CAB203 Problem solving

Harrison Leach: n11039639

1 Regular languages and finite state automata

To begin the explanation of the censor(s) solution, five features of the python regular expression library, *re*, will be explained.

`\b`, is a part of the regular expression syntax. Its purpose is to find the empty string, ε, at the beginning and end of a word.

The next feature is the function .compile(). It compiles the given regular expression into an object that can be used by a matching method such as .findall(). Its importance is significant as it allows for the introduction for the third feature, the IGNORECASE flag. This flag enables case insensitivities to the regular expression. It simplifies the re pattern greatly by including its current case's counterpart in the search. This flag can be set within compile like so: .compile(pattern, re.IGNORECASE).

The fourth is the .sub() function's ability to call a function instead of a single string.

The final function is .group(), this function returns one or more subgroups of the match. In the case of this solution, .group(0) is used as it will return the entirety of the match.

The solution was done by initially creating the pattern:

$$/b(a|an|the)/b$$

This regular expression simply searches for the beginning and end of a word, and then with use of pipes, |, the words that are accepted are either *a*, *an* or *the*. With use of the compile function and IGNORECASE flag, the regular expression object is created:

```
regex = re.compile("\\b(a|an|the)\\b", re.IGNORECASE)
```

The given string, *s*, the re object, and a helper function called hashReplace are passed into the .sub() function. When a match is made the hashReplace() function is called. This function returns the same amount of #'s as the length of .group(0). These hashes are then substituted in place of the words that have been deemed as matches.

The .findall() function is also used to check if matches have been made. If .findall() returns an empty array, [ ], nothing will happen, however if a match is made, `' <n11039639>` is appended to the end of *s*. The string, *s*, is then returned from censor(s).

2 Linear algebra

The fertiliser(an, ap, bn, bp, n, p) solution was completed with the application of linear algebra.

The farmers issue can be modelled with 2 equations:

$$an \cdot a + bn \cdot b = n$$

$$ap \cdot a + bp \cdot b = p$$

Where:

- a: is the amount of type A fertiliser
- b: is the amount of type B fertiliser
- an: is the portion of nitrogen in type A
- bn: is the portion of nitrogen in type B
- ap: is the portion of phosphate in type A
- bp: is the portion of phosphate in type B
- n: is the total amount of nitrogen
- p: is the total amount of phosphate

Using knowledge of matrices, the above equations can be converted into a single matrix equation:

$$\begin{pmatrix} an & bn \\ ap & bp \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} n \\ p \end{pmatrix}$$

By multiplying the inverse of the 2x2 matrix (from hereon will be referred to as A) on both sides of the equation, a & b can be found:

$$\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} an & bn \\ ap & bp \end{pmatrix}^{-1} \cdot \begin{pmatrix} n \\ p \end{pmatrix}$$

Using a python library, *numpy* (*np*), arrays can be created for example:

```
A = np.array([
    [an, bn],
    [ap, bp]
    ])
```

Numpy also has a linear algebra section which is called using `linalg`. The matrix that contains a & b, X, can be calculated in python using .inv(), and @ is *numpy's* symbol for matrix multiplication:

```
X = np.linalg.inv(A) @ B
```

If the determinant of A is equal to 0, this means that there is no solution and `None` should be returned. As well as this, if a or b are less than zero, return `None`.