

Java

```
public void addNode(V u) {  
    nodes.add(u);  
    edges.put(u, new LinkedList<V>());  
}
```

Java

```
public void addEdge(V s, V t) {  
    List<V> adjlist = edges.get(s);  
    adjlist.add(t);  
    if(!directed) {  
        adjlist = edges.get(t);  
        adjlist.add(s);  
    }  
}
```

Java

```
public void delEdge(V s, V t) {  
    List<V> adjlist = edges.get(s);  
    adjlist.remove(t);  
    if(!directed) {  
        adjlist = edges.get(t);  
        adjlist.remove(s);  
    }  
}
```

Übersicht

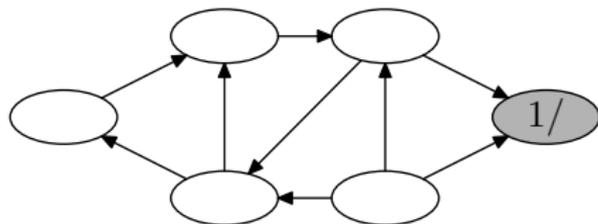
- 3 Graphalgorithmen
 - Darstellung von Graphen
 - **Tiefensuche**
 - Starke Komponenten
 - Topologisches Sortieren
 - Kürzeste Pfade
 - Netzwerkalgorithmen
 - Minimale Spannbäume

Tiefensuche

Tiefensuche ist ein sehr mächtiges Verfahren, das iterativ alle Knoten eines gerichteten oder ungerichteten Graphen besucht.

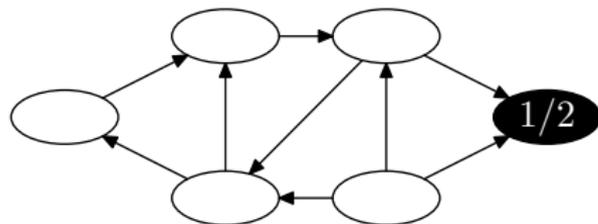
- Sie startet bei einem gegebenen Knoten und färbt die Knoten mit den Farben weiß, grau und schwarz.
- Sie berechnet einen gerichteten **Tiefensuchwald**, der bei einem ungerichteten Graph ein Baum ist.
- Sie ordnet jedem Knoten eine Anfangs- und eine Endzeit zu.
- Alle Zeiten sind verschieden.
- Die Kanten des Graphen werden als **Baum-, Vorwärts-, Rückwärts- oder Querkanten** klassifiziert.

Tiefensuche – Beispiel



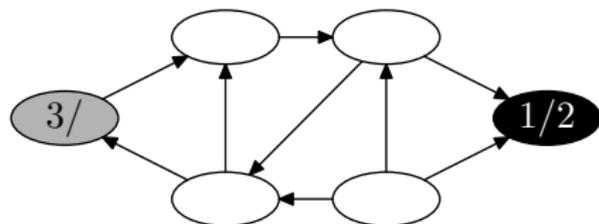
- Die Kanten des Tiefensuchwales sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



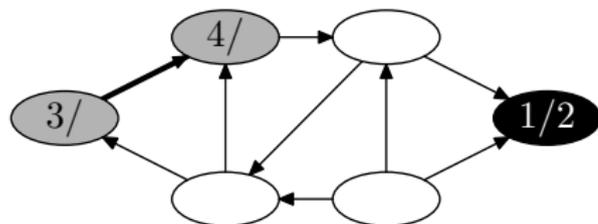
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



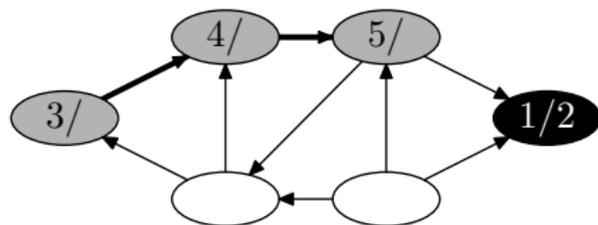
- Die Kanten des Tiefensuchwales sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



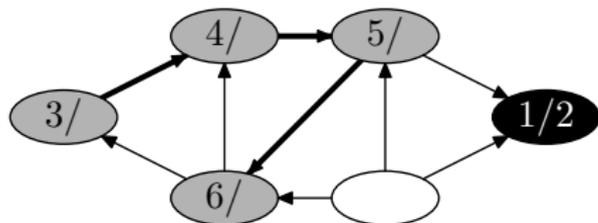
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



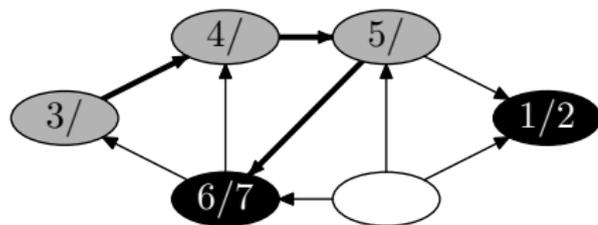
- Die Kanten des Tiefensuchwandes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



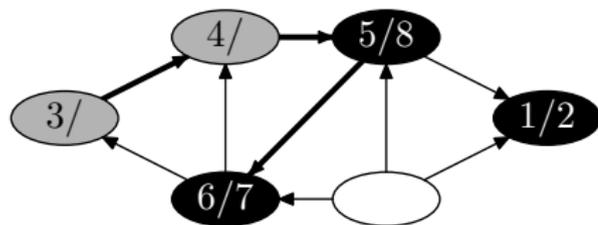
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



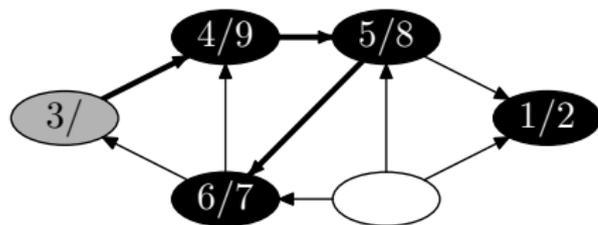
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



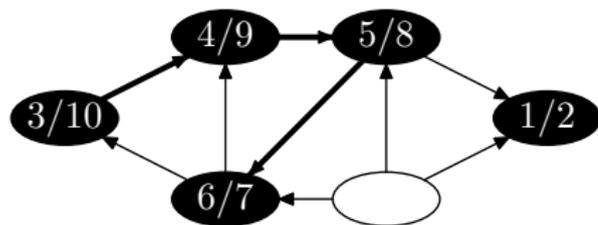
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



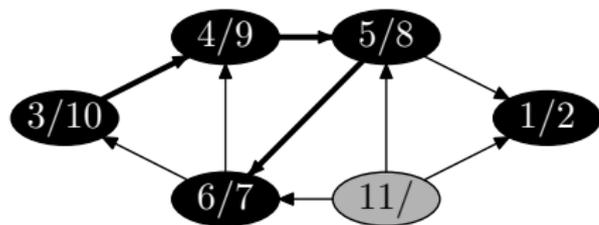
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



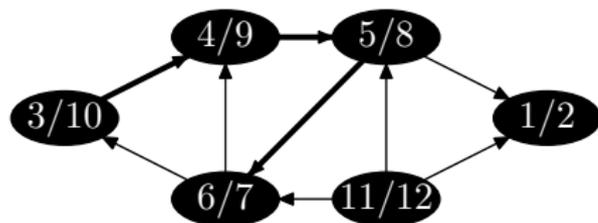
- Die Kanten des Tiefensuchwales sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Noch ein Beispiel



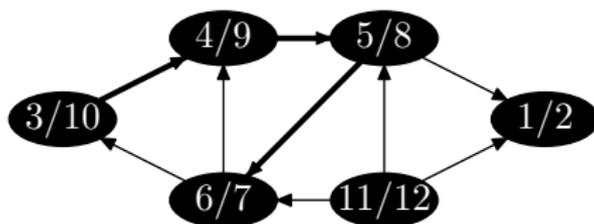
Java

```
public void DFS(Map<V, Integer> d,  
    Map<V, Integer> f, Map<V, V> p) {  
    Map<V, Integer> color = new HashMap<V, Integer>();  
    for(V u : allNodes())  
        color.put(u, WHITE);  
    int time = 0;  
    for(V u : allNodes())  
        if(color.get(u) == WHITE)  
            time = DFS(u, time, color, d, f, p);  
}
```

Java

```
public int DFS(V u, int t, Map<V, Integer> c,  
    Map<V, Integer> d, Map<V, Integer> f, Map<V, V> p) {  
    d.put(u, ++t);  
    c.put(u, GRAY);  
    for(V v : neighbors(u))  
        if(c.get(v) == WHITE) {  
            p.put(v, u);  
            t = DFS(v, t, c, d, f, p);  
        }  
    f.put(u, ++t);  
    c.put(u, BLACK);  
    return t;  
}
```

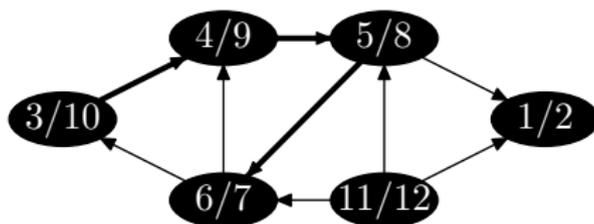
Taxonomie der Kanten



- 1 Eine **Baumkante** ist im DFS-Wald (geht von einem Knoten zu einem seiner Kinder im DFS-Wald).
- 2 Eine **Vorwärtskante** geht von einem Knoten zu einem seiner Nachfahren im DFS-Wald (aber nicht Kind).
- 3 Eine **Rückwärtskante** geht von einem Knoten zu einem seiner Vorfahren im DFS-Wald.
- 4 Eine **Querkante** verbindet zwei im DFS-Wald unvergleichbare Knoten.

Frage: Welchen Typ hat jede Kante in diesem Beispiel?

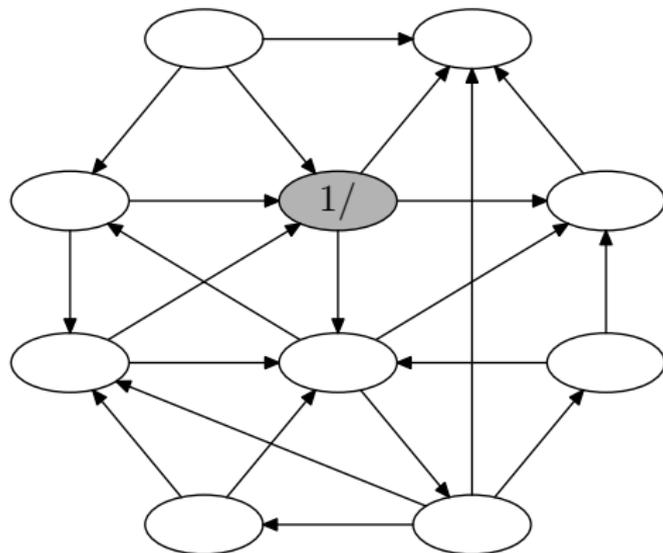
Taxonomie der Kanten



- 1 Eine **Baumkante** ist im DFS-Wald (geht von einem Knoten zu einem seiner Kinder im DFS-Wald).
- 2 Eine **Vorwärtskante** geht von einem Knoten zu einem seiner Nachfahren im DFS-Wald (aber nicht Kind).
- 3 Eine **Rückwärtskante** geht von einem Knoten zu einem seiner Vorfahren im DFS-Wald.
- 4 Eine **Querkante** verbindet zwei im DFS-Wald unvergleichbare Knoten.

