Eine Zusammenstellung aus Prüfungsprotokollen
bei Professor Verbeek mit vielen Fragen zur

Prüfungsvorbereitung Diplomprüfung Effiziente Graphenalgorithmen

Thomas Schwarze
Thomas.Schwarze@FernUni-Hagen.de

4. Februar 2004

Diese Arbeit basiert auf einer Zusammenstellung von Sandra Gesing und wurde mit dem Textsystem TEX erstellt.

Alle angeführten Nummerierungen beziehen sich auf das Buch *Graphen und Algorithmen* von Andreas Brandstädt

Fragen zu „Effiziente Graphenalgorithmen“

1 Graphen und algorithmische Graphenprobleme

1.1 Was ist ein paarer Graph?

Def. 1.1.5

$G = (V, E)$ heißt *paarer Graph* genau dann, wenn

$\exists$ eine Zerlegung von V in zwei unabhängige Knotenmengen X, Y mit
$\forall e \in E : e \notin \mathcal{P}_2(X) \wedge e \notin \mathcal{P}_2(Y)$.
$\forall (x, y) \in E : x \in X \wedge y \in Y$

$\mathcal{P}_2(X)$ bezeichnet dabei die Zweiermenge von X.

Eine Knotenmenge $U \subseteq V$ heißt *unabhängige Knotenmenge* in G genau dann, wenn $G(U, E')$ leerer Graph ist ($E' = \emptyset$).

1.2 Was ist NP–vollständig?

Def. 1.4.1

Seien A und B die Beschreibungen algorithmischer Probleme.

A heißt NP–vollständig genau dann, wenn

(1) $A \in \mathrm{NP}$

(2) $\forall B \in \mathrm{NP} : B \leq_{\mathrm{pol}} A$ (jedes B aus NP läßt sich in Polynomialzeit auf A reduzieren).

$A \in \mathrm{NP}$: A kann nichtdeterministisch in polynomieller Zeit gelöst werden.

$B \in \mathrm{NP}$ polynomialzeit reduzierbar auf A: Es existiert eine polynomialzeitberechenbare Funktion f mit der Eigenschaft $\forall x \in A \Leftrightarrow f(x) \in B$.

Das Grundkonzept eines NP–Vollständigkeitsbeweises lässt sich in vier Schritte einteilen:

(1) Zeige, dass $A \in \mathrm{NP}$ ist. Dies ist im Allgemeinen einfach, eine Lösung wird nichtdeterministisch geraten und dann deterministisch verifiziert.

(2) Auswahl eines geeigneten als NP–vollständigen Problems B. Ein Beweis, dass sich A polynomiell auf B reduzieren lässt, sollte möglichst einfach sein. Dies ist oft der Fall, wenn sich A und B sehr ähnlich sind. Die Ähnlichkeit von Problemen ist oft nicht leicht zu erkennen.

(3) Angabe einer deterministisch in polynomieller Zeit berechenbaren Transformation f, die Eingabe für A in Eingaben für B überführt.

(4) Beweis der Eigenschaft: $\forall x \in A \Leftrightarrow f(x) \in B$.

3 Durchsuchen von Graphen – Knotenreihenfolgen von Graphen

3.1 Welche Verfahren zum Durchsuchen von Graphen kennen Sie?

DFS (Depth-First-Search, Tiefensuche) und

BFS (Breadth-First-Search, Breitensuche)

3.2 Was ist der Unterschied zwischen Tiefen- und Breitensuche? Wie funktioniert die Tiefensuche?

Tiefensuche:

Der Algorithmus arbeitet rekursiv für jede Zusammenhangskomponente von einem Startknoten aus alle Pfade ab.

Auf ungerichteten Graphen:

Algo. 3.1.1

Eingabe: Ungerichteter Graph $G = (V, E)$, der durch seine Adjazenzlisten beschrieben wird

Ausgabe: Ein Wald von G mit gerichteten Kanten (eine Menge B von gerichteten Kanten, die für jede Zusammenhangskomponente ein Gerüst liefert), eine Numerierung $t(v)$ der Knoten, die die Durchlaufreihenfolge durch V bei DFS angibt.

Benutzte Datenstrukturen:
Adjazenzlisten (von deren Reihenfolge hängt die Reihenfolge beim Durchsuchen von G wesentlich ab)
Liste L mit direktem Zugriff (ein Zeiger z in L zeigt auf den ersten durch die Suche noch nicht erreichten Knoten in L, solange noch ein solcher Knoten existiert)
Array t(v) (Initialisierung mit ∞, danach Nummer nach der Reihenfolge ihres Erreichens)
Menge B

Algorithmus:

(1) Initialisierung von $B, L, t(v)$

(2) Solange ein Knoten unerreicht ist, mache (3) (Prozedur DFSEARCH(v))

(3) Nummeriere aktuellen Knoten und mache (4)

(4) Solange ein Knoten aus der Adjazenzliste des aktuellen Knotens unerreicht ist, markiere Nachbar, speichere aktuelle Kante und rufe (3) auf mit Nachbar

Die Kantenmenge E von G wird in *Baumkanten* und *Rückwärtskanten* zerlegt, hat also keine *Querkanten*, da DFS alle Nachbarn sofort durchläuft.

Auf gerichteten Graphen:

Algo. 3.3.1

Eingabe: Gerichteter Graph $G = (V, E)$, der durch seine Adjazenzlisten beschrieben wird

Ausgabe: Eine Zerlegung von E in Baumkanten T, Vorwärtskanten F, Rückwärtskanten B und Querkanten C sowie Numerierungen $t(v), f(v)$ aller Knoten aus V.

