MC558 — Análise de Algoritmos II

Cid C. de Souza   Cândida N. da Silva   Orlando Lee

26 de abril de 2023

Antes de mais nada...

- Uma versão anterior deste conjunto de slides foi preparada por Cid Carvalho de Souza e Cândida Nunes da Silva para uma instância anterior desta disciplina.
- O que vocês tem em mãos é uma versão modificada preparada para atender a meus gostos.
- Nunca é demais enfatizar que o material é apenas um guia e não deve ser usado como única fonte de estudo. Para isso consultem a bibliografia (em especial o CLR ou CLRS).

Orlando Lee

Agradecimentos (Cid e Cândida)

- Várias pessoas contribuíram direta ou indiretamente com a preparação deste material.
- Algumas destas pessoas cederam gentilmente seus arquivos digitais enquanto outras cederam gentilmente o seu tempo fazendo correções e dando sugestões.
- Uma lista destes "colaboradores" (em ordem alfabética) é dada abaixo:
  - Célia Picinin de Mello
  - José Coelho de Pina
  - Orlando Lee
  - Paulo Feofiloff
  - Pedro Rezende
  - Ricardo Dahab
  - Zanoni Dias

Terminologia do CLRS

Um caminho em um grafo $G = (V, E)$ é uma sequência:

$$P := (v_0, v_1, \dots, v_k)$$

em que $v_i \in V$ para $i = 0, 1, 2, \dots, k$ e $(v_{i-1}, v_i) \in E$ para $i = 1, 2, \dots, k$.

Um caminho é simples se todos os vértices são distintos.

Observação: em muitos textos, o usual é chamar de passeio o que CLRS chama de caminho e chamar de caminho o que CLRS chama de caminho simples.

Terminologia do CLRS

Um ciclo em um grafo $G = (V, E)$ é uma sequência:

$$C := (v_0, v_1, \ldots, v_k)$$

em que $v_0 = v_k$, $v_i \in V$ para $i = 0, 1, 2, \ldots, k$ e $(v_{i-1}, v_i) \in E$ para $i = 1, 2, \ldots, k$.

Um ciclo é simples se $v_1, v_2, \ldots, v_k$ são todos distintos.

Observação: em muitos textos, o usual é chamar de passeio fechado o que CLRS chama de ciclo e chamar de ciclo o que CLRS chama de ciclo simples.

Problema do(s) Caminho(s) Mínimo(s)

Seja G um grafo orientado e suponha que para cada aresta (u, v) associamos um peso (custo) $\omega(u, v)$. Usaremos a notação (G, ω).

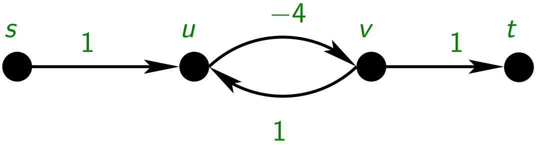
- Problema do Caminho Mínimo entre Dois Vértices:
  Dados dois vértices s e t em (G, ω), encontrar um caminho (de peso) mínimo de s a t.

- Aparentemente, este problema não é mais fácil do que o

  Problema dos Caminhos Mínimos com Mesma Origem:
  Dados (G, ω) e $s \in V[G]$, encontrar para cada vértice v de G, um caminho mínimo de s a v.

Ciclos negativos

- Um ciclo C é negativo se $\omega(C) := \sum_{e \in E(C)} \omega(e) < 0$.
- Se (G, ω) contém ciclos negativos podem existir caminhos de s a t de peso arbitrariamente pequeno.



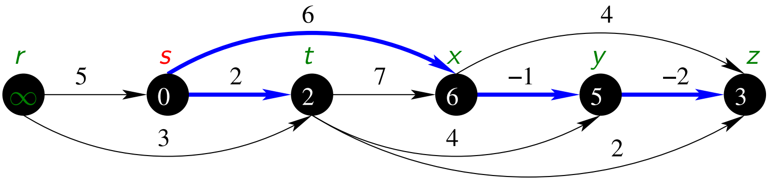
- Se existe um caminho mínimo de s a t, então existe um caminho mínimo de s a t que é simples.
- Os algoritmos que veremos trabalham com instâncias **sem** ciclos negativos.

Distância

Seja (G, ω) um grafo orientado ponderado e sejam $s, t \in V[G]$. A distância de s a t é o peso de um caminho de peso mínimo de s a t, se ele existir.
Se não existe caminho de s a t em G, então $\mathrm{dist}(s, t) = \infty$.
Se existem caminhos de s a t de peso arbitrariamente pequeno, então $\mathrm{dist}(s, t) = -\infty$.

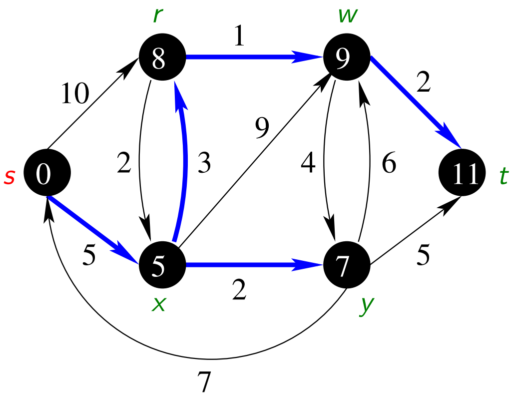
O Problema dos Caminhos Mínimos com Mesma Origem consiste em determinar $\mathrm{dist}(s, v)$ para todo $v \in V$ (e achar os caminhos mínimos também).

Exemplo: grafo orientado acíclico



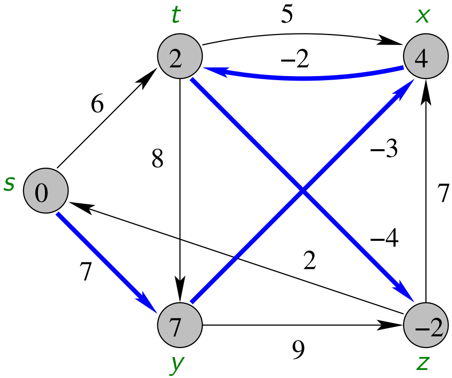
v	s	r	t	x	y	z
$\mathrm{dist}(s,v)$	0	∞	2	6	5	3

Exemplo: grafo orientado sem arestas negativas



v	s	r	x	y	w	t
$\mathrm{dist}(s, v)$	0	8	5	7	9	11

Exemplo: grafo orientado sem ciclos negativos



v	s	t	x	y	z
$\mathrm{dist}(s, v)$	0	2	4	7	−2

Ideias comuns a todos os algoritmos

- Usamos uma idéia similar à usada em Busca em Largura nos algoritmos de caminhos mínimos que veremos.
- Para cada vértice $v \in V[G]$ associamos um predecessor $\pi[v]$.
- Ao final do algoritmo obtemos uma Árvore de Caminhos Mínimos com raiz s.
- Um caminho de s a v nesta árvore é um caminho mínimo de s a v em (G, ω).

Subestrutura ótima de caminhos mínimos

Lema 24.1, CLRS Seja (G, ω) um grafo orientado e seja

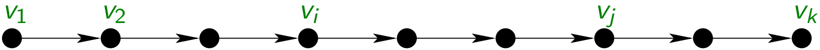
$$P = (v_1, v_2, \ldots, v_k)$$

um caminho mínimo de v_1 a v_k.

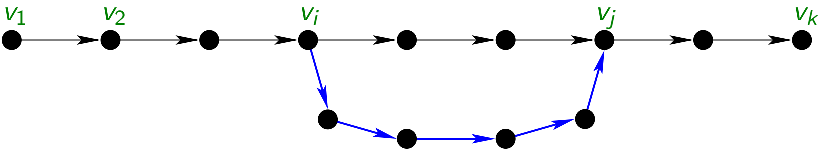
Então para quaisquer i, j com $1 \leq i \leq j \leq k$

$$P_{ij} = (v_i, v_{i+1}, \ldots, v_j)$$

é um caminho mínimo de v_i a v_j.



Prova. Suponha por contradição que $P_{ij} = (v_i, v_{i+1}, \ldots, v_j)$ não seja um caminho mínimo de v_i a v_j. Assim, existe um caminho Q de v_i a v_j tal que $\omega(Q) < \omega(P_{ij})$.



Mas então o caminho $(v_0, \ldots, v_i)Q(v_j, \ldots, v_k)$ tem peso menor que o peso de P, uma contradição. ■

Inicialização

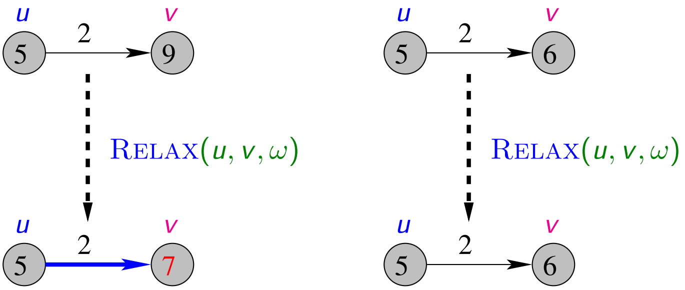
INITIALIZE-SINGLE-SOURCE(G, s)

para cada vértice $v \in V[G]$ **faça**
    $d[v] \leftarrow \infty$
    $\pi[v] \leftarrow$ NIL
$d[s] \leftarrow 0$

- O valor $d[v]$ é uma estimativa superior de $\mathrm{dist}(s, v)$, o peso de um caminho mínimo de s a v.
- Se $d[v] < \infty$, então o algoritmo encontrou até aquele momento um caminho de s a v com peso $d[v]$.
- O caminho de s a v pode ser recuperado por meio dos predecessores $\pi[\,]$, se não houver ciclos negativos.

Relaxação

Tenta melhorar a estimativa $d[v]$ examinando (u, v).



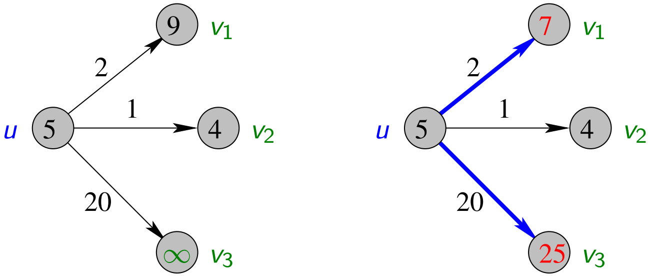
$\text{Relax}(u, v, \omega)$

1 **se** $d[v] > d[u] + \omega(u, v)$ **então**

2 $\quad d[v] \leftarrow d[u] + \omega(u, v)$

3 $\quad \pi[v] \leftarrow u$

Em cada iteração o algoritmo seleciona um vértice u e para cada vizinho v de u aplica $\textsc{Relax}(u, v, \omega)$.



$\textsc{Relax}(u, v, \omega)$

    se $d[v] > d[u] + \omega(u, v)$ **então**
        $d[v] \leftarrow d[u] + \omega(u, v)$
        $\pi[v] \leftarrow u$

Veremos três algoritmos baseados em relaxação para tipos de instâncias diferentes de Problemas de Caminhos Mínimos.

- G é **acíclico**: aplicação de ordenação topológica
- (G, ω) **não** tem arestas de **peso negativo**: algoritmo de Dijkstra
- (G, ω) tem arestas de peso negativo, mas **não** contém **ciclos negativos**: algoritmo de Bellman-Ford.

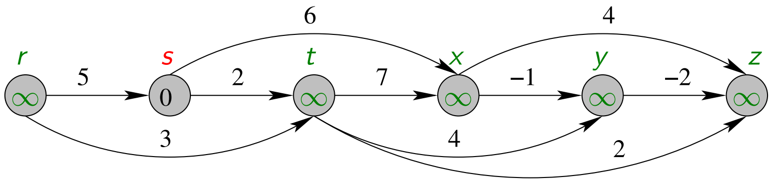
Seja G_π com conjunto de vértices $V_\pi = \{s\} \cup \{v \in V : \pi[v] \neq \text{NIL}\}$ e conjunto de arestas $E_\pi = \{(\pi[v], v) : v \in V, \pi[v] \neq \text{NIL}\}$.

Entrada: grafo orientado acíclico (G, ω) e origem $s \in V[G]$.

Saída: um vetor $d[\ ]$ tal que $d[v] = \text{dist}(s, v)$ para $v \in V[G]$ e um vetor $\pi[\ ]$ tal que G_π é uma Árvore de Caminhos Mínimos com raiz s de (G, ω).

```
DAG-SHORTEST-PATHS(G, ω, s)
1   Ordene topologicamente os vértices de G
2   INITIALIZE-SINGLE-SOURCE(G, s)
3   para cada vértice u na ordem topológica faça
4       para cada v ∈ Adj[u] faça
5           RELAX(u, v, ω)
6   devolva d, π
```

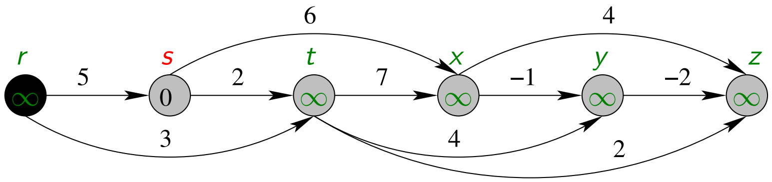
Exemplo



$\text{Relax}(u, v, \omega)$
  se $d[v] > d[u] + \omega(u, v)$
    então
    $d[v] \leftarrow d[u] + \omega(u, v)$
    $\pi[v] \leftarrow u$

para cada vértice u na o.t. **faça**
  para cada $v \in \text{Adj}[u]$ **faça**
    $\text{Relax}(u, v, \omega)$

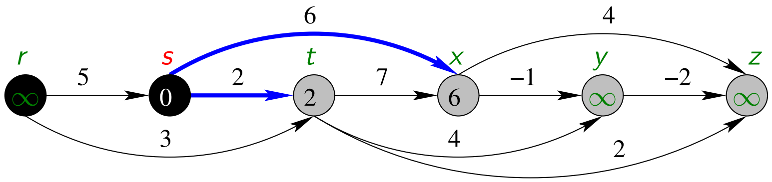
Exemplo



$\textsc{Relax}(u, v, \omega)$
  se $d[v] > d[u] + \omega(u, v)$
    então
    $d[v] \leftarrow d[u] + \omega(u, v)$
    $\pi[v] \leftarrow u$

para cada vértice u na o.t. **faça**
  para cada $v \in \mathrm{Adj}[u]$ **faça**
    $\textsc{Relax}(u, v, \omega)$

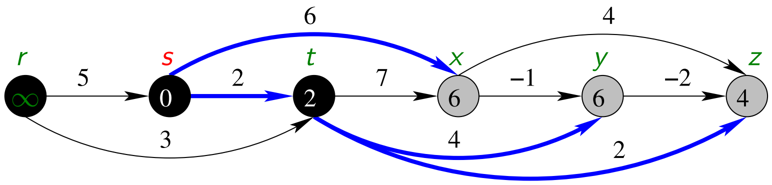
Exemplo



$\text{Relax}(u, v, \omega)$
se $d[v] > d[u] + \omega(u, v)$
então
$d[v] \leftarrow d[u] + \omega(u, v)$
$\pi[v] \leftarrow u$

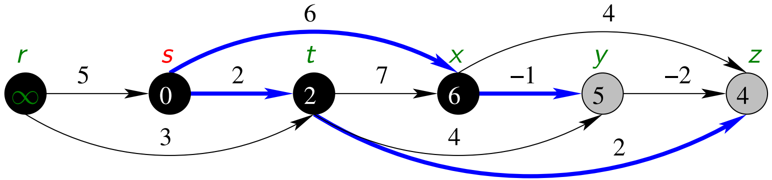
para cada vértice u na o.t. **faça**
para cada $v \in \text{Adj}[u]$ **faça**
$\text{Relax}(u, v, \omega)$

Exemplo



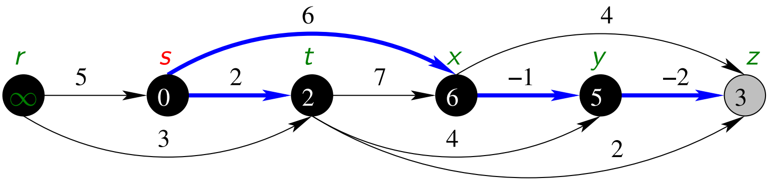
$\textsc{Relax}(u, v, \omega)$
se $d[v] > d[u] + \omega(u, v)$
então
$d[v] \leftarrow d[u] + \omega(u, v)$
$\pi[v] \leftarrow u$

para cada vértice u na o.t. **faça**
para cada $v \in \mathrm{Adj}[u]$ **faça**
$\textsc{Relax}(u, v, \omega)$



$\textsc{Relax}(u, v, \omega)$
se $d[v] > d[u] + \omega(u, v)$
então
$d[v] \leftarrow d[u] + \omega(u, v)$
$\pi[v] \leftarrow u$

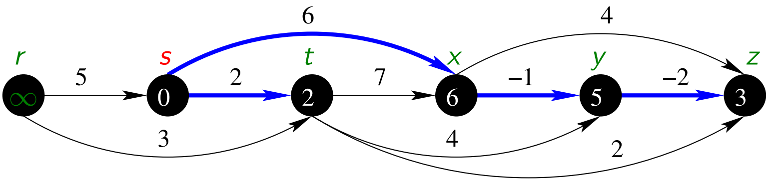
para cada vértice u na o.t. **faça**
para cada $v \in \mathrm{Adj}[u]$ **faça**
$\textsc{Relax}(u, v, \omega)$



$\textsc{Relax}(u, v, \omega)$
se $d[v] > d[u] + \omega(u, v)$
então
$d[v] \leftarrow d[u] + \omega(u, v)$
$\pi[v] \leftarrow u$

para cada vértice u na o.t. **faça**
para cada $v \in \mathrm{Adj}[u]$ **faça**
$\textsc{Relax}(u, v, \omega)$

Exemplo



$\textsc{Relax}(u, v, \omega)$
se $d[v] > d[u] + \omega(u, v)$
então
$d[v] \leftarrow d[u] + \omega(u, v)$
$\pi[v] \leftarrow u$

para cada vértice u na o.t. **faça**
para cada $v \in \mathrm{Adj}[u]$ **faça**
$\textsc{Relax}(u, v, \omega)$

Complexidade DAG-SHORTEST-PATHS

DAG-SHORTEST-PATHS(G, ω, s)
1 Ordene topologicamente os vértices de G
2 INITIALIZE-SINGLE-SOURCE(G, s)
3 **para cada** vértice u na ordem topológica **faça**
4 **para cada** $v \in \mathrm{Adj}[u]$ **faça**
5 RELAX(u, v, ω)
6 **devolva** d, π

Linha(s)	Tempo total
1	$O(V+E)$
2	$O(V)$
3-5	$O(V+E)$

Complexidade de DAG-SHORTEST-PATHS: $O(V+E)$

A corretude de DAG-SHORTEST-PATHS pode ser demonstrada de várias formas.

Vamos mostrar alguns lemas/observações que serão úteis na análise de corretude deste e dos outros algoritmos.

Desigualdade triangular

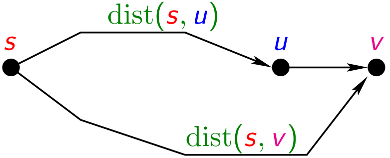
Lema 24.10, CLRS Para toda aresta (u, v) temos que

$$\text{dist}(s, v) \leq \text{dist}(s, u) + \omega(u, v).$$

Prova. Suponha que existe um caminho mínimo P de s a u. Logo $P + (u, v)$ é um caminho de s a v de peso

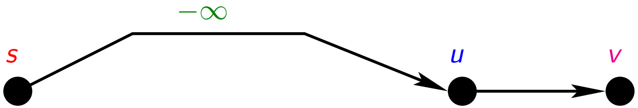
$$\omega(P) + \omega(u, v) = \text{dist}(s, u) + \omega(u, v).$$

Portanto, $\text{dist}(s, v) \leq \text{dist}(s, u) + \omega(u, v)$.



Se não existe caminho de s a u em G (i.e., $\mathrm{dist}(s,u) = \infty$), então claramente $\mathrm{dist}(s,v) \leq \mathrm{dist}(s,u) + \omega(u,v)$.

Finalmente, se existem caminhos de peso arbitrariamente pequeno de s a u (i.e., $\mathrm{dist}(s,u) = -\infty$), então $\mathrm{dist}(s,v) = -\infty$. ∎



Propriedades de um algoritmo baseado em relaxação

As propriedades que veremos valem para qualquer "algoritmo" que satisfaz as restrições abaixo.

- A inicialização é feita por Initialize-Single-Source(G, s).
- Um valor $d[v]$ (e $\pi[v]$) só pode ser modificado através de uma chamada de Relax(u, v, ω) para alguma aresta (u, v).

Observação: o "algoritmo" pode executar outras instruções, mas estamos interessados apenas nos valores de $d[\ ]$ e $\pi[\ ]$. Assim, analisamos apenas o que acontece após chamadas a Initialize-Single-Source ou Relax.

A seguir apresentamos propriedades que valem durante a execução do "algoritmo".

Propriedades de um algoritmo baseado em relaxação

```
INITIALIZE-SINGLE-SOURCE(G, s)
    para cada vértice v ∈ V[G] faça
        d[v] ← ∞
        π[v] ← NIL
    d[s] ← 0
```

```
RELAX(u, v, ω)
    se d[v] > d[u] + ω(u, v) então
        d[v] ← d[u] + ω(u, v)
        π[v] ← u
```

Lema 24.11, CLRS Em qualquer iteração, $d[v] \geq \mathrm{dist}(s, v)$ para $v \in V[G]$ e $d[v]$ nunca aumenta.
Além disso, se $d[v]$ torna-se igual a $\mathrm{dist}(s, v)$, então seu valor nunca mais muda.

Prova. Mostraremos que $d[v] \geq \mathrm{dist}(s, v)$ por indução no número de relaxações (chamadas a $\textsc{Relax}$).

Base: após a inicialização, claramente $d[v] \geq \mathrm{dist}(s, v)$ pois $d[v] = \infty$ para $v \in V - \{s\}$ e $d[s] = \infty \geq \mathrm{dist}(s, s) \in \{0, -\infty\}$.

Propriedades de um algoritmo baseado em relaxação

Hipótese de indução: suponha que $d[v] \geq \text{dist}(s, v)$ para todo $v \in V$ após um certo número de chamadas a Relax.

Vejamos o que acontece após uma próxima chamada a $\text{Relax}(u, v, \omega)$. Note que apenas o valor de $d[v]$ pode mudar.

Se $d[v]$ não é modificado, então a propriedade se mantém. Se $d[v]$ é modificado, então

$$
\begin{aligned}
d[v] &= d[u] + \omega(u, v) \\
&\geq \text{dist}(s, u) + \omega(u, v) \quad \text{(HI)} \\
&\geq \text{dist}(s, v) \quad \text{(desigualdade triangular)}
\end{aligned}
$$

e o resultado segue.

Finalmente, suponha que $d[v] = \text{dist}(s, v)$. Então o valor $d[v]$ não pode mais diminuir pois mostramos que $d[v] \geq \text{dist}(s, v)$ durante a execução do algoritmo. ■

Corolário 24.12, CLRS Se não existe caminho de s a v, então $d[v] = \mathrm{dist}(s, v) = \infty$ em qualquer iteração.

