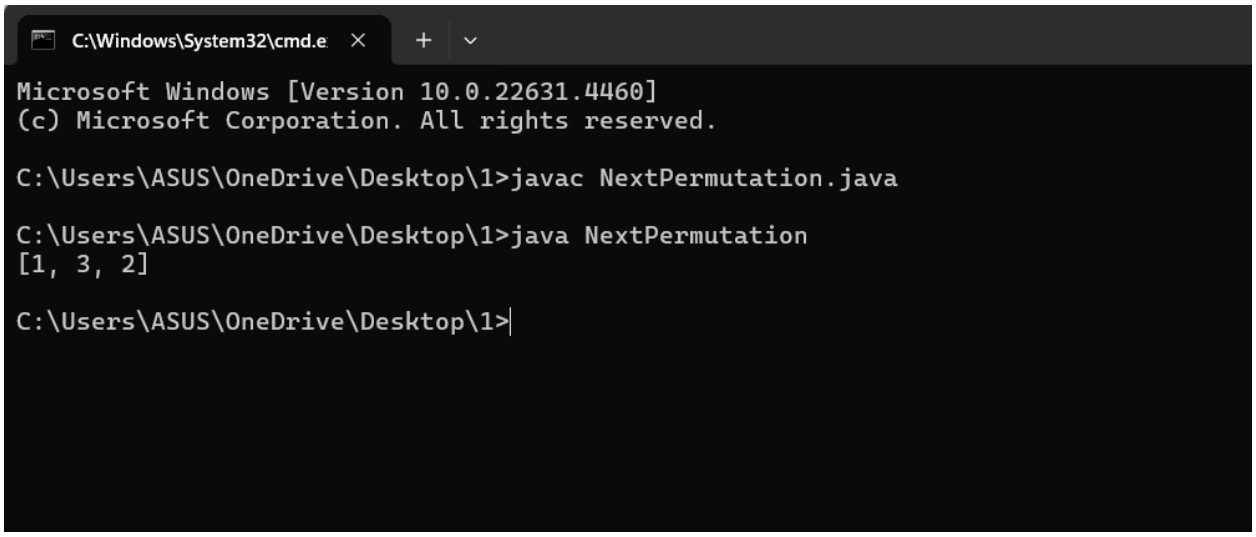
1)Next permutation:

```
class Solution {
    public static void main(String[] args) {
        int[] nums = {1, 2, 3};
        Solution sol = new Solution();
        sol.nextPermutation(nums);
        System.out.println(Arrays.toString(nums));
    }

    public void nextPermutation(int[] nums) {
        int n = nums.length;
        int i = n - 2;
        while (i >= 0 && nums[i] >= nums[i + 1]) {
            i--;
        }
        if (i >= 0) {
            int j = n - 1;
            while (nums[j] <= nums[i]) {
                j--;
            }
            swap(nums, i, j);
        }
        reverse(nums, i + 1, n - 1);
    }

    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }

    private void reverse(int[] nums, int i, int j) {
        while (i < j) {
            swap(nums, i++, j--);
```

```
        }
    }
}
```



Time Complexity:O(n)

2)Spiral Matrix

```
public class SpiralMatrix {
    public static void main(String[] args) {
        int n = 3;
        int[][] matrix = generateMatrix(n);
        for (int[] row : matrix) {
            for (int num : row) {
                System.out.print(num + " ");
            }
            System.out.println();
        }
    }

    public static int[][] generateMatrix(int n) {
        int[][] matrix = new int[n][n];
        int left = 0, right = n - 1, top = 0, bottom = n - 1;
        int num = 1;
```

```
        while (left <= right && top <= bottom) {
            for (int i = left; i <= right; i++) matrix[top][i] = num++;
            top++;

            for (int i = top; i <= bottom; i++) matrix[i][right] = num++;
            right--;

            if (top <= bottom) {
                for (int i = right; i >= left; i--) matrix[bottom][i] = num++;
                bottom--;
            }

            if (left <= right) {
                for (int i = bottom; i >= top; i--) matrix[i][left] = num++;
                left++;
            }
        }

        return matrix;
    }
}
```