Benutzte Datenstrukturen:
Adjazenzlisten (von deren Reihenfolge hängt die Reihenfolge beim Durchsuchen von G wesentlich ab)
Array t(v) (Initialisierung mit ∞, danach Nummer nach der Reihenfolge des Erreichens der Knoten)
Array f(v) (Initialisierung mit ∞, danach Nummer nach der Reihenfolge, wann die Bearbeitung der Knoten abgeschlossen wurde)
Mengen T, F, B, C

Algorithmus:

(1) Initialisierung von $T, F, B, C, t(v), f(v)$

(2) Solange ein Knoten unerreicht ist, mache (3) (Prozedur DFSEARCH(v))

(3) Mache (4) und (5) über alle Nachbarn

(4) Wenn Nachbar noch nicht erreicht, speichere aktuelle Baumkante, markiere Nachbar als erreicht und rufe (3) auf mit Nachbar

(5) Wenn Nachbar erreicht, speichere aktuelle Kante als Vorwärtskante, Rückwärtskante oder Querkante

Im Algorithmus wird die Prozedur DFSEARCH für jeden Startknoten einer Zusammenhangskomponente aufgerufen.

G ist zusammenhängend genau dann, wenn DFS auf G nur eine Kellerphase benötigt, d.h. der Keller ist erst am Ende der Ausführung der Prozedur wieder leer.

Implementierung des Algorithmus über das Sprachelement Rekursion oder Stack.

Laufzeit:

$O(|V| + |E|)$

Kantenarten:

- ungerichtete Baumkanten (Kanten des Gerüsts)
- (un)gerichtete Rückwärtskanten (Kanten außerhalb des Gerüsts zu Vorgängern)
- Querkanten (alle anderen Kanten, werden beim DFS für gerichtete Graphen gespeichert)

Breitensuche:

Anders als bei Tiefensuche geht man bei Breitensuche von einem schon erreichten Knoten nicht sofort weiter in die Tiefe, sondern bearbeitet zunächst alle Nachbarn eines erreichten Knoten. Dazu wird eine Warteschlange W eingerichtet, die alle Nachbarn eines gerade erreichten Knotens v aufnimmt. (Löschen und Hinzufügen eines Knotens in W an verschiedenen Enden der Liste W). Algo. 3.4.1

Eingabe: Ungerichteter Graph $G = (V, E)$, der durch seine Adjazenzlisten beschrieben wird

Ausgabe: Ein Wald von G mit gerichteten Kanten (eine Menge B von gerichteten Kanten, die für jede Zusammenhangskomponente ein Gerüst liefert), eine Numerierung $b(v)$ der Knoten, die die Durchlaufreihenfolge (Reihenfolge in W) durch V bei BFS angibt.

Benutzte Datenstrukturen:
Adjazenzlisten (von deren Reihenfolge hängt die Reihenfolge beim Durchsuchen von G wesentlich ab)
Liste L mit direktem Zugriff (ein Zeiger z in L zeigt auf den ersten durch die Suche noch nicht erreichten Knoten in L, solange noch ein solcher Knoten existiert)
Array $b(v)$ (Initialisierung mit ∞, danach Nummer nach der Reihenfolge ihres Erreichens)
Menge B, W

Algorithmus:

(1) Initialisierung von $B, W, L, b(v)$

(2) Solange ein Knoten unerreicht ist und W leer ist, mache (3) und (4)

(3) Füge aktuellen Knoten in W ein, nummeriere ihn und markiere ihn als erreicht

(4) Solange $W \neq \emptyset$, lösche aktuellen Knoten aus W, mache (5) (Prozedur BFSEARCH(v))

(5) Für alle noch nicht erreichten Knoten aus der Adjazenzliste des aktuellen Knotens, markiere Nachbar als erreicht, speichere aktuelle Kante und füge Nachbar in W ein.

Im Algorithmus wird die Prozedur BFSEARCH für jeden Startknoten einer Zusammenhangskomponente aufgerufen.

Laufzeit:

$O(|V| + |E|)$

3.3 Welche klassischen Anwendungen kennen Sie, in denen Breiten- und Tiefensuche typischerweise genutzt werden? Wofür kann man DFS verwenden?

Tiefensuche:

- In $O(|V| + |E|)$ entscheidbar, ob ungerichteter Graph G kreisfrei bzw. Baum ist.
- Die vielleicht wichtigste Anwendung von DFS für gerichtete Graphen ist die Bestimmung der *stark zusammenhängenden Komponenten* eines gerichteten Graphen (in Linearzeit!)

  Gerichteter Graph ist *stark zusammenhängend* $\Leftrightarrow$ für alle Knoten u, v existiert ein Pfad von u nach v und einer von v nach u

  Stark zusammenhängende Komponente: Maximal stark zusammenhängender Teilgraph

- Bestimmung von Artikulationspunkten und zweifach zusammenhängende Komponenten eines Graphen

  Ist (V, B) ein DFS-Baum von $G = (V, E)$, so ist die Wurzel r von (V, B) genau dann *Artikulationspunkt* von G, wenn $\text{outdeg}(r) \geq 2$ in (V, B) gilt.

  D.h.: Die Wurzel des DFS-Baumes ist genau dann Artikulationspunkt des Graphen, wenn sie mindestens zwei Söhne hat (weil es keine Querkanten gibt).

  Sei $L(v)$ die kleinste DFS-Nummer von Knoten, die man von v aus über Baumkanten, gefolgt von höchstens einer Rückwärtskante, erreichen kann. Ein Nichtwurzelknoten u ist genau dann Artikulationspunkt von G, wenn eine Baumkante $u \to v$ existiert mit $L(v) \geq t(u)$. (Hinter einem Artikulationspunkt vom Startknoten aus gesehen gibt es immer Knoten, deren L-Wert mindestens so groß ist wie die DFS-Nummer des Artikulationspunkts. Für alle Punkte „vor“ dem Artikulationspunkt gilt das nicht.)

  v Artikulationspunkt $\Leftrightarrow G(V \backslash v)$ hat mehr Zusammenhangskomponenten als $G(V)$

  G zweifach zusammenhängend $\Leftrightarrow$ G ist zusammenhängend und hat keinen Artikulationspunkt

  Zweifach zusammenhängende Komponente $\Leftrightarrow$ maximal zusammenhängender Teilgraph

  $\tilde{G}$: Graph der zweifach zusammenhängenden Komponenten:
  Artikulationspunkt wird zusätzlich zwischen den Komponenten eingefügt

- Topologisches Sortieren

  Topologische Ordnung eines dag (gerichteter Graph ohne gerichtete Kreise): Knotenreihenfolge (Tupel), bei der für alle Kanten $(v, w) v$ vor w steht.