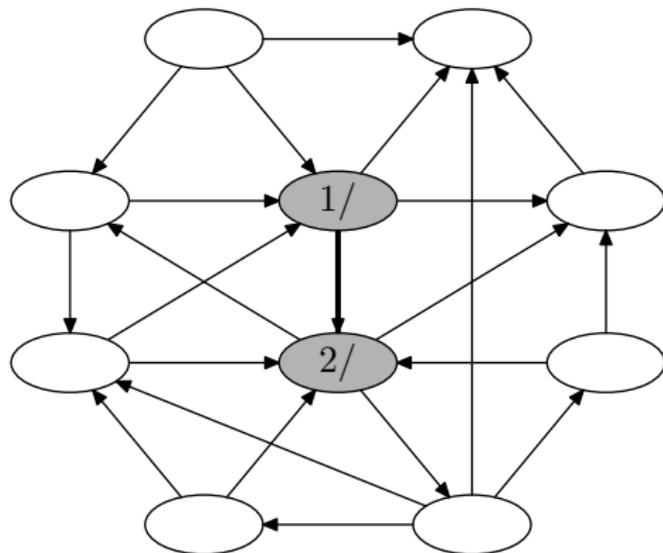
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



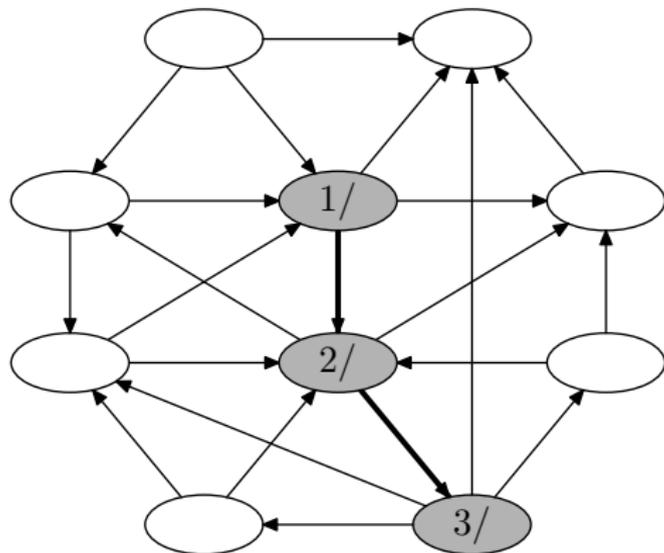
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



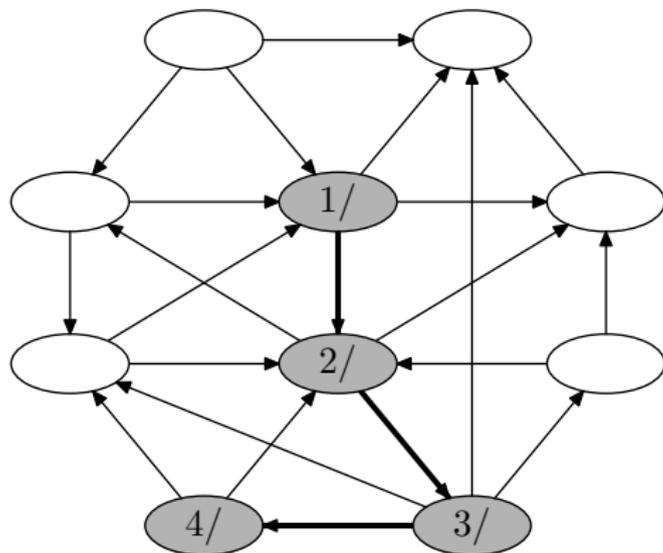
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



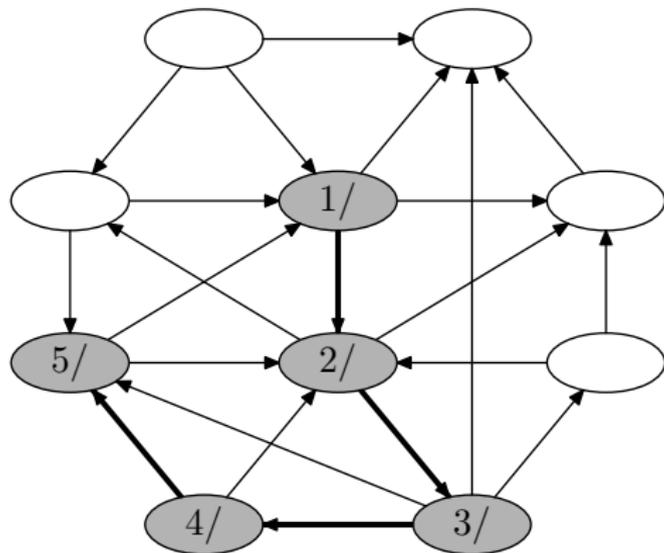
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



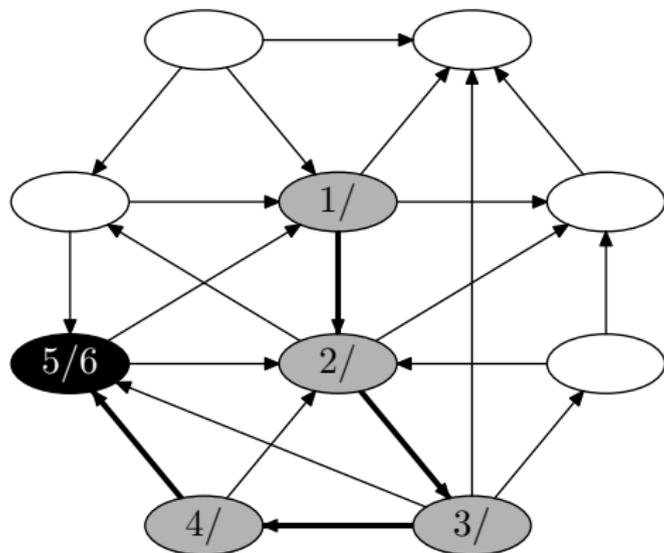
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



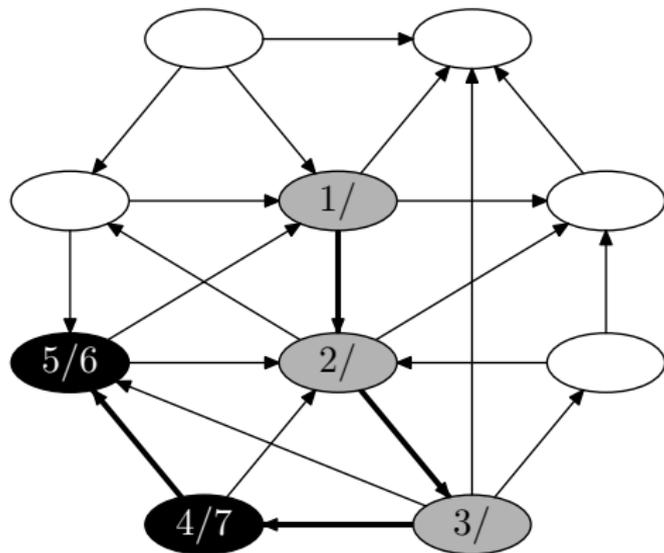
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



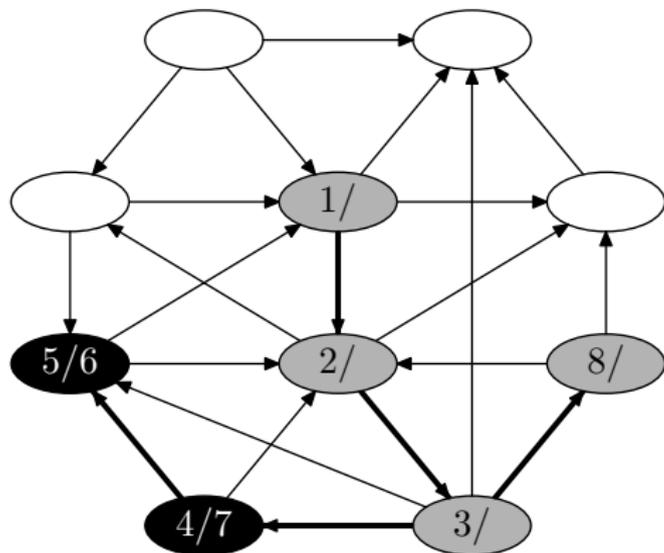
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



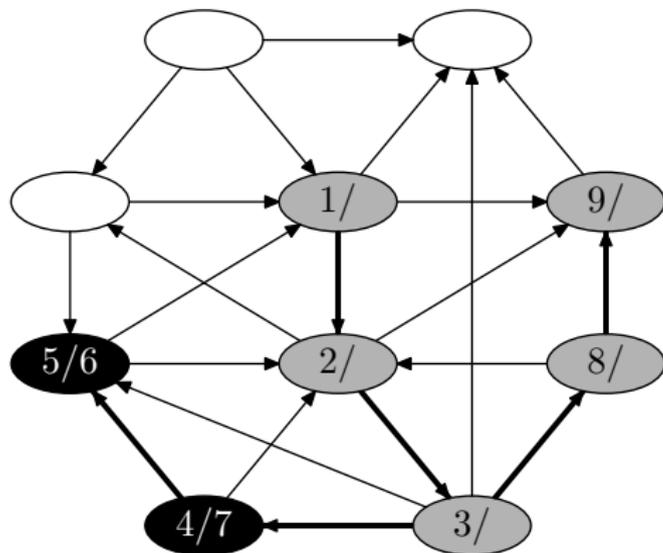
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



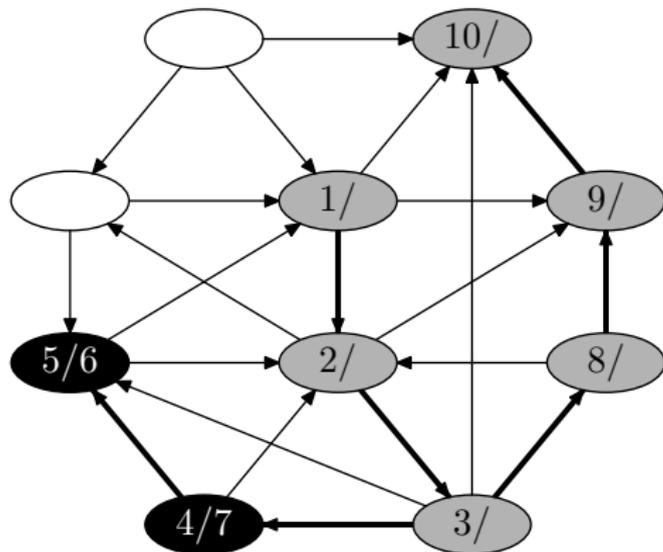
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



