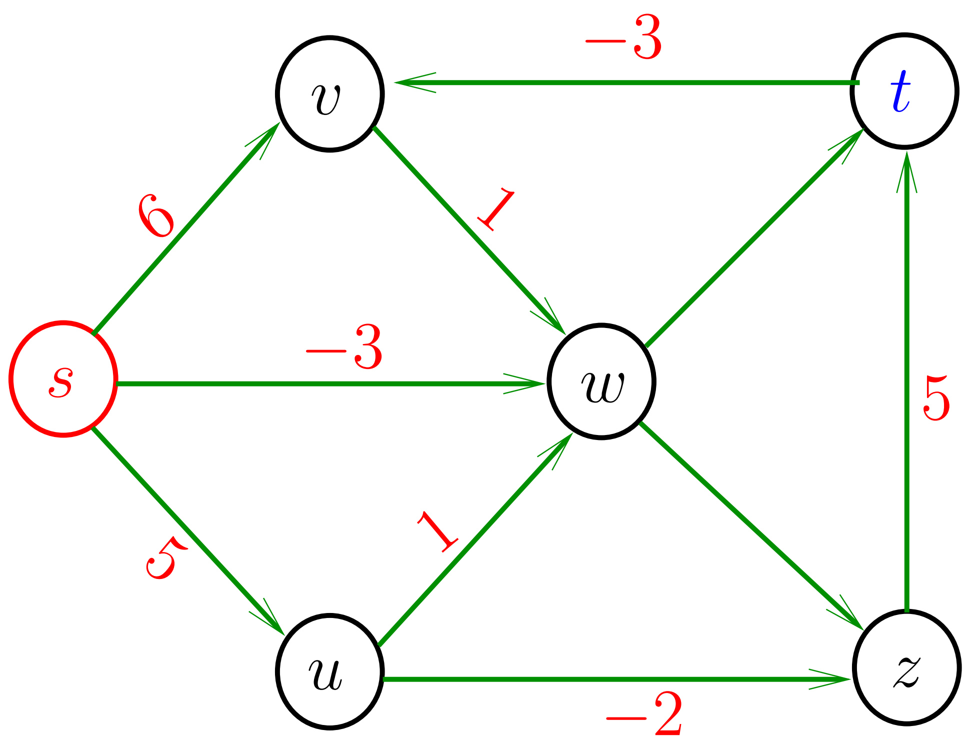
Melhores momentos

AULA 5

Custos

Uma função-custo é uma função de A em $\mathbb{Z}$.



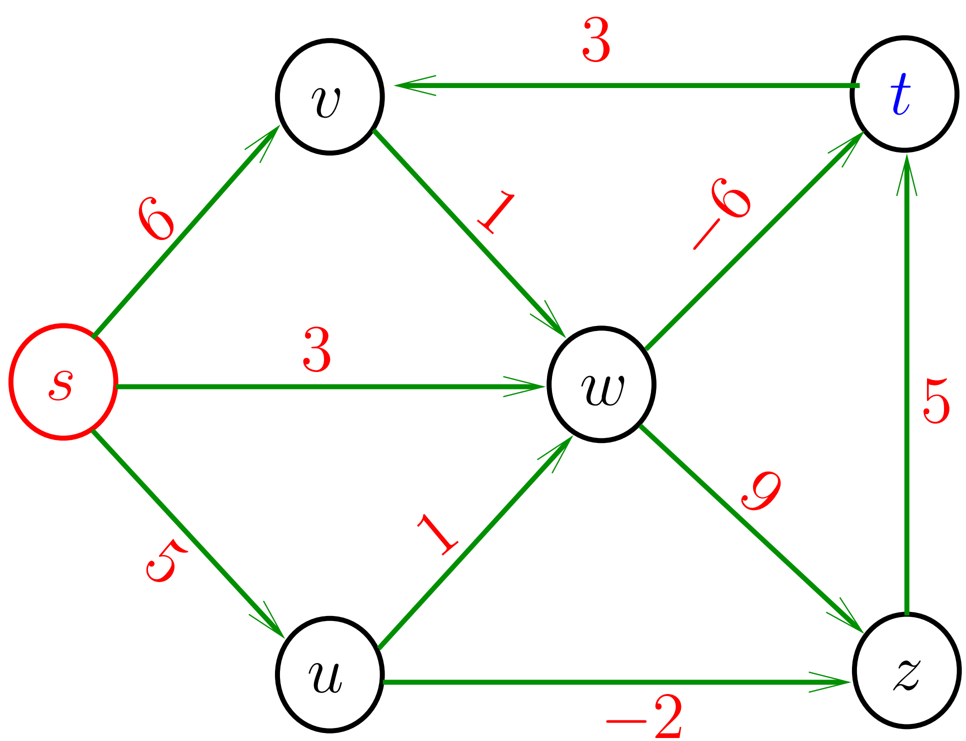
Custo de um passeio

O custo de um passeio é soma dos custos dos arcos no passeio.

O custo do passeio $\langle s, v, w, z\rangle$ é 16.

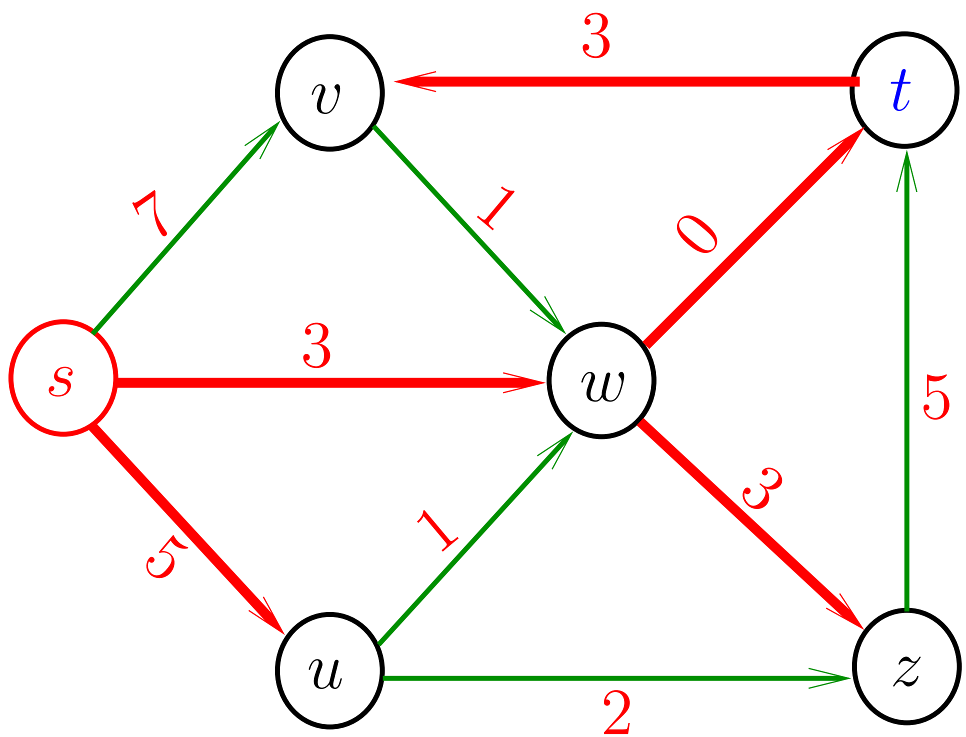
O custo do passeio $\langle s, v, w, t, v, w, z\rangle$ é 14.

O custo do passeio $\langle s, v, w, t, v, w, t, v, w, z\rangle$ é 12.



Problema

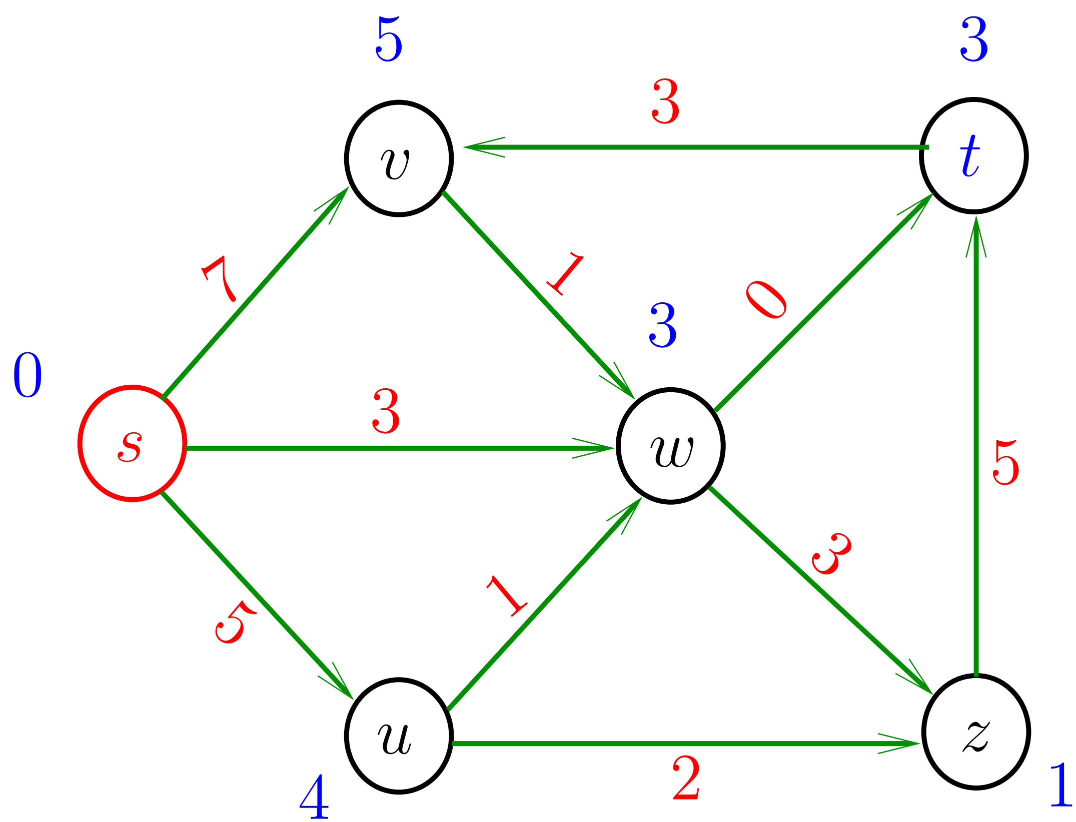
Problema do caminho mínimo sob custo não-negativo: Dada uma rede (N, A, c) com função-custo $c : A \rightarrow \mathbb{Z}_{\geq}$ e um nó s, encontrar, para cada nó t, um caminho de custo mínimo de s a t.



c-potencial

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

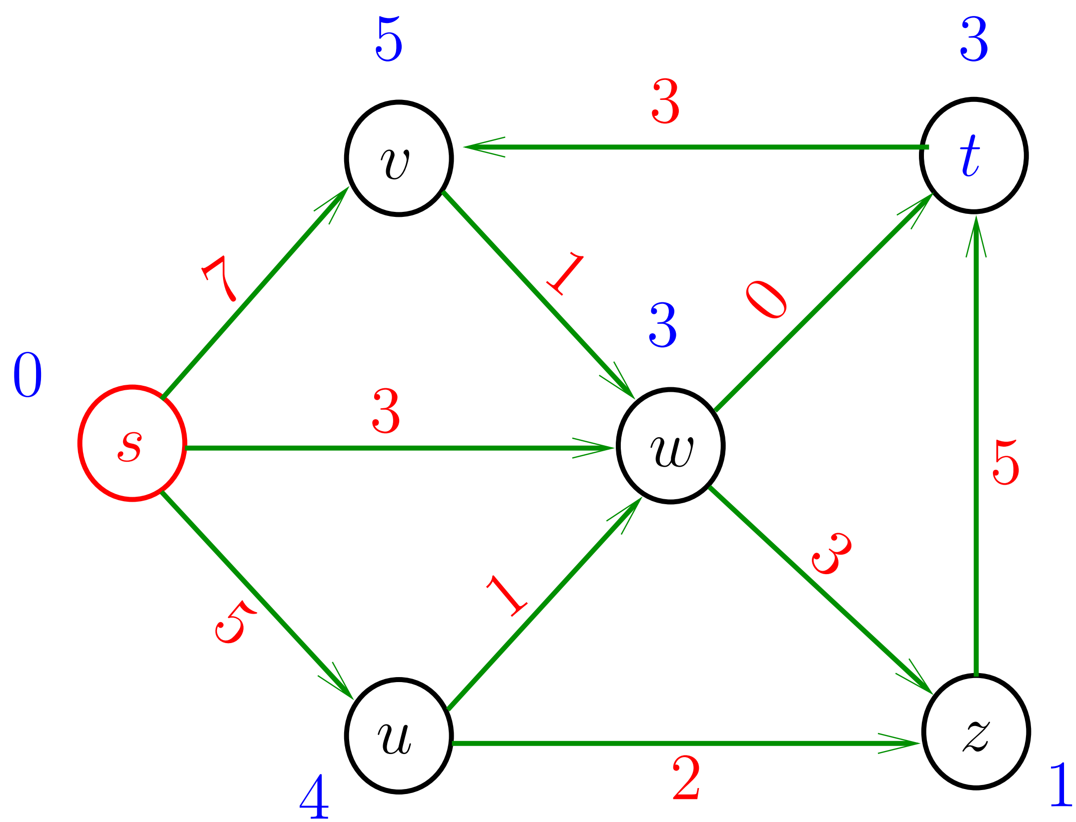
$$y(j) - y(i) \le c(ij) \quad \text{para todo arco } ij.$$



Propriedade de c-Potenciais

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

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

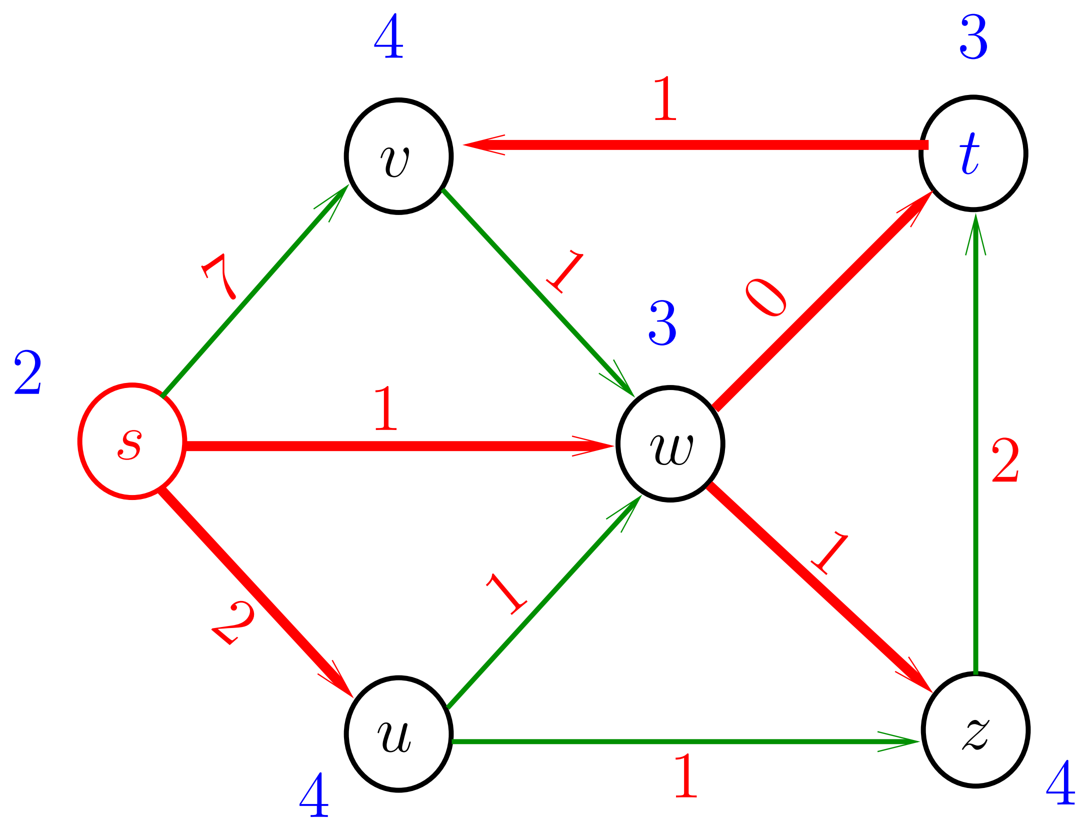


Conseqüência

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

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

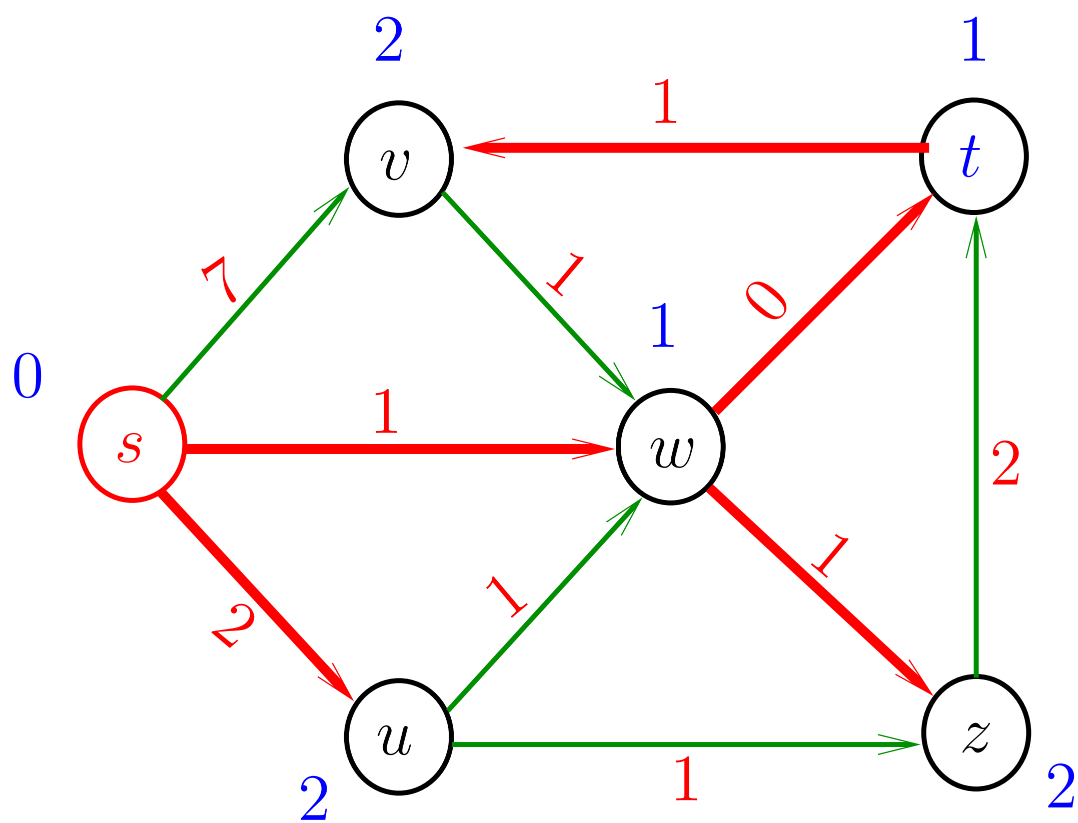
então P é uma caminho mínimo e y é um c-potencial tal que o valor de $y(t) - y(s)$ é máximo.



Potenciais ótimos

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

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



Implementação do algoritmo FORD

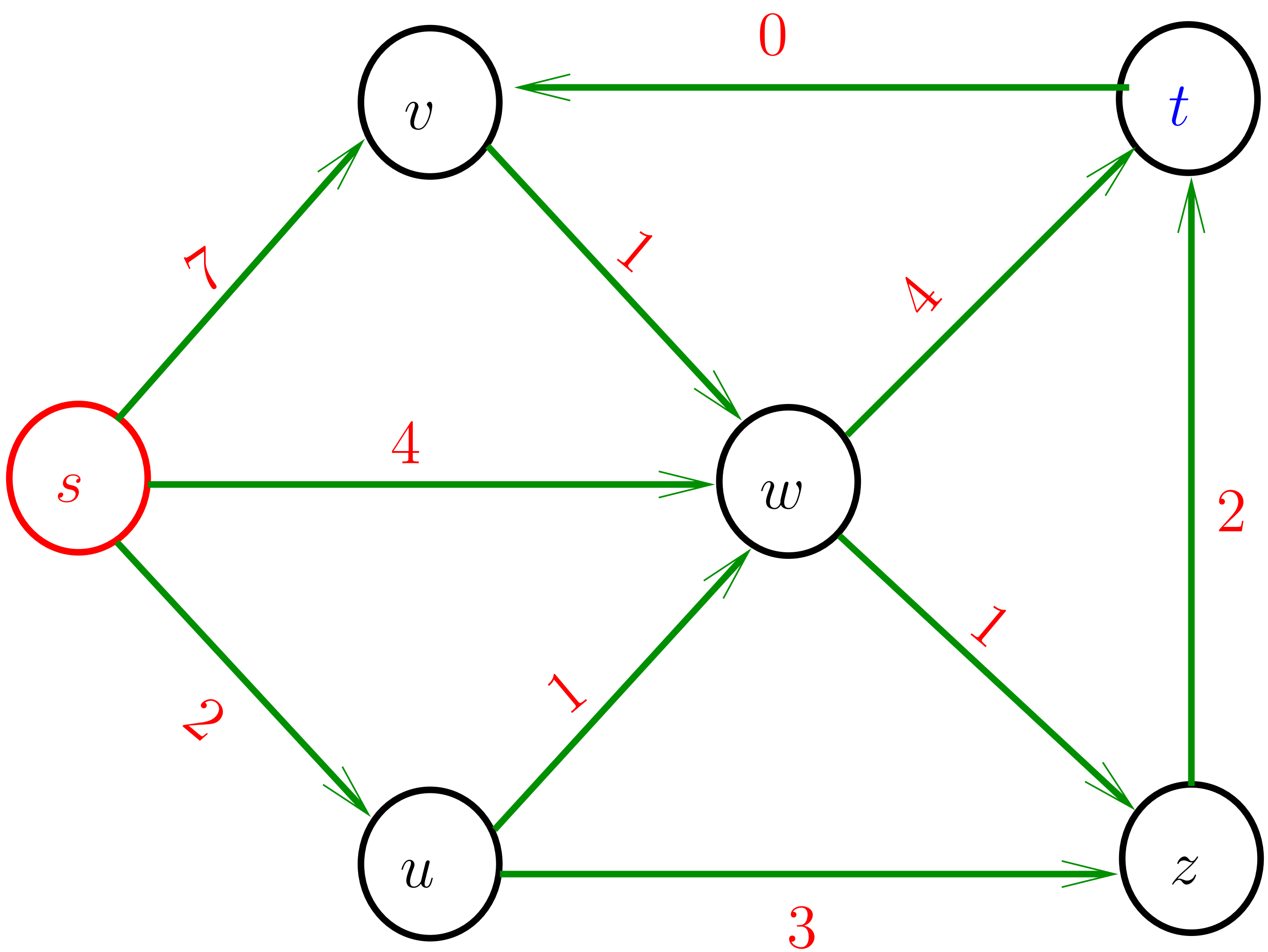
DIJKSTRA (N, A, c, s) $\triangleright$ $c \geq 0$

para cada i em N **faça**
    $y(i) \leftarrow nC + 1$ $\triangleright$ $nC + 1$ faz o papel de ∞
    $\pi(i) \leftarrow \mathrm{NIL}$
$y(s) \leftarrow 0$
$Q \leftarrow N$ $\triangleright$ Q func. como uma fila com prioridades

