# MC448 — Análise de Algoritmos I

Cid Carvalho de Souza     Cândida Nunes da Silva
Orlando Lee

30 de novembro de 2009



## Problema do(s) Caminho(s) Mínimo(s)

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

- Problema do Caminho Mínimo entre Dois Vértices: Dados dois vértices $s$ e $t$ em $(G, w)$, 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 $(G, w)$ e $s \in V[G]$, encontrar para cada vértice $v$ de $G$, um caminho mínimo de $s$ a $v$.



## Subsestrutura ótima de caminhos mínimos

Teorema. Seja $(G, w)$ um grafo orientado e seja

$$P = (v_1, v_2, \dots, v_l)$$

um caminho mínimo de $v_1$ a $v_k$.

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

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

é um caminho mínimo de $v_i$ a $v_j$.



## Representação de caminhos mínimos

- 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 $(G, w)$.

## Estimativa de distâncias

- Para cada $v \in V[G]$ queremos determinar $dist(s, v)$, o peso de um caminho mínimo de $s$ a $v$ em $(G, w)$ (distância de $s$ a $v$.)
- Os algoritmos de caminhos mínimos associam a cada $v \in V[G]$ um valor $d[v]$ que é uma estimativa da distância $dist(s, v)$.



## 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 para o peso de um caminho mínimo de $s$ a $v$.

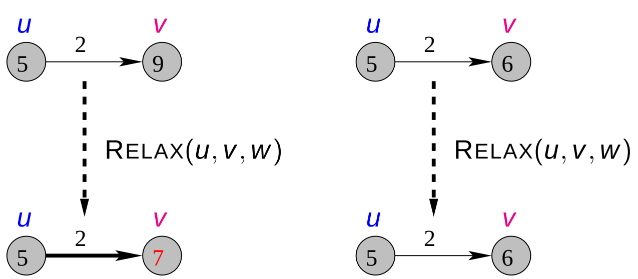
Ele indica que o algoritmo encontrou até aquele momento um caminho de $s$ a $v$ com peso $d[v]$.

O caminho pode ser recuperado por meio dos predecessores $\pi[\,]$.



## Relaxação

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

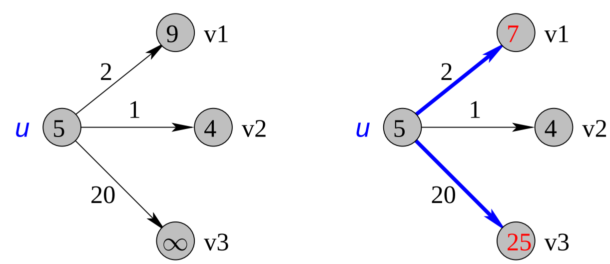



RELAX$(u, v, w)$
**se** $d[v] > d[u] + w(u, v)$
  **então** $d[v] \leftarrow d[u] + w(u, v)$
    $\pi[v] \leftarrow u$



## Relaxação dos vizinhos

Em cada iteração o algoritmo seleciona um vértice $u$ e para cada vizinho $v$ de $u$ aplica RELAX$(u, v, w)$.




RELAX$(u, v, w)$
**se** $d[v] > d[u] + w(u, v)$ **faça**
  $d[v] \leftarrow d[u] + w(u, v)$
  $\pi[v] \leftarrow u$

## Caminhos Mínimos

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

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



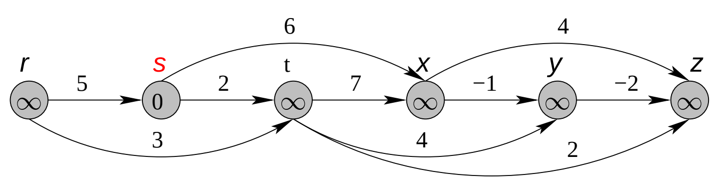
## Caminhos mínimos em grafos acíclicos

Entrada: grafo orientado acíclico $G = (V, E)$ com função peso $w$ nas arestas e uma origem $s$.
Saída: vetor $d[v] = dist(s, v)$ para $v \in V$
e uma Árvore de Caminhos Mínimos definida por $\pi[\,]$.

```
DAG-SHORTEST-PATHS(G, w, s)
Ordene topologicamente os vértices de G
INITIALIZE-SINGLE-SOURCE(G, s)
para cada vértice u na ordem topológica faça
    para cada v ∈ Adj[u] faça
        RELAX(u, v, w)
```

