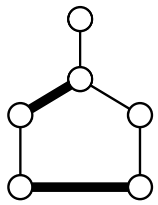
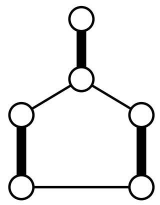
Matchings in Graphen

Es sei ein ungerichteter Graph $G = (V, E)$ gegeben. Ein *Matching* in G ist eine Teilmenge $M \subseteq E$, so dass keine zwei Kanten aus M einen Endpunkt gemeinsam haben. Ein Matching M heißt *maximal*, falls es in G kein größeres Matching M' mit $M' \supset M$ gibt. Ein Matching M heißt *Matching maximaler Kardinalität* (engl. *maximum matching*), falls es in G kein Matching M' mit $|M'| > |M|$ gibt. Wir sind an einem Algorithmus interessiert, der für einen gegebenen Graphen effizient ein Matching maximaler Kardinalität berechnet.



(a) maximales Matching



(b) Matching max. Kardinalität

Matching-Probleme treten in der Praxis meistens als Zuordnungsprobleme auf. Ein Beispiel könnte etwa die Zuordnung von Professoren zu Vorlesungen sein, wenn jeder Professor höchstens eine Vorlesung halten will: die Knoten des Eingabegraphen repräsentieren die Professoren und die Vorlesungen, eine Kante zwischen einem Professor und einer Vorlesung gibt an, dass der Professor die Vorlesung halten kann. Ein Matching maximaler Kardinalität entspricht dann genau einer Zuordnung von Professoren zu Vorlesungen, bei der so viele Vorlesungen angeboten werden können wie möglich.

Für einen Graphen $G = (V, E)$ und ein Matching $M \subseteq E$ nennen wir die Kanten aus M *gematcht* und die Kanten aus $E \setminus M$ *ungematcht*. Ein Knoten heißt *gematcht*, wenn eine seiner inzidenten Kanten gematcht ist. Knoten, die keine inzidente gematchte Kante haben, heißen *frei*. Hilfreich ist weiterhin das Konzept alternierender bzw. augmentierender Pfade bzgl. eines Matchings M:

(i) Ein einfacher Pfad $v_0, v_1, \ldots, v_r$ heißt *alternierend*, falls die Kanten (v_{i-1}, v_i) abwechselnd in M und in $E \setminus M$ sind.

(ii) Ein alternierender Pfad heißt *augmentierend*, falls er an beiden Enden ungematchte Kanten hat und nicht verlängert werden kann, also wenn beide Endknoten des Pfades frei sind.

Beachte, dass ein augmentierender Pfad auch aus einer einzigen ungematchten Kante zwischen zwei freien Knoten bestehen kann.

Man sieht leicht, dass ein Matching M mit Hilfe eines augmentierenden Pfades p zu einem Matching M' vergrößert werden kann: wenn man die gematchten Kanten des Pfades aus M herausnimmt und die ungematchten Kanten des Pfades in M einfügt, erhält man nämlich ein gültiges Matching mit einer zusätzlichen Kante. Diesen Prozess bezeichnen wir auch als *Invertieren* eines augmentierenden Pfades. Das obige Matching (b) lässt sich zum Beispiel aus Matching (a) durch ein derartiges Invertieren eines augmentierenden Pfades erhalten.

Satz 1 *Ein Matching M in $G = (V, E)$ hat genau dann maximale Kardinalität, wenn es keinen augmentierenden Pfad gibt.*

Beweis: Die Behauptung des Satzes ist äquivalent zu: Ein Matching M hat genau dann nicht maximale Kardinalität, wenn es einen augmentierenden Pfad gibt. Wir beweisen diese Behauptung. Die Richtung $\Leftarrow$ ist offensichtlich richtig. Die Richtung $\Rightarrow$ ist noch zu zeigen.

