DSA_PRACTISE_7 (19/11/2024)

1.

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.

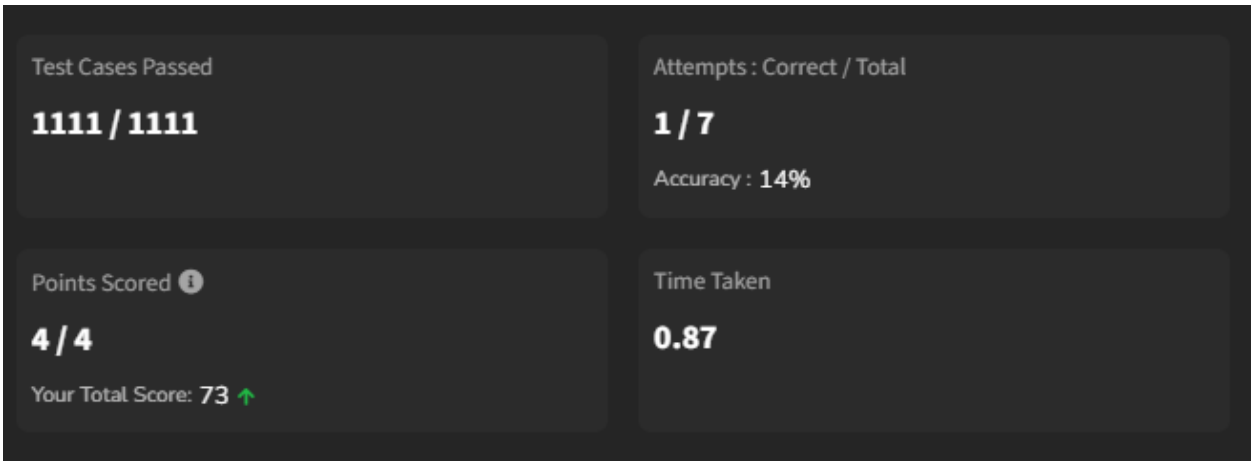
Examples:

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}.

Code:

```
class Solution {
    void nextPermutation(int[] arr) {
        int index=-1;
        for(int i=arr.length-1;i>0;i--) {
            if(arr[i]>arr[i-1]) {
                 index=i-1;
                break;
            }
        }
        if(index==-1){
          Arrays.sort(arr);
          return ;
        }
        for(int i=arr.length-1;i>=index;i--) {
            if(arr[i]>arr[index]) {
                int temp=arr[i];
                arr[i]=arr[index];
                arr[index]=temp;
                break;
            }
        }
        int i=index+1;
        int j=arr.length-1;
        while(i<j) {
            int temp=arr[i];
            arr[i]=arr[j];
            arr[j]=temp;
            i++;j--;
        }
    }
}
```

Output:



2.

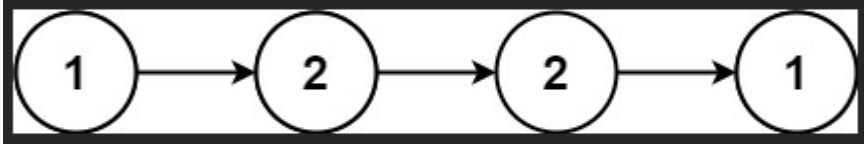
234. Palindrome Linked List

Solved

Easy | Topics | Companies

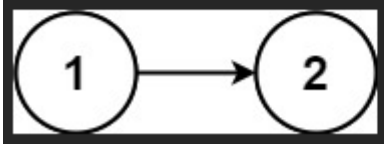
Given the `head` of a singly linked list, return `true` *if it is a palindrome or* `false` *otherwise.*

Example 1:




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

Example 2:

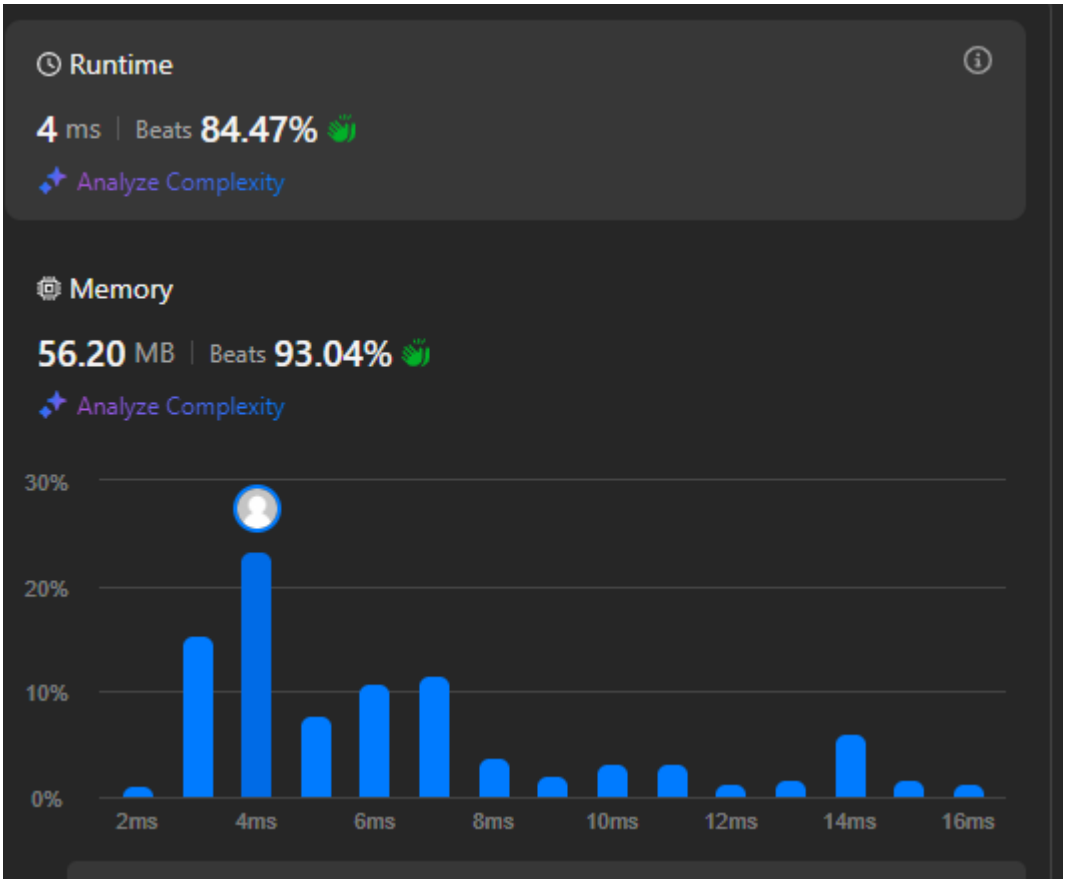



```
Input: head = [1,2]
Output: false
```

Code:

```java
class Solution {
    public boolean isPalindrome(ListNode head) {
        int[] arr=new int[100000];
        int i=0;
        while(head!=null){
            arr[i]=head.val;
            head=head.next;
            i++;
        }
        for(int j=0,k=i-1;j<k;j++,k--){
            if(arr[j]!=arr[k]){
                return false;
            }
        }
        return true;
    }
}
```

Ouput:



3.

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.

