Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger
Tomáš Gavenčiak

8. Dezember 2016

Algorithmen & Datenstrukturen Blatt P12 HS 16

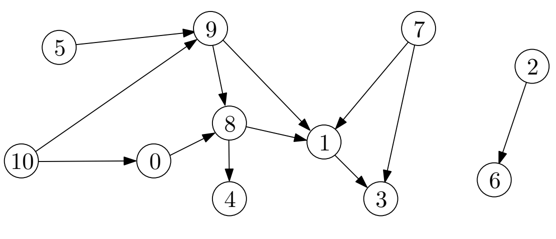
Abgabe: Bis zum 15. Dezember 2016 um 10 Uhr auf dem Judge (ausschliesslich Quellcode).

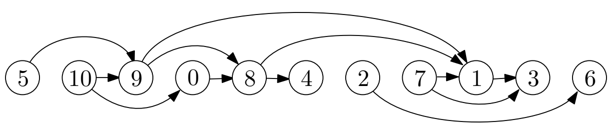
Aufgabe P12.1 *Längster Pfad in einem gerichteten, kreisfreien Graphen.*

Gegeben sei ein gerichteter, kreisfreier Graph (*directed acyclic graph, DAG*). Ihre Aufgabe ist es, einen längsten, gerichteten Pfad in diesem Graph zu berechnen. Die n Knoten des Graphen sind mit $0, 1, \ldots, n-1$ nummeriert und der Graph hat m gerichtete Kanten (*arcs*), jeweils von einem Knoten s_i zu einem Knoten t_i für $i = 1, \ldots, m$. Ein *gerichteter Pfad* der Länge l ist eine Sequenz von l verschiedenen[1] Knoten $p_1, \ldots, p_l$, sodass es eine gerichtete Kante von p_i nach p_{i+1} für jedes $i = 1, \ldots, l-1$ gibt. Ein Graph ist *kreisfrei*, wenn er keinen gerichteten Kreis enthält, oder – formal – wenn es keinen gerichteten Pfad von p_1 zu p_l gibt, sodass es auch eine Kante von p_l nach p_1 gibt (der den gerichteten Kreis schliesst).

Es sind keine effizienten Algorithmen bekannt, die einen längsten Pfad in einem *beliebigen* Graphen berechnen können[2]. Für gerichtete, kreisfreie Graphen gibt es jedoch eine einfache und effiziente Lösung. Wir schlagen folgende Vorgehensweise vor:

Finden Sie zuerst eine *topologische Sortierung* der Knoten (also eine Sortierung $v_1, \ldots, v_n$ der Knoten, sodass Kanten, die von v_i ausgehen, nur in Knoten v_j mit $j > i$ enden) wie in der Vorlesung besprochen. Benutzen Sie dann das Prinzip der Dynamischen Programmierung oder ähnliche Ansätze mit dieser Sortierung um einen längsten, gerichteten Pfad zu finden. Die detaillierte Ausarbeitung der Vorgehensweise ist Teil Ihrer Aufgabe.





[1] In einem gerichteten, kreisfreien Graphen ist jeder gerichtete Weg ein Pfad, da das wiederholtes Besuchen eines Knoten die Existenz eines gerichteten Kreises impliziert.

[2] Für weitere Informationen, suchen Sie nach “Hamiltonian path problem” und “NP-completeness” auf Wikipedia.

Beispiel Wir betrachten einen Graph mit $n = 11$, $m = 12$ und den folgenden Kanten (Startknoten→Zielknoten): 7→3, 1→3, 7→1, 2→6, 5→9, 10→9, 10→0, 0→8, 9→8, 9→1, 8→4, 8→1. Der Graph und eine seiner topologischen Sortierungen sind in der Abbildung oben dargestellt. Die Länge eines längsten, gerichteten Pfades ist 5 und es gibt drei längste Pfade 5-9-8-1-3, 10-9-8-1-3 und 10-0-8-1-3 (es gibt auch noch weitere, aber uns interessiert ausschliesslich die Länge).

Eingabe Die Eingabe besteht aus mehreren Tests. Die erste Zeile enthält die Anzahl der Tests, die folgen.

