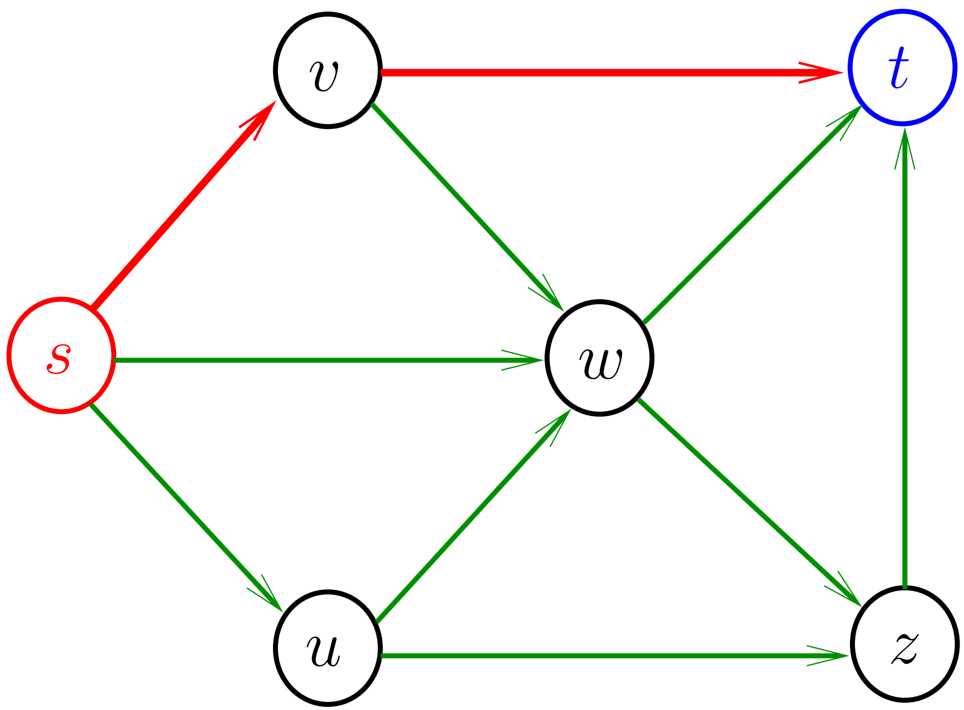
Melhores momentos

AULA 4

Problema

Problema do caminho mais curto: Dados nós s e t de um grafo (N, A), encontrar um caminho de s a t que tenha comprimento mínimo.



Algoritmo genérico

Recebe dois nós s e t de um grafo (N, A) e devolve um st-caminho P e um 1-potencial y tais que $|P| = y(t) - y(s)$.

CAMINHO-CURTO-GENÉRICO (N, A, s, t)

para cada i em N **faça**
    $y(i) \leftarrow n$ $\triangleright$ n faz o papel de ∞
    $\pi(i) \leftarrow \text{NIL}$
$y(s) \leftarrow 0$

enquanto $y(j) > y(i) + 1$ **para** algum $ij \in A$ **faça**
    $y(j) \leftarrow y(i) + 1$
    $\pi(j) \leftarrow i$

se $y(t) < n$
    então devolva o st-caminho no grafo (N, A_π) e y
    senão devolva “não há st-caminho” e y

Consumo de tempo (1)

O consumo de tempo do algoritmo
CAMINHO-CURTO-GENÉRICO é $O(n^2m)$.

Consumo de tempo (2)

O consumo de tempo do algoritmo
BUSCA-EM-LARGURA é $O(n + m)$.

Conclusão

Da propriedade dos 1-potenciais (lema da dualidade) e da correção do algoritmo CAMINHO-CURTO-GENÉRICO concluímos o seguinte:

(Teorema dualidade) Se s e t são nós de um grafo (N, A) e t está ao alcance de s então

$$\min\{|P| : P \text{ é um } st\text{-caminho}\}$$
$$= \max\{y(t) - y(s) : y \text{ é um } 1\text{-potencial}\}.$$

Outra relação min-max

Se s e t são nós de um grafo (N, A) e t está ao alcance de s então o menor comprimento de um st-caminho é igual ao número máximo de st-cortes disjuntos.

Rascunho de demonstração

É evidente que se

$$\nabla(S_0, T_0), \nabla(S_1, T_1), \ldots, \nabla(S_k, T_k)$$

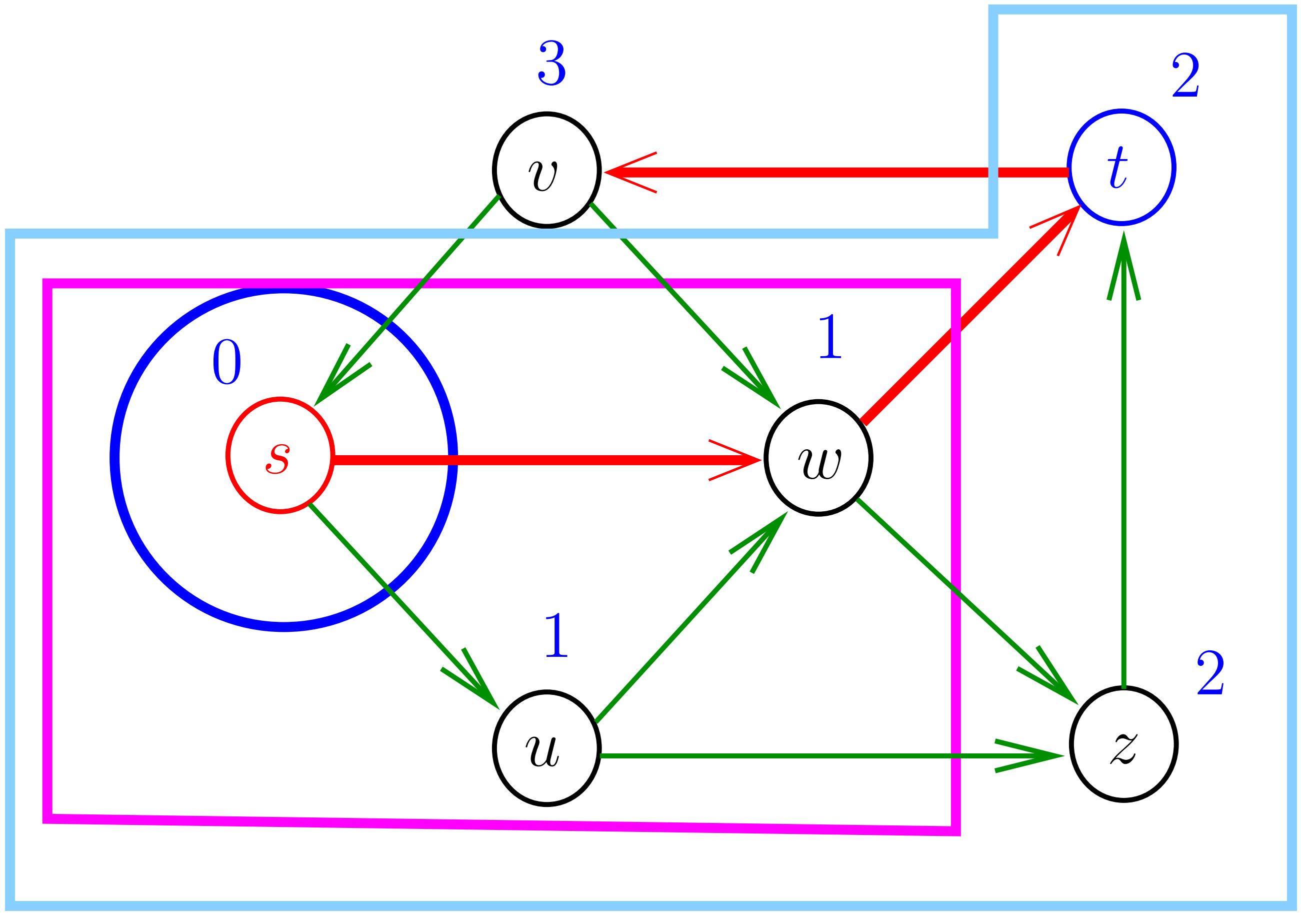
são $k+1$ st-cortes disjuntos então todo st-caminho tem comprimento $\geq k+1$.

Considere o 1-potencial y devolvido pelo algoritmo CAMINHO-CURTO-GENÉRICO e seja $k := y(t) - 1$.
Defina para $d = 0, \ldots, k$

$$S_d := \{u \in N : y(u) \leq d\} \quad \text{e} \quad T_d := N - S_d.$$

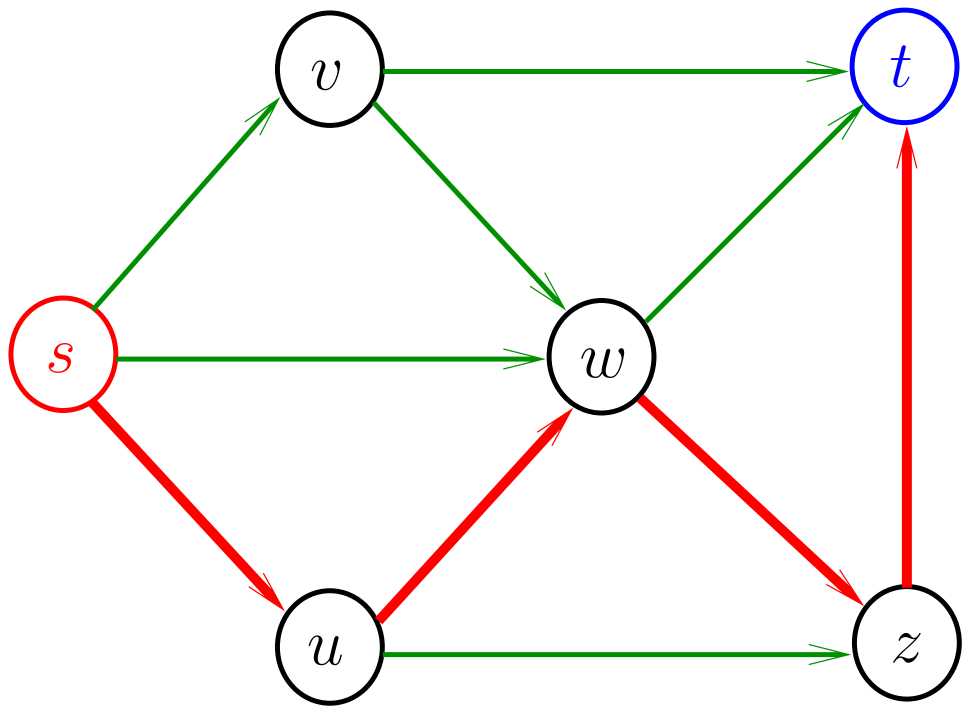
Verifique que $\nabla(S_0, T_0), \ldots, \nabla(S_k, T_k)$ são $k+1 = y(t)$ st-cortes disjuntos.

Exemplo



Grafo

Eis um grafo e um caminho de s a t.



Matrizes de incidências

Eis a matrizes de incidência M do grafo.

Onde está o caminho $\langle s, u, w, z, t \rangle$?

	sv	su	sw	uw	uz	vw	vt	wt	wz	zt
s	-1	-1	-1							
u		$+1$		-1	-1					
v	$+1$					-1	-1			
w			$+1$	$+1$		$+1$		-1	-1	
z					$+1$				$+1$	-1
t							$+1$	$+1$		$+1$

A

N

Matrizes de incidências

Rerranjei as linhas e colunas de M.

Que “cara” tem a “submatriz do caminho” $\langle s, u, w, z, t\rangle$?

	su	uw	wz	zt	sv	sw	uz	vw	vt	wt	A
s	-1				-1	-1					
u	$+1$	-1					-1				
w		$+1$	-1			$+1$		$+1$		-1	
z			$+1$	-1			$+1$				
t				$+1$					$+1$	$+1$	
v					$+1$			-1	-1		
N											

Caminhos e submatrizes

Submatrizes associadas a caminhos têm estrutura semelhante da seguinte.

$$\begin{array}{c|cccc|c} & su & uw & wz & zt & A' \\ \hline s & -1 & & & & \\ u & +1 & -1 & & & \\ w & & +1 & -1 & & \\ z & & & +1 & -1 & \\ t & & & & +1 & \\ \hline N' & & & & & \end{array}$$

Note que as colunas são linearmente independentes.

Conclusão

Da correção do algoritmo BUSCA-GENÉRICO temos o seguinte.

Para quaisquer matriz de incidências M e vetor de incidência b, vale uma e apenas umas das seguintes afirmações:

- existe um vetor x com elementos em $\{0, 1\}$ tal que $Mx = b$
- existe um vetor y tal que $yM \leq 0$ e $yb > 0$.

Nos algoritmos BUSCA-GENÉRICO e BUSCA o vetor x é representado pela função-predecessor π.

Soma de subconjuntos (*subset-sum*)

Problema: Dados inteiros não-negativos $a_1, a_2, \ldots, a_n, b$, encontrar uma solução de

$$a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = b$$

tal que $x_i = 0$ ou $x_i = 1$ para todo i.

Exemplo:

$$100x_1 + 30x_2 + 90x_3 + 35x_4 + 40x_5 + 30x_6 + 10x_7 = 160$$

tem uma solução 0-1?

Sim! $x_1 = x_2 = x_6 = 1$ e $x_3 = x_4 = x_5 = x_7 = 0$ é solução.

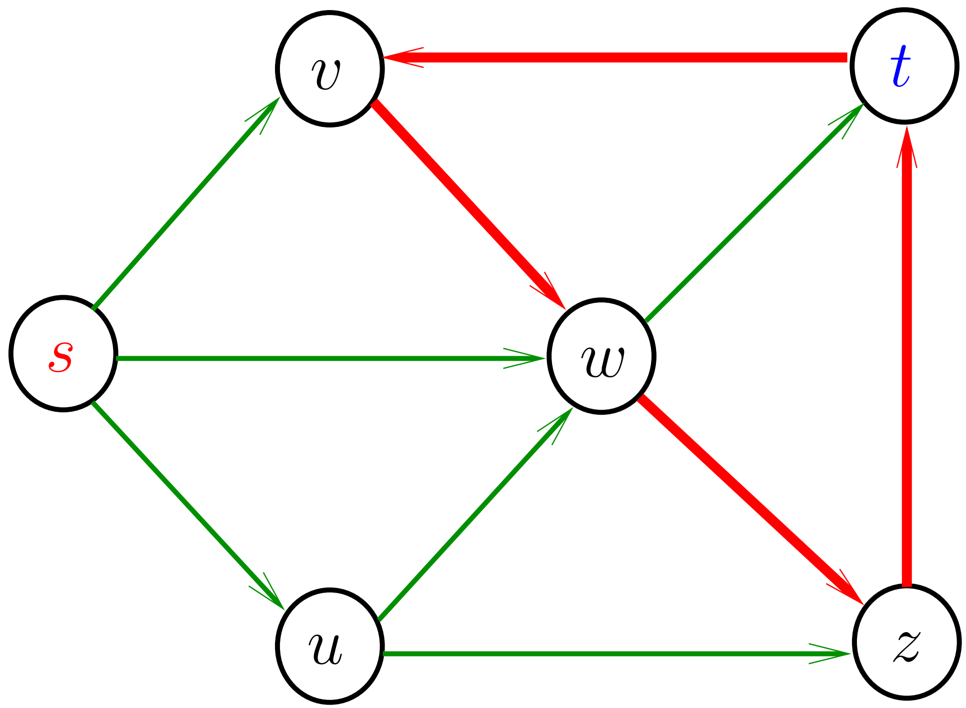
Este problema é NP-difícil.
Existe algoritmo "pseudo-polinomial" (prog. dinâmica).

