Algorithmen und Datenstrukturen
Kapitel 8 (Teil 2)
Graphen
Anwendung der Tiefensuche

Frank Heitmann
heitmann@informatik.uni-hamburg.de

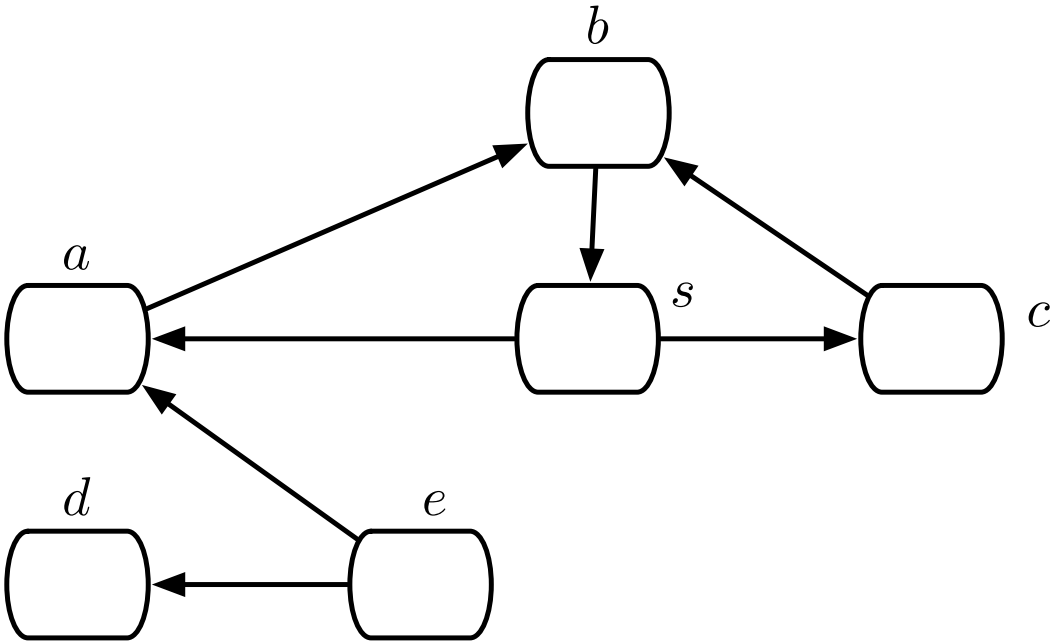
2. Dezember 2015

Graphen - Definitionen

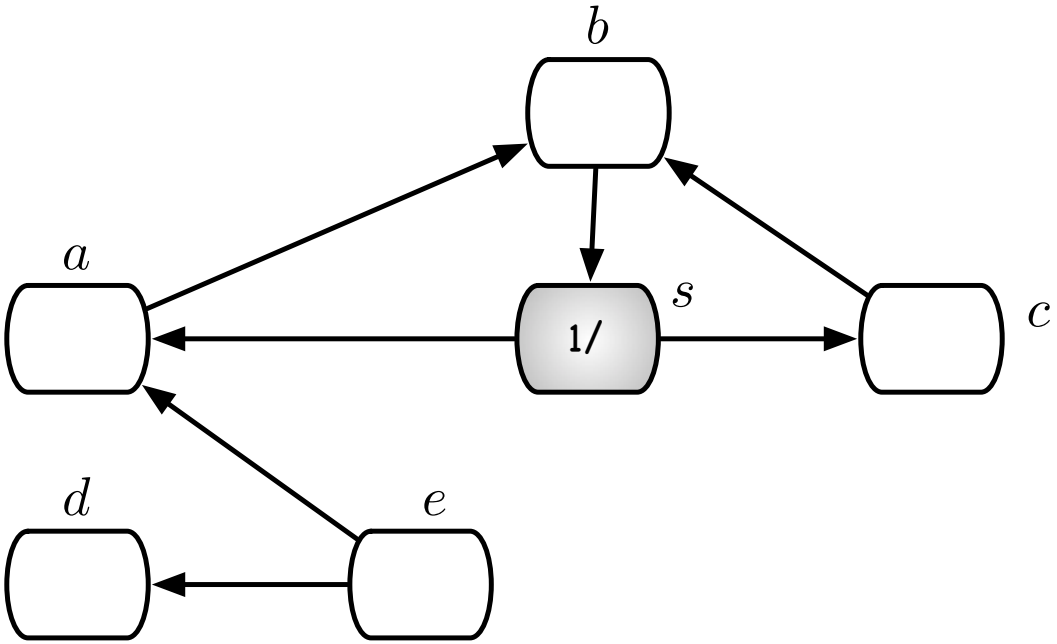
Definition

Graph, Knoten, Kante, ungerichteter Graph, gerichteter Graph, gewichteter Graph, dünn/dicht besetzt, vollständig, unabhängig, Grad, Nachbarn, Minimalgrad, Maximalgrad, Weg, Kreis, Länge (eines Weges/Kreises), Abstand, Teilgraph, induzierter (Teil-)graph, K^n, Baum, ...
Adjazenzmatrix, Adjazenzliste
Breitensuche, Tiefensuche

