Words Worth Finding:

Solving Word Search 2

By: Donal Moloney, Zirui Huang, Berker Ustura

Lost in a Sea of Letters: *Project Overview*

Project Goal

- Implement three approaches to solve the Word Search II Leetcode Hard Problem
- Compare the performance of each.

The three approaches:

1. Backtracking with Trie
2. DFS with Trie
3. DFS with Set

Finding Needles in a Haystack:

Problem Description

- Objective: Find all the words that can be constructed from adjacent cells.
- Constraints:
- Words are formed from adjacent cells on the board
- Each letter on the board is only used once in a single word.

- Expected Input:
  - 2D matrix of chars representing the board
  - Strings array representing list of words.
- Expected Output: Array of strings representing words found on the board.

Example:

Given Board:

[['c','a','t','s'],

['o','d','g','e'],

['e','a','t','i'],

['f','e','e','l']]

and Given Words:

["cat", "dog", "eat", "feel"]

Expected output: ["cat", "dog", "eat", "feel"].

Putting the Pieces Together: *Implementation Details*

Solution 1:

Backtracking With Trie

Approach:

- Backtracking with Trie data structure

Algorithm:

- Starting from each cell, explore all possible paths & use a Trie to efficiently check if the path is a valid word.

Time Complexity:

- O(M * (4*3^(L-1)))
- M (number of cells in the board) and L (maximum length of a word in the list of words).

Space Complexity:

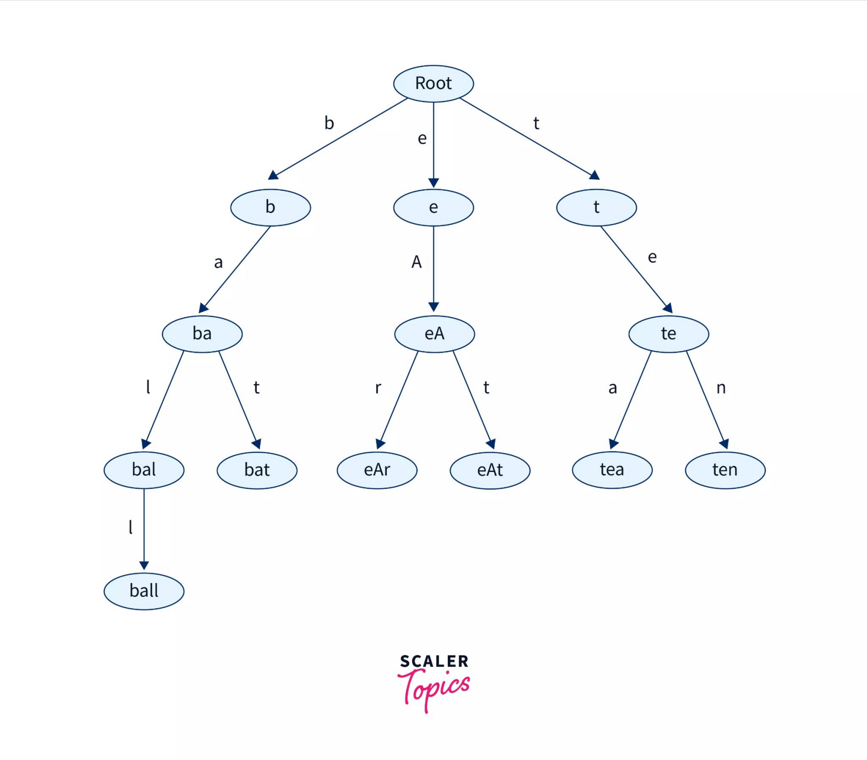
- O(N), N total number of letters in the word list.

Key Features:

- Trie allows us to efficiently check if a path is a valid word
- Backtracking helps explore all possible paths without revisiting same cell.

Solution 2:

DFS with Trie



Approach:

- Depth-first search (DFS) with Trie data structure

Algorithm:

- At each cell, explore all possible paths using DFS & use a Trie to check if path is a valid word.

Time Complexity:

- O(M(4⋅3^(L−1)))
- M (number of cells in the board), and L (max length of a word in the list of words).

Space Complexity:

- O(N + M)
- N is total number of letters in the word list, M is the number of cells in the board.

Key Features:

- Approach can simplify the code compared to backtracking; Trie allows checking if a path is a valid word.

Solution 3:

DFS With Set

Approach:

- DFS with SET data structure

Algorithm:

- At each cell, explore all possible paths using DFS with a Set to store the words list found on board.

Time Complexity:

- O(M * 4^L)
- M (number of cells in the board) and L (max length of a word in list of words).

