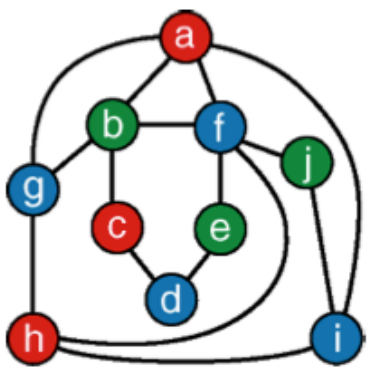
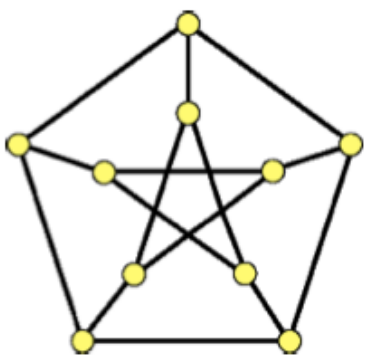
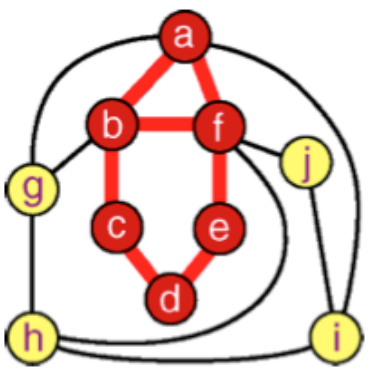
Algorithmische Graphentheorie

Vorlesung 5: Suchalgorithmen

Babeş-Bolyai Universität, Department für Informatik, Cluj-Napoca
csacarea@cs.ubbcluj.ro







Wiederholung - Bäume

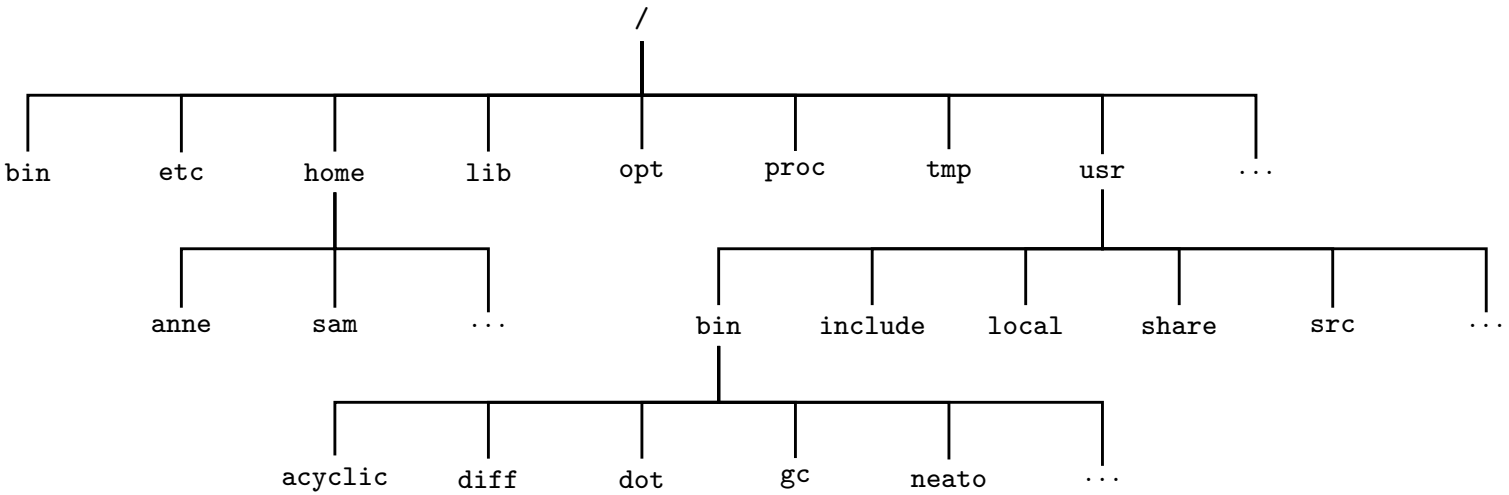


Abbildung 1: Die Hierarchie des Linux Dateisystems.

Wichtige Begriffe

- Gerichtete Bäume vs. ungerichtete Bäume
- Wurzel, Vorgänger, Nachgänger oder Kind und Eltern
- Tiefe - Entfernung von der Wurzel und Höhe - Länge des längsten Pfades von der Wurzel aus startend.
- Verzweigungsfaktor

Beispiel

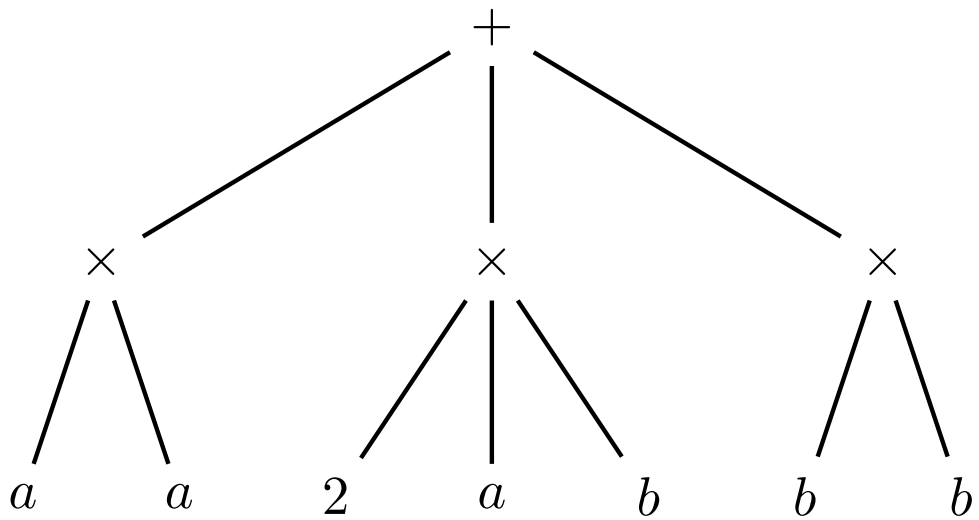


Abbildung 2: $a^2 + 2ab + b^2$.

Wiederholung - Bäume

Definition

Es sei $G = (V, E)$ ein ungerichteter Graph und $T = (V, E)$ ein Teilgraph von G. Dann heißt T ein aufspannender Baum von G, wenn $V = V$ gilt und T ein Baum ist.

Ein Graph besitzt im Allgemeinen mehr als einen spannenden Baum, manchmal sogar sehr viele. Um dies zu illustrieren betrachten wir Graphen mit möglichst vielen Kanten.

Definition

Ein ungerichteter Graph $G = (V, E)$ heißt vollständig, wenn für je zwei Knoten $v, u \in V$ gilt $\{v, u\} \in E$. Einen ungerichteten, einfachen und vollständigen Graphen mit n Knoten bezeichnet man mit K_n.

Anzahl der aufspannenden Bäume

Wir wollen nun untersuchen, wieviele spannende Bäume ein vollständiger Graph enthält.

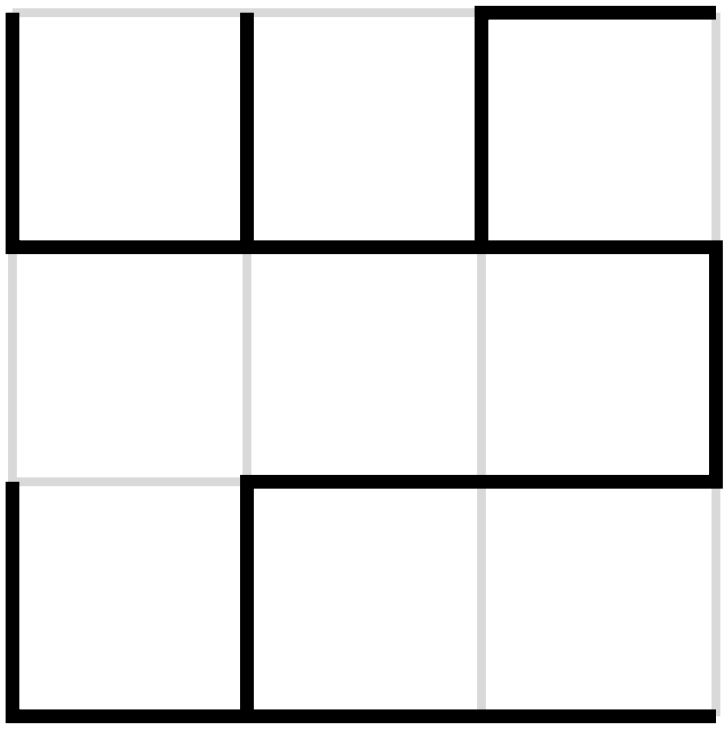
Lemma

Es sei $T = (V, E)$ ein ungerichteter Baum mit $|V| \geq 2$. Dann besitzt T mindestens zwei Knoten $v \in V$ mit $deg(v) = 1$. Diese nennt man Blätter.

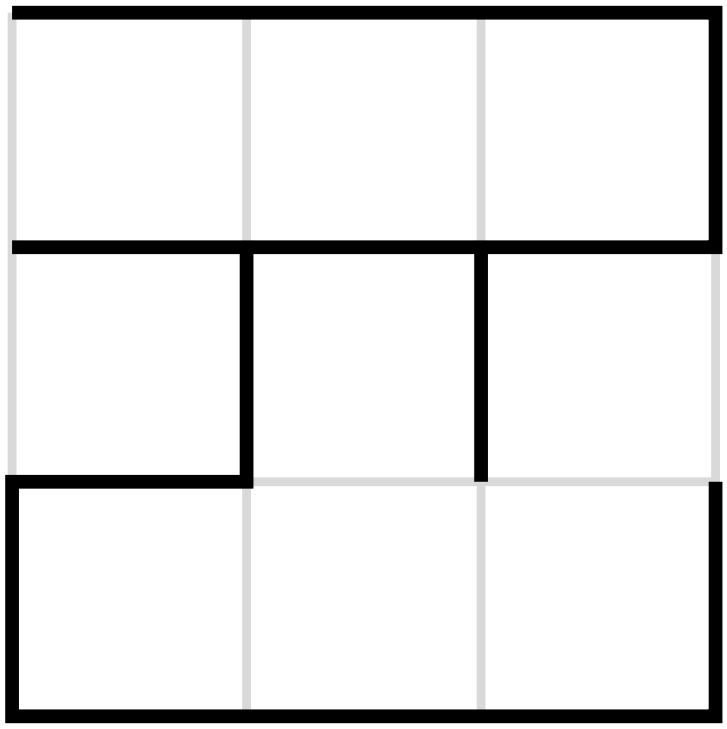
Satz (Satz von Cayley)

Für $n \geq 2$ gibt es n^{n-2} verschiedene aufspannende Bäume in einem vollständigen Graphen K_n mit eindeutig markierten Knoten.

Beispiel



(a)



(b)

Abbildung 3: Zwei aufspannende Bäume für den 4×4 Grid-Graph.

Rekursiver Aufbau von Bäumen

Ein isolierter Knoten ist ein Baum. Dieser Knoten ist die Wurzel des Baumes. Gegeben $T_1, T_2, \ldots, T_n$ eine Liste von $n > 0$ Bäume, erzeuge einen neuen Baum wie folgt:

1. Sei T der Baum mit genau einem Knoten v, die Wurzel von T.
2. Sei v_i die Wurzel des Baumes T_i.
3. Für $i = 1, 2, \ldots, n$, füge die Kanten vv_i in T ein und füge T_i zu T. Jedes v_i ist ein Kind von v.

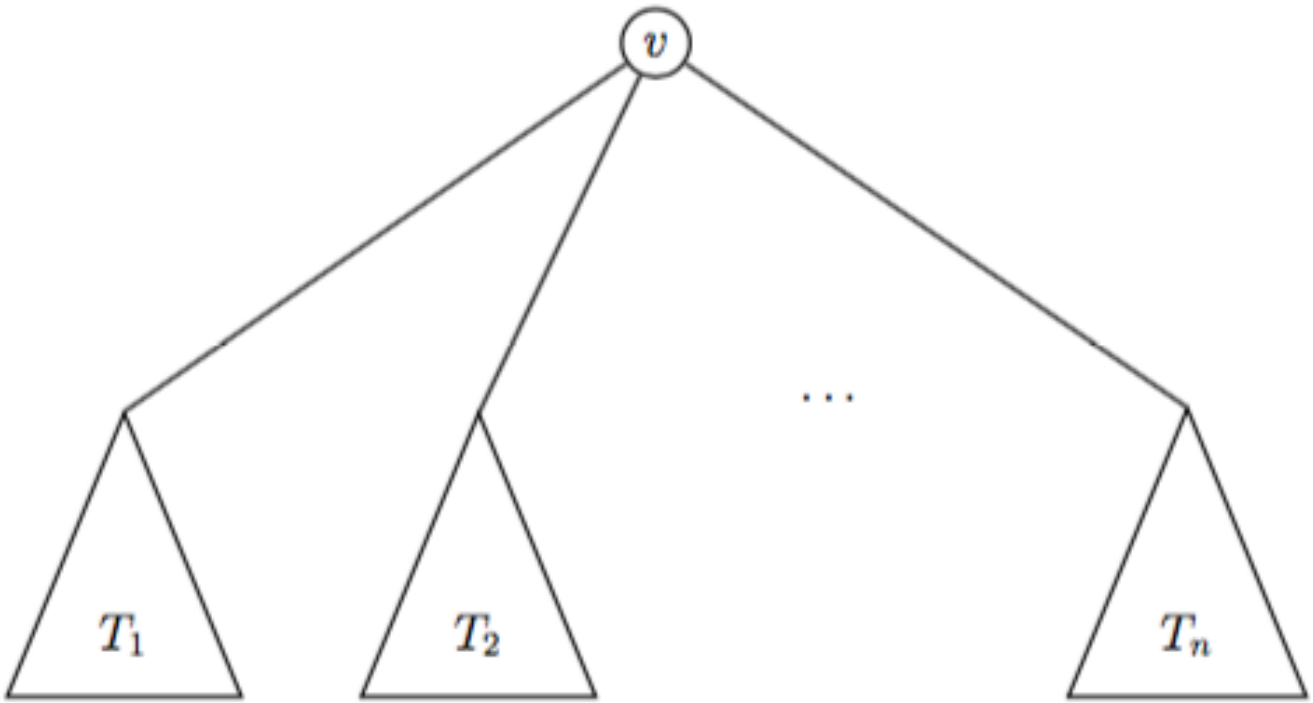
Das Ergebnis ist ein Baum T mit Wurzel v, Knotenmenge

$$V(T) = \{v\} \cup \left(\bigcup_i V(T_i) \right)$$

und Kantenmenge

$$E(T) = \bigcup_i \left(\{vv_i\} \cup E(T_i) \right).$$

Rekursiver Aufbau von Bäumen



Bäume in gerichteten Graphen

Definition

Sei $G = (V, E)$ ein gerichteter Graph, so heißt G stark zusammenhängend, wenn für je zwei Knoten $s, t \in V$ ein (gerichteter) Weg P mit Anfangsknoten s und Endknoten t existiert, und schwach zusammenhängend, wenn für je zwei Knoten $s, t \in V$ ein ungerichteter Weg P mit Anfangsknoten s und Endknoten t existiert.

