Algorithmendesign
Lösungen zu Übungsblatt 5
WS 13/14

Gruppe 2
Max Bannach

Markus Richter (614027)

Aufgabe 5.1 Wege in azyklischen Graphen

Teilaufgabe 1)

Um die maximale Anzahl an Wegen, die es zwischen zwei Knoten geben kann, zu berechnen sei jeder Knoten mit jedem nachfolgenden Knoten verbunden. Dann gibt es eine maximale Anzahl von Wegen insgesamt. Für jedes Knotenpaar sei nun n die Anzahl der Knoten, inklusive Start- und Endknoten, die ab dem Startknoten zum Zielknoten führen. Es gibt also zwischen Start- und Endknoten genau $n-2$ Knoten, die durchlaufen werden können. Für jeden Knoten kann es nun nur zwei Möglichkeiten geben:

1. Der Knoten wird genommen
2. Der Knoten wird nicht genommen

Daraus folgt, dass es maximal 2^{n-2} Wege zwischen zwei Knoten geben kann.

Teilaufgabe 2)

Für den Algorithmus benutze ich die dynamische Programmierung. Dazu verwende ich eine Memoizationsmatrix $M(i,j)$. Der Graph sei aufsteigend sortiert, sodass für den Index der sortierten Liste gilt: Wenn $i < j$, dann folgt Knoten v_j auf v_i. Weiterhin ist die Adjazenzmatrix $A(i,j)$ im Vorfeld zu berechnen. Nun erfolgt das initiale Befüllen der Matrix M wie folgt:

- Die Hauptdiagonale wird mit 0ern befüllt.
- Für Index j alle i, für die gilt $i < j$ mit 0 füllen, da es keinen Weg zurück geben kann.

Die darauf folgenden Iterationen beruhen auf der Distanz d. Diese ist zu Beginn $d = 1$, d. h. es werden alle aufeinander folgenden Knoten betrachtet, deren Indexdifferenz 1 beträgt. Dazu wird wie folgt vorgegangen: Alle Knoten mit Index i,j mit einem Abstand von $d = 1$ mit Hilfe der Adjazenzmatrix berechnen: Gibt es einen Eintrag $A(i,j)$, dann ist $M(i,j) = 1$, andernfalls 0.
Nun erhöht man die Distanz d um 1 und vergleicht somit alle Knoten v_i mit v_{i+d}, dazu betrachtet man den Wert $M(i, i+(d-1))$ und multipliziert mit $A(i+(d-1), i+d)$. Gibt es nämlich keine Verbindung zwischen den letzten beiden Knoten, macht der Weg bis zum vorletzten Knoten auch keinen Sinn. Zusätzlich kommt jeweils eine Addition für jedes $A(i,i+1)$ bis $A(i,i+d)$ hinzu, also die direkten Verbindungen. Das wird solange fortgesetzt bis alle Zellen ermittelt wurden. In jeder Zelle (i,j) ist nun die Anzahl an Wegen zwischen diesem Knotenpaar v_i und v_j ersichtlich.

Korrektheit: Der Algorithmus sorgt dafür, dass die Memoizationsmatrix initial – also noch vor der ersten Iteration – an den entsprechenden Stellen (siehe oben) mit 0ern gefüllt wird. Diese Stellen können von den darauf folgenden Iterationen nicht erreicht werden. Die erste Iteration beruht auf der Adjazenzmatrix und ergibt damit auf jeden Fall entweder eine 1 oder 0. Die darauf folgende Iteration nutzt diese Werte und zusätzlich die direkten Verbindungen, somit ergeben sich für diese Iteration Werte größer gleich der Werte aus der

vorherigen Iteration. Für jede Iteration und damit für jede Zelle ergeben sich also definitiv Werte. Diese Werte sind korrekt, da sie für jeden Iterationsschritt auf dem vorherigen Iterationsschritt zuzüglich direkter Verbindungen beruhen usw. sodass alle Werte auf direkten Verbindungen und indirekt auf den ersten Iterationsschritt beruhen, der anhand der direkten Verbindung (Adjazenzmatrix) berechnet wurde und damit korrekt ist. Damit sind alle Werte korrekt.

Laufzeit: Das anfängliche Sortieren des Graphen benötigt $\mathcal{O}(n^2)$. Das Berechnen der Adjazenzmatrix benötigt $\mathcal{O}(|V|^2)$ Das initiale Befüllen mit 0ern ist zu vernachlässigen. Anschließend werden die Zellen berechnet. Dabei wird der Abstand jedes mal um 1 erhöht und verglichen, was im Grunde dem kleinen Gauß entspricht, also $\mathcal{O}(n^2)$. Zusätzlich müssen jedes mal die direkten Verbindungen überprüft werden, was im schlimmsten Fall, wenn alle Knoten mit dem letzten Knoten verbunden sind, nahezu n sein können. Es ergibt sich also $\mathcal{O}(nn^2) = \mathcal{O}(n^3)$.

