DAP2 – Präsenzübung 13

Besprechung: 19.07.2017 — 21.07.2017

Präsenzaufgabe 13.1: (Union-Find Datenstruktur)

a) Wir verwalten eine Union-Find Datenstruktur, wie sie in der Vorlesung definiert wurde. Zeichnen Sie die Datenstruktur nach jedem `Union`-Befehl:

```
Make-Set(a)  Make-Set(g)  Make-Set(h)  Union(g,h)
Make-Set(o)  Make-Set(i)  Union(a,o)   Union(i,h)
Make-Set(z)  Union(i,z)   Make-Set(j)  Union(j,a)
Union(a,i)
```

b) Nehmen wir an, es wurde statt doppelverketter Liste als Basis für die Union-Find Datenstruktur ein AVL-Baum verwendet. Beschreiben Sie die Konstruktion der Datenstruktur. Was wären die Laufzeiten für die `Make-Set`, `Find` und `Union` Operationen in dieser Version der Union-Find Struktur? Begründen Sie Ihre Antworten.

c) Nehmen wir an, die Basisstruktur für Union-Find ist eine binäre Max-Halde (Max-Heap). Beschreiben Sie die Konstruktion der Datenstruktur. Was wären die Laufzeiten für die `Make-Set`, `Find` und `Union` Operationen in dieser Version der Union-Find Struktur? Begründen Sie Ihre Antworten.

Zur Erinnerung: Ein binärer Max-Heap ist ein binärer Baum, sodass seine Wurzel den maximalen Wert im gesamten Baum enthält, und seine linke bzw. rechte Teilbäume ebenfalls binäre Heaps sind. Für Details sehen Sie bitte die Vorlesungsfolien über Heaps (angeboten als Zusatzmaterial).

Lösung:

a) Die Union-Find Struktur aus der Vorlesung enthält eine doppelverkettete Liste, wobei jeder Element der Liste einen zusätzlichen Zeiger enthält, der auf das erste Element der Liste ("Kopf") zeigt. Wir haben die Invariante, dass bei einer `Union`-Operation die kürzere Liste am Ende der längeren Liste aufgehängt wird. Somit haben wir:

```
Union(g,h): a→      g→h→
Union(a,o): a→o→      g→h→      i→
Union(i,h): a→o→      g→h→i→
Union(i,z): a→o→      g→h→i→z→
Union(j,a): a→o→j→      g→h→i→z→
```

`Union(a,i)`: g→h→i→z→a→o→j→

b) Die Operationen `Make-Set`, `Find` und `Union` haben bei der Implementierung mit dem Array die Worst-Case Laufzeiten von beziehungsweise $\mathcal{O}(1)$, $\mathcal{O}(1)$ und $\mathcal{O}(m)$ (bei der Eingabe zweier Listen mit jeweils m und n Elementen, mit $m \leq n$). Wenn wir jetzt als Basis der Struktur einen AVL-Baum verwenden würden, müsste jeder Knoten neben Standardzeiger nach Kinderknoten und Elternknoten auch den Zeiger auf die Wurzel des Baums haben. Dann haben `Make-Set` und `Find` wieder die Laufzeiten von $\mathcal{O}(1)$ – da man einen Baum mit einem Element in konstanter Zeit erstellt, bzw. die Kopfelemente zweier AVL-Bäume in konstanter Zeil vergleichen kann.

Probleme entstehen bei der `Union`-Operation, da man die AVL-Baum-Eigenschaft aufrechterhalten muss. Wir müssen alle Elemente aus der kleineren Struktur (mit m Elemente) in AVL-Baum zur richtigen Stelle einfügen (und gleichzeitig den Balance des Baums behalten). Für m Elemente kostet das im schlimmsten Fall jeweils $\mathcal{O}(\log(m+n))$ Laufzeit (am Anfang ist das $\mathcal{O}(\log n)$, aber beim letzten Element $\mathcal{O}(\log(n+m))$) – insgesamt $\mathcal{O}(m \log(m+n))$.

c) Die binären Heaps sind oft genutzt, wenn man in konstanter Zeit das Maximum (oder das Minimum) bestimmen soll, und gleichzeitig einen binären Baum mit n Elemente und der Höhe $\mathcal{O}(\log n)$ hat. Ähnlich zur Teilaufgabe b) lassen wir jedes Element des binären Baums (mit Heapeigenschaft = des binären Heaps) auf die Wurzel zeigen. Damit haben wir die Laufzeit $\mathcal{O}(1)$ für `Make-Set` und `Find`.