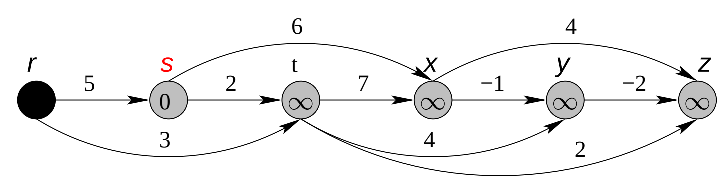


## Exemplo

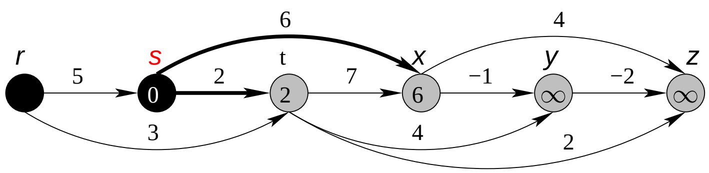





## Exemplo

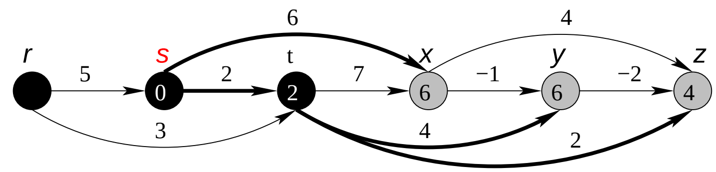


# Exemplo



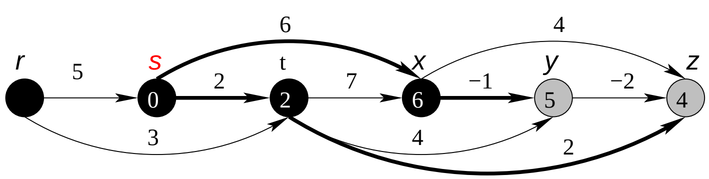



# Exemplo



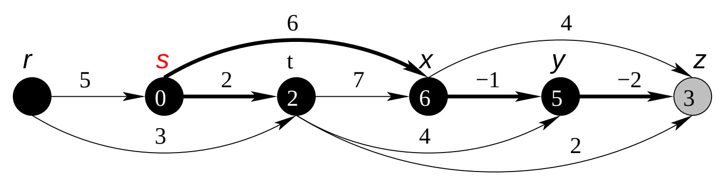



# Exemplo

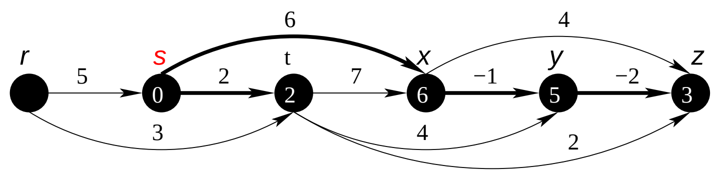





# Exemplo



## Exemplo






## Complexidade

DAG-SHORTEST-PATHS$(G, w, s)$
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$(u, v, w)$

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

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



## Corretude

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.



## Algoritmos baseados em relaxação

As propriedades que veremos são para algoritmos que satisfazem as restrições abaixo.

- O algoritmo é inicializado com INITIALIZE-SINGLE-SOURCE$(G, s)$.
- Um valor $d[v]$ (e $\pi[v]$) só pode ser modificado (reduzido) através de uma chamada de RELAX$(u, v, w)$ para alguma aresta $(u, v)$.

## Algoritmos baseados em relaxação

Ao longo do “algoritmo” as seguintes propriedades sempre valem:

- (Lema 24.11, CLRS) $d[v] \geq dist(s, v)$ para $v \in V$ e quando $d[v]$ fica igual a $dist(s, v)$, seu valor nunca mais muda.
- (Lema 24.12, CLRS) Se não existe caminho de $s$ a $v$, então temos sempre que $d[v] = dist(s, v) = \infty$.



