Informatik II (ITET)
SS 2007
Prof. Dr. F. Mattern, Dr. K. Römer
Silvia Santini (santinis@inf.ethz.ch)

ETH Zürich
Institut für Pervasive Computing
Verteilte Systeme
http://www.inf.ethz.ch/vs

Übungsserie Nr. 8

Ausgabe: 16. Mai 2007
Abgabe: 23./24. Mai 2007

1. Aufgabe: (10 Punkte) Rucksackproblem und Backtracking

Ein Dieb ist in ein Haus eingebrochen und steht nun vor dem Problem, unter den K wertvollen Gegenständen $x_1, \ldots, x_k$ eine Auswahl treffen zu müssen, so dass das zulässige Gesamtgewicht G seines Diebessackes nicht überschritten wird. Gleichzeitig soll der Wert aller eingepackten Objekte möglichst gross sein. Der Dieb kennt von jedem Gegenstand x_i dessen Wert w_i sowie dessen Gewicht g_i.

(1a) *(0 Punkte)* Wie viele Teilmengen hat eine Menge mit K Elementen?

(1b) *(2 Punkte)* Implementieren Sie ein einfaches Verfahren, welches in jedem Fall eine Menge von Gegenständen mit maximalem Gesamtwert liefert, ohne dass der Sack reisst. Inkrementieren Sie dafür einen Zähler n von 0 bis $2^K - 1$ und interpretieren Sie die jeweilige Darstellung von n als Bitstring der Länge k $(b_1, \ldots, b_k)$, wobei ein gesetztes Bit i anzeigt, dass der Gegenstand X_i eingepackt werden soll. Für jede Kombination soll geprüft werden, ob diese optimal sein könnte oder nicht. Wenn alle Kombinationen durchgelaufen worden sind, muss die gefundene optimale Kombination zurückgegeben werden.

(1c) *(1 Punkt)* Beantworten Sie die folgenden Fragen für den in Teilaufgabe *(1b)* implementierten Algorithmus:

(i) Liefert das Verfahren immer das "optimale" Ergebnis? Begründen Sie Ihre Antwort.

(ii) Gibt es immer genau *eine* optimale Lösung?

(1d) *(2 Punkte)* Geben Sie eine Funktion an, die den Zeitaufwand des Verfahrens aus Teilaufgabe *(1b)* in Abhängigkeit von der Problemgrösse k beschreibt.

(1e) *(4 Punkte)* Ein k-Tupel $(s_0, \ldots, s_{k-1})$ mit $s_i \in \{0, 1\}, 0 \leq i < k$ beschreibt, welche Gegenstände eingepackt werden sollen und welche nicht: Gegenstand X_j soll eingepackt werden genau dann, wenn $s_j = 1$ ist.

Geben Sie einen Algorithmus an, der eine Selektion ausgibt, für die gilt

$$\sum_{i=0}^{k-1} s_i w_i \text{ ist maximal, und } \sum_{i=0}^{k-1} s_i g_i \leq G$$

Implementieren Sie den Algorithmus als (rekursives) Java-Programm unter Verwendung von Backtracking. Geben Sie nach jedem Schritt eine Teillösung aus, wenn diese das bisher gefundene Optimum übertrifft.

(1f) *(1 Punkt)* Schreiben Sie eine geeignete `main` Methode, welche eine optimale Lösung und die oben genannten Teillösungen für das folgende Rucksack-Problem ausgibt:

$K = 8$
$G = 15$
$(g_0, \ldots, g_8) = (4, 7, 9, 2, 6, 1, 5, 8)$
$(w_0, \ldots, w_8) = (12, 15, 11, 10, 17, 16, 19, 11)$

Die Lösung bzw. die Teillösungen, müssen im folgenden Format:

$(s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8)$`, Gesamtgewicht, Gesamtwert`

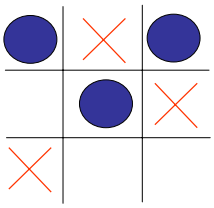
ausgegeben werden.

2. Aufgabe: (5 Punkte) Tic-Tac-Toe-Spielbaum

Betrachten Sie folgende Situation: Sie spielen mit jemandem ein strategisches Spiel wie z.B. Schach, Dame, Mühle, Go, usw. Das Spiel ist endlich, nicht vom Zufall abhängig und Sie und Ihr Gegner überblicken zu jedem Zeitpunkt die gesamte Spielsituation, d.h. Sie können alle Zugmöglichkeiten Ihres Gegners erkennen und umgekehrt. In jeder Spielsituation gibt es nur endlich viele Zugmöglichkeiten und in einer Endsituation kann eindeutig festgestellt werden, wer gewonnen hat (bzw. ob das Ergebnis "unentschieden" ist). Sie und Ihr Gegner wollen natürlich beide gewinnen.

