Projeto e Análise de Algoritmos

Caminhos mínimos

Cid Carvalho de Souza, Cândida Nunes da Silva et al.

Primeiro Semestre de 2017

Caminhos mínimos com uma origem

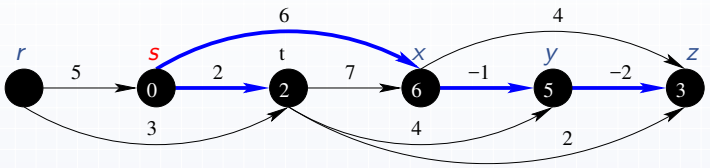
Problema do(s) Caminho(s) Mínimo(s)

Seja G um grafo direcionado 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

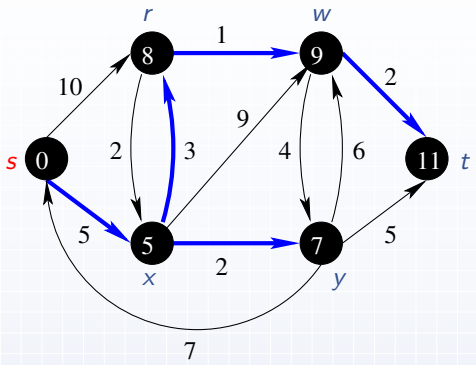
  Problema dos Caminhos Mínimos com Mesma Origem:
  Dados (G, ω) e $s \in V[G]$, encontrar para cada vértice v de G, um caminho mínimo de s a v.

Exemplo: grafo direcionado acíclico



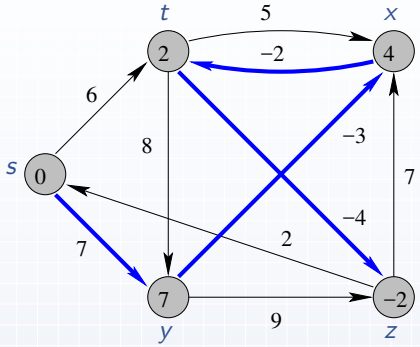
v	s	r	t	x	y	z
$\text{dist}(s, v)$	0	∞	2	6	5	3

Exemplo: grafo direcionado sem arestas negativas



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

Exemplo: grafo direcionado com arestas negativas



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

Ideias comuns a todos os algoritmos

- Usaremos uma ideia 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).

Estimativa de distâncias

- Para cada $v \in V[G]$ queremos determinar $\text{dist}(s, v)$, o peso de um caminho mínimo de s a v em (G, ω) (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 $\text{dist}(s, v)$.

Subestrutura ótima de caminhos mínimos

Teorema. Seja (G, ω) um grafo direcionado sem ciclos negativos e seja

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

um caminho mínimo de v_1 a v_k.

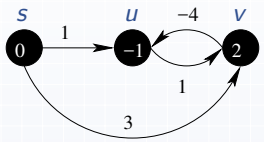
Então para quaisquer i, j com $1 \leq i \leq j \leq k$

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

é um caminho mínimo de v_i a v_j.

Subestrutura ótima de caminhos mínimos

A subestrutura ótima **não** vale se (G, ω) contém ciclos negativos.



O caminho mínimo de s a v é (s, u, v) com peso $1 + 1 = 2$.

Porém, (s, u) não é um caminho mínimo de s a u.

O caminho mínimo de s a u é (s, v, u) com peso $3 - 4 = -1$.

Inicialização

INITIALIZE-SINGLE-SOURCE(G, s)
1 **para cada** vértice $v \in V[G]$ **faça**
2 $\quad d[v] \leftarrow \infty$
3 $\quad \pi[v] \leftarrow$ NIL
4 $d[s] \leftarrow 0$

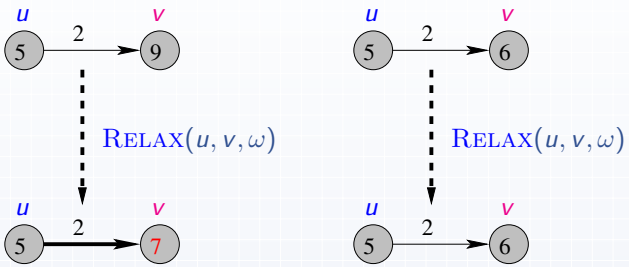
O valor $d[v]$ é uma estimativa superior para o peso de um caminho mínimo de s a v.