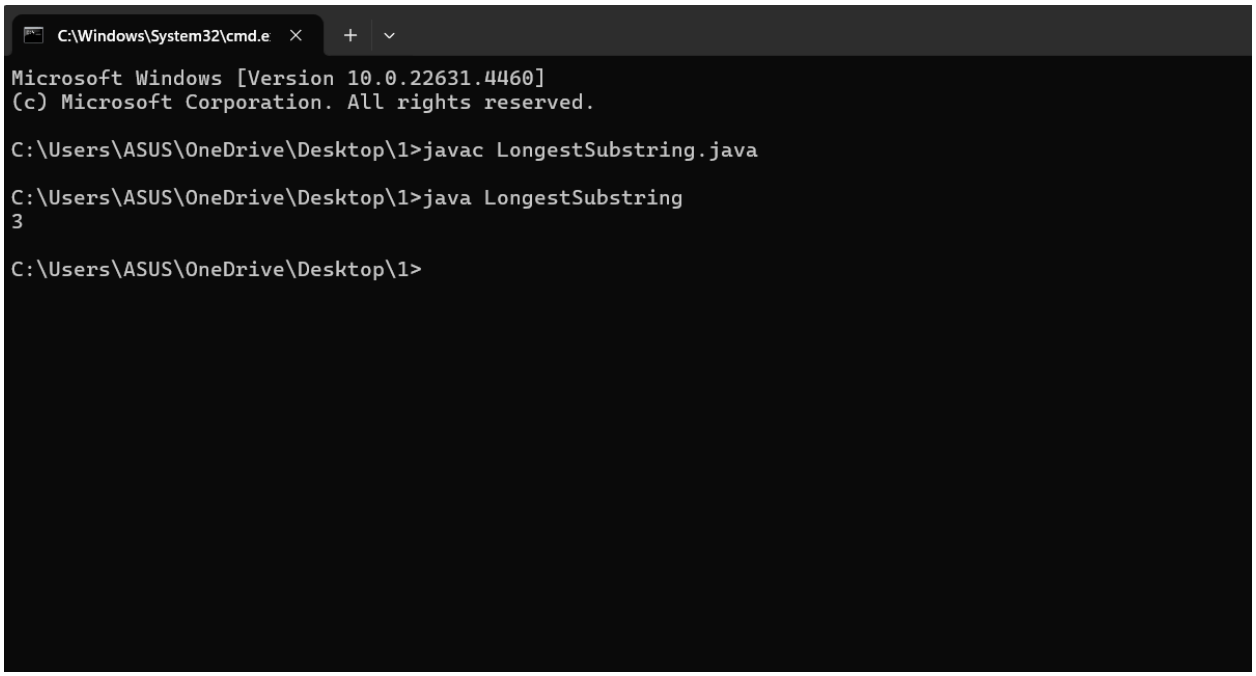
```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\OneDrive\Desktop\1>javac SpiralMatrix.java

C:\Users\ASUS\OneDrive\Desktop\1>java SpiralMatrix
1 2 3
8 9 4
7 6 5

C:\Users\ASUS\OneDrive\Desktop\1>
```

Time:O(n^2)

3)Longest substring without repeating characters

```
import java.util.*;

public class LongestSubstring {
    public static void main(String[] args) {
        String s = "abcabcbb";
        System.out.println(lengthOfLongestSubstring(s));
    }

    public static int lengthOfLongestSubstring(String s) {
        Set<Character> set = new HashSet<>();
        int left = 0, right = 0, maxLength = 0;

        while (right < s.length()) {
            if (!set.contains(s.charAt(right))) {
                set.add(s.charAt(right));
                maxLength = Math.max(maxLength, right - left + 1);
                right++;
```

```
        } else {
            set.remove(s.charAt(left));
            left++;
        }
    }

    return maxLength;
  }
}
```



