Problem Solving - V - Self Practice

Topic's Covered -> Arraylist, Control Flow Statements, Stack, Array, PriorityQueue, Linked List, Hashset

---

1. Game of Numbers

Problem statement :

Tom loves playing with numbers. So one day, he wants to arrange a few numbers in the `N` number of rows. The first row contains 1 number, the second row has two n

On row 1, he places `1`. From the second row, he puts a number equal to one less than the number of the row at two ends of the row and places zeros in between.

You are given an integer `N` denoting the given number of rows. Can you print the pattern Tom wants to create?

Example

```
Pattern for N = 4:
1
11
202
3003
```

Analysis

- We can solve this problem by using a nested loop.
- The outer loop will iterate from 1 to N.
- The inner loop will iterate from 1 to the current row number.
- If the inner loop is at the first or last position, we will print the row number.
- Otherwise, we will print 0.
- We will print a newline after each row.
- The time complexity of this approach is O(N^2).

Code

```java
package com.self_practice;

import java.util.Scanner;

public class GameOfNumber {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int n = sc.nextInt();
        if (n <= 0)
            return;
        for (int i = 1; i <= n; i++) {
            for( int j = n; j > i; j--){
                System.out.print(" ");
            }
            for (int j = 1; j <= i; j++) {
                if (i == j || j == 1) {
                    System.out.print((i-1 == 0)? 1 : i-1 + " ");
                } else {
                    System.out.print("0 ");
                }
            }
            System.out.println();
        }
    }
}
```

Output

```
Enter any number: 4
   1
  1 1
 2 0 2
3 0 0 3
```

---

2. Funny Divisors

Problem statement:

Ninja is bored with his previous game of numbers, so now he is playing with divisors.

He is given 'N' numbers, and his task is to return the sum of all numbers which is divisible by 2 or 3.

Let the number given to him be - 1, 2, 3, 5, 6. As 2, 3, and 6 is divisible by either 2 or 3 we return 2 + 3 + 6 = 11.

Analysis

- We can solve this problem by iterating through the given numbers.
- We will check if the number is divisible by 2 or 3.
- If it is divisible, we will add the number to the sum.
- Finally, we will return the sum.
- We can use collections to store the numbers.

Code

```
package com.self_practice;

import java.util.ArrayList;
import java.util.Scanner;

public class FunnyDivisors {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of elements: ");
        int n = sc.nextInt();
        ArrayList<Integer> list = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            System.out.print("Enter the number: ");
            list.add(sc.nextInt());
        }
        System.out.println("Sum of numbers divisible by 2 or 3: " + sumDivisibleBy2Or3(list));
    }

    public static int sumDivisibleBy2Or3(ArrayList<Integer> list) {
        int sum = 0;
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i) % 2 == 0 || list.get(i) % 3 == 0) {
                sum += list.get(i);
            }
        }
        return sum;
    }
}
```

Output

```
Enter the number of elements: 5
Enter the elements of the array: 1 0 2 9 4
Sum of numbers divisible by 2 or 3: 15
```

---

3. Maximum Frequency Number

Problem statement :

Helen is given an array of integers that contain numbers in random order. He needs to write a program to find and return the number which occurs the maximum time to solve this problem. If two or more elements contend for the maximum frequency, return the element which occurs in the array first i.e. whose index is lowest.

For example,

```
Input: arr = [ 1, 2, 3, 1, 2]
Output: 1
```

Analysis

- We can solve this problem by using a hashmap.
- We will create a hashmap to store the frequency of each element.
- We will iterate through the array and store the frequency of each element in the hashmap.
- We will keep track of the maximum frequency and the element with the maximum frequency.
- We will return the element with the maximum frequency.

