Problem 1. POS Tagging using **Pairs**

Mapper takes a document -> assigns POS tags to words -> parses each POS tag and emits (tag, 1)

- Example output from mapper: (NNP, 1), (JJ, 1), (VBZ, 1), (CD, 1), (NNP, 1), ...

Combiner combines on the keys and produces output like (NNP, [1, 1, ...]), (VBZ, [1, 1, ...]), ...

Reducer takes the output of combiner, sums up the values corresponding to each key and produces output that looks like (NNP, 5), (VBZ, 7), ... which is what we want.

Running time:

```
real    0m28.027s
user    1m17.501s
sys     0m4.354s
```

Problem 1. POS Tagging using **Stripes**

Mapper takes a document -> keeps key as 'Answer' and value as a map {NNP: 5, VBZ: 7, JJ: 8, ...} and then emits ('Answer', map)

Reducer gets one such map from each document and it sums up the corresponding values of POS tag frequency from each map and outputs the resultant map {NNP: 44, VBZ: 97, ...} which is our answer.

Running time:

```
real    0m26.832s
user    1m4.100s
sys     0m3.182s
```

*NOTE: The values given above are just for example purposes and are not the actual output.

Problem 2. Part 1

Mapper takes each document -> converts it to lowercase -> removes stop words -> for each word in document, it stems it and emits (word, document_name).

- Example output from mapper 1:
    - ('hello', d1), ('hello', d1), ...
    - Example output from mapper 2:
        - ('bye', d2), ('hello', d2), ...

Combiner combines on each word and produces - ('hello', [d1, d1, d2, ...]), ('bye', [d2, ...]), ...

Reducer takes the value [d1, d1, d2, ...] and removes duplicates to generate value [d1, d2, ...]. Then, corresponding to each word, we find size of its value and emit (word, size_value). Size of a word's value is equal to its document frequency.

- Example output from reducer:

- (hello, 5), (bye, 7), ...

Problem 2. Part 2

We took the top 100 words having document frequency > 14.

Mapper takes a document -> converts it to lowercase -> removes stop words -> for each word in document, it stems it and checks if it's one of the above 100 words and outputs (document_name, {word1: frequency1, word2: frequency2, ..., word100: frequency100}).

Reducer will aggregate all the word frequencies which is equal to term frequencies.

After this, TFIDF score was computed using cached document frequency and the newly computed term frequency.