Ein Knoten $s \in V$ heißt Wurzel von G, wenn für alle Knoten $t \in V$ ein gerichteter Weg P mit Anfangsknoten s und Endknoten t existiert.

Bemerkung

Ein gerichteter Graph, der eine Wurzel besitzt, ist also immer schwach zusammenhängend, aber nicht jeder schwach zusammenhängende Graph besitzt eine Wurzel. Ein gerichteter Graph ist stark zusammenhängend genau dann, wenn jeder Knoten eine Wurzel ist.

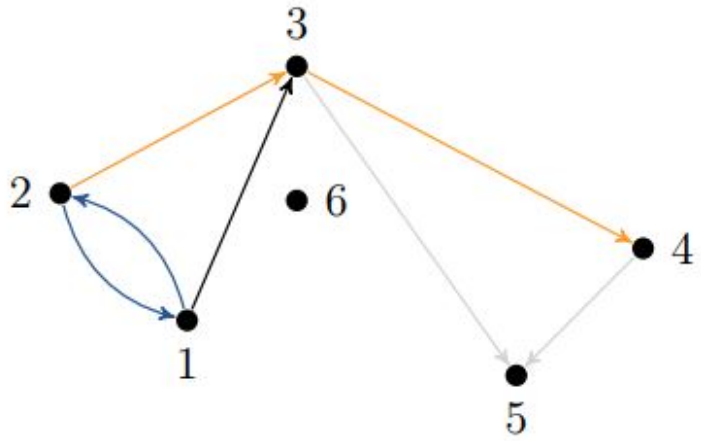


Abbildung 4: Ein gerichteter Graph der weder stark noch schwach zusammenhängend ist. Würde der Knoten 6 nicht zum Graphen gehören, so wären die Knoten 1 und 2 Wurzeln.

Bäume in gerichtete Graphen

Ungerichteter Graph ist ein Baum...

Bäume in gerichtete Graphen

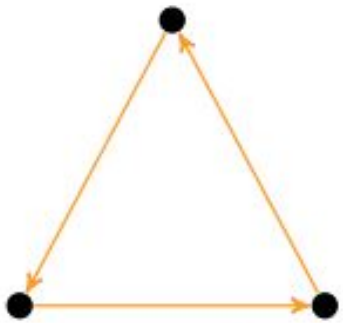
Ungerichteter Graph ist ein Baum...

genau dann, wenn er zusammenhängend und kreisfrei ist.

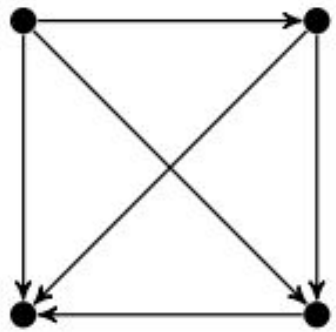
Übertragung auf gerichtete Graphen...

- Ist ein gerichteter Baum stark oder schwach zusammenhängend?
- Darf er keine gerichteten einfache Kreise enthalten oder auch keine ungerichteten einfachen Kreise?

Zusammenhängende gerichtete Graphen



(a) stark zusammenhängend, mit gerichtetem einfachem Kreis

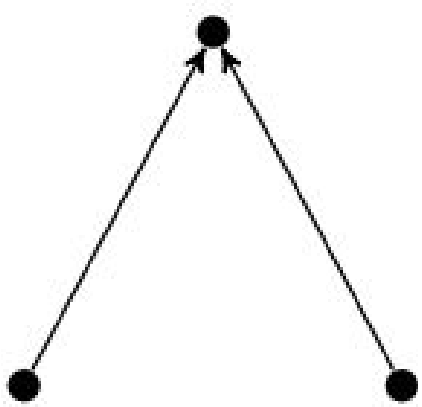


(b) schwach zusammenhängend, ohne gerichtete einfache Kreise

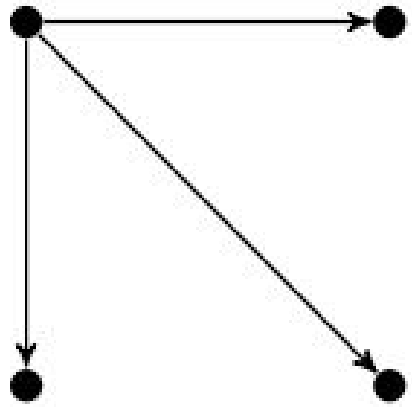
Bäume in gerichtete Graphen

Definition

Es sei $T = (V, E)$ ein gerichteter Graph. T heißt Baum, wenn der zugeordnete ungerichtete Graph ein Baum ist.



(a) Baum



(b) Wurzelbaum

Abbildung 5: Gerichtete Bäume

Bemerkung

- Bäume in gerichteten Graphen sind schwach zusammenhängend und kreisfrei $\rightarrow$ sie enthalten keine ungerichteten einfachen Kreise.
- Der Rückzug auf schwachen Zusammenhang hat allerdings den Nachteil, dass in einem gerichteten Graphen $T = (V, E)$ für zwei beliebige Knoten $s, t \in V$ nicht garantiert ist, dass es in T einen gerichteten Weg von s nach t gibt.

Definition

Es sei $T = (V, E)$ ein gerichteter Graph. T heißt Wurzelbaum mit Wurzel s, wenn T ein Baum ist, in dem s eine Wurzel ist.

Satz

Es sei $T = (V, E)$ ein gerichteter Graph und $s \in V$. Dann sind äquivalent:

(a) T ist ein Wurzelbaum mit Wurzel s.

(b) T ist ein Baum, $indeg(s) = g^-(s) = 0$ und $indeg(v) = g^-(v) = 1$ für alle $v \in V \setminus \{s\}$.

(c) s ist eine Wurzel in T, $indeg(s) = g^-(s) = 0$ und $indeg(v) = g^-(v) = \leq 1$ für alle $v \in V \setminus \{s\}$.

Beweis

(a) $\Longrightarrow$ (b) Da alle Knoten $v \in V \setminus \{s\}$ von s aus durch einen gerichteten Weg erreichbar sind, folgt für sie $g^-(v) \geq 1$. Weil außerdem der zu T gehörende ungerichtete Graph ein Baum ist, folgt

$$|V| - 1 = |E| = \underbrace{g^-(s)}_{\geq 0} + \sum_{v \in V \setminus \{s\}} \underbrace{g^-(v)}_{\geq 1} \geq |V| - 1.$$

Folglich muss $g^-(s) = 0$ und $g^-(v) = 1$ für alle $v \neq s$ gelten.

(b) $\Longrightarrow$ (c) Wir müssen nur zeigen, dass s eine Wurzel von T ist. Sei dafür $v \in V$ beliebig. Dann müssen wir zeigen, dass v durch einen gerichteten Weg von s aus erreichbar ist. Für $v = s$ ist nicht zu zeigen, sei also $v \neq s$. Da T schwach zusammenhängend ist, gibt es einen, möglicherweise ungerichteten, einfachen Weg $P = (s = v_0, e_1, v_1, e_2, \ldots, e_l, v_l = v)$. Wegen $g^-(s) = 0$ folgt $e_1 = (v_0, v_1)$, d.h. die erste Kante auf dem Weg zeigt in die richtige Richtung. Wegen $g^-(v_1) = 1$ und $v_2 \neq v_0$ muss dann aber auch $e_2 = (v_1, v_2)$ gelten, d.h. auch die zweite Kante zeigt in die richtige Richtung. Iterativ zeigt man so, dass P tatsächlich ein gerichteter Weg ist.

Beweis (2)

(c) $\Longrightarrow$ **(a)** Wir müssen zeigen, dass T ein Baum ist. Nach Voraussetzung ist s eine Wurzel in T, daher ist T schwach zusammenhängend und es gilt $|E| \geq |V|-1$. Andererseits gilt auch

$$|E| = \underbrace{g^-(s)}_{=0} + \sum_{v \in V \setminus \{s\}} \underbrace{g^-(v)}_{\leq 1} \leq |V| - 1,$$

daher ist T ein Baum. $\square$

Minimale aufspannende Bäume

In vielen Anwendungsproblemen geht es darum gewisse Knoten mit möglichst wenigen Kanten so zu verbinden, dass zwischen allen Paaren von Knoten ein Weg existiert.

Magnetschwebebahn

Städte werden als Knoten modellieren und die dazwischen technisch möglichen Verbindungen als Kanten. Wenn wir davon ausgehen, dass die Bahn immer in beide Richtungen fährt, erhalten wir so einen ungerichteten Graphen. Aus Kostengründen werden nur möglichst wenige Strecken tatsächlich gebaut, d.h. wir suchen einen aufspannenden Baum. Wir wissen schon, dass wir mindestens *Anzahl der Städte - 1* Strecken realisieren müssen. Nun kosten die möglichen Strecken aber nicht alle gleich viel. Welchen aufspannenden Baum sollen wir also wählen?

Minimale aufspannende Bäume

Broadcasting

Betrachten ein Netzwerk von Sendemasten unterschiedlicher Reichweiten, in dem eine Information kostengünstig von einem Mast an alle anderen weitergegeben werden soll, so erhalten wir einen gerichteten Graphen, in dem wir einen aufspannenden Wurzelbaum suchen.

Gewichtete Graphen und minimale spannende Bäume

Definition

Es sei $G = (V, E)$ ein ungerichteter oder gerichteter Graph. G heißt gewichtet, wenn jeder Kante $e \in E$ ein Gewicht $w(e)$ zugeordnet ist.

Definition

Es sei $G = (V, E)$ gewichteter ungerichteter Graph. Ein Teilgraph $T = (V, E_T)$ von G heißt minimaler spannender Baum von G, wenn sein Gewicht $w(T) := \sum_{e \in E_T} w(e)$ minimal unter allen spannenden Bäumen von G ist.

Charakterisierungssatz minimal Spannbäume

Theorem

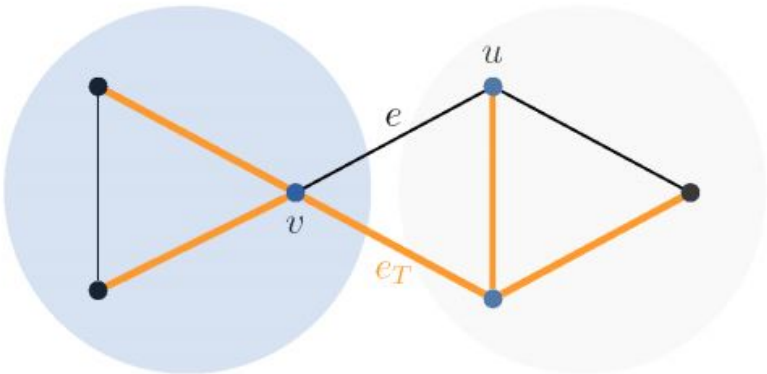
Es sei $G = (V, E)$ ein ungerichteter, gewichteter und zusammenhängender Graph mit $n = |V|$ Knoten und Kantengewichten $w(e)$ für alle $e \in E$ und $T = (V, E_T)$ ein aufspannender Baum. Dann sind äquivalent:

1. *T ist ein minimaler aufspannender Baum.*
2. *Für jede Kante $e \in E \setminus E_T$ gilt: e ist eine Kante mit maximalem Gewicht in dem eindeutigen einfachen Kreis, der entsteht, wenn man e zu T hinzufügt.*
3. *Für jede Kante $e_T \in E_T$ gilt: Entfernt man e_T aus T, so zerfällt T in zwei Zusammenhangskomponenten Z_1 und Z_2. Die Kante e_T hat minimales Gewicht, und hat in beiden Zusammenhangskomponenten jeweils einen Endknoten.*

Beweis

$1 \Rightarrow 2$

Sei T ein minimaler spannender Baum und $e = \{v, u\} \in E \setminus E_T$ beliebig. Da die Knoten v, u im Baum T durch einen eindeutigen einfachen Weg P verbunden sind, entsteht durch das Hinzufügen der Kante e zu T ein eindeutiger einfacher Kreis. Angenommen, es gibt eine Kante $e_T \in P$ mit $w(e_T) > w(e)$. Dann betrachten wir den Graphen $T = (V, (E \setminus \{e_T\}) \cup \{e\})$. Es gilt offensichtlich $w(T) < w(T)$. Außerdem hat T' wieder $n - 1$ Kanten und ist zusammenhängend. Folglich ist T' ein spannender Baum mit einem kleineren Gewicht als T. Widerspruch.



