Melhores momentos

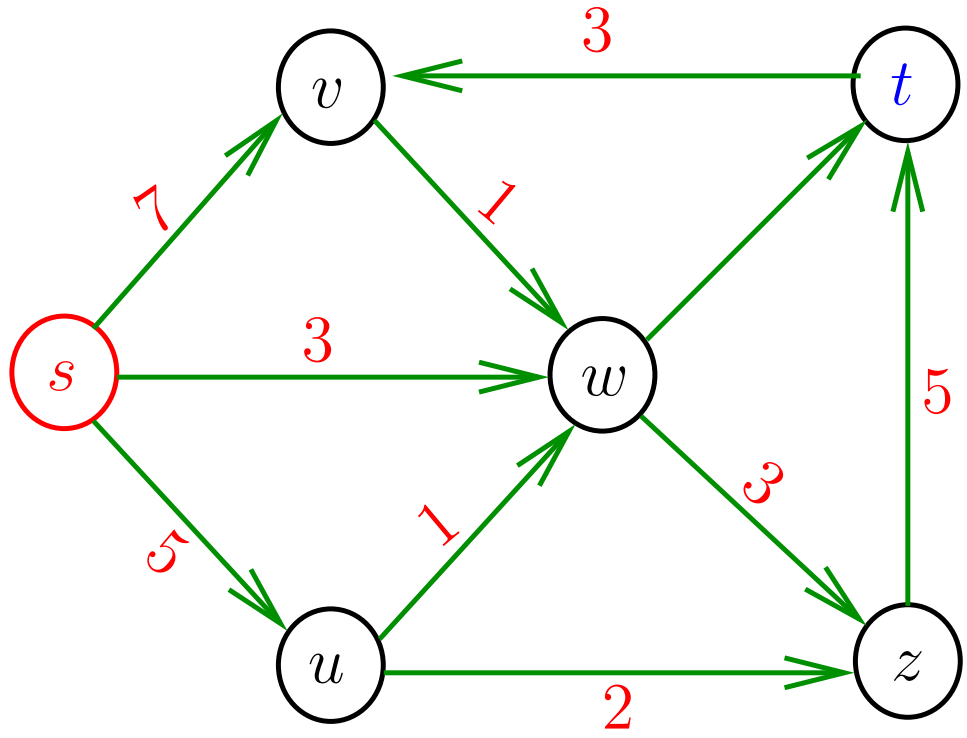
AULA 7

Problema

Problema do caminho mínimo sob custo não-negativo:
Dada uma rede (N, A, c) com função-custo $c : A \to \mathbb{Z}_{\geq}$ e um nó s, encontrar, para cada nó t, um caminho de custo mínimo de s a t.

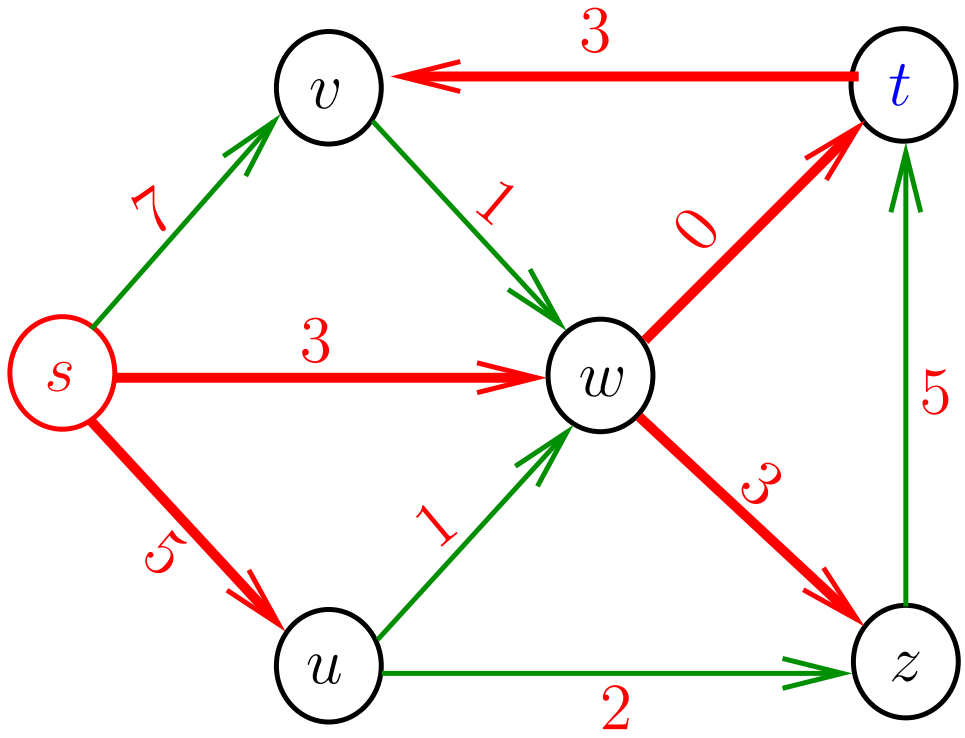
Problema

Problema do caminho mínimo sob custo não-negativo:
Dada uma rede (N, A, c) com função-custo $c : A \to \mathbb{Z}_{\geq}$ e um nó s, encontrar, para cada nó t, um caminho de custo mínimo de s a t.
Entra:



Problema

Problema do caminho mínimo sob custo não-negativo:
Dada uma rede (N, A, c) com função-custo $c : A \to \mathbb{Z}_{\geq}$ e um nó s, encontrar, para cada nó t, um caminho de custo mínimo de s a t.
Sai:



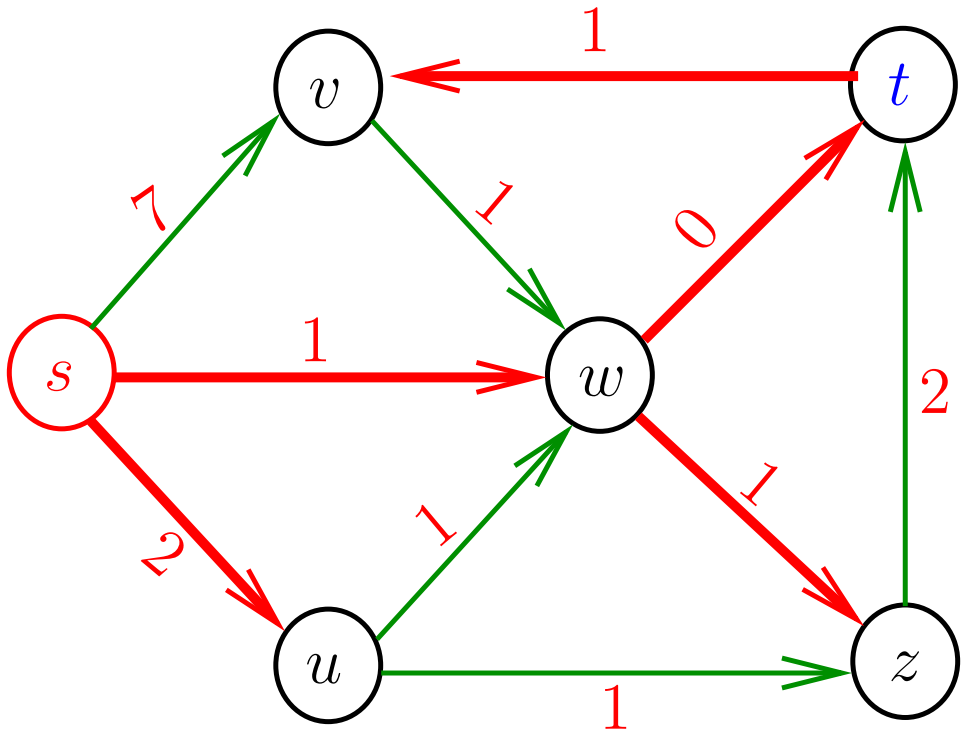
Condição de inexistência

Como é possível provar que um dado caminho de s a t tem custo mínimo?

Condição de inexistência

Como é possível provar que um dado caminho de s a t tem custo mínimo?

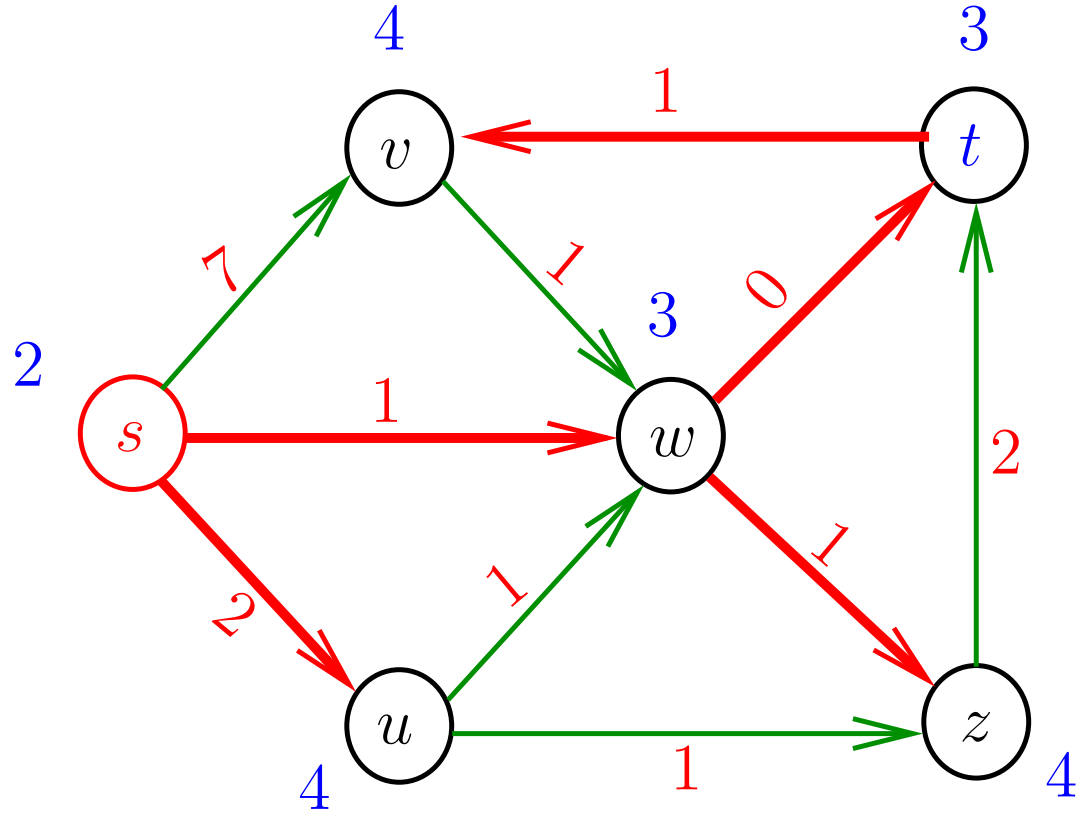
Entra:



Condição de inexistência

Como é possível provar que um dado caminho de s a t tem custo mínimo?

Sai: um potencial apropriado



c-potencial

Um c-potencial é qualquer função y de N em $\mathbb{Z}$ tal que

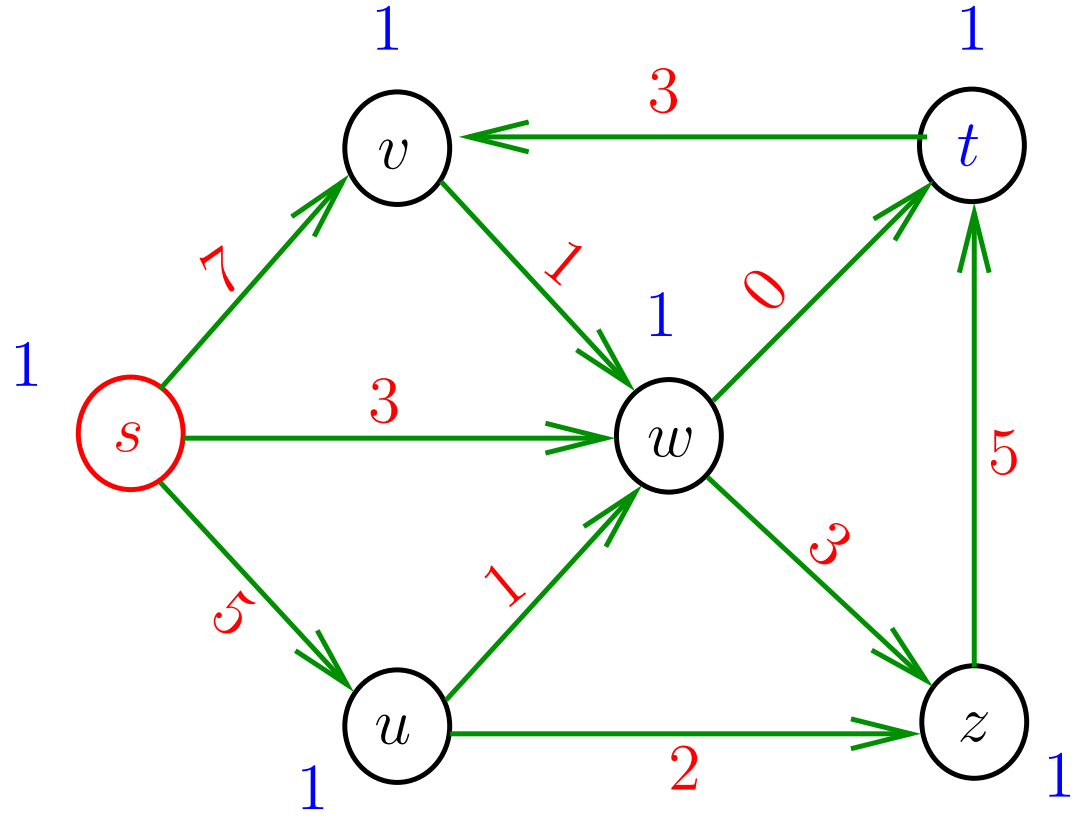
$$y(j) - y(i) \leq c(ij) \quad \text{para todo arco } ij.$$

c-potencial

Um c-potencial é qualquer função y de N em $\mathbb{Z}$ tal que

$$y(j) - y(i) \leq c(ij) \quad \text{para todo arco } ij.$$

Exemplo 1: c-potencial constante (sem graça...)

