

Vorlesung

Effiziente Algorithmen und Komplexitätstheorie

Sommersemester 2008

Ingo Wegener

Vorlesung

Ingo Wegener



Kontakt Ingo.Wegener@tu-dortmund.de
Otto-Hahn-Straße 14, Raum 302

mehr Informationen

<http://ls2-www.cs.uni-dortmund.de/~wegener>

Zur Vorlesung

Vorlesung „Effiziente Algorithmen und Komplexitätstheorie“

jedes Semester

im Sommer Schwerpunkt **Effiziente Algorithmen**

im Winter Schwerpunkt **Komplexitätstheorie**

Man kann beides hören und über beides Fachprüfungen ablegen.

Schwerpunktgebiet 4
(Algorithmen, Komplexität, Formale Modelle)

Informationen zur Vorlesung

Termine

montags 10–12 Uhr hier
donnerstags 12–14 Uhr auch hier

zentrale Quelle

<http://ls2-www.cs.uni-dortmund.de/lehre/sommer2008/ea>

dort Skript
Folien
Übungsblätter

Übungen — Organisatorisches

Übungen wöchentlich

Teilnahme extrem wichtig

Einteilung in sechs Gruppen

Termine
montags 14–16
montags 16–18
dienstags 10–12
mittwochs 10–12
mittwochs 12–14
mittwochs 16–18

Übungen — Ablauf und Konzept

grundsätzlich **Ausgabe** donnerstags in der Vorlesung
 Abgabe donnerstags in der Vorlesung

Vorsicht! Ausnahmen durch Feiertage

konkret Blatt 1 **Ausgabe** heute **Abgabe gar nicht**
 Besprechung ab 14.04.

 Blatt 2 **Ausgabe** Donnerstag **Abgabe** Donnerstag (17.04.)
 Besprechung ab 21.04.

Übungskonzept drei schriftliche Aufgaben je Blatt
 von Reproduktion bis „Knobelei“
 häufig einleitende Kurzaufgaben
 eine „Prüfungsaufgabe“

wie üblich $\geq 50\%$ aller Punkte + aktive Mitarbeit
 \rightsquigarrow „Schein“

Unterlagen

Skript

<http://ls2-www.cs.uni-dortmund.de/lehre/sommer2008/ea>

Bücher zu Algorithmen zum Beispiel

- Cormen/Leiserson/Rivest/Stein: Introduction to Algorithms
- Hochbaum (Hrsg.): Approximation Algorithms for NP-Hard Problems
- Knuth: The Art of Computer Programming
- Motwani/Raghavan: Randomized Algorithms
- Sedgewick: Algorithms

Folien jeweils nach der Vorlesung im Web

Bitte nicht (ganz) mitschreiben.

Lieber zuhören und mitdenken!

Gliederung

- Teil I **Effiziente Algorithmen**
deterministisch in Polynomialzeit

- Teil II **Approximationsalgorithmen**
im weitesten Sinn

- Teil III **Heuristiken**
sowohl problemspezifisch als auch allgemein

- Teil IV **ausgewählte Themen**
„Schönes und Interessantes“

Einordnung — Motivation

Effiziente Algorithmen

- Fortsetzung von DAP 2
Welche Algorithmen? wichtige, lehrreiche, schöne
- wesentliches Kernstück von Programmen
- lebendiges Forschungsgebiet
- Kernthema am LS 2 \rightsquigarrow Diplomarbeit

Voraussetzungen

- DAP 2
- GTI/TIfAI
- waches Interesse
- aktive Teilnahme an Vorlesung und Übungen

Tiefensuche in gerichteten Graphen

zum Aufwärmen und Einsteigen etwas **Wiederholung** von DAP 2

Tiefensuche in gerichteten Graphen

Erinnerung zentraler Algorithmus
ziemlich einfach
extrem nützlich
sehr effizient (Linearzeit)

Kantenklassifikation

Baum-Kanten (T)

Rückwärtskanten (B)

Vorwärtskanten (F)

Querkanten (C)

Algorithmus 2.1 (Tiefensuche (DFS))

