LAB MANUAL

Operation Research (OR)

2301CS729

B.Tech/Bsc(H) 5th Semester

A.Y. 2025-2026

(Darshan Institute of Engineering and Technology)

Darshan
UNIVERSITY

योग: कर्मसु कौशलम्

Problem 1. List out the applications of Operation Research in IT industry and formulate the Linear Programming Problem for any one real-world problem.

Problem 1: Solution

Applications of Operation Research in IT Industry:

1. **Resource Allocation** – Efficient use of hardware, bandwidth, and software licenses.
2. **Project Scheduling** – Planning tasks and timelines using techniques like PERT and CPM.
3. **Network Design** – Optimizing data flow, routing, and load balancing.
4. **Cloud Computing** – Managing server loads to minimize cost and maximize performance.
5. **Software Testing** – Scheduling test cases and assigning tasks to reduce testing time.
6. **Data Storage** – Allocating storage efficiently across distributed systems.
7. **Risk Management** – Assessing uncertainties and minimizing expected losses.

Linear Programming Formulation (Example: Cloud Server Load Balancing)

Objective:

Minimize the total processing time by assigning tasks to servers.

Decision Variables:

Let x1,x2x_1, x_2x1,x2 be the number of tasks assigned to Server 1 and Server 2.

Objective Function:

Minimize Z=5x1+3x2Z = 5x_1 + 3x_2Z=5x1+3x2

(where 5 and 3 are the processing times per task on servers 1 and 2)

Constraints:

1. x1+x2=50x_1 + x_2 = 50x1+x2=50 (Total tasks to assign)
2. x1≤30x_1 \leq 30x1≤30 (Server 1 capacity)
3. x2≤40x_2 \leq 40x2≤40 (Server 2 capacity)
4. x1,x2≥0x_1, x_2 \geq 0x1,x2≥0

This LP problem helps in distributing tasks efficiently while minimizing processing time.

Problem 1. Write a program for the given maximization Problem (Using Brut force method).

Maximize:

$Z=3x_1+2x_2$

Subject to: x1+x2 <= 4 and x1>=0,x2>=0

We need to find values of x1and x2 that maximize Z while satisfying the constraints.

Problem 1: Solution

```java
public class MaximizeZBruteForce {
    public static void main(String[] args) {
        int maxZ = Integer.MIN_VALUE;
        int bestX1 = 0;
        int bestX2 = 0;

        for (int x1 = 0; x1 <= 4; x1++) {
            for (int x2 = 0; x2 <= 4; x2++) {
                // Check constraint x1 + x2 <= 4
                if (x1 + x2 <= 4) {
                    int z = 3 * x1 + 2 * x2;
                    if (z > maxZ) {
                        maxZ = z;
                        bestX1 = x1;
                        bestX2 = x2;
                    }
                }
            }
        }

        System.out.println("Optimal solution found:");
        System.out.println("x1 = " + bestX1);
        System.out.println("x2 = " + bestX2);
        System.out.println("Maximum Z = " + maxZ);
    }
}
```

Problem 1: Output

```
Optimal solution found:
x1 = 4
x2 = 0
Maximum Z = 12
```

Problem 2: **Implement a solution for the following problem using Brut force method.** Distribute workloads across multiple servers to minimize processing time.

Objective: Distribute tasks across servers to minimize the **maximum processing time** on any server.

Example Scenario:

- 3 tasks with processing times: 10, 20, 30
- 2 servers

We want to assign tasks to servers such that the load is balanced (i.e., no server is overloaded). Hint: Approach (Simplified Brute Force).

