Institut für Theoretische Informatik

Leibniz Universität Hannover

Vorlesung: PD Dr. Arne Meier
Übung: Nicolas Fröhlich

Bearbeitungszeitraum 11.01.24 - 25.01.24

Datenstrukturen und Algorithmen

Hausübung 3 (Bäume, Hashing, Dynamische Programmierung)

Relevant aus dem Skript: Kapitel 9-13

Für alle Lösungen sind Begründungen anzugeben!

Organisatorisches und Hinweise zur Bearbeitung:

1. Bearbeiten Sie dieses Übungsblatt in Gruppen von 2-4 Leuten. (Das bedeutet insbesondere, dass Sie **keine Einzelabgaben** abgeben sollen!)
2. Schreiben Sie die Namen und Matrikelnummern **aller Gruppenmitglieder** sowohl in die Python-Datei als auch auf die PDF.
3. Verwenden Sie für die Programmieraufgaben die Datei *hausuebung03.py* und verändern Sie die bereits implementierten Datentypen und Funktionen nicht signifikant.
4. Erfolgreiche Testfälle garantieren **nicht**, dass ein Algorithmus korrekt implementiert wurde. Die Testfälle dienen nur dazu, grobe Fehler und falsche Ausgaben zu erkennen. Sie finden die Testfälle in der Datei *hausuebung03_test.py*.
5. Verwenden Sie keine Datenstrukturen oder Funktionen von Python außer:
   `list`, `dict`, `deque`, `PriorityQueue`, `enumerate`, `range` und `len`.
6. Geben Sie Ihre Lösung **bis spätestens** am 25.01.2024 23:59 im ILIAS Kurs unter folgendem Link ab: `https://ilias.uni-hannover.de/goto.php?target=crs_175670_rcodeYPKATwaXYQ`
7. Wenn Sie Ihre Lösung in LaTeX erstellt haben und als PDF- und Python-Datei abgeben, bekommen Sie einen Extrapunkt.

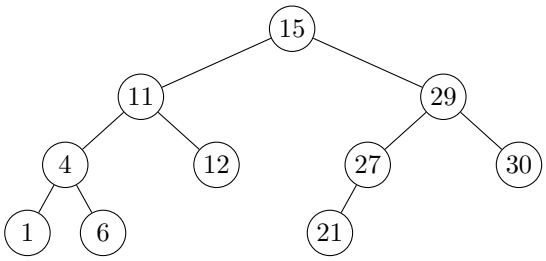
Hinweis zum Einsatz von KI: Für die Bearbeitung der Aufgaben ist der Einsatz von KI erlaubt, sofern er vollständig dokumentiert und der Abgabe beigefügt wird (Chat-GPT hat z.B. eine Share-Funktion. Alternativ auch als extra PDF).

Aufgabe 1 (3 + 3 + 4 Punkte)

a) Informieren Sie sich Online über die Datenstruktur **Splay Tree** und erklären Sie *kurz* die Funktionsweise, sowie Vor- und Nachteile im Vergleich zu AVL-Bäumen.

Geben Sie Ihre verwendeten Quellen an!

b) Führen Sie nun die Operationen `find`(12), `remove`(27) und `insert`(14) auf dem folgenden Splay Tree aus. Dokumentieren Sie Zwischenschritte!



c) Vervollständigen Sie den Splay Tree in *hausuebung03.py*, indem Sie die `splay` Methode implementieren. Kommentieren oder erklären Sie ihren Code!

Aufgabe 2 (4 + 5 + 1 Punkte)

Das **Kuckucks-Hashing**[1] verwendet zwei Hashtabellen T_1, T_2 und zwei Hashfunktionen h_1, h_2. Jedes Element x ist entweder an der Position $h_1(x)$ in der Tabelle T_1 (geschrieben als $T_1[h_1(x)]$) oder an der Position $h_2(x)$ in T_2, aber *nicht* an beiden Position. Das Einfügen eines neuen Elements x wird durch den folgenden Pseudocode beschrieben:

```
1 Procedure insert(x)
2     for MaxLoops Wiederholungen do
3         if T_1[h_1(x)] ist leer then T_1[h_1(x)] ← x; return
4         else Tausche die Werte von x und T_1[h_1(x)]
5         if T_2[h_2(x)] ist leer then T_2[h_2(x)] ← x; return
6         else Tausche die Werte von x und T_2[h_2(x)]
7     rehash()
8     insert(x)
```

