}

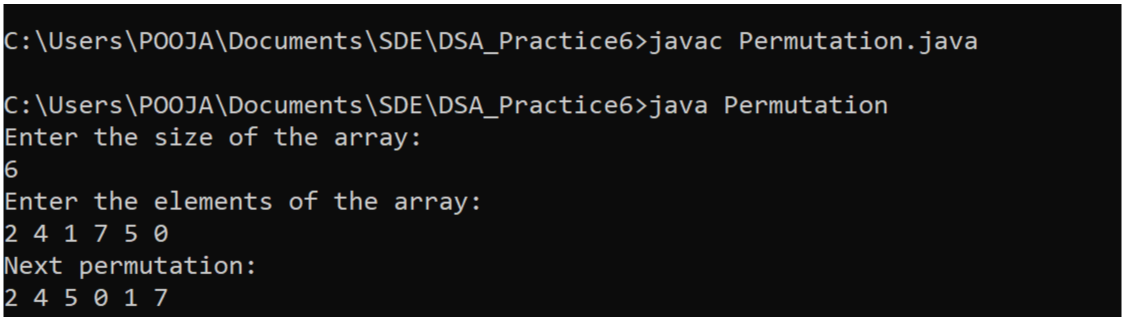
19/11/2024 DSA PRACTICE 6

1.next permutation

```
import java.util.Scanner;


class Permutation {
    public void nextPermutation(int[] nums) {
        int n = nums.length;
        int pivot = -1;
        for (int i = n - 2; i >= 0; i--) {
            if (nums[i] < nums[i + 1]) {
                pivot = i;
                break;
            }
        }
        if (pivot == -1) {
            reverse(nums, 0, n - 1);
            return;
        }
        for (int i = n - 1; i > pivot; i--) {
            if (nums[i] > nums[pivot]) {
                swap(nums, i, pivot);
                break;
            }
        }
    reverse(nums, pivot + 1, n - 1);
}

private void reverse(int[] nums, int start, int end) {
    while (start < end) {
        swap(nums, start++, end--);
    }
}

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

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the size of the array:");
    int n = scanner.nextInt();
    int[] nums = new int[n];
    System.out.println("Enter the elements of the array:");
    for (int i = 0; i < n; i++) {
        nums[i] = scanner.nextInt();
    }
    Permutation solution = new Permutation();
```

```
        solution.nextPermutation(nums);
        System.out.println("Next permutation:");
        for (int num : nums) {
            System.out.print(num + " ");
        }
    }
}
```

Time Complexity : O(n)



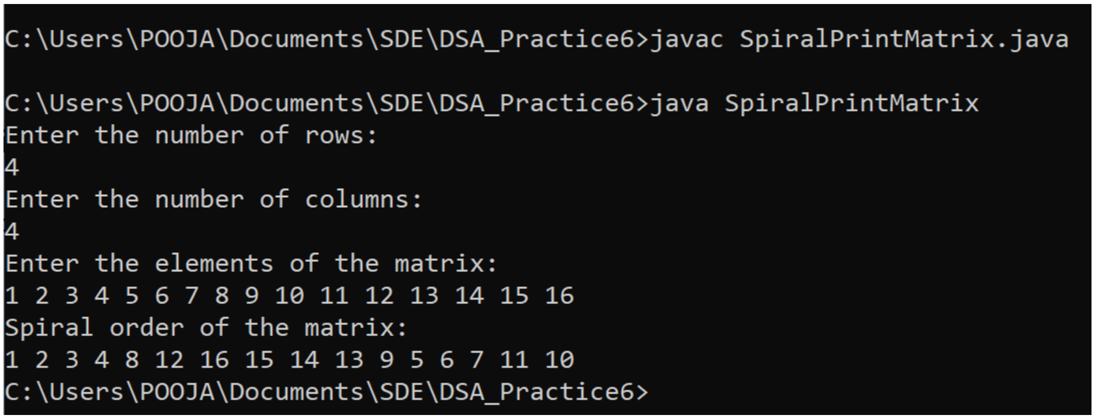
```
C:\Users\POOJA\Documents\SDE\DSA_Practice6>javac Permutation.java

C:\Users\POOJA\Documents\SDE\DSA_Practice6>java Permutation
Enter the size of the array:
6
Enter the elements of the array:
2 4 1 7 5 0
Next permutation:
2 4 5 0 1 7
```

