CSE 326 Assignment #3
Due: Wednesday, August 7th

1 The Unix `spell` command reads an input file and prints out all words in the input file which are not in some reference dictionary. Suppose the reference dictionary contains 30,000 words, and we only wish to make one pass through the input file. A simple strategy is to read the reference dictionary into a hash table and use the hash table to look for each input word as it is read.

If memory is limited, however, and the entire dictionary cannot be stored in a hash table, we can still get an efficient algorithm that almost always works. We declare an array `table`, with type `bool` or `boolean` (initialized to `false`) of size `tableSize`. As we read in a word from the reference dictionary, we set `table[hash(word)] = true`. Which of the following statements are true? Justify your answers.

(a) If a word hashes to a location with value `false`, then the word is not in the dictionary.
(b) If a word hashes to a location with value `true`, then the word is in the dictionary.

Suppose we choose `tableSize` = 300,007.

(c) Assume that an average word can be stored in 7 bytes, that a `bool` or `boolean` can be represented with one bit, and that the table of `bool`s or `boolean`s is packed as densely as possible (8 `bool` or `boolean` per byte). How much memory does the table require?
(d) What is the probability that a misspelled word is *not* recognized as a misspelling by this algorithm? What is the probability that the algorithm mistakes a properly-spelled word as a misspelling?
(e) A typical document might have about 3 actual misspellings per page of 500 words. Is this algorithm usable?

2 Explain what happens in an extendible hash table when many keys hash to similar hash values. What happens when many similar keys are inserted?

3 You have a set of elements `a`-`j` and perform the following sequence of operations on a Disjoint Sets ADT:

```
find(d)

union(d, a)

union(b, c)

union(h, j)

find(c)

union(h, b)
```

```
find(j)

union(b, a)
```

(a) Without weighted union or path compression, and choosing the root of a union by selecting the alphabetically smaller node, show the up-tree forest which results from the above operations. At what depth does node j end up?

(b) With weighted union, but without path compression, and breaking ties on unions by selecting the alphabetically smaller node, at what depth does node j end up? It is not necessary to show the resultant up-tree forest again.

(c) With weighted union and path compression, and breaking ties on unions by selecting the alphabetically smaller node, at what depth does node j end up? It is not necessary to show the resultant up-tree forest again.

4 The SortDetective lab exercise, available on the couse webpage, under "Homework".