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

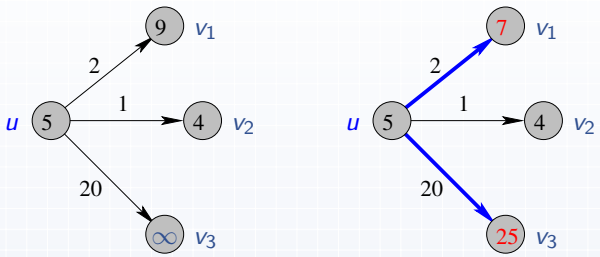
Tenta melhorar a estimativa $d[v]$ examinando (u, v).



RELAX(u, v, ω)
1 **se** $d[v] > d[u] + \omega(u, v)$ **então**
2 $\quad d[v] \leftarrow d[u] + \omega(u, v)$
3 $\quad \pi[v] \leftarrow u$

Relaxação dos vizinhos

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



$\textsc{Relax}(u, v, \omega)$
1 **se** $d[v] > d[u] + \omega(u, v)$ **então**
2 $\quad d[v] \leftarrow d[u] + \omega(u, v)$
3 $\quad \pi[v] \leftarrow u$

Caminhos Mínimos

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

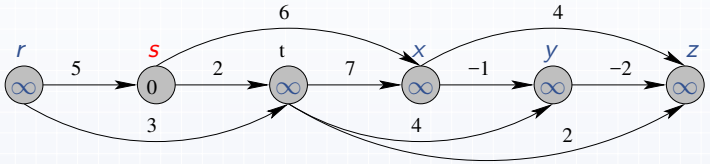
- G é acíclico: aplicação de ordenação topológica
- (G, ω) não tem arestas de peso negativo: algoritmo de Dijkstra
- (G, ω) pode ter arestas de peso negativo, mas não contém ciclos negativos: algoritmo de Bellman-Ford.

Caminhos mínimos em grafos acíclicos

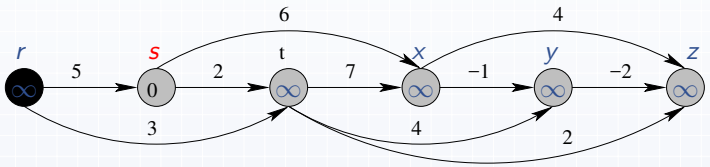
Entrada: grafo direcionado acíclico $G = (V, E)$ com função peso ω nas arestas e uma origem s.
Saída: vetor $d[v] = \text{dist}(s, v)$ para $v \in V$
e uma Árvore de Caminhos Mínimos definida por $\pi[\,]$.

$\textsc{Dag-Shortest-Paths}(G, \omega, s)$
1 Ordene topologicamente os vértices de G
2 $\textsc{Initialize-Single-Source}(G, s)$
3 **para cada** vértice u na ordem topológica **faça**
4 $\quad$ **para cada** $v \in \text{Adj}[u]$ **faça**
5 $\quad\quad \textsc{Relax}(u, v, \omega)$
6 **devolva** d, π

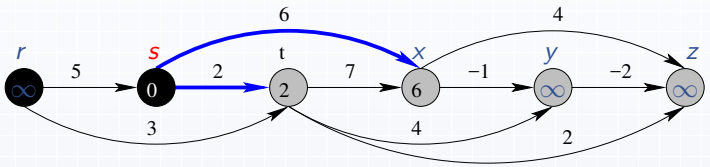
Exemplo



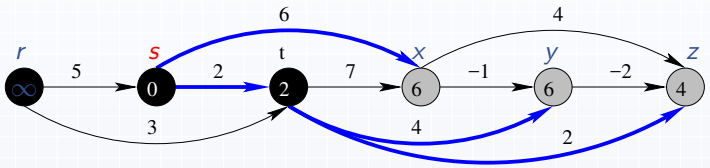
Exemplo



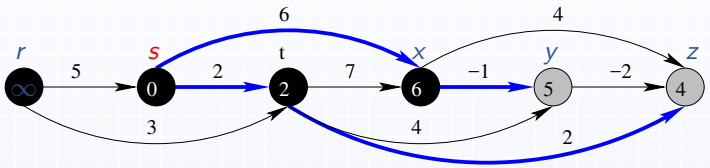
Exemplo



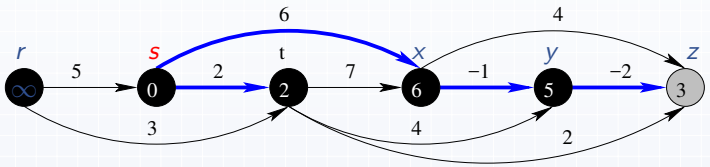
Exemplo



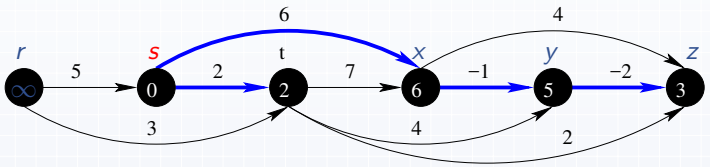
Exemplo



Exemplo



Exemplo



Complexidade

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** $v \in \text{Adj}[u]$ **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: $O(V+E)$

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 através de uma chamada de RELAX(u, v, ω) para alguma aresta (u, v).

