Maha Gowri S **19/11/2024**

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);
        }
    }
```

```
10          }
11          if(ind1==-1){
12              reverse(nums,0);
```

Saved

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]

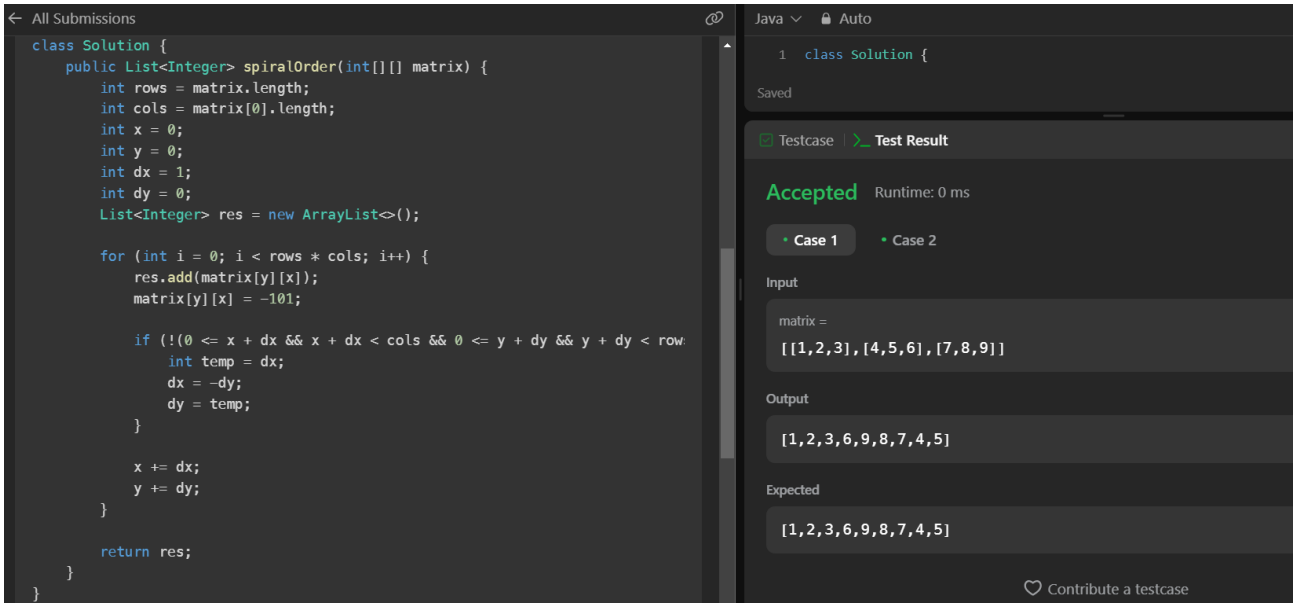
2. Spiral matrix

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:



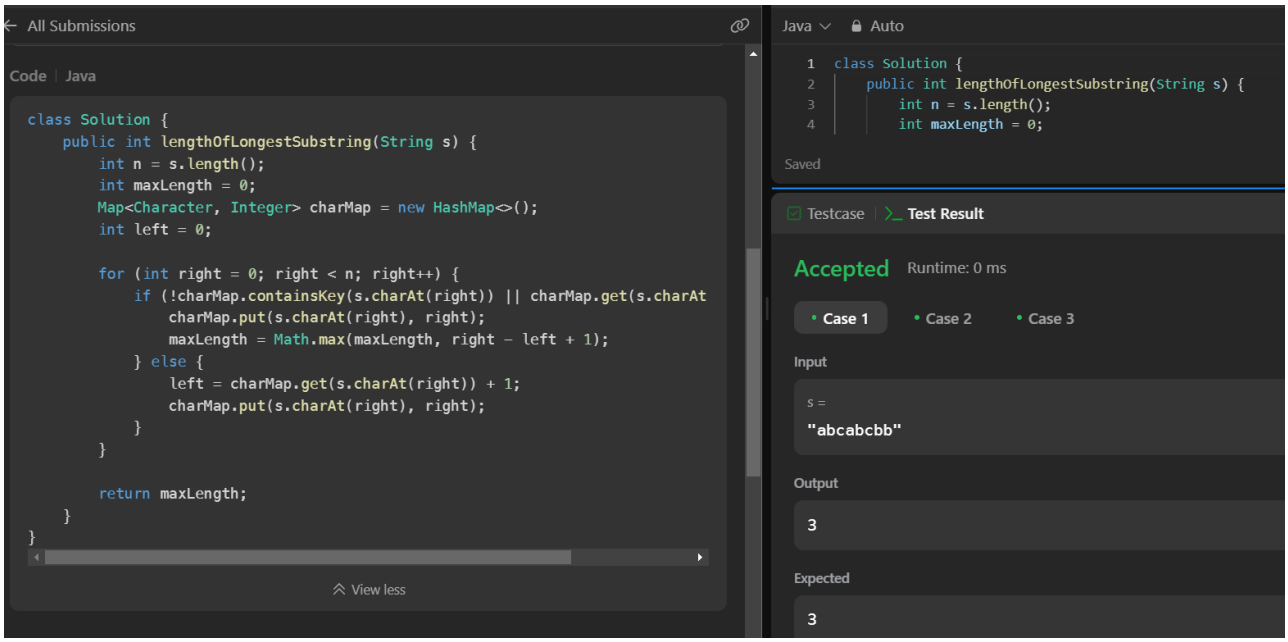
3. Longest substring without repeating characters

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:



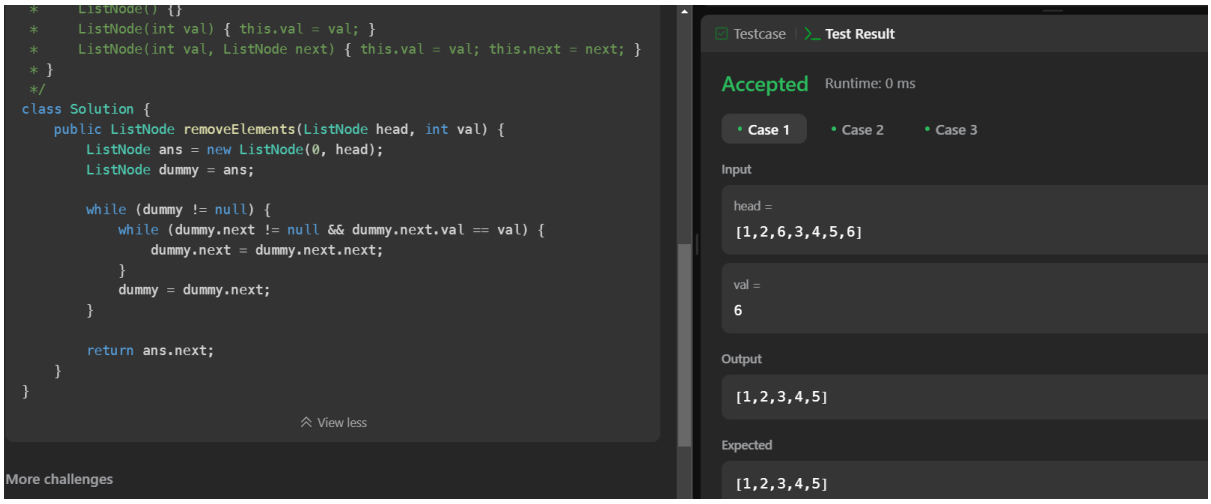
4. Remove linked list elements

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;
    }

    return ans.next;
  }
}
```

