TU Ilmenau, Fakultät IA
FG Telematik/Rechnernetze
Prof. Dr.-Ing. G. Schäfer
http://www.tu-ilmenau.de/telematik/pua

TECHNISCHE UNIVERSITÄT
ILMENAU

Programmierung und Algorithmen WS 23/24

Übungsblatt 8

Die Lösungen der Aufgaben sind bis zum 17.12.23, 23:59 Uhr abzugeben.

Die Besprechung der Aufgaben erfolgt in KW 51.

Das Fachgebiet Telematik/Rechnernetze wünscht Ihnen eine geruhsame Weihnachtszeit und einen guten Rutsch ins neue Jahr!

Bleiben Sie gesund!

Aufgabe 1 (Türme von Hanoi) — **1 + 1 + 2 Punkte**

Wiederholen Sie zunächst den in der Vorlesung vorgestellten Algorithmus zur Lösung des Problems der Türme von Hanoi (Kapitel 2, Folie 26).

(a) Geben Sie die Rekurrenzrelation für die Anzahl $T(n)$ an benötigten Bewegungen für einen Turm der Höhe $n \geq 1$ an.

(b) Können Sie das Master-Theorem verwenden, um diese Rekurrenz abzuschätzen? Begründen Sie Ihre Antwort.

(c) Beweisen Sie mittels vollständiger Induktion, dass $T(n) = 2^n - 1$ gilt.

Aufgabe 2 (Fallstricke bei Induktionsbeweisen) — **4 Punkte**

Folgender Induktionsbeweis scheint zu zeigen, dass $42 \cdot n = 0$ für alle $n \in \mathbb{N}_0$ gilt.

Beweis. (Per Induktion über n)
IA: $n = 0 \rightarrow 42 \cdot 0 = 0$
IS: $< n \rightarrow n$:
Es seien $x < n$ und $y < n$ zwei Zahlen aus $\mathbb{N}_0$ mit $x + y = n$.
Dann folgt mit Hilfe der IV (für x und y):
$42 \cdot n = 42 \cdot (x + y) = 42 \cdot x + 42 \cdot y = 0 + 0 = 0$ □

Begründen Sie, *warum* dieser Beweis falsch ist beziehungsweise wo sich der Fehler befindet.

Hinweis: „Die Aussage ist offensichtlich falsch." ist keine Begründung dafür, dass der Beweis formal nicht korrekt ist.

Aufgabe 3 (Algorithmenentwurf per Induktion) — **6 Punkte**

In Kapitel 6 wurde Ihnen ab Folie 91 ein Induktionsbeweis für ein Problem aus der Graphentheorie vorgestellt. Dabei handelt es sich um einen *konstruktiven Beweis*, d.h. es wird nicht nur die Existenz der in der Problembeschreibung definierten unabhängigen Knotenmenge bewiesen, sondern man kann aus dem Beweis auch einen rekursiven Algorithmus zur Bestimmung dieser Menge „ablesen". Implementieren Sie den *resultierenden* Algorithmus in Java.

Gehen Sie dafür davon aus, dass die n Knoten der Eingabe von $0, \ldots, n-1$ durchnummeriert sind. Die Kantenmenge E kann dann als zweidimensionales Array `int[][] E` dargestellt werden, wobei

gilt: `E[i][j] = 1`, falls die gerichtete Kante `i` $\rightarrow$ `j` existiert und `E[i][j] = 0`, sonst ("Adjazenzmatrix"). Knotenmengen können in Arrays vom Typ `int[]` abgespeichert werden.

Zum Testen finden Sie in der WebIDE (und im Moodle-Kurs) ein „Framework", welches Ihren Algorithmus auf 1000 zufällig generierten Graphen mit je 100 Knoten testet und die Lösung auf Korrektheit überprüft.

Hinweise:

- Es genügt einen Basisfall zu implementieren: Für $V = \emptyset$ ist die gesuchte Menge $S(G) = \emptyset$.
- Auch wenn es auf den ersten Blick seltsam aussieht können Sie in Java ein leeres Array mit `new int[0]` erzeugen (was natürlich der leeren Menge entspricht).
- Im Induktionsschritt werden Knoten und Kanten aus dem Graphen G entfernt (Folie 92). Bei der Implementierung mittels der Adjazenzmatrix `E` genügt es, nur die Knotenmenge für den rekursiven Aufruf anzupassen. `E` kann unverändert übergeben werden. Die zu den entfernten Knoten zugehörigen Einträge in `E` sind dann sowieso nicht mehr (gezielt) auslesbar.

Aufgabe 4 (Master-Theorem) **8 Punkte**

Das Master-Theorem besagt, dass eine Rekurrenzgleichung $T(n) = aT(\frac{n}{b}) + f(n)$, mit Konstanten $a \geq 1, b > 1$ und einer Funktion $f(n)$ über den positiven Zahlen, wie folgt asymptotisch abgeschätzt werden kann:

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{falls gilt: } \exists\, \varepsilon > 0\colon f(n) = O(n^{\log_b a - \varepsilon}) \\ \Theta(n^{\log_b a} \log n) & \text{falls gilt: } f(n) = \Theta(n^{\log_b a}) \\ \Theta(f(n)) & \text{falls gilt: } \exists\, \varepsilon > 0\colon f(n) = \Omega(n^{\log_b a + \varepsilon}) \\ & \qquad \wedge \exists\, c, 0 < c < 1, \exists\, n_0\, \forall\, n \geq n_0\colon a \cdot f(\tfrac{n}{b}) \leq c \cdot f(n) \end{cases}$$

