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 <= k < 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 <= nums.length <= 5000`
- `-10^4 <= nums[i] <= 10^4`

Code

Python3 • Auto

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

```

Saved Ln 1, Col 1

Testcase | Test Result

```
[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`.

Description | Editorial | Solutions | Submissions

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 <= k < 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 <= nums.length <= 5000`

Code

Python3 | Auto

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

        left = 0
        right = len(nums) - 1

        while left <= right:
            mid = (left + right) // 2

            # Case 1: find target
            if nums[mid] == target:
                return mid
            # Case 2: subarray on mid's left is sorted
            elif nums[mid] >= nums[left]:
                # If the key lies within this sorted half, move the right pointer to mid - 1
                if nums[left] <= target <= nums[mid]:
                    right = mid - 1
                #Otherwise, move the left pointer to mid + 1.
                else:
                    left = mid + 1
            # Case 3: subarray on mid's right is sorted.
            else:
                #If the key lies within this sorted half, move the left pointer to mid + 1.
                if nums[mid] <= target <= nums[right]:
                    left = mid + 1
                #Otherwise, move the right pointer to mid - 1.
                else:
                    right = mid - 1

        return -1 #Key not found
```

Saved | Ln 30, Col 26

Testcase | Test Result

Description | Editorial | Solutions | Submissions

200. Number of Islands

Solved

Medium | Topics | Companies

Given an `m x 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`
- `n == grid[i].length`

Code

Python3 • Auto

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

```

Saved

Ln 1, Col 1

BFS

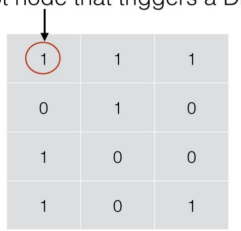
```
find all neighbours
→ pick one
→ find its neighbours
→ ...

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

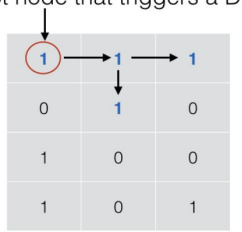
```
start: find one neighbour
→ find its neighbour
   → find its neighbour
      → ...
      ← return
   ← return
← return if can't find
  any more neighbours

done!
```

root node that triggers a DFS



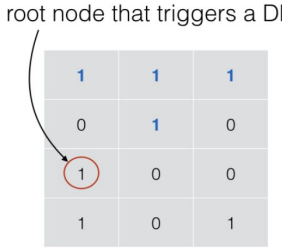
# of islands = 0

root node that triggers a DFS



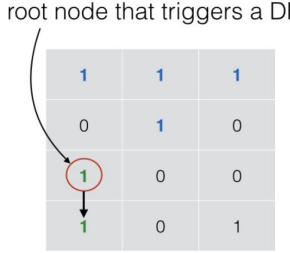
# of islands = 1

root node that triggers a DFS



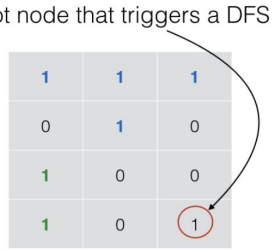
# of islands = 1

root node that triggers a DFS



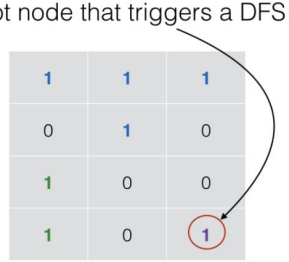
# of islands = 2

root node that triggers a DFS



# of islands = 2

root node that triggers a DFS



# 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"]
]
```

Description | Editorial | Solutions | Submissions

200. Number of Islands

Solved

Medium | Topics | Companies

Given an `m x 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`

Code

Python3 | Auto

```python
class Solution:
    def numIslands(self, grid):
        if not grid:
            return 0

        num_islands = 0
        for i in range(len(grid)):
            for j in range(len(grid[0])):
                if grid[i][j] == "1":
                    self.dfs(grid, i, j)
                    num_islands += 1

        return num_islands

    def dfs(self, grid, r, c):
        if (
            r < 0
            or c < 0
            or r >= len(grid)
            or c >= len(grid[0])
            or grid[r][c] != "1"
        ):
            return
        grid[r][c] = "0"

        self.dfs(grid, r - 1, c)
        self.dfs(grid, r + 1, c)
        self.dfs(grid, r, c - 1)
        self.dfs(grid, r, c + 1)
```

