Hoofdstuk 3

Arrays and linked lists

INTRODUCTIE

De datastructuren die in de cursus behandeld worden, worden eerst geformuleerd in de vorm van een abstract datatype (ADT). Een ADT formuleren we vervolgens als een Java interface. Dit hoofdstuk biedt het gereedschap dat we in de komende hoofdstukken zullen gebruiken om de datastructuren te implementeren met een concrete Java klasse. Deze klassen maken vaak gebruik van een array of van een geschakelde lijst.

Tekstboekparagraaf 3.1 over arrays kunt u globaal doornemen.
De subparagrafen 3.1.4 en 3.1.5 behoren niet tot de leerstof.

LEERDOELEN
Na het bestuderen van deze leereenheid wordt verwacht dat u
- het insertion-sort algoritme kunt beschrijven en kunt toepassen
- methoden uit de klasse java.util.Arrays kent en kunt toepassen
- de eigenschappen van enkel-, dubbel- en circulairgeschakelde lijsten alsmede het toevoegen en verwijderen van schakels aan die lijsten kunt omschrijven.
- een shalow of deep copy van een object kunt maken

Studeeraanwijzingen
In de bijlage vindt u diagrammen van de belangrijkste interfaces en klassen die worden besproken in dit hoofdstuk.

LEERKERN

3.1 Practical uses of arrays

Van deze paragraaf zijn figuren 3.2 en 3.4 van belang. Zij maken duidelijk dat bij toevoegen aan of verwijderen uit een array elementen moeten worden opgeschoven. Bij het bepalen van de verwerkingstijd van een algoritme spelen dergelijke verschuivingen een rol. We komen hier in hoofdstuk 6 op terug.

3.2 Singly linked lists

OPGAVE 3.1
Maak opgave R-3.4 uit het tekstboek.

3.3 Circulary linked lists

OPGAVE 3.2
Maak opgave R-3.8 uit het tekstboek.

3.4 Doubly linked lists

Markeringen (sentinels), zoals header en trailer, worden geïntroduceerd om het programmeren makkelijker te maken. Dankzij de twee sentinels hebben alle elementen uit de lijst dezelfde structuur. Ze bezitten allemaal een voorganger en een opvolger. Als er geen sentinels gebruikt zouden worden, dan zou de dubbelgeschakelde lijst verwijzingen hebben naar de eerste en de laatste schakel, en zou steeds getest moet worden of deze niet null zijn.

De figuren 3.20 en 3.22 tonen aan dat het toevoegen en verwijderen van een schakel bewerkingen zijn waarvoor slechts enkele wijzermanipulaties nodig zijn. Deze bewerkingen worden geïmplementeerd in codefragment 3.17 door de methoden addBetween en remove.

OPGAVE 3.3
Maak opgave R-3.6 uit het tekstboek. U mag er hierbij van uitgaan dat de lijst een oneven aantal schakels bevat.
De vraag over de verwerkingstijd van het algoritme kunt u pas beantwoorden na bestudering van hoofdstuk 4.

TERUGKOPPELING

Uitwerking van de opgaven

3.1 Het eerste element van de lijst wordt door head aangewezen.
De methode penultimate geeft de voorlaatste schakel van de lijst terug.

```
/** Geeft de voorlaatste schakel als deze aanwezig is,
 *  anders null.
 */
public Node<E> penultimate() {
  if (head == null) {
    return null ;
  }
  Node<E> vorige = null;
  Node<E> node = head;
  while (node.getNext() != null) {
    vorige = node;
    node = node.getNext();
  }
  return vorige;
}
```

3.2 Variabele current geeft de huidige positie aan in de circulairgeschakelde lijst.

```
public int size() {
  if (tail == null) {
     return 0;
  }
  Node<E> current = tail.getNext();
  int aantal = 1;
  while (current!= tail) {
    aantal++;
    current = current.getNext();
  }
  return aantal;
}
```

3.3 De lijst heeft een oneven aantal schakels. Daarom bestaat de middelste schakel altijd. Om de middelste schakel te vinden, start het algoritme bij header en bij trailer tegelijk en gaat het via link hopping naar het midden van de lijst.

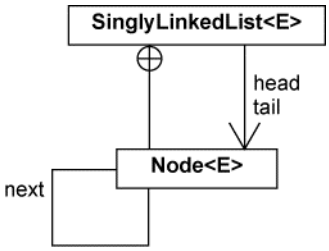
Algorithm middelste()
{
  Zoekt de middelste schakel van een dubbelgeschakelde
  lijst met oneven aantal schakels en
  met markeringen header en trailer.
}
  ***Input*:** Geen
  ***Output*:** Middelste schakel

  Laat voorste en achterste twee schakels zijn.
  voorste ← header
  achterste ← trailer
  while voorste != achterste **do**
    voorste ← voorste.getNext()
    achterste ← achterste.getPrev()
  { end while }
  return voorste

Bijlage

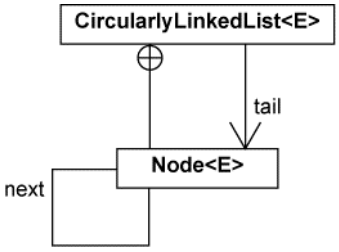
Diagrammen belangrijkste interfaces en klassen

1 Singly linked list



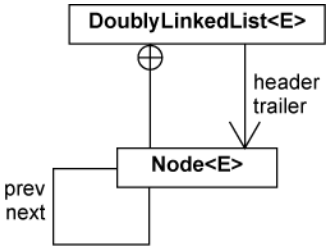
FIGUUR 3.1 Diagram van de klasse SinglyLinkedList<E> met binnenklasse Node<E>

2 Circularly linked list



FIGUUR 3.2 Diagram van de klasse CircularlyLinkedList<E> met binnenklasse Node<E>

3 Doubly linked list



FIGUUR 3.3 Diagram van de klasse DoublyLinkedList<E> met binnenklasse Node<E>

NB De cirkel met kruis in de diagrammen geeft aan dat de klasse Node<E> een binnenklasse is.