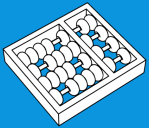
Caminhos mínimos

MO417 - Complexidade de Algoritmos I

22

06/24



Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

"The shortest distance between two points is a straight line."

Arquimedes

Caminhos mínimos com uma origem

Problema do Caminho Mínimo

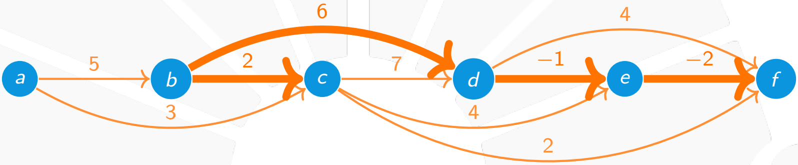
Considere um par (G, w) em que:

- G é um grafo direcionado.
- w associa um peso $w(u, v)$ para cada aresta $(\mathbf{u}, \mathbf{v})$.

Estamos interessados nos seguintes problemas:

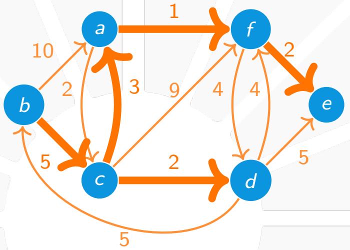
1. **Problema do caminho mínimo entre dois vértices:** Dados **s** e **t**, encontrar um caminho de peso mínimo de **s** a **t**.
2. **Problema dos caminhos mínimos com mesma origem:** Dado **s**, encontrar um caminho de peso mínimo de **s** a **v** para **TODO** vértice **v** de G.

Exemplo: grafo direcionado acíclico



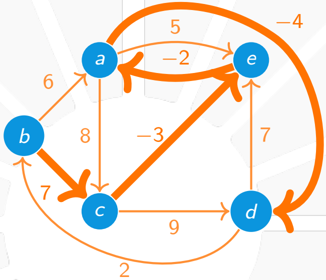
v	**a**	**b**	**c**	**d**	**e**	**f**
dist(**b**, v)	∞	0	2	6	5	3

Exemplo: grafo direcionado sem arestas negativas



v	**a**	**b**	**c**	**d**	**e**	**f**
dist(**b**, v)	8	0	5	7	11	9

Exemplo: grafo direcionado com arestas negativas



v	**a**	**b**	**c**	**d**	**e**
dist(**b**, v)	2	0	7	−2	4

Representando caminhos mínimos

A saída é similar à da busca em largura a partir de **s**:

- Para cada **v** $\in$ **V[G]**, associamos um **PREDECESSOR** π[**v**].
- O vetor π induz uma **ÁRVORE DE CAMINHOS MÍNIMOS** com raiz em **s**.
- Um caminho de **s** a **v** na árvore é um caminho mínimo de **s** a **v** no grafo G.

Definição

Problema (Caminhos mínimos com mesma origem)

Entrada: *Um grafo direcionado acíclico* $G = (V, E)$, *com peso* w *nas arestas e um vértice de origem* s.

Saída: *Um vetor* d *com* $d[v] = \text{dist}(s,v)$ *para* $v \in V$ *e um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS***.

Subestrutura ótima de caminhos mínimos

Teorema

Seja (G, w) *um grafo direcionado* ***SEM CICLOS NEGATIVOS*** *e seja*

$$P = (\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k})$$

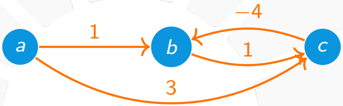
um caminho mínimo de $\mathbf{v_1}$ *a* $\mathbf{v_k}$*. Então para quaisquer índices* i, j *com* $1 \leq i \leq j \leq k$*, o subcaminho*

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

é um caminho mínimo de $\mathbf{v_i}$ *a* $\mathbf{v_j}$*.*

Subestrutura ótima de caminhos mínimos

A subestrutura ótima **NÃO** vale se o grafo tiver **CICLOS NEGATIVOS**:



- (a, b, c) é um caminho mínimo de **a** a **c** com peso $1 + 1 = 2$.
- Mas, (**a**,**b**) **NÃO** é um caminho mínimo de **a** a **b**.
- (**a**,**c**,**b**) é um caminho mínimo de **a** a **b** com peso $3 - 4 = -1$.

