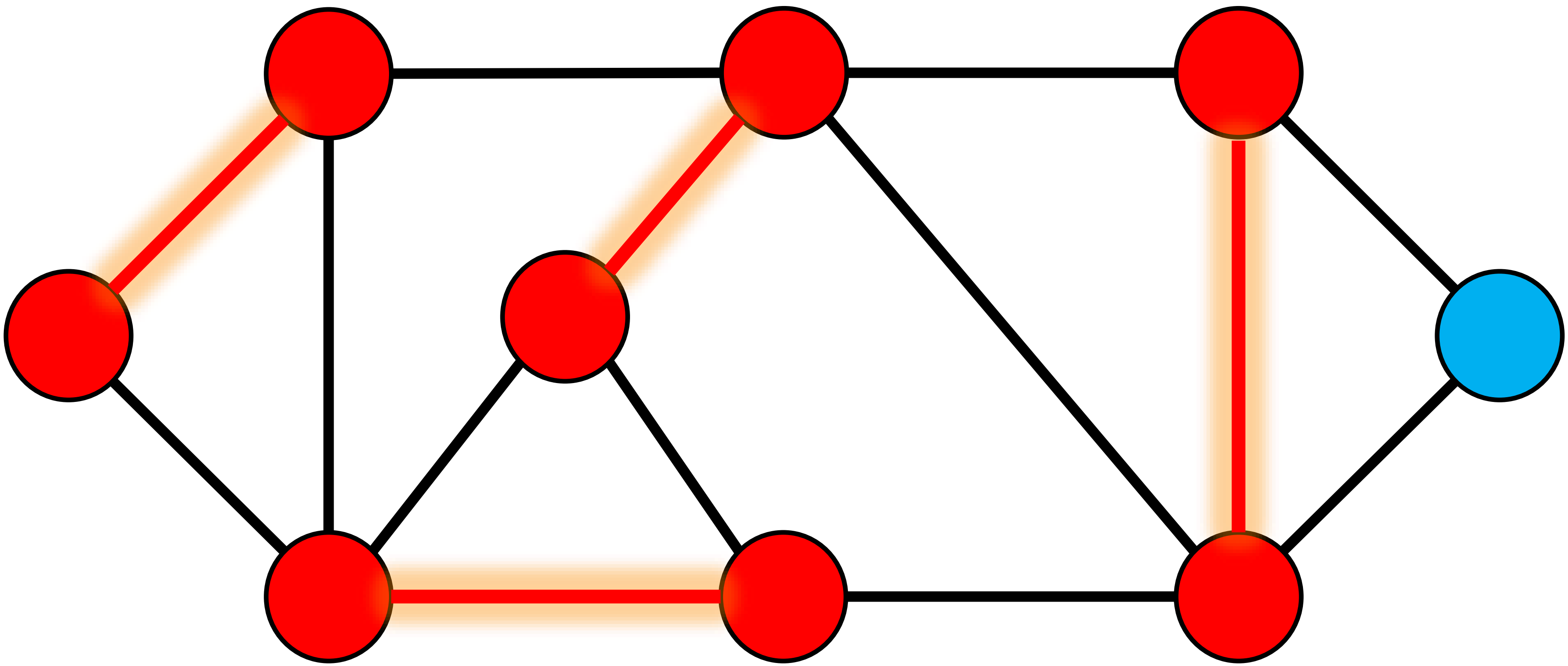
onde Δ é a diferença simétrica entre conjuntos.



novo emparelhamento M'

Emparelhamentos

Teorema (Berge): Um emparelhamento *M* é máximo se e somente não existe caminho *M*-aumentante no grafo.



Emparelhamentos

O Teorema de Berge sugere o seguinte algoritmo para encontrar um emparelhamento máximo:

Algoritmo para encontrar um emparelhamento máximo

Dado o grafo G, constrói um emparelhamento máximo M em G

M ← *um emparelhamento inicial qualquer* ***-- inicialização de M***

enquanto existe um caminho M-aumentante P em G faça

$\qquad M \leftarrow M \,\Delta\, E(P)$

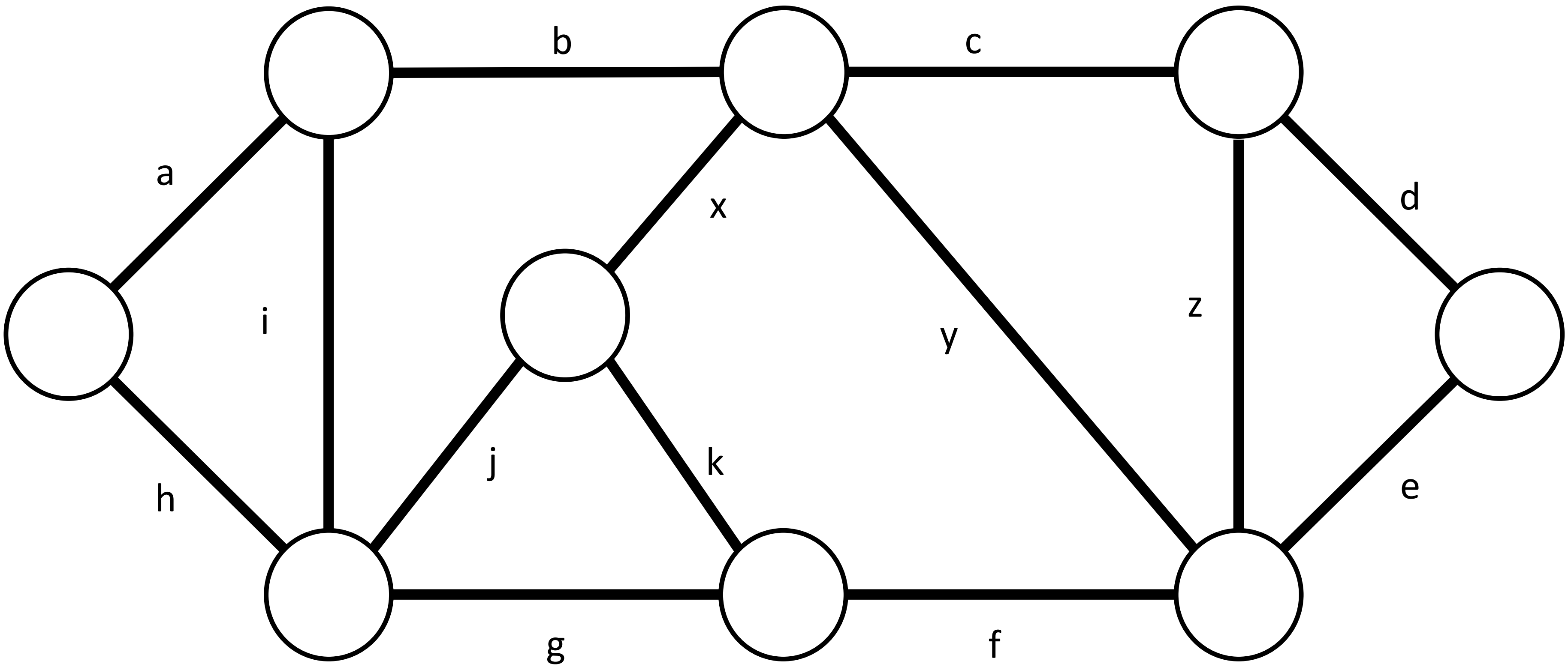
fim-enquanto

retorne M

Emparelhamentos

Exemplo: Simulação da execução do algoritmo.

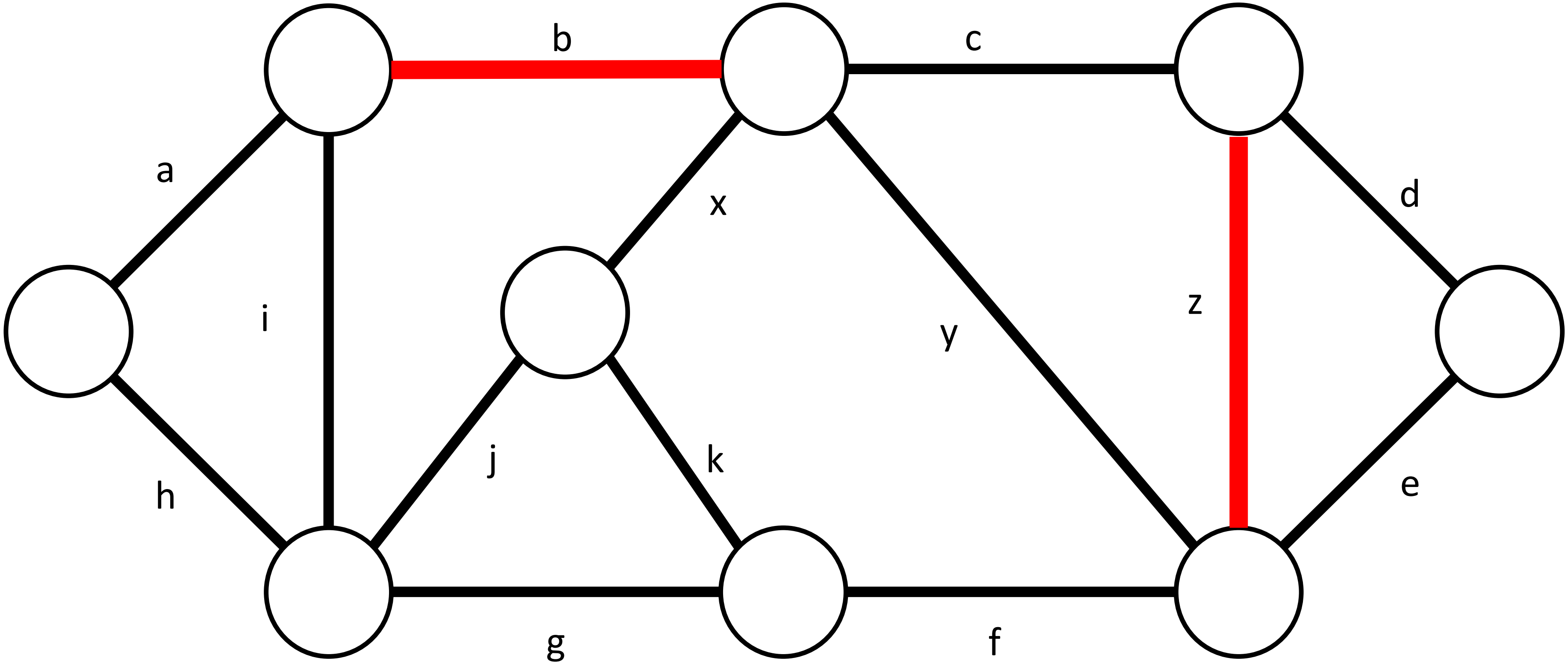
grafo de entrada



Emparelhamentos

Exemplo: Simulação da execução do algoritmo.

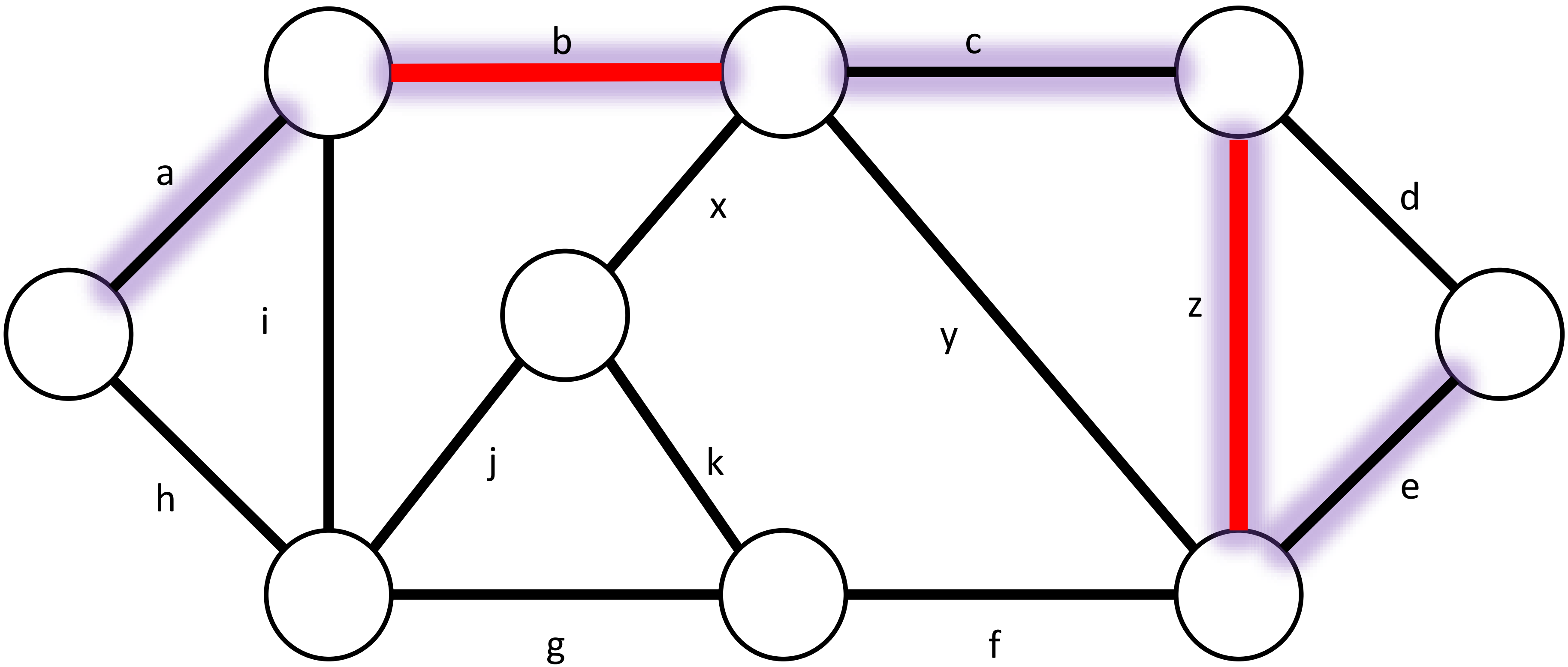
emparelhamento inicial M



Emparelhamentos

Exemplo: Simulação da execução do algoritmo.

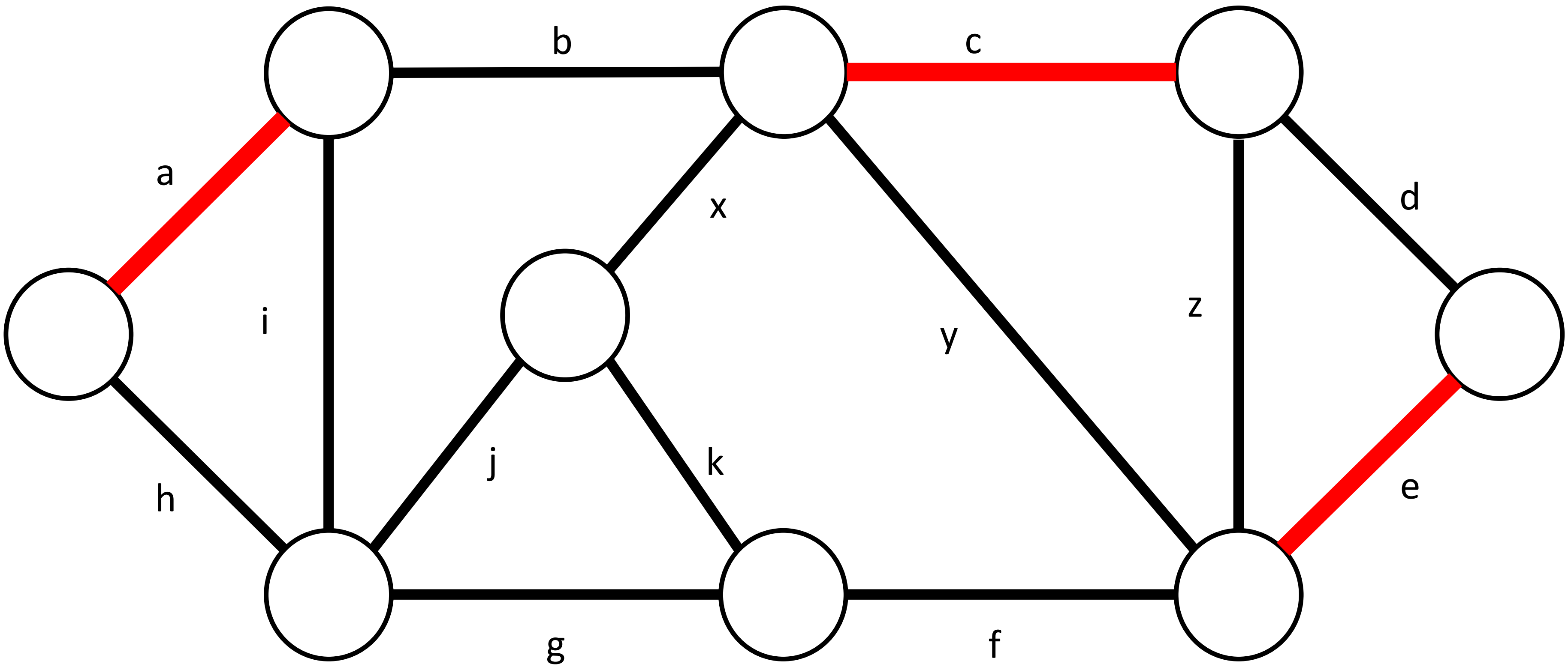
caminho M-aumentante P



Emparelhamentos

Exemplo: Simulação da execução do algoritmo.

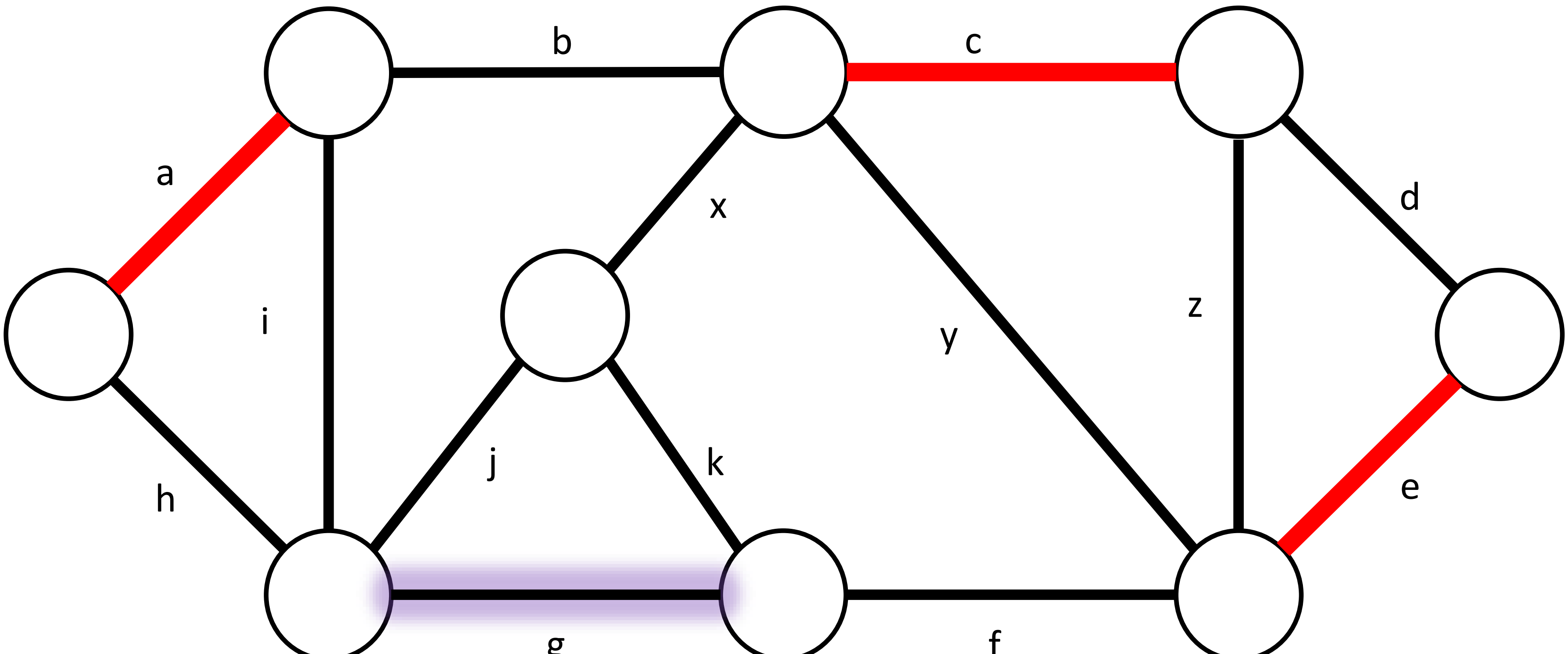
novo emparelhamento M



Emparelhamentos

Exemplo: Simulação da execução do algoritmo.

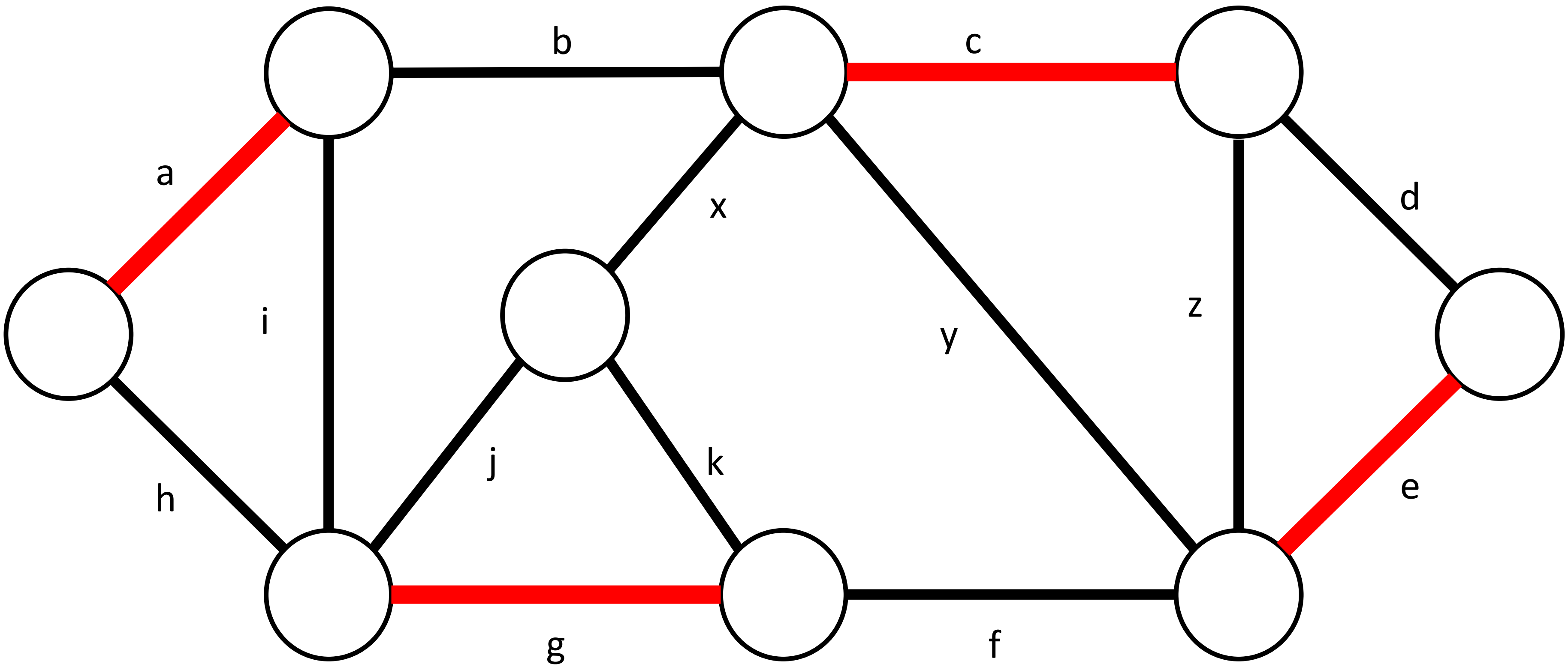
caminho M-aumentante P



Emparelhamentos

Exemplo: Simulação da execução do algoritmo.

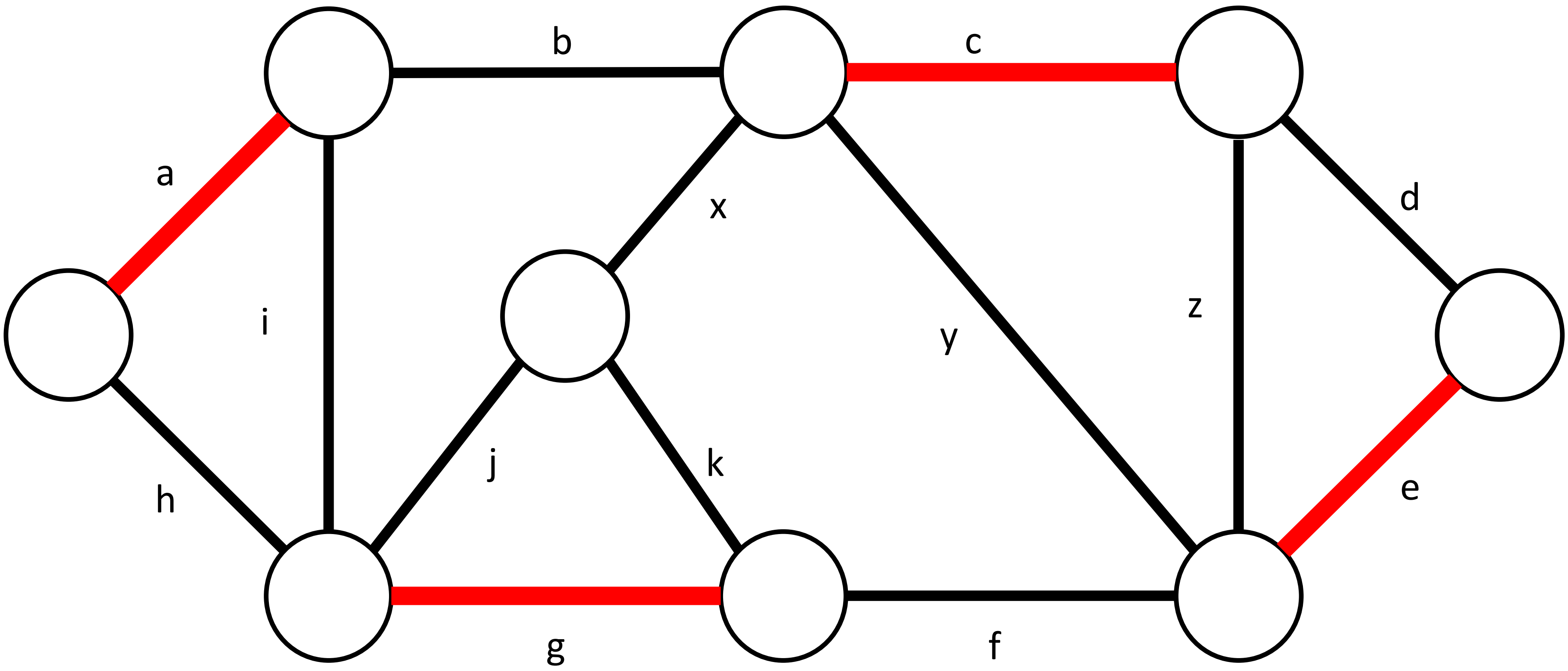
novo emparelhamento M



Emparelhamentos

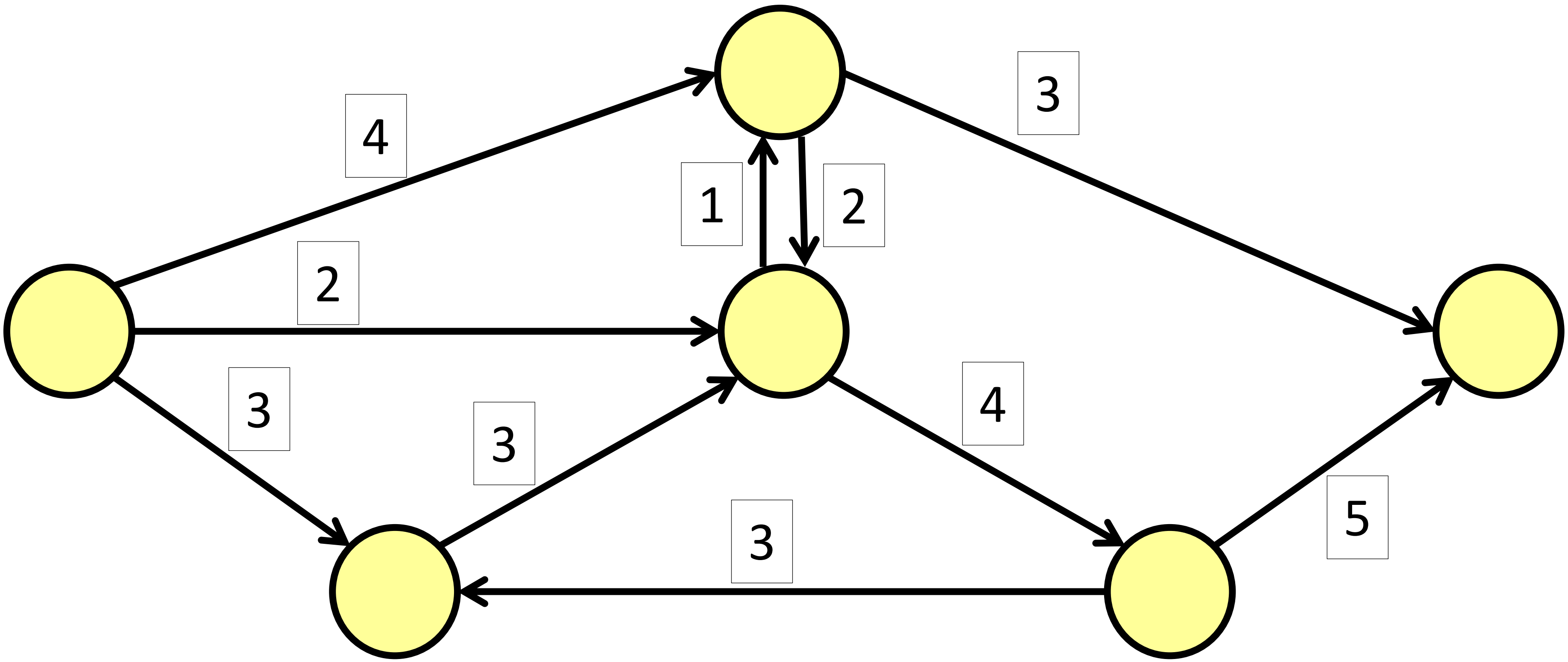
Exemplo: Simulação da execução do algoritmo.

não há caminho M-aumentante – M é máximo!



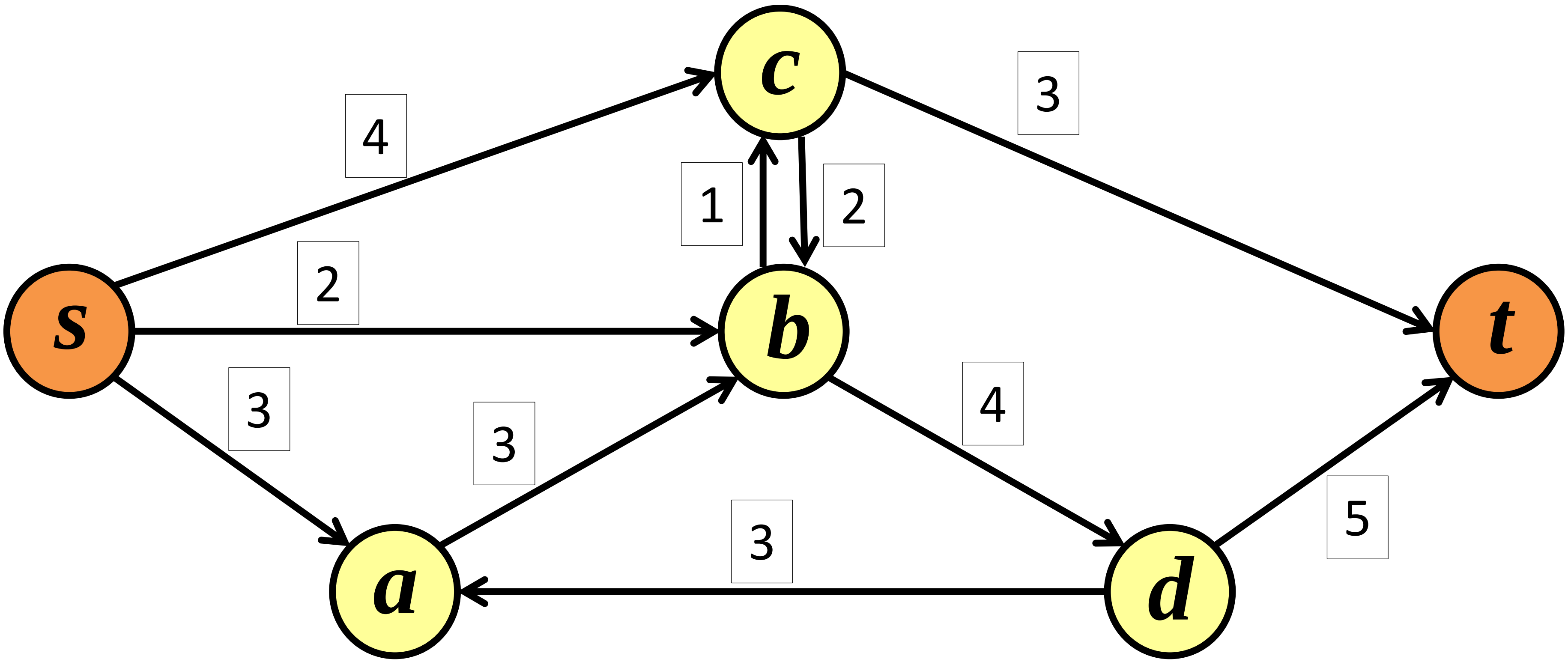
Fluxos em redes

Def.: Uma ***rede*** é um ***multidigrafo*** $D=(V, E)$ onde cada aresta e em E possui uma capacidade real $c(e) > 0$.



Fluxos em redes

Supomos que há na rede dois vértices especiais ***s*** (origem) e ***t*** (destino), tais que: ***s*** é uma fonte que alcança todos os demais, e ***t*** é um sumidouro alcançado por todos os demais.

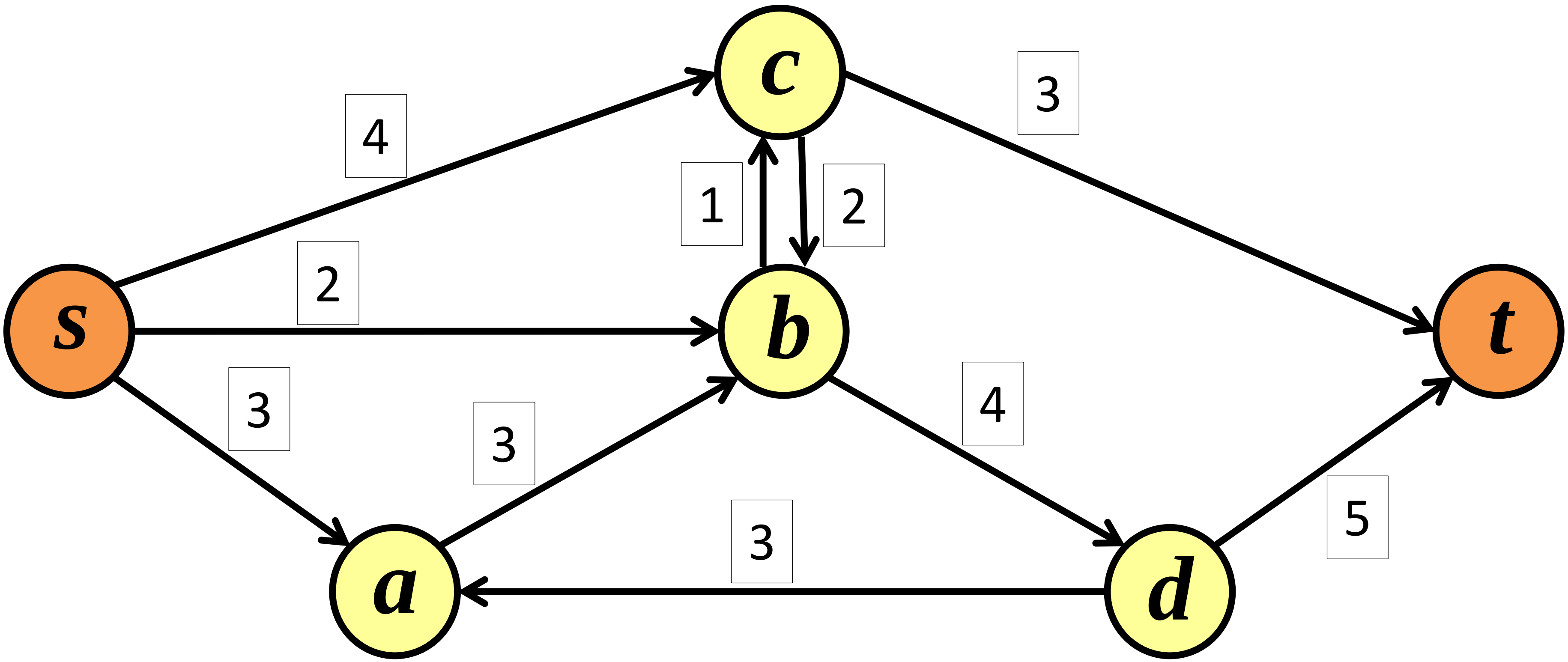


Fluxos em redes

Def.: Um ***fluxo*** é uma função $f : E \rightarrow \mathbb{R}_+$ tal que:

(1) $f(e) \leq c(e)$, para toda aresta e em E;

(2) $\Sigma_x\ f(x, v) = \Sigma_z\ f(v, z)$, para todo v em $V \setminus \{s,t\}$.

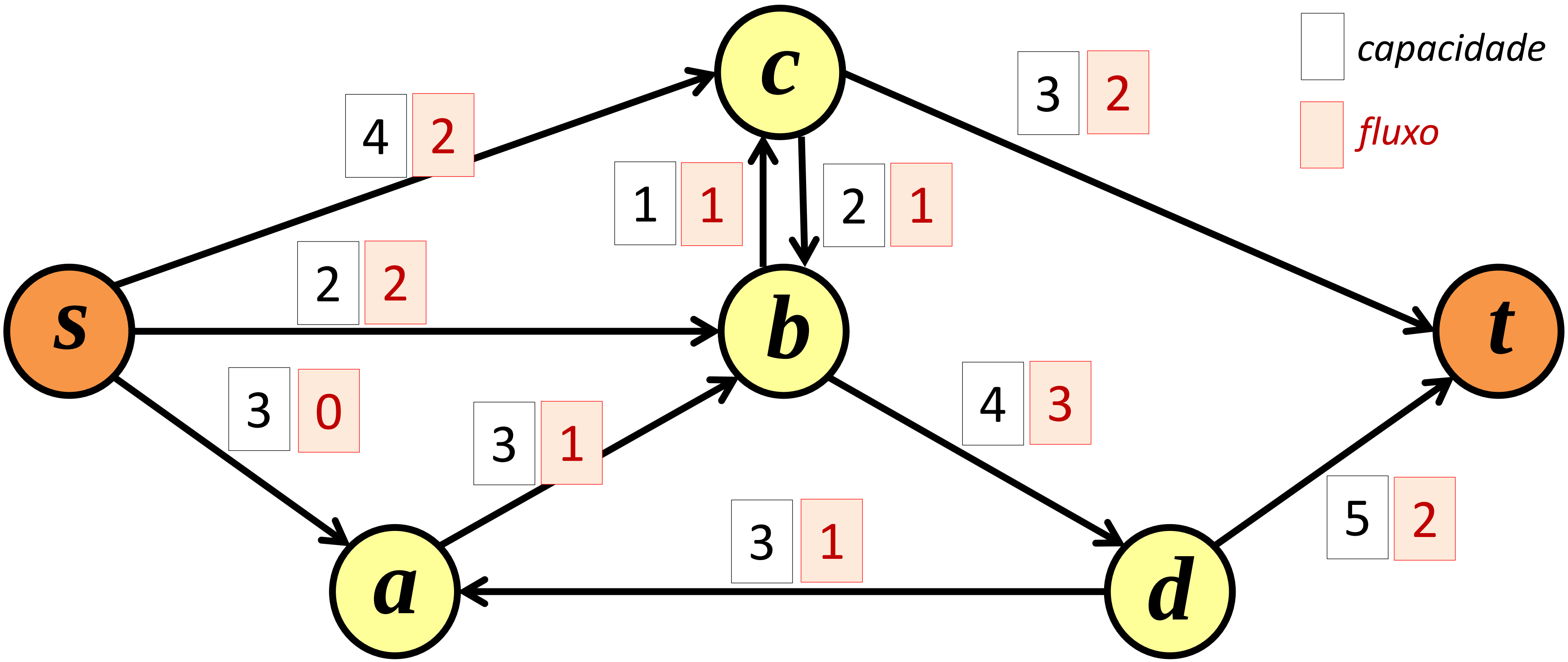


Fluxos em redes

Def.: Um ***fluxo*** é uma função $f : E \rightarrow \mathbb{R}_+$ tal que:

(1) $f(e) \leq c(e)$, para toda aresta e em E;

(2) $\Sigma_x\ f(x, v) = \Sigma_z\ f(v, z)$, para todo v em $V \setminus \{s,t\}$.

