TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Computer Vision Group
Einführung in die Informatik I für Ingenieure (MSE)
Prof. Dr. Daniel Cremers, Mohamed Souiai

WS 2012
Übungsblatt 9
04.01.2013

Aufgabe 9.1 (Ü) Queues

Eine Schlange ist eine Datenstruktur zur Speicherung von Elementen nach dem FIFO-Prinzip (First-In-First-Out). Die zweite der in der Vorlesung vorgestellten Implementierungen von Schlangen (Klasse: `Queue`) verwendet zur Speicherung der Elemente ein Array. Zwei Indizes, `first` und `last`, geben dabei die Position des ersten bzw. letzten Wertes an. Bei Bedarf wird das Array beim Einfügen von Elementen durch `enqueue()` durch ein größeres Array ersetzt. Eine Verkleinerung erfolgt nicht.

Erweitern Sie die in der Vorlesung vorgestellte Methode `dequeue()` (zum Entnehmen des nächsten Elements), so dass das Array bei Bedarf auch verkleinert wird.

Hinweis: Die zu ergänzende Klasse `Queue` finden Sie auf der Internetseite zur Übung.

Lösungsvorschlag 9.1

Analog zur Stack-Implementierung aus der Vorlesung wird das Feld halbiert, wenn es nur noch zu einem Viertel gefüllt ist. Somit wird vermieden, dass ein gerade verkleinertes Feld bei einem unmittelbar folgenden Aufruf von `enqueue()` sofort wieder vergrößert werden muss (siehe Vorlesung).

Bei der Verkleinerung des Feldes ist zu berücksichtigen, dass die Elemente in der Queue (im Gegensatz zum Stack) nicht unbedingt am Anfang des Feldes liegen, sondern

- in `a[first]`, ..., `a[last]`, falls `first < last`.
- in `a[first]`, ..., `a[a.lengh-1]`, `a[0]`, ..., `a[last]`, sonst.

Die erweiterte Methode `dequeue()`:

```
public int dequeue() {
    if(isEmpty())
        return 0;
    int result = a[first];
    if (first == last) {
        // letzter Wert wurde gelesen; das Feld ist nun leer
        first = last = -1;
    } else {
        // Anfangszeiger ein Element weiter setzen
        int length_of_array = a.length;
        first = (first + 1) % length_of_array;

        // Anzahl der enthaltenen Elemente == Differenz+1
        int number_of_elements;
        if (first < last)
            number_of_elements = last - first + 1;
```

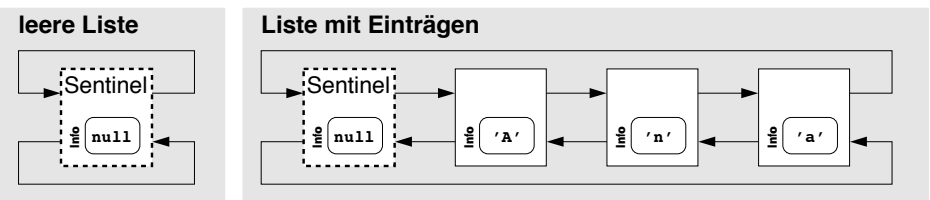
```
            else
                number_of_elements = length_of_array - first + last + 1;

            // verkleinern falls nur ein Viertel des Feldes gefÃ¼llt ist
            if (number_of_elements <= length_of_array / 4) {
                int[] b = new int[length_of_array / 2]; // Feld halbieren
                int index = 0;

                // Ã¼berprÃ¼fe wo im Feld die Element liegen
                if (first <= last) {
                    // kopiern der Elemente von first bis last
                    for (int i = first; i <= last; i++) {
                        b[index++] = a[i];
                    }
                } else {
                    // kopieren der Elemente von first bis Ende
                    for (int i = first; i < length_of_array; i++) {
                        b[index++] = a[i];
                    }
                    // kopieren der Element vom Anfang bis last
                    for (int i = 0; i <= last; i++) {
                        b[index++] = a[i];
                    }
                }
                // altes Feld durch neues ersetzen; Zeiger neu setzen
                a = b;
                first = 0;
                last = index - 1;
            }
        }
        return result;
    }
```