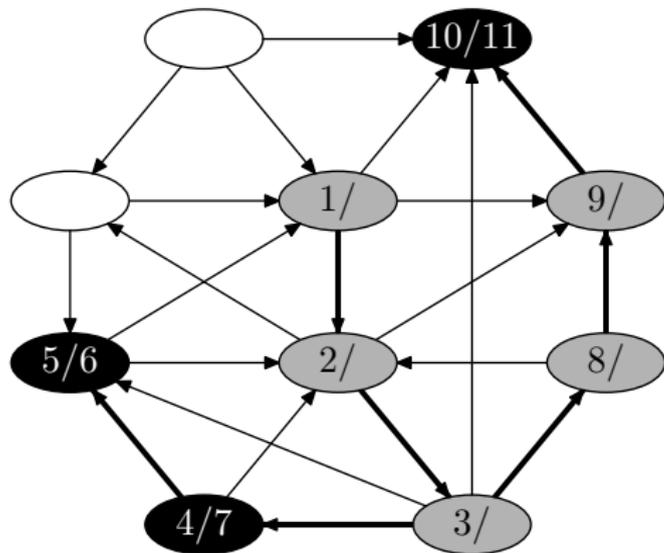
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



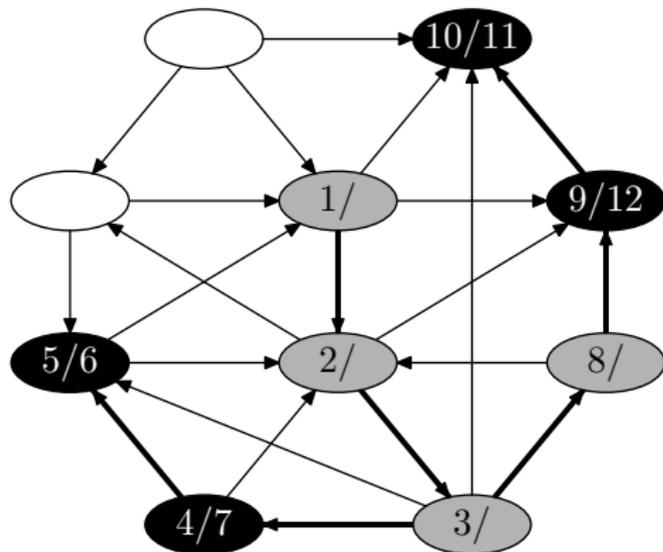
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



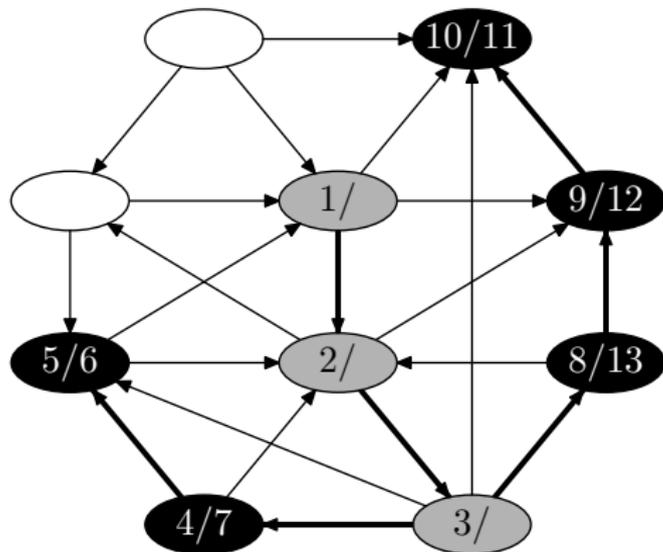
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



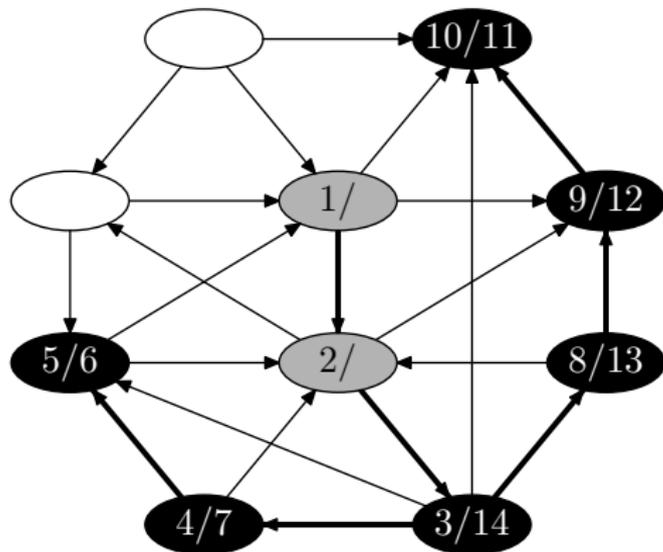
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



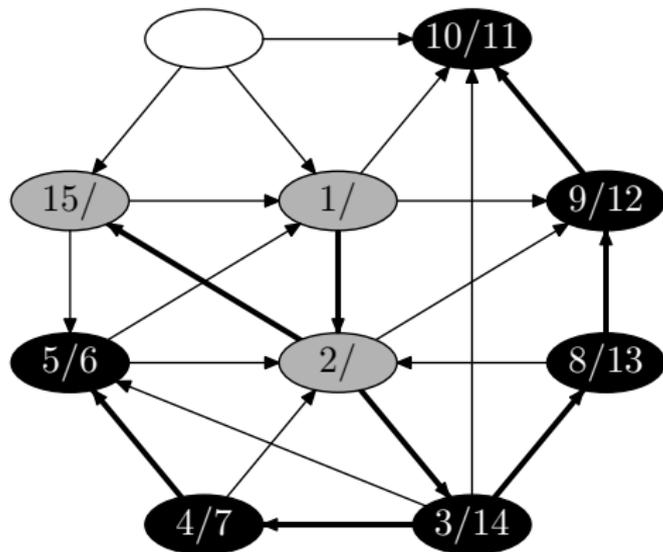
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



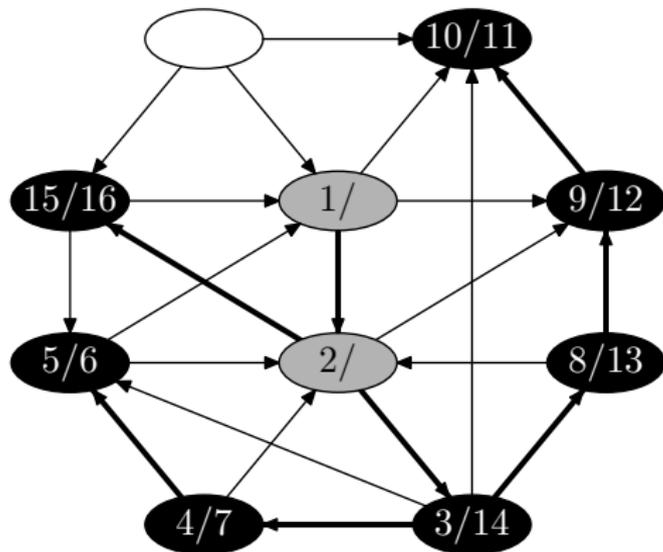
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



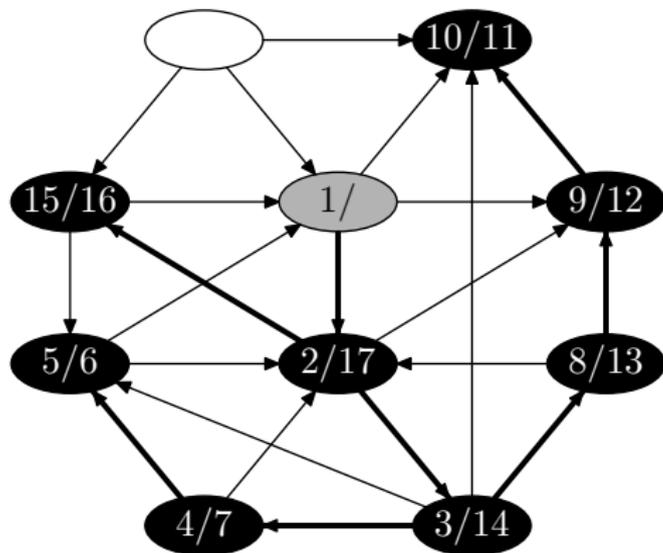
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



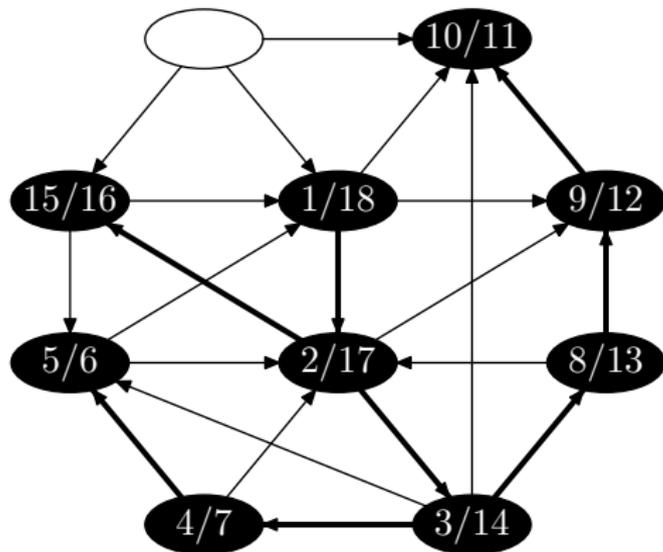
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



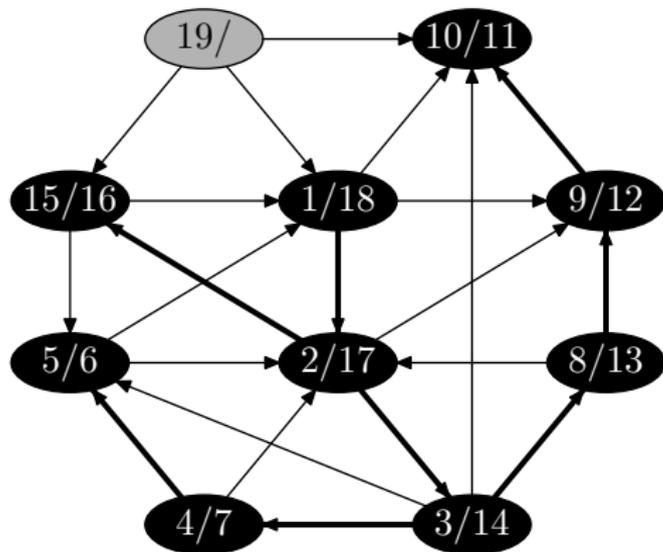
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