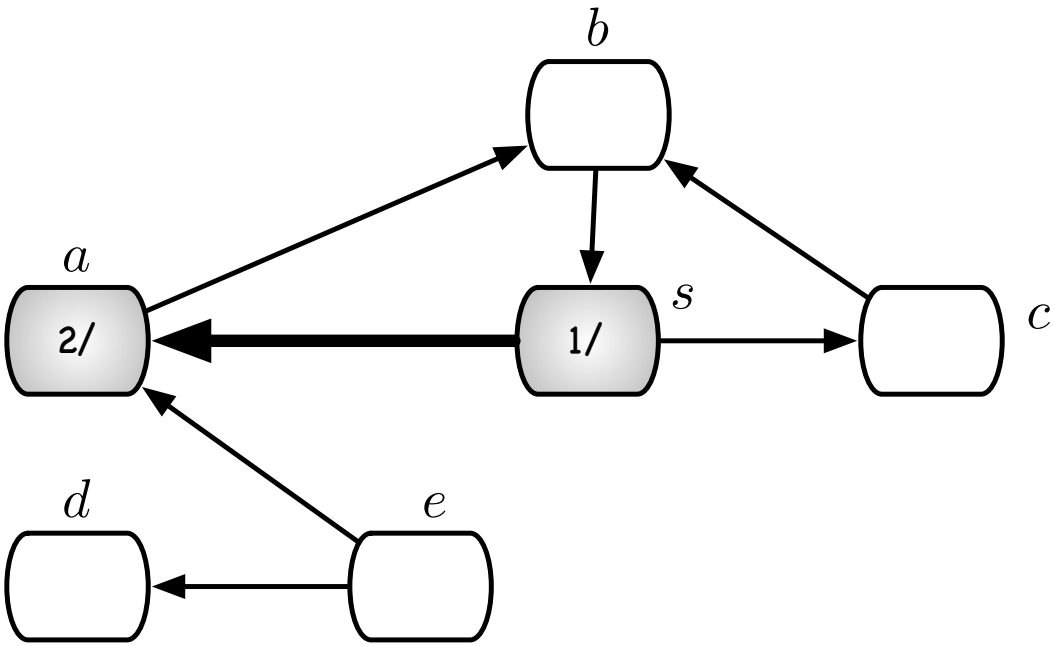
Tiefensuche - Beispiel



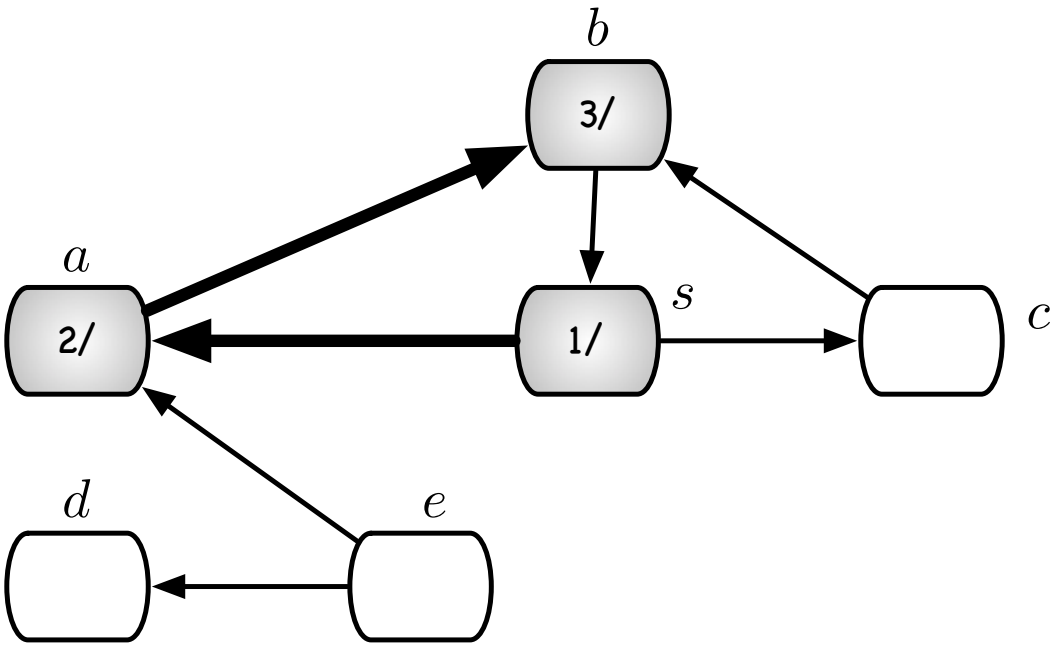
Tiefensuche - Beispiel



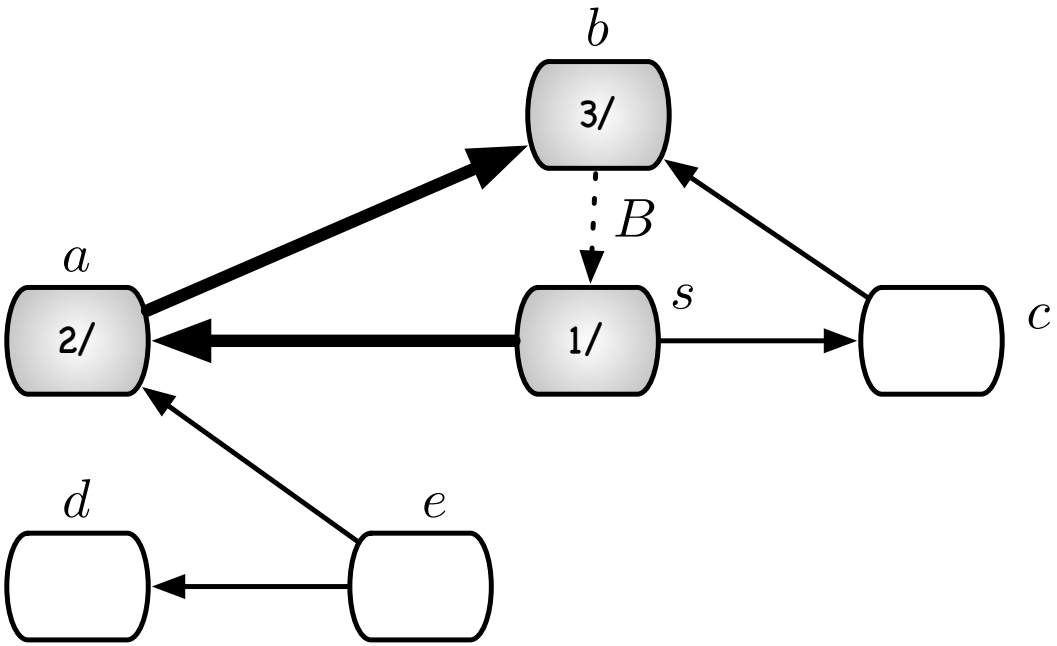
Tiefensuche - Beispiel



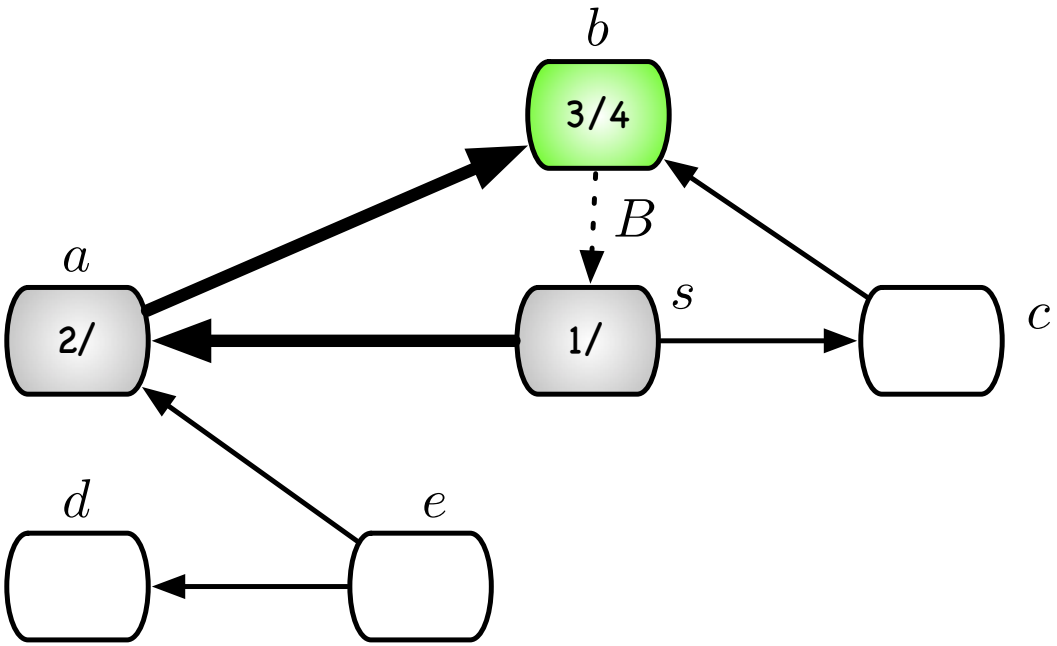
Tiefensuche - Beispiel



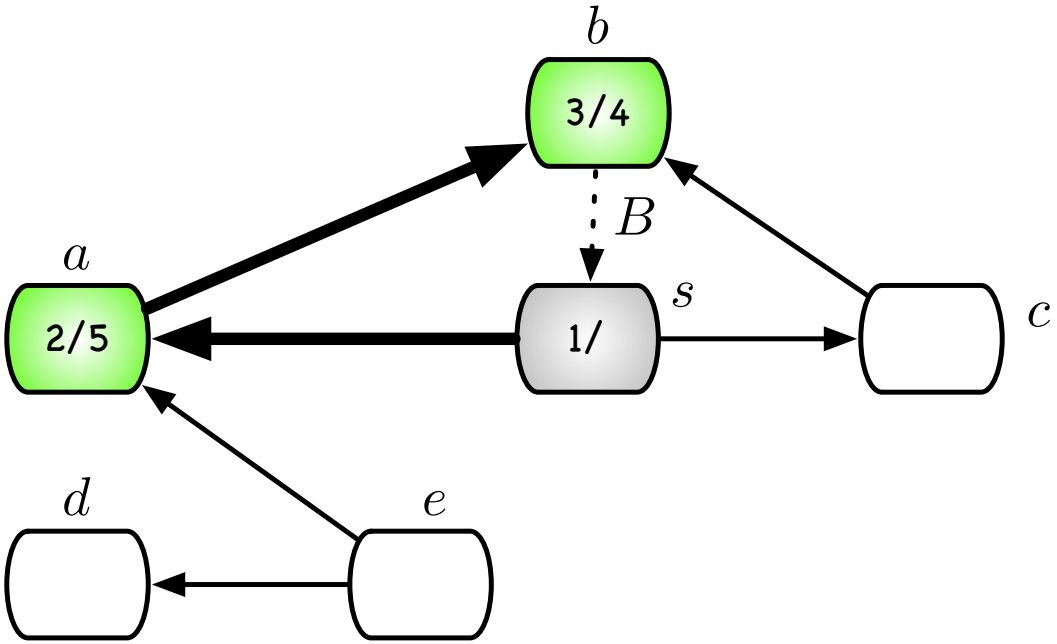
Tiefensuche - Beispiel



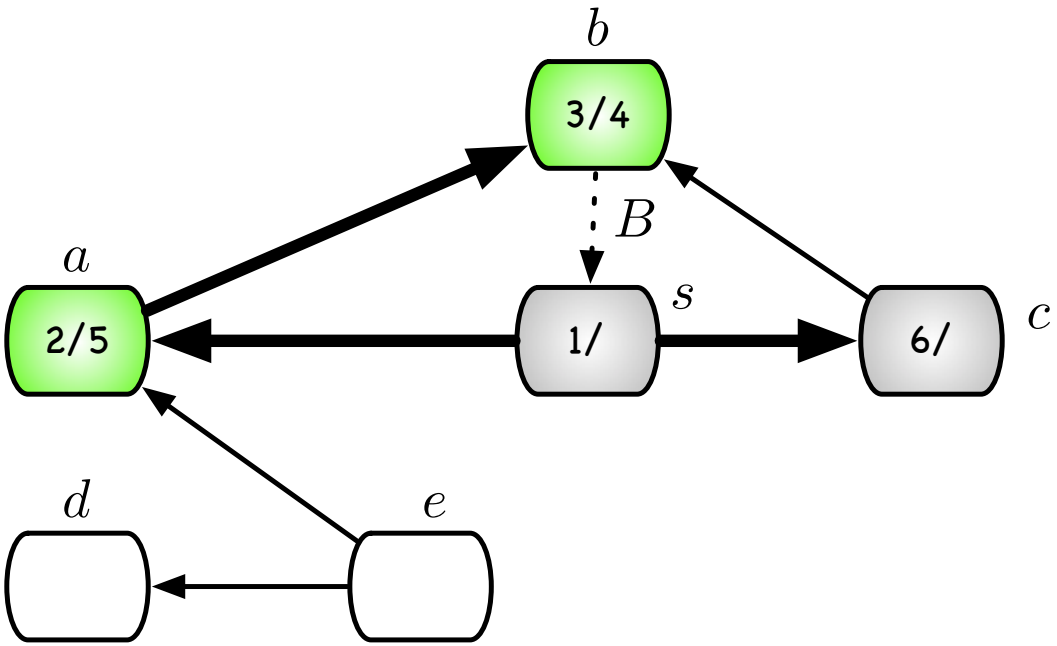
Tiefensuche - Beispiel



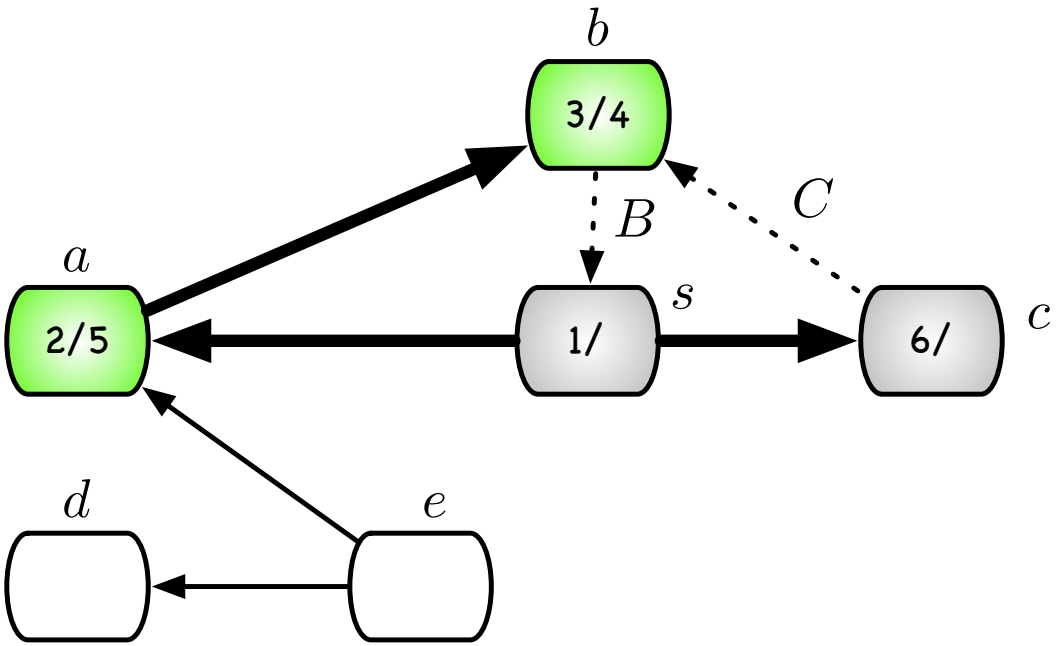
Tiefensuche - Beispiel



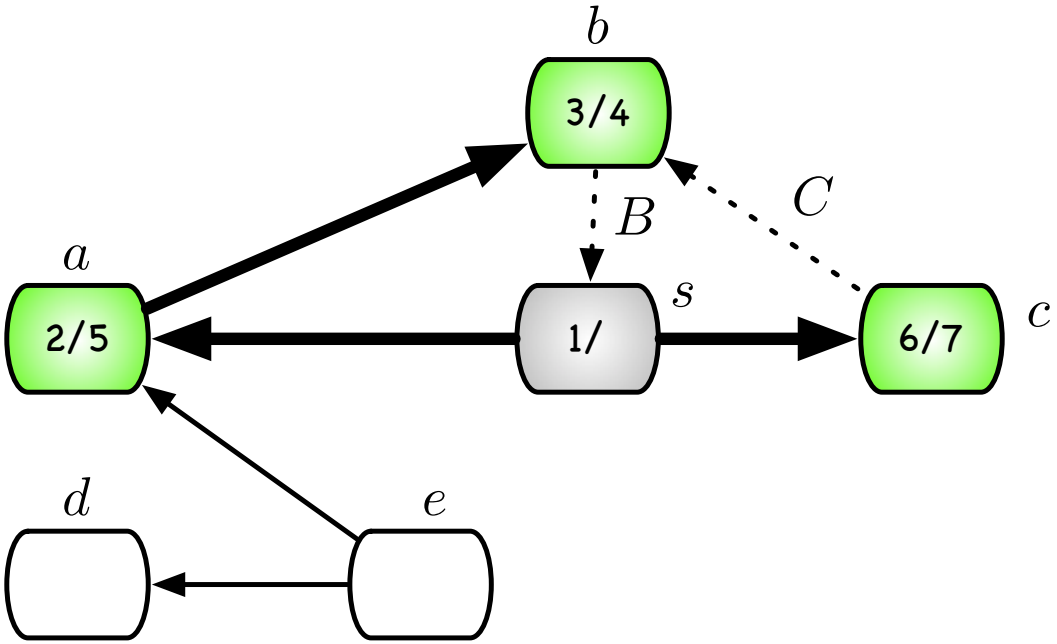
Tiefensuche - Beispiel



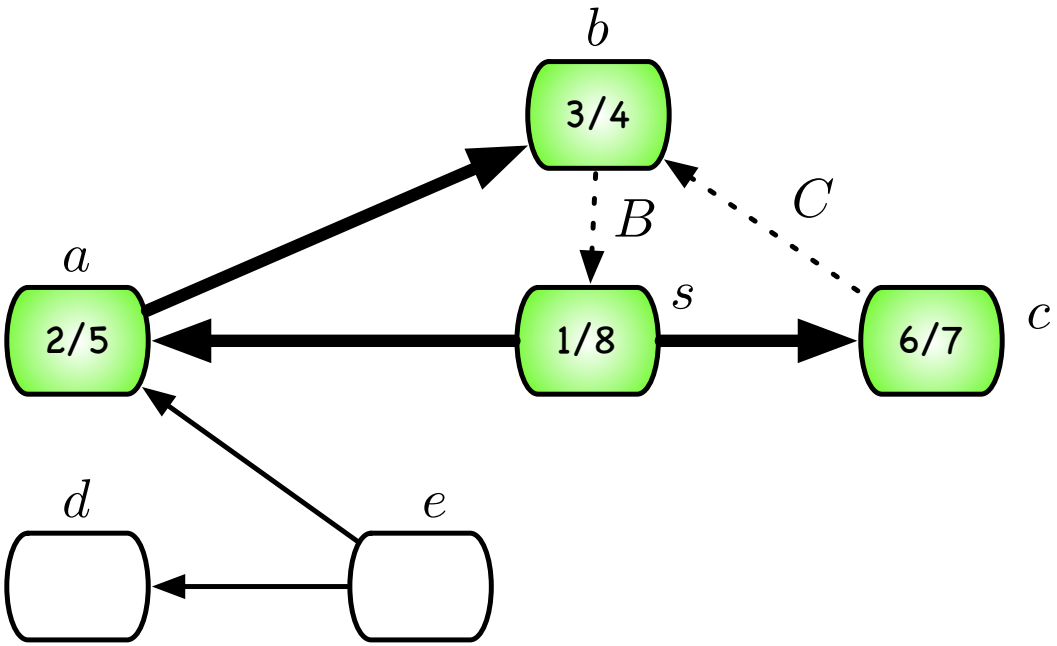
Tiefensuche - Beispiel



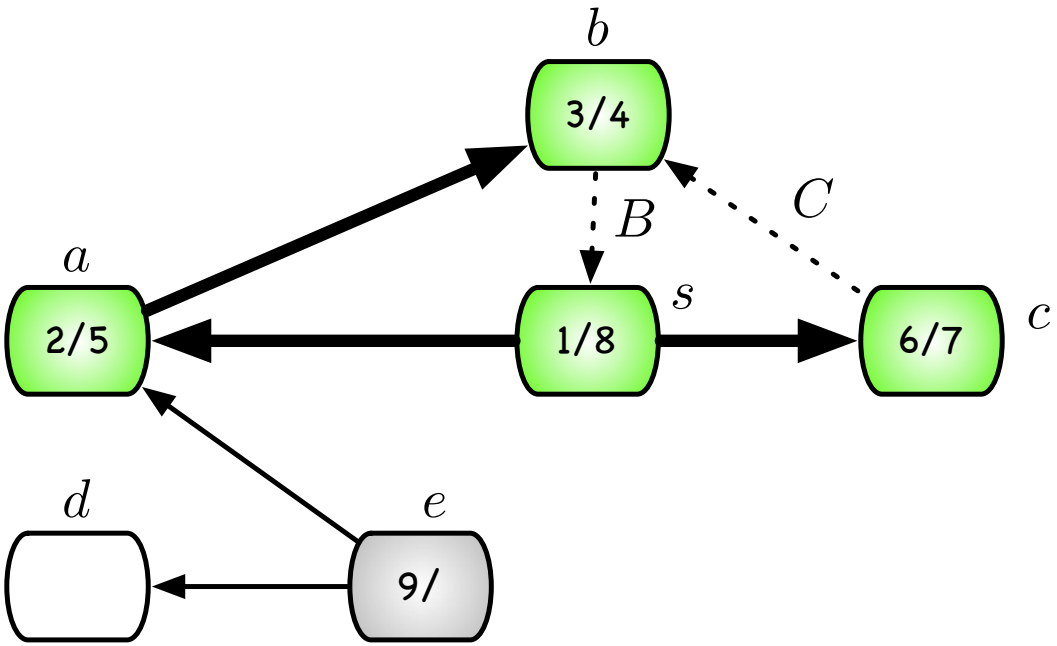
Tiefensuche - Beispiel



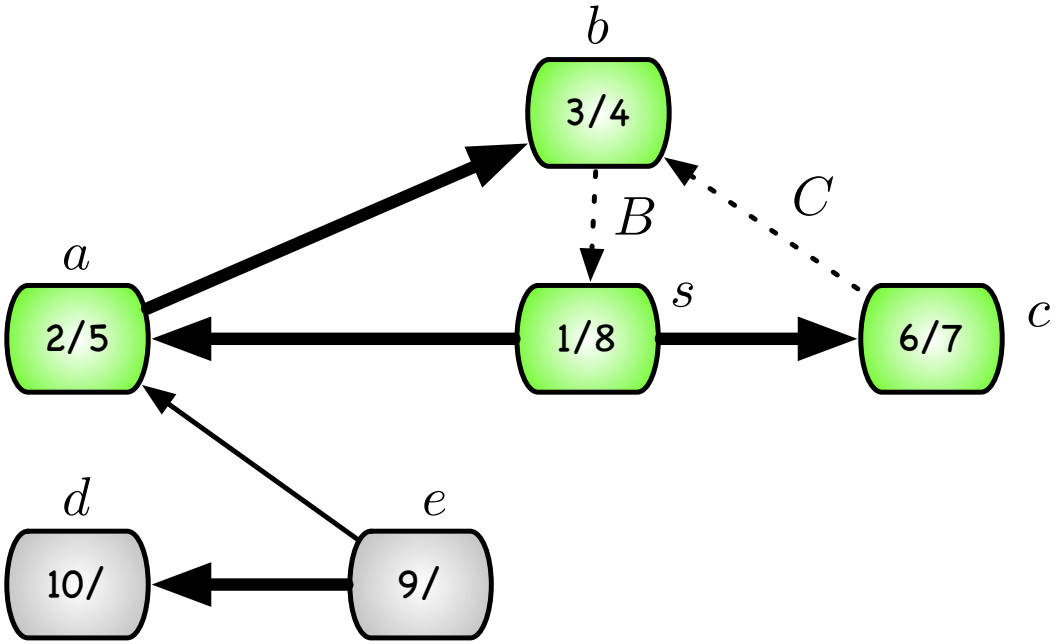
Tiefensuche - Beispiel



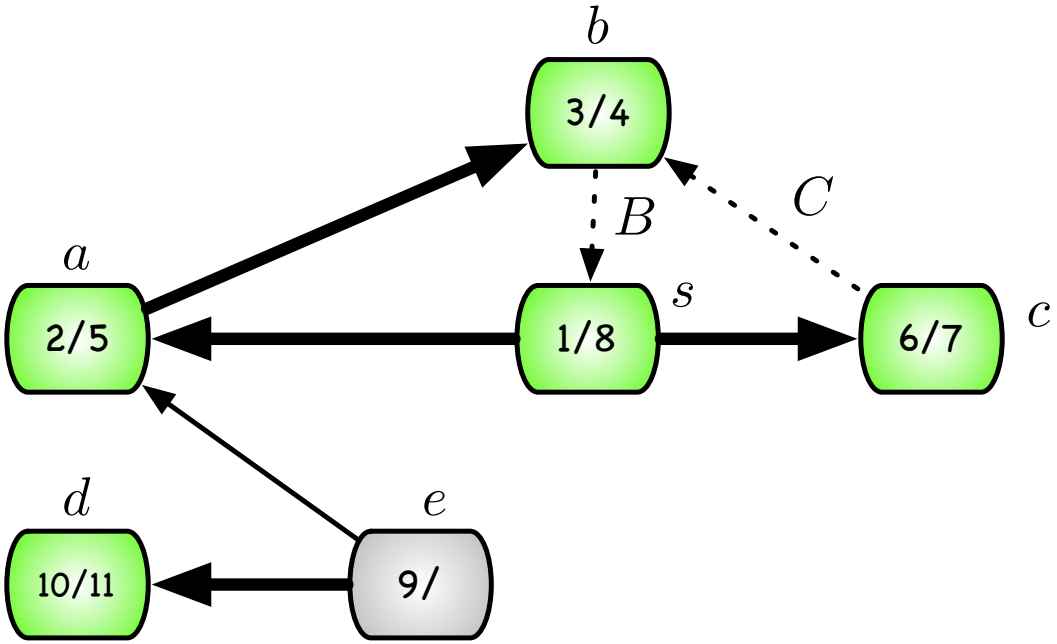
Tiefensuche - Beispiel



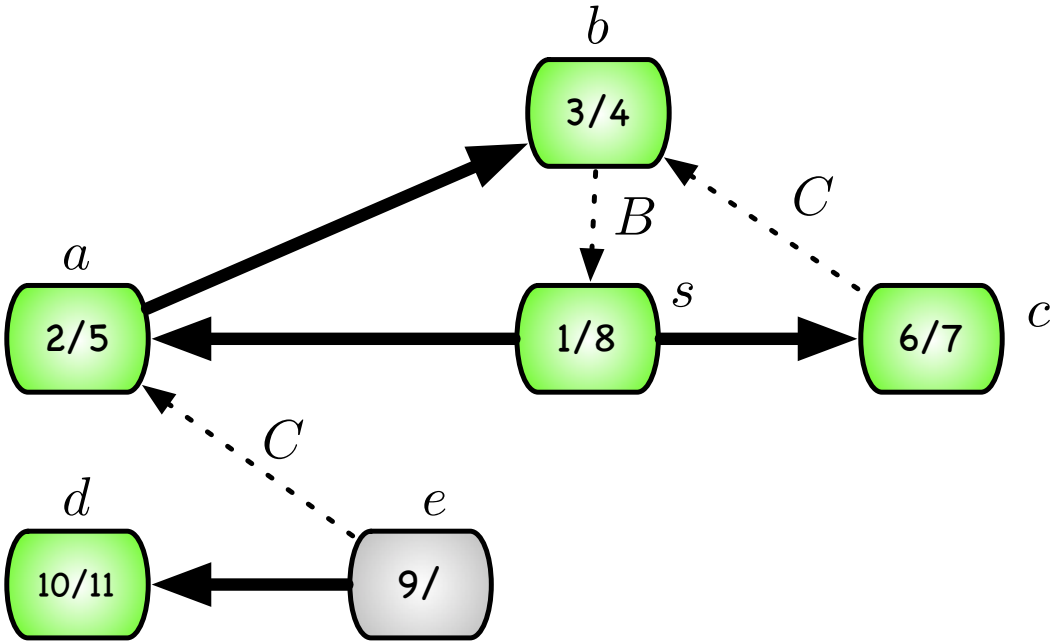
Tiefensuche - Beispiel



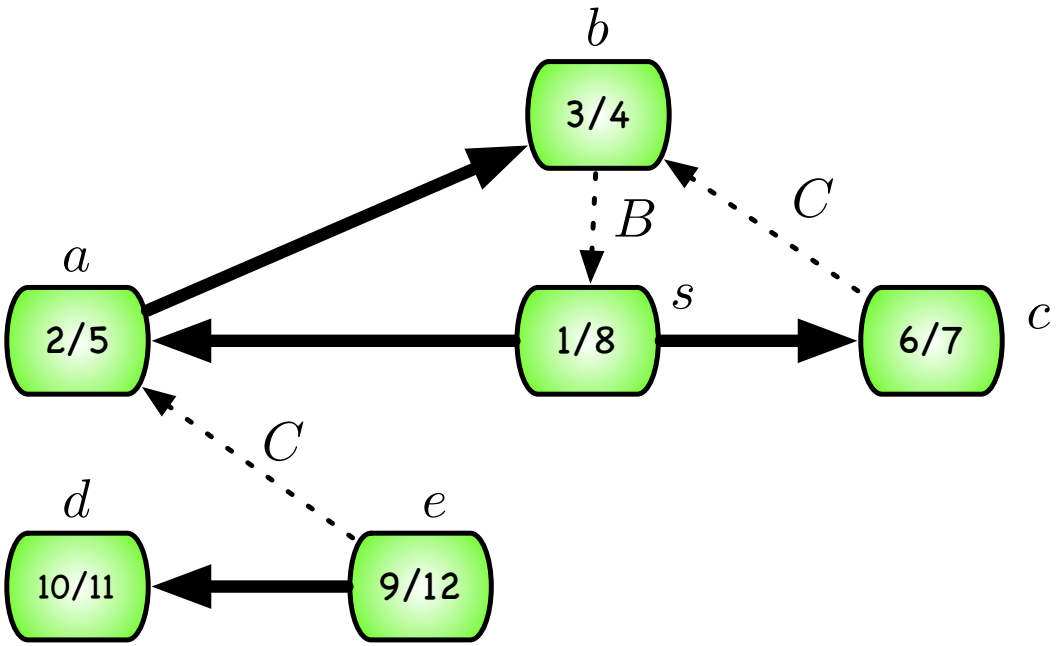
Tiefensuche - Beispiel



Tiefensuche - Beispiel



Tiefensuche - Beispiel



Tiefensuche - Besondere Kanten

Man kann bei der Tiefensuche die Kanten klassifizieren je nachdem, welche Farbe der Knoten v bei einer Kanten (u, v) hat (u ist der Knoten der gerade bearbeitet wird, v ist in $Adj[u]$):

- farbe$[v]$ = weiss: Baumkante.
- farbe$[v]$ = grau: Rückwärtskante.
- farbe$[v]$ = schwarz: Vorwärts- oder Querkante.
  - Vorwärtskante, falls $d[u] < d[v]$.
  - Querkante, falls $d[u] > d[v]$.

Tiefensuche - Besondere Kanten

Dabei ist

- Baumkante. Kanten, die zum Baum gehören. v wurde bei der Sondierung der Kante (u, v) entdeckt (erstmalig besucht).
- Rückwärtskante. Kante (u, v), die u mit einem Vorfahren v im Tiefensuchbaum verbindet.
- Vorwärtskante. (Nicht-Baum-)Kante (u, v), die u mit einem Nachfahre in v im Tiefensuchbaum verbindet.
- Querkanten. Die übrigen Kanten. Im gleichen Tiefensuchbaum, wenn der eine Knoten nicht Vorfahre des anderen ist oder zwischen zwei Knoten verschiedener Tiefensuchbäume (im Wald).

