Albert-Ludwigs-Universität
Institut für Informatik
Prof. Dr. F. Kuhn
M. Fuchs, G. Schmid

Algorithmen und Datenstrukturen
Sommersemester 2024
Musterlösung Übungsblatt 13

Abgabe: Dienstag, 23. Juli, 2022, 10:00 Uhr

Aufgabe 1: (Binäre) Heaps und Heapsort *(12 Bonus Punkte)*

(a) In vorherigen Übungsblättern haben Sie Algorithmen kennengelernt welche eine Prioritätswarteschlange erfordern, zum Beispiel der *Algorithmus von Prim* oder *Dijkstras Algorithmus*. Im dieser Aufgabe geht es darum selbst eine solche Datenstruktur zu implementieren. Implementieren Sie dafür einen *Binären Heap* basierend auf der in der Vorlesung erklärten *Array-Implementierung*. Der Heap soll folgende Funktionen unterstützen: `create`, `insert` (eines key-value pairs), `get_min` und `delete_min`. Sie dürfen dafür die Vorlage *heap.py* nutzen. *(5 Punkte)*

Hinweis: Um das `delete_min` effizient zu Implementieren tauscht man zuerst das Wurzelelement mit dem letzten Element im Array. Nun kann das letzte Element gelöscht werden. Anschließend muss man die Min-Heap Eigenschaft am Wurzelknoten prüfen und gegebenenfalls reparieren.

(b) Implementieren Sie Heapsort. Nutzen Sie dafür ihre Implementierung aus voriger Aufgabe.[1] Begründen Sie warum Heapsort eine Laufzeit von $O(n \log n)$ hat.

Begründen Sie warum es keine Prioritätswarteschlange geben kann, bei der sowohl `insert` wie auch `get_min` und `delte_min` konstante Laufzeit haben. *(3 Punkte)*

(c) In dieser Teilaufgabe betrachten wir einen *ternären* Heap. Diese sind ähnlich dem binären Heap, nur ist der zugrundeliegende Baum hier nicht binär, sondern es sind 3 Kinder von jedem Elternknoten aus erlaubt. Wir gehen wie auch beim binären Heap davon aus dass der Baum von oben nach unten (und von 'links' nach 'rechts') 'aufgefüllt' wird, sprich erst wenn der Baum bis zu einer gewissen Tiefe d vollständig ist, werden neue Knoten in Tiefe $d+1$ angelegt.

Geben Sie für einen solchen ternären Heap mit Tiefe d die minimale und maximale Anzahl von Knoten an. *(1 Punkt)*

Nehmen wir an, wir speichern die Elemente von einem ternären Heap in einem Array, beginnend ab Indexposition 1 (nicht 0). Sei i der Index von einem Knoten, der nicht die Wurzel und kein Blatt ist. Wie sind dann die Indizes der drei Kinder und wie der Index des Elternknotens? Geben sie dafür geschlossene Formeln an, ohne weitere Begründung. *(3 Punkte)*

Hinweis: Um potentielle Fließkommazahlen auf ganze Zahlen zu Runden dürfen sowohl 'Abrundklammern' $\lfloor \cdot \rfloor$, wie auch 'Aufrundklammern' $\lceil \cdot \rceil$ verwendet werden.

Musterlösung

(a) Siehe Lösungsdatei `heap.py`

[1] Sollten Sie die vorige Aufgabe nicht bearbeitet haben dürfen sie auch auf `heapq` zurückgreifen. Bei `heapq` entspricht `heappush` der `insert` und `heappop` der `delete-min` Operation aus der Vorlesung. Dabei können `heappush` und `heappop` auf Python-Listen angewendet werden (mehr Details hier).

(b) Eine Heapsort Implementierung ist ebenfalls in `heap.py` zu finden. Heaport fügt erst die n Elemente der Liste in einen Binary Heap ein (mittels `insert`). In einer zweiten Schleife wird der Heap mittels `get_min` und `delete_min` gelesen und wieder geleert. Die Laufzeiten von `insert` und `delete_min` sind logarithmisch in der Anzahl der Elemente. `get_min` sogar konstant. Somit ergibt sich folgende Gesamtlaufzeit:

$$\sum_{i=1}^{n} \underbrace{O(\log n)}_{\text{insert}} + \sum_{i=1}^{n} \left(\underbrace{O(1)}_{\text{get}} + \underbrace{O(\log n)}_{delete} \right) = O(n \log n)$$

Zur Frage warum die gegebenen 3 Operationen nicht alle konstant sein können: Angenommen dies wäre der Fall, dann hätte Heapsort eine Laufzeit von $O(n)$. Man könnte also beliebig verteilte Zahlen in linearer Zeit sortieren, dies steht im Widerspruch dazu dass jeder vergleichsbasierte Sortieralgorithmus eine Laufzeit von $\Omega(n \log n)$ hat (siehe Vorlesung 2).