AULA 5

Ciclos e matrizes de incidências

Grafo

Eis um grafo e um ciclo.



Matrizes de incidências

Eis a matrize de incidências M do grafo.

Onde está o caminho $\langle v, w, z, t, v \rangle$?

	sv	su	sw	uw	uz	vw	tv	wt	wz	zt	A
s	-1	-1	-1								
u		$+1$		-1	-1						
v	$+1$					-1	$+1$				
w			$+1$	$+1$		$+1$		-1	-1		
z					$+1$				$+1$	-1	
t							-1	$+1$		$+1$	
N											

Matrizes de incidências

Rerranjei as linhas e colunas de M.

Que “cara” tem a “submatriz do ciclo” $\langle v, w, z, t, v\rangle$?

	vw	wz	zt	tv	sv	su	sw	uw	uz	wt	A
v	-1			$+1$	$+1$						
w	$+1$	-1					$+1$	$+1$		-1	
z		$+1$	-1						$+1$		
t			$+1$	-1						$+1$	
s					-1	-1	-1				
u						$+1$			-1	-1	

N

Ciclos e submatrizes

Submatrizes associadas a ciclos têm estrutura semelhante a da seguinte.

	vw	wz	zt	tv	A'
v	-1			$+1$	
w	$+1$	-1			
z		$+1$	-1		
t			$+1$	-1	
N'					

Note que as colunas são linearmente dependentes.

Conclusão

O problema do ciclo é equivalente ao abaixo.

Problema. Dada a matriz de incidências M de um grafo (N, A), encontrar um vetor não-nulo x indexado por A tal que

- $Mx = 0$ e
- $x[ij] \in \{0, 1\}$ para todo $ij \in A$.

Conclusão

Da correção do algoritmo DAG-GENÉRICO temos o seguinte.

Para qualquer matriz de incidência M, vale uma e apenas umas das seguintes afirmações:

- existe um vetor não-nulo x com elementos em $\{0, 1\}$ tal que $Mx = 0$
- existe um vetor y tal que $yM \leq -1$.

Nos algoritmos DAG-GENÉRICO e DAG o vetor x é representado pela função-predecessor π.

Ciclos não orientados e submatrizes

Submatrizes associadas a ciclos não-orientados têm estrutura semelhante a da seguinte.

Note que as colunas são linearmente dependentes.

$$
\begin{array}{c|c|c|c|c|c|c|l}
 & su & uz & wz & wt & tv & sv & A' \\
\hline
s & -1 & & & & & -1 & \\
u & +1 & -1 & & & & & \\
z & & +1 & +1 & & & & \\
w & & & -1 & -1 & & & \\
t & & & & +1 & -1 & & \\
v & & & & & +1 & +1 & \\
\hline
N' & & & & & & &
\end{array}
$$

Conclusão

Suponha que M é a matriz de incidências de um grafo (N, A) e que A' é uma parte de A. As colunas da submatriz $M[N, A']$ são linearmente independentes se e somente se o grafo (N, A') não possui ciclos não-orientados.

Bases

Uma base de uma matriz é qualquer submatriz formada por um conjunto maximal de colunas linearmente independentes.

Bases têm uma papel importante no método Simplex de programação linear.

Suponha que M é a matriz de incidências de um grafo (N, A) e que A' é uma parte de A. A submatriz $M[N, A']$ é uma base de M se e somente se e somente se o grafo (N, A') é uma "floresta geradora".

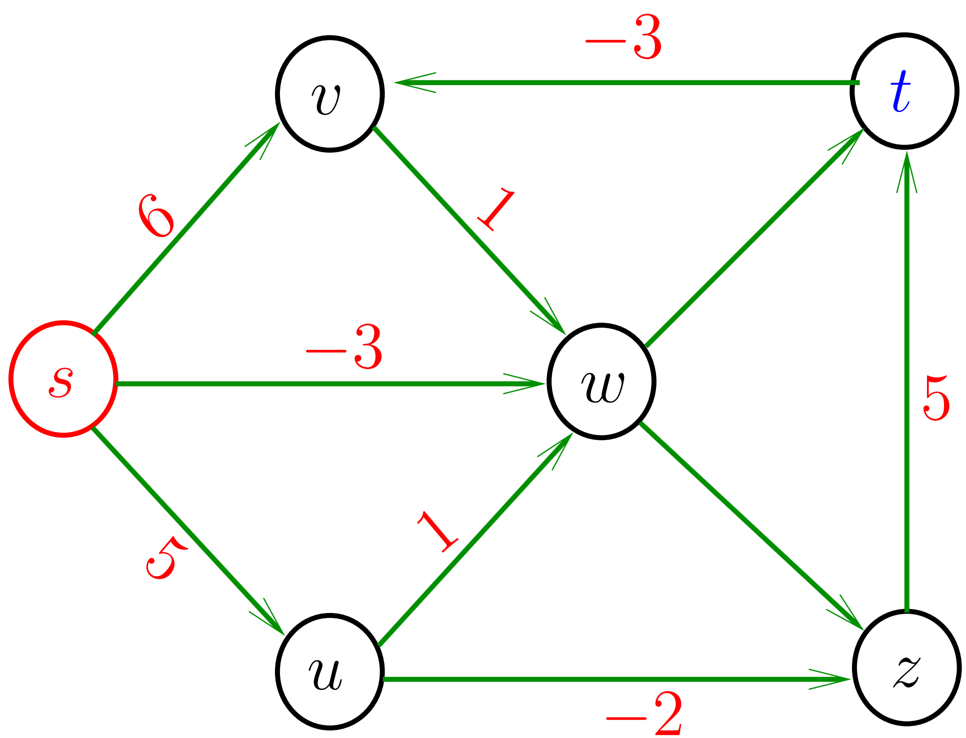
Floresta geradora $=$ subgrafo maximal sem ciclos não-orientados.

Caminhos de custo mínimo

PF 8.1, 8.2, 8.4

Custos

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



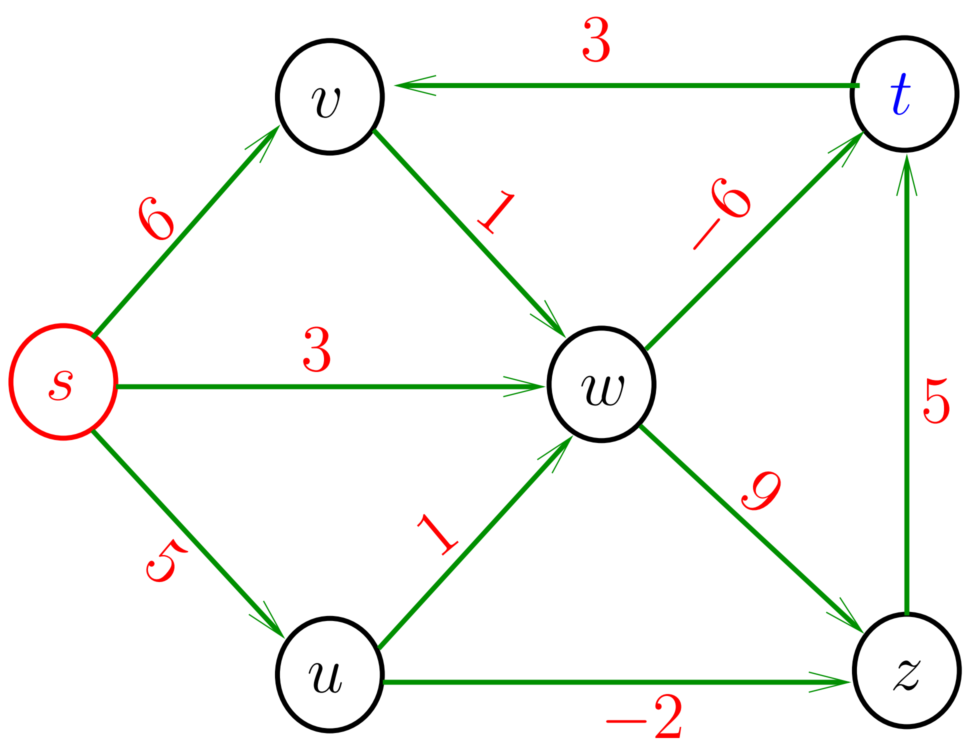
Custo de um passeio

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.

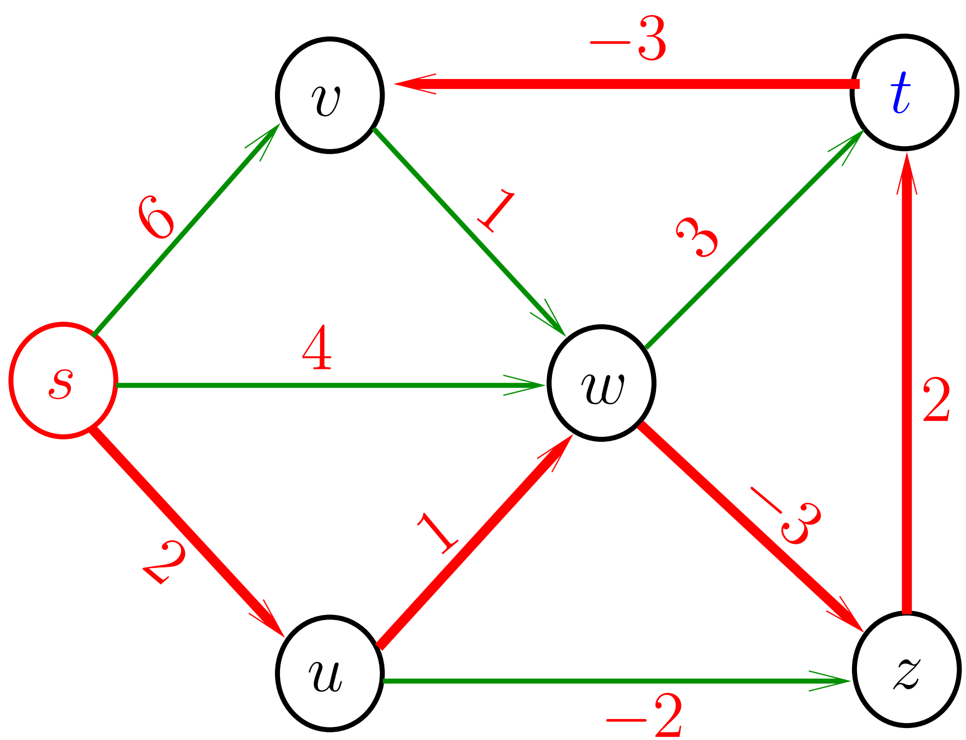
O custo do passeio $\langle s, v, w, t, v, w, t, v, w, z \rangle$ é 12.



Caminho mínimo

Um caminho P tem custo mínimo se $c(P) \leq c(P')$ para todo caminho P' com a mesma origem e término de P.

O passeio $\langle s, u, w, z, t, v \rangle$ é mínimo. Tem custo -1.



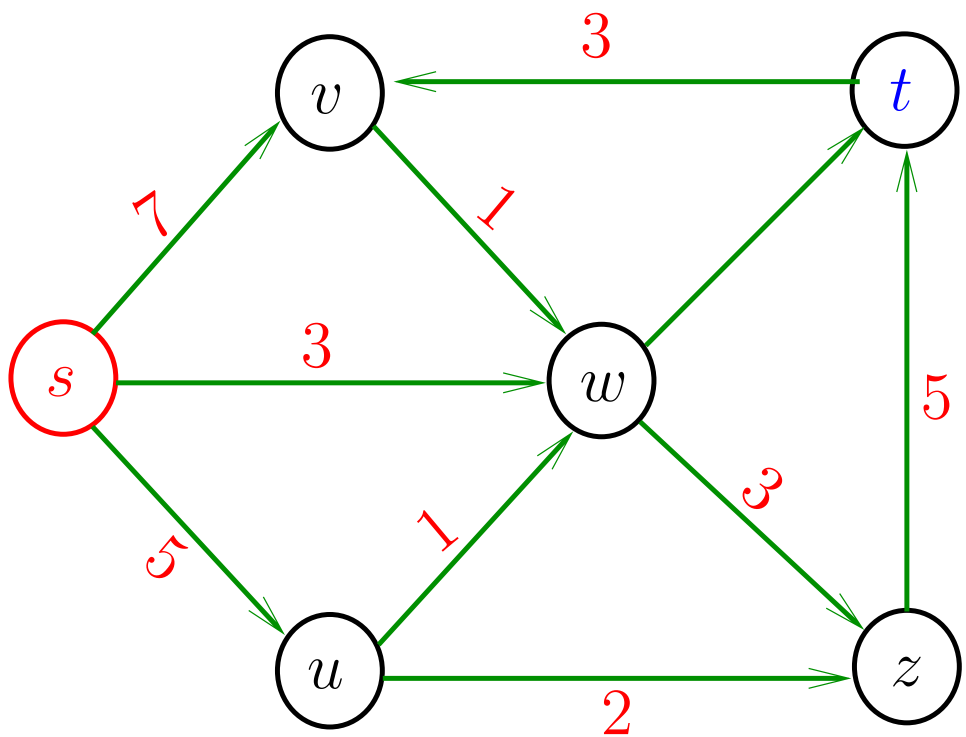
Problema

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.

Problema

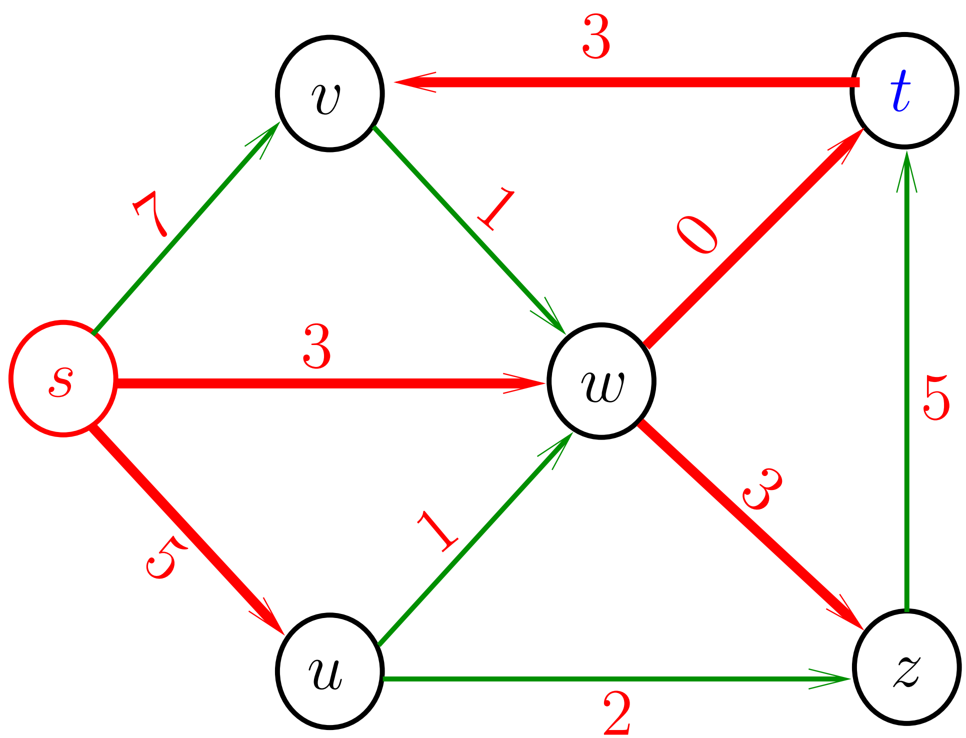
Problema do caminho mínimo sob custo não-negativo: Dada uma rede (N, A, c) com função-custo $c : A \to \mathbb{Z}_{\geq}$ e um nó s, encontrar, para cada nó t, um caminho de custo mínimo de s a t.

