DSA PRACTICE QUESTIONS

19/11/2024

1.NEXT PERMUTATION: TC:

Testcase | Accepted | Test Result

All Submissions

Accepted

Sadhana_2510 submitted at Nov 19, 2024 12:21

Editorial | Solution

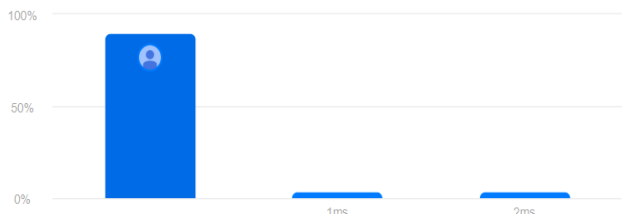
Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

16.68 MB | Beats 36.24%



Code

Python3 | Auto

```python
class Solution:
    def nextPermutation(self, nums: List[int]) -> None:
        k = len(nums) - 2
        while k >= 0 and nums[k] >= nums[k + 1]:
            k -= 1

        if k >= 0:
            l = len(nums) - 1
            while nums[l] <= nums[k]:
                l -= 1
            nums[k], nums[l] = nums[l], nums[k]

        nums[k + 1:] = reversed(nums[k +1:])

```

2. SPIRAL MATRIX: TC: O(N)

Accepted

Sadhana_2510 submitted at Nov 19, 2024 13:28

Editorial | Solution

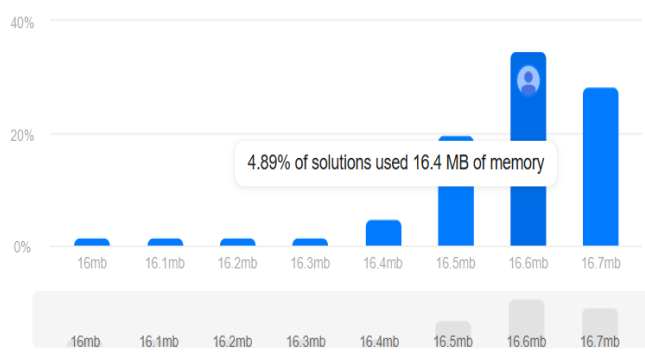
Runtime

0 ms | Beats 100.00%

Memory

16.65 MB | Beats 40.18%

Analyze Complexity



Code | Python3

```python
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        if not matrix:
```

```python
class Solution:
    def spiralOrder(self, matrix: List[List[int]]) -> List[int]:
        if not matrix:
            return []

        rows, cols = len(matrix), len(matrix[0])
        top, bottom, left, right = 0, rows-1, 0, cols-1
        result = []

        while len(result) < rows * cols:
            for i in range(left, right+1):
                result.append(matrix[top][i])
            top += 1

            for i in range(top, bottom+1):
                result.append(matrix[i][right])
            right -= 1

            if top <= bottom:
                for i in range(right, left-1, -1):
                    result.append(matrix[bottom][i])
                bottom -= 1

            if left <= right:
                for i in range(bottom, top-1, -1):
                    result.append(matrix[i][left])
                left += 1

        return result
```

Saved

3. Longest substring without repeating characters: O(N)

Accepted

submitted at Nov 19, 202

Editorial | Solution

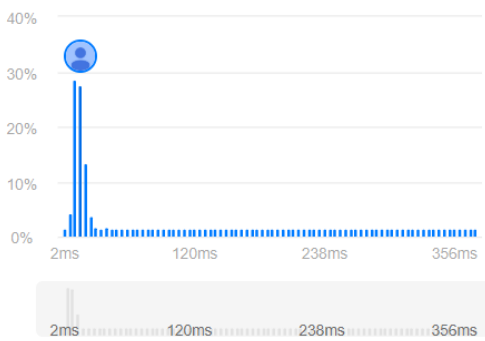
Runtime

19 ms | Beats 54.06%

Analyze Complexity

Memory

16.60 MB | Beats 96.31%




```python
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        if len(s) == 0:
            return 0

        start = 0
        end = 0
        seen = set()
        output = 0

        while end < len(s):
            if s[end] not in seen:
                seen.add(s[end])
                end += 1
                if (end-start) > output:
                    output = end-start
            else:
                while s[start] != s[end]:
                    seen.remove(s[start])
                    start += 1
                seen.remove(s[start])
                start += 1

        return output
```

