Project 2: RLE Encoding

Image compression and decompression

Overview

For this project, you will implement Run Length Encoding, or RLE, an algorithm that encodes ascii data in a compressed format. Given a text file containing an image constructed with ascii characters, RLE will encode the file in compressed form. Given a file with data in RLE compressed form, you will be able to decompress the file and recover the exact image that was originally compressed. You will use arrays as the basic data structures to implement the RLE compression and decompression algorithms.

Learning Goals:

- Implement a basic compression algorithm.
- Run JUnit tests.
- Use required style guide.

Style Guide:

Follow these style guidelines:

1. All unnecessary lines of code have been removed.
2. Proper indenting must be used.
3. Optimal use of blank lines and spaces for operators.
4. Use of camelCase for variable/parameter and method names.

5. Variables declared early (not within code), and initialized where appropriate and practical.
6. Descriptive variable and method names.

Project Specification:

Background.

RLE is a lossless encoding algorithm. That means there is no loss of data when it is used to compress and decompress data. In this project, you will be implementing RLE to compress and decompress image files composed of two ascii characters. The algorithm can be applied to any number of characters, but we are using only two characters in this project.

The files you will be encoding are in the form of rows of ascii characters. Each row will be encoded in RLE format for compression, and then each compressed row will be decoded back into the original characters. This is an example of an image you will be working with:

```
___________$$$$$$$$$$$$$$$______________
________$$$$______$$$$$$$$______________
______$$$_____$$$$$$_$$_$$______________
_____$____$$$$$$$$$__$__$$______________
___$$___$$$$$$$$$___$___$$______________
__$$__$$$$$$$$$$___$____$$______________
__$$$$$$$$$$$_____$$_____$$____$$$$$$___
$$$$$$$__________$$_______$$_$$$$$$$$$$_
_________$$$$$__$$$____$$__$__$$$$$$$$$_
______$$$$$$$$$_$$$$$$$$$$$$$$$$$$$$$$$$
_____$$$$$$$$$$$$__$$$$$__$$$$$$$$$$$$$$
____$$$__$$$$$$$$$$$$$$$$__$$$$$$$$$__$$
____$$___$$$$$$$$$$$$$$$$$_$$$$$$$$$__$$
____$$__$$$$$$$$$$$$$$$$$$_$$$$$$$$$_$$_
____$$$_$$$$$$$$$$$$$$$$$$_$$$$$$$$$$$__
____$$$__$$$$$$$$$$$$$$$$_$$$$$$$$$$$___
_____$$$$$$$$$$$$$$$$$$$__$$$$$$$$$_____
_____$$$$$$$$$$$$$$$$$$$________________
_______$$$$$$$$$$$$$$$__________________
__________$$$$$$$$$$____________________
```

cherries.txt

This image is made up of two ascii characters: the underscore, _, and the dollar sign, $.

The RLE algorithm works in the following manner: given a row of data, the algorithm replaces each row with numbers that say how many consecutive pixels are the same character, always starting with the number of occurrences of the character that is encountered first in the file.
For example, in the image above, the first row contains the following sequence of characters: 11 underscores, 15 dollar signs, 14 underscores. The RLE compressed version of this row would be: 11,15,14
Note the compressed row is in comma-separated values, or CSV, format. Also note that the length of the rows in the image file may be of different lengths (see the file mickey.txt).

Decompression: to recover the image from a compressed row, you have to know what character goes with the frequency of occurrences. In the above example, you would return a string of 15 underscores, followed by 14 dollar signs, followed by 11 underscores. What about this row:
$$$$$$$__________$$_______$$_$$$$$$$$$$_
The leading character is not the underscore. So, the compressed version cannot start with 7. The underscore was the first character encountered in the file, on row 1, so it always has to be the first number in a compressed line. In this case, the compressed row has to start with a 0. The correct RLE encoding of this line is: 0,7,10,2,7,2,1,10,1.

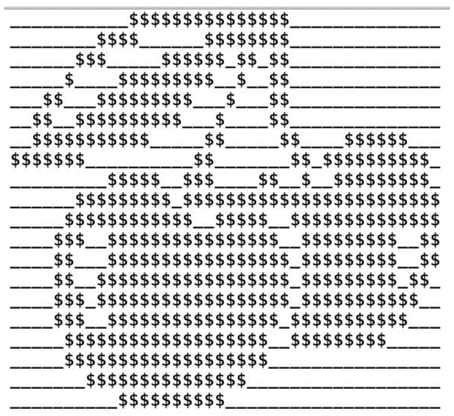
In order to decompress an RLE-compressed file, you have to include the character information in the first row. For example, the first row in the compressed cherries file would be: _,$. The first row must be in csv format as well. Note that we are only dealing with 2 characters, but this could be generalized to any number of characters.