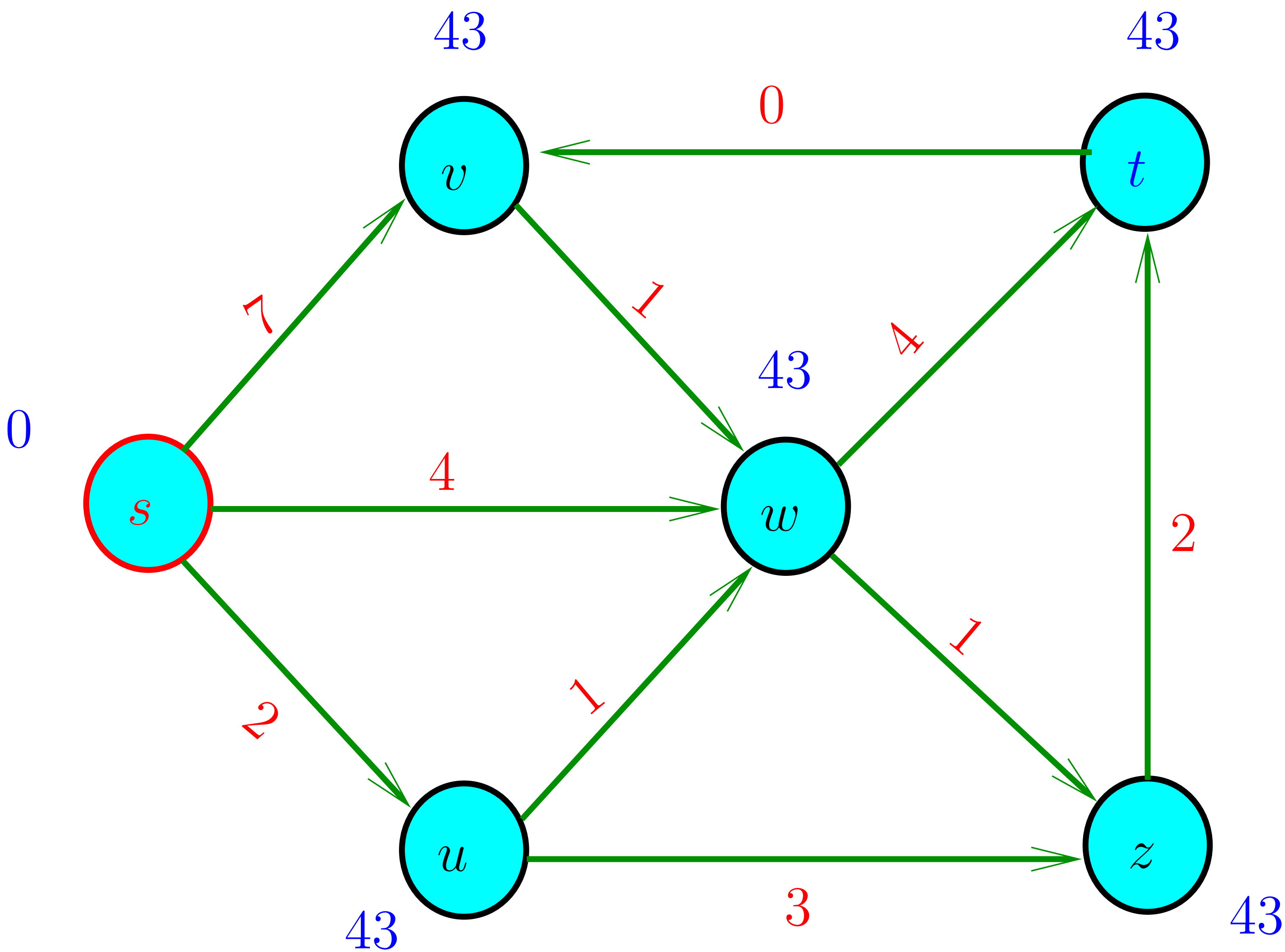
enquanto $Q \neq \langle\rangle$ **faça**
    retire de Q um nó i com $y(i)$ mínimo
    para cada ij em $A(i)$ **faça**
        se $y(j) > y(i) + c(ij)$ **então**
            $y(j) \leftarrow y(i) + c(ij)$
            $\pi(j) \leftarrow i$

