Projeto e Análise de Algoritmos II (MC558)
Caminhos mínimos

Prof. Dr. Ruben Interian

Resumo

Resumo

Revisão do conteúdo

- A distância $dist(s, v)$ é o comprimento (**número de arestas**) do caminho mais curto de s a v.
- Se v não é alcançável a partir de s, $dist(s, v) = \infty$.
- O algoritmo de busca em grafos chamado **busca em largura** usa uma fila Q, e retorna uma **árvore de busca** e as distâncias dos vértices ao vértice inicial s.

Objetivo

- Dado um grafo ponderado G e dois vértices s e t, encontrar um caminho de peso mínimo de s a t.
- Dado um grafo ponderado G, e um vértice s, encontrar caminhos de peso mínimo de s até todos os vértices $v \in V$.

Resumo

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 arco (u, v).

Vamos tentar resolver **dois 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 de fonte única:**
   Dado s, encontrar caminhos de peso mínimo de s até todos os vértices $v \in V$.
   (inclui o problema anterior)

Representando caminhos mínimos

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

- Associamos a cada $v \in V$ um predecessor $\pi[v]$.
- O vetor π representa 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.
- As distâncias dos vértices ao vértice inicial s são calculadas no vetor d (mas podem ser recuperadas a partir de π).

Definição

Problema dos caminhos mínimos com a mesma origem

Entrada:

- Grafo direcionado G, função de pesos w nos arcos, vértice de origem s.

Saída:

- Vetor $d[v]$ de distâncias para $v \in V$, vetor π definindo uma árvore de caminhos mínimos.

Subestrutura ótima de caminhos mínimos

Seja G um grafo direcionado e ponderado, e seja um caminho mínimo de v_1 a v_k:

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

Então para quaisquer vértices v_i, v_j, com $1 \leq i \leq j \leq k$, o subcaminho

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

é um caminho mínimo de v_i a v_j.

Subestrutura ótima de caminhos mínimos

Seja G um grafo direcionado e ponderado, e seja um caminho mínimo de v_1 a v_k:

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

Então para quaisquer vértices v_i, v_j, com $1 \leq i \leq j \leq k$, o subcaminho

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

é um caminho mínimo de v_i a v_j.

Essa propriedade vale para **todos** os grafos direcionados ponderados?

Subestrutura ótima de caminhos mínimos

Seja G um grafo direcionado e ponderado, e seja um caminho mínimo de v_1 a v_k:

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

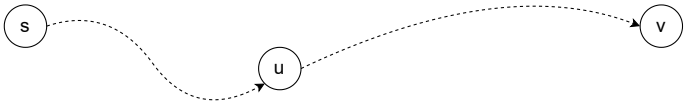
Então para quaisquer vértices v_i, v_j, com $1 \leq i \leq j \leq k$, o subcaminho

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

é um caminho mínimo de v_i a v_j.

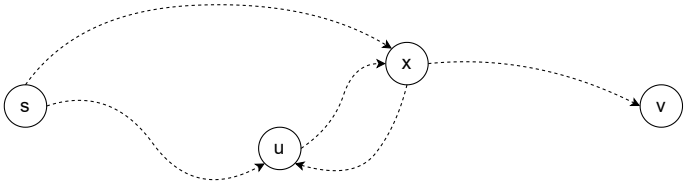
Essa propriedade vale para **todos** os grafos direcionados ponderados? – Não.

Subestrutura ótima de caminhos mínimos: intuição



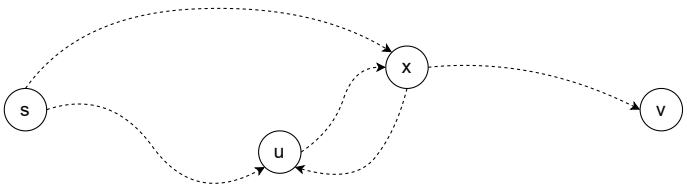
Vamos supor que o caminho P de s a v é mínimo, mas o subcaminho de s a u não é. Então existe um caminho de s a u de peso menor.

Subestrutura ótima de caminhos mínimos: intuição



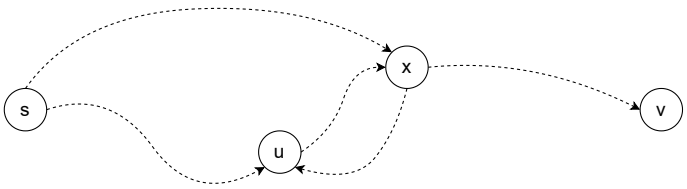
Vamos supor que o caminho P de s a v é mínimo, mas o subcaminho de s a u não é. Então existe um caminho de s a u de peso menor. Veja que ele precisa passar por P (caso contrário, haveria também um caminho menor de s a v).
Seja um vértice x que está nesse caminho e em P.

Subestrutura ótima de caminhos mínimos: intuição



Vamos supor que o caminho P de s a v é mínimo, mas o subcaminho de s a u não é.
Então existe um caminho de s a u de peso menor. Veja que ele precisa passar por P (caso contrário, haveria também um caminho menor de s a v).
Seja um vértice x que está nesse caminho e em P.
Existe um ciclo que contém os vértices u e x. O seu custo é ≤ 0 (por quê?).

