Datenstrukturen & Algorithmen

Peppo Brambilla
Universität Bern
Frühling 2018

Übersicht

Graphenalgorithmen

- Begriffe & Darstellung von Graphen
- Breitensuche & Tiefensuche
- Topologisches Sortieren
- Starke Zusammenhangskomponenten

Graphen

- Graphen bestehen aus Knoten und Kanten zwischen Paaren von Knoten
- Graphen modellieren Beziehungen zwischen Daten
- Beispiele
  - Knoten: speichern Schlüsselwerte, Kanten: „Kante von Knoten a nach b existiert, wenn $a.key < b.key$"
  - Knoten: geografische Orte, Kanten: Distanz, falls Zugstrecke existiert

Graphenprobleme

- Kürzester Pfad von Knoten A nach B (shortest path problem)
- Kürzester Pfad, der alle Knoten besucht und einen Knoten als Start- und Endpunkt hat (travelling salesman problem)
- Finde Zusammenhangskomponenten, d.h. Untermengen von Knoten die gegenseitig miteinander verbunden sind
- Viele mehr, z.B. Grundlagen für Optimierungsalgorithmen

Darstellung von Graphen

- Graph $G = (V, E)$
  - Menge von Knoten V
  - Menge von Kanten E
- Anzahl Knoten $|V|$
- Anzahl Kanten $|E|$
- Laufzeit ausgedrückt in $|V|$ und $|E|$
  - Vereinfachte Notation, Beispiel $O(V + E)$

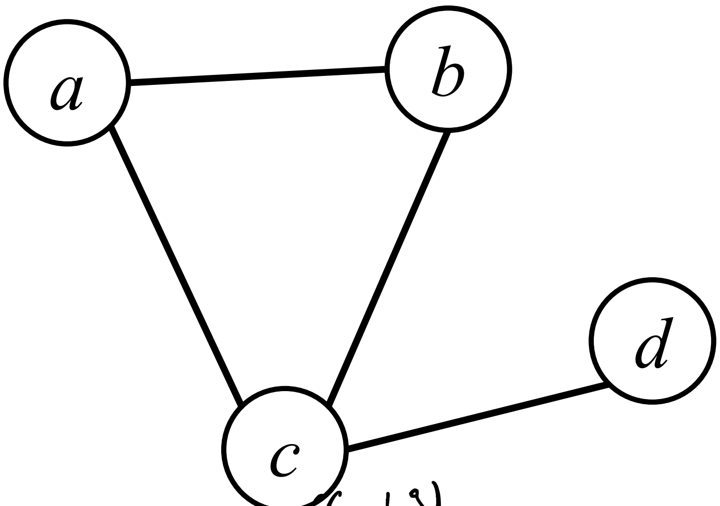
Darstellung von Graphen

Begriffe

- Ungerichtete Graphen: Kanten haben keine Orientierung
- Gerichtete Graphen: Kanten haben je eine von zwei möglichen Orientierungen
- Gewichtete Graphen: Jede Kante hat reelle Zahl als Gewicht
- Grad eines Knoten
  - Anzahl Kanten, die diesen Knoten enthalten
  - Gerichtete Graphen: Unterscheiden Eingangsgrad und Ausgangsgrad

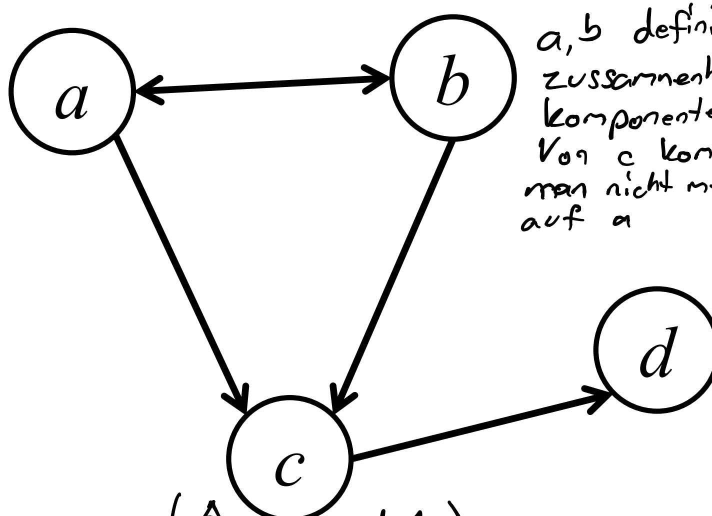
Darstellung von Graphen

- Knoten $V = \{a, b, c, d\}$
- Kanten
  - Ungerichtet: beide Richtungen der Kante implizit mitgemeint $E = \{(a,b), (a,c), (b,c), (c,d)\}$
  - Gerichtet $E = \{(a,b), (b,a), (a,c), (b,c), (c,d)\}$



(Grad 3)

ungerichtet



a, b definieren zusammenhangs-komponente. Von c kommt man nicht mehr auf a

(Ausgangsgrad 1
Eingangsgrad 2)

gerichtet

Begriffe

- Pfad: Sequenz von Knoten die über Kanten entsprechender Richtung verbunden sind
- Erreichbar: Knoten ist von einem anderen erreichbar, wenn es einen Pfad gibt, der beide Knoten verbindet. Notation: $u \to v$
- Zyklus: Pfad, mit gleichem Anfangs- und End-Knoten
- Zusammenhangskomponenten: Äquivalenzklassen der Knoten in ungerichteten Graphen bezüglich der Relation „erreichbar von“
- Starke Zusammenhangskomponenten: Äquivalenzklassen der Knoten in gerichteten Graphen bezüglich der Relation „gegenseitig erreichbar von“