Algoritmos baseados em relaxação

Veremos algoritmos com as seguintes características:

1. Inicializam *d* e π com uma sub-rotina INITIALIZE-SINGLE-SOURCE.
2. Alteram *d* e π apenas com uma sub-rotina RELAX.

Esses algoritmos mantêm algumas invariantes:

- Existe um caminho de **s** a **v** com peso *d*[**v**].
- Esse caminho pode ser recuperado por meio π.
- Assim, *d*[**v**] é sempre **MAIOR OU IGUAL** a dist(**s,v**).
- Queremos que no final valha *d*[**v**] = dist(**s,v**).

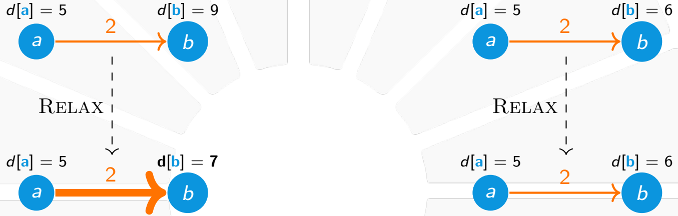
Inicialização

Algoritmo: INITIALIZE-SINGLE-SOURCE(G, s)

```
para cada v ∈ V[G]
    d[v] ← ∞
    π[v] ← NIL
d[s] ← 0
```

Relaxação

Tenta melhorar a estimativa $d[\mathbf{v}]$ examinando **(u,v)**.

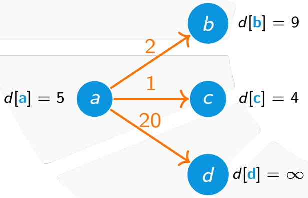


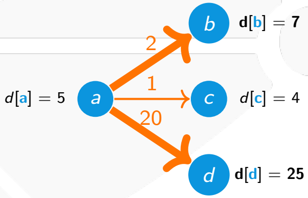
Algoritmo: $\textsc{Relax}(u, v, w)$

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

Relaxação dos vizinhos

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





Casos do problema de caminhos mínimos

Veremos três algoritmos baseados em relaxação para tipos de subcasos diferentes do Problema de caminhos mínimos:

1. **APLICAÇÃO DE ORDENAÇÃO TOPOLÓGICA**: G é acíclico:
2. **ALGORITMO DE DIJKSTRA**: (G, w) não tem arestas de peso negativo.
3. **ALGORITMO DE BELLMAN-FORD**: (G, w) pode ter arestas de peso negativo, mas não contém ciclos negativos.

Caminhos mínimos em grafos acíclicos

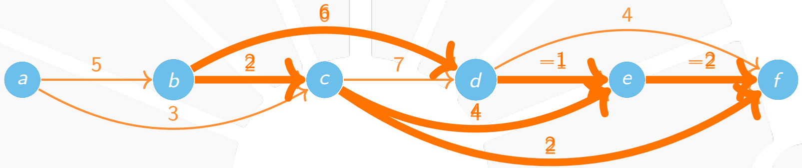
Caminhos mínimos em grafos acíclicos

Problema

Entrada: *Um grafo direcionado acíclico* $G = (\boldsymbol{V}, \boldsymbol{E})$, *uma função de peso* w *nas arestas e um vértice origem* s.

Saída: *Um vetor* d *com* $d[\boldsymbol{v}] = \mathrm{dist}(\boldsymbol{s,v})$ *para* $\boldsymbol{v} \in \boldsymbol{V}$ *e um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS***.

Exemplo



	a	b	c	d	e	f
d	∞	0	2	6	5 6	4 3

Algoritmo

Algoritmo: $\textsc{Dag-Shortest-Paths}(G, w, s)$

```
1 ordene topologicamente os vértices de G
2 INITIALIZE-SINGLE-SOURCE(G, s)
3 para cada u na ordem topológica
4     para cada v ∈ Adj[u]
5         RELAX(u, v, w)
6 devolva d, π
```

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

Complexidade de $\textsc{Dag-Shortest-Paths}$: $O(V+E)$.

Correção

- Há diversas estratégias para demonstrar que Dag-Shortest-Paths esteja correto.
- Vamos demonstrar algumas propriedades que valem porque o algoritmo é baseado em relaxação.
- Essas propriedades também serão úteis para analisar os outros algoritmos depois.

