Technische Hochschule Rosenheim
Technical University of Applied Sciences

Übung 07: Hashtabellen, binäre Suchbäume

Aufgabe 1: Lineares Sondieren mit „Eager Delete“

Gegeben ist eine Hashtabelle der Größe $m = 11$. Zur Kollisionsauflösung wird ***lineares Sondieren*** verwendet. Die Hashtabelle ist zu Beginn leer.

a) Es werden der Reihe nach die 7 Schlüssel 10, 22, 31, 4, 15, 28, 59 eingefügt. Wie ist die Belegung nach dem Einfügen aller Schlüssel?

b) Nun löschen Sie Schlüssel 4 mit dem Verfahren der Vorlesung („*Eager Delete*“), siehe rechter Code.
Geben Sie die Arraybelegung von `keys` am Ende jeder Iteration der **zweiten** `while`-Schleife an.

```
int i = hash(key);
while (!key.equals(keys[i])) {
    i = (i + 1) % m;
}
keys[i] = null;
vals[i] = null;
i = (i + 1) % m;
while (keys[i] != null) {
    Key   keyToRehash = keys[i];
    Value valToRehash = vals[i];
    keys[i] = null;
    vals[i] = null;
    n--;
    put(keyToRehash, valToRehash);
    i = (i + 1) % m;
}
n--;
```

Aufgabe 2: Binäre Suchbäume

a) Gegeben sind die Schlüssel $\{1, 2, 3, 4, 5, 6, 7\}$. Zeichnen Sie sowohl einen ***beliebigen*** binären Suchbaum der Höhe 2 als auch der Höhe 6, der alle diese Schlüssel enthält.
Beachten Sie: Die Höhe ist die Anzahl der Kanten auf dem längsten Weg von der Wurzel zu einem Blatt.

b) In welcher Reihenfolge werden die Knoten des ***binären Suchbaums*** der Höhe 2 von ***a)*** ausgegeben, falls *Preorder* bzw. *Inorder* bzw. *Postorder* verwendet wird?

c) Welcher binäre Suchbaum ergibt sich, wenn man der Reihe nach die folgenden Schlüssel einfügt:
`E A S Y Q U T I O`
Hinweis: Der Buchstabe `E` ist beispielsweise größer ist als der Buchstabe `A`.

d) Löschen Sie die Schlüssel aus dem Ergebnis von a) in der Reihenfolge des Einfügens, also:
`E A S Y Q U T I O`
Zeichen Sie den binären Suchbaum nach ***jedem*** Löschvorgang. Falls der zu löschende Knoten 2 Kinder hat, soll **konsequent** **der *Nachfolger, nicht der Vorgänger,*** an die Stelle des zu löschenden Knotens gesetzt werden.

e) Bei der Preorder-Traversierung eines binären Suchbaums ergibt sich die folgende Besuchsreihenfolge: `20, 10, 5, 1, 7, 15, 30, 25, 35, 32, 40`
Zeichnen Sie den binären Suchbaum und erklären Sie Ihre Herangehensweise!

Aufgabe 3: Lineares Sondieren mit „*Lazy Delete*“

Anders als in Aufgabe 1) sollen die beim Löschen benötigten Maßnahmen auf später verschoben werden („*Lazy Delete*“).

a) Studieren Sie das gewünschte Verhalten durch Spielen mit der Animation:
https://www.cs.usfca.edu/~galles/visualization/ClosedHash.html
2, 60, 89 einfügen, dann 60 löschen, dann 3 einfügen

b) Modifizieren Sie die Java-Klasse `HashtableProbingLazyDelete.java`! Beim Löschen eines Key-Value-Paares wird der Value auf `null` gesetzt, der dazugehörige Key jedoch nicht gelöscht. Das dient zur Erkennung eines ungültigen Eintrags, was voraussetzt, dass kein gültiger Key mit

einem `null` als Value existieren darf. Erst spätere `resize()`-Operationen oder das Einfügen eines neuen Schüssels können solche ungültigen Schlüssel entfernen.

Hinweise / Tipps:

- Zählen Sie in einer Variable `numInvalid` mit, wie viele Einträge aktuell **ungültig** sind.
- Für das Einfügen gilt: Falls die Belegung mit ***gültigen*** Tabelleneinträgen zu Beginn des Einfügens $\geq 50\%$ ist, soll die Tabelle verdoppelt werden. Falls die Belegung mit ***gültigen* UND *ungültigen*** Tabelleneinträgen **zusammen** $\geq 50\%$ ist, soll ein „Rehashing“ durchgeführt werden, ohne die Tabellengröße zu verändern.
- Für das Löschen gilt: Belegen die ***gültigen*** Tabelleneinträge am Ende der Löschmethode $\leq$ 12,5%, soll die Tabelle halbiert werden.
- Der Code von `delete` nach dem „*Eager Delete*“-Verfahren ist auskommentiert, Sie können diesen nehmen und anpassen.
- Welche Funktionen müssen angepasst werden?
- Testen Sie mit JUnit!