

**1.Next Permutation****Code:**

```
class Solution {
    public void nextPermutation(int[] nums) {
        int ind1=-1;
        int ind2=-1;
        for(int i=nums.length-2;i>=0;i--){
            if(nums[i]<nums[i+1]){
                ind1=i;
                break;
            }
        }
        if(ind1==-1){
            reverse(nums,0);
        }

        else{
            for(int i=nums.length-1;i>=0;i--){
                if(nums[i]>nums[ind1]){
                    ind2=i;
                    break;
                }
            }

            swap(nums,ind1,ind2);
            reverse(nums,ind1+1);
        }
    }
}
```

```

    }

    void swap(int[] nums,int i,int j){

        int temp=nums[i];

        nums[i]=nums[j];

        nums[j]=temp;

    }

    void reverse(int[] nums,int start){

        int i=start;

        int j=nums.length-1;

        while(i<j){

            swap(nums,i,j);

            i++;

            j--;

        }

    }

}

```

## Output:

```

class Solution {
public void nextPermutation(int[] nums) {
    int ind1=-1;
    int ind2=-1;
    for(int i=nums.length-2;i>=0;i--){
        if(nums[i]<nums[i+1]){
            ind1=i;
            break;
        }
    }
    if(ind1==-1){
        reverse(nums,0);
    }

    else{
        for(int i=nums.length-1;i>=0;i--){
            if(nums[i]>nums[ind1]){
                ind2=i;
                break;
            }
        }

        swap(nums, ind1, ind2);
        reverse(nums, ind1+1);
    }
}
}

```

Testcase | **Test Result**

**Accepted** Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

nums =  
[1, 2, 3]

Output

[1, 3, 2]

Expected

[1, 3, 2]

## 2. Spiral matrix

### Code:

```
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int x = 0;
        int y = 0;
        int dx = 1;
        int dy = 0;
        List<Integer> res = new ArrayList<>();

        for (int i = 0; i < rows * cols; i++) {
            res.add(matrix[y][x]);
            matrix[y][x] = -101;

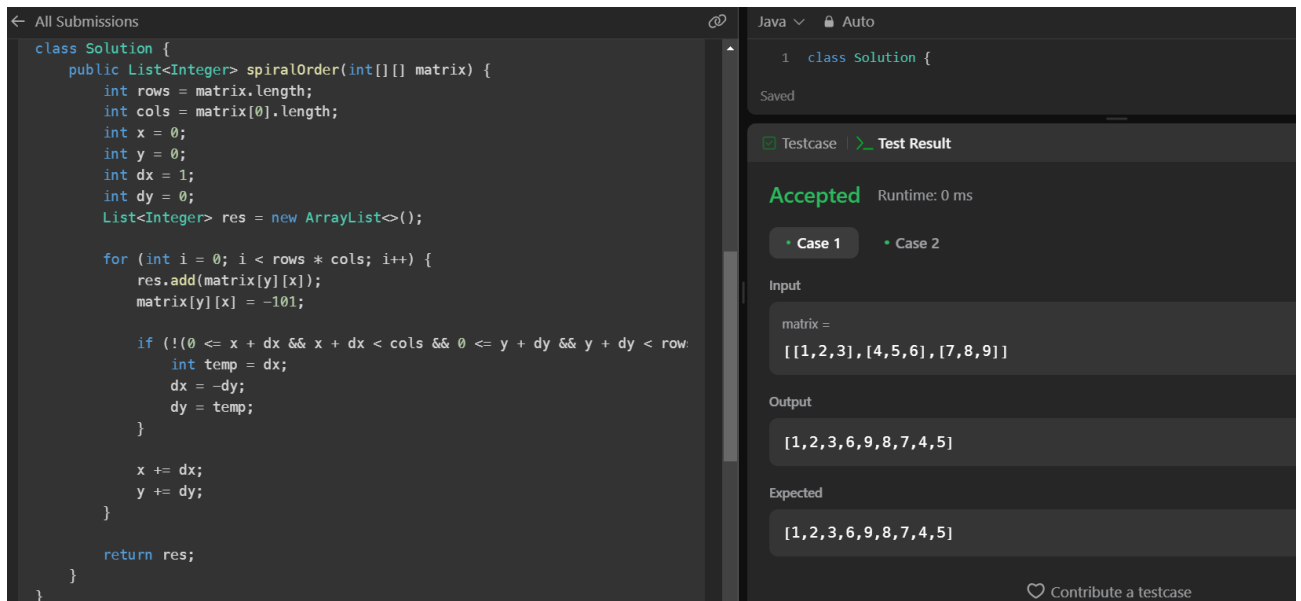
            if (!(0 <= x + dx && x + dx < cols && 0 <= y + dy && y + dy < rows) ||
matrix[y+dy][x+dx] == -101) {
                int temp = dx;
                dx = -dy;
                dy = temp;
            }

            x += dx;
            y += dy;
        }

        return res;
    }
}
```

```
}
```

