Graphentheorie: Algorithmen 1

Programmieren und Software-Engineering
Theorie

23. Dezember 2025

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:

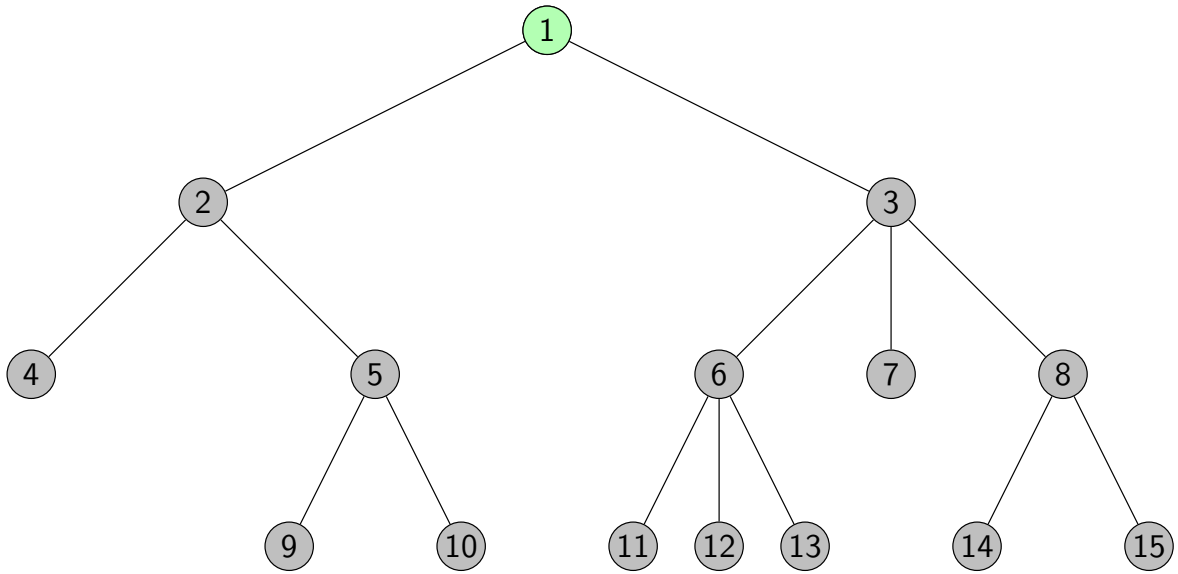
- Ein Knoten wird **entdeckt**, wenn er das erste Mal besucht wird.
- Ein Knoten wird **fertiggestellt/abgeschlossen**, wenn er das letzte Mal verlassen wird.

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

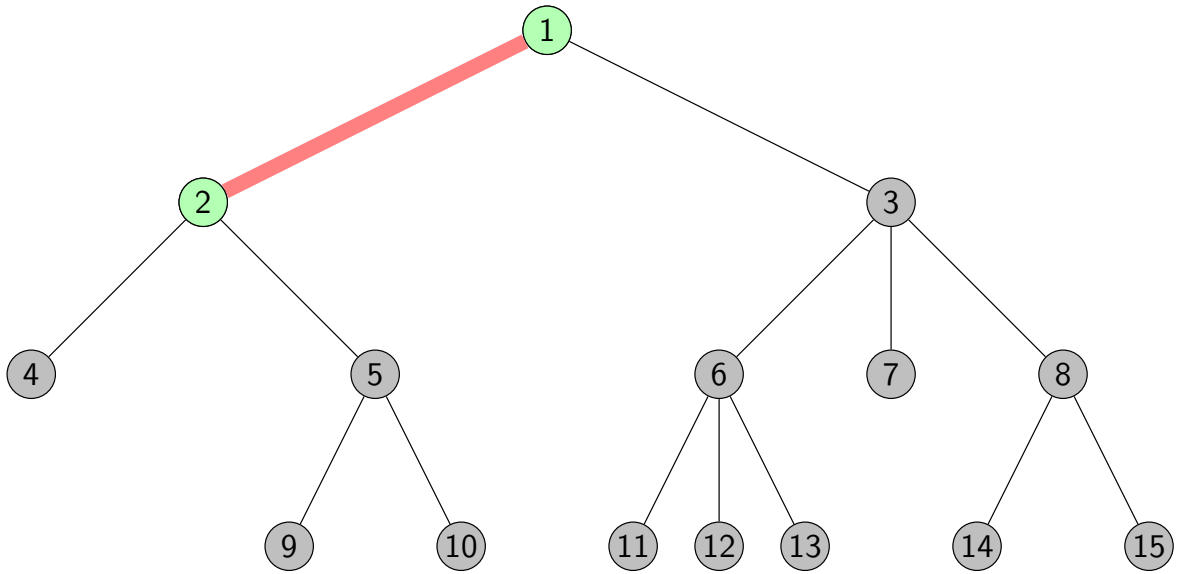
Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.



$\tau_d(1) = 1,$

Breitensuche

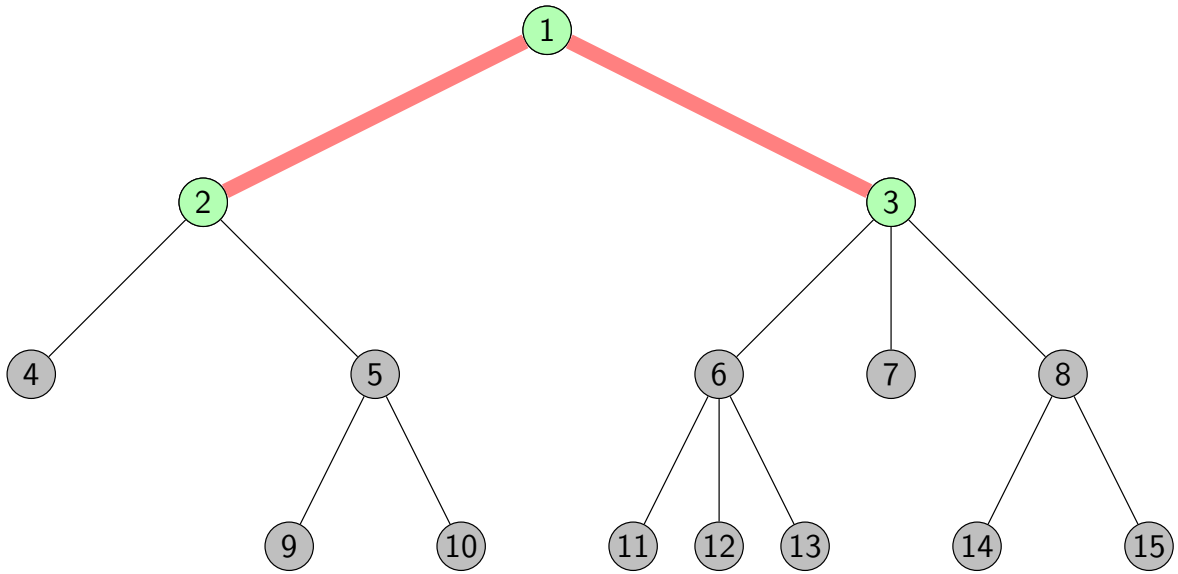
Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.



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

Breitensuche

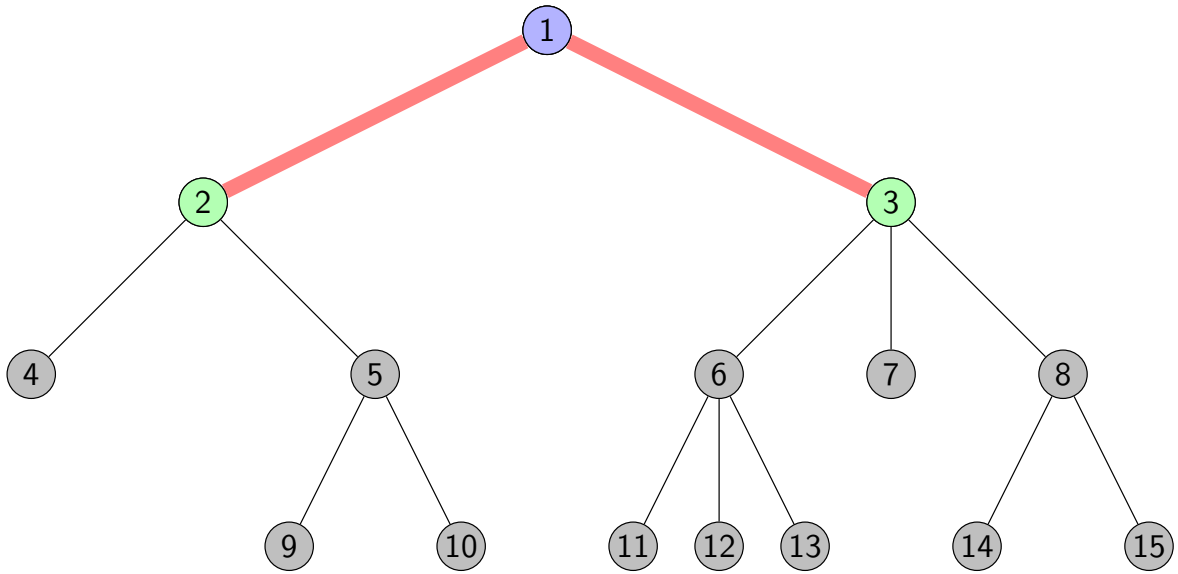
Beispiel: Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.