Aufgabe 9.2 (H) Doppelt verkettete String-Listen

In dieser Übung soll eine *doppelt verkettete* Liste realisiert werden. In einer solchen Liste wird in jedem Listen-Eintrag (Klasse `Entry`) eine Referenz auf den Vorgänger, eine Referenz auf den Nachfolger sowie eine Referenz auf die eigentliche Information gespeichert. Wir implementieren die doppelt verkettete Liste zyklisch. Dabei gibt es einen ausgezeichneten Listen-Eintrag, der keine Information speichert (das sogenannte *Sentinel-Element*) und sowohl vor dem ersten als auch hinter dem letzten eigentlichen Listen-Eintrag steht (siehe Abbildung). Der erste (bzw. letzte) eigentliche Listen-Eintrag ist dann der Nachfolger (bzw. Vorgänger) des Sentinel-Elementes. Insbesondere wird die leere Liste wie in der nachfolgenden Abbildung repräsentiert.



Implementieren Sie die Klassen `Entry` und `LinkedList`. Ein Objekt der Klasse `LinkedList` soll eine zyklisch doppelt verkettete Liste repräsentieren und muss dementsprechend eine Referenz auf das Sentinel-Element halten. In der Klasse `LinkedList` soll Folgendes definiert sein:

a) Ein Konstruktor zum Erzeugen von leeren Listen.
b) Eine Methode `void append(String s)`, die den String `s` am Ende der Liste anhängt.
c) Eine Methode `String toString()`, die die Liste von Strings ausgibt.
d) `int size()`, die die Länge der Liste zurückliefert;
e) Eine Methode `boolean contains(String s)`, welche darüber Auskunft gibt, ob eine Liste einen String, der `s` gleicht, enthält.

Lösungsvorschlag 9.2

Achtung: Diese Klasse am Besten Schritt für Schritt wachsen lassen, nicht gleich von Anfang an z.B. das Size-Attribut mitführen. Ruhig zeigen, dass man Methoden auch mal im Nachhinein nochmal anpassen muß. Beim contains() darauf hinarbeiten, dass man die null-Pointer Problematik anspricht. Entweder man geht davon aus, dass die Liste nie um null-Strings erweitert werden konnte (dann muß append() angepasst werden), oder man passt hier beim .equals() auf!

```
public class Entry {
        String element;
        Entry next;
        Entry previous;

        Entry(String element, Entry next, Entry previous) {
                this.element = element;
                this.next = next;
                this.previous = previous;
        }

        public String getElement() {
                return element;
        }

        public String toString() {
                return "[Entry : " + element + "]";
        }
}

public class LinkedList {
    private Entry sentinel;
    public LinkedList() {
        size = 0; // Achtung - erst ab Teilaufgabe d hinzugefuegt
        sentinel = new Entry(null, null, null);
        sentinel.previous = sentinel.next = sentinel;
    }

    public void append(String s){
        Entry insertBefore = sentinel;
        if (s!=null){ // Konsistenzbedingung: Die Liste enthaelt keine
            null ausser im Sentinel
             Entry newEntry = new Entry(s, insertBefore, insertBefore.
                previous);
             newEntry.previous.next = newEntry;
             newEntry.next.previous = newEntry;
             size++; // Achtung - erst ab Teilaufgabe d hinzugefuegt
        }
    }
    public String toString() {
        String retVal = "[";
```

```
        for (Entry e = sentinel.next; e != sentinel; e = e.next) {
            retVal = retVal + e.getElement();
            if (e.next != sentinel) // Man ist nicht bem letzten
               Element, also "," ausgeben
                {
                    retVal = retVal + ",";
                }
        }
        return retVal + "]";
    }

    private int size;
    public int size(){
        return size;
    }
    public boolean contains(String s){
        for (Entry e = sentinel.next; e != sentinel; e = e.next) {
            if (e.getElement().equals(s))
                return true;
        }
        return false;
    }



}
```