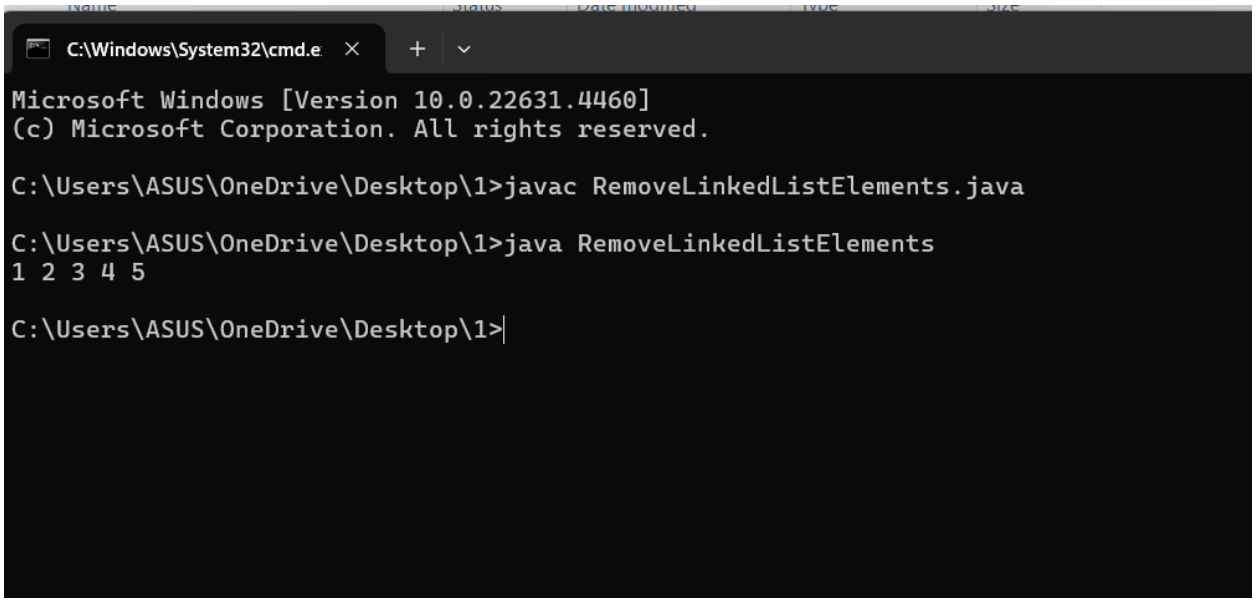
Time:O(n)

4)Remove linked list elements

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

```
public class RemoveLinkedListElements {
    public static void main(String[] args) {
        ListNode head = new ListNode(1);
        head.next = new ListNode(2);
        head.next.next = new ListNode(6);
        head.next.next.next = new ListNode(3);
        head.next.next.next.next = new ListNode(4);
        head.next.next.next.next.next = new ListNode(5);
        head.next.next.next.next.next.next = new ListNode(6);

        int val = 6;
        head = removeElements(head, val);

        printList(head);
    }

    public static ListNode removeElements(ListNode head, int val) {
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode current = dummy;

        while (current.next != null) {
            if (current.next.val == val) {
                current.next = current.next.next;
            } else {
                current = current.next;
            }
        }

        return dummy.next;
    }

    public static void printList(ListNode head) {
        while (head != null) {
```

```
            System.out.print(head.val + " ");
            head = head.next;
        }
        System.out.println();
    }
}
```



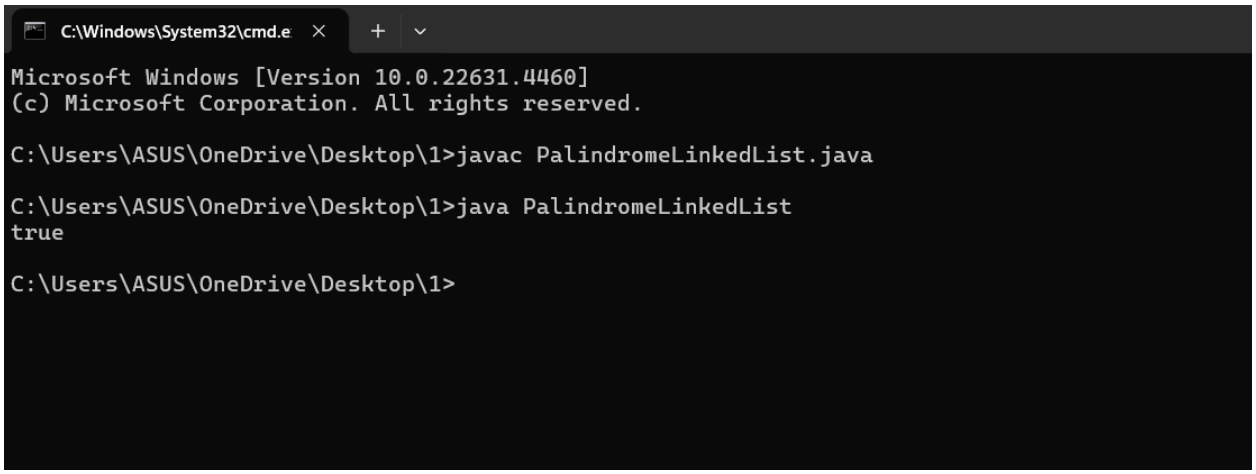
Time:O(n)

