Description | Editorial | Solutions | Submissions

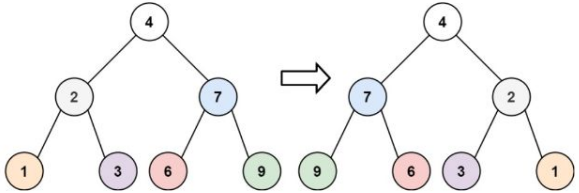
226. Invert Binary Tree

Solved

Easy | Topics | Companies

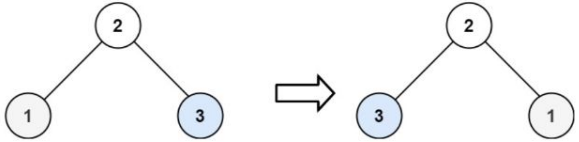
Given the `root` of a binary tree, invert the tree, and return *its root*.

Example 1:




```
Input: root = [4,2,7,1,3,6,9]
Output: [4,7,2,9,6,3,1]
```

Example 2:




```
Input: root = [2,1,3]
Output: [2,3,1]
```

Example 3:

```
Input: root = []
Output: []
```

Code

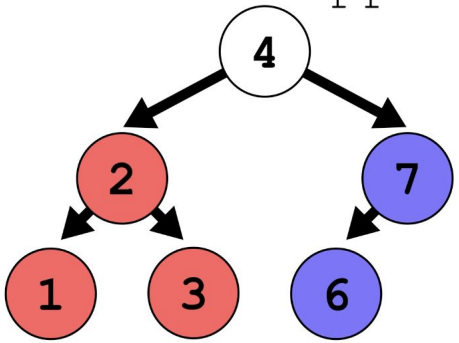
Python3 • Auto

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def invertTree(self, root: Optional[TreeNode]) -> Optional[TreeNode]:

```

Saved

Ln 1, Col 1

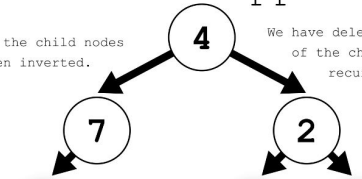
Recursive Approach

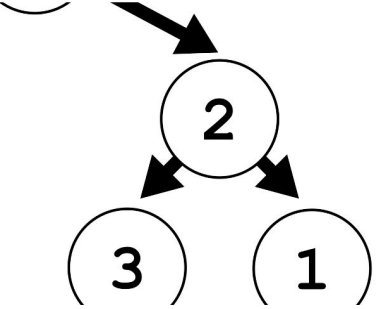


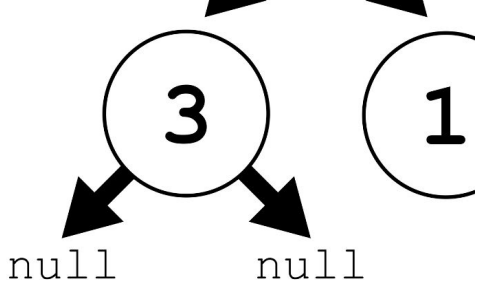
Recursive Approach

Assume that the child nodes have been inverted.

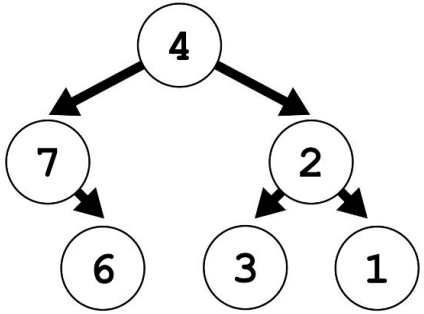
We have delegated the inversion of the child trees to our recursive calls.

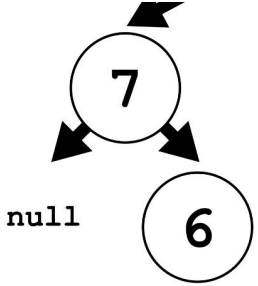






Play





Tree

Run Submit

Description | Editorial | Solutions | Submissions

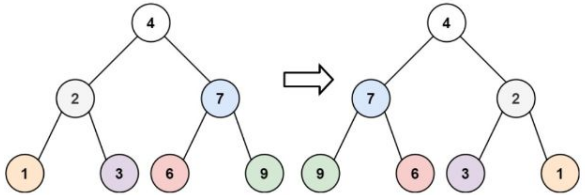
226. Invert Binary Tree

Solved

Easy | Topics | Companies

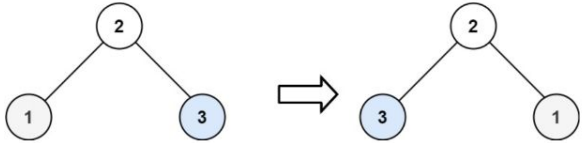
Given the `root` of a binary tree, invert the tree, and return *its root*.

Example 1:




```
Input: root = [4,2,7,1,3,6,9]
Output: [4,7,2,9,6,3,1]
```

Example 2:




```
Input: root = [2,1,3]
Output: [2,3,1]
```

Example 3:

