Datenstrukturen und Algorithmen

Peter Rossmanith

Theoretische Informatik, RWTH Aachen

26. Juli 2011

Beweis.

$$R_{k-1} = \frac{g^{(k)}(\xi)}{k!} f(n)^k \text{ für ein } \xi \in [-|f(n)|, |f(n)|],$$

falls z nahe bei 0.

Frage: Hängt ξ von $f(n)$ ab?

$$\Rightarrow R_{k-1} \leq \max_{\xi \in [-\epsilon, \epsilon]} \frac{g^{(k)}(\xi)}{k!} f(n)^k$$

für ein $\epsilon > 0$ bis auf endlich viele n.

Da $g^{(k)}$ stetig und $[-\epsilon, \epsilon]$ kompakt ist, existiert das Maximum nach dem Satz von Weierstraß.

Also gilt $R_{k-1} = O(f(n)^k)$. □

Beweis.

$$R_{k-1} = \frac{g^{(k)}(\xi)}{k!} f(n)^k \text{ für ein } \xi \in [-|f(n)|, |f(n)|],$$

falls z nahe bei 0.

$$\Rightarrow R_{k-1} \leq \max_{\xi \in [-\epsilon, \epsilon]} \frac{g^{(k)}(\xi)}{k!} f(n)^k$$

für ein $\epsilon > 0$ bis auf endlich viele n.

Frage: Existiert das Maximum? Da $g^{(k)}$ stetig und $[-\epsilon, \epsilon]$ kompakt ist, existiert das Maximum nach dem Satz von Weierstraß.

Also gilt $R_{k-1} = O(f(n)^k)$. □

Beweis.

$$R_{k-1} = \frac{g^{(k)}(\xi)}{k!} f(n)^k \text{ für ein } \xi \in [-|f(n)|, |f(n)|],$$

falls z nahe bei 0.

$$\Rightarrow R_{k-1} \leq \max_{\xi \in [-\epsilon, \epsilon]} \frac{g^{(k)}(\xi)}{k!} f(n)^k$$

für ein $\epsilon > 0$ bis auf endlich viele n.

Da $g^{(k)}$ stetig und $[-\epsilon, \epsilon]$ kompakt ist, existiert das Maximum nach dem Satz von Weierstraß.

Also gilt $R_{k-1} = O(f(n)^k)$. □

Beweis.

$$R_{k-1} = \frac{g^{(k)}(\xi)}{k!} f(n)^k \text{ für ein } \xi \in [-|f(n)|, |f(n)|],$$

falls z nahe bei 0.

$$\Rightarrow R_{k-1} \leq \max_{\xi \in [-\epsilon, \epsilon]} \frac{g^{(k)}(\xi)}{k!} f(n)^k$$

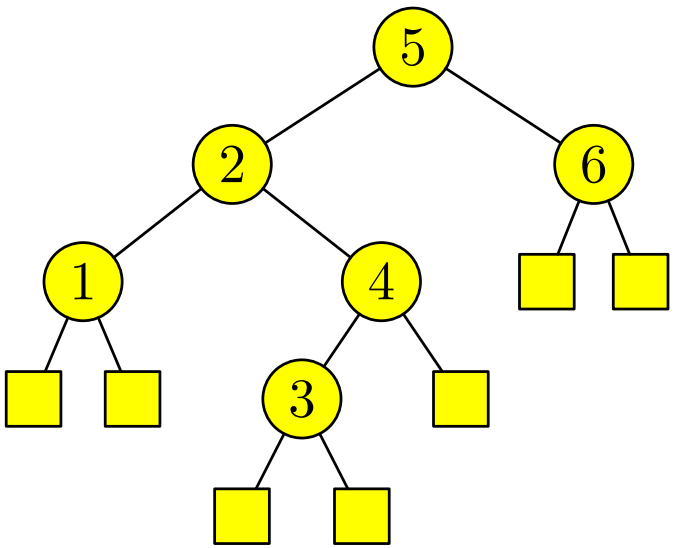
für ein $\epsilon > 0$ bis auf endlich viele n.

Da $g^{(k)}$ stetig und $[-\epsilon, \epsilon]$ kompakt ist, existiert das Maximum nach dem Satz von Weierstraß.

Also gilt $R_{k-1} = O(f(n)^k)$. □

Binäre Suchbäume – Löschen

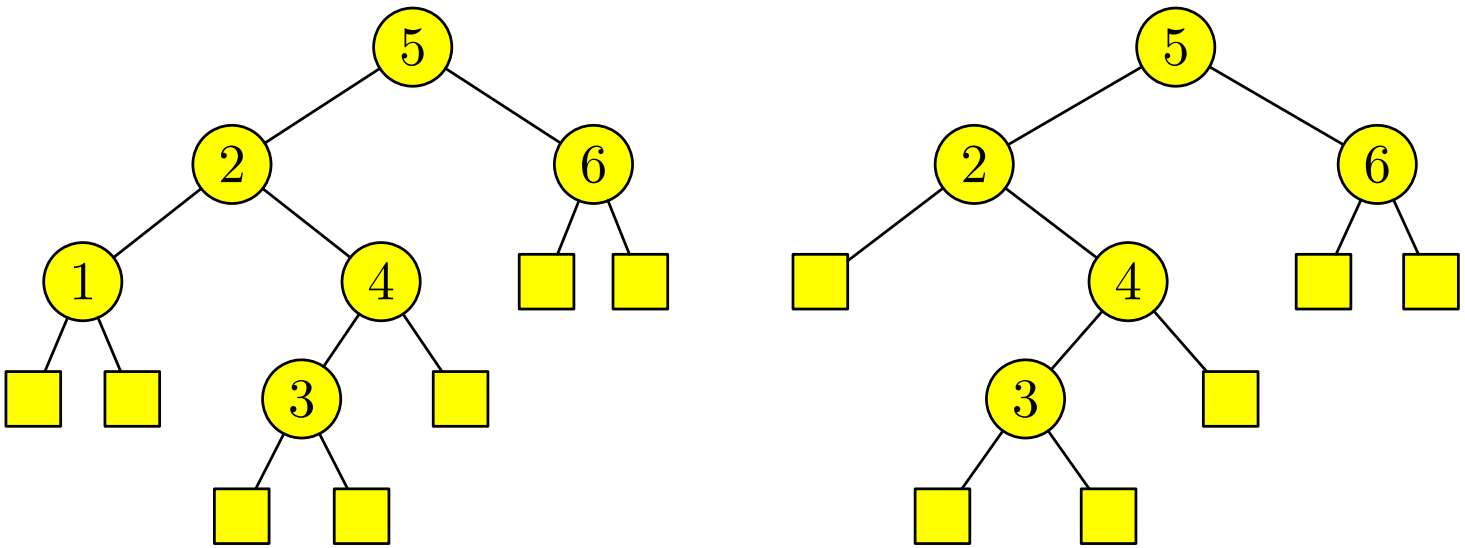
Löschen eines Blatts:



Ein Blatt kann durch Zeigerverbiegen gelöscht werden.

Binäre Suchbäume – Löschen

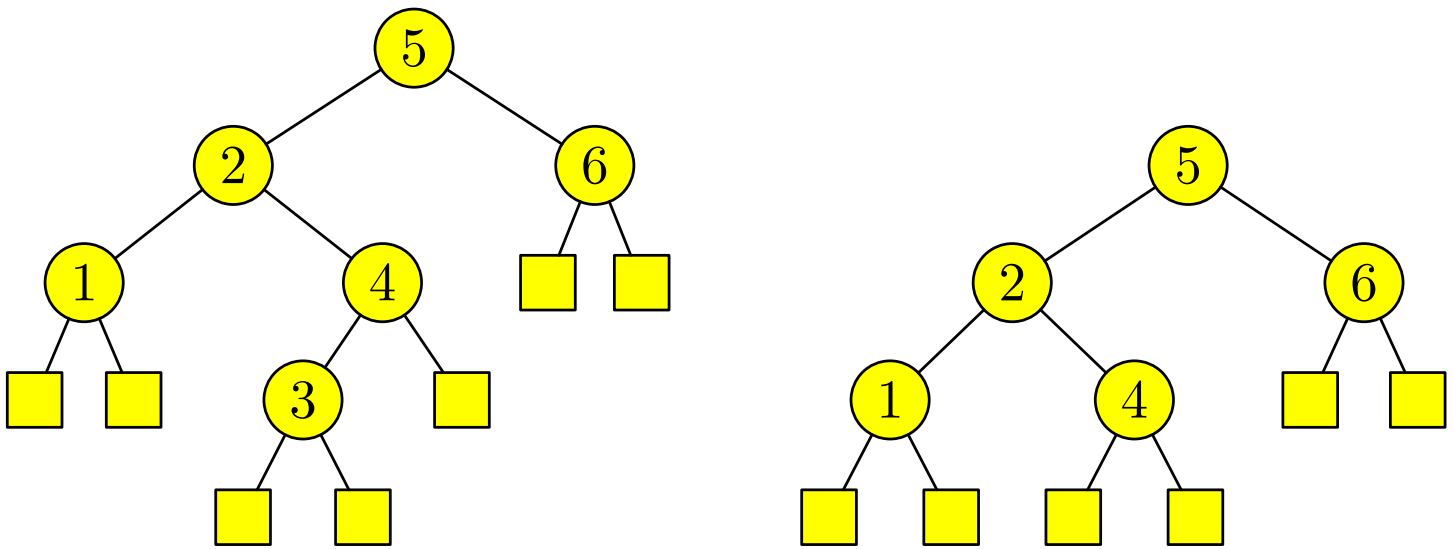
Löschen eines Blatts:



Ein Blatt kann durch Zeigerverbiegen gelöscht werden.

Binäre Suchbäume – Löschen

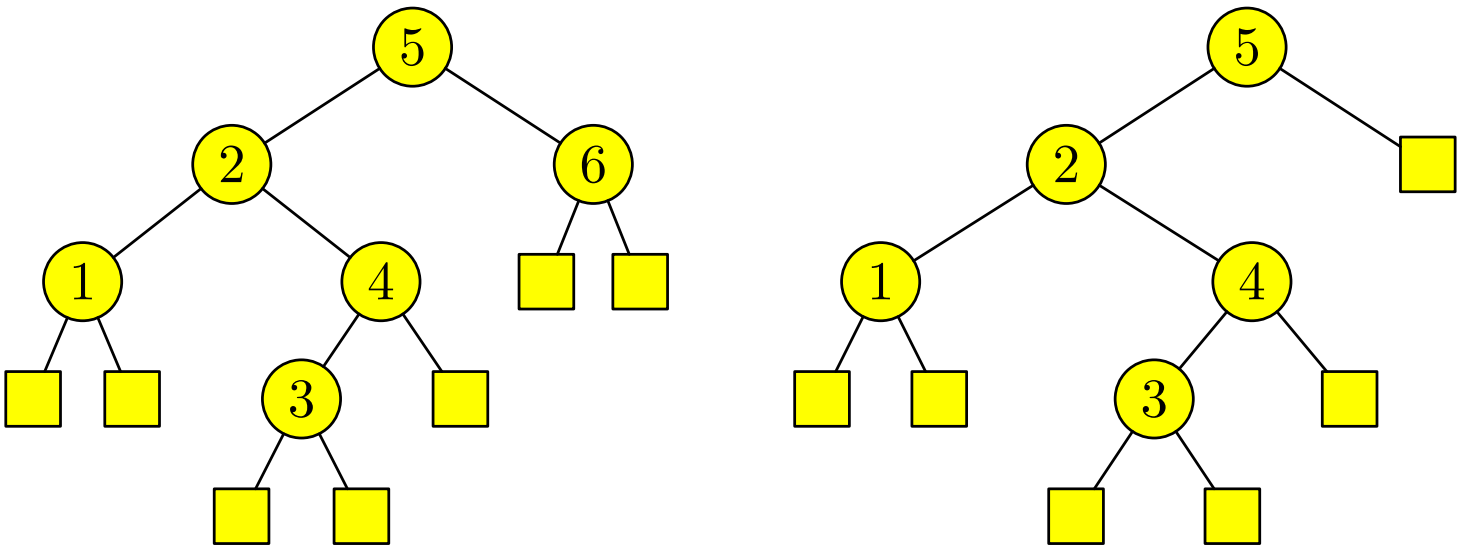
Löschen eines Blatts:



Ein Blatt kann durch Zeigerverbiegen gelöscht werden.

Binäre Suchbäume – Löschen

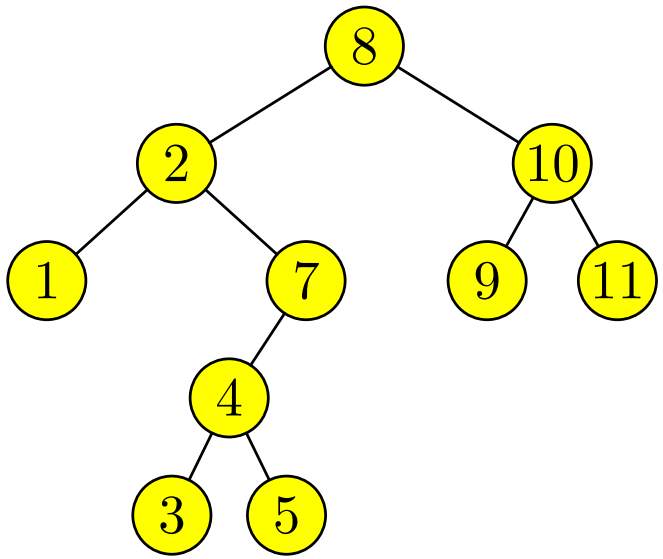
Löschen eines Blatts:



Ein Blatt kann durch Zeigerverbiegen gelöscht werden.

Binäre Suchbäume – Löschen

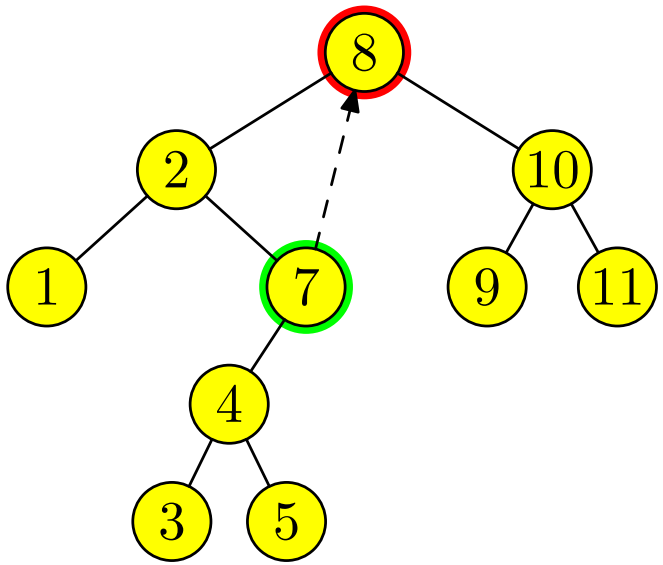
Löschen eines Knotens mit linkem Kind:



1. Finde den größten Knoten im linken Unterbaum
2. Kopiere seinen Inhalt
3. Lösche ihn

Binäre Suchbäume – Löschen

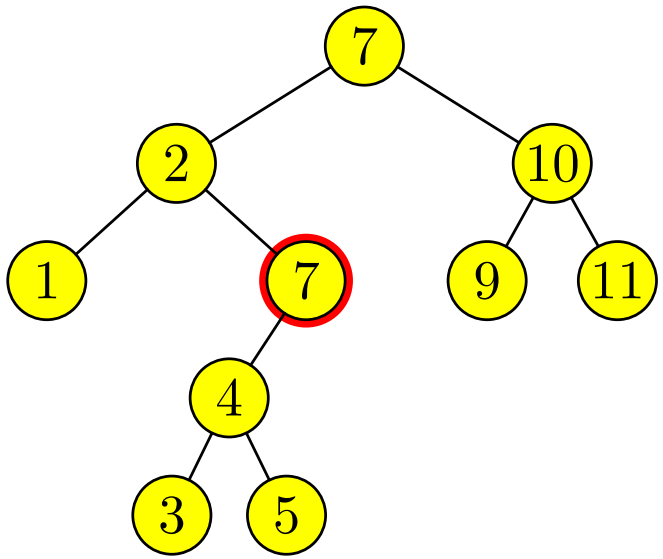
Löschen eines Knotens mit linkem Kind:



1. Finde den größten Knoten im linken Unterbaum
2. Kopiere seinen Inhalt
3. Lösche ihn

Binäre Suchbäume – Löschen

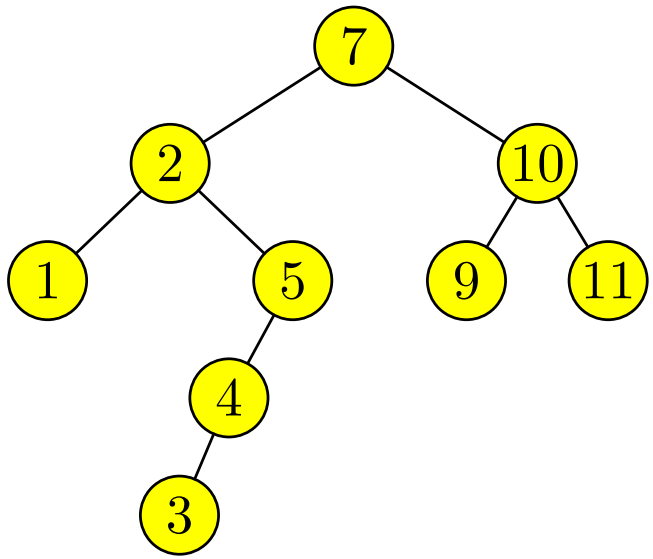
Löschen eines Knotens mit linkem Kind:



1. Finde den größten Knoten im linken Unterbaum
2. Kopiere seinen Inhalt
3. Lösche ihn

Binäre Suchbäume – Löschen

Löschen eines Knotens mit linkem Kind:



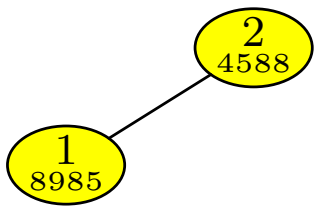
1. Finde den größten Knoten im linken Unterbaum
2. Kopiere seinen Inhalt
3. Lösche ihn

Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.

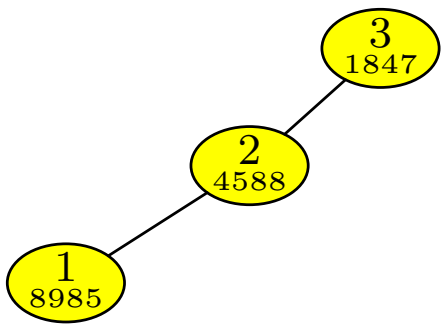
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



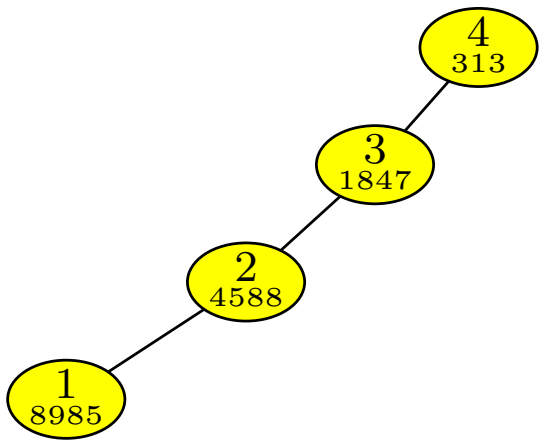
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



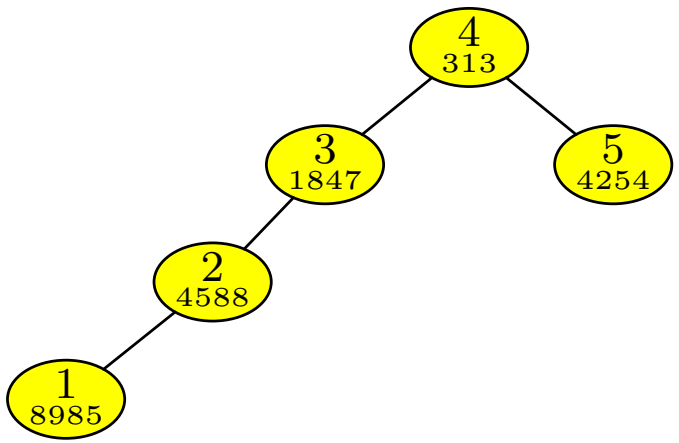
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



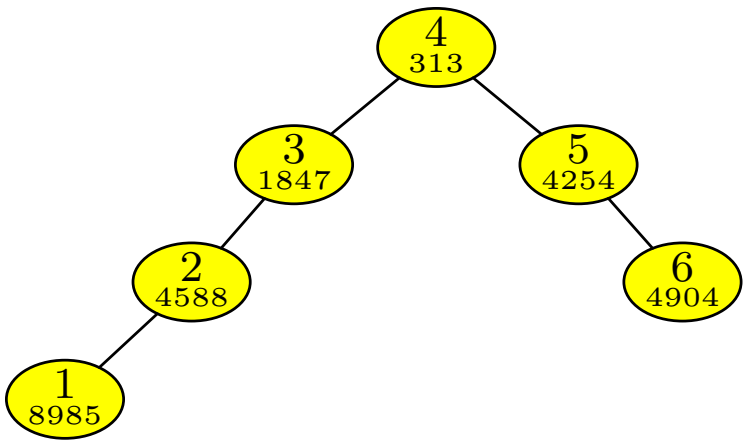
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



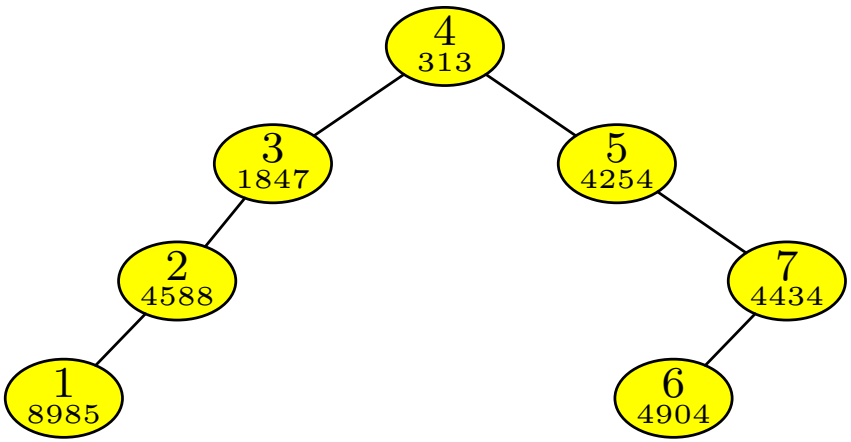
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



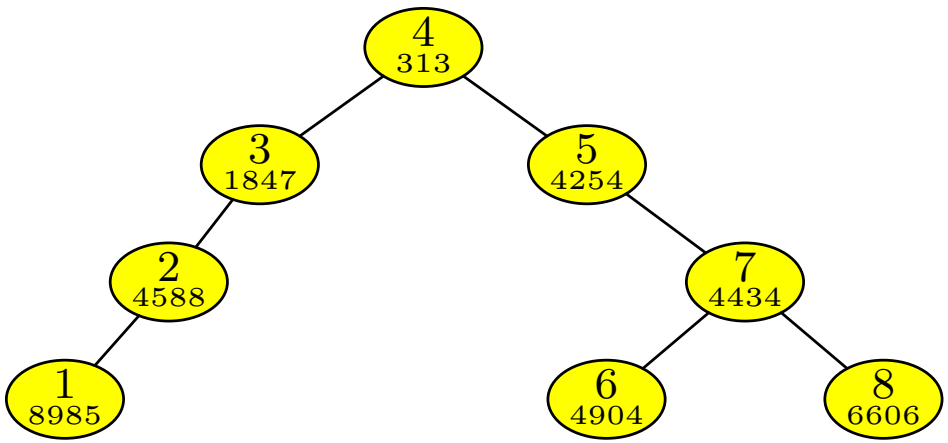
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



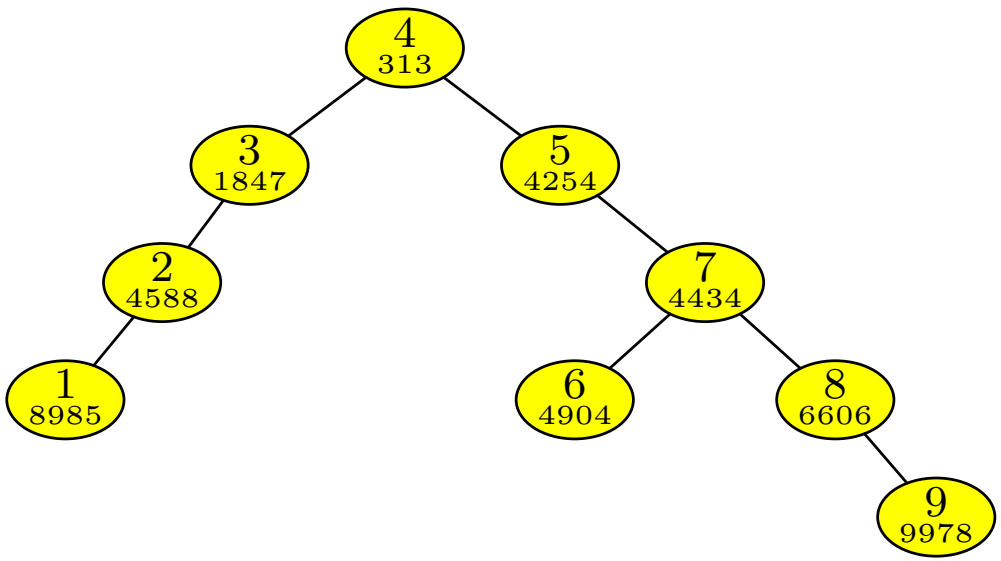
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



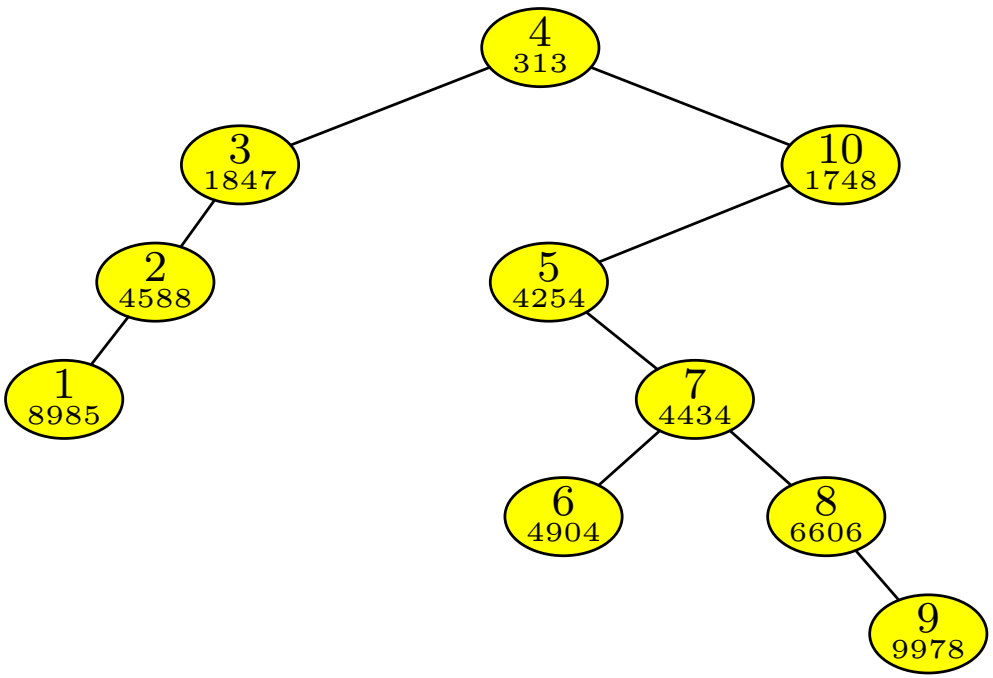
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



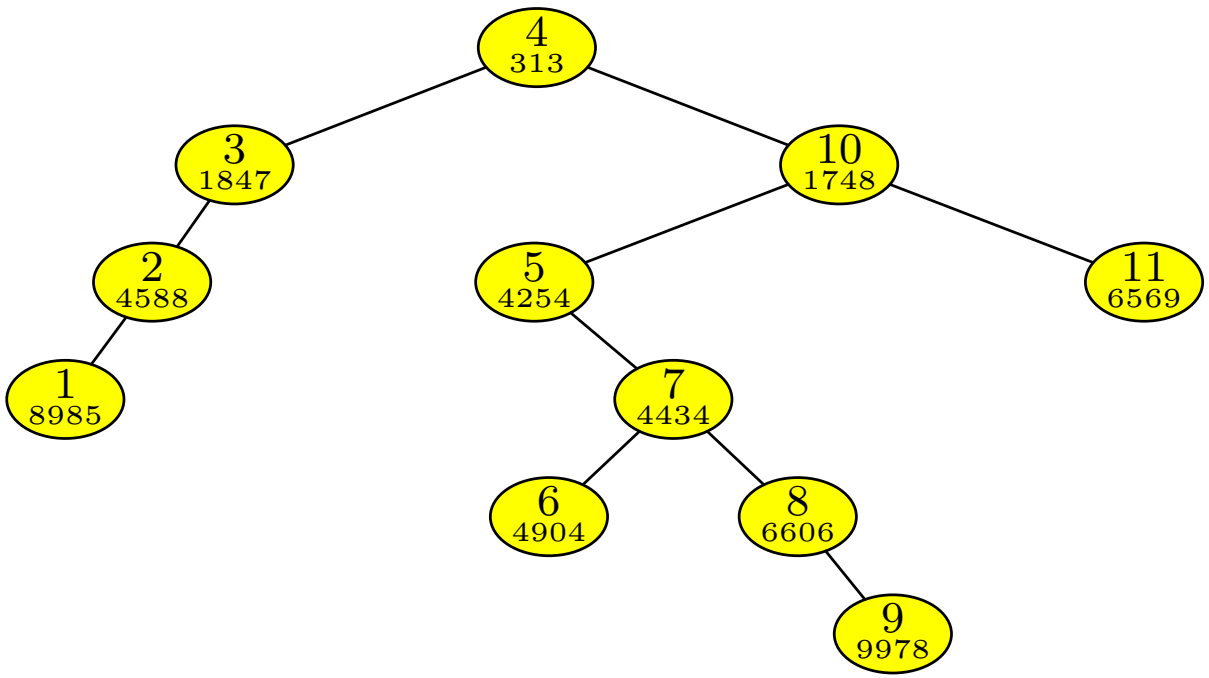
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



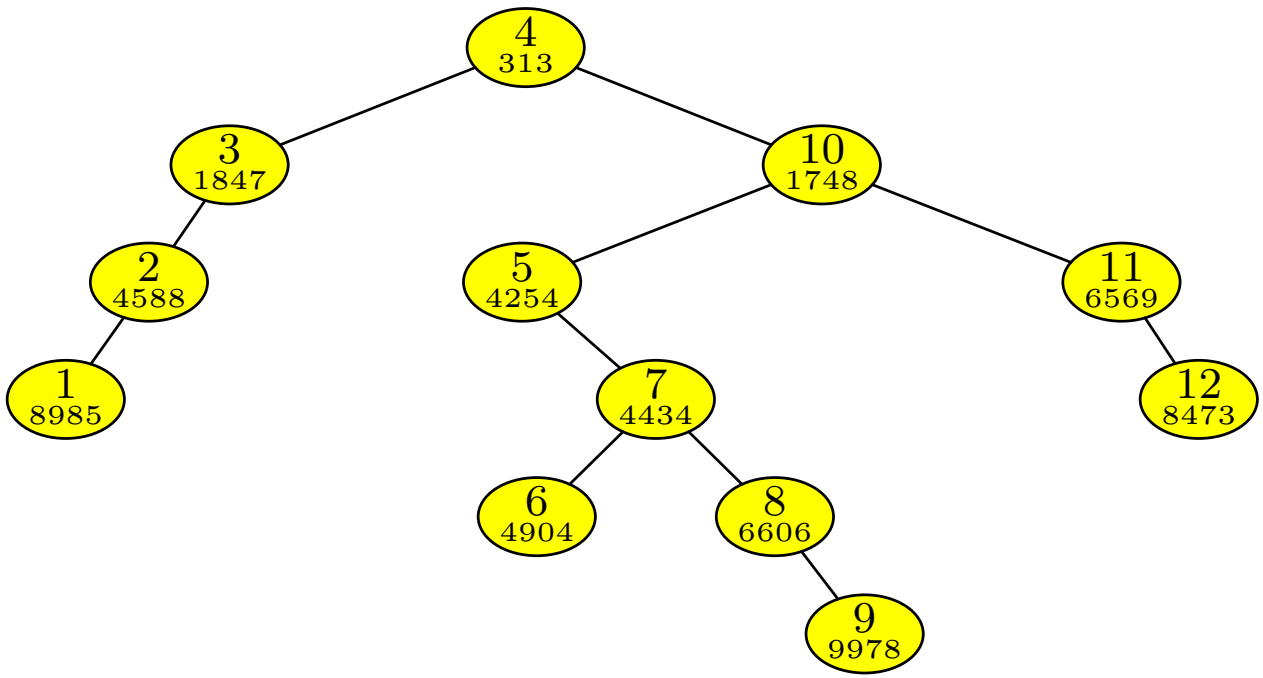
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



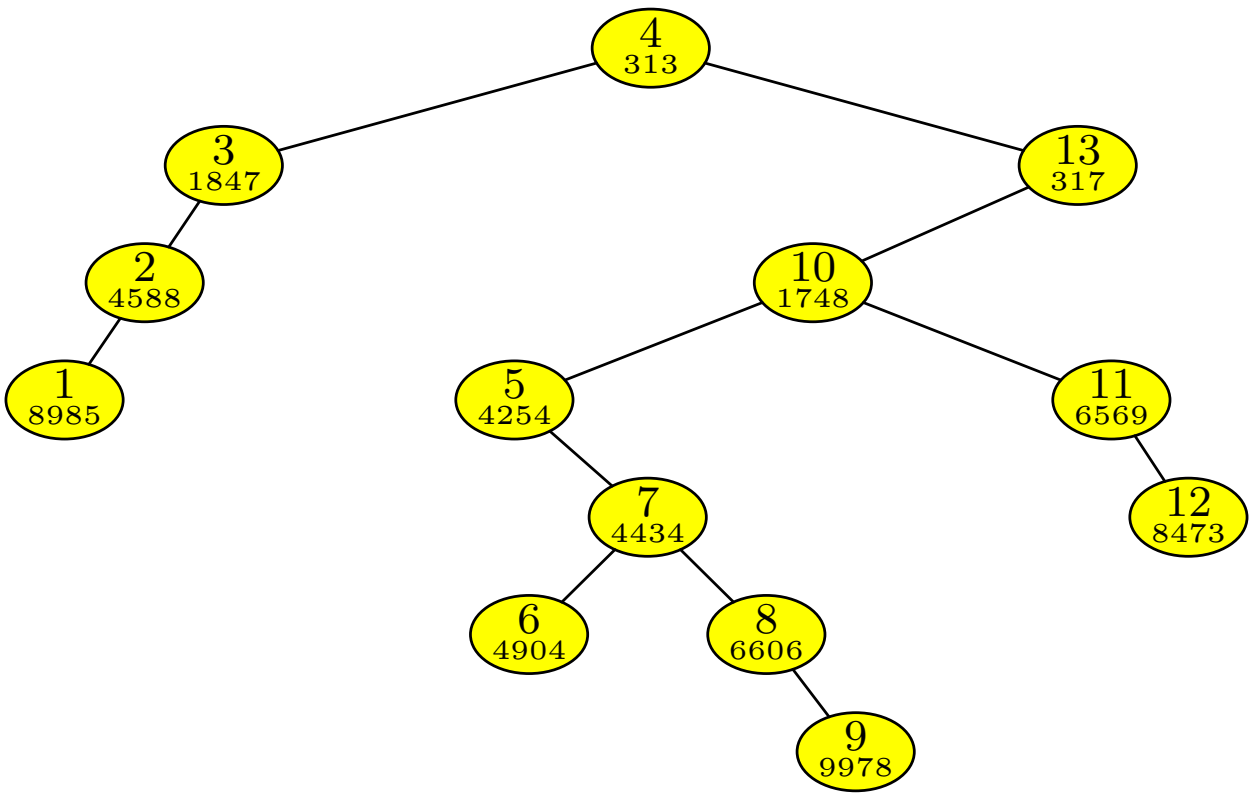
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



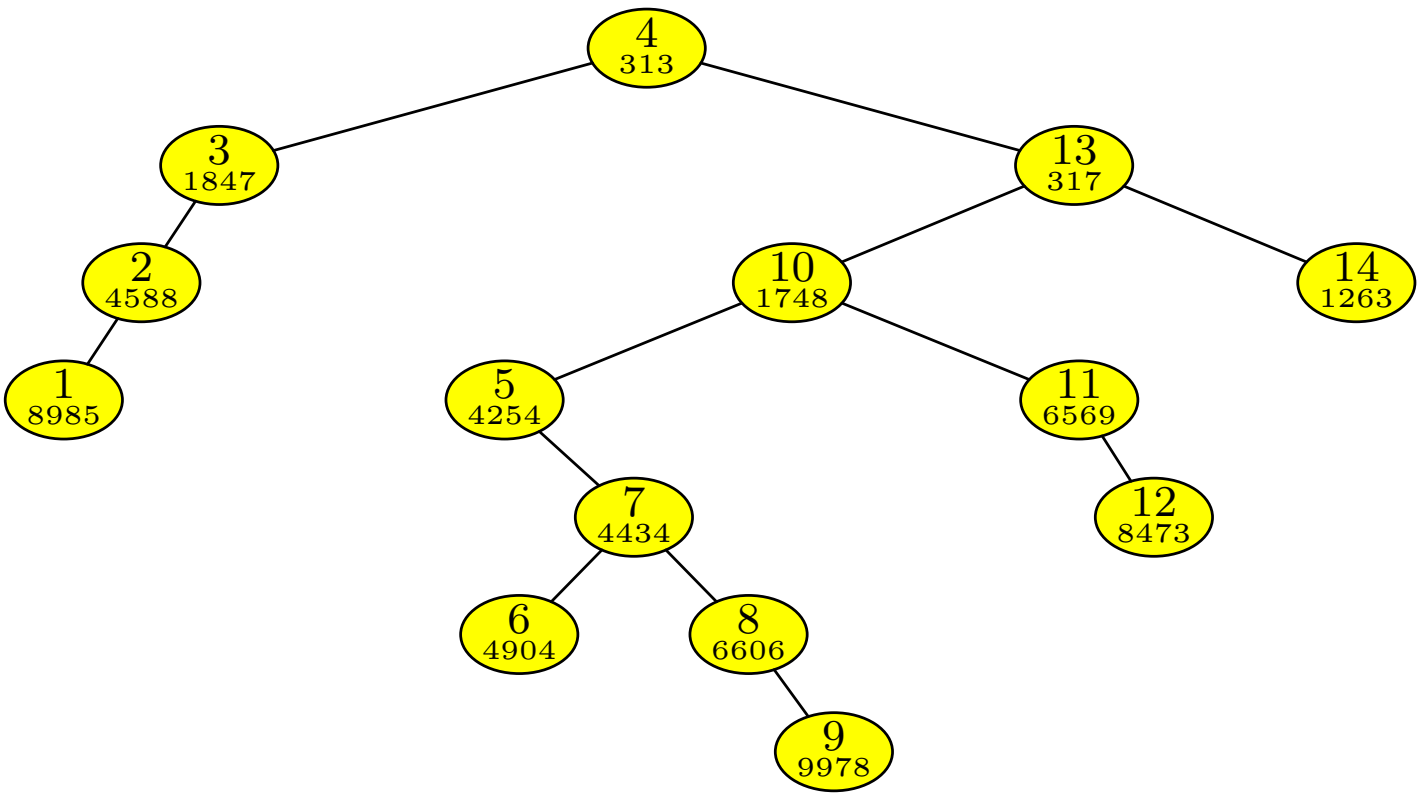
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



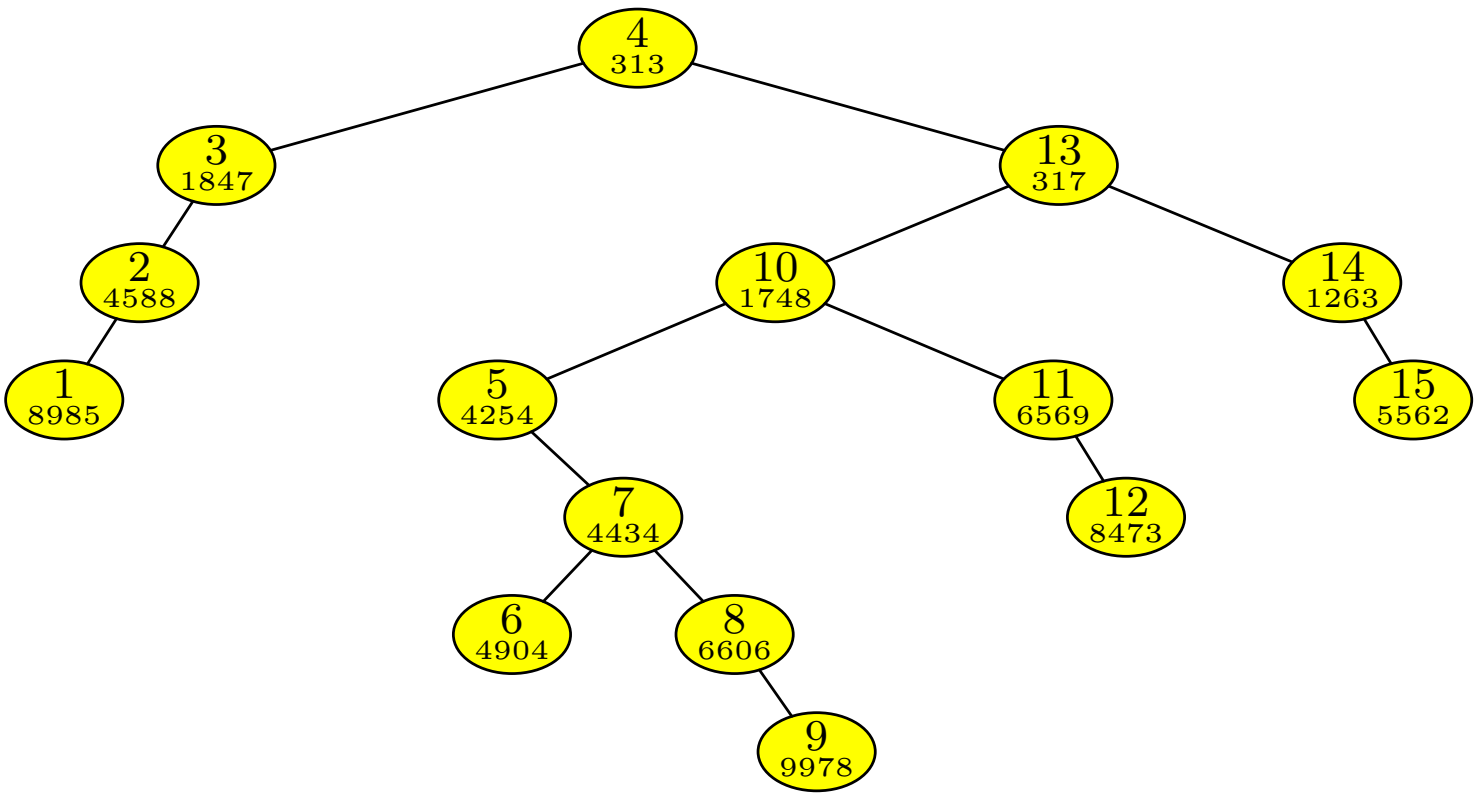
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



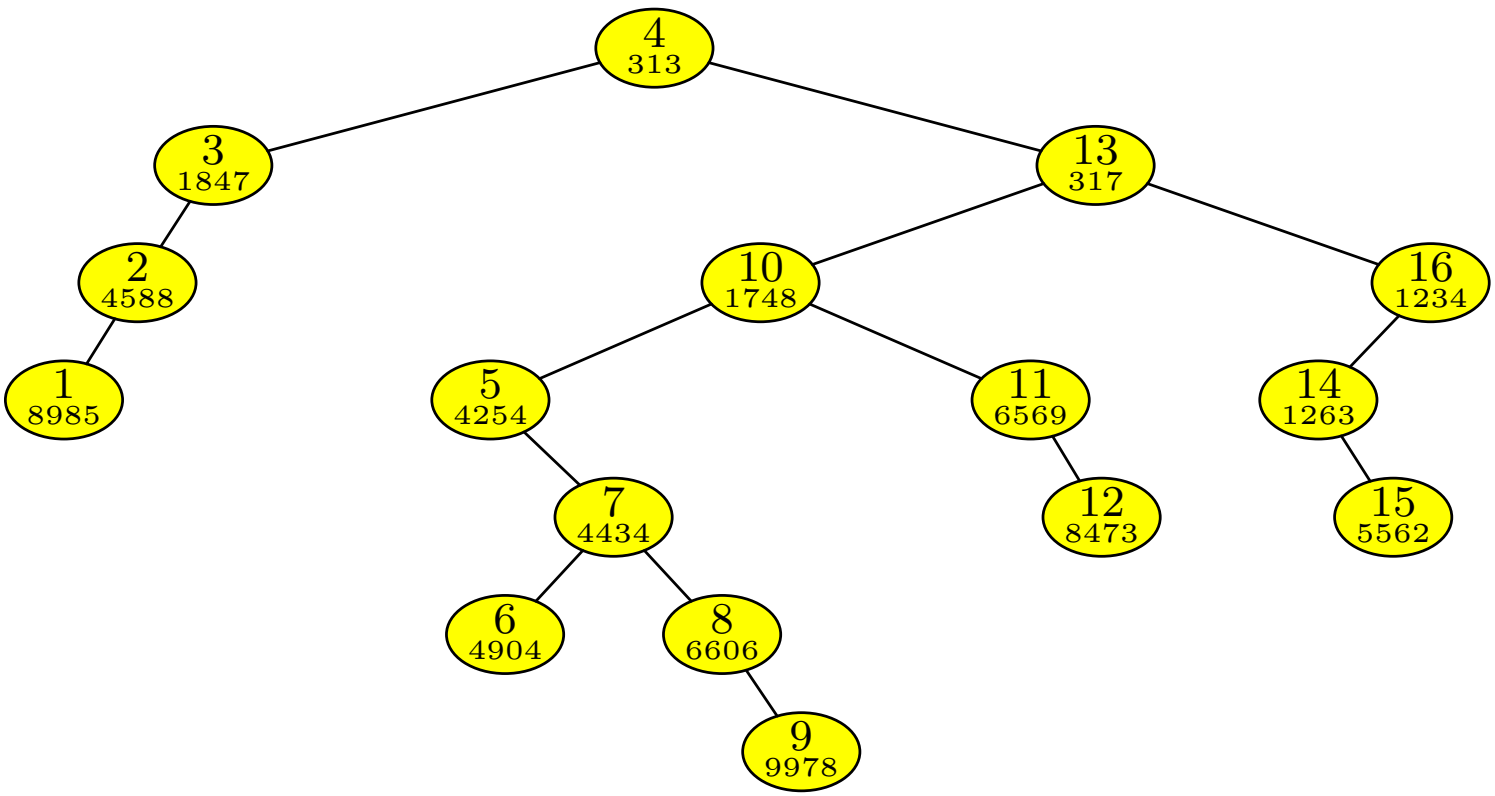
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



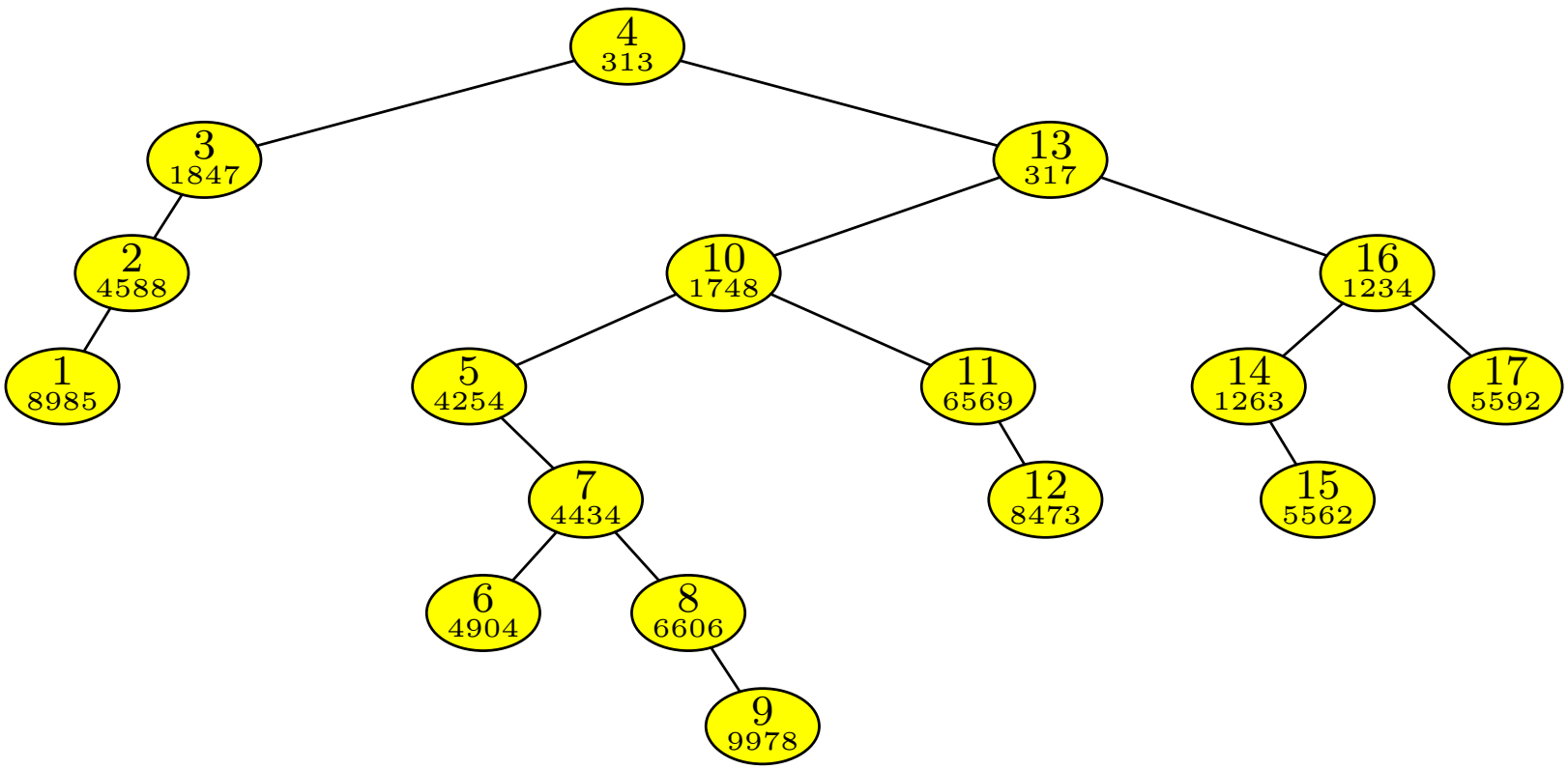
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



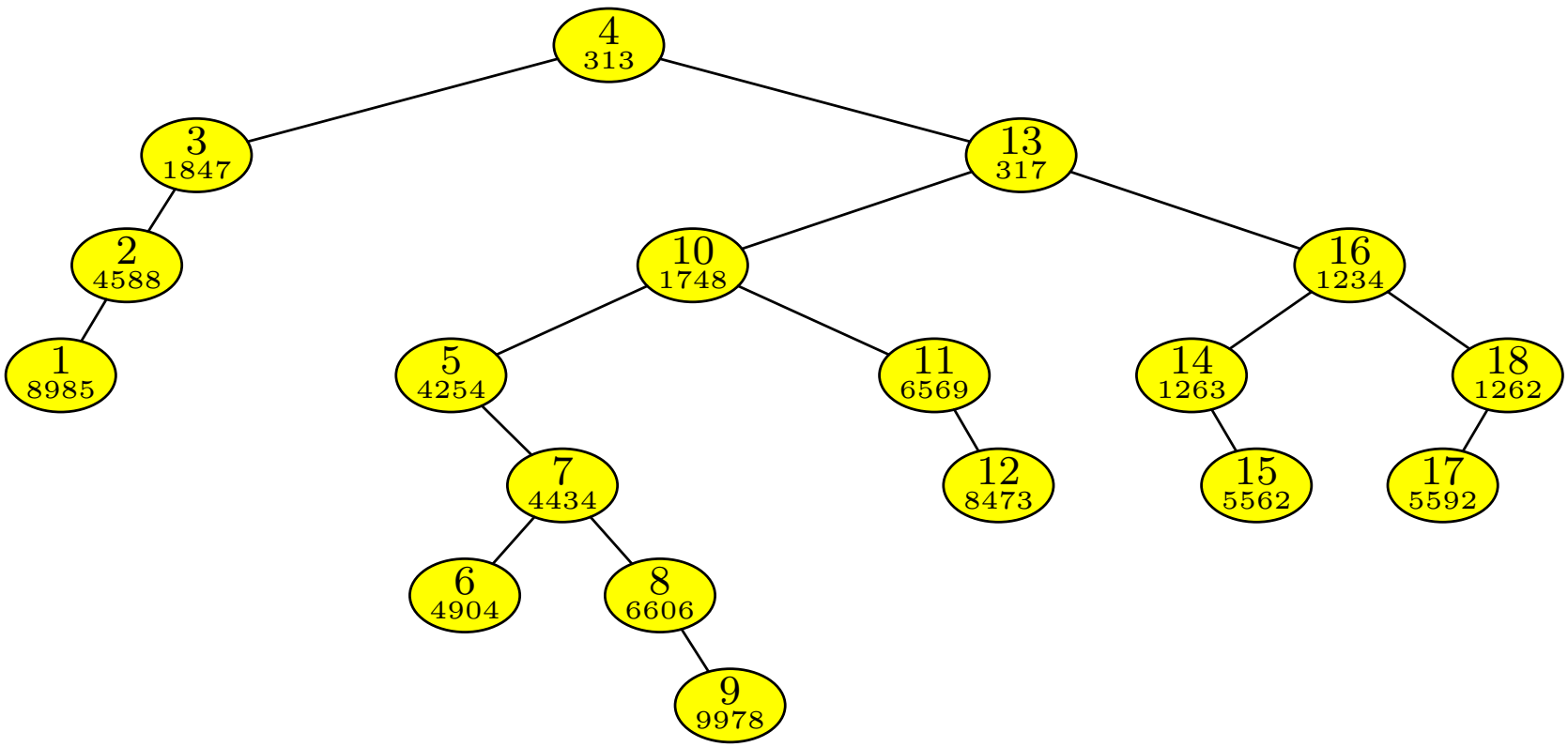
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



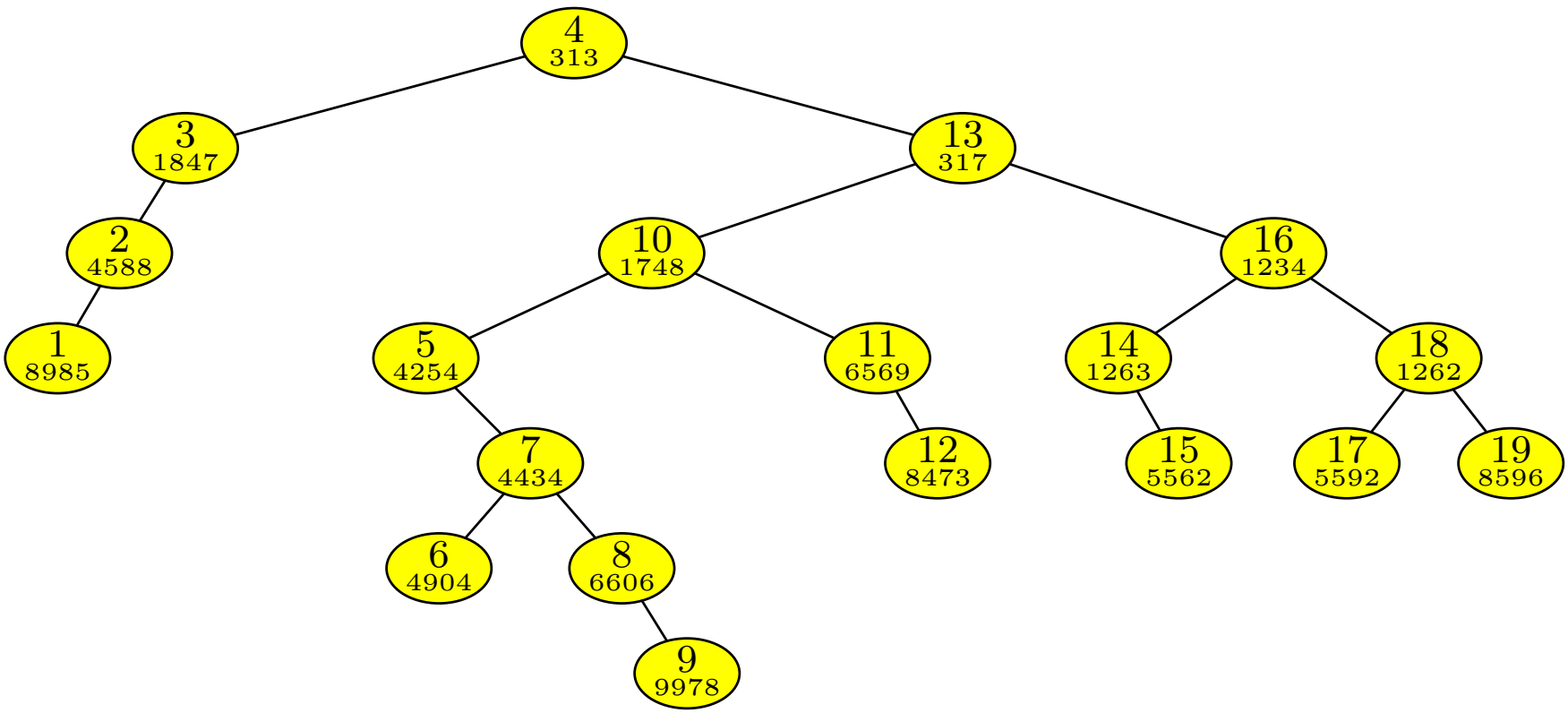
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



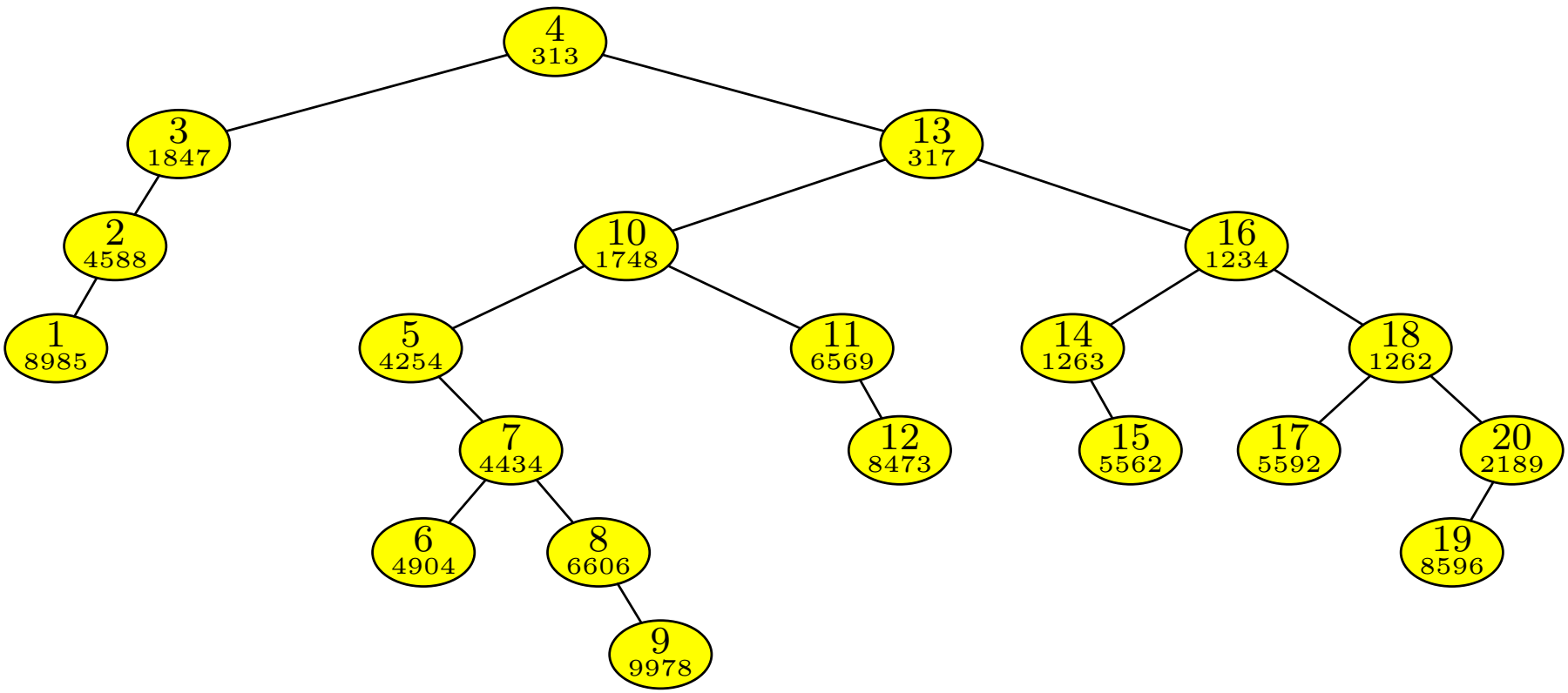
Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.

