Análise e Síntese de Algoritmos

Caminhos Mais Curtos com Fonte Única [CLRS, Cap. 24]

2011/2012

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares
  - Árvores abrangentes
  - Caminhos mais curtos
  - Fluxos máximos
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica
  - Algoritmos greedy
- Tópicos Adicionais [CLRS, Cap.32-35]
  - Emparelhamento de Cadeias de Caracteres
  - Complexidade Computacional
  - Algoritmos de Aproximação

Resumo

Caminhos Mais Curtos

Definições

- Dado um grafo $G = (V, E)$, dirigido, com uma função de pesos $w : E \rightarrow I\!R$, define-se o peso de um caminho p, onde $p =< v_0, v_1, \ldots, v_k >$, como a soma dos pesos dos arcos que compõem p:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- O peso do caminho mais curto de u para v é definido por:

$$\delta(u,v) = \begin{cases} \min \{w(p) : u \rightarrow_p v\} & \text{se existe caminho de } u \text{ para } v \\ \infty & \text{caso contrário} \end{cases}$$

- Um caminho mais curto de u para v é qualquer caminho p tal que $w(p) = \delta(u,v)$

Caminhos Mais Curtos

Problemas de Caminhos Mais Curtos

- Caminhos Mais Curtos com Fonte Única (SSSPs)
  - Identificar o caminho mais curto de um vértice fonte $s \in V$ para qualquer outro vértice $v \in V$

Caminhos Mais Curtos

Problemas de Caminhos Mais Curtos

- Caminhos Mais Curtos com Fonte Única (SSSPs)
  - Identificar o caminho mais curto de um vértice fonte $s \in V$ para qualquer outro vértice $v \in V$
- Caminhos Mais Curtos com Destino Único
  - Identificar o caminho mais curto de qualquer vértice $v \in V$ para um vértice destino $t \in V$

Caminhos Mais Curtos

Problemas de Caminhos Mais Curtos

- Caminhos Mais Curtos com Fonte Única (SSSPs)
  - Identificar o caminho mais curto de um vértice fonte $s \in V$ para qualquer outro vértice $v \in V$
- Caminhos Mais Curtos com Destino Único
  - Identificar o caminho mais curto de qualquer vértice $v \in V$ para um vértice destino $t \in V$
- Caminho Mais Curto entre Par Único
  - Identificar caminho mais curto entre dois vértices u e v

Caminhos Mais Curtos

Problemas de Caminhos Mais Curtos

- Caminhos Mais Curtos com Fonte Única (SSSPs)
  - Identificar o caminho mais curto de um vértice fonte $s \in V$ para qualquer outro vértice $v \in V$
- Caminhos Mais Curtos com Destino Único
  - Identificar o caminho mais curto de qualquer vértice $v \in V$ para um vértice destino $t \in V$
- Caminho Mais Curto entre Par Único
  - Identificar caminho mais curto entre dois vértices u e v
- Caminhos Mais Curtos entre Todos os Pares (APSPs)
  - Identificar um caminho mais curto entre cada par de vértices de V

Caminhos Mais Curtos

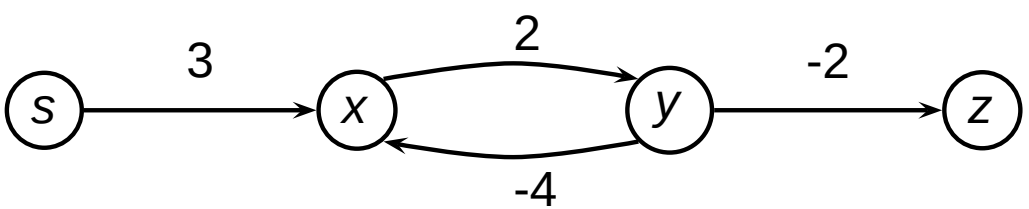
Ciclos Negativos

- Arcos podem ter pesos com valor negativo
- É possível a existência de ciclos com peso total negativo
  - Se ciclo negativo não atingível a partir da fonte s, então $\delta(s, v)$ bem definido
  - Se ciclo negativo atingível a partir da fonte s, então os pesos dos caminhos mais curtos não são bem definidos
  - Neste caso, é sempre possível encontrar um caminho mais curto de s para qualquer vértice incluído no ciclo e define-se $\delta(s, v) = -\infty$

Caminhos Mais Curtos

Ciclos Negativos

- Arcos podem ter pesos com valor negativo
- É possível a existência de ciclos com peso total negativo
  - Se ciclo negativo não atingível a partir da fonte s, então $\delta(s,v)$ bem definido
  - Se ciclo negativo atingível a partir da fonte s, então os pesos dos caminhos mais curtos não são bem definidos
  - Neste caso, é sempre possível encontrar um caminho mais curto de s para qualquer vértice incluído no ciclo e define-se $\delta(s,v) = -\infty$



$$w(< s,x,y,z >) = 3$$
$$w(< s,x,y,x,y,z >) = 1$$
$$w(< s,x,y,x,y,x,y,z >) = -1$$
$$\ldots$$

Caminhos Mais Curtos

Identificação de Ciclos Negativos

- Dijkstra: requer pesos não negativos
- Bellman-Ford: identifica ciclos negativos e reporta a sua existência

Caminhos Mais Curtos com Fonte Única

Representação de Caminhos Mais Curtos

- Para cada vértice $v \in V$ associar predecessor $\pi[v]$
- Após identificação dos caminhos mais curtos, $\pi[v]$ indica qual o vértice anterior a v num caminho mais curto de s para v
- Sub-grafo de predecessores $G_\pi = (V_\pi, E_\pi)$:

$$V_\pi = \{v \in V : \pi[v] \neq \text{NIL}\} \cup \{s\}$$

$$E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$$

Caminhos Mais Curtos com Fonte Única

Representação de Caminhos Mais Curtos

- Uma árvore de caminhos mais curtos é um sub-grafo dirigido $G' = (V', E')$, $V' \subseteq V$ e $E' \subseteq E$, tal que:
  - V' é o conjunto de vértices atingíveis a partir de s em G
  - G' forma uma árvore com raiz s
  - Para todo o $v \in V'$, o único caminho de s para v em G' é um caminho mais curto de s para v em G

Observações:

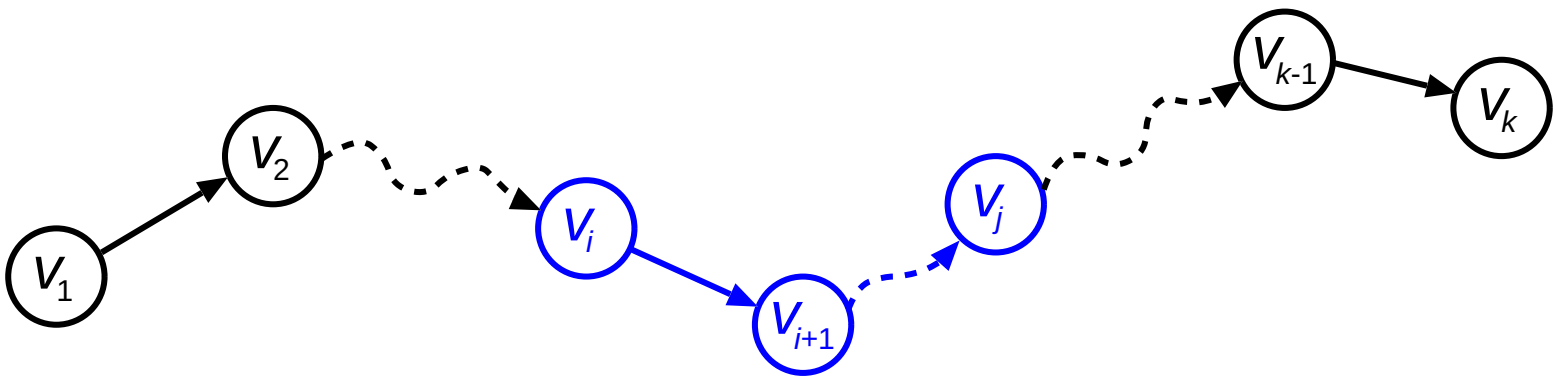
- Após identificação dos caminhos mais curtos de G a partir de fonte s, G' é dado por $G_\pi = (V_\pi, E_\pi)$
- Dados os mesmos grafo G e vértice fonte s, G' não é necessariamente único

Propriedades dos Caminhos Mais Curtos

Sub-estrutura óptima

Sub-caminhos de caminhos mais curtos são caminhos mais curtos

- Seja $p =< v_1, v_2, \ldots, v_k >$ um caminho mais curto entre v_1 e v_k, e seja $p_{ij} =< v_i, v_{i+1}, \ldots, v_j >$ um sub-caminho de p entre v_i e v_j.
- Então p_{ij} é um caminho mais curto entre v_i e v_j
  - Porquê? Se existisse caminho mais curto entre v_i e v_j então seria possível construir caminho entre v_1 e v_k mais curto do que p;
    Contradição, dado que p é um caminho mais curto entre v_1 e v_k.