Darstellung von Graphen

Adjazenzlisten

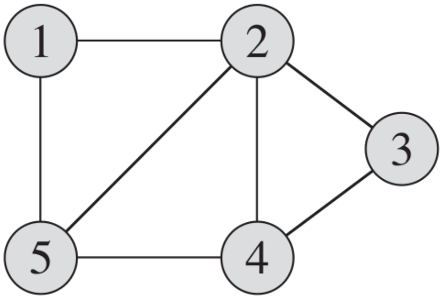
- Feld Adj der Grösse $|V|$ von Listen
  - Eine Liste pro Knoten
- Liste von Knoten u enthält alle Knoten v so dass $(u, v) \in E$
- Funktioniert für gerichtete und ungerichtete Graphen
- Speicher: $O(V + E)$
- Zeit um zu bestimmen ob $(u, v) \in E$: $O\underbrace{(Grad(u))}_{\text{worst-case: Ganze Liste durchsuchen}}$

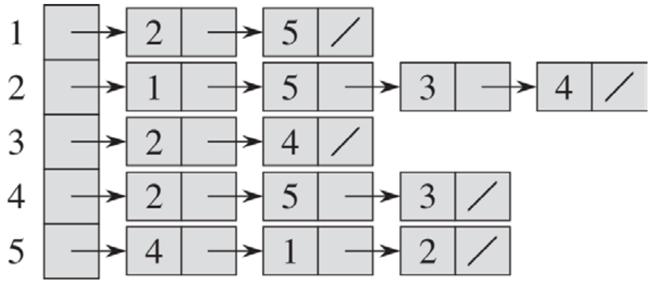
Darstellung von Graphen

Adjazenzmatrix

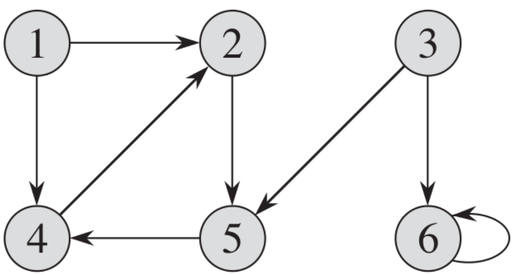
- Matrix A der Grösse $|V| \times |V|$
- Elemente $a_{ij} = \begin{cases} 1 & \text{wenn } (i,j) \in E, \\ 0 & \text{sonst.} \end{cases}$
- Speicher: $O(V^2)$
- Zeit, um zu bestimmen ob $(u,v) \in E$: konstant
- Gewichtete Graphen: Gewichte direkt in Matrix speichern

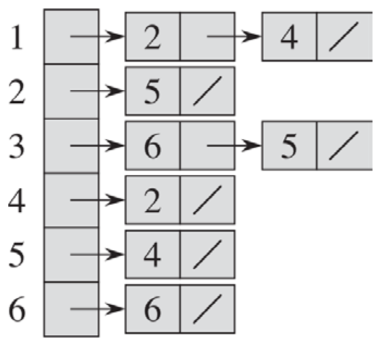
Darstellung von Graphen





	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0





	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Darstellung von Graphen

- Adjazenz**listen** vorteilhaft für dünn besetzte Graphen
  - $|E|$ viel kleiner als $|V^2|$
- Adjazenz**matrix** vorteilhaft für dichte Graphen
  - Jeder Knoten mit fast jedem verbunden
  - $|E|$ nahe an $|V^2|$
- Adjazenzmatrix symmetrisch für ungerichtete Graphen
  - $a_{ij} = a_{ji}$, müssen nur obere oder untere Hälfte der Matrix speichern

Übersicht

Graphenalgorithmen

- Begriffe & Darstellung von Graphen
- Breitensuche & Tiefensuche
- Topologisches Sortieren
- Starke Zusammenhangskomponenten

Breitensuche

- Findet alle Knoten, die von einem Startknoten aus erreichbar sind, d.h. durch eine Serie von Kanten verbunden sind.
- Berechnet zusätzlich Abstand (kleinste Anzahl Kanten), um jeden Knoten zu erreichen
  - Knoten werden in ansteigendem Abstand gefunden
- (Einige Details aus Buch weggelassen hier)

