Datenstrukturen und Algorithmen

Abgabe: 14.06.2017

Georg C. Dorndorf Matr.Nr. 366511
Adrian C. Hinrichs Matr.Nr. 367129

# 4	# 5	# 6	Σ
6/6	7/8	3/6	16/20

Aufgabe 4

a)

Sei k der in die Hastabelle einzufügende Schlüssel. Sei unsere Hashfunktion h durch die der Divisionsmethode gegeben. Also $h(k) = k \mod m$. Die Länge des Arrays ist als 11 gegeben wir wählen also $m = 11$.

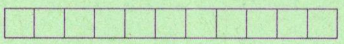
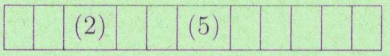


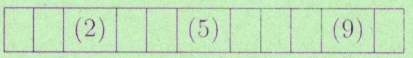
Abbildung 1: Hashtabelle als Array mit 11 Feldern.

Wir fügen den Schlüssel 2 in unsere Tabelle ein und berechnen $h(2) = 2 \mod 11 = 2$. Also wird der Schlüssel wie folgt eingefügt.

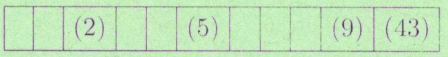
Wir fügen den Schlüssel 5 in unsere Tabelle ein und berechnen $h(5) = 5 \mod 11 = 5$. Also wird der Schlüssel wie folgt eingefügt.



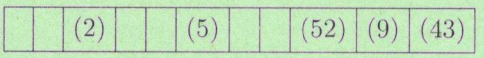
Wir fügen den Schlüssel 9 in unsere Tabelle und berechnen $h(9) = 9 \mod 11 = 9$. Also wird der Schlüssel wie folgt eingefügt.



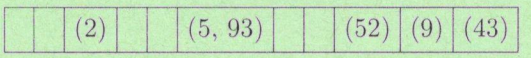
Wir fügen den Schlüssel 43 in unsere Tabelle und berechnen $h(43) = 43 \mod 11 = 10$. Also wird der Schlüssel wie folgt eingefügt.



Wir fügen den Schlüssel 52 in unsere Tabelle und berechnen $h(52) = 52 \mod 11 = 8$. Also wird der Schlüssel wie folgt eingefügt.



Wir fügen den Schlüssel 93 in unsere Tabelle und berechnen $h(93) = 93 \mod 11 = 5$. Es entsteht eine Kollision, da Kollisionen in der vorliegenden Hashtabelle durch Verkettungen aufgelöst werden sollen wird der Schlüssel wie folgt eingefügt.



b)

Sei T die aus Aufgabe a) resultierende Hastabelle. Es gibt $m = 11$ Positionen in der Hashtabelle. Unter der Annahme[1], dass die Menge der möglichen Schlüssel exakt der Menge der in der Aufgabe genannten Werte entspricht haben wir $n := 6$ mögliche Schlüssel. Nach Vorlesung lässt sich der Füllgrad der Tabelle T jetzt durch $\alpha(n, m) = \frac{n}{m} = \frac{6}{11} = 0.\overline{54} \approx 0.55$ berechnen.

Aufgabe 5

a)

Nach aufgabenstellung ist $c = 0.62$, offensichtlich ist $m = 8$. Die hashfunktion ist also

$$\begin{aligned} h(k) &= \lfloor m \cdot (k \cdot c \mod 1) \rfloor \\ &= \lfloor 8 \cdot (k \cdot 0.62 \mod 1) \rfloor \end{aligned} \tag{I}$$

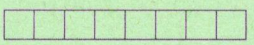


Abbildung 2: Leere Hash-Tabelle

Es gilt:

$$\begin{aligned} h(3) &= \lfloor 8 \cdot (3 \cdot 0.62 \mod 1) \rfloor \\ &= \lfloor 8 \cdot (1.86 \mod 1) \rfloor \\ &= \lfloor 8 \cdot .86 \rfloor \\ &= \lfloor 6.88 \rfloor \\ &= 6 \end{aligned}$$

Also wird 3 wie folgt eingeordnent:

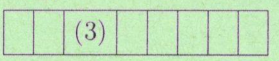


Abbildung 3: Nach einordnung des Elementes 3

[1] Diese Annahme wird mangels näherer Definition des Wertebereichs der Schlüssel in der Aufgabe getroffen.

Es gilt:

$$\begin{aligned} h(9) &= \lfloor 8 \cdot (9 \cdot 0.62 \mod 1) \rfloor \\ &= \lfloor 8 \cdot (5.58 \mod 1) \rfloor \\ &= \lfloor 8 \cdot .58 \rfloor \\ &= \lfloor 4.64 \rfloor \\ &= 4 \end{aligned}$$

Also wird 9 wie folgt eingeordnent:

		(3)		(9)			

Abbildung 4: Nach einordnung des Elementes 9

Es gilt:

$$\begin{aligned} h(6) &= \lfloor 8 \cdot (6 \cdot 0.62 \mod 1) \rfloor \\ &= \lfloor 8 \cdot (3.72 \mod 1) \rfloor \\ &= \lfloor 8 \cdot .72 \rfloor \\ &= \lfloor 5.76 \rfloor \\ &= 5 \end{aligned}$$