Propriedade de algoritmos baseados em relaxação

Ao longo de um algoritmo baseado em relaxação sempre vale:

- **Limite superior:** Vale $d[\mathbf{v}] \geq \text{dist}(\mathbf{s,v})$ e, tão logo $d[\mathbf{v}]$ alcança $\text{dist}(\mathbf{s,v})$, nunca mais muda.
- **Inexistência de caminho:** Se não existe caminho de **s** a **v**, então $d[\mathbf{v}] = \infty$.
- **Subgrafo de predecessores:** Se $d[\mathbf{v}] < \infty$, então o subgrafo dos predecessores induzido por π é um caminho de peso $d[\mathbf{v}]$.
- **Convergência:** Se p é um caminho mínimo de **s** até **v** terminando com a aresta **(u,v)** e $d[\mathbf{u}] = \text{dist}(\mathbf{s,u})$, então ao relaxar **(u,v)**, $d[\mathbf{v}] = \text{dist}(\mathbf{s,v})$, que nunca mais muda.
- **Relaxamento de caminho:** Se $p = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_k)$ é um caminho mínimo de $\mathbf{s} = \mathbf{v}_0$ a $\mathbf{v}_k$ e relaxamos as arestas de p na ordem $(\mathbf{v}_0, \mathbf{v}_1),\ (\mathbf{v}_1, \mathbf{v}_2),\ \ldots,\ (\mathbf{v}_{k-1}, \mathbf{v}_k)$, então $d[\mathbf{v}_k] = \text{dist}(\mathbf{s}, \mathbf{v}_k)$.

  A propriedade vale mesmo se tivermos realizado quaisquer outras relaxações durante a execução.

Correção de DAG-SHORTEST-PATHS

- Seja $\mathbf{v}$ um vértice e suponha que $p = (\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_k})$ é um caminho mínimo de $\mathbf{s} = \mathbf{v_0}$ a $\mathbf{v} = \mathbf{v_k}$.
- Como $\mathbf{v_0}$, $\mathbf{v_1}$, $\ldots$, $\mathbf{v_k}$ aparecem em ordem na ordenação topológica, as arestas $(\mathbf{v_0}, \mathbf{v_1})$, $\ldots$, $(\mathbf{v_{k-1}}, \mathbf{v_k})$ são relaxadas em ordem.
- Logo, pela propriedade do **Relaxamento de caminho**, o algoritmo computa corretamente $d[\mathbf{v}] = \mathrm{dist}(\mathbf{s},\mathbf{v})$ para cada $\mathbf{v} \in \mathbf{V}$.

Também é fácil ver que π define uma árvore de caminhos mínimos. Por quê?

Perguntas

Vamos fazer alguns exercícios?

Exercício 1

Como se resolve o problema de encontrar um caminho de peso **MÁXIMO** de **s** a **t** em um grafo direcionado acíclico (G, w)?

Exercício 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 têm o mesmo peso $C > 0$?

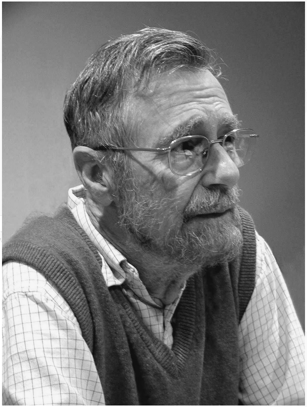
Algoritmo de Dijkstra

Sobre o algoritmo

Veremos agora um algoritmo para caminhos mínimos em grafos que podem conter ciclos, mas **SEM ARESTAS DE PESOS NEGATIVO**.

O algoritmo também é **BASEADO EM RELAXAÇÃO**.

Edsger Wybe Dijkstra (11/05/1930 – 06/08/2002)



Principais áreas de atuação:

- Algoritmos em grafos.
- Programação concorrente e distribuída.
- Sistemas operacionais.
- Compiladores e linguagens de programação.

Publicação

1957 *M. Leyzorek, R.S. Gray, A.A. Johnson, W.C. Ladew, S.R. Meaker, R.M. Petry* e *R.N. Seitz. Investigation of model techniques - First annual report - 6 June 1956 - 1 July 1957 - A study of model techniques for communication systems, Case Institute of Technology, Cleveland, Ohio*.

1958 *G.B. Dantzig. On the shortest route through a network, The RAND Corporation, Santa Monica, California*.

1959 *E.W. Dijkstra. A note on two problems in connexion with graphs, Numerische Mathematik*.