```
Input: root = []
Output: []
```

Code

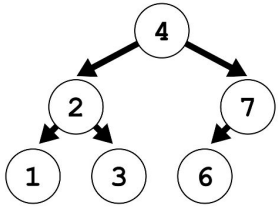
Python3 • Auto

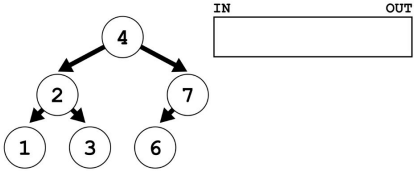
```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def invertTree(self, root: Optional[TreeNode]) -> Optional[TreeNode]:
        if not root:
            return None

        right = self.invertTree(root.right)
        left = self.invertTree(root.left)
        root.left = right
        root.right = left
        return root
```

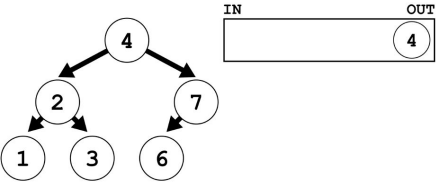
Saved

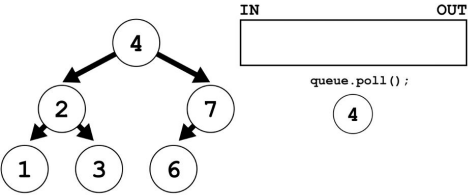
Ln 16, Col 20

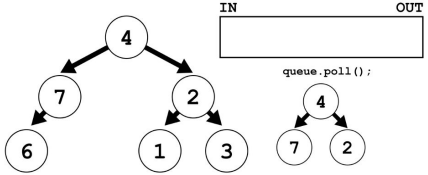
Iterative Approach

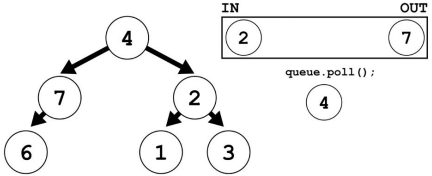


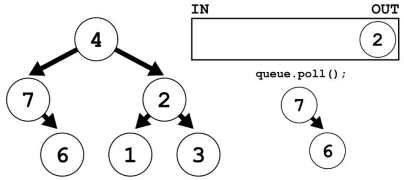


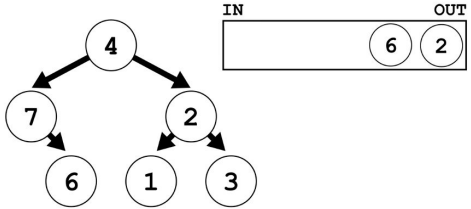


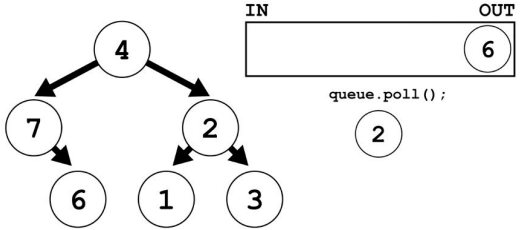


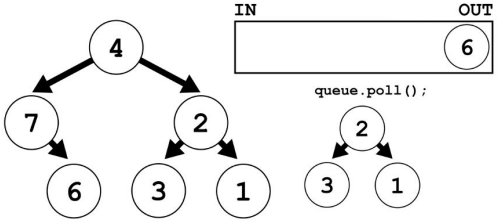


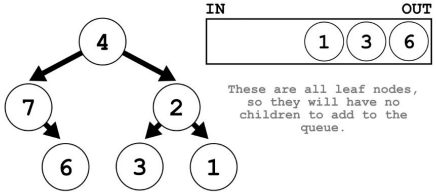












These are all leaf nodes, so they will have no children to add to the queue.

Tree

Run Submit

Description | Editorial | Solutions | Submissions

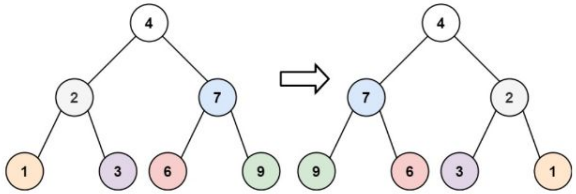
226. Invert Binary Tree

Solved

Easy | Topics | Companies

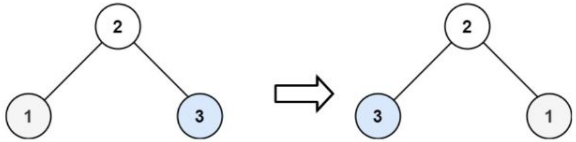
Given the `root` of a binary tree, invert the tree, and return *its root*.

Example 1:




```
Input: root = [4,2,7,1,3,6,9]
Output: [4,7,2,9,6,3,1]
```

Example 2:




```
Input: root = [2,1,3]
Output: [2,3,1]
```

Example 3:

```
Input: root = []
Output: []
```

14.2K 142

Code

Python3 • Auto

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def invertTree(self, root: Optional[TreeNode]) -> Optional[TreeNode]:
        if not root:
            return None

        queue = collections.deque([root])
        while queue:
            current = queue.popleft()
            current.left, current.right = current.right, current.left

            if current.left:
                queue.append(current.left)

            if current.right:
                queue.append(current.right)

        return root
```

Saved

Ln 23, Col 20

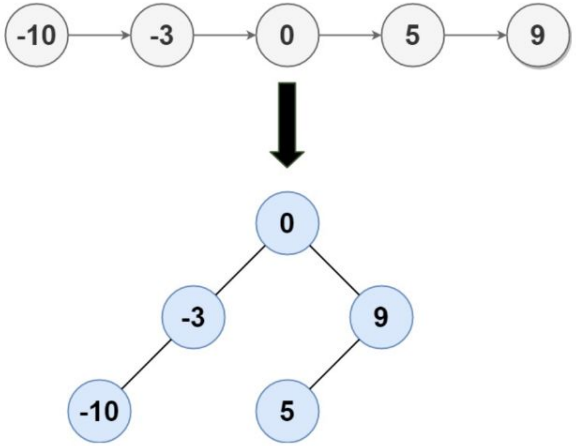
Testcase | Test Result

109. Convert Sorted List to Binary Search Tree

Medium | Topics | Companies

Given the `head` of a singly linked list where elements are sorted in **ascending order**, convert *it to a* ***height-balanced*** *binary search tree.*

Example 1:




```
Input: head = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: One possible answer is [0,-3,9,-10,null,5], which represents the shown height balanced BST.
```

Example 2:

Code

Python3 • Auto

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def sortedListToBST(self, head: Optional[ListNode]) -> Optional[TreeNode]:

```

Tree

Run Submit 0

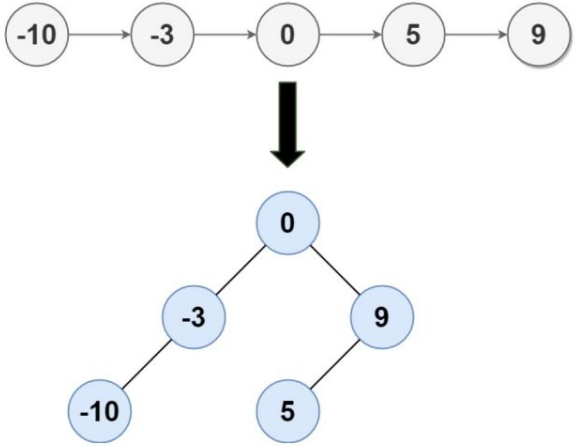
Description | Editorial | Solutions | Submissions

109. Convert Sorted List to Binary Search Tree

Medium | Topics | Companies

Given the `head` of a singly linked list where elements are sorted in **ascending order**, convert *it to a* ***height-balanced*** *binary search tree.*

Example 1:




```
Input: head = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: One possible answer is [0,-3,9,-10,null,5], which represents the shown
height balanced BST.
```

Example 2:

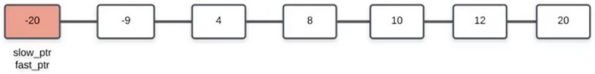
Code

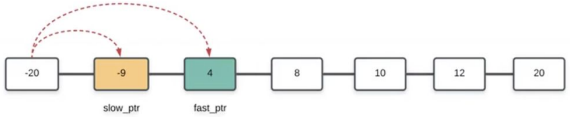
Python3 • Auto

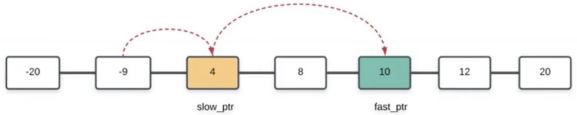
```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def sortedListToBST(self, head: Optional[ListNode]) -> Optional[TreeNode]:

```

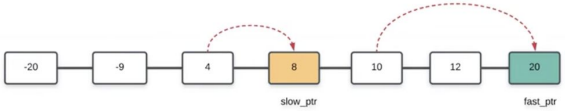
Saved

Ln 1, Col 1







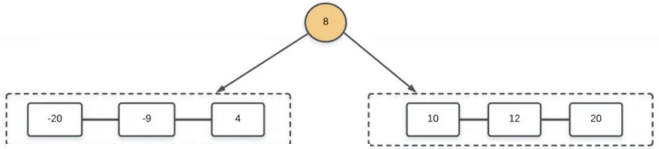


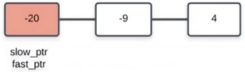
Root of the BST

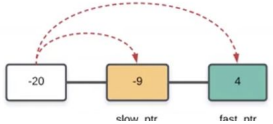
Left Half

Root of the BST

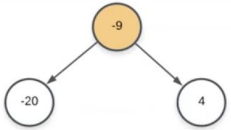
Right Half

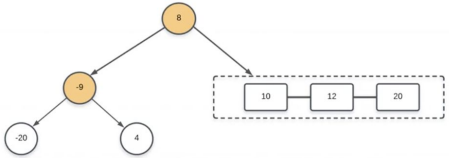


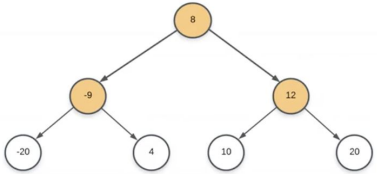




Middle Element







Tree

Run Submit 0

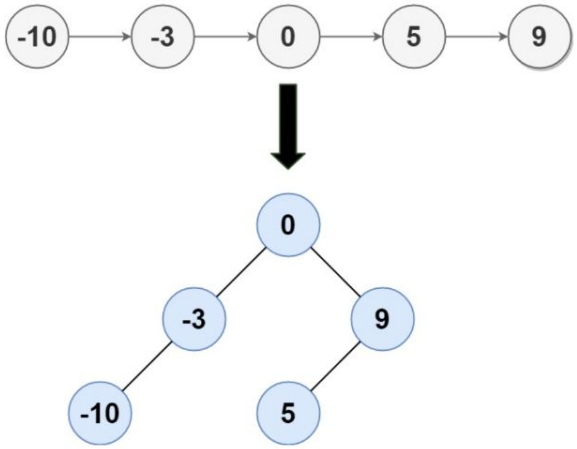
Description | Editorial | Solutions | Submissions

109. Convert Sorted List to Binary Search Tree

Medium | Topics | Companies

Given the `head` of a singly linked list where elements are sorted in **ascending order**, convert *it to a* ***height-balanced*** *binary search tree.*

Example 1:




```
Input: head = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: One possible answer is [0,-3,9,-10,null,5], which represents the shown
height balanced BST.
```

Example 2:

Code

Python3 • Auto

```python
class Solution:
    def findMiddle(self, head: ListNode) -> ListNode:
        # The pointer used to disconnect the left half from the mid node.
        prevPtr = None
        slowPtr = head
        fastPtr = head
        # Iterate until fastPr doesn't reach the end of the linked list.
        while fastPtr and fastPtr.next:
            prevPtr = slowPtr
            slowPtr = slowPtr.next
            fastPtr = fastPtr.next.next

        # Handling the case when slowPtr was equal to head.
        if prevPtr:
            prevPtr.next = None
        return slowPtr
    def sortedListToBST(self, head: ListNode) -> TreeNode:
        # If the head doesn't exist, then the linked list is empty
        if not head:
            return None
        mid = self.findMiddle(head)
        # The mid becomes the root of the BST.
        node = TreeNode(mid.val)

