

Análise de Algoritmos

**Parte destes slides são adaptações de slides
do Prof. Paulo Feofiloff e do Prof. José Coelho de Pina.**

Árvore geradora mínima

CLRS Cap 23

Árvore geradora mínima

Seja G um grafo conexo.

Uma árvore T em G é **geradora** se contém todos os vértices de G .

Problema: Dado G conexo com peso w_e para cada aresta e , encontrar árvore geradora em G de peso mínimo.

O peso de uma árvore é a soma do peso de suas arestas.

Tal árvore é chamada de **árvore geradora mínima** em G .

MST: minimum spanning tree

Arestas seguras

Seja $G = (V, E)$ um grafo conexo.

Função w que atribui um peso w_e para cada aresta $e \in E$.

$A \subseteq E$ contido em alguma MST de (G, w) .

Arestas seguras

Seja $G = (V, E)$ um grafo conexo.

Função w que atribui um peso w_e para cada aresta $e \in E$.

$A \subseteq E$ contido em alguma MST de (G, w) .

Aresta $e \in E$ é **segura** para A se

$A \cup \{e\}$ está contido em alguma MST de (G, w) .

Se A não é uma MST, então existe aresta segura para A .

Arestas seguras

Seja $G = (V, E)$ um grafo conexo.

Função w que atribui um peso w_e para cada aresta $e \in E$.

$A \subseteq E$ contido em alguma MST de (G, w) .

Aresta $e \in E$ é **segura** para A se

$A \cup \{e\}$ está contido em alguma MST de (G, w) .

Se A não é uma MST, então existe aresta segura para A .

GENÉRICO (G, w)

1 $A \leftarrow \emptyset$

2 **enquanto** A não é geradora **faça**

3 encontre aresta segura e para A

4 $A \leftarrow A \cup \{e\}$

5 **devolva** A

Arestas seguras

Corte em G : partição $(S, V \setminus S)$.

Aresta e **cruza o corte** $(S, V \setminus S)$ se exatamente um de seus extremos está em S .

Arestas seguras

Corte em G : partição $(S, V \setminus S)$.

Aresta e **cruza o corte** $(S, V \setminus S)$ se
exatamente um de seus extremos está em S .

Corte **respeita** $A \subseteq E$: nenhuma aresta de A o cruza.

Arestas seguras

Corte em G : partição $(S, V \setminus S)$.

Aresta e **cruza o corte** $(S, V \setminus S)$ se exatamente um de seus extremos está em S .

Corte **respeita** $A \subseteq E$: nenhuma aresta de A o cruza.

Teorema: Se A está contida em MST de (G, w) , então e com $w(e)$ mínimo em corte que respeita A é segura para A .

Arestas seguras

Corte em G : partição $(S, V \setminus S)$.

Aresta e **cruza o corte** $(S, V \setminus S)$ se exatamente um de seus extremos está em S .

Corte **respeita** $A \subseteq E$: nenhuma aresta de A o cruza.

Teorema: Se A está contida em MST de (G, w) , então e com $w(e)$ mínimo em corte que respeita A é segura para A .

Prova: Seja T uma MST em (G, w) que contém A . Se e está em T , não há nada mais a provar.

Arestas seguras

Corte em G : partição $(S, V \setminus S)$.

Aresta e **cruza o corte** $(S, V \setminus S)$ se exatamente um de seus extremos está em S .

Corte **respeita** $A \subseteq E$: nenhuma aresta de A o cruza.

Teorema: Se A está contida em MST de (G, w) , então e com $w(e)$ mínimo em corte que respeita A é segura para A .

Prova: Seja T uma MST em (G, w) que contém A .
Se e está em T , não há nada mais a provar.

Se e não está em T , seja C o único circuito em $T + e$,
e seja $f \in C$ com $w(f)$ máximo que cruza o mesmo corte
(distinta de e em caso de empate).

Arestas seguras

Corte em G : partição $(S, V \setminus S)$.

Aresta e **cruza o corte** $(S, V \setminus S)$ se exatamente um de seus extremos está em S .

Corte **respeita** $A \subseteq E$: nenhuma aresta de A o cruza.

Teorema: Se A está contida em MST de (G, w) , então e com $w(e)$ mínimo em corte que respeita A é segura para A .

Prova: Seja T uma MST em (G, w) que contém A . Se e está em T , não há nada mais a provar.