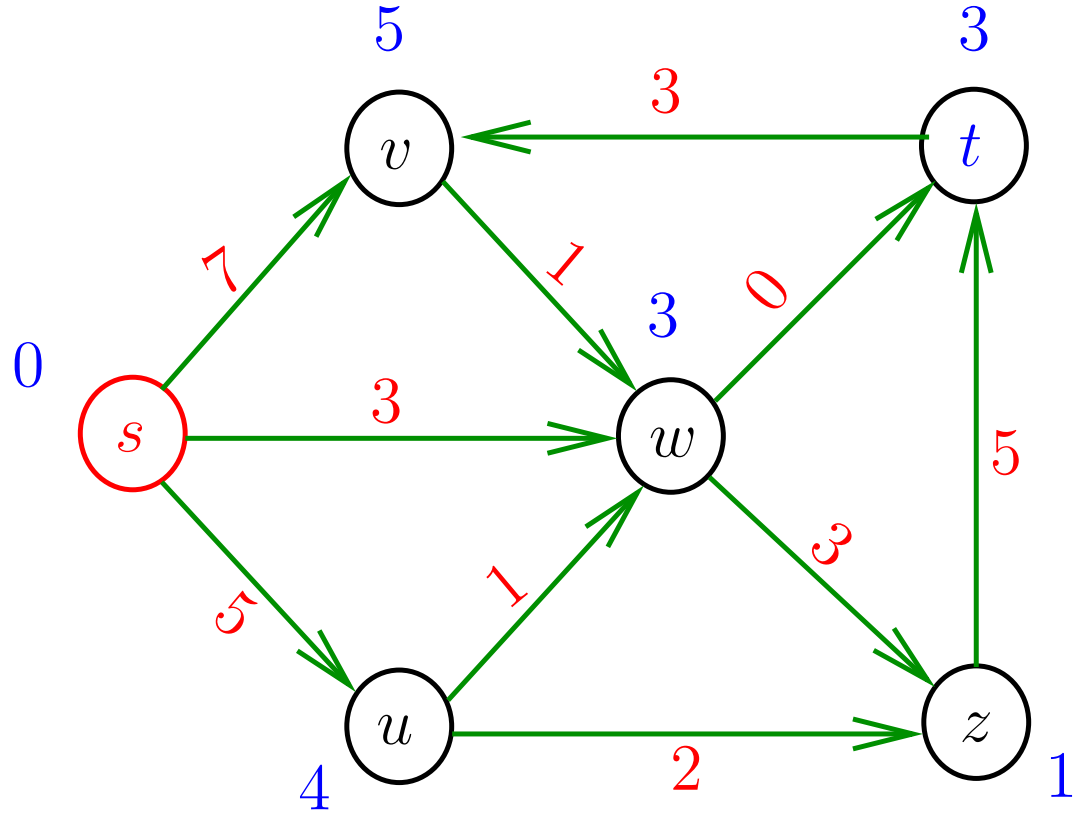


c-potencial

Um c-potencial é qualquer função y de N em $\mathbb{Z}$ tal que

$$y(j) - y(i) \leq c(ij) \quad \text{para todo arco } ij.$$

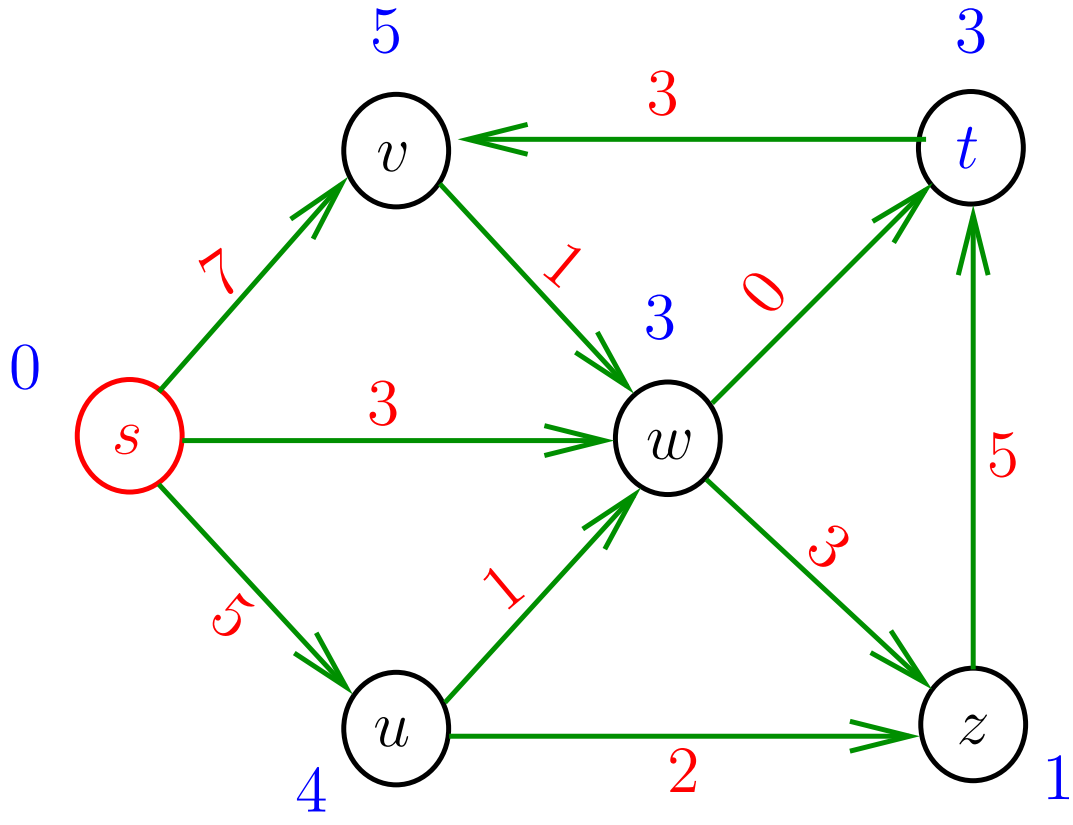
Exemplo 2: um c-potencial menos "bobo"



Propriedade de c-Potenciais

(Lema da dualidade.) Se P é um passeio de s a t e y é um c-potencial então

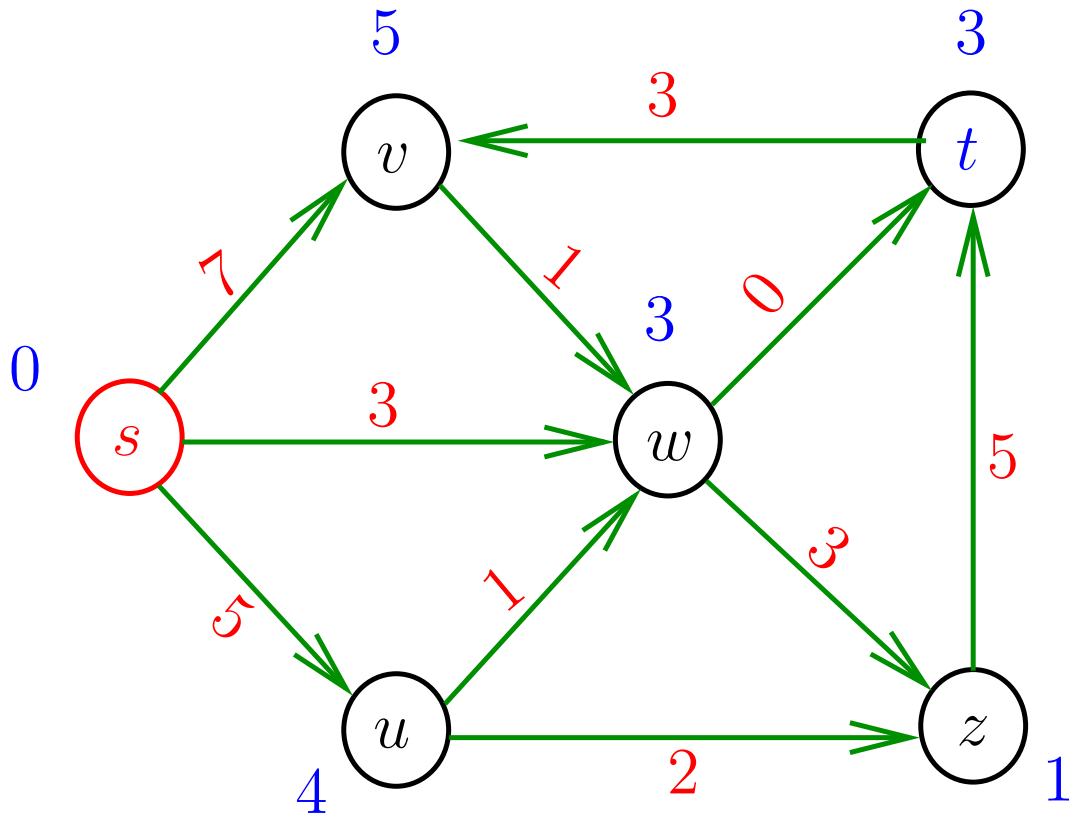
$$c(P) \geq y(t) - y(s).$$



Propriedade de c-Potenciais

Para mostrar que não existe um caminho de s a t de comprimento $< \lambda$ basta exibir um 1-potencial tal que

$$y(t) - y(s) \geq \lambda.$$

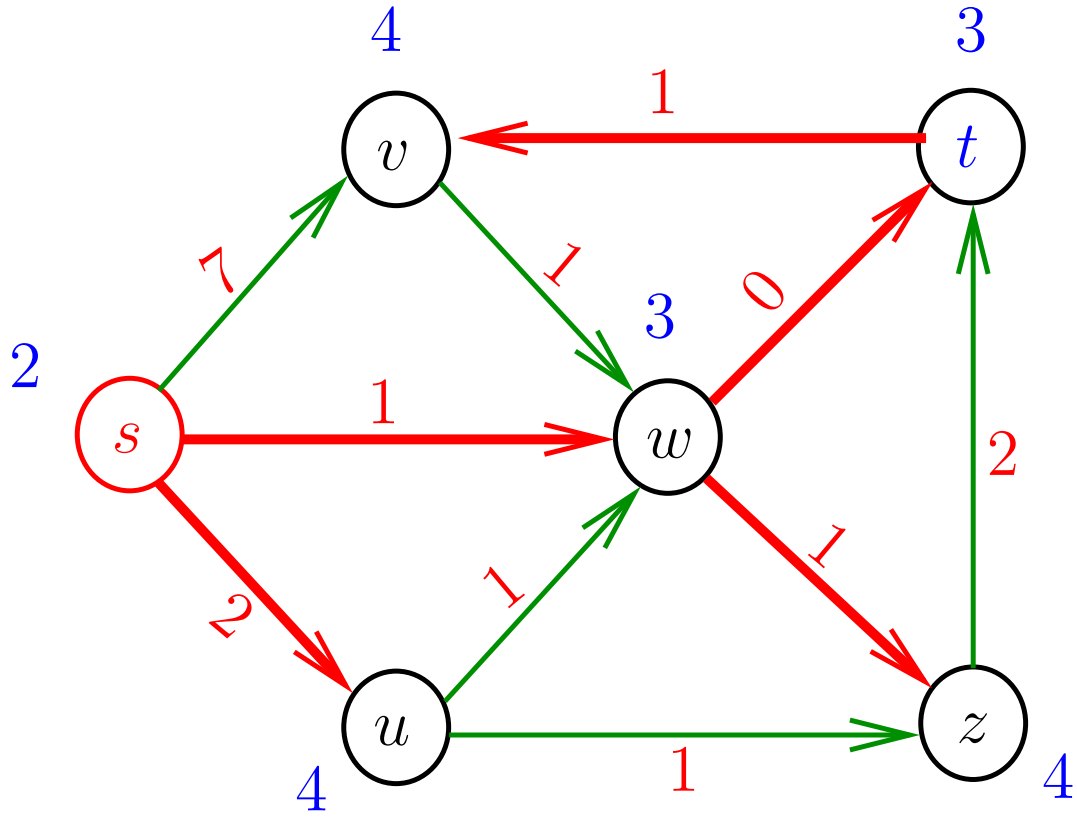


Conseqüência

Se P é um caminho de s a t e y é um c-potencial tais que

$$c(P) = y(t) - y(s),$$

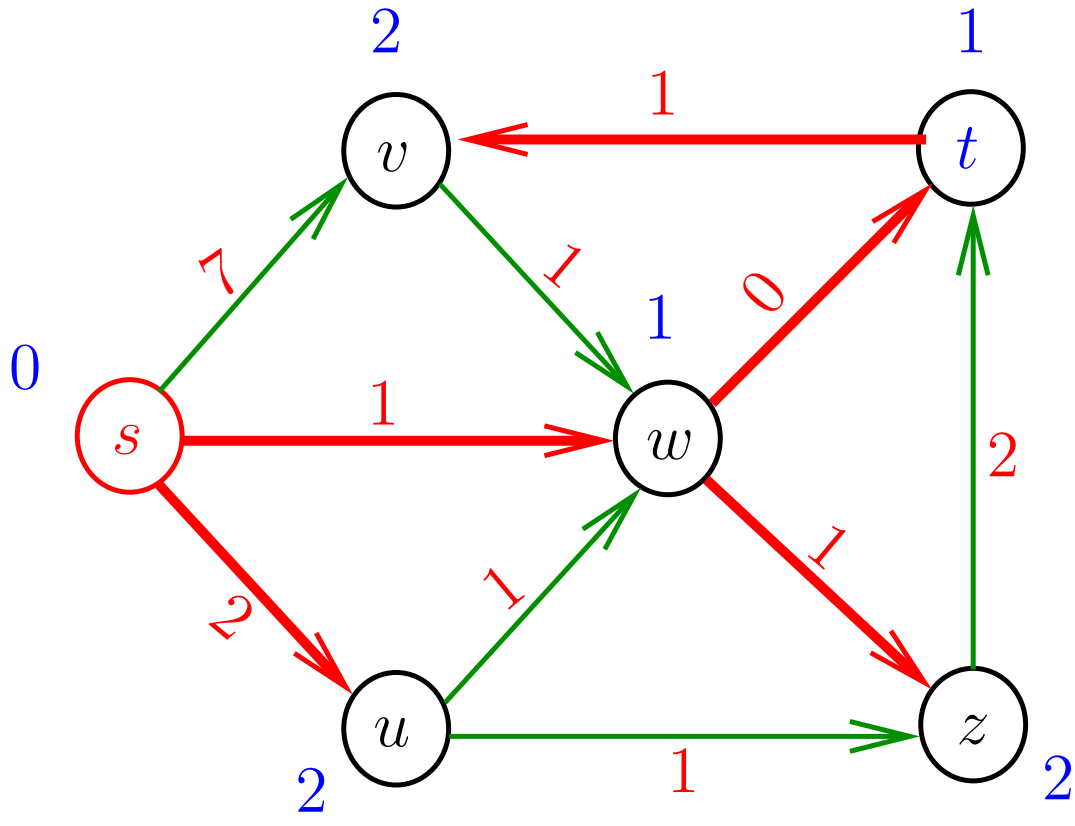
então P é uma caminho mínimo e y é um c-potencial tal que o valor de $y(t) - y(s)$ é máximo.



Potenciais ótimos

Um c-potencial y é $(s, *)$-ótimo se, para todo nó t, existe um caminho P de s a t tal que

$$c(P) = y(t) - y(s),$$



AULA 8

Caminhos de custo mínimo

PF 8.1, 8.2, 8.4

Algoritmo genérico

Recebe uma rede (N, A, c) com função custo $c : A \to \mathbb{Z}_{\geq}$ e um nó s e devolve uma função-predecessor π e uma função-potencial y com a seguinte propriedade: para todo nó t, a função π determina um caminho P de s a t tal que $c(P) = y(t) - y(s)$.

FORD (N, A, c, s) $\quad \triangleright\ c \geq 0$

1 **para cada** i em N **faça**
2 $\quad y(i) \leftarrow nC + 1 \quad \triangleright\ C := \max\{c(ij) : ij \in A\}$
3 $\quad \pi(i) \leftarrow \text{NIL}$
4 $y(s) \leftarrow 0$
5 **enquanto** $y(j) > y(i) + c(ij)$ **para** algum $ij \in A$ **faça**
6 $\quad y(j) \leftarrow y(i) + c(ij)$
7 $\quad \pi(j) \leftarrow i$
8 **devolva** π e y

Invariantes

Na linha 5, antes da verificação da condição "$y(j) > y(i) + c(ij) \ldots$" valem as seguintes invariantes:

(i0) para cada arco pq no grafo de predecessores tem-se $y(q) - y(p) \geq c(pq)$;

(i1) $\pi(s) = \text{NIL}$ e $y(s) = 0$;

(i2) para cada nó v distinto de s, $y(v) < nC + 1 \Leftrightarrow \pi(v) \neq \text{NIL}$;

(i3) para cada nó v, se $\pi(v) \neq \text{NIL}$ então existe um caminho de s a v no grafo de predecessores.

Correção

Início da última iteração:

- y é um c-potencial
- Se $y(t) < nC + 1$ então, por (i2), vale que $\pi(t) \neq \text{NIL}$. Logo, de (i3), segue que existe um st-caminho P no grafo de predecessores. Desta forma, (i0) e (i1) implicam que

$$c(P) \geq y(t) - y(s) = y(t).$$

  Da propriedade dos c-potencias, concluímos que P é um st-caminho (de custo) mínimo.
- Se $y(t) = nC + 1$, então (i1) implica que $y(t) - y(s) = nC + 1$ e da propriedade dos c-potencias concluímos que não existe caminho de s a t no grafo.

Conclusão: o algoritmo faz o que promete.

Conclusão

Da propriedade dos c-potenciais (lema da dualidade) e da correção do algoritmo FORD concluímos o seguinte:

(Teorema dualidade) Se s e t são nós de uma rede (N, A, c) com função custo $c : A \to \mathbb{Z}_{\geq}$ e t está ao alcance de s então

$$\min\{c(P) : P \text{ é um } st\text{-caminho}\}$$
$$= \max\{y(t) - y(s) : y \text{ é um } c\text{-potencial}\}.$$

Consumo de tempo

O número de execuções do bloco de linhas 5–7 é

$$< n(nC + 1) = n^2C + n.$$

linha	consumo de todas as execuções da linha
1-3	$\mathrm{O}(n)$
4	$\mathrm{O}(1)$
5	$(n^2C + n)\mathrm{O}(m) = \mathrm{O}(n^2m(C + 1))$
6-7	$(n^2C + n)\mathrm{O}(1) = \mathrm{O}(n^2(C + 1))$
8	$\mathrm{O}(n)$
total	$2\,\mathrm{O}(n) + \mathrm{O}(1) + \mathrm{O}(n^2m(C + 1)) + \mathrm{O}(n^2(C + 1))$ $= \mathrm{O}(n^2m(C + 1))$

Conclusão

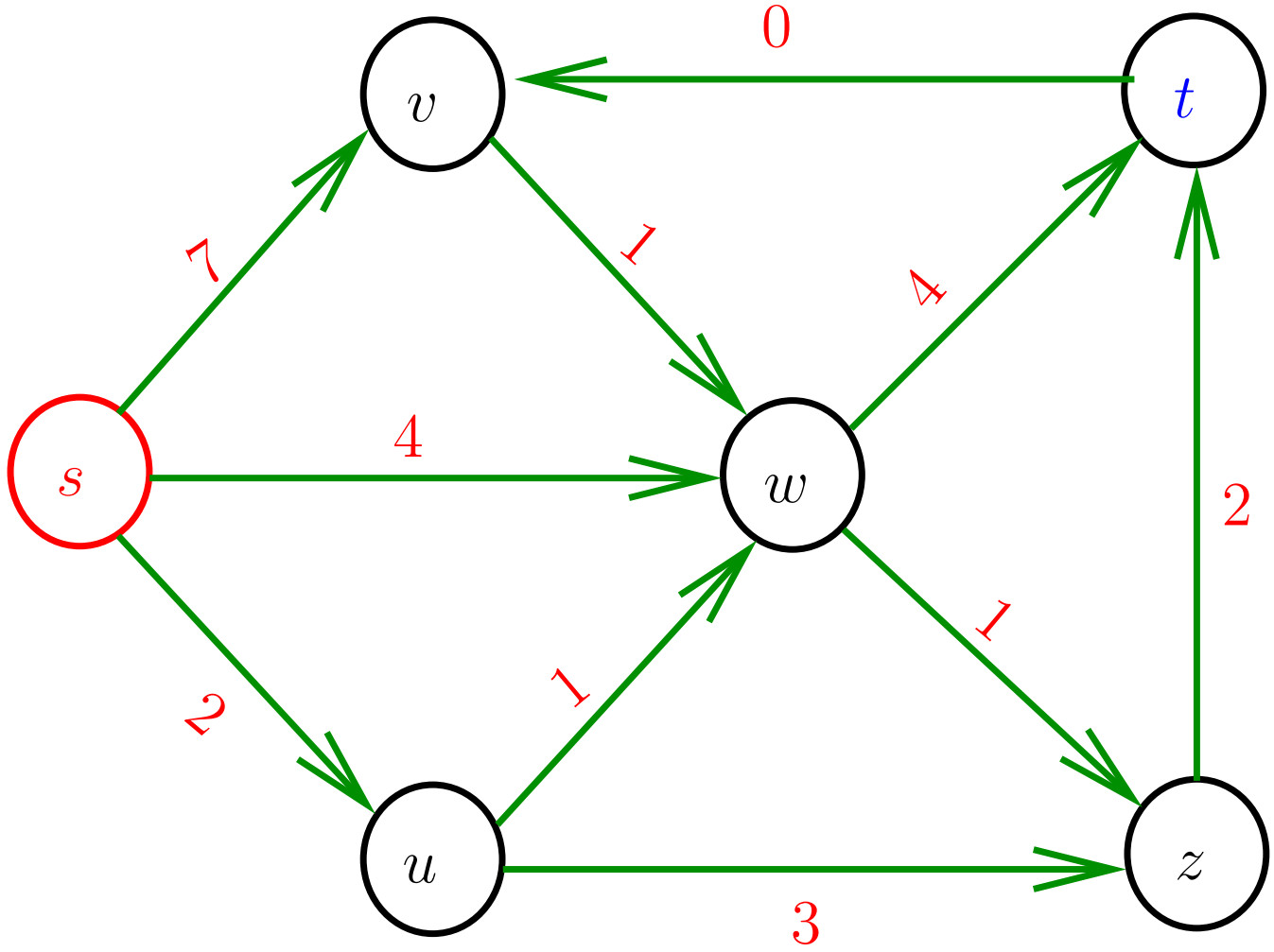
O consumo de tempo do algoritmo CAMINHO-CURTO-GENÉRICO é $\mathrm{O}(n^2 m(C+1))$.

Este consumo de tempo não é polinomial.

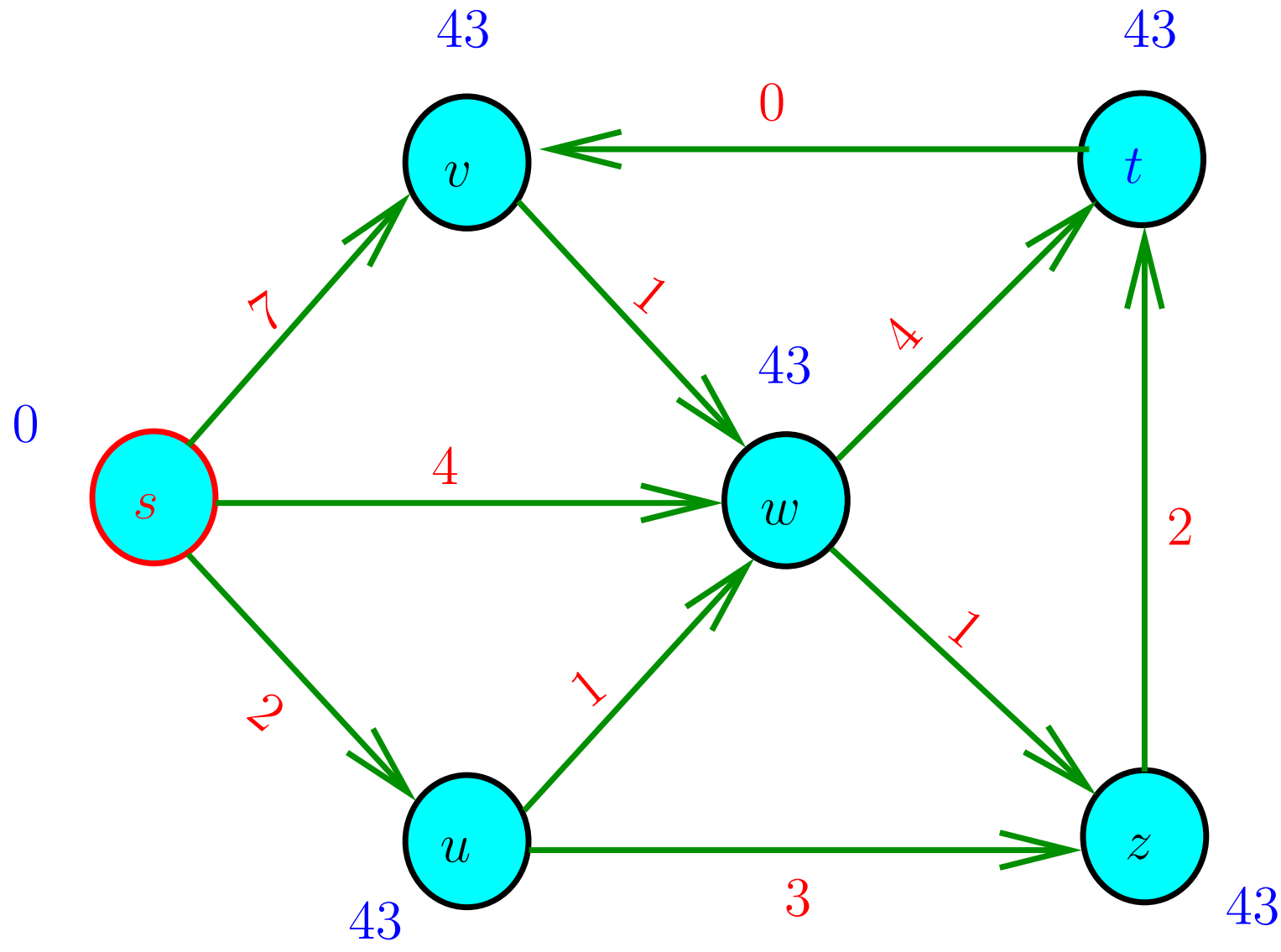
Implementação do algoritmo Ford

```
DIJKSTRA (N, A, c, s)      ▷ c ≥ 0
 1   para cada i em N faça
 2       y(i) ← nC + 1      ▷ nC + 1 faz o papel de ∞
 3       π(i) ← NIL
 4   y(s) ← 0
 5   Q ← N      ▷ Q func. como uma fila com prioridades
 6   enquanto Q ≠ ⟨⟩ faça
 7       retire de Q um nó i com y(i) mínimo
 8       para cada ij em A(i) faça
 9           se y(j) > y(i) + c(ij) então
10               y(j) ← y(i) + c(ij)
11               π(j) ← i
12   devolva π e y
```