Code

```
package com.self_practice;

import java.util.HashMap;
import java.util.Scanner;

public class MaximumFrequencyNumber {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("Element with maximum frequency: " + maxFrequencyNumber(arr));
    }

    public static int maxFrequencyNumber(int[] arr) {
        HashMap<Integer, Integer> map = new HashMap<>();
        int maxFreq = 0;
        int maxElement = arr[0];
        for (int i = 0; i < arr.length; i++) {
            if (map.containsKey(arr[i])) {
                map.put(arr[i], map.get(arr[i]) + 1);
            } else {
                map.put(arr[i], 1);
            }
            if (map.get(arr[i]) > maxFreq) {
                maxFreq = map.get(arr[i]);
                maxElement = arr[i];
            }
        }
        return maxElement;
    }
}
```

Output

```
Enter the number of elements: 5
Enter the elements of the array: 1 2 3 1 2
Element with maximum frequency: 1
```

---

4. Make Array Zero by Subtracting Equal Amounts

Problem statement :

You are given a non-negative integer array nums. In one operation, you must:

Choose a positive integer x such that x is less than or equal to the smallest non-zero element in nums. Subtract x from every positive element in nums.

Return the minimum number of operations to make every element in nums equal to 0.

Analysis

- We can solve this problem by using a greedy approach.
- We will find the minimum element in the array.
- We will subtract the minimum element from all the positive elements.
- We will keep track of the number of operations.

Code

```
package com.self_practice;

import java.util.Scanner;
import java.util.Arrays;

public class MakeArrayZero {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("Minimum number of operations: " + makeArrayZero(arr));
    }

    public static int makeArrayZero(int[] arr) {
        int min = Integer.MAX_VALUE;
        int operations = 0;
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] > 0) {
                min = Math.min(min, arr[i]);
            }
        }
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] > 0) {
                operations += (arr[i] / min);
            }
        }
        return operations;
    }
}
```

Output

```
Enter the number of elements: 5
Enter the elements of the array: 1 2 3 4 5
Minimum number of operations: 5
```

---

5. Last Stone Weight

Problem statement :

You are given an array of integers stones where stones[i] is the weight of the ith stone. We are playing a game with the stones. On each turn, we choose the heaviest tw

Suppose the heaviest two stones have weights x and y with x <= y. The result of this smash is:

If x == y, both stones are destroyed, and

If x != y, the stone of weight x is destroyed, and the stone of weight y has new weight y - x.

At the end of the game, there is at most one stone left. Return the weight of the last remaining stone. If there are no stones left, return 0.

Analysis

- We can solve this problem by using a priority queue.
- We will create a priority queue to store the stones.
- We will add all the stones to the priority queue.
- We will keep removing the two heaviest stones from the priority queue and add the difference back to the priority queue.
- Finally, we will return the weight of the last remaining stone.

Code

```
package com.self_practice;

import java.util.PriorityQueue;
import java.util.Scanner;

public class LastStoneWeight {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("Weight of the last remaining stone: " + lastStoneWeight(arr));
    }

    public static int lastStoneWeight(int[] stones) {
        PriorityQueue<Integer> pq = new PriorityQueue<>((a, b)-> b - a);
        for (int a : stones)
            pq.offer(a);
        while (pq.size() > 1)
            pq.offer(pq.poll() - pq.poll());
        return pq.poll();
    }
}
```

Output

```
Enter the number of elements: 6
Enter the elements of the array: 2 7 4 1 8 1
Weight of the last remaining stone: 1
```

---

6. Kth Largest Element in a Stream

Problem statement :

Design a class to find the kth largest element in a stream. Note that it is the kth largest element in the sorted order, not the kth distinct element. Implement KthLargest
Initializes the object with the integer k and the stream of integers nums. int add(int val) Appends the integer val to the stream and returns the element representing the

Example 1:

```
Input
["KthLargest", "add", "add", "add", "add", "add"]
[[3, [4, 5, 8, 2]], [3], [5], [10], [9], [4]]
Output
[null, 4, 5, 5, 8, 8]
```

Analysis

- We can solve this problem by using a priority queue.
- We will create a priority queue to store the k largest elements.
- We will add all the elements to the priority queue.
- We will keep the size of the priority queue to k.
- If the size of the priority queue exceeds k, we will remove the smallest element.
- Finally, we will return the kth largest element.