Propriedades dos Caminhos Mais Curtos

Sub-estrutura óptima

Sub-caminhos de caminhos mais curtos são caminhos mais curtos

- Seja $p =< v_1, v_2, \ldots, v_k >$ um caminho mais curto entre v_1 e v_k, e seja $p_{ij} =< v_i, v_{i+1}, \ldots, v_j >$ um sub-caminho de p entre v_i e v_j.
- Então p_{ij} é um caminho mais curto entre v_i e v_j
  - Porquê? Se existisse caminho mais curto entre v_i e v_j então seria possível construir caminho entre v_1 e v_k mais curto do que p;
    Contradição, dado que p é um caminho mais curto entre v_1 e v_k.

Peso de um Caminho Mais Curto

Seja $p =< s, \ldots, v >$ um caminho mais curto entre s e v, que pode ser decomposto em $p_{su} =< s, \ldots, u >$ e (u, v). Então
$\delta(s, v) = \delta(s, u) + w(u, v)$

- p_{su} é caminho mais curto entre s e u
- $\delta(s, v) = w(p) = w(p_{su}) + w(u, v) = \delta(s, u) + w(u, v)$

Propriedades dos Caminhos Mais Curtos

Relação caminho mais curto com arcos do grafo

Para todos os arcos $(u,v) \in E$ verifica-se $\delta(s,v) \leq \delta(s,u) + w(u,v)$

- Caminho mais curto de s para v não pode ter mais peso do que qualquer outro caminho de s para v
- Assim, peso do caminho mais curto de s para v não superior ao peso do caminho mais curto de s para u seguido do arco (u,v) (i.e. exemplo de um dos caminhos de s para v)

Operação de Relaxação

Operação de Relaxação

- Operação básica dos algoritmos para cálculo dos caminhos mais curtos com fonte única.
- d[v]: denota a estimativa do caminho mais curto de s para v; limite superior no valor do peso do caminho mais curto;
- Algoritmos aplicam sequência de relaxações dos arcos de G após inicialização para actualizar a estimativa do caminho mais curto

Initialize-Single-Source(G, s)

1 **for** each $v \in V[G]$
2 **do** $d[v] \leftarrow \infty$
3 $\pi[v] \leftarrow NIL$
4 $d[s] \leftarrow 0$

Relax(u, v, w)

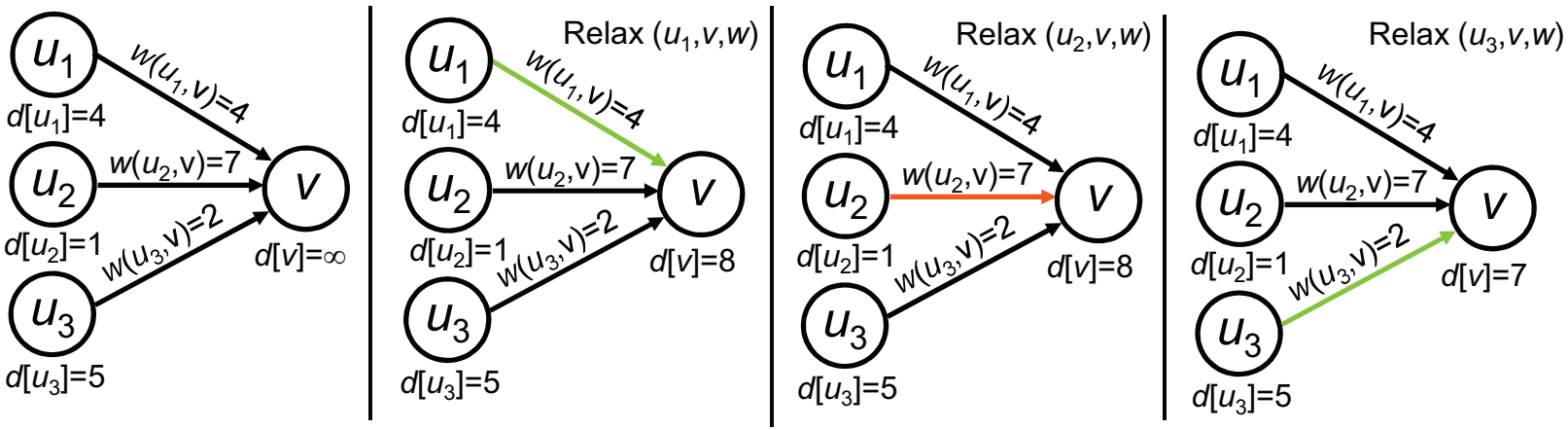
1 **if** $d[v] > d[u] + w(u, v)$
2 **then** $d[v] \leftarrow d[u] + w(u, v)$
3 $\pi[v] \leftarrow u$

Propriedades da Relaxação

Propriedades da Relaxação

Após relaxar arco (u,v), temos que $d[v] \leq d[u] + w(u,v)$

- Se $d[v] > d[u] + w(u,v)$ antes da relaxação, então $d[v] = d[u] + w(u,v)$ após relaxação
- Se $d[v] \leq d[u] + w(u,v)$ antes da relaxação, então $d[v] \leq d[u] + w(u,v)$ após relaxação
- Em qualquer caso, $d[v] \leq d[u] + w(u,v)$ após relaxação



Propriedades da Relaxação (2)

Propriedades da Relaxação

$d[v] \geq \delta(s,v)$ para qualquer $v \in V[G]$ e para qualquer sequência de passos de relaxação nos arcos de G. Se $d[v]$ atinge o valor $\delta(s,v)$, então o valor não é mais alterado

- $d[v] \geq \delta(s,v)$ é válido após inicialização
- Prova por contradição para os restantes casos:
  - Seja v o primeiro vértice para o qual a relaxação do arco (u,v) causa $d[v] < \delta(s,v)$
  - Após relaxar arco: $d[u] + w(u,v) = d[v] < \delta(s,v) \leq \delta(s,u) + w(u,v)$
  - Pelo que, $d[u] < \delta(s,u)$ antes da relaxação de (u,v); Contradição, dado que v seria o primeiro vértice para o qual $d[v] < \delta(s,v)$
- Após ter $d[v] = \delta(s,v)$, o valor de $d[v]$ não pode decrescer; e pela relaxação também não pode crescer!

Propriedades da Relaxação (3)

Propriedades da Relaxação

Seja $p =< s, \ldots, u, v >$ um caminho mais curto em G, e seja Relax(u, v, w) executada no arco (u, v). Se $d[u] = \delta(s, u)$ antes da chamada a Relax(u, v, w), então $d[v] = \delta(s, v)$ após a chamada a Relax(u, v, w)

- Se $d[u] = \delta(s, u)$ então valor de $d[u]$ não é mais alterado
- Após relaxar arco (u, v):
  $d[v] \leq d[u] + w(u, v) = \delta(s, u) + w(u, v) = \delta(s, v)$
- Mas, $d[v] \geq \delta(s, v)$, pelo que $d[v] = \delta(s, v)$, e não se altera !

Resumo Propriedades Caminhos Mais Curtos

Sub-estrutura óptima

Sub-caminhos de caminhos mais curtos são caminhos mais curtos

- Seja $p =< v_1, v_2, \ldots, v_k >$ um caminho mais curto entre v_1 e v_k, e seja $p_{ij} =< v_i, v_{i+1}, \ldots, v_j >$ um sub-caminho de p entre v_i e v_j.
- Então p_{ij} é um caminho mais curto entre v_i e v_j

Sub-estrutura óptima

Seja $p =< s, \ldots, v >$ um caminho mais curto entre s e v, que pode ser decomposto em $p_{su} =< s, \ldots, u >$ e (u, v). Então $\delta(s, v) = \delta(s, u) + w(u, v)$

Relação caminho mais curto com arcos do grafo

Para todos os arcos $(u, v) \in E$ verifica-se $\delta(s, v) \leq \delta(s, u) + w(u, v)$

Resumo Propriedades Operação Relaxação

Propriedades da Relaxação

Após relaxar arco (u,v), temos que $d[v] \leq d[u] + w(u,v)$

Propriedades da Relaxação (2)

$d[v] \geq \delta(s,v)$ para qualquer $v \in V[G]$ e para qualquer sequência de passos de relaxação nos arcos de G. Se $d[v]$ atinge o valor $\delta(s,v)$, então o valor não é mais alterado

Propriedades da Relaxação (3)

Seja $p =< s,\ldots,u,v >$ um caminho mais curto em G, e seja Relax(u,v,w) executada no arco (u,v). Se $d[u] = \delta(s,u)$ antes da chamada a Relax(u,v,w), então $d[v] = \delta(s,v)$ após a chamada a Relax(u,v,w)