Anmerkung

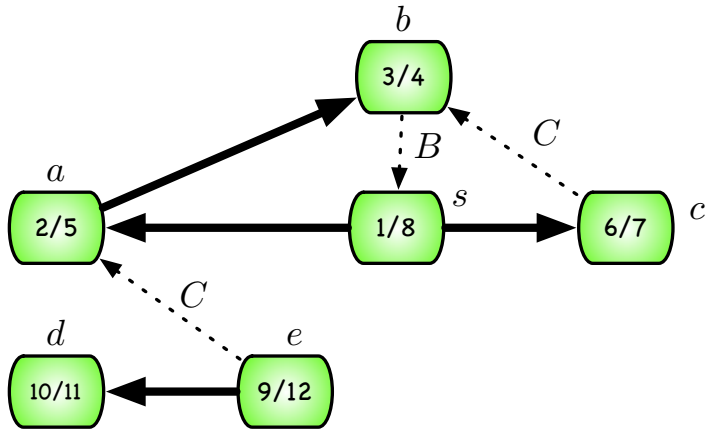
Bei einem ungerichteten Graphen gibt es nur Baum- und Rückwärtskanten.

Tiefensuche - Wichtige Eigenschaften

Es folgen einige wichtige - und nützliche - Eigenschaften der Tiefensuche...

(Beweise kann man im [Cormen] nachlesen. Nur ca. eine Seite)

Tiefensuche - Klammerungstheorem (1/2)

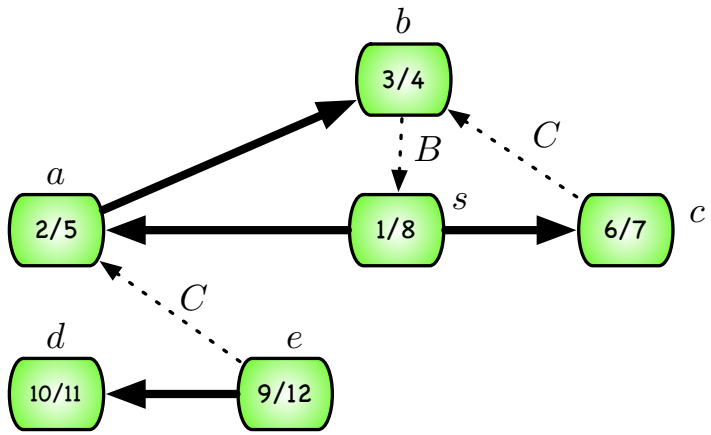


Satz

Bei der Tiefensuche erfüllt jedes Paar u, v von Knoten genau eine der folgenden Bedingungen:

1. *Die Intervalle* $[d[u], f[u]]$ *und* $[d[v], f[v]]$ *sind paarweise disjunkt und keiner der Knoten u und v ist im Tiefensuchwald ein Nachfahre des anderen.*

Tiefensuche - Klammerungstheorem (2/2)

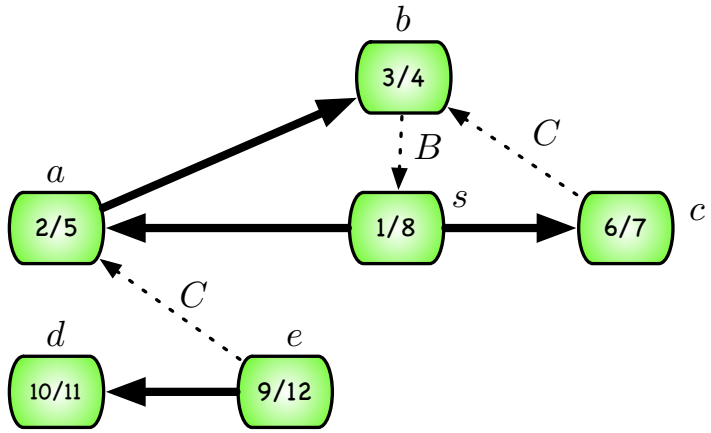


Satz

Bei der Tiefensuche erfüllt jedes Paar u, v von Knoten genau eine der folgenden Bedingungen:

2. *Das Intervall* $[d[u], f[u]]$ *ist vollständig im Intervall* $[d[v], f[v]]$ *enthalten und u ist im Tiefensuchwald ein Nachfahre von v.*
3. *Wie* 2. *nur mit u und v vertauscht.*

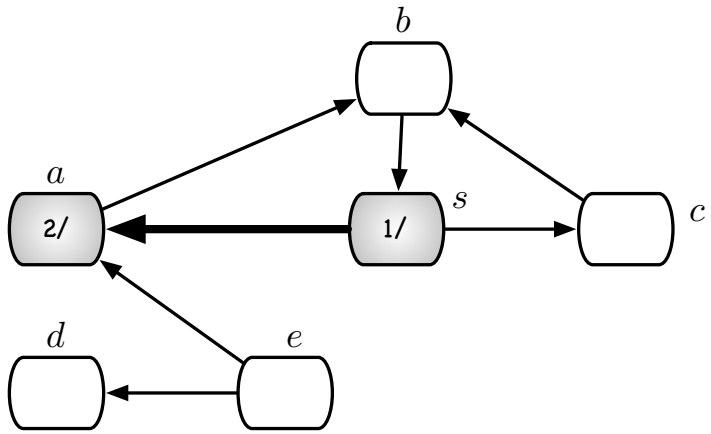
Tiefensuche - Intervalle der Nachfahren



Satz

Im Tiefensuchwald von G ist v genau dann ein echter Nachfahre von u, wenn $d[u] < d[v] < f[v] < f[u]$ gilt.

Tiefensuche - Theorem der weißen Pfade



Satz

Im Tiefensuchwald von G ist v genau dann ein Nachfahre von u, wenn v zur Zeit d[u] (u wird entdeckt) von u aus entlang eines nur aus weißen Knoten bestehenden Pfades erreichbar ist.

Problemstellung

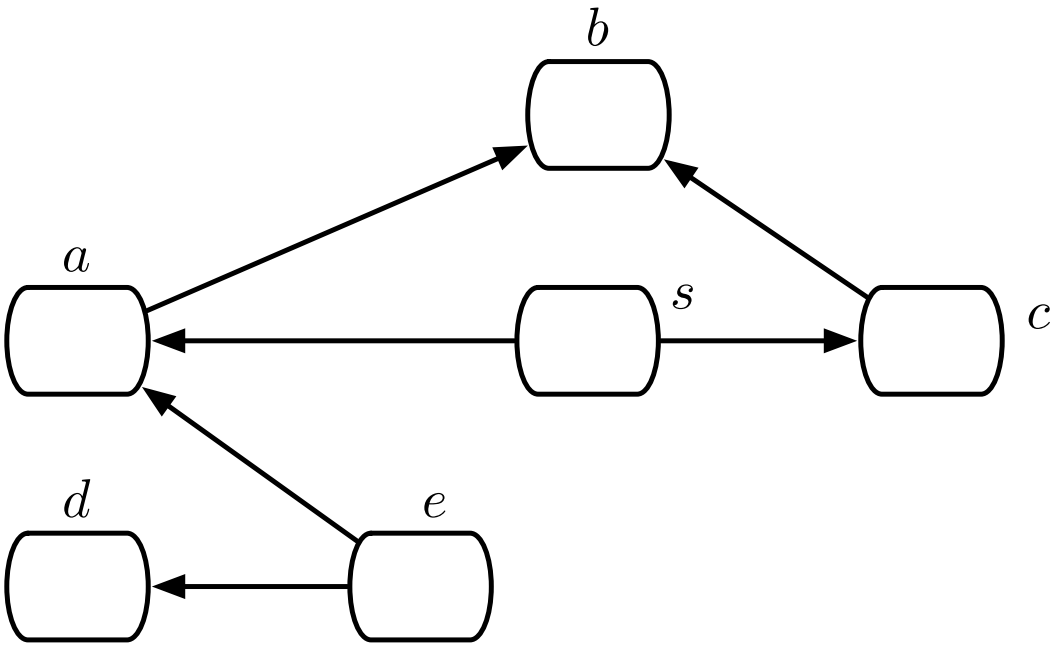
Definition (Topologisches Sortieren)

Eingabe: Gegeben ein gerichteter azyklischer Graph $G = (V, E)$.
Gesucht: Eine *topologischer Sortierung*, d.h. eine lineare Anordnung der Knoten mit der Eigenschaft, dass u in der Anordnung vor v liegt, falls es in G eine Kante (u, v) gibt.

