Aufgaben zur Veranstaltung
Algorithmen und Datenstrukturen, SS 2020

H. Pflug, J. Dietel FH Aachen, Campus Jülich; IT Center, RWTH Aachen

Hausaufgaben 2 06./07.04.2020

Abgabe der Lösung am 13.04.2020

Aufgabe 1

Sehen Sie sich die Implementation einer Queue mit Hilfe einer LinkedList auf der Seite

https://www.geeksforgeeks.org/queue-linked-list-implementation/

an.

a) Der prinzipiell gut programmierte und kommentiere Code weist ein paar Schwächen auf. Verbessern Sie diese zunächst.
   - Trennen Sie die Testklasse von der eigentlichen Code-Klasse.
   - Korrigieren Sie die Zugangsmodifizierer (`public`, `private`).
   - Setzen Sie einzeilige `if`-Blöcke in geschweifte Klammern.
   - Nennen Sie Klasse `Queue` in `MyQueue` um, um Verwechslungen mit dem Java-Interface `Queue` auszuschließen.

b) Die Klasse kann nur `int`-Werte speichern. Ändern Sie sie so, dass sie generische Typen speichern kann.
   - Sowohl die Klasse `MyQueue<T>` als auch die Klasse `QNode<T>` braucht einen generischen Parameter.
   - Der Konstruktor beispielsweise von `QNode<T>` heißt nicht

     ```
     public QNode<T>(T key) { … }
     ```

     sondern

     ```
     public QNode(T key) { … }
     ```

c) Ändern Sie die Methode `dequeue` so ab, dass der entnommene Wert zurückgegeben wird:

   ```
   public T dequeue()
   ```

   Hinweis: Verwechseln Sie nicht `dequeue` (etwas aus der Warteschlange entnehmen) und `deque` (double ended queue).

d) Fügen Sie der Klasse `MyQueue` folgende Methoden hinzu:

- `public void clear()`
  Löscht die Queue
- `public String toString()`
  Gibt eine String-Darstellung der Form [Element1, Element2, …] zurück.
- `public void enqueueAtFront(T key);`
  Stellt ein neues (Vordrängler-)Element vorne in die Warteschlange (wie bei einem Stack).

Aufgabe 2

Schreiben Sie eine Klasse `MyHashSet<K>` für eine Menge, die mit einer Hashtabelle mit Teillisten implementiert wird. Verwenden Sie dazu die Vorlage aus dem ILIAS-Kurs.

- Fügen Sie folgende Methoden hinzu:
  - `public boolean delete(K element)`
    Löscht das angegebene Element, falls es existiert. Gibt `true` zurück, wenn das Element existiert hat. Die Anzahl der Teillisten wird dadurch nicht verkleinert.
  - `public boolean contains(K element)`
    Gibt zurück, ob das Element existiert.
  - `public String toString()`
    Gibt eine String-Darstellung der Form [Element1, Element2, …] zurück. Die Reihenfolge der Elemente ist unbestimmt.

- Prüfen Sie mit der Funktion `hashTest`, ob die Klasse korrekt läuft, und notieren Sie die Laufzeit.

- Ändern Sie die Methode `getHash`, so ab, dass die Hashfunktion besonders schlecht ist (siehe unten). Überprüfen Sie, ob die Klasse immer noch korrekt funktioniert, und notieren Sie die neue Laufzeit.

```
private int getHash(K element) {
      return 0;
}
```

- Ändern Sie die Klasse so ab, dass die Hashtabelle dynamisches Hashing benutzt. Wenn der Füllgrad (Anzahl der Elemente geteilt durch Anzahl der Teillisten) der Hashtabelle den Wert 2 übersteigt, soll die Anzahl der Teillisten verdoppelt werden. Dazu müssen sämtliche Elemente neu einsortiert werden.
  Hinweis: Die Anzahl der Teillisten muss beim Löschen von Elementen nicht wieder verkleinert werden.
- Überprüfen Sie erneut Funktionalität und Laufzeit mit der Testfunktion. Testen Sie sowohl die gute als auch die schlechte Hashfunktion.
- Stellen Sie in einer Tabelle alle 4 Laufzeitmessungen der jeweiligen O-Klasse gegenüber.

Zu Aufgabe 2 soll am Ende abgegeben werden:

a) Der Code der Hashtabelle mit dynamischem Hashing und „guter“ Hashfunktion.
b) Die Tabelle mit den O-Klassen der add-Funktionen und den Laufzeitmessungen.