## Algoritmos baseados em relaxação — continuação

- (Lema 24.15, CLRS) Seja $P$ um caminho mínimo de $s$ a $v$ cuja última aresta é $(u, v)$ e suponha que $d[u] = dist(s, u)$ antes de uma chamada $\text{RELAX}(u, v, w)$. Então após a chamada, $d[v] = dist(s, v)$.
- (Lema 24.16, CLRS) Seja $P = (v_0 = s, v_1, \ldots, v_k)$ um caminho mínimo de $v_0$ a $v_k$ e suponha que as arestas $(v_0, v_1)$, $(v_1, v_2)$,...,$(v_{k-1}, v_k)$ são relaxadas nesta ordem. Então $d[v_k] = dist(s, v_k)$.



## Corretude de DAG-SHORTEST-PATHS

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 $(v_0, v_1)$, $(v_1, v_2)$,...,$(v_{k-1}, v_k)$.

Logo, pelo Lema 24.16 o algoritmo computa corretamente $d[v] = dist(s, v)$ para todo $v \in V$.

Também é fácil ver que $\pi[\,]$ define uma Árvore de Caminhos Mínimos.



## Algoritmo de Dijkstra

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

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

## Algoritmo de Dijkstra

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

e devolve

- para cada $v \in V[G]$, 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 é um caminho mínimo de $s$ a $v$ em $(G, w)$.



## Algoritmo de Dijkstra

```
DIJKSTRA(G, w, 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, w)
```

O conjunto $Q$ é implementado como uma fila de prioridade.

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



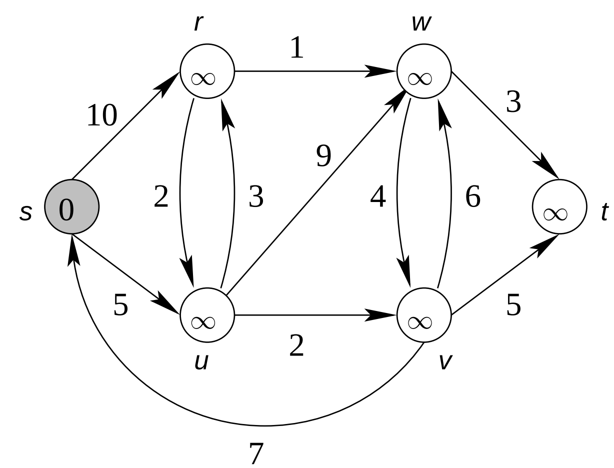
## Intuição do algoritmo

Em cada iteração, o algoritmo de DIJKSTRA

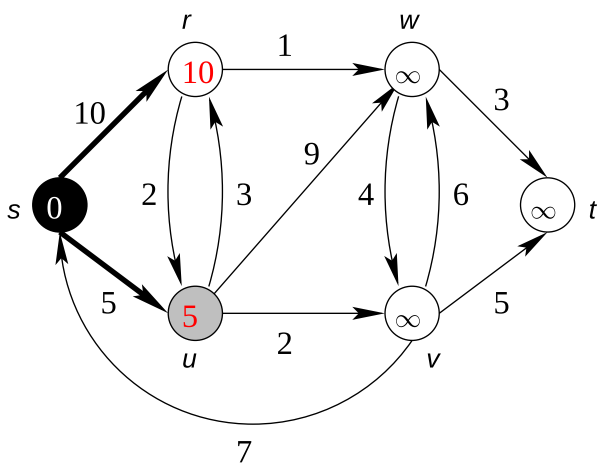
- escolhe um vértice $u$ fora do conjunto $S$ que esteja mais próximo a esse e acrescenta-o a $S$,
- atualiza as distâncias estimadas dos vizinhos de $u$ e
- atualiza a Árvore dos Caminhos Mínimos.