Problem 2: Solution

```java
import java.util.*;

public class TaskDistributionBruteForce {

    static int[] tasks = {10, 20, 30};
    static int serversCount = 2;
    static int minMaxLoad = Integer.MAX_VALUE;
    static int[][] bestAssignment;

    public static void main(String[] args) {
        int n = tasks.length;
        int[] assignment = new int[n];

        bestAssignment = new int[n][serversCount];

        // Start recursive brute-force exploration
        assignTasks(0, assignment);

        // Output the best result
        System.out.println("Minimum maximum load: " + minMaxLoad);
        System.out.println("Best assignment of tasks:");
        for (int s = 0; s < serversCount; s++) {
            System.out.print("Server " + (s + 1) + ": ");
            for (int t = 0; t < tasks.length; t++) {
                if (bestAssignment[t][s] == 1) {
                    System.out.print("Task " + tasks[t] + " ");
                }
            }
            System.out.println();
        }
    }

    // Recursive method to assign tasks
    static void assignTasks(int taskIndex, int[] assignment) {
        if (taskIndex == tasks.length) {
            // All tasks assigned, check this configuration
            int[] loads = new int[serversCount];
            for (int i = 0; i < tasks.length; i++) {
                loads[assignment[i]] += tasks[i];
            }
            int currentMaxLoad = Arrays.stream(loads).max().getAsInt();

            if (currentMaxLoad < minMaxLoad) {
                minMaxLoad = currentMaxLoad;
                // Store the assignment
                for (int i = 0; i < tasks.length; i++) {
                    Arrays.fill(bestAssignment[i], 0);
                    bestAssignment[i][assignment[i]] = 1;
                }
            }
            return;
        }

        // Try assigning current task to each server
        for (int server = 0; server < serversCount; server++) {
            assignment[taskIndex] = server;
            assignTasks(taskIndex + 1, assignment);
        }
    }
}

```

Problem 2: Output

```
Minimum maximum load: 30
Best assignment of tasks:
Server 1: Task 10 Task 20
Server 2: Task 30
```

Problem 1: Implement a solution for the following problem using simplex method.

A factory makes **Product A** and **Product B.**

- Each unit of A needs:
  - 1 hours of machine time
  - 2 hours of labor
- Each unit of B needs:
  - 2 hours of machine time
  - 1 hour of labor

The factory has:

- **100 machine hours** available
- **100 labor hours** available

Profit:

- A earns ₹30 per unit
- B earns ₹20 per unit

Goal: Find how many units of A and B to make to **maximize profit**, without exceeding the resource limits.

Problem 1: Solution

```
import java.util.*;

public class Simp {

    public static void main(String[] args) {
        double[][] tableau = {
            {1, 2, 1, 0, 100},
            {2, 1, 0, 1, 100}
        };

        int[] cj = {30, 20, 0, 0, 0};
        int[] cb = {0, 0};
        int[] basicVar = {2, 3};

        int[] zj = new int[5];
        int[] cjmzj = new int[5];
        double[] ratio = new double[2];

        while (true) {
            zj = calculateZj(new int[5], cb, tableau);
            cjmzj = calculateCjMinusZj(new int[5], cj, zj);

            printRow("Zj:      ", zj);
            printRow("Cj - Zj: ", cjmzj);

            if (isOptimal(cjmzj)) {
                System.out.println("Optimal solution reached.");
                break;
            }

            int keyCol = findKeyColumn(cjmzj);
            int keyRow = findKeyRow(tableau, keyCol, ratio);

            if (keyRow == -1) {
                System.out.println("Unbounded solution.");
                return;
            }

            cb[keyRow] = cj[keyCol];
            basicVar[keyRow] = keyCol;

            performPivot(tableau, keyRow, keyCol);

            System.out.println("Updated Tableau:");
            printTableau(tableau);
        }

        printFinalSolution(tableau, basicVar, cj);
    }

    public static int[] calculateZj(int[] zj, int[] cb, double[][] tableau) {
        for (int j = 0; j < tableau[0].length; j++) {
            zj[j] = 0;
            for (int i = 0; i < tableau.length; i++) {
                zj[j] += cb[i] * tableau[i][j];
            }
        }
        return zj;
    }

    public static int[] calculateCjMinusZj(int[] cjmzj, int[] cj, int[] zj) {
        for (int i = 0; i < cj.length; i++) {
            cjmzj[i] = cj[i] - zj[i];
        }
        return cjmzj;
    }

    public static boolean isOptimal(int[] cjmzj) {
        for (int i = 0; i < cjmzj.length - 1; i++) {
            if (cjmzj[i] > 0) return false;
        }
        return true;
    }

    public static int findKeyColumn(int[] cjmzj) {
        int max = 0, keyCol = -1;
        for (int i = 0; i < cjmzj.length - 1; i++) {
            if (cjmzj[i] > max) {
                max = cjmzj[i];
                keyCol = i;
            }
        }
        return keyCol;
    }

    public static int findKeyRow(double[][] tableau, int keyCol, double[] ratio)
{
        int keyRow = -1;
        double min = Double.MAX_VALUE;
        int rhsCol = tableau[0].length - 1;

        for (int i = 0; i < tableau.length; i++) {
            if (tableau[i][keyCol] > 0) {
                ratio[i] = tableau[i][rhsCol] / tableau[i][keyCol];
                if (ratio[i] < min) {
                    min = ratio[i];
                    keyRow = i;
                }
            } else {
                ratio[i] = Double.MAX_VALUE;
            }
        }
        return keyRow;
    }

    public static void performPivot(double[][] tableau, int keyRow, int keyCol)
{
        double pivot = tableau[keyRow][keyCol];
        for (int j = 0; j < tableau[0].length; j++) {
            tableau[keyRow][j] /= pivot;
        }

        for (int i = 0; i < tableau.length; i++) {
            if (i != keyRow) {
                double factor = tableau[i][keyCol];
                for (int j = 0; j < tableau[0].length; j++) {
                    tableau[i][j] -= factor * tableau[keyRow][j];
                }
            }
        }
    }
```

```java

    public static void printRow(String label, int[] row) {
        System.out.print(label);
        for (int val : row) {
            System.out.printf("%6d", val);
        }
        System.out.println();
    }

    public static void printTableau(double[][] tableau) {
        System.out.println("Tableau:");
        for (double[] row : tableau) {
            for (double val : row) {
                System.out.printf("%8.2f", val);
            }
            System.out.println();
        }
        System.out.println();
    }

    public static void printFinalSolution(double[][] tableau, int[] basicVar,
int[] cj) {
        int varCount = cj.length - 1;
        double[] solution = new double[varCount];
        int rhsCol = tableau[0].length - 1;

        for (int i = 0; i < tableau.length; i++) {
            if (basicVar[i] < varCount) {
                solution[basicVar[i]] = tableau[i][rhsCol];
            }
        }

        System.out.println("Final Solution:");
        for (int i = 0; i < varCount; i++) {
            System.out.printf("x%d = %.2f\n", i + 1, solution[i]);
        }

        double Z = 0;
        for (int i = 0; i < tableau.length; i++) {
            Z += cb[i] * tableau[i][rhsCol];
        }

        System.out.printf("Maximum Profit Z = ₹%.2f\n", Z);
    }
}
```