1. Initialisierung

$i := 0; T := \emptyset; B := \emptyset; C := \emptyset; F := \emptyset$

Für alle $v \in V$: $\text{num}(v) := 0; \text{status}(v) := 0$

2. Für alle $v \in V$: If $\text{num}(v) = 0$ Then DFS(v)

DFS(v)

1. $\text{status}(v) := 1; i := i + 1; \text{num}(v) := i$

2. Für alle $w \in \text{Adj}(v)$

If $\text{num}(w) = 0$

Then $T := T \cup \{(v, w)\}; \text{DFS}(w)$

Else If $\text{num}(w) > \text{num}(v)$

Then $F := F \cup \{(v, w)\}$

Else If $\text{status}(w) = 1$

Then $B := B \cup \{(v, w)\}$

Else $C := C \cup \{(v, w)\}$

3. $\text{status}(v) := 2$

Tiefensuche in gerichteten Graphen: Beispiel

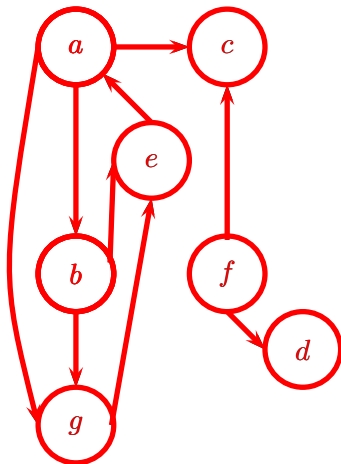
v	$\text{num}(v)$	$\text{status}(v)$
a	1	2
b	2	2
c	5	2
d	6	2
e	3	2
f	7	2
g	4	2

$$T = (a, b), (b, e), (b, g), (a, c)$$

$$F = (a, g)$$

$$B = (e, a)$$

$$C = (g, e), (f, c), (f, d)$$



Tiefensuche — Na, und?

Wen interessiert denn das?

Jemanden, der

- wissen will, ob ein Graph zusammenhängend ist.
- wissen will, ob ein Graph kreisfrei ist.
- einen DAG topologisch sortieren will.
- eine kontextfreie Grammatik in Chomsky-NF bringen will.
- wissen will, ob ein Graph stark zusammenhängend ist.

Stark zusammenhängend?

neuer Begriff starker Zusammenhang

Starker Zusammenhang

Definition 2.2 (starker Zusammenhang)

Sei $G = (V, E)$ ein gerichteter Graph. Gibt es einen gerichteten Weg von v nach w (mit $v, w \in V$), so schreiben wir $v \rightarrow w$. Gilt $v \rightarrow w$ und $w \rightarrow v$, so schreiben wir $v \leftrightarrow w$.

Zwei Knoten $v, w \in V$ heißen stark zusammenhängend, wenn $v \leftrightarrow w$ gilt.

Die Äquivalenzklassen von V bezüglich \leftrightarrow heißen *starke Zusammenhangskomponenten*.

Ist das wohldefiniert?

Ist \leftrightarrow eine Äquivalenzrelation?

Äquivalenzrelation

- reflexiv $v \leftrightarrow v$ (Weglänge 0)
- symmetrisch $v \leftrightarrow w \Leftrightarrow w \leftrightarrow v$ (nach Definition)
- transitiv $v \leftrightarrow w$ und $w \leftrightarrow x \Rightarrow v \leftrightarrow x$ (Konkatenation)

Berechnung der starken Zusammenhangskomponenten

Algorithmus 2.3

1. Führe DFS-Traversierung von $G = (V, E)$ durch, vergib dabei in absteigender Reihenfolge die f -Nummern $n, n - 1, \dots, 1$ an die Knoten. Knoten $v \in V$ erhält seine f -Nummer beim Abschluss des Aufrufs von $\text{DFS}(v)$.
2. Berechne $G^* := (V, E^*)$ mit $E^* := \{(w, v) \mid (v, w) \in E\}$.
3. Führe DFS-Traversierung von G^* durch, durchlaufe dabei im Rahmenalgorithmus die Knoten $v \in V$ in Reihenfolge aufsteigender f -Nummern, Start in dem Knoten v mit $f[v] = 1$.
4. Gib die T -Bäume der zweiten DFS-Traversierung als starke Zusammenhangskomponenten aus.

einfach zu implementieren ✓

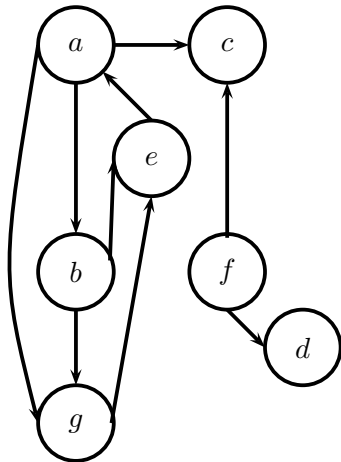
Laufzeit $O(|V| + |E|)$ ✓

Korrektheit? Sind das wirklich die SZHK?

