Prof. Dr. Leif Kobbelt
Stefan Dollase, Ira Fesefeldt, Alexandra Heuschling, Gregor Kobsik

Übung 6

Tutoraufgabe 1 (Gerrymandering):

Verschiedene Wahlsysteme haben verschiedene Stärken und Schwächen. Eine Anforderung an ein Wahlsystem ist, dass es nach der Wahl nicht nur einen Präsident für das ganze Land gibt, sondern auch einen Vertreter des lokalen Wahlkreises. Falls der Präsident jedoch nicht direkt, sondern von den Wahlkreisvertretern gewählt wird, so ergibt sich das Problem des Gerrymandering: Das Wahlergebnis hängt stark davon ab welche Personen welchem Wahlkreis zugeordnet sind. Die Person, welche bestimmt wer welchem Wahlkreis zugeordnet ist hat somit einen starken Einfluss auf den Wahlausgang. Wir betrachten dieses Problem nun in einem Zweiparteiensystem, das heißt jede Person kann entweder den Vertreter von Partei A oder den Vertreter von Partei B wählen. Die Wahlkreisvertreter können wiederum entweder den Präsidenten der Partei A oder den Präsidenten der Partei B wählen. Jeder Wahlkreisvertreter entscheidet sich für den Präsidenten der eigenen Partei.

a) Angenommen jeder Wahlkreis besteht aus 5 Personen. Wie viel Prozent der Wählerstimmen müssen mindestens für Vertreter von Partei A abgegeben werden, damit der Präsident von Partei A durch die Wahlkreisvertreter gewählt werden kann?

b) Angenommen jeder Wahlkreis besteht aus einer beliebigen aber festen Anzahl $k \in \mathbb{N}$ Personen.

i. Wie viel Prozent der Wählerstimmen müssen mindestens für Vertreter von Partei A abgegeben werden, damit der Präsident von Partei A durch die Wahlkreisvertreter gewählt werden kann?

ii. Wie muss k gewählt werden um mit einem möglichst geringen Prozentsatz trotzdem zu gewinnen.

Hinweise:

- Gehen Sie davon aus, dass in allen Wahlkreisen und während der Präsidentschaftswahl die Parteien nie gleich viele Stimmen bekommen.

Tutoraufgabe 2 (Selektionsalgorithmus mit 9er-Gruppen):

In der Vorlesung haben Sie gesehen, dass der Median der Mediane in linearer Zeit berechnet werden kann. Dazu wurde mithilfe des Medians der Mediane ein Pivot-Element gewählt. Bei der Berechnung des Medians der Mediane wurde zunächst die Eingabe in 5er-Gruppe unterteilt, anschließend jeweils der Median jeder 5er-Gruppe gebildet und schließlich der Median all dieser Mediane gebildet.
Angenommen die Eingabe wäre nicht in 5er-Gruppen eingeteilt worden, sondern in **9er-Gruppen**. Der restliche Selektionsalgorithmus bleibt hingegen unverändert.
Arbeitet der Selektionsalgorithmus weiterhin in linearer Zeit? Beweisen Sie Ihre Antwort.

Tutoraufgabe 3 (Geschlossenes Hashing mit großen Hashtabellen):

In dieser Aufgabe beschäftigen wir uns mit der erwarteten Anzahl an Sondierungen in einer Hashtabelle bei Nutzung von geschlossenem Hashing. Dafür möchten wir zeigen, dass bei Nutzung einer Hashtabelle, die ausreichend groß ist (genauer, dass $n \leq \frac{m}{2}$ mit n die Anzahl der Eingaben und m die Größe der Hashtabelle) die Länge einer solche Sondierungssequenz im Averagecase lediglich logarithmisch groß in der Größe der Eingabe ist.
Dazu nehmen wir an, dass wir *uniformes hashing* nutzen, d.h. für eine zufällige Eingabe (Schlüssel) x und eine uniforme Hashfunktion mit $j$ter Sondierung $h(x, j)$ auf eine Menge der Größe m ist jede Sequenz $h(x, 1) \dots h(x, m)$ aus unterschiedlichen Hashwerten gleich wahrscheinlich. Insbesondere ist damit auch jeder Wert für $h(x, j)$ gleich wahrscheinlich.
Sei hierfür X_i die Anzahl der Sondierungen, die beim $i$ten Einfügen in die Hashtabelle vorgenommen werden müssen.

