# a Bag of Make Me Words

January 7, 2017

A standard part of text analysis—for machine learning, machine translation, sentiment analysis, and so on—is transforming a document into a "bag of words" representation. A "bag of words" is a data structure that maps the unique words in the document to the number of times each one appears. For example, the phrase "make me a hot dog, a chili dog, and a bag of words" would become a 'bag' like: [a:3, and:1, bag:1, chili:1, dog:2, hot:1, make:1, me:1, of:1, words:1], with each word and its number of appearances. (We put the words in sorted order, but that's not necessary.) Note that the phrase has 13 words total but only 10 unique words (because "a" is repeated twice and "dog" once in the original phrase).

Specifically, you want to create an algorithm that—given a list of words $W$ containing $m$ words total but only $n$ unique words, where $n \leq m$—produces a complete list of tuples (pairs) of words and their numbers of occurrences. The result should have $n$ entries (exactly one for each unique word) and the total of the numbers of ocurrences across all entries should be $m$, but these entries can be in any order.

Most approaches we consider will use hash tables. For our purposes, such a hash table supports 6 operations. Here they are described for a standard hash table using chaining:

**ConstructHashTable($e$)** creates and returns a hash table of size $e$. (The hash table will be empty, but its underlying array will have $e$ entries. Takes time proportional to $e$.)

**Size($T$)** returns the number of keys in hash table $T$. (Takes constant time.)

**Contains($T$, $k$)** returns true if hash table $T$ contains an entry for the key $k$ and false otherwise. (Takes expected constant time, worst-case linear time.)

**Insert($T$, $k$, $v$)** inserts key $k$ with value $v$ into hash table $T$. If $k$ is already in $T$, overwrites its value with $v$. (Takes expected constant time, worst-case linear time.)

**Find($T$, $k$)** finds key $k$ in hash table $T$ and returns the value associated with it. If $k$ is not in $T$, produces an error instead. (Takes expected constant time, worst-case linear time.)

**Entries($T$)** returns a list of all the key/value pairs in hash table $T$. (If the table currently has $n$ keys (i.e., Size($T$) $= n$) and its underlying array has $e$ entries, this takes time proportional to $n + e$.)

# 1 Asymptotic Bound

Here is a correct approach to this problem using a standard hash table with chaining:

```
procedure ConvertToBag(W)
    T ← ConstructHashTable(|W|)          ▷ |W| is likely bigger than we need but not incorrect.
    for each word w in W do
        if Contains(T, w) then
            c_w ← Find(T, w)
            Insert(T, w, c_w + 1)
        else
            Insert(T, w, 1)
        end if
    end for
    return Entries(T)
end procedure
```

Give and **briefly** justify—including annotating the algorithm above to indicate how each part contributes to your bound—a good asymptotic bound on the **worst-case** runtime of a call to ConvertToBag in terms of $m$—the length of $W$, i.e., $m = |W|$—and $n$—the number of unique words in $W$.