Entra:



Problema

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



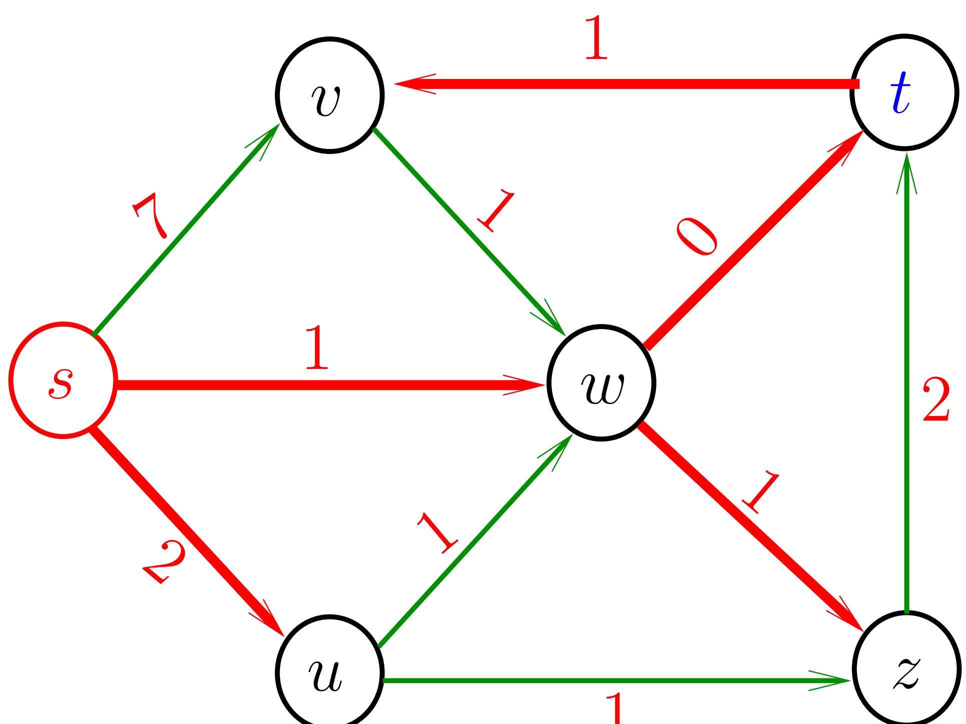
Condição de inexistência

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?

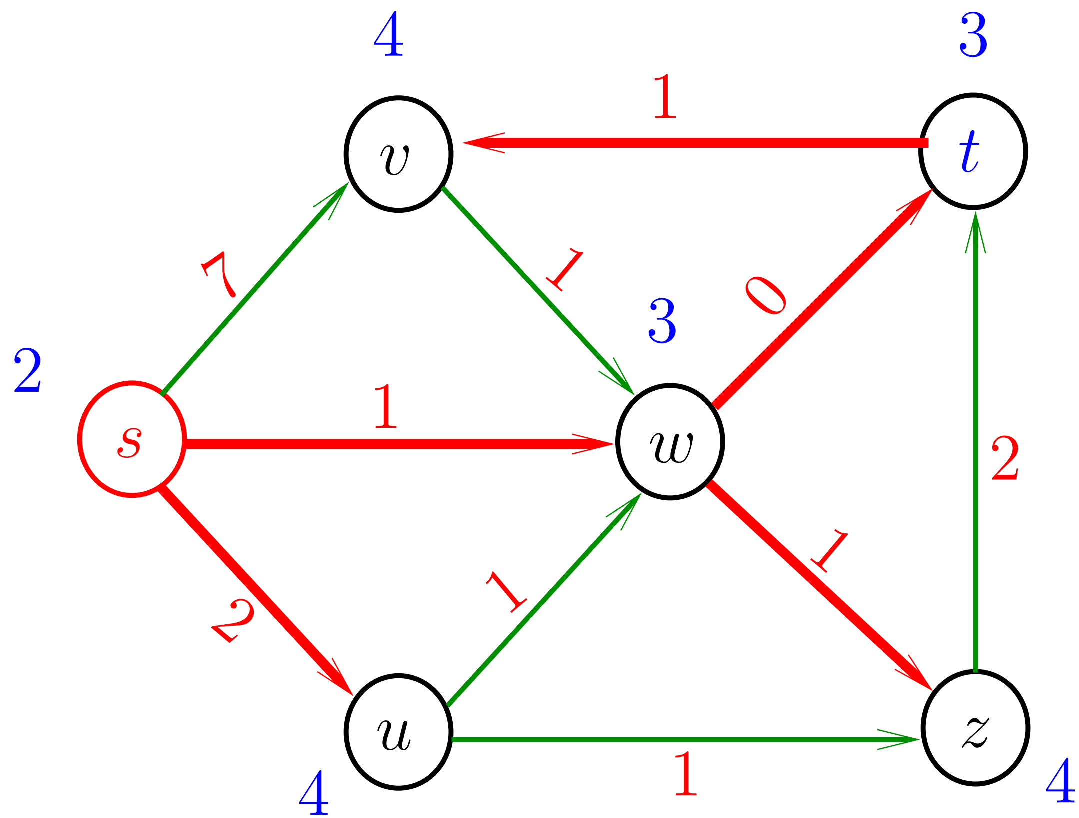
Entra:



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

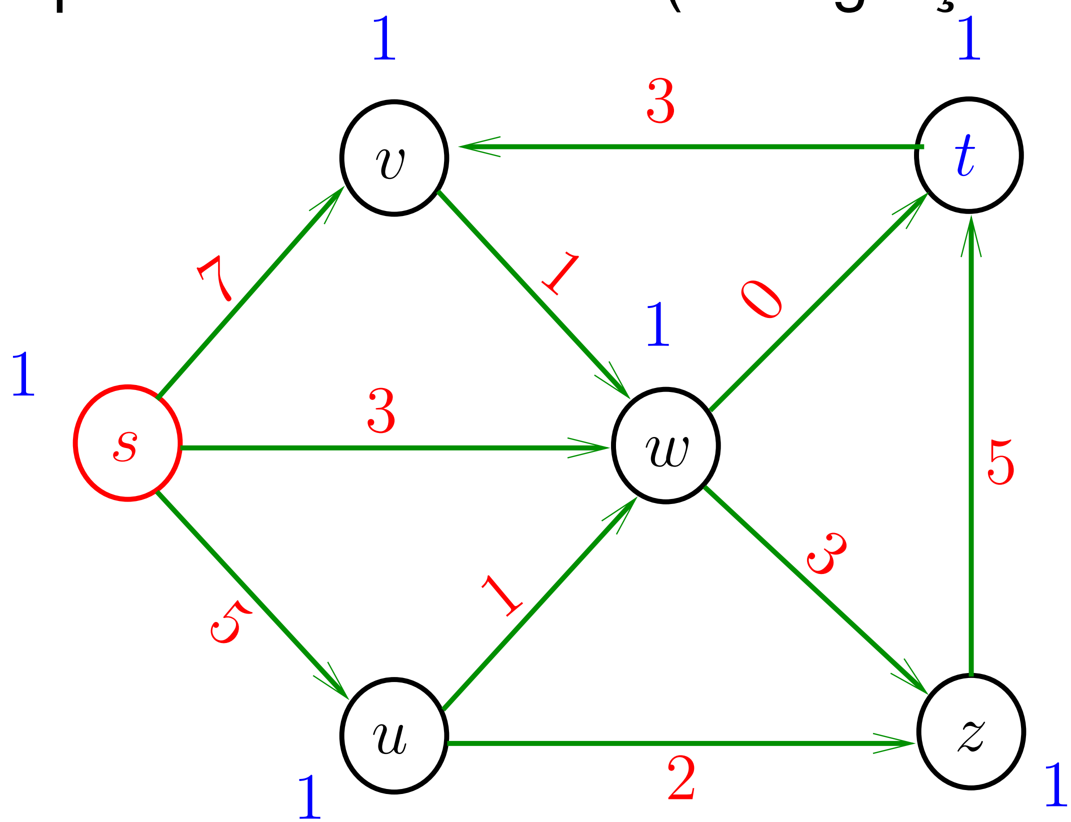
$$y(j) - y(i) \leq c(ij) \quad \text{para todo arco } ij.$$

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. . .)

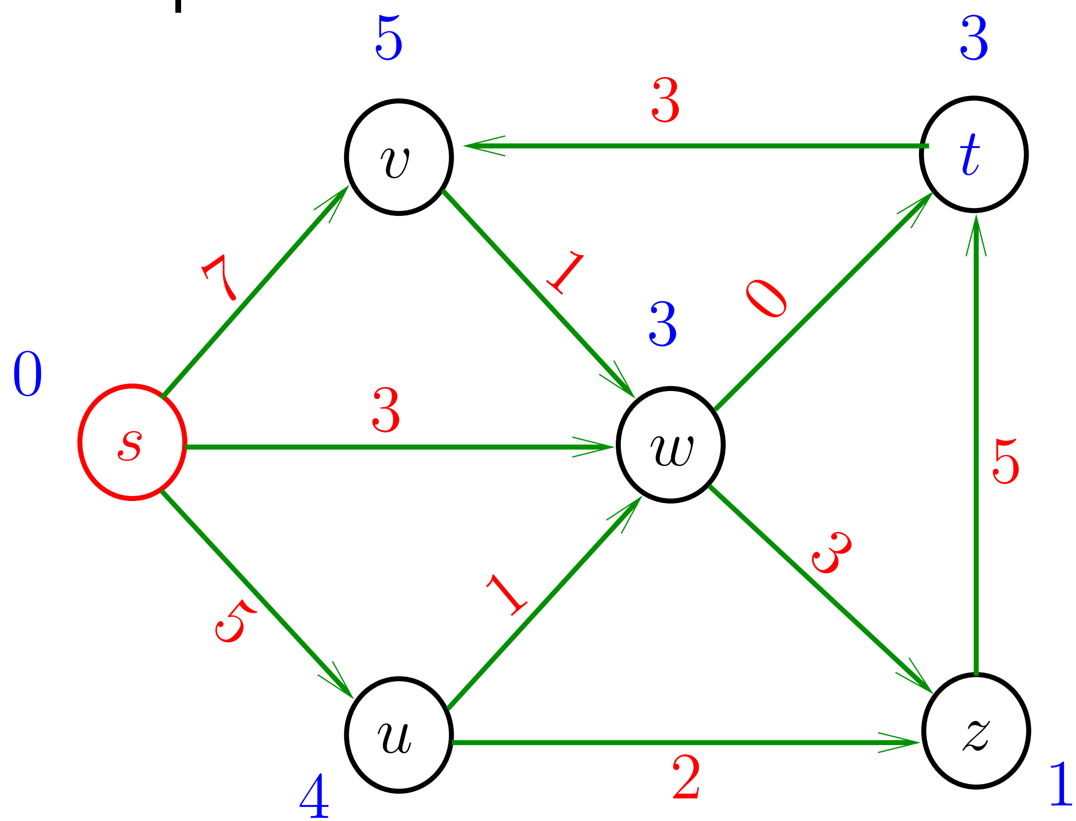


c-potencial

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

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

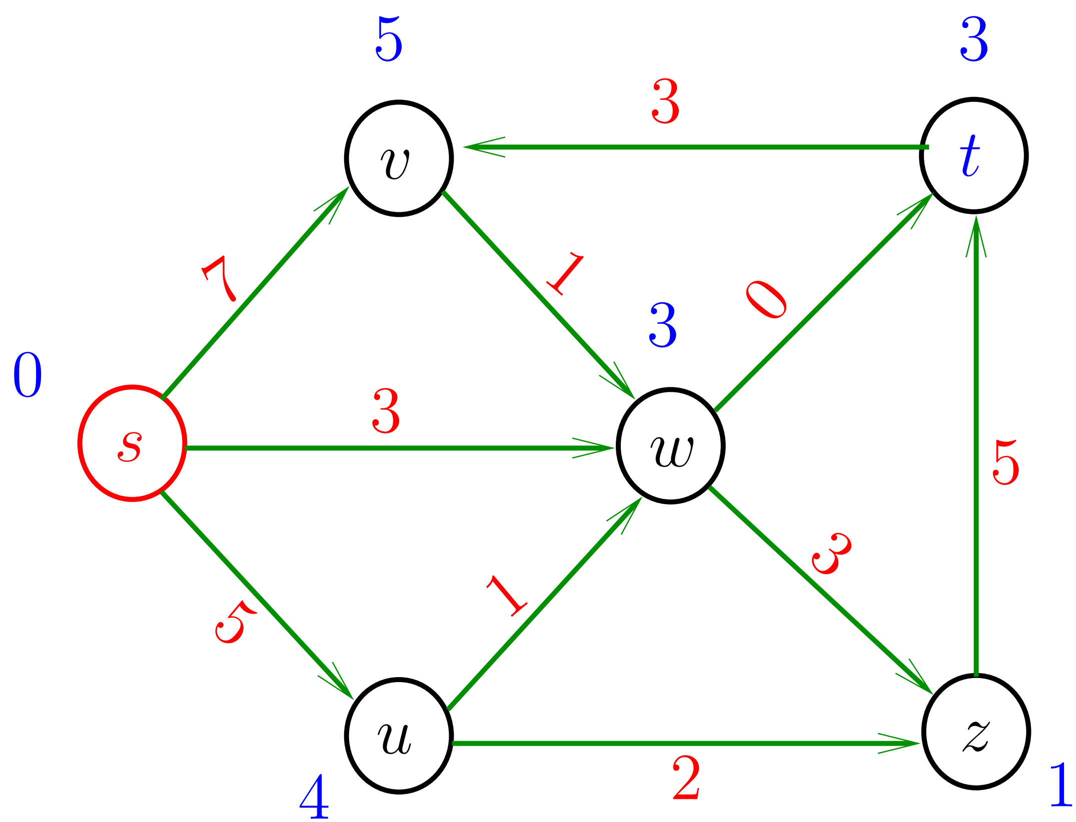
Exemplo 2: um c-potencial menos "bobo"