Jeder Test besteht aus zwei Zeilen: Die erste Zeile enthält die Ganzzahlen $1 \leq n \leq 10\,000$ und $0 \leq m \leq 10\,000$, durch Leerzeichen getrennt. Die zweite Zeile enthält $2m$ Ganzzahlen $s_1, t_1, s_2, t_2, \ldots, s_m, t_m$, die Start- und Zielknoten der gerichteten Kanten (mit $s_i, t_i \in \{1, \ldots, m\}$ für alle Zahlen), durch Leerzeichen getrennt..

Der Graph enthält keine Schleifen (Kanten von einem Knoten v zum selben Knoten v) oder mehrere, parallele Kanten von u nach v. Falls es eine Kante von u nach v gibt, gibt es keine Kante von v nach u geben, da das die Kreisfreiheit verletzen würde. Die gegebene Graph kann, aber muss nicht zusammenhängend sein (siehe Beispiel unten). Beachten Sie auch, dass es möglicherweise *ungerichtete* Kreise (d.h. Kreise, wenn wir die Richtung der Kanten ignorieren) gibt, wie im Beispiel. Die Kanten sind in keiner bestimmten Reihenfolge gegeben.

Ausgabe Geben Sie für jeden Test die Länge des längsten gerichteten Pfades in einer separaten Zeile aus.

Beispiel

Eingabe (für das obige Beispiel und einen leeren Graph)

```
2
11 12
7 3 1 3 7 1 2 6 5 9 10 9 10 0 0 8 9 8 9 1 8 4 8 1
5 0
```

Ausgabe

```
5
1
```

Bonus Sie erhalten einen Bonuspunkt pro 100 Punkte auf dem Judge (abgerundet). Insgesamt können Sie bis zu 200 Punkte auf dem Judge erhalten. Damit alle Tests auf dem Judge erfolgreich sind, sollte die Laufzeit Ihres Programms in $\mathcal{O}(m + n)$ liegen.

Senden Sie Ihr `Main.java` unter folgendem Link ein: `https://judge.inf.ethz.ch/team/websubmit.php?cid=18985&problem=DA_P12.1`. Das Passwort für die Einschreibung ist "`quicksort`".

Hinweise

Wir stellen für diese Aufgabe eine Programmvorlage als Eclipse Projektarchiv auf der Vorlesungswebseite zur Verfügung. In der Vorlage wird die Eingabe bereits eingelesen. Nach dem Einlesen steht der Graph in drei verschiedenen Repräsentationen zur Verfügung: Eine Liste von Start- und Zielknoten s_i und t_i der m Kanten; für jeden Knoten v eine Liste von Nachbarn, die durch von v ausgehende Kante mit v verbunden sind (Knoten mit einer gerichteten Kante *von*

v); für jeden Knoten v eine Liste der Nachbarn, die durch zu v eingehende Kanten mit v verbunden sind (Knoten mit einer gerichteten Kante *zu* v). Beachten Sie, dass eine Adjazenzmatrix für diese Aufgabe nicht geeignet ist.

Das Archiv enthält weitere Testdaten, damit Sie lokal testen können. Ausserdem stellen wir zusätzlich ein `Judge.java` Programm zur Verfügung, das Ihr Programm `Main.java` mit allen verfügbaren Testdaten testet – öffnen und starten sie dazu einfach `Judge.java` in derselben Weise wie Sie `Main.java` starten würden. Die Art und Weise, wie Ihr Programm `Main.java` arbeitet, wird dadurch nicht beeinflusst. `Judge.java` soll lediglich das lokale Testen erleichtern. Die zur Verfügung gestellten Testdaten sind nicht die Testdaten, welche der Judge verwendet, und im Vergleich nicht so umfangreich.

Zusatzaufgabe

Wenn Sie die oben beschriebene Aufgabe lösen können und die Länge l eines längsten Pfades finden können, wie können Sie alle gerichteten, längsten Pfade in $\mathcal{O}(m+n)$ zählen? (Sie können annehmen, dass die Anzahl der Pfade mit dem Datentyp `int` repräsentiert werden kann.) Für diese Zusatzaufgabe gibt es keine Bonuspunkte.