Starke Zusammenhangskomponenten — Beispiel

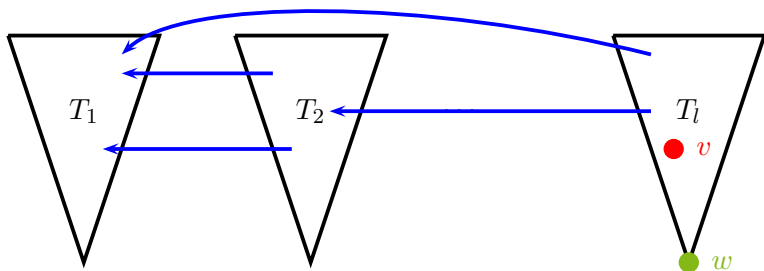
v	$\text{num}(v)$	$f(v)$
a	1	3
b	2	5
c	5	4
d	6	2
e	3	7
f	7	1
g	4	6

- SZHK
- 1: f
 - 2: d
 - 3: a, e, b, g
 - 4: c



Vorüberlegungen zur Korrektheit

DFS auf G liefert T -Bäume T_1, T_2, \dots, T_l .



Kanten zwischen den Bäumen verlaufen nur von rechts nach links.

Also ist jede SZHK vollständig in einem T -Baum.

Betrachte **Wurzel w von T_l** : w hat minimale f -Nummer: $f[w] = 1$

In G^* kann DFS(w) keinen Knoten $\notin T_l$ erreichen.

Betrachte **Knoten v mit $w \rightarrow v$ in G^*** : $v \leftrightarrow w$

Das gilt für alle Knoten im T^* -Baum mit Wurzel w .

Also wird die SZHK von w korrekt berechnet.

Korrektheitsbeweis

per Induktion über die Anzahl k der SZHK:

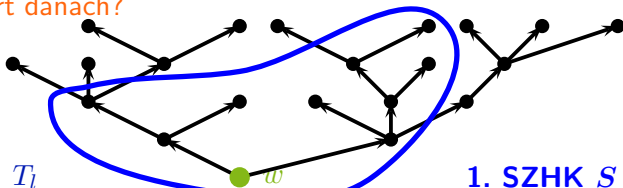
Induktionsanfang $k = 1$

wie gesehen korrekt berechnet ✓

Induktionsschritt:

wie gesehen erste SZHK korrekt berechnet

Was passiert danach?



Beobachtung: $T_1, T_2, \dots, T_{i-1}, T_i \setminus S$ entspricht einer DFS-Traversierung von $G \setminus S$ in passender Reihenfolge
 $G \setminus S$ ist ein Graph mit $k - 1$ SZHK.

Induktionsvoraussetzung ✓ \rightsquigarrow



Ein anderes Graphenproblem

Definition 3.1 (Matching)

In einem ungerichteten Graphen $G = (V, E)$ heißt eine Kantenmenge $M \subseteq E$ mit

$\forall \{u, v\}, \{w, x\} \in M: \{u, v\} \cap \{w, x\} = \emptyset$ **Matching**.

Wir suchen ein Matching maximaler Kardinalität.

wichtige Graphenklasse **bipartite Graphen**

Definition 3.3 (bipartit)

Ein ungerichteter Graph $G = (V, E)$ heißt **bipartit**, wenn es eine disjunkte Knotenzerlegung von $V = V_1 \dot{\cup} V_2$ gibt, so dass keine Kante ganz in V_1 oder V_2 verläuft.

Matchings in bipartiten Graphen

Wie berechnet man maximale Matchings in bipartiten Graphen?

