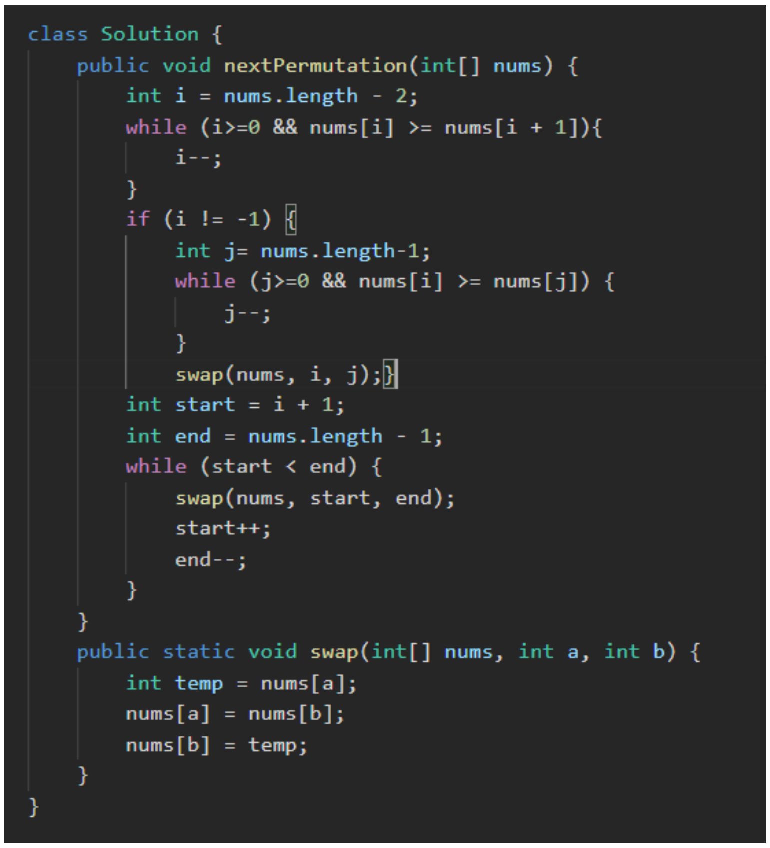
DSA PRACTICE PROBLEMS – SET 7

Madhulekha R – AIML - [19/11/2024]

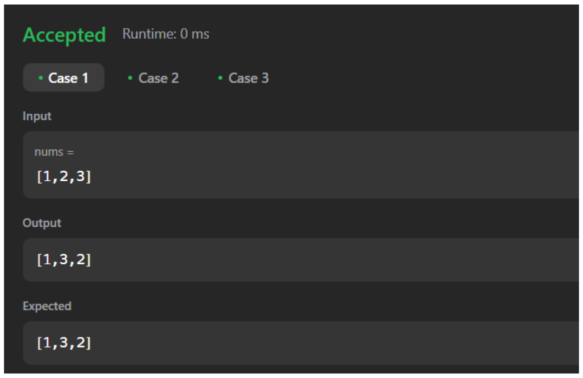
1. Question: Next Permutation - [Leetcode]

Code:



```
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;
    }
}
```

Output:



```
Accepted   Runtime: 0 ms

• Case 1   • Case 2   • Case 3

Input
nums =
[1,2,3]

Output
[1,3,2]

Expected
[1,3,2]
```