## Output:



```
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int x = 0;
        int y = 0;
        int dx = 1;
        int dy = 0;
        List<Integer> res = new ArrayList<>();

        for (int i = 0; i < rows * cols; i++) {
            res.add(matrix[y][x]);
            matrix[y][x] = -101;

            if (!(0 <= x + dx && x + dx < cols && 0 <= y + dy && y + dy < row)) {
                int temp = dx;
                dx = -dy;
                dy = temp;
            }

            x += dx;
            y += dy;
        }

        return res;
    }
}
```

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]

Contribute a testcase

### 3. Longest substring without repeating characters

#### Code:

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        int n = s.length();
        int maxLength = 0;
        Map<Character, Integer> charMap = new HashMap<>();
        int left = 0;

        for (int right = 0; right < n; right++) {
            if (!charMap.containsKey(s.charAt(right)) || charMap.get(s.charAt(right)) < left) {
                charMap.put(s.charAt(right), right);
                maxLength = Math.max(maxLength, right - left + 1);
            } else {
                left = charMap.get(s.charAt(right)) + 1;
            }
        }
    }
}
```

```

        charMap.put(s.charAt(right), right);
    }
}

return maxLength;
}
}

```

### Output:

The screenshot displays a code editor with a Java solution for the 'Longest Substring Without Repeating Characters' problem. The code uses a sliding window approach with a HashMap to track the characters in the current window. The test result on the right shows 'Accepted' with a runtime of 0 ms for the input 'abcabcbb'.

```

class Solution {
    public int lengthOfLongestSubstring(String s) {
        int n = s.length();
        int maxLength = 0;
        Map<Character, Integer> charMap = new HashMap<>();
        int left = 0;

        for (int right = 0; right < n; right++) {
            if (!charMap.containsKey(s.charAt(right)) || charMap.get(s.charAt(right)) < left) {
                charMap.put(s.charAt(right), right);
                maxLength = Math.max(maxLength, right - left + 1);
            } else {
                left = charMap.get(s.charAt(right)) + 1;
                charMap.put(s.charAt(right), right);
            }
        }

        return maxLength;
    }
}

```

Testcase | Test Result

**Accepted** Runtime: 0 ms

Case 1 Case 2 Case 3

Input

s = "abcabcbb"

Output

3

Expected

3

## 4. Remove linked list elements

### Code:

```

class Solution {
    public ListNode removeElements(ListNode head, int val) {
        ListNode ans = new ListNode(0, head);
        ListNode dummy = ans;

        while (dummy != null) {
            while (dummy.next != null && dummy.next.val == val) {
                dummy.next = dummy.next.next;
            }
            dummy = dummy.next;
        }
    }
}

```

```

    }

    dummy = dummy.next;
}

return ans.next;
}
}

```

## Output:

The screenshot shows a code editor with a Java solution for removing elements from a linked list. The code defines a `ListNode` class and a `Solution` class with a `removeElements` method. The method uses a dummy node and a while loop to traverse the list and remove nodes with the specified value. The test results panel on the right shows that the solution is accepted for all three test cases. The input for Case 1 is a linked list with values [1, 2, 6, 3, 4, 5, 6] and a value of 6 to be removed. The output is [1, 2, 3, 4, 5], which matches the expected result.