devolva π e y

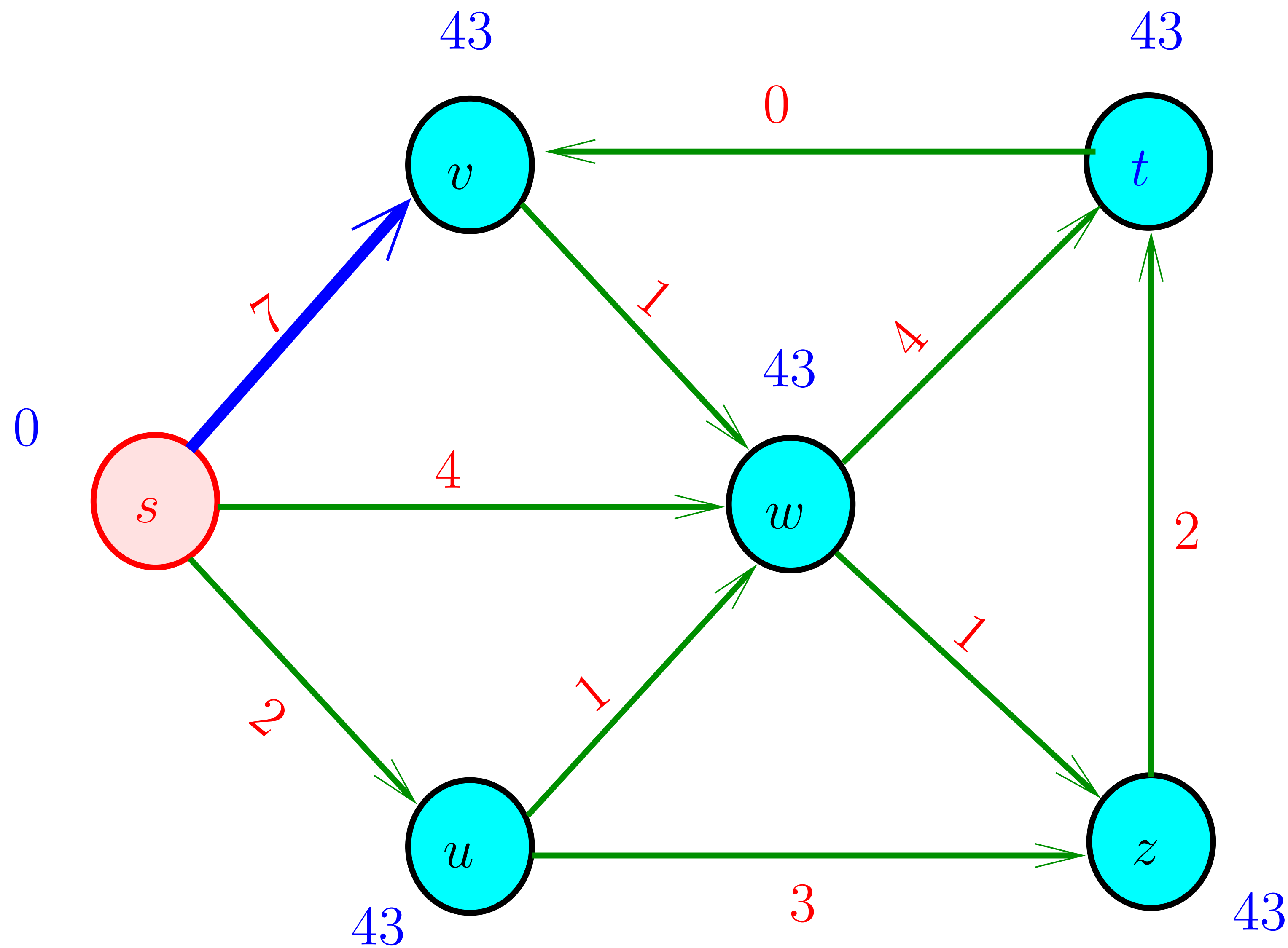
Simulação



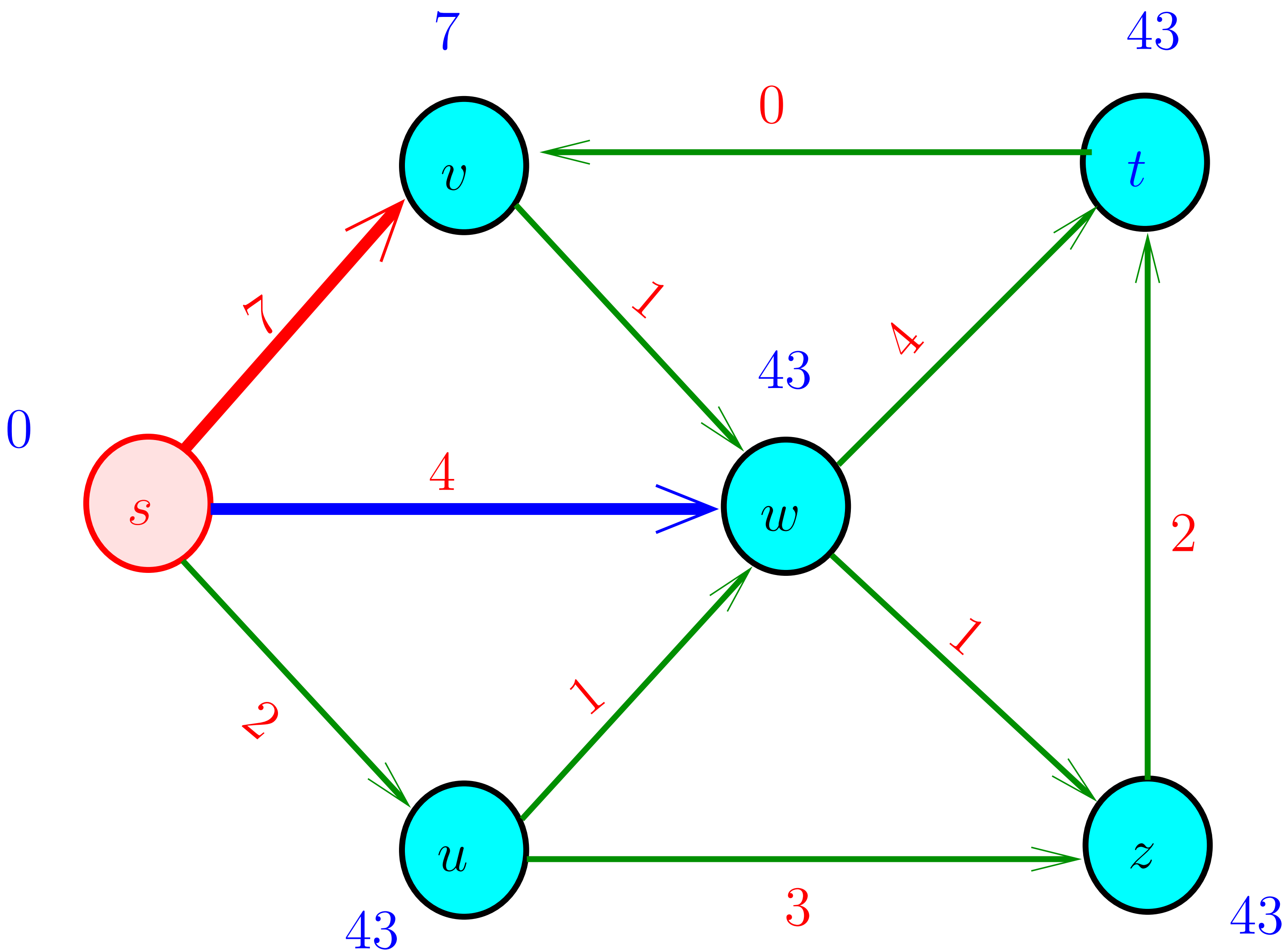
Simulação



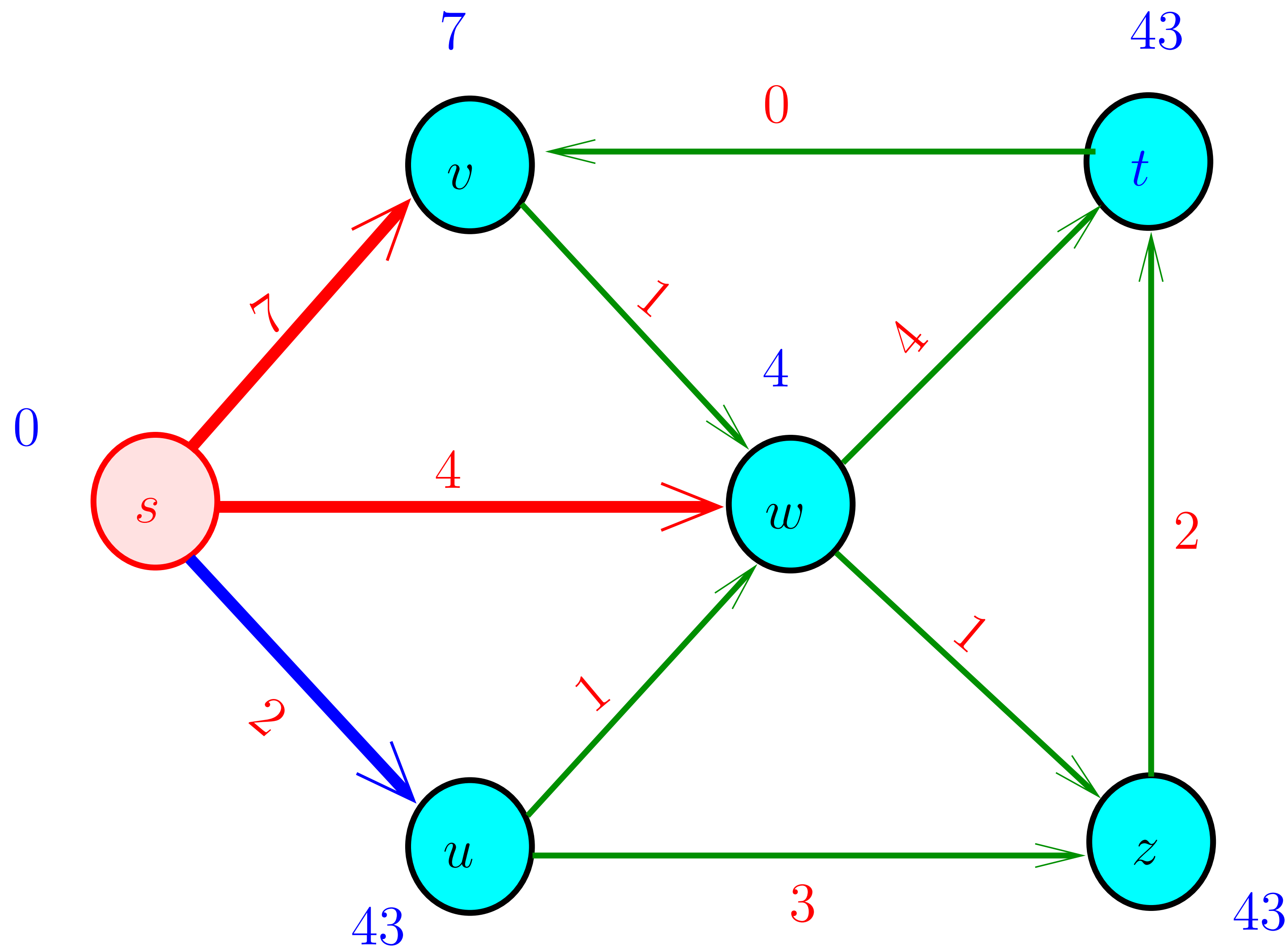
Simulação



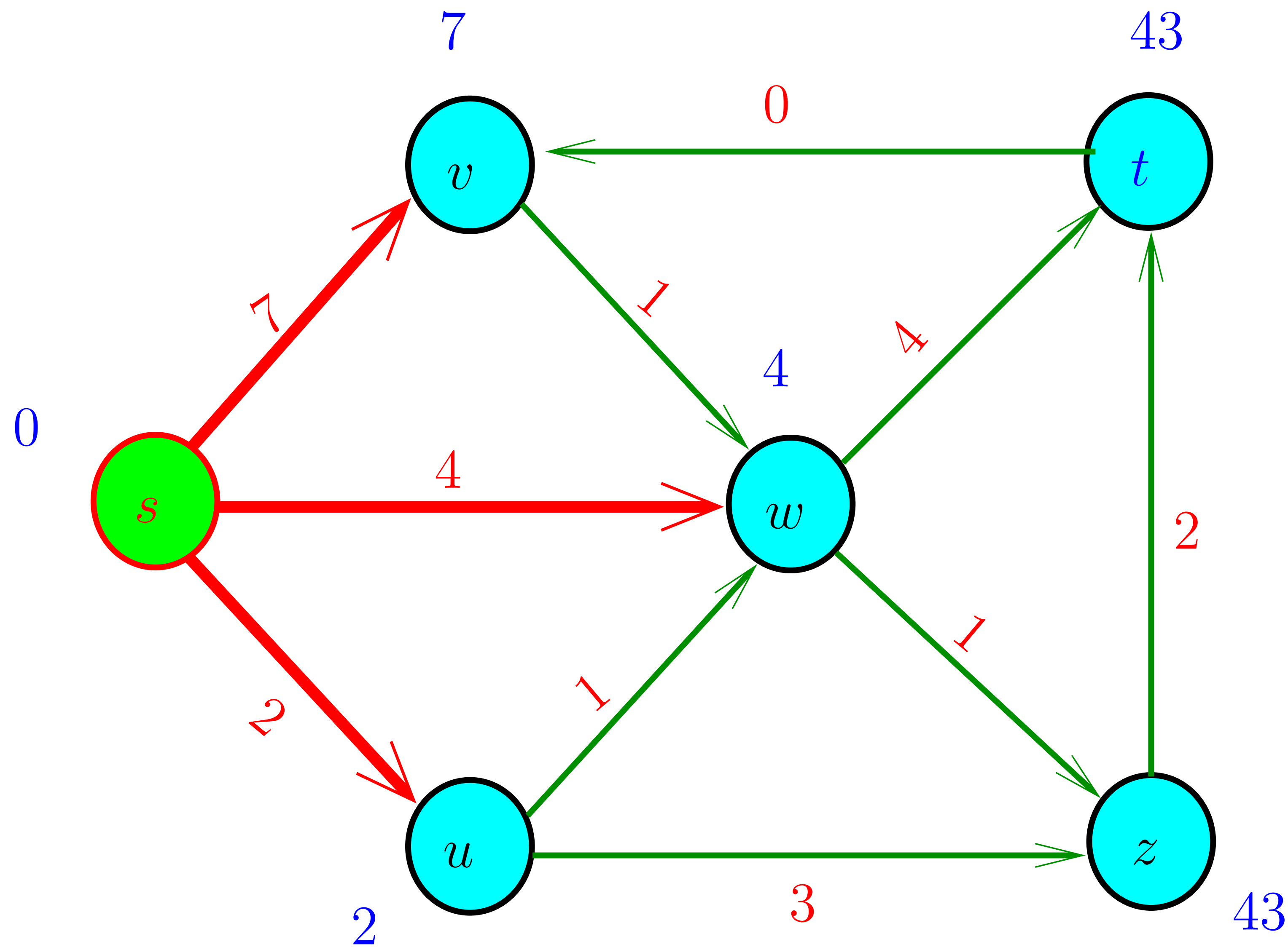
Simulação



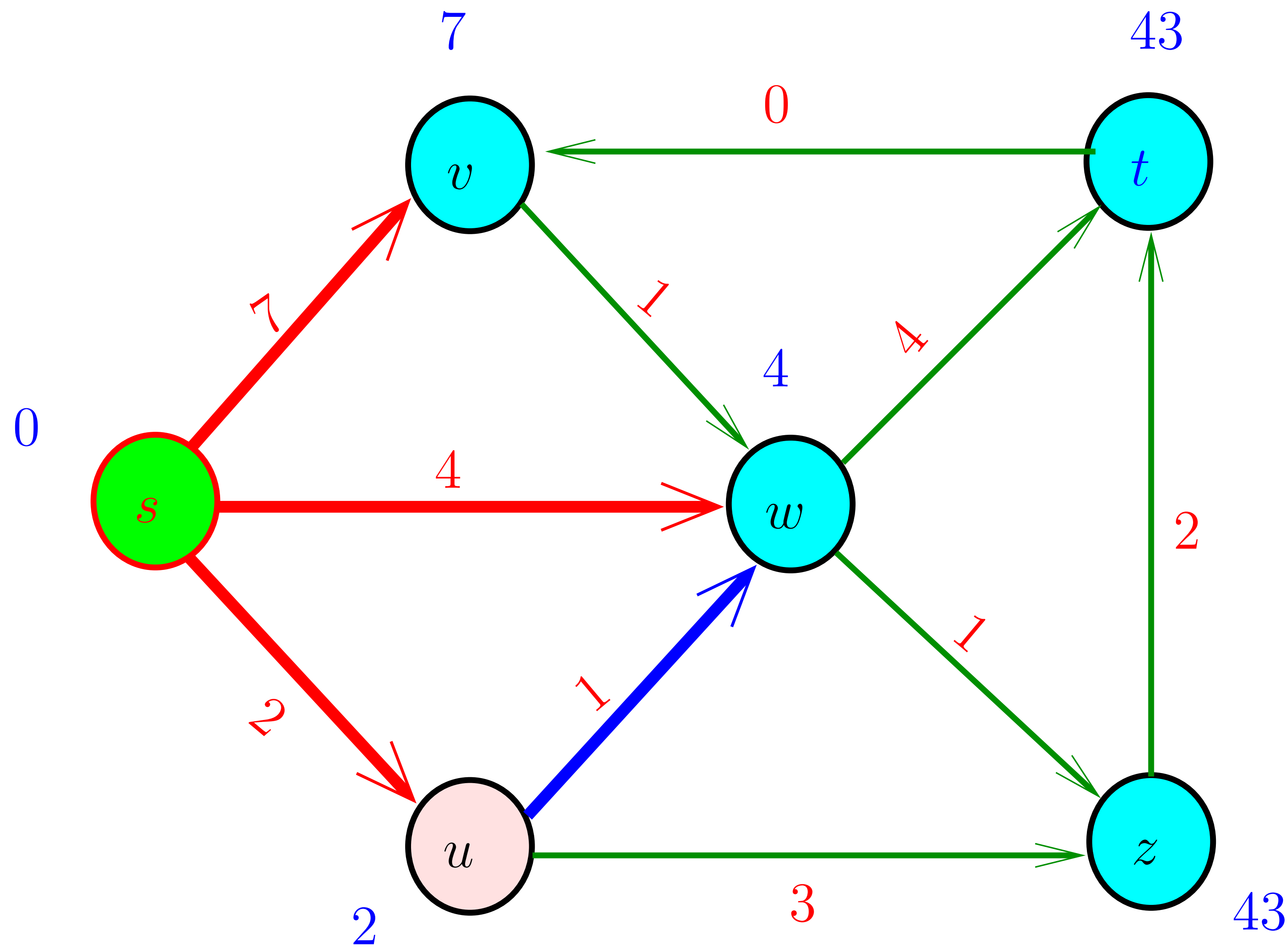
Simulação



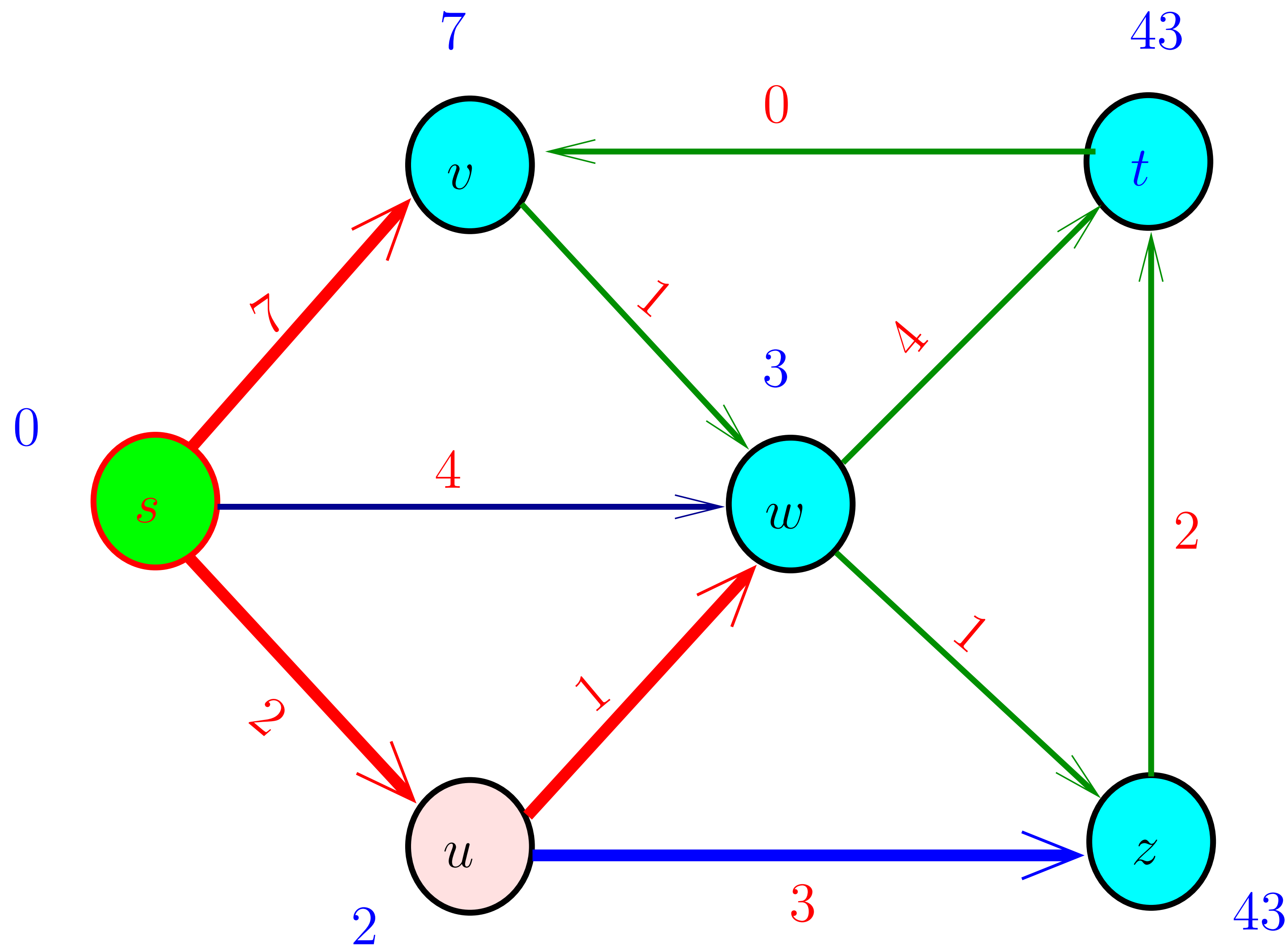
Simulação



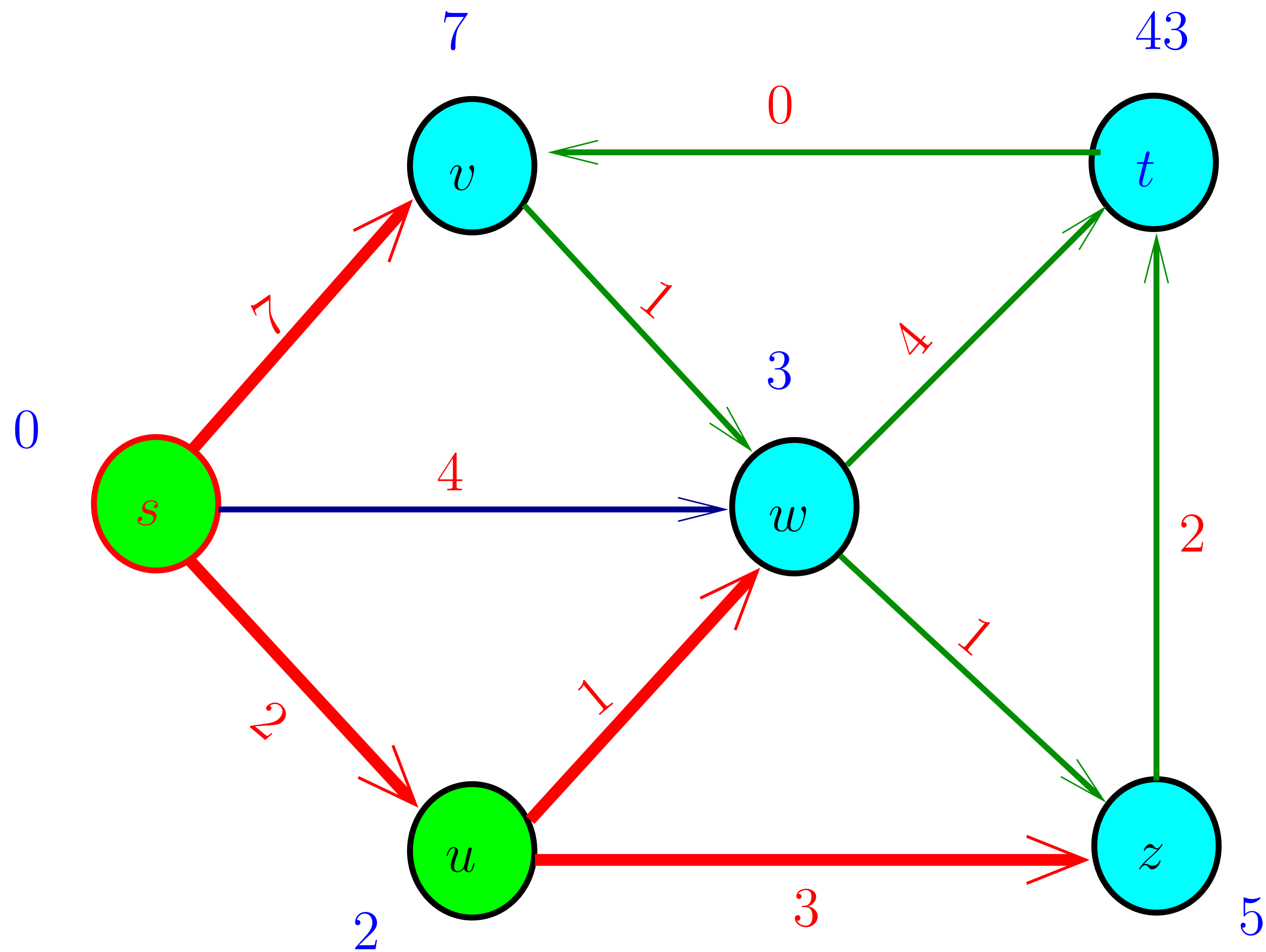
Simulação



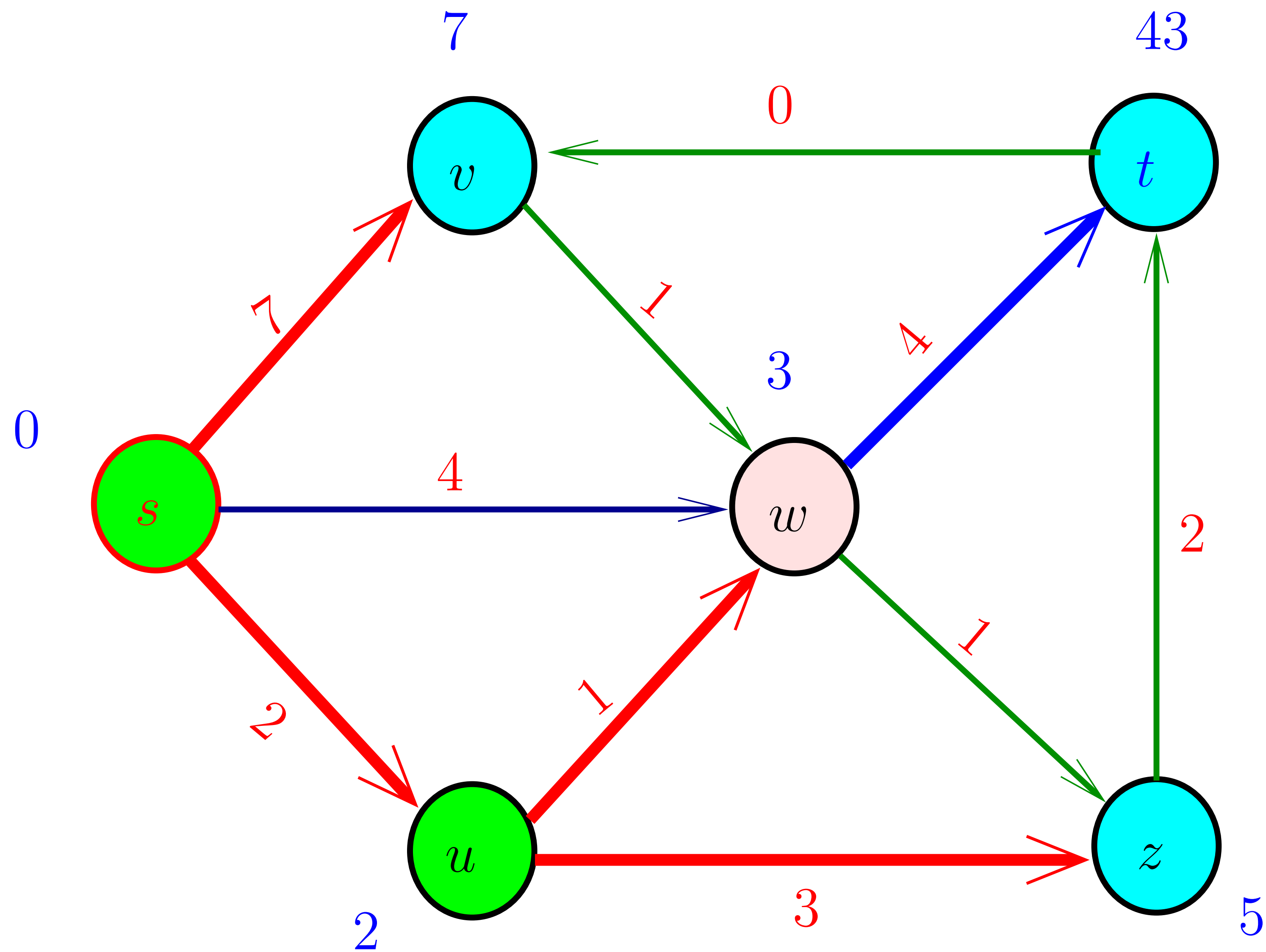
Simulação



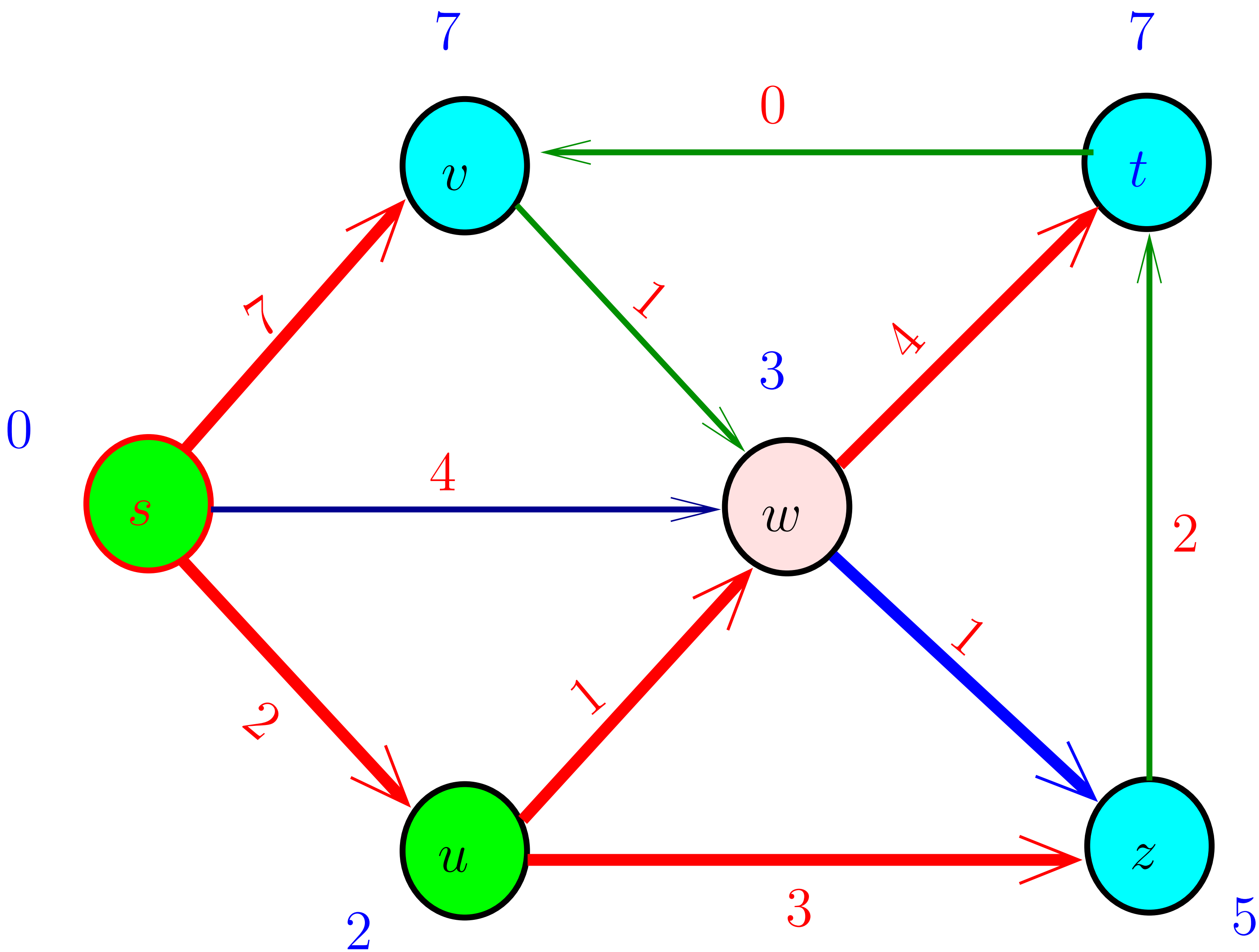
Simulação



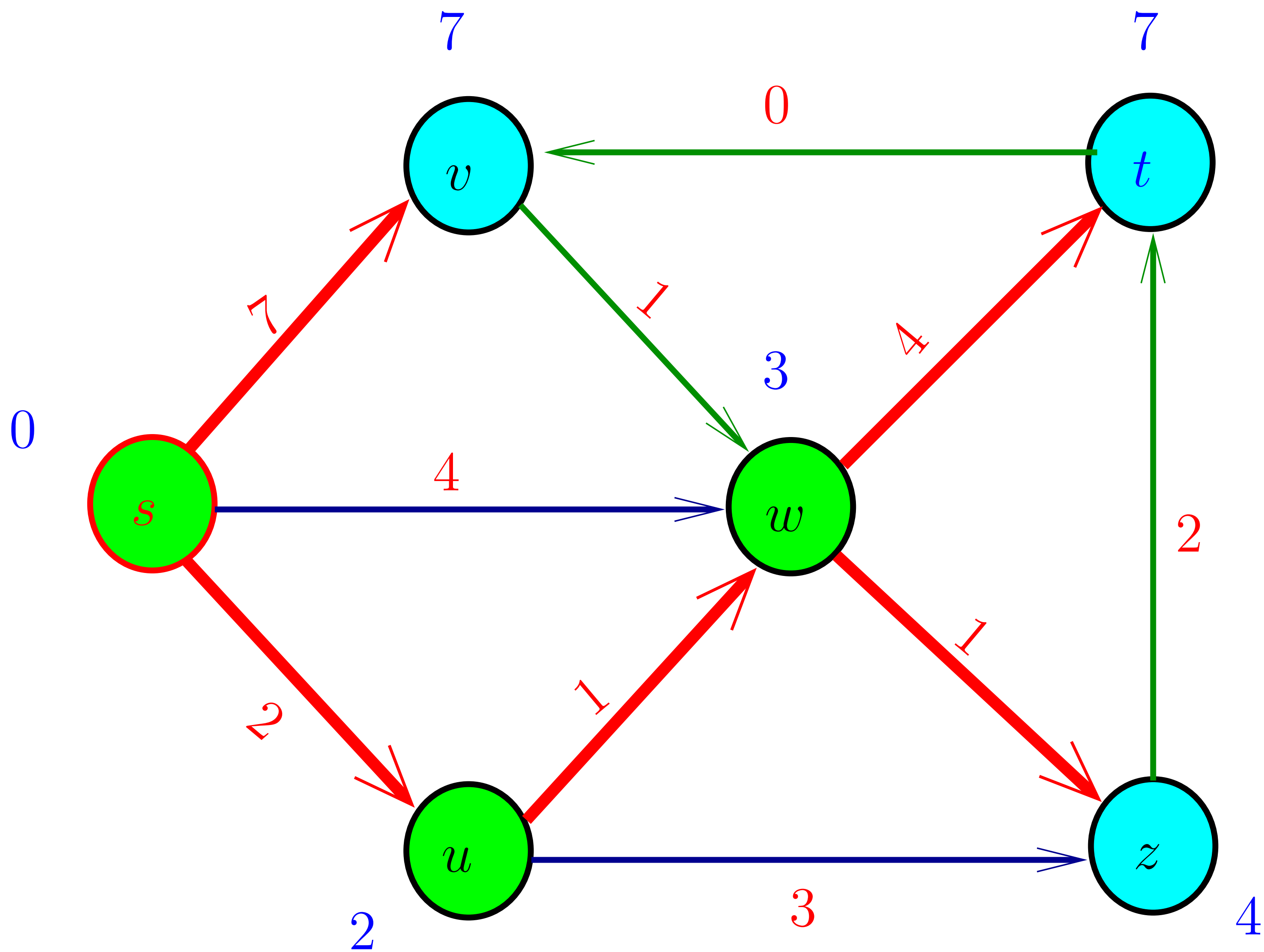
Simulação



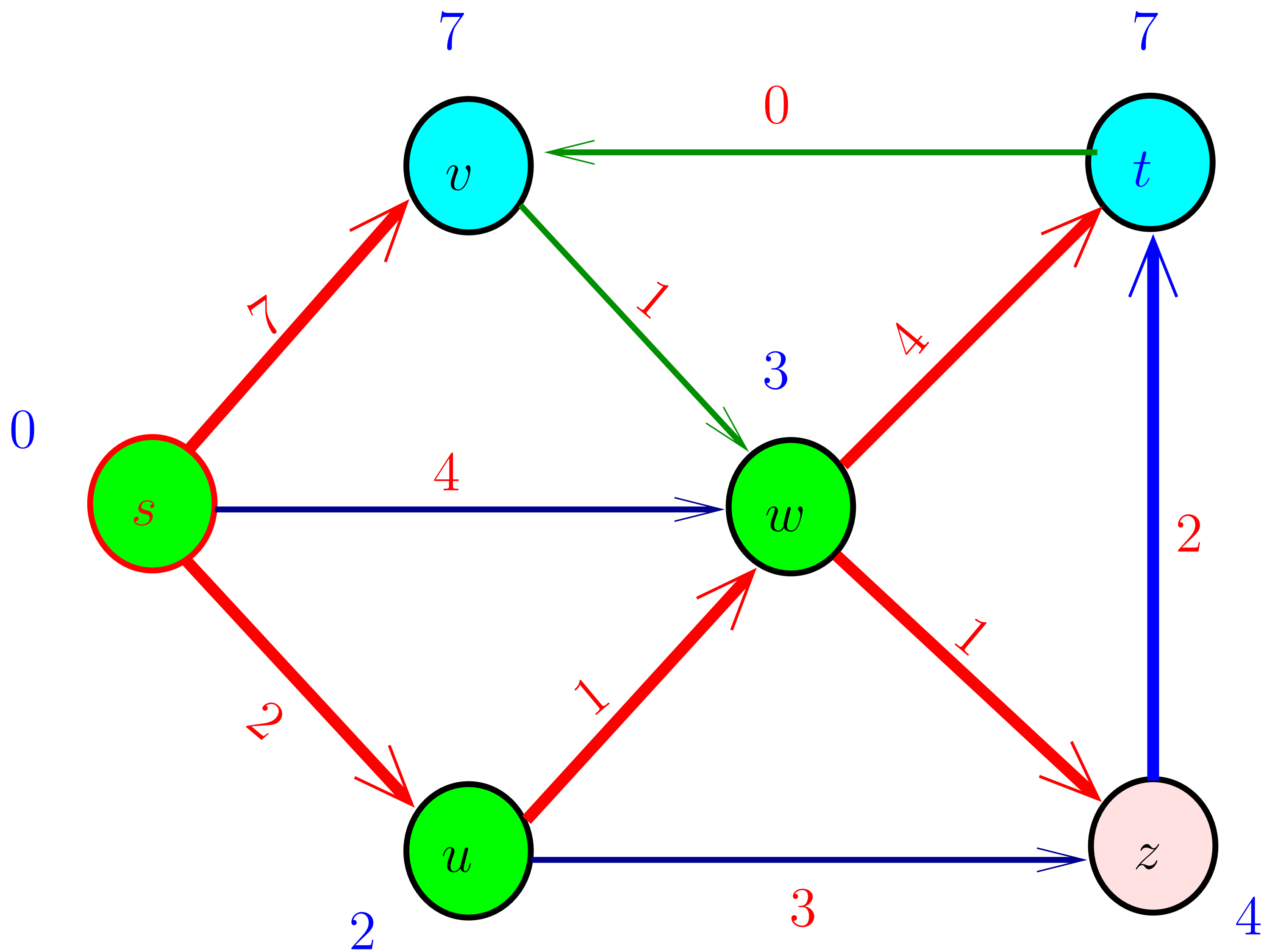
Simulação



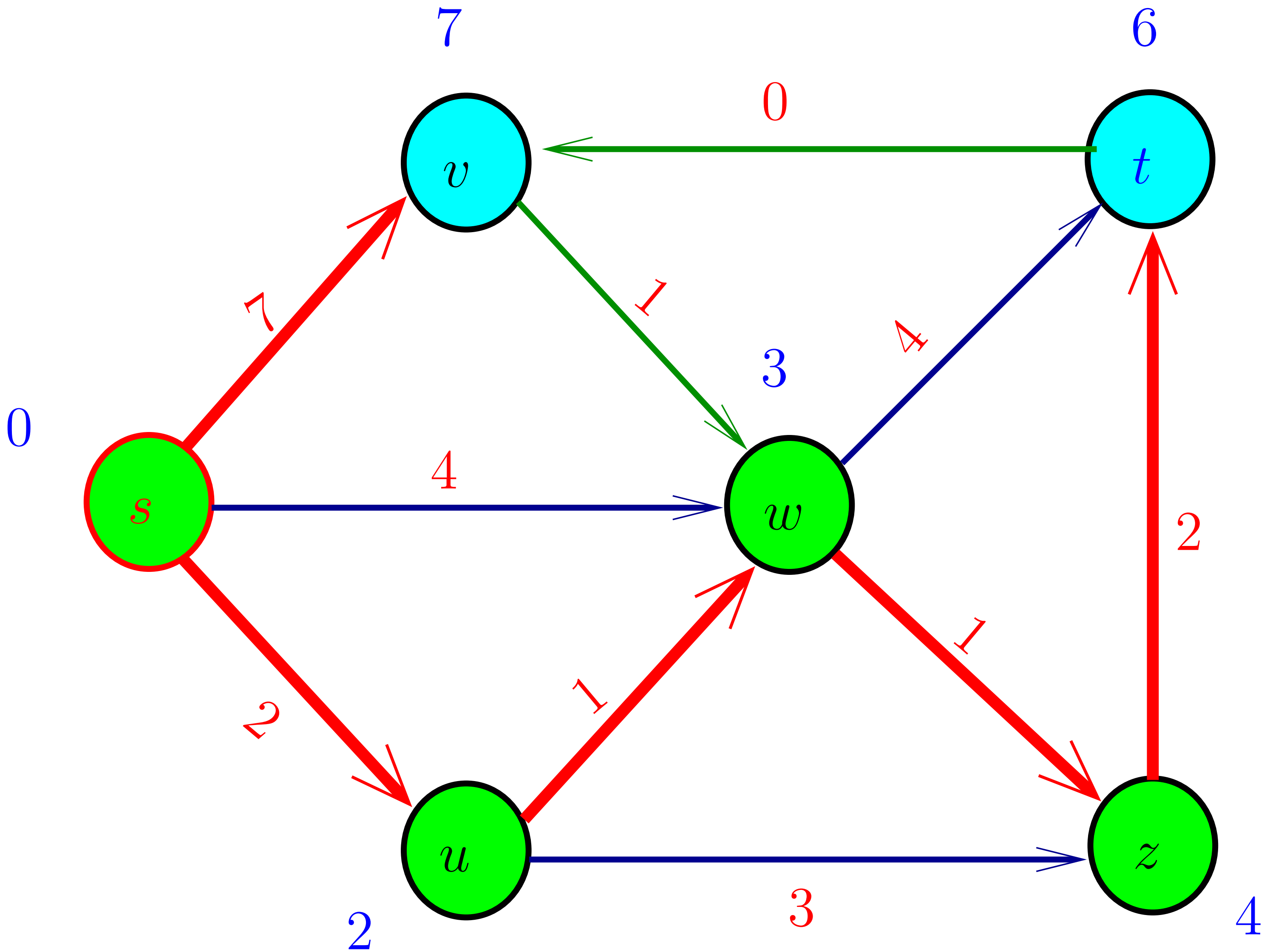
Simulação



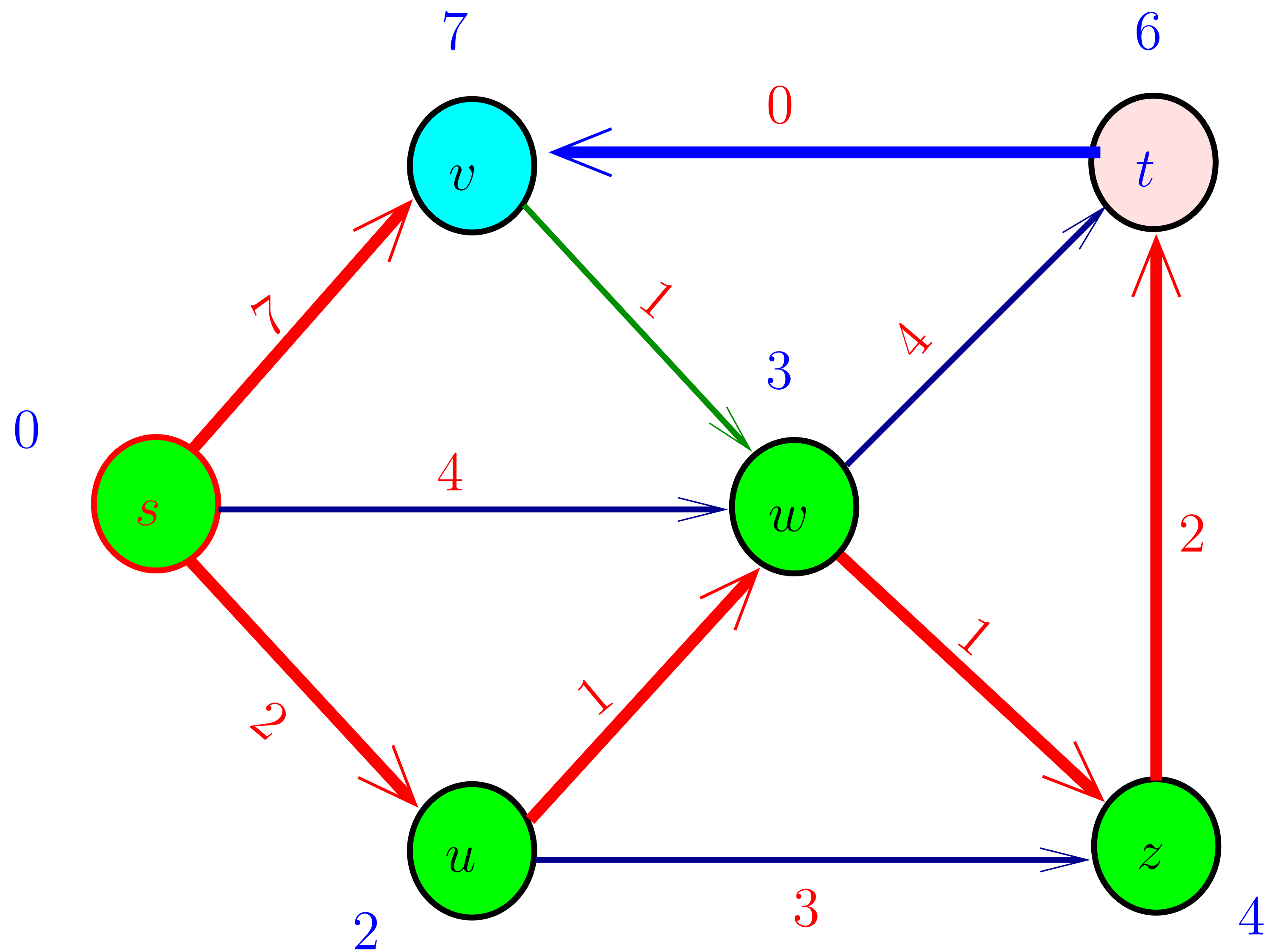
Simulação



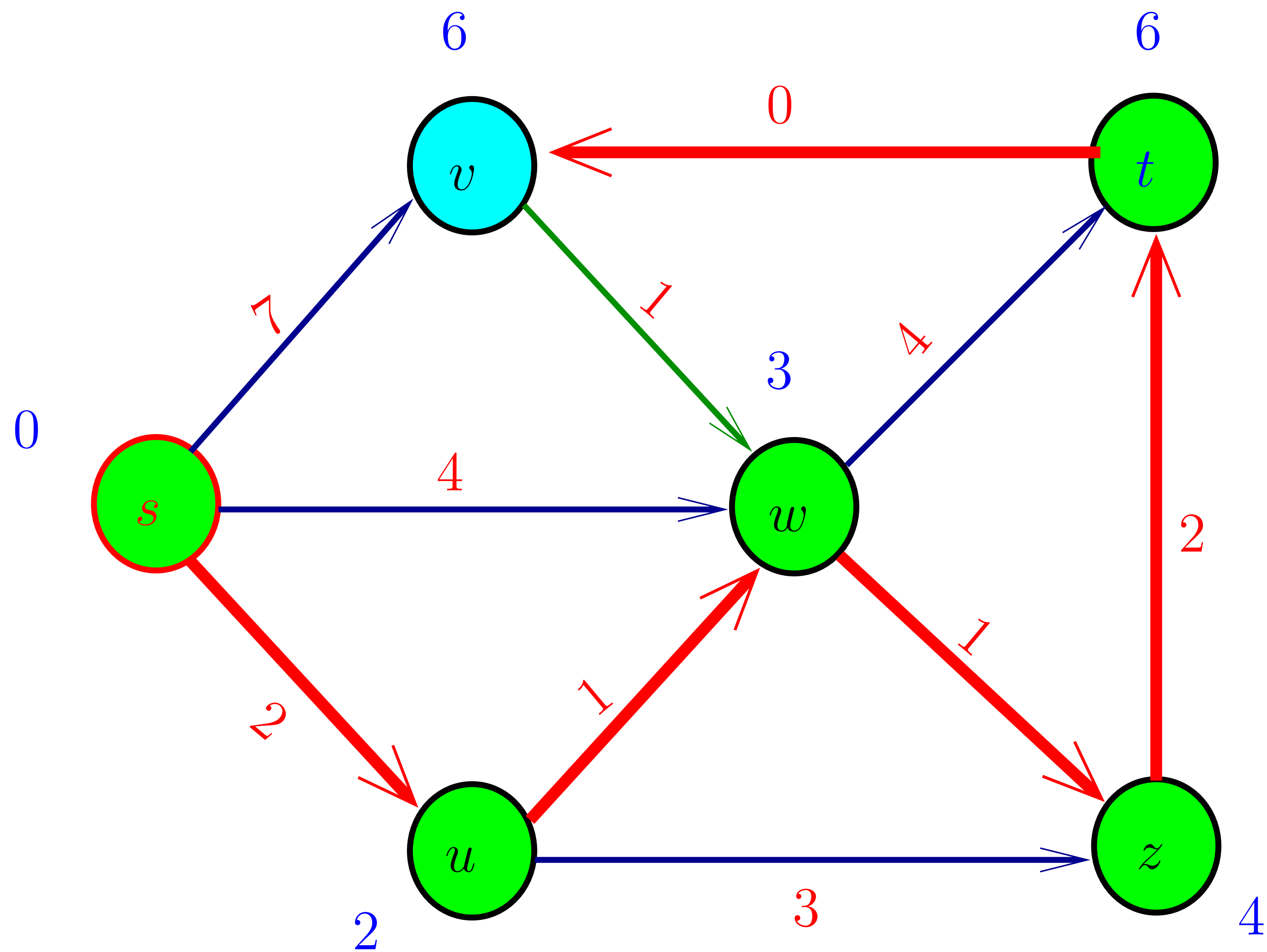
Simulação



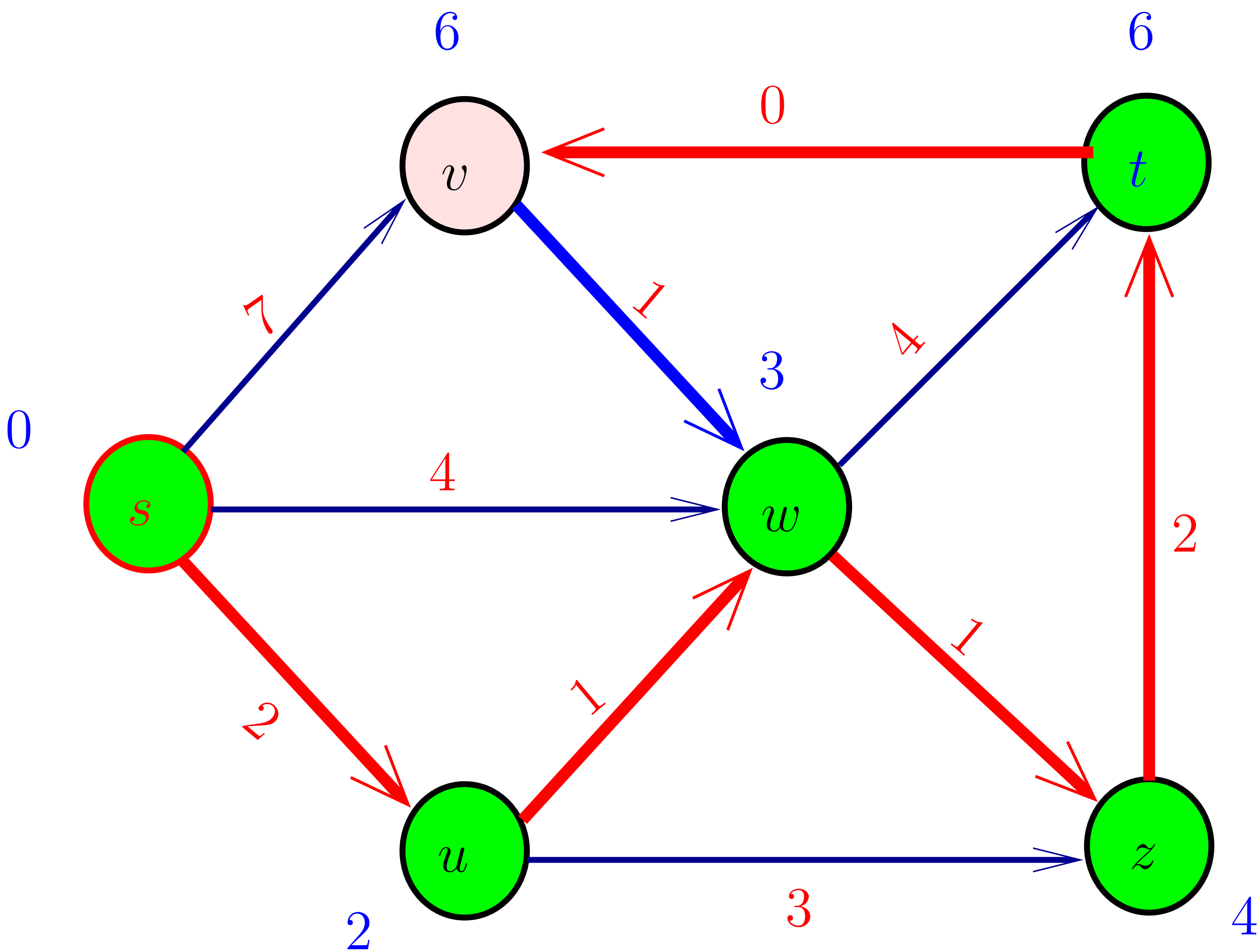
Simulação



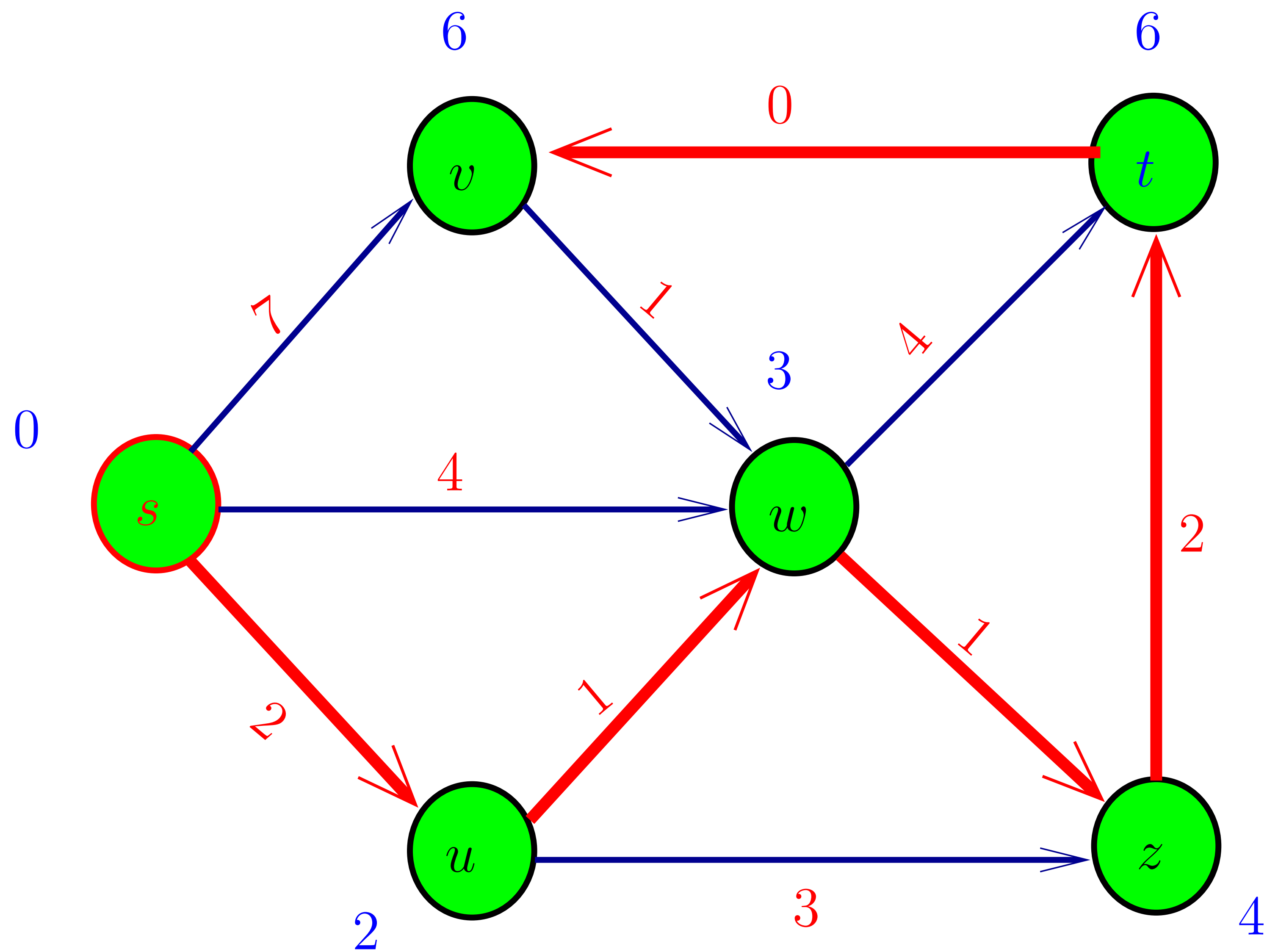
Simulação



Simulação



Simulação



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 $\quad \mathrm{O}(1) + 4\,\mathrm{O}(n) + \mathrm{O}(m) + \mathrm{O}(n^2)$

$$= \mathrm{O}(n^2)$$

Conclusão

O consumo de tempo do algoritmo DIJKSTRA é $O(n^2)$.

Implementação com min-heap

HEAP-DIJKSTRA (N, A, c, s) $\quad \triangleright\ c \geq 0$

para cada i em N **faça**

  $y(i) \leftarrow nC + 1$ $\quad \triangleright\ nC + 1$ faz o papel de ∞

  $\pi(i) \leftarrow \mathrm{NIL}$

$y(s) \leftarrow 0$

$Q \leftarrow$ BUILD-MIN-HEAP (N) $\quad \triangleright\ Q$ é um min-heap

enquanto $Q \neq \langle\rangle$ **faça**

  $i \leftarrow$ EXTRACT-MIN (Q)

  para cada ij em $A(i)$ **faça**