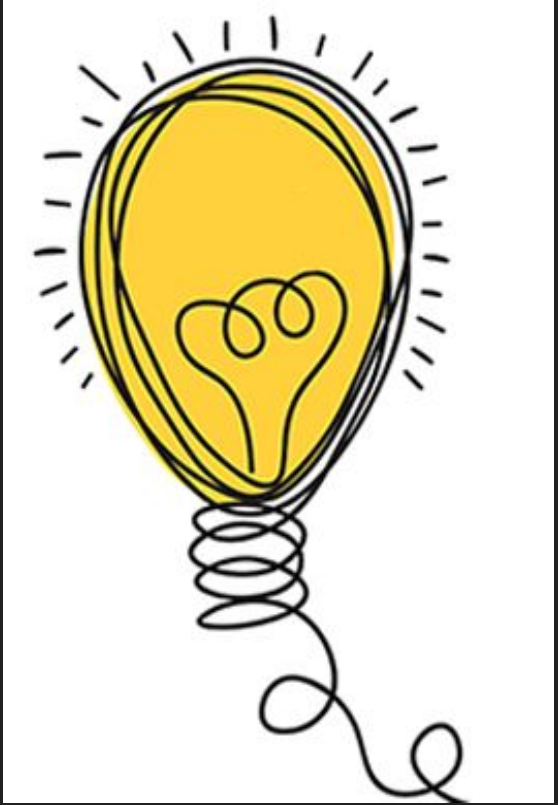
Space Complexity:

- O(N), N total number of letters in the word list.

Key Features:

- DFS / Set can simplify the code compared to backtracking or using a Trie
- Set allows the store of list of words found on the board without requiring a Trie.

PROS
CONS



Backtracking with Trie Pros / Cons

Pros:

- Can be faster than the DFS approaches for smaller boards and lists of words.
- Trie allows for efficient checking if path forms valid word.
- Adaptable to solve various word search problems.

Cons:

- May not find all possible words on the board.
- Can be slow & memory intensive for large boards & words.
- Needs careful management of the Trie to allow correct update when backtracking.

Comparison with DFS with Trie:

- DFS with Trie may be slower for smaller boards and lists of words.
- DFS with Trie may be easier to understand
- DFS with Trie less memory intensive (doesn’t need to store multiple paths).

Comparison with DFS with Set:

- DFS with Set slower for smaller boards & lists of words.
- DFS with Set may be simpler to implement.
- DFS with Set may be more memory-efficient.
- DFS with Set may not find all possible words on the board.

DFS with Trie Pros / Cons

Pros:

- Simpler implementation than backtracking.
- Efficient checking if path forms a valid word.
- Adaptable to solve various word search problems.

Cons:

- May not find all possible words on the board.
- Can be slow & use a lot of memory for larger boards / lists of words.
- During DFS traversal careful management needed to ensure Trie updates correctly.

Comparison with Backtracking with Trie:

- DFS with Trie may be easier to understand and implement.
- Backtracking with Trie may be faster for smaller boards and lists of words.
- DFS with Trie may require less memory.

Comparison with DFS with Set:

- DFS with Trie may be faster for larger boards and lists of words.
- DFS with Trie may have better prefix search capabilities.
- DFS with Trie may be harder implement.
- DFS with Set may not find all possible words on board.

DFS & Set Pros / Cons

Pros:

- Simple & easy to implement compared to other.
- Uses less memory than the Trie approach.
- Adaptable to solve various word search problems.

Cons:

- May not find all possible words on board.
- Can be slow for larger boards / word lists.
- Set doesn’t allow efficient prefix search.
- Unsuitable for frequently updated word lists.

Comparison with Backtracking with Trie:

- DFS with Set may be simpler to implement.
- DFS with Set may be slower for smaller boards & lists of words.
- DFS with Set may be more memory-efficient.

Comparison with DFS with Trie:

- DFS with Set may be simpler to implement.
- DFS with Trie may be faster for larger boards and lists of words.
- DFS with Trie may have better prefix search capabilities.

Real Time of Each Algorithm

We ran two separate tests for each algorithm, using a short and long input. The times are listed in the following table (runtime measurements is in ns):

	Short	**Long**
Backtracking w/ Trie	35500	42600
DFS w/ Trie	15700	32800
DFS w/ Set	208500	292569774800

Final Considerations

- Each solution has strengths & weaknesses.
- Backtracking with Trie (BTT): good for finding all possible words on the board & can be highly efficient for small to medium-sized boards / lists of words.
- DFS with Trie (DFST): is simpler to implement than BTT & can be very efficient for small to medium-sized boards / lists of words.
- DFS with Set (DFSS): easiest to implement; good for finding a subset of words on the board, may not explore all possible paths & slow for larger boards or words list.
- The best option depends on: board size, word size, freq. of updates to words etc.
- Understanding the pros & cons helps choose the correct solution for the job.

The End