  Algorithmus:

  Eingabe: Ein dag in Adjazenzlistendarstellung

  Ausgabe: Eine topologische Ordnung $\text{topnum}(v)$

  (1) Stecke alle Knoten v mit $\text{indeg}(v) = 0$ in eine Queue Q

  (2) Solange $Q \neq \emptyset$, mache (3)

(3) Entnimm einen Knoten aus Q, nummeriere ihn, verringere den indeg aller Nachbarn und füge Nachbarn mit indeg$= 0$ in Q ein.

Laufzeit:

$\mathrm{O}(|V| + |E|)$

- Test für paare Graphen durch abwechselndes Färben der Knoten und Berücksichtigung der Rückwärtskanten.

Breitensuche:

- Algorithmus von Dinitz zur Lösung des Maximalflussproblems (Schichtennetzwerk, Phase 1), dessen Grundlage BFS ist (s.u.)
- Paralleles Suchen im Labyrinth.

4 Minimalgerüste, greedy–Algorithmus und Matroide

4.1 Was ist ein Minimalgerüst?

Def. 4.1.1

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

$H = (V, E')$ heißt *Gerüst* oder auch *Spannbaum* von G genau dann, wenn $E' \subseteq E$ und H ist Baum.

H heißt *Minimalgerüst* auf G genau dann, wenn H minimales Gewicht unter allen Gerüsten von G hat.

4.2 Welche Verfahren kennen Sie, um Minimalgerüste zu bestimmen?

Es existieren mehrere Verfahren, die alle auf folgendem Lemma beruhen: Lemma 4.1.1

Teilt man die Kantenmenge eines zusammenhängenden ungerichteten Graphen auf in Mengen U und $V \backslash U$ und bestimmt eine Kante e_0 zwischen U und $V \backslash U$ mit minimalem Gewicht, so gibt es ein Minimalgerüst T, das die Kante e_0 enthält.

(Zum Konstruieren eines Minimalgerüsts kann man also immer die kleinste Kante vom Bekannten zum Unbekannten nehmen.)

Beweis:

Es sei T ein Minimalgerüst von G. Falls T e_0 nicht enthält, so füge e_0 zu T hinzu. Wegen der Gerüsteigenschaft von T entsteht damit genau ein Kreis. Dieser Kreis enthält mindestens e_0 und eine weitere Kante $e' = (u, v)$ mit $u \in U$ und $v \in V \backslash U$. Außerdem ist $c(e_0) \leq c(e')$. Bildet man nun $(T \cup \{e_0\}) \backslash \{e'\})$, so ist dies ein Gerüst von G, das ebenfalls Minimalgerüst ist.

Die drei Algorithmen:

Algorithmus von Jarnik, Prim, Dijkstra

Algo. 4.1.1

Eingabe: Ein endlicher ungerichteter zusammenhängender Graph G mit $V = \{1, \ldots, n\}$ und eine Kostenfunktion $c : e \to \mathbb{R}$ mit $c(i, j) = \infty$ genau dann, wenn $\{i, j\}$ keine Kante ist.

Ausgabe: Ein Minimalgerüst von G

Datenstrukturen:

Menge B der Kanten
Menge V
Menge U der besuchten Knoten
Array c über Kosten,
Array d über Kosten von dem Startknoten aus (Metaknoten in U)

Algorithmus:

(1) Markiere den ersten Knoten als besucht, initialisiere B mit leerer Menge, U mit erstem Knoten, d mit c

(2) Solange nicht alle Knoten besucht sind, mache (3)

(3) Bestimme die kleinste Kante, die von U nach $V \backslash U$ geht, markiere die Kante und den neuen Knoten als besucht, minimiere d für alle Knoten aus $V \backslash U$ (setze d auf den Abstand zwischen den Knoten aus $V \backslash U$ zum jeweils nächstgelegenen aus U)

Laufzeit:

$\mathrm{O}(|V|^2)$

$\mathrm{O}(|V| + |E|)$ bei Verwendung von AF-heaps (Ableitungen von Fibonacci-heaps)

J aus Matroid (E, J) entspricht dem Minimalgerüst.

Algorithmus von Kruskal

Algo. 4.1.2

Dieser Algorithmus nutzt aus, dass sich die Gewichte der Kanten in $\mathrm{O}(m \cdot \log m) = \mathrm{O}(m \cdot \log n)$ Schritten der Größe nach sortieren lassen und man Zeit spart, wenn man für kleines m diese Kantenreihenfolge nutzt; dabei entstehen zunächst evtl. mehrere disjunkte Teilgerüste und Kanten werden hinzugenommen, wenn sie bisher disjunkte Teilgerüste miteinander verbinden.

Eingabe: Ein endlicher ungerichteter zusammenhängender Graph G mit $V = \{1, \ldots, n\}$, eine Kostenfunktion $c : e \to \mathbb{R}$ mit $c(i,j) = \infty$ genau dann, wenn $\{i,j\}$ keine Kante ist und aufsteigend sortierten Kanten.

Ausgabe: Ein Minimalgerüst von G

Algorithmus:

(1) Initialisiere das Mengensystem MS mit allen Knoten einzeln. Führe (2) aus, bis es nur noch eine Menge gibt.

