Graphentheorie: Bäume

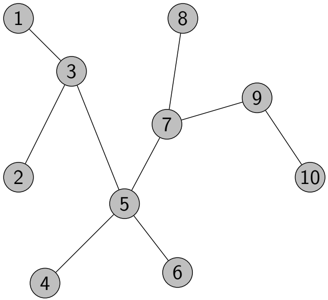
Programmieren und Software-Engineering
Theorie

2. September 2025

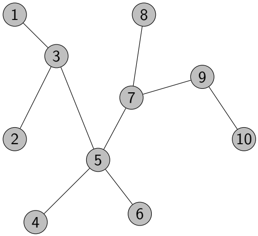
Bäume

Definition (Baum)

Ein zusammenhängender Graph der keine Kreise enthält wird *Baum T* genannt.



Bäume



Ein Baum T kann durch folgende äquivalente Aussagen charakterisiert werden:

- T ist ein Baum
- T ist zusammenhängend und kreisfrei
- Zwei beliebige Knoten von T sind durch genau einen Weg verbunden
- T hat $n-1$ Kanten und ist zusammenhängend
- T hat $n-1$ Kanten und ist kreisfrei
- T ist zusammenhängend und jede Kante ist eine Brücke

Begriffe

- **Grad eines Baumes:** maximaler Grad eines Knoten des Baumes
- **Blatt:** Knoten mit Grad 1 heißen *Blätter* des Baumes. Blätter haben nur einen Nachbarn. Jeder Baum hat mindestens ein Blatt.
- **Ast:** Die Kanten eines Baumes werden auch als *Äste* bezeichnet.
- **Innerer Knoten:** Ein Knoten heißt *innerer Knoten* wenn er kein Blatt ist.

Anmerkung: Bäume haben zahlreiche Anwendungen in der Informatik als Datenstrukturen, z.B. können Objekte in Bäumen so abgespeichert werden, dass sie schnell gesucht (gefunden) werden können!

Wurzelbäume, Arboreszenzen

Definition (Wurzelbaum, Arboreszenz)

Ein gerichteter Graph G heißt *Wurzelbaum* oder *Arboreszenz*, wenn er

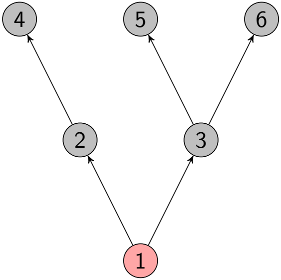
- zusammenhängend ist, und
- es genau einen Knoten $w \in V(G)$ gibt mit $d^{-}(w) = 0$, und
- für alle anderen Knoten $v \in V(G)$ gilt $d^{-}(v) = 1$.

Hierbei bezeichnet $d^{-}(v)$ den *Eingangsgrad* (engl. in-degree), also die Anzahl der zum Knoten führenden Kanten. Der *Ausgangsgrad* $d^{+}(v)$ (engl. out-degree) bezeichnet hingegen die vom Knoten wegführenden Kanten.

Anmerkung: Ein Wurzelbaum ist zusammenhängend und hat genau einen Knoten mit Eingangsgrad 0. Jeder andere Knoten hat den Eingangsgrad 1. Eine Arboreszenz ist ein gerichteter Graph dessen Schatten ein Baum ist, sodass ein Knoten w ausgezeichnet ist (Wurzel), und jede Kante der Arboreszenz von w weggerichtet ist.

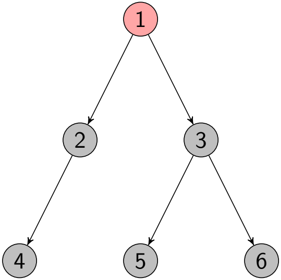
Wurzelbäume, Arboreszenzen

Beispiel: Ursprünglich wurden Wurzelbäume so gezeichnet:



Wurzelbäume, Arboreszenzen

Beispiel: Heute wird die Wurzel immer oben gezeichnet.



Traversierung von Bäumen

Wir betrachten im Folgenden Algorithmen zur Traversierung von Bäumen. Diese können jedoch auch zur Traversierung von Graphen im Allgemeinen verwendet werden.

Ziel: Wir wollen die Knoten eines Baumes systematisch durchlaufen, mit dem Ziel einen bestimmten Knoten zu finden.

- **Breitensuche (Breadth-First-Search (BFS)):** In jedem Schritt werden zunächst alle Nachbarknoten eines Knoten besucht, bevor von dort aus weitere Pfade gebildet werden.
- **Tiefensuche (Depth-First-Search (DFS)):** Ein Pfad wird vollständig in die Tiefe durchlaufen, bevor etwaige Abzweigungen verwendet werden.

Traversierung von Bäumen

Zur Beschreibung der Suchverfahren werden folgende Begriffe benötigt:

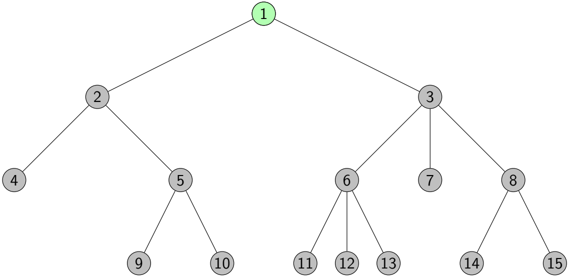
- Ein Knoten wird **entdeckt**, wenn er das erste Mal besucht wird.
- Ein Knoten wird **fertiggestellt/abgeschlossen**, wenn er das letzte Mal verlassen wird.

Für manche Anwendungen ist es wichtig festzuhalten, wann ein Knoten *entdeckt*, bzw. abgeschlossen wurde. Dazu führen wir einen Zähler τ mit, der in jedem Schritt um 1 erhöht wird.

- Wird ein Knoten v das erste mal besucht ("entdeckt"), so setzen wir $\tau_d(v) = \tau{++}$
- Wird ein Knoten v das letzte mal verlassen ("abgeschlossen"), so setzen wir $\tau_f(v) = \tau{++}$

Breitensuche

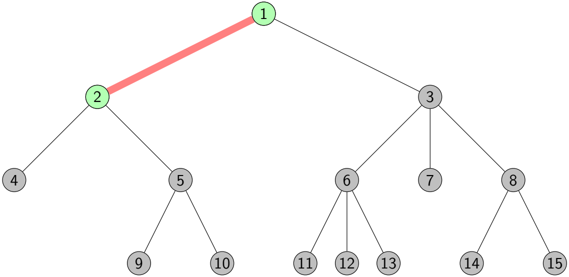
Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.



$\tau_d(1) = 1,$

Breitensuche

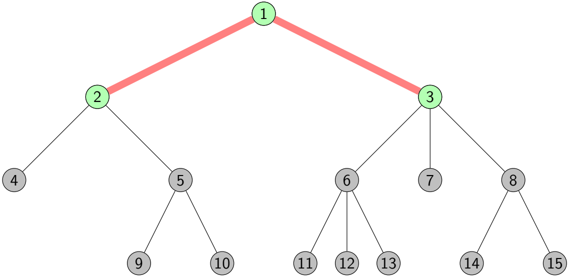
Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.



$\tau_d(1) = 1$, $\tau_d(2) = 2$,

Breitensuche

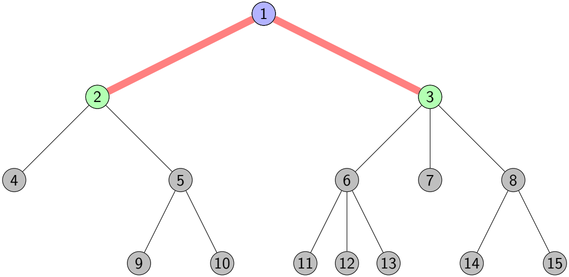
Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.