## Exemplo (CLRS modificado)

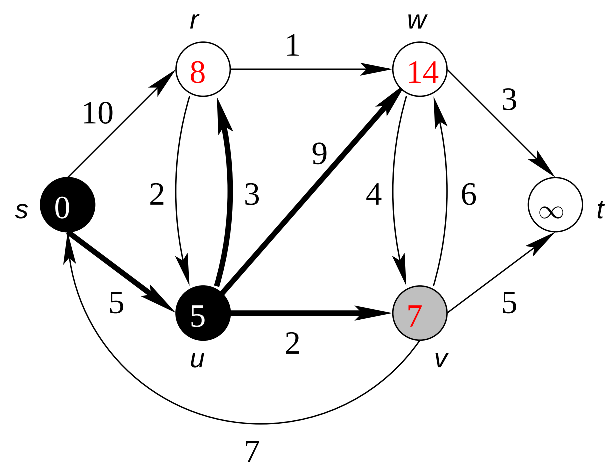


# Exemplo (CLRS modificado)



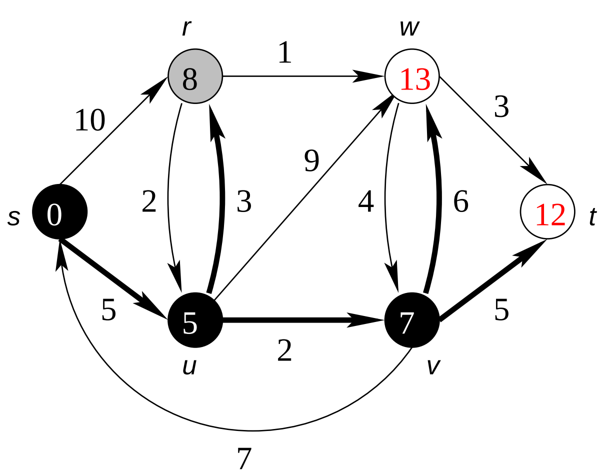



# Exemplo (CLRS modificado)



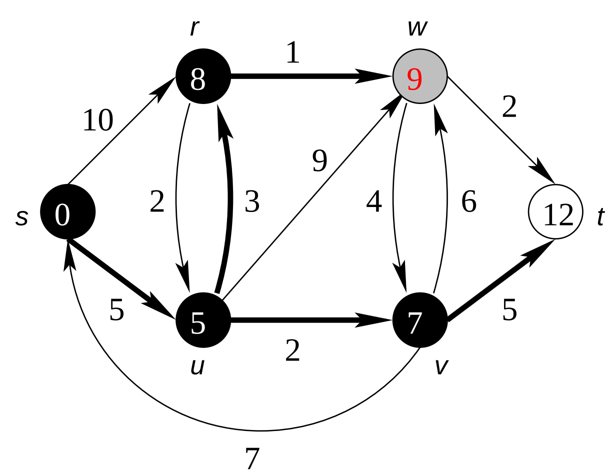



# Exemplo (CLRS modificado)

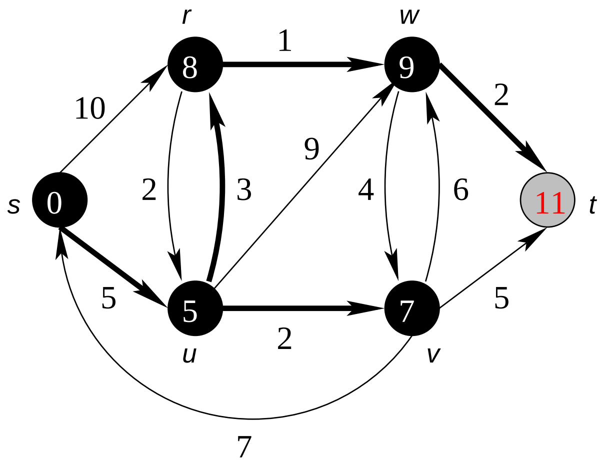





# Exemplo (CLRS modificado)

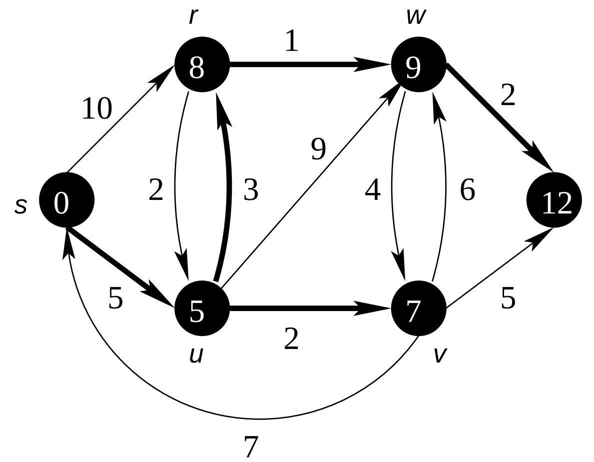


## Exemplo (CLRS modificado)






## Exemplo (CLRS modificado)






## Correção do algoritmo

Precisamos provar que quando o algoritmo pára, temos que

- $d[v] = dist(s, v)$ para todo $v \in V[G]$ e
- $\pi[\,]$ define uma Árvore de Caminhos Mínimos.
  Mais precisamente, o conjunto

  $\{(\pi[x], x) : x \in V[G] - \{s\}\}$

  forma uma Árvore de Caminhos Mínimos.



## Alguns invariantes simples

Os seguintes invariantes valem em cada iteração do algoritmo DIJKSTRA (linha 4).

- Se $d[x] < \infty$ então $\pi[x] \in S$.
- Para cada vértice $x$ em $S$, a seqüência

  $(s, \ldots, \pi[\pi[x]], \pi[x], x)$

  é um caminho de $s$ a $x$ com peso $d[x]$.

## Outros invariantes simples

- $dist(s,x) \leq d[x]$ para cada vértice $x$ em $S$.