Algoritmo de Dijkstra

Organização do Algoritmo

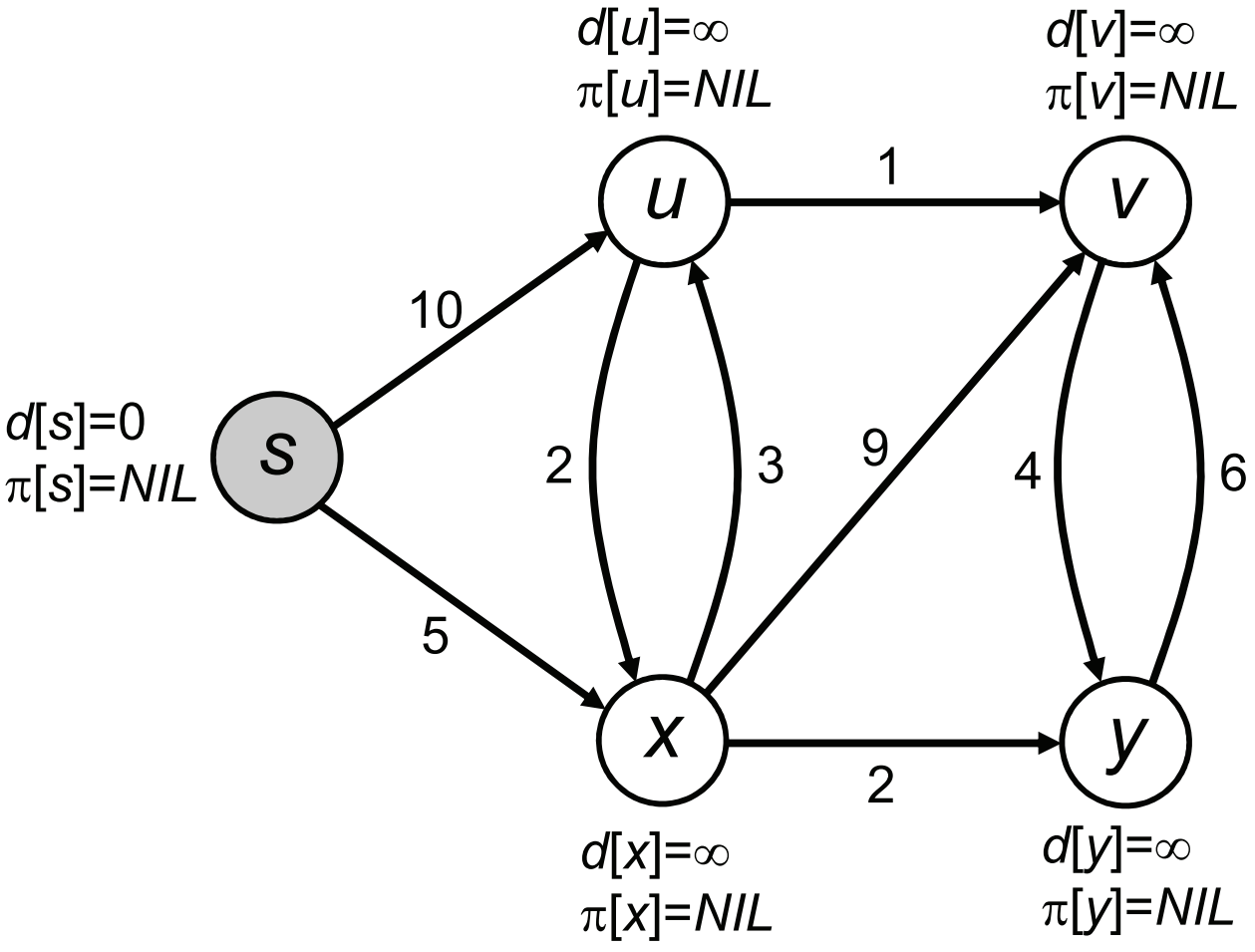
- Todos os arcos com pesos não negativos
- Algoritmo Greedy
- Algoritmo mantém conjunto de vértices S com pesos dos caminhos mais curtos já calculados
- A cada passo algoritmo selecciona vértice u em $V - S$ com menor estimativa do peso do caminho mais curto
  - vértice u é inserido em S
  - arcos que saem de u são relaxados

Algoritmo de Dijkstra

Dijkstra(G, w, s)

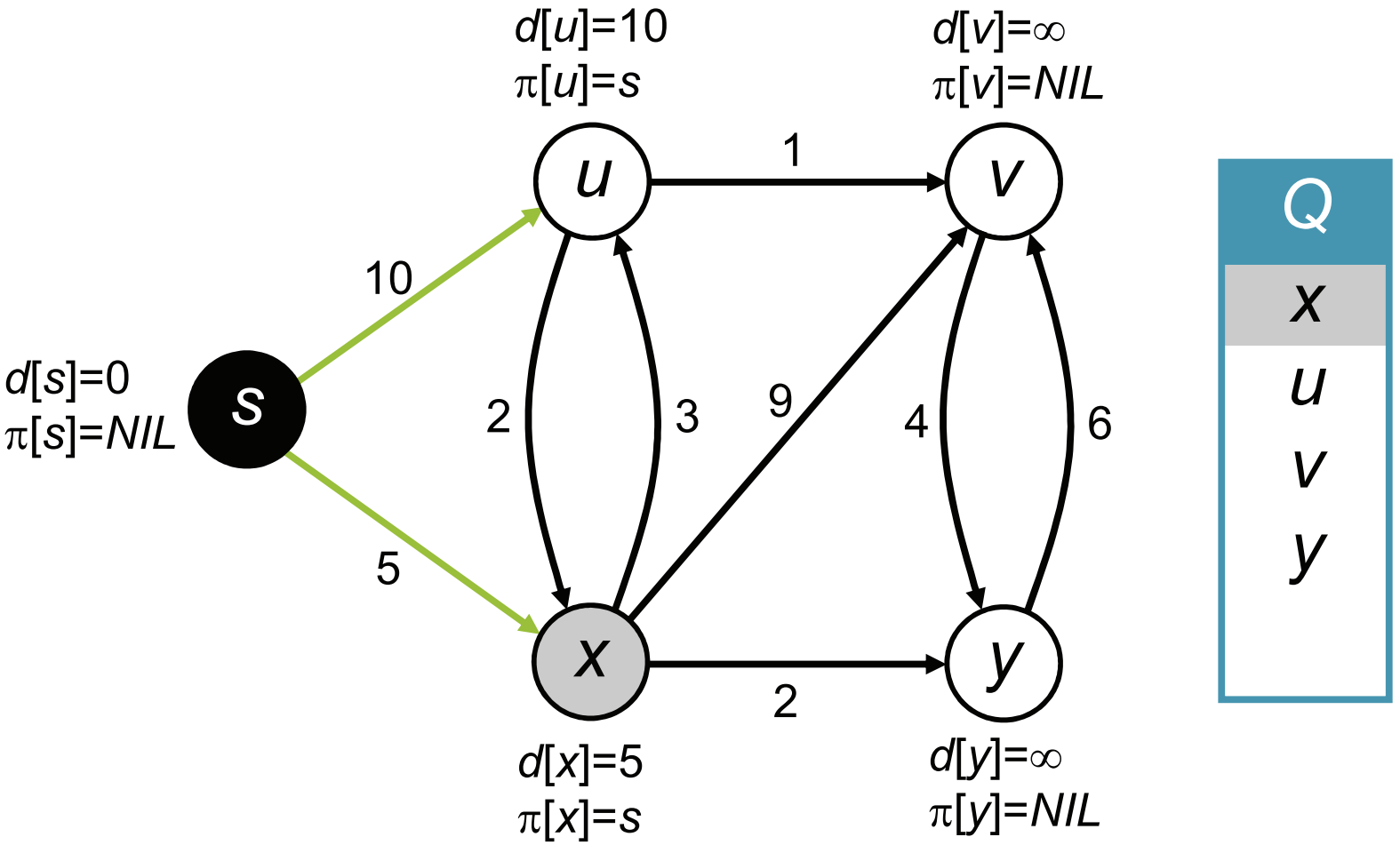
```
1  Initialize-Single-Source(G, s)
2  S ← ∅
3  Q ← V[G]                          ▷ Fila de Prioridade
4  while Q ≠ ∅
5        do u ← Extract-Min(Q)
6           S ← S ∪ {u}
7           for each v ∈ Adj[u]
8                 do Relax(u, v, w)   ▷ Actualização de Q
```