Aufgabe 5.2 MAXFLOW/MINCUT

Teilaufgabe 1)

Siehe Anhang.

Teilaufgabe 2)

Für den minimalen Schnitt wird der Graph in zwei disjunkte Mengen Q und S aufgeteilt, wobei gilt: $Q = \{q, b\}, S = \{a, c, d, s\}$. Der $MAXFLOW(G) = 21$, daher muss nach dem Maxflow-Mincut-Theorem auch gelten, dass die Schnittkapazität $\gamma(Q, S) = 21$. Die Berechnung erfolgt folgendermaßen:

$\gamma(Q, S) = \sum_{(v,u)\in E, v\in Q, u\in S} \gamma(v, u) = \gamma(q, a) + \gamma(q, c) + \gamma(b, d) + \gamma(b, s) = 8 + 5 + 3 + 5 = 21$
Dieser Schnitt entspricht somit dem minimalen Schnitt.

Teilaufgabe 3)

Man erweitere den GENERAL-MAXFLOW-Algorithmus vor dem letzten end-Marker wie folgt:

```
1  Algorithmus Erweiterung GENERAL-MAXFLOW:
2  Eingabe: Der Restgraph R = (G', V')
3  Ausgabe: Mengen Q und S. Entsprechen dem Schnitt.
4  Mengen Q := {q}, S := V' \ {q}
5
6  FOR alle e' ∈ E' DO
7    IF e', so dass e'(q,v) THEN
8        Q := Q ∪ {v};
9        S := S \ {v};
10   ENDIF
11 OD.
12
13 RETURN Q,S;
```

Korrektheit: Der Algorithmus rechnet mit dem Restgraphen R, der aus dem bereits berechneten MAXFLOW resultiert. Dadurch, dass nur Knoten in die Menge Q aufgenommen werden, die im Restgraphen mit q verbunden sind, wird garantiert, dass sämtliche Ausflüsse aus der Menge Q die volle Kapazität nutzen. Es gibt darüber hinaus immer eine

Unterteilung in Q und S, selbst wenn es keine Verbindung von q zu einem Nachfolgeknoten v_i im Restgraphen gibt. Dann besteht die Menge Q aus der Quelle q selbst, während S aus $V' \setminus \{q\}$ besteht. Der Algorithmus ist also nicht nur korrekt, sondern auch optimal.

Laufzeit: Man betrachtet sämtliche $e' \in E'$, um nach einer Verbindung zu q zu prüfen. Daraus resultiert $\mathcal{O}(|E|)$. GENERAL-MAXFLOW dominiert jedoch mit einer Laufzeit von $\mathcal{O}(|E||V|^2)$, somit ist die Laufzeit insgesamt die selbige.

Aufgabe 5.3 Wege in azyklischen Graphen

Teilaufgabe 1)

Nein, eine maximale monoton steigende Teilfolge ist nicht immer eindeutig. Gegenbeispiel: Sei die Folge $X = 3, 2, 1$. Nun lassen sich drei Teilfolgen Y_1, Y_2, Y_3 finden, sodass gilt: $Y_1 = 3, Y_2 = 2, Y_3 = 1$. Jede dieser Teilmengen ist monoton und maximal, denn alle anderen Kombinationen wären nicht mehr monoton steigend.

Teilaufgabe 2)

Für den Algorithmus nutze ich wieder die dynamische Programmierung. Die Idee ist nun folgende: Die Folge $X = x_1, ..., x_n$ wird von hinten nach vorne durchlaufen. Angefangen bei $i = n$ wird ein Wert $l(i)$ – die Länge der monotonen Zahlenfolge ab einschließlich dieser Stelle – gespeichert. Bei der Berechnung werden aber nur $j > i$ betrachtet, wenn gilt $x_j \geq x_i$, dann entspricht $l(i)$=$l(i) + 1$, da die Bedingung weiterhin gilt und die Länge damit für die monotone Folge mit i um 1 zunimmt. Andernfalls ist der Wert an der Stelle i gleich 0. Die Lösung ist nun das i für das $l(i)$ maximal wird.

Korrektheit: Der Algorithmus liefert für jedes i definitiv einen Wert. Durch Iterieren über alle i ergeben sich i Werte, wobei einer der maximale ist, der immer vom Algorithmus gefunden wird. Der Algorithmus ist also korrekt.

Laufzeit: Für jedes i sind bis zu n Vergleiche notwendig, daher hierfür eine Laufzeit von $\mathcal{O}$(n). Dies muss nun n mal ausgeführt werden, weswegen sich insgesamt eine Laufzeit von $\mathcal{O}(n^2)$ ergibt.