        # Base case when there is just one element in the linked list
        if head == mid:
            return node

        # Recursively form balanced BSTs using the left and right halves of the original list.
        node.left = self.sortedListToBST(head)
        node.right = self.sortedListToBST(mid.next)
        return node
```

- This disconnects the **left half** of the list from the middle.
- Important for dividing the list into two halves for recursion.

- Base case: if the list has only **one node**, that node becomes the root and recursion ends.

Saved

Ln 24, Col 1

Description | Editorial | Solutions | Submissions

208. Implement Trie (Prefix Tree)

Medium | Topics | Companies

A **trie** (pronounced as "try") or **prefix tree** is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. There are various applications of this data structure, such as autocomplete and spellchecker.

Implement the Trie class:

- `Trie()` Initializes the trie object.
- `void insert(String word)` Inserts the string `word` into the trie.
- `boolean search(String word)` Returns `true` if the string `word` is in the trie (i.e., was inserted before), and `false` otherwise.
- `boolean startsWith(String prefix)` Returns `true` if there is a previously inserted string `word` that has the prefix `prefix`, and `false` otherwise.

Example 1:

```
Input
["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
[[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
Output
[null, null, true, false, true, null, true]

Explanation
Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // return True
trie.search("app");     // return False
trie.startsWith("app"); // return True
trie.insert("app");
trie.search("app");     // return True
```

Code

Python3 • Auto

```python
class Trie:

    def __init__(self):


    def insert(self, word: str) -> None:


    def search(self, word: str) -> bool:


    def startsWith(self, prefix: str) -> bool:



# Your Trie object will be instantiated and called as such:
# obj = Trie()
# obj.insert(word)
# param_2 = obj.search(word)
# param_3 = obj.startsWith(prefix)
```

Saved

Ln 1, Col 1

Problem List

Run Submit

Description | Editorial | Solutions | Submissions

208. Implement Trie (Prefix Tree)

Medium | Topics | Companies

A **trie** (pronounced as "try") or **prefix tree** is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. There are various applications of this data structure, such as autocomplete and spellchecker.

Implement the Trie class:

- `Trie()` Initializes the trie object.
- `void insert(String word)` Inserts the string `word` into the trie.
- `boolean search(String word)` Returns `true` if the string `word` is in the trie (i.e., was inserted before), and `false` otherwise.
- `boolean startsWith(String prefix)` Returns `true` if there is a previously inserted string `word` that has the prefix `prefix`, and `false` otherwise.

Example 1:

```
Input
["Trie", "insert", "search", "search", "startsWith", "insert", "search"]
[[], ["apple"], ["apple"], ["app"], ["app"], ["app"], ["app"]]
Output
[null, null, true, false, true, null, true]

Explanation
Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // return True
trie.search("app");     // return False
trie.startsWith("app"); // return True
trie.insert("app");
trie.search("app");     // return True
```

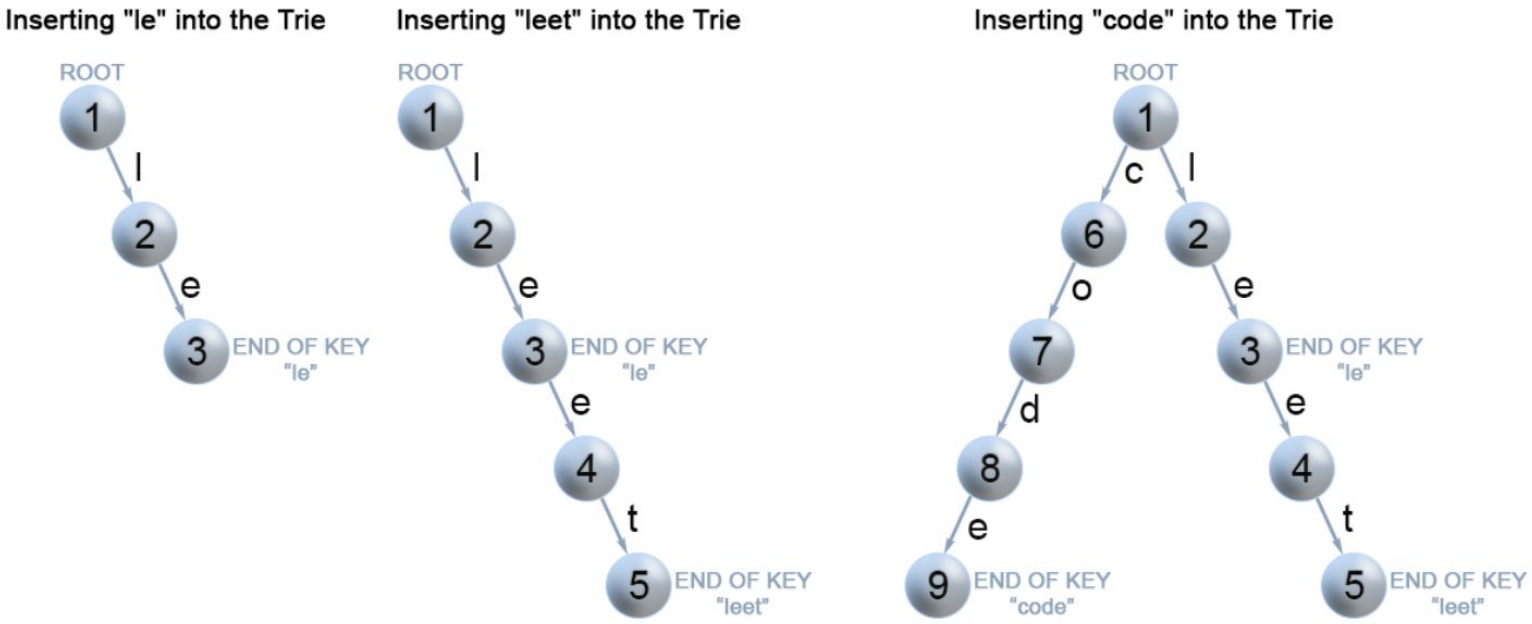
Code

Python3 • Auto

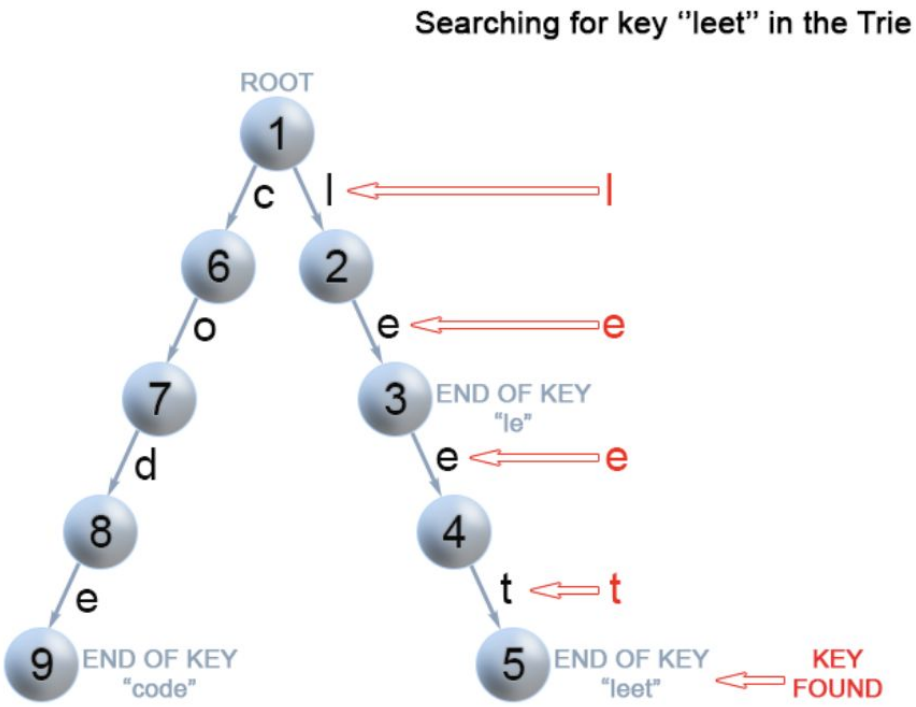
```python
class Trie:

    def __init__(self):


    def insert(self, word: str) -> None:


    def search(self, word: str) -> bool:


    def startsWith(self, prefix: str) -> bool:



# Your Trie object will be instantiated and called as such:
# obj = Trie()
# obj.insert(word)
# param_2 = obj.search(word)
# param_3 = obj.startsWith(prefix)
```

Saved

Ln 1, Col 1

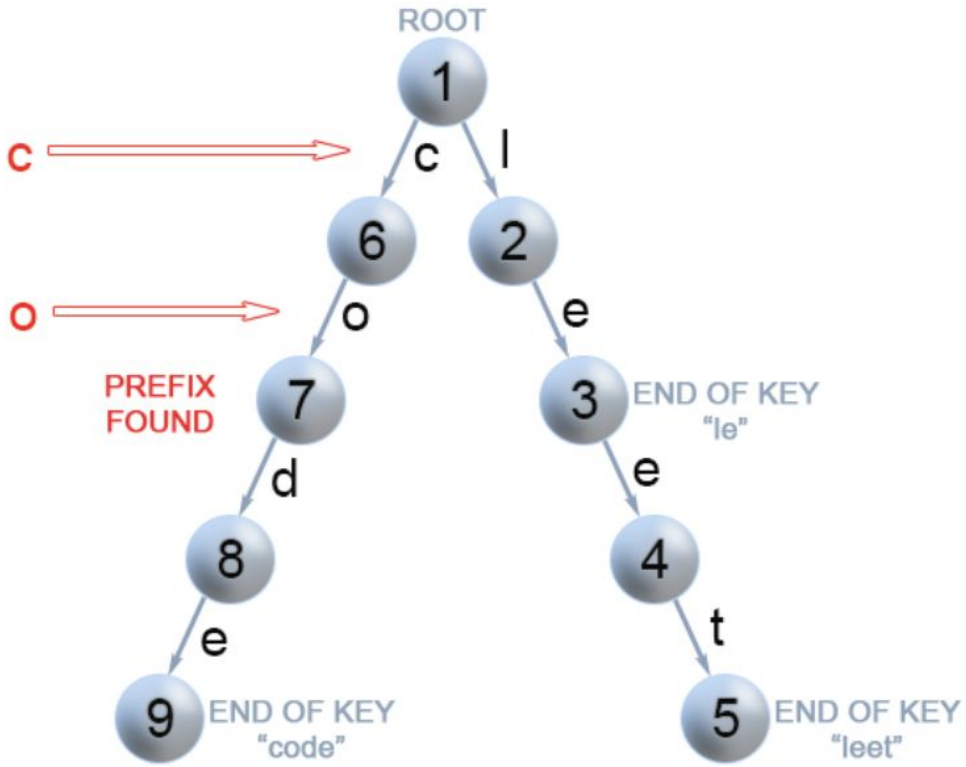


Building a Trie from dataset {le, leet, code}



Searching for a key in a Trie from dataset {le, leet, code}

Searching for "co" in the Trie



Searching for a prefix in a Trie from dataset {le, leet, code}

Problem List

Run Submit

Description | Editorial | Solutions | Submissions

208. Implement Trie (Prefix Tree)

Medium | Topics | Companies

A **trie** (pronounced as "try") or **prefix tree** is a tree data structure used to efficiently store and retrieve keys in a dataset of strings. There are various applications of this data structure, such as autocomplete and spellchecker.

Implement the Trie class:

- `Trie()` Initializes the trie object.
- `void insert(String word)` Inserts the string `word` into the trie.
- `boolean search(String word)` Returns `true` if the string `word` is in the trie (i.e., was inserted before), and `false` otherwise.
- `boolean startsWith(String prefix)` Returns `true` if there is a previously inserted string `word` that has the prefix `prefix`, and `false` otherwise.

Example 1:

```
Input
["Trie", "insert", "search", "search", "startsWith",
"insert", "search"]
[[], ["apple"], ["apple"], ["app"], ["app"], ["app"],
["app"]]
Output
[null, null, true, false, true, null, true]

Explanation
Trie trie = new Trie();
trie.insert("apple");
trie.search("apple");   // return True
trie.search("app");     // return False
trie.startsWith("app"); // return True
```

Code

Python3 • Auto

```python
class TrieNode:
    def __init__(self, char = ""):
        self.char = char
        self.children = {}
        self.is_end = False
        # self.counter = 0
class Trie:

    def __init__(self):
        self.root = TrieNode()

    def insert(self, word: str) -> None:
        node = self.root
        for char in word:
            if char in node.children:
                node = node.children[char]
            else:
                new_node = TrieNode(char)
                node.children[char] = new_node
                node = new_node
        node.is_end = True
        # node.counter += 1


    def search(self, word: str) -> bool:
        node = self.root
        for char in word:
            if char not in node.children:
                return False
            node = node.children[char]

        # Reached at the end of word
        # return True if word is present, i.e is_end = True else False
        return node.is_end

    def startsWith(self, prefix: str) -> bool:
        """
        Returns if there is any word in the trie that starts with the given prefix.
        """
        node = self.root
        for char in prefix:
            if char not in node.children:
                return False
            node = node.children[char]
        return True
```

Saved

Ln 8, Col 1

Problem List

Run Submit

Description | Editorial | Solutions | Submissions

215. Kth Largest Element in an Array

Solved

Medium Topics Companies

Given an integer array `nums` and an integer `k`, return *the* k^{th} *largest element in the array*.

Note that it is the kth largest element in the sorted order, not the kth distinct element.

Can you solve it without sorting?

Example 1:

```
Input: nums = [3,2,1,5,6,4], k = 2
Output: 5
```

Example 2:

```
Input: nums = [3,2,3,1,2,4,5,5,6], k = 4
Output: 4
```

Constraints:

- $1 \le k \le nums.length \le 10^5$
- $-10^4 \le nums[i] \le 10^4$

Seen this question in a real interview before? 1/5

Yes No

Accepted 2.6M | Submissions 3.9M | Acceptance Rate 67.2%

Topics

17.4K 237

Code

Python3 • Auto

```python
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:

```

Saved Ln 1, Col 1

Testcase | Test Result

Problem List

Run Submit

Description | Editorial | Solutions | Submissions

215. Kth Largest Element in an Array

Solved

Medium | Topics | Companies

Given an integer array `nums` and an integer `k`, return *the* k^{th} *largest element in the array*.

Note that it is the kth largest element in the sorted order, not the kth distinct element.

Can you solve it without sorting?

Example 1:

```
Input: nums = [3,2,1,5,6,4], k = 2
Output: 5
```

Example 2:

```
Input: nums = [3,2,3,1,2,4,5,5,6], k = 4
Output: 4
```

Constraints:

- $1 \le k \le nums.length \le 10^5$
- $-10^4 \le nums[i] \le 10^4$

Seen this question in a real interview before? 1/5

Yes | No

Accepted 2.6M | Submissions 3.9M | Acceptance Rate 67.2%

Topics

17.4K 237

Code

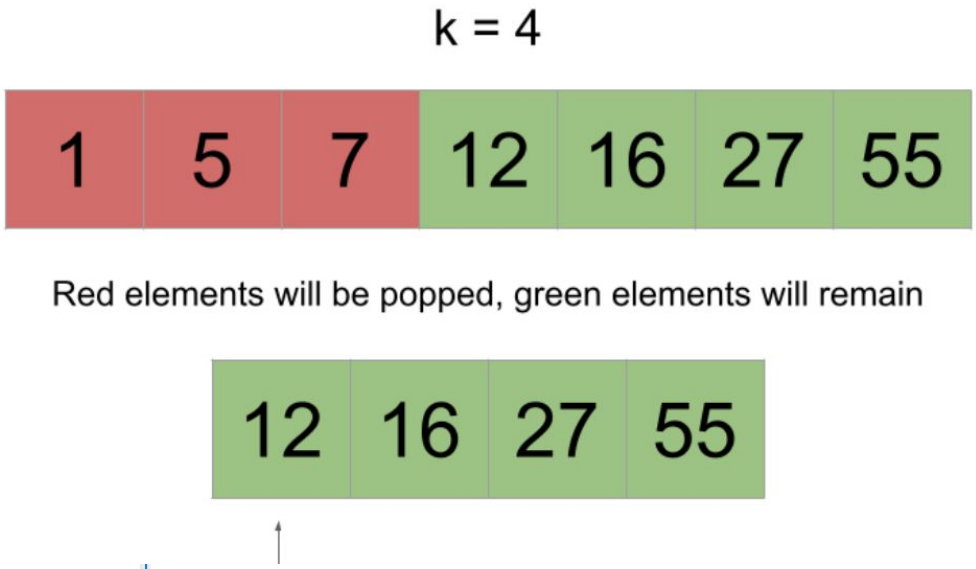
Python3 • Auto

```python
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:

```

Saved

Ln 1, Col 1

Testcase | Test Result



Problem List

Run Submit

Description | Editorial | Solutions | Submissions

215. Kth Largest Element in an Array

Solved

Medium | Topics | Companies

Given an integer array `nums` and an integer `k`, return *the* k^{th} *largest element in the array*.

Note that it is the kth largest element in the sorted order, not the kth distinct element.

Can you solve it without sorting?

Example 1:

```
Input: nums = [3,2,1,5,6,4], k = 2
Output: 5
```

Example 2:

```
Input: nums = [3,2,3,1,2,4,5,5,6], k = 4
Output: 4
```

Constraints:

- $1 <= k <= nums.length <= 10^5$
- $-10^4 <= nums[i] <= 10^4$

Seen this question in a real interview before? 1/5

Yes | No

Accepted 2.6M | Submissions 3.9M | Acceptance Rate 67.2%

Code

Python3 • Auto