$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$,

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

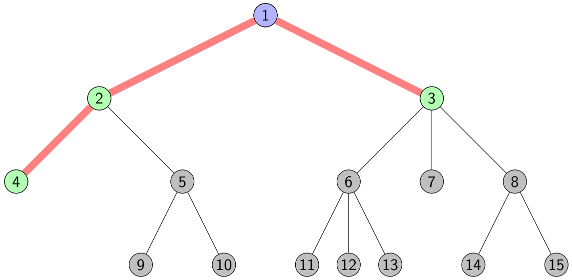


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$,

$\tau_f(1) = 4$,

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

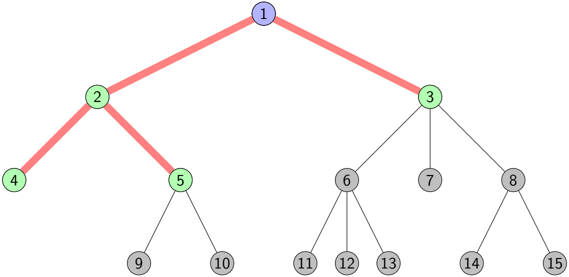


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$,

$\tau_f(1) = 4$,

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

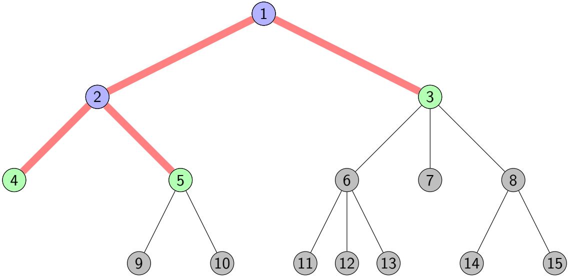


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$,

$\tau_f(1) = 4$,

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

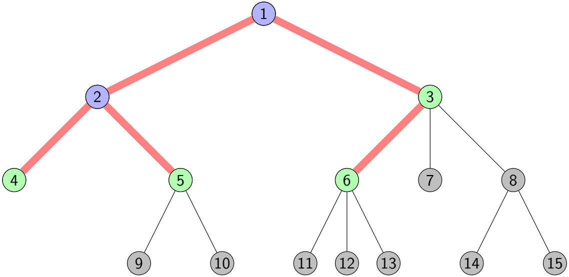


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$,

$\tau_f(1) = 4$, $\tau_f(2) = 7$,

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

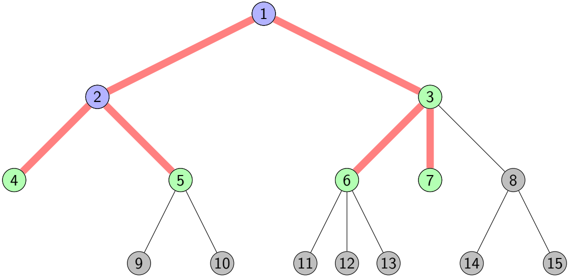


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$,

$\tau_f(1) = 4$, $\tau_f(2) = 7$,

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

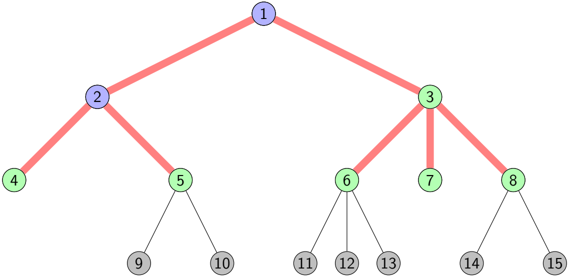


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$,

$\tau_f(1) = 4$, $\tau_f(2) = 7$,

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

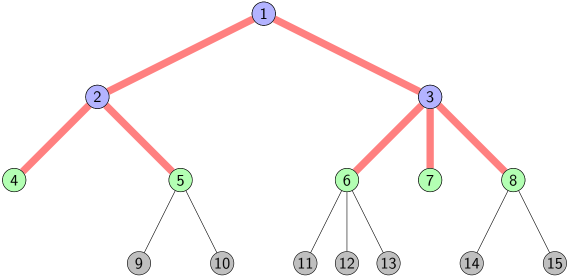


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$,

$\tau_f(1) = 4$, $\tau_f(2) = 7$,

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

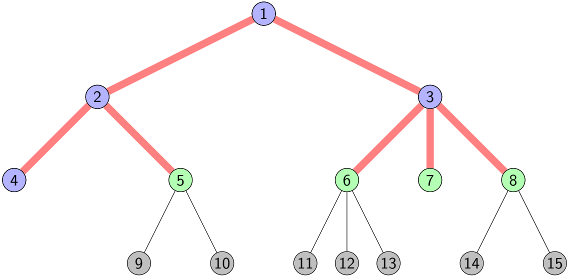


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$,

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$,

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

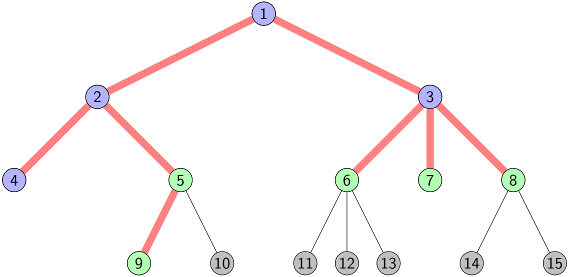


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$,

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$,

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

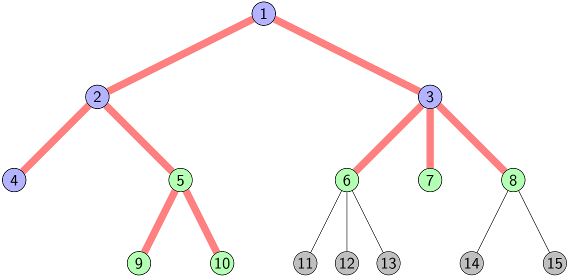


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$,

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$,

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden (“entdeckt”) wurde.

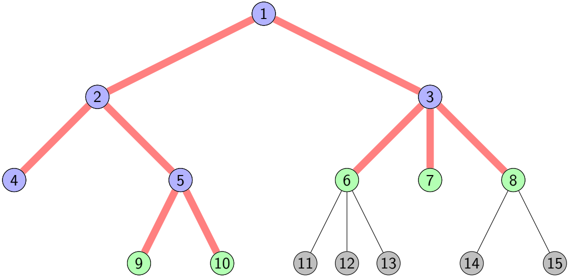


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14$,

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$,

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

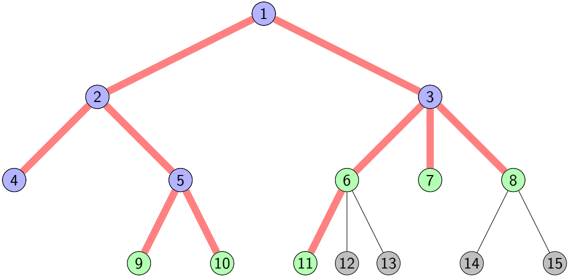


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14$,

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$, $\tau_f(5) = 15$,

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

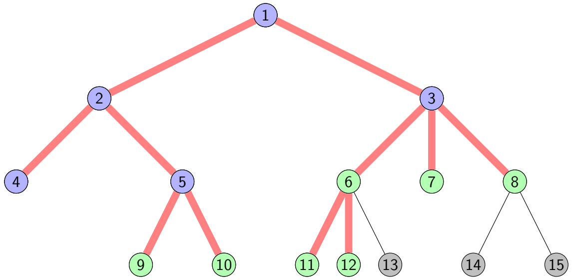


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \ldots$

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

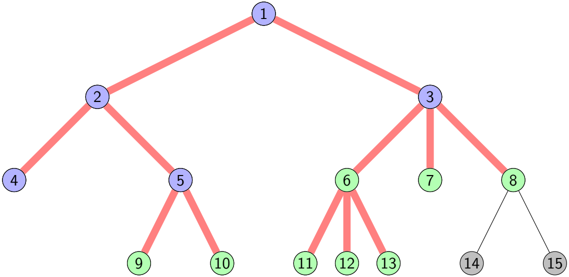


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14$, $\ldots$

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$, $\tau_f(5) = 15$, $\ldots$

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

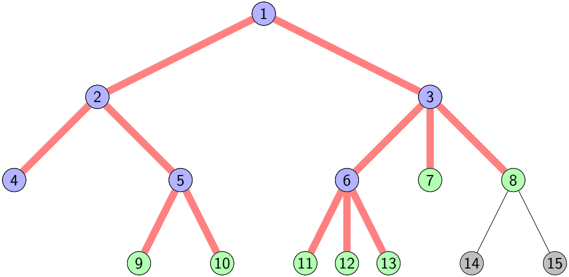


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14$, $\ldots$

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$, $\tau_f(5) = 15$, $\ldots$

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

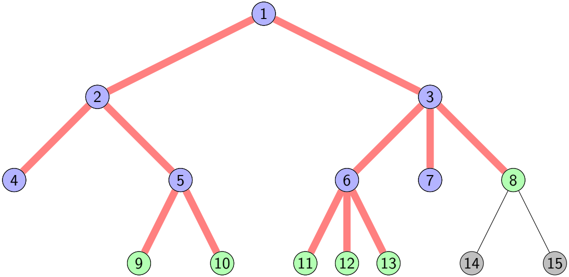


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14, \ldots$

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$, $\tau_f(5) = 15, \ldots$

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

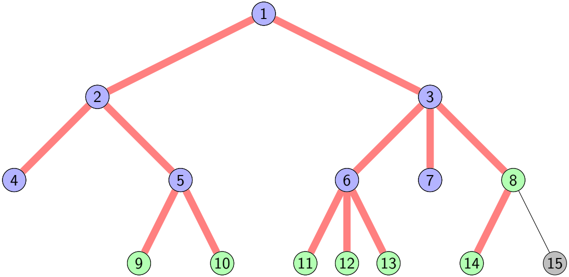


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14$, ...