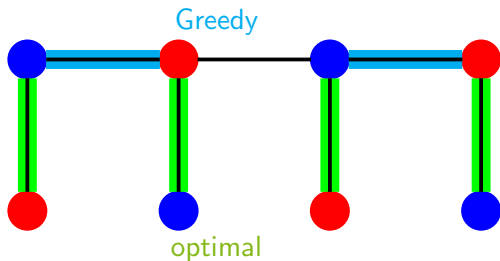
Idee **greedy**

Starte mit \emptyset und füge so lange eine Kante hinzu,
bis keine Kante mehr hinzufügar ist.

Algorithmus 3.4 (Greed Matching-Algorithmus)

1. $M := \emptyset$
2. If $\exists e \in E: M \cup e$ ist Matching
Then $M := M \cup e$; Weiter bei 2.
3. Ausgabe M

Matchings in bipartiten Graphen



Theorem 3.5

Sei $G = (V, E)$ ein ungerichteter Graph, M_{opt} ein maximales Matching in G .

Der Greedyalgorithmus berechnet ein Matching M_{greedy} mit $|M_{\text{greedy}}| \geq |M_{\text{opt}}|/2$.

Sogar für bipartite Graphen ist $|M_{\text{greedy}}| = |M_{\text{opt}}|/2$ möglich. ✓

Beweis der Approximationsgüte

Beweis.

V_{greedy} : Menge der zu M_{greedy} inzidenten Knoten

klar $|V_{\text{greedy}}| = 2 |M_{\text{greedy}}|$

Jede Kante $e \in E$, hat einen Knoten in V_{greedy} , sonst wäre e gewählt worden.

Das gilt also auch für jede Kante aus M_{opt} .

also $|M_{\text{opt}}| \leq |V_{\text{greedy}}| = 2 |M_{\text{greedy}}|$

also $|M_{\text{greedy}}| \geq |M_{\text{opt}}| / 2$ □

klar Wir wollen maximale Matchings.

Wir setzen voraus Graph zusammenhängend
(also $n - 1 \leq |E| \leq \binom{n}{2}$)

sonst unabhängig auf Zusammenhangskomponenten

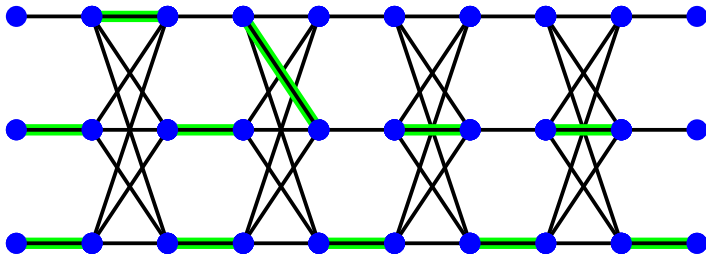
Zentrale Begriffe

Definition 3.6 (Matching-Begriffe)

$G = (V, E)$ ungerichteter Graph, $M \subseteq E$ Matching auf G .

- $e \in M$ heißt **Matching-Kante**.
- $e \notin M$ heißt **freie Kante**.
- $v \in e$ mit Matching-Kante e heißen **besetzt**.
- $v \in V$ nicht besetzt heißt **frei**.
- $P = (v_1, v_2, \dots, v_k)$ mit $\{v_{2i-1}, v_{2i}\}$ freie und $\{v_{2i}, v_{2i+1}\}$ Matching-Kante heißt **M -alternierend**.
- Einfacher M -alternierender Pfad $P = (v_1, \dots, v_k)$ mit v_1 und v_k frei heißt **M -verbessernd**.

Beispiel zu Matching-Begriffen



M -verbessernde Pfade

Theorem 3.7

M maximal \Leftrightarrow Es gibt keinen M -verbessernden Pfad.

Beweis.

Wir beweisen

M nicht maximal \Leftrightarrow Es gibt M -verbessernden Pfad.

„ \Leftarrow “

Sei $P = (v_1, \dots, v_k)$ M -verbessernder Pfad.

Beobachtung k gerade (also $k = 2j$ mit $j \in \mathbb{N}$)

Mache M -Kanten $\{v_2, v_3\}, \{v_4, v_5\}, \dots, \{v_{2j-2}, v_{2j-1}\}$ zu freien Kanten.

