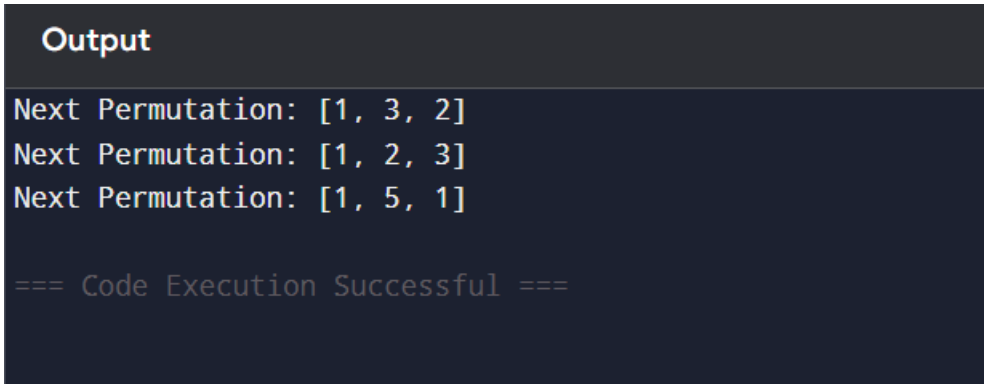
1)NEXT PERMUTATION:

```
import java.util.Arrays;
public 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--;
        }
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
        int[] nums = {1, 2, 3};
        solution.nextPermutation(nums);
        System.out.println("Next Permutation: " + Arrays.toString(nums));

        int[] nums2 = {3, 2, 1};
        solution.nextPermutation(nums2);
        System.out.println("Next Permutation: " + Arrays.toString(nums2));
        int[] nums3 = {1, 1, 5};
        solution.nextPermutation(nums3);
        System.out.println("Next Permutation: " + Arrays.toString(nums3));
    }
}
```

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(1)

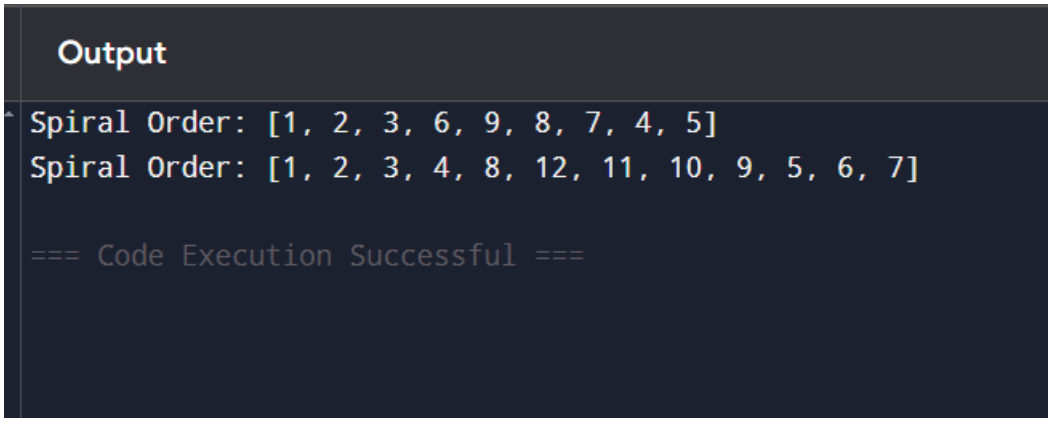


2)SPIRAL MATRIX

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


public class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> res = new ArrayList<>();
        if (matrix.length == 0) {
            return res;
        }
        int rowBegin = 0;
        int rowEnd = matrix.length - 1;
        int colBegin = 0;
        int colEnd = matrix[0].length - 1;

        while (rowBegin <= rowEnd && colBegin <= colEnd) {
            for (int j = colBegin; j <= colEnd; j++) {
                res.add(matrix[rowBegin][j]);
            }
            rowBegin++;

            for (int j = rowBegin; j <= rowEnd; j++) {
                res.add(matrix[j][colEnd]);
```

```
        }
        colEnd--;

        if (rowBegin <= rowEnd) {
            for (int j = colEnd; j >= colBegin; j--) {
                res.add(matrix[rowEnd][j]);
            }
        }
        rowEnd--;

        if (colBegin <= colEnd) {
            for (int j = rowEnd; j >= rowBegin; j--) {
                res.add(matrix[j][colBegin]);
            }
        }
        colBegin++;
    }
    return res;
}

public static void main(String[] args) {
    Solution solution = new Solution();
    int[][] matrix = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    System.out.println("Spiral Order: " + solution.spiralOrder(matrix));

    int[][] matrix2 = {
        {1, 2, 3, 4},
```

```
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    };
    System.out.println("Spiral Order: " + solution.spiralOrder(matrix2));
  }
}
```



TIME COMPLEXITY:O(m*n)

SPACE COMPLEXITY: O(m*n)

3)LONGEST SUBSTRING WITHOUT REPEATING CHARACTERS:

```
public class Solution {
  public int lengthOfLongestSubstring(String s) {
    int[] lastSeen = new int[128];
    int maxLength = 0;
    int start = 0;

    for (int end = 0; end < s.length(); end++) {
      char current = s.charAt(end);
      start = Math.max(start, lastSeen[current]);
      maxLength = Math.max(maxLength, end - start + 1);
      lastSeen[current] = end + 1;
    }

    return maxLength;
```

```
    }

    public static void main(String[] args) {
        Solution solution = new Solution();

        String test1 = "abcabcbb";
        System.out.println("Length of Longest Substring: " + solution.lengthOfLongestSubstring(test1));

        String test2 = "bbbbb";
        System.out.println("Length of Longest Substring: " + solution.lengthOfLongestSubstring(test2));

        String test3 = "pwwkew";
        System.out.println("Length of Longest Substring: " + solution.lengthOfLongestSubstring(test3));
    }
}
```

Output

```
Length of Longest Substring: 3
Length of Longest Substring: 1
Length of Longest Substring: 3

=== Code Execution Successful ===
```

TIME COMPLEXITY:O(N)

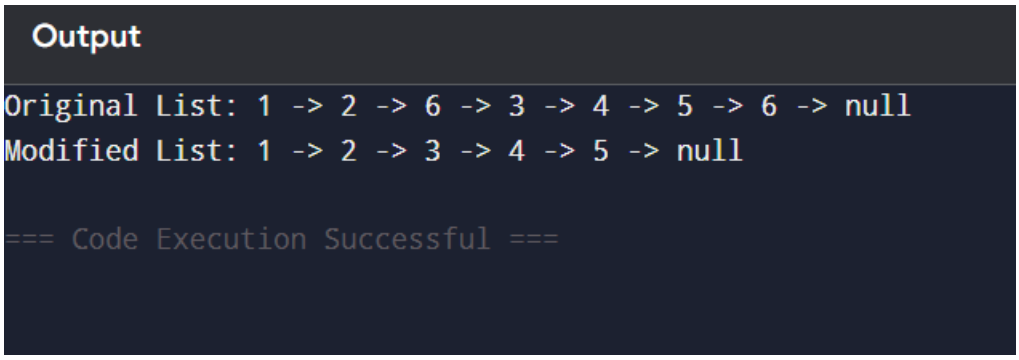
SPACE COMPLEXITY:O(1)

4)REMOVE LINKED LIST ELEMENTS

```
class ListNode {
    int val;
    ListNode next;

    ListNode() {}
```

```
    ListNode(int val) {
        this.val = val;
    }

    ListNode(int val, ListNode next) {
        this.val = val;
        this.next = next;
    }
}

public class Main {
    public ListNode removeElements(ListNode head, int val) {
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode curr = dummy;

        while (curr.next != null) {
            if (curr.next.val == val) {
                curr.next = curr.next.next;
            } else {
                curr = curr.next;
            }
        }
        return dummy.next;
    }

    public static void printList(ListNode head) {
        while (head != null) {
            System.out.print(head.val + " -> ");
            head = head.next;
```

```
        }
        System.out.println("null");
    }

    public static void main(String[] args) {
        Main solution = new Main();

        ListNode head = new ListNode(1, new ListNode(2, new ListNode(6, new ListNode(3, new ListNode(4, new ListNode(5, new ListNode(6)))))));
        System.out.print("Original List: ");
        printList(head);

        head = solution.removeElements(head, 6);
        System.out.print("Modified List: ");
        printList(head);
    }
}
```



TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(1)

5)PALINDROME LINKED LIST:

```
class ListNode {
    int val;
    ListNode next;
```

```
    ListNode() {}

    ListNode(int val) {
        this.val = val;
    }

    ListNode(int val, ListNode next) {
        this.val = val;
        this.next = next;
    }
}

public class Main {
    private static final int[] nums = new int[100000];

    public boolean isPalindrome(ListNode head) {
        int[] c = nums;
        ListNode current = head;
        int i = 0;

        while (current != null) {
            c[i] = current.val;
            i++;
            current = current.next;
        }

        int s = 0;
        int e = i;

        while (s < e) {
            if (c[s++] != c[--i]) {
```

```
                return false;
            }
        }

        return true;
    }

    public static void main(String[] args) {
        Main solution = new Main();

        ListNode head = new ListNode(1, new ListNode(2, new ListNode(2, new ListNode(1))));
        System.out.println("Is Palindrome: " + solution.isPalindrome(head));

        ListNode head2 = new ListNode(1, new ListNode(2));
        System.out.println("Is Palindrome: " + solution.isPalindrome(head2));
    }
}
```

TIME COMPLEXITY:O(N)

SPACE COMPLEXITY:O(1)

Output