Propriedade de c-Potenciais

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

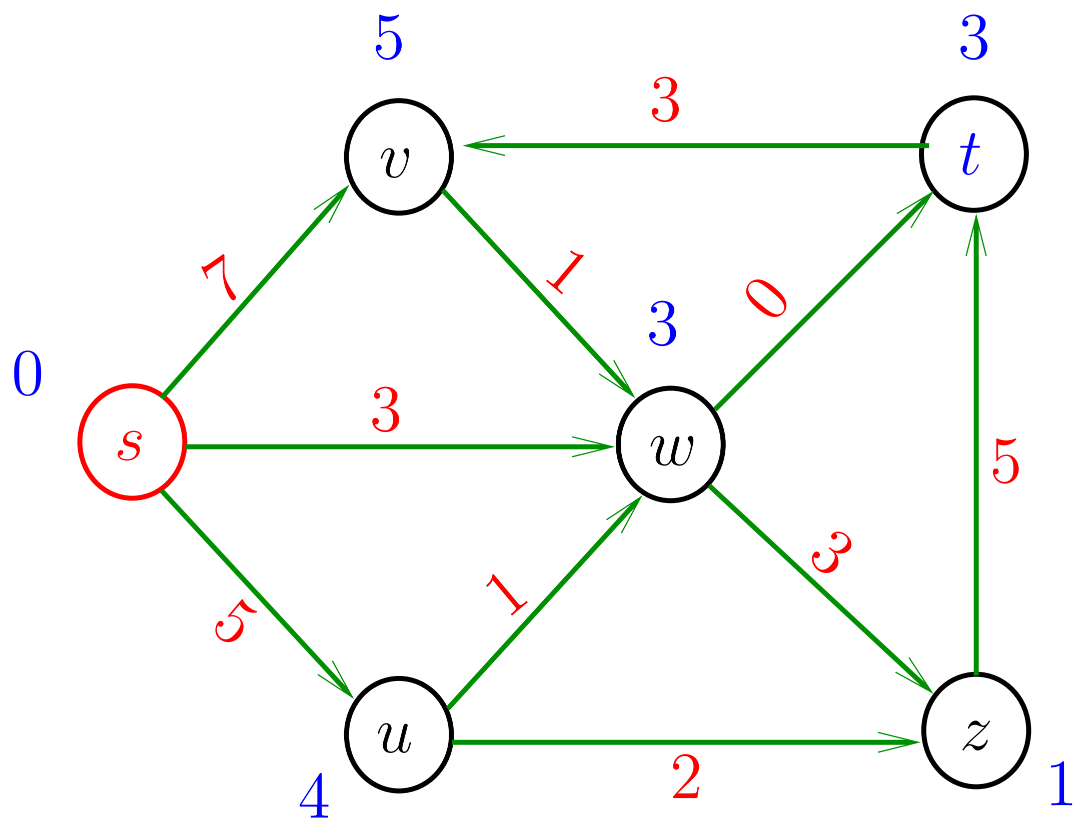
$$c(P) \geq y(t) - y(s).$$



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

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

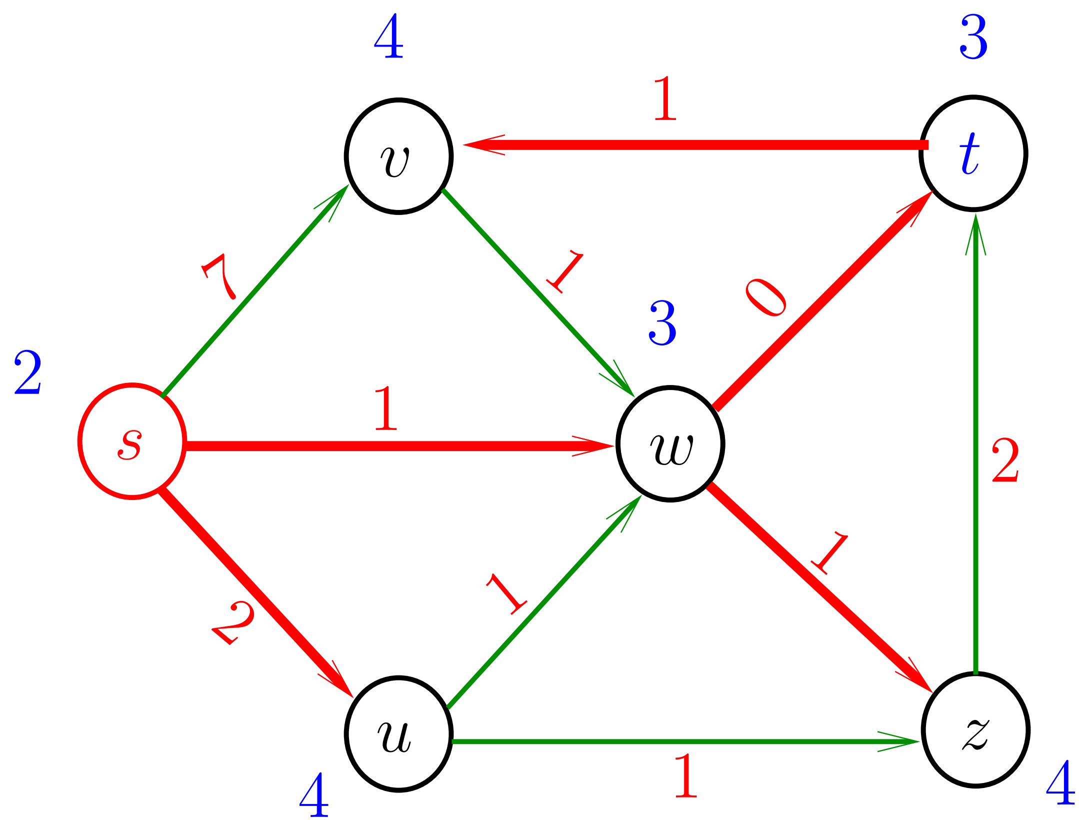


Conseqüência

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

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

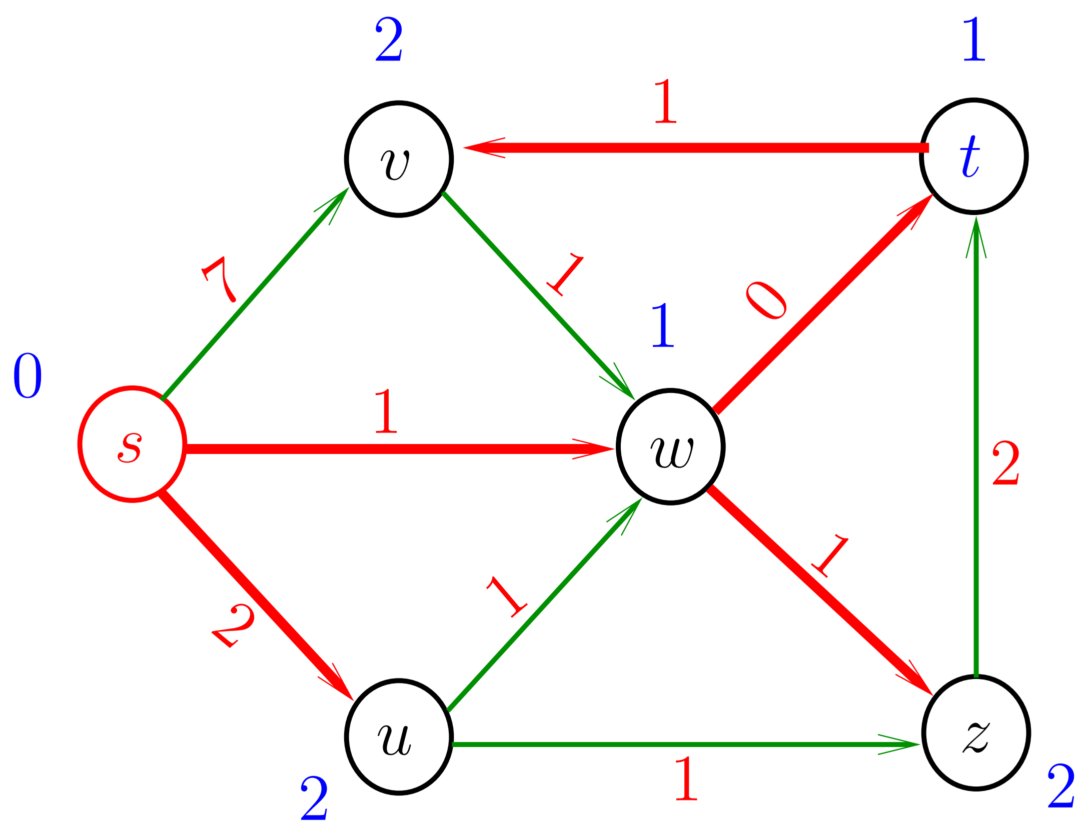
então P é uma caminho mínimo e y é um c-potencial tal que o valor de $y(t) - y(s)$ é máximo.



Potenciais ótimos

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

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



Algoritmo genérico

Recebe uma rede (N, A, c) com função custo $c : A \rightarrow \mathbb{Z}_{\geq}$ 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) $\quad \triangleright\ c \geq 0$

para cada i em N **faça**
 $\quad y(i) \leftarrow nC + 1 \quad \triangleright\ C := \max\{c(ij) : ij \in A\}$
 $\quad \pi(i) \leftarrow \mathrm{NIL}$
$y(s) \leftarrow 0$

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

devolva π e y

Invariantes

Na linha 4, 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) $\pi(s) = \mathrm{NIL}$ e $y(s) = 0$;

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

(i3) para cada nó v, se $\pi(v) \neq \mathrm{NIL}$ 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 $\quad 2\,\mathrm{O}(n) + \mathrm{O}(1) + \mathrm{O}(n^2m(C+1)) + \mathrm{O}(n^2(C+1))$

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

Conclusão

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

Este consumo de tempo não é polinomial.

Implementação do algoritmo Ford

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 ∞

  $\pi(i) \leftarrow$ NIL

$y(s) \leftarrow 0$

$Q \leftarrow N$ $\quad \triangleright\ Q$ 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**

