Bemerkung:
Es gibt Algorithmen für minimale Spannbäume der Komplexität $O(m + n \cdot \log n)$
und, für dünnbesetzte Graphen, der Komplexität $O(m \cdot \log^* n)$, wobei

$$\log^* x = \min_{n \in \mathbb{N}} \left\{ n : \underbrace{\log\Big(\log\big(\cdots \log(x) \cdots\big)\Big)}_{n} < 1 \right\}.$$

5. Spezielle Pfade

5.1 Eulersche Pfade und Kreise

Definition 308

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 309

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

Beweis:

„⇒“

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.

Beweis (Forts.):

„$\Leftarrow$“

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 310

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

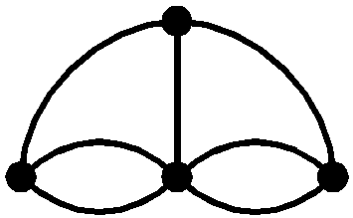
```
        if not empty(Q) then
          v := Q.remove()
        fi
      until E = ∅
    end
```

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 311 (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

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

$\mathrm{d}(u, v)$ 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

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

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 312

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 313

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

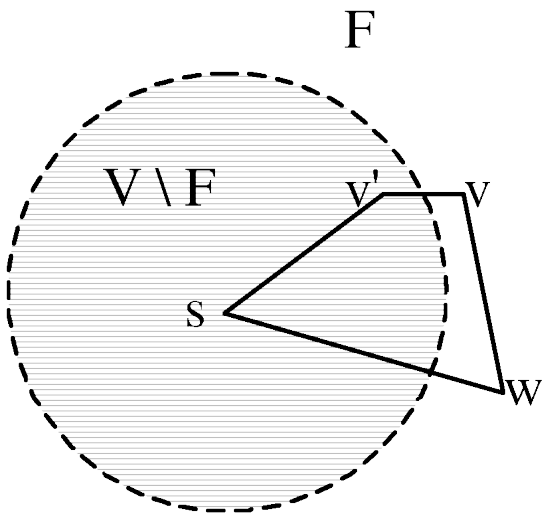
Beweis:

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

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

Beweis (Forts.):

Diese Situation illustriert folgendes Bild:



Nach Annahme muss dann gelten:

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

Damit wäre $\mathrm{d}(w)$ aber kleiner als $\mathrm{d}(v)$, 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 314

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 315

maximales Matching
(aber nicht Maximum)

Maximum-Matching
(natürlich auch maximal)

7.1 Matchings in bipartiten Graphen

Satz 316 („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.

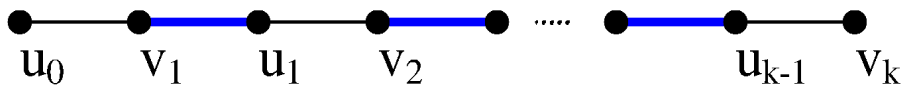
Fall 1: Die BFS findet in V einen ungematchten Knoten v. Dann stoppen wir.

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.

Beweis (Forts.):

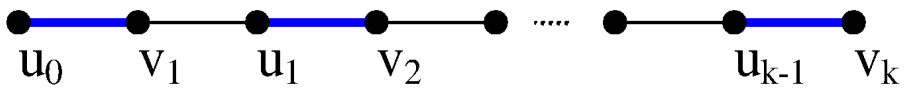
„$\Leftarrow$“(Fortsetzung)

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 317

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 318

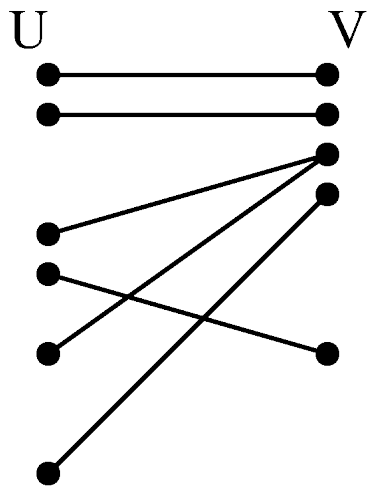
Es gilt:

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

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.