Output:



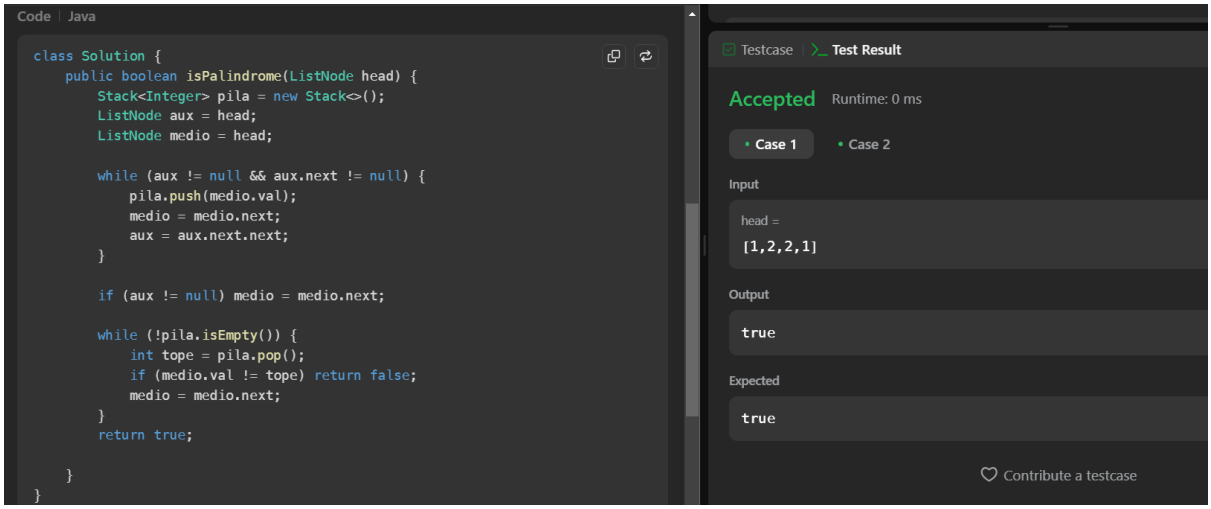
5. Palindrome linked list

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:



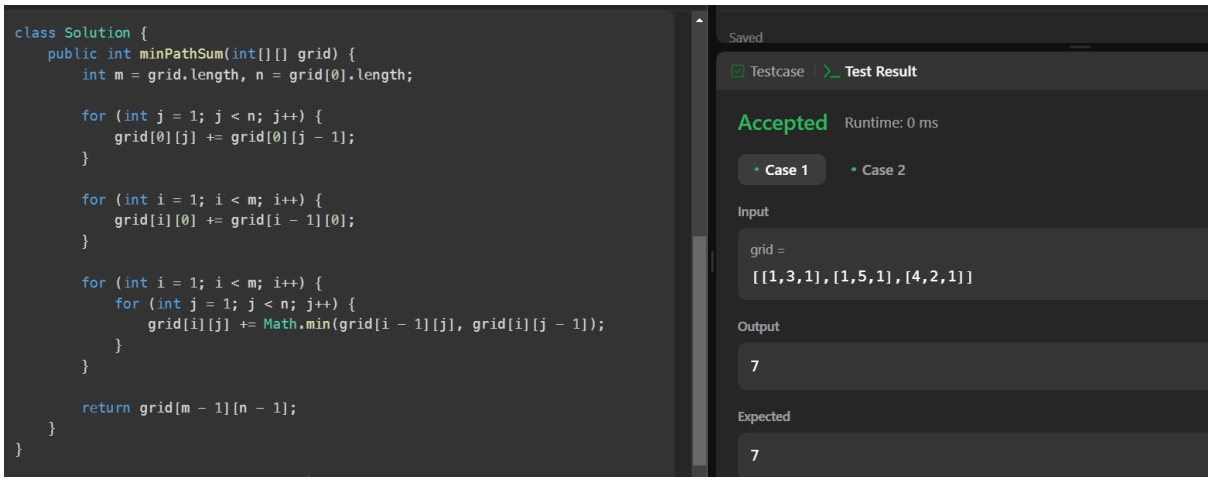
6. Minimum path sum

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:

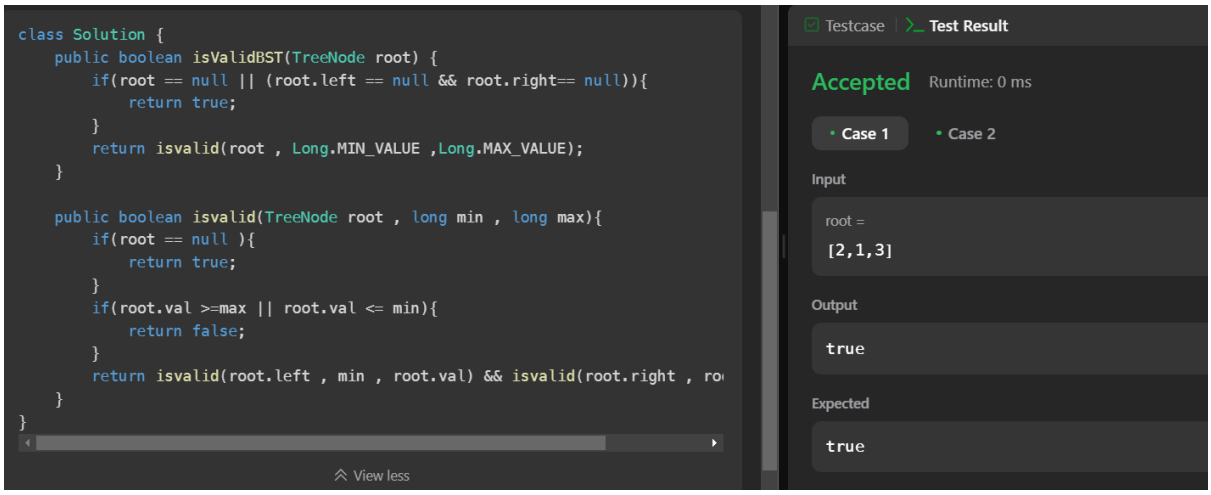


7. Validate binary search tree

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:



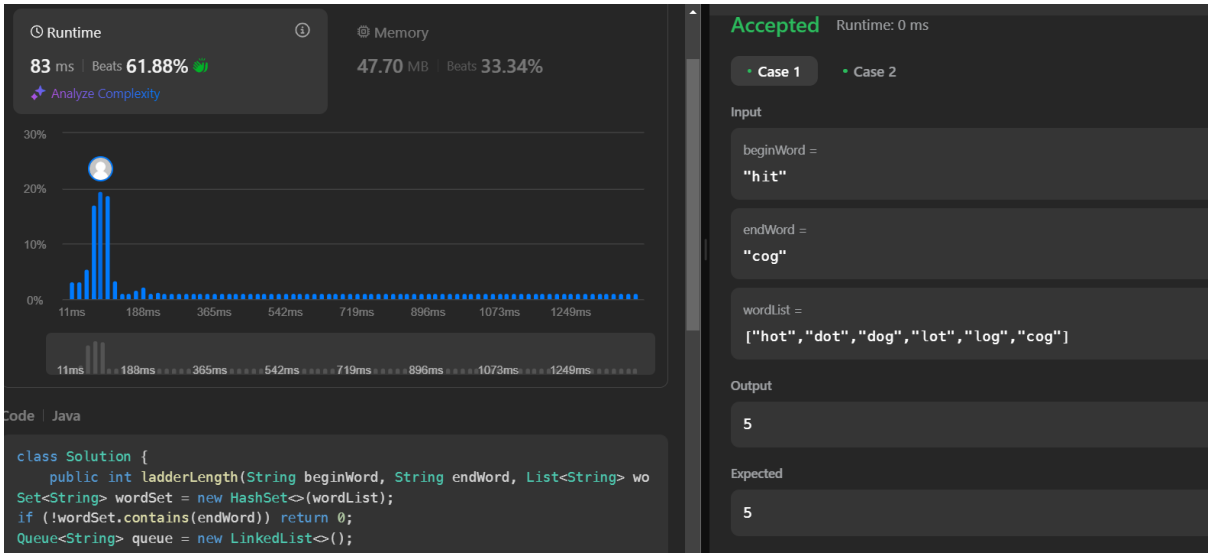
8.Word ladder

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

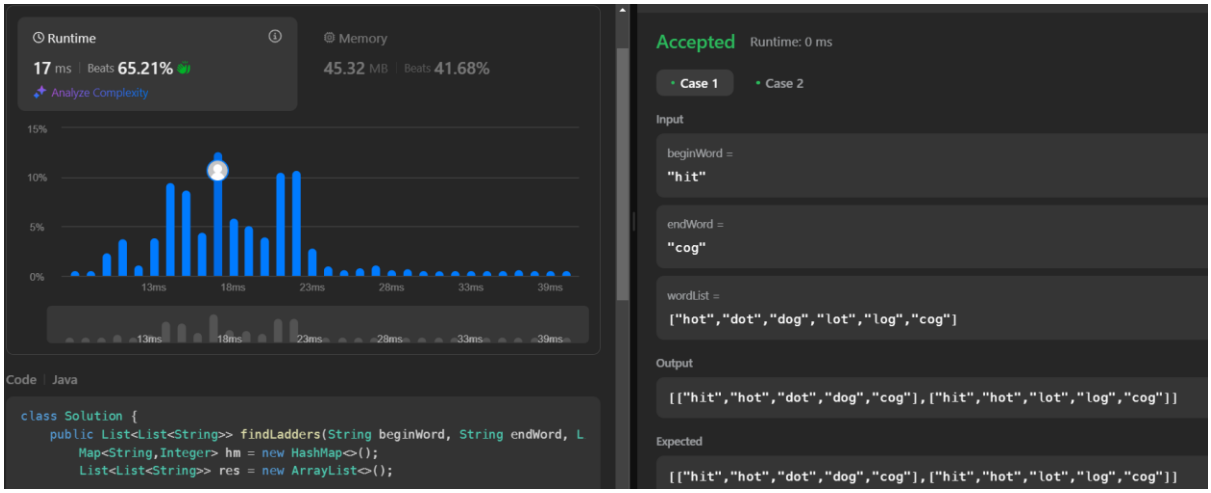
Code:

```
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

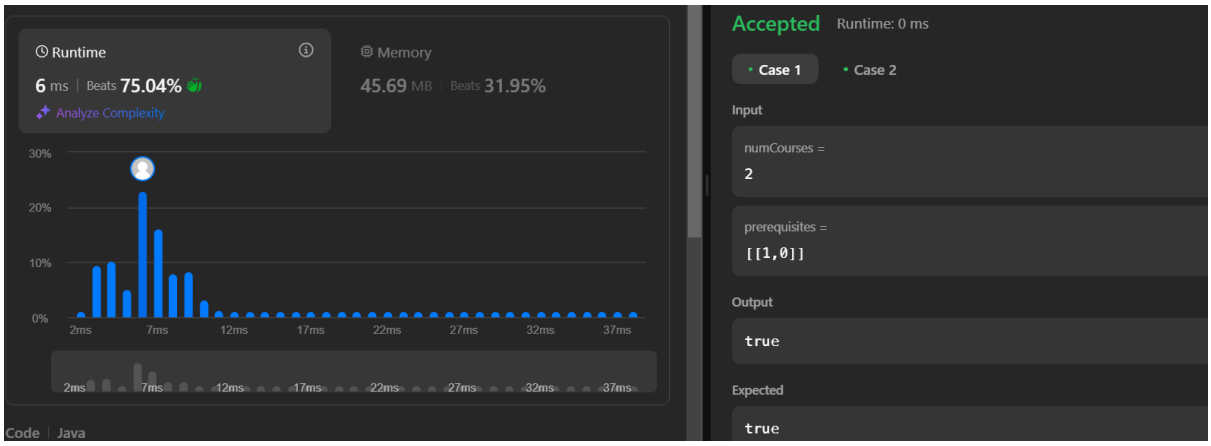
Code:

```
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:



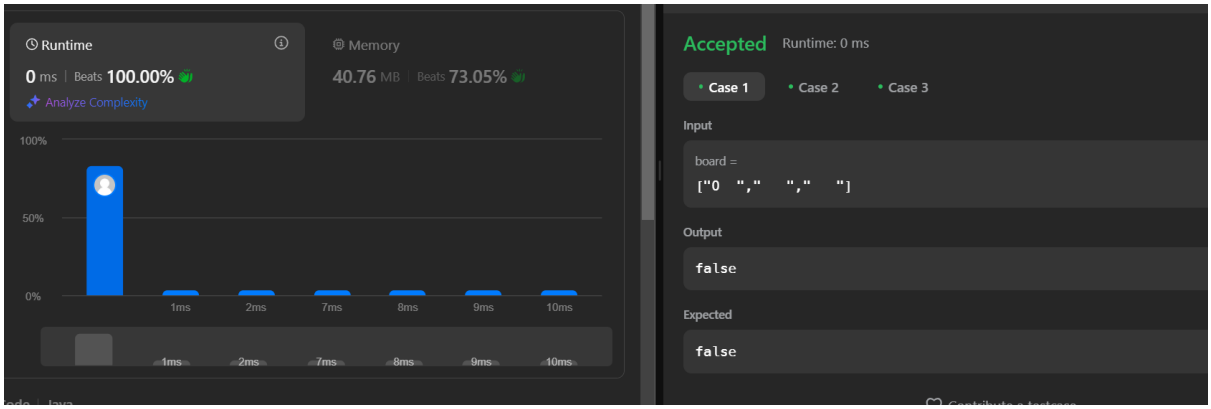
11. Design tic tac toe

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:


Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
40.76 MB | Beats 73.05%
100%
50%
0%
1ms
2ms
7ms
8ms
9ms
10ms
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
board =
["0 "," "," "]
Output
false
Expected
false