Also wird 6 wie folgt eingeordnent:

		(3)		(9)	(6)		

Abbildung 5: Nach einordnung des Elementes 6

Es gilt:

$$\begin{aligned} h(6) &= \lfloor 8 \cdot (6 \cdot 0.62 \mod 1) \rfloor \\ &= \lfloor 8 \cdot (3.72 \mod 1) \rfloor \\ &= \lfloor 8 \cdot .72 \rfloor \\ &= \lfloor 5.76 \rfloor \\ &= 5 \end{aligned}$$

Also wird 6 wie folgt eingeordnent:

		(3)		(9)	(6)		

Abbildung 6: Nach einordnung des Elementes 6

Es gilt:

$$\begin{aligned} h(97) &= \lfloor 8 \cdot (97 \cdot 0.62 \mod 1) \rfloor \\ &= \lfloor 8 \cdot (60.14 \mod 1) \rfloor \\ &= \lfloor 8 \cdot .14 \rfloor \\ &= \lfloor 1.12 \rfloor \\ &= 1 \end{aligned}$$

Also wird 97 wie folgt eingeordnent:

	(97)	(3, 7)		(9)	(6)		

Abbildung 7: Nach einordnung des Elementes 97

Es gilt:

$$\begin{aligned} h(17) &= \lfloor 8 \cdot (17 \cdot 0.62 \mod 1) \rfloor \\ &= \lfloor 8 \cdot (10.54 \mod 1) \rfloor \\ &= \lfloor 8 \cdot .54 \rfloor \\ &= \lfloor 4.32 \rfloor \\ &= 4 \end{aligned}$$

Also wird 17 wie folgt eingeordnent:

	(97)	(3, 7)		(9, 17)	(6)		

Abbildung 8: Nach einordnung des Elementes 17

Es gilt:

$$\begin{aligned} h(77) &= \lfloor 8 \cdot (17 \cdot 0.62 \mod 1) \rfloor \\ &= \lfloor 8 \cdot (47.74 \mod 1) \rfloor \\ &= \lfloor 8 \cdot .74 \rfloor \\ &= \lfloor 5.92 \rfloor \\ &= 5 \end{aligned}$$

Also wird 77 wie folgt eingeordnent:

	(97)	(3, 7)		(9, 17)	(6, 77)		

Abbildung 9: Nach einordnung des Elementes 77

Es gilt:

$$\begin{aligned} h(41) &= \lfloor 8 \cdot (41 \cdot 0.62 \mod 1) \rfloor \\ &= \lfloor 8 \cdot (25.42 \mod 1) \rfloor \\ &= \lfloor 8 \cdot .42 \rfloor \\ &= \lfloor 3.36 \rfloor \\ &= 3 \end{aligned}$$

Also wird 41 wie folgt eingeordnent:

	(97)	(3, 7)	(41)	(9, 17)	(6, 77)		

Abbildung 10: Nach einordnung des Elementes 41

b)

Sei T die aus Aufgabe a) resultierende Hastabelle. Es gibt $m = 8$ Positionen in der Hashtabelle. Unter der Annahme [2], dass die Menge der möglichen Schlüssel exakt der Menge der in der Aufgabe genannten Werte entspricht haben wir $n := 8$ mögliche Schlüssel. Nach Vorlesung lässt sich der Füllgrad der Tabelle T jetzt durch $\alpha(n, m) = \frac{n}{m} = \frac{8}{8} = 1$ berechnen.

[2] Diese Annahme wird mangels näherer Definition des Wertebereichs der Schlüssel in der Aufgabe getroffen.

Georg C. Dorndorf Matr.Nr. 366511
Adrian C. Hinrichs Matr.Nr. 367129

Aufgabe 6

Nein es ist nicht sinnvoll die Listen zur Kollisionsauflösung der Hashtabelle sortiert zu halten. Dies liegt daran, dass die Laufzeit für das sortiert halten von Listen mit den aktuell bekannten Algorithmen im Worst-Case aus $\Theta(n)$ ist, da um die Liste sortiert zu halten bei jeder Einfügeoperation in die Liste eine lineare Suche durchgeführt werden muss. Des weiteren ist das sortiert halten der Liste überflüssig, da das extrahieren eines Elements in jedem Fall folgender Laufzeitabschätzung gehorcht:

Das extrahieren eines Elements aus einer Liste aus der Hashtabelle benötigt im Worst-Case ebenfalls – unabhängig davon ob die Liste sortiert ist oder nich – eine Laufzeit aus $\Theta(n)$, da auch hier wieder die lineare Suche die schnellste ist.

Die Laufzeit um eine Element in die Liste einzufügen ohne diese dabei soriert zu halten ist aus $\Theta(1)$, da das Element einfach am Anfang der Liste eingefügt werden kann.

Das Löschen eines Elements benötigt offensichtlicherweise die gleiche Laufzeit wie das Suchen eines Elements, da auch hier zunächst das Element gesucht werden muss.

Die Laufzeit für eine erfolgreiche Suche ist mit Vorlesung 1 und 2 ist ebenfalls aus $\Theta(n)$.

Insgesamt ergibt sich also, dass es effizienter ist die Listen zur Kollisionsauflösung nicht sortiert zu halten.

Average case beachten! -3

/3