    valor $\leftarrow y(i) + c(ij)$

    se $y(j) >$ *valor* **então**

      DECREASE-KEY(*valor*, j, Q)

      $\pi(j) \leftarrow i$

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 $\quad 4\,\mathrm{O}(n) + 2\,\mathrm{O}(m) + \mathrm{O}(n \lg n) + \mathrm{O}(m\ \lg n)$

$= \mathrm{O}(m \lg n)$ (supondo (N, A) conexo)

Conclusão

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

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

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

AULA 6

Caminhos de custo mínimo

PF 6.1, 6.2, 6.3, 6.4, 6.5

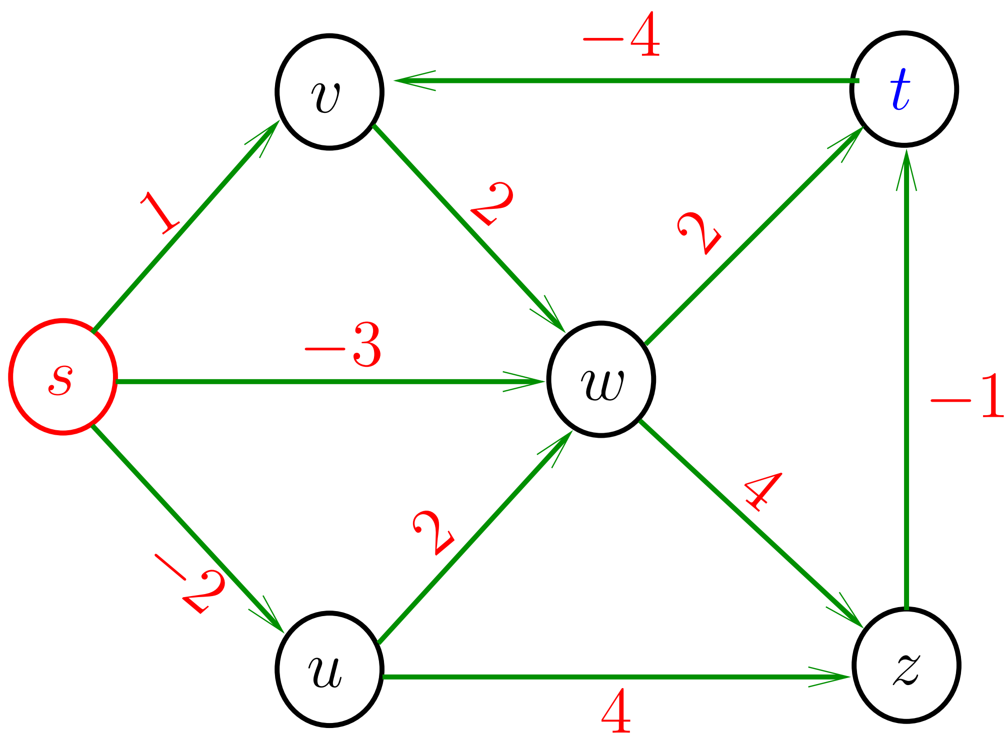
Problema

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

Problema

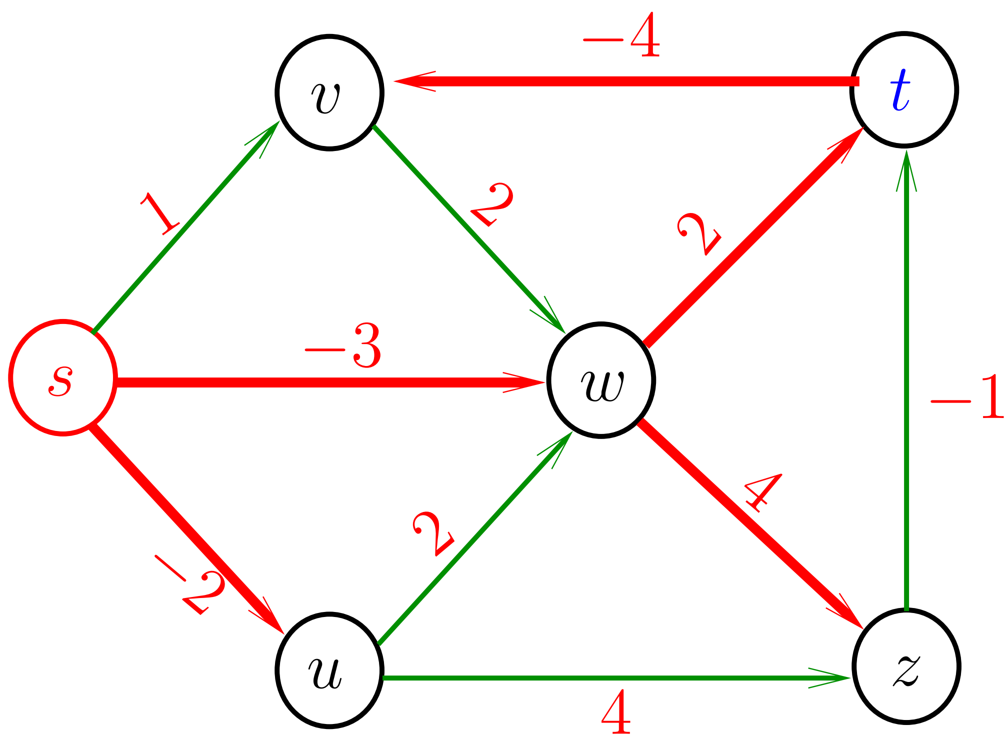
Problema do caminho de custo mínimo: Dada uma rede (N, A, c) com função-custo $c : A \to \mathbb{Z}$ e um nó s, encontrar, para cada nó t, um caminho de custo mínimo de s a t.

Entra:



Problema

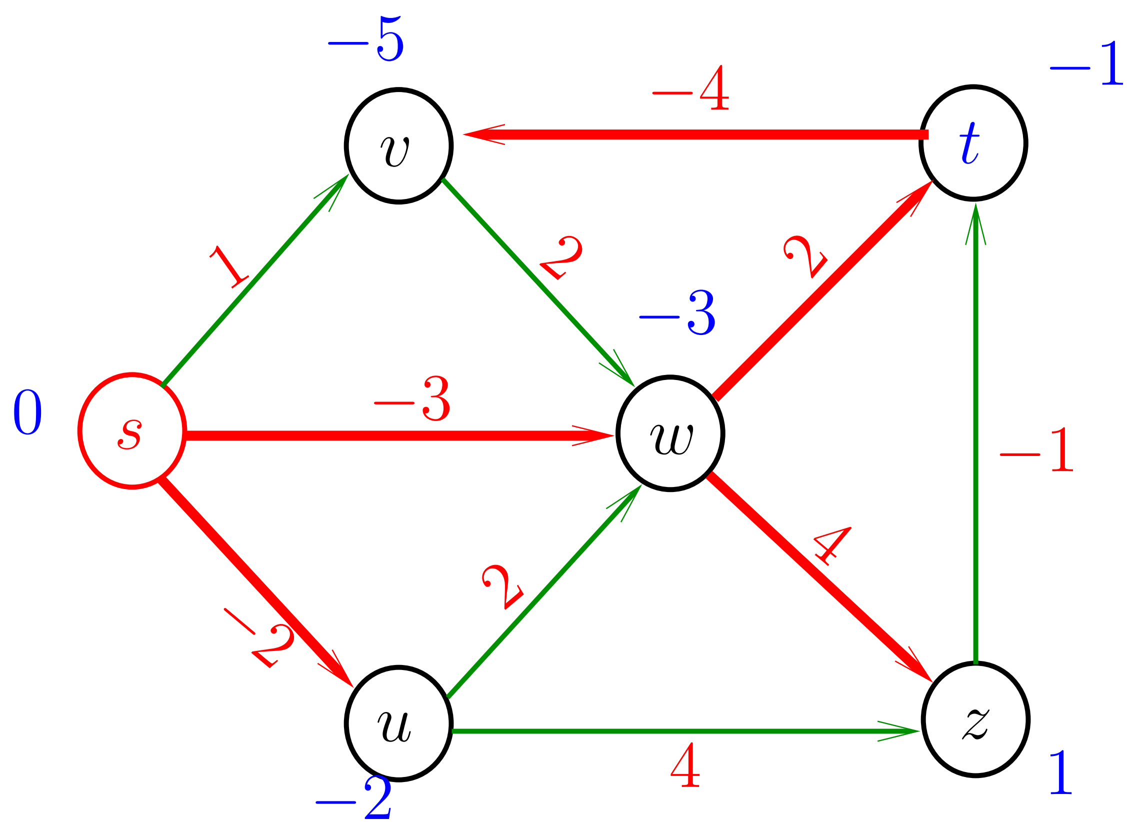
Problema do caminho de custo mínimo: Dada uma rede (N, A, c) com função-custo $c : A \to \mathbb{Z}$ e um nó s, encontrar, para cada nó t, um caminho de custo mínimo de s a t.
Sai:



Propriedade de c-Potenciais

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

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

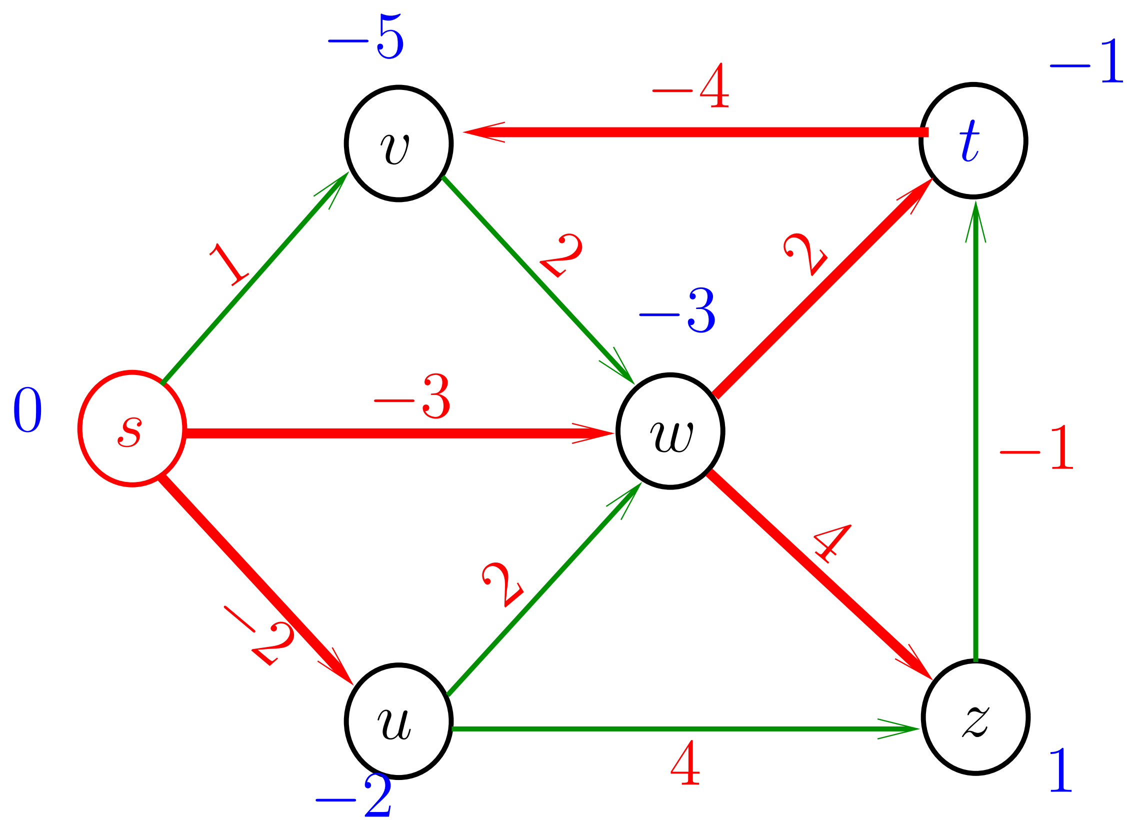


Conseqüência

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

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

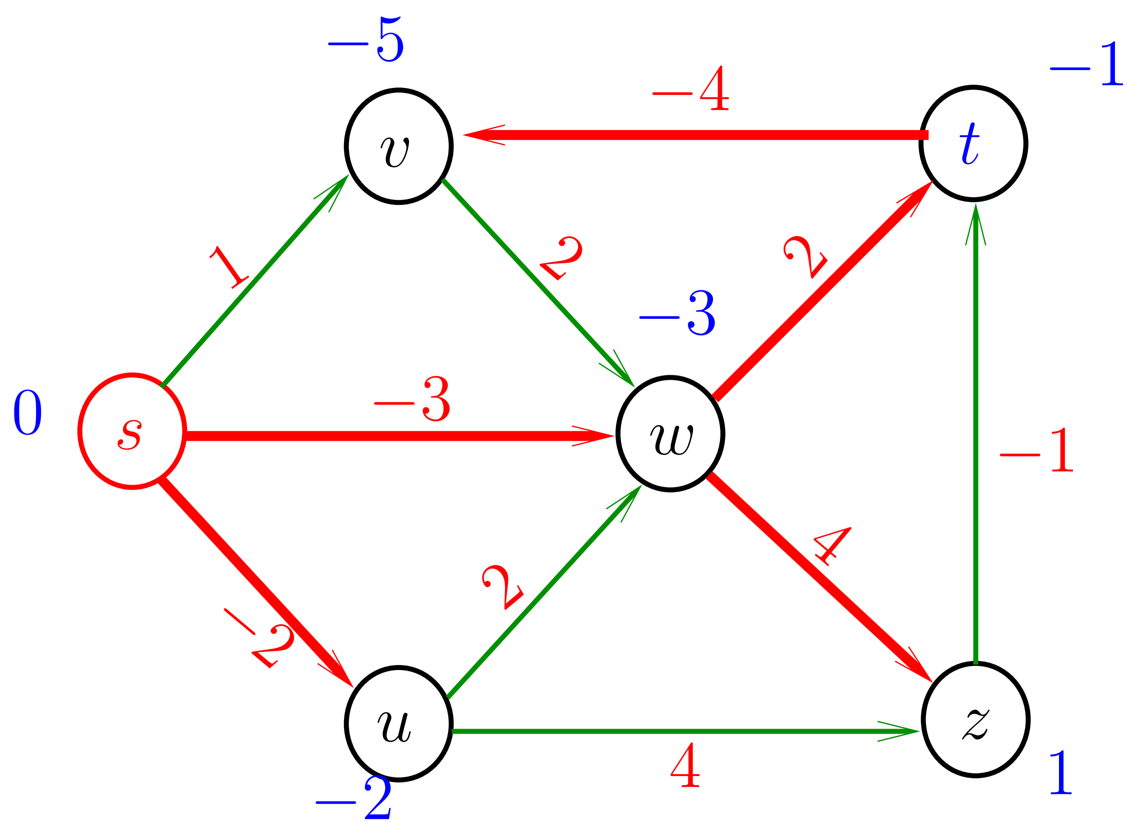
então P é uma caminho mínimo e y é um c-potencial tal que o valor de $y(t) - y(s)$ é máximo.