Problema

Problema

Entrada: *Um grafo direcionado* $G = (\boldsymbol{V}, \boldsymbol{E})$, *uma função de peso* w *nas arestas (sem arestas de peso negativo) e um vértice origem* $\boldsymbol{s}$.

Saída: *Um vetor* d *com* $d[\boldsymbol{v}] = \mathrm{dist}(\boldsymbol{s},\boldsymbol{v})$ *para* $\boldsymbol{v} \in \boldsymbol{V}$ *e um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS***.

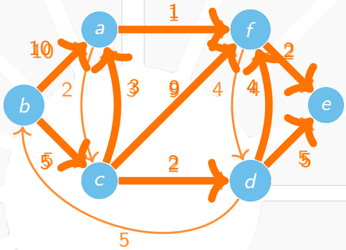
O algoritmo

Algoritmo: $\textsc{Dijkstra}(G, w, s)$

$\textsc{Initialize-Single-Source}(G, s)$
$S \leftarrow \emptyset$
$Q \leftarrow V[G]$ **enquanto** $Q \neq \emptyset$
  $u \leftarrow \textsc{Extract-Min}(Q)$
  $S \leftarrow S \cup \{u\}$
  para cada $v \in Adj[u]$
    $\textsc{Relax}(u, v, w)$
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



	a	b	c	d	e	f
d	8	0	5	7	11	9

Correção do algoritmo

Precisamos provar que algoritmo está correto, temos que mostrar que, quando o algoritmo termina:

1. $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$ para todo $\mathbf{v} \in \mathbf{V[G]}$ e
2. π induz uma **ÁRVORE DE CAMINHOS MÍNIMOS**.

Observe que:

- DIJKSTRA é baseado em relaxação.
- Pela propriedade do **Subgrafo de predecessores**, sabemos que o vetor π induz uma árvore que testemunha d.
- Assim, basta mostrar que de fato $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$.

Invariante

Iremos mostrar que no no início de cada iteração da linha 4 no algoritmo DIJKSTRA, vale $d[x] = \text{dist}(s, x)$ **para cada** $x \in S$.

- **Inicialização.** No início, **S=**$\emptyset$, então a invariante vale trivialmente.
- **Manutenção.**
  - Suponha que a invariante vale para S no início da iteração.
  - Nessa iteração, DIJKSTRA escolhe um vértice **u** com menor $d[\mathbf{u}]$ em **Q** e o adiciona a **S**.
  - Queremos mostrar que a invariante vale para $\mathbf{S} \cup \{\mathbf{u}\}$.
  - Basta verificar que neste instante $d[\mathbf{u}] = \text{dist}(\mathbf{s},\mathbf{u})$.

Demonstração

Sejam:

- P um caminho mínimo de **s** a **u** (i.e., com peso dist(**s**,**u**)).
- **y** o primeiro vértice de P que não pertence a **S**
- **x** o vértice em P que precede **y**.

Suponha que $d[\mathbf{u}] > \text{dist}(\mathbf{s},\mathbf{u})$:

- Pela hipótese de indução, $d[\mathbf{x}] = \text{dist}(\mathbf{s},\mathbf{x})$ pois $\mathbf{x} \in \mathbf{S}$. Logo,

$$\begin{aligned} d[\mathbf{y}] &\leq d[\mathbf{x}] + w(\mathbf{x},\mathbf{y}) \quad \text{(pois relaxamos } (\mathbf{x},\mathbf{y})\text{)} \\ &= \text{dist}(\mathbf{s},\mathbf{x}) + w(\mathbf{x},\mathbf{y}) \\ &\leq w(P) = \text{dist}(\mathbf{s},\mathbf{u}) < d[\mathbf{u}]. \end{aligned}$$

- Mas, $d[\mathbf{y}] < d[\mathbf{u}]$ contraria a escolha de **u**.
- Portanto, $d[\mathbf{u}] \leq \text{dist}(\mathbf{s},\mathbf{u})$.

Concluindo que $d[\mathbf{u}] = \text{dist}(\mathbf{s},\mathbf{u})$ (propriedade de **Limite superior**).

Demonstração

Para terminar a demonstração, observe que:

- Após a última iteração, **S** é o conjunto dos vértices atingíveis por **s**.
- Por **Inexistência de caminho**, se um vértice **v** não é atingível, então $d[\mathbf{v}] = \infty$.
- Portanto, para todo $\mathbf{v} \in \mathbf{V}$, vale $d[\mathbf{v}] = \text{dist}(\mathbf{s},\mathbf{v})$.

