

## Laboratorul 11

# Introducere

În acest laborator vom exersa manipularea listelor și tipurilor de date prin implementarea catorva colecții de tip tabel asociativ cheie-valoare.

Aceste colecții vor trebui să aibă următoarele facilități

- ▶ crearea unei colecții vide
- ▶ crearea unei colecții cu un element
- ▶ adăugarea/actualizarea unui element într-o colecție
- ▶ căutarea unui element într-o colecție
- ▶ ștergerea (marcarea ca șters a) unui element dintr-o colecție
- ▶ obținerea listei cheilor
- ▶ obținerea listei valorilor
- ▶ obținerea 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

1. `keys`
2. `values`
3. `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
```

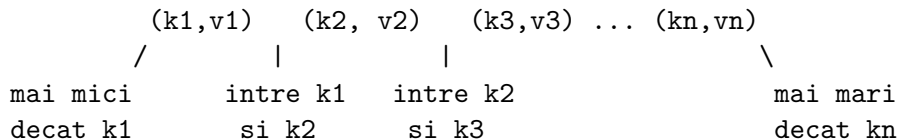
Observati ca tipul valorilor este `Maybe value`. Acest lucru se face pentru a reduce timpul operatiei de stergere prin simpla marcare a unui nod ca fiind sters. Un nod sters va avea valoarea `Nothing`.

Faceti `SearchTree` instantă a clasei `Collection`.

B-Arbori

## B-arbori: Introducere

B-arborii sunt o generalizare a arborilor (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 subarboarele.



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:

$(k_1, v_1)$	$(k_2, v_2)$	$\dots (k_n, v_n)$	OverLimit
/	/	/	/
mai mici decat $k_1$	intre $k_1$ si $k_2$	intre $k_{n-1}$ si $k_n$	mai mari decat $k_n$

## Exercitiul 3

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

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

1. Faceti `Element` instanta a clasei de tipuri `Eq` astfel incat constructorii `Element` sunt egali oricand cheile sunt egale.
2. 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.
3. Faceti `Element` instanta a clasei de tipuri `Show` astfel incat `Element k (Just v)` sa fie afisat ca perechea  $(k,v)$ , iar `Element k` 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 instantă a clasei Collection. Observatii:

- ▶ Cautarea unei chei / a locului unei chei  $k$ :
  - ▶ Se caută în lista de perechi (subarbore, element)
  - ▶ Până când se găsește un element cu cheia  $\geq k$
  - ▶ Dacă nu e egală se repetă procesul în subarborii corespunzător
- ▶ Inserarea unui nou element
  - ▶ Se caută cheia ca mai sus; Dacă e găsită înlocuim valoarea
  - ▶ Dacă nu e găsită, este creat un arbore singleton și e *marcat*
  - ▶ La întoarcere, dacă subarborii procesați sunt *marcați*
    - ▶ el va fi de forma  $ss-(k, v)-sd$  (nod cu un singur element)
    - ▶ Se “sparge” perechea curentă  $(a, e)$  în  $(ss, (k, v))$  ( $sd, e$ )
    - ▶ dacă numărul de elemente a ajuns  $2 \cdot \text{ordin} + 1$  se sparge în două; devine de forma  $ss-(k, v)-sd$  și e *marcat*