Breitensuche

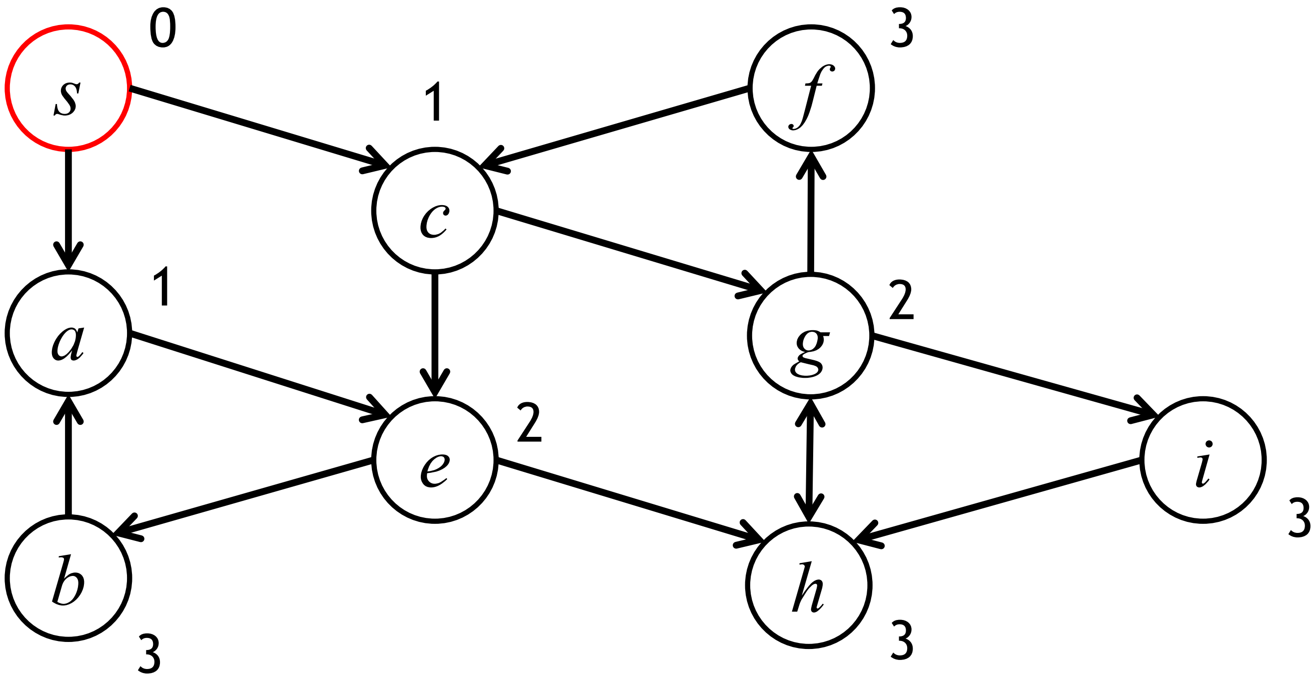
- Eingabe: Graph $G = (V, E)$, gerichtet oder ungerichtet, Startknoten s
- Berechne: Distanz $v.d$ von s nach v, für alle v
- Idee: Sende Welle ausgehend von s
  - Trifft zuerst alle Knoten erreichbar über eine Kante von s
  - Von da aus alle Knoten erreichbar über zwei Kanten von s
  - etc.
- Wellenfront in FIFO Warteschlange Q verwaltet
  - Knoten v in Q falls Welle v erreicht hat, aber noch nicht von v weitergeleitet wurde

Breitensuche

BFS(G, s)

for each vertex $u \in G.V \setminus \{s\}$
    $u.d = \infty$ // Abstand ∞ für alle Knoten
$s.d = 0$
$Q = \emptyset$ // Leere Schlange
ENQUEUE(Q, s)
while $Q \neq \emptyset$ // Ab hier aussenden der Welle
    $u =$ DEQUEUE(Q)
    for each $v \in G.Adj[u]$
        if $v.d == \infty$
            $v.d = u.d + 1$
            ENQUEUE(Q, v)

Breitensuche



FIFO Warteschlange: $s, a, c, e, g, b, h, f, i$

v.d

Q: ~~s~~ a c

$\boxed{\text{Q: } \cancel{s}\ \cancel{a}\ c}$ usw. ~~s~~ ~~a~~ ~~c~~ ~~e~~ g f h i

s	0	0	
a	∞	1	
b	∞	∞	
c	∞	∞	1
e	∞	∞	
f	∞	∞	
g	∞	∞	
h	∞	∞	
i	∞	∞	

s	0
a	1
b	3
c	1
e	2
f	∞
g	2
h	3
i	

Breitensuche

- Zeitkomplexität $O(V + E)$
  - $O(V)$ weil jeder Knoten höchstens einmal in FIFO eingetragen wird
  - $O(E)$ weil jeder Knoten höchstens einmal aus FIFO genommen wird, und Kanten nur untersucht werden, wenn Knoten aus Queue genommen wird → jede Kante wird einmal (gerichtet) oder zweimal (ungerichtet) untersucht
- Anmerkung: weitere Konzepte zur Breitensuche (Vorgängerteilgraph, Breitensuchbaum) aus Buch gehören auch zum Prüfungsstoff

Tiefensuche

- Suche geht „tiefer“ in Graphen wenn immer möglich
  - Untersuche die Kanten als nächstes, die vom zuletzt entdeckten Knoten ausgehen, der noch ungeprüfte Kanten hat
- Gefundene Knoten werden mit Zeitstempel versehen
  - Zeit $v.d$, wann Knoten v zuerst gefunden
  - Zeit $v.f$, wann letzte Kante von v geprüft