Algoritmo de Dijkstra: Exemplo

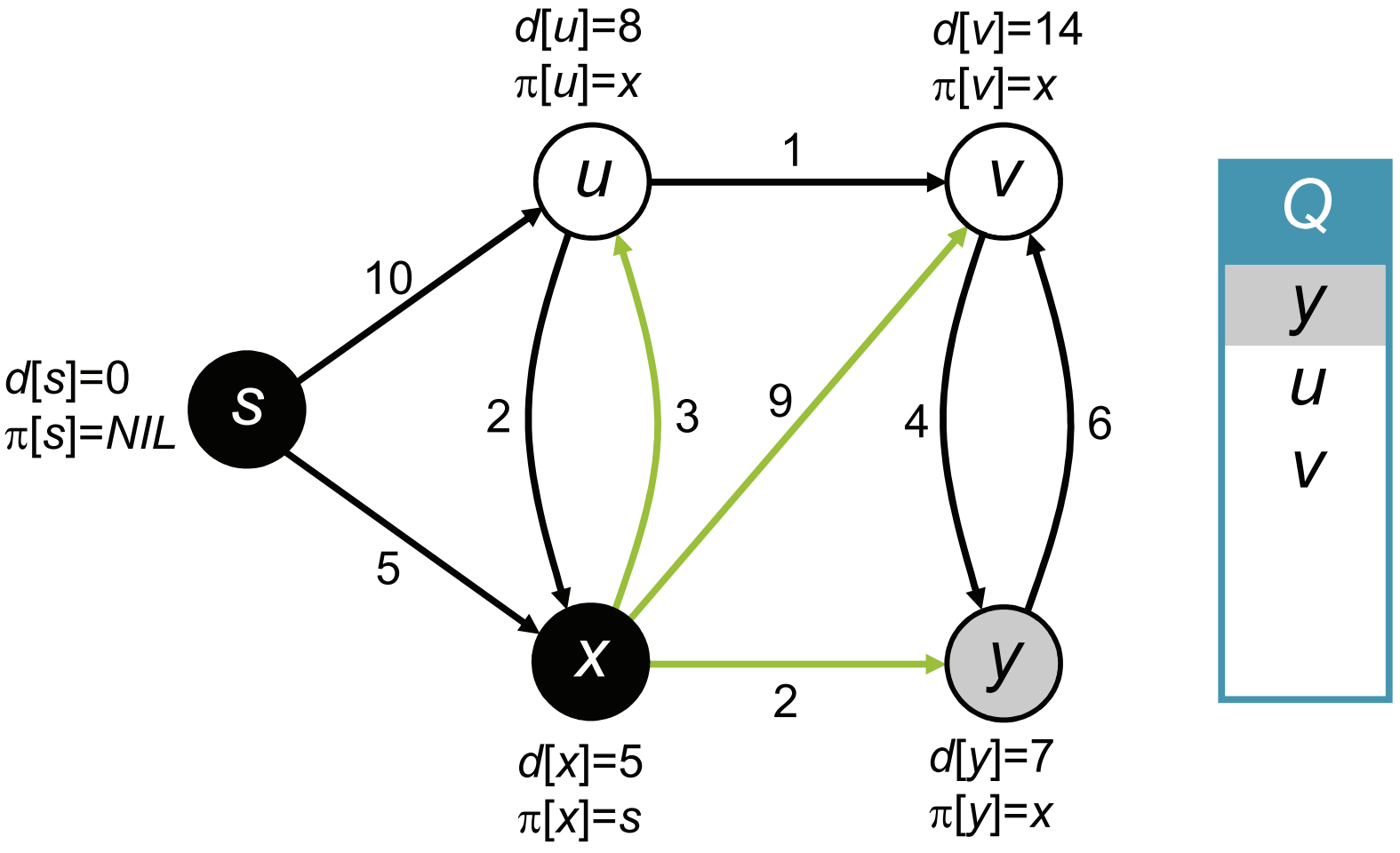


Q
s
u
v
x
y

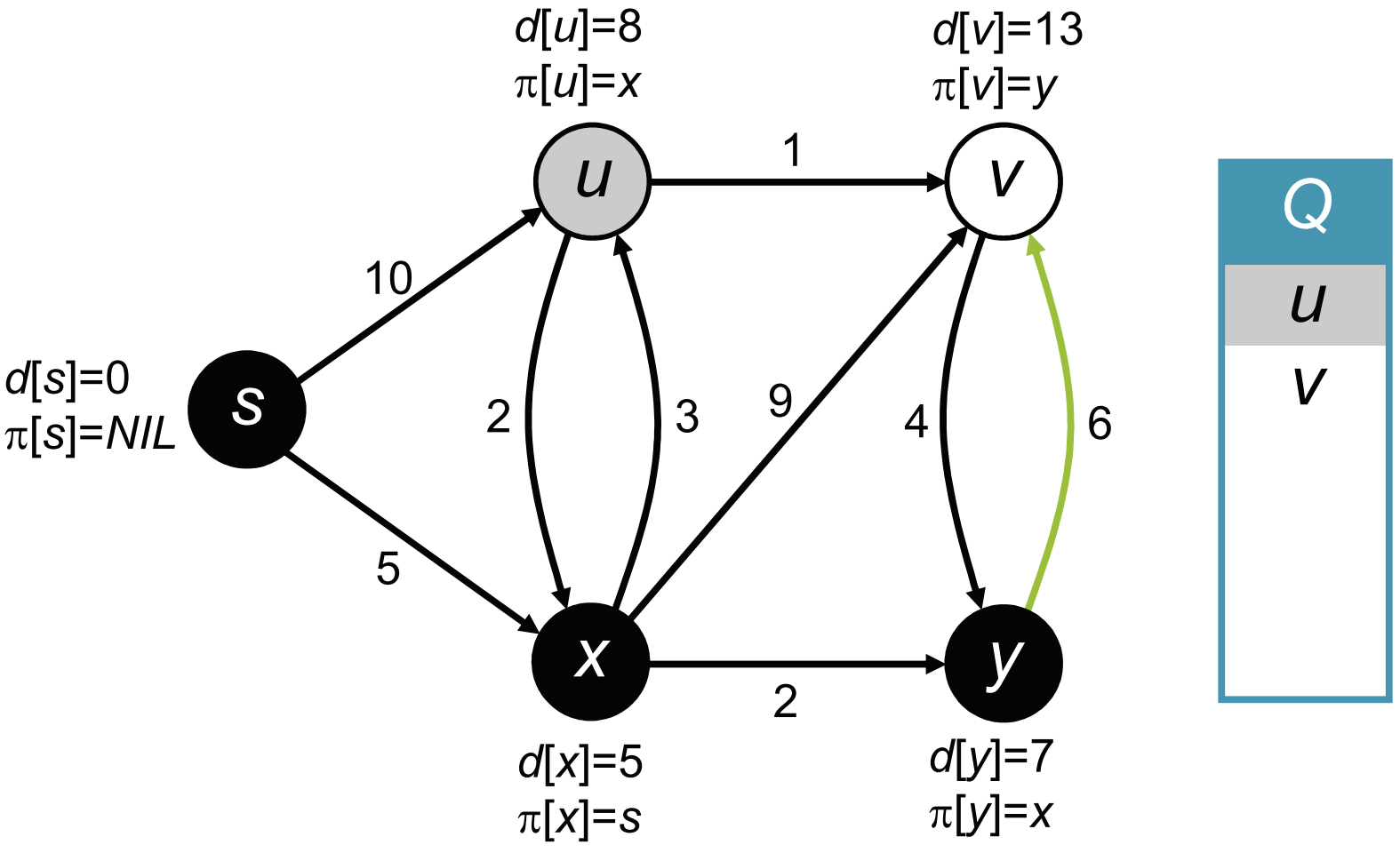
Algoritmo de Dijkstra: Exemplo



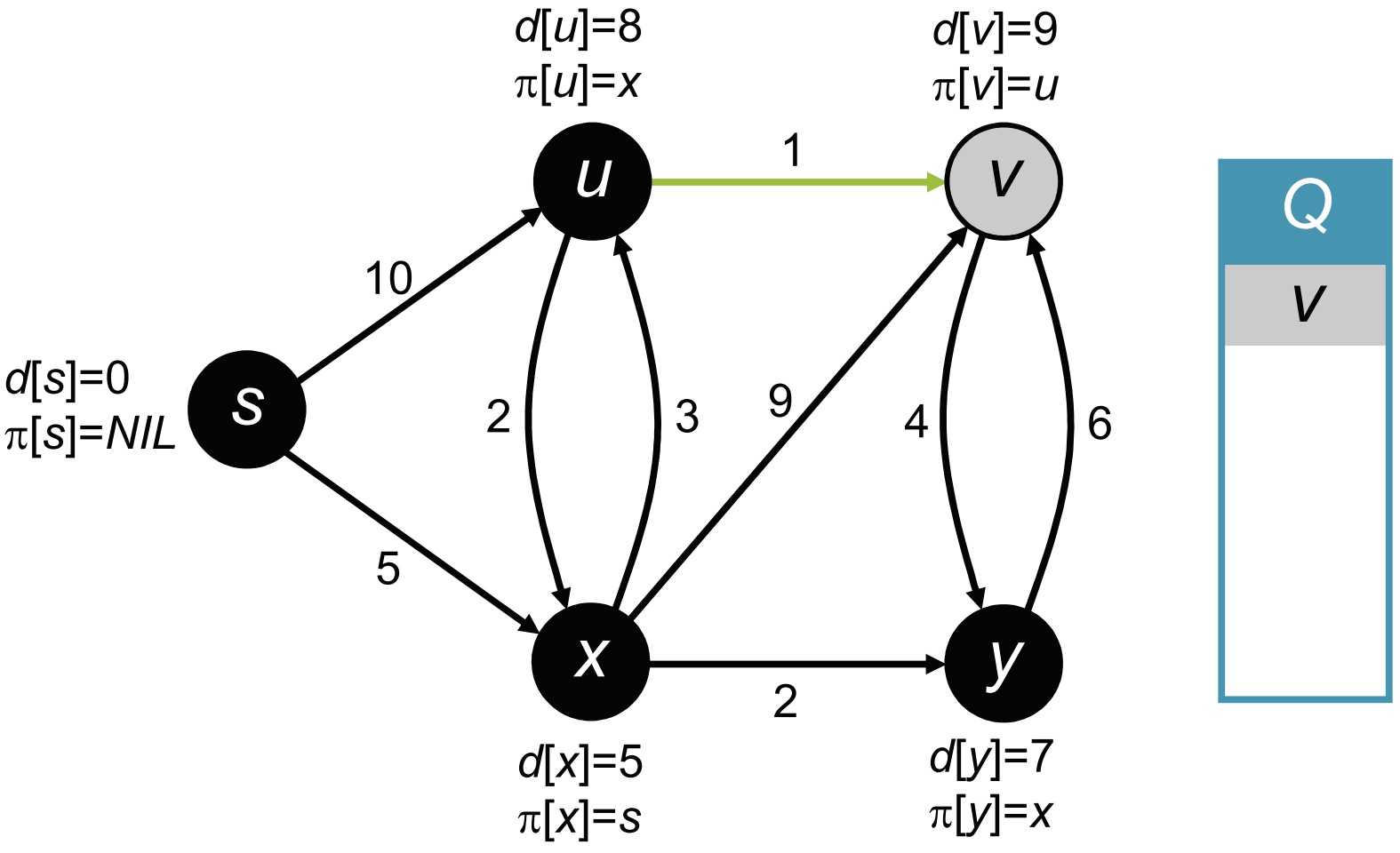
Algoritmo de Dijkstra: Exemplo



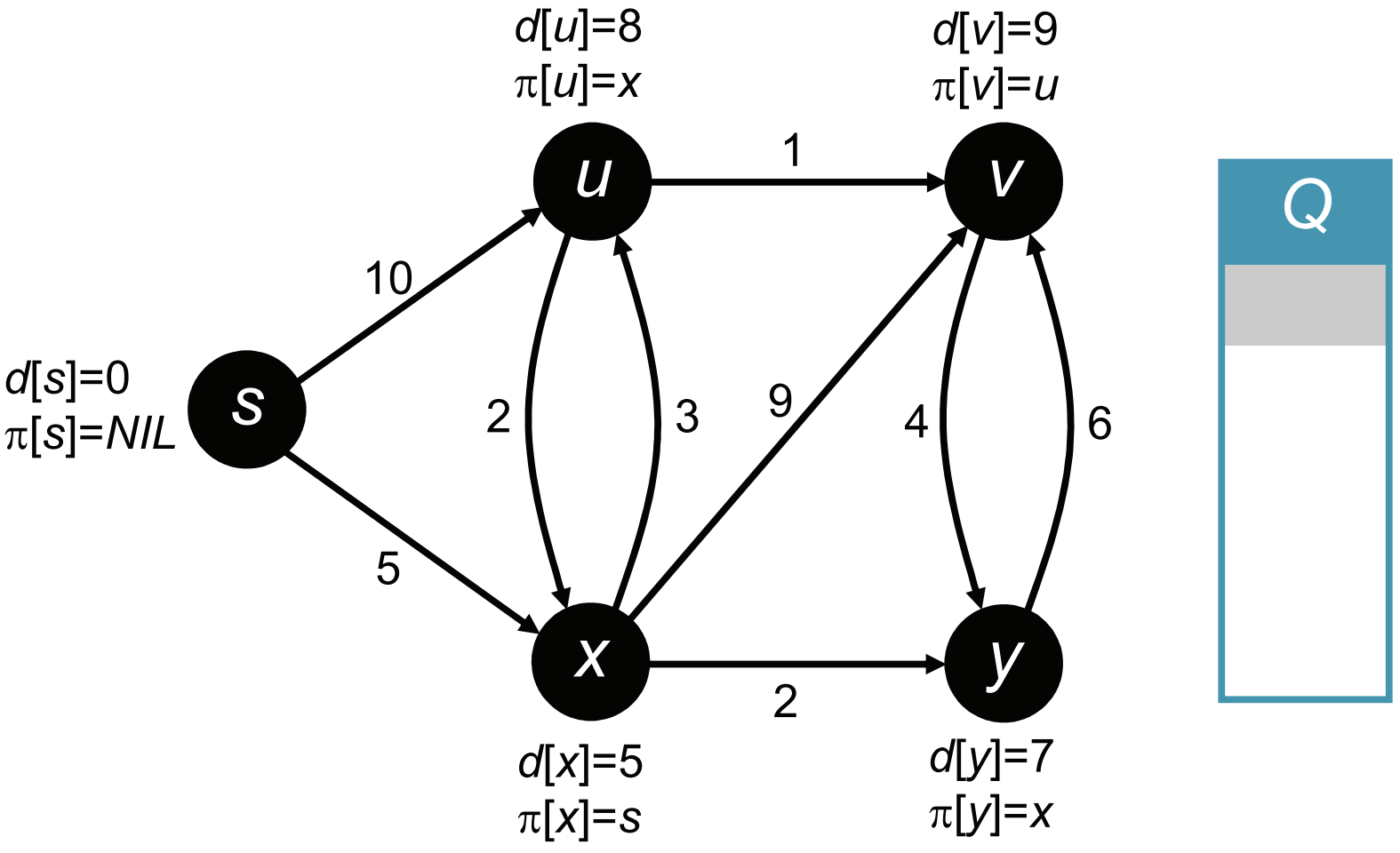
Algoritmo de Dijkstra: Exemplo



Algoritmo de Dijkstra: Exemplo



Algoritmo de Dijkstra: Exemplo



Algoritmo de Dijkstra

Complexidade

- Estrutura muito semelhante ao Algoritmo de Prim para cálculo de MST
- Fila de prioridade baseada em amontoados (heap)
- Quando um vértice é extraído da fila Q, implica actualização de Q
  - Cada vértice é extraído apenas 1 vez $O(V)$
  - Actualização de Q: $O(lg\ V)$
  - Então, $O(V\ lg\ V)$
- Para cada arco (i.e. $O(E)$) operação de relaxação é aplicada apenas 1 vez. Cada operação de relaxação pode implicar uma actualização de Q em $O(lg\ V)$
- Complexidade algoritmo Dijkstra: $O((V+E)\ lg\ V)$

Algoritmo de Dijkstra

Correcção do Algoritmo

Provar invariante do algoritmo: $d[u] = \delta(s,u)$ quando u adicionado ao conjunto S, e que a igualdade é posteriormente mantida

- Prova por contradição. Assume-se que existe um primeiro vértice u tal que $d[u] \neq \delta(s,u)$ quando u é adicionado a S
