Brücken (Forts.)

Es sei $G = (V, E)$ ein Graph und $l \in \mathbb{N}$.

Satz

Ist $u \in V$ zu l Brücken inzident, so besitzt G mindestens l von u verschiedene Knoten von ungeradem Grad.

Folgerung

Haben in einem Graphen alle Knoten geraden Grad, so besitzt er keine Brücken.

Distanz

Es sei $G = (V, E)$ ein Graph.

Definition

Es seien $v, w \in V$.

- Ist $v \sim w$, dann sei

  $d(v, w) := \min\{l \in \mathbb{N}_0 \mid \text{in } G \text{ ex. } v\text{-}w\text{-Pfad der Länge } l\} \in \mathbb{N}_0$.

- Ist $v, w \in V$ mit $v \not\sim w$, dann sei $d(v, w) := \infty$.

- Wir nennen $d(v, w)$ die *Distanz* zwischen v und w.

Distanz

Es sei $G = (V, E)$ ein Graph.

Bemerkung

Für alle $v, w \in V$ gelten:

- $d(v, w) = 0 \Leftrightarrow v = w$,
- $d(v, w) < \infty \Leftrightarrow v \sim w$.

G ist genau dann zusammenhängend, wenn gilt:
$d(v, w) < \infty$ für alle $v, w \in V$.

Breitensuche

Es sei $G = (V, E)$ ein Graph und $w \in V$.

Die *Breitensuche* ist ein Algorithmus, der, beginnend mit $w \in V$, alle Knoten der Zusammenhangskomponente von w mit aufsteigender Distanz zu w durchläuft.

Anwendungen

- Berechnung der Zusammenhangskomponenten von G.
- Berechnung der Distanzen $d(v, w)$ für v in der Zusammenhangskomponente von w.
- Berechnung kürzester Pfade von jedem v zu w.

Breitensuche (Forts.)

```
BREITENSUCHE(Γ, w)
 1  initialisiere array d[1, ..., n] mit allen Einträgen gleich ∞
 2  initialisiere array p[1, ..., n] mit allen Einträgen gleich NIL
 3  initialisiere leere queue Q (FIFO)
 4  d[w] ← 0
 5  INSERT(Q, w)
 6  while Q ist nicht leer
 7  do v ← EXTRACT(Q)
 8      for u ∈ Γ(v)
 9      do if d[u] = ∞
10            then INSERT(Q, u)
11                 d[u] ← d[v] + 1
12                 p[u] ← v
13  return d, p
```

Breitensuche (Forts.)

Kommentare (zum Algorithmus)

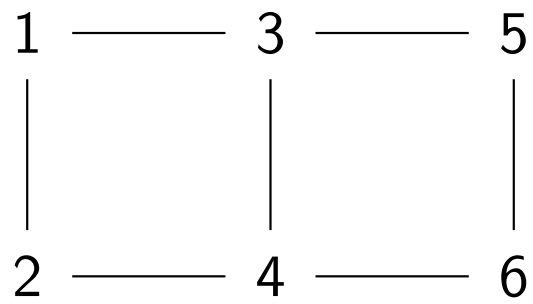
- Eingabe:
  - Γ: Adjazenzliste des Graphen $G = (V, E)$ mit $V = \underline{n}$
  - w: Knoten $w \in V$
- Der `array` $d[1, \ldots, n]$ enthält nach der Terminierung an Position v den Wert $d(w, v)$.
- Der `array` $p[1, \ldots, n]$ enthält nach der Terminierung an Position v einen Knoten u, der auf einem w-v-Pfad der Länge $d(w, v)$ unmittelbar vor v kommt.
- queue ist eine Warteschlange im „First-in-first-out“-Modus.
- Der Aufruf $\textsc{Insert}(Q, x)$ hängt das Element x an das Ende der Warteschlange.
- Der Aufruf $\textsc{Extract}(Q)$ entnimmt das Element, das am Anfang der Warteschlange steht.

Breitensuche (Forts.)

Bemerkung

Der Verlauf der Breitensuche und das Ergebnis p hängen von der Anordnung der Mengen $\Gamma(v)$ in der Adjazenzliste von G ab.