```python
class Solution:
    def findKthLargest(self, nums, k):
        heap = []
        for num in nums:
            heapq.heappush(heap, num)
            if len(heap) > k:
                heapq.heappop(heap)

        return heap[0]
```

Saved Ln 9, Col 23

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums =

[3,2,1,5,6,4]

Problem List

Run Submit

Description | Editorial | Solutions | Submissions

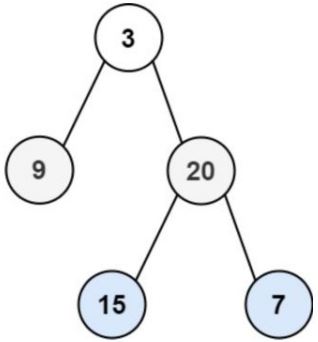
102. Binary Tree Level Order Traversal

Solved

Medium | Topics | Companies

Given the `root` of a binary tree, return *the level order traversal of its nodes' values*. (i.e., from left to right, level by level).

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: [[3],[9,20],[15,7]]
```

Example 2:

```
Input: root = [1]
Output: [[1]]
```

Example 3:

```
Input: root = []
Output: []
```

Code

Python3 • Auto

```python
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right
class Solution:
    def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:

```

Saved

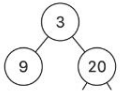
Ln 1, Col 1

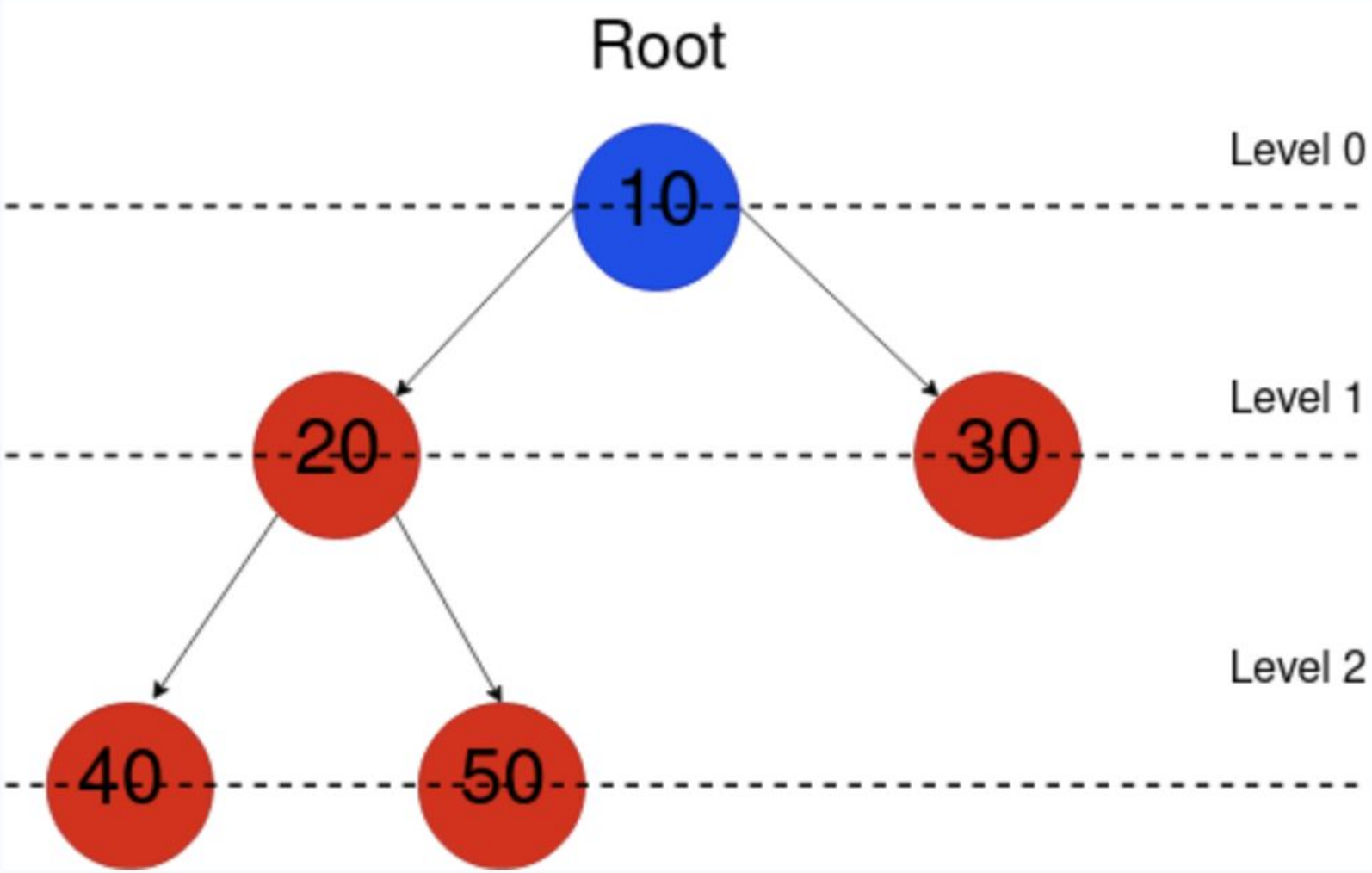
Testcase | Test Result

Case 1 | Case 2 | Case 3

root =

```
[3,9,20,null,null,15,7]
```




Root
Level 0
10
Level 1
20
30
Level 2
40
50

Problem List

Run Submit

Description | Editorial | Solutions | Submissions

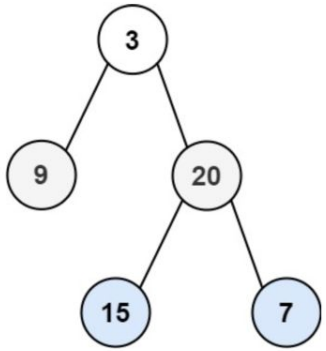
102. Binary Tree Level Order Traversal

Solved

Medium | Topics | Companies

Given the `root` of a binary tree, return *the level order traversal of its nodes' values*. (i.e., from left to right, level by level).

