Análise e Síntese de Algoritmos

Caminhos mais curtos entre todos os pares

CLRS Cap.25

Prof. Pedro T. Monteiro

IST - Universidade de Lisboa

2024/2025

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica
  - Algoritmos greedy
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares
  - Caminhos mais curtos [CLRS, Cap.22,24-25]
  - Árvores abrangentes
  - Fluxos máximos
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Tópicos Adicionais [CLRS, Cap.32-35]
  - Complexidade Computacional

Resumo

Caminhos mais curtos entre todos os pares

Motivação

Considere que está a gerir o serviço de reencaminhamento de mercadorias mundial de uma operadora

- Existe um conjunto de armazens em locais específicos no mundo
- Existem rotas pré-definidas e com custos associados
- O serviço recebe pedidos tais como:
  "enviar mercadoria X do local A para o local B"
- O serviço deve estar automatizado por forma a conseguir satisfazer todos os possíveis pedidos de reencaminhamento de mercadorias

Motivação
Encontrar caminhos mais curtos entre todos os pares de vértices

- Se pesos não negativos, utilizar algoritmo de Dijkstra, assumindo cada vértice $v \in V$ como fonte: $O(V \times (V + E)\ lg\ V)$ (ou $O(V^3\ lg\ V)$ se o grafo for denso)

- Se existem pesos negativos, utilizar algoritmo de Bellman-Ford, assumindo cada vértice como fonte: $O(V \times VE)$ (ou $O(V^4)$ se o grafo é denso)

Objectivo: Encontrar algoritmos mais eficientes

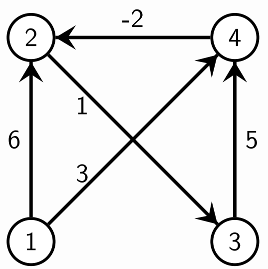
Representação
Dado um grafo $G = (V, E)$, e $n = |V|$, podemos representar através de uma matriz de adjacências:

- Pesos dos arcos: matriz W $(n \times n)$

$$w_{ij} = \begin{cases} 0 & \text{se } i = j \\ \text{peso do arco } (i, j) & \text{se } i \neq j, (i, j) \in E \\ \infty & \text{se } i \neq j, (i, j) \notin E \end{cases}$$

- Caminhos mais curtos: matriz D $(n \times n)$
  - d_{ij} é o peso do caminho mais curto entre os vértices i e j
  - $d_{ij} = \delta(v_i, v_j)$

Exemplo representação



Grafo

	1	2	3	4
1	0	6	∞	3
2	∞	0	1	∞
3	∞	∞	0	5
4	∞	-2	∞	0

Matriz W

	1	2	3	4
1	0	1	2	3
2	∞	0	1	6
3	∞	3	0	5
4	∞	-2	-1	0

Matriz D

Representação

- Representação dos predecessores: matriz Π $(n \times n)$
- $\pi_{ij} = \text{NIL}$ se $i = j$ ou não existe caminho de i para j
- Caso contrário: π_{ij} denota o predecessor de j num caminho mais curto de i para j

- Sub-grafo de predecessores $G_{\pi,i} = (V_{\pi,i}, E_{\pi,i})$:

$$V_{\pi,i} = \{j \in V : \pi_{ij} \neq \text{NIL}\} \cup \{i\}$$

$$E_{\pi_i} = \{(\pi_{ij}, j) \in E : j \in V_{\pi,i} \backslash \{i\}\}$$

- Sub-grafo de predecessores $G_{\pi,i}$ é induzido pela linha i da matriz Π

Definições

Solução recursiva

Algoritmo Floyd-Warshall [CLRS, Cap.25]

Algoritmo Johnson [CLRS, Cap.25]

Solução Recursiva

Abordagem recursiva top-down:

- Propriedade de sub-estrutura óptima dos caminhos mais curtos (aula T10)
  Sub-caminhos de caminhos mais curtos são também caminhos mais curtos
- $d_{ij}^{(m)}$: denota o peso mínimo dos caminhos do vértice i para o vértice j não contendo mais do que m arcos
- Com $m = 0$ (sem arcos), existe caminho de i para j se e só se $i = j$