Subestrutura ótima de caminhos mínimos: intuição

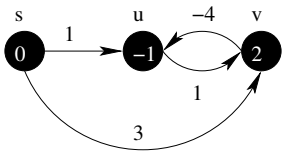


Vamos supor que o caminho P de s a v é mínimo, mas o subcaminho de s a u não é.
Então existe um caminho de s a u de peso menor. Veja que ele precisa passar por P (caso contrário, haveria também um caminho menor de s a v).
Seja um vértice x que está nesse caminho e em P.
Existe um ciclo que contém os vértices u e x. O seu custo é ≤ 0 (por quê?).

A propriedade da subestrutura ótima não vale para grafos direcionados que contêm ciclos de custo negativo.

Subestrutura ótima de caminhos mínimos: exemplo

A propriedade da subestrutura ótima **não vale** se o grafo tiver ciclos negativos:



- (s, u, v) é um caminho mínimo de s a v com peso $1 + 1 = 2$.
- (s, u), com peso 1, **não** é um caminho mínimo de s a u.
- (s, v, u) é um caminho mínimo de s a u com peso $3 - 4 = -1$.

Algoritmos baseados em relaxação

Todos os algoritmos que iremos estudar:

1. Inicializam d e π usando o método **InitializeSingleSource**.
2. Alteram d e π apenas usando o método **Relax** (relaxação).

Nesses algoritmos:

- Se $d[v] \neq \infty$, existe um caminho de s a v com peso $d[v]$.
- Esse caminho pode ser recuperado por meio de π.
- $d[v]$ é a estimativa atual para $\text{dist}(s, v)$, veja que $d[v] \geq \text{dist}(s, v)$.
- Queremos no final da execução $d[v] = \text{dist}(s, v)$.

Inicialização

InitializeSingleSource (G, s)

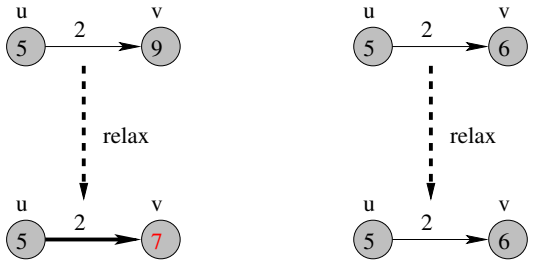
1: **para cada** $v \in V$ **faça**
2: $\quad d[v] \leftarrow \infty$
3: $\quad \pi[v] \leftarrow \text{NIL}$
4: $d[s] \leftarrow 0$

Relaxação

A relaxação tenta melhorar $d[v]$ passando por (u, v).

Relax (u, v, w)

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

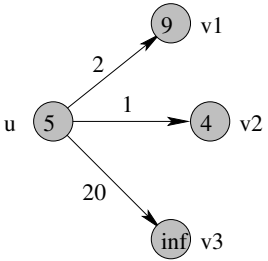


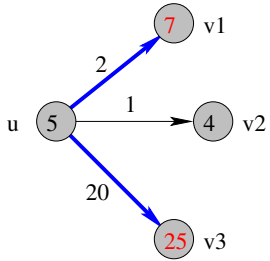
Relaxação

Se $u \in V$, para cada $v \in Adj[u]$ podemos aplicar **Relax**(u, v, w).

Relax (u, v, w)

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





Casos do problema de caminhos mínimos

Veremos três algoritmos baseados em relaxação para casos diferentes de instâncias (grafos G com pesos w nos arcos) do problema de caminhos mínimos:

1. O grafo G é direcionado acíclico: aplicação da ordenação topológica.
2. Não há arcos de peso negativo em w: algoritmo de Dijkstra.
3. Há arcos de peso negativo em w, mas não há ciclos negativos: algoritmo de Bellman-Ford.

Caminhos mínimos em grafos direcionados acíclicos

Entrada:

- Grafo direcionado acíclico G, função de peso w nos arcos, origem s.

Saída:

- Vetor $d[v]$ de distâncias para $v \in V$, vetor π definindo uma árvore de caminhos mínimos.

Caminhos mínimos em grafos direcionados acíclicos

Entrada:

- Grafo direcionado acíclico G, função de peso w nos arcos, origem s.

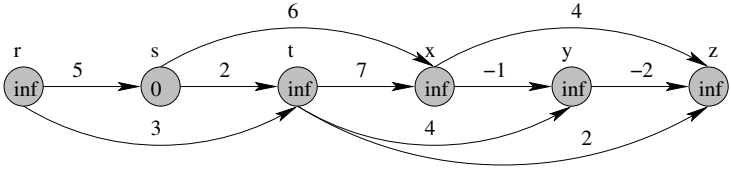
Saída:

- Vetor $d[v]$ de distâncias para $v \in V$, vetor π definindo uma árvore de caminhos mínimos.

