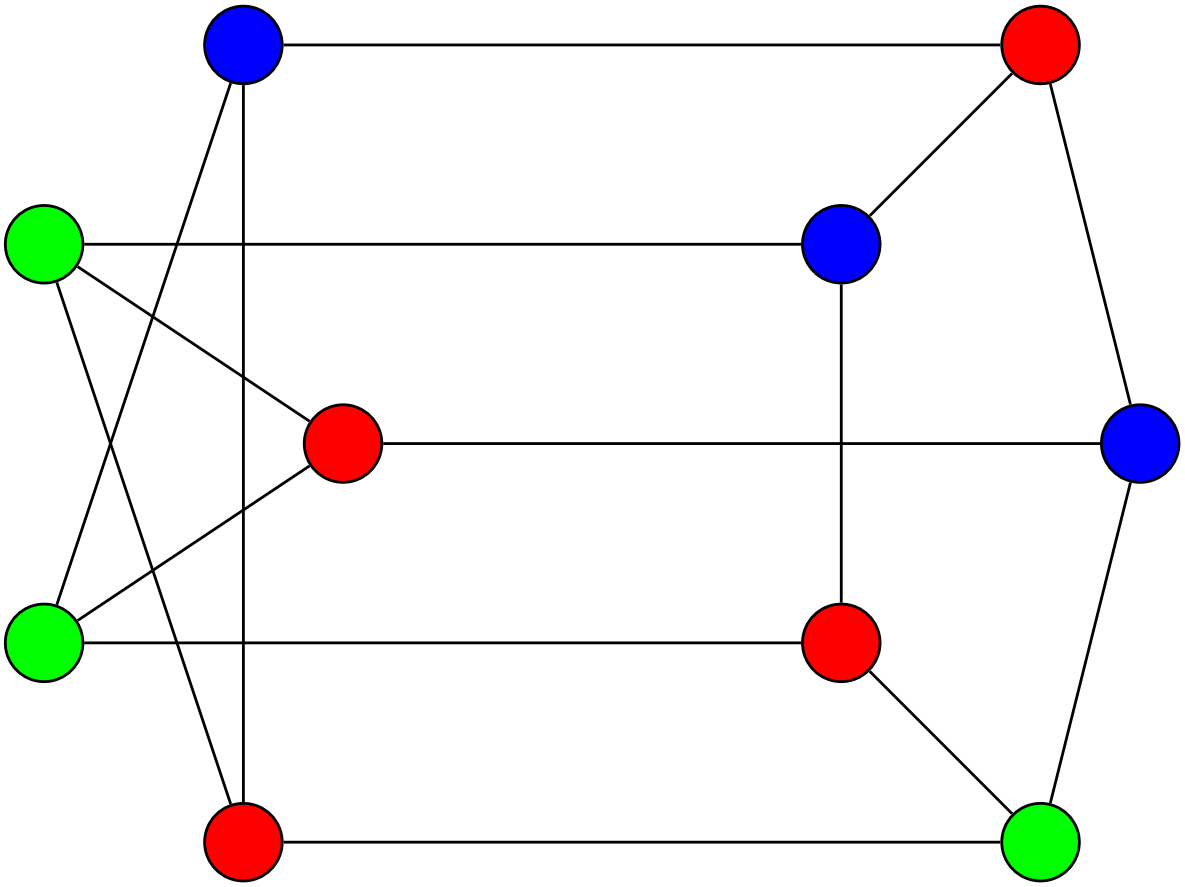
Centro de Matemática, Computação e Cognição
Universidade Federal do ABC
Profa. Carla Negri Lintzmayer

Notas de aula da disciplina Teoria dos Grafos

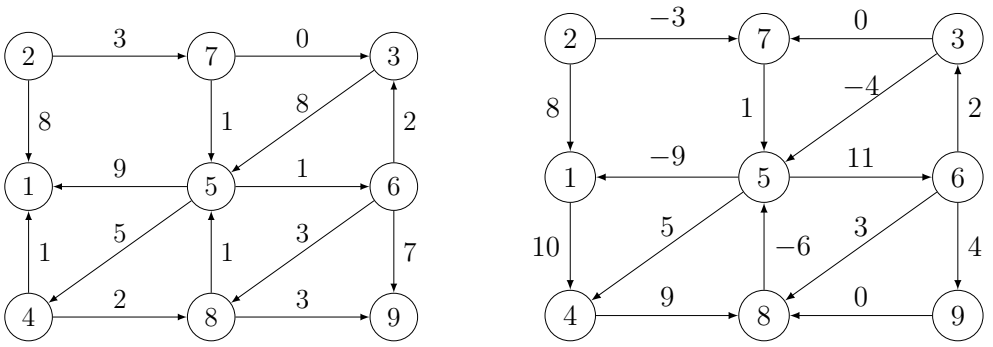


Esse material é um compilado dos seguintes:

- Sedgewick, R.. *Algorithms in C, part 5: graph algorithms*. 3rd ed. Addison-Wesley. 2002.
- Bondy, J. A.; Murty, U. S. R.. *Graph Theory*. Graduate Texts in Mathematics. Springer. New York. 2008.
- Lintzmayer, C. N.; Mota, G. O.. *Notas de aulas - Análise de algoritmos e estruturas de dados*. Em construção.