Prova. Pelo Lema 24.11, em qualquer iteração $d[v] \geq \mathrm{dist}(s, v) = \infty$. Como $d[v] = \infty$ após a inicialização, o resultado segue. ■

$\textsc{Relax}(u, v, \omega)$

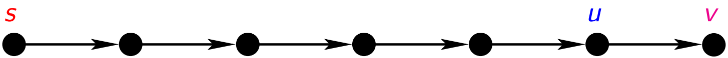
se $d[v] > d[u] + \omega(u, v)$ **então**
    $d[v] \leftarrow d[u] + \omega(u, v)$
    $\pi[v] \leftarrow u$

Lema 24.13 Imediatamente após uma chamada $\textsc{Relax}(u, v)$, temos que $d[v] \leq d[u] + \omega(u, v)$. ■

Propriedades de um algoritmo baseado em relaxação

Lema 24.14, CLRS Seja P um caminho mínimo de s a v cuja última aresta é (u, v) e suponha que $d[u] = \mathrm{dist}(s, u)$ e que uma sequência de relaxações que inclui $\textsc{Relax}(u, v)$ é executada. Então imediatamente após isto, temos que $d[v] = \mathrm{dist}(s, v)$ e $d[v]$ nunca mais muda.

Prova. Note que $\omega(P) = \mathrm{dist}(s, v) = \mathrm{dist}(s, u) + \omega(u, v)$ pelo Lema 24.1. Então após a chamada $\textsc{Relax}(u, v, \omega)$, temos $d[v] \leq d[u] + \omega(u, v) = \mathrm{dist}(s, u) + \omega(u, v) = \mathrm{dist}(s, v)$. Pelo Lema 24.11, segue que $d[v] = \mathrm{dist}(s, v)$ e $d[v]$ nunca mais muda. ■

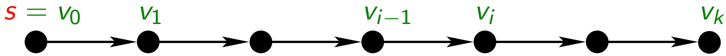


Propriedades de um algoritmo baseado em relaxação

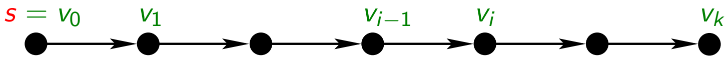
Lema 24.15, CLRS Seja $P = (v_0 = s, v_1, \ldots, v_k)$ um caminho mínimo de $s = v_0$ a v_k e suponha que as arestas (v_0, v_1),. . . , (v_{k-1}, v_k) são relaxadas nesta ordem.
Então após as relaxações, temos $d[v_k] = \mathrm{dist}(s, v_k)$ e $d[v_k]$ nunca mais muda.

Provaremos por indução em i que após a relaxação da aresta (v_{i-1}, v_i), temos $d[v_i] = \mathrm{dist}(s, v_i)$.

Base: $i = 0$ (antes de relaxar qualquer aresta de P). Claramente, $d[s] = 0 = \mathrm{dist}(s, s)$.



Hipótese de indução: suponha que $d[v_{i-1}] = \mathrm{dist}(s, v_{i-1})$.
Como $(v_0, \ldots, v_i)$ é um caminho mínimo de s a v_i (Lema 24.1), segue do Lema 24.14 que após $\textsc{Relax}(v_{i-1}, v_i, \omega)$ temos que $d[v_i] = \mathrm{dist}(s, v_i)$ e este valor nunca mais muda pelo Lema 24.11. ∎



Propriedades de um algoritmo baseado em relaxação

Seja G_π com conjunto de vértices $V_\pi = \{s\} \cup \{v \in V : \pi[v] \neq \text{NIL}\}$ e conjunto de arestas $E_\pi = \{(\pi[v], v) : v \in V, \pi[v] \neq \text{NIL}\}$.

Lema 24.16, CLRS Suponha que (G, ω) **não** contém ciclos negativos atingíveis por s. Então em qualquer iteração após a chamada INITIALIZE-SINGLE-SOURCE, o subgrafo G_π é uma árvore de raiz s. ■

Lema 24.17, CLRS Suponha que (G, ω) **não** contém ciclos negativos atingíveis por s. Suponha que sejam feitas uma chamada a INITIALIZE-SINGLE-SOURCE e uma sequência de chamadas a RELAX que resulta em $d[v] = \mathrm{dist}(s, v)$ para todo $v \in V$. Então G_π é uma Árvore de Caminhos Mínimos. ■

Omitimos as provas aqui.

Correção de Dag-Shortest-Paths

Dag-Shortest-Paths(G, ω, s)

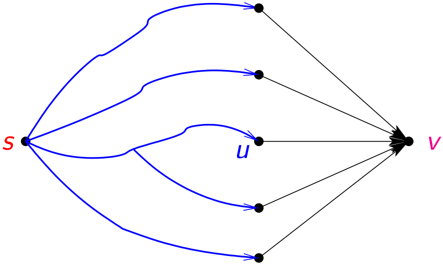
Ordene topologicamente os vértices de G
Initialize-Single-Source(G, s)
para cada vértice u na ordem topológica **faça**
  para cada $v \in \mathrm{Adj}[u]$ **faça**
    Relax(u, v, ω)
devolva d, π

Como os vértices estão em ordem topológica, as arestas de qualquer caminho mínimo $P = (v_0 = s, v_1, \ldots, v_k)$ são relaxadas na ordem (v_0, v_1),. . . ,(v_{k-1}, v_k).

Logo, pelo Lema 24.15 no final do algoritmo $d[v] = \mathrm{dist}(s, v)$ para todo $v \in V$.

Pelo Lema 24.17, o vetor $\pi[\,]$ define uma Árvore de Caminhos Mínimos com raiz s. Isto mostra que Dag-Shortest-Paths funciona corretamente.

Outra forma de mostrar a correção de Dag-Shortest-Paths é observar que vale a seguinte recorrência para $\mathrm{dist}(s, v)$:



$$\mathrm{dist}(s, v) = \min_{u : v \in \mathrm{Adj}[u]} \mathrm{dist}(s, u) + \omega(u, v).$$

DAG-SHORTEST-PATHS(G, ω, s)

1. Ordene topologicamente os vértices de G
2. INITIALIZE-SINGLE-SOURCE(G, s)
3. **para cada** vértice u na ordem topológica **faça**
4.   **para cada** $v \in \mathrm{Adj}[u]$ **faça**
5.   RELAX(u, v, ω)
6. **devolva** d, π

A correção de DAG-SHORTEST-PATHS segue por indução e da fórmula de recorrência

$$\mathrm{dist}(s, v) = \min_{u : v \in \mathrm{Adj}[u]} \mathrm{dist}(s, u) + \omega(u, v).$$

Perguntas

Pergunta 1. Como se resolve o problema de encontrar um caminho de peso máximo de s a t em um grafo orientado acíclico (G, ω)?

Pergunta 2. Como se resolve o Problema do Caminho Mínimo de s a t em tempo linear para um grafo orientado em que todas as arestas tem o mesmo peso $C > 0$?

Veremos agora um algoritmo para caminhos mínimos em grafos que podem conter ciclos, mas sem arestas de peso negativo.

O algoritmo foi proposto por E.W. Dijkstra e é bastante similar ao algoritmo de Prim para o problema da Árvore Geradora Mínima.

O algoritmo de Dijkstra recebe um grafo orientado (G, ω) (sem arestas de peso negativo) e um vértice s de G

e devolve

- para cada $v \in V[G]$, o valor $d[v] = \mathrm{dist}(s, v)$ (ou seja, o peso de um caminho mínimo de s a v) e
- uma Árvore de Caminhos Mínimos com raiz s.

  Um caminho de s a v nesta árvore tem peso $d[v]$, ou seja, é um caminho mínimo de s a v em (G, ω).

Revisão: algoritmos baseados em relaxação

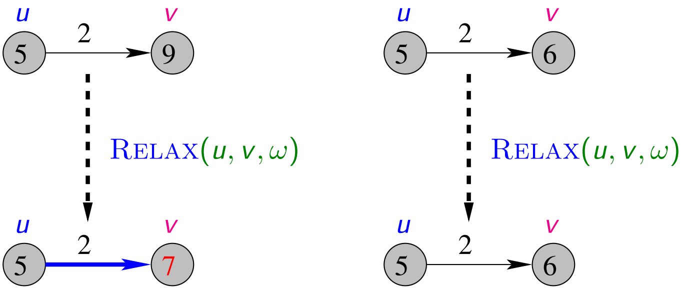
INITIALIZE-SINGLE-SOURCE(G, s)

```
1    para cada vértice v ∈ V[G] faça
2        d[v] ← ∞
3        π[v] ← NIL
4    d[s] ← 0
```

- O valor $d[v]$ é uma estimativa superior de $\mathrm{dist}(s, v)$, o peso de um caminho mínimo de s a v.
- Se $d[v] < \infty$, então o algoritmo encontrou até aquele momento um caminho de s a v com peso $d[v]$.
- O caminho de s a v pode ser recuperado por meio dos predecessores $\pi[\,]$, se não houver ciclos negativos.

Revisão: relaxação

Tenta melhorar a estimativa $d[v]$ examinando (u, v).



$\text{Relax}(u, v, \omega)$

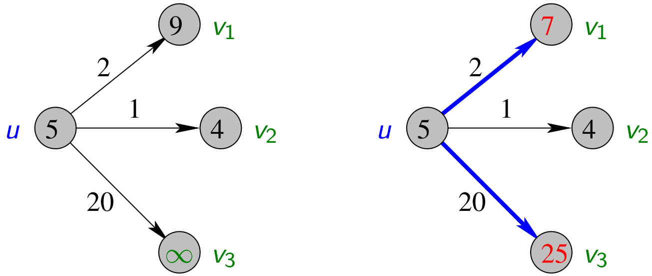
se $d[v] > d[u] + \omega(u, v)$

  então $d[v] \leftarrow d[u] + \omega(u, v)$

    $\pi[v] \leftarrow u$

Revisão: relaxação dos vizinhos

Em cada iteração o algoritmo seleciona um vértice u e para cada vizinho v de u aplica $\textsc{Relax}(u, v, \omega)$.



$\textsc{Relax}(u, v, \omega)$

se $d[v] > d[u] + \omega(u, v)$ **faça**

$\quad d[v] \leftarrow d[u] + \omega(u, v)$

$\quad \pi[v] \leftarrow u$

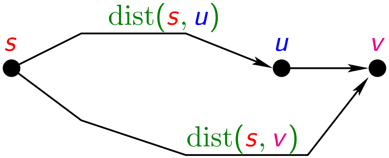
As propriedades que veremos valem para qualquer "algoritmo" que satisfaz as restrições abaixo.

- A inicialização é feita por INITIALIZE-SINGLE-SOURCE(G, s).
- Um valor $d[v]$ (e $\pi[v]$) só pode ser modificado através de uma chamada de RELAX(u, v, ω) para alguma aresta (u, v).

Revisão: algoritmos baseados em relaxação

Lema 24.10, CLRS Para toda aresta (u, v) temos que

$$\mathrm{dist}(s, v) \leq \mathrm{dist}(s, u) + \omega(u, v).$$



Lema 24.14, CLRS Seja P um caminho mínimo de s a v cuja última aresta é (u, v) e suponha que $d[u] = \mathrm{dist}(s, u)$ e que uma sequência de relaxações que inclui $\textsc{Relax}(u, v)$ é executada. Então imediatamente após isto, temos que $d[v] = \mathrm{dist}(s, v)$ e $d[v]$ nunca mais muda.

Seja G_π com conjunto de vértices $V_\pi = \{s\} \cup \{v \in V : \pi[v] \neq \text{NIL}\}$ e conjunto de arestas $E_\pi = \{(\pi[v], v) : v \in V, \pi[v] \neq \text{NIL}\}$.