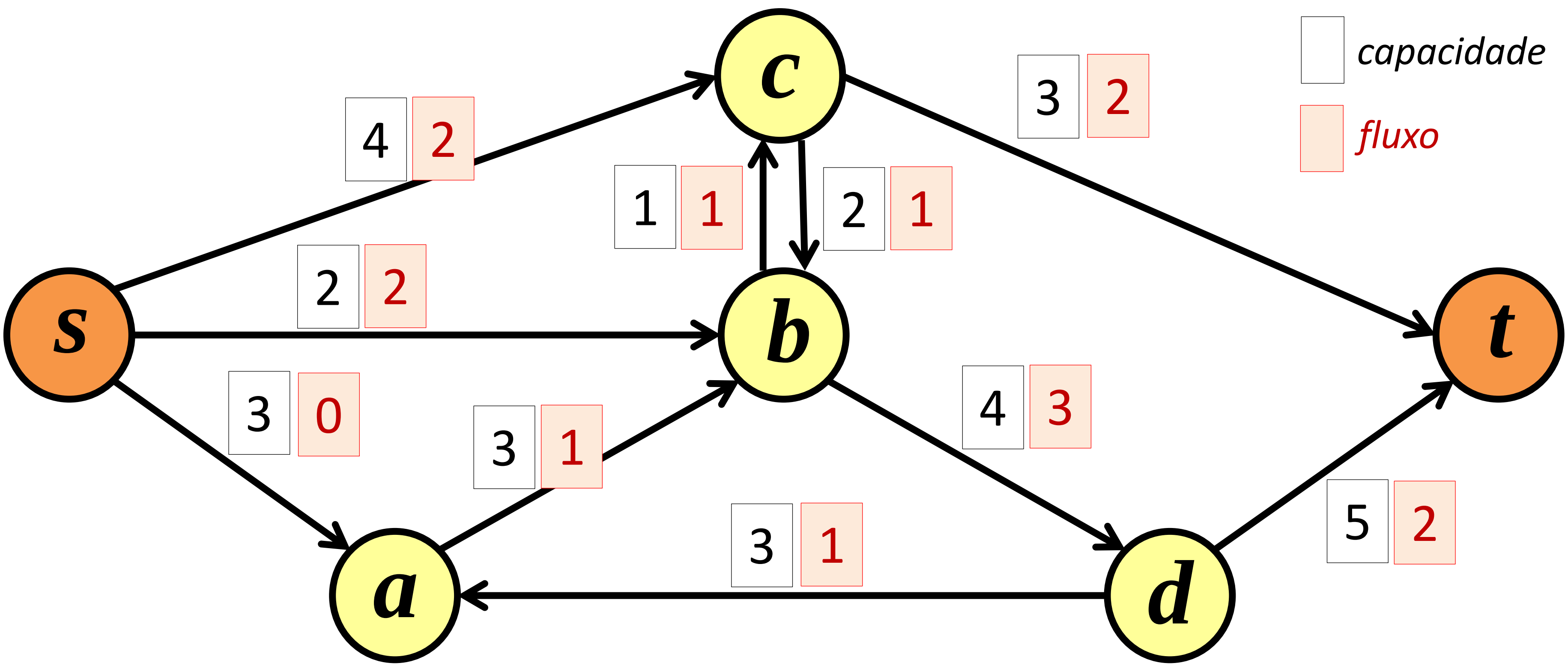


Fluxos em redes

Def.: Um ***fluxo*** é uma função $f : E \rightarrow \mathbb{R}_+$ tal que:

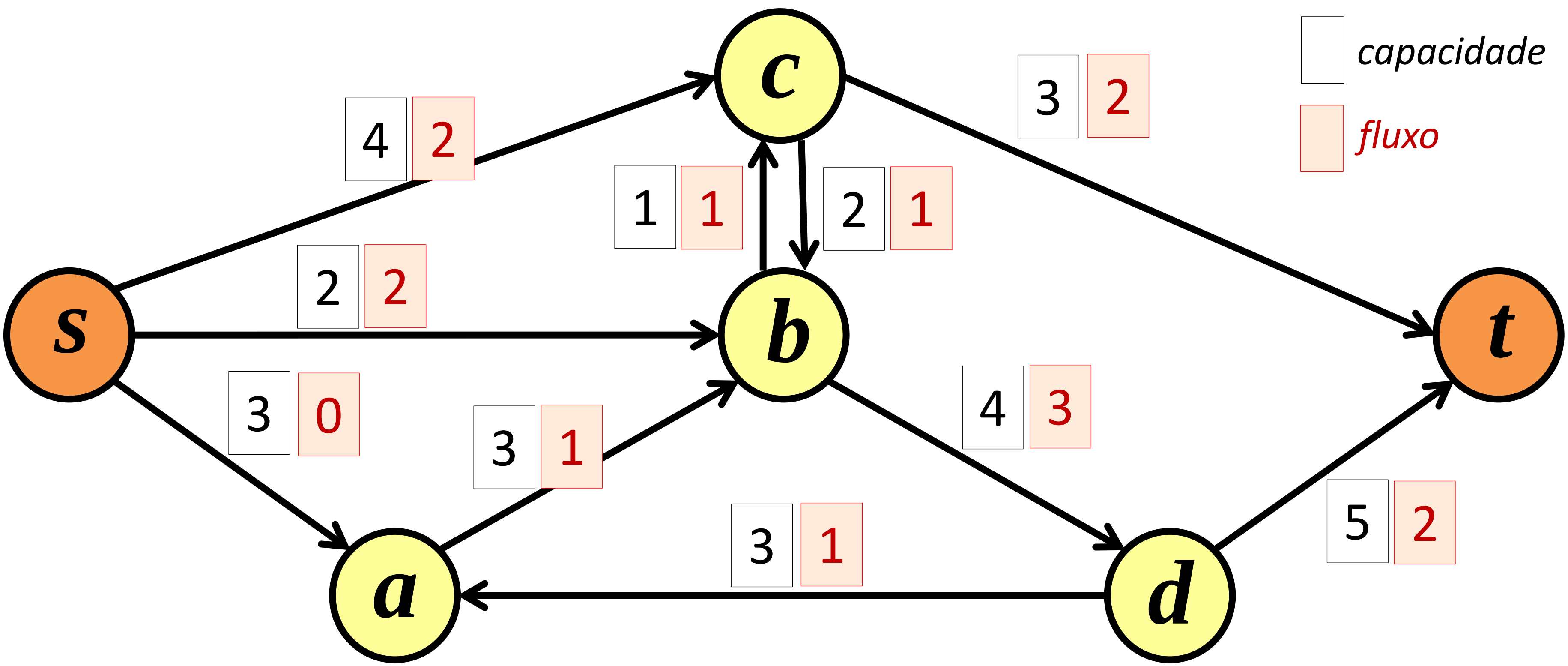
(1) f não ultrapassa as capacidades **(*propr. de* viabilidade)**

(2) f se conserva em todo $v \neq s,t$ **(*propr. de* conservação)**



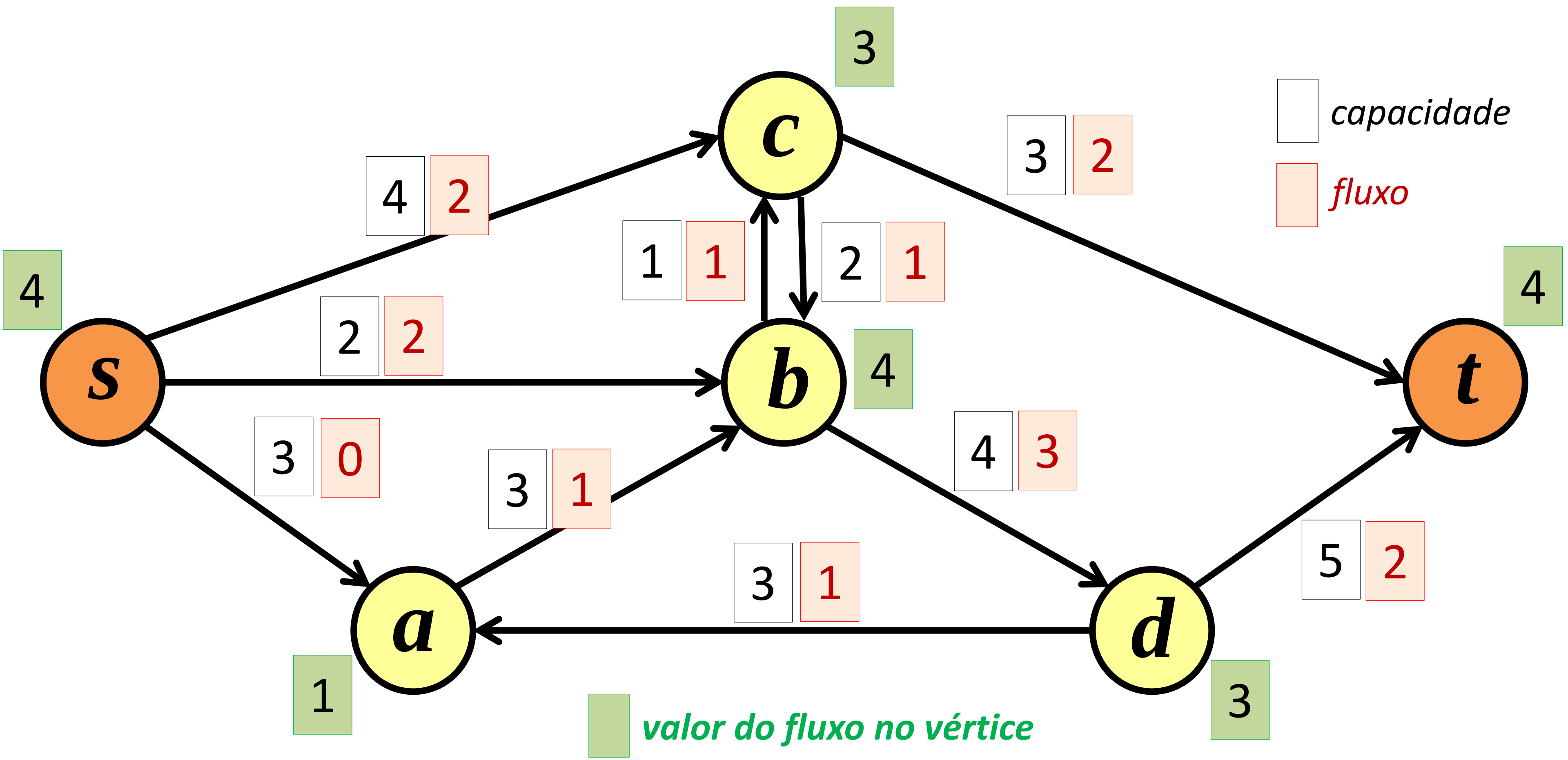
Fluxos em redes

O valor do fluxo em $v \neq s,t$ vale $\Sigma_x\ f(x, v)$ (ou $\Sigma_z\ f(v, z)$).
O valor do fluxo em s é $\Sigma_z\ f(\mathbf{s}, z)$, e em t é $\Sigma_x\ f(x, \mathbf{t})$.



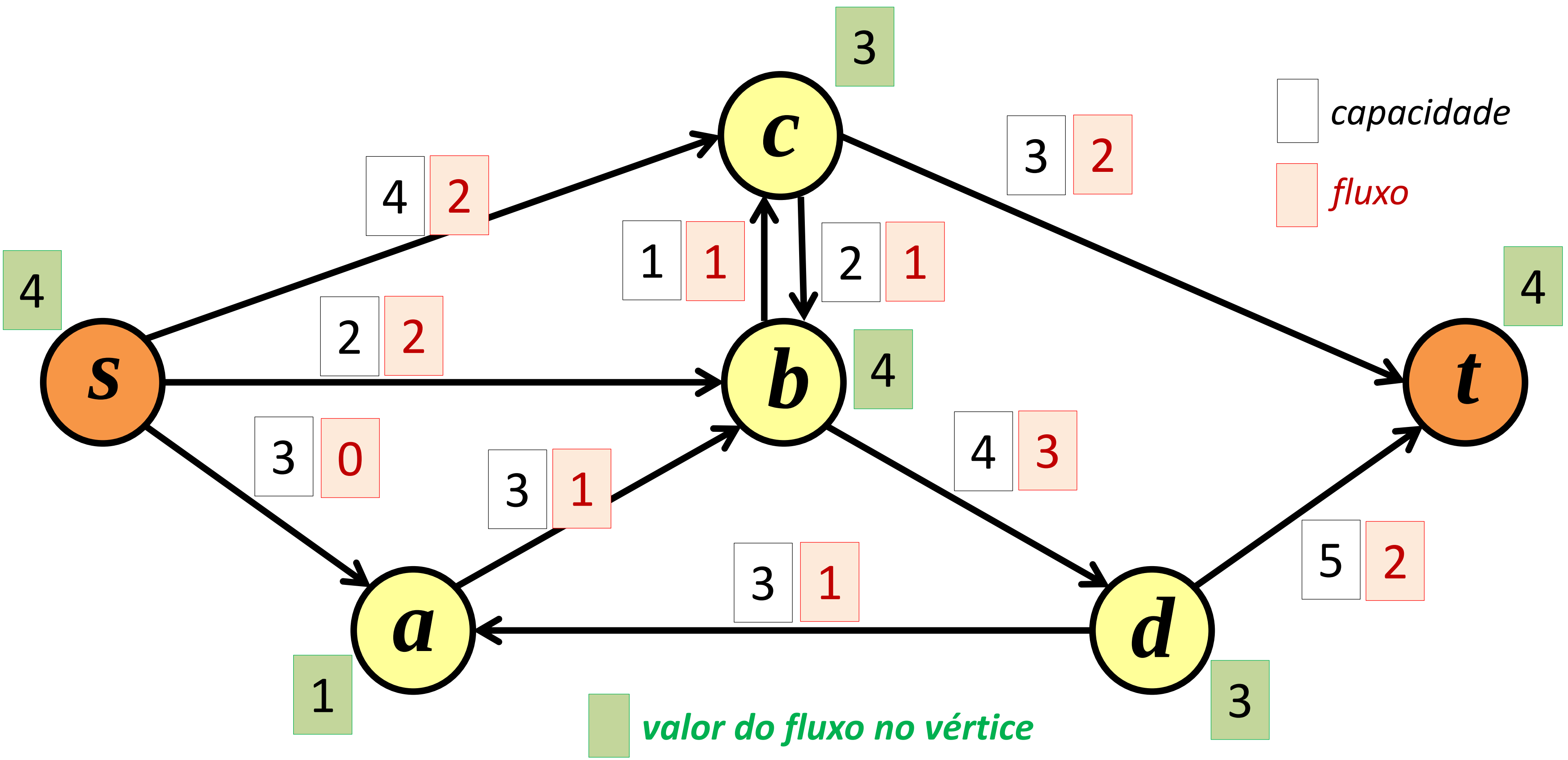
Fluxos em redes

O valor do fluxo em $v \neq s,t$ vale $\Sigma_x\ f(x, v)$ (ou $\Sigma_z\ f(v, z)$).
O valor do fluxo em s é $\Sigma_z\ f(\mathbf{s}, z)$, e em t é $\Sigma_x\ f(x, \mathbf{t})$.



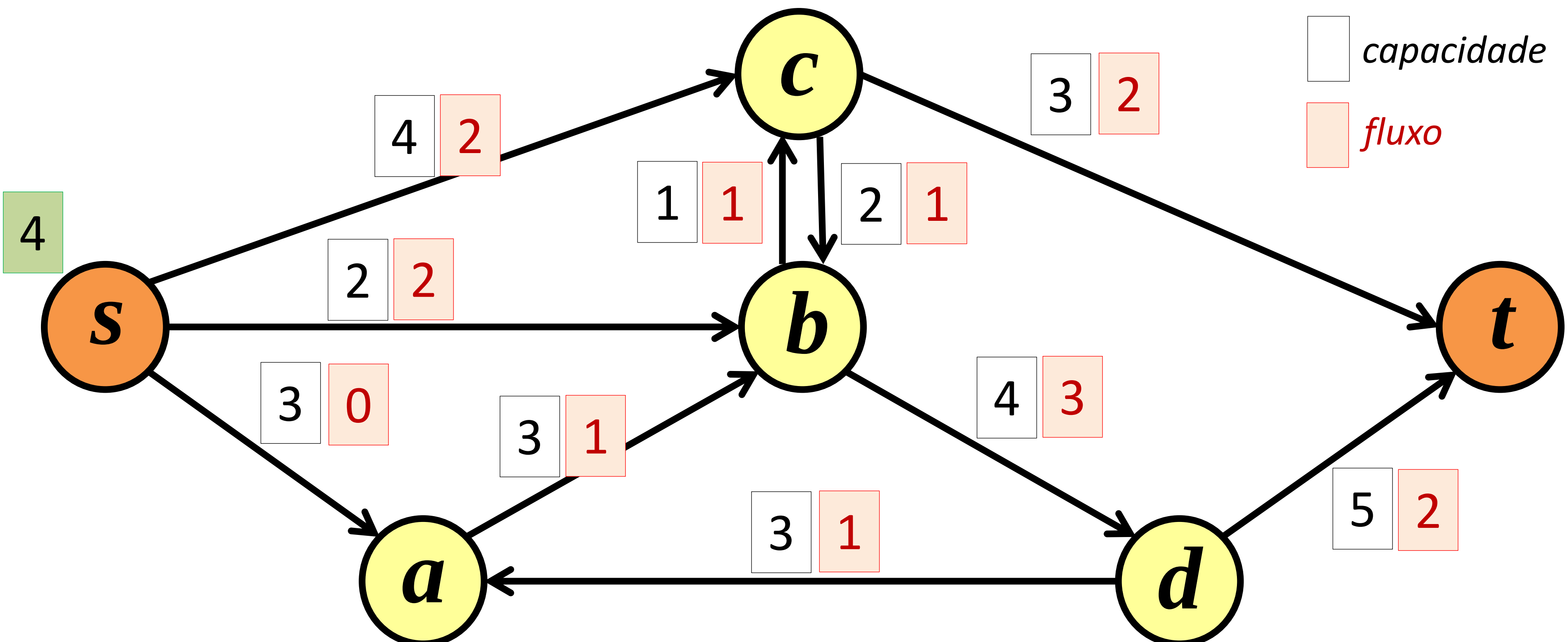
Fluxos em redes

O valor do fluxo na rede D é denotado por $f(D)$ e é definido como $f(D) = \Sigma_z\ f(\mathbf{s}, z)$ (valor do fluxo em s)



Fluxos em redes

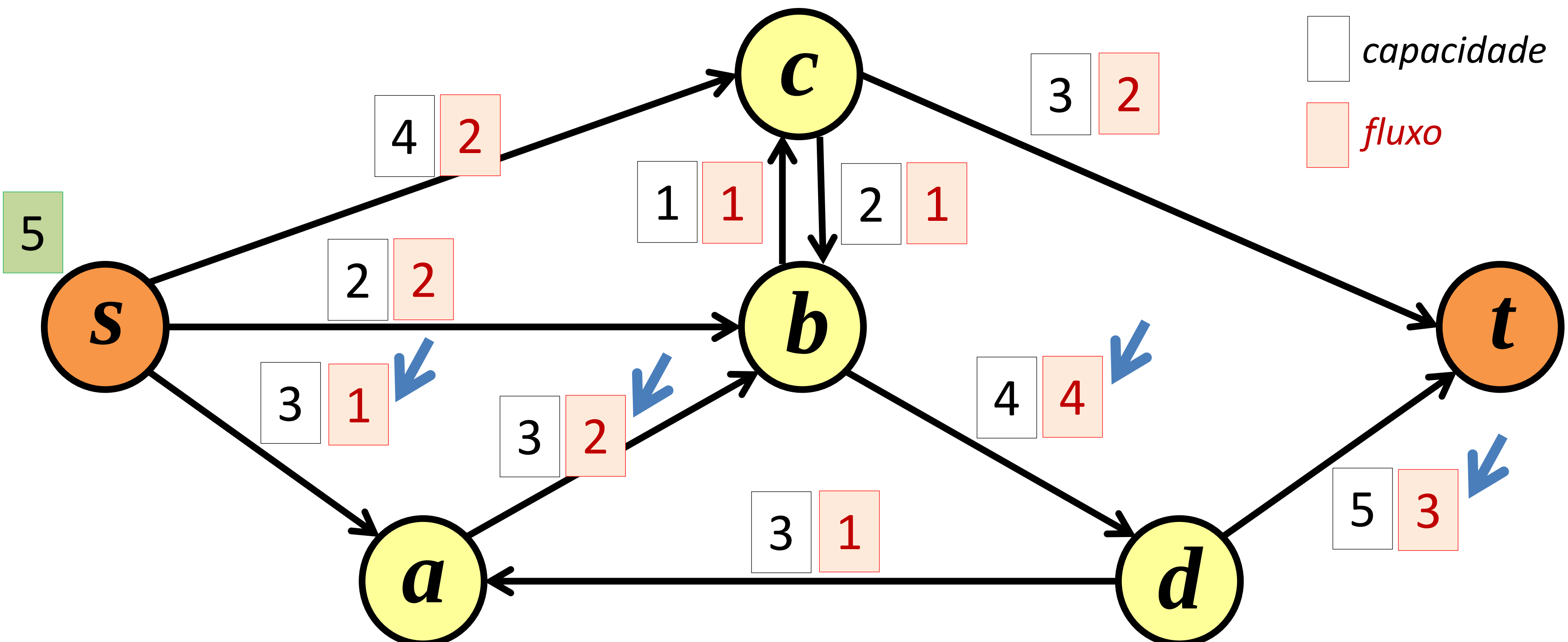
O valor do fluxo na rede D é denotado por $f(D)$ e é definido como $f(D) = \Sigma_z\ f(\mathbf{s}, z)$ (valor do fluxo em s)



valor do fluxo na rede é 4

Fluxos em redes

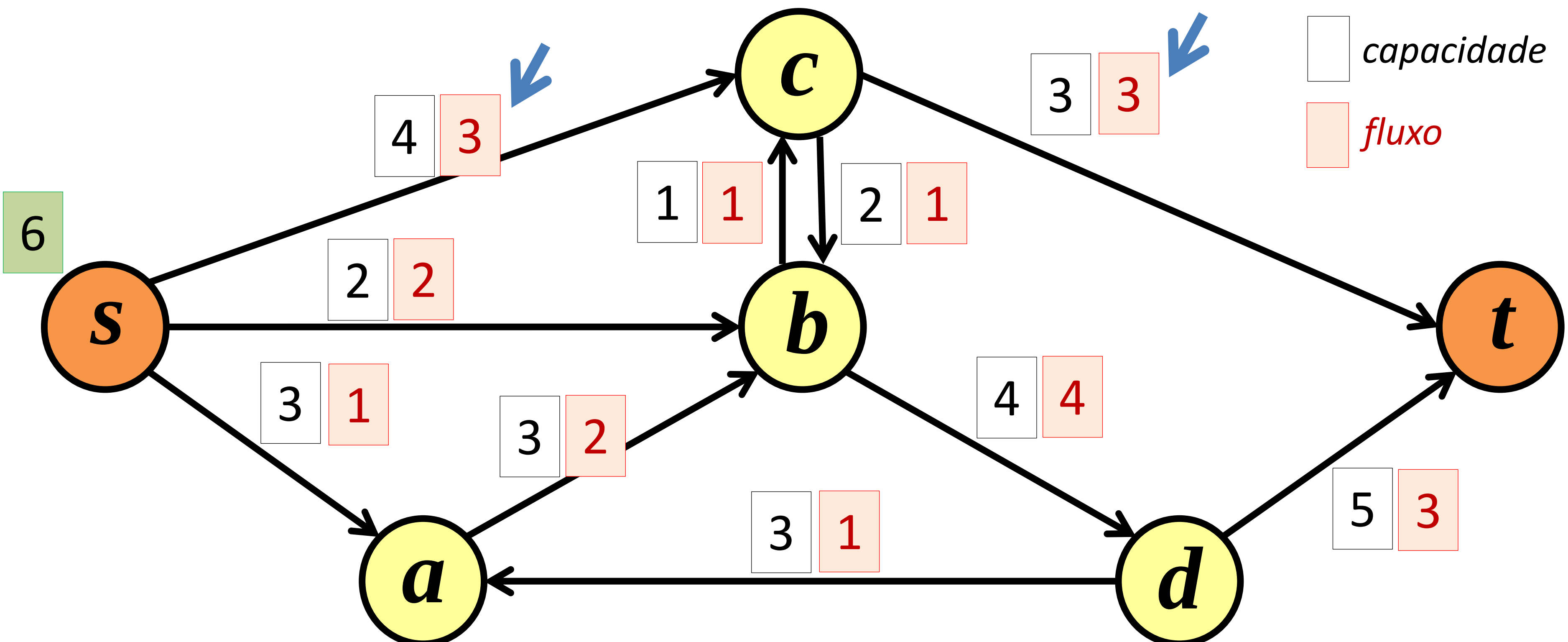
O valor do fluxo na rede D é denotado por $f\,(D)$ e é definido como $f\,(D) = \Sigma_z\; f\,(\mathbf{s}, z)$ (valor do fluxo em s)



valor do fluxo na rede é 5

Fluxos em redes

O valor do fluxo na rede D é denotado por $f(D)$ e é definido como $f(D) = \Sigma_z\ f(\mathbf{s}, z)$ (valor do fluxo em s)

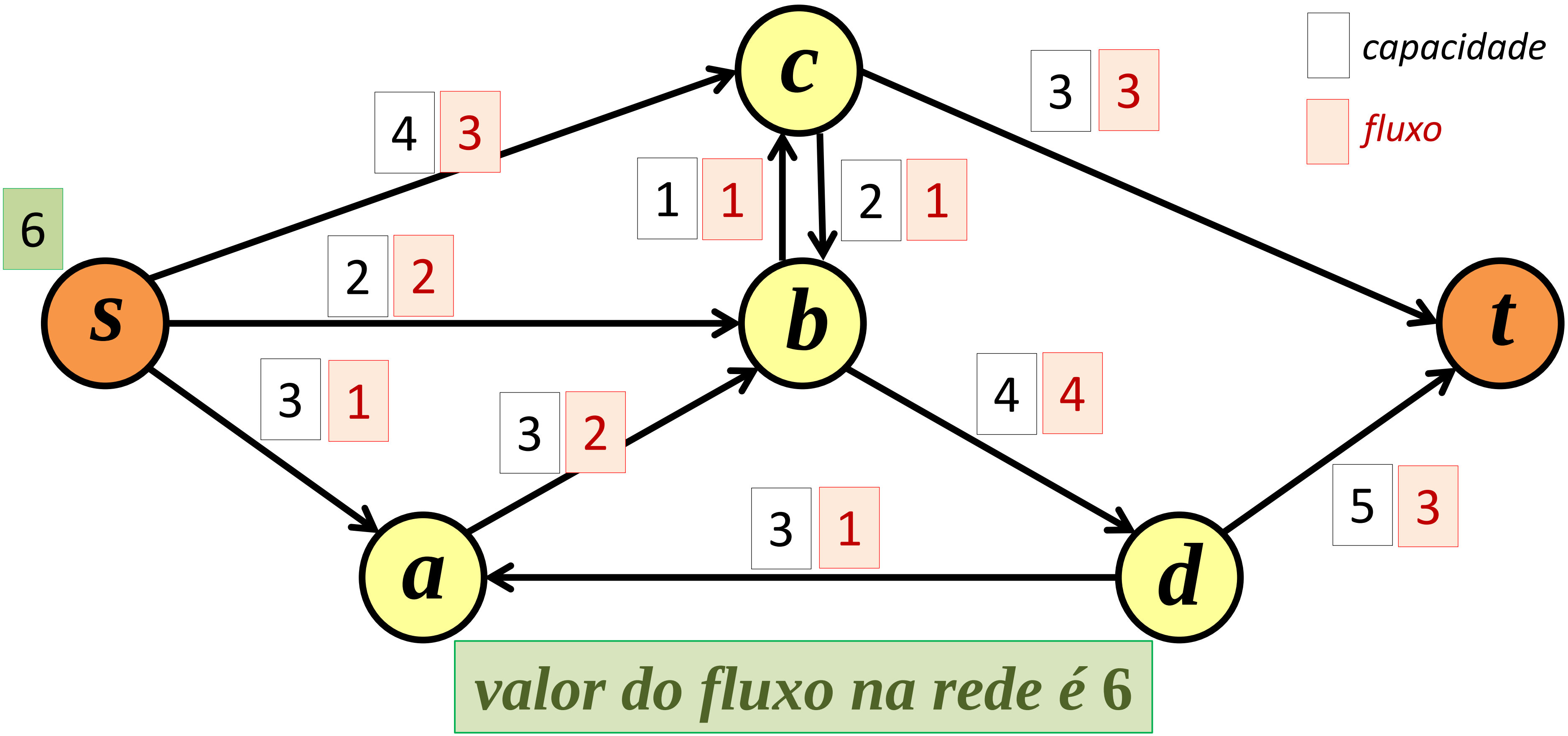


valor do fluxo na rede é 6

Fluxos em redes

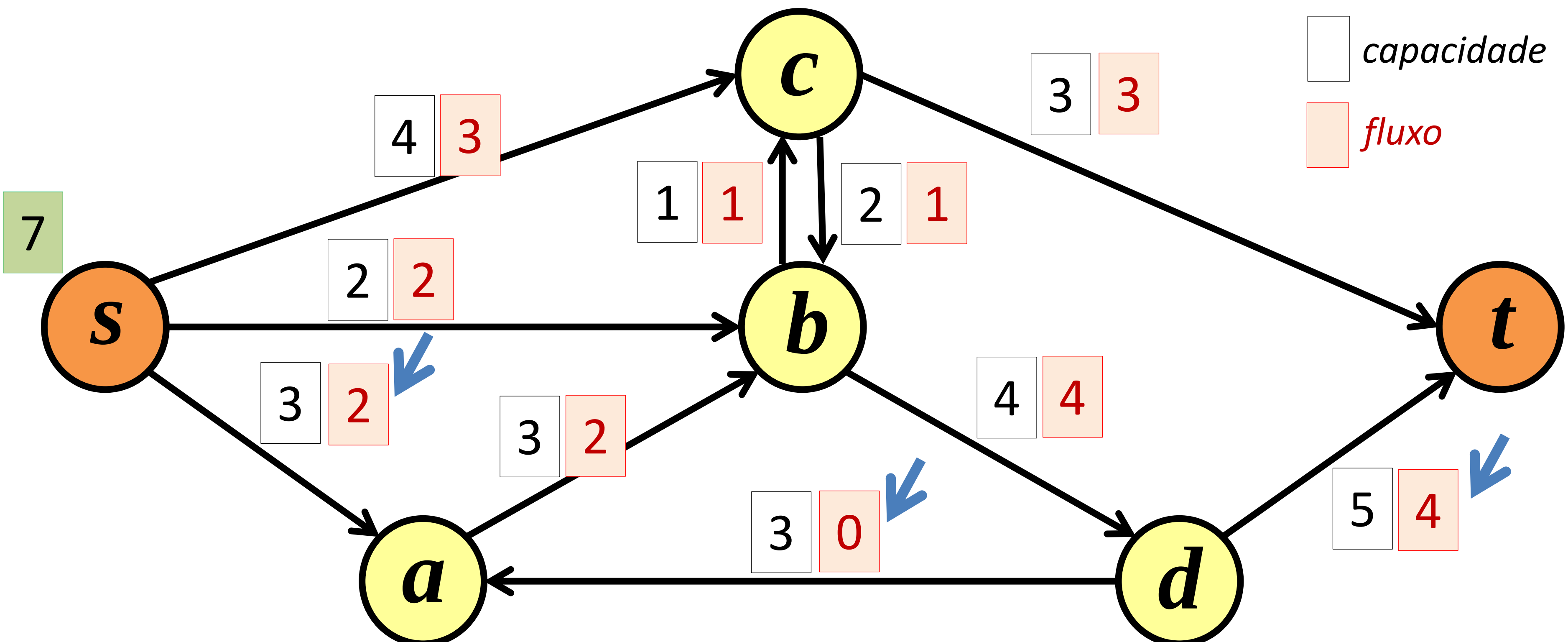
Problema do Fluxo Máximo:

Dada uma rede D, encontrar um fluxo f de valor máximo.



Fluxos em redes

Problema do Fluxo Máximo:
Dada uma rede D, encontrar um fluxo f de valor máximo.

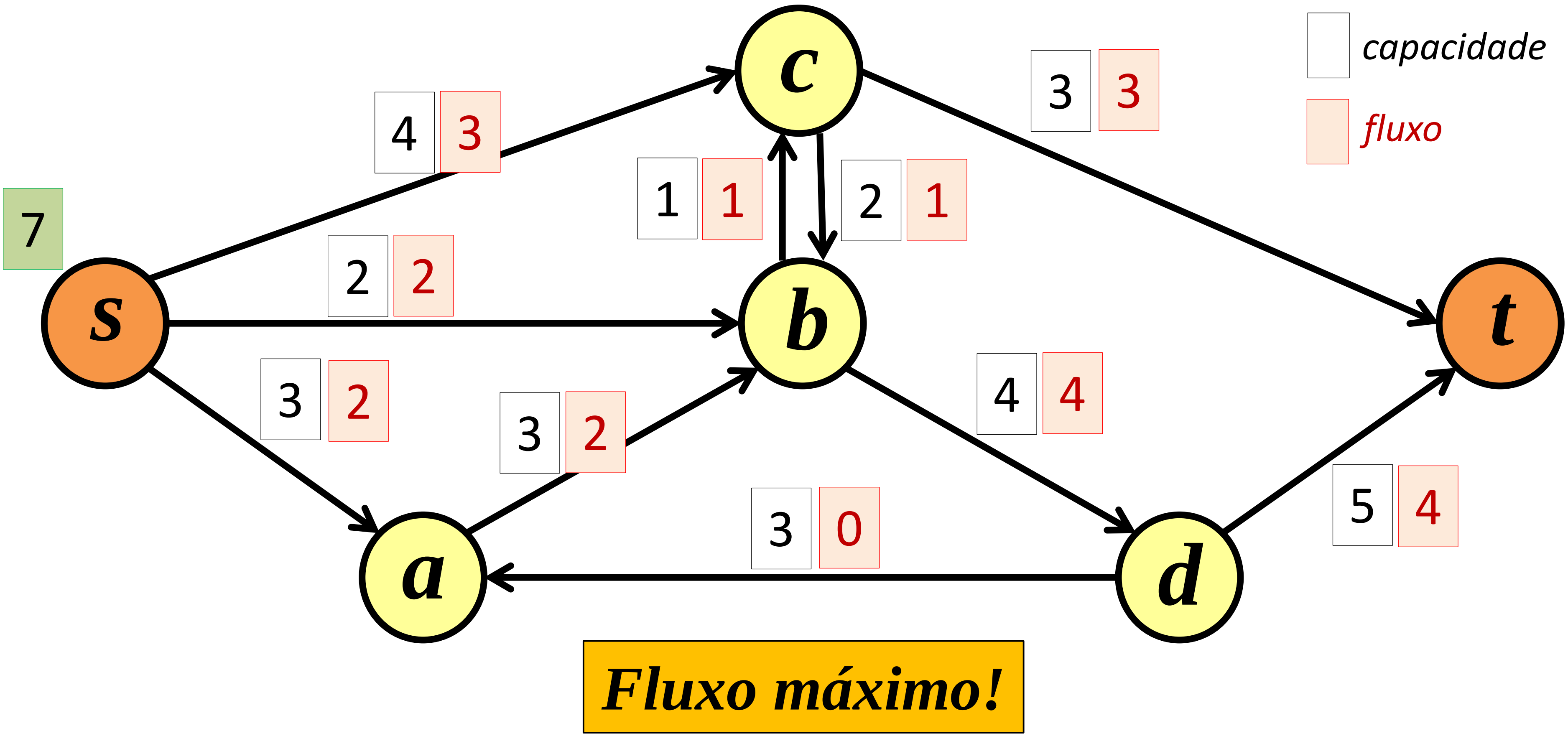


valor do fluxo na rede é 7

Fluxos em redes

Problema do Fluxo Máximo:

Dada uma rede *D*, encontrar um fluxo *f* de valor máximo.

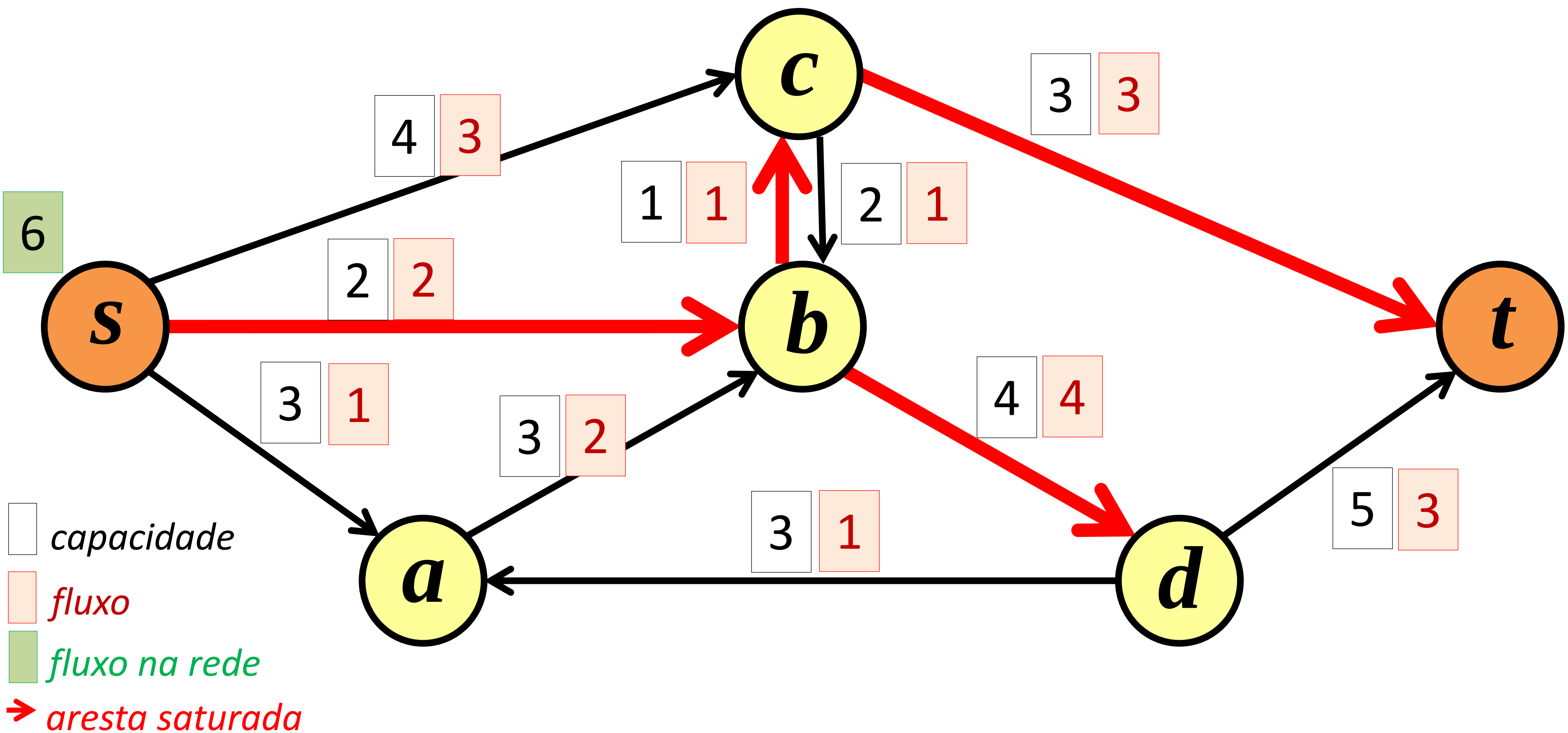


Fluxos em redes

Def.: Uma aresta está ***saturada*** quando $f(e) = c(e)$.

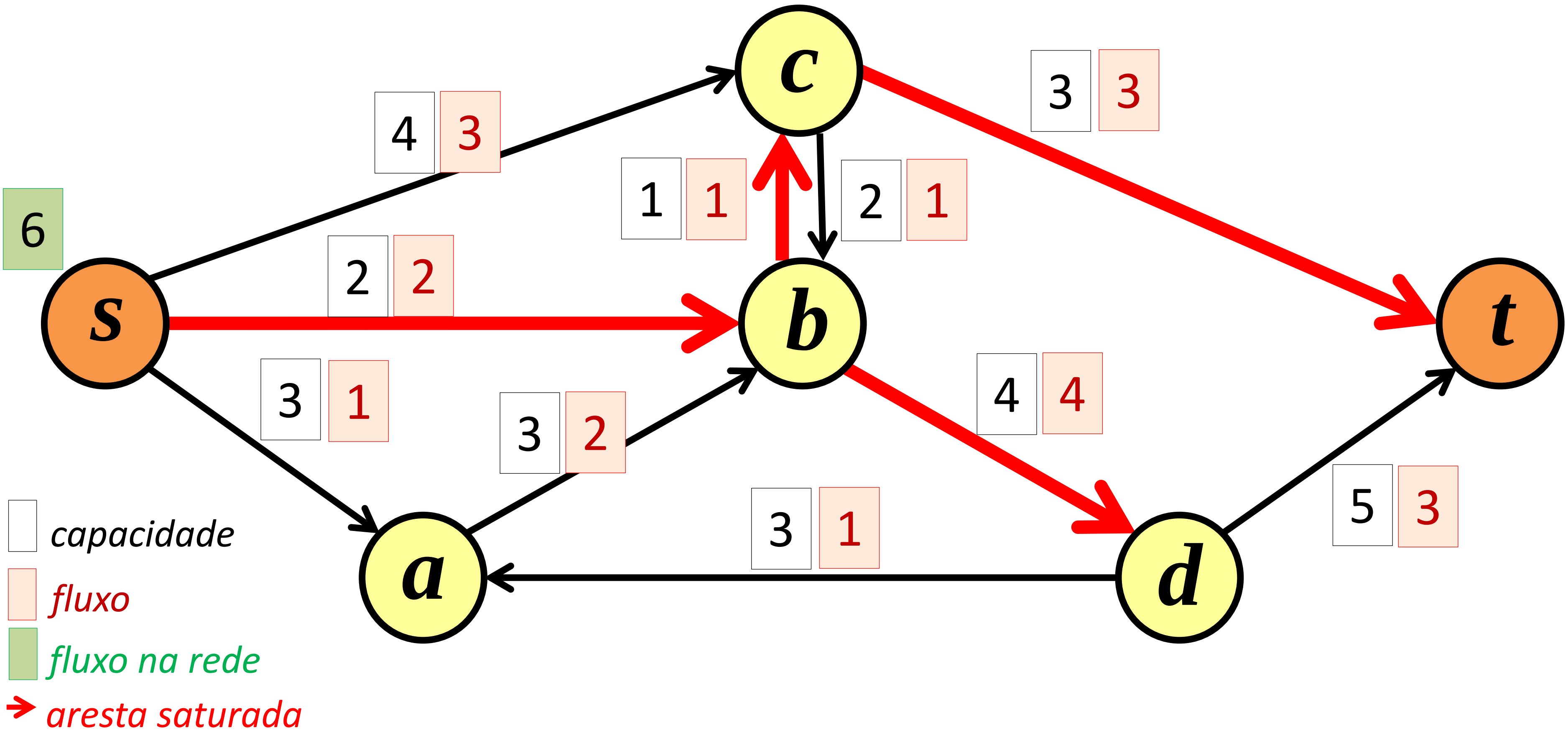
Fluxos em redes

Def.: Uma aresta está ***saturada*** quando $f(e) = c(e)$.



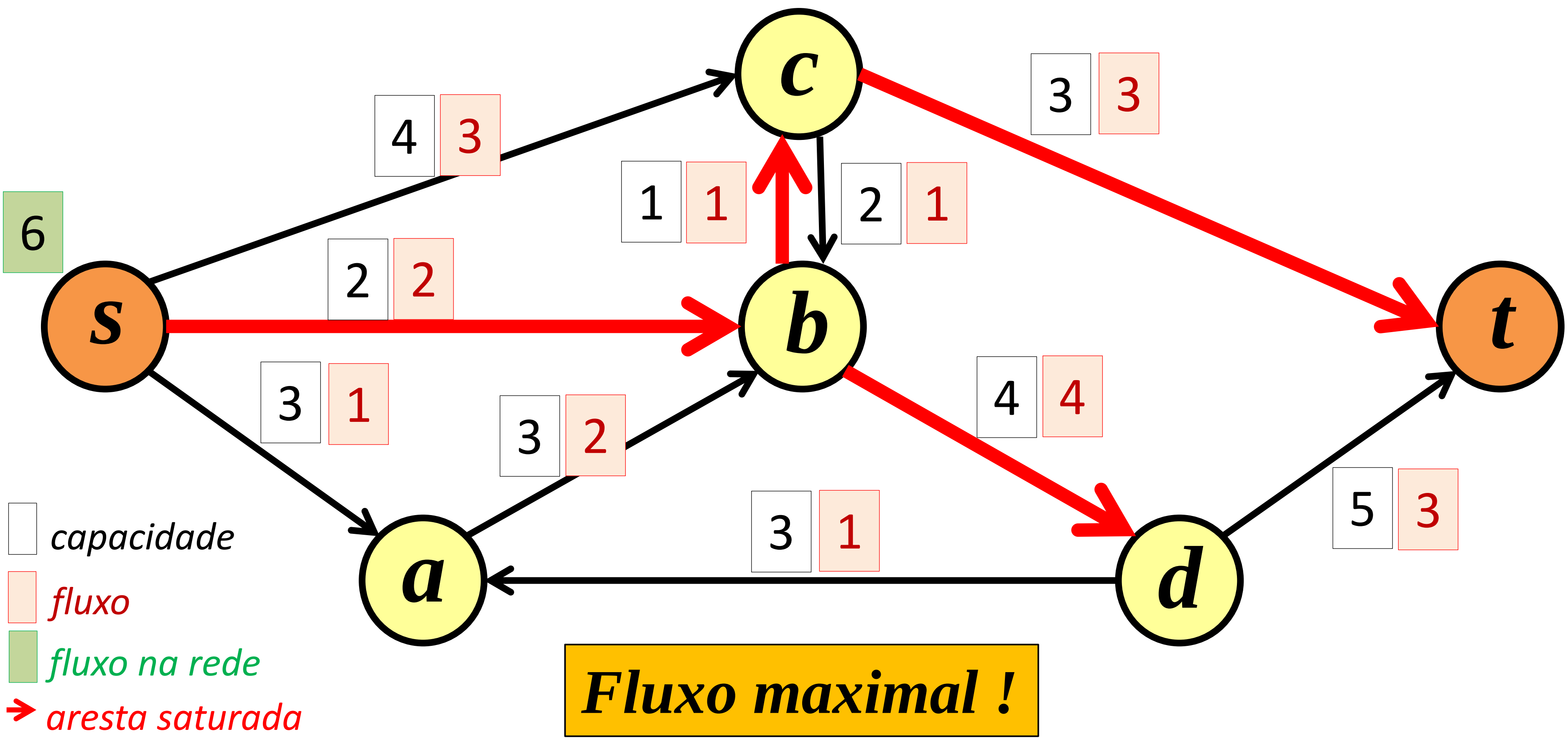
Fluxos em redes

Def.: Um fluxo é dito ***maximal*** quando todo caminho direcionado de ***s*** a ***t*** contém uma aresta saturada.



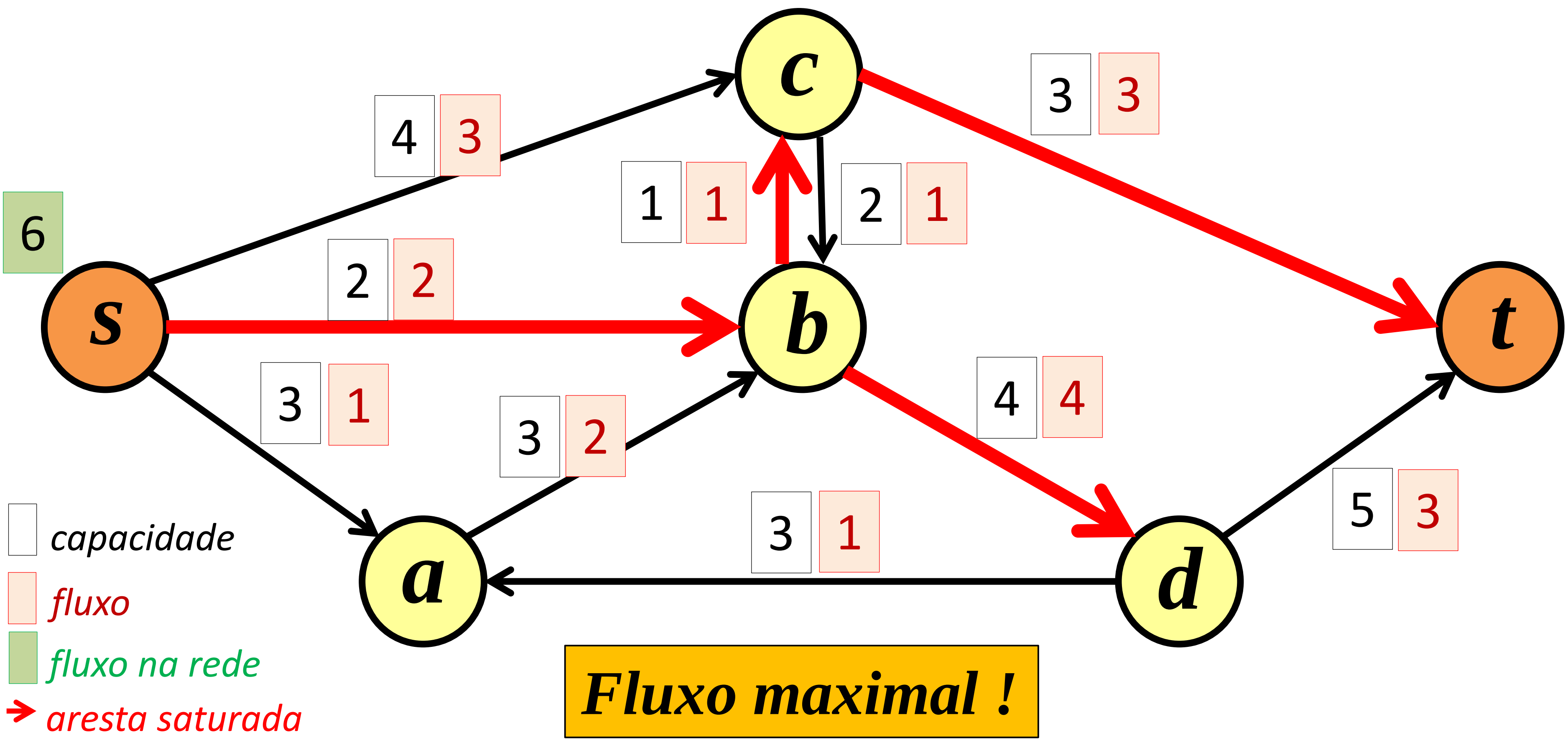
Fluxos em redes

Def.: Um fluxo é dito ***maximal*** quando todo caminho direcionado de ***s*** a ***t*** contém uma aresta saturada.