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{se } i = j \\ \infty & \text{se } i \neq j \end{cases}$$

- Para $m \geq 1$ (com arcos):

$$d_{ij}^{(m)} = \min\{d_{ij}^{(m-1)}, \min_{1 \leq k \leq n} \{d_{ik}^{(m-1)} + w_{kj}\}\}$$

Solução Recursiva

Abordagem construtiva bottom-up: (para calcular $D^{(m)}$ à custa de $D^{(m-1)}$ e W)

Extend-Shortest-Paths(D,W)

```
n = rows[W]
D' = matrix(n × n)
for i = 1 to n do
  for j = 1 to n do
    d'_ij = ∞
    for k = 1 to n do
      d'_ij = min(d'_ij, d_ik + w_kj)
    end for
  end for
end for
return D'
```


Solução Recursiva

Observações

- Nota: $D^{(1)} = W$
- Calcular sequência de matrizes $D^{(1)}, \ldots, D^{(n-1)}$,
  onde $D^{(n-1)}$ contém os pesos dos caminhos mais curtos
  calcular $D^{(i)}$ em função de $D^{(i-1)}$ (e de W)

Complexidade

- $\Theta(n^3)$ para cada matriz
- $\Theta(n^4)$ para cálculo de $D^{(n)}$ (sequência de n matrizes)
  - Embora seja possível melhorar, reduzindo número de matrizes calculadas: $O(n^3 \lg n)$
  - A cada iteração, calcular $D^{(2i)}$ em função de $D^{(i)}$ e de $D^{(i)}$

Resumo

Definições

Solução recursiva

Algoritmo Floyd-Warshall [CLRS, Cap.25]

Algoritmo Johnson [CLRS, Cap.25]

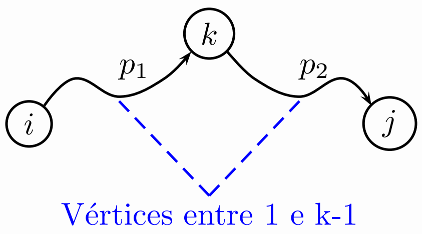
Algoritmo Floyd-Warshall

Definições

- Caracterização de caminho mais curto $p = \langle v_1, v_2, \ldots, v_{l-1}, v_l \rangle$
  - Vértices intermédios de caminho p são $\{v_2, \ldots, v_{l-1}\}$
- Considerar todos os caminhos entre i e j com vértices intermédios retirados de um conjunto $\{1, \ldots, k\} \subseteq V$ e seja p um caminho (simples) mais curto
- Se k não é vértice intermédio de p, então todos os vértices intermédios de p estão em $\{1, \ldots, k-1\}$
- Se k é vértice intermédio de p, então existem caminhos p_1 e p_2, respectivamente de i para k e de k para j com vértices intermédios em $\{1, \ldots, k\}$
  - k não é vértice intermédio de p_1 e de p_2
  - p_1 e p_2 com vértices intermédios em $\{1, \ldots, k-1\}$

Algoritmo Floyd-Warshall

Formulação



$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{se } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{se } k \geq 1 \end{cases}$$

Algoritmo Floyd-Warshall

Floyd-Warshall(W)

```
n = rows[W]
D(0) = W
for k = 1 to n do
  D(k) = new matrix(n×n)
  for i = 1 to n do
    for j = 1 to n do
      d(k)ij = min(d(k-1)ij, d(k-1)ik + d(k-1)kj)
    end for
  end for
end for
return D(n)
```

Complexidade

- Tempo: $\Theta(n^3)$
- Espaço: $\Theta(n^3)$

Algoritmo Floyd-Warshall

Optimizações

- evitar uma matriz nova por cada passo do algoritmo
- linha e a coluna k não são alteradas na iteração k:

$$d_{ik}^{(k)} = \min(d_{ik}^{(k-1)}, d_{ik}^{(k-1)} + d_{kk}^{(k-1)})$$

$$d_{kj}^{(k)} = \min(d_{kj}^{(k-1)}, d_{kk}^{(k-1)} + d_{kj}^{(k-1)})$$

Nota: $d_{kk}^{(k-1)} = 0$

Algoritmo Floyd-Warshall

Floyd-Warshall(D,W)

