

Übungen zu Objektorientierte Programmierung 2

frieder.loch@ost.ch

Woche 14: Repetitionsaufgaben/Prüfungsvorbereitung - Lösungen

Diese Aufgaben sind für die Prüfungsvorbereitung gedacht. Sie enthalten sowohl Programmier-, wie auch Wissensaufgaben und ähneln im Typ und der Komplexität möglichen Prüfungsaufgaben. Die Programmieraufgaben müssen nicht in Java programmiert werden, sondern können auch als Pseudocode geschrieben werden. Der Pseudocode muss ausreichend genug sein um, zum Beispiel, eine genaue Analyse mittels der O-Notation zuzulassen.

Aufgabe 1: Rekursion

String reverse

Rekursiv

Implementieren Sie eine rekursive Methode, die einen String umkehrt. Aus abcd wird dcba

```
public String reverseString(String text) {  
    // TODO  
}
```

Iterativ

Erstellen Sie nun eine iterative Variante dieser Methode.

Ziffern-Summieren

Implementieren Sie eine rekursive Methode, welche alle Zahlen, die aus n Ziffern bestehen und eine bestimmte Quersumme ergeben berechnet. Zum Beispiel alle zweistelligen Zahlen, welche die Summe 7 ergeben. Die Resultate werden in der `resultList` gespeichert.

Resultat: 16 25 34 43 52 61 70.

```
public void findNdigitNums(String currentNumber, int n, int target, List<Integer> resultList) {  
    // TODO  
}
```

Aufgabe 2: Laufzeitverhalten

a: Laufzeit der Funktion auslesen

Bestimmen Sie die Laufzeitklasse des folgenden Algorithmus in der O-Notation?

```
public static void function(int[] array) {  
    for (int i = 1; i < array.length + 1; i++) {  
        for (int i1 = 1; i1 < array.length; i1*=2) {  
            System.out.println(array[i - 1]);  
        }  
    }  
}
```

$O(n * \log n)$

b: Binary Search

Mit der binären suche, können sie relativ effizient eine sortierte Liste durchsuchen.

Jeder Algorithmus hat jedoch einen oder mehrere Best-Case und einen Worst-Case. Geben Sie mindestens ein Best-Case- und ein Worst-Case-Szenario der Binärsuche an.

Best-Case: Das gesuchte Element ist in der Mitte $\rightarrow O(1)$ Worst-Case: Das gesuchte Element ist nicht in der Liste $O(\log n)$ Worst-Case: Das gesuchte Element ist am tiefsten Punkt des Baums, der während der suche erstellt wird $O(\log n)$

c: Laufzeit-Beweis

Beweisen Sie: $8n^4 + 3n^2 + 4$ ist $O(n^4)$

Gesucht: $c > 0$ und $n_0 \geq 1$, so dass $8n^4 + 3n^2 + 4 \leq cn^4$ für $n \geq n_0$

$$\begin{aligned} 8n^4 + 3n^2 + 4 &\leq cn^4 \\ \Leftrightarrow 3n^2 + 4 &\leq cn^4 - 8n^4 \\ \Leftrightarrow 3n^2 + 4 &\leq n^4(c - 8) \\ \Leftrightarrow \frac{3n^2 + 4}{(c - 8)} &\leq n^4 \\ \Rightarrow c &= 9 \\ \Leftrightarrow 3n^2 + 4 &\leq n^4 \\ \Leftrightarrow 4 &\leq n^4 - 3n^2 \\ \Leftrightarrow 4 &\leq n^4 - 3n^2 \\ \Rightarrow n &= 2 \end{aligned}$$