5)Palindrome linked list

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

public class PalindromeLinkedList {
    public static void main(String[] args) {
        ListNode head = new ListNode(1);
```

```
    head.next = new ListNode(2);
    head.next.next = new ListNode(2);
    head.next.next.next = new ListNode(1);

    System.out.println(isPalindrome(head));
}

public static boolean isPalindrome(ListNode head) {
    if (head == null || head.next == null) {
        return true;
    }

    ListNode slow = head;
    ListNode fast = head;
    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
    }

    ListNode secondHalf = reverse(slow);
    ListNode firstHalf = head;
    while (secondHalf != null) {
        if (firstHalf.val != secondHalf.val) {
            return false;
        }
        firstHalf = firstHalf.next;
        secondHalf = secondHalf.next;
    }

    return true;
}

public static ListNode reverse(ListNode head) {
    ListNode prev = null;
    ListNode current = head;
```

```
        while (current != null) {
            ListNode next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }
        return prev;
    }
}
```

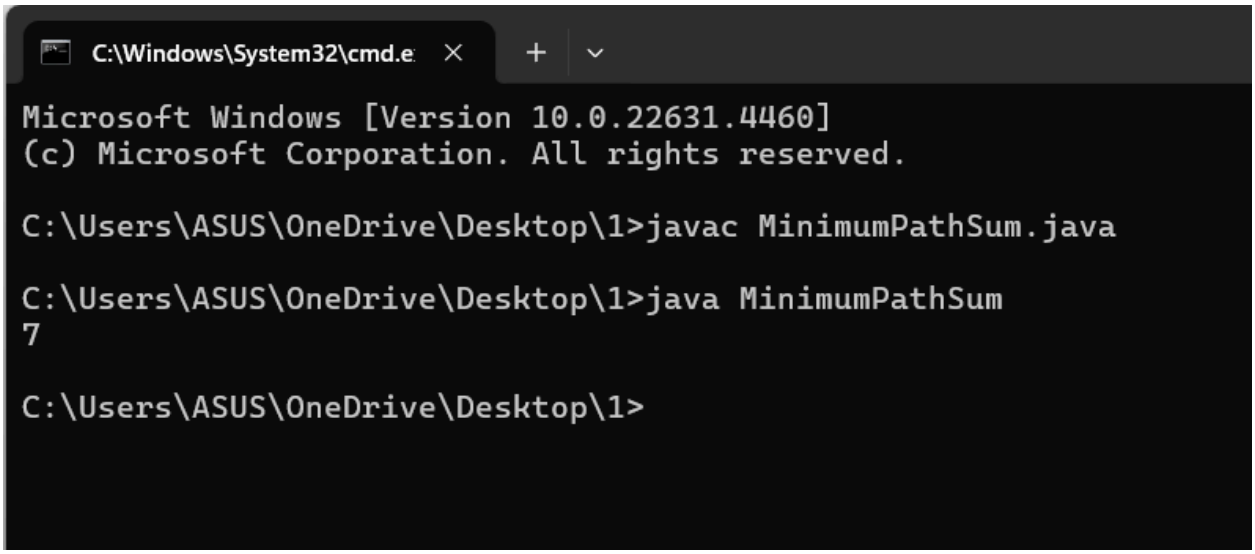


Time:O(n)

6)Minimum path sum