Beweis (2)

$2 \Rightarrow 3$

- Es sei T ein aufspannender Baum mit der Eigenschaft aus (2).
- Angenommen es gibt eine Kante $e_T \in E_T$, für die (3) verletzt ist, d.h. es gibt eine Kante e mit Endknoten in Z_1 und Z_2, für die gilt $w(e) < w(e_T)$.
- Die Kanten e und e_T verbinden die gleichen Zusammenhangskomponenten, folglich muss $e \notin E_T$.
- Fügen wir die Kante e zu T hinzu, so entsteht ein eindeutiger einfacher Kreis, auf dem auch e_T liegen muss, da beide Kanten die Zusammenhangskomponenten Z_1 und Z_2 verbinden.
- Widerspruch zu (2), da e nicht die Kante mit maximalem Gewicht auf diesem Kreis ist.

Beweis (3)

$3 \Rightarrow 1$

- Sei T ein aufspannender Baum mit Eigenschaft (3) und $T^* = (V, E^*)$ ein minimaler aufspannender Baum, der maximal viele Kanten mit T gemeinsam hat.
- T und T^* sind aufspannende Bäume, d.h. sie haben gleich viele Kanten. Gilt $T \neq T^*$, so muss es also mindestens eine Kante $e_T \in E_T \setminus E^*$ geben.
- Entfernen wir e_T aus T, so zerfällt T in zwei Zusammenhangskomponenten Z_1 und Z_2.
- Da T^* ein aufspannender Baum ist, muss es eine Kante $e^* \in E^*$ geben, die Endknoten in Z_1 und Z_2 hat.
- Wegen $e_T \notin E^*$ folgt $e^* \neq e_T$ und daher $w(e^*) \geq w(e_T)$.

Beweis (4)

$3 \Rightarrow 1$

- Betrachten wir nun den Teilgraphen $T' = (V, (E^* \setminus \{e^*\}) \cup \{e_T\})$, so gilt $w(T') \leq w(T^*)$ und T' ist ebenfalls ein aufspannender Baum (vergleiche $1 \Rightarrow 2$).
- T^* ist ein minimaler aufspannender Baum, dann muss $w(T') = w(T^*)$ gelten.
- Wir haben also einen minimalen aufspannenden Baum T' konstruiert, der eine Kante mehr mit T gemeinsam hat als T^*, ein Widerspruch!
- Folglich muss $T = T^*$ gelten, d.h. T ist ein minimaler aufspannender Baum.

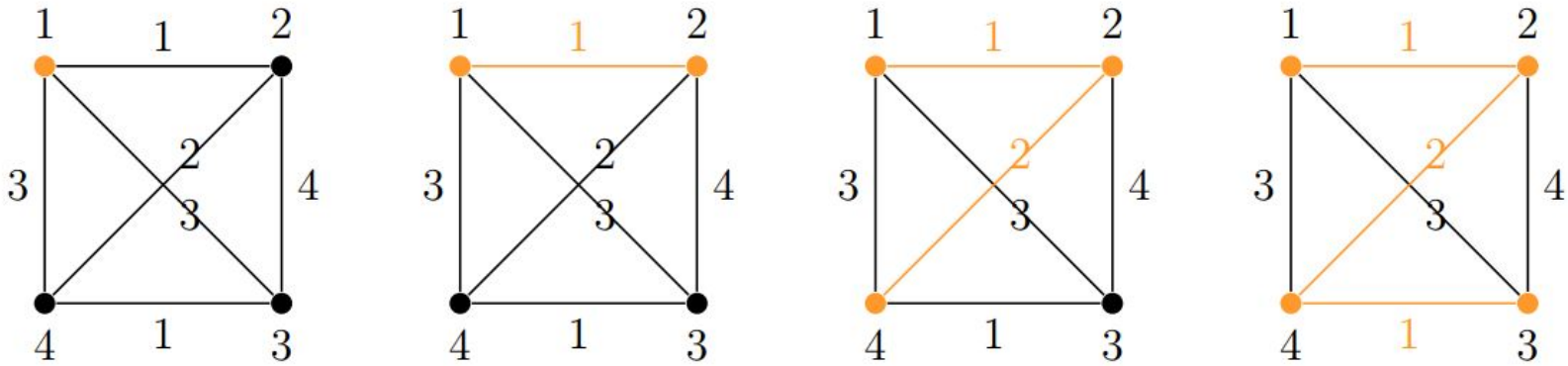
Algorithmus von Prim

Wie findet man einen minimalen spannenden Baum mit kleinstem Gewicht?

Idee

Starte mit einem beliebigen Knoten. Der Baum ist zusammenhängend, also muss von diesem Knoten eine Kante ausgehen. Wähle hierfür diejenige mit kleinstem Gewicht. Nun hat man einen Teilgraphen mit zwei Knoten und einer Kante. Von diesem Teilgraphen muss wiederum eine Kante ausgehen zu einem noch nicht enthaltenen Knoten. Wähle hierfür wieder die mit kleinstem Gewicht und setze das Verfahren fort, bis alle Knoten verbunden sind.

Algorithmus von Prim (2)



(a) Start (b) 1. Iteration (c) 2. Iteration (d) 3. Iteration

Algorithmus von Prim (3)

Input: A weighted connected graph $G = (V, E)$ with weight function w.
Output: A minimum spanning tree T of G.

```
1  for each v ∈ V do
2      cost[v] ← ∞
3      parent[v] ← NULL
4  r ← arbitrary vertex of V
5  cost[r] ← 0
6  Q ← V
7  while Q ≠ ∅ do
8      u ← extractMin(Q)
9      for each v ∈ adj(u) do
10         if v ∈ Q and w(u, v) < cost[v] then
11             parent[v] ← u
12             cost[v] ← w(u, v)
13 T ← {(v, parent[v]) | v ∈ V − {r}}
14 return T
```

Beispiel (1)

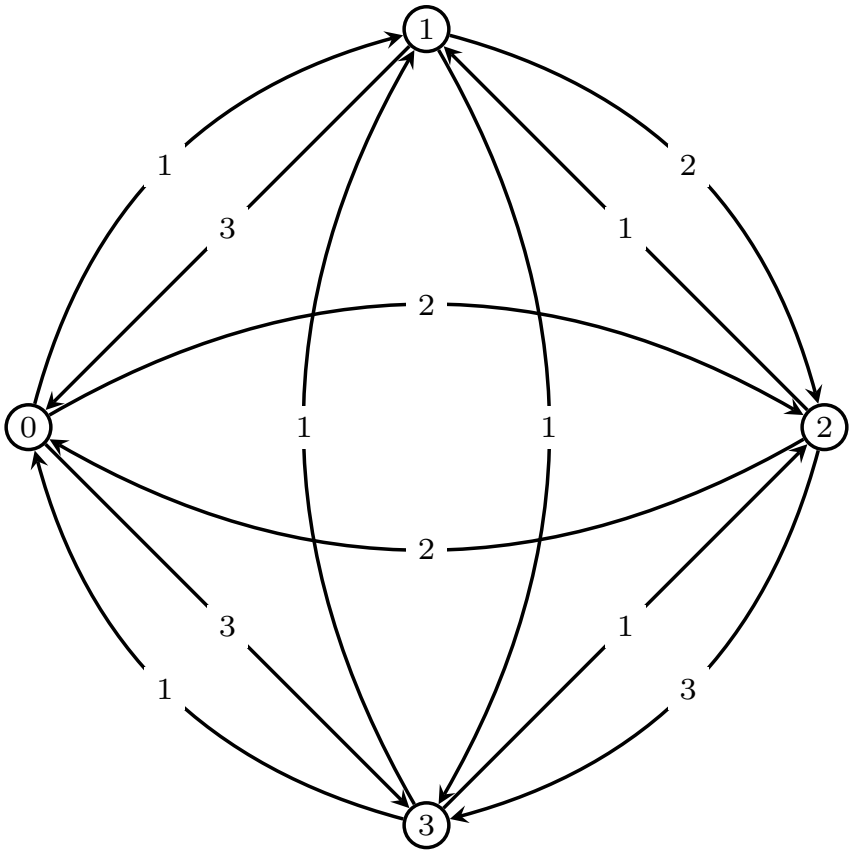


Abbildung 6: Gerichteter Graph

Beispiel (1)

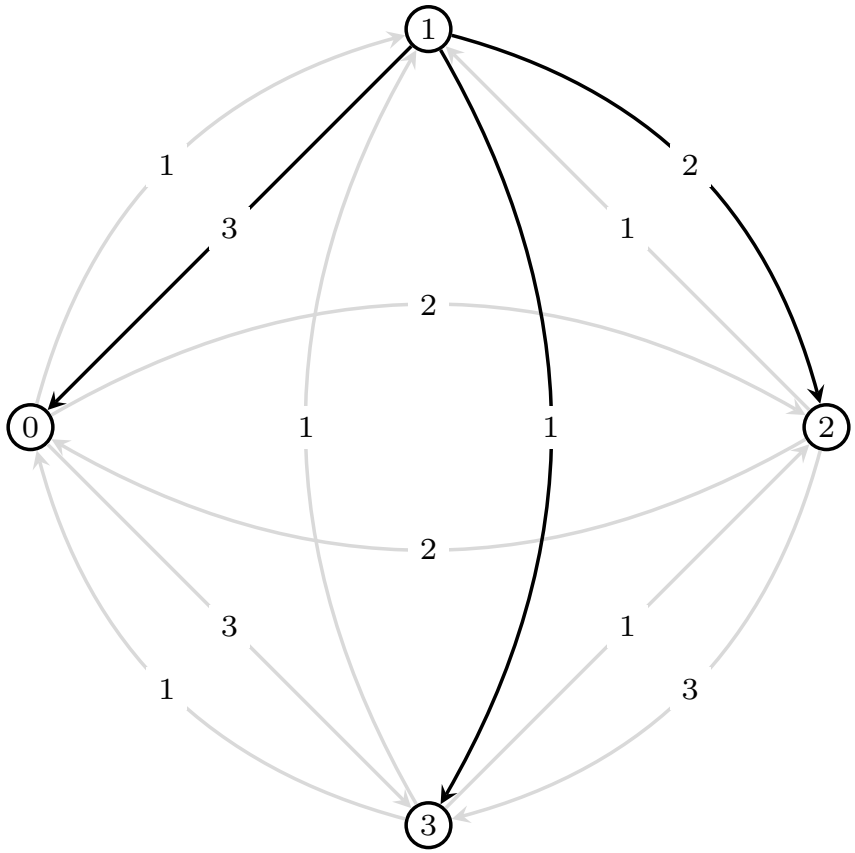


Abbildung 7: Nach der ersten Iteration der `while` Schleife

Beispiel (1)

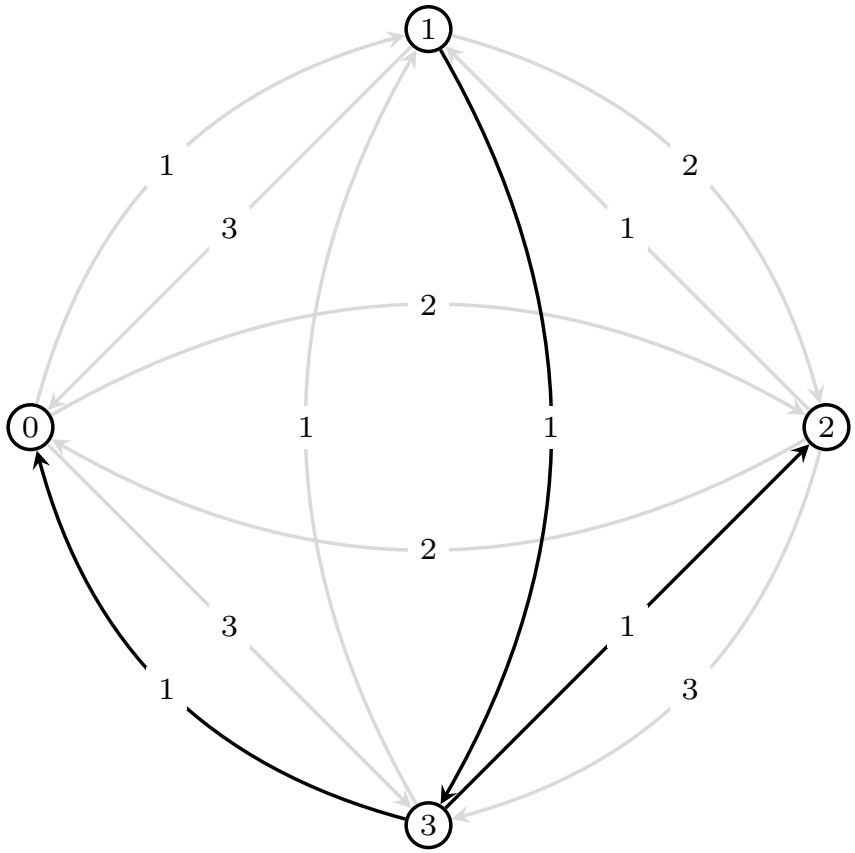


Abbildung 8: Nach der zweiten Iteration der `while` Schleife

Beispiel (1)