Fluxos em redes

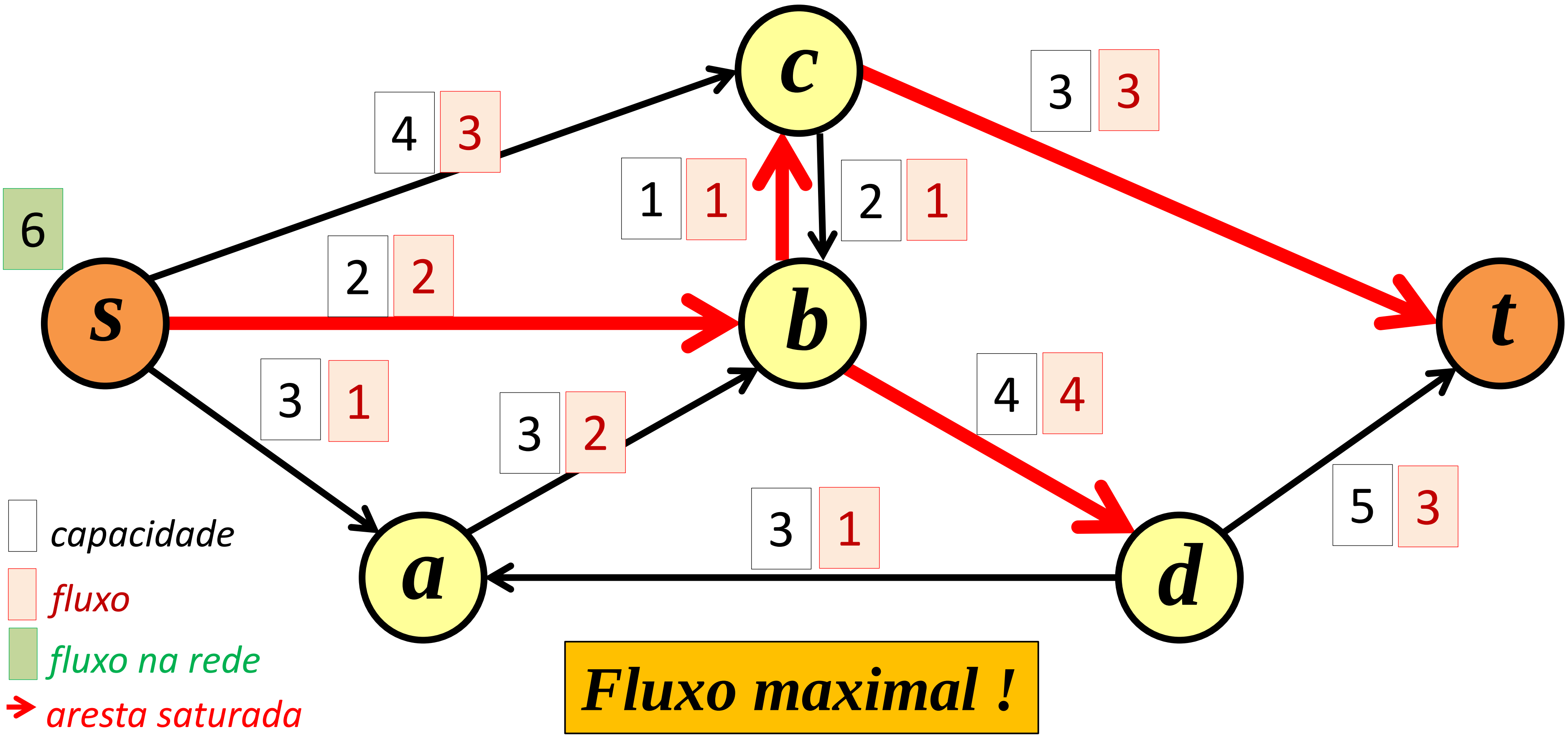
Um fluxo é maximal quando ***não é possível aumentar o seu valor apenas com acréscimo de fluxo nas arestas.***



Fluxo maximal !

Fluxos em redes

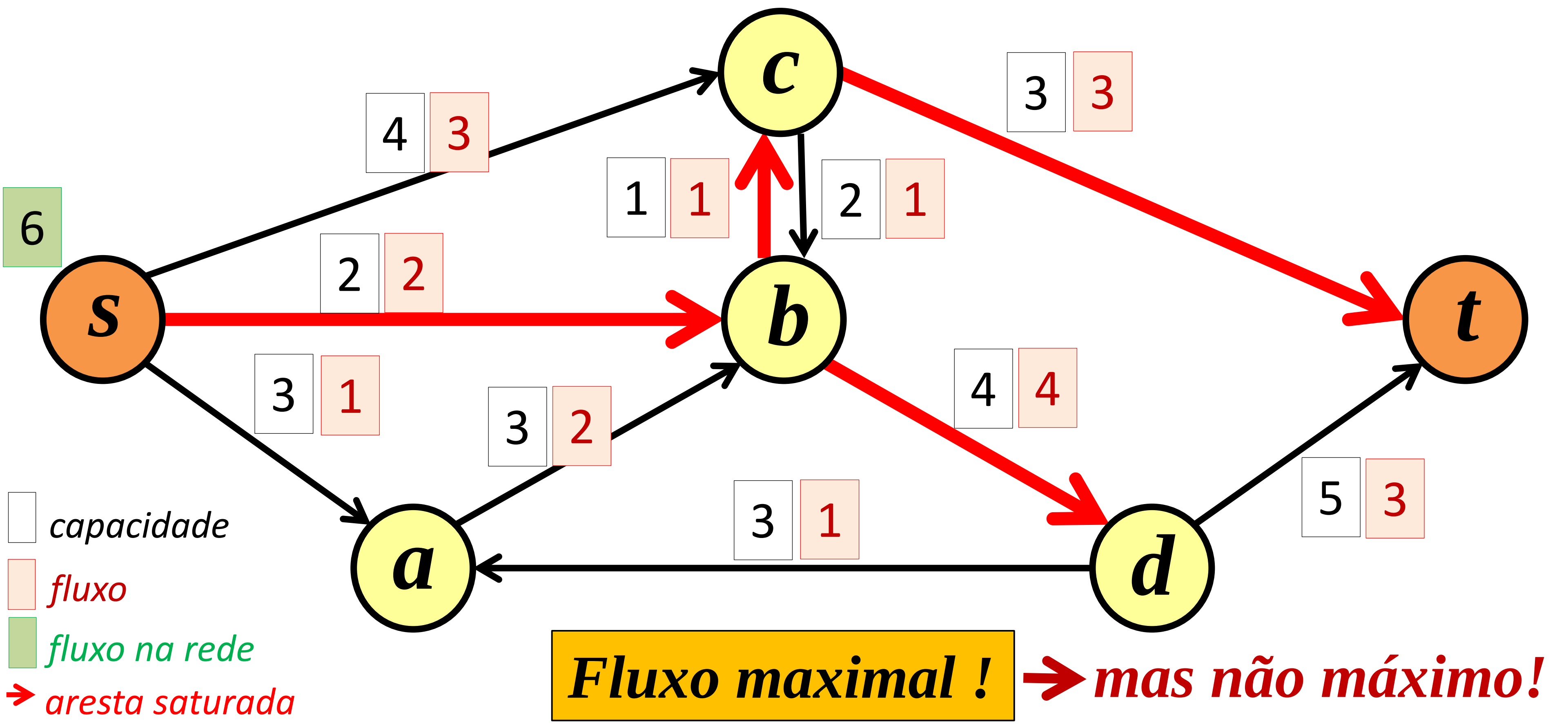
- Todo fluxo máximo é maximal.
- Nem todo fluxo maximal é máximo.



Fluxo maximal !

Fluxos em redes

- Todo fluxo máximo é maximal.
- Nem todo fluxo maximal é máximo.



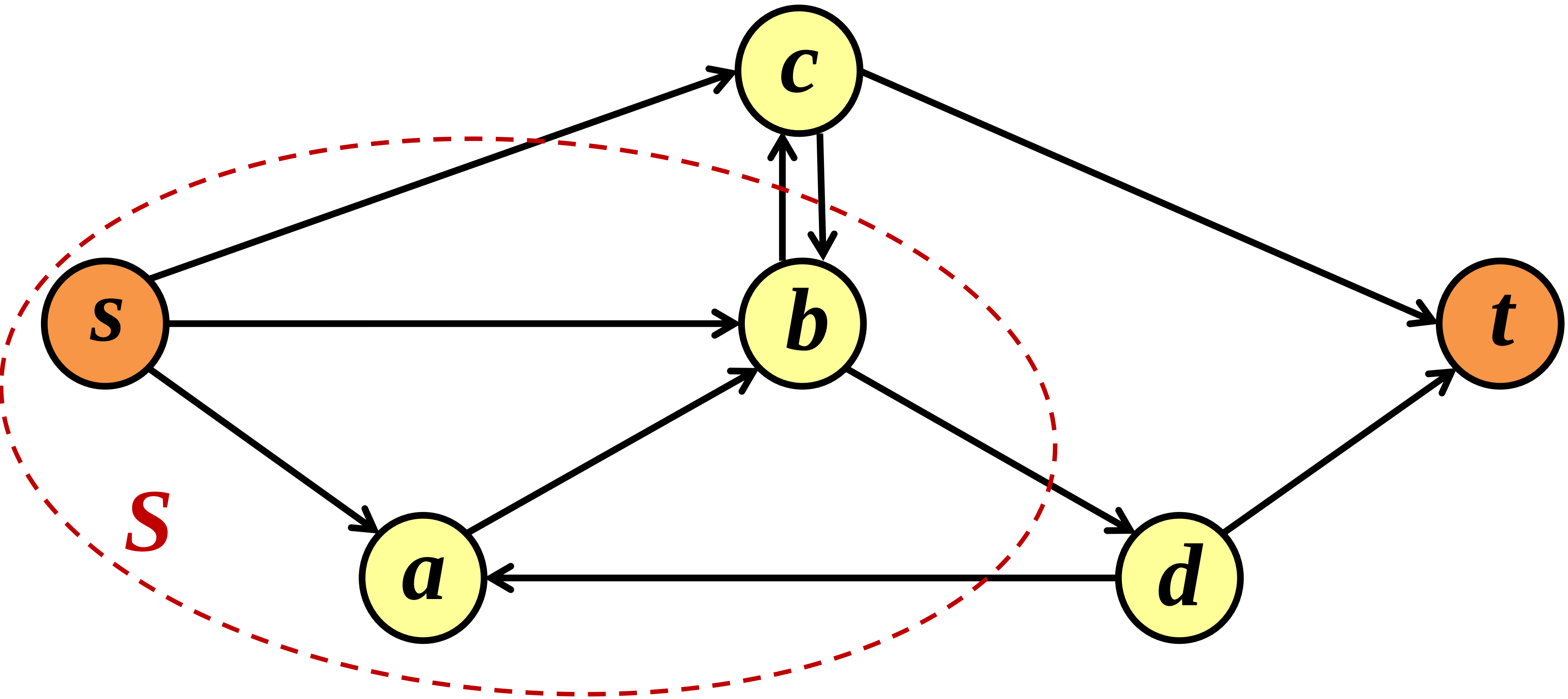
Fluxo maximal ! → **mas não máximo!**

Fluxos em redes

Def.: Seja $S \subseteq V$ tal que $s \in S$ e $t \notin S$. Seja $S' = V - S$.
Um ***corte*** (S, S') é o conjunto de arestas com um extremo em S e o outro em S'.

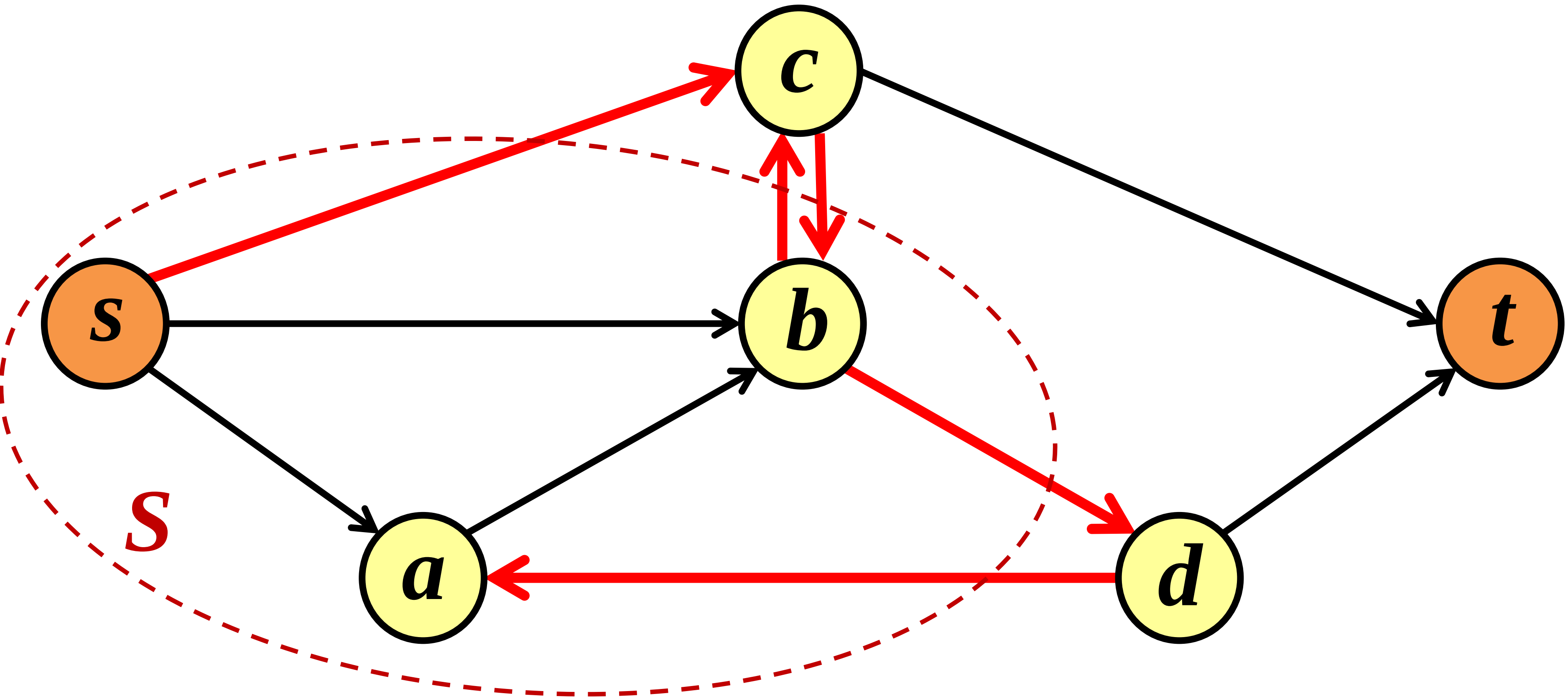
Fluxos em redes

Def.: Seja $S \subseteq V$ tal que $s \in S$ e $t \notin S$. Seja $S' = V - S$.
Um ***corte*** (S, S') é o conjunto de arestas com um extremo em S e o outro em S'.



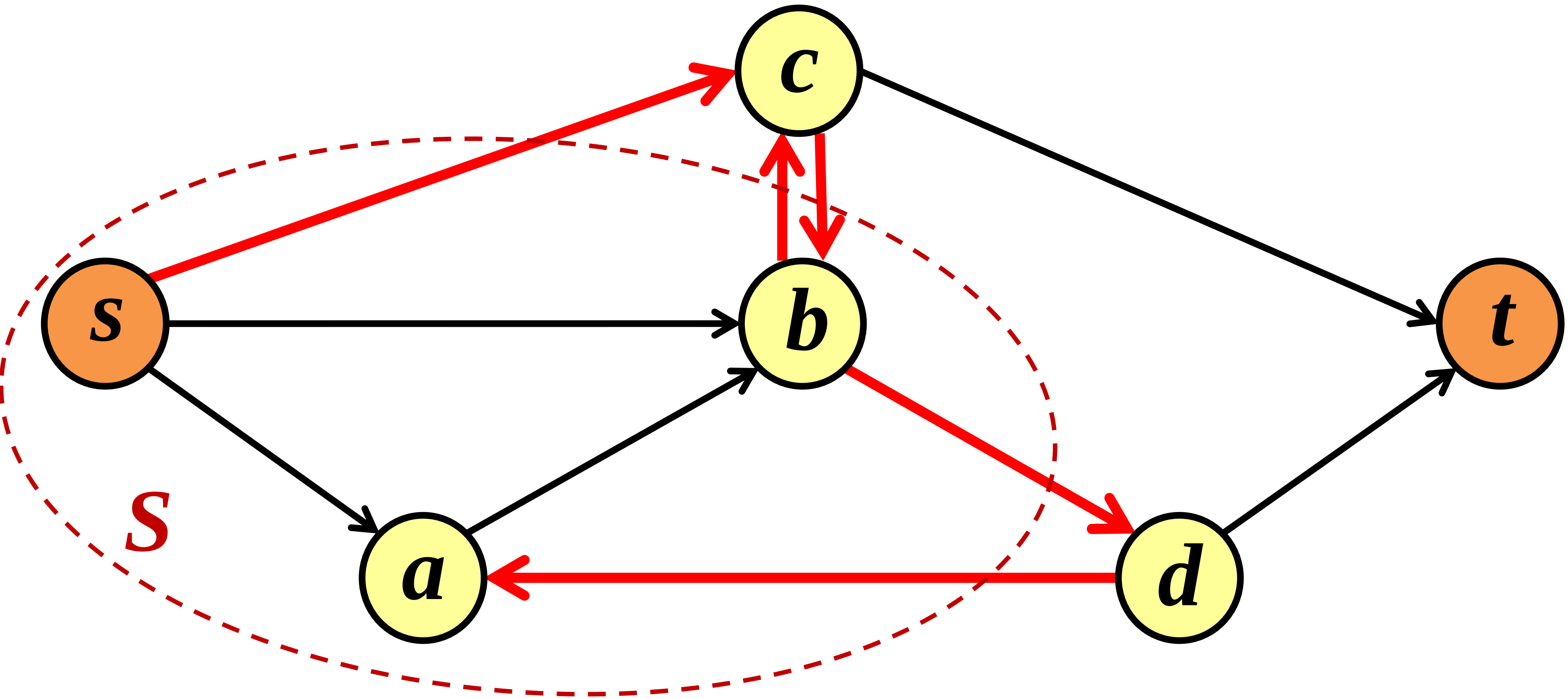
Fluxos em redes

Def.: Seja $S \subseteq V$ tal que $s \in S$ e $t \notin S$. Seja $S' = V - S$.
Um ***corte*** (S, S') é o conjunto de arestas com um extremo em S e o outro em S'.



Fluxos em redes

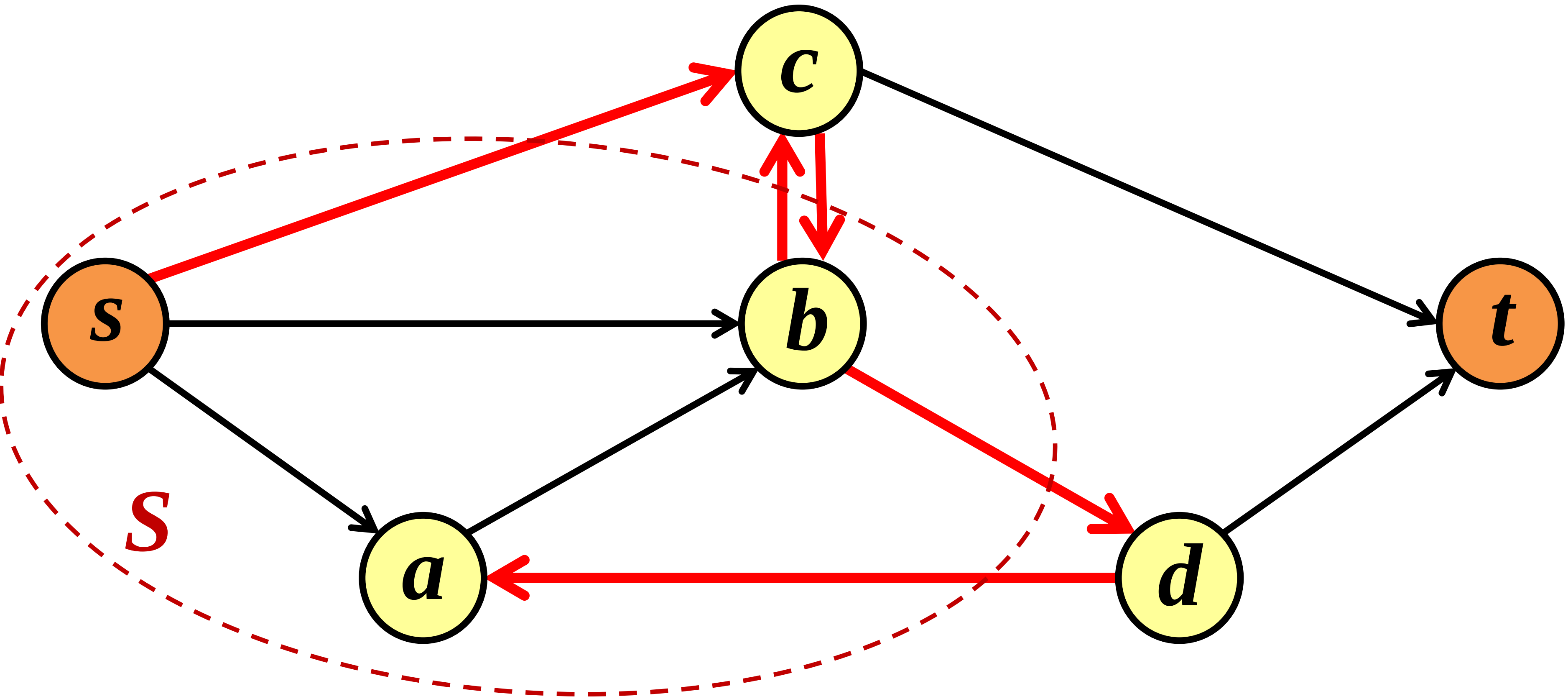
Idéia: Todo caminho direcionado de ***s*** a ***t*** tem que passar por alguma aresta do corte.



Fluxos em redes

$S = \{s, a, b\}$ $S' = \{t, c, d\}$

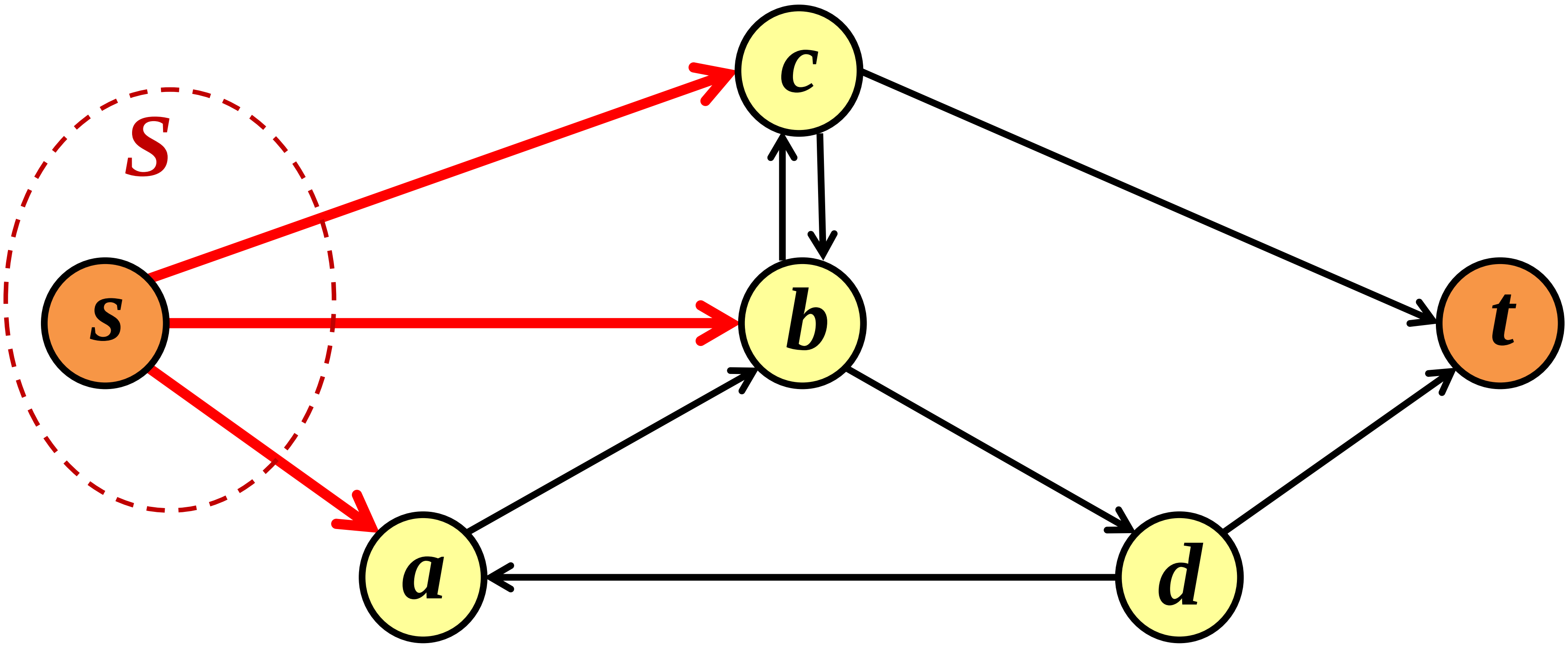
$(S, S') = \{ (s, c), (b, c), (c, b), (b, d), (d, a) \}$



Fluxos em redes

$S = \{s\}$ $\qquad\qquad S' = \{t, a, b, c, d\}$

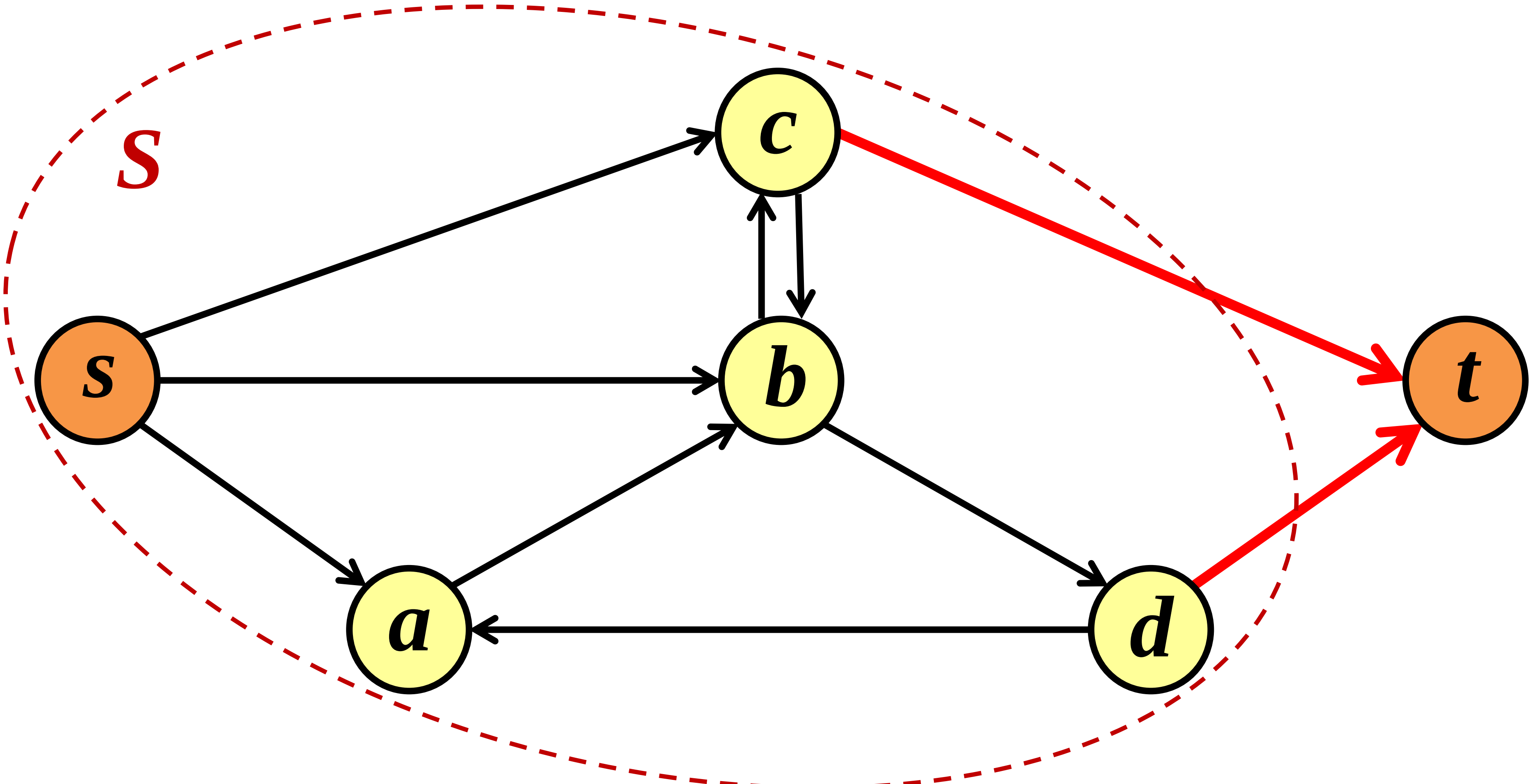
$(S, S') = \{\ (s, a), (s, b), (s, c)\ \}$



Fluxos em redes

$S = \{s, a, b, c, d\}$ $\quad\quad$ $S' = \{t\}$

$(S, S') = \{ (c, t), (d, t) \}$



Fluxos em redes

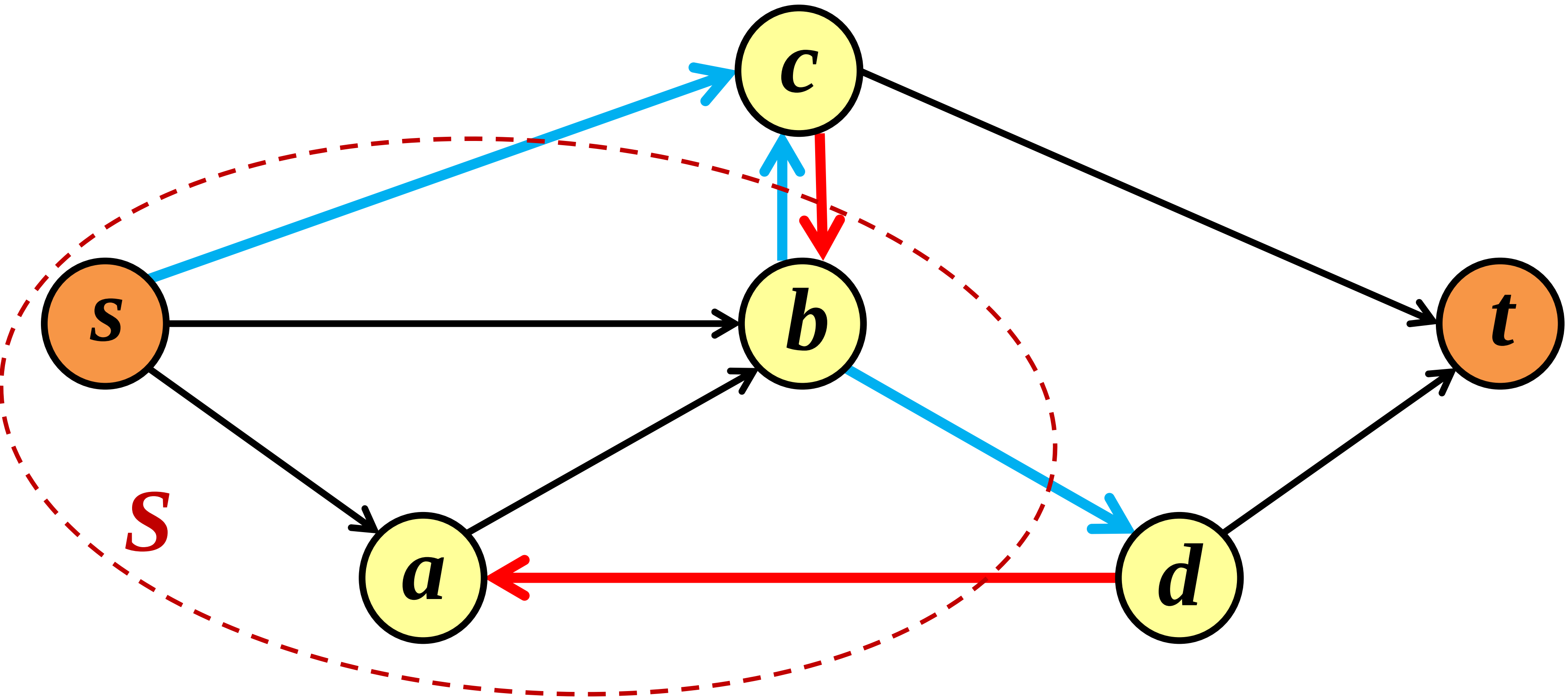
Def.: $\boldsymbol{(S, S')^+}$ é o conj. de arestas que vão de S a S'.

$\boldsymbol{(S, S')^-}$ é o conj. de arestas que vão de S' a S.

Fluxos em redes

Def.: $(S, S')^{+}$ é o conj. de arestas que vão de S a S'.

$(S, S')^{-}$ é o conj. de arestas que vão de S' a S.

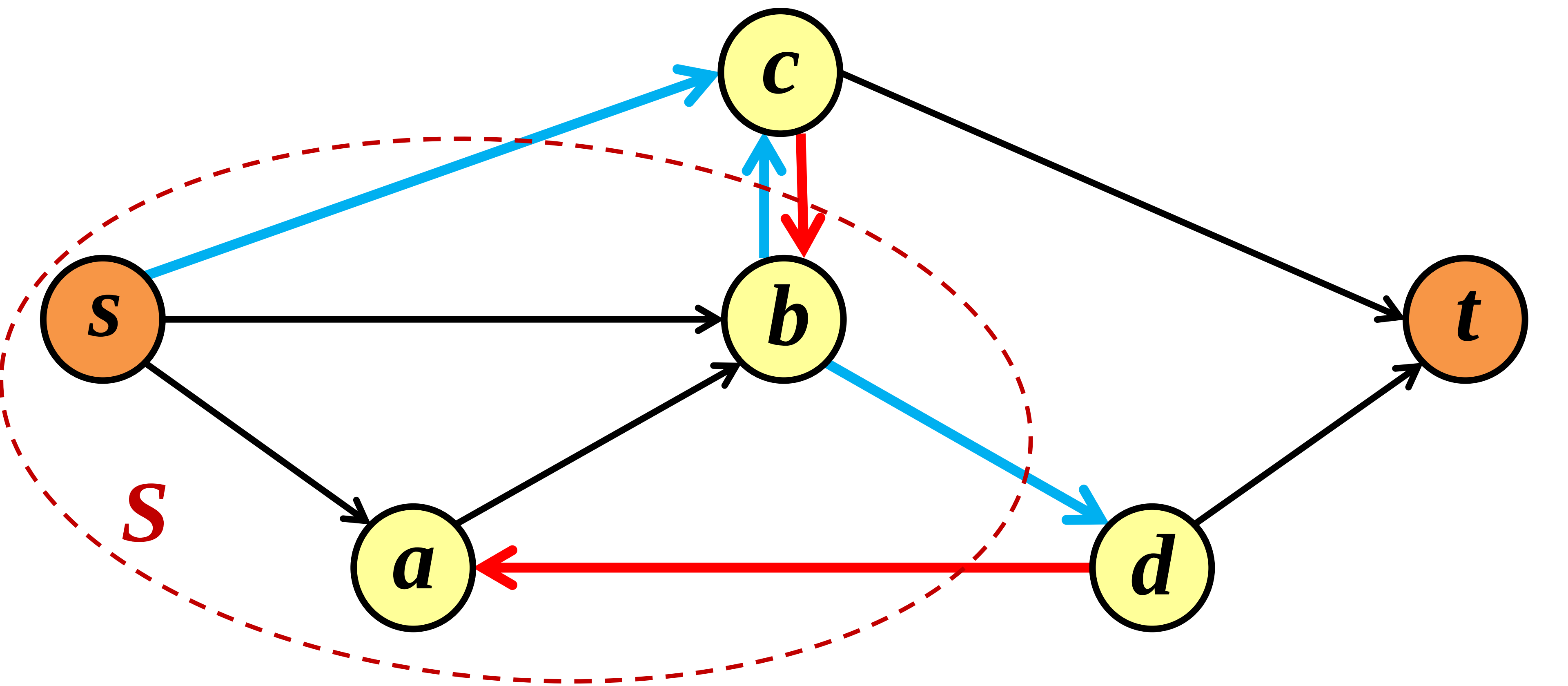


Fluxos em redes

$S = \{\, s, a, b \,\}$

$(S, S')^{+} = \{\, (s, c), (b, c), (b, d) \,\}$

$(S, S')^{-} = \{\, (c, b), (d, a) \,\}$

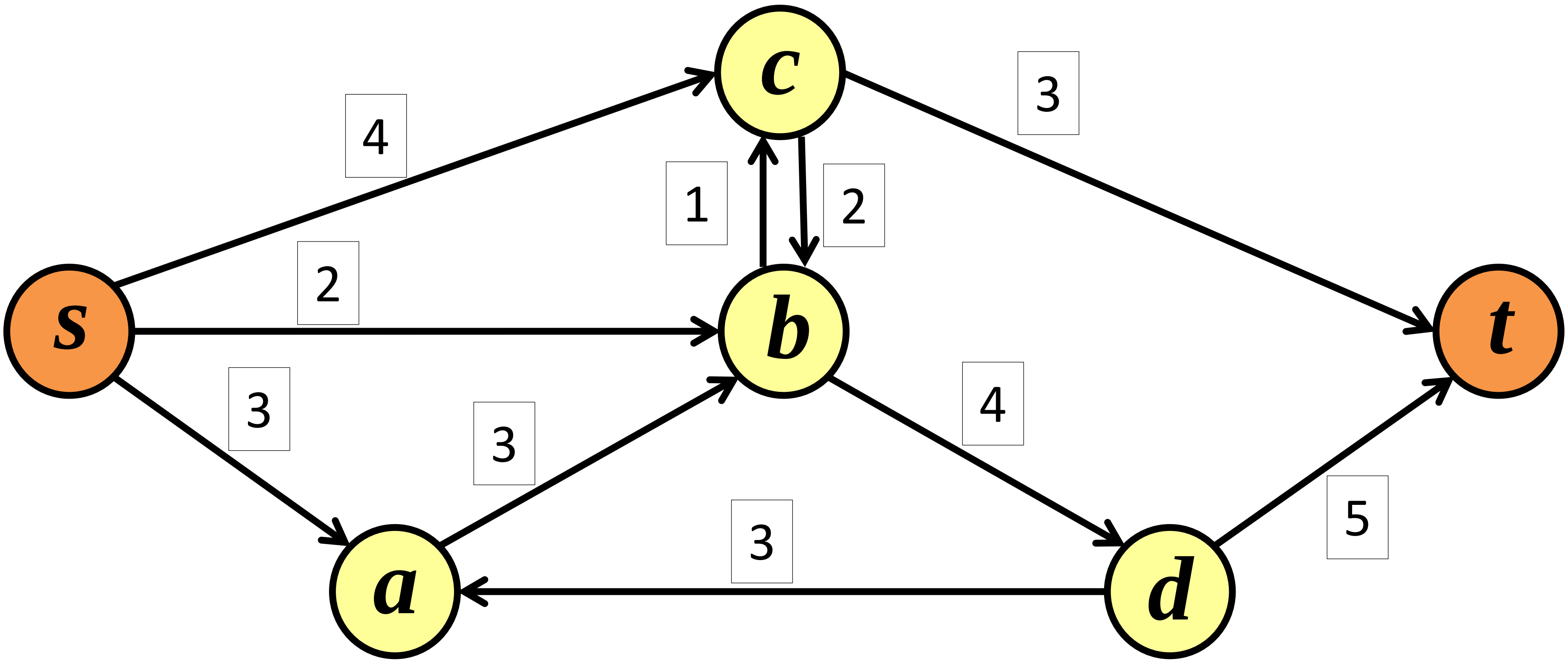


Fluxos em redes

Def.: A ***capacidade c(S, S') de um corte (S, S')*** é a soma das capacidades das arestas de $(S, S')^{+}$.

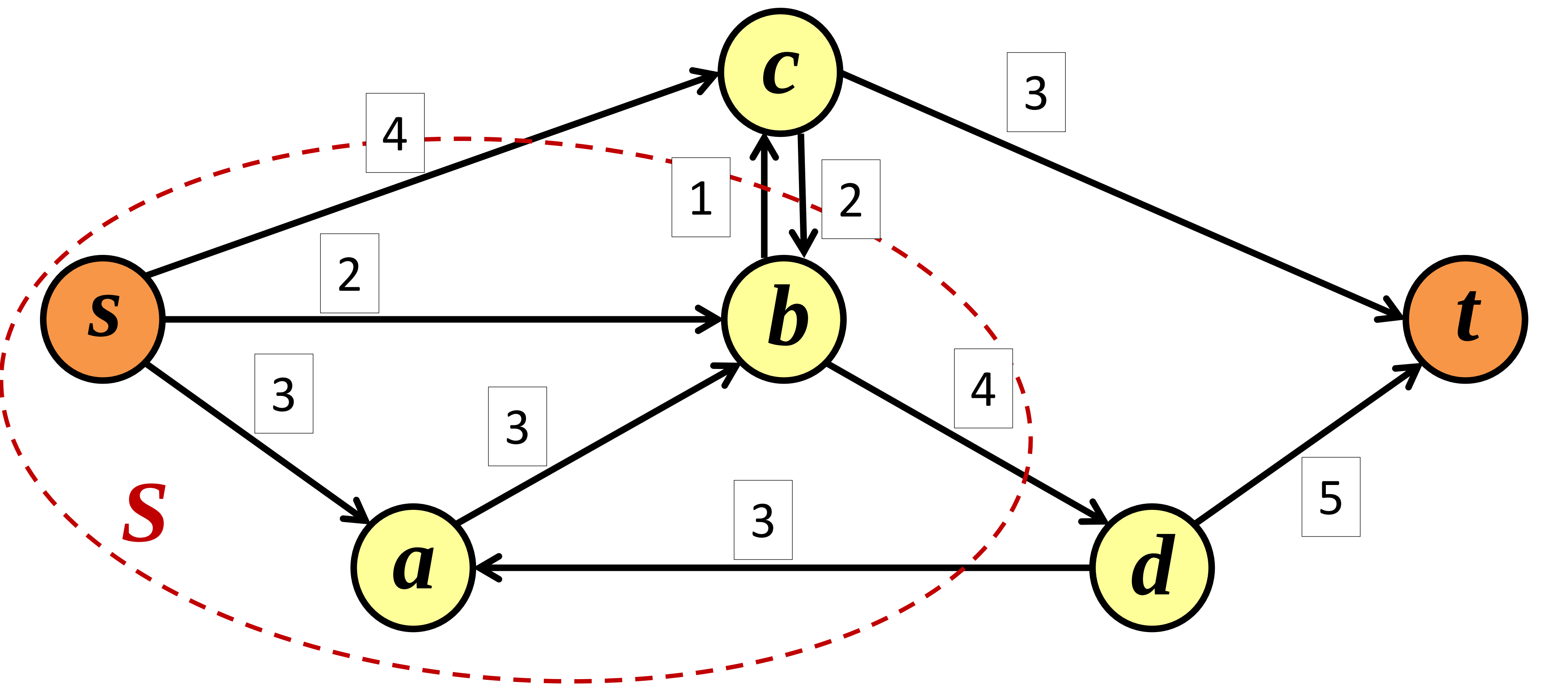
Fluxos em redes

Def.: A ***capacidade c*****(*S*, *S'*)** ***de um corte*** **(*S*, *S'*)** é a soma das capacidades das arestas de **(*S*, *S'*)**$^+$.