Dijkstra precisa de arestas com peso não negativo

Na demonstração, supomos que **NÃO HÁ ARESTAS NEGATIVAS**

- Se a hipótese não valer, o algoritmo de Dijkstra pode falhar.
- Encontre um grafo com arestas negativas para o qual o algoritmo de Dijkstra **NÃO** funciona (exercício).
- Existe um exemplo com 4 vértices, com apenas uma aresta negativa e sem ciclos de peso negativo.

Complexidade de tempo

Depende de como a fila de prioridade Q é implementada:

- Operações INSERT, EXTRACT-MIN, DECREASE-KEY.

Observe que:

- Os passos INITIALIZE-SINGLE-SOURCE e $Q \leftarrow V[G]$ escondem chamadas a INSERT.
- RELAX esconde chamada a DECREASE-KEY.

Linha(s)	Tempo total
1–3	$\lvert V\rvert$ chamadas a INSERT
5	$\lvert V\rvert$ chamadas a EXTRACT-MIN
8	$\lvert E\rvert$ chamadas a DECREASE-KEY

Complexidade de Dijkstra:
$O(|V| \times \text{INSERT} + |V| \times \text{EXTRACT-MIN} + |E| \times \text{DECREASE-KEY})$.

Complexidade de tempo

Total: $O(|V| \times \textsc{Insert} + |V| \times \textsc{Extract-Min} + |E| \times \textsc{Decrease-Key})$

Tipo de fila	Insert	Extract-Min	Decrease-Key	TOTAL
Vetor	$O(1)$	$O(V)$	$O(1)$	$O(V^2)$
Min-Heap	$O(\log V)$	$O(\log V)$	$O(\log V)$	$O((V+E)\log V)$
Fibonacci	$O(1)$	$O(\log V)$	$O(1)$	$O(V \log V + E)$

Algoritmo de Bellman-Ford

Arestas vs ciclos de peso negativo

- O algoritmo de Dijkstra resolve o Problema dos Caminhos Mínimos quando (G, w) **NÃO TEM 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

- O Problema dos Caminhos Mínimos para instâncias com ciclos negativos é NP-**DIFÍCIL**.
    - Acreditamos que **NÃO** existem algoritmos eficientes para resolver problemas NP-difíceis.
    - Assim, vamos nos restringir ao Problema de Caminhos Mínimos **SEM CICLOS NEGATIVOS**.
- Um algoritmo que resolve o problema restrito é o algoritmo de Bellman-Ford, que também é baseado em relaxação.

Definição do problema

Problema

Entrada: *Um grafo direcionado* $G = (\boldsymbol{V}, \boldsymbol{E})$*, uma função de peso* w *nas arestas e um vértice origem* $\boldsymbol{s}$.

Saída: FALSE, *se existe um ciclo negativo atingível a partir de* $\boldsymbol{s}$. *Caso contrário, além de* TRUE, *também devolve um vetor* d *com* $d[\boldsymbol{v}] = \mathrm{dist}(\boldsymbol{s},\boldsymbol{v})$ *para* $\boldsymbol{v} \in \boldsymbol{V}$ *e um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS***.

Ideia do algoritmo de Bellman-Ford

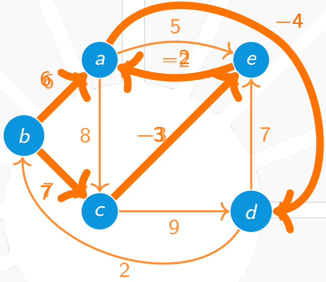
Relaxamento de caminho: Para **QUALQUER** caminho mínimo $(v_0, v_1, \ldots, v_k)$, queremos relaxar (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) em ordem.

1. Executamos RELAX para todas as arestas:
   $\Rightarrow$ (v_0, v_1) relaxada.
2. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow$ (v_0, v_1), (v_1, v_2) relaxadas em ordem.
3. Executamos novamente RELAX para todas as arestas:
   $\Rightarrow$ (v_0, v_1), (v_1, v_2), (v_2, v_3) relaxadas em ordem.
4. Repetimos esse passo até $|V| - 1$ vezes. Por quê?
   $\Rightarrow$ (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k) relaxadas em ordem.

Podemos verificar se o grafo contém **CICLOS NEGATIVOS** executando mais uma vez:

- Se algum valor $d[v]$ diminuir, então há ciclo negativo.

