2/1/25

Java notes

1.finding common numbers in the given arrays

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        // Example arrays
        int[] arr1 = {1, 2, 3, 4, 5};
        int[] arr2 = {4, 5, 6, 7, 8};

        // Finding common numbers
        List<Integer> commonNumbers = findCommonNumbers(arr1, arr2);

        // Print the result
        System.out.println("Common numbers: " + commonNumbers);
    }

    public static List<Integer> findCommonNumbers(int[] arr1, int[] arr2) {
        // Use a HashSet for fast lookups
        Set<Integer> set1 = new HashSet<>();
        for (int num : arr1) {
            set1.add(num);
        }

        // Store common elements
```

```
        List<Integer> common = new ArrayList<>();
        for (int num : arr2) {
            if (set1.contains(num)) {
                common.add(num);
            }
        }
        return common;
    }
}
```

Output:

Common numbers: [4, 5]

2. add 2 numbers to get the target value from the given 2 set of arrays

1st-

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        // Example arrays and target
        int[] arr1 = {1, 2, 3, 4, 5};
        int[] arr2 = {6, 7, 8, 9, 10};
        int target = 10;

        // Find the pair
        List<int[]> result = findPairWithTargetSum(arr1, arr2, target);
```

```
        // Print the result
        if (result.isEmpty()) {
            System.out.println("No pairs found with the given target value.");
        } else {
            System.out.println("Pairs that sum up to " + target + ":");
            for (int[] pair : result) {
                System.out.println("(" + pair[0] + ", " + pair[1] + ")");
            }
        }
    }

    public static List<int[]> findPairWithTargetSum(int[] arr1, int[] arr2, int target) {
        // Use a HashSet to store elements of the first array
        Set<Integer> set1 = new HashSet<>();
        for (int num : arr1) {
            set1.add(num);
        }

        // List to store pairs
        List<int[]> pairs = new ArrayList<>();

        // Check if the complement of each element in arr2 exists in set1
        for (int num : arr2) {
            int complement = target - num;
```

```
            if (set1.contains(complement)) {
                pairs.add(new int[]{complement, num});
            }
        }
        return pairs;
    }
}
```

Output:

Pairs that sum up to 10:

(4, 6)

(3, 7)

(2, 8)

(1, 9)

2nd-

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        // Example arrays and target
        int[] arr1 = {1, 2, 3, 4, 5};
        int[] arr2 = {6, 7, 8, 9, 10};
        int target = 10;
        // Find the pairs
        List<int[]> result = findPairWithTargetSum(arr1, arr2, target);
```

```
        // Print the result
        if (result.isEmpty()) {
            System.out.println("No pairs found with the given target value.");
        } else {
            System.out.println("Pairs that sum up to " + target + ":");
            for (int[] pair : result) {
                System.out.println("(" + pair[0] + ", " + pair[1] + ")");
            }
        }
    }

    public static List<int[]> findPairWithTargetSum(int[] arr1, int[] arr2, int target) {
        // Use a HashSet to store elements of the first array
        Set<Integer> set1 = new HashSet<>();
        List<int[]> pairs = new ArrayList<>();

        // Using a single for loop to iterate through arr1
        for (int num : arr1) {
            set1.add(num);

            // Simultaneously, use a while loop to iterate through arr2
            int i = 0; // Index for arr2
            while (i < arr2.length) {
```

```
            int complement = target - arr2[i];
            if (set1.contains(complement)) {
                pairs.add(new int[]{complement, arr2[i]});
            }
            i++; // Increment index for the while loop
        }
    }

    return pairs;
  }
}
```

output:

Pairs that sum up to 10:

(1, 9)

(2, 8)

(1, 9)

(3, 7)

(2, 8)

(1, 9)

(4, 6)

(3, 7)

(2, 8)

(1, 9)

(4, 6)

(3, 7)

(2, 8)

(1, 9)

Leetcode

63-Unique Paths II:

```
class Solution {
    public int uniquePathsWithObstacles(int[][] obstacleGrid) {
        int rows = obstacleGrid.length, cols = obstacleGrid[0].length;
        int[][] dp = new int[rows][cols];
        if (obstacleGrid[0][0] == 1 || obstacleGrid[rows - 1][cols - 1] ==
1) {
            return 0;
        }
        dp[0][0] = 1;
        for (int i = 1; i < rows; i++) {
            dp[i][0] = (obstacleGrid[i][0] == 0 && dp[i - 1][0] == 1) ? 1
: 0;
        }
        for (int j = 1; j < cols; j++) {
            dp[0][j] = (obstacleGrid[0][j] == 0 && dp[0][j - 1] == 1) ? 1
: 0;
        }
        for (int i = 1; i < rows; i++) {
            for (int j = 1; j < cols; j++) {
                if (obstacleGrid[i][j] == 0) {
                    dp[i][j] += dp[i - 1][j] + dp[i][j - 1];
                }
                else {
                    dp[i][j] = 0;
                }
            }
        }
        return dp[rows - 1][cols - 1];
    }
}
```

64-Minimum Path Sum:

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

    }
}
```

139. Word Break:

```java
class Solution {
    public boolean wordBreak(String s, List<String> wordDict) {
        int n=s.length();
        boolean[] result=new boolean[n+1];
        result [0]=true;
        int maxLength=0;
        for(String word:wordDict){
            maxLength=Math.max(maxLength,word.length());
        }
        for(int i=1;i<=n;i++){
            for(int j=i-1;j>=Math.max(i-maxLength-1,0);j--){
                if(result[j] && wordDict.contains(s.substring(j,i))){
                    result[i]=true;
                    break;
                }
            }
        }
        return result[n];
    }
}
```

72. Edit Distance:

```
class Solution {
    public int minDistance(String word1, String word2) {
        int m=word1.length();
        int n=word2.length();
        int [][] dp= new int[m+1][n+1];
        for(int i=0;i<=m;i++){
            dp[i][0] =i;
        }
        for(int j=0;j<=n;j++){
            dp[0][j]=j;
        }
        for(int i=1;i<=m;i++){
            for(int j=1;j<=n;j++){
                if(word1.charAt(i-1)==word2.charAt(j-1)){
                    dp[i][j]=dp[i-1][j-1];
                }else{
                    dp[i][j]=Math.min(dp[i-1][j],Math.min(dp[i][j-1],dp[i-
1][j-1]))+1;
                }
            }
        }
        return dp[m][n];
    }

}
```

94. Binary Tree Inorder Traversal:

1st-using recursion:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
```

```
 */
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        recursion(root,result);
        return result;
        }
        public void recursion(TreeNode node,List<Integer> result){
            if(node ==null){
                return ;
            }
            recursion(node.left,result);
            result.add(node.val);
            recursion(node.right,result);
        }

}
```

2nd- using stack:

```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
         List<Integer> result = new ArrayList<>();
        Stack<TreeNode> stack = new Stack<>();

        TreeNode current = root;

        while (current != null || !stack.isEmpty()) {
            while (current != null) {
                stack.push(current);
                current = current.left;
            }

            current = stack.pop();
            result.add(current.val);
            current = current.right;
        }

        return result;  }
}
```

144. Binary Tree Preorder Traversal

```java
class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> res=new ArrayList<>();
        recur(root,res);
        return res;

    }
    public void recur(TreeNode node,List<Integer> res)
    {
        if(node==null)
        {
            return ;
        }
        res.add(node.val);
        recur(node.left,res);
        recur(node.right,res);
    }
}
```