Aufgaben zur Veranstaltung
Algorithmen und Datenstrukturen, SS 2022

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

Klausurvorbereitung, Blatt 3 28.06.2022

Aufgabe 1

Gegeben sei das folgende Programmfragment:

```
public static void unknown(int[] numbers) {
    boolean swapped = true;
    for(int i = numbers.length - 1; i > 0 && swapped; i--) {
        swapped = false;
        for (int j = 0; j < i; j++) {
            if (numbers[j] > numbers[j+1]) {
                int temp = numbers[j];
                numbers[j] = numbers[j+1];
                numbers[j+1] = temp;
                swapped = true;
            }
        }
    }
}
```

a) Wenden Sie den obigen Algorithmus auf das folgende Feld an. Geben Sie den Inhalt des Feldes jeweils nach Durchlaufen der äußeren Schleife an.

	22	**108**	**74**	**25**	**42**	**3**	**112**	**15**
i=7	22	74	25	42	3	108	15	112
i=6	22	25	42	3	74	15	108	112
i=5	22	25	3	42	15	74	108	112
i=4	22	3	25	15	42	74	108	112
i=3	3	22	15	25	42	74	108	112
i=2	3	15	22	25	42	74	108	112
i=1	3	15	22	25	42	74	108	112

b) Was leistet der Algorithmus? *Aufsteigendes Sortieren einer Liste von Elementen*

c) Wie ist die Laufzeitkomplexität im Best-Case und im Worst-Case. Geben Sie jeweils eine möglichst kleine obere Schranke mit Hilfe der O-Notation an.
Best-Case (Liste schon sortiert): O(n), Worst-Case: $O(n^2)$

Aufgabe 2

Schreiben Sie eine Klasse *HashSet* für eine Menge von Strings mit folgenden Eigenschaften:

- *HashSet* implementiert eine Hashtabelle mit Teillisten. Verwenden Sie als Teillisten Objekte der Klasse *ArrayList*.
- Die Elemente sind Strings, in denen ausschließlich Großbuchstaben vorkommen. Sie müssen die Strings nicht auf Korrektheit überprüfen.
- *Hash* enthält die Methoden
  `public void add (String s)`
  zum Hinzufügen eines Strings in die Hash-Tabelle und
  `public boolean contains(String s)`
  zur Überprüfung, ob ein String in der Hash-Tabelle vorhanden ist.
- Die Hashfunktion ist die Position des 1. Buchstabens des Strings im Alphabet. Beispiel: „HALLO“ ergibt 8. Hinweis: A hat den ASCII-Wert 65.
- Wird versucht ein String einzufügen, der schon vorhanden ist, wird nichts getan.

```
public class HashSet {
	private ArrayList<String>[] teillisten = new ArrayList[27];

	private int hash(String x) {
		return x.charAt(0)-64;  //"HALLO" -> 8
	}

	public boolean contains(String s) {
		if (teillisten[hash(s)]==null) {
			//Teilliste nicht vorhanden
			return false;
		}
		return teillisten[hash(s)].contains(s);
	}

	public void add(String s) {
		if (teillisten[hash(s)]==null) {
			//Teilliste nicht vorhanden -> erstellen
			teillisten[hash(s)]=new ArrayList<String>();
			teillisten[hash(s)].add(s);
			return;
		}
		ArrayList<String> l = teillisten[hash(s)];
		int index = l.indexOf(s);
		if (index==-1) {
			//Element noch nicht vorhanden -> erstellen
			l.add(s);
		}
	}
}
```

Aufgabe 3

Ein Binärbaum für Integer-Zahlen soll implementiert werden.

a) Schreiben Sie die Klasse *Node* für einen Knoten des Binärbaums.
b) Für die Klasse *BinaryTree* sollen Sie die nötigen Attribute, sowie zwei Konstruktoren schreiben. Die Konstruktoren sind:

```
public BinaryTree(int x)
//Erzeugt einen neuen Baum mit dem einzigen Element x

public BinaryTree(BinaryTree b)
//Kopiert den Binaerbaum.
```