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

Breitensuche

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

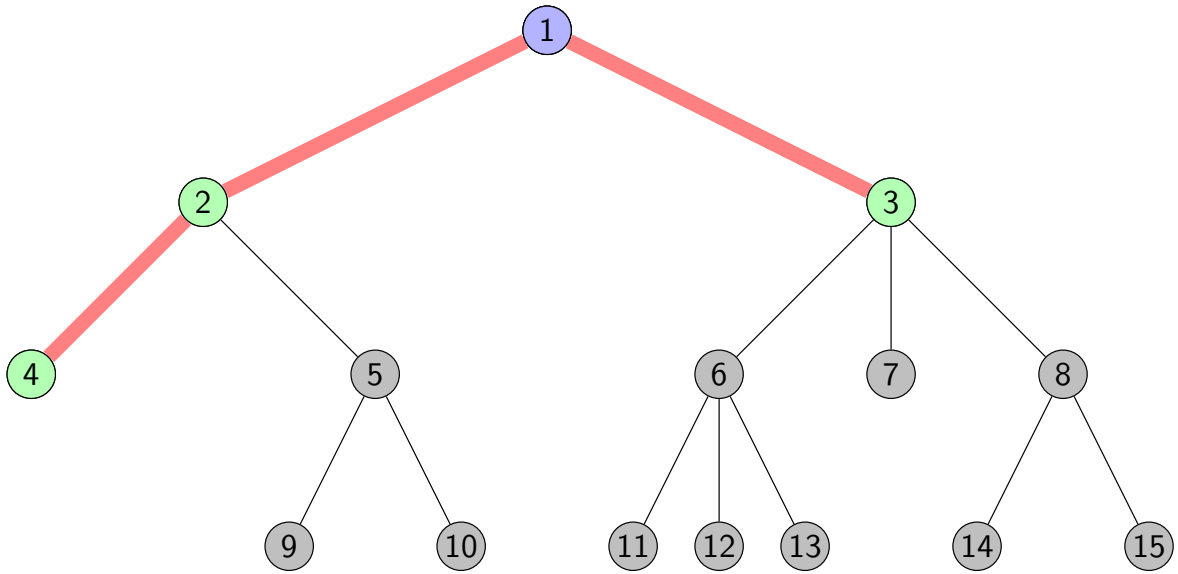


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3,$

$\tau_f(1) = 4,$

Breitensuche

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

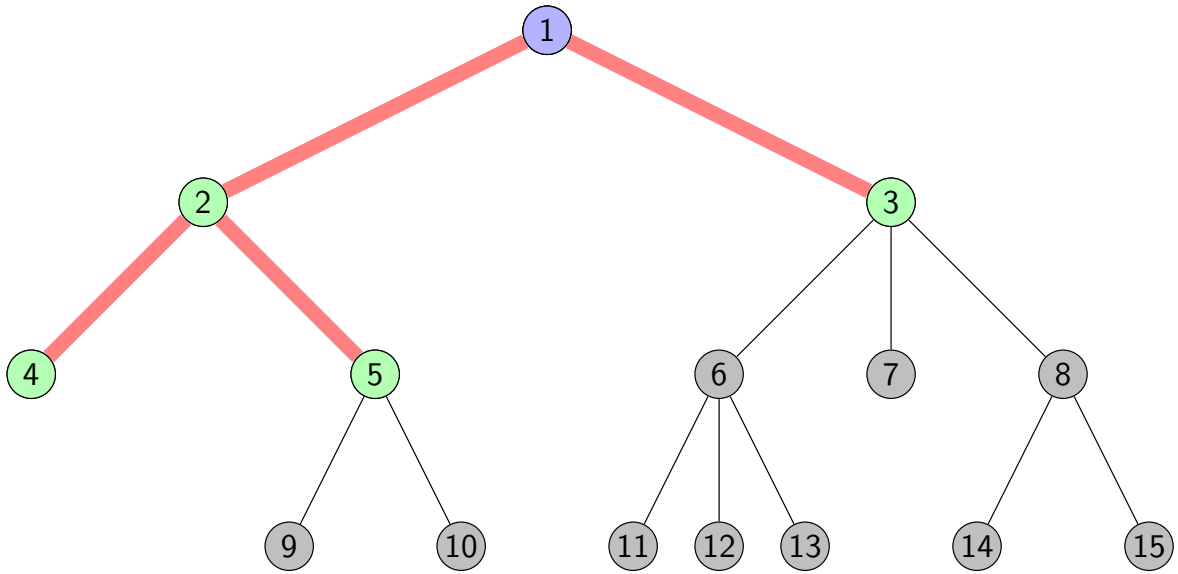


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5,$

$\tau_f(1) = 4,$

Breitensuche

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

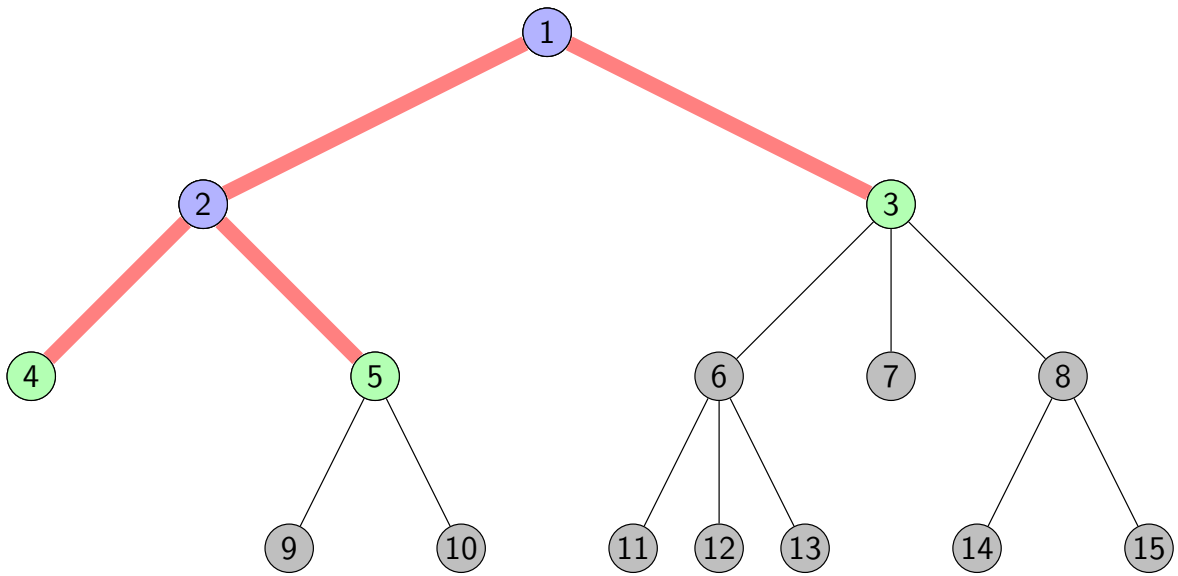


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6,$

$\tau_f(1) = 4,$

Breitensuche

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

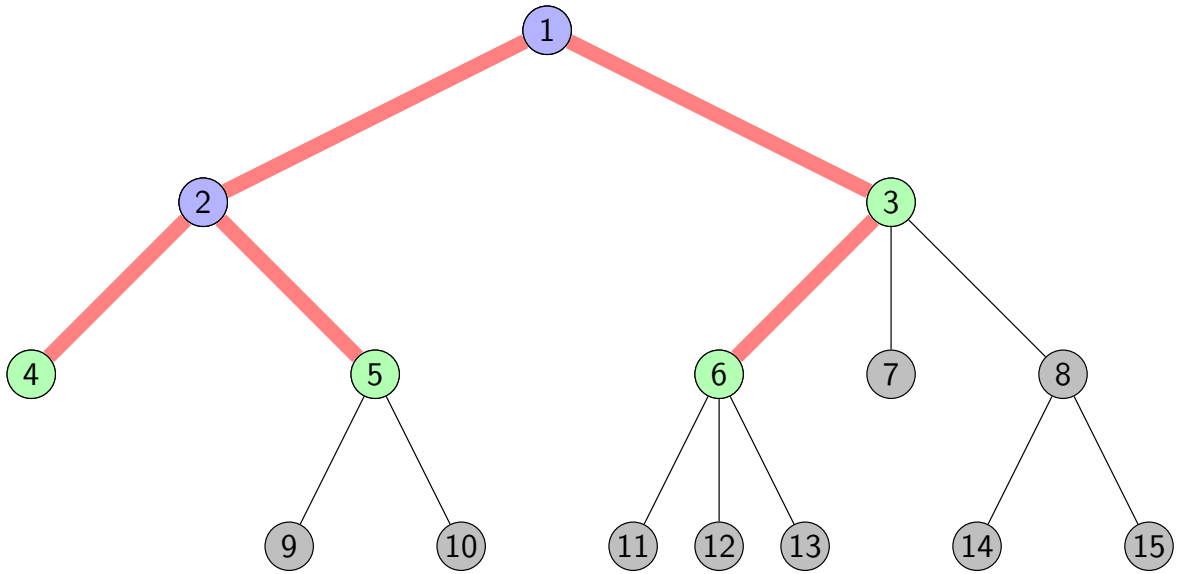


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6,$

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

Breitensuche

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

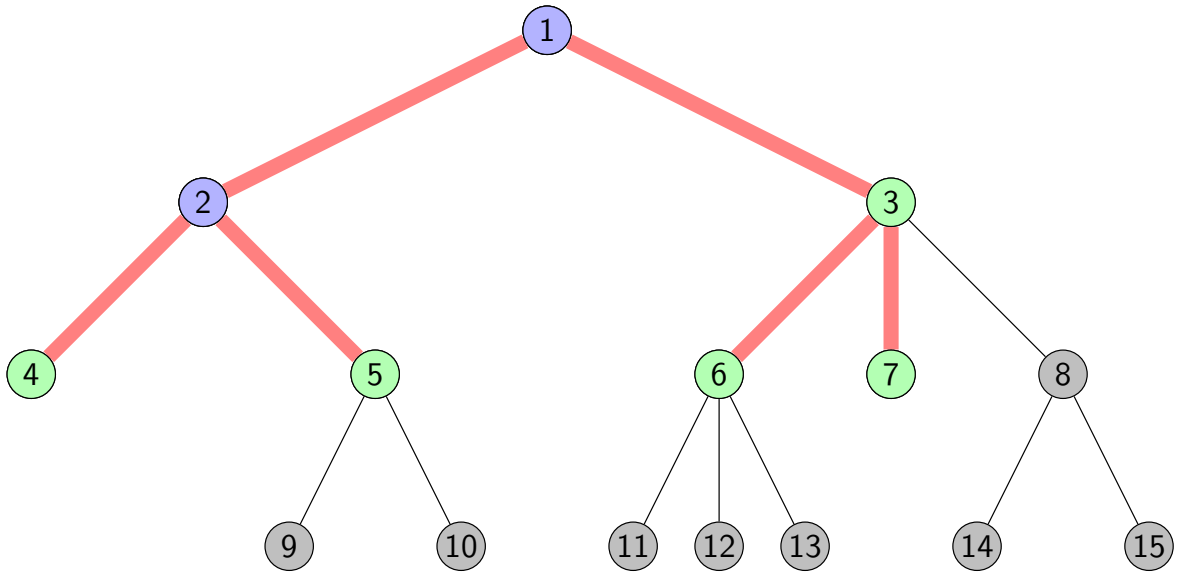


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8,$

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

Breitensuche

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

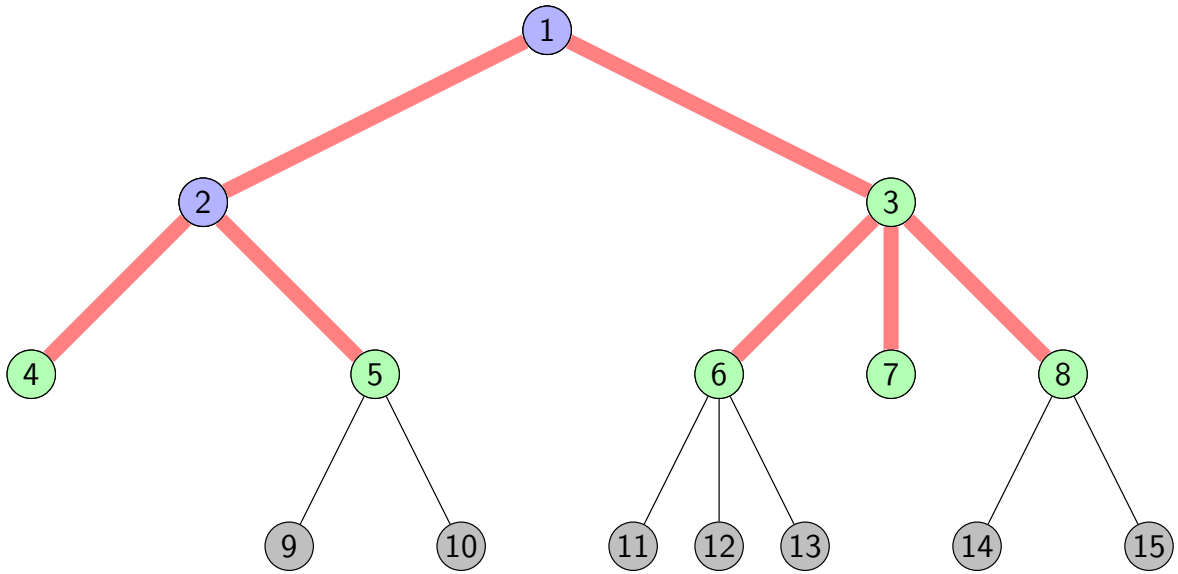


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9,$

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

Breitensuche

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

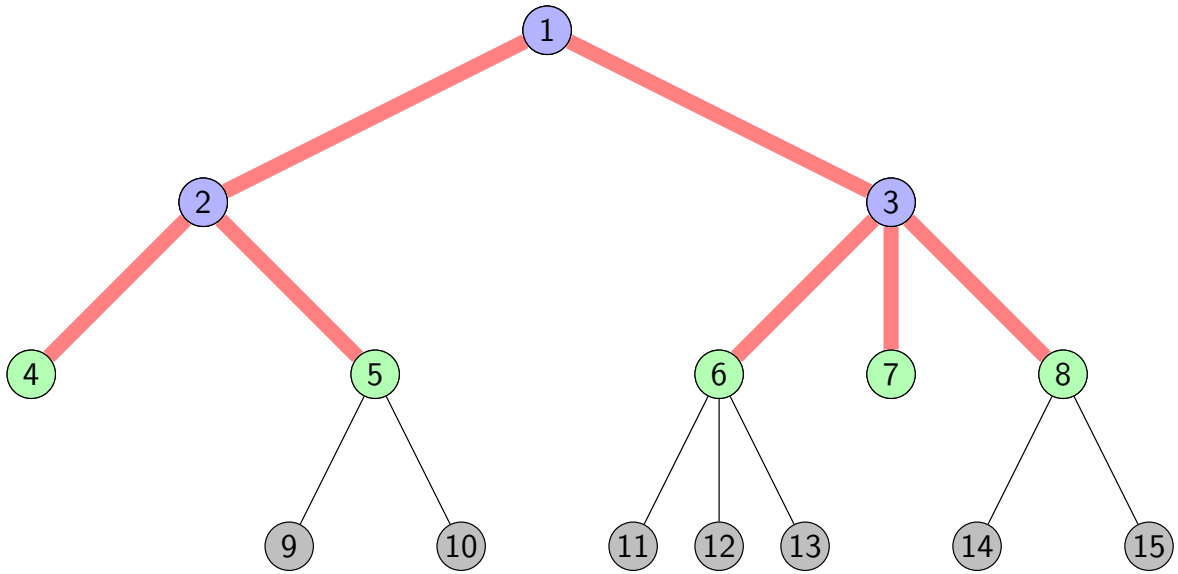


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10,$

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

Breitensuche

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

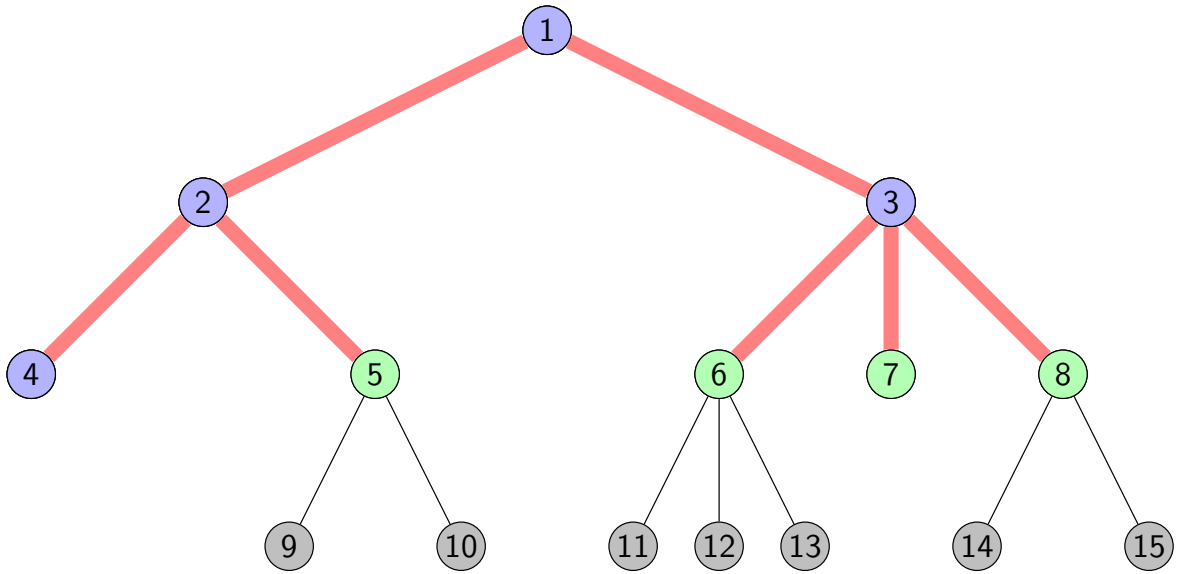


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10,$

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

Breitensuche

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

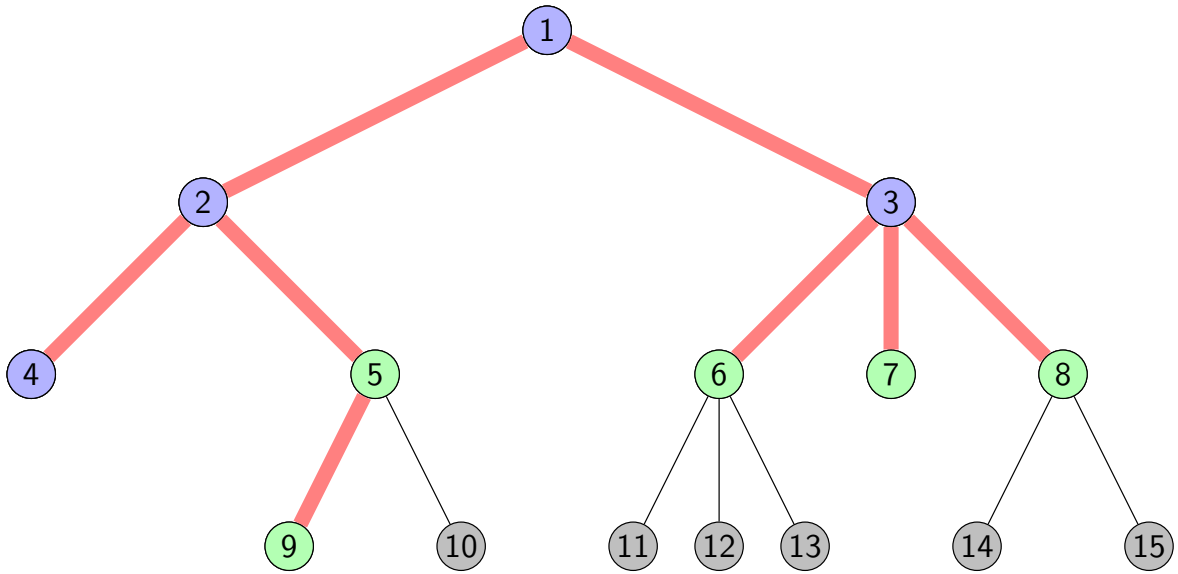


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10,$

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

Breitensuche

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

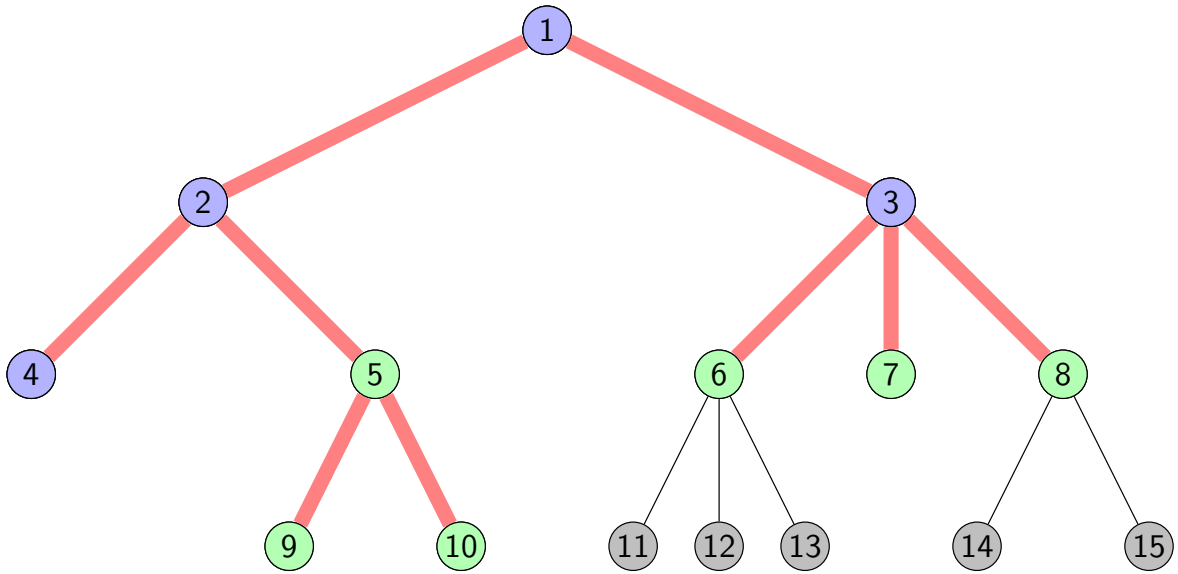


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13,$

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

Breitensuche

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

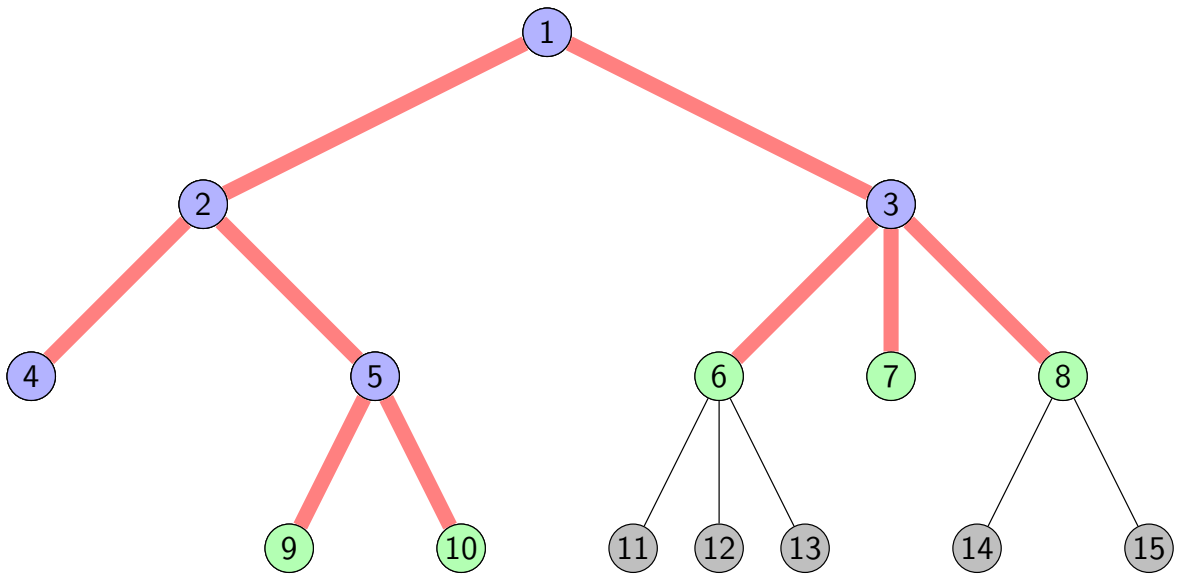


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14,$

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

Breitensuche

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

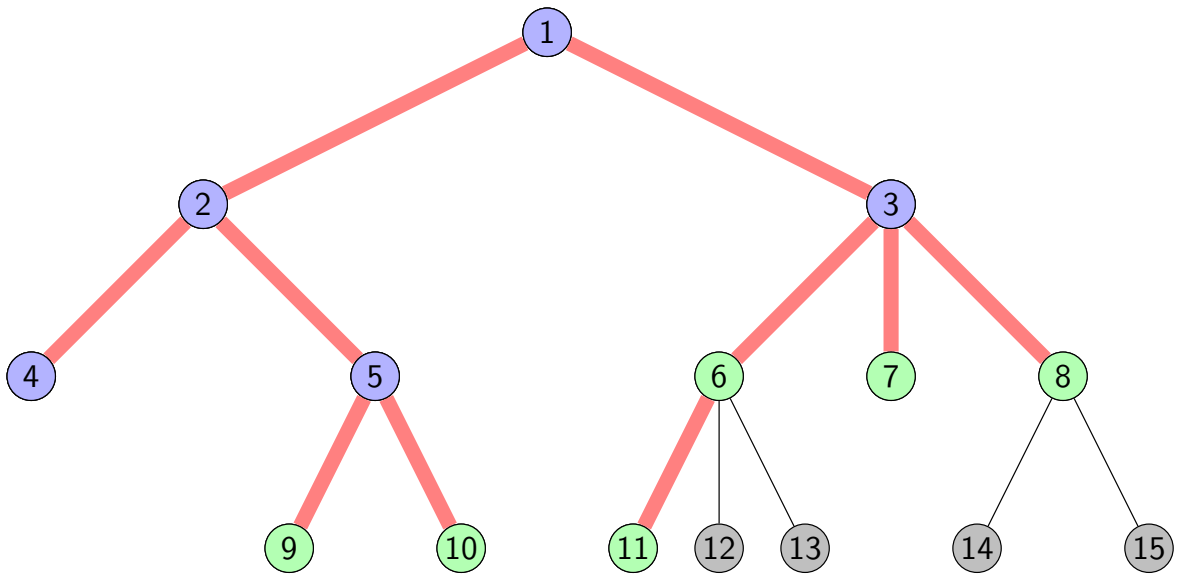


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14,$

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

Breitensuche

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

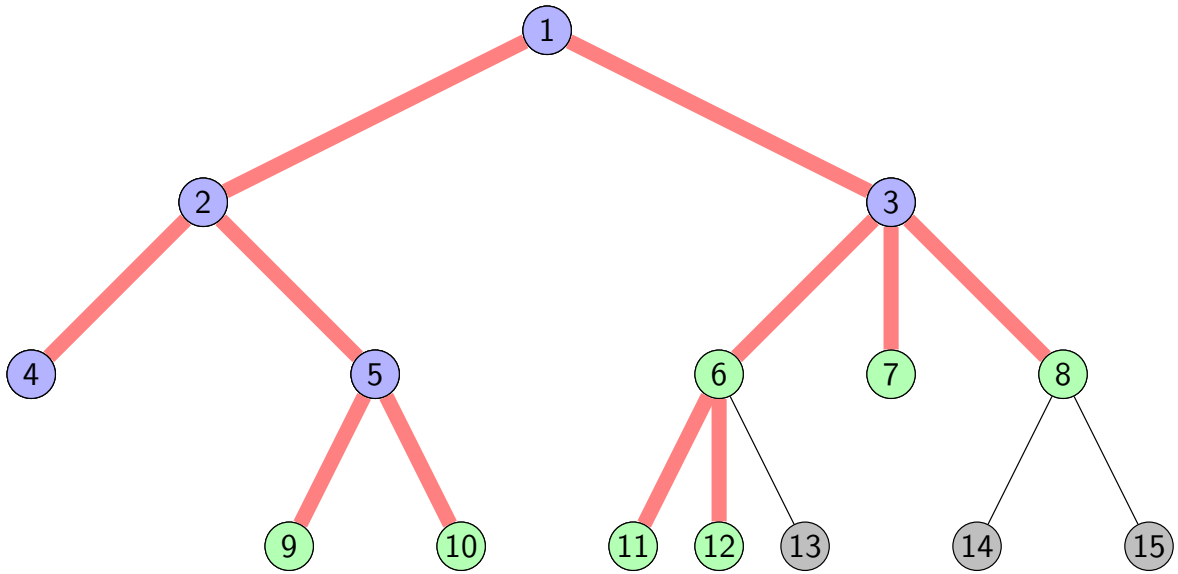


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

Breitensuche

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

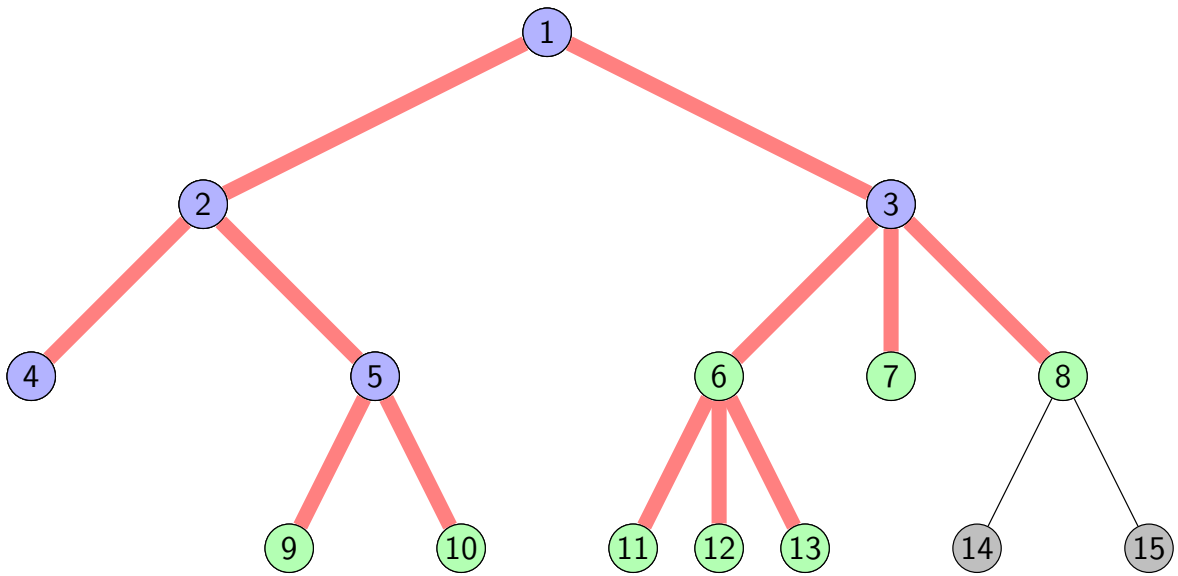


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

Breitensuche

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

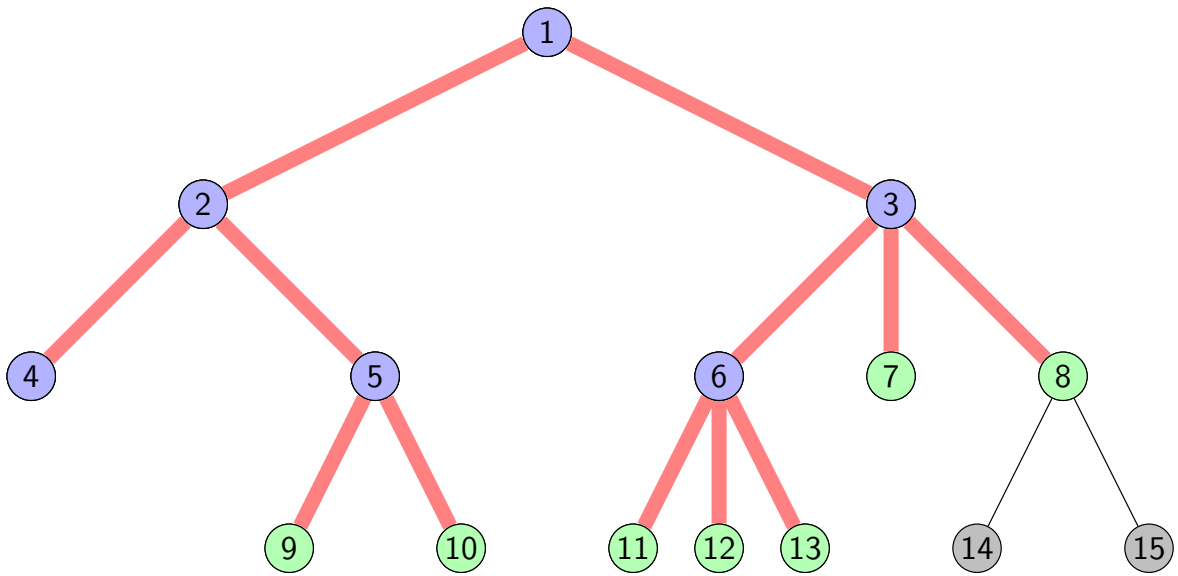