- Necessariamente temos que $u \neq s$ porque $d[s] = \delta(s,s) = 0$
- $S \neq \emptyset$ porque $s \in S$ quando u é adicionado a S
- Tem que existir caminho mais curto de s para u, dado que caso contrário teriamos $d[u] = \delta(s,u) = \infty$

Algoritmo de Dijkstra

Correcção do Algoritmo (2)

Pressuposto: u é o primeiro vértice tal que $d[u] \neq \delta(s,u)$ quando u é adicionado a S

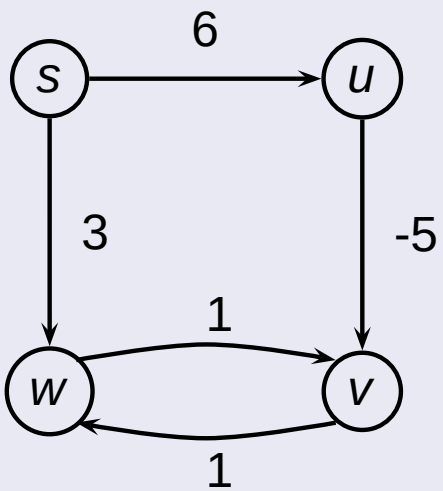
- Seja $p =< s,\ldots,x,y,\ldots,u >$ o caminho mais curto de s para u
- Tem que existir pelo menos um vértice do caminho p que ainda não esteja em S, caso contrário, $d[u] = \delta(s,u)$ devido à relaxação dos arcos que compõem o caminho p
- Seja (x,y) um arco de p tal que $x \in S$ e $y \notin S$
  - Temos que $d[x] = \delta(s,x)$ porque $x \in S$ e u é o primeiro vértice em que isso não ocorre
  - Temos também que $d[y] = \delta(s,y)$ porque o arco (x,y) foi relaxado quando x foi adicionado a S
  - Como y é predecessor de u no caminho mais curto até u, então $\delta(s,y) \leq \delta(s,u)$, porque os pesos dos arcos são não-negativos
  - Mas se u é adicionado a S antes de y, temos que $d[u] \leq d[y]$. Logo, $d[u] \leq \delta(s,y) \leq \delta(s,u)$. O que contradiz o pressuposto de $d[u] \neq \delta(s,u)$.

