Westfälische Hochschule

Prof. Dr. Manfred Meyer
Fachbereich 6 / Bocholt
Professur BWL / Wirtschaftsinformatik

Algorithmen und Datenstrukturen
(Studiengang Wirtschaftsinformatik 3. Semester)

Aufgabenstellung für das Praktikum am 2.11.2023

Aufgabe 9:

Die Firma Schnell & Sicher, die Sie aus der Vorlesung kennen, beauftragt Sie damit, ein Java-Programm zu schreiben, mit dem alle Aufträge erfasst und verwaltet werden können. Modellieren sie dazu eine Klasse `Auftrag`, die mindestens das Attribut `auftragsnummer` enthält. Außerdem soll es möglich sein, einen Auftrag zu einer bestimmten Auftragsnummer zu suchen.
Realisieren Sie dies über eine *Lineare Suche* und programmieren Sie die Funktion:

```
public static int linsearch (Auftrag[] auftraege,
                             int auftragsnummer)
```

Diese Funktion soll im Erfolgsfall den Index des gesuchten Auftrags im Feld `auftraege` zurückliefern oder -1 falls kein Auftrag zu der angegebenen Auftragsnummer vorhanden ist.

Erstellen Sie nun im Hauptprogramm mehrere Aufträge, lassen Sie vom Benutzer eine Auftragsnummer eingeben und suchen Sie nach diesem Auftrag. Falls der Auftrag gefunden wird, geben Sie bitte seine Daten aus, andernfalls den Hinweis, dass kein Auftrag zu der angegebenen Auftragsnummer existiert.

Aufgabe 10:

Das Suchen mit Hilfe einer Linearen Suche hat sich ja als nicht sonderlich zweckmäßig erwiesen. Sie werden daher erneut von der Firma Schnell & Sicher damit beauftragt, ein verbessertes Java-Programm zu schreiben, mit dem man nach einer bestimmten Auftragsnummer suchen kann. Dies soll jetzt über eine *Binäre Suche* erfolgen.

Realisieren Sie daher die Funktion

```
public static int binsearch (Auftrag[] auftraege,
                             int auftragsnummer)
```

Auch hier soll die Funktion im Erfolgsfall wieder den Index des gesuchten Auftrags im Feld `auftraege` zurückliefern oder -1 falls kein Auftrag zu der angegebenen Auftragsnummer vorhanden ist.

Erstellen Sie nun wieder im Hauptprogramm mehrere Aufträge, lassen Sie vom Benutzer eine Auftragsnummer eingeben und suchen Sie nach diesem Auftrag. Falls der Auftrag gefunden wird, geben Sie bitte seine Daten aus, andernfalls den Hinweis, dass kein Auftrag zu der angegebenen Auftragsnummer existiert.

Westfälische Hochschule

Prof. Dr. Manfred Meyer
Fachbereich 6 / Bocholt
Professur BWL / Wirtschaftsinformatik

Kurzer Rückblick: Terminierung, Korrektheit und Komplexität

Bei der Untersuchung der Binären Suche haben wir als Voraussetzungen festgehalten, dass die Elemente paarweise vergleichbar und sortiert abgelegt sein müssen. Nehmen Sie nun einmal an, dass die Sortierung nicht vollständig gegeben ist, das Feld also nicht komplett sortiert ist.

a) Wie wirkt sich dieser Umstand auf die Terminierung des Verfahrens aus?
b) Wie wirkt sich dieser Umstand auf die Korrektheit des Verfahrens aus?
c) Wie wirkt sich dieser Umstand auf die Komplexität des Verfahrens aus?

Aufgabe 11:

Die Binäre Suche hat bei der Firma Schnell & Sicher zu einer deutlichen Verbesserung bei der Suche nach den Aufträgen geführt, noch besser ist aber – wie Sie ja bereits in der Vorlesung gehört haben – eine *Hashing-Basierte Suche*.

Implementieren Sie daher nun ein entsprechendes Suchverfahren mit einer Kollisionsbehandlung in Form der Offenen Adressierung.

Verwenden Sie dabei zur Speicherung der Aufträge ein Feld der Länge 100 als Hashtabelle und als Hashfunktion die letzten beiden Dezimalstellen der Auftragsnummer. Das heißt dann also, dass beispielsweise ein Auftrag mit der Auftragsnummer 456 an der Position 56 im Feld gespeichert wird. Definieren Sie dazu eine Funktion

```
public static int getHashIndex(Auftrag[] auftraege, int auftragsnummer)
```

zur Ermittlung der Position für einen Auftrag `auftrag` in der Hashtabelle (`auftraege`) durch Berechnung des Ausdrucks `auftragsnummer % auftraege.length`.

