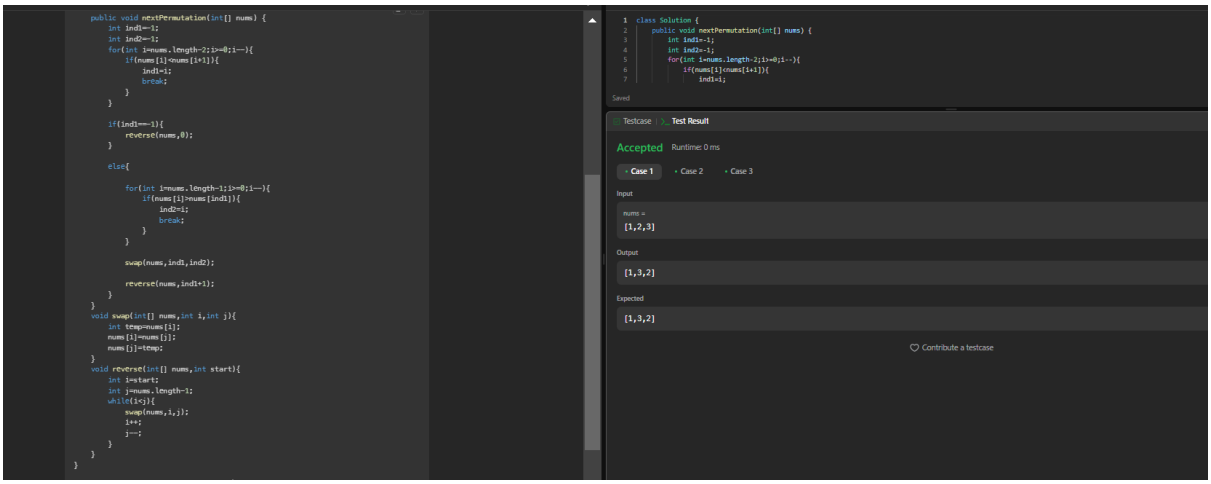
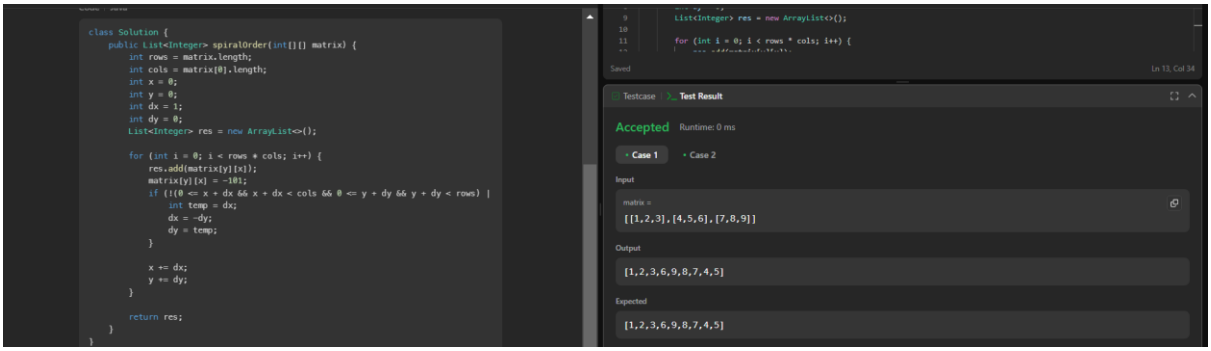
19/11/2024

DSA PRACTICE DAY 7

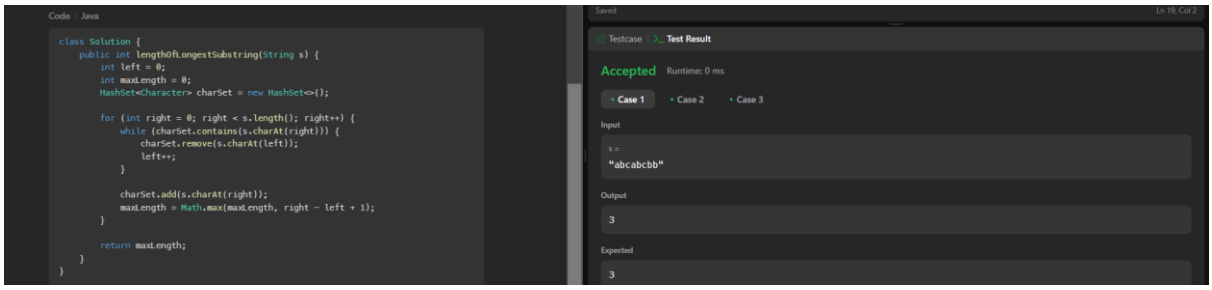
1. Next Permutation:



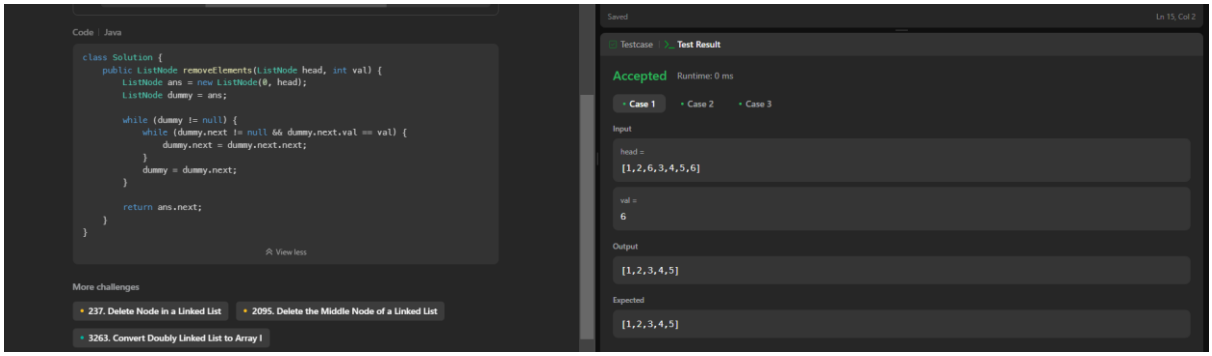
2. Spiral Matrix



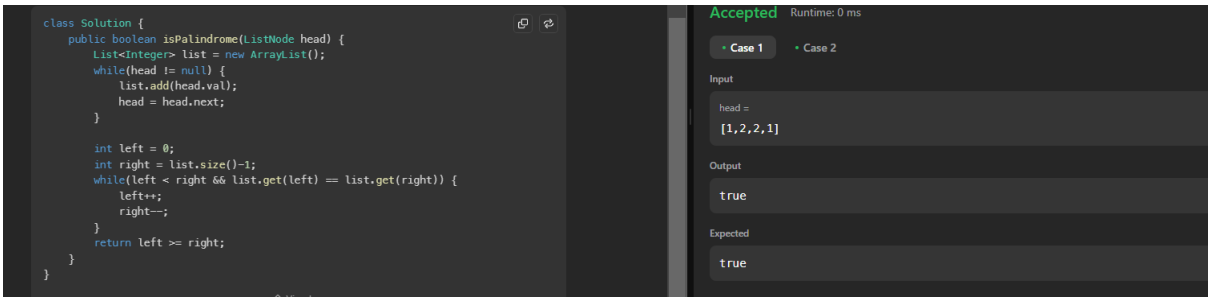
3. Longest substring without repeating characters