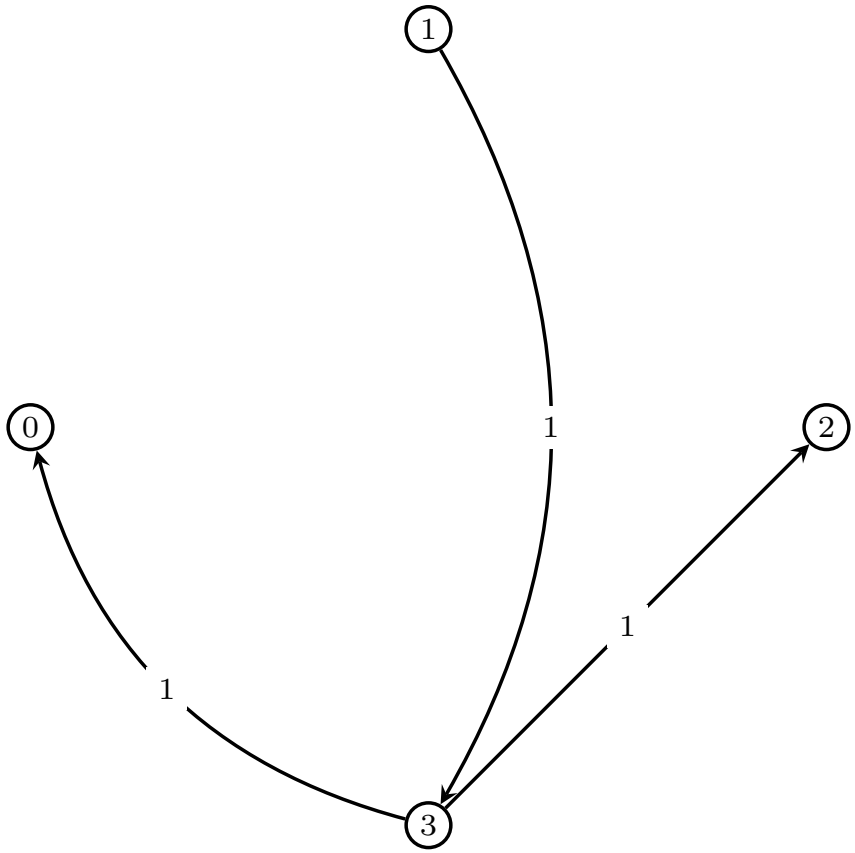


Abbildung 9: Minimaler aufspannender Baum

Beispiel (2)

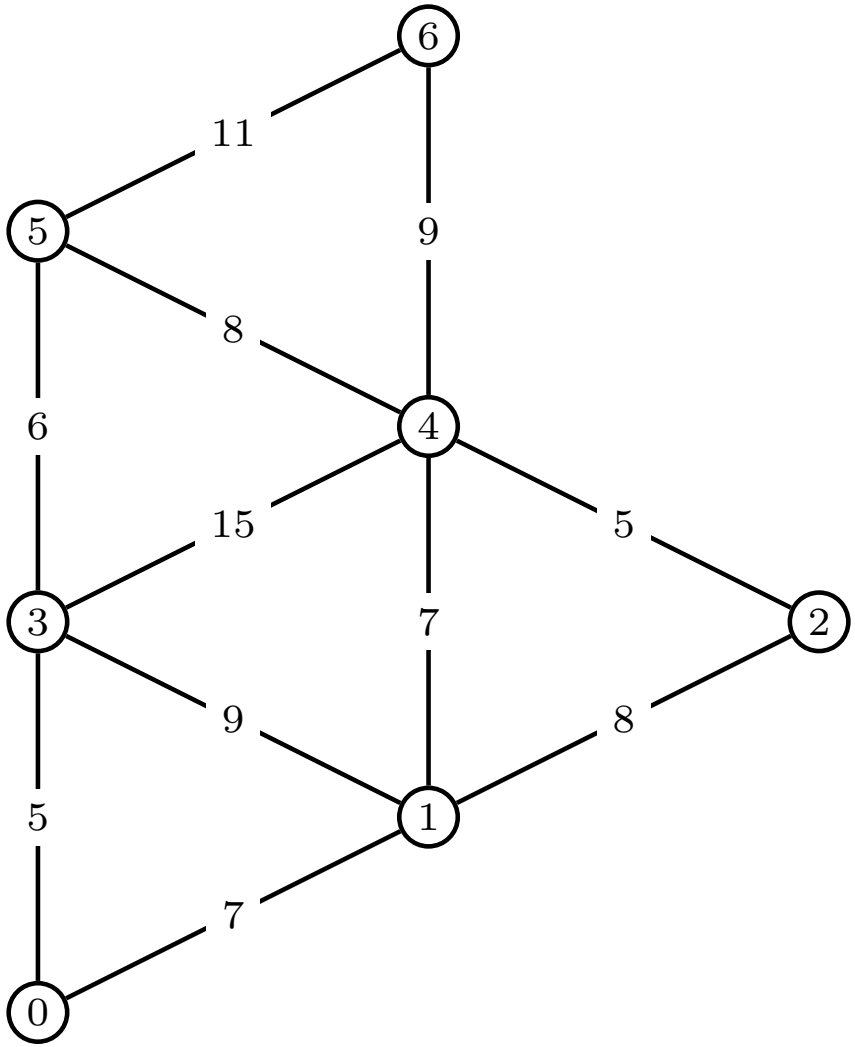


Abbildung 10: Ungerichteter Graph

Beispiel (2)

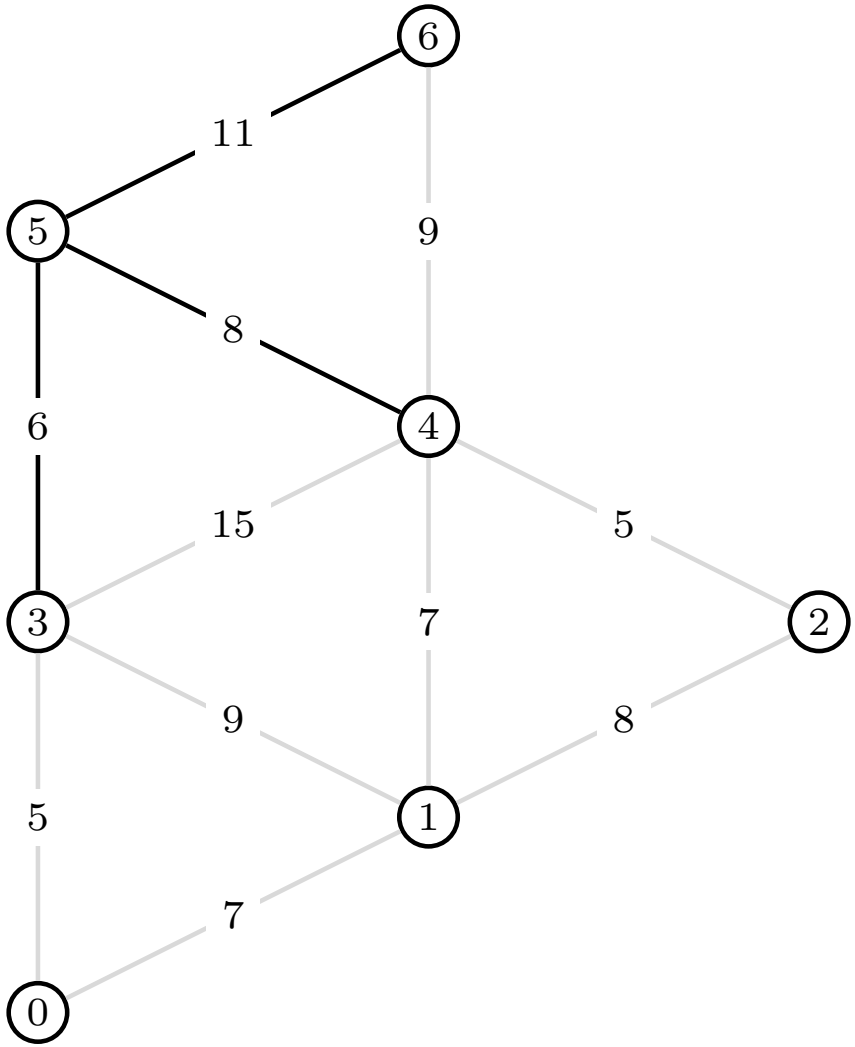


Abbildung 11: Nach der ersten Iteration der `while` Schleife

Beispiel (2)

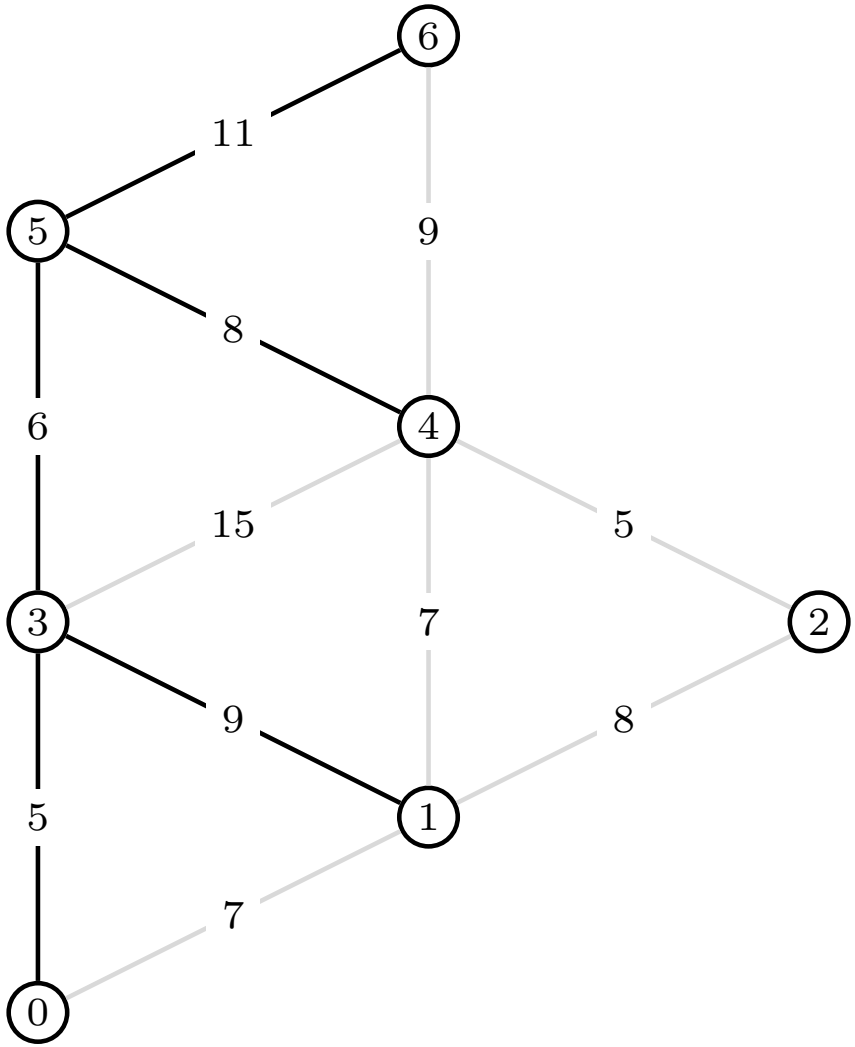


Abbildung 12: Nach der zweiten Iteration der `while` Schleife

Beispiel (2)

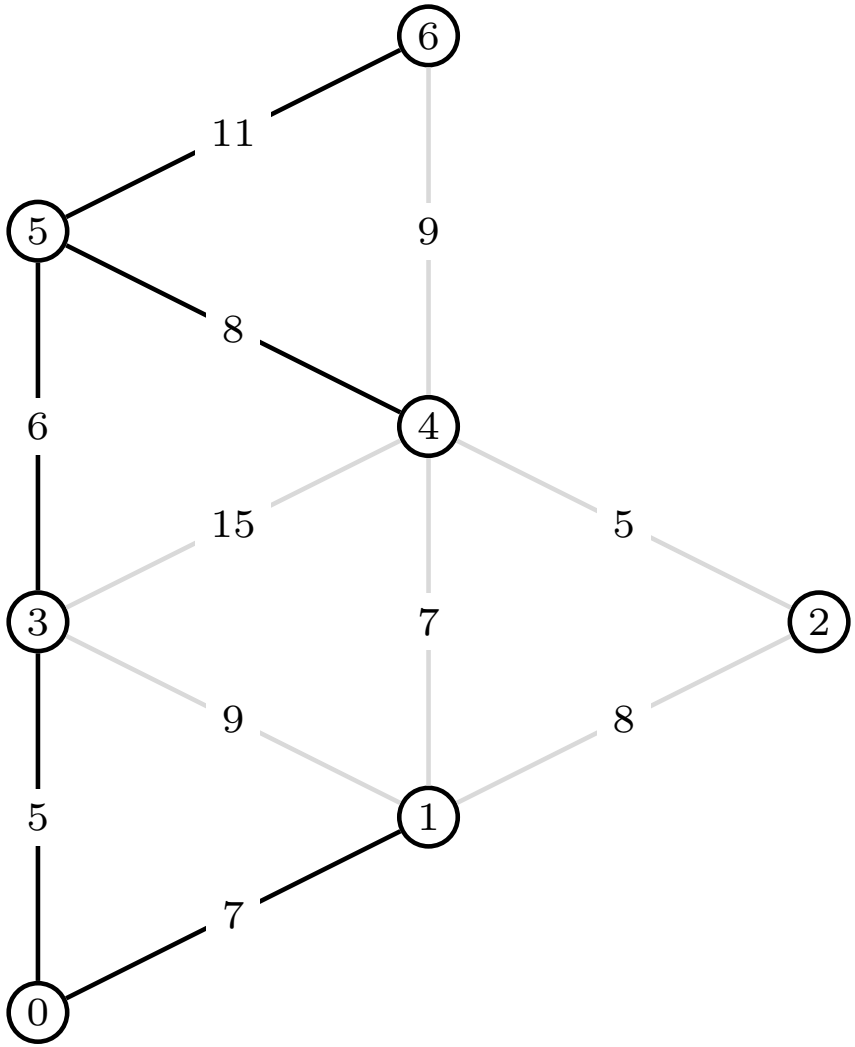


Abbildung 13: Nach der dritten Iteration der `while` Schleife

Beispiel (2)