2.Spiral matrix

```
import java.util.Scanner;


public class SpiralPrintMatrix {


    public static void spiralPrint(int m, int n, int[][] a) {
        int top = 0, bottom = m - 1, left = 0, right = n - 1;
        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; ++i) {
                System.out.print(a[top][i] + " ");
            }
            top++;
        for (int i = top; i <= bottom; ++i) {
            System.out.print(a[i][right] + " ");
        }
        right--;
        if (top <= bottom) {
            for (int i = right; i >= left; --i) {
                System.out.print(a[bottom][i] + " ");
            }
            bottom--;
        }
        if (left <= right) {
            for (int i = bottom; i >= top; --i) {
                System.out.print(a[i][left] + " ");
            }
            left++;
        }
    }
}

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

    System.out.println("Enter the number of rows:");
    int m = scanner.nextInt();

    System.out.println("Enter the number of columns:");
    int n = scanner.nextInt();
```

```
        int[][] matrix = new int[m][n];

        System.out.println("Enter the elements of the matrix:");

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

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

                matrix[i][j] = scanner.nextInt();

            }

        }


        System.out.println("Spiral order of the matrix:");

        spiralPrint(m, n, matrix);

    }

}
```

Time Complexity : O(m*n)



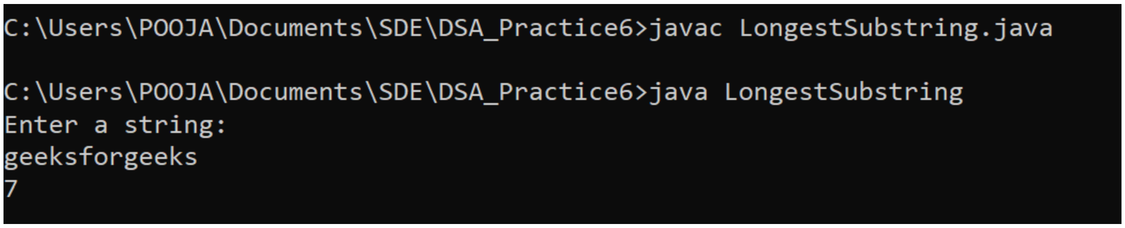
```
C:\Users\POOJA\Documents\SDE\DSA_Practice6>javac SpiralPrintMatrix.java

C:\Users\POOJA\Documents\SDE\DSA_Practice6>java SpiralPrintMatrix
Enter the number of rows:
4
Enter the number of columns:
4
Enter the elements of the matrix:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Spiral order of the matrix:
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
C:\Users\POOJA\Documents\SDE\DSA_Practice6>
```

3.Longest substring without repeating characters

```
import java.util.Scanner;


class LongestSubstring{


    static int longestUniqueSubstr(String s) {

        if (s.length() == 0)
```

```
            return 0;
        if (s.length() == 1)
            return 1;

        int maxLength = 0;
        boolean[] visited = new boolean[256];
        int left = 0, right = 0;

        while (right < s.length()) {
            while (visited[s.charAt(right)]) {
                visited[s.charAt(left)] = false;
                left++;
            }
            visited[s.charAt(right)] = true;
            maxLength = Math.max(maxLength, (right - left + 1));
            right++;
        }
        return maxLength;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a string:");
        String s = scanner.nextLine();
        System.out.println(longestUniqueSubstr(s));
    }
}
```

Time Complexity : O(n)



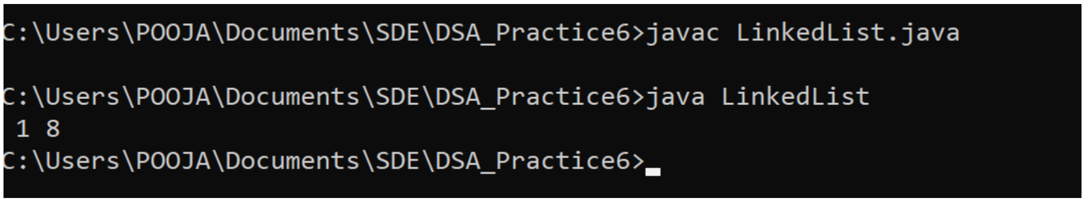
```
C:\Users\POOJA\Documents\SDE\DSA_Practice6>javac LongestSubstring.java

C:\Users\POOJA\Documents\SDE\DSA_Practice6>java LongestSubstring
Enter a string:
geeksforgeeks
7
```