(2) Wähle eine minimale Kante (v, w), streiche sie in der Kantenliste. Gehören v und w zu verschiedenen Menge in MS, so vereinige diese Mengen und füge (v, w) zum Gerüst hinzu.

Laufzeit:

$\mathrm{O}(|E| \cdot \log |V|)$

$O(|E| \cdot \log|E|)$

(es geht erheblich schneller, aber nicht weiter erklärt, in welcher Ordnung)

J aus Matroid (E, J) entspricht allen Wäldern des Graphen G

Laufzeitvergleich Jarnik, Prim, Dijkstra mit Kruskal:

Für Graphen, die viele Kanten haben, ist der Algorithmus von Jarnik, Prim, Dijkstra überlegen, sonst der Algorithmus von Kruskal.

Algorithmus von Boruvka, Sollin

Algo. 4.1.3

Der Algorithmus eignet sich besonders zur parallelen Ausführung. Er beginnt mit einem Wald, bei dem jeder Baum aus einem isolierten Knoten besteht. In einer Iteration bestimmt der Algorithmus simultan für jeden Baum des Waldes die kürzeste Kante, die einen Knoten mit einem anderen Baum verbindet.

Alle diese Kanten werden zu einem zu konstruierenden Minimalgerüst hinzugefügt, wobei gesichert werden muß, dass keine Zyklen entstehen.

Dieser Prozess wird fortgesetzt, bis nur noch ein Baum vorhanden ist. Da sich bei jeder Iteration (bis auf Spezialfälle am Ende) die Zahl der Bäume mindestens halbiert, braucht der Algorithmus höchstens $O(\log|V|)$ Iterationen.

Eingabe: Ein endlicher ungerichteter zusammenhängender Graph G mit $V = \{1, \ldots, n\}$, eine Kostenfunktion $c : e \to \mathbb{R}$ mit $c(i, j) = \infty$ genau dann, wenn $\{i, j\}$ keine Kante ist und lexikographisch sortierten Kanten.

Ausgabe: Ein Minimalgerüst von G

Algorithmus:

(1) Initialisiere das Mengensystem MS mit allen Knoten einzeln. Führe (2) aus, bis es nur noch eine Menge gibt.

(2) Wähle für jeden Teilbaum die lexikographisch minimale Kante zu einem anderen Teilbaum, füge diese Kante zum Gerüst hinzu und vereinige die Knotenmengen.

Laufzeit:

$O(|E| \cdot \log|V|)$

(ergibt sich aus dem Sortieraufwand für die Kantenfolge und der Zahl $\log n$ der Iterationen)

J aus Matroid (E, J) entspricht allen Wäldern des Graphen G.

4.3 Wie funktioniert der Algorithmus von Kruskal?

(s.o.)

4.4 Korrektheitsnachweis von Kruskal?

Induktiv durch wiederholte Anwendung von Lemma 4.1.1:

Teilt man die Kantenmenge eines zusammenhängenden ungerichteten Graphen auf in Mengen U und $V \setminus U$ und bestimmt eine Kante e_0 zwischen U und $V \setminus U$ mit minimalem Gewicht, so gibt es ein Minimalgerüst T, das die Kante e_0 enthält.

(Zum Konstruieren eines Minimalgerüsts kann man also immer die kleinste Kante vom Bekannten zum Unbekannten nehmen.)

Induktionsanfang: für $n \in \{0, 1, 2\}$ trivial

Induktionsvoraussetzung: für n wird ein Minimalgerüst ermittelt

Induktionsschritt: für $n + 1$ Anwendung des Satzes zwischen dem Minimalgerüst mit n Knoten und dem hinzugefügten Knoten.

4.5 Wie funktioniert der greedy-Algorithmus?

Algo. 4.2.1

Eingabe: Eine endliche Menge E und eine Kostenfunktion $c : E \to \mathbb{R}$, einen Mengensystem $J \in \mathcal{P}(E)$ mit $\emptyset \in J$.

Ausgabe: $I \subseteq J$.

Algorithmus

(1) Solange noch Elemente in E sind, mache (2)

(2) Entnimm ein Element $e \in E$ mit maximalen Kosten und füge es zu I hinzu, falls $(I \cup e) \in J$ ist.

Gesamtkosten:

$c(I) = \sum_{e \in E} c(e)$

Bei der Bestimmung von Minimalgerüsten geht es um Minimum-Bildung statt Maximum-Bildung wie bei dem greedy-Algorithmus. Dies lässt sich jedoch leicht durch Umformulieren erreichen, indem statt der Kostenfunktion $c(e), e \in E$, die Kostenfunktion $c'(e) = c_{max} - c(e)$ verwendet wird.

4.6 Was bedeutet hier die Bezeichnung „greedy“?

Der Name greedy (= gefräßig, raffgierig) erklärt sich dadurch, dass ein greedy-Algorithmus nach der Methode „Nimm immer das größte Stück“ vorgeht. Dadurch wird immer ein lokales Optimum bestimmt.

4.7 Kann man mit dem greedy-Algorithmus auch ein globales Extremum bestimmen? Wann ist der greedy-Algorithmus optimal?

Der greedy-Algorithmus ist optimal genau dann, wenn $c(I) = max\{c(I') : I \in J\}$ für jede Kostenfunktion $c : E \to \mathbb{R}$.

Das greedy-Prinzip ist optimal für (E, J) ist Matroid - ohne den Aufwand, der beim dynamischen Programmieren betrieben werden muß (Listen, Arrays zum Auffinden des globalen Optimums).

Ist $J \subseteq \mathcal{P}(E)$ nach unten abgeschlossen, so sind folgende Aussagen äquivalent: Satz 4.2.1

(1) Der greedy-Algorithmus ist optimal.

