6.4 Topologisches Sortieren

In vielen Anwendungen ist eine Halbordnung von Elementen durch eine binäre Relation gegeben, die unmittelbar durch einen zyklenfreien gerichteten Graphen $G = (V, E)$ dargestellt werden kann. Die Knoten V des Graphen entsprechen den Elementen und die Kanten $(u, v) \in E$ beschreiben Beziehungen wie „u kommt vor v“ oder „u ist kleiner als v“. Gesucht ist eine vollständige Ordnung (d.h. eine lineare Anordnung oder Nummerierung) der Elemente V, die mit der gegebenen Relation (E) verträglich ist. Diesen Vorgang nennt man *topologisches Sortieren*.

Beispiel: Anziehen von Kleidungsstücken

- Elemente: Hose, Mantel, Pullover, Socken, Schuhe, Unterhemd, Unterhose
- Relationen:
  - Unterhemd vor Pullover
  - Unterhose vor Hose
  - Pullover vor Mantel
  - Hose vor Mantel
  - Hose vor Schuhe
  - Socken vor Schuhe

Die Veranschaulichung dieser Beziehungen als sogenannter Relationsgraph zeigt Abbildung 6.3.

Eine topologische Sortierung beschreibt nun eine gültige (sequentielle) Folge der Kleidungsstücke um sie anzuziehen, beispielsweise:

(Unterhose, Unterhemd, Hose, Pullover, Socken, Schuhe, Mantel)

Formal sei die topologische Sortierung durch eine Abbildung $ord : V \to 1, \ldots, n$ mit $n = |V|$ notiert. Das heißt für unser Beispiel: $ord(\text{Unterhose}) = 1$, $ord(\text{Unterhemd}) = 2$ usw.

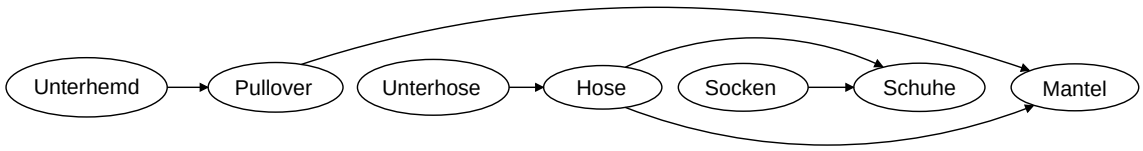

Abbildung 6.3: Beispiel eines Relationsgraphen

Allgemein kann es aber durchaus mehrere gültige Ordnungen geben, so würde in unserem Beispiel auch die Ordnung (Unterhemd, Pullover, Socken, Unterhose, Hose, Schuhe, Mantel) obige Beziehungen erfüllen.

Lemma: Es gibt zumindest eine gültige topologische Ordnung, wenn die Beziehungen zyklenfrei sind, d.h., der Relationsgraph kreisfrei ist.

Beweis: Dies kann man durch Induktion über die Knotenanzahl zeigen. Falls $|V| = 1$, dann existiert offensichtlich die topologische Sortierung, in der dem einen Element $u \in V$ $ord(u) = 1$ zugewiesen ist. Falls $|V| > 1$, so betrachtet man einen Knoten v mit $d^-(v) = 0$ (Eingangsgrad 0, d.h. ohne eingehende Kanten) und definiert $ord(v) = 1$. Auf Grund der Zyklenfreiheit muss ein solcher Knoten immer existieren. Danach entfernt man v, wodurch ein um einen Knoten verkleinerter Graph entsteht. An dessen topologische Sortierung wird v vorne angefügt.

Aus diesem Prinzip ergibt sich auch unmittelbar ein grundsätzlicher Ansatz zum topologischen Sortieren:

1. Suche einen Knoten $v \in V$ mit $d^-(v) = 0$
2. Füge v in der topologischen Sortierung als nächstes Element hinzu
3. Lösche v mit allen inzidenten Kanten aus dem Graphen
4. Solange der Graph nicht leer ist, gehe zu (1)

Es ist noch zu klären, wie jeweils ein Knoten $v \in V$ mit $d^-(v) = 0$ effizient gefunden werden kann. Eine Möglichkeit wäre, bei einem beliebigen Knoten zu beginnen und Kanten rückwärts zu verfolgen. Auf Grund der Zyklenfreiheit würde man hiermit nach spätestens $O(n)$ Schritten zu einem Knoten mit Eingangsgrad 0 kommen.

Effizienter ist es jedoch, den jeweils aktuellen Eingangsgrad zu jedem Knoten zu speichern und auf dem aktuellen Stand zu halten. Dann genügt es, anstatt einen Knoten aus G tatsächlich zu entfernen, die Eingangsgrade seiner direkten Nachfolger zu verringern. Um einen Knoten mit Eingangsgrad 0 schnell zu finden, verwalten wir die Menge aller Knoten mit aktuellem Eingangsgrad 0. Hiermit ergibt sich folgender verfeinerter Lösungsansatz, der in Algorithmus 36 noch konkreter dargestellt ist.

1. Speichere für jeden Knoten $v \in V$ die Anzahl der noch nicht besuchten Vorgänger in $inDegree[v]$
2. Speichere alle Knoten v, für die $inDegree[v] = 0$ gilt, in der Menge Q (z.B. implementiert als eine Queue)
3. Q enthält demnach alle Knoten, die als nächstes besucht werden dürfen.

4. Nimm einen Knoten u aus Q, entferne ihn aus Q
5. Speichere u in der topologischen Sortierung als nächstes Element
6. Reduziere $inDegree[v]$, für alle $(u, v) \in E$, um 1 und aktualisiere Q
7. Solange Q nicht leer ist gehe zu (4)

Algorithmus 36 Top-Sort(G, **var** ord)

```
1: ∀v ∈ V : inDegree[v] = 0;
2: für alle (u, v) ∈ E {
3:    inDegree[v] = inDegree[v] + 1;
4: }
5: Q = {v ∈ V | inDegree[v] = 0};
6: k = 1;
7: solange Q ist nicht leer {
8:    nimm ersten Knoten u aus Q; Q = Q \ {u};
9:    ord(u) = k;
10:   k = k + 1;
11:   für alle Knoten v ∈ N+(u) {
12:      inDegree[v] = inDegree[v] − 1;
13:      falls inDegree[v] == 0 dann {
14:         Q = Q ∪ {v};
15:      }
16:   }
17: }
18: falls k ≠ n + 1 dann {
19:    G ist nicht azyklisch und es existiert keine topologische Sortierung;
20: }
```

Dieser skizzierte Algorithmus zur topologischen Sortierung ist eine modifizierte *Breitensuche*. Bei der Breitensuche werden – im Gegensatz zur Tiefensuche – alle Nachfolger eines Knotens immer abgearbeitet, *bevor* deren weitere Nachfolger verfolgt werden.

Laufzeit

Sei $n = |V|$ die Anzahl der Elemente und $m = |E|$ die Anzahl der Beziehungen. Die Initialisierung von $inDegree$ und Q benötigt Laufzeit $\Theta(n+m)$. Die zentrale Schleife hat (im erfolgreichen Fall) $\Theta(n)$ Wiederholungen. Insgesamt wird darin jede Kante genau einmal inspiziert, das entsprechende Aktualisieren von $inDegree$ und Q benötigt nur eine konstante zusätzliche Zeit pro Kante. Somit ist auch die Gesamtlaufzeit des topologischen Sortierens $\Theta(n + m)$.