```
public class MinimumPathSum {
    public static void main(String[] args) {
        int[][] grid = {
            {1, 3, 1},
            {1, 5, 1},
            {4, 2, 1}
        };
        System.out.println(minPathSum(grid));
    }

    public static 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 j = 1; j < n; j++) {
            grid[0][j] += grid[0][j - 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];
    }
}
```

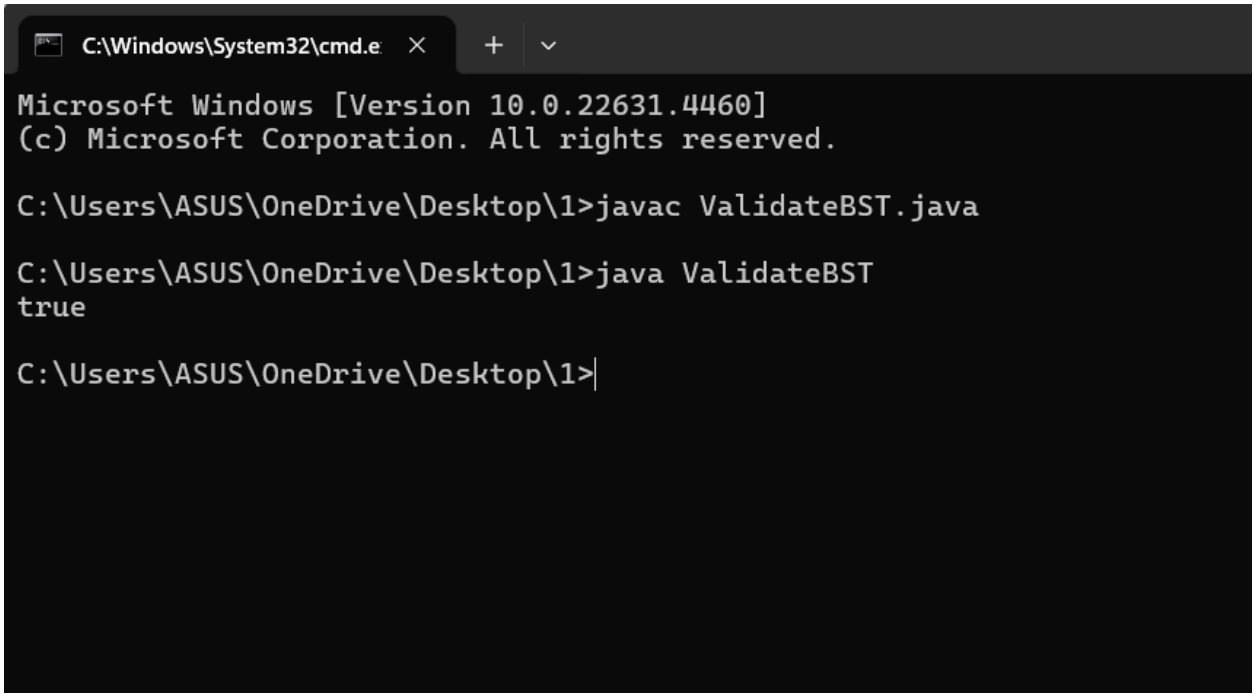


Time:O(mxn)

7)Validate binary search tree

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int val) {
        this.val = val;
    }
}

public class ValidateBST {
    public static void main(String[] args) {
        TreeNode root = new TreeNode(2);
        root.left = new TreeNode(1);
        root.right = new TreeNode(3);

        System.out.println(isValidBST(root));
    }

    public static boolean isValidBST(TreeNode root) {
        return isValidBST(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    public static boolean isValidBST(TreeNode root, long min, long max) {
        if (root == null) {
            return true;
        }

        if (root.val <= min || root.val >= max) {
            return false;
        }

        return isValidBST(root.left, min, root.val) && isValidBST(root.right,
root.val, max);
```

```
    }
}
```