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$, $\tau_f(5) = 15$, ...

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

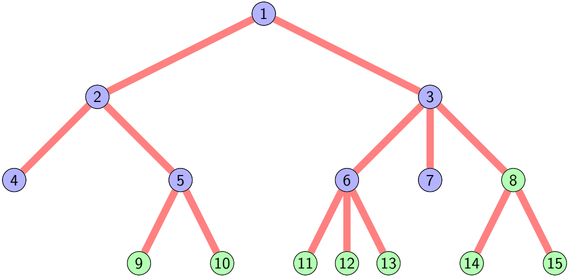


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14$, ...

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$, $\tau_f(5) = 15$, ...

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

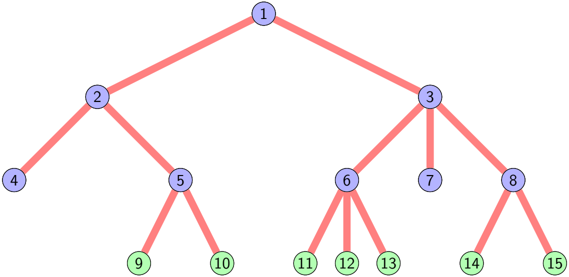


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14$, ...

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$, $\tau_f(5) = 15$, ...

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

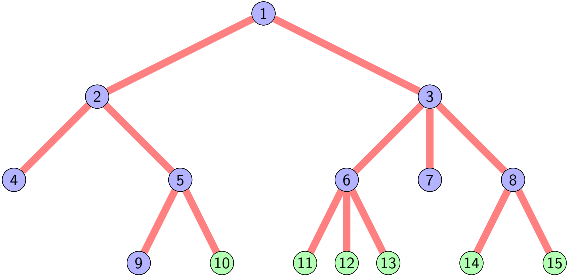


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \ldots$

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

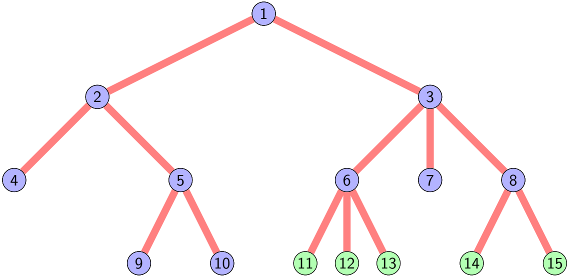


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \ldots$

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

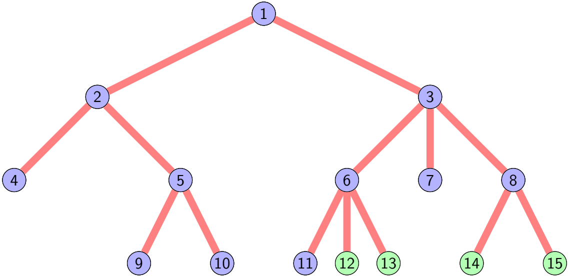


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14$, $\ldots$

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$, $\tau_f(5) = 15$, $\ldots$

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

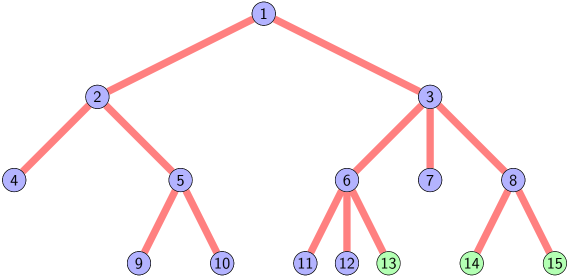


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14$, $\ldots$

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$, $\tau_f(5) = 15$, $\ldots$

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

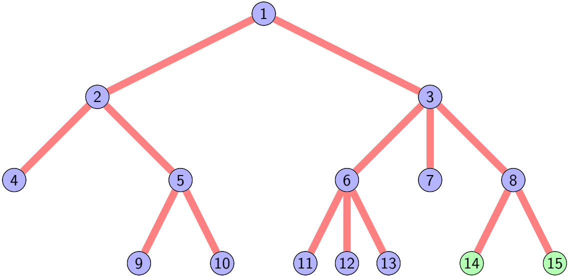


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14$, ...

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$, $\tau_f(5) = 15$, ...

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

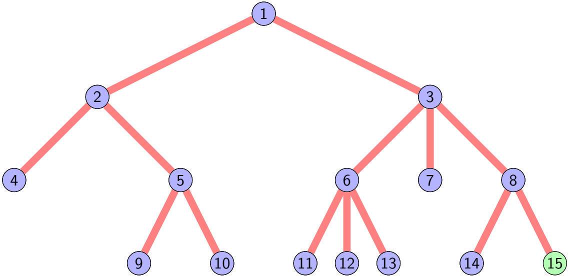


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14, \ldots$

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$, $\tau_f(5) = 15, \ldots$

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

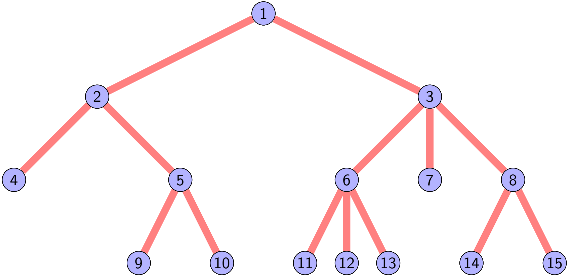


$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14$, $\ldots$

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$, $\tau_f(5) = 15$, $\ldots$

Breitensuche

Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.



$\tau_d(1) = 1$, $\tau_d(2) = 2$, $\tau_d(3) = 3$, $\tau_d(4) = 5$, $\tau_d(5) = 6$, $\tau_d(6) = 8$, $\tau_d(7) = 9$, $\tau_d(8) = 10$, $\tau_d(9) = 13$, $\tau_d(10) = 14, \ldots$

$\tau_f(1) = 4$, $\tau_f(2) = 7$, $\tau_f(3) = 11$, $\tau_f(4) = 12$, $\tau_f(5) = 15, \ldots$

Breitensuche: Datenstruktur

In nahezu allen Programmiersprachen existiert eine Datenstruktur namens **Queue** (Warteschlange). Elemente können hinzugefügt werden ("hinten anstellen"), und werden geordnet abgespeichert. Das Element das als erstes hinzugefügt wurde, kann entnommen werden ("Nächster!")