Lema 24.16, CLRS Suponha que (G, ω) **não** contém ciclos negativos atingíveis por s. Então em qualquer iteração após a chamada INITIALIZE-SINGLE-SOURCE, o subgrafo G_π é uma árvore de raiz s. ∎

Lema 24.17, CLRS Suponha que (G, ω) **não** contém ciclos negativos atingíveis por s. Suponha que sejam feitas uma chamada a INITIALIZE-SINGLE-SOURCE e uma sequência de chamadas a RELAX que resulta em $d[v] = \mathrm{dist}(s, v)$ para todo $v \in V$. Então G_π é uma Árvore de Caminhos Mínimos. ∎

Ideia do algoritmo de Dijkstra

- Suponha que encontramos até o momento um conjunto S formado pelos vértices mais próximos de s.
  (Na primeira iteração $S = \{s\}$).
- O algoritmo mantém também uma Árvore de Caminhos Mínimos formada apenas por vértices de S.
- A ideia é estender o conjunto S (i.e., a árvore) acrescentando o vértice u em $V - S$ que esteja mais próximo de s.

  Um detalhe importante é como encontrar tal vértice.

$\textsc{Dijkstra}(G, \omega, s)$

$\quad \textsc{Initialize-Single-Source}(G, s)$
$\quad S \leftarrow \emptyset$
$\quad Q \leftarrow V[G]$
$\quad$ **enquanto** $Q \neq \emptyset$ **faça**
$\quad\quad u \leftarrow \textsc{Extract-Min}(Q)$
$\quad\quad S \leftarrow S \cup \{u\}$
$\quad\quad$ **para cada** vértice $v \in \mathrm{Adj}[u]$ **faça**
$\quad\quad\quad \textsc{Relax}(u, v, \omega)$
$\quad$ **devolva** d, π

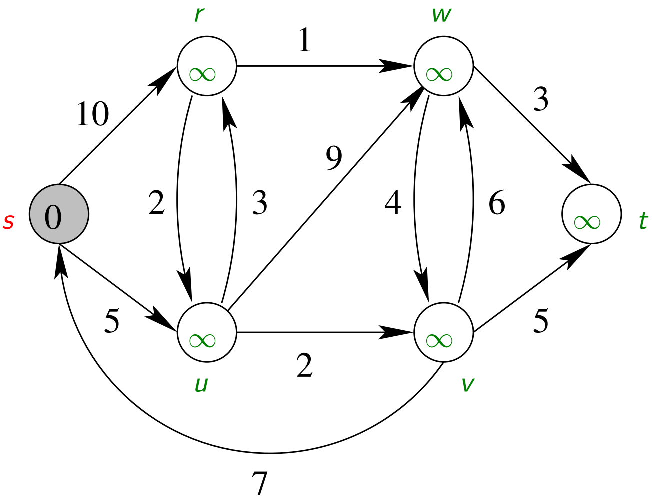
O conjunto Q é implementado como uma fila de prioridade com chave d.

O conjunto S não é realmente necessário, mas simplifica a análise do algoritmo.

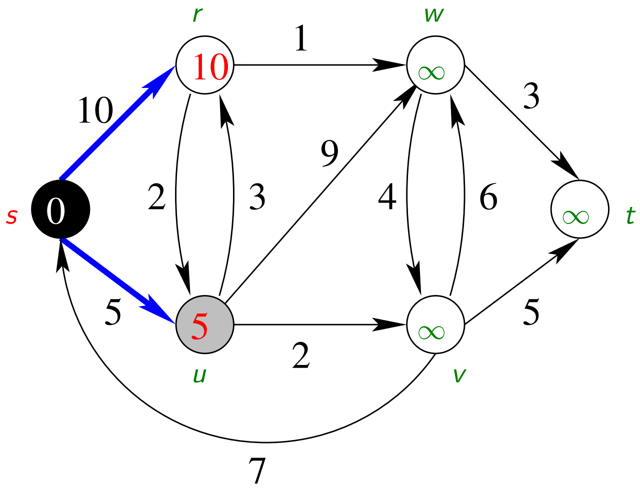
$\textsc{Dijkstra}(G, \omega, s)$
1 $\textsc{Initialize-Single-Source}(G, s)$
2 $S \leftarrow \emptyset$
3 $Q \leftarrow V[G]$
4 **enquanto** $Q \neq \emptyset$ **faça**
5 $\quad u \leftarrow \textsc{Extract-Min}(Q)$
6 $\quad S \leftarrow S \cup \{u\}$
7 $\quad$ **para cada** vértice $v \in \mathrm{Adj}[u]$ **faça**
8 $\quad\quad \textsc{Relax}(u, v, \omega)$
9 **devolva** d, π

Mostraremos que em cada iteração da linha 5, o vértice u com $d[u]$ mínimo é realmente o vértice de $V[G] - S$ que está mais próximo de s.

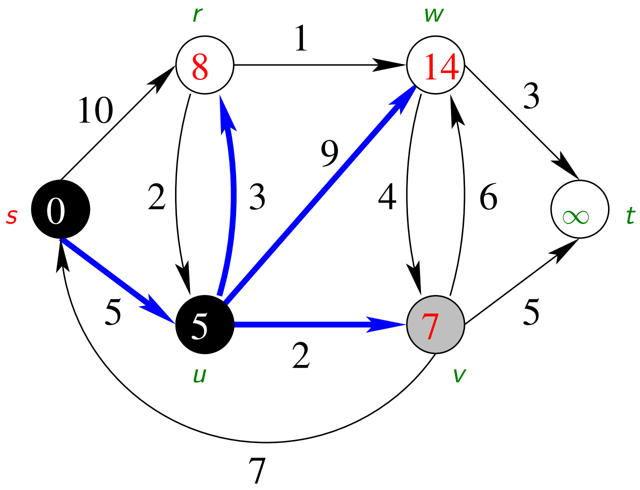
Exemplo (CLRS modificado)



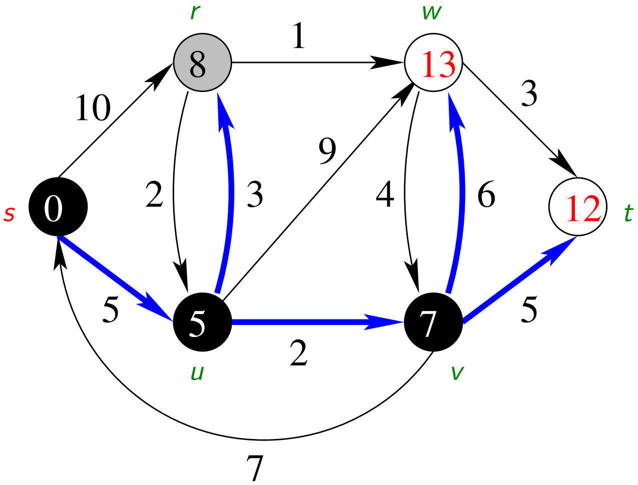
Exemplo (CLRS modificado)



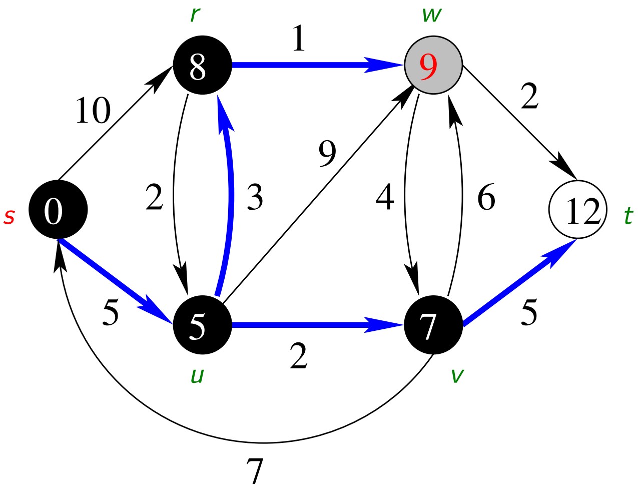
Exemplo (CLRS modificado)



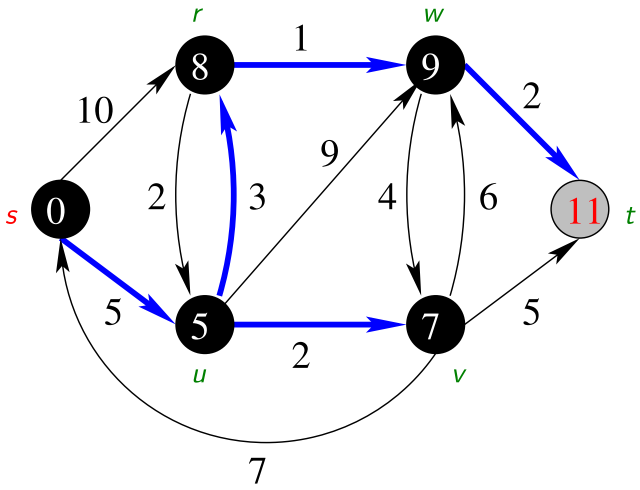
Exemplo (CLRS modificado)



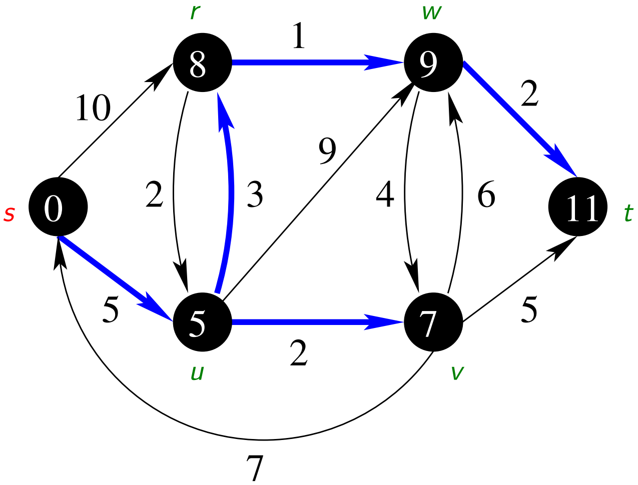
Exemplo (CLRS modificado)



Exemplo (CLRS modificado)



Exemplo (CLRS modificado)



Correção do algoritmo

Precisamos provar que quando o algoritmo pára, temos que

- $d[v] = \mathrm{dist}(s, v)$ para todo $v \in V[G]$ e
- o grafo G_π é uma Árvore de Caminhos Mínimos com raiz s.

Invariante principal

O seguinte invariante vale no início de cada iteração da linha 4 no algoritmo DIJKSTRA.

Invariante: $d[x] = \mathrm{dist}(s, x)$ para cada $x \in S$.

- Claramente o invariante vale na primeira iteração pois $S = \emptyset$. Também vale no início da segunda iteração pois $S = \{s\}$ e $d[s] = 0$.
- No final do algoritmo, S é o conjunto dos vértices atingíveis por s. Portanto, se o invariante vale, para cada $v \in V[G]$, o valor $d[v]$ é exatamente a distância de s a v.

Demonstração do invariante

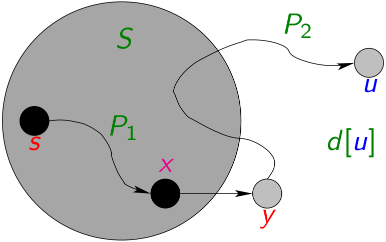
- O algoritmo de DIJKSTRA escolhe um vértice u com menor $d[u]$ em Q e atualiza $S \leftarrow S \cup \{u\}$.
- Basta verificar então que neste momento $d[u] = \mathrm{dist}(s, u)$.

Suponha por contradição que em alguma iteração o algoritmo escolhe u tal que $d[u] > \mathrm{dist}(s, u)$.

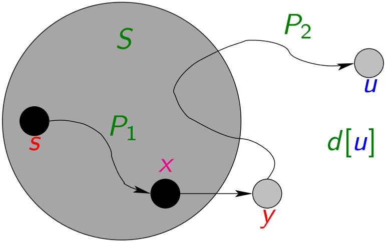
Podemos supor que u é o primeiro vértice para o qual isto ocorre.
Note que $u \neq s$.

Demonstração

Seja P um caminho mínimo de s a u (ou seja, com peso $\mathrm{dist}(s, u)$). Seja y o primeiro vértice de P que não pertence a S. Seja x o vértice em P que precede y.



