Inhaltsverzeichnis

Untere Schranken für deterministische Online Algorithmen

Wir werden nun eine untere Schranke für deterministische Online Algorithmen für das Listen-Zugriff-Problem herleiten. Wir werden uns hier auf das statische Modell beschränken. Da das statische Modell ein Spezialfall des dynamischen Modells ist, gilt diese Schranke natürlich auch für den dynamischen Fall. Wir werden folgenden Satz beweisen:

Satz. *Jeder* deterministische *Online Algorithmus für das statische Listen-Zugriff-Problem hat ein competitive ratio von mindestens* $2 - \frac{2}{\ell+1}$*, wobei* ℓ *die Anzahl der Datensätze in der Liste bezeichnet.*

Beweis: Sei eine Anfangsliste der Länge ℓ gegeben. Wir werden nun eine sogenannten *grausamen* Gegenspieler einsetzen. Dieser Gegenspieler fragt immer den Datensatz an, der am Ende der Liste steht. Er maximiert also die Kosten des Online Algorithmus. Nehmen wir einmal an, dass die Länge der Anfragesequenz n ist. Dann sind die Gesamtkosten des Online Algorithmus natürlich ℓn.
Wie können wir nun zeigen, dass es einen Offline Algorithmus gibt, der diese Anfragesequenz besser bearbeiten kann. Dazu werden wir uns eine Menge von Offline Algorithmen definieren und berechnen, wieviel Zeit diese im *Durchschnitt* brauchen, um die Anfragesequenz zu beantworten. Danach benutzen wir, dass es mindestens einen Offline Algorithmus geben muss, der genauso gut ist wie der Durchschnitt.
Wir betrachten $\ell!$ verschiedene Offline Algorithmen. Jeder dieser Algorithmen entspricht einer der $\ell!$ Permutationen der Anfangsordnung unserer Liste. Genauer gesagt, ordnet ein solcher Offline Algorithmus zu Beginn die Liste entsprechend dieser Permutation an und verändert die Ordnung der Liste danach nicht mehr. Das heisst, die Kosten für einen solchen Offline Algorithmus bestehen aus den Kosten für diese Umordnung plus den Kosten für das beantworten der Anfragen. Die Kosten für die Umordnung sind $O(\ell^2)$ (vgl. Bubblesort) also eine Konstante, da sie nicht von n abhängt.
Nun berechnen wir die durchschnittlichen Zugriffskosten eines dieser $\ell!$ Offline Algorithmen. Dazu müssen wir die Gesamtkosten aller Algorithmen berechnen und durch $\ell!$ teilen. Um die Gesamtkosten zu berechnen, machen wir folgende Beobachtung. Wenn wir einen Zugriff auf einen Datensatz x haben, dann kann x an jeder beliebigen Position der Liste stehen. Wenn wir nun festhalten, dass x an Position i steht, dann gibt es für die übrigen Position noch $(\ell-1)!$ Möglichkeiten und die Kosten für die Anfrage von x sind i. Daher gilt, dass die Gesamtkosten für eine Anfrage von x genau

$$\sum_{i=1}^{\ell} i \cdot (\ell-1)! = (\ell-1)! \cdot \frac{\ell(\ell+1)}{2}$$

sind. Damit sind die Gesamtkosten für alle n Anfragen

$$n(\ell-1)! \cdot \frac{\ell(\ell+1)}{2} + \ell! \cdot O(\ell^2) \ .$$

Es ergeben sich also Durchschnittkosten von

$$\frac{1}{2} n(\ell+1) + O(\ell^2) \ .$$

Daher gibt es auch einen Offline Algorithmus, der höchstens diese Kosten hat. Für $n \to \infty$ strebt das Verhältnis zwischen Online Kosten und Offline Kosten gegen $\frac{2\ell}{\ell+1}$.

Literatur