Example 1:




```
Input: root = [3,9,20,null,null,15,7]
Output: [[3],[9,20],[15,7]]
```

Example 2:

```
Input: root = [1]
Output: [[1]]
```

Example 3:

```
Input: root = []
```

15.6K 99

Code

Python3 • Auto

```python
class Solution:
    def levelOrder(self, root: TreeNode) -> List[List[int]]:
        levels = []
        if not root:
            return levels
        level = 0
        queue = deque(
            [
                root,
            ]
        )
        while queue:
            # start the current level
            levels.append([])
            # number of elements in the current level
            level_length = len(queue)
            for i in range(level_length):
                node = queue.popleft()
                # fulfill the current level
                levels[level].append(node.val)
                # add child nodes of the current level
                # in the queue for the next level
                if node.left:
                    queue.append(node.left)
                if node.right:
                    queue.append(node.right)

            # go to next level
            level += 1

        return levels
```

Saved

Ln 21, Col 47

Testcase | Test Result

Accepted Runtime: 0 ms

127. Word Ladder

Solved

Hard | Topics | Companies

A **transformation sequence** from word `beginWord` to word `endWord` using a dictionary `wordList` is a sequence of words `beginWord -> $s_1$ -> $s_2$ -> ... -> $s_k$` such that:

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

Constraints:

- `1 <= beginWord.length <= 10`
- `endWord.length == beginWord.length`

Code

Python3 • Auto

```python
class Solution:
    def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) -> int:

```

words	dot	hot	dog	lot	log	cog
src	hit					
dest	cog					
set	dot	hot	dog	lot	log	cog
queue						
level	0					

Create a set using the available words for faster lookup, a queue for the BFS aprroach, and a length variable to store the length of the sequence.

words	dot	hot	dog	lot	log	cog
src	hit					
dest	cog					
set	dot	hot	dog	lot	log	cog
queue	hit					
level	1					

Push the src word ("hit") into the queue and increment the length.

words	dot	hot	dog	lot	log	cog
src	hit					
dest	cog					
set	dot	hot	dog	lot	log	cog
queue	hit					
level	1					

Pop a word ("hit") from the queue, and check all words in the set that differ by one character from the popped word.

words	dot	hot	dog	lot	log	cog
src	hit					
dest	cog					
set	dot	hot	dog	lot	log	cog
queue	hot					
level	2					

Push all the words ("hot") into the queue that differ by one character from the popped word ("hit"), remove them from the set, and increment the length.

words	dot	hot	dog	lot	log	cog
src	hit					
dest	cog					
set	dot		dog	lot	log	cog
queue	hot					
level	2					

Pop a word ("hot") from the queue, and check all the words that differ by one character from the popped word.

words	dot	hot	dog	lot	log	cog
src	hit					
dest	cog					
set	dot		dog	lot	log	cog
queue	dot	lot				
level	3					

Push all the words ("dot" and "lot") into the queue that differ by one character from the popped word "hot", remove them from the set, and increment the length.

words	dot	hot	dog	lot	log	cog
src	hit					
dest	cog					
set			dog		log	cog
queue	dot	lot				
level	3					

Pop a word ("dot") from the queue, and check all the words that differ by one character from the popped word.

words	dot	hot	dog	lot	log	cog
src	hit					
dest	cog					
set			dog		log	cog
queue	lot	dog				
level	3					

Push all the words ("dog") into the queue that differ by one character from the popped word "dot", and remove them from the set.

words	dot	hot	dog	lot	log	cog
src	hit					
dest	cog					
set					log	cog
queue	lot	dog				
level	3					

Pop a word ("lot") from the queue, and check all the words that differ by one character from the popped word.

words	dot	hot	dog	lot	log	cog
src	hit					
dest	cog					
set					log	cog
queue	dog	log				
level	4					

Push all the words ("log") into the queue that differ by one character from the popped word "lot", remove them from the set, and increment the length.

words	dot	hot	dog	lot	log	cog
src	hit					
dest	cog					
set						cog
queue	dog	log				
level	4					

Pop a word ("dog") from the queue, and check all the words that differ by one character from the popped word.

words	dot	hot	dog	lot	log	cog
src	hit					
dest	cog					
set						cog
queue	log					
level	5					

Since the word "cog" is the **dest** word, we return **length + 1** as the length of the shortest transformation sequence.

Problem List

Run Submit 0 Premium

Description | Editorial | Solutions | Submissions

127. Word Ladder

Solved

Hard | Topics | Companies

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.
```

Constraints:

- `1 <= beginWord.length <= 10`

Code

Python3 | Auto

```python
class Solution:
    def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) -> int:
        # Create a set of all words in the word list for quick lookup.
        wordSet = set(wordList)
        # If endWord is not in the word list, return 0.
        if endWord not in wordSet:
            return 0
        # Use a queue to perform BFS (Breadth-First Search).
        wordQueue = deque([beginWord])
        # Distance from the beginWord (initially 1).
        distance = 1
        while wordQueue:
            size = len(wordQueue)
            distance += 1  # Increase distance at each level of BFS

            for _ in range(size):
                currWord = wordQueue.popleft()

                # Try changing each character of the current word
                for i in range(len(currWord)):
                    for j in range(26):  # Try all lowercase letters
                        temp = currWord[:i] + chr(ord('a') + j) + currWord[i+1:]

                        # If the new word matches the endWord, return the distance
                        if temp == endWord:
                            return distance

                        # If the new word is in the set, add it to the queue and remove it from the set
                        if temp in wordSet:
                            wordQueue.append(temp)
                            wordSet.remove(temp)

        return 0  # Return 0 if no transformation sequence is found
```

Saved

Ln 34, Col 9