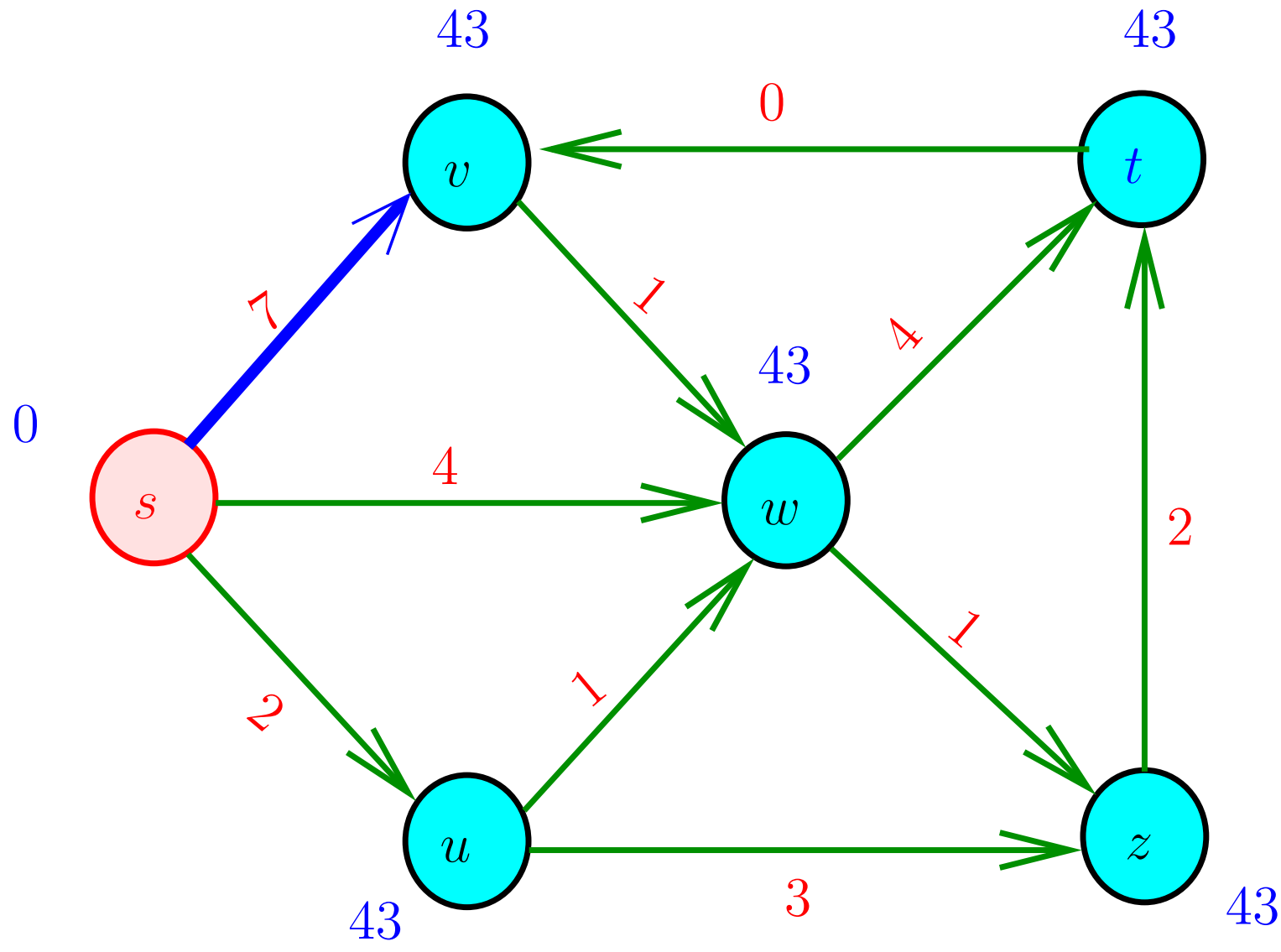
Simulação



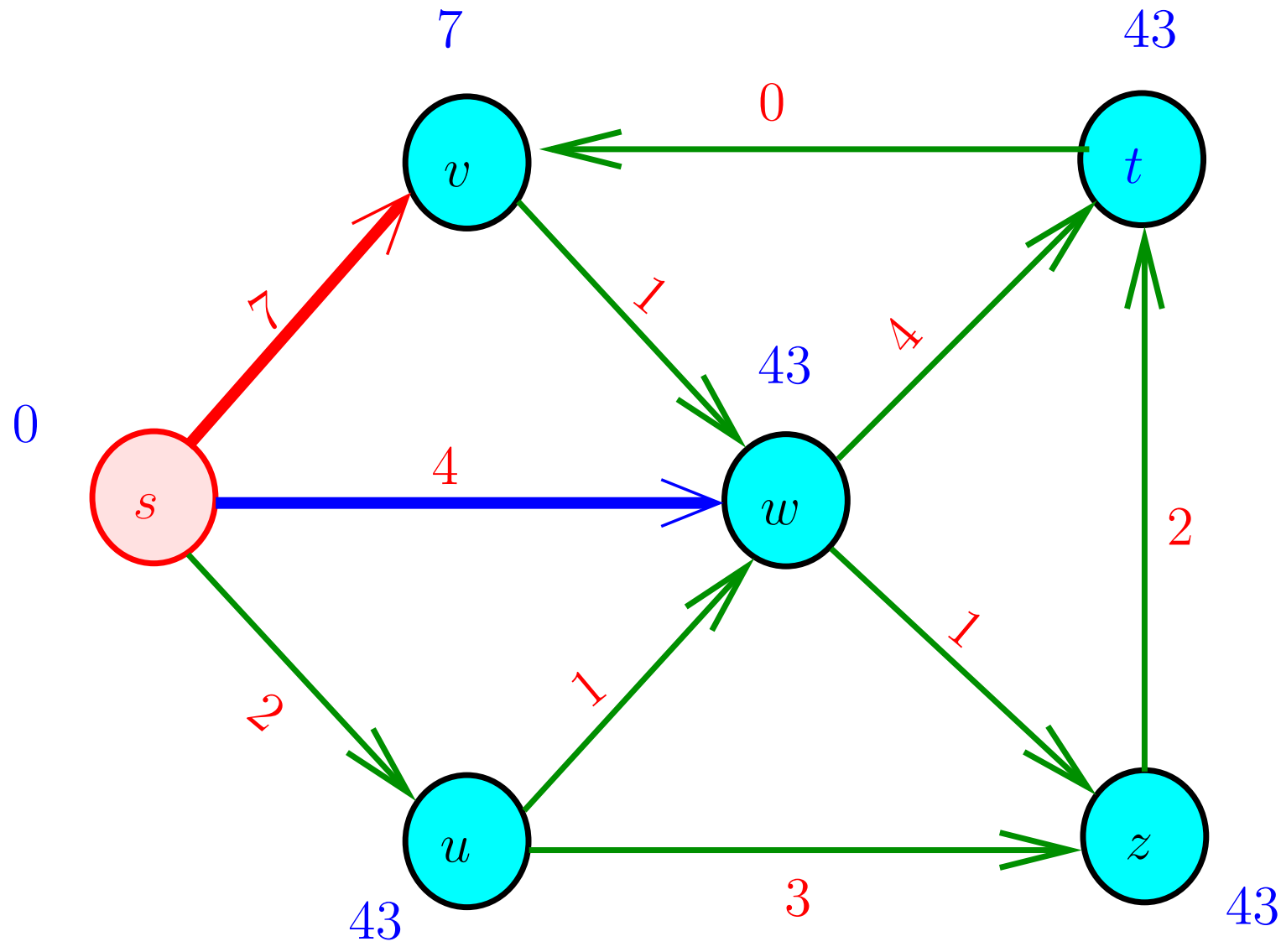
Simulação



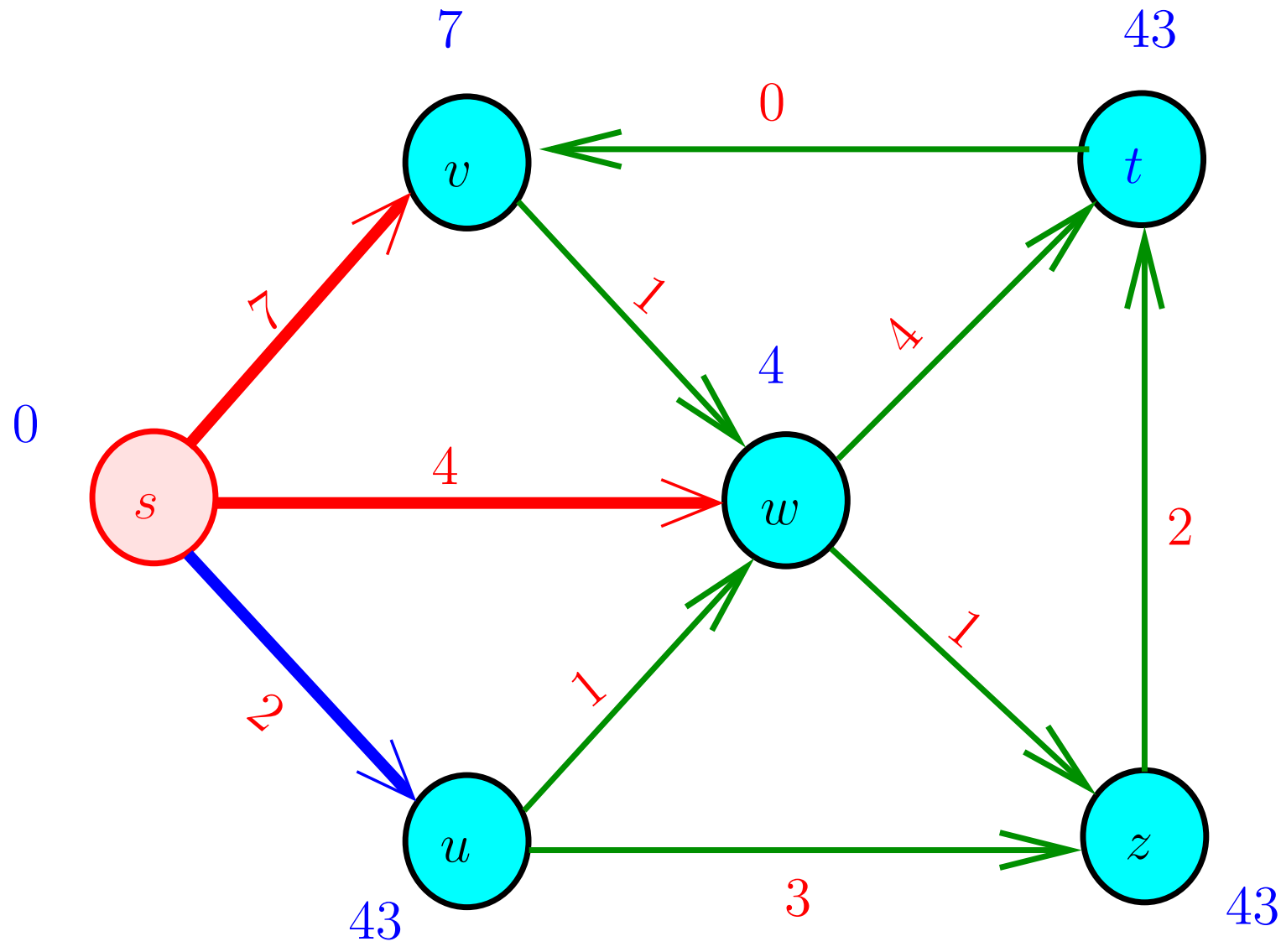
Simulação



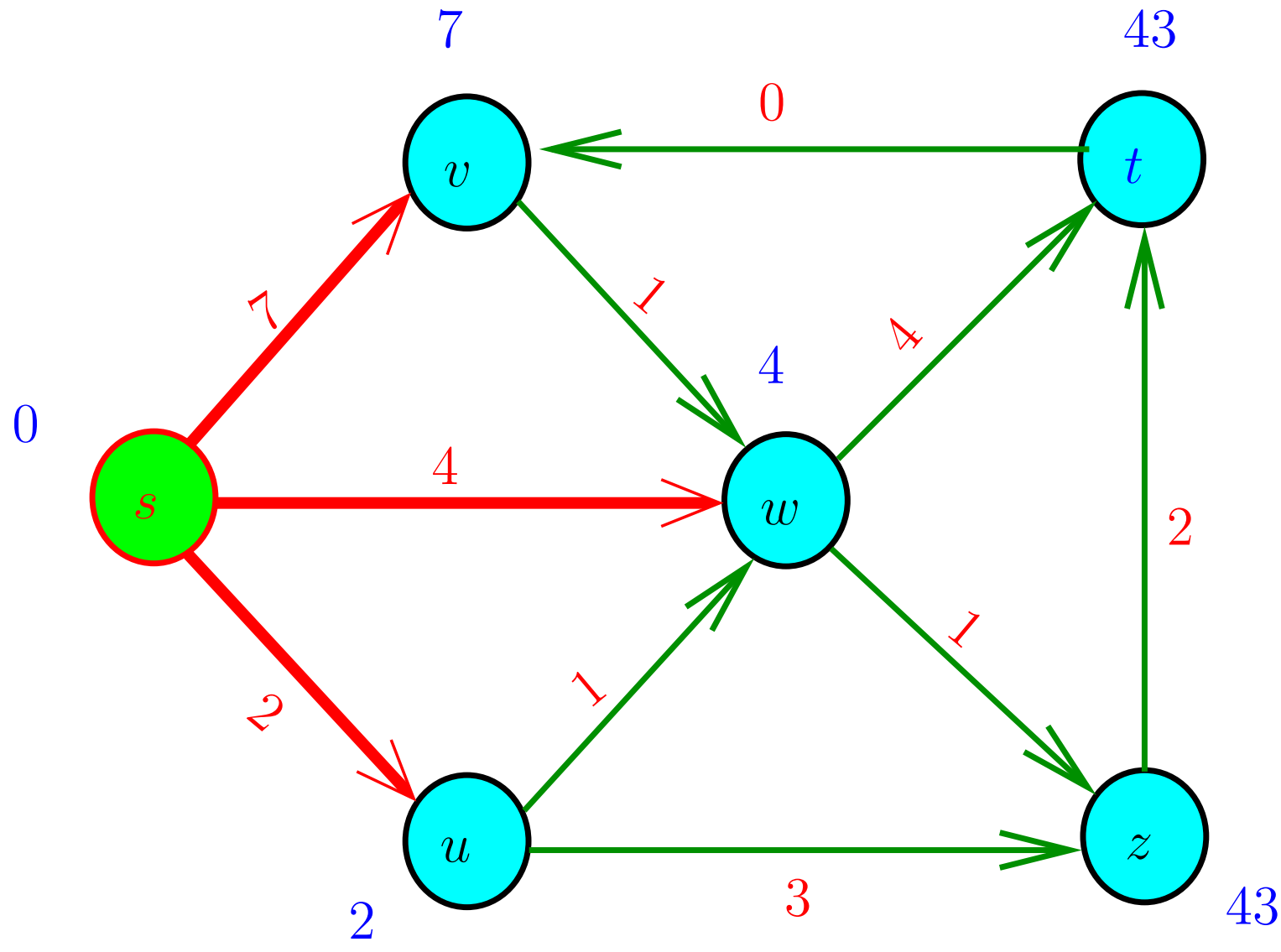
Simulação



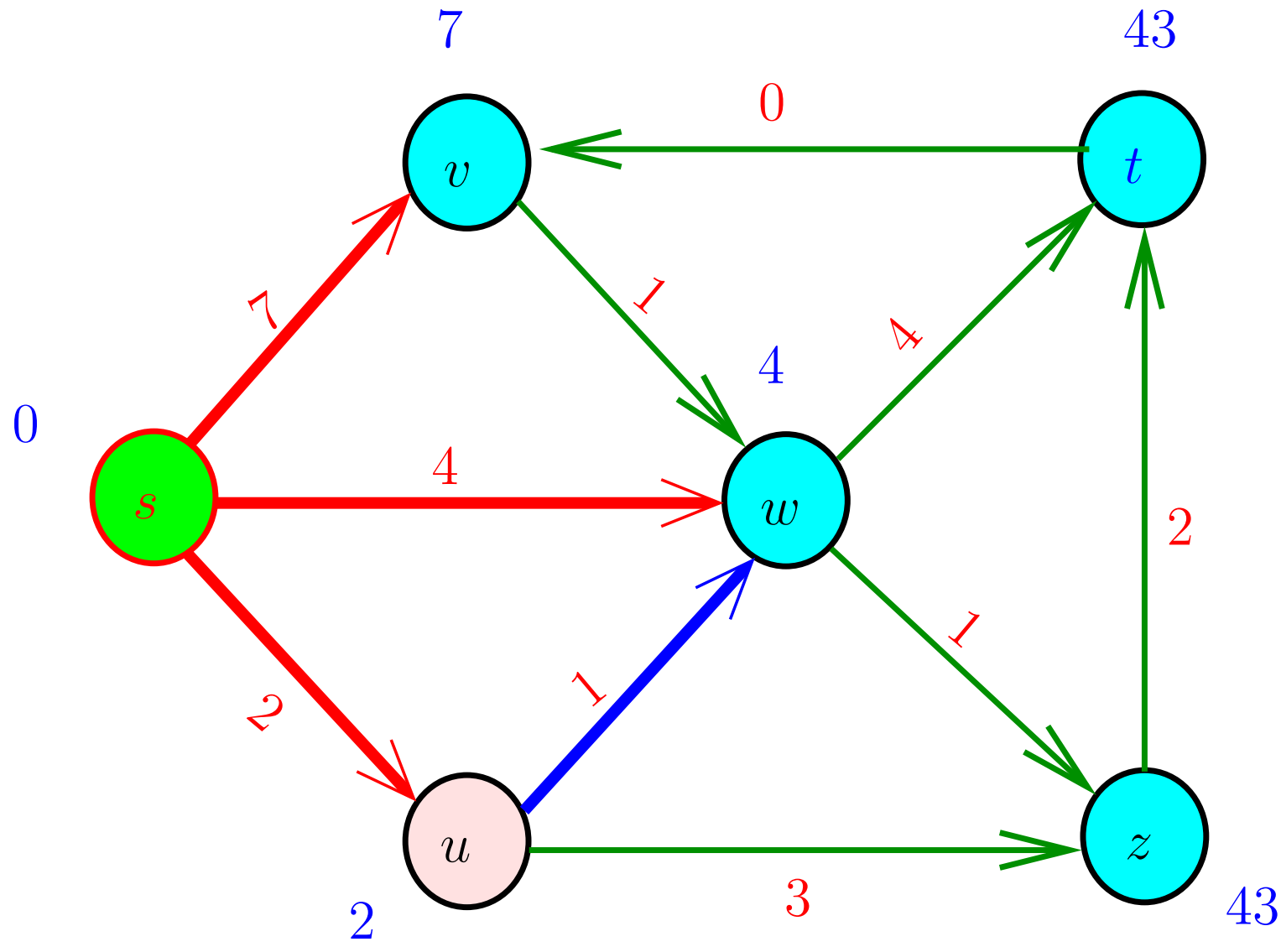
Simulação



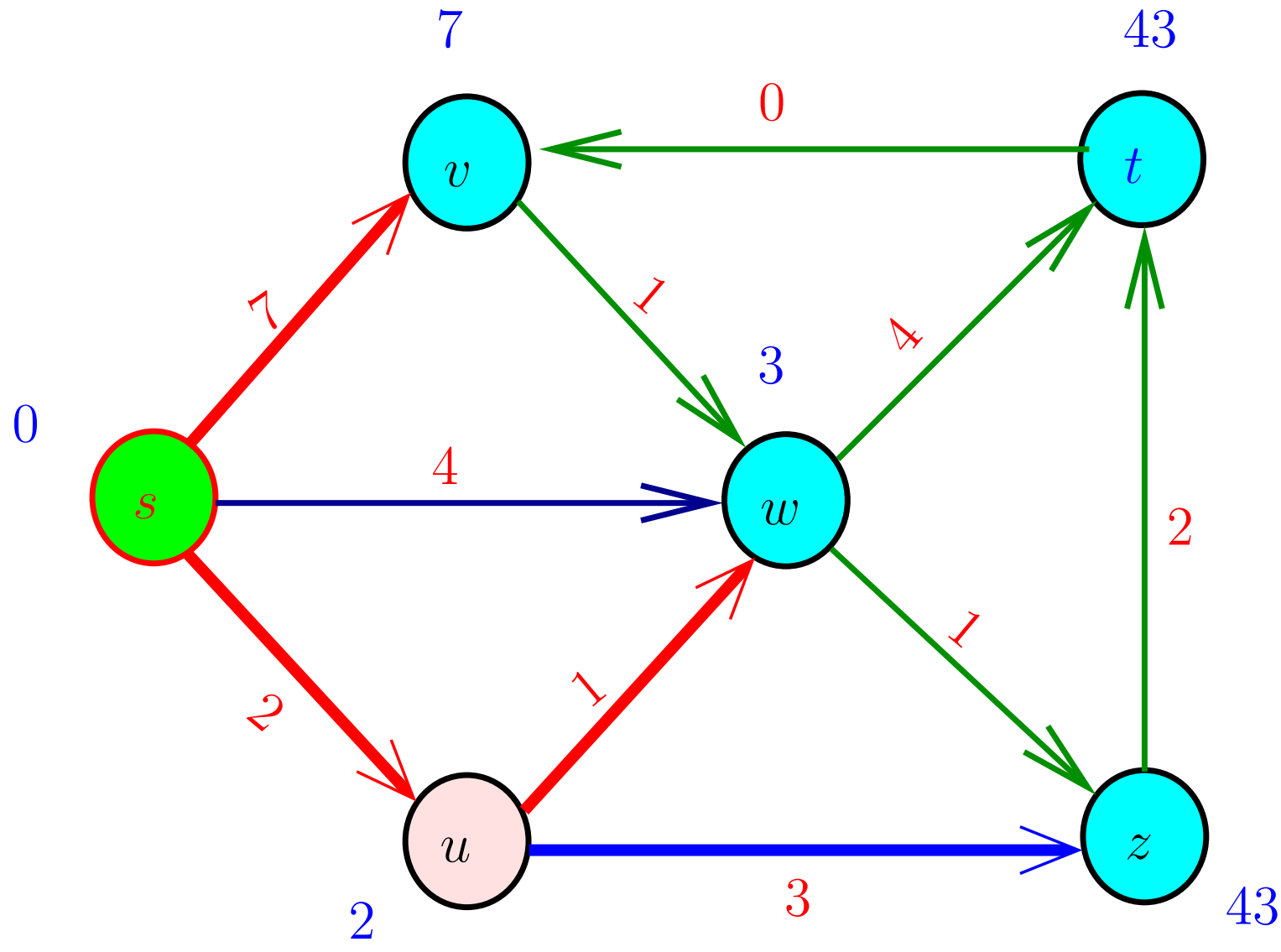
Simulação



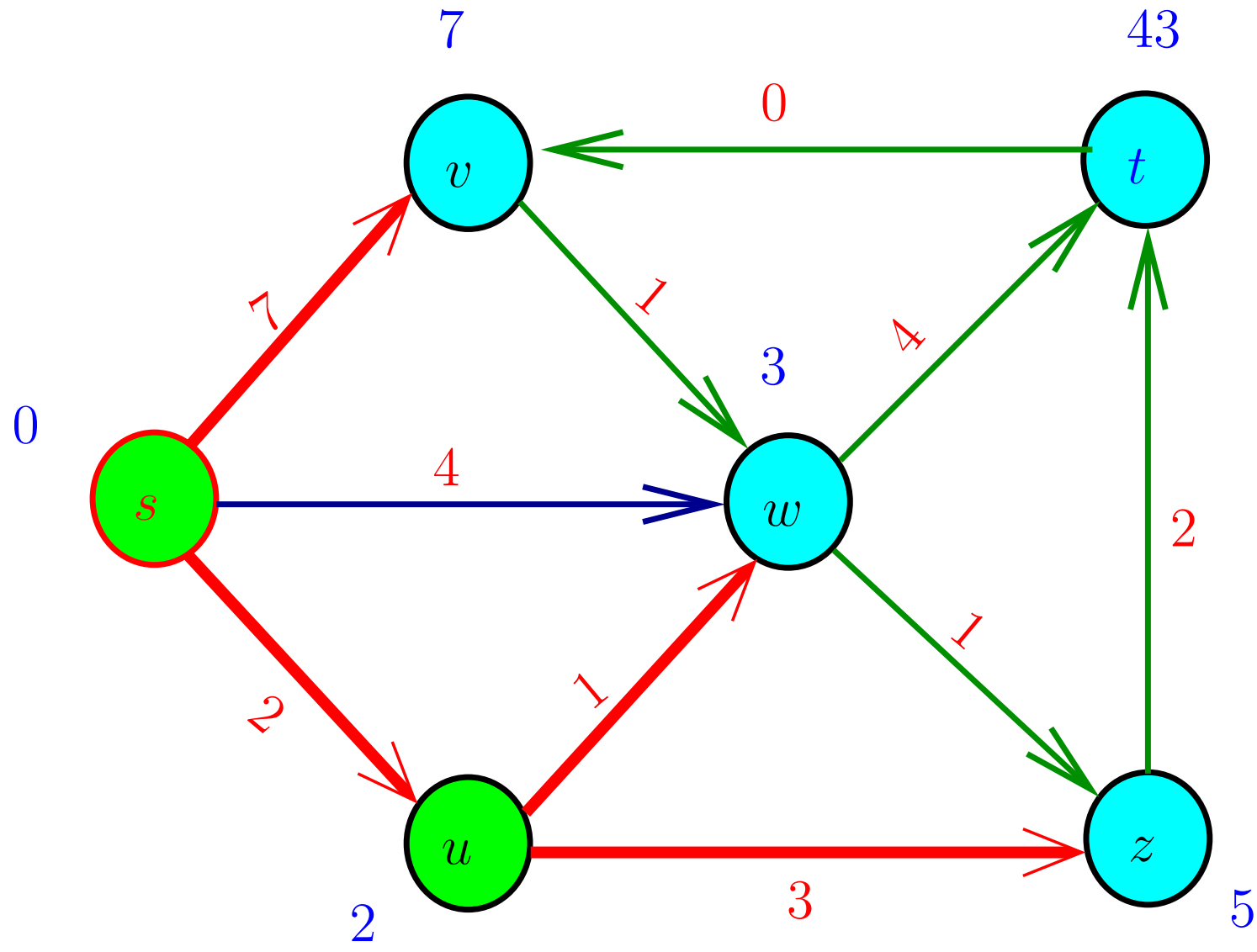
Simulação



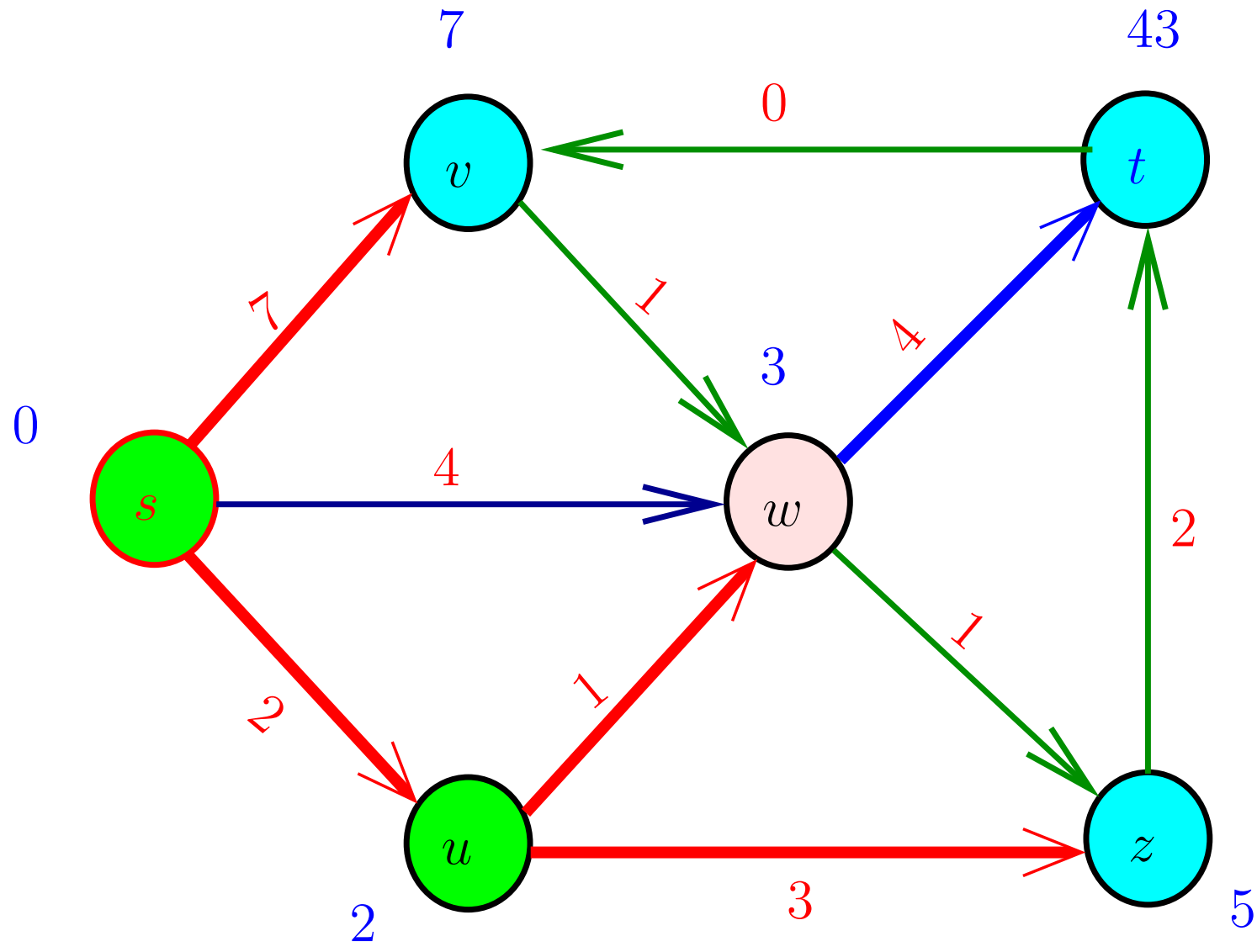
Simulação



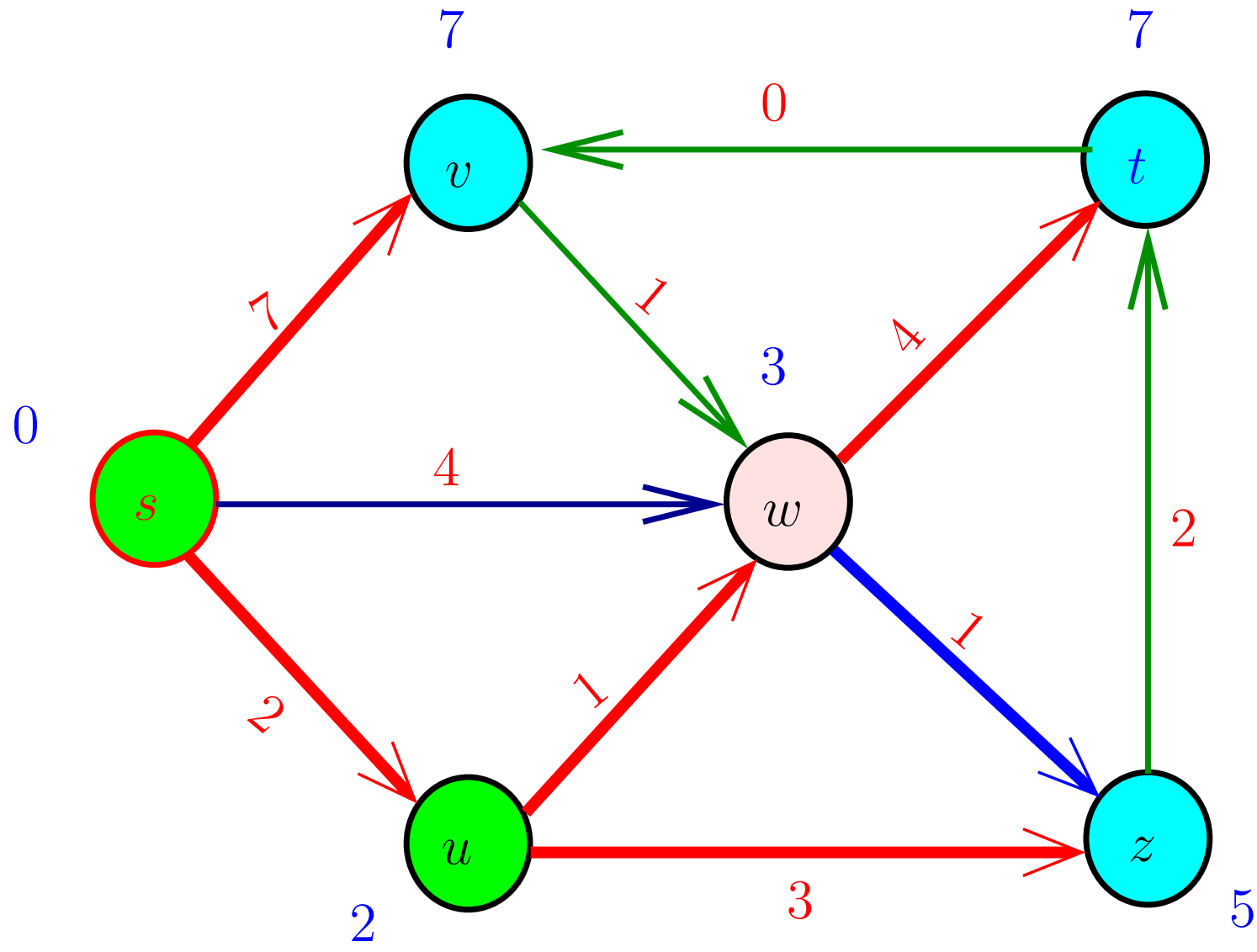
Simulação



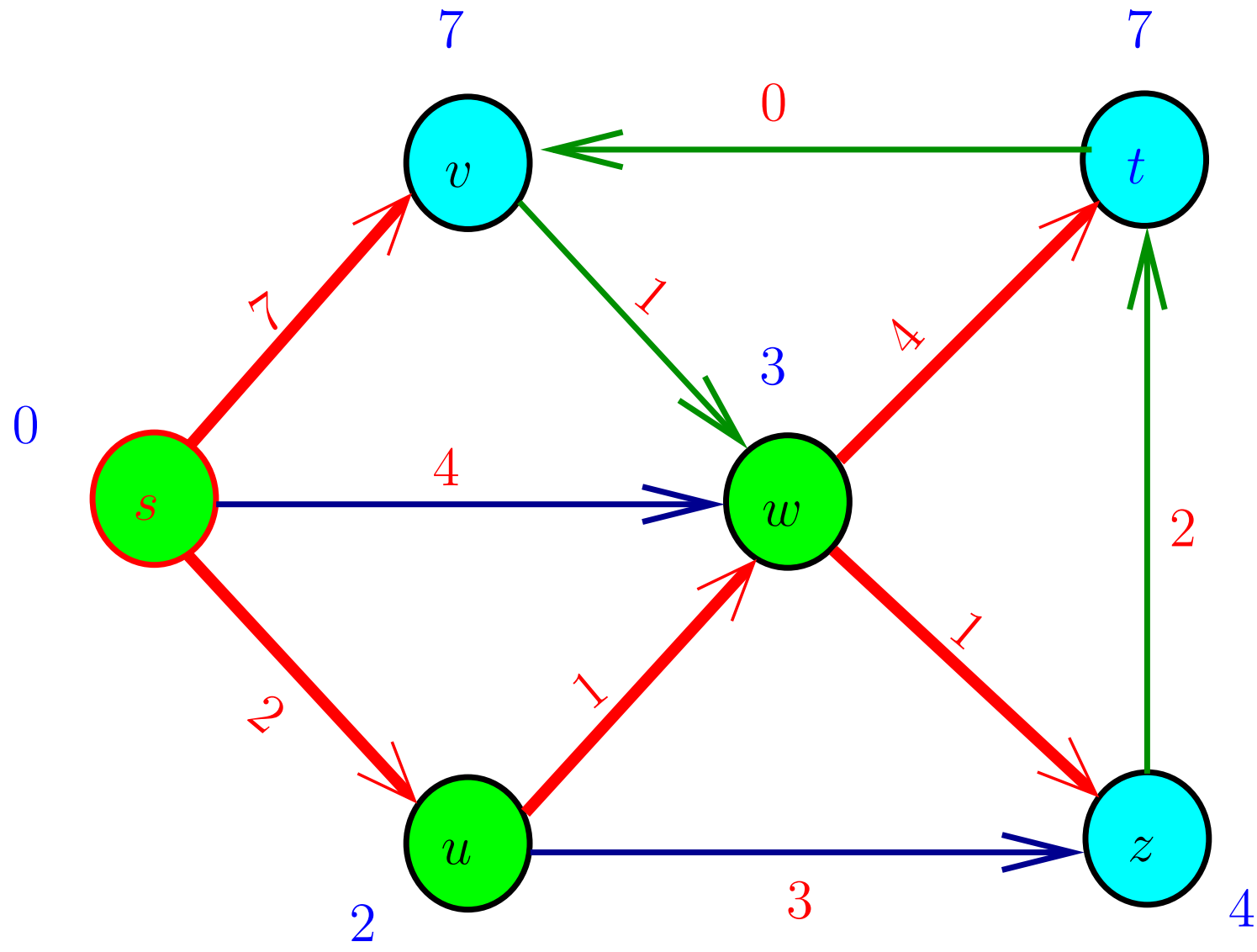
Simulação



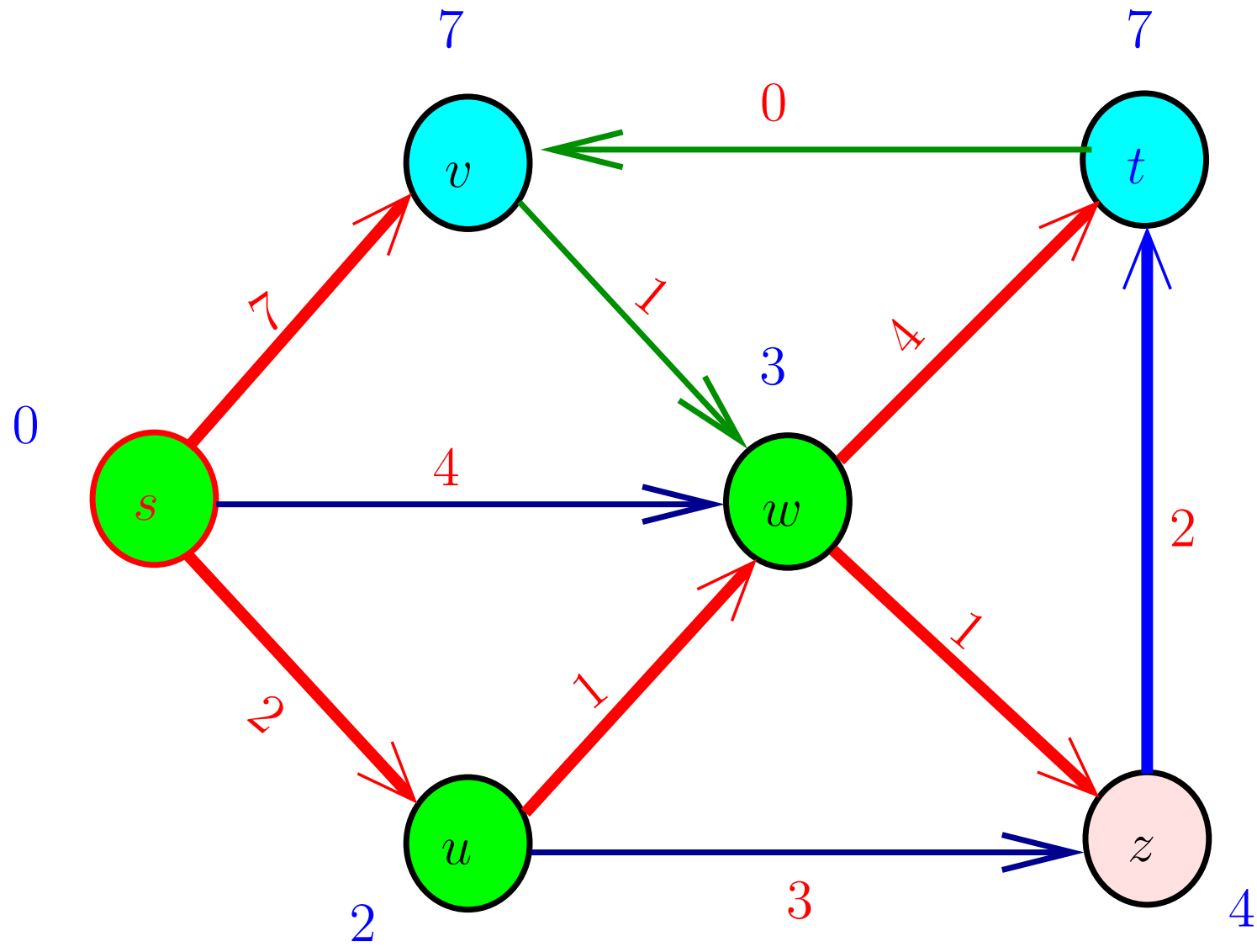
Simulação



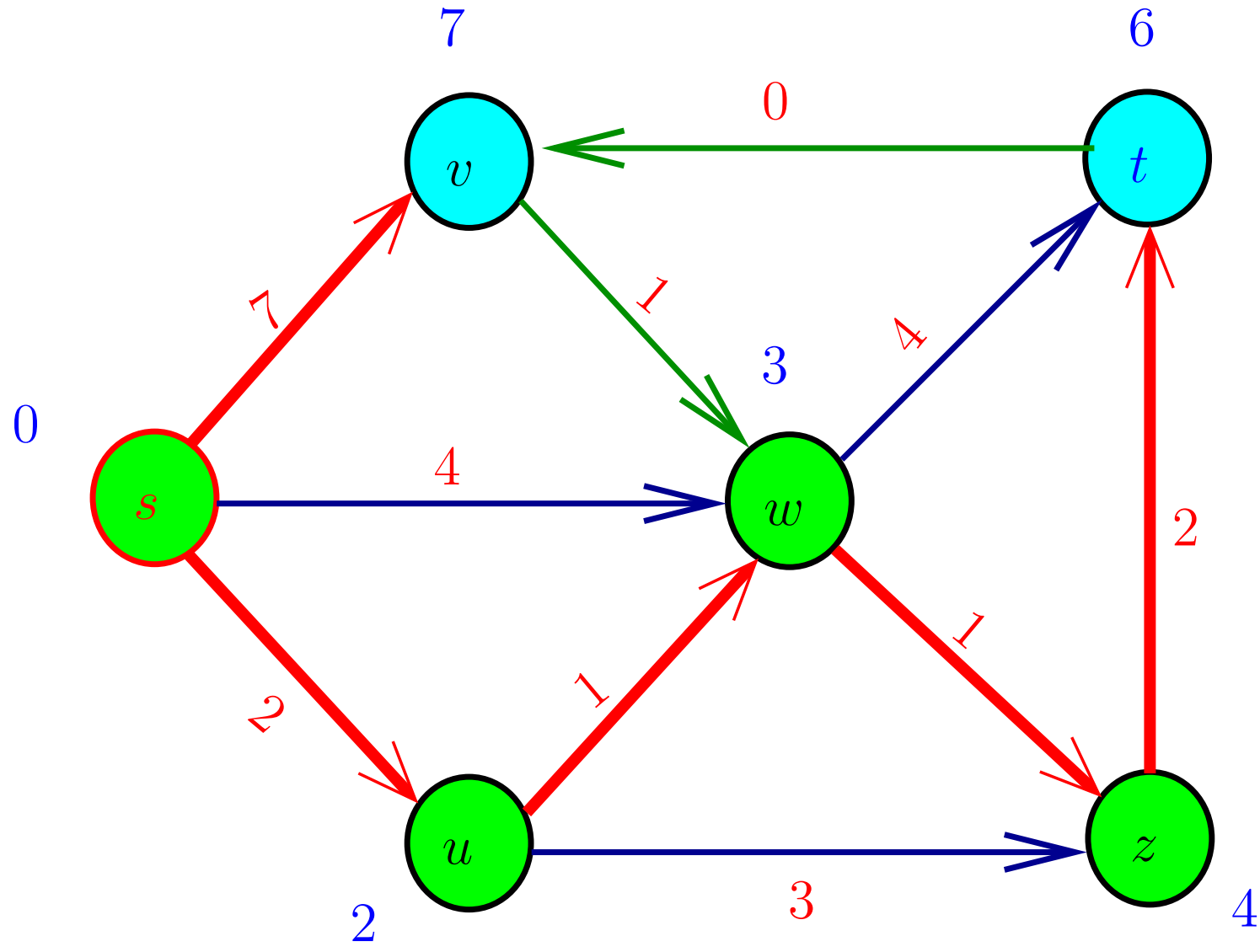
Simulação



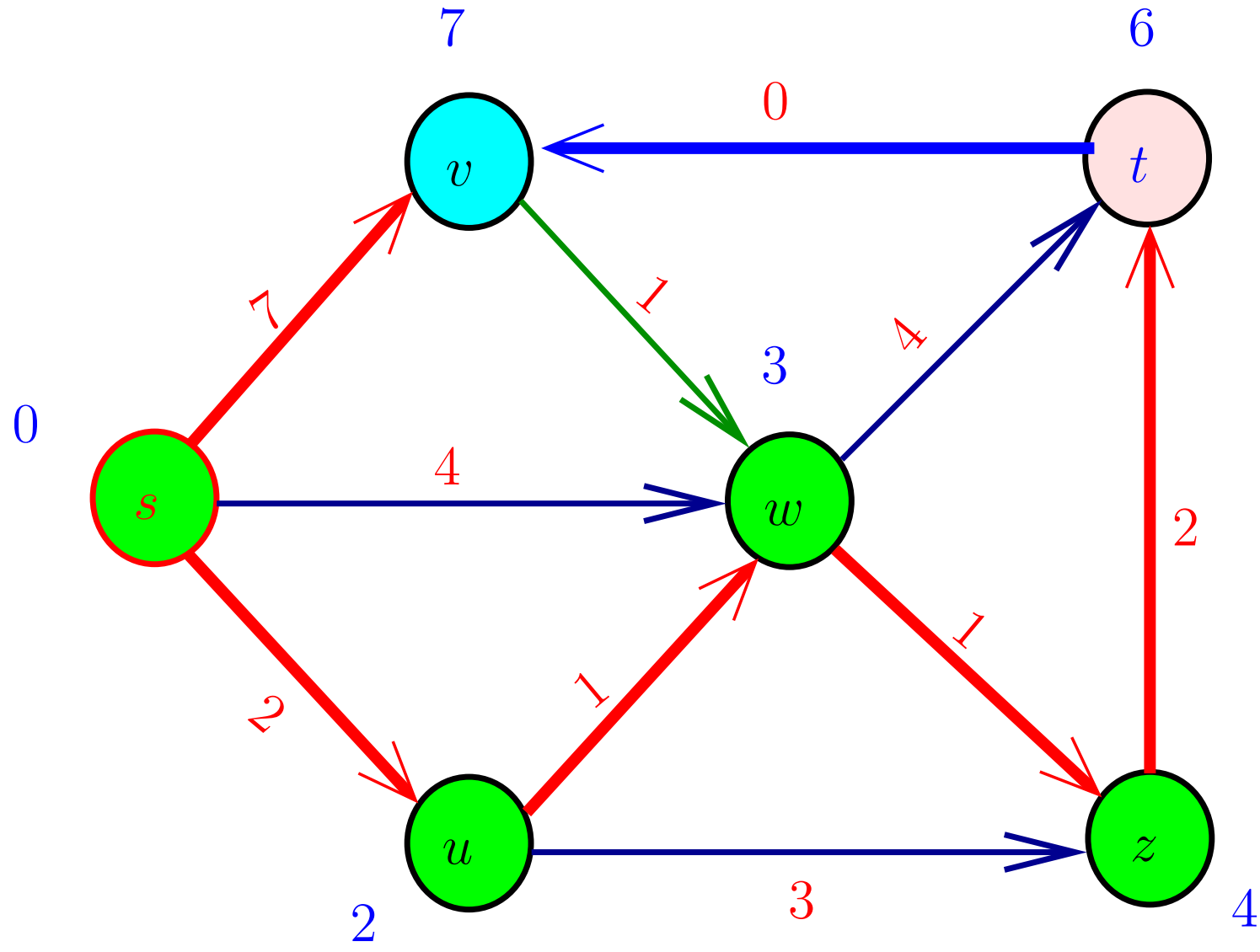
Simulação



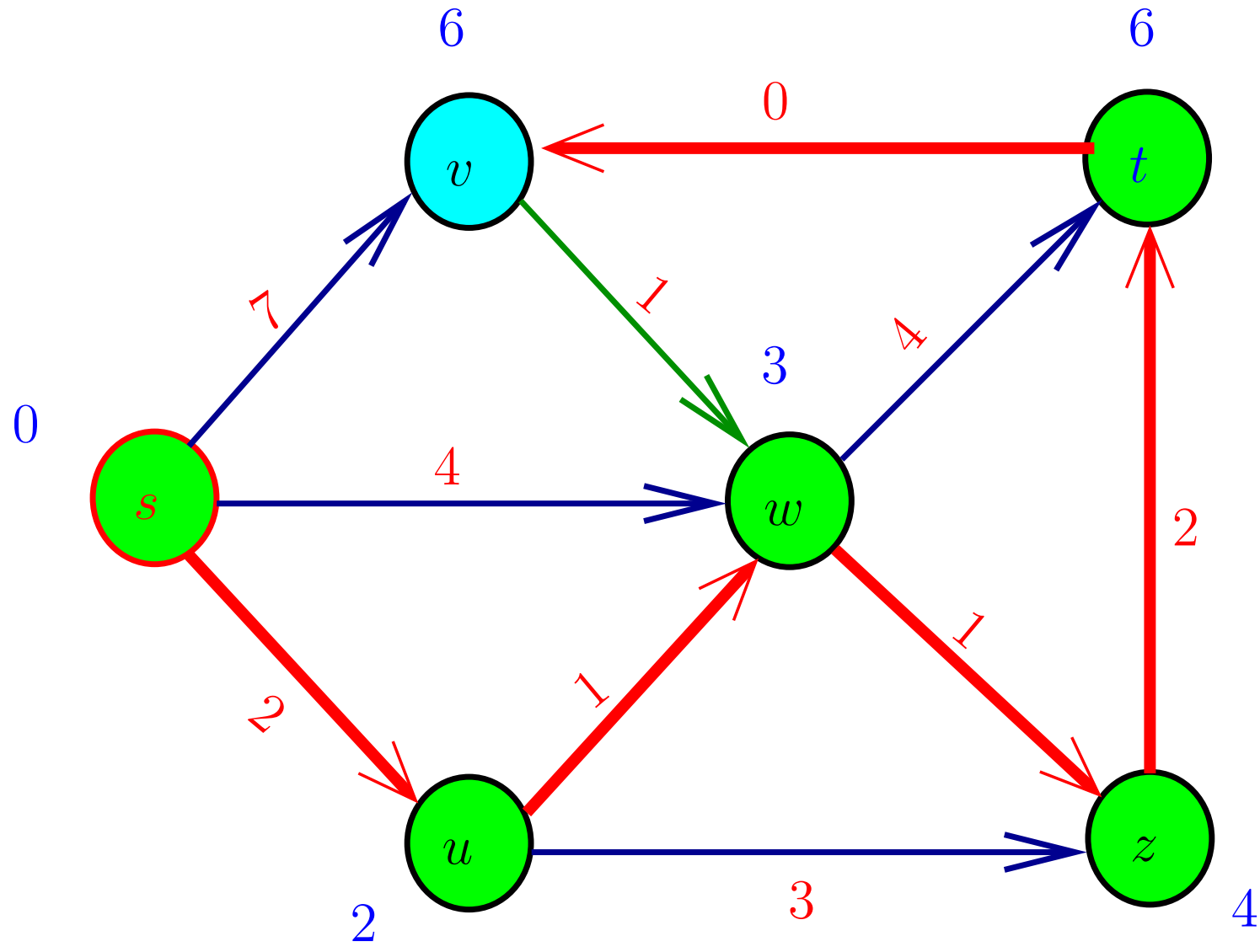
Simulação



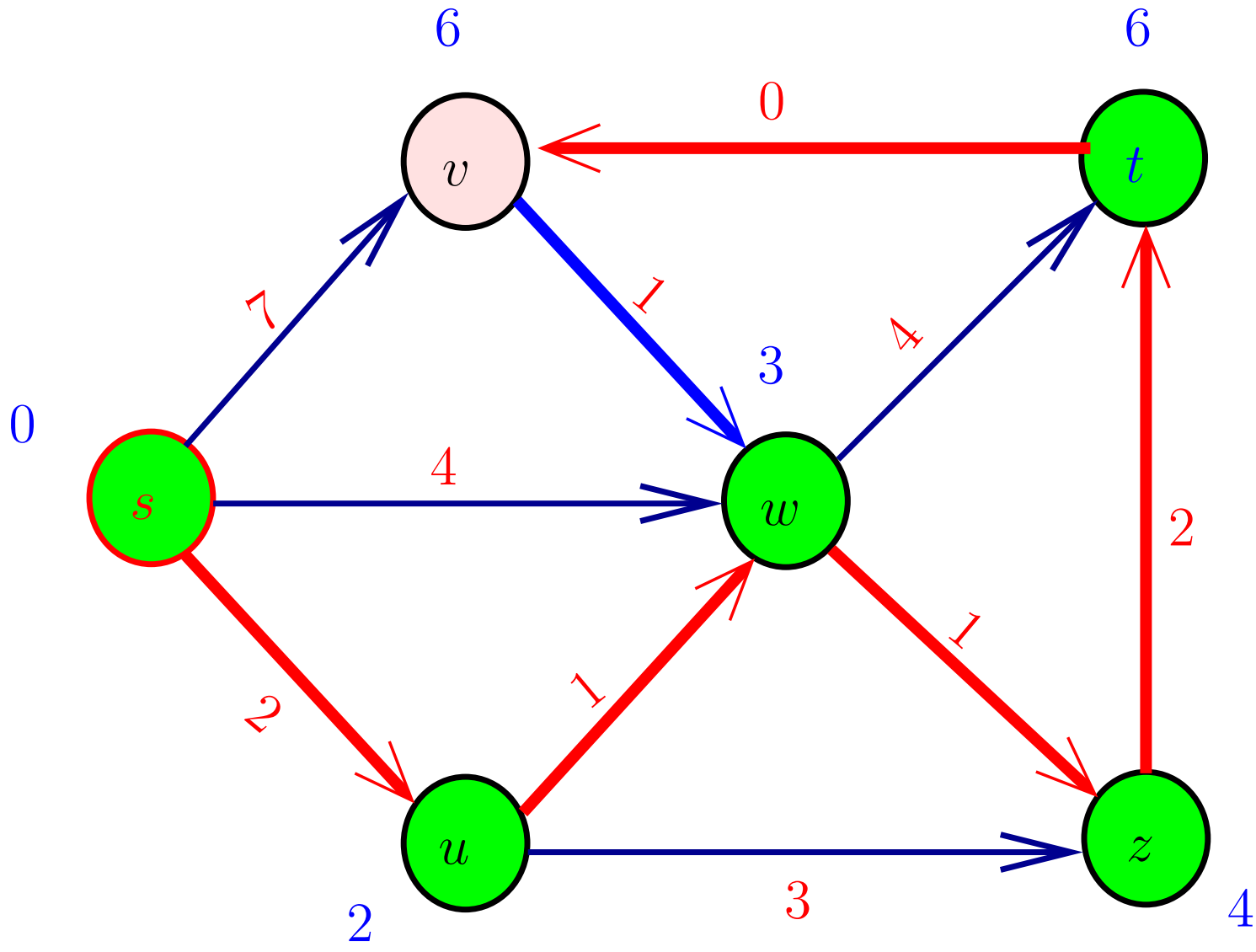
Simulação



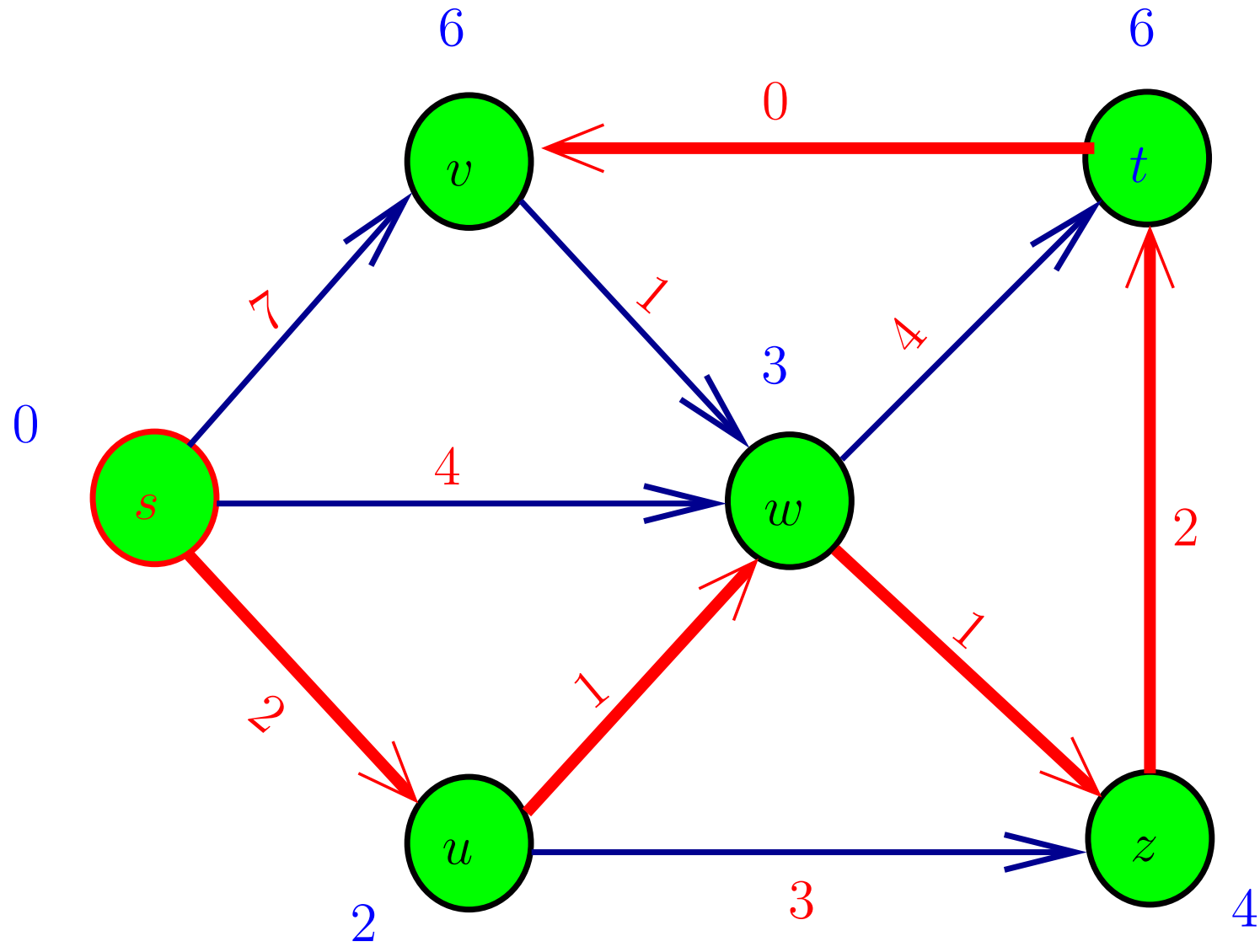
Simulação



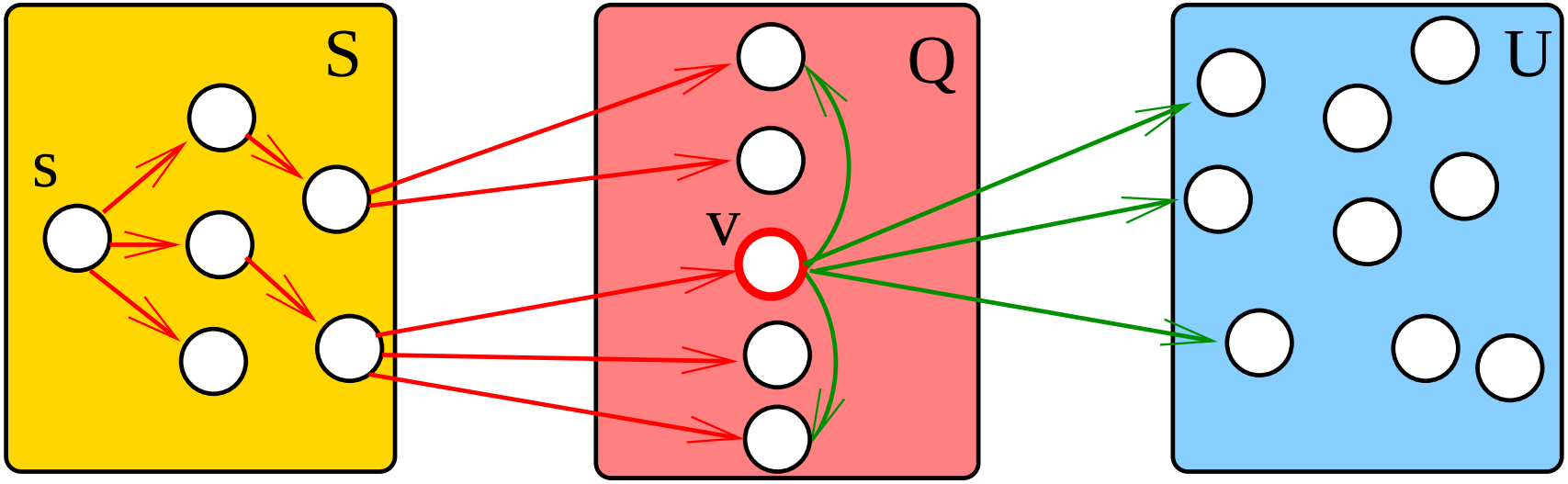
Simulação



