

Question 1 leetcode:-

Valid sudoku:

```
class Solution {
    int grid_value=9;
    public boolean isValidSudoku(char[][] board) {
        for(int i=0;i<grid_value;i++){
            for(int j=0;j<grid_value;j++){
                if(board[i][j]!='.'){
                    if(!isValid(board,i,j,board[i][j])){
                        return false;
                    }
                }
            }
        }
        return true;
    }

    public boolean isValid(char[][] board,int row,int col,char valchar){
        board[row][col]='.';
        for(int i=0;i<grid_value;i++){
            if(board[row][i]==valchar) return false;

            if(board[i][col]==valchar) return false;

            int valrow=row-row%3;
            int valcol=col-col%3;
            for(int j=valrow;j<valrow+3;j++){
                for(int k=valcol;k<valcol+3;k++){
                    if(board[j][k]==valchar){
                        return false;
                    }
                }
            }
        }
        board[row][col]=valchar;
        return true;
    }
}
```

Description
Editorial
Solutions (6.2K)
Submissions

Accepted

Next question

62. Unique Paths

More challenges

2133. Check if Every Row and Column Contains All Numbers

All statuses

All languages

Accepted

a few seconds ago

Java

i Java

Auto

12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37

}
return true;
}
public boolean isValid(char[][] board,int row,int col,char valchar){
board[row][col]='.';
for(int i=0;i<grid\_value;i++){
if(board[row][i]==valchar) return false;
if(board[i][col]==valchar) return false;
int valrow=row-row%3;
int valcol=col-col%3;
for(int j=valrow;j<valrow+3;j++){
for(int k=valcol;k<valcol+3;k++){
if(board[j][k]==valchar){
return false;
}
}
}
}
board[row][col]=valchar;
return true;
}
}

Question 2:

```
import java.util.List;
import java.util.ArrayList;
class Solution {
    public static List<String> generateParenthesis(int n) {
        List<String> result = new ArrayList<>();
        backtrack(result, "", 0, 0, n);
        return result;
    }

    private static void backtrack(List<String> result, String current, int open,
int close, int max) {
        if (current.length() == max * 2) {
            result.add(current);
            return;
        }

        if (open < max) {
            backtrack(result, current + "(", open + 1, close, max);
        }
        if (close < open) {
            backtrack(result, current + ")", open, close + 1, max);
        }
    }
}

public class Parentheses{
    public static void main(String[] args) {

    }
}
```

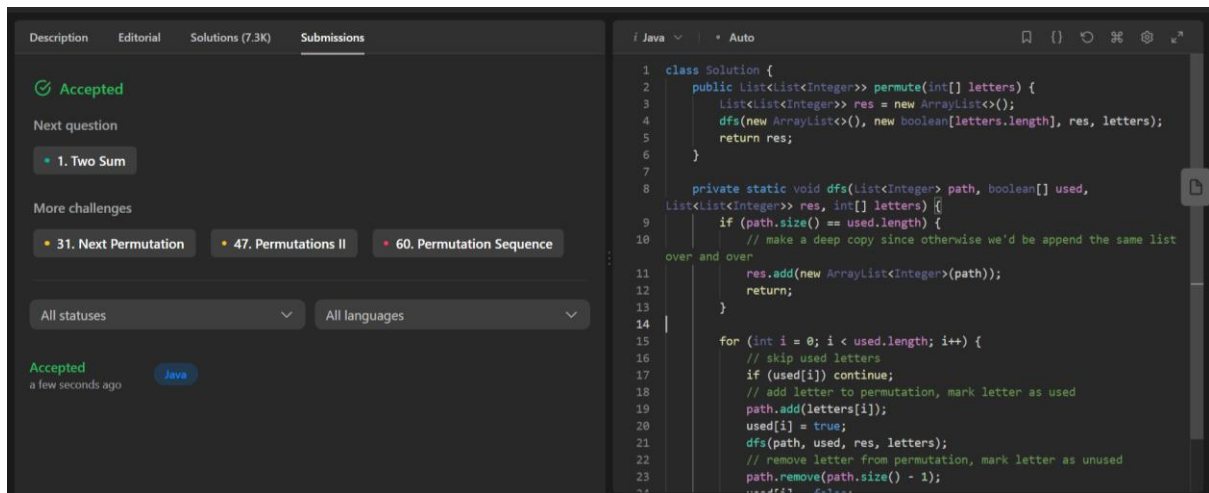
```
4 public static List<String> generateParenthesis(int n) {
5     List<String> result = new ArrayList<>();
6     backtrack(result, "", 0, 0, n);
7     return result;
8 }
9
10 private static void backtrack(List<String> result, String current, int
open, int close, int max) {
11     if (current.length() == max * 2) {
12         result.add(current);
13         return;
14     }
15
16     if (open < max) {
17         backtrack(result, current + "(", open + 1, close, max);
18     }
19     if (close < open) {
20         backtrack(result, current + ")", open, close + 1, max);
21     }
22 }
23
24 public class Parentheses{
25     public static void main(String[] args) {
26
27     }
28 }
```

