Programming and Data Structures with Python: Problem Set 4

1. Adapt your implementation of Bloomfilters for the previous assignment to be space efficient. That is, it should use exactly 1 bit per entry (and hence in each byte it should store 8 values).

2. Write a python function `firstWords` which takes a string as input and prints out all the words that occur as the first word of a sentence in that string. This has to be accomplished directely without using the `re` library.

3. Each machine on the network is assigned an IP address by a DHCP server. The information on what IP to assign to each machine is stored in the dhcp server as a sequence of entries. Here is typical entry:

   ```
   host conf01 { hardware ethernet 60:03:08:f0:b7:04; fixed-address 192.168.54.101;}
   ```

   It associates a name (`conf01`) with a hardware address (`60:03:08:f0:b7:04`) and an IP address (`192.168.54.101`). Each hardware address consists of 6 pairs of hexadecimal digits. The digits a, b, c, d, e, f may appear in lower case or upper case. The 6 pairs are separated either by `:` as above or `-` as below:

   ```
   host conf02 { hardware ethernet 60-03-08-9C-67-72; fixed-address 192.168.54.102;}
   ```

   The IP address is a sequence of 4 numbers separated by `.`s. Each number ranges from 0 to 255 and hence may need 1 or 2 or 3 digits. There is no restriction on the number of spaces between different words in each entry. Any line beginning with a `#` is a comment and should be ignored.

   Your task is to write a python program that takes such a file containing a list of dhcp entries insterspersed with comment lines as input and does the following. It outputs the name, the hardware address and the IP address of each entry, one per line, separated by commas. Your program must normalize the hardware address entries so that each entry uses only upper case letters and also uses only `:` and not `-`. If either the hardware address or the IP address is not valid then you must output the word `Error` at the end of the line.

   For instance, if the input file contains the following lines:

   ```
   # The entries begin here
   host conf01 { hardware ethernet 60:03:08:f0:b7:04; fixed-address 192.168.54.101;}
   host conf02 { hardware ethernet 60-03-08-9G-67-75; fixed-address 192.168.54.102;}
   # This is just a comment
   host conf03 { hardware ethernet 60-03-88-9C-67-72; fixed-address 192.268.54.203;}
   host conf04 { hardware ethernet 60-03-08-9C-67-72; fixed-address 192.168.54.84;}
   ```

Then the output should be:

```
conf01,60:03:08:F0:B7:04,192.168.54.101
conf02,60:03:08:9G:67:75,192.168.54.102 Error
conf03,60:03:88:9C:67:72,192.268.54.203 Error
conf04,60:03:08:9C:67:72,192.167.54.84
```

4. Write a python program `article.py` which replaces all occurrences of the word *a* in a text which is followed by a word beginning with a vowel by *an* and replaces all occurrences of the word *an* that is followed by a word beginning with a consonant by the word *a*.

5. Write a python program `allBST.py` which takes an integer `k` as a command-line argument and prints all the binary search trees whose elements are `1,2,...,k`. For example for `k = 4` your output should consist of the following 12 trees. The order in which you output the trees is not important.

```
(4 (3 (2 1 _) _) _)
(4 (3 (1 _ 2) _) _)
(4 (2 1 3) _)
(4 (1 _ (2 _ 3)) _)
(3 (2 1 _) 4)
(3 (1 _ 2) 4)
(2 1 (3 _ 4))
(2 1 (4 3 _))
(1 _ (2 _ (3 _ 4)))
(1 _ (3 2 4))
(1 _ (4 (2 _ 3) _))
(1 _ (4 (3 2 _)))
```