Hinweis:

- Aufgrund der Annahme $n \leq \frac{m}{2}$ können wir etwas bessere Abschätzungen angeben als in der Vorlesung.

a) Wir nehmen an, wir nutzen uniformes Hashing. Zeigen Sie, dass die Wahrscheinlichkeit, dass wir für die $i$te Eingabe mit $i \leq n$ in eine Hashtabelle der Größe m mit $n \leq \frac{m}{2}$ echt mehr als k viele Sondierungen vornehmen müssen, maximal $\frac{1}{2^k}$ ist. Formal ausgedrückt, dass $\mathbb{P}(X_i > k) < \frac{1}{2^k}$ gilt.

b) Zeigen Sie, dass die Wahrscheinlichkeit, dass die $i$te Eingabe mit $i \leq n$ in eine Hashtabelle der Größe m mit $n \leq \frac{m}{2}$ mehr als $2\log_2(n)$ viele Sondierungen benötigt, in $O(\frac{1}{n^2})$ ist. Formal ausgedrückt, dass $\mathbb{P}(X_i > 2\log_2(n)) = O(\frac{1}{n^2})$ gilt.

c) Sei $X = \max_{1 \leq i \leq n} X_i$ die maximale Anzahl an Sondierung aller n Eingaben in eine Hashtabelle der Größe m mit $n \leq \frac{m}{2}$. Zeigen Sie, dass die Wahrscheinlichkeit, dass X größer ist als $2\log_2(n)$ in $O(\frac{1}{n})$ ist. Formal ausgedrückt, dass $\mathbb{P}(X > 2\log_2(n)) = O(\frac{1}{n})$.

d) Sei $X = \max_{1 \leq i \leq n} X_i$ die maximale Anzahl an Sondierung aller n Eingaben in eine Hashtabelle der Größe m mit $n \leq \frac{m}{2}$. Zeigen Sie, dass die maximalen Sondierungen pro Eingabe X im Averagecase in $O(\log(n))$ ist. Formal ausgedrückt, dass $\mathbb{E}(X) = O(\log(n))$.

Aufgabe 4 (Selektion des zweitkleinsten Elements): 7 Punkte

Entwerfen Sie einen Algorithmus, welcher aus n Elementen das zweitkleinste auswählt und dafür höchstens $n + \lceil \log_2(n) \rceil$ Vergleiche von Elementen benötigt. Dabei kann es hilfreich sein zusätzlich das kleinste Element zu finden. Begründen Sie Ihre Antwort.

Hinweis:

- Sie dürfen beliebig viele Vergleiche zwischen Booleans nutzen.

Aufgabe 5 (Selektionsalgorithmus mit 7er-Gruppen): 6 Punkte

In der Vorlesung haben Sie gesehen, dass der Median der Mediane in linearer Zeit berechnet werden kann. Dazu wurde mithilfe des Medians der Mediane ein Pivot-Element gewählt. Bei der Berechnung des Medians der Mediane wurde zunächst die Eingabe in 5er-Gruppe unterteilt, anschließend jeweils der Median jeder 5er-Gruppe gebildet und schließlich der Median all dieser Mediane gebildet.
Angenommen die Eingabe wäre nicht in 5er-Gruppen eingeteilt worden, sondern in **7er-Gruppen**. Der restliche Selektionsalgorithmus bleibt hingegen unverändert.
Arbeitet der Selektionsalgorithmus weiterhin in linearer Zeit? Beweisen Sie Ihre Antwort.

Aufgabe 6 (Hashing): 2 + 3 + 2 = 7 Punkte

a) Wir speichern n Objekte in einer einfach verketteten Liste mit Schlüssel key[o] und Hashwert h(key[o]) für Objekte o. Die Schlüssel sind beliebig lange Zeichenketten mit minimal Länge d und die Hashwerte haben maximal Länge m mit $m << d$.