Saved

4. Remove linked list elements: TC:O(N)

Testcase | Test Result | Accepted ×

All Submissions

Accepted

submitted at Nov 19, 202

Editorial | Solution

Runtime

1 ms | Beats 94.73%

Analyze Complexity

Memory

45.86 MB | Beats 13.82%

Code

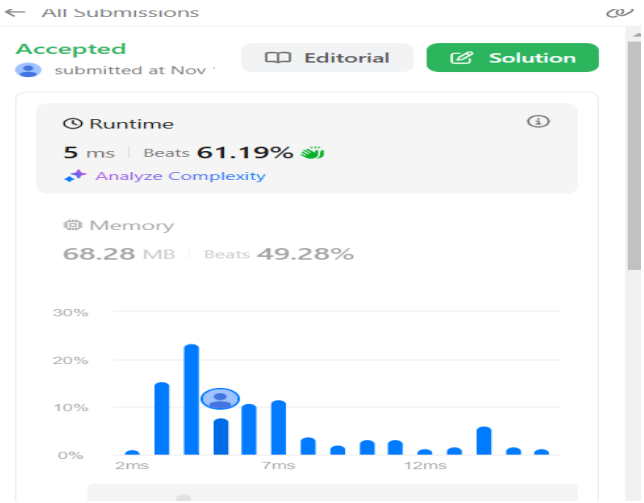
Java | Auto

```java
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        ListNode temp = new ListNode(0) , curr = temp;
        temp.next = head;
        while(curr.next != null ){
            if(curr.next.val == val) curr.next = curr.next.next;
            else curr = curr.next;
        }
        return temp.next;
    }
}
```

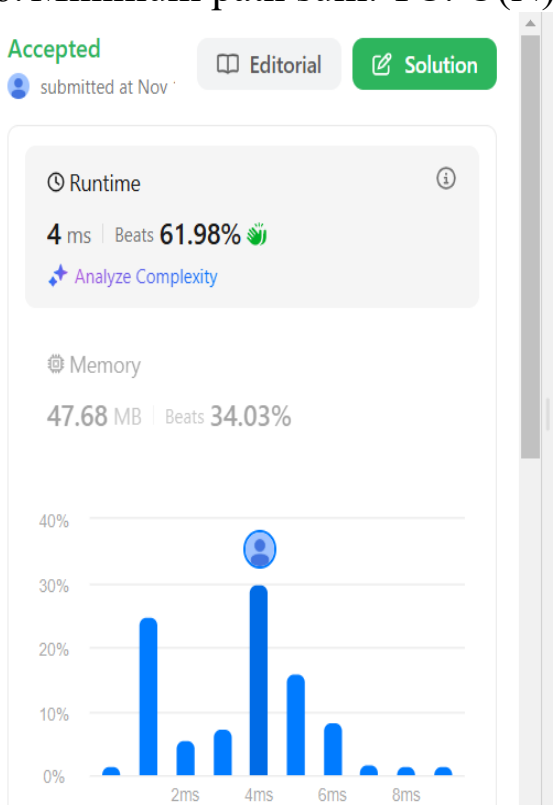
5.Palindrome linked list: TC:O(N)

```
class Solution {
    public boolean isPalindrome(ListNode head) {
        ListNode fast = head;
        ListNode slow = head;
        while(fast!=null && fast.next!=null){
            fast = fast.next.next;
            slow = slow.next;
            if(fast==slow){
                break;
            }
        }
        ListNode left =  head;
        ListNode right = reverseLinkedList(slow);
        slow.next = null;
        while(right!=null && left!=null){
            if(left.val!=right.val){
                return false;
            }
            left = left.next;
            right = right.next;

        }
        return true;
    }
    private ListNode reverseLinkedList(ListNode head){
        ListNode prev = null;
        ListNode newHead =  head;
        while(newHead !=null){
            ListNode temp  =  newHead.next;
            newHead.next = prev;
            prev =  newHead;
            newHead  =  temp;
        }
        return prev;
    }
}
```



6. Minimum path sum: TC: O(N)




```java
class Solution {
    public int minPathSum(int[][] grid) {
    if(grid.length == 0) return 0;
    int row = grid.length;
    int col = grid[0].length;

    for(int i=0;i<row; i++) {
        for(int j=0; j<col; j++) {
            int leftSum = (j>0) ? grid[i][j-1] : Integer.MAX_VALUE;
            int topSum = (i>0) ? grid[i-1][j] : Integer.MAX_VALUE;
            if(i==0 && j==0) continue;

            grid[i][j] += Math.min(leftSum, topSum);
        }
    }
    return grid[row-1][col-1];
 }
}
```

7. Validate binary search tree:

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

18.95 MB | Beats 18.01%

75%

50%

```python
class Solution:
    def isValidBST(self, root: Optional[TreeNode]) -> bool:
        def validate(root, lower, upper):
            if not root:
                return True

            # validate root first
            if root.val <= lower or root.val >= upper:
                return False

            return validate(root.left, lower, root.val) and validate(root.right, root.val, upper)

        return validate(root, float('-inf'), float('inf'))
```

8. Word ladder:

Accepted

submitted at Nov

Editorial | Solution

Runtime

224 ms | Beats 54.59%

Analyze Complexity

Memory

18.02 MB | Beats 68.72%

60%

40%

```python
class Solution:
    def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) -> int:
        wordSet, dist, q = set(wordList), {}, deque()
        q.append(beginWord)
        dist[beginWord] = 1

        while q:
            cur = q.popleft()
            if cur == endWord: return dist[cur]
            for i in range(len(cur)):
                temp = list(cur)
                for j in range(ord('a'), ord('z') + 1):
                    temp[i] = chr(j)
                    next = ''.join(temp)
                    if next in wordSet and next not in dist:
                        dist[next] = dist[cur] + 1
                        q.append(next)
        return 0
```