12 Aulas 16 e 17: caminhos mínimos

- Seja G um digrafo com custos nas arestas dados por $w\colon E(G) \to \mathbb{R}$.
  - De forma geral, dado um subdigrafo $H \subseteq G$, denotamos por $w(H)$ o valor $\sum_{e \in E(H)} w(e)$, denominado *custo* de H.
  - Particularmente, o *custo* $w(P)$ de um caminho $P = (v_1, v_2, \ldots, v_k)$ é a soma dos custos das arestas de P, isto é, $w(P) = \sum_{i=1}^{k-1} w(v_i v_{i+1})$.
- Definimos a *distância (ponderada) entre u e v*, denotada $dist_G^w(u, v)$, como sendo o custo de um uv-caminho de menor custo, ou ∞ se não houver uv-caminho.
- Dizemos que um caminho de menor custo (cujo custo é igual à distância) é um *caminho mínimo*.
  - Um st-caminho mínimo não é necessariamente único.
- Exemplo: no digrafo da esquerda, um 5 8-caminho mínimo tem custo 4 e no digrafo da direita, tem custo 7.



- Existem basicamente três variações de problemas de caminhos mínimos.

> *Problema do Caminho Mínimo de Único Destino*
>
> **Entrada:** digrafo G, $w\colon E(G) \to \mathbb{R}$ e vértices $s, t \in V(G)$.
>
> **Objetivo:** calcular $dist_G^w(s, t)$.

- Algoritmos clássicos que o resolvem:
  - A*: tempo $O(E \log V)$

Problema do Caminho Mínimo de Única Fonte

Entrada: digrafo G, $w\colon E(G) \to \mathbb{R}$ e vértice $s \in V(G)$.

Objetivo: calcular $dist_G^w(s, v)$ para todo $v \in V(G)$.

- Algoritmos clássicos que o resolvem:
  - BFS, se $w(e) = w(f)$ para toda $e, f \in E(G)$: tempo $O(V + E)$
  - DFS, se G é um DAG: tempo $O(V + E)$
  - Dijkstra, 1959, se $w(e) \geq 0$ para toda $e \in E(G)$: tempo $O(E \log V)$
  - Bellman-Ford, 1956: tempo $\Theta(VE)$

Problema do Caminho Mínimo entre Todos os Pares

Entrada: digrafo G, $w\colon E(G) \to \mathbb{R}$.

Objetivo: calcular $dist_G^w(u, v)$ para todo par $u, v \in V(G)$.

- Algoritmos clássicos que o resolvem:
  - Floyd-Warshall, 1962: tempo $\Theta(V^3)$
  - Johnson, 1977: tempo $\Theta(VE \log V)$
- Observações:
  - Algoritmos para o problema do Caminho Mínimo entre Todos os Pares certamente resolvem os outros dois problemas.
  - Algoritmos para o problema do Caminho Mínimo de Única Fonte podem ser usados para resolver o problema do Caminho Mínimo entre Todos os Pares: execute-os uma vez para cada vértice sendo fonte.
  - Alguns tempos de execução podem ser melhorados em grafos.
  - Melhor algoritmo para todos os pares: tempo $O(V^3 \log\log V / \log^2 V)$ (Han & Takaoka, 2014).

12.1 Grafos, digrafos e custos negativos

- Algoritmos que lidam com problemas de encontrar distâncias em (di)grafos não funcionam corretamente quando o grafo possui arestas com pesos negativos ou o digrafo possui ciclos com pesos negativos.
- Com o que se sabe até o momento em Ciência da Computação, não é possível existir um algoritmo eficiente que resolva problemas de distância nessas situações.
  - O problema de caminhos mínimos em digrafos com ciclos de custo negativo é NP-difícil.
- Lembre-se que dado um grafo G, seu digrafo associado é o digrafo $D(G)$ com conjunto de vértices $V(D(G)) = V(G)$ e $\{u, v\} \in E(G)$ se e somente se $(u, v) \in E(D(G))$ e $(v, u) \in E(D(G))$.
- Se G é um grafo com custos nas arestas dados por uma função w, então podemos dar custos aos arcos de $D(G)$ com uma função w' tal que $w'(uv) = w'(vu) = w(uv)$.
  - Note que caminhos mínimos em G são caminhos mínimos em $D(G)$ e vice-versa.
  - Se G tem uma aresta com $w(uv) < 0$, então (u, v, u) é um ciclo de custo negativo em $D(G)$.
- Se conseguirmos resolver o problema em qualquer tipo de digrafo (que não tenha ciclo negativo), então conseguiremos resolvê-lo, em particular, para digrafos que são digrafos associados de grafos.

