WS 2005/06

Diskrete Strukturen

Ernst W. Mayr

Fakultät für Informatik
TU München

http://www14.in.tum.de/lehre/2005WS/ds/

3. Februar 2006

5. Spezielle Pfade

5.1 Eulersche Pfade und Kreise

Definition 306

Ein Pfad bzw. Kreis in einem Graphen (Digraphen) heißt eulersch, wenn er jede Kante des Graphen genau einmal enthält.
Ein Graph (Digraph) heißt eulersch, wenn er einen eulerschen Kreis enthält.

Satz 307

Ein Graph besitzt genau dann einen eulerschen Kreis (Pfad), wenn er zusammenhängend ist und alle (alle bis auf zwei) Knoten geraden Grad haben.

Beweis:

„$\Rightarrow$“

Ein eulerscher Graph muss notwendigerweise zusammenhängend sein. Die Knotengrade müssen gerade sein, da für jede zu einem Knoten (auf dem eulerschen Kreis) hinführende Kante auch eine von diesem Knoten weiterführende Kante existieren muss, da sonst der eulersche Kreis nicht fortgeführt werden kann.

„⇐“

Konstruktion des eulerschen Kreises: Man suche einen beliebigen Kreis im Graphen (muss aufgrund der Voraussetzungen existieren). Sind noch Kanten unberücksichtigt, suche man auf dem Kreis einen Knoten, der zu noch nicht verwendeten Kanten inzident ist.

Nach Voraussetzung muss sich wieder ein Kreis finden lassen, der vollständig aus noch nicht berücksichtigten Kanten besteht. Diesen füge man zum bereits gefundenen Kreis hinzu, worauf sich ein neuer Kreis ergibt.

Dieses Verfahren läßt sich fortführen, bis keine Kanten mehr unberücksichtigt sind und damit ein eulerscher Kreis gefunden ist.

□

Satz 308

Ein Digraph besitzt genau dann einen eulerschen Kreis (Pfad), wenn er stark zusammenhängend ist und für alle Knoten der In-Grad gleich dem Aus-Grad ist (wenn für einen Knoten In-Grad $=$ Aus-Grad -1, für einen weiteren Knoten In-Grad $=$ Aus-Grad $+1$ gilt und für alle anderen Knoten der In-Grad gleich dem Aus-Grad ist).

Beweis:

Der Beweis ist analog zum Beweis des vorhergehenden Satzes. □

Algorithmus zum Finden eines eulerschen Kreises:

```
algorithm Eulerian_Circle(V,E)
  EC := ∅
  select v=v_0 ∈ V
  do
    C := ∅
    while N(v)≠∅ do
      select w ∈ N(v)
      E := E\{v,w}
      C := C ∪ {v,w}
      if N(v)≠∅ then Q.add(v) fi
      v := w
    od
    co Neuer Kreis oc
    if C≠∅ then EC := EC ∪ C fi
```

Fortsetzung

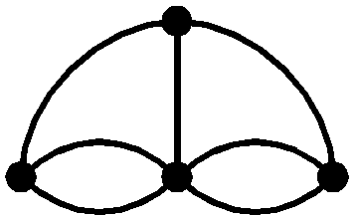
```
        if not empty(Q) then
          v := Q.remove()
        fi
      until E = ∅
    end
```

Laufzeit des Algorithmus: $\Theta(|E|)$.
Laufzeit der `while`-Schleife: $O(|E|)$, der `do`-`until`-Schleife ohne Durchlaufen der `while`-Schleife: $O(|V|)$ und damit ebenfalls $O(|E|)$, da der Graph zusammenhängend ist.

5.2 Hamiltonsche Pfade

Ein Pfad (Kreis) in einem Graphen (Digraphen) heißt hamiltonsch, wenn er jeden Knoten genau einmal enthält.
Ein Graph (Digraph) heißt hamiltonsch, wenn er einen hamiltonschen Kreis enthält.

Beispiel 309 (Das Königsberger Brückenproblem)



Dieser Graph besitzt einen hamiltonschen Kreis, aber weder einen eulerschen Kreis noch einen eulerschen Pfad.

Die Aufgabe, einen hamiltonschen Kreis zu finden, ist wesentlich schwerer als einen eulerschen Kreis zu finden; es ist ein $\mathcal{NP}$-vollständiges Problem.

6. Kürzeste Wege

Gegeben sind ein (Di)Graph $G = (V, E)$ und eine Gewichtsfunktion $w : E \to \mathbb{R}^+ \cup \{+\infty\}$. O. B. d. A. sei G vollständig, damit auch zusammenhängend.

Sei $u = v_0, v_1, v_2, \ldots, v_n = v$ ein Pfad in G. Die Länge dieses Pfades ist

$$\sum_{i=0}^{n-1} w(v_i, v_{i+1}).$$

d(u, v) sei die Länge eines kürzesten Pfades von u nach v.

Problemstellungen:

1. Gegeben $u, v \in V$, berechne $\mathrm{d}(u, v)$.
2. Gegeben $u \in V$, berechne für alle $v \in V$ die Länge $\mathrm{d}(u, v)$ eines kürzesten Pfades von u nach v (sssp, single source shortest path).
3. Berechne für alle $(u, v) \in V^2$ die kürzeste Entfernung $\mathrm{d}(u, v)$ (apsp, all pairs shortest path).