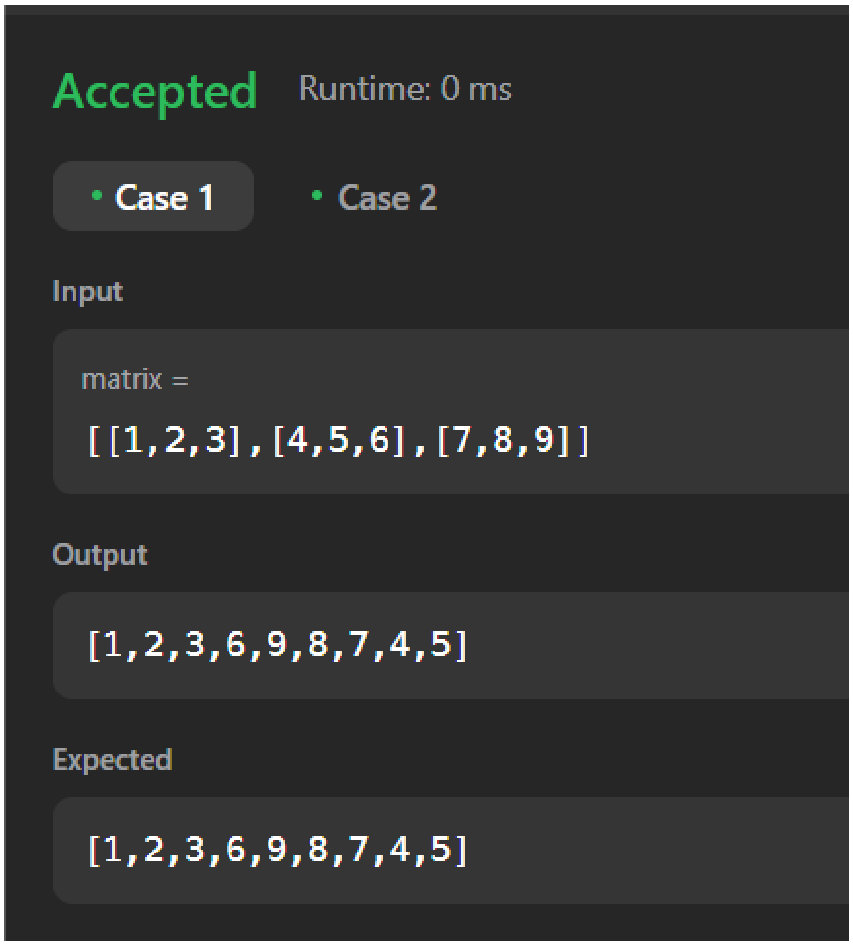
Time Complexity: O(n)
Space Complexity: O(1)

2. **Question: Spiral Matrix - [Leetcode]**

Code:

```java
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        int d = matrix.length-1;
        int r = matrix[0].length-1;
        int u=0,l=0,i=0,j=0,n=0;
        int a = (r+1)*(d+1);
        ArrayList<Integer> list = new ArrayList<>();
        while(true){
            while(j<=r){
                list.add(matrix[i][j]);
                n++;
                j++;
            }
            if(n==a){
                break;
            }
            u++;
            j--;
            i=u;
            while(i<=d){
                list.add(matrix[i][j]);
                n++;
                i++;
            }
            if(n==a){
                break;
            }
            r--;
            i--;
            j=r;
            while(j>=l){
                list.add(matrix[i][j]);
                n++;
                j--;
            }
            if(n==a){
                break;
            }
            d--;
            j++;
            i=d;
            while(i>=u){
                list.add(matrix[i][j]);
                n++;
                i--;
            }
            if(n==a){
                break;
            }
            l++;
            i++;
            j=l;
        }
        return list;
    }
}
```

Output:



Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

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

Output

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

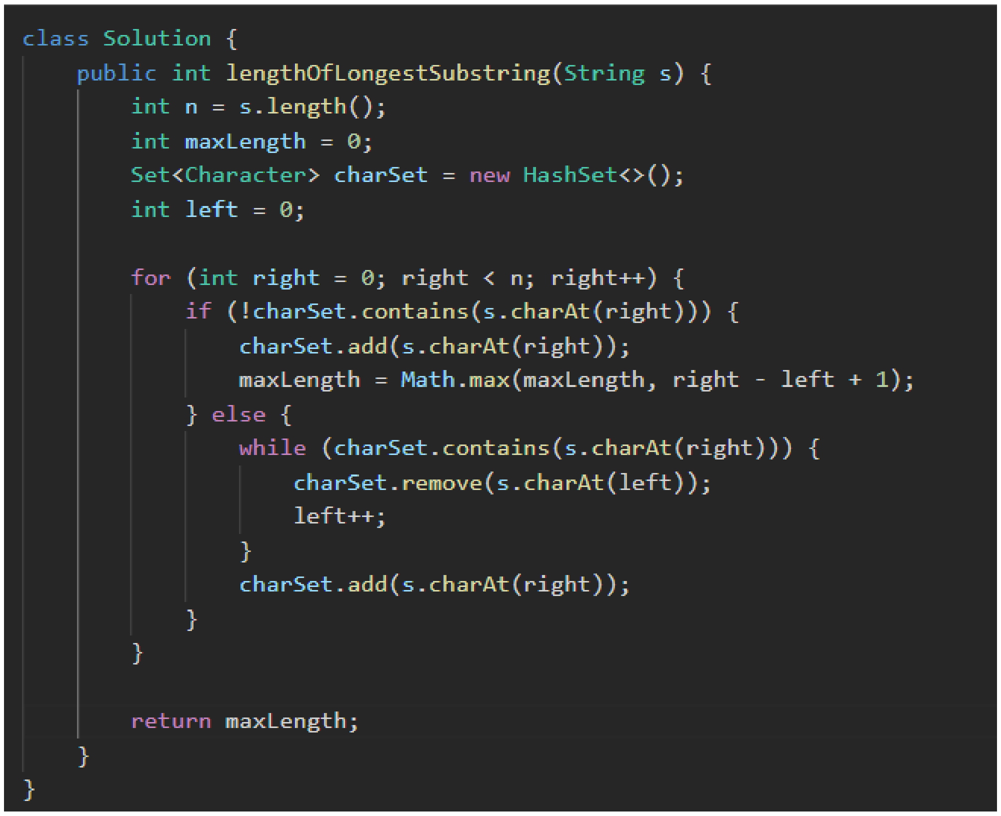
Expected

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

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

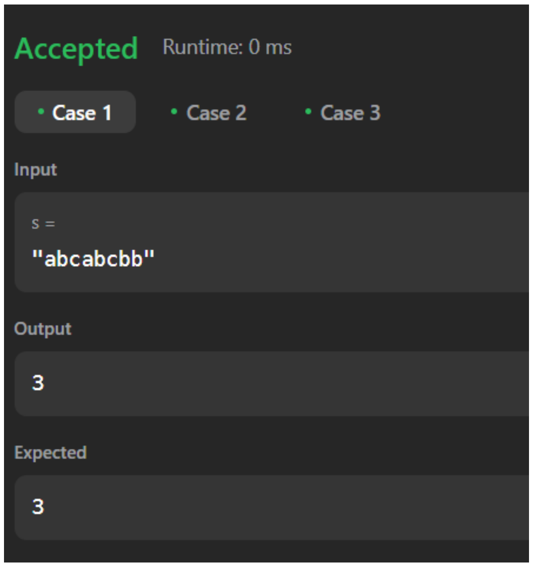
3. **Question: Longest Substring without repeating character - [Leetcode]**

Code:



```java
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int n = s.length();
        int maxLength = 0;
        Set<Character> charSet = new HashSet<>();
        int left = 0;

        for (int right = 0; right < n; right++) {
            if (!charSet.contains(s.charAt(right))) {
                charSet.add(s.charAt(right));
                maxLength = Math.max(maxLength, right - left + 1);
            } else {
                while (charSet.contains(s.charAt(right))) {
                    charSet.remove(s.charAt(left));
                    left++;
                }
                charSet.add(s.charAt(right));
            }
        }

        return maxLength;
    }
}
```

Output:

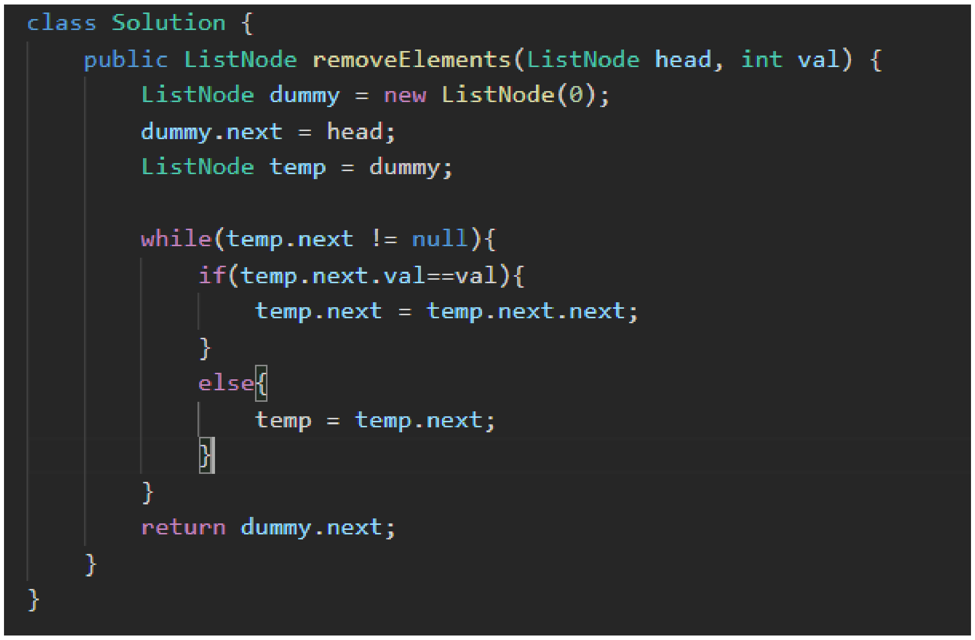


```
Accepted   Runtime: 0 ms

• Case 1   • Case 2   • Case 3

Input
s =
"abcabcbb"

Output
3

Expected
3
```