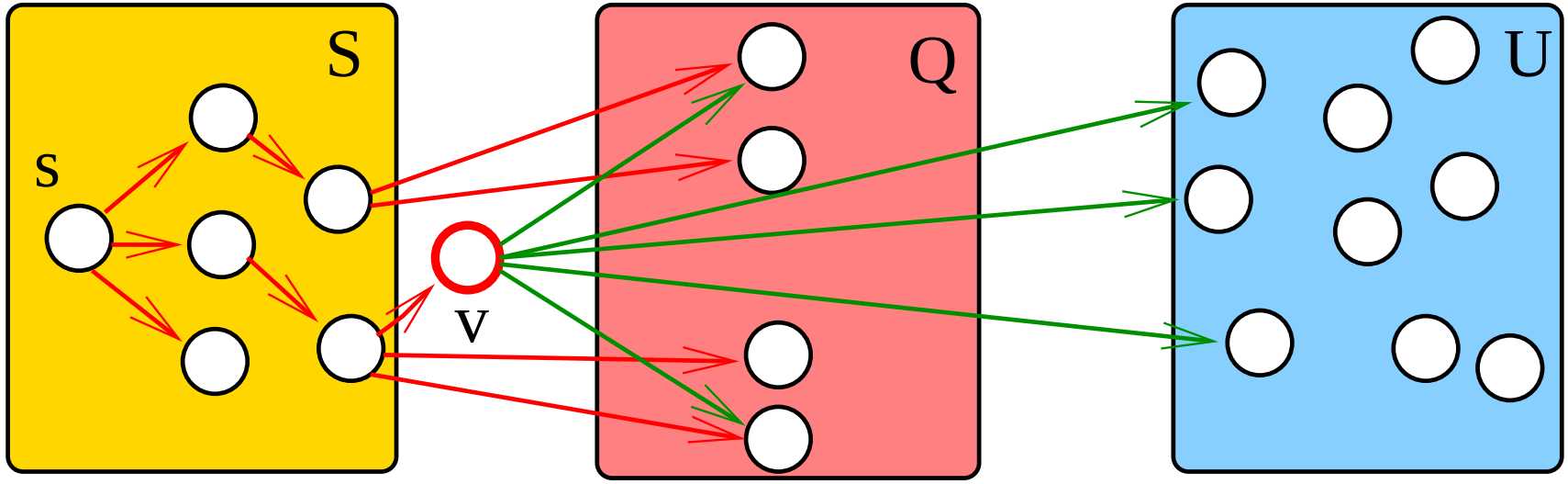
Simulação



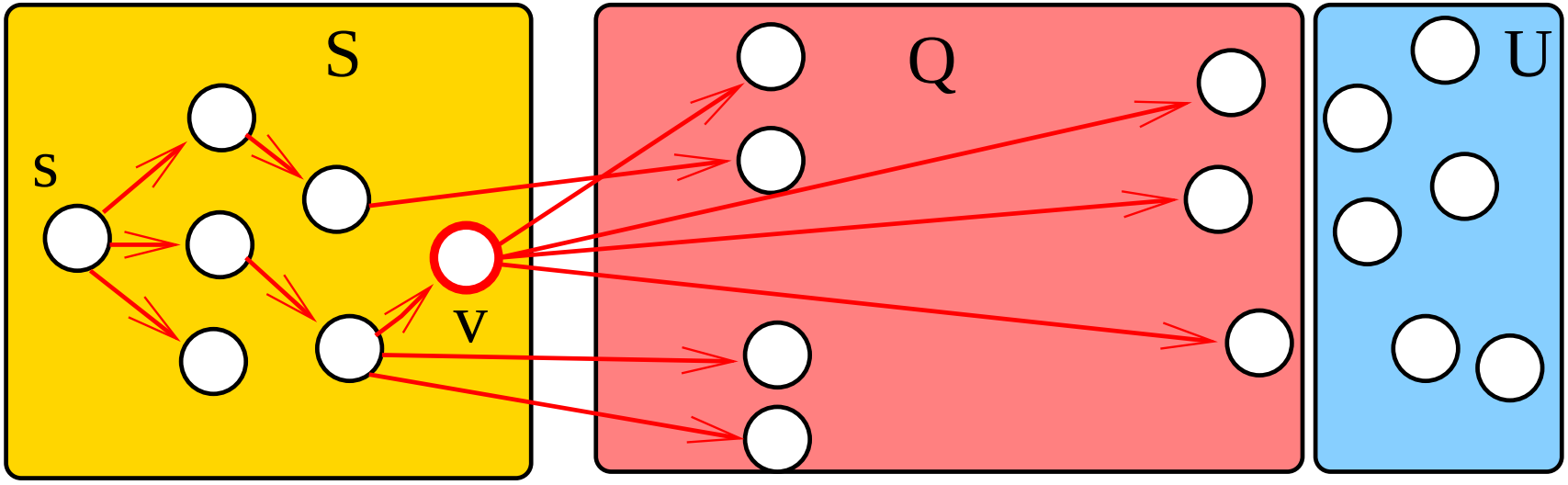
Iteração



Iteração



Iteração



Invariantes

Na linha 6, antes da verificação da condição "$Q \neq \langle\rangle$" valem as seguintes invariantes:

(i0) para cada arco pq no grafo de predecessores tem-se $y(q) - y(p) = c(pq)$ (igual!);

(i1) $\pi(s) = \text{NIL}$ e $y(s) = 0$;

(i2) para cada nó v distinto de s, $y(v) < nC + 1 \Leftrightarrow \pi(v) \neq \text{NIL}$;

(i3) para cada nó v, se $\pi(v) \neq \text{NIL}$ então existe um caminho de s a v no grafo de predecessores.

Mais invariantes

Na linha 6, antes da verificação da condição $Q \neq \langle\rangle$ valem, além de (i0)-(i3), as seguintes invariantes:

(i4) para cada arco pq com $y(q) - y(p) > c(pq)$ tem-se que p e q estão Q;

(i5) (monotonicidade) para quaisquer w em $N - Q$, i em Q vale que

$$y(w) \leq y(i).$$

Consumo de tempo

O número de iterações é $< n$.