Sei M ein Matching in G, das nicht maximale Kardinalität hat. Sei M' ein Matching in G mit maximaler Kardinalität. Betrachte den Graphen $G' = (V, M \oplus M')$, wobei $M \oplus M' = (M \cup M') \setminus (M \cap M')$ die symmetrische Differenz von M und M' ist. Offensichtlich haben in G' alle Knoten Grad ≤ 2. Also besteht G' aus einer Reihe von einfachen Pfaden und Kreisen, die alle alternierend bzgl. M sind. M' lässt sich aus M durch Invertieren aller dieser Pfade und Kreise erhalten. Da das Invertieren von Kreisen und von nicht-augmentierenden Pfaden das Matching nicht vergrößert, müssen unter den Pfaden auch $|M'| - |M| \geq 1$ augmentierende Pfade sein. $\square$

Aus dem Beweis des Satzes folgt sogar, dass es in G bzgl. M genau $|M'| - |M|$ knotendisjunkte augmentierende Pfade gibt, wobei M' ein Matching maximaler Kardinalität ist. Da sich die $|M|$ gematchten Kanten auf diese $|M'| - |M|$ augmentierenden Pfade verteilen, lässt sich auch ableiten, dass es einen augmentierenden Pfad der Länge höchstens $2 \cdot \lfloor |M|/(|M'| - |M|) \rfloor + 1$ gibt.

Aufgrund des Satzes wissen wir auch, dass wir ein Matching maximaler Kardinalität finden können, indem wir mit einem beliebigen Matching M, zum Beispiel $M = \emptyset$, starten und so lange augmentierende Pfade suchen und diese invertieren, bis es keinen augmentierenden Pfad mehr gibt. Es ist nur noch zu klären, wie die Suche nach augmentierenden Pfaden realisiert werden soll.

Dabei stellt es sich zunächst als sehr günstig heraus, nicht beliebige augmentierende Pfade zu suchen, sondern eine maximale Menge *kürzester* augmentierender Pfade. Sei M das aktuelle Matching und sei ℓ die Länge (Anzahl Kanten) eines kürzesten augmentierenden Pfades bzgl. M. Dann suchen wir eine Menge $p_1, p_2, \ldots, p_r$ von knotendisjunkten augmentierenden Pfaden der Länge ℓ, so dass es keinen weiteren solchen knotendisjunkten Pfad mehr gibt, und invertieren alle diese Pfade $p_1, \ldots, p_r$. Dieses Vorgehen hat den Vorteil, dass wir höchstens $O(\sqrt{|V|})$-mal augmentierende Pfade suchen müssen, wie Hopcroft und Karp gezeigt haben. Lässt sich die Suche nach augmentierenden Pfaden in Zeit $O(|V| + |E|) = O(|E|)$ realisieren, erhält man also einen Matching-Algorithmus mit Gesamtlaufzeit $O(\sqrt{|V|}|E|)$. Im nächsten Abschnitt wird dies für den Fall bipartiter Graphen konkret beschrieben.

1 Matchings in bipartiten Graphen

Ein Graph $G = (V, E)$ heißt *bipartit*, wenn sich seine Knotenmenge V so in disjunkte Teilmengen V_1 und V_2 aufteilen lässt, dass alle Kanten einen Knoten aus V_1 mit einem aus V_2 verbinden. Bei vielen in der Praxis auftretenden Zuordnungsproblemen ist der entstehende Graph tatsächlich bipartit, so auch beim eingangs erwähnten Zuordnungsproblem mit Professoren und Vorlesungen.

In bipartiten Graphen lässt sich die Suche nach einer maximalen Menge knotendisjunkter augmentierender Pfade durch *simultane Breitensuche* und anschließende *Tiefensuche* effizient in linearer Zeit $O(|V| + |E|)$ realisieren (Algorithmus von Hopcroft und Karp, 1973). Genauer sieht der Algorithmus so aus:

- setze $M := \emptyset$

- **while** (true) **do**
  simultane Breitensuche;
  if (es existiert ein augmentierender Pfad)
  then Tiefensuche nach knotendisjunkten kürzesten augmentierenden Pfaden;
  else break;
  fi
  od

- gib M als Matching maximaler Kardinalität aus

Eine Ausführung des Rumpfes der while-Schleife nennen wir *Iteration*. Wir wissen, dass der Algorithmus auch im Worst-Case nur $O(\sqrt{|V|})$ Iterationen ausführt. Jede Iteration besteht aus einer simultanen Breitensuche und, falls das Matching noch nicht maximale Kardinalität hat, einer anschließenden Tiefensuche. Sowohl die simultane Breitensuche als auch die Tiefensuche können in Zeit $O(|V| + |E|)$ ausgeführt werden, so dass sich eine Gesamtlaufzeit von $O(\sqrt{|V|}|E|)$ ergibt.

1.1 Simultane Breitensuche

Bei gegebenem aktuellen Matching M ist es die Aufgabe der simultanen Breitensuche, erstens zu entscheiden, ob es in dem Graphen überhaupt noch augmentierende Pfade gibt, und zweitens den Knoten des Graphen Level-Werte zuzuordnen, mit deren Hilfe die anschließende Tiefensuche eine maximale Menge kürzester knotendisjunkter augmentierender Pfade finden kann.