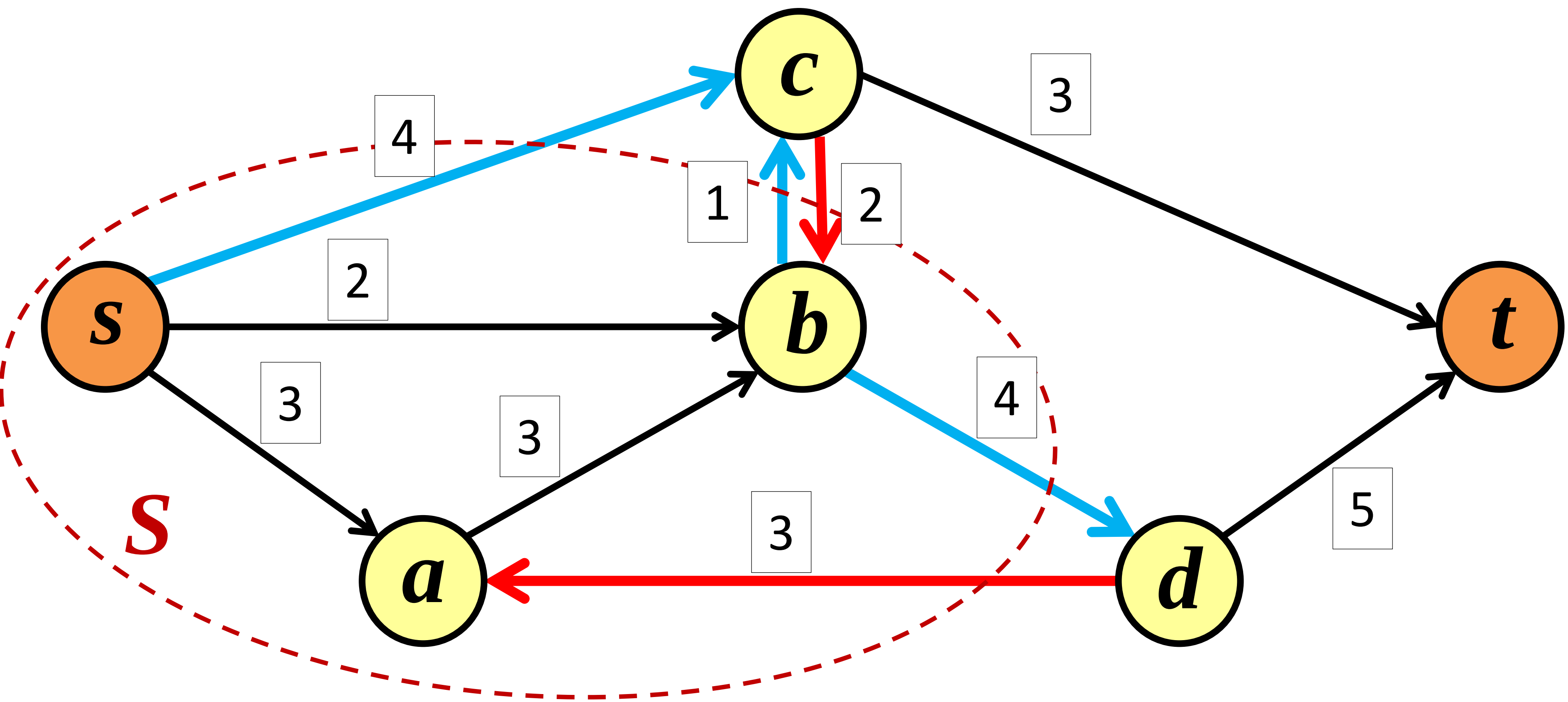
Fluxos em redes

Def.: A ***capacidade c(S, S') de um corte (S, S')*** é a soma das capacidades das arestas de $(S, S')^{+}$.



Fluxos em redes

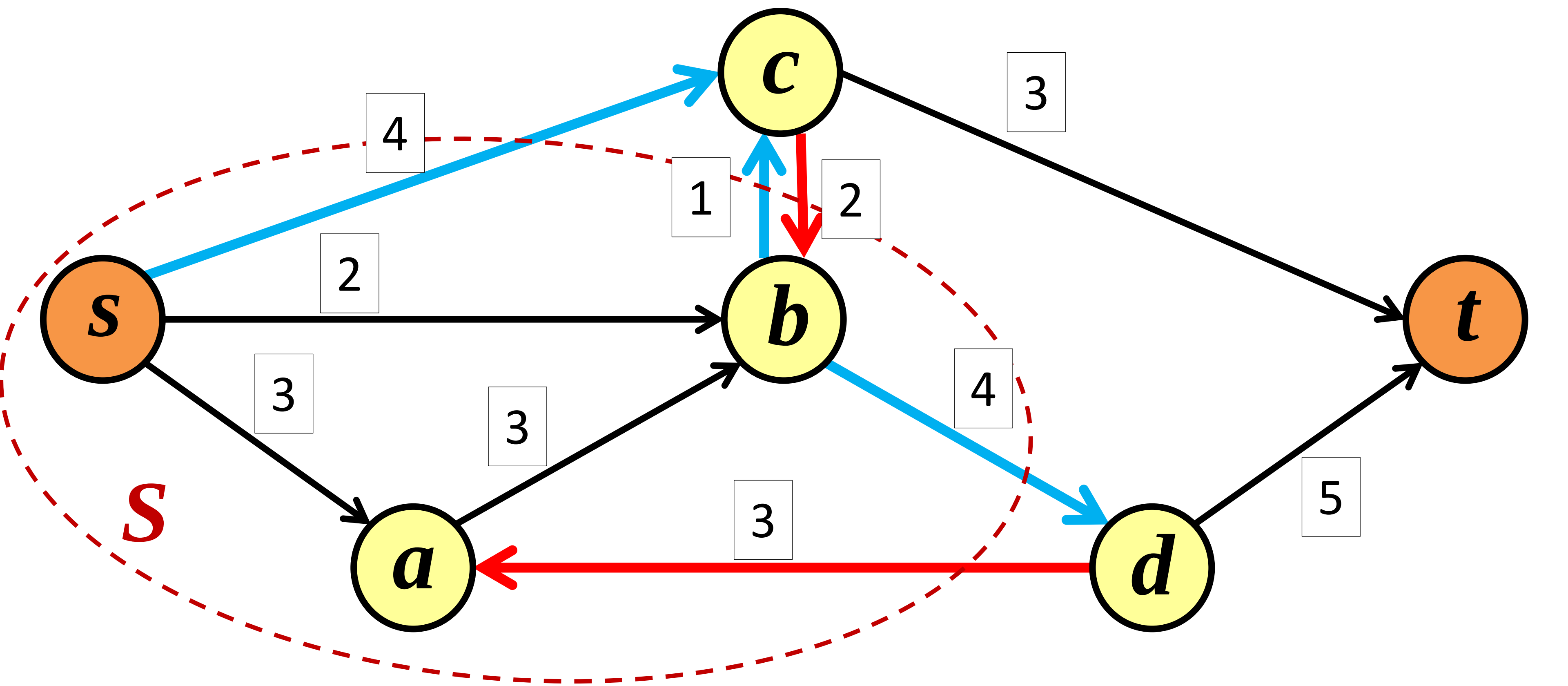
Def.: A ***capacidade c(S, S') de um corte (S, S')*** é a soma das capacidades das arestas de ***(S, S')***$^{+}$.



Fluxos em redes

$$S = \{\, s, a, b \,\}$$

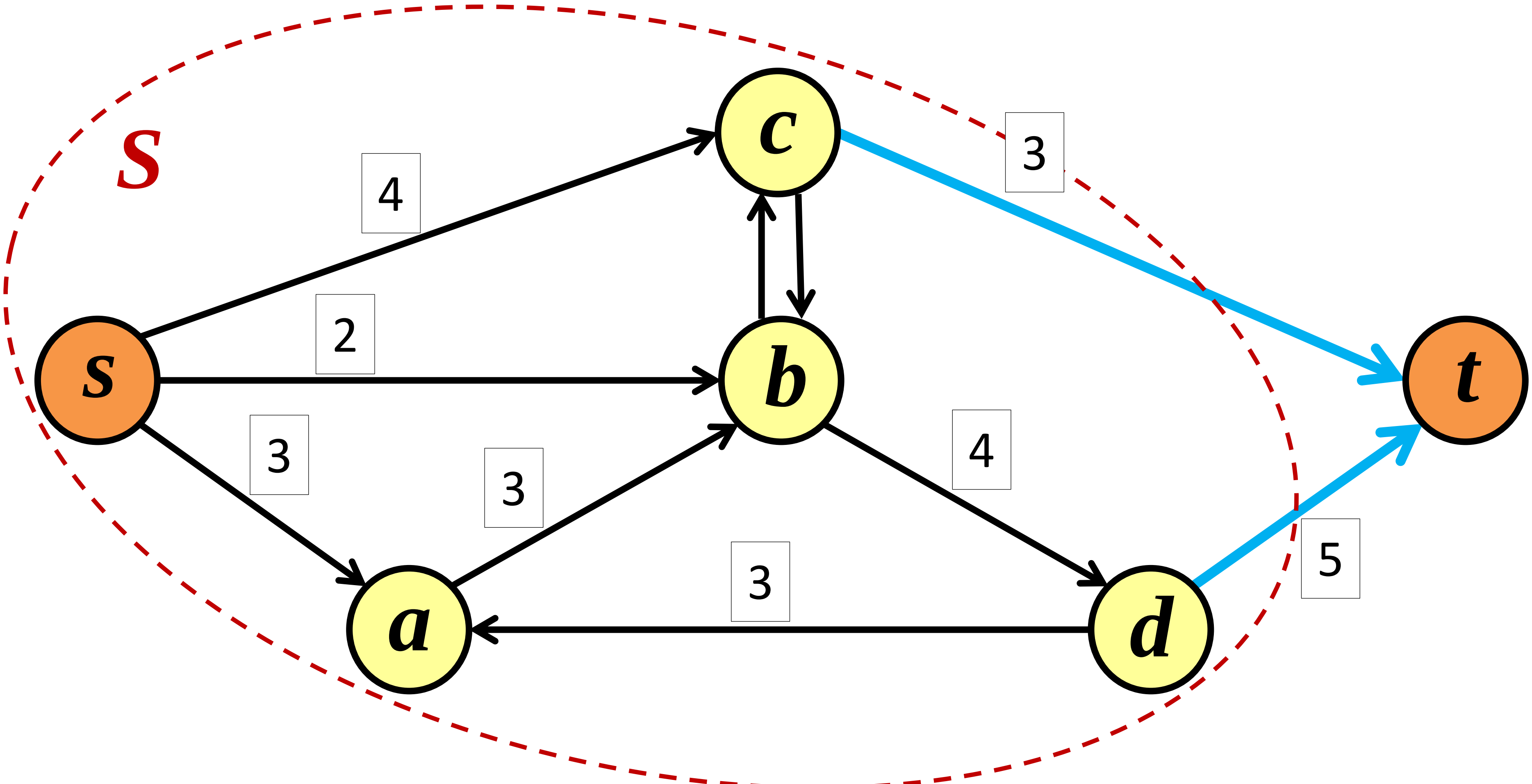
$$c(S, S') = c(s, c) + c(b, c) + c(b, d) = 4 + 1 + 4 = 9$$



Fluxos em redes

$$S = \{s, a, b, c, d\}$$

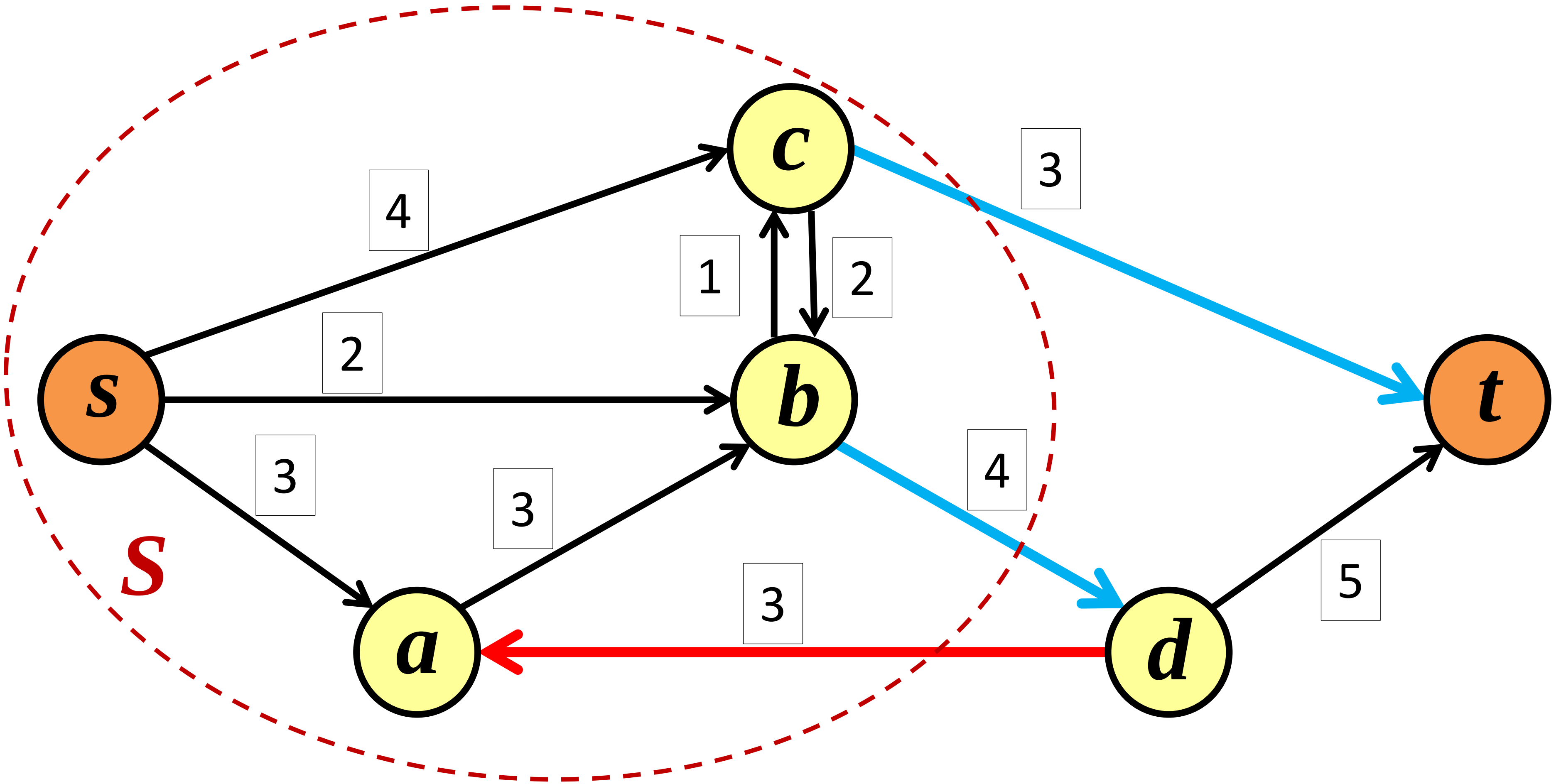
$$c(S, S') = c(c, t) + c(d, t) = 3 + 5 = 8$$



Fluxos em redes

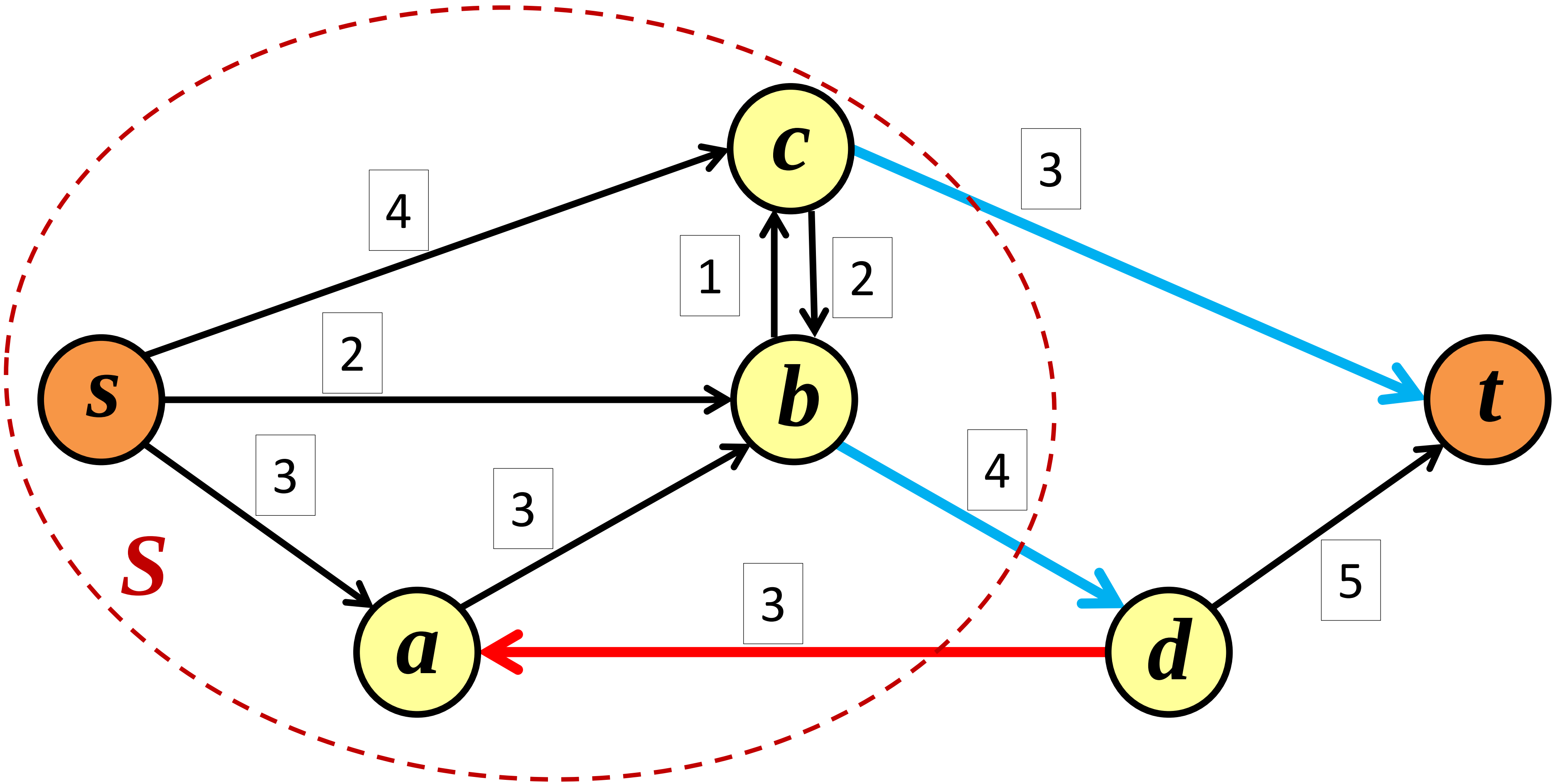
$S = \{\ s, a, b, c\ \}$

$c(S, S') = c(b, d) + c(c, t) = 3 + 4 = 7$



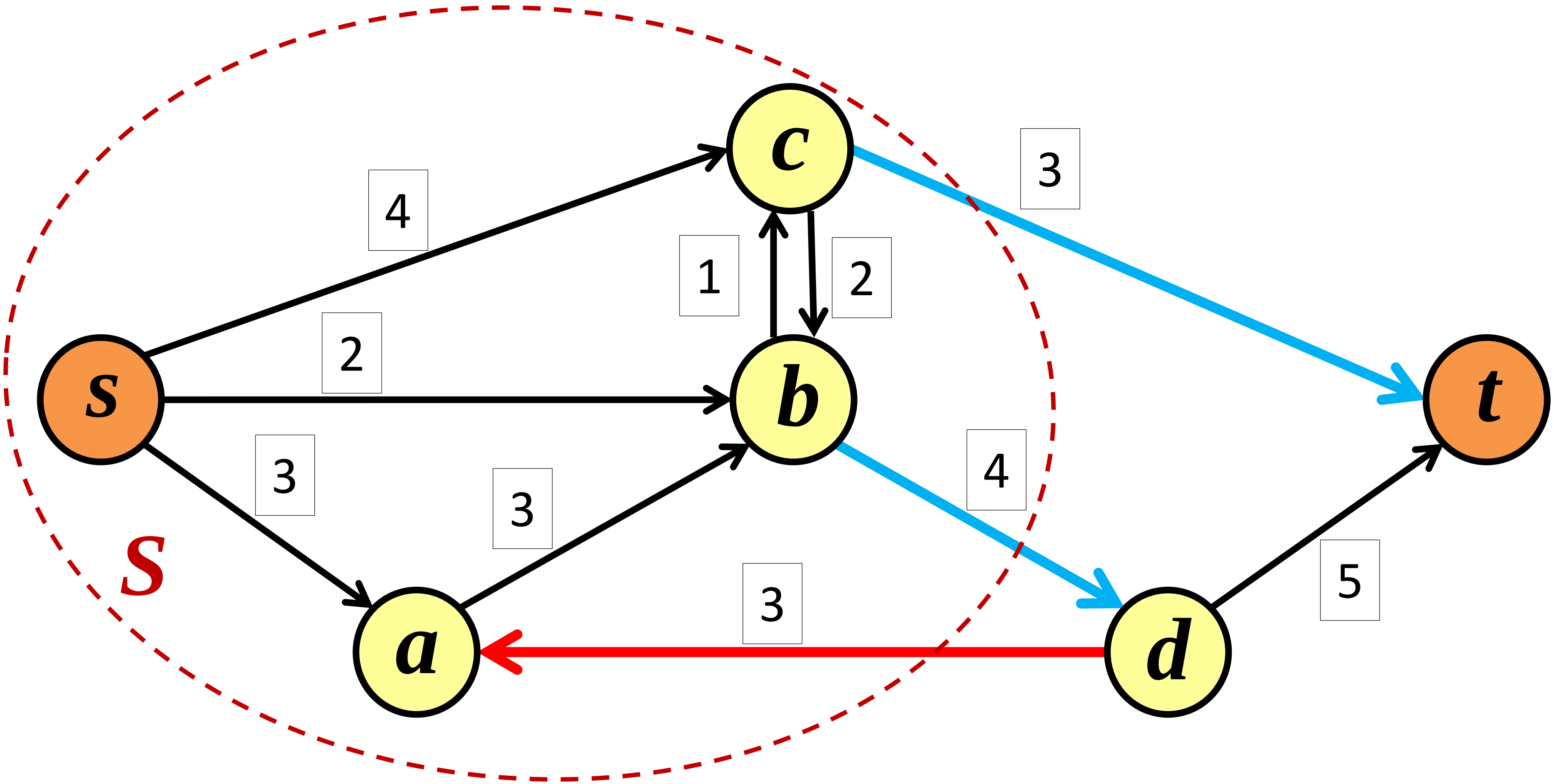
Fluxos em redes

Observe que os cortes têm capacidades cada vez menores!



Fluxos em redes

Problema do Corte Mínimo: Dada uma rede D, encontrar um corte (S, S') com a menor capacidade possível.

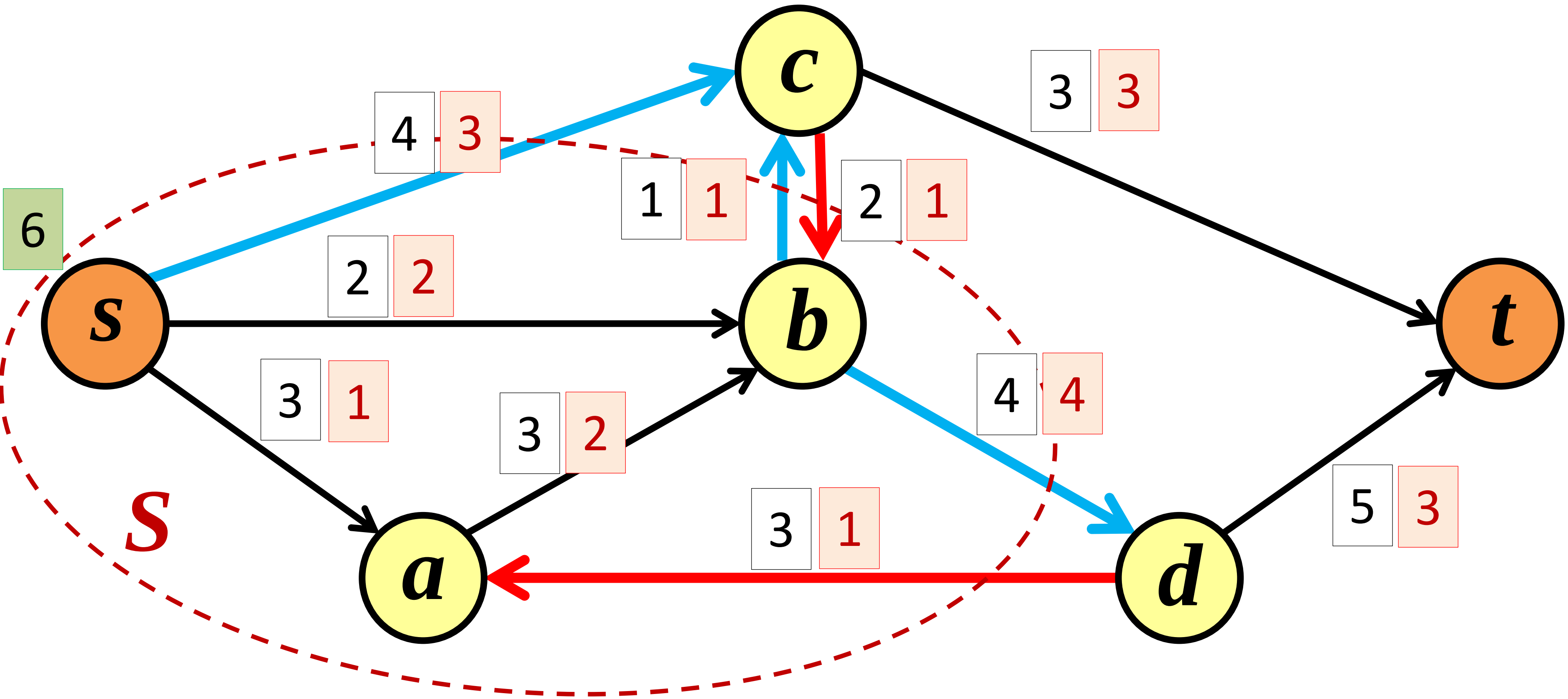


Fluxos em redes

Def.: O ***fluxo f(S, S') em um corte (S, S')*** é a soma dos fluxos nas arestas de $(S, S')^{+}$ menos a soma dos fluxos nas arestas de $(S, S')^{-}$.

Fluxos em redes

Def.: O ***fluxo f(S, S') em um corte (S, S')*** é a soma dos fluxos nas arestas de $(S, S')^+$ menos a soma dos fluxos nas arestas de $(S, S')^-$.

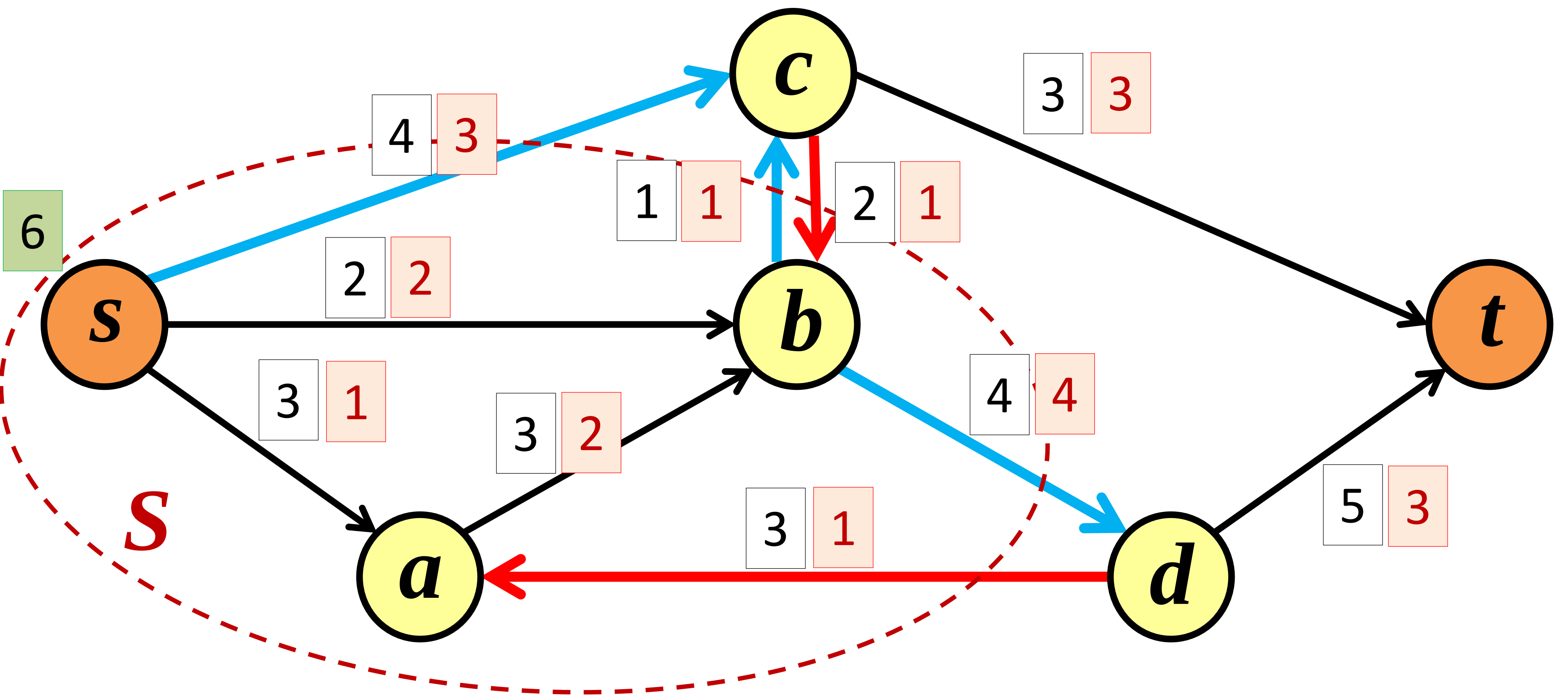


Fluxos em redes

$$S = \{\, s, a, b \,\}$$

$$f(S, S') = f(s, c) + f(b, c) + f(b, d) - f(c, b) - f(d, a) =$$

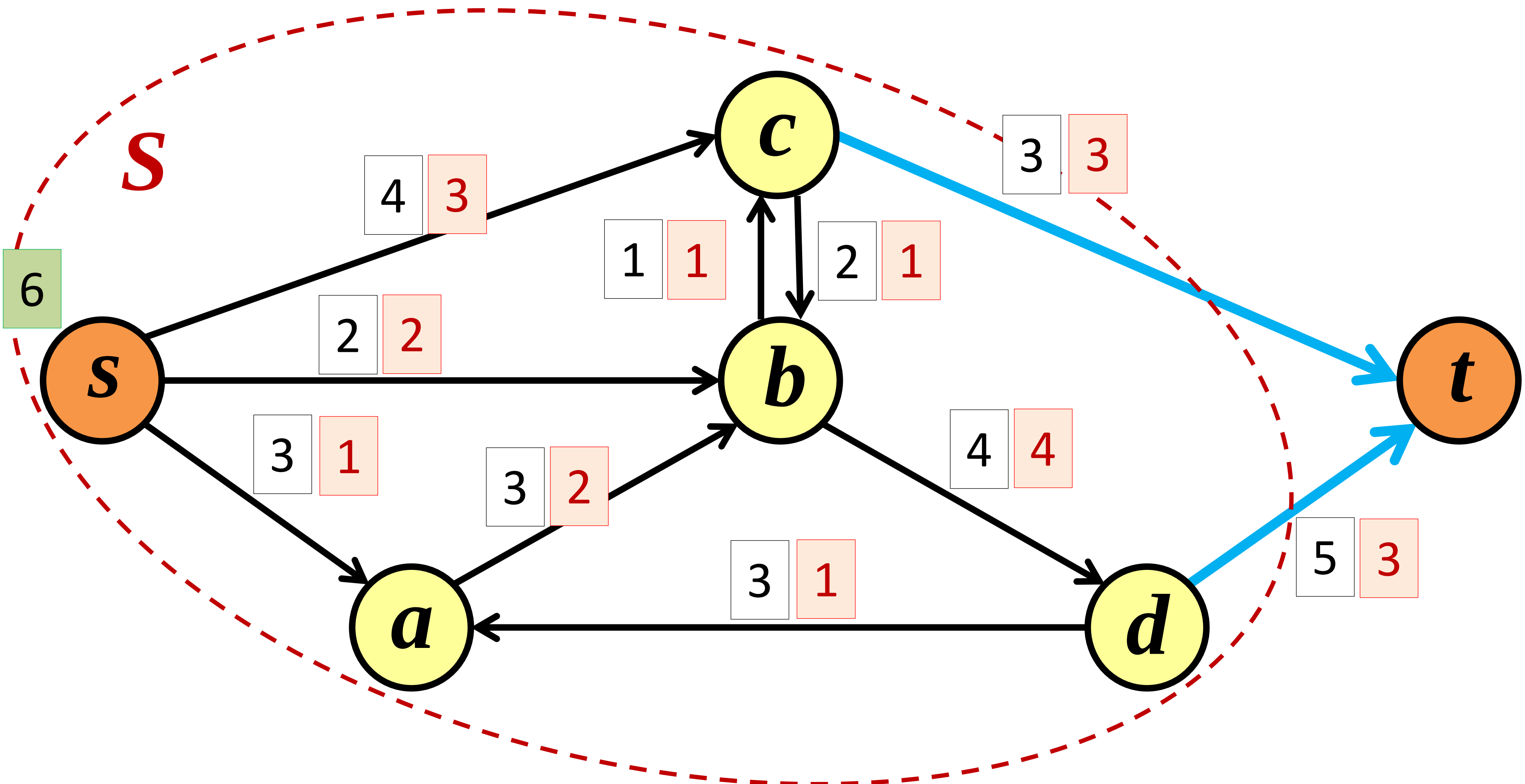
$$3 + 1 + 4 - 1 - 1 = 6$$



Fluxos em redes

$S = \{s, a, b, c, d\}$

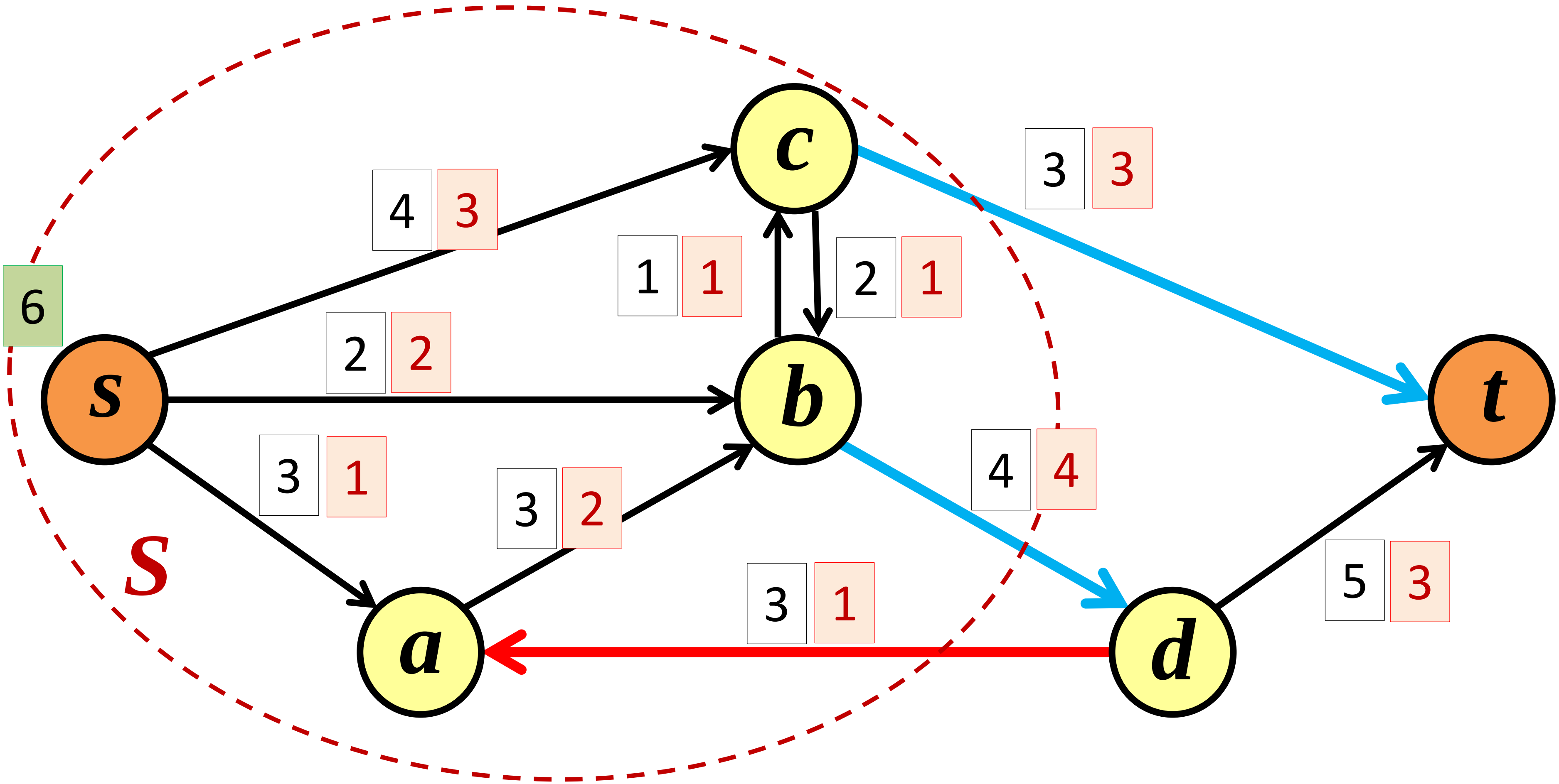
$f(S, S') = f(c, t) + f(d, t) = 3 + 3 = 6$



Fluxos em redes

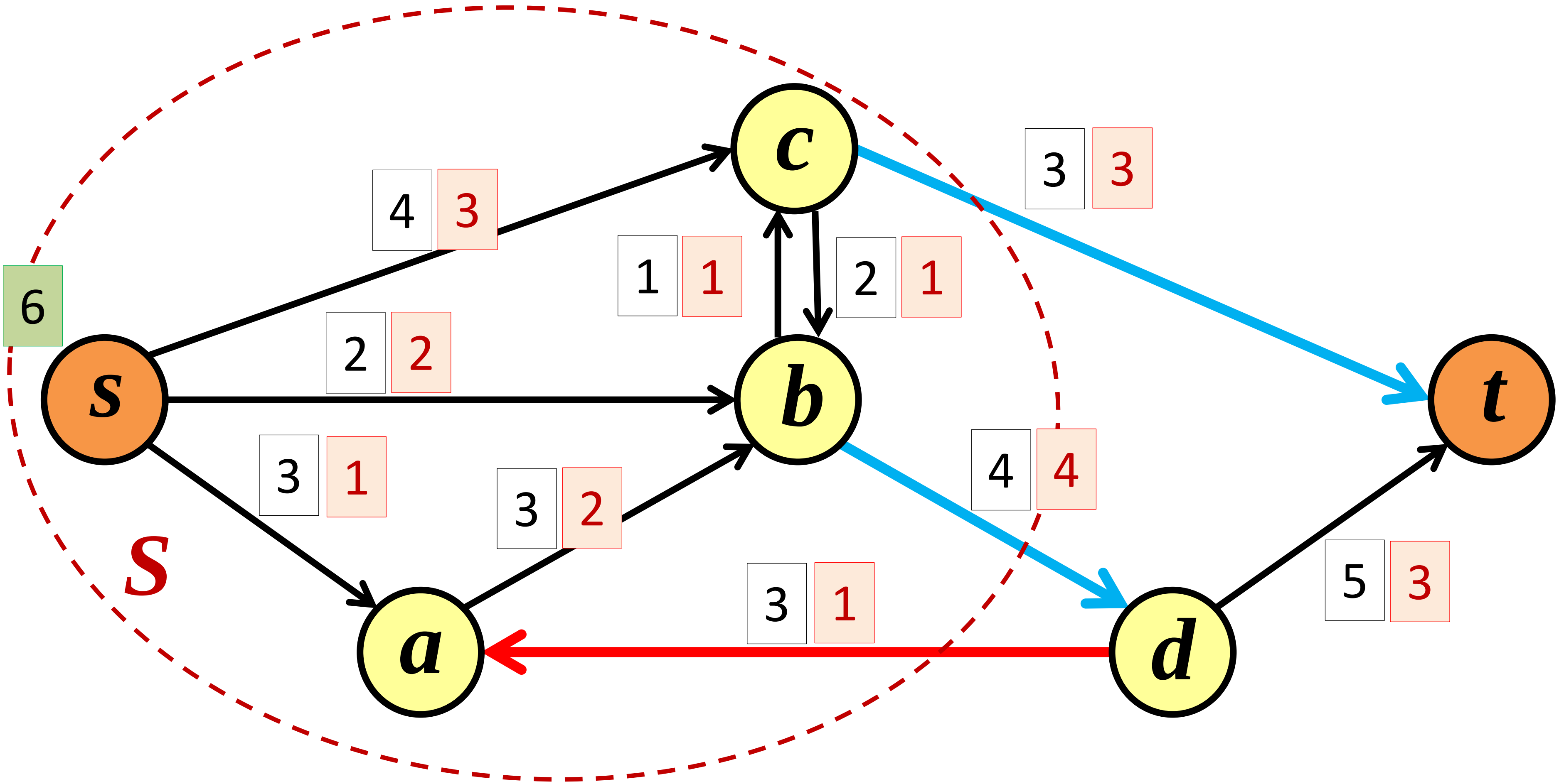
$S = \{ s, a, b, c \}$

$f(S, S') = f(b, d) + f(c, t) - f(d, a) = 3 + 4 - 1 = 6$



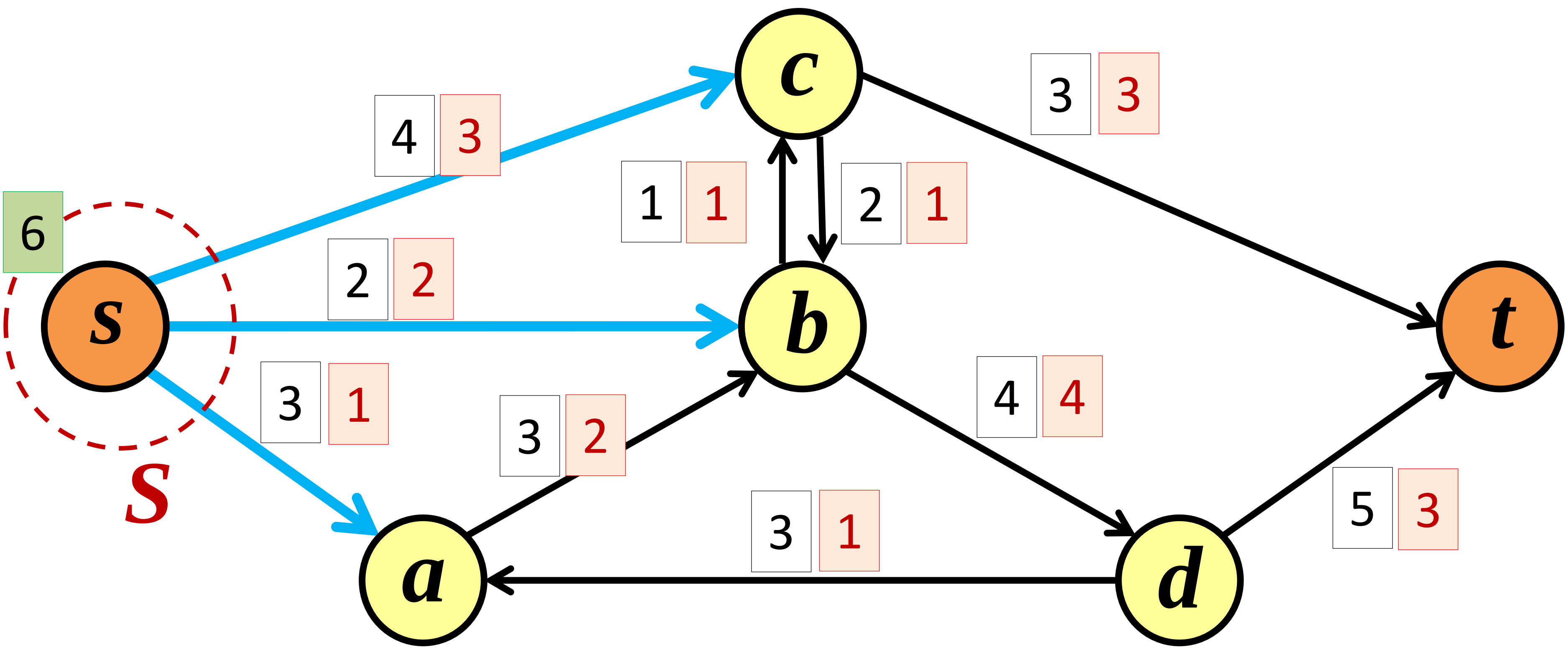
Fluxos em redes

Observe que o valor do fluxo em qualquer corte é sempre o mesmo, e é igual ao fluxo f(D) na rede!