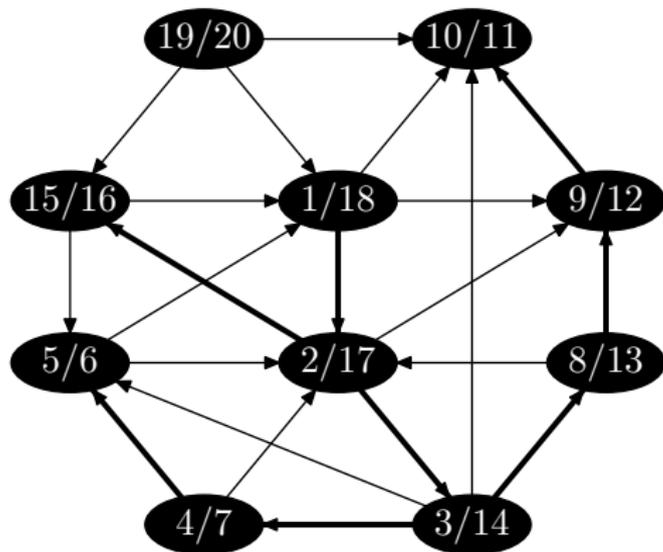
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



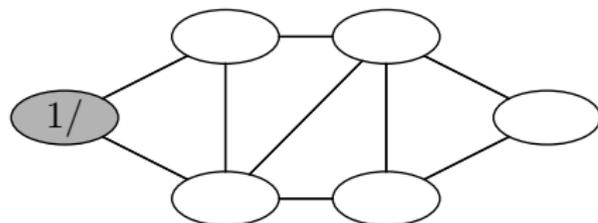
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



Frage: Welchen Typ hat jede Kante in diesem Beispiel?

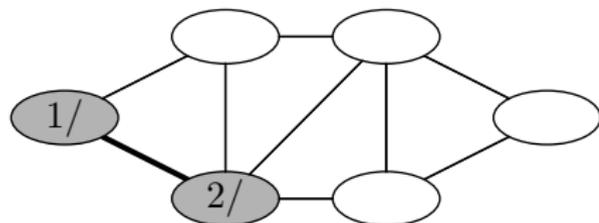
DFS – Ungerichtete Graphen



Wir erhalten immer einen Baum, wenn der Graph zusammenhängend ist.

Implementierung: Eine ungerichtete Kante wird durch Kanten in beide Richtungen dargestellt.

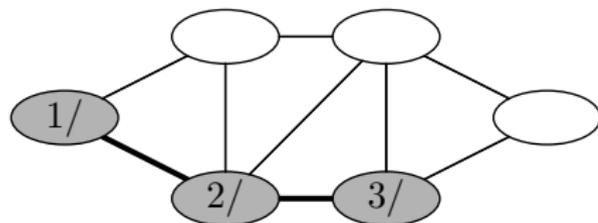
DFS – Ungerichtete Graphen



Wir erhalten immer einen Baum, wenn der Graph zusammenhängend ist.

Implementierung: Eine ungerichtete Kante wird durch Kanten in beide Richtungen dargestellt.

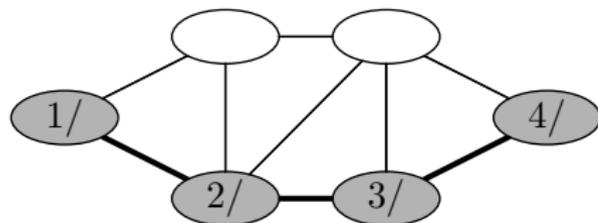
DFS – Ungerichtete Graphen



Wir erhalten immer einen Baum, wenn der Graph zusammenhängend ist.

Implementierung: Eine ungerichtete Kante wird durch Kanten in beide Richtungen dargestellt.

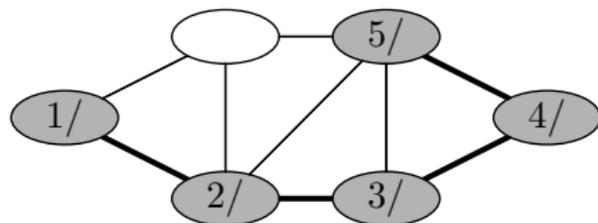
DFS – Ungerichtete Graphen



Wir erhalten immer einen Baum, wenn der Graph zusammenhängend ist.

Implementierung: Eine ungerichtete Kante wird durch Kanten in beide Richtungen dargestellt.

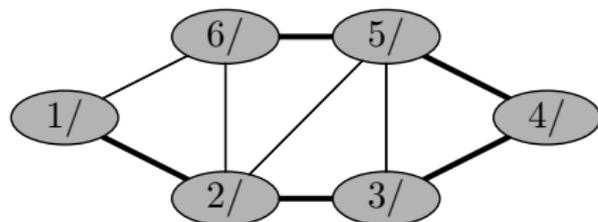
DFS – Ungerichtete Graphen



Wir erhalten immer einen Baum, wenn der Graph zusammenhängend ist.

Implementierung: Eine ungerichtete Kante wird durch Kanten in beide Richtungen dargestellt.

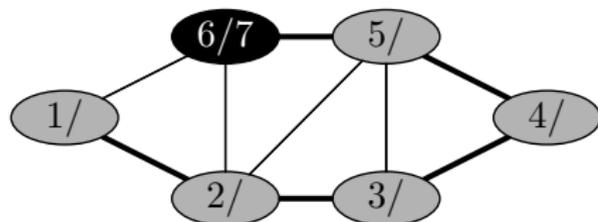
DFS – Ungerichtete Graphen



Wir erhalten immer einen Baum, wenn der Graph zusammenhängend ist.

Implementierung: Eine ungerichtete Kante wird durch Kanten in beide Richtungen dargestellt.

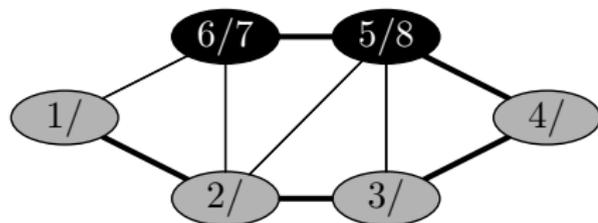
DFS – Ungerichtete Graphen



Wir erhalten immer einen Baum, wenn der Graph zusammenhängend ist.

Implementierung: Eine ungerichtete Kante wird durch Kanten in beide Richtungen dargestellt.

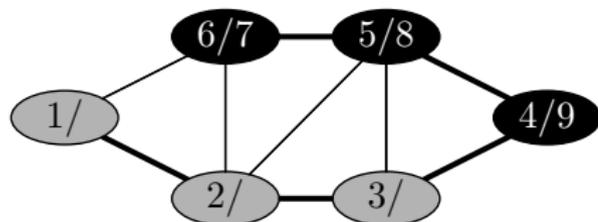
DFS – Ungerichtete Graphen



Wir erhalten immer einen Baum, wenn der Graph zusammenhängend ist.

Implementierung: Eine ungerichtete Kante wird durch Kanten in beide Richtungen dargestellt.

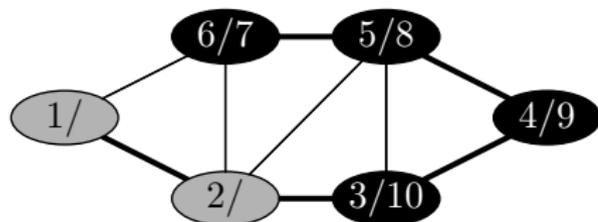
DFS – Ungerichtete Graphen



Wir erhalten immer einen Baum, wenn der Graph zusammenhängend ist.

Implementierung: Eine ungerichtete Kante wird durch Kanten in beide Richtungen dargestellt.

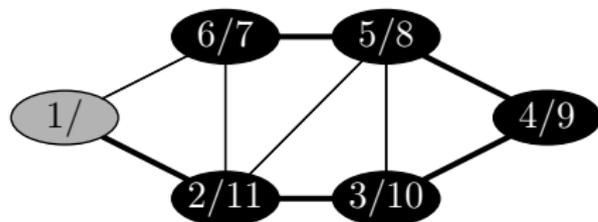
DFS – Ungerichtete Graphen



Wir erhalten immer einen Baum, wenn der Graph zusammenhängend ist.

Implementierung: Eine ungerichtete Kante wird durch Kanten in beide Richtungen dargestellt.

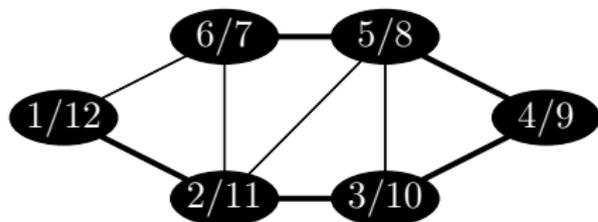
DFS – Ungerichtete Graphen



Wir erhalten immer einen Baum, wenn der Graph zusammenhängend ist.

Implementierung: Eine ungerichtete Kante wird durch Kanten in beide Richtungen dargestellt.

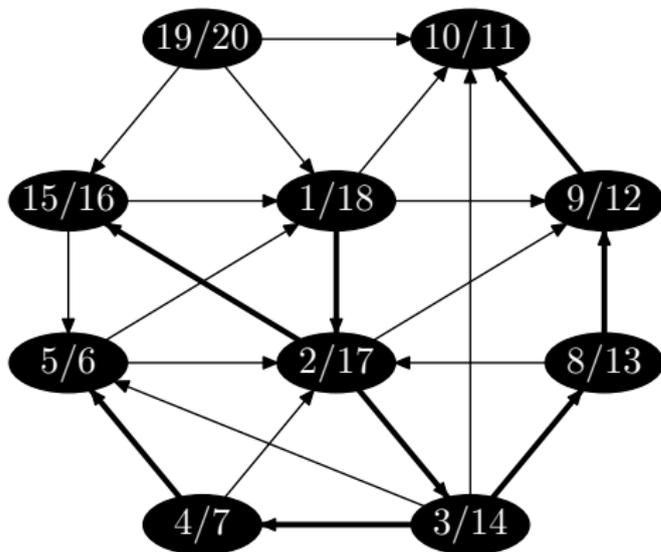
DFS – Ungerichtete Graphen



Wir erhalten immer einen Baum, wenn der Graph zusammenhängend ist.

Implementierung: Eine ungerichtete Kante wird durch Kanten in beide Richtungen dargestellt.