Algoritmo de Dijkstra

Pesos Negativos no Grafo

Os pesos do grafo têm que ser não-negativos para garantir a correcção do algoritmo

Exemplo



Execução do Algoritmo Dijkstra:

- Analisar w com $d[w] = 3$
- Analisar v com $d[v] = 4$
- Analisar u com $d[u] = 6$
- No final temos que $d[w] = 3 \neq \delta(s, w) = 2$

Algoritmo de Bellman-Ford

Organização do Algoritmo

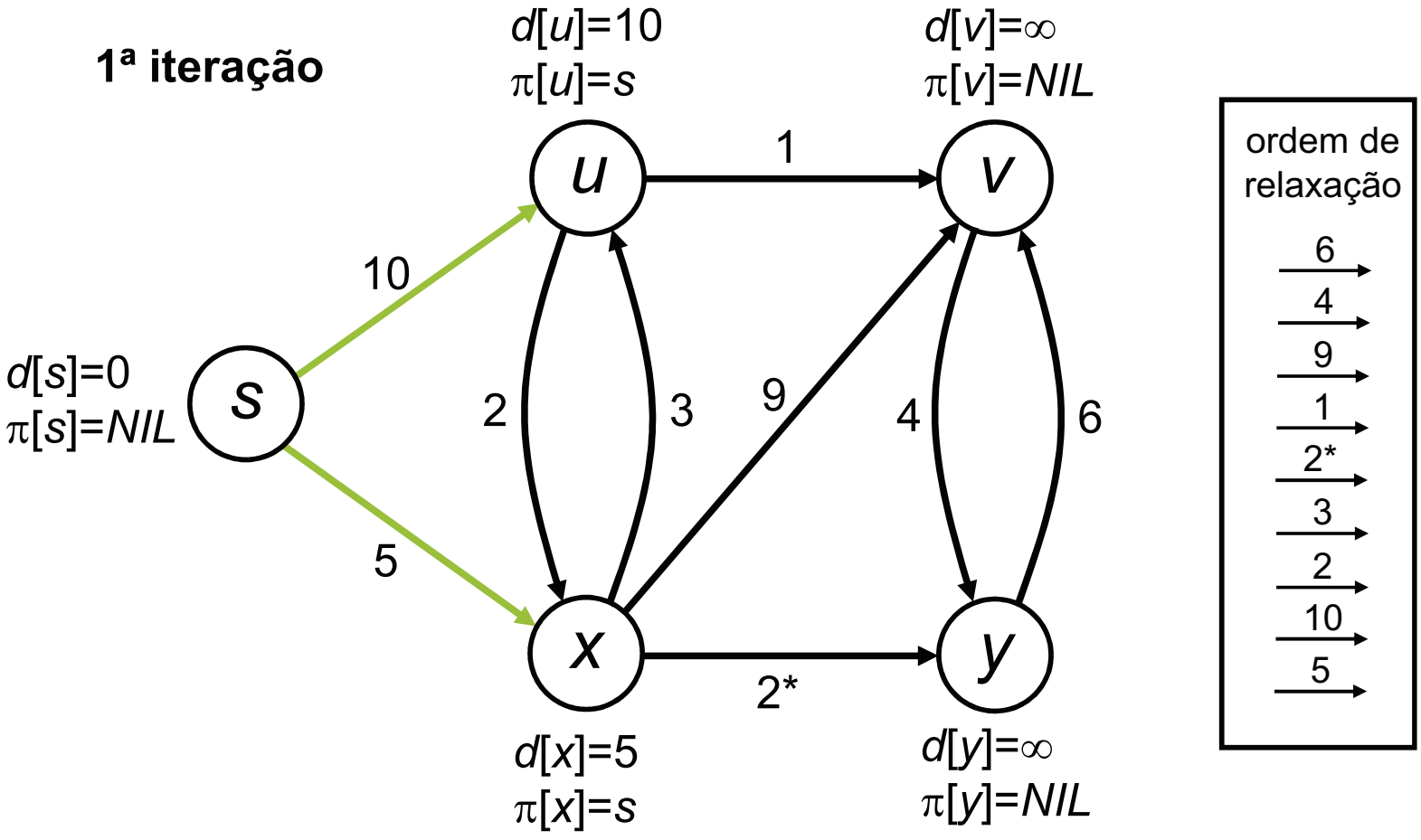
- Permite pesos negativos e identifica existência de ciclos negativos
- Baseado em sequência de passos de relaxação
- Apenas requer manutenção da estimativa associada a cada vértice

Algoritmo de Bellman-Ford

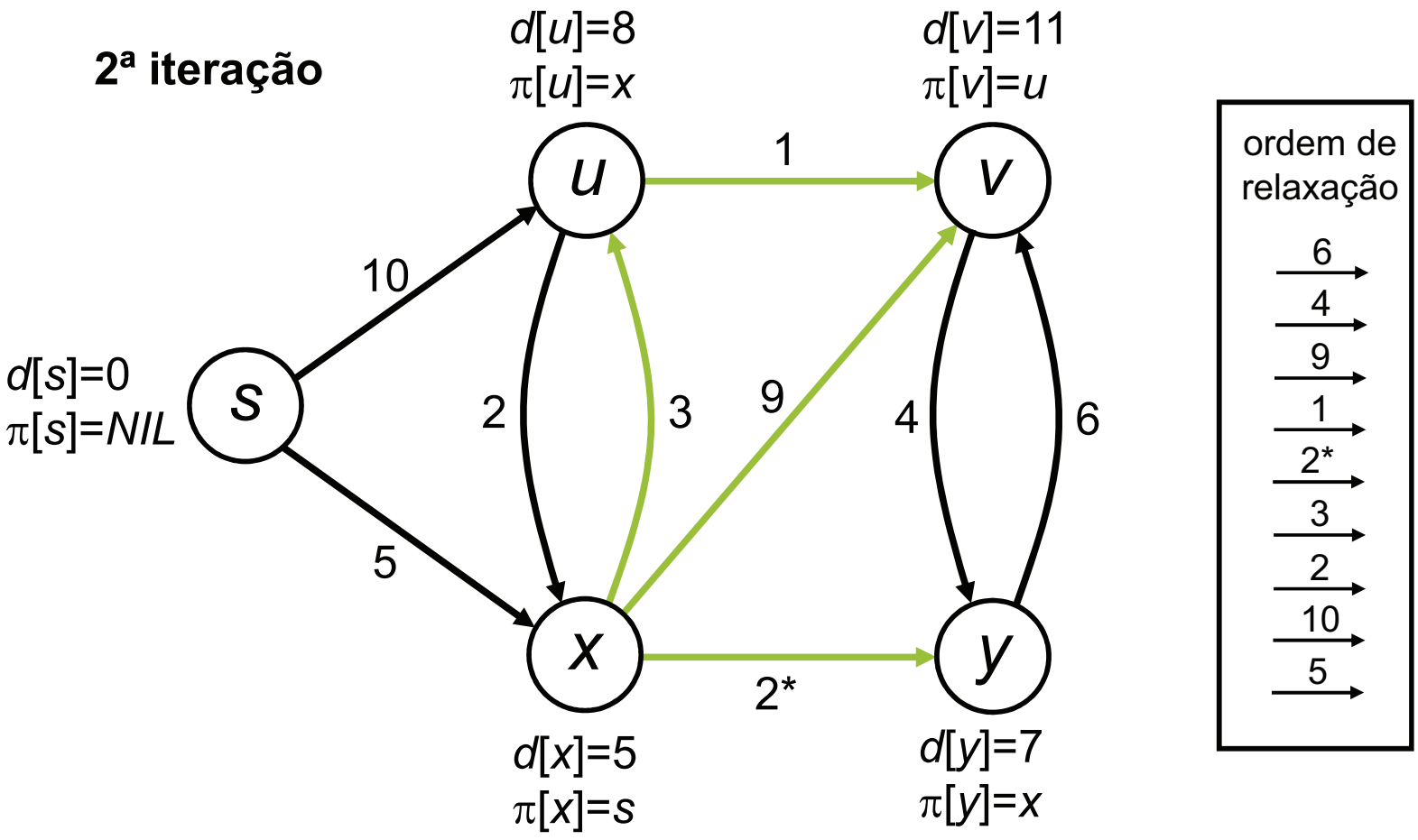
Bellman-Ford(G, w, s)

```
1  Initialize-Single-Source(G, s)
2  for i ← 1 to |V[G]| − 1
3        do for each (u, v) ∈ E[G]
4              do Relax(u, v, w)
5  for each (u, v) ∈ E[G]
6        do if d[v] > d[u] + w(u, v)
7              then return FALSE          ▷ Ciclo negativo
8  return TRUE                           ▷ Sem ciclos negativos
```

