127. Word Ladder
Hard

A transformation sequence from word `beginWord` to word `endWord` using a dictionary wordList is a sequence of words `beginWord -> s1 -> s2 -> ... -> sk` such that:

- Every adjacent pair of words differs by a single letter.
- Every `si` for `1 <= i <= k` is in `wordList`. Note that `beginWord` does not need to be in wordList.
- `sk == endWord`

Given two words, `beginWord` and `endWord`, and a dictionary `wordList`, return the number of words in the shortest transformation sequence from `beginWord` to `endWord`, or 0 if no such sequence exists.

```
Example 1:

Input: beginWord = "hit", endWord = "cog",
wordList = ["hot","dot","dog","lot","log","cog"]
Output: 5
Explanation: One shortest transformation sequence
is "hit" -> "hot" -> "dot" -> "dog" -> cog", which
is 5 words long.

Example 2:

Input: beginWord = "hit", endWord = "cog",
wordList = ["hot","dot","dog","lot","log"]
Output: 0
Explanation: The endWord "cog" is not in wordList,
therefore there is no valid transformation
sequence.
```

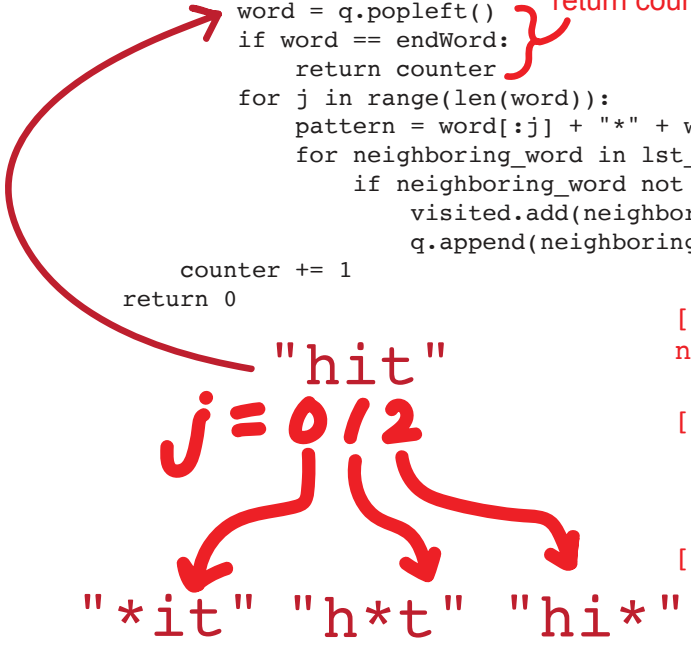
```python
import collections
class Solution:
    def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) -> int:
        if endWord not in wordList:
            return 0

        lst_of_neighbors = collections.defaultdict(list)

        wordList.append(beginWord)

        for word in wordList:
            for j in range(len(word)):
                pattern = word[:j] + "*" + word[j + 1 :]
                lst_of_neighbors[pattern].append(word)

        visited = set([beginWord])
        q = deque([beginWord])
        counter = 1

        while q:
            for i in range(len(q)):
                word = q.popleft()
                if word == endWord:
                    return counter
                for j in range(len(word)):
                    pattern = word[:j] + "*" + word[j + 1 :]
                    for neighboring_word in lst_of_neighbors[pattern]:
                        if neighboring_word not in visited:
                            visited.add(neighboring_word)
                            q.append(neighboring_word)
            counter += 1
        return 0
```

```
lst_of_neighbors = {
    '*ot': ['hot', 'dot', 'lot'],
    'h*t': ['hot', 'hit'],
    'ho*': ['hot'],
    'd*t': ['dot'],
    'do*': ['dot', 'dog'],
    '*og': ['dog', 'log', 'cog'],
    'd*g': ['dog'],
    'l*t': ['lot'],
    'lo*': ['lot', 'log'],
    'l*g': ['log'],
    'c*g': ['cog'],
    'co*': ['cog'],
    '*it': ['hit'],
    'hi*': ['hit']
}
```

if popped word is equal to end word, then the algorithm ends return counter

create pattern

reference `lst_of_neighbors[pattern]`



[1] Create an adjacency list of possible neighbors patterns

[2] visited -> stores visited combinations
q -> queue for permutations
counter -> result

[3] bfs algorithm