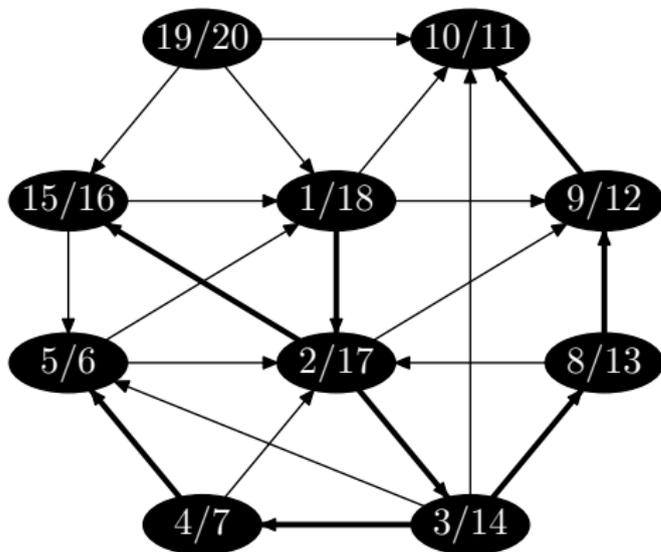
Taxonomie der Kanten



Betrachte Kante (u, v) :

- 1 $d(u) < d(v)$ und $f(v) < f(u) \iff$
Baum- oder Vorwärtskante
- 2 $d(v) < d(u)$ und $f(u) < f(v) \iff$
Rückwärtskante
- 3 sonst Querkante

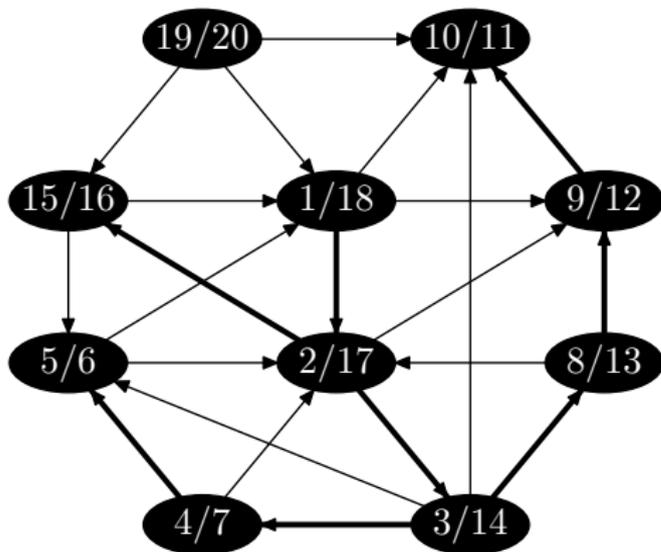
Taxonomie der Kanten



Betrachte Kante (u, v) :

- 1 $d(u) < d(v)$ und $f(v) < f(u) \iff$
Baum- oder
Vorwärtskante
- 2 $d(v) < d(u)$ und $f(u) < f(v) \iff$
Rückwärtskante
- 3 sonst Querkante

Taxonomie der Kanten



Betrachte Kante (u, v) :

- 1 $d(u) < d(v)$ und $f(v) < f(u) \iff$
Baum- oder
Vorwärtskante
- 2 $d(v) < d(u)$ und $f(u) < f(v) \iff$
Rückwärtskante
- 3 sonst Querkante

Tiefensuche

Theorem

Gegeben sei ein gerichteter Graph $G = (V, E)$ (in Adjazenzlistendarstellung).

Durch Tiefensuche kann ein DFS-Wald inklusive der Funktionen $d: V \rightarrow \mathbf{N}$, $f: V \rightarrow \mathbf{N}$ in $O(|V| + |E|)$ Schritten (also in linearer Zeit) berechnet werden.

Beweis.

(Skizze) Solange ein Knoten grau ist, wird jede inzidente Kante einmal besucht. Jeder Knoten wechselt seine Farbe nur zweimal, jedesmal mit konstantem Aufwand.

Jede Kante wird daher ebenfalls nur einmal besucht. □

Zusammenhangskomponenten

Definition

Es sei $G = (V, E)$ ein ungerichteter Graph. Ein **Pfad** der Länge k von u_1 nach u_{k+1} ist eine Folge $(u_1, u_2), (u_2, u_3), \dots, (u_k, u_{k+1})$ von Kanten aus E wobei u_1, \dots, u_{k+1} paarweise verschieden sind.

Wir sagen u und v sind **zusammenhängend**, wenn es einen Pfad von u nach v gibt.

Eine Menge $C \subseteq V$ ist eine **Zusammenhangskomponente**, wenn alle Knoten in C zusammenhängend sind und es keine echte Obermenge von C mit dieser Eigenschaft gibt.

(Alternativ: Die Zusammenhangskomponenten sind die Äquivalenzklassen der Relation „zusammenhängend“.)

Zusammenhangskomponenten

Definition

Es sei $G = (V, E)$ ein ungerichteter Graph. Ein **Pfad** der Länge k von u_1 nach u_{k+1} ist eine Folge $(u_1, u_2), (u_2, u_3), \dots, (u_k, u_{k+1})$ von Kanten aus E wobei u_1, \dots, u_{k+1} paarweise verschieden sind.

Wir sagen u und v sind **zusammenhängend**, wenn es einen Pfad von u nach v gibt.

Eine Menge $C \subseteq V$ ist eine **Zusammenhangskomponente**, wenn alle Knoten in C zusammenhängend sind und es keine echte Obermenge von C mit dieser Eigenschaft gibt.

(Alternativ: Die Zusammenhangskomponenten sind die Äquivalenzklassen der Relation „zusammenhängend“.)

Zusammenhangskomponenten

Definition

Es sei $G = (V, E)$ ein ungerichteter Graph. Ein **Pfad** der Länge k von u_1 nach u_{k+1} ist eine Folge $(u_1, u_2), (u_2, u_3), \dots, (u_k, u_{k+1})$ von Kanten aus E wobei u_1, \dots, u_{k+1} paarweise verschieden sind.

Wir sagen u und v sind **zusammenhängend**, wenn es einen Pfad von u nach v gibt.

Eine Menge $C \subseteq V$ ist eine **Zusammenhangskomponente**, wenn alle Knoten in C zusammenhängend sind und es keine echte Obermenge von C mit dieser Eigenschaft gibt.

(Alternativ: Die Zusammenhangskomponenten sind die Äquivalenzklassen der Relation „zusammenhängend“.)

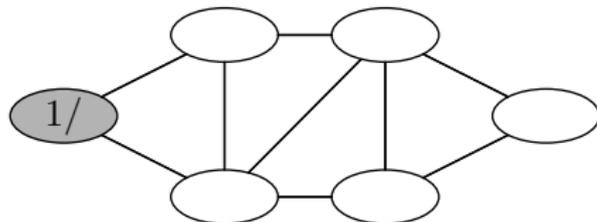
Zusammenhangskomponenten

Theorem

Die Zusammenhangskomponenten eines ungerichteten Graphen können in linearer Zeit gefunden werden.

Beweis.

Die Knoten, die jeweils in einem Aufruf der Tiefensuche schwarz werden, gehören zu einer Komponente. □



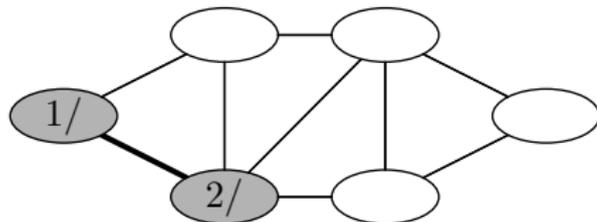
Zusammenhangskomponenten

Theorem

Die Zusammenhangskomponenten eines ungerichteten Graphen können in linearer Zeit gefunden werden.

Beweis.

Die Knoten, die jeweils in einem Aufruf der Tiefensuche schwarz werden, gehören zu einer Komponente. □



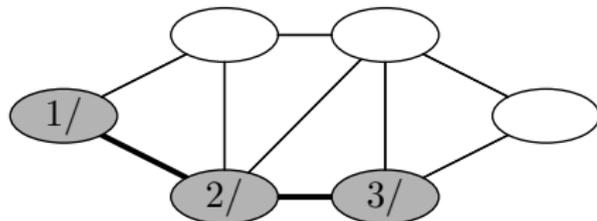
Zusammenhangskomponenten

Theorem

Die Zusammenhangskomponenten eines ungerichteten Graphen können in linearer Zeit gefunden werden.

Beweis.

Die Knoten, die jeweils in einem Aufruf der Tiefensuche schwarz werden, gehören zu einer Komponente. □



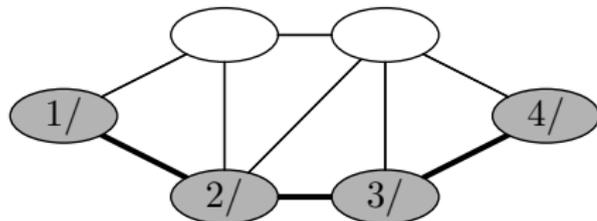
Zusammenhangskomponenten

Theorem

Die Zusammenhangskomponenten eines ungerichteten Graphen können in linearer Zeit gefunden werden.

Beweis.

Die Knoten, die jeweils in einem Aufruf der Tiefensuche schwarz werden, gehören zu einer Komponente. □



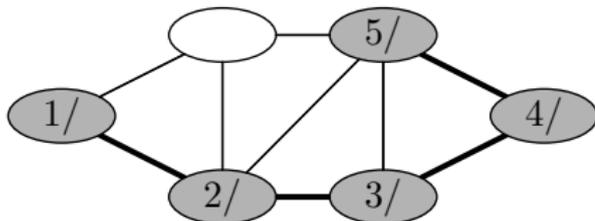
Zusammenhangskomponenten

Theorem

Die Zusammenhangskomponenten eines ungerichteten Graphen können in linearer Zeit gefunden werden.

Beweis.

Die Knoten, die jeweils in einem Aufruf der Tiefensuche schwarz werden, gehören zu einer Komponente. □



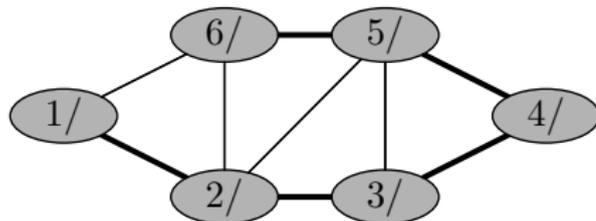
Zusammenhangskomponenten

Theorem

Die Zusammenhangskomponenten eines ungerichteten Graphen können in linearer Zeit gefunden werden.

Beweis.

Die Knoten, die jeweils in einem Aufruf der Tiefensuche schwarz werden, gehören zu einer Komponente. □



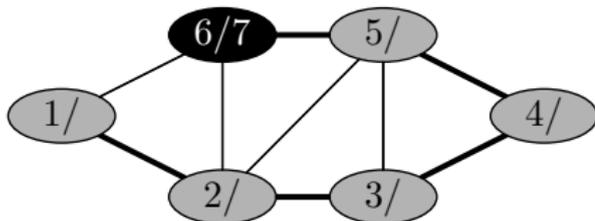
Zusammenhangskomponenten

Theorem

Die Zusammenhangskomponenten eines ungerichteten Graphen können in linearer Zeit gefunden werden.