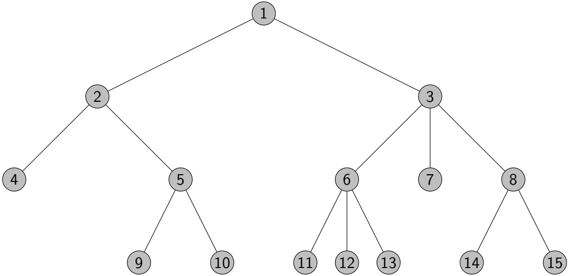
Breitensuche: Algorithmus

1. Füge Startknoten (Wurzel) in Queue (Warteschlange) ein und markiere ihn als entdeckt
2. Entnimm Knoten am Beginn der Queue
   - Markiere Knoten als entdeckt
   - Wenn dies der gesuchte Knoten ist $\Rightarrow$ Fertig!
   - Sonst: füge alle unbesuchten[1] Nachbarn dieses Knotens in die Queue ein
3. Wenn die Warteschlange leer ist wurden alle Knoten bereits besucht $\Rightarrow$ Fertig
4. Gehe zu Schritt (2)

[1] nicht entdeckt, nicht abgeschlossen

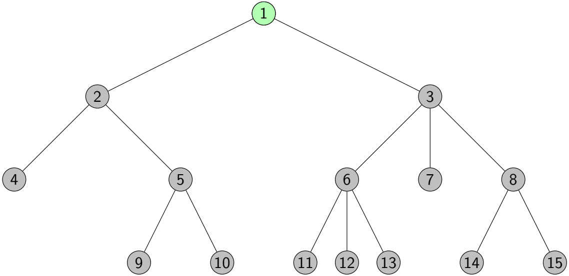
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



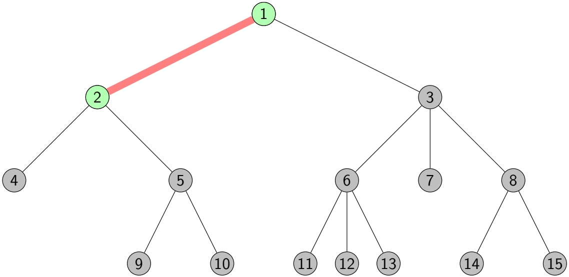
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



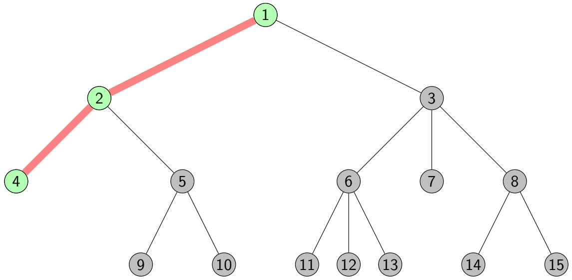
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



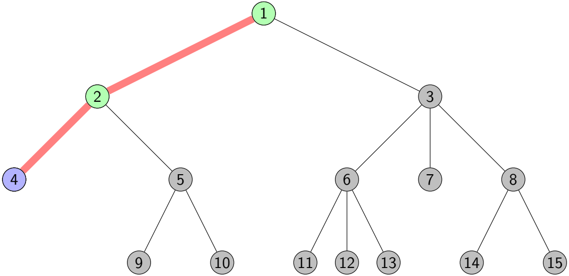
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



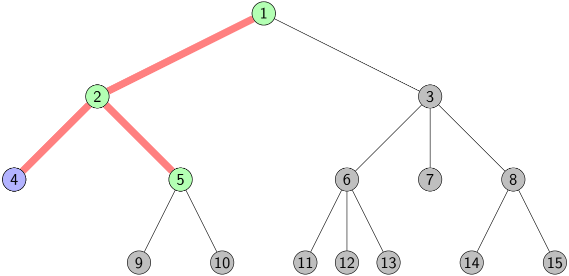
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



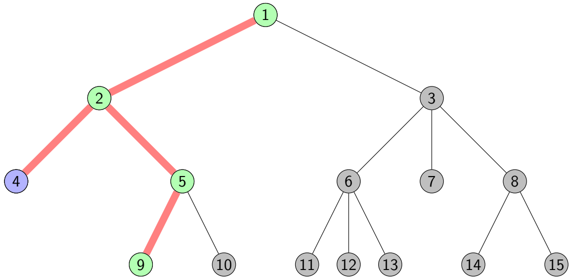
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



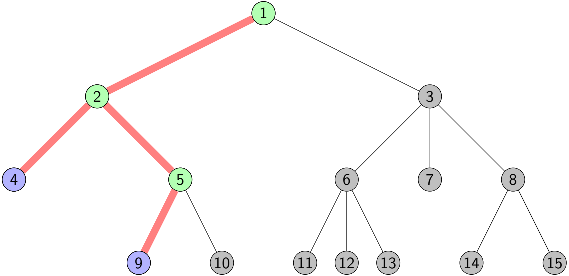
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



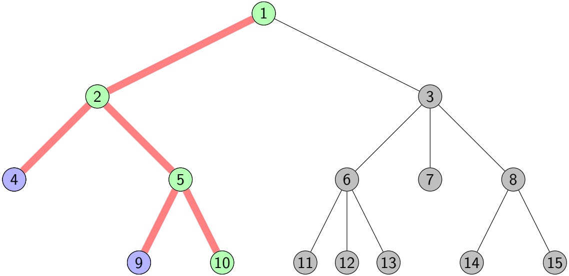
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



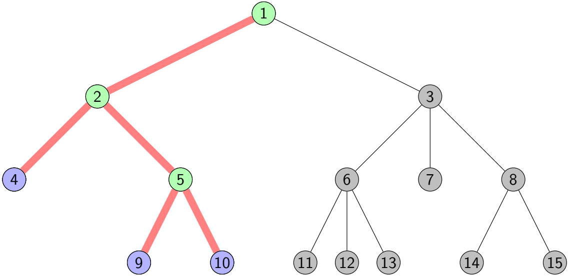
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



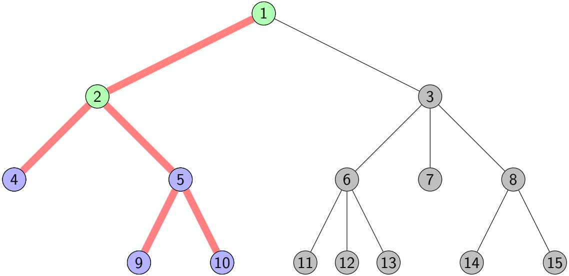
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



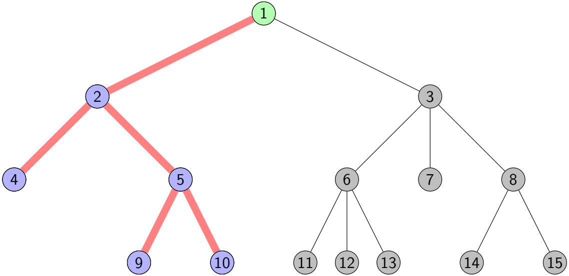
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



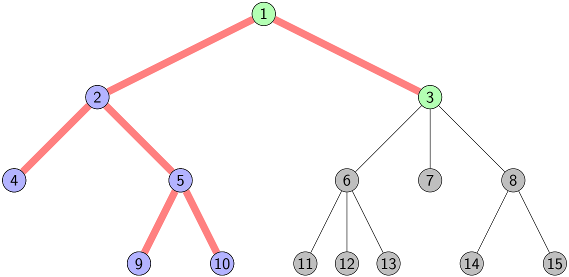
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



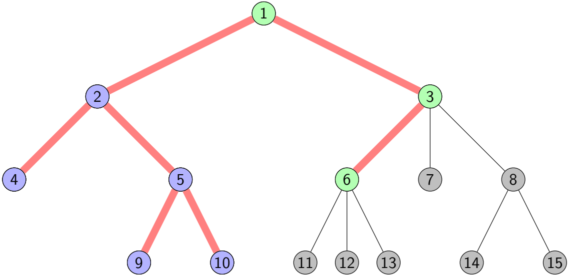
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



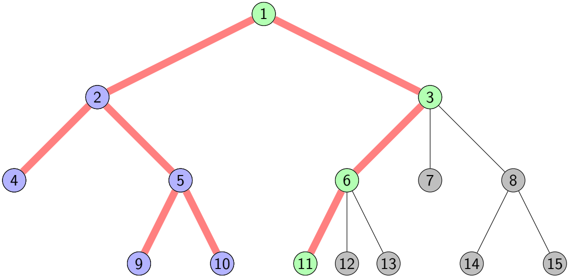
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



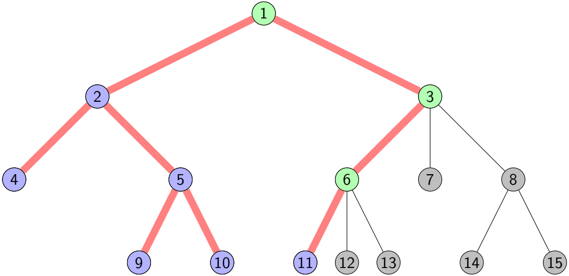
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