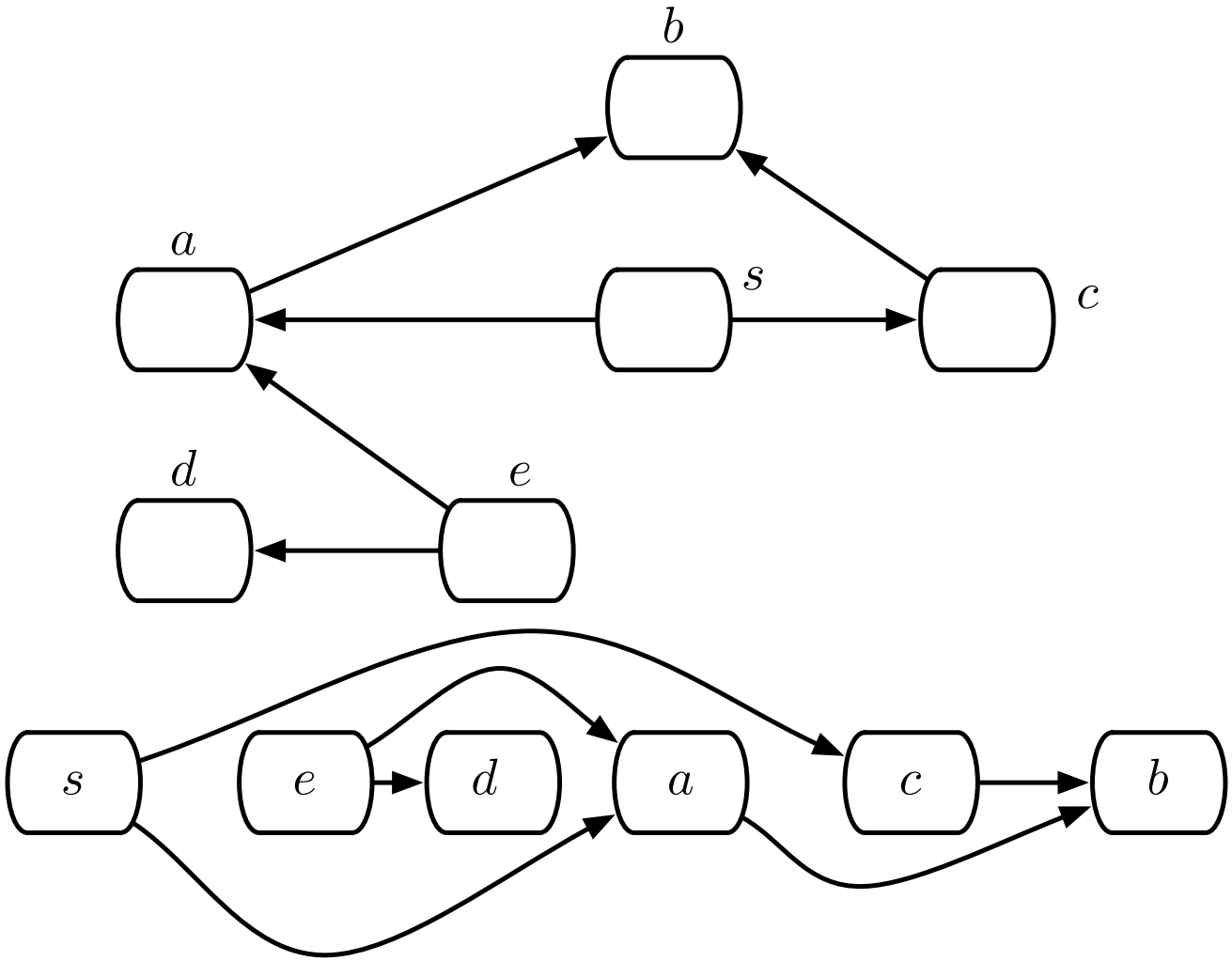
Anmerkung

Dieses Problem tritt häufig bei der Modellierung von Ereignissen auf, die mit einer Priorität (bzw. Abhängigkeiten) versehen sind (und entsprechend bearbeitet werden sollen).

Ein Beispiel



Ein Beispiel



TopoSort - Algorithmus und Idee

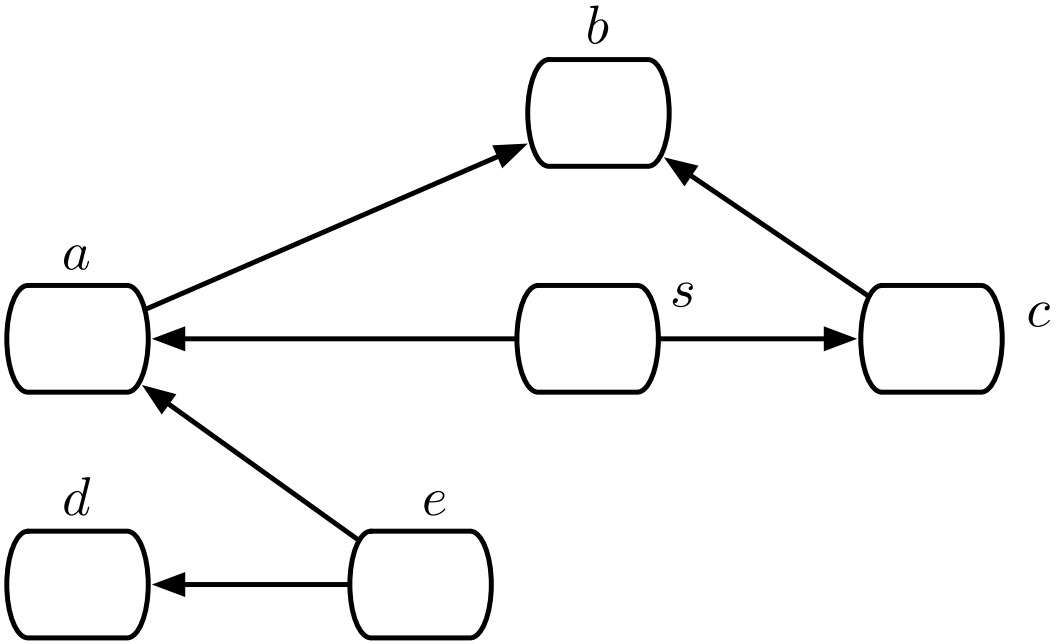
Algorithmus 1 TopoSort(G)

1: Berechne die Endzeiten $f[v]$ für alle $v \in V$ durch Aufruf von DFS(G).
2: Füge jeden abgearbeiteten Knoten an den Kopf einer verketteten Liste ein.
3: Liefere die verkettete Liste zurück.

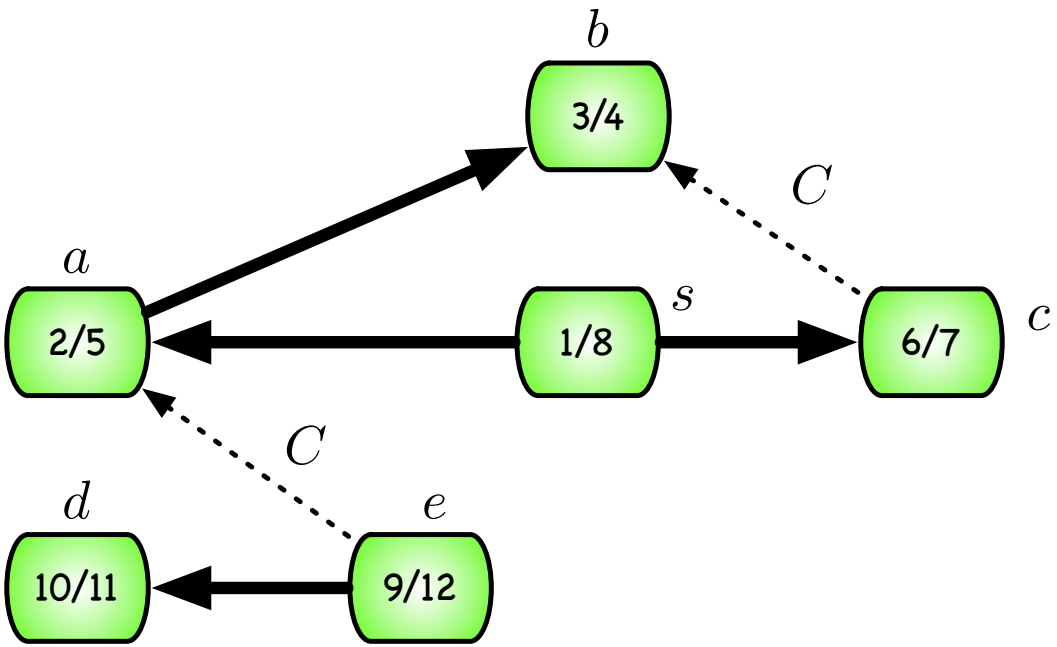
Anmerkung

Die Kernidee ist es die Knoten in umgekehrter Reihenfolge ihrer Abarbeitungszeiten auszugeben (d.h. der Knoten, der zuletzt abgearbeitet wurde, wird zuerst ausgegeben usw.).

TopoSort - Beispiel

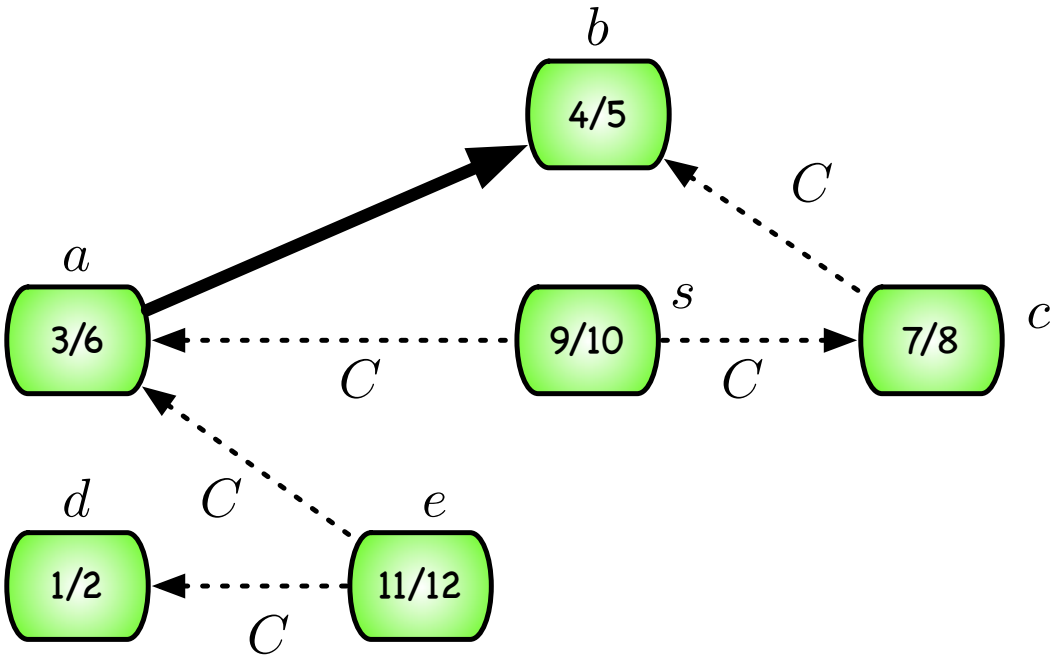


TopoSort - Beispiel



e, d, s, c, a, b

TopoSort - Beispiel



e, s, c, a, b, d

Zur Nachbereitung: Beweise

Zur Nachbereitung

Gleich kommt ein Beweis, den wir in der Vorlesung zum Anlass genommen haben, um Widerspruchsbeweise zu wiederholen. Wir haben dabei ganz generell über Widerspruchsbeweise geredet. Bei den Erklärungen war aber die Kontraposition recht zentral, was sich dann ein bisschen unglücklich vermischte. Hier daher noch ein paar Folien, um das exakt zu behandeln.

Zur Nachbereitung: Beweise

Zur Nachbereitung

Will man eine Aussage X beweisen, so nimmt man in einem **Widerspruchsbeweis** (auch: indirekter Beweis) das Gegenteil von X an (also $\neg X$) und zeigt, dass dies zu einem Widerspruch führt (d.h. zu einer falschen Aussage, die nicht sein kann wie z.B. $0 = 1$ oder "der Kreis C aus 4 Knoten besteht aus 5 Knoten").
Da $\neg X$ dann nicht sein kann (da daraus Unsinn folgt), darf man dann schlussfolgern, dass X gelten muss. (Wichtig hierfür ist noch, dass die Aussage X tatsächlich wahr oder falsch sein muss. Mit anderen mathematischen Formalismen haben wir es hier aber i.A. nicht zu tun.)

Zur Nachbereitung: Beweise

Zur Nachbereitung

Will man eine Aussage $X = A \Rightarrow B$ beweisen, also eine "Wenn A, dann B"-Aussage, dann nimmt man im direkten Beweis die Gültigkeit von A an und zeigt, dass (unter dieser Annahme) B gilt. Dies zeigt dann $A \Rightarrow B$.

Man kann aber auch die **Kontraposition** von $X = A \Rightarrow B$ zeigen. Dies ist die Aussage $\neg B \Rightarrow \neg A$, d.h. man zeigt, dass aus der Annahme $\neg B$ die Aussage $\neg A$ folgt.

Dass dies auch X zeigt, kann man sich wie folgt klarmachen: Gilt $\neg B \Rightarrow \neg A$, dann kann aus A nun nicht $\neg B$ folgen, denn aus $\neg B$ folgt ja $\neg A$ und damit hätten wir dann gezeigt, dass $\neg A$ aus A folgt, was ein Widerspruch ist. Aus A muss also B folgen, d.h. $A \Rightarrow B$ muss gelten. (Eine andere Möglichkeit sich dies klarzumachen ist, zu bemerken, dass $A \Rightarrow B$ und $\neg B \Rightarrow \neg A$ äquivalente aussagenlogische Formeln sind.)

Zur Nachbereitung: Beweise

Zur Nachbereitung

Zeigt man nun $X = A \Rightarrow B$ per *Widerspruchsbeweis*, dann geht man wegen $\neg X \equiv \neg(A \Rightarrow B) \equiv \neg(\neg A \vee B) \equiv A \wedge \neg B$ von $A \wedge \neg B$ aus und führt dies zu einem Widerspruch.

Im gleich folgenden Beispiel kommt man (um $A \Rightarrow B$ zu zeigen) von $\neg B$ zu $\neg A$ (und wäre damit schon fertig, wenn es einem einfach um die Kontraposition geht). Nun steht in den Folien aber Widerspruchsbeweis (was also eigentlich gar nicht nötig wäre). Man hat damit aber auch einen Widerspruchsbeweis geführt, denn bei diesem wäre man ja von $A \wedge \neg B$ ausgegangen. Mit dem $\neg B$ kommt man nun zu $\neg A$ und dann hat man mit dem A von dem man ja auch ausging nun aber einen Widerspruch.