Tiefensuche

- Eingabe: Graph $G = (V, E)$, gerichtet oder ungerichtet, kein Startknoten
- Berechne: 2 Zeitstempel für jeden Knoten v
  - Entdeckungszeit $v.d$
  - Prüfung aller Kanten beendet $v.f$
- Suchstrategie: sobald Knoten entdeckt, folge seinen Kanten
- Farbe $v.color$ zeigt Zustand des Knotens
  - WHITE: noch nicht entdeckt
  - GRAY: entdeckt, aber noch nicht fertig (hat Kanten, die noch nicht verfolgt wurden)
  - BLACK: fertig (alle Kanten verfolgt)

Tiefensuche

```
DFS(G)

for each vertex u ∈ G.V
        u.color = WHITE
        u.π = NIL          // Knoten über welcher u entdeckt wurde = NIL
time = 0
for each vertex u ∈ G.V
        if u.color == WHITE
                DFS-VISIT(G, u)
```

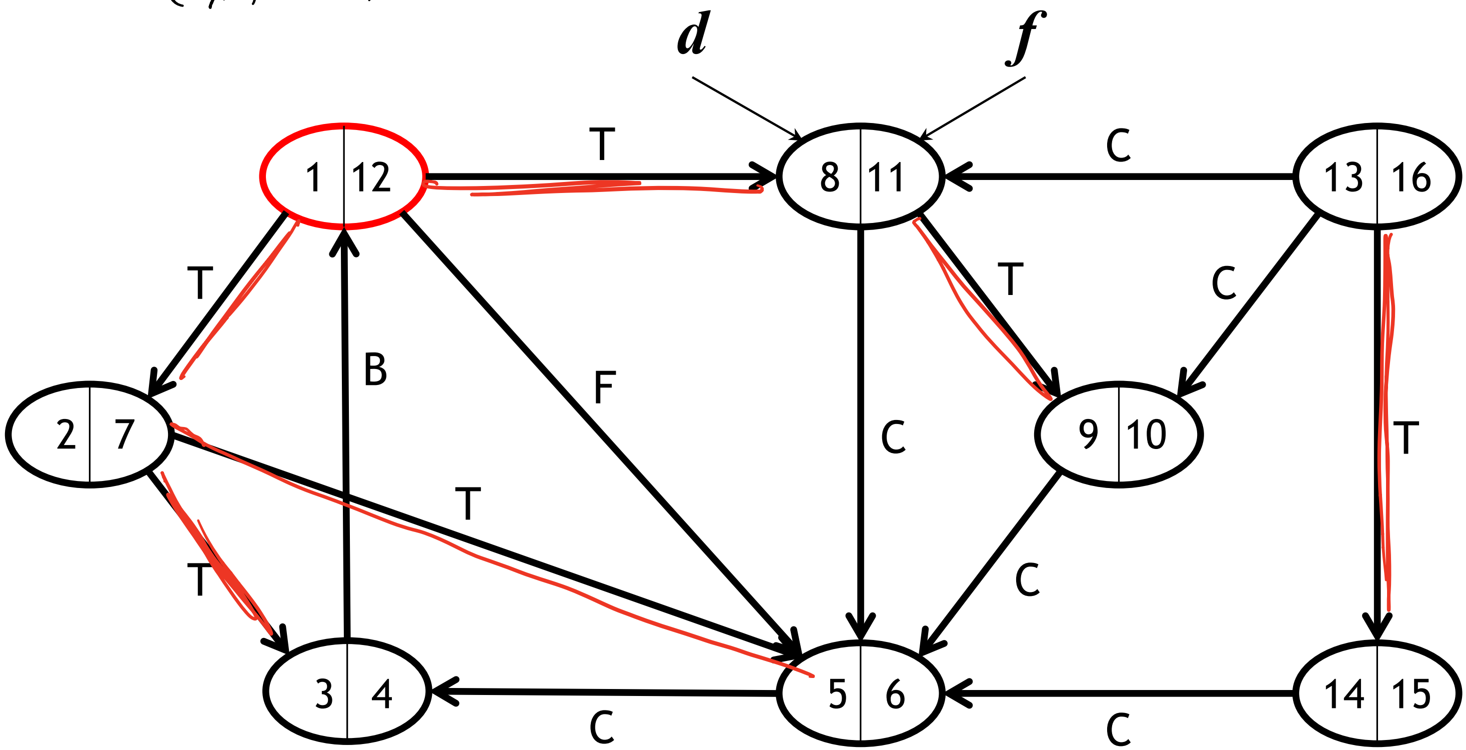
```
DFS-VISIT(G, u)

time = time + 1                 // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]           // explore edge (u, v)
        if v.color == WHITE
                v.π = u
                DFS-VISIT(G, v)
u.color = BLACK                 // blacken u; it is finished
time = time + 1
u.f = time
```

Tiefensuche

$v.d \leq u.f$

(1,(2,(3,4) usw.