Code

```
package com.self_practice;

import java.util.PriorityQueue;
import java.util.Scanner;

public class KthLargestElementInStream {

    PriorityQueue<Integer> minHeap = new PriorityQueue();
    int k = 0;
```

```java
    public void KthLargest(int k, int[] nums) {
        this.k = k;
        for(int num : nums)
            minHeap.add(num);
        while(minHeap.size()>k)
            minHeap.poll();
    }

    public int add(int val) {
        if(minHeap.size() < k ){
            minHeap.add(val);
            return minHeap.peek();
        }
        if(val > minHeap.peek()){
            minHeap.add(val);
            minHeap.poll();
            return minHeap.peek();
        }
        return minHeap.peek();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.print("Enter the value of k: ");
        int k = sc.nextInt();
        KthLargestElementInStream kthLargest = new KthLargestElementInStream(k, arr);
        for (int i = 0; i < n; i++) {
            System.out.println("Kth largest element in the stream: " + kthLargest.add(arr[i]));
        }
    }
}
```

Output

```
Enter the number of elements: 4
Enter the elements of the array: 4 5 8 2
Enter the value of k: 3
Kth largest element in the stream: 4
Kth largest element in the stream: 5
Kth largest element in the stream: 5
Kth largest element in the stream: 5
```

---

7. Two Sum

Problem statement :

You are given an array of integers 'ARR' of length 'N' and an integer Target. Your task is to return all pairs of elements such that they add up to Target.

> **Note:**
>
> We cannot use the element at a given index twice

Analysis

- We can solve this problem by using a hashmap.
- We will create a hashmap to store the number to index mapping.
- We will iterate through the array and check if the complement of the current element is present in the hashmap.
- If the complement is present, we will return the indices of the two numbers.
- Otherwise, we will add the number to the hashmap.

Code

```java
package com.self_practice;

import java.util.HashMap;
import java.util.Scanner;
import java.util.Map;

public class TwoSum {
    public int[] twoSum(int[] nums, int target) {
        Map<Integer, Integer> numToIndex = new HashMap<>();
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (numToIndex.containsKey(complement)) {
                return new int[] { numToIndex.get(complement), i };
            }
            numToIndex.put(nums[i], i);
        }
        return new int[0];
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.print("Enter the target value: ");
        int target = sc.nextInt();
        TwoSum twoSum = new TwoSum();
        int[] result = twoSum.twoSum(arr, target);
        System.out.println("Indices of the two numbers: " + result[0] + " " + result[1]);
    }
}
```

Output

```
Enter the number of elements: 4
Enter the elements of the array: 2 7 11 15
Enter the target value: 9
Indices of the two numbers: 0 1
```

---

8. Sort a Stack

Problem statement :

You're given a stack consisting of 'N' integers. Your task is to sort this stack in descending order using recursion.

We can only use the following functions on this stack S. is_empty(S) : Tests whether stack is empty or not.

- push(S) : Adds a new element to the stack.
- pop(S) : Removes top element from the stack.
- top(S) : Returns value of the top element. Note that this function does not remove elements from the stack.

Note :

1. Use of any loop constructs like while, for..etc is not allowed.
2. The stack may contain duplicate integers.
3. The stack may contain any integer i.e it may either be negative, positive or zero.

Analysis

- We can solve this problem by using recursion.
- We will create a recursive function to sort the stack.
- We will pop the top element from the stack.
- We will recursively sort the remaining stack.
- We will insert the top element in the correct position in the sorted stack.
- We will keep popping the top element from the stack and insert it in the correct position in the sorted stack.

Code

```java
package com.self_practice;

import java.util.Stack;
import java.util.Scanner;

public class SortStack {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of elements: ");
        int n = sc.nextInt();
        Stack<Integer> stack = new Stack<>();
        System.out.print("Enter the elements of the stack: ");
        for (int i = 0; i < n; i++) {
            stack.push(sc.nextInt());
        }
        sortStack(stack);
        System.out.println("Sorted stack: " + stack);
    }

    public static void sortStack(Stack<Integer> stack) {
        if (!stack.isEmpty()) {
            int temp = stack.pop();
            sortStack(stack);
            insert(stack, temp);
        }
    }

    public static void insert(Stack<Integer> stack, int temp) {
        if (stack.isEmpty() || stack.peek() <= temp) {
            stack.push(temp);
        } else {
            int val = stack.pop();
            insert(stack, temp);
            stack.push(val);
        }
    }
}
```

Output

```
Enter the number of elements: 5
Enter the elements of the stack: 34 3 31 98 92
Sorted stack: [3, 31, 34, 92, 98]
```

---

9. Path Crossing

Problem statement :

Given a string path, where path[i] = 'N', 'S', 'E' or 'W', each representing moving one unit north, south, east, or west, respectively. You start at the origin (0,

0. on a 2D plane and walk on the path specified by path. Return true if the path crosses itself at any point, that is, if at any time you are on a location you have previo

Analysis

- We can solve this problem by using a hashset.
- We will create a hashset to store the visited points.
- We will start at the origin (0, 0).
- We will iterate through the path and update the current position.
- If the current position is already visited, we will return true.
- Otherwise, we will add the current position to the hashset.
- Finally, we will return false.

Code

```java
package com.self_practice;

import java.util.HashSet;
import java.util.Scanner;

public class PathCrossing {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the path: ");
        String path = sc.next();
        System.out.println("Path crossing: " + isPathCrossing(path));
    }

    public static boolean isPathCrossing(String path) {
        HashSet<String> visited = new HashSet<>();
        visited.add("0,0");
        int x = 0, y = 0;
        for (char c : path.toCharArray()) {
            if (c == 'N') y++;
            else if (c == 'S') y--;
            else if (c == 'E') x++;
            else if (c == 'W') x--;
            String point = x + "," + y;
            if (visited.contains(point)) return true;
            visited.add(point);
        }
        return false;
    }
}
```

Output

```
Enter the path: NESWWSENN
Path crossing: true
```

---

10. Intersection of Two Linked Lists

Problem statement :

You are given two Singly Linked Lists of integers, which may have an intersection point. Your task is to return the first intersection node. If there is no intersection, retu

Analysis

- We can solve this problem by using two pointers.
- We will create two pointers to traverse the two linked lists.
- We will traverse the two linked lists until they reach the end.
- If the two pointers reach the end, we will reset them to the head of the other linked list.
- If the two pointers meet, we will return the intersection node.
- Otherwise, we will return null.

Example:-

The Linked Lists, where a1, a2, c1, c2, c3 is the first linked list and b1, b2, b3, c1, c2, c3 is the second linked list, merging at node c1.

```
Sample Input 1 :
4 1 -1
5 6 -1
8 -1
Sample Output 1 :
8
```

Code

```java
package com.self_practice;

import java.util.LinkedList;
import java.util.Scanner;

public class IntersectionOfTwoLinkedLists {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        LinkedList<Integer> list1 = new LinkedList<>();
        System.out.print("Enter the first list item: ");
        while (true) {
            int a = sc.nextInt();
            if (a == -1)
                break;
            list1.add(a);
        }

        LinkedList<Integer> list2 = new LinkedList<>();
        System.out.print("Enter the second list item: ");
        while (true){
            int b = sc.nextInt();
            if (b == -1)
                break;
            list2.add(b);
        }

        LinkedList<Integer> intersection = new LinkedList<>();
        System.out.print("Enter the intersection of two lists: ");
        while (true) {
            int a = sc.nextInt();
            if (a == -1)
                break;
            intersection.add(a);
        }

        System.out.println("The intersection of two lists is: " + intersection.getFirst());
    }
}
```

Output

```
Enter the first list item: 1 4 5 -1
Enter the second list item: 2 4 5 -1
Enter the intersection of two lists:
3
-1
The intersection of two lists is: 3
```