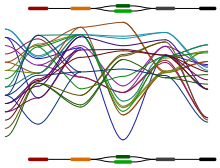
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK



Übungszettel 6

Hashing

Abgabe bis Donnerstag 16.12.2011 und Montag 19.12.2011 in den Tutorien

Übungszettel zur Vorlesung *Algorithmendesign* im Wintersemester 2011

Till Tantau, Michael Elberfeld, Kristina Fell, Stephanie Freitag, Malte Schmitz

Ziele dieses Blattes

1. Hashing mit linearer Sondierung und Kuckuck-Hashing verstehen, anwenden und implementieren können.
2. Dynamisches Hashing analysieren können

Tutoriumsaufgaben (unbenotet, nicht abzugeben)

Aufgabe 6.1 Hashing mit dynamischer Größenanpassung, mittel, Einzelarbeit

Im Folgenden soll eine Hashtabelle mit linearer Sondierung, dynamischer Größenanpassung und einer universellen Familie $H_p = \{h_{a,b} \mid a,b \in \{1,\dots,p-1\}\}$ mit $h_{a,b}(x) = ((ax+b) \bmod p) \bmod r$ von Hashfunktionen simuliert werden. Das dynamische Verhalten der Tabelle ist durch die Parameter $\alpha_{\min} = 1/8$, $\alpha_{\text{ideal}} = 1/4$, und $\alpha_{\max} = 1/2$ beschrieben.

1. Starten Sie mit einer Tabelle der Größe $r = 4$ und fügen Sie nacheinander Elemente mit den Schlüsseln 3, 5, 1 und 2 ein. Verwenden Sie hierzu die Hash-Funktion $h_{10,16} \in H_{31}$. Hierbei wird eine Größenanpassung notwendig, verwenden Sie nach der Größenanpassung die Funktion $h_{7,8} \in H_{31}$.
2. Löschen Sie aus der Tabelle nun die Werte 2 und 5. Geben Sie dabei explizit die Berechnung der Hashwerte und den gesamten Inhalt der Tabelle nach jeder Löschoperation an.

Aufgabe 6.2 Hashing mit dynamischer Größenanpassung allgemein analysieren, mittel, Einzelarbeit und Gruppenarbeit

In der Vorlesung wurde der folgende Satz für $\alpha_{\min} = 1/8$, $\alpha_{\text{ideal}} = 1/4$ und $\alpha_{\max} = 1/2$ mittels einer amortisierten Analyse bewiesen:

▶ Satz

Für eine Hash-Tabelle ohne Größenanpassung mögen die Basis-Operationen je Kosten 1 *verursachen.*
Dann kann für die Hash-Tabelle mit Größenanpassung mit $\alpha_{\min} < \alpha_{ideal} < \alpha_{\max}$ *eine Folge von* m *Einfüge- und Löschen-Operationen höchstens Kosten* $O(m)$ *verursachen.*

1. Führen Sie den Beweis aus der Vorlesung, aber verwenden Sie dieses mal die Potentialfunktion $\Phi(T_i) = 12\,|n_i - r_i/4|$.
2. Ändern Sie Ihren Beweis so ab, dass er den Satz für beliebige $\alpha_{\min}$, α_{ideal} und $\alpha_{\max}$ zeigt. Ersetzen Sie dazu den Wert 12 in der Potentialfunktion durch einen geeigneten Wert, der nur von $\alpha_{\min}$, α_{ideal} und $\alpha_{\max}$ abhängt und verallgemeinern Sie Ihre Analyse.

Aufgabe 6.3 Kuckuck-Hashing simulieren, mittel, Einzelarbeit

In dieser Aufgabe werden wir das Kuckuck-Hashing an einem Beispiel simulieren. Hierbei verwenden wir die Hashfunktion $h_{11,14} \in H_{17}$ für die erste der beiden Tabellen und die Hashfunktion $h_{1,10} \in H_{17}$ für die zweite. Eine dynamische Größenanpassung soll in dieser Aufgabe erst einmal nicht vorgenommen werden.

Starten Sie mit zwei leeren Tabellen der Größe $r = 3$ für das Kuckuck-Hashing. Fügen Sie nacheinander die Elemente 3, 5, 2, 7 und 10 ein.

Hausaufgaben (benotet, abzugeben)

Aufgabe 6.4 Hashing-Verfahren implementieren, mittel bis schwer, 16 Punkte

In dieser Aufgabe soll das Hashing mit linearer Sondierung und Kuckucks-Hashing jeweils mit dynamischer Größenanpassung in Java implementiert werden. Gehen Sie dabei wie folgt vor:

1. Erstellen Sie zwei Klassen, die jeweils das Interface

```
public interface Hashfunction {
  int hash(int key);
}
```

   implementieren. Eine Klasse `SimpleHash` soll zur Berechnung der einfachen Hashfunktion $h(x) = x \bmod r$ verwendet werden können, die andere Klasse `UniversalHash` soll Hashfunktionen der Familie H_p repräsentieren.
2. Implementieren Sie Hashing mit linearer Sondierung zunächst ohne dynamische Größenanpassung. Kapseln Sie die Implementierung in eine Klasse, die das Interface

```
public interface HashMap {
  void insert(int key);
  void delete(int key);
  boolean search (int key);
}
```

   implementiert. Objekte dieser Klassen repräsentieren Hashtabellen, die auf linearer Sondierung basieren. Erweitern Sie Ihre Klasse um eine Methode `rehash`, die beim Einfügen und Löschen aufgerufen wird, um die Größe der Tabelle anzupassen. Sie können annehmen, dass $\alpha_{\text{min}} = 1/8$, $\alpha_{\text{ideal}} = 1/4$ und $\alpha_{\text{max}} = 1/2$ gilt.
3. Implementieren Sie Kuckuck-Hashing zunächst ohne dynamische Größenanpassung in einer Klasse, die das Interface `HashMap` implementiert. Nehmen Sie an, dass $\varepsilon = 1$ gilt. Erweitern Sie das Kuckuck-Hashing um eine geeignete `rehash`-Methode. Sie können zur Vereinfachung $\alpha_{\text{min}} = 1/16$, $\alpha_{\text{ideal}} = 1/8$ und $\alpha_{\text{max}} 1/4$ annehmen.
4. Testen Sie Ihr Programm für verschiedene Beispieleingaben. Fügen Sie Ihrer Abgabe zwei dieser Eingaben bei und erläutern Sie ausführlich, wie man das Programm kompilieren, starten und testen kann. Kommentieren Sie das Programm umfassend.