Problem 1: Output

```
Zj:          0     0     0     0     0
Cj - Zj:    30    20     0     0     0
Updated Tableau:
    1.00    2.00    1.00    0.00  100.00
    0.50   -1.00    0.00    1.00   50.00

Zj:         15   -20     0     0    50
Cj - Zj:    15    40     0     0   -50
Updated Tableau:
    1.00    0.00    1.00    2.00  200.00
    0.50    1.00    0.00   -1.00   50.00

Zj:         40    20     0     0   100
Cj - Zj:   -10     0     0     0  -100
Optimal solution reached.

Final Solution:
x1 = 40.00
x2 = 30.00
Maximum Profit Z = ₹1800.00
```

Problem 1: Implement a solution for the following problem using North West Corner method find a minimum cost.

A company has: 2 factories (suppliers) with supplies: S1 = 20, S2 = 30, 3 warehouses (consumers) with demands: D1 = 10, D2 = 25, D3 = 15

Transportation costs per unit:

	D1	D2	D3
S1	8	6	10
S2	9	7	4

Note: Students are suggested to solve the above problem using Vogel's Approximation Method.

Problem 1: Solution

```java
public class NorthWestCorner {

    public static void main(String[] args) {
        int[] supply = {20, 30};
        int[] demand = {10, 25, 15};

        int[][] cost = {
            {8, 6, 10},
            {9, 7, 4}
        };

        int[][] allocation = new int[2][3];

        int i = 0, j = 0;
        int[] supplyLeft = supply.clone();
        int[] demandLeft = demand.clone();

        while (i < supply.length && j < demand.length) {
            int alloc = Math.min(supplyLeft[i], demandLeft[j]);
            allocation[i][j] = alloc;
            supplyLeft[i] -= alloc;
            demandLeft[j] -= alloc;

            if (supplyLeft[i] == 0 && i < supply.length - 1) {
                i++;
            } else if (demandLeft[j] == 0 && j < demand.length - 1) {
                j++;
            } else if (supplyLeft[i] == 0 && demandLeft[j] == 0) {
                if (i < supply.length - 1) {
                    i++;
                } else if (j < demand.length - 1) {
                    j++;
                }
            }
        }

        // Calculate total cost
        int totalCost = 0;
        System.out.println("Allocation:");
        for (i = 0; i < supply.length; i++) {
            for (j = 0; j < demand.length; j++) {
                System.out.printf("%4d", allocation[i][j]);
                totalCost += allocation[i][j] * cost[i][j];
            }
            System.out.println();
        }

        System.out.println("Total Transportation Cost = " + totalCost);
    }
}
```