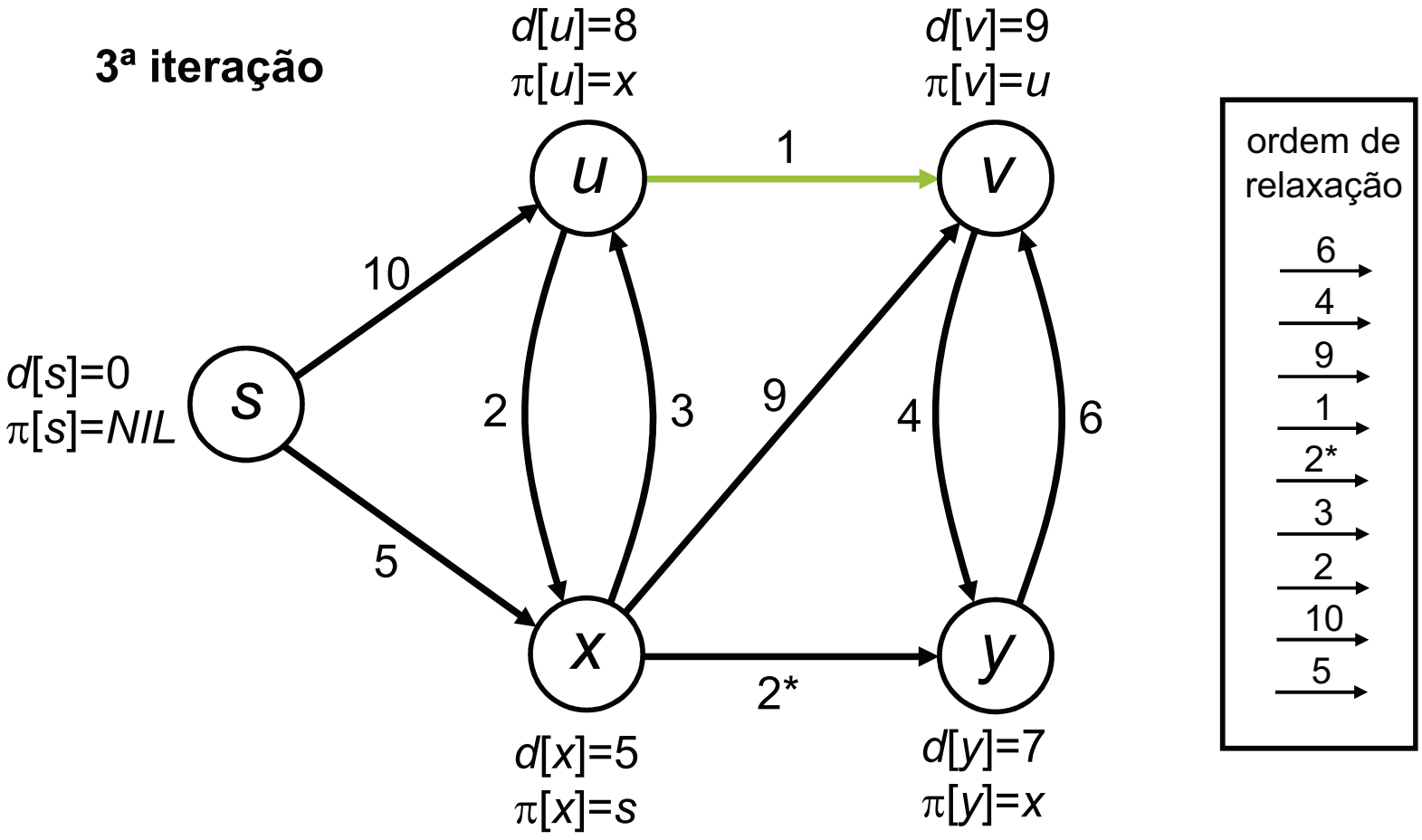
Algoritmo de Bellman-Ford: Exemplo



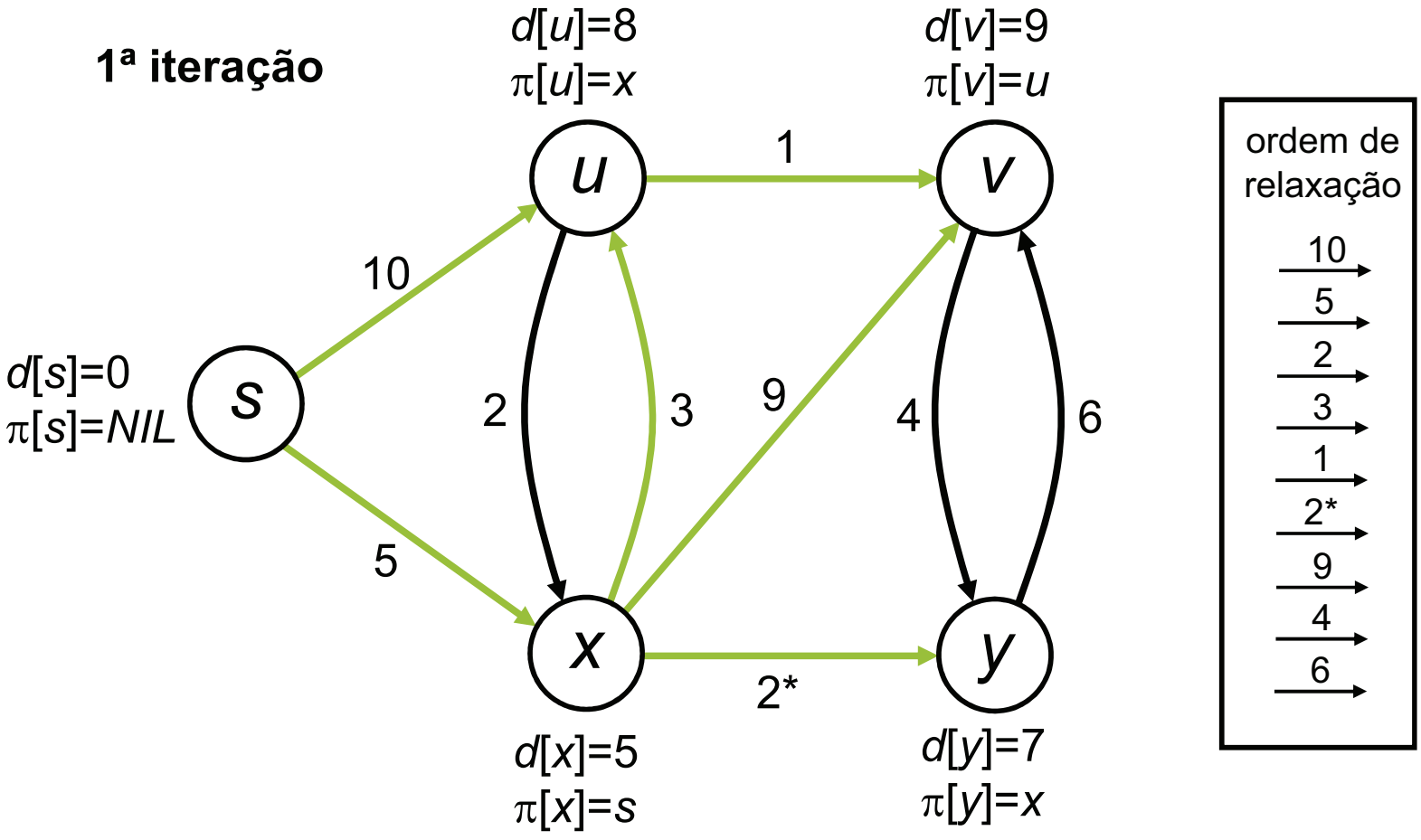
Algoritmo de Bellman-Ford: Exemplo



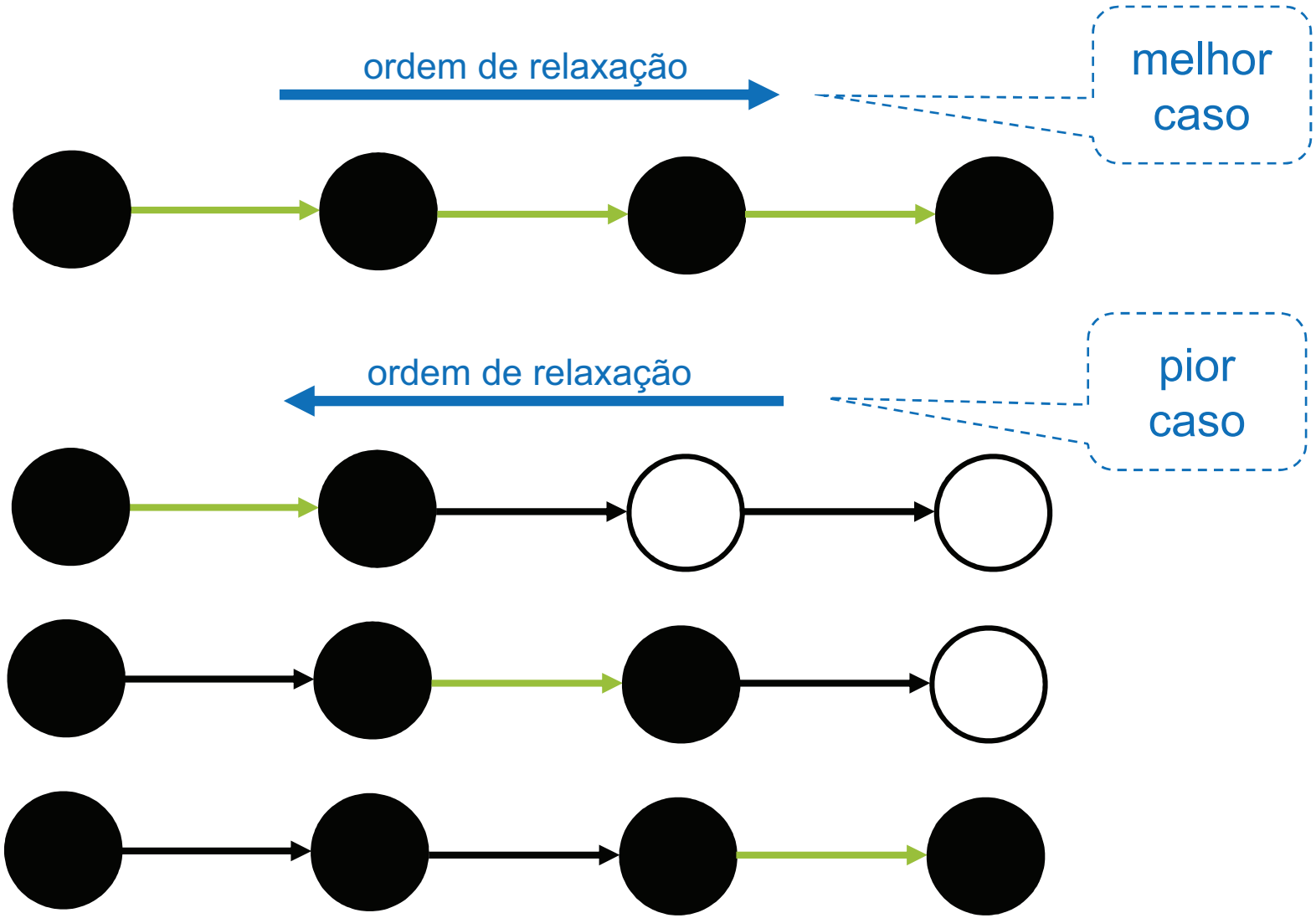
Algoritmo de Bellman-Ford: Exemplo



Algoritmo de Bellman-Ford: Outro exemplo



Algoritmo de Bellman-Ford: Outro exemplo



Algoritmo de Bellman-Ford

Complexidade

- Inicialização: $\Theta(V)$
- A complexidade do ciclo `for` é $O(VE)$ devido aos dois ciclos, em V e em E.
  - Em cada iteração todos os arcos são relaxados
- Complexidade algoritmo Bellman-Ford: $O(VE)$

Algoritmo de Bellman-Ford

Correcção

Se $G = (V, E)$ não contém ciclos negativos, então após a aplicação do algoritmo de Bellman-Ford, $d[v] = \delta(s, v)$ para todos os vértices atingíveis a partir de s

- Seja v atingível a partir de s, e seja $p =< v_0, v_1, \ldots, v_k >$ um caminho mais curto entre s e v, com $v_0 = s$ e $v_k = v$
- p é simples, pelo que $k \leq |V| - 1$
- Prova: provar por indução que $d[v_i] = \delta(s, v_i)$ para $i = 0, 1, \ldots, k$, após iteração i sobre os arcos de G, e que valor não é alterado posteriormente
  - Base: $d[v_0] = \delta(s, v_0) = 0$ após inicialização (e não se altera)
  - Passo indutivo: assumir $d[v_{i-1}] = \delta(s, v_{i-1})$ após iteração (i-1)
  - Arco (v_{i-1}, v_i) relaxado na iteração i, pelo que $d[v_i] = \delta(s, v_i)$ após iteração i (e não se altera)

Algoritmo de Bellman-Ford

Correcção (2)

Se $G = (V, E)$ não contém ciclos negativos, o algoritmo de Bellman-Ford retorna TRUE. Se o grafo contém algum ciclo negativo atingível a partir de s, o algoritmo retorna FALSE

- Se não existem ciclos negativos, resultado anterior assegura que para qualquer arco $(u, v) \in E$, $d[v] \leq d[u] + w(u, v)$, pelo que teste do algoritmo falha para todo o (u, v) e o valor retornado é TRUE
- Caso contrário, na presença de pelo menos um ciclo negativo atingível a partir de s, $c =< v_0, v_1, \ldots, v_k >$, onde $v_0 = v_k$, temos que $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$

Algoritmo de Bellman-Ford

Correcção (3)

Se $G = (V, E)$ não contém ciclos negativos, o algoritmo de Bellman-Ford retorna TRUE. Se o grafo contém algum ciclo negativo atingível a partir de s, o algoritmo retorna FALSE