Beweis.

Die Knoten, die jeweils in einem Aufruf der Tiefensuche schwarz werden, gehören zu einer Komponente. □



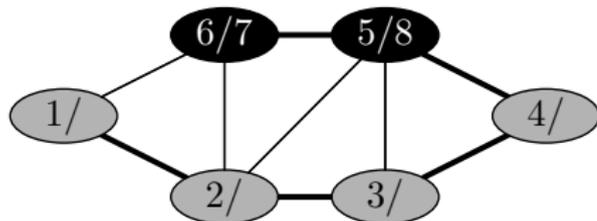
Zusammenhangskomponenten

Theorem

Die Zusammenhangskomponenten eines ungerichteten Graphen können in linearer Zeit gefunden werden.

Beweis.

Die Knoten, die jeweils in einem Aufruf der Tiefensuche schwarz werden, gehören zu einer Komponente. □



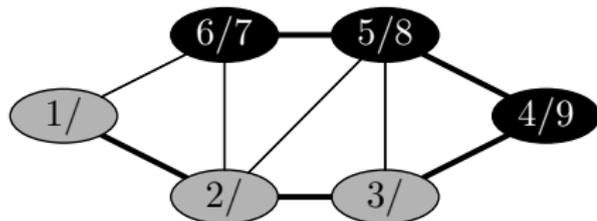
Zusammenhangskomponenten

Theorem

Die Zusammenhangskomponenten eines ungerichteten Graphen können in linearer Zeit gefunden werden.

Beweis.

Die Knoten, die jeweils in einem Aufruf der Tiefensuche schwarz werden, gehören zu einer Komponente. □



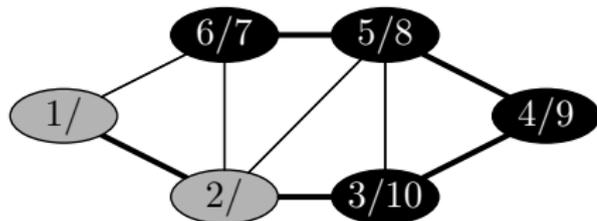
Zusammenhangskomponenten

Theorem

Die Zusammenhangskomponenten eines ungerichteten Graphen können in linearer Zeit gefunden werden.

Beweis.

Die Knoten, die jeweils in einem Aufruf der Tiefensuche schwarz werden, gehören zu einer Komponente. □



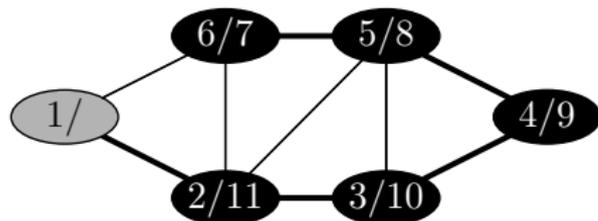
Zusammenhangskomponenten

Theorem

Die Zusammenhangskomponenten eines ungerichteten Graphen können in linearer Zeit gefunden werden.

Beweis.

Die Knoten, die jeweils in einem Aufruf der Tiefensuche schwarz werden, gehören zu einer Komponente. □



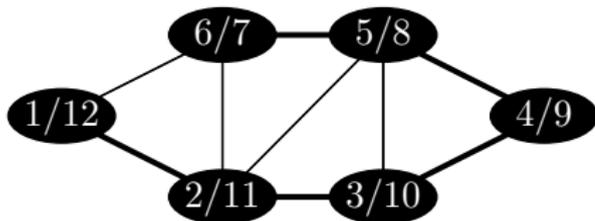
Zusammenhangskomponenten

Theorem

Die Zusammenhangskomponenten eines ungerichteten Graphen können in linearer Zeit gefunden werden.

Beweis.

Die Knoten, die jeweils in einem Aufruf der Tiefensuche schwarz werden, gehören zu einer Komponente. □



Finden von Kreisen

Theorem

Gegeben sei ein gerichteter Graph G . Dann können wir in linearer Zeit feststellen, ob G azyklisch ist (keine Kreise enthält).

Beweis.

Führe eine Tiefensuche auf G aus.

Behauptung: G ist genau dann azyklisch, wenn es keine Rückwärtskanten gibt.

\Rightarrow Wenn es eine Rückwärtskante von u nach v gibt, dann gibt es auch einen Pfad von v nach u im DFS-Wald. Dies ist ein Kreis.

\Leftarrow Angenommen es gibt einen Kreis. Sei u der Knoten auf dem Kreis mit minimalem $d(u)$ und v der Knoten auf dem Kreis vor u . Dann gilt $d(u) < d(v)$ und $f(v) < d(u)$.

Also ist (v, u) eine Rückwärtskante. □

Finden von Kreisen

Theorem

Gegeben sei ein gerichteter Graph G . Dann können wir in linearer Zeit feststellen, ob G azyklisch ist (keine Kreise enthält).

Beweis.

Führe eine Tiefensuche auf G aus.

Behauptung: G ist genau dann azyklisch, wenn es keine Rückwärtskanten gibt.

⇒ Wenn es eine Rückwärtskante von u nach v gibt, dann gibt es auch einen Pfad von v nach u im DFS-Wald. Dies ist ein Kreis.

⇐ Angenommen es gibt einen Kreis. Sei u der Knoten auf dem Kreis mit minimalem $d(u)$ und v der Knoten auf dem Kreis vor u . Dann gilt $d(u) < d(v)$ und $f(v) < d(u)$.

Also ist (v, u) eine Rückwärtskante. □

Finden von Kreisen

Theorem

Gegeben sei ein gerichteter Graph G . Dann können wir in linearer Zeit feststellen, ob G azyklisch ist (keine Kreise enthält).

Beweis.

Führe eine Tiefensuche auf G aus.

Behauptung: G ist genau dann azyklisch, wenn es keine Rückwärtskanten gibt.

⇒ Wenn es eine Rückwärtskante von u nach v gibt, dann gibt es auch einen Pfad von v nach u im DFS-Wald. Dies ist ein Kreis.

⇐ Angenommen es gibt einen Kreis. Sei u der Knoten auf dem Kreis mit minimalem $d(u)$ und v der Knoten auf dem Kreis vor u . Dann gilt $d(u) < d(v)$ und $f(v) < d(u)$.

Also ist (v, u) eine Rückwärtskante. □

Finden von Kreisen

Theorem

Gegeben sei ein gerichteter Graph G . Dann können wir in linearer Zeit feststellen, ob G azyklisch ist (keine Kreise enthält).

Beweis.

Führe eine Tiefensuche auf G aus.

Behauptung: G ist genau dann azyklisch, wenn es keine Rückwärtskanten gibt.

\Rightarrow Wenn es eine Rückwärtskante von u nach v gibt, dann gibt es auch einen Pfad von v nach u im DFS-Wald. Dies ist ein Kreis.

\Leftarrow Angenommen es gibt einen Kreis. Sei u der Knoten auf dem Kreis mit minimalem $d(u)$ und v der Knoten auf dem Kreis vor u . Dann gilt $d(u) < d(v)$ und $f(v) < d(u)$.

Also ist (v, u) eine Rückwärtskante. □

Übersicht

- 3 Graphalgorithmen
 - Darstellung von Graphen
 - Tiefensuche
 - **Starke Komponenten**
 - Topologisches Sortieren
 - Kürzeste Pfade
 - Netzwerkalgorithmen
 - Minimale Spannbäume

Starke Zusammenhangskomponenten

Definition

Es sei $G = (V, E)$ ein gerichteter Graph. Ein **Pfad** der Länge k von u_1 nach u_{k+1} ist eine Folge $(u_1, u_2), (u_2, u_3), \dots, (u_k, u_{k+1})$ von Kanten aus E wobei u_1, \dots, u_{k+1} paarweise verschieden sind.

Eine Menge $C \subseteq V$ ist eine **starke Zusammenhangskomponente**, wenn es Pfade zwischen allen Paaren von Knoten in C gibt und es keine echte Obermenge von C mit dieser Eigenschaft gibt.

Die starken Komponenten sind eine Verfeinerung der Zusammenhangskomponenten des entsprechenden ungerichteten Graphen.

Starke Zusammenhangskomponenten

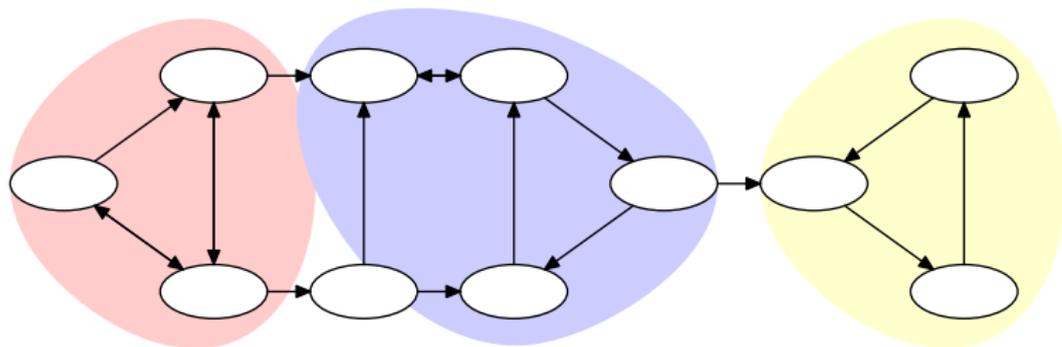
Definition

Es sei $G = (V, E)$ ein gerichteter Graph. Ein **Pfad** der Länge k von u_1 nach u_{k+1} ist eine Folge $(u_1, u_2), (u_2, u_3), \dots, (u_k, u_{k+1})$ von Kanten aus E wobei u_1, \dots, u_{k+1} paarweise verschieden sind.

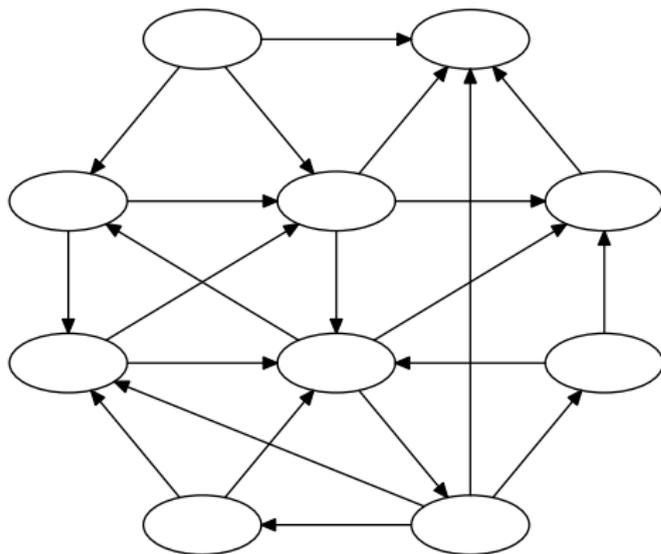
Eine Menge $C \subseteq V$ ist eine **starke Zusammenhangskomponente**, wenn es Pfade zwischen allen Paaren von Knoten in C gibt und es keine echte Obermenge von C mit dieser Eigenschaft gibt.