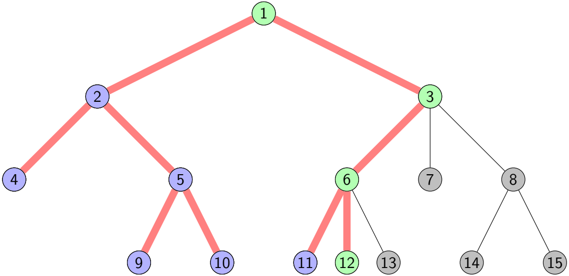
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



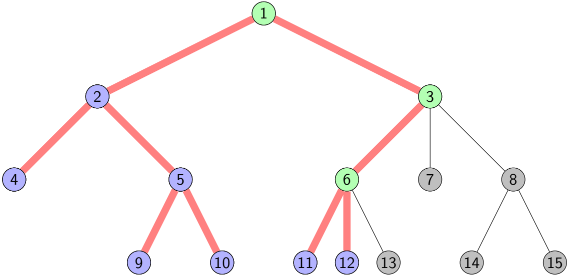
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



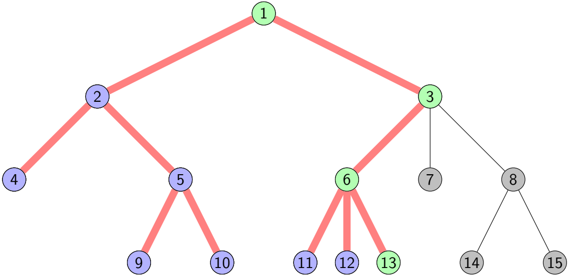
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



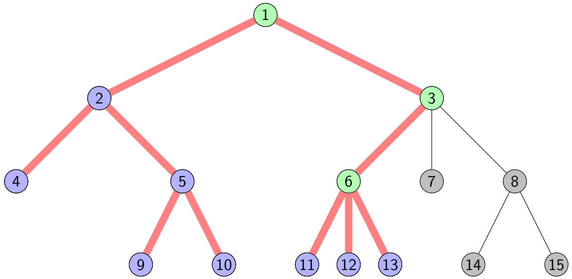
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



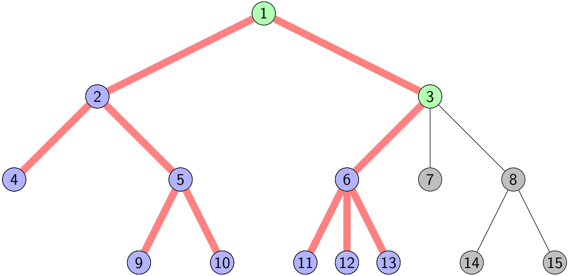
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



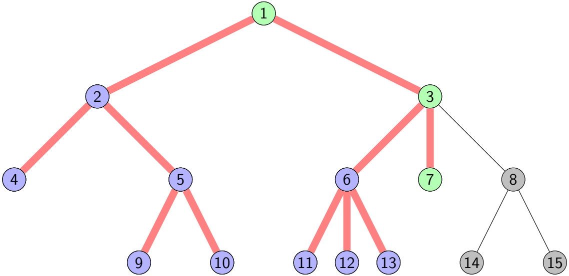
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



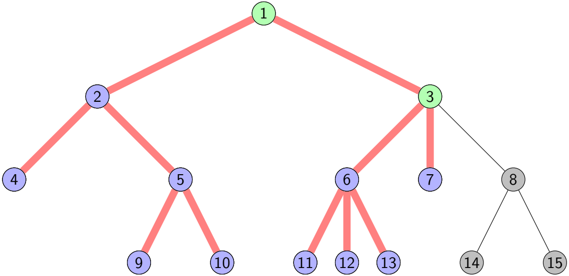
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



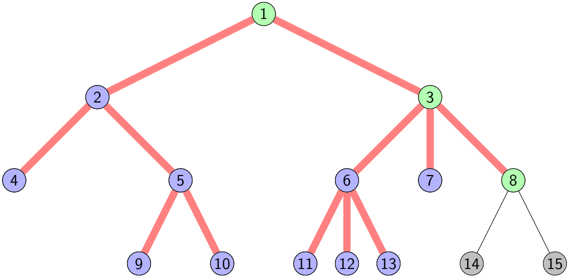
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



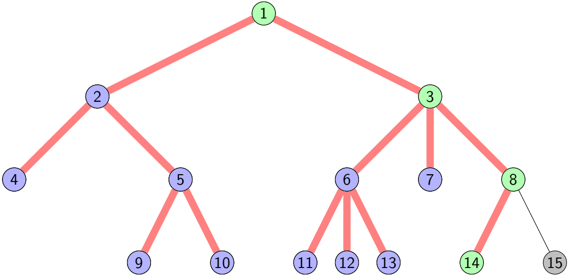
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



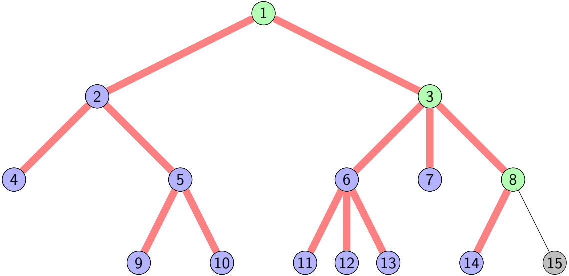
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



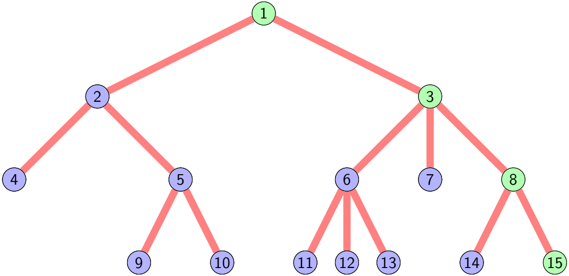
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



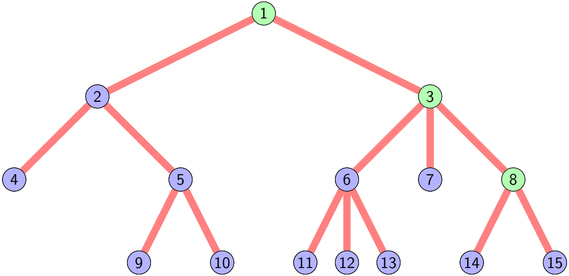
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



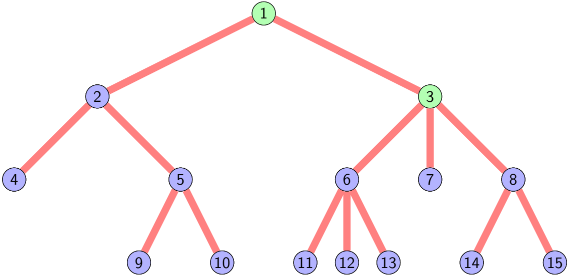
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



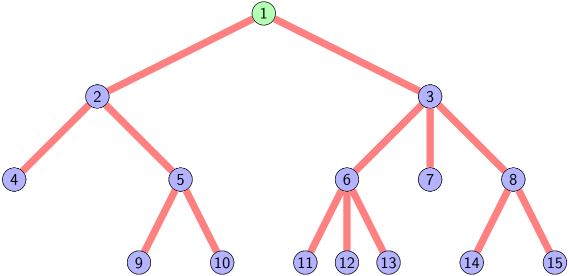
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



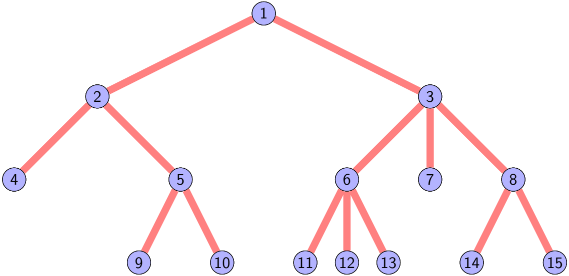
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



Tiefensuche: Datenstruktur

In nahezu allen Programmiersprachen existiert eine Datenstruktur namens **Stack** (Stapel).
Elemente können hinzugefügt werden ("oben drauflegen"), und werden geordnet abgespeichert. Das Element das als *letztes* hinzugefügt wurde, kann entnommen werden (oberstes Element vom Stapel nehmen).