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

Breitensuche

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

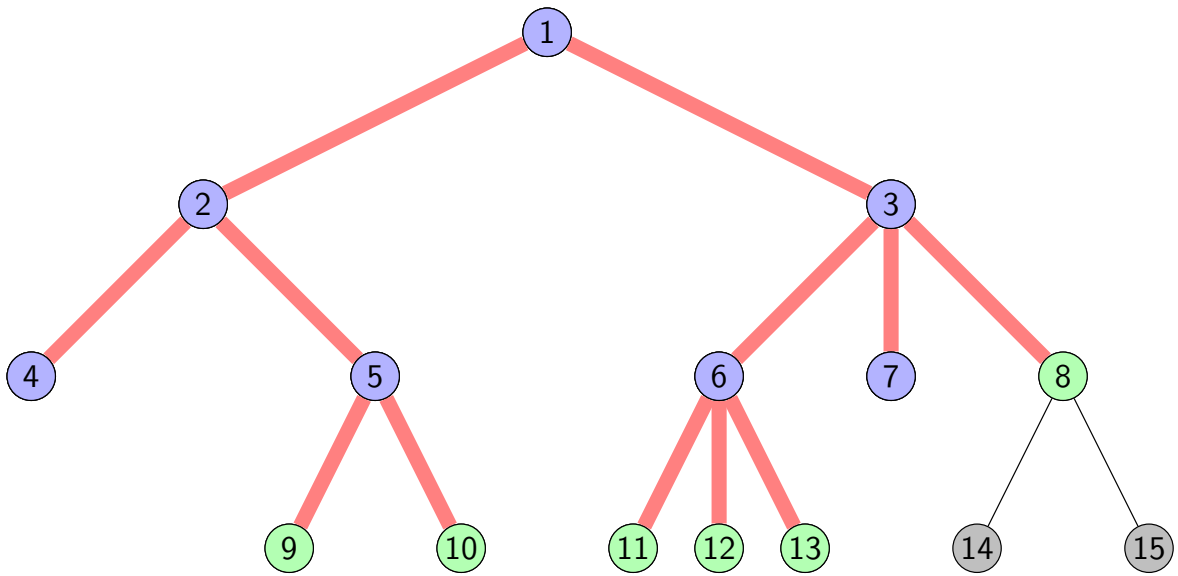


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

Breitensuche

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

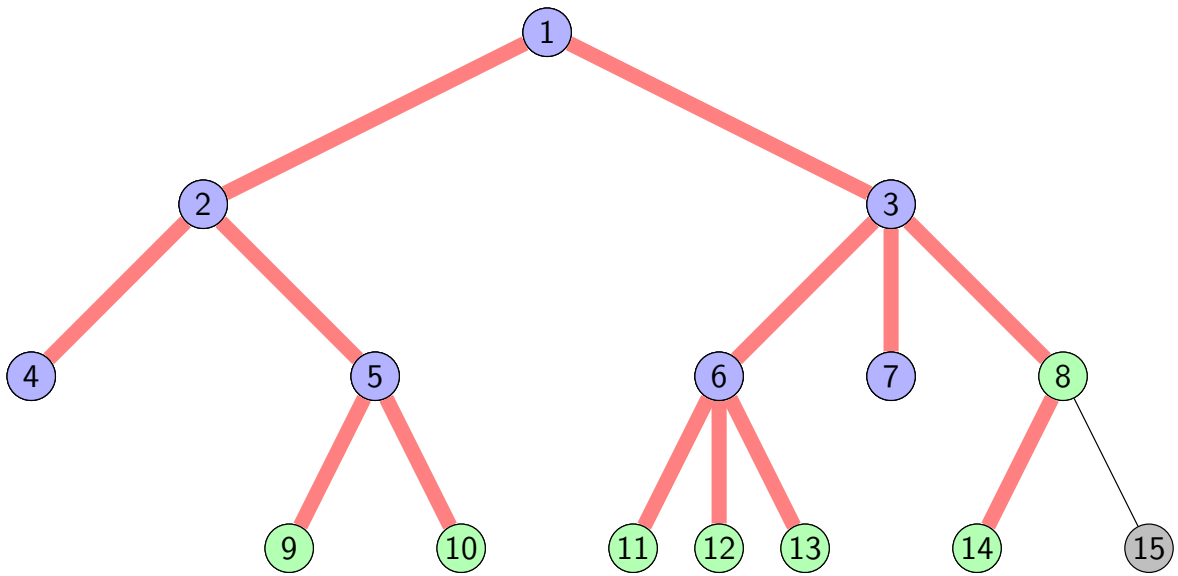


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

Breitensuche

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

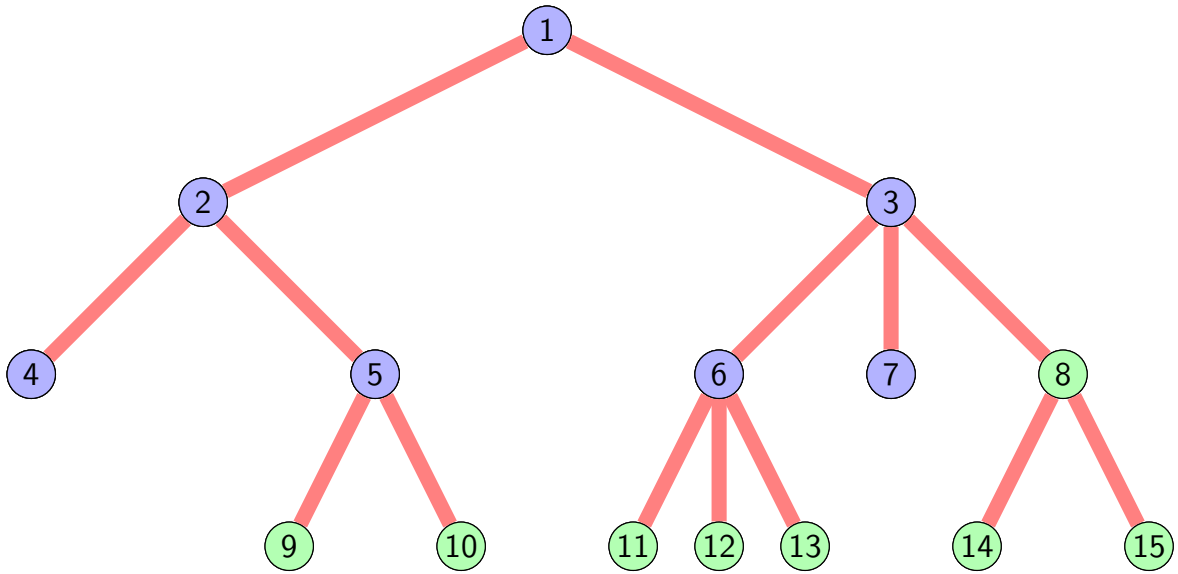


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

Breitensuche

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

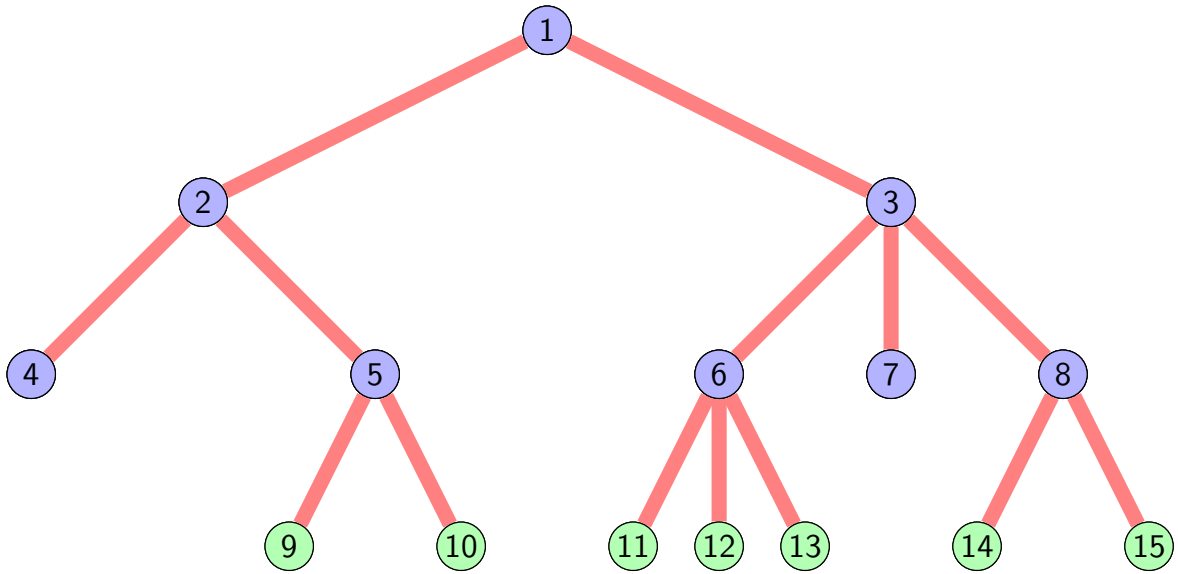


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

Breitensuche

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

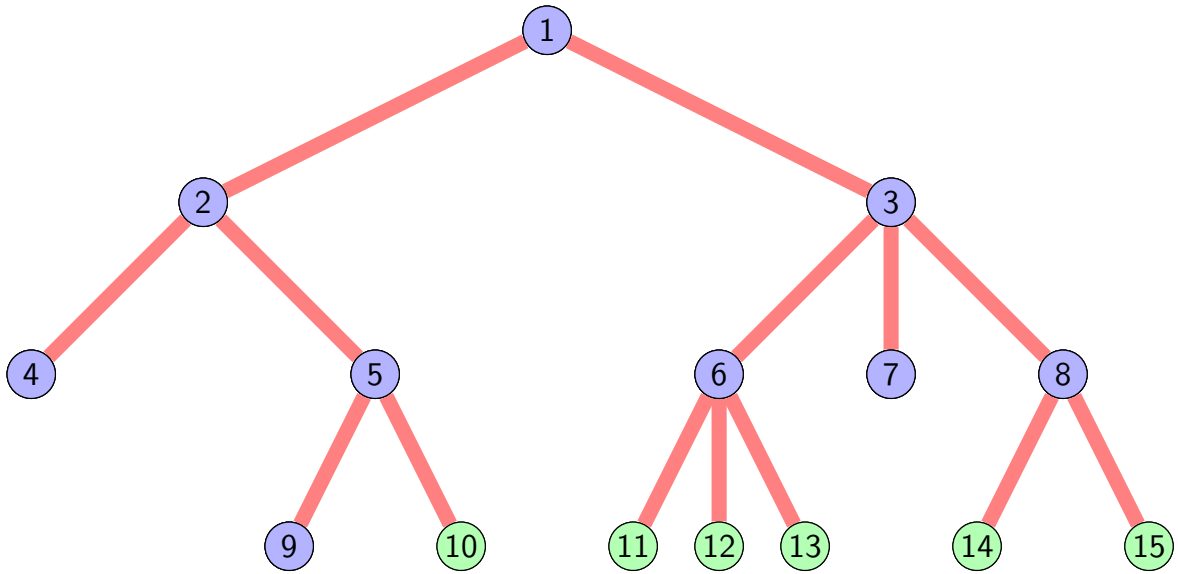


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

Breitensuche

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

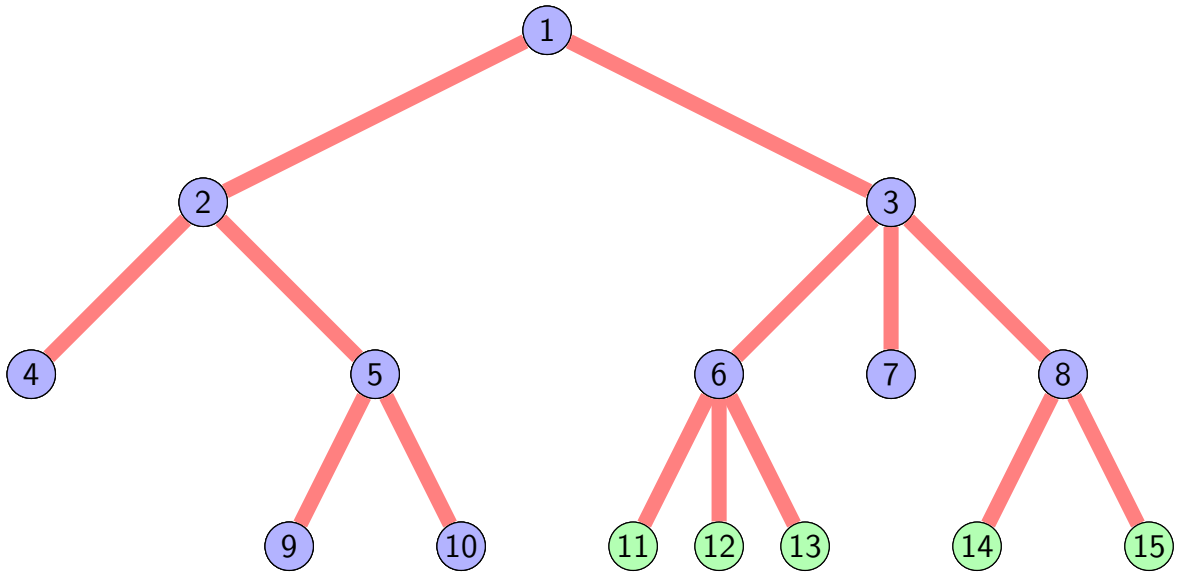


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

Breitensuche

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

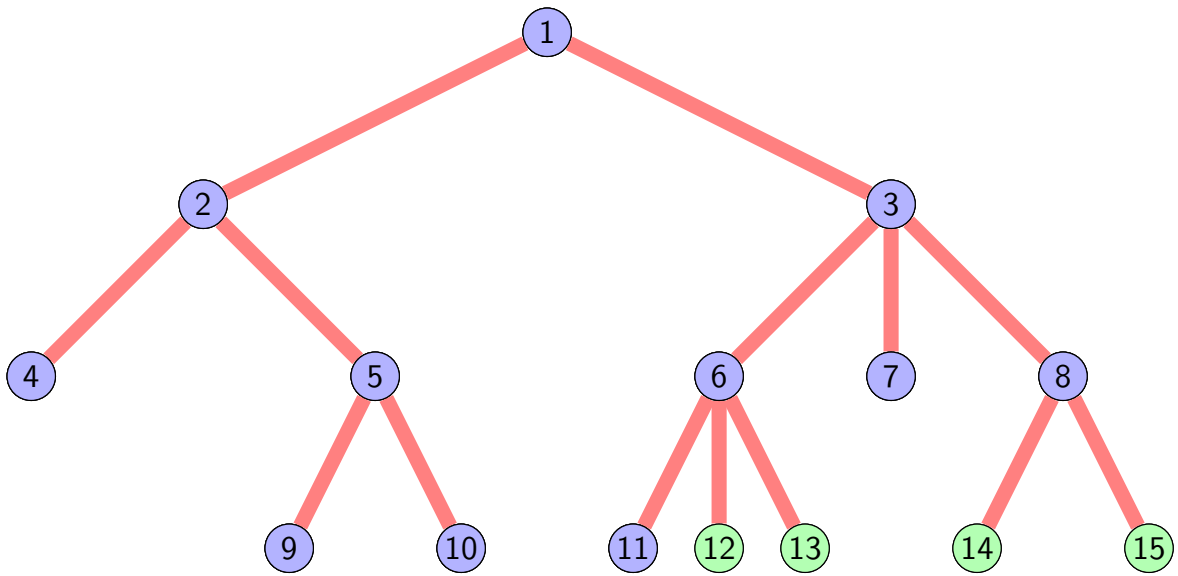


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

Breitensuche

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

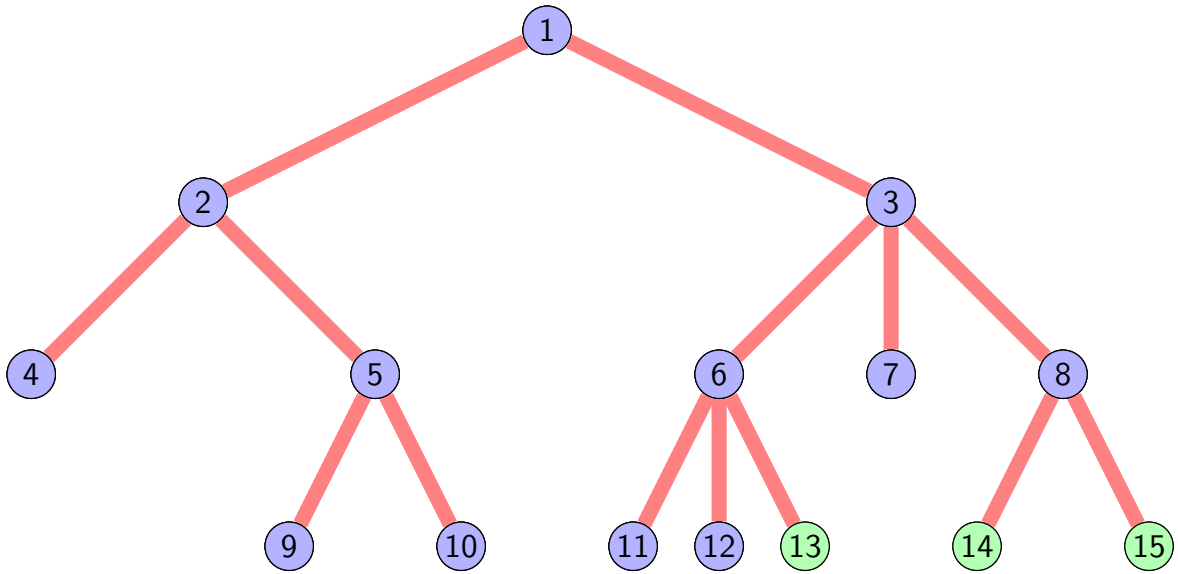


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

Breitensuche

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

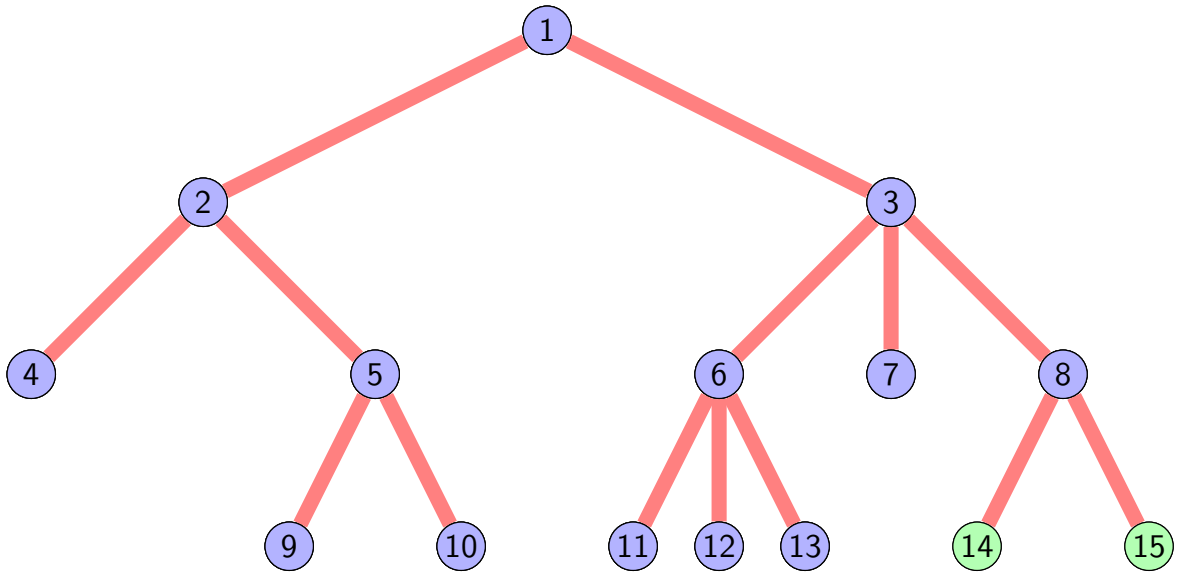


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

Breitensuche

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

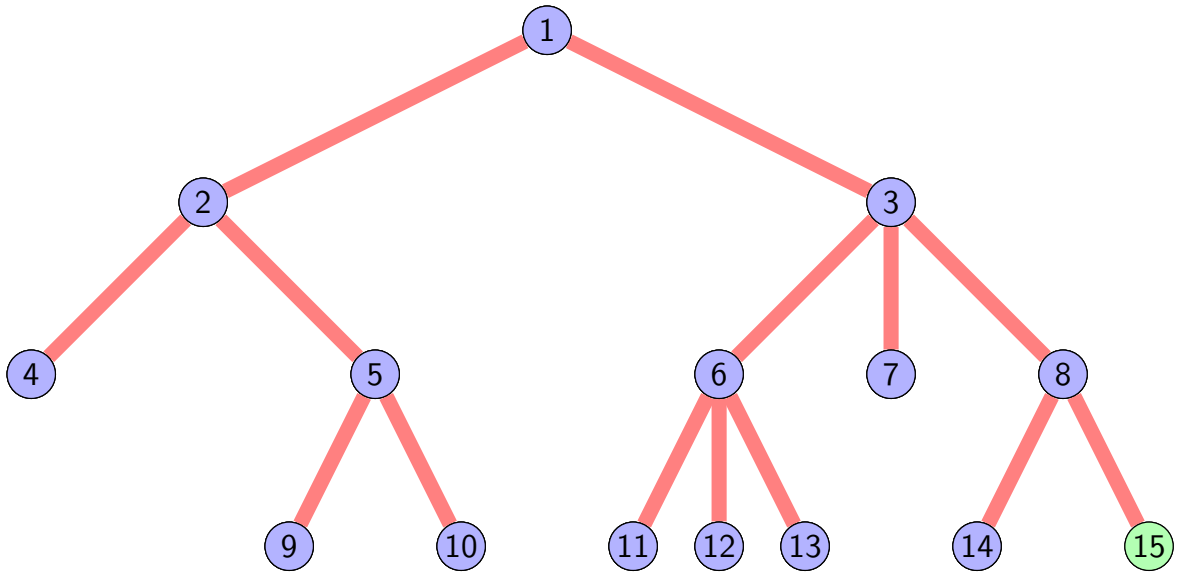


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

Breitensuche

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

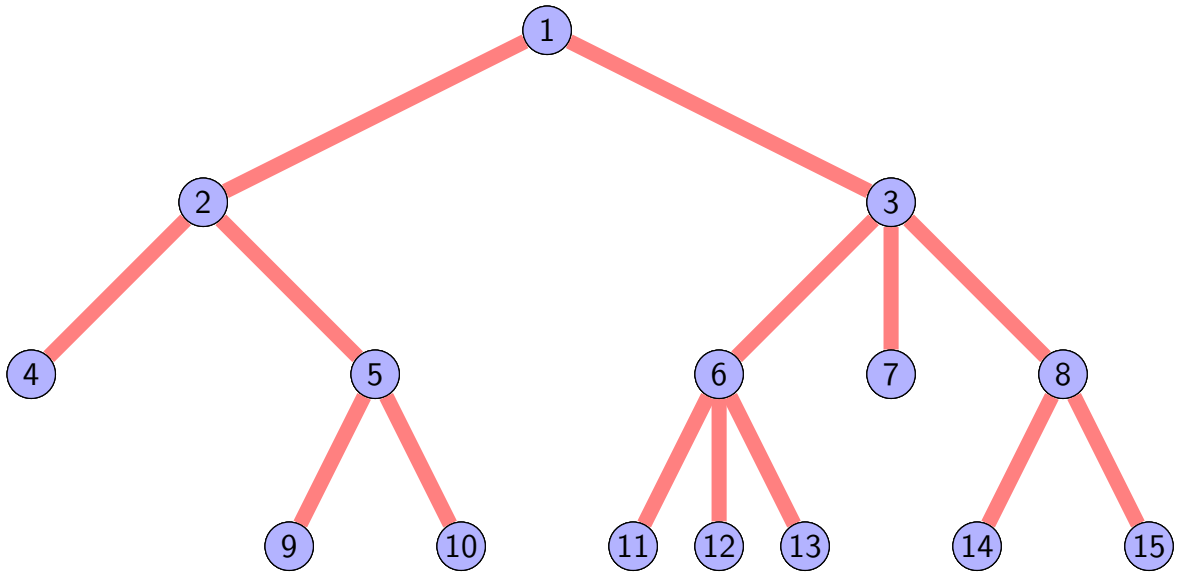


$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \ldots$

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

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!")

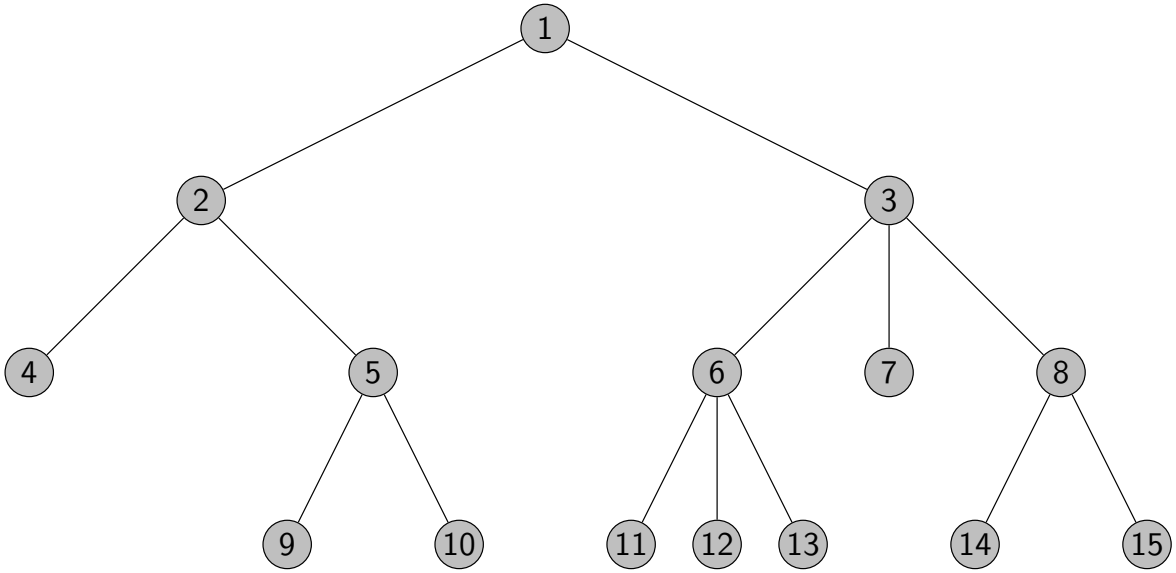
Breitensuche: Algorithmus

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

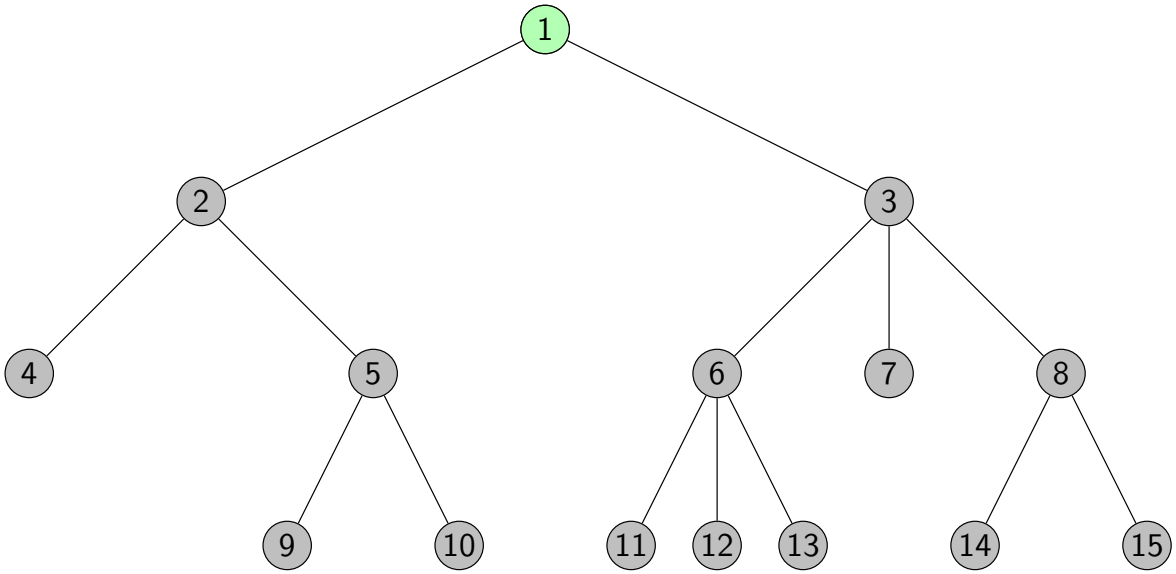
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



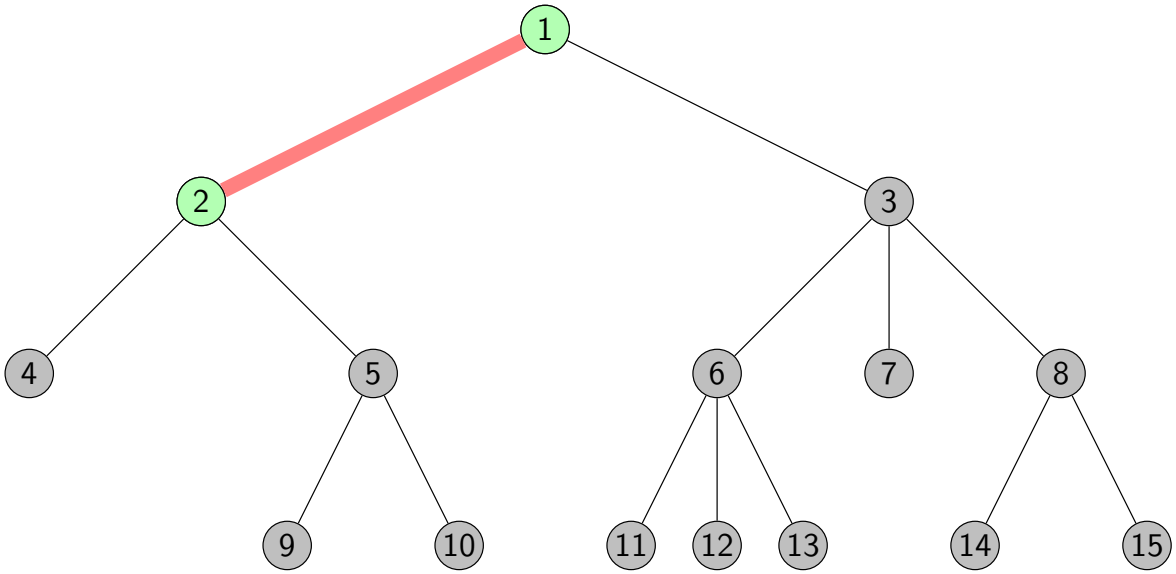
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



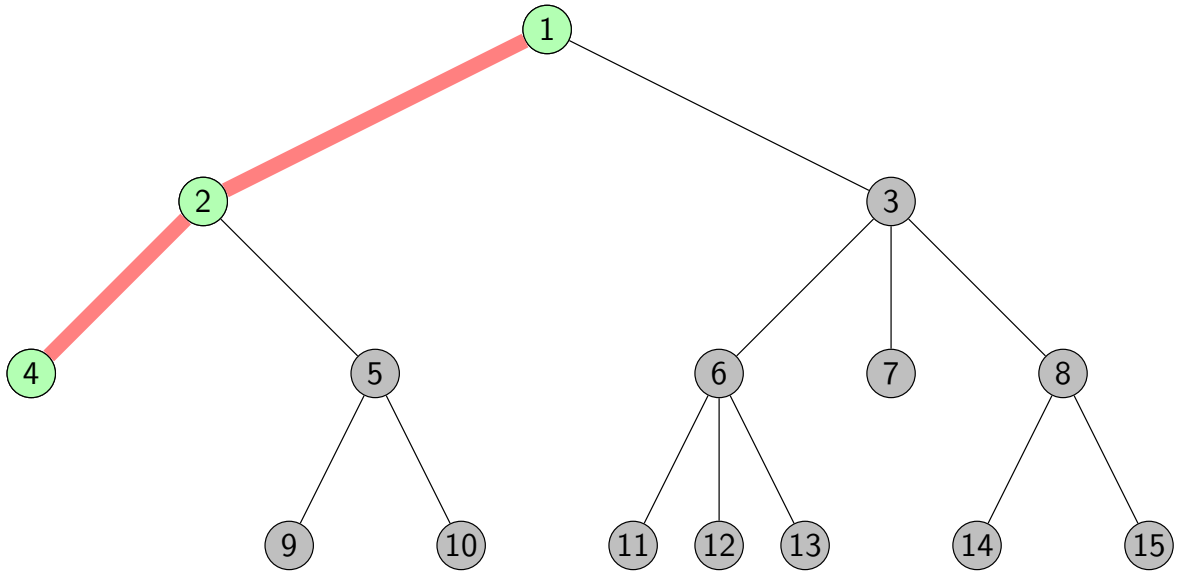
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



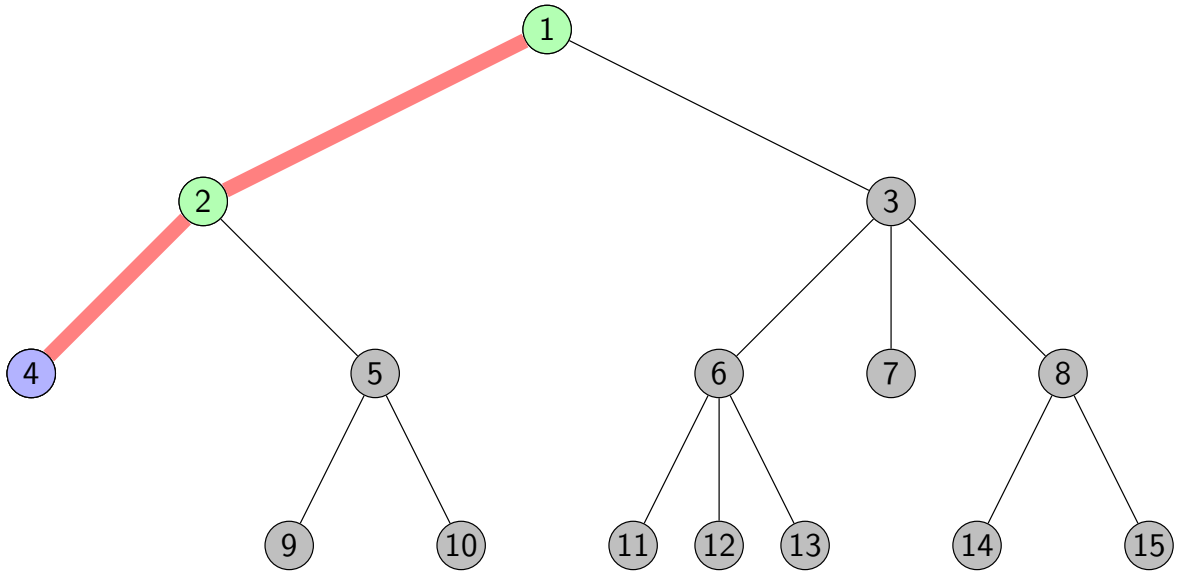
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



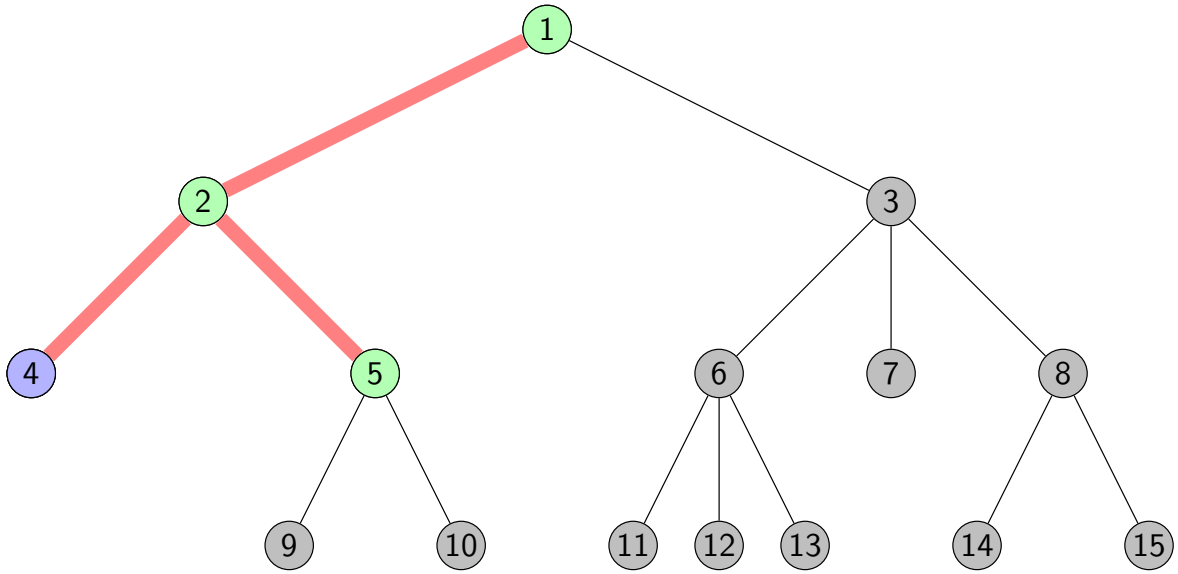
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



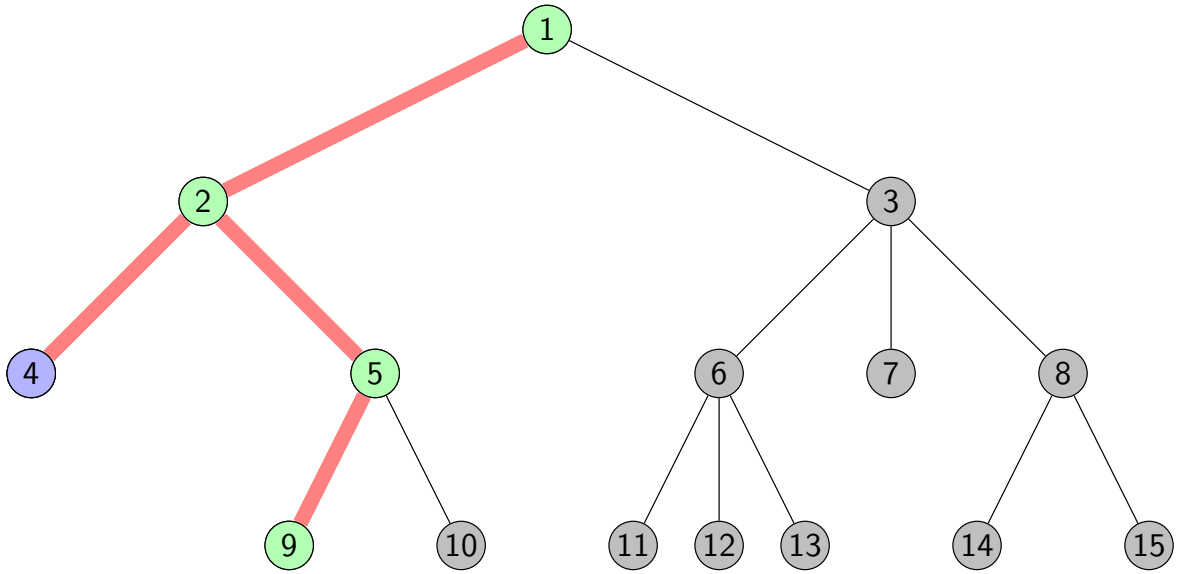
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



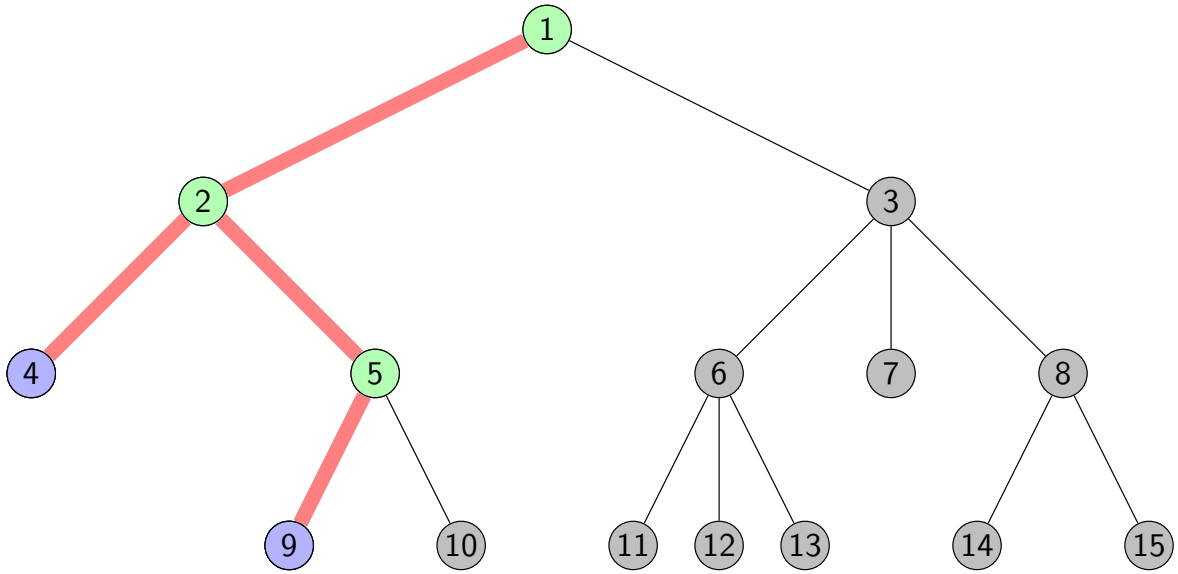
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



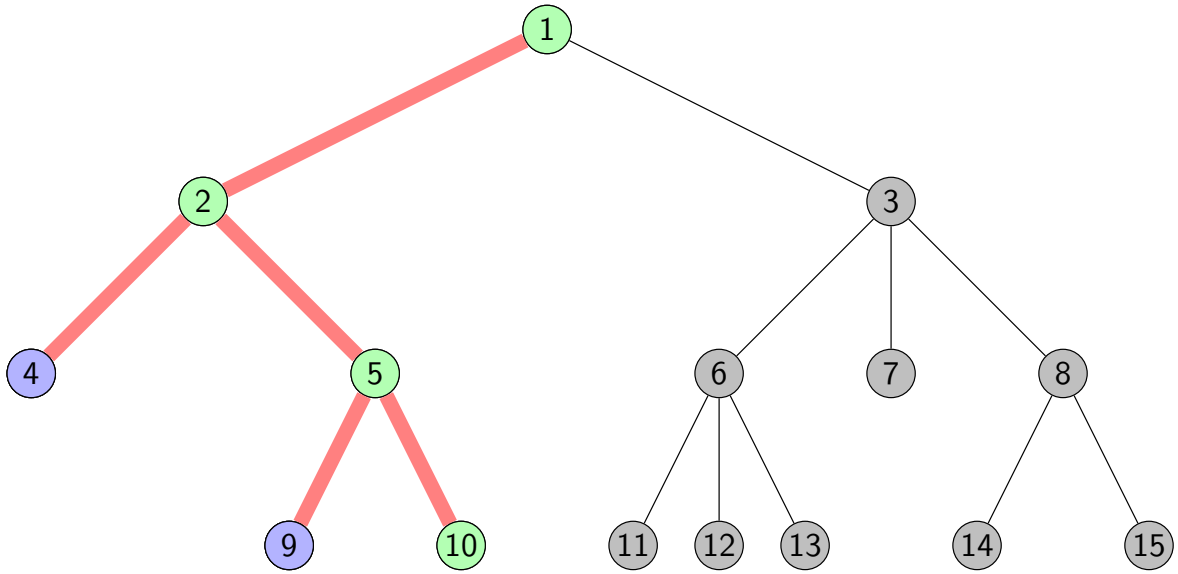
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



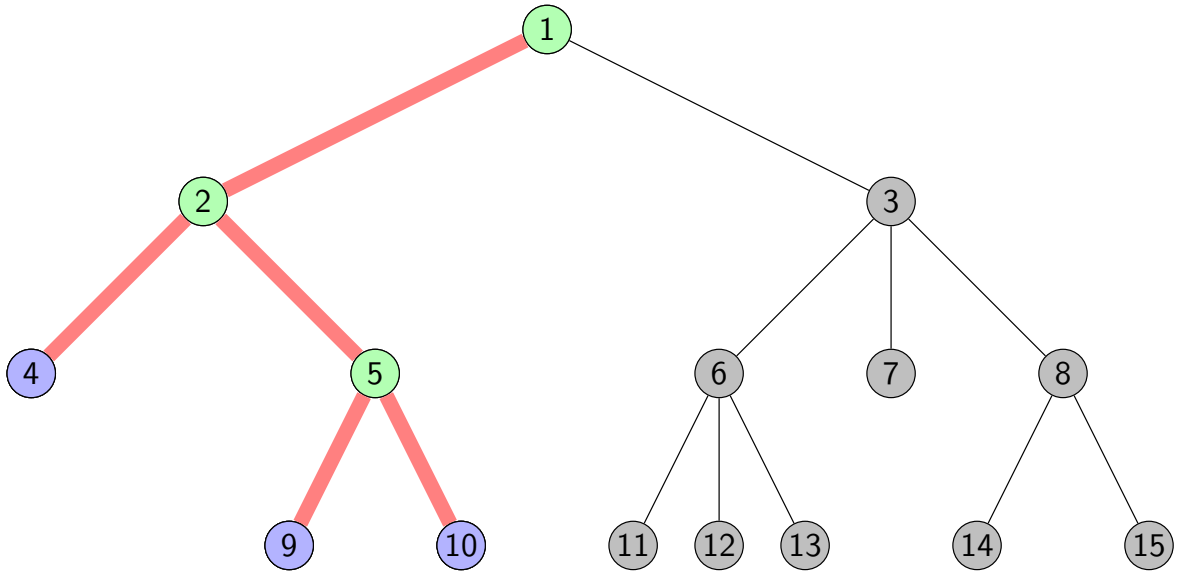
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



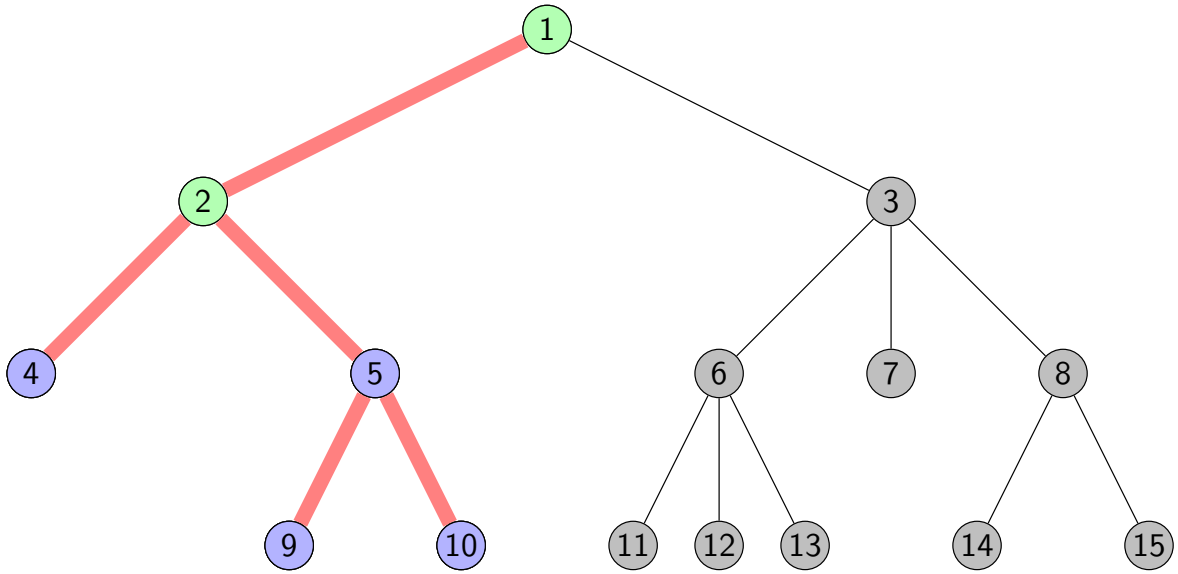
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