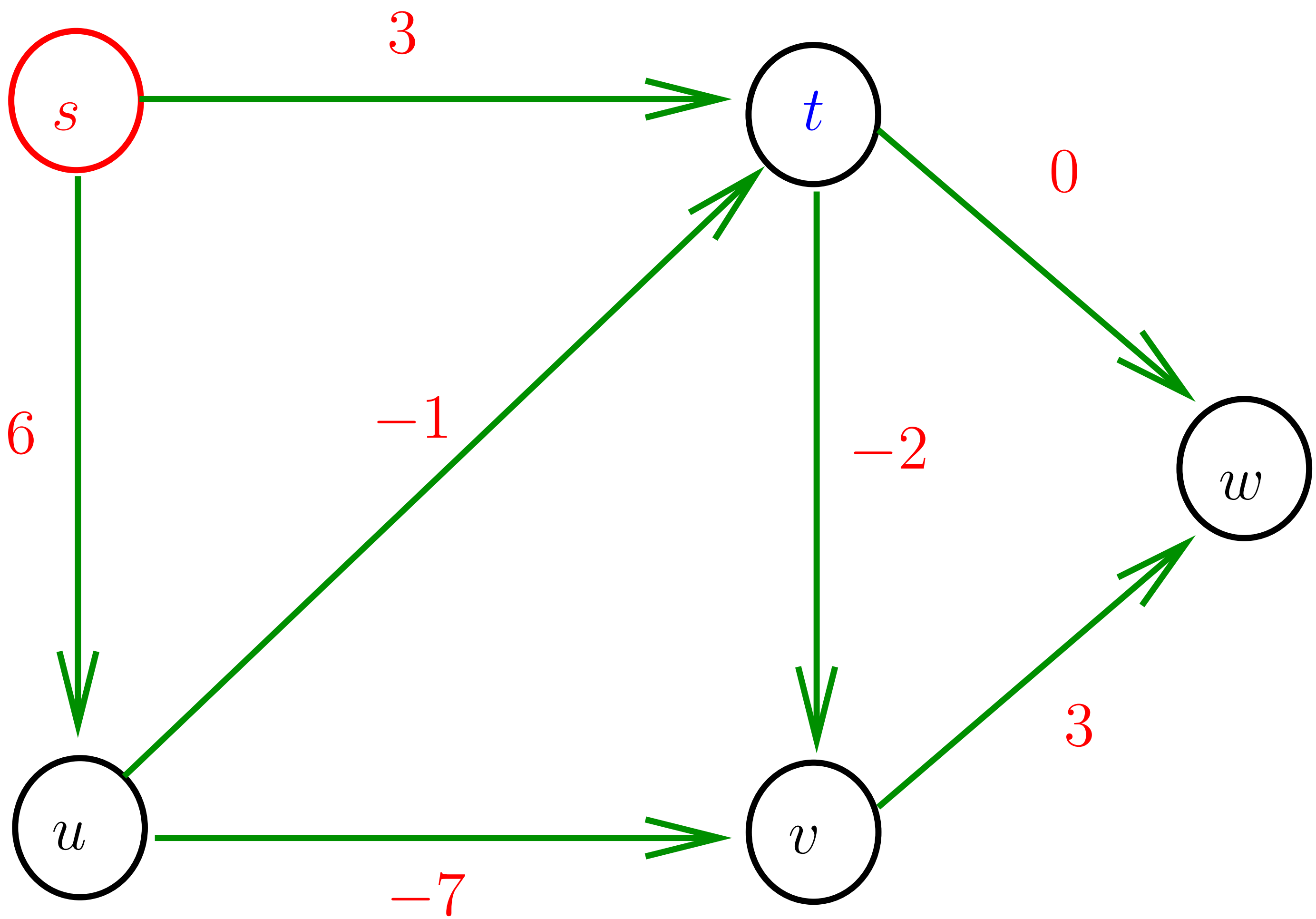
Potenciais ótimos

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

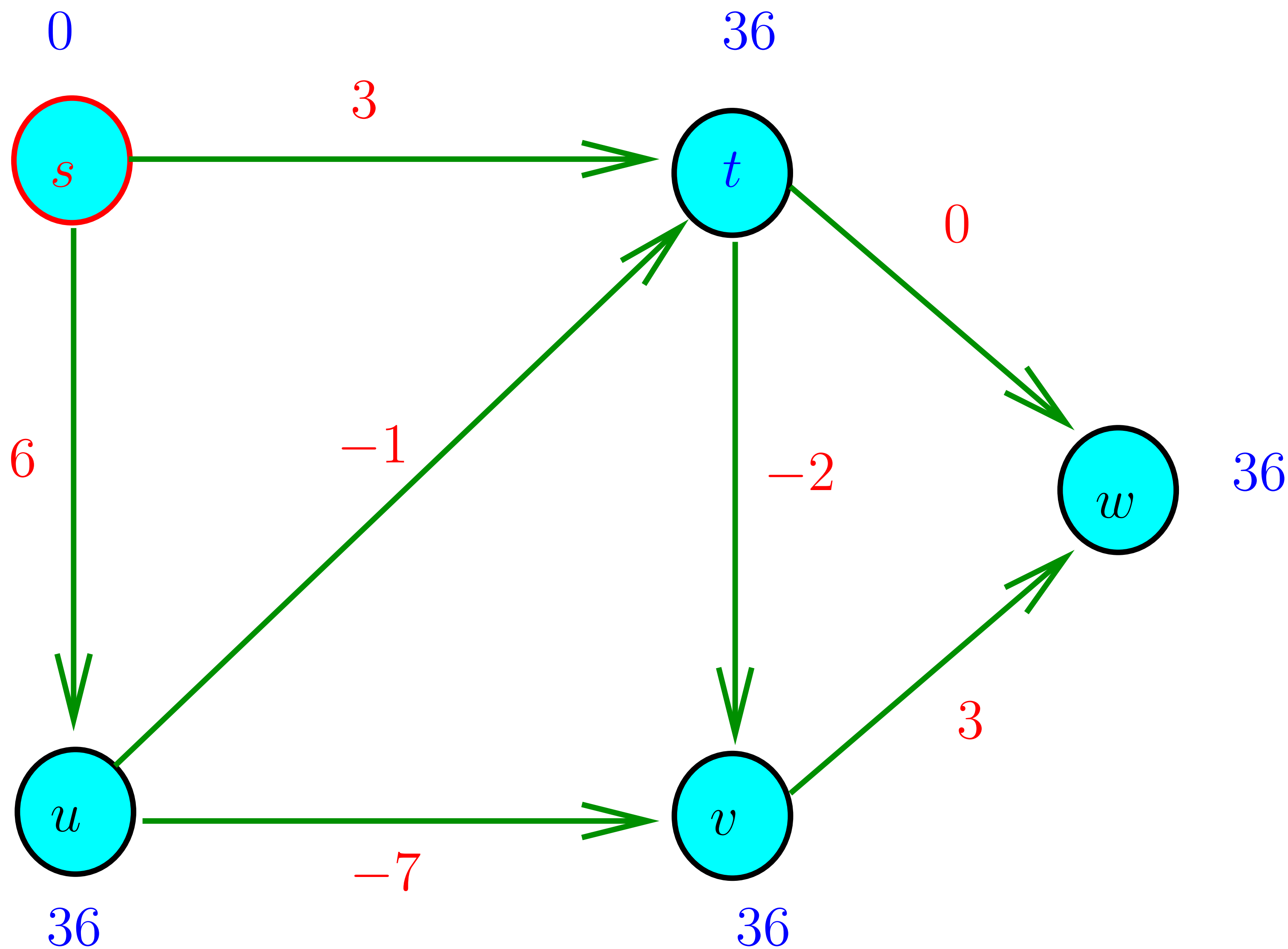
$$c(P) = y(t) - y(s),$$



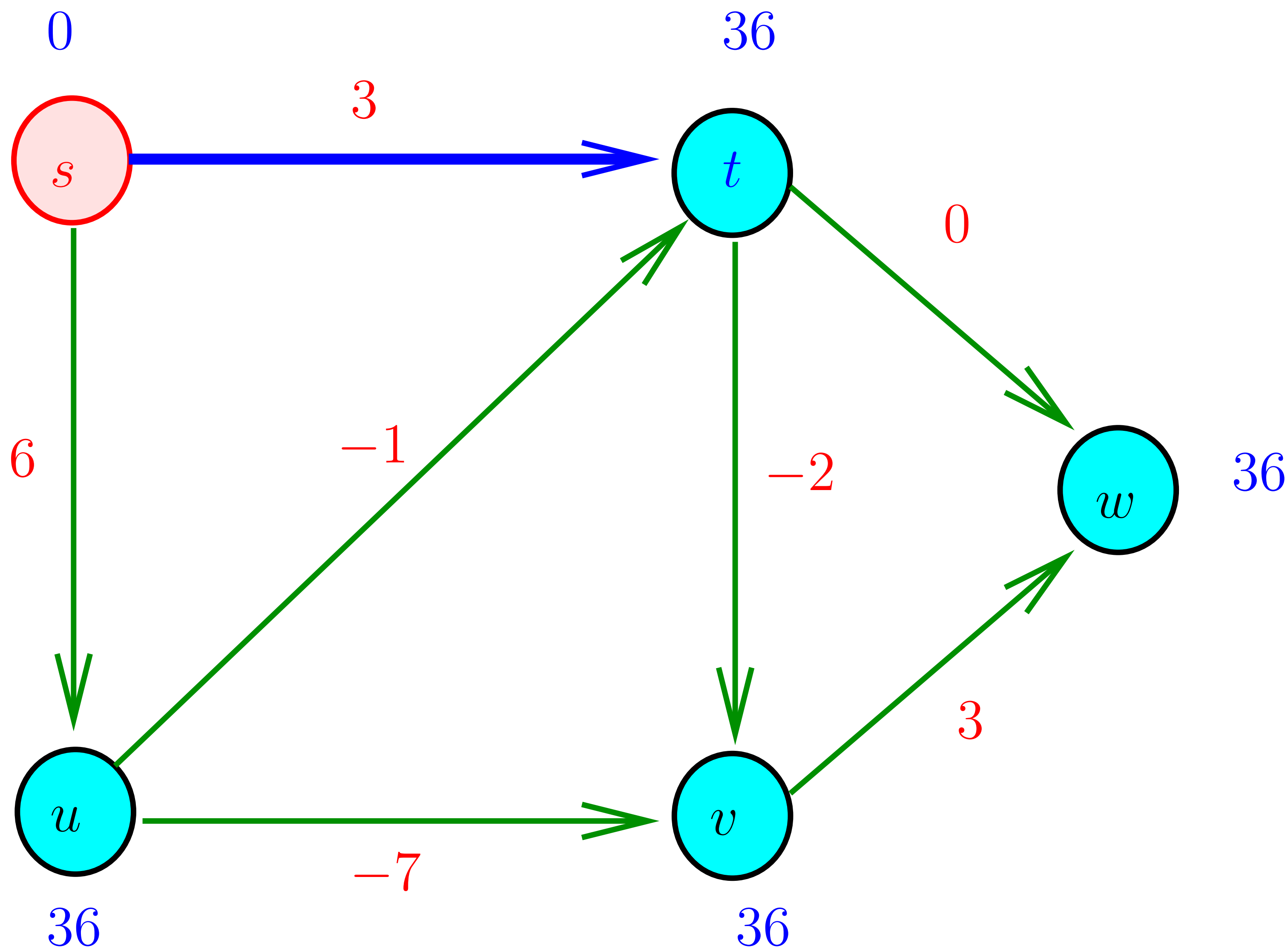
Apelemos para Dijkstra



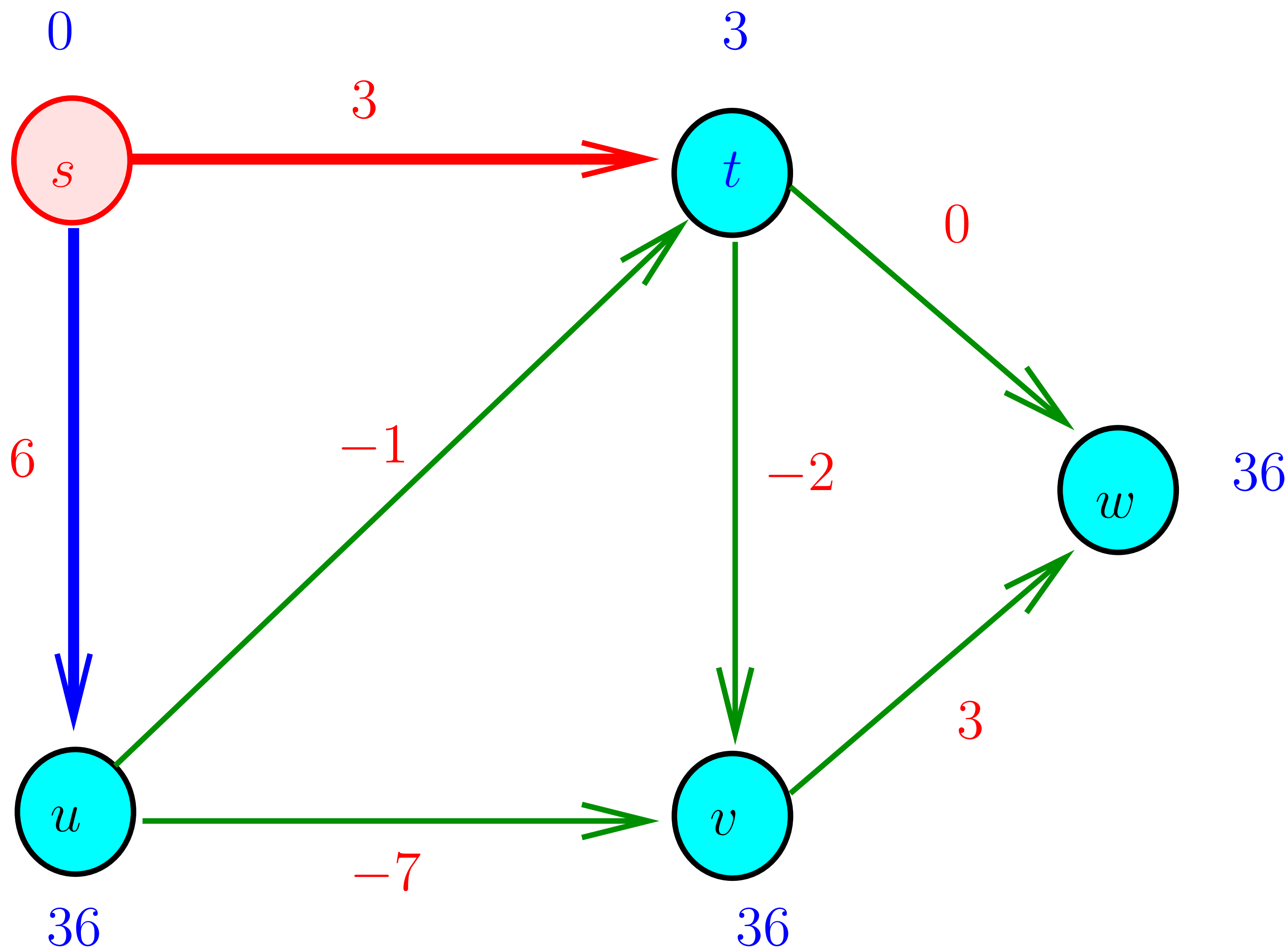
Apelemos para Dijkstra



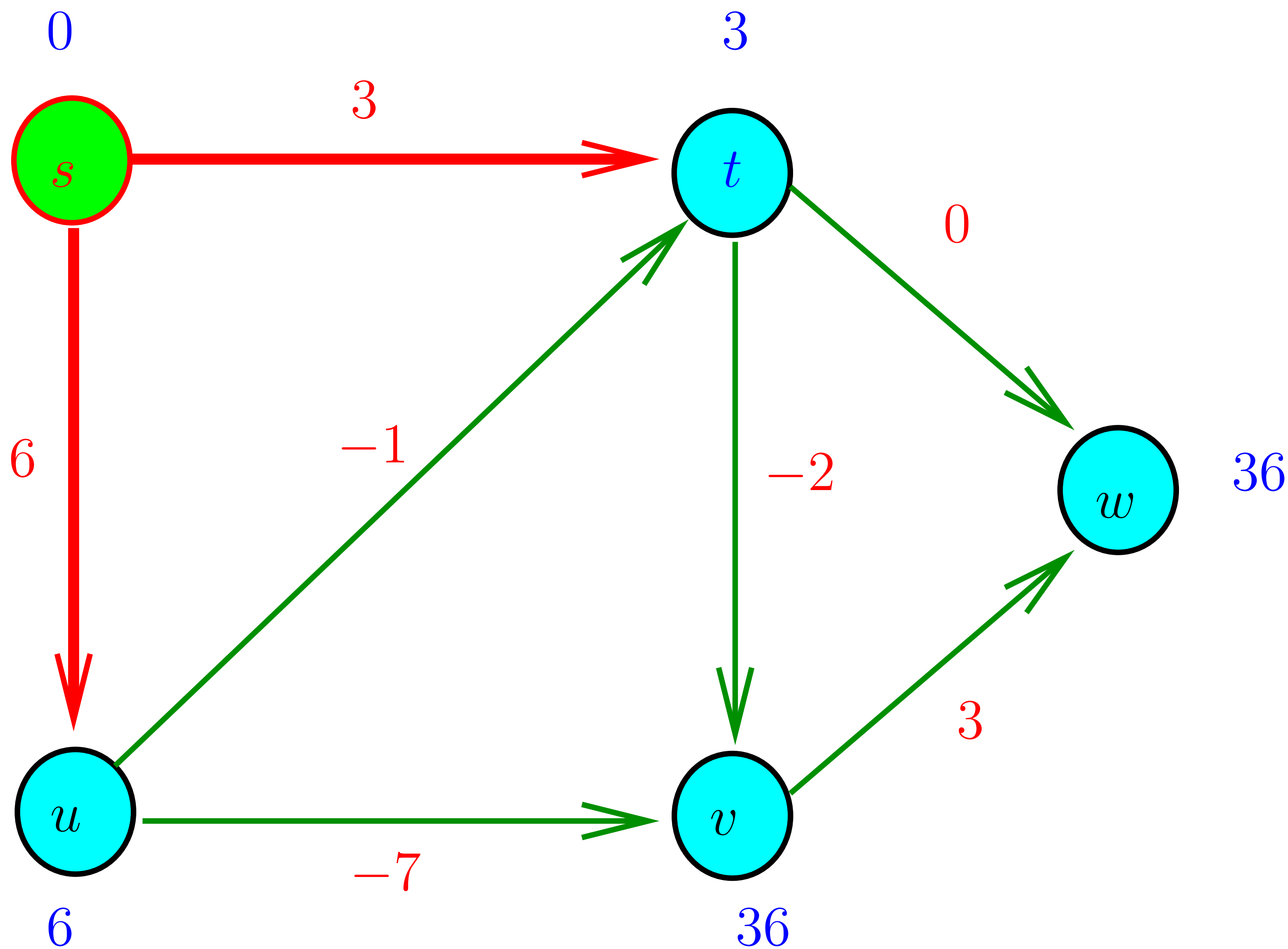
Apelemos para Dijkstra



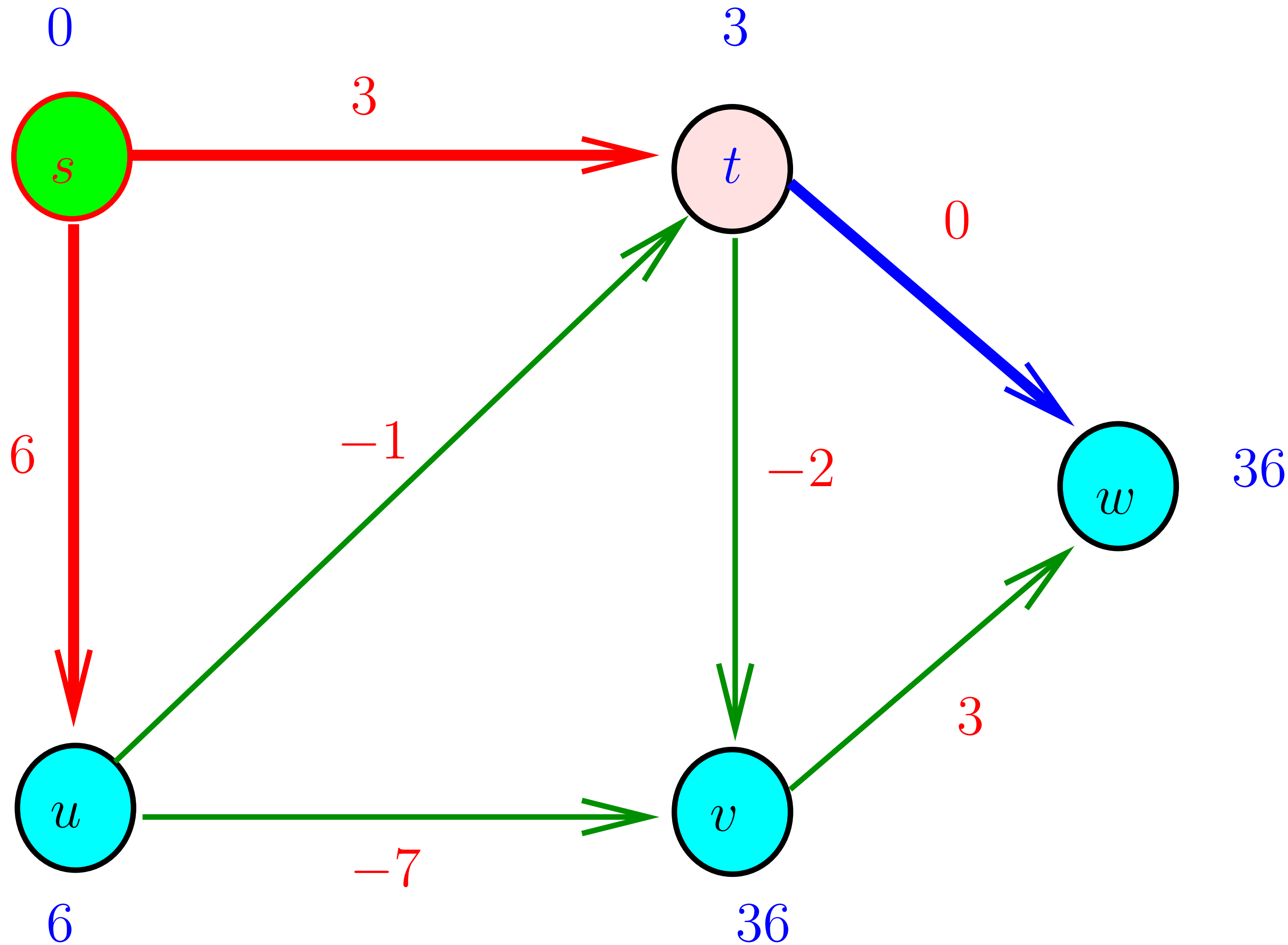
Apelemos para Dijkstra



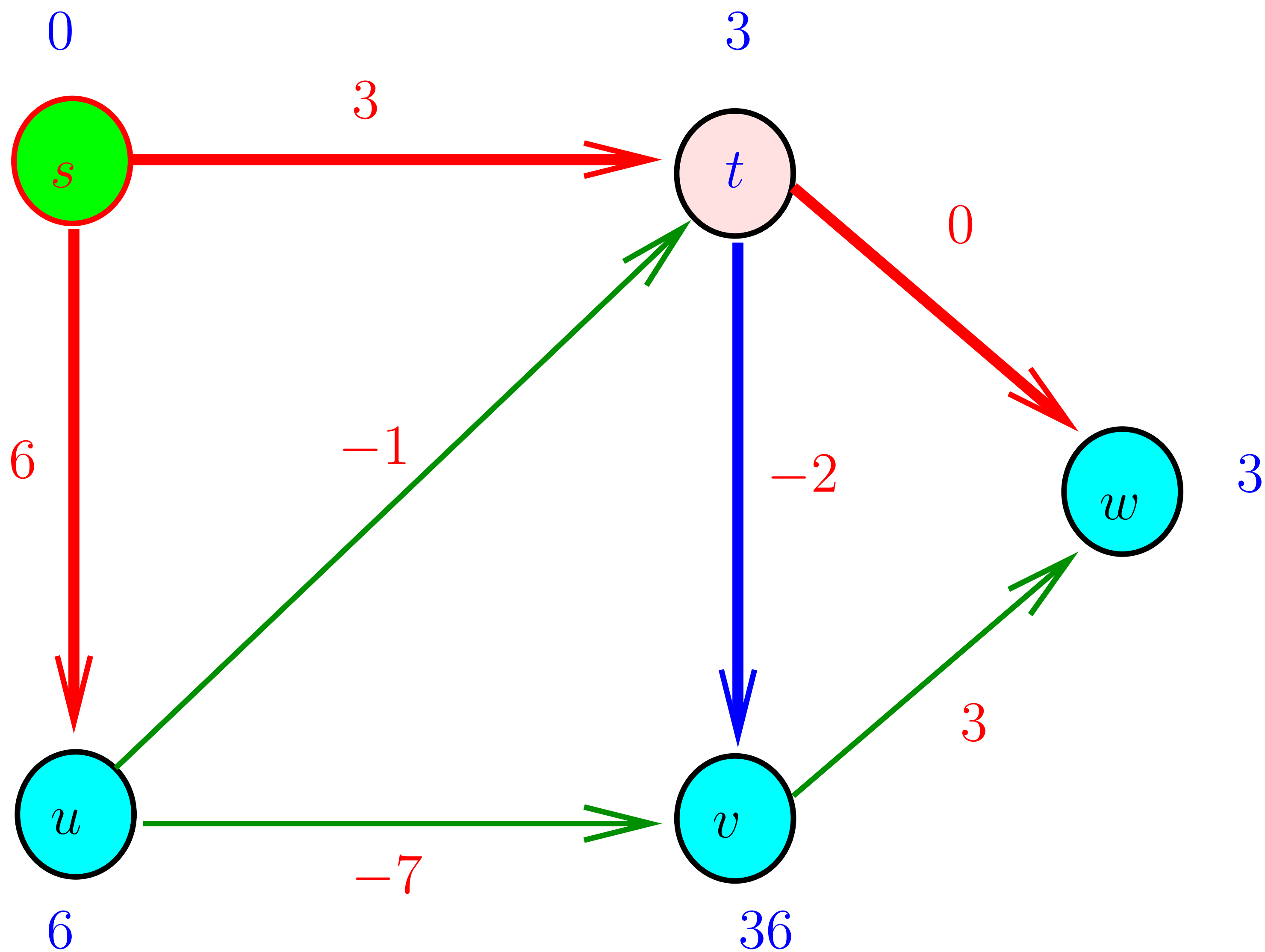
Apelemos para Dijkstra



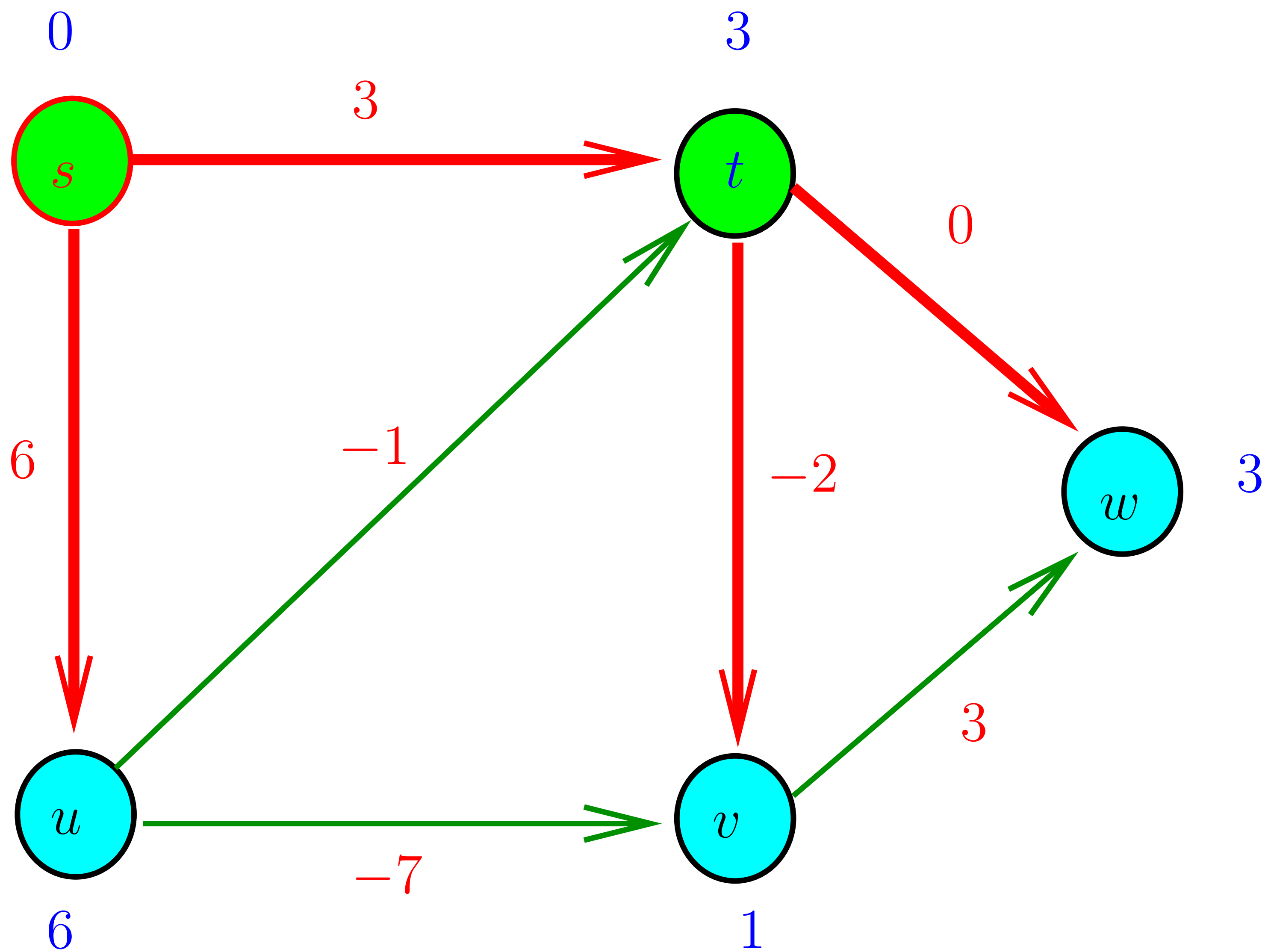
Apelemos para Dijkstra



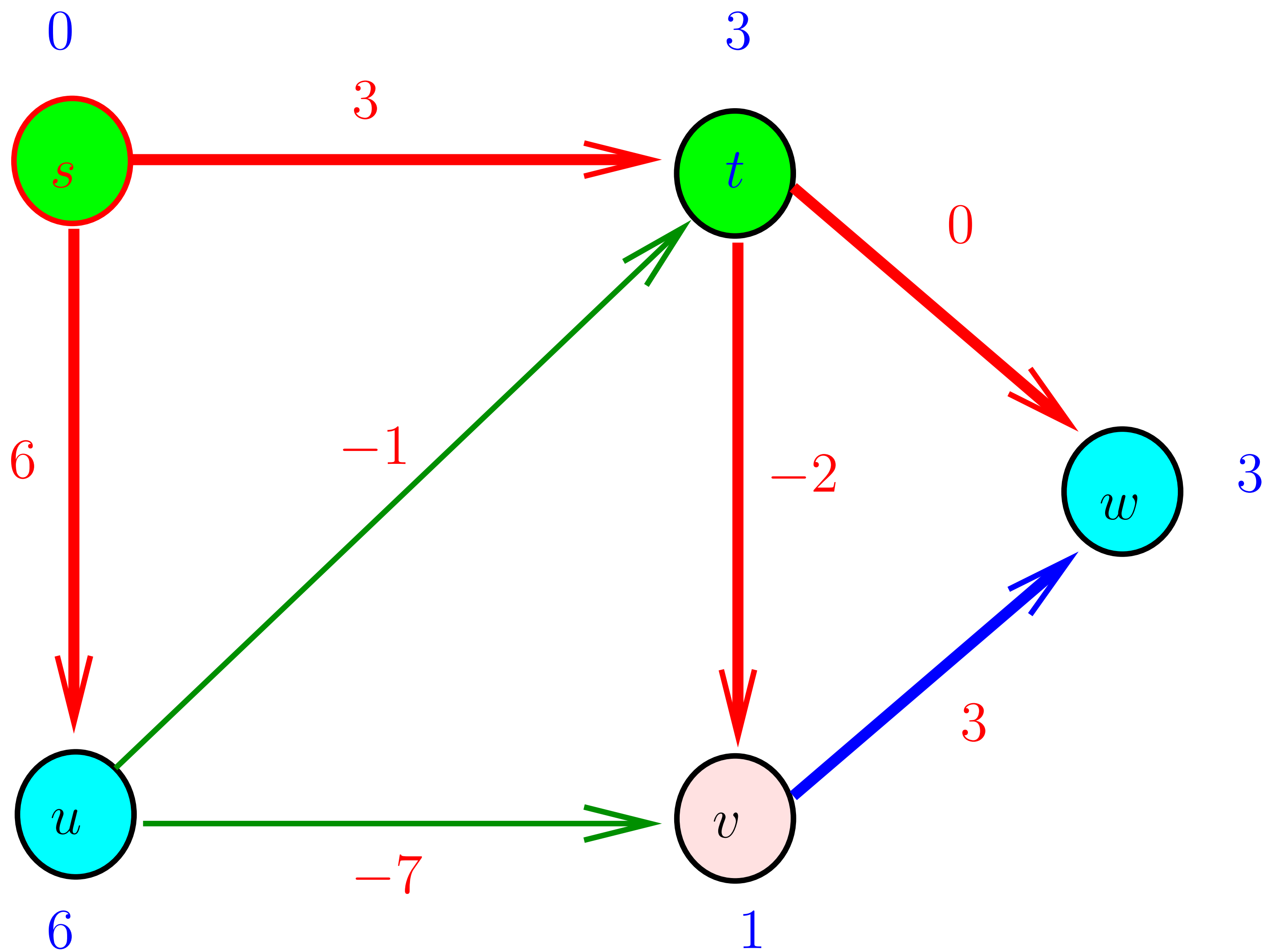
Apelemos para Dijkstra



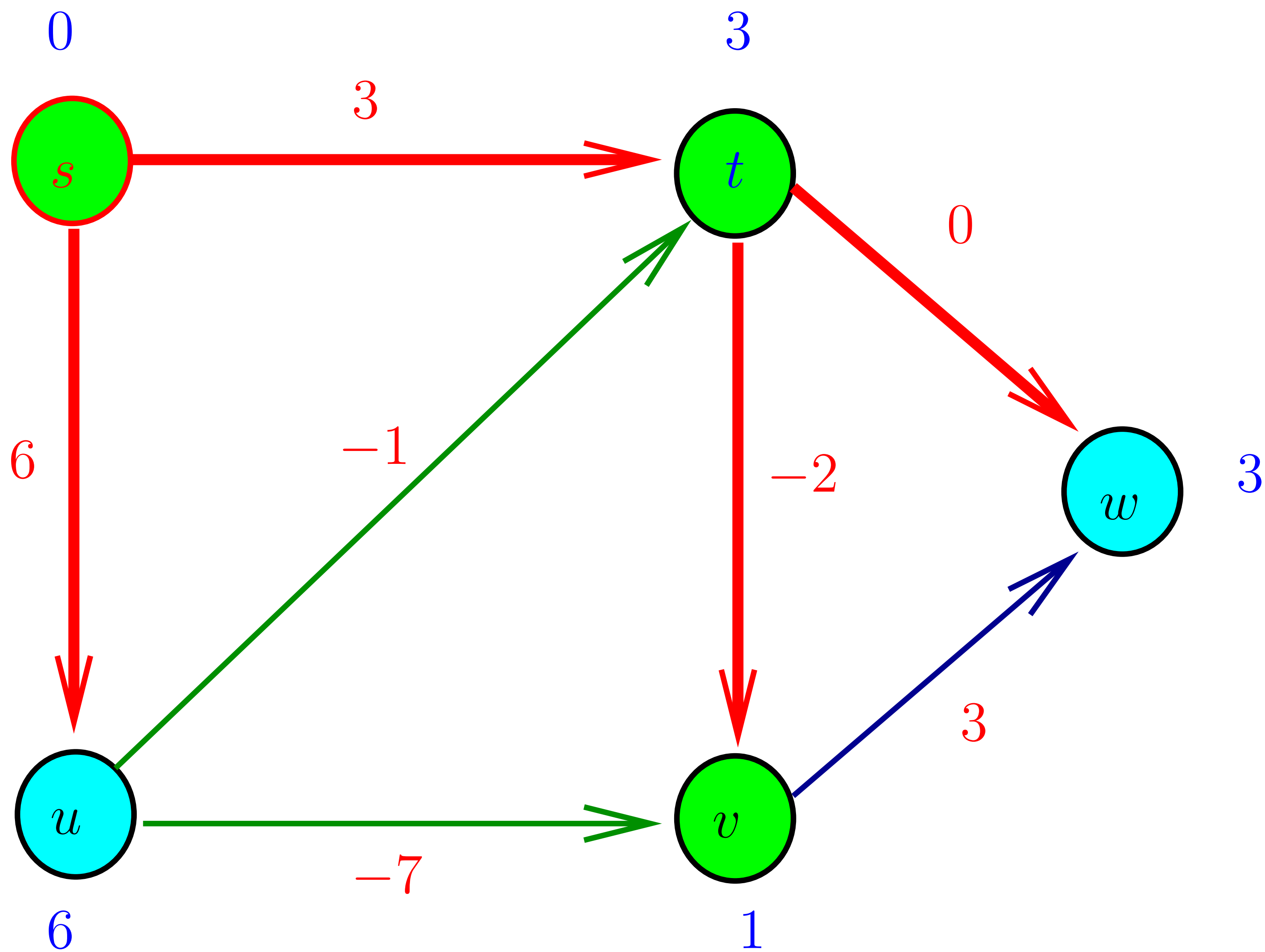
Apelemos para Dijkstra



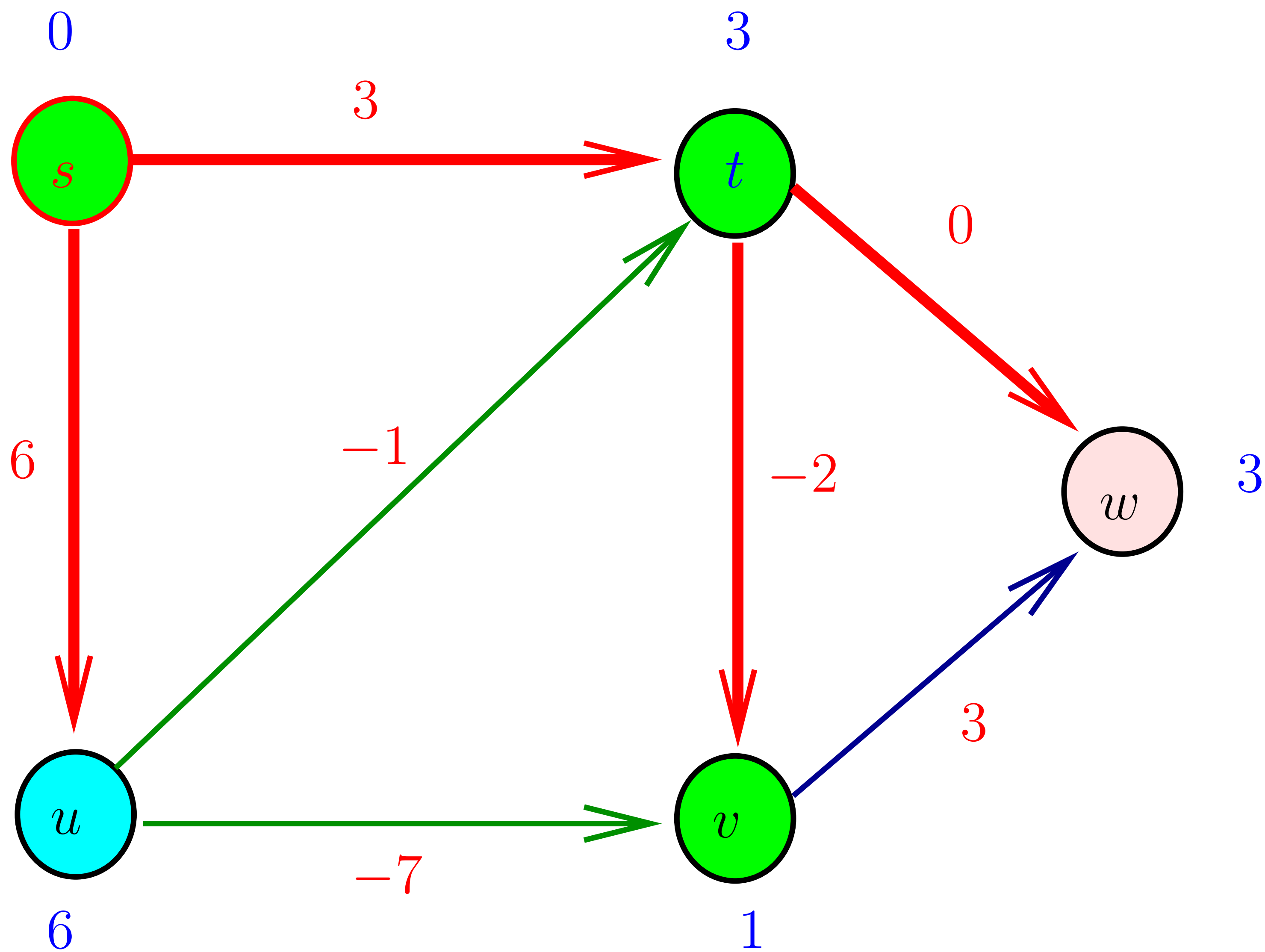
Apelemos para Dijkstra



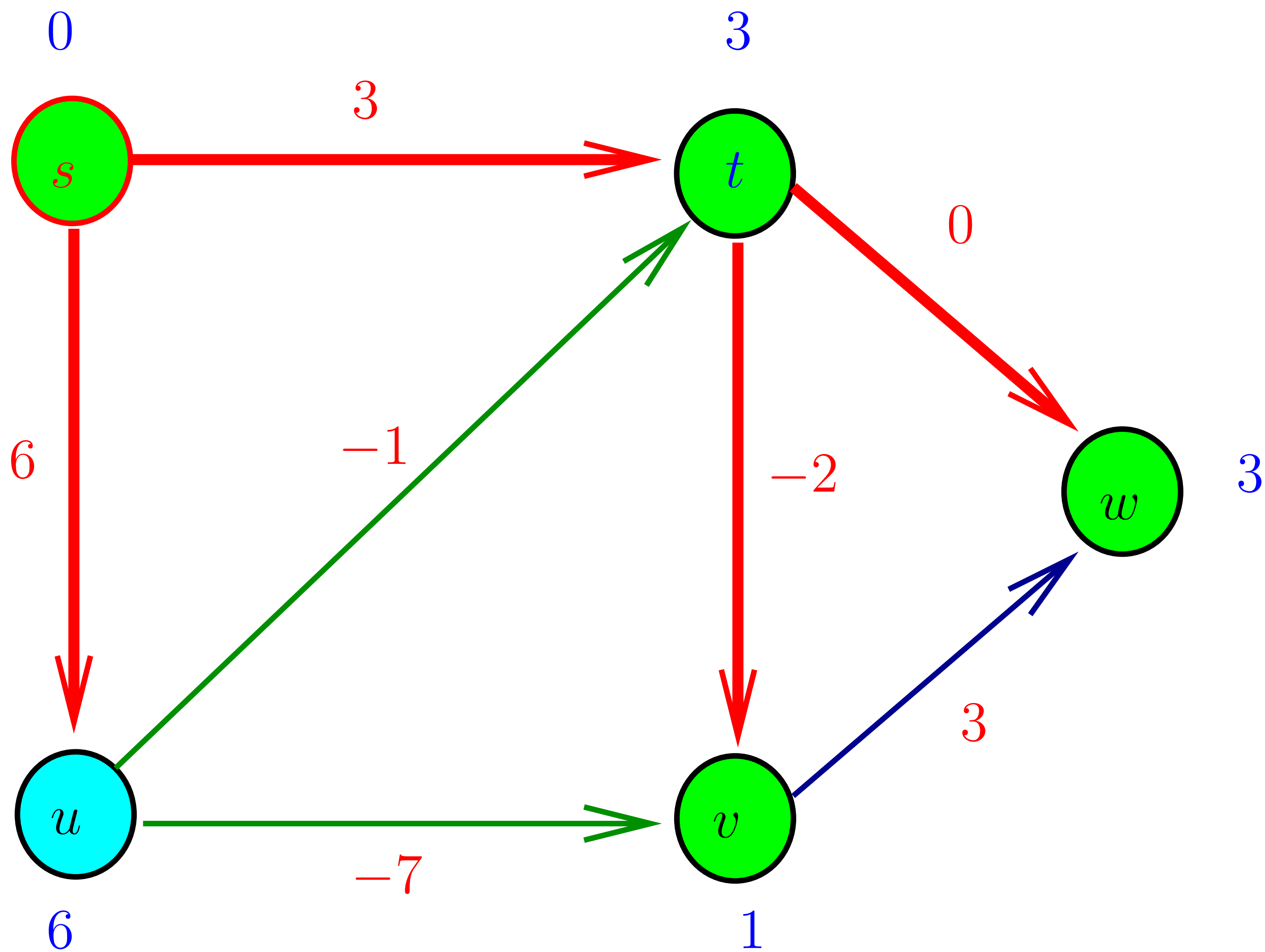
Apelemos para Dijkstra



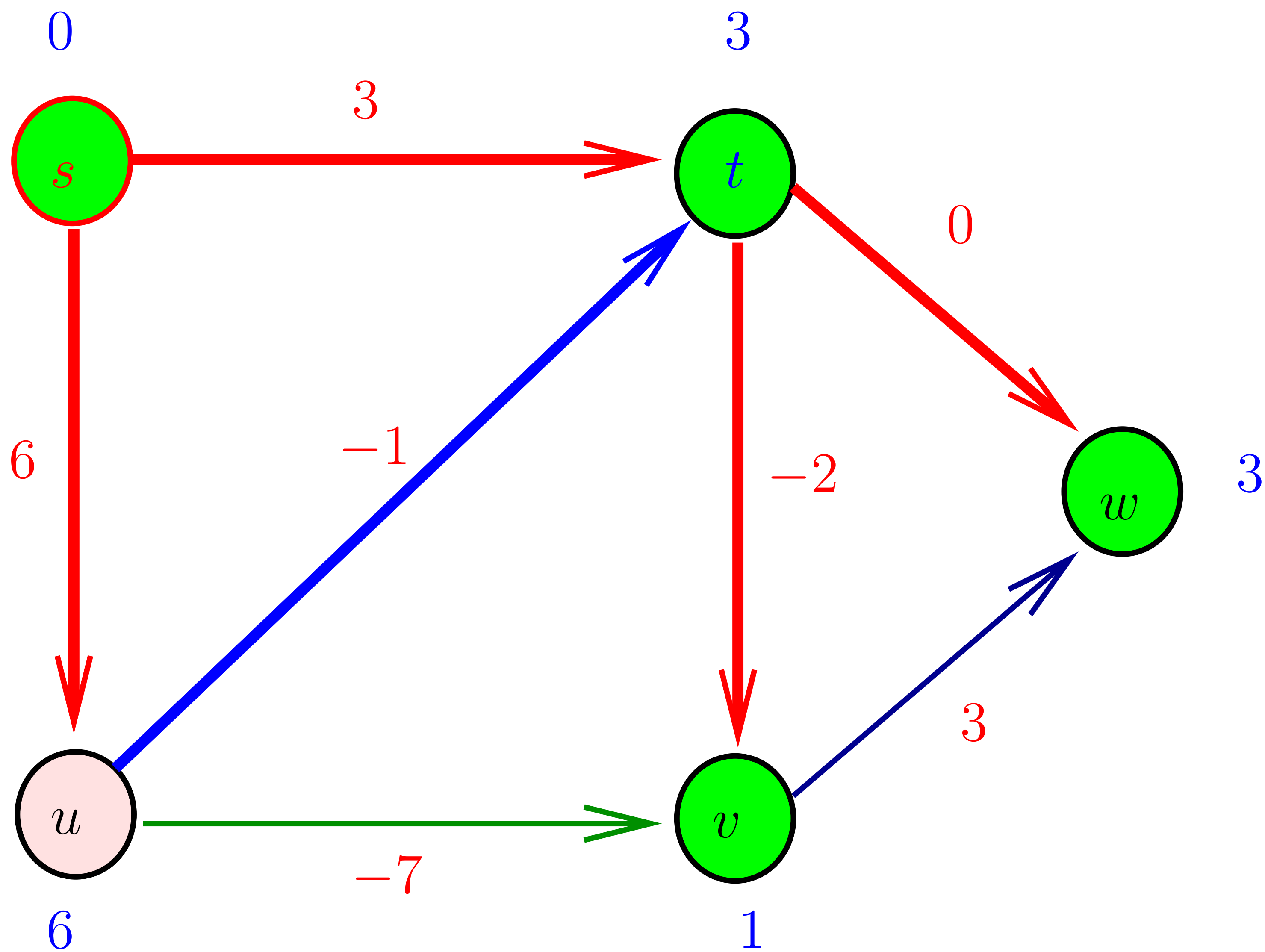
Apelemos para Dijkstra



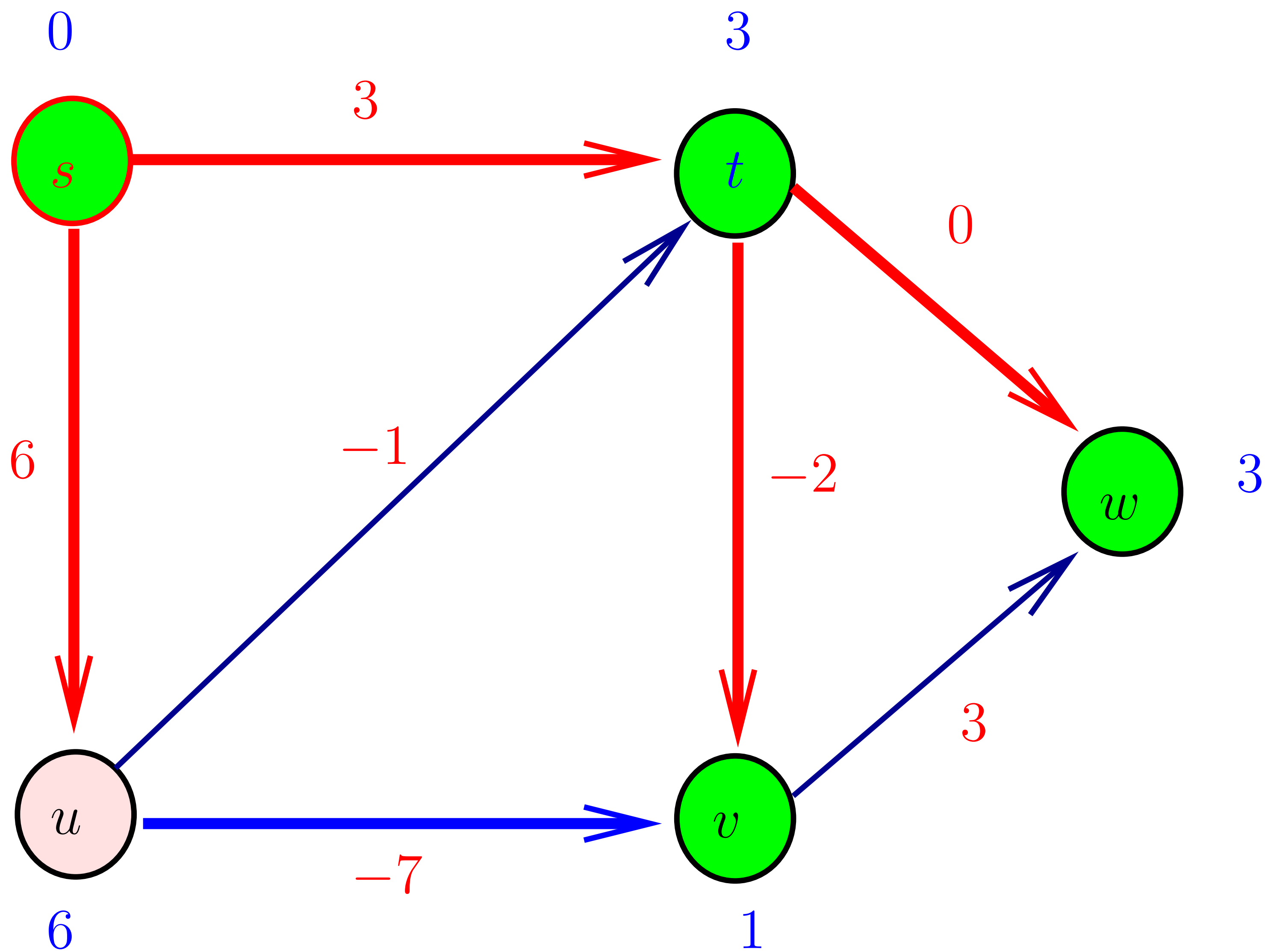
Apelemos para Dijkstra



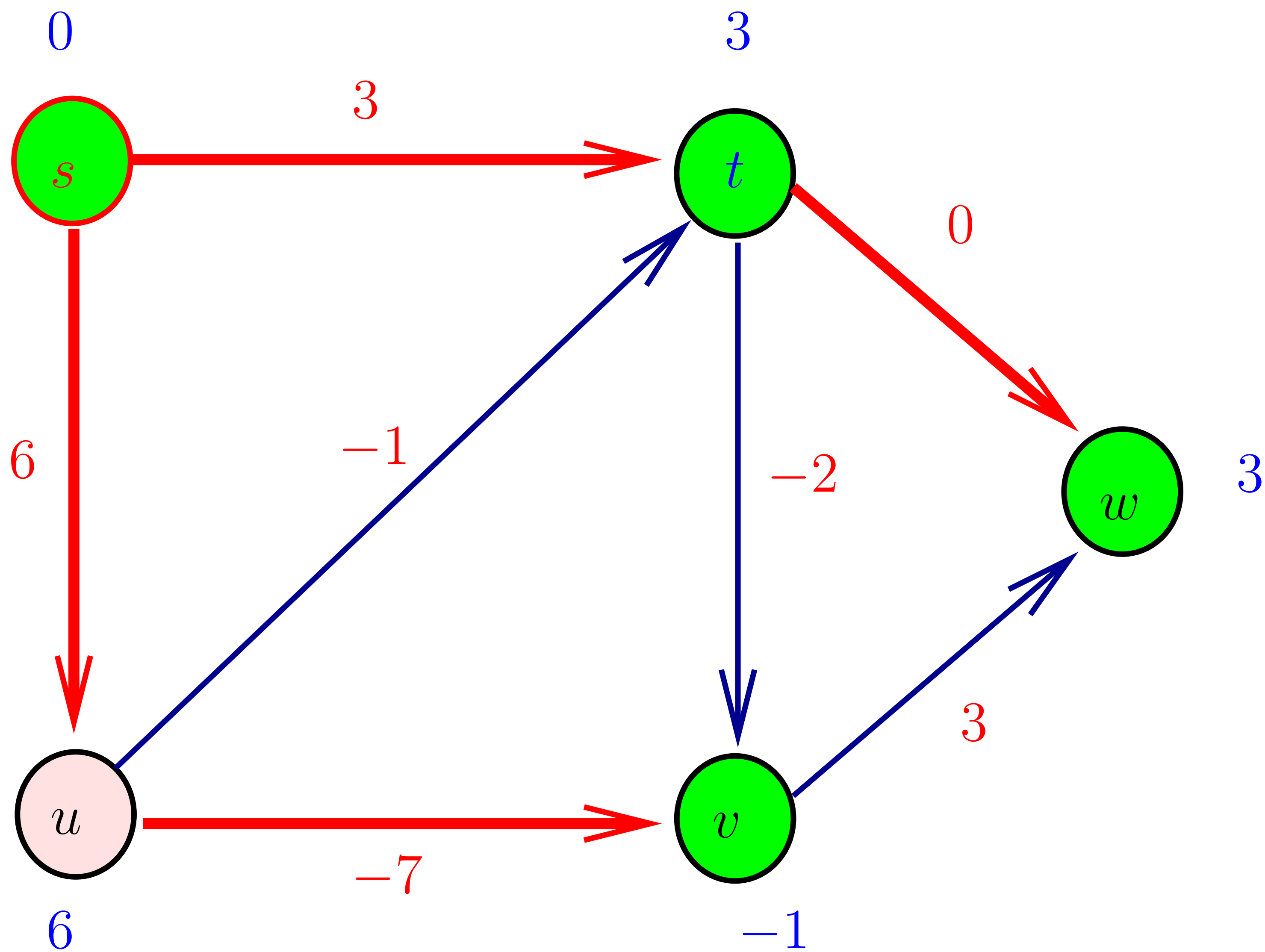
Apelemos para Dijkstra



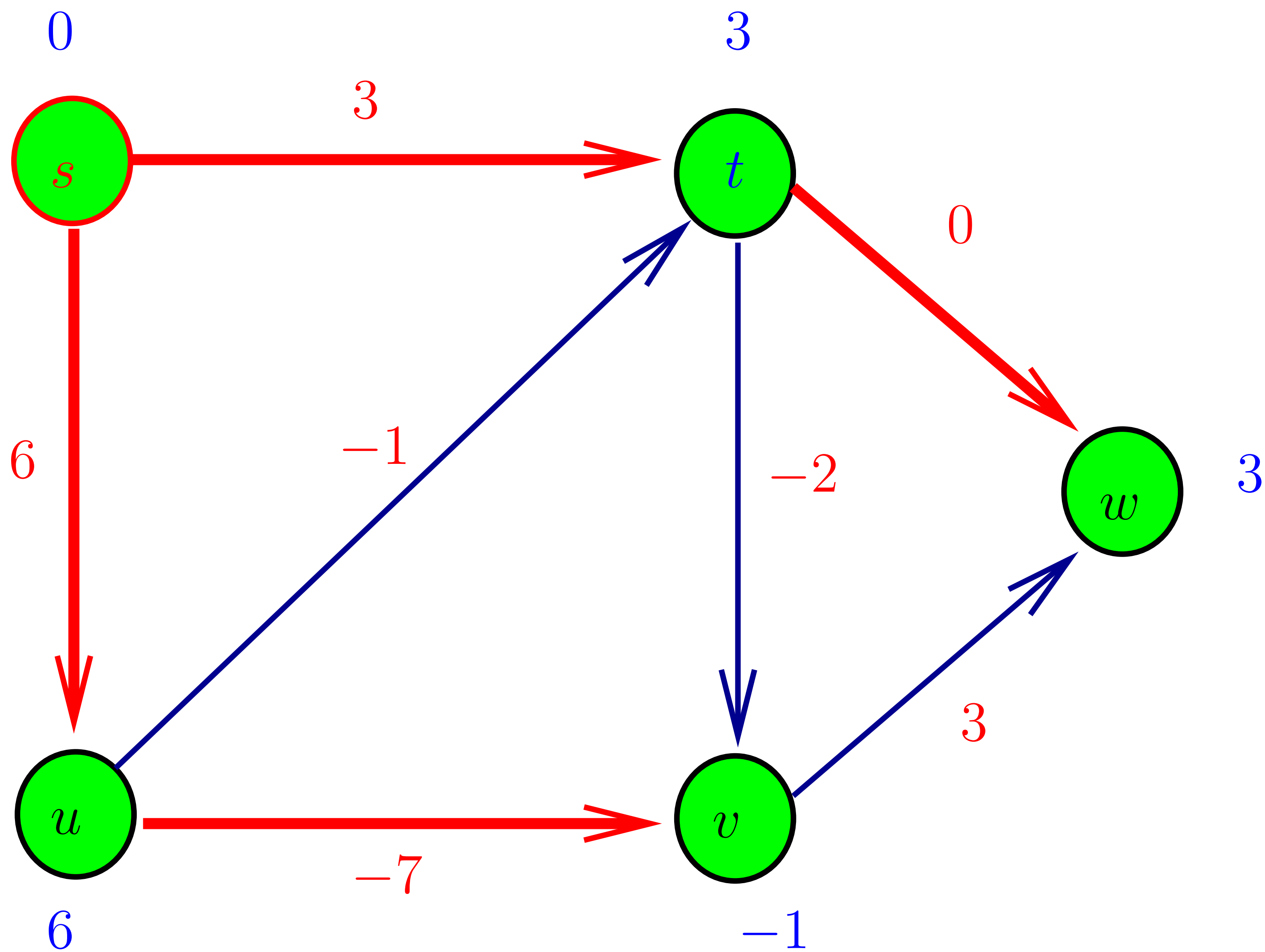
Apelemos para Dijkstra



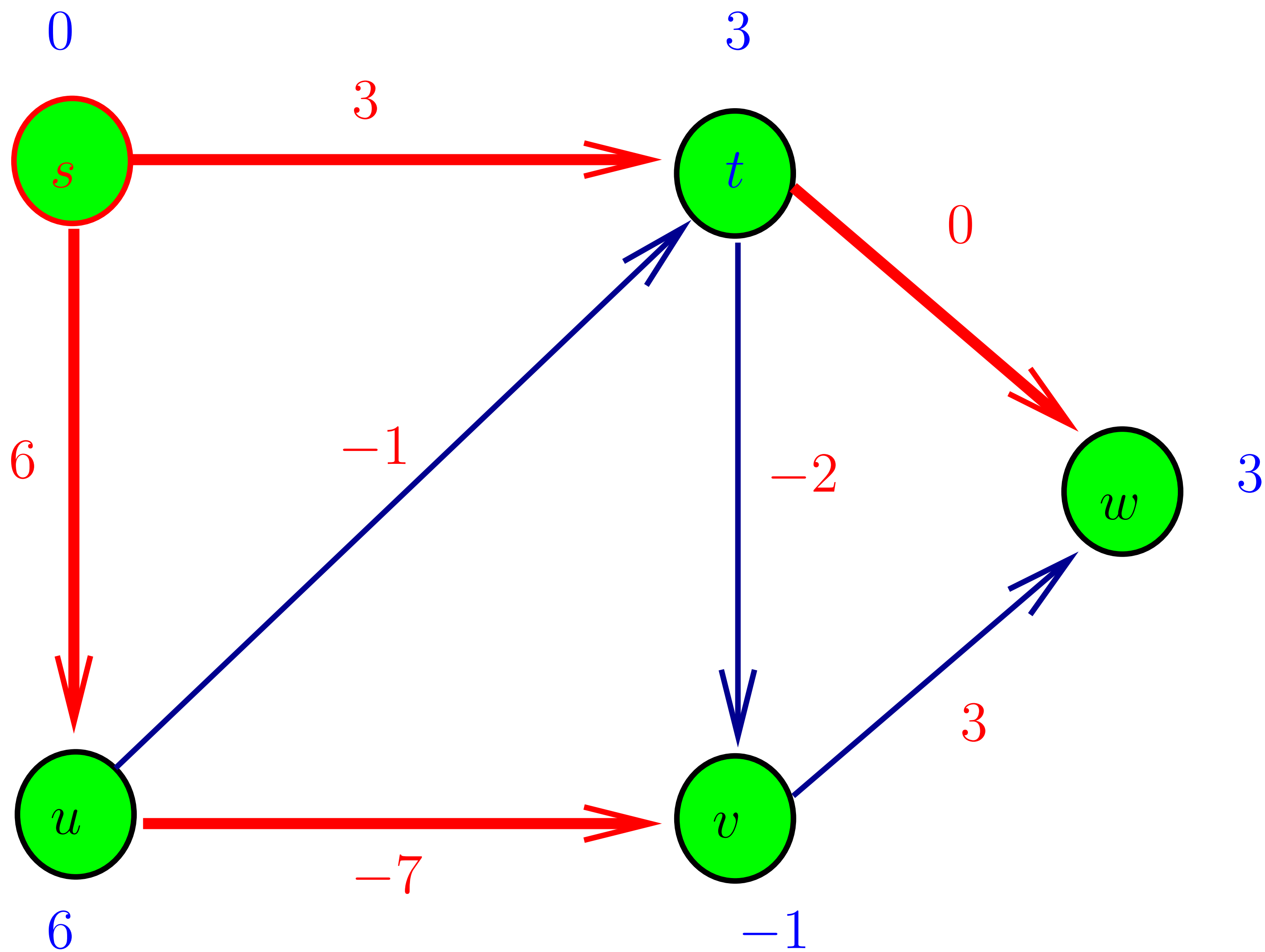
Apelemos para Dijkstra



Apelemos para Dijkstra



Apelemos para Dijkstra



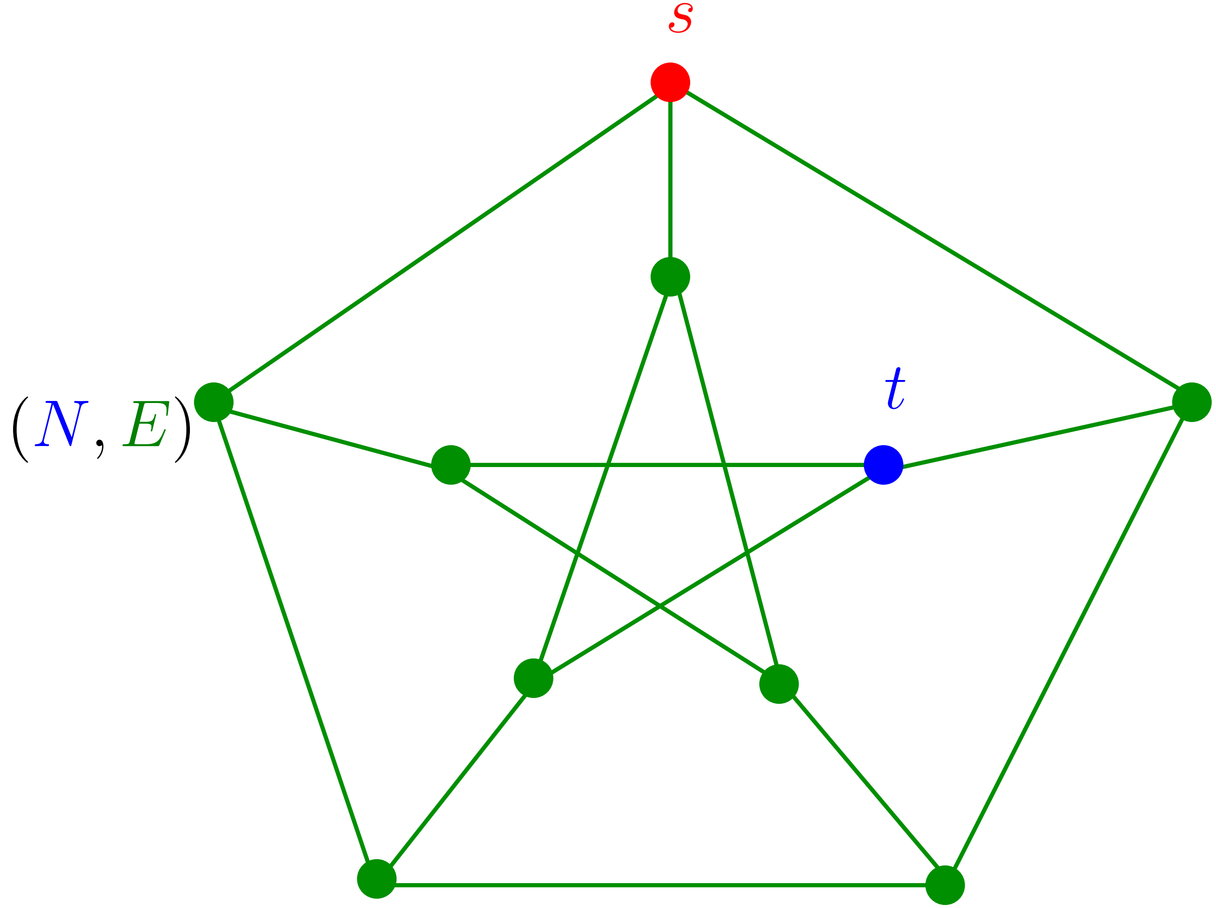
O sw-caminho mínimo tem custo 2 e **não** 3…

Caminhos hamiltonianos

Problema: Dados nós s e t de grafo não-orientado (N, E) encontrar um caminho hamiltoniano entre s e t.

Caminhos hamiltonianos

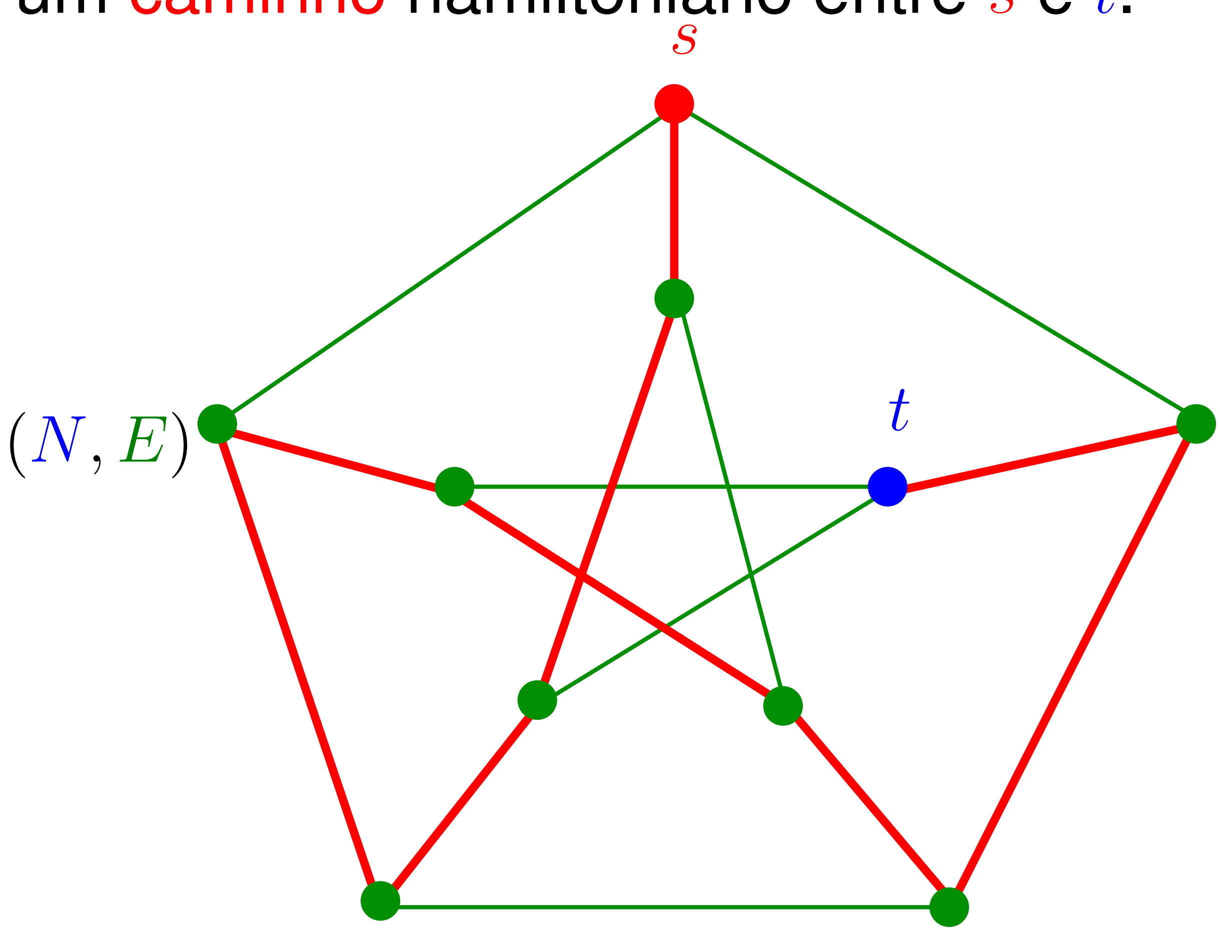
Problema: Dados nós s e t de grafo não-orientado (N, E) encontrar um caminho hamiltoniano entre s e t.



Este problema é NP-difícil.

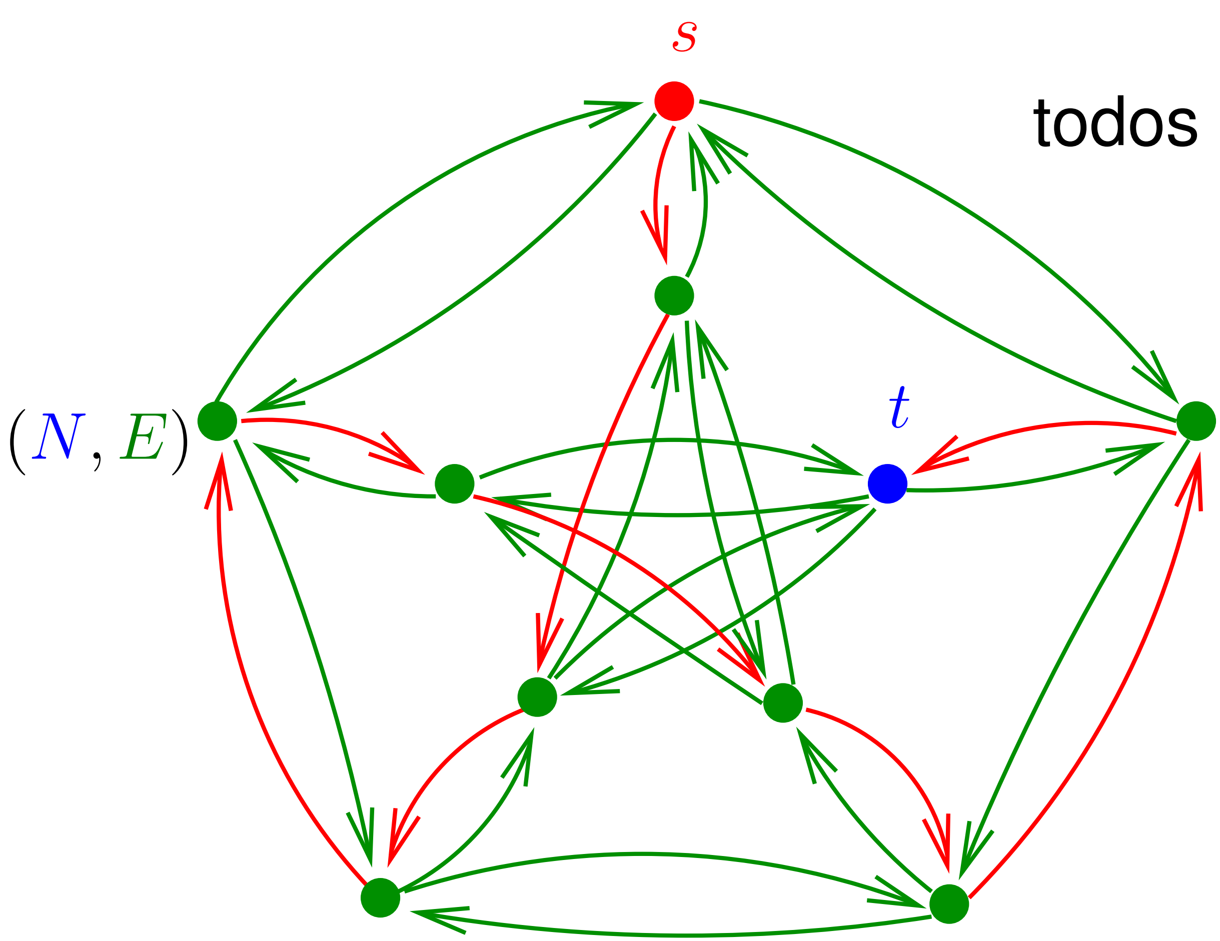
Caminhos hamiltonianos

Problema: Dados nós s e t de grafo não-orientado (N, E) encontrar um caminho hamiltoniano entre s e t.



Este problema é NP-difícil.

Redução polinomial



todos custos = -1

(N, E) possui um st-caminho hamiltoniano

$\Leftrightarrow$

(N, A) possui um st-caminho de custo $-(n-1)$.

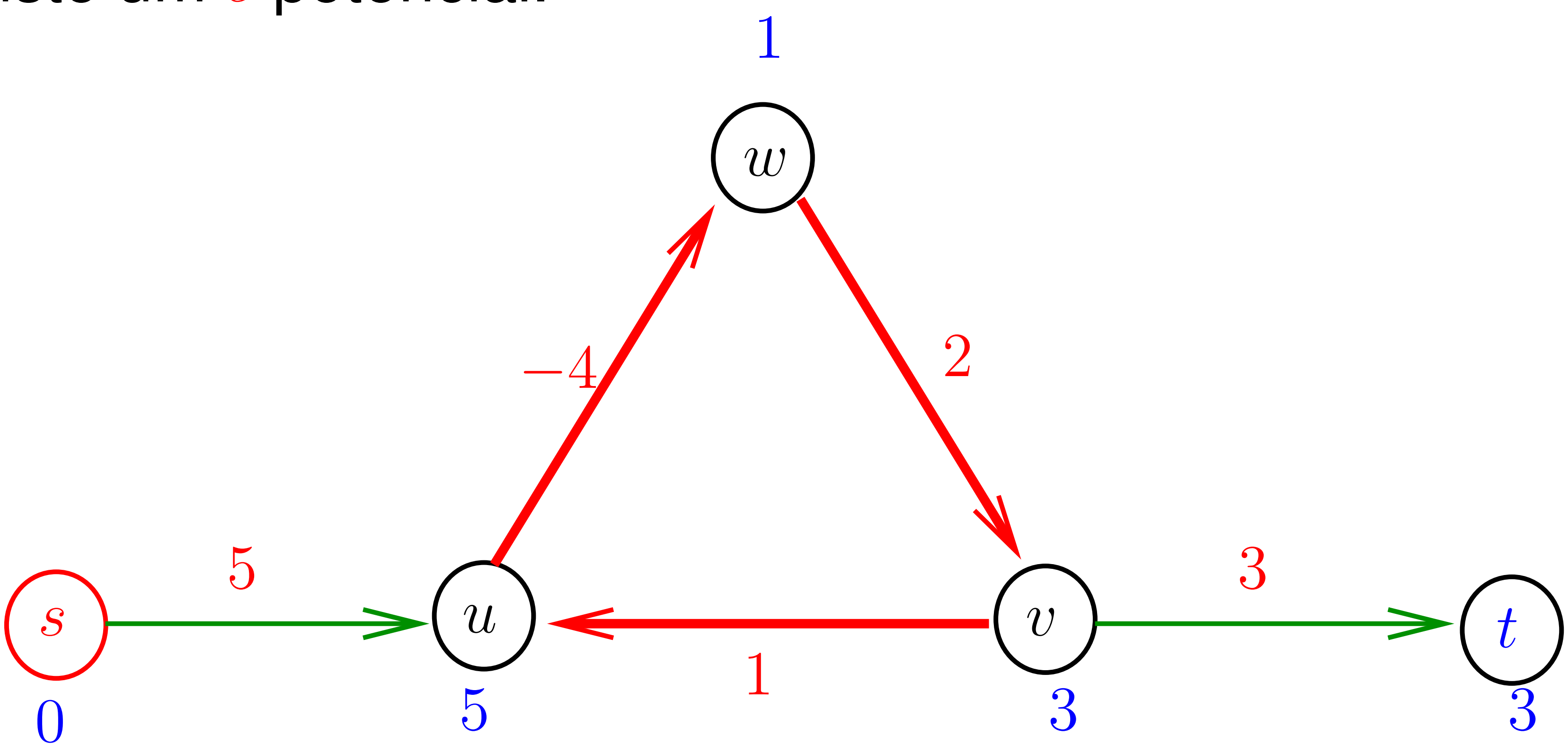
Conclusão

O problema do caminho mínimo é tão difícil quanto o problema do caminho hamiltoniano.

O problema do caminho de custo mínimo é NP-difícil.

Ciclos negativos

Se a rede possui um ciclo (de custo) negativo então não existe um c-potencial.



Vamos supor que...

Vamos supor que:

- a rede não tem ciclo negativo
- todo nó da rede está ao alcance de s