Tiefensuche: Algorithmus (*)

Algorithm 1: Tiefensuche

Function *DFS*($G = (V, E)$, *Startknoten* v, *Gesuchter Knoten* s)
  Result: vertex $s \in V$, if exists
  Stack S;
  markiere v als entdeckt;
  S.push(v); // Lege v auf Stapel
  while S *not empty* **do**
    $v = S$.pop(); // nimm obersten Knoten vom Stapel
    if v *gesuchter Knoten* s **then**
      return v;
    markiere v als abgeschlossen;
    for *all* $[v, u] \in E(G)$ **do**
      if u *noch nicht besucht* **then**
        markiere u als entdeckt;
        S.push(u);

Anmerkungen

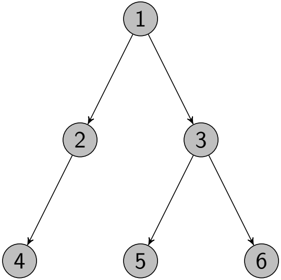
- Die Algorithmen können auch für allgemeine Graphen (und nicht nur Bäume) verwendet werden.
- Dabei werden schon besuchte Knoten *nicht* erneut besucht!
- Anwendungen BFS:
  - 2-färbbarkeit
  - Kürzester Pfad zwischen zwei Knoten
  - Kürzeste-Kreise-Problem
- Anwendungen DFS:
  - Test auf Kreisfreiheit
  - Topologische Sortierung
  - Starke Zusammenhangskomponente

Binärbaum

In einem Wurzelbaum bezeichnet man alle Knoten u als Kinder von v wenn eine Kante (v, u) existiert.

Definition (Binärbaum)

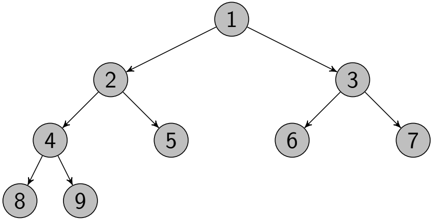
Ein Binärbaum ist ein Wurzelbaum in dem jeder Knoten maximal zwei Kinder hat.



Voller Binärbaum

Definition (Voller Binärbaum)

Ein Binärbaum ist *voll*, wenn alle Knoten entweder 0 oder 2 Kinder haben.



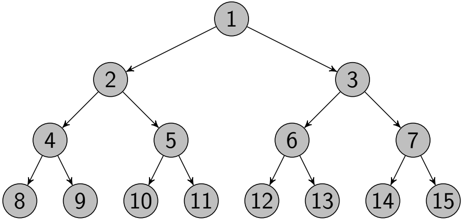
Vollständiger Binärbaum

Definition (Tiefe eines Knotens)

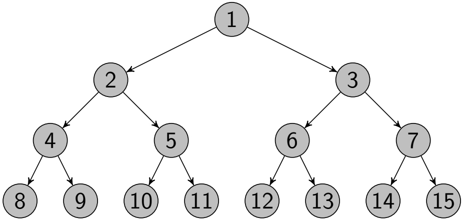
Die Tiefe eines Knotens ist die Distanz zur Wurzel.

Definition (Vollständiger Binärbaum)

Ein vollständiger Binärbaum ist ein voller Binärbaum in dem alle Blätter die gleiche Tiefe haben.



Vollständiger Binärbaum



Tiefe	Anzahl Knoten
0	1
1	2
2	4
3	8

Vollständiger Binärbaum

In jeder Ebene vedoppelt sich die Anzahl der Knoten.

Zusammenhang: Tiefe – Anzahl an Knoten pro Ebene

Die Tiefe t hängt mit der Anzahl der Knoten n in dieser Ebene wie folgt zusammen:

$$2^t = n$$

Tiefe eines vollständigen Binärbaumes ist der *Logarithmus Dualis* der Anzahl der Blätter (hier ebenso mit n bezeichnet):

$$t = \mathrm{ld} n$$

Der Logarithmus Dualis hängt mit dem natürlichen Logaritmus wie folgt zusammen:

$$\mathrm{ld} a = \log_2 a = \frac{\ln a}{\ln 2}$$

Minimale Spannbäume

Definition (Spannbaum)

Ein *Spannbaum* T zu einem Graphen $G = (V, E)$ ist ein (auf-)spannender Teilgraph von G der ein Baum ist.

Wir betrachten nun ungerichtete, schlichte und zusammenhängende Graphen $G = (V, E)$ mit einer Gewichtsfunktion $w : E \to \mathbb{R}^+$ auf den Kanten $e \in E(G)$:

Definition (Minimaler Spannbaum)

Ein *Minimaler Spannbaum* T von $G = (V, E)$ mit $w : E \to \mathbb{R}^+$ für alle $e \in E(G)$ ist ein zusammenhängender Teilgraph mit $|E(T)| = |V(G)| - 1$ der alle Knoten enthält, und für den gilt:

$$\sum_{e \in E(T)} w(e) = \min$$

Ein Minimaler Spannbaum (Minimum Spanning Tree (MST)) ist also ein Baum mit minimalen Kantengewichten (=Kantenkosten).

Algorithmus von Kruskal

Algorithm 2: Algorithmus von Kruskal

Function *Kruskal(Graph* $G = (V, E)$*)*

  Result: Minimum Spanning Tree T

  Ordne alle Kanten $e \in E(G)$ aufsteigend nach $w(e)$;

  $\Rightarrow L = (e_1, e_2, \ldots, e_n)$ mit $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_n)$

  $V(T) = V(G)$;

  $E(T) = \emptyset$;

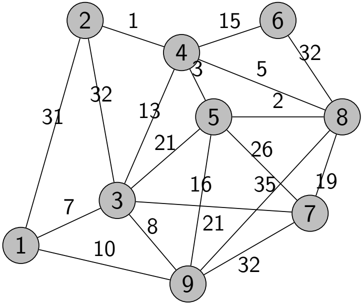
  for $e \in L$ **do**

    if $T = (V, E(T) \cup \{e\})$ *ist kreisfrei* **then**

      $E(T) = E(T) \cup \{e\}$;

- Die Kreisfreiheit kann mit DFS berechnet werden.
- Nach Hinzufügen von $n - 1$ Kanten kann der Algorithmus beendet werden.

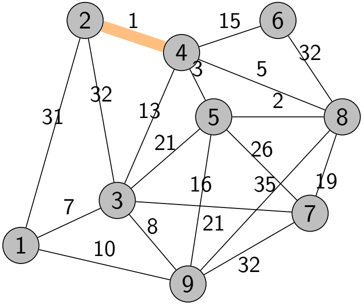
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

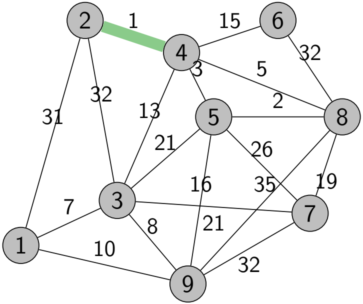
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

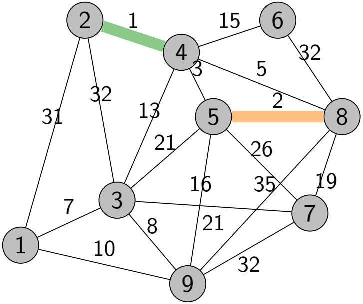
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

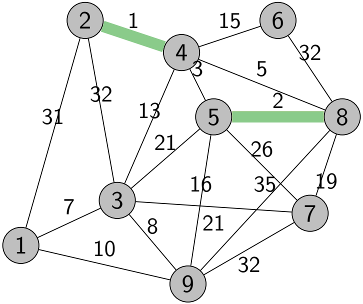
$$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$$
$$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$$
$$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$$
$$w([3,7]) = 21, w([5,7]) = 26, \ldots$$