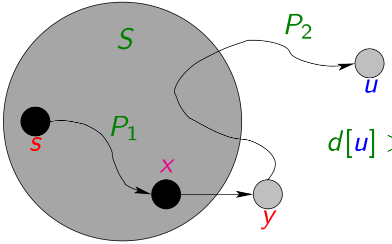
$d[u] > \mathrm{dist}(s, u)$?



$d[u] > \mathrm{dist}(s, u)$?

Primeiro, mostraremos que $d[y] = \mathrm{dist}(s, y)$.

Por hipótese, quando x foi colocado em S, $d[x] = \mathrm{dist}(s, x)$.
Neste instante, (x, y) foi relaxada e portanto, pelo Lema 24.14 temos $d[y] = \mathrm{dist}(s, y)$.



$d[u] > \mathrm{dist}(s, u)$?

Então

$$\begin{aligned} d[y] &= \mathrm{dist}(s, y) \\ &= \omega(P_1) + \omega(x, y) \\ &\leq \omega(P_1) + \omega(x, y) + \omega(P_2) \\ &= \mathrm{dist}(s, u) < d[u]. \end{aligned}$$

Mas então $d[y] < d[u]$ o que contraria a escolha de u.

Demonstração

Isto mostra que ao final do algoritmo de Dijkstra, temos $d[v] = \mathrm{dist}(s, v)$ para todo $v \in V[G]$. Do Lema 24.17 segue que G_π é uma Árvore de Caminhos Mínimos com raiz s de (G, ω). $\blacksquare$

A prova acima implica que os vértices u são colocados em S em ordem não-decrescente dos valores $\mathrm{dist}(s, u)$.

Na demonstração foi importante o fato de não haver arestas negativas no grafo. De fato, não se pode garantir que o algoritmo de Dijkstra funciona se o grafo de entrada não satisfaz esta restrição.

Exercício. Encontre um grafo orientado ponderado com 4 vértices para o qual o algoritmo de Dijkstra não funciona. Há pelo menos um exemplo com apenas uma única aresta negativa e sem ciclos de peso negativo.

Complexidade de tempo

```
Dijkstra(G, ω, s)
    Initialize-Single-Source(G, s)
    S ← ∅
    Q ← V[G]
    enquanto Q ≠ ∅ faça
        u ← Extract-Min(Q)
        S ← S ∪ {u}
        para cada vértice v ∈ Adj[u] faça
            Relax(u, v, ω)
    devolva d, π
```

Depende de como a fila de prioridade Q é implementada.

Complexidade do algoritmo de Dijkstra

- As linhas 1–3 correspondem a $|V|$ chamadas a Insert.
- O laço da linha 4 é executado $O(V)$ vezes.
  Total: $O(V)$ chamadas a Extract-Min.
- O laço das linhas 7–8 é executado $O(E)$ vezes no total.
  Ao atualizar uma chave na linha 8 é feita uma *chamada implícita* a Decrease-Key.
  Total: $O(E)$ chamadas a Decrease-Key.
- **Tempo total**:
  $O(V)$ Insert+ $O(V)$ Extract-Min+
  $O(E)$ Decrease-Key

Complexidade do algoritmo de Dijkstra

Tempo total

$O(V)$ INSERT+ $O(V)$ EXTRACT-MIN+
$O(E)$ DECREASE-KEY

- Implementando Q como um vetor (coloque $d[v]$ na posição v do vetor), INSERT e DECREASE-KEY gastam tempo $\Theta(1)$ e EXTRACT-MIN gasta tempo $O(V)$, resultando em um total de $O(V^2 + E) = O(V^2)$.
- Implementando a fila de prioridade Q como um min-heap, INSERT, EXTRACT-MIN e DECREASE-KEY gastam tempo $O(\lg V)$, resultando em um total de $O((V + E) \lg V)$.
- Usando heaps de Fibonacci (EXTRACT-MIN é $O(\lg V)$ e INSERT e DECREASE-KEY são $O(1)$) a complexidade cai para $O(V \lg V + E)$.

Arestas/ciclos de peso negativo

- O algoritmo de Dijkstra resolve o Problema dos Caminhos Mínimos quando (G, ω) **não** possui arestas de peso negativo.
- Quando (G, ω) possui arestas de peso negativo, o algoritmo de Dijkstra não funciona.
- Uma das dificuldades com arestas de peso negativo é a possível existência de ciclos negativos.

Ciclos negativos — uma dificuldade

- Pode-se mostrar que o Problema dos Caminhos Mínimos para instâncias com ciclos negativos é NP-difícil.
- Informalmente, se um problema pertence à classe dos problemas NP-difíceis então acredita-se que não existe algoritmo eficiente para resolvê-lo.
- Este é o principal motivo do porquê nos restringimos ao Problema de Caminhos Mínimos sem ciclos negativos.

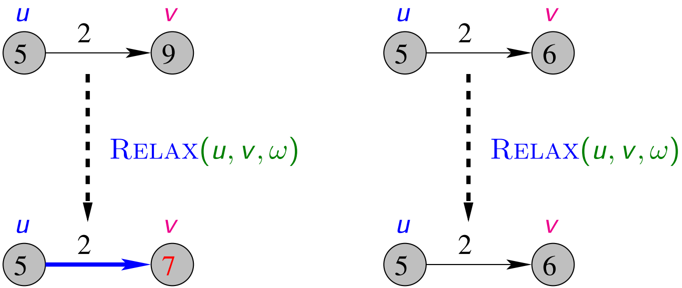
Revisão: inicialização

$\text{INITIALIZE-SINGLE-SOURCE}(G, s)$

1 **para cada** vértice $v \in V[G]$ **faça**
2 $\quad d[v] \leftarrow \infty$
3 $\quad \pi[v] \leftarrow \text{NIL}$
4 $d[s] \leftarrow 0$

- O valor $d[v]$ é uma estimativa superior de $\mathrm{dist}(s, v)$, o peso de um caminho mínimo de s a v.
- Se $d[v] < \infty$, então algoritmo encontrou até aquele momento um caminho de s a v com peso $d[v]$.
- O caminho de s a v pode ser recuperado por meio dos predecessores $\pi[\,]$, se não houver ciclos negativos.

Tenta melhorar a estimativa $d[v]$ examinando (u, v).



Relax(u, v, ω)

1 **se** $d[v] > d[u] + \omega(u, v)$

2 **então** $d[v] \leftarrow d[u] + \omega(u, v)$

3 $\pi[v] \leftarrow u$

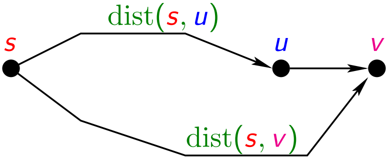
As propriedades que veremos valem para qualquer "algoritmo" que satisfaz as restrições abaixo.

- A inicialização é feita por INITIALIZE-SINGLE-SOURCE(G, s).
- Um valor $d[v]$ (e $\pi[v]$) só pode ser modificado através de uma chamada de RELAX(u, v, ω) para alguma aresta (u, v).

Revisão: algoritmos baseados em relaxação

Lema 24.10, CLRS Para toda aresta (u, v) temos que

$$\mathrm{dist}(s, v) \leq \mathrm{dist}(s, u) + \omega(u, v).$$



Lema 24.15, CLRS Seja $P = (v_0 = s, v_1, \ldots, v_k)$ um caminho mínimo de $s = v_0$ a v_k e suponha que as arestas (v_0, v_1),. . . , (v_{k-1}, v_k) são relaxadas nesta ordem.

Então após as relaxações, temos $d[v_k] = \mathrm{dist}(s, v_k)$ e $d[v_k]$ nunca mais muda.

Seja G_π com conjunto de vértices $V_\pi = \{s\} \cup \{v \in V : \pi[v] \neq \text{NIL}\}$ e conjunto de arestas $E_\pi = \{(\pi[v], v) : v \in V, \pi[v] \neq \text{NIL}\}$.

Lema 24.16, CLRS Suponha que (G, ω) **não** contém ciclos negativos atingíveis por s. Então em qualquer iteração após a chamada INITIALIZE-SINGLE-SOURCE, o subgrafo G_π é uma árvore de raiz s. ∎

Lema 24.17, CLRS Suponha que (G, ω) **não** contém ciclos negativos atingíveis por s. Suponha que sejam feitas uma chamada a INITIALIZE-SINGLE-SOURCE e uma sequência de chamadas a RELAX que resulta em $d[v] = \text{dist}(s, v)$ para todo $v \in V$. Então G_π é uma Árvore de Caminhos Mínimos. ∎

Ideia do algoritmo de Bellman-Ford

- Descreveremos a seguir o algoritmo de Bellman-Ford que resolve o Problema dos Caminhos Mínimos em grafos que podem ter arestas de peso negativo, mas não contém ciclos negativos.
- Na verdade, o algoritmo consegue resolver problema mesmo que tenham ciclos negativos, mas não sejam atingíveis pela origem s.
- Ele também consegue detectar se existe um ciclo negativo atingível por s.

- Na iteração $i = 1$ executamos $\textsc{Relax}$ para todas as arestas. Isto garante que a primeira aresta de qualquer caminho mínimo será relaxada.
- Repetimos este passo para as iterações $i = 2, 3, \ldots, |V| - 1$. Por que $|V| - 1$?
  Porque um caminho (simples) tem no máximo $|V| - 1$ arestas. Assim, após essas $|V| - 1$ iterações, garantidamente, todas as arestas de um caminho mínimo foram relaxadas na ordem desejada.
- É preciso fazer mais uma iteração de relaxar todas as arestas para verificar se o grafo contém ciclos negativos.
  Se houver, não há garantia de que os valores obtidos nas $|V| - 1$ primeiras iterações estão corretas.

O algoritmo de Bellman-Ford

O algoritmo de Bellman-Ford recebe um grafo orientado (G, ω) (possivelmente com arestas de peso negativo) e um vértice origem s de G.

Ele devolve um valor booleano

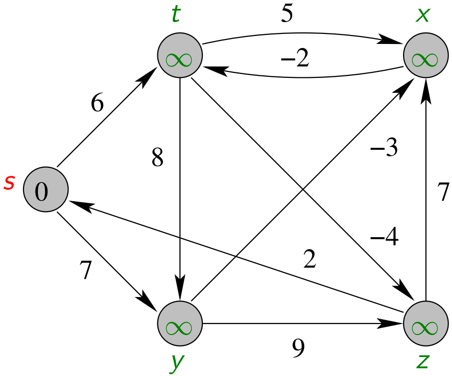
- FALSE se existe um ciclo negativo atingível a partir de s, ou
- TRUE e neste caso devolve também uma Árvore de Caminhos Mínimos com raiz s.

O algoritmo de Bellman-Ford

```
BELLMAN-FORD(G, ω, s)
    INITIALIZE-SINGLE-SOURCE(G, s)
    para i ← 1 até |V[G]| − 1 faça
        para cada aresta (u, v) ∈ E[G] faça
            RELAX(u, v, ω)
    para cada aresta (u, v) ∈ E[G] faça
        se d[v] > d[u] + ω(u, v)
            então devolva FALSE
    devolva TRUE, d, π
```

Complexidade de tempo: $O(VE)$

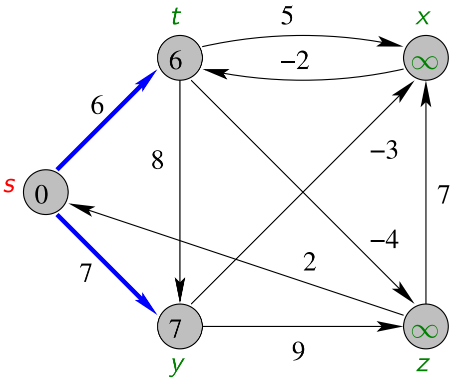
Exemplo (CLRS)



Ordem:

$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

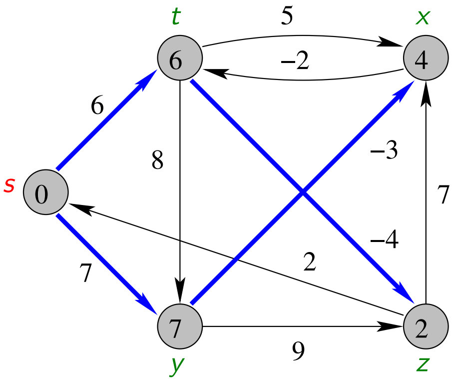
Exemplo (CLRS)



Ordem:

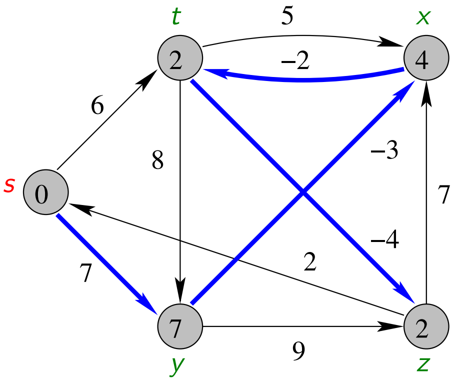
$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

Exemplo (CLRS)



Ordem:
$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$.

