Baum => Ein Graph in den je zwei Knoten durch genau einen?! Weg verbunden sind.

Wald => Ein Graph dessen Komponente Bäume sind

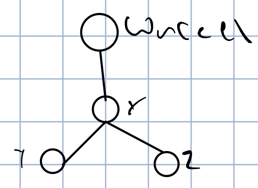
Satz 11.3 (Baum)
Sei G ein Graph mit n Knoten und k Kanten. Dann sind die folgenden Aussagen äquivalent:

(i) G ist ein Baum.

(ii) G ist ein zusammenhängender Graph ohne Kreise.

(iii) G ist zusammenhängend, aber nimmt man irgendeine Kante weg, zerfällt G in zwei Komponenten.

(iv) G ist zusammenhängend und $n = k + 1$ (G hat einen Knoten mehr als Kanten).



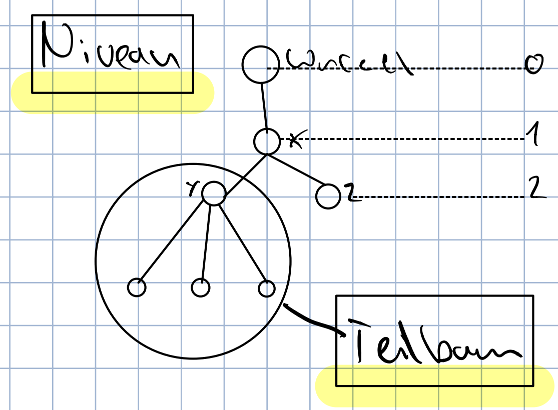
Wurzel => Ein Baum mit besonders ausgezeichneten Knoten

Definition 11.5 (Eltern- und Kindknoten, innere Knoten, Blätter)

(i) Sind die Knoten u und v durch eine Kante verbunden und ist u näher an der Wurzel als v, so heißt u ***Elternknoten*** von v und v ***Kindknoten*** von u.

(ii) Führt ein Weg von u nach v, wobei von u ausgehend zum Kind, Enkel, Urenkel und so weiter gegangen wird, so heißt u ***Vorfahre*** von v und v ***Nachkomme*** von u.

(iii) Knoten mit Nachkommen heißen ***innere Knoten***, Knoten ohne Nachkommen ***Blätter***.



Definition 11.6 (Niveau, Höhe)

(i) Das ***Niveau*** eines Knotens v ist der Abstand zwischen v und der Wurzel.

(ii) Das größte auftretende Niveau ist die ***Höhe*** des Baumes.

Satz 11.7 (Teilbäume)
In einem Wurzelbaum bildet jeder Knoten zusammen mit all seinen Nachkommen und den dazugehörigen Kanten einen ***Teilbaum***, der selbst ein Wurzelbaum ist.

11.2. Aufspannender Baum

Definition 11.8 (aufspannender Baum)
Ein Teilgraph $T(V_T, E_T)$ eines zusammenhängenden Graphen $G(V, E)$ mit den Eigenschaften

(i) T hat die gleichen Knoten wie G, also $V_T = V$,

(ii) T ist ein Baum

heißt ***aufspannender Baum*** des Graphen G.

Ein aufspannender Baum wird gelegentlich auch ***Gerüst*** genannt.

Satz 11.9 (Existenz)
Jeder zusammenhängende Graph enthält einen aufspannenden Baum.

Beweis: Wenn der Graph G ein Baum ist, dann ist nichts zu beweisen.

Ist er kein Baum, enthält er einen Kreis C. Entfernen wir aus C eine Kante e, ist der Teilgraph $G_1 = G_1(V, E \setminus \{e\})$ nach wie vor zusammenhängend.