(2) Sind $I_p, I_{p+1} \in J$ mit $|I_p| = p$ und $|I_{p+1}| = p + 1$, so existiert ein Element e aus $I_{p+1} \backslash I_p$ mit $(I_p \cup \{e\}) \in J$ (e – ein Austauschelement)

(3) (E, J) ist Matroid

Die Austauscheigenschaft ist äquivalent zu:

Für jedes $A \subseteq E$ haben alle maximalen Untermengen von A, die in J liegen, dieselbe Mächtigkeit.

Beweis (1) $\Rightarrow$ (2):

Angenommen, (2) wäre nicht erfüllt und es existieren $I_p, I_{p+1} \in J$ mit $|I_p| = p$ und $|I_{p+1}| = p + 1$ und für kein Element $e \in I_{p+1} \backslash I_p$ ist $(I_p \cup \{e\}) \in J$. Sei $c(e) = p + 2$ für $e \in I_p$, $c(e) = p + 1$ für $e \in I_{p+1} \backslash Ip$, $c(e) = 0$ für $e \notin I_p \cup I_{p+1}$, dann findet der greedy-Algorithmus nur $p(p + 2) = 2p^2 + p$, dabei wäre $(p + 1)(p + 1) = p^2 + 2p + 1$ das Maximum.

4.8 Was ist ein Matroid?

Def. 4.2.2

Sei E eine endliche Menge und $J \subseteq \mathcal{P}(E)$.

$M(E, J)$ heißt *Matroid* genau dann, wenn $J \neq \emptyset$ und es gelten die folgenden zwei Bedingungen:

(1) Für alle $A \in J, B \subseteq A$ gilt $B \in J$. (J ist nach unten Teilmengen-abgeschlossen)

(2) Für jedes $A \subseteq E$ haben alle maximalen Untermengen von A, die in J liegen, dieselbe Mächtigkeit. Diese Mächtigkeit ist *Rang* von A.

Weitere Matroideigenschaften:

Sei $M = (E, J)$ ein Matroid. Für $A \subseteq E$ ist Def. 4.3.1

$\text{rang}_M(A) = \max\{|X| : X \subseteq A \wedge X \in J\}$

die Rangfunktion von M. (Maximale Größe einer Teilmenge von $A \in J$).

Für $A, B \subseteq E$ und $x, y \in E$ gilt: Satz 4.3.1

(1) $0 \leq \text{rang}(A) \leq |A|$

(2) $A \in J$ gdw. $\text{rang}(A) = |A|$

(3) Ist $A \subseteq B$, so gilt $\text{rang}(A) \leq \text{rang}(B)$

(4) $\text{rang}(A) \leq \text{rang}(A \cup \{x\}) \leq \text{rang}(A) + 1$

(5) $\text{rang}(A) + \text{rang}(B) \geq \text{rang}(A \cup B) + \text{rang}(A \cap B)$

(6) Ist $\text{rang}(A \cup \{x\}) = \text{rang}(A \cup \{y\}) = \text{rang}(A)$, so ist $\text{rang}(A \cup \{x, y\}) = \text{rang}(A)$

Es sei E eine nicht notwendig endliche Menge und $\sigma : \mathcal{P}(E) \to \mathcal{P}(E)$ eine Funktion.
σ heißt *Hüllenoperator* auf E genau dann, wenn Def. 4.3.2

(1) Für alle $A \subseteq E$ gilt $A \subseteq \sigma(A)$

(2) Für alle $A, B \subseteq E$ mit $A \subseteq B$ gilt $\sigma(A) \subseteq \sigma(B)$.

(3) Für alle $A \subseteq E$ gilt $\sigma(\sigma(A)) = \sigma(A)$.

Jeder Hüllenoperator ist genau dann Hüllenoperator eines Matroids, wenn zusätzlich gilt: Folg. 4.3.1

(4) $\forall x, y \in E, \forall X \subseteq E$ gilt $y \notin \sigma(X) \land y \in \sigma(X \cup \{x\}) \to x \in \sigma(X \cup \{y\})$

Beispiele: z.B. 4.2.1

(1) J = Menge der *linear unabhängigen* Mengen in einem endlichdimensionalen Vektorraum E über einem endlichen Körper.

(2) Es sei $G = (V, E)$ zusammenhängender ungerichteter Graph und J die Menge der kreisfreien Teilmengen von E. Dann ist (E, J) Matroid, da folgendes gilt: Ist $A \subseteq E$ und sind $I, I' \subseteq A$ maximal bzgl. J, sowie $V_1, \ldots, V_k$ die durch A definierten Zusammenhangskomponenten, so sind die Einschränkungen von I, I' auf V_i jeweils Gerüste auf V_i. Diese haben genau $V_i - 1$ Kanten, also ist $|I| = |I'|$.

4.9 Ist der Algorithmus von Jarnik, Prim, Dijkstra optimal?

Der Algorithmus bestimmt ein Minimalgerüst, also optimal.

4.10 Kennen Sie ein Beispiel, bei dem der Algorithmus von Jarnik, Prim, Dijkstra nicht optimal ist?

TSP.

Jarnik, Prim und Dijkstra finden ein Optimum in Polynomialzeit, TSP ist aber NP-vollständig. Wäre der Algorithmus von Jarnik, Prim und Dijkstra optimal für das TSP, dann wäre P = NP bewiesen.

Erklärung für greedy-Algorithmus:

Soll der Algorithmus die Lösung dieses Problems liefern, so muß der Lösungsraum J die Kantenmengen aller Hamiltonkreise auf G enthalten. Diese Kantenmengen sind die maximalen Mengen in J. Außerdem enthält J dann alle Teilmengen dieser Mengen.

(Beispiel mit vollständigen Graph $|V| = 4$)

Es können maximale Untermengen von A gefunden werden, die in J liegen, die verschiedene Größe haben $\Rightarrow$ J ist kein Matroid $\Rightarrow$ Algorithmus ist nicht optimal.