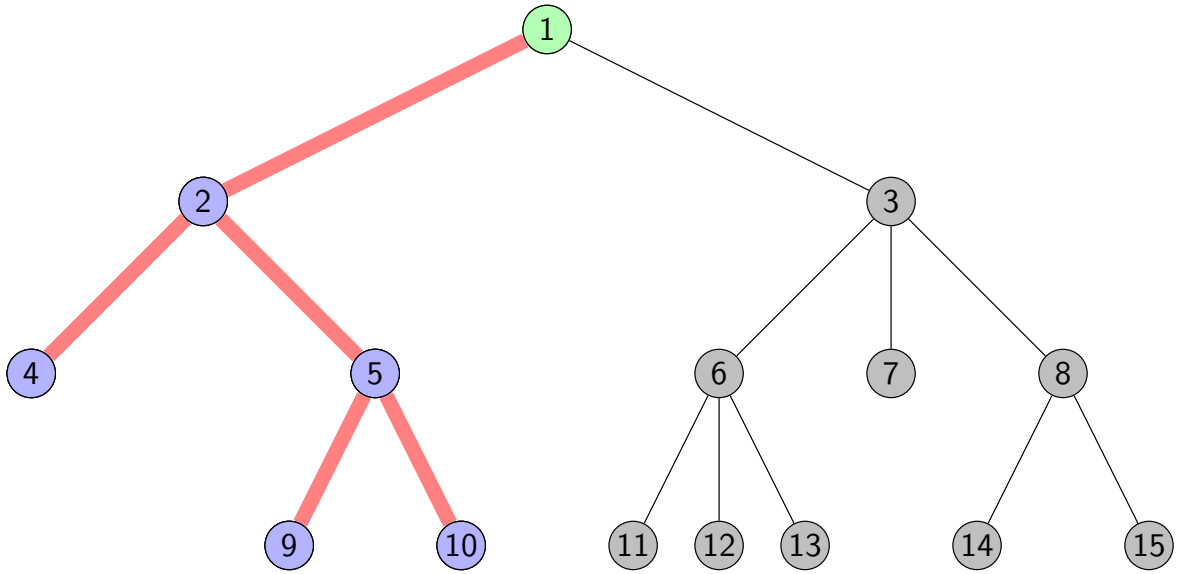
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



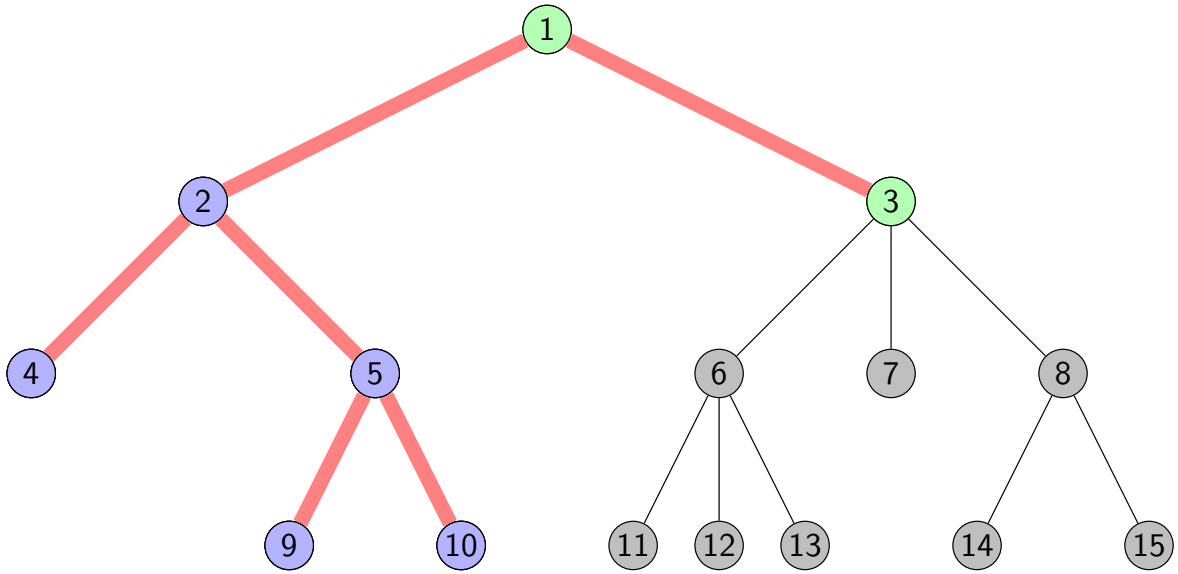
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



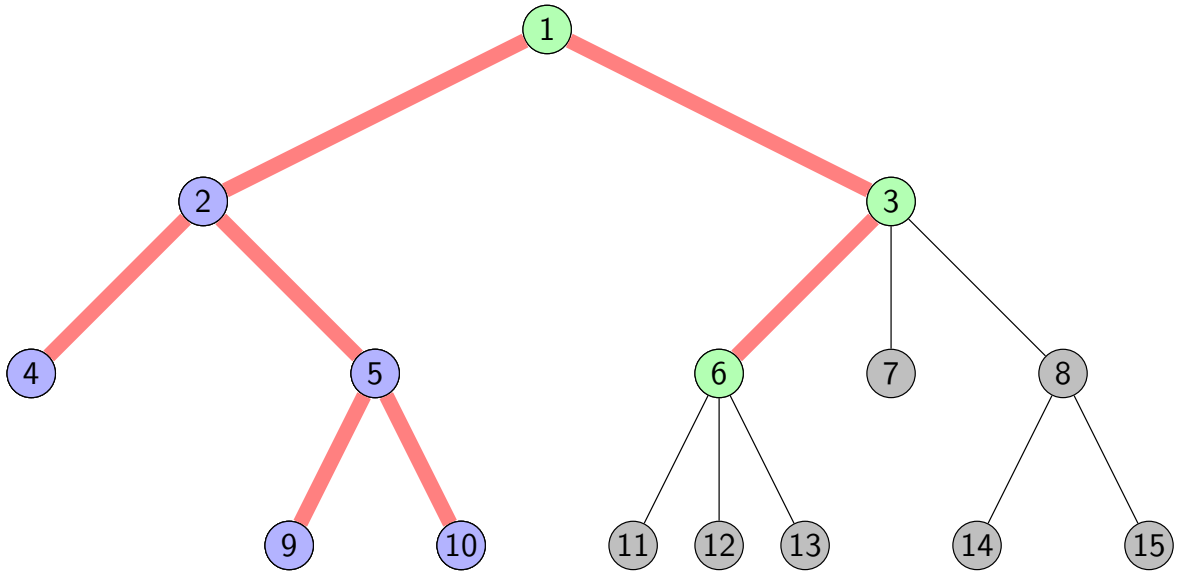
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



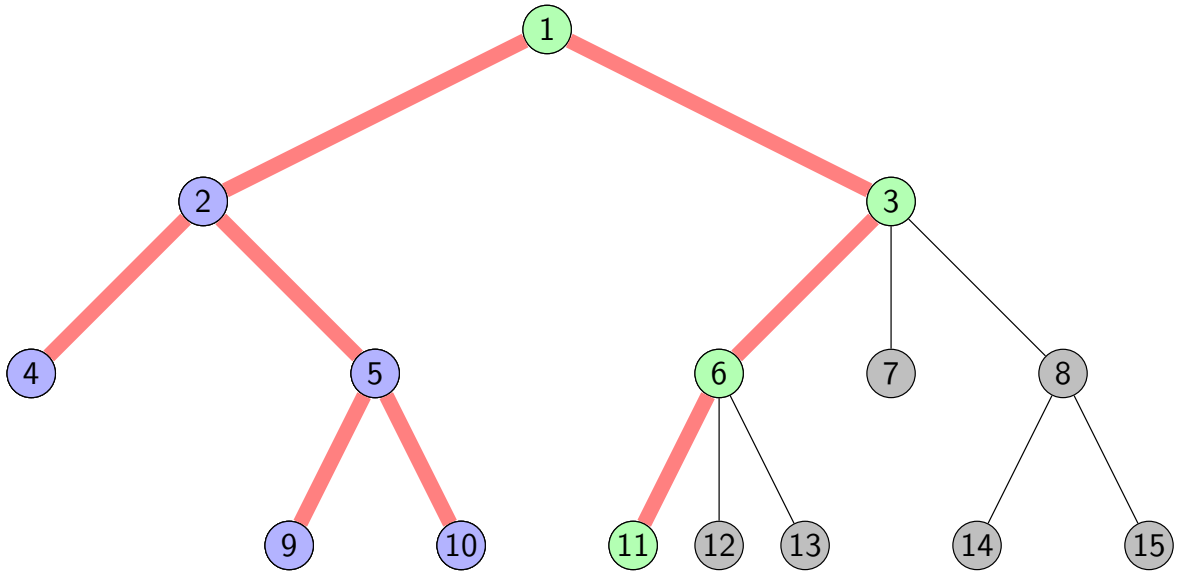
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



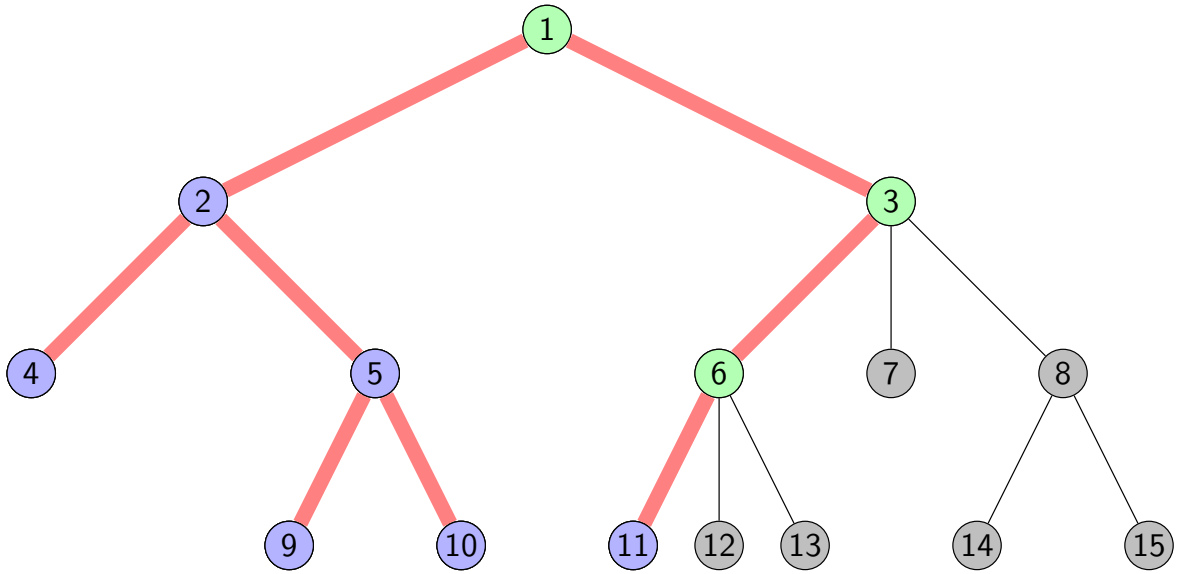
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



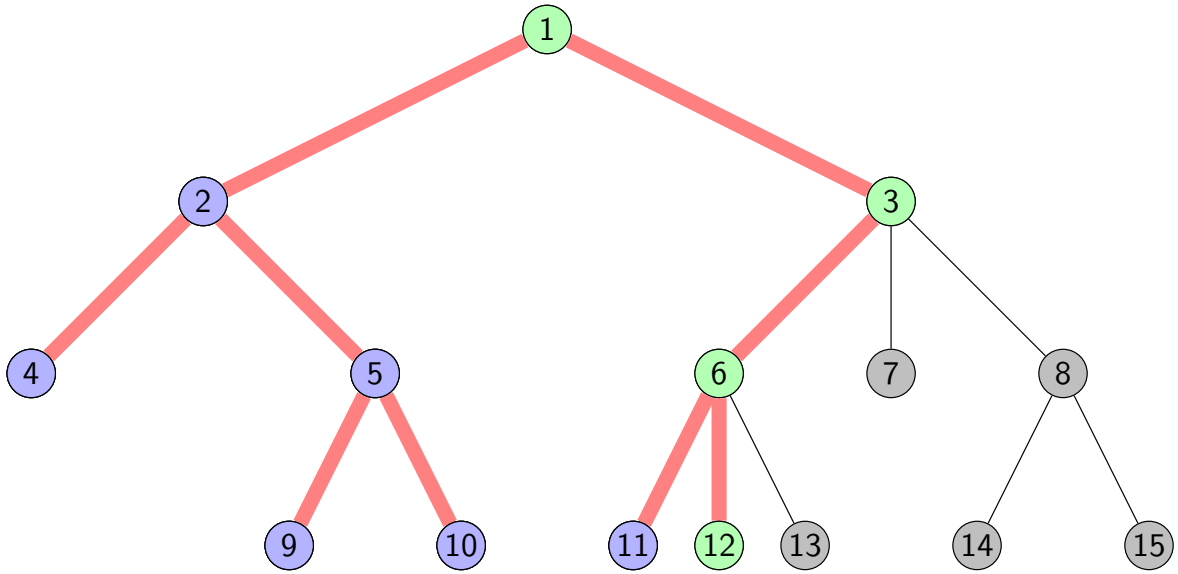
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



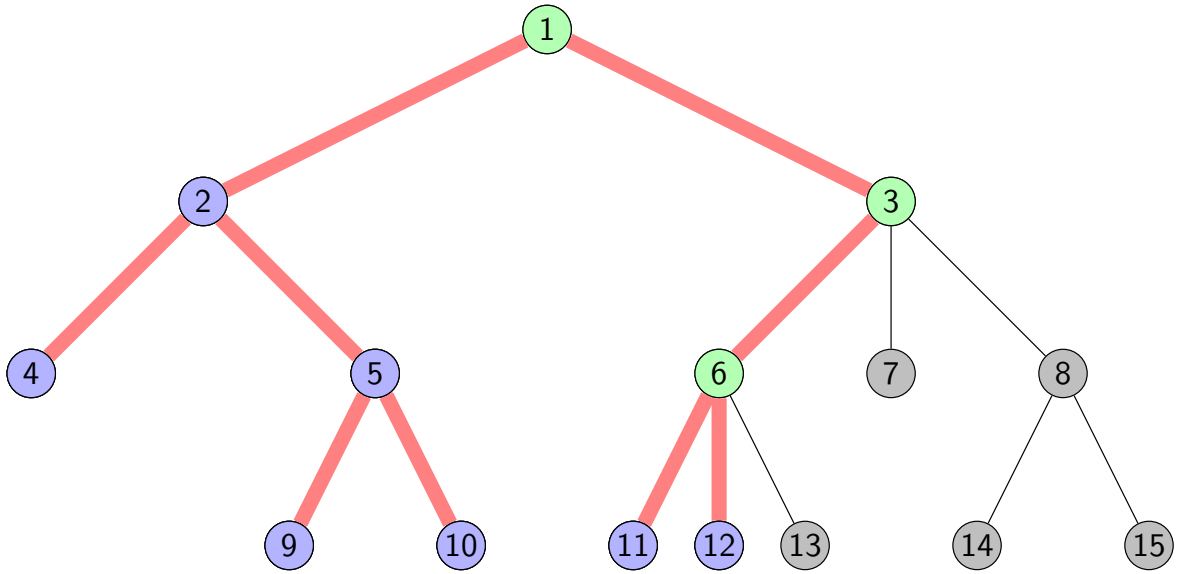
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



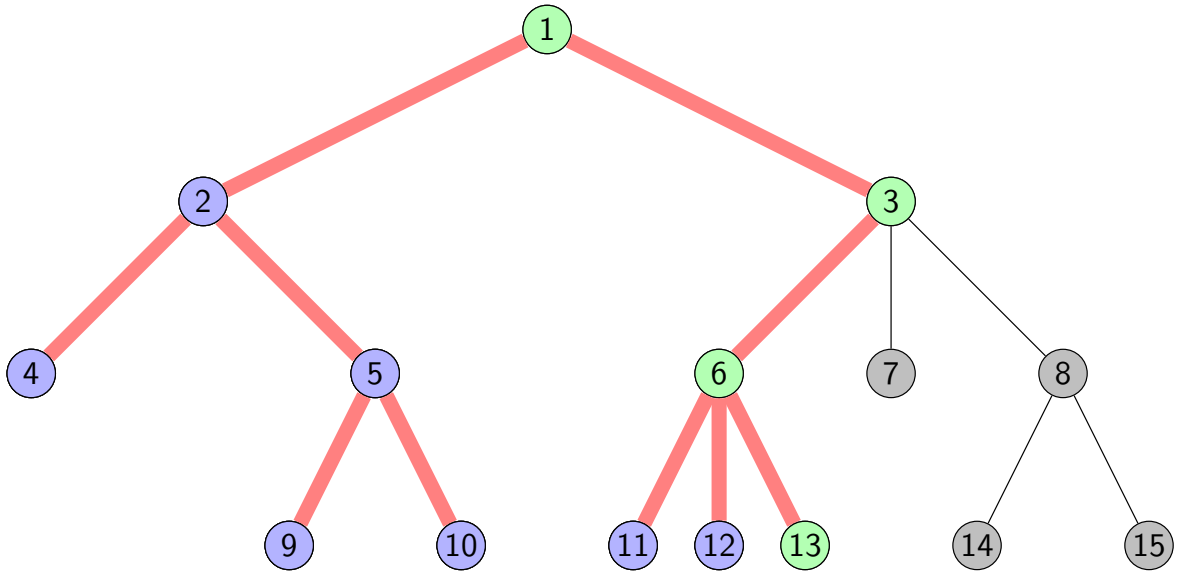
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



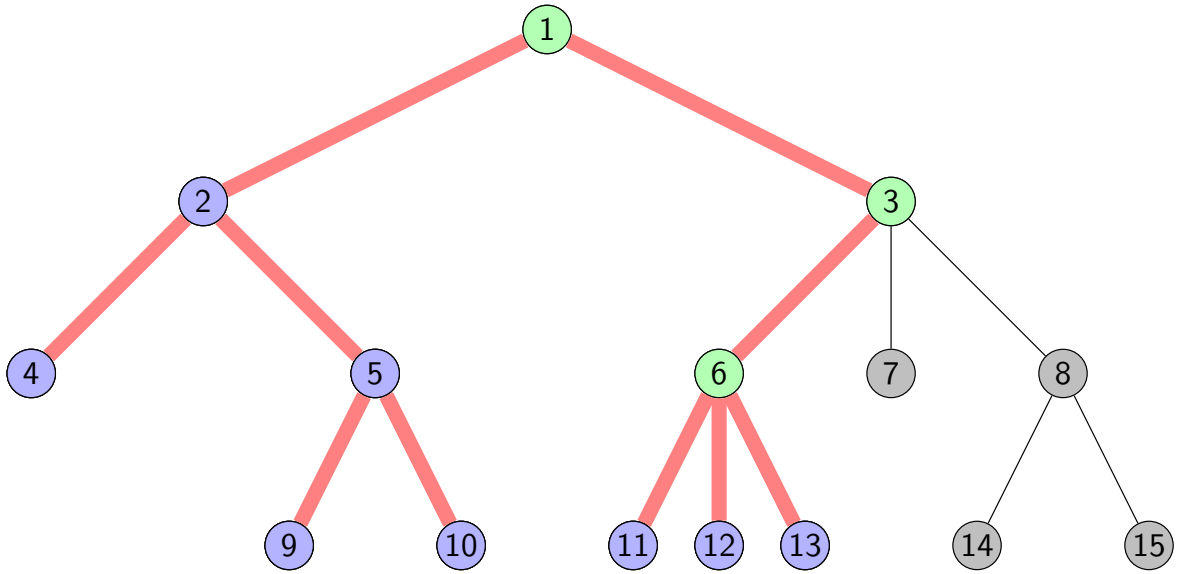
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



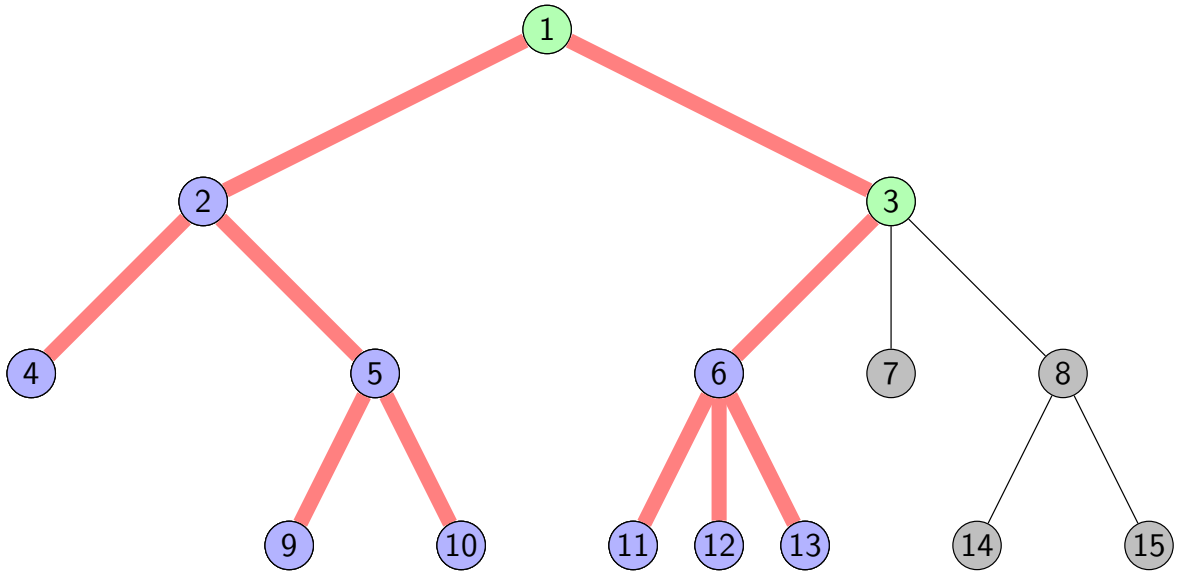
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



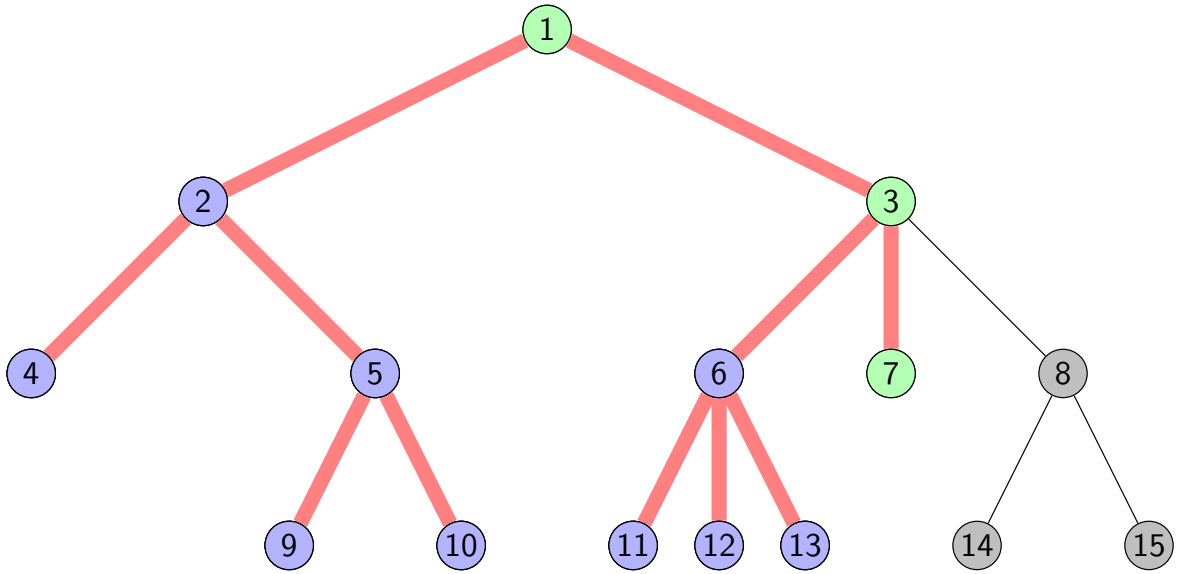
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



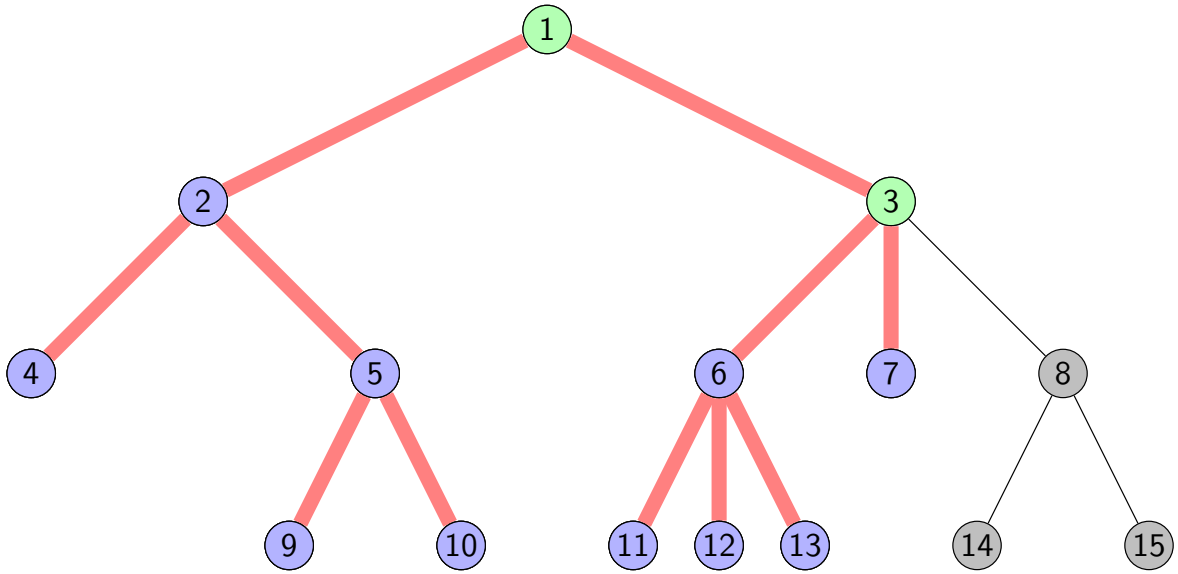
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



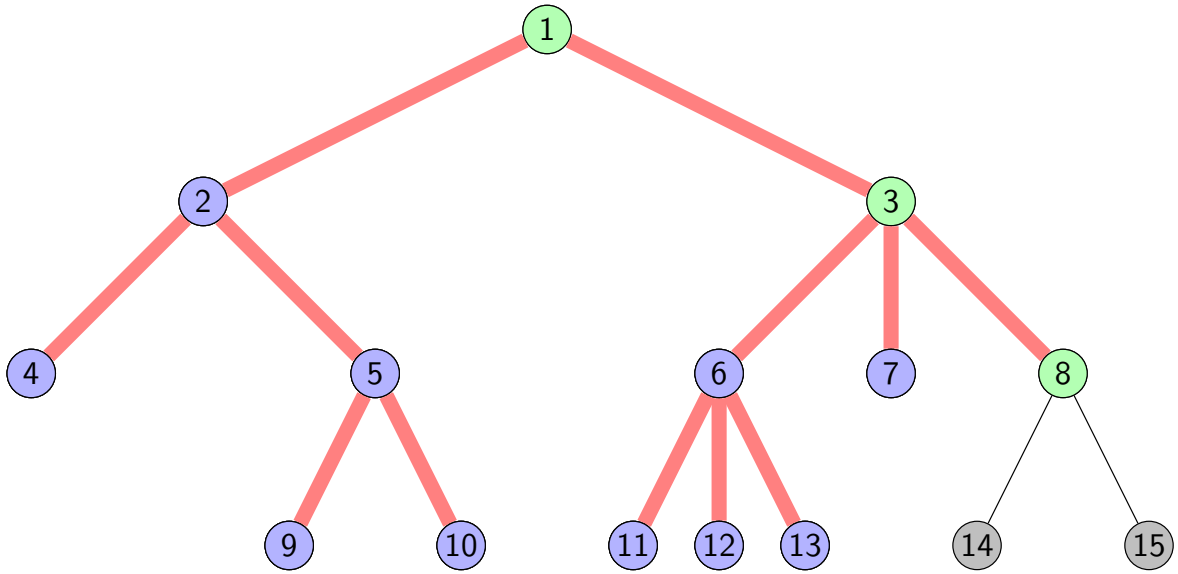
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



