Einführung in die Informatik 2

Prof. Dr. Seidl, J. Kranz, N. Hartmann, J. Brunner
WS 2016/17

Übungsblatt 10
Abgabefrist: 16.1.2017

Aufgabe 10.1 (P) Module

Diskutieren Sie Module und Signaturen von Modulen in der Gruppe. Besprechen Sie dabei insbesondere folgende Themen:

- Wozu dienen Module (entsprechend dem, was bisher über Module in der Vorlesung gelernt wurde)?
- Wieso definiert man Module und Signaturen oft getrennt?
- Wieso kann es sinnvoll sein, Typen in Signaturen abstrakt zu lassen und welche Konsequenzen hat es?
- Welche Parallelen zwischen Modulen und bekannten Konzepten, insbesondere in Java, gibt es?

Aufgabe 10.2 (P) Sortierhaufen

In dieser Aufgabe dürfen Sie *HeapSort* implementieren. Es sei zunächst die folgende Signatur für Haufen[1] gegeben:

```
module type Heap = sig
  type 'a t
  exception Empty

  val create : unit -> 'a t
  val is_empty : 'a t -> bool
  val insert : 'a -> 'a t -> 'a t
  val delete_max : 'a t -> ('a * 'a t)
end
```

Wir wollen als Grundlage für unseren Haufen einen binären Baum verwenden. Es sei folgender Datentyp zur Repräsentation binärer Bäume definiert:

```
type 'a btree =
   Leaf
 | Node of int * 'a * 'a btree * 'a btree
```

Ein binärer Baum ist genau dann ein Haufen, wenn für jeden Knoten gilt, dass sein Wert größer oder gleich aller Werte in den Teilbäumen ist (Haufen-Eigenschaft). Insbesondere ist der größte Wert in der Wurzel gespeichert. Zusätzlich wird gefordert, dass der in dem Knoten gespeicherte Integer-Wert die Länge des kürzesten Pfades zu einem Blatt ist. Beispielsweise ist `Node (1, 'c', Node (1, 'a', Leaf, Leaf), Leaf)` ein Haufen.

Gehen Sie nun bei der Implementierung wie folgt vor.

[1] Es wird in dieser Aufgabe zur besseren Verständlichkeit für deutschsprachige Studenten konsequent von *Haufen* gesprochen.

1. Implementieren Sie die Funktion `insert`, sodass der Aufruf `insert x h` den Wert `x` in den Haufen `h` einfügt. Dabei soll sowohl darauf geachtet werden, die Haufen-Eigenschaft nicht zu verletzen, als auch die Tiefe des Haufens, falls möglich, nicht zu erhöhen. Ein Wert `x` wird wie folgt in den Haufen eingefügt: Zunächst wird `x` mit dem Wert `w` an der Wurzel des Haufens verglichen. Der größere der beiden Werte wird der neue Wert der Wurzel, während der kleinere in einen Teil-Haufen eingefügt wird. Um eine Degeneration zu verhindern, ist dabei derjenige Teil-Haufen auszuwählen, der den kürzesten Pfad zu einem Blatt hat. Beispielsweise soll der Aufruf durch Einfügen von `'b'` in `(Node (1, 'c', Node (1, 'a', Leaf, Leaf), Leaf))` ein Haufen `Node (2, 'c', Node (1, 'a', Leaf, Leaf), Node(1, 'b', Leaf, Leaf))` entstehen. Wichtig dabei ist, dass das Element `'b'` in den kleineren Teil-Haufen eingefügt worden ist.
2. Implementieren Sie die Funktion `delete_max`, die ein Element aus dem Haufen löscht, indem sie ein Paar bestehend aus dem Wert der Wurzel und einem aus den verbleibenden Elementen bestehenden Haufen liefert. Dazu kann es sinnvoll sein, eine Funktion zu schreiben, die zwei Haufen zu einem vereinigt. Beispielsweise soll aus dem Haufen `Node (2, 'c', Node (1, 'a', Leaf, Leaf), Node (1, 'b', Leaf, Leaf))` das Tupel `('c', Node(1, 'b', Node (1, 'a', Leaf, Leaf), Leaf))` entstehen.
3. Verwenden Sie Ihre Implementierung zur Realisierung des HeapSort-Algorithmus.
4. Testen Sie Ihre Implementierung!

Lösungsvorschlag 10.2

Die Lösung befindet sich in der Datei `ocaml/p10_sol.ml`.

Aufgabe 10.3 (P) Primfaktoren

Schreiben Sie eine Funktion `val factors : int -> int list`, die zu einer gegebenen Zahl eine Liste ihrer Primfaktoren in aufsteigender Ordnung zurückliefert.

Betrachten Sie folgendes Beispiel:

```
# factors 315;;
- : int list = [3; 3; 5; 7]
```

Hinweis: d dividiert n gdw. $n \ mod \ d = 0$.

Allgemeine Hinweise zur Hausaufgabenabgabe

Die Hausaufgabenabgabe bezüglich dieses Blattes erfolgt über das Ocaml-Abgabesystem. Sie erreichen es unter `https://vmnipkow3.in.tum.de`. Sie können hier Ihre Abgaben einstellen und erhalten Feedback, welche Tests zu welchen Aufgaben erfolgreich durchlaufen. Sie können Ihre Abgabe beliebig oft testen lassen. Bitte beachten Sie, dass Sie Ihre Abgabe stets als **eine einzige UTF8-kodierte Textdatei mit dem Namen *ha10.ml* hochladen müssen**. Die hochgeladene Datei muss ferner den Signaturen in *ha10.mli* entsprechen.