- Prova por contradição. Admitir que algoritmo retorna TRUE na presença de ciclo negativo. Pelo que $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$, para $i = 1, \ldots, k$
- Somando as desigualdades ao longo do ciclo temos que:

$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- Note-se que $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$ por ser um ciclo
- Temos então que $\sum_{i=1}^{k} w(v_{i-1}, v_i) > 0$, o que contradiz a existência de um ciclo negativo. Logo, o algoritmo retorna FALSE

Caminhos mais curtos em DAGs

DAG-Shortest-Path(G, w, s)

1 Ordenação topológica dos vértices de G
2 Initialize-Single-Source(G, s)
3 **for** each $u \in V[G]$ por ordem topológica
4 **do for** each $v \in Adj[u]$
5 **do** Relax(u, v, w)

Complexidade:

Caminhos mais curtos em DAGs

DAG-Shortest-Path(G, w, s)

```
1  Ordenação topológica dos vértices de G
2  Initialize-Single-Source(G, s)
3  for each u ∈ V[G] por ordem topológica
4        do for each v ∈ Adj[u]
5              do Relax(u, v, w)
```

Complexidade: $O(V + E)$

Caminhos mais curtos em DAGs

Correcção

Dado $G = (V, E)$, dirigido, acíclico, como resultado do algoritmo, temos que $d[v] = \delta(s, v)$ para todo o $v \in V$

- Seja v atingível a partir de s, e seja $p =< v_0, v_1, \ldots, v_k >$ um caminho mais curto entre s e v, com $v_0 = s$ e $v_k = v$
- Ordenação topológica implica que analisados por ordem (v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)
- Prova por indução: $d[v_i] = \delta(s, v_i)$ sempre que cada vértice v_i é terminado
  - Base: Estimativa de s não alterada após inicialização; $d[s] = d[v_0] = \delta(s, v_0) = 0$
  - Indução: $d[v_{i-1}] = \delta(s, v_{i-1})$ após terminar análise de v_{i-1}
  - Relaxação do arco (v_{i-1}, v_i) causa $d[v_i] = \delta(s, v_i)$, pelo que $d[v_i] = \delta(s, v_i)$ após terminar análise de v_i

Caminhos mais curtos em DAGs

Resumo

Algoritmo Dijkstra

- Só permite pesos não negativos
- Complexidade: $O((V+E)\ lg\ V)$

Algoritmo Bellman-Ford

- Permite pesos negativos e identifica ciclos negativos
- Complexidade: $O(VE)$

Caminhos mais curtos em DAGs

- Grafos aciclicos. Podemos fazer ordenação topológica dos vértices
- Complexidade: $O(V+E)$