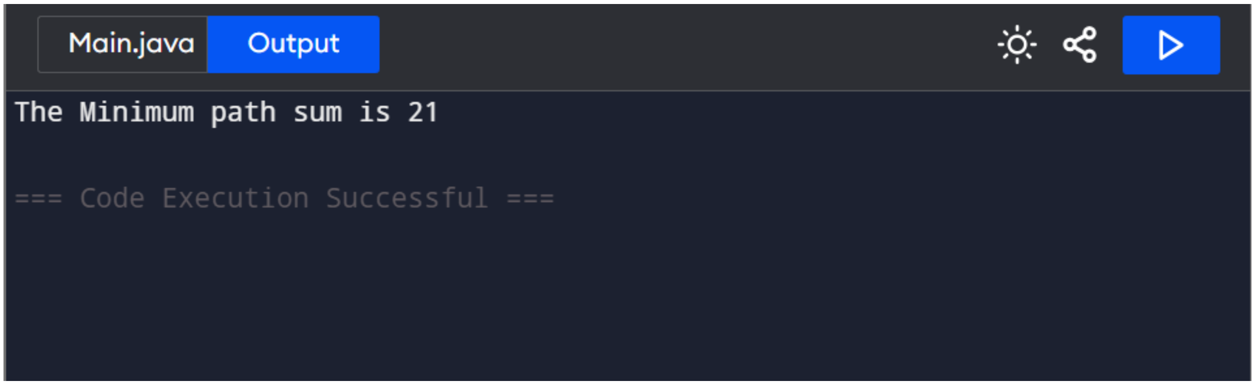
1.MINIMUM PATH SUM:

CODE:

```
import java.util.*;

class TUF {

    static int minSumPath(int n, int m, int[][] matrix) {

        int prev[] = new int[m];

        for (int i = 0; i < n; i++) {

            int temp[] = new int[m];

            for (int j = 0; j < m; j++) {

                if (i == 0 && j == 0)

                    temp[j] = matrix[i][j];

                else {

                    int up = matrix[i][j];

                    if (i > 0)

                        up += prev[j];

                    else

                        up += (int) Math.pow(10, 9);

                    int left = matrix[i][j];

                    if (j > 0)

                        left += temp[j - 1];

                    else

                        left += (int) Math.pow(10, 9);

                    temp[j] = Math.min(up, left);

                } }

            prev = temp;

        }

        return prev[m - 1];

    }
    public static void main(String args[]) {
        int matrix[][] = {
            {5, 9, 6},
            {11, 5, 2}
        };
        int n = matrix.length;
        int m = matrix[0].length;
        System.out.println(minSumPath("The Minimum path sum is+" n, m, matrix));
    }
}
```

OUTPUT:



```
The Minimum path sum is 21

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

Time Complexity: O(M*N)

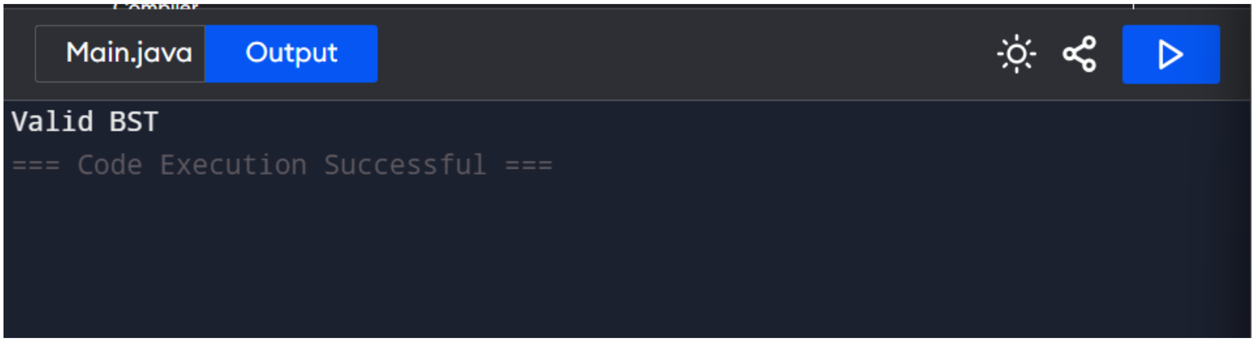
Space Complexity: O(N)

2. Validate Binary Search Tree:

CODE:

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Node root = new Node(7);
        root.left = new Node(5);
        root.left.left = new Node(3);
        root.left.right = new Node(6);
        root.right = new Node(10);
        root.right.left = new Node(9);
        root.right.right = new Node(15);
        Solution ob = new Solution();
        boolean ans = ob.isValidBST(root);
        if (ans == true) {
            System.out.print("Valid BST");
        } else {
            System.out.print("Invalid BST");
        }
    }
}
class Node {
    int data;
    Node left, right;
    Node(int data) {
        this.data = data;
        left = null;
        right = null;
    }
}
class Solution {
    private boolean checkBST(Node node, long min, long max) {
        if (node == null) return true;
        if (node.data <= min || node.data >= max) return false;

        if (checkBST(node.left, min, node.data) && checkBST(node.right, node.data,
        max))
       {
            return true;
        }
        return false;
    }
    public boolean isValidBST(Node root) {
        return checkBST(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }
}
```