- O conjunto $\{(\pi[x],x) \,:\, x \in S - \{s\}\}$ forma uma árvore.
- Se $x \in S$ e $y \in V[G] - S$ então $d[y] \leq d[x] + w(x,y)$.

  Isto vale pois quando $x$ foi inserido em $S$, foi executado RELAX$(x,y,w)$.



## Invariante principal

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

Invariante: $d[x] = dist(s,x)$ 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

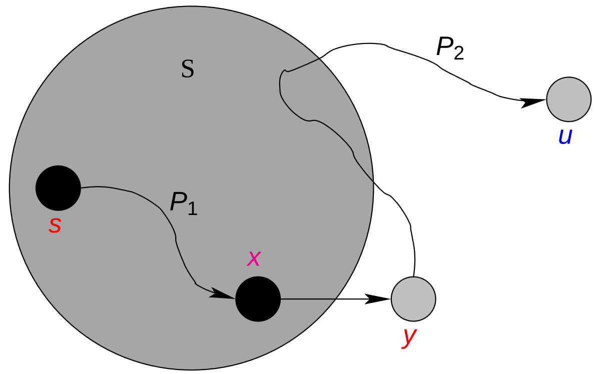
- O algoritmo de DIJKSTRA escolhe um vértice $u$ com menor $d[u]$ em $Q$ e atualiza $S \leftarrow S \cup \{u\}$.
- Basta verificar então que neste momento $d[u] = dist(s,u)$.

Suponha que $d[u] > dist(s,u)$ por contradição.



## Demonstração

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



## Demonstração

- $dist(s,u) < d[u]$ (por hipótese).
- $d[x] = dist(s,x)$ pois $x \in S$.
- Então

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

- Mas então $d[y] < d[u]$ o que contraria a escolha de $u$.
  Logo, $d[u] = dist(s,u)$ e o invariante vale.



## Dijkstra precisa de arestas com peso não-negativo

Note que 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 esta hipótese não for válida.

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, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← ∅
3  Q ← V[G]
4  enquanto Q ≠ ∅ faça
5      u ← EXTRACT-MIN(Q)
6      S ← S ∪ {u}
7      para cada vértice v ∈ Adj[u] faça
8          RELAX(u, v, w)
```

Depende de como a fila de prioridade $Q$ é implementada.



## Complexidade de tempo

A fila de prioridade $Q$ é mantida com as seguintes operações:

- INSERT (implícito na linha 3)
- EXTRACT-MIN (linha 5) e
- DECREASE-KEY (implícito em RELAX na linha 8).

São executados (no máximo) $|V|$ operações EXTRACT-MIN.

Cada vértice $u \in V[G]$ é inserido em $S$ (no máximo) uma vez e cada aresta $(u,v)$ com $v \in \mathrm{Adj}[u]$ é examinada (no máximo) uma vez nas linhas 7-8 durante todo o algoritmo. Assim, são executados no máximo $|E|$ operações DECREASE-KEY.

## Complexidade de tempo

No total temos $|V|$ chamadas a EXTRACT-MIN e $|E|$ chamadas a 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 DECREASE-KEY é $O(1)$) a complexidade reduz para $O(V \lg V + E)$.



## Arestas/ciclos de peso negativo

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



## Ciclos negativos — uma dificuldade

- Se um ciclo negativo $C$ é atingível a partir da fonte $s$, em princípio o problema não tem solução pois o "caminho" pode passar ao longo do ciclo infinitas vezes obtendo caminhos cada vez menores.
- Naturalmente, podemos impor a restrição de que os caminhos tem que ser simples, sem repetição de vértices. Entretanto, esta versão do problema é NP-difícil.
- Assim, vamos nos restringir ao Problema de Caminhos Mínimos sem ciclos negativos.



## O algoritmo de Bellman-Ford

O algoritmo de Bellman-Ford recebe um grafo orientado $(G, w)$ (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

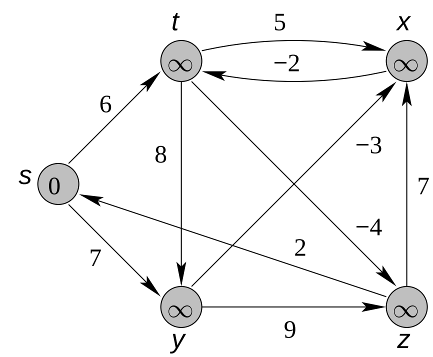
BELLMAN-FORD$(G, w, s)$

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

Complexidade de tempo: $O(VE)$



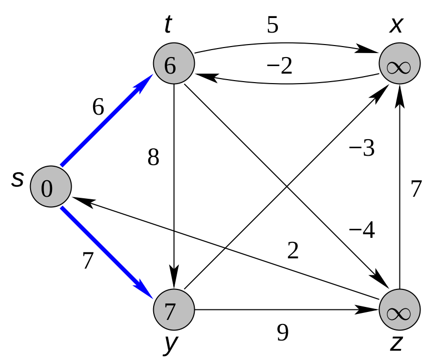
## Exemplo (CLRS)




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



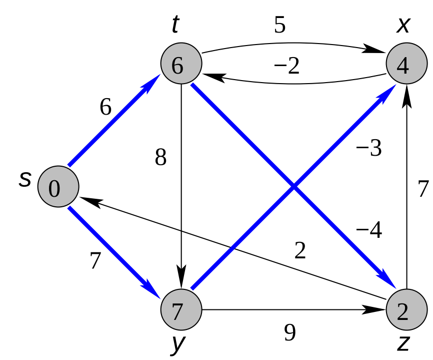
## Exemplo (CLRS)




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

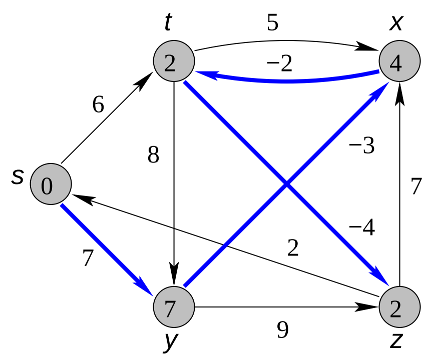


## Exemplo (CLRS)




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

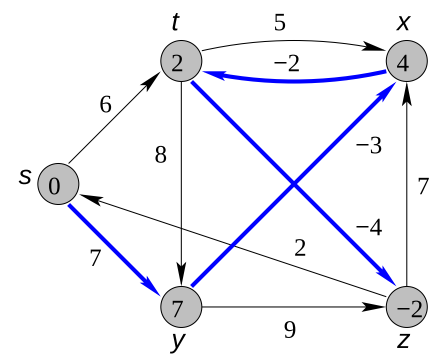
## Exemplo (CLRS)




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



## Exemplo (CLRS)




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



## Corretude do algoritmo

Teorema 24.4 (CLRS).
Se $(G,w)$ não contém ciclos negativos atingíveis por $s$, então no final

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

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



## Corretude do algoritmo

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 $v_0$ a $v_k$ e suponha que as arestas $(v_0,v_1)$, $(v_1,v_2)$,$\ldots$,$(v_{k-1},v_k)$ são relaxadas nesta ordem. Então $d[v_k] = dist(s,v_k)$.

## Corretude do algoritmo

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, $d[v] = d[v_k] = dist(s, v)$.



## Corretude do algoritmo

Se $v$ é um vértice não atingível por $s$ pode-se mostrar que $d[v] = \infty$ no final (Exercício).

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

Pode-se verificar que $\pi[\,]$ define uma Árvore de Caminhos Mínimos (veja CLRS para detalhes).



## Corretude do algoritmo

Agora falta mostrar que Bellman-Ford devolve TRUE.

Seja $(u, v)$ uma aresta de $G$. Então

$$\begin{aligned} d[v] &= dist(s, v) \\ &\leq dist(s, u) + w(u, v) \\ &= d[u] + w(u, v), \end{aligned}$$

