CAB203 Problem Solving

Ryan Indrananda: n10852565

1 Regular Languages and Finite State Automata Problem

Regular expressions (or regex) are any sequence of characters by which forms a pattern to search, particularly used to identify whether a string contains (matches) said pattern[1]. It is a way to specify a regular language over an alphabet Σ. These patterns are able to be defined by and corresponds to finite state automata, which is a model of computation defining a set of states S, starting state $s0 \in S$, a set of accepting states $A \subseteq S$, and a state change function $\delta : S \times \Sigma \rightarrow S$.[2]

The problem states that, for a sample string (starting state), all occurrences of articles must be replaced by #'s corresponding to the number of letters in the redacted article (state change function).

The set of articles is:

$$\{a, an, the\}$$

Therefore, the decision problem is deciding whether a given word within a string explicitly is in the set of articles. If yes, then replace said article with an appropriate number of #'s. The solution to this problem was achieved through the use of the following three Regular Expression functions and syntaxes with the use of Python: re.subn, the \b syntax, and the re.IGNORECASE flag.

Let's describe what and why each of these were relevant to the solution and thus used. The re.subn function performs similarly to the re.sub function, which takes a string then replaces the leftmost non-overlapping occurrence of a defined pattern with a defined replacement. For example, if 'the' is the pattern and '###' the replacement, the function will search through the entire string for every occurrence of 'the' and replace it directly with '###'. Subn does the same, however it returns a tuple containing the new string and the number of replacements made. As such tuple indexing was done to extract only the final string, done by adding '[0]' to the end of the subn function.[3]

---

[1] (Python RegEx, 2019)
[2] CAB203 Lecture 10, Slide 20
[3] (Re — Regular Expression Operations — Python 3.7.2 Documentation, 2009)

The \b syntax essentially works as a boundary which matches an empty string at either the beginning or end of a defined word (in this case each word in the set of articles). For example, r'\bthe\b' matches 'the', 'the.', and '(word) the (word)', but will not match 'therefore' or '2the'. Implementing this search pattern method into the subn function will appropriately search for every occurrence of each word in the set of articles.[4]

Finally, the re.IGNORECASE flag in combination with the subn function will ensure that the string matches for each of our articles using the \b syntax is done case-insensitively. This means that r'\bthe\b' with the flag will match 'The', 'tHe', 'THE', etc.[4]

An example of a complete subn search will look as such:

r = re.subn(r'\ba\b', '#', s, flags = re.IGNORECASE)[0]

Where *r* is the final string after replacement and *s* is a sample string by which to conduct the # replacements. The code will first replace all 'a's in the string, then all 'an's, then finally all 'the's. Subn works such that if no replacements are done, the final string is simply the initial sample string. Therefore, the final string will either be all the articles replaced with the appropriate amount of #'s, or simply the initial string with no replacements. For final strings that do have replacements, student numbers are appended.

2 Linear Algebra Problem

Before further solving, let's first discuss some prerequisite knowledge involving matrices and matrix operations. A 'matrix' is a 'rectangular arrangement of data elements', whereby an $m\ x\ n$ matrix has *m* rows and *n* columns[5]. A sample 2 x 2 matrix is:

$$\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

To symbolise the entries within the matrix, a_{ij} determines the *i*'th row and *j*'th column of matrix *a*. We may also multiply vectors (essentially matrices with only one row and column) by matrices. If we have a vector *x* and we have an *m x n* matrix *A* where *m* = *n*, it is possible to multiply them to generate a separate vector *z* = A*x*, where $z_j = \sum_{k=1}^{n} a_{jk}x_k$. For example, multiplying our $\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$ matrix by a vector $\begin{pmatrix} 2 \\ 2 \end{pmatrix}$ will result in $\begin{matrix} (1x2) + (2x2) \\ (2x2) + (1x2) \end{matrix} = \begin{pmatrix} 6 \\ 6 \end{pmatrix}$.

[4] (Re — Regular Expression Operations — Python 3.7.2 Documentation, 2009)
[5] CAB203 Lecture 11, Slide 19

Matrices may also have inverses, whereby for matrix *A*, (A^{-1} *A*) *x* = *x*. A square matrix *A* has an inverse if and only if it's determinant is non-zero. To calculate a 2 x 2 matrix's determinant:

$$\det\begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc$$

To calculate a 2 x 2 matrix's inverse:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

From above, we can see that dividing by a determinant that is zero is not possible[6].

Let us now discuss the problem and relevant solution. This problem relates to the solving of simultaneous equations in the form of:

$$ax + by = e$$
$$cx + dy = f$$

For this problem, we can substitute and define some of these variables. *a* is the amount of nitrogen in 1kg of type A fertiliser, *b* is the amount of nitrogen in 1kg of type B fertiliser, *c* is the amount of phosphate in 1kg of type A fertiliser, *d* is the amount of phosphate in 1kg of type B fertiliser, *e* is amount of nitrogen required by the crop, and *f* is the amount of phosphate required by the crop. We want to find *x* and y, where *x* is amount of fertiliser of type A required and *y* is the amount of fertiliser of type B required. The aforementioned form of solving simultaneous equations can be represented as an equation in the form of vectors:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} e \\ f \end{pmatrix}$$

Then *x* and *y* are solvable by finding the inverse of the matrix:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1}\begin{pmatrix} e \\ f \end{pmatrix}$$ [7]

[6] CAB203 Lecture 11, Slides 35 and 36
[7] CAB203 Lecture 11, Slide 47

As an example, let's say that we know the following:

- $a = 0.3$
- $b = 0.1$
- $c = 0.2$
- $d = 0.4$
- $e = 10$
- $f = 20$

We can substitute this into the previous solvable equation:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.3 & 0.1 \\ 0.2 & 0.4 \end{pmatrix}^{-1} \begin{pmatrix} 10 \\ 20 \end{pmatrix}$$

$$\begin{pmatrix} 0.3 & 0.1 \\ 0.2 & 0.4 \end{pmatrix}^{-1} = \begin{pmatrix} 4 & -1 \\ -2 & 3 \end{pmatrix}$$

$$\begin{pmatrix} 4 & -1 \\ -2 & 3 \end{pmatrix} \begin{pmatrix} 10 \\ 20 \end{pmatrix} = \begin{pmatrix} 20 \\ 40 \end{pmatrix}$$

Therefore $x = 20$, $y = 40$. We can say that the farmer requires 20 of fertiliser of type A and 40 of fertiliser of type B for the given knowledge. The overall Python implementation was done using the numpy library, specifically the numpy.array and numpy.linalg.inv functions.

3 References

1. re — Regular expression operations — Python 3.7.2 documentation. (2009). Python.org. https://docs.python.org/3/library/re.html
2. Python RegEx. (2019). W3schools.com. https://www.w3schools.com/python/python_regex.asp
3. CAB203 Lectures 10 and 11