OUTPUT:



```
Valid BST
=== Code Execution Successful ===
```

Time Complexity: O(N)

Space Complexity: O(1)

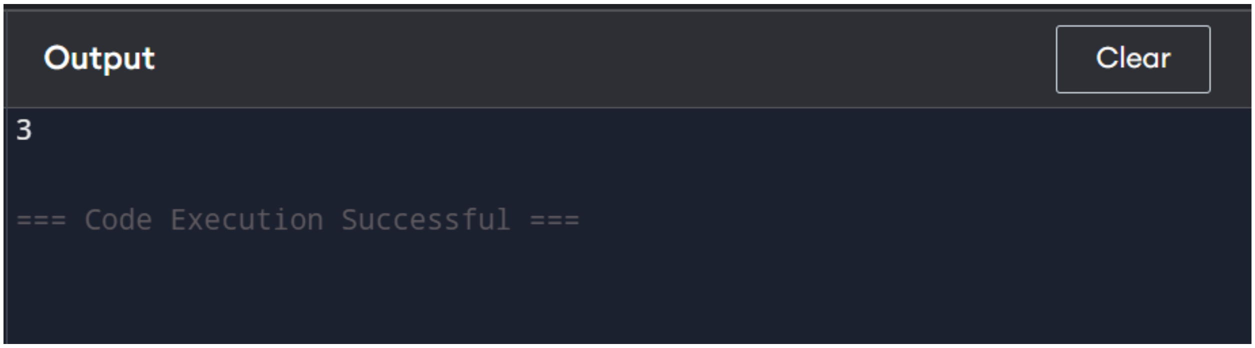
3. Word Ladder:

CODE:

```
import java.util.*;
import java.lang.*;
import java.io.*;
class Main {
    public static void main(String[] args) throws IOException {
        String startWord = "der", targetWord = "dfs";
        String[] wordList = {
            "des",
            "der",
            "dfr",
            "dgt",
            "dfs"
        };
        Solution obj = new Solution();
        int ans = obj.wordLadderLength(startWord, targetWord, wordList);
        System.out.print(ans);
        System.out.println();
    }

class Pair {
    String first;
    int second;
    Pair(String _first, int _second) {
        this.first = _first;
        this.second = _second;    }
}
class Solution {
```

```
public int wordLadderLength(String startWord, String targetWord, String[] wordList) {
   Queue < Pair > q = new LinkedList < > ();
   q.add(new Pair(startWord, 1));
   Set < String > st = new HashSet < String > ();
   int len = wordList.length;
   for (int i = 0; i < len; i++) {
      st.add(wordList[i]); }
   st.remove(startWord);
   while (!q.isEmpty()) {
      String word = q.peek().first;
      int steps = q.peek().second;
      q.remove();
      if (word.equals(targetWord) == true) return steps;
      for (int i = 0; i < word.length(); i++) {
         for (char ch = 'a'; ch <= 'z'; ch++) {
            char replacedCharArray[] = word.toCharArray();
            replacedCharArray[i] = ch;
            String replacedWord = new String(replacedCharArray);
            if (st.contains(replacedWord) == true) {
               st.remove(replacedWord);
               q.add(new Pair(replacedWord, steps + 1));
            }}}}
   return 0;
}}
```

