CS 410/510 Advanced Functional Programming

Assignment 1

Warming up with recursion and immutable data

Katie Casamento

Spring 2023
Portland State University

This first assignment will be a warm-up to review core functional programming techniques. You may complete this assignment in any programming language of your choice, but pay close attention to the requirements!

For this assignment, you will implement a minimal form of compression and decompression for text data. The compression technique we'll focus on is known as "dictionary compression" or "dictionary coding".

1 Compression

Your compression program will read a text file as input and treat it as a sequence of "words" separated by whitespace. This is the entire definition of a "word" in this algorithm: a substring that is surrounded by whitespace (or the start or end of the string) and contains no whitespace. For example, in the string `"ab,c d 1!2! #$%"`, the words are `"ab,c"`, `"d"`, `"1!2!"`, and `"#$%"`.

For the purposes of this program, "whitespace" is defined as the space, tab, and newline characters. We won't test your code with any other whitespace characters.

Consider this sample input file:

```
abcd nop efghi
  efghi jklm
jklm nop efghi
```

The output of your compression program on this input file should look like this:

```
abcd efghi jklm nop
0 3 1
  1 2
2 3 1
```

The first line of the output is special: it's the "dictionary" in our "dictionary compression". It lists each unique "word" in the file exactly once, each separated by a single space, sorted in lexicographic ("alphabetical") order.

The rest of the output is the text of the original file with each "word" replaced by its index in the "dictionary". Note that all of the whitespace from the input file is included in the output; only the words are replaced.

(To be clear, lexicographic order is what you get when you compare characters with the $<$ or $>$ operators in a programming language: if your language lets you compare strings with $<$ or a function like `strcmp`, or if it has a built-in function for sorting a list of strings, that will be the correct ordering. You shouldn't have to implement a lexicographic string comparison function yourself.)

2 Decompression

Your decompression program will read a text file as input and treat it as a "compressed" text file, in the same format that your compression program outputs.

Consider this sample input file:

```
qrs tuvw xyz
0  0  1
2  1 1
```

The output of your decompression program on this input file should look like this:

```
qrs  qrs  tuvw
xyz  tuvw tuvw
```

Note how the spacing in the output is the same as in the input.

3 Requirements

Your code must not modify **any** variables or data structures after their creation. This means no variable reassignment or field/property/pointer reassignment. This does rule out some purely-imperative languages like raw assembly code, but **nearly** any programming language should still work, as long as you have a form of recursion to use. Make sure to ask if you're unclear on what this requirement means in your language of choice.

If the language you use has a way to mark things as "const" or "final", that's strongly suggested, but not strictly required, as long as you **use** all of your variables and data structures in an immutable way. Similarly, it's fine to use a language like Python, which doesn't technically support "immutable variables", as long as you're not actually doing any mutation.

You may do IO at any point in your program - your code does not need to be fully "pure" by Haskell standards. The only forbidden side-effect is stateful mutation.

You may use external libraries in your code; if you do, please provide sufficient build instructions so that I can run your code even if I don't already have those libraries installed on my machine.

Your code must implement both the compression algorithm and the decompression algorithm given above. You can implement both algorithms in a single program with command-line or GUI options to choose between them, or you can implement the two algorithms in two separate programs.

There are no efficiency or performance requirements, apart from that your code has to finish running within a couple minutes in order for me to reasonably grade it. It's fine if your program isn't very fast on large inputs.

Your program should handle invalid input with some kind of user-friendly error message, not a segfault or an uncaught exception. The specific text content of your error messages will not be graded.

Your program should take input in the form of a path to a file that will be read. In most languages, the easiest way to do this will be to take a file name as a command-line argument. If you're using a web language, you might want to throw together an HTML form for the input path.

The output of your program should be printed to standard output in the console. If you're using a web language, print to the browser developer console.

If you're using a weird language that doesn't have any kind of file input or doesn't have any kind of output console, let me know and we'll work out an alternative to these requirements.

These are the only requirements beyond the algorithm specifications given above. You can choose any program structure you want as long as you follow these requirements.

4 Submitting your work

Submit your work to the Canvas assignment page. You can submit either a zip file of your code or a link to a repository on a code-sharing service like GitLab or GitHub. If you submit a link, please keep your repository private and only give me permission to view it.

Make sure to submit all of the files required to build your code, including any build configuration or project configuration files. Don't include any build artifacts in your submission, like `.class` files in a Java project or the `node_modules` folder in a Node.js project. If you use Git and you submit a zip file, don't include your `.git` folder in the zip file.

Include a `README` file with a short explanation of how to build and run your code. You can assume that I know how to install the standard tools for the language you're using (unless it's really obscure), but I might not be familiar with the build process for projects written in the language. If your code depends on any external libraries, make sure to include instructions for how to acquire them.

5 Grading

Your code will be graded on a scale of 20 points based on whether it implements the algorithm correctly and follows the assignment requirements correctly.