Algorithmen und Datenbanken

Protokoll: Übung 1 Hashtabelle

Teammitglieder

Lakheshar Philipp if18b048

Streith Roman if18b144

Verwaltung der Kursdaten

Das Objekt „HashTable“

Dieses Objekt repräsentiert, wie es schon der Name verrät, den gesamten Hashtable in welchem die einzelnen Objekte „Stock“ eingefügt werden. Das Objekt „Stock“ hat folgende Struktur:

`private String name;`	Hier wird der Name der Aktie gespeichert
`private String[] latestHistory`	Diese Array speichert die Kursdaten der letzten 30 Tage
`private String wkn;`	Hier wird die Wertpapierkennnummer gespeichert
`private String symbol;`	IN diesem String wird das jeweilige Kürzel der Aktien festgehalten

Beim Initialisieren einer neuen Aktie werden jedoch nur die Variablen „name“, „wkn“ und „smybol“ befüllt. Die Array „latesHistory“ ist mit einem initial wert „null“ definiert. Die Befüllung der Historie findet erste statt, wenn ein Benutzer im Hauptmenü die Auswahl „Import“ trifft und das Kürzel einer geeigneten, im Hashtable bereits verfügbare, Aktie eingibt. Diese Daten werden aus einer CSV-Datei nach Aktualität gefiltert und in die Variable „latestHistory“ eingelesen. Sollten weniger als 30 Tage an Daten in der zur Verfügung gestellten CSV-Datei vorhanden sein wird der gesamte Datensatz der Datei eingelesen und die Größe der benötigten Array angepasst.

Das Objekt „SymbolHashTable“

Jedes Mal, wenn eine neue Aktie eingelesen oder gelöscht wird simultan eine weitere Hashtabelle „SymbolHashTable“ mit einem Objekt „SymbolReference“, mit folgenden Daten befüllt:

`private String symbol;`	Speichert das Kürzel einer Aktie
`private int indexInTable;`	Repräsentiert den Index der Aktien mit oben gespeichertem Kürzel im „HashTable“

Diese zweite Hashtabelle ist nötig um die Suche und das Löschen einer Aktie auch mittels des Kürzels der Aktie zu ermöglichen. Auch diese Hashtabelle bedient sich der Hashfunktion als auch der Kollisionsbehandlung.

Die Hashfunktion

Die Hashfunktion welche jedem neuen Eintrag einen Index im „Hashtable“ zuordnet ist wie folgt aufgebaut:

- Es wird eine Variable namens "index" mit dem wert "0" initialisiert.
- Danach wird mit Hilfe einer Schleife jeder Buschstabe des Aktiennamens, in folgender Form ein Wert zugewiesen
    - Zuerst wird er ASCII-Wert des aktuellen Buchstaben ermittelt.
    - Dieser ASCII-Wert wird nun mit der Zahle 53 hoch dem aktuellen Index des Buchstaben genommen. (wie in der Informatik gewohnt starten wir bei 0).
    - Um im Bereich eines „integers“ zu bleiben wird das Ergebnis bei jedem einzelnen Durchgang Modulo 2003 (Größe unseres Hashtables) genommen.

      (Hier war es wichtig dies bei jedem einzelne Buchstaben zu tun da es sonst leicht passieren kann, dass der Datentyp „int“ an seine Grenzen stößt und nicht mehr anwachsen kann. Somit werden Aktien, wenn der Modulo erst am Schluss der Hashfunktion angewandt wird den gleichen Index zugeordnet und immer eine Kollision verursachen.)

- Die Summe aller Werte werden so zum Index des Hashtables.
- Daraus ergibt sich folgende Formel
    - $f(w_i)$ = ASCII-WERT(w_i)
    - $\sum_{i=0}^{n} f(w_i) \cdot 53^i$

Danach wird auf Kollision geprüft.

Kollisionserkennung

Nachdem der Hashwert ermittelt wurde wird geprüft ob der so erhaltenen Index im Hashtable auch tatsächlich frei ist. Sollte dies nicht der Fall sein wir mittels der Quadratischen Sondierung ein freier Index ermittelt.

Die Formel hierfür sieht wie folgt aus:

- „h(x)“: jener Index an welchem die Ursprüngliche Kollision stattgefunden hat.
- „i“: der aktuelle „Kollisionswert“
- „m“: Größe des Hashtables

$$h_i(x) = \left(h(x) + (-1)^{i+1} \cdot \left\lceil \frac{i}{2} \right\rceil^2 \right) \bmod m$$

Dieser Vorgang wird solange wiederholt (und „i“ immer um 1 erhöht) bis ein freier Index oder ein bereits bestehender identer Eintrag gefunden wurde.

Der Löschalgorithms

Um einen Eintrag löschen zu können muss dieser erst einmal gefunden werden. Gefunden wird dieser (wenn vorhanden) mit Hilfe der oben beschriebenen Hashfunktion und wenn nötig in Kombination mit der ebenso oben beschriebenen Kollisionserkennung.

Die Suche nach einer Aktie geht wie folgt von statten:

- Zuerst wird die Hashtabelle anhand des vom User eingegebenen Inputs mittels der Hashfunktion welche den dazugehörigen Index zurückliefet durchsucht.

- Sollte es sich an dieser Stelle nicht um den gesuchten Eintrag handeln wird die Kollisionserkennung durchgeführt.

- Sollte es einen Index in der Hashtabelle geben, egal ob von der Hashfunktion oder der Kollisionserkennung ermittelt, welcher kein Objekt Aktie beinhalten so muss der User einen Wert eingegeben haben zu welchem es keinen Eintrag in der Tabelle gibt und die Suche wird abgebrochen.

- Bei erfolgreicher Suche wird zuerst anhand des Kürzels der eigentlichen Aktie die Referenz im „SymbolHashTable“ gelöscht und anschließend die tatsächliche Aktie.

Aufwandsabschätzung

Vergleich der Hashtabelle mit einem einfachen Array nach O-Notation:

Datenstruktur	Array	Verkettete Liste	Hashtabelle
Aufwand Suchen	$O(\log^2 N)$ in einem sortierten Array (mit binary search), ansonsten O(N)	O(N)	O(1)
Aufwand Einfügen	O(N)	O(N)	O(1)
Aufwand Löschen (wenn bereits gefunden)	O(N)	O(1)	O(1)
Ausartung des Aufwands (worst-case)	-	-	O(N), wenn viele Kollisionen auftreten und die Hashtabelle sehr voll ist.
Speicherbedarf	Nur Daten	Daten + 1 Verweis	Nur Daten

Codespezifischer Aufwand in Anzahl d. Befehle/Anweisungen:

Hashfunktion:

```
private int hashFunction(String StockName) {
   int index = 0;
   for (int i = 0; i < StockName.length(); i++) {
      index += StockName.charAt(i) * Math.pow(53, i) % 2003;
   }
   return index % 2003;
}
```

2 Anweisungen + (Länge des Aktiennamens * 5)

Mit ca. 5 Anweisungen als Summe von: charAt(i), pow(53, i), „*“, „%“ und „+“. Für genauere Angaben könnten String.charAt() und vor allem Math.pow() auch berücksichtigt werden.

Sondierungsfunktion:

```
private int collisionHandling(int collisionindex, int index) {
   double retVal = (collisionindex + Math.pow((-1), index + 1) *
            (Math.ceil((index / 2.0)) * Math.ceil((index / 2.0)))) % 2003;
   return (int) retVal;
}
```

10 Anweisungen als Summe aus: *pow*(), 2 * ceil, „%“, 2 * „/“, 2 * „+“, 2 * „*“

Überschlagsmäßige Kollisionskosten:

K = f * (2 Anw. (Overhead) + 10 Anw. (Sondierungsfkn.-Kosten))

Faktor f = 2 wenn Sowohl Name als auch Symbol kollidieren, ansonsten 1.

Füllgrad bei n = 1k: α = 1000/2003 = 49,93%

d.f.: Anzahl der Durchschnittlichen Kollisionen bei n = 1k: $(1 - \alpha)^{-1} = 1{,}9972 = \sim 2$

Funktion	Best Case - Hash leer	Worst Case mit $n_{max} \neq 1k$ (1 Platz frei, 2002 Koll.)	Worst Case mit $n_{max} = 1k$ (~1000/2003 bes.)
Einfügen	$16 + (L_N + L_S) * 5$	$48064 + (L_N + L_S) * 5$	$40 + (L_N + L_S) * 5$
Suchen	$12 + L_N * 5$ (nach Name) $21 + (L_N + L_S) * 5$ (nach Symbol)	$24036 + L_N * 5$ (nach Name) $48069 + (L_N + L_S) * 5$ (nach Symbol)	$36 + L_N * 5$ (nach Name) $45 + (L_N + L_S) * 5$ (nach Symbol)
Löschen	$20 + (L_N + L_S) * 5$	$48068 + (L_N + L_S) * 5$	$44 + (L_N + L_S) * 5$

L_N … Länge des Aktiennamens L_S … Länge des Aktiensymbols