Se e não está em T , seja C o único circuito em $T + e$, e seja $f \in C$ com $w(f)$ máximo que cruza o mesmo corte (distinta de e em caso de empate).

Então $T' := T + e - f$ é MST e contém $A \cup \{e\}$. ■

Árvore geradora mínima

Os dois próximos algoritmos se enquadram no genérico.

Árvore geradora mínima

Os dois próximos algoritmos se enquadram no genérico.

O primeiro é o **algoritmo de Kruskal** que, em cada iteração, escolhe uma aresta segura mais leve possível.

(O algoritmo de Kruskal foi apresentado na aula passada.)

Árvore geradora mínima

Os dois próximos algoritmos se enquadram no genérico.

O primeiro é o **algoritmo de Kruskal** que, em cada iteração, escolhe uma aresta segura mais leve possível.

(O algoritmo de Kruskal foi apresentado na aula passada.)

O segundo é o **algoritmo de Prim**,
que mantém uma árvore T que contém um vértice s ,
acrescentando em cada iteração uma aresta segura a T .

Árvore geradora mínima

Os dois próximos algoritmos se enquadram no genérico.

O primeiro é o **algoritmo de Kruskal** que, em cada iteração, escolhe uma aresta segura mais leve possível.

(O algoritmo de Kruskal foi apresentado na aula passada.)

O segundo é o **algoritmo de Prim**, que mantém uma árvore T que contém um vértice s , acrescentando em cada iteração uma aresta segura a T .

Os dois produzem uma MST de (G, w) .

$n := |V(G)|$ e $m := |E(G)|$.

Algoritmo de Prim

PRIM (G, w)

- 1 seja s um vértice arbitrário de G
- 2 **para** $v \in V(G) \setminus \{s\}$ **faça** $\text{key}[v] \leftarrow \infty$
- 3 $\text{key}[s] \leftarrow 0$ $\pi[s] \leftarrow \text{nil}$
- 4 $Q \leftarrow V(G)$
- 5 **enquanto** $Q \neq \emptyset$ **faça**
- 6 $u \leftarrow \text{EXTRACT-MIN}(Q)$
- 7 **para cada** $v \in \text{adj}(u)$ **faça**
- 8 **se** $v \in Q$ e $w(uv) < \text{key}[v]$
- 9 **então** $\pi[v] \leftarrow u$ $\text{key}[v] \leftarrow w(uv)$
 ▷ DecreaseKey implícito na alteração do $\text{key}[v]$
- 10 **devolva** π

Algoritmo de Prim

PRIM (G, w)

```
1  seja  $s$  um vértice arbitrário de  $G$ 
2  para  $v \in V(G) \setminus \{s\}$  faça  $\text{key}[v] \leftarrow \infty$ 
3   $\text{key}[s] \leftarrow 0$     $\pi[s] \leftarrow \text{nil}$ 
4   $Q \leftarrow V(G)$ 
5  enquanto  $Q \neq \emptyset$  faça
6       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
7      para cada  $v \in \text{adj}(u)$  faça
8          se  $v \in Q$  e  $w(uv) < \text{key}[v]$ 
9              então  $\pi[v] \leftarrow u$   $\text{key}[v] \leftarrow w(uv)$ 
                $\triangleright$  DecreaseKey implícito na alteração do  $\text{key}[v]$ 
10 devolva  $\pi$ 
```

Consumo de tempo das operações na fila de prioridade:

Linha 4: $\Theta(n)$ **EXTRACT-MIN** e **DECREASE-KEY** : $O(\lg n)$

Algoritmo de Prim

PRIM (G, w)

```
1  seja  $s$  um vértice arbitrário de  $G$ 
2  para  $v \in V(G) \setminus \{s\}$  faça  $\text{key}[v] \leftarrow \infty$ 
3   $\text{key}[s] \leftarrow 0$     $\pi[s] \leftarrow \text{nil}$ 
4   $Q \leftarrow V(G)$ 
5  enquanto  $Q \neq \emptyset$  faça
6       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
7      para cada  $v \in \text{adj}(u)$  faça
8          se  $v \in Q$  e  $w(uv) < \text{key}[v]$ 
9              então  $\pi[v] \leftarrow u$   $\text{key}[v] \leftarrow w(uv)$ 
               $\triangleright$  DecreaseKey implícito na alteração do  $\text{key}[v]$ 
10 devolva  $\pi$ 
```