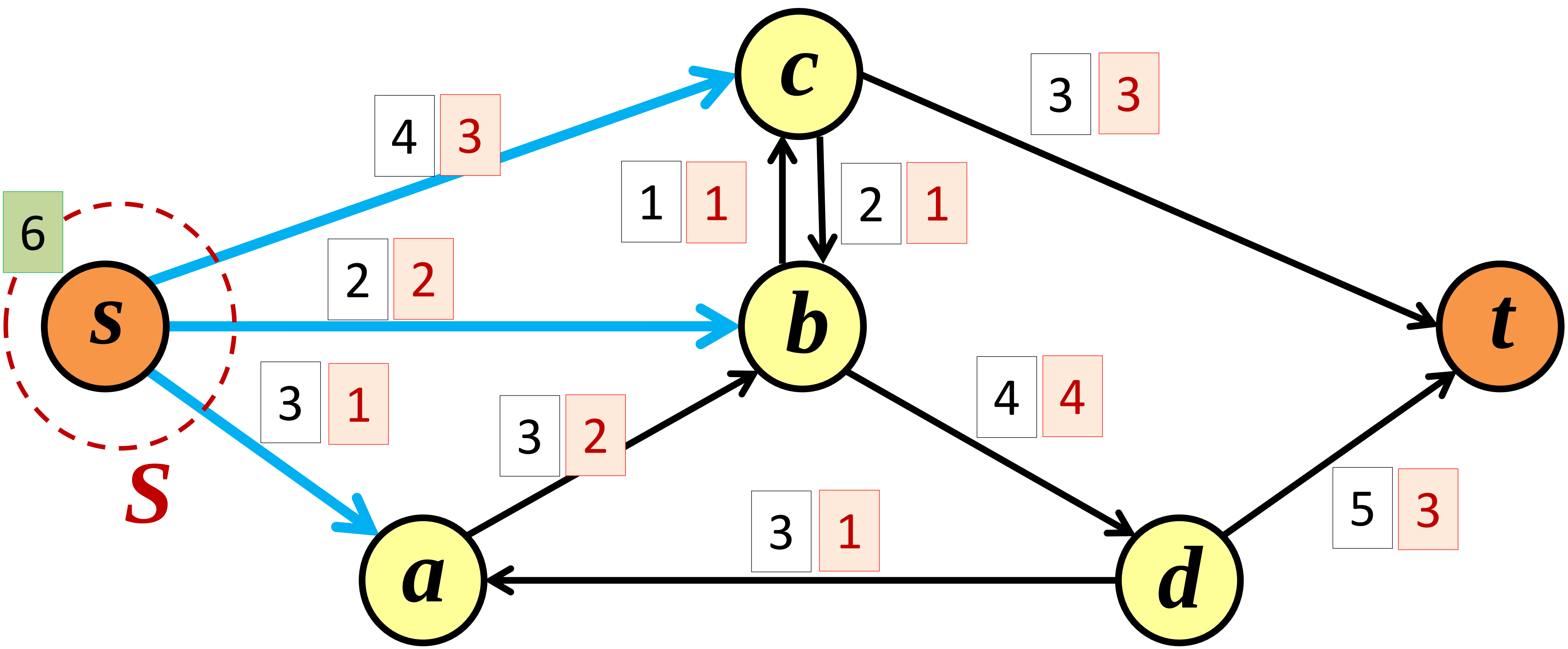
Fluxos em redes

f(D) = f(S, S') para S = {s}.



Fluxos em redes

Teorema: Seja *f* um fluxo em uma rede *D* e (*S*, *S*’) um corte qualquer em *D*. Então *f*(*S*, *S*’) = *f* (*D*).

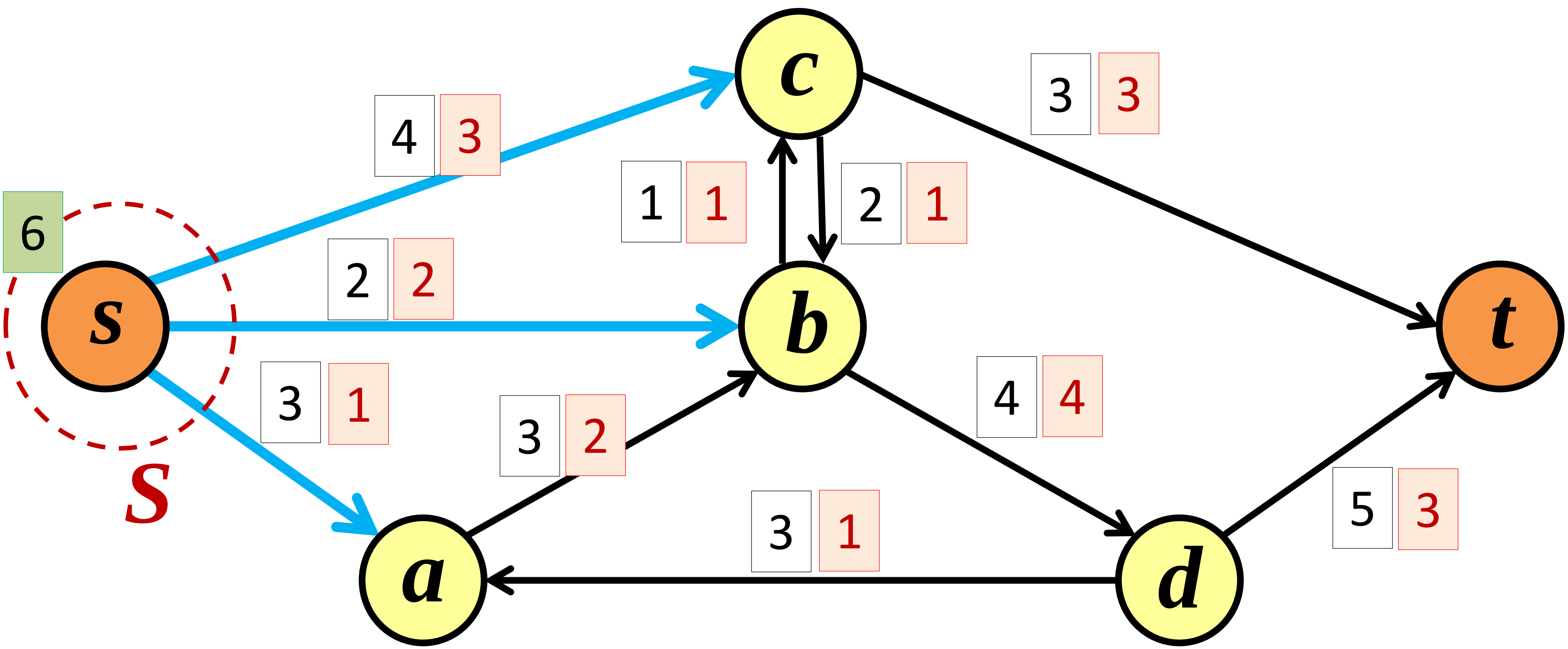


Fluxos em redes

Teorema: Seja f um fluxo qualquer em uma rede D, e (S, S') um corte qualquer em D. Então $f(D) \leq c(S, S')$.

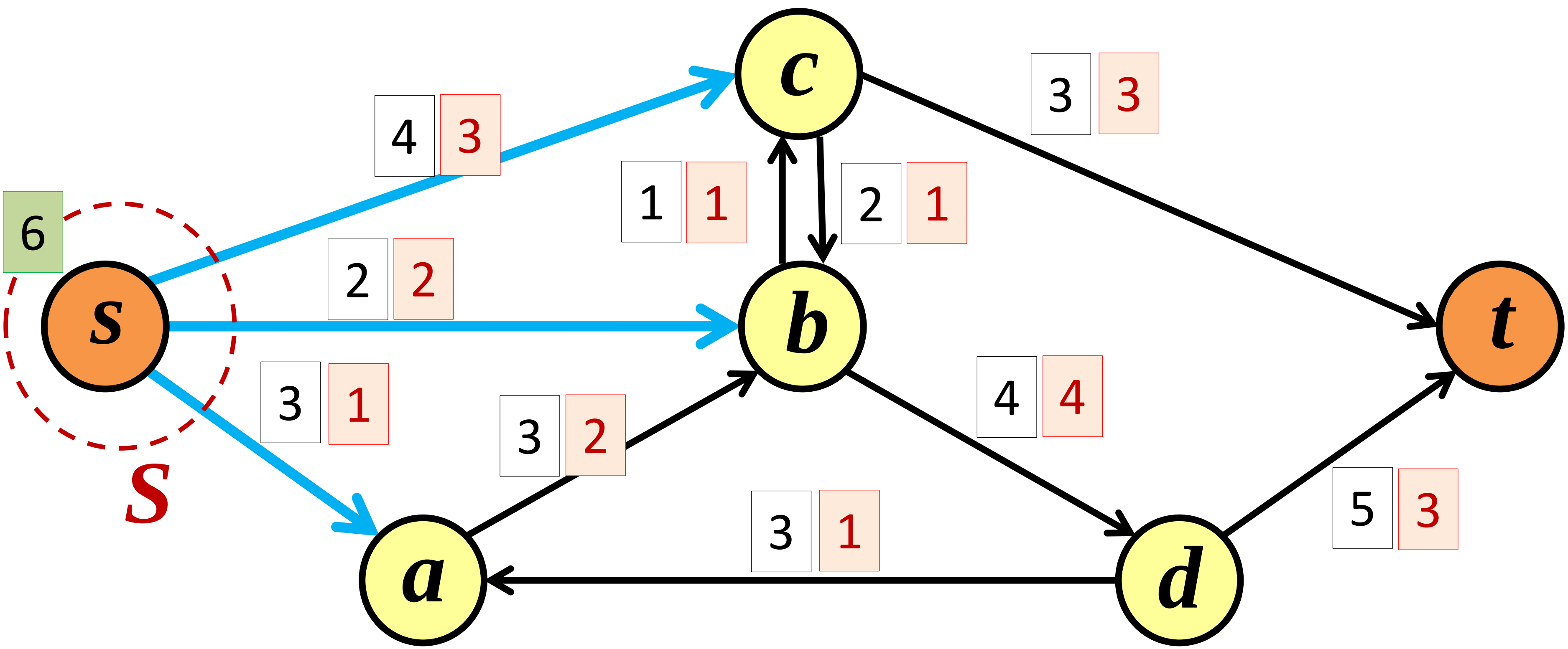
Fluxos em redes

Teorema: Seja f um fluxo qualquer em uma rede D, e (S, S') um corte qualquer em D. Então $f(D) \leq c(S, S')$.



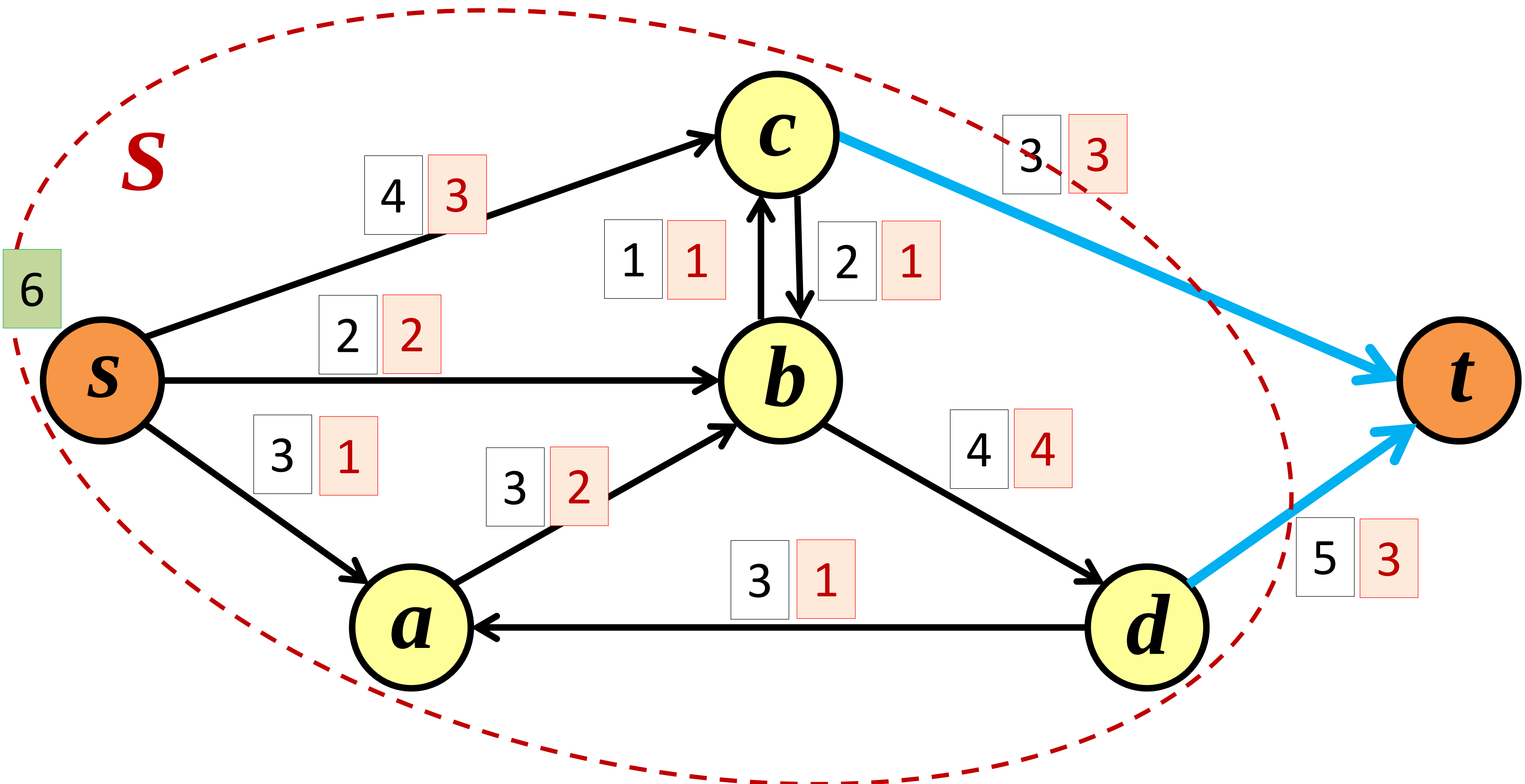
Fluxos em redes

$f(D) = \mathbf{6} \leq c(S, S') = \mathbf{9}$



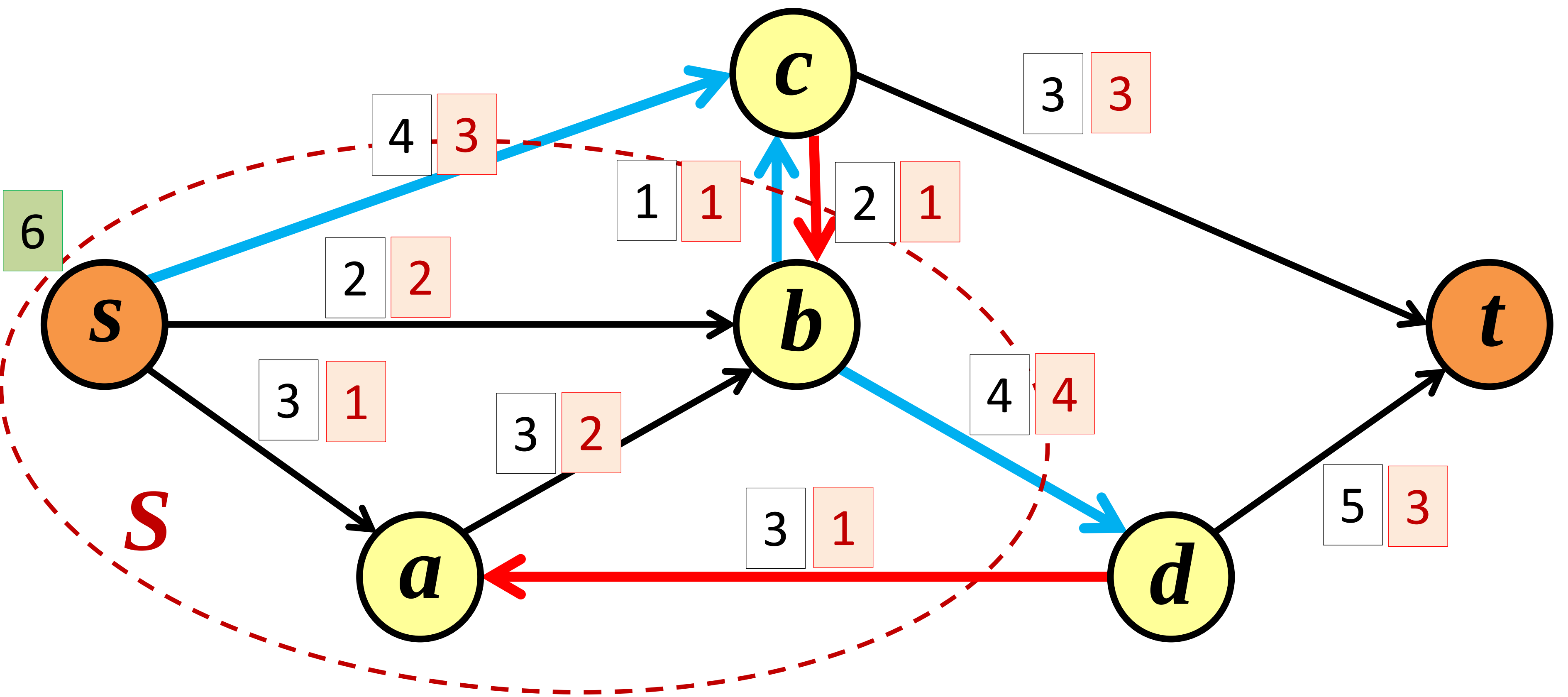
Fluxos em redes

$$f\,(D) = \mathbf{6} \leq\ c(S, S'\,) = \mathbf{8}$$



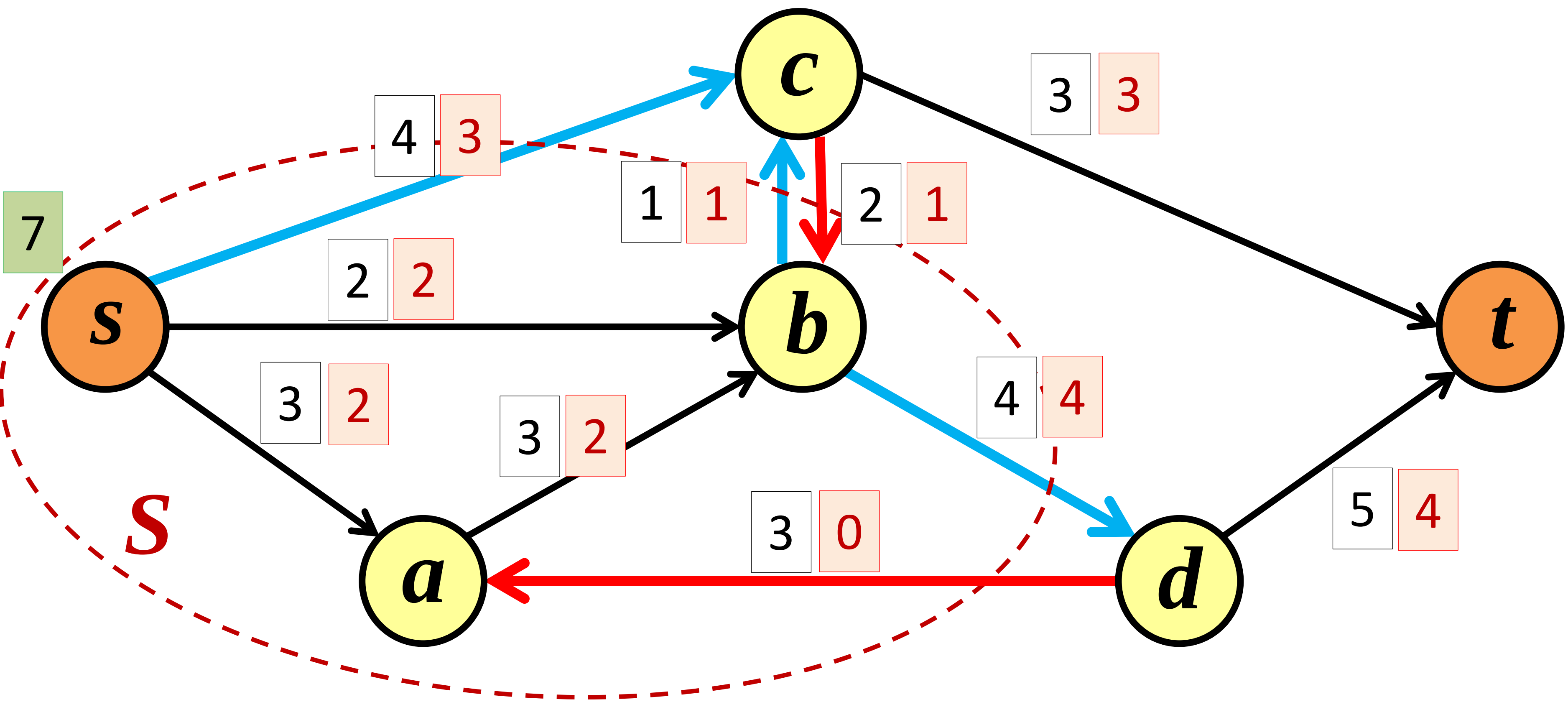
Fluxos em redes

$f\,(D) = \mathbf{6} \leq \; c(S, S') = \mathbf{9}$



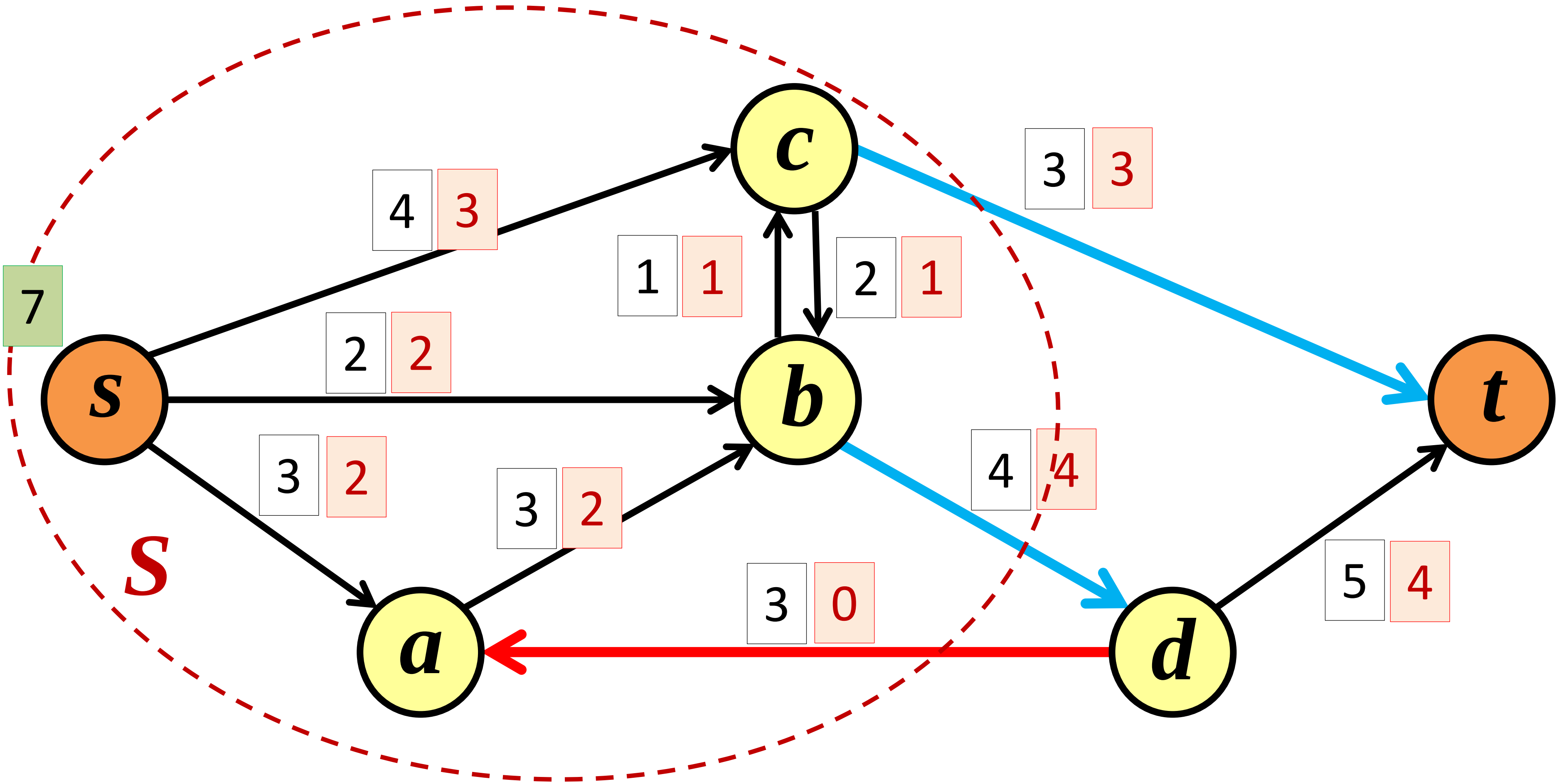
Fluxos em redes

$f\,(D) = \mathbf{7} \leq c(S, S') = \mathbf{9}$



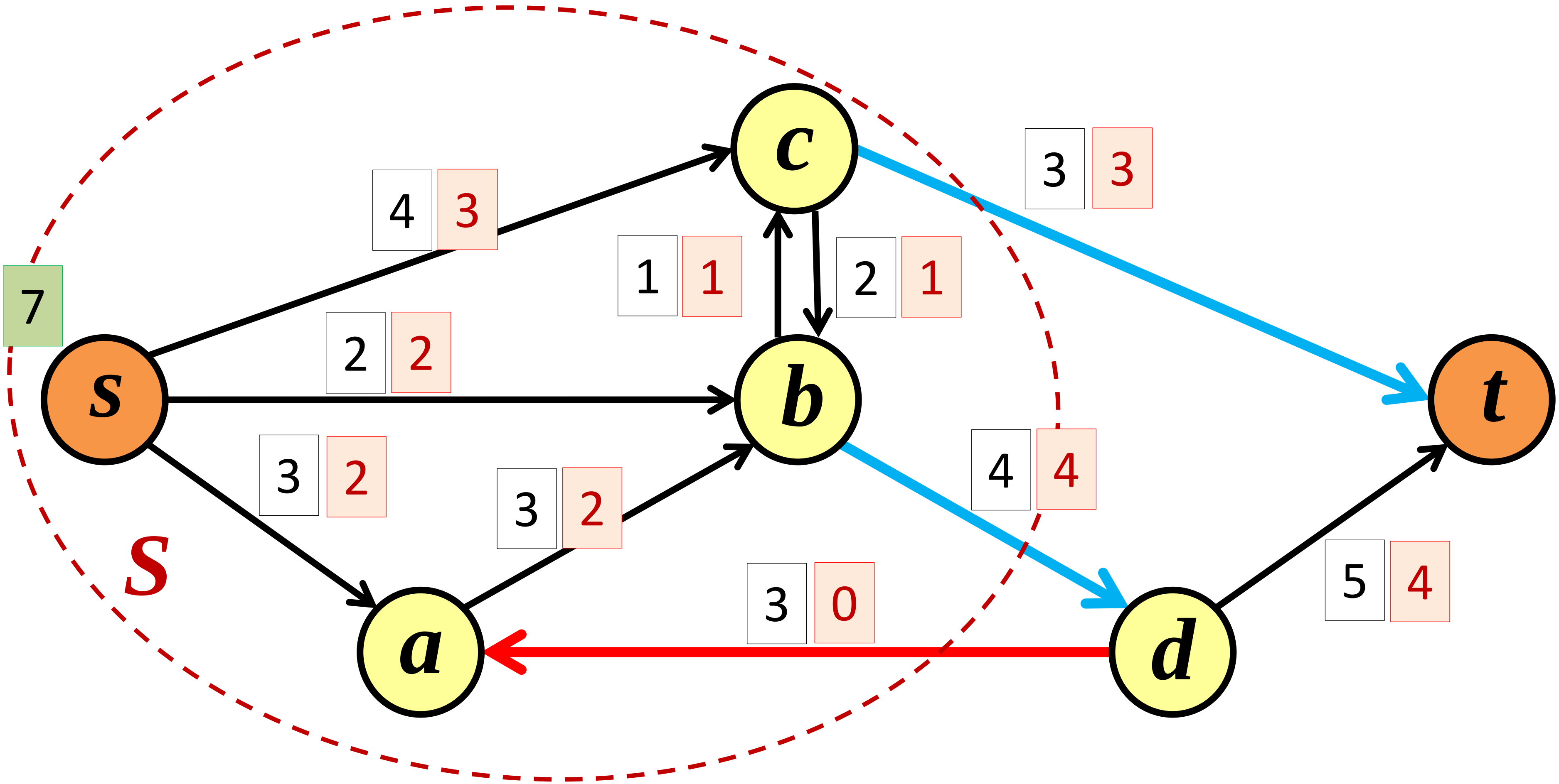
Fluxos em redes

$$f\,(D) = \mathbf{7} \leq c(S, S') = \mathbf{7}$$



Fluxos em redes

Teorema: Seja *f* um fluxo **máximo** em uma rede *D*, e (*S*, *S*’) um corte **mínimo** em *D*. Então *f* (*D*) = *c*(*S*, *S*’).



Fluxos em redes

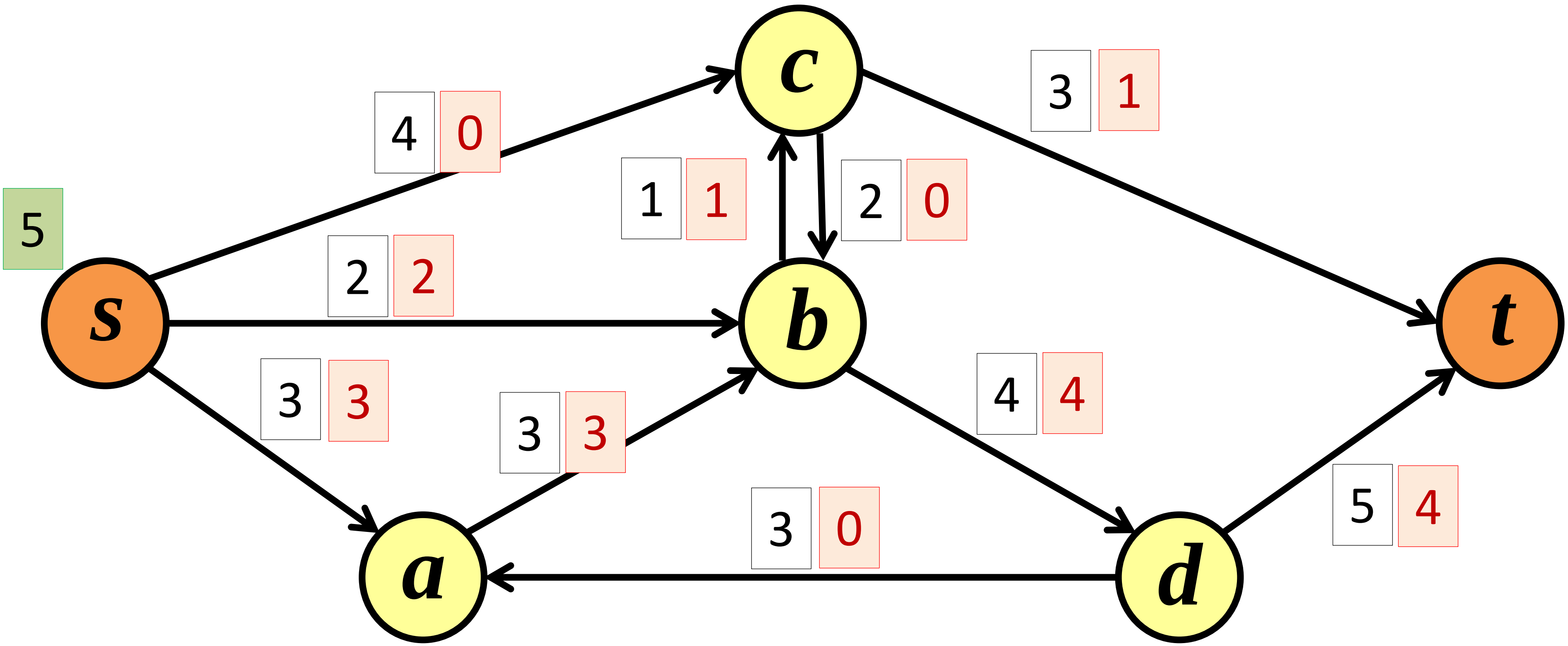
Def.: Dado um fluxo f numa rede D,

- uma aresta $\boldsymbol{e}$ é ***direta*** se $\boldsymbol{c(e) - f(e) > 0}$
- uma aresta $\boldsymbol{e}$ é ***contrária*** se $\boldsymbol{f(e) > 0}$

Fluxos em redes

Def.: Dado um fluxo f numa rede D,

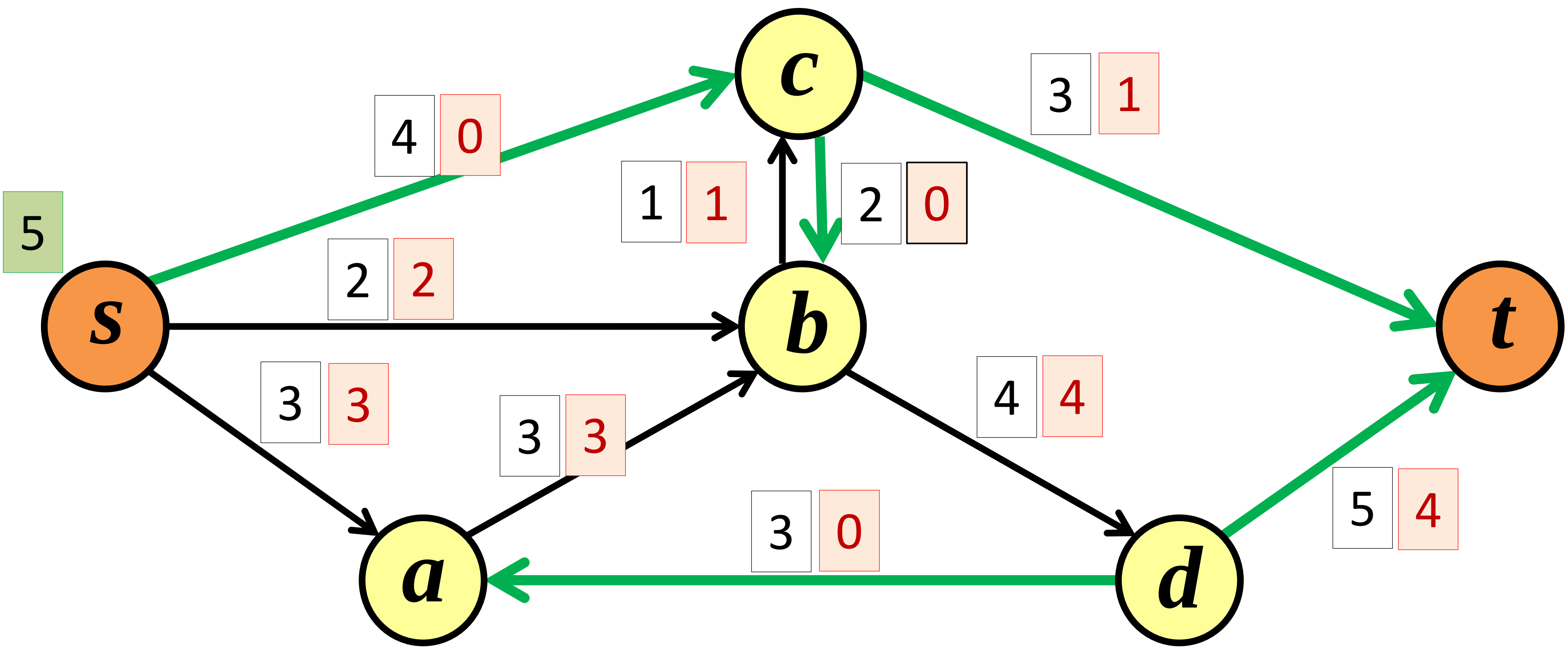
- uma aresta $\boldsymbol{e}$ é ***direta*** se $\boldsymbol{c(e) - f(e) > 0}$
- uma aresta $\boldsymbol{e}$ é ***contrária*** se $\boldsymbol{f(e) > 0}$



Fluxos em redes

Def.: Dado um fluxo f numa rede D,

- uma aresta ***e*** é ***direta*** se $\mathbf{c(e) - f(e) > 0}$
- uma aresta ***e*** é ***contrária*** se $\mathbf{f(e) > 0}$

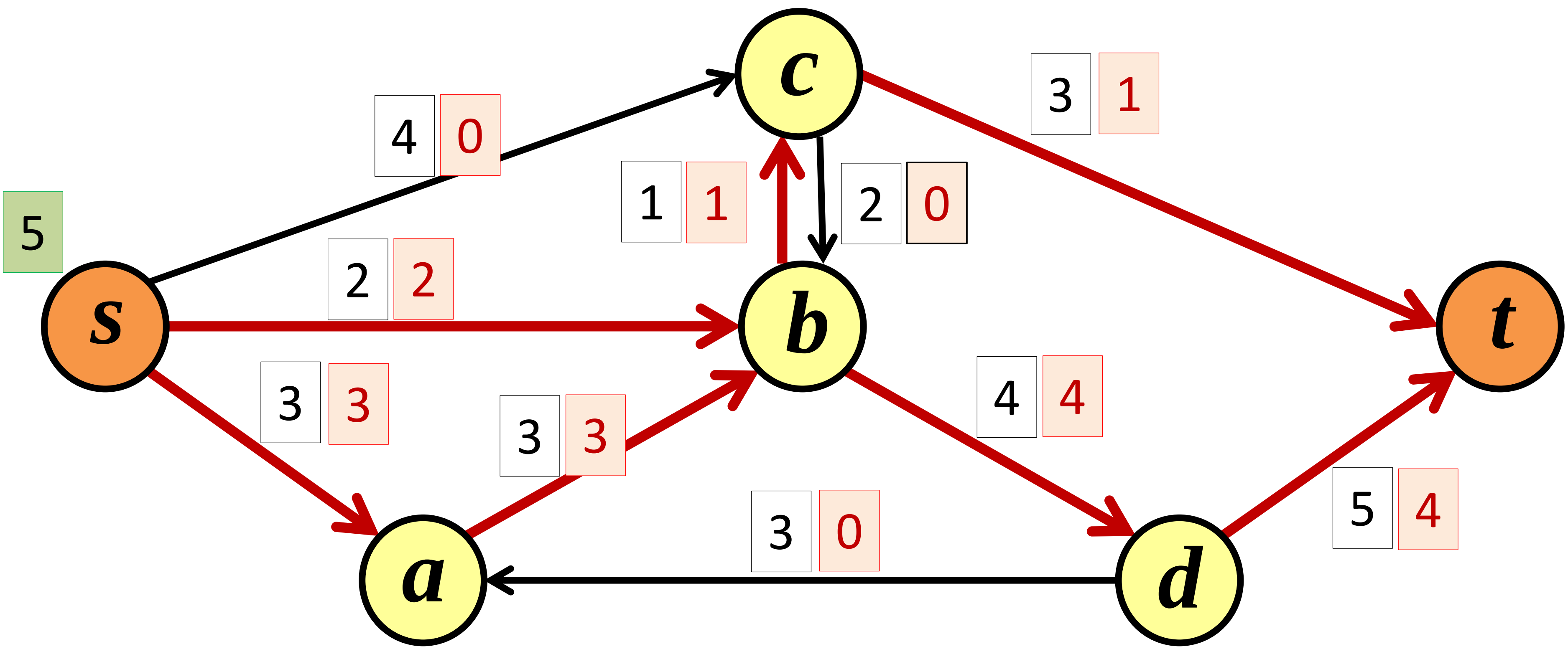


arestas diretas

Fluxos em redes

Def.: Dado um fluxo f numa rede D,

- uma aresta $\boldsymbol{e}$ é ***direta*** se $c(e) - f(e) > 0$
- uma aresta $\boldsymbol{e}$ é ***contrária*** se $f(e) > 0$

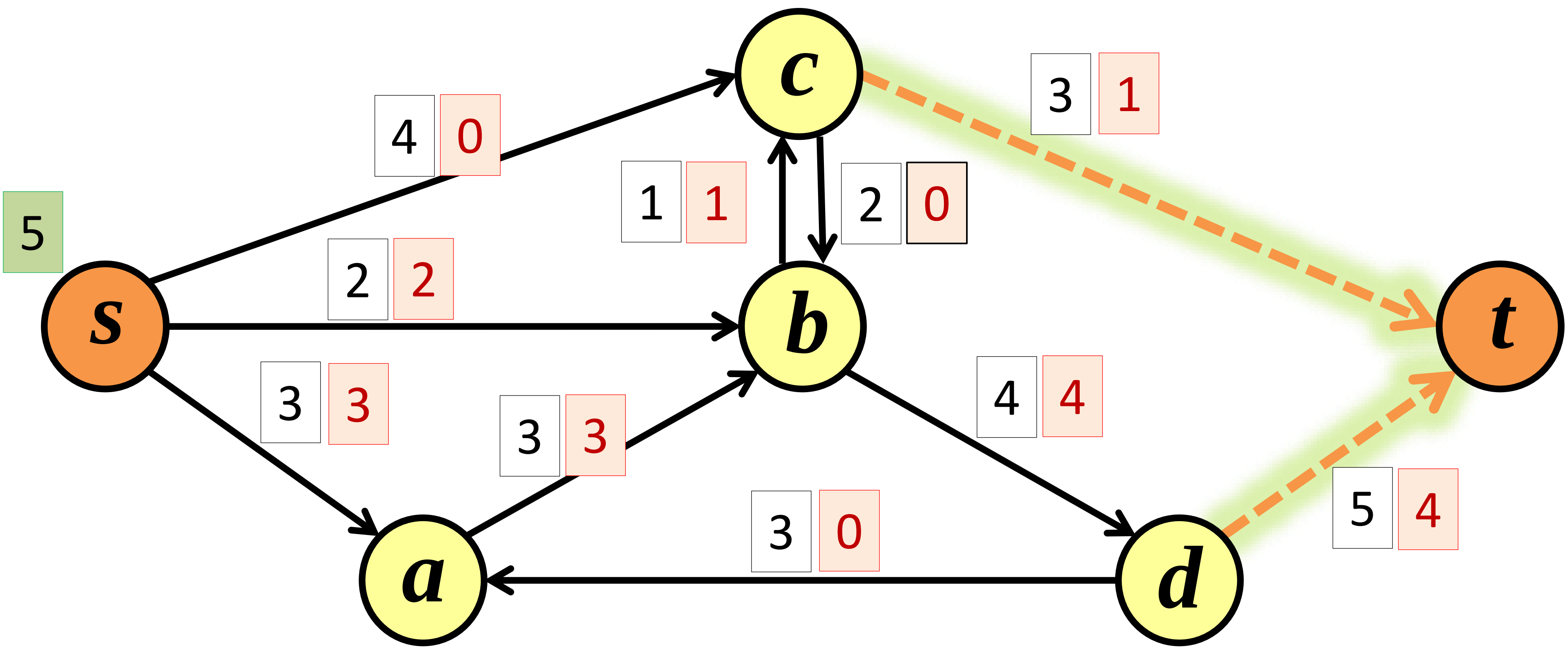


arestas contrárias

Fluxos em redes

Def.: Dado um fluxo f numa rede D,

- uma aresta $\boldsymbol{e}$ é ***direta*** se $\boldsymbol{c(e) - f(e) > 0}$
- uma aresta $\boldsymbol{e}$ é ***contrária*** se $\boldsymbol{f(e) > 0}$



arestas simultaneamente diretas/contrárias

Fluxos em redes

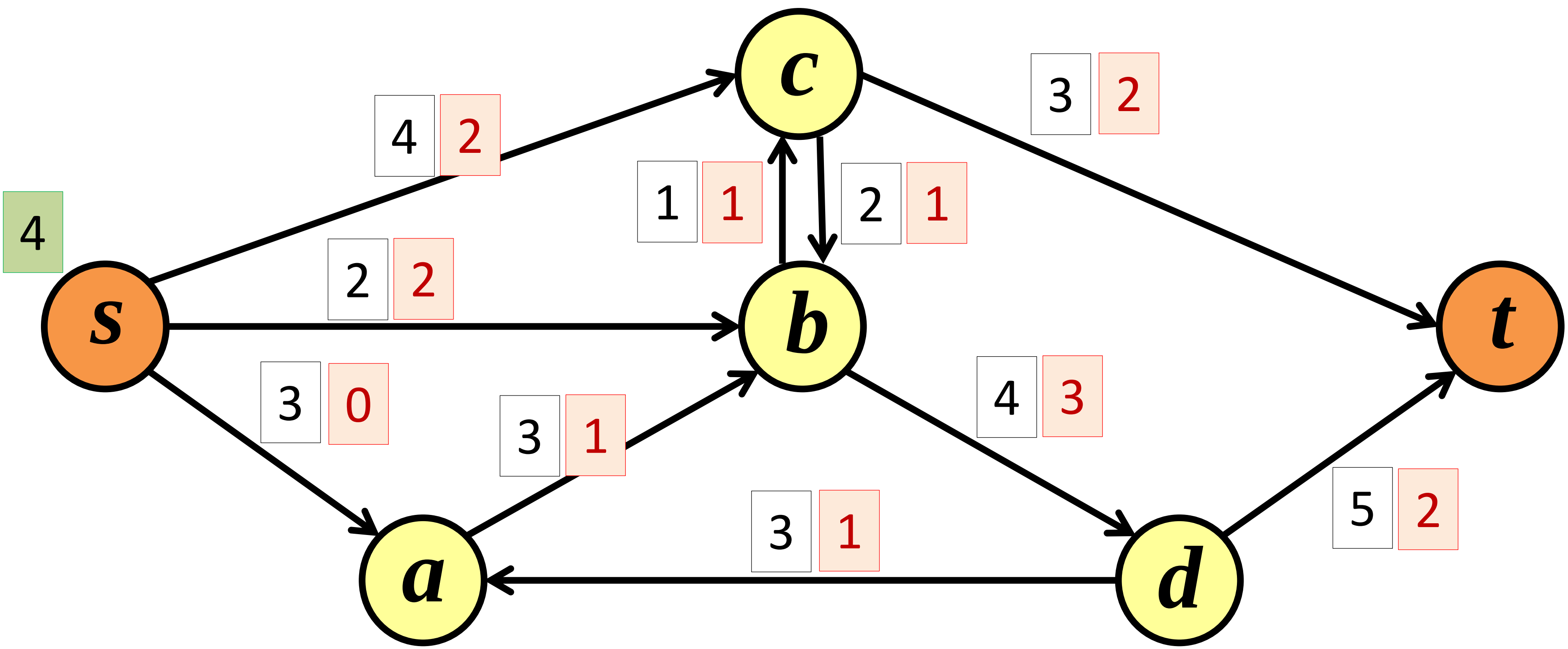
Def.: Dados D e f, define-se a ***rede residual D' = (V, E')***:

- **(*x*,*y*)** é ***direta*** $\longrightarrow$ **(*x*,*y*)** $\in E'$ c/ capacidade $c'(x,y) = c(x,y) - f(x,y)$
- **(*x*,*y*)** é ***contrária*** $\longrightarrow$ **(*y*,*x*)** $\in E'$ c/ capacidade $c'(y,x) = f(x,y)$

Fluxos em redes

Def.: Dados D e f, define-se a ***rede residual D' = (V, E')***:

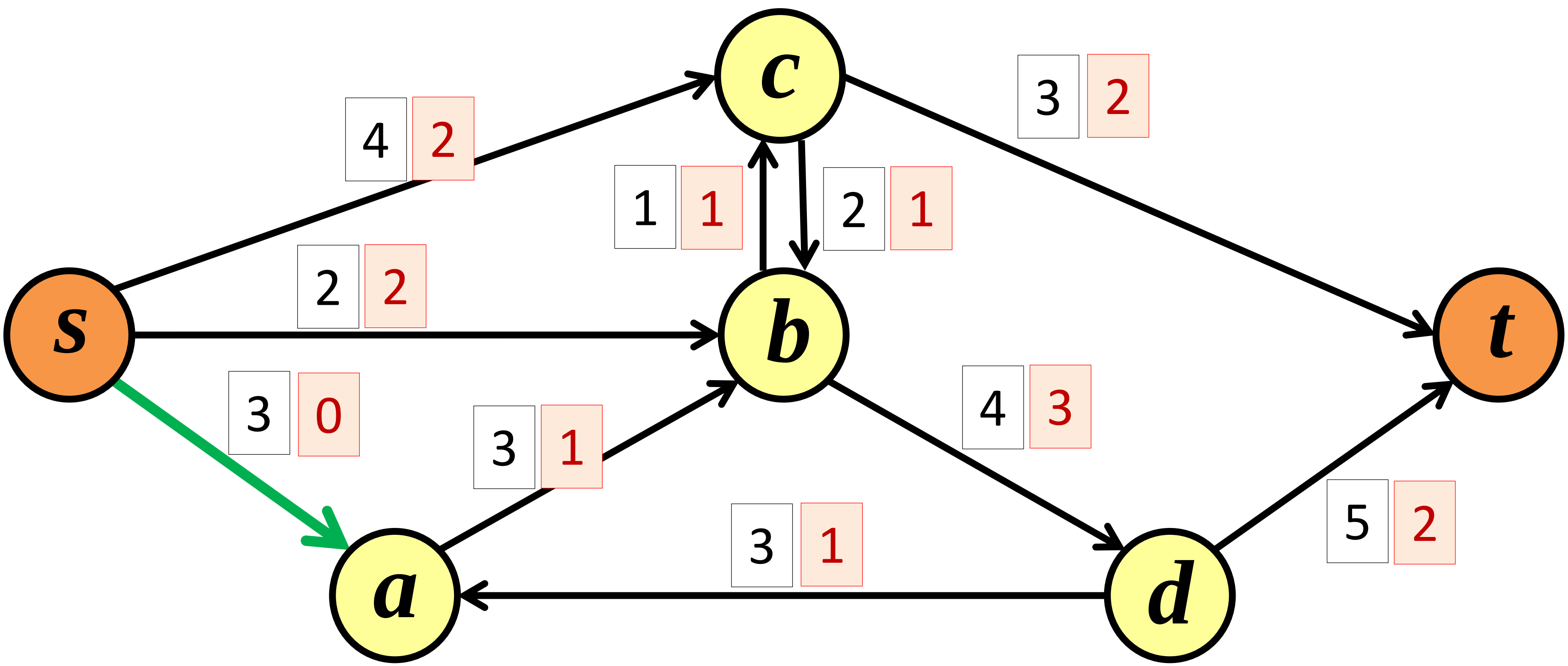
- **(x,y)** é ***direta*** $\rightarrow$ **(x,y)** $\in E'$ c/ capacidade $c'(x,y) = c(x,y) - f(x,y)$
- **(x,y)** é ***contrária*** $\rightarrow$ **(y,x)** $\in E'$ c/ capacidade $c'(y,x) = f(x,y)$



Fluxos em redes

Def.: Dados D e f, define-se a ***rede residual D' = (V, E')***:

- **(*x*,*y*)** é ***direta*** $\longrightarrow$ **(*x*,*y*)** $\in E'$ c/ capacidade $c'(x,y) = c(x,y) - f(x,y)$
- **(*x*,*y*)** é ***contrária*** $\longrightarrow$ **(*y*,*x*)** $\in E'$ c/ capacidade $c'(y,x) = f(x,y)$

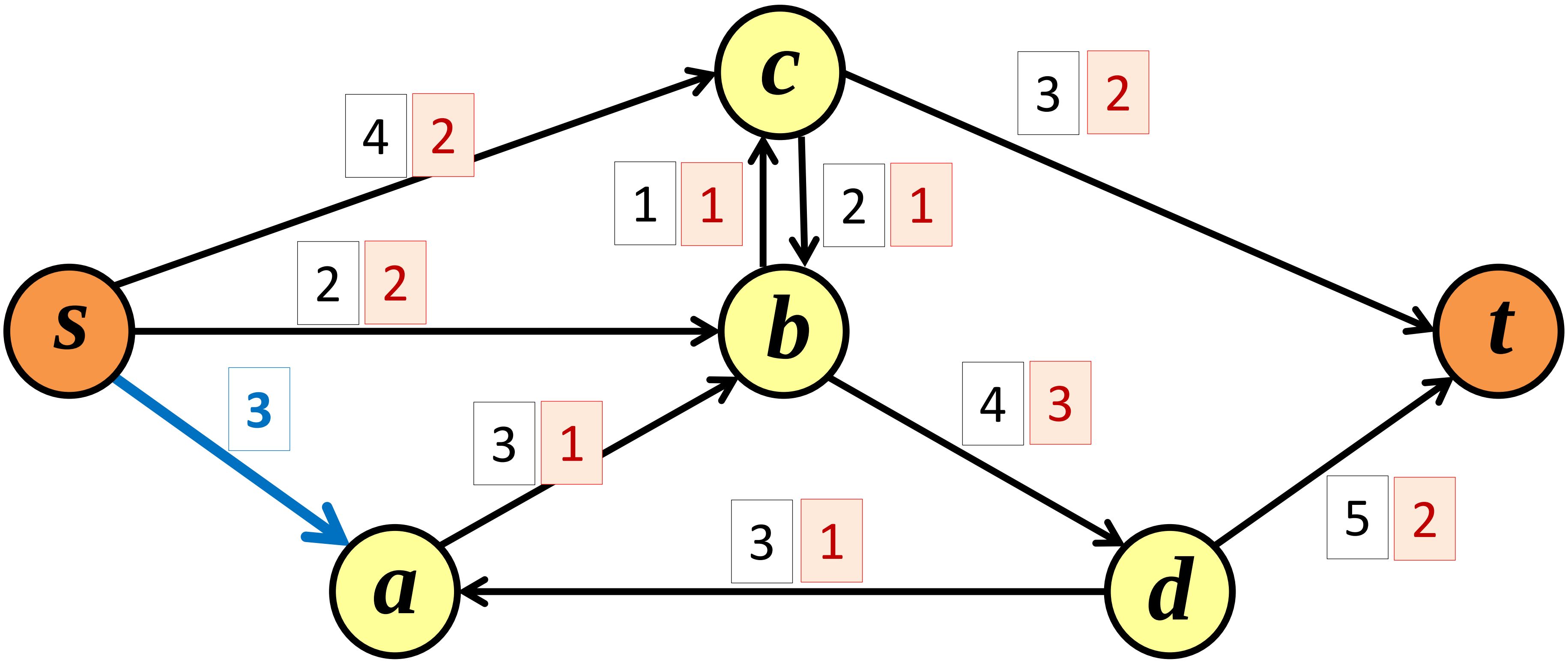


aresta direta de D

Fluxos em redes

Def.: Dados D e f, define-se a ***rede residual D' = (V, E')***:

- **(*x*,*y*)** é ***direta*** $\longrightarrow$ **(*x*,*y*)** $\in E'$ c/ capacidade $c'(x,y) = c(x,y) - f(x,y)$
- **(*x*,*y*)** é ***contrária*** $\longrightarrow$ **(*y*,*x*)** $\in E'$ c/ capacidade $c'(y,x) = f(x,y)$

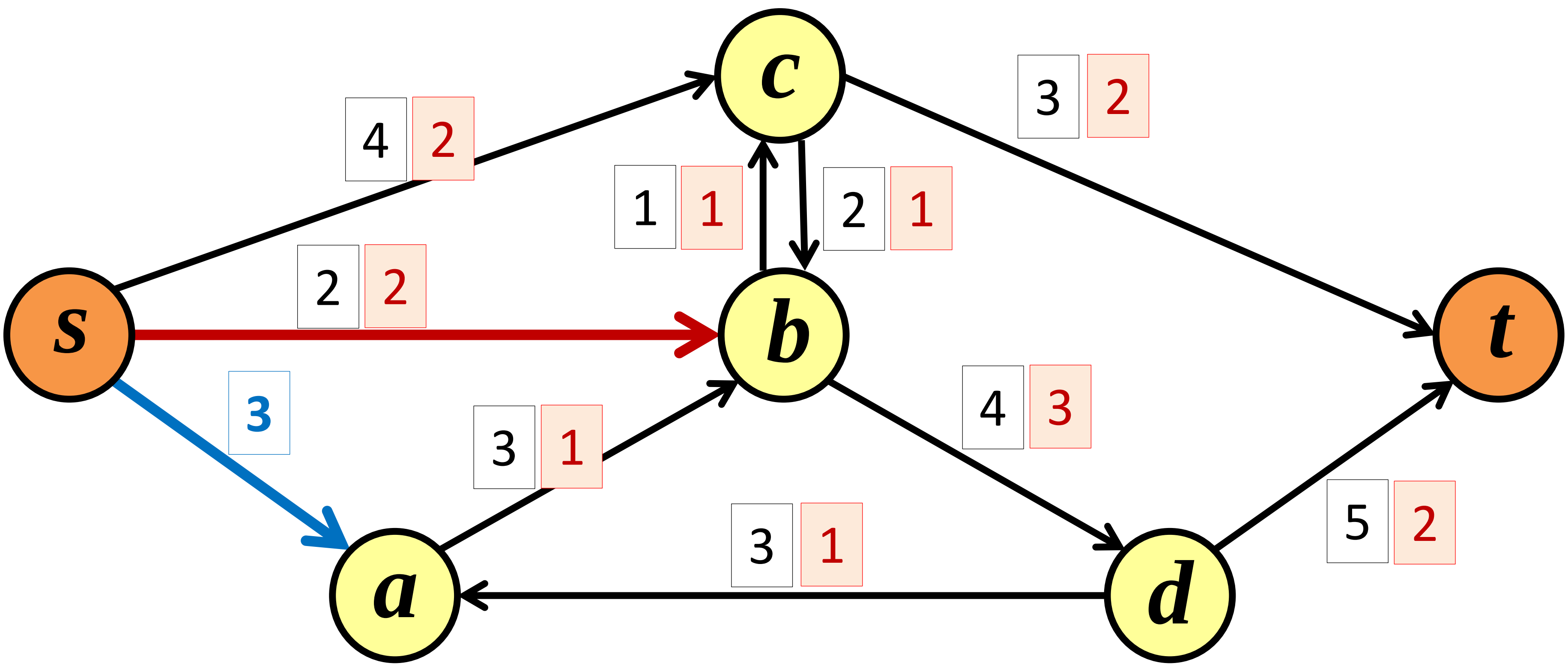


aresta de D'

Fluxos em redes

Def.: Dados D e f, define-se a ***rede residual $D' = (V, E')$***:

- (x,y) é ***direta*** $\rightarrow$ $(x,y) \in E'$ c/ capacidade $c'(x,y) = c(x,y) - f(x,y)$
- (x,y) é ***contrária*** $\rightarrow$ $(y,x) \in E'$ c/ capacidade $c'(y,x) = f(x,y)$

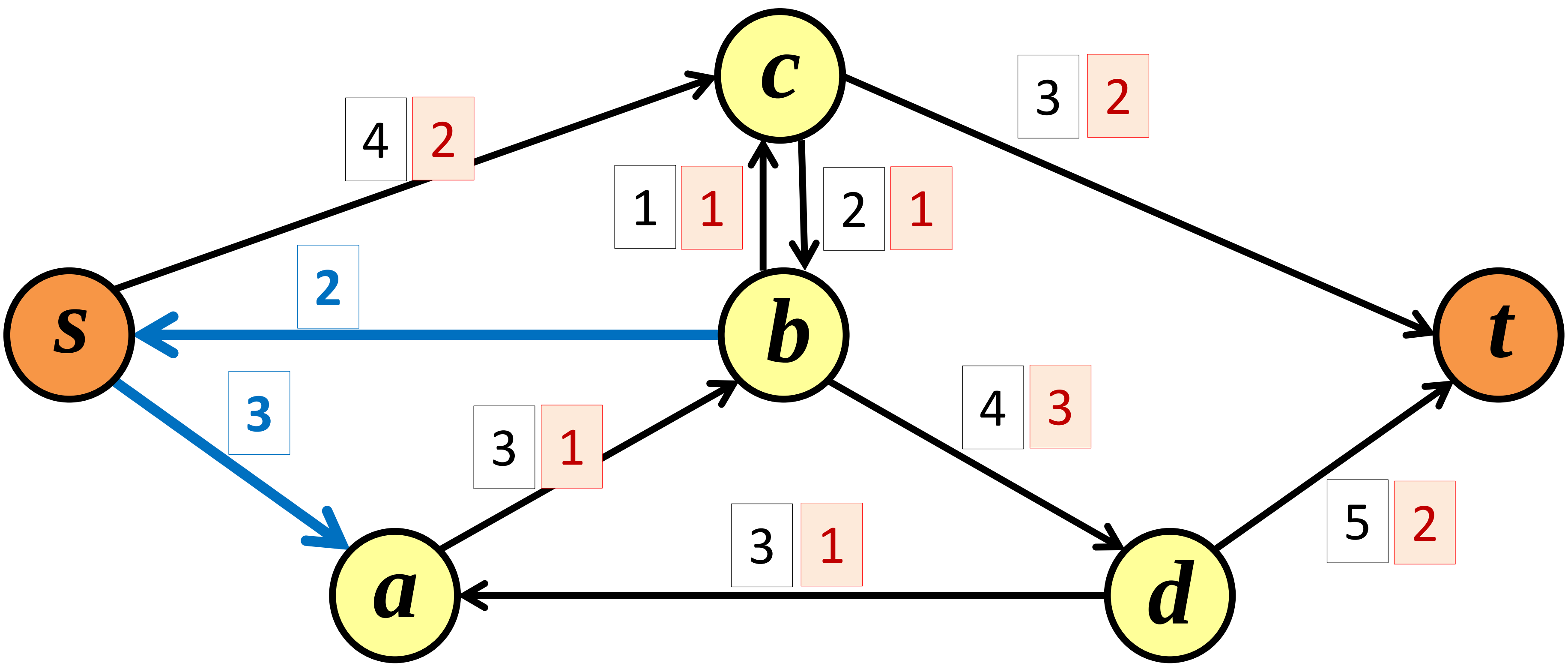


aresta contrária de D

Fluxos em redes

Def.: Dados D e f, define-se a ***rede residual*** $\boldsymbol{D' = (V, E')}$:

- (x,y) é ***direta*** $\longrightarrow$ $(x,y) \in E'$ c/ capacidade $c'(x,y) = c(x,y) - f(x,y)$
- (x,y) é ***contrária*** $\longrightarrow$ $(y,x) \in E'$ c/ capacidade $c'(y,x) = f(x,y)$

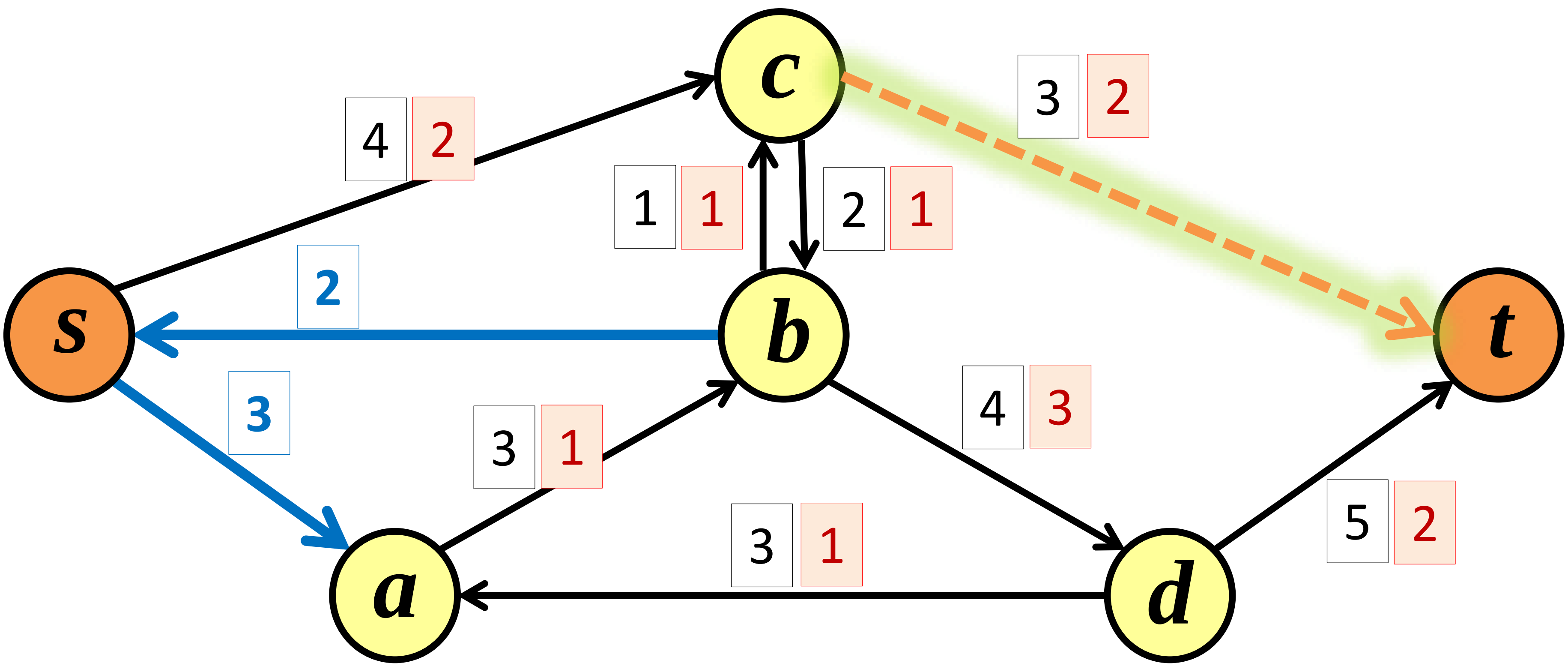


aresta de D'

Fluxos em redes

Def.: Dados D e f, define-se a ***rede residual D' = (V, E')***:

- **(x,y)** é ***direta*** ⟶ **(x,y)** ∈ E' c/ capacidade $c'(x,y) = c(x,y) - f(x,y)$
- **(x,y)** é ***contrária*** ⟶ **(y,x)** ∈ E' c/ capacidade $c'(y,x) = f(x,y)$

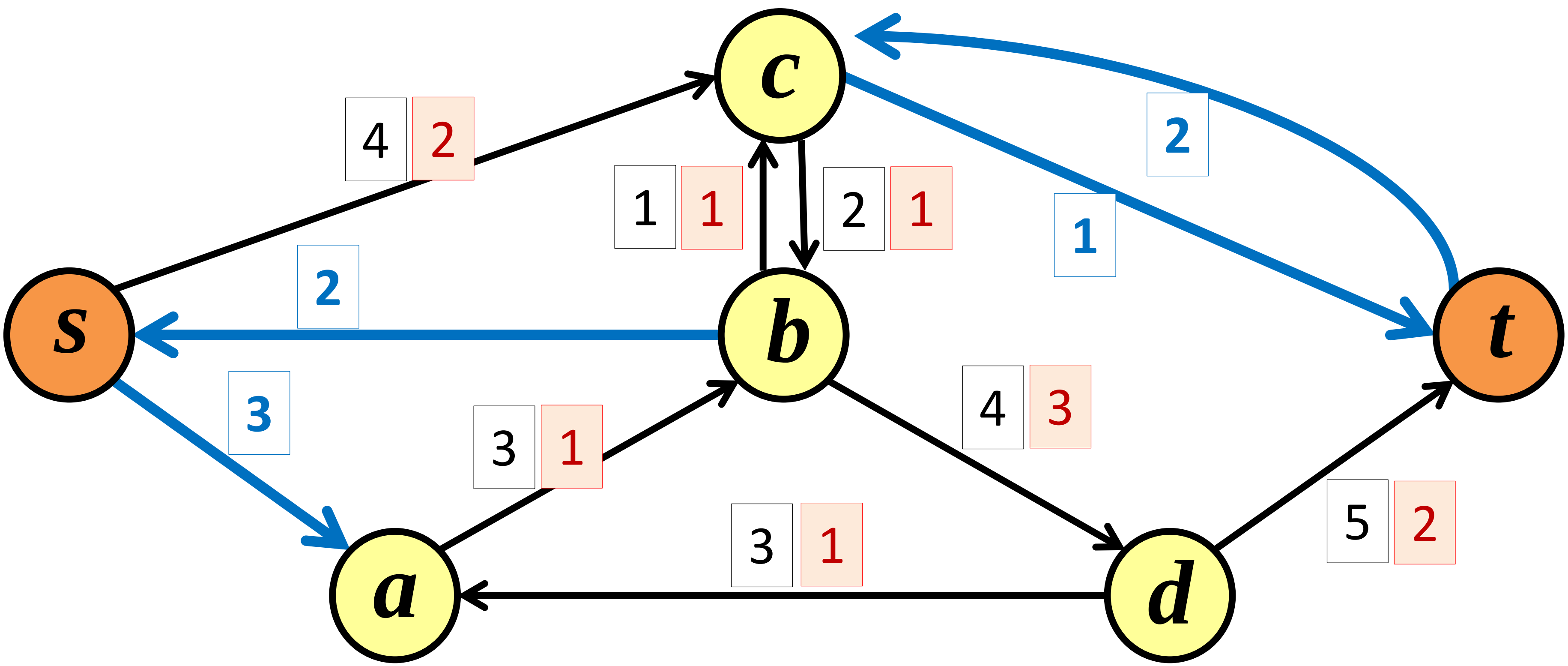


aresta simultaneamente direta/contrária de D

Fluxos em redes

Def.: Dados D e f, define-se a ***rede residual D' = (V, E')***:

- **(x,y)** é ***direta*** $\rightarrow$ **(x,y)** $\in E'$ c/ capacidade $c'(x,y) = c(x,y) - f(x,y)$
- **(x,y)** é ***contrária*** $\rightarrow$ **(y,x)** $\in E'$ c/ capacidade $c'(y,x) = f(x,y)$

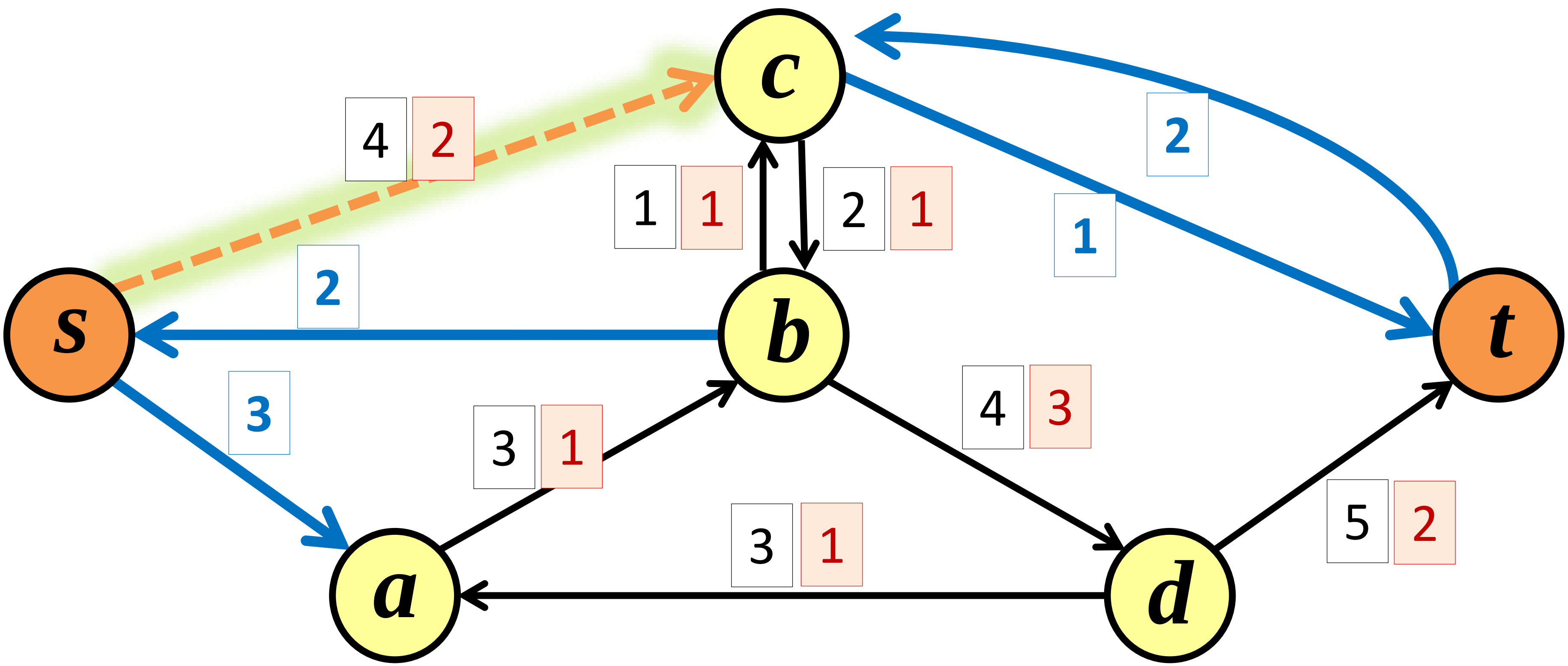


arestas de D'

Fluxos em redes

Def.: Dados D e f, define-se a ***rede residual*** $D' = (V, E')$:

- (x,y) é ***direta*** $\longrightarrow$ $(x,y) \in E'$ c/ capacidade $c'(x,y) = c(x,y) - f(x,y)$
- (x,y) é ***contrária*** $\longrightarrow$ $(y,x) \in E'$ c/ capacidade $c'(y,x) = f(x,y)$

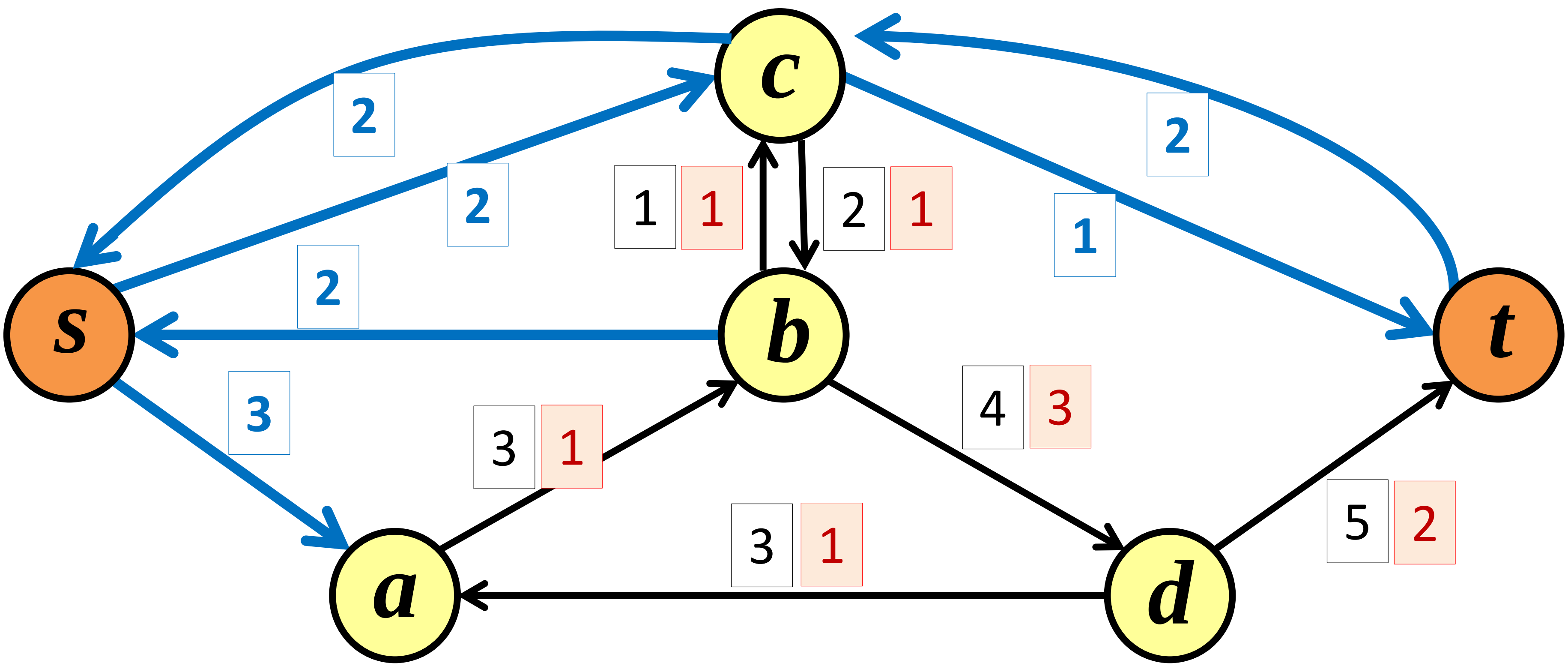


aresta simultaneamente direta/contrária de D

Fluxos em redes

Def.: Dados *D* e *f*, define-se a ***rede residual D' = (V, E')***:

- **(*x*,*y*)** é ***direta*** → **(*x*,*y*)** ∈ *E'* c/ capacidade $c'(x,y) = c(x,y) - f(x,y)$
- **(*x*,*y*)** é ***contrária*** → **(*y*,*x*)** ∈ *E'* c/ capacidade $c'(y,x) = f(x,y)$

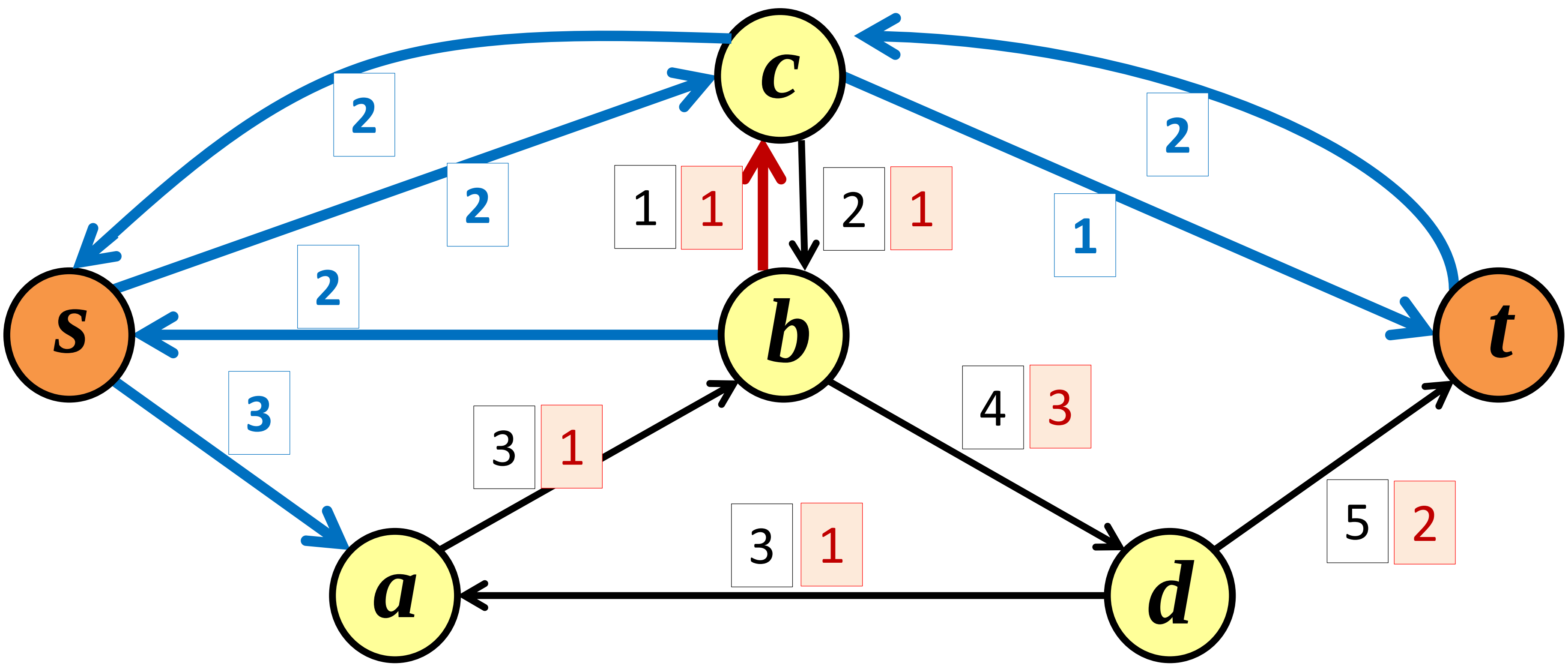


arestas de D'

Fluxos em redes

Def.: Dados D e f, define-se a ***rede residual*** $D' = (V, E')$:

- **(x,y)** é ***direta*** $\longrightarrow$ **(x,y)** $\in E'$ c/ capacidade $c'(x,y) = c(x,y) - f(x,y)$
- **(x,y)** é ***contrária*** $\longrightarrow$ **(y,x)** $\in E'$ c/ capacidade $c'(y,x) = f(x,y)$

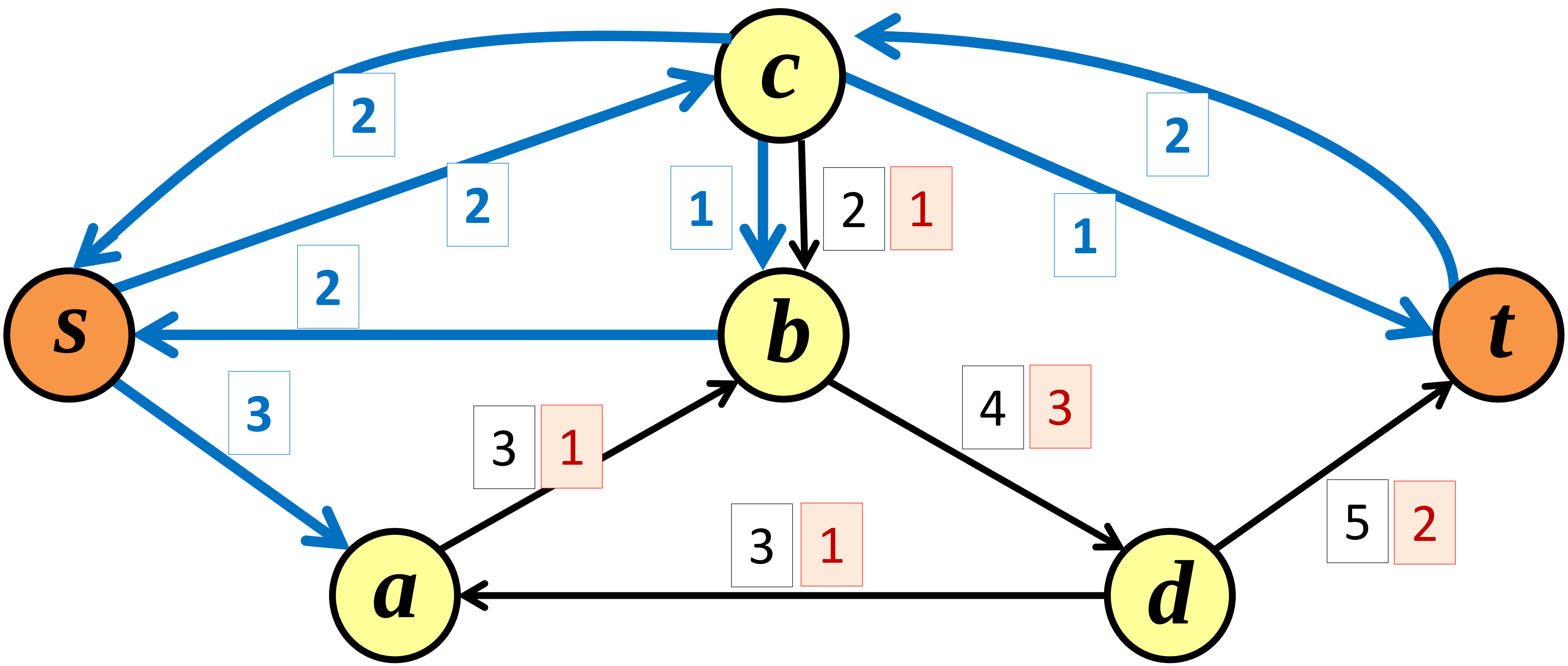


aresta contrária de D

Fluxos em redes

Def.: Dados D e f, define-se a ***rede residual D' = (V, E')***:

- **(*x*,*y*)** é ***direta*** $\longrightarrow$ **(*x*,*y*)** $\in E'$ c/ capacidade $c'(x,y) = c(x,y) - f(x,y)$
- **(*x*,*y*)** é ***contrária*** $\longrightarrow$ **(*y*,*x*)** $\in E'$ c/ capacidade $c'(y,x) = f(x,y)$

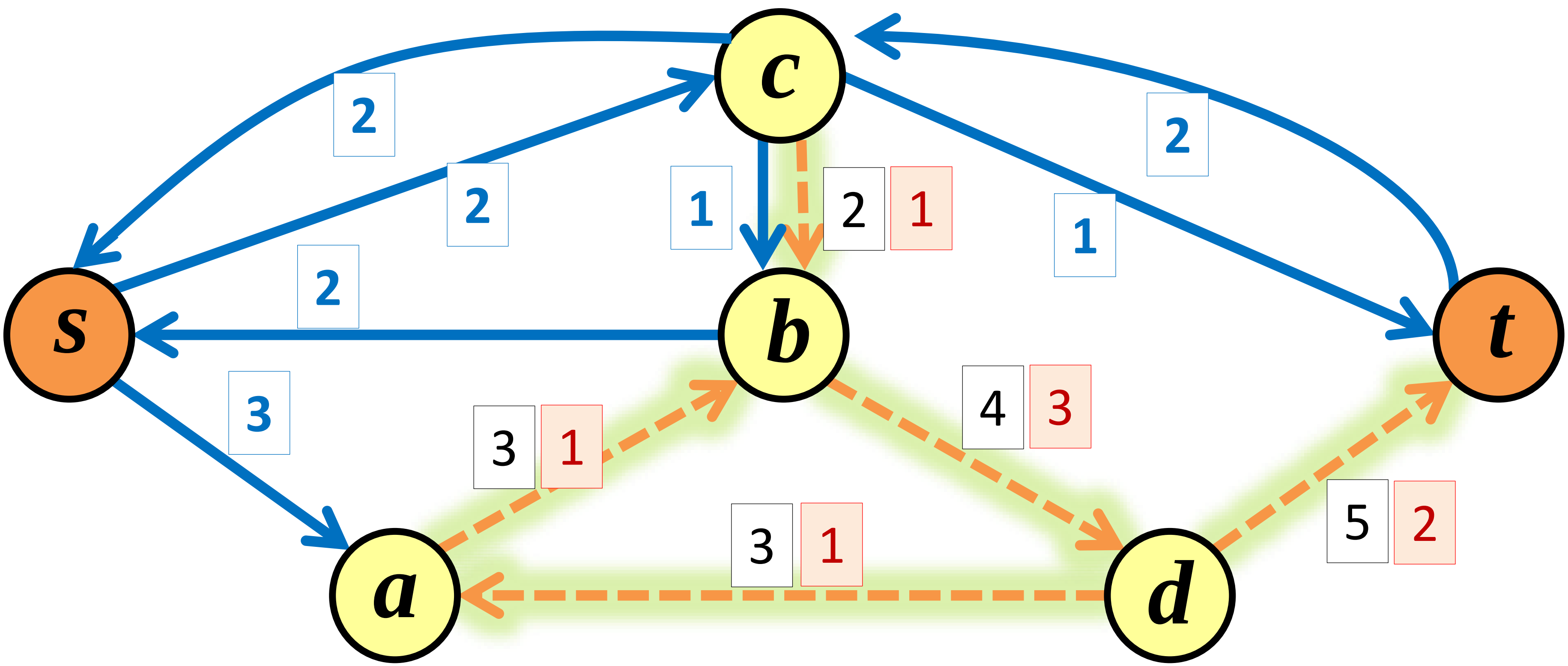


aresta de D'

Fluxos em redes

Def.: Dados D e f, define-se a ***rede residual D' = (V, E')***:

- $(\boldsymbol{x,y})$ é ***direta*** $\rightarrow$ $(\boldsymbol{x,y}) \in E'$ c/ capacidade $c'(x,y) = c(x,y) - f(x,y)$
- $(\boldsymbol{x,y})$ é ***contrária*** $\rightarrow$ $(\boldsymbol{y,x}) \in E'$ c/ capacidade $c'(y,x) = f(x,y)$

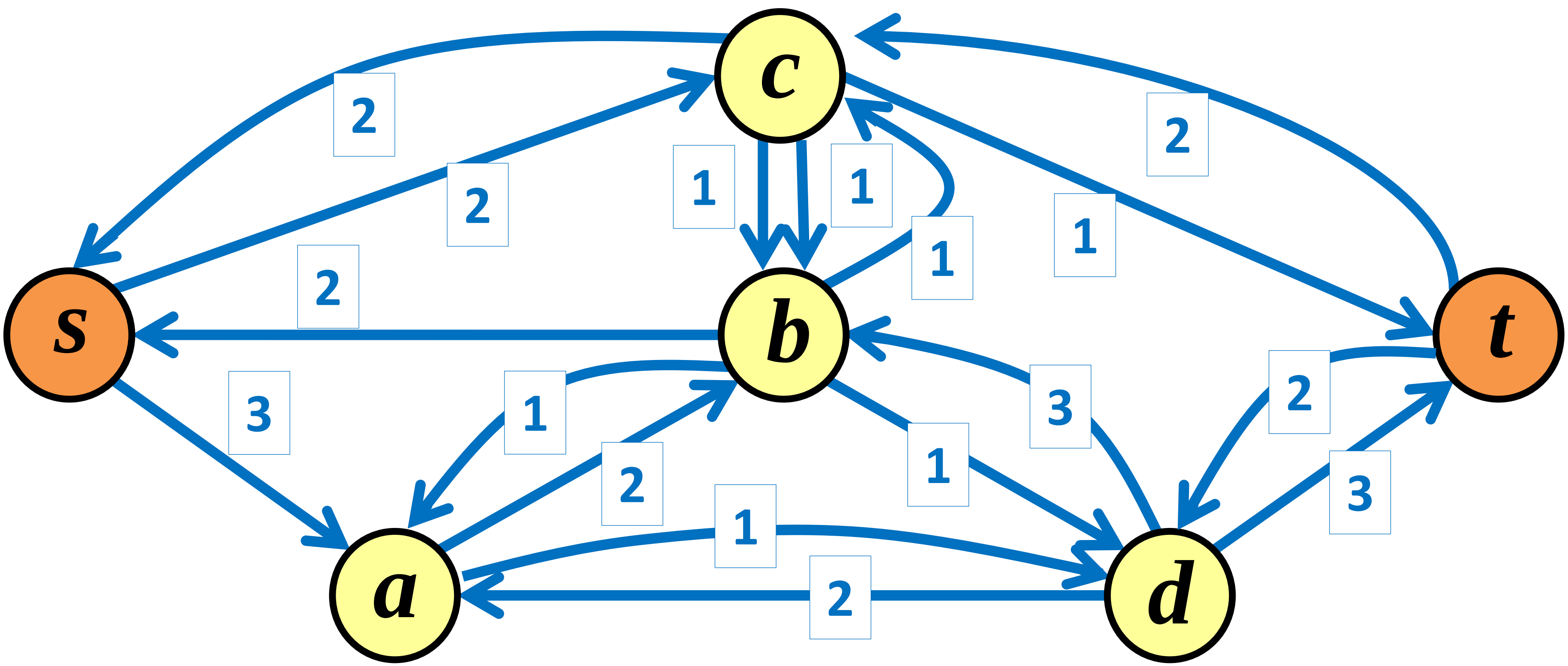


arestas simultaneamente diretas/contrárias de D

Fluxos em redes

Def.: Dados *D* e *f*, define-se a ***rede residual D' = (V, E')***:

- **(*x*,*y*)** é ***direta*** → **(*x*,*y*)** ∈ *E'* c/ capacidade *c'*(*x*,*y*) = *c*(*x*,*y*) − *f*(*x*,*y*)
- **(*x*,*y*)** é ***contrária*** → **(*y*,*x*)** ∈ *E'* c/ capacidade *c'*(*y*,*x*) = *f*(*x*,*y*)



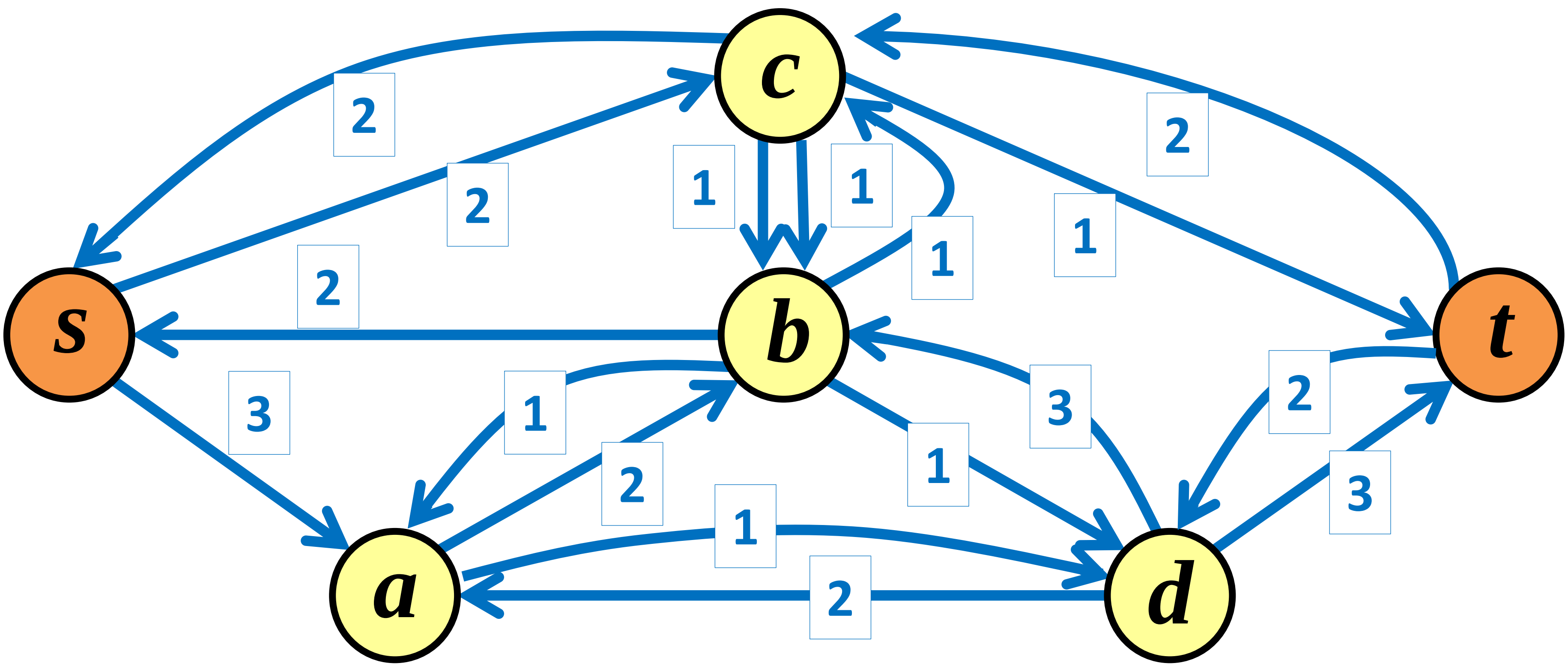
rede residual D' completa

Fluxos em redes

Def.: Uma aresta ***e*** da rede residual ***D'*** é ***aresta de aumento de fluxo*** se é criada a partir de uma aresta ***direta*** de ***D*.**

Fluxos em redes

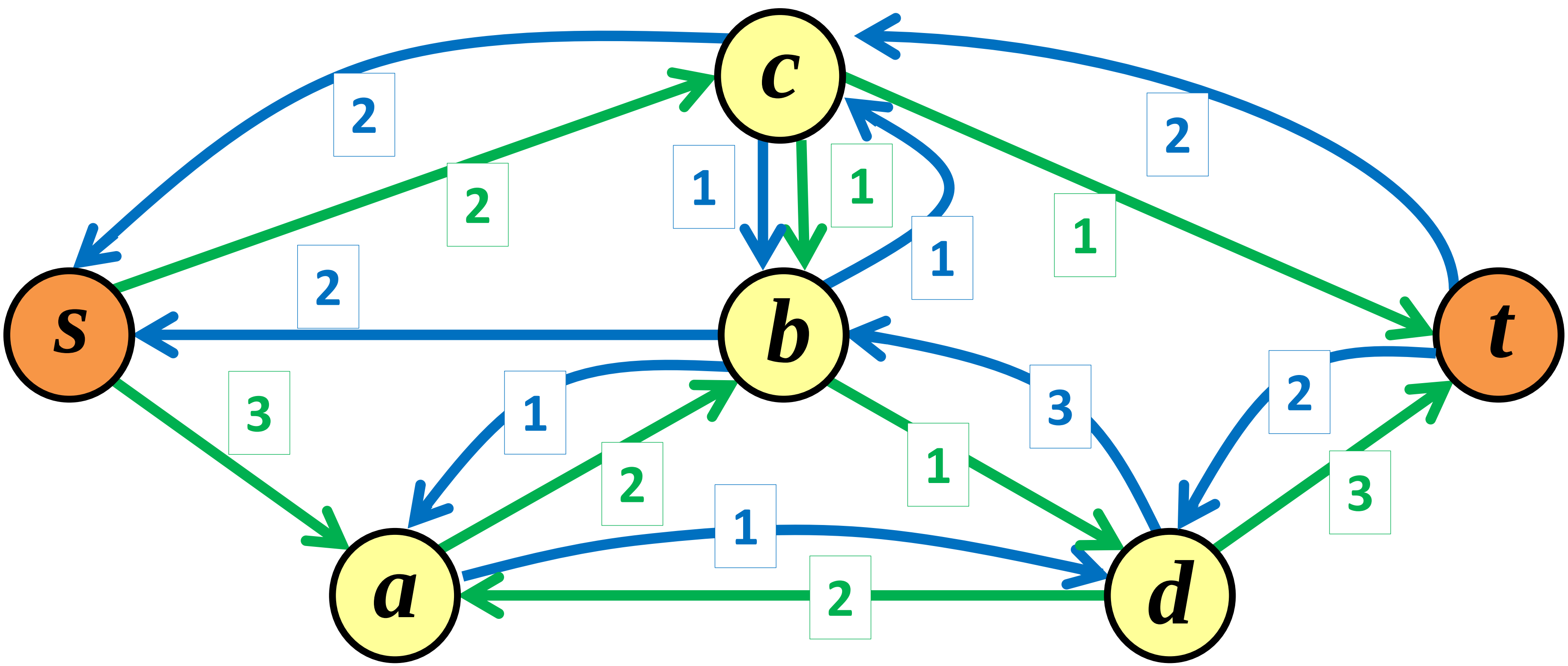
Def.: Uma aresta ***e*** da rede residual ***D'*** é ***aresta de aumento de fluxo*** se é criada a partir de uma aresta ***direta*** de ***D*.**



rede residual D'

Fluxos em redes

Def.: Uma aresta ***e*** da rede residual ***D'*** é ***aresta de aumento de fluxo*** se é criada a partir de uma aresta ***direta*** de ***D*.**



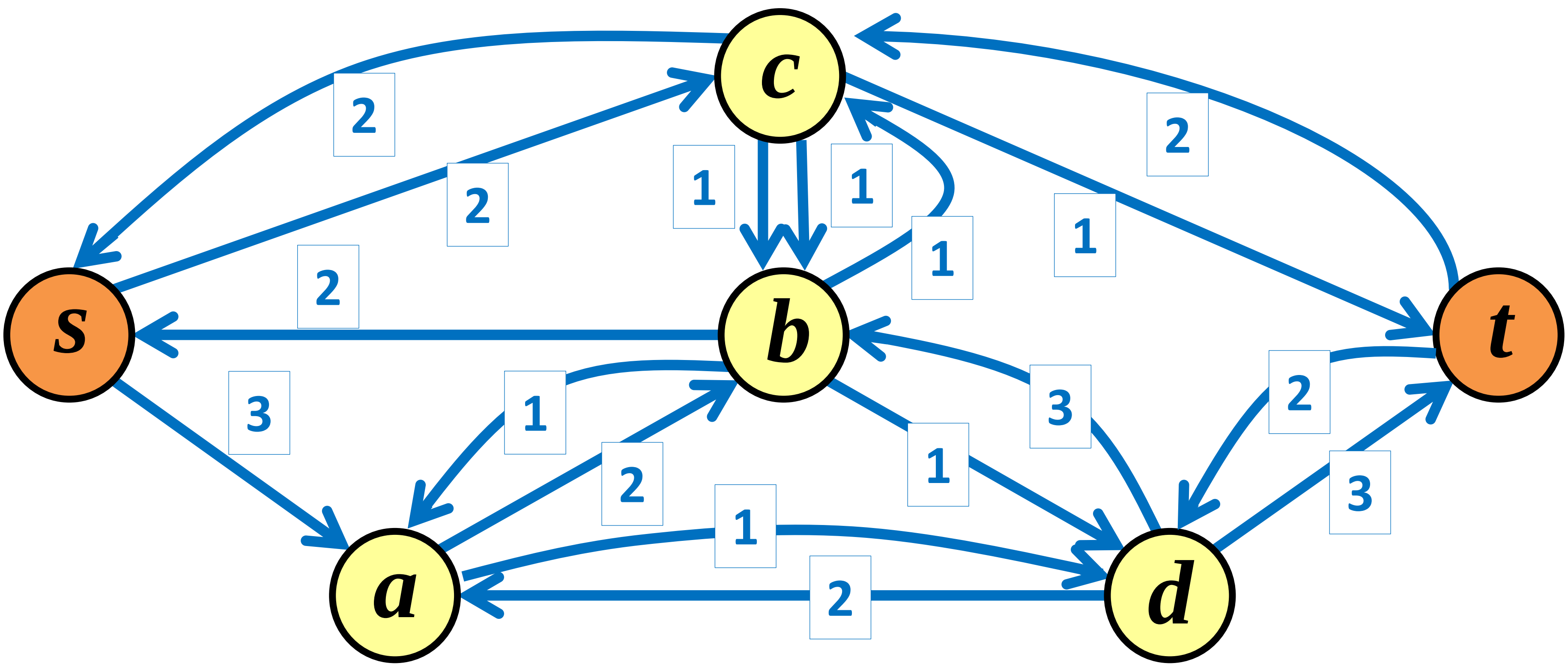
arestas de aumento de fluxo

Fluxos em redes

Def.: Uma aresta ***e*** da rede residual ***D'*** é ***aresta de redução de fluxo*** se é criada a partir de uma aresta ***contrária*** de ***D*.**

Fluxos em redes

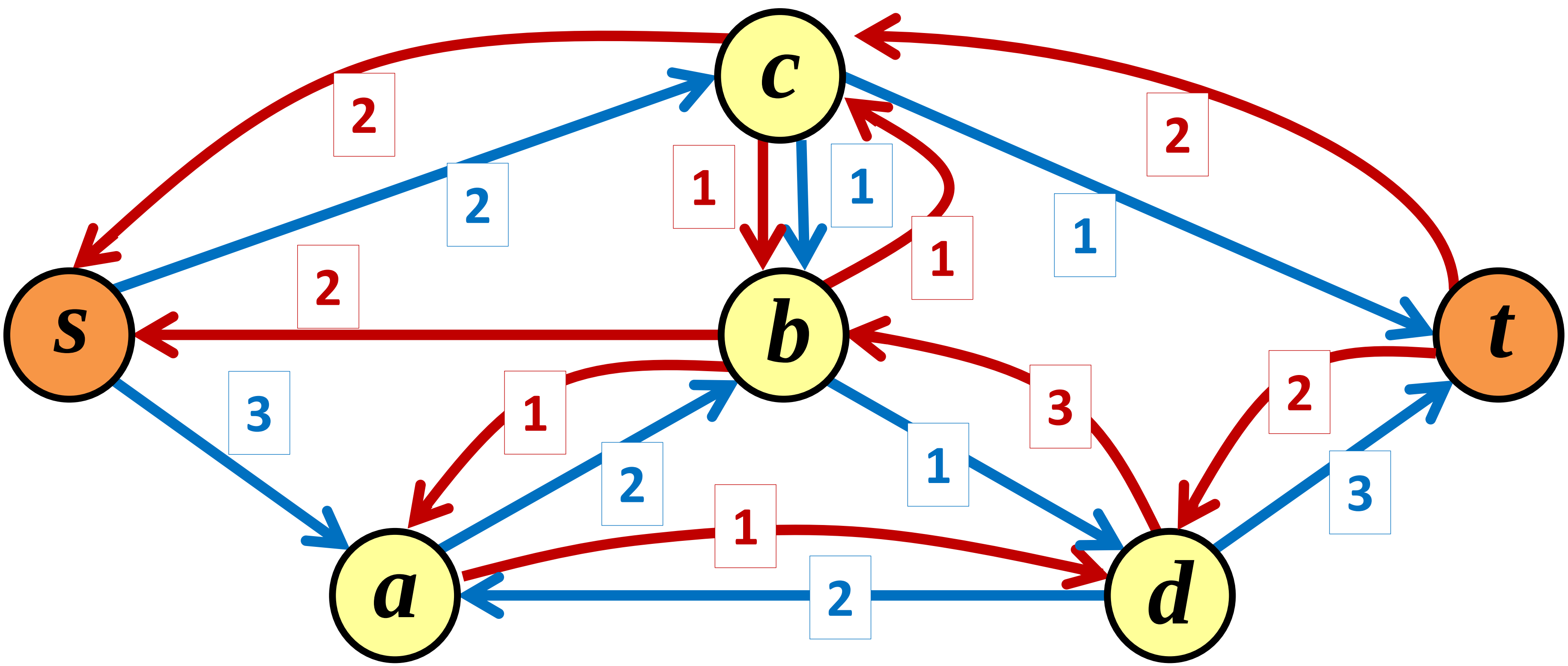
Def.: Uma aresta ***e*** da rede residual ***D'*** é ***aresta de redução de fluxo*** se é criada a partir de uma aresta ***contrária*** de ***D*.**



rede residual D'

Fluxos em redes

Def.: Uma aresta ***e*** da rede residual ***D'*** é ***aresta de redução de fluxo*** se é criada a partir de uma aresta ***contrária*** de ***D*.**



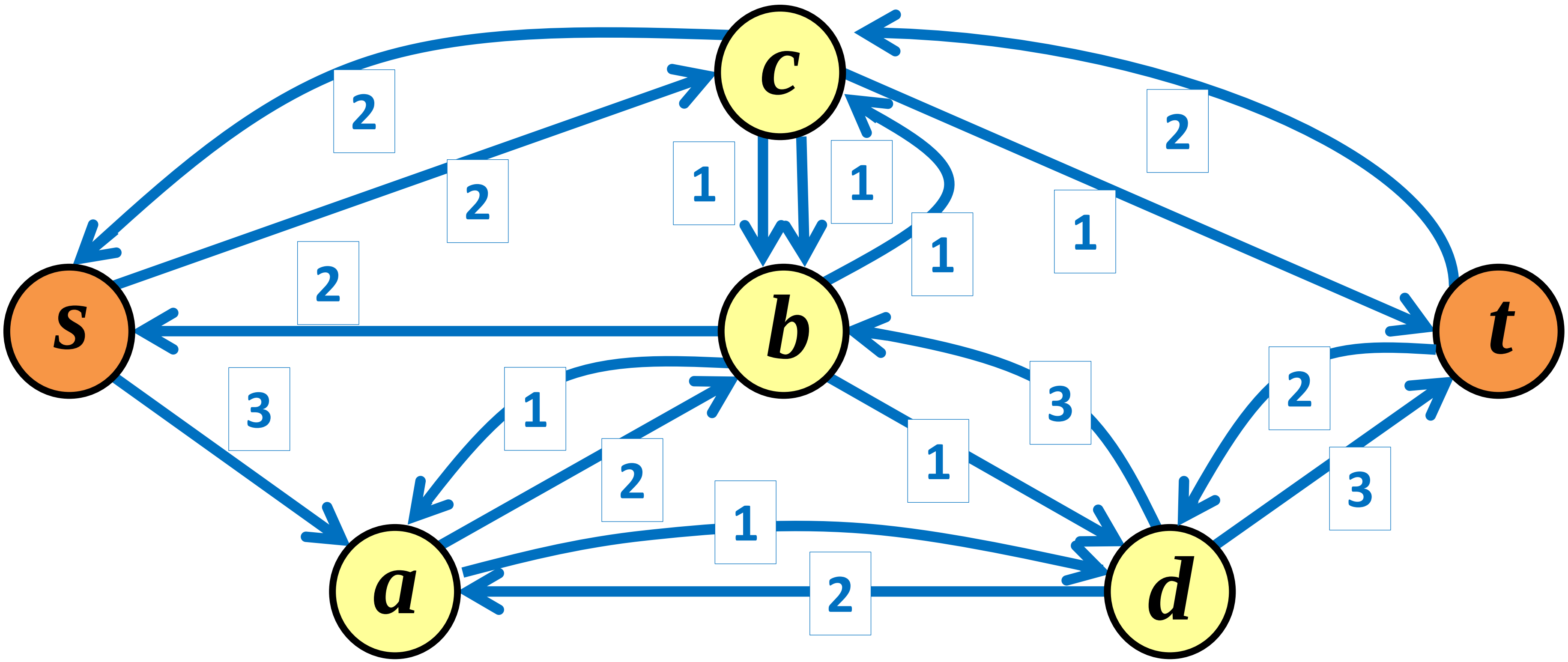
arestas de redução de fluxo

Fluxos em redes

Note que a rede residual D' é na verdade um "mapa" das possíveis variações de fluxo nas arestas!

Fluxos em redes

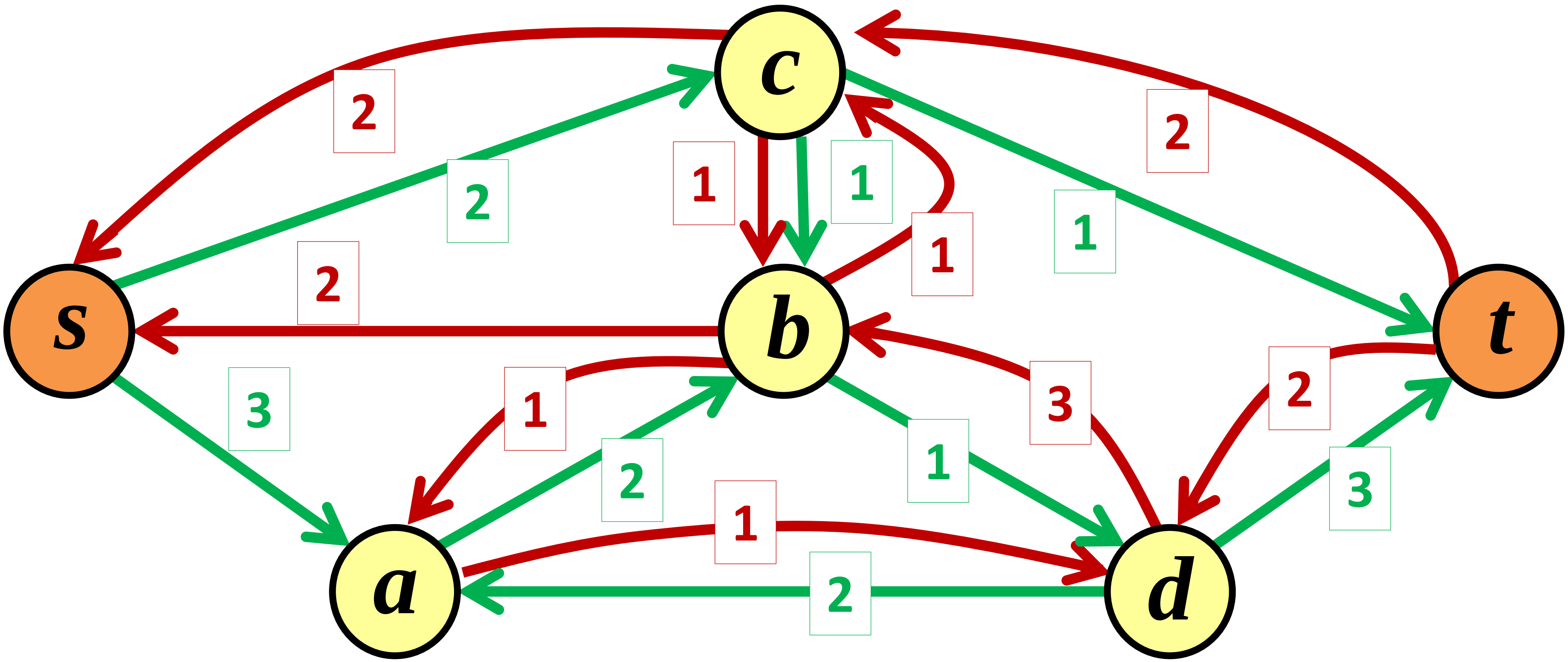
Note que a rede residual D' é na verdade um "mapa" das possíveis variações de fluxo nas arestas!



rede residual D'

Fluxos em redes

Note que a rede residual D’ é na verdade um “mapa” das possíveis variações de fluxo nas arestas!



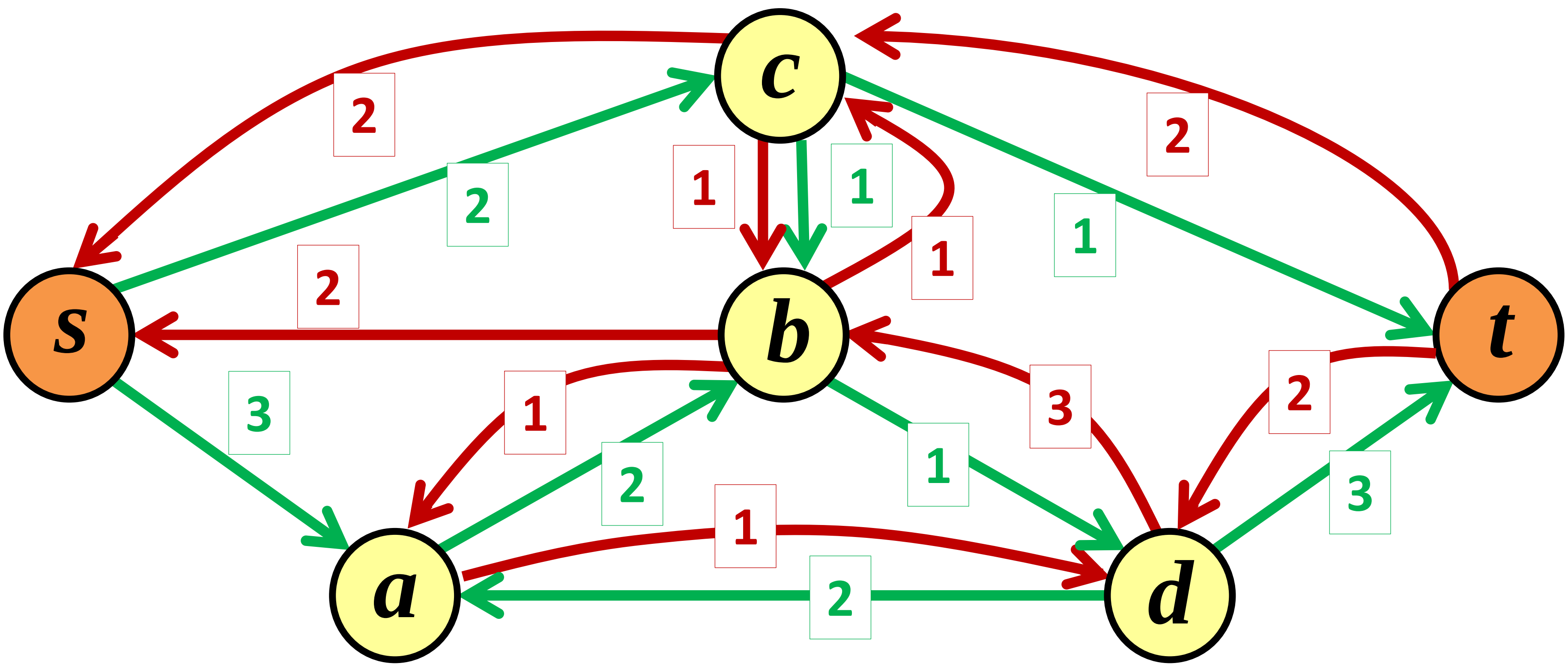
arestas de aumento/redução de fluxo em D’

Fluxos em redes

Def.: Um ***caminho aumentante*** é um caminho de ***s*** a ***t*** na rede residual ***D'***.

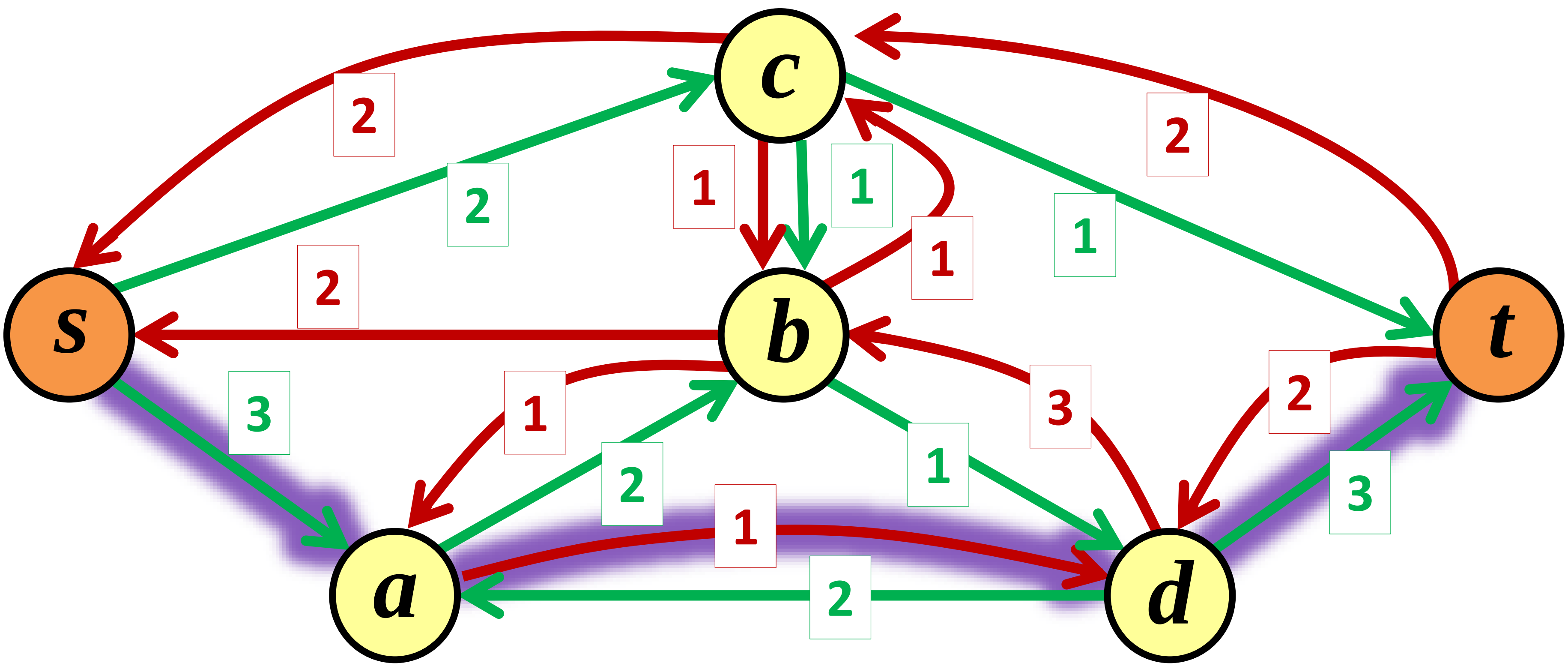
Fluxos em redes

Def.: Um ***caminho aumentante*** é um caminho de ***s*** a ***t*** na rede residual ***D'***.



Fluxos em redes

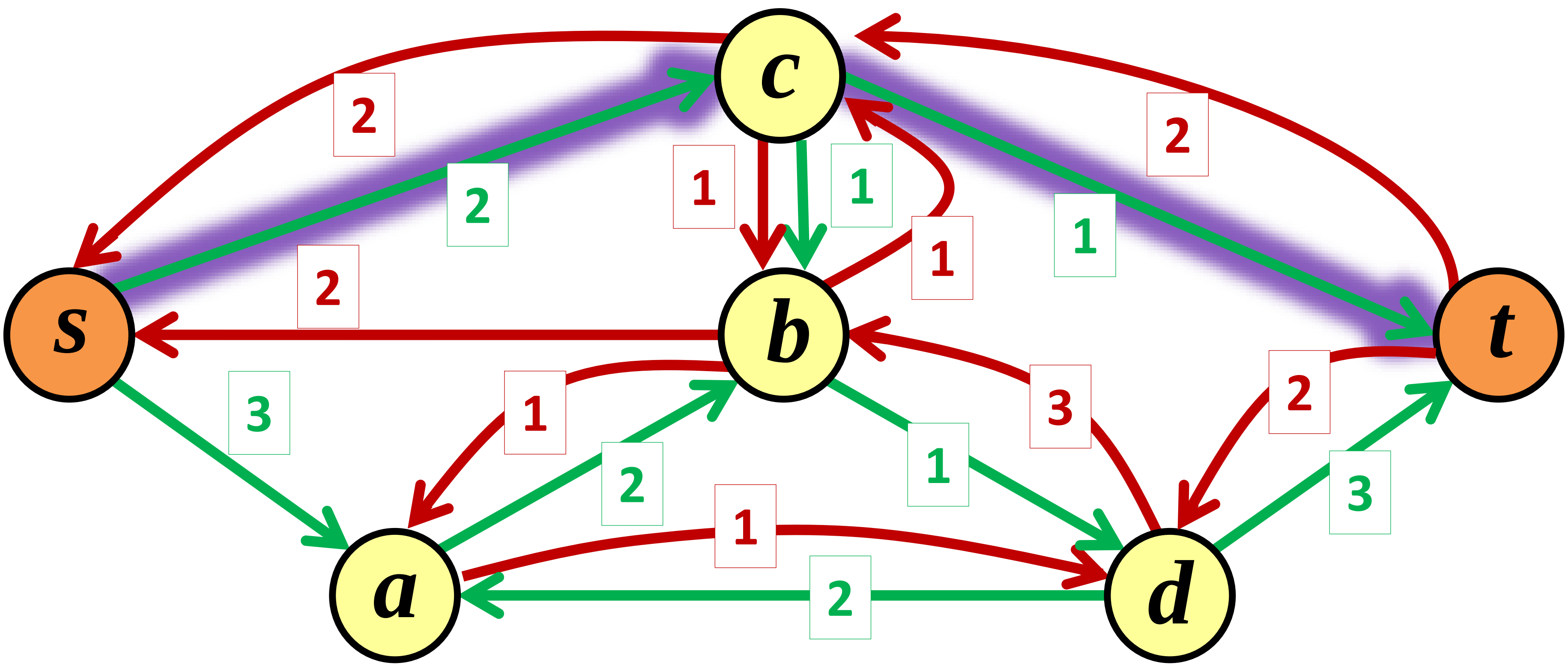
Def.: Um ***caminho aumentante*** é um caminho de ***s*** a ***t*** na rede residual ***D'***.



exemplo de caminho aumentante

Fluxos em redes

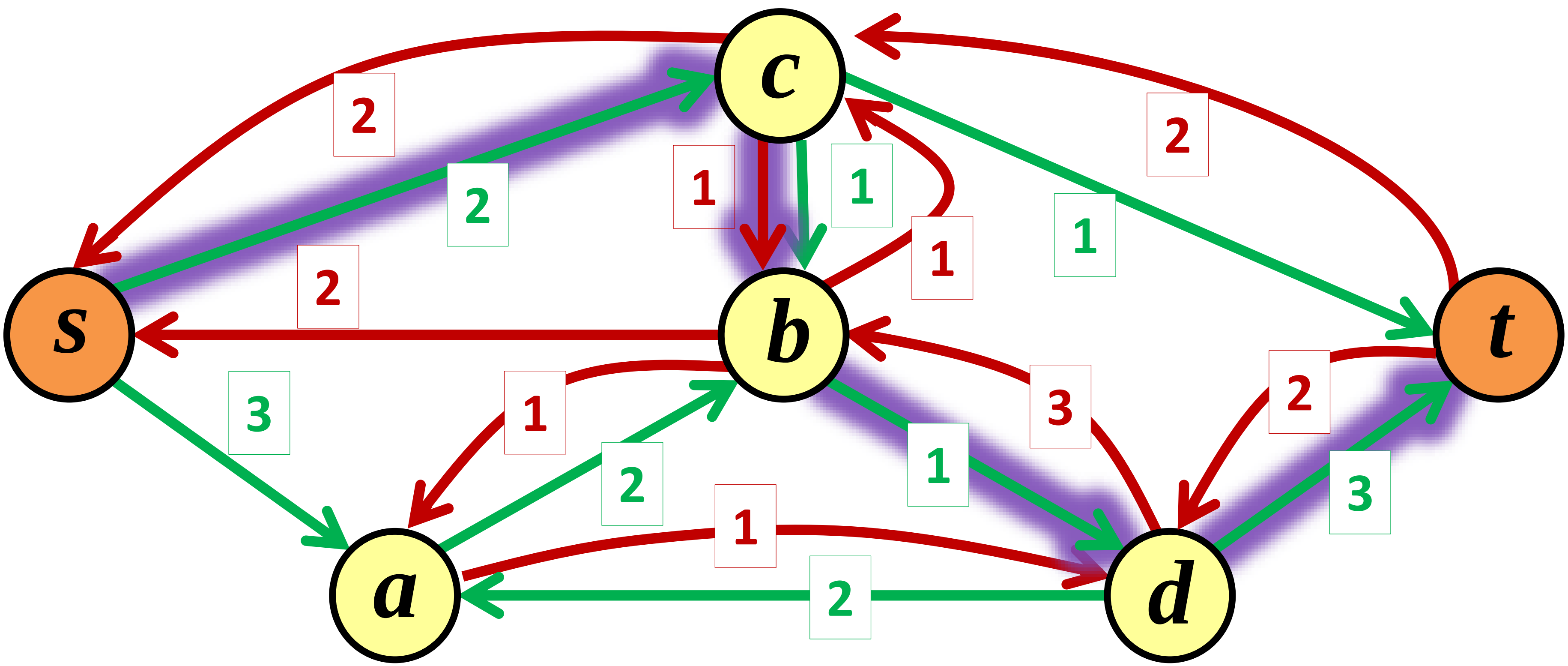
Def.: Um ***caminho aumentante*** é um caminho de ***s*** a ***t*** na rede residual ***D'***.



exemplo de caminho aumentante

Fluxos em redes

Def.: Um ***caminho aumentante*** é um caminho de ***s*** a ***t*** na rede residual ***D'***.



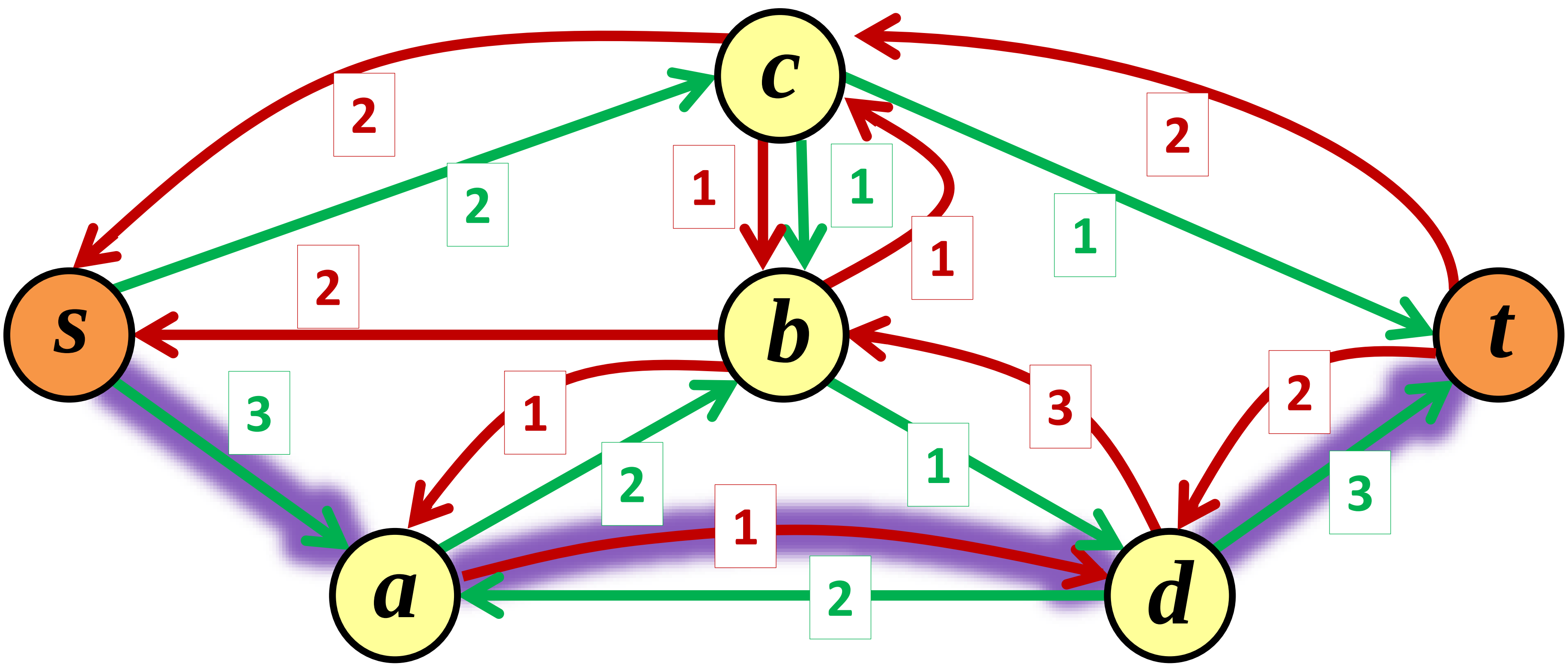
exemplo de caminho aumentante

Fluxos em redes

Def.: O ***gargalo*** de um caminho aumentante em ***D'*** é a menor capacidade de uma aresta ao longo deste caminho.

Fluxos em redes

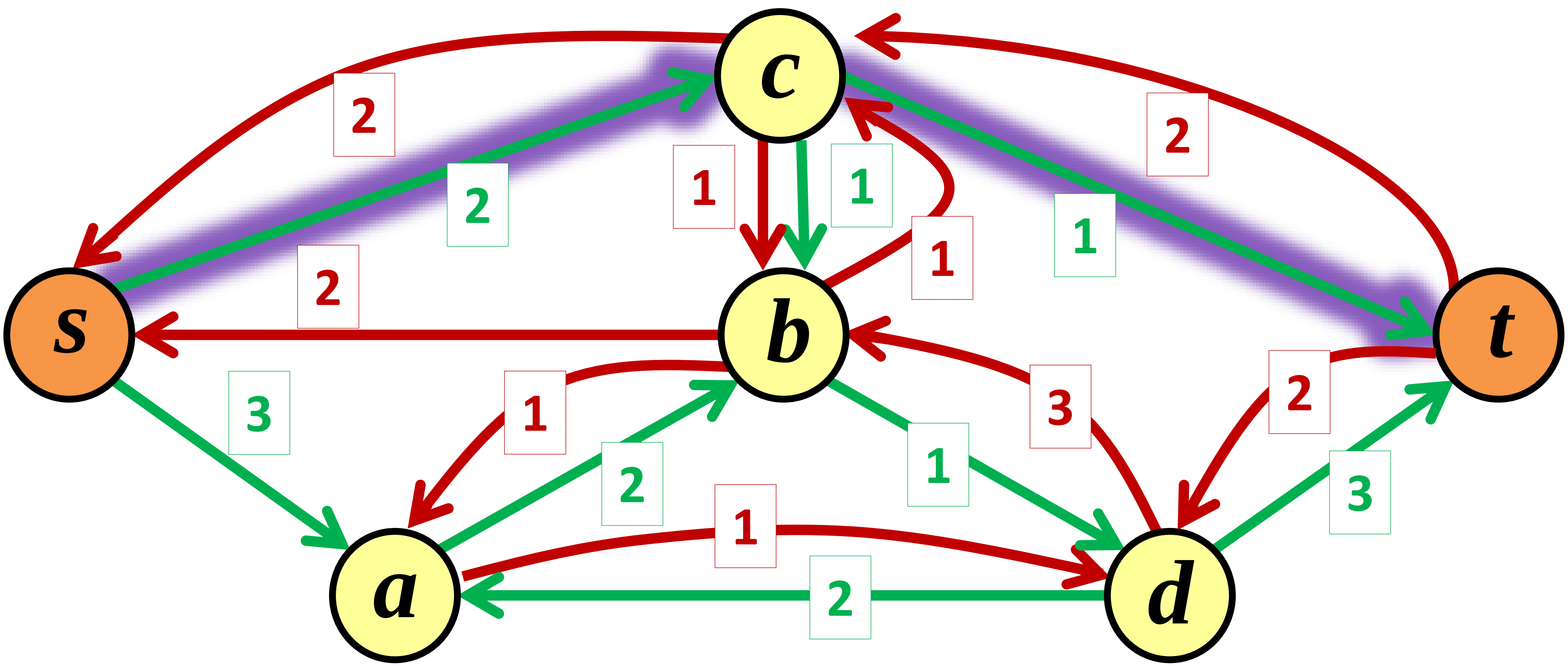
Def.: O ***gargalo*** de um caminho aumentante em ***D'*** é a menor capacidade de uma aresta ao longo deste caminho.



gargalo igual a 1

Fluxos em redes

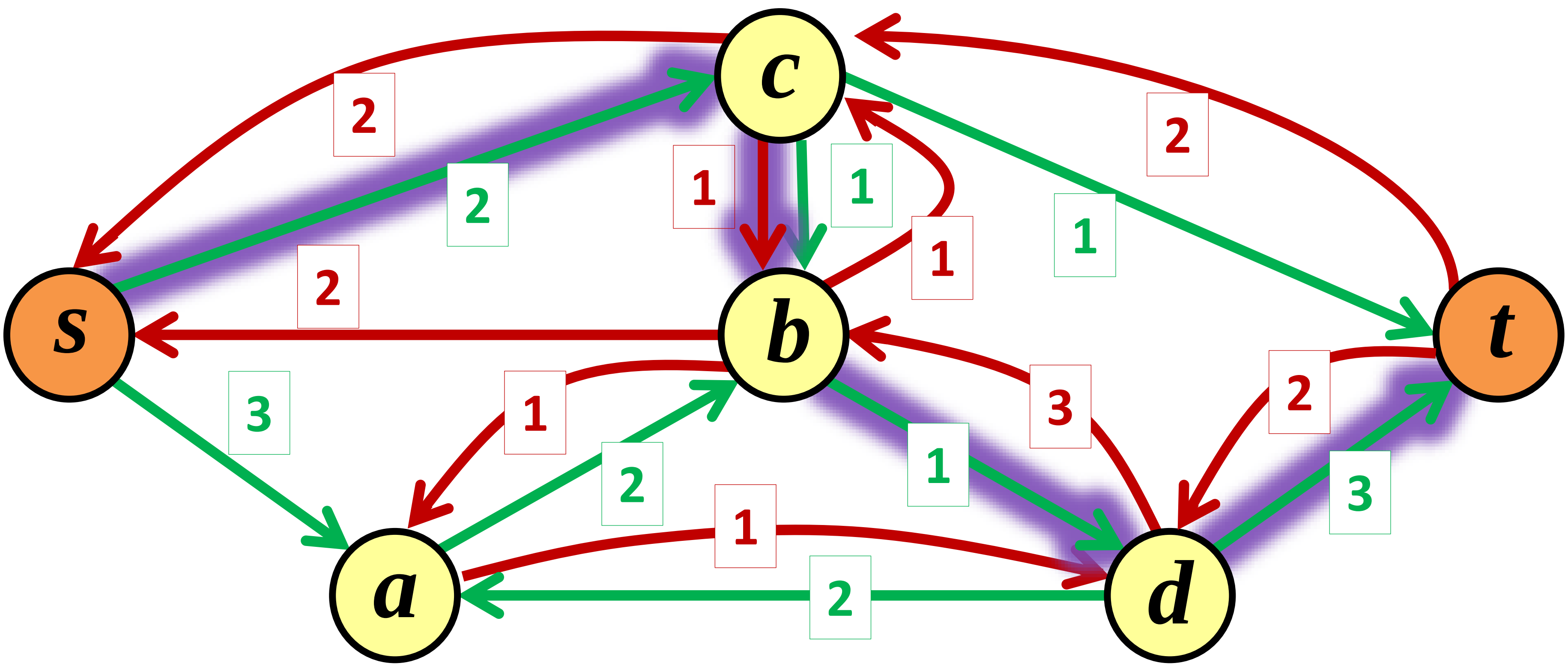
Def.: O ***gargalo*** de um caminho aumentante em ***D'*** é a menor capacidade de uma aresta ao longo deste caminho.