```

*   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) {
        ListNode ans = new ListNode(0, head);
        ListNode dummy = ans;

        while (dummy != null) {
            while (dummy.next != null && dummy.next.val == val) {
                dummy.next = dummy.next.next;
            }
            dummy = dummy.next;
        }

        return ans.next;
    }
}

```

Testcase Test Result

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]

## 5. Palindrome linked list

### Code:

```

class Solution {
    public boolean isPalindrome(ListNode head) {
        Stack<Integer> pila = new Stack<>();
        ListNode aux = head;
        ListNode medio = head;

        while (aux != null && aux.next != null) {
            pila.push(medio.val);
            medio = medio.next;
            aux = aux.next.next;
        }
    }
}

```

```

    }

    if (aux != null) medio = medio.next;

    while (!pila.isEmpty()) {
        int tope = pila.pop();
        if (medio.val != tope) return false;
        medio = medio.next;
    }

    return true;
}
}

```

## Output:

The screenshot displays a code editor on the left and a test result panel on the right. The code is a Java class named `Solution` with a method `isPalindrome` that takes a `ListNode` head and returns a boolean. The method uses a stack to store the values of the first half of the linked list and then compares them with the second half. The test result panel shows that the code passed both test cases, with a runtime of 0 ms.

```

class Solution {
    public boolean isPalindrome(ListNode head) {
        Stack<Integer> pila = new Stack<>();
        ListNode aux = head;
        ListNode medio = head;

        while (aux != null && aux.next != null) {
            pila.push(medio.val);
            medio = medio.next;
            aux = aux.next.next;
        }

        if (aux != null) medio = medio.next;

        while (!pila.isEmpty()) {
            int tope = pila.pop();
            if (medio.val != tope) return false;
            medio = medio.next;
        }

        return true;
    }
}

```

Testcase | Test Result

**Accepted** Runtime: 0 ms

- Case 1
- Case 2

Input

head =  
[1,2,2,1]

Output

true

Expected

true

Contribute a testcase

## 6. Minimum path sum

### Code:

```

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:

```

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];
    }
}

```

Saved

Testcase | Test Result

**Accepted** Runtime: 0 ms

• Case 1 • Case 2

Input

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

Output

7

Expected

7

## 7. Validate binary search tree

### Code:



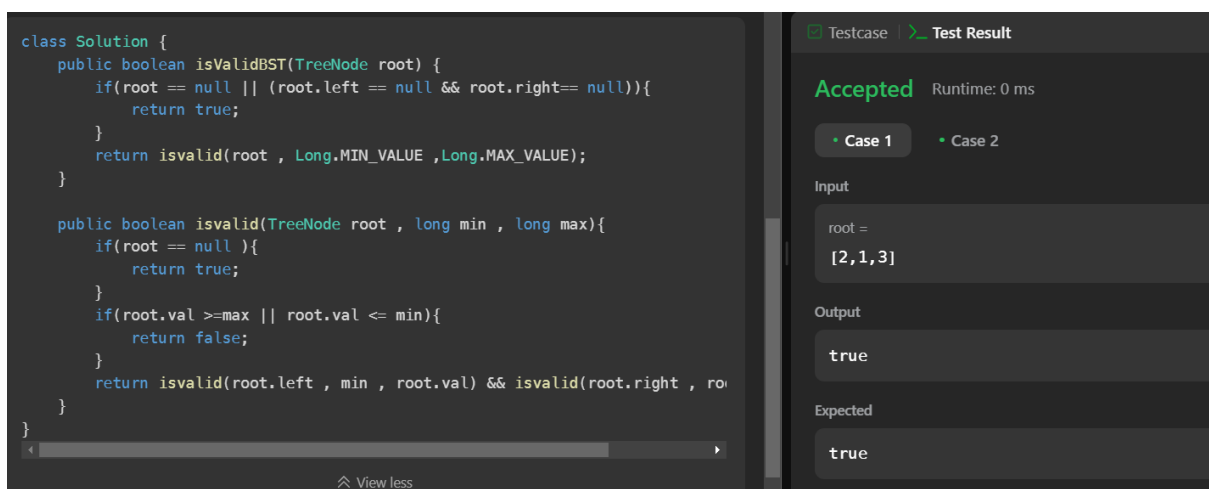
```

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:



The screenshot displays a code editor with the Java code for the `isValidBST` function. The code is color-coded and includes comments. To the right of the code editor, there is a 'Testcase' tab and a 'Test Result' tab. The 'Test Result' tab shows the test status as 'Accepted' with a runtime of 0 ms. Below this, there are two test cases: 'Case 1' and 'Case 2'. The input for 'Case 1' is 'root = [2,1,3]' and the output is 'true'. The expected output is also 'true'.