Zur Nachbereitung: Beweise

Zur Nachbereitung

Das zuletzt beschriebene Vorgehen ist dann nützlich, wenn man im Beweis die Aussage A an einer Stelle benötigt (und sei es nur zur Illustration). Im nachfolgenden Beweis wird sie aber gar nicht benutzt und daher wäre es besser dort statt Widerspruchsbeweis einfach zu schreiben "wir zeigen die Kontraposition".

Zur Nachbereitung

Noch ein Exkurs: Will man $C \Rightarrow D$ zeigen, so hatten wir gesagt man nimmt C an und versucht zu D zu kommen. Nimmt man nun C an, so kann man gleich als nächstes noch $\neg D$ annehmen, gelangt man damit zu einem Widerspruch, so weiß man wie ganz oben beim Widerspruchsbeweis erwähnt, dass $\neg D$ nicht gelten darf und also D gelten muss und schon ist man von C zu D gekommen. Auch hier ist es also möglich weitere Annahmen einzufügen und so mit mehr Dingen arbeiten zu können. (Man kann sich so klar machen, dass die beiden eben vorgestellten Vorgehen im Grunde identisch sind.)

TopoSort - Analyse (Korrektheit)

Satz

Ein gerichteter Graph G ist genau dann azyklisch, wenn eine Tiefensuche auf G keine Rückwärtskanten liefert.

Beweis.

Richtung von links nach rechts durch Widerspruchsbeweis: Angenommen die Tiefensuche liefert eine Rückwärtskante (u, v), dann ist v ein Vorfahre von u im Tiefensuchbaum(/wald), d.h. es gibt in G einen Pfad von v nach u und dann mittels der Kante (u, v) einen Kreis. (Widerspruch!) □

TopoSort - Analyse (Korrektheit)

Satz

Ein gerichteter Graph G ist genau dann azyklisch, wenn eine Tiefensuche auf G keine Rückwärtskanten liefert.

Beweis.

Richtung von rechts nach links durch Widerspruchsbeweis: Angenommen der Graph enthält einen gerichteten Kreis C. Sei v der erste Knoten, der auf C entdeckt wird und sei (u, v) die vorherige Kante auf C. Da die Knoten auf C einen weißen Pfad von v nach u bilden, wird wegen des Theorems der weißen Pfade u ein Nachfolger von v im Tiefensuchwald. Die Kante (u, v) wird dann eine Rückwärtskante. (Widerspruch!) □

TopoSort - Analyse (Korrektheit)

Satz

`TopoSort`(G) *erzeugt für einen gerichteten azyklischen Graphen* G *eine topologische Sortierung.*

Beweis.

Es genügt $f[v] < f[u]$ zu zeigen, wenn es in G eine Kante (u, v) gibt. Sei (u, v) also eine von `DFS`(G) sondierte Kante. v kann nicht grau sein, da dies sonst eine Rückwärtskante wäre (im Widerspruch zum vorherigen Satz). v muss also weiß oder schwarz sein. Ist v schwarz, so ist er bereits abgearbeitet und es gilt $f[v] < f[u]$. Ist v weiß, so wird er ein echter Nachfahre von u und es gilt (Intervalle der Nachfahren) $d[u] < d[v] < f[v] < f[u]$. □

TopoSort - Analyse (Laufzeit)

Die Kernidee von TopoSort:

- Bestimme die Abarbeitungszeit $f[v]$ für jeden Knoten v durch Tiefensuche.
- Speichere jeden abgearbeiteten Knoten am Kopf einer Liste.
- Gebe die Knoten nach fallenden Werten von $f[v]$ aus, beginnnend mit dem größten Wert (d.h. gib obige Liste aus).

Satz

`TopoSort`(G) *ist korrekt und arbeitet in* $\Theta(V + E)$.

Problemstellung

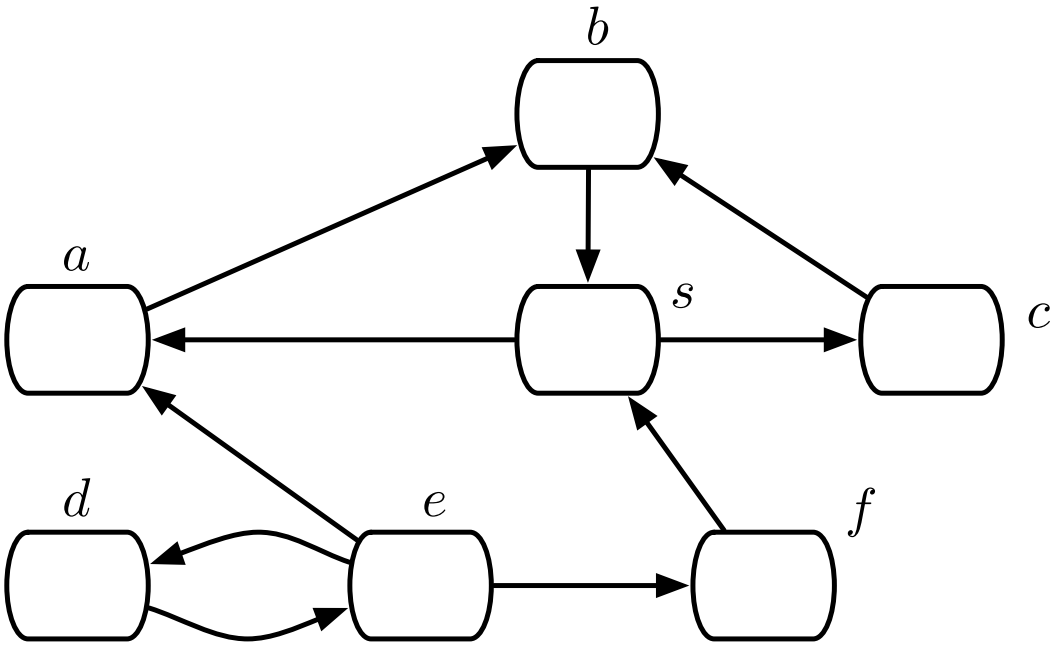
Definition (Bestimmung starker Zusammenhangskomponenten)

Eingabe: Gegeben ein gerichteter Graph $G = (V, E)$.
Gesucht: Die starken Zusammenhangskomponenten (SCCs) des Graphen, d.h. maximale Menge $C \subseteq V$ derart, dass für jedes Paar $u, v \in C$ sowohl $u \Rightarrow v$ als auch $v \Rightarrow u$ gilt (es gibt einen Pfad von u nach v und andersherum).

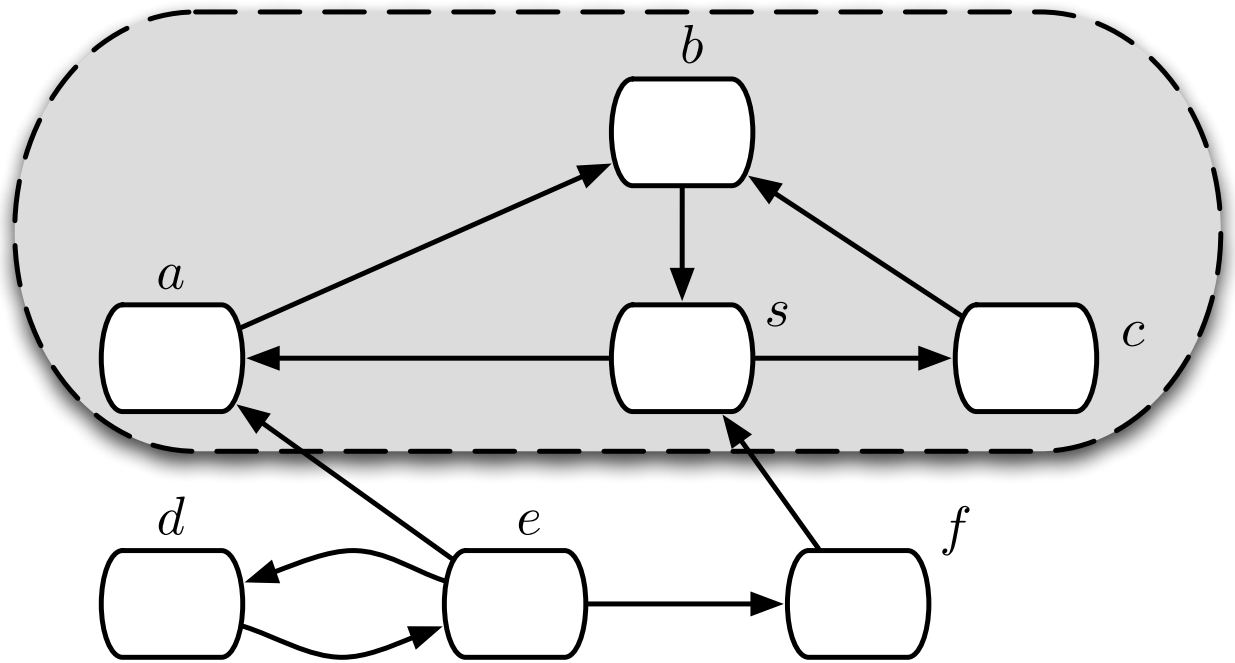
Anmerkung

Dieses Problem tritt oft bei gerichteten Graphen auf. Zunächst wird der Graph in seine starken Zusammenhangskomponenten zerlegt und dann werden diese separat betrachtet. (Oft werden die Lösungen anschließend noch entsprechend der Verbindungen zwischen den einzelnen Komponenten zusammengefügt.)

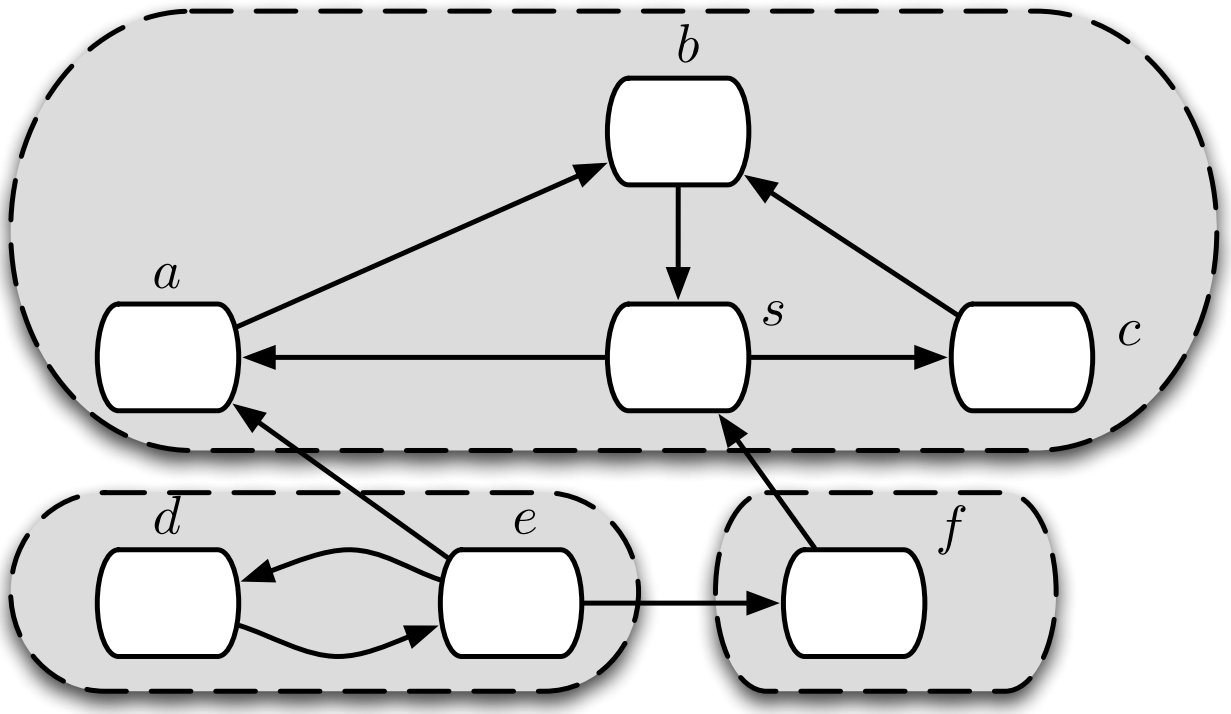
Ein Beispiel



Ein Beispiel



Ein Beispiel



Der Komponentengraph

Oft ist auch der Komponentengraph gesucht...

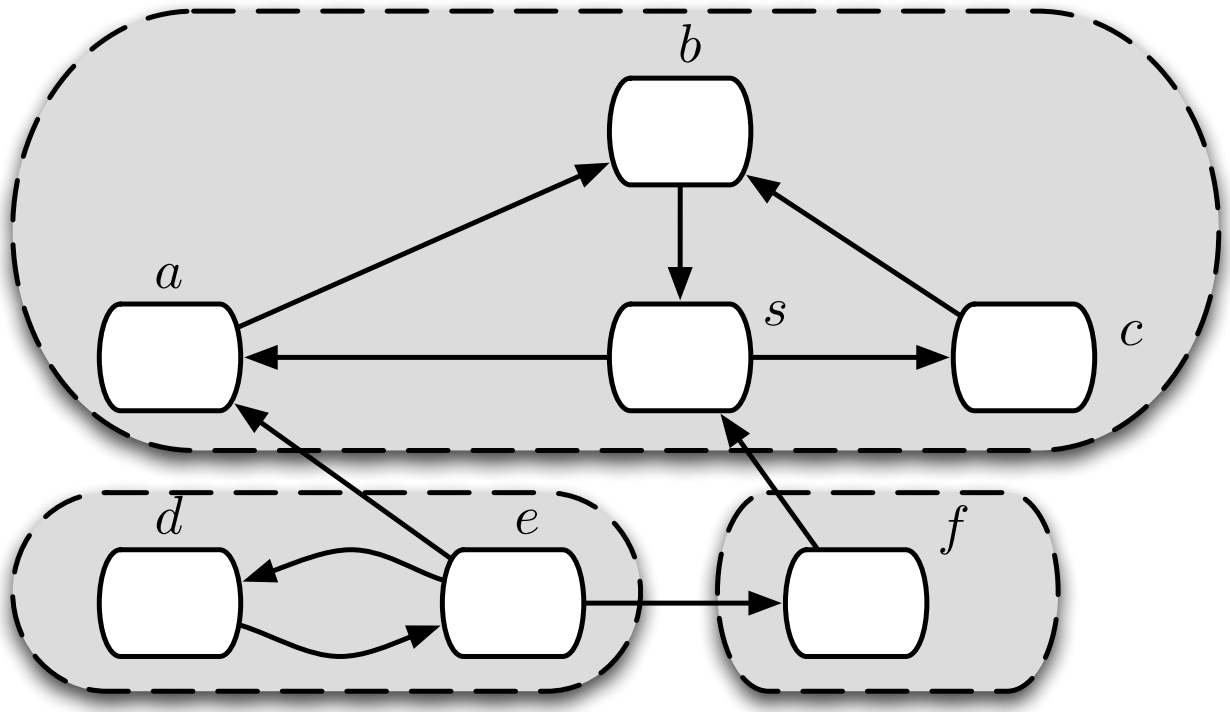
Definition

Sei G ein gerichteter Graph mit Zusammenhangskomponenten $C_1, \ldots, C_k$. Der Komponentengraph $G^{scc} = (V^{scc}, E^{scc})$ ist definiert durch $V^{scc} = \{v_1, \ldots, v_k\}$ (jeder Knoten v_i repräsentiert dabei eine Zusammenhangskomponente C_i) und (v_i, v_j) ist genau dann eine Kante in E^{scc}, wenn es in G einen Knoten $x \in C_i$, einen Knoten $y \in C_j$ und eine Kante (x, y) gibt.