Wie können Sie die Hashwerte ausnutzen, um ein Objekt o in der Liste zu suchen? Begründen Sie ihre Antwort.

b) In dieser Aufgabe nehmen wir einfaches uniformes Hashing an. Das heißt für eine zufällige Eingabe x und eine Hashfunktion h ist jeder Hashwert für $h(x)$ gleich wahrscheinlich.

Berechnen Sie die erwartete Anzahl an Kollisionen beim Hashen der verschiedenen Werte $k_1, \dots k_n$, das heißt berechnen Sie $\mathbb{E}(\ |\{\{k_i, k_j\} \mid k_i \neq k_j \text{ und } h(k_i) = h(k_j)\}|\ )$.

Hinweise:

- Beachten Sie, dass wir z.b. bei vier Elementen k_1, k_2, k_3, k_4 mit $h(k_1) = h(k_2) = h(k_3) = h(k_4)$ die 6 Kollisionen $\{k_1, k_2\}, \{k_1, k_3\}, \{k_1, k_4\}, \{k_2, k_3\}, \{k_2, k_4\}, \{k_3, k_4\}$ haben.
- Wegen einfachen uniformen Hashings einer Hashfunktion $h : A \to B$ folgt, dass für $x \neq y$ die Wahrscheinlichkeit, dass x und y gleich gehasht werden, uniform ist, also dass $\mathbb{P}(h(x) = h(y)) = \frac{1}{|B|}$.

- Nutzen Sie aus, dass der Erwartungswert linear ist, also dass $\mathbb{E}(\sum_{i=0}^{n} X_i) = \sum_{i=0}^{n} \mathbb{E}(X_i)$ gilt.

c) Angenommen wir benutzen offenes Hashing und wollen verhindern, dass bei einem zu hohen Füllgrad (= Anzahl der eingefügten Elemente geteilt durch die Größe der Hash-Tabelle) die Listen zu lang werden und dadurch die Suche in den Listen den Zugriffsaufwand dominiert.

Wir könnten dies erreichen, indem wir die Größe der Tabelle dynamisch anpassen, d.h. jedesmal, wenn der Füllgrad der Tabelle z.B. den Wert 2 übersteigt, erzeugen wir eine neue Tabelle mit doppelter Größe und überführen alle Element aus der alten Tabelle in die neue.

Ist dies eine sinnvolle Lösung?

Aufgabe 7 (Programmierung in Python - Binäre Suchbäume):

4 + 6 + 3 + 3 + 4 = 20 Punkte

Bearbeiten Sie die Python Programmieraufgaben. In dieser praktischen Aufgabe werden Sie sich mit Bäumen, genauer mit binären Bäumen und binären Suchbäumen, auseindersetzen. Diese Aufgabe dient dazu einige Konzepte der Vorlesung zu wiederholen.
Zum Bearbeiten der Programmieraufgabe können Sie einfach den Anweisungen des Notebooks *blatt06-python.ipynb* folgen. Das Notebook steht in der .zip-Datei zum Übungsblatt im Lernraum zur Vergügung.
Ihre Implementierung soll immer nach dem *# YOUR CODE HERE* Statement kommen. Ändern Sie keine weiteren Zellen.
Laden Sie spätestens bis zur Deadline dieses Übungsblatts auch Ihre Lösung der Programmieraufgabe im Lernraum hoch. Die Lösung des Übungsblatts und die Lösung der Programmieraufgabe muss im Lernraum an derselben Stelle hochgeladen werden. Die Lösung des Übungsblatts muss dazu als .pdf-Datei hochgeladen werden. Die Lösung der Programmieraufgabe muss als .ipynb-Datei hochgeladen werden.

Übersicht der zu bearbeitenden Aufgaben:

a) Binäre Suchbäume

- insert()
- is_binary_search_tree()
- bin_tree_2_list()
- list_2_bin_tree()
- bin_tree_2_bin_search_tree()