Karlsruher Institut für Technologie

Maximilian Katzmann
Hans-Peter Lehmann
Stefan Walzer

Übungsblatt 12

Randomisierte Algorithmik – Wintersemester 2023/2024

Aufgaben zur Vorlesung vom 25.1.2024
Abgabe im ILIAS bis 1.2.2024, 11:30 Uhr
Besprechung am 1.2.2024, 11:30 Uhr

Achte insbesondere bei handschriftlichen Abgaben auf Lesbarkeit. Die Abgabe erfolgt über das Übungsmodul im ILIAS. Gib Deine Ausarbeitungen in *einer* PDF-Datei ab.

Folgende Aufgabe hat nicht wirklich mit randomisierten Algorithmen zu tun, außer dass wir die Abschätzung in der Vorlesung gebraucht haben. Daher ist sie optional.

Aufgabe 1 – Bonus: Approximationen von e

Du weißt sicher, dass $e = \lim_{n\to\infty}(1+\frac{1}{n})^n$ gilt (das ist sogar eine gängige *Definition* von e). Leite daraus ab, dass die folgenden beiden Gleichungen für alle $n \in \mathbb{N}$ gelten:

$$(1+\tfrac{1}{n})^n \leq e \leq (1+\tfrac{1}{n})^{n+1}$$
$$(1-\tfrac{1}{n})^n \leq e^{-1} \leq (1-\tfrac{1}{n})^{n-1}.$$

Folgende Schritte bieten sich an.

(i) Zeige dass für alle $x \in \mathbb{R}$ gilt $1+x \leq e^x$.
Hinweis: Betrachte die Ableitungen bei $x = 0$.

(ii) Folgere aus (i) die linken beiden Ungleichungen.

(iii) Zeige, dass die rechten Seiten monoton in n fallen.

(iv) Folgere aus (iii) und einer Grenzwertbetrachtung die rechten beiden Ungleichungen.

Folgende Aufgabe ist diese Woche als einzige abzugeben.

Aufgabe 2 – Counting Bloomfilter (a.k.a.: Count-Min-Sketch)

Ein Counting Bloomfilter ist zunächst wie ein Bloomfilter: Er verwaltet n Schlüssel in einem Array der Größe m, der *load factor* ist $\alpha := \frac{n}{m}$ und es gibt k Hashfunktionen. Die Parameter seien wie in der Vorlesung gewählt, sodass ein (gewöhnlicher) Bloomfilter eine Falsch-Positiv-Wahrscheinlichkeit von ε hätte. Das Array enthält nun keine Bits mehr sondern natürliche Zahlen. Neben insert und query gibt es nun als weitere Operation delete.

Algorithm insert(x):
  for $i \in [k]$ **do**
    $A[h_i(x)]{+}{+}$

Algorithm delete(x):
  for $i \in [k]$ **do**
    $A[h_i(x)]{-}{-}$

Algorithm query(x):
  for $i \in [k]$ **do**
    if $A[h_i(x)] = 0$ **then**
      return false
  return true

Wir lassen zu, dass ein Schlüssel *mehrfach* in den Counting Bloomfilter eingefügt wird. Insofern ist das verwaltete Objekt S nun eine *Multimenge*, deren n Elemente $\{x_1, \ldots, x_n\}$ jeweils mit einer Vielfachheit $a_1, \ldots, a_n$ vorliegen. Die Operation insert(x) soll der Multimenge S eine Kopie von x hinzufügen, das heißt die Vielfachheit von x erhöhen, falls x bereits in S enthalten war oder ein weiteres Element mit Vielfachheit 1 in S aufnehmen, falls x noch nicht in S enthalten war. Die Bedeutung von delete ist analog.

Wie zuvor auch darf query falsch-positive, aber keine falsch-negativen Antworten liefern.

(a) Wir fordern, dass delete nur für solche Elemente aufgerufen wird, die auch wirklich in S enthalten sind (mit Vielfachheit mindestens 1). Was geht kaputt wenn wir das nicht fordern?

Mit Counting Bloomfiltern lassen sich noch zusätzliche nützliche Operationen implementieren.

(b) Implementiere eine Operation count, die für $x \in D$ einen Schätzwert count(x) für die Vielfachheit a von x in S angibt. Zeige, dass $\Pr[a \neq \text{count}(x)] \leq \varepsilon$ gilt.

(c) Das wichtigste Argument dafür (Counting-) Bloomfilter zu verwenden, ist der geringe Speicherplatz im Vergleich zu einer exakten Datenstruktur. Wir sollten daher besser nicht große Integer Datentypen (etwa 64 Bit) für die Zähler verwenden. Stellen wir uns einen Anwendungsfall vor, in dem die „meisten“ Zähler niemals 8 Bit überschreiten. Wir nutzen daher ein Array A von 8-Bit Zählern. Diskutiere (kurz) die folgenden Vorschläge zum Umgang mit Zählerüberläufen. Was sind die Vorteile und welche Kompromisse werden eingegangen?

- Alice verhindert lediglich Zählerüberläufe, passt also die $A[h_i(x)]{+}{+}$ Operation in insert und die $A[h_i(x)]{-}{-}$ Operation in delete so an, dass der Zähler nur inkrementiert bzw. dekrementiert wird, wenn der maximale darstellbare Wert bzw. der minimale darstellbare Wert noch nicht erreicht ist.
- Bob schlägt vor, einen Zähler, der den Wert $(11111111)_2 = 255$ erreicht hat „einzufrieren“, das heißt zukünftige insert oder delete Operationen werden den Zähler nie mehr anpassen.
- Carol schlägt vor, die Bitfolge $(11111111)_2 = 255$ als Markierung dafür zu verwenden, dass der wahre Zählerwert Größer als 254 ist. Der wahre Zählerwert wird in diesem Sonderfall in einer Hashtabelle gespeichert.