Hilfreich für die Analyse solcher Spiele sind sogenannte *Spielbäume*. Die Wurzel steht dabei für die aktuelle Spielsituation, in der Spieler A am Zug ist. Jeder zulässige Zug für Spieler A führt zu einer neuen Situation, der als Kindknoten der Wurzel dargestellt wird. Nun ist Spieler B an der Reihe, dessen Möglichkeiten wiederum zu verschiedenen Situationen führen usw.

Betrachten Sie die folgende Situation eines Tic-Tac-Toe-Spiels auf einem 3x3-Feld [1]:



Dabei sind Ihre Spielzüge jeweils mit $\times$, die Ihres Gegners mit $\bullet$ eingezeichnet. Sie sind am Zug.

[1] Falls Sie das Spiel nicht kennen, finden Sie die Regeln z.B. bei: `http://de.wikipedia.org/wiki/Tic-Tac-Toe` oder `http://www.msoworld.com/Games/info/ttt-info.html`.

(2a) *(2 Punkte)* Zeichnen Sie einen Spielbaum für den restlichen Verlauf dieses Tic-Tac-Toe-Spiels. Markieren Sie dabei jeden Knoten mit der Spielsituation und mit dem Namen desjenigen, der an der Reihe ist.

(2b) *(2 Punkte)* Der Baum soll nun so von unten nach oben markiert werden, dass Sie wissen, welche Alternativen zu welchem Spielergebnis für Sie führen. Dabei sollen die Attribute 1 (Sieg), 0 (Unentschieden) und -1 (Niederlage) verwendet werden. Überlegen Sie sich, wie das Attribut eines Knotens aufgrund der Attribute der Nachfolger berechnet wird, wenn Sie an der Reihe sind, bzw. wenn Ihr Gegner an der Reihe ist (vgl. Sie dazu Skript S.177).

(2c) *(1 Punkt)* Welchen Schluss ziehen Sie aus dem von Ihnen markierten Baum für Ihren Zug aus der oben gegebenen Spielsituation?

3. Aufgabe: (8 Punkte) Reversi (Teil 2)

Laden Sie sich die aktuelle Version der Reversi-Software und Dokumentation von der Vorlesungs-Webseite[2] herunter und überschreiben Sie ggf. die Verzeichnisse *reversi*, *humanPlayer* und *javadoc* in ihrer alten Version. Nun stehen Ihnen alle Source-Files und die komplette Dokumentation zur Verfügung.

Ihr Reversi-Computerspieler soll in dieser Aufgabe eine gegebene Spielsituation bewerten können, um z.B. aus der Menge der möglichen Züge den besten auszuwählen.

(3a) *(3 Punkte)* Überlegen Sie sich, wie Sie eine solche Bewertung durchführen würden. Erläutern Sie Ihre Idee für die Implementierung Ihrer Bewertungsfunktion mittels Pseudocode, und geben Sie diesen bei Ihrem Tutor ab.

(3b) *(5 Punkte)* Erstellen Sie einen neuen Computerspieler (angelehnt an den zufällig handelnden Spieler aus Aufgabe 2 des Übungsblatts 7), sodass die Methode `nextMove()` den laut Bewertungsfunktion besten Zug zurückliefert. Dabei sollten Sie beachten:

- In dieser Aufgabe sollen Sie nur den *nächsten* Zug betrachten und noch keinen Baum aufbauen. Sie sollen jeden (aus der gegenwärtigen Spielsituation möglichen) Zug "simulieren" und die neue Spielsituation dann bewerten.
- Um verschiedene mögliche Züge nacheinander auswerten zu können, müssen Sie sich die Ausgangssituation des Spielbretts merken (bzw. in einem geeigneten Objekt speichern).
- Für das Speichern des Spielbretts und dem Ausführen von Zügen können Sie teilweise existierende Funktionalität des Reversi-Paketes mitbenutzen. Schauen Sie dazu in die Dokumentation. Allerdings ist es Ihnen freigestellt, wie Sie ihr Programm entwickeln!
- Evaluieren Sie die verschiedenen möglichen neuen Spielsituationen mit der Methode aus Teilaufgabe a).

[2] `http://www.vs.inf.ethz.ch/edu/i2/reversi/`

(3c) *(keine Bewertung)* Wenn Sie wollen, können Sie nun ihren Computerspieler gegen den "zufälligen" Spieler des letzten Übungsblattes antreten lassen, und schauen, ob er wirklich besser geworden ist!

WICHTIG:
Vergessen Sie nicht, Teilfunktionalitäten zwischendurch ausgiebig zu testen, z.B. durch geeignete Text-Ausgaben. So lässt sich die Aufgabe auch leichter handhaben. Bitte vergewissern Sie sich, dass man Ihre Lösung leicht ausführen kann. Geben Sie in der E-Mail die zugehörige Kommandozeile mit an.