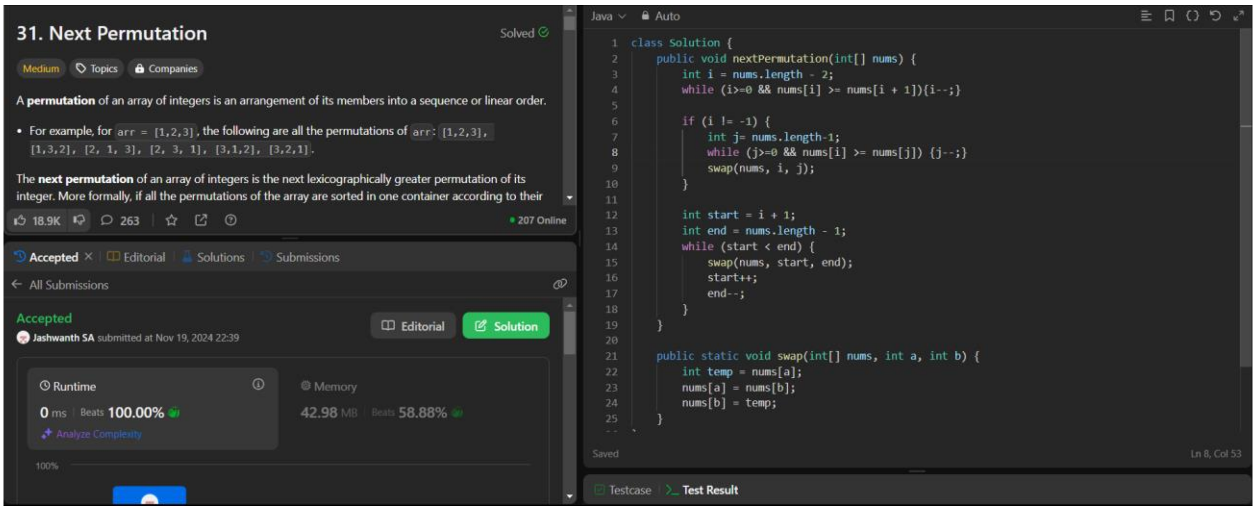
DSA practice 7 (19-11-24)

By Jashwanth SA, CSE-C

1. **Next permutation:**



31. Next Permutation

Solved

Medium · Topics · Companies

A **permutation** of an array of integers is an arrangement of its members into a sequence or linear order.

- For example, for `arr = [1,2,3]`, the following are all the permutations of `arr`: `[1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1]`.

The **next permutation** of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their