Jeder Pfad in einem bipartiten Graphen G besucht abwechselnd einen Knoten in V_1 und einen in V_2. Da alle augmentierenden Pfade ungerade Länge haben, beginnen sie jeweils bei einem freien Knoten in V_1 und enden bei einem freien Knoten in V_2. Dabei verwenden sie in Richtung von V_1 nach V_2 jeweils eine ungematchte Kante und in Richtung von V_2 nach V_1 eine gematchte Kante. Wir ordnen jedem Knoten v einen Wert $level[v]$ zu, den wir am Beginn jeder simultanen Breitensuche mit -1 initialisieren und der am Ende der Breitensuche die Länge eines kürzesten alternierenden Pfades von einem freien Knoten aus V_1 zu v repräsentieren soll.

Um die Länge eines kürzesten augmentierenden Pfades zu bestimmen, lässt man eine Breitensuche gleichzeitig bei allen freien Knoten aus V_1 beginnen: aus diesem Grund spricht man von *simultaner* Breitensuche. Dabei initialisiert man die Queue einfach am Anfang nicht mit einem einzigen Startknoten, sondern fügt alle freien Knoten aus V_1 in die Queue ein. Für jeden dieser Knoten v setzt man außerdem $level[v] = 0$.

Die weitere Breitensuche läuft dann in *Phasen* ab, die abwechselnd vom Typ 1 bzw. vom Typ 2 sind:

Typ 1: zu Anfang der Phase bilden die Knoten aus der Queue eine Teilmenge von V_1; in der Phase wird jeder solche Knoten v aus der Queue geholt und alle **ungematchten** Nachbarkanten $e = \{v, w\}$ betrachtet: falls w noch nicht besucht wurde, setzt man $level[w] = level[v] + 1$ und fügt w hinten an die Queue an; am Ende der Phase besteht die Queue nur aus Knoten, die in V_2 sind.

Typ 2: zu Anfang der Phase bilden die Knoten aus der Queue eine Teilmenge von V_2; in der Phase wird jeder solche Knoten v aus der Queue geholt und seine **gematchte**

Nachbarkante $e = \{v, w\}$ betrachtet: falls w noch nicht besucht wurde, setzt man $level[w] = level[v] + 1$ und fügt w hinten an die Queue an; am Ende der Phase besteht die Queue aus Knoten, die in V_1 sind.

Die Breitensuche terminiert, wenn die Queue nach einer Phase vom Typ 1 einen freien Knoten w enthält (dann gibt es einen augmentierenden Pfad der Länge $level[w]$, der bei w endet, und es wird die anschließende Tiefensuche gestartet) oder wenn die Queue leer wird (dann gibt es keinen augmentierenden Pfad und das aktuelle Matching hat bereits maximale Kardinalität).

1.2 Tiefensuche nach augmentierenden Pfaden

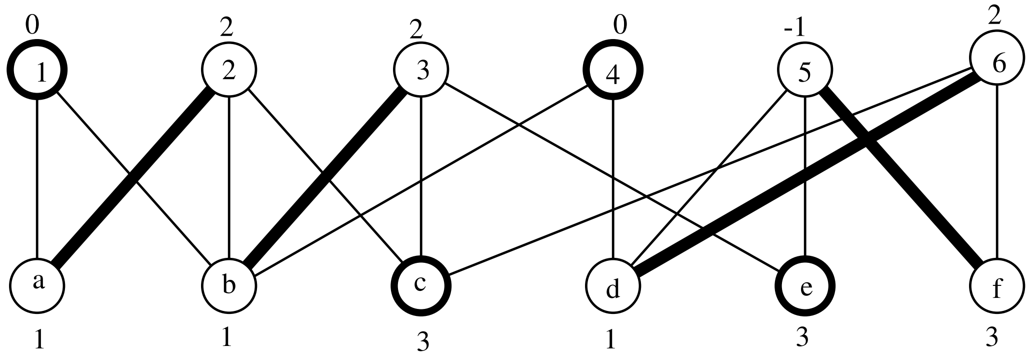
Wenn bei der simultanen Breitensuche nach ℓ Phasen ein freier Knoten aus V_2 erreicht wurde, so wissen wir jetzt, dass es mindestens einen kürzesten augmentierenden Pfad der Länge ℓ gibt, und wollen nun mittels Tiefensuche eine maximale Menge solcher kürzester augmentierender Pfade berechnen. Genauer werden wir dazu nicht eine einzige Tiefensuche verwenden, sondern für jeden freien Knoten v aus V_1 eine eigene Tiefensuche starten, um von v aus (falls möglich) einen augmentierenden Pfad der Länge ℓ zu finden. Bei diesen Tiefensuchen betrachten wir ausschließlich Kanten, die den folgenden Bedingungen genügen (andere Kanten werden ignoriert):