The complete compressed file would be:

```
_,$
11,15,14
8,4,6,8,14
6,3,5,6,1,2,1,2,14
5,1,4,9,2,1,2,2,14
3,2,3,9,3,1,3,2,14
2,2,2,10,3,1,4,2,14
2,11,5,2,5,2,4,6,3
0,7,10,2,7,2,1,10,1
9,5,2,3,4,2,2,1,2,9,1
6,9,1,24
5,12,2,5,2,14
4,3,2,16,2,9,2,2
4,2,3,17,1,9,2,2
4,2,2,18,1,9,1,2,1
4,3,1,18,1,11,2
4,3,2,16,1,11,3
5,19,2,9,5
5,19,16
7,15,18
10,10,20
```

RLE_cherries.txt

The compressed files your code produces will have a “RLE_” prefix attached to the original file name. Your decompressed files will have a “DECOMP_” prefix.

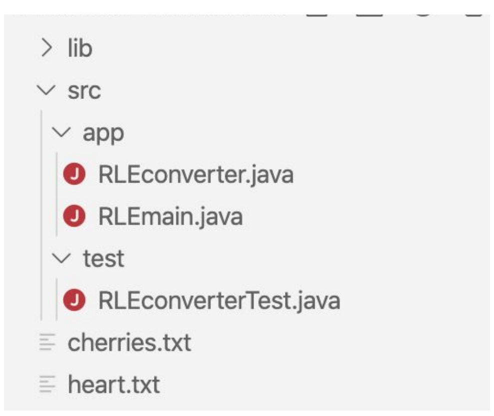


```
___________$$$$$$$$$$$$$$$______________
________$$$$______$$$$$$$$______________
______$$$_____$$$$$$_$$_$$______________
_____$____$$$$$$$$$__$__$$______________
___$$___$$$$$$$$$___$___$$______________
__$$__$$$$$$$$$$___$____$$______________
__$$$$$$$$$$$_____$$_____$$____$$$$$$___
$$$$$$$__________$$_______$$_$$$$$$$$$$_
_________$$$$$__$$$____$$__$__$$$$$$$$$_
______$$$$$$$$$_$$$$$$$$$$$$$$$$$$$$$$$$
_____$$$$$$$$$$$$__$$$$$__$$$$$$$$$$$$$$
____$$$__$$$$$$$$$$$$$$$$__$$$$$$$$$__$$
____$$___$$$$$$$$$$$$$$$$$_$$$$$$$$$__$$
____$$__$$$$$$$$$$$$$$$$$$_$$$$$$$$$_$$_
____$$$_$$$$$$$$$$$$$$$$$$_$$$$$$$$$$$__
____$$$__$$$$$$$$$$$$$$$$_$$$$$$$$$$$___
_____$$$$$$$$$$$$$$$$$$$__$$$$$$$$$_____
_____$$$$$$$$$$$$$$$$$$$________________
_______$$$$$$$$$$$$$$$__________________
__________$$$$$$$$$$____________________
```

DECOMP_RLE_cherries.txt

Code Structure.

The project comes with these files:



```
> lib
v src
  v app
    RLEconverter.java
    RLEmain.java
  v test
    RLEconverterTest.java
  cherries.txt
  heart.txt
```

There are several JUnit tests for you to run in **`RLEconverterTest.java`** that fail initially. There are also two ascii text files to work with: **`heart.txt`** and **`cherries.txt.`**

When you first run the main method in **`RLEmain.java`** with **`con.compressFile("heart.txt");`** it creates the **`RLE_heart.txt`** file in the starter code folder with null contents. This is because of incomplete methods.
Similarly, if you run the main method with **`con.decompressFile("RLE_heart.txt");`** it creates the **`DECOMP_RLE_heart.txt`** file in the starter code folder with null contents, as some methods are incomplete. You will be developing the TODO tasks in the **`RLEconverter.java`** file.

Tasks.

There are six TODOs in `RLEconverter.java`. ***Do not modify any existing instance variables, method signatures or any of the starter code provided. Do not add any new classes to the project. You may add any additional instance variables or helper methods you wish.***

The code is symmetrical with regard to compression methods and decompression methods.

Compression:
The main method for compressing a (decompressed) file is:

`public void compressFile(String fileName) throws IOException`
This method code is provided.