18.9K · 263 · 207 Online

Accepted · Editorial · Solutions · Submissions

All Submissions

Accepted
Jashwanth SA submitted at Nov 19, 2024 22:39

Editorial · Solution

Runtime
0 ms | Beats 100.00%
Analyze Complexity

Memory
42.98 MB | Beats 58.88%

100%

Java · Auto

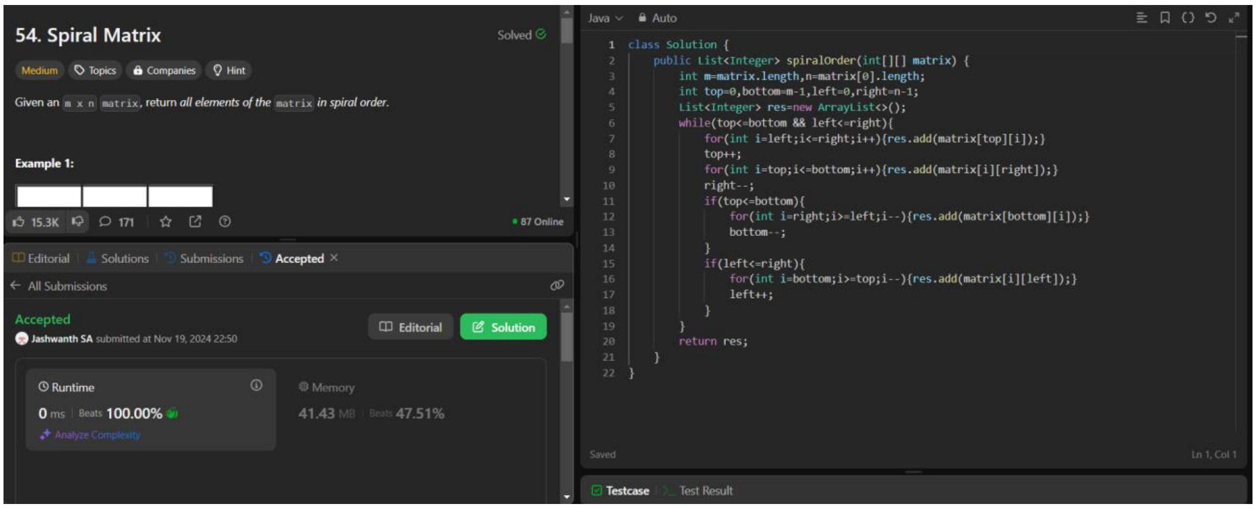
```java
class Solution {
    public void nextPermutation(int[] nums) {
        int i = nums.length - 2;
        while (i>=0 && nums[i] >= nums[i + 1]){i--;}

        if (i != -1) {
            int j= nums.length-1;
            while (j>=0 && nums[i] >= nums[j]) {j--;}
            swap(nums, i, j);
        }

        int start = i + 1;
        int end = nums.length - 1;
        while (start < end) {
            swap(nums, start, end);
            start++;
            end--;
        }
    }

    public static void swap(int[] nums, int a, int b) {
        int temp = nums[a];
        nums[a] = nums[b];
        nums[b] = temp;
    }
```

Saved · Ln 8, Col 53

Testcase · Test Result

Time complexity: O(n)

2. **Spiral matrix:**



54. Spiral Matrix

Solved

Medium · Topics · Companies · Hint