Consumo de tempo do Prim: $O(m \lg n)$

Algoritmo de Prim

PRIM (G, w)

```
1  seja  $s$  um vértice arbitrário de  $G$ 
2  para  $v \in V(G) \setminus \{s\}$  faça  $\text{key}[v] \leftarrow \infty$ 
3   $\text{key}[s] \leftarrow 0$     $\pi[s] \leftarrow \text{nil}$ 
4   $Q \leftarrow V(G)$ 
5  enquanto  $Q \neq \emptyset$  faça
6       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
7      para cada  $v \in \text{adj}(u)$  faça
8          se  $v \in Q$  e  $w(uv) < \text{key}[v]$ 
9              então  $\pi[v] \leftarrow u$   $\text{key}[v] \leftarrow w(uv)$ 
                $\triangleright$  DecreaseKey implícito na alteração do  $\text{key}[v]$ 
10 devolva  $\pi$ 
```

Consumo de tempo do Prim: $O(m \lg n)$

Consumo de tempo com Fibonacci heap: $O(m + n \lg n)$

Caminhos mais curtos

CLRS Secs 24.3 e 25.2

Caminhos mais curtos

$G = (V, E)$: grafo **orientado**

Função c que atribui um comprimento c_e para cada $e \in E$.

Caminhos mais curtos

$G = (V, E)$: grafo **orientado**

Função c que atribui um comprimento c_e para cada $e \in E$.

Para vértices u e v , a **distância** de u a v é o comprimento de um caminho de u a v de comprimento mínimo.

Caminhos mais curtos

$G = (V, E)$: grafo **orientado**

Função c que atribui um comprimento c_e para cada $e \in E$.

Para vértices u e v , a **distância** de u a v é o comprimento de um caminho de u a v de comprimento mínimo.

Problema 1: Dados G , c e um vértice s de G , encontrar a distância de s a cada vértice de G .

Caminhos mais curtos

$G = (V, E)$: grafo **orientado**

Função c que atribui um comprimento c_e para cada $e \in E$.

Para vértices u e v , a **distância** de u a v é o comprimento de um caminho de u a v de comprimento mínimo.

Problema 1: Dados G , c e um vértice s de G , encontrar a distância de s a cada vértice de G .

Problema 2: Dados G e c , encontrar a distância entre todo par de vértices de G .

Caminhos mais curtos

$G = (V, E)$: grafo **orientado**

Função c que atribui um comprimento c_e para cada $e \in E$.

Para vértices u e v , a **distância** de u a v é o comprimento de um caminho de u a v de comprimento mínimo.

Problema 1: Dados G , c e um vértice s de G , encontrar a distância de s a cada vértice de G .

Problema 2: Dados G e c , encontrar a distância entre todo par de vértices de G .

Algoritmo de Dijkstra: comprimentos não negativos

Caminhos mais curtos

$G = (V, E)$: grafo **orientado**

Função c que atribui um comprimento c_e para cada $e \in E$.

Para vértices u e v , a **distância** de u a v é o comprimento de um caminho de u a v de comprimento mínimo.

Problema 1: Dados G , c e um vértice s de G , encontrar a distância de s a cada vértice de G .

Problema 2: Dados G e c , encontrar a distância entre todo par de vértices de G .

Algoritmo de Dijkstra: comprimentos não negativos

Algoritmo de Floyd-Warshall: sem circuitos negativos

Circuitos negativos

$G = (V, E)$: grafo **orientado**

Função c que atribui um comprimento c_e para cada $e \in E$.

Para vértices u e v , a **distância** de u a v é o comprimento de um caminho entre u e v de comprimento mínimo.

Circuitos negativos

$G = (V, E)$: grafo **orientado**

Função c que atribui um comprimento c_e para cada $e \in E$.

Para vértices u e v , a **distância** de u a v é o comprimento de um caminho entre u e v de comprimento mínimo.

Quando há um circuito de comprimento negativo no grafo, a distância entre certos vértices pode ficar mal-definida.

Circuitos negativos

$G = (V, E)$: grafo **orientado**

Função c que atribui um comprimento c_e para cada $e \in E$.

Para vértices u e v , a **distância** de u a v é o comprimento de um caminho entre u e v de comprimento mínimo.

Quando há um circuito de comprimento negativo no grafo, a distância entre certos vértices pode ficar mal-definida.

Poderíamos dar “voltas” num circuito negativo, cada vez obtendo um “caminho” de comprimento menor.