(c) **Minimale Anzahl**: Der Baum ist bis zur Tiefe $d-1$ vollständig hat aber nur einen weiteren Knoten:

$$1 + \sum_{i=0}^{d-1} 3^i = 1 + \frac{3^d - 1}{3 - 1} = \frac{3^d + 1}{2}$$

Maximale Anzahl: Baum ist inklusive Tiefe d vollständig.

$$\sum_{i=0}^{d} 3^i = \frac{3^{d+1} - 1}{3 - 1} = \frac{3^{d+1} - 1}{2}$$

Index linkes Kind: $3 \cdot i - 1$
Index mittleres Kind: $3 \cdot i$
Index rechts Kind: $3 \cdot i + 1$
Index Elternknoten: $\left\lfloor \frac{i+1}{3} \right\rfloor$

Aufgabe 2: Verschiedene Hashverfahren *(8 Bonus Punkte)*

(a) Sei $h(s, j) := h_1(s) - 2j \bmod m$ und $h_1(x) := x + 2 \bmod m$. Fügen Sie folgende Schlüssel (in dieser Reihenfolge) 51, 13, 21, 30, 23 und 72 in eine Hashtabelle der Größe $m = 7$ ein. Nutzen dafür die Hashfunktion h und *lineares Sondieren* zur Kollisionsvermeidung. (Folgende Tabelle soll den finalen Zustand nach dem einfügen aller Zahlen zeigen.) *(1 Punkt)*

0	1	2	3	4	5	6

(b) Angenommen wir wollen die selbe Zahlenreihe wie in a) durch *quadratisches Sondieren* in eine Tabelle der Größe $m = 7$ speichern. Welche der folgenden Hashfunktionen wäre die *geeignetere* Wahl? Begründen Sie ihre Antwort!

- $h_1(x, i) := x + 6i + 2i^2 \bmod m$
- $h_2(x, i) := x + i + 4i^2 \bmod m$

Geben Sie die vollständige Hashtabelle unter Verwendung der *geeigneteren* Hashfunktion in folgender Tabelle an. *(2 Punkte)*

0	1	2	3	4	5	6

(c) Sei $h(s, j) := h_1(s) + j \cdot h_2(s) \bmod m$ wobei $h_1(x) = x \bmod m$ und $h_2(x) = 1 + \big(x \bmod (m-1)\big)$. Fügen Sie die Schlüssel 28, 59, 47, 13, 39, 69, 12 in die Hashtabelle der Größe $m = 11$. Verwenden Sie *Doppel-Hashing* zur Kollisionsvermeidung. *(2 Punkte)*

0	1	2	3	4	5	6	7	8	9	10

(d) Gegeben 2 Hashfunktionen $h_1(x) := x + 2 \bmod m$ und $h_2(x) := 3x \bmod m$ mit $m = 7$. Finden Sie 3 voneinander unterschiedliche natürliche Zahlen $u, v, w \in \mathbb{N}$, so dass $h_1(u) = h_1(v) = h_1(w)$ und $h_2(u) = h_2(v) = h_2(w)$ aber $h_1(x) \neq h_2(x)$ für $x \in \{u, v, w\}$. Markieren Sie in folgender Tabelle an welcher Position u und v unter Verwendung von *Cuckoo Hashing* stehen.

0	1	2	3	4	5	6

Würde man w nun auch noch hinzufügen, entsteht ein Zyklus. Um diesem zu entgehen, führen wir einen **Rehash** durch, indem wir unser Tabelle auf $m' = 11$ vergrößern und zwei neue Hashfunktionen h'_1 und h'_2 nutzen. Finden Sie h'_1 und h'_2 mit der Eigenschaft dass u, v und w in die neue Tabelle eingefügt werden können, ohne einen Zyklus zu erzeugen. (h'_1 und h'_2 sollen hierbei verschiedene Hashfunktionen der Form $(ax \bmod m')$ mit $a \not\equiv 0$ sein.) *(3 Punkte)*

Musterlösung

(a)

30	13	21	72	51	23	
0	1	2	3	4	5	6

(b) h_2 ist ungeeignet, da für die 23 kein entsprechend freies Feld gefunden werden kann:

$$h_2(51,0) = 2$$
$$h_2(13,0) = 6$$
$$h_2(21,0) = 0$$
$$h_2(30,6) = 5$$
$$h_2(23,0) = 2$$
$$h_2(23,1) = 0$$
$$h_2(23,2) = 6$$
$$h_2(23,3) = 6$$
$$h_2(23,4) = 0$$
$$h_2(23,5) = 2$$
$$h_2(23,6) = 5$$

Die mit h_1 ausgefüllte Tabelle ist folgende:

21	23	51	30		72	13
0	1	2	3	4	5	6

(c)

	69	13	47	59	39	28	12			
0	1	2	3	4	5	6	7	8	9	10

(d) Wir wählen $u := 2$, $v := 9$ und $w := 16$. Da diese Werte sich nur um ein vielfaches von m unterscheiden gilt die geforderte erste Bedingung. Die zweite Bedingung gilt auch da $h_1(u) = h_1(v) = h_1(w) = 4 \neq 6 = h_2(w) = h_2(v) = h_2(u)$. Die folgende Tabelle ergibt sich dadurch, dass u von seiner ersten Stelle 'verdrängt' und mittels h_2 an seine Alternativposition gesetzt wird.

				v		u
0	1	2	3	4	5	6

Also neue Funktionen wählen wir $h_1'(x) = x \bmod 11$ und $h_2'(x) = 2x \bmod 11$.