Problem 1: Output

```
Allocation:
  10  10   0
   0  15  15
Total Transportation Cost = 415
```

Problem 1: **Implement a solution for the following problem using Assignment Problem (Hungarian method).**

Let's say there are **3 bugs** and **3 developers.** The following table shows the **effort (cost)** of each developer fixing a particular bug:

Developer / Bug	Bug 1	Bug 2	Bug 3
Developer 1	4	2	3
Developer 2	2	5	6
Developer 3	3	7	1

We want to assign the bugs to developers in such a way that the total effort (or cost) is minimized.

Problem 1: Solution

```java
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Scanner;

public class Assignment {

    public static void main(String[] args) {
        int assignment_count = 4, assignee_count = 4;
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter number of assignments: ");
        assignment_count = sc.nextInt();

        System.out.println("Enter number of assignees: ");
        assignee_count = sc.nextInt();

        if (assignee_count > assignment_count) {
            System.out.println("Not enough assignments to assign all assignees
to.");
            sc.close();
            return;
        }

        if (assignee_count < assignment_count) {
            System.out.println("Not enough assignees to assign all
assignments.");
        }

        int[][] unit_cost = takeArray(sc, assignee_count, assignment_count);
        sc.close();

        int[][] assignment_matrix = copyOf2D(unit_cost);

        ArrayList<int[]> assigned = makeAssignment(assignment_matrix);

        int cost = 0;
        for (int[] pos : assigned) {
            cost += unit_cost[pos[0]][pos[1]];
        }

        System.out.println("Minimum assignment cost: " + cost);
        System.out.println("Assignments:");
        for (int[] pos : assigned) {
            System.out.println("Assignee " + pos[0] + " -> Assignment " + pos[1]
+
                               " (Cost: " + unit_cost[pos[0]][pos[1]] + ")");
        }
    }

    public static ArrayList<int[]> makeAssignment(int[][] arr) {
        for (int i = 0; i < arr.length; i++) {
            int min = arr[i][0];
            for (int j = 1; j < arr[0].length; j++) {
                if (arr[i][j] < min)
                    min = arr[i][j];
            }

            if (min != 0) {
                for (int j = 0; j < arr[0].length; j++) {
                    arr[i][j] -= min;
                }
            }
        }

        for (int j = 0; j < arr[0].length; j++) {
            int min = arr[0][j];
            for (int i = 1; i < arr.length; i++) {
                if (arr[i][j] < min)
                    min = arr[i][j];
            }

            for (int i = 0; i < arr.length; i++) {
                arr[i][j] -= min;
            }
        }

        ArrayList<int[]> assigned = assignZeros(arr);
        while (assigned.size() != arr.length) {
            revisedMatrix(arr, assigned);
```

```java
            assigned = assignZeros(arr);
        }

        return assigned;
    }

    public static void revisedMatrix(int[][] arr, ArrayList<int[]> assigned) {
        boolean[] checkedRow = new boolean[arr.length];
        boolean[] checkedCol = new boolean[arr[0].length];

        for (int i = 0; i < arr.length; i++) {
            if (checkedRow[i])
                continue;
            boolean found = false;
            for (int j = 0; j < arr[0].length; j++) {
                if (isAssigned(i, j, assigned)) {
                    found = true;
                    break;
                }
            }
            if (!found) {
                checkRow(arr, i, checkedRow, checkedCol, assigned);
            }
        }

        int min = Integer.MAX_VALUE;
        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr[0].length; j++) {
                if (checkedRow[i] && !checkedCol[j]) {
                    if (arr[i][j] < min) {
                        min = arr[i][j];
                    }
                }
            }
        }

        if (min == Integer.MAX_VALUE) {
