

Vasanthan_T_DSA_Practice-7

19/11/2024

1. Next Permutation

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 lexicographical order, then the **next permutation** of that array is the permutation that follows it in the sorted container. If such arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order).

- For example, the next permutation of arr = [1,2,3] is [1,3,2].
- Similarly, the next permutation of arr = [2,3,1] is [3,1,2].
- While the next permutation of arr = [3,2,1] is [1,2,3] because [3,2,1] does not have a lexicographical larger rearrangement.

Given an array of integers nums, *find the next permutation of* nums.

The replacement must be in place and use only constant extra memory.

Example 1:

Input: nums = [1,2,3]

Output: [1,3,2]

Code:

```
class Solution {  
    public void nextPermutation(int[] nums) {  
        int n=nums.length;  
        int ind1=-1;
```

```

    int ind2=-1;
    for(int i=n-2;i>=0;i--){
        if(nums[i]<nums[i+1]){
            ind1=i;
            break;
        }
    }
    if(ind1==-1){
        reverseFunction(nums,0);
    }
    else{
        for(int i=n-1;i>ind1;i--){
            if(nums[i]>nums[ind1]){
                ind2=i;
                break;
            }
        }
        swap(nums,ind1,ind2);
        reverseFunction(nums,ind1+1);
    }
}

public void reverseFunction(int[] nums,int l){
    int i=l;
    int j=nums.length-1;
    while(i<=j){
        swap(nums,i,j);
        i++;
        j--;
    }
}

```

```

    }

}

public void swap(int[] nums,int l,int r){

    int temp=nums[l];

    nums[l]=nums[r];

    nums[r]=temp;

}

}

```

Output:

The screenshot displays a coding platform interface for the '31. Next Permutation' problem. The problem is marked as 'Solved' with a green checkmark. The difficulty is 'Medium'. The description explains that a permutation of an array of integers is an arrangement of its members into a sequence or linear order. It provides an example: for `arr = [1,2,3]`, the permutations are `[1,2,3]`, `[1,3,2]`, `[2, 1, 3]`, `[2, 3, 1]`, `[3,1,2]`, and `[3,2,1]`. It defines the 'next permutation' as the next lexicographically greater permutation. For example, the next permutation of `arr = [1,2,3]` is `[1,3,2]`. Similarly, the next permutation of `arr = [2,3,1]` is `[3,1,2]`. While the next permutation of `arr = [3,2,1]` is `[1,2,3]` because `[3,2,1]` does not have a greater permutation.

The submission details show that the code was 'Accepted' by user 'SomasundaramM' on Nov 21, 2024, at 06:47. The runtime is 0 ms, which beats 100.00% of other submissions. The memory usage is 42.82 MB, which beats 75.22% of other submissions. The test result is also 'Accepted' with a runtime of 0 ms.

Time Complexity: $O(n)$

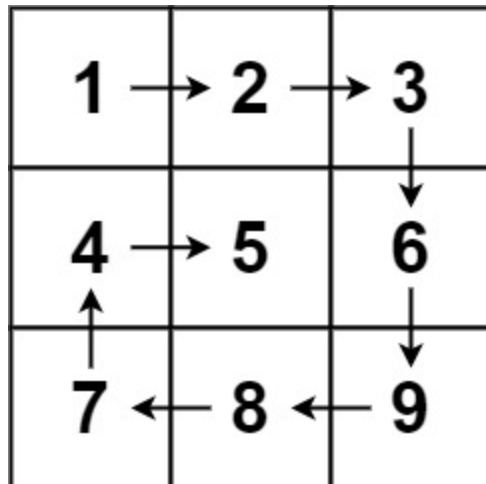
Space Complexity: $O(1)$

2.Spiral Matrix

1. Spiral Matrix

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

Example 1:



Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]

Output: [1,2,3,6,9,8,7,4,5]

Code:

```
class Solution {  
    public List<Integer> spiralOrder(int[][] matrix) {  
        List<Integer> ans = new ArrayList<>();  
        int l = 0, r = matrix[0].length - 1, t = 0, b = matrix.length - 1;  
        while (t <= b && l <= r) {  
            for (int i = l; i <= r; i++) {  
                ans.add(matrix[t][i]);  
            }  
            t++;  
            for (int i = t; i <= b; i++) {  
                ans.add(matrix[i][r]);  
            }  
            r--;  
            for (int i = r; i >= l; i--) {  
                ans.add(matrix[b][i]);  
            }  
            b--;  
            for (int i = b; i >= t; i--) {  
                ans.add(matrix[i][l]);  
            }  
            l++;  
        }  
    }  
}
```

```

        r--;
        if (t <= b) {
            for (int i = r; i >= l; i--) {
                ans.add(matrix[b][i]);
            }
            b--;
        }
        if (l <= r) {
            for (int i = b; i >= t; i--) {
                ans.add(matrix[i][l]);
            }
            l++;
        }
    }
    return ans;
}
}

```

Output:

Description

Editorial

Solutions

Submissions

54. Spiral Matrix

Solved

Medium Topics Companies Hint

Given an $m \times n$ matrix, return all elements of the matrix in spiral order.

Example 1:

1	→ 2	→ 3
4	→ 5	↓ 6
↑ 7	← 8	← 9

15.3K 171 40 Online

Code

Accepted

All Submissions

Editorial

Solution

Accepted

SomasundaramM submitted at Nov 21, 2024 06:50

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

41.16 MB | Beats 91.65%

Testcase

Test Result

Case 1

Case 2

+

matrix =

Source

Time Complexity: $O(m*n)$

Space Complexity: $O(n)$

3. Longest Substring without Repeating Character

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

Example 1:

Input: *s* = "abcabcbb"

Output: 3

Explanation: The answer is "abc", with the length of 3.

Solution:

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int n=s.length();
        if(n==0 || n==1){
            return n;
        }
        int max=-1;
        boolean[] visited=new boolean[256];
        int left=0,right=0;
        while(right<n){
            while(visited[s.charAt(right)]){
                visited[s.charAt(left)]=false;
                left++;
            }
            visited[s.charAt(right)]=true;
            max=Math.max(max,right-left+1);
            right++;
        }
        return max;
    }
}
```

Output:

The screenshot shows the LeetCode interface for the problem '3. Longest Substring Without Repeating Characters'. The problem is marked as 'Solved'. The description states: 'Given a string s, find the length of the longest substring without repeating characters.' Example 1: Input: s = "abcabcbb", Output: 3, Explanation: The answer is "abc", with the length of 3. Example 2: Input: s = "bbbbb", Output: 1, Explanation: The answer is "b", with the length of 1. The submission details show it was accepted by SomasundaramM on Nov 21, 2024, at 06:55. The runtime is 7 ms, beating 39.02% of submissions. The memory usage is 45.15 MB, beating 14.47% of submissions. The test result is 'Accepted' with a runtime of 0 ms.

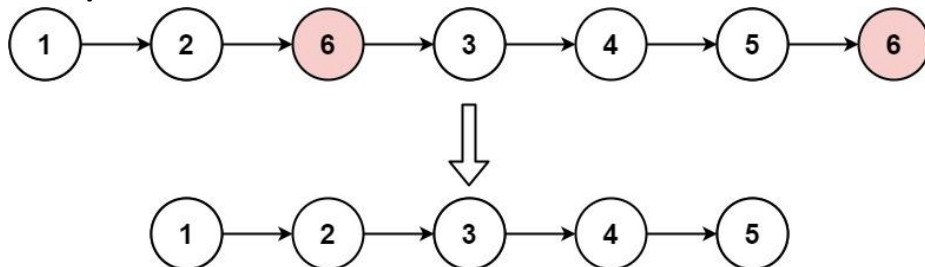
Time Complexity: $O(n)$

Space Complexity: $O(1)$

2. Remove Linked List Elements

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*.

Example 1:



Input: head = [1,2,6,3,4,5,6], val = 6

Output: [1,2,3,4,5]

Solution:

```
/**
 * 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) {
        if(head==null) return head;
        ListNode dummy=new ListNode(0);
        dummy.next=head;
        ListNode current=dummy;
        while(current.next!=null){
            if(current.next.val==val){
                current.next=current.next.next;
            }
            else current=current.next;
        }
        return dummy.next;
    }
}

```

Output:

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.

Example 1:

Input: head = [1,2,6,3,4,5,6], val = 6
Output: [1,2,3,4,5]

Runtime: 1 ms | Beats 94.73%
Memory: 45.41 MB | Beats 57.08%

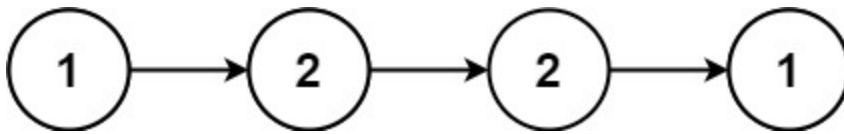
Time Complexity: $O(n)$

Space Complexity: $O(1)$

3. Palindrome Linked List

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

Example 1:



Input: head = [1,2,2,1]

Output: true

Code:

```

/**
 * 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) {
        ListNode slow=head,fast=head;
        while(fast.next!=null && fast.next.next!=null){
            slow=slow.next;
            fast=fast.next.next;
        }
        ListNode head2=reverse(slow.next);
        slow.next=null;
        while(head!=null && head2!=null){
            if(head.val!=head2.val){
                return false;
            }
            head=head.next;
            head2=head2.next;
        }
        return true;
    }

    static ListNode reverse(ListNode head){
        ListNode prev,next,curr;
        prev=null;
        curr=head;
    }
}

```

```

while(curr!=null){
    next=curr.next;
    curr.next=prev;
    prev=curr;
    curr=next;
}
return prev;
}
}

```

Output:

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.

Example 1:

Diagram: 1 → 2 → 2 → 1

Input: `head = [1,2,2,1]`
Output: `true`

Example 2:

Diagram: 1 → 2

Runtime: 5 ms | Beats 61.02%
 Memory: 68.74 MB | Beats 26.05%

Testcase: 2 Test Result

Case 1 Case 2 +

</> Source ⓘ

Time Complexity: $O(n)$

Space Complexity: $O(1)$

4. Minimum Path Sum

Given a $m \times n$ 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.

Example 1:

1	3	1
1	5	1
4	2	1

Input: grid = [[1,3,1],[1,5,1],[4,2,1]]

Output: 7

Explanation: Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.

Solution:

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

Output:

The screenshot displays a LeetCode submission for the 'Minimum Path Sum' problem. The submission is 'Accepted' and was made by 'SomasundaramM' on Nov 21, 2024, at 07:04. The runtime is 3 ms, which beats 69.75% of other submissions. The memory usage is 47.57 MB, which beats 45.32% of other submissions. A bar chart at the bottom shows the submission's performance relative to others. The code is written in Java and implements a dynamic programming solution. It initializes a grid of the same size as the input, setting the first row and first column values. Then, it iterates through the rest of the grid, calculating the minimum path sum for each cell based on its top and left neighbors. Finally, it returns the value at the bottom-right cell.

```
1 class Solution {
2     public int minPathSum(int[][] grid) {
3         int m=grid.length,n=grid[0].length;
4         for(int i=1;i<n;i++){
5             grid[0][i]+=grid[0][i-1];
6         }
7         for(int i=1;i<m;i++){
8             grid[i][0]+=grid[i-1][0];
9         }
10        for(int i=1;i<m;i++){
11            for(int j=1;j<n;j++){
12                grid[i][j]+=Math.min(grid[i-1][j],grid[i][j-1]);
13            }
14        }
15        return grid[m-1][n-1];
16    }
17 }
```

Time Complexity: $O(n)$

Space Complexity: $O(1)$