```

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);
    }
}

```

Testcase | Test Result

**Accepted** Runtime: 0 ms

• Case 1 • Case 2

Input

root =  
[2,1,3]

Output

true

Expected

true

View less

## 8. Word ladder

### Code:

```
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;
    }

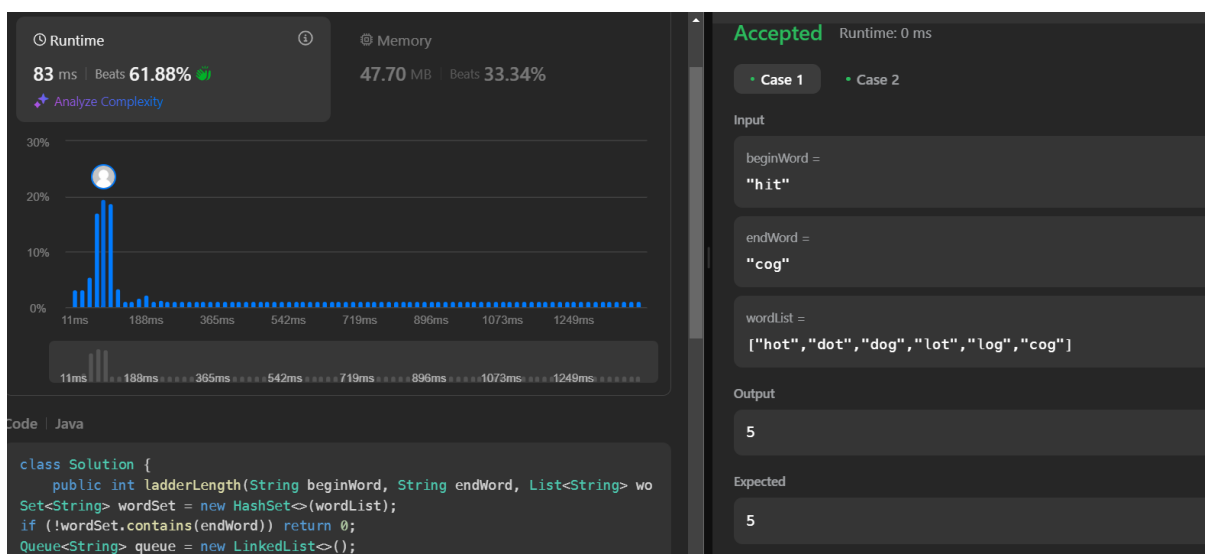
    private List<String> getNeighbors(String word, Set<String> wordSet) {
```

```

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:



## 9.Word ladder -II

### 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);  
  
        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();

```

```

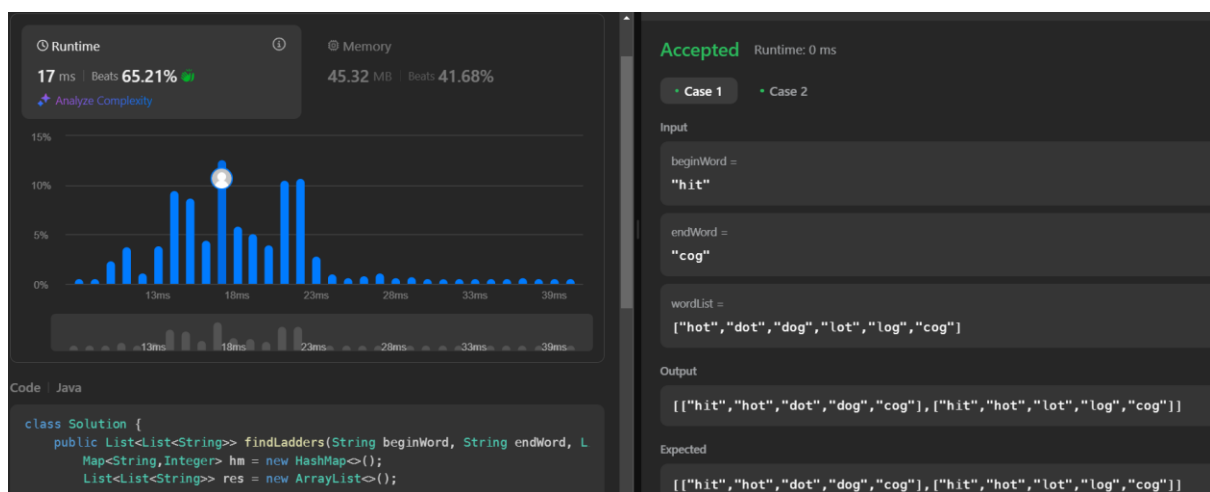
        replaceChars[i] = ch;