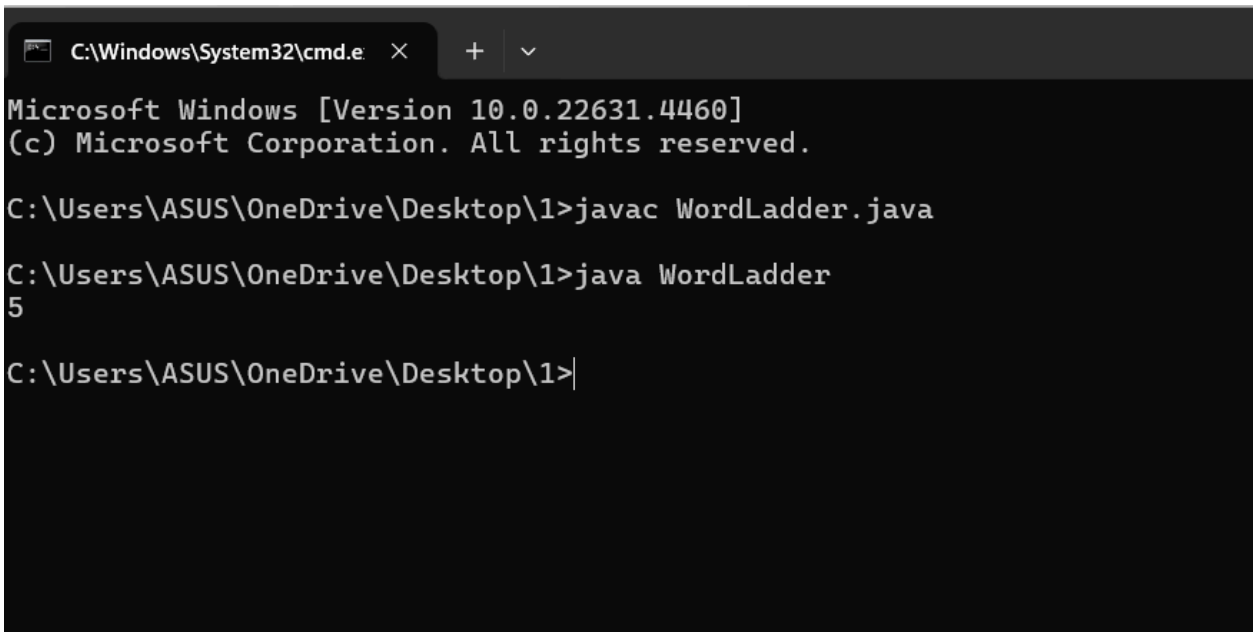
Time:O(n)

8)Word ladder

```
import java.util.*;

public class WordLadder {
    public static void main(String[] args) {
        String beginWord = "hit";
        String endWord = "cog";
        List<String> wordList = Arrays.asList("hot", "dot", "dog", "lot", "log",
"cog");

        System.out.println(ladderLength(beginWord, endWord, wordList));
    }
```

```
    public static int ladderLength(String beginWord, String endWord,
List<String> wordList) {
        if (!wordList.contains(endWord)) {
            return 0;
        }

        Set<String> wordSet = new HashSet<>(wordList);
        Queue<String> queue = new LinkedList<>();
        queue.offer(beginWord);
        int level = 1;

        while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                String word = queue.poll();
                if (word.equals(endWord)) {
                    return level;
                }

                char[] chars = word.toCharArray();
                for (int j = 0; j < chars.length; j++) {
                    char originalChar = chars[j];
                    for (char c = 'a'; c <= 'z'; c++) {
                        if (chars[j] != c) {
                            chars[j] = c;
                            String newWord = new String(chars);
                            if (wordSet.contains(newWord)) {
                                queue.offer(newWord);
                                wordSet.remove(newWord);
                            }
                        }
                    }
                    chars[j] = originalChar;
                }
            }
```

```
            level++;
        }
        return 0;
    }
}
```



Time:O(nxmx26)