12.2 Princípios genéricos de algoritmos para Caminhos Mínimos

Teorema 12.1. *Seja D um digrafo e $w\colon E(D) \to \mathbb{R}$ tal que D não tem ciclo negativo. Se $P = (v_0, v_1, \ldots, v_k)$ é um v_0v_k-caminho mínimo, então $(v_i, v_{i+1}, \ldots, v_j)$ é um v_iv_j-caminho mínimo para qualquer $0 \leq i \leq j \leq k$.*

Lema 12.2. *Seja D um digrafo e $w\colon E(D) \to \mathbb{R}$ tal que D não tem ciclo negativo. Dados $u, v \in V(D)$, se Q é um uv-passeio em D, então existe um uv-caminho P tal que $w(P) \leq w(Q)$.*

12.3 Caminhos Mínimos de Única Fonte

- Os algoritmos manterão, para cada vértice $u \in V(D)$:
  - $dist[u]$, para armazenar o custo de um su-caminho encontrado pelo algoritmo (estimativa de distância);
  - $pred[u]$, para armazenar o predecessor de u no su-caminho encontrado pelo algoritmo.
- *Relaxar* uma aresta xy significa verificar se é possível utilizá-la para melhorar a estimativa de distância até y:

```
1: Função RELAXA(x, y)
2:     Se dist[x] + w(xy) < dist[y] então        ▷ Cuidado!!! Se dist[x] ≠ ∞
3:         dist[y] ← dist[u] + w(xy)
4:         pred[y] ← x
```

- Cuidados de implementação:
  - Matematicamente, $\infty \not> \infty + c$ para toda constante $c \in \mathbb{R}$.
  - Mas e quando ∞ for INT_MAX ou DOUBLE_MAX? E quando $c < 0$?

- Os algoritmos sempre inicializam $dist[s] = 0$ e $dist[v] = \infty$ para todo $v \in V(D) \backslash \{s\}$.

- O resultado a seguir mostra que eles sempre vão ter, em $dist[v]$, uma estimativa de distância que é pelo menos o valor da distância real.

Proposição 12.3. *Seja D um digrafo, $w\colon E(D) \to \mathbb{R}$ e $s \in V(D)$ um vértice qualquer. Se D não possui ciclos de custo negativo, então um algoritmo que modifica dist apenas por meio de relaxações sempre vai manter $dist[x] \geq dist^w_D(s, x)$ para todo $x \in V(D)$.*

Demonstração. Vamos provar, por indução em k, que após k relaxações nós temos $dist[x] \geq dist^w_D(s, x)$ para todo $x \in V(D)$.

Quando $k = 0$, nenhuma relaxação ocorreu. De fato, neste momento temos $dist[s] = 0 \geq dist^w_D(s, s) = 0$ e $dist[x] = \infty \geq dist^w_D(s, x)$ para todo $x \neq s$.

Agora suponha $k > 0$ e considere que a k-ésima relaxação será da aresta uv.

Neste momento, apenas $dist[v]$ tem chances de ser modificado, de forma que após a relaxação já podemos afirmar que $dist[x] \geq dist^w_D(s,x)$ para todo $x \neq v$. Ademais, como antes da relaxação já sabemos que $dist[v] \geq dist^w_D(s,v)$, precisamos mostrar que essa relação se mantém apenas se $dist[v]$ muda. Logo, podemos considerar que $dist[v] = dist[u] + w(uv)$ após a relaxação. Como um su-caminho mínimo seguido do arco uv é um sv-passeio, vale que $dist^w_D(s,v) \leq dist^w_D(s,u) + w(uv)$. Já sabíamos que $dist[u] \geq dist^w_D(s,u)$. Juntando as inequações, temos

$$dist^w_D(s,v) \leq dist^w_D(s,u) + w(uv) \leq dist[u] + w(uv) = dist[v]\,,$$

de onde vemos que se mantém $dist[v] \geq dist^w_D(s,v)$. □

- O teorema a seguir nos ajuda a provar mais facilmente a corretude de alguns desses algoritmos.

Teorema 12.4. *Seja D um digrafo ponderado por $w\colon E(D) \to \mathbb{R}$ tal que D não tem ciclo negativo. Seja $s \in V(D)$ e dist um vetor indexado pelos vértices tal que, para todo $v \in V(D)$ vale que:*

1. *$dist[v]$ é o custo de algum sv-caminho, caso exista;*
2. *$dist[v] = \infty$ se não há sv-caminho;*
3. *$dist[s] = 0$.*

Para todo $v \in V(D)$, $dist[v] = dist^w_D(s,v)$ se e somente se todo arco de D não pode ser relaxado.

Demonstração. Primeiro vamos mostrar que se, para todo $v \in V(D)$, $dist[v] = dist^w_D(s,v)$, então todo arco de D não pode ser relaxado. Suponha, para fins de contradição, que existe um arco $xy \in E(D)$ que pode ser relaxado, isto é, tal que $dist[x] + w(xy) < dist[y]$. Assim, podemos assumir que $dist[x] \neq \infty$ e, portanto, existe um sx-caminho P tal que $w(P) = dist[x]$. Seja Q construído a partir de P inserindo o vértice y ao final. Note que Q é um sy-passeio tal que $w(Q) = w(P)+w(xy)$. Pelo Lema 12.2, existe um sy-caminho Q' tal que $w(Q') \leq w(Q)$. Assim,

$$w(Q') \leq w(P) + w(xy) = dist[x] + w(xy) < dist[y] = dist^w_D(s,v)\,,$$

o que é um absurdo.