Beispiel



Breitensuche (Forts.)

Beispiel

Die Listen $\Gamma(v)$ sind aufsteigend angeordnet.

d	p	Q	v	$\Gamma(v)$	$d[u] = \infty$
$[0, \infty, \infty, \infty, \infty, \infty]$	$[-, -, -, -, -, -]$	$[1]$	1	$[2, 3]$	$[2, 3]$
$[0, 1, 1, \infty, \infty, \infty]$	$[-, 1, 1, -, -, -]$	$[2, 3]$	2	$[1, 4]$	$[4]$
$[0, 1, 1, 2, \infty, \infty]$	$[-, 1, 1, 2, -, -]$	$[3, 4]$	3	$[1, 4, 5]$	$[5]$
$[0, 1, 1, 2, 2, \infty]$	$[-, 1, 1, 2, 3, -]$	$[4, 5]$	4	$[2, 3, 6]$	$[6]$
$[0, 1, 1, 2, 2, 3]$	$[-, 1, 1, 2, 3, 4]$	$[5, 6]$	5	$[3, 6]$	$[]$
$[0, 1, 1, 2, 2, 3]$	$[-, 1, 1, 2, 3, 4]$	$[6]$	6	$[4, 5]$	$[]$
$[0, 1, 1, 2, 2, 3]$	$[-, 1, 1, 2, 3, 4]$	$[]$			

Breitensuche (Forts.)

Beispiel

Die Listen $\Gamma(v)$ sind absteigend angeordnet.

d	p	Q	v	$\Gamma(v)$	$d[u] = \infty$
$[0, \infty, \infty, \infty, \infty, \infty]$	$[-, -, -, -, -, -]$	$[1]$	1	$[3, 2]$	$[3, 2]$
$[0, 1, 1, \infty, \infty, \infty]$	$[-, 1, 1, -, -, -]$	$[3, 2]$	3	$[5, 4, 1]$	$[5, 4]$
$[0, 1, 1, 2, 2, \infty]$	$[-, 1, 1, 3, 3, -]$	$[2, 5, 4]$	2	$[4, 1]$	$[]$
$[0, 1, 1, 2, 2, \infty]$	$[-, 1, 1, 3, 3, -]$	$[5, 4]$	5	$[6, 3]$	$[6]$
$[0, 1, 1, 2, 2, 3]$	$[-, 1, 1, 3, 3, 5]$	$[4, 6]$	4	$[6, 3, 2]$	$[]$
$[0, 1, 1, 2, 2, 3]$	$[-, 1, 1, 3, 3, 5]$	$[6]$	6	$[5, 4]$	$[]$
$[0, 1, 1, 2, 2, 3]$	$[-, 1, 1, 3, 3, 5]$	$[]$			

Tiefensuche

Bemerkung

- Die *Tiefensuche* ist ein Algorithmus mit der gleichen Ein- und Ausgabe wie die Breitensuche.
- In jedem Schritt der Breitensuche wird die Distanz zu w möglichst beibehalten.
- In jedem Schritt der Tiefensuche wird die Distanz zu w möglichst vergrößert.
- Sie wird realisiert, indem die queue (FIFO) durch einen stack (LIFO= „Last-in-first-out“) ersetzt wird.

Dijkstras Algorithmus

Definition
Ein *gewichteter Graph* ist ein Tripel $G = (V, E, f)$ mit:
(V, E) ist ein Graph und f eine *Gewichtsfunktion* $f : E \to \mathbb{R}_{\geq 0}$.

Definition
Es sei $G = (V, E, f)$ ein gewichteter Graph.

- Es sei $z = (v_0, \ldots, v_l)$ ein Kantenzug in G.
  $f(z) := \sum_{i=1}^{l} f(v_{i-1}v_i)$ heißt das *Gewicht* von z.
- Für alle $v, w \in V$ mit $v \sim w$ definieren wir die *Distanz* zwischen v und w als

$$d(v, w) := \min\{f(z) \mid z \text{ ist } v\text{-}w\text{-Pfad in } G\} \in \mathbb{R}_{\geq 0}.$$

- Für alle $v, w \in V$ mit $v \not\sim w$ wird $d(v, w) := \infty$ gesetzt.

