2. Dezember 2013

HAW Hamburg

Dokumentation

Zu Aufgabenblatt 08 aus der Vorlesungsreihe „Algorithmen und Datenstrukturen“

Dokumentation

Zu Aufgabenblatt 08 aus der Vorlesungsreihe „Algorithmen und Datenstrukturen“

# ÜbungsAufgabe 8.1

*Mergesort.*

## Teilaufgabe 1

*Erweitern Sie Ihr Listen-Interface um eine Methode, die überprüft, ob eine Liste aufsteigend sortiert ist. Implementieren Sie diese.*

@Override

**public** **boolean** isIncreasingMonoton() {

**if**(tail.top() == **null**) {

**return** **true**;

} **else** {

**return** ((**int**) **this**.top()) <= ((**int**) **this**.tail.top()) && tail.isIncreasingMonoton();

}

}

## Teilaufgabe 2

*Implementieren Sie eine Methode, die eine Zufallsliste einer vorgegebenen Länge n erzeugt, d.h. die Methode erzeugt zufällig eine Liste mit n Elementen. Testen Sie diese Methode.*

**public** IList random(**int** n) {

IList akku = **new** MyList();

Random generator = **new** Random();

**for**(**int** i = 0; i < n ; i++) {

akku.head(generator.nextInt(1000));

}

**return** akku;

}

## Teilaufgabe 3

*Entwickeln Sie eine Methode, die eine sortiert Zufallsliste einer vorgegebenen Länge n erzeugt, d.h. die Methode erzeugt zufällig eine Liste mit n Elementen mit der Eigenschaft, dass die Elemente monoton ansteigend sind.  
Testen Sie diese Methode (in Bezug auf das Aufsteigen) mit der zuvor implementierten Test-Methode.*

@Override

**public** IList sortIncreasingMonoton(**int** n) {

IList akku = **new** MyList();

IList randomList = **this**.random(n);

**while**(randomList.top() != **null**) {

**int** elem = (**int**) randomList.head();

IList temp = akku;

**int** i = 0;

**while**(**true**) {

**if**(temp.top() == **null** || (**int**) temp.top() >= elem) {

akku.insert(elem, i);

**break**;

} **else** {

i++;

temp = temp.tail();

}

}

}

**return** akku;

}

## Teilaufgabe 4

*Entwickeln Sie eine Methode, die zwei sortierte Listen „mergen“ kann.*

@Override

**public** IList merge(IList list) {

IList akku = **new** MyList();

IList list1 = **this**;

IList list2 = list;

**while**(list1.top() != **null** && list2.top() != **null**) {

**if**((**int**) list1.top() < (**int**) list2.top()){

akku.insert(list1.head(), akku.length());

} **else** {

akku.insert(list2.head(), akku.length());

}

}

**while**(list1.top() != **null**) {

akku.insert(list1.head(), akku.length());

}

**while**(list2.top() != **null**) {

akku.insert(list2.head(), akku.length());

}

**return** akku;

}

## Teilaufgabe 5

*Testen Sie Ihre Merge-Methoden, indem Sie jeweils zwei sortierte Zufallslisten erzeugen, diese mergen und überprüfen, ob das Ergebnis eine sortierte Liste ist. Wiederholen Sie dieses Experiment mit Listen verschiedener Länge.*

Jo, läuft.

## Teilaufgabe 6

*Implementieren Sie – aufbauend auf Ihrer Merge-Methode – das Mergesort-Verfahren.*

@Override

**public** IList mergeSort() {

**if**(**this**.length() <= 1) {

**return** **this**;

} **else** {

IList list1 = **new** MyList();

IList list2 = **new** MyList();

**int** initSize = **this**.length() / 2;

**while**(**this**.top() != **null**) {

**if**(**this**.length() > initSize) {

list1.head(**this**.head());

} **else** {

list2.head(**this**.head());

}

}

**return** list1.mergeSort().merge(list2.mergeSort());

}

}

## Teilaufgabe 7

*Testen Sie Ihre Mergesort-Methode, indem Sie eine Zufallsliste erzeugen, diese sortieren und überprüfen, ob das Ergebnis eine sortierte Liste ist. Wiederholen Sie dieses Experiment mit Listen verschiedener Längen.*

Jo, läuft.