Präsenzübung 2

Besprechung: 29.10.18 – 2.11.18

Aufgabe 1: Hashing (mündlich, keine Punkte)

Wir haben in der Vorlesung Hashing mit verketteten Listen als auch Hashing mit offener Adressierung kennengelernt. Hashing mit offener Adressierung haben wir dabei nur im Spezialfall von "linearem Sondieren" betrachtet, wo ein Element in die nächste freie Position, die auf die von der Hashfunktion eigentlich vorgesehene Position folgt, geschrieben wird.

Allgemeiner kann Hashing mit offener Adressierung wie folgt behandelt werden: Jede Position in der Hashtabelle kann höchstens ein Element speichern. Kollisionen werden wie folgt behandelt: Zu Beginn jeder INSERT-Operation eines neuen Elements mit Key k wird ein Kollisionszähler $i = 0$ gesetzt, welcher von der Hashfunktion $h(k, i)$ berücksichtigt wird: Ist Position $h(k, i)$ bereits belegt, dann wird i inkrementiert und die nächste Position $h(k, i)$ getestet, solange bis ein freier Platz gefunden wurde.

Beantworten Sie die folgenden beiden Fragen jeweils für den Fall von verketteten Listen als auch für den Fall von offener Adressierung.

a) Wieviele Elemente können maximal in einem Hashtable der Größe s gespeichert werden?

b) Sei m die Anzahl einzufügender Elemente. Was sind Vor-/Nachteile von $m \gg s$ und $m \ll s$?

Sei H nun eine leere Hashtabelle der Größe $s = 7$ mit den Positionen $0, \ldots, 6$. Fügen Sie die folgenden Elemente der Reihe nach in H ein: $9, 12, 2, 31, 10, 4$.

(c) ... unter Kollisionsbehandlung mittels verketteter Listen und $h(k) = k \bmod 7$

(d) ... mittels offener Adressierung und linearem Sondieren: $h(k, i) = (k + i) \bmod 7$

(e) ... mittels offener Adressierung und "doppeltem Hashing": $h(k, i) = (h_1(k) + i \cdot h_2(k)) \bmod 7$ für $h_1(k) = k \bmod 7$ und $h_2(k) = 1 + (k \bmod 6)$

Aufgabe 2: Eine Anwendung der Binären Suche (mündlich, keine Punkte)

Sei A ein sortiertes Array der Größe n und sei z eine gegebene Zahl. Das Ziel dieser Übung ist es, folgende Frage zu beantworten: Gibt es in A zwei verschiedene Elemente x und y, so dass $x+y = z$?

(a) Finden Sie einen (naiven) Algorithmus, der diese Frage in quadratischer Zeit $\mathcal{O}(n^2)$ beantwortet. Geben Sie den Pseudocode für einen solchen Algorithmus an und erklären Sie, warum Ihr Algorithmus die Laufzeit $\mathcal{O}(n^2)$ hat.

(b) Verwenden Sie nun die binäre Suche (Tipp: $y = z - x$), um einen effizienteren Algorithmus zu finden, der die obige Frage in einer Laufzeit von $\mathcal{O}(n \log n)$ beantworten kann. Geben Sie auch hier den Pseudocode an und begründen Sie die Korrektheit Ihres Algorithmus. Erklären Sie, warum Ihr Algorithmus die angegebene Laufzeit hat.

(c) Es ist klar, dass ein Algorithmus für die obige Frage *mindestens* die Laufzeit $\Omega(n)$ benötigt. Versuchen Sie, einen Algorithmus zu finden, der obige Frage in einer Laufzeit von tatsächlich $\mathcal{O}(n)$ beantwortet. Geben Sie den Pseudocode an und begründen Sie die Laufzeit und die Korrektheit Ihres Algorithmus. Tipp: Nennen Sie die Variablen x und y in *links* und *rechts* um - und verwenden Sie die Variablen entsprechend. Für den Korrektheitsbeweis machen Sie sich eine Invariante zunutze: In einem beliebigen Schritt im Algorithmus, was muss sicher gestellt sein, damit der Algorithmus korrekt ist. Es gibt eventuell Bereiche im Array, die Sie nicht mehr "betreten" werden.

Aufgabe 3: Heaps (mündlich, keine Punkte)

Betrachten Sie die folgenden drei in Array-Schreibweise gegebenen Binärbäume (diese Reihenfolge der Indizierung heißt auch "Level-Order"):

$$T_1 = \boxed{6}\boxed{3}\boxed{9}\boxed{2}\boxed{1}\boxed{7} \quad T_2 = \boxed{9}\boxed{7}\boxed{5}\boxed{8}\boxed{3}\boxed{6}\boxed{2} \quad T_3 = \boxed{9}\boxed{6}\boxed{8}\boxed{3}\boxed{5}\boxed{1}\boxed{4}\boxed{2}$$

Nur einer davon (T_a) erfüllt die Max-Heap-Eigenschaft. Ein zweiter (T_b) verletzt die Max-Heap-Eigenschaft an genau einer Stelle. Der dritte (T_c) kann nur durch mehrere Operationen in einen gültigen Max-Heap umgewandelt werden.

a) Bestimmen Sie $a, b, c \in \{1, 2, 3\}$.

b) Führen Sie auf T_b die HEAPIFY-Operation an der Fehlstelle aus. Geben Sie Zwischenschritte und das Resultat in Level-Order an.

c) Führen Sie auf T_a nacheinander die folgenden Operationen aus: Fügen Sie eine 7 hinzu mit INSERT(7), extrahieren Sie das Maximum mit EXTRACTMAX, und verringern Sie die 8 auf 0 mit DECREASE($8 \mapsto 0$). Geben Sie nach jeder Operation das Ergebnis in Level-Order an.

d) Erstellen Sie aus T_c einen Max-Heap mittels der Operation BUILDMAXHEAP. Geben Sie das Resultat in Level-Order an.

e) In BUILDMAXHEAP wird die HEAPIFY-Operation auf dem zugrundeliegenden Array von rechts nach links gehend angewandt. Kann man stattdessen auch von links nach rechts gehen (d.h. man beginnt an der Wurzel und geht dann in jedem darauffolgenden Level von links nach rechts)? Beweisen oder widerlegen Sie!

f) Erstellen Sie aus den originalen Array-Daten von T_c einen *ternären* Heap, indem sie ganz analog zum binären Fall die Einträge Level für Level von links nach rechts in einen ternären Baum schreiben. Erfüllt das Ergebnis die Max-Heap-Eigenschaft?