4.Remove linked list elements

```
class Node {
    int data;
    Node next;

    Node(int new_data) {
        data = new_data;
        next = null;
    }
}

public class LinkedList {

    static Node deleteOccurrences(Node head, int key) {
        Node curr = head, prev = null;

        while (curr != null) {
            if (curr.data == key) {
                if (prev == null) {
                    head = curr.next;
                } else {
                    prev.next = curr.next;
                }
                curr = curr.next;
            } else {
                prev = curr;
                curr = curr.next;
            }
        }

        return head;
    }

    static void printList(Node curr) {
        while (curr != null) {
            System.out.print(" " + curr.data);
            curr = curr.next;
        }
    }

    public static void main(String[] args) {
        Node head = new Node(2);
        head.next = new Node(2);
        head.next.next = new Node(1);
        head.next.next.next = new Node(8);
        head.next.next.next.next = new Node(2);

        int key = 2;

        head = deleteOccurrences(head, key);
        printList(head);
```

```
        }

}
```

Time Complexity : O(n)



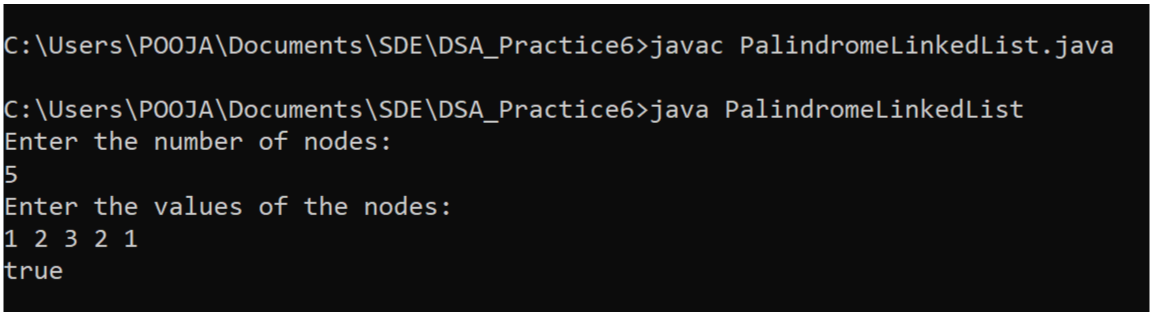
```
:\Users\POOJA\Documents\SDE\DSA_Practice6>javac LinkedList.java

:\Users\POOJA\Documents\SDE\DSA_Practice6>java LinkedList
 1 8
:\Users\POOJA\Documents\SDE\DSA_Practice6>
```

5.Palindrome linked list

```
import java.util.Scanner;


class Node {

        int data;

        Node next;

        Node(int d) {

                data = d;

                next = null;

        }

}


class PalindromeLinkedList{


        static Node reverseList(Node head) {

                Node prev = null;

                Node curr = head;

                Node next;

                while (curr != null) {

                        next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}

static boolean isIdentical(Node n1, Node n2) {
    while (n1 != null && n2 != null) {
        if (n1.data != n2.data)
            return false;
        n1 = n1.next;
        n2 = n2.next;
    }
    return true;
}

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

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

    Node head2 = reverseList(slow.next);
```

```
        slow.next = null;

        boolean ret = isIdentical(head, head2);

        head2 = reverseList(head2);
        slow.next = head2;

        return ret;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of nodes:");
        int n = scanner.nextInt();

        System.out.println("Enter the values of the nodes:");
        Node head = new Node(scanner.nextInt());
        Node temp = head;

        for (int i = 1; i < n; i++) {
            temp.next = new Node(scanner.nextInt());
            temp = temp.next;
        }

        if (isPalindrome(head))
            System.out.println("true");
        else
            System.out.println("false");
    }
}
```

Time Complexity : O(n)