- $C := \max\{|c(ij)| : ij \in A\}$

Algoritmo genérico

Recebe uma rede (N, A, c) com função custo $c : A \to \mathbb{Z}$ sem ciclos negativos e um nó s e devolve uma função-predecessor π e um c-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)

para cada i em N **faça**

  $y(i) \leftarrow nC + 1$ $\triangleright$ $C := \max\{|c(ij)| : ij \in A\}$

  $\pi(i) \leftarrow \text{NIL}$

$y(s) \leftarrow 0$

enquanto $y(j) > y(i) + c(ij)$ **para** algum $ij \in A$ **faça**

  $y(j) \leftarrow y(i) + c(ij)$

  $\pi(j) \leftarrow i$

devolva π e y

Invariantes

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

(i1) para cada arco pq no grafo de predecessores tem-se $y(q) - y(p) \geq c(pq)$;

(i2) se pq é um arco tal que $y(q) - y(p) \geq c(pq)$, então **não** há um qp-caminho no grafo de predecessores.

(i3) $\pi(s) = \text{NIL}$ e $y(s) = 0$;

(i4) se $y(t) < \infty$, então há um st-caminho no grafo de predecessores.

(i5) Se $S := \{v : y(v) < \infty\}$, então, para cada v em S temos que $y(v) < |S|C'$, onde

$$C' := \max\{|c(pq)| : pq \in A, p \in S, q \in S\}.$$

Correção

Início da última iteração:

- y é um c-potencial
- Se $y(t) < \infty$ então, por (i4), segue que existe um st-caminho P no grafo de predecessores. Logo, (i1) e (i3) 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.

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 $c : A \to \mathbb{Z}$, sem ciclos negativos, 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

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

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

total $\quad 2\,\mathrm{O}(n) + \mathrm{O}(1) + \mathrm{O}(n^2m(C+1)) + \mathrm{O}(n^2(C+1))$

$$= \mathrm{O}(n^2m(C+1))$$

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.

Algoritmo de Ford-Bellman

FORD-BELLMAN (N, A, c, s)

para cada i em N **faça**

  $y(i) \leftarrow \infty$

  $\pi(i) \leftarrow \text{NIL}$

$y(s) \leftarrow 0$

repita $n-1$ vezes

  para cada arcor ij em A **faça**

    se $y(j) > y(i) + c(ij)$

      então $y(j) y(i) + c(ij)$

        $\pi(j) \leftarrow i$

devolva π e y

Consumo de tempo

O número de execuções do bloco de linhas 5–9 é

$$n - 1 < n.$$

linha	consumo de todas as execuções da linha
1-3	$\mathrm{O}(n)$
4	$\mathrm{O}(1)$
5	$n\mathrm{O}(1) = \mathrm{O}(n)$
6-9	$n\mathrm{O}(m) = \mathrm{O}(nm)$
10	$\mathrm{O}(n)$