T: Baumkante (tree) (neu Entdeckung)

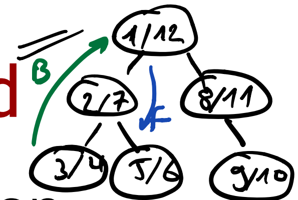
B: Rückwärtskante (back)
(treffen auf Grau)

C: Querkante (cross) (alles andere)

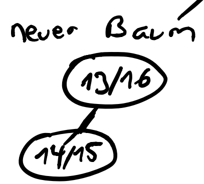
F: Vorwärtskante (forward)
(treffen auf Knoten, welche schon abgearbeitet wurden)

Tiefensuche

- Zeitkomplexität
  - Analyse ähnlich wie Breitensuche
  - $\Theta(V+E)$ statt $O(V+E)$, weil jeder Knoten und jede Kante besucht werden
- Tiefensuche führt zu Tiefensuchwald



  - Besteht aus mehreren Tiefensuchbäumen
  - Bäume bestehen aus Kanten (u, v), so dass $u.color = \text{GRAY}$ und $v.color = \text{WHITE}$ waren, als (u, v) gefunden wurde
  - Details im Buch, Kapitel 22.3



Macht Unterschied ob ich bei 13/16 anfange oder bei 1/12. Bei 13/16 entsteht nur 1 Baum.

Eigenschaften der Tiefensuche

- Klammerungstheorem
  Es gibt genau 3 Möglichkeiten für Beziehung zwischen 2 Intervallen $[u.d, u.f], [v.d, v.f]$:
  1. Intervalle $[u.d, u.f]$ und $[v.d, v, f]$ sind paarweise disjunkt.
  2. Intervall $[u.d, u, f]$ ist vollständig in $[v.d, v.f]$ enthalten.
  3. Intervall $[v.d, v.f]$ ist vollständig in $[u.d, u.f]$ enthalten.
- Wie Klammern
  - Ok: ()[] oder ([]) oder [()]
  - Kommt nicht vor: ([)] oder [(])

Eigenschaften der Tiefensuche

- Intervalle der Nachfahren: Knoten v ist Nachfahre von u im Tiefensuchwald, genau wenn $u.d < v.d < v.f < u.f$
- Theorem der weissen Pfade: Knoten v ist Nachfahre von u im Tiefensuchwald, genau wenn Knoten v zur Zeit $u.d$ von u aus entlang eines Pfades erreichbar ist, der nur weisse Knoten enthält

Eigenschaften der Tiefensuche

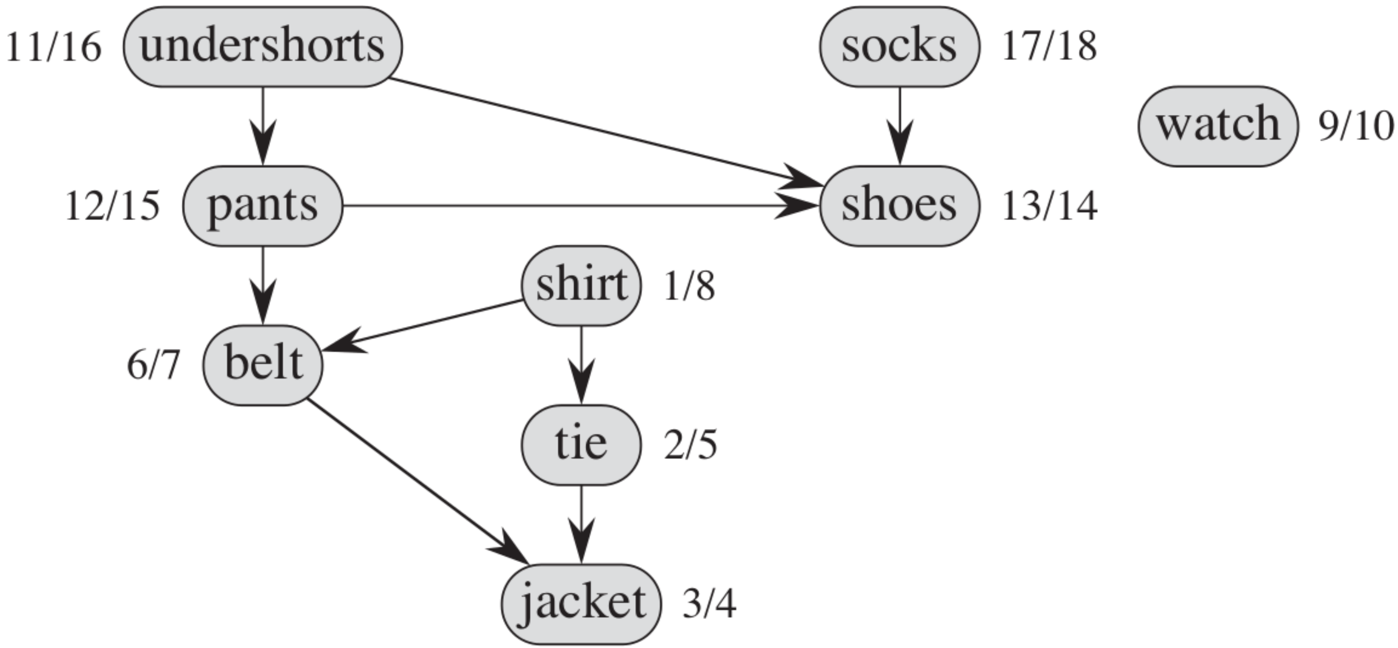
- Klassifikation der Kanten
  - Baumkanten T: Kante, die im Tiefensuchwald vorkommt
  - Rückwärtskante B: (u, v), wobei u ein Nachfolger von v im Tiefensuchbaum ist
  - Vorwärtskante F: (u, v), wobei u ein Vorgänger von v im Tiefensuchbaum ist
  - Querkante C: alle anderen
- Theorem: Bei der Tiefensuche in einem ungerichteten Graphen ist jede Kante entweder Baumkante oder Rückwärtskante