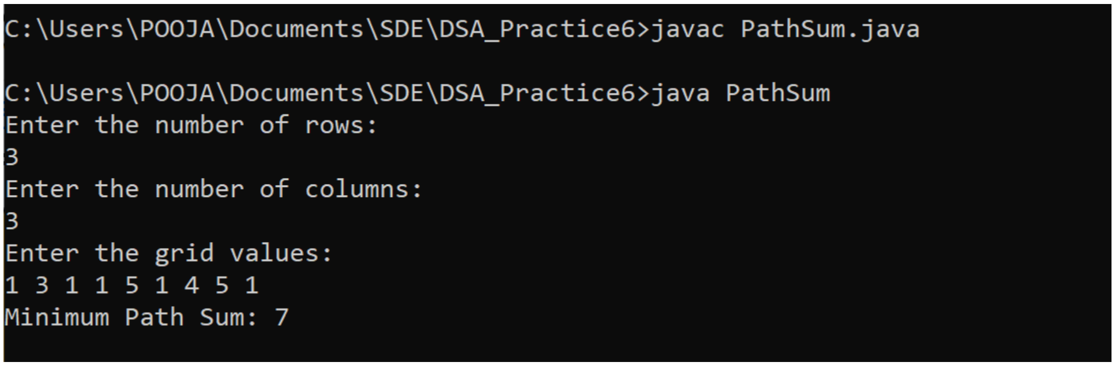


```
C:\Users\POOJA\Documents\SDE\DSA_Practice6>javac PalindromeLinkedList.java

C:\Users\POOJA\Documents\SDE\DSA_Practice6>java PalindromeLinkedList
Enter the number of nodes:
5
Enter the values of the nodes:
1 2 3 2 1
true
```

6.Minimum path sum

```
import java.util.Scanner;


class PathSum {
    public int minPathSum(int[][] grid) {
        int m = grid.length, n = grid[0].length;


        for (int j = 1; j < n; j++) {
            grid[0][j] += grid[0][j - 1];
        }


        for (int i = 1; i < m; i++) {
            grid[i][0] += grid[i - 1][0];
        }


        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];
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of rows:");
        int m = scanner.nextInt();
        System.out.println("Enter the number of columns:");
        int n = scanner.nextInt();

        int[][] grid = new int[m][n];
        System.out.println("Enter the grid values:");
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                grid[i][j] = scanner.nextInt();
            }
        }

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

Time Complexity : O(m × n)



```
C:\Users\POOJA\Documents\SDE\DSA_Practice6>javac PathSum.java

C:\Users\POOJA\Documents\SDE\DSA_Practice6>java PathSum
Enter the number of rows:
3
Enter the number of columns:
3
Enter the grid values:
1 3 1 1 5 1 4 5 1
Minimum Path Sum: 7
```

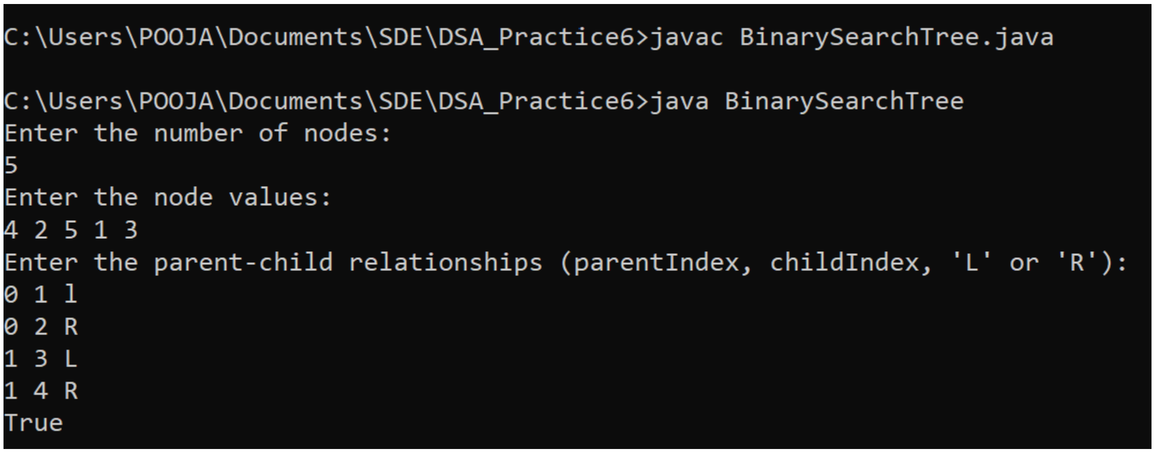
7.Validate binary search tree

```
import java.util.Scanner;


class Node {
    int data;
    Node left, right;


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


class BinarySearchTree {


    static boolean isBST(Node root) {
        Node curr = root;
        Node pre;
        int prevValue = Integer.MIN_VALUE;
```