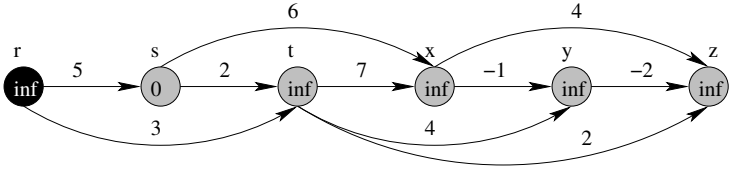
DagShortestPaths (G, w, s)

1: Ordene topologicamente os vértices de G
2: **InitializeSingleSource**(G, s)
3: **para cada** $u \in V$ na ordem topológica **faça**
4: **para cada** $v \in Adj[u]$ **faça**
5: **Relax**(u, v, w)
devolva d, π

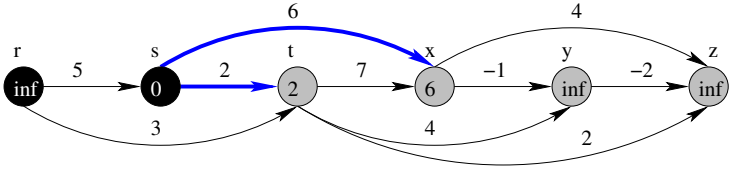
Exemplo



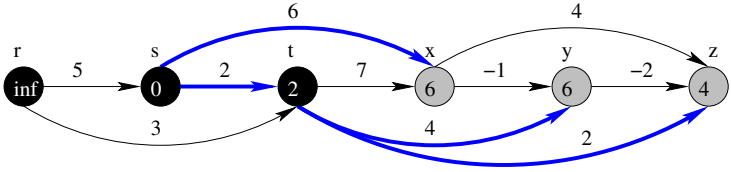
Exemplo



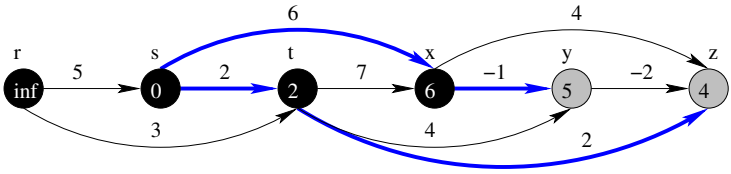
Exemplo



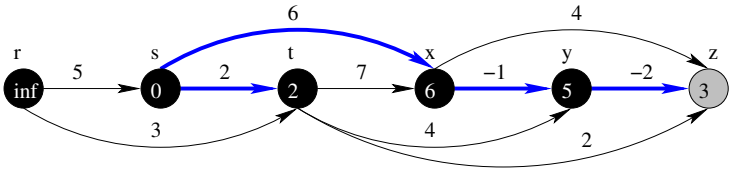
Exemplo



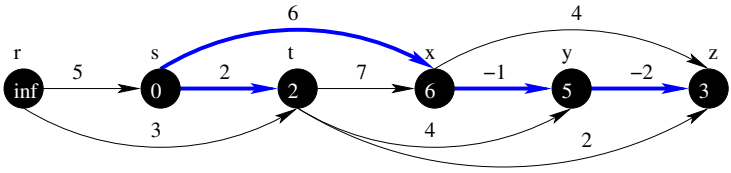
Exemplo



Exemplo



Exemplo



Complexidade

DagShortestPaths (G, w, s)

1: Ordene topologicamente os vértices de G
2: **InitializeSingleSource**(G, s)
3: **para cada** $u \in V$ na ordem topológica **faça**
4: **para cada** $v \in Adj[u]$ **faça**
5: **Relax**(u, v, w)
devolva d, π

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

Complexidade: $O(V + E)$.

Correção

Correção do **DagShortestPaths**:

- Vamos usar algumas propriedades dos algoritmos baseados em relaxação.
- Elas também são úteis para analisar os outros algoritmos para o problema.

Propriedade de algoritmos baseados em relaxação

Ao longo da execução de **um algoritmo** baseado em relaxação sempre valem as propriedades:

- **Limite superior**
  $d[v] \geq \text{dist}(s, v)$ e, se $d[v]$ alcança o valor $\text{dist}(s, v)$, nunca mais muda. $\rightarrow$ C24.11

Propriedade de algoritmos baseados em relaxação

Ao longo da execução de **um algoritmo** baseado em relaxação sempre valem as propriedades:

- **Limite superior**
  $d[v] \geq \text{dist}(s, v)$ e, se $d[v]$ alcança o valor $\text{dist}(s, v)$, nunca mais muda. $\rightarrow$ C24.11
- **Inexistência de caminho**
  Se não existe nenhum caminho de s a v, então $d[v] = \infty$. $\rightarrow$ C24.12

Propriedade de algoritmos baseados em relaxação

Ao longo da execução de **um algoritmo** baseado em relaxação sempre valem as propriedades:

- **Limite superior**
  $d[v] \geq \text{dist}(s, v)$ e, se $d[v]$ alcança o valor $\text{dist}(s, v)$, nunca mais muda. → C24.11
- **Inexistência de caminho**
  Se não existe nenhum caminho de s a v, então $d[v] = \infty$. → C24.12
- **Subgrafo de predecessores**
  Se $d[v] = \text{dist}(s, v)$ para todo $v \in V$, então o subgrafo induzido por π ("subgrafo dos predecessores") é uma árvore de caminhos mínimos. → C24.17