Algoritmos baseados em relaxação

Ao longo do "algoritmo" as seguintes propriedades sempre valem:

- (Lema 24.11, CLRS) $d[v] \geq \text{dist}(s, v)$ para $v \in V$ e $d[v]$ nunca aumenta. Além disso, se $d[v]$ fica igual a $\text{dist}(s, v)$, então seu valor nunca mais muda.
- (Corolário 24.12, CLRS) Se não existe caminho de s a v, então temos que $d[v] = \text{dist}(s, v) = \infty$ em qualquer iteração.

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

Corretude de DAG-SHORTEST-PATHS

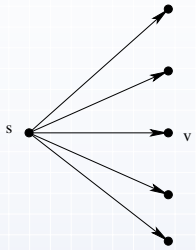
Como os vértices estão em ordem topológica, as arestas de qualquer caminho mínimo $P = (v_1 = s, v_2, \ldots, v_k)$ são relaxadas na ordem $(v_1, v_2), \ldots, (v_{k-1}, v_k)$.

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

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

Corretude de DAG-SHORTEST-PATHS

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



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

Corretude de 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

A correção de DAG-SHORTEST-PATHS segue por indução e da validade 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 direcionado acíclico (G, ω)?

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

Algoritmo de Dijkstra

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 direcionado (G, ω) (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 ($d[v] = \text{dist}(s, v)$)
- 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, ω).

Revisão: algoritmos baseados em relaxação

INITIALIZE-SINGLE-SOURCE(G, s)
1 **para cada** vértice $v \in V[G]$ **faça**
2 $\quad d[v] \leftarrow \infty$
3 $\quad \pi[v] \leftarrow$ NIL
4 $d[s] \leftarrow 0$

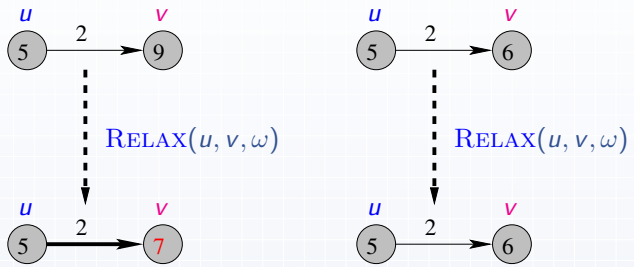
O valor $d[v]$ é uma estimativa superior para o peso de um caminho mínimo de s a v.

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 $\pi[\,]$.

Revisão: relaxação

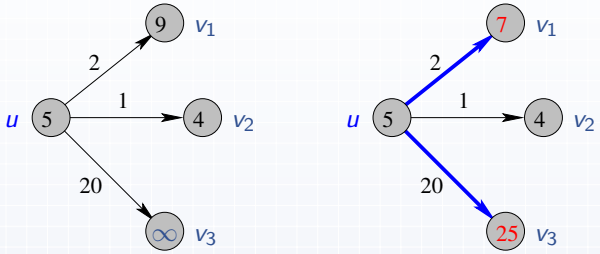
Tenta melhorar a estimativa $d[v]$ examinando (u, v).



RELAX(u, v, ω)
1 **se** $d[v] > d[u] + \omega(u, v)$ **então**
2 $\quad d[v] \leftarrow d[u] + \omega(u, v)$
3 $\quad \pi[v] \leftarrow u$

Revisão: relaxação dos vizinhos

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



$\textsc{Relax}(u, v, \omega)$
1 **se** $d[v] > d[u] + \omega(u, v)$ **então**
2 $\quad d[v] \leftarrow d[u] + \omega(u, v)$
3 $\quad \pi[v] \leftarrow u$

Revisão: algoritmos baseados em relaxação

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

- O algoritmo é inicializado com $\textsc{Initialize-Single-Source}(G, s)$.
- Um valor $d[v]$ (e $\pi[v]$) só pode ser modificado através de uma chamada de $\textsc{Relax}(u, v, \omega)$ para alguma aresta (u, v).