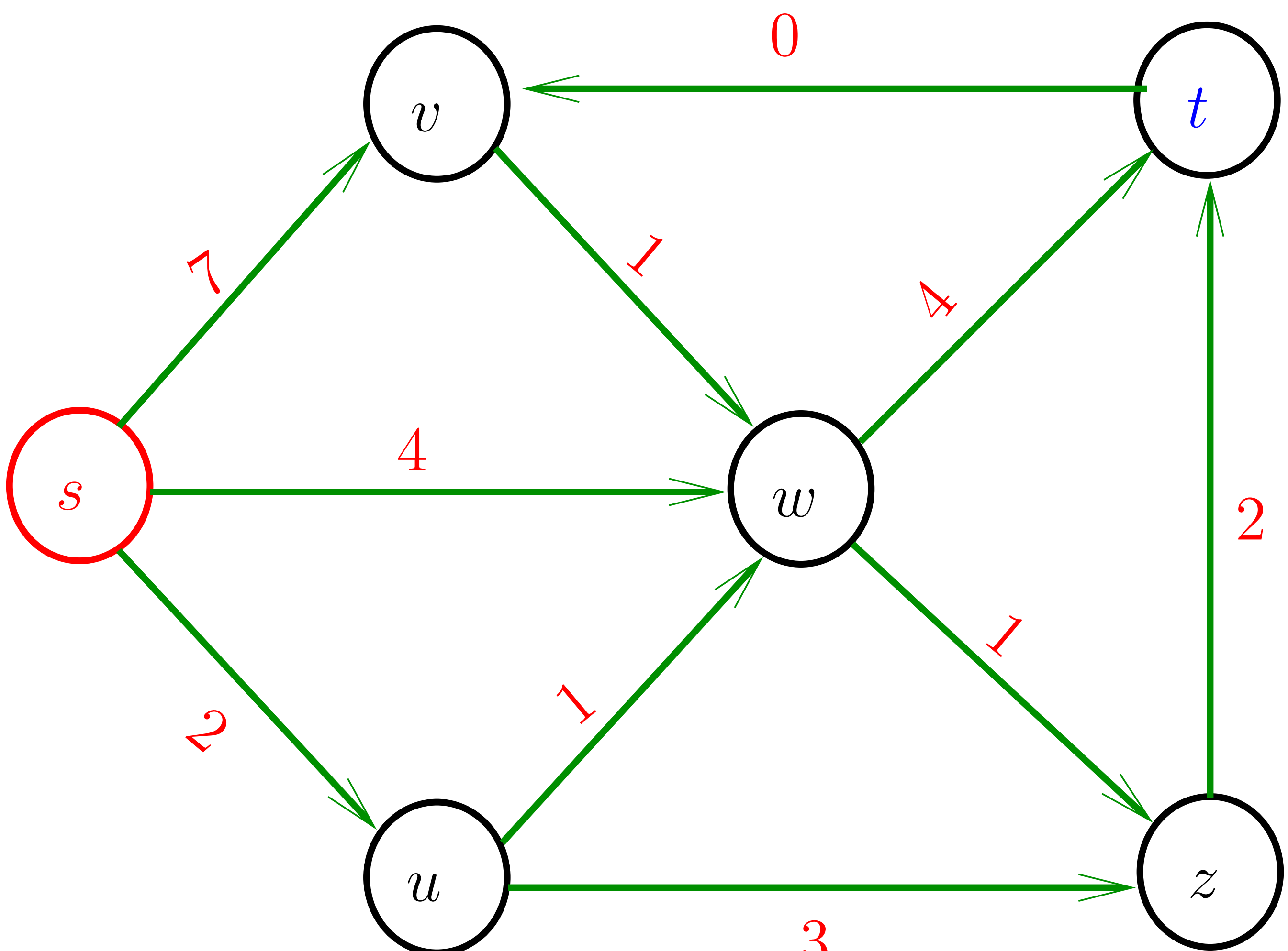
    se $y(j) > y(i) + c(ij)$ **então**

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

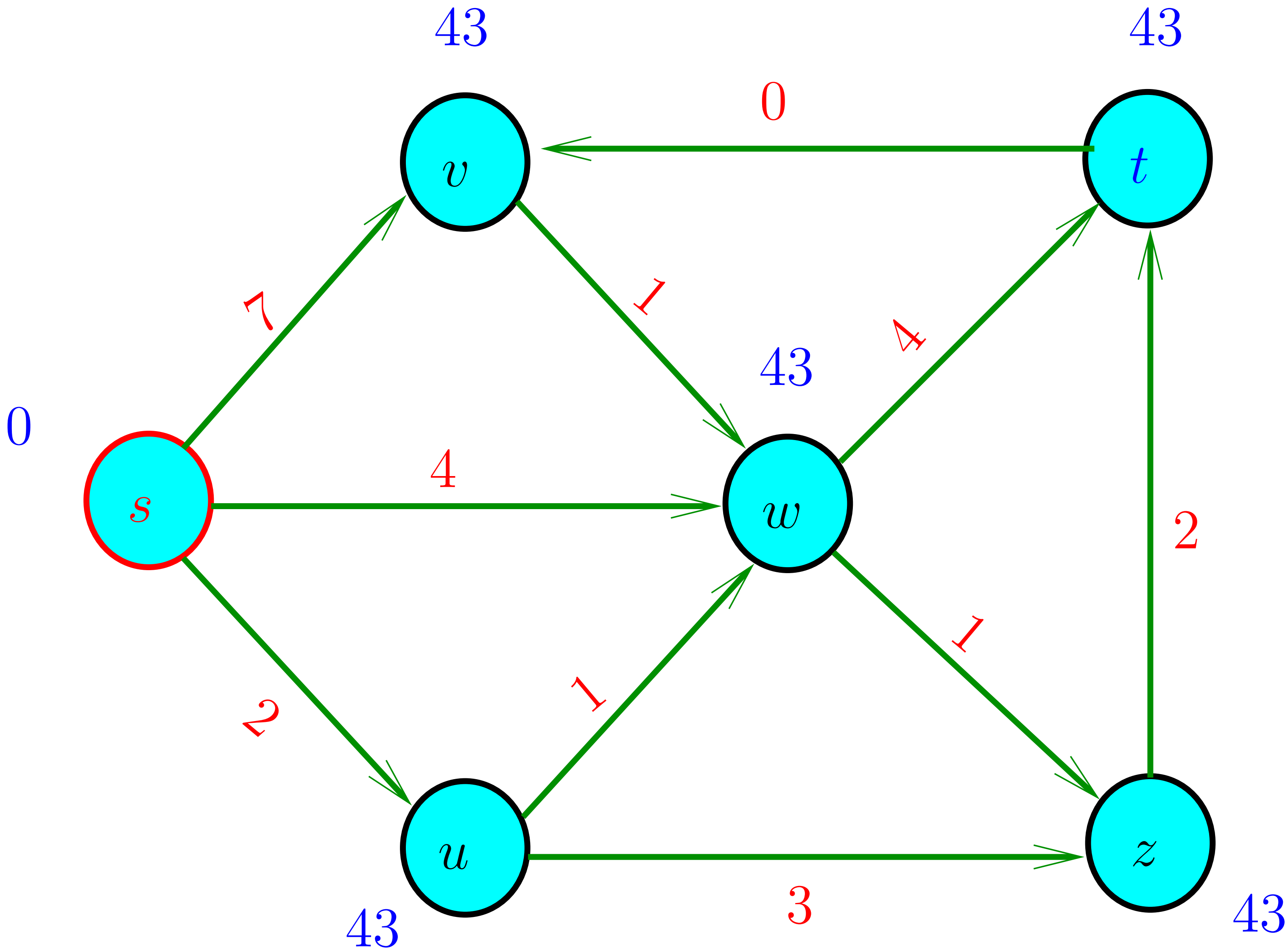
      $\pi(j) \leftarrow i$

devolva π e y

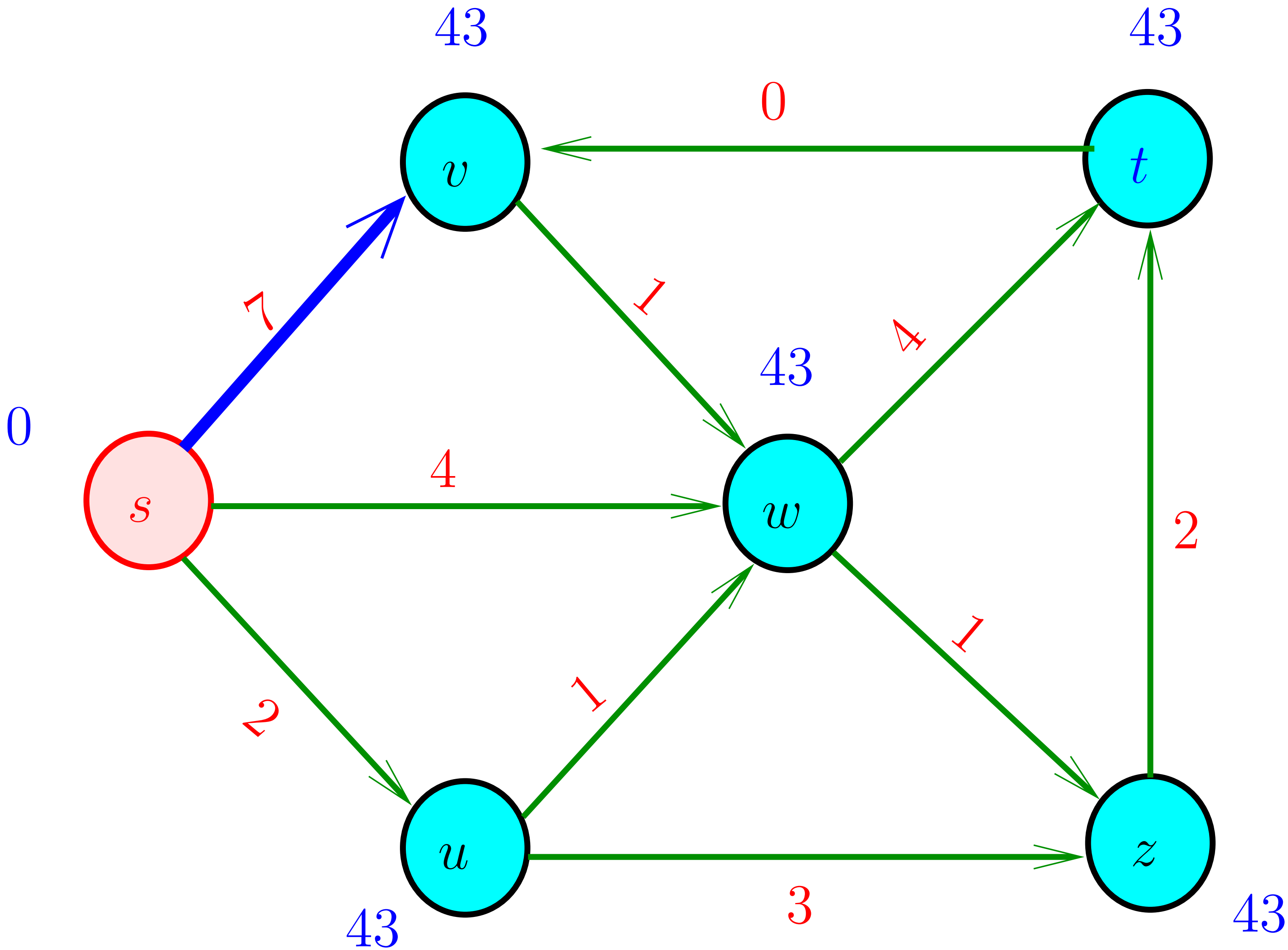
Simulação



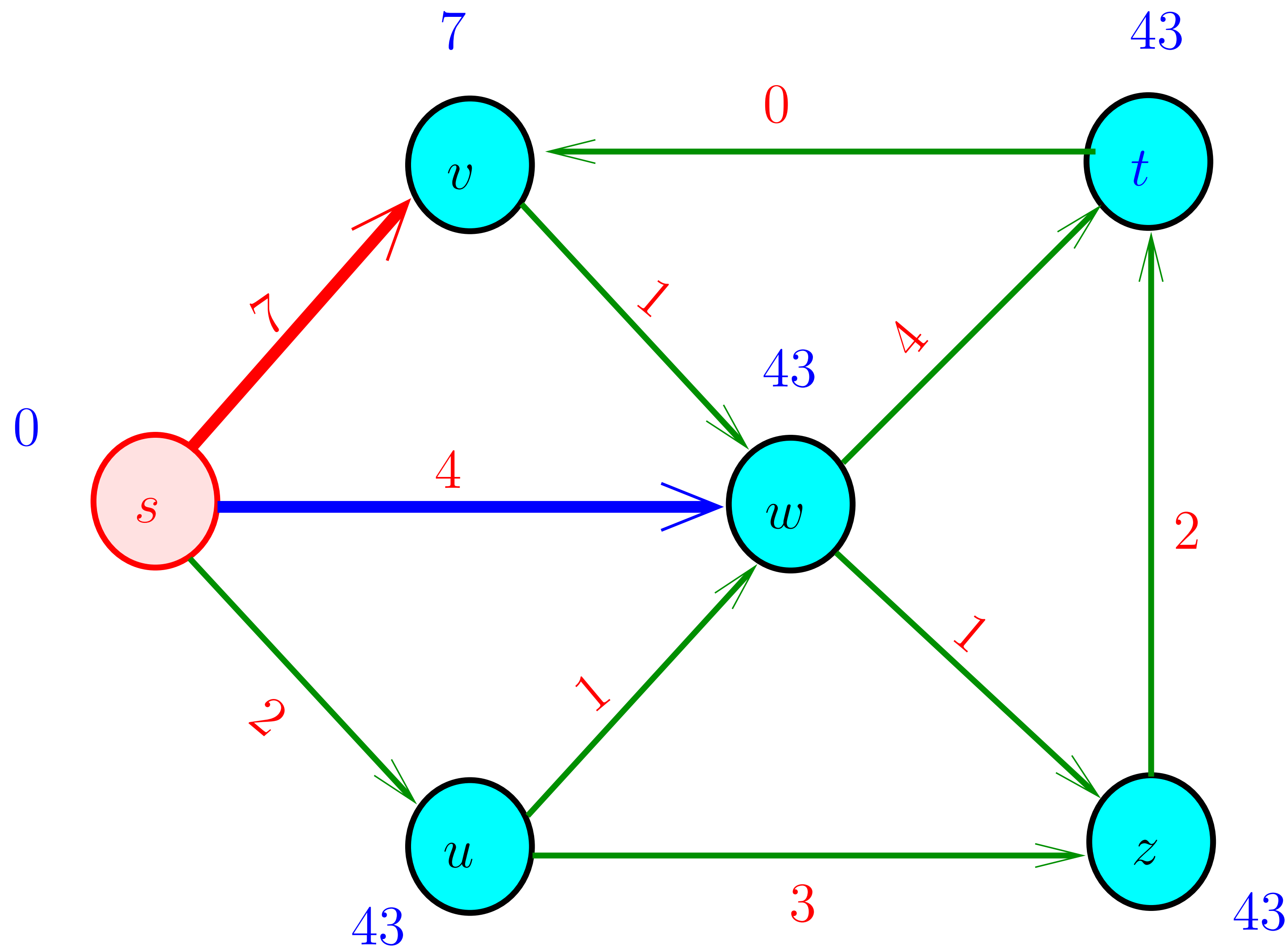
Simulação



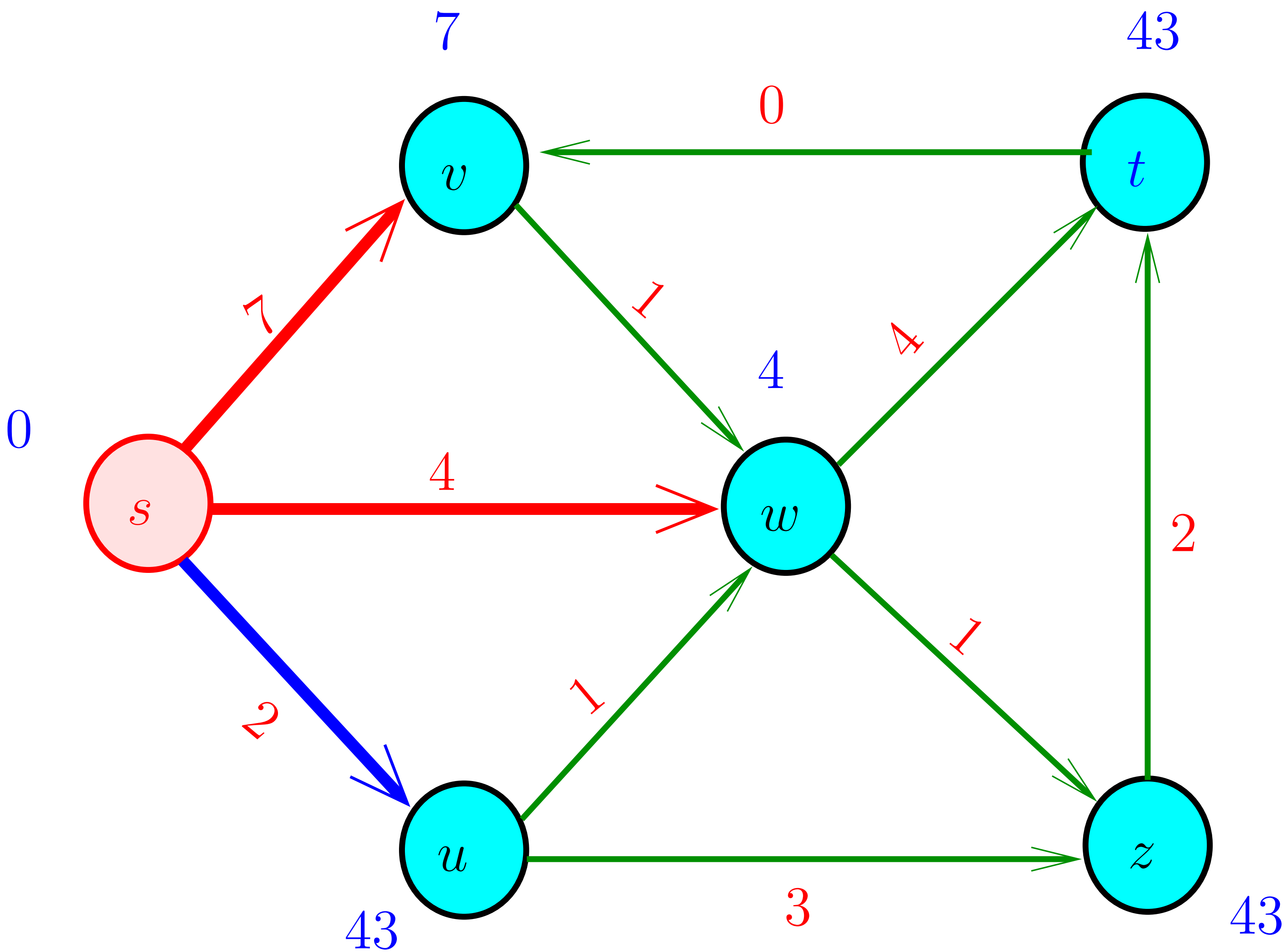
Simulação



