Word Break

Core Idea (DP):

Define dp[i] = true iff the prefix s[0...i] can be segmented.

Transition:

- For each i from 1...n, set dp[i] = true if there exists a j<i such that:
  - dp[j] == true
  - s[j...i] is in wordDict.

Answer is dp[n].

How to make it fast:

- Precompute a set from wordDict for O(1) membership.
- Let Lmin = min length of words, Lmax = max length of words.
  - Only check j where i-j is in [Lmin, Lmax]. This prunes a ton.
- Break early when you set dp[i] = true.

Complexity:

- Worst case ~ O(n * (# lengths checked)) = O(n * (Lmax - Lmin + 1)), with O(n) space.

Alternative: BFS on indices:

Think of indices as nodes. From index i, you can jump to i+len(w) for any word w that matches s[i...i+len(w)]. Use a queue and a visited boolean array to avoid revisiting i. Return true if you ever reach n. This often performs similarly to DP.

Small walkthrough:

s = "applepenapple", dict = {apple, pen}.

- Lmin = 3, Lmax = 5
- dp[5] becomes true via "apple", then dp[8] via "pen", then dp[13] via "apple".
  Finial dp[n] = true.

Extra optimization (optional):

- Groups dictionary words by length; iterate only those lengths.
- If you like tries, you can traverse a trie from each i and mark dp[end] when a terminal node is hit, but the simple DP is usually enough here.