Goethe-Universität Frankfurt am Main
Institut für Informatik
Theorie Komplexer Systeme
Dr. Mariano Zelke

29. Mai 2012

Datenstrukturen

Sommersemester 2012

Übungsblatt 4

Abgabe: *bis 12. Juni 2012, 8.^{15} Uhr (vor der Vorlesung oder in Raum RM 11-15/113)*

Bitte achten Sie darauf, dass Sie auf der Abgabe Ihrer Lösung Ihren ***Namen****, Ihre* ***Matrikelnummer*** *und Ihre* ***Übungsgruppe*** *angeben. Fehlt eine dieser Angaben, müssen Sie mit* ***Punktabzug*** *rechnen. Mehrseitige Abgaben müssen zusammengeheftet werden.*

Eine Teilaufgabe gilt nur dann als bearbeitet, wenn neben der Lösung auch die notwendigen Begründungen angegeben sind – es sei denn, die Teilaufgabe ist mit einem $*$ markiert.

Aufgabe 1: (10 Punkte)

Es wird langsam unübersichtlich im Universum der Marvel Comicverfilmungen. Seit Kurzem umfasst es sechs Filme: *Iron Man* (erschienen 2008), *Der unglaubliche Hulk* (2008), *Iron Man 2* (2010), *Thor* (2011), *Captain America* (2011) und *The Avengers* (2012). Diese Reihe wird mit den geplanten Filmen weiter anwachsen: *Iron Man 3* (geplant 2013), *Thor 2* (2013), *Captain America 2* (2014) sowie *The Avengers 2* (2015).

Es wird dann bei diesen zehn Filmen selbst für Profis nicht leicht zu entscheiden sein, in welcher Reihenfolge sie angeschaut werden sollten, um den meisten Sinn zu ergeben. Ein Ziel dieser Aufgabe ist es, diese Reihenfolge zu bestimmen. Dazu benutzen wir die nebenstehende Tabelle, die uns aus gut informierten Kreisen zugespielt wurde und die für jeden Film angibt, welche Filme nützlicherweise davor angeschaut werden sollten.

Film	Davor anzuschauen sind
Iron Man	*Captain America*
Der unglaubliche Hulk	*Thor*, *Captain America*
Iron Man 2	*Iron Man*
Thor	*Iron Man*, *Iron Man 2*
Captain America	
The Avengers	*Der unglaubliche Hulk*
Iron Man 3	*Iron Man*, *Iron Man 2*, *Thor 2*
Thor 2	*Thor*, *Captain America 2*
Captain America 2	*Captain America*, *The Avengers 2*
The Avengers 2	*The Avengers*, *Thor*

(a*) Modellieren Sie die Abhängigkeiten der Filme als gerichteten Graphen $G = (V, E)$. Dabei soll jeder Film einem Knoten in V entsprechen und für $a, b \in V$ soll eine gerichtete Kante (a, b) genau dann in E sein, wenn der durch a repräsentierte Film vor dem durch b repräsentierten Film anzuschauen ist. Geben Sie G in graphischer Darstellung an.

Für einen gerichteten Graphen $G = (V, E)$ mit n Knoten ist eine *topologische Sortierung* eine Reihenfolge $v_1, v_2, \ldots, v_n$ der Knoten aus V, so dass es keine Kante (v_i, v_j) in E gibt mit $j < i$.

(b*) Geben Sie eine topologische Sortierung Ihres Graphen aus Teilaufgabe **(a)** an.

(c) Zeigen Sie, dass jeder gerichtete azyklische Graph eine topologische Sortierung besitzt. Dabei dürfen Sie den Fakt benutzen, dass es in jedem gerichteten azyklischen Graphen mindestens eine Quelle gibt.

(d*) Geben Sie einen Algorithmus in C++ oder Pseudocode an, der für einen gerichteten Graphen in Adjazenzlisten-Darstellung alle Quellen bestimmt.

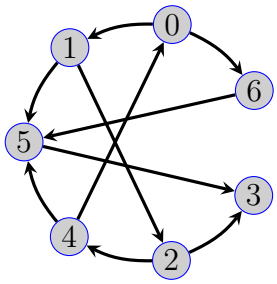
Aufgabe 2: (10 Punkte)

Es sei der nebenstehende gerichtete Graph G in graphischer Darstellung gegeben.

(a*) Geben Sie die Adjazenzmatrix von G an.

(b*) Geben Sie G in Adjazenzlisten-Darstellung an. Sortieren Sie dabei die Knoten in jeder Adjazenzliste aufsteigend.

(c*) Geben Sie für G den Wald der Tiefensuche an, der entsteht, wenn Tiefensuche in Knoten 0 startet und die unbesuchten Nachbarn jedes Knotens in aufsteigender Reihenfolge besucht werden. Klassifizieren Sie jede Kante von G als Baum-, Rückwärts-, Vorwärts-, oder Querkante.



G

(d*) Geben Sie für G den Wald der Breitensuche an, der entsteht, wenn Breitensuche in Knoten 0 startet und die unbesuchten Nachbarn jedes Knotens in aufsteigender Reihenfolge in die Queue eingefügt werden. Auf Grundlage dieses Waldes lassen sich wie bei der Tiefensuche alle Kanten in Baum-, Rückwärts-, Vorwärts-, oder Querkanten einteilen. Klassifizieren Sie alle Kanten von G entsprechend.

Aufgabe 3: (14 Punkte)

(a) Es sei ein ungerichteter Graph $G = (V, E)$ durch seine Adjazenzliste gegeben. Geben Sie einen *nicht rekursiven* Algorithmus in C++ oder Pseudocode an, der Tiefensuche auf G ausführt. Dabei sollen die Knoten von G in genau der gleichen Reihenfolge als besucht markiert werden, wie in der rekursiven Variante aus der Vorlesung und es soll eine Laufzeit von $\mathcal{O}(|V| + |E|)$ erreicht werden.

(b) Der Algorithmus Breitensuche aus der Vorlesung benutzt eine Queue `q` zur Verwaltung der Knoten. Für einen Graphen G definieren wir $\texttt{q}_{max}(G)$ als die größte Anzahl von Knoten, die sich während der Breitensuche auf G *gleichzeitig* in `q` befinden.

Geben Sie zwei Graphen G_1 und G_2 mit jeweils acht Knoten an, so dass $\texttt{q}_{max}(G_1)$ so groß wie möglich und $\texttt{q}_{max}(G_2)$ so klein wie möglich ist.

Aufgabe 4: (8 Punkte)

(a*) Geben Sie einen binären Baum T_1 mit zehn Knoten und einen binären Baum T_2 mit 15 Knoten an, die jeweils Heap-Struktur besitzen.

(b) Gegeben sei das folgende Array `H[21]`:

0	38	35	28	19	29	23	25	11	17	27	14	20	2	23	1	9	3	18	15	24

Dabei enthält die nullte Zelle `H[0]` des Arrays den Dummy-Wert 0, der ohne Bedeutung ist. `H` definiert einen binären Baum mit Heap-Struktur, der 20 Knoten hat. Besitzt dieser Baum auch Heap-Ordnung?

(c) Es sei ein Array `A` gegeben, das Heap für einen Baum T mit elf Knoten ist, die paarweise verschiedene Prioritäten haben. An welchen Stellen im Array `A` kann sich die kleinste Priorität befinden?

---

*Für diese Teilaufgabe ist keine Begründung erforderlich.