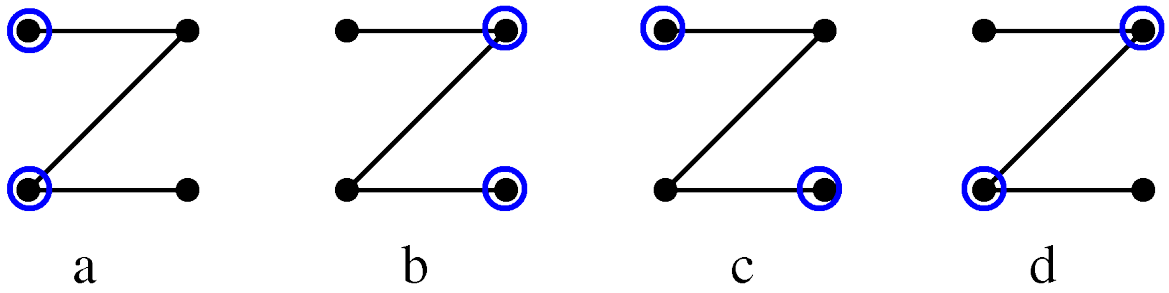
Beweis (Forts.):

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. $\square$

Definition 319

$D \subseteq U \uplus V$ heißt Träger oder Knotenüberdeckung (*vertex cover*, *VC*) von G, wenn jede Kante in G zu mindestens einem $u \in D$ inzident ist.

Beispiel 320



In den Fällen a, b und d sind Träger gezeigt, in c nicht.

Satz 321

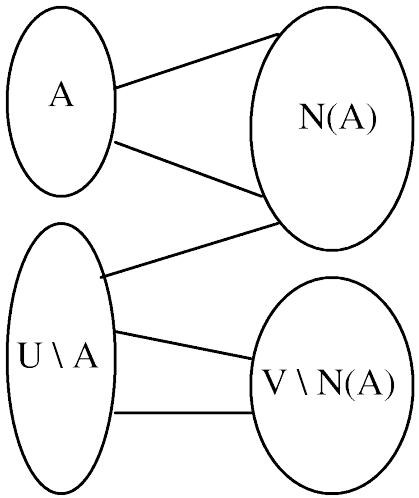
Es gilt:

$$\max\{|M|; M \text{ Matching}\} = \min\{|D|; D \text{ Träger}\}$$

Beweis:

„$\leq$“ Offensichtlich.

„$\geq$“ Für ein geeignetes $A \subseteq U$ gilt
$\mathrm{m}(G) = |U| - \delta(G) = |U \setminus A| + |N(A)|$:



$(U \setminus A) \cup N(A)$ ist Träger von G.

□

Sei

$$M = \left(m_{ij}\right)_{\substack{1 \le i \le n \\ 1 \le j \le n}}$$

eine (quadratische) Matrix mit $m_{ij} \geq 0$. Alle Zeilen- und Spaltensummen von M seien gleich $r > 0$.

Man ordnet nun M den bipartiten Graphen $G = (U, V, E)$ zu mit

$$U = \{u_1, \ldots, u_n\}, V = \{v_1, \ldots, v_n\} \text{ und } \{u_i, v_j\} \in E \Leftrightarrow m_{ij} > 0.$$

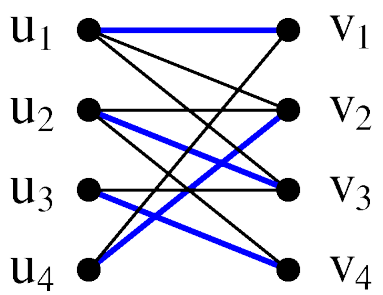
Ein Matching in G entspricht einer Menge von Positionen in M, die alle in verschiedenen Zeilen und Spalten liegen.

Beispiel 322

Die Matrix

$$\begin{pmatrix} \boxed{3} & 1 & 1 & 0 \\ 0 & 1 & \boxed{2} & 2 \\ 0 & 0 & 2 & \boxed{3} \\ 2 & \boxed{3} & 0 & 0 \end{pmatrix}$$

entspricht dem Graphen



Bemerkung:
Ein Träger D von G ist also eine Menge von Zeilen und Spalten von M, die zusammen alle Einträge $m_{ij} > 0$ enthalten.

Definition 323
Eine Menge von Positionen (in M), die alle in verschiedenen Zeilen und in verschiedenen Spalten liegen, heißt Diagonale von M.