Description | Editorial | Solutions | Submissions

200. Number of Islands

Solved

Medium | Topics | Companies

Given an `m x 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`

Code

Python3 | Auto

```python
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        if not grid:
            return 0

        nr = len(grid)
        nc = len(grid[0])
        num_islands = 0

        for r in range(nr):
            for c in range(nc):
                if grid[r][c] == "1":
                    num_islands += 1
                    grid[r][c] = 0  # mark as visited
                    neighbors = []
                    neighbors.append((r, c))
                    while neighbors:
                        row, col = neighbors.pop(0)
                        if row - 1 >= 0 and grid[row - 1][col] == "1":
                            neighbors.append((row - 1, col))
                            grid[row - 1][col] = "0"
                        if row + 1 < nr and grid[row + 1][col] == "1":
                            neighbors.append((row + 1, col))
                            grid[row + 1][col] = "0"
                        if col - 1 >= 0 and grid[row][col - 1] == "1":
                            neighbors.append((row, col - 1))
                            grid[row][col - 1] = "0"
                        if col + 1 < nc and grid[row][col + 1] == "1":
                            neighbors.append((row, col + 1))
                            grid[row][col + 1] = "0"
        return num_islands
```

Description | Editorial | Solutions | Submissions

690. Employee Importance

Medium | Topics | Companies

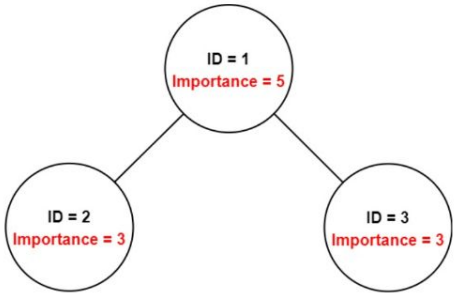
You have a data structure of employee information, including the employee's unique ID, importance value, and direct subordinates' IDs.

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:



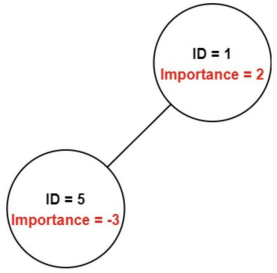

```
Input: employees = [[1,5,[2,3]],[2,3,[]],[3,3,[]]], id = 1
Output: 11
Explanation: Employee 1 has an importance value of 5 and has two direct subordinates: employee 2 and employee 3.
They both have an importance value of 3.
Thus, the total importance value of employee 1 is 5 + 3 + 3 = 11.
```

Code

Python3 • Auto

```python
"""
# Definition for Employee.
class Employee:
    def __init__(self, id: int, importance: int, subordinates: List[int
        self.id = id
        self.importance = importance
        self.subordinates = subordinates
"""

class Solution:
    def getImportance(self, employees: List['Employee'], id: int) -> i

```

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,

Description | Accepted | Editorial | Solutions | Submissions

690. Employee Importance

Solved

Medium | Topics | Companies

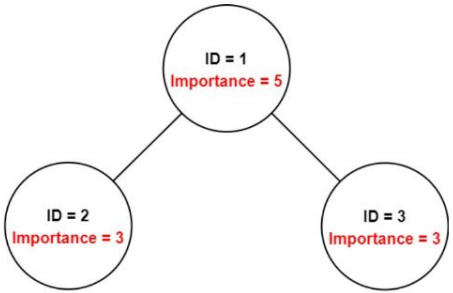
You have a data structure of employee information, including the employee's unique ID, importance value, and direct subordinates' IDs.

