A Hashtabelle konkret

Gegeben sei eine Hashtabelle H mit 11 Einträgen und einfach verketteter Liste zur Kollisionsauflösung. Dabei sei die Hashfunktion h definiert über $h(x) = x \mod 11$. Wir verwenden in dieser Aufgabe die Zahlenfolge $F = [75, 44, 46, 53, 14, 2, 40, 61, 87, 86]$.

A.1 Einfügen (2 Punkte)

Fügen Sie die Zahlen aus F in der gegebenen Reihenfolge in H ein und geben Sie den Zustand von H nach dem Einfügen von 14 und nach dem Einfügen von 86 an.

Musterlösung:

Es gilt: $h(75) = 9$, $h(44) = 0$, $h(46) = 2$, $h(53) = 9$, $h(14) = 3$, $h(2) = 2$, $h(40) = 7$, $h(61) = 6$, $h(87) = 10$ und $h(86) = 9$.

Nach dem Einfügen von 14

0	1	2	3	4	5	6	7	8	9	10
→ 44		→ 46	→ 14						→ 53 → 75	

Nach dem Einfügen von 86

0	1	2	3	4	5	6	7	8	9	10
→ 44		→ 2 → 46	→ 14			→ 61	→ 40		→ 86 → 53 → 75	→ 87

A.2 Suchen (1 Punkt)

Nehmen Sie an, dass nach dem Einfügen aller Zahlen $x \in F$ nach jeder Zahl mit gleicher Wahrscheinlichkeit $p(x)$ gesucht wird (d.h., $p(x) = \frac{1}{|F|}$ für alle $x \in F$). Wie hoch sind dann die erwarteten Kosten für die erfolgreiche Suche in H nach einem beliebigen Element $x \in F$?

Musterlösung:

Erwartete Kosten: $\frac{14}{10}$

Algorithmen-Übungsblatt 4
Musterlösung Aufgabe B

SS 2018

B Funktionale Relationen (4 Punkte)

Hinweis: Die Aufgabe ist etwas unglücklich gestellt worden. Vorauszusetzen, die Anzahl der Kollisionen einer Hashtabelle sei (deterministisch) konstant beschränkt, ist eine kühne, in der Realität nicht haltbare Behauptung. Bei geeigneter Wahl der Hashfunktion und der Größe des Hashtabellen-Arrays kann man höchstens eine *erwartet* konstant beschränkte Kollisionsanzahl annehmen, also auch nur *erwartete* Laufzeit in $\mathcal{O}(|R|)$. Außerdem verletzt die Aufgabe mit der Annahme $\text{key}((x,y)) = x$ die Konvention der VL, dass zwei Elemente genau dann gleich sind, falls ihre Keys gleich sind, was sicherlich nicht für alle Paare (x,y) gilt. Die folgende Musterlösung allerdings verwendet die unrealistischen Annahmen der Aufgabenstellung, um weiterer Verwirrung vorzubeugen.

Die einfachste Datenstruktur zur Darstellung binärer Relationen ist eine verkettete Liste von Tupeln $(a,b) \in R$, was wir unten auch als Eingabe verwenden.

Für die Funktion *isFunctional* benutzen wir eine Hashtabelle mit verketteten Listen und einer universellen Hashfunktion. Die Größe der Hashtabelle muss hierbei in $\Omega(|A|)$ liegen, damit die Anzahl der Kollisionen konstant beschränkt ist (siehe Aufgabenstellung).

Es ergibt sich folgender Algorithmus:

```
function isFunctional(R : List of A × B) : Bool
    H := new Hashtable
    c := 0    // Zähler für Anzahl der verschiedenen a ∈ A

    // Überprüfe, ob Relation rechtseindeutig
    foreach p = (a, b) ∈ R do
        p' := (a', b') := H.find(a)
        if p' = ⊥ then
            H.insert(p)
            c++
        else if b' ≠ b then
            return false    // Rechtseindeutigkeit verletzt

    // Überprüfe, ob Relation linkstotal
    if c ≠ |A| then
        return false

    return true
```

Die Laufzeit des Algorithmus liegt in $\mathcal{O}(|R|)$, da die for-Schleife $|R|$-mal durchlaufen wird und in der for-Schleife die *insert*- und *find*-Operationen wegen der als konstant vorausgesetzten Kollisionsanzahl in $\mathcal{O}(1)$ laufen. Die Überprüfung auf Linkstotalität erfolgt ebenfalls in konstanter Zeit (unter der Annahme, dass $|A|$ in $\mathcal{O}(1)$ abfragbar ist).

C Mengenoperationen (3 Punkte)

Sei M die Menge aller Matrikelnummern. Weiter gibt es eine Menge $W \subseteq M$ von Studenten, die sich über WebInscribe bei einem Tutorium angemeldet haben und eine Menge $K \subseteq M$ von Studenten, die sich für die Klausur angemeldet haben. Geben Sie einen Algorithmus an, der in $\mathcal{O}(|K| + |W|)$ die Mengen $A = K \cap W$, $B = K \setminus W$ und $C = W \setminus K$ von Studenten berechnet, die bei beiden Veranstaltungen angemeldet sind, sowie von Studenten, die jeweils nur an einer teilnehmen. Begründen Sie die Laufzeit Ihres Algorithmus.

C Mengenoperationen Musterlösung

Für alle drei Algorithmen A, B, C, die jeweils die Mengen A, B, C berechnen, verwenden wir eine Hashtabelle H. Um die geforderte Laufzeit zu garantieren, muss diese Hashtabelle ausreichend groß sein, daher in $\Omega(|W|)$ bzw. in $\Omega(|K|)$ (siehe Vorlesung am 07.05.18, Folie 16).

Algorithmus A

Sei $A := \emptyset$. Hashe alle $k \in K$ ($H.insert(k)$). Wende nun $find(w)$ auf alle $w \in W$ an und überprüfe so, ob w bereits in H ist. Wenn dem so ist, dann sei nun $w \in A$.

Algorithmus B

Sei $B := \emptyset$. Hashe alle $w \in W$. Überprüfe nun für alle $k \in K$, ob $find(k) = \perp$. In diesem Fall sei $k \in B$.

Algorithmus C

Sei $C := \emptyset$. Hashe alle $k \in K$. Überprüfe nun für alle $w \in W$, ob $find(w) = \perp$. In diesem Fall sei $w \in C$.

Für die Mengen A, B, C kann eine beliebige Datenstruktur verwendet werden, welche das Einfügen von Elementen in $\mathcal{O}(1)$ unterstützt.
Aufgrund der Größe von H ist die Laufzeit von $insert$, $find$ und $remove$ jeweils erwartet in $\mathcal{O}(1)$. Das Einfügen, Löschen, oder Finden von n Elementen in der Hashtabelle ist damit erwartet in $\mathcal{O}(n)$. Damit sind die Algorithmen A, B, C erwartet in $\mathcal{O}(|K| + |W|)$.