OUTPUT:



```
3


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

Time Complexity: O(N * M * 26)

Space Complexity: O(N)

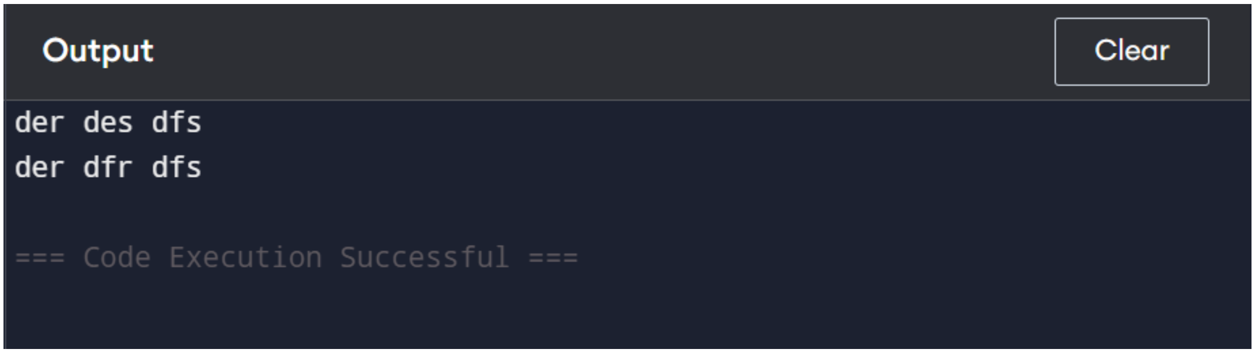
4. Word ladder –II

CODE:

```
import java.util.*;

import java.lang.*;

import java.io.*;

class comp implements Comparator < ArrayList < String >> {

   public int compare(ArrayList < String > a, ArrayList < String > b) {

      String x = "";

      String y = "";

      for (int i = 0; i < a.size(); i++)

         x += a.get(i);

      for (int i = 0; i < b.size(); i++)

         y += b.get(i);

      return x.compareTo(y);

   }
}

public class Main {
    public static void main(String[] args) throws IOException {
        String startWord = "der", targetWord = "dfs";
        String[] wordList = {
            "des",
            "der",
            "dfr",
            "dgt",
            "dfs"
        };

        Solution obj = new Solution();
        ArrayList < ArrayList < String >> ans = obj.findSequences(startWord, targetWord, wordList);
        if (ans.size() == 0)
            System.out.println(-1);
        else {
            Collections.sort(ans, new comp());
            for (int i = 0; i < ans.size(); i++) {
                for (int j = 0; j < ans.get(i).size(); j++) {
                    System.out.print(ans.get(i).get(j) + " ");
                }
                System.out.println();
            }
        }
    }
}
```

```
class Solution {
    public ArrayList < ArrayList < String >> findSequences(String startWord, String targetWord,
        String[] wordList) {
        Set < String > st = new HashSet < String > ();
        int len = wordList.length;
        for (int i = 0; i < len; i++) {
            st.add(wordList[i]);
        }
        Queue < ArrayList < String >> q = new LinkedList < > ();
        ArrayList < String > ls = new ArrayList < > ();
        ls.add(startWord);
        q.add(ls);
        ArrayList < String > usedOnLevel = new ArrayList < > ();
        usedOnLevel.add(startWord);
        int level = 0;
        ArrayList < ArrayList < String >> ans = new ArrayList < > ();
        int cnt = 0;
        while (!q.isEmpty()) {
            cnt++;
            ArrayList < String > vec = q.peek();
            q.remove();
             if (vec.size() > level) {
                level++;
                for (String it: usedOnLevel) {
                    st.remove(it);
                }
            }

            String word = vec.get(vec.size() - 1);
            if (word.equals(targetWord)) {

                if (ans.size() == 0) {

                    ans.add(vec);

                } else if (ans.get(0).size() == vec.size()) {

                    ans.add(vec);

                }

            }

            for (int i = 0; i < word.length(); i++) {

                for (char c = 'a'; c <= 'z'; c++) {

                    char replacedCharArray[] = word.toCharArray();

                    replacedCharArray[i] = c;

                    String replacedWord = new String(replacedCharArray);

                    if (st.contains(replacedWord) == true) {

                        vec.add(replacedWord);

                        ArrayList < String > temp = new ArrayList < > (vec);

                        q.add(temp);

                        usedOnLevel.add(replacedWord);

                        vec.remove(vec.size() - 1);

                    }

                }

            }

        }

        return ans;

    }
```

OUTPUT:



```
der des dfs
der dfr dfs

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

Time Complexity: $O(N \times L)$

Space Complexity: O(N×L)