Agora, vamos provar que se todo arco de D não pode ser relaxado, então $dist[v] = dist_D^w(s, v)$ para todo $v \in V(D)$. Seja $x \in V(D)$. Se não há sx-caminho em D, sabemos, por hipótese, que $dist[x] = \infty = dist_D^w(s, x)$, e o resultado nesse caso segue. Então suponha que há sx-caminho em D. Seja $P = (v_0 = s, v_1, \ldots, v_k = x)$ um sx-caminho mínimo em D, isto é, $w(P) = dist_D^w(s, x)$. Como todo arco de D não pode ser relaxado, então sabemos que

$$dist[v_i] \leq dist[v_{i-1}] + w(v_{i-1}v_i)\,, \quad \text{para } i = 1, 2, \ldots, k\,.$$

Somando as k expressões acima, note que temos

$$dist[v_k] \leq dist[v_0] + \sum_{i=1}^{k} w(v_{i-1}v_i) = dist[s] + w(P) = 0 + w(P) = dist_D^w(s, x)\,.$$

Como, por hipótese, $dist[v_k] = dist[x]$ é o custo de um sx-caminho em D, só pode ser o caso de termos $dist[x] = dist_D^w(s, x)$, como queríamos. □

12.3.1 Algoritmo para DAGs

- Esse algoritmo simplesmente percorre os vértices do grafo de acordo com uma ordenação topológica, relaxando todos os arcos que saem do vértice.

```
1: Função CamMinUniFon-DAG(D, w, s)
2:     Para cada v ∈ V(D) faça
3:         pred[v] ← −1
4:         dist[v] ← ∞
5:     pred[s] ← s
6:     dist[s] ← 0
7:     S ← OrdenaTopologica(D)                    ▷ S = (v_1, ..., v_n)
8:     Para i ← 1 até |V(D)| faça
9:         Para cada x ∈ N^+(v_i) faça
10:            Relaxa(v_i, x)
```

- Ele claramente leva tempo $O(V + E)$: $O(V)$ no laço de inicialização + $O(V + E)$ na ordenação topológica + $O(V + E)$ no laço da relaxação.

Teorema 12.5. *Seja D um DAG, $w\colon E(D) \to \mathbb{R}$ e $s \in V(D)$ um vértice qualquer. O algoritmo CamMinUniFon-DAG(D, w, s) resolve o problema do Caminho Mínimo de Ùnica Fonte em digrafos acíclicos.*

Demonstração. Primeiro note que todo arco $xy \in E(D)$ é relaxado exatamente uma vez, quando $x = v_i$ para algum $i = 1, \ldots, k$. Logo após isso, teremos $dist[x]+w(xy) \geq dist[y]$, o que continuará válido até o fim da execução: $dist[y]$ só pode reduzir devido a relaxações de outros arcos zy, e $dist[x]$ não mudará devido à ordenação topológica. Então pelo Teorema 12.4, $dist[v] = dist_D^w(s, v)$ para todo $v \in V(D)$. □

12.3.2 Algoritmo de Dijkstra

- A ideia do algoritmo de Dijkstra é similar à dos algoritmos de busca e ao algoritmo de Prim: vamos crescer uma arborescência a partir de um vértice inicial, que no caso é o vértice s dado na entrada do problema.
  - Essa arborescência conterá sv-caminhos mínimos, para todo v para os quais existe caminho a partir de s.
- Inicialmente, nenhum vértice está visitado, s tem estimativa de distância $dist[s] = 0$, e todos os outros vértices têm estimativa de distância $dist[v] = \infty$.
- A cada iteração, um vértice não visitado x é escolhido para ser visitado, momento a partir do qual $dist[x]$ e $pred[x]$ não mudarão mais. Neste momento, todos os arcos que saem de x são relaxados.
- A forma como Dijkstra faz para escolher o próximo vértice x a ser visitado é gulosa: escolhe-se o vértice que, naquele momento, tem a menor estimativa de distância dentre os não visitados.
- Perceba que em nenhum momento o algoritmo de Dijkstra verifica se o digrafo de entrada possui arcos de peso negativo.
- De fato, ele encerra sua execução normalmente. Acontece, porém, que ele não calcula os pesos dos caminhos mínimos *corretamente*.

```
1: Função DIJKSTRA(D, w, s)
2:     Para cada v ∈ V(D) faça
3:         pred[v] ← −1
4:         dist[v] ← ∞
5:         visitado[v] ← 0
6:     dist[s] ← 0
7:     pred[s] ← s
8:     Enquanto houver vértice u com visitado[u] = 0 faça
9:         seja x um vértice não visitado com menor valor dist[x]
10:        visitado[x] ← 1
11:        Para cada y ∈ N⁺(x) faça
12:            RELAXA(x, y)
```

Teorema 12.6. *Seja D um digrafo, $w\colon E(D) \to \mathbb{R}_{\geq 0}$ e $s \in V(D)$ um vértice qualquer. O algoritmo DIJKSTRA(D, w, s) resolve o problema do Caminho Mínimo de Ùnica Fonte.*

Demonstração. Vamos denotar por $dist_i[v]$ o valor em $dist[v]$ na i-ésima iteração do laço **enquanto**. Note que $dist_i[v] \geq dist_j[v]$ para qualquer $v \in V(D)$ e $j > i$, pois só mudamos estimativas de distância quando elas diminuem, por meio de relaxação. Note ainda que qualquer arco de D é relaxado exatamente uma vez, no momento em que sua cauda é visitada.

Agora fixe um arco $xy \in E(D)$ qualquer e vamos mostrar que, ao final da execução, xy não poderá ser relaxado. Suponha que x é visitado na k-ésima iteração. Assim, logo após visitarmos x, teremos $dist_k[x] + w(xy) \geq dist_k[y]$.

Assim, se em alguma iteração $\ell > k$ tivermos $dist_\ell[x] + w(xy) < dist_\ell[y]$, deve ser porque $dist[x]$ mudou (e não $dist[y]$). Formalmente, visitamos algum vértice z e relaxamos o arco zx de forma que

$$dist_\ell[z] + w(zx) = dist_\ell[x] < dist_k[x]\,.$$

Mas note que, na k-ésima iteração, escolhemos x porque $dist_k[x] \leq dist_k[v]$ para todo $v \in V(D)$ não visitado. Particularmente, z não estava visitado, de forma que $dist_k[x] \leq dist_k[z]$, o que implica que $dist_k[x] \leq dist_\ell[z]$. (Se tivermos $dist_k[x] > dist_\ell[z]$, então como $dist_\ell[z] \geq dist_k[z]$, teríamos $dist_k[x] > dist_k[z]$, o que não é

verdade.) Mas então, usando a inequação acima e o fato que $w(zx) \geq 0$, temos

$$dist_\ell[z] \leq dist_\ell[z] + w(zx) < dist_k[x] \leq dist_\ell[z]\,,$$

o que é um absurdo.

Então nenhum arco pode ser relaxado ao final da execução, de forma que, pelo Teorema 12.4, $dist[v] = dist^w_D(s, v)$ para todo $v \in V(D)$ ao final. □

- Podemos fazer uma implementação direta do algoritmo apresentado, que a todo momento busca pelo menor valor armazenado no vetor *dist*.
  - Inicializar os vértices leva tempo $O(V)$, o laço **enquanto** irá executar $O(V)$ vezes, a linha 9 irá executar $O(V)$ vezes com cada execução levando tempo $O(V)$, e analisar todas arestas que saem de um vértice x leva tempo $O(d^+(x))$ sendo que fazemos isso para todos os vértices.
  - Assim, o algoritmo leva tempo $O(V^2 + E)$ ao todo.
- Porém, note que a operação mais custosa é a que procura pelo menor valor no vetor *dist*, e conhecemos uma estrutura de dados muito boa para fazer isso!
- Podemos fazer uso de uma Heap (binária):
  - Todos os vértices não visitados devem estar na Heap.
  - A prioridade de um vértice v é o valor em $dist[v]$ (multiplicado por -1, já que quanto menor o valor em *dist*, maior a prioridade do vértice).
- Assim, RemoveDaHeap devolve o próximo vértice que deve ser visitado.
- Com isso, o tempo de execução passa a ser $O(E \log V)$:
  - São $O(V)$ chamadas iniciais ao InsereNaHeap, cada uma com tempo $O(\log V)$;
  - São $O(V)$ chamadas ao RemoveDaHeap, cada uma com tempo $O(\log V)$;
  - São $O(E)$ chamadas ao Relaxa, que por sua vez chama o AlteraHeap, que leva tempo $O(\log V)$;
  - Então o tempo total é $O((V + E) \log V)$, que é $O(E \log V)$ quando $E \geq V$.

```
1: Função RelaxaDijkstra(x, y)
2:     Se dist[x] + w(xy) < dist[y] então                    ▷ Cuidado!!!
3:         dist[y] ← dist[u] + w(xy)
4:         pred[y] ← x
5:         AlteraHeap(H, v, −dist[v])
```

```
1: Função Dijkstra(D, w, s)
2:     Seja Q uma Heap
3:     Para cada v ∈ V(D) faça
4:         pred[v] ← −1
5:         dist[v] ← ∞
6:         InsereNaHeap(H, v, −dist[v])
7:     dist[s] ← 0
8:     pred[s] ← s
9:     AlteraHeap(H, s, 0)
10:    Enquanto Q ≠ ∅ faça
11:        x ← RemoveDaHeap(Q)
12:        Para cada y ∈ N⁺(x) faça
13:            RelaxaDijkstra(x, y)
```

12.3.3 Algoritmo de Bellman-Ford

- O algoritmo de Bellman-Ford resolve o problema dos Caminhos Mínimos de Única Fonte mesmo quando os arcos do digrafo de entrada têm peso negativo.
- Mais ainda, quando existe um ciclo de peso total negativo, o algoritmo identifica a existência de tal ciclo.
- A ideia principal é tentar, em $|V(D)| - 1$ iterações, melhorar a estimativa de distância conhecida a partir de s para todos os vértices v analisando todos os arcos de D em cada iteração.
  - O número $|V(D)| - 1$ não é mágico: qualquer caminho em um digrafo tem no máximo $|V(D)| - 1$ arestas.
- A intuição por trás dessa ideia é garantir que, dado um sv-caminho mínimo, o algoritmo relaxe os arcos desse caminho em ordem.

```
1: Função QuaseBellmanFord(D, w, s)
2:     Para cada v ∈ V(D) faça
3:         pred[v] ← −1
4:         dist[v] ← ∞
5:     dist[s] ← 0
6:     pred[s] ← s
7:     Para i ← 1 até |V(D)| faça
8:         Para cada xy ∈ E(D) faça
9:             Relaxa(x, y)
```

Lema 12.7. *Seja D um digrafo, $w\colon E(D) \to \mathbb{R}$ e $s \in V(D)$ um vértice qualquer. Se D não possui ciclos de custo negativo, então ao final da execução de QuaseBellmanFord(D, w, s), para todo $v \in V(D)$ vale que:*

1. *$dist[v] = \infty$ se não há sv-caminho;*
2. *$dist[v] = dist^w_D(s, v)$ se há sv-caminho;*
3. *$dist[s] = 0$.*

Demonstração. Seja $v \in V(D)$ tal que não há sv-caminho, isto é, $dist^w_D(s, v) = \infty$. Pela Proposição 12.3, sabemos que $dist[v] \geq \infty$. Como não é possível $dist[v] > \infty$, só podemos ter $dist[v] = \infty$.

Agora seja $v \in V(D)$ tal que há sv-caminho, isto é, $dist^w_D(s, v) \neq \infty$. Seja $P = (v_0 = s, v_1, \ldots, v_k = v)$ um sv-caminho mínimo. Vamos provar por indução em i que $dist[v_i] \leq \sum_{j=1}^{i} w(v_{j-1}v_j)$ ao final da i-ésima iteração do laço **para** da linha 7.

Se $i = 0$, nenhuma iteração ocorreu. Note que $dist[s] = 0$ e, de fato, $\sum_{j=1}^{0} w(v_{j-1}v_j) = 0$. Agora seja $i > 0$ e suponha, por hipótese de indução, que logo após a $(i-1)$-ésima iteração, $dist[v_{i-1}] \leq \sum_{j=1}^{i-1} w(v_{j-1}v_j)$. Durante a i-ésima iteração, o algoritmo relaxa todas as arestas de D, em particular $v_{i-1}v_i$. Então, assim que essa aresta for relaxada temos $dist[v_i] \leq dist[v_{i-1}] + w(v_{i-1}v_i)$, o que implica

$$dist[v_i] \leq dist[v_{i-1}] + w(v_{i-1}v_i) \leq \sum_{j=1}^{i-1} w(v_{j-1}v_j) + w(v_{i-1}v_i) = \sum_{j=1}^{i} w(v_{j-1}v_j)\,.$$

Concluímos então que $dist[v_k = v] \leq dist^w_D(s, v)$ após a k-ésima iteração. Pela Proposição 12.3, sabemos que $dist[v] \geq dist^w_D(s, v)$ também. Logo, só pode ser o caso

de $dist[v] = dist^w_D(s, v)$. Como o laço executa $V - 1$ vezes e $k \leq V - 1$, o resultado segue.

Por fim, note que $dist[s] = 0$ antes do segundo laço **para** começar e que se esse valou mudou ao final é porque $dist[s] < 0$, já que só modificamos $dist$ por meio de relaxações. Mas pela Proposição 12.3, só podemos ter $dist[s] \geq 0$. Logo, de fato $dist[s] = 0$ ao fim. □

- Mas e com relação a ciclos de custo negativo?

Teorema 12.8. *Seja D um digrafo e $w\colon E(D) \to \mathbb{R}$. Se C é um ciclo de custo negativo em D, então C contém um arco que pode ser relaxado.*

- Por causa do teorema acima, depois das $|V(D)| - 1$ iterações, o algoritmo ainda verifica uma última vez se há arestas que podem ser relaxadas.
- A seguir temos a versão completa do algoritmo de Bellman-Ford, que devolve $Verdadeiro$ quando funciona e $Falso$ quando não funciona.

```
1:  Função BellmanFord(D, w, s)
2:      Para cada v ∈ V(D) faça
3:          pred[v] ← −1
4:          dist[v] ← ∞
5:      dist[s] ← 0
6:      pred[s] ← s
7:      Para i ← 1 até |V(D)| − 1 faça
8:          Para cada xy ∈ E(D) faça
9:              Relaxa(x, y)
10:     Para cada xy ∈ E(D) faça
11:         Se dist[x] + w(xy) < dist[y] então      ▷ arco xy pode ser relaxado
12:             Devolve Falso
13:     Devolve Verdadeiro
```

Teorema 12.9. *Seja D um digrafo, $w\colon E(D) \to \mathbb{R}$ e $s \in V(D)$ um vértice qualquer. Se D não possui ciclos com custo negativo, então o algoritmo BellmanFord(D, w, s) resolve o problema do Caminho Mínimo de Ùnica Fonte e devolve $Verdadeiro$. Caso contrário, o algoritmo devolve $Falso$.*

Demonstração. Primeiro suponha que D não tem ciclos com custo negativo. Vamos

provar que $dist[v] = dist^w_D(s, v)$ para todo $v \in V(D)$ ao final e que o algoritmo devolve $Verdadeiro$. Pelo Lema 12.7, sabemos que ao final do laço **para** da linha 7 vale que $dist[v] = dist^w_D(s, v)$ para qualquer $v \in V(D)$. Então o Teorema 12.4 nos permite concluir que nenhum arco pode ser relaxado, o que implica que a condição da linha 10 não será satisfeita e o algoritmo devolverá $Verdadeiro$.

Suponha agora que D tem ciclos com custo negativo. Neste caso, por causa do Teorema 12.8, sabemos que quando o último laço para executar, certamente algum arco xy poderá ser relaxado, fazendo com que a condição da linha 10 seja verdadeira, e o algoritmo devolverá $Falso$. □

- Note que o primeiro laço leva tempo $O(V)$, o segundo laço leva tempo $O(VE)$, e o último laço leva tempo $O(E)$, de forma que o tempo total do algoritmo de Bellman-Ford é $O(VE)$.

12.4 Caminhos Mínimos entre Todos os Pares

- Os algoritmos manterão, para cada par de vértices $u, v \in V(D)$:
  - $dist[u][v]$, para armazenar o custo de um uv-caminho encontrado pelo algoritmo;
  - $pred[u][v]$, para armazenar o predecessor de v no uv-caminho encontrado pelo algoritmo;

12.4.1 Algoritmo de Floyd-Warshall

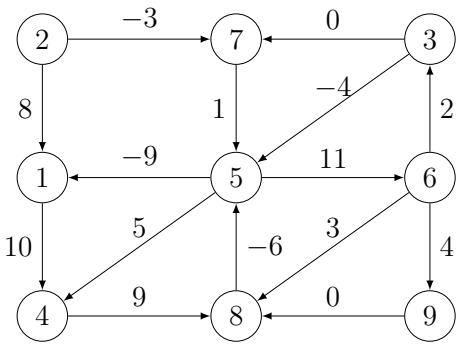
- O algoritmo de Floyd-Warshall calcula xy-caminhos mínimos por meio de *relaxações de caminhos*, isto é, verificando se é possível utilizar um novo vértice v para melhorar a estimativa de distância entre x e y:

```
1: Função RELAXA(x, y, v)
2:     Se dist[x][v] + dist[v][y] < dist[x][y] então        ▷ Cuidado!!! Se dist[x][v] ≠ ∞
3:         dist[x][y] ← dist[x][v] + dist[v][y]
4:         pred[x][y] ← pred[v][y]
```

- Nesta seção, vamos considerar que $V(D) = \{1, 2, \ldots, V\}$.

```
1: Função QuaseFloydWarshall(D, w)
2:     Para k ← 1 até V faça
3:         Para i ← 1 até V faça
4:             Para j ← 1 até V faça
5:                 Relaxa(i, j, k)
```

- Defina $V_k = \{1, 2, \ldots, k\}$. Assim, $V_0 = \emptyset$.
- Vamos definir um *V_k-caminho* como sendo um caminho cujos vértices internos estão em V_k apenas.
  - Por exemplo, no digrafo a seguir, todos os V_5-caminhos entre 3 e 8 são: $(3, 5, 1, 4, 8)$ e $(3, 5, 4, 8)$.
  - Todos os V_6-caminhos entre 3 e 8 são $(3, 5, 1, 4, 8)$, $(3, 5, 4, 8)$ e $(3, 5, 6, 8)$.
  - Não existem V_4-caminhos entre 3 e 8.
- Vamos definir d_{ij}^k como sendo o custo de um V_k-caminho mínimo entre i e j.
  - $(3, 5, 1, 4, 8)$ é um V_6-caminho mínimo entre 3 e 8 e $d_{38}^6 = 6$.



Teorema 12.10. *Seja D um digrafo e $w\colon E(D) \to \mathbb{R}$ tal que D não tem ciclo de custo negativo. Para qualquer par $i, j \in V(D)$ vale que*

$$d_{ij}^k = \begin{cases} 0 & \text{se } k = 0 \text{ e } i = j \\ w(ij) & \text{se } k = 0 \text{ e } ij \in E(D) \\ \infty & \text{se } k = 0 \text{ e } ij \notin E(D) \\ \min\{d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\} & \text{se } k > 0\,. \end{cases}$$

Demonstração. O resultado claramente vale quando $k = 0$. Vamos considerar então que $k \geq 1$.

Seja P um V_k-caminho mínimo entre i e j (então $w(P) = d_{ij}^k$). Note que há apenas

duas possibilidades: $k \in V(P)$ ou $k \notin V(P)$.