Example 1:

```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

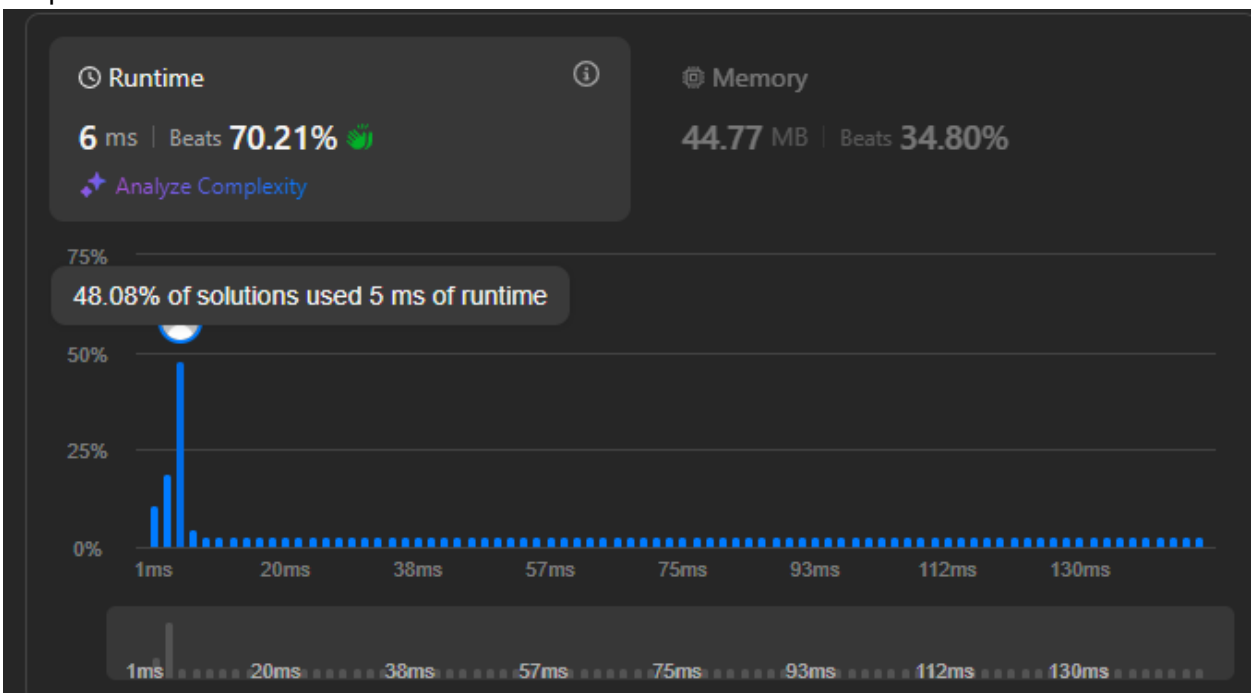
```
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Code:

```java
class Solution {
    public int lengthOfLongestSubstring(String s) {
        HashMap<Character,Integer> h1=new HashMap<>();
        int l=0,r=0;
        int len=0;
        while(r<s.length()){
            if(!h1.containsKey(s.charAt(r))){
                h1.put(s.charAt(r),r);
            }else{
                if(h1.get(s.charAt(r))>=l) {
                    l=h1.get(s.charAt(r))+1;
                }
                h1.put(s.charAt(r),r);
            }
            len=Math.max(len,r-l+1);
            r++;
        }
        return len;
    }
}
```

Output:

Output:


Runtime
6 ms | Beats 70.21%
Analyze Complexity
Memory
44.77 MB | Beats 34.80%
75%
48.08% of solutions used 5 ms of runtime
50%
25%
0%
1ms
20ms
38ms
57ms
75ms
93ms
112ms
130ms