***gargalo igual a* 1**

Fluxos em redes

Def.: O ***gargalo*** de um caminho aumentante em ***D'*** é a menor capacidade de uma aresta ao longo deste caminho.



gargalo igual a 1

Fluxos em redes

Lema: Dados D e f, se há um caminho aumentante em D' cujo gargalo é g, então existe um fluxo f_{novo} em D com valor $f_{novo}(D) = f(D) + g$.

Fluxos em redes

Lema: Dados $\boldsymbol{D}$ e $\boldsymbol{f}$, se há um caminho aumentante em $\boldsymbol{D'}$ cujo gargalo é $\boldsymbol{g}$, então existe um fluxo $\boldsymbol{f_{novo}}$ em $\boldsymbol{D}$ com valor $\boldsymbol{f_{novo}(D) = f\,(D) + g}$.

Demonstração:

Seja $\boldsymbol{e'_1\; e'_2\; e'_3 \ldots e'_k}$ um caminho aumentante em $\boldsymbol{D'}$ com gargalo $\boldsymbol{g}$.

Sejam $\boldsymbol{e_1 , e_2 , e_3 , \ldots , e_k}$ as arestas correspondentes em $\boldsymbol{D}$.

Para $\boldsymbol{j = 1, 2, \ldots, k}$, faça:

- se $\boldsymbol{e'_j}$ é ***aresta de aumento de fluxo***, então $\boldsymbol{f_{novo}(e_j) = f\,(e_j) + g}$
- se $\boldsymbol{e'_j}$ é ***aresta de redução de fluxo***, então $\boldsymbol{f_{novo}(e_j) = f\,(e_j) - g}$

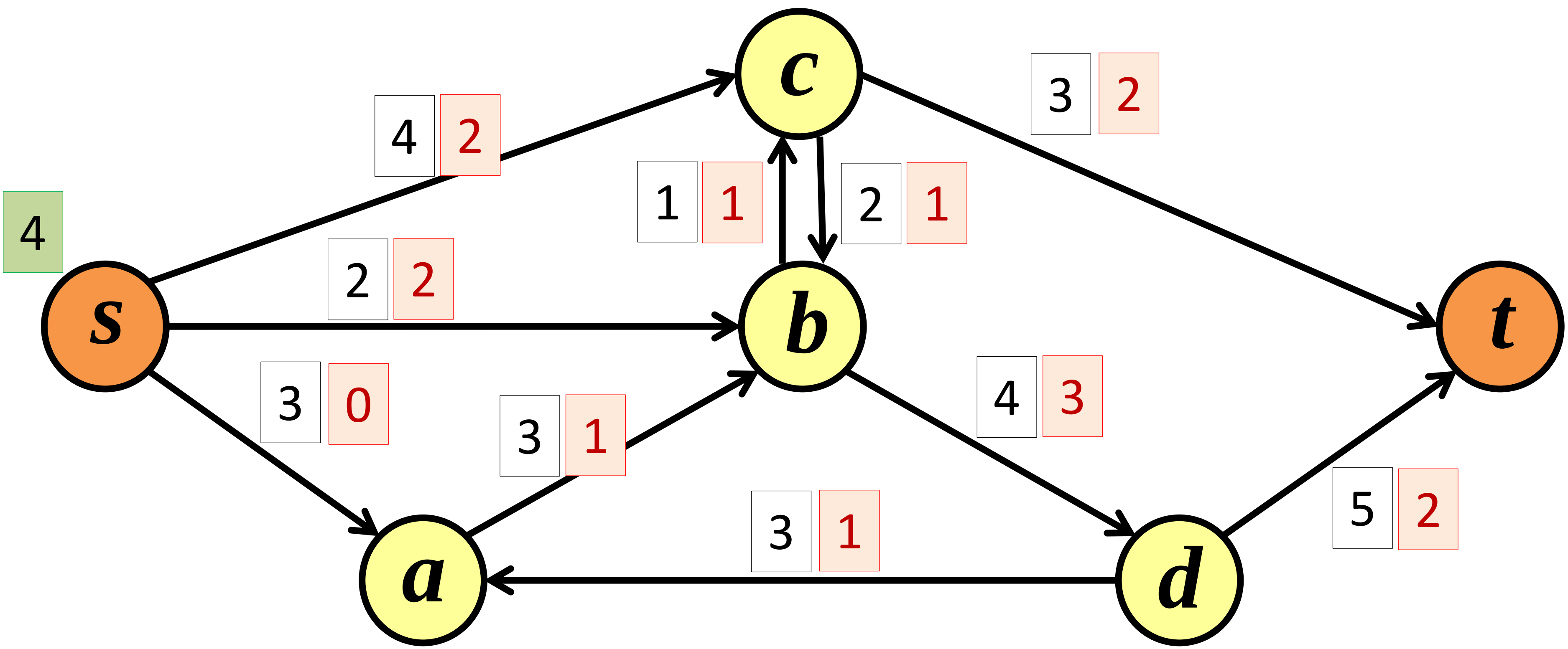
Para as demais arestas de D, faça $\boldsymbol{f_{novo}(e_j) = f\,(e_j)}$.

Fluxos em redes

Ilustração do lema

Fluxos em redes

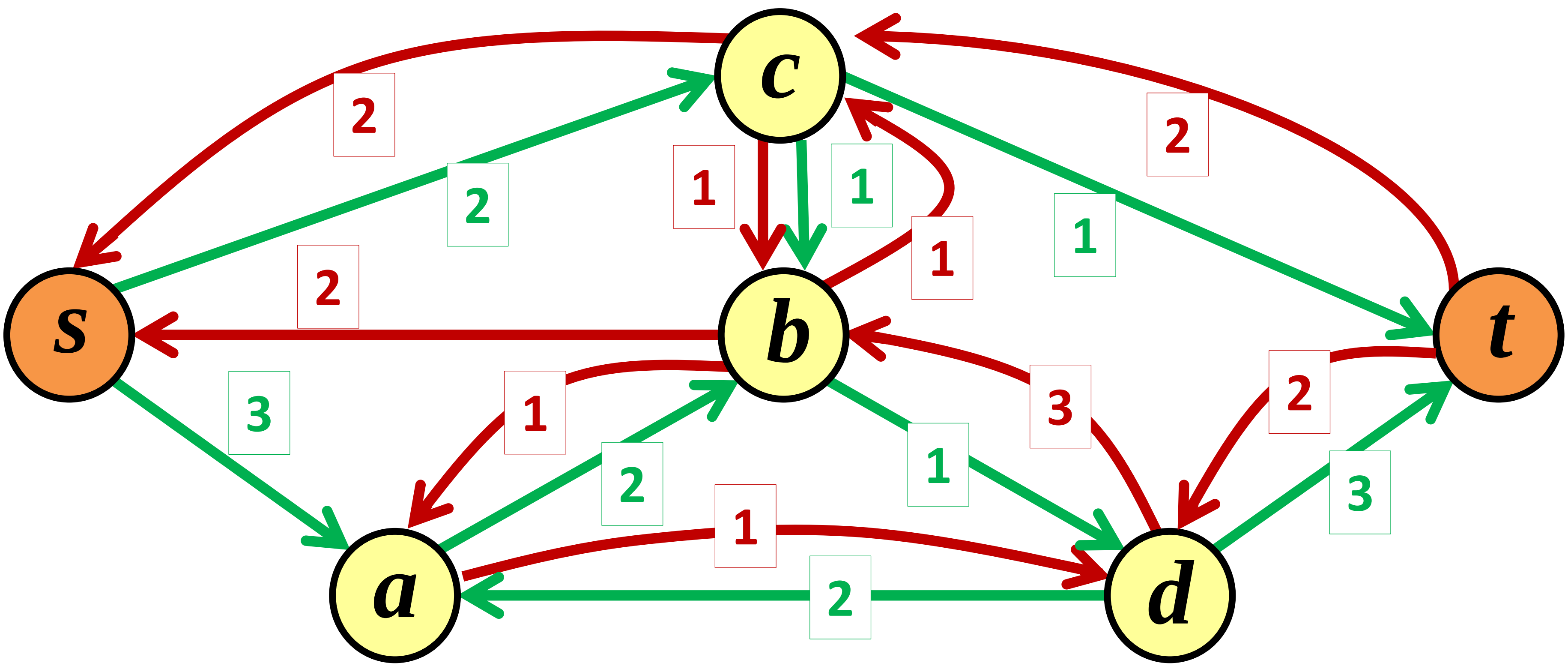
Ilustração do lema



rede D com fluxo f tal que f(D)=4

Fluxos em redes

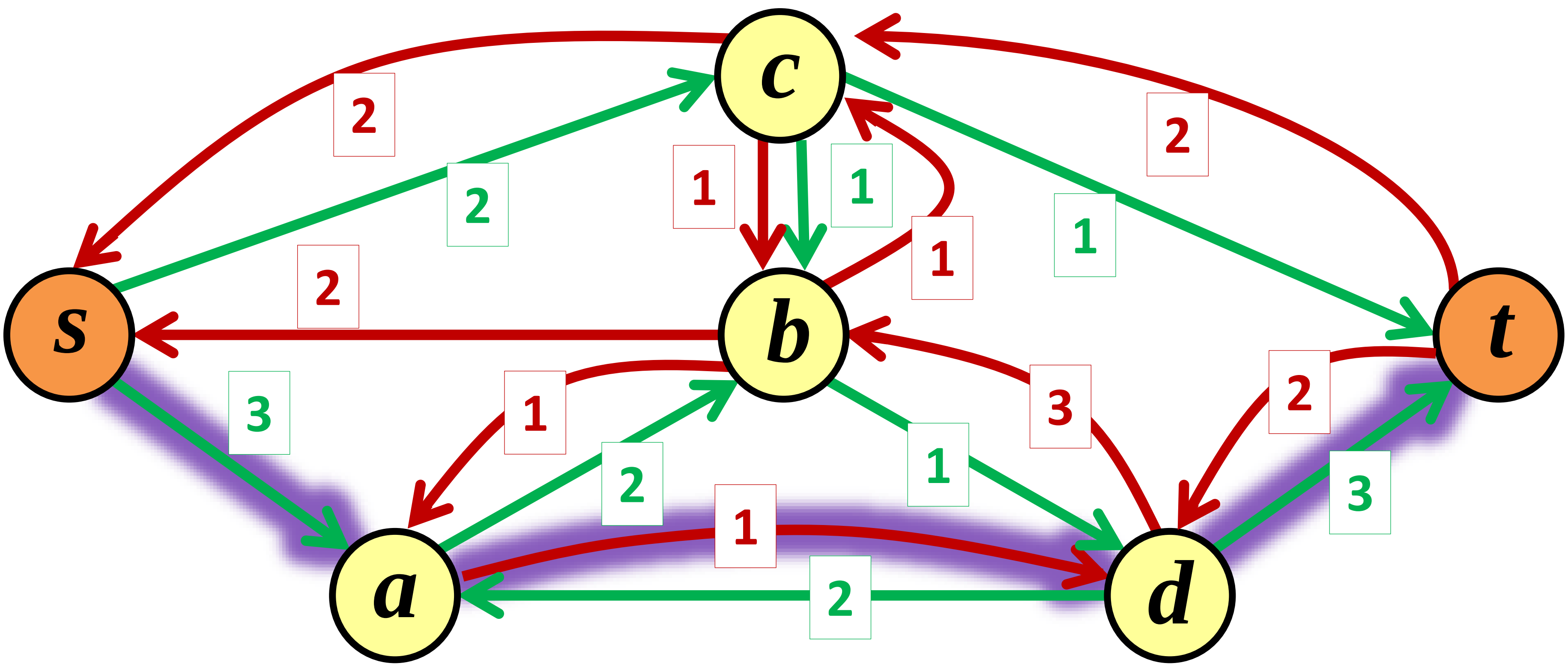
Ilustração do lema



rede residual D'de D

Fluxos em redes

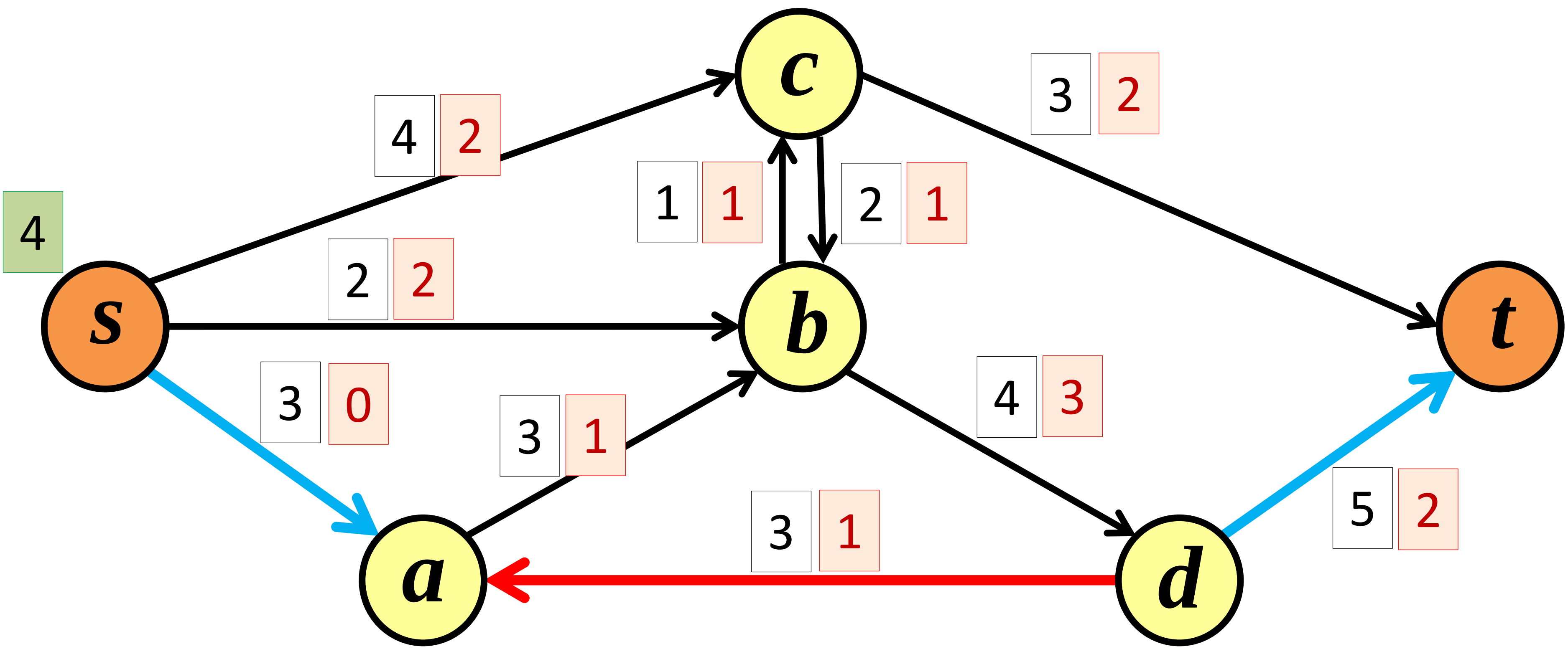
Ilustração do lema



caminho aumentante em D'com gargalo 1

Fluxos em redes

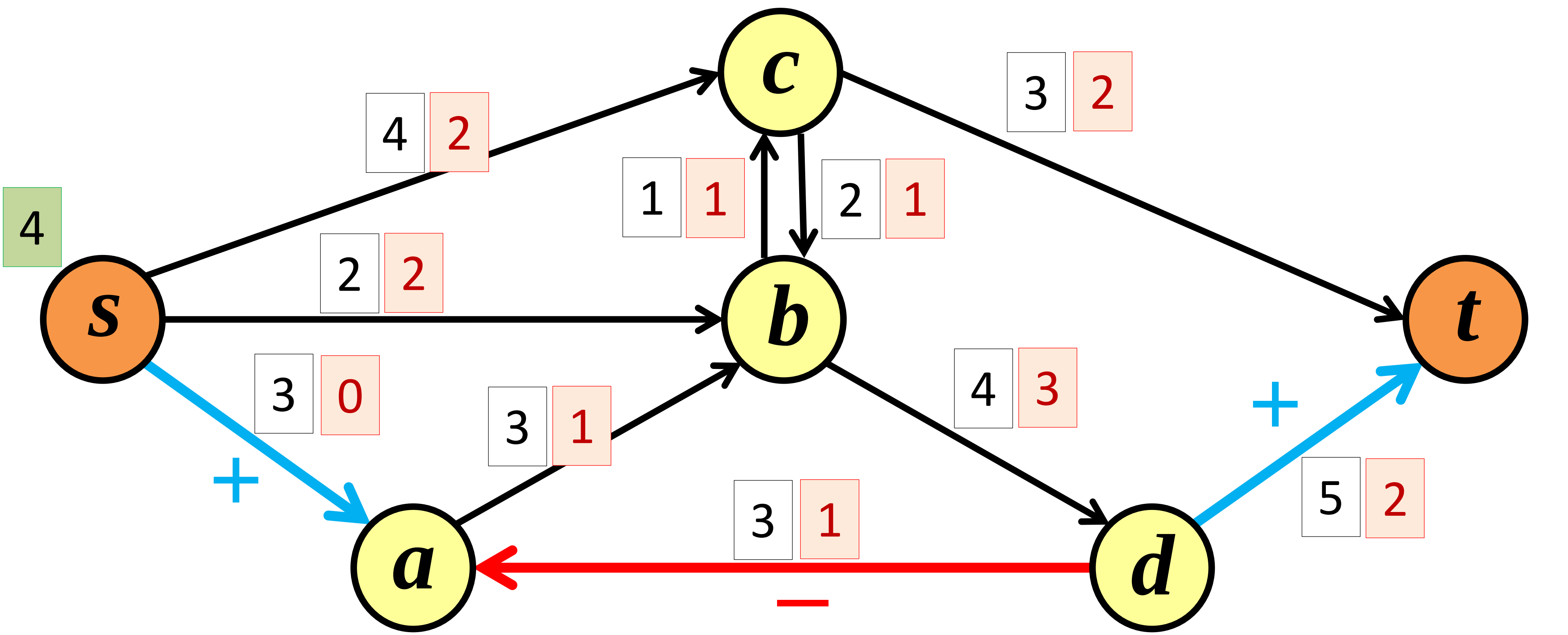
Ilustração do lema



arestas correspondentes em D

Fluxos em redes

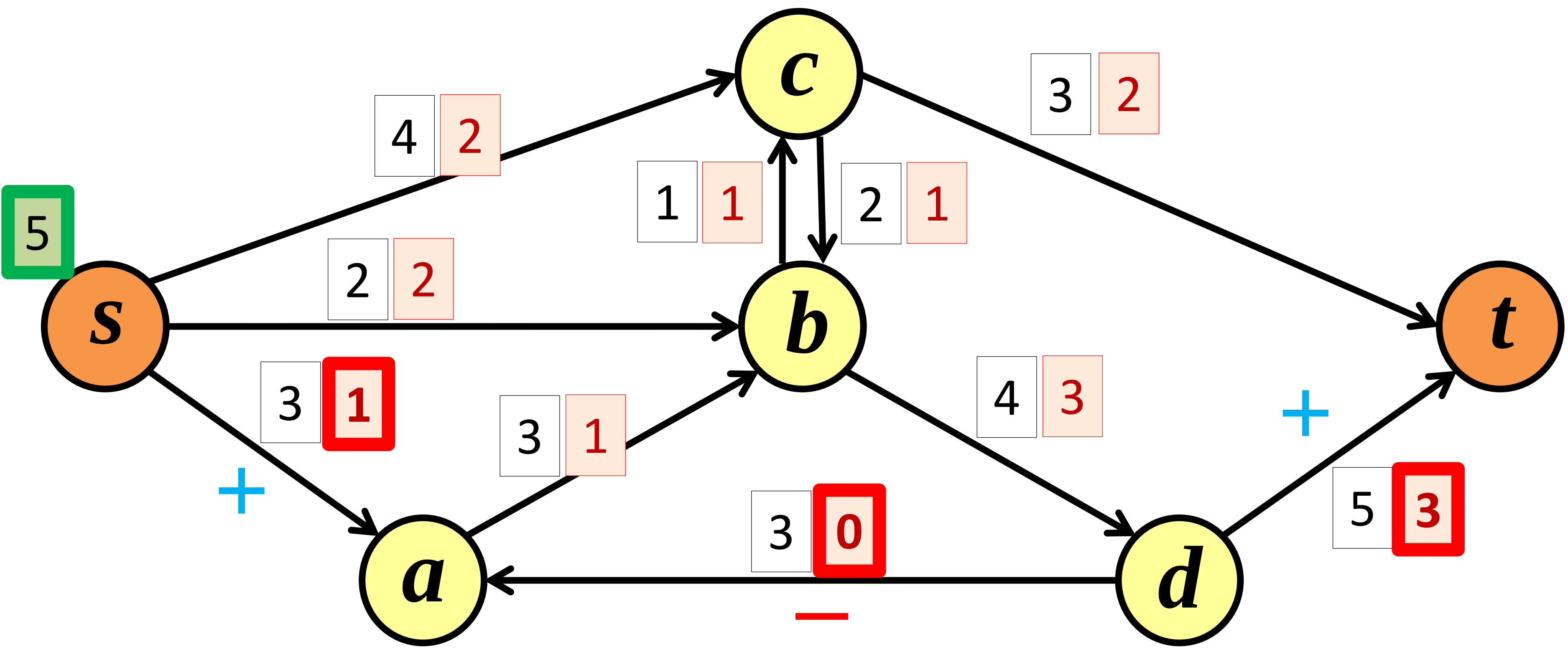
Ilustração do lema



arestas correspondentes em D

Fluxos em redes

Ilustração do lema



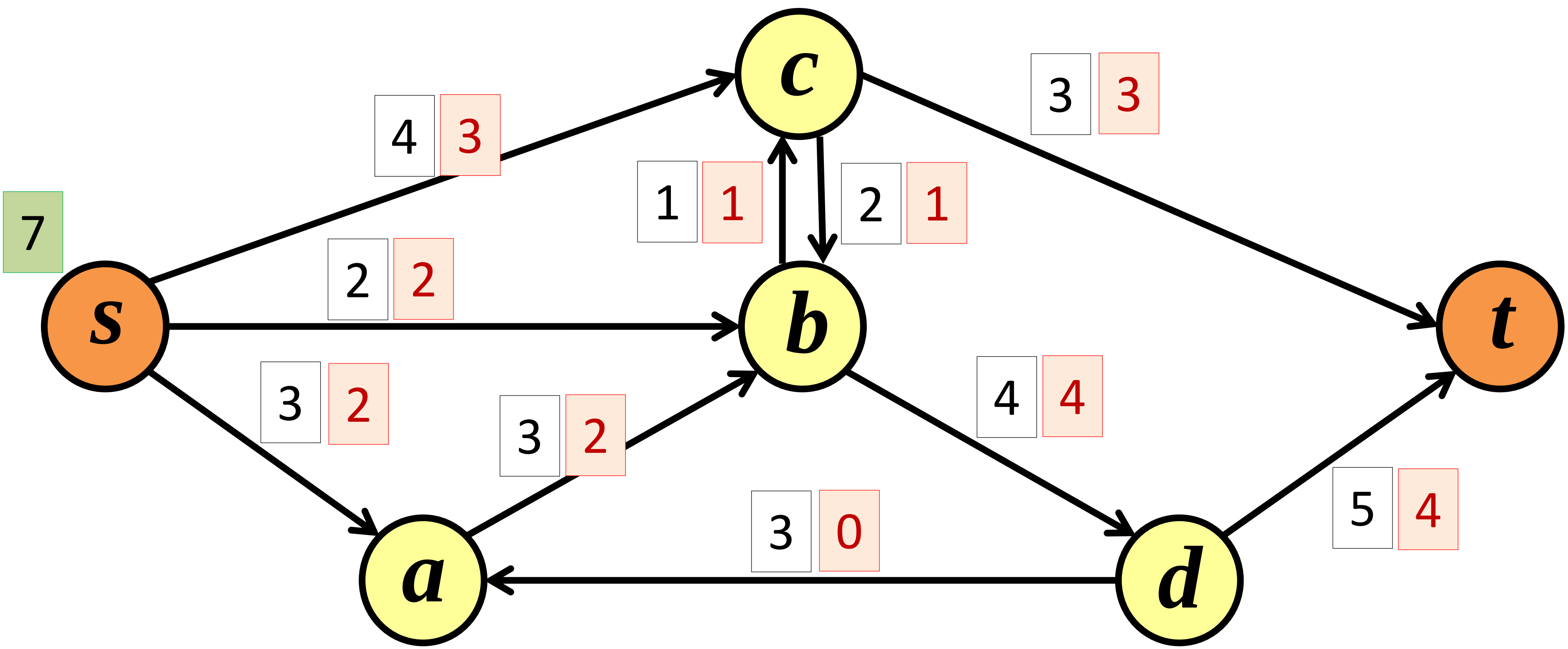
novo fluxo f_{novo} em D tal que $f_{novo}(D) = 5$

Fluxos em redes

Teorema: Dados ***D*** e ***f***, temos que:
f é fluxo máximo **sss** não há caminho aumentante em ***D'***.

Fluxos em redes

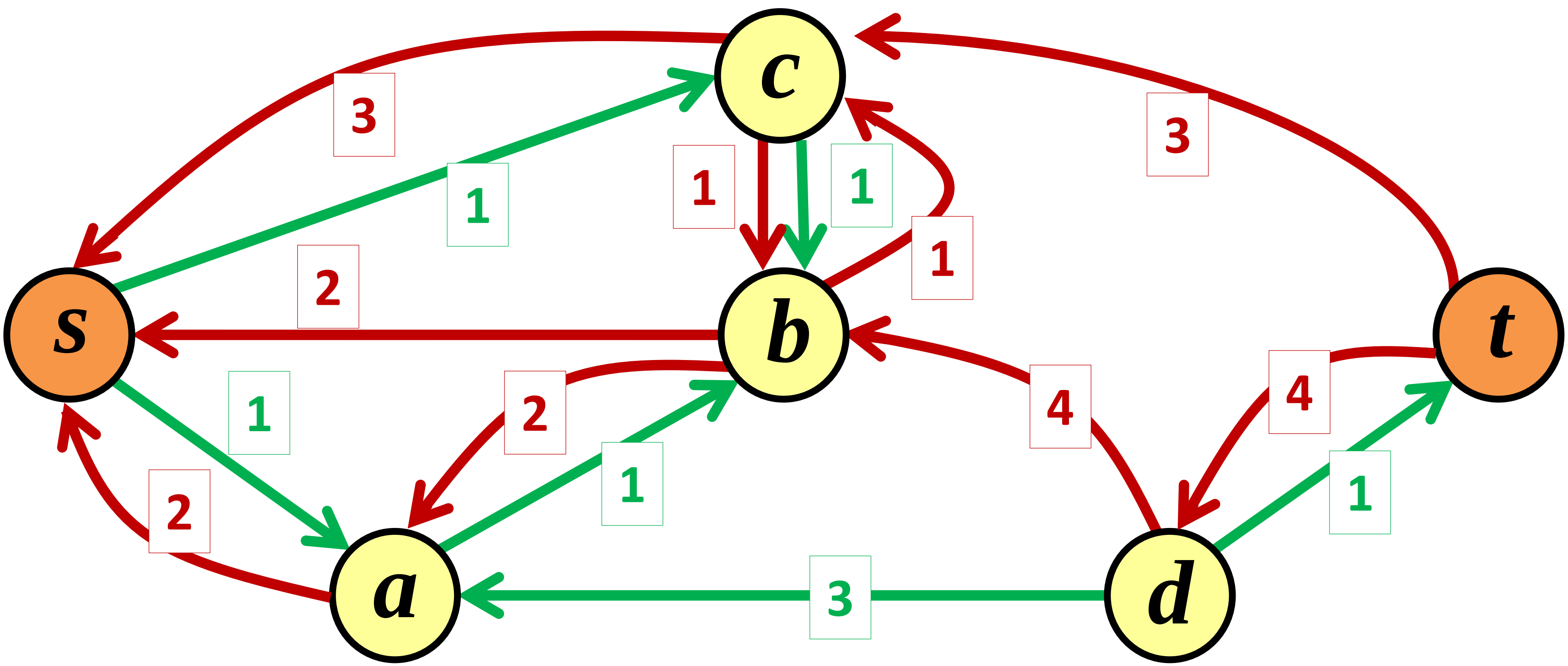
Teorema: Dados ***D*** e ***f***, temos que:
f é fluxo máximo **sss** não há caminho aumentante em ***D’***.



fluxo f máximo com valor f (D) = 7

Fluxos em redes

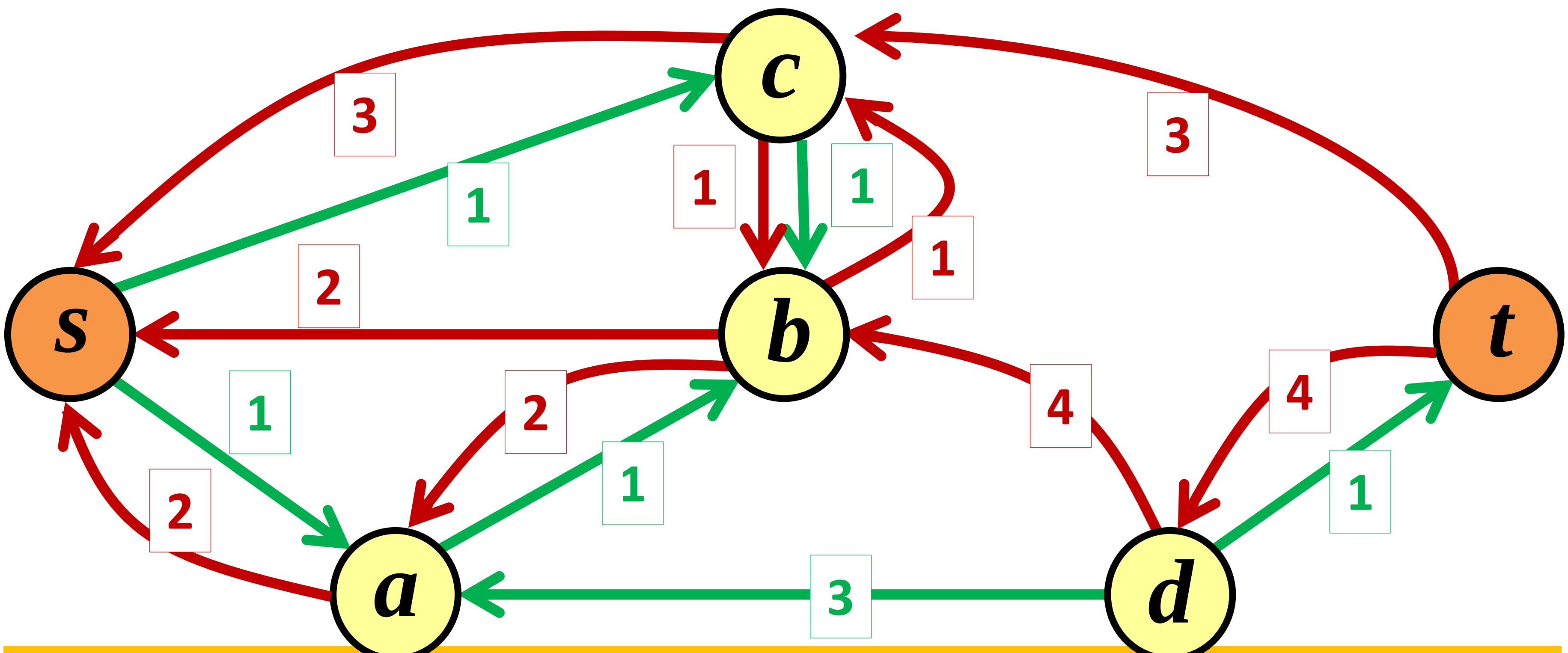
Teorema: Dados ***D*** e ***f***, temos que:
f é fluxo máximo **sss** não há caminho aumentante em ***D'***.



rede residual D' relativa ao fluxo f

Fluxos em redes

Teorema: Dados ***D*** e ***f***, temos que:
f é fluxo máximo **sss** não há caminho aumentante em ***D'***.



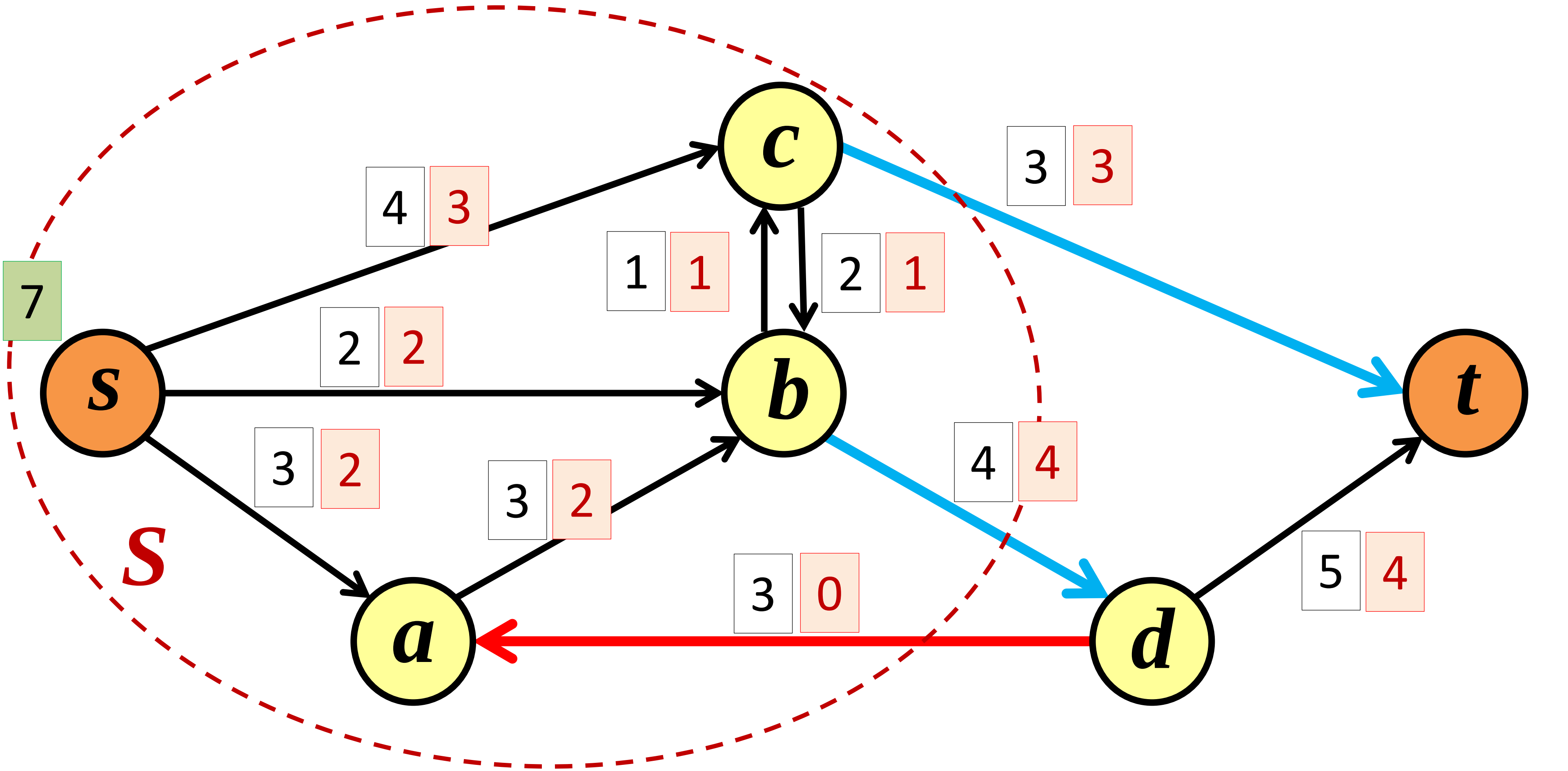
não há caminho aumentante na rede residual D'!

Fluxos em redes

Teorema: Seja f um fluxo **máximo** em uma rede D, e (S, S') um corte **mínimo** em D. Então $f(D) = c(S, S')$.

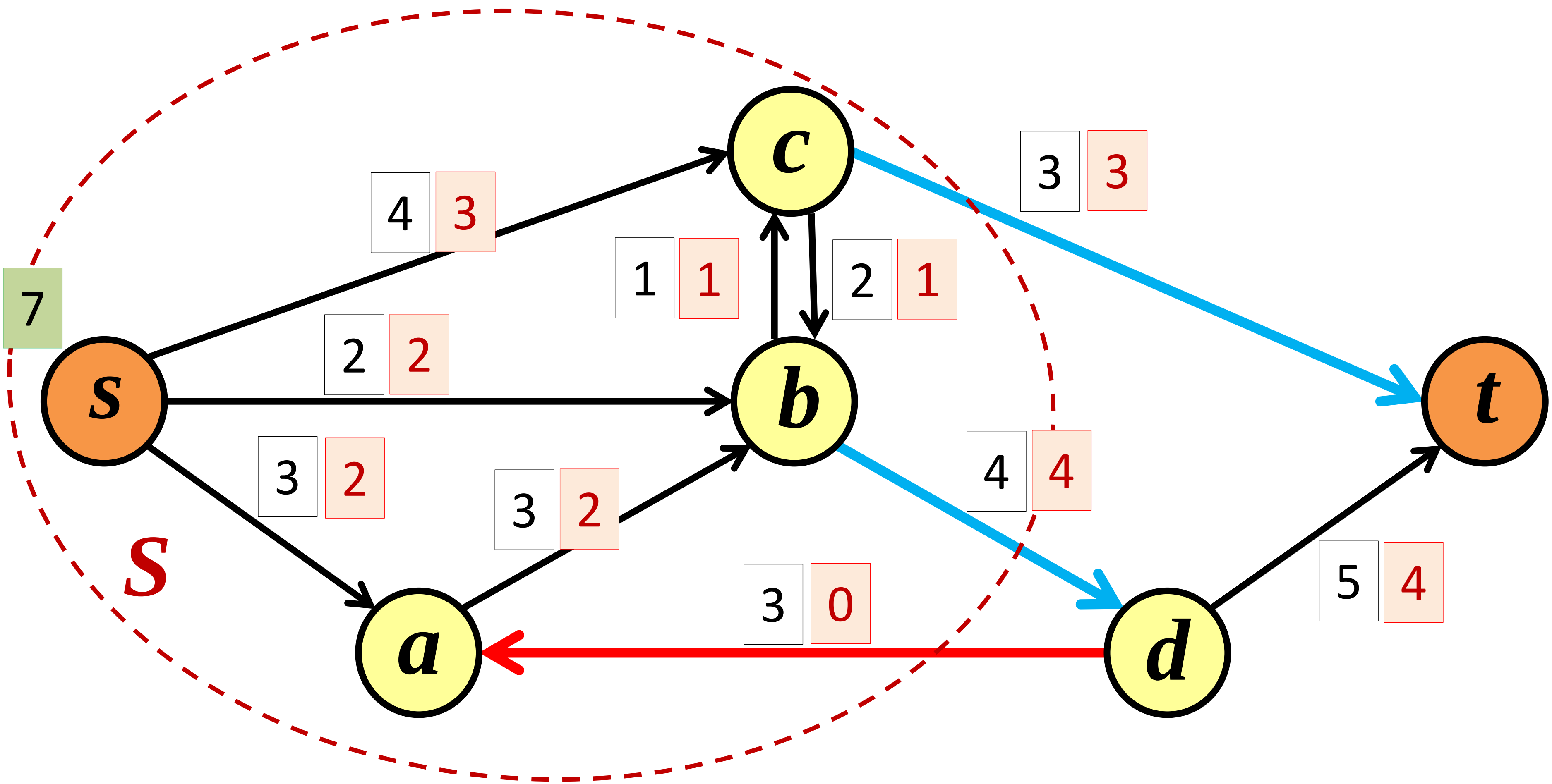
Fluxos em redes

Teorema: Seja f um fluxo **máximo** em uma rede D, e (S, S') um corte **mínimo** em D. Então $f(D) = c(S, S')$.



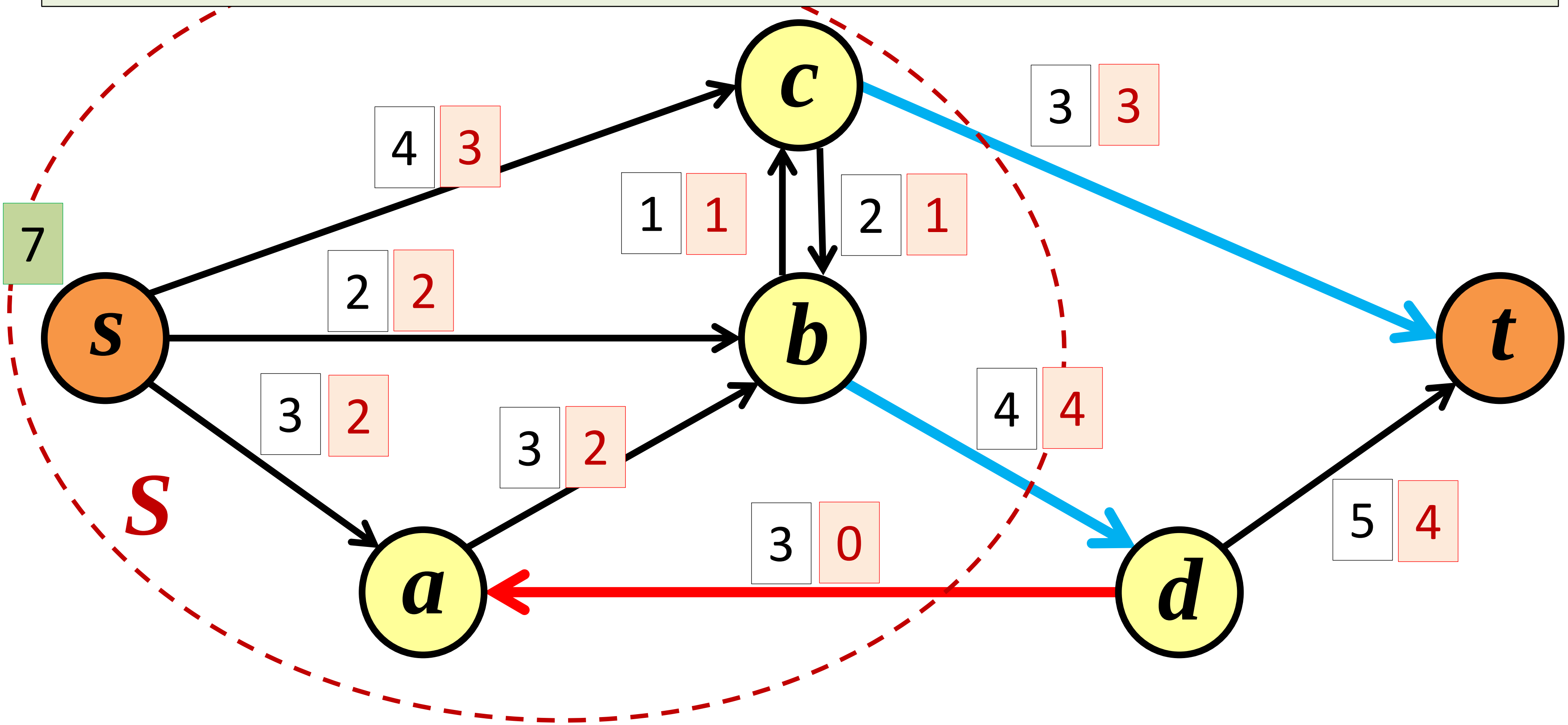
Fluxos em redes

$f\,(D) = 7 = c(S, S')$



Fluxos em redes

Corolário: Sejam f fluxo e (S, S') corte em D. Então: (S, S') é mínimo **sss** toda aresta de $(S, S')^+$ está saturada e toda aresta de $(S, S')^-$ tem fluxo zero.



Fluxos em redes

Algoritmo para determinar um fluxo máximo

Entrada: *Uma rede D*

f ← *fluxo inicial qualquer*

D' ← *rede residual de D relativa ao fluxo f*

enquanto D' *tem caminho aumentante* $e'_1\ e'_2\ e'_3\ \ldots\ e'_k$ ***faça***

    g ← *gargalo do caminho aumentante* $e'_1\ e'_2\ e'_3\ \ldots\ e'_k$

    para $j = 1, 2, \ldots, k$ ***faça***

        seja e_j a aresta de D correspondente a e'_j

        se e'_j *é aresta de aumento de fluxo* ***então*** $f(e_j) = f(e_j) + g$

        se e'_j *é aresta de redução de fluxo* ***então*** $f(e_j) = f(e_j) - g$

    fim-para

    D' ← *rede residual de D relativa ao novo fluxo f*

fim-enquanto

retornar f