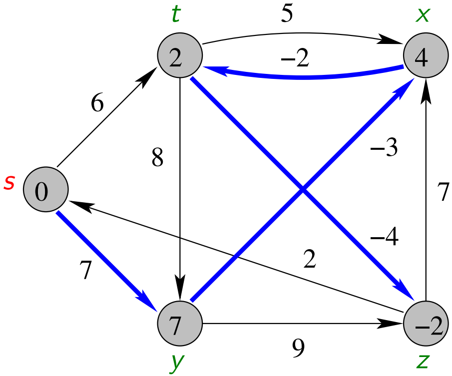
Exemplo (CLRS)



Ordem:

$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$.

Exemplo (CLRS)

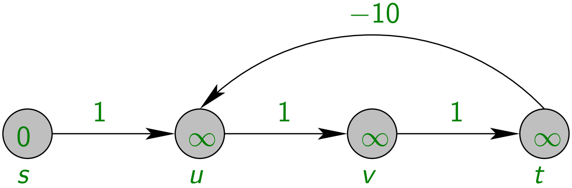


Ordem:

$(t, x), (t, y), (t, z), (x, t), (y, x), (y, z), (z, x), (z, s), (s, t), (s, y)$.

Exemplo com ciclo negativo

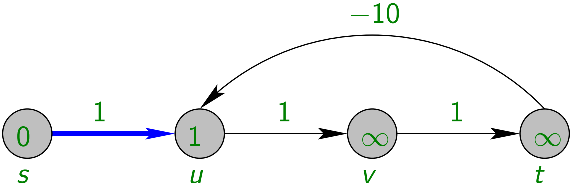
Inicialização.



Ordem: $(t, u), (v, t), (u, v), (s, u)$.

Exemplo com ciclo negativo

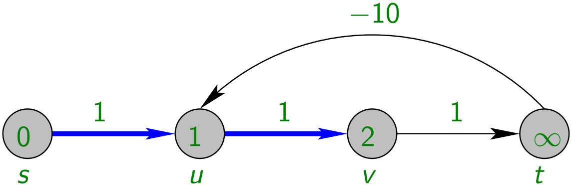
Final da iteração 1.



Ordem: $(t, u), (v, t), (u, v), (s, u)$.

Exemplo com ciclo negativo

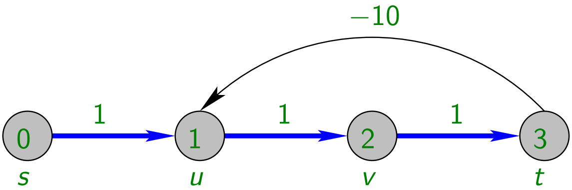
Final da iteração 2.



Ordem: $(t, u), (v, t), (u, v), (s, u)$.

Exemplo com ciclo negativo

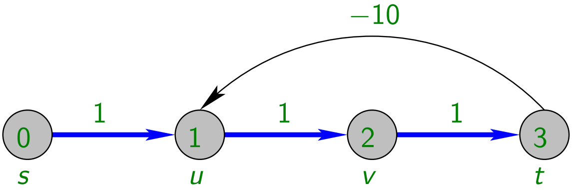
Final da iteração 3.



Ordem: $(t, u), (v, t), (u, v), (s, u)$.

Exemplo com ciclo negativo

Iteração 4: $1 = d[u] > d[t] + \omega(t, u) = 3 - 10 = -7$



Ordem: $(t, u), (v, t), (u, v), (s, u)$.

Teorema 24.4 (CLRS).

Se (G, ω) não contém ciclos negativos atingíveis por s, então no final

- o algoritmo devolve TRUE,
- $d[v] = \mathrm{dist}(s, v)$ para $v \in V$ e
- $\pi[\,]$ define uma Árvore de Caminhos Mínimos.

Se (G, ω) contém ciclos negativos atingíveis por s, então no final o algoritmo devolve FALSE.

Correção do algoritmo Bellman-Ford

Primeiramente, vamos supor que o grafo não possui ciclos negativos atingíveis por s.

Relembrando...

Lema 24.15, CLRS Seja $P = (v_0 = s, v_1, \ldots, v_k)$ um caminho mínimo de $s = v_0$ a v_k e suponha que as arestas (v_0, v_1),..., (v_{k-1}, v_k) são relaxadas nesta ordem.
Então após as relaxações, temos $d[v_k] = \mathrm{dist}(s, v_k)$ e $d[v_k]$ nunca mais muda.

Seja v um vértice atingível por s e seja

$$P = (v_0 = s, v_1, \ldots, v_k = v)$$

um caminho mínimo de s a v.

Note que P tem no máximo $|V| - 1$ arestas. Cada uma das $|V| - 1$ iterações do laço das linhas 2–4 relaxa todas as $|E|$ arestas.

Na iteração i a aresta (v_{i-1}, v_i) é relaxada.

Logo, pelo Lema 24.16, $d[v] = d[v_k] = \mathrm{dist}(s, v)$.

Se v é um vértice não atingível por s então $d[v] = \infty$, pois $d[v]$ foi inicializado com este valor e ao longo do algoritmo vale que $d[v] \geq \mathrm{dist}(s, v) = \infty$.

Assim, no final do algoritmo $d[v] = \mathrm{dist}(s, v)$ para $v \in V$.

Pelo Lema 24.17, G_π é uma Árvore de Caminhos Mínimos com raiz s de (G, ω).

Correção do algoritmo Bellman-Ford

Agora falta mostrar que BELLMAN-FORD devolve TRUE.

Seja (u, v) uma aresta de G. Então

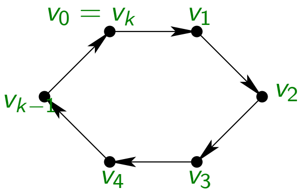
$$\begin{aligned} d[v] &= \mathrm{dist}(s, v) \\ &\leq \mathrm{dist}(s, u) + \omega(u, v) \quad \text{(Lema 24.10)} \\ &= d[u] + \omega(u, v). \end{aligned}$$

Assim, nenhum dos testes da linha 6 faz com que o algoritmo devolva FALSE. Logo, ele devolve TRUE.

Suponha que (G, ω) contém um ciclo negativo atingível por s.

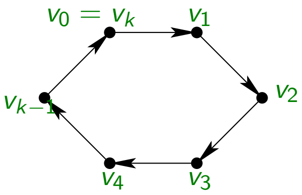
Seja $C = (v_0, v_1, \dots, v_k = v_0)$ um tal ciclo.

Então $\sum_{i=1}^{k} \omega(v_{i-1}, v_i) < 0$.



Correção do algoritmo Bellman-Ford

Suponha por contradição que o algoritmo devolve TRUE.
Então $d[v_i] \leq d[v_{i-1}] + \omega(v_{i-1}, v_i)$ para $i = 1, 2 \ldots, k$.



Somando as desigualdades ao longo do ciclo temos

$$\begin{aligned}\sum_{i=1}^{k} d[v_i] &\leq \sum_{i=1}^{k}(d[v_{i-1}] + \omega(v_{i-1}, v_i)) \\ &= \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} \omega(v_{i-1}, v_i).\end{aligned}$$

Como $v_0 = v_k$, temos que $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$.

Mas então $\sum_{i=1}^{k} \omega(v_{i-1}, v_i) \geq 0$ o que contraria o fato do ciclo ser negativo.
Isto conclui a prova de correção.

Redução para PCM sem pesos negativos

- Às vezes é possível transformar uma instância (G, ω) com arestas de peso negativo (mas **sem** ciclos negativos) em uma instância equivalente (G, ω') **sem** arestas de peso negativo.
- Suponha que você conhece um valor $p[v]$ para todo $v \in V$ que satisfaz a seguinte propriedade:

$$p[v] \leq p[u] + \omega(u, v) \quad \text{para toda aresta } (u, v) \in E.$$

- Por exemplo, a função $p[v] := \mathrm{dist}(s, v)$ satisfaz esta propriedade! Mas suporemos que não é este o caso, pois o que queremos determinar é exatamente $\mathrm{dist}(s, v)$ para $v \in V$.

- Considere a instância (G, ω') onde
$$\omega'(u,v) = \omega(u,v) + p[u] - p[v] \quad \text{para toda aresta } (u,v) \in E.$$
Note que $\omega'(u,v) \geq 0$ para toda aresta $(u,v) \in E$.
- Seja $P = (s = v_0, v_1, \ldots, v_{k-1}, v_k = t)$ um caminho de s a t em G. Então

$$\begin{aligned}
\omega'(P) &= \sum_{i=1}^{k} \omega'(v_{i-1}, v_i) \\
&= \sum_{i=1}^{k} \Big(\omega(v_{i-1}, v_i) + p[v_{i-1}] - p[v_i]\Big) \\
&= \sum_{i=1}^{k} \omega(v_{i-1}, v_i) + \sum_{i=1}^{k} (p[v_{i-1}] - p[v_i]) \\
&= \omega(P) + p[v_0] - p[v_k] = \omega(P) + p[s] - p[t].
\end{aligned}$$

Redução para PCM sem pesos negativos

- Considere a instância (G, ω') onde

$$\omega'(u, v) = \omega(u, v) + p[u] - p[v] \quad \text{para toda aresta } (u, v) \in E.$$

- Seja $P = (s = v_0, v_1, \ldots, v_{k-1}, v_k = t)$ um caminho de s a t em G.
- Então $\omega'(P) = \omega(P) + p[s] - p[t]$.
- Como p está fixo, encontrar um caminho mínimo de s a t em (G, ω) é equivalente a encontrar um caminho mínimo de s a t em (G, ω').

Redução para PCM sem pesos negativos

- O único problema com esta ideia é que para encontrar uma tal função p, aparentemente é preciso resolver um PCM com arestas de peso negativo, o que **invalida** a ideia de modo geral.
- Entretanto, há situações em que tal função p é conhecida e neste caso pode-se usar o algoritmo de Dijkstra que é mais eficiente que o o algoritmo de Bellman-Ford que resolve o PCM com arestas de peso negativo.
- Uma função p como descrita é chamada potencial em alguns contextos. Veremos como usar o algoritmo de Bellman-Ford para encontrar uma função potencial.

Considere o seguinte sistema de desigualdades lineares:

$$
\begin{aligned}
x_1 - x_2 &\leq 0 \\
x_1 - x_5 &\leq -1 \\
x_2 - x_5 &\leq 1 \\
x_3 - x_1 &\leq 5 \\
x_4 - x_1 &\leq 4 \\
x_4 - x_3 &\leq -1 \\
x_5 - x_3 &\leq -3 \\
x_5 - x_4 &\leq -3
\end{aligned}
$$

Queremos encontrar valores $x_1, x_2, \ldots, x_n$ que satisfazem estas restrições de diferença.