Entweder ist jetzt G_1 ein aufspannender Baum oder er besitzt noch einen Kreis C_1, den wir wieder durch Entfernen einer Kante zum Weg machen. Dies setzen wir solange fort, bis wir nach endlich vielen Schritt einen Baum erhalten. ■

11.3. Aufspannende Bäume und Suchalgorithmen

11.3.1. Breitensuche oder Breadth-First-Search (BFS)

Die Knoten des Graphen G mit n Knoten werden der Breite nach durchsucht und die Knoten werden aufgezählt.

Algorithmus 11.10 (Breitensuche oder Breadth-First-Search (BFS))

1. Starte mit einem beliebigen Knoten v. Dieser erhält die Nummer 1 und wird aktueller Knoten.
2. Der aktuelle Knoten habe die Nummer i. Es seien bereits die Nummern $1, \dots, r$ vergeben.

   Falls $r = n$, dann STOP: Ein ***aufspannender Baum*** ist gefunden.
3. Ansonsten besuche die noch nicht nummerierten Nachbarn von i. Sie erhalten sukzessive die Nummern $r+1, r+2, \dots$ Füge die zugehörigen Kanten $[i,(r+1)], [i, r+2], \dots$ zum Baum hinzu.
4. Falls Knoten $i+1$ existiert, wird er aktueller Knoten. Weiter bei Schritt 2.

   Falls Knoten $i+1$ nicht existiert, dann STOP: G ist ***nicht zusammenhängend***.

Beispiel : BFS

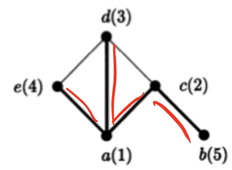


Abbildung 11.6.: Der abgebildete Graph ist zusammenhängend und hat $n = 5$ Knoten.

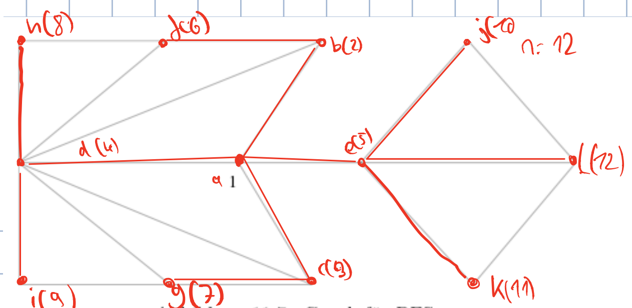


Abbildung 11.7.: Graph für BFS.

11.3.2. Tiefensuche oder Depth-First-Search (DFS)

Andererseits können wir mit einer Tiefensuche zuerst ganz bis zu einem Blatt absteigen, dann wieder auftauchen und uns so von links nach rechts durch einen Wurzelbaum tasten.

Die Knoten des Graphen G mit n Knoten werden der Tiefe nach durchsucht und die Knoten werden aufgezählt.

Algorithmus 11.11 (Tiefensuche oder Depth-First-Search (DFS))

1. Starte mit einem beliebigen Knoten v. Dieser erhält die Nummer 1 und wird aktueller Knoten.
2. Der aktuelle Knoten habe die Nummer i. Es seien bereits die Nummern $1, \dots, r$ vergeben.

   Falls $r = n$, dann STOP: Ein ***aufspannender Baum*** ist gefunden.
3. Ansonsten untersuche die noch nicht nummerierten Nachbarn von i.

   Fall 1: Es gibt noch nicht nummerierte Nachbarn von i.

   Ein nicht nummerierter Knoten erhält die Nummer $r+1$. Füge die Kante $[i, r+1]$ zum Baum hinzu. Knoten $r+1$ wird aktueller Knoten, i wird Vorgängerknoten.

   Weiter bei Schritt 2.
4. Fall 2: Es gibt keine nicht nummerierten Nachbarn von i mehr.

   Fall 2.1: $i > 1$

   Gehe zurück zum Vorgängerknoten von i. Dieser wird aktueller Knoten.

   Weiter bei Schritt 2.

   Fall 2.2: $i = 1$

   STOP: G ist ***nicht zusammenhängend***.

Beispiel : DFS

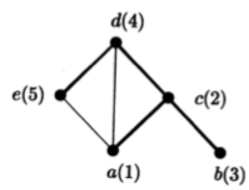


Abbildung 11.10.: Der abgebildete Graph ist zusammenhängend und hat 5 Knoten.

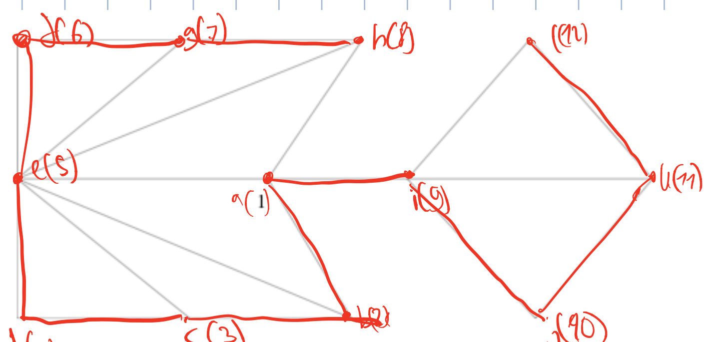


Abbildung 11.11.: Graph für BFS.

11.4. Bewertete Graphen und kürzeste Wege

11.4.1. Grundlegende Definitionen

Definition 11.12 (Bewerteter Graph)
Gibt es zur Kantenmenge E eines Graphen $G(V,E)$ eine Funktion $\omega : E \to \mathbb{R}$, wird ω eine ***Kantenbewertung*** und $G(V,E,\omega)$ ein ***bewerteter Graph*** genannt.

Man spricht auch von ***Kantengewichten*** ω und ***gewichteten Graphen*** $G(V,E,\omega)$.

Definition 11.13 (Minimal aufspannender Baum)
In einem zusammenhängenden und bewerteten Graphen $G(V,E,\omega)$ heißt ein aufspannender Baum T mit minimalem Gewicht

$$\omega(T) = \sum_{k \in E} \omega(k) = \min$$

minimaler aufspannenender Baum.

Stellt ein gegebener bewerteter Graph ein Kommunikationsnetz oder einen Straßenplan dar, taucht die Frage nach einem optimalen Schaltplan oder einem Streckenplan mit minimalen Kosten auf. Dabei soll jede Schaltstation mit jeder anderen kommunizieren. Das ergibt die Frage nach minimalen aufspannenden Bäumen.

Der erste Gedanke könnte sein, einfach alle aufspannenden Bäume eines Graphen zu suchen und dann jenen mit minimalem Gewicht auszusuchen. Dass dies nicht praktikabel ist, sieht man am folgenden Beispiel.

Algorithmus 11.14 (Kruskal-Algorithmus)
Gegeben ist ein zusammenhängender und bewerteter Graph $G(V,E,\omega)$ mit n Knoten. Gesucht ist ein minimaler aufspannender Baum T.

Algorithmus:

1. Initialisierung: Setze den Baum $T = \emptyset$ und den Zählindex $i = 0$.
2. Wähle die noch nicht besuchte Kante mit dem kleinsten Gewicht.

   Schließt diese einen Kreis in T, wird sie verworfen. Ansonsten wird sie T hinzugefügt und $i = i + 1$ gesetzt.
3. Ist $i = n - 1$ dann STOP: Ein minimaler aufspannender Baum ist gefunden.

   Ansonsten wiederhole Schritt 2.

Beispiel : Kruskal

Wende den Kruskal-Algorithmus auf das Eingangsbeispiel eines bewerteten Graphen an:

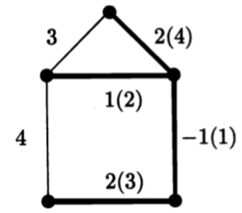
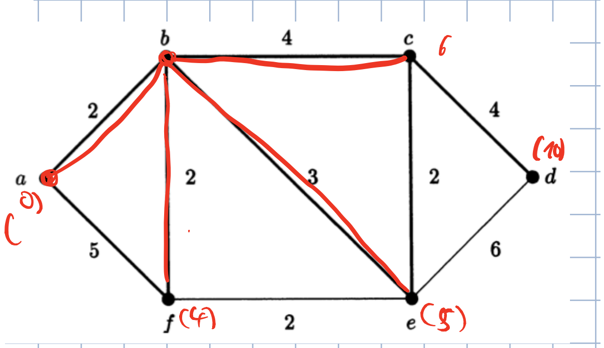


Abbildung 11.15.: Der minimale aufspannende Baum enthält die Kanten mit den Zahlen in Klammern, die die Besuchsreihenfolge angeben.

Satz 11.15 (Kruskal-Algorithmus)
Für einen zusammenhängenden und bewerteten Graphen $G(V,E,\omega)$ bestimmt der Kruskal-Algorithmus einen minimalen aufspannenden Baum.

11.4.3. Dijstra-Algorithmus zum Finden kürzester Wege



Der Kruskal-Algorithmus beantwortet die Frage, wie man in einem Graphen möglichst schnell oder preiswert von jedem Knoten zu jedem beliebigen anderen Knoten gelangen kann.

Definition 11.16 (Gewichtete Länge)
Für einen bewerteten Graphen $G(V, E, \omega)$ mit positiven Gewichten, $\forall e \in E: \ \omega(e) \geq 0$, ist die ***gewichtete Länge*** eines Weges $P(u, v) = uv_1v_2 \dots v_nv$ definiert als die Summe

$$\ell(P) = \sum_{e \in P(u,v)} \omega(e)\,.$$

Algorithmus 11.17 (Dijkstra-Algorithmus) Kanten

1. Initialisierung: Wähle $u_0 = u$, $V_0 = \{u_0\}$, $E_0 = \underline{\emptyset}$, $\ell(u_0) = 0$.
2. Seien die Knoten $V_i = \overset{\text{Knoten}}{\{u_0, u_1, \dots, u_i\}}$ und die Kanten $E_i = \overset{\text{Kanten}}{\{e_1, \dots, e_i\}}$ bereits berechnet.

   Falls $i = n - 1$, dann STOP: Der gesuchte Baum ist gefunden.
3. Ansonsten bestimme für alle Kanten $e = [v, w]$ mit $v \in V_i$ und $w \in V \setminus V_i$ (also Kanten mit bereits besuchten Anfangsknoten und noch unbesuchten Endknoten) die Zahl
   $$f(e) = \ell(v) + \omega(e)$$
   und wähle die Kante e^*, für die $f(e)$ minimal ist, $f(e^*) = \min f(e)$.
4. Mit $e^* = [v^*, w^*]$ setze
   $$\begin{array}{lll} u_{i+1} = w^* & V_{i+1} = V_i \cup \{u_{i+1}\} & \ell(u_{i+1}) = f(e^*) \\ e_{i+1} = e^* & E_{i+1} = E_i \cup \{e_{i+1}\} & i \mapsto i + 1 \end{array}$$
   und wiederhole Schritt 2.

$l(a) = 0$

$l(b) = 2$

$l(f) = 4$

$l(e) = 5$

$l(c) = 6$

$l(d) = 10$

$V = \{a, b, f, e, c, d\}$

$E = \{[a,b], [b,f], [b,e] [b,c] [c,d]\}$

Satz 11.18 (Dijkstra-Algorithmus)
Für einen zusammenhängenden bewerteten Graphen $G(V, E, \omega)$ mit n Knoten und positiven Gewichten, $\forall e \in E: \ \omega(e) \geq 0$, sowie einen fest vorgegebenen Knoten u bestimmt der Dijkstra-Algorithmus einen aufspannenden Baum, so dass der Weg von u zu jedem anderen Knoten $v \in V$ jeweils minimale gewichtete Länge hat.