Aufgabe 10.4 (H) Modulare Wörterbücher [12 Punkte]

In dieser Aufgabe soll das Implementieren von Modulsignaturen geübt werden. Dazu ist die folgende Signatur für Wörterbücher (*Maps*) gegeben:

```
1 module type Map = sig
2   type ('k, 'v) t
3
4   val create : 'k key_info -> ('k, 'v) t
5   val size : ('k, 'v) t -> int
6   val insert : ('k, 'v) t -> 'k -> 'v -> ('k, 'v) t
7   val remove : ('k, 'v) t -> 'k -> ('k, 'v) t
8 end
```

Die Funktionen haben dabei jeweils folgende Bedeutung:

- `create`: Erzeugt ein neues Wörterbuch; die übergebene `key_info` enthält dabei nötige Operationen auf den Schlüsseln.
- `size`: Gibt die Anzahl der Elemente im Wörterbuch zurück
- `insert`: Fügt ein neues Mapping in das Wörterbuch ein; sollte zum übergebenen Schlüssel bereits ein Eintrag vorhanden sein, so soll das jeweilige Datum ersetzt werden.
- `remove`: Entfernt ein Element aus dem Wörterbuch; ist kein passendes Element vorhanden, soll nichts passieren (insbesondere soll kein Fehler erzeugt werden).

Bearbeiten Sie die folgenden Teilaufgaben.

1. Implementieren Sie das Modul `HashMap`, welches die obige Signatur verwendet. Nutzen Sie für Ihre Implementierung *Hashing with Chaining*. Verwenden Sie als Repräsentation der Hashtabelle ein Array[2] der Größe 37. Ihre Datenstruktur soll *veränderbar* sein (das Einfügen und Löschen von Elementen erzeugt keine neue Hashtabelle).
2. Implementieren Sie das Modul `TreeMap`, welches die obige Signatur verwendet. Die Daten sollen hier in einem (unbalancierten) binären Buchbaum gespeichert werden. Nutzen Sie den in `ha10_angabe.ml` gegebenen Typen `bin_tree` sowie die ebenfalls gegebenen Funktionen `insert` und `remove` für Ihr Modul.

[2] `https://realworldocaml.org/v1/en/html/imperative-programming-1.html`

Lösungsvorschlag 10.4

Die Lösung befindet sich in der Datei `ocaml/ha10_sol.ml`.

Korrekturhinweise:

- Die Tests prüfen nicht, ob die `HashMap` änderbar, die `TreeMap` allerdings nicht änderbar ist (2 Punkte Abzug wenn nicht).
- Die Tests prüfen nicht, ob die Werte zu Schlüsseln richtig gesetzt und aktualisiert werden (2 Punkte).

Punkteverteilung (jeweils pro Map):

1. 1 Punkt
2. 1 Punkt
3. 2 Punkte
4. 2 Punkte

Aufgabe 10.5 (H) Karge Vektoren [8 Punkte]

In dieser Aufgabe soll ein Datentyp für Vektoren und Operationen auf diesen implementiert werden. Analog zu dünn besetzten Matrizen, enthalten dünn besetzte Vektoren hauptsächlich Nullen, welche man nicht speichern will. Wir definieren uns daher einen Datentyp, der den Index (beginnend mit 0) und den Wert an dieser Position als Liste speichert.

Als Invariante soll gelten, dass nach jeder Operation nur Werte ungleich 0 in der Datenstruktur enthalten sind. Also evaluiert z.B. sowohl `add [1,1] [1,-1]`, als auch `set 3 0 [3,1]` zu `[]`.

Gegeben sei die folgende Signatur für Vektoren:

```
module type Vector = sig
  type t

  val empty : t
  val set : int -> int -> t -> t
  val add : t -> t -> t
  val mul : int -> t -> t
  val sprod : t -> t -> int
end
```

Implementieren Sie ein Modul `SparseVector` entsprechend obiger Signatur. Die einzelnen Funktionen haben dabei folgende Bedeutung:

- `empty`: Gibt einen leeren Vektor zurück
- `set i v a`: Setzt den Wert an Stelle i des Vektors a auf den Wert v

- `add a b`: Addiert durch komponentenweise Summe: $\vec{a} + \vec{b} = \begin{pmatrix} a_1 \\ a_2 \\ \dots \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \dots \end{pmatrix} = \begin{pmatrix} a_1 + b_1 \\ a_2 + b_2 \\ \dots \end{pmatrix}$
- `mul r a`: Multipliziert mit einem Skalar: $r\vec{a} = \begin{pmatrix} ra_1 \\ ra_2 \\ \dots \end{pmatrix}$
- `sprod a b`: Berechnet das Skalarprodukt: $\langle \vec{a}, \vec{b} \rangle = \sum_{i=1}^{n} a_i b_i$

Lösungsvorschlag 10.5
Die Lösung befindet sich in der Datei `ocaml/ha10_sol.ml`.

Korrekturhinweise:

- Die Tests prüfen nicht, ob 0er im Vektor vermieden werden (2 Punkte)

Punkteverteilung:

1. 1 Punkt (`set`)
2. 2 Punkte (`add`)
3. 2 Punkte (`mul`)
4. 3 Punkte (`sprod`)