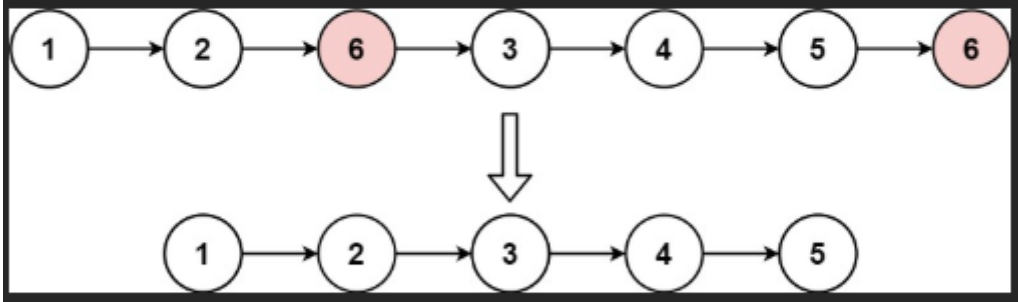
4. # 203. Remove Linked List Elements

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]
```

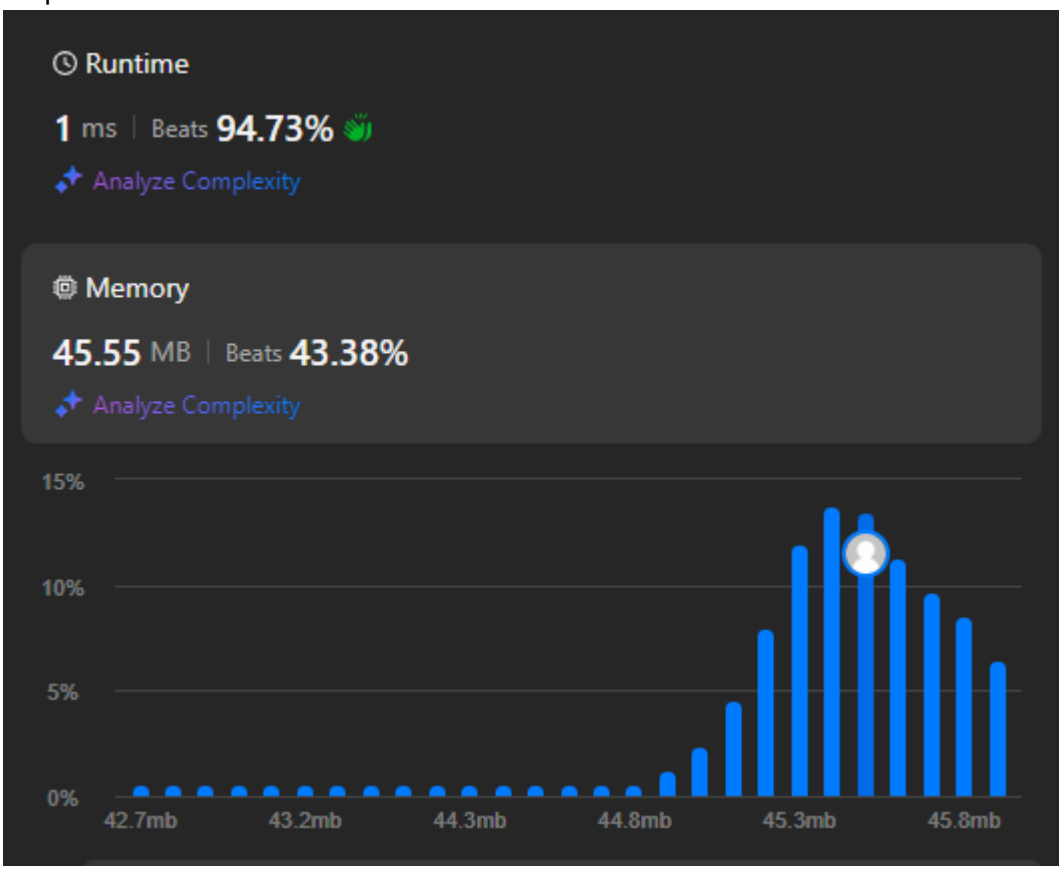
Example 2:

```
Input: head = [], val = 1
Output: []
```

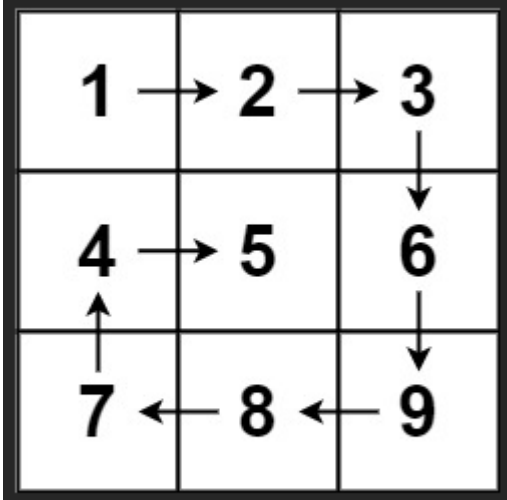
Code:

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

output:



5. **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> result = new ArrayList<>();
        if (matrix == null || matrix.length == 0) {
            return result;
        }

        int rows = matrix.length, cols = matrix[0].length;
        int left = 0, right = cols-1, top = 0, bottom = rows-1;

        while (left <= right && top <= bottom) {
            for (int i = left; i <= right; i++) {
                result.add(matrix[top][i]);
            }
            top++;
```

```
        for (int i = top; i <= bottom; i++) {
            result.add(matrix[i][right]);
        }
        right--;

        if (top <= bottom) {
            for (int i = right; i >= left; i--) {
                result.add(matrix[bottom][i]);
            }
            bottom--;
        }

        if (left <= right) {
            for (int i = bottom; i >= top; i--) {
                result.add(matrix[i][left]);
            }
            left++;
        }
    }
    return result;
  }
}
```

Output:

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

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.
```

6.

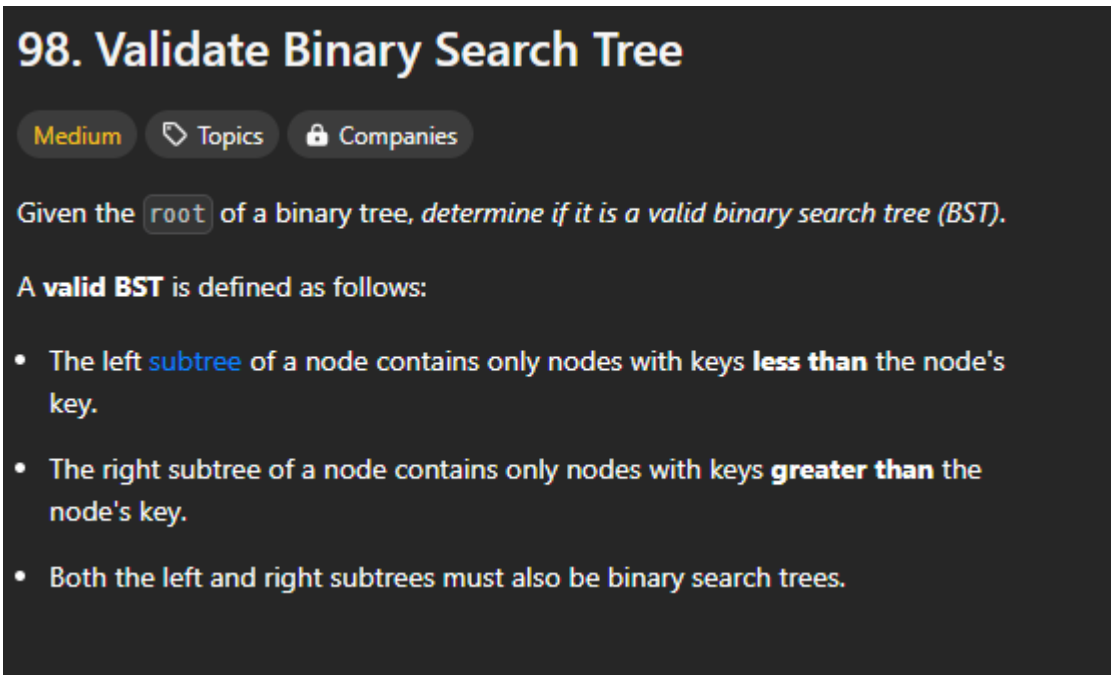
Code:

```
class Solution {
    public int minPathSum(int[][] grid) {
        int m = grid.length;
        int n = grid[0].length;

        for (int i = 1; i < m; i++) {
            grid[i][0] += grid[i-1][0];
        }

        for (int j = 1; j < n; j++) {
```

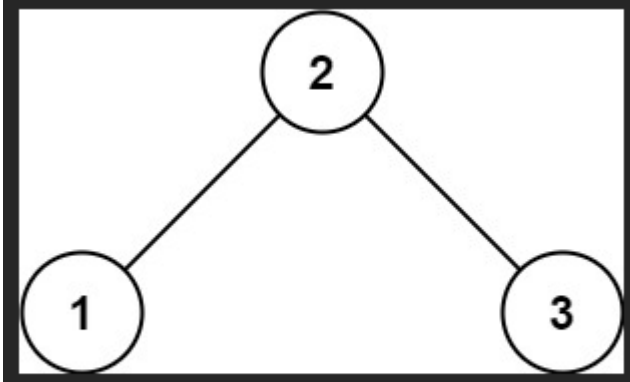
```
            grid[0][j] += grid[0][j-1];
        }

        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];
    }
}
```

Output: 7

7.

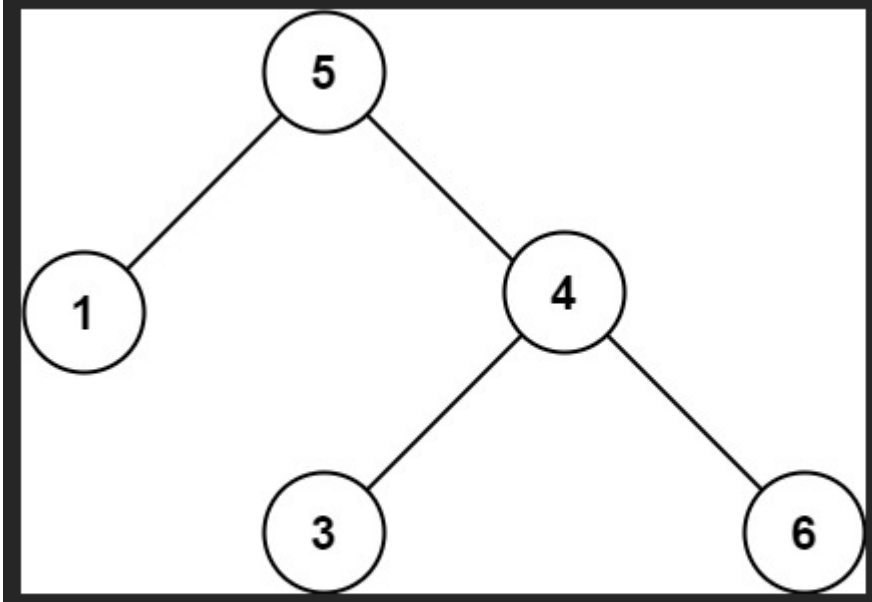


Example 1:




```
Input: root = [2,1,3]
Output: true
```

Example 2:




```
Input: root = [5,1,4,null,null,3,6]
Output: false
Explanation: The root node's value is 5 but its right child's
value is 4.
```

Code:

```java
import java.util.*;
class Solution {
    public boolean isValidBST(TreeNode root) {
        return checkvalid(root,Long.MIN_VALUE,Long.MAX_VALUE);
    }
    public boolean checkvalid(TreeNode root,long min,long max){
        if(root==null){
            return true;
        }
        if(root.val >= max || root.val<=min){
            return false;
        }
        return checkvalid(root.left,min,root.val) &&checkvalid(root.right,root.val,max);
    }
}
```

Output:

```
true
```

Word ladder 1:

8.

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.*

> **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.

Code:

```java
public int ladderLength(String beginWord, String endWord, Set<String> wordList) {
    wordList.add(endWord);
    Queue<String> queue = new LinkedList<String>();
    queue.add(beginWord);
    int level = 0;
    while(!queue.isEmpty()){
        int size = queue.size();
        for(int i = 0; i < size; i++){
            String cur = queue.remove();
            if(cur.equals(endWord)){ return level + 1;}
            for(int j = 0; j < cur.length(); j++){
                char[] word = cur.toCharArray();
                for(char ch = 'a'; ch < 'z'; ch++){
                    word[j] = ch;
                    String check = new String(word);
                    if(!check.equals(cur) && wordList.contains(check)){
                        queue.add(check);
                        wordList.remove(check);
                    }
                }
            }
        }
        level++;
    }
    return 0;
}
```

Output:

```
5
```

9.

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 *all the* ***shortest transformation sequences*** *from* `beginWord` *to* `endWord`, *or an empty list if no such sequence exists. Each sequence should be returned as a list of the words* `[beginWord, s1, s2, ..., sk]`.

Example 1:

```
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]
Output: [["hit","hot","dot","dog","cog"],["hit","hot","lot","log","cog"]]
Explanation: There are 2 shortest transformation sequences:
"hit" -> "hot" -> "dot" -> "dog" -> "cog"
"hit" -> "hot" -> "lot" -> "log" -> "cog"
```

Example 2:

```
Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log"]
Output: []
```

Output:

```
[["hit","hot","dot","dog","cog"],["hit","hot","lot","log","cog"]]
```

10.

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] = [a_i, b_i]` indicates that you **must** take course `b_i` first if you want to take course `a_i`.

- 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`.

Example 1:

```
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0. So it is possible.
```

Example 2:

```
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1.
So it is impossible.
```

output:

```java
class Solution {
    public boolean detectCycle(List<List<Integer>> v, int src, int[] rst, int[] vis) {
        vis[src] = 1;
        rst[src] = 1;
        for (int x : v.get(src)) {
            if (vis[x] == 0 && detectCycle(v, x, rst, vis)) {
                return true;
            } else if (rst[x] == 1) {
                return true;
            }
        }
        rst[src] = 0;
        return false;
    }

    public boolean canFinish(int numCourses, int[][] prerequisites) {
        List<List<Integer>> v = new ArrayList<>();
        for (int i = 0; i < numCourses; i++) {
            v.add(new ArrayList<>());
        }
        Stack<Integer> s = new Stack<>();
        int[] vis = new int[numCourses];
        int[] rst = new int[numCourses];
        for (int[] x : prerequisites) {
            v.get(x[1]).add(x[0]);
        }
        for (int i = 0; i < numCourses; i++) {
            if (vis[i] == 0 && detectCycle(v, i, rst, vis)) {
                return false;
            }
        }
        return true;
    }
}
```

Output:

```
true
```