Komprese

Vaším úkolem je implementovat funkce na kompresi a dekompresi dat pomocí algoritmu Lempel-Ziv-Welch (LZW). Tento kompresní algoritmus se používá například v datovém formátu GIF a volitelně také pro kompresi souborů PDF.

Odevzdávat budete jeden soubor s modulem `LZW`, který definuje typová synonyma

```
type Codeword = Int
type Encoding = [Codeword]
```

a exportuje následující funkce:

```
compress :: String -> Encoding
decompress :: Encoding -> String
```

Cílem úlohy je pochopit standardní algoritmus z existující literatury a implementovat ho v jazyce Haskell. Proto vás pro popis algoritmu odkážeme na anglickou Wikipedii, kde jsou komprese a dekomprese srozumitelně popsané včetně příkladů.

Jak je vidět na definovaných typech, bohatě nám stačí, když vaše funkce `compress` bude komprimovat vstup do posloupnosti *čísel pevné velikosti*, a ne přímo do posloupnosti bitů. Podobně funkce `decompress` na vstupu dostane přímo posloupnost čísel. Pokud se ale cítíte odvážně, *můžete si upravit* typová synonyma `Codeword` a `Encoding` a produkovat a dekomprimovat přímo poslopnost bitů.

Příklady

Pro ověření funkčnosti vaší implementace nabízíme několik příkladů komprese a dekomprese pomocí algoritmu LZW. Stejně jako na anglické Wikipedii za abecedu uvažujeme velká písmena anglické abecedy (`'A'` s kódem `1` až `'Z'` s kódem `26`) a speciální symbol `'#'`, který reprezentuje konec vstupu a má kód `0`. Ke znaku `'#'` se můžete chovat jako ke každému jinému znaku a nemusíte se zabývat případy, kdy ve slově není na konci nebo v něm je víckrát.

Příklad z anglické Wikipedie:

```
ghci> compress "TOBEORNOTTOBEORTOBEORNOT#"
[20,15,2,5,15,18,14,15,20,27,29,31,36,30,32,34,0]
ghci> decompress [20,15,2,5,15,18,14,15,20,27,29,31,36,30,32,34,0]
"TOBEORNOTTOBEORTOBEORNOT#"
```

Další příklady:

```
ghci> compress "ABAB#"
[1,2,27,0]
ghci> compress "ABABABABABAB#"
[1,2,27,29,28,31,0]
ghci> compress "FINRODFINGOLFINANDFINARFIN#"
[6,9,14,18,15,4,27,14,7,15,12,33,1,14,32,28,1,18,33,0]
ghci> decompress [7,9,14,1,12,15,12,31,2,18,9,27,4,1,0]
"GINALOLLOBRIGIDA#"
ghci> decompress [14,5,2,5,14,1,4,16,18,9,19,20,1,22,5,13,13,
                  5,12,15,2,1,18,22,21,20,44,5,22,9,26,5,34,58,
                  14,51,30,1,34,1,26,4,14,25,11,1,31,12,0]
"NEBENADPRISTAVEMMELOBARVUTELEVIZEPREPNUTENAPRAZDNYKANAL#"
```

Rady na závěr

Samozřejmě nejen můžete, ale také byste měli, definovat vlastní pomocné funkce. Ty ale nebudou z modulu `LZW` exportované, a tudíž viditelné pro uživatele.

Zkuste přemýšlet o časové složitosti. Zejména pro reprezentaci kódovacího a dekódovacího slovníku byste měli použít datovou strukturu `Map` z modulu `Data.Map`, a ne obyčejné seznamy, ve kterých má vyhledávání lineární časovou složitost. Konkrétně se vám může hodit definovat si následující pomocné typy:

```haskell
type CompressionDict = Map String Codeword
type DecompressionDict = Map Codeword String
```

Doporučujeme přepsat si předchozí příklady do svého programu jako jednotkové testy pomocí `HUnit`. Stejně tak doporučujeme zamyslet se nad vhodnými vlastnostmi, které by měly vaše funkce splňovat, a testovat je pomocí modulu `QuickCheck`. Například pokud nějaký text komprimujete a výsledek dekomprimujete, měl by vám vyjít původní text.

Pro předávání aktuálního slovníku při kompresi a dekompresi se vám bude hodit monáda `State`. Tuto monádu *nemusíte* použít a můžete dostat plný počet bodů i za *opravdu pěkné* řešení, které ji nepoužívá. Avšak pokud ji použijete, budeme při opravě shovívavější a vy se něco nového naučíte.

Aby byly prakticky použitelné, vaše funkce by měly pracovat líně. Například výpočet výrazu `take 10 $ compress (cycle "AB")` by měl skončit.

Nebojte se často kontrolovat dokumentaci používaných modulů na Hackage či využívat vyhledávač Hoogle. V případě problémů a dotazů se nebojte obrátit se na nás například na diskuzním fóru.