9)Course schedule

```
import java.util.*;

public class CourseSchedule {
    public static void main(String[] args) {
        int numCourses = 2;
        int[][] prerequisites = {{1, 0}};

        System.out.println(canFinish(numCourses, prerequisites));
    }

    public static boolean canFinish(int numCourses, int[][] prerequisites) {
        List<List<Integer>> graph = new ArrayList<>();
```

```
        for (int i = 0; i < numCourses; i++) {
            graph.add(new ArrayList<>());
        }

        for (int[] pre : prerequisites) {
            graph.get(pre[1]).add(pre[0]);
        }

        int[] visited = new int[numCourses]; // 0 = unvisited, 1 = visiting, 2 =
visited

        for (int i = 0; i < numCourses; i++) {
            if (visited[i] == 0 && hasCycle(graph, visited, i)) {
                return false;
            }
        }
        return true;
    }

    public static boolean hasCycle(List<List<Integer>> graph, int[] visited, int
course) {
        if (visited[course] == 1) return true;          if (visited[course] == 2) return
false;

        visited[course] = 1;
        for (int neighbor : graph.get(course)) {
            if (hasCycle(graph, visited, neighbor)) {
                return true;
            }
        }

        visited[course] = 2;
        return false;
    }
}
```

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\OneDrive\Desktop\1>javac CourseSchedule.java

C:\Users\ASUS\OneDrive\Desktop\1>java CourseSchedule
true

C:\Users\ASUS\OneDrive\Desktop\1>
```

Time:O(V+E)

10)Design tic tac toe

```
import java.util.Scanner;

public class TicTacToe {
    private char[][] board;
    private char currentPlayer;

    public TicTacToe() {
        board = new char[3][3];
        currentPlayer = 'X';
        initializeBoard();
    }

    public void initializeBoard() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                board[i][j] = ' ';
            }
```

```
        }
    }

    public void printBoard() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(board[i][j]);
                if (j < 2) System.out.print("|");
            }
            System.out.println();
            if (i < 2) System.out.println("-----");
        }
    }

    public boolean makeMove(int row, int col) {
        if (row < 0 || row >= 3 || col < 0 || col >= 3 || board[row][col] != ' ') {
            return false;
        }
        board[row][col] = currentPlayer;
        return true;
    }

    public boolean checkWinner() {
        for (int i = 0; i < 3; i++) {
            if (board[i][0] == currentPlayer && board[i][1] == currentPlayer &&
board[i][2] == currentPlayer) {
                return true;
            }
            if (board[0][i] == currentPlayer && board[1][i] == currentPlayer &&
board[2][i] == currentPlayer) {
                return true;
            }
        }
        if (board[0][0] == currentPlayer && board[1][1] == currentPlayer &&
board[2][2] == currentPlayer) {
```

```
            return true;
        }
        if (board[0][2] == currentPlayer && board[1][1] == currentPlayer &&
board[2][0] == currentPlayer) {
            return true;
        }
        return false;
    }

    public boolean isBoardFull() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (board[i][j] == ' ') {
                    return false;
                }
            }
        }
        return true;
    }

    public void switchPlayer() {
        currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
    }

    public static void main(String[] args) {
        TicTacToe game = new TicTacToe();
        Scanner scanner = new Scanner(System.in);

        while (true) {
            game.printBoard();
            System.out.println("Player " + game.currentPlayer + "'s turn:");
            System.out.print("Enter row (0-2): ");
            int row = scanner.nextInt();
            System.out.print("Enter column (0-2): ");
            int col = scanner.nextInt();
```

```
            if (!game.makeMove(row, col)) {
                System.out.println("Invalid move! Try again.");
                continue;
            }

            if (game.checkWinner()) {
                game.printBoard();
                System.out.println("Player " + game.currentPlayer + " wins!");
                break;
            }

            if (game.isBoardFull()) {
                game.printBoard();
                System.out.println("It's a tie!");
                break;
            }

            game.switchPlayer();
        }
        scanner.close();
    }
}
```

```
C:\Windows\System32\cmd.e

Microsoft Windows [Version 10.0.22631.4460]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ASUS\OneDrive\Desktop\1>javac TicTacToe.java

C:\Users\ASUS\OneDrive\Desktop\1>java TicTacToe
 | |
-----
 | |
-----
 | |
Player X's turn:
Enter row (0-2): 1
Enter column (0-2): 1
 | |
-----
 |X|
-----
 | |
Player O's turn:
Enter row (0-2):
```

Time:O(1)