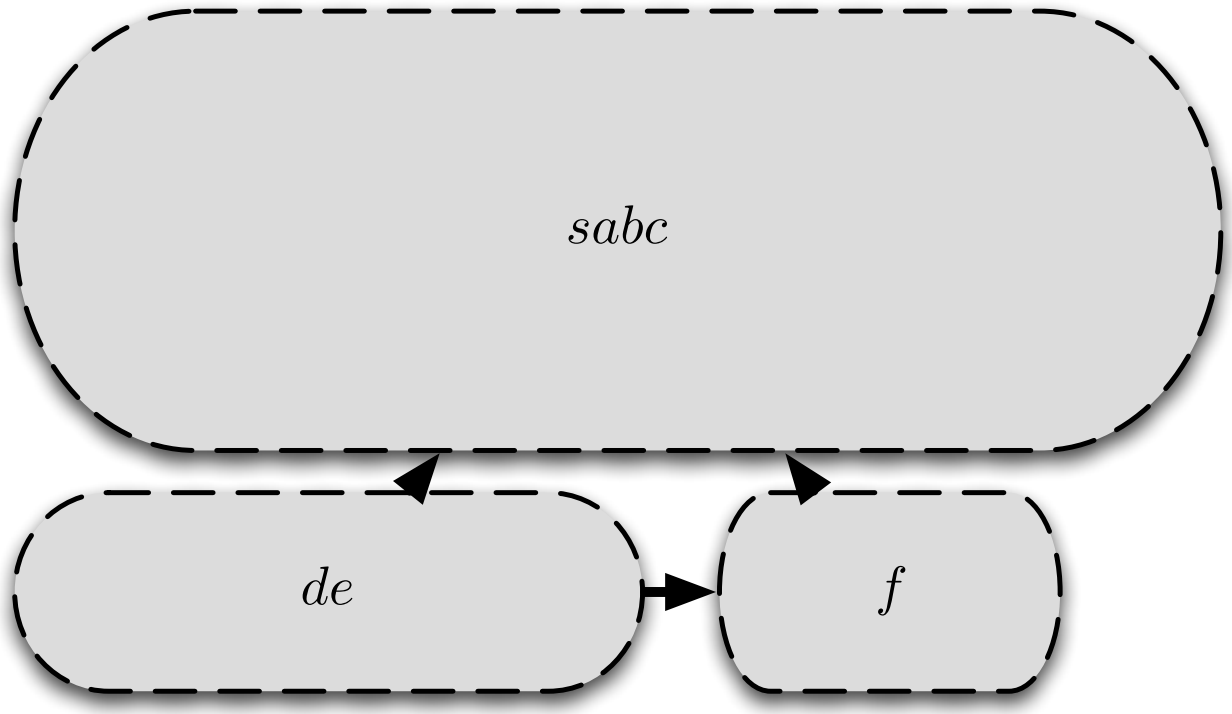
Anmerkung

Der Komponentengraph lässt sich aus den SCCs ermitteln.

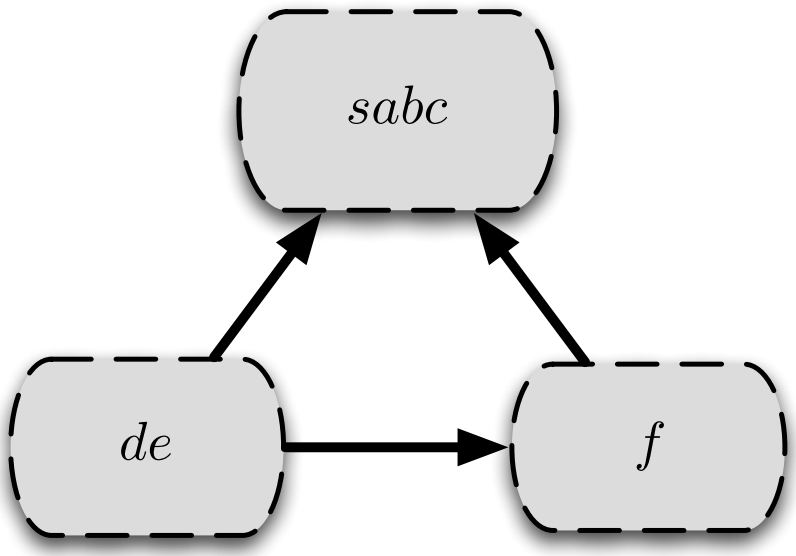
Der Komponentengraph - Beispiel



Der Komponentengraph - Beispiel



Der Komponentengraph - Beispiel



SCC - Algorithmus und Idee

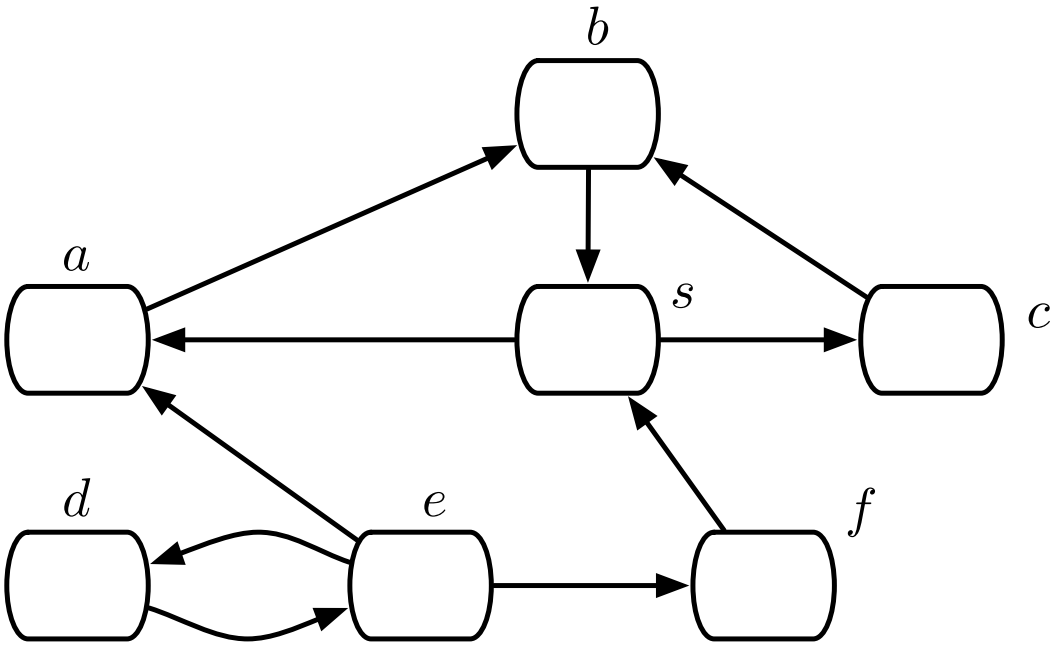
Algorithmus 2 SCC(G)

1: Aufruf von DFS(G) zur Berechnung der Endzeiten $f[v]$.
2: Berechne G^T (transponierter Graph).
3: Aufruf von DFS(G^T), betrachte in der Hauptschleife von DFS die Knoten jedoch in der Reihenfolge fallender $f[v]$ (in Zeile 1 berechnet).
4: Die Knoten jedes in Zeile 3 berechneten Baumes sind eine (separate) strenge Zusammenhangskomponente.

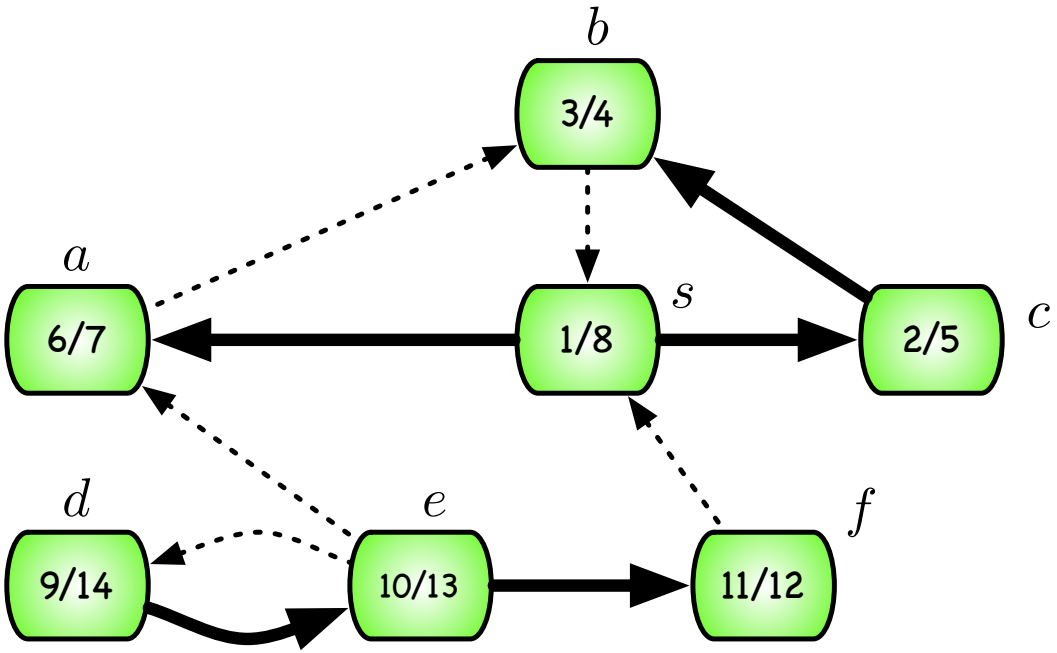
Anmerkung

Der zu einem Graphen $G = (V, E)$ transponierte Graph ist definiert durch $G^T = (V, E^T)$ mit $E^T = \{(u, v) \mid (v, u) \in E\}$. Ist G in Adjazenzlisten-Darstellung gegeben, kann G^T in Zeit $O(V + E)$ berechnet werden.