Given an `m x n` `matrix`, return *all elements of the* `matrix` *in spiral order.*

Example 1:

15.3K · 171 · 87 Online

Editorial · Solutions · Submissions · Accepted

All Submissions

Accepted
Jashwanth SA submitted at Nov 19, 2024 22:50

Editorial · Solution

Runtime
0 ms | Beats 100.00%
Analyze Complexity

Memory
41.43 MB | Beats 47.51%

Java · Auto

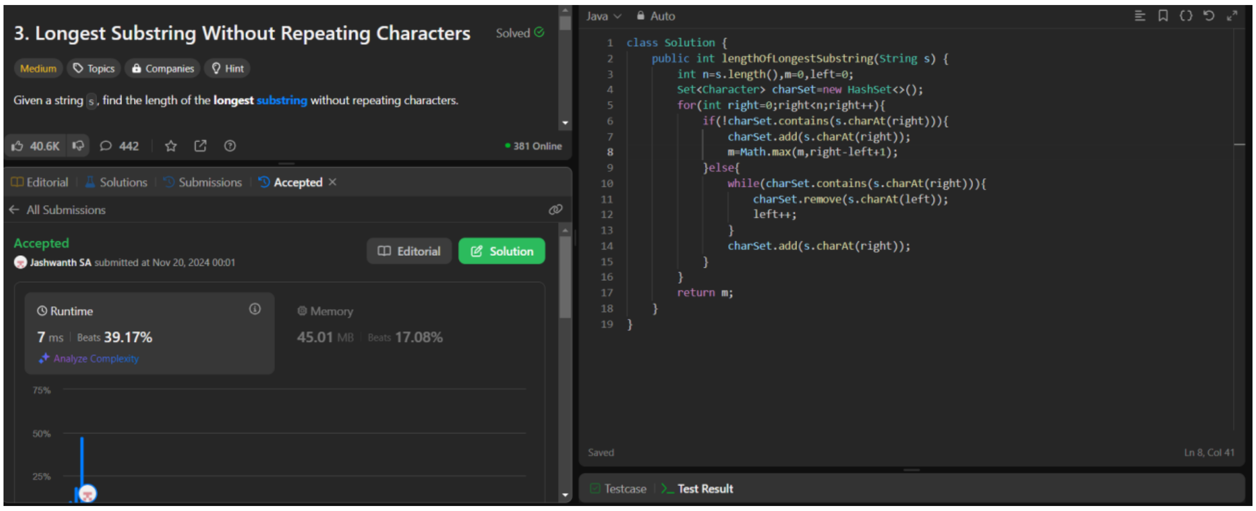
```java
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        int m=matrix.length,n=matrix[0].length;
        int top=0,bottom=m-1,left=0,right=n-1;
        List<Integer> res=new ArrayList<>();
        while(top<=bottom && left<=right){
            for(int i=left;i<=right;i++){res.add(matrix[top][i]);}
            top++;
            for(int i=top;i<=bottom;i++){res.add(matrix[i][right]);}
            right--;
            if(top<=bottom){
                for(int i=right;i>=left;i--){res.add(matrix[bottom][i]);}
                bottom--;
            }
            if(left<=right){
                for(int i=bottom;i>=top;i--){res.add(matrix[i][left]);}
                left++;
            }
        }
        return res;
    }
}
```