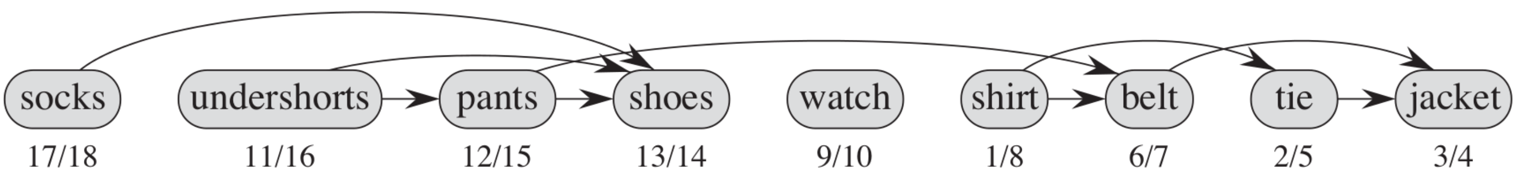
Übersicht

Graphenalgorithmen

- Begriffe & Darstellung von Graphen
- Breitensuche & Tiefensuche (in Prüfung: Bei guten Knoten anfangen)
- Topologisches Sortieren
- Starke Zusammenhangskomponenten

Topologisches Sortieren





Topologisches Sortieren

- Für gerichtete azyklische Graphen (directed acyclic graph, DAG)
- Nützlich zur Darstellung von Abläufen mit partieller Ordnung
  - Transitiv: $a \leq b \leq c \rightarrow a \leq c$
  - Reflexiv: $a \leq a$
  - Antisymmetrisch: $a \leq b, b \leq a \rightarrow a = b$
  - Aber nicht zwingend $a \leq b$ oder $b \leq a$, d.h. Ordnungsrelation kann für gewisse Paare undefiniert sein

