Text Verification Problem
Possibly corrupted text file. Does it correspond to real text?

Given string $S$ of length $n$:

1. Determine if $S$ is "valid" text (i.e. made up of valid words)
2. If so, "reconstruct" $S$ - split into valid words

Assumptions:
1. All lowercase
2. No punctuation
3. Given a dictionary that takes as input string $w$:

$$\text{Dict}(w) = \begin{cases} 
  \text{true} & \text{if } w \text{ is a valid word (including "a" and "I")} \\
  \text{false} & \text{otherwise}
\end{cases}$$

Note: $S$ = "thinknot". 2 correct solutions: think not, thin knot

Example: $S$ = think nothing

Possible to solve in $O(n^2)$, $n = 151$
Text Verification Solution

Problem exhibits optimal substructure property so use D.P.

(1) ID array of size n+1, called A (index 0 is for Base Case).

(2) \( A[i,j] = \) validity (true or false) of prefix of \( S \) ending at \( i \) (i.e. \( S_1 \ldots i \))

Two ways that prefix ending at \( i \) can be valid:

- Word \( S_1 S_2 S_i \)
- OR
- \( S_i S_{k+1} \ldots S_i \)

(3) D.P. Formulation:

\[
A[i,j] = \begin{cases} 
    \text{true} & \text{if } i = 0 \text{ (Base Case)} \\
    \text{true} & \text{if } \overline{\text{Dict}(S_1 S_2 \ldots S_i)} = \text{true} \text{ and } i > 0 \\
    \text{true} & \text{if there exists } k \text{ s.t. } 1 \leq k < i \text{ such that } \\
    A[k] = \text{true (valid)} \text{ and } \\
    \overline{\text{Dict}(S_{k+1} \ldots i)} = \text{true (and } i > 0) \\
    \text{false} & \text{otherwise}
\end{cases}
\]

Base Case: if \( i = 0 \) \( \Rightarrow \) true (empty string)
(4) Fill A with for loop from $i = 1 \ldots n$.

(5) Answer (valid text?) in $A_{[n]}$.

(6) RunTime: $O(n^2)$ (for each entry, search previous)

(7) Actual solution (reconstructed text)?

ex: $O(\text{thinknothing})$
(7) Reconstruct text? (Assuming valid text)

1. Start at $A[E_{10}]$, if true.
2. Save $S_{k+1}...S_i$ (ex. $k=7$, save "thing")
3. Go to $A[E_k]$, Repeat

1. $i=n$
2. Go to $A[E_3]$, if true:
3.   if from Case 1
4.     Save $S_i...S_i$
5.   else //from Case 2
6.     Save $S_{k+1}...i$
7.     $i=k$ //Go to $A[E_k]$
8. Repeat from 2.

\[ i=1 \] \[ k=7 \] \[ \text{Save: "thing"} \]
\[ 7 \] \[ 5 \] \[ \text{No} \]
\[ 5 \] \[ \text{Think} \]
unathing
Lempel-Ziv

Drawbacks of Huffman's?
- must read entire file before encoding
- encodes only single characters (not strings)

Ex: AAA BAAA

Huffman's

```
          0
         / \
        A   B
        0   1
```

⇒ A = 0  B = 1
encoded: 0001000

Better: AAA = 0  B = 1  encoded: 010

Also, Huffman encodes: char → binary
In practice: char → ASCII → binary

Assume: have ASCII table (code for all 256 characters)

Goals
- encode text while reading
- encode strings instead of chars.

"sliding window" approach

Ex: the cat the hat

With fixed-length sliding window of size 2
encode: th, he, ec, ... (th already encoded) the, heh, eh...
But we want longer strings (not fixed-length)!  

So, start with window of size 2 (all strings of length 1 already have ASCII codes) and extend the window

when should we extend?  
⇒ When the current string has already been encoded

ex: the cat the hat

  th - encode

  he - "

  ec - "

  th - already encoded, extend window, encode "the"

  eh - encode

Frequently checking whether a string has a code
How to do this quickly?

Store strings, codes in a hash table/dictionary

Recall: All single chars are already in the table