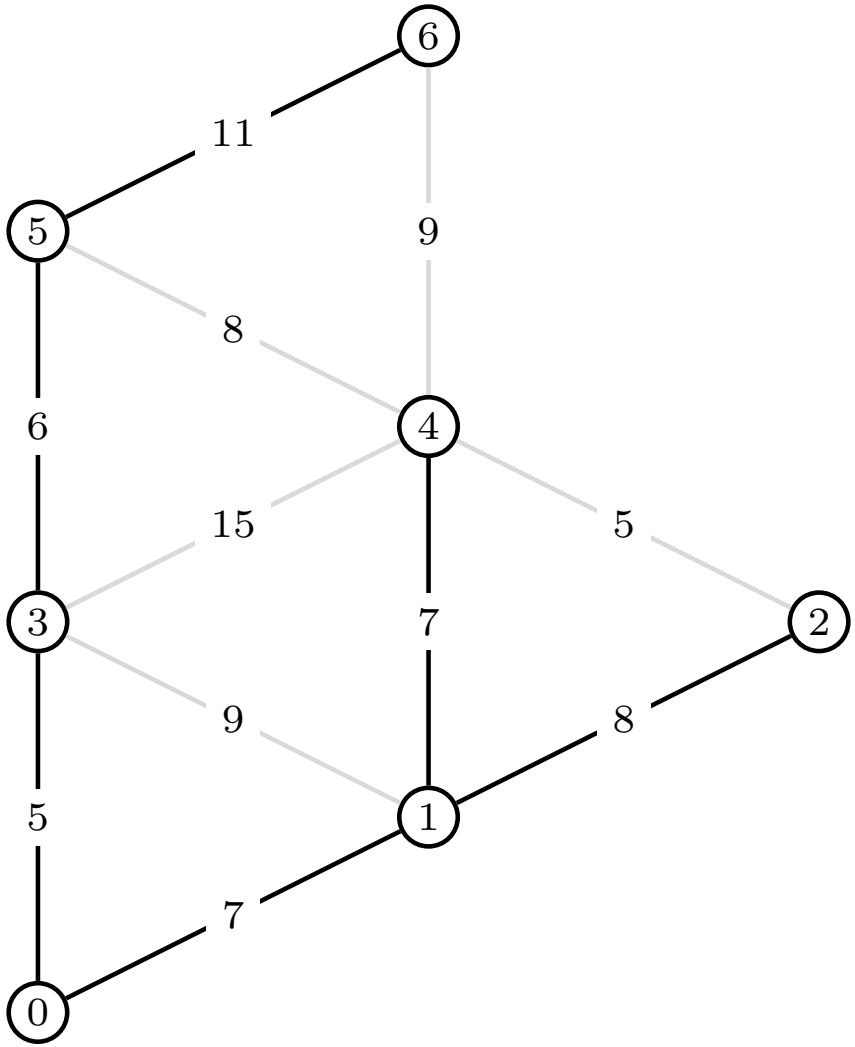


Abbildung 14: Nach der vierten Iteration der `while` Schleife

Beispiel

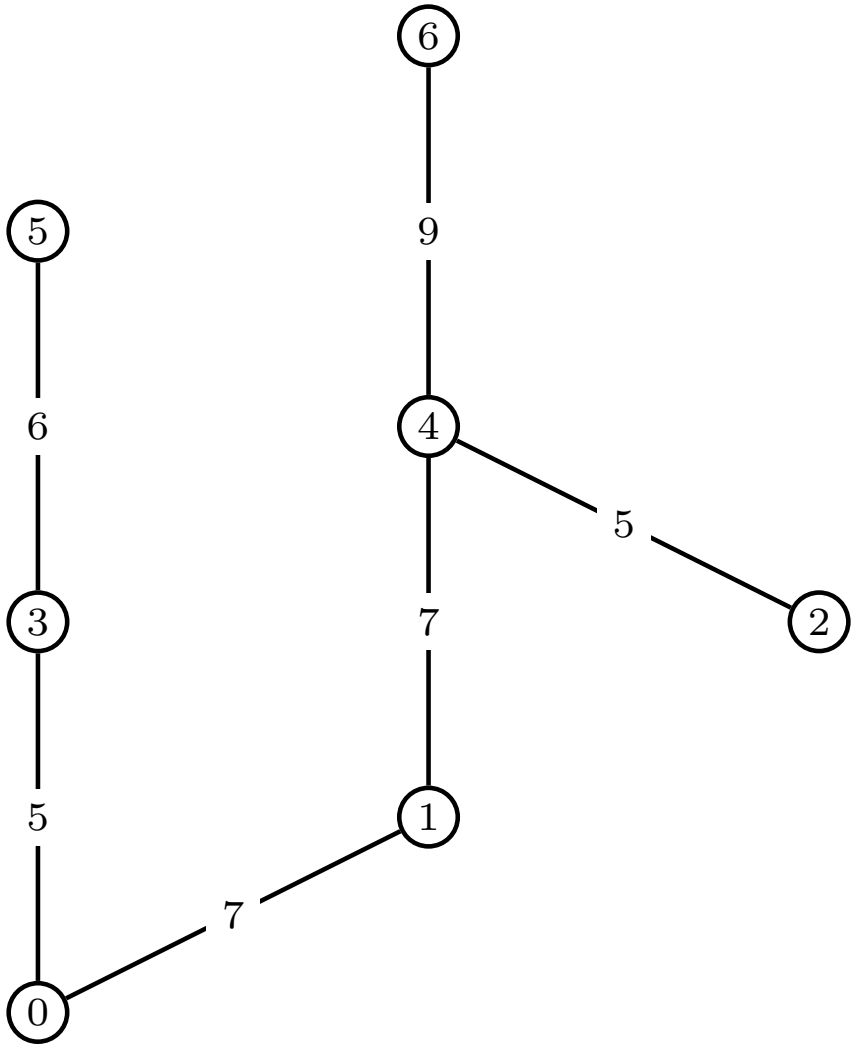


Abbildung 15: Minimaler aufspannender Baum

Wohldefiniertheit

Satz

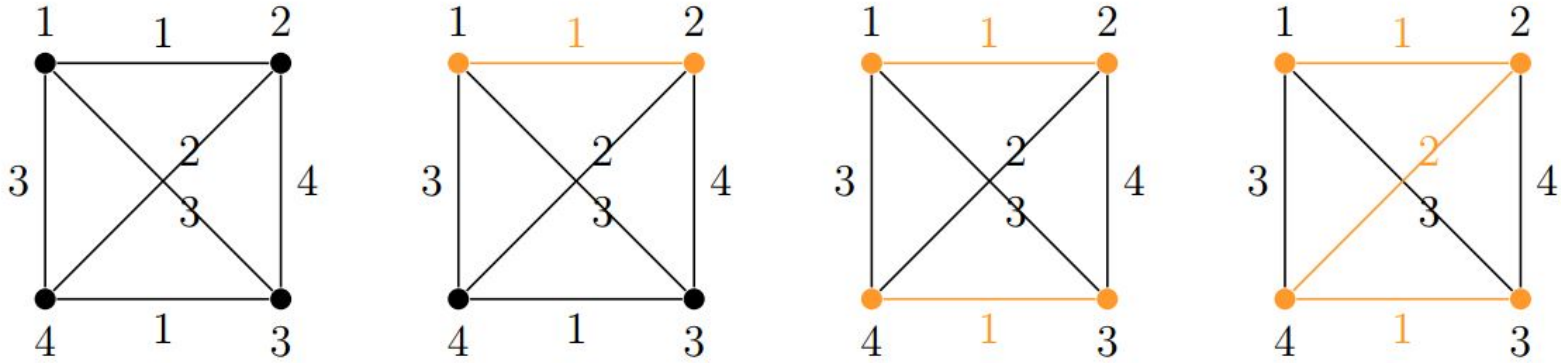
Es sei $G = (V, E)$ ein ungerichteter, gewichteter und zusammenhängender Graph mit $n := |V|$ Knoten und Kantengewichten $w(e)$ für alle $e \in E$. Dann ist der Algorithmus von Prim wohldefiniert und der erzeugte Teilgraph $T := (V_n, E_n)$ ein minimaler spannender Baum von G.

Bemerkung

Der Algorithmus von Prim erzeugt ausgehend von einem Knoten eine immer größere Zusammenhangskomponente mit möglichst kleinem Gewicht. Bei den entstehenden Teilgraphen wird also der Fokus darauf gelegt, dass sie zusammenhängend sind. Dass sie gleichzeitig auch kreisfrei sind, folgt daraus, dass man die Zusammenhangskomponente in jedem Schritt vergrößern will.

Algorithmus von Kruskal (Wiederholung)

Man kann die Prioritäten auch umkehren: Kann man stattdessen eine Folge von Teilgraphen erzeugen, bei denen man immer eine Kante mit möglichst kleinem Gewicht dazu nimmt, solange der Graph kreisfrei bleibt? Die zwischendrin entstehenden Teilgraphen wären dann nicht notwendig zusammenhängend. Der letzte jedoch muss zusammenhängend sein, da man sonst noch eine weitere Kante hinzu nehmen könnte.



(a) Start (b) 1. Iteration (c) 2. Iteration (d) 3. Iteration

Algorithmus von Kruskal

Input: A connected weighted graph $G = (V, E)$ with weight function w.
Output: A minimum spanning tree of G.

```
1 m ← |V|
2 T ← ∅
3 sort E = {e_1, e_2, ..., e_n} by weights so that w(e_1) ≤ w(w_2) ≤ ··· ≤ w(e_n)
4 for i ← 1, 2, ..., n do
5     if e_i ∉ E(T) and T ∪ {e_i} is acyclic then
6         T ← T ∪ {e_i}
7     if |T| = m − 1 then
8         return T
```

Wohldefiniertheit

Satz

Es sei $G = (V, E)$ ein ungerichteter, gewichteter und zusammenhängender Graph mit $n := |V|$ Knoten und Kantengewichten $w(e)$ für alle $e \in E$. Dann ist der Algorithmus von Kruskal wohldefiniert und der erzeugte Teilgraph $T := (V_n, E_n)$ ein minimaler spannender Baum von G.

Borůvka - Algorithmus

- Findet einen minimalen aufspannenden Baum für gewichtete zusammenhängende Graphen $G = (V, E)$ deren Kantengewichte verschieden voneinander sind.
- Zeitkomplexität: $O(m \log n)$ für $n = |V|, m = |E|$.
- In der Initialisierungsphase wird ein aufspanennder Wald erzeugt, der alle Knoten enthält und keine Kanten
- Für jede Zusammenhangskomponente wird die Kante mit minimalem Gewicht gewählt um den Baum zu erzeugen.
- Kann parallelisiert werden!

Borůvka - Algorithmus

Input: A weighted connected graph $G = (V, E)$ with weight function w. All the edge weights of G are distinct.
Output: A minimum spanning tree T of G.

```
1 n ← |V|
2 T ← K̄_n
3 while |E(T)| < n − 1 do
4     for each component T′ of T do
5         e′ ← edge of minimum weight that leaves T′
6         E(T) ← E(T) ∪ e′
7 return T
```