Revisão: algoritmos baseados em relaxação

Ao longo do "algoritmo" as seguintes propriedades sempre valem:

- (Lema 24.11, CLRS) $d[v] \geq \text{dist}(s, v)$ para $v \in V$ e $d[v]$ nunca aumenta. Além disso, se $d[v]$ fica igual a $\text{dist}(s, v)$, então seu valor nunca mais muda.
- (Corolário 24.12, CLRS) Se não existe caminho de s a v, então temos que $d[v] = \text{dist}(s, v) = \infty$ em qualquer iteração.

Ideia do algoritmo de Dijkstra

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

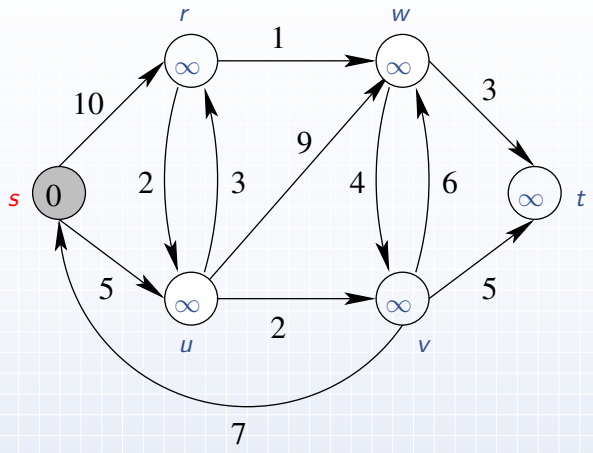
Algoritmo de Dijkstra

```
DIJKSTRA(G, ω, 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, ω)
9  devolva d, π
```

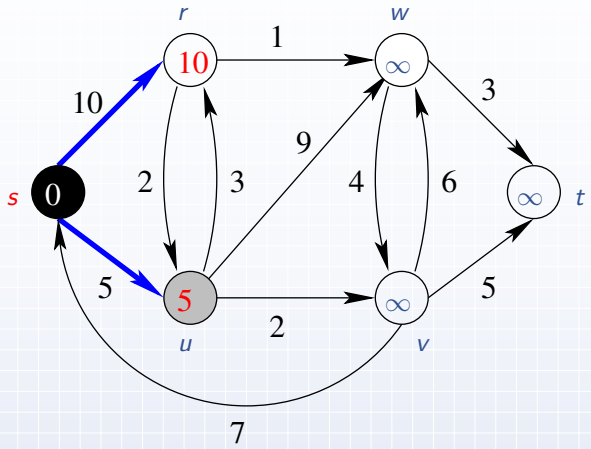
O conjunto Q é implementado como uma fila de prioridade com chave d.

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

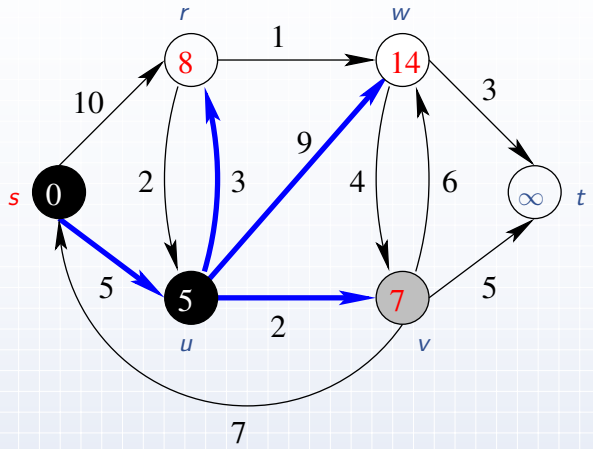
Exemplo (CLRS modificado)



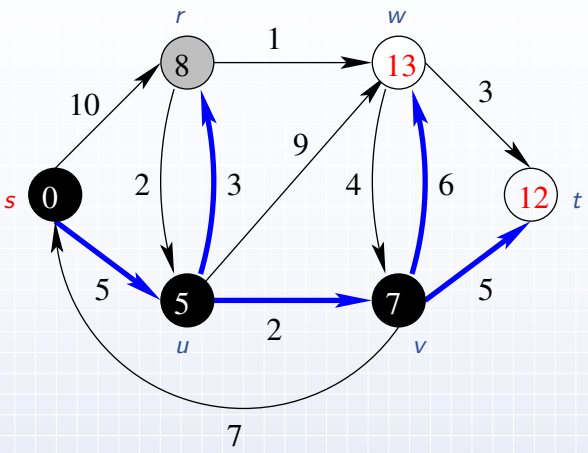
Exemplo (CLRS modificado)