Algoritmos baseados em relaxação

- **Convergência**
  Se P é um caminho mínimo de s a v terminando no arco (u, v) e $d[u] = \text{dist}(s, u)$, então ao relaxar (u, v), $d[v] = \text{dist}(s, v)$, e $d[v]$ nunca mais muda. $\rightarrow$ C24.14

Algoritmos baseados em relaxação

- **Convergência**
  Se P é um caminho mínimo de s a v terminando no arco (u, v) e $d[u] = \text{dist}(s, u)$, então ao relaxar (u, v), $d[v] = \text{dist}(s, v)$, e $d[v]$ nunca mais muda. $\rightarrow$ C24.14

- **Relaxamento de caminho**
  Se $P = (v_0, v_1, \ldots, v_k)$ é um caminho mínimo de $s = v_0$ a v_k e relaxamos os arcos de P na ordem (v_0, v_1), (v_1, v_2), $\ldots$, (v_{k-1}, v_k), então $d[v_k] = \text{dist}(s, v_k)$, independente de quaisquer outras relaxações realizadas. $\rightarrow$ C24.15

Correção de DagShortestPaths

Correção do algoritmo **DagShortestPaths**:

- Seja v um vértice e suponha que $P = (v_0, v_1, \ldots, v_k)$ é um caminho mínimo de $s = v_0$ a $v = v_k$.
- Como v_0, v_1, ..., v_k aparecem nessa mesma ordem na ordenação topológica, os arcos (v_0, v_1), $\ldots$, (v_{k-1}, v_k) são relaxadas nessa ordem.

Correção de DagShortestPaths

Correção do algoritmo **DagShortestPaths**:

- Seja v um vértice e suponha que $P = (v_0, v_1, \ldots, v_k)$ é um caminho mínimo de $s = v_0$ a $v = v_k$.
- Como v_0, v_1, ..., v_k aparecem nessa mesma ordem na ordenação topológica, os arcos (v_0, v_1), $\ldots$, (v_{k-1}, v_k) são relaxadas nessa ordem.
- A propriedade do **relaxamento de caminho** garante que o algoritmo computa corretamente $d[v] = \text{dist}(s, v)$ para cada $v \in V$.

Correção de DagShortestPaths

Correção do algoritmo **DagShortestPaths**:

- Seja v um vértice e suponha que $P = (v_0, v_1, \ldots, v_k)$ é um caminho mínimo de $s = v_0$ a $v = v_k$.
- Como v_0, v_1, ..., v_k aparecem nessa mesma ordem na ordenação topológica, os arcos (v_0, v_1), $\ldots$, (v_{k-1}, v_k) são relaxadas nessa ordem.
- A propriedade do **relaxamento de caminho** garante que o algoritmo computa corretamente $d[v] = \text{dist}(s, v)$ para cada $v \in V$.

Pela propriedade do **subgrafo de predecessores**, o vetor π que devolve o algoritmo define uma árvore de caminhos mínimos.

Perguntas

1. Como encontrar um caminho **de peso máximo** de s a t em um grafo direcionado acíclico e ponderado G?

Perguntas

1. Como encontrar um caminho **de peso máximo** de s a t em um grafo direcionado acíclico e ponderado G?

2. Como encontrar o caminho mínimo de s a t **em tempo linear** para um grafo direcionado em que todos os arcos têm o mesmo peso $C > 0$?

Algoritmo de Dijkstra

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

O algoritmo foi proposto por **Edsger Dijkstra**, cientista da computação holandês, conhecido por suas contribuições nas áreas de algoritmos, de linguagens de programação, sistemas operacionais (sistema de semáforos), Prêmio Turing em 1972.

Algoritmo de Dijkstra

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

O algoritmo foi proposto por **Edsger Dijkstra**, cientista da computação holandês, conhecido por suas contribuições nas áreas de algoritmos, de linguagens de programação, sistemas operacionais (sistema de semáforos), Prêmio Turing em 1972.

> *Ciência da computação tem tanto a ver com o computador como a Astronomia com o telescópio, a Biologia com o microscópio, ou a Química com os tubos de ensaio. A Ciência não estuda ferramentas, mas o que fazemos e o que descobrimos com elas.*
>
> Edsger Dijkstra

Algoritmo de Dijkstra

Entrada:

- Grafo direcionado G, função de peso $w \geq 0$ nos arcos, origem s.

Saída:

- Vetor $d[v]$ de distâncias para $v \in V$, vetor π definindo uma árvore de caminhos mínimos.

Algoritmo de Dijkstra

Ideia do algoritmo de Dijkstra:

- A cada momento durante a execução, mantemos um conjunto S é formado pelos vértices mais próximos de s (na primeira iteração $S = \{s\}$).
- A cada momento, a arvore de caminhos mínimos (em construção) é formada por vértices de S.
- Em cada iteração, a ideia é estender o conjunto S cada vez mais adicionando a ele o vértice em $V \setminus S$ que esteja mais próximo de s.

O "detalhe" é como encontrar de forma eficiente este vértice mais próximo de s.

