Instructions: Homework must be typeset in \LaTeX.

CSV Format Verifier

The CSV format is a common, text-only format for sharing tabular data. For the purposes of this assignment, the CSV format has the following requirements:

1. Values are separated by either a comma (,) or a newline, but not both.

2. The last value in a CSV may terminate with or without a newline, but it may not terminate with a comma.

3. Values may be an empty string (e.g., 1,,3 or 1,"",3).

4. If a value is intended to contain a comma, then the value must be surrounded by double quotes (e.g., the value "South Bend, Indiana").

5. Any value may be surrounded by double quotes, however the entirety of the value must be surrounded by the quotes (e.g., 12"3" is not valid). No whitespace may appear between the surrounding double quotes and the separating comma or newline. Whitespace is, by definition, part of a value (with the exception of the newline at the end of each line of text).

6. If a value is intended to contain a newline, then the value must be surrounded by double quotes.

7. If a value is intended to contain a double quote, then it must be escaped by placing two double quotes back-to-back and surrounding the entire value with double quotes (e.g. "That's ""good"" for now").

8. Valid Characters are all the numeric and alphabetical characters, as well as the space character (\textvisiblespace, or \textvisiblespace in \LaTeX), and the characters ,.". No other characters should be considered valid input. Use \text{\textvisiblespace} when in math mode, if you want to get rid of warnings.

There is typically one further requirement for the CSV format: Each line must contain the same number of values. Because this verification cannot be performed by a FSM, we will ignore it for now.
1. (20 Points) For this problem, you are to create a Python (version 3, please) CSV Format Verifier as a FSM. That is, given an input string (which may comprise multiple lines of text), you are to model a FSM that will verify that the string is in a valid CSV format. As such, your code should only examine each character from the input once, and may only use a single variable to track the state of the FSM.

You only need to turn in your verifying function, included in your PDF. It should be named isValidCSV() and can follow this format (information for how to cleanly format source code is included at the end of this document):

```python
def isValidCSV(text):
    state = "your starting state"
    for ch in text:
        # Examine the ch and change state here...
        # Basically all logic is here...
        # May return False at any time if the string does not validate!
        # Did the CSV validate?
    return True
```

**Hints:**

- Use descriptive names for your states rather than something like $q_0$.
- The intent is that I should be able to copy-and-paste your function into my own code, and it will return either True or False letting me know whether or not the string provided (text in the function argument) is in a valid CSV format.
- Yes, we will be testing your code to see if it is correct.

2. (20 Points) Visualize your state machine as a DFA and give a formal definition of your machine. It should be obvious that the labels on the edges can get messy very quickly. Invent a notation that you can use to cleanly and succinctly express these otherwise long definitions and use this notation in your DFA and transition matrix.

3. (35 Points) Convert your DFA into a Regular Expression. Show each step of the conversion process. It is HIGHLY recommended that you use symbols to represent the edge labels. In fact, you can invent any symbol notation that you like, provided that you adequately define the symbols at the appropriate step.
4. (15 Points) Convert the final Regular Expression from the previous step (which was defined using your invented notation) into an actual Regular Expression. Yes, it will be quite large.

Example:

Suppose you defined the following symbols as:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Represents</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>${1</td>
</tr>
<tr>
<td>$B$</td>
<td>${a-z</td>
</tr>
<tr>
<td>$C$</td>
<td>$AA</td>
</tr>
<tr>
<td>$D$</td>
<td>$BC^*$</td>
</tr>
</tbody>
</table>

Suppose your final Regular Expression from the previous step is: $AB|D$. This would expand as follows:

$AB|D$

$\{1|3|5|7|9\}B|D$

$\{1|3|5|7|9\}\{a-z|A-Z|\}|D$

$\{1|3|5|7|9\}\{a-z|A-Z|\}|BC^*$

$\{1|3|5|7|9\}\{a-z|A-Z|\}|\{a-z|A-Z|\}C^*$

$\{1|3|5|7|9\}\{a-z|A-Z|\}|\{a-z|A-Z|\}|(AA|B)^*$

$\{1|3|5|7|9\}\{a-z|A-Z|\}|\{a-z|A-Z|\}|(\{1|3|5|7|9\}\{A|B\}^*)$

$\{1|3|5|7|9\}\{a-z|A-Z|\}|\{a-z|A-Z|\}|(\{1|3|5|7|9\}\{1|3|5|7|9\}|B)^*$

$\{1|3|5|7|9\}\{a-z|A-Z|\}|\{a-z|A-Z|\}|(\{1|3|5|7|9\}\{1|3|5|7|9\}|\{a-z|A-Z|\})^*$

There are a few important things to note about this requirement. (1.) No, you do not have to show as many steps as I did. You could, for example, replace all of the $D$’s at once, followed by all of the $C$’s, etc. (2.) Clever use of macros (see the \LaTeX intro doc) can make this a relatively simple and error-free process. (3.) I used a range notation (see $B$) so that I don’t have to type the entire alphabet. This is acceptable when the meaning is clear (or otherwise defined). If you prefer, you may use an official RegEx syntax, as long as you properly identify it.
Here are a few helpful \LaTeX{} snippets. First, in your preamble, add the following, which will allow you to include nicely-formatted source code:

\begin{verbatim}
\usepackage{minted}
\definecolor{shadecolor}{rgb}{.95, .95, .95}
\setminted[python]{
  linenos=TRUE,
  breaklines=TRUE,
  breakanywhere=TRUE,
  bgcolor=shadecolor
}
\end{verbatim}

You can now include your source code directly into your \LaTeX{} document like this (This exact code snippet was shown earlier):

\begin{verbatim}
def isValidCSV(text):
    state = "your starting state"
    for ch in text:
        # Examine the ch and change state here...
        # Basically all logic is here...
        # May return False at any time if the string does not validate!
        # Did the CSV validate?
    return True
\end{verbatim}