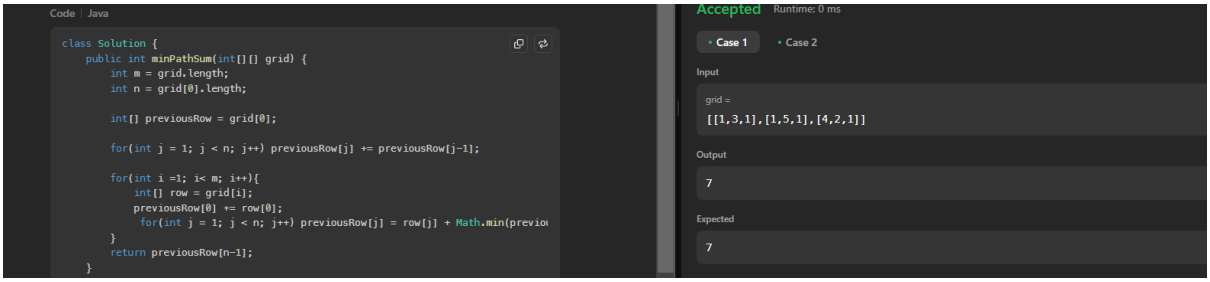
4. Remove linked list elements



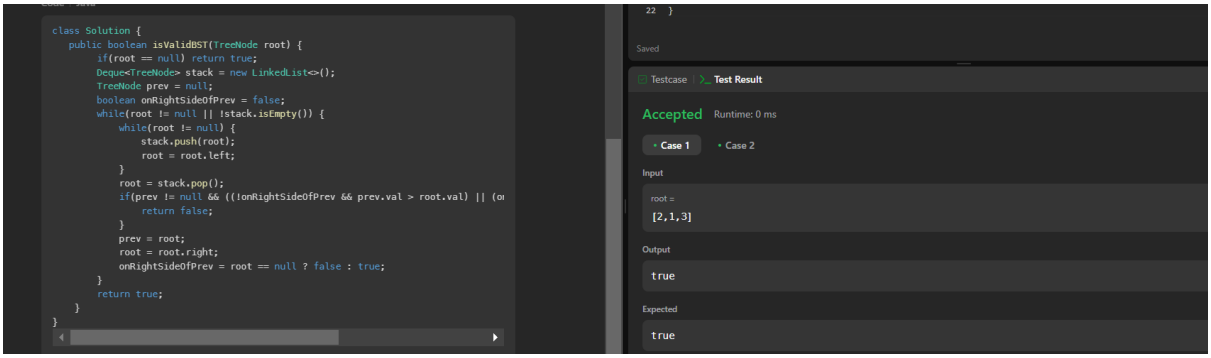
5. Palindrome linked list



6. Minimum path sum



7. Validate binary search tree



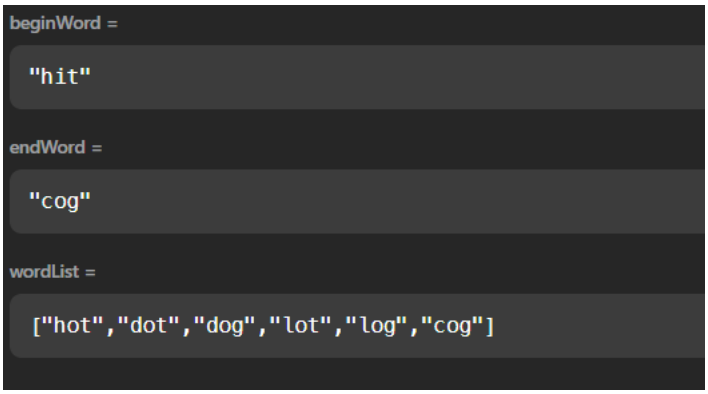
8. Word ladder

Program:

```
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

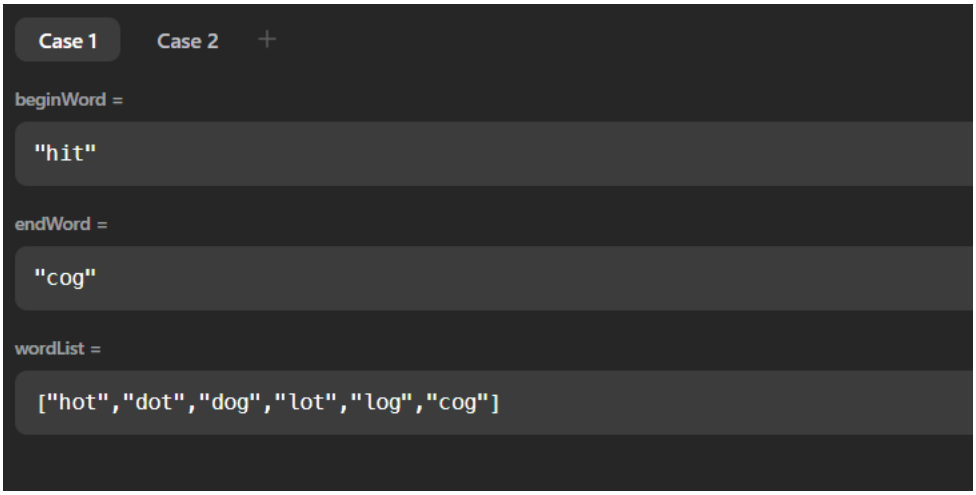
Program:

```
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:



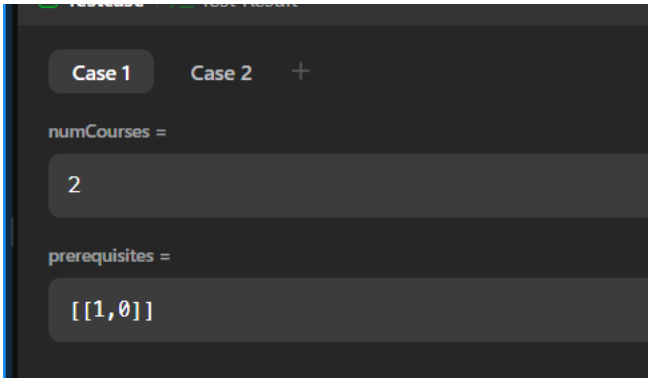
10. Course schedule

Program:

```
import java.util.ArrayList;
import java.util.List;

class Solution {
    private List<Integer> orderList;

    public int[] findOrder(int numCourses, int[][] prerequisites) {
        orderList = new ArrayList<>();
        if(canFinish(numCourses, prerequisites)) {
            return orderList.stream().mapToInt(i -> i).toArray();
        } else {
            return new int[]{};
        }
    }

    public boolean canFinish(int numCourses, int[][] prerequisites) {
        ArrayList<Integer>[] adj = new ArrayList[numCourses];

        for (int i = 0; i < numCourses; i++) {
            adj[i] = new ArrayList<>();
        }

        for (int[] pre : prerequisites) {
            adj[pre[0]].add(pre[1]);
        }

        int[] visited = new int[numCourses];

        for (int i = 0; i < numCourses; i++) {
            if (!dfs(i, visited, adj)) {
                return false;
            }
        }
```

```
        return true;
    }

    public boolean dfs(int node, int[] visited, ArrayList<Integer>[] adj) {
        if (visited[node] == 1) {
            return false;
        }

        if (visited[node] == 2) {
            return true;
        }

        visited[node] = 1;

        for (int n : adj[node]) {
            if (!dfs(n, visited, adj)) {
                return false;
            }
        }

        visited[node] = 2;
        orderList.add(node);

        return true;
    }
}
```

Output:



11. Design tic tac toe:

Program:

```
class Solution {
    public boolean validTicTacToe(String[] board) {
        int[] arr=new int[2];
        boolean xwin=false,owin=false;
        int xdiag=0,odiag=0;
        for(int i=0;i<3;i++){
            int x=0,o=0;
            for(int j=0;j<3;j++)
            {
                if(i==0){
                    if(board[i].charAt(j)=='X' && board[i+1].charAt(j)=='X' &&
board[i+2].charAt(j)=='X')xwin=true;
                    if(board[i].charAt(j)=='O' && board[i+1].charAt(j)=='O' &&
board[i+2].charAt(j)=='O')owin=true;
                }
                if(board[i].charAt(j)=='X'){
                    if(i==j)xdiag++;
                    arr[1]++;
                    x++;
                }
                else if(board[i].charAt(j)=='O'){
                    if(i==j)odiag++;
                    arr[0]++;
                    o++;
                }
```

```
            }
            if(o==3 && owin)return false;
            if(owin && xwin)return false;
            if(x==3)xwin=true;
            else if(o==3)owin=true;
        }

        if(xdiag==3)xwin=true;
        if(odiag==3)owin=true;

        if(board[0].charAt(2)=='X' && board[1].charAt(1)=='X' && board[2].charAt(0)=='X')xwin=true;
        if(board[0].charAt(2)=='O' && board[1].charAt(1)=='O' && board[2].charAt(0)=='O')owin=true;

        if(arr[0]>=arr[1] && xwin || (arr[1]>arr[0] && owin))return false;
        if(arr[0]>arr[1] || Math.abs(arr[0]-arr[1])>1)return false;

        if(xwin&&owin) return false;
        else return true;
    }
}
```

Output: