

33. Search in Rotated Sorted Array

Solved

Medium Topics Companies

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` ($1 \leq k < \text{nums.length}$) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of target if it is in nums, or -1 if it is not in nums*.

You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1:

Input: `nums = [4,5,6,7,0,1,2]`, `target = 0`
Output: `4`

Example 2:

Input: `nums = [4,5,6,7,0,1,2]`, `target = 3`
Output: `-1`

Example 3:

Input: `nums = [1]`, `target = 0`
Output: `-1`

Constraints:

- $1 \leq \text{nums.length} \leq 5000$
- $-10^4 \leq \text{nums}[i] \leq 10^4$

 Code

Python3 • Auto


```
1 class Solution:
2     def search(self, nums: List[int], target: int) -> int:
3
```

 Saved

Ln 1, Col 1

```
[4,5,6,7,0,1,2]
```

```
L     M     R
```

L: Left pointer

M: Middle pointer (= (L + R) // 2)

R: Right pointer

```
[4,5,6,7,0,1,2]
```

```
L M
```

```
R
```

```
target = 0
```

```
[4,5,6,7,0,1,2]
```

```
L
```

```
R
```

```
M
```

```
target = 0
```

if `nums[M] >= nums[L]`
-> **True**, all numbers are
sorted in ascending order
between M and L
-> **False**, all numbers are
sorted in ascending order
between M and R

```
[4,5,6,7,0,1,2]
```

```
M L     R
```

```
target = 0
```

```
[4,5,6,7,0,1,2]
```

```
L M R
```

```
target = 0
```

```
[5,6,0,1,2,3,4]
```

```
L     M     R
```

```
target = 0
```

```
if nums[M] >= nums[L] → false
```

That means all numbers between L and M are not sorted and all numbers between M and R are sorted, so we can use range of M and R to find the target.

In this case, M should be the smallest and R is the biggest so,

```
if nums[M] <= target <= nums[R]
```

If true, L should be M + 1 .

If false, R should be M - 1 .

33. Search in Rotated Sorted Array

Solved

Medium Topics Companies

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index k ($1 \leq k < \text{nums.length}$) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of target if it is in nums*, or `-1` if it is not in `nums`.

You must write an algorithm with $O(\log n)$ runtime complexity.

Example 1:

Input: `nums = [4,5,6,7,0,1,2]`, `target = 0`
Output: `4`

Example 2:

Input: `nums = [4,5,6,7,0,1,2]`, `target = 3`
Output: `-1`

Example 3:

Input: `nums = [1]`, `target = 0`
Output: `-1`

Constraints:

- $1 \leq \text{nums.length} \leq 5000$

Code

Python3 Auto

Menu Bookmarks Copy Paste

```
1 class Solution:
2     def search(self, nums: List[int], target: int) -> int:
3
4         left = 0
5         right = len(nums) - 1
6
7         while left <= right:
8             mid = (left + right) // 2
9
10            # Case 1: find target
11            if nums[mid] == target:
12                return mid
13            # Case 2: subarray on mid's left is sorted
14            elif nums[mid] >= nums[left]:
15                # If the key lies within this sorted half, move the right pointer to mid - 1
16                if nums[left] <= target <= nums[mid]:
17                    right = mid - 1
18                #Otherwise, move the left pointer to mid + 1.
19            else:
20                left = mid + 1
21            # Case 3: subarray on mid's right is sorted.
22        else:
23            #If the key lies within this sorted half, move the left pointer to mid + 1.
24            if nums[mid] <= target <= nums[right]:
25                left = mid + 1
26            #Otherwise, move the right pointer to mid - 1.
27        else:
28            right = mid - 1
29
30        return -1 #Key not found
```

Saved

Ln 30, Col 26

Testcase Test Result

200. Number of Islands

Solved 🟢

Medium

🔖 Topics

🏢 Companies

Given an $m \times n$ 2D binary grid `grid` which represents a map of '1's (land) and '0's (water), return the number of islands.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
Input: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
```

Output: 1

Example 2:

```
Input: grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
```

Output: 3

Constraints:

- $m == \text{grid.length}$
- $n == \text{grid}[i].\text{length}$

```
1 class Solution:
2     def numIslands(self, grid: List[List[str]]) -> int:
3
```

BFS

find all neighbours
 → pick one
 → find its neighbours
 → ...

done!

0	1	1	0	0
1	1	1	0	1
1	1	0	0	0
0	1	0	1	1
0	0	0	1	1

DFS

start: find one neighbour
 → find its neighbour
 → find its neighbour
 → ...
 ← return
 ← return
 ← return if can't find
 any more neighbours

done!

root node that triggers a DFS

1	1	1
0	1	0
1	0	0
1	0	1

of islands = 0

root node that triggers a DFS

1	1	1
0	1	0
1	0	0
1	0	1

of islands = 1

root node that triggers a DFS

1	1	1
0	1	0
1	0	0
1	0	1

of islands = 1

root node that triggers a DFS

1	1	1
0	1	0
1	0	0
1	0	1

of islands = 2

root node that triggers a DFS

1	1	1
0	1	0
1	0	0
1	0	1

of islands = 2

root node that triggers a DFS

1	1	1
0	1	0
1	0	0
1	0	1

of islands = 3

```
1.  grid = [  
    ["1","1","0","0","0"],  
    ["1","1","0","0","0"],  
    ["0","0","1","0","0"],  
    ["0","0","0","1","1"]  
]
```

```
2.  [  
    ["0","0","0","0","0"],  
    ["0","0","0","0","0"],  
    ["0","0","1","0","0"],  
    ["0","0","0","1","1"]  
]
```

```
3.  [  
    ["0","0","0","0","0"],  
    ["0","0","0","0","0"],  
    ["0","0","0","0","0"],  
    ["0","0","0","1","1"]  
]
```

200. Number of Islands

Solved [Medium](#) [Topics](#) [Companies](#)

Given an $m \times n$ 2D binary grid `grid` which represents a map of '1's (land) and '0's (water), return the number of islands.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

```
Input: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
```

Output: 1

Example 2:

```
Input: grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
```

Output: 3

Constraints:

- $m == \text{grid.length}$

</> Code

Python3  Auto

```
1 class Solution:
2     def numIslands(self, grid):
3         if not grid:
4             return 0
5
6         num_islands = 0
7         for i in range(len(grid)):
8             for j in range(len(grid[0])):
9                 if grid[i][j] == "1":
10                     self.dfs(grid, i, j)
11                     num_islands += 1
12
13         return num_islands
14
15     def dfs(self, grid, r, c):
16         if (
17             r < 0
18             or c < 0
19             or r >= len(grid)
20             or c >= len(grid[0])
21             or grid[r][c] != "1"
22         ):
23             return
24         grid[r][c] = "0"
25
26         self.dfs(grid, r - 1, c)
27         self.dfs(grid, r + 1, c)
28         self.dfs(grid, r, c - 1)
29         self.dfs(grid, r, c + 1)
```

200. Number of Islands

Solved 🟢

Medium 🔖 Topics 🔒 Companies

Given an $m \times n$ 2D binary grid `grid` which represents a map of '1's (land) and '0's (water), return the number of islands.

An **island** is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1:

Input: `grid = [`
 `["1","1","1","1","0"],`
 `["1","1","0","1","0"],`
 `["1","1","0","0","0"],`
 `["0","0","0","0","0"]`
 `]`

Output: 1

Example 2:

Input: `grid = [`
 `["1","1","0","0","0"],`
 `["1","1","0","0","0"],`
 `["0","0","1","0","0"],`
 `["0","0","0","1","1"]`
 `]`

Output: 3

Constraints:

- `m == grid.length`

```
1 class Solution:
2     def numIslands(self, grid: List[List[str]]) -> int:
3         if not grid:
4             return 0
5
6         nr = len(grid)
7         nc = len(grid[0])
8         num_islands = 0
9
10        for r in range(nr):
11            for c in range(nc):
12                if grid[r][c] == "1":
13                    num_islands += 1
14                    grid[r][c] = 0 # mark as visited
15                    neighbors = []
16                    neighbors.append((r, c))
17                    while neighbors:
18                        row, col = neighbors.pop(0)
19                        if row - 1 >= 0 and grid[row - 1][col] == "1":
20                            neighbors.append((row - 1, col))
21                            grid[row - 1][col] = "0"
22                        if row + 1 < nr and grid[row + 1][col] == "1":
23                            neighbors.append((row + 1, col))
24                            grid[row + 1][col] = "0"
25                        if col - 1 >= 0 and grid[row][col - 1] == "1":
26                            neighbors.append((row, col - 1))
27                            grid[row][col - 1] = "0"
28                        if col + 1 < nc and grid[row][col + 1] == "1":
29                            neighbors.append((row, col + 1))
30                            grid[row][col + 1] = "0"
31        return num_islands
```


690. Employee Importance

Medium Topics Companies

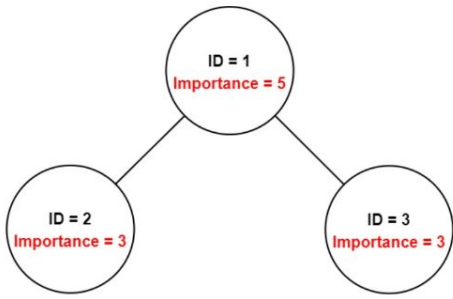
You have a data structure of employee information, including the employee's unique ID, importance value, and direct subordinates' IDs.

You are given an array of employees `employees` where:

- `employees[i].id` is the ID of the i^{th} employee.
- `employees[i].importance` is the importance value of the i^{th} employee.
- `employees[i].subordinates` is a list of the IDs of the direct subordinates of the i^{th} employee.

Given an integer `id` that represents an employee's ID, return the **total importance value of this employee and all their direct and indirect subordinates**.

Example 1:



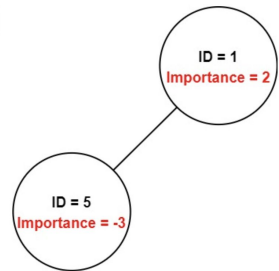
Input: employees = [[1,5,[2,3]],[2,3,[],[3,3,[]]], id = 1
Output: 11
Explanation: Employee 1 has an importance value of 5 and has two direct subordinates: employee 2 and employee 3. They both have an importance value of 3. Thus, the total importance value of employee 1 is 5 + 3 + 3 = 11.

</> Code

Python3 Auto

```
1 """
2 # Definition for Employee.
3 class Employee:
4     def __init__(self, id: int, importance: int, subordinates: List[int]):
5         self.id = id
6         self.importance = importance
7         self.subordinates = subordinates
8 """
9
10 class Solution:
11     def getImportance(self, employees: List['Employee'], id: int) -> int:
12
```

Example 2:



Input: employees = [[1,2,[5]],[5,-3,[]]], id = 5
Output: -3
Explanation: Employee 5 has an importance value of -3 and has no direct subordinates. Thus, the total importance value of employee 5 is -3.

Saved

Ln 1,



690. Employee Importance

Solved 🟢

Medium Topics Companies

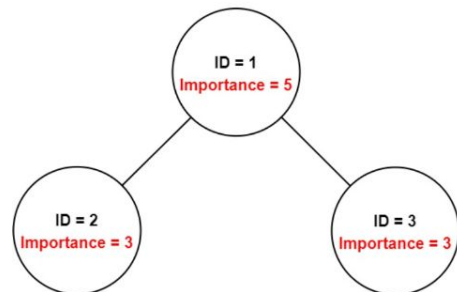
You have a data structure of employee information, including the employee's unique ID, importance value, and direct subordinates' IDs.

You are given an array of employees `employees` where:

- `employees[i].id` is the ID of the i^{th} employee.
- `employees[i].importance` is the importance value of the i^{th} employee.
- `employees[i].subordinates` is a list of the IDs of the direct subordinates of the i^{th} employee.

Given an integer `id` that represents an employee's ID, return the **total importance value of this employee and all their direct and indirect subordinates**.

Example 1:



Input: `employees = [[1,5,[2,3]],[2,3,[],[3,3,[]]]`, `id = 1`

Output: 11

2.2K 28 ☆ ↗ ?

13 Online



Run



Submit



</> Code

Python3 • Auto

≡ 📖 { } ↶ ↷ ↵

```
1 class Solution(object):
2     def getImportance(self, employees, query_id):
3         # Step 1: Create a dictionary to easily find employees by their ID
4         employee_map = {}
5         for employee in employees:
6             employee_map[employee.id] = employee
7
8         # Step 2: Define a recursive function to compute total importance
9         def get_total_importance(employee_id):
10             # Get the Employee object for the current ID
11             employee = employee_map[employee_id]
12
13             # Start with this employee's own importance
14             total = employee.importance
15
16             # Now add the importance of each of their subordinates
17             for sub_id in employee.subordinates:
18                 total += get_total_importance(sub_id) # recursive call
19
20             # Return the total importance
21             return total
22
23         # Step 3: Start the recursion from the queried employee
24         return get_total_importance(query_id)
25
```

Saved

Ln 25, Col 1

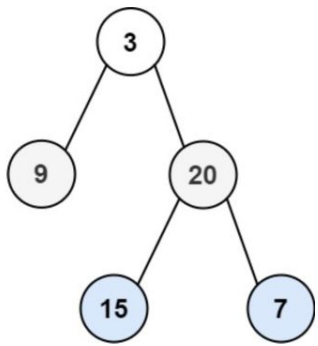
☑ Testcase ➤ Test Result

102. Binary Tree Level Order Traversal

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

```
1 # Definition for a binary tree node.
2 # class TreeNode:
3 #     def __init__(self, val=0, left=None, right=None):
4 #         self.val = val
5 #         self.left = left
6 #         self.right = right
7 class Solution:
8     def levelOrder(self, root: Optional[TreeNode]) -> List[List[int]]:
9
```

📄 Saved

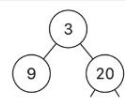
Ln 1, Col 1

✅ Testcase ➤ Test Result

Case 1 Case 2 Case 3 +

root =

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



Root

Level 0

10

Level 1

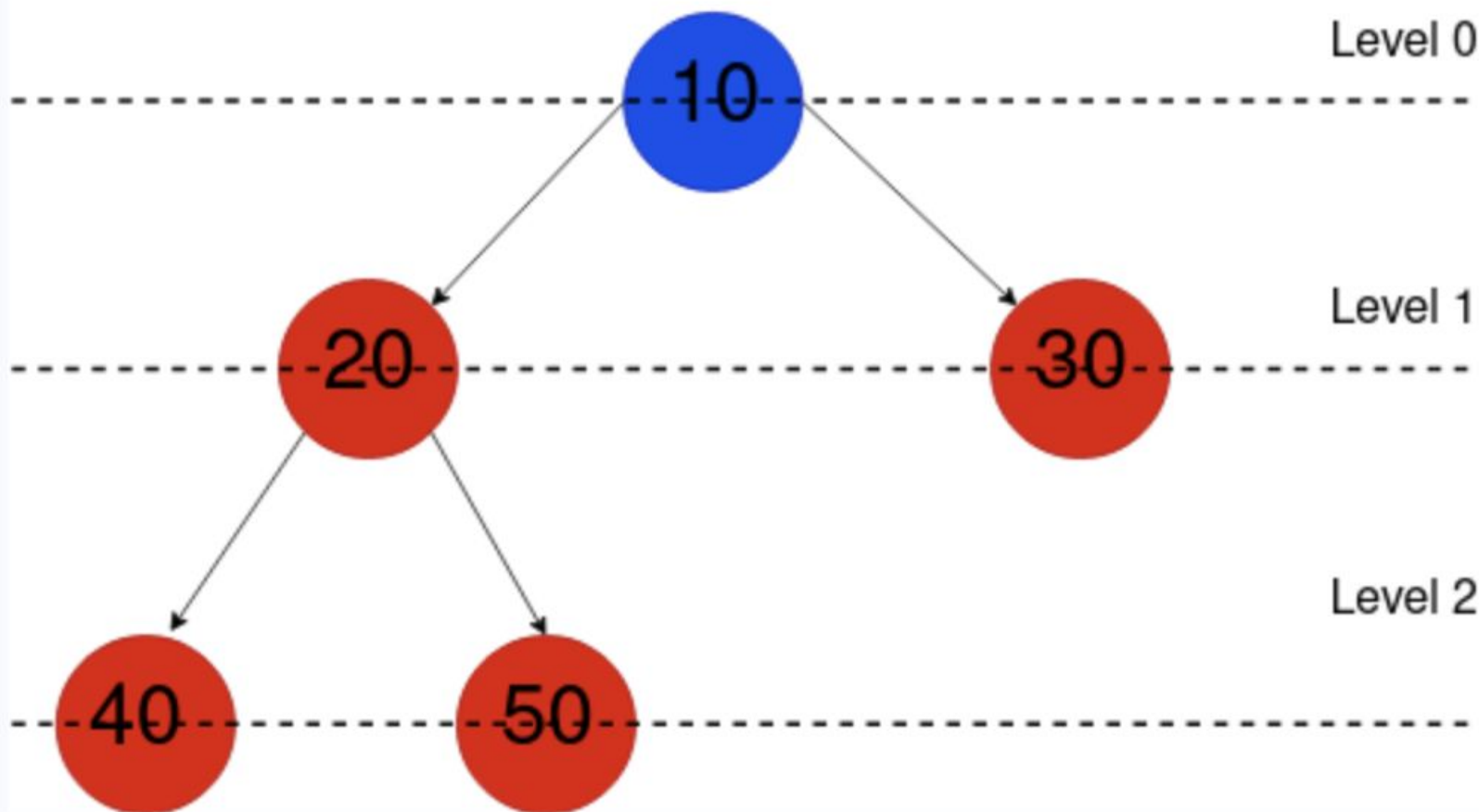
20

30

Level 2

40

50



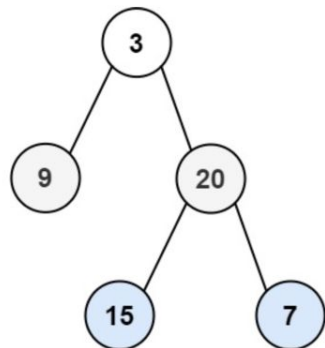
102. Binary Tree Level Order Traversal

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 = []`

</> Code

Python3 • Auto

☰ 📖 ⌵ ↶ ↷ ↺ ↻