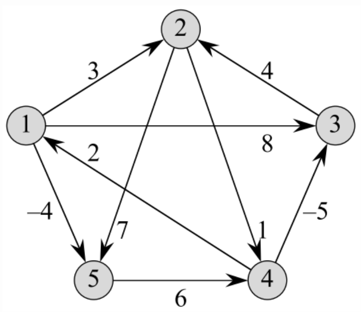
```
n = rows[W]
D = W
for k = 1 to n do
  for i = 1 to n do
    for j = 1 to n do
      d_ij = min(d_ij, d_ik + d_kj)
    end for
  end for
end for
return D
```

Complexidade

- Tempo: $\Theta(n^3)$
- Espaço: $\Theta(n^2)$

Algoritmo Floyd-Warshall

Exemplo [CLRS, Fig 25.1]



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

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

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

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

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

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

Algoritmo Floyd-Warshall

Fecho Transitivo de um Grafo Dirigido

Dado um grafo $G = (V, E)$ dirigido, o fecho transitivo é definido por $G^* = (V, E^*)$ tal que:

$$E^* = \{(i, j) : \text{ existe caminho de } i \text{ para } j \text{ em } G\}$$

Algoritmo

- Atribuir a cada arco peso 1 e utilizar algoritmo de Floyd-Warshall
- Se $d_{ij} \neq \infty$, então $(i, j) \in E^*$
- Complexidade: $\Theta(n^3)$

Resumo

Definições

Solução recursiva

Algoritmo Floyd-Warshall [CLRS, Cap.25]

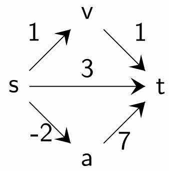
Algoritmo Johnson [CLRS, Cap.25]

Algoritmo Johnson

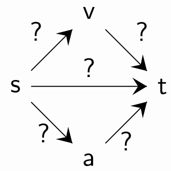
Intuição

- Se arcos com pesos não negativos, utilizar Dijkstra para cada vértice
- Caso contrário, reduzir a um problema com pesos não negativos

Exemplo



Que transformação
podemos fazer
para que os pesos
sejam não-negativos?
$\Rightarrow$



(somar 2 em todos os arcos, altera a natureza dos caminhos mais curtos)

Algoritmo Johnson

Intuição

Se arcos com pesos negativos, utilizar Bellman-Ford para

- efectuar repesagem dos arcos, *i.e.*, calcular novo conjunto de pesos não negativos w', tal que:
  - Um caminho mais curto de u para v com função w é também caminho mais curto com função w'
  - Para cada arco (u, v) o peso $w'(u, v)$ é não negativo

Algoritmo Johnson

Repesagem dos arcos

- Dado $G = (V, E)$, com função de pesos w e de repesagem $h : V \to \mathbb{R}$, seja $w'(u, v) = w(u, v) + h(u) - h(v)$
- Seja $p = \langle v_0, v_1, \ldots, v_k \rangle$. Então $w(p) = \delta(v_0, v_k)$ se e só se $w'(p) = \delta'(v_0, v_k) = \delta(v_0, v_k) + h(v_0) - h(v_k)$
- Existe ciclo negativo com w se e só se existe ciclo negativo com w'
- Verificar que $w'(p) = w(p) + h(v_0) - h(v_k)$

$$
\begin{aligned}
w'(p) &= \sum_{i=1}^{k} w'(v_{i-1}, v_i) \\
&= \sum_{i=1}^{k} (w(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)) \\
&= \sum_{i=1}^{k} w(v_{i-1}, v_i) + h(v_0) - h(v_k) \\
&= w(p) + h(v_0) - h(v_k)
\end{aligned}
$$

Propriedades da repesagem dos arcos

Caminhos mais curtos mantém-se após a repesagem!
Se p é caminho mais curto com função de peso w, então também é caminho mais curto com função de peso w'