Nutzen Sie das Master-Theorem, um die folgenden Rekurrenzgleichungen abzuschätzen, bzw. geben Sie an wo dies nicht möglich ist. Begründen Sie Ihre Lösung. Geben Sie in jedem Fall zunächst a, b und $f(n)$ an. Geben Sie weiterhin passende Konstanten an sobald notwendig (z.B. ε und c).[1]

(a) $T(n) = 27 \cdot T\left(\frac{n}{3}\right) + 2n^{2.5}\sqrt{n}$

(b) $T(n) = T\left(\frac{n}{2}\right) + \sqrt{n}$

(c) $T(n) = 64 \cdot T\left(\frac{n}{4}\right) + 3n^2$

(d) $T(n) = 0.5 \cdot T\left(\frac{n}{2}\right) + n$

[1] Das Master-Theorem wird am Mittwoch, dem 13.12.2023, in der Vorlesung besprochen.

Die restlichen Aufgaben sind optional und gehen nicht in die Bonuspunkteberechnung mit ein, können aber trotzdem zur Korrektur abgegeben werden. Sollten Sie sich entscheiden, die Robocode-Aufgabe zu bearbeiten, so senden Sie Ihren Quelltext (ausnahmsweise nicht als PDF) bis zum 03.01.2024, 9:00 Uhr per Mail an den Übungsleiter Ihrer Seminargruppe.

Aufgabe 5 (Robocode)

Robocode[2] ist ein Framework zum spielerischen Training der Sprache Java. Ihre Aufgabe besteht darin, einen Roboter zu programmieren. Durch Ihre Programmlogik gesteuert, tritt dieser Roboter dann in einem Wettbewerb gegen die Roboter anderer Spieler an. Der letzte einsatzfähige Roboter hat gewonnen. Programmieren Sie einen solchen Roboter und schicken Sie ihn rechtzeitig an uns. Wir werden zum Übungstermin alle Einsendungen gegeneinander antreten lassen.

Hinweis: Um die Aufgabe zu bearbeiten, laden Sie sich die Robocode Simulationsumgebung[3] herunter und richten Sie sie ein. Gute Anleitungen für den Einstieg gibt es im RoboWiki[4]. Für den Wettkampf werden Standard-Bedingungen angenommen (Spielfeldgröße 800x600 etc.).

Um „Doping“ zu verhindern sind folgende Punkte zu berücksichtigen (wenn Sie einzelne Punkte noch nicht verstehen ist das nicht schlimm, sie werden dann auch keine Gefahr laufen gegen sie zu verstoßen):

- Verwenden Sie nur das Robocode, Math und Color Package (`import robocode.*; import java.math.*; import java.awt.Color;`).
- Leiten Sie Ihren Roboter von der Klasse `Robot` ab (nicht etwa `AdvancedRobot` oder ähnliches).
- Kopieren Sie keinen Code!
- Schreiben Sie lesbaren Code mit Kommentaren.
- Geben Sie Ihrem Roboter eine hübsche Farbe.

[2] http://robocode.sourceforge.net/
[3] https://sourceforge.net/projects/robocode/files/robocode/1.9.5.2/
[4] http://robowiki.net und http://robowiki.net/wiki/Robocode/My_First_Robot

Aufgabe 6 (Traveling Weihnachtsmann)

Jedes Jahr steht der Weihnachtsmann vor folgendem Problem: Er hat eine Liste von n Häusern, zu denen er Geschenke bringen muss. Des Weiteren hat er eine Tabelle (ein zweidimensionales Feld `int[][] a`), in denen die Distanzen zwischen je zwei Häusern vermerkt sind: Dabei gibt `a[i][j]` die Entfernung zwischen dem Haus `i` und dem Haus `j` an ($\texttt{i}, \texttt{j} \in \{0, 1, \ldots n-1\}$).
Da der Weihnachtsmann seine Arbeit natürlich schnellstmöglich verrichten will, ist er an einer *optimale Rundreise* interessiert: Eine Rundreise gibt dabei eine Reihenfolge aller zu besuchenden Häuser an (also eine Permutation von $\{0, 1, \ldots, n-1\}$). Wie der Name Rundreise bereits andeutet, muss der Weihnachtsmann nach Besuch des letzten Hauses wieder zum ersten Haus zurückkehren. Eine optimale Rundreise ist zudem eine solche Rundreise, welche unter allen möglichen Rundreisen die kürzeste *Gesamtdistanz* (Summe der Einzeldistanzen) aufweist.
Schreiben Sie für den Weihnachtsmann ein Java-Programm, welches dieses Problem löst. Zur Verifikation Ihrer Lösung finden Sie im Moodle und der WebIDE eine Beispieltabelle `twm.txt`. Die Gesamtdistanz einer optimalen Rundreise ist dabei 100.

Hinweise:

- Sie dürfen davon ausgehen, dass die Tabelle symmetrisch ist, d.h. dass die Richtung zwischen zwei Häusern keine Rolle für die Distanz spielt (`a[i][j] = a[j][i]`).
- Ohne geschickte Implementierung wird Ihr Algorithmus die Berechnung nicht mehr vor Weihnachten abschließen ... und das wäre doch gar nicht gut.
- Zum Einlesen der Tabelle können Sie die Klasse `pua.FileUtils` verwenden:

```
1 import pua.*;
2 ...
3 String path = FileUtils.getFilePath(); // in der WebIDE einfach direkt einen Pfad angeben
4 int[][] distances = FileUtils.readIntMatrix(path);
```