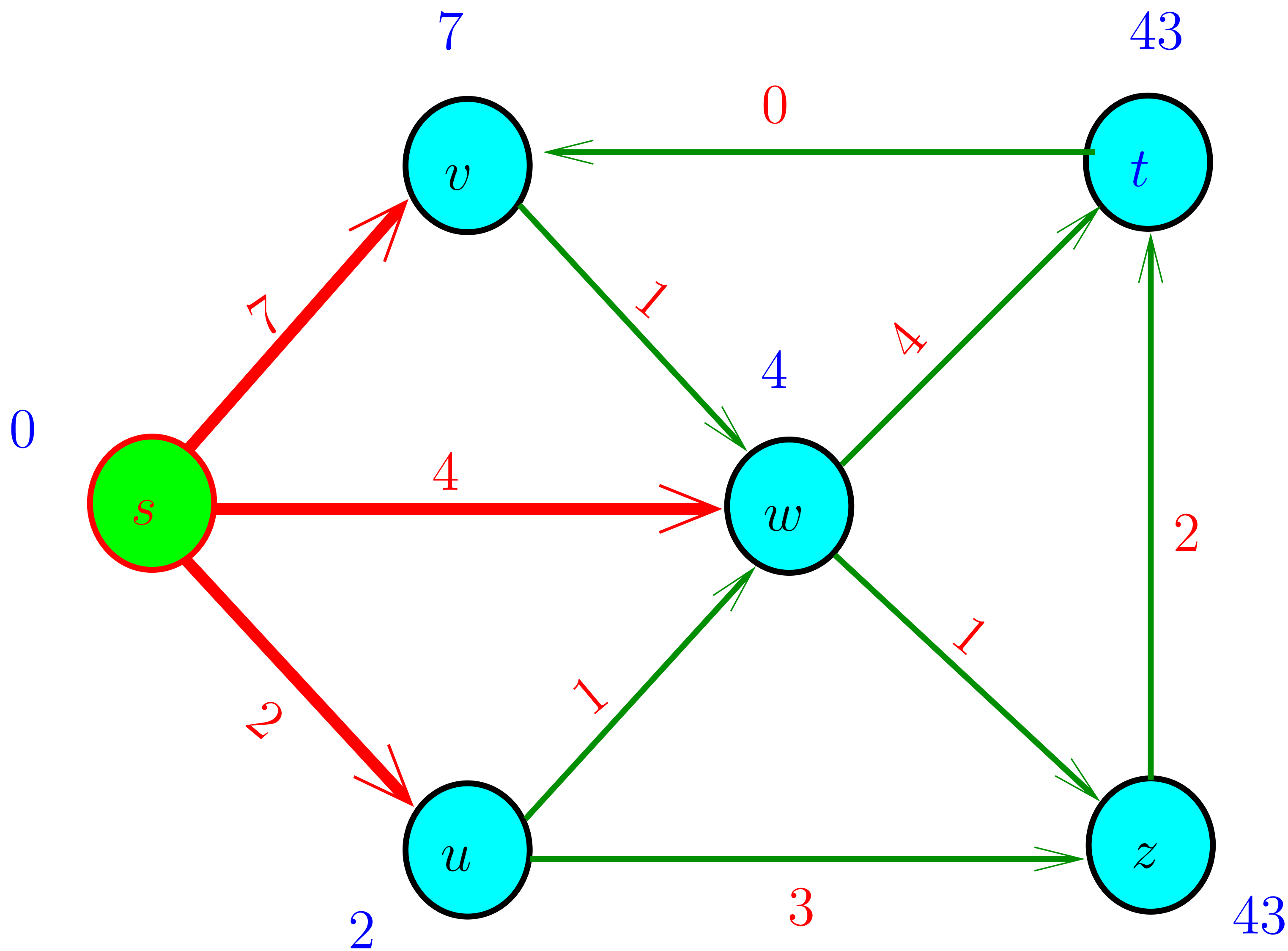
Simulação



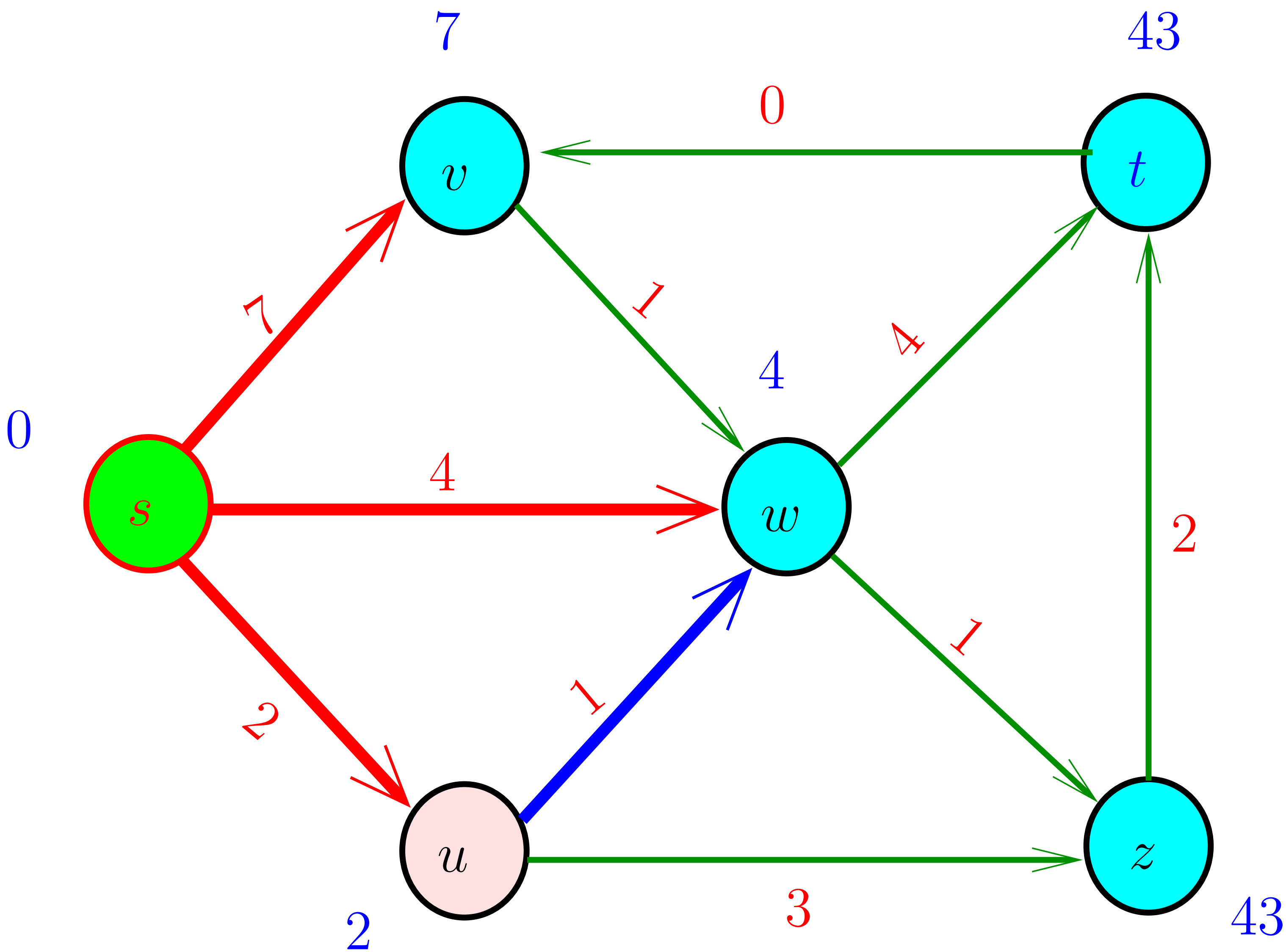
Simulação



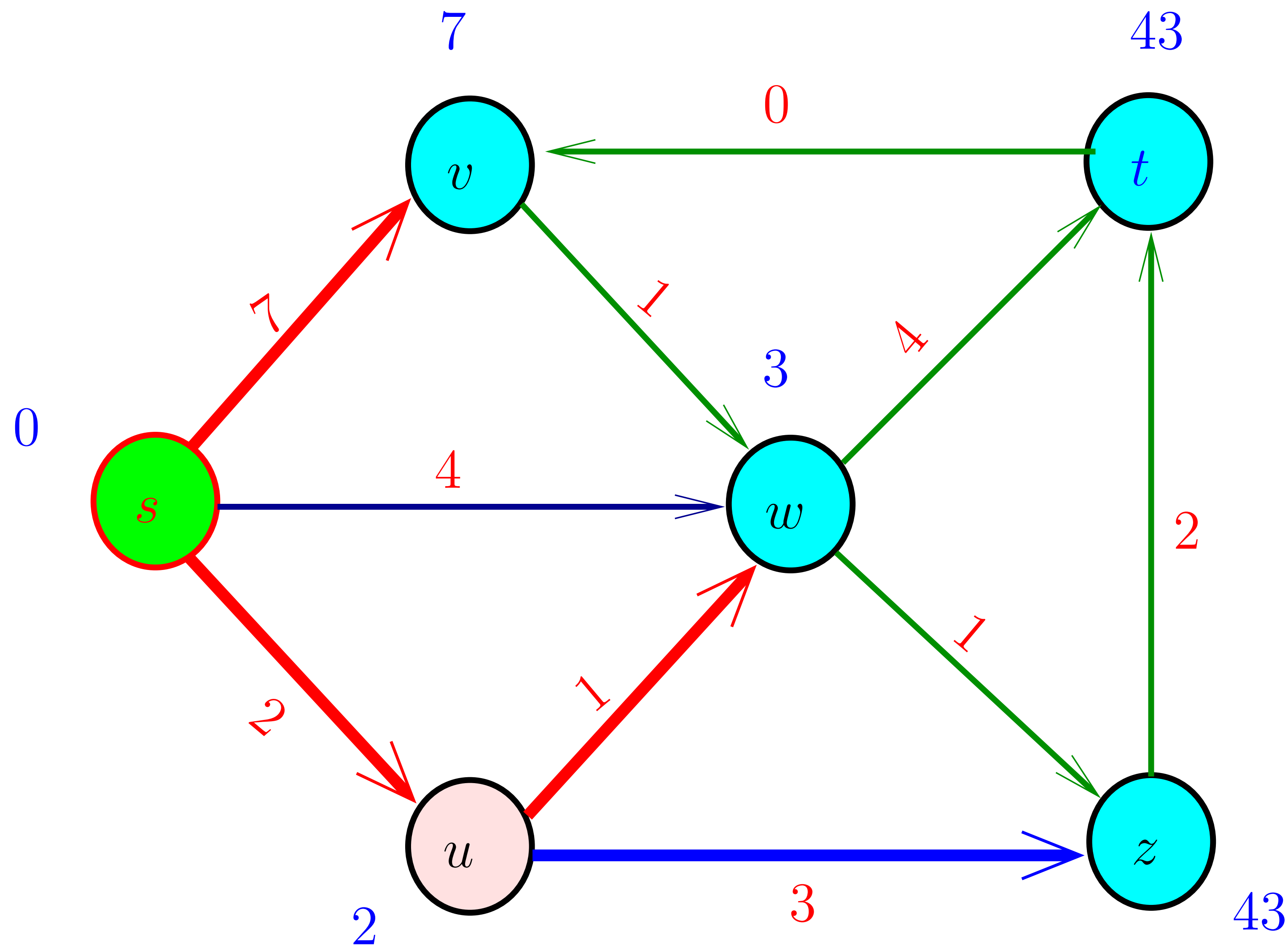
Simulação



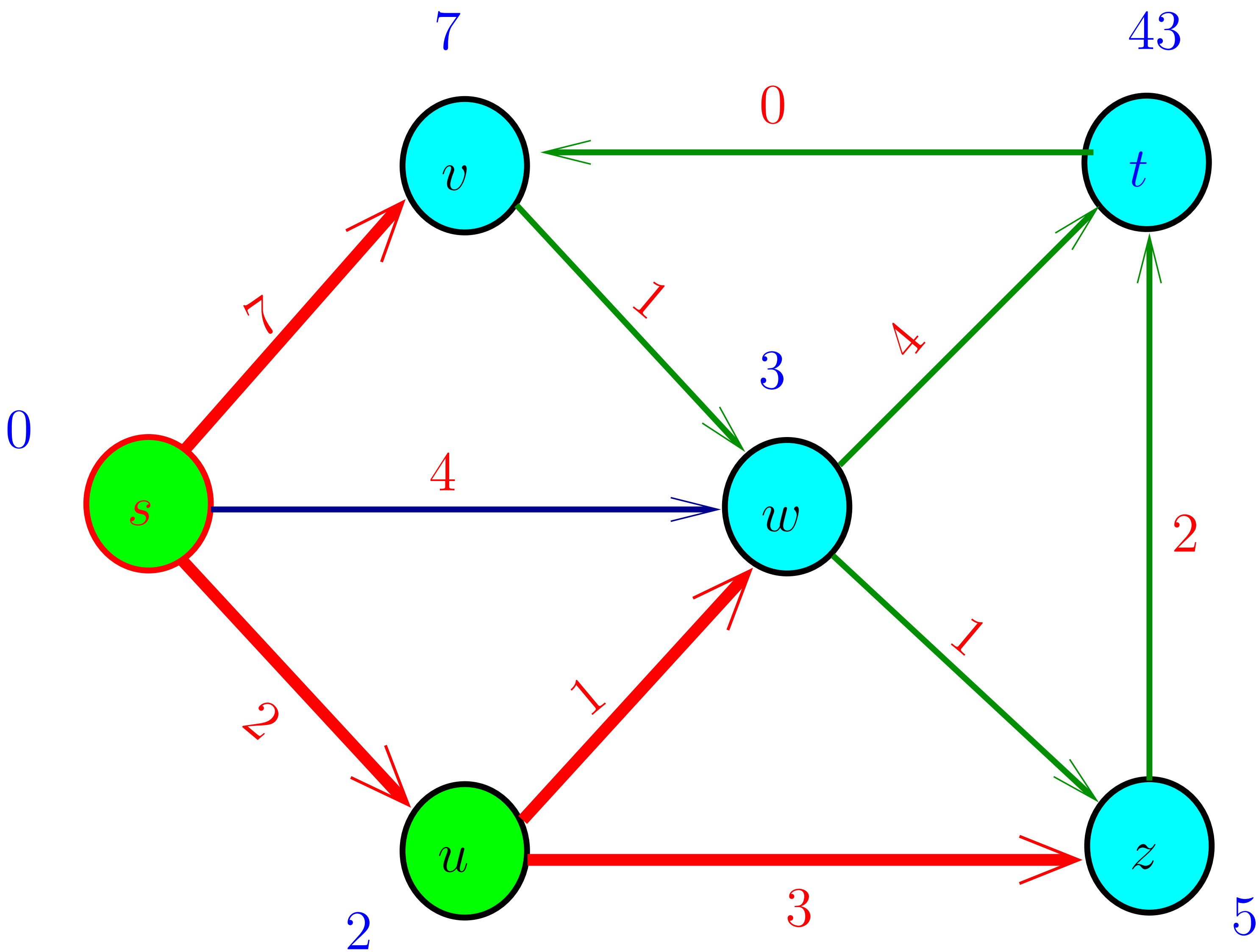
Simulação



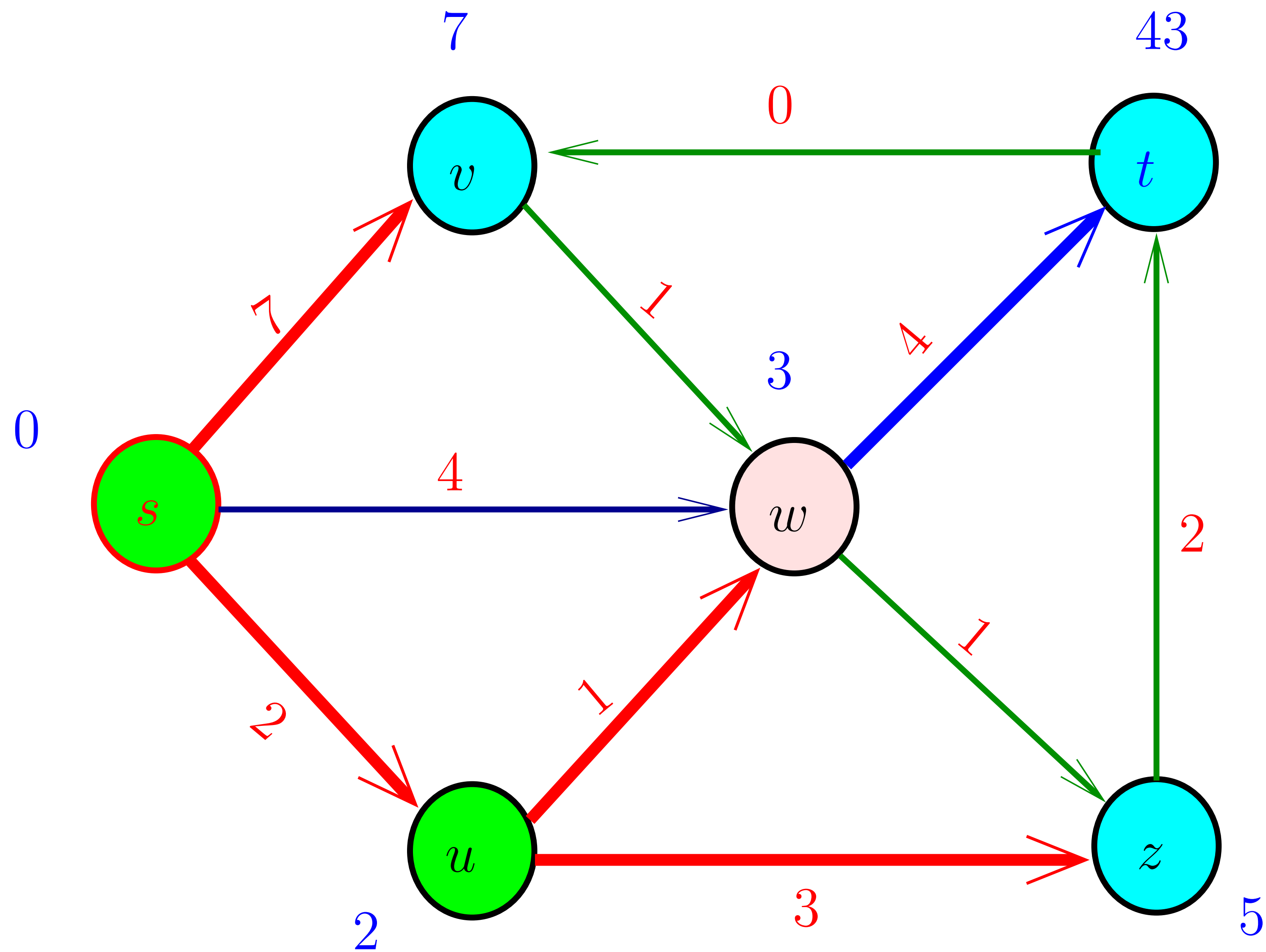
Simulação



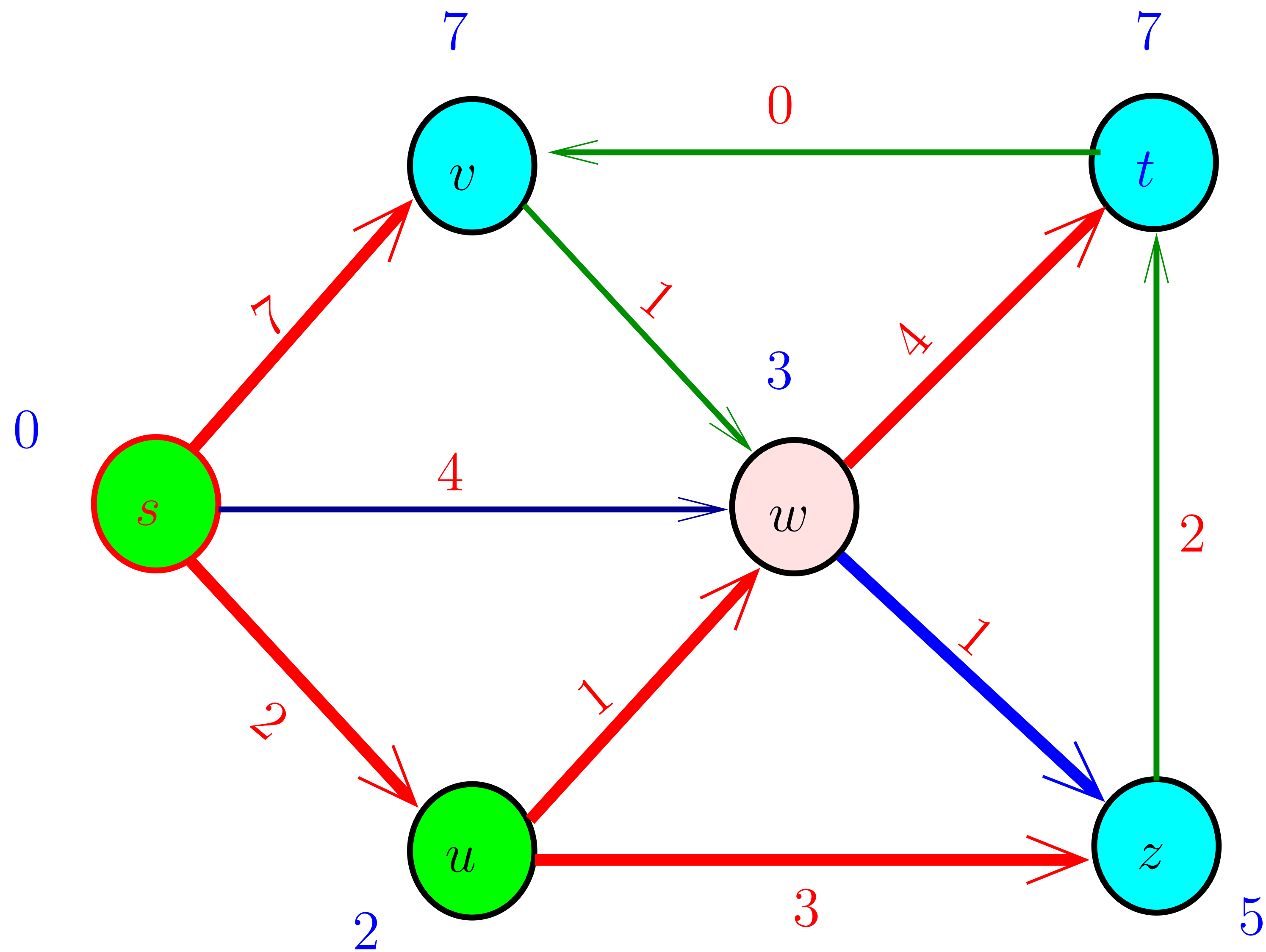
Simulação



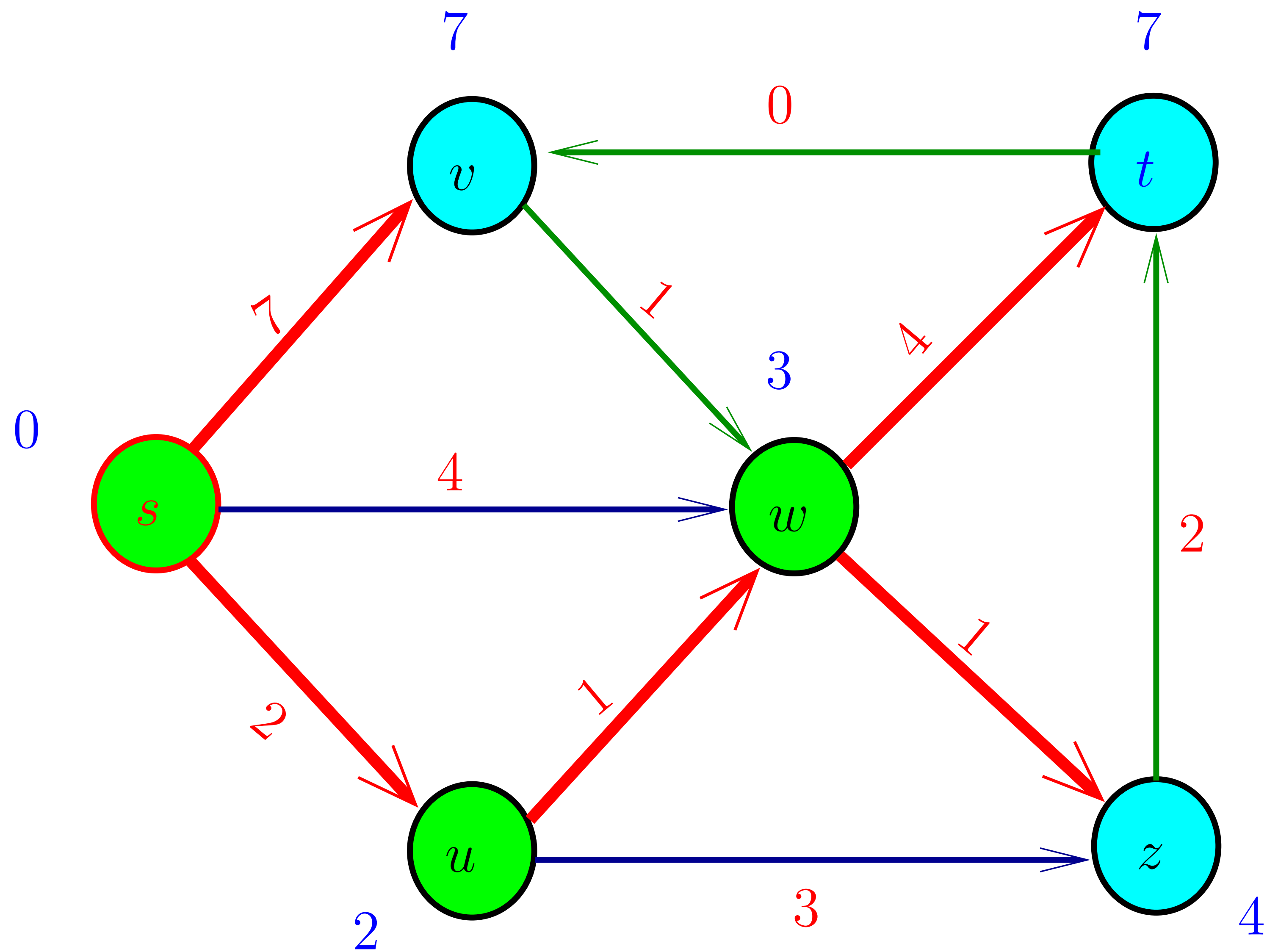
Simulação



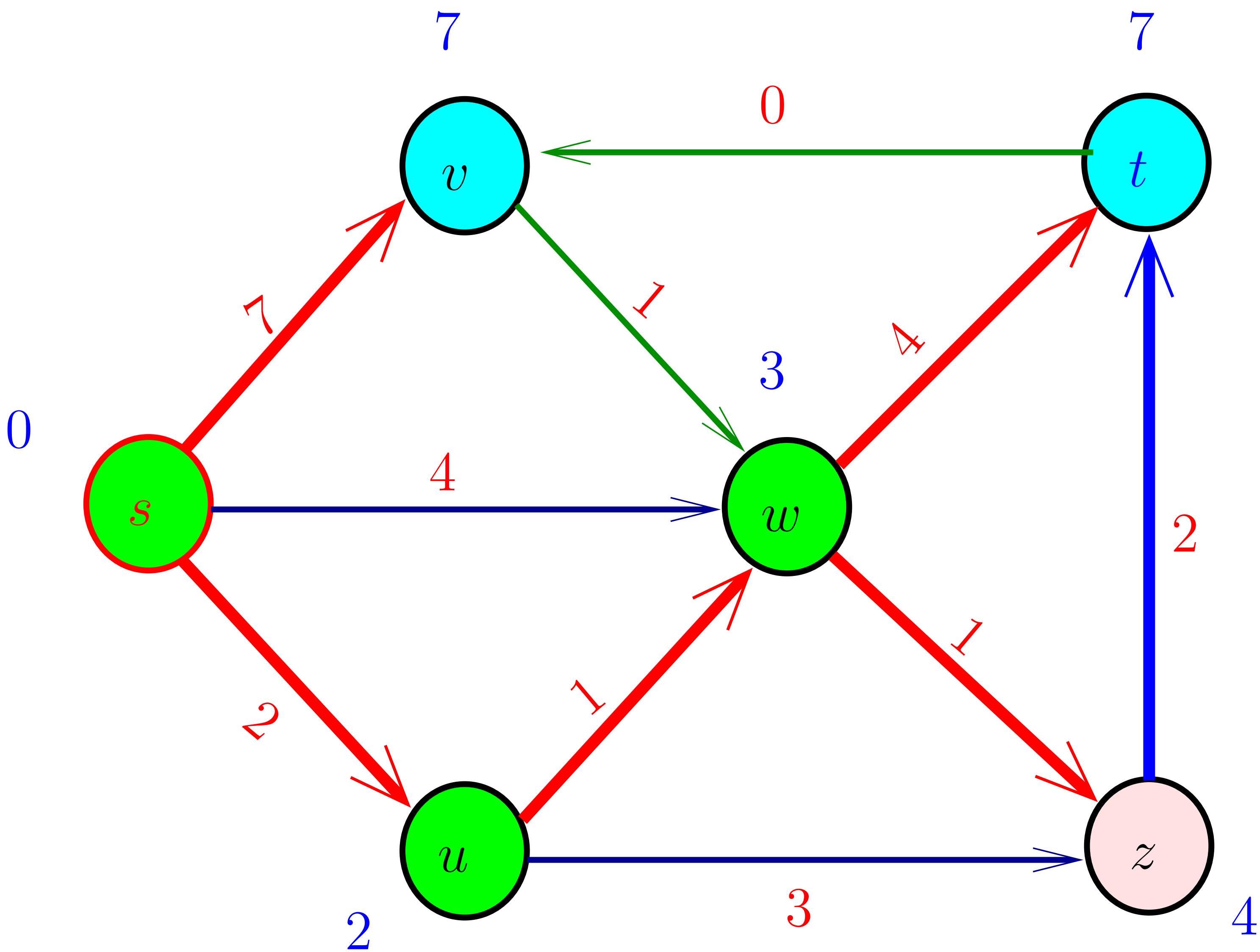