Algorithmus von Kruskal



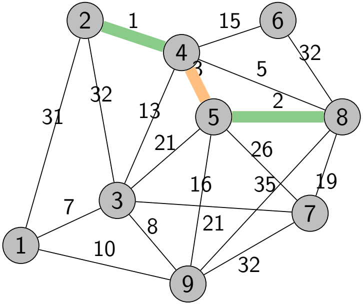
Aufsteigend sortierte Kanten:
$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

Algorithmus von Kruskal



Aufsteigend sortierte Kanten:
$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

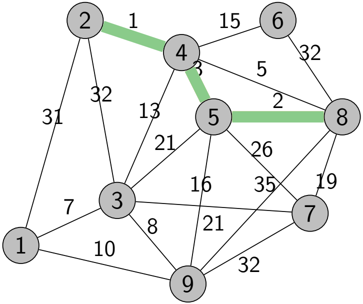
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

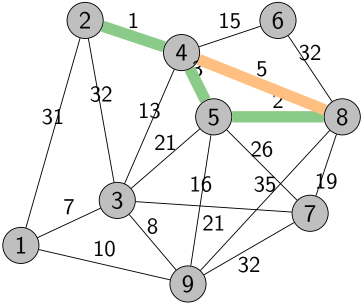
$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \dots$

Algorithmus von Kruskal



Aufsteigend sortierte Kanten:
$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

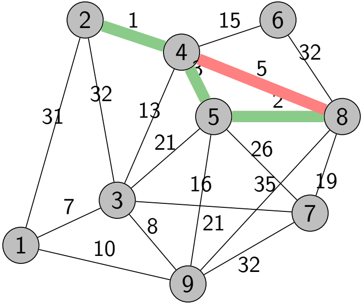
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

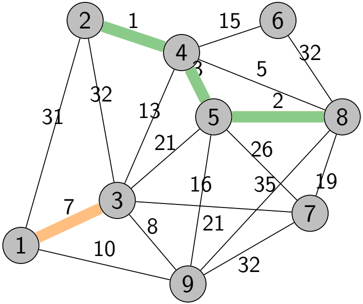
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

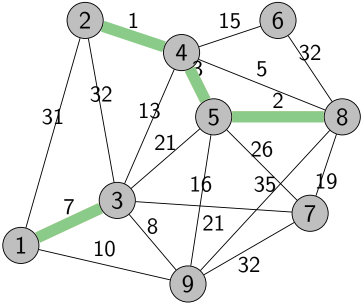
$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

Algorithmus von Kruskal



Aufsteigend sortierte Kanten:
$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

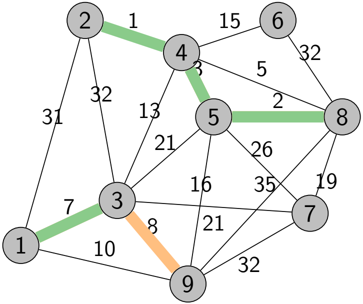
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

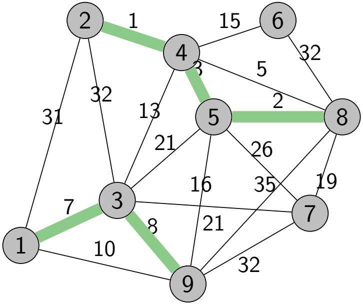
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

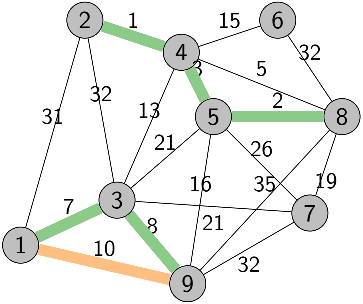
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

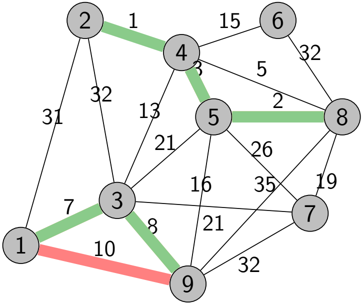
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

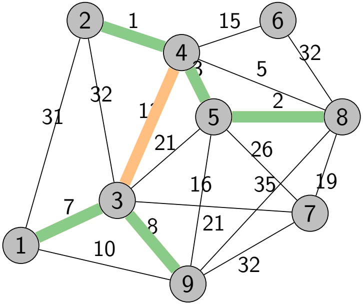
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

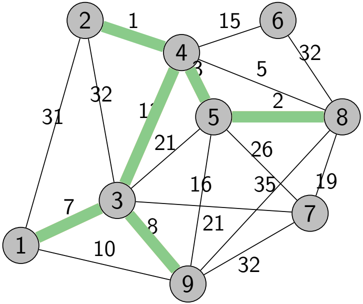
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

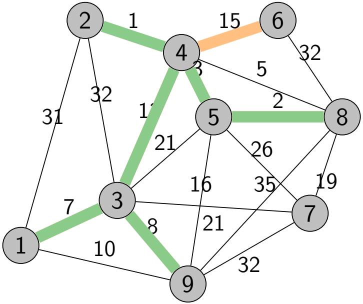
$$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$$
$$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$$
$$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$$
$$w([3,7]) = 21, w([5,7]) = 26, \dots$$

Algorithmus von Kruskal



Aufsteigend sortierte Kanten:
$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

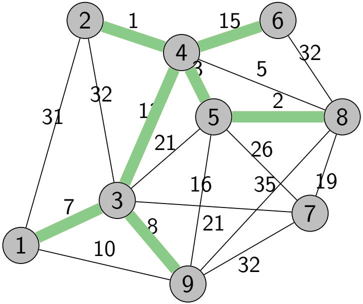
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

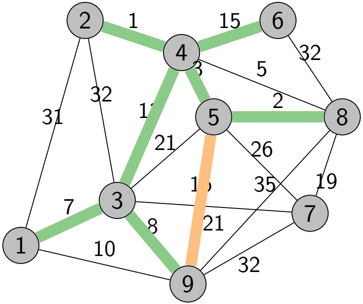
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

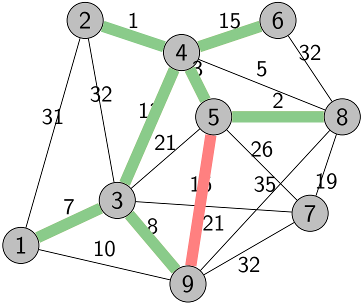
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

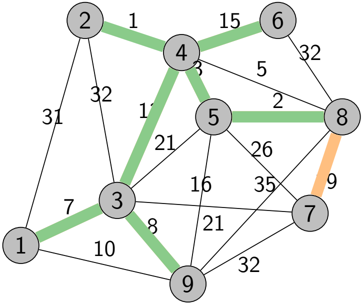
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

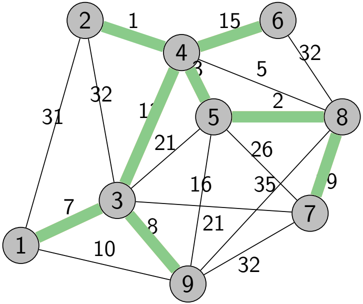
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$w([2,4]) = 1, w([5,8]) = 2, w([4,5]) = 3, w([4,8]) = 5$
$w([1,3]) = 7, w([3,9]) = 8, w([1,9]) = 10, w([3,4]) = 13$
$w([4,6]) = 15, w([5,9]) = 16, w([7,8]) = 19, w([3,5]) = 21$
$w([3,7]) = 21, w([5,7]) = 26, \ldots$

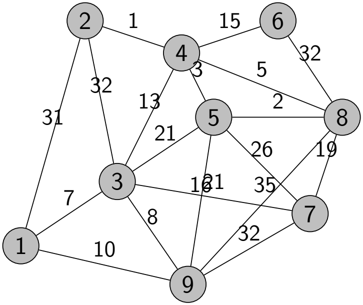
Algorithmus von Prim

Der Algorithmus von Prim konstruiert ebenfalls einen MST.