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



## Corretude do algoritmo

Agora suponha que $(G, w)$ contém um ciclo negativo atingível por $s$.

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

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

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

## Corretude do algoritmo

Somando as desigualdades ao longo do ciclo temos

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

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

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



## Caminhos mínimos entre todos os pares

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

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

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



## O algoritmo de Floyd-Warshall

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 $(G, w)$ pode ter arestas negativas, mas suporemos que não contém ciclos negativos.

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



## Estrutura de um caminho mínimo

Seja $P = (v_1, v_2, \ldots, v_l)$ um caminho (simples).

Um vértice intermediário de $P$ é qualquer vértice de $P$ distinto de $v_1$ e $v_l$, ou seja, em $\{v_2, \ldots, v_{k-1}\}$.

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

## Estrutura de um caminho mínimo

Sejam $i$ e $j$ dois vértices de $G$. Considere todos os caminhos de $i$ a $j$ cujos vértices intermediários pertencem a $\{1, \ldots, k\}$. Seja $P$ um caminho mínimo entre todos eles.

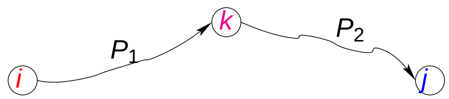
O algoritmo de Floyd-Warshall explora a relação entre $P$ e um caminho mínimo de $i$ a $j$ com vértices intermediários em $\{1, \ldots, k-1\}$.

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



## Estrutura de um caminho mínimo

- Se $k$ é um vértice intermediário 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 intermediários em $\{1, \ldots, k-1\}$
- $P_2$ é um caminho mínimo de $k$ a $j$ com vértices intermediários 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 intermediários em $\{1, 2, \ldots, k\}$.

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

Temos a seguinte recorrência:

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

Assim, queremos calcular a matrix $D^{(n)} = (d_{ij}^{(n)})$ com $d_{ij}^{(n)} = dist(i,j)$.



## Algoritmo de Floyd-Warshall

A entrada do algoritmo é a matriz $W = (w(i,j))$ com $n = |V|$ linhas e colunas.
A saída é a matriz $D^{(n)}$.

FLOYD-WARSHALL($W$)
$D^{(0)} \leftarrow W$
**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 precisa devolver também uma matriz $\Pi = (\pi_{ij})$ tal que $\pi_{ij} = \text{NIL}$ se $i = j$ ou se não existe caminho de $i$ a $j$, e caso contrário, $\pi_{ij}$ é o predecessor de de $j$ em algum caminho mínimo a partir de $i$.

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

Quando $k = 0$ temos

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



## Como encontrar os caminhos?

Para $k \geq 1$ procedemos da seguinte forma. Considere um caminho mínimo $P$ de $i$ a $j$.

Se $k$ não aparece em $P$ então tomamos como predecessor de $j$ o predecessor de $j$ em um caminho mínimo de $i$ a $j$ com vértices intermediários em $\{1, 2, \dots, k-1\}$.

Caso contrário, tomamos como predecessor de $j$ o predecessor de $j$ em um caminho mínimo de $k$ a $j$ com vértices intermediários em $\{1, 2, \dots, k-1\}$.
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!

Cid Carvalho de Souza, Cândida Nunes da Silva, Orlando Lee MC448 — Análise de Algoritmos