Prof. Dr. Gerd Stumme, Folke Eisterlehner, Dominik Benz
Fachgebiet Wissensverarbeitung

07.07./09.07.2009

12. Präsenzübung „Algorithmen und Datenstrukturen“

Sommersemester 2009

Hash-Verfahren

Aufgabe 1 (Duplikaterkennung)

Der folgende Algorithmus überprüft mit Hilfe von Hashing (mit Verkettung), ob ein gegebenes `int`-Array `a` duplikatfrei ist. Bestimmen Sie seine Worst-Case-Laufzeit in O-Notation für den Fall, dass `a` duplikatfrei ist.

```
for (int i: a) {
  if (hashtable.contains(i) { print("Nicht duplikatfrei"); }
  else { hashtable.insert(i); }
}
print("duplikatfrei");
```

Aufgabe 2 (Dynamische Hashtabelle)

a) Wir betrachten eine Hashtabelle, welche die Operationen Suchen und Einfügen unterstützt. Zur Kollisionsbehandlung wird Verkettung mit unsortierten Listen verwendet. Wie üblich bezeichnet m die (aktuelle) Größe der Hashtabelle und n die Anzahl der (aktuell) gespeicherten Datensätze. Die Schlüssel sind gleichverteilte natürliche Zahlen i, als Hashfunktion wird $h(i) = i \mod m$ verwendet.

Unsere Hashtabelle hat die Besonderheit, dass immer wenn n bei einer Einfüge-Operation den Wert m erreicht, m verdoppelt wird. In diesem Fall legt man zunächst eine neue Hashtabelle der Größe m (neuer Wert) an und überträgt dann alle Elemente der alten in die neue Hashtabelle. Dabei bestimmt man die neue Position mit Hilfe der neuen Hashfunktion (also der mit dem neuen m). Erst danach wird die eigentliche Einfüge-Operation ausgeführt.

Welcher der Datentypen

- `D[][]`,
- `ArrayList<D>[]`,
- `LinkedList<D>[]`,
- `ArrayList<D[]>`,
- `ArrayList<ArrayList<D>>`,
- `ArrayList<LinkedList<D>>` und

- `LinkedList<ArrayList<D>>`

erscheint Ihnen zur Implementierung der Datenstruktur aus Aufgabe a) am besten geeignet? Begründen Sie und geben Sie dabei für die übrigen Datentypen an, warum sie nicht in Frage kommen oder weniger gut geeignet sind. (Der Begriff „unsortierte Liste“ aus Aufgabe a) soll dabei **`ArrayList`** und **`Array`** nicht ausschließen.) Wenn nötig, können Sie Ihre Antwort von weiteren Bedingungen abhängig machen.

b) Warum ist die Verdopplungsstrategie in diesem Kontext ungünstig? Welche (bessere) Alternative gibt es?

Aufgabe 3 (Anwendungsfälle)

Betrachten Sie die im Folgenden beschriebenen Anwendungsszenarien und tragen Sie in nachstehender Tabelle eine, welche der Datentypen

i) Stack

ii) Warteschlange

iii) balancierter Suchbaum

iv) Hashtabelle

v) Prioritätswarteschlange

für das jeweilige Szenario gut geeignet (**`+`**), welche nicht geeignet (**`-`**) und welche weder besonders gut, noch besonders schlecht geeignet (**`0`**) sind.

1.) Es sollen die Titel aller mp3-Dateien auf der Festplatte eines durchschnittlichen Studenten verwaltet werden. Dies beinhaltet:
   - Einfügen neuer Titel
   - Suchen nach einem Titel
   - Sortierte Ausgabe aller Titel

2.) Bei einem Versandhaus sollen eingehende Bestellungen verwaltet werden.

3.) In einem Krankenhaus sollen die Patienten bei der Aufnahme in das Computersystem übertragen werden, um anschließend behandelnden Ärzten und anderen Resourcen zugeteilt zu werden.

4.) Die Lese-Zugriffe auf einen Suchbaum mit mehreren Milliarden Einträgen sollen noch weiter beschleunigt werden, indem in einem zusätzlichen Cache die letzten 1000 Such-Anfragen gespeichert werden. Dabei sei die Anzahl von Suchanfragen pro Schlüssel mit geringer Varianz normalverteilt.

	Stack	Warteschlange	Suchbaum	Hashtabelle	Prioritätswarteschlange
1.)					
2.)					
3.)					
4.)					

Aufgabe 4 (Arbeit mit Hashtabellen)

Gegeben sei die Schlüsselfolge $a = 50, 59, 68, 32, 23, 14, 41$. Fügen Sie diese sukzessive in

a) eine Hashtabelle t_1 der Größe $m_1 := 7$

b) eine Hashtabelle t_2 der Größe $m_2 := 9$

ein. Die Tabellen seien anfänglich leer, wobei ein leerer Eintrag durch „_“ dargestellt wird.

Verwenden Sie $h_i(x) := x \bmod m_i$ als *Hashfunktion* ($i = 1, 2$) und *quadratisches Sondieren* zur *Kollisionsbehandlung*: Kommt es beim Einfügen eines Elements k in t_i zu einer Kollision (d.h. $t_i[h_i(k)] \neq \_$), so wird fortlaufend

$$
\begin{gathered}
(h_i(k) + 1) \bmod m_i, \\
(h_i(k) - 1) \bmod m_i, \\
(h_i(k) + 4) \bmod m_i, \\
(h_i(k) - 4) \bmod m_i, \\
(h_i(k) + 9) \bmod m_i, \\
(h_i(k) - 9) \bmod m_i, \\
\ldots
\end{gathered}
$$

als neuer Hashwert berechnet, bis ein freier Platz gefunden wird.

Geben Sie den Inhalt der Hashtabelle nach der jeweils letzten Einfügeoperation an und notieren Sie in einer Tabelle zu jedem Schlüssel die Anzahl der jeweils benötigten Sondierungsschritte.

Der Inhalt einer Hashtabelle t soll dabei wie folgt angegeben werden:

```
0: -
1: a3
2: -
3: -
4: a2
```

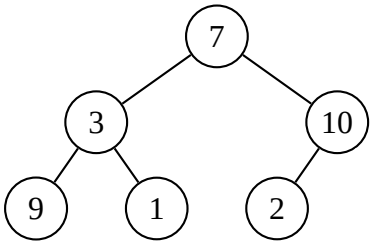
(t enthält in Postion 1 den Wert a_3, an Postion 4 den Wert a_2 und ist ansonsten leer)

Aufgabe 5 (Anwendungen von Hashfunktionen)

Welche weiteren Anwendungen von Hashfunktionen fallen Ihnen ein?

Aufgabe 6 (Anwendungen von Hashfunktionen)

Betrachten Sie folgenden Binärbaum:



- Stellen Sie die hierarchische Struktur des Baumes als Array dar, wie auf den Folien 6-81 dargestellt.
- Wandeln Sie den Baum in einen Heap um. Skizzieren Sie das Ergebnis jedes Zwischenschritts sowohl in Baum- als auch in Array-Notation (siehe Darstellung auf den Folien 6-84/85).