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

12. Juni 2012

Datenstrukturen

Sommersemester 2012

Übungsblatt 5

Abgabe: *bis 26. 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)

(a*) Das Array H sei als Heap mit den folgenden Einträgen gegeben:

0	29	27	23	25	14	8	17	2	15	4	7	3

(i) Geben Sie den binären Baum T an, der durch H repräsentiert wird.

(ii) Nun wird mittels `insert(26)` eine neue Priorität eingefügt. Stellen Sie die Situation in T sowohl direkt vor dem internen Aufruf von `repair_up()` als auch direkt nach der Beendigung von `repair_up()` dar.

(iii) Anschließend wird `delete_max()` ausgeführt. Stellen Sie die Situation in T für alle wesentlichen Zwischenschritte dar, die während der Ausführung von `delete_max()` und dem anschließenden `repair_down()` auftreten.

(b) Sei H ein beliebiges Heap-Array, dass n Prioritäten enthält. Außerdem sei ein numerischer Wert ℓ gegeben. Geben Sie einen Algorithmus in C++ oder Pseudocode an, der alle Prioritäten von H ausgibt, die mindestens so groß wie ℓ sind. Wenn es k solche Prioritäten gibt, so sollte Ihr Algorithmus eine Laufzeit von $\mathcal{O}(k)$ erreichen.

Aufgabe 2: (10 Punkte)

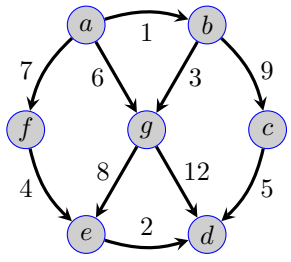
Gegeben sei eine feste Grundmenge $U = \{1, 2, 3, \ldots, u\}$. Das Ziel dieser Aufgabe ist der Entwurf einer Datenstruktur zur Repräsentation einer Teilmenge $M \subseteq U$, die anfangs leer ist. Die Datenstruktur soll die Operationen `add(`i`)` (füge ein Element $i \in U$ zu M hinzu) und `delete(`i`)` (entferne i aus M) unterstützen. Außerdem sollen die Operationen `lookup(`i`)` (ist i Element von M?) und `max()` (gibt das größte Element in M aus, ohne M zu ändern) bereit gestellt werden.

Entwerfen Sie eine Datenstruktur, die `add(`i`)` und `delete(`i`)` in Zeit $\mathcal{O}(\log |M|)$ sowie `lookup(`i`)` und `max()` in Zeit $\mathcal{O}(1)$ unterstützt.

Für diese Aufgabe ist *kein* Code gefordert. Es reicht aus, wenn Sie beschreiben, wie Ihre Datenstruktur aufgebaut ist, wie die vier Operationen darauf arbeiten und wie die jeweils geforderte Laufzeit erreicht wird. Analysieren Sie auch den Speicherbedarf Ihrer Datenstruktur.

Aufgabe 3: (9 Punkte)

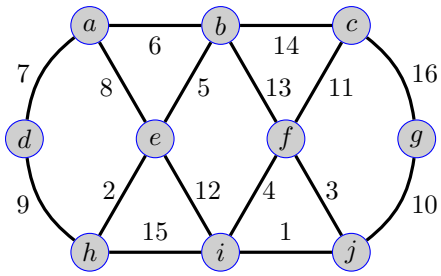
(a*) Es sei der nebenstehende gerichtete Graph $G_1 = (V, E)$ mit Kantengewichten gegeben.



G_1

Beschreiben Sie den Durchlauf des Algorithmus von Dijkstra auf G_1, wenn der Algorithmus mit dem Startknoten a beginnt. Geben Sie dafür zu jedem Knoten $v \in V \setminus \{a\}$ an, wie der Algorithmus zu Beginn in (1) den Wert `distanz[`v`]` setzt. Geben Sie weiterhin für jeden Schritt in (2) an, welcher Knoten in diesem Schritt zu S hinzugefügt wird und welche `distanz`-Werte wie verändert werden.

(b*) Es sei der abgebildete ungerichtete Graph $G_2 = (V, E)$ mit Kantengewichten gegeben.



G_2

(i) Geben Sie die Reihenfolge der Kanten an, in der der Algorithmus von Prim einen minimalen Spannbaum von G_2 konstruiert, wenn als Startknoten der Knoten a gewählt wird.

(ii) Geben Sie die Reihenfolge der Kanten an, in der der Algorithmus von Kruskal einen minimalen Spannbaum von G_2 konstruiert.

(c) Gilt für jeden ungerichteten Graphen $G = (V, E)$ mit Kantengewichten, dass der minimale Spannbaum eindeutig ist?

Aufgabe 4: (13 Punkte)

Wir betrachten in dieser Aufgabe binäre Suchbäume. Eigentlich speichert jeder Knoten in einem solchen Baum einen Schlüssel *und* eine Information. Für die folgenden Aufgaben allerdings können Sie die in jedem Knoten gespeicherte Information vernachlässigen. Es reicht aus, jeden Knoten mit dem Schlüssel gleichzusetzen, den er speichert.

(a*) **(i)** Wir betrachten einen binären Suchbaum B_1, der anfangs leer ist. In B_1 werden nun in dieser Reihenfolge die Schlüssel 12, 20, 8, 25, 5, 18, 6, 16, 30, 2, 22 eingefügt. Stellen Sie B_1 direkt nach allen Einfügungen dar.

(ii) Stellen Sie die Situation in B_1 dar, nachdem anschließend der Schlüssel 8 durch die `remove`-Operation entfernt wurde.

(iii) Stellen Sie die Situation in B_1 dar, nachdem daran anschließend der Schlüssel 20 durch die `remove`-Operation entfernt wurde.

(b*) Sei B_2 ein binärer Suchbaum, für den die Besuchsreihenfolge der Schlüssel in Preorder wie folgt bekannt ist: 8, 5, 3, 4, 6, 9, 10. Geben Sie B_2 in graphischer Darstellung an.

(c) Es seien zwei binäre Suchbäume T_1 und T_2 gegeben mit $y_1 < y_2$ für jedes Schlüsselpaar y_1 in T_1 und y_2 in T_2. Außerdem sei der Schlüssel w_1 an der Wurzel von T_1 und der Schlüssel w_2 an der Wurzel von T_2 bekannt.

Gesucht ist ein Algorithmus zum Verschmelzen der beiden Suchbäume, also zur Konstruktion eines binären Suchbaumes T, der alle Schlüssel aus T_1 und T_2 enthält. Dabei soll die Tiefe von T die größere der beiden Tiefen von T_1 und T_2 möglichst wenig überschreiten und der Algorithmus soll in Zeit $\mathcal{O}(\min\{\text{Tiefe}(T_1), \text{Tiefe}(T_2)\})$ laufen. Hierbei ist zu beachten, dass die Tiefen von T_1 und T_2 nicht von vornherein bekannt sind.

Für diese Aufgabe ist *kein* Code gefordert, eine Beschreibung der wesentlichen Schritte des Algorithmus zusammen mit der Begründung der Korrektheit und einer Laufzeitanalyse reichen aus.

---

*Für diese Teilaufgabe ist keine Begründung erforderlich.