Verwenden Sie zum Kopieren eine rekursive Methode.

```
public class Node {
	public int value; public Node left; public Node right;

	public Node(int value) {
		this.value = value;
	}
}

public class BinaryTree {
	private Node root;

	public BinaryTree(int x) {
		this.root = new Node(x);
	}

	public BinaryTree(BinaryTree b) {
		if (b.root == null) {
			return;
		}
		//root-Node kopieren
		root = new Node(b.root.value);
		Node from = b.root;
		Node to = root;
		//Rechten und linken Teilbaum kopieren
		copySubtrees(from, to);
	}

	private void copySubtrees(Node from, Node to) {
		if (from.left != null) {
			//Linken Teilbaum kopieren
			to.left = new Node(from.left.value);
			copySubtrees(from.left, to.left);
		}
		if (from.right != null) {
			//Rechten Teilbaum kopieren
			to.right = new Node(from.right.value);
			copySubtrees(from.right, to.right);
		}
	}
```

Aufgabe 4

Fügen Sie der Klasse BinaryTree aus Aufgabe 3 eine Methode

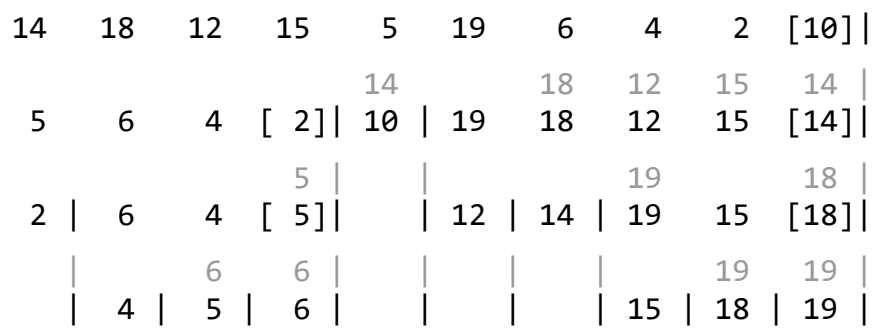
```
public int getMaxLevel()
```

hinzu. Die Methode soll zurückgeben, in welchem Level sich der Knoten mit dem größten Wert befindet.

Hinweis: Die Wurzel hat den Level 0. Der Binärbaum ist kein binärer Suchbaum.

```
public int getMaxLevel() {
	//ret: [Level des max Werts, max Wert]
	int[] ret = {0, root.value};
	getMaxLevel(0, root, ret);
	return ret[0];
}

private void getMaxLevel(int level, Node n, int[] max) {
	if (n == null) {
		return;
	}
	if (n.value>max[1]) {
		max[0] = level;
		max[1] = n.value;
	}
	getMaxLevel(level + 1, n.left, max);
	getMaxLevel(level + 1, n.right, max);
}
```

Aufgabe 5



Aufgabe 6

Ein Heap (größtes Element in der Wurzel) ist in eine ArrayList eingebettet.

Schreiben Sie eine Funktion

```
public static void insertElement(ArrayList<Integer> heap, int newElement)
```

welche das neue Element an die richtige Stelle im Heap einfügt.

Hinweis: Kommentieren Sie die Einbettung in das Array sowie die Methode zur Bewegung der Elemente im Heap.

```
public static void insertElement(ArrayList<Integer> heap, int newEle-
ment) {
	if (heap.size() == 0) {
		heap.add(0); //Element 0: Dummy
		heap.add(newElement);
	} else {
		int pos = heap.size();
		heap.add(newElement);
		//Vertauschen nach oben
		while (pos > 1 && heap.get(pos / 2) < heap.get(pos)) {
			Collections.swap(heap, pos, pos / 2);
			pos /= 2;
		}
	}
}
```

Aufgabe 7

Schreiben Sie eine Methode

```
public static int findElement(int[] array, int val)
```

die im Feld array das Element mit dem Wert val sucht und seine Position zurückgibt. Falls das Element nicht vorkommt, wird -1 zurückgegeben.

Benutzen Sie dabei die binäre Suche. Gehen Sie davon aus, dass das Feld aufsteigend sortiert ist.