            System.out.println("Couldn't find valid assignments.");
            return;
        }

        for (int i = 0; i < arr.length; i++) {
            for (int j = 0; j < arr[0].length; j++) {
                if (!checkedRow[i] && checkedCol[j]) {
                    arr[i][j] += min;
                } else if (checkedRow[i] && !checkedCol[j]) {
                    arr[i][j] -= min;
                }
            }
            }
        }

    public static void checkRow(int[][] arr, int row, boolean[] checkedRow,
boolean[] checkedCol, ArrayList<int[]> assigned) {
        for (int j = 0; j < arr[0].length; j++) {
            if (!checkedCol[j] && arr[row][j] == 0) {
                checkedCol[j] = true;
                checkCol(arr, j, checkedRow, checkedCol, assigned);
            }
        }
    }

    public static void checkCol(int[][] arr, int col, boolean[] checkedRow,
boolean[] checkedCol, ArrayList<int[]> assigned) {
        for (int i = 0; i < arr.length; i++) {
            if (!checkedRow[i] && arr[i][col] == 0 && isAssigned(i, col,
assigned)) {
                checkedRow[i] = true;
                checkRow(arr, i, checkedRow, checkedCol, assigned);
            }
        }
    }

    public static ArrayList<int[]> assignZeros(int[][] arr) {
        ArrayList<int[]> assigned = new ArrayList<>();
        boolean[] rowAssigned = new boolean[arr.length];
        boolean[] colAssigned = new boolean[arr[0].length];

        for (int i = 0; i < arr.length; i++) {
            int count = 0;
            int zeroInd = -1;
            for (int j = 0; j < arr[0].length; j++) {
                if (arr[i][j] == 0 && !colAssigned[j]) {
                    count++;
                    zeroInd = j;
                }
            }
            if (count == 1 && !rowAssigned[i]) {
                assigned.add(new int[]{i, zeroInd});
                rowAssigned[i] = true;
                colAssigned[zeroInd] = true;
            }
        }

        for (int j = 0; j < arr[0].length; j++) {
            int count = 0;
            int zeroInd = -1;
            for (int i = 0; i < arr.length; i++) {
                if (arr[i][j] == 0 && !rowAssigned[i]) {
                    count++;
                    zeroInd = i;
                }
            }
            if (count == 1 && !colAssigned[j]) {
                assigned.add(new int[]{zeroInd, j});
                rowAssigned[zeroInd] = true;
                colAssigned[j] = true;
            }
        }

        for (int i = 0; i < arr.length; i++) {
            if (rowAssigned[i]) continue;
            for (int j = 0; j < arr[0].length; j++) {
                if (arr[i][j] == 0 && !colAssigned[j]) {
                    assigned.add(new int[]{i, j});
                    rowAssigned[i] = true;
                    colAssigned[j] = true;
                    break;
                }
            }
        }

        return assigned;
    }

    public static boolean isCrossed(int i, int j, ArrayList<int[]> list) {
        for (int[] temp : list) {
            if (temp[0] == i || temp[1] == j) return true;
        }
        return false;
    }

    public static boolean isAssigned(int i, int j, ArrayList<int[]> list) {
        for (int[] temp : list) {
            if (temp[0] == i && temp[1] == j) return true;
        }
        return false;
    }

    public static int[][] copyOf2D(int[][] arr) {
        int[][] copy = new int[arr.length][arr[0].length];
        for (int i = 0; i < arr.length; i++) {
            System.arraycopy(arr[i], 0, copy[i], 0, arr[0].length);
        }
        return copy;
    }
```

```java
    public static int[][] takeArray(Scanner sc, int m, int n) {
        int[][] arr = new int[m][n];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print("Enter unit cost for assignee " + i + " to 
complete assignment " + j + ": ");
                arr[i][j] = sc.nextInt();
            }
            System.out.println();
        }
        return arr;
    }
}
```