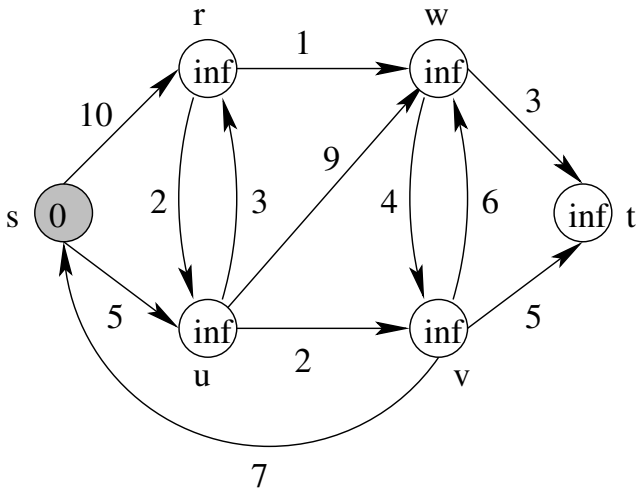
Algoritmo de Dijkstra

Dijkstra (G, w, s)

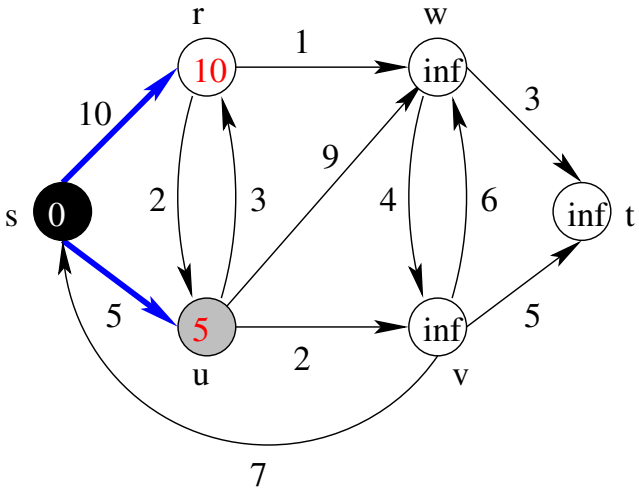
1: **InitializeSingleSource**(G, s)
2: $S \leftarrow \emptyset$
3: $Q \leftarrow$ **Build**(V)
4: **enquanto** $Q \neq \emptyset$ **faça**
5: $\quad u \leftarrow$ **ExtractMin**(Q)
6: $\quad S \leftarrow S \cup \{u\}$
7: $\quad$ **para cada** $v \in Adj[u]$ **faça**
8: $\quad\quad$ **Relax**(u, v, w)
devolva d, π

Usamos uma fila de prioridade Q com chave d. O conjunto S simplifica a análise, mas não é realmente necessário para o funcionamento do algoritmo.