linha	consumo de todas as execuções da linha
1-3	$\mathrm{O}(n)$
4	$\mathrm{O}(1)$
5	$\mathrm{O}(n)$
6	$n\,\mathrm{O}(1) = \mathrm{O}(n)$
7	$n\,\mathrm{O}(n) = \mathrm{O}(n^2)$
8-11	$m\,\mathrm{O}(1) = \mathrm{O}(m)$
12	$\mathrm{O}(n)$
total	$\mathrm{O}(1) + 4\,\mathrm{O}(n) + \mathrm{O}(m) + \mathrm{O}(n^2)$ $= \mathrm{O}(n^2)$

Conclusão

O consumo de tempo do algoritmo DIJKSTRA é $\mathrm{O}(n^2)$.

Implementação com min-heap

```
HEAP-DIJKSTRA (N, A, c, s)        ▷ c ≥ 0
 1   para cada i em N faça
 2       y(i) ← nC + 1    ▷ nC + 1 faz o papel de ∞
 3       π(i) ← NIL
 4   y(s) ← 0
 5   Q ← BUILD-MIN-HEAP (N)    ▷ Q é um min-heap
 6   enquanto Q ≠ ⟨⟩ faça
 7       i ← EXTRACT-MIN (Q)
 8       para cada ij em A(i) faça
 9           valor ← y(i) + c(ij)
10           se y(j) > valor então
11               DECREASE-KEY(valor, j, Q)
12               π(j) ← i
13   devolva π e y
```

Consumo de tempo

O número de iterações é $< n$.