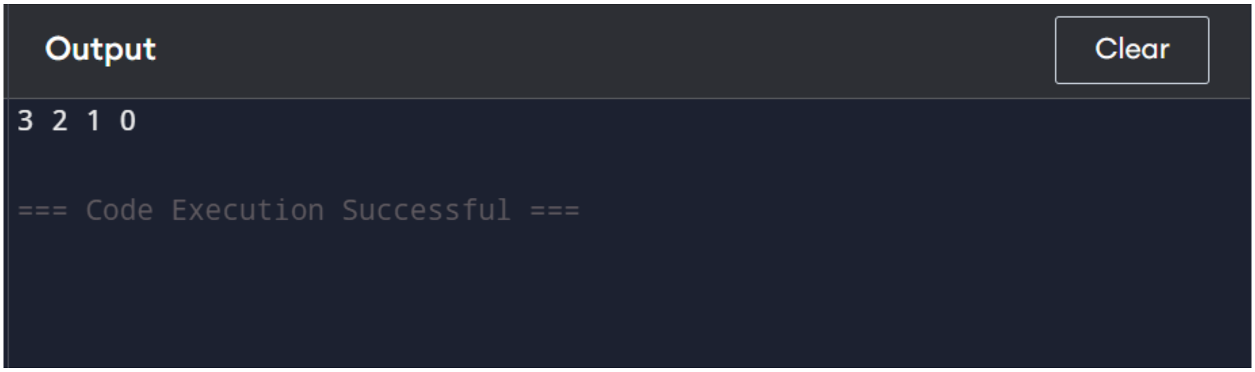
5. Course schedule:

CODE:

```
import java.util.*;
class Solution {
  static int[] findOrder(int n, int m, ArrayList<ArrayList<Integer>> prerequisites) {
    ArrayList<ArrayList<Integer>> adj = new ArrayList<>();
    for (int i = 0; i < n; i++) {
      adj.add(new ArrayList<>());
    }

    for (int i = 0; i < m; i++) {
adj.get(prerequisites.get(i).get(1)).add(prerequisites.get(i).get(0));
    }
    int indegree[] = new int[n];
    for (int i = 0; i < n; i++) {
      for (int it : adj.get(i)) {
        indegree[it]++;
      }
```

```
        }
        Queue<Integer> q = new LinkedList<Integer>();
        for (int i = 0; i < n; i++) {
            if (indegree[i] == 0) {
                q.add(i);
            }
        }
        int topo[] = new int[n];
        int ind = 0;
        while (!q.isEmpty()) {
            int node = q.peek();
            q.remove();
            topo[ind++] = node;
            for (int it : adj.get(node)) {
                indegree[it]--;
                if (indegree[it] == 0) q.add(it);
            }
        }
        if (ind == n) return topo;
        int[] arr = {};
        return arr;
    }
}
public class tUf {
    public static void main(String[] args) {
        int N = 4;
        int M = 3;
        ArrayList<ArrayList<Integer>> prerequisites = new ArrayList<>();
        for (int i = 0; i < N; i++) {
            prerequisites.add(i, new ArrayList<>());
        }
        prerequisites.get(0).add(0);
        prerequisites.get(0).add(1);
        prerequisites.get(1).add(1);
        prerequisites.get(1).add(2);
        prerequisites.get(2).add(2);
        prerequisites.get(2).add(3);
        int[] ans = Solution.findOrder(N, M, prerequisites);
        for (int task : ans) {
            System.out.print(task + " ");
        }
        System.out.println("");
    }
}
```