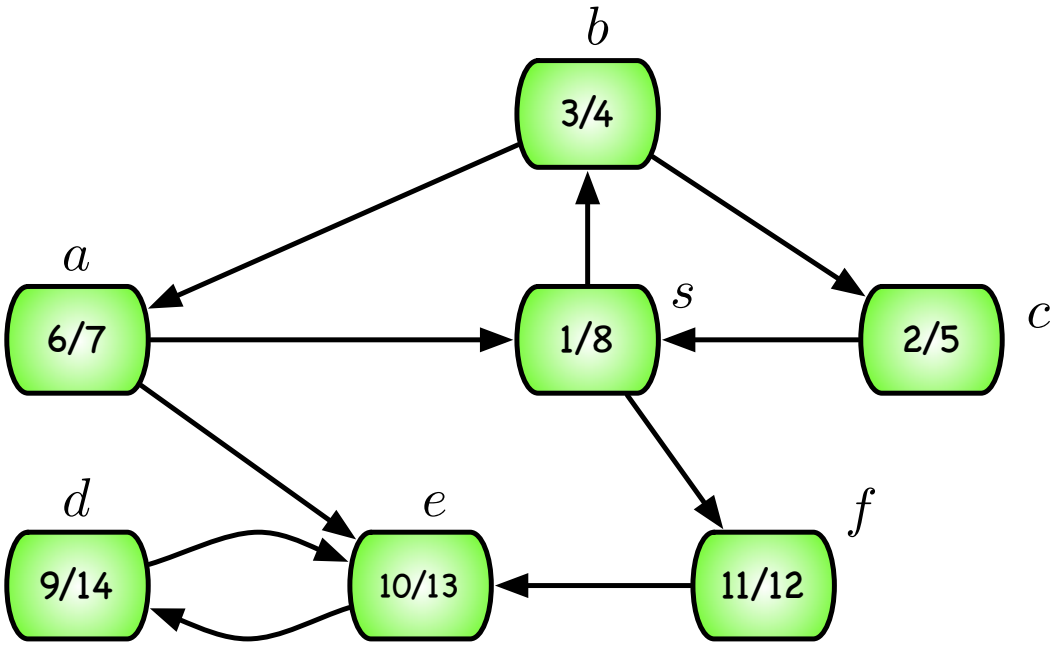
SCC - Beispiel



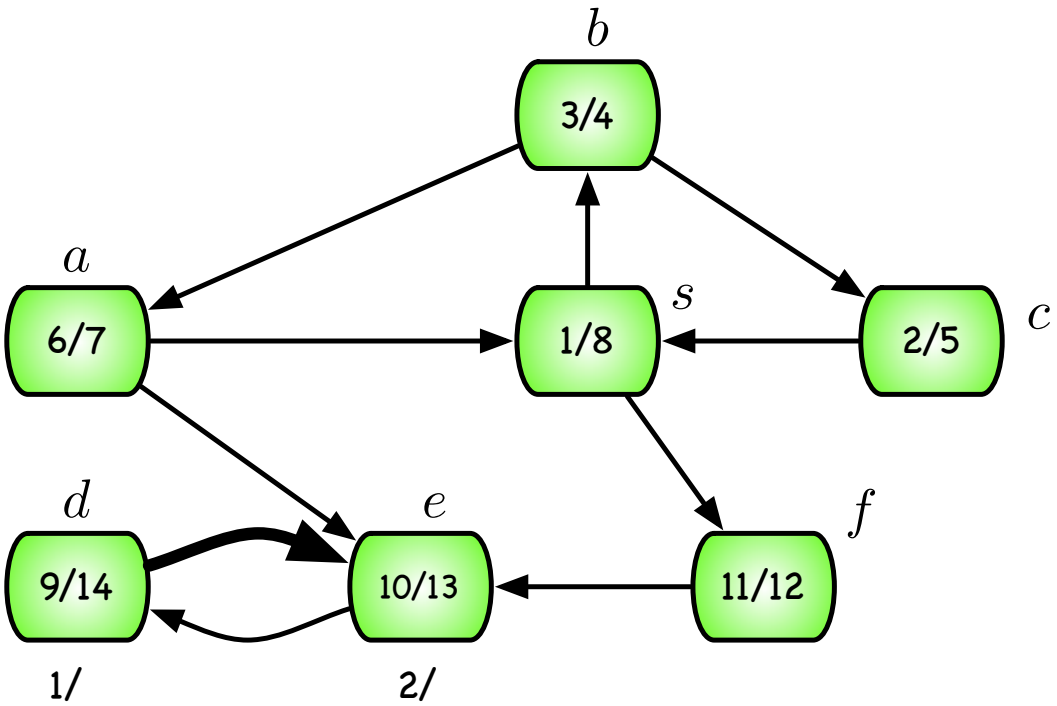
SCC - Beispiel



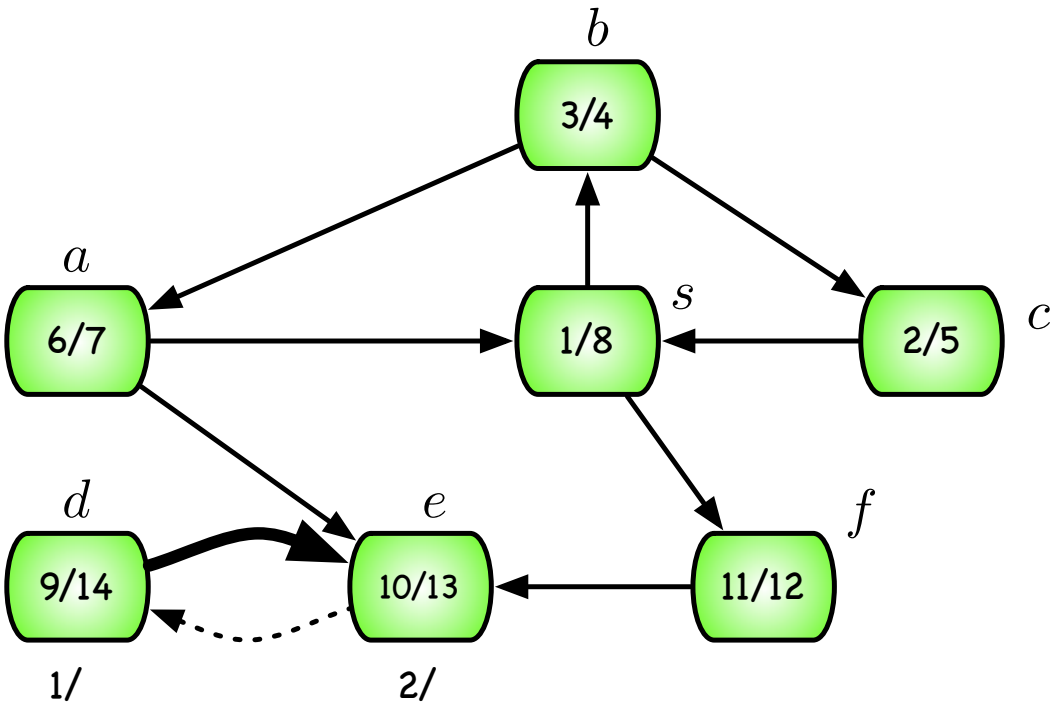
SCC - Beispiel



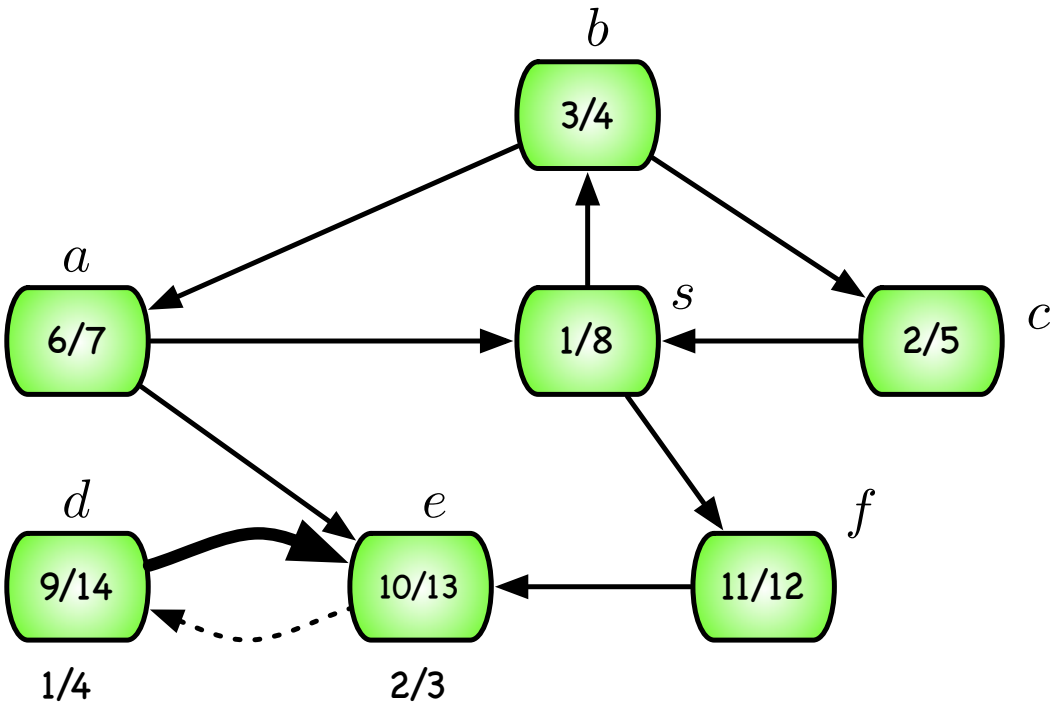
SCC - Beispiel



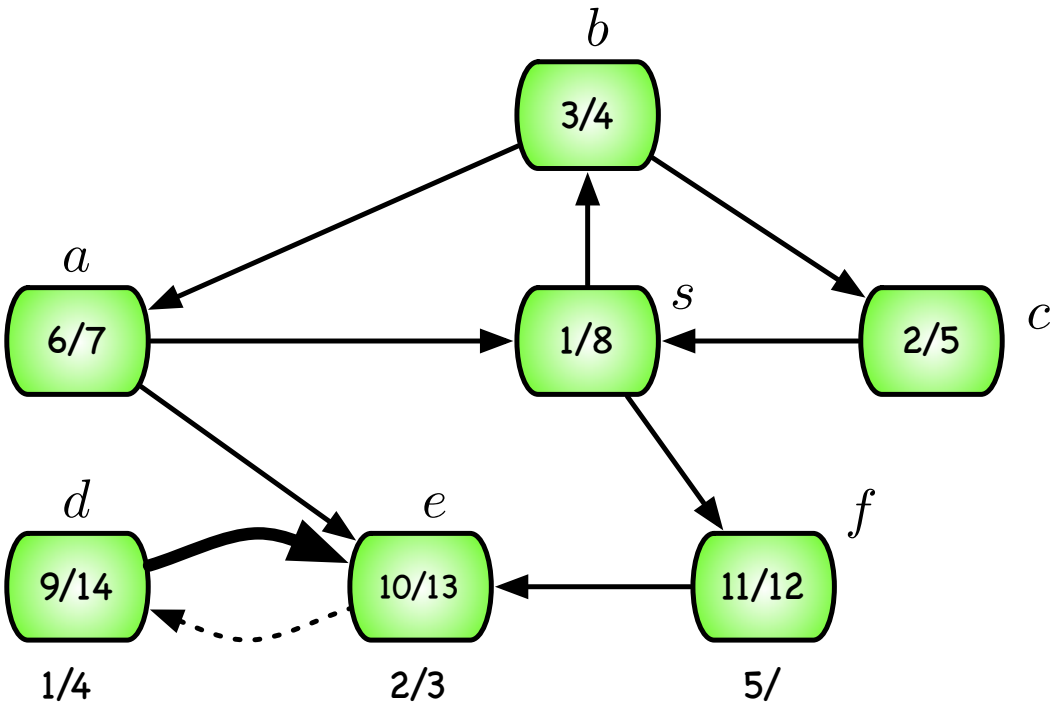
SCC - Beispiel



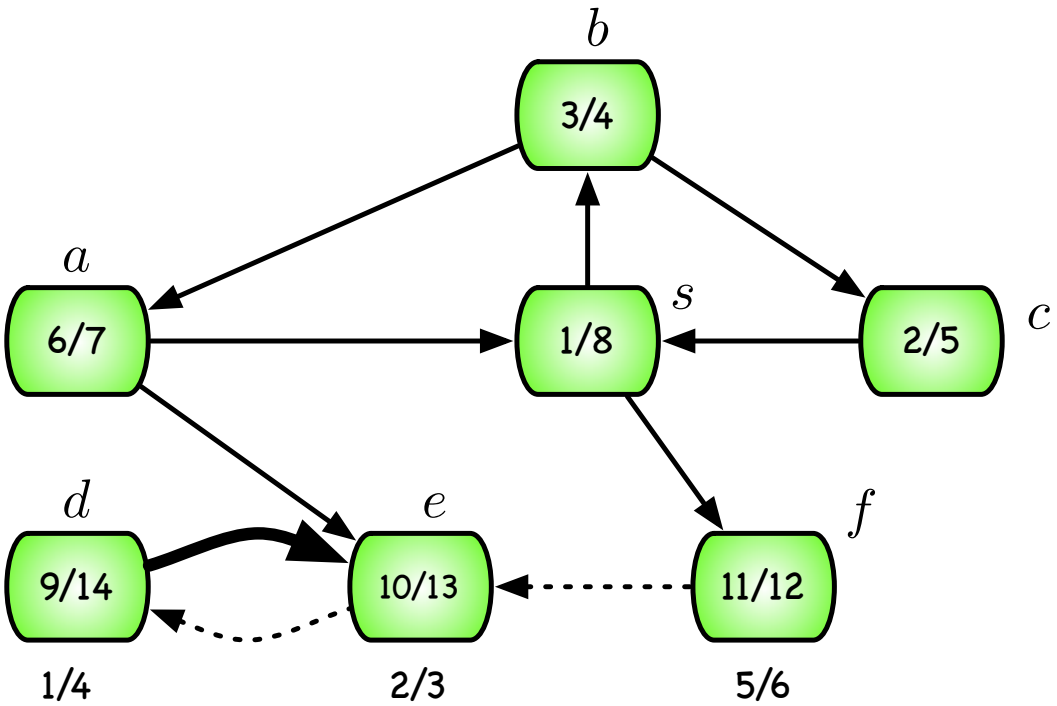
SCC - Beispiel



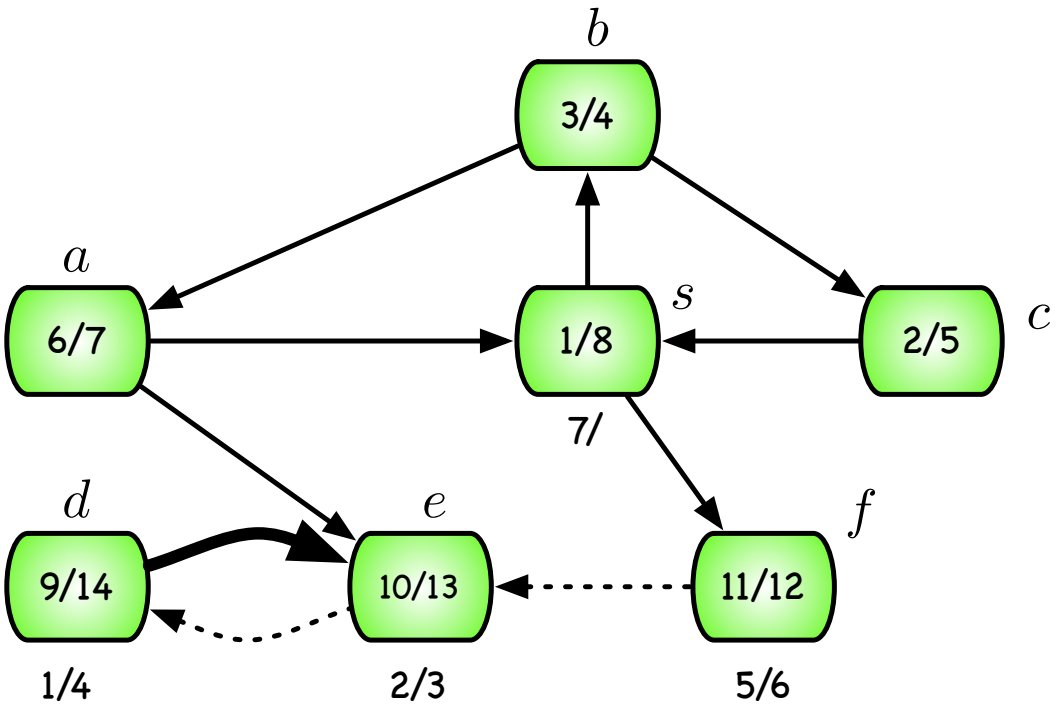
SCC - Beispiel



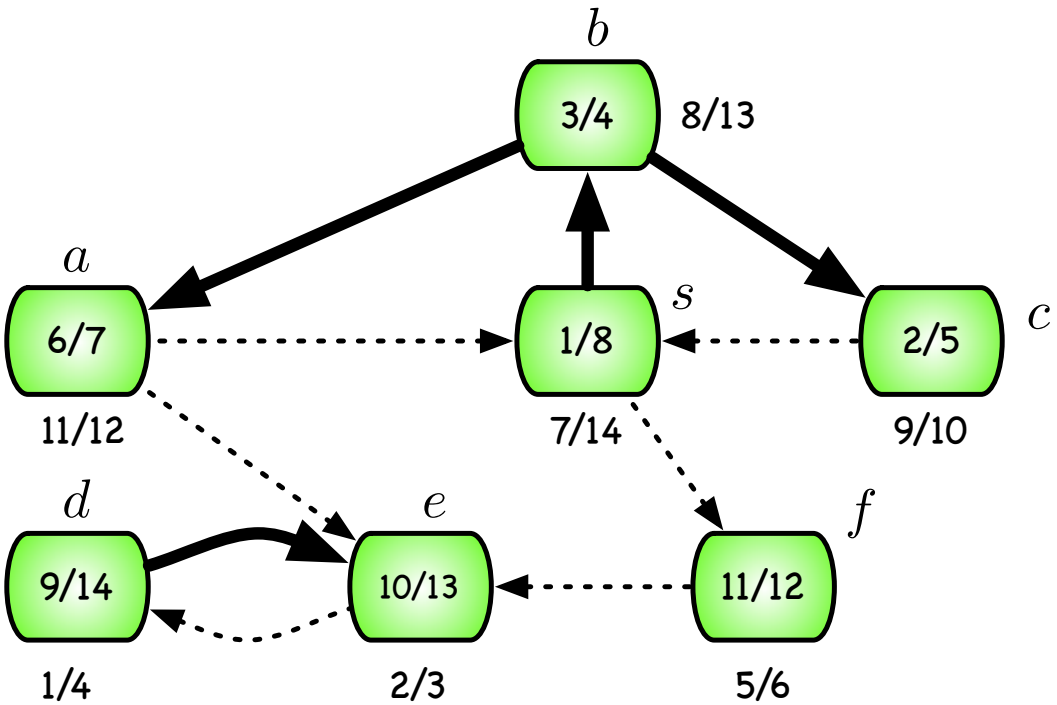
SCC - Beispiel



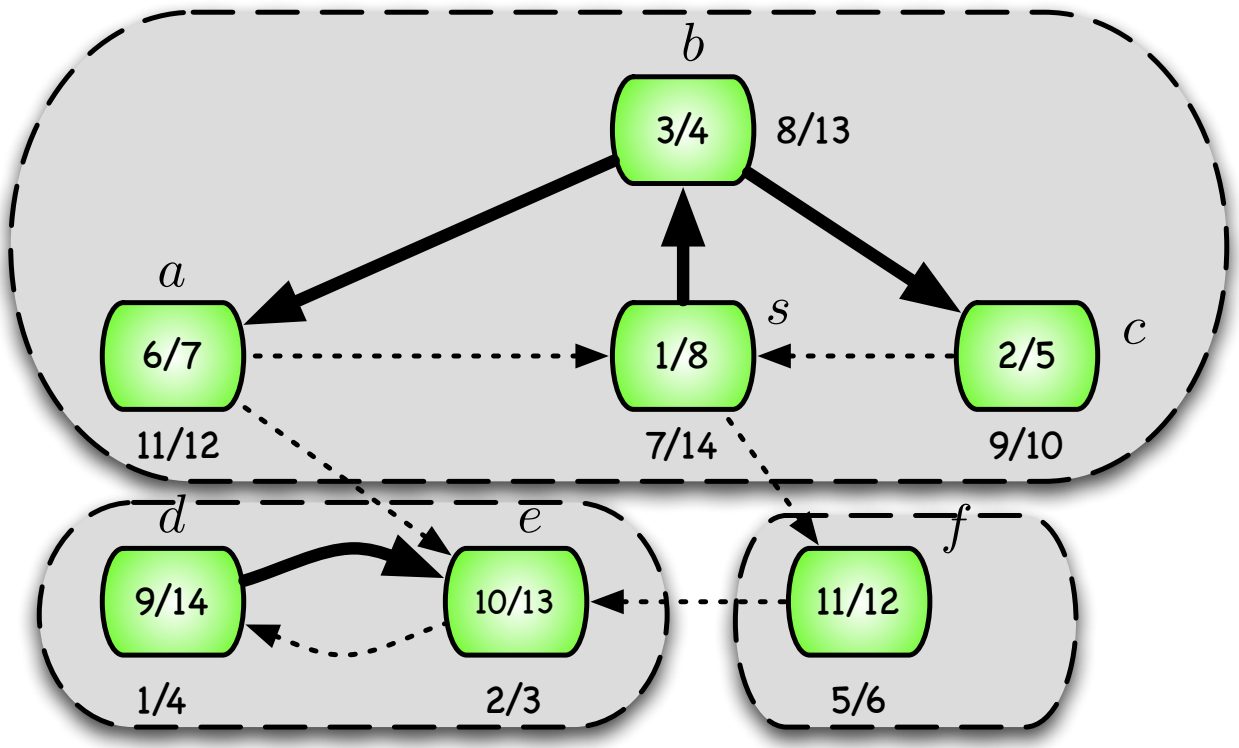
SCC - Beispiel



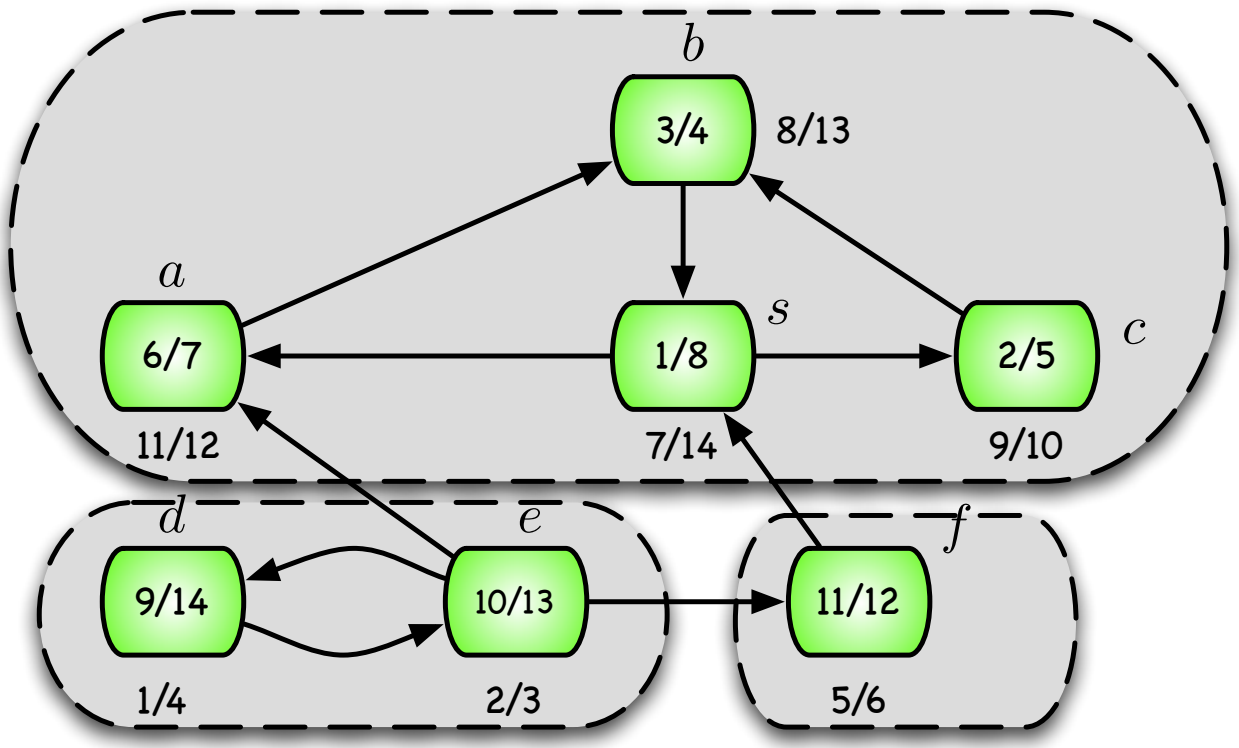
SCC - Beispiel



SCC - Beispiel



SCC - Beispiel



SCC - Analyse

Satz

- *G und G^T haben die gleichen SCCs.*
- *Seien C und C' zwei verschiedene SCCs eines DAG $G = (V, E)$.*
  - *Seien $u, v \in C$ und $u', v' \in C'$. Gibt es in G einen Pfad $u \Rightarrow u'$, so gibt es keinen Pfad $v' \Rightarrow v$ in G.*
  - *Gibt es $(u, v) \in E$ mit $u \in C$ und $v \in C'$, so gilt $f(C) > f(C')$.*
  - *Gibt es $(u, v) \in E^T$ mit $u \in C$ und $v \in C'$, so gilt $f(C) < f(C')$.*
- $\texttt{SCC}(G)$ *ist korrekt und hat eine Laufzeit in $\Theta(V + E)$.*

Für Interessierte

Interessierte Leser finden die Beweise in Kapitel 22 im [Cormen] (ca. 3 Seiten)

Zusammenfassung und Ausblick

- Graphen
  - Grundlagen, Adjazenzmatrix, Adjazenzliste
  - Breiten- & Tiefensuche
  - Anwendung der Tiefensuche
    - Topologisches Sortieren
    - Bestimmung starker Zusammenhangskomponenten
- (Über-)Nächstes Mal:
  - Minimale Spannbäume
  - Bestimmen der kürzesten Pfade
  - Flüsse

Topologisches Sortieren - Problemstellung

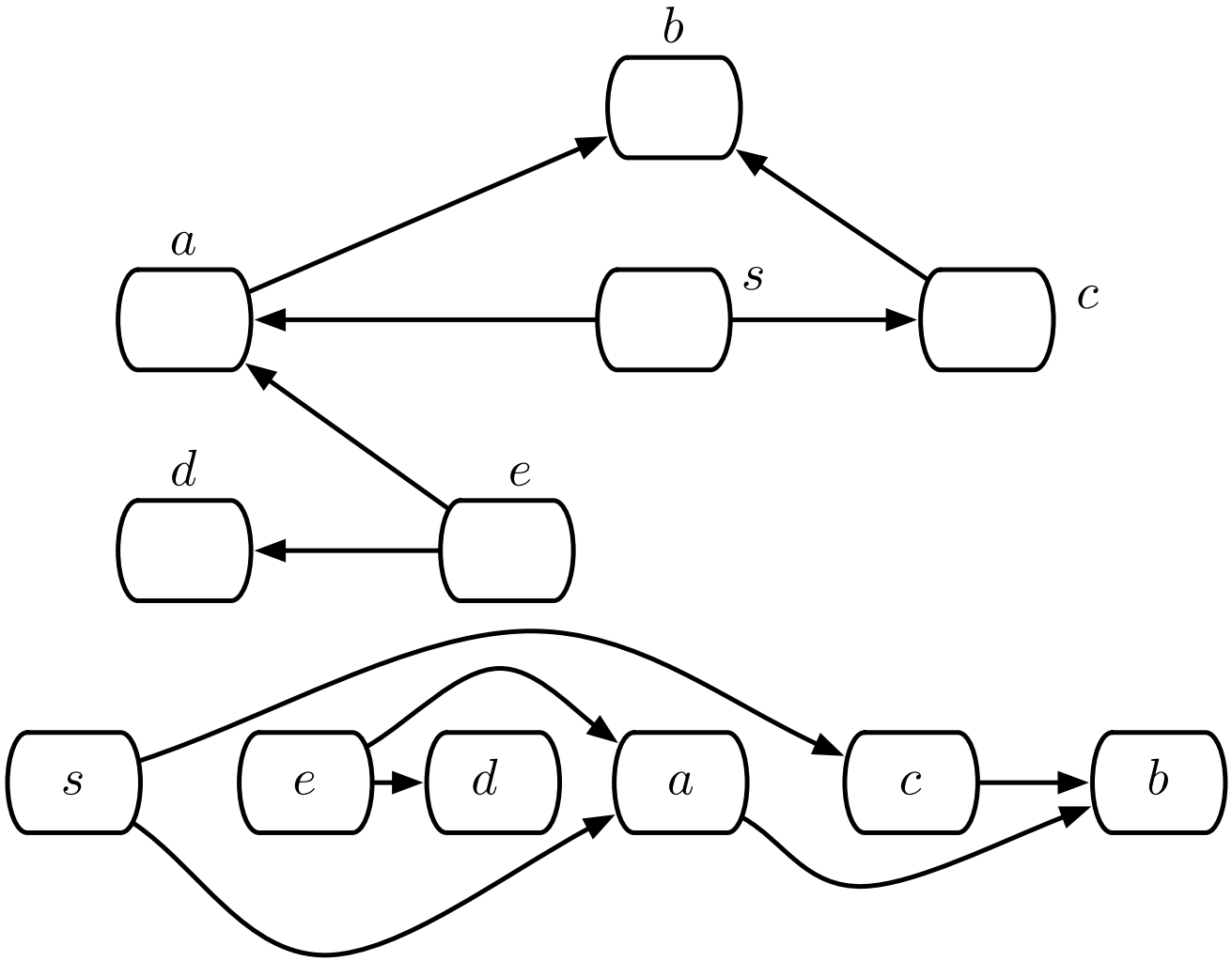
Definition (Topologisches Sortieren)

Eingabe: Gegeben ein gerichteter azyklischer Graph $G = (V, E)$.
Gesucht: Eine *topologischer Sortierung*, d.h. eine lineare Anordnung der Knoten mit der Eigenschaft, dass u in der Anordnung vor v liegt, falls es in G eine Kante (u, v) gibt.