Se $k \notin V(P)$, então observe que os vértices internos de P estão em V_{k-1}. De fato, podemos afirmar que P é um V_{k-1}-caminho mínimo entre i e j. Se não fosse, haveria um V_{k-1}-caminho mínimo P' entre i e j tal que $w(P') < w(P)$, que claramente é uma contradição pois P' é um V_k-caminho entre i e j. Neste caso, concluímos que $w(P) = d_{ij}^{k-1}$.

Se $k \in P$, então $P = (i, \ldots, k, \ldots, j)$ pode ser dividido em dois caminhos $P_1 = (i, \ldots, k)$ e $P_2 = (k, \ldots, j)$. Note que P_1 é um V_{k-1}-caminho entre i e k e P_2 é um V_{k-1}-caminho entre k e j. De fato, podemos afirmar que P_1 é um V_{k-1}-caminho mínimo entre i e k e P_2 é um V_{k-1}-caminho mínimo entre k e j. Se P_1 não fosse um tal caminho, então haveria um V_{k-1}-caminho mínimo P_1' entre i e k tal que $w(P_1') < w(P_1)$ de forma que P_1' seguido de P_2 seria um V_k-passeio entre i e j de custo menor do que $w(P)$ e que conteria um V_k-caminho entre i e j de custo possivelmente ainda menor (Lema 12.2), o que é uma contradição. O mesmo vale se P_2 não fosse um tal caminho. Neste caso, concluímos que $w(P) = d_{ik}^{k-1} + d_{kj}^{k-1}$. □

```
1:  Função FloydWarshall(D, w)
2:      Para i ← 1 até V faça
3:          pred[i][i] ← i
4:          dist[i][i] ← 0
5:          Para j ← 1 até V faça
6:              Se ij ∈ E(D) então
7:                  pred[i][j] ← i
8:                  dist[i][j] ← w(ij)
9:              Senão
10:                 pred[i][j] ← −1
11:                 dist[i][j] ← ∞
12:     Para k ← 1 até V faça
13:         Para i ← 1 até V faça
14:             Para j ← 1 até V faça
15:                 Relaxa(i, j, k)
16:     Para i ← 1 até V faça
17:         Se dist[i][i] < 0 então
18:             Devolve Falso
19:     Devolve Verdadeiro
```

Teorema 12.11. *Seja D um digrafo e $w\colon E(D) \to \mathbb{R}$. Se D não tem ciclos de custo negativo, então ao final da execução do laço* ***para*** *da linha 12 temos $dist[i][j] = d^V_{ij}$.*

Demonstração. Provaremos por indução em k que, após a k-ésima iteração do laço **para** da linha 12, temos $dist[i][j] = d^k_{ij}$ para todo par $i, j \in V(D)$. Isso implica diretamente no resultado desejado.

Se $k = 0$, então não houve ainda iteração do laço e de fato temos $dist[i][j] = d^0_{ij}$ para todo par $i, j \in V(D)$.

Suponha então $k > 1$ e suponha que $dist[u][v] = d^{k-1}_{uv}$ para todo par $u, v \in V(D)$. Note que durante a execução, o algoritmo faz

$$dist[i][j] = \min\{dist[i][j], dist[i][k] + dist[k][j]\}\,.$$

Como, por hipótese, $dist[i][j] = d^{k-1}_{ij}$ (antes da mudança), $dist[i][k] = d^{k-1}_{ik}$ e $dist[k][j] = d^{k-1}_{kj}$, então o resultado segue devido ao Teorema 12.10. □

Teorema 12.12. *Seja D um digrafo e $w\colon E(D) \to \mathbb{R}$. Seja $dist$ a matriz construída pelo algoritmo de Floyd-Warshall quando executado sobre D, w. Existe $dist[i][i] < 0$ se e somente se D possui um ciclo com custo negativo.*

Teorema 12.13. *Seja D um digrafo, $w\colon E(D) \to \mathbb{R}$. Se D não possui ciclos com custo negativo, então o algoritmo* FloydWarshall*(D, w) resolve o problema do Caminho Mínimo entre Todas as Fontes e devolve $Verdadeiro$. Caso contrário, o algoritmo devolve $Falso$.*

Demonstração. Primeiro suponha que D não tem ciclos com custo negativo. Então pelo Teorema 12.11, sabemos que ao final da execução do laço **para** da linha 12 temos $dist[i][j] = dist^w_D(i, j)$. O Teorema 12.12 nos permite concluir que $dist[i][i] \geq 0$ para todo $i \in V(D)$, o que significa que a condição da linha 17 não será satisfeita e o algoritmo devolverá $Verdadeiro$.

Suponha agora que D tem ciclos com custo negativo. Nesse caso, o Teorema 12.12 nos permite concluir que existe algum $i \in V(D)$ tal que $dist[i][i] < 0$, o que significa que a condição da linha 17 será satisfeita e o algoritmo irá devolver $Falso$. □

- Note que o primeiro laço leva tempo $O(V^2+E) = O(V^2)$, o segundo laço leva tempo $O(V^3)$ e o último laço leva tempo $O(V)$, de forma que o tempo total do algoritmo de Floyd-Warshall é $O(V^3)$.