        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:



## 10. Course schedule

### Code:

```

class Solution {
    public boolean canFinish(int numCourses, int[][] prerequisites) {
        int counter = 0;
        if (numCourses <= 0) {
            return true;

```

```
}
```

```
int[] inDegree = new int[numCourses];  
List<List<Integer>> graph = new ArrayList<>();  
for (int i = 0; i < numCourses; i++) {  
    graph.add(new ArrayList<>());  
}
```

```
for (int[] edge : prerequisites) {  
    int parent = edge[1];  
    int child = edge[0];  
    graph.get(parent).add(child);  
    inDegree[child]++;  
}
```

```
Queue<Integer> sources = new LinkedList<>();  
for (int i = 0; i < numCourses; i++) {  
    if (inDegree[i] == 0) {  
        sources.offer(i);  
    }  
}
```

```
while (!sources.isEmpty()) {  
    int course = sources.poll();  
    counter++;  
  
    for (int child : graph.get(course)) {  
        inDegree[child]--;  
        if (inDegree[child] == 0) {  
            sources.offer(child);  
        }  
    }  
}
```

```

    }

    }

}

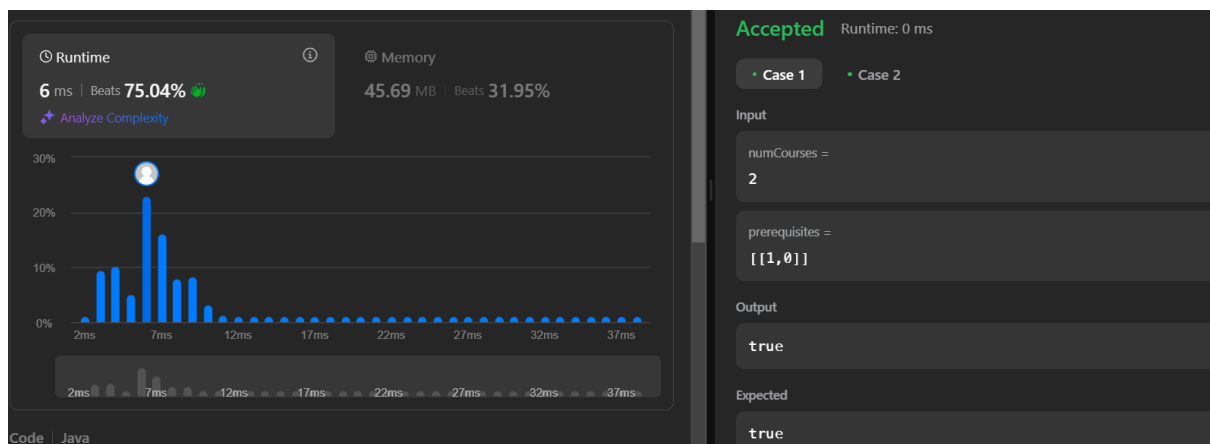
return counter == numCourses;

}

}

```

## Output:



## 11. Design tic tac toe

### Code:

```

class Solution {
    public boolean validTicTacToe(String[] board) {
        int turns = 0;
        boolean xwin = false;
        boolean owin = false;
        int[] rows = new int[3];
        int[] cols = new int[3];
        int diag = 0;
        int antidiag = 0;

        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {

```



```

        if (board[i].charAt(j) == 'X') {
            turns++; rows[i]++; cols[j]++;
            if (i == j) diag++;
            if (i + j == 2) antidiag++;
        } else if (board[i].charAt(j) == 'O') {
            turns--; rows[i]--; cols[j]--;
            if (i == j) diag--;
            if (i + j == 2) antidiag--;
        }
    }
}

xwin = rows[0] == 3 || rows[1] == 3 || rows[2] == 3 ||
      cols[0] == 3 || cols[1] == 3 || cols[2] == 3 ||
      diag == 3 || antidiag == 3;

owin = rows[0] == -3 || rows[1] == -3 || rows[2] == -3 ||
      cols[0] == -3 || cols[1] == -3 || cols[2] == -3 ||
      diag == -3 || antidiag == -3;

if (xwin && turns == 0 || owin && turns == 1) {
    return false;
}

return (turns == 0 || turns == 1) && (!xwin || !owin);
}
}

```

**Output:**