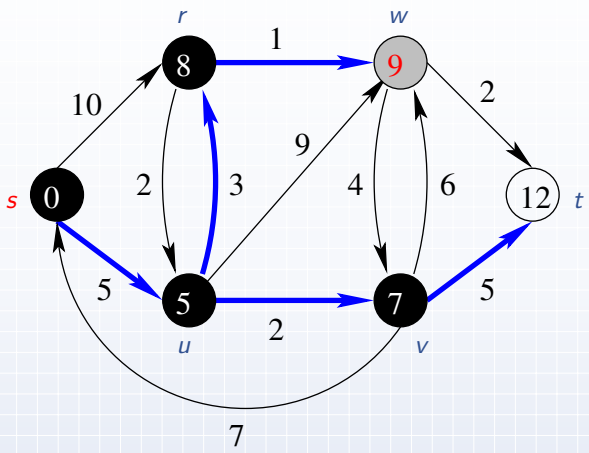
Exemplo (CLRS modificado)



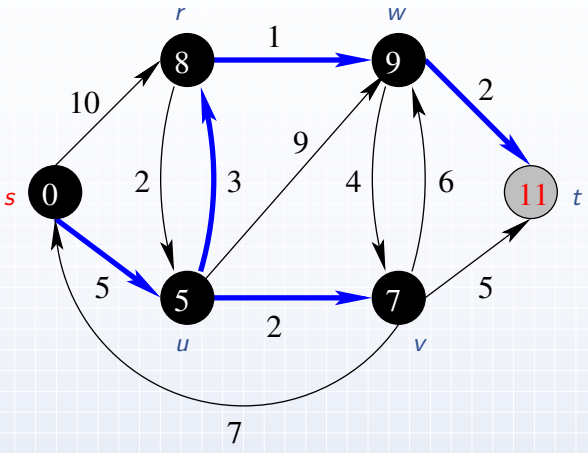
Exemplo (CLRS modificado)



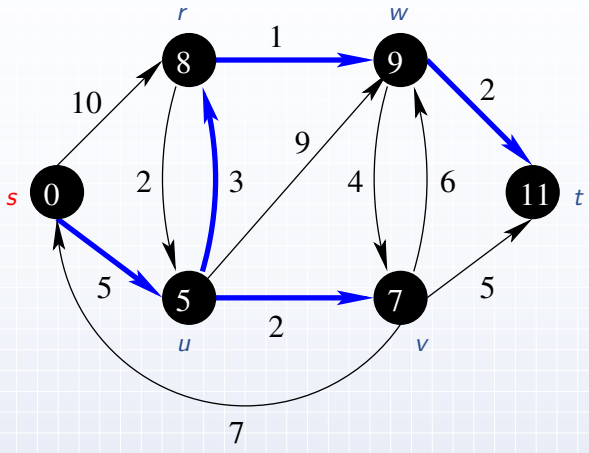
Exemplo (CLRS modificado)



Exemplo (CLRS modificado)



Exemplo (CLRS modificado)



Correção do algoritmo

Precisamos provar que quando o algoritmo para, temos que

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

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

  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 $\pi[x] \in S$.
- O conjunto

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

  induz uma árvore com conjunto de vértices S.
- Para cada vértice x em S, o caminho de s a x na árvore tem peso $d[x]$.

Outros invariantes simples

- $\text{dist}(s, x) \leq d[x]$ para cada vértice x em S.
- Se $(x, y) \in E[G]$, $x \in S$ e $y \in V[G] - S$ então

  $$d[y] \leq d[x] + \omega(x, y).$$

  Isto vale pois quando x foi inserido em S, foi executado RELAX(x, y, ω) e depois disso, $d[x]$ nunca muda e $d[y]$ nunca aumenta.

Invariante principal

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

Invariante: $d[x] = \text{dist}(s, x)$ 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.

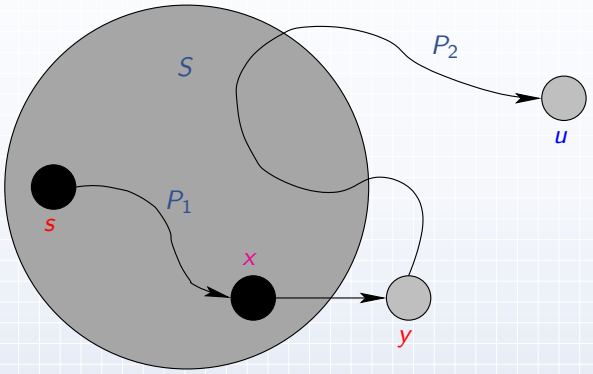
Demonstração do invariante

- 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 $d[u] = \text{dist}(s, u)$.