Saved · Ln 1, Col 1

Testcase · Test Result

Time complexity: O(n*m)

3. Longest substring without repeating characters:



3. Longest Substring Without Repeating Characters — Solved

Medium · Topics · Companies · Hint

Given a string `s`, find the length of the **longest** substring without repeating characters.

Accepted — Jashwanth SA submitted at Nov 20, 2024 00:01

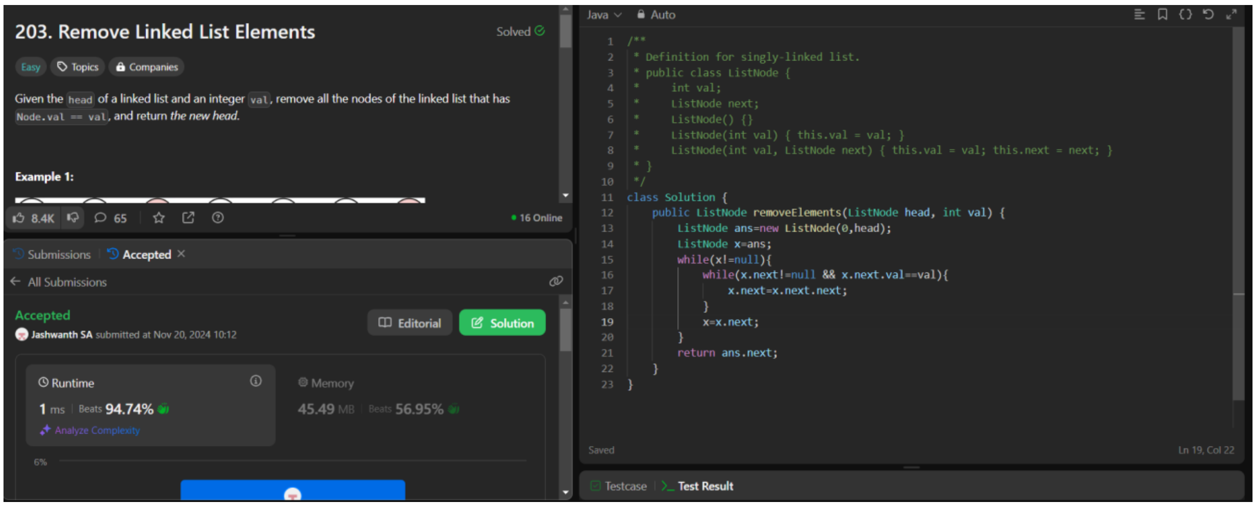
Runtime: 7 ms | Beats 39.17%

Memory: 45.01 MB | Beats 17.08%

```java
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int n=s.length(),m=0,left=0;
        Set<Character> charSet=new HashSet<>();
        for(int right=0;right<n;right++){
            if(!charSet.contains(s.charAt(right))){
                charSet.add(s.charAt(right));
                m=Math.max(m,right-left+1);
            }else{
                while(charSet.contains(s.charAt(right))){
                    charSet.remove(s.charAt(left));
                    left++;
                }
                charSet.add(s.charAt(right));
            }
        }
        return m;
    }
}
```

Time complexity: O(n)

4. Remove Linked list elements



203. Remove Linked List Elements — Solved

Easy · Topics · Companies

Given the `head` of a linked list and an integer `val`, remove all the nodes of the linked list that has `Node.val == val`, and return *the new head*.

Accepted — Jashwanth SA submitted at Nov 20, 2024 10:12

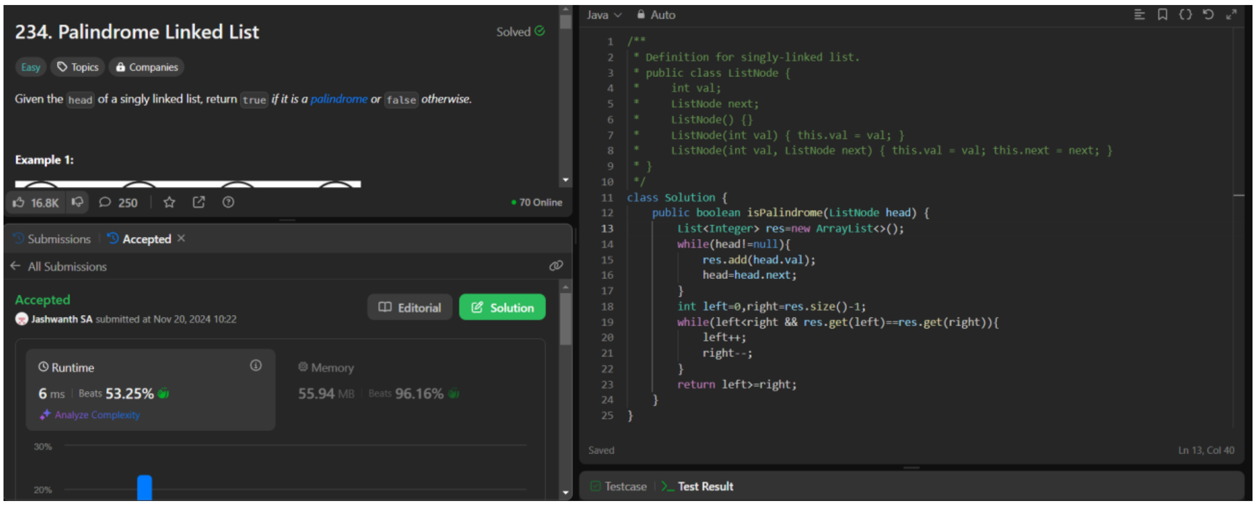
Runtime: 1 ms | Beats 94.74%

Memory: 45.49 MB | Beats 56.95%

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        ListNode ans=new ListNode(0,head);
        ListNode x=ans;
        while(x!=null){
            while(x.next!=null && x.next.val==val){
                x.next=x.next.next;
            }
            x=x.next;
        }
        return ans.next;
    }
}
```

Time complexity: O(n)

5. Palindrome linked list:



234. Palindrome Linked List — Solved

Easy · Topics · Companies

Given the `head` of a singly linked list, return `true` *if it is a palindrome or* `false` *otherwise.*

Accepted — Jashwanth SA submitted at Nov 20, 2024 10:22

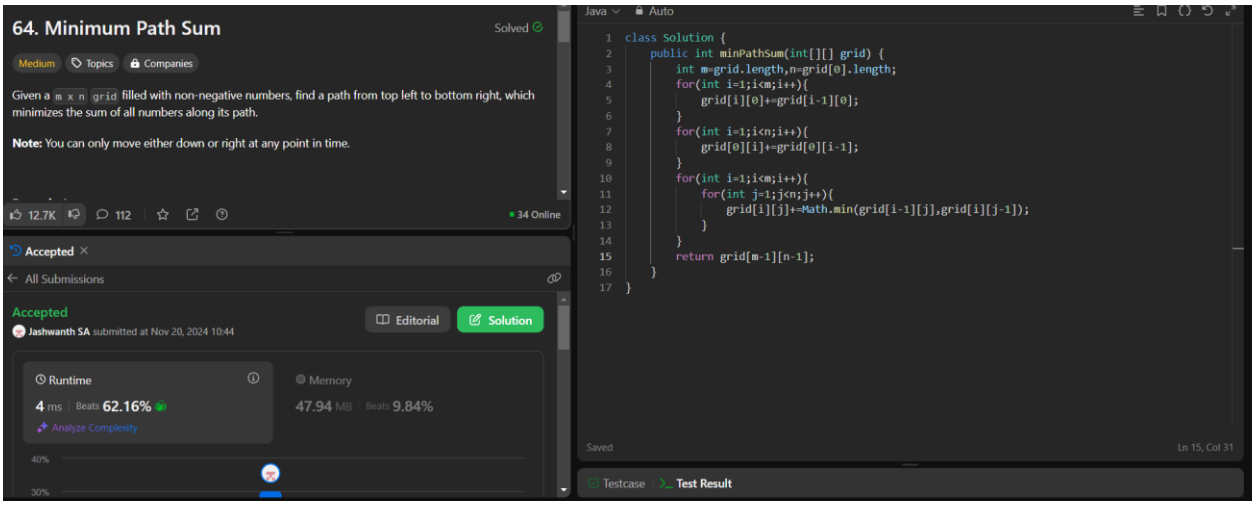
Runtime: 6 ms | Beats 53.25%

Memory: 55.94 MB | Beats 96.16%

```java
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {}
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public boolean isPalindrome(ListNode head) {
        List<Integer> res=new ArrayList<>();
        while(head!=null){
            res.add(head.val);
            head=head.next;
        }
        int left=0,right=res.size()-1;
        while(left<right && res.get(left)==res.get(right)){
            left++;
            right--;
        }
        return left>=right;
    }
}
```

Time complexity: O(n)

6. Minimum path sum:



64. Minimum Path Sum

Solved

Medium · Topics · Companies

Given a `m x n` `grid` filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

Note: You can only move either down or right at any point in time.

12.7K · 112 · 34 Online

Accepted

All Submissions

Accepted
Jashwanth SA submitted at Nov 20, 2024 10:44

Editorial · Solution

Runtime
4 ms | Beats 62.16%
Analyze Complexity

Memory
47.94 MB | Beats 9.84%

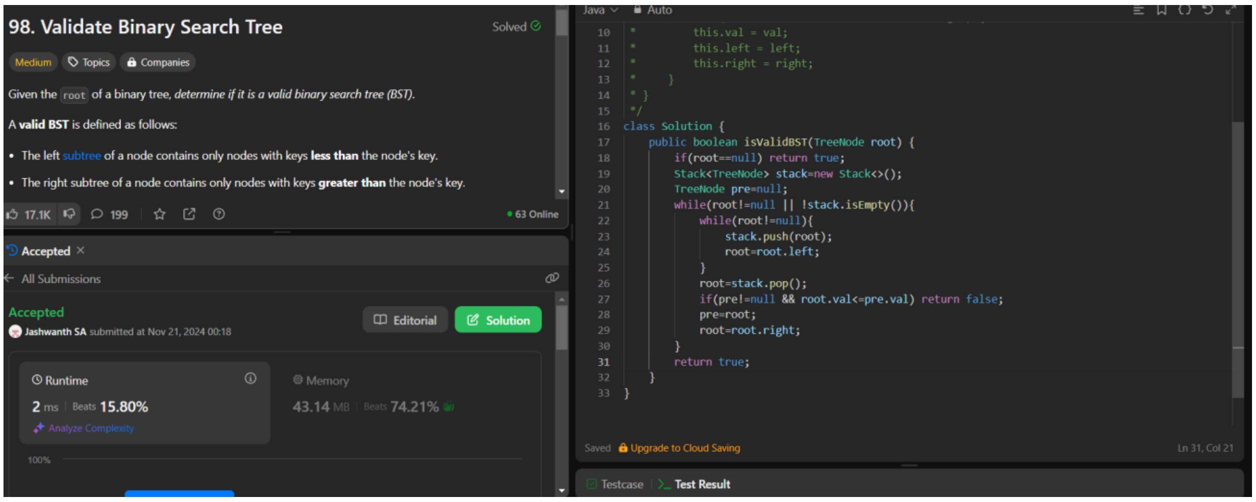
```java
class Solution {
    public int minPathSum(int[][] grid) {
        int m=grid.length,n=grid[0].length;
        for(int i=1;i<m;i++){
            grid[i][0]+=grid[i-1][0];
        }
        for(int i=1;i<n;i++){
            grid[0][i]+=grid[0][i-1];
        }
        for(int i=1;i<m;i++){
            for(int j=1;j<n;j++){
                grid[i][j]+=Math.min(grid[i-1][j],grid[i][j-1]);
            }
        }
        return grid[m-1][n-1];
    }
}
```

Saved · Ln 15, Col 31

Testcase · Test Result

Time complexity: O(n^2)

7. Validate binary search tree:



98. Validate Binary Search Tree

Solved

Medium · Topics · Companies

Given the `root` of a binary tree, *determine if it is a valid binary search tree (BST)*.

A **valid BST** is defined as follows:

- The left subtree of a node contains only nodes with keys **less than** the node's key.
- The right subtree of a node contains only nodes with keys **greater than** the node's key.

17.1K · 199 · 63 Online

Accepted

All Submissions

Accepted
Jashwanth SA submitted at Nov 21, 2024 00:18

Editorial · Solution

Runtime
2 ms | Beats 15.80%
Analyze Complexity

Memory
43.14 MB | Beats 74.21%

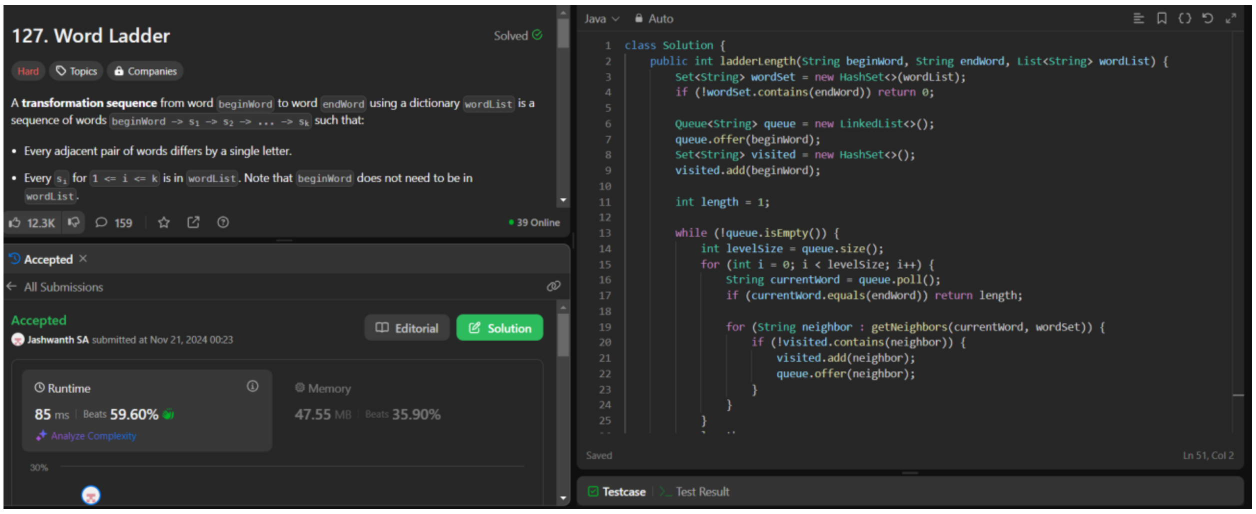
```java
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isValidBST(TreeNode root) {
        if(root==null) return true;
        Stack<TreeNode> stack=new Stack<>();
        TreeNode pre=null;
        while(root!=null || !stack.isEmpty()){
            while(root!=null){
                stack.push(root);
                root=root.left;
            }
            root=stack.pop();
            if(pre!=null && root.val<=pre.val) return false;
            pre=root;
            root=root.right;
        }
        return true;
    }
}
```

Saved · Upgrade to Cloud Saving · Ln 31, Col 21

Testcase · Test Result

Time complexity: O(n)

8. Word ladder:



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`.

12.3K · 159 · 39 Online

Accepted

All Submissions

Accepted
Jashwanth SA submitted at Nov 21, 2024 00:23

Editorial · Solution

Runtime
85 ms | Beats 59.60%
Analyze Complexity

Memory
47.55 MB | Beats 35.90%

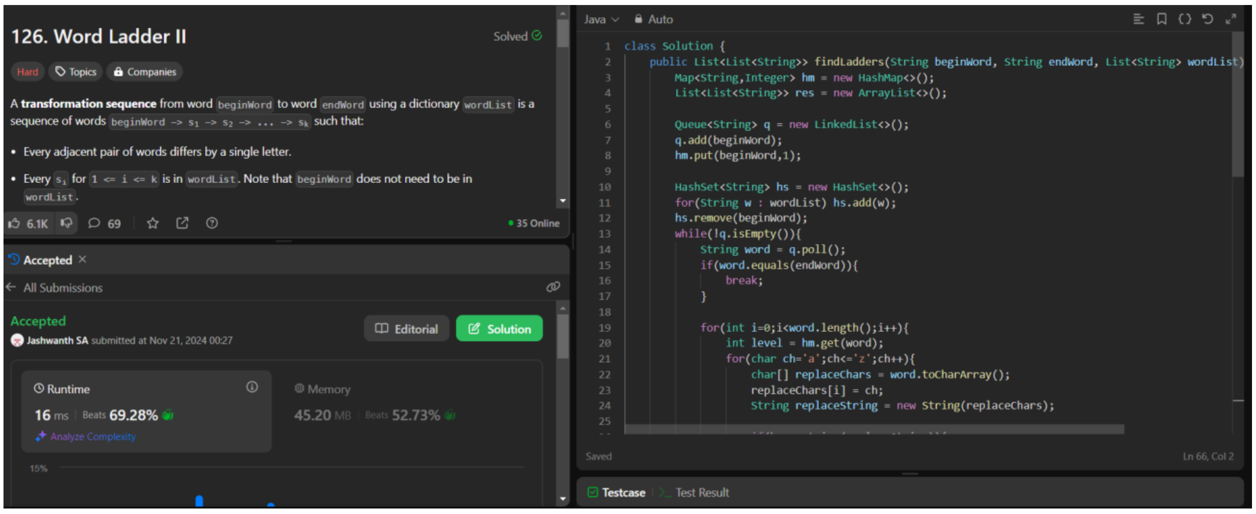
```java
class Solution {
    public int ladderLength(String beginWord, String endWord, List<String> wordList) {
        Set<String> wordSet = new HashSet<>(wordList);
        if (!wordSet.contains(endWord)) return 0;

        Queue<String> queue = new LinkedList<>();
        queue.offer(beginWord);
        Set<String> visited = new HashSet<>();
        visited.add(beginWord);

        int length = 1;

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            for (int i = 0; i < levelSize; i++) {
                String currentWord = queue.poll();
                if (currentWord.equals(endWord)) return length;

                for (String neighbor : getNeighbors(currentWord, wordSet)) {
                    if (!visited.contains(neighbor)) {
                        visited.add(neighbor);
                        queue.offer(neighbor);
                    }
                }
            }
```

Saved · Ln 51, Col 2

Testcase · Test Result

Time complexity: O(n^3)

9. Word ladder 2:



126. Word Ladder II

Solved

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`.

Accepted
Jashwanth SA submitted at Nov 21, 2024 00:27

Runtime
16 ms | Beats 69.28%

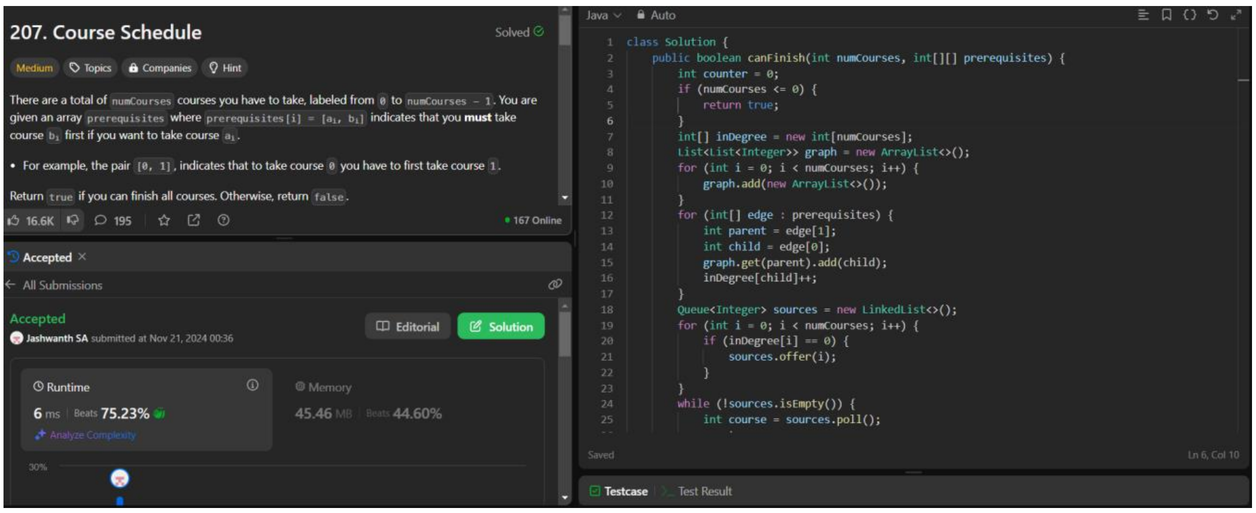
Memory
45.20 MB | Beats 52.73%

```java
class Solution {
    public List<List<String>> findLadders(String beginWord, String endWord, List<String> wordList)
        Map<String,Integer> hm = new HashMap<>();
        List<List<String>> res = new ArrayList<>();

        Queue<String> q = new LinkedList<>();
        q.add(beginWord);
        hm.put(beginWord,1);

        HashSet<String> hs = new HashSet<>();
        for(String w : wordList) hs.add(w);
        hs.remove(beginWord);
        while(!q.isEmpty()){
            String word = q.poll();
            if(word.equals(endWord)){
                break;
            }

            for(int i=0;i<word.length();i++){
                int level = hm.get(word);
                for(char ch='a';ch<='z';ch++){
                    char[] replaceChars = word.toCharArray();
                    replaceChars[i] = ch;
                    String replaceString = new String(replaceChars);

```

Saved   Ln 66, Col 2

Time complexity: O(n^3)

10. Course schedule:



207. Course Schedule

Solved

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [ai, bi]` indicates that you **must** take course `bi` first if you want to take course `ai`.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

Return `true` if you can finish all courses. Otherwise, return `false`.

Accepted
Jashwanth SA submitted at Nov 21, 2024 00:36

Runtime
6 ms | Beats 75.23%

Memory
45.46 MB | Beats 44.60%

```java
class Solution {
    public boolean canFinish(int numCourses, int[][] prerequisites) {
        int counter = 0;
        if (numCourses <= 0) {
            return true;
        }
        int[] inDegree = new int[numCourses];
        List<List<Integer>> graph = new ArrayList<>();
        for (int i = 0; i < numCourses; i++) {
            graph.add(new ArrayList<>());
        }
        for (int[] edge : prerequisites) {
            int parent = edge[1];
            int child = edge[0];
            graph.get(parent).add(child);
            inDegree[child]++;
        }
        Queue<Integer> sources = new LinkedList<>();
        for (int i = 0; i < numCourses; i++) {
            if (inDegree[i] == 0) {
                sources.offer(i);
            }
        }
        while (!sources.isEmpty()) {
            int course = sources.poll();
```

Saved   Ln 6, Col 10

Time complexity: O(n)