```

1 class Solution:
2     def levelOrder(self, root: TreeNode) -> List[List[int]]:
3         levels = []
4         if not root:
5             return levels
6         level = 0
7         queue = deque(
8             [
9                 root,
10            ]
11        )
12
13        while queue:
14            # start the current level
15            levels.append([])
16            # number of elements in the current level
17            level_length = len(queue)
18            for i in range(level_length):
19                node = queue.popleft()
20                # fulfill the current level
21                levels[level].append(node.val)
22                # add child nodes of the current level
23                # in the queue for the next level
24                if node.left:
25                    queue.append(node.left)
26                if node.right:
27                    queue.append(node.right)
28
29            # go to next level
30            level += 1
31
32        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 -> s1 -> s2 -> ... -> sk` such that:

- Every adjacent pair of words differs by a single letter.
- Every s_i for $1 \leq i \leq k$ 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:

- $1 \leq beginWord.length \leq 10$
- `endWord.length == beginWord.length`

Code

Python3 • Auto

```
1 class Solution:
2     def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) -> int:
3
```

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 approach, 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.

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` $\rightarrow s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_k$ such that:

- Every adjacent pair of words differs by a single letter.
- Every s_i for $1 \leq i \leq k$ 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" \rightarrow "hot" \rightarrow "dot" \rightarrow "dog" \rightarrow "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:

- $1 \leq beginWord.length \leq 10$

Code

Python3 Auto

```
1 class Solution:
2     def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) -> int:
3         # Create a set of all words in the word list for quick lookup.
4         wordSet = set(wordList)
5         # If endWord is not in the word list, return 0.
6         if endWord not in wordSet:
7             return 0
8         # Use a queue to perform BFS (Breadth-First Search).
9         wordQueue = deque([beginWord])
10        # Distance from the beginWord (initially 1).
11        distance = 1
12        while wordQueue:
13            size = len(wordQueue)
14            distance += 1 # Increase distance at each level of BFS
15
16            for _ in range(size):
17                currWord = wordQueue.popleft()
18
19                # Try changing each character of the current word
20                for i in range(len(currWord)):
21                    for j in range(26): # Try all lowercase letters
22                        temp = currWord[:i] + chr(ord('a') + j) + currWord[i+1:]
23
24                        # If the new word matches the endWord, return the distance
25                        if temp == endWord:
26                            return distance
27
28                        # If the new word is in the set, add it to the queue and remove it from the
29                        set
30                        if temp in wordSet:
31                            wordQueue.append(temp)
32                            wordSet.remove(temp)
33
34        return 0 # Return 0 if no transformation sequence is found
```

Saved

Ln 34, Col 9