Technische Universität München
Fakultät für Informatik
Prof. Dr. H. Seidl

SoS 2005
Übungsblatt 8
17. Juni 2005

Übungen zu Einführung in die Informatik II

Aufgabe 17 Hash-Tabellen

In dieser Aufgabe soll, ähnlich zur Aufgabe 30 (Blatt 10) aus dem vergangenen Semester, eine *Hash-Map* implementiert werden. Als Implementierungssprache soll hier jedoch OCaml statt Java verwendet werden.

a) Definieren Sie einen Datentyp `('a, 'b) hash_map` zur Repräsentation von *Hash-Maps*. Eine *Hash-Map* ordnet einem Schlüssel vom Typ `'a` einen Wert vom Typ `'b` mit Hilfe einer *Hash-Tabelle* zu. In einer *Hash-Map* soll zudem die Anzahl der in der *Hash-Map* enthaltenen Zuordnungen sowie die Funktion zur Berechnung des Hash-Wertes eines Wertes vom Typ `'a` gespeichert werden.

   Für den Datentyp `('a, 'b) hash_map` sollen folgende Funktionen definiert werden:

   - Eine Funktion `init`, die eine leere *Hash-Map* erzeugt.
   - Eine Funktion `put : ('a, 'b) hash_map -> 'a * 'b -> bool`, die als Argumente eine *Hash-Map* `hm` sowie ein als Zuordnung aufgefasstes Paar `(k,v)` erhält. Ein Aufruf dieser Funktion soll die Zuordnung `(a,b)` der *Hash-Map* hinzufügen. Ist dem Schlüssel `a` bereits ein Wert zugeordnet, so soll dieser Überschieben werden. Zurückliefern soll die Funktion `true` genau dann, wenn die Anzahl der Zuordnungen in `hm` größer geworden ist.
   - Eine Funktion `list2hash_map` die zu einer Liste vom Typ `('a * 'b) list` eine *Hash-Map* vom Typ `('a, 'b) hash_map` erzeugt.
   - Eine Funktion `get : ('a, 'b) hash_map -> 'a -> 'b option`, die, falls dem übergebenen Schlüssel ein Wert `v` zugeordnet ist, `Some v` zurückliefert. Andernfalls soll `None` zurückgeliefert werden.
   - Eine Funktion `remove : ('a, 'b) hash_map -> 'a -> 'b option`, die, genau wie die Funktion `get`, den zugeordneten Wert zurückliefert und zusätzlich die Zuordnung aus der *Hash-Map* entfernt.
   - Eine Funktion `iter : ('a * 'b -> unit) -> ('a, 'b) hash_map -> unit`, die als Argumente eine Funktion `f : ('a * 'b -> unit)` und eine *Hash-Map* `hm` erhält und dann die Funktion `f` auf alle Zuordnungen der *Hash-Map* anwendet.

b) Erweitern Sie Ihre Implementation um die Unterstützung eines *Load-Faktors* `lf`. Nimmt die Anzahl der in der *Hash-Map* enthaltenen Zuordnungen zu, so soll überprüft werden, ob diese Anzahl multipliziert dem dem *Load-Faktor* größer als die Größe der *Hash-Tabelle* ist. Ist dies der Fall, so soll ein *Rehashing* angestoßen werden, dass die Größe der *Hash-Tabelle* verdoppelt.

Aufgabe 18 Berechnung von Binomialkoeffizienten

Der Binomialkoeffizient $\binom{n}{k}$ läßt sich mit Hilfe der Formeln $\binom{n}{k} = \binom{n}{n-k}$ und $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$ effizient rekursiv berechnen.

Definieren Sie eine Funktion `bin : int * int -> int`, die den Binomialkoeffizienten berechnet. Verwenden Sie den in der vorherigen Aufgabe definierten Datentyp `hash_map`, um mehrfache Berechnungen des gleichen Binomialkoeffizienten zu vermeiden.