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

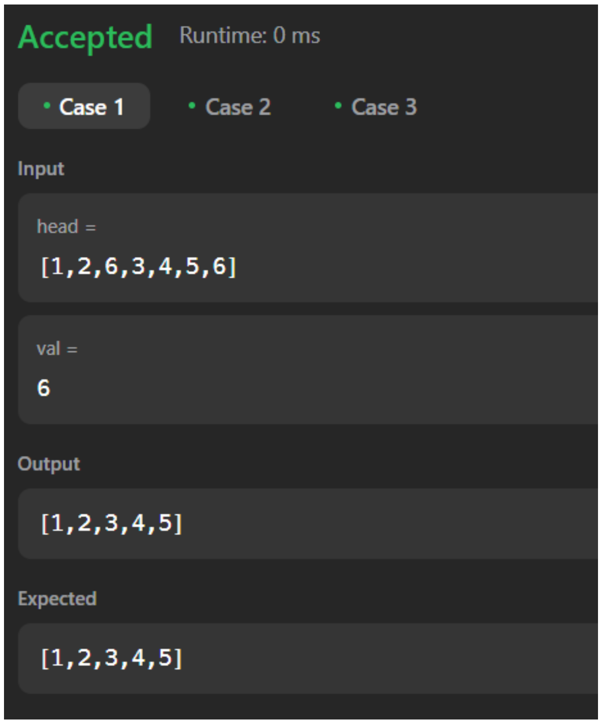
4. Question: Remove Linked list elements - [Leetcode]

Code:



```
class Solution {
    public ListNode removeElements(ListNode head, int val) {
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode temp = dummy;

        while(temp.next != null){
            if(temp.next.val==val){
                temp.next = temp.next.next;
            }
            else{
                temp = temp.next;
            }
        }
        return dummy.next;
    }
}
```

Output:



```
Accepted   Runtime: 0 ms

• Case 1   • Case 2   • Case 3

Input

head =
[1,2,6,3,4,5,6]

val =
6

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

Expected
[1,2,3,4,5]
```

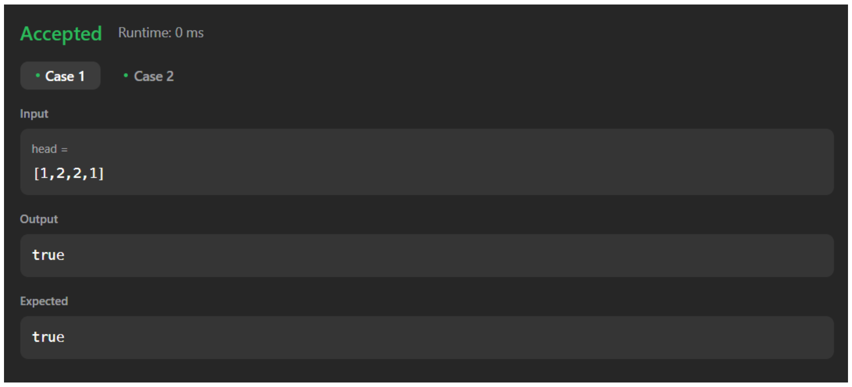
Time Complexity: O(n)
Space Complexity: O(1)

5. **Question: Palindrome linked list - [Leetcode]**

Code:

```
class Solution {
    public boolean isPalindrome(ListNode head) {
        if(head==null || head.next==null) return true;
        ListNode Mid=Mid(head);
        ListNode RevList=Reverse(Mid.next);
        ListNode T1=head;
        ListNode T2=RevList;
        Mid.next = null;
        while(T2!=null){
            if (T1.val != T2.val) return false;
            T1 = T1.next;
            T2 = T2.next;
        }
        return true;
    }
    public ListNode Mid(ListNode head){
        ListNode T=head;
        ListNode fast=head;
        ListNode slow=head;
        while(fast.next!=null && fast.next.next!=null){
            slow=slow.next;
            fast=fast.next.next;
        }
        return slow;
    }
    public ListNode Reverse(ListNode head){
        ListNode T=head;
        ListNode Prev=null;
        while(T!=null){
            ListNode Front=T.next;
            T.next=Prev;
            Prev=T;
            T=Front;
        }
        return Prev;
    }
}
```

Output:

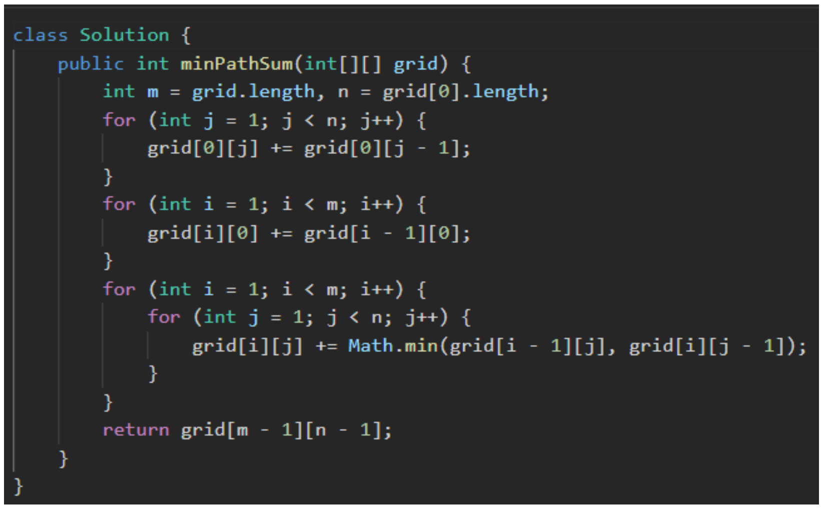


```
Accepted   Runtime: 0 ms
Case 1   Case 2
Input
head =
[1,2,2,1]
Output
true
Expected
true
```

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

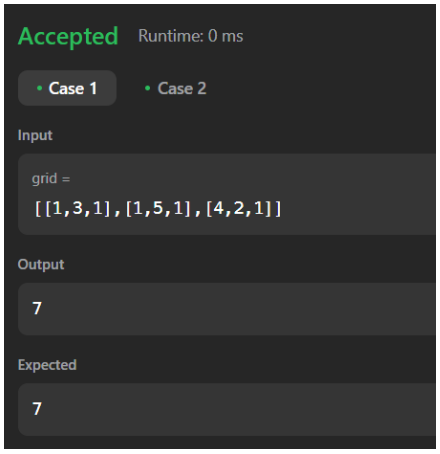
6. **Question: Minimum Path Sum - [Leetcode]**

Code:



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



```
Accepted   Runtime: 0 ms
Case 1   Case 2
Input
grid =
[[1,3,1],[1,5,1],[4,2,1]]
Output
7
Expected
7
```

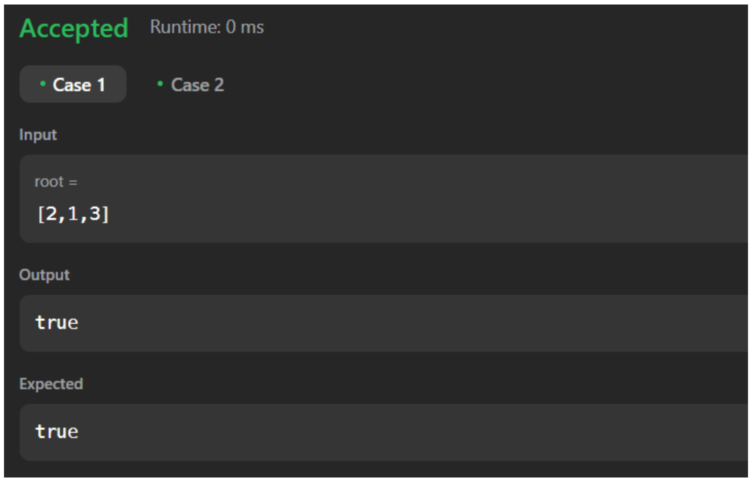
Time Complexity: O(m**n)
Space Complexity: O(1)

7. **Question: Validate Binary Search Tree - [Leetcode]**

Code:

```
class Solution {
    public boolean isValidBST(TreeNode root) {
        if(root == null || (root.left == null && root.right== null)){
            return true;
        }
        return isvalid(root , Long.MIN_VALUE ,Long.MAX_VALUE);
    }

    public boolean isvalid(TreeNode root , long min , long max){
        if(root == null ){
            return true;
        }
        if(root.val >=max || root.val <= min){
            return false;
        }
        return isvalid(root.left , min , root.val) && isvalid(root.right , root.val ,
max);
    }
}
```

Output:



Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =
```
[2,1,3]
```

Output
```
true
```

Expected
```
true
```

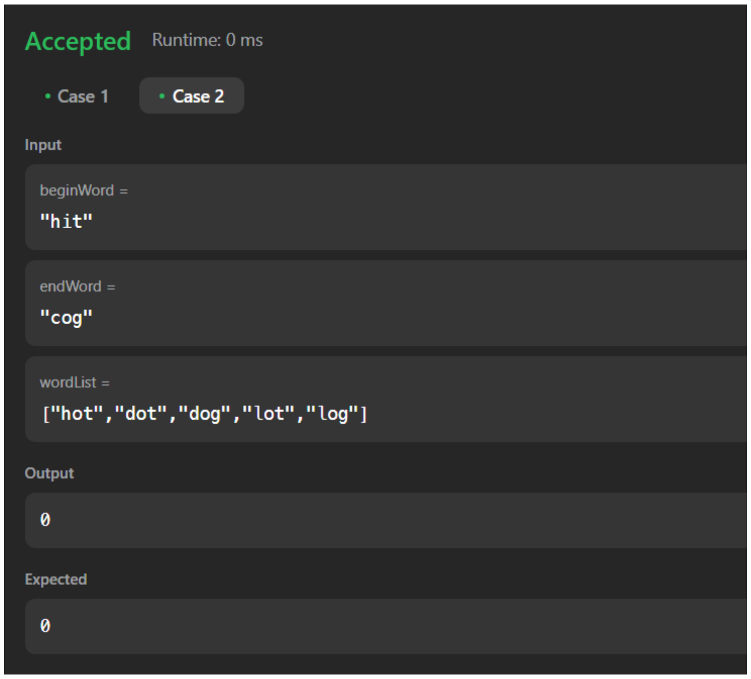
Time Complexity: O(log n)
Space Complexity: O(log n)

8. **Question: Word ladder - [Leetcode]**

Code:

```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);
                    }
                }
            }
            length++;
        }

        return 0;
    }

    private List<String> getNeighbors(String word, Set<String> wordSet) {
        List<String> neighbors = new ArrayList<>();
        char[] wordChars = word.toCharArray();

        for (int i = 0; i < wordChars.length; i++) {
            char originalChar = wordChars[i];
            for (char c = 'a'; c <= 'z'; c++) {
                if (c == originalChar) continue;
                wordChars[i] = c;
                String transformedWord = new String(wordChars);
                if (wordSet.contains(transformedWord)) {
                    neighbors.add(transformedWord);
                }
            }
            wordChars[i] = originalChar;
        }

        return neighbors;
    }
}
```

Output:



```
Accepted    Runtime: 0 ms

• Case 1    • Case 2

Input

beginWord =
"hit"

endWord =
"cog"

wordList =
["hot","dot","dog","lot","log"]

Output

0

Expected

0
```

Time Complexity: O (m*n)
Space Complexity: O (m+n)

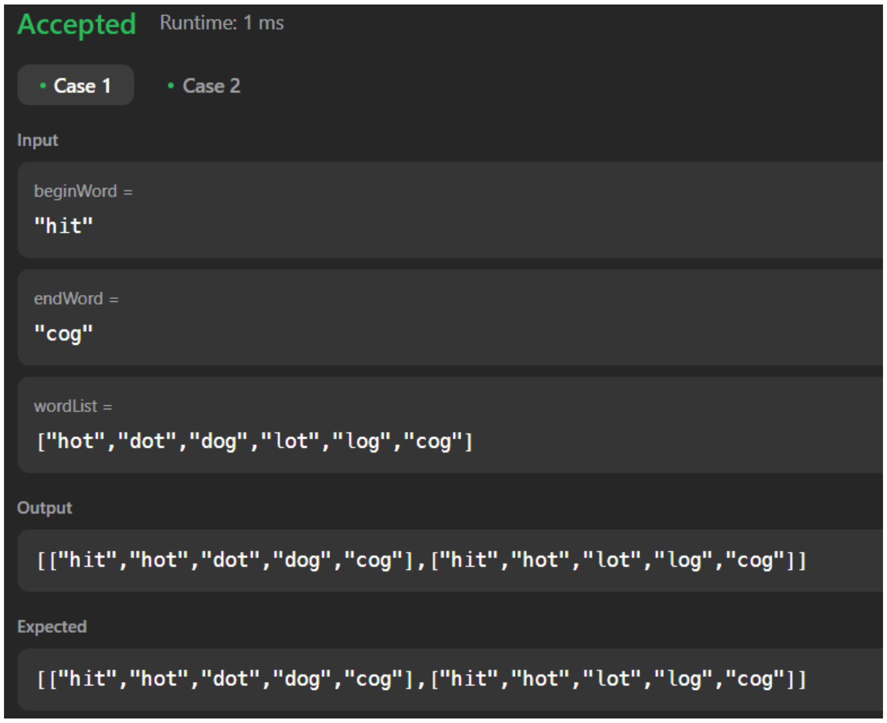
9. **Question: Word ladder II - [Leetcode]**