Dijkstras Algorithmus (Forts.)

```
DIJKSTRA(Γ, w, f)
 1  initialisiere array d[1, ..., n] mit allen Einträgen gleich ∞
 2  initialisiere array p[1, ..., n] mit allen Einträgen gleich NIL
 3  initialisiere priority queue Q mit Elementen 1, ..., n und
 4  allen Prioritäten = ∞
 5  d[w] ← 0
 6  INSERT(Q, w, d[w])
 7  while Q nicht leer
 8  do v ← EXTRACTMIN(Q)
 9      for u ∈ Γ[v]
10      do if d[v] + f(uv) < d[u]
11             then d[u] ← d[v] + f(uv)
12                  p[u] ← v
13                  INSERT(Q, u, d[u])
14  return d, p
```

Dijkstras Algorithmus (Forts.)

Kommentare (zum Algorithmus)

- Eingabe:
  - Γ: Adjazenzliste des Graphen $G = (V, E)$ mit $V = \underline{n}$
  - w: Knoten $w \in V$
  - f: Liste der Werte $f(v), v \in V$
- Der `array` $d[1, \ldots, n]$ enthält nach der Terminierung an Position v den Wert $d(w, v)$.
- Der `array` $p[1, \ldots, n]$ enthält nach der Terminierung an Position v einen Knoten u, der auf einem w-v-Pfad der Distanz $d(w, v)$ unmittelbar vor v kommt.

Dijkstras Algorithmus (Forts.)

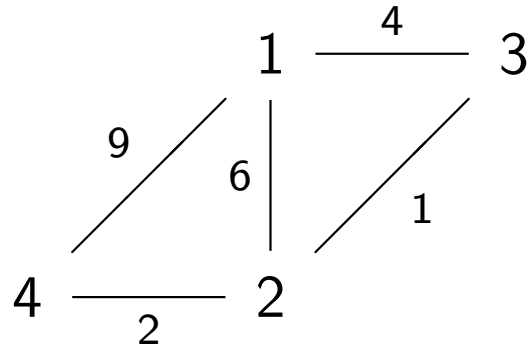
Kommentare (zum Algorithmus), Forts.

- `priority queue` ist eine *Vorrangwarteschlange*, bei der jedem ihrer Element ein *Prioritätswert* zugeordnet ist.
- Der Aufruf $\textsc{Insert}(Q, x, k)$ fügt das Element x in die Warteschlange ein und ordnet x die Priorität $k \geq 0$ zu.

  Falls x bereits in der Warteschlange enthalten ist, wird nur die Priorität neu auf k gesetzt.
- Der Aufruf $\textsc{ExtractMin}(Q)$ entnimmt das Element mit der niedrigsten Priorität.

Dijkstras Algorithmus (Forts.)

Beispiel



d	p	Q	v	$\Gamma(v)$	$d[v] + f(uv) < d[u]$
$[0, \infty, \infty, \infty]$	$[-, -, -, -]$	$\{1, 2, 3, 4\}$	1	$[2, 3, 4]$	$[2, 3, 4]$
$[0, 6, 4, 9]$	$[-, 1, 1, 1]$	$\{2, 3, 4\}$	3	$[1, 2]$	$[2]$
$[0, 5, 4, 9]$	$[-, 3, 1, 1]$	$\{2, 4\}$	2	$[1, 3, 4]$	$[4]$
$[0, 5, 4, 8]$	$[-, 3, 1, 2]$	$\{4\}$	4	$[1, 2]$	$[]$
$[0, 5, 4, 8]$	$[-, 3, 1, 2]$	$\{\}$			

Hamiltonkreise und Eulertouren

Definition

Es sei $G = (V, E)$ ein Graph.

- Ein Kreis der Länge n_G in G heißt *Hamiltonkreis*.
- Eine Tour der Länge m_G in G heißt *Eulertour*.

Hamiltonkreise und Eulertouren (Forts.)

Es sei $G = (V, E)$ ein Graph.