6.1 Der Floyd-Warshall-Algorithmus für apsp

Gegeben sind ein (Di)Graph $G = (V, E)$ und eine Gewichtsfunktion $w : E \to \mathbb{R}^+ \cup \{+\infty\}$. Sei o. B. d. A. $V = \{0, \ldots, n-1\}$. Eine Gewichtsmatrix ist wie folgt definiert:

$$D = \big(w(v_i, v_j)\big)_{\substack{0 \le i < n \\ 0 \le j < n}}$$

Ziel ist es, eine $n \times n$-Matrix mit den Einträgen

$$d_{ij} = \text{ Länge eines kürzesten Weges von } i \text{ nach } j$$

zu berechnen. Dazu werden induktiv Matrizen $D^{(k)}$ mit Einträgen

$$d_{ij}^{(k)} = \left(\begin{array}{c} \text{Länge eines kürzesten Weges von } i \text{ nach } j, \\ \text{so dass alle inneren Knoten} < k \text{ sind} \end{array} \right)$$

erzeugt.

```
algorithm Floyd
  for i=0 to n-1 do
    for j=0 to n-1 do
      D^0[i,j] := w(v_i,v_j)
    od
  od
  for k=0 to n-1 do
    for i=0 to n-1 do
      for j=0 to n-1 do
        D^{k+1}[i,j] := min{D^k[i,j],
                            D^k[i,k]+D^k[k,j]}
      od
    od
  od
end
```

Satz 310

Der Floyd-Algorithmus berechnet für alle $u, v \in V^2$ die Länge eines kürzesten Weges zwischen u und v, und zwar mit Zeitbedarf $\Theta(n^3)$ und Platzbedarf $\Theta(n^2)$.

Beweis:

Ersichtlich aus Algorithmus. □

Bemerkungen:

1. Zur Bestimmung der eigentlichen Pfade (und nicht nur der Entfernungen) muss bei der Minimum-Bestimmung jeweils das k gespeichert werden.
2. Der Algorithmus funktioniert auch, wenn *negative Kantengewichte* vorhanden sind, es jedoch keine *negativen Kreise* gibt.
3. Die Erweiterung auf Digraphen ist offensichtlich.

6.2 Dijkstras Algorithmus für sssp

Gegeben sind ein (Di)Graph $G = (V, E)$, ein Knoten $s \in V$ und eine Gewichtsfunktion $w : E \to \mathbb{R}^+ \cup \{\infty\}$.

```
algorithm Dijkstra
  F:=V\{s}
  for all v ∈ F do d[v] := w(s,v) od
  co d[s]=0 oc
  while F≠∅ do
    bestimme v ∈ F mit d[v] minimal
    F:=F\{v}
    for all w ∈ N(v) do
      d[w] := min{d[w], d[v]+w(v,w)}
    od
  od
end
```

Satz 311

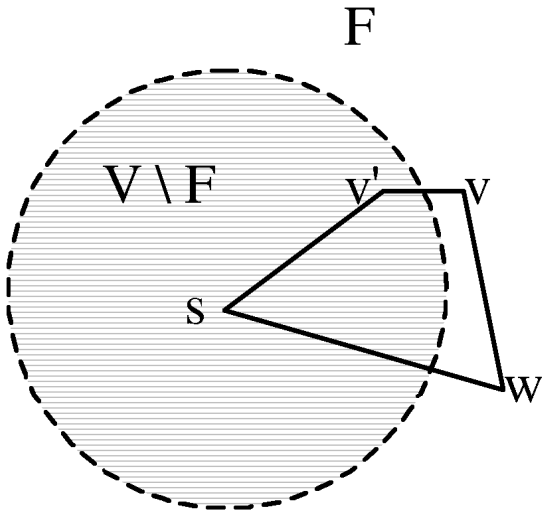
Dijkstras Algorithmus berechnet $d(s,v)$ für alle $v \in V$; der Zeitaufwand ist $O(n^2)$, der Platzbedarf $O(n+m)$.

Beweis:

Zeit- und Platzbedarf sind aus dem Algorithmus ersichtlich. Die Korrektheit zeigen wir mit einem Widerspruchsbeweis:

Annahme: v sei der erste Knoten, so dass $d(s,v)$ falsch (d. h. zu groß) berechnet wird.

Diese Situation illustriert folgendes Bild:



Nach Annahme muss dann gelten:

$$\mathsf{d}(w) + w(w,v) < \mathsf{d}(s,v') + \mathsf{d}(v',v) = \mathsf{d}(v) \ .$$

Damit wäre $\mathsf{d}(w)$ aber kleiner als $\mathsf{d}(v)$, und der Algorithmus hätte w und nicht v gewählt. □

Bemerkung:
Mit besseren Datenstrukturen (priority queues – z. B. Fibonacci heaps) kann Dijkstras Algorithmus so implementiert werden, dass er z. B. in Zeit $O(m + n \cdot \log n)$ läuft.