Anmerkung

Dieses Problem tritt häufig bei der Modellierung von Ereignissen auf, die mit einer Priorität (bzw. Abhängigkeiten) versehen sind (und entsprechend bearbeitet werden sollen).

Ein Beispiel



TopoSort - Algorithmus und Idee

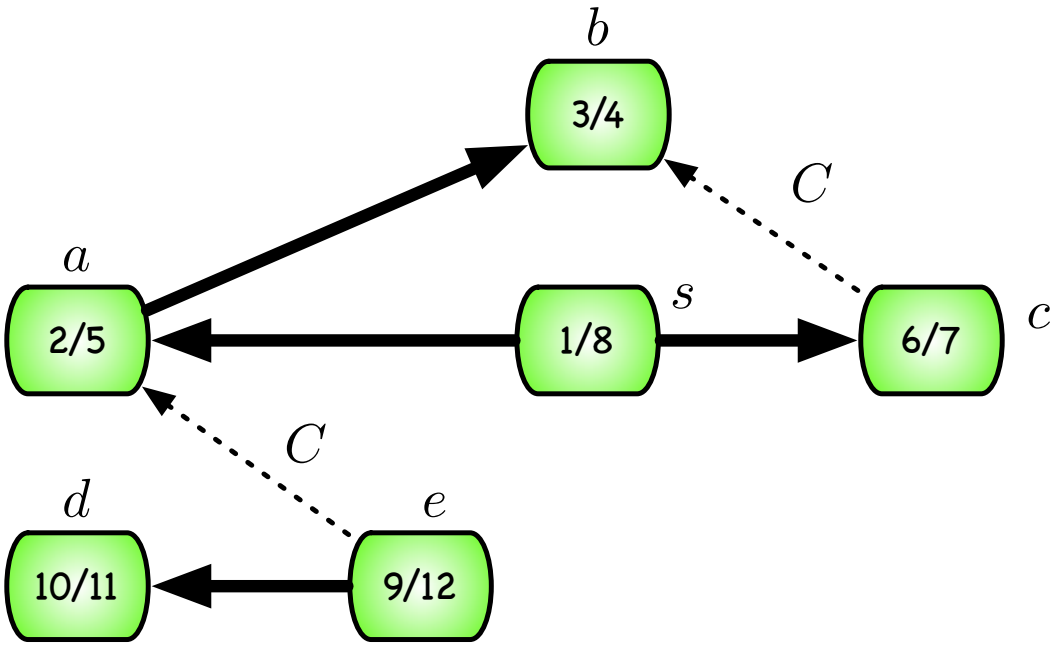
Algorithmus 3 TopoSort(G)

1: Berechne die Endzeiten $f[v]$ für alle $v \in V$ durch Aufruf von DFS(G).
2: Füge jeden abgearbeiteten Knoten an den Kopf einer verketteten Liste ein.
3: Liefere die verkettete Liste zurück.

Satz

TopoSort(G) *ist korrekt und arbeitet in* $\Theta(V + E)$.

TopoSort - Beispiel



e, d, s, c, a, b

Problemstellung

Definition (Bestimmung starker Zusammenhangskomponenten)

Eingabe: Gegeben ein gerichteter Graph $G = (V, E)$.
Gesucht: Die starken Zusammenhangskomponenten des Graphen, d.h. maximale Menge $C \subseteq V$ derart, dass für jedes Paar $u, v \in C$ sowohl $u \Rightarrow v$ als auch $v \Rightarrow u$ gilt (es gibt einen Pfad von u nach v und andersherum).

Anmerkung

Dieses Problem tritt oft bei gerichteten Graphen auf. Zunächst wird der Graph in seine starken Zusammenhangskomponenten zerlegt und dann werden diese separat betrachtet. (Oft werden die Lösungen anschließend noch entsprechend der Verbindungen zwischen den einzelnen Komponenten zusammengefügt.)

SCC - Algorithmus und Idee

Algorithmus 4 SCC(G)

1: Aufruf von DFS(G) zur Berechnung der Endzeiten $f[v]$.
2: Berechne G^T (transponierter Graph).
3: Aufruf von DFS(G^T), betrachte in der Hauptschleife von DFS die Knoten jedoch in der Reihenfolge fallender $f[v]$ (in Zeile 1 berechnet).
4: Die Knoten jedes in Zeile 3 berechneten Baumes sind eine (separate) strenge Zusammenhangskomponente.

Satz

SCC(G) *ist korrekt und arbeitet in* $\Theta(V + E)$.