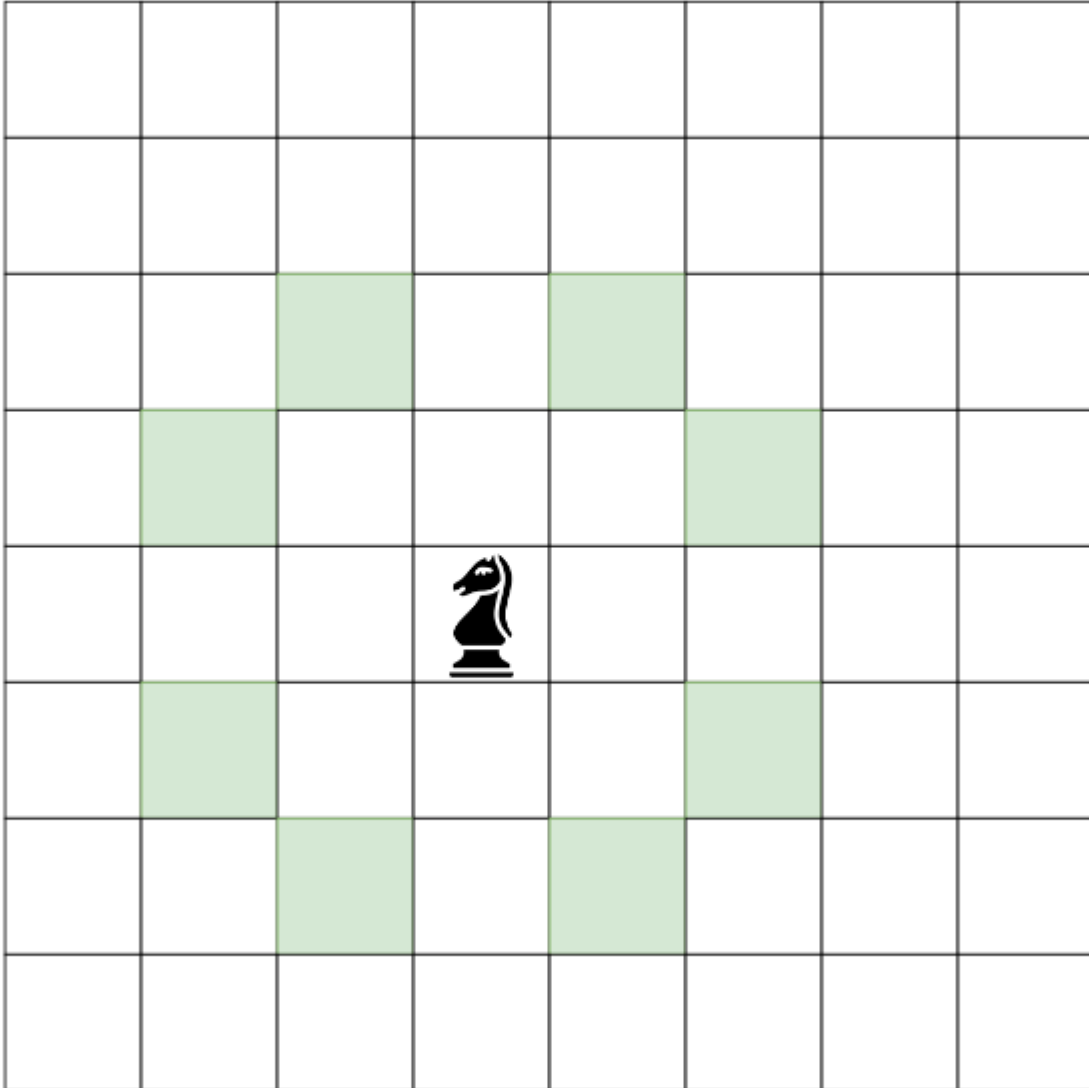
Solution: $c = 9, n_0 = 2$ Es gibt auch weitere Lösungen für c und n .

Aufgabe 3: Backtracking

Knight's Tour

Man stelle sich ein N mal N grosses Schachbrett vor. Dort befindet sich ein Knight(Springer). Bei der Knight's Tour geht es darum einen Weg zu finden, so dass der Knight auf jedem Feld genau einmal war.

Ein Springer darf sich auf dem Schachbrett nur in einem L bewegen. Im Bild unten sind alle möglichen Felder grün



markiert.

In einem 2-Dimensionalen Array kann sich der Knight wie folgt bewegen:

		(i-2, j-1)		(i-2, j+1)			
	(i-1, j-2)				(i-1, j+2)		
			(i, j)				
	(i+1, j-2)				(i+1, j+2)		
		(i+2, j-1)		(i+2, j+1)			

Ihre Aufgabe ist es nun die Klasse KnightsTour zu implementieren:

```
public class KnightsTour {
    // N x N Schachbrett
    public static final int N = 5;

    //Alle möglichen bewegungen des Knights
    public static final int[] row = {2, 1, -1, -2, -2, -1, 1, 2, 2};
    public static final int[] col = {1, 2, 2, 1, -1, -2, -2, -1, 1};

    //Überprüfen, ob Position (x, y) noch innerhalb des Schachbretts ist
    private static boolean isValid(int x, int y) {

    }

    // Rekursive Funktion, die für das Backtracking benötigt wird
    public static boolean knightTour(int[][] visited, int x, int y, int pos) {
        // jetzige Position markieren

        // Abbruchsbedingung: Wenn alle Felder besucht sind abbrechen (return true)

        // Alle erlaubten Bewegungen überprüfen und falls Valid weiter rekursiv probieren

        // Markierung des jetzigen Felds entfernen und backtracken
    }

    public static void main(String[] args) {
```

```

// Abbildung des Schachbretts, welches auch den Ablauf festhält
int[][] visited = new int[N][N];
int pos = 1;

// Erster Aufruf
knightTour(visited, 3, 3, pos);
}

```

Aufgabe 4: Iterator/ Binärbaum

Implementieren Sie eine Klasse `TreeIterator`. Der Iterator ermöglicht es durch einen binären Suchbaum zu iterieren.

```
public class TreeIteratorPreorder<T> implements Iterator<T> {  
  
    public TreeIteratorPreorder(Node<T> root) {  
    }  
  
    @Override  
    public boolean hasNext() {  
    }  
  
    @Override  
    public T next() {  
    }  
}
```

Aufgabe 5: Sortieren in verschiedenen Laufzeiten

Sie starten mit einem unsortierten Array, welches jeweils zufällig viele 0, 1, 2 beinhaltet. Ihre Aufgabe besteht darin, das Array so zu sortieren, dass zuerst alle 0, dann alle 1, dann alle 2 stehen.

a: Sortieren Sie mit einem $O(n^2)$ Algorithmus

Mögliche Algorithmen: Selection Sort, Bubble Sort, Insertion Sort

b: Sortieren Sie mit einem $O(n * \log(n))$ Algorithmus

Möglicher Algorithmus: Merge Sort

c: Sortieren Sie mit einem $O(n)$ Algorithmus (Schwer)

Mögliche Algorithmen: Counting Sort, Dutch National Flag Problem