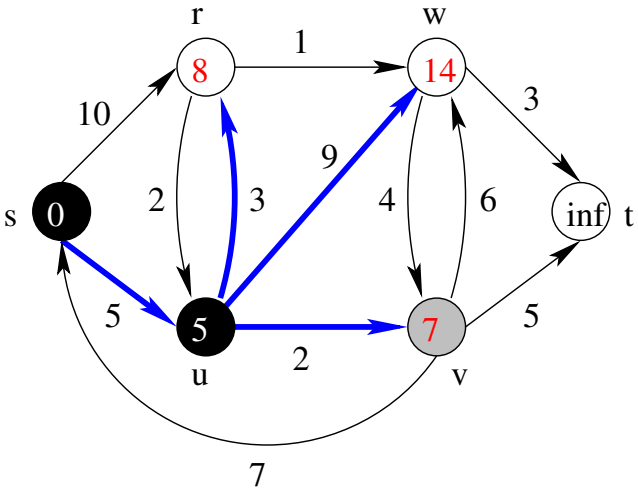
Exemplo



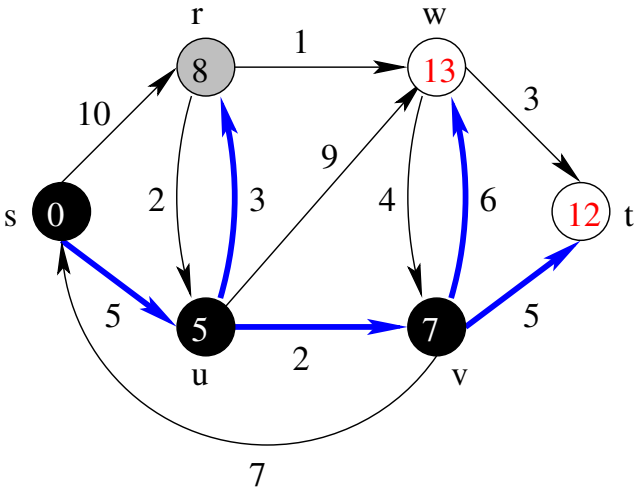
Exemplo



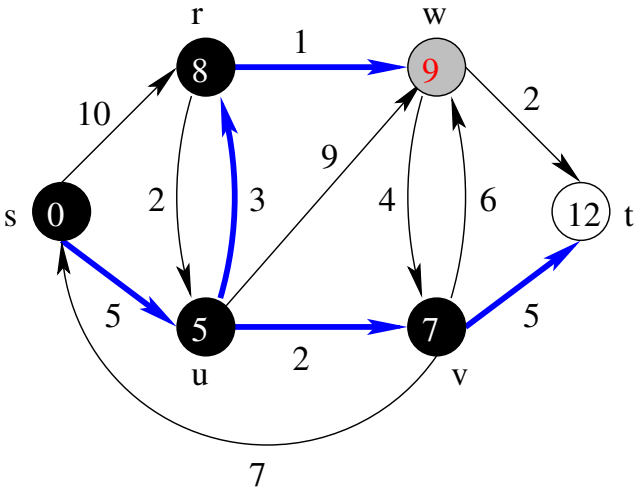
Exemplo



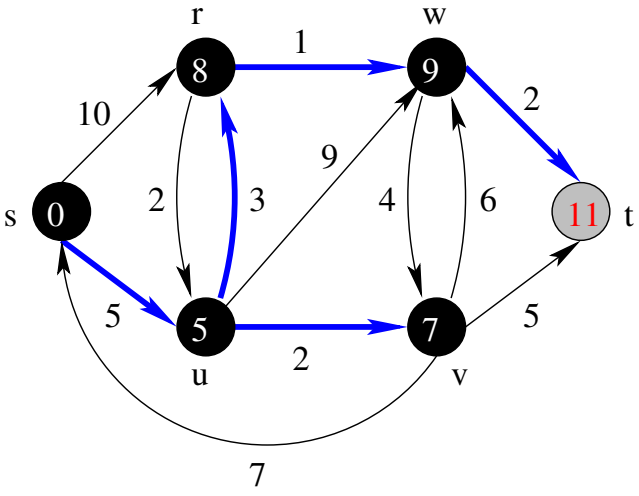
Exemplo



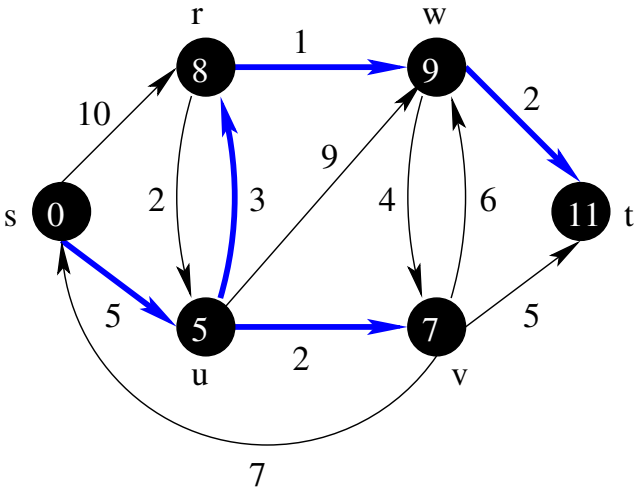
Exemplo



Exemplo



Exemplo



Correção do algoritmo de Dijkstra

Correção: precisamos provar que, quando o algoritmo termina:

1. $d[v] = \text{dist}(s, v)$ para todo $v \in V$;
2. π induz uma árvore de caminhos mínimos.

Correção do algoritmo de Dijkstra

Correção: precisamos provar que, quando o algoritmo termina:

1. $d[v] = \text{dist}(s, v)$ para todo $v \in V$;
2. π induz uma árvore de caminhos mínimos.

Veja que **Dijkstra** é baseado em relaxação.

- Na prática, precisamos mostrar apenas que, de fato, $d[v] = \text{dist}(s, v)$.
- Se provarmos o ponto anterior, pela propriedade do **subgrafo de predecessores**, o vetor π induz uma árvore de caminhos mínimos de s a cada vértice $v \in V$.

Correção do algoritmo de Dijkstra: Demonstração

Iremos mostrar por indução que no início de cada iteração, ao avaliar a condição de parada na linha 4 do algoritmo, **para cada** $x \in S$, $d[x] = \text{dist}(s, x)$.

Correção do algoritmo de Dijkstra: Demonstração

Iremos mostrar por indução que no início de cada iteração, ao avaliar a condição de parada na linha 4 do algoritmo, **para cada** $x \in S$, $d[x] = \text{dist}(s, x)$.

- **Inicialização**
  No início, $S = \emptyset$, então não há nada a provar.

Correção do algoritmo de Dijkstra: Demonstração

Iremos mostrar por indução que no início de cada iteração, ao avaliar a condição de parada na linha 4 do algoritmo, **para cada** $x \in S$, $d[x] = \text{dist}(s, x)$.

- **Inicialização**
  No início, $S = \emptyset$, então não há nada a provar.
- **Manutenção**
  - Vamos supor que a proposição era válida para S no início da iteração anterior. Na nova iteração, **Dijkstra** escolhe um vértice u com menor $d[u]$ em Q e adiciona a S.

Correção do algoritmo de Dijkstra: Demonstração

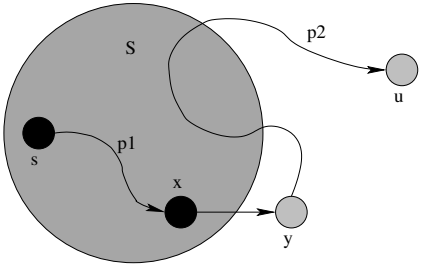
Iremos mostrar por indução que no início de cada iteração, ao avaliar a condição de parada na linha 4 do algoritmo, **para cada** $x \in S$, $d[x] = \text{dist}(s, x)$.