Beispiel: Borůvka - Algorithmus

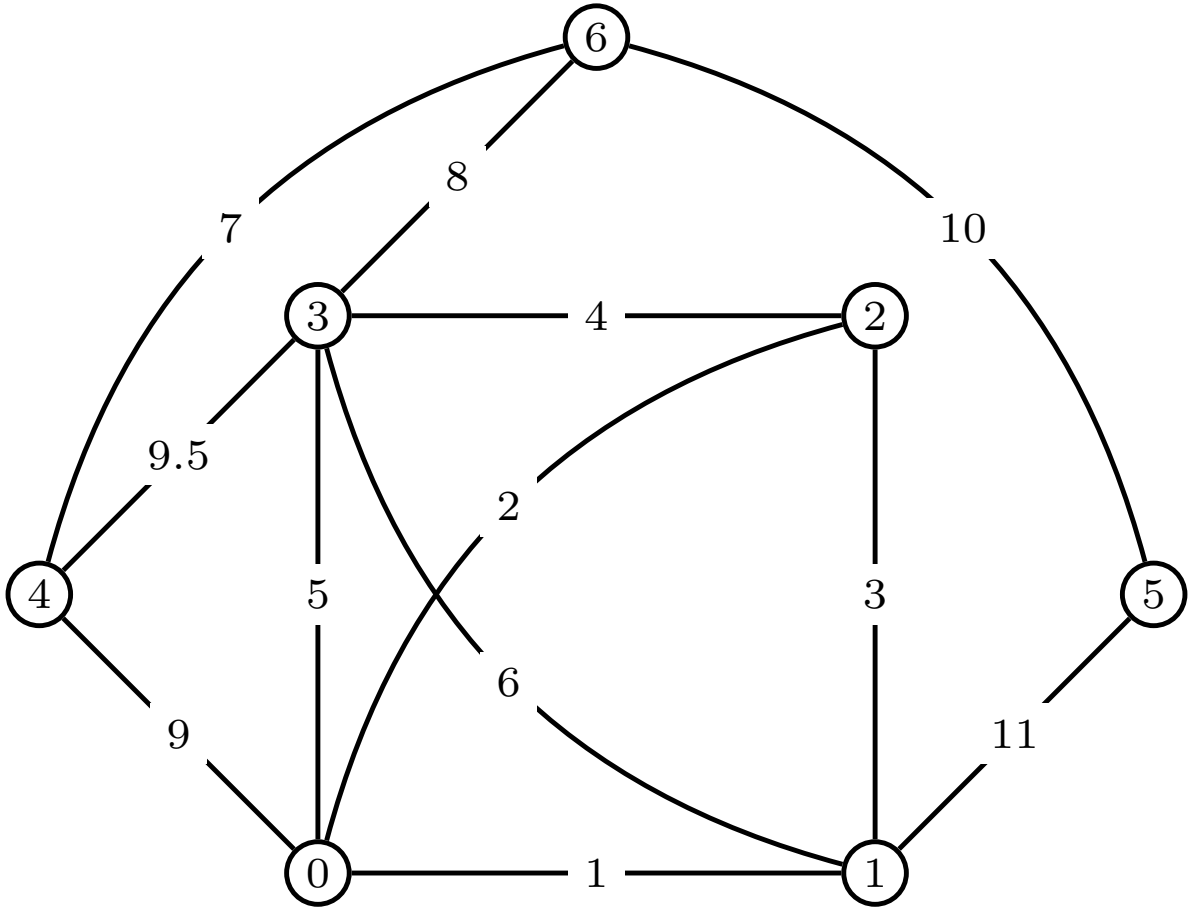


Abbildung 16: Ungerichteter Graph

Beispiel: Borůvka - Algorithmus

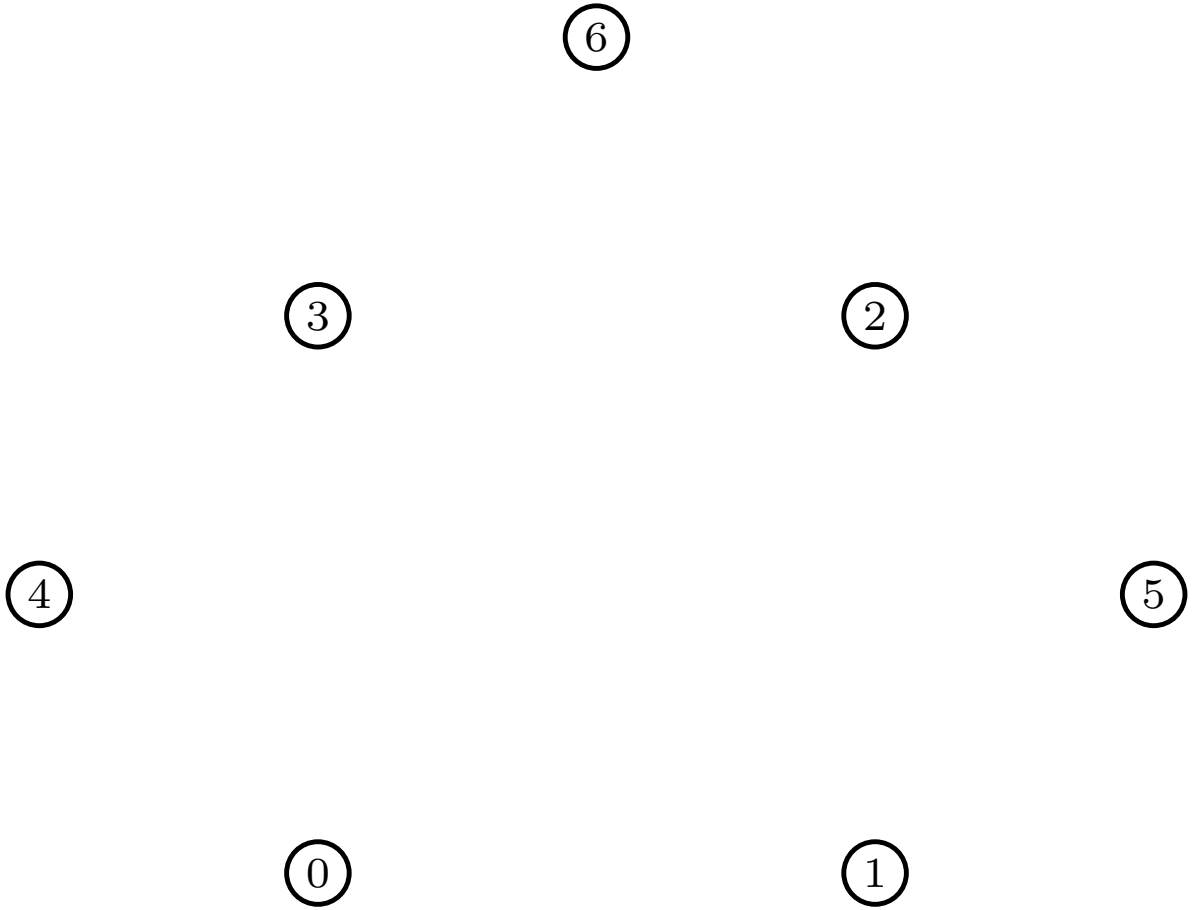


Abbildung 17: Nullte Iteration der `while` Schleife

Beispiel: Borůvka - Algorithmus

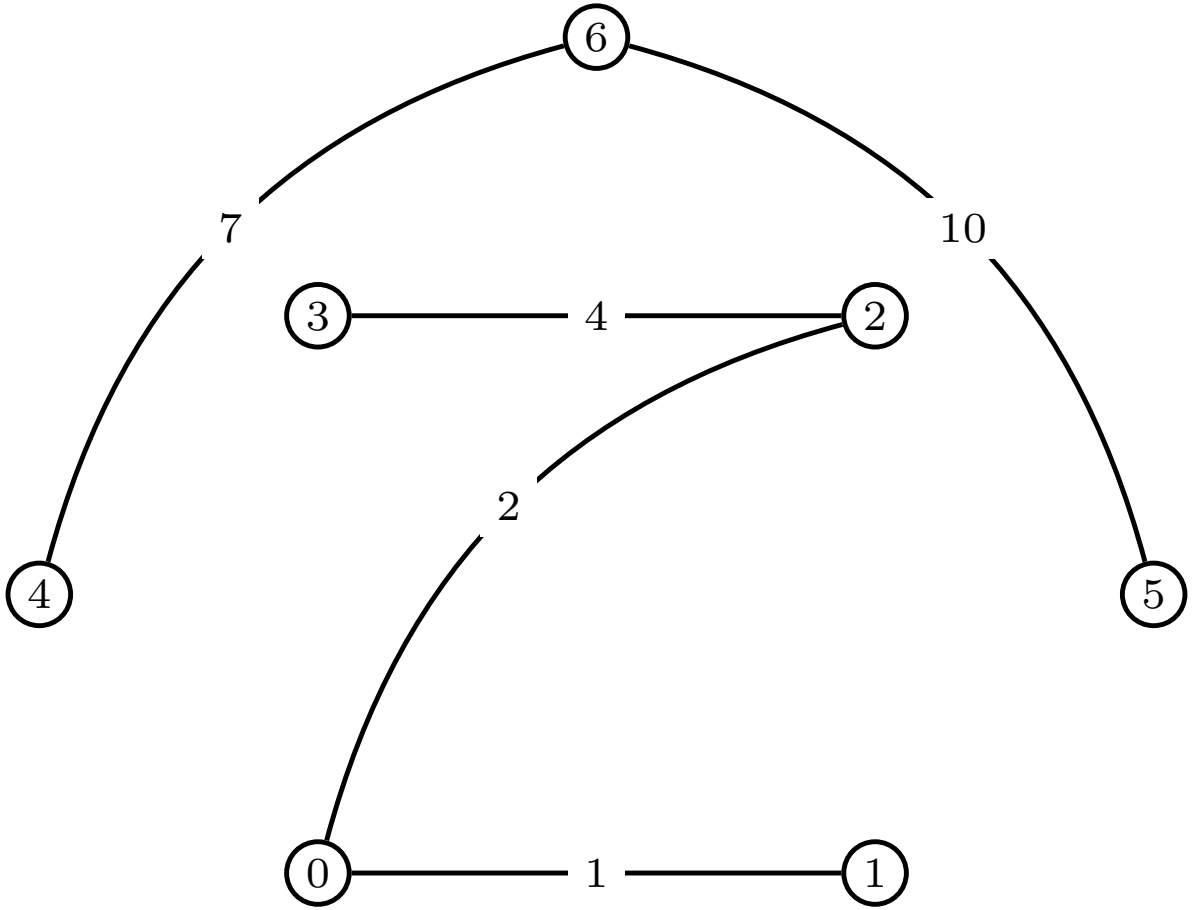


Abbildung 18: Erste Iteration der `while` Schleife

Beispiel: Borůvka - Algorithmus

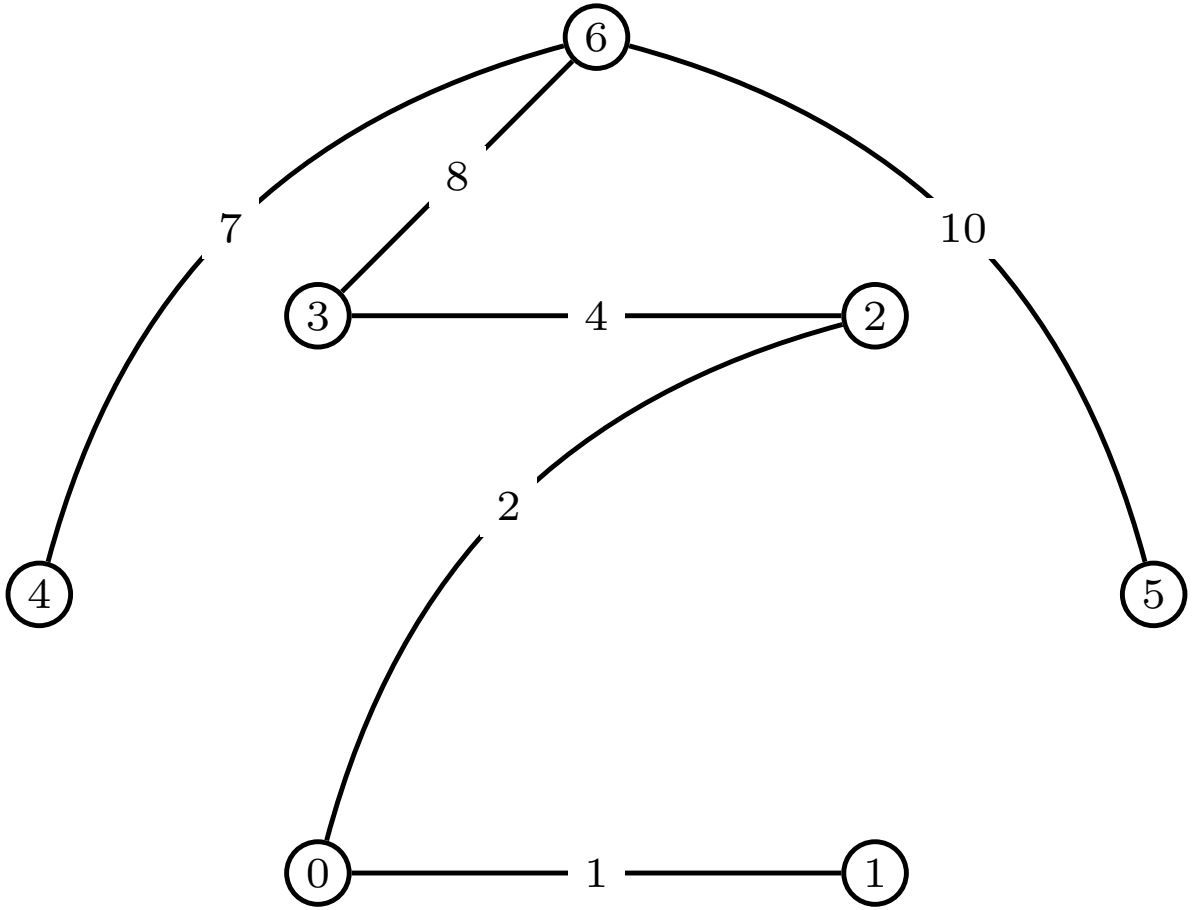


Abbildung 19: Zweite Iteration der `while` Schleife

Zusammenfassung

Die Algorithmen von Prim, Kruskal und Borůvka sind sogenannte Greedy-Verfahren. Als Greedy-Verfahren bezeichnet man einen Algorithmus, der versucht ein Problem zu lösen, indem er in jeder Iteration das tut, was gerade am besten erscheint. Dies muss global gesehen nicht die beste Entscheidung sein und führt auch nicht immer zum Erfolg, wie man an dem folgenden Beispiel sieht. Ein Greedy-Verfahren um von einem Berggipfel herunter zu finden wäre zum Beispiel immer in die Richtung des steilsten Abstiegs zu gehen. Wenn man Glück hat, kommt man so am Fuße des Berges an. Hat man hingegen Pech, so landet man in einer Senke, aus der man mit diesem Ansatz nicht mehr herausfindet. Dass Greedy-Verfahren zur Bestimmung minimaler spannender Bäume funktionieren liegt, etwas vereinfacht, daran, dass man einen minimalen spannenden Baum eines Graphen aus minimalen spannenden Bäumen von Teilgraphen zusammenbauen kann.