Circuitos negativos

$G = (V, E)$: grafo **orientado**

Função c que atribui um comprimento c_e para cada $e \in E$.

Para vértices u e v , a **distância** de u a v é o comprimento de um caminho entre u e v de comprimento mínimo.

Quando há um circuito de comprimento negativo no grafo, a distância entre certos vértices pode ficar mal-definida.

Poderíamos dar “voltas” num circuito negativo, cada vez obtendo um “caminho” de comprimento menor.

Assim definimos a distância $\delta(u, v)$ como

$-\infty$, caso exista circuito negativo alcançável de u ,
e o comprimento de um caminho mais curto de u a v c.c.

Propriedades

P : caminho mais curto de s a t

Subestrutura ótima:

Subcaminhos de P são caminhos mais curtos.

Propriedades

P : caminho mais curto de s a t

Subestrutura ótima:

Subcaminhos de P são caminhos mais curtos.

Lema: Dados G e c , seja $P = \langle v_1, \dots, v_k \rangle$ um caminho mais curto em G de v_1 a v_k . Para todo $1 \leq i \leq j \leq k$, $P_{ij} := \langle v_i, \dots, v_j \rangle$ é um caminho mais curto de v_i a v_j .

Propriedades

P : caminho mais curto de s a t

Subestrutura ótima:

Subcaminhos de P são caminhos mais curtos.

Lema: Dados G e c , seja $P = \langle v_1, \dots, v_k \rangle$ um caminho mais curto em G de v_1 a v_k . Para todo $1 \leq i \leq j \leq k$, $P_{ij} := \langle v_i, \dots, v_j \rangle$ é um caminho mais curto de v_i a v_j .

Corolário: Para G e c , se o último arco de um caminho mais curto de s a t é o arco ut , então $\delta(s, t) = \delta(s, u) + c(ut)$.

Propriedades

P : caminho mais curto de s a t

Subestrutura ótima:

Subcaminhos de P são caminhos mais curtos.

Lema: Dados G e c , seja $P = \langle v_1, \dots, v_k \rangle$ um caminho mais curto em G de v_1 a v_k . Para todo $1 \leq i \leq j \leq k$, $P_{ij} := \langle v_i, \dots, v_j \rangle$ é um caminho mais curto de v_i a v_j .

Corolário: Para G e c , se o último arco de um caminho mais curto de s a t é o arco ut , então $\delta(s, t) = \delta(s, u) + c(ut)$.

Lema: Para G , c e s ,

$\delta(s, v) \leq \delta(s, u) + c(uv)$ para todos os arcos uv .

Algoritmo de Dijkstra

π : representa os caminhos mínimos até s

d : guarda a distância de cada vértice a s .

DIJKSTRA (G, c, s)

```
1  para  $v \in V(G)$  faça  $d[v] \leftarrow \infty$   $\pi[s] \leftarrow \text{nil}$ 
2   $d[s] \leftarrow 0$ 
3   $Q \leftarrow V(G)$   $\triangleright$  chave de  $v$  é  $d[v]$ 
4  enquanto  $Q \neq \emptyset$  faça
5       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      para cada  $v \in \text{adj}(u)$  faça
7          se  $v \in Q$  e  $d[v] > d[u] + c(uv)$ 
8              então  $\pi[v] \leftarrow u$   $d[v] \leftarrow d[u] + c(uv)$ 
9  devolva  $(\pi, d)$ 
```

Algoritmo de Dijkstra

π : representa os caminhos mínimos até s

d : guarda a distância de cada vértice a s .

DIJKSTRA (G, c, s)

```
1  para  $v \in V(G)$  faça  $d[v] \leftarrow \infty$   $\pi[s] \leftarrow \text{nil}$ 
2   $d[s] \leftarrow 0$ 
3   $Q \leftarrow V(G)$   $\triangleright$  chave de  $v$  é  $d[v]$ 
4  enquanto  $Q \neq \emptyset$  faça
5       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      para cada  $v \in \text{adj}(u)$  faça
7          se  $v \in Q$  e  $d[v] > d[u] + c(uv)$ 
8              então  $\pi[v] \leftarrow u$   $d[v] \leftarrow d[u] + c(uv)$ 
9  devolva  $(\pi, d)$ 
```

$d[u]$: comprimento de um caminho mínimo de s a u
cujos vértices internos estão fora de Q

Algoritmo de Dijkstra

π : representa os caminhos mínimos até s

d : guarda a distância de cada vértice a s .