7. Matchings

Definition 312

Sei $G = (V, E)$ ein Graph.

1. $M \subseteq E$ heißt Matching, falls alle Kanten in M paarweise disjunkt sind.
2. M heißt maximales Matching, falls es kein Matching M' in G gibt mit $M \subsetneq M'$.
3. M heißt Matching maximaler Kardinalität (aka Maximum Matching), falls es in G kein Matching M' mit $|M'| > |M|$ gibt.
4. $\mathrm{m}(G)$ ist die Kardinalität eines Maximum Matchings in G.

Beispiel 313

maximales Matching
(aber nicht Maximum)

Maximum-Matching
(natürlich auch maximal)

7.1 Matchings in bipartiten Graphen

Satz 314 („Heiratssatz“)

Sei $G = (U, V, E)$ ein bipartiter Graph. Dann ist $\mathrm{m}(G) = |U|$ genau dann, wenn gilt:

$$(\forall A \subseteq U)\big[|A| \leq |N(A)|\big]$$

Beweis:

„$\Rightarrow$“

Offensichtlich.

„$\Leftarrow$“

Sei M ein Maximum Matching in G.
Annahme: Ein Knoten $u = u_0 \in U$ sei in M *ungematcht*.

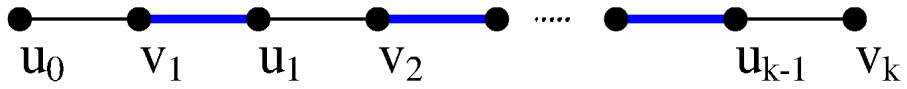
Wir beginnen in u_0 eine BFS, wobei wir in den ungeraden Schichten (also von U aus) nur ungematchte und in den geraden Schichten (also von V aus) nur gematchte Kanten verwenden. Querkanten bleiben außer Betracht.

Fall 1: Die BFS findet in V einen ungematchten Knoten v. Dann stoppen wir.

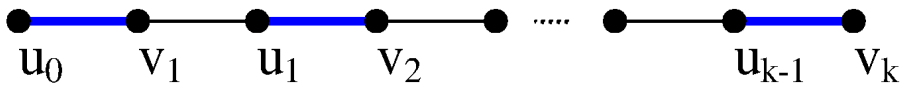
Fall 2: Nach Vollendung einer geraden Schicht (mit gematchten Kanten) sind alle Blätter des BFS-Baums gematcht. Seien U' (bzw. V') die Knoten des aktuellen BFS-Baums in U (bzw. V). Gemäß Annahme ist $|U'| > |V'|$, die alternierende BFS kann also fortgesetzt werden. Da G endlich ist, muss schließlich Fall 1 eintreten.

„⇐“(Fortsetzung)

Also existiert per Konstruktion ein Pfad wie in folgender Abbildung:



Ein solcher Pfad, bei dem sich gematchte und ungematchte Kanten abwechseln, heißt alternierender Pfad. Sind, wie hier, Anfangs- und Endknoten ungematcht, heißt der Pfad auch augmentierend.
Vertauscht man auf diesem Pfad gematchte und ungematchte Kanten, erhält man dadurch ein Matching M' mit $|M'| = |M| + 1$, was wiederum einen Widerspruch darstellt:



Definition 315

Man definiert für einen bipartiten Graphen $G = (U, V, E)$ die Kenngröße:

$$\delta := \delta(G) := \max_{A \subseteq U} \{ |A| - |N(A)| \}$$

Da bei der Maximumsbildung auch $A = \emptyset$ sein kann, ist $\delta \geq 0$.

Satz 316

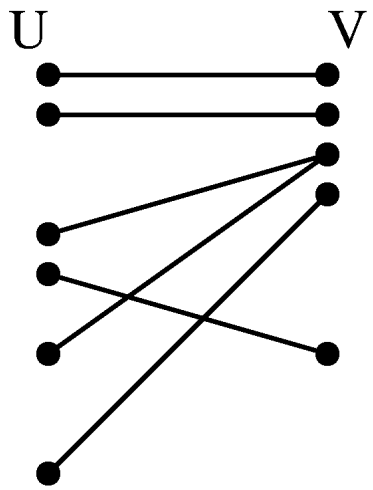
Es gilt:

$$\mathrm{m}(G) = |U| - \delta \ .$$

Beweis:

Dass $\mathrm{m}(G) \leq |U| - \delta$ gilt, ist offensichtlich. Wir zeigen nun noch, dass auch $\mathrm{m}(G) \geq |U| - \delta$ gilt, damit ist der Satz bewiesen.

Betrachte folgenden Graphen:



Man fügt nun δ neue Knoten hinzu. Von diesen gehen Kanten zu allen Knoten in U, so dass ein $K_{|U|,\delta}$ entsteht.

Der neue Graph erfüllt die Voraussetzungen des Heiratssatzes.
Damit gibt es im neuen Graphen ein Matching M' mit $|M'| = |U|$.
Daraus folgt, dass es im alten Graphen ein Matching der Kardinalität $\geq |U| - \delta$ geben muss. $\square$