Berechnung kürzester Pfade

- Sei $G = (V, E)$ ein Graph mit positiv gewichteten Kanten $w(e) \geq 0$ für alle $e \in E$.
- Die Länge oder die Distanz eines $u - v$ Pfades P von $u \in V$ nach $v \in V$ ist die Summe aller Gewichten aller Kanten von P.
- Sei $d(u, v)$ der kleinste Wert $d(P)$ aller Pfade von u nach v. Ein Pfad P mit $d(P) = d(u, v)$ heißt kürzester Pfad von u nach v.
- Diese Definitionen können auch für gewichtete Graphen mit negative Gewichte übertragen werden.

Berechnung kürzester Pfade

- Es existieren viele Algorithmen, die den kürzesten Pfad in einem gewichteten Graphen berechnen.
- Die Länge des kürzesten Pfades ist immer echt kleiner als die Ordnung des Graphen.

Lemma

Sei v ein Knoten in einem zusammenhängenden Graph $G = (V, E)$ mit Ordnung $n = |V|$. Falls die Kantengewichte alle positiv sind, dann existiert ein kürzester Pfad von v zu jedem anderen Knoten $w \in V$ mit höchstens $n - 1$ Kanten.

Berechnung kürzester Pfade

- Klassifikation der Algorithmen
- Gewicht Bestimmung: durchsucht den Graph und bestimmt Gewichte, die im Durchlauf des Algorithmus nicht mehr verändert werden
- Gewicht Veränderung: ändert die Gewichte im Suchverfahren
  - Kann auch negative Gewichte behandeln, falls das Gewicht eines jeden Zyklus positiv ist.

Allgemeiner Suchalgorithmus für die Berechnung kürzester Pfade

Input: A weighted graph or digraph $G = (V, E)$, where the vertices are numbered as $V = \{1, 2, \ldots, n\}$. A starting vertex s.

Output: A list D of distances from s to all other vertices. A list P of parent vertices such that $P[v]$ is the parent of v.

```
1  D ← [∞, ∞, ..., ∞]                          /* n copies of ∞ */
2  C ← list of candidate vertices to visit
3  while length(C) > 0 do
4      select v ∈ C
5      C ← remove(C, v)
6      for each u ∈ adj(v) do
7          if D[u] > D[v] + w(vu) then
8              D[u] ← D[v] + w(vu)
9              P[u] ← v
10             if u ∉ C then
11                 add u to C
12 return (D, P)
```

Algorithmus von Dijkstra

- Findet den kürzesten Weg in positiv gewichteten Graphen
- Verallgemeinerung der BFS/Breitensuche
- Ist ein Greedy Algorithmus.

Algorithmus von Dijkstra

Input: An undirected or directed graph $G = (V, E)$ that is weighted and has no self-loops. The order of G is $n > 0$. A vertex $s \in V$ from which to start the search. Vertices are numbered from 1 to n, i.e. $V = \{1, 2, \ldots, n\}$.

Output: A list D of distances such that $D[v]$ is the distance of a shortest path from s to v. A list P of vertex parents such that $P[v]$ is the parent of v, i.e. v is adjacent from $P[v]$.

```
1  D ← [∞, ∞, ..., ∞]                    /* n copies of ∞ */
2  D[s] ← 0
3  P ← [ ]
4  Q ← V                                  /* list of nodes to visit */
5  while length(Q) > 0 do
6      find v ∈ Q such that D[v] is minimal
7      Q ← remove(Q, v)
8      for each u ∈ adj(v) ∩ Q do
9          if D[u] > D[v] + w(vu) then
10             D[u] ← D[v] + w(vu)
11             P[u] ← v
12 return (D, P)
```

Beispiel: Algorithmus von Dijkstra

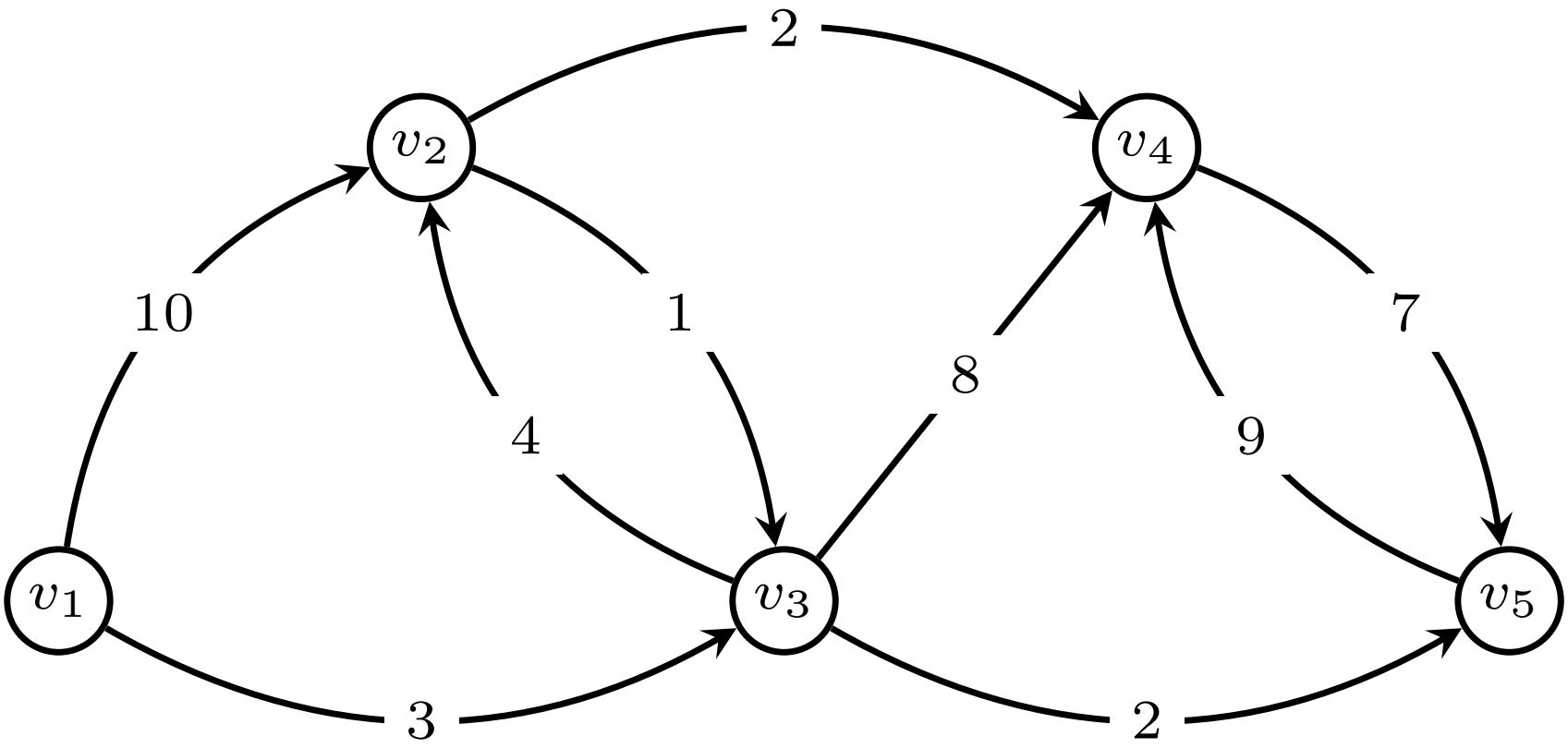


Abbildung 20: Gerichteter Graph

Beispiel: Algorithmus von Dijkstra

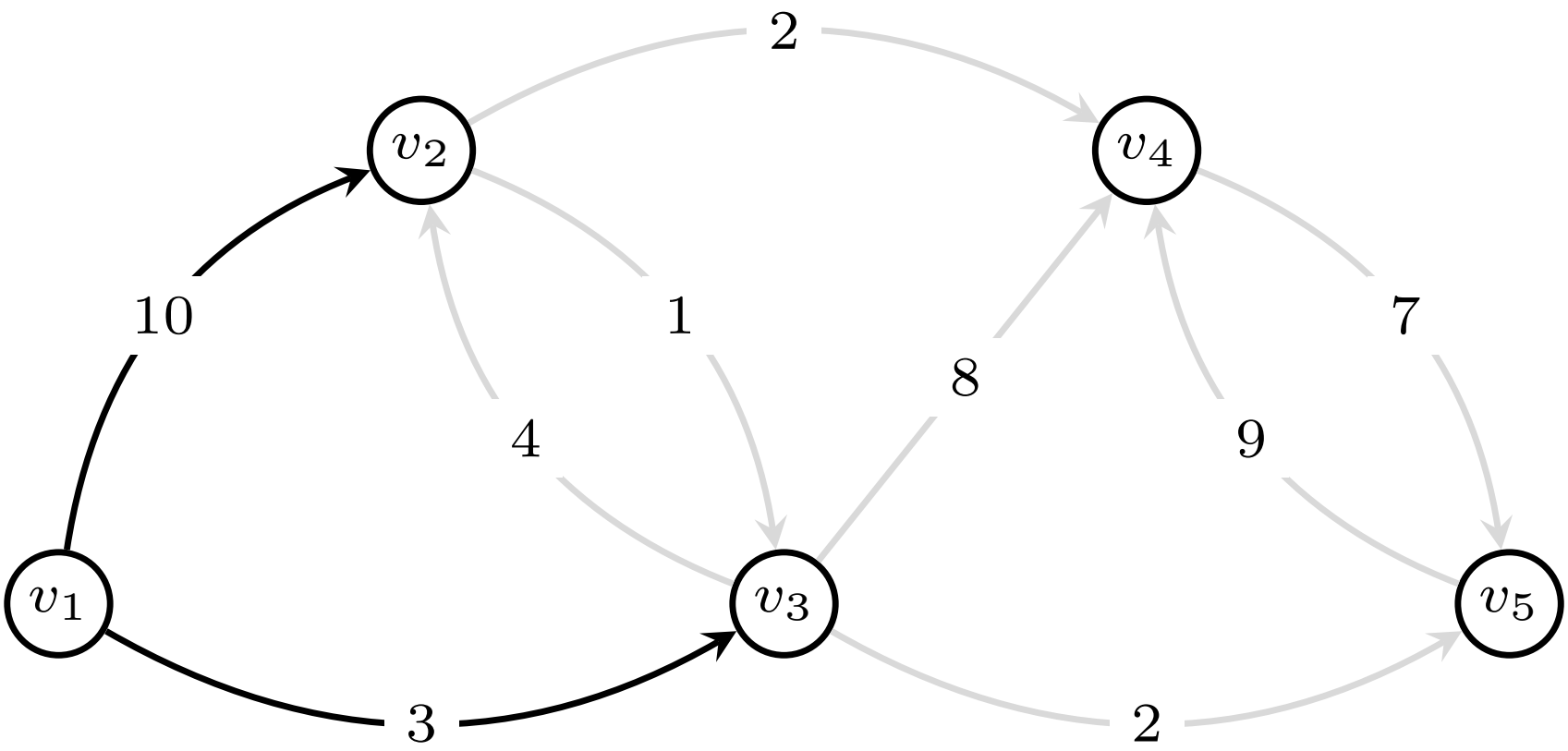


Abbildung 21: Erste Iteration der `while` Schleife

Beispiel: Algorithmus von Dijkstra

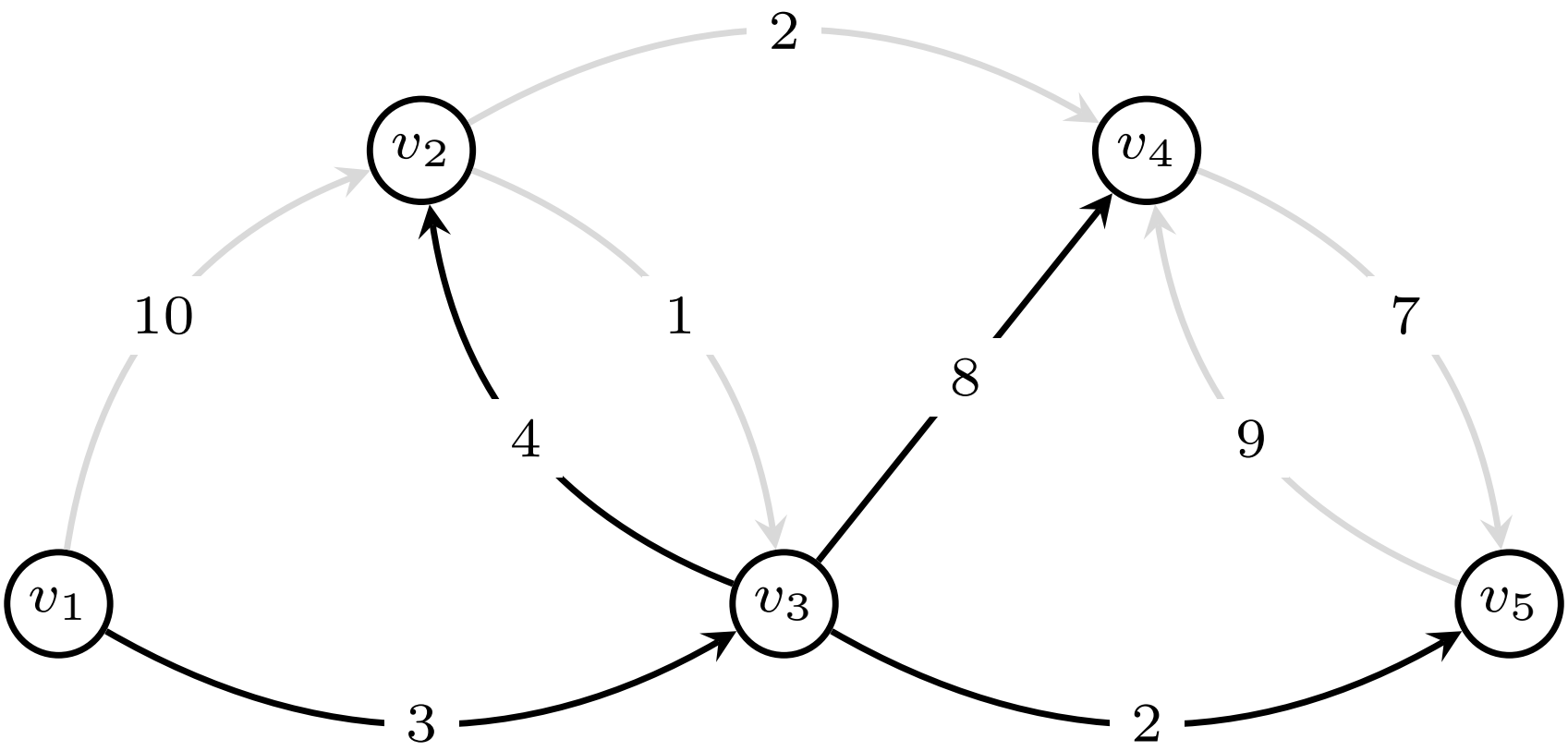


Abbildung 22: Zweite Iteration der `while` Schleife

Beispiel: Algorithmus von Dijkstra

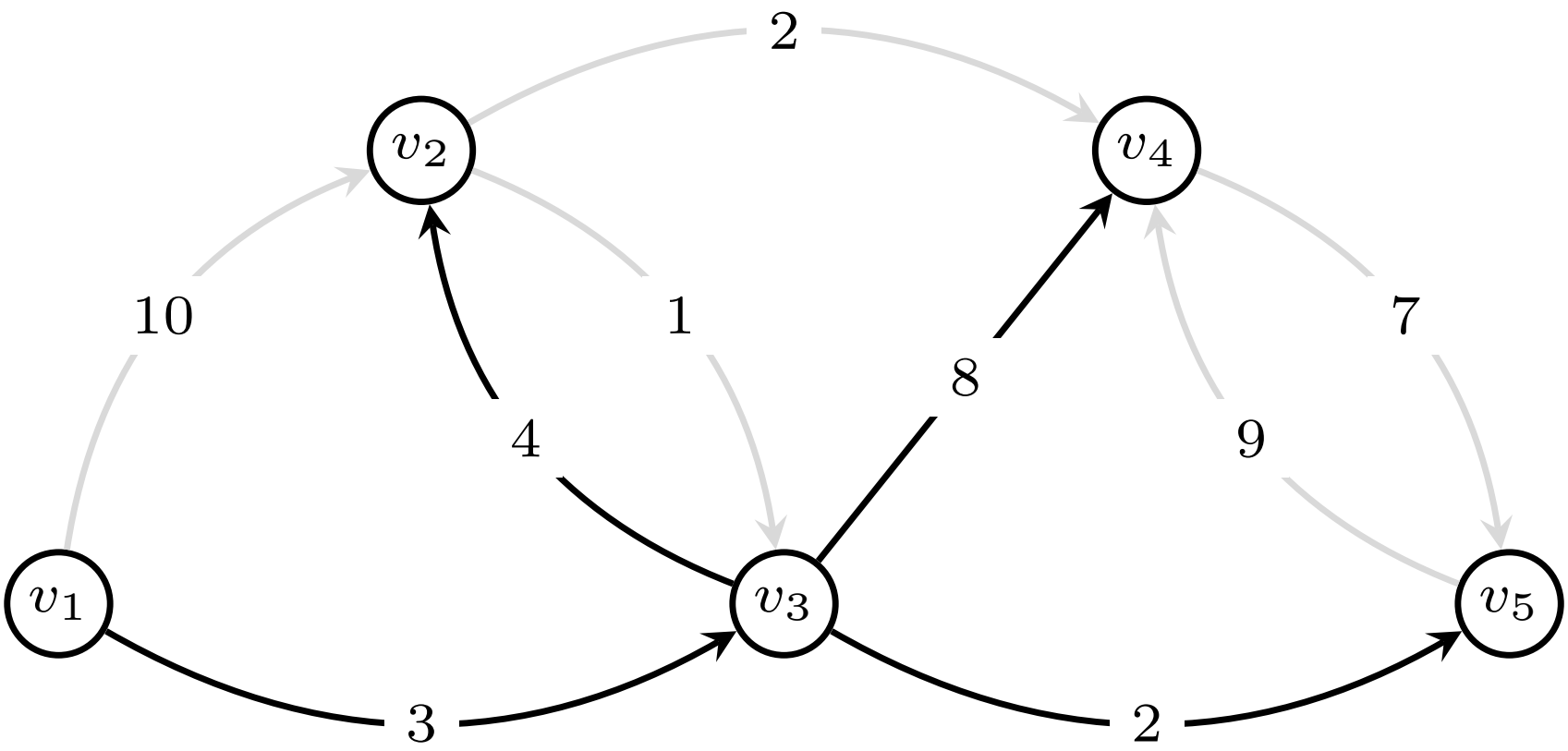


Abbildung 23: Dritte Iteration der `while` Schleife

Beispiel: Algorithmus von Dijkstra

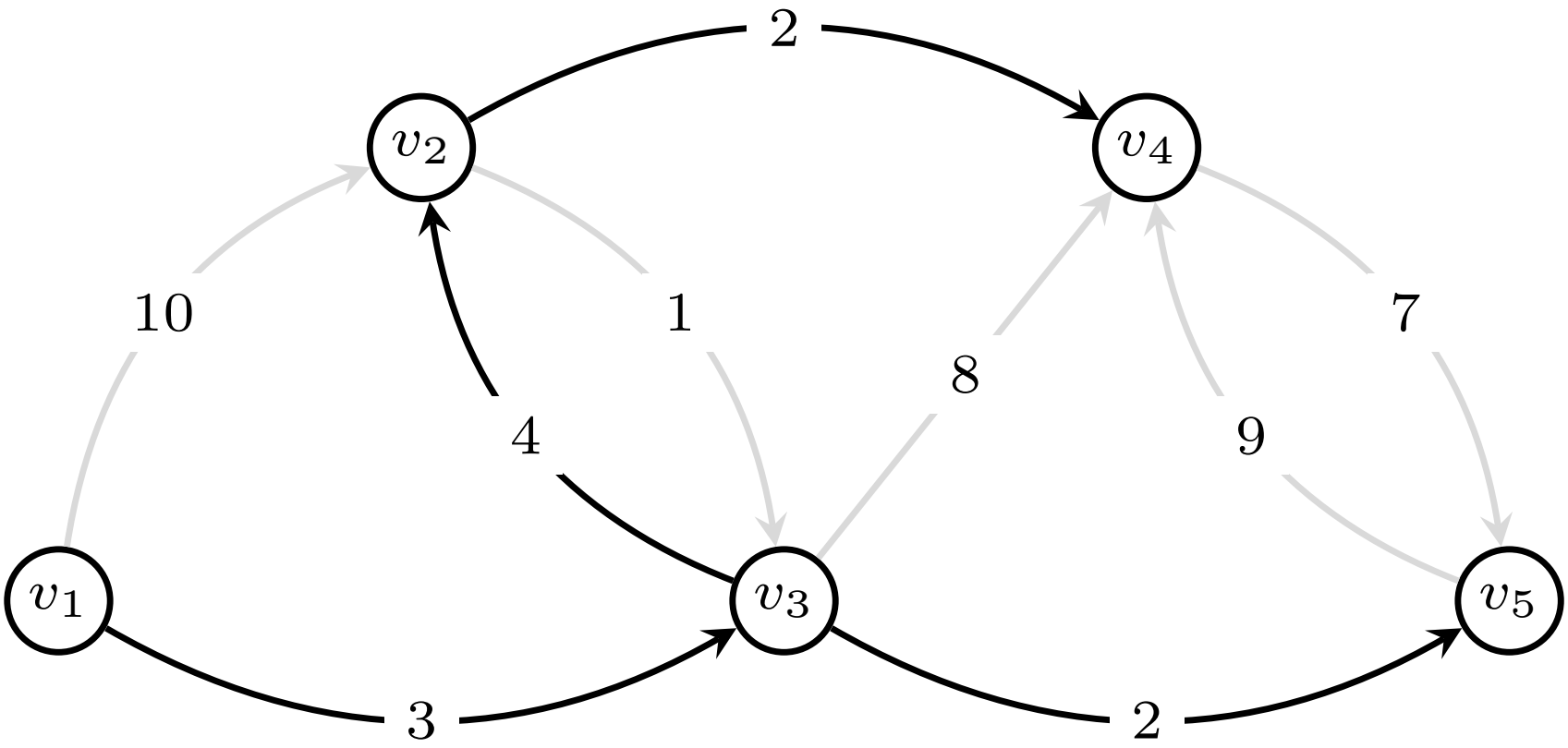


Abbildung 24: Vierte Iteration der `while` Schleife

Beispiel: Algorithmus von Dijkstra

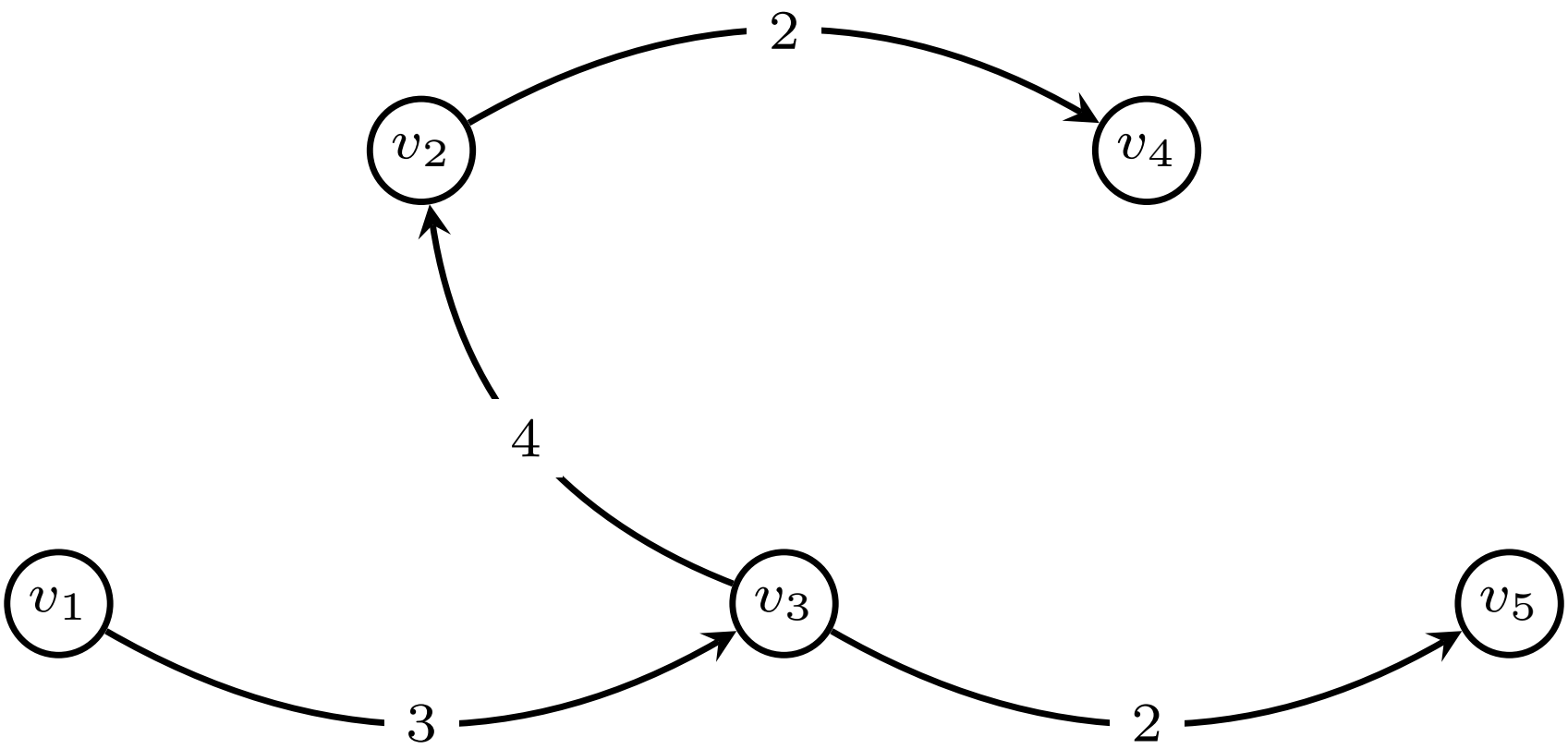


Abbildung 25: Kürzester Weg

Beispiel: Algorithmus von Dijkstra

v_1	v_2	v_3	v_4	v_5
$\underline{(0,-)}$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$	$(\infty,-)$
	$(10,v_1)$	$\underline{(3,v_1)}$	$(11,v_3)$	$\underline{(5,v_3)}$
	$\underline{(7,v_3)}$		$\underline{(9,v_2)}$	

Tabelle 1: Zwischenschritte des Dijkstra Algorithmus.

Beispiel: Algorithmus von Dijkstra

Kürzeste Wege und Entfernungen

$$
\begin{aligned}
v_1\text{-}v_2 &: v_1, v_3, v_2 & d(v_1, v_2) &= 7 \\
v_1\text{-}v_3 &: v_1, v_3 & d(v_1, v_3) &= 3 \\
v_1\text{-}v_4 &: v_1, v_3, v_2, v_4 & d(v_1, v_4) &= 9 \\
v_1\text{-}v_5 &: v_1, v_3, v_5 & d(v_1, v_5) &= 5
\end{aligned}
$$