The word ladders game: Version 2.0
Algorithm for searching a network

- First we will focus only on determining if there is a path from the source word to the target word.
- We will think about recording such a path later.

**Idea:**
- If we are at a particular word, we will then examine all neighbors of that word that have not already been examined.
- These new neighbors will be stored so that we can examine their neighbors later.
More details

- A word is said to have been processed if we have *examined* all its neighbors.

- We will maintain two sets of words:
  - *reached*: these are words that have been reached in the exploration, but not processed yet.
  - *processed*: these are words that have been reached and have also been processed.
Typical step

- In the “typical step” of the algorithm we will pick out a word from the reached set and process it.

- Pseudocode:
  - Pick an arbitrary word $w$ from the reached set
  - For each neighbor of $w$:  
    - If neighbor has not already been reached or processed then add the neighbor to the reached set.
  - Add $w$ to processed set.
Stopping conditions

- One of two things have to happen for the algorithm to stop:
  - If target is found, i.e., if target enter the reached set then we have detected a path from source to target.
  - If the reached set becomes empty, i.e., there is nothing left to explore, then there is no path from source to target.
def searchWordNetwork(source, target, D):
    # The values of keys are not useful at this stage of the program and so we use 0 as dummy values.
    processed = {source:0}
    reached = {}
    for e in D[source]:
        reached[e] = 0

    # Repeat until reached set becomes empty or target is reached
    while reached:
        if target in reached:
            return True
        # Pick an item in reached and process it
        item = reached.popitem() # returns an arbitrary key-value pair as a tuple
        newWord = item[0]

        # Find all neighbors of this item and add new neighbors to reached
        processed[newWord] = 0
        for neighbor in D[newWord]:
            if neighbor not in reached and neighbor not in processed:
                reached[neighbor] = 0

    return False