Mache freie Kanten $\{v_1, v_2\}, \{v_3, v_4\}, \dots, \{v_{2j-1}, v_{2j}\}$ zu M -Kanten.

Beobachtung Das ist möglich.

Beobachtung M wächst dadurch um 1.

Beweis der Gegenrichtung

„ M nicht maximal $\Rightarrow \exists M$ -verbessernden Pfad“

Sei M nicht maximal.

$\exists M' : |M'| > |M|$

Betrachte $M \oplus M'$

$= \{e \mid e \in M \wedge e \notin M'\} \cup \{e \mid e \notin M \wedge e \in M'\}$

Beobachtung in $M \oplus M'$ alle Knotengrade ≤ 2

also $M \oplus M'$ zerfällt in disjunkte Pfade und Kreise

immer M -Kante und M' -Kante abwechselnd

also alle Kreise haben gerade Länge

$|M'| > |M| \Rightarrow \exists$ Pfad P mit mehr M' - als M -Kanten

P ist M -verbessernd



Anwendung einfacher Matching-Algorithmus

Maximale Matchings in bipartiten Graphen

Erinnerung $G = (V, E)$ **bipartit**
 $\Leftrightarrow \exists V_1, V_2: (V = V_1 \dot{\cup} V_2) \wedge (V_1^2 \cap E = \emptyset) \wedge (V_2^2 \cap E = \emptyset)$

Algorithmus

1. $M := \emptyset$
2. Berechne M -verbessernden Pfad P .
3. If kein P gefunden,
Then Exit mit Ausgabe M .
4. $M := M \oplus P$
5. Weiter bei 2.

klar endlich, weil $\leq n/2$ Iterationen

also Korrektheit klar ✓

Aber auch effizient?

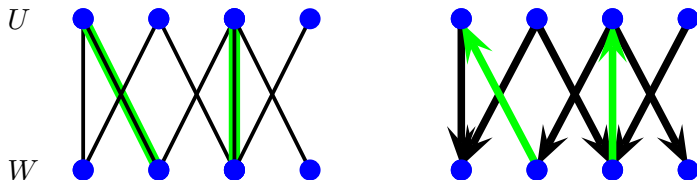
M -verbessernde Pfade finden

Sei $G = (U \cup W, E)$ bipartiter Graph, M Matching auf G .

Wir „richten“ G :

$G_M = (U \cup W, E_M)$ gerichteter Graph mit

- $(u, w) \in E_M$ für $\{u, w\} \in E \setminus M, u \in U, w \in W$
- $(w, u) \in E_M$ für $\{u, w\} \in E \cap M, u \in U, w \in W$



Beobachtungen

- jeder gerichtete Weg in G_M ist M -alternierender Pfad in G
- jeder gerichtete Weg in G_M von freiem U -Knoten zu freiem W -Knoten ist M -verbessernder Pfad in G

Ausformulierter Algorithmus

Algorithmus 3.8 (einfacher Matching-Algorithmus)

1. $M := \emptyset$
2. Konstruiere gerichteten Graphen $G' = (U \cup W, E')$ mit
 $E' = \{(u, w) \mid u \in U, w \in W, \{u, w\} \in M\}$
 $\cup \{(w, u) \mid u \in U, w \in W, \{u, w\} \notin M\}$.
3. Suche mit Breitensuche beginnend in allen freien Knoten
 einen gerichteten Pfad zu einem freien Knoten.
 Sei P dieser Pfad.
4. If P gefunden
 Then $M := M \oplus P$; Weiter bei 2.
 Else Ausgabe M .

Über Algorithmus 3.8

Theorem 3.9

Algorithmus 3.8 berechnet in Zeit $O(ne) = O(n^3)$ ein maximales Matching für einen bipartiten Graphen $G = (U \dot{\cup} W, E)$ mit $|U \dot{\cup} W| = n$ und $|E| = e$.

Beweis.

Laufzeit $\leq n/2$ Breitensuche ✓

Korrektheit gerichtete Pfade „frei \rightsquigarrow frei“
 $\Leftrightarrow M$ -verbessernde Pfade

klar Breitensuche findet solche Pfade



Geht es nicht schneller?