```
while (curr != null) {
    if (curr.left == null) {
        if (curr.data <= prevValue) {
            return false;
        }
        prevValue = curr.data;
        curr = curr.right;
    } else {
        pre = curr.left;
        while (pre.right != null && pre.right != curr) {
            pre = pre.right;
        }

        if (pre.right == null) {
            pre.right = curr;
            curr = curr.left;
        } else {
            pre.right = null;
            if (curr.data <= prevValue) {
                return false;
            }
            prevValue = curr.data;
            curr = curr.right;
        }
    }
}

return true;
```

```
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of nodes:");
        int n = scanner.nextInt();

        if (n == 0) {
            System.out.println("Tree is empty");
            return;
        }

        System.out.println("Enter the node values:");
        Node[] nodes = new Node[n];
        for (int i = 0; i < n; i++) {
            nodes[i] = new Node(scanner.nextInt());
        }

        System.out.println("Enter the parent-child relationships (parentIndex, childIndex, 'L' or 'R'):");
        for (int i = 0; i < n - 1; i++) {
            int parentIndex = scanner.nextInt();
            int childIndex = scanner.nextInt();
            char direction = scanner.next().charAt(0);

            if (direction == 'L') {
                nodes[parentIndex].left = nodes[childIndex];
            } else {
                nodes[parentIndex].right = nodes[childIndex];
            }

        }


        if (isBST(nodes[0])) {

            System.out.println("True");

        } else {

            System.out.println("False");

        }

    }

}
```

Time Complexity : O(n)



```
C:\Users\POOJA\Documents\SDE\DSA_Practice6>javac BinarySearchTree.java

C:\Users\POOJA\Documents\SDE\DSA_Practice6>java BinarySearchTree
Enter the number of nodes:
5
Enter the node values:
4 2 5 1 3
Enter the parent-child relationships (parentIndex, childIndex, 'L' or 'R'):
0 1 l
0 2 R
1 3 L
1 4 R
True
```

8.Word ladder

```
import java.util.*;


class ShortestChain {


    static int shortestChainLen(String start, String target, Set<String> D) {

        if (start.equals(target)) return 0;

        if (!D.contains(target)) return 0;
```

```
int level = 0, wordLength = start.length();
Queue<String> Q = new LinkedList<>();
Q.add(start);

while (!Q.isEmpty()) {
    ++level;
    int sizeOfQ = Q.size();

    for (int i = 0; i < sizeOfQ; ++i) {
        char[] word = Q.peek().toCharArray();
        Q.remove();

        for (int pos = 0; pos < wordLength; ++pos) {
            char origChar = word[pos];

            for (char c = 'a'; c <= 'z'; ++c) {
                word[pos] = c;

                if (String.valueOf(word).equals(target)) return level + 1;
                if (!D.contains(String.valueOf(word))) continue;

                D.remove(String.valueOf(word));
                Q.add(String.valueOf(word));
            }

            word[pos] = origChar;
        }
```

```
            }
        }

        return 0;
    }

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

        System.out.println("Enter the start word:");
        String start = scanner.next();

        System.out.println("Enter the target word:");
        String target = scanner.next();

        System.out.println("Enter the number of words in the dictionary:");
        int n = scanner.nextInt();
        Set<String> D = new HashSet<>();

        System.out.println("Enter the words in the dictionary:");
        for (int i = 0; i < n; i++) {
            D.add(scanner.next());
        }

        System.out.println("Length of shortest chain is: " + shortestChainLen(start, target, D));
    }
}
```

Time Complexity : $O(N^2 * M)$

9.Word ladder -II

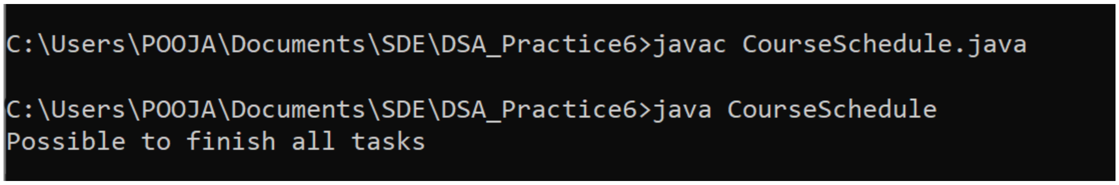
```
import java.util.*;


public class WordLadder2 {
    public List<List<String>> findLadders(String beginWord, String endWord, List<String> wordList) {
        Map<String, Integer> hm = new HashMap<>();
        List<List<String>> res = new ArrayList<>();


