BLG Challenge - "Phonewords Coding Challenge"

1. Why I chose "Phonewords Coding Challenge"

   After going through the 2 projects' descriptions, I chose to go with the "Phonewords Coding Challenge" because I found it is more challenging than the "Bike-Simulation Coding Challenge". Particularly, the phonewords project requires a more complicated algorithm to solve the problem than the "Bike-Simulation" which, I think, mainly relies on if statements. Another challenge with the Phonewords project is that the design needs to take the dictionary file size into consideration to balance between the performance and memory resource used.

2. My solution
   a. Assumptions

      Since there are no requirements for the size of the dictionary file, I assume the dictionary contains only English words, which is around 194,000 words (http://www.gwicks.net/dictionaries.htm).

   b. Design
      - My approach is I want to optimize the performance of the process of the looking up words. Since 194,000 words is not a big number, all the words in the dictionary will be loaded into memory; particularly, I use a HashMap in which the key is the encoded number and the value is a list of words which have the same encoded number.
      - To optimise the process of looking up all phone numbers in a file, all the phone numbers will be loaded into memory, a Java List which will be parallely processed using Java 8 parallel stream.

   c. Algorithm to find phonewords
      - High level view
        - When finding all possible phonewords of a phone number
          - Find all substrings of the given phone number which can be translated into words in the dictionary.
          - Group the substrings into different combinations which I call patterns.
          - Eliminate invalid patterns.
          - Translate the patterns into phonewords.
          - Eliminates invalid phonewords. E.g. contains 2 consecutive digits
      - Complexity
        - If choosing the basic operation is when looking up a number in the dictionary. The cost to look up an exact number is O(1). The cost to find possible words of a number is $O(L^2)$ in which L is the length of the phone number.

3. Source Code & Instruction
   a. Code repository:

      https://github.com/luantrongtran/phonewords

   b. Instruction to run:
      - In the repository, there is an executable jar file located in the **bin** folder.
      - Running the app in terminal, assuming that the terminal is in **bin** folder; the **dictionary.txt** is the dictionary file and the; and the **phonenumbers.txt** containing phone number to be looked up

```
java -jar phonewords-1.0.jar -dictionary dictionary.txt -input phonenumbers.txt
```

      - You can point to your own dictionary file and input file in different directories.
      - All the found phonewords will be written into **output.txt** located in the **bin** folder