Universität Bremen

Praktische Informatik 3 WS 14/15

11. Übungsblatt

Ausgabe: 13.01.15
Abgabe: 23.01.15

Christoph Lüth
Sandor Herms
Daniel Müller
Jan Radtke
Henrik Reichmann
Sören Schulze
Felix Thielke

11.1 *Schiebung!* *20 Punkte*

In diesem Übungsblatt wollen wir Haskell mit sich selbst beschäftigen. Es soll "Schiebepuzzle" lösen, in denen sich $n^2 - 1$ Puzzlesteine auf einem $n \times n$-Feld befinden. Sie können verschoben werden, indem eines der (höchstens) vier umliegenden Steine auf das freie Feld geschoben werden. Ein Puzzle befindet sich einem Startzustand, und muss dann durch Verschieben der Steine in einen definierten Zielzustand gebracht werden (siehe Abb. 1). Eine Lösung des Puzzles ist also eine Sequenz von Verschiebeoperationen.

L	O	I
A	M	D
E	N	

2	5	4	8
1	3		11
9	6	10	7
13	14	15	12

Abbildung 1: Zwei Schiebepuzzles. Links ergeben die Buchstaben auf dem Feld im Zielzustand ein Wort, rechts die Zahlen von 1 bis 15 in aufsteigender Reihenfolge.

1. Modellieren Sie das Spielfeld geeignet als einen Datentyp `Board a` (die Typvariable sind die Puzzlesteine, in Abb. 1 links beispielsweise `Char`), und die Bewegung des freien Feldes als vierelementigen Aufzählungstyp `Dir`.
2. Schreiben Sie eine Funktion `solve`, welche einen gewünschen Zielzustand des Feldes, einen momentanen Startzustand und ggf. weitere Parameter wie die Größe des Feldes als Argumente nimmt, und als Rückgabe eine Liste `[Dir]` von Bewegungen zurückgibt, welche den momentanen Zustand in den Zielzustand überführen, d.h. das Puzzle lösen.
3. Erweitern Sie Ihr Programm, so dass es mit einem Puzzle in geeigneter Form aus der Kommandozeile aufgerufen werden kann. Als Argumente sollen ein Dateiname übergeben werden, in dem das Puzzle in geeigneter textueller Repräsentation steht.
4. Erweiteren Sie das Programm weiter, so dass es die Lösung ansprechend ausgibt.
5. Nicht alle möglichen Anordnungen sind lösbare Puzzles. Schreiben Sie ein zweites Programm, welches Puzzles *generiert*, indem es eine Liste von Wörtern liest, Wörter der geeigneten Länge (die Größe n des Spielfeldes soll ein Parameter sein) herausfiltert, und aus diesen eines zufällig herauswählt. Dieses Wort dient als Zielzustand; aus ihm soll durch eine parametrisierbare Anzahl von zulässigen Bewegungen der Startzustand des Puzzles erzeugt werden.
6. Sie können jetzt Ihren Puzzlelöser mit dem Puzzlegenerator testen. Bis zu welcher Länge des Lösungsweges kann Ihr Löser das Puzzle knacken?

Hinweise: Ein möglicher Lösungsansatz ust, ausgehend vom momentanen Zustand einen Suchgraphen aufzubauen, dessen Kanten mögliche Züge sind, und dessen Knoten die entstehenden Brettzustände sind. Kern der Suche ist eine geeignete Bewertungsfunktion, beispielsweise indem gezählt wird, wie weit jeder Stein von seiner Zielposition entfernt ist. Gesucht wird, indem von dem momentanen Brettzustand alle Nachfolger berechnet, deren Bewertung ermittelt, und sie mit dieser Bewertung in eine Vorrangwarteschlange eingefügt werden. Danach wird mit dem ersten Element der Vorrangwarteschlange die Suche rekursiv fortgesetzt.

Punkteverteilung:

- Lösung eines Puzzles *12 Punkte*
- Aufruf aus der Kommandozeile, ansprechende Darstellung des Lösungsweges *3 Punkte*
- Generierung von Puzzles *5 Punkte*

? *Verständnisfragen*

1. Warum sind endrekursive Funktionen im allgemeinen schneller als nicht-endrekursive Funktionen? Unter welchen Voraussetzungen kann ich eine Funktion in endrekursive Form überführen?
2. Ist eine in allen Argumenten als strikt erkannte und übersetzte Funktion immer schneller in der Ausführung als dieselbe als nicht-strikt übersetzte Funktion?
3. Warum kann ich die Funktion `seq` nicht in Haskell definieren?