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
Explanation: Employee 1 has an importance value of 5 and has two direct subordinates: employee 2
```

2.1K | 22

Code

Python3 | Auto

```python
"""
# Definition for Employee.
class Employee:
    def __init__(self, id: int, importance: int, subordinates: List[int]):
        self.id = id
        self.importance = importance
        self.subordinates = subordinates
"""

class Solution(object):
    def getImportance(self, employees, query_id):
        emap = {e.id: e for e in employees}
        def dfs(eid):
            employee = emap[eid]
            return (employee.importance +
                    sum(dfs(eid) for eid in employee.subordinates))
        return dfs(query_id)
```

Saved | Ln 17, Col 29

Testcase | Test Result

Accepted Runtime: 55 ms

Case 1 | Case 2

Description | Editorial | Solutions | Submissions

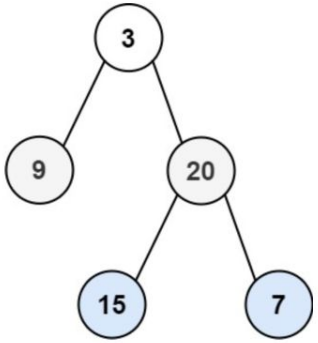
102. Binary Tree Level Order Traversal

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]]:

```

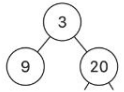
Saved Ln 1, Col 1

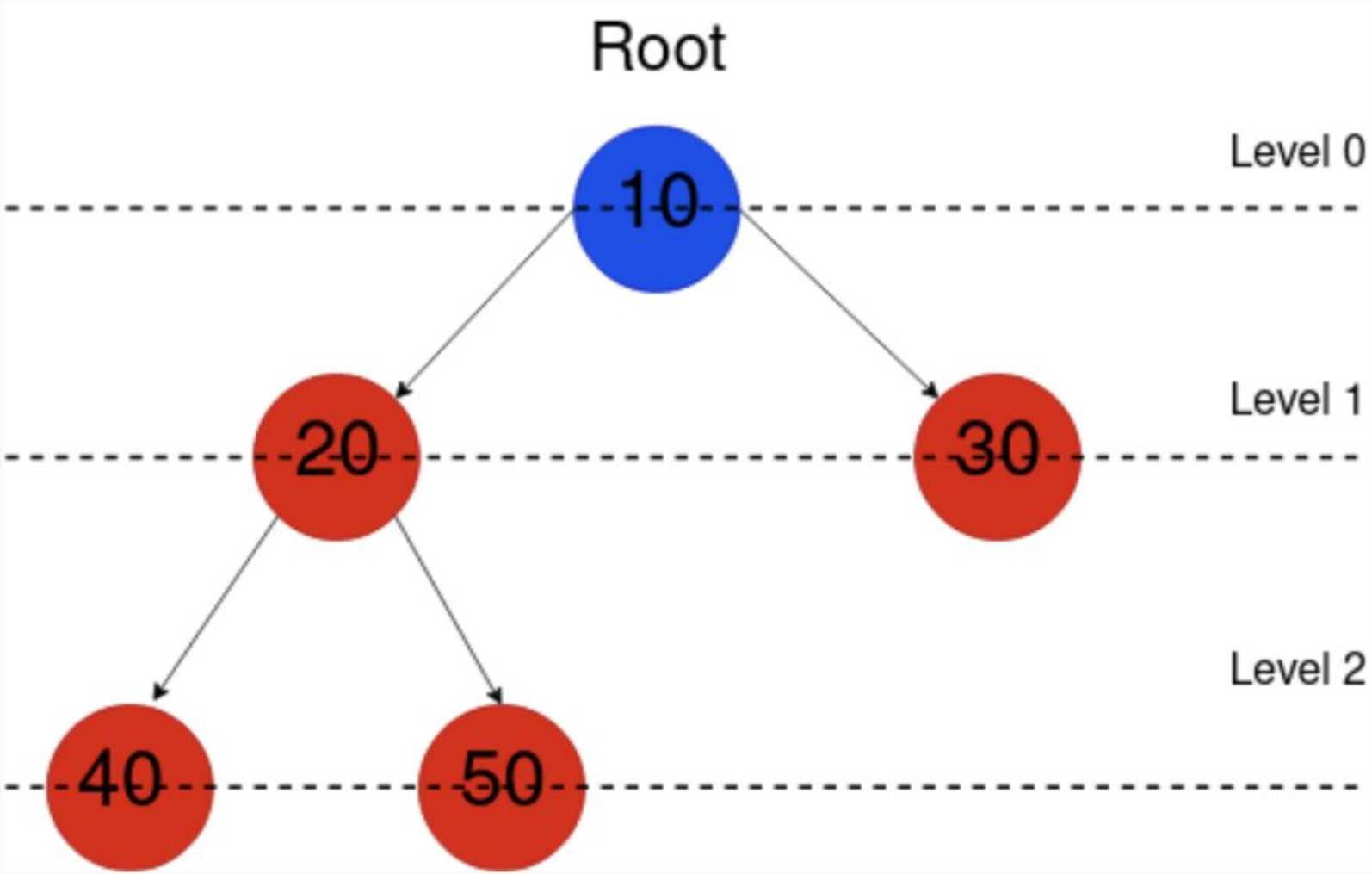
Testcase | Test Result

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

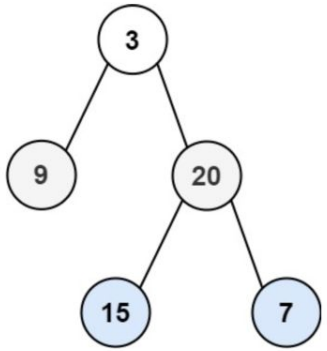
102. Binary Tree Level Order Traversal

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 -> 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:

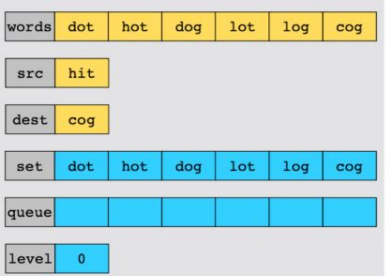
- `1 <= beginWord.length <= 10`
- `endWord.length == beginWord.length`

Code

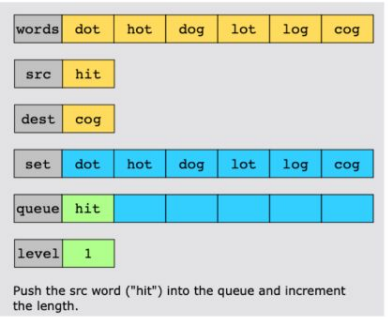
Python3 • Auto

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

```

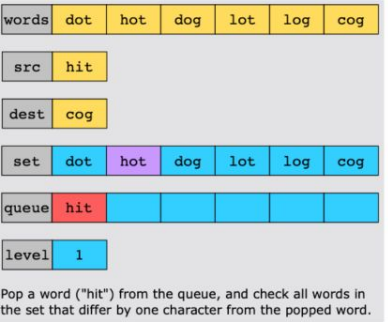
Saved Ln 3, Col 9



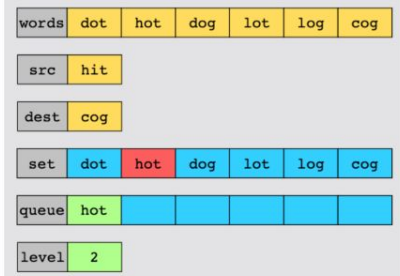
Create a set using the available words for faster lookup, a queue for the BFS aprroach, and a length variable to store the length of the sequence.



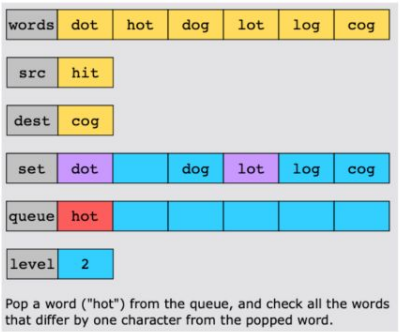
Push the src word ("hit") into the queue and increment the length.



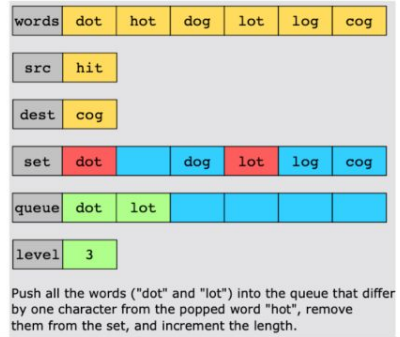
Pop a word ("hit") from the queue, and check all words in the set that differ by one character from the popped word.



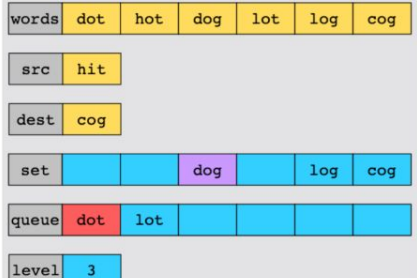
Push all the words ("hot") into the queue that differ by one character from the popped word ("hit"), remove them from the set, and increment the length.



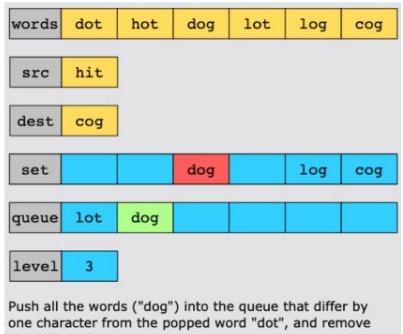
Pop a word ("hot") from the queue, and check all the words that differ by one character from the popped word.



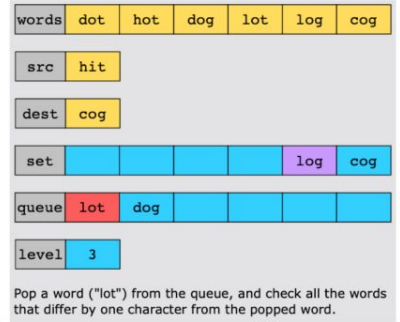
Push all the words ("dot" and "lot") into the queue that differ by one character from the popped word "hot", remove them from the set, and increment the length.



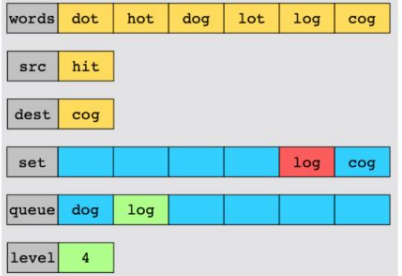
Pop a word ("dot") from the queue, and check all the words that differ by one character from the popped word.



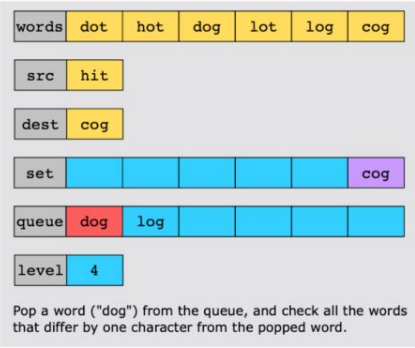
Push all the words ("dog") into the queue that differ by one character from the popped word "dot", and remove them from the set.



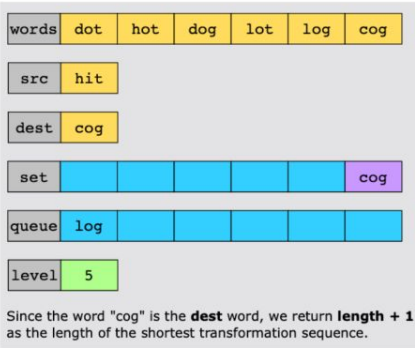
Pop a word ("lot") from the queue, and check all the words that differ by one character from the popped word.



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.



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



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

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