Vorlesung 3:

Graphenalgorithmen

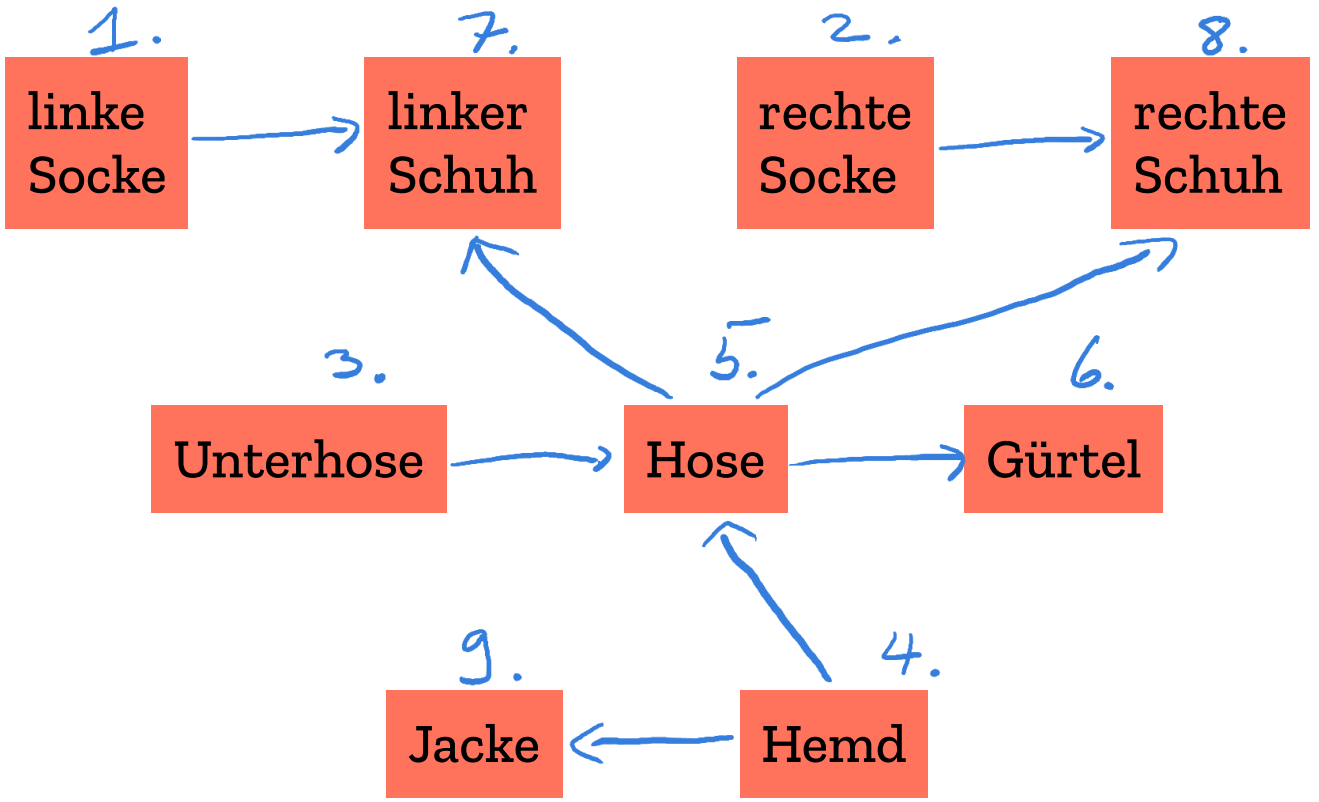
Markus Püschel
David Steurer
Peter Widmayer

PDF download
goo.gl/yM3SPq

Algorithmen und Datenstrukturen, Herbstsemester 2017, ETH Zürich

Gerichtete Graphen und Abhängigkeiten

Beispiel: Abhängigkeiten beim Anziehen von Kleiderstücken

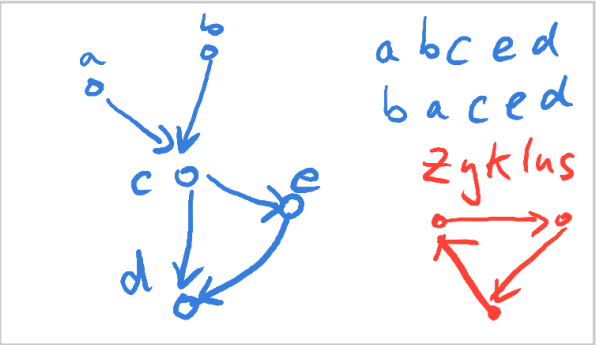


Gesucht: *Reihenfolge, die alle Abhängigkeiten berücksichtigt*

Topologische Sortierung

Definition: eine Folge $v_1, \ldots, v_n$ von Knoten ist eine *topologische Sortierung* von $G = (V, E)$ falls für jede Kante $(v_i, v_j) \in E$ gilt, dass $i < j$, und $V = \{v_1, \ldots, v_n\}$

Gibt es immer eine solche Sortierung?



Theorem: Ein Graph hat eine topologische Sortierung genau dann wenn er keinen Zyklus enthält (*«azyklisch»*)

Zu zeigen: für einen azyklischen Graphen können wir immer eine topologische Sortierung finden

Lemma: Ein azyklischer Graph enthält immer eine *Senke* (d.h., Ausgangsgrad 0)

Beweis: sei $G = (V, E)$ ein azyklischer Graph

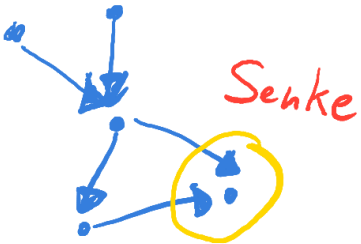
Behauptung: jeder Weg in G hat Länge $\leqslant n$

ein längerer Weg würde mindestens einen Knoten mehrfach besuchen (*Schubfachprinzip*) und damit einen Zyklus enthalten

sei $v_1, \ldots, v_k$ ein Weg maximaler Länge

(wohldefiniert wegen Behauptung)

dann ist v_k eine Senke (ansonsten würde jeder Nachfolger von v_k einen längeren Weg geben)



Theorem: *(wiederholt)* Jeder azyklische Graph hat eine topologische Sortierung

Beweis: Induktion über Knotenanzahl $n \geqslant 1$

Induktionsanfang: «$A(1)$ gilt» ✔

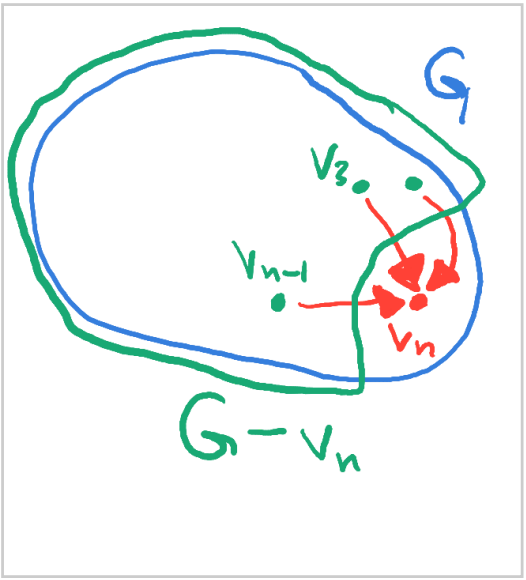
Induktionsschritt: «$A(n-1) \to A(n)$ gilt für alle $n \geqslant 2$»

sei v_n eine Senke (existiert laut Lemma)

laut *Induktionshypothese* hat G ohne v_n eine topologische Sortierung $v_1, \ldots, v_{n-1}$

dann ist $v_1, \ldots, v_{n-1}, v_n$ eine topologische Sortierung von G

q.e.d.

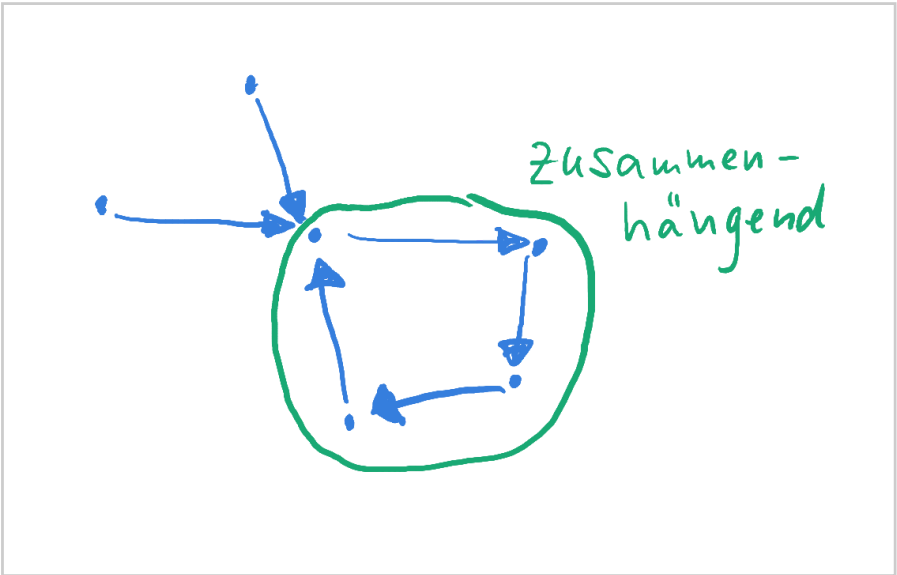


Zusammenhang

sei $G = (V, E)$ ein Graph (gerichtet oder ungerichtet)

Definition: eine Knotenmenge $W \subseteq V$ ist (stark) *zusammenhängend* in G, falls für alle Knoten $u, v \in W$ gilt, dass v von u erreichbar ist (d.h. $\exists$ Weg von u nach v)

Begriff: $G = (V, E)$ ist *zusammenhängend* falls V in G zusammenhängt



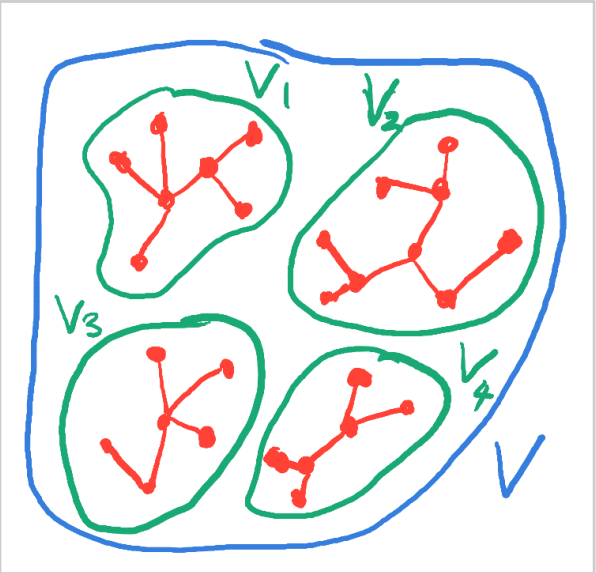
Zusammenhangskomponenten

Theorem: für jeden ungerichteten Graphen $G = (V, E)$ gibt es eine Partition $\{V_1, \ldots, V_k\}$ von V, so dass $V_1, \ldots, V_k$ jeweils zusammenhängend sind und jede Kante von G in genau einem Teil V_i verläuft

Begriff: die Teile $V_1, \ldots, V_k$ heissen *Zusammenhangskomponenten*

Ausblick in diskrete Mathematik: Zusammenhang in ungerichteten Graph ist eine *Äquivalenzrelation*

Ausblick: bei gerichteten Graphen, kann die Zusammenhangsstruktur komplizierter sein

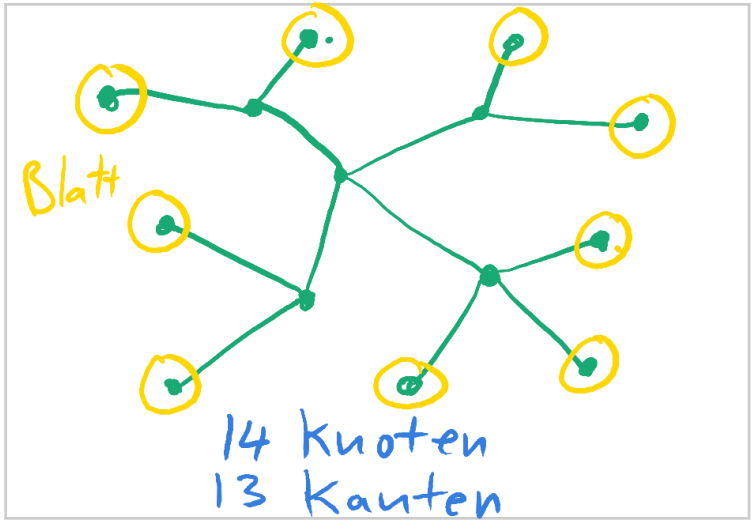


Bäume

Definition: ein *Baum* ist ein zusammenhängender ungerichteter Graph ohne Kreis

Begriffe: ein *Wald* ist ein ungerichteter Graph ohne Kreis; die Zusammenhangskomponenten eines Walds sind Bäume

ein *Blatt* ist ein Knoten mit Grad 1



Eigenschaften: jeder Baum mit $n \geqslant 2$ Knoten hat mindestens ein Blatt; jeder Baum hat genau $n - 1$ Kanten

jeder zusammenhängende Graph enthält mindestens einen Baum mit derselben Knotenmenge (genannt *Spannbaum*)

Effizienzbegriff von Algorithmen

es gibt *effiziente* Algorithmen (z.B. Gale–Shapley) und *ineffiziente* Algorithmen (z.B. alle möglichen Färbungen eines Graphen durchprobieren)

Wie können wir die Effizienz von Algorithmen formalisieren?

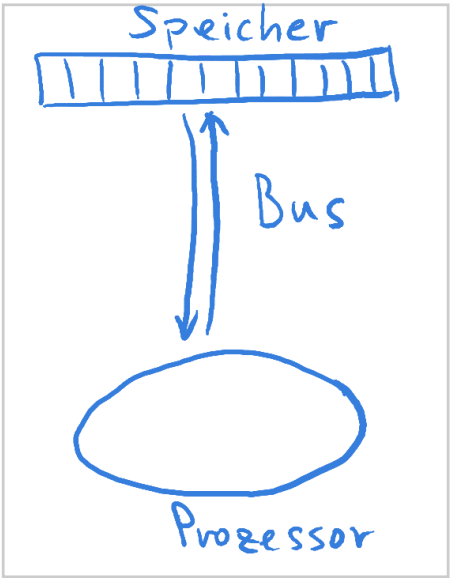
die genaue Laufzeit eines Computerprogramms hängt von vielen Details ab (z.B. Computerarchitektur und Compiler-Optimierung)

um effiziente und ineffiziente Algorithmen zu unterscheiden, stellt sich heraus, dass diese Details unwichtig sind

daher betrachten wir ein vereinfachtes *Berechnungsmodell*, um diese Details zu abstrahieren

Berechnungsmodell

Komponenten



- *Speicher:* a-priori unbeschränkt viele, adressierbare Speicherzellen
- *Prozessor:* führt *elementare Operationen* aus wie Rechenoperationen (Addition, Subtraktion, Multiplikation, Division), Vergleichsoperationen (=,<,>), Lese- und Schreibzugriffe auf Speicher
- *Bus:* verbindet Prozessor und Speicher

Anmerkung: eine Speicherzelle kann verschiedene Daten enthalten, z.B. ein Bit (0 oder 1), eine Zahl (nicht viel grösser als Zahlen in der Eingabe), konstante Anzahl von Bits oder Zahlen (unabhängig von «Grösse» der Eingabe)

Laufzeit

Begriff: *Laufzeit* eines Algorithmus für eine Eingabe ist die Anzahl der elementaren Operationen, die der Prozessor während der Berechnung ausführt

die Laufzeit könnte von vielen Details der Eingabe abhängen

daher beschränken wir die Laufzeit als Funktion der «Grösse» der Eingabe (mögliche Begriffe der «Grösse» sind problemabhängig; z.B. die Anzahl der Frauen und Männer bei stabilen Zuordnungen)

Definition: ein Algorithmus A hat *Laufzeit* f falls $f(n) =$ maximale Laufzeit von A über alle Eingaben der Grösse n

bei manchen Problemen hängt die «Grösse» von mehreren Parametern ab (z.B. Knotenzahl und Kantenzahl bei Graphenproblemen)

dann ist die Laufzeit eine Funktion dieser Parameter

Asymptotische Notation

Überlegung: aufgrund unseres vereinfachten Berechnungsmodell macht es wenig Sinn zwischen den Laufzeiten $10 \cdot n$ und $10000 \cdot n$ zu unterscheiden (da die tatsächlichen Kosten verschiedener elementarer Operationen sich um teils grosse konstante Faktoren unterscheiden)

daher ignorieren wir *konstante Faktoren* (unabhängig von Eingabegrösse)

Schreibweise: $f(n) \leqslant O(g(n))$ falls es eine Konstante $C > 0$ (unabhängig von n!) gibt, so dass $f(n) \leqslant C \cdot g(n)$ für alle $n \geqslant 1$ gilt

Beispiele: $10000 \cdot n \leqslant O(n)$, $\quad 1000000 \cdot n + n^2 \leqslant O(n^2)$,
$n^2 \not\leqslant O(10000 \cdot n)$, $\quad n^2 \leqslant O(n^3)$

- Gale–Shapley Algorithmus hat *Laufzeit* $\leqslant O(n^2)$ für Eingaben mit n Frauen und Männer
- naiver Algorithmus zur Färbung mit k Farben hat *Laufzeit* $\leqslant O(k^n \cdot (n+m))$ für Graphen mit n Knoten und m Kanten

Asymptotische Notation—Übung

«$f(n) \leqslant O(g(n))$» steht für folgende Aussage über zwei Funktionen f und g:

$$\exists C > 0.\ \forall n \geqslant 1.\ f(n) \leqslant C \cdot g(n)$$

formalisiert: f kleiner gleich g bis auf konstanten Faktor

Gilt $100 \cdot n + n^2 \leqslant O(n^2)$?

ja, Aussage "$\exists C > 0.\ \forall n \geq 1.\ 100 \cdot n + n^2 \leq C \cdot n^2$" gilt.

Beweis: Wähle $C = 101$. Dann gilt:

$100n + n^2 \leq 100n \cdot \underbrace{n}_{\geq 1} + n^2 = C \cdot n^2$ ✓

Gilt $n^2 \leqslant O(1000 \cdot n)$? Nein, Aussage "$\exists C > 0\ \forall n \geq 1.\ n^2 \leq C \cdot 1000n$" ist falsch!

Beweis: Für jedes $C > 0$, gibt es ein $n \geq 1$, nämlich $n = C \cdot 1000 + 1$, sodass $n^2 > C \cdot 1000 \cdot n$

Representationen von Graphen (im Computerspeicher)

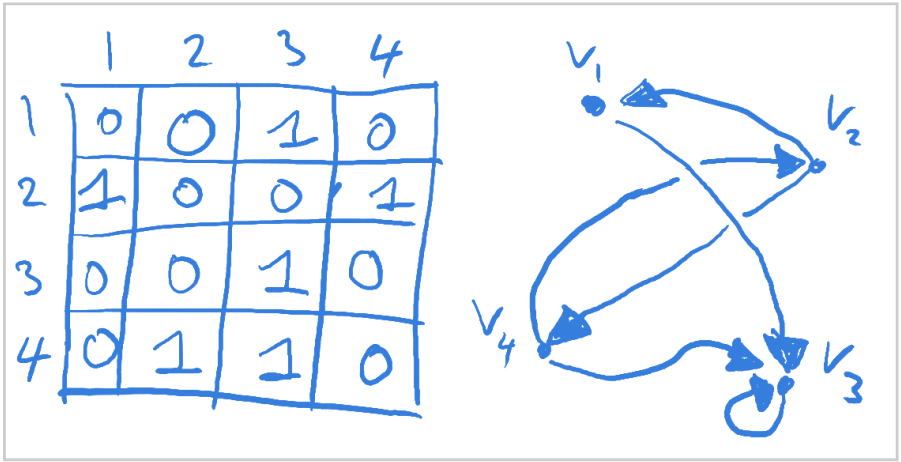
gerichteter Graph $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$

Definition: die *Adjazenzmatrix* von G ist eine Tabelle A mit n Zeilen und n Spalten, so dass für den Eintrag A_{ij} in Zeile i und Spalte j gilt

$$A_{ij} = \begin{cases} 0 & (v_i, v_j) \notin E \\ 1 & (v_i, v_j) \in E \end{cases}$$

Laufzeit $O(1)$ um zu testen ob G bestimmte Kante (v_i, v_j) enthält

Laufzeit $O(n)$ um alle Nachfolger eines Knoten v_i aufzuzählen



Representationen von Graphen (im Computerspeicher)

gerichteter Graph $G = (V, E)$ mit $V = \{v_1, \ldots, v_n\}$

Definition: die *Adjazenzliste* von G ist eine Tabelle A mit n Einträgen, so dass Eintrag A_i eine Liste aller Nachfolger von v_i enthält

Laufzeit $O(\deg^+(v_i) + 1)$ um zu testen ob G bestimmte Kante (v_i, v_j) enthält

Laufzeit $O(\deg^+(v_i) + 1)$ um alle Nachfolger eines Knoten v_i aufzuzählen