Simulação



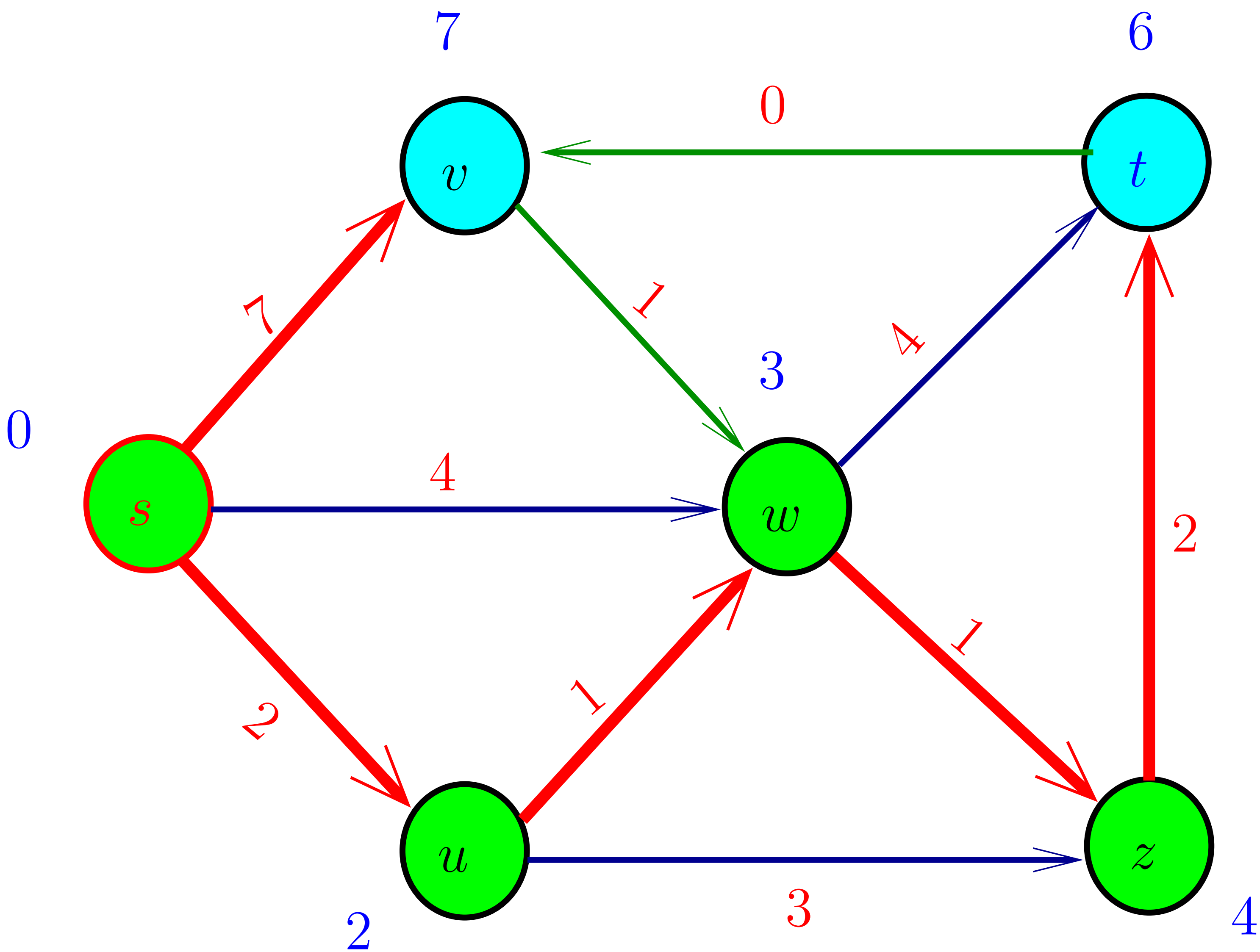
Simulação



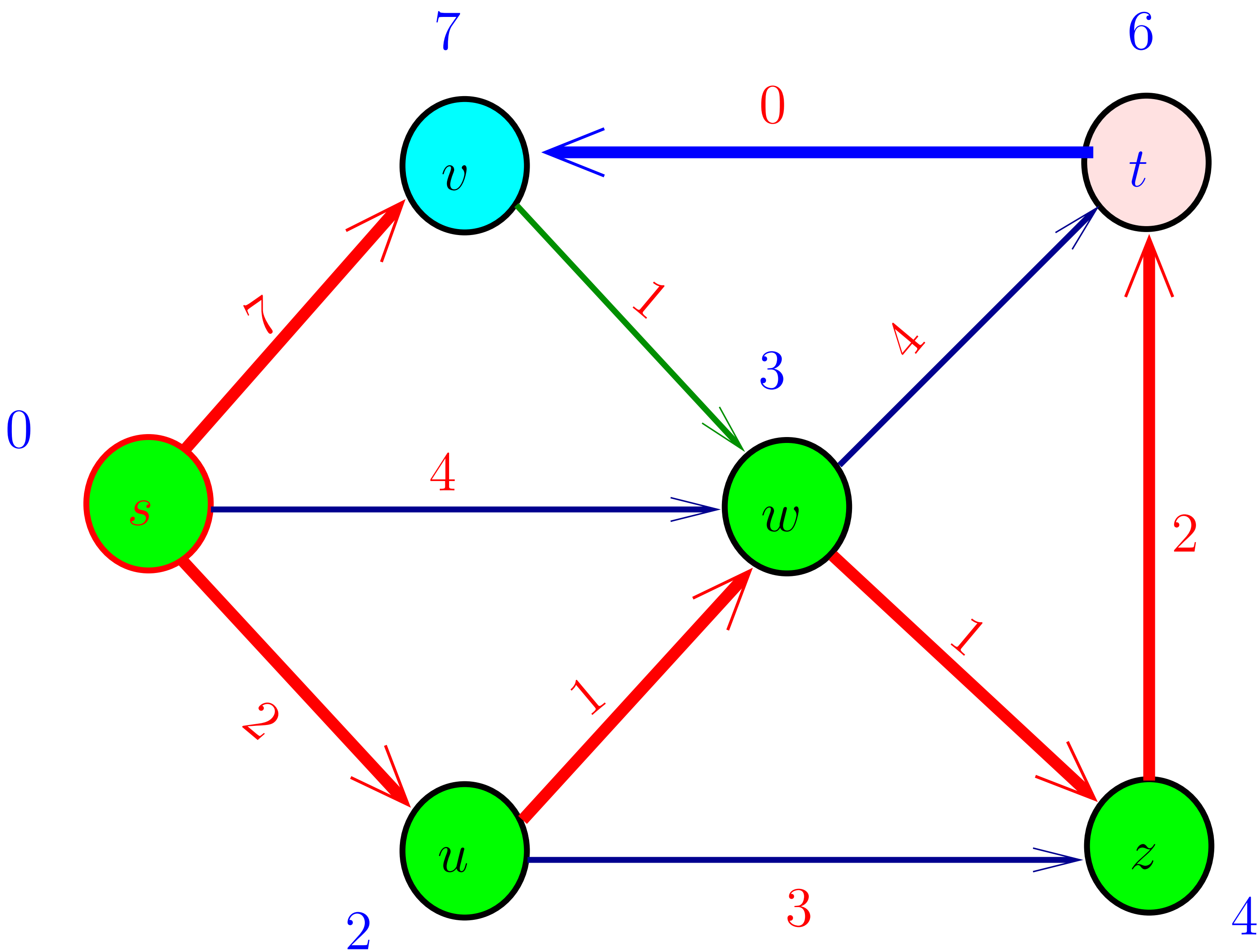
Simulação



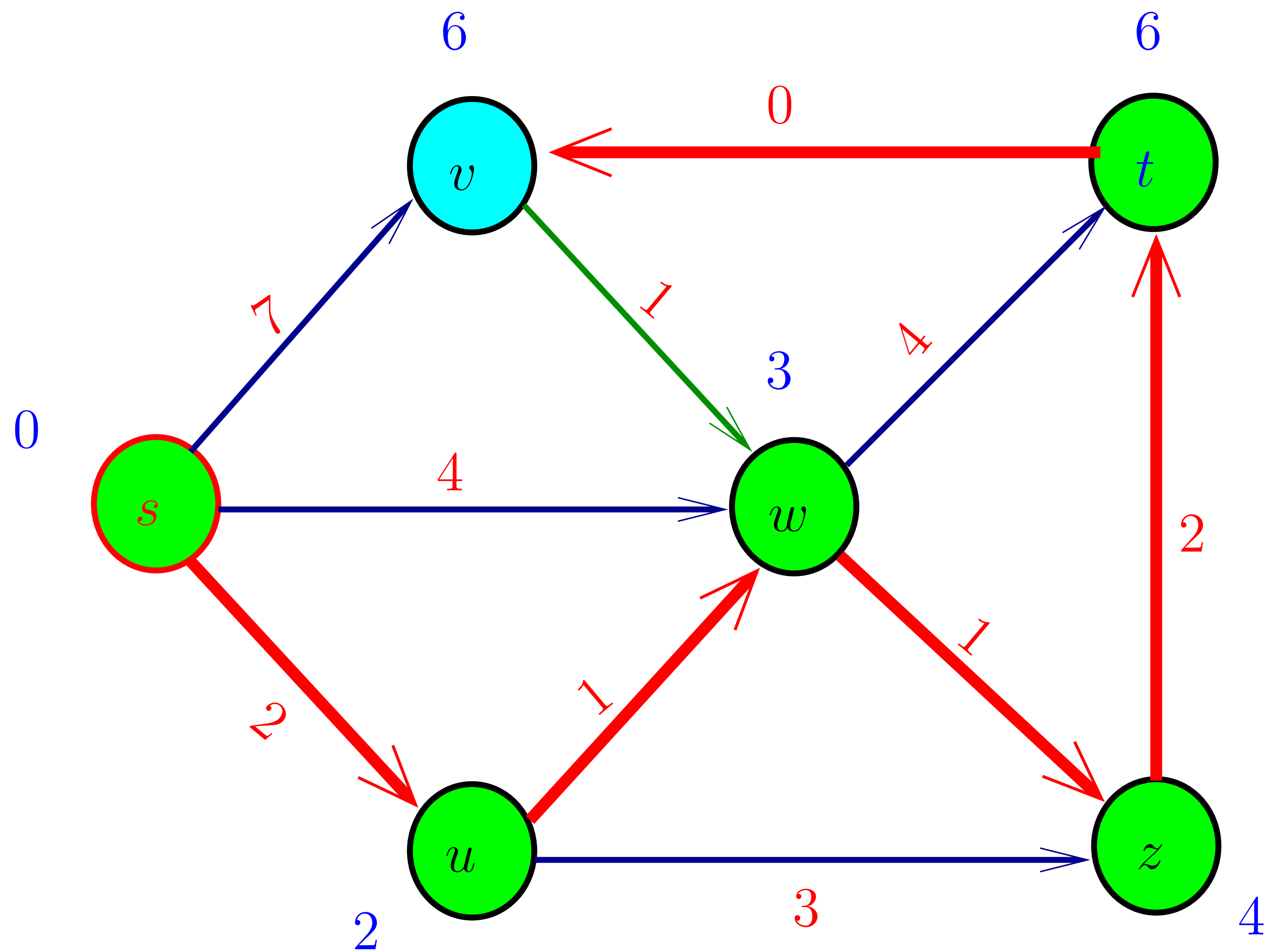
Simulação



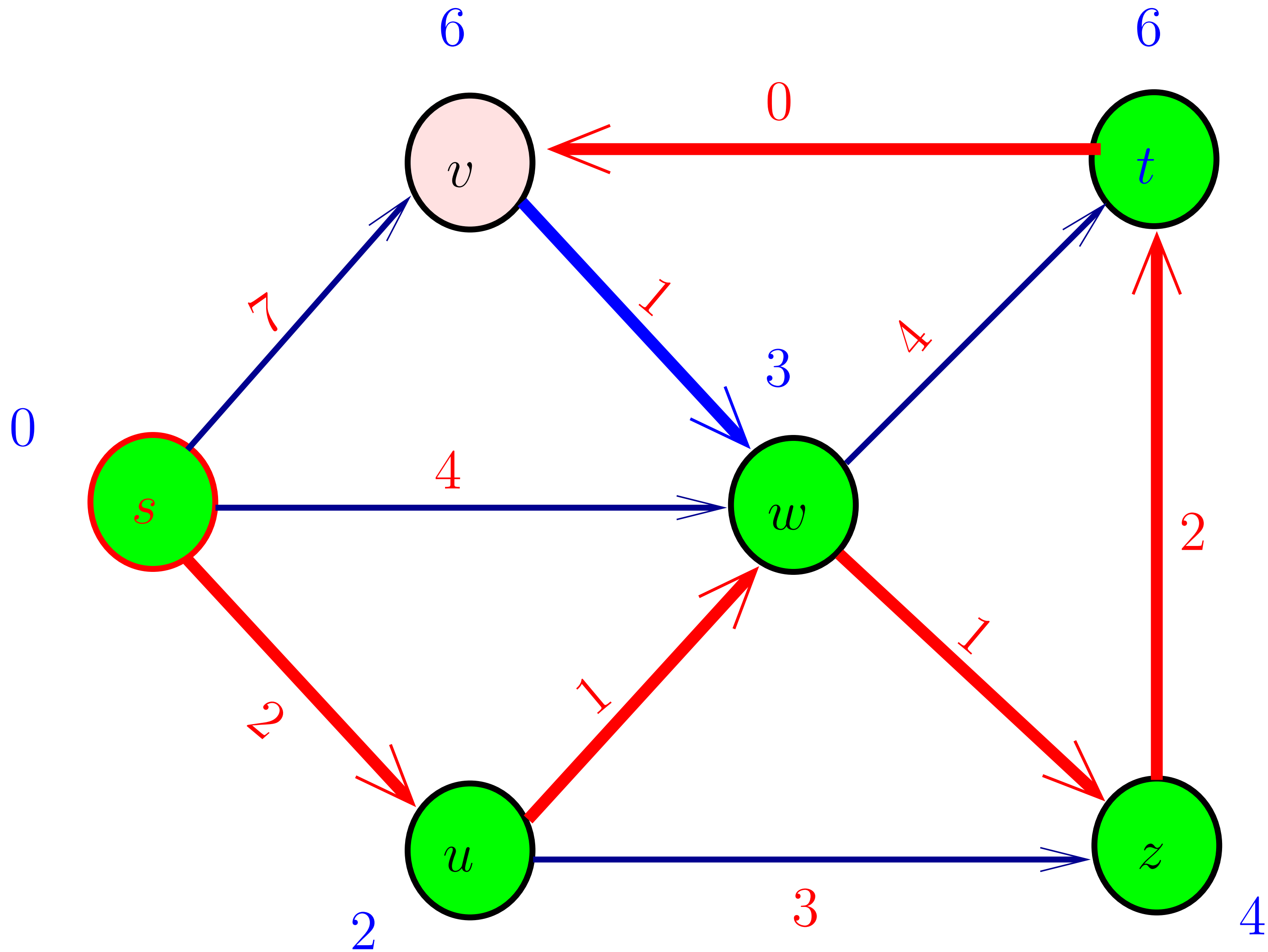
Simulação



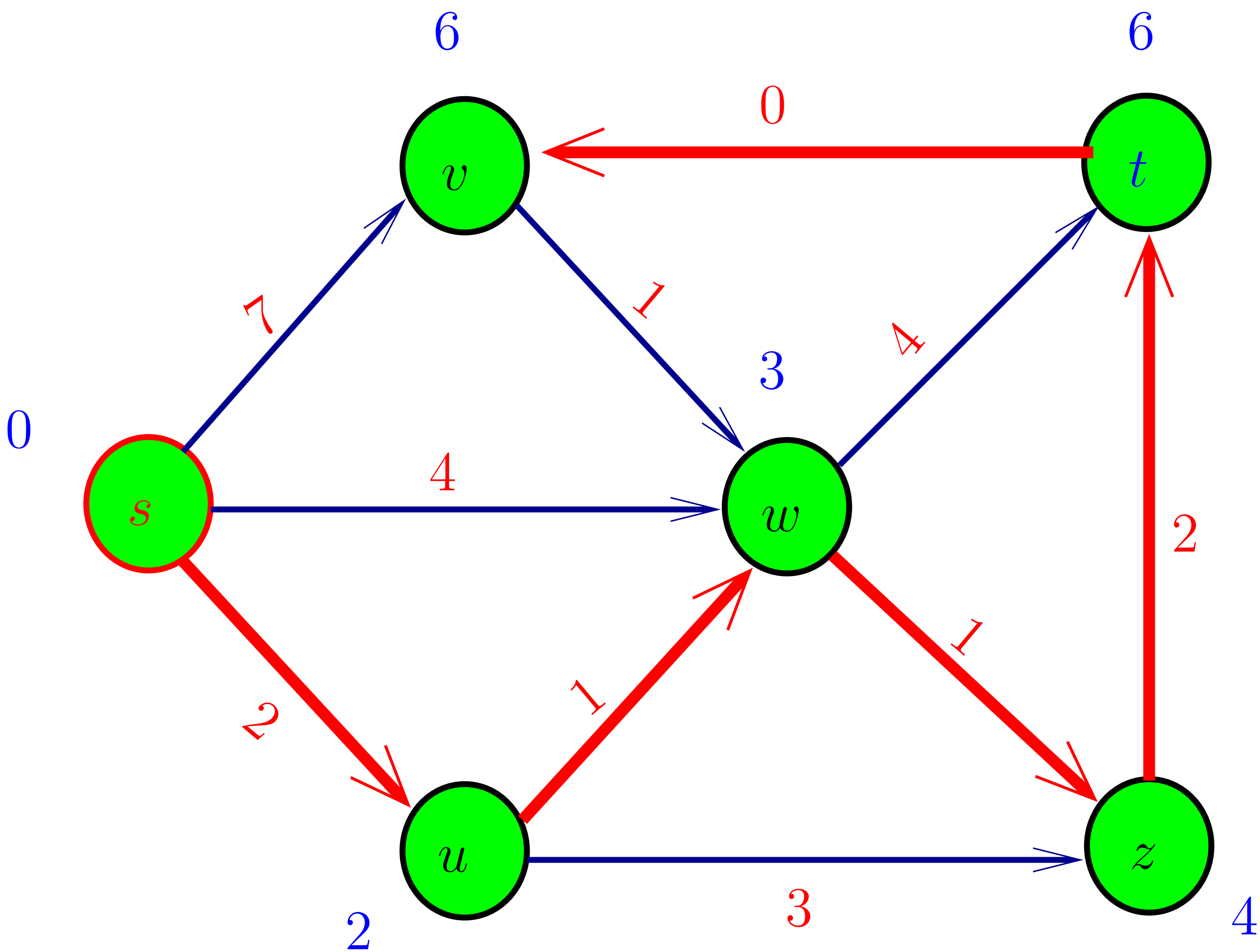
Simulação



Simulação



Simulação



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

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

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 ∞

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

$y(s) \leftarrow 0$

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

$= \mathrm{O}(m \lg n)$ (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	$\mathrm{O}(1)$	$\mathrm{O}(\lg(nC)$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 $\mathrm{O}(m + n \lg n)$.