Bemerkung

- Ein geschlossener Kantenzug $(v_0, \ldots, v_l)$ ist genau dann ein Hamiltonkreis, wenn in der Auflistung $v_0, \ldots, v_{l-1}$ jeder Knoten aus V genau einmal vorkommt.
- Ein geschlossener Kantenzug $(v_0, \ldots, v_l)$ ist genau dann eine Eulertour, wenn in der Auflistung $v_0v_1, v_1v_2, \ldots, v_{l-1}v_l$ jede Kante aus E genau einmal vorkommt.

Definition

Ein (nicht notwendig geschlossener) Kantenzug $(v_0, \ldots, v_l)$ heißt *Eulerzug*, wenn in der Auflistung $v_0v_1, v_1v_2, \ldots, v_{l-1}v_l$ jede Kante aus E genau einmal vorkommt.

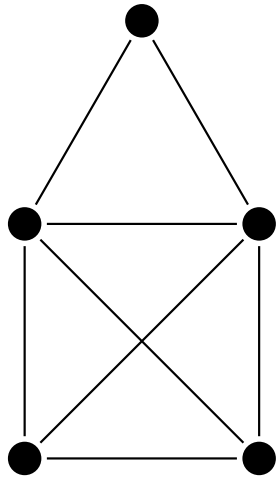
Hamiltonkreise und Eulertouren (Forts.)

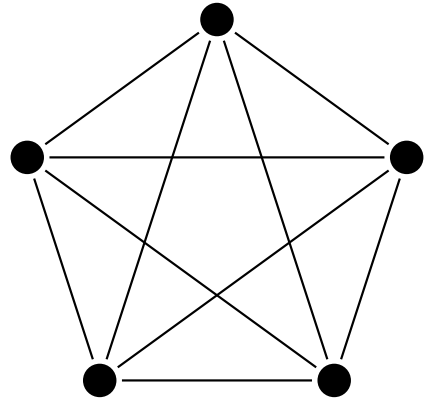
Beispiel

- Das Straßennetz einer Stadt sei durch einen Graphen modelliert (Knoten: Kreuzungen, Kanten: Straßenabschnitte).

  Der Fahrer eines Schneeräumfahrzeuges sucht eine Eulertour.

- Der Graph „Haus vom Nikolaus“ besitzt einen Eulerzug, aber keine Eulertour.
-





Eulertouren

Es sei $G = (V, E)$ ein Graph.

Bemerkung

- Existiert in G eine Eulertour, so gelten:
  - Alle Knoten von G haben geraden Grad, und
  - höchstens eine Zusammenhangskomponente von G ist nicht-trivial.

- Existiert in G ein Hamiltonkreis, so gelten:
  - Jeder Knoten von G hat Grad ≥ 2, und
  - G ist zusammenhängend und $n_G \geq 3$.

Eulertouren (Forts.)

Es sei $G = (V, E)$ ein zusammenhängender Graph.

Satz

Hat G genau zwei Knoten u, v mit ungeradem Grad, dann existiert ein u-v-Eulerzug in G.

Folgerung

Der Graph G besitzt genau dann eine Eulertour, wenn alle Knoten von G geraden Grad haben.

Eulertouren (Forts.)

Es sei $G = (V, E)$ ein zusammenhängender Graph, dessen Knoten geraden Grad haben. Die folgende Prozedur `FLEURY` berechnet eine Eulertour.

```
FLEURY(V, E)
 1  initialisiere leere Liste T
 2  v ← beliebiger Knoten aus V
 3  APPEND(T, v)
 4  while E ist nicht leer
 5  do if deg v = 1
 6        then w ← einziger Nachbar von v
 7        else w ← ein Nachbar von v mit vw keine Brücke
 8     APPEND(T, w)
 9     E ← E \ {vw}
10     v ← w
11  return T
```

Hamiltonkreise

Satz

Es sei $G = (V, E)$ ein zusammenhängender Graph mit $n_G \geq 3$.

Falls für alle $u, v \in V$ mit $u \neq v$ und $uv \notin E$ gilt

$$\deg u + \deg v \geq n_G,$$

so besitzt G einen Hamiltonkreis.

Bemerkung

Erfüllt G die Voraussetzungen des Satzes und ist n_G gerade, so gilt

$$m_G \geq \frac{n_G^2}{4}.$$