Universität Konstanz
Fachbereich Informatik & Informationswissenschaft
Prof. Dr. Ulrik Brandes / Dr. Andreas Karrenbauer / Dr. Jürgen Lerner / Martin Mader

Algorithmen und Datenstrukturen
WS 2010/2011

9. Übungsblatt

Ausgabe: 15. Dezember 2010 **Abgabe:** 22. Dezember 2010, 10 Uhr
Die Bearbeitung in Zweiergruppen ist ausdrücklich erwünscht.

Aufgabe 1: Bloomfilter — 10 Punkte

Bloomfilter (nach ihrem Erfinder, Burton H. Bloom benannt) sind mit Hashing eng verwandte probabilistische Datenstrukturen. Mit sehr wenig Speicherplatz kann man Objektmengen repräsentieren und anfragen, ob ein Objekt enthalten ist; allerdings kann es vorkommen, dass ein nicht enthaltenes Objekt fälschlicherweise als enthalten gemeldet wird.

Ein Bloomfilter besteht aus m Bits, die zu Beginn 0 sind, und k verschiedenen Hashfunktionen mit Wertebereich $\{0, \ldots, m-1\}$. Ein Objekt wird hinzugefügt, indem seine k Hashwerte berechnet und die entsprechenden Bits auf 1 gesetzt werden. Wenn man wissen will, ob ein Objekt in der Menge enthalten ist, berechnet man seine k Hashwerte; falls mindestens eines dieser Bits im Filter 0 ist, kann das Objekt nicht enthalten sein; falls alle 1 sind, ist es mit großer Wahrscheinlichkeit enthalten, mit kleiner Wahrscheinlichkeit nicht.

Nehmen Sie an, dass jede der k Hashfunktionen jede der m Positionen mit gleicher Wahrscheinlichkeit wählt.

(a) Geben Sie die Wahrscheinlichkeit an, dass nach Einfügen eines Objekts in einen leeren Bloomfilter ein bestimmtes Bit 0 ist.

(b) Geben Sie die Wahrscheinlichkeit an, dass nach Einfügen von n Objekten in einen leeren Bloomfilter ein bestimmtes Bit 0 ist.

(c) Es werde ein nicht in der Menge enthaltenes Objekt angefragt, indem die k Hashwerte berechnet und die entsprechenden Positionen im Bloomfilter betrachtet werden. Geben Sie die Wahrscheinlichkeit an, dass an all diesen Positionen 1 steht, obwohl das Objekt nicht enthalten ist (*false positive*).

Nehmen Sie hierbei statistische Unabhängigkeit zwischen den verschiedenen Bits an.

(d) Skizzieren Sie wie ein Bloomfilter sinnvollerweise zusammen mit anderen Datenstrukturen verwendet werden kann um Anfragen *korrekt* (d. h., ohne false positives) zu beantworten. Was gewinnt man überhaupt durch die Verwendung des Bloomfilters?

Aufgabe 2: Hashtabelle über Verkettung  10 Punkte

(a) Leiten Sie eine Klasse `u09.name.StringTable` von der Klasse `material.u09.AbstractStringTable` ab. Diese soll eine Hashtabelle als Array von einfach verketteten Listen implementieren. (8 Punkte)

- Sie können die Listen realisieren, indem Sie von `HashEntry` ableiten, und die abgeleitete Klasse um die notwendige Zeigerstruktur erweitern. Das Element im Array repräsentiert dann den Kopf der Liste.
- Entwerfen Sie eine geeignete Hashfunktion `hashValue(k)`. Sie können darin die Methode `String.hashCode()` verwenden.
- Wenn bei `insert(k,e)` der Schlüssel k bereits in der Hashtabelle vorhanden ist, soll das entsprechende Element überschrieben werden. In diesem Fall soll das überschriebene Element zurückgegeben werden, sonst `null`.
- Die Methoden `find(k)` und `delete(k)` sollen `null` zurückgeben, falls kein Eintrag mit Schlüssel k vorhanden ist.
- Stellen Sie einen Konstruktor `StringTable(int capacity)` zur Verfügung, durch den die initiale Kapazität der Hashtabelle bestimmt wird.
- Überschreiben Sie die `toString()` Methode in `StringTable`, so dass die Listenstruktur sichtbar ist. Eine Ausgabe könnte z.B. so aussehen:

$$[() \mid (b,9) \mid (a,1) \rightarrow (ab,15) \mid ()]$$

(b) Verwalten Sie das Array dynamisch. Falls der Belegungsfaktor *nach* einer Einfügeoperation den Wert 0.75 ($\beta > 0.75$), soll die Kapazität verdoppelt und die Hashtabelle dementsprechend reorganisiert werden. Implementieren Sie dazu die Methode `rehash()`. (2 Punkte)