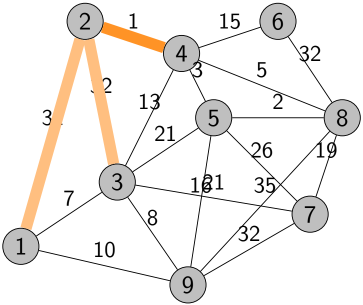
Algorithm 3: Algorithmus von Prim

Function *Prim(Graph G)*
    Result: Minimum Spanning Tree T
    $E(T) = \emptyset$;
    $V(T) = \text{ein zufaellig gewaehlter Startknoten}$;
    while $V(T) \subset V(G)$ **do**
        Sei E' die Menge aller Kanten zwischen Knoten
        aus $V(T)$ und $V(G) \backslash V(T)$
        $e' = \text{argmin}_{e \in E'} w(e)$; // Kante mit kleinstem Gewicht
        $E(T) = E(T) \cup e'$;
        Füge neuen Knoten von e' zu $V(T)$ hinzu;

Algorithmus von Prim

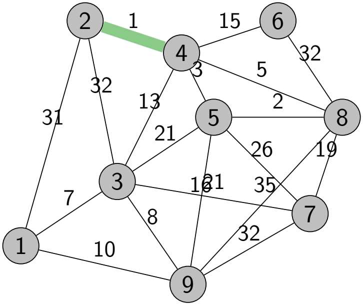


Algorithmus von Prim

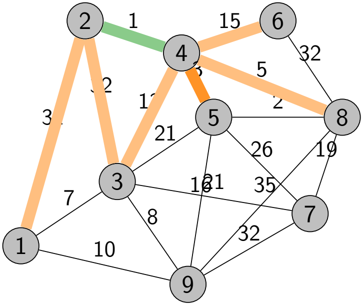


Startknoten: 2

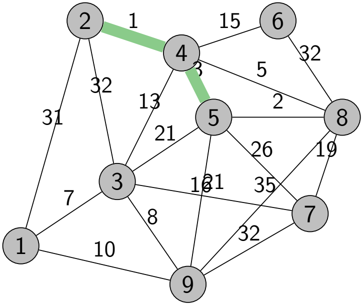
Algorithmus von Prim



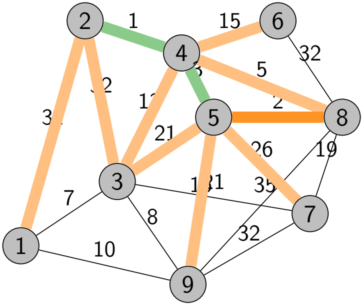
Algorithmus von Prim



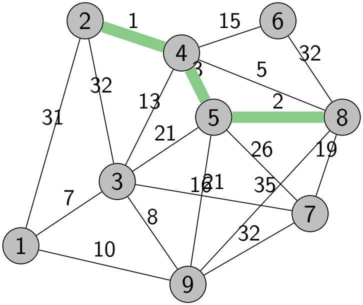
Algorithmus von Prim



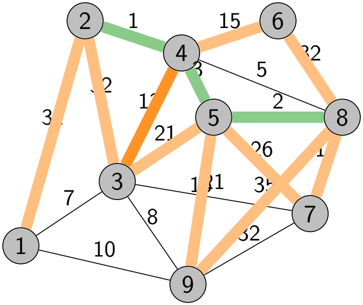
Algorithmus von Prim



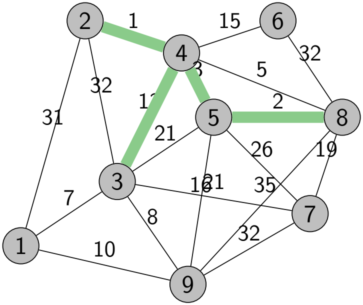
Algorithmus von Prim



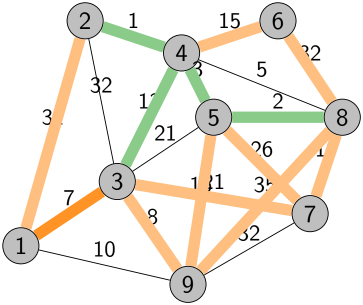
Algorithmus von Prim



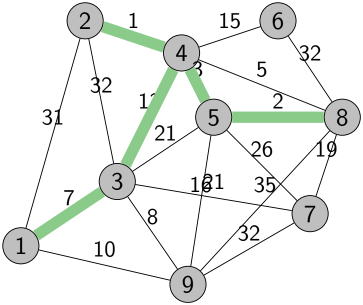
Algorithmus von Prim



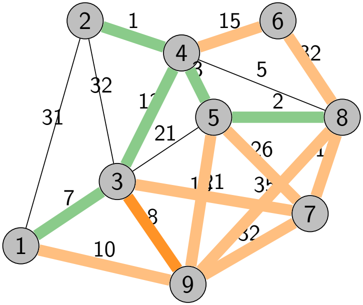
Algorithmus von Prim



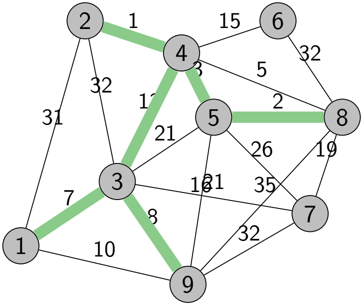
Algorithmus von Prim



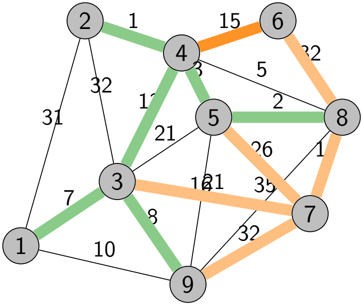
Algorithmus von Prim



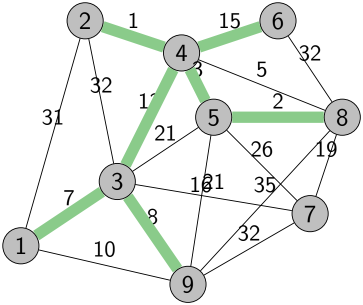
Algorithmus von Prim



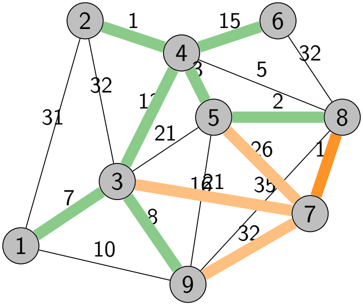
Algorithmus von Prim



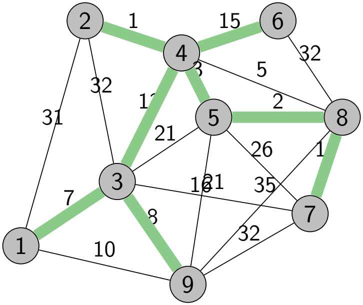
Algorithmus von Prim



Algorithmus von Prim



Algorithmus von Prim



Vergleich: Kruskal vs. Prim

- Beide Algorithmen (Kruskal und Prim) finden den MST in gewichteten Graphen.
- Bei *dicht*[2] besetzten Graphen ist der Algorithmus von Prim jedoch effizienter.
- Bei *dünn*[3] besetzten Graphen ist Kruskal besser.
- Beim Algorithmus von Kurskal müssen die Kanten vorab sortiert werden, was bei vielen Kanten einen erheblichen Aufwand darstellt (sogar den Hauptaufwand des gesamten Verfahrens).
- Bei vergleichsweise wenigen Kanten überwiegen jedoch die Vorteile der einfacheren darauffolgenden Schritte.

[2] $|E| \in O(|V|^2)$, d.h. die Anzahl der Kanten liegt in der Größenordnung der Anzahl der Kanten eines vollständigen Graphen.

[3] $|E| \in O(|V|)$, d.h. die Anzahl der Kanten liegt in der Größenordnung der Anzahl der Knoten; der Graph enthält also relativ wenige Kanten.

Zusammenfassung (MST)

- Beide Algorithmen (Kruskal, Prim) sind sogenannte **Greedy-Algorithmen**
- Sie wählen in jedem Schritt ("gierig") die nächst-beste Erweiterung der Teillösung
- Normalerweise erreicht man mit einer derartigen Strategie nur **Näherungslösungen** (d.h. nicht die insgesamt beste Lösung)
- In diesem Fall finden jedoch beide Greedy-Algorithmen das **globale Optimum**, d.h. die insgesamt beste Lösung
- ...der Minimale Spannbaum kann also (wie der Euler-Zyklus) leicht gefunden werden.
- Andere Spannbaum-Probleme (Varianten) sind jedoch wesentlich schwieriger!