Die starken Komponenten sind eine Verfeinerung der Zusammenhangskomponenten des entsprechenden ungerichteten Graphen.

Starke Komponenten – Beispiel



Es gibt genau vier starke Komponenten.

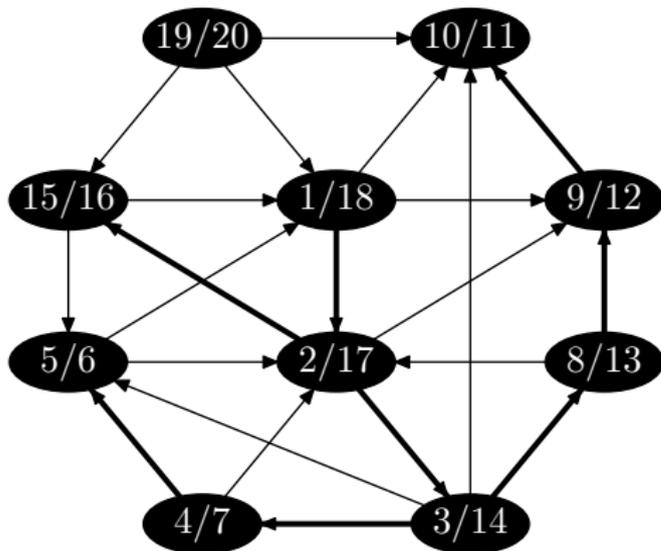


Lemma

Der Knoten mit der größten finish time nach einer DFS ist in einer Quellenkomponente.

Beweis.

Er ist Wurzel eines DFS-Baums T . Wenn es einen anderen DFS-Baum gäbe, von dem eine Kante nach T führte, dann müßte dieser danach besucht werden. □

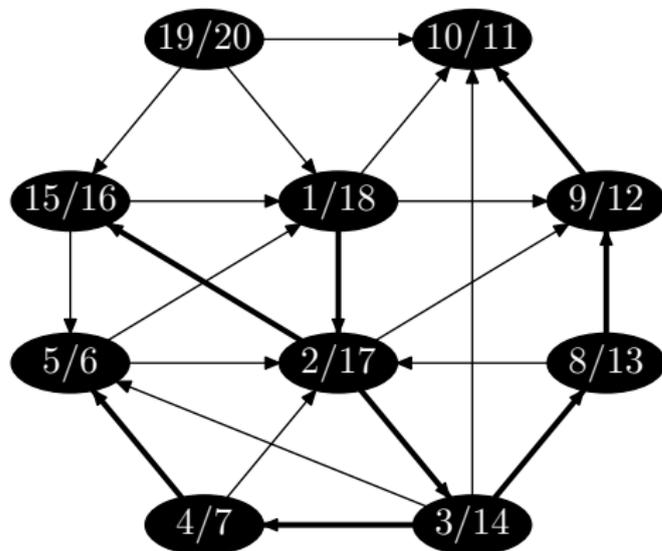


Lemma

Der Knoten mit der größten finish time nach einer DFS ist in einer Quellenkomponente.

Beweis.

Er ist Wurzel eines DFS-Baums T . Wenn es einen anderen DFS-Baum gäbe, von dem eine Kante nach T führte, dann müßte dieser danach besucht werden. □

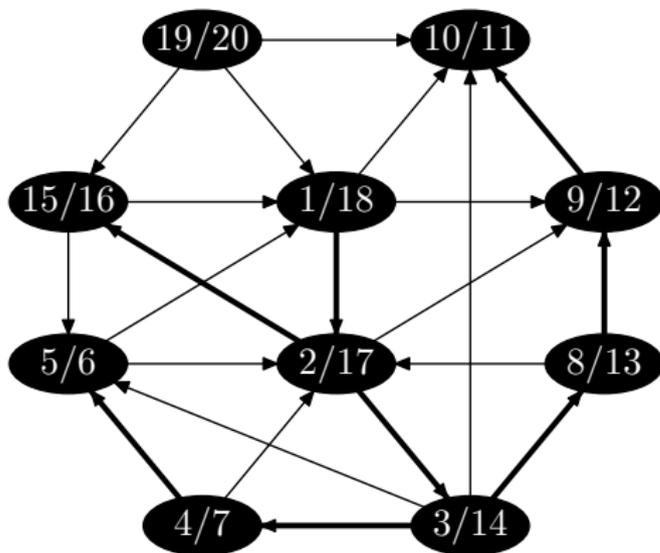


Lemma

Der Knoten mit der größten finish time nach einer DFS ist in einer Quellenkomponente.

Beweis.

Er ist Wurzel eines DFS-Baums T . Wenn es einen anderen DFS-Baum gäbe, von dem eine Kante nach T führte, dann müßte dieser danach besucht werden. □



Quellenkomponente:

Starke Komponente, in die keine Kante hineinführt

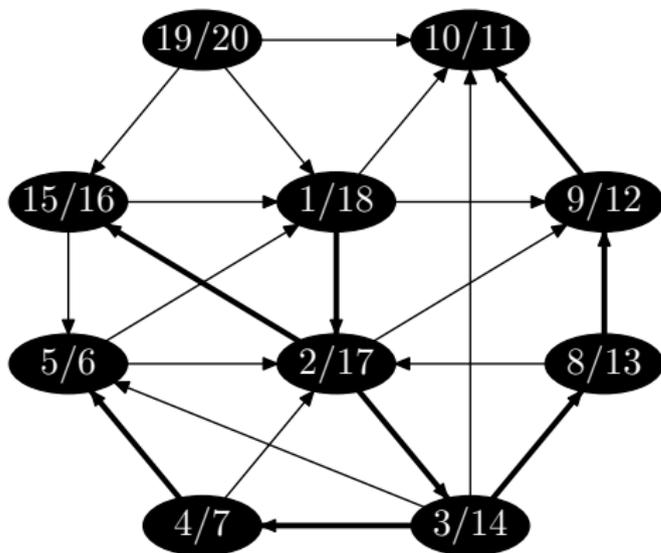
Senkenkomponente: Starke Komponente, aus der keine Kante herausführt

G_r : Wie G , aber Richtung der Kanten umgekehrt

Korollar

Der Knoten mit der größten finish time nach einer DFS in G_r ist in einer Senkenkomponente von G .

Die starken Komponenten von G und G_r sind identisch.



Quellenkomponente:

Starke Komponente, in die keine Kante hineinführt

Senkenkomponente: Starke Komponente, aus der keine Kante herausführt

G_r : Wie G , aber Richtung der Kanten umgekehrt

Korollar

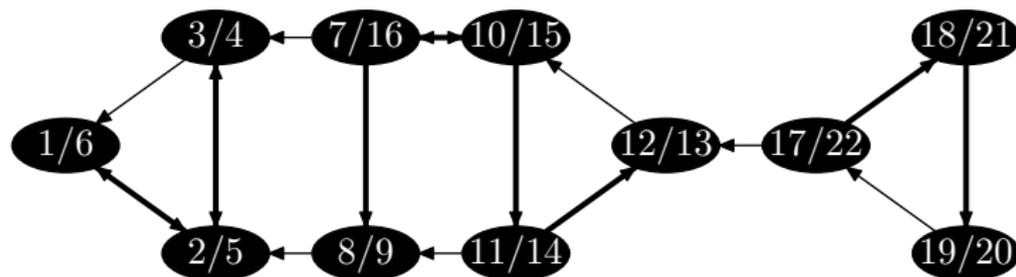
Der Knoten mit der größten finish time nach einer DFS in G_r ist in einer Senkenkomponente von G .

Die starken Komponenten von G und G_r sind identisch.

Starke Komponenten

Lemma

Gegeben sei ein gerichteter Graph G mit einem DFS-Wald. Der DFS-Wald bleibt samt discover und finish times ein möglicher DFS-Wald, wenn die Quellenkomponente entfernt wird, die den Knoten mit maximaler finish time enthält.

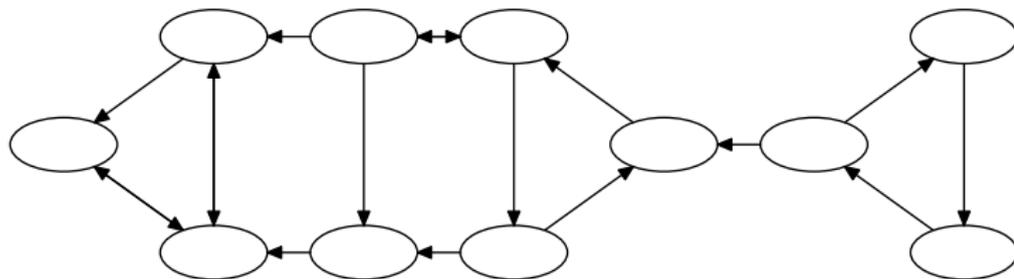


Beweis.

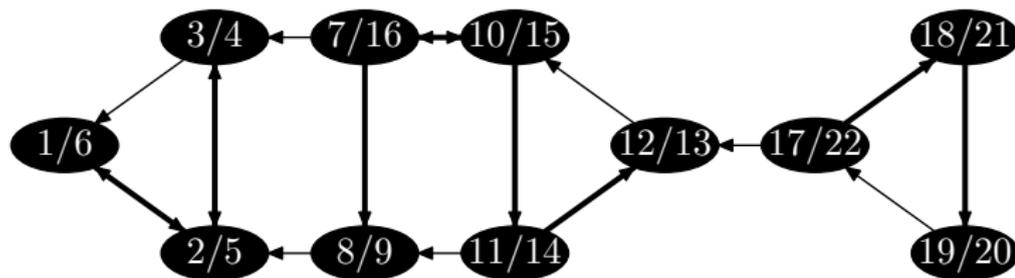
Der Baum spannt die ganze starke Komponente auf. Er wurde als letzter besucht. □

