

Definição de matroides do CLRS

U : conjunto finito arbitrário.

$\mathcal{C}$: família não vazia de subconjuntos de U hereditária.
(subconjuntos de conjuntos em $\mathcal{C}$ estão em $\mathcal{C}$)

$\mathcal{C}$ é um **matroide** se, para todo par A, B de conjuntos de $\mathcal{C}$ para os quais $|A| < |B|$, existe $e \in B \setminus A$ tal que $A \cup \{e\} \in \mathcal{C}$.

Definição da aula passada

U : conjunto finito arbitrário.

$\mathcal{C}$: família não vazia de subconjuntos de U hereditária.

Problema 1: Dado um peso binário w_e para cada e de U , encontrar um conjunto de $\mathcal{C}$ de peso máximo.

GULOSO ($U, w, \mathcal{C}$)

- 1 $U_1 \leftarrow \{e \in U : w_e = 1\} \quad S \leftarrow \emptyset$
- 2 **enquanto** existe e em U_1 tal que $S \cup \{e\} \in \mathcal{C}$ **faça**
- 3 $S \leftarrow S \cup \{e\}$
- 4 **devolva** S

GULOSO encontra um conjunto de $\mathcal{C}$ de peso **maximal**.

$\mathcal{C}$ é um **matroide** se
todo conjunto de $\mathcal{C}$ de peso **maximal** tem o mesmo tamanho.

Matroides e o método guloso

U : conjunto finito.

$\mathcal{C}$: família não vazia de subconjuntos de U hereditária.

Pesos positivos para os elementos de U .

Problema 2: Encontrar um conjunto de $\mathcal{C}$ de peso máximo.

GULOSO ($U, w, \mathcal{C}$)

- 1 $(e_1, \dots, e_n) \leftarrow \text{ORDENE}(U, w)$ ▷ ordem dos pesos
- 2 $S \leftarrow \emptyset$
- 3 **para** $i \leftarrow 1$ **até** n **faça**
- 4 **se** $S \cup \{e_i\} \in \mathcal{C}$
- 5 **então** $S \leftarrow S \cup \{e_i\}$
- 6 **devolva** S

Teorema: Se $\mathcal{C}$ é um matroide,
então o algoritmo acima resolve o Problema 2.

Exemplos de matroides

Dado um espaço vetorial, a coleção de todos os conjuntos de vetores LI deste espaço é um matroide.

Exemplos de matroides

Dado um espaço vetorial, a coleção de todos os conjuntos de vetores LI deste espaço é um matroide.

Dado um grafo G , a coleção de arestas de todas as florestas de G é um matroide.

Exemplos de matroides

Dado um espaço vetorial, a coleção de todos os conjuntos de vetores LI deste espaço é um matroide.

Dado um grafo G , a coleção de arestas de todas as florestas de G é um matroide.

Grafo bipartido $G = (U \times V, E)$.

M_U : todos os conjuntos S de arestas de G tq no máximo uma aresta de S é incidente a cada vértice de U .

M_U é um matroide.

Exemplos de matroides

Dado um espaço vetorial, a coleção de todos os **conjuntos de vetores LI** deste espaço é um matroide.

Dado um grafo G , a coleção de arestas de todas as **florestas** de G é um matroide.

Grafo bipartido $G = (U \times V, E)$.

M_U : todos os conjuntos S de arestas de G tq no máximo uma aresta de S é incidente a cada vértice de U .

M_U é um matroide.

M_V : a coleção análoga com V no lugar de U .

Claro que M_V também é um matroide.

Exemplos de matroides

Dado um espaço vetorial, a coleção de todos os **conjuntos de vetores LI** deste espaço é um matroide.

Dado um grafo G , a coleção de arestas de todas as **florestas** de G é um matroide.

Grafo bipartido $G = (U \times V, E)$.

M_U : todos os conjuntos S de arestas de G tq no máximo uma aresta de S é incidente a cada vértice de U .

M_U é um matroide.

M_V : a coleção análoga com V no lugar de U .

Claro que M_V também é um matroide.

E $M_U \cap M_V$? É ou não é um matroide?

Exemplos de matroides

Grafo bipartido $G = (U \times V, E)$.

M_U : todos os conjuntos S de arestas de G tq no máximo uma aresta de S é incidente a cada vértice de U .

M_V : a coleção analoga com V no lugar de U .

M_U e M_V são matroides.

Exemplos de matroides

Grafo bipartido $G = (U \times V, E)$.

M_U : todos os conjuntos S de arestas de G tq no máximo uma aresta de S é incidente a cada vértice de U .

M_V : a coleção analoga com V no lugar de U .

M_U e M_V são matroides.

O que é $M_U \cap M_V$?

O que é um conjunto da coleção $M_U \cap M_V$ em G ?

Exemplos de matroides

Grafo bipartido $G = (U \times V, E)$.

M_U : todos os conjuntos S de arestas de G tq no máximo uma aresta de S é incidente a cada vértice de U .