Code:

```
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);
```

```java
        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);

                    if(hs.contains(replaceString)){
                        q.add(replaceString);
                        hm.put(replaceString,level+1);
                        hs.remove(replaceString);
                    }
                }
            }
        }

        if(hm.containsKey(endWord) == true){
            List<String> seq = new ArrayList<>();
            seq.add(endWord);
            dfs(endWord,seq,res,beginWord,hm);
        }
        return res;
    }

    public void dfs(String word,List<String> seq,List<List<String>> res,String
beginWord,Map<String,Integer> hm){
        if(word.equals(beginWord)){
            List<String> ref = new ArrayList<>(seq);
            Collections.reverse(ref);
            res.add(ref);
            return;
        }

        int level = hm.get(word);
        for(int i=0;i<word.length();i++){
            for(char ch ='a';ch<='z';ch++){
                char replaceChars[] = word.toCharArray();
                replaceChars[i] = ch;
                String replaceStr = new String(replaceChars);
                if(hm.containsKey(replaceStr) && hm.get(replaceStr) == level-1){
                    seq.add(replaceStr);
                    dfs(replaceStr,seq,res,beginWord,hm);
                    seq.remove(seq.size()-1);
                }
            }
        }
    }
}
```

Output:



```
Accepted   Runtime: 1 ms

Case 1   Case 2

Input

beginWord =
"hit"

endWord =
"cog"

wordList =
["hot","dot","dog","lot","log","cog"]

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

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

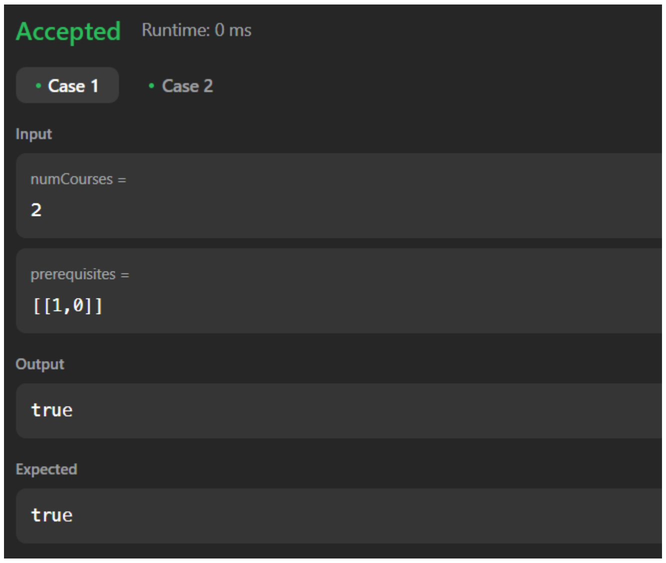
Time Complexity: O (n*m)
Space Complexity: O ((n+k) * m)

10. **Question: Course Schedule - [Leetcode]**

Code:

```
class Solution {
    static int v;
    static int e;

    static ArrayList<Integer> graph[];

    static void addEdge(int a, int b){
        graph[b].add(a);
    }

    public boolean canFinish(int num, int[][] pre) {

        v = num;
        e = pre.length;
        graph = new ArrayList[v];
        for(int i=0; i<v; i++){
            graph[i] = new ArrayList<>();
        }

        for(int i=0; i< pre.length; i++){
            addEdge(pre[i][0],pre[i][1]);
        }

        int indegree[] = new int[v];
        for(int i=0; i< pre.length; i++){
            indegree[pre[i][0]]++;
        }


        Queue<Integer> q = new LinkedList<>();
        for(int i=0; i<indegree.length; i++){
            if(indegree[i]==0) q.add(i);
        }
        while (q.size()!=0){
            int a = q.remove();
            for(var x : graph[a]){
                indegree[x]--;
                if(indegree[x]==0) q.add(x);
            }
        }

        for(int i=0; i<indegree.length; i++){
            if(indegree[i]!=0) return false;
        }
        return true;
    }
}
```

Output:



```
Accepted Runtime: 0 ms
• Case 1 • Case 2
Input
numCourses =
2
prerequisites =
[[1,0]]
Output
true
Expected
true
```

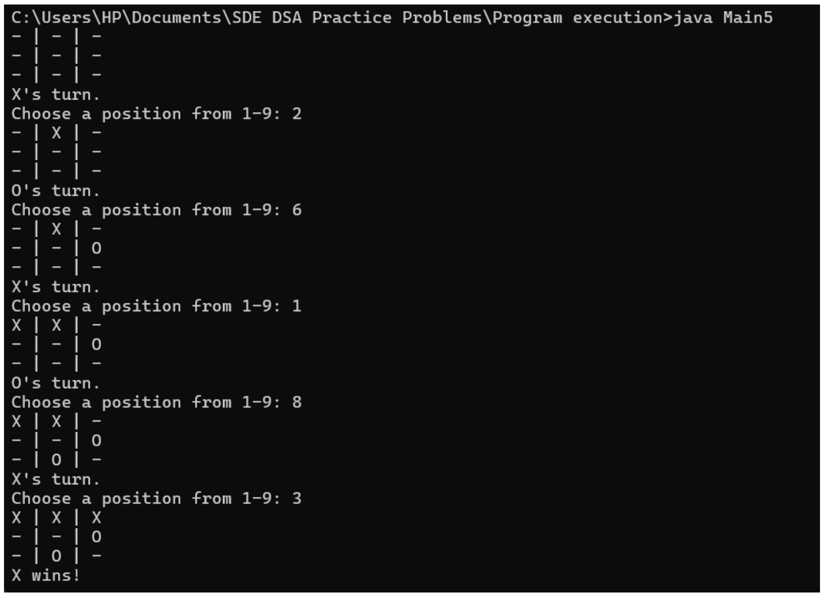
Time Complexity: O (n+m)
Space Complexity: O (n+m)

11. **Question: Design tic tac toe - [GFG]**

Code:

```
import java.util.Scanner;

public class Main5{
    static String[] board = {"-", "-", "-", "-", "-", "-", "-", "-", "-"};

    static void printBoard() {
        System.out.println(board[0] + " | " + board[1] + " | " + board[2]);
        System.out.println(board[3] + " | " + board[4] + " | " + board[5]);
        System.out.println(board[6] + " | " + board[7] + " | " + board[8]);
    }

    static void takeTurn(String player) {
        Scanner scanner = new Scanner(System.in);
        System.out.println(player + "'s turn.");
        System.out.print("Choose a position from 1-9: ");
        int position = scanner.nextInt() - 1;
        while (position < 0 || position > 8 || !board[position].equals("-")) {
            System.out.print("Invalid input or position already taken. Choose a different position: ");
            position = scanner.nextInt() - 1;
        }
        board[position] = player;
        printBoard();
    }

    static String checkGameOver(){
        if ((board[0].equals(board[1]) && board[1].equals(board[2]) && !board[0].equals("-")) ||
                (board[3].equals(board[4]) && board[4].equals(board[5]) && !board[3].equals("-")) ||
                (board[6].equals(board[7]) && board[7].equals(board[8]) && !board[6].equals("-")) ||
                (board[0].equals(board[3]) && board[3].equals(board[6]) && !board[0].equals("-")) ||
                (board[1].equals(board[4]) && board[4].equals(board[7]) && !board[1].equals("-")) ||
                (board[2].equals(board[5]) && board[5].equals(board[8]) && !board[2].equals("-")) ||
                (board[0].equals(board[4]) && board[4].equals(board[8]) && !board[0].equals("-")) ||
                (board[2].equals(board[4]) && board[4].equals(board[6]) && !board[2].equals("-"))) {
            return "win";
        }
        else if (!String.join("", board).contains("-")) {
            return "tie";
        }
        else {
            return "play";
        }
    }

    public static void main(String[] args) {
        printBoard();
        String currentPlayer = "X";
        boolean gameOver = false;
        while (!gameOver) {
            takeTurn(currentPlayer);
            String gameResult = checkGameOver();
            if (gameResult.equals("win")) {
                System.out.println(currentPlayer + " wins!");
                gameOver = true;
            } else if (gameResult.equals("tie")) {
                System.out.println("It's a tie!");
                gameOver = true;
            } else {
                currentPlayer = currentPlayer.equals("X") ? "O" : "X";
            }
        }
    }
}
```

Output:



```
C:\Users\HP\Documents\SDE DSA Practice Problems\Program execution>java Main5
- | - | -
- | - | -
- | - | -
X's turn.
Choose a position from 1-9: 2
- | X | -
- | - | -
- | - | -
O's turn.
Choose a position from 1-9: 6
- | X | -
- | - | O
- | - | -
X's turn.
Choose a position from 1-9: 1
X | X | -
- | - | O
- | - | -
O's turn.
Choose a position from 1-9: 8
X | X | -
- | - | O
- | O | -
X's turn.
Choose a position from 1-9: 3
X | X | X
- | - | O
- | O | -
X wins!
```

Time Complexity: O(n^3)
Space Complexity: O(n^2)