DIJKSTRA (G, c, s)

```
1  para  $v \in V(G)$  faça  $d[v] \leftarrow \infty$   $\pi[s] \leftarrow \text{nil}$ 
2   $d[s] \leftarrow 0$ 
3   $Q \leftarrow V(G)$   $\triangleright$  chave de  $v$  é  $d[v]$ 
4  enquanto  $Q \neq \emptyset$  faça
5       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      para cada  $v \in \text{adj}(u)$  faça
7          se  $v \in Q$  e  $d[v] > d[u] + c(uv)$ 
8              então  $\pi[v] \leftarrow u$   $d[v] \leftarrow d[u] + c(uv)$ 
9  devolva ( $\pi, d$ )
```

Invariantes: $d[u] = \delta(s, u)$ **se** $u \notin Q$
 $d[u] \geq \delta(s, u)$ **se** $u \in Q$

Algoritmo de Dijkstra

DIJKSTRA (G, c, s)

```
1  para  $v \in V(G)$  faça  $d[v] \leftarrow \infty$   $\pi[s] \leftarrow \text{nil}$ 
2   $d[s] \leftarrow 0$ 
3   $Q \leftarrow V(G)$   $\triangleright$  chave de  $v$  é  $d[v]$ 
4  enquanto  $Q \neq \emptyset$  faça
5       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      para cada  $v \in \text{adj}(u)$  faça
7          se  $v \in Q$  e  $d[v] > d[u] + c(uv)$ 
8              então  $\pi[v] \leftarrow u$   $d[v] \leftarrow d[u] + c(uv)$ 
9  devolva  $(\pi, d)$ 
```

Invariantes: $d[u] = \delta(s, u)$ **se** $u \notin Q$
 $d[u] \geq \delta(s, u)$ **se** $u \in Q$

Algoritmo de Dijkstra

DIJKSTRA (G, c, s)

```
1  para  $v \in V(G)$  faça  $d[v] \leftarrow \infty$   $\pi[s] \leftarrow \text{nil}$ 
2   $d[s] \leftarrow 0$ 
3   $Q \leftarrow V(G)$   $\triangleright$  chave de  $v$  é  $d[v]$ 
4  enquanto  $Q \neq \emptyset$  faça
5       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      para cada  $v \in \text{adj}(u)$  faça
7          se  $v \in Q$  e  $d[v] > d[u] + c(uv)$ 
8              então  $\pi[v] \leftarrow u$   $d[v] \leftarrow d[u] + c(uv)$ 
9  devolva  $(\pi, d)$ 
```

Invariantes: $d[u] = \delta(s, u)$ **se** $u \notin Q$
 $d[u] \geq \delta(s, u)$ **se** $u \in Q$

Onde usamos que
não há arestas de comprimento negativo?

Algoritmo de Dijkstra

DIJKSTRA (G, c, s)

```
1  para  $v \in V(G)$  faça  $d[v] \leftarrow \infty$   $\pi[s] \leftarrow \text{nil}$ 
2   $d[s] \leftarrow 0$ 
3   $Q \leftarrow V(G)$   $\triangleright$  chave de  $v$  é  $d[v]$ 
4  enquanto  $Q \neq \emptyset$  faça
5       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      para cada  $v \in \text{adj}(u)$  faça
7          se  $v \in Q$  e  $d[v] > d[u] + c(uv)$ 
8              então  $\pi[v] \leftarrow u$   $d[v] \leftarrow d[u] + c(uv)$ 
9  devolva  $(\pi, d)$ 
```

Algoritmo de Dijkstra

DIJKSTRA (G, c, s)

```
1  para  $v \in V(G)$  faça  $d[v] \leftarrow \infty$   $\pi[s] \leftarrow \text{nil}$ 
2   $d[s] \leftarrow 0$ 
3   $Q \leftarrow V(G)$   $\triangleright$  chave de  $v$  é  $d[v]$ 
4  enquanto  $Q \neq \emptyset$  faça
5       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      para cada  $v \in \text{adj}(u)$  faça
7          se  $v \in Q$  e  $d[v] > d[u] + c(uv)$ 
8              então  $\pi[v] \leftarrow u$   $d[v] \leftarrow d[u] + c(uv)$ 
9  devolva  $(\pi, d)$ 
```

Se Q for uma lista simples:

Linha 3 e **EXTRACT-MIN** : $O(n)$

Algoritmo de Dijkstra

DIJKSTRA (G, c, s)

```
1  para  $v \in V(G)$  faça  $d[v] \leftarrow \infty$   $\pi[s] \leftarrow \text{nil}$ 
2   $d[s] \leftarrow 0$ 
3   $Q \leftarrow V(G)$   $\triangleright$  chave de  $v$  é  $d[v]$ 
4  enquanto  $Q \neq \emptyset$  faça
5       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      para cada  $v \in \text{adj}(u)$  faça
7          se  $v \in Q$  e  $d[v] > d[u] + c(uv)$ 
8              então  $\pi[v] \leftarrow u$   $d[v] \leftarrow d[u] + c(uv)$ 
9  devolva  $(\pi, d)$ 
```

Se Q for uma lista simples:

Linha 3 e EXTRACT-MIN : $O(n)$

Consumo de tempo do Dijkstra: $O(n^2)$

Algoritmo de Dijkstra

DIJKSTRA (G, c, s)

```
1  para  $v \in V(G)$  faça  $d[v] \leftarrow \infty$   $\pi[s] \leftarrow \text{nil}$ 
2   $d[s] \leftarrow 0$ 
3   $Q \leftarrow V(G)$   $\triangleright$  chave de  $v$  é  $d[v]$ 
4  enquanto  $Q \neq \emptyset$  faça
5       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      para cada  $v \in \text{adj}(u)$  faça
7          se  $v \in Q$  e  $d[v] > d[u] + c(uv)$ 
8              então  $\pi[v] \leftarrow u$   $d[v] \leftarrow d[u] + c(uv)$ 
9  devolva  $(\pi, d)$ 
```

Algoritmo de Dijkstra

DIJKSTRA (G, c, s)

```
1  para  $v \in V(G)$  faça  $d[v] \leftarrow \infty$   $\pi[s] \leftarrow \text{nil}$ 
2   $d[s] \leftarrow 0$ 
3   $Q \leftarrow V(G)$   $\triangleright$  chave de  $v$  é  $d[v]$ 
4  enquanto  $Q \neq \emptyset$  faça
5       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      para cada  $v \in \text{adj}(u)$  faça
7          se  $v \in Q$  e  $d[v] > d[u] + c(uv)$ 
8              então  $\pi[v] \leftarrow u$   $d[v] \leftarrow d[u] + c(uv)$ 
9  devolva  $(\pi, d)$ 
```

Consumo de tempo das operações na fila de prioridade:

Inicialização: $O(n)$ **EXTRACT-MIN** e **DECREASE-KEY** : $O(\lg n)$

Algoritmo de Dijkstra

DIJKSTRA (G, c, s)

```
1  para  $v \in V(G)$  faça  $d[v] \leftarrow \infty$   $\pi[s] \leftarrow \text{nil}$ 
2   $d[s] \leftarrow 0$ 
3   $Q \leftarrow V(G)$   $\triangleright$  chave de  $v$  é  $d[v]$ 
4  enquanto  $Q \neq \emptyset$  faça
5       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      para cada  $v \in \text{adj}(u)$  faça
7          se  $v \in Q$  e  $d[v] > d[u] + c(uv)$ 
8              então  $\pi[v] \leftarrow u$   $d[v] \leftarrow d[u] + c(uv)$ 
9  devolva  $(\pi, d)$ 
```

Consumo de tempo do Dijkstra: $O(m \lg n)$

Algoritmo de Dijkstra

DIJKSTRA (G, c, s)

```
1  para  $v \in V(G)$  faça  $d[v] \leftarrow \infty$   $\pi[s] \leftarrow \text{nil}$ 
2   $d[s] \leftarrow 0$ 
3   $Q \leftarrow V(G)$   $\triangleright$  chave de  $v$  é  $d[v]$ 
4  enquanto  $Q \neq \emptyset$  faça
5       $u \leftarrow \text{EXTRACT-MIN}(Q)$ 
6      para cada  $v \in \text{adj}(u)$  faça
7          se  $v \in Q$  e  $d[v] > d[u] + c(uv)$ 
8              então  $\pi[v] \leftarrow u$   $d[v] \leftarrow d[u] + c(uv)$ 
9  devolva  $(\pi, d)$ 
```

Consumo de tempo do Dijkstra: $O(m \lg n)$

Consumo de tempo com Fibonacci heap: $O(m + n \lg n)$