6 Das Maximalflussproblem

6.1 Was ist das Maximalflussproblem?

Def. 6.1.3

Das Maximalflussproblem ist die folgende Aufgabe:

Eingabe: Ein Netzwerk N

Ausgabe: Ein Maximalfluss f auf N

6.2 Definition von Netzwerk?

Def. 6.1.1

$N = (G, q, s, c)$ ist Netzwerk genau dann, wenn

(1) $G = (V, E)$ ist gerichteter endlicher schlichter Graph,

(2) $q, s \in V$ sind spezielle Knoten $q \neq s$, (q = Quelle, s = Senke)

(3) $c : E \to \mathbb{R}^+$ ist eine Kantengewichtsfunktion mit $c(e) > 0$ für alle Kanten $e \in E$.

6.3 Wie bestimmt man den Maximalfluss?

Algo. 6.2.1

Der Algorithmus von Ford/Fulkerson

Der Algorithmus von Ford/Fulkerson sucht sukzessive nach flussvergrößernden Pfaden, vergrößert den Fluss entlang dieser Pfade und hält an, falls kein solcher Pfad mehr existiert.

Eingabe: Ein Netzwerk N

Ausgabe: Ein Maximalfluss f auf N

Algorithmus:

(1) Beginne mit dem Nullfluss

(2) Solange ein flussvergrößernder Pfad existiert, konstruiere einen solchen und erhöhe den Gesamtfluss durch diesen Pfadfluss

Max-flow-min-cut-Theorem:

In Netzwerken mit ganzzahligen Kapazitäten ist der maximale Gesamtfluss F ganzzahlig und wird nach endlich vielen Schritten im Ford/Fulkerson–Algorithmus durch $F = c(S)$ für eine Schnittmenge S erreicht.

Man kann Netzwerke mit irrationalen Kapazitäten konstruieren, so dass der Algorithmus nicht hält.

Zugehörige Definitionen:

Auf Pfaden von q nach s kann man Vorwärtskanten und Rückwärtskanten unterscheiden.

Sei $P = (e_1, \dots, e_k)$ eine Kantenfolge und f eine Flussfunktion auf N. Der Δ-Wert einer Kante ist bei Vorwärtskanten Kapazitäten $-$ Fluss, bei Rückwärtskanten der Fluss dieser Kante. Def. 6.2.2

$$\Delta(e_i) := \begin{cases} c(e_i) - f(e_i) & \text{falls } e_i \text{ Vorwärtskante in } P \text{ ist} \\ f(e_i) & \text{sonst} \end{cases}$$

Eine Kante ist nützlich in einer Richtung, wenn der Fluss in dieser Richtung größer sein könnte.

Ein Pfad P von q nach s heißt nützlich, wenn die Flussvergrößerung $\Delta(P) = \min\{\Delta(e_i)\}$ positiv ist.

Durch Berücksichtigung eines flussvergrößerndes Pfades entsteht aus einem Fluss f ein neuer Fluss f': Def. 6.2.3

$$f'(e) := \begin{cases} f(e) + \Delta(P) & \text{falls } e \text{ Vorwärtskante in } P \text{ ist} \\ f(e) - \Delta(P) & \text{falls } e \text{ Rückwärtskante in } P \text{ ist} \\ f(e) & \text{falls } e \text{ nicht in } P \text{ vorkommt} \end{cases}$$

Der Algorithmus von Dinitz

Algo. 6.3.4

Es werden nicht nacheinander sondern simultan flussvergrößernde Pfade gleicher Länge bestimmt und damit ein bestimmtes Schichtennetzwerk gesättigt. Danach wird ein neues Schichtennetzwerk mit mehr Schichten konstruiert, das nur noch flussvergrößernde Pfade größerer Länge enthalten kann, bis eine maximal mögliche Länge erreicht ist. Grundlage dieses Vorgehens ist BFS.

Führe die Phasen 1–3 immer wieder aus, bis es keinen nützlichen Pfad von q nach s mehr gibt (das wird in Schritt 1 entdeckt). Die Anzahl der erforderlichen Durchläufe ist höchstens n, weil Pfade immer länger werden und höchstens eine Länge von n erreichen können.

Phase 1: Algo. 6.3.1

Konstruktion eines Schichtennetzwerkes aus nützlichen Kanten ($\mathrm{O}(n + m)$)

- Startknoten ist erste Schicht
- Über nützliche Kanten (vorwärts oder rückwärts) wird die nächste Schicht erreicht
- Endknoten ist die letzte Schicht (s)

Phase 2: Algo. 6.3.2

Bestimmung eines pfadsättigenden Flusses ($O(n \cdot m)$)

(1) Laufe mit DFS von q bis s und vergrößere den Fluss auf diesem Pfad sättigend

(2) Markiere dabei ausgelastete Kanten und Kanten in Sackgassen als blockiert (backtracking)
(Ist das Kantengewicht konstant 1, so werden gleich alle Kanten blockiert)

(3) Wiederhole 1 - 3, bis je Pfad mindestens eine Kante blockiert ist

Phase 3: Algo. 6.3.3

Flussvergrößerung

- Durchlaufe alle Kanten und trage alle in Phase 2 bestimmten Flüsse mit richtigem Vorzeichen ein

Laufzeit: Folgerung 6.3.1

$O(|V|^2 \cdot |E|)$, da Phase 2 eine Komplexität von $O(|V| \cdot |E|)$ hat und diese höchstens $|V|$-fach ausgeführt wird.

7 Unabhängige Knoten- und Kantenmengen

7.1 Was versteht man unter einer maximalen Zuordnung?

Def. 7.1.1

Es sei $G = (V, E)$ ungerichteter Graph. Eine Kantenmenge $E' \subseteq E$ heißt Zuordnung oder unabhängige Kantenmenge (matching) genau dann, wenn je zwei Kanten aus E' keinen Knoten gemeinsam haben.

$\alpha_E(G)$ ist die maximale Größe einer Zuordnung ($\max\{|E'| : E'$ Zuordnung in $G\}$).

