Haskell Tutorial: Maps

September 27, 2019

[1]:
```
:opt no-lint
```

0.1 Maps

Haskell has support for maps that map a key to a value. Mathematically, you may think of a map as a binary relation mapping keys to values. To use the map functions, you have to import `Data.Map`. There are actually two different kinds of maps in the Haskell library: `Data.Map.Strict` provides finite maps for (sometimes called dictonaries) for situations where all values will be forced to exist. If laziness is required, i.e., you want to store something potentially infinite in a map, you should use `Data.Map.Lazy`. Haskell maps are pretty powerful. This document only provides a very basic overview. For a full description, see the official documentation.

[2]:
```
import qualified Data.Map as Map
```

A common way to create maps is to create them from a list of tuples. But there are also convenient ways to create an empty map or a map has a single member, called a *singleton*.

[3]:
```
m0 = Map.empty
m1 = Map.singleton "Eve" 42
m2 = Map.fromList $ zip ["hello", "world"] [1..]
```

The `null` function can be used to test whether a map is empty while the `size` function returns the number of elements of a map.

[4]:
```
map Map.null [m0, m1, m2]
map Map.size [m0, m1, m2]
```

```
[True,False,False]
```

```
[0,1,2]
```

Maps can be converted back to lists using the `toList`, `toAscList`, and `toDescList` functions. The `toAscList`, and `toDescList` functions return the map tuples ordered by the keys.

[5]:
```
Map.toList m2
Map.toAscList m2
Map.toDescList m2
```

```
[("hello",1),("world",2)]
```

```
[("hello",1),("world",2)]
```

```
[("world",2),("hello",1)]
```

The `elems` function returns the list of all values while the `keys` function returns the list of keys.

[6]:
```
Map.elems m2
Map.keys m2
```

```
[1,2]
```

```
["hello","world"]
```

There are many ways to lookup values for key. First, you can use the `member` function to test whether a certain key exists in the map. The `!` operator returns the value of a key that exists in a map or it throws an error. There are other functions that avoid throwing an error. A simple solution is `findWithDefault`, which returns the value of a key in a map or a default value.

[7]:
```
Map.member "hello" m2
m2 Map.! "hello"
Map.findWithDefault 0 "hello" m2
Map.findWithDefault 0 "hello" m1
```

```
True
```

```
1
```

```
1
```

```
0
```

0.1.1 TODO

- explain insert, delete, update
- explain unions, intersection, difference
- explain map, mapWithKey, filter, mapKeys

[8]:
```

```