Tipp: Benutzen Sie einen rekursiven Algorithmus und schreiben Sie dazu die Methode

```
private static int findElement(int[] array, int val, int first, int last)
```

die `val` im Teilfeld zwischen `first` und `last` (einschließlich) sucht.

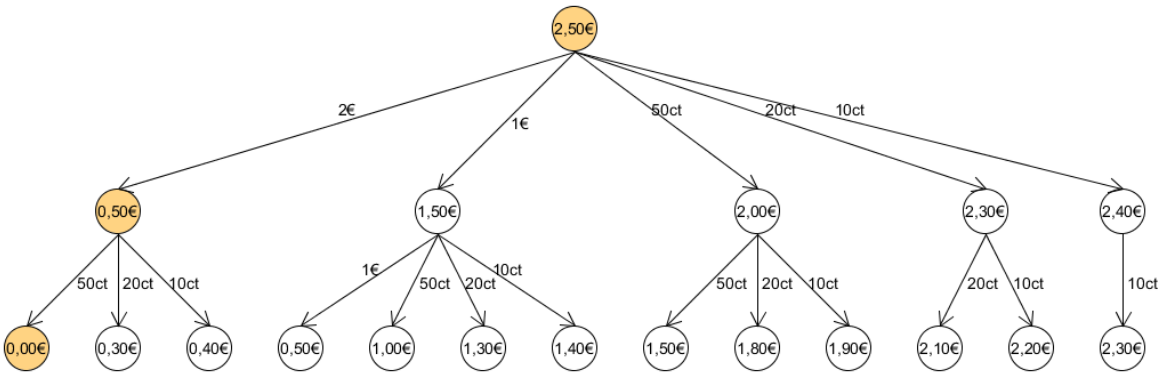
```
public static int findElement(int[] array, int val) {
      return findElement(array, val, 0, array.length-1);
}


//Binäre Suche in einem sortierten Feld zwischen den Indizes
//first und last (jeweils einschliesslich).
private static int findElement(int[] array, int val, int first, int last) {
      if (first>last) {
            return -1;
      }
      int mid =(first+last)/2;
      if (val==array[mid]) {
            return mid;
      }
      if (val>array[mid]) {
            return findElement(array, val, mid+1, last);
      } else {
            return findElement(array, val, first, mid-1);
      }
}
```

Aufgabe 8

Ihre Aufgabe ist es, einen Fahrkartenautomaten zu simulieren. Dabei gibt es die folgenden Randbedingungen:

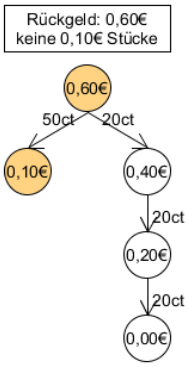
- Abstufung der Fahrpreise in 10 Cent-Beträgen
- Akzeptiert werden Münzen ab 10 Cent und Scheine bis 10 Euro
- Nur Münzen (ab 10 Cent) werden als Wechselgeld ausgegeben (2€, 1€, 50ct, 20ct und 10ct)
- Der Bestand an Wechselgeld ist endlich und wird mit simuliert.
- Die Ausgabe soll mit möglichst wenigen Geldstücken erfolgen.

a) Gehen Sie zunächst von einem Automaten aus, der ausreichend Wechselgeld enthält und zeichnen Sie die ersten drei Ebenen des entsprechenden Suchbaums für die Rückgabe von 2,50€. Welcher Zweig des Suchbaums wird mit einem Greedy-Algorithmus durchlaufen?



Der Greedy-Algorithmus versucht in jedem Schritt die größtmögliche Münze auszugeben und verfolgt den orange markierten Pfad. Bei ausreichend Wechselgeld führt der Greedy-Algorithmus bei dieser Zusammensetzung von Wechselgeld immer zum optimalen Ergebnis.

b) Betrachten Sie nun einen Automaten der keine 10ct Stücke mehr enthält und 60ct als Wechselgeld zurückgeben soll. Zeichnen Sie den kompletten Suchbaum und benennen Sie, welches Problem mit dem Greedy-Algorithmus auftritt. Beschreiben Sie anschließend kurz, mit welchen Entwurfsprinzipien dieses Problem behoben werden kann.