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

Demonstração

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



Demonstração

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

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

- Mas então $d[y] < d[u]$ o que contraria a escolha de u. Logo, $d[u] = \text{dist}(s, u)$ 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 direcionado 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)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 $S \leftarrow \emptyset$
3 $Q \leftarrow V[G]$
4 **enquanto** $Q \neq \emptyset$ **faça**
5 $u \leftarrow$ EXTRACT-MIN(Q)
6 $S \leftarrow S \cup \{u\}$
7 **para cada** vértice $v \in \text{Adj}[u]$ **faça**
8 RELAX(u, v, ω)
9 **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 de tempo

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 $O(V \lg V + E)$.

Algoritmo de Bellman-Ford

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 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

- 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.
- Assim, vamos nos restringir ao Problema de Caminhos Mínimos sem ciclos negativos.

Revisão: inicialização

INITIALIZE-SINGLE-SOURCE(G, s)
1 **para cada** vértice $v \in V[G]$ **faça**
2 $\quad d[v] \leftarrow \infty$
3 $\quad \pi[v] \leftarrow$ NIL
4 $d[s] \leftarrow 0$

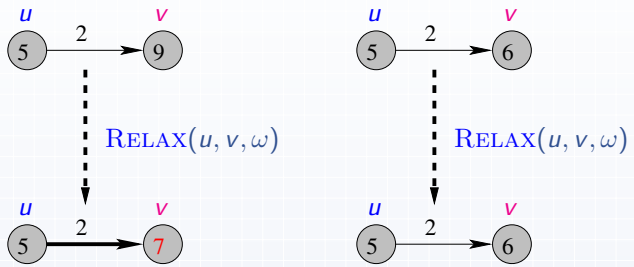
O valor $d[v]$ é uma estimativa superior para o peso de um caminho mínimo de s a v.

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 $\pi[\,]$.

Revisão: relaxação

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



RELAX(u, v, ω)
1 **se** $d[v] > d[u] + \omega(u, v)$ **então**
2 $\quad d[v] \leftarrow d[u] + \omega(u, v)$
3 $\quad \pi[v] \leftarrow u$

Revisão: algoritmos baseados em relaxação

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

- O algoritmo é inicializado com $\textsc{Initialize-Single-Source}(G, s)$.
- Um valor $d[v]$ (e $\pi[v]$) só pode ser modificado através de uma chamada de $\textsc{Relax}(u, v, \omega)$ para alguma aresta (u, v).

Revisão: algoritmos baseados em relaxação

Ao longo do "algoritmo" as seguintes propriedades sempre valem:

- (Lema 24.11, CLRS) $d[v] \geq \text{dist}(s, v)$ para $v \in V$ e $d[v]$ nunca aumenta. Além disso, se $d[v]$ fica igual a $\text{dist}(s, v)$, então seu valor nunca mais muda.
- (Corolário 24.12, CLRS) Se não existe caminho de s a v, então temos que $d[v] = \text{dist}(s, v) = \infty$ em qualquer iteração.

Revisão: algoritmos baseados em relaxação

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

Ideia do algoritmo de Bellman-Ford

- 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 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 direcionado (G, ω) (possivelmente com arestas de peso negativo) e um vértice origem s de G.

Ele devolve um valor booleano

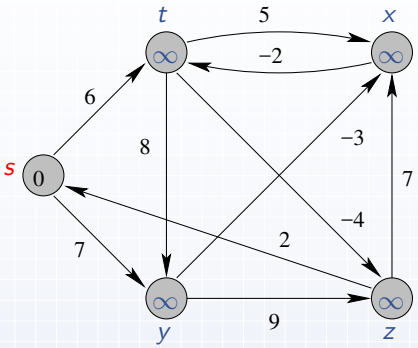
- FALSE se existe um ciclo negativo atingível a partir de s, ou
- TRUE e neste caso devolve também uma Árvore de Caminhos Mínimos com raiz s.