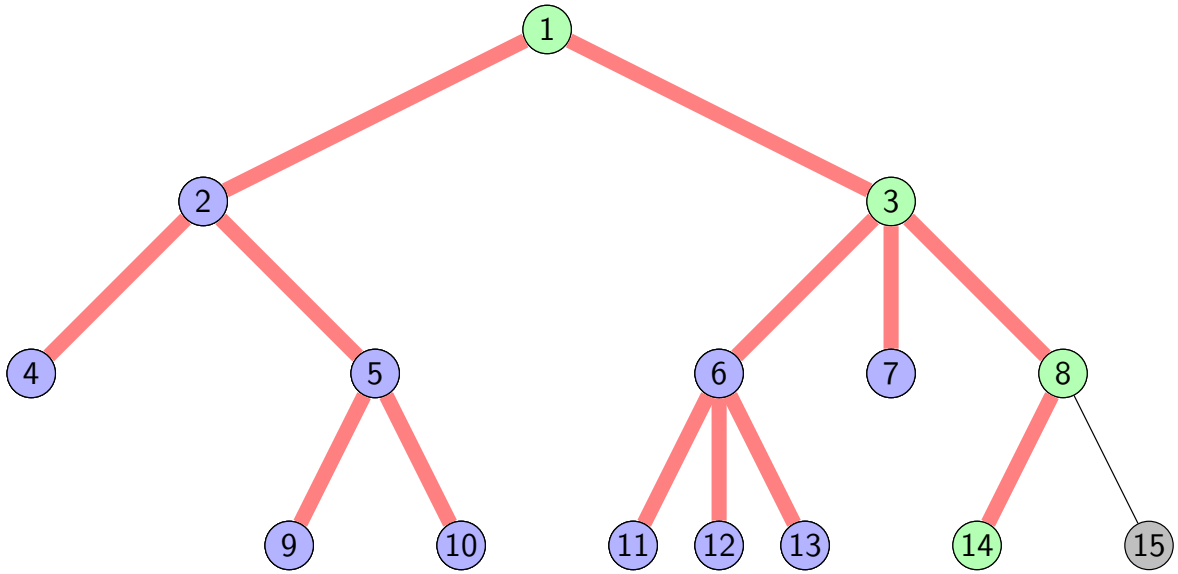
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



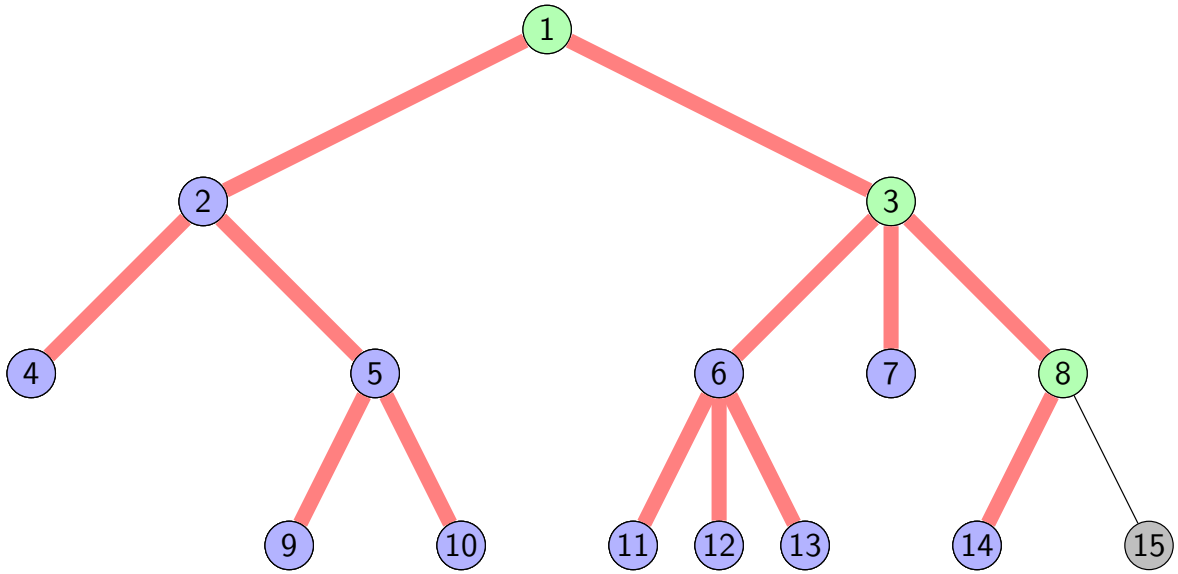
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



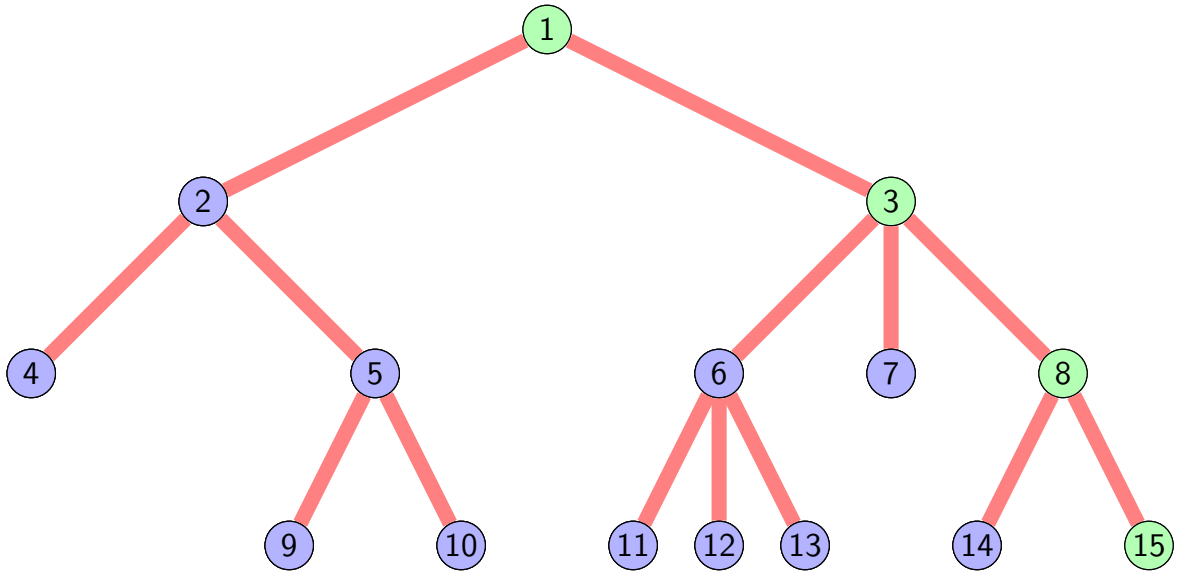
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



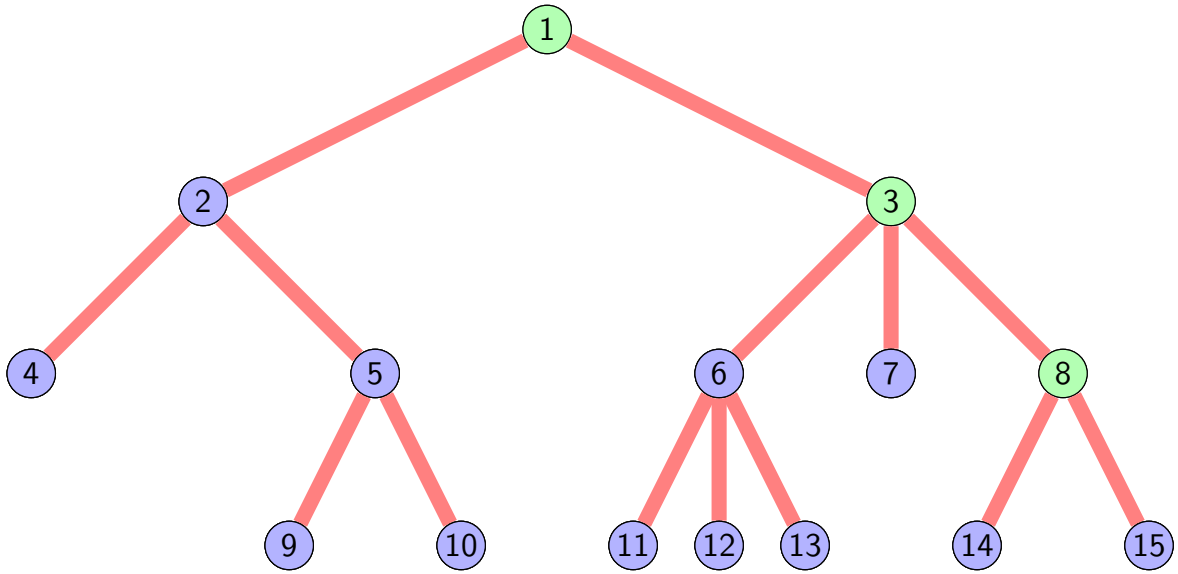
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



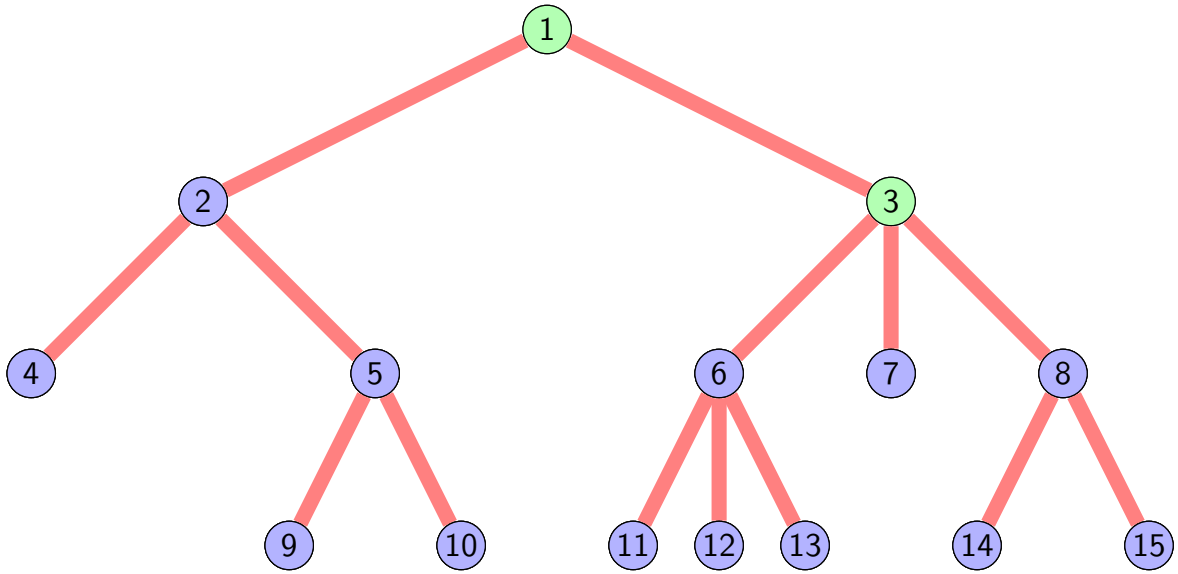
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



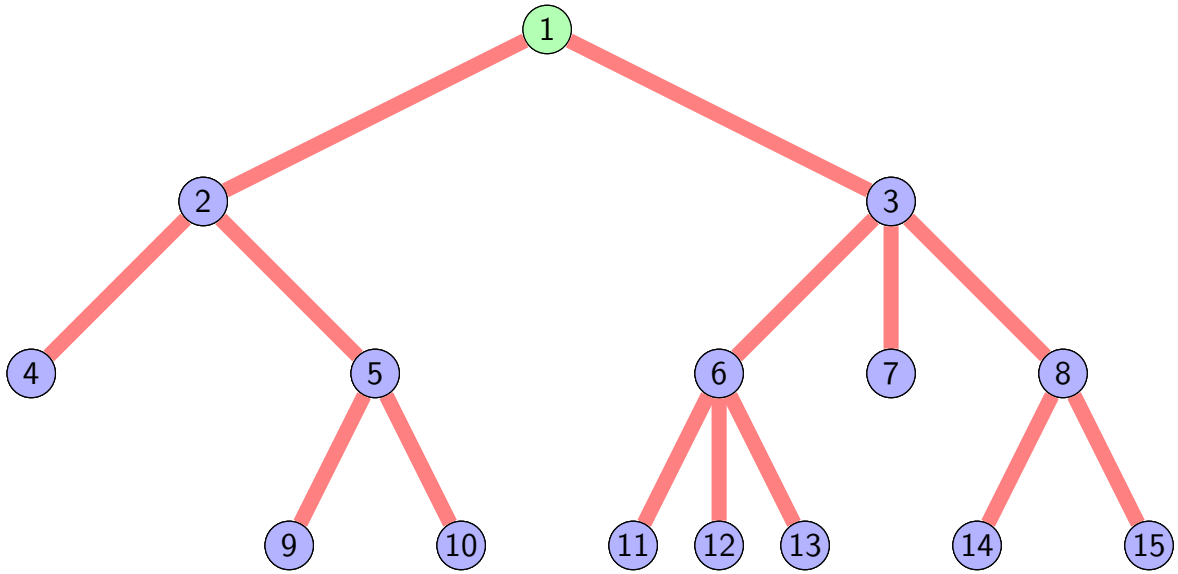
Tiefensuche

Beispiel: Beispiel einer Tiefensuche:



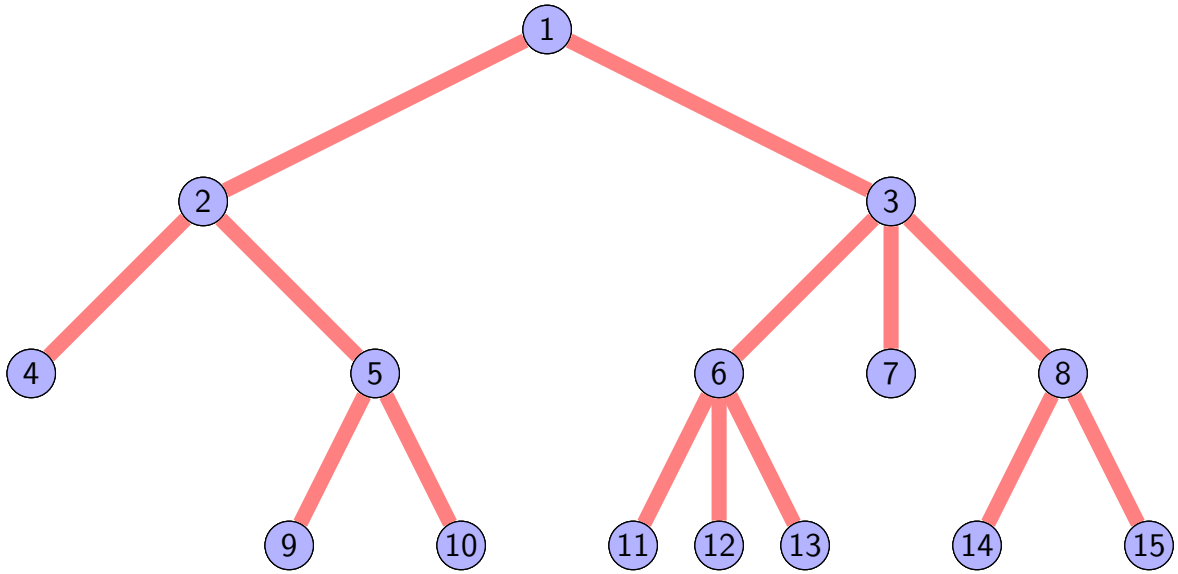
Tiefensuche

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 (*)

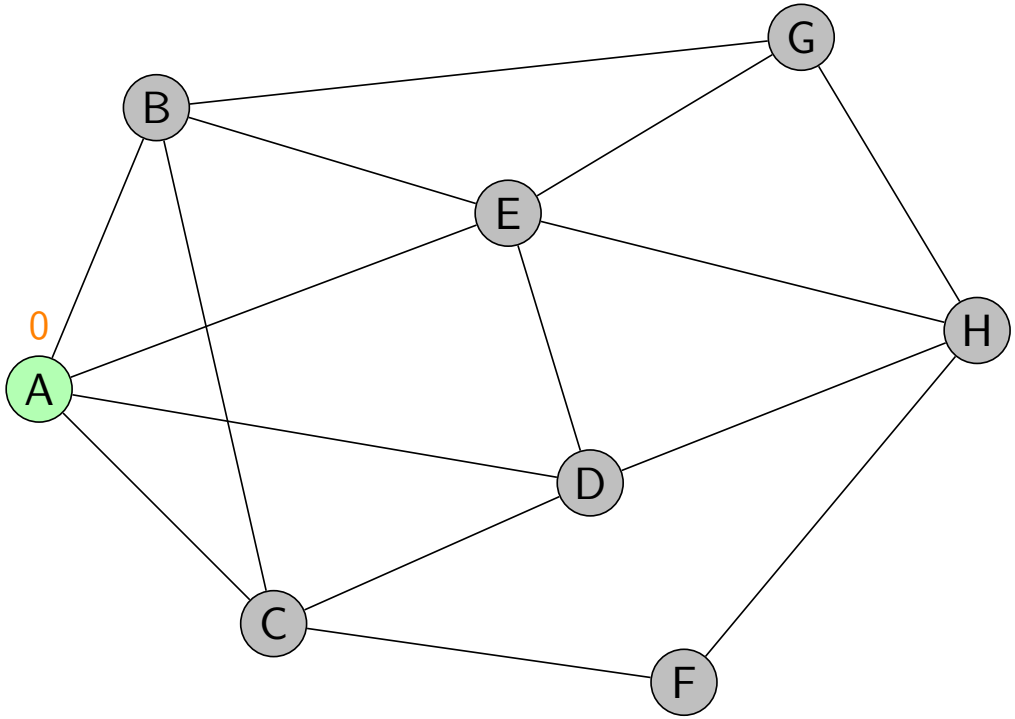
Algorithm 1: Tiefensuche

```
1  Function DFS(G = (V, E), Startknoten v, Gesuchter Knoten s)
       Result: vertex s ∈ V, if exists
2      Stack S;
3      markiere v als entdeckt;
4      S.push(v); // Lege v auf Stapel
5      while S not empty do
6          v = S.pop(); // nimm obersten Knoten vom Stapel
7          if v gesuchter Knoten s then
8              return v;
9          markiere v als abgeschlossen;
10         for all [v, u] ∈ E(G) do
11             if u noch nicht besucht then
12                 markiere u als entdeckt;
13                 S.push(u);
```

Anmerkungen

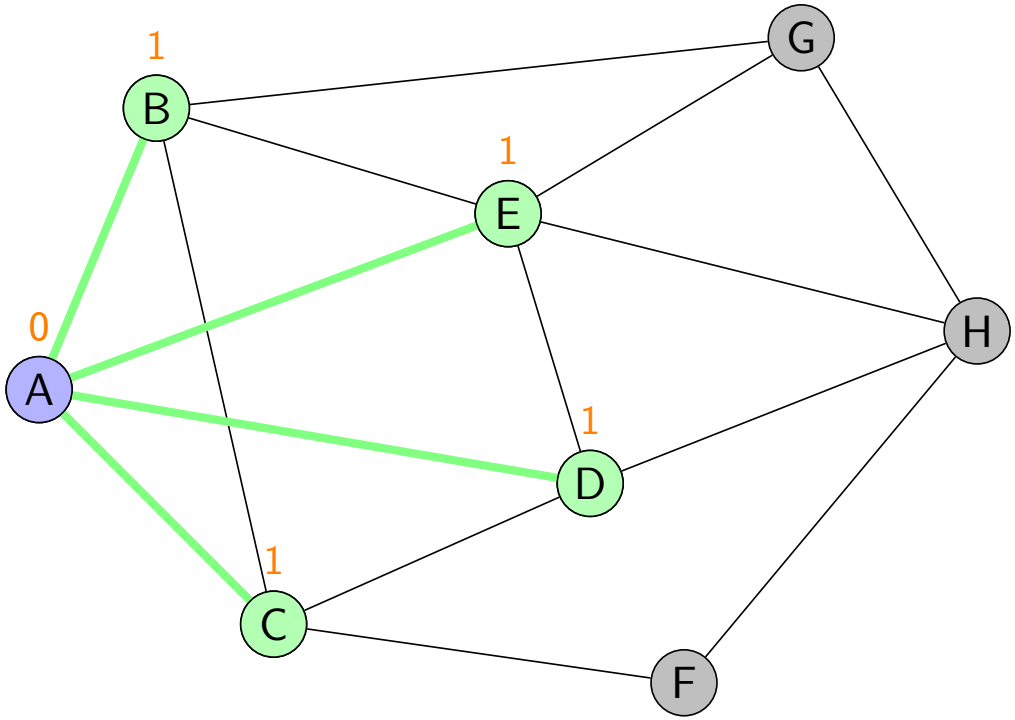
- Die Algorithmen können auch für allgemeine Graphen (und nicht nur Bäume) verwendet werden.
- Dabei werden schon besuchte Knoten *nicht* erneut besucht!
- Anwendungen BFS:
  - 2-färbbarkeit
  - Kürzester Pfad zwischen zwei Knoten
  - Kürzeste-Kreise-Problem
- Anwendungen DFS:
  - Test auf Kreisfreiheit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

Breitensuche auf allgemeinen Graphen



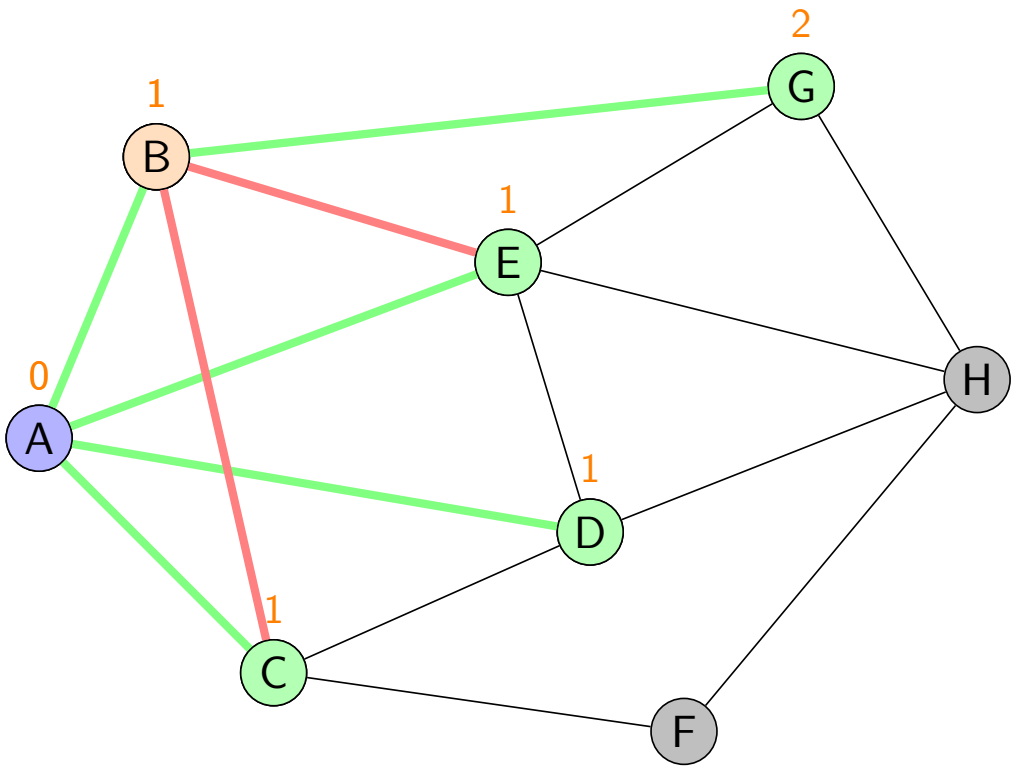
Start mit Knoten A

Breitensuche auf allgemeinen Graphen



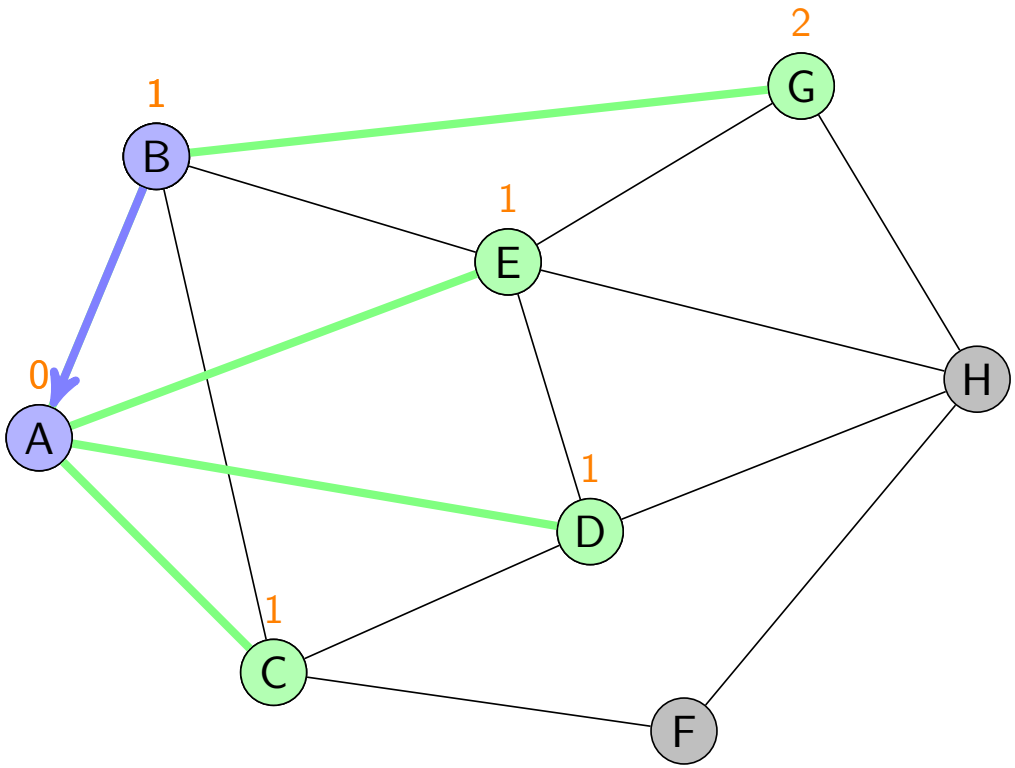
B, C, D, E werden entdeckt (Distanz jeweils 1), A fertiggestellt

Breitensuche auf allgemeinen Graphen



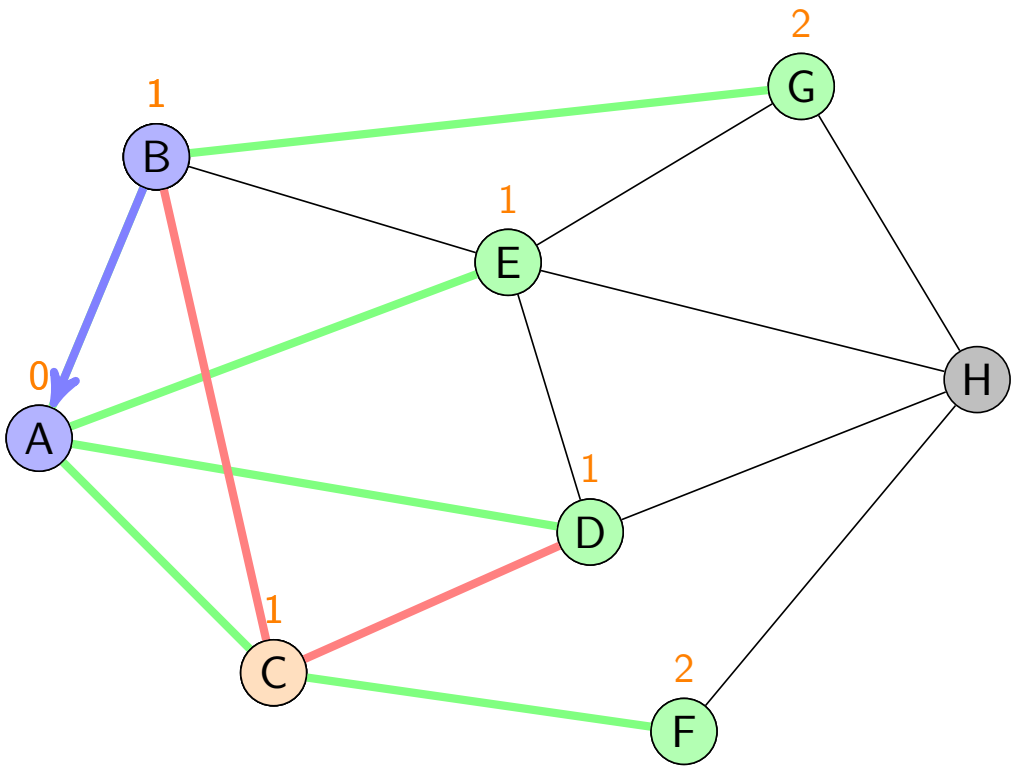
Fortsetzung bei B: [B, E] wird nicht betrachtet, da E schon besucht; G wird entdeckt (Distanz 2)

Breitensuche auf allgemeinen Graphen



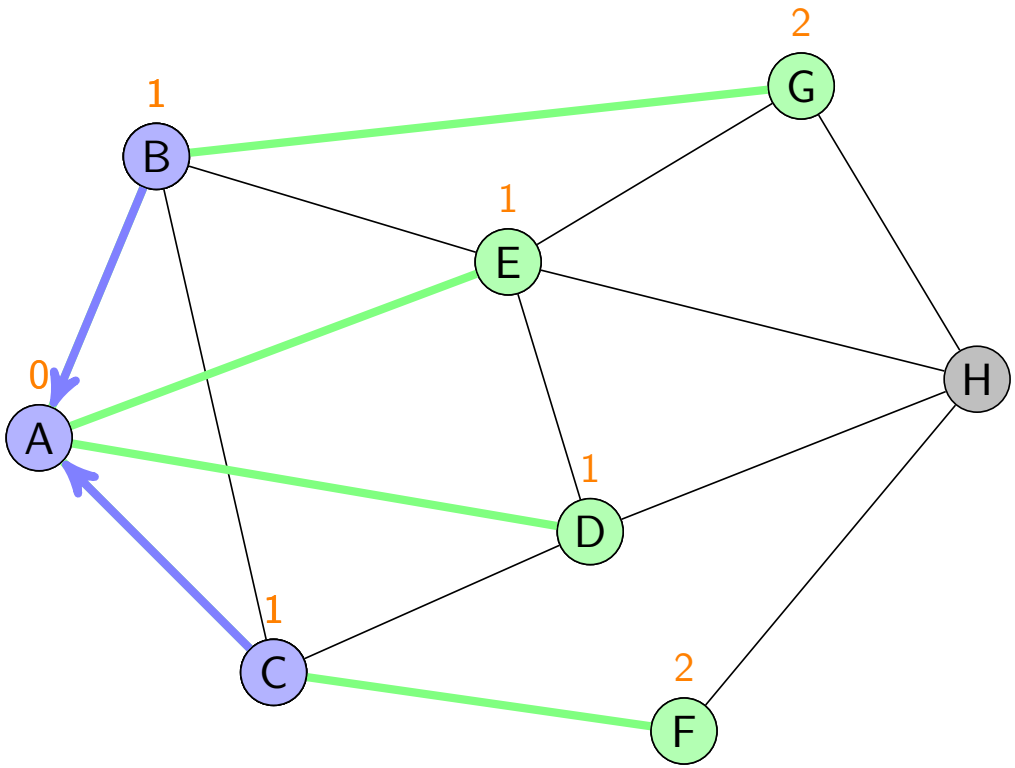
B wird fertiggestellt, und Verweis auf Vorgänger A gespeichert.

