Laboratorul 9

Introducere

In acest laborator vom exersa manipularea listelor si tipurilor de date prin implementarea catorva colectii de tip tabela asociativa cheie-valoare.

Aceste colectii vor trebui sa aiba urmatoarele facilitati

- crearea unei colectii vide
- crearea unei colectii cu un element
- adaugarea/actualizarea unui element intr-o colectie
- cautarea unui element intr-o colectie
- stergerea (marcarea ca sters a) unui element dintr-o colectie
- obtinerea listei cheilor
- obtinerea listei valorilor
- obtinerea listei elementelor

```
import           Prelude hiding (lookup)
import qualified Data.List as List
```

Clasa Collection

```
class Collection c where
  empty :: c key value
  singleton :: key -> value -> c key value
  insert
      :: Ord key
      => key -> value -> c key value -> c key value
  lookup :: Ord key => key -> c key value -> Maybe value
  delete :: Ord key => key -> c key value -> c key value
  keys :: c key value -> [key]
  values :: c key value -> [value]
  toList :: c key value -> [(key, value)]
  fromList :: Ord key => [(key,value)] -> c key value
```

Exercitiul 0

Adaugati definitii implicite (in functie de functiile celelalte) pentru

a. `keys`
b. `values`
c. `fromList`

Exercitiul 1

Fie tipul listelor de perechi de forma cheie-valoare:

```
newtype PairList k v
  = PairList { getPairList :: [(k, v)] }
```

Faceti `PairList` instanta a clasei `Collection`.

Exercitiul 2

Fie tipul arborilor binari de cautare (ne-echilibrati):

```
data SearchTree key value
  = Empty
  | Node
      (SearchTree key value) -- elemente cu cheia mai mica
      key                    -- cheia elementului
      (Maybe value)          -- valoarea elementului
      (SearchTree key value) -- elemente cu cheia mai mare
```

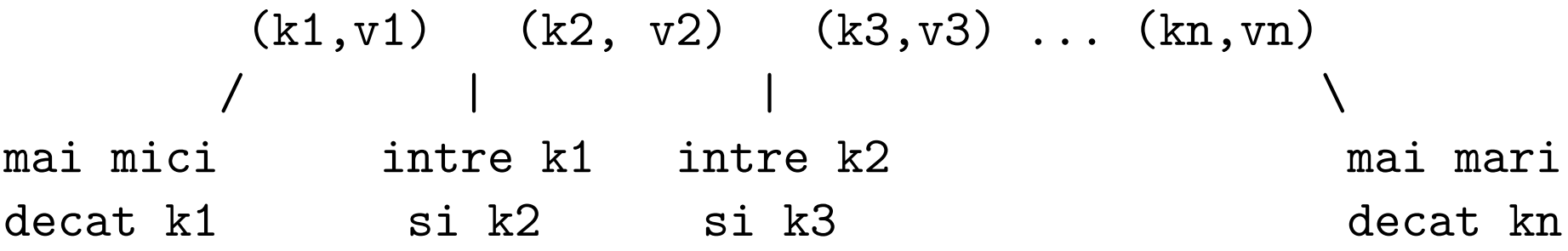
Observati ca tipul valorilor este `Maybe value`. Acest lucru se face pentru a reduce timpul operatiei de stergere prin simpla marcare a unui nod ca fiind sters. Un nod sters va avea valoarea `Nothing`.

Faceti `SearchTree` instanta a clasei `Collection`.

B-Arbori

B-arbori: Introducere

B-arborii sunt o generalizare a abrorilor (echilibrati) de cautare in care un nod poate contine mai multe elemente, si intre oricare doua elemente se gaseste un subarbore care contine elemente cu cheile intre elementele intre care se afla subarborele.



Dat fiind un parametru `d`, un B-arbore de ordin `2*d+1` are urmatoarele proprietati:

- Este echilibrat: toate frunzele se afla la aceeasi adancime
- Orice nod are cel mult `2*d` elemente (si `2*d+1` subarbori)
- Orice nod in afara de radacina are cel putin `d` elemente

```
order = 1
```

B-arbori in Haskell: Design

In continuarea acestui laborator vom incerca sa implementam o versiune de B-arbori in Haskell.

Pentru a reprezenta un nod, am putea sa separam elementele de subarbori, dar este mult mai convenabil sa le imperechem. O idee simpla e sa imperechem fiecare subarbore cu elementul imediat in dreapta lui.

Dar ce vom face cu ultimul subarbore? O idee ar fi sa creem un nou element artificial care sa fie mai "mare" decat orice alt element. Atunci ar arata cam asa:

```
     (k1,v1)   (k2, v2)  ... (kn,vn)      OverLimit
    /         /              /           /
mai mici   intre k1      intre kn-1   mai mari
decat k1    si k2          si kn       decat kn
```

Exercitiul 3

Definim un tip de date al elementelor (extinse cu `Overlimit`):

```
data Element k v
  = Element k (Maybe v)
  | OverLimit
```

a. Faceti `Element` instanta a clasei de tipuri `Eq` astfel incat constructorii `Element` sunt egali oricand cheile sunt egale.

b. Faceti `Element` instanta a clasei de tipuri `Ord` astfel incat compararea constructorilor `Element` sa se reduca la compararea cheilor, si `OverLimit` sa fie cel mai mare element.

c. Faceti `Element` instanta a clasei de tipuri `Show` astfel incat `Element k (Just v)` sa fie afisat ca perechea `(k,v)`, iar `Element k Nothing` sa fie `k`, iar `Overlimit` sa fie sirul vid.

Exercitiul 4

Fie urmatorul tip de date care defineste B-arbori:

```
data BTree key value
  = BEmpty
  | BNode [(BTree key value, Element key value)]
```

Faceti `BTree` instanta a clasei `Collection`. Observatii:

- Cautarea unei chei / a locului unei chei `k`:
  - Se cauta in lista de perechi (subarbore,element)
  - Pana cand se gaseste un element cu cheia `>= k`
  - Daca nu e egala se repeta procesul in subarborele corespunzator
- Inserarea unui nou element
  - Se cauta cheia ca mai sus; Daca e gasita inlocuim valoarea
  - Daca nu e gasita, este creat un arbore singleton si e *marcat*
  - La intoarcere, daca subarborele procesat e *marcat*
    - el va fi de forma `ss-(k, v)-sd` (nod cu un singur element)
    - Se "sparge" perechea curenta `(a,e)` in `(ss, (k,v)) (sd, e)`
    - daca numarul de elemente a ajuns 2*ordin+1 se sparge in doua; devine de forma `ss-(k,v)-sd` si e *marcat*