Eine Diagonale der Größe n muss in M existieren, denn falls M keine solche Diagonale hat, gibt es nach Satz 321 e Zeilen und f Spalten mit $e + f < n$, die zusammen alle Einträge > 0 von M enthalten.
Die Gesamtsumme der Einträge in M wäre dann

$$n \cdot r = \sum_{i,j} m_{ij} \leq (e + f) \cdot r < r \cdot n,$$

was offensichtlich ein Widerspruch ist.

Sei c_1 der minimale Eintrag > 0 in M, und sei P_1 die zu einer Diagonale der Größe n gehörige Permutationsmatrix (d. h. Einträge $= 1$ an den Positionen der Diagonale, 0 sonst).

Dann gilt:

$$M_1 := M - c_1 P_1$$

ist eine $n \times n$-Matrix mit allen Zeilen- und Spaltensummen $= r - c_1$. Die Matrix M_1 enthält damit mehr Nullen als M.

Damit haben wir gezeigt:

Satz 324

Sei M wie oben. Dann gibt es für ein geeignetes k Konstanten $c_i > 0$ und Permutationsmatrizen P_i, $i = 1, \ldots, k$, so dass gilt

$$M = \sum_{i=1}^{k} c_i P_i \qquad \sum_{i=1}^{k} c_i = r \ .$$

7.2 Konstruktion optimaler Matchings

Satz 325

Ein Matching M ist genau dann Maximum, wenn es dazu keinen augmentierenden Pfad gibt.

Beweis:

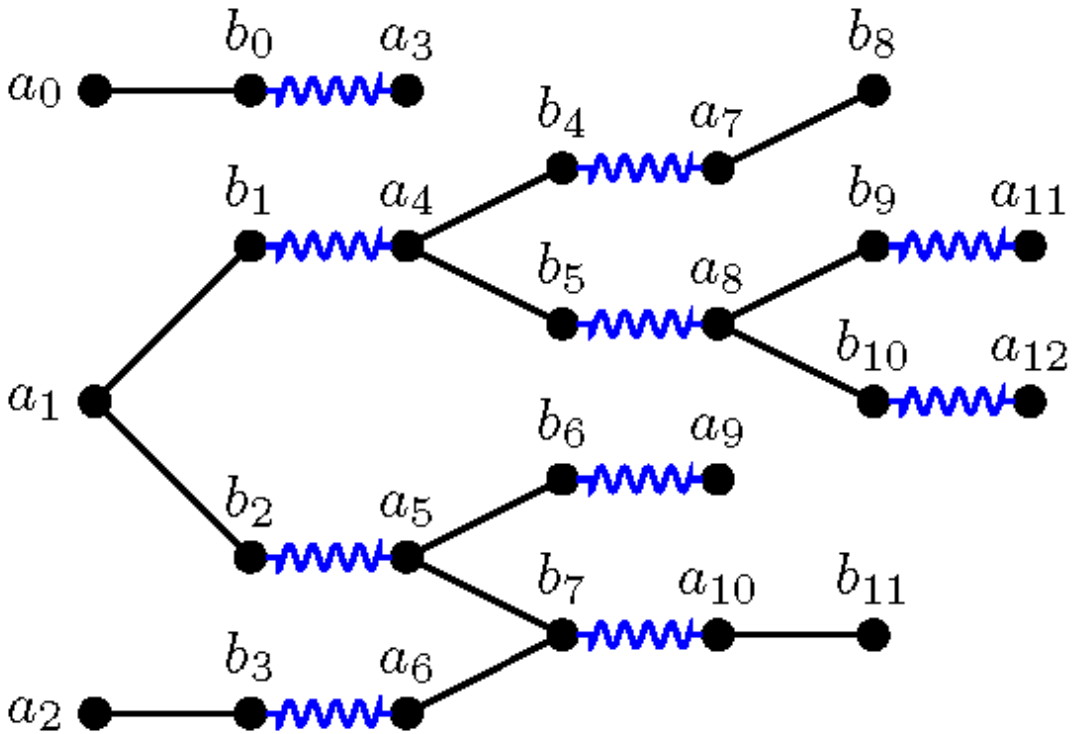
„$\Rightarrow$“ Offensichtlich.

„$\Leftarrow$“ Sei M ein Matching, zu dem es keinen augmentierenden Pfad gibt. Annahme, M sei kein Maximum Matching, es existiere ein Maximum Matching M'. Betrachte nun $M \Delta M'$. Die Zusammenhangskomponenten dieses Graphen sind alternierende Pfade und Kreise gerader Länge. Da $|M'| > |M|$ gilt, muss es einen alternierenden Pfad mit ungerader Länge geben, der mit Kanten aus M' beginnt und endet. Dies ist aber ein *augmentierender* Pfad.