Breitensuche auf allgemeinen Graphen



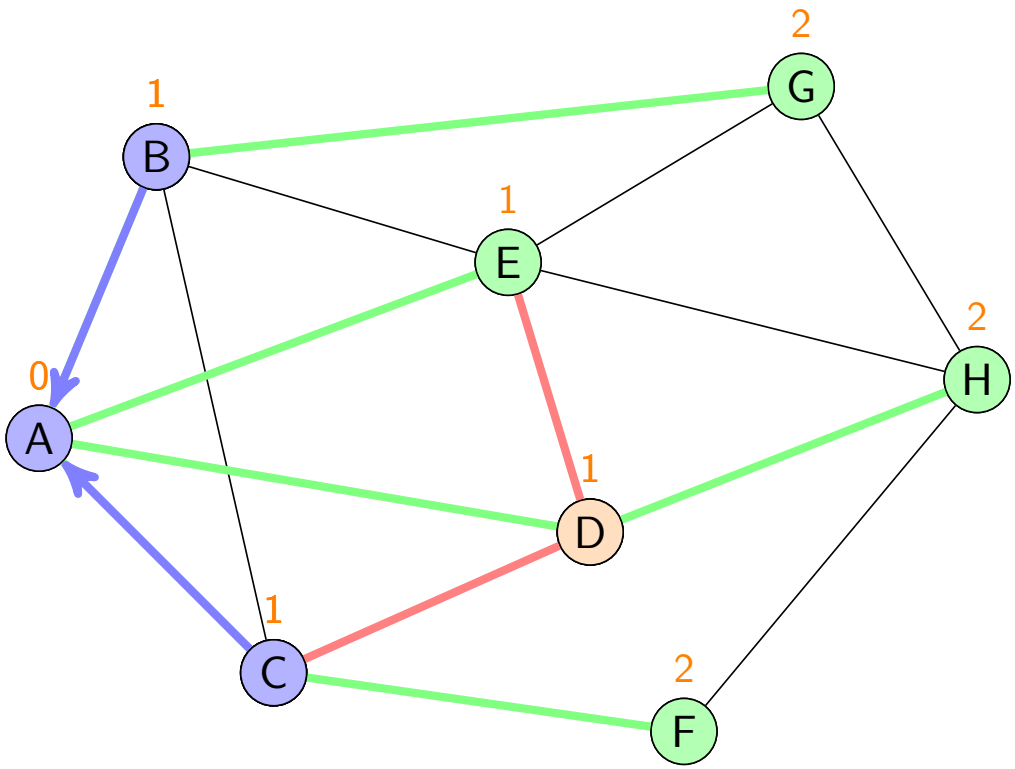
Fortsetzung bei C: [C, D] wird nicht betrachtet, da D schon besucht. F wird entdeckt (Distanz 2)

Breitensuche auf allgemeinen Graphen



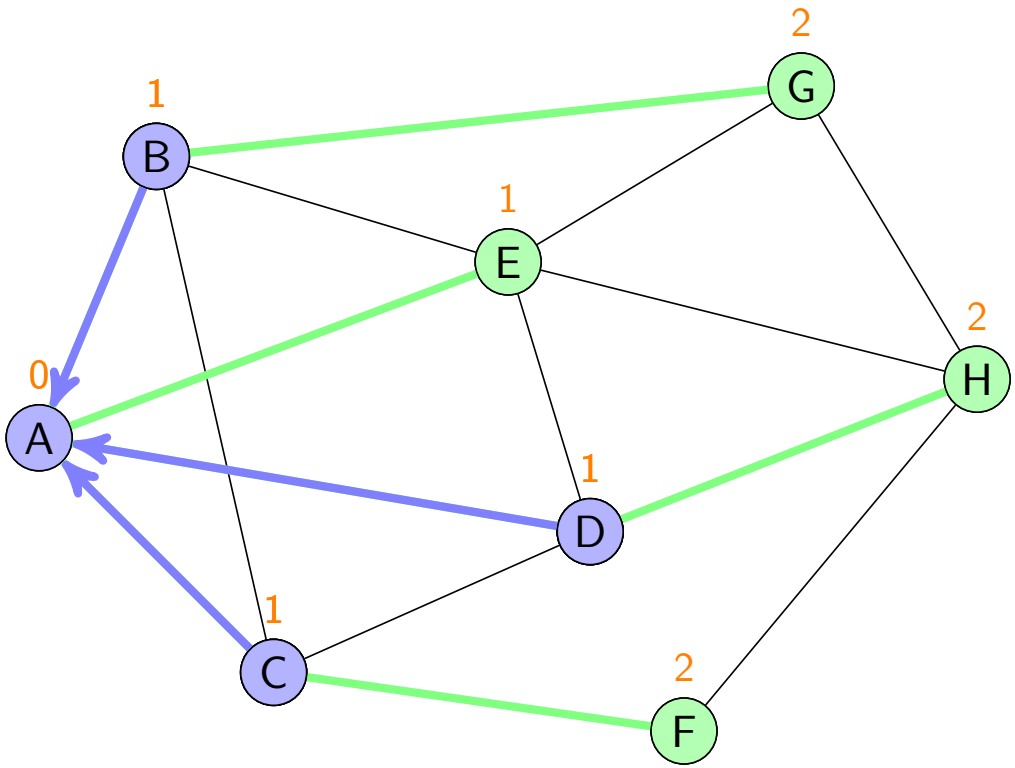
C wird fertiggestellt, Verweis auf Vorgänger A gespeichert.

Breitensuche auf allgemeinen Graphen



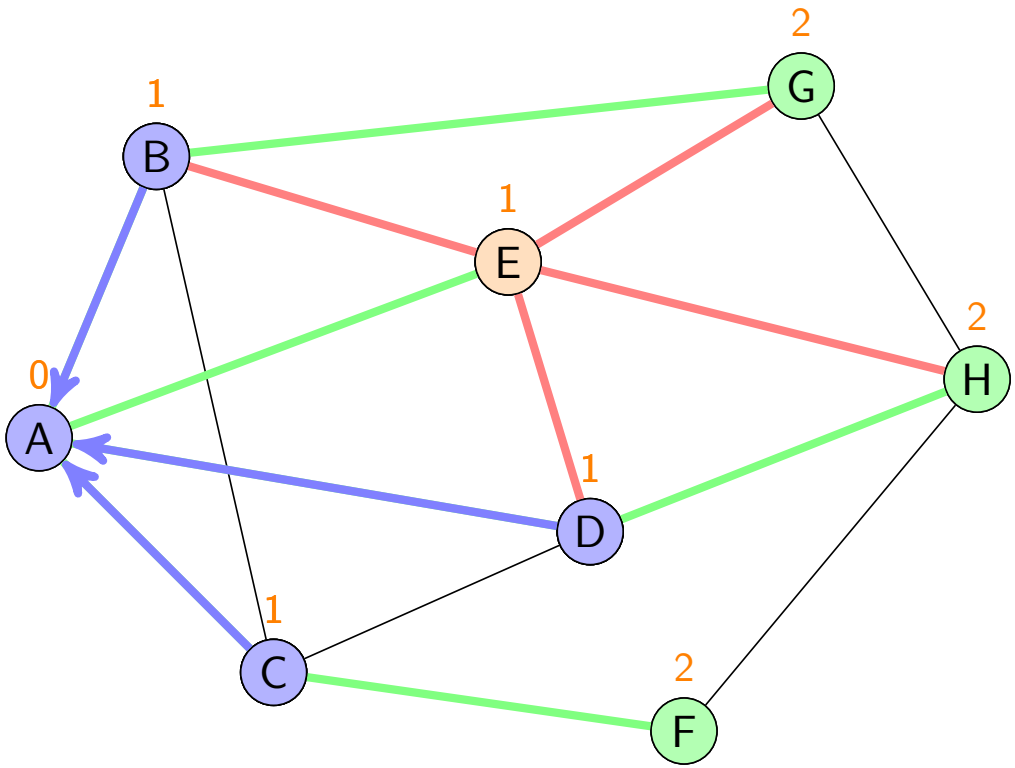
Fortsetzung bei D. Kanten zu C und E werden nicht betrachtet (da jeweils schon besucht). H wird entdeckt (Distanz 2)

Breitensuche auf allgemeinen Graphen



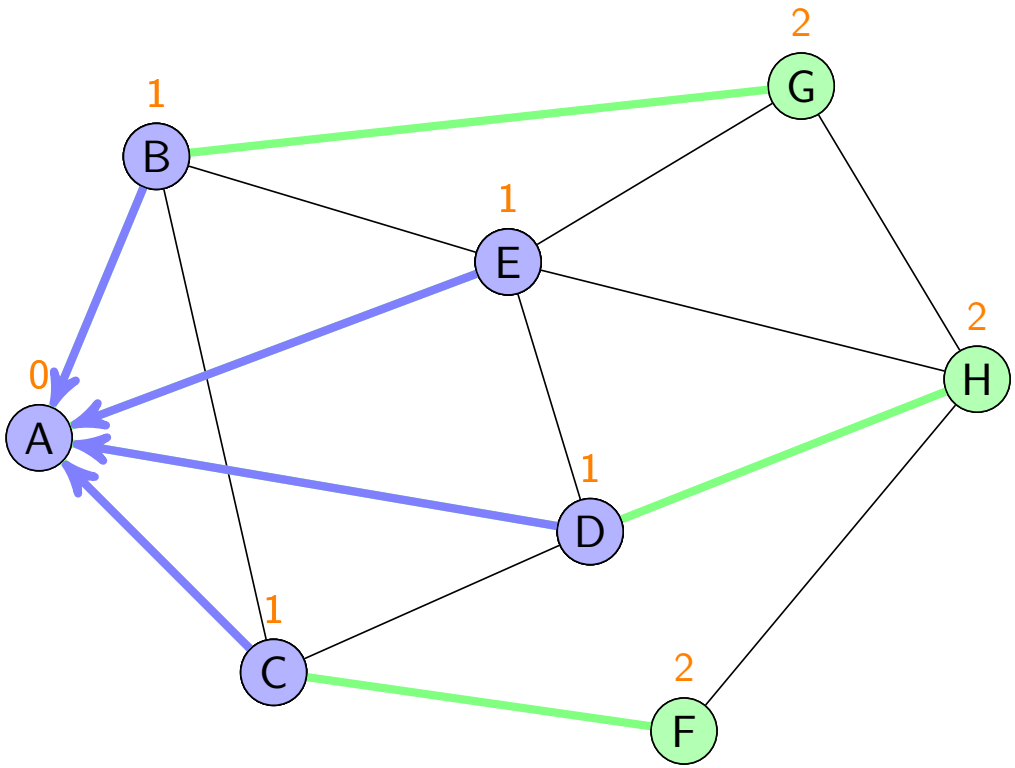
Fertigstellung von D, Verweis auf Vorgänger A

Breitensuche auf allgemeinen Graphen



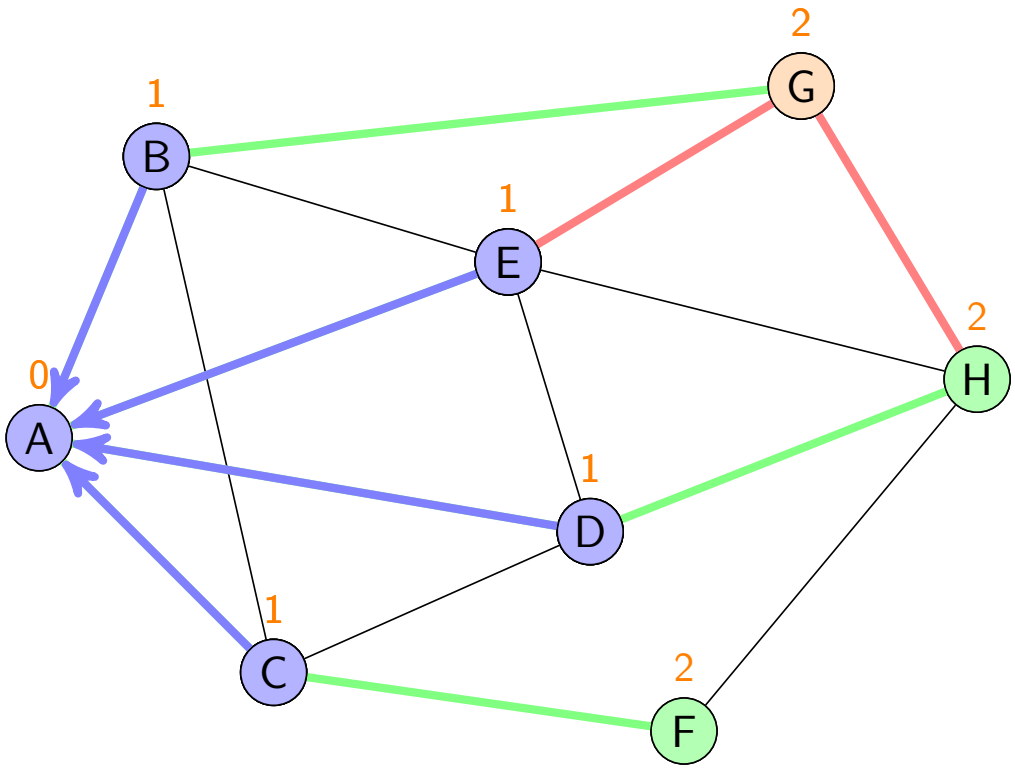
Fortsetzung mit E, jedoch werden keine neuen Knoten entdeckt.

Breitensuche auf allgemeinen Graphen



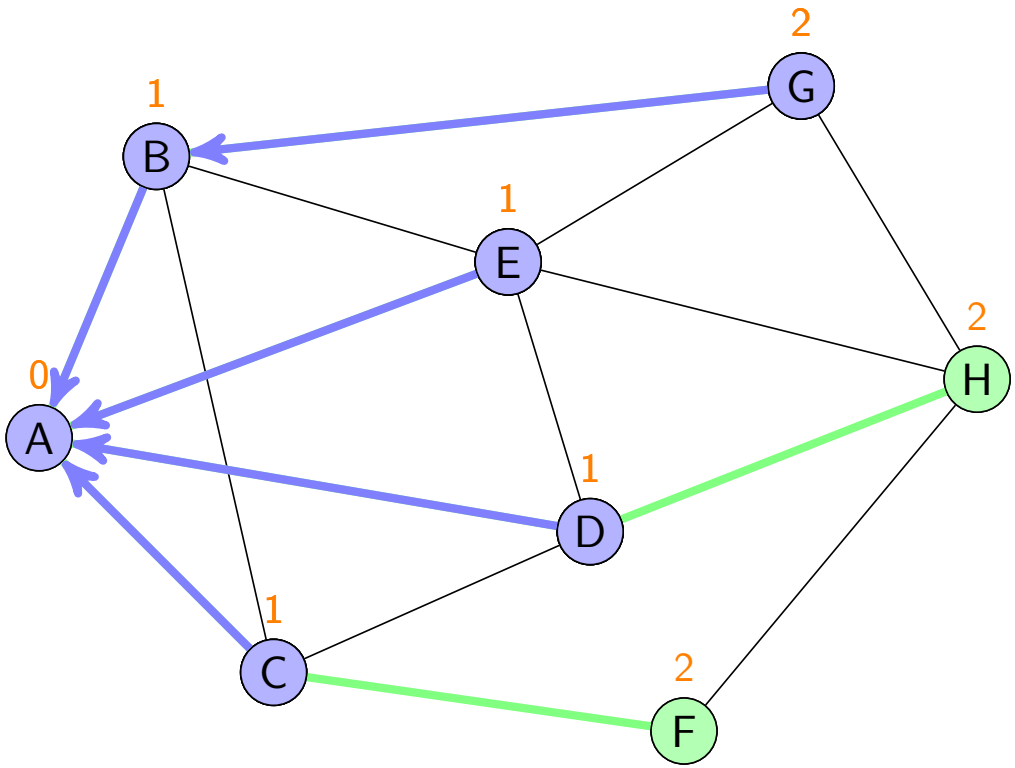
Fertigstellung von E

Breitensuche auf allgemeinen Graphen



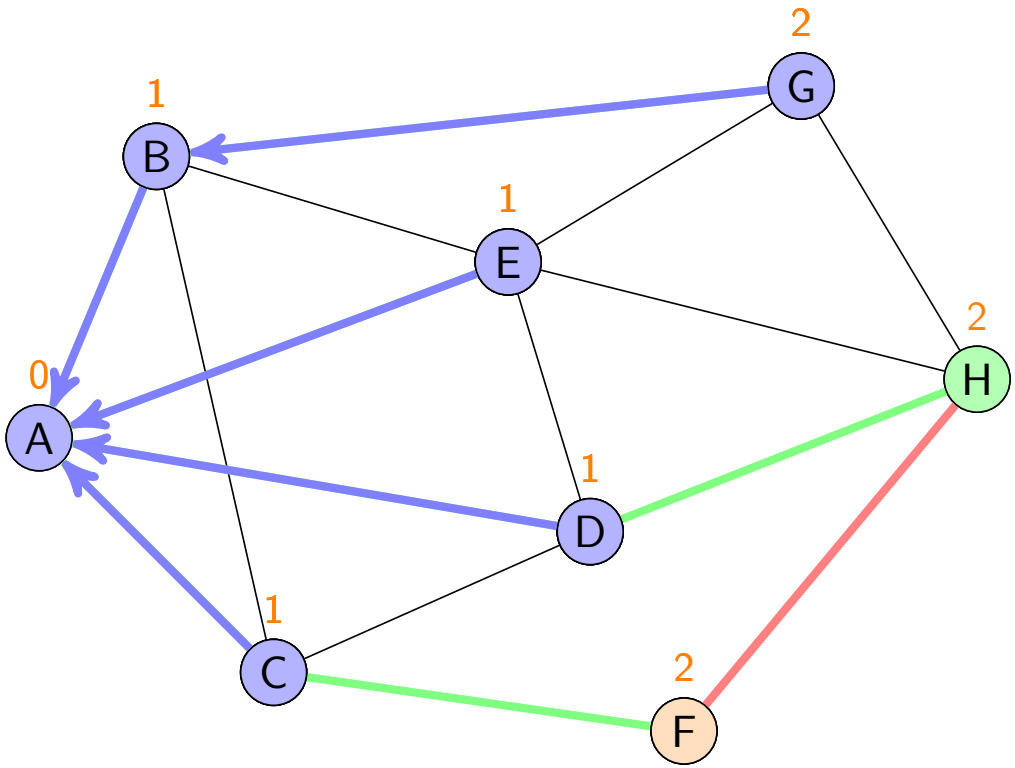
Fortsetzung mit G, jedoch werden auch hier keine neuen Knoten entdeckt.

Breitensuche auf allgemeinen Graphen



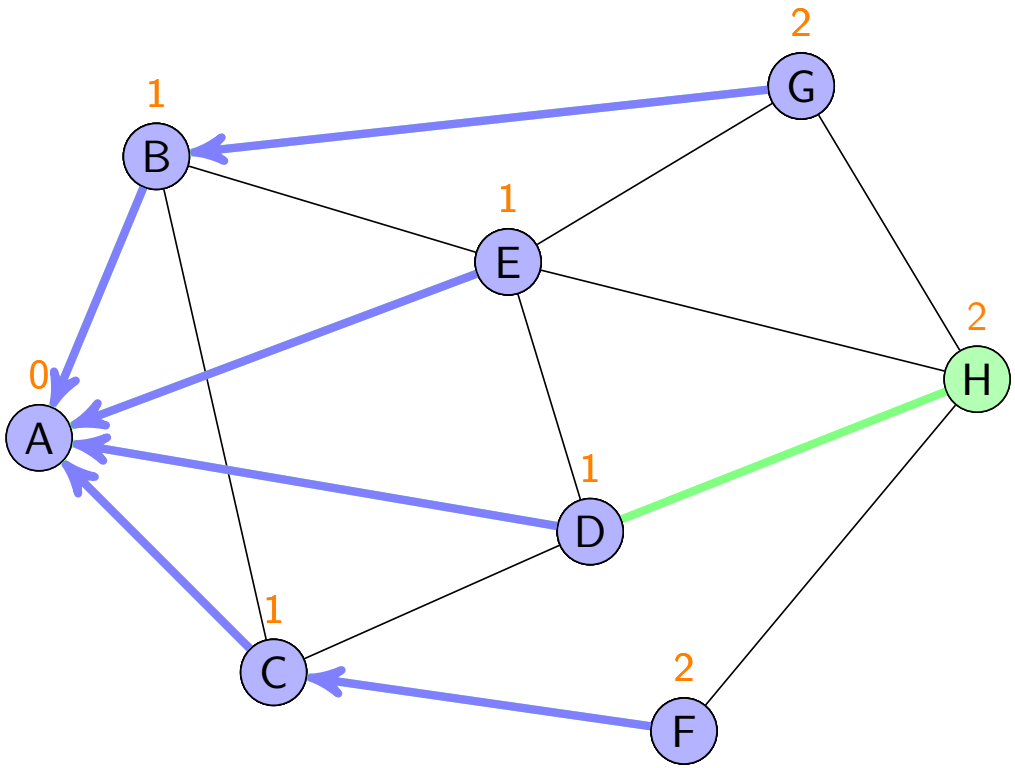
Fertigstellung von G

Breitensuche auf allgemeinen Graphen



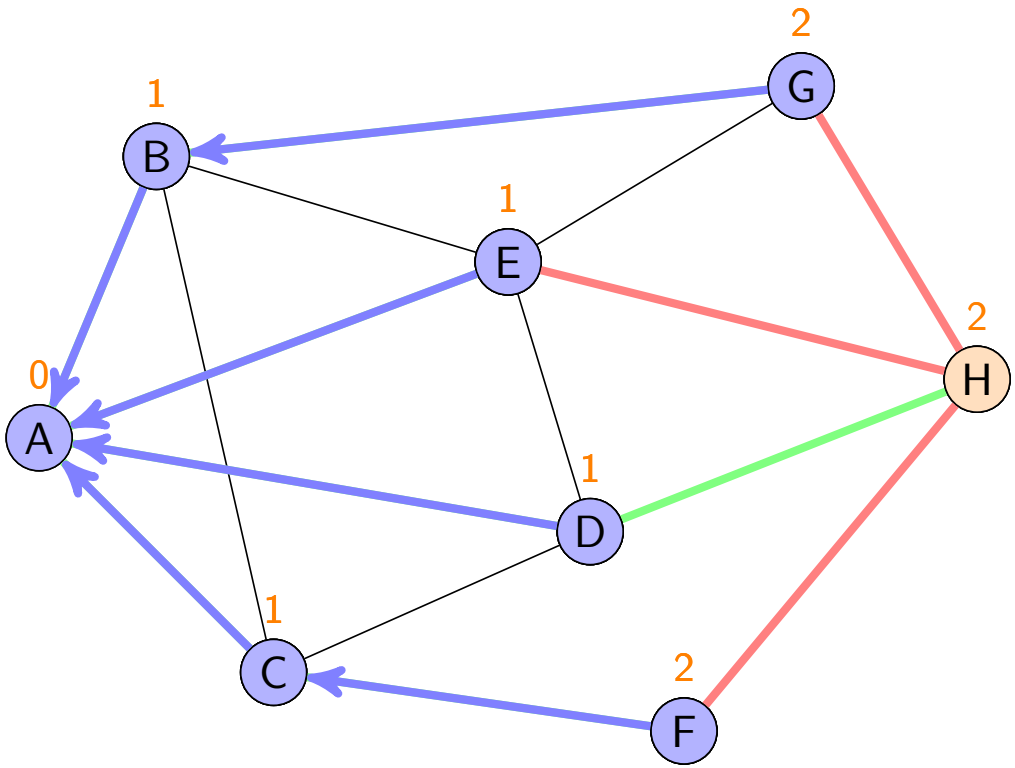
Fortsetzung mit F

Breitensuche auf allgemeinen Graphen



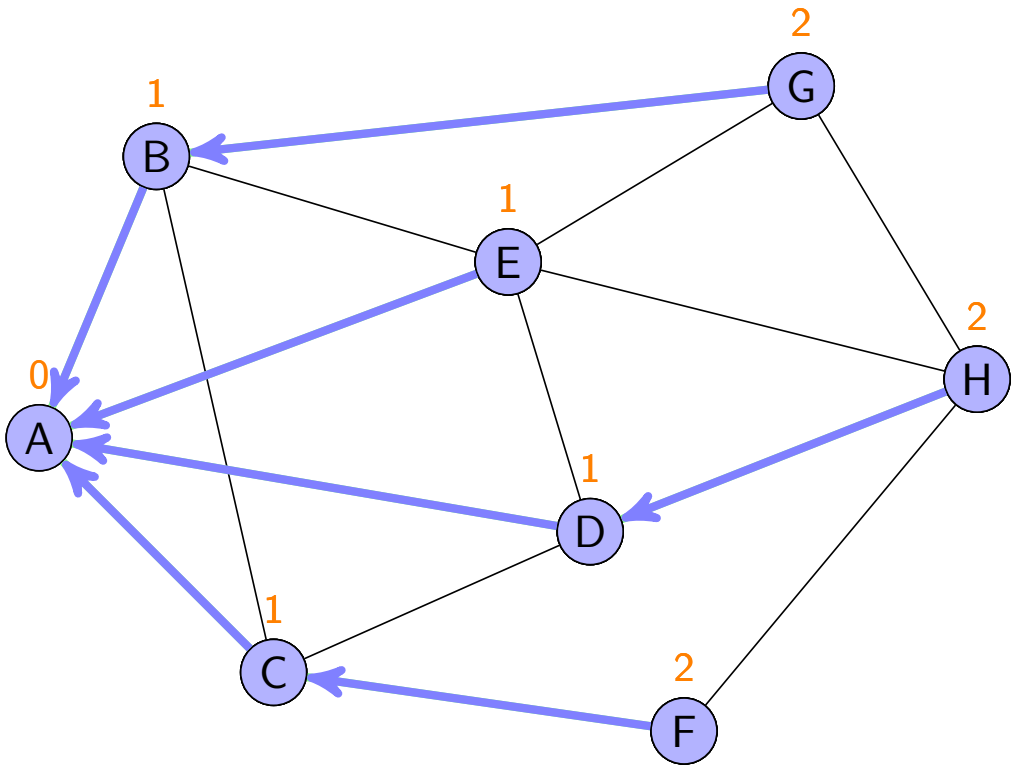
F wird fertiggestellt.

Breitensuche auf allgemeinen Graphen



Fortsetzung mit H

Breitensuche auf allgemeinen Graphen



H wird fertiggestellt.

Breitensuche auf allgemeinen Graphen

- Es entsteht ein (Baum) mit Startknoten als Wurzel
- Bei nicht erreichbaren Knoten im ersten Aufruf wird die Breitensuche erneut aufgerufen
- Dadurch entsteht ein Wald (=mehrere Bäume)
- Distanzberechnung durch Breitensuche möglich, da jeder Knoten um 1 höhere Distanz als sein Vorgänger hat.

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).

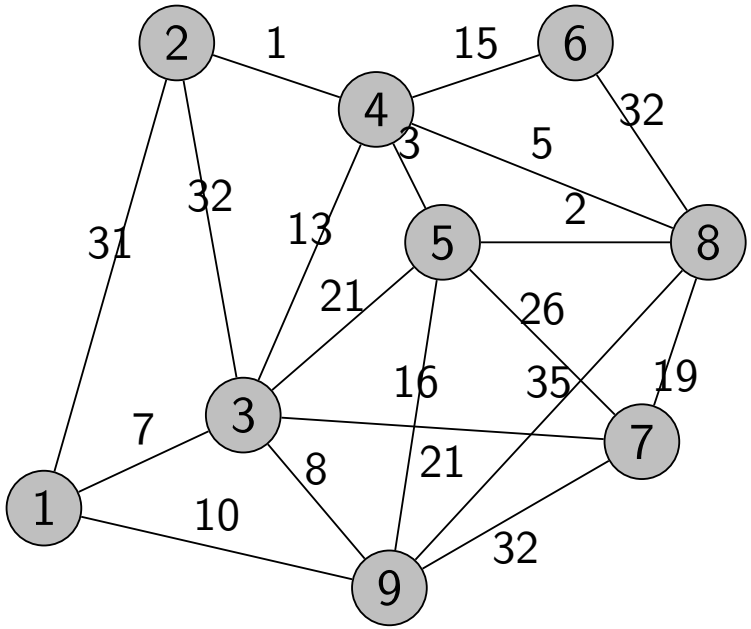
Algorithmus von Kruskal

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.

