Table of Contents

Standard Collections

Benson Joeris
http://bjoeris.com

Overview

- Functional collections
- Set
- Map
- Seq
- Vector

Table of Contents

Functional Collections

```haskell
data List a = Empty | Cons a (List a)
```

- Immutable

```haskell
updateFirst :: List a -> a -> List a
upadteFirst Empty y = Empty
updateFirst (Cons x xs) y = Cons y xs
```

Functional Collections

- Immutable

```
array = new int[10];
for(int i=0; i<10; i++)
  array[i] = i+1;
```

  - Modification $\rightarrow$ Copy (seems slow)
  - Immutable data can be shared
  - Copy $\rightarrow$ use reference to original (fast)

- Queries are no problem

Table of Contents

Set

- Unordered collection

```
import Data.Set
```

Set

```
empty :: Set a
```

```
insert :: a -> Set a -> Set a
```

```
delete :: a -> Set a -> Set a
```

```
union :: Set a -> Set a -> Set a
```

```
member :: a -> Set a -> Bool
```

Set Restrictions

- Set of functions?

```
triple :: Int -> Int
triple x = x + x + x

triple' :: Int -> Int
triple' x = 3 * x

funSet :: Set (Int -> Int)
funSet = insert triple empty

problem :: Bool
problem = member triple' funSet
```

Set Restrictions

```
class Eq a
```

```
class Ord a
```

Set Restrictions

```haskell
empty :: Set a

insert :: Ord a => a -> Set a -> Set a

delete :: Ord a => a -> Set a -> Set a

union :: Ord a => Set a -> Set a -> Set a

member :: Ord a => a -> Set a -> Bool
```

Table of Contents

Map

- Key-value pair collection

```
import qualified Data.Map
```

```
data Map k a
```

Map

```haskell
empty :: Map k a
```

```haskell
insert :: Ord k => k -> a -> Map k a -> Map k a
```

```haskell
delete :: Ord k => k -> Map k a -> Map k a
```

```haskell
union :: Ord k => Map k a -> Map k a -> Map k a
```

```haskell
lookup :: Ord k => k -> Map k a -> Maybe a
```

Table of Contents

Seq

- Ordered collection

```
import Data.Sequence
```

Seq

```haskell
empty :: Seq a
```

Seq

```
(<|) :: a -> Seq a -> Seq a
```

- Same as cons `(:)` for lists

```
(|>) :: Seq a -> a -> Seq a
```

```
(><) :: Seq a -> Seq a -> Seq a
```

Seq Pattern Matching

```
~~length :: Seq a -> Int~~
~~length empty = 0~~
~~length (x <| xs) = 1 + length xs~~
```

View Patterns

```
{-# LANGUAGE ViewPatterns #-}
```

```
:set -XViewPatterns
```

```
length :: Seq a -> Int
length (viewl ->  EmptyL) = 0
length (viewl -> x :< xs) = 1 + length xs
```

```
viewl :: Seq a -> ViewL a
```

```
data ViewL a
  = EmptyL
  | a :<  (Seq a)
```

View Patterns

```haskell
length' :: Seq a -> Int
length' (viewr -> EmptyR) = 0
length' (viewr -> xs :> x) = 1 + length xs
```

Seq Performance

- Seq usually faster than list

Table of Contents

Summary

- Purely functional collections
- Set
- Map
- Seq