Eine Kollision, die es aufzulösen gilt, ergäbe sich dann zum Beispiel, wenn ein weiterer Auftrag mit der Nummer 756 gespeichert werden soll, da die Hashfunktion auch hier als Position 56 liefert. In diesem Fall soll nach einer Ausweichposition in der Hashtabelle gesucht werden. Verwenden Sie hierzu die *lineare Sondierung*, probieren Sie also einfach die nächste Position im Feld und das solange bis entweder eine freie Position gefunden wird oder Sie feststellen, dass die Hashtabelle bereits vollständig gefüllt ist.

Realisieren Sie dazu zunächst eine Funktion

```
public static boolean insertIntoHashTable(Auftrag[] auftraege,
                                          Auftrag auftrag)
```

zum Einfügen eines neuen Auftrags (`auftrag`) in eine Hashtabelle (`auftraege`) wie oben beschrieben. Falls das Einfügen geklappt hat, soll `true` zurückgegeben werden, andernfalls (falls die Tabelle voll ist) `false`.

Realisieren Sie nun eine Funktion

```
public static Auftrag getFromHashTable(Auftrag[] auftraege,
                                       int auftragsnummer)
```

zum Suchen eines Auftrags zu der angegebenen Auftragsnummer (`auftragsnummer`) in der Hashtabelle (`auftraege`). Falls zu der angegebenen Auftragsnummer ein Auftrag gefunden wird, so wird dieser zurückgegeben, andernfalls der Wert **`null`**.

Erstellen Sie nun im Hauptprogramm wiederum mehrere Aufträge und fügen Sie diese mit der Operation `insertIntoHashTable()` in die Hashtabelle ein.

Lassen Sie dann vom Benutzer eine Auftragsnummer eingeben und suchen Sie mit der Operation `getFromHashTable()` nach diesem Auftrag. Falls der Auftrag gefunden wird, geben Sie bitte seine Daten aus, andernfalls den Hinweis, dass kein Auftrag zu der angegebenen Auftragsnummer existiert.

Zusatzaufgabe:

Recherchieren Sie, welche der besprochenen Strategien zur Kollisionsbehandlung (offenes Hashing oder offene Adressierung) in Java bei der Implementierung der Ihnen bekannten Datenstruktur `HashSet<E>` zum Einsatz kommt.
Was können Sie zu Details der Implementierung herausfinden (Implementierung der „Überlauf-Struktur“ beim offenen Hashing bzw. Funktion zur Berechnung der Ausweichposition bei offener Adressierung)?

Challenge:

Eine Implementierung der Hashing-basierten Suche mittels offener Adressierung haben Sie in Aufgabe 9 realisiert. Da hier im Kollisionsfall linear nach einer Ausweichposition gesucht wird, bezeichnet man dies auch als sog. „linear probing“. Eine weitere Implementierung dieser Art enthält die Klasse `LinProbHashTable`, die Sie im letzten Leseauftrag kennengelernt haben und die Ihnen in Moodle als Download zur Verfügung steht.

a) Ergänzen Sie diese um eine Implementierung für die folgende Operation

   ```
   public ElementType<? extends Comparable>[] toArray (),
   ```

   die alle gespeicherten Elemente der Reihe nach (so wie in der internen Hashtabelle gespeichert) in einem neu erzeugten Feld ablegt und dieses zurückliefert. Die Länge des Feldes entspricht der Anzahl der aktuell gespeicherten Elemente.

b) Ergänzen Sie diese außerdem um eine Implementierung für die folgende Operation

   ```
   public boolean remove (ElementType<? extends Comparable> elem)
   ```

   zum Entfernen eines Elements `elem`, die genau dann **`true`** zurückliefert, wenn das Element `elem` enthalten war und jetzt gelöscht wurde. Testen Sie Ihre Implementierung mit einer `LinProbHashTable<String>`, in die Sie einige `String` -Objekte einfügen, löschen, wieder hinzufügen usw. und schließlich alle Elemente unter Verwendung der in Teil a) implementierten Operation ausgeben lassen.

c) Untersuchen Sie die Implementierung einer `LinProbHashTable<Integer>` im Hinblick auf unterschiedliche Ladefaktoren.

   Erzeugen Sie dazu eine Instanz mit der Kapazität 1000 und befüllen diese schrittweise zunächst mit 100, 200, 300 und in Hunderterschritten weiter bis 900 und schließlich noch 925, 950, 975, 980, 985, 990, 995, 997 und 999 zufällig erzeugten verschiedenen Integer-Objekten und messen für jeden dieser Füllstände getrennt die Zeit, die benötigt wird, um anschließend der Reihe nach über die Operation `get()` nach allen diesen Elementen zu suchen.

   Fassen Sie die Messergebnisse in einer Tabelle zusammen, also insgesamt 18 Messwerte für unterschiedliche Ladefaktoren von 10% bis 99%. Welche Abhängigkeit können Sie erkennen?

Viel Spaß und Erfolg bei der Bearbeitung der Aufgaben!