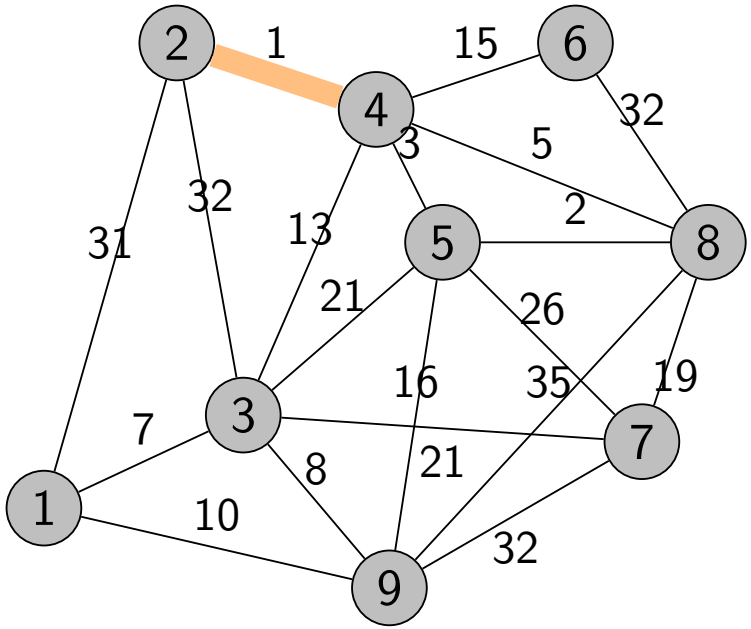
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

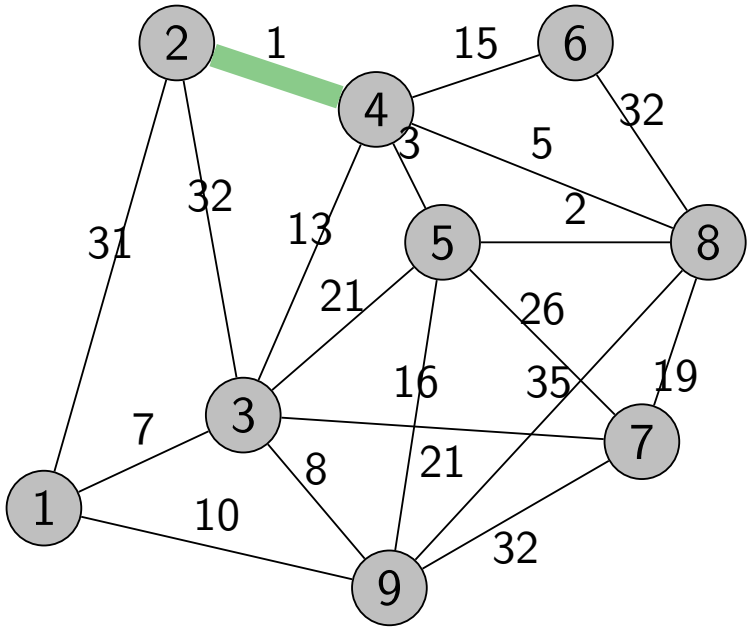
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

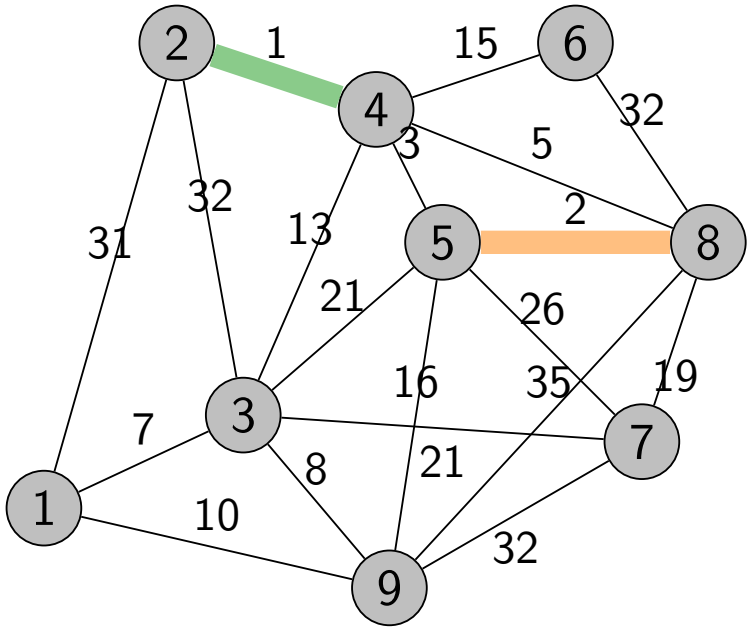
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

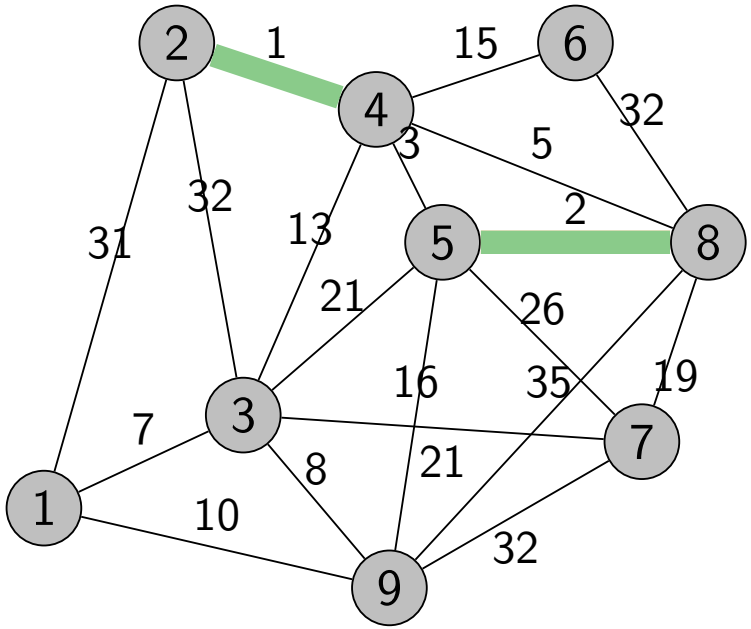
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

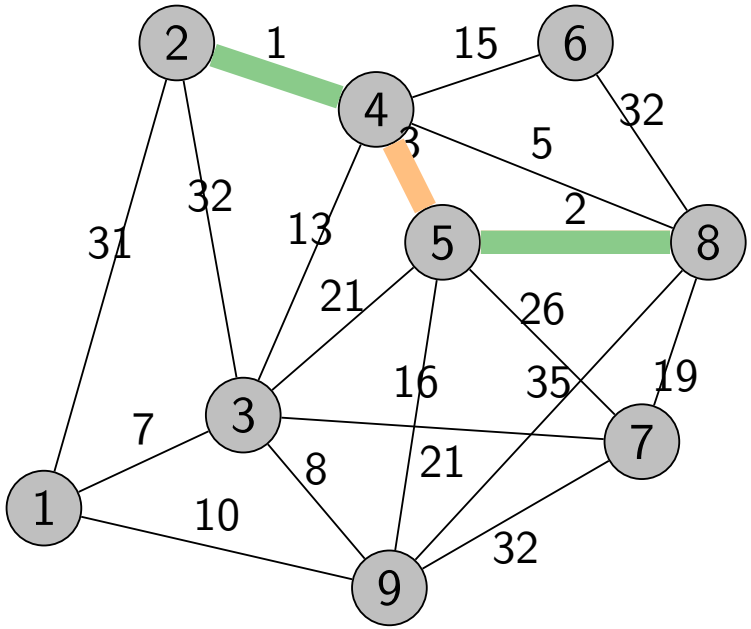
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

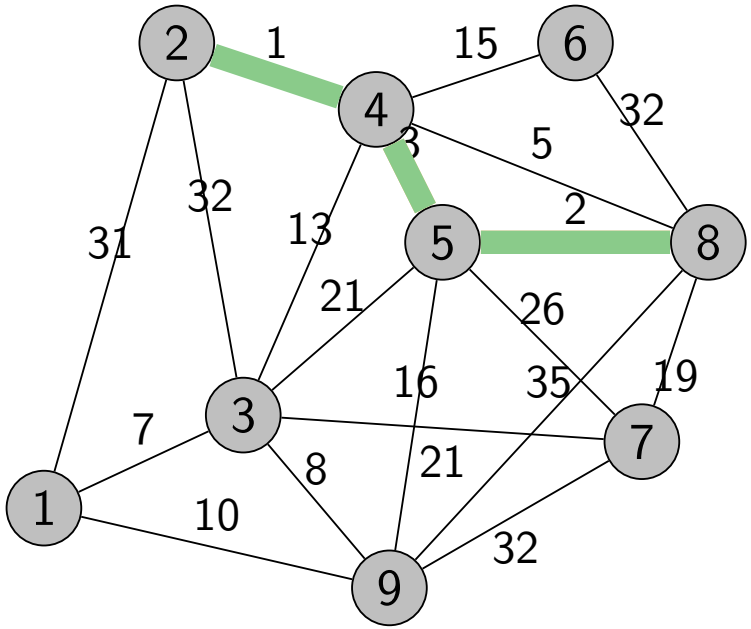
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

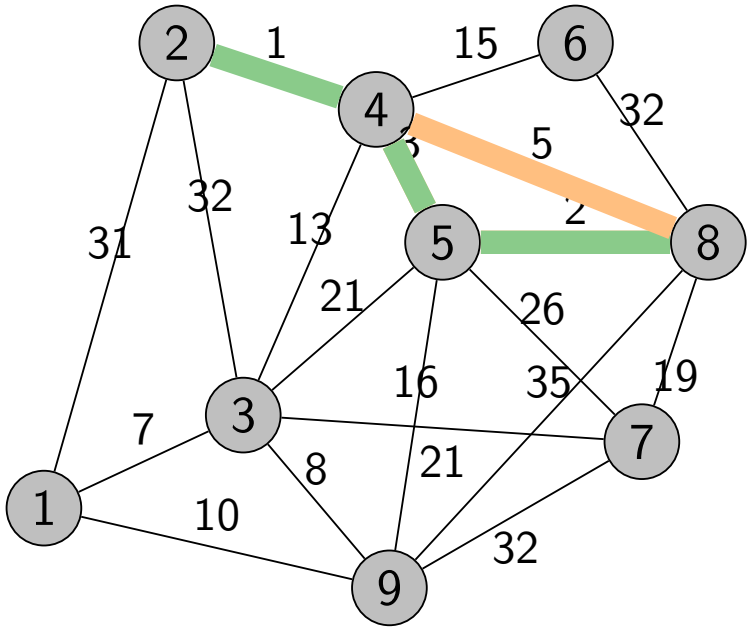
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

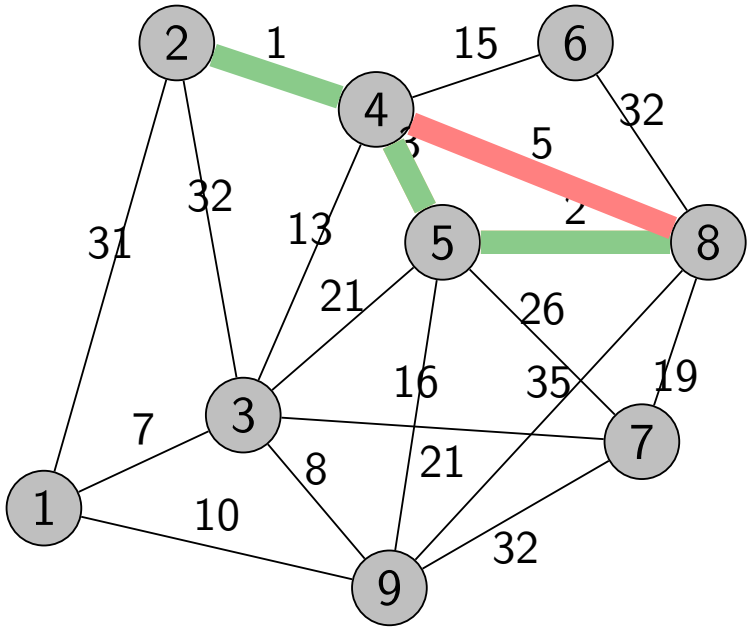
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

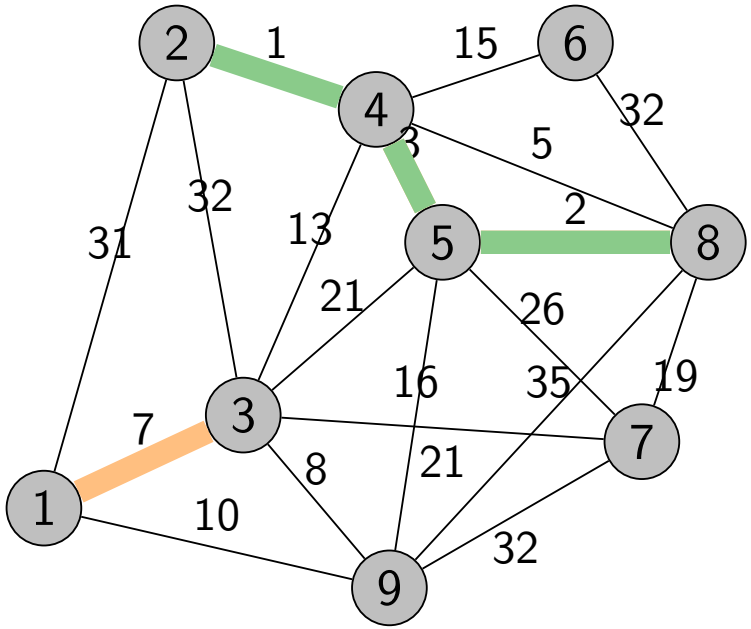
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

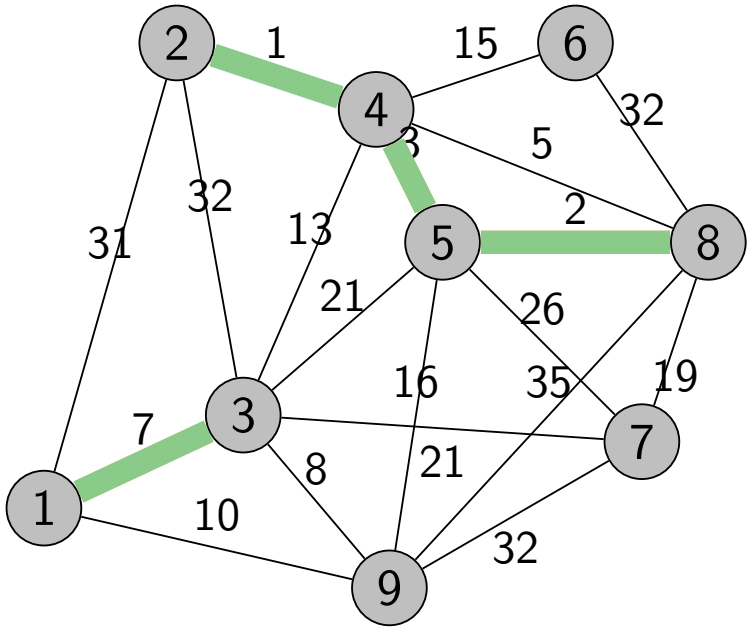
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

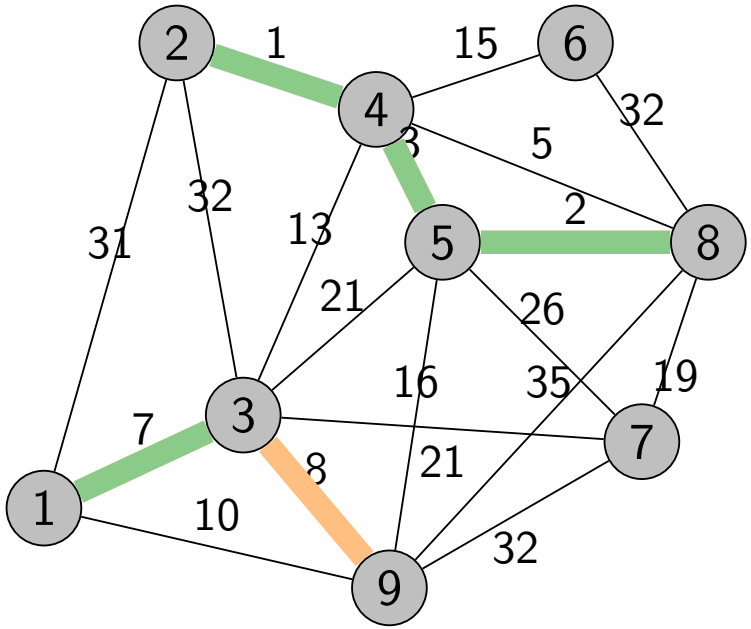
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

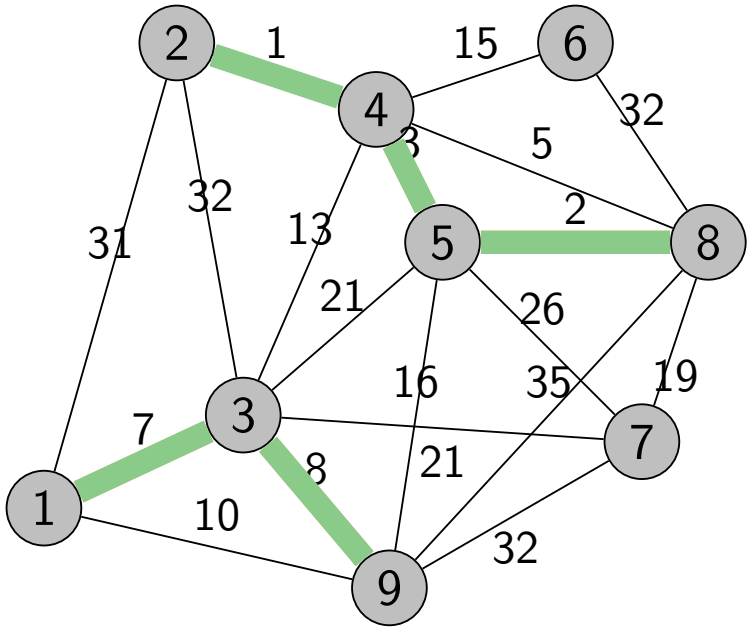
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

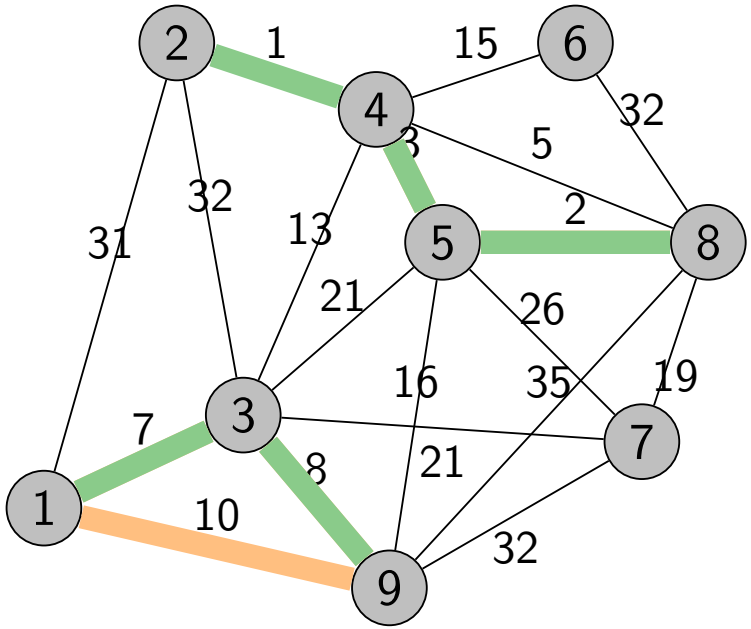
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

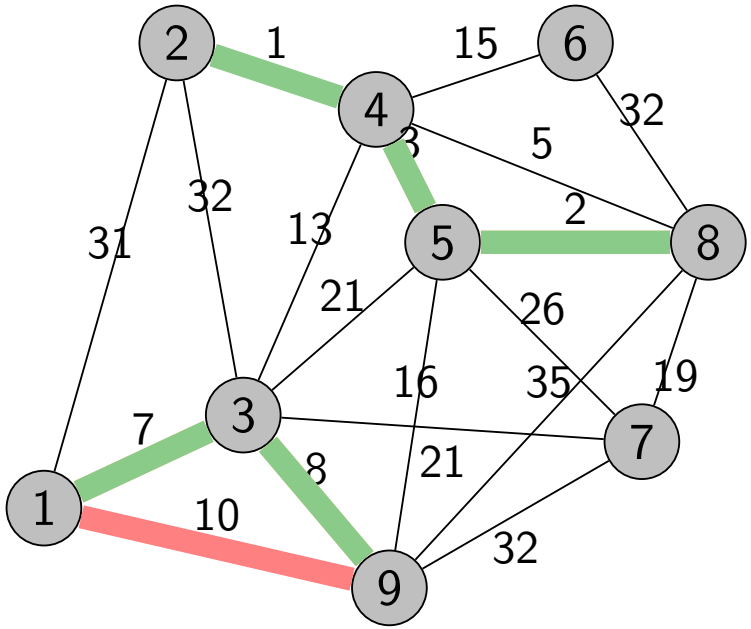
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

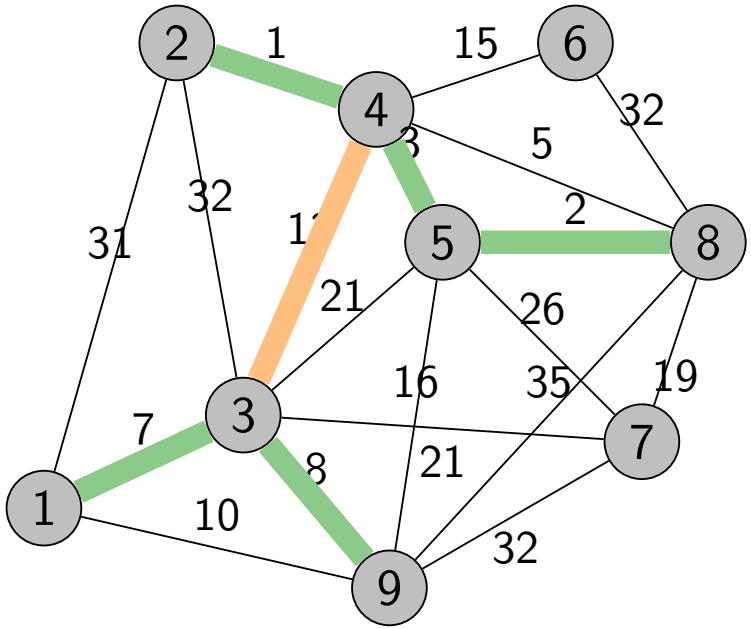
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

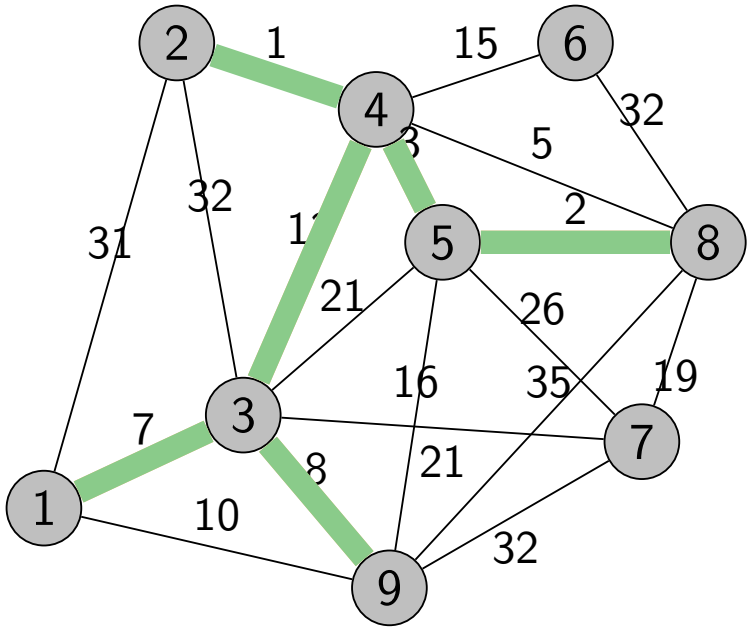
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

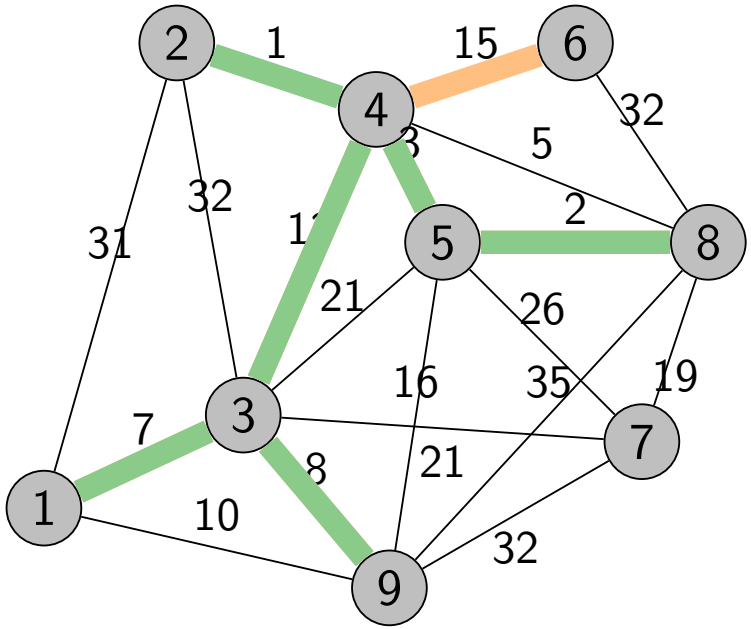
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

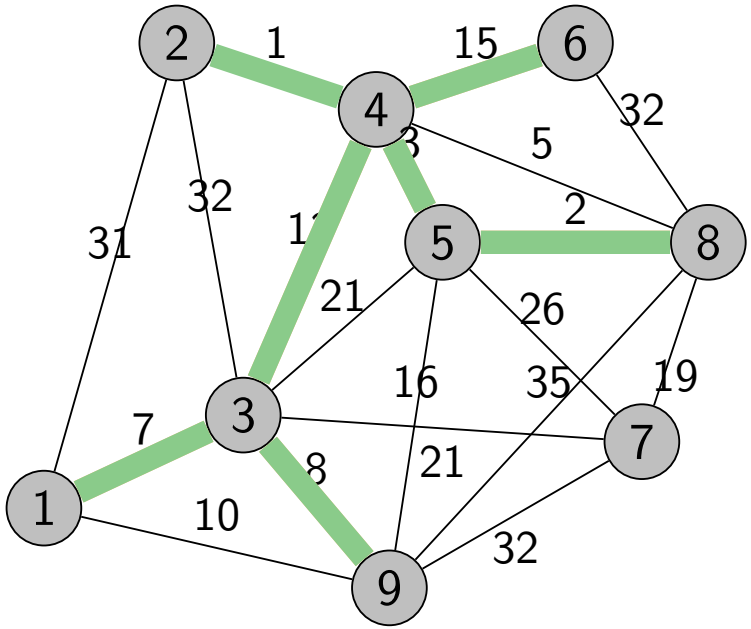
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

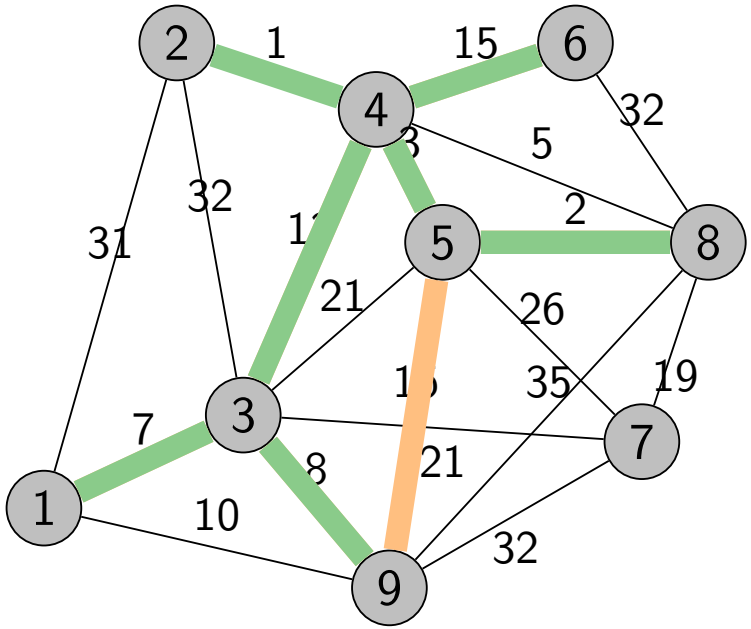
Algorithmus von Kruskal



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:

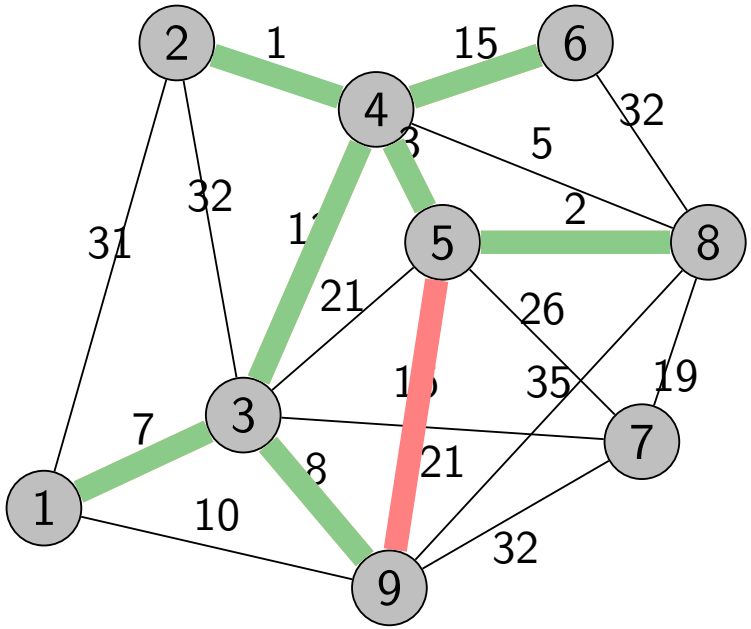
$w(\lceil 2,4 \rceil) = 1, w(\lceil 5,8 \rceil) = 2, w(\lceil 4,5 \rceil) = 3, w(\lceil 4,8 \rceil) = 5$

$w(\lceil 1,3 \rceil) = 7, w(\lceil 3,9 \rceil) = 8, w(\lceil 1,9 \rceil) = 10, w(\lceil 3,4 \rceil) = 13$