- Verificar que $w(p) = \delta(v_0, v_k) \rightarrow w'(p) = \delta'(v_0, v_k)$
- Hipótese: existe outro caminho mais curto p_z de v_0 para v_k após a repesagem tal que $w'(p_z) < w'(p)$

$$w(p_z) + h(v_0) - h(v_k) = w'(p_z) < w'(p) = w(p) + h(v_0) - h(v_k)$$

- Para que fosse verdade, teríamos que $w(p_z) < w(p)$, o que contradiz o facto de p ser caminho mais curto com função de peso w

Observação

Para quaisquer caminhos p_1, p_2 entre v_0 e v_k, verifica-se que
$w(p_1) < w(p_2) \leftrightarrow w'(p_1) < w'(p_2)$

Prova

$w'(p) = \delta'(v_0, v_k) \rightarrow w(p) = \delta(v_0, v_k)$
Muito semelhante à anterior em que se admite existir um caminho mais curto p_z com função de peso w

Ciclos Negativos

Existe ciclo negativo com w se e só se existe com w'

- Considere-se que o caminho $p_c = \langle v_0, \ldots, v_k \rangle$ define um ciclo negativo.
  Então, $v_0 = v_k$ e $w(p_c) < 0$
- $w'(p_c) = w(p_c) + h(v_0) - h(v_k) = w(p_c)$, dado que $v_0 = v_k$

Resumo

Caminhos mais curtos e ciclos negativos não se alteram com a mudança da função de peso
$\mathbf{w'(u,v) = w(u,v) + h(u) - h(v)}$

Organização do algoritmo

- Dado $G = (V, E)$, criar $G' = (V', E')$ definido do seguinte modo:
  - $V' = V \cup \{s\}$
  - $E' = E \cup \{(s, v) : v \in V\}$
  - $\forall v \in V : w(s, v) = 0$
- Ciclos negativos são detectados pelo algoritmo de Bellman-Ford aplicado a G'
- Se não existirem ciclos negativos:
  - Definir $h(v) = \delta(s, v)$
  - Pela propriedade dos caminhos mais curtos, para cada arco (u, v), temos que $h(v) \leq h(u) + w(u, v)$
  - Logo, $w'(u, v) = w(u, v) + h(u) - h(v) \geq 0$
- Executar Dijkstra para todo o $u \in V$ com função de peso w'
  - Cálculo $\delta'(u, v)$ para todo o $u \in V$
  - Cálcular de volta $\delta(u, v) = \delta'(u, v) - h(u) + h(v)$

Algoritmo Johnson

Johnson(G)

```
representar G'
if not Bellman-Ford(G', w, s) then
  return "Indicar ciclo negativo"
end if
h(v) ← δ(s, v), calculado com Bellman-Ford
for each (u, v) ∈ G.E do
  w'(u, v) = w(u, v) + h(u) - h(v)
end for
for each u ∈ G.V do
  executar Dijkstra(G, w', u)
  calcular δ'(u, v)
  for each v ∈ G.V do
    d_uv = δ(u,v) = δ'(u, v) + h(v) - h(u)
  end for
end for
return D
```

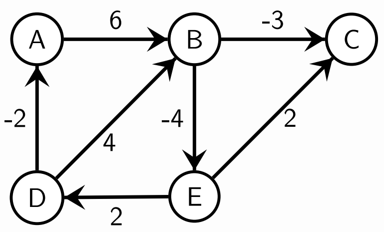

Algoritmo Johnson

Complexidade

- Bellman-Ford (1 vez): $O(VE)$
- Executar Dijkstra para cada vértice $v \in V$: $O(V(V+E)\ lg\ V)$
- Total: $O(V(V+E)\ lg\ V)$
- Útil para grafos esparsos

Algoritmo Johnson

Exercício (fazer casa/quadro)
Calcule os valores de $h(u)$ para todos os vértices $u \in V$ do grafo
Calcule também os pesos w' de todos os arcos após a repesagem



Questões?

Dúvidas?