Eine Zuordnung E' heißt größte Zuordnung genau dann, wenn $|E'| = \alpha_E(G)$.

7.2 Wie bestimmt man die größte Zuordnung?

Für paare Graphen:

(Zuordnungsproblem: Bestimmung einer größten Zuordnung, im paaren Graphen auch als Heiratsproblem bezeichnet)

Das Zuordnungsproblem für paare Graphen ist auf das Maximalflussproblem zurückführbar.

Das Netzwerk $N(G, q, s, c)$ für paare Graphen G wird konstruiert durch hinzufügen von q und s, ungerichtete Kanten werden durch gerichtete in eine Richtung ersetzt, gerichtete Kanten zu q und s hinzugefügt und $c(e) = 1$ für alle entstandenen Kanten gesetzt. Insbesondere ist durch $c(e) = 1$ für alle e, was zur Folge hat, dass im Algorithmus von Dinitz in flussvergrößernden Pfaden jeweils nicht nur eine, sondern alle Kanten des Pfades gesperrt werden. Dadurch Laufzeit höchstens von $O(|V| \cdot |E|)$.

Durch Streichung von Kanten im konstruierten Netzwerk und beobachten des Flusses, ob der Maximalfluss sich ändert, finden einer maximalen Zuordnung.

$\alpha_E(G) =$ Maximalfluss F

Satz von König:

Folgerung 7.1.2

Für paare Graphen ist $\alpha_E(G) = \beta_V(G)$,
d.h. die Maximalgröße einer Zuordnung ist gleich der Minimalgröße einer Knotenüberdeckung.

Für beliebige Graphen:

Die Ergänzungspfadmethode ist die Bestimmung einer größten Zuordnung, indem mit leerer Zuordnung begonnen wird und so lange Ergänzungen ausgeführt werden, bis kein Ergänzungspfad mehr existiert. Das Problem dabei ist das Auffinden der Ergänzungspfade, da G Kreise ungerader Länge enthalten kann.

Sei $G = (V, E)$ ungerichteter schlichter Graph und $M \subseteq E$ eine Zuordnung in G.

M-alternierender Pfad: Pfad, dessen Kanten abwechselnd in M und $E \setminus M$ liegen.

M-Ergänzungspfad: M-alternierender Pfad, der mit Knoten x beginnt und y endet, die nicht durch M abgedeckt werden.

M ist genau dann eine Zuordnung maximaler Größe, wenn kein M-Ergänzungspfad existiert.

7.3 Komplexität des Zuordnungsproblems für paare Graphen?

Folgerung 7.1.1

Das Zuordnungsproblem für paare Graphen lässt sich durch Dinitz in $O(|V| \cdot |E|)$ Schritten lösen, wobei im Skript sogar noch $O(\sqrt{|V|} \cdot |E|)$ angegeben ist.

7.4 Komplexität der Bestimmung einer maximalen Zuordnung im beliebigen Graphen?

Vor Def. 7.3.3

Das Zuordnungsproblem für allgemeine Graphen lässt sich in $O(\sqrt{|V|} \cdot |E|)$ Schritten lösen.

7.5 Was ist denn, wenn ich ein Netzwerk habe, in dem es mehrere mögliche Maximalflüsse gibt?

Der Ford/Fulkerson Algorithmus bestimmt genau einen Maximalfluss und damit eine maximale Zuordnung in paaren Graphen.

7.6 Bei Ford/Fulkerson müssen ja nicht unbedingt ganzzahlige Flüsse herauskommen?

Wir definieren $c(e) = 1$, damit kommen ganzzahlige Flüsse heraus.

7.7 Es gibt verschiedene Kenngrößen. Was können Sie über deren Komplexität sagen?

$\alpha_V(G)$: *max.* Größe einer *unabhängigen Knotenmenge*

$\alpha_E(G)$: *max.* Größe einer *unabhängigen Kantenmenge (Zuordnung)*

$\beta_V(G)$: *min.* Größe einer *Knotenüberdeckung*

$\beta_E(G)$: *min.* Größe einer *Kantenüberdeckung*

$\chi_V(G)$: *chromatische Zahl* von G:

minimale Farbenanzahl für Knotenfärbung

min$\{k$: es existiert eine Knotenfärbung $V_1 \cup \ldots \cup V_k$ von $G\}$

$\chi_E(G)$: *chromatischer Index* von G:

minimale Farbenanzahl für Kantenfärbung

min$\{k$: es existiert eine Kantenfärbung von $E_1, \ldots, E_k$ von $G\}$

Knotenüberdeckung: Überdeckung aller Kanten durch Knoten $(e \in E, V' \subseteq V)$

Kantenüberdeckung: Überdeckung aller Knoten durch Kanten $(v \in V, E' \subseteq E)$

Eine Zerlegung in disjunkte Knotenmengen heißt *Knotenfärbung*
$\Leftrightarrow$ jede Knotenmenge ist unabhängige Knotenmenge Def. 7.5.1
$\Leftrightarrow$ keine zwei gleichfarbigen Knoten haben eine Kante gemeinsam

Eine Zerlegung in disjunkte Kantenmengen heißt *Kantenfärbung*
$\Leftrightarrow$ jede Kantenmenge ist Zuordnung (unabhängige Kantenmenge) Def. 7.6.1
$\Leftrightarrow$ keine zwei gleichfarbigen Kanten haben einen Knoten gemeinsam