OUTPUT:



```
3 2 1 0

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

Time Complexity:O(N+M)

Space Complexity: O(N+M)

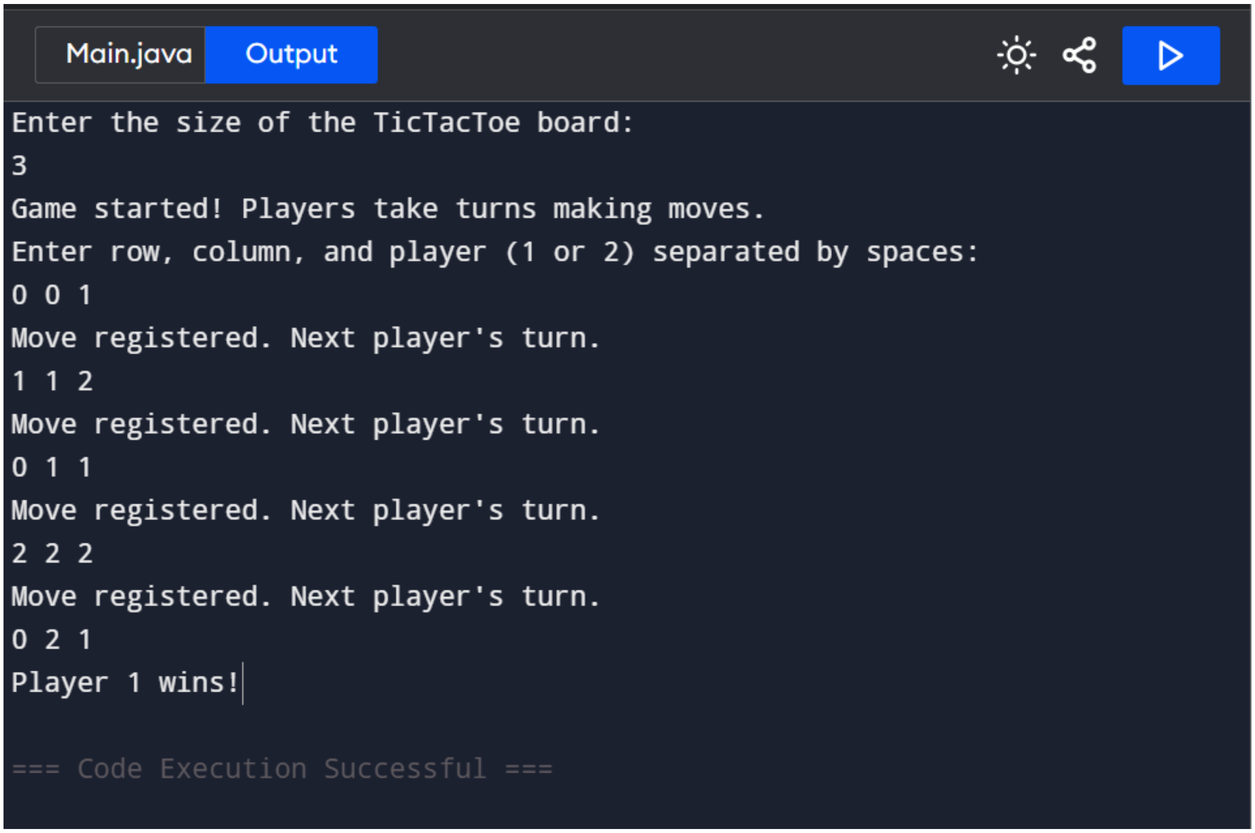
6. Design tic tac toe:

CODE:

```
import java.util.Scanner;

public class TicTacToe {

    private int size;

    private int[][] moveCounters;

    public TicTacToe(int n) {

        moveCounters = new int[2][n * 2 + 2];

        this.size = n;

    }

    public int move(int row, int col, int player) {

        moveCounters[player - 1][row]++;

        moveCounters[player - 1][col + size]++;

        if (row == col) {

            moveCounters[player - 1][size * 2]++;

        }

        if (row + col == size - 1) {

            moveCounters[player - 1][size * 2 + 1]++;

        }

        if (moveCounters[player - 1][row] == size

                || moveCounters[player - 1][col + size] == size

                || moveCounters[player - 1][size * 2] == size

                || moveCounters[player - 1][size * 2 + 1] == size) {

            return player;

        }

        return 0;

    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of the TicTacToe board:");
        int n = sc.nextInt();
        TicTacToe game = new TicTacToe(n);
        System.out.println("Game started! Players take turns making moves.");
        System.out.println("Enter row, column, and player (1 or 2) separated by spaces:");
        int moves = 0;
        while (true) {
            int row = sc.nextInt();
            int col = sc.nextInt();
            int player = sc.nextInt();
            if (row < 0 || row >= n || col < 0 || col >= n || (player != 1 && player != 2)) {
                System.out.println("Invalid move. Try again.");
                continue;
            }
            int result = game.move(row, col, player);
            moves++;
            if (result != 0) {
                System.out.println("Player " + result + " wins!");
                break;
            }
            if (moves == n * n) {
                System.out.println("It's a draw!");
                break;
            }
            System.out.println("Move registered. Next player's turn.");
        }
        sc.close();
    }
```

}

OUTPUT:



```
Enter the size of the TicTacToe board:
3
Game started! Players take turns making moves.
Enter row, column, and player (1 or 2) separated by spaces:
0 0 1
Move registered. Next player's turn.
1 1 2
Move registered. Next player's turn.
0 1 1
Move registered. Next player's turn.
2 2 2
Move registered. Next player's turn.
0 2 1
Player 1 wins!

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

Time Complexity: O(n^2)

Space Complexity: O(n)