linha	consumo de todas as execuções da linha
1-4	$\mathrm{O}(n)$
5	$\mathrm{O}(n)$
6	$n\,\mathrm{O}(1) = \mathrm{O}(n)$
7	$n\,\mathrm{O}(\lg n) = \mathrm{O}(n \lg n)$
8-10	$m\,\mathrm{O}(1) = \mathrm{O}(m)$
11	$m\,\mathrm{O}(\lg n) = \mathrm{O}(m \lg n)$
12	$m\,\mathrm{O}(1) = \mathrm{O}(m)$
13	$\mathrm{O}(n)$
total	$4\,\mathrm{O}(n) + 2\,\mathrm{O}(m) + \mathrm{O}(n \lg n) + \mathrm{O}(m \lg n)$
	$= \mathrm{O}(m \lg n)$ (supondo (N, A) conexo)

Conclusão

O consumo de tempo do algoritmo HEAP-DIJKSTRA é $\mathrm{O}(m \lg n)$.

Este consumo de tempo é assintoticamente menor que o do algoritmo DIJKSTRA para redes "esparsas" (tecnicamente falando $m = \mathrm{O}(n^2 / \lg n)$).

Consumo de tempo (resumo)

	heap	d-heap	fibonacci heap
INSERT	$\mathrm{O}(\lg n)$	$\mathrm{O}(\log_D n)$	$\mathrm{O}(1)$
EXTRACT-MIN	$\mathrm{O}(\lg n)$	$\mathrm{O}(\log_D n)$	$\mathrm{O}(\lg n)$
DECREASE-KEY	$\mathrm{O}(\lg n)$	$\mathrm{O}(\log_D n)$	$\mathrm{O}(1)$