        Queue<String> q = new LinkedList<String>(); // Explicit type specified
        q.add(beginWord);
        hm.put(beginWord, 1);


        HashSet<String> hs = new HashSet<>();
        for (String w : wordList) hs.add(w);
        hs.remove(beginWord);


        while (!q.isEmpty()) {
            String word = q.poll();
            if (word.equals(endWord)) {
                break;
            }


            for (int i = 0; i < word.length(); i++) {
                int level = hm.get(word);
                for (char ch = 'a'; ch <= 'z'; ch++) {
                    char[] replaceChars = word.toCharArray();
                    replaceChars[i] = ch;
                    String replaceString = new String(replaceChars);

                    if (hs.contains(replaceString)) {
                        q.add(replaceString);
                        hm.put(replaceString, level + 1);
                        hs.remove(replaceString);
                    }
                }
            }
        }

        if (hm.containsKey(endWord)) {
            List<String> seq = new ArrayList<>();
            seq.add(endWord);
            dfs(endWord, seq, res, beginWord, hm);
        }
        return res;
    }

    public void dfs(String word, List<String> seq, List<List<String>> res, String beginWord, Map<String, Integer> hm) {
        if (word.equals(beginWord)) {
            List<String> ref = new ArrayList<>(seq);
            Collections.reverse(ref);
            res.add(ref);
            return;
        }
        int level = hm.get(word);
        for (int i = 0; i < word.length(); i++) {
            for (char ch = 'a'; ch <= 'z'; ch++) {
                char[] replaceChars = word.toCharArray();
                replaceChars[i] = ch;
                String replaceStr = new String(replaceChars);

                if (hm.containsKey(replaceStr) && hm.get(replaceStr) == level - 1) {
                    seq.add(replaceStr);
                    dfs(replaceStr, seq, res, beginWord, hm);
                    seq.remove(seq.size() - 1);
                }
            }
        }
    }

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

        System.out.println("Enter the start word:");
        String beginWord = sc.next();

        System.out.println("Enter the end word:");
        String endWord = sc.next();

        System.out.println("Enter the number of words in the word list:");
        int n = sc.nextInt();
```

```
        List<String> wordList = new ArrayList<>();
        System.out.println("Enter the words in the word list:");
        for (int i = 0; i < n; i++) {
            wordList.add(sc.next());
        }

        WordLadder2 wl = new WordLadder2();
        List<List<String>> result = wl.findLadders(beginWord, endWord, wordList);

        System.out.println("Shortest transformation sequences:");
        for (List<String> seq : result) {
            System.out.println(seq);
        }

        sc.close();
    }
}
```

10. Course schedule

```
import java.util.*;

public class CourseSchedule {

    static class pair {
        int first, second;

        pair(int first, int second) {
            this.first = first;
            this.second = second;
        }
    }

    static ArrayList<ArrayList<Integer>> make_graph(int numTasks, Vector<pair> prerequisites) {
        ArrayList<ArrayList<Integer>> graph = new ArrayList<>(numTasks);
        for (int i = 0; i < numTasks; i++) {
            graph.add(new ArrayList<>());
        }
        for (pair pre : prerequisites) {
            graph.get(pre.second).add(pre.first);
        }
        return graph;
    }

    static boolean dfs_cycle(ArrayList<ArrayList<Integer>> graph, int node, boolean onpath[], boolean visited[]) {
        if (visited[node]) return false;
        onpath[node] = visited[node] = true;
        for (int neigh : graph.get(node)) {
            if (onpath[neigh] || dfs_cycle(graph, neigh, onpath, visited)) return true;
        }
        return onpath[node] = false;
    }

    static boolean canFinish(int numTasks, Vector<pair> prerequisites) {
        ArrayList<ArrayList<Integer>> graph = make_graph(numTasks, prerequisites);
        boolean onpath[] = new boolean[numTasks];
        boolean visited[] = new boolean[numTasks];
        for (int i = 0; i < numTasks; i++) {
            if (!visited[i] && dfs_cycle(graph, i, onpath, visited)) return false;
        }
        return true;
    }

    public static void main(String args[]) {
        int numTasks = 4;
        Vector<pair> prerequisites = new Vector<>();
        prerequisites.add(new pair(1, 0));
        prerequisites.add(new pair(2, 1));
        prerequisites.add(new pair(3, 2));
        if (canFinish(numTasks, prerequisites)) {
            System.out.println("Possible to finish all tasks");
        } else {
            System.out.println("Impossible to finish all tasks");
        }
    }
}
```

Time Complexity : O(V+E)