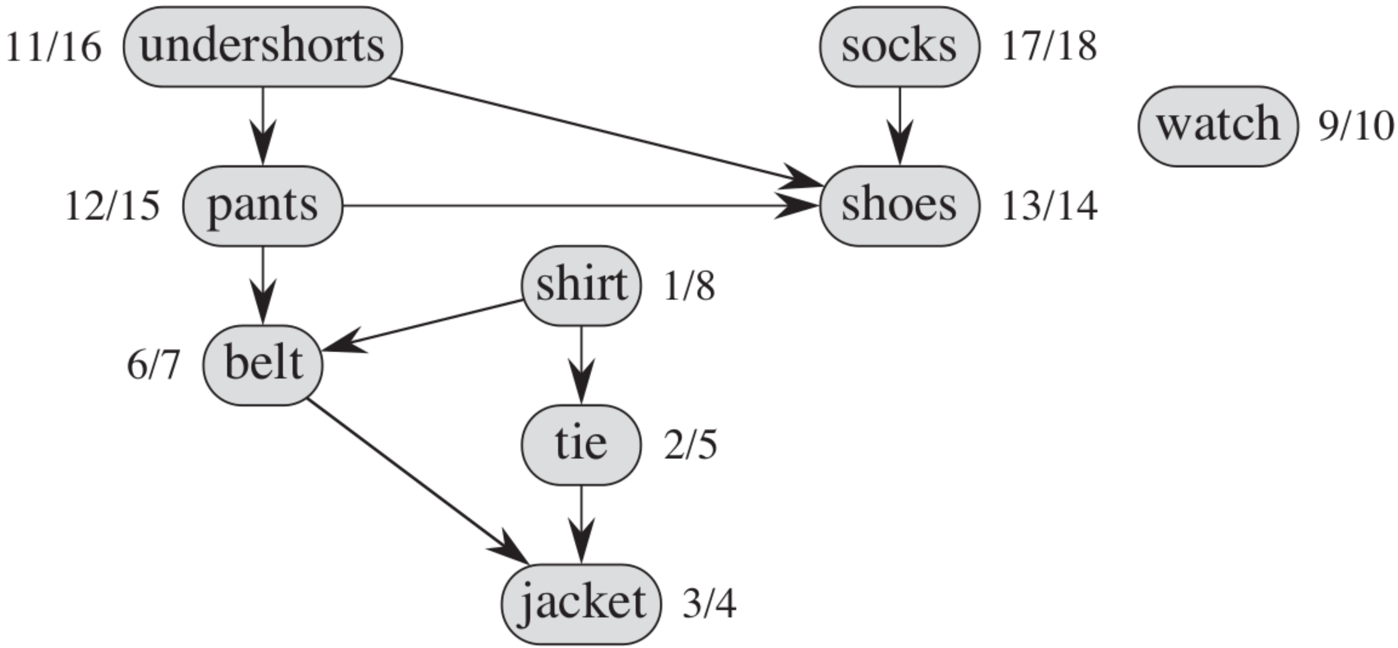
Topologisches Sortieren

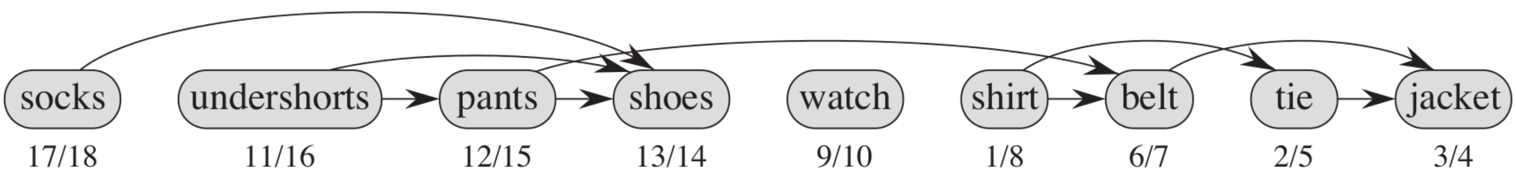
- „Finde eine totale Ordnung, die sich in eine gegeben partielle Ordnung einfügt“
- „Finde Anordnung der Knoten eines gerichteten Graphen auf einer Linie, so dass alle gerichteten Kanten von links nach rechts zeigen“

Topologisches Sortieren

- Topologische Sortierung eines DAG: lineare Sequenz der Knoten, so dass u vor v erscheint, wenn $(u, v) \in E$
- Algorithmus
  - Rufe $DFS(G)$ auf
  - Füge jeden abgearbeiteten Knoten am Kopf einer verketteten Liste ein
  - Gib die verkettete Liste zurück

Topologisches Sortieren





Topologisches Sortieren

- Lemma: Ein gerichteter Graph ist genau dann azyklisch wenn DFS keine Rückwärtskanten im Tiefensuchwald enthält.
- Beweisidee: Zeige
  - Rückwärtskante → Zyklus
  - Zyklus → Rückwärtskante

Korrektheit des Algorithmus

- Knoten werden absteigend nach Endzeiten sortiert
$$\cdots > u.f > \cdots > v.f > \cdots$$
- Sortierung ist korrekt, falls für alle Knoten u, v gilt:
$$u.f > v.f \Rightarrow (v, u) \notin E$$
- Es genügt zu zeigen: $(u, v) \in E \rightarrow v.f < u.f$
- Sei $(u, v) \in E$.
Wenn v über Kante (u, v) entdeckt wird, dann kann v nicht grau sein, da sonst v Vorgänger von u wäre, also (u, v) Rückwärtskante, Widerspruch zum vorherigen Lemma. → also ist v weiss oder schwarz (2 Fälle)
- Fall 1: v ist schwarz, dann gilt sicher $v.f < u.f$, da $u.f$ noch nicht gesetzt ist.
- Fall 2: v ist weiss, dann gilt nach dem Klammer-Theorem auch $v.f < u.f$

Übersicht

Graphenalgorithmen

- Begriffe & Darstellung von Graphen
- Breitensuche & Tiefensuche
- Topologisches Sortieren
- Starke Zusammenhangskomponenten

Starke Zusammenhangskomponenten

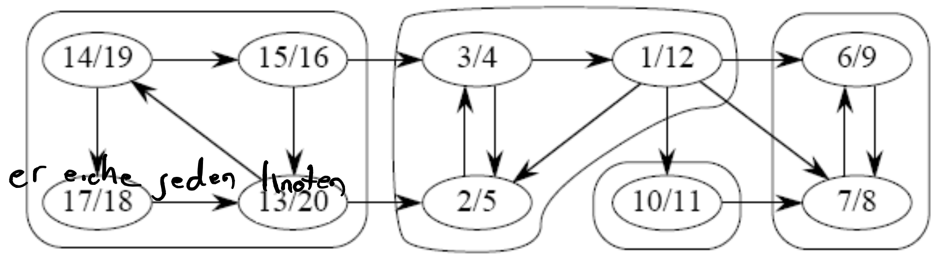
- Grundlage für viele Algorithmen für gerichtete Graphen
- Häufiges Muster
  1. Berechne starke Zusammenhangs-komponenten
  2. Führe gewünschten Algorithmus separat für jeder Komponente aus
  3. Füge Lösungen zusammen

Starke Zusammenhangskomponenten

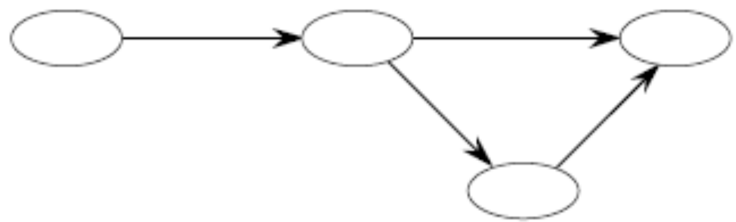
- Sei $G = (V, E)$ gerichteter Graph
- Starke Zusammenhangskomponente (strongly connected component, SCC) von G ist maximale Menge von Knoten $C \subseteq V$ so dass für alle $u, v \in C$ sowohl $u \to v$ als auch $v \to u$
- Algorithmus braucht transponierten Graphen $G^T = (V, E^T), E^T = \{(u, v) | (v, u) \in E\}$
- Beobachtung:
  G und G^T haben dieselben SCC