$w(\lceil 4,6 \rceil) = 15, w(\lceil 5,9 \rceil) = 16, w(\lceil 7,8 \rceil) = 19, w(\lceil 3,5 \rceil) = 21$

$w(\lceil 3,7 \rceil) = 21, w(\lceil 5,7 \rceil) = 26, \ldots$

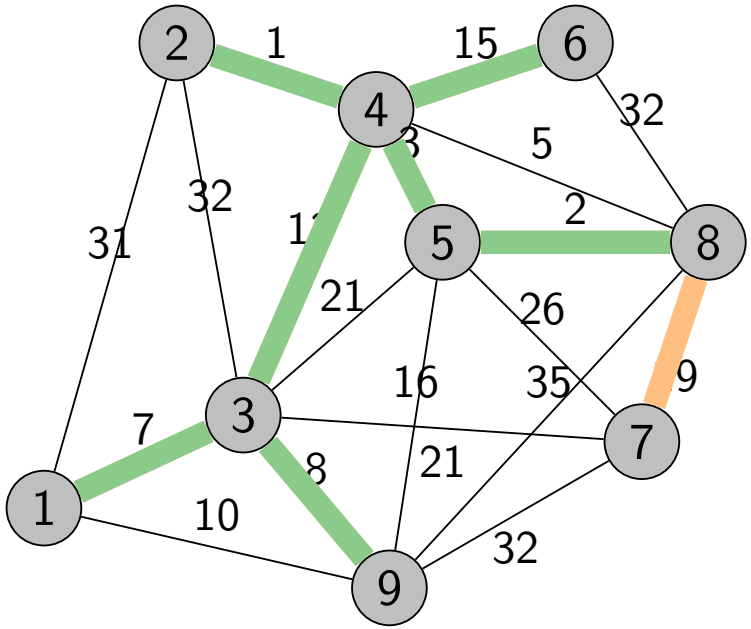
Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

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

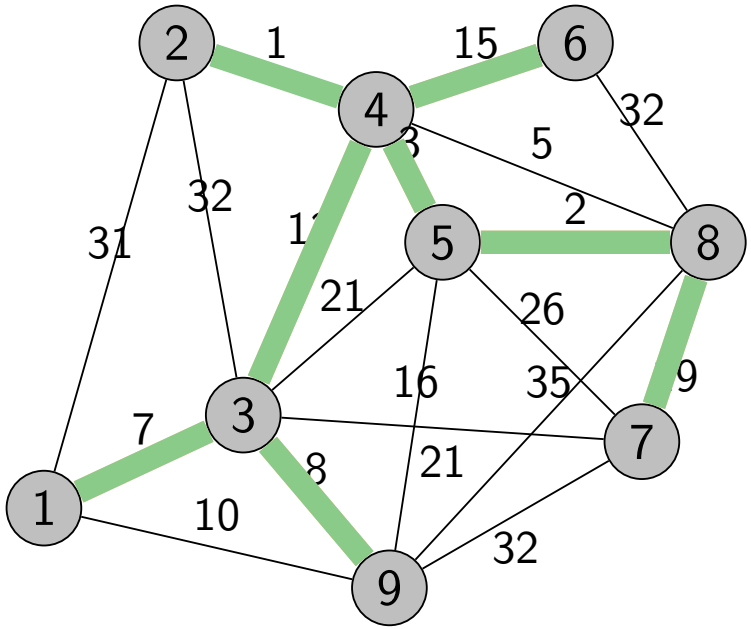
Algorithmus von Kruskal



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:

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

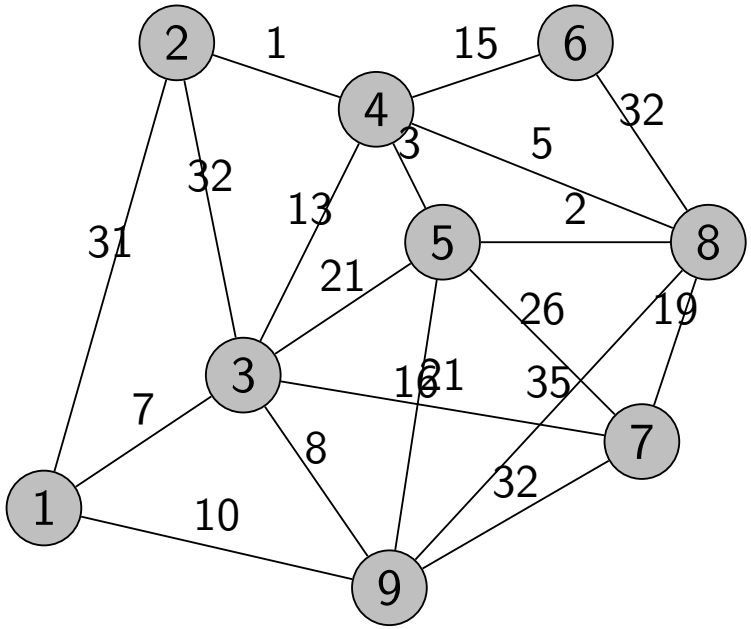
Algorithmus von Prim

Der Algorithmus von Prim konstruiert ebenfalls einen MST.

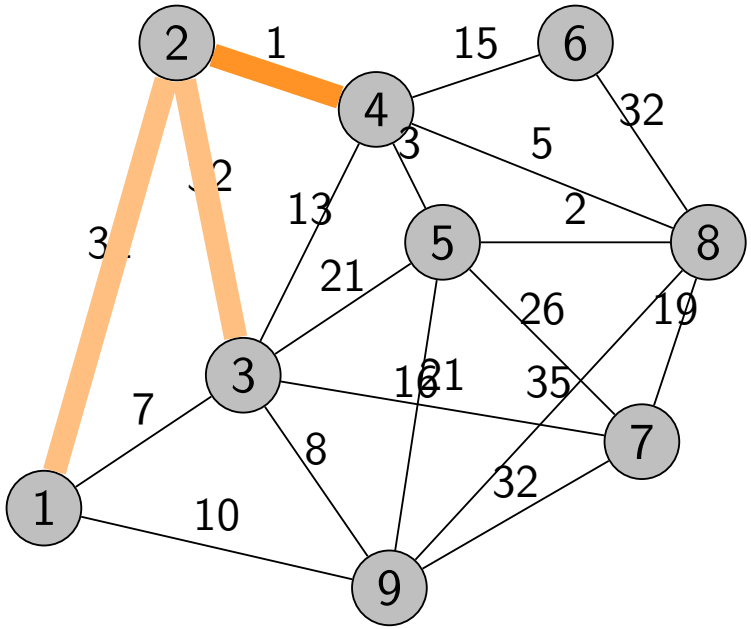
Algorithm 3: Algorithmus von Prim

```
1 Function Prim(Graph G)
      Result: Minimum Spanning Tree T
2     E(T) = ∅;
3     V(T) = ein zufaellig gewaehlter Startknoten;
4     while V(T) ⊂ V(G) do
5         Sei E' die Menge aller Kanten zwischen Knoten
6         aus V(T) und V(G)\V(T)
7         e' = argmin_{e∈E'} w(e); // Kante mit kleinstem Gewicht
8         E(T) = E(T) ∪ e';
9         Füge neuen Knoten von e' zu V(T) hinzu;
```

Algorithmus von Prim

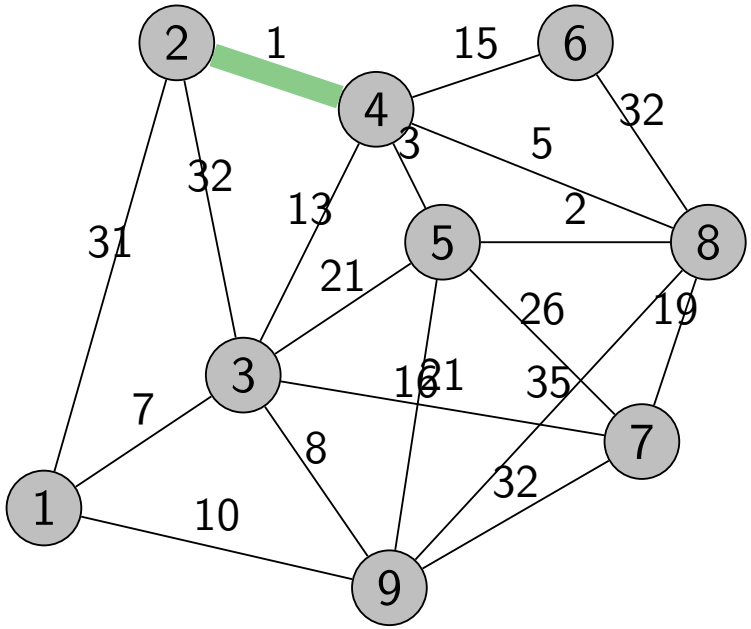


Algorithmus von Prim

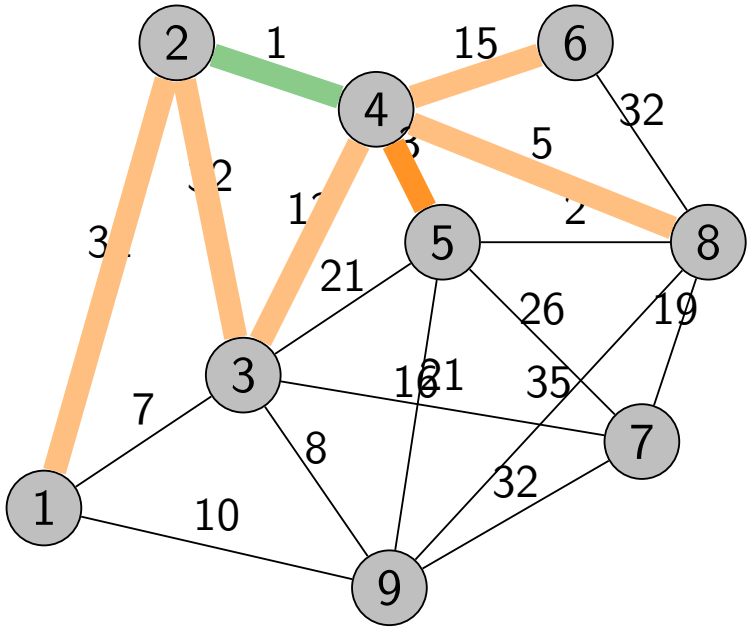


Startknoten: 2

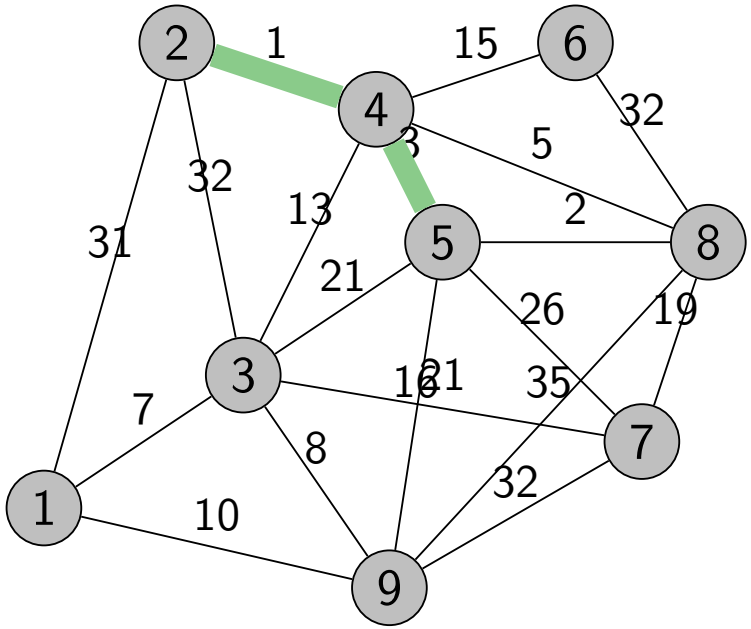
Algorithmus von Prim



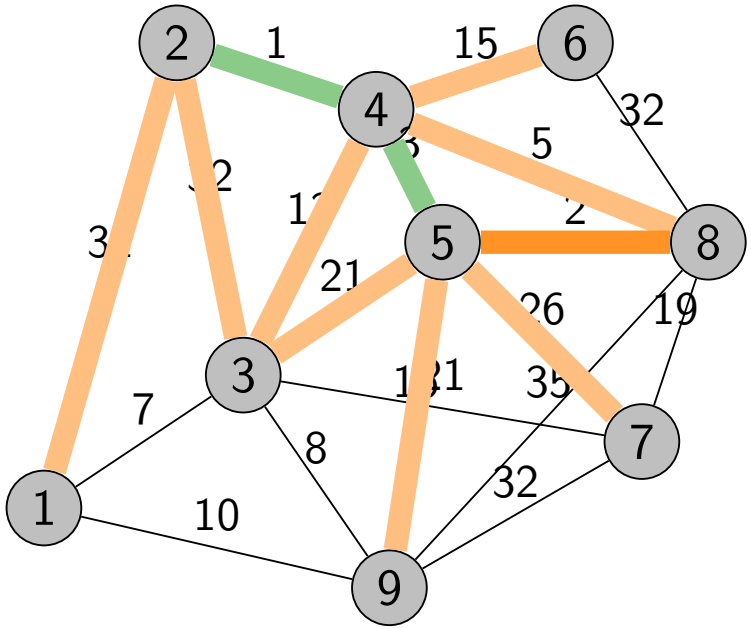
Algorithmus von Prim



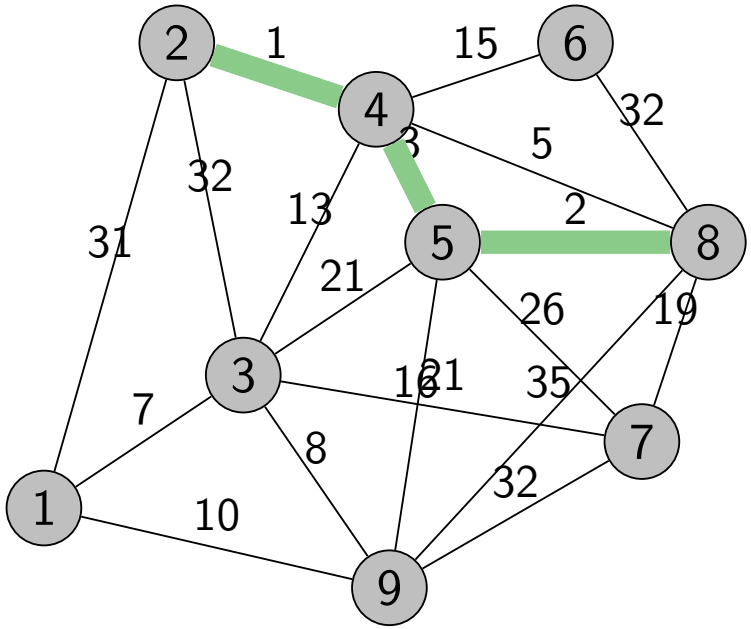
Algorithmus von Prim



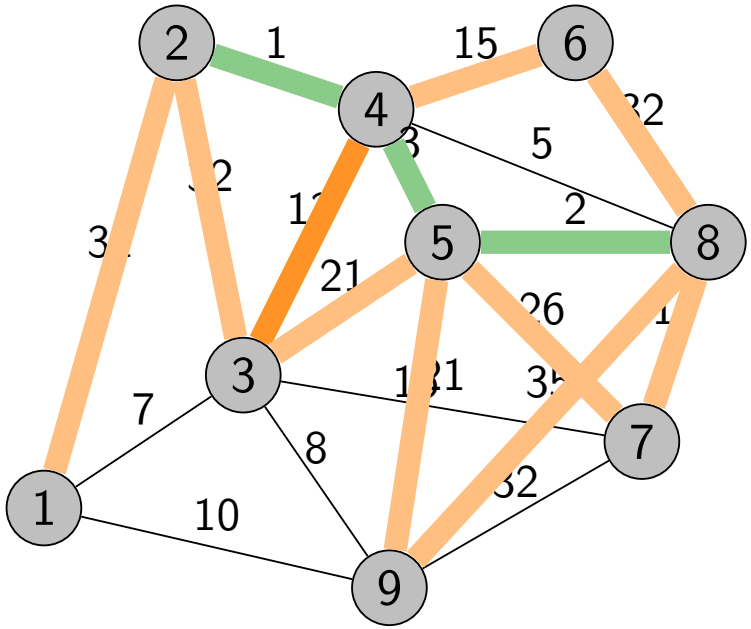
Algorithmus von Prim



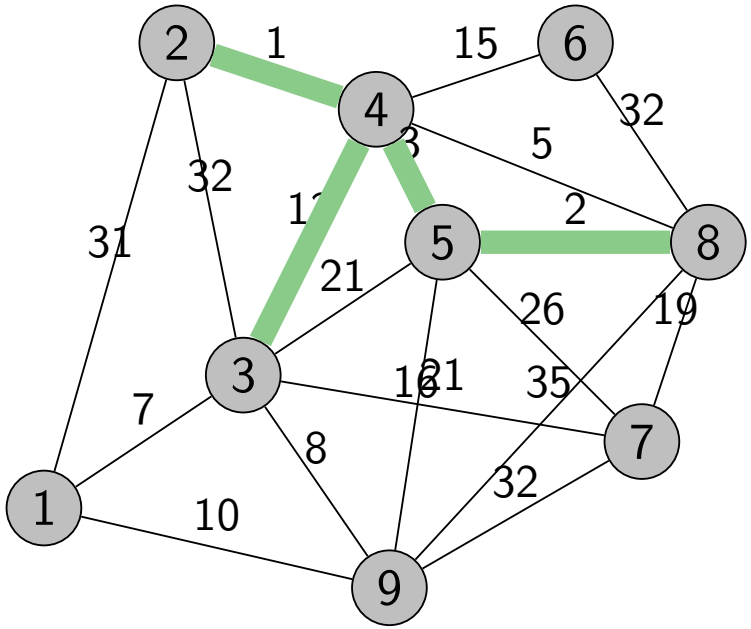
Algorithmus von Prim



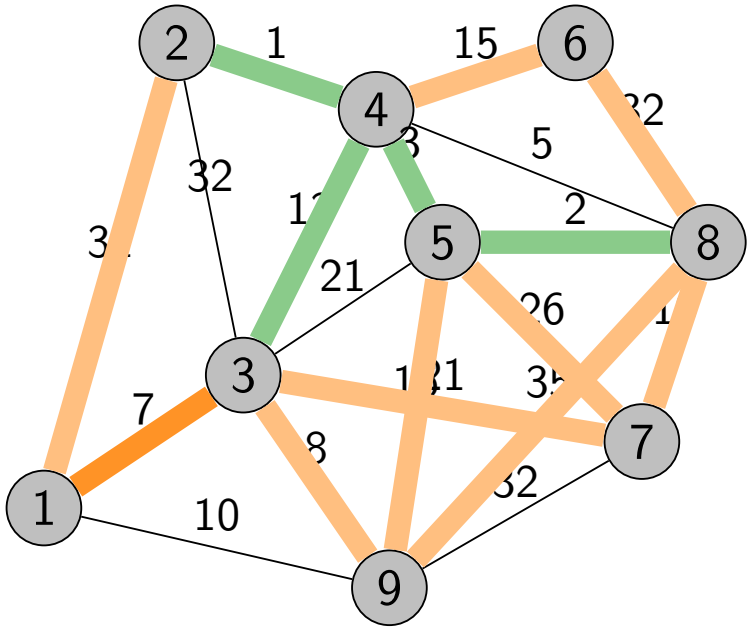
Algorithmus von Prim



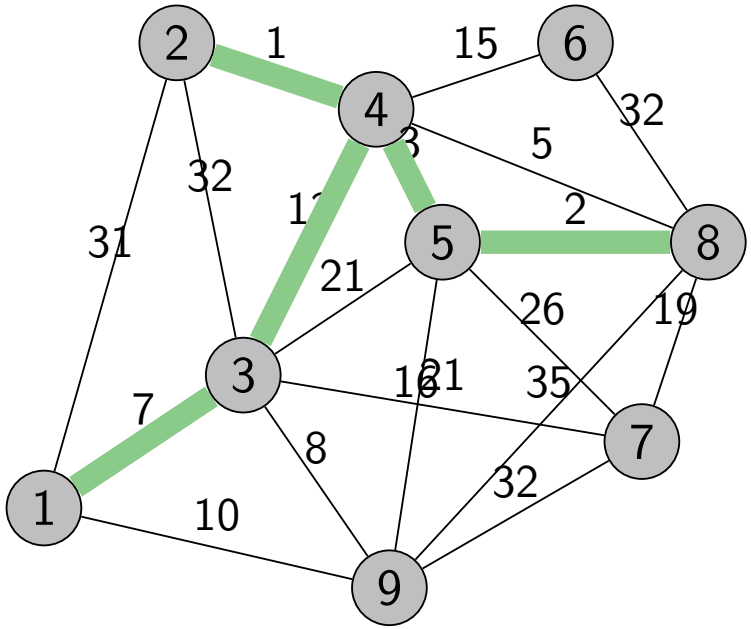
Algorithmus von Prim



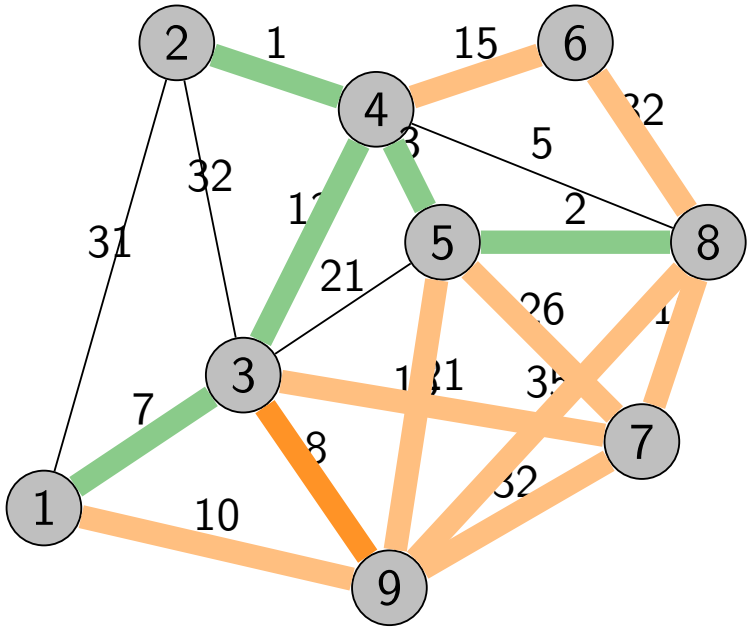
Algorithmus von Prim



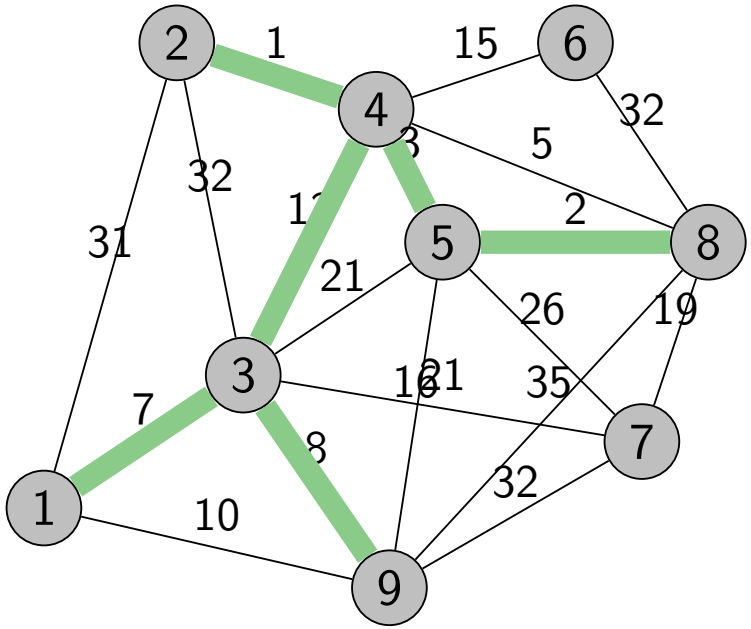
Algorithmus von Prim



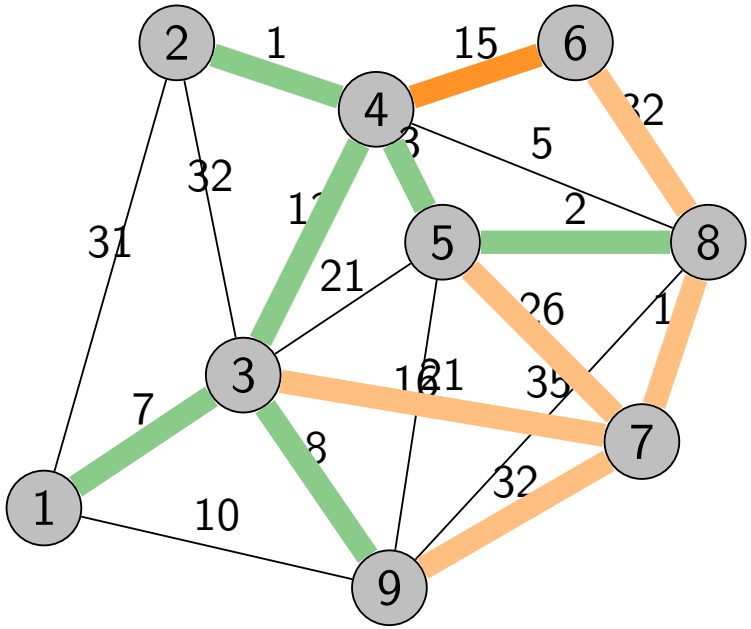
Algorithmus von Prim



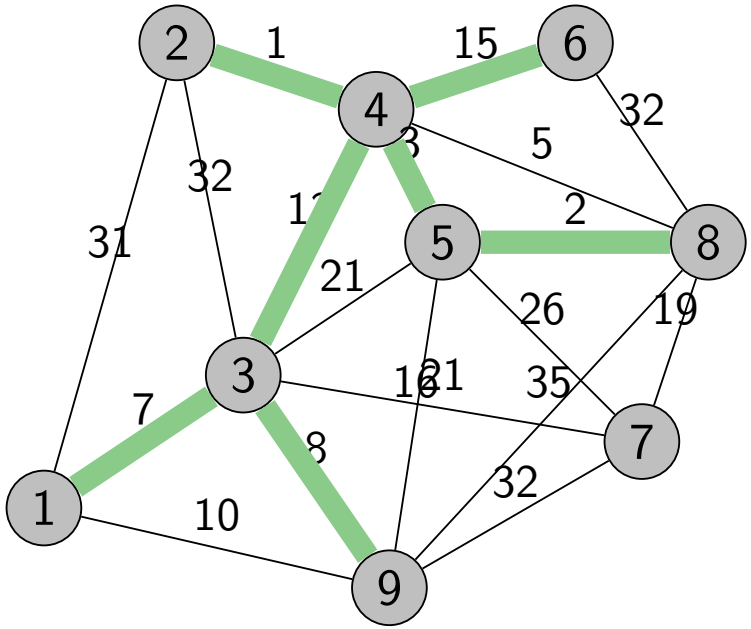
Algorithmus von Prim



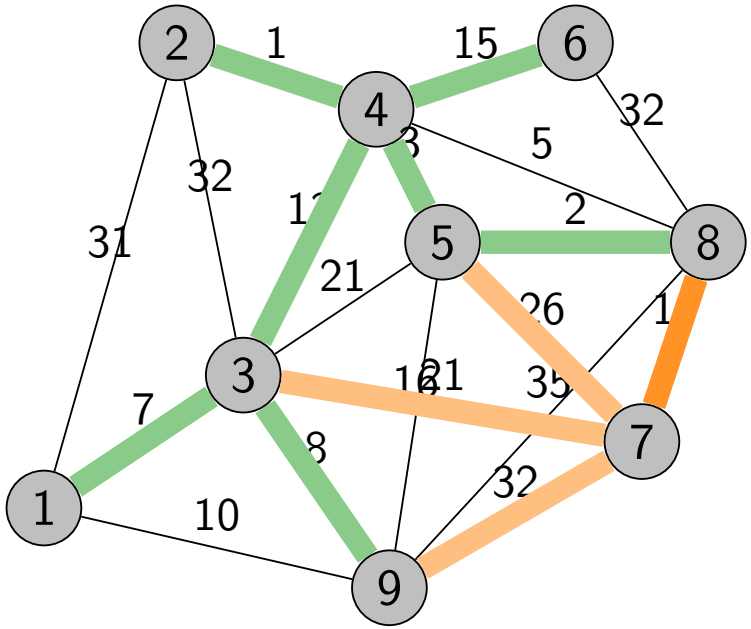
Algorithmus von Prim



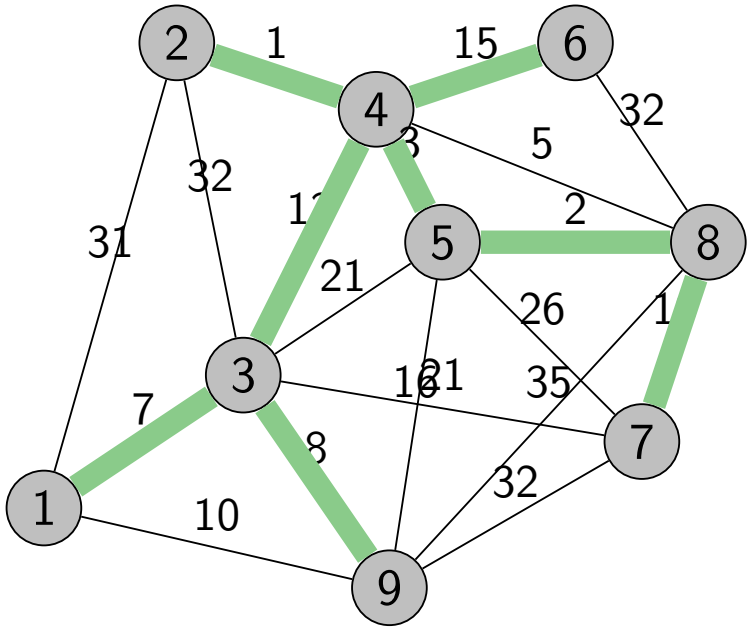
Algorithmus von Prim



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!