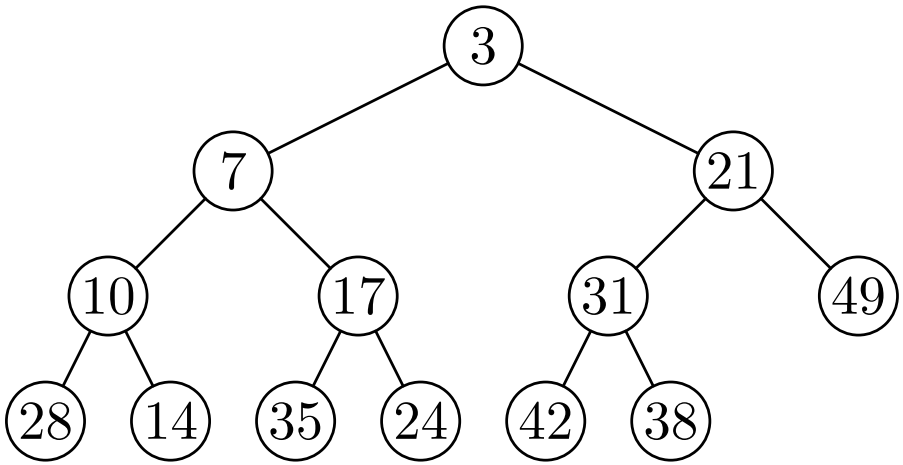


Einfügen von 20 Elementen in einen Treap

Die Prioritäten sind zufällig gewählt.



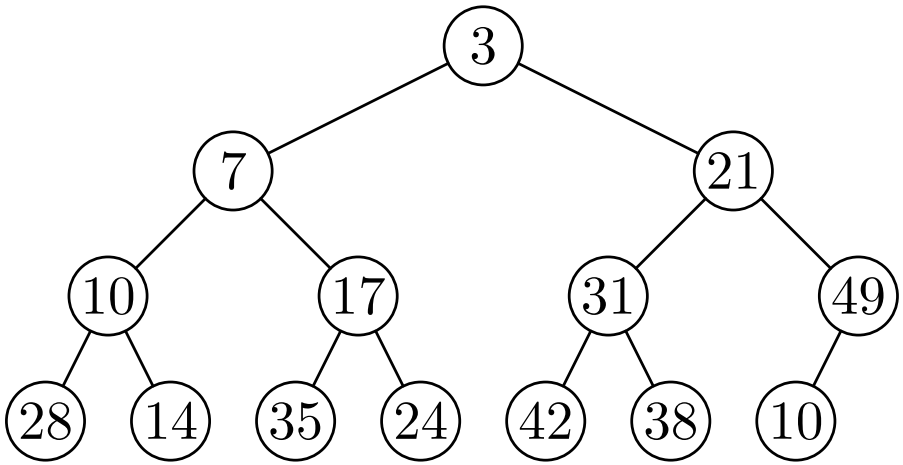
Einfügen in einen Heap



Es soll der Schlüssel 10 eingefügt werden.

1. Hänge den Schlüssel an das Ende
2. Die Heap-Eigenschaft ist jetzt verletzt
3. Lasse den neuen Schlüssel zur richtigen Stelle aufsteigen.

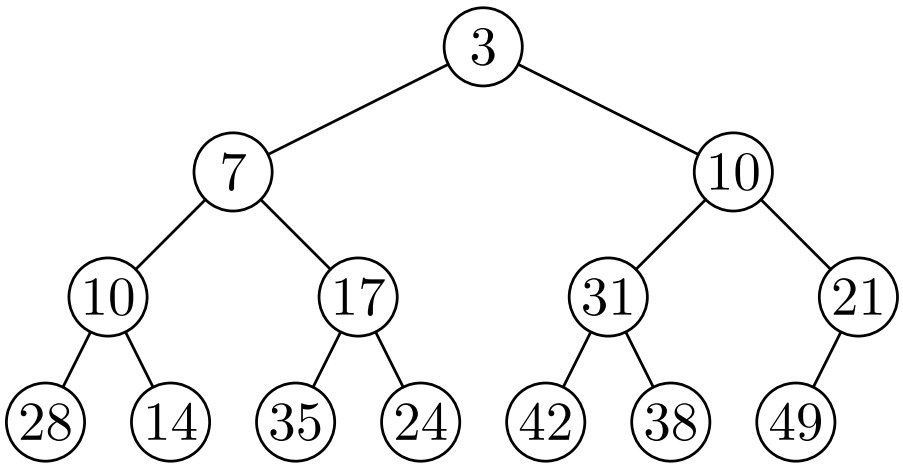
Einfügen in einen Heap



Es soll der Schlüssel 10 eingefügt werden.

1. Hänge den Schlüssel an das Ende
2. Die Heap-Eigenschaft ist jetzt verletzt
3. Lasse den neuen Schlüssel zur richtigen Stelle aufsteigen.

Einfügen in einen Heap



Es soll der Schlüssel 10 eingefügt werden.

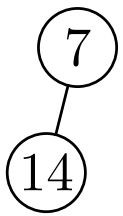
1. Hänge den Schlüssel an das Ende
2. Die Heap-Eigenschaft ist jetzt verletzt
3. Lasse den neuen Schlüssel zur richtigen Stelle aufsteigen.

So wird der Heap schrittweise aufgebaut.

Wir bilden einen Heap aus den Schlüsseln 7, 14, 21, 28, 35, 42, 49, 3, 10, 17, 24, 31, und 38:

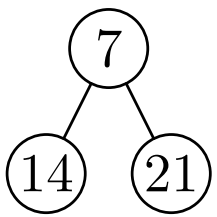
So wird der Heap schrittweise aufgebaut.

Wir bilden einen Heap aus den Schlüsseln 7, 14, 21, 28, 35, 42, 49, 3, 10, 17, 24, 31, und 38:



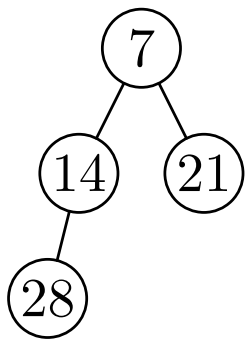
So wird der Heap schrittweise aufgebaut.

Wir bilden einen Heap aus den Schlüsseln 7, 14, 21, 28, 35, 42, 49, 3, 10, 17, 24, 31, und 38:



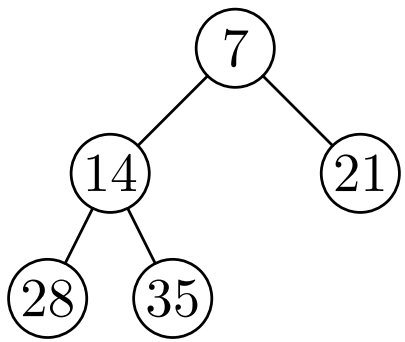
So wird der Heap schrittweise aufgebaut.

Wir bilden einen Heap aus den Schlüsseln 7, 14, 21, 28, 35, 42, 49, 3, 10, 17, 24, 31, und 38:



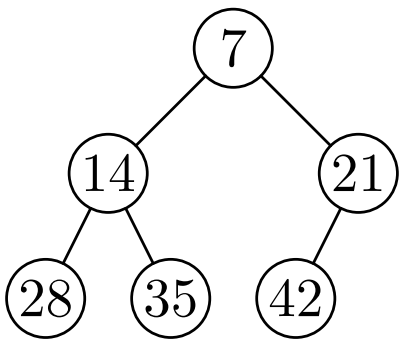
So wird der Heap schrittweise aufgebaut.

Wir bilden einen Heap aus den Schlüsseln 7, 14, 21, 28, 35, 42, 49, 3, 10, 17, 24, 31, und 38:



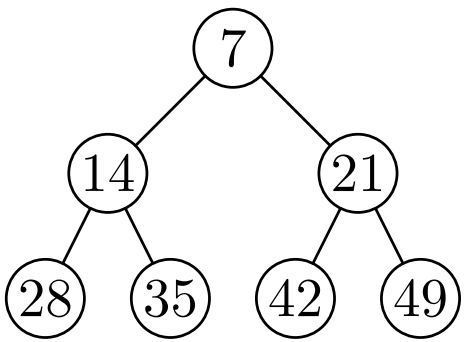
So wird der Heap schrittweise aufgebaut.

Wir bilden einen Heap aus den Schlüsseln 7, 14, 21, 28, 35, 42, 49, 3, 10, 17, 24, 31, und 38:



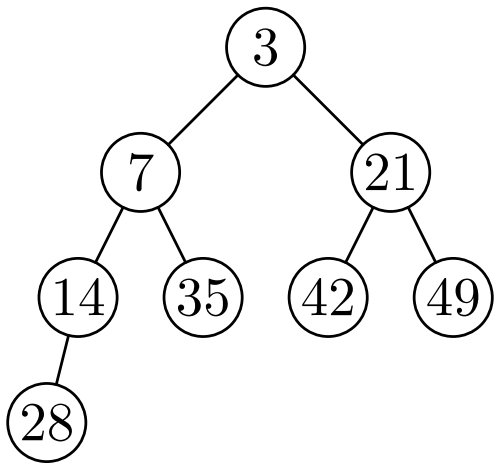
So wird der Heap schrittweise aufgebaut.

Wir bilden einen Heap aus den Schlüsseln 7, 14, 21, 28, 35, 42, 49, 3, 10, 17, 24, 31, und 38:



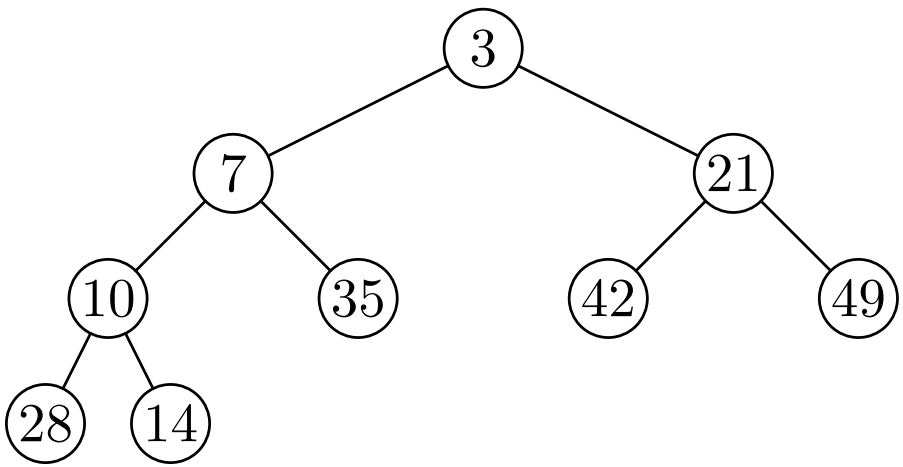
So wird der Heap schrittweise aufgebaut.

Wir bilden einen Heap aus den Schlüsseln 7, 14, 21, 28, 35, 42, 49, 3, 10, 17, 24, 31, und 38:



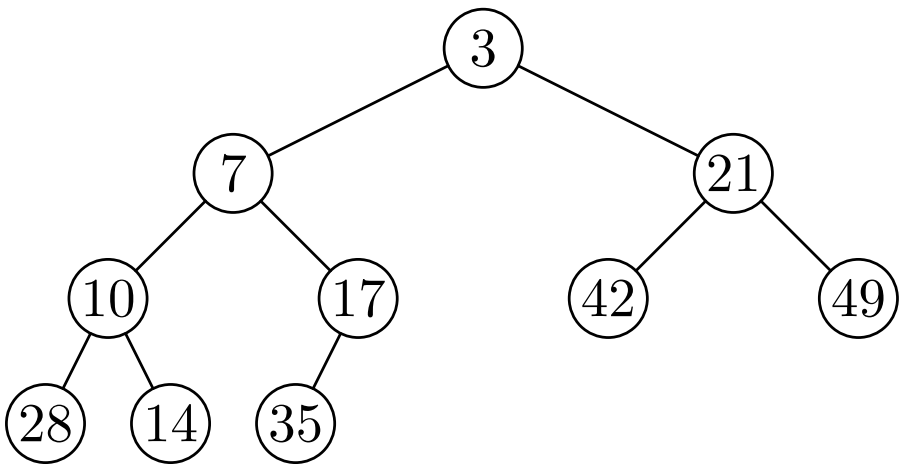
So wird der Heap schrittweise aufgebaut.

Wir bilden einen Heap aus den Schlüsseln 7, 14, 21, 28, 35, 42, 49, 3, 10, 17, 24, 31, und 38:



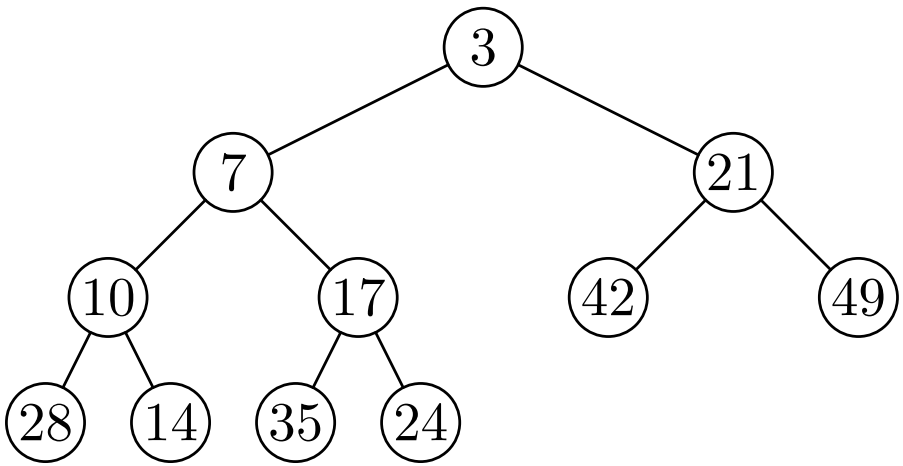
So wird der Heap schrittweise aufgebaut.

Wir bilden einen Heap aus den Schlüsseln 7, 14, 21, 28, 35, 42, 49, 3, 10, 17, 24, 31, und 38:



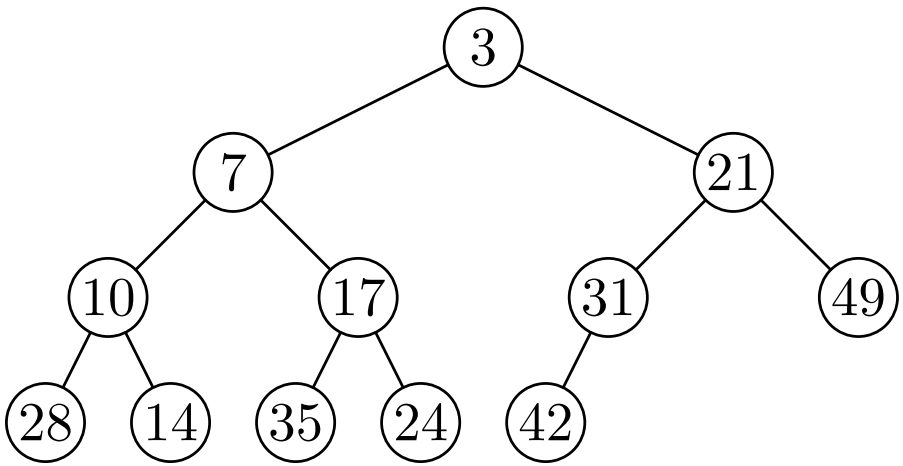
So wird der Heap schrittweise aufgebaut.

Wir bilden einen Heap aus den Schlüsseln 7, 14, 21, 28, 35, 42, 49, 3, 10, 17, 24, 31, und 38:



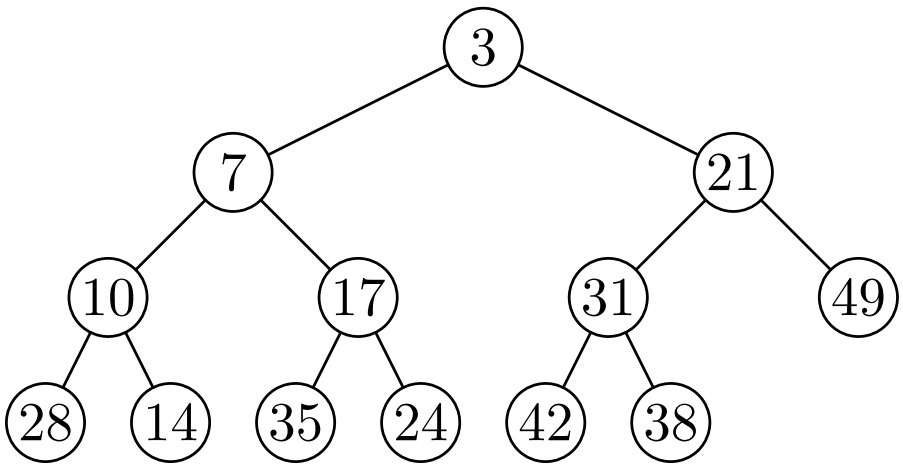
So wird der Heap schrittweise aufgebaut.

Wir bilden einen Heap aus den Schlüsseln 7, 14, 21, 28, 35, 42, 49, 3, 10, 17, 24, 31, und 38:

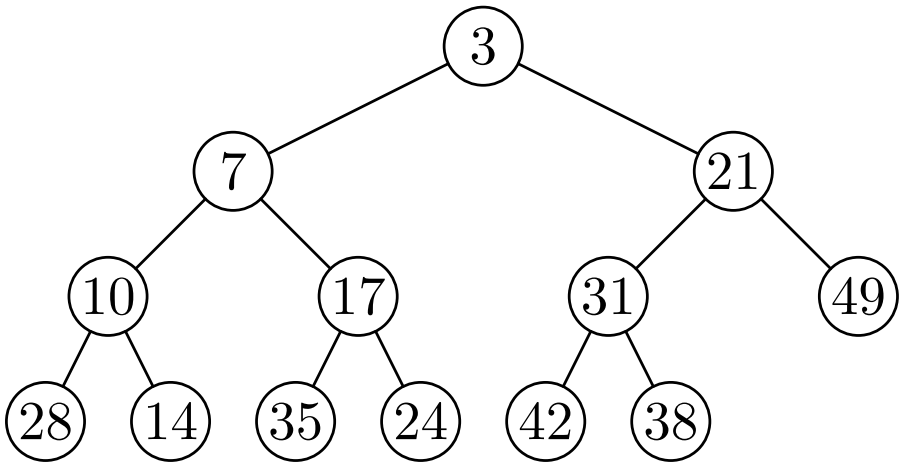


So wird der Heap schrittweise aufgebaut.

Wir bilden einen Heap aus den Schlüsseln 7, 14, 21, 28, 35, 42, 49, 3, 10, 17, 24, 31, und 38:

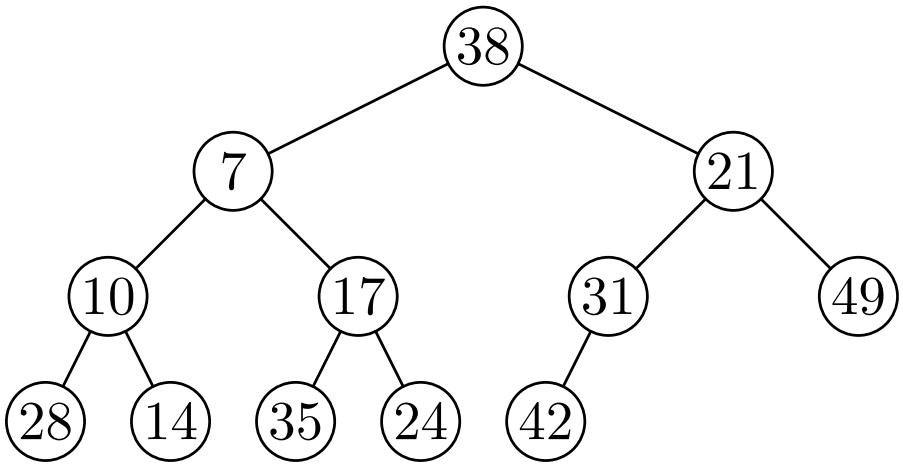


Entfernen der Wurzel – extract-min



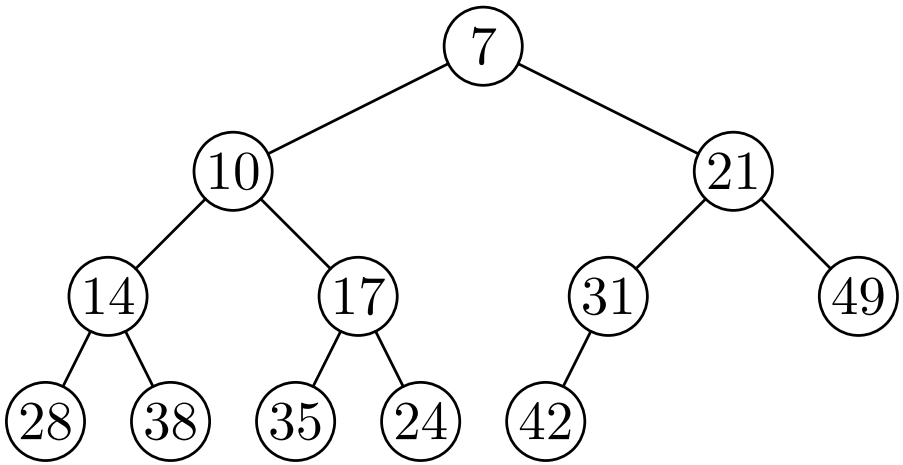
1. Ersetze den Schlüssel der Wurzel durch den Schlüssel des letzten Knoten
2. Lösche den letzten Knoten
3. Jetzt ist die Heap-Eigenschaft verletzt
4. Lasse den Schlüssel in der Wurzel hinuntersinken.

Entfernen der Wurzel – extract-min



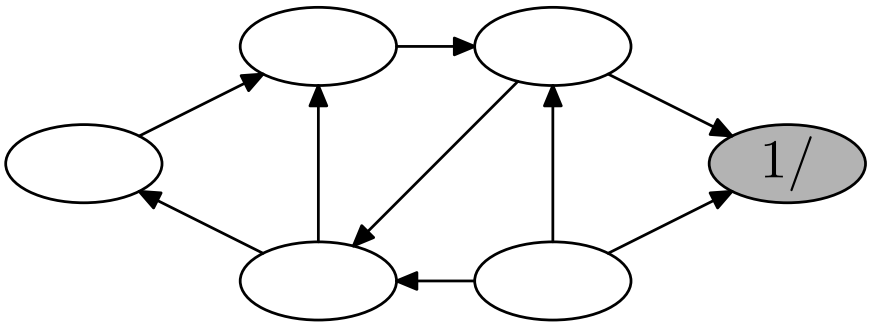
1. Ersetze den Schlüssel der Wurzel durch den Schlüssel des letzten Knoten
2. Lösche den letzten Knoten
3. Jetzt ist die Heap-Eigenschaft verletzt
4. Lasse den Schlüssel in der Wurzel hinuntersinken.

Entfernen der Wurzel – extract-min



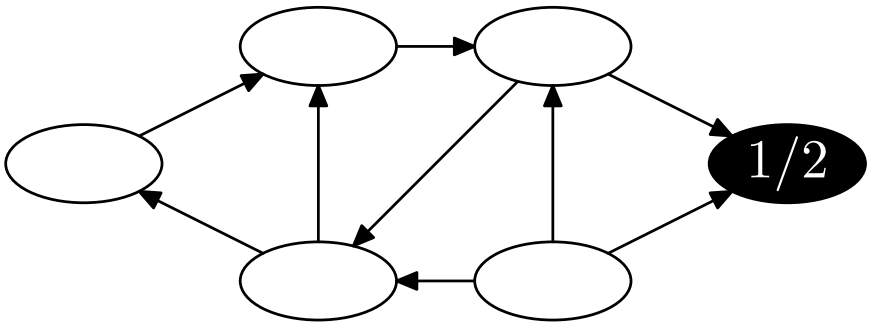
1. Ersetze den Schlüssel der Wurzel durch den Schlüssel des letzten Knoten
2. Lösche den letzten Knoten
3. Jetzt ist die Heap-Eigenschaft verletzt
4. Lasse den Schlüssel in der Wurzel hinuntersinken.

Tiefensuche – Beispiel



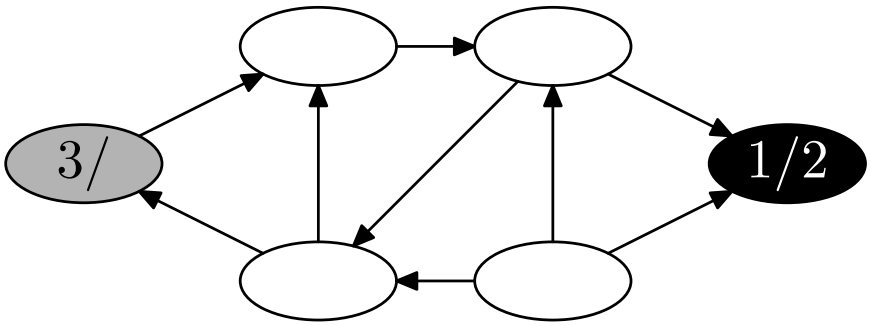
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



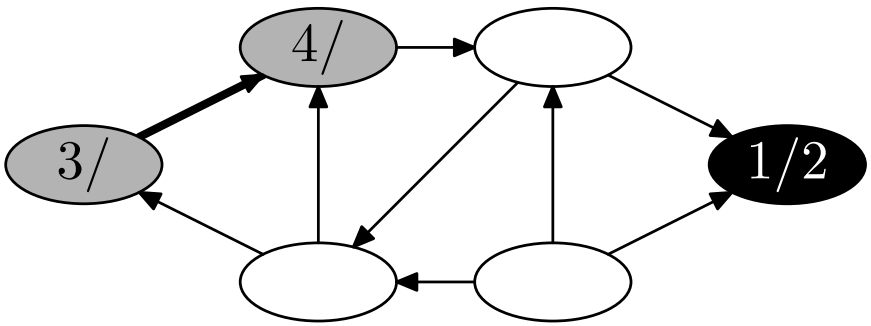
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



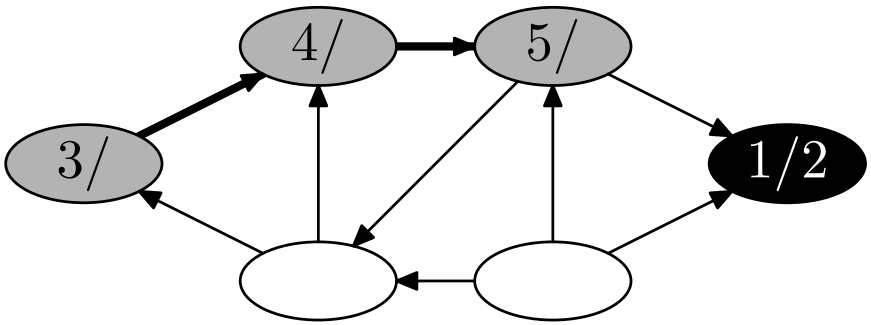
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



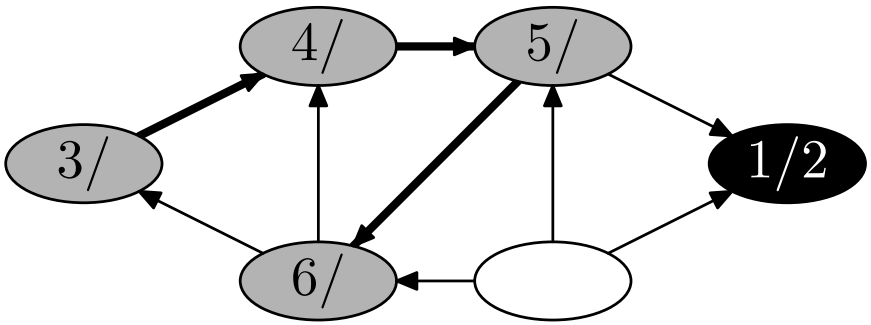
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



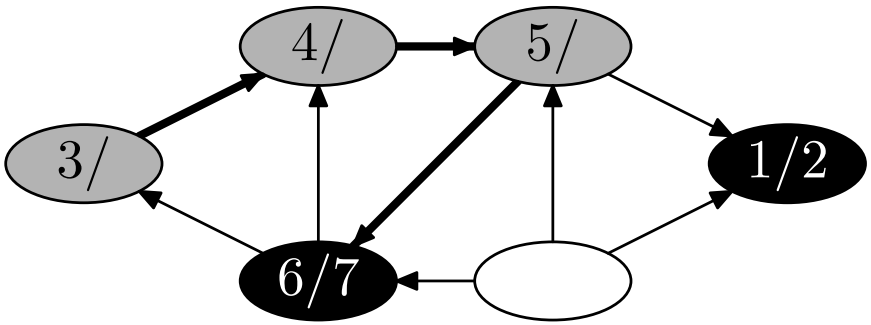
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



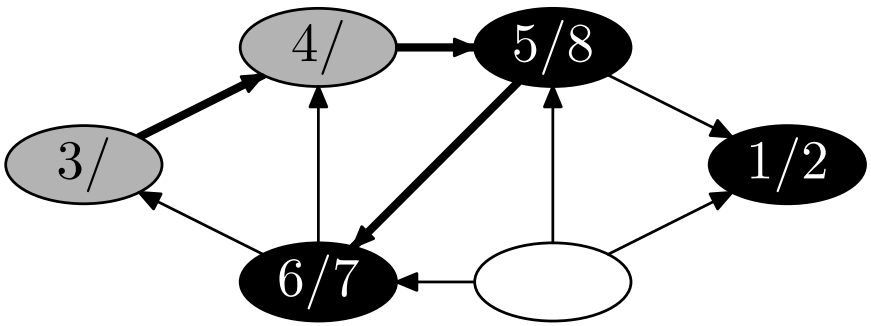
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



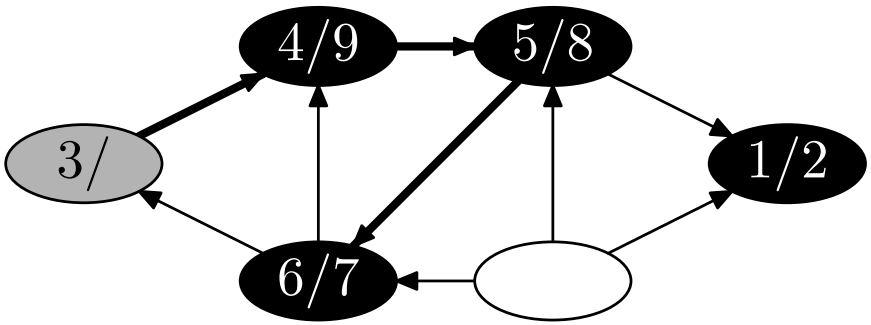
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



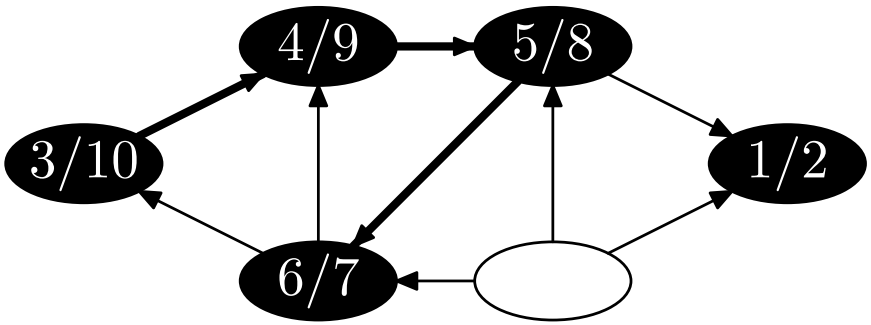
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



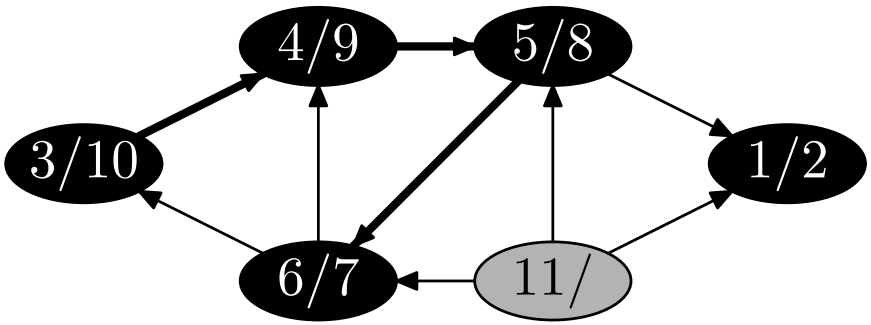
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



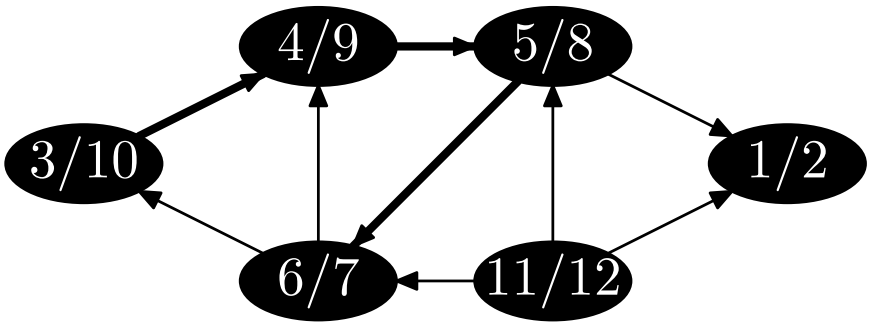
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



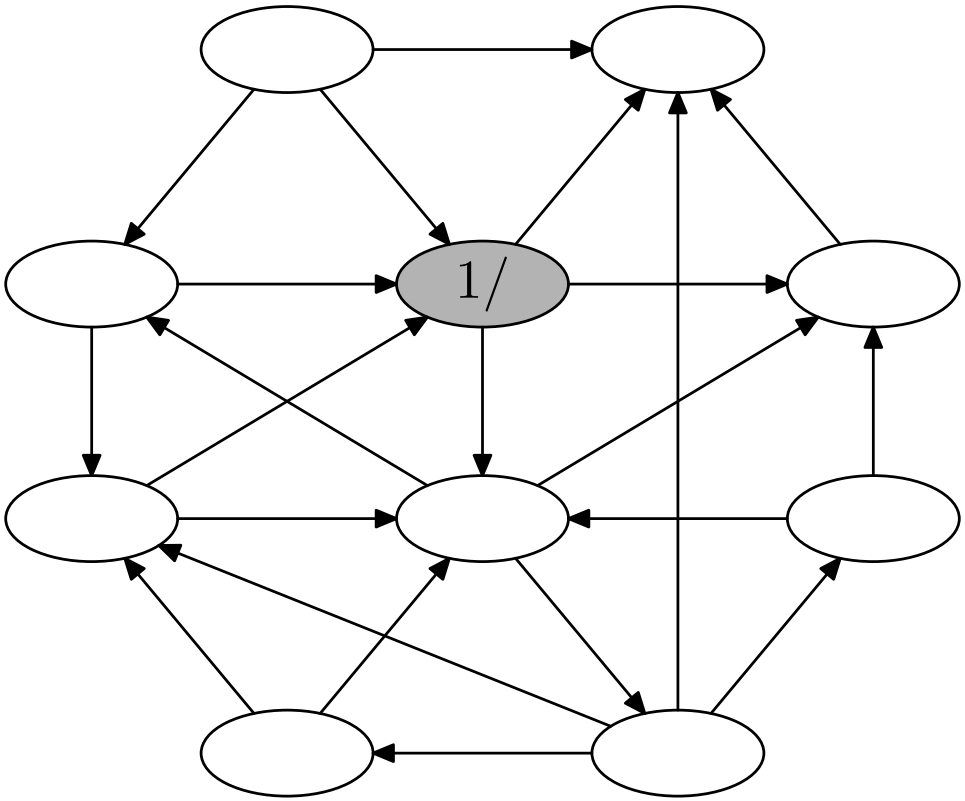
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Tiefensuche – Beispiel



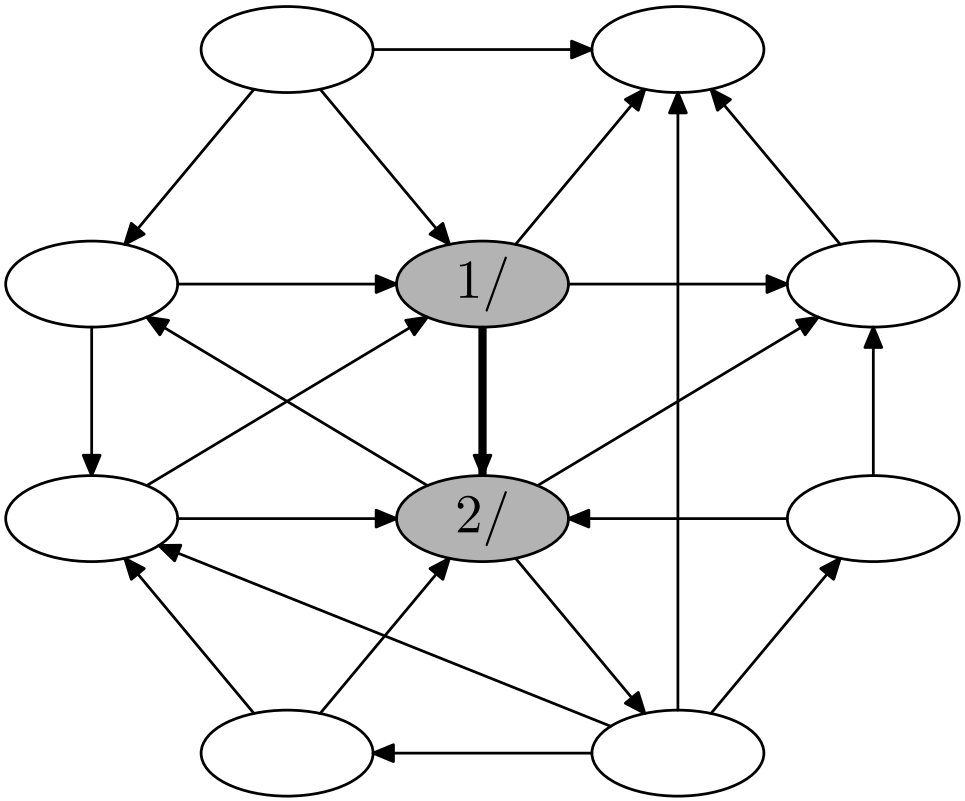
- Die Kanten des Tiefensuchwaldes sind dick dargestellt.
- Ein Knoten ist anfangs weiß.
- Ein Knoten ist grau, während er aktiv ist.
- Danach wird er schwarz.

Taxonomie der Kanten



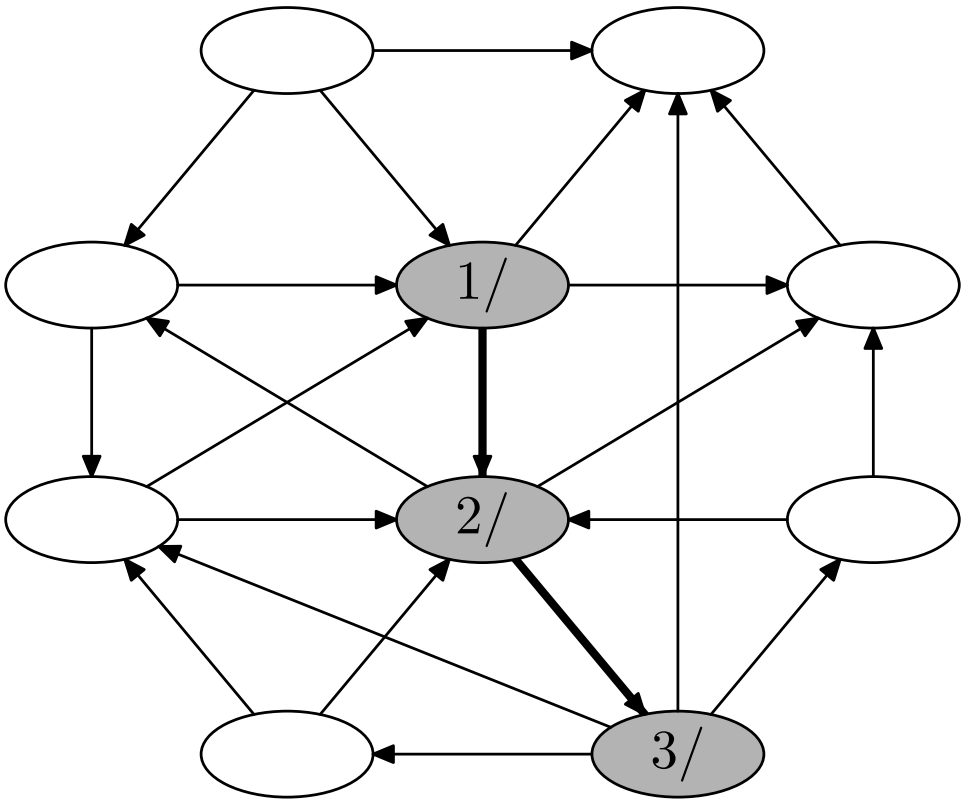
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



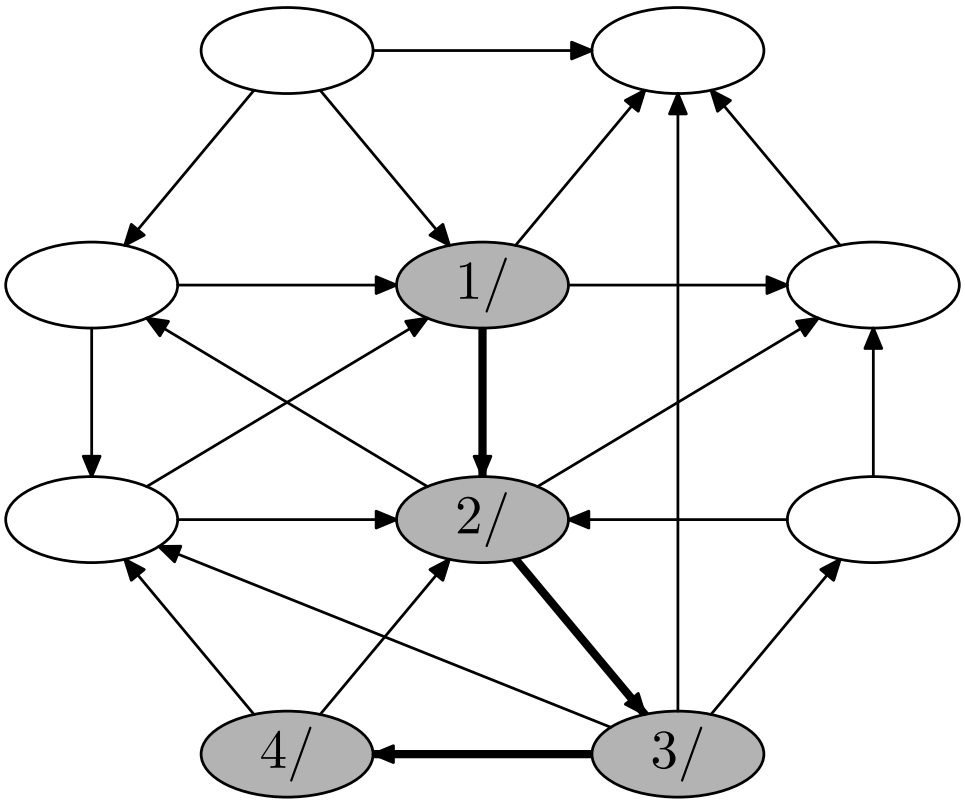
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



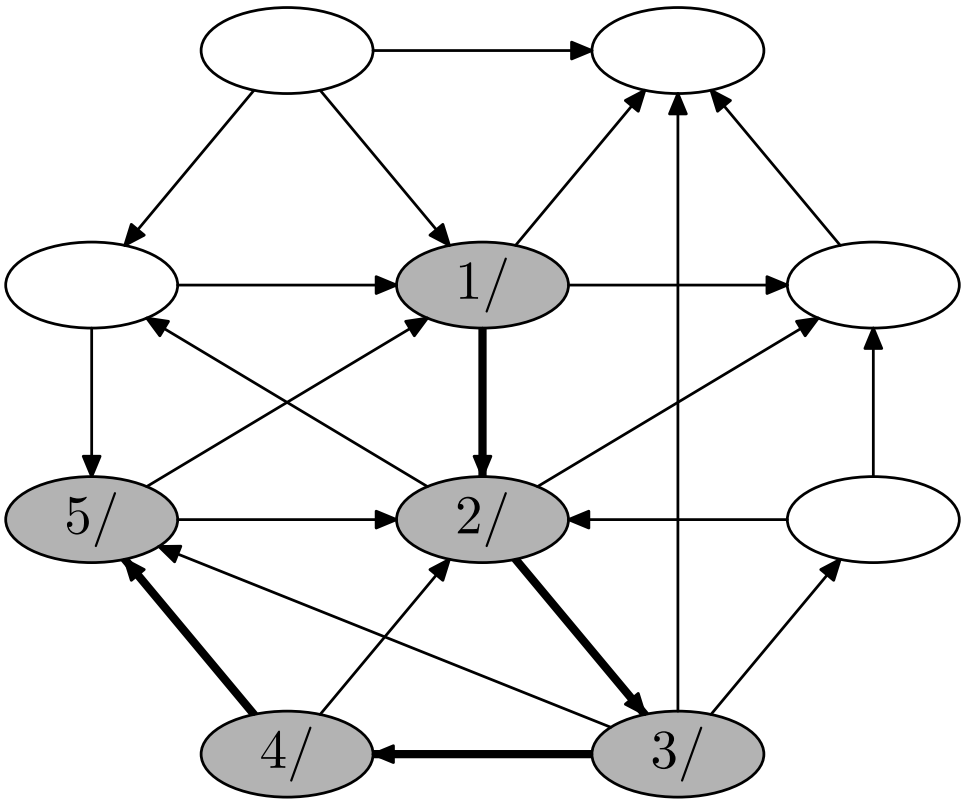
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



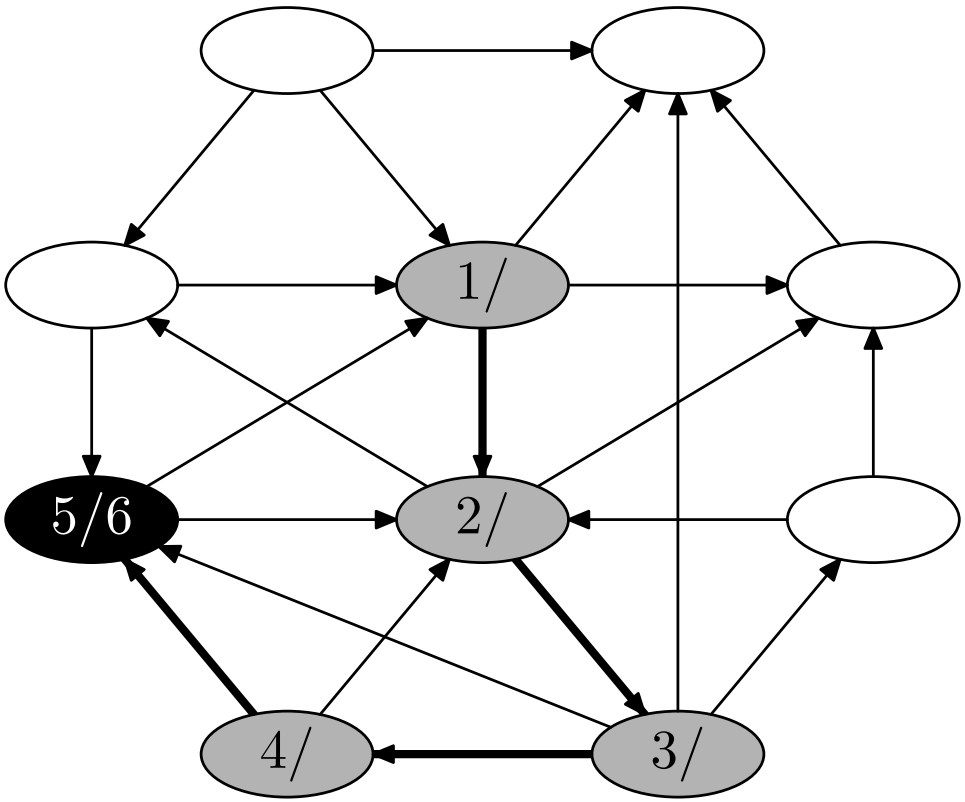
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



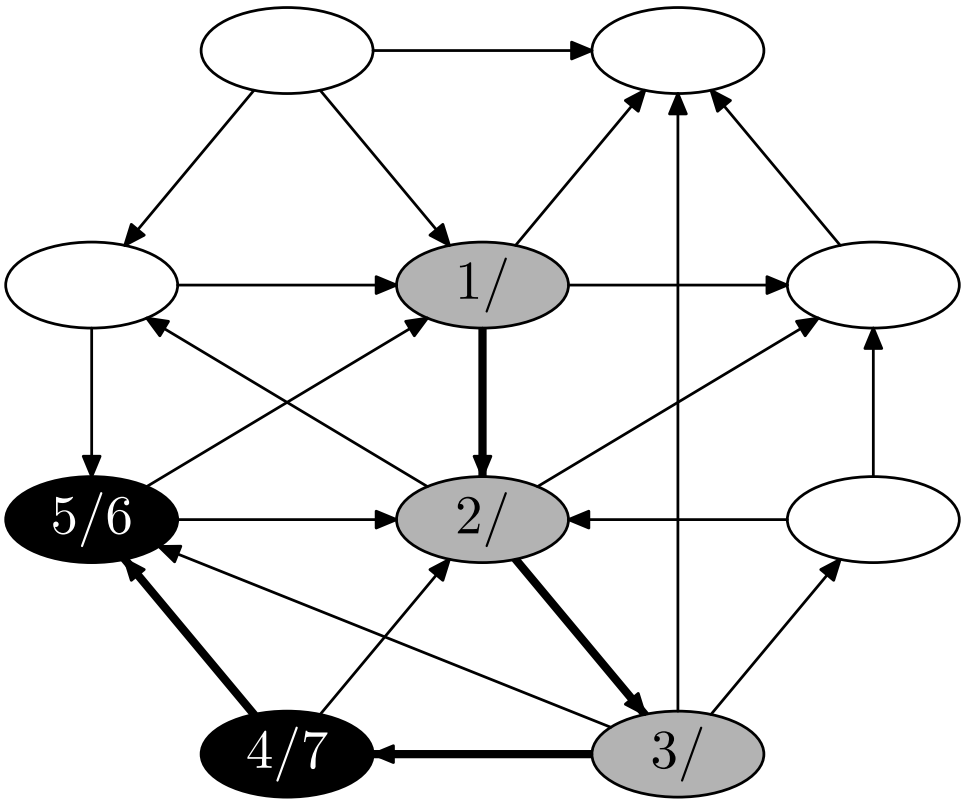
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



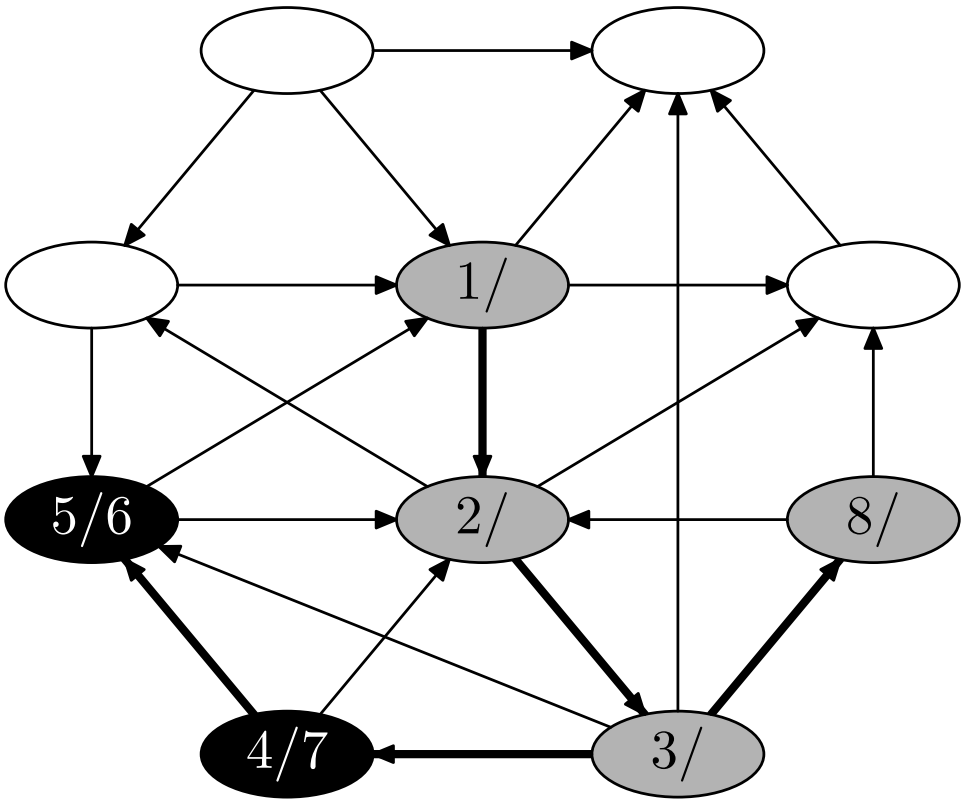
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



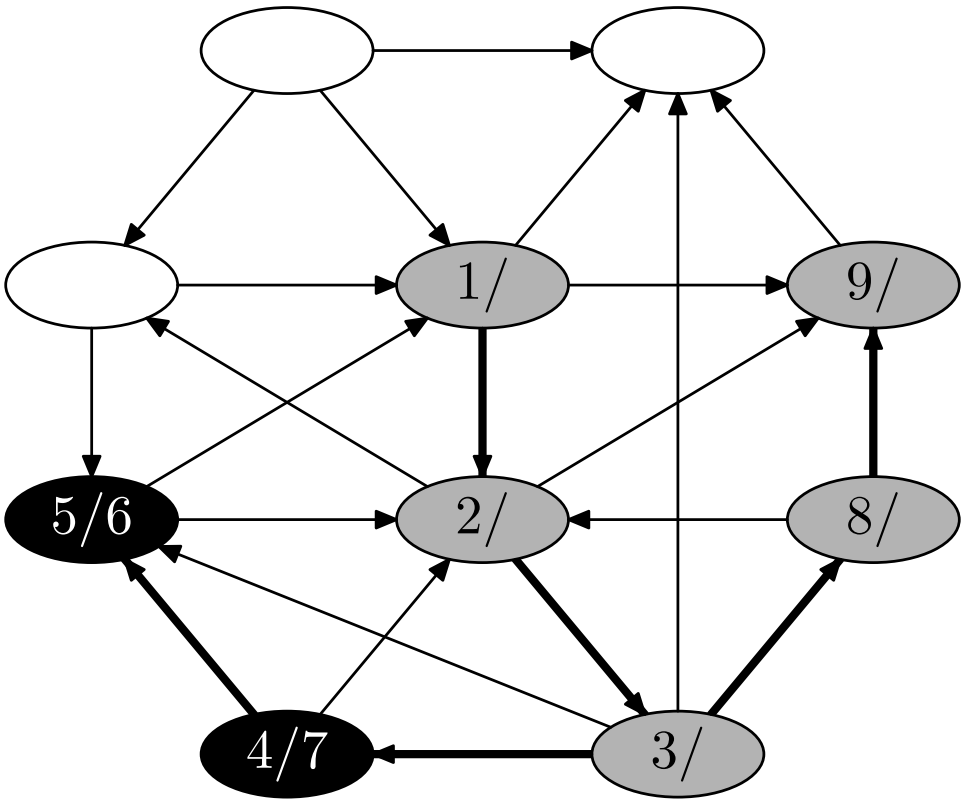
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



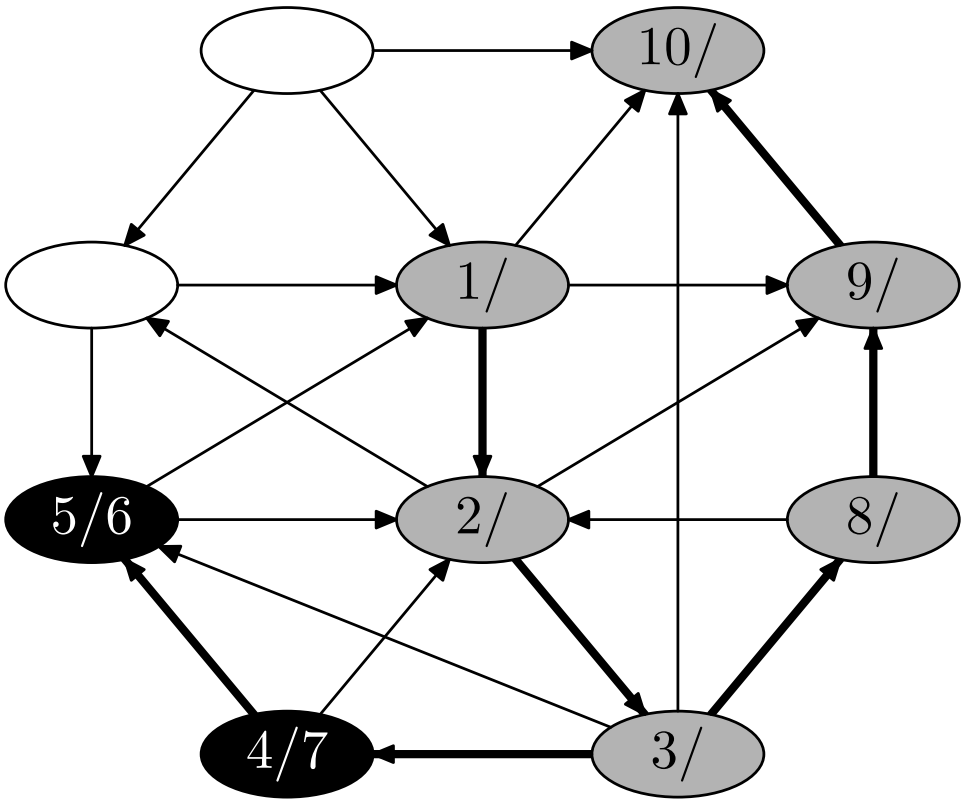
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



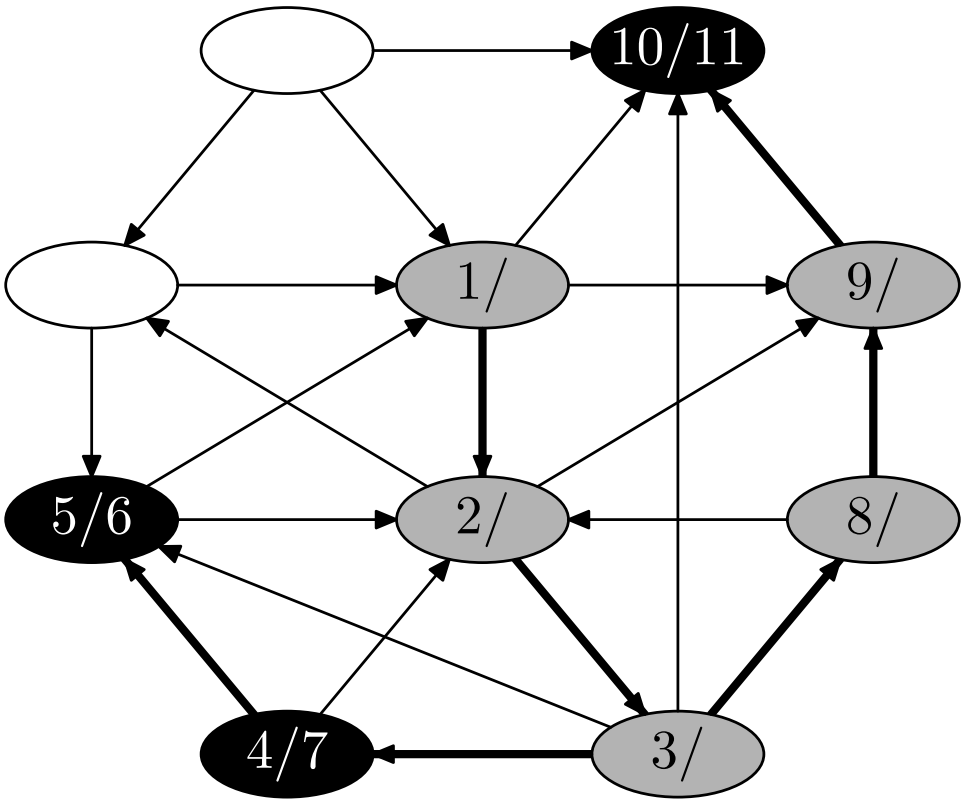
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