□

Der Algorithmus zur Konstruktion optimaler Matchings ist eine parallele (simultane) alternierende Breitensuche.

Beispiel 326 (Konstruktion im bipartiten Graph)



Ergebnisse und Erweiterungen:

	bipartit	allgemein
ungewichtet	$O(\sqrt{\|V\|} \cdot \|E\|)$	$O(\sqrt{\|V\|} \cdot \|E\|)$
gewichtet	$O(\|V\| \cdot (\|E\| + \|V\| \cdot \log(\|V\|)))$	$O(\|V\| \cdot \|E\| \cdot \log(\|V\|))$

Siehe auch:
Zvi Galil: Efficient algorithms for finding maximum matchings in graphs, ACM Computing Surveys 18 (1986), pp. 23–38

Hier ist das offizielle Ende
der Vorlesung
Diskrete Strukturen
im Wintersemester 2010/11

Die folgenden Folien sind zusätzliches Material aus früheren Vorlesungen.

7.3 Reguläre bipartite Graphen

Lemma 327

Sei $G = (U, V, E)$ ein k-regulärer bipartiter Graph ($k \in \mathbb{N}$). Dann hat G ein perfektes Matching.

Beweis:

Sei $A \subseteq U$ und $B = N(A) \subseteq V$. Dann ist $|A| \leq |B|$, da ja alle von A ausgehenden Kanten in B enden und, falls $|B| < |A|$, es in B damit einen Knoten mit Grad $> k$ geben müsste. □

Korollar 328

Sei $G = (U, V, E)$ ein k-regulärer bipartiter Graph ($k \in \mathbb{N}$). Dann lässt sich E als disjunkte Vereinigung von k perfekten Matchings darstellen.

7.4 Transversalen

Definition 329

Sei $G = (U, V, E)$ ein bipartiter Graph, M ein Matching in G, und $A \subseteq U$ die in M gematchte Teilmenge der Knotenmenge U. Dann heißt A eine Transversale in G.

Satz 330

Sei $G = (U, V, E)$ ein bipartiter Graph, $\mathcal{T} \subseteq 2^U$ die Menge der Transversalen in G. Dann ist $(U, \mathcal{T})$ ein Matroid.

Beweis:

Die ersten beiden Bedingungen für ein Matroid sind klarerweise erfüllt:

1. $\emptyset \in \mathcal{T}$
2. $B \subset A, A \in \mathcal{T} \Rightarrow B \in \mathcal{T}$

Beweis (Forts.):

Seien nun A und A' Transversalen mit den zugehörigen Matchings M und M', und sei $|A'| = |A| + 1$, also auch $|M'| = |M| + 1$.
Betrachte $M' \Delta M$.
Dann muss $M' \Delta M$ (mindestens) einen Pfad ungerader Länge enthalten, der mit einer Kante in M' beginnt und mit einer Kante in M' endet (und dazwischen abwechselnd Kanten in M bzw. M' enthält). Dieser Pfad ist ein augmentierender Pfad bzgl. M, und einer der beiden Endpunkte liegt in $A' \setminus A$, kann also zu A hinzugenommen werden. □

Anwendung: gewichtetes Zuweisungsproblem, Variante 1

n Nutzer wollen jeweils auf eine aus einer nutzerspezifischen Teilmenge von insgesamt m Ressourcen zugreifen. Jede Ressource kann aber nur von höchstens einem Nutzer in Anspruch genommen werden. Der Wert einer Zuweisung von Ressourcen zu (interessierten) Nutzern ergibt sich als die Summe

$$\sum_{i \in A} w_i \ ,$$

wobei die Zuweisung einem Matching in dem durch Nutzer, Ressourcen und Zugriffswünsche gegebenen Graphen entspricht, $w_i \in \mathbb{R}^+$ ein Gewicht für jeden Nutzer $i \in \{1, \ldots, n\}$ ist, und A die durch die Zuweisung (das Matching) bedachte Teilmenge der Nutzer ist.

