School of Computer Science, University of Nottingham

G51PGP Programming Paradigms, Spring 2019

Graham Hutton

Haskell Coursework II – String Compressor

Deadline: Wednesday 27th March 2019, 12 noon

This coursework covers the material from Lectures 1–7, and is worth 7.5% of the module mark. Half of the marks are awarded for correctness, and half for programming style. In particular, you should aim to make your definitions as simple, clear and elegant as possible. You should attempt to complete the coursework in your own time, but if you get stuck you can ask for help from the tutors during the weekly lab sessions.

This coursework may either be solved on your own, or jointly with ONE other student taking the module; larger groups are not permitted. In the case of jointly produced solutions, both students must be present at the time of assessment to explain the solution and answer questions, and both will receive the same marks.

Assessment will be carried out by oral examination during the lab session – nothing needs to be handed in. When you have completed the coursework ask a tutor to examine your solution. The tutor will then ask you some questions to test your understanding. Note that tutors will only be available to answer questions and assess your solution during the official G51PGP lab sessions (Wednesdays, 11am to 1pm, A32).

Introduction

A *compression scheme* is a method for encoding of string of characters so that it takes up less space. One of the simplest examples is *run-length encoding*, which is useful for strings that contain long runs of repeated characters. To compress a string using this scheme, one simply replaces each such run by a single instance of the repeated character along with a count of the number of times it was repeated. For example, the string

```
aaaaabbbbcc
```

would be compressed to

```
a5b4c2
```

For simplicity, each character in the compressed string will be followed by a single digit, which means that runs of more than nine characters must be broken up into a sequence of runs of at most nine characters. For example, the string

```
ddddddddddddd
```

would be compressed to

```
d9d3
```

The aim of this coursework is to write a Haskell program to compress and decompress a string using this form of run-length encoding.

Compression

1. Define a function

   ```
   chomp :: String -> String
   ```

   that selects a run of repeated characters from the start of a string, with the run being as long as possible. For example:

   ```
   > chomp "aaaaabbbbcc"
   "aaaaa"

   > chomp "ddddddddddddd"
   "ddddddddddddd"
   ```

2. Using `chomp`, define a function

   ```
   munch :: String -> String
   ```

   that selects a run of repeated characters from the start of a string, with the run comprising at most nine characters. For example:

   ```
   > munch "aaaaabbbbcc"
   "aaaaa"

   > munch "ddddddddddddd"
   "ddddddddd"
   ```

3. Using `munch`, define a function

   ```
   runs :: String -> [String]
   ```

   that splits a string into a list of runs of repeated characters, with each run comprising at most nine characters. For example:

   ```
   > runs "aaaaabbbbcc"
   ["aaaaa","bbbb","cc"]

   > runs "ddddddddddddd"
   ["ddddddddd","ddd"]
   ```

4. Using `runs`, define a function

```
encode :: String -> [(Char,Int)]
```

that transforms a string into a list of pairs comprising the character from each run together with its number of repetitions. For example:

```
> encode "aaaaabbbbcc"
[('a',5),('b',4),('c',2)]

> encode "ddddddddddddd"
[('d',9),('d',3)]
```

5. Define a function

```
flatten :: [(Char,Int)] -> String
```

that flattens a list of pairs of characters and digits to a string. For example:

```
> flatten [('a',5),('b',4),('c',2)]
"a5b4c2"

> flatten [('d',9),('d',3)]
"d9d3"
```

6. Using `encode` and `flatten`, define a function

```
compress :: String -> String
```

that compresses a string using run-length encoding. For example:

```
> compress "aaaaabbbbcc"
"a5b4c2"

> compress "ddddddddddddd"
"d9d3"
```

Decompression

7. Define a function

   ```
   decode :: [(Char,Int)] -> String
   ```

   that performs the inverse function to `encode`. For example:

   ```
   > decode [('a',5),('b',4),('c',2)]
   "aaaaabbbbcc"

   > decode [('d',9),('d',3)]
   "dddddddddddd"
   ```

8. Define a function

   ```
   expand :: String -> [(Char,Int)]
   ```

   that performs the inverse function to `flatten`. For example:

   ```
   > expand "a5b4c2"
   [('a',5),('b',4),('c',2)]

   > expand "d9d3"
   [('d',9),('d',3)]
   ```

9. Using `decode` and `expand`, define a function

   ```
   decompress :: String -> String
   ```

   that performs the inverse function to `compress`. For example:

   ```
   > decompress "a5b4c2"
   "aaaaabbbbcc"

   > decompress "d9d3"
   "dddddddddddd"
   ```