Question 3:

```
class Solution {
    public List<List<Integer>> permute(int[] letters) {
        List<List<Integer>> res = new ArrayList<>();
        dfs(new ArrayList<>(), new boolean[letters.length], res, letters);
        return res;
    }

    private static void dfs(List<Integer> path, boolean[] used,
List<List<Integer>> res, int[] letters) {
        if (path.size() == used.length) {
            // make a deep copy since otherwise we'd be append the same list over
            // and over
            res.add(new ArrayList<Integer>(path));
            return;
        }

        for (int i = 0; i < used.length; i++) {
            // skip used letters
            if (used[i]) continue;
            // add letter to permutation, mark letter as used
            path.add(letters[i]);
            used[i] = true;
            dfs(path, used, res, letters);
            // remove letter from permutation, mark letter as unused
            path.remove(path.size() - 1);
            used[i] = false;
        }
    }
}
```



Question 4:

```

public class RatInAMaze {

    public static void main(String[] args) {

        int[][] maze = {

            {1, 0, 0, 0},

            {1, 1, 0, 1},

            {0, 1, 0, 0},

            {1, 1, 1, 1}

        };

        solveMaze(maze);

    }

    public static void solveMaze(int[][] maze) {

        int rows = maze.length;

        int columns = maze[0].length;

        int[][] solution = new int[rows][columns];

        if (solveMazeUtil(maze, 0, 0, solution)) {

            printSolution(solution);

        } else {

            System.out.println("No solution exists.");
        }
    }
}

```

```
}  
}
```

```
private static boolean solveMazeUtil(int[][] maze, int row, int col, int[][] solution) {  
    int rows = maze.length;  
    int columns = maze[0].length;  
  
    if (row >= 0 && row < rows && col >= 0 && col < columns && maze[row][col] == 1) {  
        solution[row][col] = 1;  
        if (row == rows - 1 && col == columns - 1) {  
            return true;  
        }  
        if (solveMazeUtil(maze, row, col + 1, solution)) {  
            return true;  
        }  
        if (solveMazeUtil(maze, row + 1, col, solution)) {  
            return true;  
        }  
        if (solveMazeUtil(maze, row - 1, col, solution)) {  
            return true;  
        }  
        if (solveMazeUtil(maze, row, col - 1, solution)) {  
            return true;  
        }  
        solution[row][col] = 0;  
        return false;  
    }  
  
    return false;  
}
```

```

private static void printSolution(int[][] solution) {

    int rows = solution.length;

    int columns = solution[0].length;

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            System.out.print(solution[i][j] + " ");
        }
        System.out.println();
    }
}
}

```

Output

Clear

```

java -cp /tmp/JvtuP4WoDM RatInAMaze
1 0 0 0
1 1 0 0
0 1 0 0
0 1 1 1
|

```

Question 5:

```

class Solution {
    public boolean exist(char[][] board, String word) {
        boolean result=false;
        int m=board.length;
        int n=board[0].length;
        for(int i=0;i<m;i++){
            for(int j=0;j<n;j++){
                if(search(board,word,i,j,0)){
                    return true;
                }
            }
        }
        return result;
    }
}

```

```

    }
    private boolean search(char [][] board,String word,int row,int col,int k){
        int m=board.length;
        int n=board[0].length;
        //if length exceed
        if(row<0 || col<0 || row>=m || col>=n ){
            return false;
        }
        if(board[row][col]==word.charAt(k)){
            char temp=board[row][col];
            board[row][col]='#';
            if(k==word.length()-1){
                return true;
            }

            else if(search(board,word,row-1,col,k+1) || search(board,word,row,col-1,k+1) || search(board,word,row+1,col,k+1) || search(board,word,row,col+1,k+1)){
                return true;
            }
            board[row][col]=temp;
        }
        return false;
    }
}

```

Description
Editorial
Solutions (4.6K)
Submissions

Accepted
Next question
76. Minimum Window Substring
More challenges
212. Word Search II
All statuses
All languages
Accepted a few seconds ago
Java

```

14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
}
private boolean search(char [][] board,String word,int row,int col,int k)
{
    int m=board.length;
    int n=board[0].length;
    //if length exceed
    if(row<0 || col<0 || row>=m || col>=n ){
        return false;
    }
    if(board[row][col]==word.charAt(k)){
        char temp=board[row][col];
        board[row][col]='#';
        if(k==word.length()-1){
            return true;
        }

        else if(search(board,word,row-1,col,k+1) || search(board,word,row,col-1,k+1) || search(board,word,row+1,col,k+1) || search(board,word,row,col+1,k+1)){
            return true;
        }
        board[row][col]=temp;
    }
    return false;
}

```