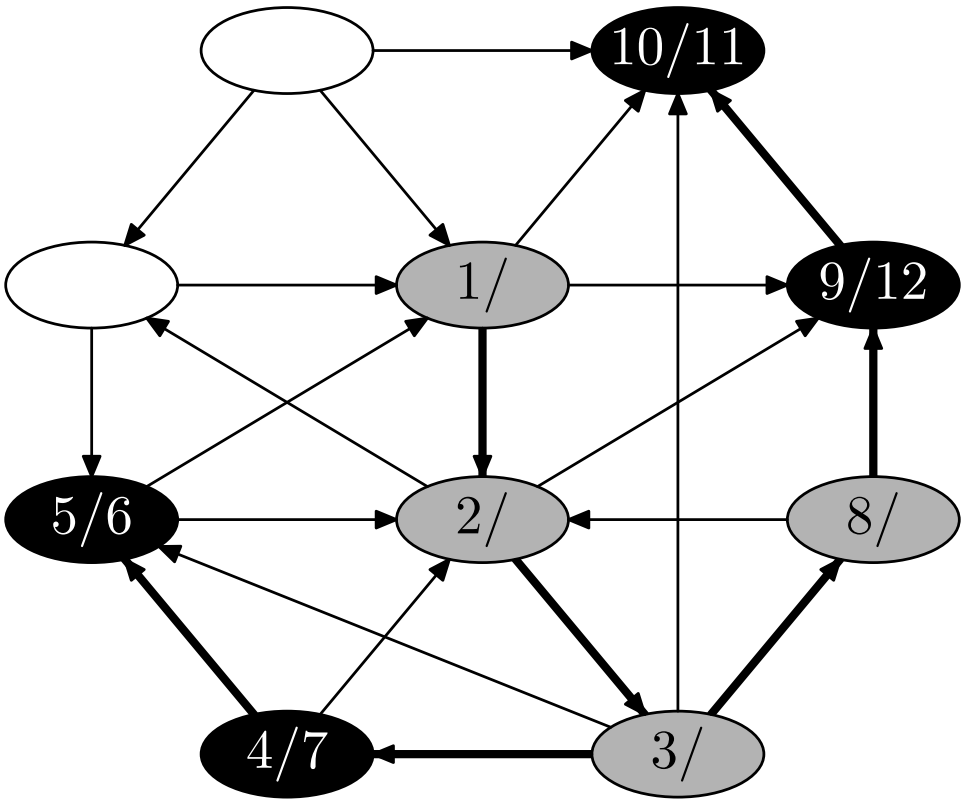
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



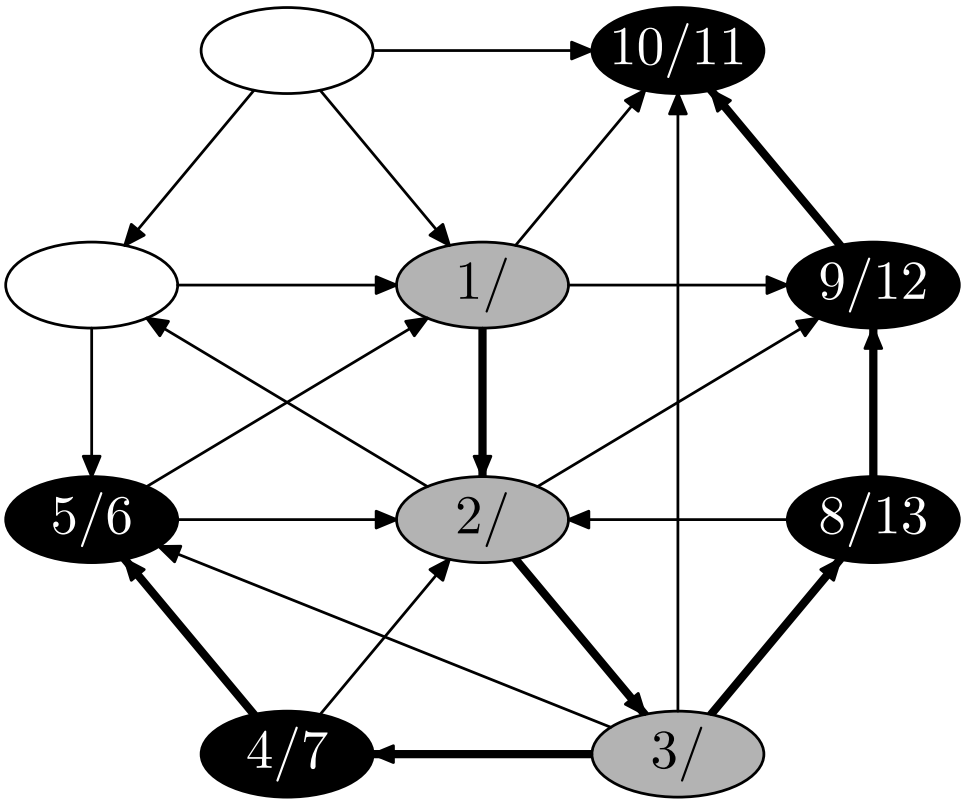
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



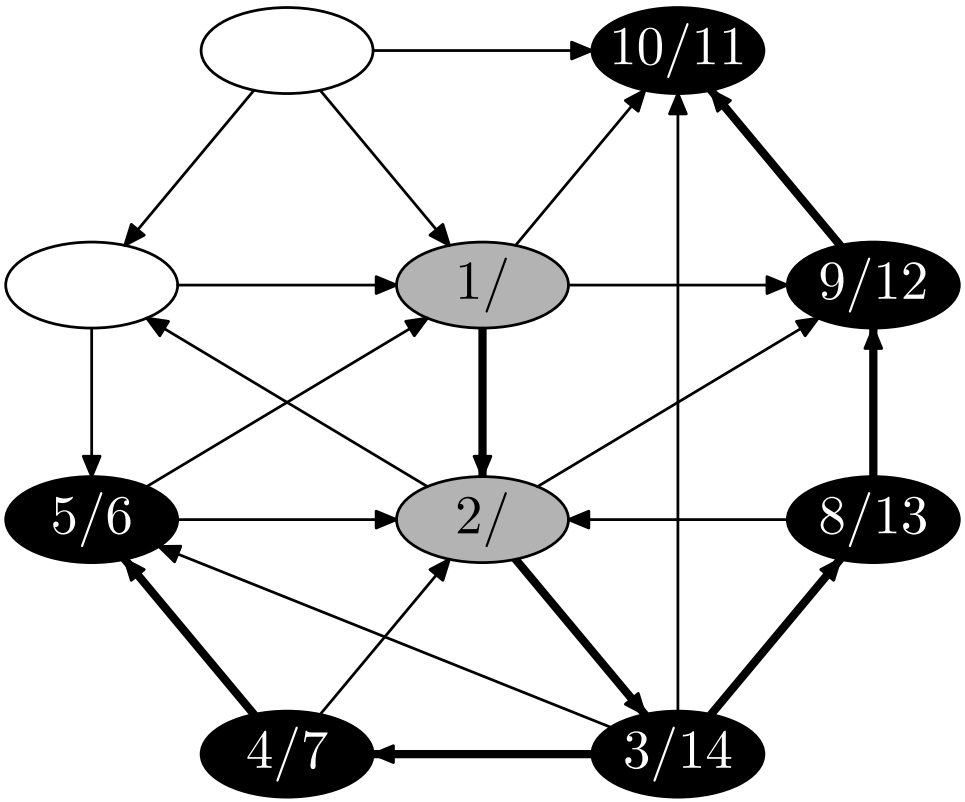
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



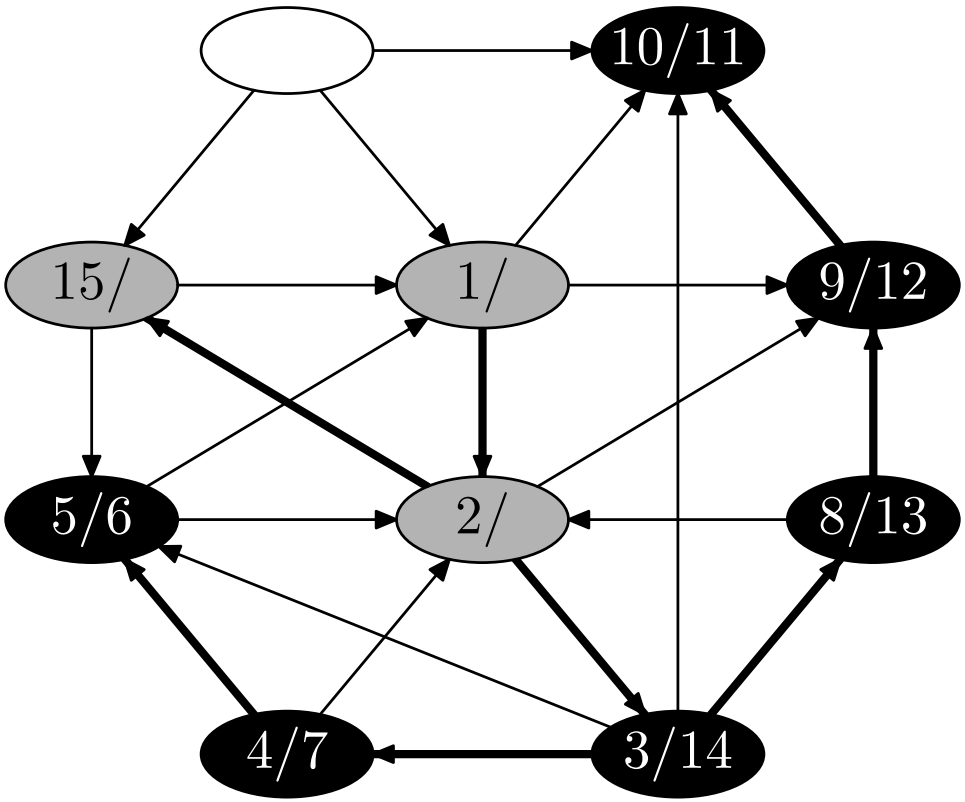
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



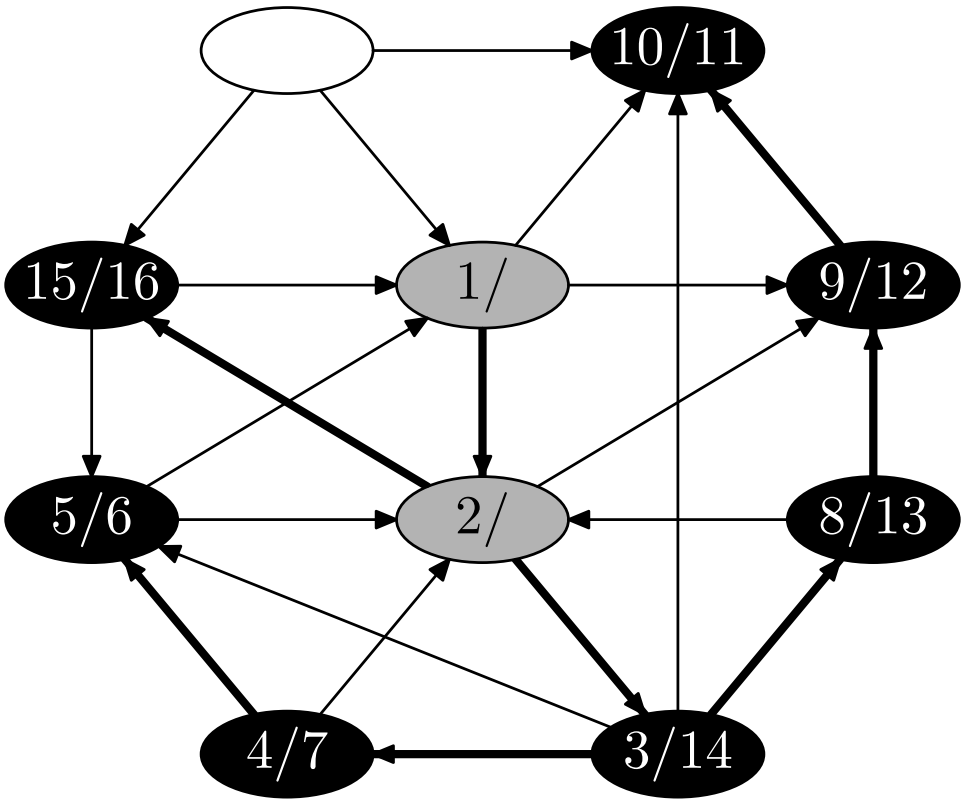
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



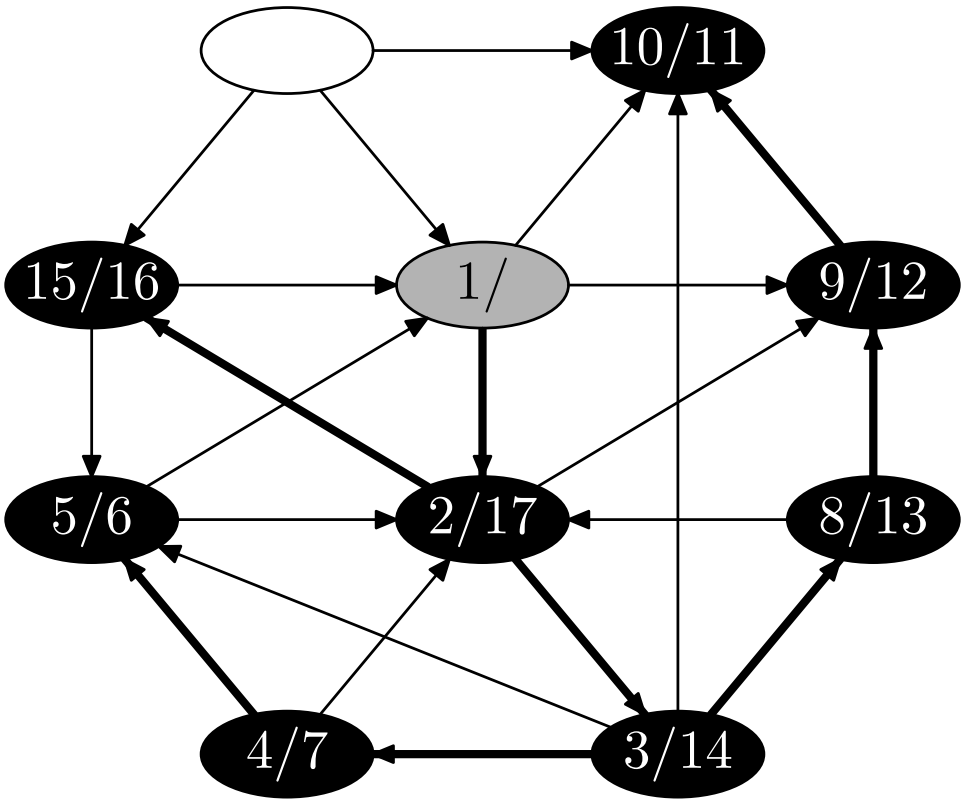
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



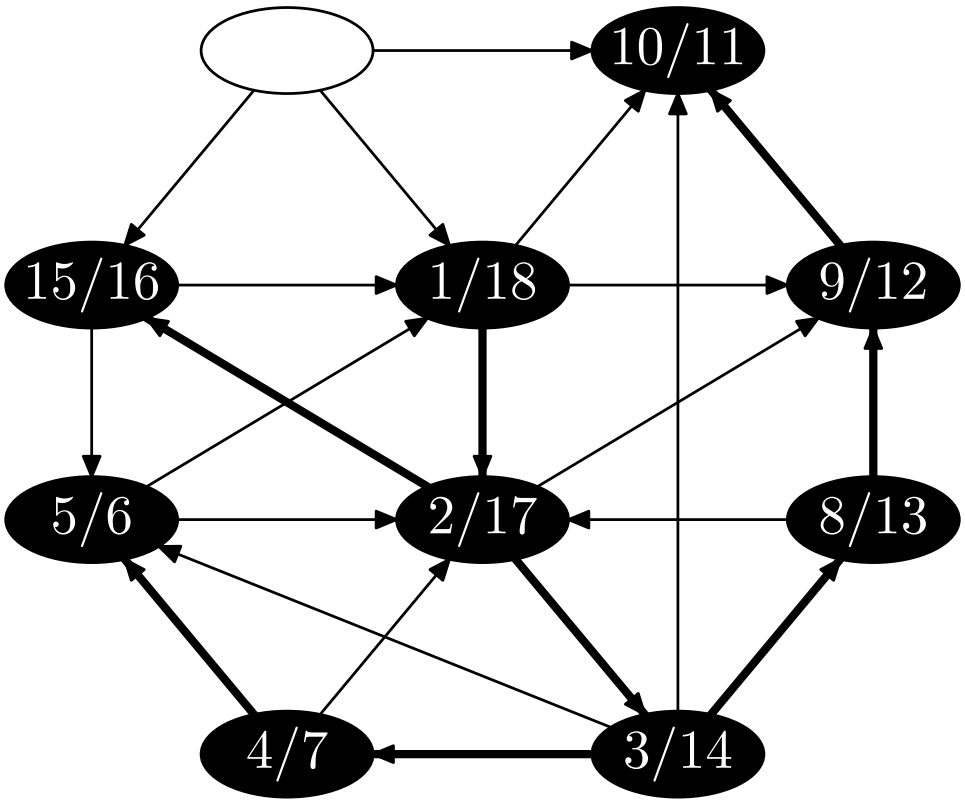
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



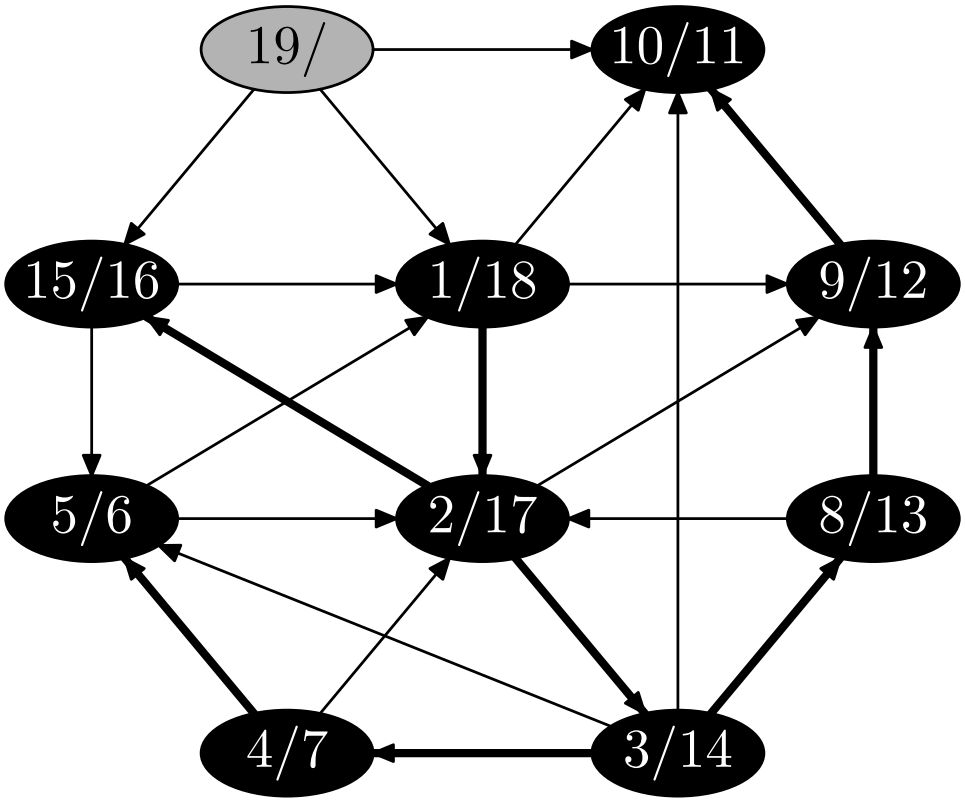
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



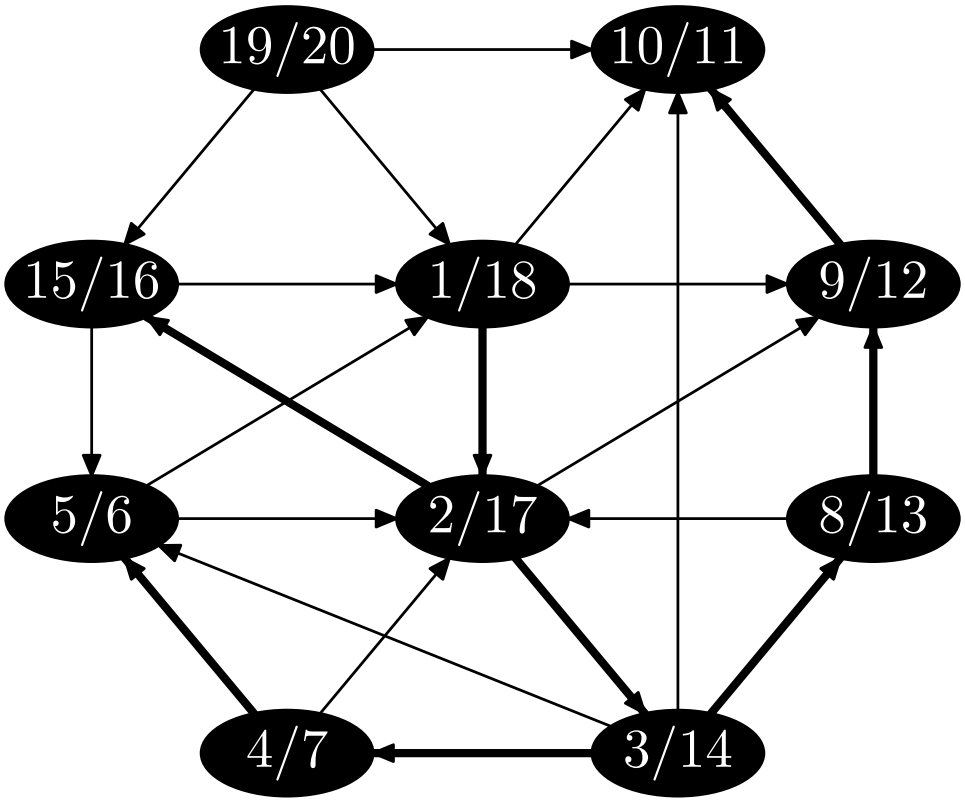
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



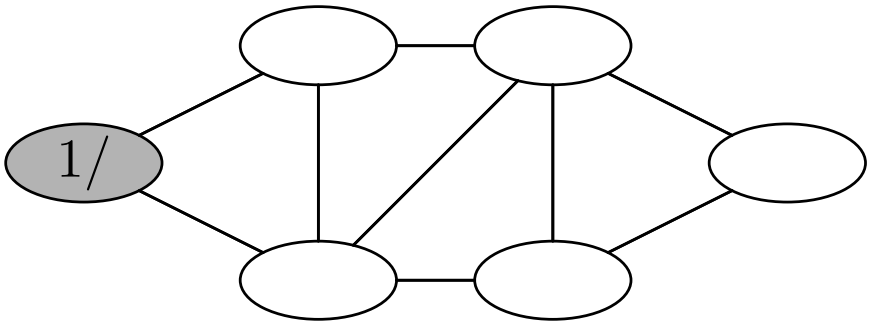
Frage: Welchen Typ hat jede Kante in diesem Beispiel?

Taxonomie der Kanten



Frage: Welchen Typ hat jede Kante in diesem Beispiel?

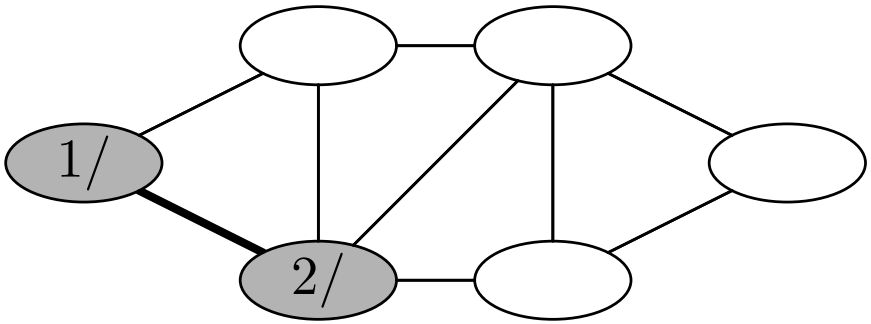
DFS – Ungerichtete Graphen



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

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

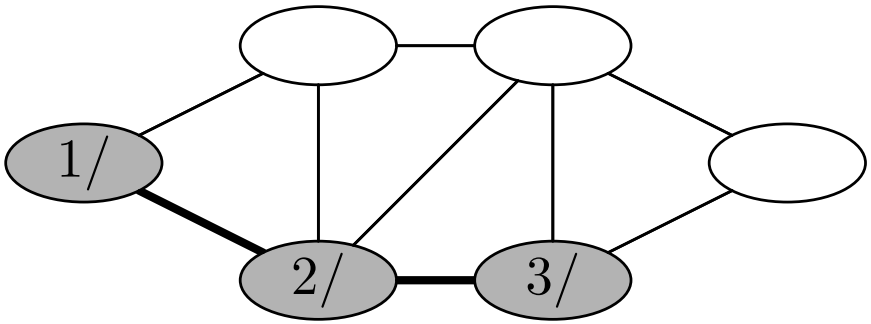
DFS – Ungerichtete Graphen



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

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

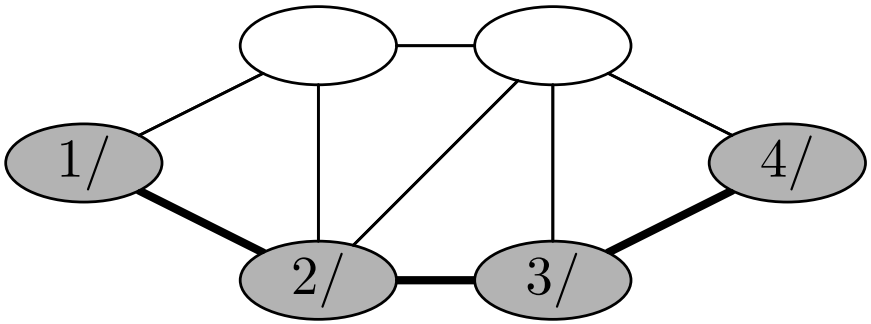
DFS – Ungerichtete Graphen



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

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

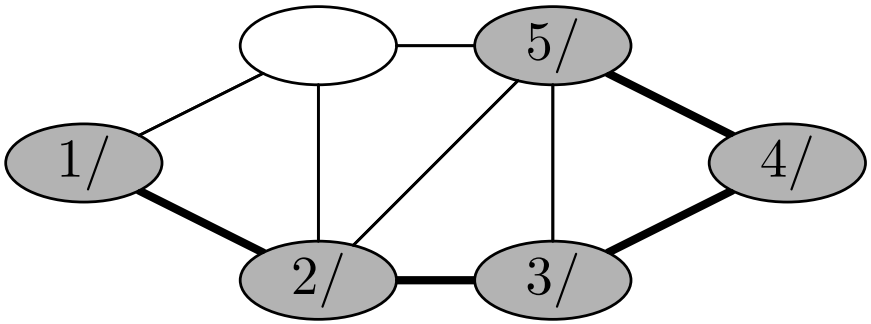
DFS – Ungerichtete Graphen



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

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

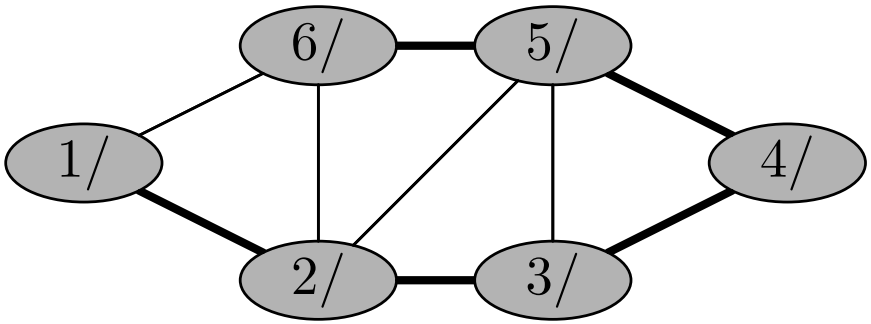
DFS – Ungerichtete Graphen



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

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

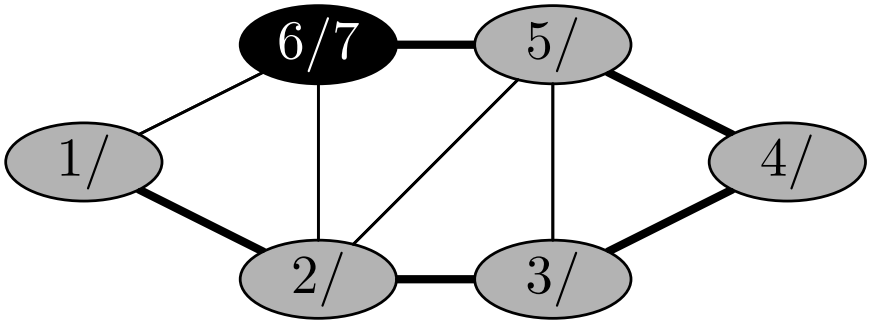
DFS – Ungerichtete Graphen



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

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

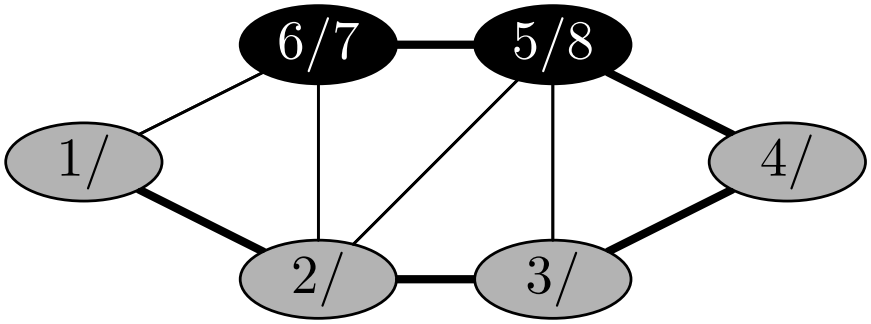
DFS – Ungerichtete Graphen



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

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

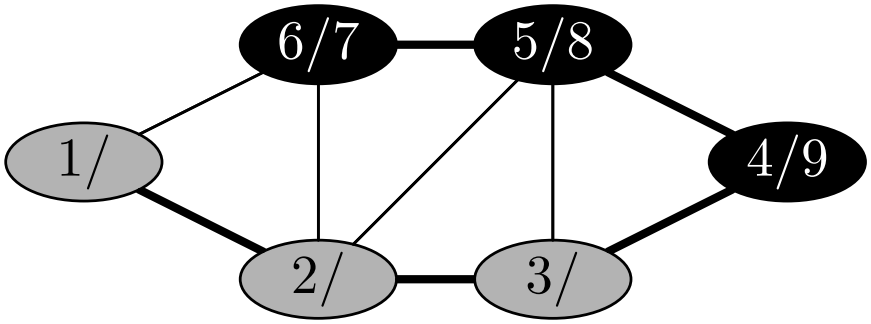
DFS – Ungerichtete Graphen



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

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

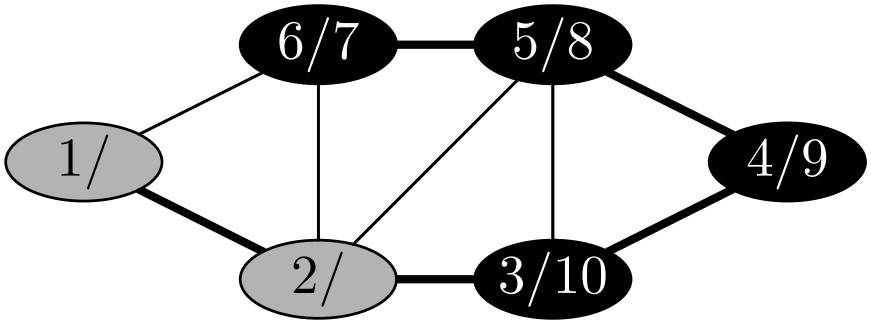
DFS – Ungerichtete Graphen



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

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

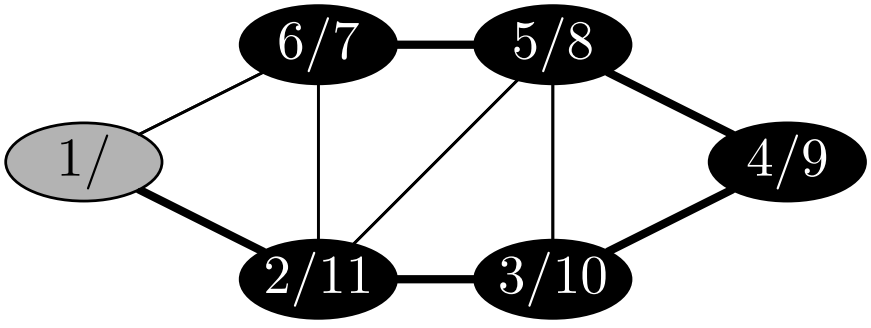
DFS – Ungerichtete Graphen



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

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

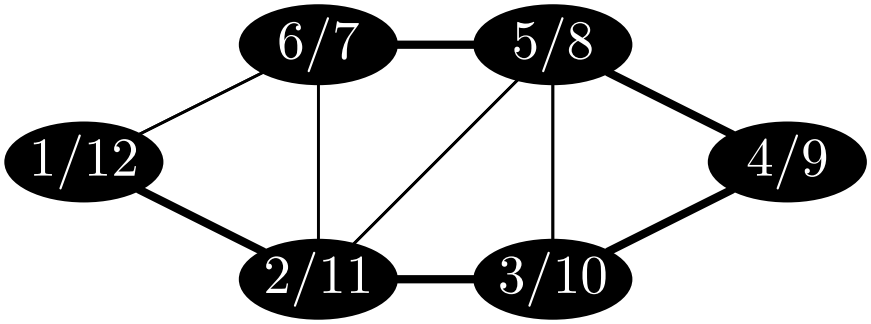
DFS – Ungerichtete Graphen



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.

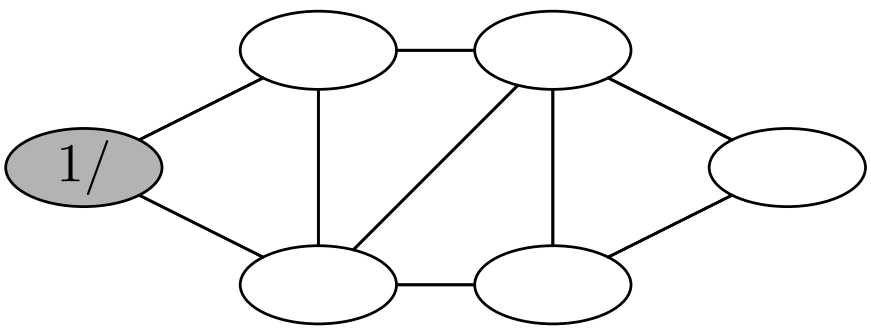
Zusammenhangskomponenten

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. □



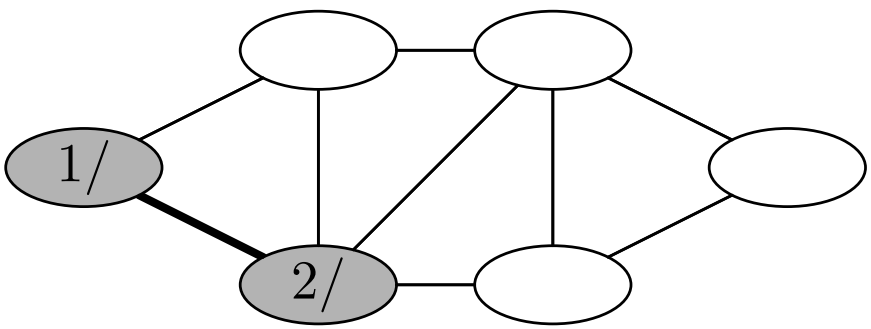
Zusammenhangskomponenten

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. □



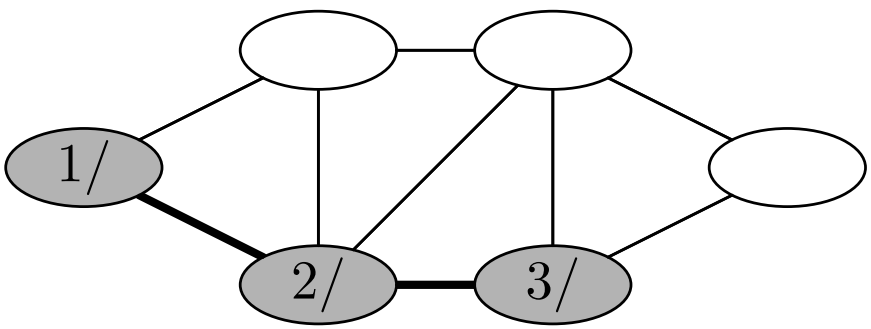
Zusammenhangskomponenten

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. □



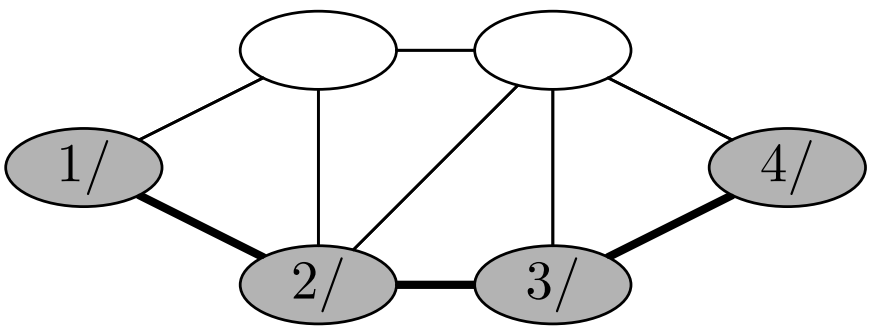
Zusammenhangskomponenten

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. □



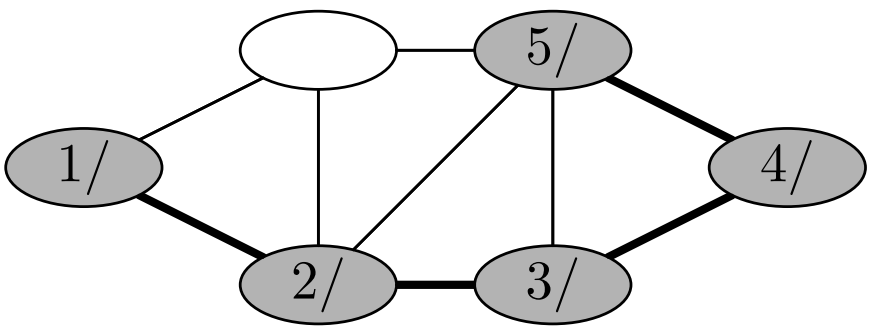
Zusammenhangskomponenten

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. □



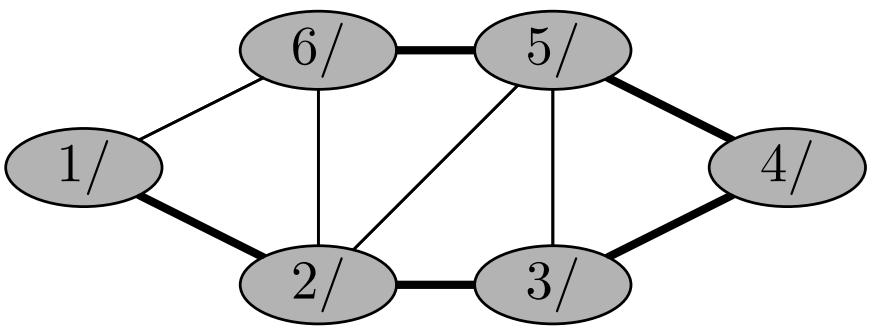
Zusammenhangskomponenten

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. □



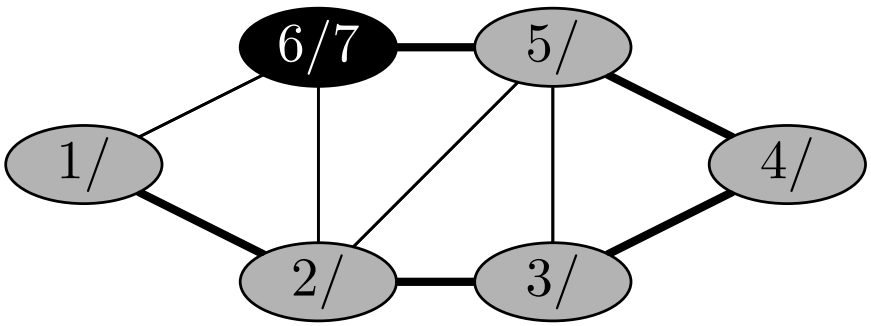
Zusammenhangskomponenten

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. □



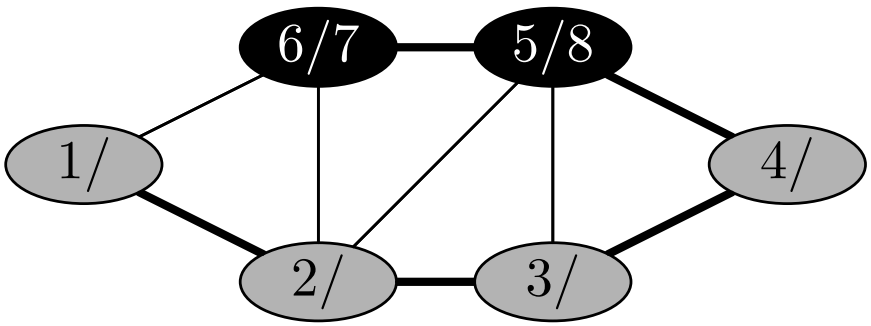
Zusammenhangskomponenten

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. □



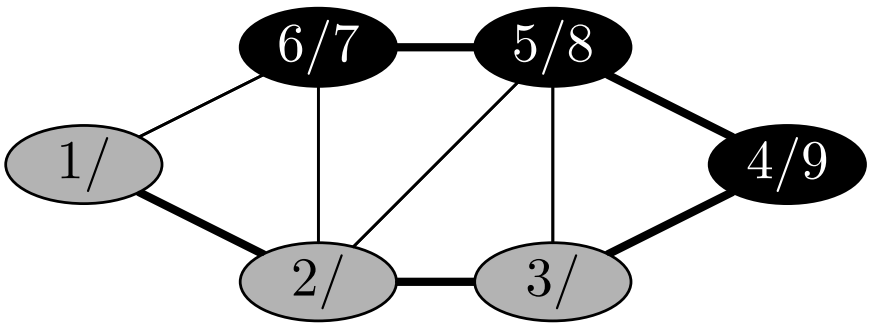
Zusammenhangskomponenten

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. □



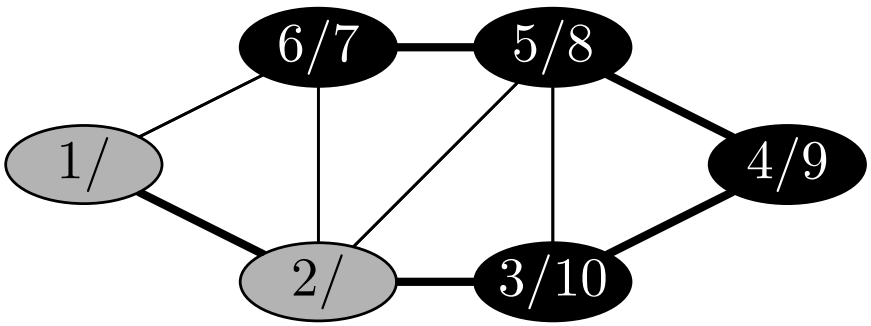
Zusammenhangskomponenten

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. □



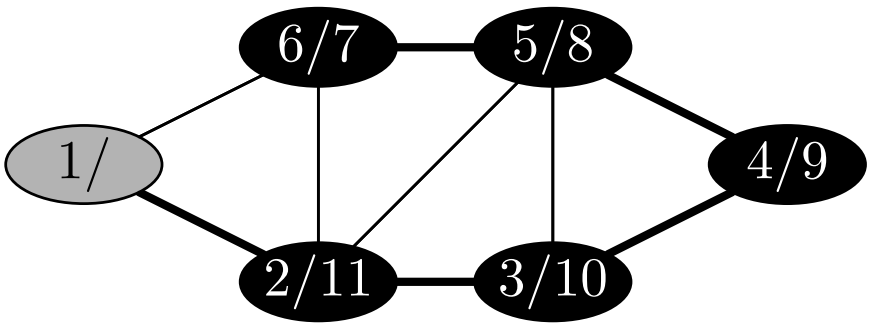
Zusammenhangskomponenten

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. □



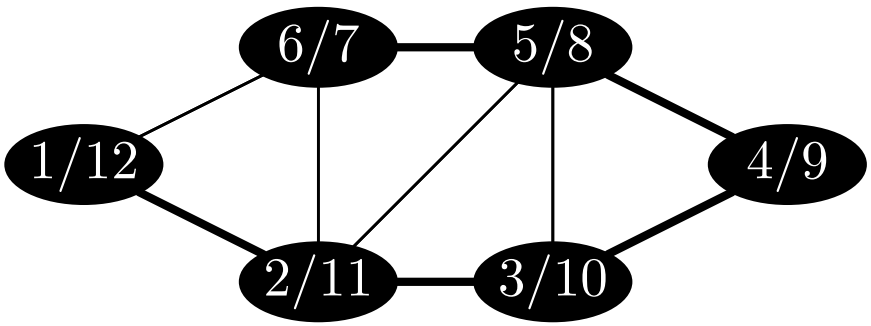
Zusammenhangskomponenten

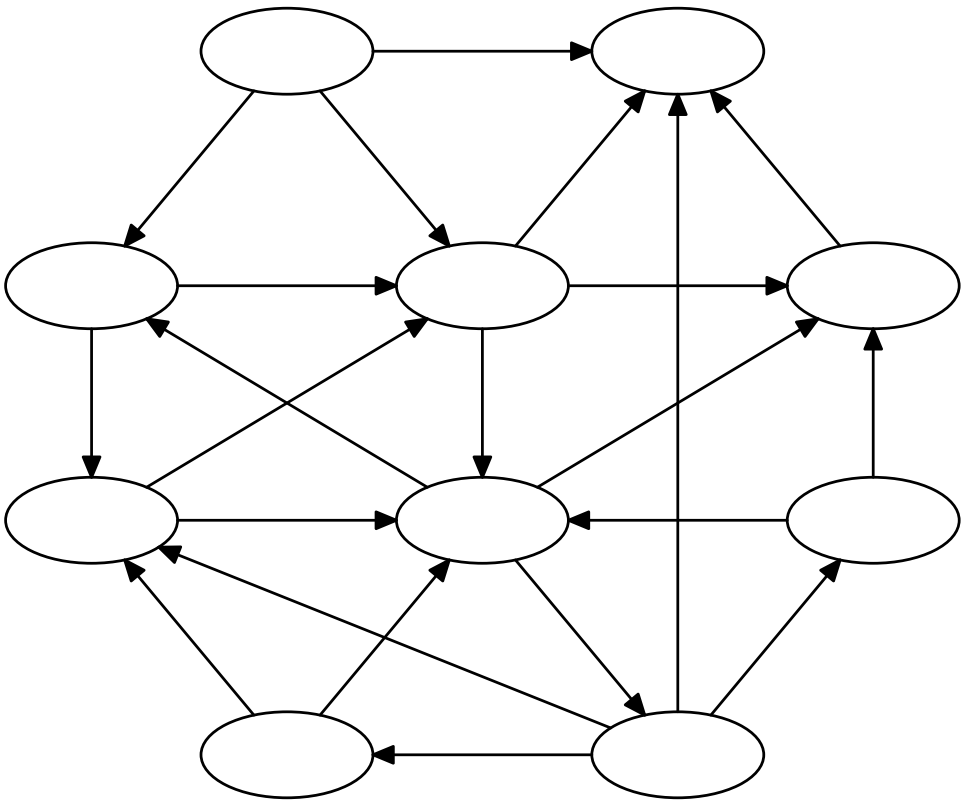
Theorem

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. □



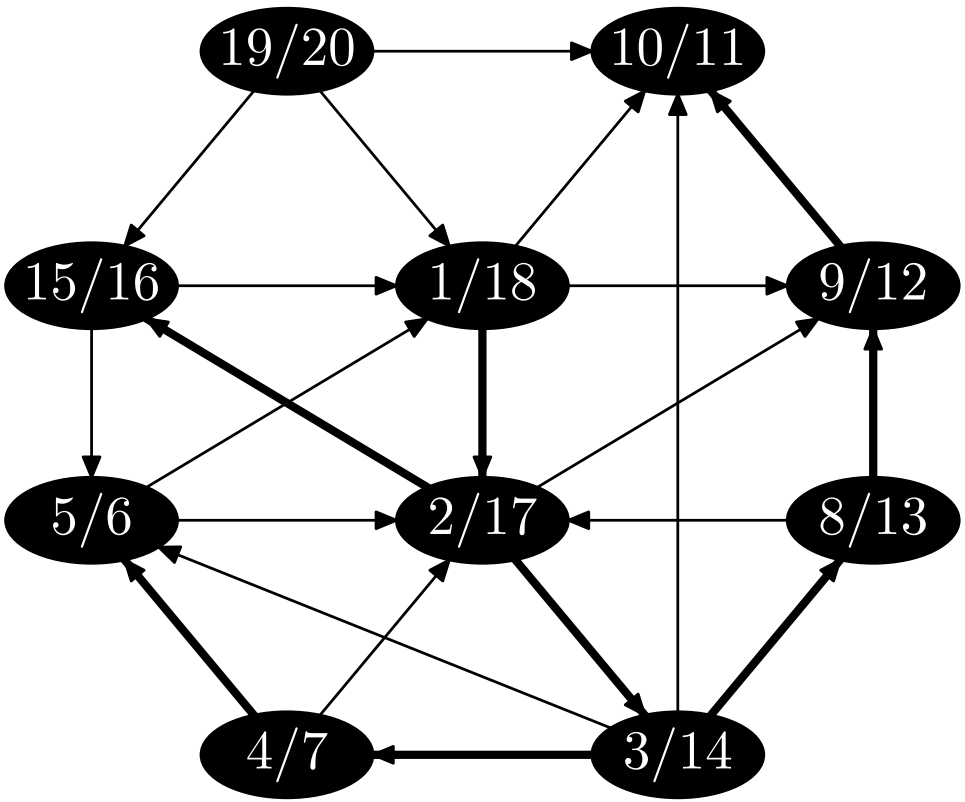


Lemma

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. □

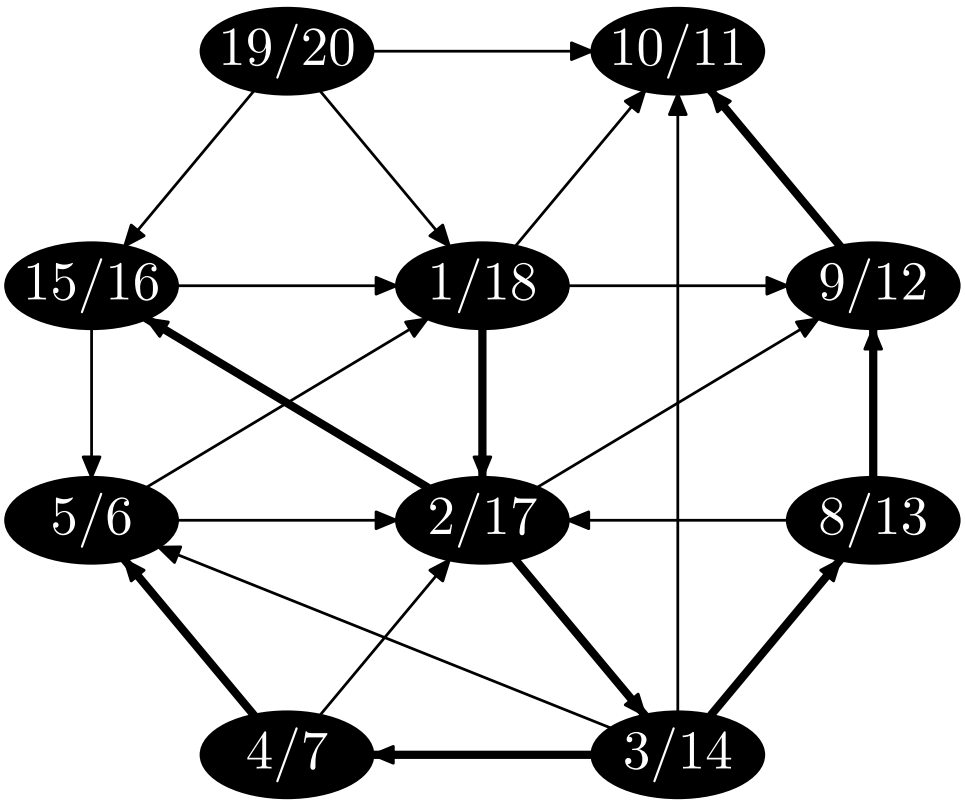


Lemma

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. □



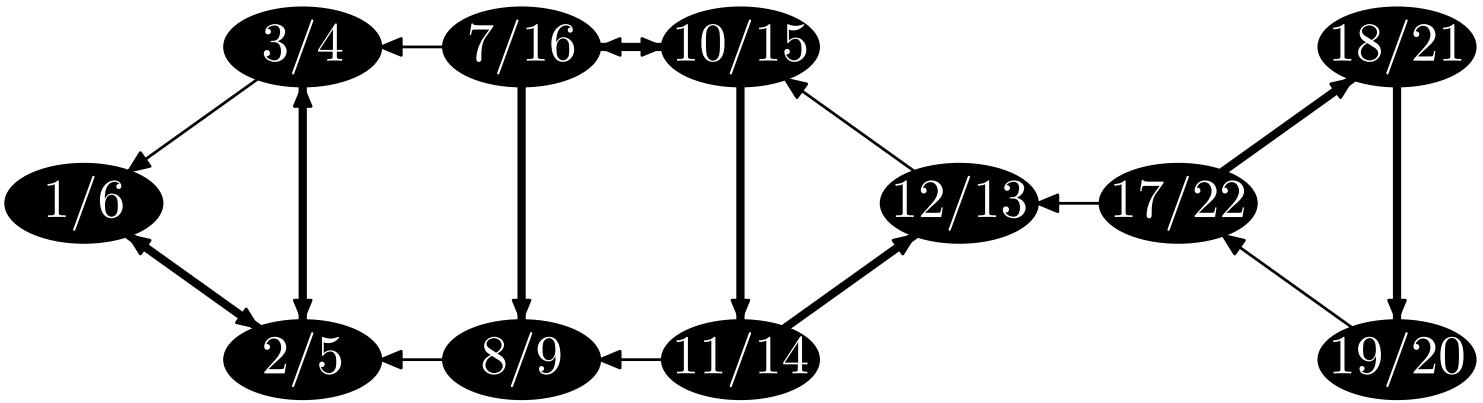
Lemma

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

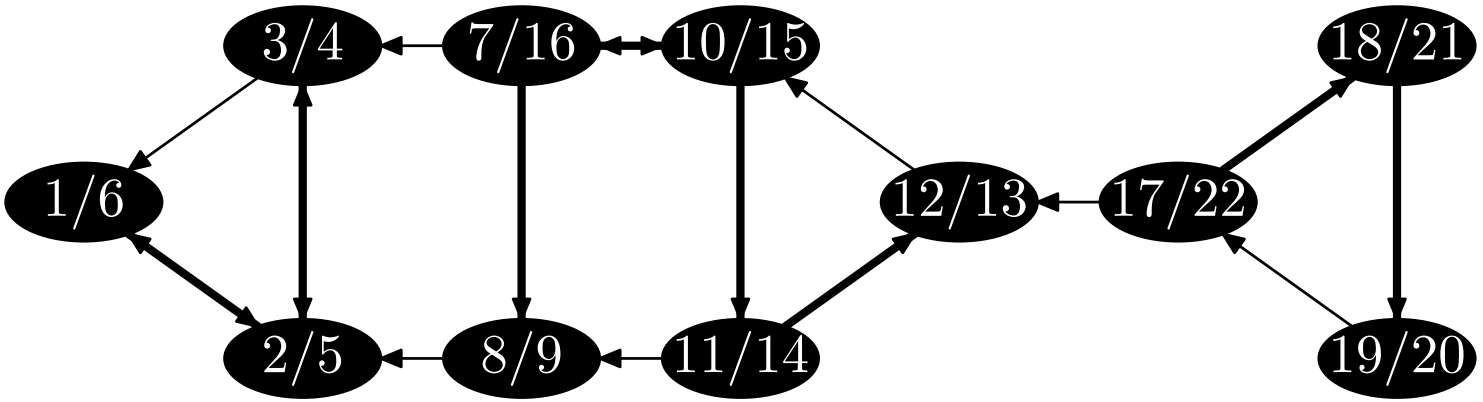
Beweis.

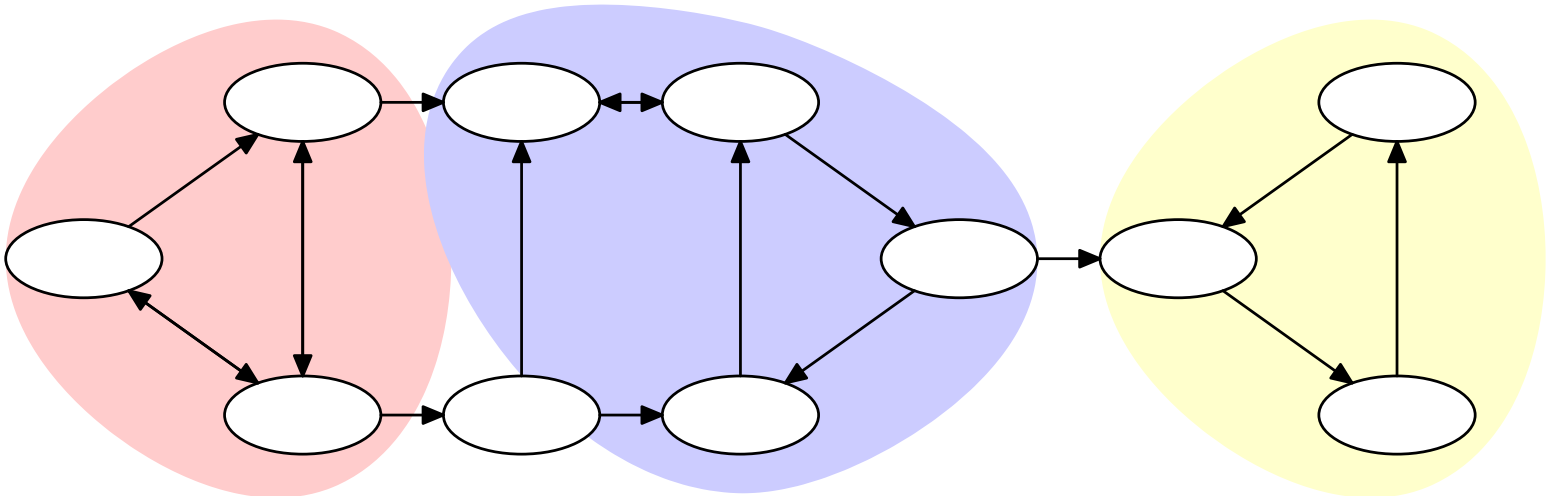
Er ist Wurzel eines DFS-Baums T. Wenn es einen anderen DFS-Baum gäbe, von dem eine Kante nach T führte, dann müßte dieser danach besucht werden. □

Starke Komponenten – Algorithmus von Kosaraju

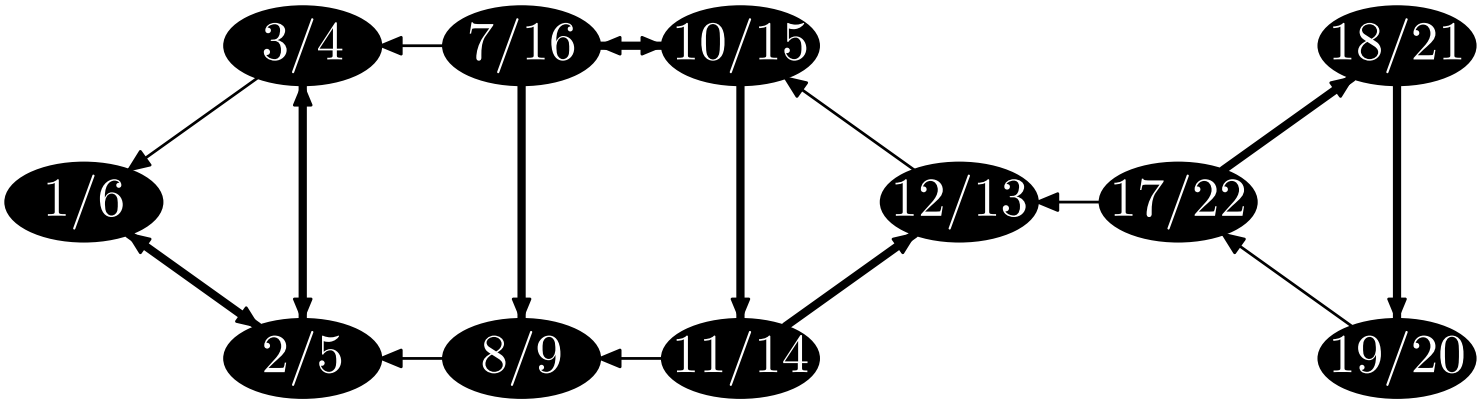


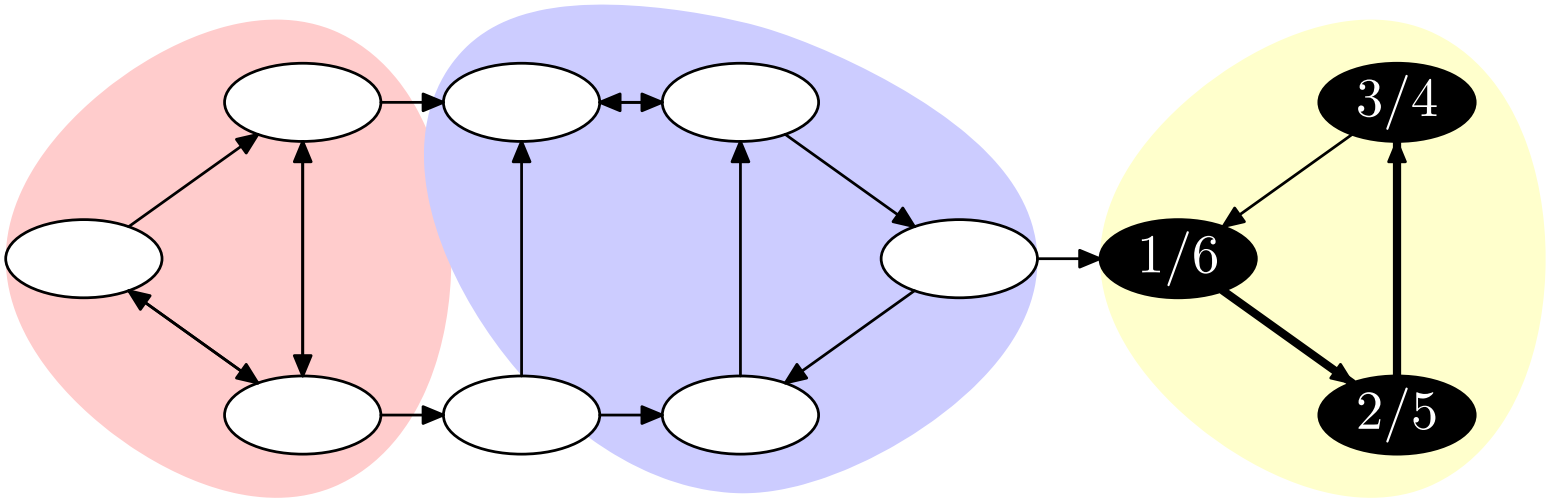
Starke Komponenten – Algorithmus von Kosaraju



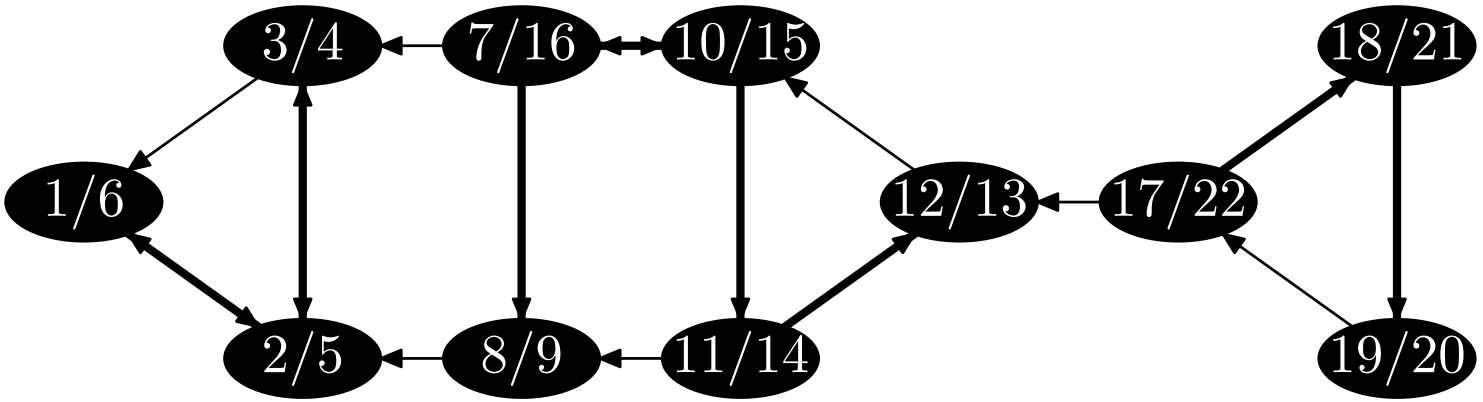


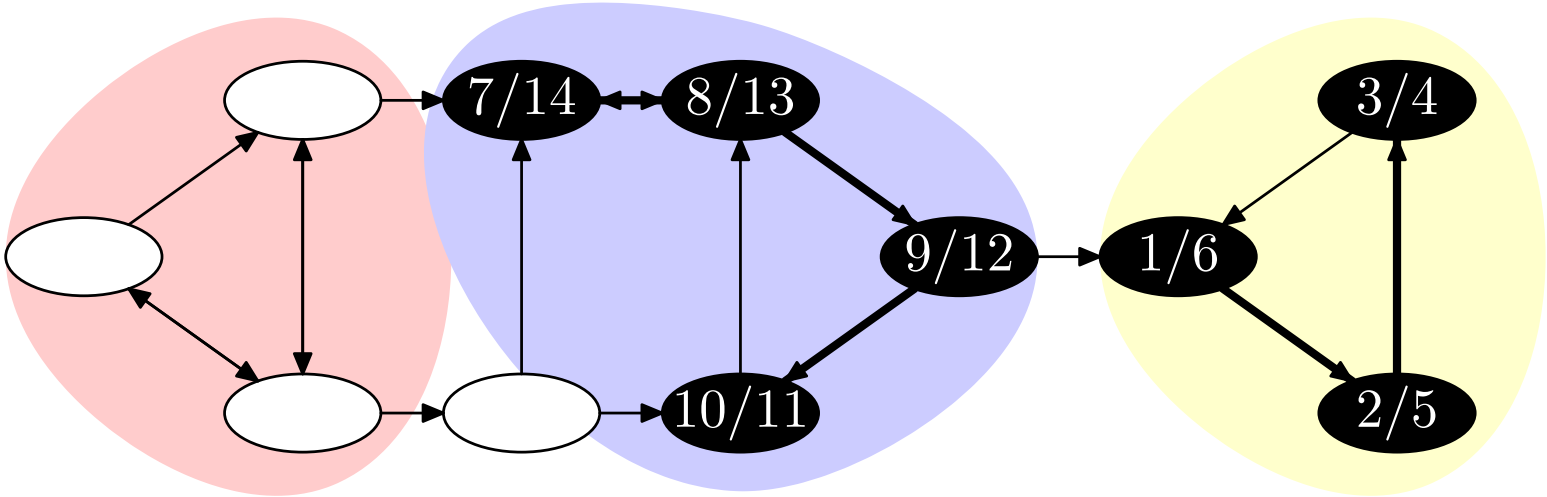
Starke Komponenten – Algorithmus von Kosaraju



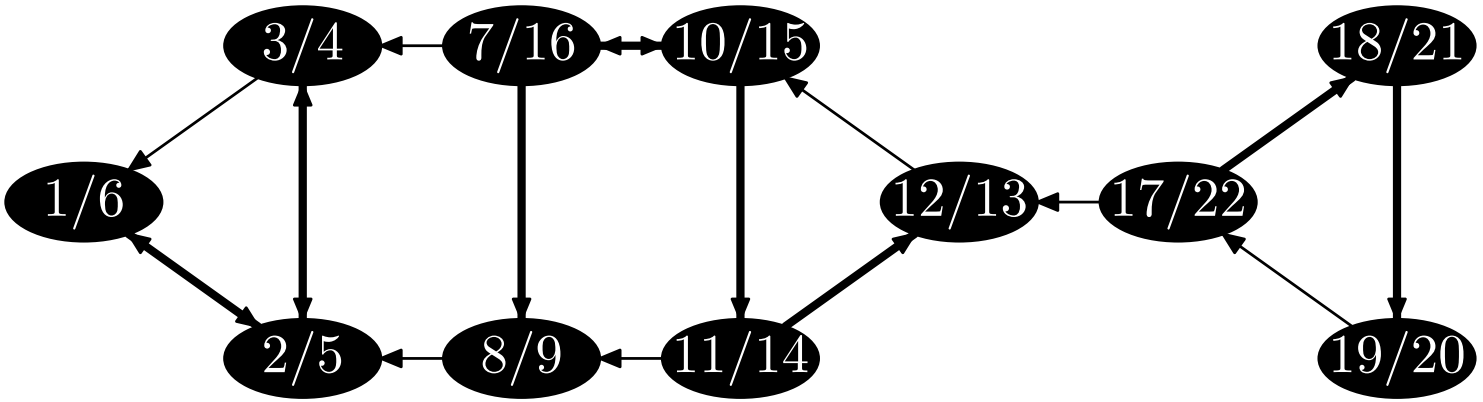


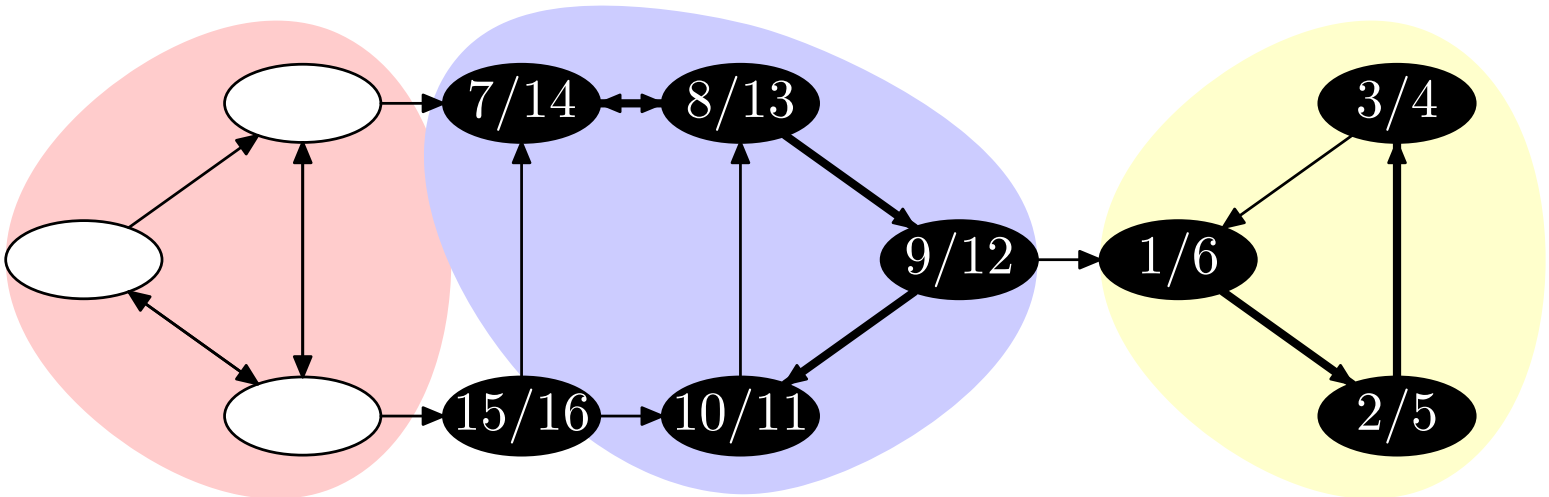
Starke Komponenten – Algorithmus von Kosaraju



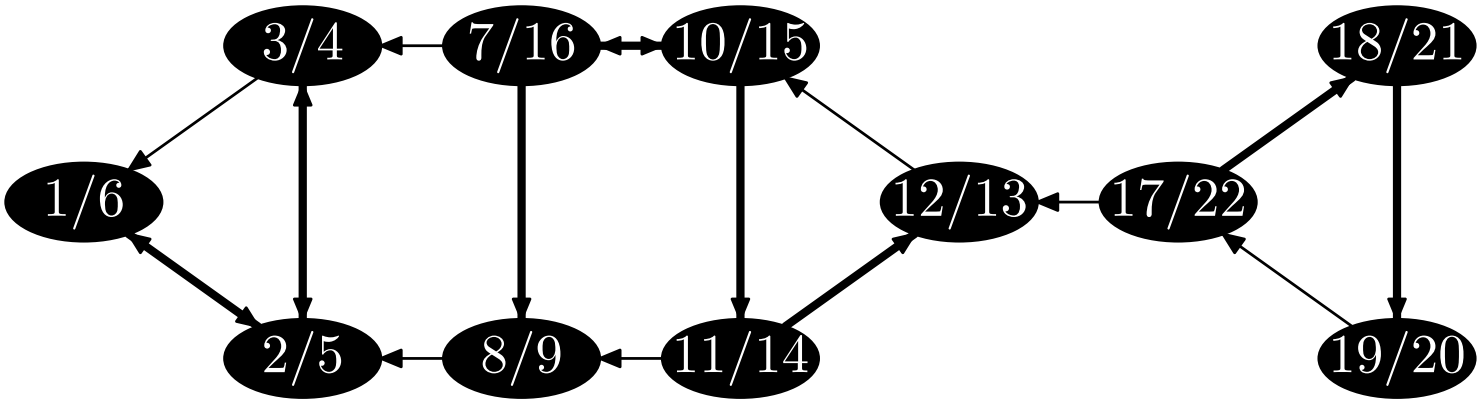


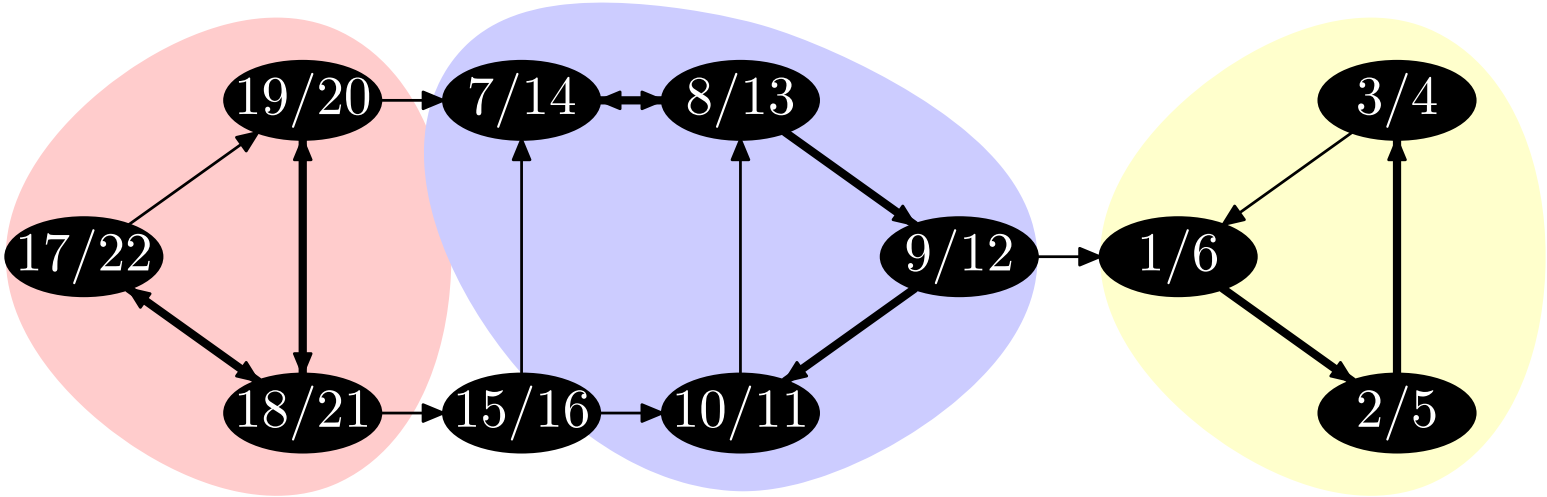
Starke Komponenten – Algorithmus von Kosaraju



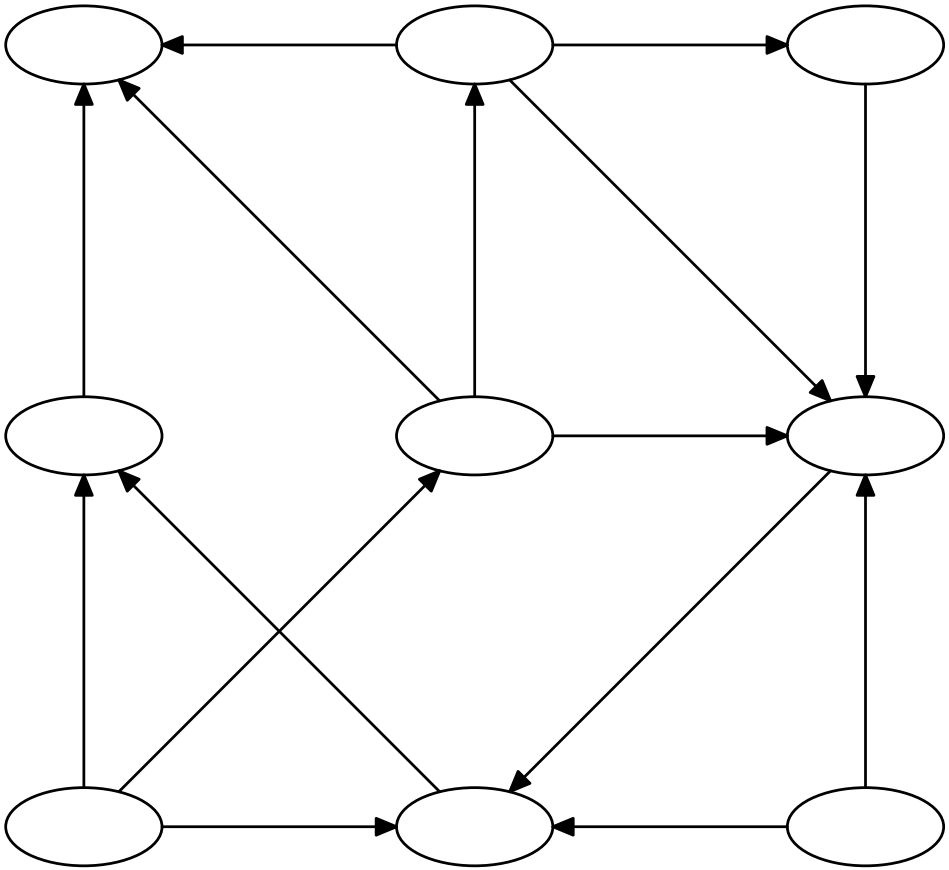


Starke Komponenten – Algorithmus von Kosaraju



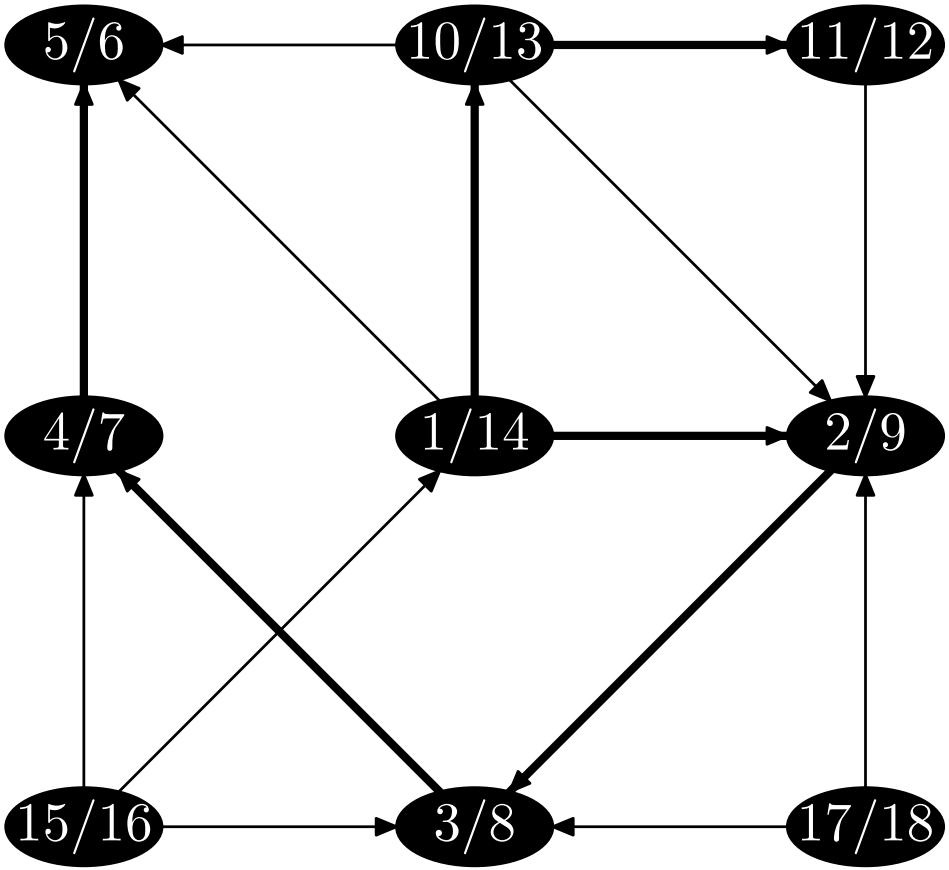


Topologisches Sortieren – Beispiel

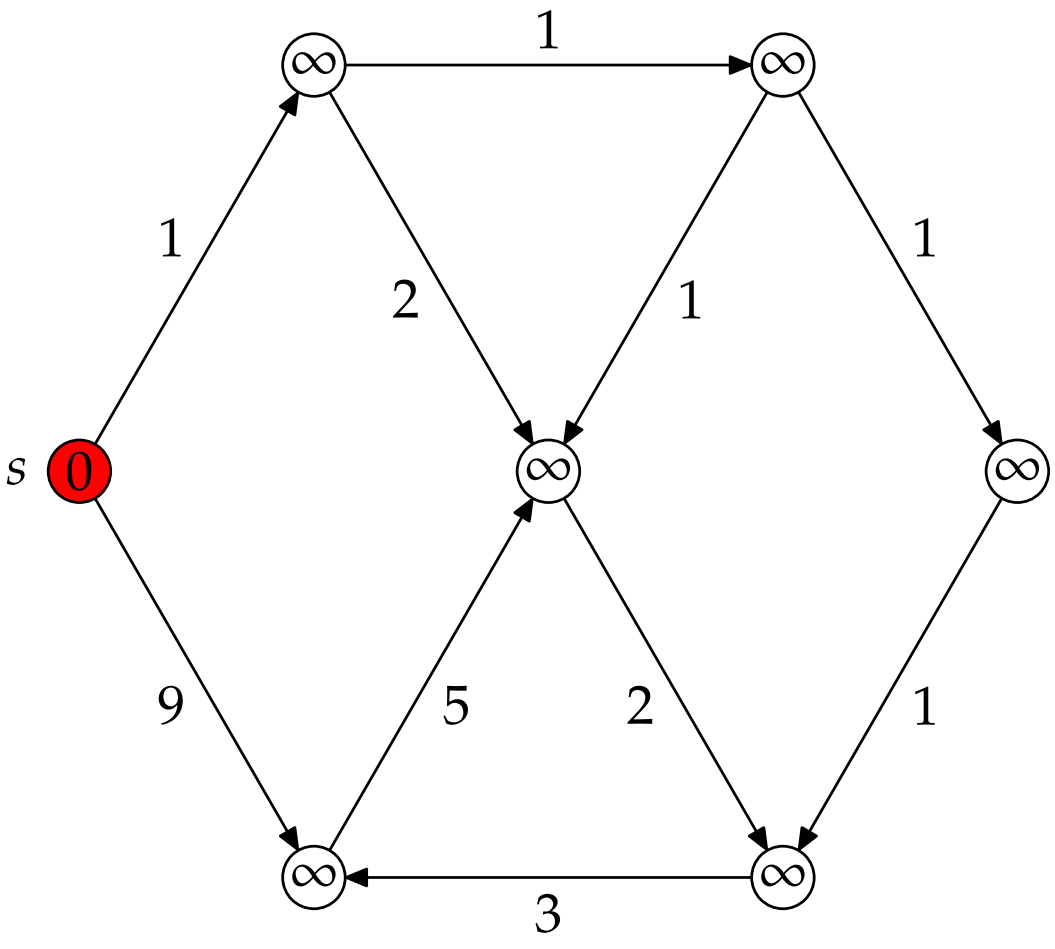


Topologisches Sortieren in $O(|V| + |E|)$ Schritten.

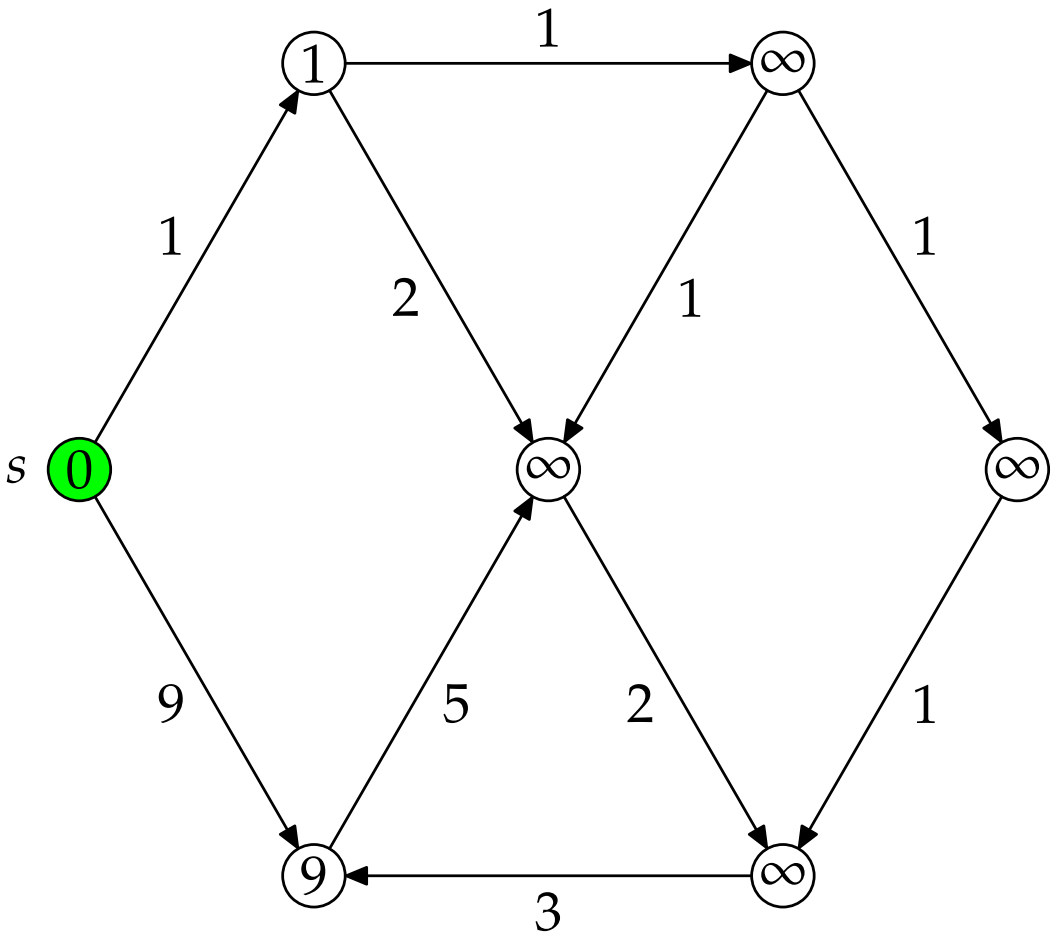
Topologisches Sortieren – Beispiel



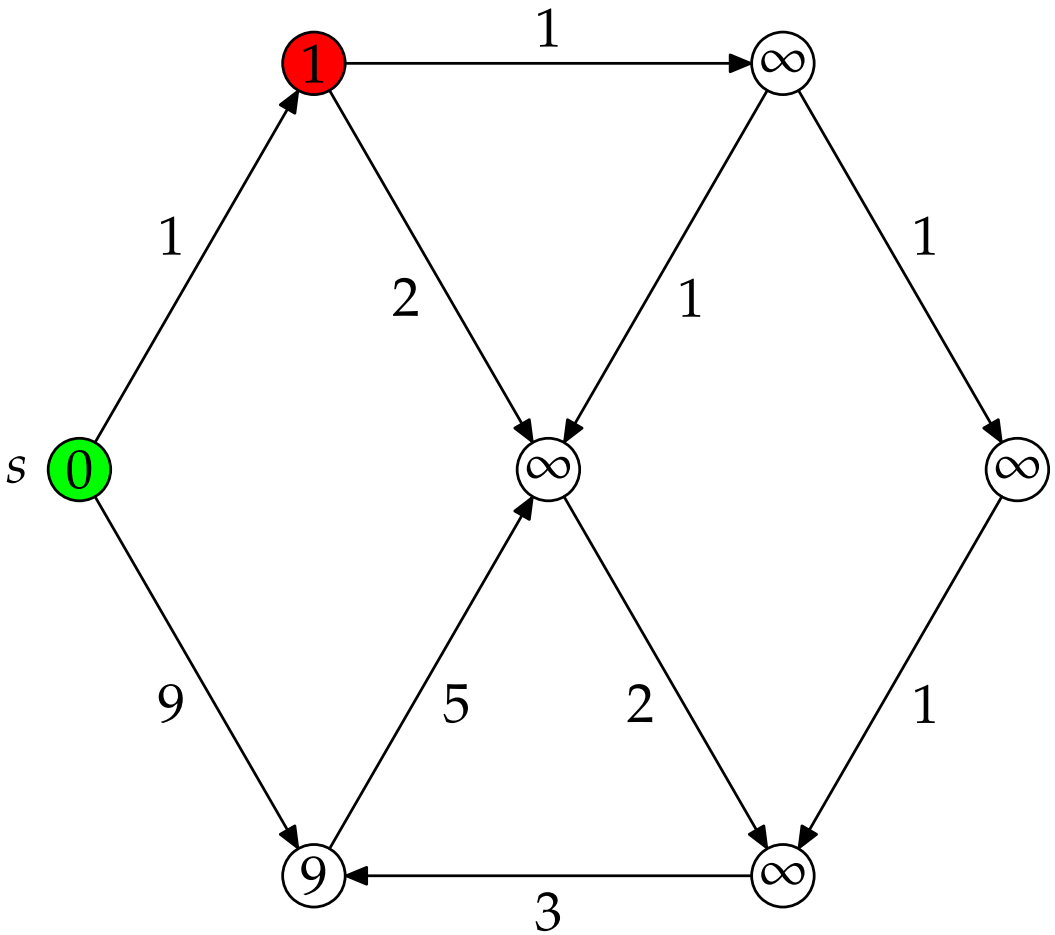
Topologisches Sortieren in $O(|V| + |E|)$ Schritten.



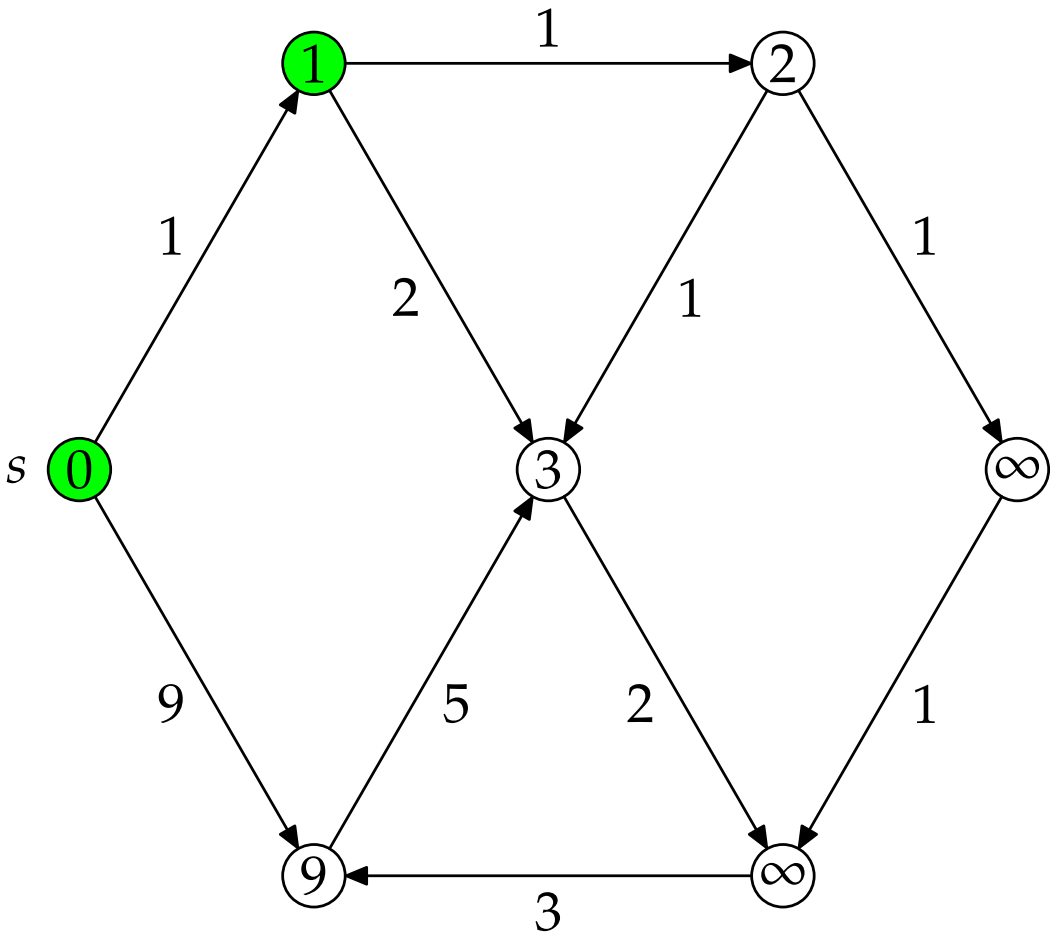
- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.



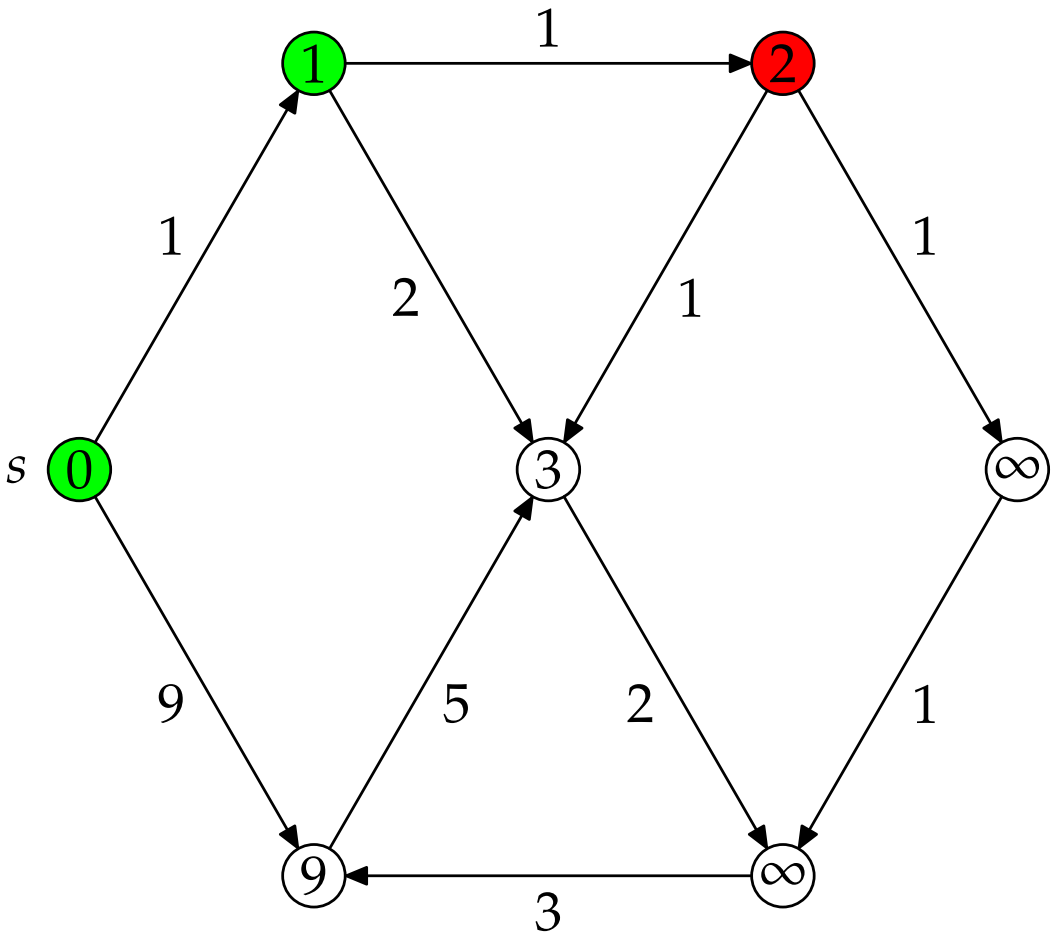
- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.



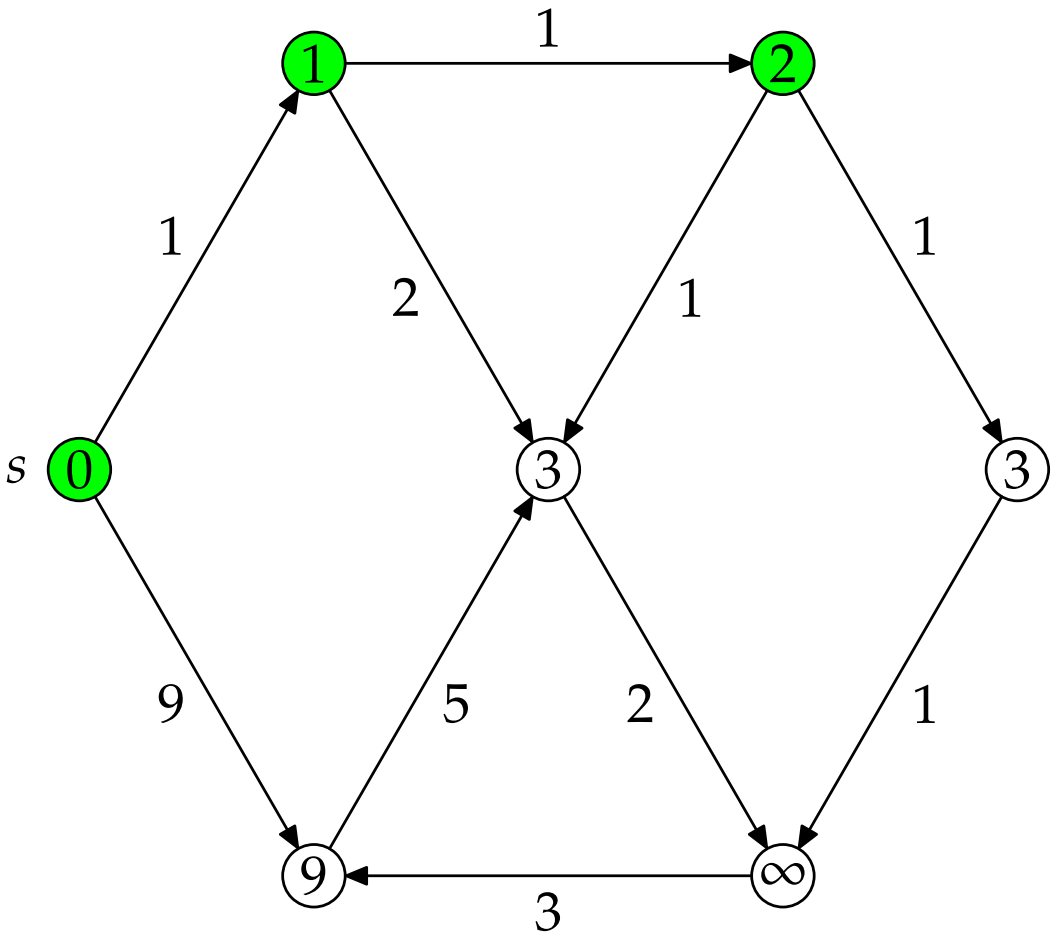
- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.



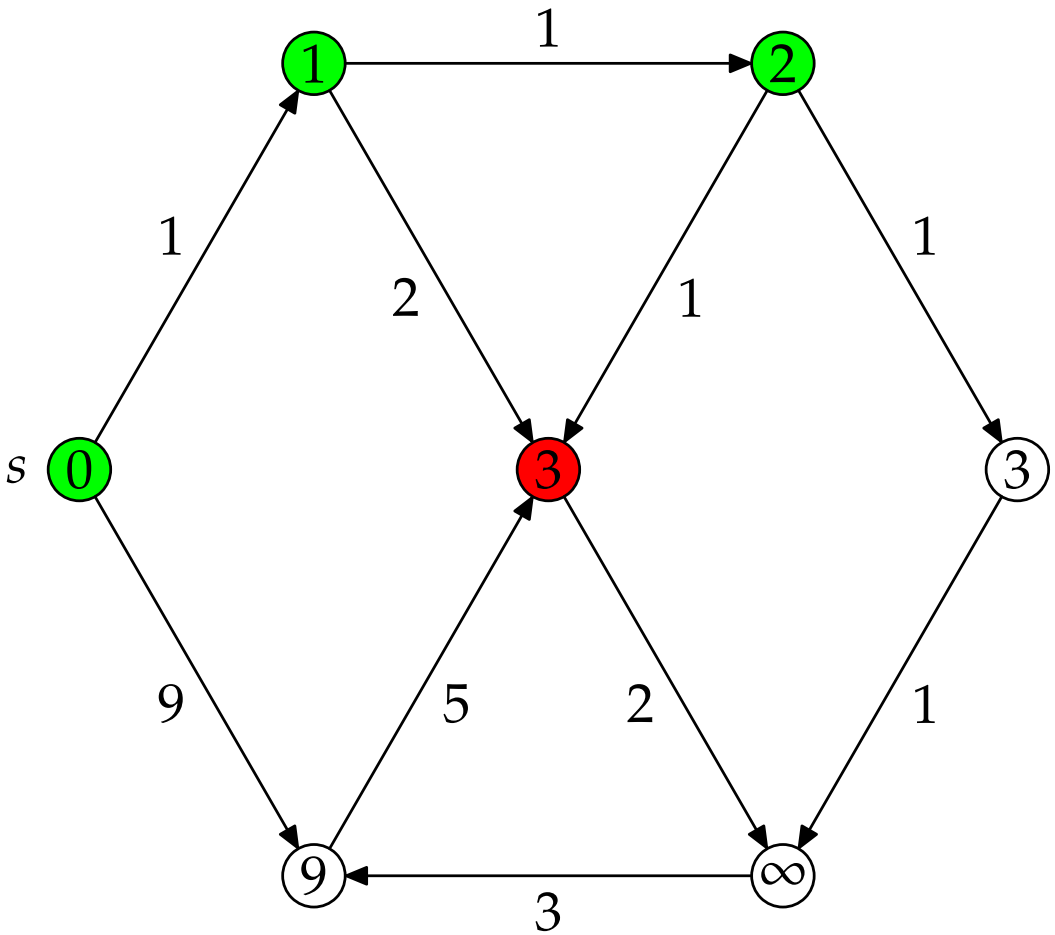
- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.



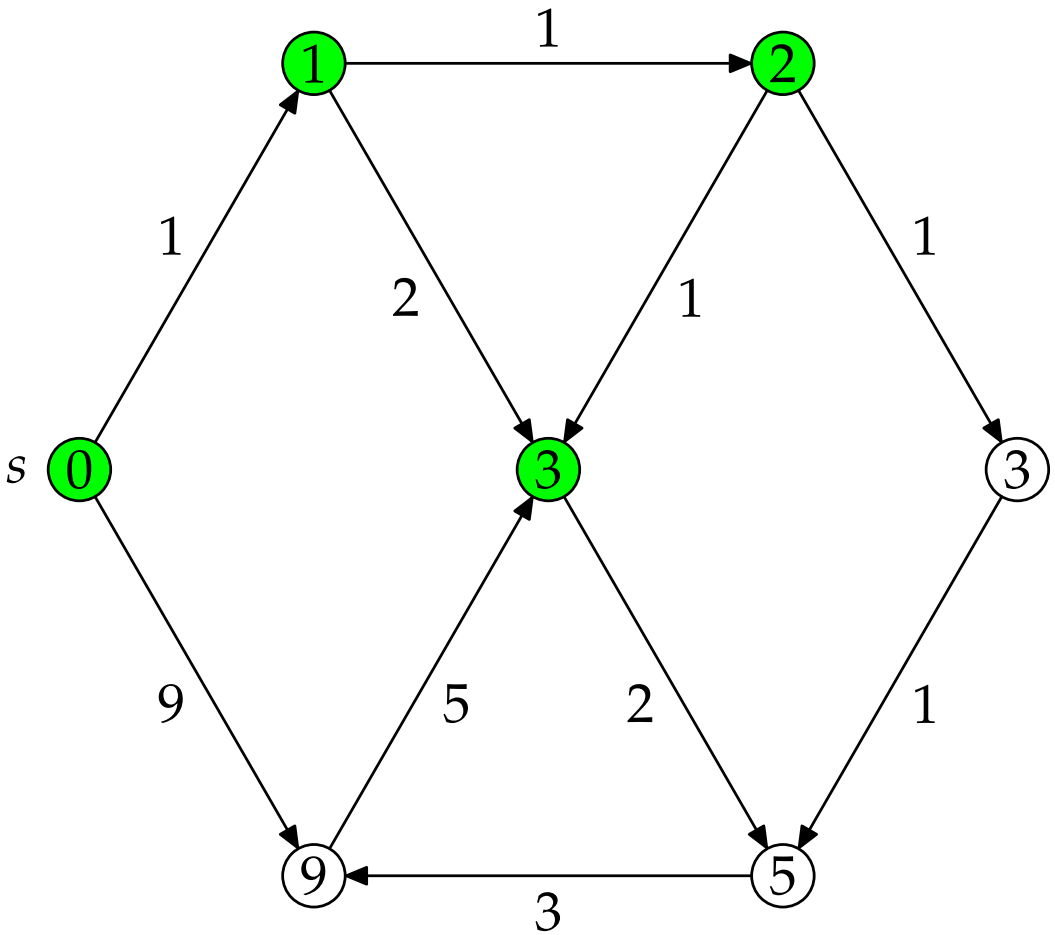
- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.



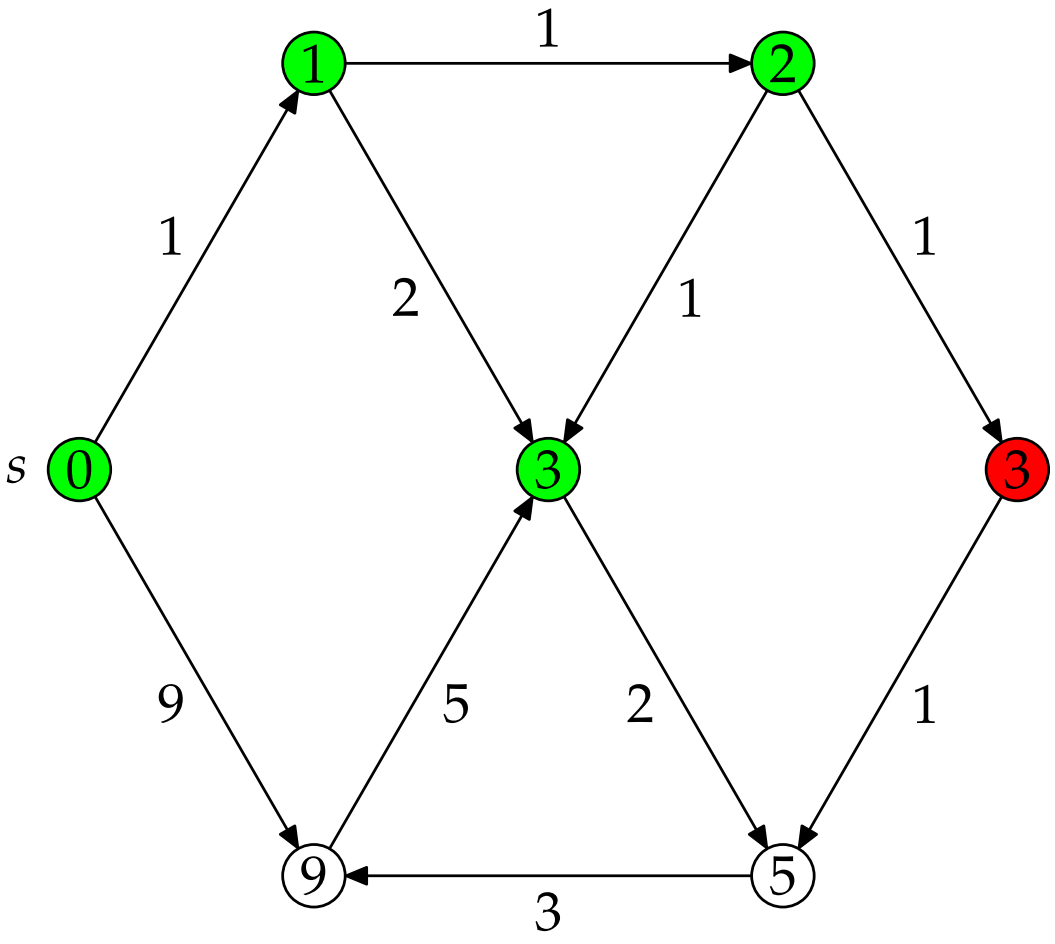
- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.



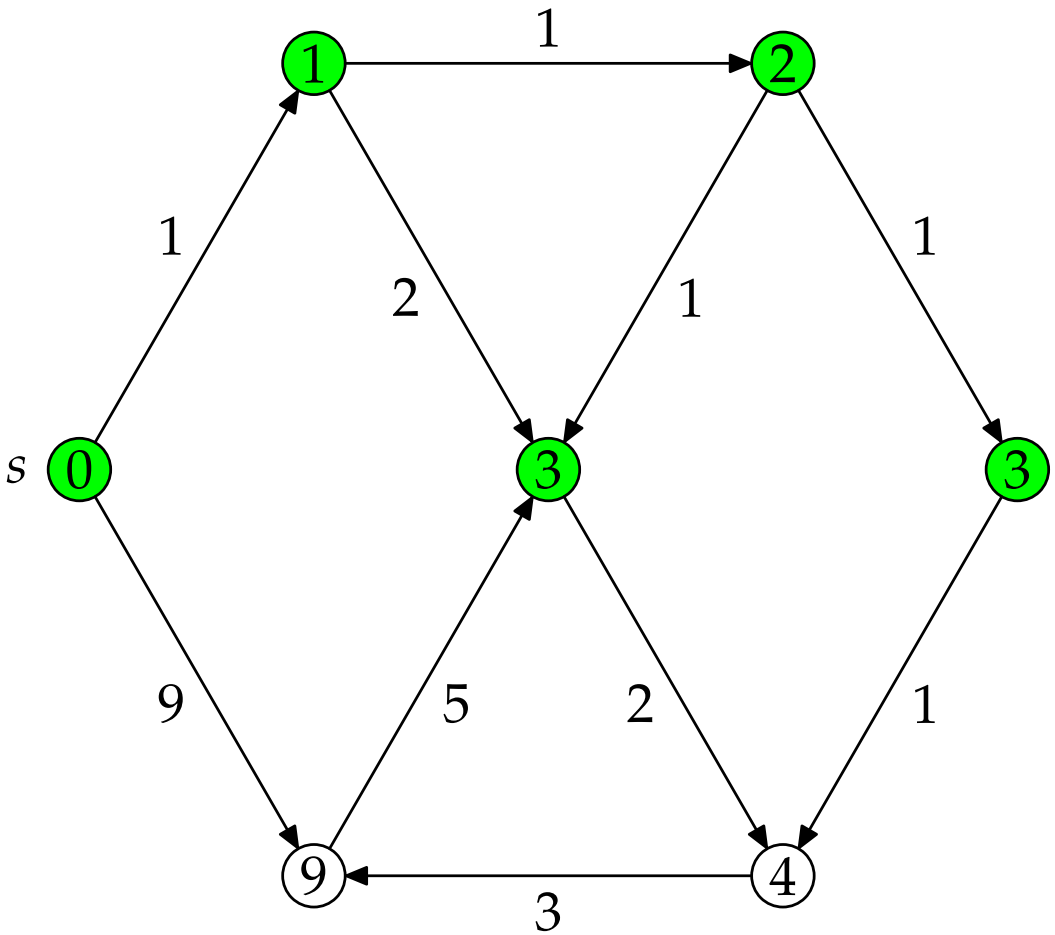
- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.



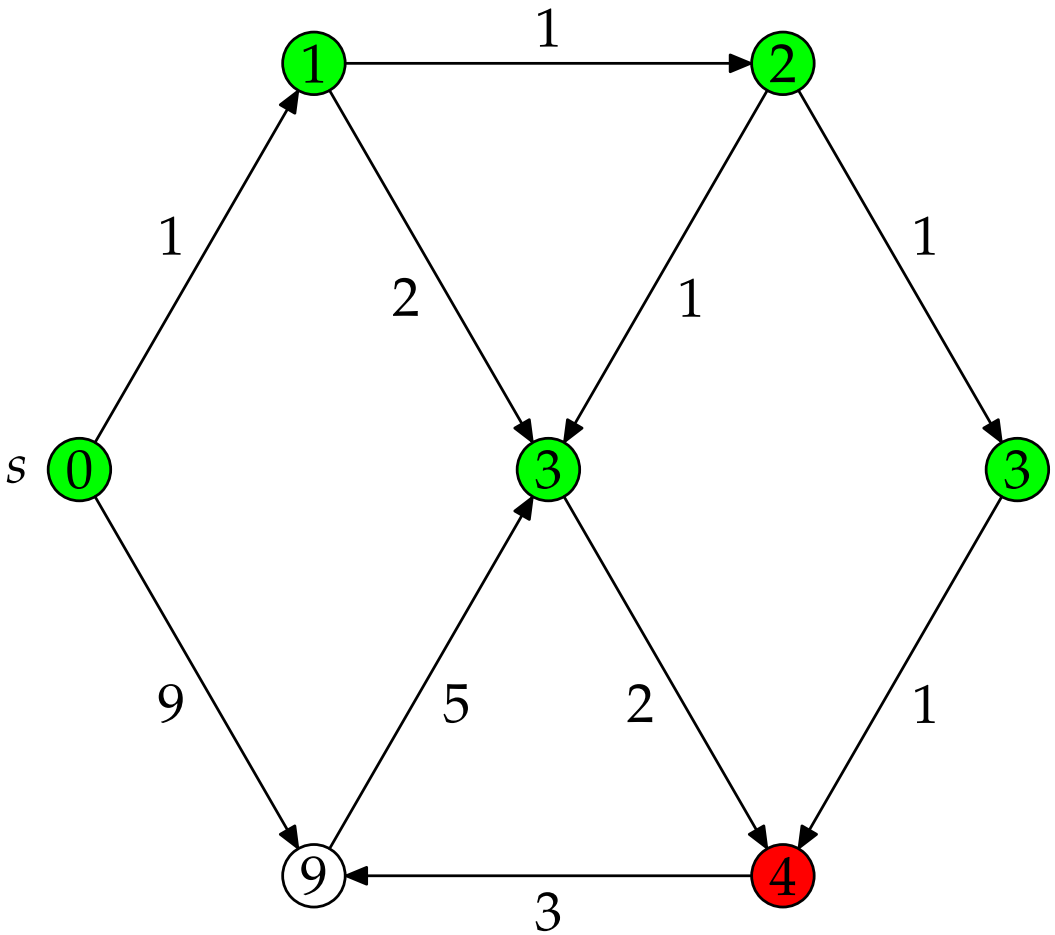
- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.



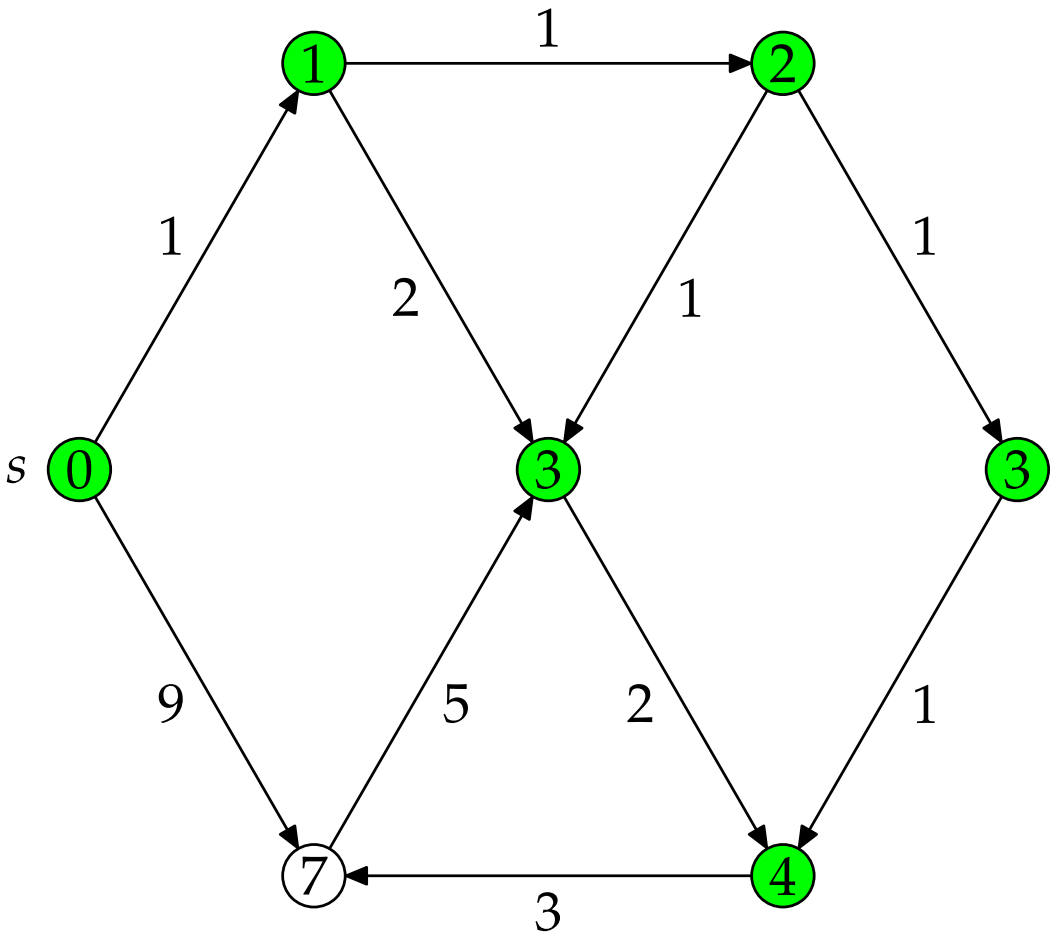
- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.



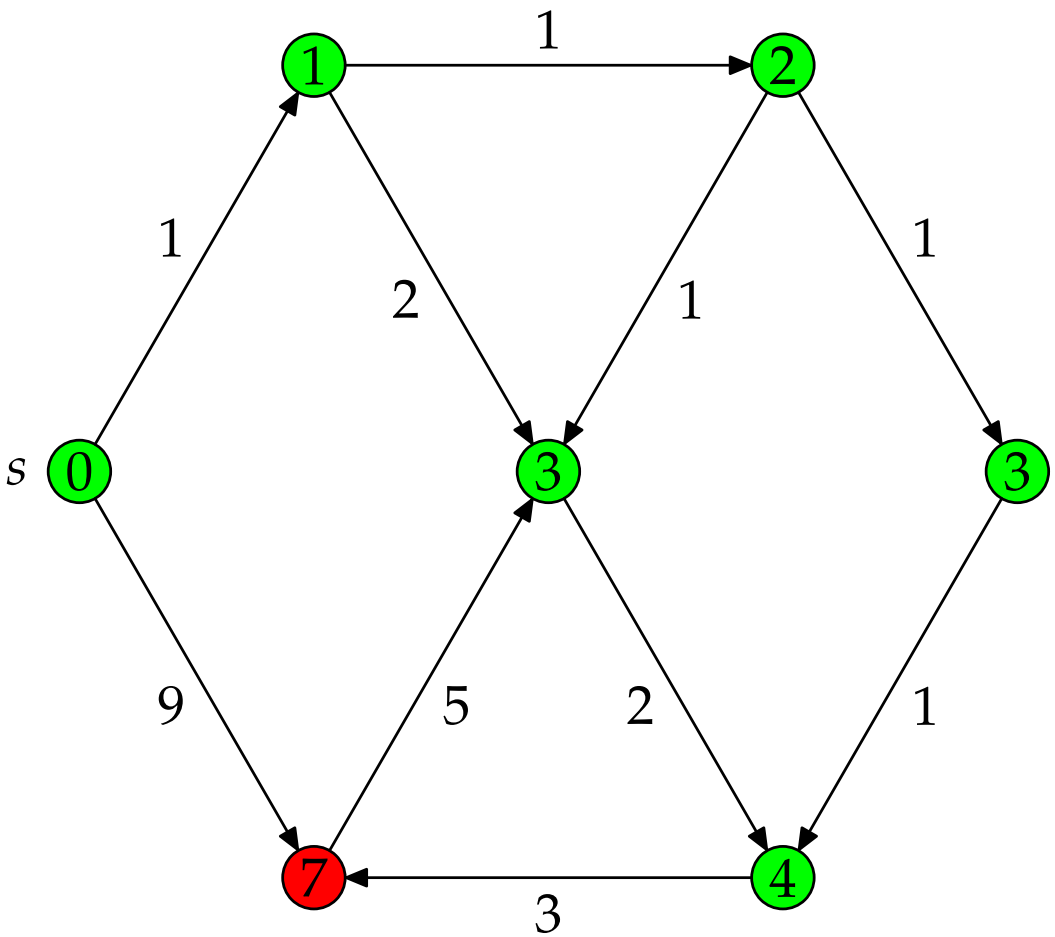
- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.



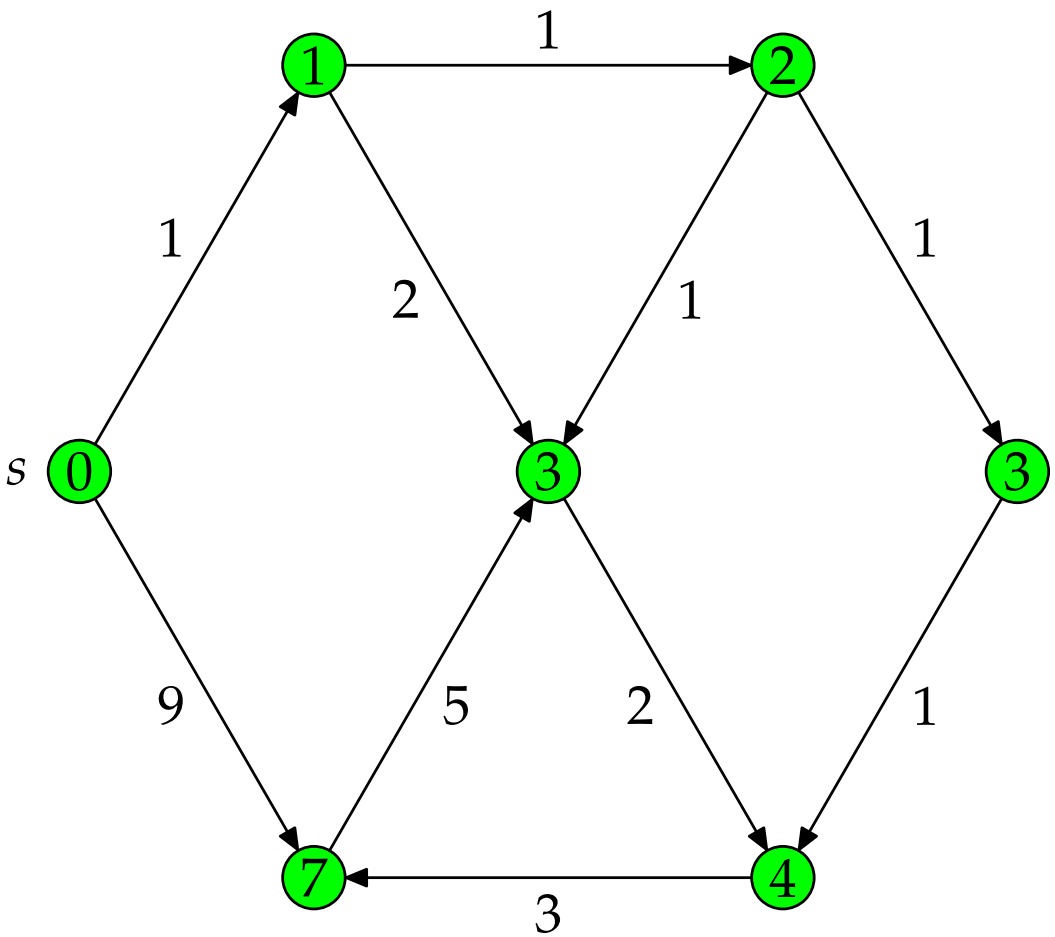
- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.



- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.



- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.



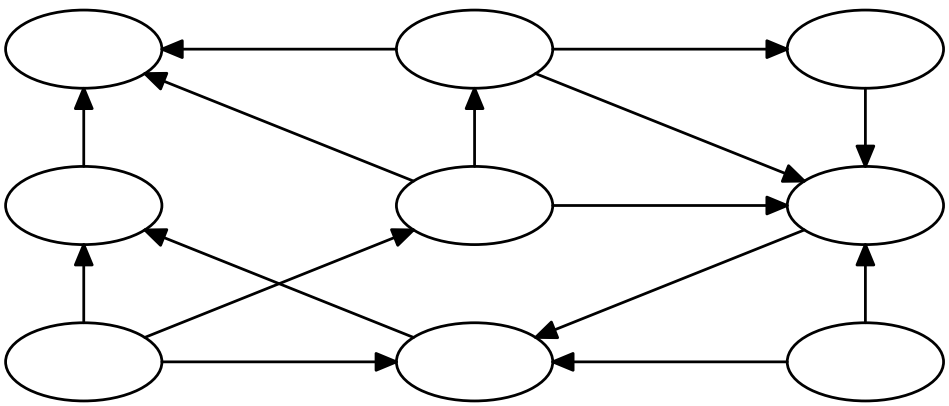
- Grüne Knoten: F
- Roter Knoten aktiv, relaxiert seine Nachbarn
- Weiße Knoten enthalten Abstand zu s über grüne Knoten.

Spezialfall DAG

Können wir kürzeste Pfade in DAGs schneller finden?

Theorem

Kürzeste Pfade von einem Knoten s in einem DAG können in linearer Zeit gefunden werden.

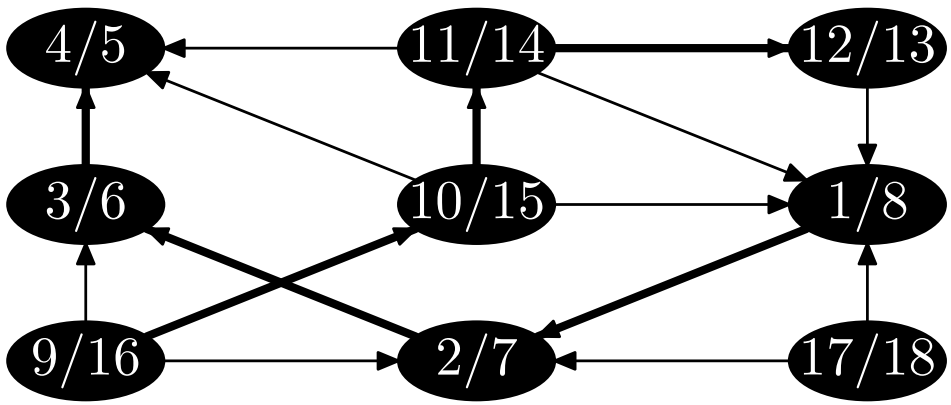


Spezialfall DAG

Können wir kürzeste Pfade in DAGs schneller finden?

Theorem

Kürzeste Pfade von einem Knoten s in einem DAG können in linearer Zeit gefunden werden.



Spezialfall DAG

Können wir kürzeste Pfade in DAGs schneller finden?

Theorem

Kürzeste Pfade von einem Knoten s in einem DAG können in linearer Zeit gefunden werden.

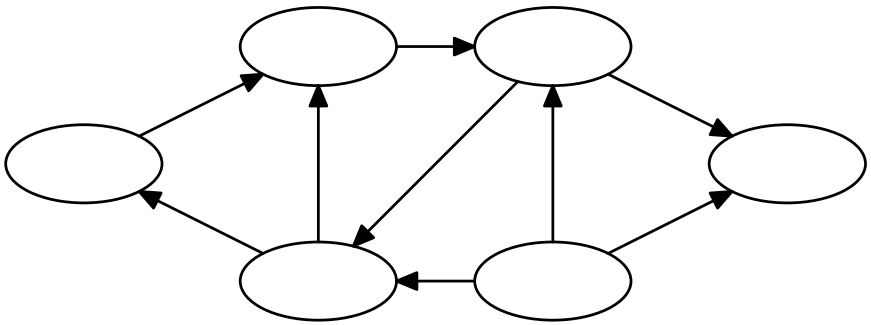
Beweis.

Relaxiere Knoten in topologischer Reihenfolge.

Laufzeit: $O(|V| + |E|)$.
Korrektheit: Jeder Knoten, der relaxiert, kennt zu diesem Zeitpunkt seinen echten Abstand. □

Frage: Sind negative Gewichte hier erlaubt?

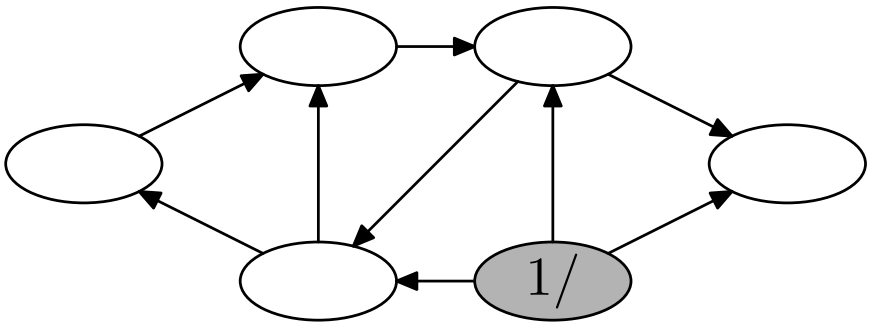
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

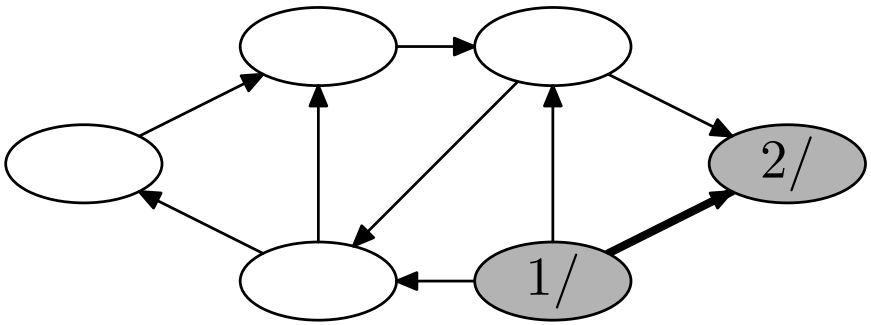
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

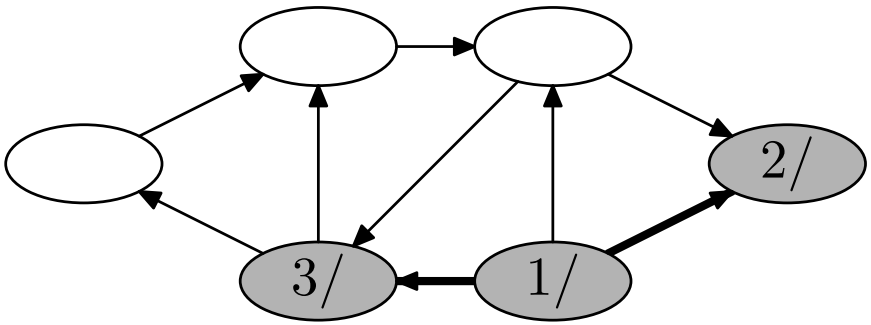
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

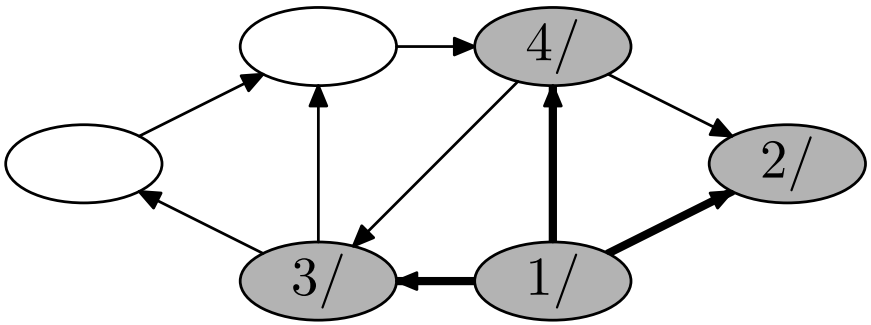
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

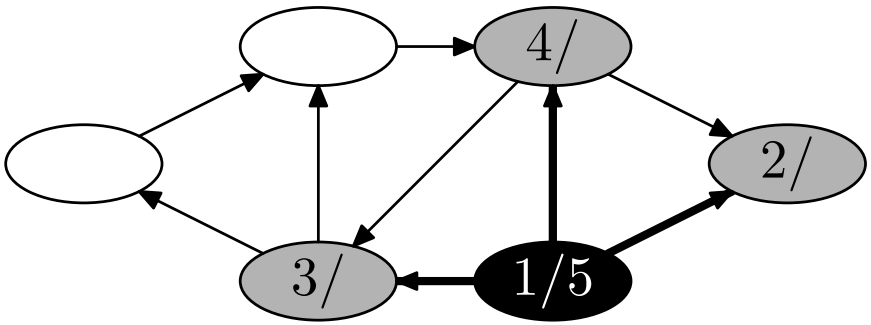
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

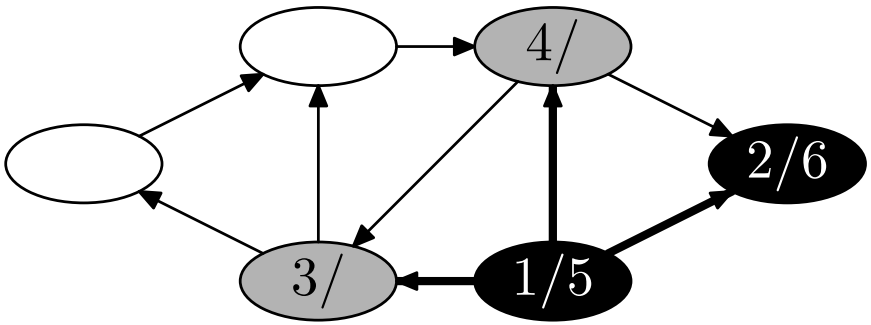
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

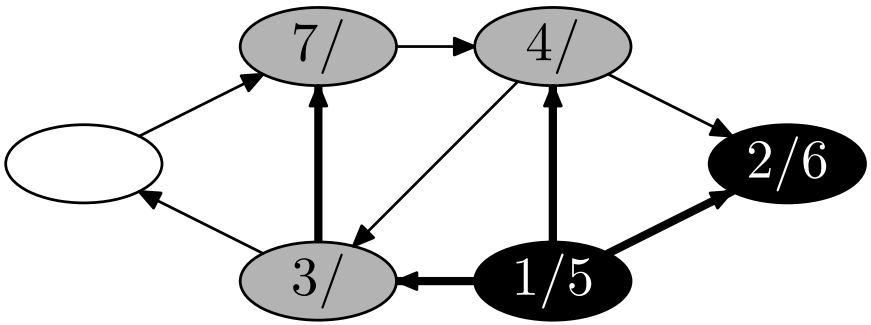
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

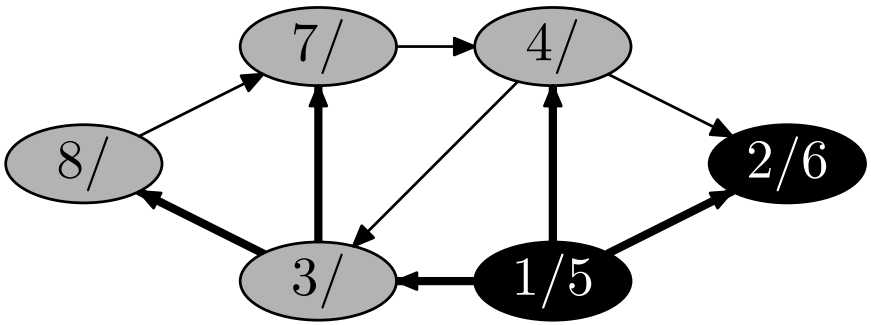
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

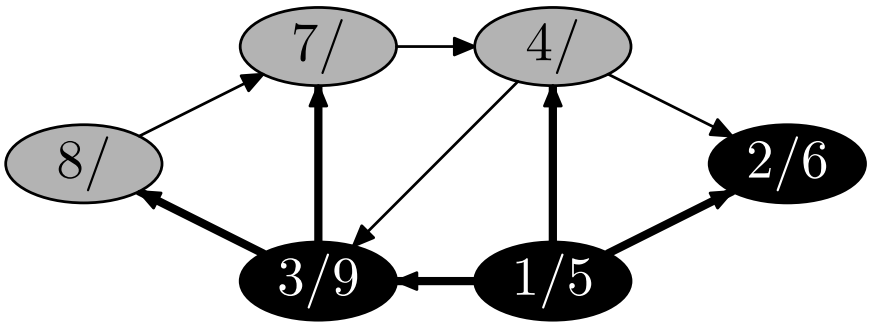
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

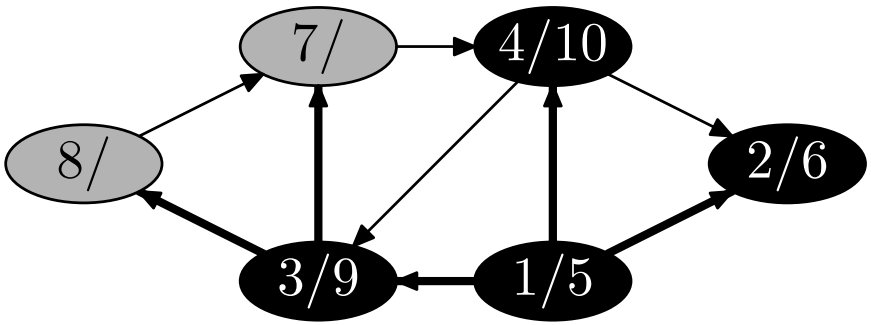
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

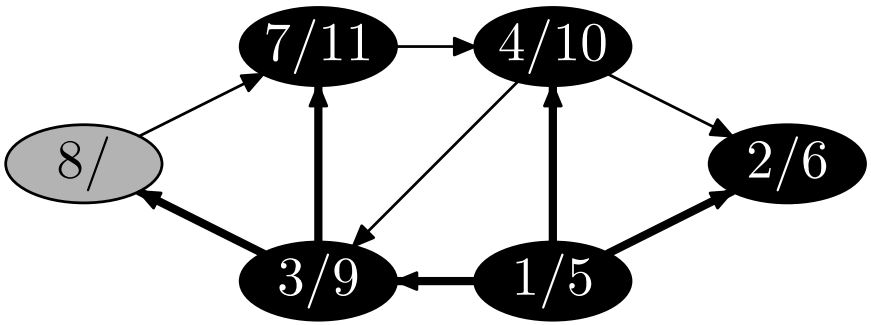
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

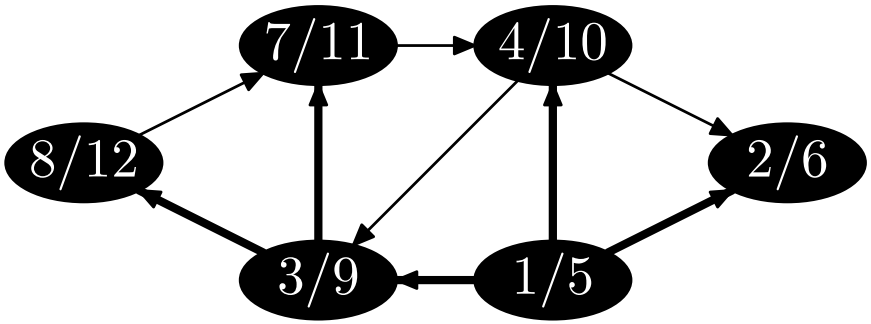
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

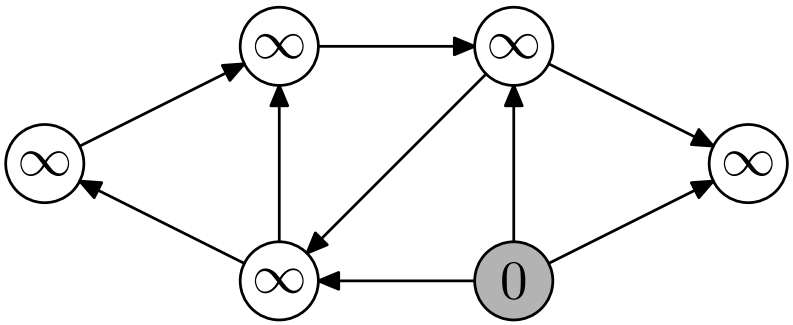
Breitensuche



Breitensuche ist das „Gegenteil“ von Tiefensuche.

- Tiefensuche: Aktive Knoten auf Stack
- Breitensuche: Aktive Knoten in FIFO-Queue
- Intelligente Suche: Weder Stack noch Queue
- Breitensuchbaum enthält kürzeste Pfade (ungewichtet)
- Discovery- und Finishzeiten wenig Anwendungen

Breitensuche



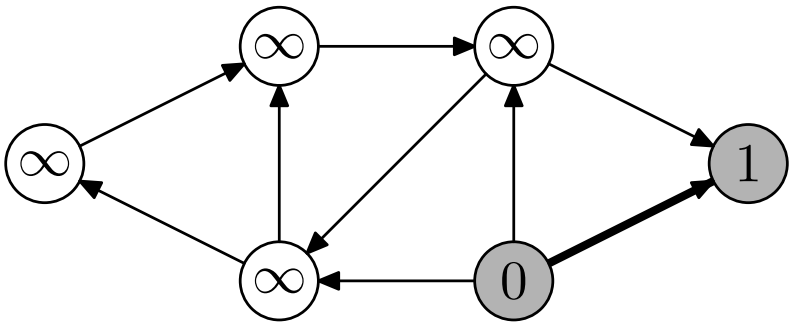
Knoten wird aktiv:

Abstand berechnen und Kante in BFS-Baum

Anwendung:

Alle Abstände und kürzesten Wege von s berechnen

Breitensuche



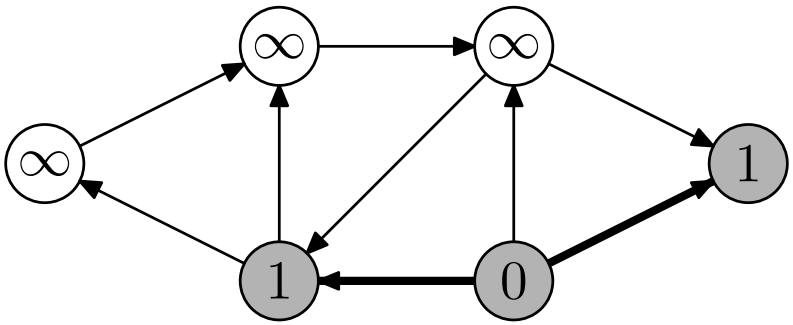
Knoten wird aktiv:

Abstand berechnen und Kante in BFS-Baum

Anwendung:

Alle Abstände und kürzesten Wege von s berechnen

Breitensuche



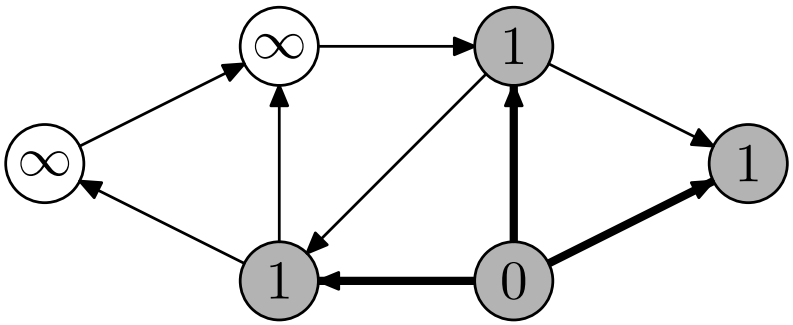
Knoten wird aktiv:

Abstand berechnen und Kante in BFS-Baum

Anwendung:

Alle Abstände und kürzesten Wege von s berechnen

Breitensuche



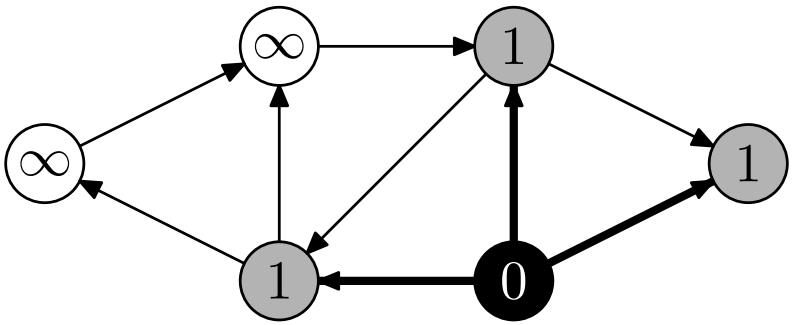
Knoten wird aktiv:

Abstand berechnen und Kante in BFS-Baum

Anwendung:

Alle Abstände und kürzesten Wege von s berechnen

Breitensuche



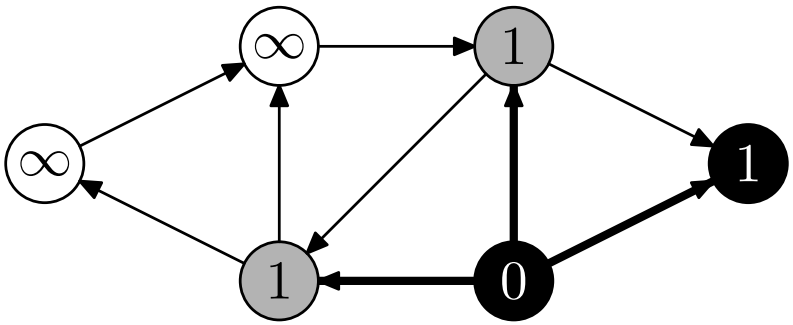
Knoten wird aktiv:

Abstand berechnen und Kante in BFS-Baum

Anwendung:

Alle Abstände und kürzesten Wege von s berechnen

Breitensuche



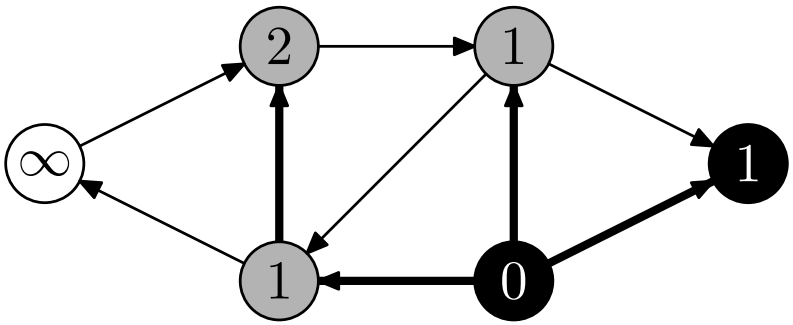
Knoten wird aktiv:

Abstand berechnen und Kante in BFS-Baum

Anwendung:

Alle Abstände und kürzesten Wege von s berechnen

Breitensuche



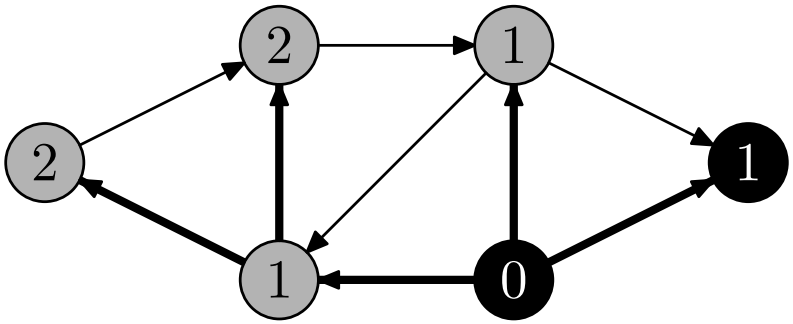
Knoten wird aktiv:

Abstand berechnen und Kante in BFS-Baum

Anwendung:

Alle Abstände und kürzesten Wege von s berechnen

Breitensuche



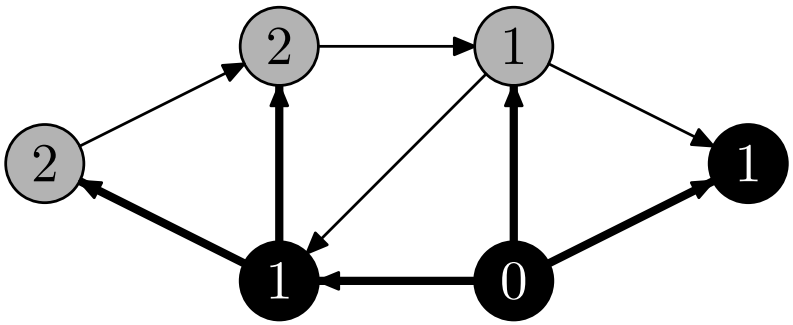
Knoten wird aktiv:

Abstand berechnen und Kante in BFS-Baum

Anwendung:

Alle Abstände und kürzesten Wege von s berechnen

Breitensuche



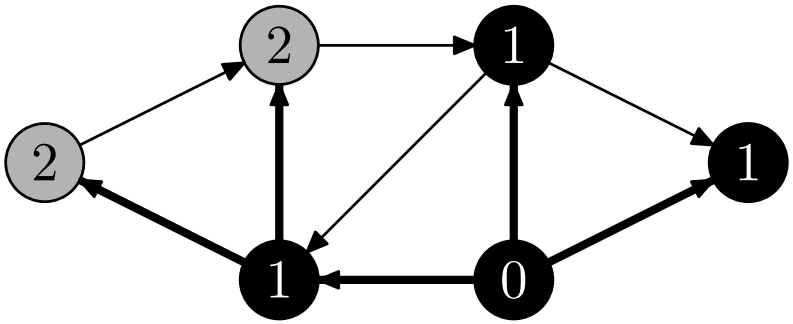
Knoten wird aktiv:

Abstand berechnen und Kante in BFS-Baum

Anwendung:

Alle Abstände und kürzesten Wege von s berechnen

Breitensuche



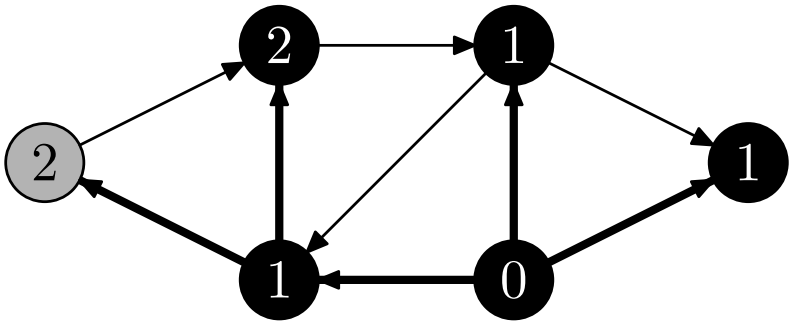
Knoten wird aktiv:

Abstand berechnen und Kante in BFS-Baum

Anwendung:

Alle Abstände und kürzesten Wege von s berechnen

Breitensuche



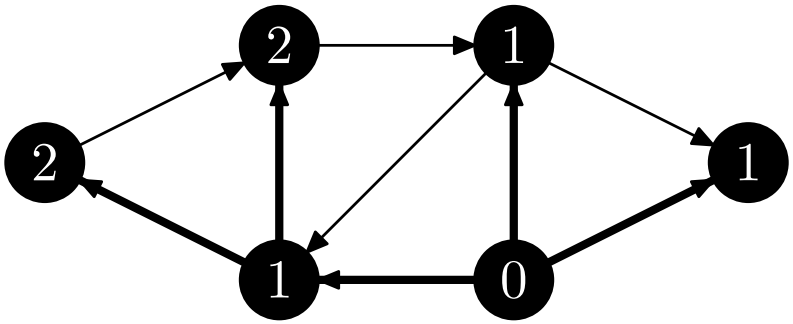
Knoten wird aktiv:

Abstand berechnen und Kante in BFS-Baum

Anwendung:

Alle Abstände und kürzesten Wege von s berechnen

Breitensuche



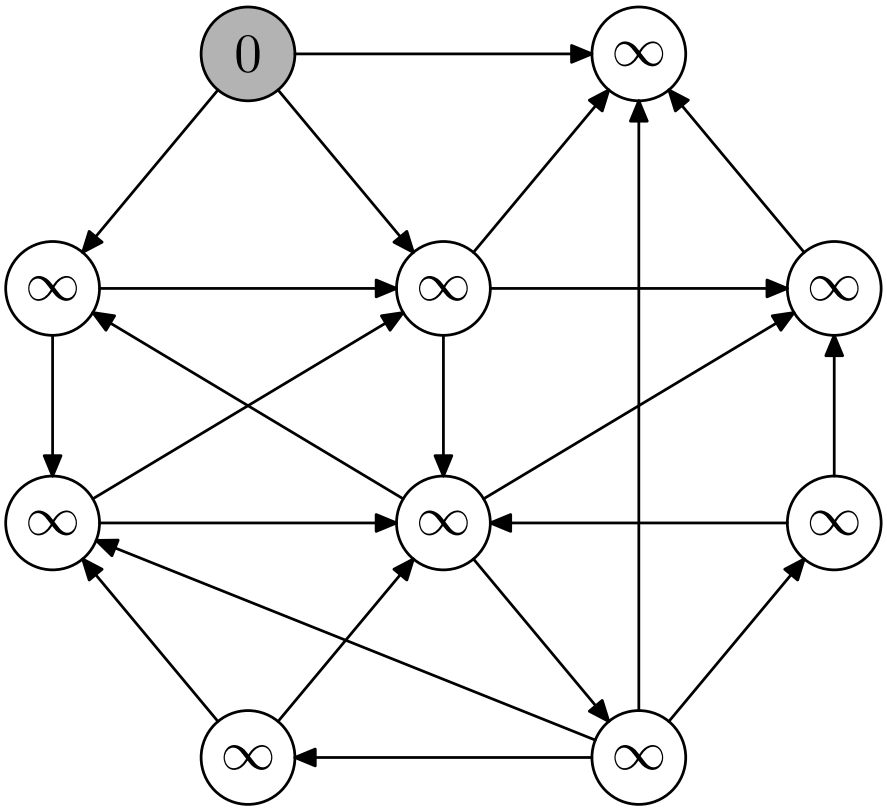
Knoten wird aktiv:

Abstand berechnen und Kante in BFS-Baum

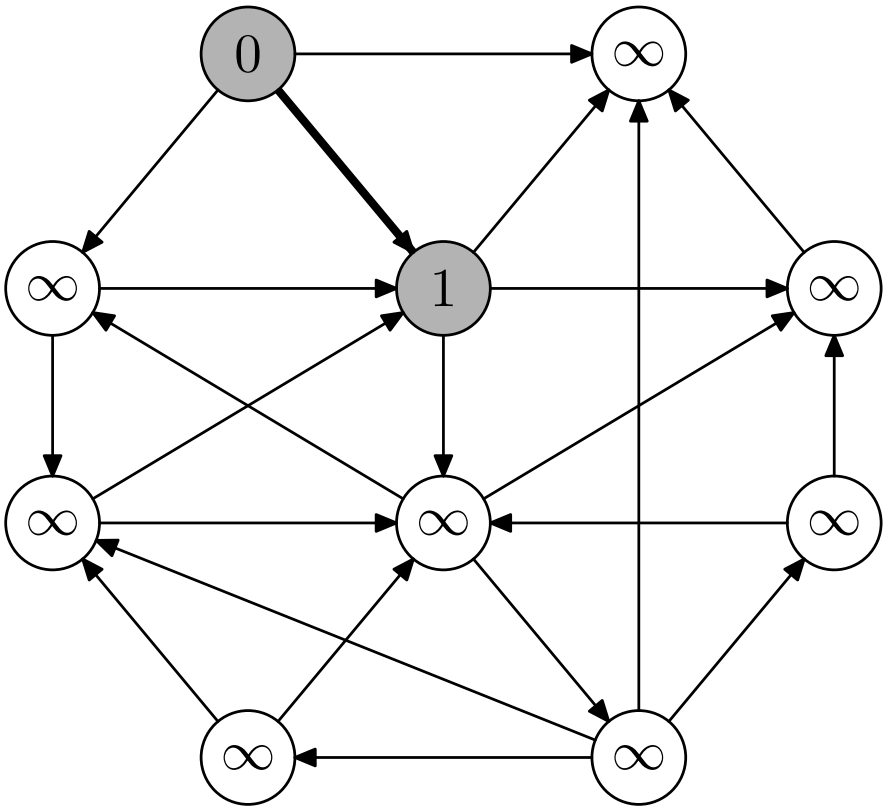
Anwendung:

Alle Abstände und kürzesten Wege von s berechnen

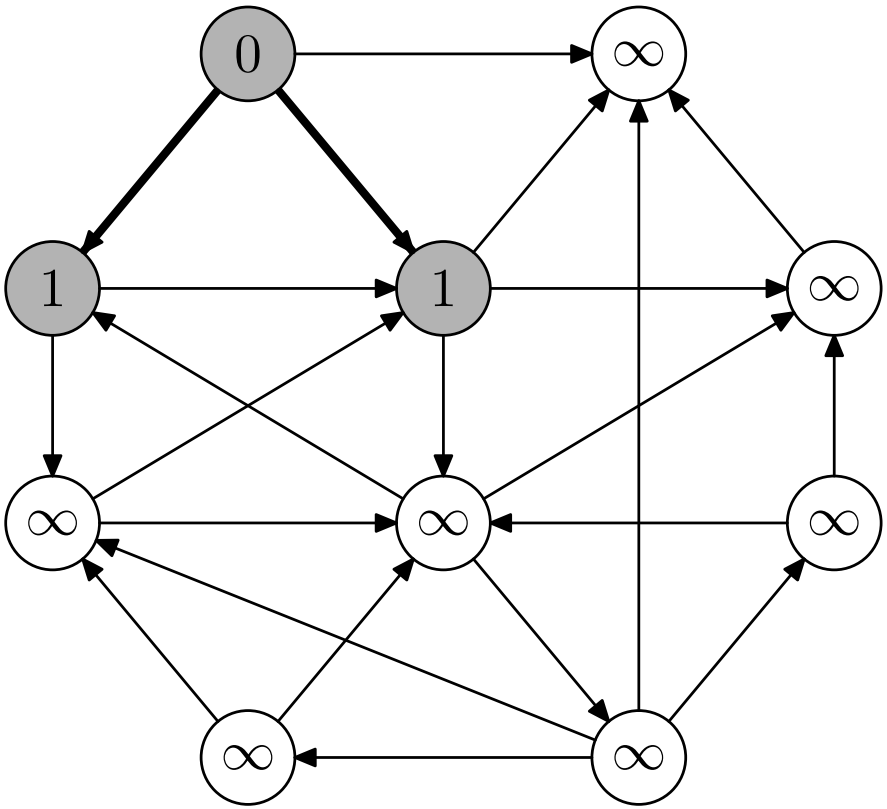
Breitensuche – Größeres Beispiel



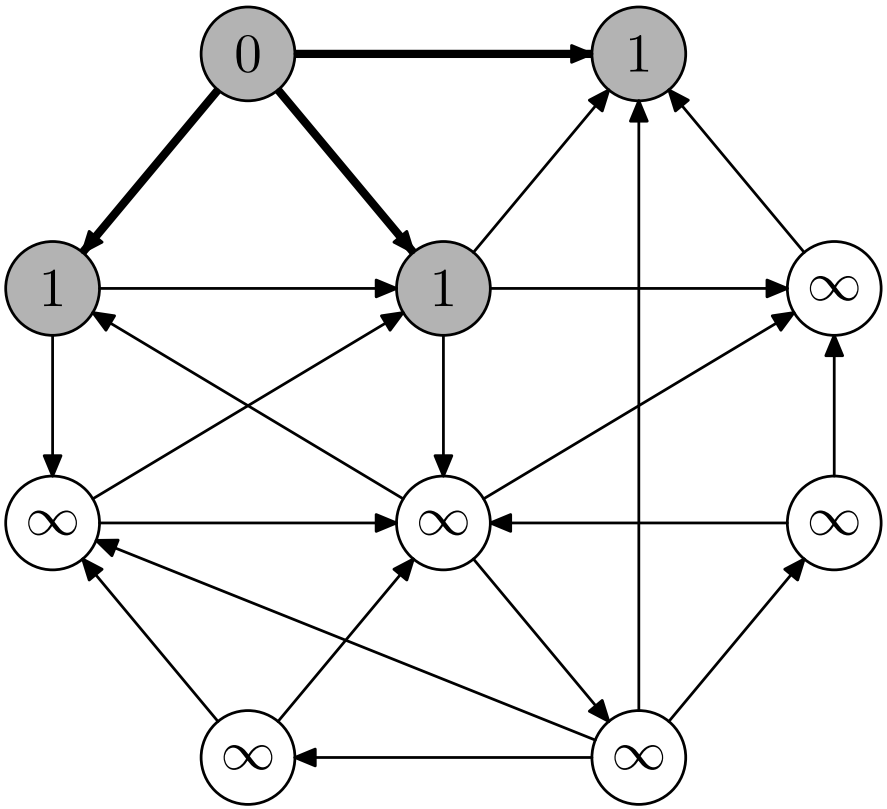
Breitensuche – Größeres Beispiel



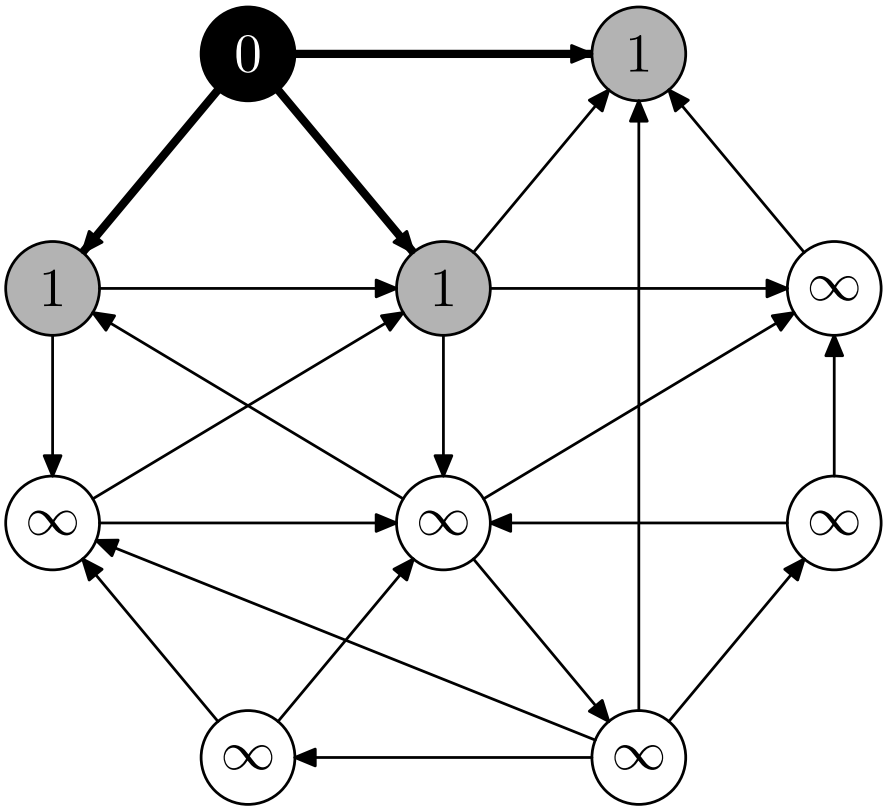
Breitensuche – Größeres Beispiel



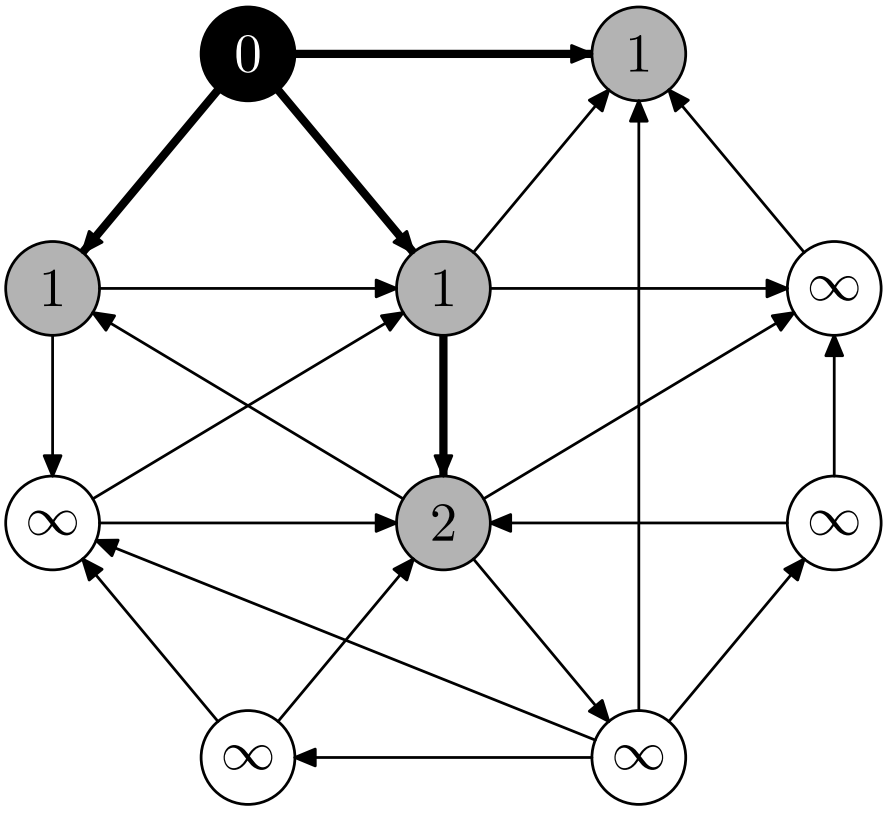
Breitensuche – Größeres Beispiel



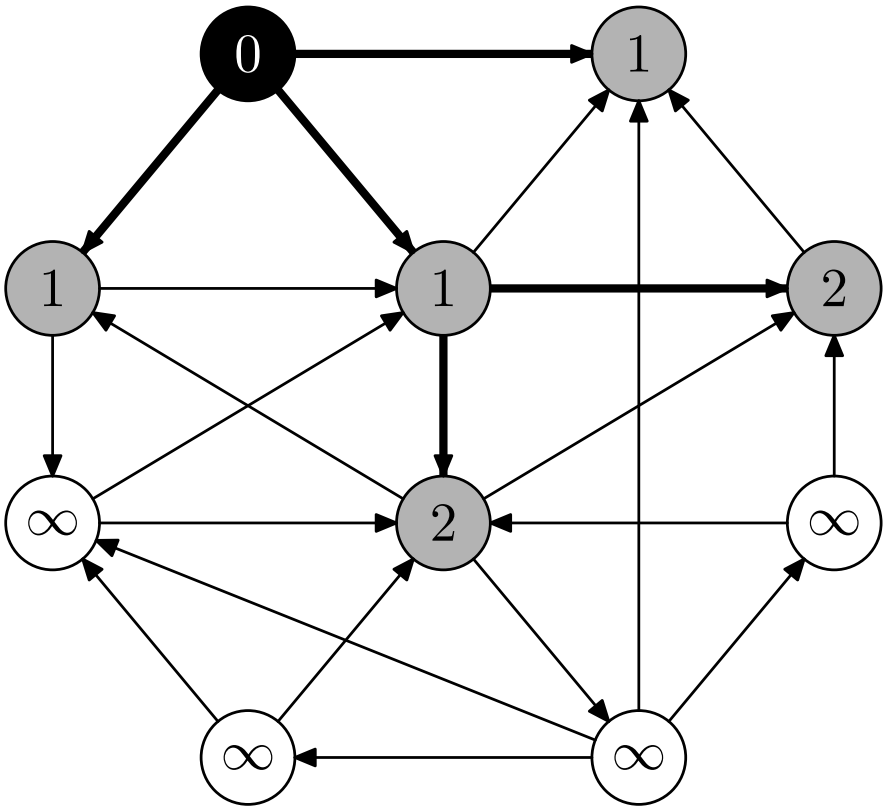
Breitensuche – Größeres Beispiel



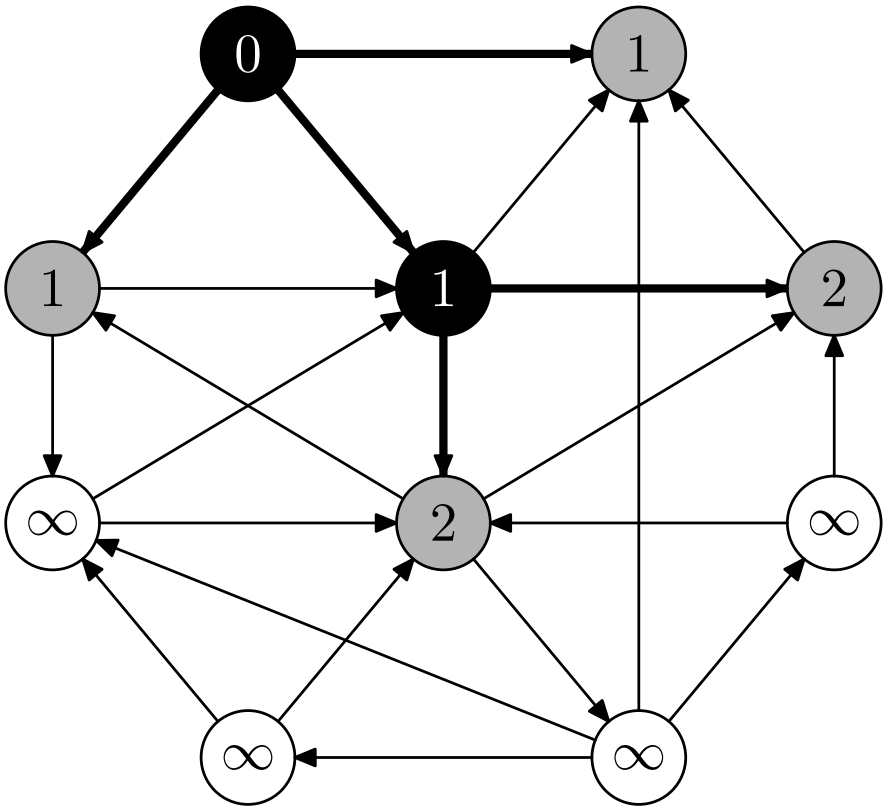
Breitensuche – Größeres Beispiel



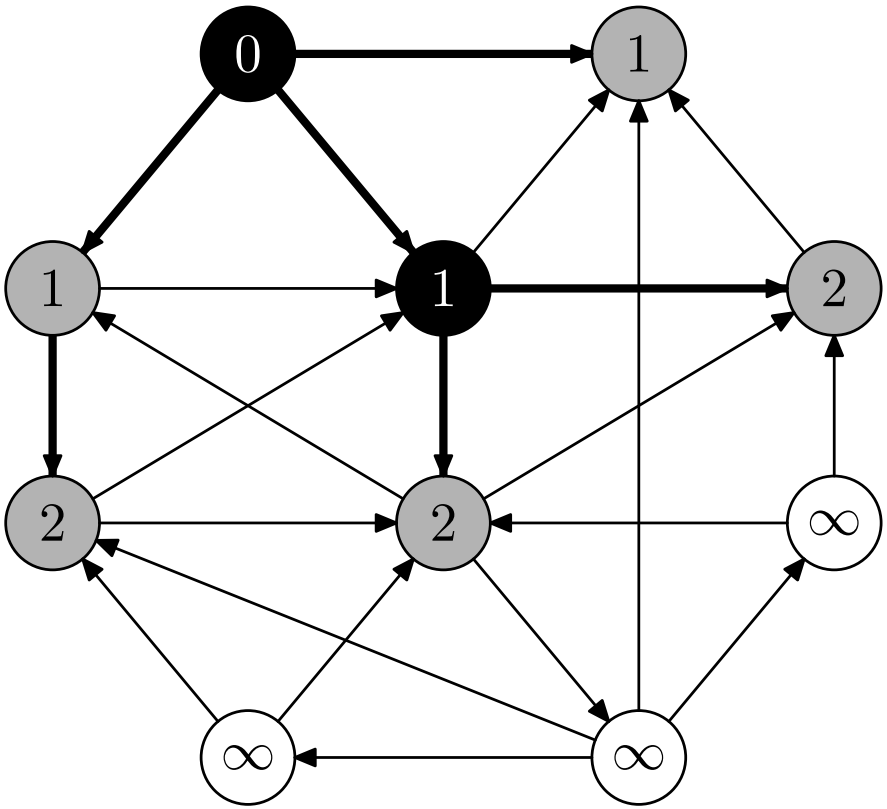
Breitensuche – Größeres Beispiel



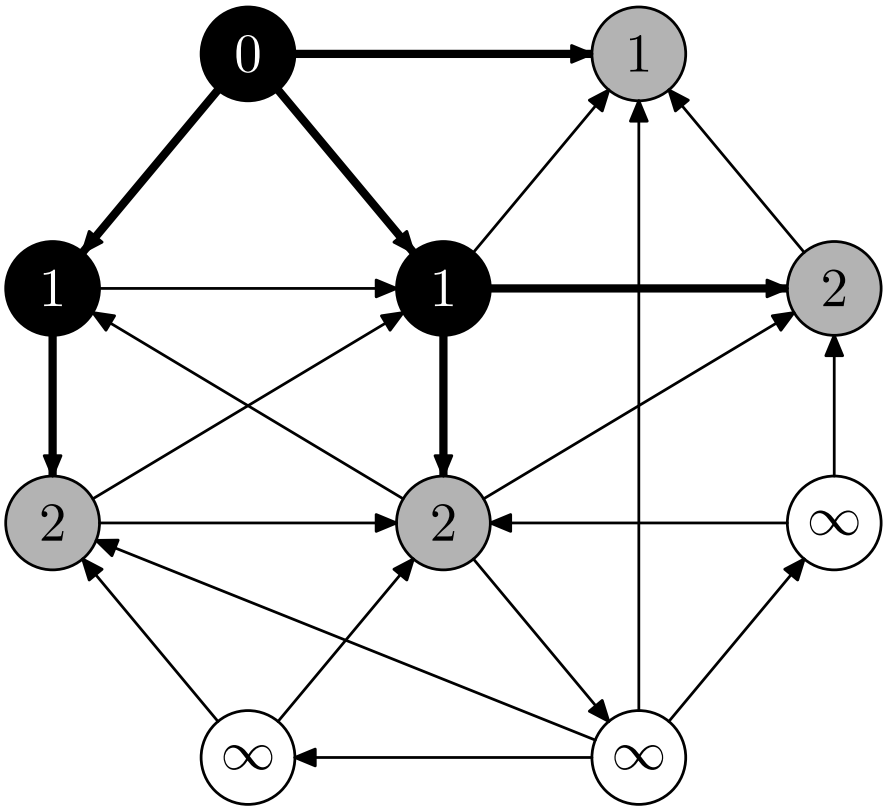
Breitensuche – Größeres Beispiel



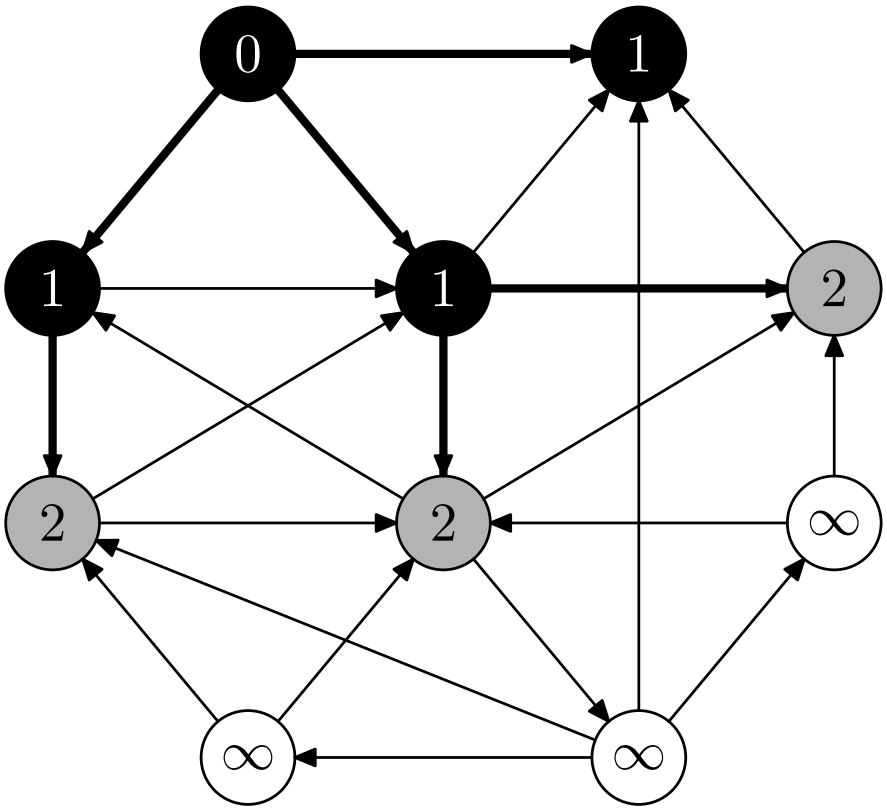
Breitensuche – Größeres Beispiel



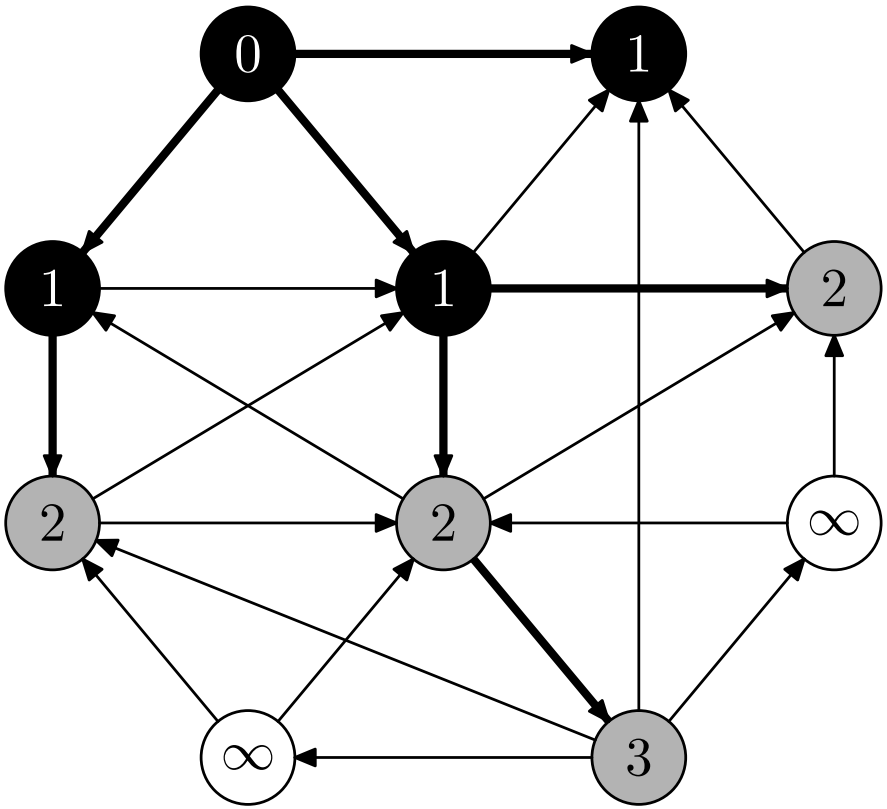
Breitensuche – Größeres Beispiel



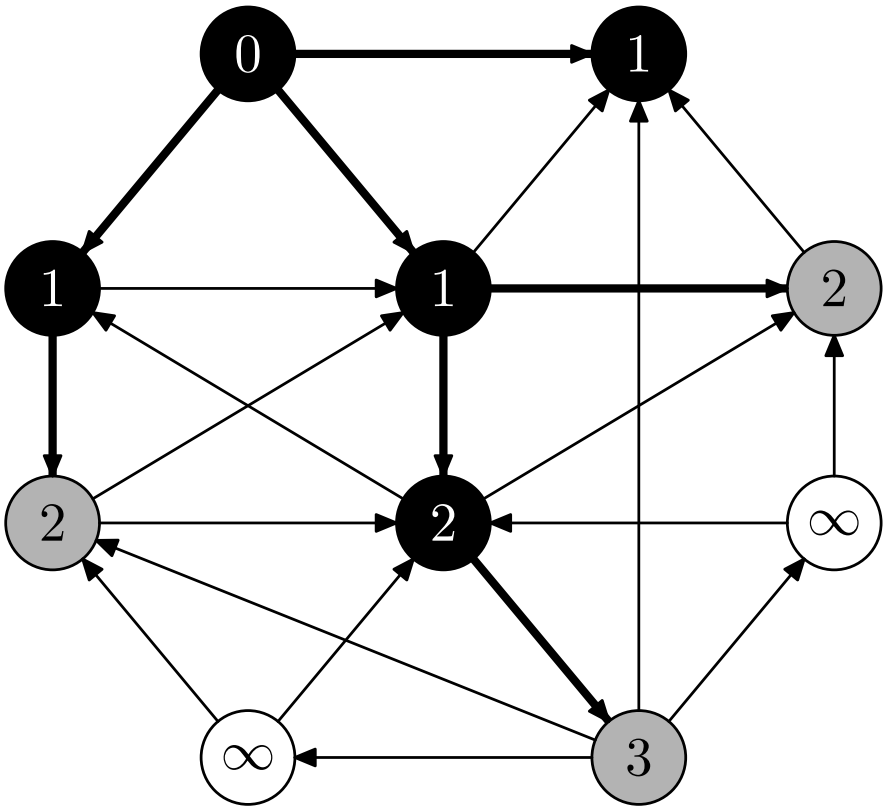
Breitensuche – Größeres Beispiel



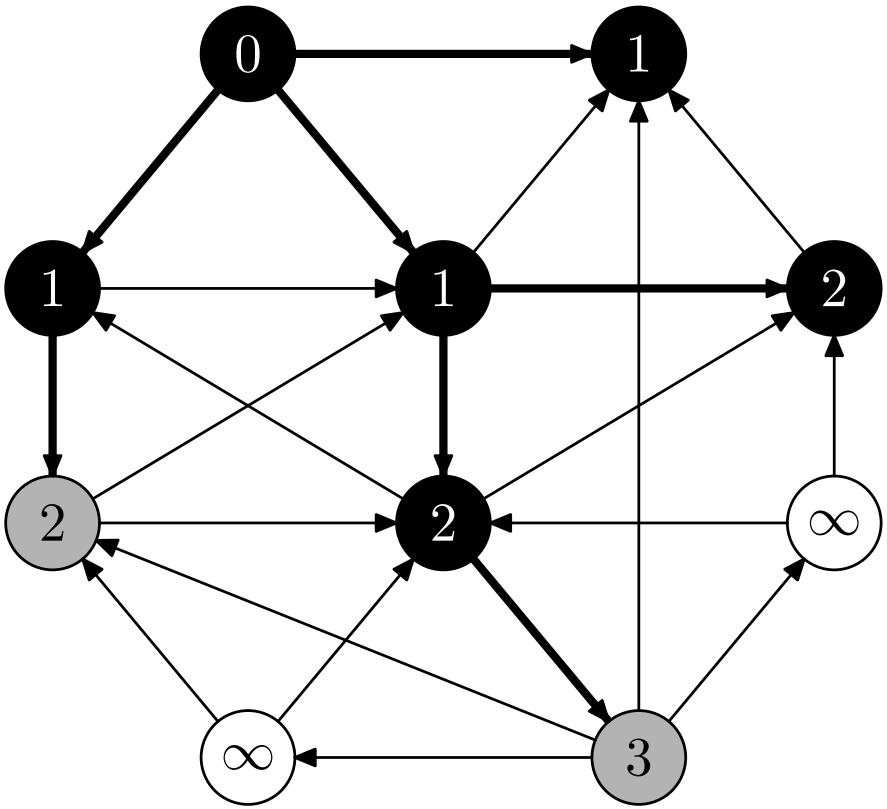
Breitensuche – Größeres Beispiel



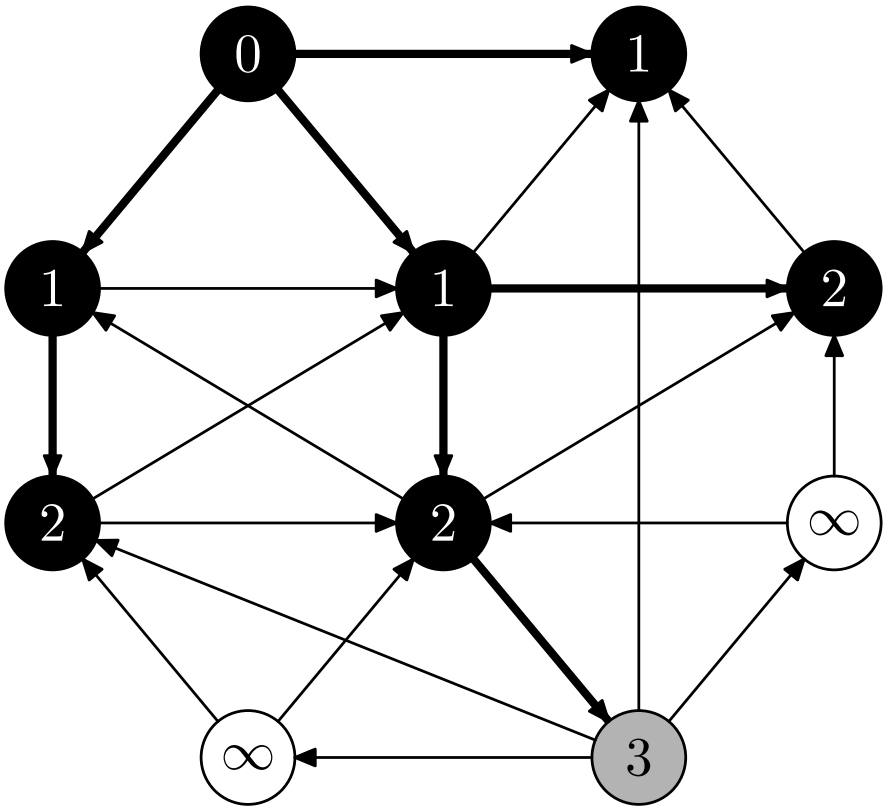
Breitensuche – Größeres Beispiel



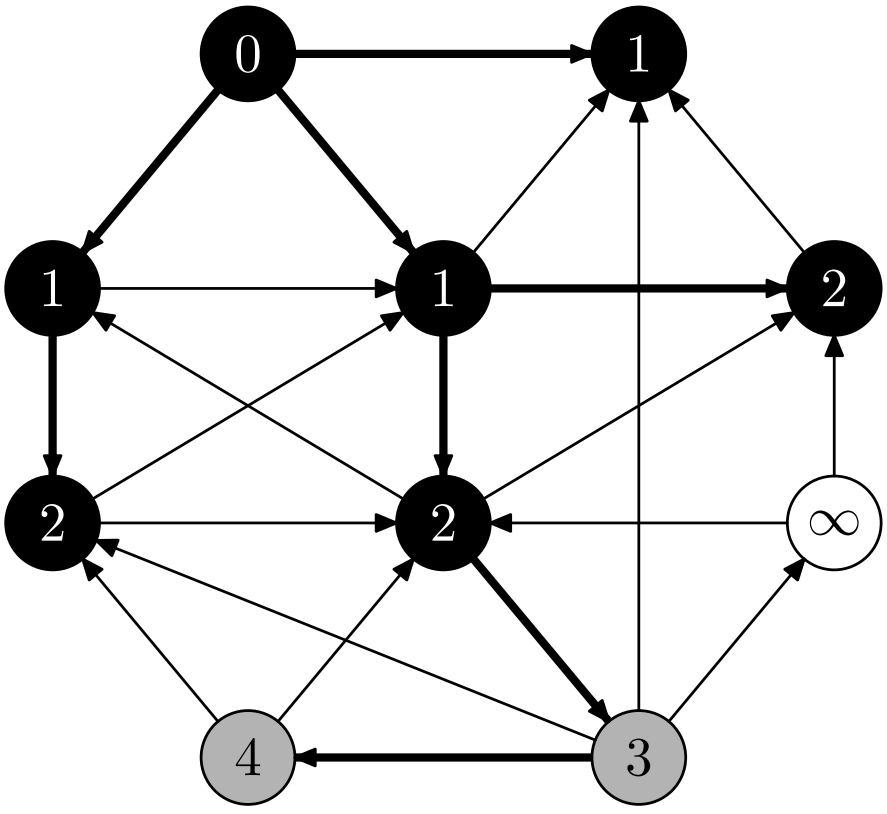
Breitensuche – Größeres Beispiel



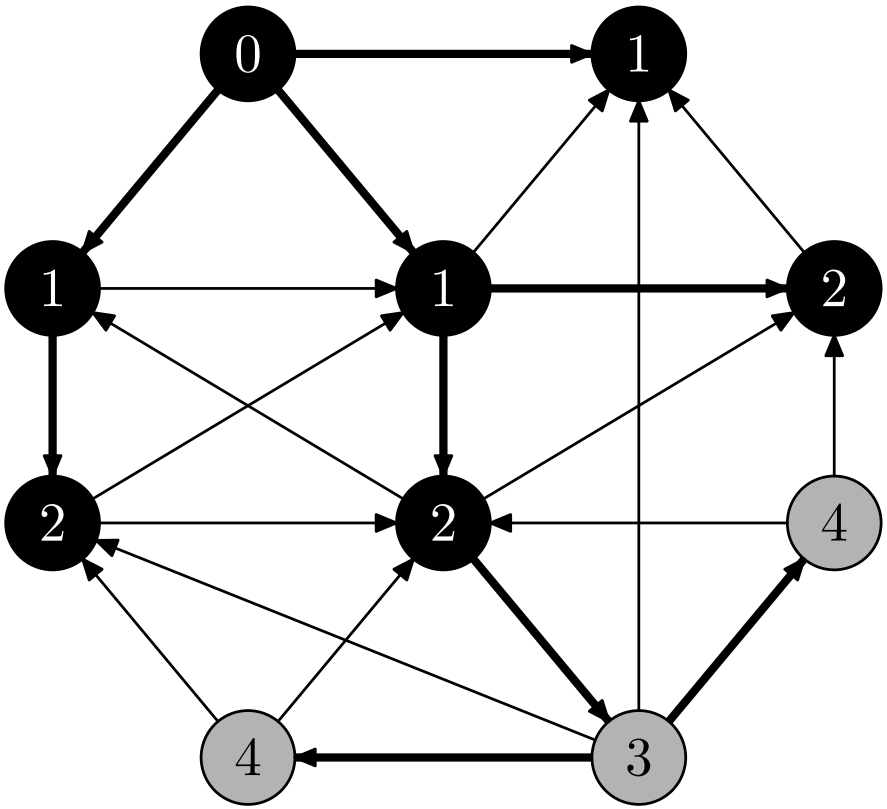
Breitensuche – Größeres Beispiel



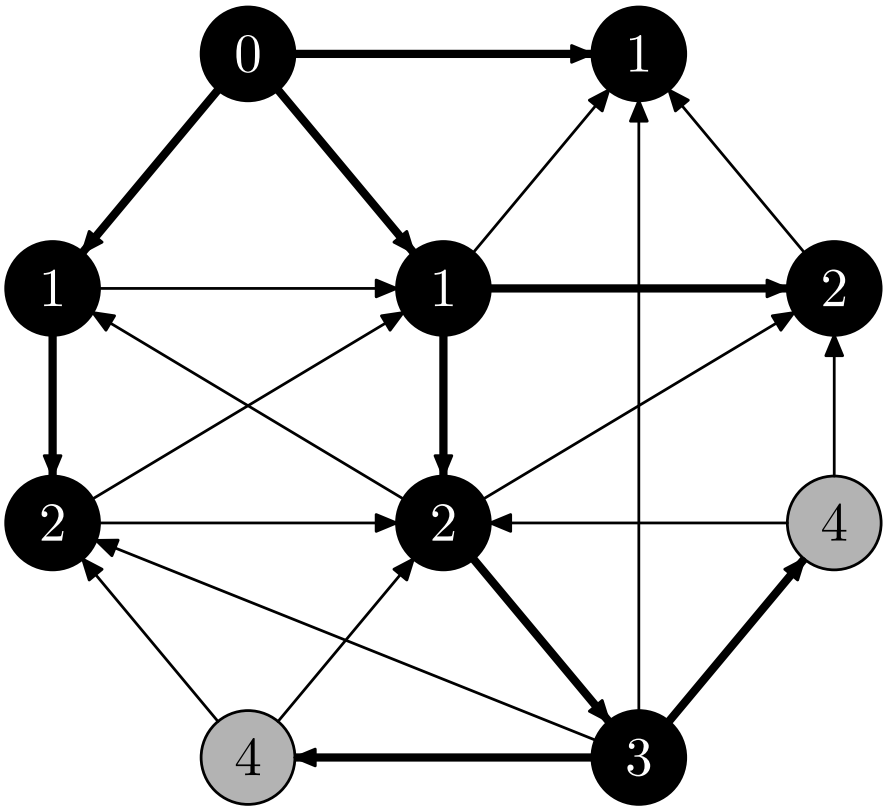
Breitensuche – Größeres Beispiel



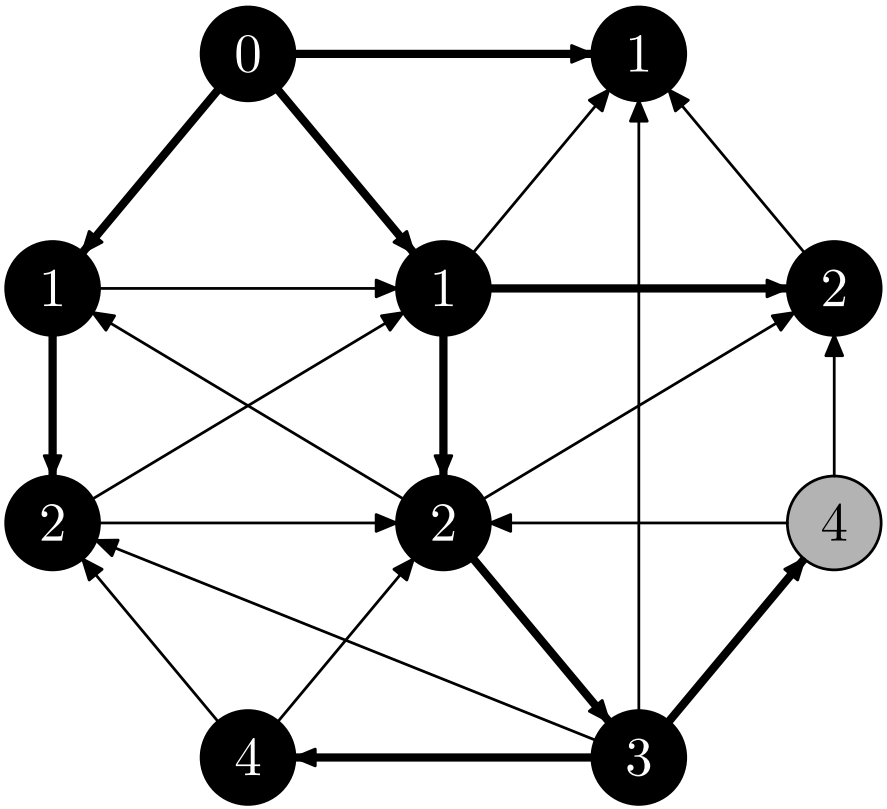
Breitensuche – Größeres Beispiel



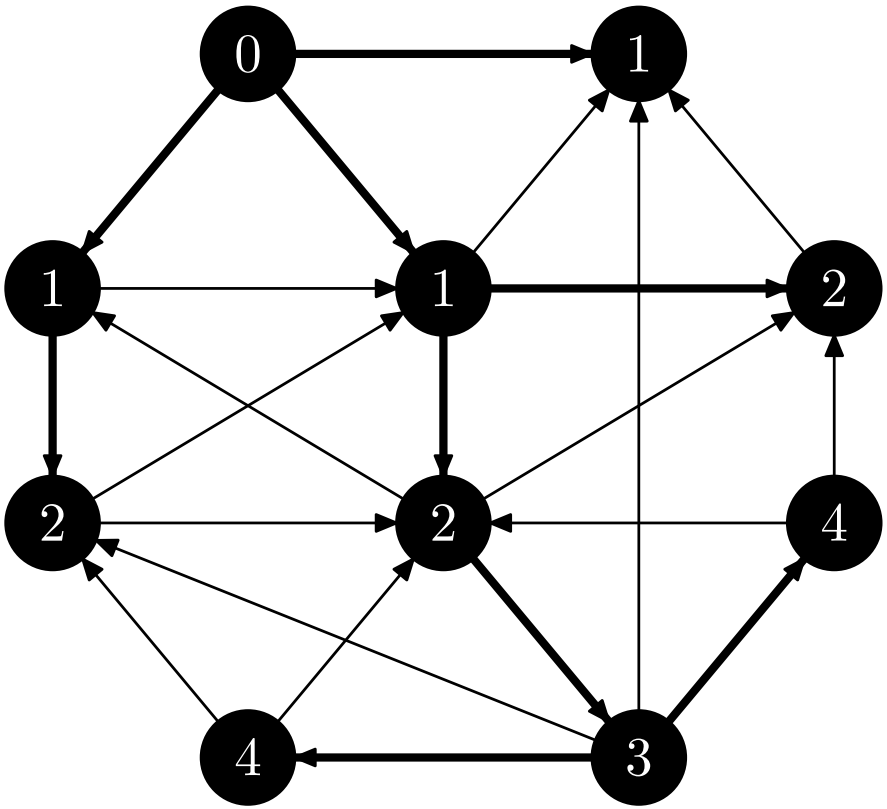
Breitensuche – Größeres Beispiel



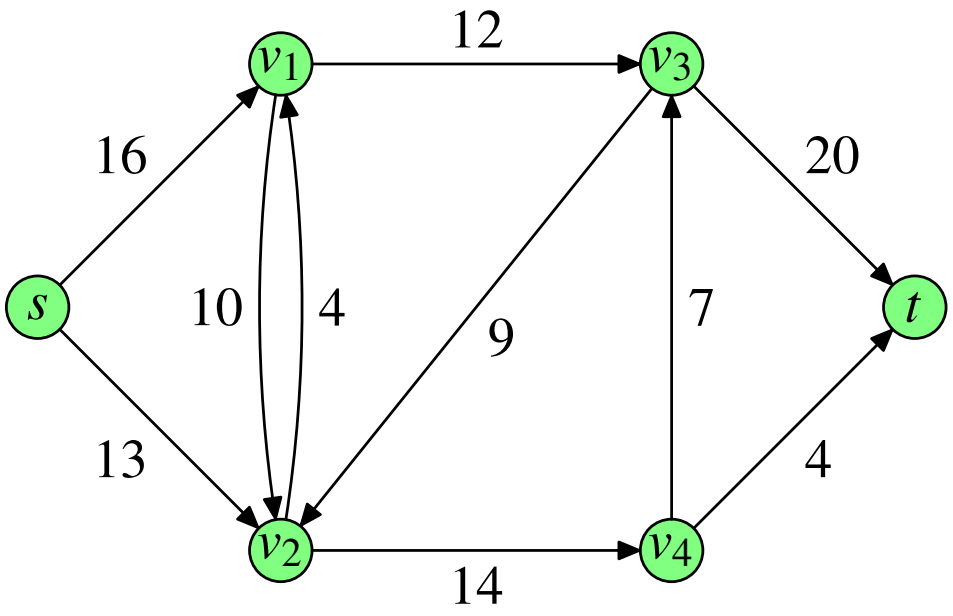
Breitensuche – Größeres Beispiel



Breitensuche – Größeres Beispiel



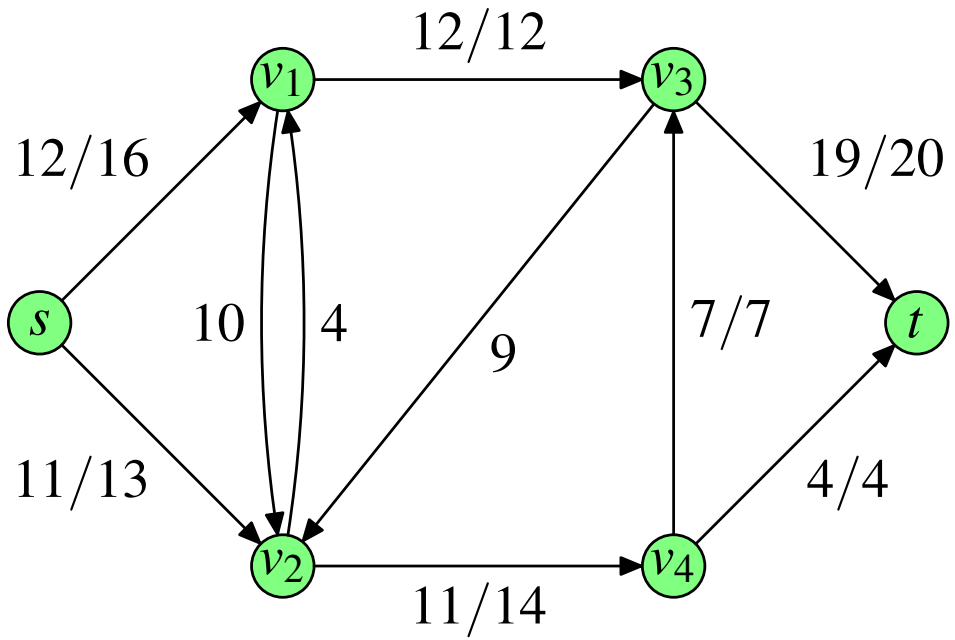
Was ist der maximale Fluß?



Der maximale Fluß ist 23.

Die Kanten sind mit $c(u, v)$ beschriftet oder mit $f(u, v)/c(u, v)$, falls $f(u, v) > 0$.

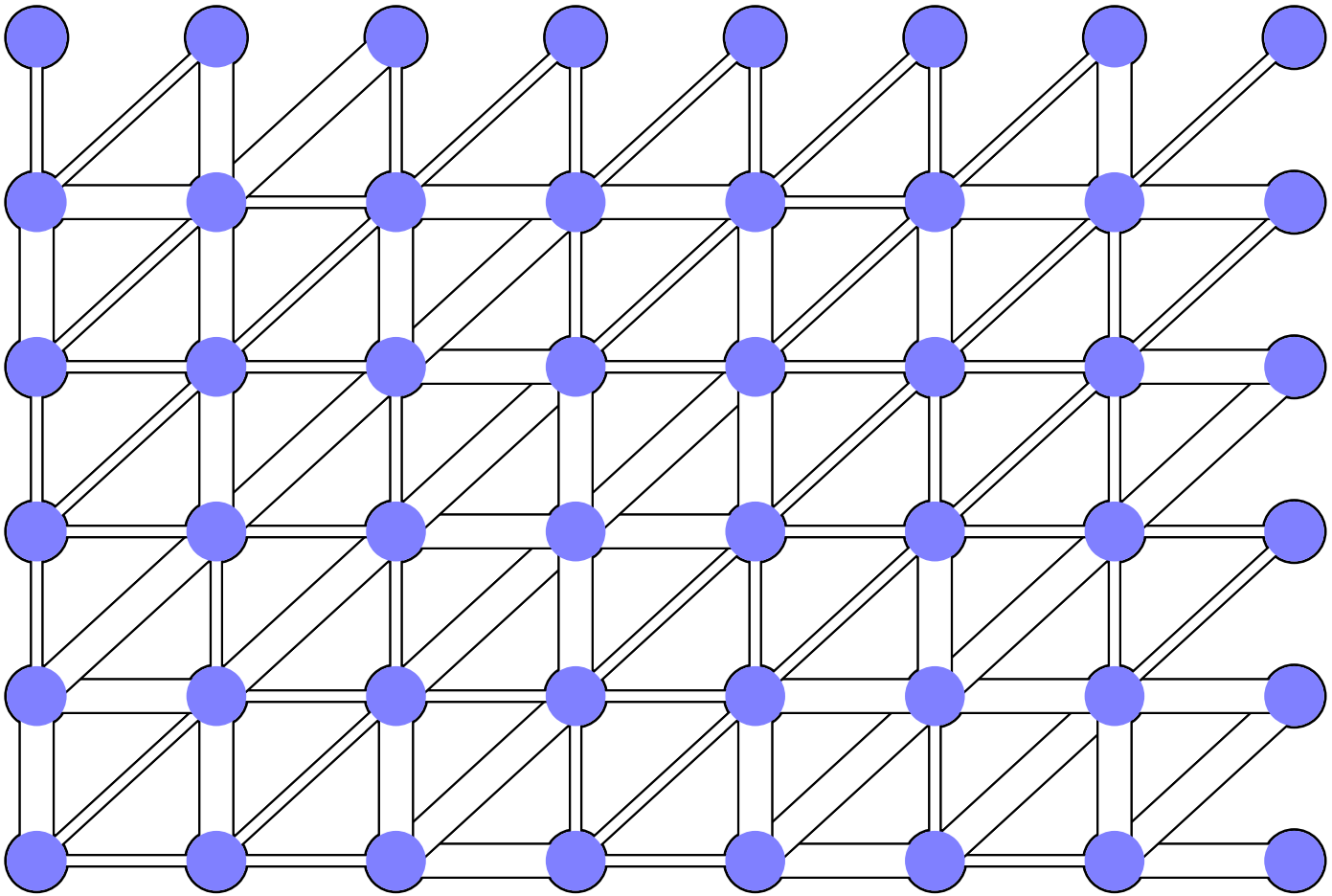
Was ist der maximale Fluß?



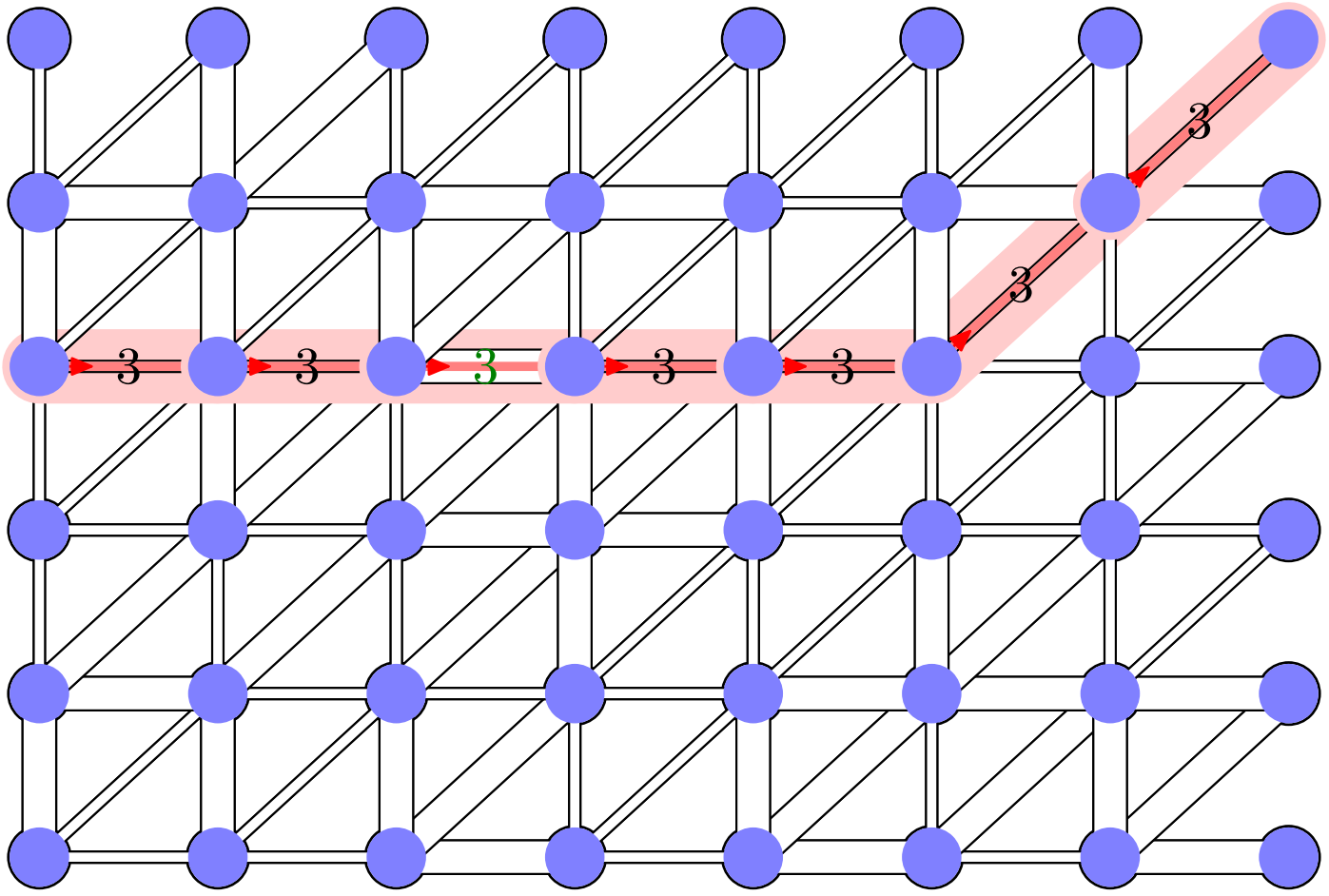
Der maximale Fluß ist 23.

Die Kanten sind mit $c(u, v)$ beschriftet oder mit $f(u, v)/c(u, v)$, falls $f(u, v) > 0$.

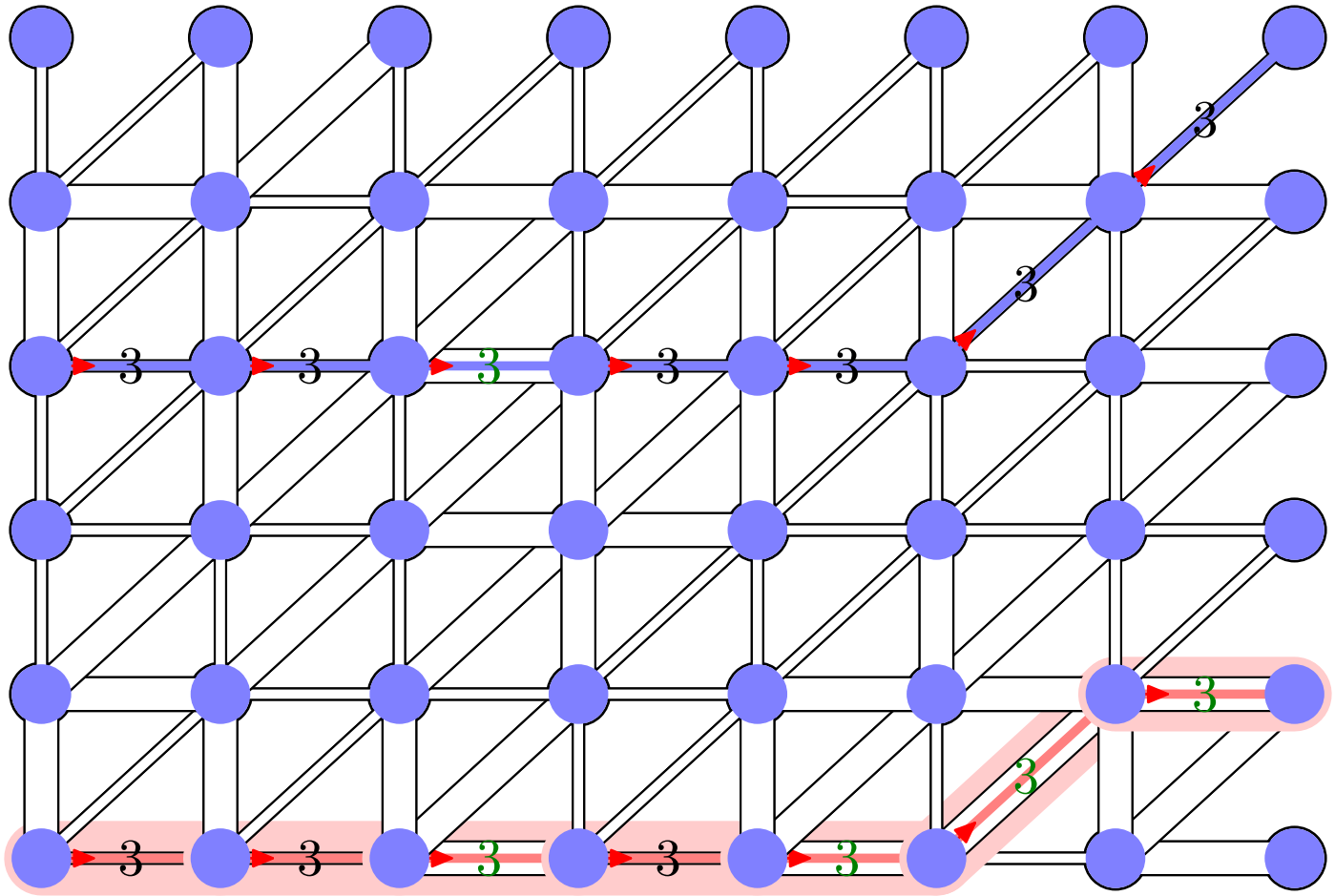
Beispiel



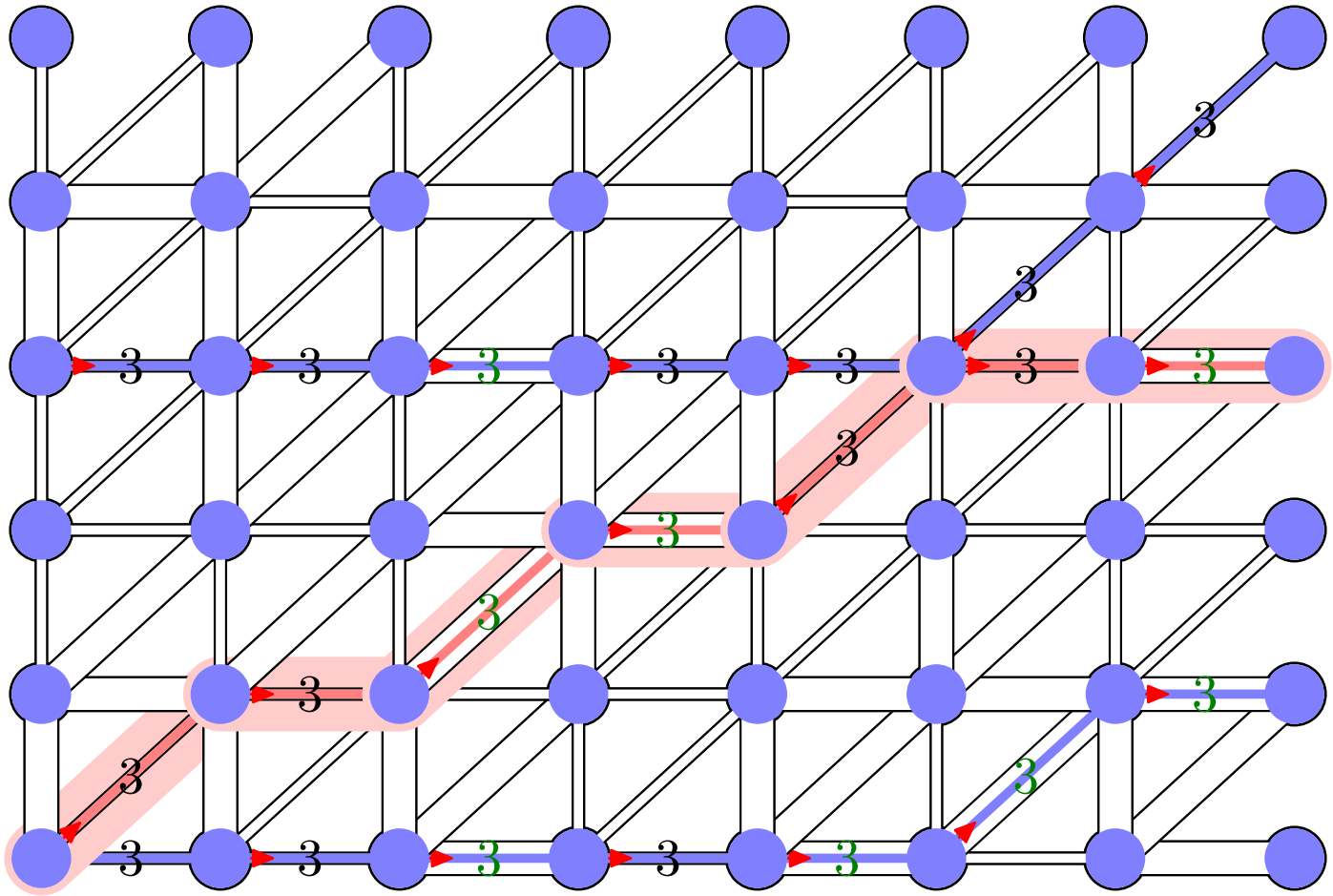
Beispiel



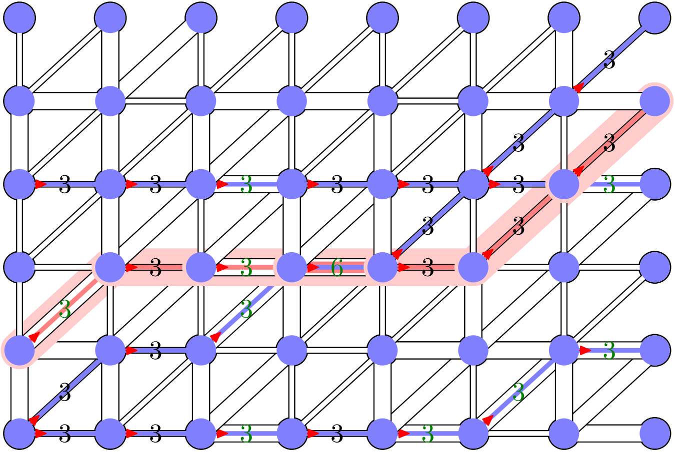
Beispiel



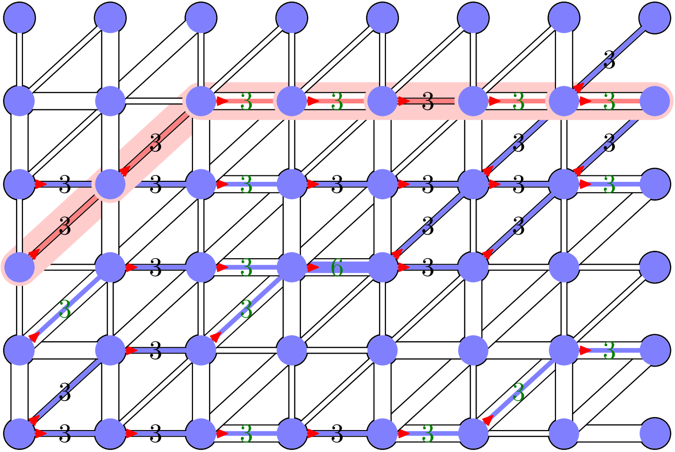
Beispiel



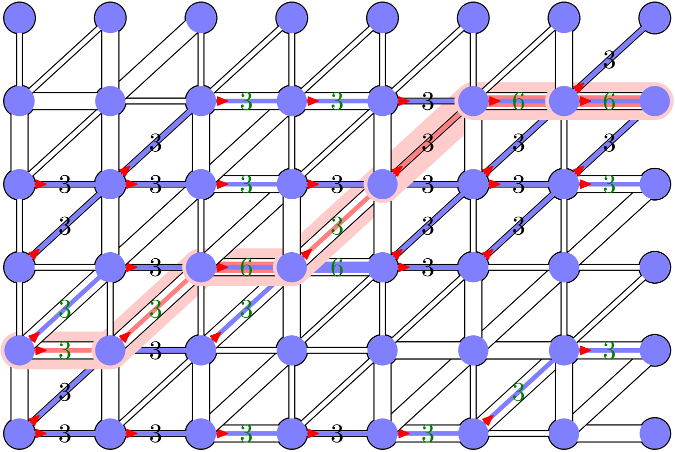
Beispiel



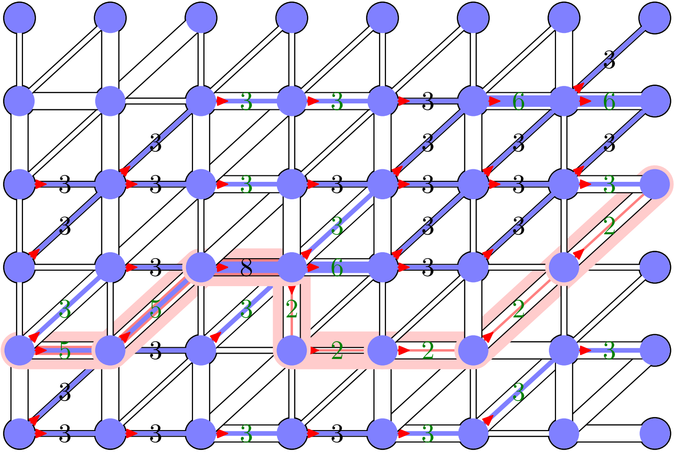
Beispiel



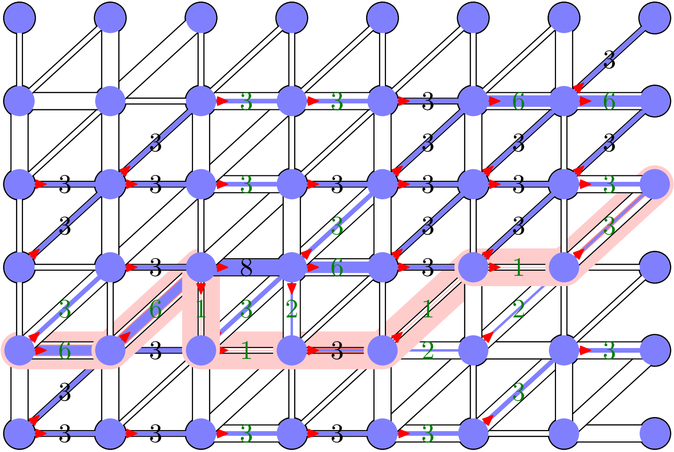
Beispiel



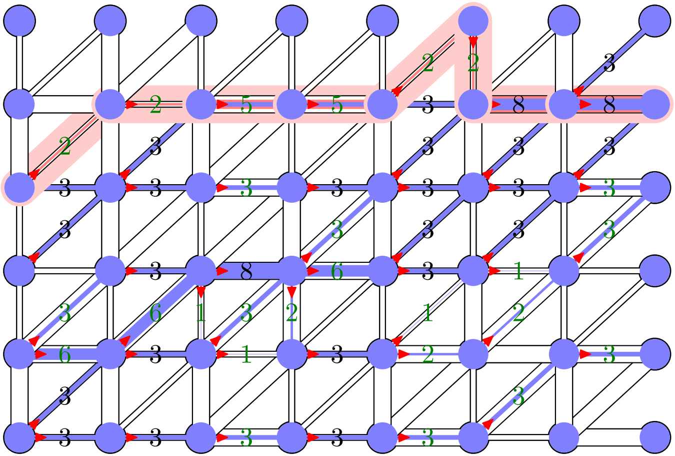
Beispiel



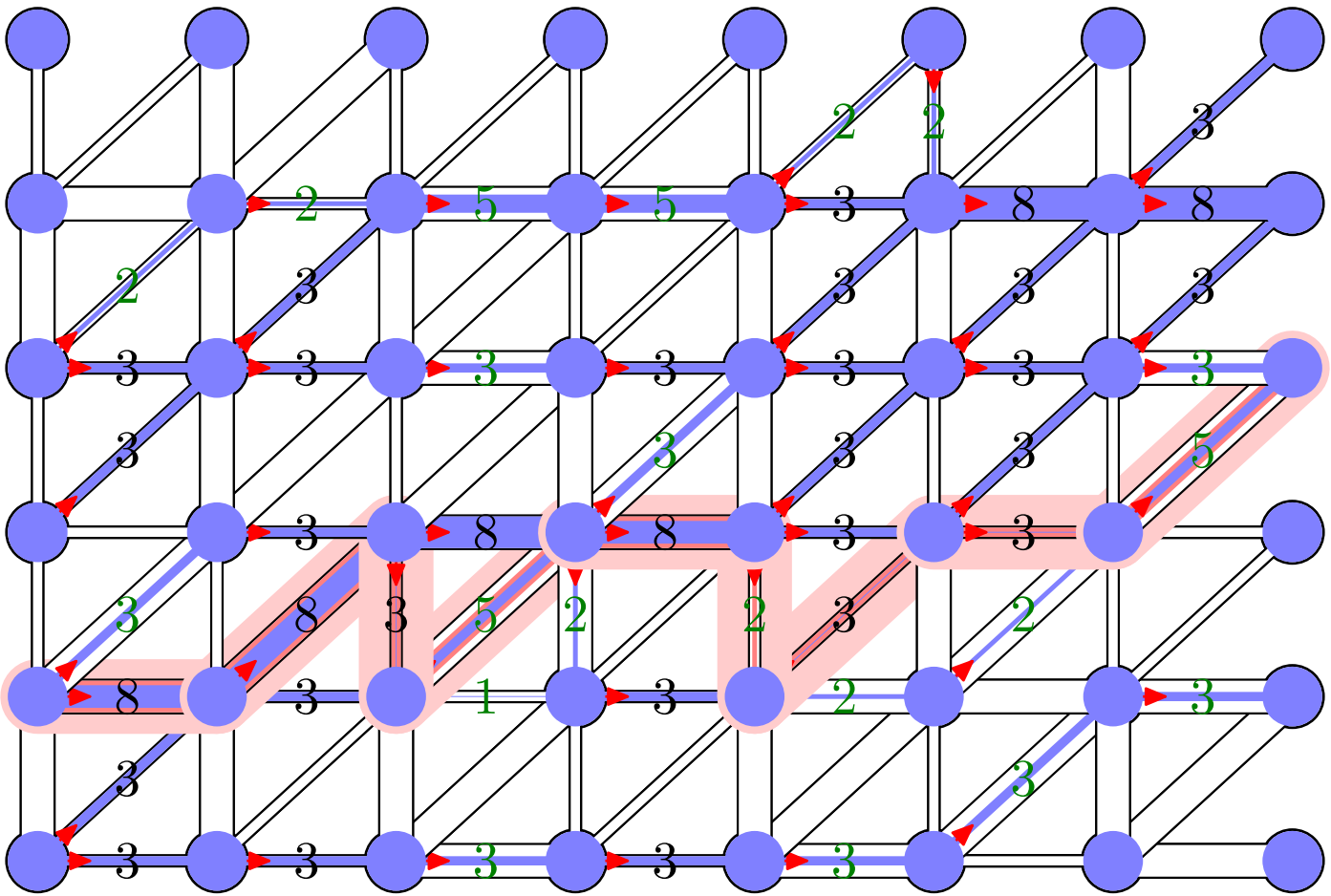
Beispiel



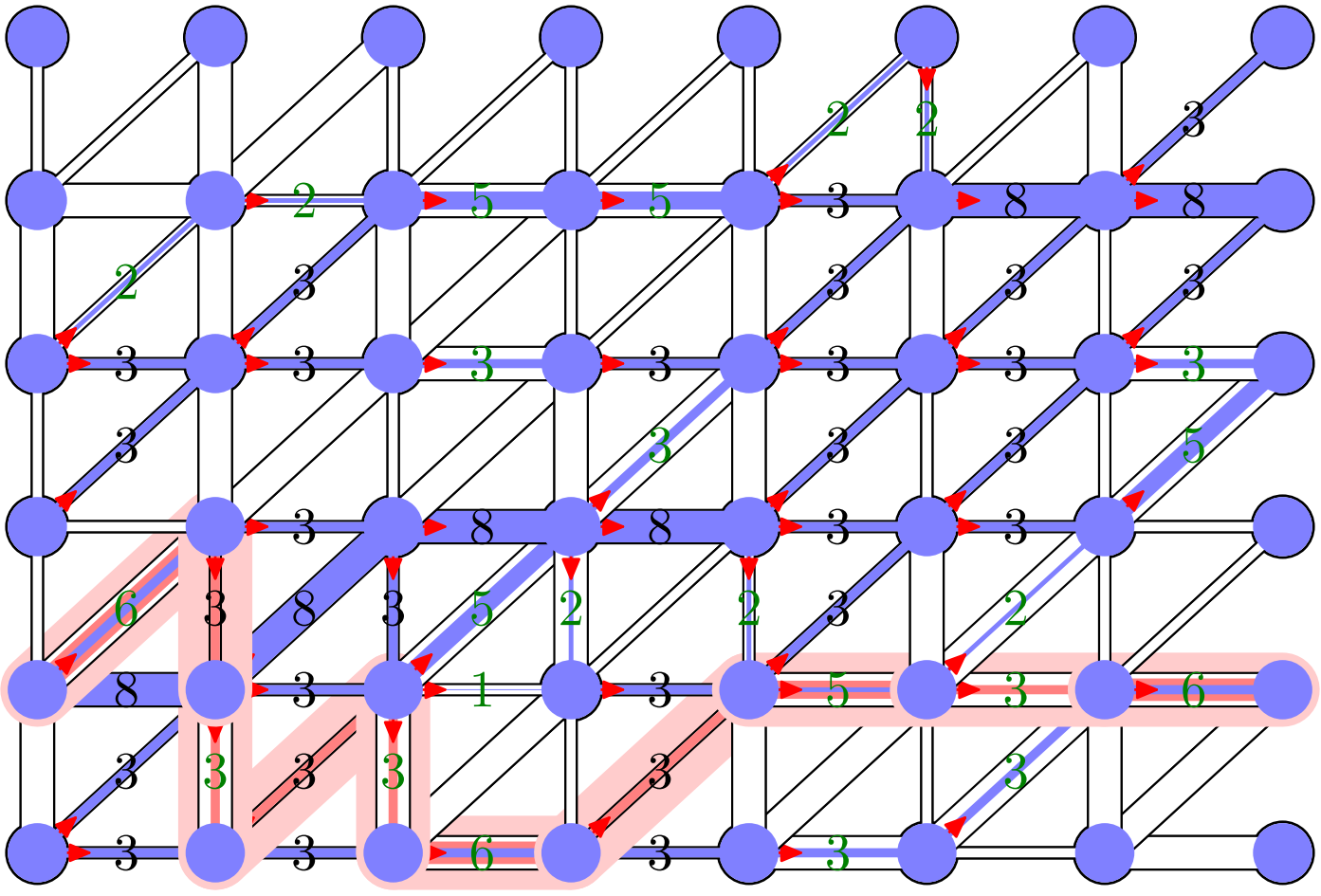
Beispiel


2 2 3
2 5 5 3 8 8
2 3 3 3 3
3 3 3 3 3 3 3
3 3 3 3 3
3 8 6 3 1
3 6 1 3 2 1 2
6 3 1 3 2 3
3 3
3 3 3 3 3

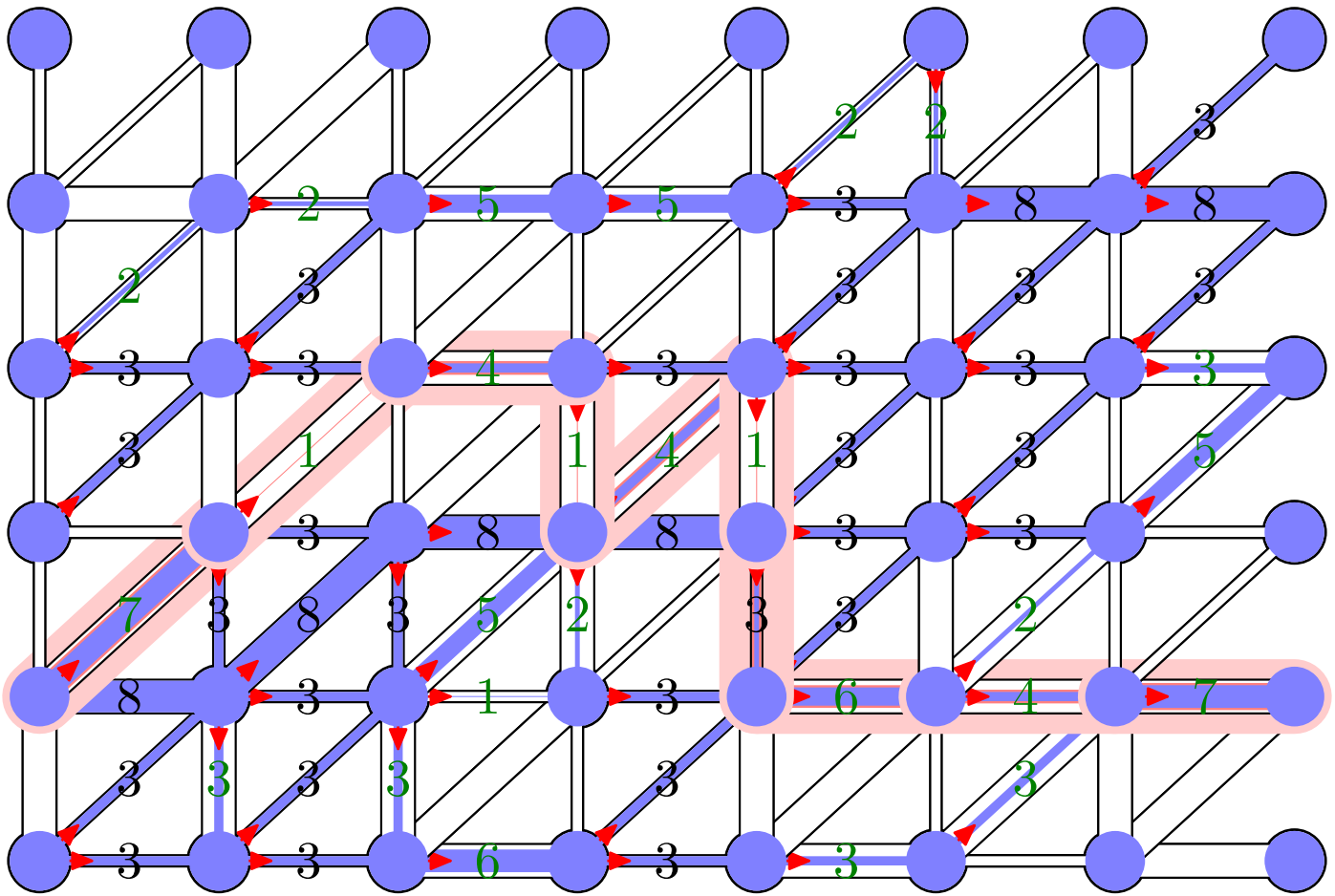
Beispiel


2 2 3
2 5 5 3 8 8
2 3 3 3 3 3
3 3 3 3 3 3 3
3 3 3 3 3 3 3 5
3 3 8 8 3 3
3 8 3 5 2 2 3 2
8 3 3 1 3 2 3
3 3 3 3 3 3 3 3

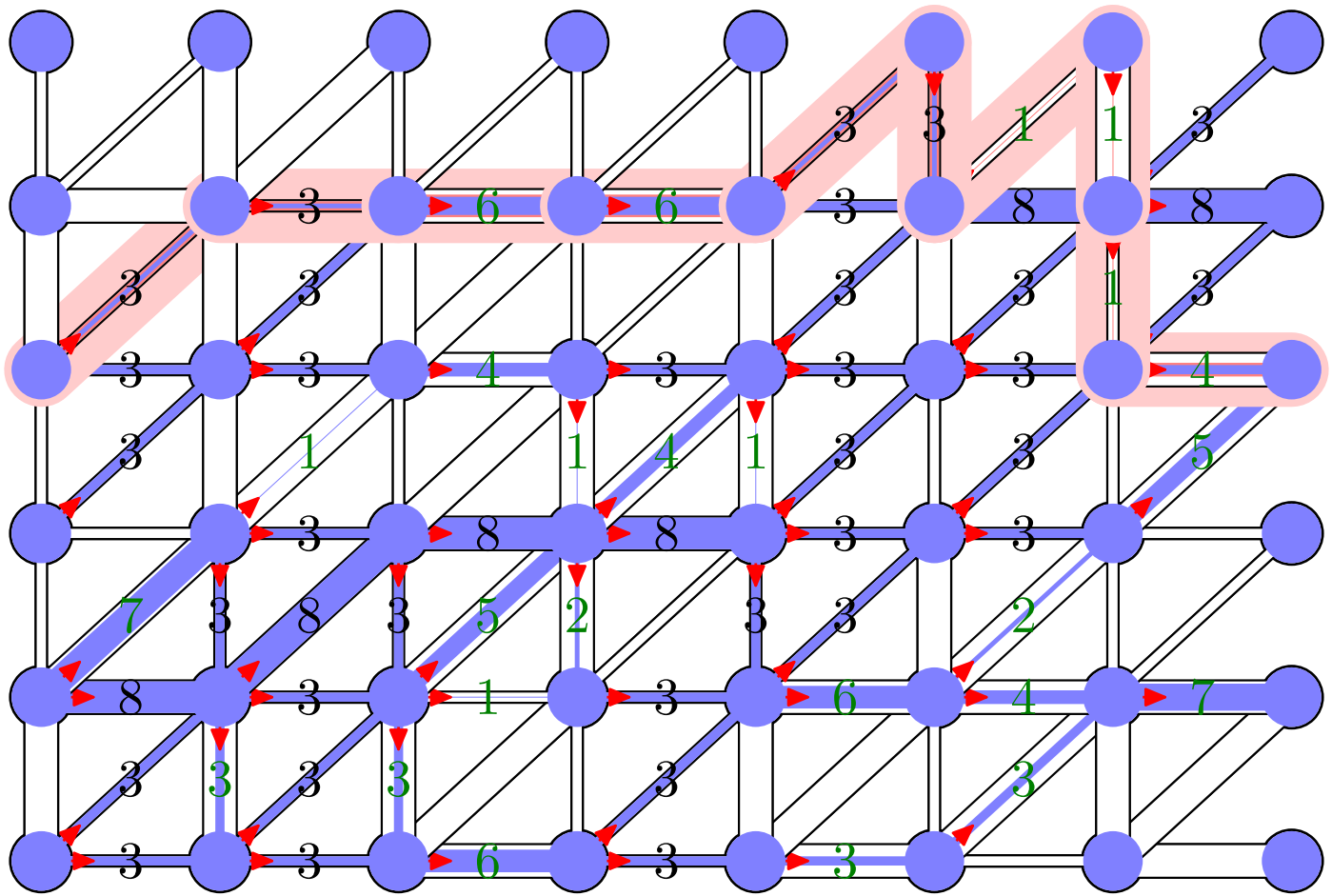
Beispiel


2 2 3
2 5 5 3 8 8
2 3 3 3 3 3
3 3 3 3 3 3 3
3 3 3 3 3 3 3 5
3 8 8 3 3 3
6 3 8 3 5 2 2 3 2
8 3 1 3 5 3 6
3 3 3 3 3
3 3 6 3 3

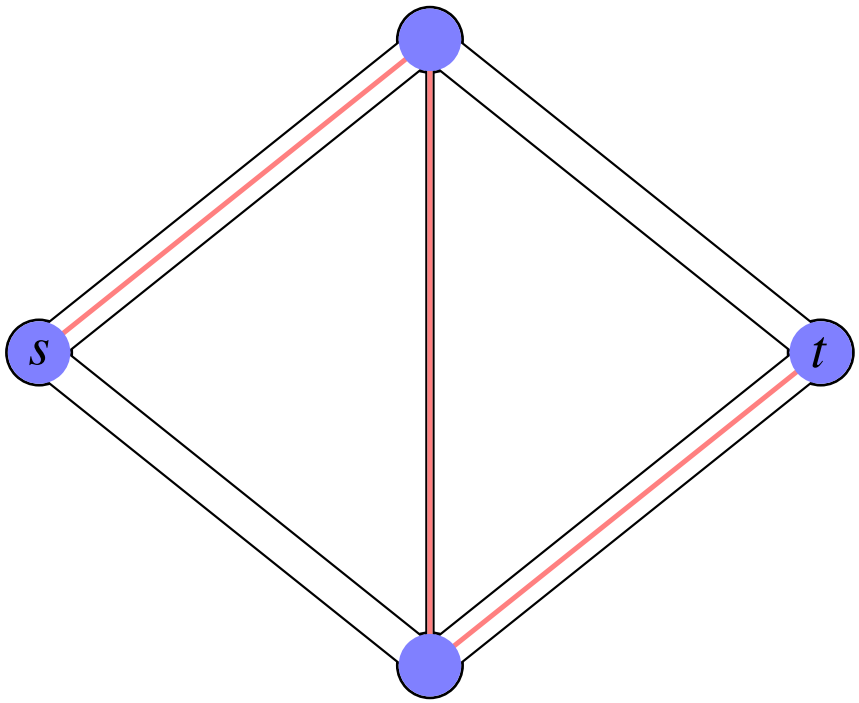
Beispiel


2 2 3
2 5 5 3 8 8
2 3 3 3 3
3 3 3 4 3 3 3 3
3 1 1 4 1 3 3 5
3 8 8 3 3
7 3 8 3 5 2 3 2
8 3 1 3 6 4 7
3 3 3 3 3
3 3 6 3 3

Beispiel

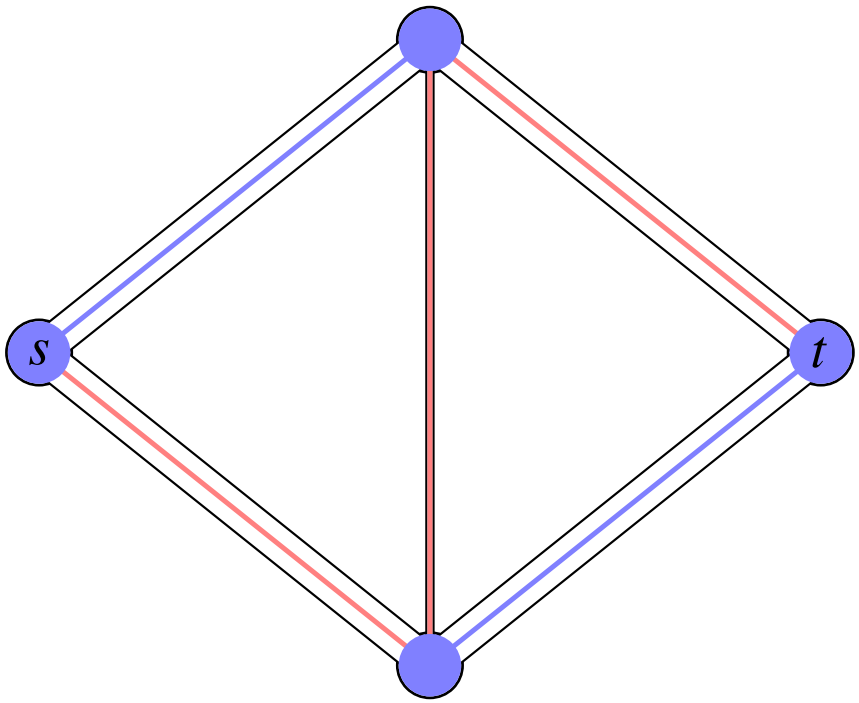

3 3 1 1 3
3 6 6 3 8 8
3 3 1 3
3 3 4 3 3 4
1 1 4 1 5
3 8 8 3
7 3 8 3 5 2 2
8 3 1 3 6 4 7
3 3 3 3 3
3 6 3

Laufzeit der Ford–Fulkerson–Methode



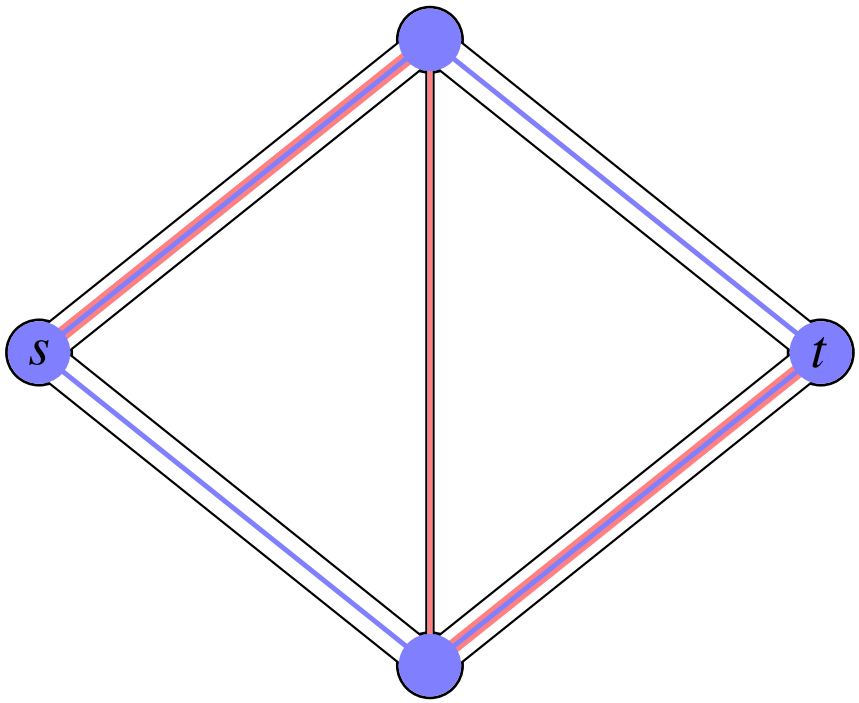
Die Laufzeit kann beliebig schlecht sein.