```
Is Palindrome: true
Is Palindrome: false

=== Code Execution Successful ===
```

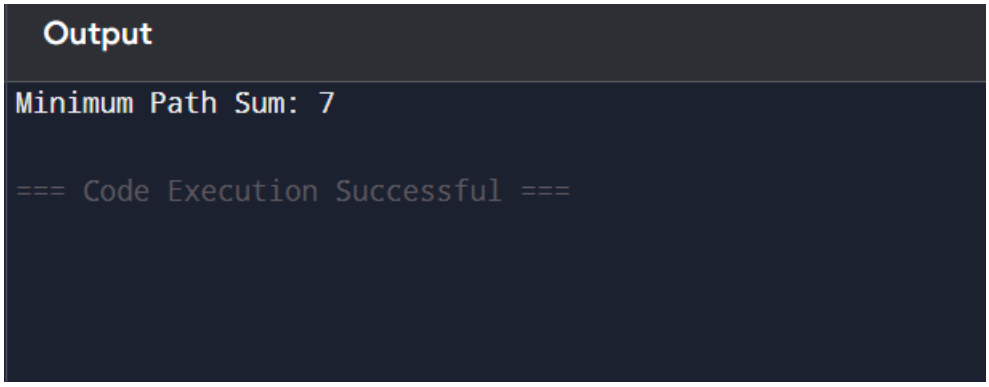
6)MINIMUM PATH SUM

```
public class Main {
    public int minPathSum(int[][] grid) {
        int dp[][] = new int[grid.length + 1][grid[0].length + 1];
```

```
        return sum(grid, 0, 0, dp);
    }

    private int sum(int arr[][], int i, int j, int dp[][]) {
        if (i >= arr.length || j >= arr[0].length) {
            return Integer.MAX_VALUE;
        }

        if (i == arr.length - 1 && j == arr[0].length - 1) {
            return arr[i][j];
        }

        if (dp[i][j] != 0) {
            return dp[i][j];
        }

        return dp[i][j] = arr[i][j] + Math.min(sum(arr, i, j + 1, dp), sum(arr, i + 1, j, dp));
    }

    public static void main(String[] args) {
        Main solution = new Main();

        int[][] grid = {
            {1, 3, 1},
            {1, 5, 1},
            {4, 2, 1}
        };

        System.out.println("Minimum Path Sum: " + solution.minPathSum(grid));
    }
}
```

TIME COMPLEXITY:O(m*n)

SPACE COMPLEXITY: O(m*n)



7)BST OR NOT:

```
class Node {
    int data;
    Node left, right;

    Node(int value) {
        data = value;
        left = right = null;
    }
}

class Main {
    static int maxValue(Node node) {
        if (node == null) return Integer.MIN_VALUE;
        return Math.max(node.data, Math.max(maxValue(node.left), maxValue(node.right)));
    }

    static int minValue(Node node) {
        if (node == null) return Integer.MAX_VALUE;
        return Math.min(node.data, Math.min(minValue(node.left), minValue(node.right)));
```

```
    }

    static boolean isBST(Node node) {
        if (node == null) return true;

        if (node.left != null && maxValue(node.left) >= node.data) return false;

        if (node.right != null && minValue(node.right) <= node.data) return false;

        return isBST(node.left) && isBST(node.right);
    }

    public static void main(String[] args) {
        Node root = new Node(4);
        root.left = new Node(2);
        root.right = new Node(5);
        root.left.left = new Node(1);
        root.left.right = new Node(3);

        if (isBST(root)) {
            System.out.println("True");
        } else {
            System.out.println("False");
        }
    }
}
```

Output

```
True

=== Code Execution Successful ===
```

8)COURSE SCHEDULE:

```
import java.util.ArrayList;


class Main {
   public boolean canFinish(int n, int[][] prerequisites) {
      ArrayList<ArrayList<Integer>> G = new ArrayList<>();
      for (int i = 0; i < n; i++) {
         G.add(new ArrayList<>());
      }
      int[] degree = new int[n];
      ArrayList<Integer> bfs = new ArrayList<>();
      for (int[] e : prerequisites) {
         G.get(e[1]).add(e[0]);
         degree[e[0]]++;
      }
      for (int i = 0; i < n; ++i) if (degree[i] == 0) bfs.add(i);
      for (int i = 0; i < bfs.size(); ++i)
         for (int j : G.get(bfs.get(i)))
            if (--degree[j] == 0) bfs.add(j);
      return bfs.size() == n;
   }

   public static void main(String[] args) {
      Main solution = new Main();
      int n = 2;
      int[][] prerequisites = {{1, 0}};
      System.out.println(solution.canFinish(n, prerequisites)); // Output: true
   }
}
```

Output

```
true

=== Code Execution Successful ===
```