Next Permutation

Difficulty: **Medium** Accuracy: **40.66%** Submissions: **144K+** Points: **4**

Given an array of integers **arr[]** representing a permutation, implement the **next permutation** that rearranges the numbers into the lexicographically next greater permutation. If no such permutation exists, rearrange the numbers into the lowest possible order (i.e., sorted in ascending order).

Note - A permutation of an array of integers refers to a specific arrangement of its elements in a sequence or linear order.

Examples:

Input: arr = [2, 4, 1, 7, 5, 0]
Output: [2, 4, 5, 0, 1, 7]
Explanation: The next permutation of the given array is {2, 4, 5, 0, 1, 7}.

Input: arr = [3, 2, 1]
Output: [1, 2, 3]
Explanation: As arr[] is the last permutation, the next permutation is the lowest one.

Input: arr = [3, 4, 2, 5, 1]
Output: [3, 4, 5, 1, 2]
Explanation: The next permutation of the given array is {3, 4, 5, 1, 2}.

Constraints:
$1 \leq arr.size() \leq 10^5$
$1 \leq arr[i] \leq 10^5$

```cpp
void nextPermutation(vector<int> &nums)
{
    int n = nums.size(), ind = -1;
    for (int i = n - 2; i >= 0; i--)
    {
        if (nums[i] < nums[i + 1])
        {
            ind = i;
            break;
        }
    }
    for (int i = n - 1; i >= ind && ind != -1; i--)
    {
        if (nums[i] > nums[ind])
        {
            swap(nums[i], nums[ind]);
            break;
        }
    }
    reverse(nums.begin() + ind + 1, nums.end());
}
```

Time Complexity : O(n)

Space Complexity :O(1)

Longest substring with distinct characters

Difficulty: **Easy** Accuracy: **31.32%** Submissions: **128K+** Points: **2**

Given a string **s**, find the length of the longest substring with all distinct characters.

Examples:

Input: s = "geeksforgeeks"
Output: 7
Explanation: "eksforg" is the longest substring with all distinct characters.

Input: s = "aaa"
Output: 1
Explanation: "a" is the longest substring with all distinct characters.

Input: s = "abcdefabcbb"
Output: 6
Explanation: The longest substring with all distinct characters is "abcdef", which has a length of 6.

Constraints:
$1 <= s.size() <= 10^5$

All the characters are in lowercase.

```cpp
int lengthOfLongestSubstring(string s)
{
    int n = s.size();
    int maxi = 0;
    int cnt = 0;
    unordered_map<char, int> mp;
    int left = 0, right;
    for (right = 0; right < n; right++)
    {
        mp[s[right]]++;
        while (mp[s[right]] > 1)
        {
            mp[s[left]] -= 1;
            left += 1;
        }
        maxi = max(maxi, right - left + 1);
    }

    return maxi;
}
```

Time Complexity : O(n)
Space Complexity : O(n)

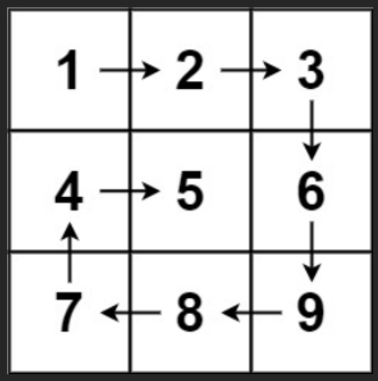
54. Spiral Matrix

Solved

Medium | Topics | Companies | Hint

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

Example 1:



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

```cpp
vector<int> spiralOrder(vector<vector<int>> &matrix)
{
    int left = 0, right = matrix[0].size();
    int top = 0, bottom = matrix.size();
    vector<int> ans;

    while (left < right && top < bottom)
    {
        for (int i = left; i < right; i++){
            ans.push_back(matrix[top][i]);
        }
        top += 1;
        for (int i = top; i < bottom; i++){
            ans.push_back(matrix[i][right - 1]);
        }
        right--;
        if (top < bottom){
            for (int i = right - 1; i >= left; i--){
                ans.push_back(matrix[bottom - 1][i]);
            }
            bottom--;
        }
        if (left < right){
            for (int i = bottom - 1; i >= top; i--){
                ans.push_back(matrix[i][left]);
            }
            left++;
        }
    }
    return ans;
}
```

Time Complexity : O(n*m)

Space Complexity: O(n*m)

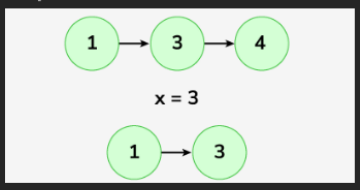
Delete in a Singly Linked List

Difficulty: **Easy** Accuracy: **39.85%** Submissions: **210K+** Points: **2**

Given a singly linked list and an integer, **x**. Delete the x^{th} node (1-based indexing) from the singly linked list.

Examples:

Input: Linked list: 1 -> 3 -> 4, x = 3
Output: 1 -> 3



Explanation: After deleting the node at the 3rd position (1-base indexing), the linked list is as 1 -> 3.

Input: Linked list: 1 -> 5 -> 2 -> 9, x = 2
Output: 1 -> 2 -> 9
Explanation: After deleting the node at 2nd position (1-based indexing), the linked list is as 1 -> 2 -> 9.

Constraints:
2 <= size of linked list <= 10^6
1 <= x <= size of linked list

s

```
ListNode *removeElements(ListNode *head, int val)
{
    ListNode *ptr = head;
    if (head == NULL)
        return NULL;

    while (head != NULL && head->val == val)
    {
        head = head->next;
    }
    while (ptr->next != NULL)
    {
        if (ptr->next->val == val)
        {
            ptr->next = ptr->next->next;
        }
        else
            ptr = ptr->next;
    }
    return head;
}
```

Time Complexity : O(n)
Space Complexity : O(1)

Palindrome Linked List

Difficulty: **Medium** Accuracy: **41.48%** Submissions: **345K+** Points: **4**

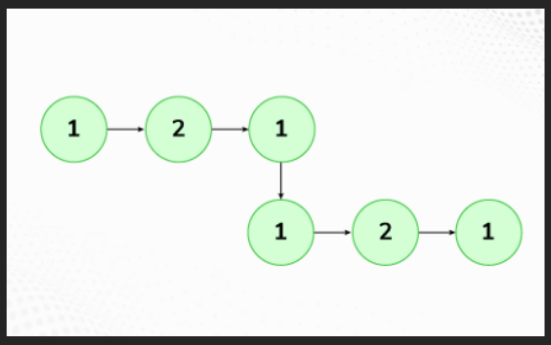
Given a singly linked list of integers. The task is to check if the given linked list is palindrome or not.

Examples:

Input: LinkedList: 1->2->1->1->2->1

Output: true

Explanation: The given linked list is 1->2->1->1->2->1 , which is a palindrome and Hence, the output is true.



```
ListNode *reverseLinkedList(ListNode *head){
    if (head == NULL || head->next == NULL)
        return head;
    ListNode *newHead = reverseLinkedList(head->next);
    ListNode *front = head->next;
    front->next = head;
    head->next = NULL;
    return newHead;
}
bool isPalindrome(ListNode *head){
    if (head == NULL || head->next == NULL)
        return true;
    ListNode *slow = head;
    ListNode *fast = head;
    while (fast->next != NULL && fast->next->next != NULL)
    {
        slow = slow->next;
        fast = fast->next->next;
    }
    ListNode *newHead = reverseLinkedList(slow->next);
    ListNode *first = head;
    ListNode *second = newHead;
    while (second != NULL)
    {
        if (first->val != second->val)
        {
            reverseLinkedList(newHead);
            return false;
        }
        first = first->next;
        second = second->next;
    }
    reverseLinkedList(newHead);
    return true;
}
```

Time Complexity :

Space Complexity:

64. Minimum Path Sum

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.

Example 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.
```

Example 2:

```
Input: grid = [[1,2,3],[4,5,6]]
Output: 12
```

Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 200`
- `0 <= grid[i][j] <= 200`

```cpp
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;

int minPathSum(vector<vector<int>> &grid)
{
    int rows = grid.size(), cols = grid[0].size();
    vector<vector<int>> ans(rows + 1, vector<int>(cols + 1, INT_MAX));
    ans[rows - 1][cols] = 0;

    for (int i = rows - 1; i >= 0; i--)
    {
        for (int j = cols - 1; j >= 0; j--)
        {
            ans[i][j] = grid[i][j] + min(ans[i + 1][j], ans[i][j + 1]);
        }
    }

    return ans[0][0];
}
```

Time Complexity: O(row*col)

Space Complexity: Without optimization: O(row×col).
With in-place optimization: O(1).

Check for BST

Difficulty: **Easy** Accuracy: **25.37%** Submissions: **546K+** Points: **2**

Given the root of a binary tree. Check whether it is a BST or not.

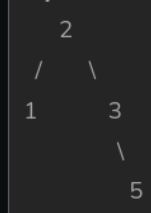
Note: We are considering that BSTs can not contain duplicate Nodes.

A **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.
- Both the left and right subtrees must also be binary search trees.

Examples:

Input:



Output: true

Explanation:

The left subtree of every node contains smaller keys and right subtree of every node contains greater. Hence, the tree is a BST.

```
bool isValid(TreeNode *root, long minVal, long maxVal)
{
    if (root == NULL)
        return true;

    if (root->val >= maxVal || root->val <= minVal)
        return false;

    return isValid(root->left, minVal, root->val) && isValid(root->right, root->val, maxVal);
}
bool isValidBST(TreeNode *root)
{
    return isValid(root, LONG_MIN, LONG_MAX);
}
```

Time Complexity : O(n)

Space Complexity : O(1)