Laufzeit der Ford–Fulkerson–Methode



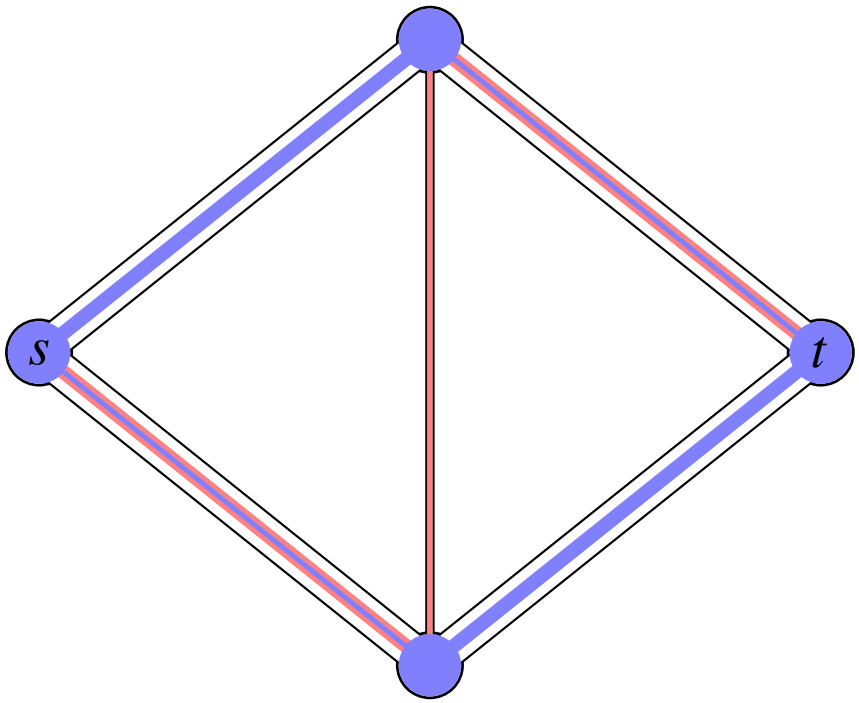
Die Laufzeit kann beliebig schlecht sein.

Laufzeit der Ford–Fulkerson–Methode



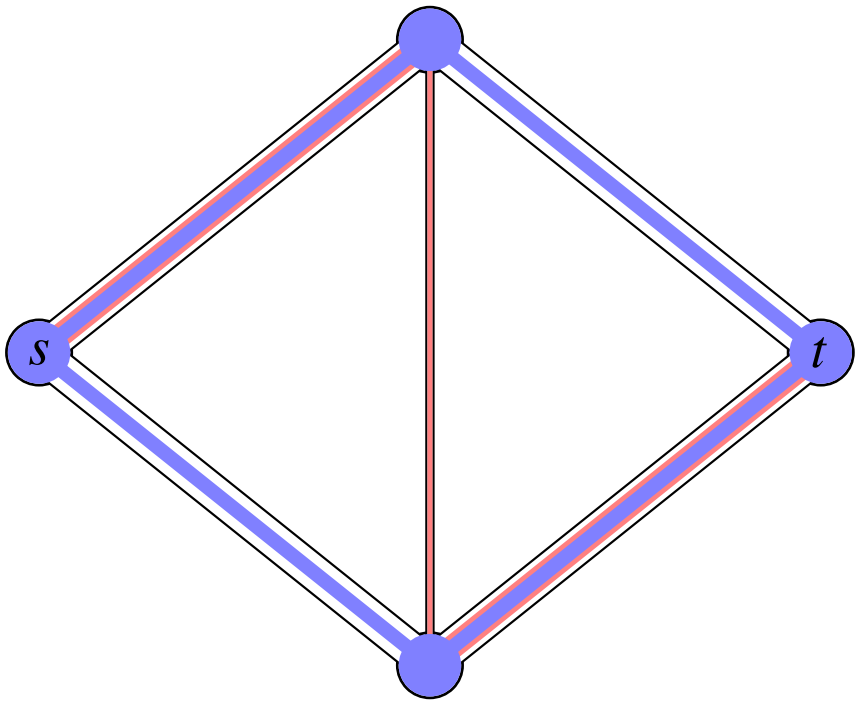
Die Laufzeit kann beliebig schlecht sein.

Laufzeit der Ford–Fulkerson–Methode



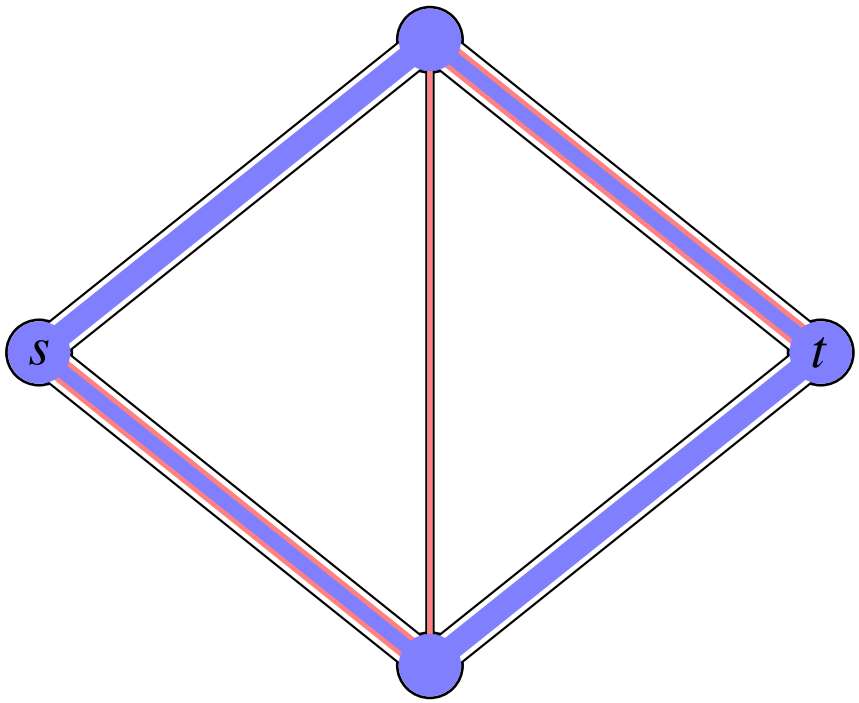
Die Laufzeit kann beliebig schlecht sein.

Laufzeit der Ford–Fulkerson–Methode



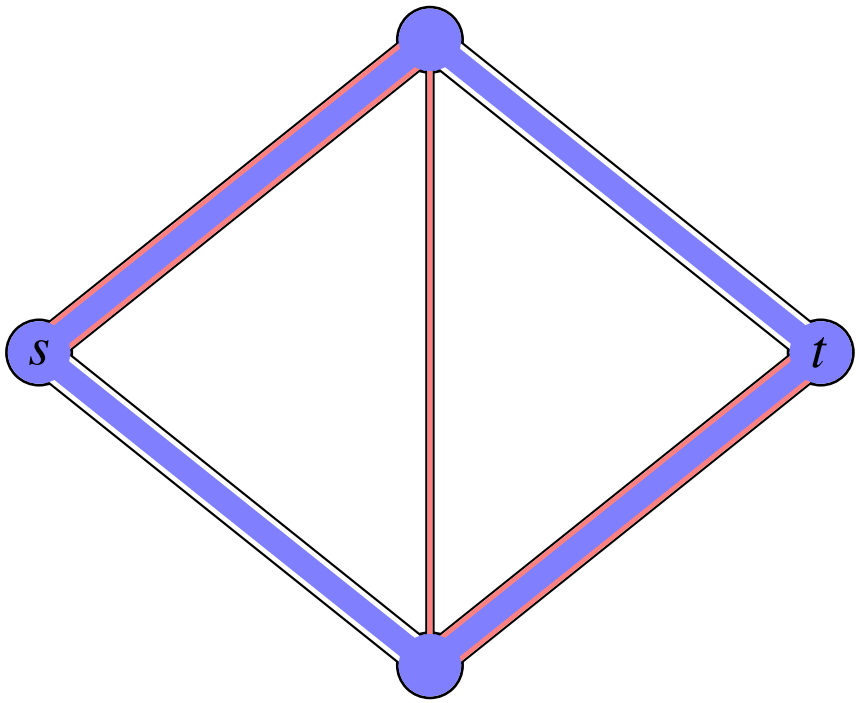
Die Laufzeit kann beliebig schlecht sein.

Laufzeit der Ford–Fulkerson–Methode



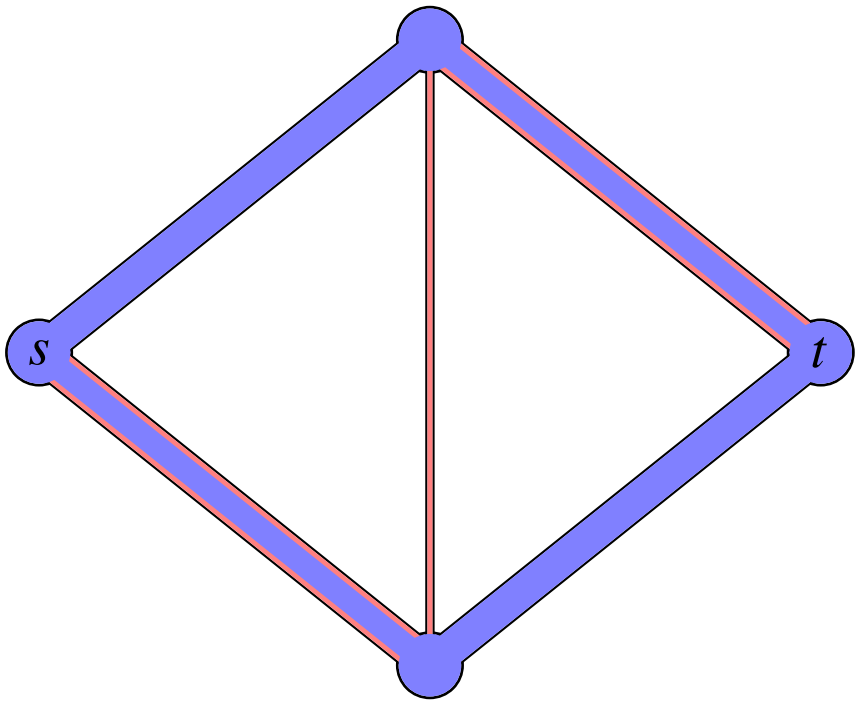
Die Laufzeit kann beliebig schlecht sein.

Laufzeit der Ford–Fulkerson–Methode



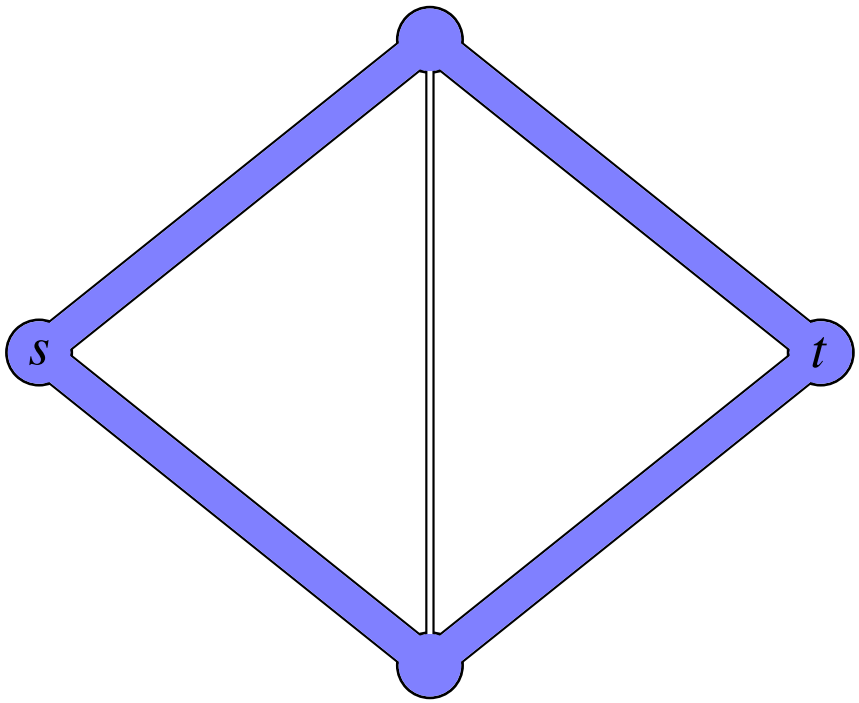
Die Laufzeit kann beliebig schlecht sein.

Laufzeit der Ford–Fulkerson–Methode



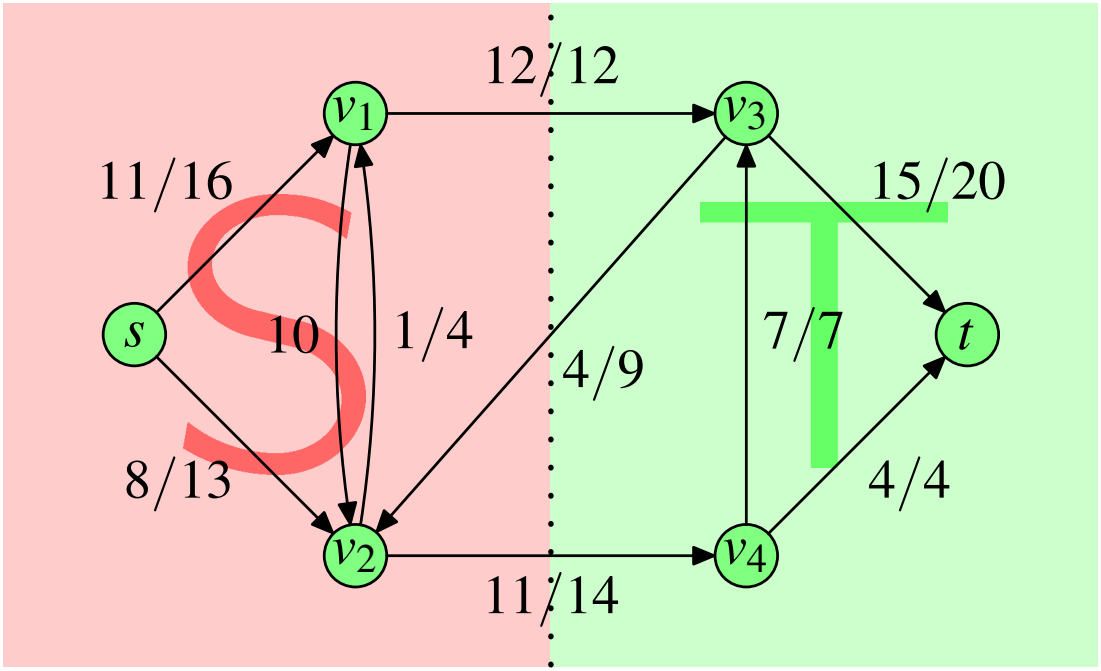
Die Laufzeit kann beliebig schlecht sein.

Laufzeit der Ford–Fulkerson–Methode



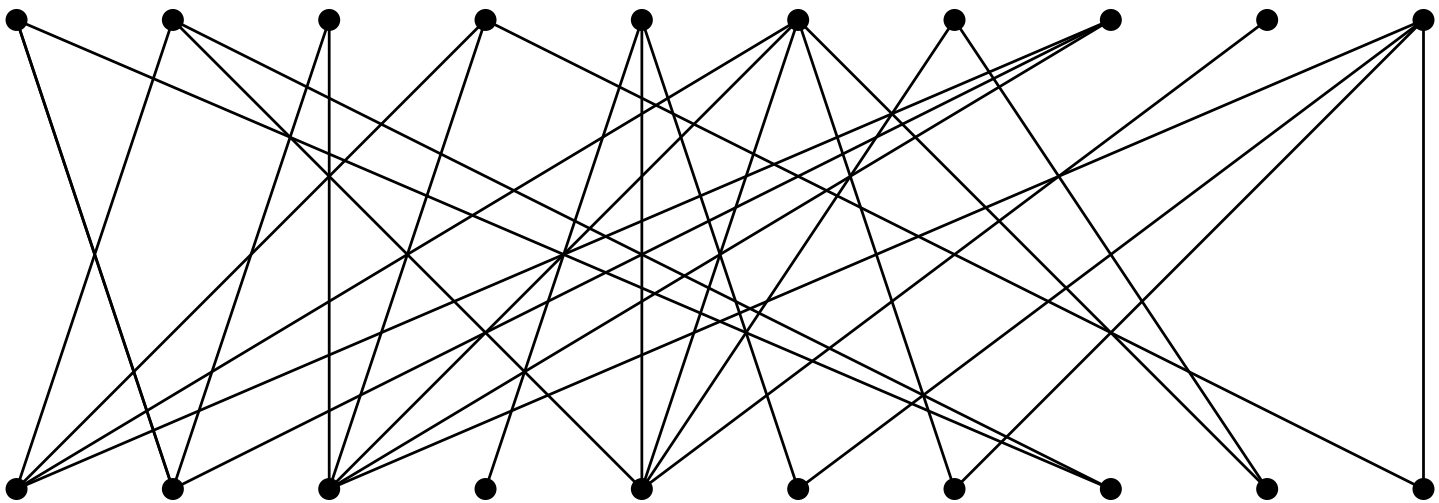
Die Laufzeit kann beliebig schlecht sein.

Laufzeit der Ford–Fulkerson–Methode

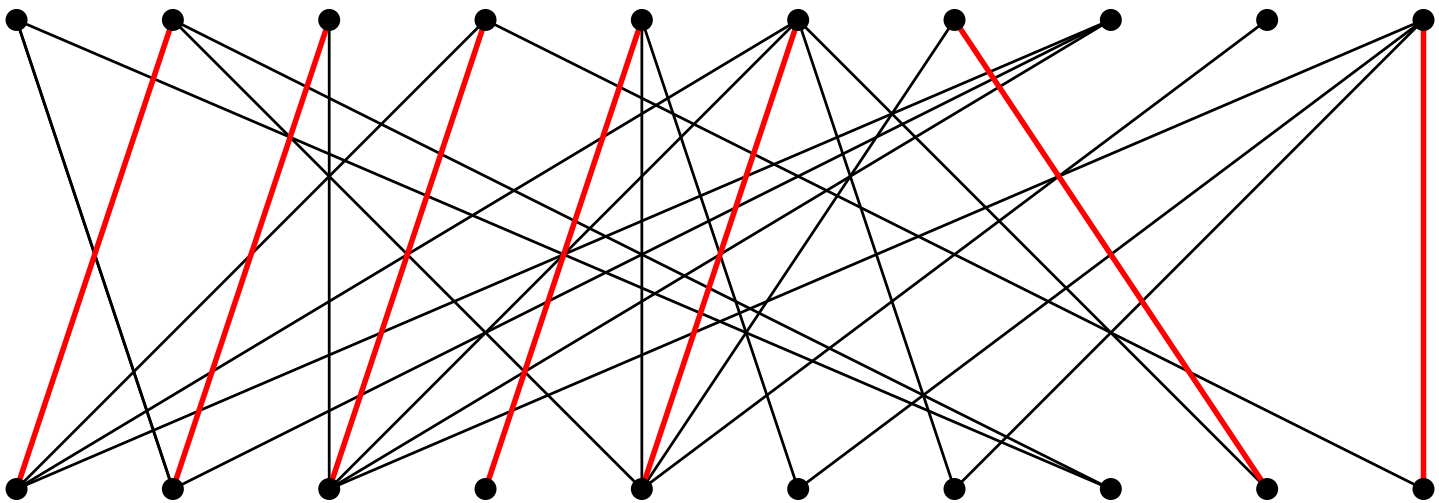


Die Laufzeit kann beliebig schlecht sein.

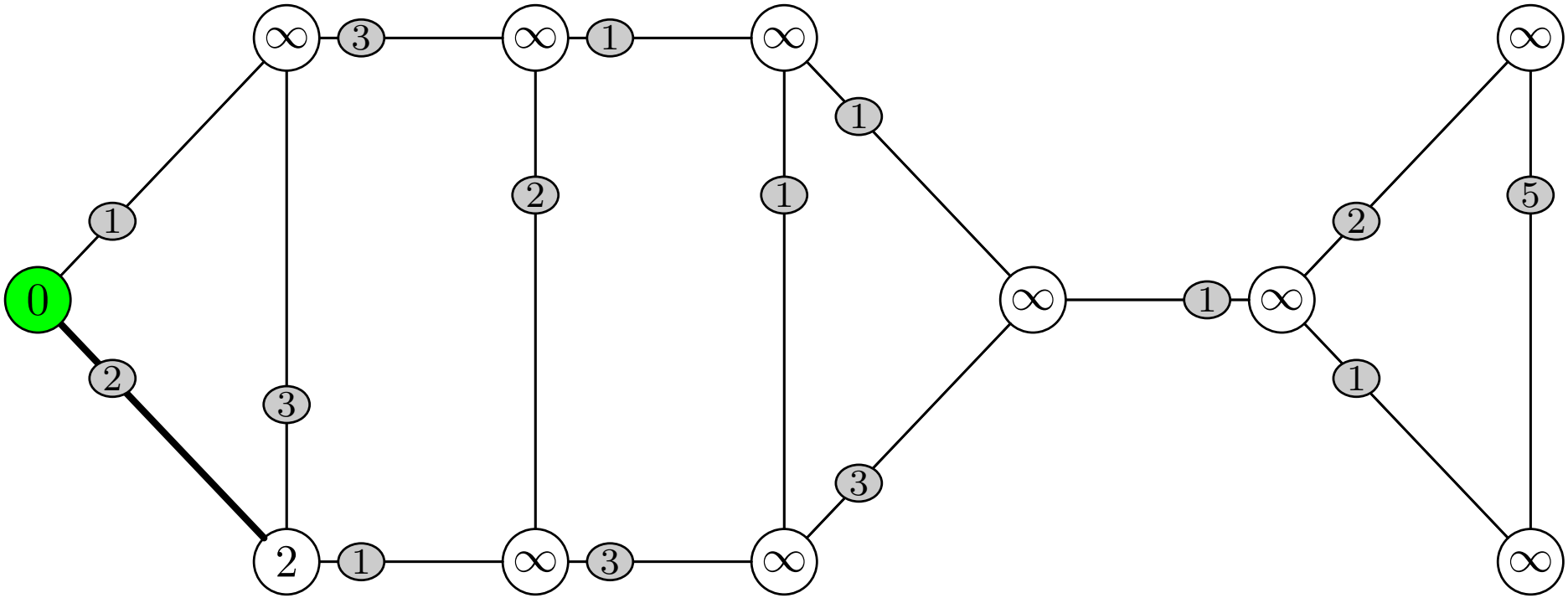
Beispiel



Beispiel

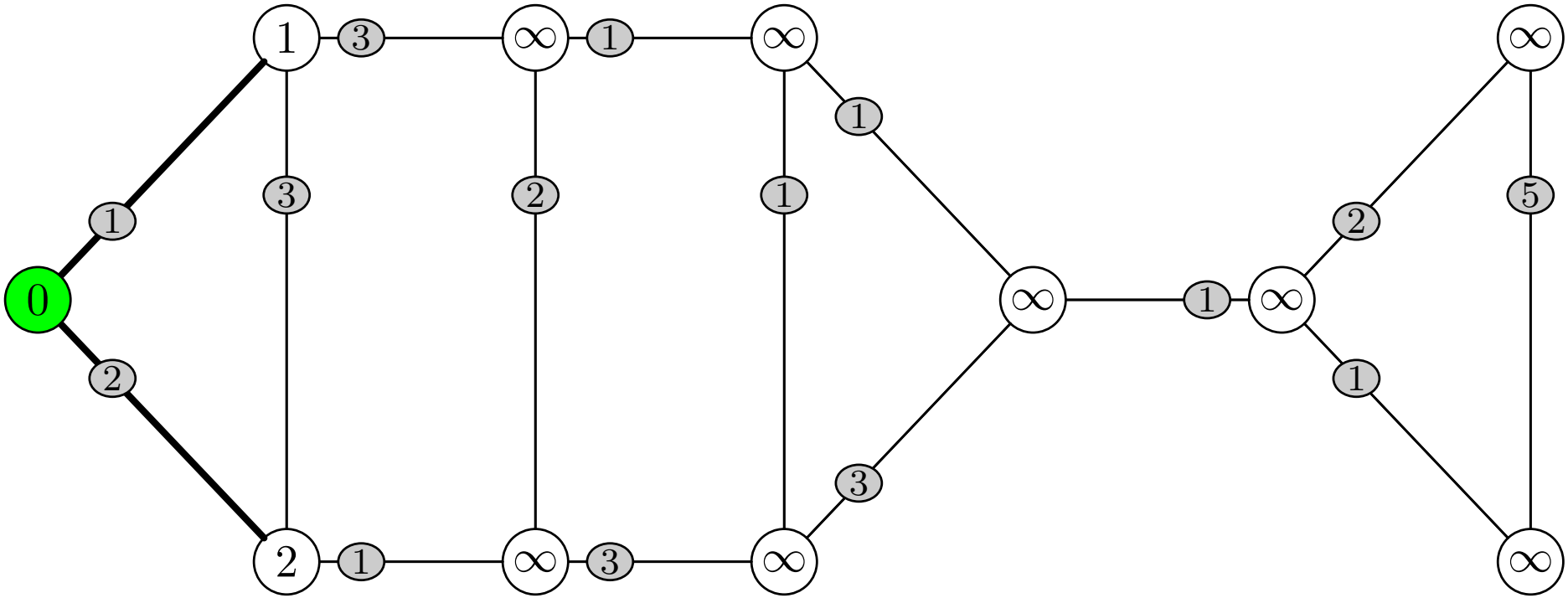


Der Algorithmus von Prim – Beispiel



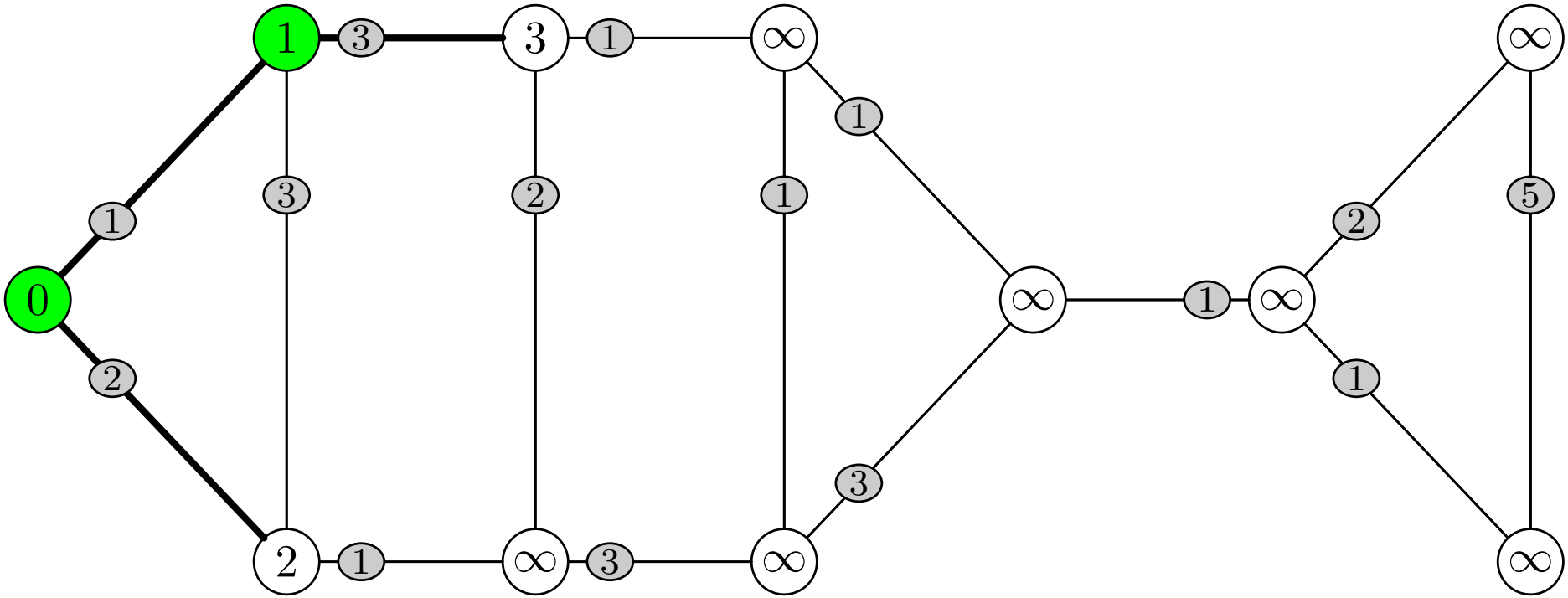
- Beginne mit leerem Baum (nur Wurzel)
- Hänge wiederholt billigste Kante an ohne einen Kreis zu schließen

Der Algorithmus von Prim – Beispiel



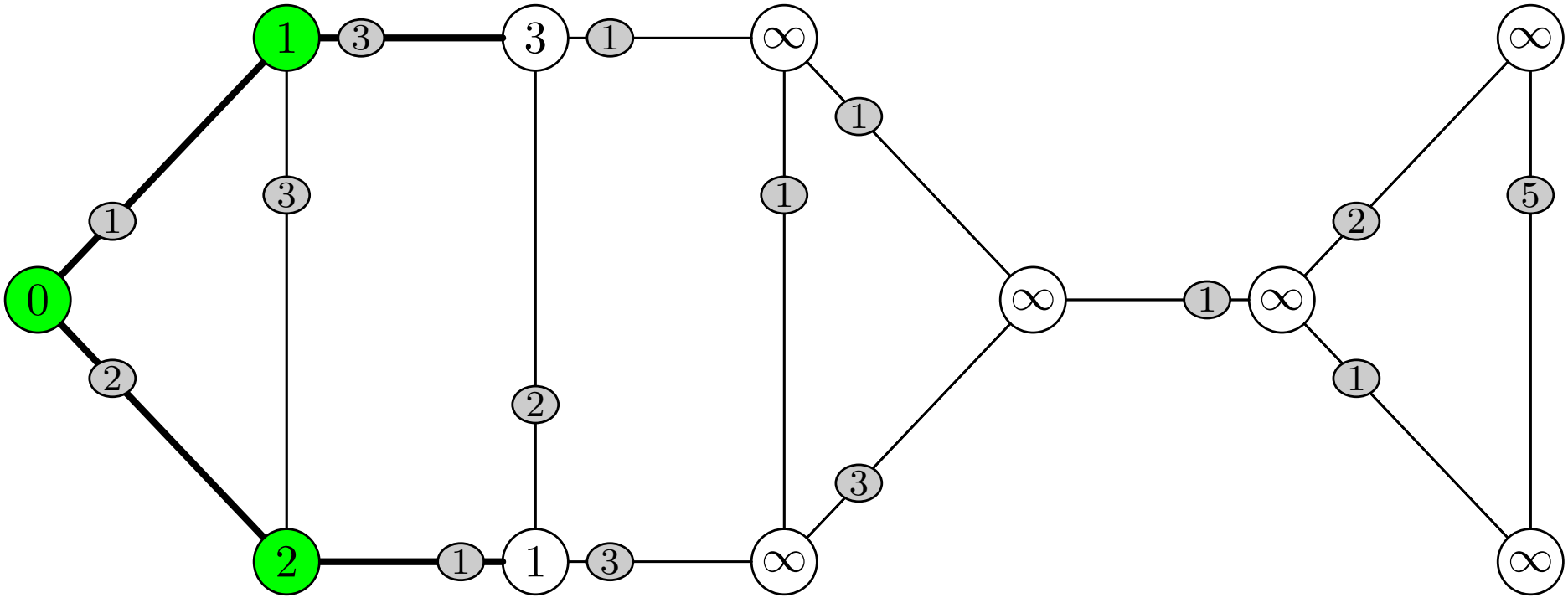
- Beginne mit leerem Baum (nur Wurzel)
- Hänge wiederholt billigste Kante an ohne einen Kreis zu schließen

Der Algorithmus von Prim – Beispiel



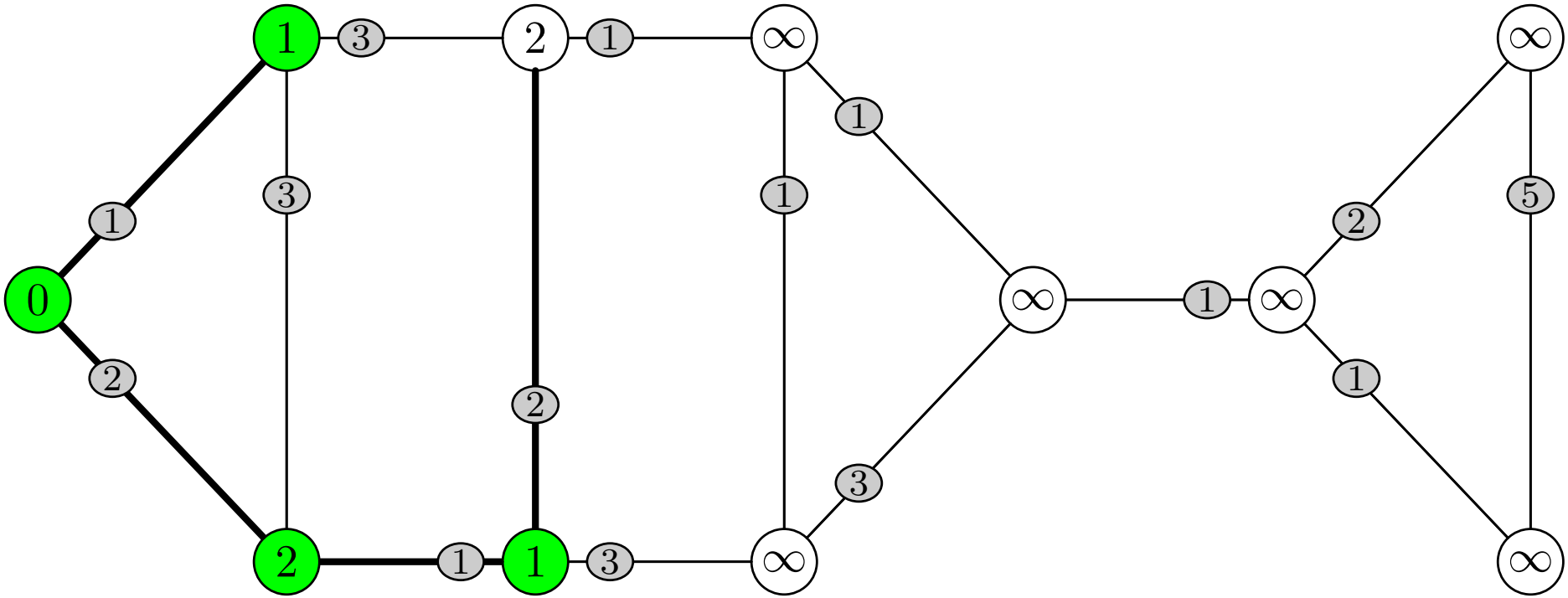
- Beginne mit leerem Baum (nur Wurzel)
- Hänge wiederholt billigste Kante an ohne einen Kreis zu schließen

Der Algorithmus von Prim – Beispiel



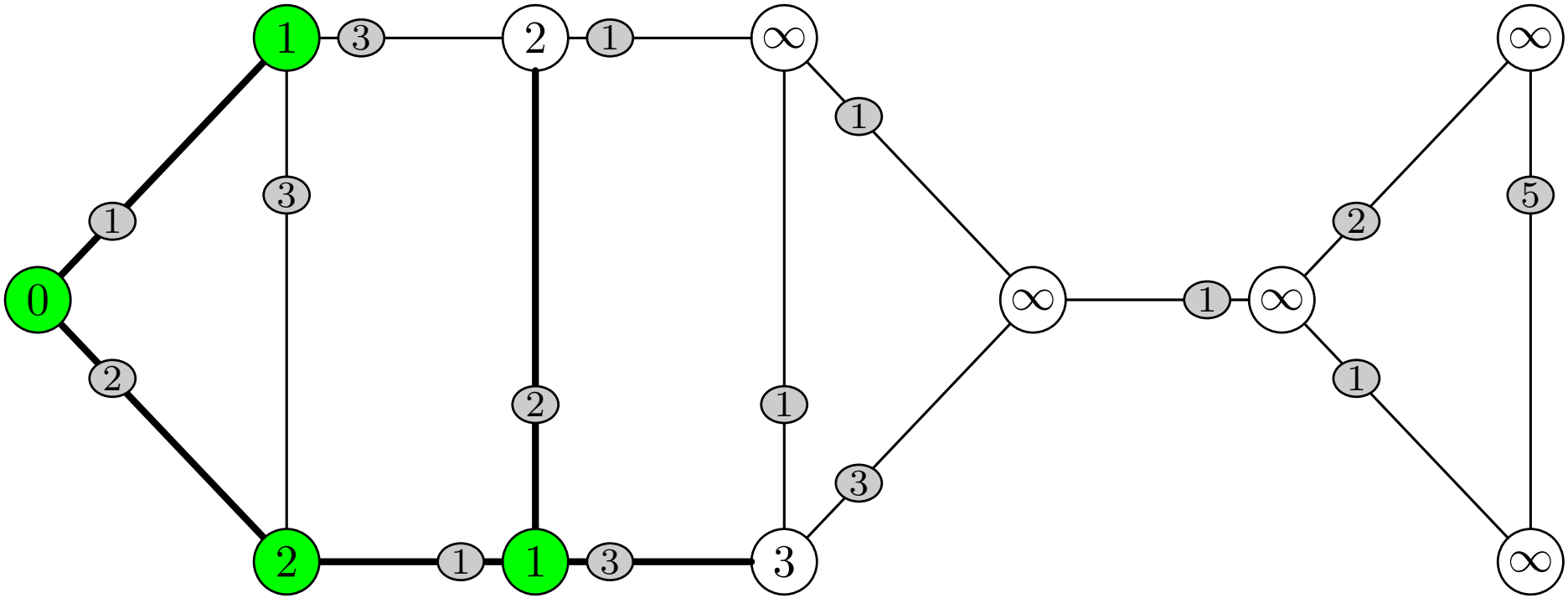
- Beginne mit leerem Baum (nur Wurzel)
- Hänge wiederholt billigste Kante an ohne einen Kreis zu schließen

Der Algorithmus von Prim – Beispiel



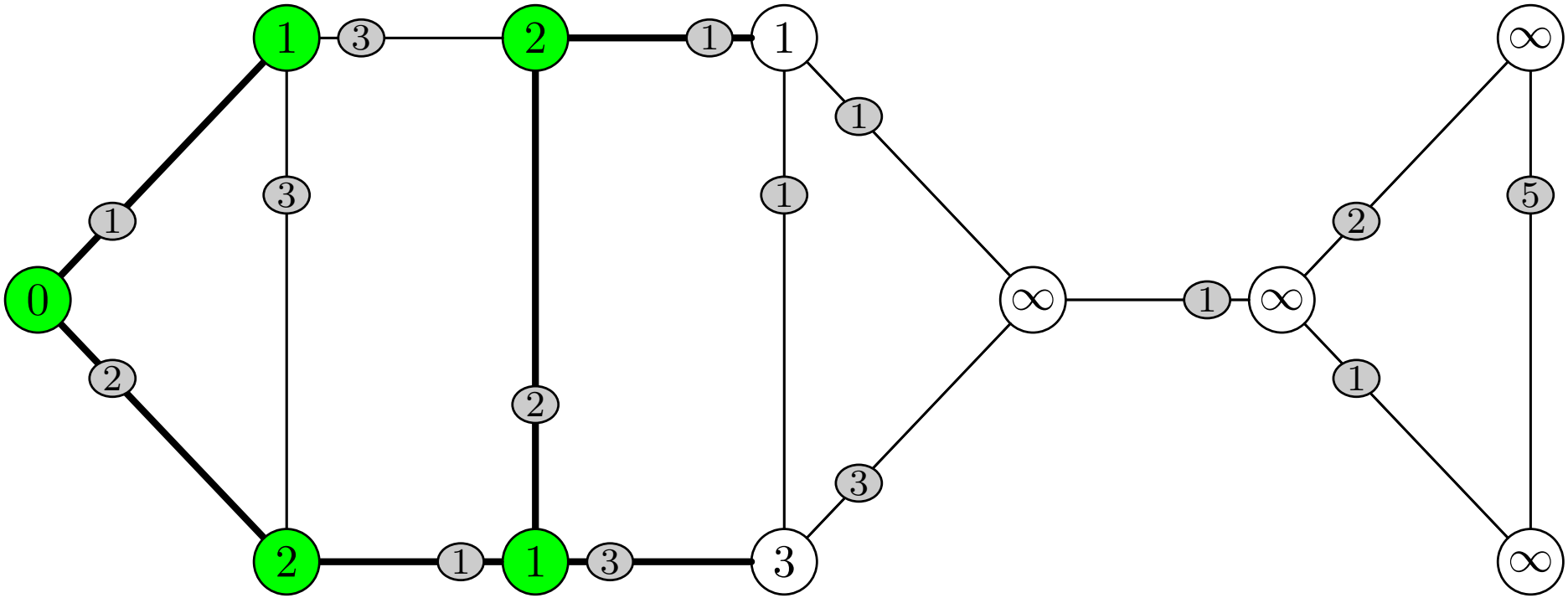
- Beginne mit leerem Baum (nur Wurzel)
- Hänge wiederholt billigste Kante an ohne einen Kreis zu schließen

Der Algorithmus von Prim – Beispiel



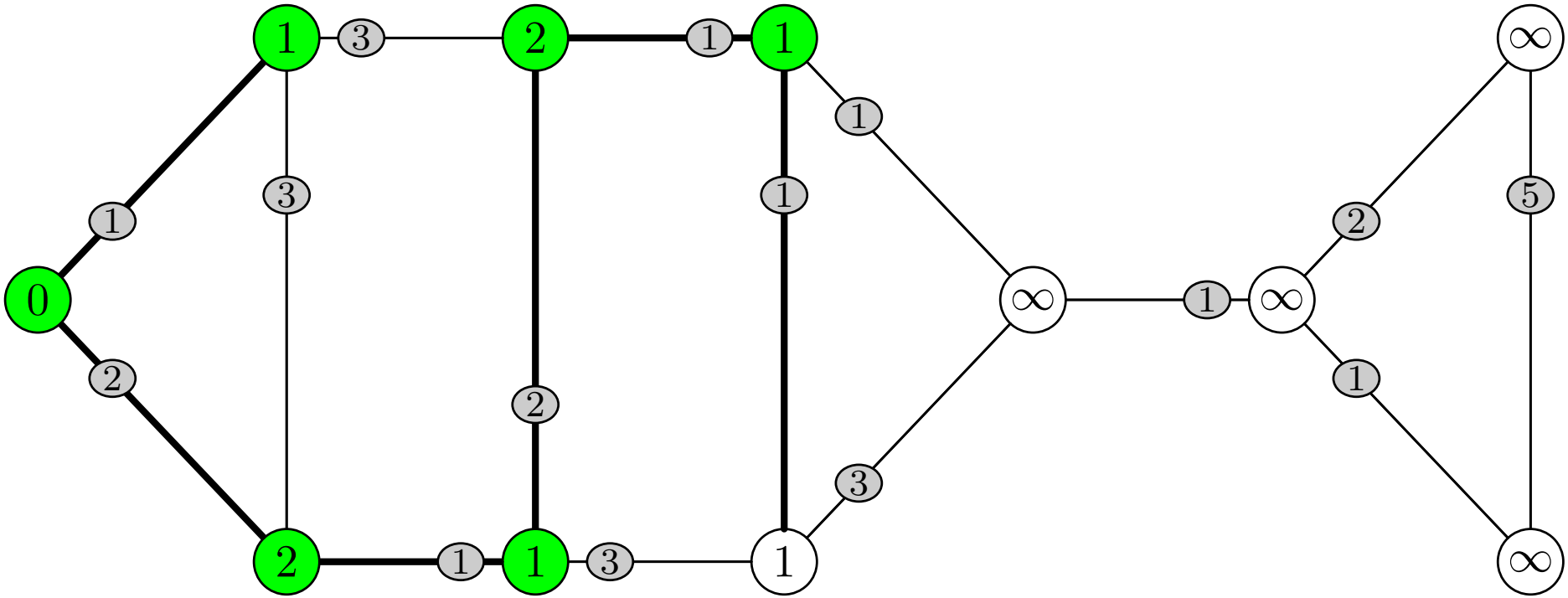
- Beginne mit leerem Baum (nur Wurzel)
- Hänge wiederholt billigste Kante an ohne einen Kreis zu schließen

Der Algorithmus von Prim – Beispiel



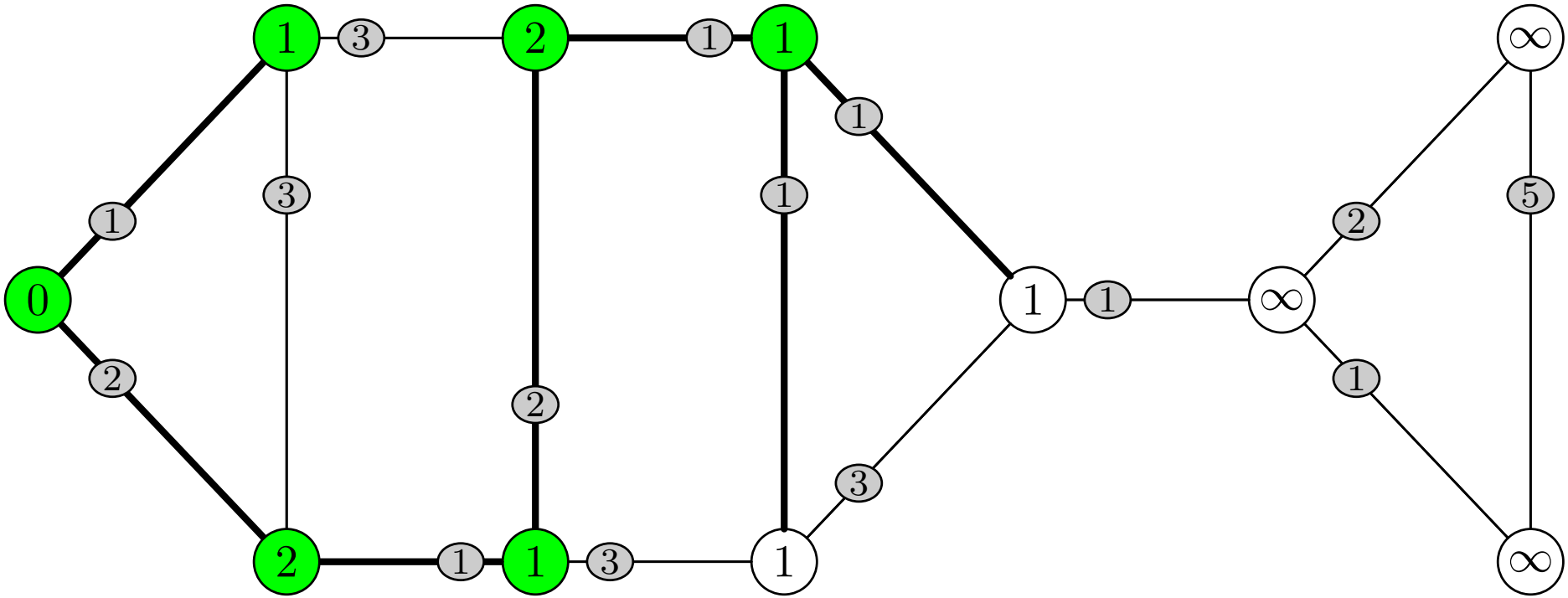
- Beginne mit leerem Baum (nur Wurzel)
- Hänge wiederholt billigste Kante an ohne einen Kreis zu schließen

Der Algorithmus von Prim – Beispiel



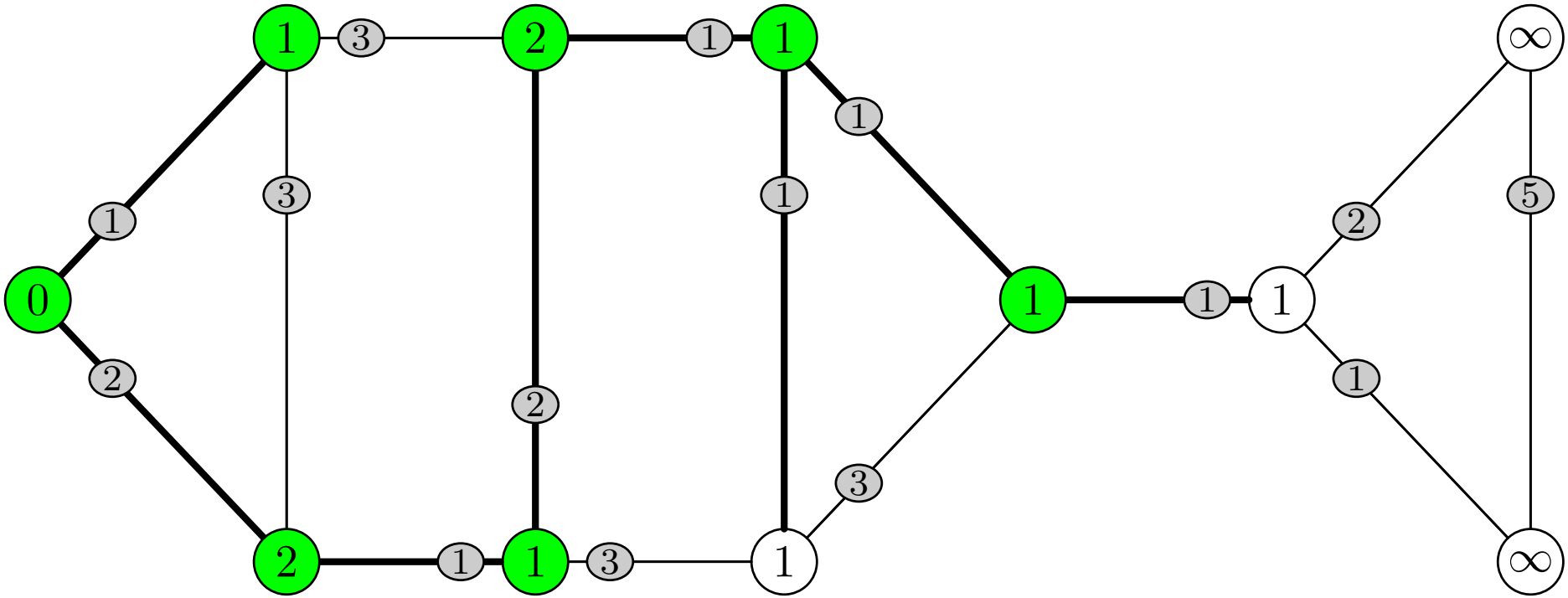
- Beginne mit leerem Baum (nur Wurzel)
- Hänge wiederholt billigste Kante an ohne einen Kreis zu schließen

Der Algorithmus von Prim – Beispiel



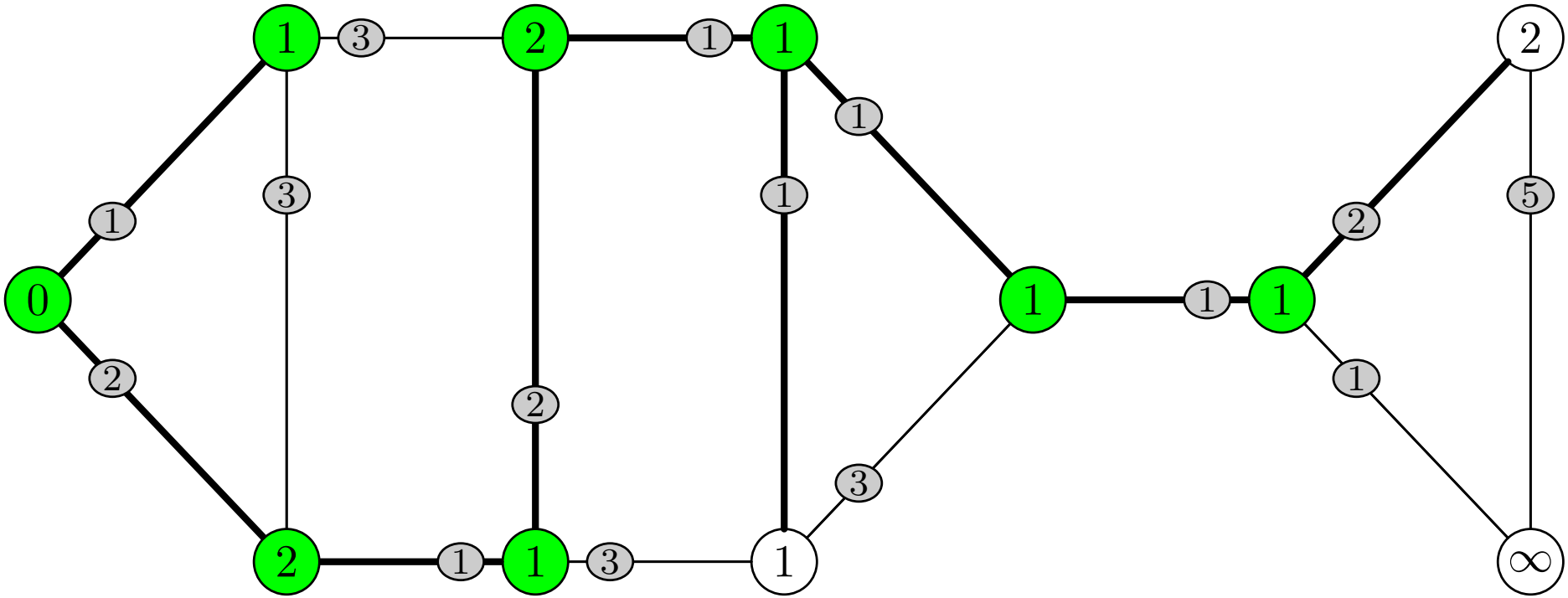
- Beginne mit leerem Baum (nur Wurzel)
- Hänge wiederholt billigste Kante an ohne einen Kreis zu schließen

Der Algorithmus von Prim – Beispiel



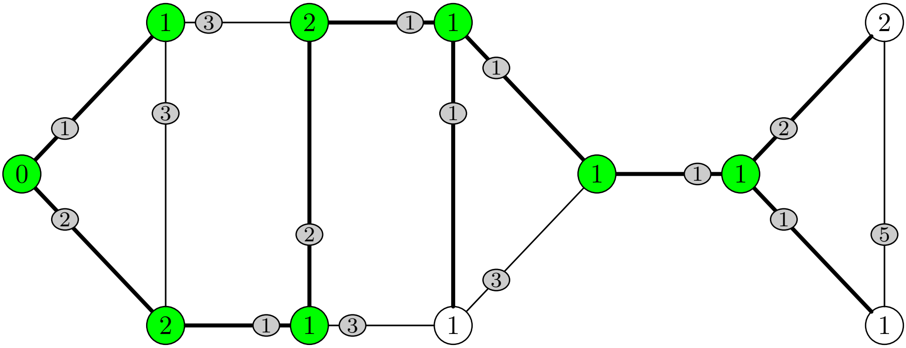
- Beginne mit leerem Baum (nur Wurzel)
- Hänge wiederholt billigste Kante an ohne einen Kreis zu schließen

Der Algorithmus von Prim – Beispiel



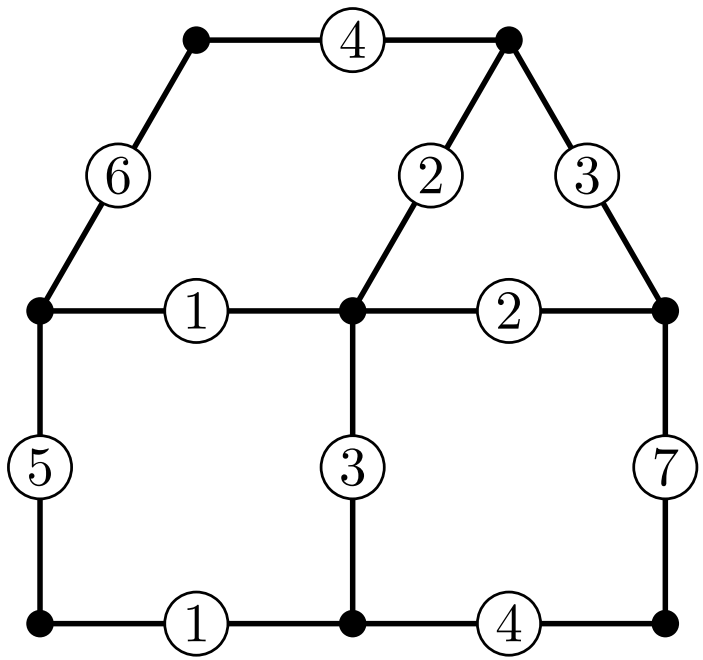
- Beginne mit leerem Baum (nur Wurzel)
- Hänge wiederholt billigste Kante an ohne einen Kreis zu schließen

Der Algorithmus von Prim – Beispiel

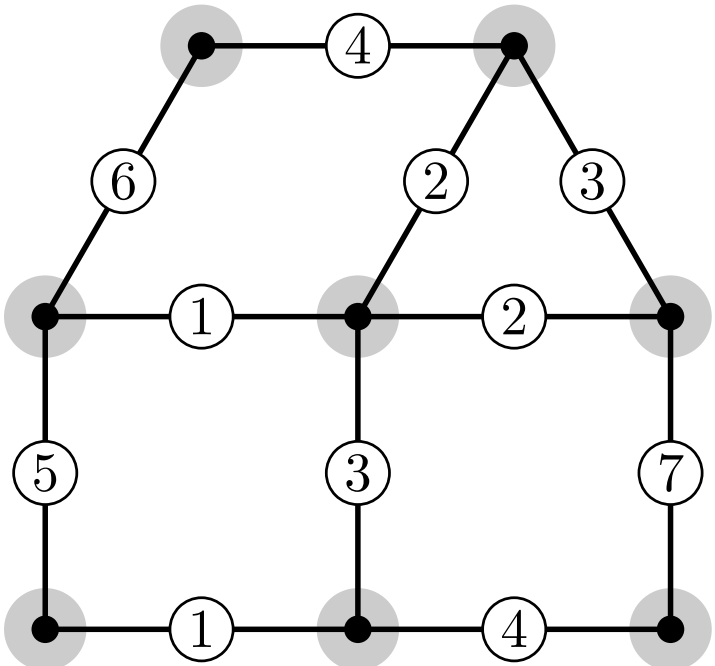


- Beginne mit leerem Baum (nur Wurzel)
- Hänge wiederholt billigste Kante an ohne einen Kreis zu schließen

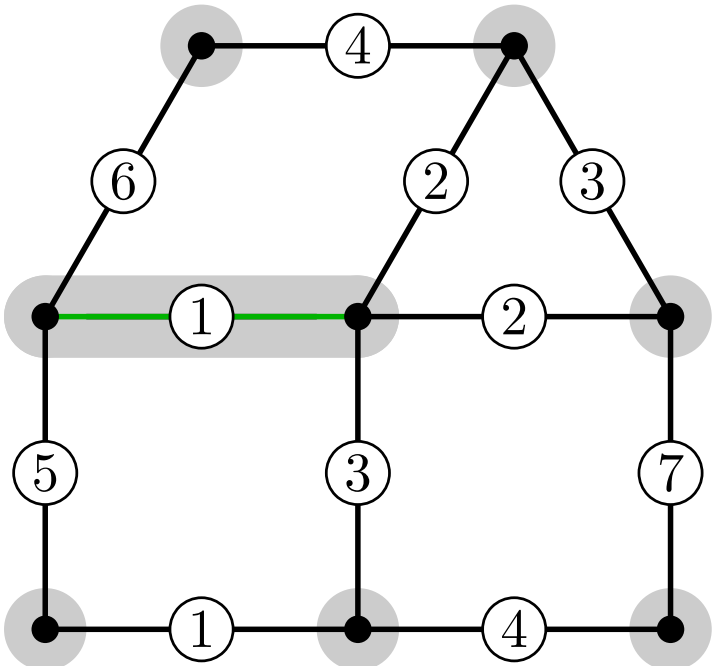
Kruskal: Minimaler Spannbaum



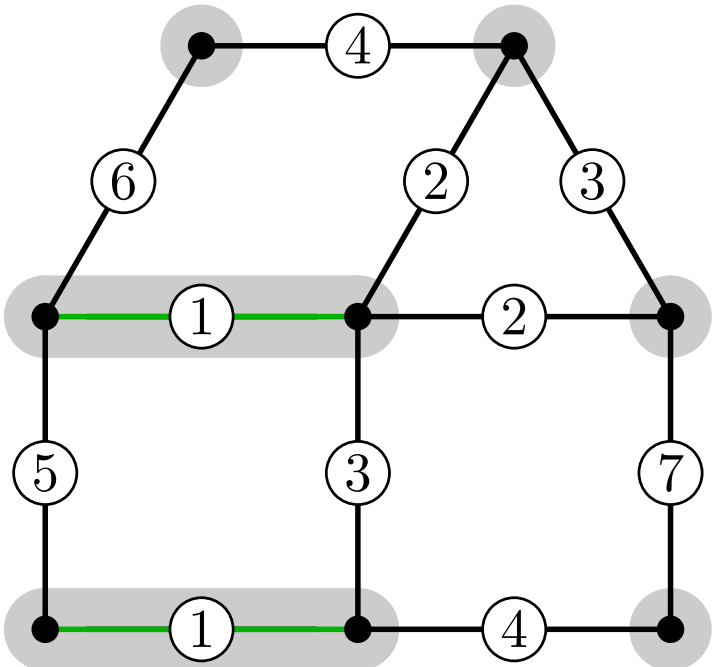
Kruskal: Minimaler Spannbaum



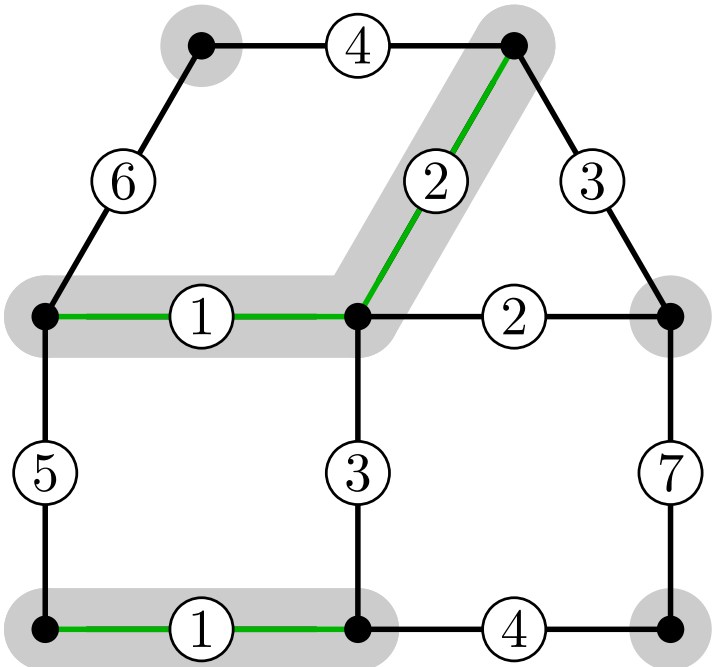
Kruskal: Minimaler Spannbaum



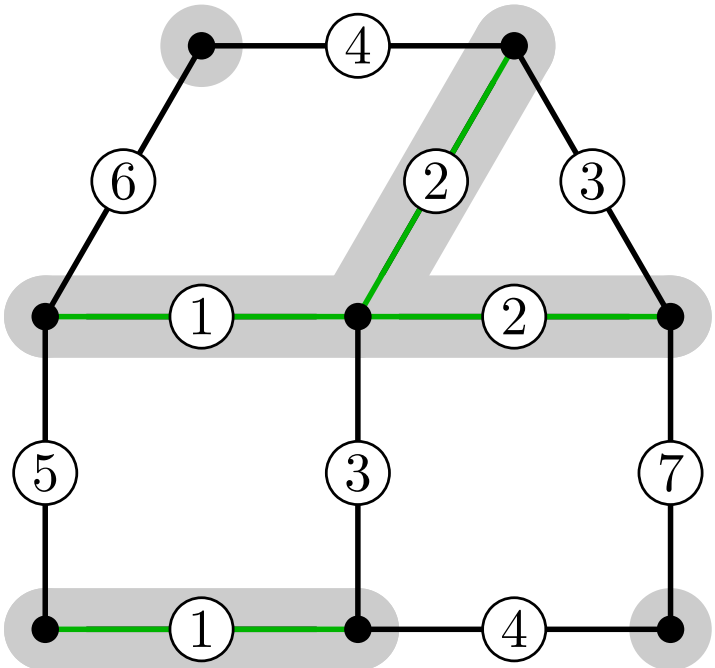
Kruskal: Minimaler Spannbaum



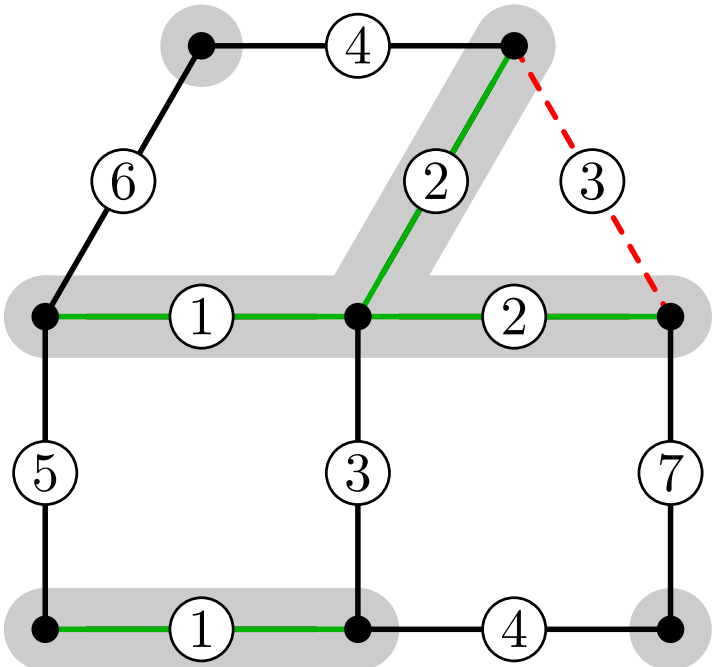
Kruskal: Minimaler Spannbaum



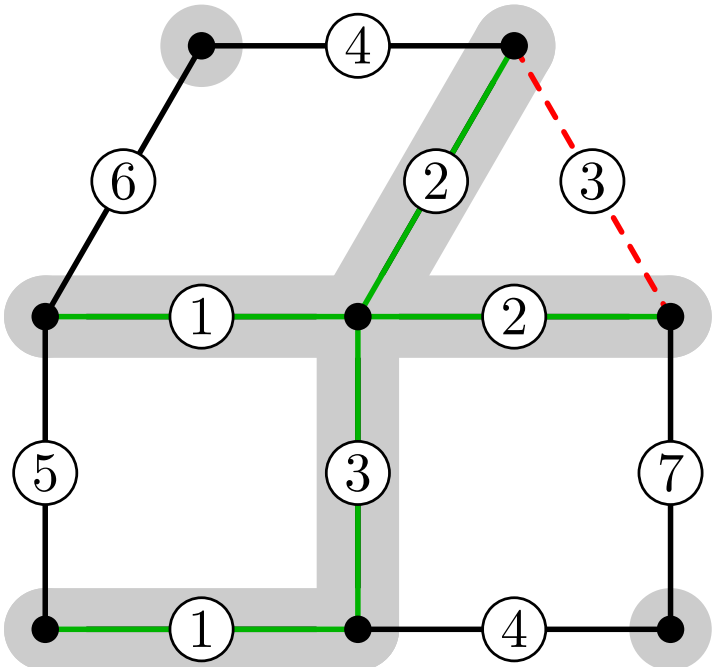
Kruskal: Minimaler Spannbaum



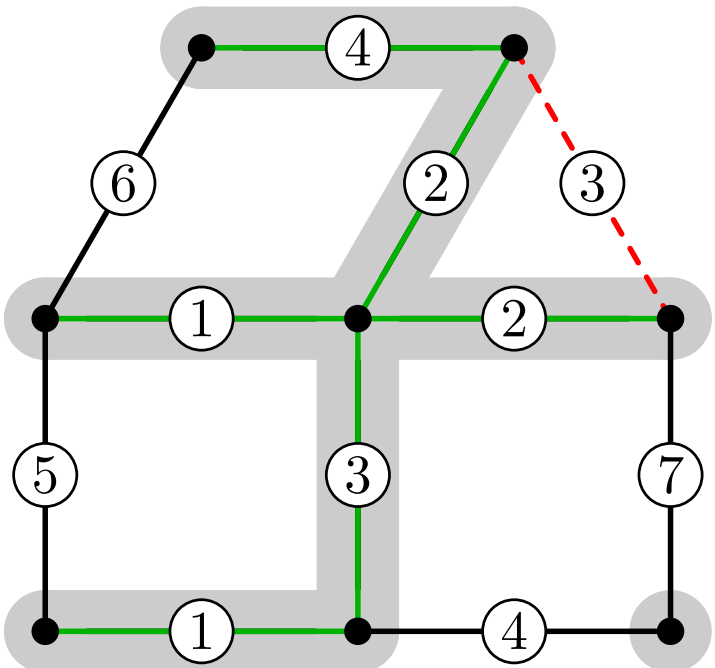
Kruskal: Minimaler Spannbaum



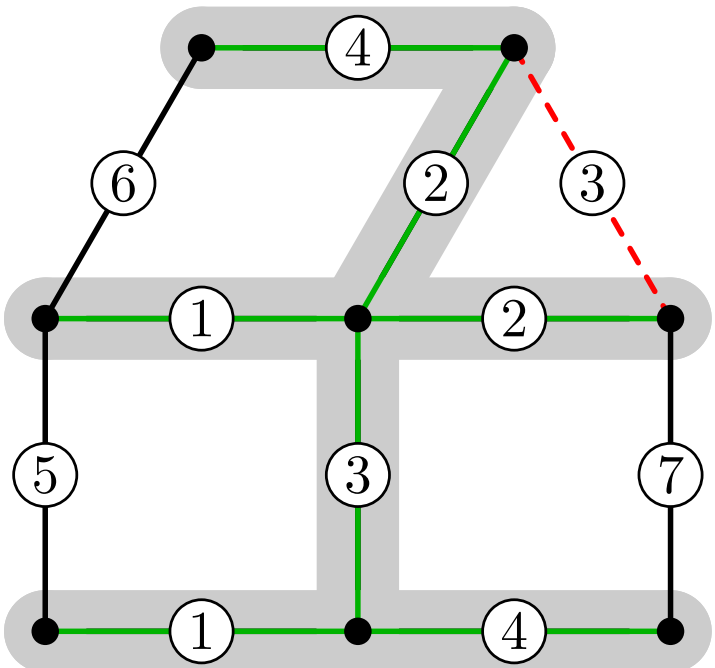
Kruskal: Minimaler Spannbaum



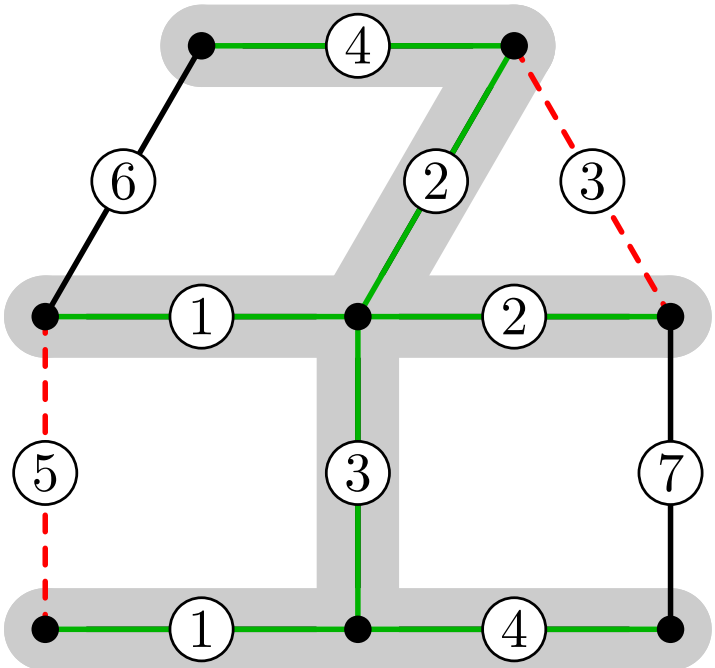
Kruskal: Minimaler Spannbaum



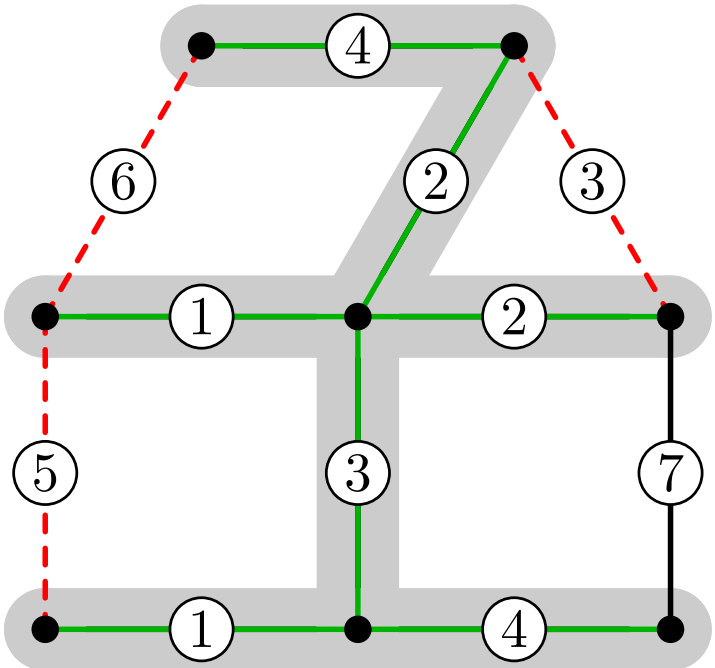
Kruskal: Minimaler Spannbaum



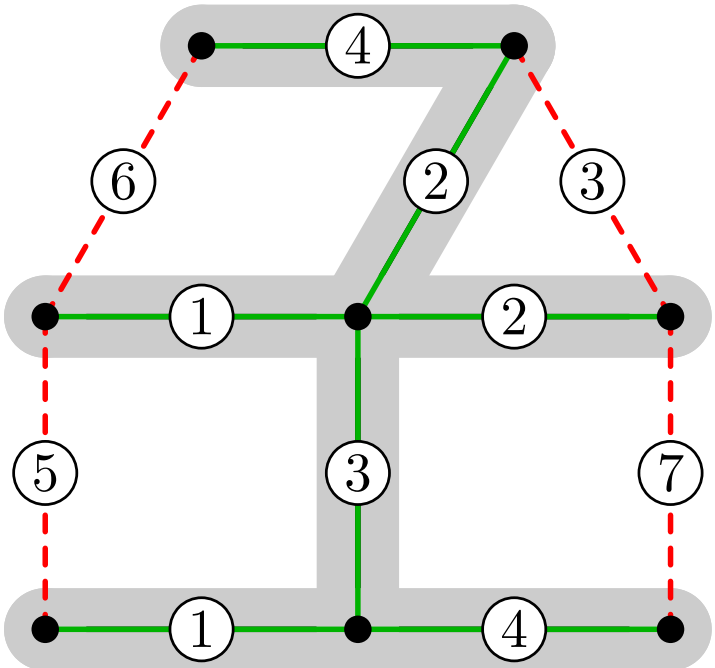
Kruskal: Minimaler Spannbaum



Kruskal: Minimaler Spannbaum



Kruskal: Minimaler Spannbaum



Union-Find: naiv

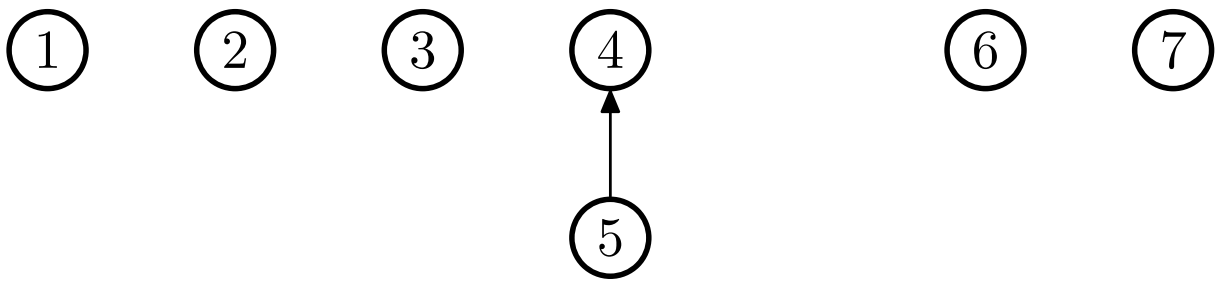
union(a, b): Hänge *find*(a) bei *find*(b) ein

Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$...

Union-Find: naiv

union(a, b): Hänge *find*(a) bei *find*(b) ein

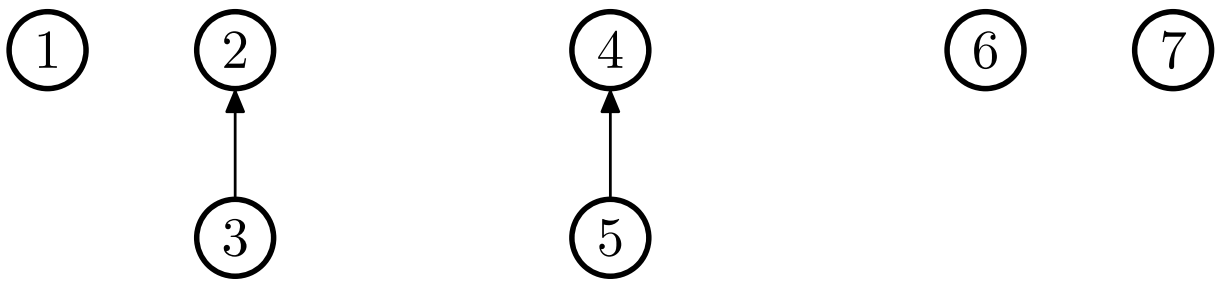
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)\ldots$



Union-Find: naiv

union(a, b): Hänge *find*(a) bei *find*(b) ein

Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)\ldots$



Union-Find: naiv

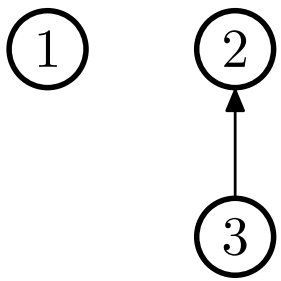
union(a, b): Hänge *find*(a) bei *find*(b) ein

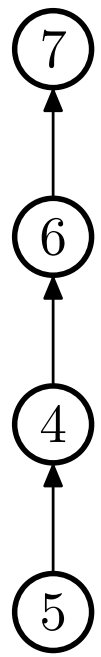
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)\ldots$

Union-Find: naiv

union(a, b): Hänge *find*(a) bei *find*(b) ein

Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)\ldots$





Union-Find: union by rank

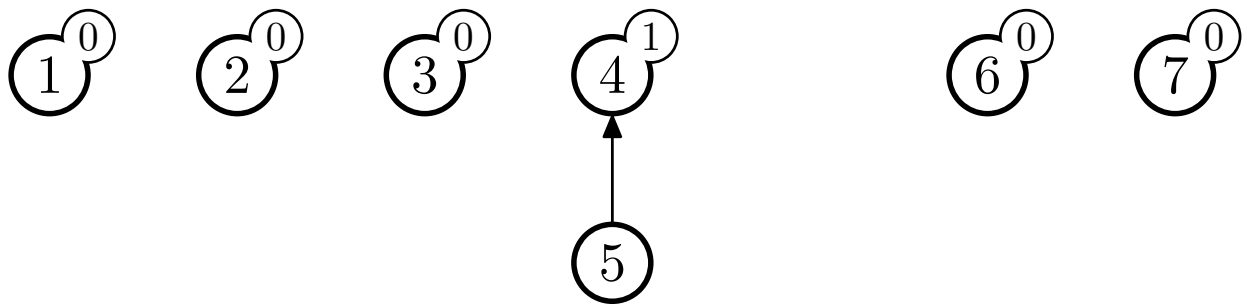
union(a, b): Verwende die ranghöhere Wurzel

Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$, *union*$(3, 4)$

Union-Find: union by rank

union(a, b): Verwende die ranghöhere Wurzel

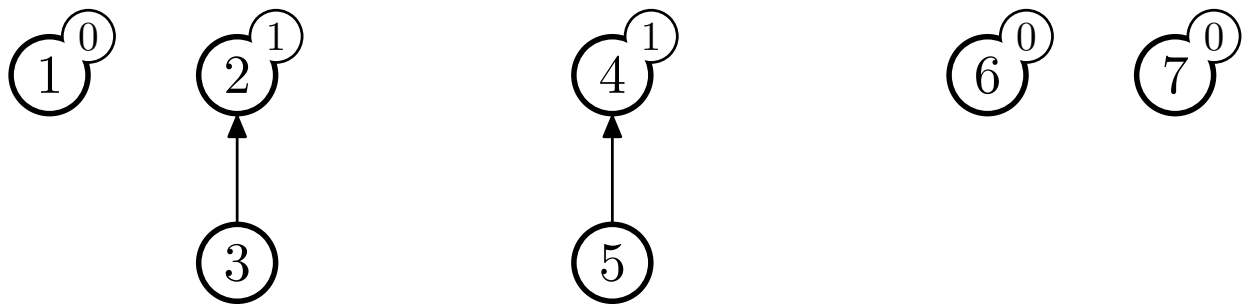
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$, *union*$(3, 4)$



Union-Find: union by rank

union(a, b): Verwende die ranghöhere Wurzel

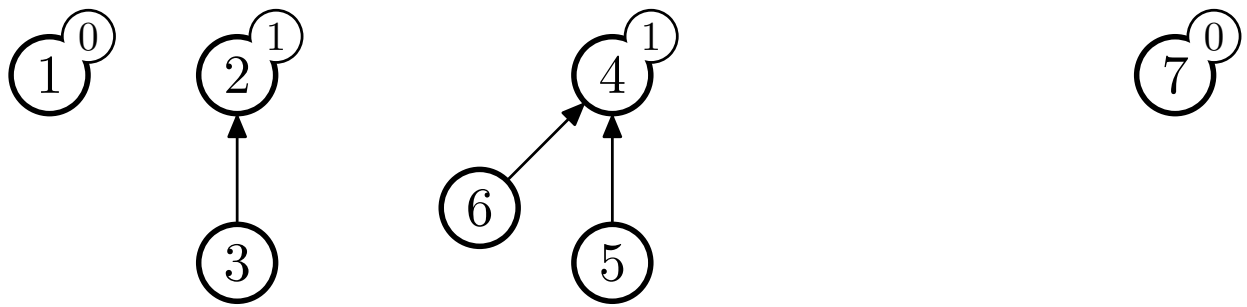
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$, *union*$(3, 4)$



Union-Find: union by rank

union(a, b): Verwende die ranghöhere Wurzel

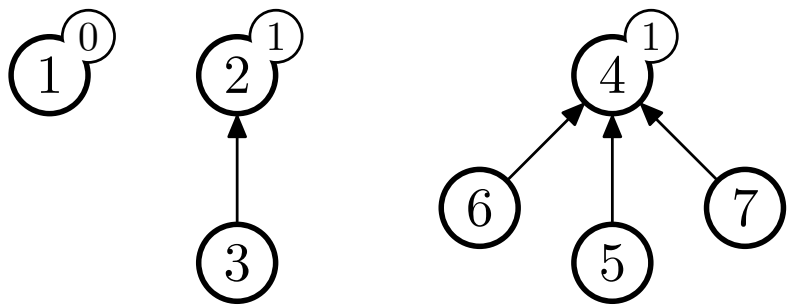
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$, *union*$(3, 4)$



Union-Find: union by rank

union(a, b): Verwende die ranghöhere Wurzel

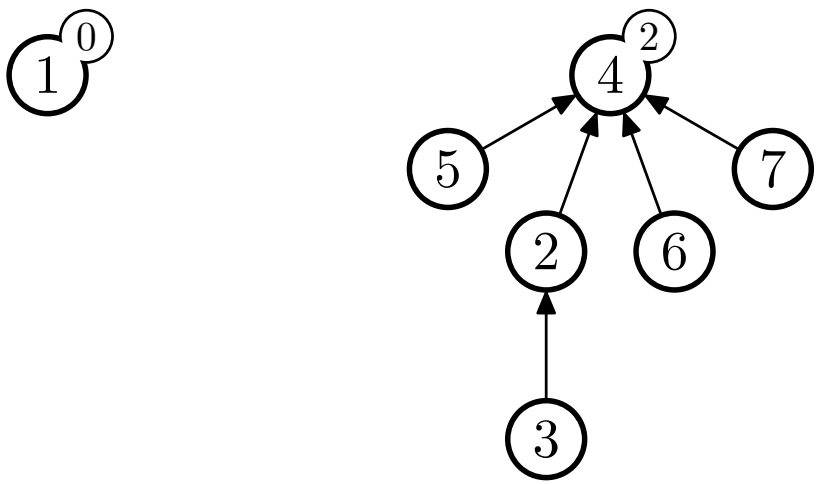
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$, *union*$(3, 4)$



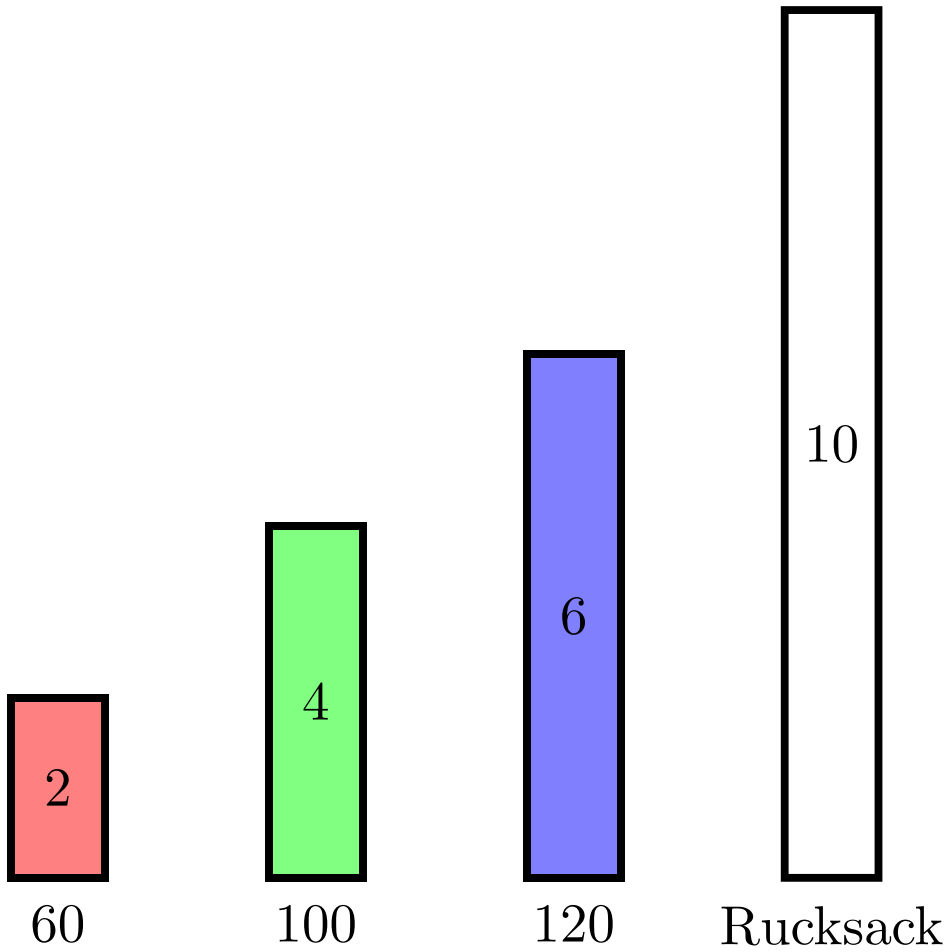
Union-Find: union by rank

union(a, b): Verwende die ranghöhere Wurzel

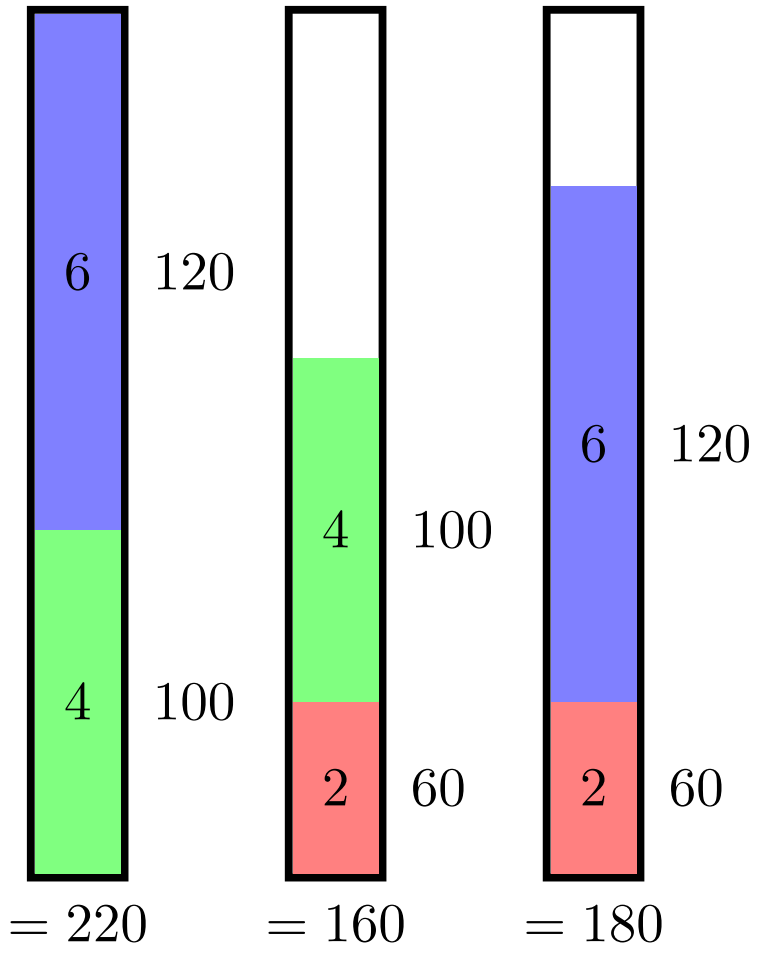
Beispiel: *union*$(5, 4)$, *union*$(3, 2)$, *union*$(5, 6)$, *union*$(4, 7)$, *union*$(3, 4)$



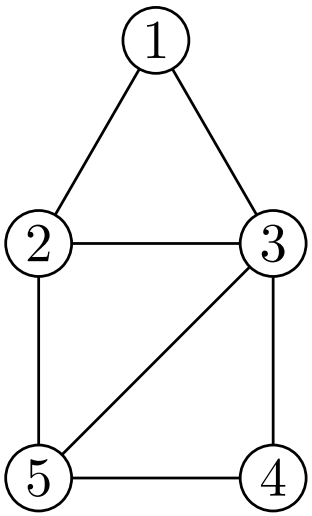
Knapsack



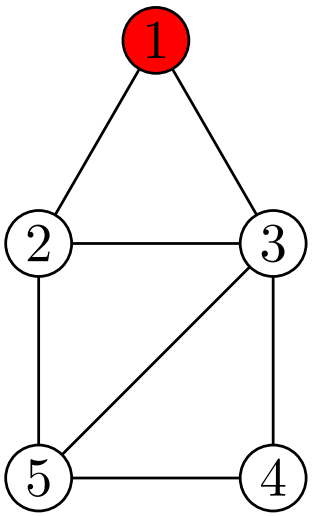
Knapsack



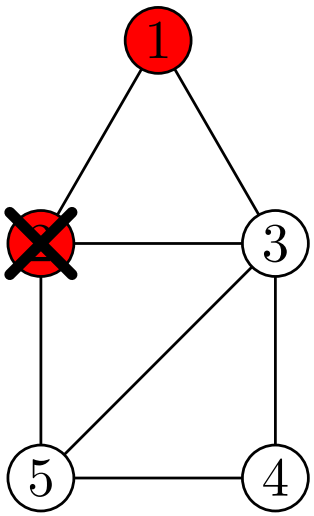
Backtracking



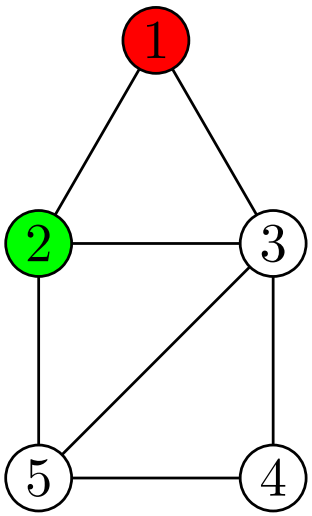
Backtracking



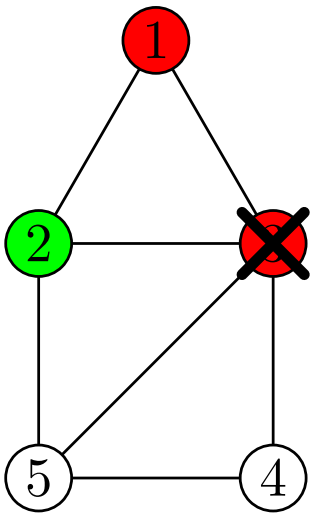
Backtracking



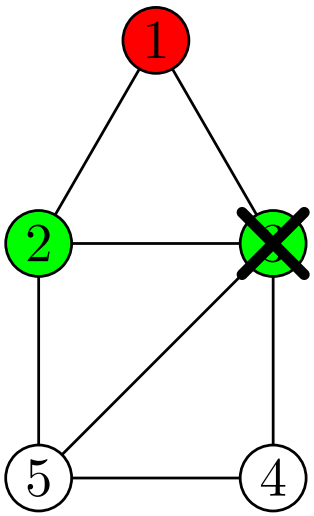
Backtracking



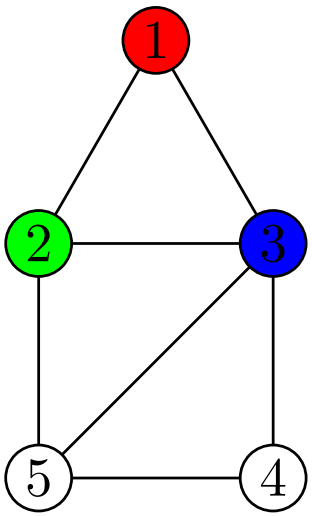
Backtracking



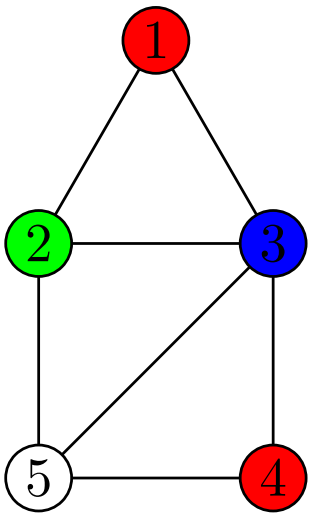
Backtracking



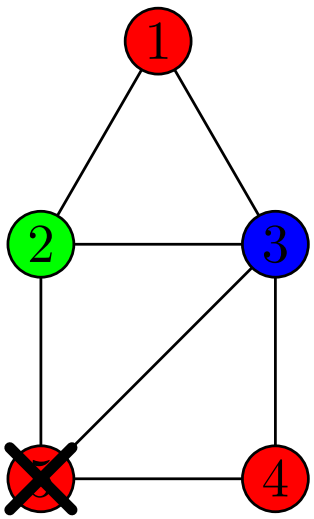
Backtracking



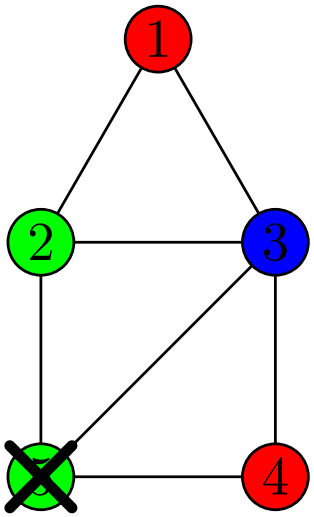
Backtracking



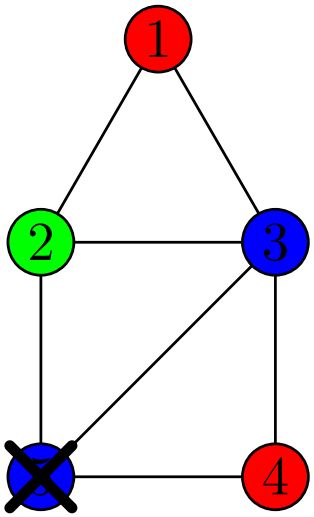
Backtracking



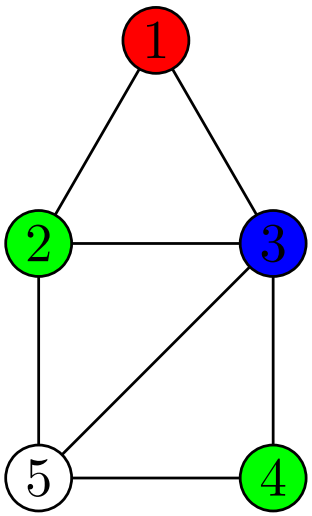
Backtracking



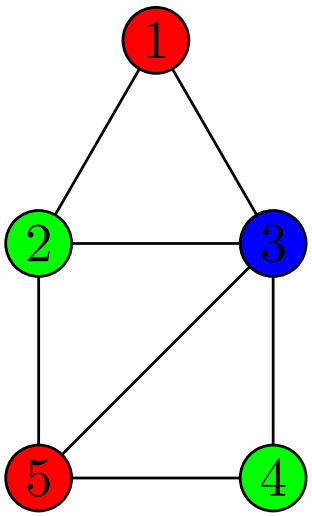
Backtracking



Backtracking

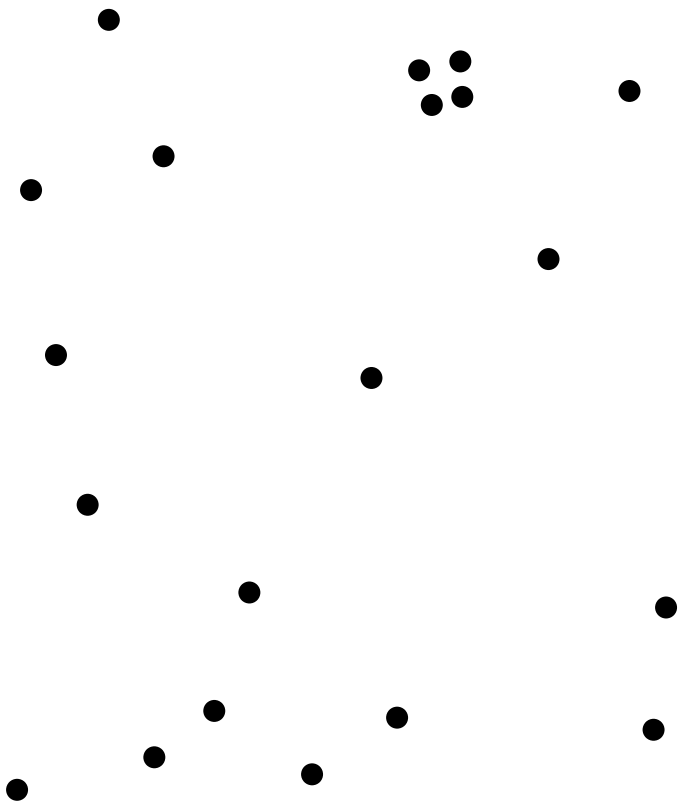


Backtracking



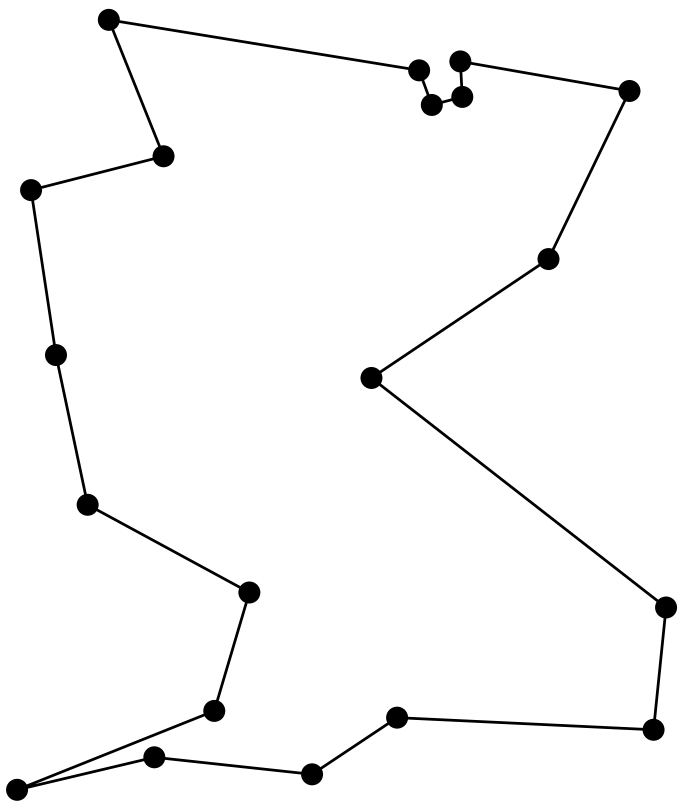
Das Problem des Handlungsreisenden

Ein kleines Beispiel:



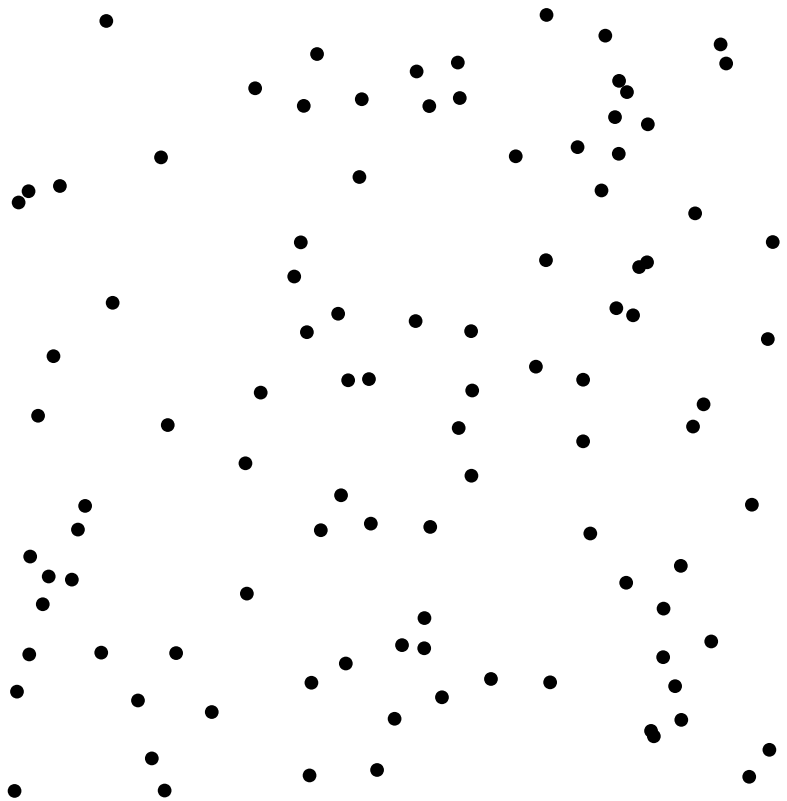
Das Problem des Handlungsreisenden

Ein kleines Beispiel:



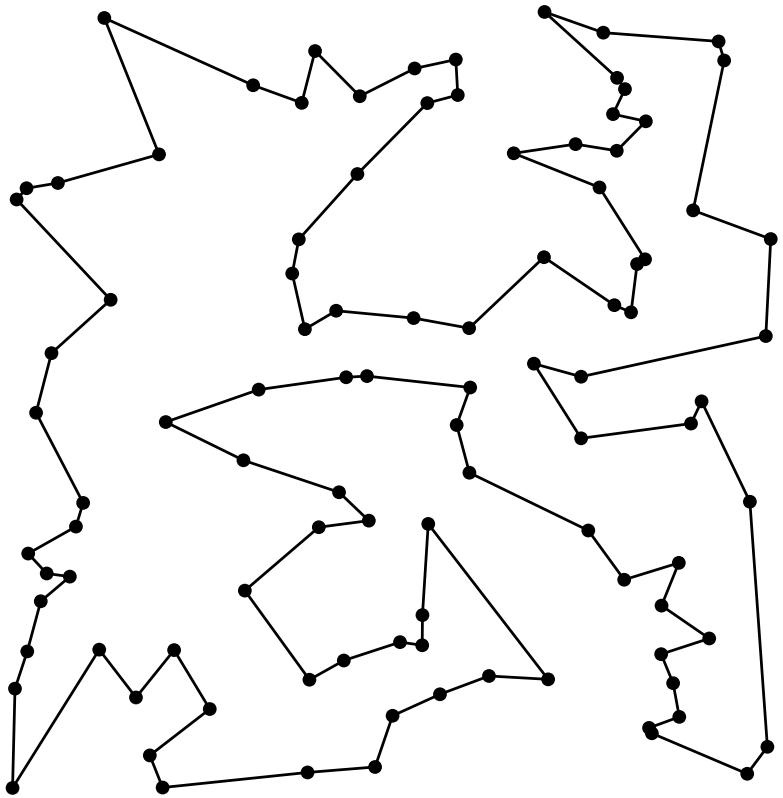
Das Problem des Handlungsreisenden

Ein mittelgroßes Beispiel:

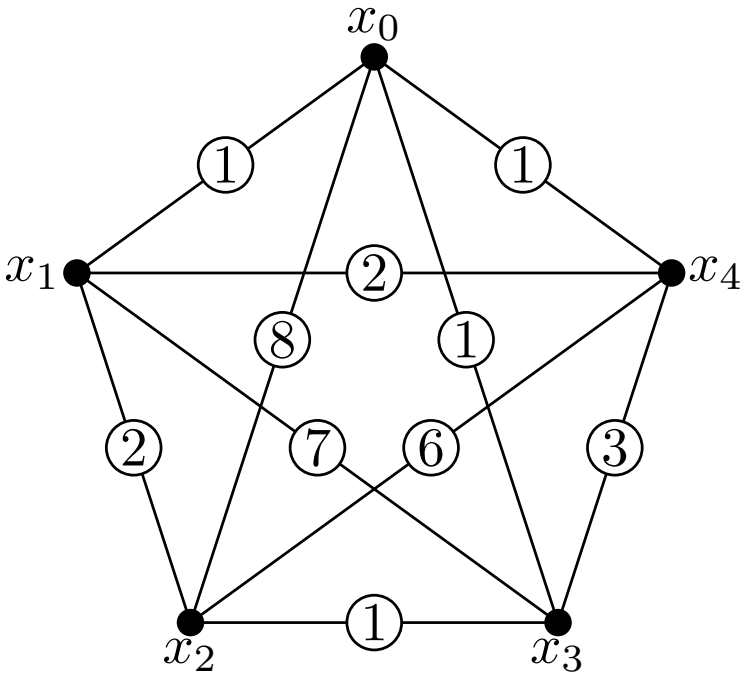


Das Problem des Handlungsreisenden

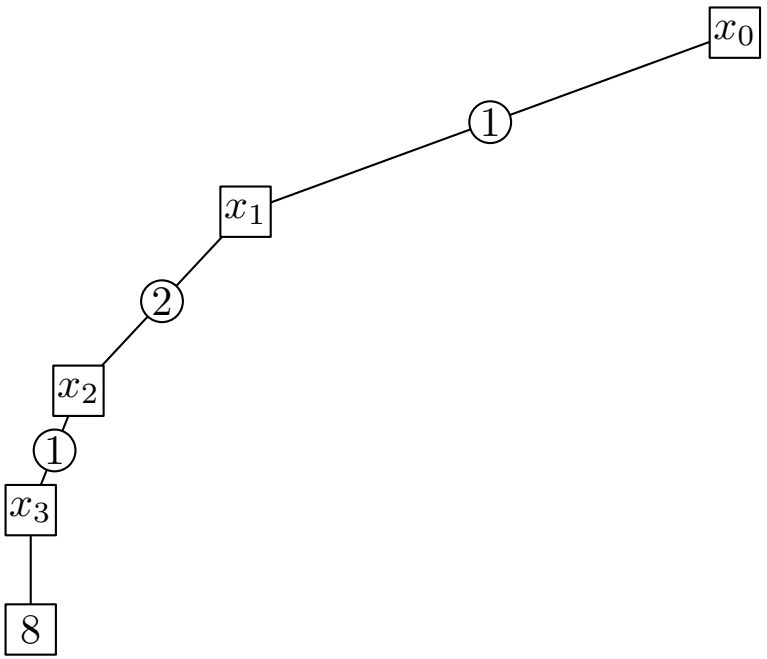
Ein mittelgroßes Beispiel:



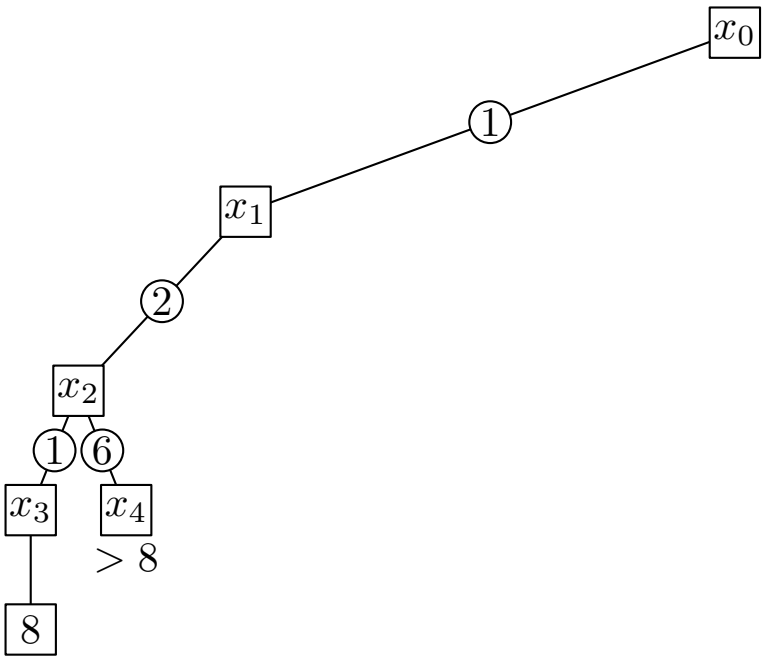
Branch and Bound - Beispiel TSP



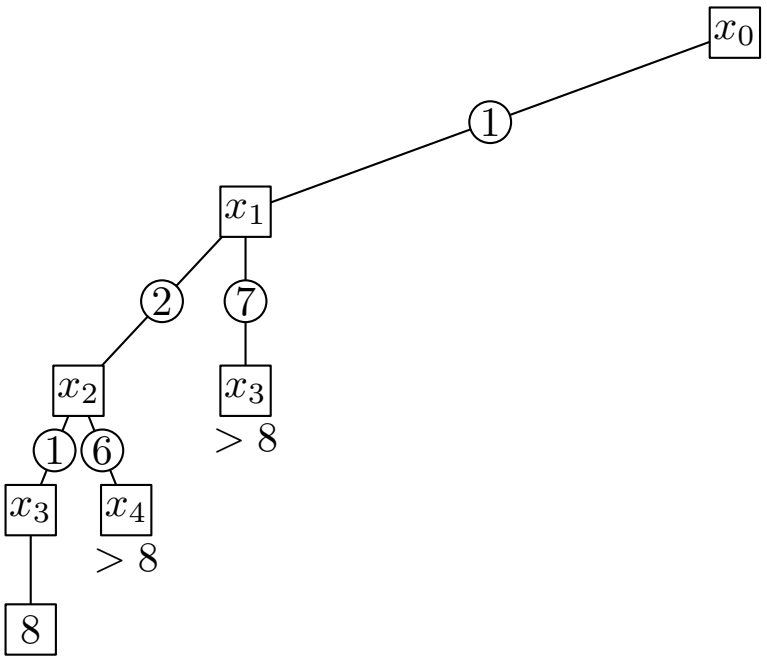
Branch and Bound - Beispiel TSP



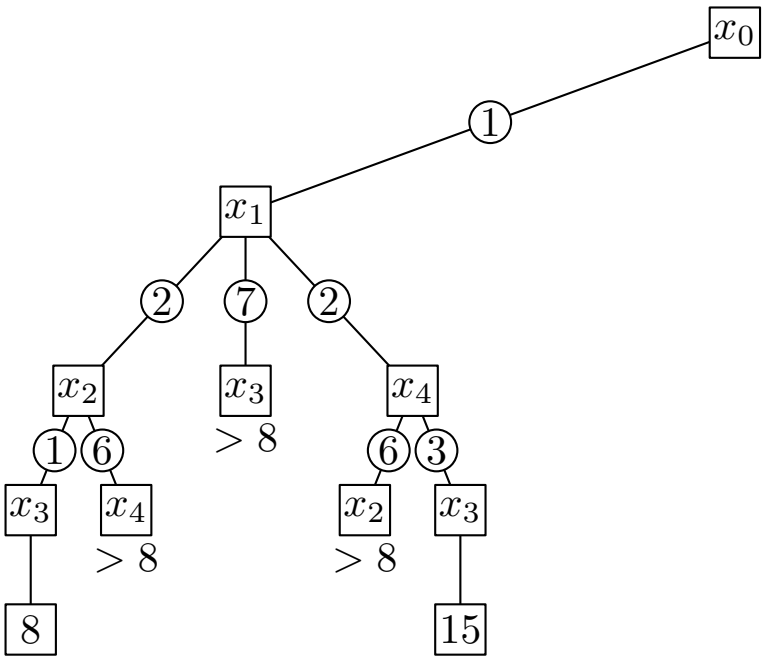
Branch and Bound - Beispiel TSP



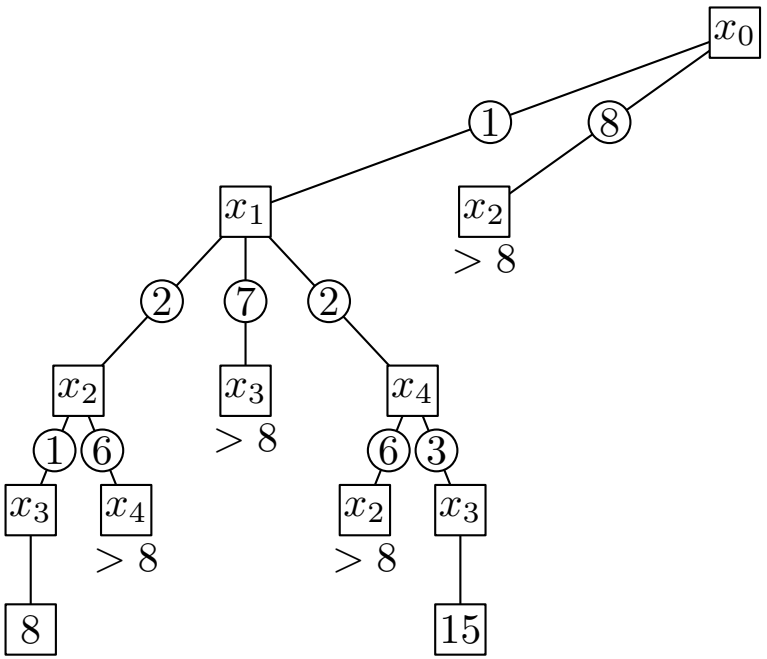
Branch and Bound - Beispiel TSP



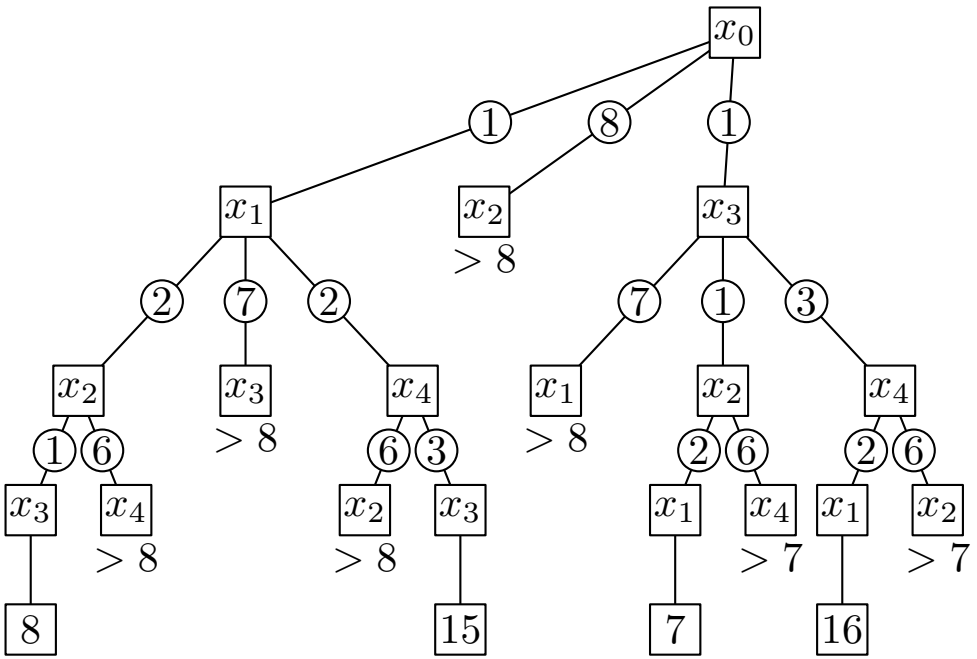
Branch and Bound - Beispiel TSP



Branch and Bound - Beispiel TSP



Branch and Bound - Beispiel TSP



Branch and Bound - Beispiel TSP