Sistemas de restrições de diferenças

Sistemas de restrições de diferença tem várias aplicações.

$$
\begin{aligned}
x_1 - x_2 &\leq 0 \\
x_1 - x_5 &\leq -1 \\
x_2 - x_5 &\leq 1 \\
x_3 - x_1 &\leq 5 \\
x_4 - x_1 &\leq 4 \\
x_4 - x_3 &\leq -1 \\
x_5 - x_3 &\leq -3 \\
x_5 - x_4 &\leq -3
\end{aligned}
$$

Por exemplo, a incógnita x_i pode representar o instante do tempo que um evento i deve ocorrer. Uma restrição $x_j - x_i \leq b_k$ diz que uma certa quantidade de tempo b_k deve passar entre os eventos i e j. Um evento poderia ser a execução de uma certa tarefa durante a fabricação de um produto.

Aplicação: Sistemas de restrições de diferenças

Considere o seguinte sistema de desigualdades lineares:

$$
\begin{aligned}
x_1 - x_2 &\leq 0 \\
x_1 - x_5 &\leq -1 \\
x_2 - x_5 &\leq 1 \\
x_3 - x_1 &\leq 5 \\
x_4 - x_1 &\leq 4 \\
x_4 - x_3 &\leq -1 \\
x_5 - x_3 &\leq -3 \\
x_5 - x_4 &\leq -3
\end{aligned}
$$

Podemos resolver um sistema deste tipo para encontrar uma função potencial p de um grafo (G, ω). Uma função potencial de (G, ω) deve satisfazer $p(v) - p(u) \leq \omega(u, v)$ para cada aresta (u, v) de G. Assim, temos uma desigualdade $x_v - x_u \leq \omega(u, v)$ para cada aresta (u, v) de G.

Podemos reescrever as restrições matricialmente:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}$$

Um sistema de restrições de diferença é um sistema da forma $Ax \leq b$ onde A é uma matriz com entradas $\{-1, 0, 1\}$ em que cada linha há exatamente um 1 e um -1.

Podemos reescrever as restrições matricialmente:

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \leq \begin{pmatrix} 0 \\ -1 \\ 1 \\ 5 \\ 4 \\ -1 \\ -3 \\ -3 \end{pmatrix}$$

Uma solução é $x = (-5, -3, 0, -1, -4)$.
Outra solução é $x' = (0, 2, 5, 4, 1)$.

Sistemas de restrições de diferenças

Lema 24.8 (CLRS) Seja $x = (x_1, \ldots, x_n)$ uma solução de um sistema $Ax \leq b$ de restrições de diferença. Seja d uma constante. Então

$$x + d = (x_1 + d, \ldots, x_n + d)$$

também é uma solução de $Ax \leq b$.

Mostraremos a seguir como encontrar uma solução de um sistema $Ax \leq b$ de restrições de diferença resolvendo um problema de caminhos mínimos.

Grafo de restrições

A matriz A de dimensões $m \times n$ pode ser vista como a transposta de uma matriz de incidência de um grafo orientado (veja a lista de exercícios) com n vértices e m arestas.

Cada vértice v_i corresponde a uma variável x_i.

Cada aresta (v_i, v_j) corresponde a uma restrição $x_j - x_i \leq b_k$.

A este grafo acrescentamos um vértice origem v_0.

Formalmente, dado o sistema $Ax \leq b$ de restrições de diferença, construímos o grafo $G = (V, E)$ tal que

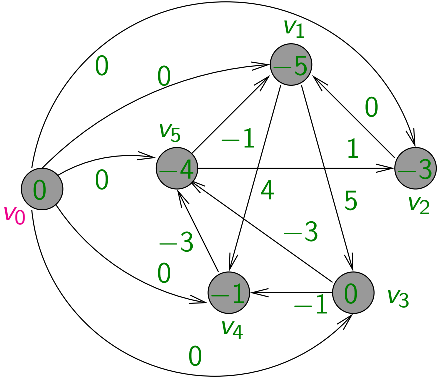
$$V = \{v_0, v_1, \ldots, v_n\}$$

e

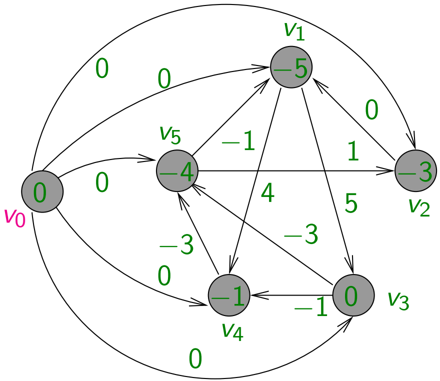
$$\begin{aligned} E &= \{(v_i, v_j) : x_j - x_i \leq b_k \text{ é uma restrição}\} \\ &\quad \cup \{(v_0, v_1), (v_0, v_2), \ldots, (v_0, v_n)\}. \end{aligned}$$

Para cada restrição $x_j - x_i \leq b_k$ temos uma aresta (v_i, v_j) com peso $\omega(v_i, v_j) = b_k$. O peso da aresta (v_0, v_i) é igual a zero para todo v_i.

Grafo de restrições



Uma restrição $x_j - x_i \leq b_k$ corresponde a uma aresta (v_i, v_j) com peso b_k.
Note que todo vértice é atingível a partir de v_0.

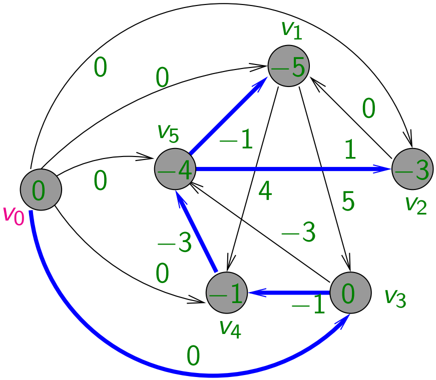


Uma restrição $x_j - x_i \le b_k$ corresponde a uma aresta (v_i, v_j) com peso b_k.
Ideia. Note que se $x_i = \mathrm{dist}(v_0, v_i)$ então:

$$x_j - x_i = \mathrm{dist}(v_0, v_j) - \mathrm{dist}(v_0, v_i) \le \omega(v_i, v_j) = b_k$$

e x é uma solução do sistema de diferenças.

Grafo de restrições



Uma solução viável é $x = (-5, -3, -0, -1, -4)$.

Teorema 24.9 (CLRS) Seja $Ax \leq b$ um sistema de restrições de diferença e seja $G = (V, E)$ o grafo de restrições associado.
Se G não contém ciclos negativos, então

$$x = (\mathrm{dist}(v_0, v_1), \mathrm{dist}(v_0, v_2), \ldots, \mathrm{dist}(v_0, v_n))$$

é uma solução viável do sistema.
Se G contém ciclos negativos, então o sistema não possui solução viável.

Suponha que (G,ω) não contém ciclos negativos. Para ver que $x_i = \mathrm{dist}(v_0, v_i)$ para $i = 1, 2, \ldots, n$ é uma solução do sistemas de diferença lembre-se que $\mathrm{dist}(v_0, v_j) \leq \mathrm{dist}(v_0, v_i) + \omega(v_i, v_j)$ para toda aresta (v_i, v_j).

Assim,

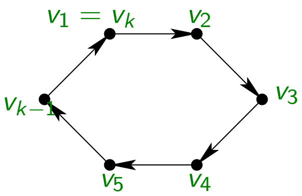
$$x_j - x_i = \mathrm{dist}(v_0, v_j) - \mathrm{dist}(v_0, v_i) \leq \omega(v_i, v_j) = b_k$$

e x é uma solução do sistema de diferenças.

Grafo de restrições e sistemas de diferenças

Suponha agora que (G, ω) contém um ciclo negativo C. Como a origem v_0 é uma fonte, ela não pertence a nenhum ciclo. Podemos supor sem perda de generalidade que:

$$C = (v_1, v_2, \ldots, v_k = v_1).$$



Podemos supor sem perda de generalidade que:

$$C = (v_1, v_2, \ldots, v_k = v_1).$$

Suponha por contradição que existe uma solução viável x do sistema de diferenças. Então

$$\begin{aligned} x_2 - x_1 &\leq \omega(v_1, v_2) \\ x_3 - x_2 &\leq \omega(v_2, v_3) \\ &\vdots \\ x_{k-1} - x_{k-2} &\leq \omega(v_{k-2}, v_{k-1}) \\ x_k - x_{k-1} &\leq \omega(v_{k-1}, v_k). \end{aligned}$$

Somando todas as desigualdades, obtemos $0 \leq \omega(C)$, o que é uma contradição. Logo, se (G, ω) contém um ciclo negativo, então o sistema não tem solução viável. ∎

Resolvendo um sistema de restrições de diferença

O Teorema 24.9 nos diz que podemos usar o algoritmo de Bellman-Ford (com origem v_0) para resolver um sistema de restrições de diferença!

Como todo vértice é atingível a partir de v_0, se existir um ciclo negativo, este será detectado pelo algoritmo.

Se não existir ciclo negativo, então as distâncias computadas pelo algoritmo formam uma solução viável do sistema.

Se A é uma matriz $m \times n$, então o grafo de restrições G possui $n+1$ vértices e $n+m$ arestas. Assim, usando o algoritmo de Bellman-Ford podemos encontrar uma solução em tempo $O((n+1)(n+m)) = O(n^2+nm)$.

Caminhos mínimos entre todos os pares

O problema agora é dado um grafo (G, ω) encontrar para todo par u, v de vértices um caminho mínimo de u a v.

Obviamente podemos executar $|V|$ vezes um algoritmo de Caminhos Mínimos com Mesma Origem.

- Se (G, ω) não possui arestas negativas, então podemos usar o algoritmo de Dijkstra implementando a fila de prioridade como

  um vetor: $|V|.O(V^2) = O(V^3)$ ou
  min-heap binário: $|V|.O(E \lg V) = O(VE \lg V)$ ou
  heap de Fibonacci: $|V|.O(V \lg V + E) = O(V^2 \lg V + VE)$.

- Se (G, ω) possui arestas negativas podemos usar o algoritmo de Bellman-Ford: $|V|.O(VE) = O(V^2 E)$.

Veremos agora um método direto para resolver o problema que é assintoticamente melhor se G é denso.

O algoritmo de Floyd-Warshall baseia-se em programação dinâmica e resolve o problema em tempo $O(V^3)$.

O grafo (G, ω) pode ter arestas negativas, mas suporemos que não contém ciclos negativos.

Adotaremos a convenção de que se (i, j) não é uma aresta de G então $\omega(i, j) = \infty$.

Seja $P = (v_1, v_2, \dots, v_\ell)$ um caminho.

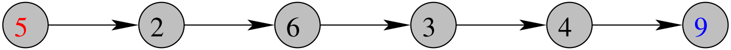
Um vértice interno de P é qualquer vértice de P distinto de v_1 e v_l, ou seja, em $\{v_2, \dots, v_{\ell-1}\}$.

Para simplificar, suponha que $V = \{1, 2, \dots, n\}$.

Estrutura de um caminho mínimo

Sejam i e j dois vértices de G. Considere todos os caminhos de i a j cujos vértices internos pertencem a $\{1, \ldots, k\}$.

Exemplo: $i = 5$, $j = 9$ e $k = 7$.



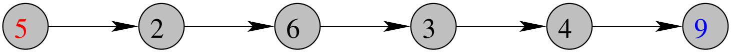
Seja P um **caminho mínimo** de i a j **com esta forma**.

Estrutura de um caminho mínimo

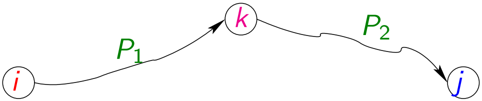
O algoritmo de Floyd-Warshall explora a relação entre P e certos caminhos mínimos com vértices internos em $\{1, \ldots, k-1\}$.

Caso 1: Se k não é um vértice interno de P então P é um caminho mínimo de i a j com vértices internos em $\{1, \ldots, k-1\}$.

Exemplo: $i = 5$, $j = 9$ e $k = 7$.



