Satz: Ein gerichteter Graph besitzt genau dann eine topologische Sortierung, wenn er azyklisch ist.

"$\Rightarrow$": Ein Graph möge eine topologische Sortierung ord: $V \rightarrow \mathbb{N}$ mit $\forall\, x,y \in V$ mit $x \neq y$ gilt: $(x,y) \in E^* \Rightarrow ord(x) < ord(y)$ besitzen. Betrachte dann einen geschlossenen Weg in G: $(x_0, x_1, ..., x_{k-1}, x_0)$ mit $k \geq 0$. Wegen $(x_0,x_0) \in E^*$ muss dann $ord(x_0) < ord(x_0)$ gelten, Widerspruch.

"$\Leftarrow$": Setze i = 1. Ein gerichteter azyklischer Graph besitzt mindestens einen Knoten x mit dem Eingangsgrad 0. Setze ord(x) = i, erhöhe i um 1, entferne den Knoten x und die zu ihm inzidenten Kanten (es entsteht ein gerichteter azyklischer Graph G') und fahre rekursiv mit G' fort. So erhält jeder Knoten u aus G eine Nummer ord(u). Wenn nun $(x,y) \in E^*$ in G gilt, so kann der Eingangsgrad von y bei dieser Konstruktion erst dann 0 werden, wenn der Knoten x bereits entfernt ist. Dies heißt aber: ord(x) < ord(y).

Der Beweis liefert zugleich einen Algorithmus zur Berechnung einer topologischen Sortierung:

```
i := 0;
for j in 1..n loop
      wähle einen Knoten u mit Eingangsgrad 0;
      i:=i+1;  setze ord(u) = i;
      entferne u aus dem Graphen;
end loop;
```

Man beachte, dass sich der Eingangsgrad der Knoten bei jedem Entfernen eines Knoten verändern kann.

Da es mehrere Knoten mit dem Eingangsgrad 0 geben kann und da jede Auswahl eines solchen Knotens zu einer topologischen Sortierung führt, sind topologische Sortierungen in der Regel nicht eindeutig.

Probleme bei diesem Algorithmus:
- Wie findet man am Anfang rasch einen Knoten mit dem Eingangsgrad 0?
- Wie aktualisiert man die Eingangsgrade beim Entfernen eine Knotens?

```
k := Anfang;              -- Die Eingangsgrade werden in zahl1 gespeichert
while k /= null loop  k.zahl1 := 0;  k:=k.NKn;  end loop;
k := Anfang;                        -- Gehe alle Kanten durch und erhöhe den
while k /= null loop  edge := k.EIK;   -- Eingangsgrad des Zielknotens
    while  edge /= null  loop
        edge.EKn.zahl1 := edge.EKn.zahl1 + 1;
        edge := edge.NKa;
    end loop;
    k := k.NKn;
end loop;
```

Dieses Programmstück ermittelt alle Eingangsgrade in O(n+m) Schritten.

Um schnell einen Knoten mit Eingangsgrad 0 finden zu können, speichern wir alle diese Knoten in einer Liste "GradNull":

```
k := Anfang; "Setze GradNull auf die leere Liste";
while k /= null loop
      if k.zahl1 = 0 then "füge k an GradNull an" end if;
      k:=k.NKn;  end loop;
```

Nun müssen wir noch die Eingangsgrade aktualisieren, sobald der Knoten u aus dem Graphen entfernt wird.

Hierzu erniedrigen wir die Eingangsgrade aller Knoten, die über die Kantenliste von u erreichbar sind. Falls hierbei ein Eingangsgrad auf 0 absinkt, wird der entsprechende Knoten in GradNull aufgenommen. (u sei der Verweis auf den als nächstes zu entfernenden Knoten mit Eingangsgrad 0.) Dies liefert:

```
edge := u.EIK;
while  edge /= null  loop
    edge.EKn.zahl1 := edge.EKn.zahl1 - 1;
    if edge.EKn.zahl1 = 0
        then "füge edge.EKn an GradNull an" end if;
    edge := edge.NKa;
end loop;
"entferne nun den Knoten u aus dem Graphen G"
```

Die restlichen Details überlassen wir den Leser(inne)n.

Beachten Sie: Entweder arbeitet man auf einer Kopie von G und trägt zusätzlich zahl1 im Original ein oder man verwendet einen Booleschen Wert, um zu simulieren, dass man den jeweiligen Knoten gelöscht hat.

Zeitkomplexität des topologischen Sortierens: O(n+m).
Platzkomplexität O(n) (bei Verwendung eines Booleschen Wertes).

Begründung zur Zeitkomplexität:

Jeder Knoten erhält seinen Eingangsgrad: O(n+m).

Aufbau der Liste GradNull: O(n).

n mal: Einen Knoten mit Eingangsgrad 0 finden (=nimm stets den ersten Knoten in der Liste GradNull) und später entfernen, insgesamt: O(n).

m mal insgesamt: Eingangsgrade erniedrigen und Knoten mit Eingangsgrad 0 an GradNull anhängen: O(n+m).

Alle übrigen Operationen (i:=i+1; setze ord(u)=i; usw.) erfordern insgesamt höchstens O(n) Schritte.