total	$3\,\mathrm{O}(n) + \mathrm{O}(1) + \mathrm{O}(nm)$ $= \mathrm{O}(nm)$

Conclusão

> O consumo de tempo do algoritmo FORD-BELLMAN é $O(nm)$.

Este consumo de tempo é polinomial.

Invariantes

O algoritmo FORD-BELLMAN, além das invariantes (i1)-(i5), mantém ainda a seguinte invariante.

Chamemos de passo cada execução das linhas 6–9.

Após o k-ésimo passo vale que

(i7) se $y(t) < \infty$ e P é um st-passeio com $\leq k$ arcos então

$$y(t) \leq c(P).$$

Rascunho da demonstração de (i7)

Suponha que y é a função-potencial no início do passo $k+1$. Devido à invariante (i7), para todo nó i e todo si-caminho P_i com $\leq k$ arcos tem-se que

$$y(i) \leq c(P_i)$$

Seja y' é a função-potencial no fim do passo $k+1$.

Seja j um nó e $P_j = P_i \cdot \langle ij \rangle$ um sj-passeio com $\leq k+1$ arcos. Temos que

$$\begin{aligned} y'(j) &\leq y(i) + c(ij) \quad \text{(serviço do passo)} \\ &\leq c(P_i) + c(ij) \quad |P_i| \leq k \\ &= c(P_j). \end{aligned}$$

Portanto, (i7) vale com y' e $k+1$ nos papéis de y e k.

Implementação FIFO de Ford-Bellman

FIFO-FORD-BELLMAN (N, A, c, s)

para cada i em N **faça**
    $y(i) \leftarrow \infty$
    $\pi(i) \leftarrow \text{NIL}$
$y(s) \leftarrow 0$
$L \leftarrow s$ $\triangleright$ L funciona como uma fila
enquanto $L \neq \langle\rangle$ **faça**
    retire o primeiro elemento, digamos i, de L
    para cada ij em $A(i)$ **faça**
        se $y(j) > y(i) + c(ij)$
            então $y(j) \leftarrow y(i) + c(ij)$
                $\pi(j) \leftarrow i$
                se $j \notin L$
                    então acrescente j ao final de L
devolva π e y

Implementação FIFO de Ford-Bellman

FIFO-FORD-BELLMAN (N, A, c, s)

para cada i em N **faça**
    $y(i) \leftarrow \infty \quad \pi(i) \leftarrow \text{NIL}$
$y(s) \leftarrow 0 \quad Q \leftarrow \langle s \rangle$
enquanto $Q \neq \langle\rangle$ **faça**
    $L \leftarrow Q$
    $Q \leftarrow \emptyset$
    enquanto $L \neq \langle\rangle$ **faça**
        retire o primeiro elemento, digamos i, de L
        para cada ij em $A(i)$ **faça**
            se $y(j) > y(i) + c(ij)$ **então**
                $y(j) \leftarrow y(i) + c(ij)$
                $\pi(j) \leftarrow i$
                se $j \notin Q$ **então**
                    acrescente j ao final de Q
devolva π e y

Invariantes

O algoritmo FIFO-FORD-BELLMAN, além das invariantes (i1)-(i6), mantém ainda a seguinte invariante.