Exemplo



	a	b	c	d	e
d	∞ 6 2	0	∞ 7	∞ 2 −2	∞ 4

Ordem: **(a,c),(a,d),(a,e),(c,d),(c,e),(d,b),(d,e),(e,a),(b,a),(b,c)**.

O algoritmo de Bellman-Ford

Algoritmo: $\textsc{Bellman-Ford}(G, w, s)$

$\textsc{Initialize-Single-Source}(G, s)$
para $i \leftarrow 1$ **até** $|V[G]| - 1$
  para cada $(u, v) \in E[G]$
    $\textsc{Relax}(u, v, w)$
para cada $(u, v) \in E[G]$
  se $d[v] > d[u] + w(u, v)$
    devolva FALSE
devolva TRUE, d, π

Complexidade de tempo: $O(VE)$

Correção do algoritmo Bellman-Ford

Teorema

BELLMAN-FORD *devolve:*

- FALSE, *se existe um ciclo negativo atingível a partir de* s.
- TRUE, *caso contrário; neste caso devolve também:*
  - *Um vetor* d *com* $d[v] = \text{dist}(s,v)$ *para* $v \in V$.
  - *Um vetor* π *definindo uma* ***ÁRVORE DE CAMINHOS MÍNIMOS***.

Correção do algoritmo Bellman-Ford

Primeiro, suponha que não há ciclos negativos atingíveis por **s**.

Considere **v** $\in$ **V[G]** e os valores de d e π após o primeiro laço:

- Se **v** não é atingível, $d[\mathbf{v}] = \infty$ (por **Inexistência de caminho**).
- Senão, existe caminho mínimo $(\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_k})$ de $\mathbf{s} = \mathbf{v_0}$ a $\mathbf{v} = \mathbf{v_k}$.
- Como $k \leq |V| - 1$, então $(\mathbf{v_0}, \mathbf{v_1})$, $(\mathbf{v_1}, \mathbf{v_2})$, $\ldots$, $(\mathbf{v_{k-1}}, \mathbf{v_k})$ foram relaxadas **NESTA ORDEM**.
- Por **Relaxamento de caminho**, $d[\mathbf{v}] = \mathrm{dist}(\mathbf{s},\mathbf{v})$.
- Também, por **Sugrafo de predecessores**: π induz um caminho mínimo de **s** a **v**.

Correção do algoritmo Bellman-Ford

Também temos que mostrar que nesse caso BELLMAN-FORD devolve TRUE.

- Considere d imediatamente após o primeiro laço.
- Nesse instante, $d[\mathbf{v}] = \mathrm{dist}(\mathbf{s},\mathbf{v})$ para todo vértice $\mathbf{v}$.
- Por **Convergência**, sabemos que d nunca mais muda.
- Portanto, o teste da linha 6 falha sempre.
- Concluindo que o algoritmo devolve TRUE.

Correção do algoritmo Bellman-Ford

Suponha agora que (G, w) contenha **CICLO NEGATIVO** alcançável por $\mathbf{s}$.

Queremos mostrar que o algoritmo devolve FALSE:

- Seja $C = (\mathbf{v_0}, \mathbf{v_1}, \ldots, \mathbf{v_k} = \mathbf{v_0})$ um ciclo tal que

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

- Suponha, por contradição que o algoritmo devolve TRUE.
- Como relaxamos cada aresta $(\mathbf{v_{i-1}}, \mathbf{v_i})$:

$$d[\mathbf{v_i}] \leq d[\mathbf{v_{i-1}}] + w(\mathbf{v_{i-1}}, \mathbf{v_i}).$$

Correção do algoritmo Bellman-Ford

- Somando as desigualdades anteriores para cada aresta do ciclo, temos:

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

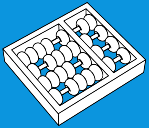
- Como $\mathbf{v_0} = \mathbf{v_k}$, temos que $\sum_{i=1}^{k} d[\mathbf{v_i}] = \sum_{i=1}^{k} d[\mathbf{v_{i-1}}]$.
- Logo, $0 \leq \sum_{i=1}^{k} w(\mathbf{v_{i-1}}, \mathbf{v_i}) = w(C)$.
- Mas isso é uma contradição, pois C é ciclo negativo.
- Concluindo que, nesse caso, o algoritmo devolve FALSE.

Caminhos mínimos

MO417 - Complexidade de Algoritmos I

22

06/24



Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br