O algoritmo de Bellman-Ford

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

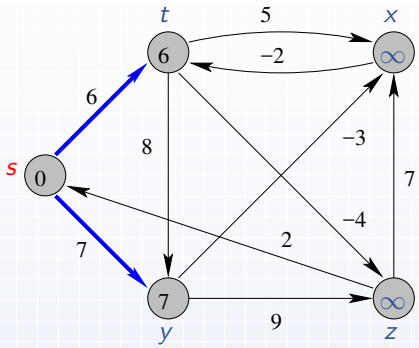
Complexidade de tempo: $O(VE)$

Exemplo (CLRS)



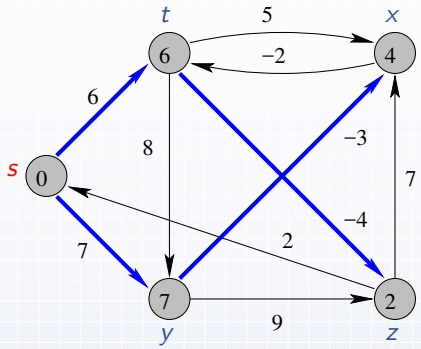
Ordem:
$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

Exemplo (CLRS)



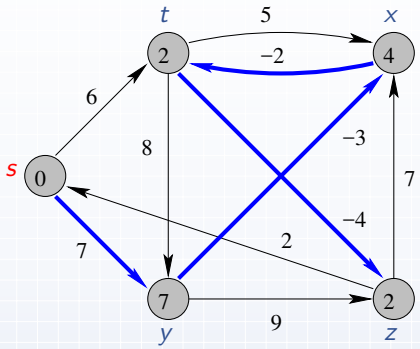
Ordem:
$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

Exemplo (CLRS)



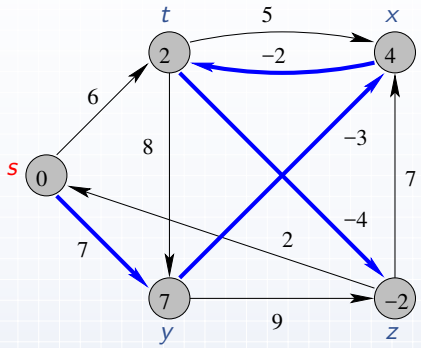
Ordem:
$(t,x),(t,y),(t,z),(x,t),(y,x),(y,z),(z,x),(z,s),(s,t),(s,y)$.

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

Corretude do algoritmo Bellman-Ford

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.

Corretude do algoritmo Bellman-Ford

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

- o algoritmo devolve TRUE,
- $d[v] = \text{dist}(s, v)$ 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.

Corretude do algoritmo Bellman-Ford

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

Relembrando...

(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 $d[v_k] = \text{dist}(s, v_k)$.

Corretude do algoritmo Bellman-Ford

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, $d[v] = d[v_k] = \text{dist}(s, v)$.

Corretude do algoritmo Bellman-Ford

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

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

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

Corretude do algoritmo Bellman-Ford

Agora falta mostrar que BELLMAN-FORD devolve TRUE.

Seja (u, v) uma aresta de G. Então

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

A desigualdade acima vale pois supomos que (G, w) não contém ciclos negativos.

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

Corretude do algoritmo Bellman-Ford

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

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

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

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

Corretude do algoritmo Bellman-Ford

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 corretude.

Aplicação: Sistemas de restrições de diferenças

Considere o seguinte sistema de desigualdades lineares:

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

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.

Sistemas de restrições de diferenças

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.

Sistemas de restrições de diferenças

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 direcionado (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.

Grafo de restrições

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

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

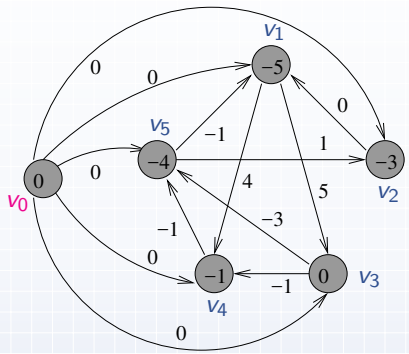
e

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

Note que todo vértice é atingível a partir de v_0.

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

Grafo de restrições

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

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

é uma solução viável do sistema.
Se G contém ciclos negativos, então o sistema não possui 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 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: $|V|.O(VE) = O(V^2 E)$.

Caminhos mínimos entre todos os pares de vértices

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, 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 $\omega(i, j) = \infty$.

Estrutura de um caminho mínimo

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

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

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

Estrutura de um caminho mínimo

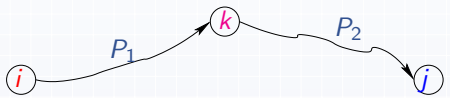
Sejam i e j dois vértices de G. Considere todos os caminhos de i a j cujos vértices internos pertencem a $\{1, \ldots, k\}$. Seja P um caminho mínimo entre todos eles.