- Ist der aktuelle Knoten $u \in V_1$, so betrachten wir von u aus Kanten $e = \{u, w\}$ mit $e \in E \setminus M$ und $level[w] = level[u] + 1$.
- Ist der aktuelle Knoten $u \in V_2$, so betrachten wir von u aus Kanten $e = \{u, w\}$ mit $e \in M$ und $level[w] = level[u] + 1$.

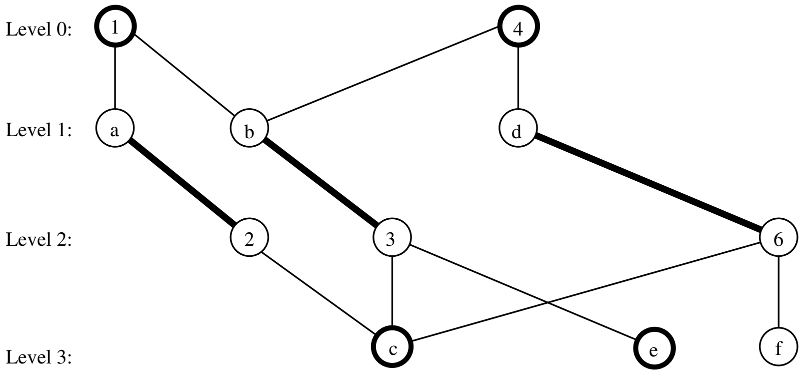
Erreicht eine solche Tiefensuche von einem freien Knoten v aus V_1 einen freien Knoten w aus V_2, so ist der zugehörige Pfad von v nach w ein kürzester augmentierender Pfad. In diesem Fall wird der Pfad sofort invertiert (Kanten, die in M waren, werden aus M herausgenommen, und Kanten, die noch nicht in M waren, werden in M eingefügt), alle Knoten u auf dem Pfad werden durch Setzen von $level[u] = -1$ für weitere Tiefensuchen gesperrt, und die nächste Tiefensuche wird beim nächsten freien Knoten aus V_1 gestartet. Läuft eine Tiefensuche in eine Sackgasse, d.h. es wurde noch kein augmentierender Pfad gefunden und es gibt vom aktuellen Knoten u aus keine den obigen Bedingungen genügende Kante mehr, so wird $level[u] = -1$ gesetzt und die Tiefensuche am Vorgänger von u fortgesetzt. Die Level-Werte der Knoten werden während der Tiefensuchen nur dann verändert, wenn ein augmentierender Pfad oder eine Sackgasse gefunden wurde: in diesem Fall wird durch Setzen des Level-Wertes auf -1 erreicht, dass die betroffenen Knoten für den Rest der Tiefensuche ignoriert werden.

1.3 Beispiel des Ablaufs einer Iteration

Der bipartite Graph $G = (V_1 \cup V_2, E)$ mit $V_1 = \{1, 2, 3, 4, 5, 6\}$ und $V_2 = \{a, b, c, d, e, f\}$ und das aktuelle Matching M seien wie folgt gegeben (gematchte Kanten sind fett gezeichnet, freie Knoten sind dick umrandet):



Die Zahlen außerhalb der Knoten geben den $level$-Wert an, der sich bei der simultanen Breitensuche ergibt. Nach der dritten Phase der simultanen Breitensuche befinden sich die Knoten c, e und f in der Queue. Da darunter auch zwei freie Knoten (c und e) sind, terminiert die simultane Breitensuche, und es wird nacheinander bei den freien Knoten aus V_1, also bei Knoten 1 und 4, jeweils eine Tiefensuche gestartet. Dabei werden nur die folgenden Kanten betrachtet:



Es könnten dann z.B. die Pfade 1, a, 2, c und 4, b, 3, e gefunden werden, die eine maximale Menge knotendisjunkter kürzester augmentierender Pfade darstellen. (Auch der Pfad 4, b, 3, c alleine wäre übrigens eine solche Menge.) Nach Invertierung dieser beiden Pfade ergibt sich folgendes Matching, das bereits maximale Kardinalität hat.