Starke Zusammenhangskomponenten

Starke Zusammenhangs-komponenten

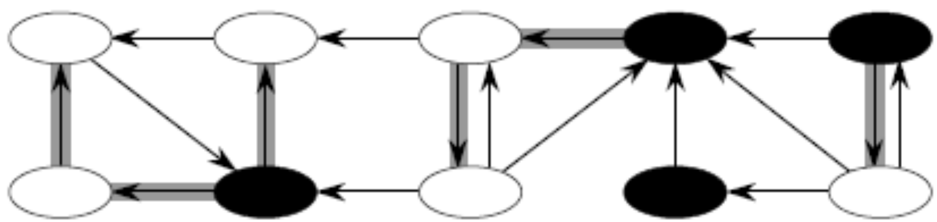


Komponenten-graph



SCC Algorithmus

1. Do DFS
2. G^{T}
3. DFS (roots blackened)



Komponentengraphen

- Komponentengraph $G^{SCC} = (V^{SCC}, E^{SCC})$
  - V^{SCC} hat einen Knoten für jede SCC in G
  - E^{SCC} hat eine Kante wenn es eine Kante zwischen entsprechenden SCCs in G gibt
- Lemma: G^{SCC} ist ein DAG
  - D.h. seien C und C' verschiedene SCCs in G, sei u, v in C, u', v' in C', und sei Pfad $u \to u'$ in G
  - Dann kann Pfad $v' \to v$ nicht in G sein
- Beweis durch Widerspruch: Falls $v' \to v$ existieren würde, wären C und C' dieselbe Zusammenhangskomponente

Starke Zusammenhangskomponenten

Algorithmus

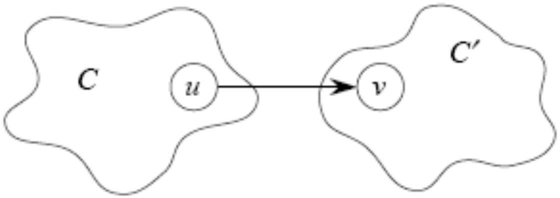
1. Berechne $DFS(G)$ und Endzeiten $u.f$
2. Berechne transponierten Graphen G^T
3. Berechne $DFS(G^T)$, wobei in der DFS Hauptschleife die Knoten in der Reihenfolge fallender $u.f$ (wie in Schritt 1 berechnet) betrachtet werden
4. Gib Knoten jedes Tiefensuchbaumes aus Schritt 3 als eine separate SCC aus

Erklärung

- Idee: Knoten in zweiter DFS auf transponiertem Graphen in fallender Reihenfolge nach Endzeiten von ersten DFS → Knoten des Komponentengraphen werden topologisch sortiert besucht
- Notation
  - $u.d$ und $u.f$ beziehen sich auf Resultat der ersten DFS
  - $d(U)$ früheste Startzeit aller Knoten in $U \subset V$
  - $f(U)$ späteste Endzeit aller Knoten in $U \subset V$

Erklärung

- Lemma: Seien C und C' verschiedene SCCs in $G = (V, E)$. Sei Kante (u, v) in E so dass u in C und v in C'. Dann $f(C) > f(C')$.



- Korollar: Seien C und C' verschiedene SCCs. Existiere Kante (u, v) in $\boldsymbol{E^T}$ mit u in C und v in C'. Dann $f(C) < f(C')$.

- Korollar: Seien C und C' verschiedene SCCs, $f(C) > f(C')$. Dann kann es in $\boldsymbol{E^T}$ keine Kante von C nach C' geben.

Erklärung Lemma

- Falls $d(C) < d(C')$
  - Sei x erster in C entdeckter Knoten. Zur Zeit $x.d$ sind alle Knoten in C und C' weiss
  - Theorem der weissen Pfade: alle Knoten in C und C' sind Nachkommen von x in DFS Baum
  - Klammerungstheorem: $x.f = f(C) > f(C')$
- Falls $d(C) > d(C')$
  - Sei y erster in C' entdeckter Knoten
  - Zur Zeit $y.d$ sind alle Knoten in C' weiss
  - Alle Knoten in C' sind Nachkommen von y
  - Zur Zeit $y.d$ sind alle Knoten in C weiss
  - Wegen Kante (u, v) gibt es keinen Pfad von C' nach C. Kein Knoten in C erreichbar von y
  - Zur Zeit $y.f$ sind alle Knoten in C weiss
  - Für alle w in C gilt $w.f > y.f$, und $f(C) > f(C')$

Erklärung

Intuition für Algorithmus

- Bei DFS auf G^T starte mit SCC C so, dass $f(C)$ Maximum
- Korollar: Weil $f(C) > f(C')$ für alle anderen Komponenten C', gibt es in $\boldsymbol{E^T}$ keine Kanten von C nach C'
- DFS besucht nur Kanten in C
- Nächste Wurzel in zweiter DFS ist in SCC C', so dass $\mathrm{f}(C')$ Maximum über alle SCCs ausser C. DFS besucht alle Knoten in C', und alle Kanten aus C' gehen nach C, welche schon besucht wurden.
- Deshalb nur Knoten in C' besucht
  - Etc. mit nächster Wurzel
- Von jeder neuen Wurzel der zweiten DFS, erreichen nur
  - Knoten in seiner SCC
  - Knoten, die bereits besucht wurden in zweiter DFS
- Besuchen Knoten von $(G^T)^{SCC}$ in umgekehrter topologisch sortierte Reihenfolge

Nächstes Mal

- Mehr Graphenalgorithmen