- **Inicialização**
  No início, $S = \emptyset$, então não há nada a provar.
- **Manutenção**
  - Vamos supor que a proposição era válida para S no início da iteração anterior. Na nova iteração, **Dijkstra** escolhe um vértice u com menor $d[u]$ em Q e adiciona a S.
  - Queremos mostrar que a proposição vale para $S \cup \{u\}$. Basta verificar então que neste instante $d[u] = \text{dist}(s, u)$.

Correção do algoritmo de Dijkstra: Demonstração

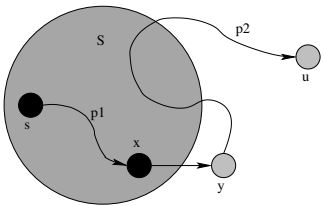
Seja:

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



Correção do algoritmo de Dijkstra: Demonstração

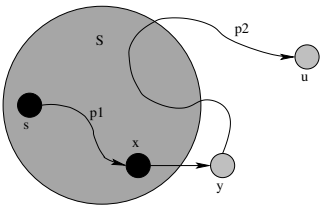
Queremos mostrar que $d[u] = \text{dist}(s, u)$.
Vamos supor, **por contradição**, que $d[u] > \text{dist}(s, u)$.



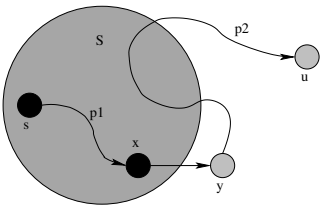
Correção do algoritmo de Dijkstra: Demonstração

Queremos mostrar que $d[u] = \text{dist}(s, u)$.
Vamos supor, **por contradição**, que $d[u] > \text{dist}(s, u)$.

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



Correção do algoritmo de Dijkstra: Demonstração

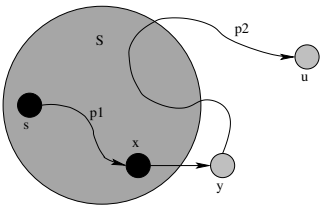


Queremos mostrar que $d[u] = \text{dist}(s, u)$.
Vamos supor, **por contradição**, que $d[u] > \text{dist}(s, u)$.

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

$$d[y] \quad \leq$$

Correção do algoritmo de Dijkstra: Demonstração

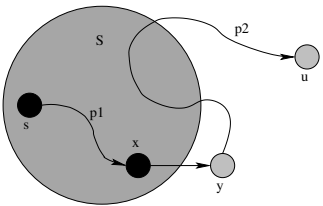


Queremos mostrar que $d[u] = \text{dist}(s, u)$.
Vamos supor, **por contradição**, que $d[u] > \text{dist}(s, u)$.

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

$$d[y] \leq d[x] + w(x, y) \quad \text{(pois já relaxamos } (x, y)\text{)}$$

Correção do algoritmo de Dijkstra: Demonstração

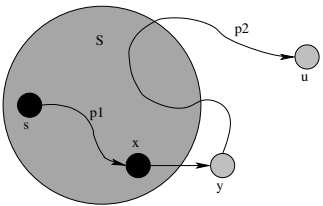


Queremos mostrar que $d[u] = \text{dist}(s, u)$.
Vamos supor, **por contradição**, que $d[u] > \text{dist}(s, u)$.

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

$$\begin{aligned} d[y] &\leq d[x] + w(x, y) \quad \text{(pois já relaxamos } (x, y)\text{)} \\ &= \text{dist}(s, x) + w(x, y) \quad \text{(invariante)} \end{aligned}$$

Correção do algoritmo de Dijkstra: Demonstração

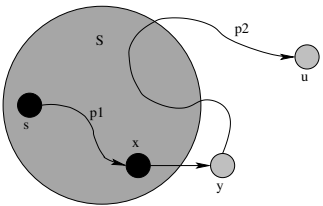


Queremos mostrar que $d[u] = \text{dist}(s, u)$.
Vamos supor, **por contradição**, que $d[u] > \text{dist}(s, u)$.

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

$$\begin{aligned} d[y] &\leq d[x] + w(x, y) \quad \text{(pois já relaxamos } (x, y)) \\ &= \text{dist}(s, x) + w(x, y) \quad \text{(invariante)} \\ &\leq \text{dist}(s, x) + w(x, y) + w(P_2) \end{aligned}$$

Correção do algoritmo de Dijkstra: Demonstração

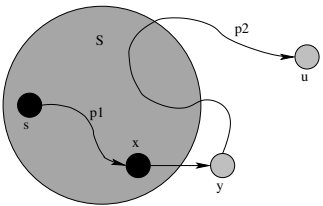


Queremos mostrar que $d[u] = \text{dist}(s, u)$.
Vamos supor, **por contradição**, que $d[u] > \text{dist}(s, u)$.

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

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

Correção do algoritmo de Dijkstra: Demonstração

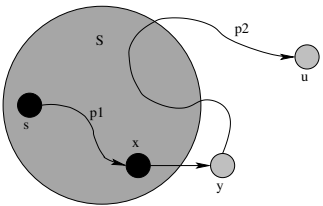


Queremos mostrar que $d[u] = \text{dist}(s, u)$.
Vamos supor, **por contradição**, que $d[u] > \text{dist}(s, u)$.

