Using Dictionaries: Examples
Example: Efficiency of Dictionaries

We want to generate 50,000 distinct random integers in the range $[1, 100,000]$. 

**ALGORITHM**

1. Start with an empty list $L$.
2. Pick a random integer $r$ in $[1, 100,000]$
3. Append to $L$ if $r$ is not in $L$
4. **STOP** if $\text{len}(L)$ is 50,000; otherwise return to (2).
Using Dictionaries makes a huge difference

- Step (3) in which we are repeatedly testing if \( r \) is in \( L \) is extremely inefficient.

- In situations such as this, we should use Python dictionaries instead.

- This is even though we only care about the keys and there is no notion of associated values.

- List based implementation: 36.4886 s
- Dictionary-based implementation: 0.1781 s
def makeRandomList():
    D = {}
    while len(D) < 50000:
        r = random.randint(1, 100000)
        if r not in D:
            D[r] = 0
    return D
The word Network Example

- The *word network* will be stored in two structures:

  o **wordList**: the list of 5757 words

  o **neighborDict**: a dictionary whose keys are elements of wordList and the value of each key w is the list of neighbors of w.

- **Example**: So neighborDict will look like {“aargh”: [], “abaca”: [“abaci”, “aback”], “abaci”: [“abaca”, “aback”],...}
for i in range(len(wordList)):
    for j in range(i+1, len(wordList)):
        if areNeighbors(wordList[i], wordList[j]):
            neighborDict[wordList[i]].append(wordList[j])
            neighborDict[wordList[j]].append(wordList[i])
Using `neighborDict`: Examples

- Write a function that finds the word with maximum number of neighbors.
- Write a function that finds all words that are exactly two hops away from a given word.