7.5 Gewichtetes Matching in bipartiten Graphen

Wir betrachten nun eine zweite Variante eines Zuweisungsproblems, das durch bipartite Graphen $G = (U, V, E)$ mit einer Gewichtsfunktion $w : E \to \mathbb{R}^+$ gegeben ist. Das Gewicht eines Matchings $M \subseteq E$ ist dann

$$\sum_{e \in M} w(e) \; .$$

Wir können o.B.d.A. annehmen, dass $|U| = |V|$ $(= n)$ und G vollständig bipartit (also $G = K_{n,n}$) ist, indem wir zunächst die kleinere der beiden Mengen U und V mit zusätzlichen Knoten auffüllen und dann die fehlenden Kanten durch Kanten mit Gewicht 0 ersetzen.

Damit suchen wir in G *optimale perfekte* Matchings. Wir können das Problem, ein perfektes Matching maximalen Gewichts zu finden, reduzieren auf das Problem, ein perfektes Matching minimalen Gewichts zu bestimmen, indem wir jedes Gewicht $w(e)$ durch

$$\max_{e \in E} w(e) \; - \; w(e)$$

ersetzen.

Wir nehmen daher an, dass wir o.B.d.A. ein perfektes Matching minimalen Gewichts in (G, w) suchen.

Für die folgende Diskussion nehmen wir zur Vereinfachung weiter an, dass alle Gewichte $\in \mathbb{N}_0$ sind.

Sei

$$W = \left(w_{ij}\right)_{\substack{1 \le i \le n \\ 1 \le j \le n}}$$

die zu (G, w) gehörige Gewichtsmatrix, und sei

$$P = \left(p_{ij}\right)_{\substack{1 \le i \le n \\ 1 \le j \le n}}$$

eine Permutationsmatrix (d.h., jede Zeile und jede Spalte von P enthält genau eine 1 und ansonsten nur Einträge 0).

Die Permutationsmatrix P entspricht einem perfekten Matching M in G mit Gewicht

$$\sum_{i,j} p_{ij} w_{ij} \ .$$

Beobachtung:
Wenn wir von jedem Element einer Zeile (oder Spalte) in W einen festen Betrag p subtrahieren, verringert sich das Gewicht eines jeden perfekten Matchings M um diesen Betrag p, die relative Ordnung (nach Gewicht) unter den perfekten Matchings bleibt bestehen, insbesondere gehen optimale Matchings wieder in optimale Matchings über.

Wir führen nun solche Zeilen- und Spaltenumformungen durch, um eine Diagonale mit möglichst vielen Einträgen $= 0$ zu erhalten.

Beispiel 331

Sei

$$W = \begin{pmatrix} 9 & 11 & 12 & 11 \\ 6 & 3 & 8 & 5 \\ 7 & 6 & 13 & 11 \\ 9 & 10 & 10 & 7 \end{pmatrix}$$

Nachdem wir von jeder Zeile das minimale Gewicht subtrahieren, erhalten wir

$$W' = \begin{pmatrix} 0 & 2 & 3 & 2 \\ 3 & 0 & 5 & 2 \\ 1 & 0 & 7 & 5 \\ 2 & 3 & 3 & 0 \end{pmatrix}$$

Beispiel (Forts.)

$$W' = \begin{pmatrix} 0 & 2 & 3 & 2 \\ 3 & 0 & 5 & 2 \\ 1 & 0 & 7 & 5 \\ 2 & 3 & 3 & 0 \end{pmatrix}$$

Nachdem wir von jeder Spalte das minimale Gewicht subtrahieren, erhalten wir

$$W'' = \begin{pmatrix} 0 & 2 & 0 & 2 \\ 3 & 0 & 2 & 2 \\ 1 & 0 & 4 & 5 \\ 2 & 3 & 0 & 0 \end{pmatrix}$$

Beispiel (Forts.)

Diese Matrix enthält eine Diagonale der Größe 3 mit Einträgen $= 0$*:*

$$W'' = \begin{pmatrix} \boxed{0} & 2 & 0 & 2 \\ 3 & 0 & 2 & 2 \\ 1 & \boxed{0} & 4 & 5 \\ 2 & 3 & \boxed{0} & 0 \end{pmatrix}$$

Aus Satz 321 folgt, dass die maximale Länge einer 0-Diagonale gleich der minimalen Anzahl von Zeilen und Spalten ist, die alle 0en bedecken.
Falls wir noch keine 0-Diagonale der Länge n haben, iterieren wir folgenden Algorithmus:

1. finde eine minimale Anzahl von e Zeilen und f Spalten $(e + f < n)$, die zusammen alle Einträge $= 0$ enthalten;
2. sei w das Minimum der nicht überdeckten Elemente;
3. subtrahiere w von den $n - e$ nicht überdeckten Zeilen;
4. addiere w zu den f überdeckten Spalten.