Starke Komponenten – Algorithmus von Kosaraju

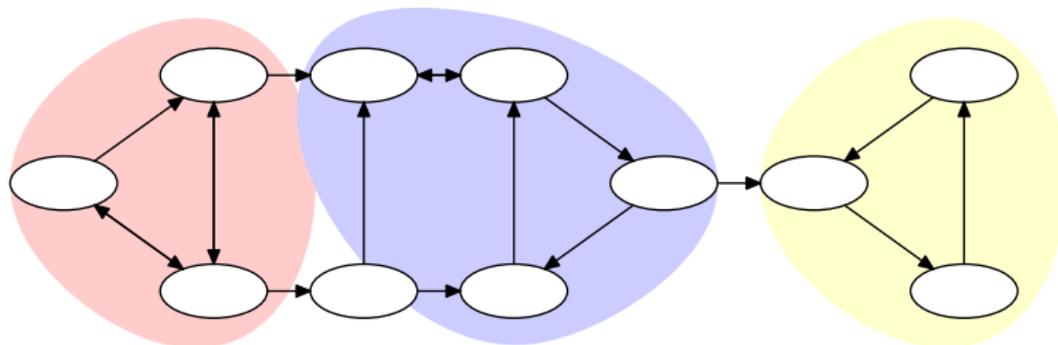
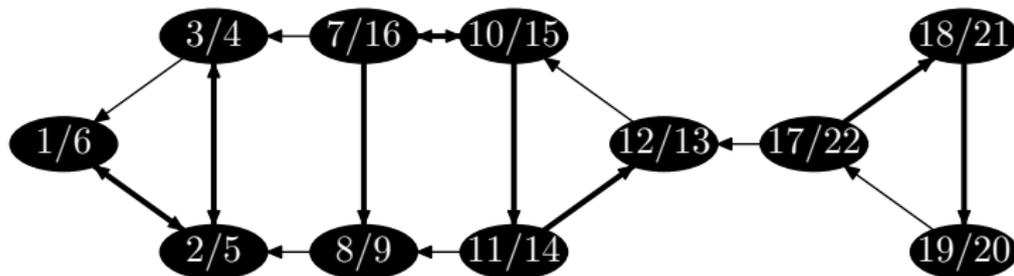
- 1 Bilde G_r (Kanten umdrehen)
- 2 Führe Tiefensuche auf G_r aus
- 3 Ordne die Knoten nach absteigender finish time $f(v)$
- 4 Führe in dieser Reihenfolge Tiefensuche auf G aus
- 5 Jeder Baum im DFS-Wald spannt eine starke Komponente auf



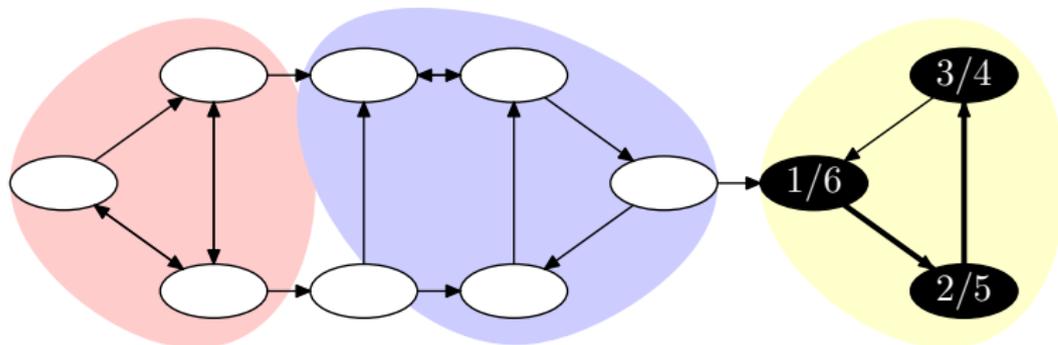
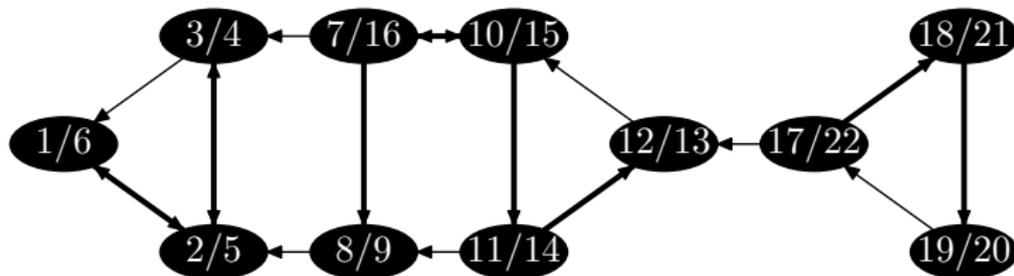
Starke Komponenten – Algorithmus von Kosaraju



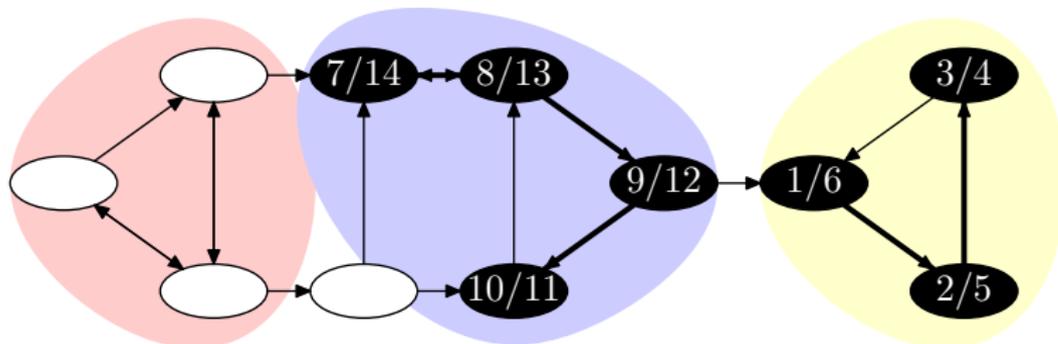
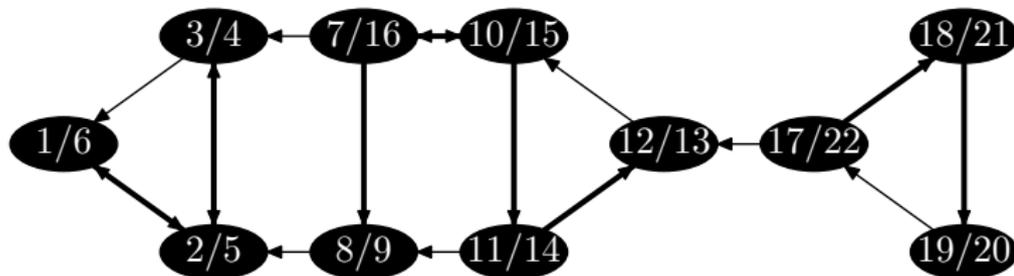
Starke Komponenten – Algorithmus von Kosaraju



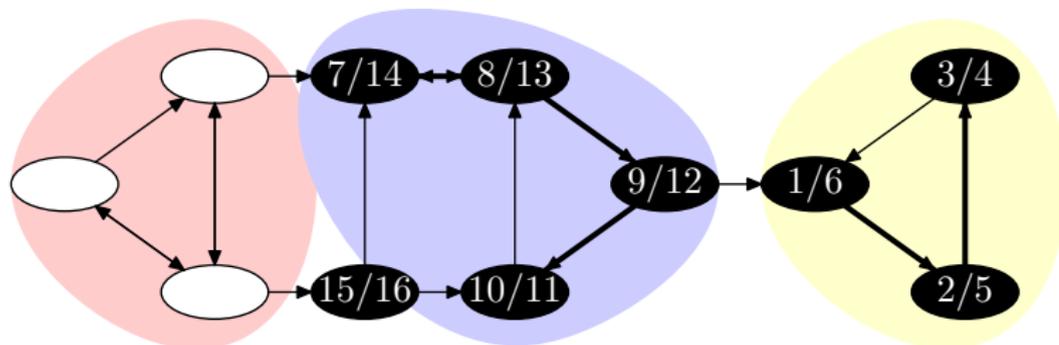
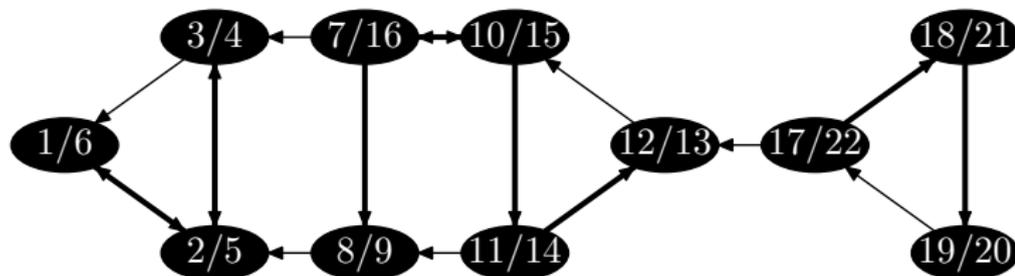
Starke Komponenten – Algorithmus von Kosaraju



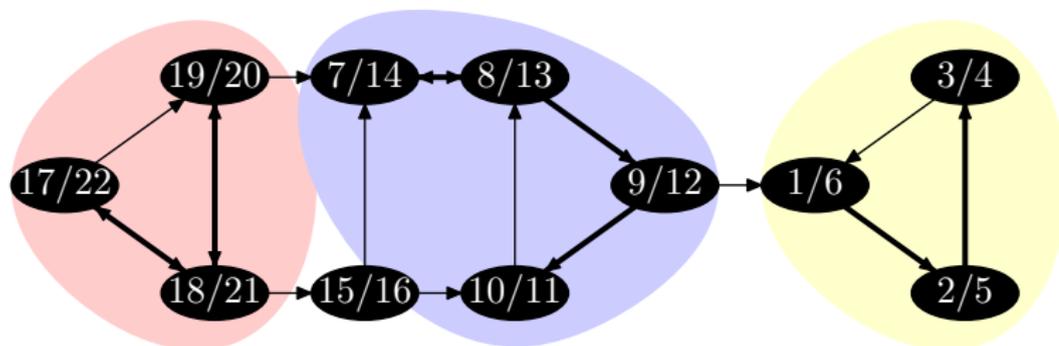
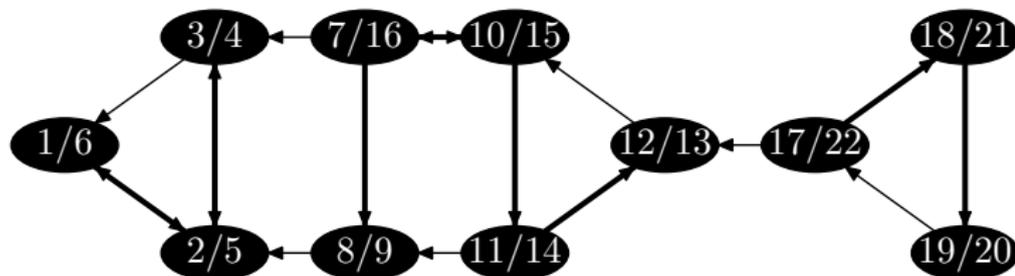
Starke Komponenten – Algorithmus von Kosaraju



Starke Komponenten – Algorithmus von Kosaraju



Starke Komponenten – Algorithmus von Kosaraju



Theorem

Der Algorithmus von Kosaraju berechnet die starken Komponenten eines gerichteten Graphen.

Beweis.

Die DFS auf G beginnt in einem Knoten einer Senkenkomponente von G . Dabei entsteht ein DFS-Baum, der diese starke Komponente aufspannt.

Die DFS fährt dann wieder in einer Senkenkomponente des Restgraphen fort und so weiter. □

