Universität Stuttgart

Institut für Visualisierung
und Interaktive Systeme

Übungsblatt 10

Datenstrukturen und Algorithmen (SS 2018)

Abgabe: Mittwoch, 11.07.2018, 23:55 Uhr — Besprechung: ab Montag, 18.07.2018

Bitte lösen Sie die Übungsaufgabe in **Gruppen von 3 Studenten** und wählen EINEN Studenten aus, welcher die Lösung im ILIAS als **PDF** als **Gruppenabgabe** (unter Angabe aller Gruppenmitglieder) einstellt. Bitte erstellen Sie dazu ein **Titelblatt**, welches die Namen der Studenten, die Matrikelnummern, und die E-Mail-Adressen enthält.

Aufgabe 1 Textalgorithmen *[Punkte: 5]*

Wenden Sie den Algorithmus zur Berechnung der Levenshtein-Distanz auf die Eingabe "Sommer" und "Sonne" an. Die folgenden Vorschriften sind dabei vorgegeben:

$$D_{i,j} = min \begin{cases} D_{i-1,j-1} + 0, & \text{gleiches Zeichen} \\ D_{i-1,j-1} + 1, & \text{Zeichen ersetzen} \\ D_{i,j-1} + 1, & \text{Zeichen einfügen} \\ D_{i-1,j} + 1, & \text{Zeichen löschen} \end{cases}$$

(a) *(4 Punkte)* Füllen Sie dazu die gegebene Tabelle aus:

	ϵ	S	o	m	m	e	r
ϵ							
S							
o							
n							
n							
e							

(b) *(1 Punkt)* Geben Sie die Levenshtein-Distanz der Wörter "Sommer" und "Sonne" an. An welcher Position in der Tabelle kann dieser Wert nach Ausführung des Algorithmus abgelesen werden?

Levenshtein-Distanz:

.....

Position in der Tabelle:

.....

.....

Aufgabe 2 Hashing *[Punkte: 9]*

Gegeben sei ein Programm zur Verwaltung von Sitzplatzreservierungen in einem Kino. Dabei werden die Reservierungen für die Sitznummern 1-111 mittels einer Hashfunktion gespeichert.

Die folgende Hashfunktion soll verwendet werden:

- $h(x) = x \mod 11$, mit x = Sitznummer. Indexbereich der Hashtabelle 0–10.
  (Beispiel: $h(42) = 9$*)*

Gegeben sind nun die Sitznummern 8, 87, 24, 41, 96, 69, 36, 33 und 30.

(a) *(3 Punkte)* Zeichnen Sie die resultierende Hashtabelle für offenes Hashing mit Verkettung.

Index	Entry
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

(b) *(3 Punkte)* Zeichnen Sie die resultierende Hashtabelle für geschlossenes Hashing mit Linearem Sondieren mit $c = 2$.

Index	Entry
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

(c) *(3 Punkte)* Zeichnen Sie die resultierende Hashtabelle für geschlossenes Hashing mit Quadratischem Sondieren (wie in der Vorlesung besprochen).

Index	Entry
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Aufgabe 3 Impl Geschlossene Hashmap *[Punkte: 16]*

In der Vorlesung wurde besprochen wie geschlossenes Hashing funktioniert. In dem im ILIAS bereitgestellten Projekt existiert bereits das Interface `IHashMap<K, V>`, welches die u.g. grundlegenden Funktionen auf einer Hashmap definiert. Zusätzlich existiert in diesem Projekt bereits die abstrakte Basisklasse `AbstractHashMap<K, V>`, welche das Interface bereits implementiert und einige möglicherweise nützliche Funktionen bereitstellt. Implementieren Sie in der ebenfalls bereits existierenden Klasse `ClosedHashMap<K, V>` die noch nicht implementierten Funktionen. Beachten Sie, dass bereits eine Implementierung der Funktion `remove()` gegeben ist. Die `ClosedHashMap` verwendet das Divisionverfahren mit linearem Sondieren. Dem bereits existierenden Konstruktor wird ein Sondierwert (`skipping`) übergeben. Implementieren Sie die Klasse `ClosedHashMap` mit folgenden Funktionen:

- `put(K key, V value)` fügt einen Wert und einen Schlüssel in die Hashmap ein. Falls bereits ein Wert mit demselben Schlüssel vorhanden ist, geben Sie diesen alten Wert zurück und ersetzen Sie diesen durch den neuen. Sind bereits `getRehashThreshold()` viele Elemente in der Datenstruktur vorhanden, muss diese vergrößert und ein Rehashing durchgeführt werden. Rufen Sie hierzu die vorgegebene Funktion `enlargeAndRehash()` auf.
- `contains(K key)` überprüft, ob der gegebene Schlüssel in der Hashmap vorhanden ist. Geben Sie `true` zurück falls dies der Fall ist; andernfalls geben Sie `false` zurück.
- `retrieve(K key)` gibt den Wert zum übergebenen Schlüssel zurück. Falls dieser Schlüssel nicht in der Hashmap existiert, geben Sie `null` zurück.
- `keyValuePairIterator()` gibt einen Iterator über die Schlüssel-Werte Paare zurück. Dieser traversiert das `entries`-Array in aufsteigender Reihenfolge (Start bei Index 0). Die `remove()`-Funktion braucht nicht implementiert werden (wirft eine `UnsupportedOperationException`).
- `iterator()` gibt einen Iterator über die Werte zurück. Dieser traversiert das `entries`-Array in aufsteigender Reihenfolge (Start bei Index 0). Sie können den `keyValuePairIterator` intern hierfür verwenden. Die `remove()`-Funktion braucht nicht implementiert werden (wirft eine `UnsupportedOperationException`).

Hinweis: Sie dürfen keine zusätzlichen Bibliotheken verwenden.

Aufgabe 4 (Bonus) Hashfunktionen *[Punkte: 6]*

Im Folgenden sind drei mögliche Hashfunktionen für diverse Mengen gegeben. Geben Sie jeweils an, ob die Kriterien der Surjektivität und der Gleichverteiltheit erfüllt sind. Begründen Sie Ihre Antwort.

(a) *(2 Punkte)* Menge: eine große repräsentative Menge englischer Wörter
Hashfunktion: UTF-8 Wert (0 - 127) des ersten Buchstabens

(b) *(2 Punkte)* Menge: alle Java-Befehle mit bis zu 6 Buchstaben (inklusive Operatoren)
Hashfunktion: Wortlänge (1 - 6)

(c) *(2 Punkte)* Menge: Die Studenten einer DSA Vorlesung
Hashfunktion: Die erreichte Gesamtpunktzahl über alle Übungsblätter

Aufgabe 5 (Bonus) Wissenschaftliche Recherche zu Hashfunktionen *[Punkte: 4]*

Im Artikel *Hashing Functions*, der 1975 in der Fachzeitschrift *The Computer Journal* publiziert wurde, werden verschiedene Hashfunktionen vorgestellt und eine chronologische Übersicht über alternative Speicher- und Information-Retrieval-Verfahren gegeben.

Hinweis: Viele wissenschaftliche Publikationen sind innerhalb des Universitätsnetzes kostenlos verfügbar. Suchen Sie diese beispielsweise mit Google Scholar[1].

(a) *(1 Punkt)* Suchen Sie die Veröffentlichung aus dem Jahr 1975. Nennen Sie die Namen aller Autoren dieser Veröffentlichung.

(b) *(3 Punkte)* Nennen Sie drei Hashfunktionen und drei alternative Speicher- und Information-Retrieval-Verfahren, mit denen sich der Artikel befasst.

Aufgabe 6 (Bonus) Hashfunktionen *[Punkte: 2]*

(a) *(1 Punkt)* Welche der folgenden Eigenschaften braucht eine Hashfunktion *nicht* zu erfüllen?

- ◯ Surjektivität
- ◯ Kollisionsfreiheit
- ◯ Gleichverteiltheit
- ◯ Berechenbarkeit in konstanter Zeit

(b) *(1 Punkt)* Beim geschlossenen Hashing wird mit Hilfe von Quadratischen Sondieren folgendes erreicht:

- ◯ Vermehrt Sekundärkollisionen und damit Clusterbildung
- ◯ Reduziert Sekundärkollisionen und damit Clusterbildung
- ◯ Reduziert Primärkollisionen und damit Clusterbildung
- ◯ Vermehrt Primärkollisionen und damit Clusterbildung

[1] `http://scholar.google.com`