Die Gewichte ändern sich also wie folgt:

1. um $-w$, falls (i, j) nicht überdeckt ist;
2. um 0, falls (i, j) von einer Zeile *oder* Spalte überdeckt ist, aber nicht beides;
3. um $+w$, falls (i, j) von einer Zeile *und* einer Spalte überdeckt ist.

Insbesondere sind die resultierenden Gewichte wieder ≥ 0.
Die Anzahl der doppelt (von Zeilen *und* Spalten) überdeckten Positionen ist $e \cdot f$, die Anzahl der nicht überdeckten Positionen ist

$$n^2 - n(e+f) + ef \ .$$

Der resultierende Gewichtsunterschied ist daher

$$\begin{aligned} \Delta w &= (ef)w - (n^2 - n(e+f) + ef)w \\ &= (n(e+f) - n^2)w < 0 \end{aligned}$$

Damit muss unsere Iteration enden und wir finden eine 0-Diagonale der Länge n, entsprechend einer optimalen Zuordnung.

Beispiel (Forts.)

In unserem Beispiel ergibt sich

$$W'' = \begin{pmatrix} \boxed{0} & \boxed{2} & \boxed{0} & \boxed{2} \\ 3 & \boxed{0} & 2 & 2 \\ 1 & \boxed{0} & 4 & 5 \\ \boxed{2} & \boxed{3} & \boxed{0} & \boxed{0} \end{pmatrix}$$

Der Algorithmus bestimmt $w = 1$*:*

$$\Rightarrow \begin{pmatrix} 0 & 2 & 0 & 2 \\ \boxed{2} & \boxed{-1} & \boxed{1} & \boxed{1} \\ \boxed{0} & \boxed{-1} & \boxed{3} & \boxed{4} \\ 2 & 3 & 0 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 0 & 3 & \boxed{0} & 2 \\ 2 & \boxed{0} & 1 & 1 \\ \boxed{0} & 0 & 3 & 4 \\ 2 & 4 & 0 & \boxed{0} \end{pmatrix}$$

Beispiel (Forts.)

In dem durch die Matrix

$$W = \begin{pmatrix} 9 & 11 & \boxed{12} & 11 \\ 6 & \boxed{3} & 8 & 5 \\ \boxed{7} & 6 & 13 & 11 \\ 9 & 10 & 10 & \boxed{7} \end{pmatrix}$$

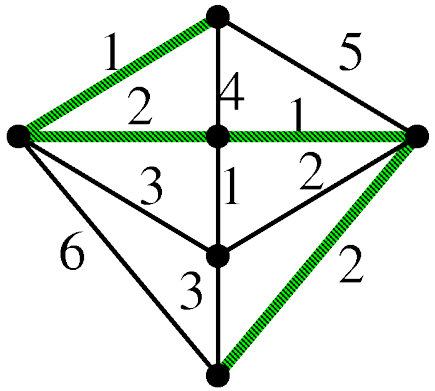
gegebenen bipartiten Graphen hat also das durch die markierten Kanten gegebene perfekte Matching minimales Gewicht.

Bemerkung: Bei geeigneter Implementierung ist die Laufzeit des Algorithmus $O(n^3)$.

7.6 Das Problem des chinesischen Postboten

Gegeben ist ein zusammenhängender, gewichteter Multigraph $G = (V, E, w)$.
Gesucht ist ein Kreis minimalen Gewichts, der jede Kante mindestens einmal enthält.

Beispiel 332 (In der optimalen Lösung werden die dickeren grünen Kanten zweimal verwendet)



Algorithmus: Sei U die Menge der Knoten ungeraden Grades, $|U| = 2k$.

1. Bestimme $\mathrm{d}(u, v)$ für alle $u, v \in U$.
2. Bestimme auf dem K_{2k} mit Kantengewichtung $w(\{u, v\}) = \mathrm{d}(u, v)$ ein perfektes Matching M minimalen Gewichts.
3. Füge die den Kanten in M entsprechenden kürzesten Pfade in G ein und bestimme im resultierenden Graphen einen Eulerkreis. Dieser ist eine Lösung.