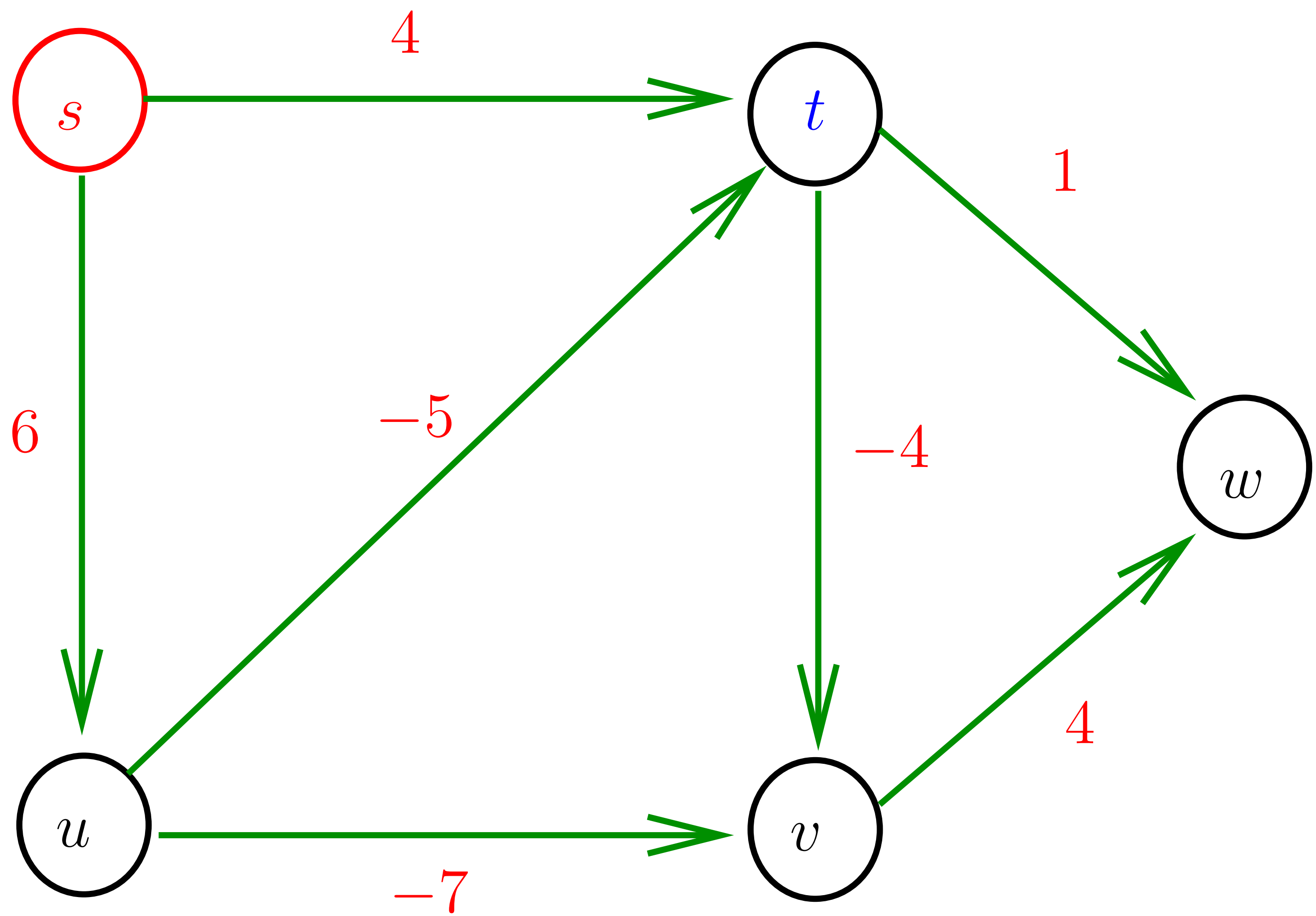
Chamemos de passo cada execução das linhas 7–13.

Após o k-ésimo passo vale que

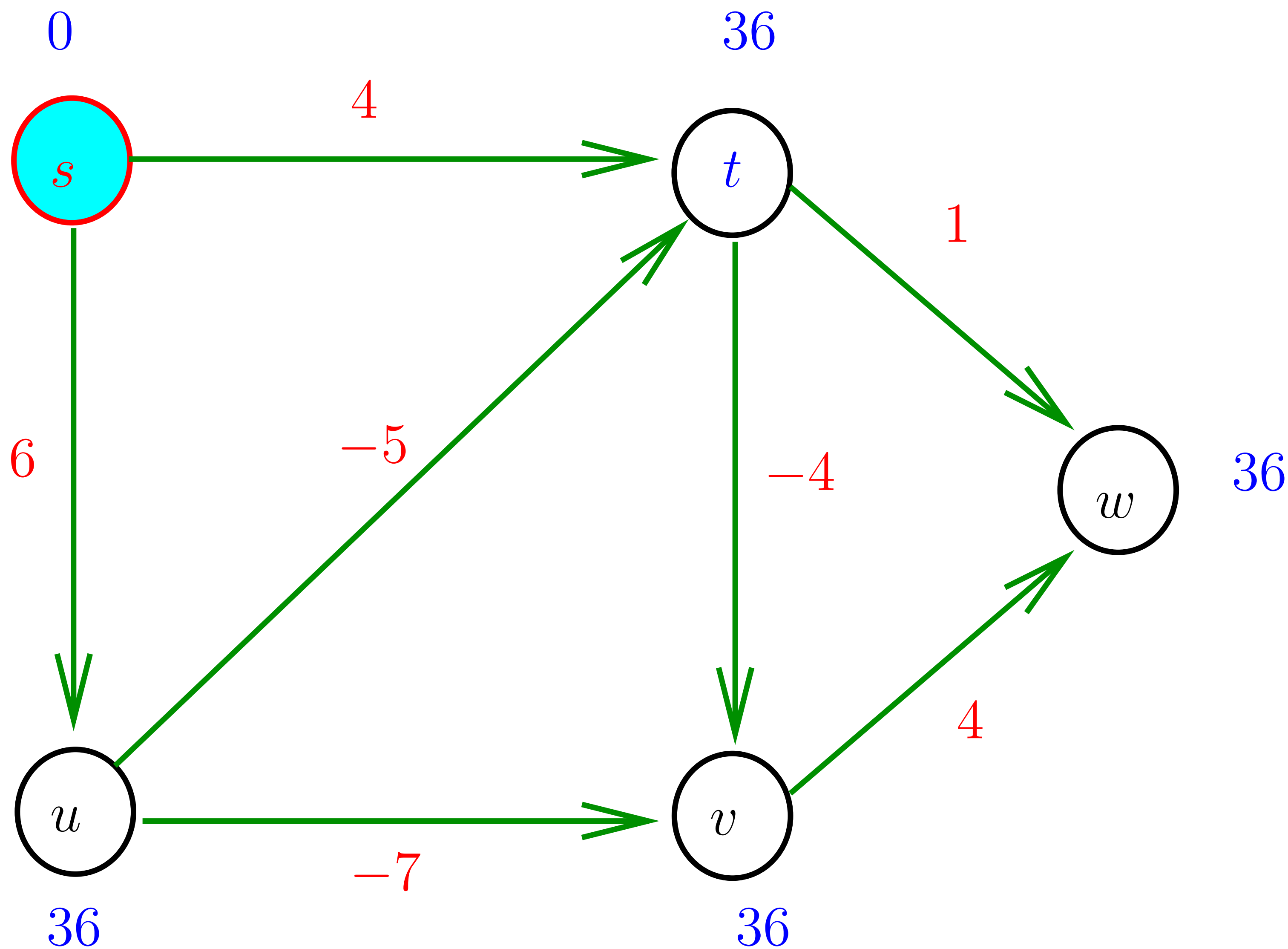
(i7) se $y(t) < \infty$ e P é um st-passeio com $\leq k$ arcos então

$$y(t) \leq c(P).$$

FIFO-Ford-Bellman

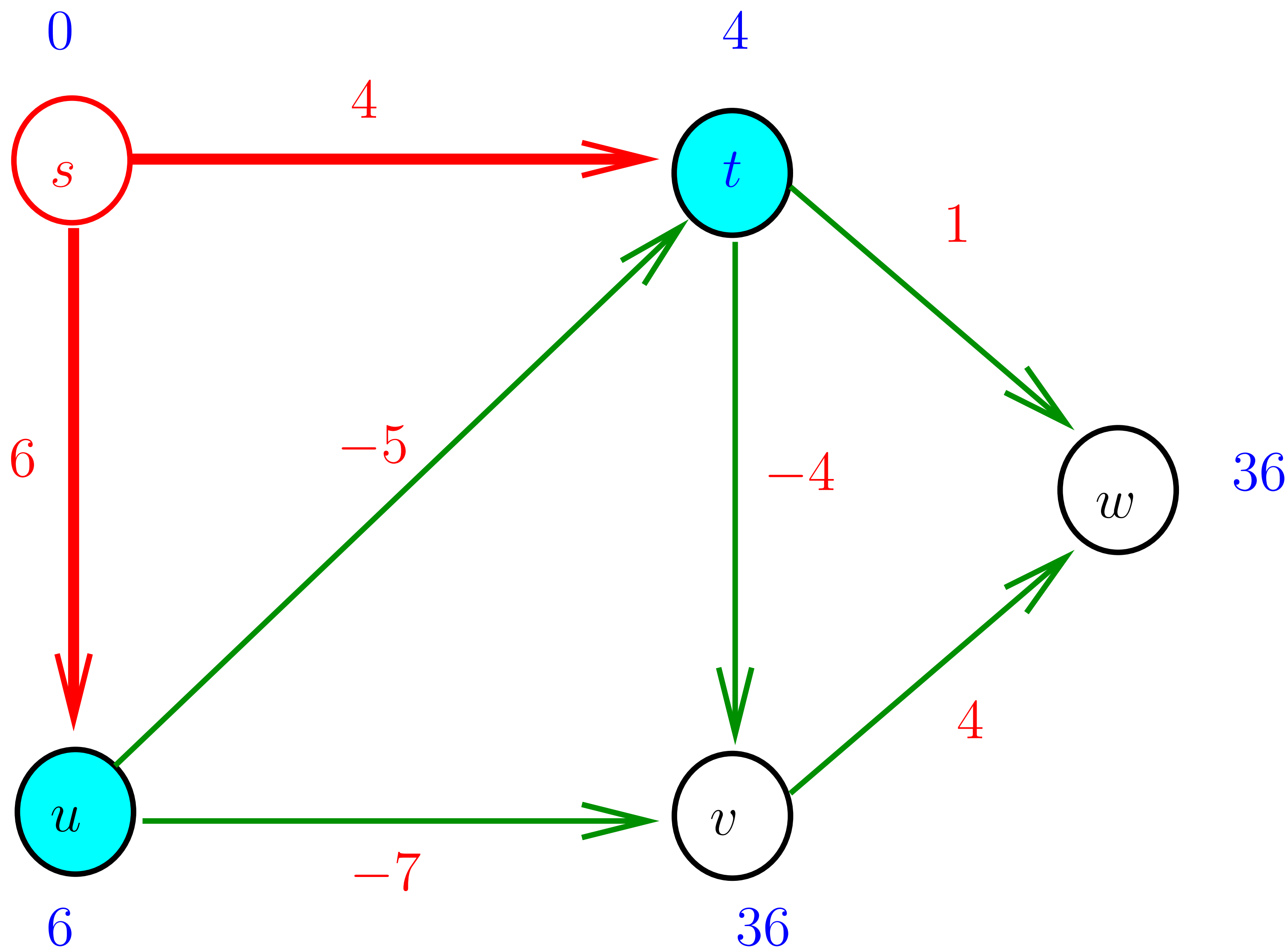


FIFO-Ford-Bellman



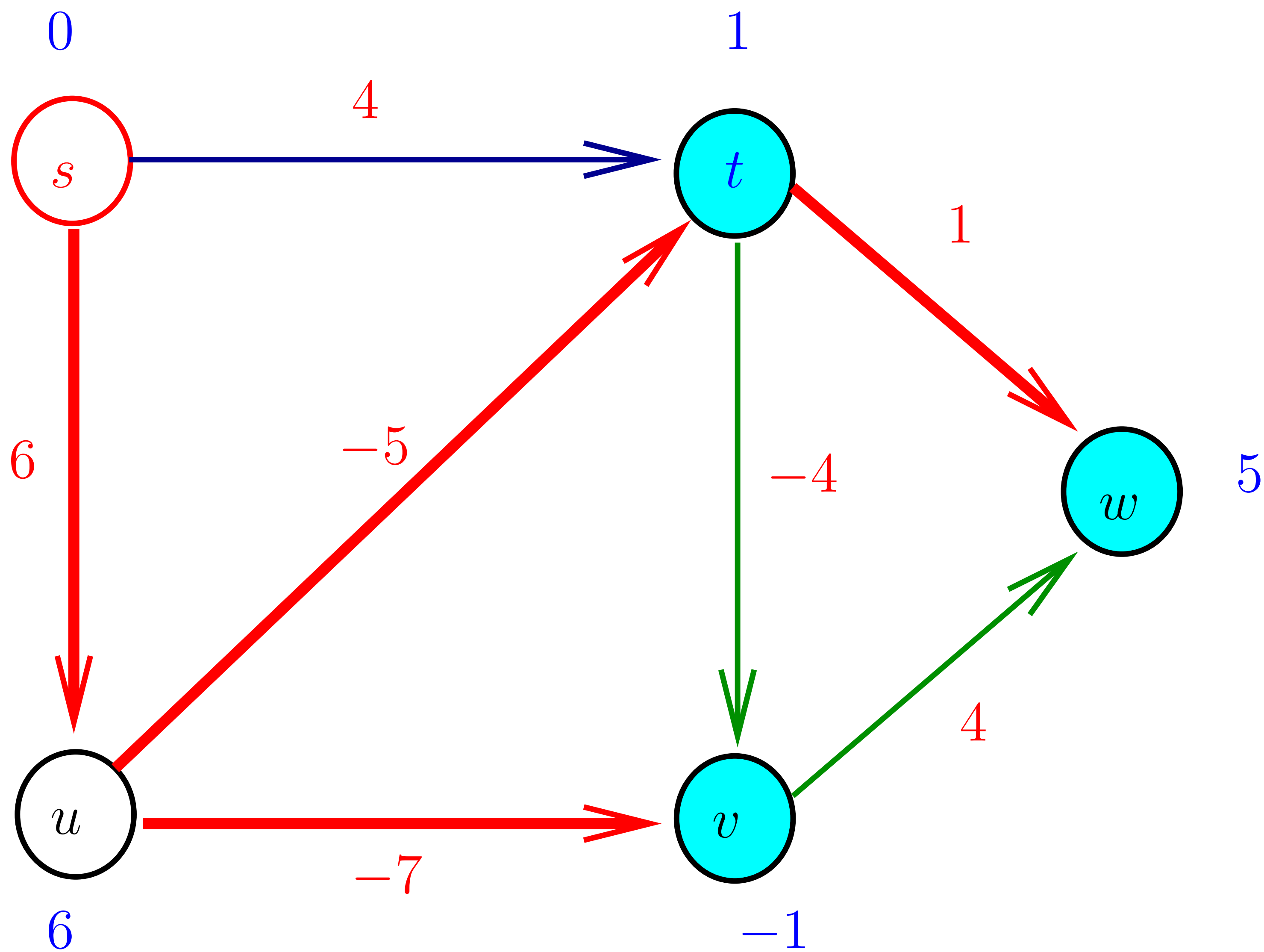
Início passo 0

FIFO-Ford-Bellman



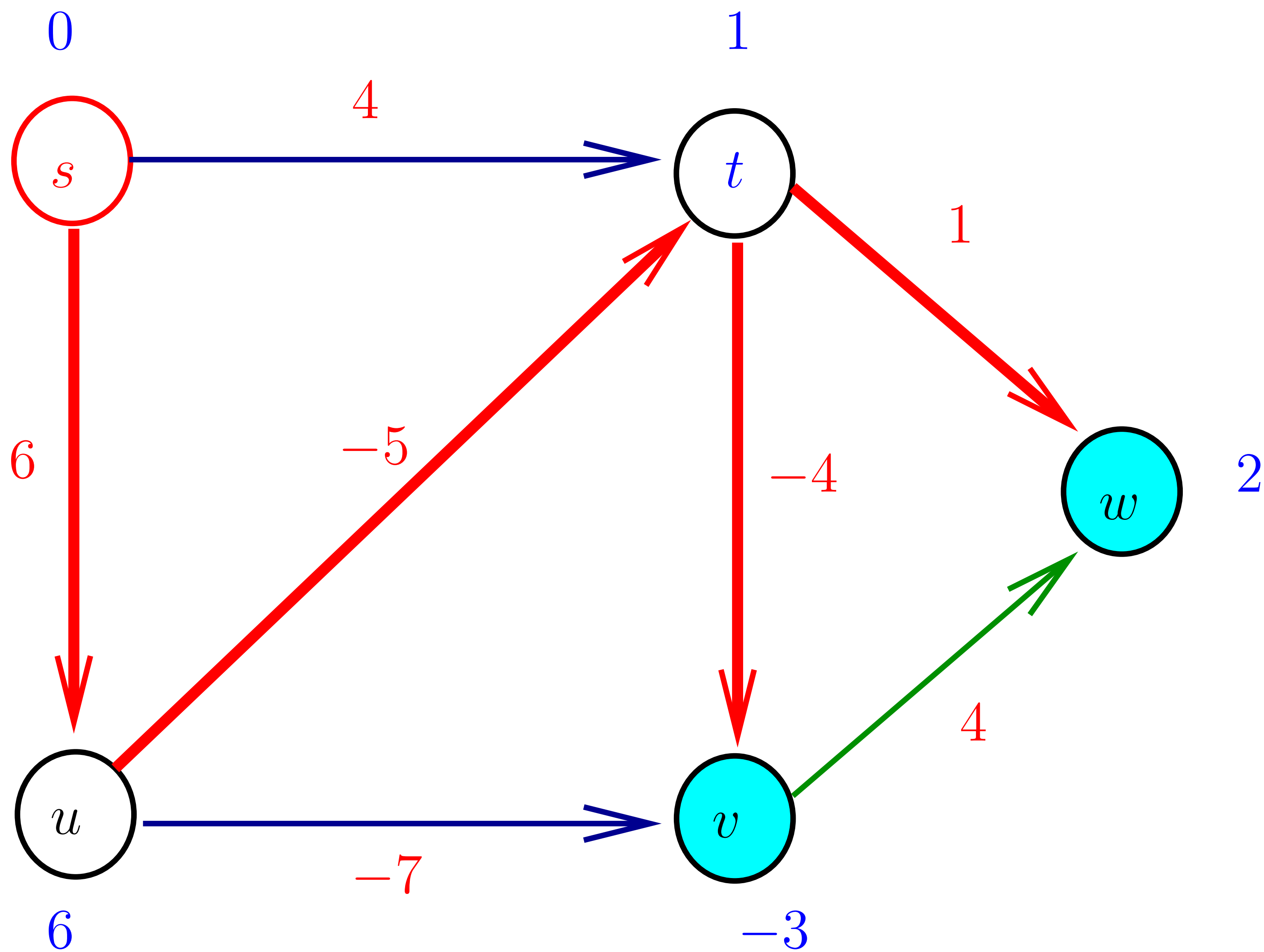
Fim passo 1

FIFO-Ford-Bellman



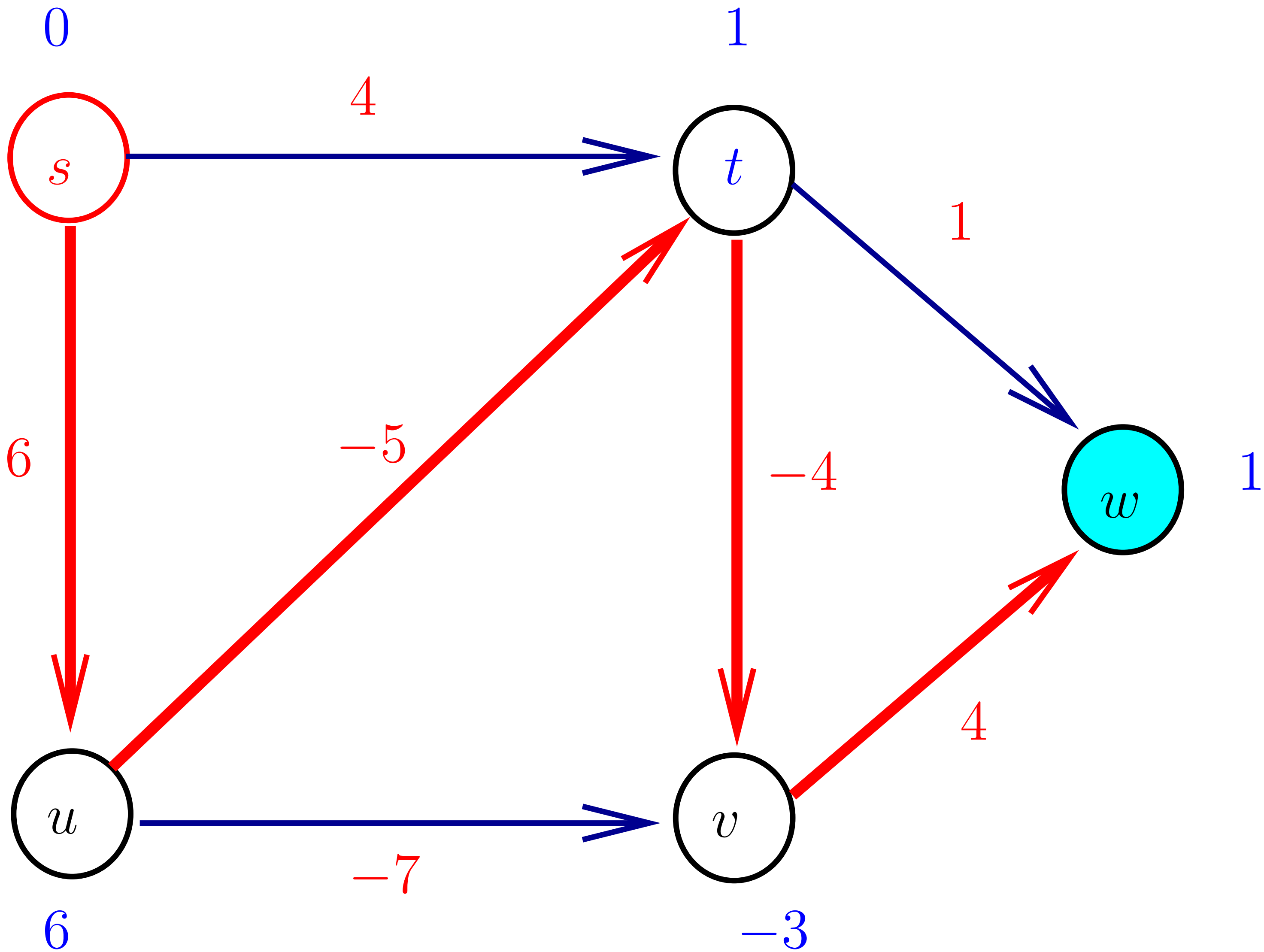
Fim passo 2

FIFO-Ford-Bellman



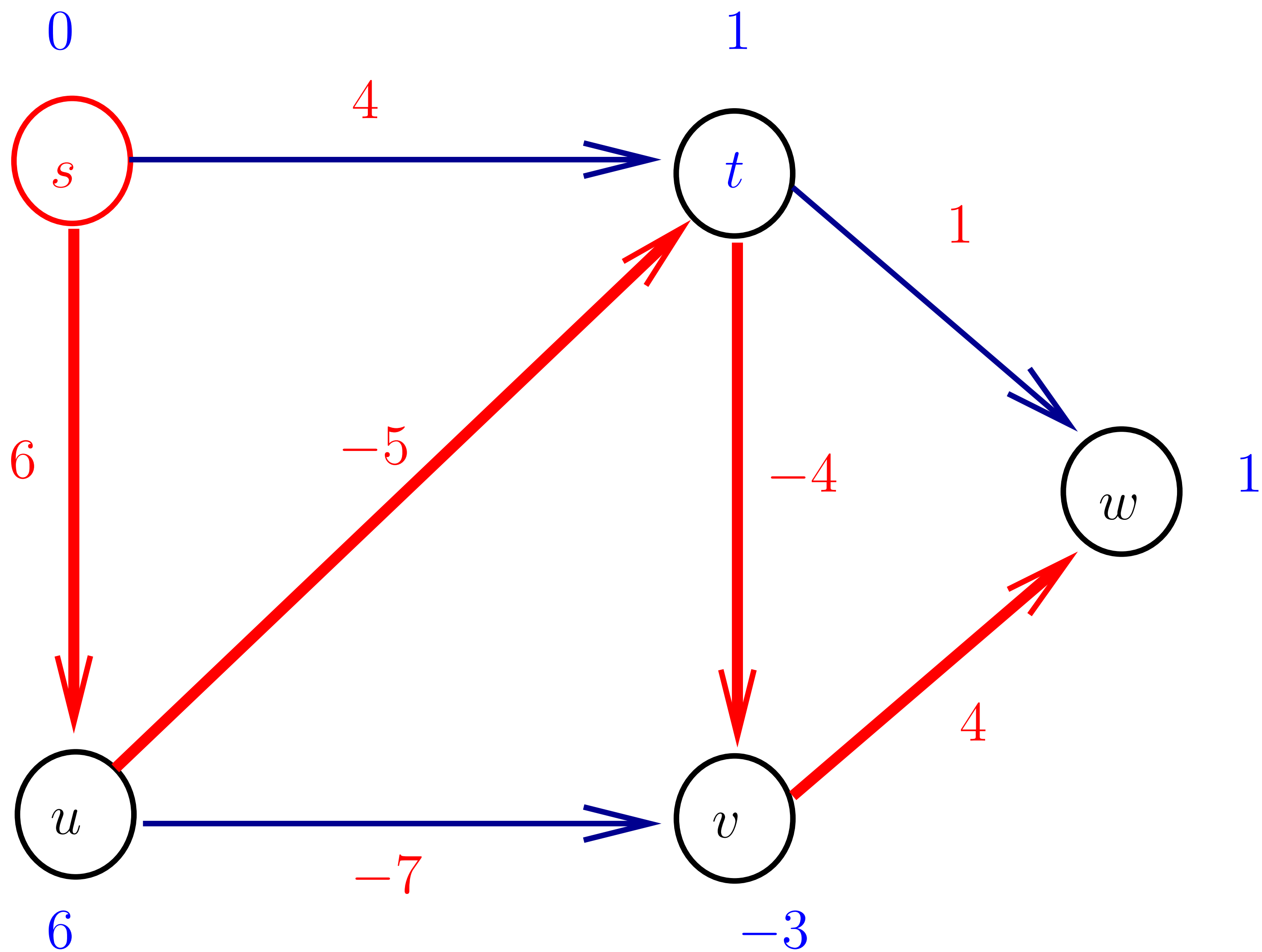
Fim passo 3

FIFO-Ford-Bellman



Fim passo 4

FIFO-Ford-Bellman



Fim passo 5