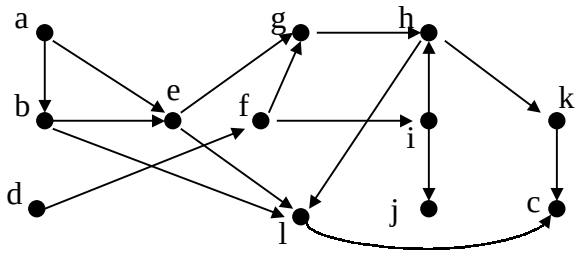
Wir betrachten folgenden Algorithmus TopSort, der in O(n+m) Schritten arbeitet und im Wesentlichen gleich der Tiefensuche ist (n=Zahl der Knoten, m = Zahl der Kanten):

```
procedure TS (u: NextKnoten) is
begin  u.besucht := true;
       for alle Nachfolgerknoten v von u loop
             if not v.besucht then TS (v); end if; end loop;
       u.zahl2 := nummer;  nummer := nummer - 1;
end;

for alle Knoten k loop k.besucht := false; end loop;
nummer := n;
for alle Knoten k loop
          if not k.besucht then TS (k); end if; end loop;
```

Hinweis: Diese zahl2-Werte heißen in der Literatur auch "finish"-Werte.

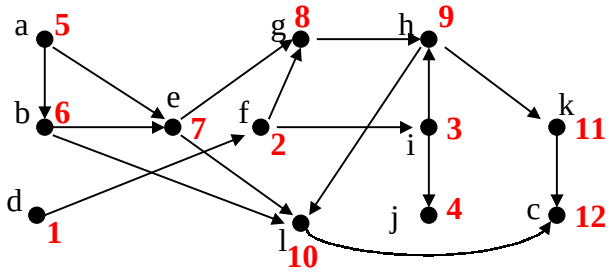
Beispiel: Azyklischer Graph mit n=12 Knoten und m=16 Kanten



Die Reihenfolge der Knoten in der Adjazenzliste sei k, e, h, a, f, j, i, d, c, b, g, l.

Dann liefert der obige Algorithmus TopSort folgende Zahlen zahl2 für die Knoten, sofern man beim Nachfolger zuerst der Kante folgt, die bzgl. der Uhr die kleinste Zahl hat.

Beispiel: Azyklischer Graph mit n=12 Knoten und m=16 Kanten



Dies ist eine topologische Sortierung!

Ist das Zufall? [Nein.]
Versuchen Sie zu beweisen, dass der Algorithmus TopSort für azyklische Graphen stets eine topologische Sortierung liefert (vgl. Buch Algorithmik von Schöning, S. 208). Was liefert er für beliebige gerichtete Graphen?

7.3 Kürzeste Wege

Gegeben sei ein gerichteter Graph G = (V, E, δ) mit $\delta : E \to \mathbb{R}^+$ (= Menge der nichtnegativen reellen Zahlen) und ein Knoten $u \in V$. Gesucht werden alle kürzesten Abstände δ(u,v) für alle Knoten $v \in V$.

Hierfür verwendet man in der Regel den Dijkstra-Algorithmus, der sich wie eine Breitensuche (gewichtet bzgl. der Entfernungsfunktion δ) über den Graph vorarbeitet. Zeitkomplexität: $O((n+m) \cdot \log(n))$.

Sucht man die kürzesten Abstände zwischen allen Knoten, so kann man den Dijkstra-Algorithmus für jeden Knoten einmal durchführen oder den Floyd-Algorithmus mit Zeitkomplexität $\Theta(n^3)$ verwenden.

Diese Algorithmen sind sehr gut in Lehrbüchern beschrieben. Wir erläutern sie an der Tafel.

(Edgar W. Dijkstra = holländischer Wissenschaftler; einer der Pioniere der Informatik. Der Algorithmus wurde 1959 veröffentlicht.)

7.4 Minimale Spannbäume

Gegeben sei ein ungerichteter Graph G = (V, E, δ) mit δ : E→ IR (= Menge der reellen Zahlen).
Gesucht wird ein minimaler Spannbaum, also ein Baum B=(V,E_B,δ) mit gleicher Knotenmenge, der Teilgraph von G ist und dessen Gewicht δ(B) minimal ist bzgl. aller Spannbäume von G.

Hierfür verwendet man meist folgende zwei Algorithmen: Der Prim-Algorithmus arbeitet wie der Dijkstra-Algorithmus, wählt aber in jedem Schritt die kleinste am Rande der bisher besuchten Knoten liegende Kante aus, die keinen Zyklus bildet. Zeitkomplexität O((n+m)·log(n)).
Der Kruskal-Algorithmus sortiert die Kanten bzgl. δ, beginnt dann mit der kleinsten Kante und nimmt jeweils die nächste Kante hinzu, sofern diese Kante keinen Zyklus mit den bereits ausgewählten Kanten bildet. Zeitkomplexität: O(m·log(m)) bei geeigneter Implementierung.

Auch diese Algorithmen sind sehr gut in Lehrbüchern beschrieben.

J.B.Kruskal und R.C.Prim veröffentlichten ihre Algorithmen 1956 bzw. 1957.

E N D E

der Vorlesung 2002

Beachten Sie bitte auch die vierseitige Zusammenfassung über die Vorlesungsinhalte, die Prüfungen und die Statistiken und empfehlen Sie uns Ihren Verwandten, Freunden und sonstigen Interessenten für die Informatik, Softwaretechnik, Wirtschaftsinformatik, Informationstechnik, Computerlinguistik, Automatisierungstechnik, Mathematik, technischen Kybernetik, Betriebswirtschaftslehre, Physik usw. weiter. Viel Erfolg bei den Prüfungen!