
Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: Jeweilige Tutorübung in der Woche vom 19. bis 23. Mai

Tutoraufgabe 1

Wir betrachten eine Schlüsselmenge Key und eine Hash-Tabelle der Größe $m \geq 2$. Zeigen Sie, dass es eine Familie von Hashfunktionen gibt, die für diese Schlüsselmenge und diese Hash-Tabelle 1-universell ist und gleichzeitig alle konstanten Funktionen $f_i(x) = i$ mit $i \in \{0, \dots, m-1\}$ enthält.

Tutoraufgabe 2

Die Werte $\{\text{Apfel, Banane, Kirsche, Himbeere, Melone}\}$ sollen in einer Hash-Tabelle der Größe $m = 4$ untergebracht werden. Es seien folgende Hashfunktionen gegeben:

f_1 : Apfel \mapsto 4	Banane \mapsto 2	Kirsche \mapsto 2	Himbeere \mapsto 1	Melone \mapsto 4
f_2 : Apfel \mapsto 3	Banane \mapsto 4	Kirsche \mapsto 2	Himbeere \mapsto 3	Melone \mapsto 4
f_3 : Apfel \mapsto 2	Banane \mapsto 2	Kirsche \mapsto 4	Himbeere \mapsto 1	Melone \mapsto 1
f_4 : Apfel \mapsto 1	Banane \mapsto 3	Kirsche \mapsto 3	Himbeere \mapsto 4	Melone \mapsto 4
g_1 : Apfel \mapsto 1	Banane \mapsto 1	Kirsche \mapsto 3	Himbeere \mapsto 2	Melone \mapsto 3
g_2 : Apfel \mapsto 2	Banane \mapsto 4	Kirsche \mapsto 2	Himbeere \mapsto 3	Melone \mapsto 4
g_3 : Apfel \mapsto 4	Banane \mapsto 4	Kirsche \mapsto 1	Himbeere \mapsto 4	Melone \mapsto 2
g_4 : Apfel \mapsto 3	Banane \mapsto 1	Kirsche \mapsto 2	Himbeere \mapsto 3	Melone \mapsto 3
g_5 : Apfel \mapsto 4	Banane \mapsto 2	Kirsche \mapsto 2	Himbeere \mapsto 2	Melone \mapsto 3

In der Vorlesung haben wir den Begriff der c -universellen Hashfunktionen kennengelernt.

- Geben Sie für die Familie $\mathcal{H}_1 = \{f_1, f_2, f_3, f_4\}$ das kleinste c an, so dass \mathcal{H}_1 c -universell ist.
- Finden Sie eine möglichst kleine Familie $\mathcal{H}_2 \subseteq \{g_1, g_2, g_3, g_4, g_5\}$, die 1-universell ist.

Begründen Sie Ihre Aussagen.

Tutoraufgabe 3

Veranschaulichen Sie Hashing mit Linear Probing und Hashing mit Chaining. Die Größe der Hash-Tabelle ist dabei jeweils $m = 11$. Führen Sie die folgenden Operationen aus.

Insert 3, 11, 9, 7, 14, 23, 4, 12, 15, 8, 1
Delete 23
Insert 25

Verwenden Sie die Hashfunktion

$$h_5(x) = 5x \bmod 11.$$

Bei dieser Aufgabe sind die Schlüssel der Elemente die Elemente selbst.

Im Kontext von Hashing mit Linear Probing soll beim Löschen die dritte Methode aus der Vorlesung verwendet werden, das heißt, die Wiederherstellung der folgenden Invariante: Für jedes Element e in der Hash-Tabelle mit Schlüssel $k(e)$, aktueller Position j und optimaler Position $i = h_5(k(e))$ sind alle Positionen $i, (i + 1) \bmod m, (i + 2) \bmod m, \dots, j$ der Hash-Tabelle belegt. Bei dieser Aufgabe soll keine dynamische Größenanpassung der Hash-Tabelle stattfinden.

Hausaufgabe 1

Wir betrachten eine Folge von $m \geq 1$ Inkrement-Operationen iterativ angewandt auf $z_0 = 0$, das heißt, dass die k -te Inkrementoperation σ_k die Zahl $z_{k-1} = k - 1$ zur Zahl $z_k = k$ inkrementiert. Wir nehmen an, dass alle Zahlen in Binärdarstellung geschrieben werden. Außerdem nehmen wir (vereinfachend) an, dass jeder Bitflip Laufzeit 1 hat und die Laufzeit jeder Inkrement-Operation gerade die Anzahl der Bits ist, die durch die Operation geflippt werden.