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

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

Correção do algoritmo de Dijkstra: Demonstração

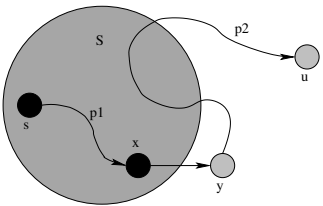


Queremos mostrar que $d[u] = \text{dist}(s, u)$.
Vamos supor, **por contradição**, que $d[u] > \text{dist}(s, u)$.

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

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

Correção do algoritmo de Dijkstra: Demonstração



Queremos mostrar que $d[u] = \text{dist}(s, u)$.
Vamos supor, **por contradição**, que $d[u] > \text{dist}(s, u)$.

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

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

- Mas daí $d[y] < d[u]$, o que contraria a escolha de u.

Então, na verdade, $d[u] \leq \text{dist}(s, u)$. Concluímos que $d[u] = \text{dist}(s, u)$.

Correção do algoritmo de Dijkstra: Demonstração

Para terminar a demonstração, veja que:

- Pela propriedade de **inexistência de caminho**, se um vértice v não é alcançável, então $d[v] = \infty$.

Correção do algoritmo de Dijkstra: Demonstração

Para terminar a demonstração, veja que:

- Pela propriedade de **inexistência de caminho**, se um vértice v não é alcançável, então $d[v] = \infty$.
- Portanto, para todo $v \in V$, $d[v] = \text{dist}(s, v)$. ■

Dijkstra precisa de arcos com peso não negativo

Lembre que tínhamos assumido que não há arcos de peso negativo.

- Se há arcos de peso negativo, o algoritmo de **Dijkstra** pode falhar.
- **Exercício**: Encontre um grafo com arcos negativos para o qual **Dijkstra** não funciona.

Dijkstra precisa de arcos com peso não negativo

Lembre que tínhamos assumido que não há arcos de peso negativo.

- Se há arcos de peso negativo, o algoritmo de **Dijkstra** pode falhar.
- **Exercício**: Encontre um grafo com arcos negativos para o qual **Dijkstra** não funciona.

  Dica: existe um exemplo com apenas 4 vértices, apenas um arco negativo e sem ciclos de peso negativo.

Complexidade de tempo

Dijkstra (G, w, s)

1: **InitializeSingleSource**(G, s)
2: $S \leftarrow \emptyset$
3: $Q \leftarrow$ **Build**(V)
4: **para cada** $Q \neq \emptyset$ **faça**
5: $\quad u \leftarrow$ **ExtractMin**(Q)
6: $\quad S \leftarrow S \cup \{u\}$
7: $\quad$ **para cada** $v \in Adj[u]$ **faça**
8: $\quad\quad$ **Relax**(u, v, w) $\quad \triangleright$ Operação DecreaseKey
devolva d, π

Depende de como a fila de prioridade Q, com as operações **Insert**, **ExtractMin**, **DecreaseKey**, é implementada.

Complexidade do algoritmo de Dijkstra

- **InitializeSingleSource** é $O(V)$
- **Build**(V) são $|V|$ vezes **Insert**.
- **Relax** possui uma chamada a **DecreaseKey**.

Complexidade do algoritmo de Dijkstra

- **InitializeSingleSource** é $O(V)$
- **Build**(V) são $|V|$ vezes **Insert**.
- **Relax** possui uma chamada a **DecreaseKey**.

Linha(s)	Tempo total
1–3	$\|V\|$ chamadas a **Insert**
5	$\|V\|$ chamadas a **ExtractMin**
8	$O(E)$ chamadas a **DecreaseKey**

Complexidade de Dijkstra: $O(|V| \times \textbf{Insert} + |V| \times \textbf{ExtractMin} + |E| \times \textbf{DecreaseKey})$.

Complexidade de tempo

Total: $O(|V| \times \textbf{Insert} + |V| \times \textbf{ExtractMin} + |E| \times \textbf{DecreaseKey})$.

Tipo de fila	**Insert**	**ExtractMin**	**DecreaseKey**	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	$O(1)$	$O(\log V)$	$O(1)$	$O(V \log V + E)$

Resumo

Síntese

- Podemos resolver o problema do **caminho mínimo entre dois vértices** e o problema dos **caminhos mínimos de fonte única**.
- Se o grafo é acíclico, podemos usar um algoritmo específico que usa a **ordenação topológica**.
- Para quaisquer outros grafos sem arcos negativos, deve ser usado o **Algoritmo de Dijkstra**.

Aplicações

- Algoritmos para calcular de forma rápida rotas entre dois locais físicos dentro de uma cidade (Google Maps).
- Em **teoria dos jogos**, para identificar soluções que usem o menor número de movimentos.
- Em robótica, engenharia de transportes, pesquisa operacional, entre muitas outras áreas.

Material bibliográfico e exercícios

T. Cormen et al. Algoritmos - Teoria e Prática (3a ed.). – **Cap. 24**

Exercícios: ver exercícios no final dos (sub)capítulos do Cap. 24.

Quem encontrar um errinho nestes slides leva +0.1 na disciplina.

Dúvidas

Dúvidas?