M_V : a coleção analoga com V no lugar de U .

M_U e M_V são matroides.

O que é $M_U \cap M_V$?

O que é um conjunto da coleção $M_U \cap M_V$ em G ?

Cada conjunto de $M_U \cap M_V$ é um **emparelhamento** em G .

Exemplos de matroides

Grafo bipartido $G = (U \times V, E)$.

M_U : todos os conjuntos S de arestas de G tq no máximo uma aresta de S é incidente a cada vértice de U .

M_V : a coleção analoga com V no lugar de U .

M_U e M_V são matroides.

O que é $M_U \cap M_V$?

O que é um conjunto da coleção $M_U \cap M_V$ em G ?

Cada conjunto de $M_U \cap M_V$ é um **emparelhamento** em G .

Esta coleção é ou não é um matroide?

Exemplos de matroides

Grafo bipartido $G = (U \times V, E)$.

M_U : todos os conjuntos S de arestas de G tq no máximo uma aresta de S é incidente a cada vértice de U .

M_V : a coleção analoga com V no lugar de U .

M_U e M_V são matroides.

O que é $M_U \cap M_V$?

O que é um conjunto da coleção $M_U \cap M_V$ em G ?

Cada conjunto de $M_U \cap M_V$ é um **emparelhamento** em G .

Esta coleção é ou não é um matroide?

Não é... (nem todo emparelhamento maximal é máximo)

Mas é a **interseção** de dois matroides.

Exemplos de matroides

Grafo bipartido $G = (U \times V, E)$.

M_U : todos os conjuntos S de arestas de G tq no máximo uma aresta de S é incidente a cada vértice de U .

M_V : a coleção analoga com V no lugar de U .

M_U e M_V são matroides.

$M_U \cap M_V$ é a coleção dos **emparelhamentos** de G .

Esta coleção não é um matroide.

Mas é a **interseção** de dois matroides.

Exemplos de matroides

Grafo bipartido $G = (U \times V, E)$.

M_U : todos os conjuntos S de arestas de G tq no máximo uma aresta de S é incidente a cada vértice de U .

M_V : a coleção analoga com V no lugar de U .

M_U e M_V são matroides.

$M_U \cap M_V$ é a coleção dos **emparelhamentos** de G .

Esta coleção não é um matroide.

Mas é a **interseção** de dois matroides.

Existe algoritmo polinomial para encontrar um conjunto (de peso) máximo na interseção de dois matroides.

Caminhos mais curtos

CLRS Secs 24.3

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 (para custos positivos):

Subcaminhos de P são caminhos mais curtos.

Propriedades

P : caminho mais curto de s a t

Subestrutura ótima (para custos positivos):

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 (para custos positivos):

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 (para custos positivos):

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 ,

$$\delta(s, v) \leq \delta(s, u) + c(uv) \text{ para todo arco } uv.$$

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  $d[v] \leftarrow \infty$ 
2   $d[s] \leftarrow 0$     $\pi[s] \leftarrow \text{nil}$ 
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  $d[u] + c(uv) < d[v]$ 
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  $d[v] \leftarrow \infty$ 
2   $d[s] \leftarrow 0$     $\pi[s] \leftarrow \text{nil}$ 
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  $d[u] + c(uv) < d[v]$ 
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  $d[v] \leftarrow \infty$ 
2   $d[s] \leftarrow 0$     $\pi[s] \leftarrow \text{nil}$ 
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  $d[u] + c(uv) < d[v]$ 
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 $d[v] \leftarrow \infty$
- 2 $d[s] \leftarrow 0$ $\pi[s] \leftarrow \text{nil}$
- 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 (π, 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 $d[v] \leftarrow \infty$
- 2 $d[s] \leftarrow 0$ $\pi[s] \leftarrow \text{nil}$
- 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 (π, 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** $d[v] \leftarrow \infty$
- 2 $d[s] \leftarrow 0$ $\pi[s] \leftarrow \text{nil}$
- 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** (π, 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** $d[v] \leftarrow \infty$
- 2 $d[s] \leftarrow 0$ $\pi[s] \leftarrow \text{nil}$
- 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** (π, d)

Consumo de tempo com 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** $d[v] \leftarrow \infty$
- 2 $d[s] \leftarrow 0$ $\pi[s] \leftarrow \text{nil}$
- 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** (π, d)

Consumo de tempo com fila de prioridade:

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

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

Algoritmo de Dijkstra

DIJKSTRA (G, c, s)

- 1 **para** $v \in V(G)$ **faça** $d[v] \leftarrow \infty$
- 2 $d[s] \leftarrow 0$ $\pi[s] \leftarrow \text{nil}$
- 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** (π, d)

Consumo de tempo

com lista simples: $O(n^2)$

com fila de prioridade: $O(m \lg n)$

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

Aula que vem

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

Algoritmo de Floyd-Warshall: sem circuitos negativos