Caso 2: Se k é um vértice interno de P então P pode ser dividido em dois caminhos P_1 (com início em i e fim em k) e P_2 (com início em k e fim em j).



- P_1 é um caminho mínimo de i a k com vértices internos em $\{1, \ldots, k-1\}$
- P_2 é um caminho mínimo de k a j com vértices internos em $\{1, \ldots, k-1\}$.

Recorrência para caminhos mínimos

Seja $d_{ij}^{(k)}$ o peso de um caminho mínimo de i a j com vértices internos em $\{1, 2, \ldots, k\}$.

Quando $k = 0$ então $d_{ij}^{(0)} = \omega(i, j)$.

Temos a seguinte recorrência:

$$d_{ij}^{(k)} = \begin{cases} \omega(i,j) & \text{se } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{se } k \geq 1. \end{cases}$$

Note que $d_{ij}^{(n)} = \mathrm{dist}(i, j)$.

Podemos calcular as matrizes $D^{(k)} = (d_{ij}^{(k)})$ para $k = 1, 2 \ldots, n$.

A resposta do problema é $D^{(n)}$.

Algoritmo de Floyd-Warshall

A entrada do algoritmo é a matriz $W = (\omega(i,j))$ com $n = |V|$ linhas e colunas.

A saída é a matriz $D^{(n)}$.

$\textsc{Floyd-Warshall}(W, n)$

$D^{(0)} \leftarrow W$
para $k \leftarrow 1$ **até** n **faça**
    para $i \leftarrow 1$ **até** n **faça**
        para $j \leftarrow 1$ **até** n **faça**
            $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
devolva $D^{(n)}$

Complexidade: $O(V^3)$

Como encontrar os caminhos?

O algoritmo devolve também uma matriz $\Pi = (\pi_{ij})$ tal que (a) $\pi_{ij} = \mathrm{NIL}$ se $i = j$ ou se não existe caminho de i a j, **ou**
(b) π_{ij} é o predecessor de j em algum caminho mínimo a de i a j, caso contrário.

Podemos computar os predecessores ao mesmo tempo que o algoritmo calcula as matrizes $D^{(k)}$. Determinamos uma seqüência de matrizes $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)}$ e $\pi_{ij}^{(k)}$ é o predecessor de j em um caminho mínimo de i a j com vértices internos em $\{1, 2, \ldots, k\}$.

Quando $k = 0$ temos

$$\pi_{ij}^{(0)} = \begin{cases} \mathrm{NIL} & \text{se } i = j \text{ ou } \omega(i,j) = \infty, \\ i & \text{se } i \neq j \text{ e } \omega(i,j) < \infty. \end{cases}$$

Como encontrar os caminhos?

Para $k \geq 1$ procedemos da seguinte forma. Considere um caminho mínimo P de i a j com vértices internos em $\{1, 2, \ldots, k\}$.

Se k não aparece em P então tomamos como $\pi_{ij}^{(k)}$ o predecessor de j em um caminho mínimo de i a j com vértices internos em $\{1, 2, \ldots, k-1\}$, ou seja, $\pi_{ij}^{(k-1)}$.

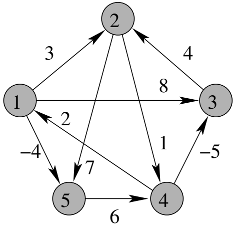
Caso contrário, tomamos como $\pi_{ij}^{(k)}$ o predecessor de j em um caminho mínimo de k a j com vértices internos em $\{1, 2, \ldots, k-1\}$, ou seja, $\pi_{kj}^{(k-1)}$.

Formalmente,

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{se } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{se } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$

Exercício. Incorpore esta parte no algoritmo!

Exemplo

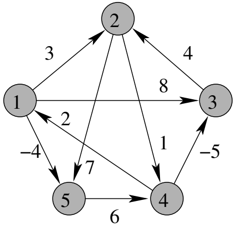


$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} N & 1 & 1 & N & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & N & N \\ 4 & N & 4 & N & N \\ N & N & N & 5 & N \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} N & 1 & 1 & N & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & N & N \\ 4 & 1 & 4 & N & 1 \\ N & N & N & 5 & N \end{pmatrix}$$

Exemplo

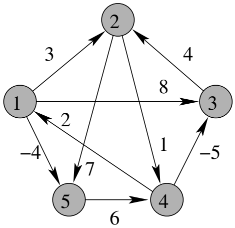


$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} N & 1 & 1 & N & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & N & N \\ 4 & 1 & 4 & N & 1 \\ N & N & N & 5 & N \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} N & 1 & 1 & 2 & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & 2 & 2 \\ 4 & 1 & 4 & N & 1 \\ N & N & N & 5 & N \end{pmatrix}$$

Exemplo

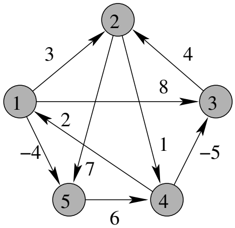


$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} N & 1 & 1 & 2 & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & 2 & 2 \\ 4 & 1 & 4 & N & 1 \\ N & N & N & 5 & N \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} N & 1 & 1 & 2 & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & 2 & 2 \\ 4 & 3 & 4 & N & 1 \\ N & N & N & 5 & N \end{pmatrix}$$

Exemplo

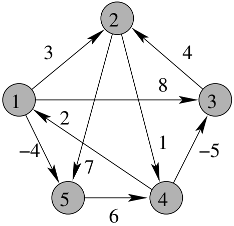


$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} N & 1 & 1 & 2 & 1 \\ N & N & N & 2 & 2 \\ N & 3 & N & 2 & 2 \\ 4 & 3 & 4 & N & 1 \\ N & N & N & 5 & N \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} N & 1 & 4 & 2 & 1 \\ 4 & N & 4 & 2 & 1 \\ 4 & 3 & N & 2 & 1 \\ 4 & 3 & 4 & N & 1 \\ 4 & 3 & 4 & 5 & N \end{pmatrix}$$

Exemplo



$$d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} N & 1 & 4 & 2 & 1 \\ 4 & N & 4 & 2 & 1 \\ 4 & 3 & N & 2 & 1 \\ 4 & 3 & 4 & N & 1 \\ 4 & 3 & 4 & 5 & N \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} N & 3 & 4 & 5 & 1 \\ 4 & N & 4 & 2 & 1 \\ 4 & 3 & N & 2 & 1 \\ 4 & 3 & 4 & N & 1 \\ 4 & 3 & 4 & 5 & N \end{pmatrix}$$

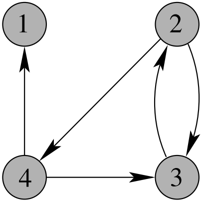
Fecho transitivo de grafos orientados

Seja $G = (V, E)$ um grafo orientado com $V = \{1, 2, \ldots, n\}$.

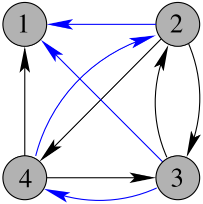
Suponha que para cada par $i, j \in V$, queremos determinar se existe um caminho de i a j em G.

O fecho transitivo de $G = (V, E)$ é o grafo $G^* = (V, E^*)$ onde

$$E^* = \{(i, j) : \text{existe um caminho de } i \text{ a } j \text{ em } G\}.$$



G



G^*

Fecho transitivo de grafos orientados

Um modo de determinar o fecho transitivo de um grafo $G = (V, E)$ em tempo $\Theta(V^3) = \Theta(n^3)$ é atribuir peso 1 a cada aresta de E e executar Floyd-Warshall. Se $d_{ij} < n$ então existe um caminho de i a j. Caso contrário, não existe tal caminho.

Há outra forma de fazer isto com mesma complexidade assintótica, mas que pode economizar tempo e espaço na prática.

A ideia é adaptar o algoritmo de Floyd-Warshall substituindo as operações aritméticas min e $+$ pelas operações lógicas $\vee$ (OR lógico) e $\wedge$ (AND lógico).

Fecho transitivo de grafos orientados

Para i, j, k em $\{1, 2, \ldots, n\}$, seja $t_{i,j}^{(k)}$ o valor lógico da expressão "existe um caminho de i a j em G com vértices internos em $\{1, 2, \ldots, k\}$".

Assim, $t_{i,j}^{(k)} = 1$ se:

1. existe um caminho de i a j em G com vértices internos em $\{1, 2, \ldots, k-1\}$, ou seja, $t_{i,j}^{(k-1)} = 1$, ou
2. existem um caminho de i a k em G com vértices internos em $\{1, 2, \ldots, k-1\}$ e um caminho de k a j em G com vértices internos em $\{1, 2, \ldots, k-1\}$, ou seja, $t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)} = 1$.

Fecho transitivo de grafos orientados

Para i, j, k em $\{1, 2, \ldots, n\}$, seja $t_{i,j}^{(k)}$ o valor lógico da expressão "existe um caminho de i a j em G com vértices internos em $\{1, 2, \ldots, k\}$".

Portanto,

$$t_{ij}^{(0)} = \begin{cases} 0 & \text{se } i \neq j \text{ e } (i,j) \notin E, \\ 1 & \text{se } i = j \text{ ou } (i,j) \in E. \end{cases}$$

e para $k \geq 1$,

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right).$$

Assim, (i,j) é uma aresta do fecho transitivo G^* se e somente se $t_{i,j}^{(n)} = 1$.

Como no Floyd-Warshall, calculamos as matrizes $T^{(k)} = (t_{ij}^{(k)})$.

A entrada do algoritmo é uma matriz de adjacência A com $n = |V|$ linhas e colunas.

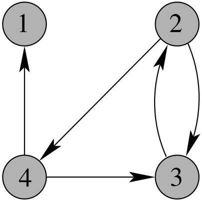
A saída é a matriz $T^{(n)}$.

TRANSITIVE-CLOSURE(A, n)

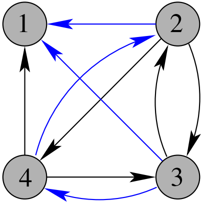
$T^{(0)} \leftarrow A + I_n$
para $k \leftarrow 1$ **até** n **faça**
  para $i \leftarrow 1$ **até** n **faça**
    para $j \leftarrow 1$ **até** n **faça**
      $t_{ij}^{(k)} \leftarrow t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right)$
devolva $T^{(n)}$

Complexidade: $O(V^3)$

Exemplo



G

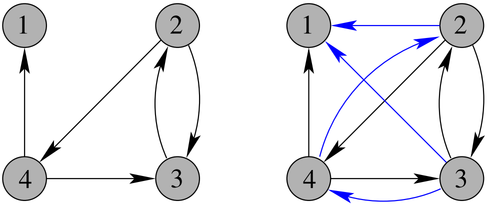


G^*

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right).$$

$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

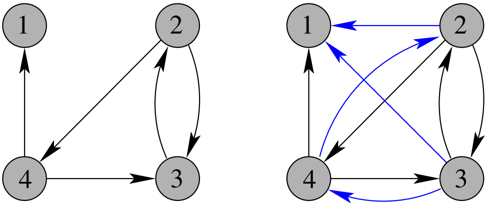
Exemplo



G $\qquad\qquad$ G^*

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right).$$

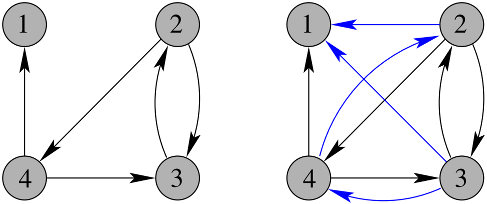
$$T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$



G $\qquad\qquad$ G^*

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right).$$

$$T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$



G $\qquad$ G^*

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee \left(t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}\right).$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Note que poderíamos usar um bit para cada posição das matrizes $T^{(k)}$ e as operações lógicas $\vee$ e $\wedge$ podem ser implementadas com operadores de bits.

Isto reduz consideravelmente o espaço do ponto de vista prático e pode acelerar o algoritmo se a implementação de operações lógicas for eficiente. Note que isto não afeta a análise assintótica.