You will implement the following methods, marked by TODOs:

TODO1:
`public String compressLine(String line, char[] fileChars)`
This method returns a compressed string of its argument in RLE format. It uses the `fileChars` array in this process. The first char in the `fileChars` array is the first ascii character encountered in the original file.

TODO2:
`String[] compressLines(String[] lines)`
Processed each line in its argument. This method calls compressLine to do that task. Note that the two ascii characters must be known before a line can be compressed. The fileChar array is provided for that purpose.

TODO3:
`getCompressedFileStr(String[] compressed, char[] fileChars)`
This method assembles the string in csv format that will be written to the compressed file. The compressed file must begin with the characters used in the image file.

Decompression:
The main method for decompressing a compressed file is:
`public void decompressFile(String fileName) throws IOException`
This method code is provided.

You will implement the following three methods, marked by TODOs:

TODO4:
`public String decompressLine(String line, char[] fileChars)`
This method returns a decompressed string of its argument. It uses the fileChars array in this process. The first char in the fileChars array is the first ascii character encountered in the original file.

TODO5:
`public String[] decompressLines(String[] lines)`
Processed each line in its argument. This method calls decompressLine to do that task. Note that the two ascii characters will be the first row of an RLE compressed file.

TODO6:
`public String getDecompressedFileStr(String[] decompressed)`
This method assembles the string that will be written to the decompressed file. This string is the same string as in the original file.

Export and Submit

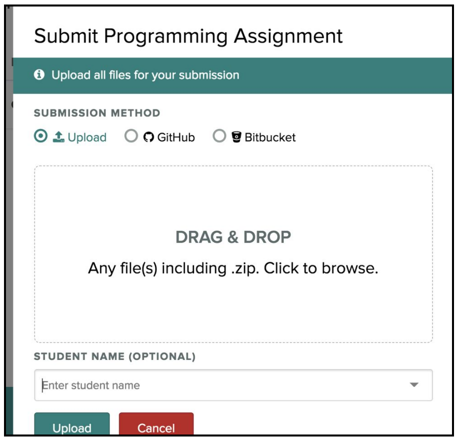
Step 1: Export your project
Within VSCode click on the "*View* > *Command Palette*…" menu option. Then type into the Command Palette: "*Archive Folder*" and hit enter. This will produce a Zip file of your project folder. You can then upload that zip file to the corresponding project assignment in Gradescope. You can add the Archive extension to VSCode if you don't have it.

If your zip file is not in the correct form, the autograder will not process your code and you will not receive any credit.
The correct file structure is:

```
ProjectRLECompression-starter
  src
    app
        RLEconverter.java
```

Note that we will not grade code manually. If your code does not compile or if your zip file is not correctly structured you will not receive any credit for the project.

Step 2: Submit the zip file to Gradescope
Log into Gradescope, select theProject 2 assignment, and submit the zip file for grading.


Submit Programming Assignment
Upload all files for your submission
SUBMISSION METHOD
Upload
GitHub
Bitbucket
DRAG & DROP
Any file(s) including .zip. Click to browse.
STUDENT NAME (OPTIONAL)
Enter student name
Upload
Cancel

There are usually more tests in the autograder than provided with the starter code. If your code passes all provided tests, it is a good indication that your code will pass all of the autograder tests. If that is not the case, you are likely not considering some of the "edge" cases in your algorithm. Re-examine your code, use the debugger to troubleshoot. Pose specific questions on Piazza for some outside help. Remember that these projects take longer than you estimate. Starting early means you have more time to get help if you are stuck.

The autograder will not run successfully if you do submit a **correctly formatted zip file**- it has to have the same **names and directory structure as described on page 6 above**. The autograder will also not run if your code **does not compile**, or if you i**mport libraries** that were not specifically allowed in the instructions.

Remember, you can re-submit the assignment to gradescope as many times as you want, until the deadline. If it turns out you missed something and your code doesn't pass 100% of the tests, you can keep working until it does. Attend office hours for help or post your questions in Piazza.

Dev Tips

- Use your debugger! It is very useful in finding exactly where your program stops doing what you want it to do.
- Debug your code in your development environment- that is what it is designed for. Do not use gradescope as a way to develop your code- it will not be helpful.
- Start early! Projects are starting to get a little longer, so if you get stuck you need to make sure you have enough time to seek help.
- Seek help when you get stuck. We have office hours and Piazza chat specifically for you to ask questions when you need assistance. Use public posts as much as possible so we don't have to answer the same question multiple times, only use a private post if you need us to see your code or have questions specific to you.

- Submit to gradescope at least once, even if you aren’t completely done. There is a huge difference between a 50% and a 0%. That said, aim for 100%.