Zwei binäre Max-Heaps A und B (mit jeweils m und n Elemente) haben (im Vergleich zur AVL-Bäume) die Eigenschaft, dass man den maximalen Wert in Wurzel findet. Somit ist die Wurzel von `Union`(A, B) die größere von Wurzel(A) und Wurzel(B). Sei dies Wurzel(A). Sein rechter Teilbaum wird jetzt ganzer B, da alle seine Knoten kleiner als die Wurzel(A). Für den neuen linken Teilbaum muss man diesen Prozess rekursiv wiederholen für die "alten" linken und rechten Kindknoten – immer größeren als Wurzel auswählen, den Teilbaum mit kleinerem Wurzel vollständig übernehmen, und in anderem weiter Rekursion aufrufen, solange die Blätterknoten nicht erreicht sind.

Die Laufzeit für diesen Prozess ist $\mathcal{O}(\max\{\log m, \log n\})$. Da man aber die Eigenschaft der Union-Find Struktur wiederherstellen soll, muss man alle Elemente in neuem Heap auf neue Wurzel zeigen. Da alle Knoten, die früher in A gewesen sind, schon auf Wurzel(A) zeigen, müssen "nur" die Zeiger der Elemente aus B jetzt auf Wurzel(A) zeigen. Dieser Prozess benötigt $\mathcal{O}(\max\{m, n\})$ Zeit. Insgesamt brauchen wir für `Union` die Zeit $\mathcal{O}(\max\{\log m, \log n\}) + \mathcal{O}(\max\{m, n\}) = \mathcal{O}(\max\{m, n\})$, was gleich wie die Zeit bei den Implementation mit verketteten Listen ist.

Interessant: man kann aber die Worst-Case-Laufzeit der `Union`-Operation deutlich verbessern (nach $\mathcal{O}(\log n)$), wenn man eine andere Art der Heaps verwendet – sogenannte *Binomial Heaps*, deren Wurzel mehrere Kinderknoten haben kann, aber bei den die Operation `Find` ebenfalls $\mathcal{O}(\log n)$ kosten wird. Wir verweisen die Leser auf Buch von Cormen, Leiserson, Rivest und Stein, wo eine solche Struktur detalliert beschrieben ist.

Präsenzaufgabe 13.2: (Zweiter Übungstest, Aufgabe 1)

Kreuzen Sie jede Beziehung an, die gültig ist. Es gibt pro vollständig richtig ausgefüllter Zeile einen Punkt.

f	g	$f(n) \in O(g(n))$	$f(n) \in \Omega(g(n))$	$f(n) \in \Theta(g(n))$
$\frac{1}{10} \cdot 4^n$	$4n^4 + 3n^3$			
$\sqrt[4]{n}$	$(\ln n^2)^3 + 10$			
$n^5 - 5n^3 - 10n$	$10n^5 + 100n^4 + 1000n^3$			
$\sqrt[3]{n} \log^3 n$	$\frac{2}{3}n$			

Präsenzaufgabe 13.3: (Zweiter Übungstest, Aufgabe 2)

Gegeben ist die folgende Rekursionsgleichung:

$$T(n) = \begin{cases} 16 \cdot T\left(\frac{n}{4}\right) + n^3 & \text{, wenn } n > 1 \\ 1 & \text{, sonst.} \end{cases}$$

Sie können im Folgenden annehmen, dass n eine Viererpotenz ist.

a) Leiten Sie eine Funktion $f(n)$ her, für die $T(n) \in \Theta(f(n))$ gilt.

b) Zeigen Sie für die in Aufgabenteil a) angegebene Funktion f, dass $T(n) \in \mathcal{O}(f(n))$ gilt. Nutzen Sie dazu Induktion.

Präsenzaufgabe 13.4: (Zweiter Übungstest, Aufgabe 3)

Die Lufttemperaturen in einer Messstation auf Grönland wurden jeden Tag über n Tage gemessen und die Werte, die alle ganze Zahlen sind, in dem Array $A[1..n]$ gespeichert. Es fiel auf, dass die Temperatur zuerst bis zum Tag k, $1 < k < n$, immer strikt gesunken ist, und ab Tag k immer strikt gestiegen ist. Wir suchen den Index des Tags k.

Wenn beispielsweise das Array $A = \{5, 2, -1, -3, 0, 2\}$ ist, wollen wir den Index $k = 4$ bestimmen.

a) Entwerfen Sie einen Algorithmus, der den Index k des minimalen Elements in dem Array A bestimmt. Geben Sie den Algorithmus auch in Pseudocode an. Für die volle Punktzahl wird ein Algorithmus erwartet, dessen Laufzeit durch $\mathcal{O}(\log n)$ beschränkt ist.

b) Analysieren Sie die Laufzeit Ihres Algorithmus.

c) Beweisen Sie die Korrektheit Ihres Algorithmus.

Präsenzaufgabe 13.5: (Zweiter Übungstest, Aufgabe 4)

Die Aufgabe wurde im Heimblatt 13 als Aufgabe 3 gestellt.