Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dr. Hanjo Täubig
Tobias Lieber

Sommersemester 2011
Übungsblatt 10
26. Juli 2011

Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: In der jeweiligen Tutorübung

Aufgabe 1

Bestimmen Sie die Laufzeit des BFS-Algorithmus. Betrachten Sie zwei Implementierungen: Zum einen basierend auf einer Adjazenzmatrix und zum anderen basierend auf einer Adjazenzliste.

Lösungsvorschlag

- Adjazenzmatrix:

  Um alle Nachbarn eines Knoten v in die Queue einzufügen müssen alle möglchen Nachbarn $(n-1)$ überprüft werden. Da dies n mal geschieht erhalten wir also eine Laufzeit von $\mathcal{O}(n^2)$.

- Adjazenzliste:

  Hier haben wir direkten Zugriff auf die Nachbarn. Jeder der Nachbarn muss überprüft werden. Damit fallen pro Knoten $\mathcal{O}(\deg(v))$ Rechenoperationen an.

  Da $\sum_{v \in V} \deg(v) = |E|$ gilt, wird also jede Kante nur konstant oft angeschaut und wir erhalten eine Laufzeit von $\mathcal{O}(m+n)$

Aufgabe 2

Gegeben Sei die Adjazenzmatrix $A = (a_{ij})_{1 \leq i,j \leq n}$ eines gerichteten, einfachen Graphen G mit n Knoten. Geben Sie einen Algorithmus an, der eine Senke in $\mathcal{O}(n)$ Schritten findet. Eine Senke sei (abweichend von der üblichen Definition) ein Knoten v, der Eingrad $n-1$ und keine ausgehenen Kanten hat.

Lösungsvorschlag

Da wir hier mit einfachen Graphen arbeiten sei im Folgenden immer $i \neq j$. Wir betrachten die Bedeutung die wir aus einer Abfrage an die Adjazenzmatrix folgern können. Ist $a_{ij} = 1$ so geht eine Kante von Knoten i zu Knoten j. Daraus folgt, dass Knoten i keine Senke sein kann (da er Ausgrad größer 0 hat). Knoten j kann weiterhin eine Senke sein, sein Eingrad ist mindestens 1.
Auf der anderen Seite gilt: Ist $a_{ij} = 0$ so kann Knoten j keine Senke sein, da der Eingrad nicht mehr $n-1$ sein kann, da wir einfache Graphen betrachten. Knoten i kann eine Senke sein, da sein Ausgrad zumindestens nicht von der möglichen Kante von i nach j abhängt.

Betrachten wir nun folgende Vorgehensweise:

- Wir stellen $\lfloor \frac{n}{2} \rfloor$ Anfragen für die Kanten $(2i+1, 2i+2)$ mit $i \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 1\}$ an A. Daraus merken wir uns jeweils die noch möglichen Knoten B für eine Senke.

  Für B und einem eventuell noch nicht getesteten Knoten wiederholen wir das Verfahren bis schlussendlich nur noch ein Knoten x übrig ist.

  Wir können nun anhand der Adjazenzmatrix überprüfen ob der Knoten x tatsächlich eine Senke ist ($n-1$ weitere Tests).

- Alternatives Verfahren: Wir beginnen mit der Knotenmenge V, der Kandidaten für die Senke. Wir vergleichen zwei beliebige Knoten aus der Menge V und entfernen den Knoten, der aus den Folgerungen oben definitiv keine Senke sein kann.

  Das ganze wiederholen wir, bis nur noch ein Kandidat für die Senke übrig ist, welchen wir dann Explizit überprüfen. (Wie in der ersten Möglichkeit am Ende). Da hier $n-1$ Vergleiche bis zur einelementigen Kandidatenmenge ausgeführt werden und nocheinmal $n-1$ Vergleiche für die Bestätigung/Ablehnung der Senke, benötigt der Algoritmus also $\mathcal{O}(n)$ Zeit.

Aufgabe 3

In dieser Aufgabe wollen wir ein alternatives Verfahren zur topologischen Sortierung auf einem DAG G betrachten. Dazu nehmen wir an, dass uns weder die üblichen Graphoperationen zum Traversieren einer Kante oder Abfragen der Nachbarschaft eines Knotens, noch irgendwelche „höheren Datenstrukuren“ wie z.B. Queues, Listen, PriorityQueues oder Stacks zur Verfügung stehen.
Die einzige Operation, die wir benutzen können ist eine Tiefensuche (DFS), die Arrays `dfs_num` und `finish_num` zurückgibt, die zu den jeweiligen Knoten die entsprechenden Werte beeinhalten. Außerdem stehen uns `for`-Schleifen und die übliche Addition bzw. Subtraktion zur Verfügung.

a) Geben Sie einen Algorithmus (Pseudocode reicht) an, der dieselbe asymptotische Laufzeit erreicht, wie der in der Vorlesung vorgestellte Algorithmus.

b) Begründen Sie die Laufzeit Ihres Algorithmus kurz.

c) Beweisen Sie die Korrektheit Ihres Algorithmus.

Lösungsvorschlag

1. Wir führen eine DFS auf dem Graphen G aus. Daraus erhalten wir ein Array mit der Finish-Reihenfolge. In einer For-Schleife setzen wir für alle Knoten i die Ausgabe als $n -$ `finish_num[i]`.

2. Laut Vorlesung benötigt die DFS gerade $\mathcal{O}(n+m)$ Zeit. Dann wird für jeden Knoten noch eine Rechenoperation ausgeführt ($\mathcal{O}(n)$). Also erhalten wir als Laufzeit insgesamt $\mathcal{O}(n+m)$.

3. Da die Eingabe ein DAG ist (Voraussetzung), wissen wir aus der Vorlesung: Wenn $(v, w) \in E$: dann gilt: `finish_num[v]` $>$ `finish_num[w]`. Also gilt: $(v, w) \in E \Rightarrow n -$ `finish_num[v]` $< n -$ `finish_num[w]` wobei $n -$ `finish_num[x]` gerade die oben definierte Nummer in der topologischen Sortierung ist. Damit gilt `topo[v]` $<$ `topo[w]` wenn $(v, w) \in E$ ist. Dies ist aber gerade die Bedingung an eine topologische Sortierung und unser Algorithmus ist damit korrekt.