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 $\{1, \ldots, k-1\}$.

Estrutura de um caminho mínimo

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

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 $D^{(n)}$.

FLOYD-WARSHALL(W, n)
1 $D^{(0)} \leftarrow W$
2 **para** $k \leftarrow 1$ **até** n **faça**
3 **para** $i \leftarrow 1$ **até** n **faça**
4 **para** $j \leftarrow 1$ **até** n **faça**
5 $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
6 **devolva** $D^{(n)}$

Complexidade: $O(V^3)$

Como encontrar os caminhos?

O algoritmo devolve também uma matriz $\Pi = (\pi_{ij})$ tal que
(a) $\pi_{ij} = \text{NIL}$ 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 partir de i, caso contrário.

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

Quando $k = 0$ temos

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

Como encontrar os caminhos?

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

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, $\pi_{ij}^{(k-1)}$.

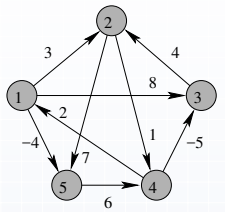
Caso contrário, tomamos como $\pi_{ij}^{(k)}$ o predecessor de j em um caminho mínimo de k a j com vértices internos em $\{1, 2, \ldots, k-1\}$, ou seja, $\pi_{kj}^{(k-1)}$.

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!

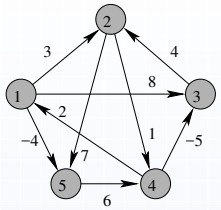
Exemplo



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

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

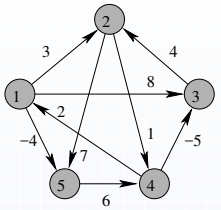
Exemplo



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

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

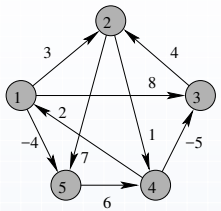
Exemplo



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

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

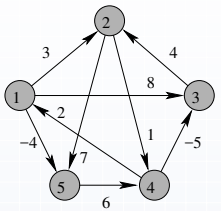
Exemplo



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

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

Exemplo



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

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

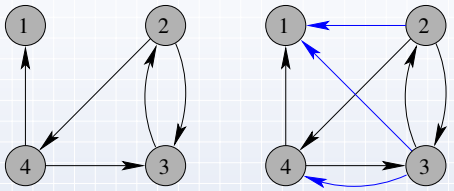
Fecho transitivo de grafos direcionados

Seja $G = (V, E)$ um grafo direcionado 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 direcionados

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 direcionados

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

Note que

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

e para $k \geq 1$,

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

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

Algoritmo de Transitive-Closure

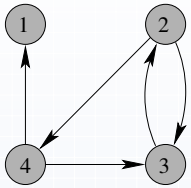
A entrada do algoritmo é uma matriz de adjacência A com $n = |V|$ linhas e colunas.

A saída é a matriz $T^{(n)}$.

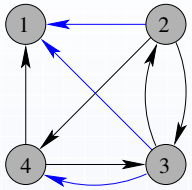
```
TRANSITIVE-CLOSURE(A, n)
1  T^(0) ← A + I_n
2  para k ← 1 até n faça
3     para i ← 1 até n faça
4        para j ← 1 até n faça
5           t_ij^(k) ← t_ij^(k-1) ∨ (t_ik^(k-1) ∧ t_kj^(k-1))
6  devolva T^(n)
```

Complexidade: $O(V^3)$

Exemplo



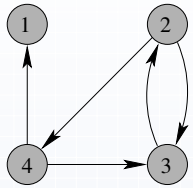
G



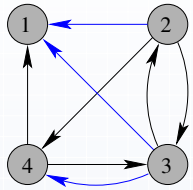
G^*

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

Exemplo



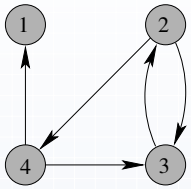
G



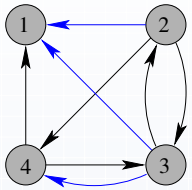
G^*

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

Exemplo



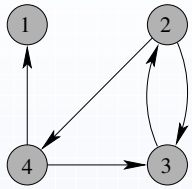
G



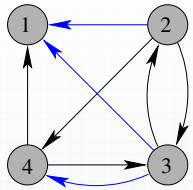
G^*

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

Exemplo



G



G^*

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

Algoritmo de Transitive-Closure

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.