Intuitiv wird also ein Element eingefügt, indem das bereits vorhandene Element "rausgeschmissen" wird und in die zweite Tabelle wandert, wo es einen Platz erhält, indem das dort vorhandene Element "rausgeschmissen" wird. Das Ganze wird so lange wiederholt, bis ein leerer Platz gefunden wird oder es zu viele Iterationen gibt. Im letzteren Fall werden beide Tabellen mit neuen Hashfunktionen neu aufgebaut und ein neuer Versuch gestartet.

a) Wenden Sie das Kuckucks-Hashing auf die Zahlen $10, 37, 29, 34, 28, 44, 18, 17, 22, 42, 49$ an. Nutzen Sie die Hashfunktionen

$$h_1(x) \coloneqq x \mod 7,$$
$$h_2(x) \coloneqq \lfloor 7 \cdot (x \cdot \Phi^{-1} - \lfloor x \cdot \Phi^{-1} \rfloor) \rfloor$$

und zwei Hashtabellen der Größe 7.

Hinweis: $\Phi^{-1} \approx 0.618034$ ist das Inverse des goldenen Schnitts. Bei den hier gewählten Zahlen und Hashfunktionen kommt es *nicht* zum rehashing!

b) Vervollständigen Sie die Methoden `insert`, `delete`, `lookup` in der Klasse `CuckooHashtable`. Kommentieren oder erklären Sie ihren Code!

c) Erklären Sie *kurz* Vor- und Nachteile vom Kuckucks-Hashing im Vergleich zum linearen Sondieren aus der Vorlesung.

Aufgabe 3 (2 + 6 + 2 Punkte)

Betrachten Sie folgendes Szenario:

In den Klauen eines mächtigen Drachen gefangen, bietet sich dir eine scheinbare Chance zur Freiheit. Der Drache reicht dir einen 20-seitigen Würfel und verkündet, dass deine Befreiung allein von der Summe der geworfenen Augenzahlen abhängt. Die Bedingung: Du musst eine von dir bestimmte Summe größer 20 erreichen, um deine Freiheit zu erlangen. Du darfst den Würfel so oft werfen, wie du möchtest, doch Vorsicht ist geboten, denn sobald die geworfene Summe größer ist als die von dir gewählte Zahl, endet deine Hoffnung auf Freiheit. Welche Zahl solltest du mit Bedacht wählen, um die größtmögliche Wahrscheinlichkeit für deine Freilassung zu gewährleisten?[2]

Wählen wir 25 und würfeln eine 15 und eine 10, so hätten wir gewonnen. Wenn wir stattdessen eine 6, 12 und 9 würfeln, dann hätten wir verloren.

[1] `https://www.sciencedirect.com/science/article/pii/S0196677403001925`

[2] Inspiriert von `https://www.youtube.com/watch?v=zF814SmPY5A`. Beachten Sie die Korrektur in den Kommentaren!

Wir wollen das Ganze auf Würfel mit beliebig vielen Seiten verallgemeinern. Die Wahrscheinlichkeit die Zahl x mit einem n-seitigen Würfel zu erreichen ist

$$p(x,n) = \begin{cases} \frac{1}{n} + \frac{1}{n} \cdot \left(\sum_{i=1}^{x-1} p(i,n) \right) & \text{wenn } x \leq n \\ \frac{1}{n} \cdot \left(\sum_{i=x-n}^{x-1} p(i,n) \right) & \text{sonst.} \end{cases}$$

Das Ziel dieser Aufgabe ist es $x_0 > n$ zu bestimmen, so dass $p(x_0, n)$ maximal ist, also $p(x_0, n) \geq p(x, n)$ für alle $x > n$ gilt. Das bedeutet, dass x_0 die Zahl größer n ist, die am wahrscheinlichsten als Summe von Würfen mit einem n-seitigen Würfel erreicht wird. Sie können annehmen, dass $x_0 \leq 2n + 1$ gilt.

a) Bestimmen Sie x_0 für einen 4-seitigen Würfel. Geben Sie ihre Rechnung vollständig an!

b) Implementieren Sie mit Hilfe der dynamischen Programmierung einen Algorithmus in Python, welcher x_0 in Abhänigkeit von n effizient berechnet.

   Effizient bedeutet hier, dass alle Testfälle in wenigen Sekunden berechnet werden können.

   Erklären Sie ihren Algorithmus und gehen Sie insbesondere auf ihre Verwendung von dynamischer Programmierung ein!

c) Erläutern Sie die Laufzeit ihres Algorithmus.