```
C:\Users\POOJA\Documents\SDE\DSA_Practice6>javac CourseSchedule.java

C:\Users\POOJA\Documents\SDE\DSA_Practice6>java CourseSchedule
Possible to finish all tasks
```

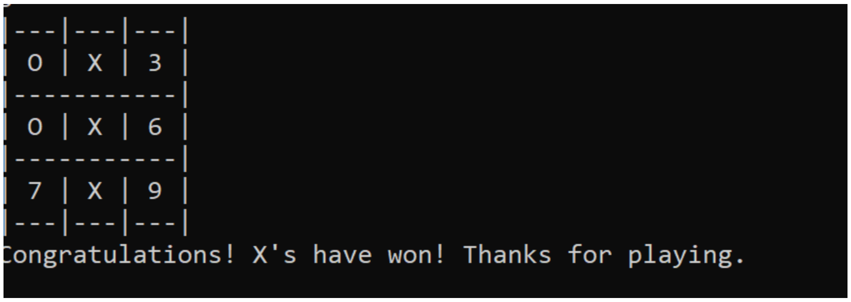
11.Design tic tac toe

```
import java.util.*;
```

```
public class TicTacToe {

    static String[] board;
    static String turn;

    static String checkWinner() {
        for (int a = 0; a < 8; a++) {
            String line = null;
            switch (a) {
                case 0 -> line = board[0] + board[1] + board[2];
                case 1 -> line = board[3] + board[4] + board[5];
                case 2 -> line = board[6] + board[7] + board[8];
                case 3 -> line = board[0] + board[3] + board[6];
                case 4 -> line = board[1] + board[4] + board[7];
                case 5 -> line = board[2] + board[5] + board[8];
                case 6 -> line = board[0] + board[4] + board[8];
                case 7 -> line = board[2] + board[4] + board[6];
            }
            if (line.equals("XXX")) return "X";
            if (line.equals("OOO")) return "O";
        }
        for (int a = 0; a < 9; a++) {
            if (Arrays.asList(board).contains(String.valueOf(a + 1))) break;
            else if (a == 8) return "draw";
        }
        System.out.println(turn + "'s turn; enter a slot number to place " + turn + " in:");
        return null;
```

```
}

static void printBoard() {
    System.out.println("|---|---|---|");
    System.out.println("| " + board[0] + " | " + board[1] + " | " + board[2] + " |");
    System.out.println("|-----------|");
    System.out.println("| " + board[3] + " | " + board[4] + " | " + board[5] + " |");
    System.out.println("|-----------|");
    System.out.println("| " + board[6] + " | " + board[7] + " | " + board[8] + " |");
    System.out.println("|---|---|---|");
}

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    board = new String[9];
    turn = "X";
    String winner = null;

    for (int a = 0; a < 9; a++) board[a] = String.valueOf(a + 1);

    System.out.println("Welcome to 3x3 Tic Tac Toe.");
    printBoard();
    System.out.println("X will play first. Enter a slot number to place X in:");

    while (winner == null) {
        int numInput;
        try {
            numInput = in.nextInt();
```

```
                if (!(numInput > 0 && numInput <= 9)) {
                    System.out.println("Invalid input; re-enter slot number:");
                    continue;
                }
            } catch (InputMismatchException e) {
                System.out.println("Invalid input; re-enter slot number:");
                in.next();
                continue;
            }
            if (board[numInput - 1].equals(String.valueOf(numInput))) {
                board[numInput - 1] = turn;
                turn = turn.equals("X") ? "O" : "X";
                printBoard();
                winner = checkWinner();
            } else {
                System.out.println("Slot already taken; re-enter slot number:");
            }
        }

        if (winner.equalsIgnoreCase("draw")) {
            System.out.println("It's a draw! Thanks for playing.");
        } else {
            System.out.println("Congratulations! " + winner + "'s have won! Thanks for playing.");
        }
        in.close();
    }
}
```



```
|---|---|---|
| O | X | 3 |
|-----------|
| O | X | 6 |
|-----------|
| 7 | X | 9 |
|---|---|---|
Congratulations! X's have won! Thanks for playing.
```