DIJKSTRA	$\mathrm{O}(m \lg n)$	$\mathrm{O}(m \log_D n)$	$\mathrm{O}(m + n \lg n)$

	bucket heap	radix heap
INSERT	$\mathrm{O}(1)$	$\mathrm{O}(\lg(nC)$R
EXTRACT-MIN	$\mathrm{O}(C)$	$\mathrm{O}(\lg(nC)$
DECREASE-KEY	$\mathrm{O}(1)$	$\mathrm{O}(m + n \lg(nC))$
DIJKSTRA	$\mathrm{O}(m + nC)$	$\mathrm{O}(m + n \lg(nC))$

Algoritmo DIJKSTRA e ordenação

Devido a relação invariante da monotonicidade (i5) tem-se que:

> O algoritmo DIJKSTRA retira os nós da fila Q na linha 7 do algoritmo em ordem não-decrescente das suas distância a partir de s.

Conclusão: o consumo de tempo do algoritmo DIJKSTRA é $\Omega(n \log n)$

Conclusão

Como o algoritmo DIJKSTRA examina na linha 9 cada arco uma vez temos que:

O consumo de tempo (no "modelo comparação-adição") do algoritmo DIJKSTRA é $\Omega(m + n \lg n)$.

Usando fibonacci-heaps, Fredman e Tarjan obtiveram uma implementação do algoritmo DIJKSTRA que consome tempo $\mathrm{O}(m + n \lg n)$.