Größe	*In kubischen/ beliebigen Graphen*	*In paaren Graphen*
$\alpha_V(G)$ INDEPENDENT SET max. unabhäng. Knotenmenge	NP–vollständig für kubische Graphen $\alpha_V + \beta_V = \|V\|$, da β_V NP-vollständig	Polynomiell $\alpha_V = \beta_E$
$\alpha_E(G)$ Zuordnung, Matching	Polynomiell Ergänzungspfadmethode $O(\sqrt{\|V\|} \cdot \|E\|)$	Polynomiell Maximalfluss: $O(\|V\| \cdot \|E\|)$
$\beta_V(G)$ VERTEX COVER min. Knotenüberdeckung	NP–vollständig für kubische Graphen	Polynomiell Satz von König $\beta_V = \alpha_E$
$\beta_E(G)$ min. Kantenüberdeckung	Polynomiell Satz von Gallai: $\alpha_E + \beta_E = \|V\|$	Polynomiell
$\chi_V(G)$ chromatische Zahl	NP–vollständig für kubische Graphen	Linear da Test auf paare Graphen
$\chi_E(G)$ chromatischer Index	NP–vollständig für kubische Graphen	Polynomiell mit Ergänzungspfadmethode

7.8 Kann man mit einem Netzwerk auch die zu einer maximalen Zuordnung gehörenden Kanten bestimmen?

Ja, Dinitz bestimmt den Pfad. Alle Kanten mit Fluss $= 1$.

7.9 Was wissen Sie über unabhängige Knotenmengen?

Was wissen Sie über unabhängige Kantenmengen?

Beliebige Graphen: Eigenschaft 7.2.1

$S \subseteq V$ ist unabhängige Knotenmenge genau dann, wenn $V \backslash S$ Knotenüberdeckung ist.

$\alpha_V(G) + \beta_V(G) = |V|$ Folgerung 7.2.1

Beliebige Graphen mit Minimalgrad > 0: Satz 7.2.1

Gallai: $\alpha_E(G) + \beta_E(G) = |V|$

Paarer Graph mit Minimalgrad > 0: Folgerung 7.2.2

$\alpha_V(G) = \beta_E(G)$

Paare Graphen: Satz 7.1.1

$\alpha_E(G) =$ Maximalfluss F

Satz von König: Folgerung 7.1.2

Für paare Graphen ist $\alpha_E(G) = \beta_V(G)$,
d.h. die Maximalgröße einer Zuordnung ist gleich der Minimalgröße einer Knotenüberdeckung.

7.10 Wie bestimmt man unabhängige Knotenmengen in paaren Graphen, in beliebigen Graphen?

Im paaren Graphen:

$\alpha_V(G) + \beta_V(G) = |V|$, Folgerung 7.2.1
$S \subseteq V$ ist unabhängige Knotenmenge genau dann, wenn $V \backslash S$ Knotenüberdeckung ist. Eigenschaft 7.2.1
Nach König ist $\alpha_E(G) = \beta_V(G) \implies \alpha_V(G) = |V| - \beta_E(G)$.

7.11 In welcher Komplexität kann ich denn eine unabhängige Knotenmenge bestimmen?

NP–vollständig, Reduktion auf VERTEX COVER

7.12 Wie findet man die α_E?

Im paaren Graphen: Maximalflussbestimmung, Beispiel 7.1.1

im beliebigen Graphen: Ergänzungspfadmethode Satz 7.3.1

7.13 Wie kann man erkennen, dass α_V NP–vollständig ist?

α_V ist INDEPENDENT SET Selbsttest 1.4.1

$\alpha_V(G) + \beta_V(G) = |V|$ Folgerung 7.2.1

Wenn $\beta_V(G)$ (VERTEX COVER) NP–vollständig ist, ist $\alpha_V(G)$ auch NP–vollständig.

7.14 Wie sieht es mit der Komplexität von χ_E und χ_V aus?

$\chi_V(G)$	NP–vollständig für bel. Graphen	Linear für paare Graphen
$\chi_E(G)$	NP–vollständig für bel. Graphen	Polynomiell für paare Graphen

7.15 Was heißt Knoten- und Kantenfärbungen?

Eine Zerlegung in disjunkte Kantenmengen heißt *Kantenfärbung*
$\Leftrightarrow$ jede Kantenmenge ist Zuordnung (unabhängige Kantenmenge)
$\Leftrightarrow$ keine zwei gleichfarbigen Kanten haben einen Knoten gemeinsam

Eine Zerlegung in disjunkte Knotenmengen heißt *Knotenfärbung*
$\Leftrightarrow$ jede Knotenmenge ist unabhängige Knotenmenge
$\Leftrightarrow$ keine zwei gleichfarbigen Knoten haben eine Kante gemeinsam

7.16 Was ist die chromatische Zahl, was der chromatische Index?

$\chi_V(G)$: *chromatische Zahl* von G:

minimale Farbenanzahl für Knotenfärbung

min{k: es existiert eine Knotenfärbung $V_1 \cup \ldots \cup V_k$ von G}

$\chi_E(G)$: *chromatischer Index* von G:

minimale Farbenanzahl für Kantenfärbung

min{k: es existiert eine Kantenfärbung von $E_1, \ldots, E_k$ von G}

7.17 Was sind minimale Knotenüberdeckungen?

$\alpha_V(G) + \beta_V(G) = |V|$

$S \subseteq V$ ist unabhängige Knotenmenge genau dann, wenn $V \backslash S$ Knotenüberdeckung ist.

7.18 Idee für Algorithmus für 2-Farbigkeit

Test über DFS für paare Graphen durch abwechselndes Färben der Knoten und Berücksichtigung der Rückwärtskanten.

7.19 Wie zeigt man die Äquivalenz zwischen der fehlenden Existenz eines Ergänzungspfades und der Maximalität der Zuordnung?

Angenommen, es existiert ein M-Ergänzungsweg P. P beginnt und endet mit Knoten, die durch M überdeckt sind, d.h. die erste und letzte Kante von P ist aus $E \backslash M$: P enthält j Kanten aus M und $j+1$ Kanten aus $E \backslash M$. Außerdem ist zu keinem Knoten aus P eine weitere Kante aus M inzident, da M Zuordnung ist. Setzt man nun $M' = (M \backslash P) \cup (P \backslash M)$, so ist $|M'| = |M| + 1$, und M' ist ebenfalls Zuordnung.