Da die Zahl $z_m = m$ als letzte produzierte Zahl von allen Zahlen z_0, \dots, z_m die größte Bitlänge besitzt, ist klar, dass jede Inkrement-Operation nicht mehr als $\lceil \log m \rceil$ viele Bits flippt. Eine pessimistische Analyse würde also zeigen, dass die Worst-Case-Laufzeit maximal $m \cdot \lceil \log m \rceil \in \mathcal{O}(m \log m)$ beträgt. Mithilfe einer amortisierten Analyse können wir aber eine bessere Schranke zeigen.

Finden Sie ein Amortisationsschema Δ für das die amortisierte Laufzeit einer Operationsfolge der Länge m in $\mathcal{O}(m)$ liegt. Zeigen Sie außerdem, dass Ihr Amortisationsschema zulässig ist, indem Sie nachweisen, dass das Token-Konto stets nichtnegativ ist. Können Sie Δ so finden, dass die amortisierte Laufzeit einer Operationsfolge der Länge m sogar nicht größer als $2m$ ist?

Hinweis: Bezeichne $B_{0 \rightarrow 1}(\sigma_k)$ und $B_{1 \rightarrow 0}(\sigma_k)$ die Anzahl der Bitflips von 0 zu 1 bzw. von 1 zu 0 durch die k -te Inkrement-Operation σ_k . Machen Sie die Anzahl der Token, die durch σ_k auf das Konto eingezahlt bzw. abgehoben werden, von $B_{0 \rightarrow 1}(\sigma_k)$ und $B_{1 \rightarrow 0}(\sigma_k)$ abhängig. Gibt es eine generelle Beziehung zwischen einem Bitflip von 0 zu 1 und von 1 zu 0 an einer festen betrachteten Binärstelle?

Hausaufgabe 2

Veranschaulichen Sie Hashing mit Linear Probing und Hashing mit Chaining. Die Größe der Hash-Tabelle ist dabei jeweils $m = 13$. Führen Sie die folgenden Operationen aus.

Insert	16, 3, 12, 17, 29, 10, 24
Delete	16
Insert	5, 1, 14
Delete	10
Insert	10
Delete	1

Verwenden Sie die Hashfunktion

$$h_3(x) = 3x \bmod 13.$$

Bei dieser Aufgabe sind die Schlüssel der Elemente die Elemente selbst.

Im Kontext von Hashing mit Linear Probing soll beim Löschen die dritte Methode aus der Vorlesung verwendet werden, das heißt, die Wiederherstellung der folgenden Invariante: Für jedes Element e in der Hash-Tabelle mit Schlüssel $k(e)$, aktueller Position j und optimaler Position $i = h_3(k(e))$ sind alle Positionen $i, (i + 1) \bmod m, (i + 2) \bmod m, \dots, j$ der Hash-Tabelle belegt. Bei dieser Aufgabe soll keine dynamische Größenanpassung der Hash-Tabelle stattfinden.

Hausaufgabe 3

Bestimmen Sie die Wahrscheinlichkeit, dass von n Personen mindestens zwei Personen am gleichen Tag im Jahr Geburtstag haben. Erläutern Sie den Zusammenhang zwischen Hashing und dem eben gezeigten Geburtstagsparadoxon.

Hinweis: Bestimmen Sie die Wahrscheinlichkeit, des Komplementär-Ereignisses: Keine zwei Person haben am gleichen Tag Geburtstag. Ein Jahr hat hier immer 365 Tage.

Hausaufgabe 4

Implementieren Sie die Methoden `insert`, `remove` und `find` für Hashing mit Linear Probing. Verwenden Sie die Hashfunktion

$$h(x) = ax \bmod m.$$

Stellen Sie sicher, dass nach dem Löschen eines Elements die folgende Invariante: Für jedes Element e in der Hash-Tabelle mit Schlüssel $k(e)$, aktueller Position j und idealer Position $i = h(k(e))$ sind alle Positionen $i, (i + 1) \bmod m, (i + 2) \bmod m, \dots, j$ der Hash-Tabelle belegt. Bei dieser Aufgabe soll keine dynamische Größenanpassung der Hash-Tabelle stattfinden. Ist die Hash-Tabelle bereits voll, wenn eine Insert-Operation stattfindet, dann soll der Inhalt der Hash-Tabelle nicht verändert werden.

Verwenden Sie für Ihre Implementierung die auf der Übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die Klasse `LinHash`.

Achten Sie bei der Abgabe Ihrer Aufgabe darauf, dass Ihre Klasse `LinHash` heißt und auf den Rechnern der Linuxhalle (`lxhalle.informatik.tu-muenchen.de`) mit der bereitgestellten Datei `main_h` kompiliert werden kann. Anderenfalls kann eine Korrektur nicht garantiert werden. Achten Sie darauf, dass Ihr Quelltext ausreichend kommentiert ist.

Schicken Sie die Lösung per Email mit dem Betreff `[GAD] Gruppe <Gruppennummer>` an ihren Tutor.