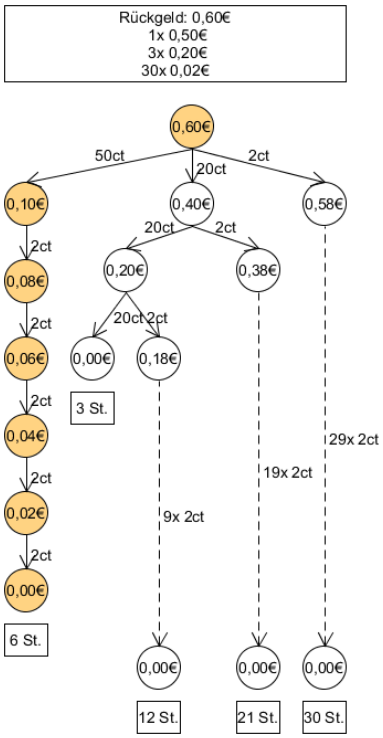
Sind keine 10ct Münzen mehr vorhanden so kann nicht mehr jeder Pfad zu einem zulässigen Ergebnis führen. Der Greedy-Algorithmus versucht die größtmöglichen Münzen auszugeben, findet jedoch kein Ergebnis mehr, nachdem er einmal eine 50ct Münze ausgegeben hat. Hier könnte ein Backtracking Ansatz verwendet werden. Führt ein Pfad nicht zu einem Ergebnis werden solange die zuletzt hinzugefügten Münzen entfernt, bis der Algorithmus einen anderen Pfad findet, der zu einem zulässigen Ergebnis führt.

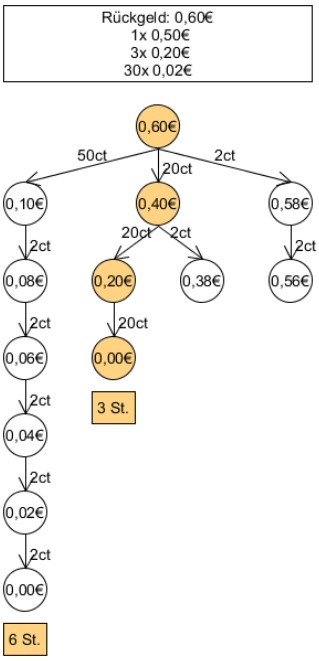
c) Was ändert sich bei der Suche, wenn der Automat nun außerdem noch 5ct und 2ct Stücke als Wechselgeld verwenden kann und aktuell folgendes Wechselgeld zur Verfügung hat

- 1x 50ct
- 3x 20ct
- 50x 2ct

Es soll 0,60€ als Wechselgeld zurückgegeben werden. Zeichnen Sie den Suchbaum und geben Sie unter den gefundenen Lösungen jeweils die Anzahl der benötigten Münzen an.

Bei dieser Zusammensetzung an Wechselgeld findet der Backtracking-Ansatz aus b nicht mehr die optimale Lösung.





d) Mit welchem Entwurfsprinzip lässt sich die Suche in Aufgabenteil c verbessern?
Beschreiben Sie das Vorgehen und kennzeichnen Sie die Optimierungen im Suchbaum.

Um weiterhin die optimale Lösung zu finden muss ein größerer Teil des Baumes durchsucht werden. Dabei dienen bereits gefundene Lösung als Kriterium um die Überprüfung von Zweigen des Baumes, die nicht zu einem besseren Ergebnis führen können, frühzeitig abzubrechen (Branch and Bound). Das Kriterium kann jedes Mal aktualisiert werden wenn ein besseres Ergebnis gefunden wurde.
Beispiel: die erste gefundene Lösung besteht aus 6 Münzen der zweite Pfad führt zu einer besseren Lösung welche aus 3 Münzen besteht. Da nun bereits eine Lösung mit drei Münzen gefunden wurde kann jeder Teilbaum der keine Lösung mit zwei Münzen gefunden hat ignoriert werden.