Problem 1: Output

```
Enter number of assignments:
3
Enter number of assignees:
3
Enter unit cost for assignee 0 to complete assignment 0: 4
Enter unit cost for assignee 0 to complete assignment 1: 2
Enter unit cost for assignee 0 to complete assignment 2: 3

Enter unit cost for assignee 1 to complete assignment 0: 2
Enter unit cost for assignee 1 to complete assignment 1: 5
Enter unit cost for assignee 1 to complete assignment 2: 6

Enter unit cost for assignee 2 to complete assignment 0: 3
Enter unit cost for assignee 2 to complete assignment 1: 7
Enter unit cost for assignee 2 to complete assignment 2: 1

Minimum assignment cost: 5
Assignments:
Assignee 0 -> Assignment 1 (Cost: 2)
Assignee 1 -> Assignment 0 (Cost: 2)
Assignee 2 -> Assignment 2 (Cost: 1)
```

Problem 1: Implement a solution for the following problem using Assignment Problem. The IT Project Team Formation problem involves assigning the right skills to teams to maximize the project's overall performance. The challenge is to allocate team members to various tasks based on their skills in such a way that the project's performance is maximized.

This problem can be formulated as an assignment problem where:

The tasks are the "jobs" that need to be done.

The team members are the "agents" who can perform these tasks.

The performance (effort or cost) of each team member performing a task is given by a matrix.

The goal is to assign each team member to the right task in such a way that the overall project performance is maximized.

Input: The input matrix represents the **performance** of each team member on each task. The rows represent team members, and the columns represent tasks. Each element in the matrix shows the performance score of a team member on a specific task.

Problem 1: Solution

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

public class ITProjectTeamFormation {

    private static final int SIZE = 4;

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

        System.out.println("Enter the number of team members/tasks (square
matrix size): ");
        int n = sc.nextInt();

        int[][] performance = new int[n][n];
        System.out.println("Enter the performance matrix (row-wise): ");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                performance[i][j] = sc.nextInt();
            }
        }
        sc.close();

        int maxValue = findMax(performance);
        int[][] cost = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                cost[i][j] = maxValue - performance[i][j];
            }
        }

        Hungarian hungarian = new Hungarian(n, cost);
        int totalCost = hungarian.solve();

        int totalPerformance = 0;
        System.out.println("\nAssignments:");
        for (int i = 0; i < n; i++) {
            int j = hungarian.assignment[i];
            totalPerformance += performance[i][j];
            System.out.println("Team Member " + (i + 1) + " assigned to Task " +
(j + 1) +
                    " (Performance: " + performance[i][j] + ")");
        }

        System.out.println("\nTotal Maximum Performance: " + totalPerformance);
    }

    private static int findMax(int[][] matrix) {
        int max = Integer.MIN_VALUE;
        for (int[] row : matrix) {
            for (int val : row) {
                if (val > max) max = val;
            }
        }
        return max;
    }
}

class Hungarian {
    int n;
    int[][] cost;
    int[] assignment;

    public Hungarian(int n, int[][] cost) {
        this.n = n;
        this.cost = cost;
        this.assignment = new int[n];
        Arrays.fill(this.assignment, -1);
    }

    public int solve() {
        int[] u = new int[n + 1];
        int[] v = new int[n + 1];
        int[] p = new int[n + 1];
        int[] way = new int[n + 1];

        for (int i = 1; i <= n; i++) {
            p[0] = i;
            int[] minv = new int[n + 1];
            boolean[] used = new boolean[n + 1];
            Arrays.fill(minv, Integer.MAX_VALUE);

            int j0 = 0;
            do {
                used[j0] = true;
                int i0 = p[j0];
                int delta = Integer.MAX_VALUE;
                int j1 = -1;
                for (int j = 1; j <= n; j++) {
                    if (!used[j]) {
                        int cur = cost[i0 - 1][j - 1] - u[i0] - v[j];
                        if (cur < minv[j]) {
                            minv[j] = cur;
                            way[j] = j0;
                        }
                        if (minv[j] < delta) {
                            delta = minv[j];
                            j1 = j;
                        }
                    }
                }

                for (int j = 0; j <= n; j++) {
                    if (used[j]) {
                        u[p[j]] += delta;
                        v[j] -= delta;
                    } else {
                        minv[j] -= delta;
                    }
                }
                j0 = j1;
            } while (p[j0] != 0);

            do {
                int j1 = way[j0];
                p[j0] = p[j1];
                j0 = j1;
            } while (j0 != 0);
        }

        for (int j = 1; j <= n; j++) {
            assignment[p[j] - 1] = j - 1;
        }
        return -v[0];
    }
}
```

Problem 1: Output

```
Enter the number of team members/tasks (square matrix size):
3
Enter the performance matrix (row-wise):
10 2 8
7 5 9
3 12 4

Assignments:
Team Member 1 assigned to Task 1 (Performance: 10)
Team Member 2 assigned to Task 3 (Performance: 9)
Team Member 3 assigned to Task 2 (Performance: 12)

Total Maximum Performance: 31
```

Problem 1: Implement a solution for CPU Scheduling Problem: Minimizing Waiting and Turnaround Time

You are given multiple processes with their burst times (execution times). Your task is to assign CPU time to each process so that the average waiting time and average turnaround time are minimized.

Hint:

Implement a SJF algorithm

Burst Time: Time required by a process for execution.

Waiting Time: Time a process waits in the ready queue.

Waiting Time=Turnaround Time−Burst Time

Turnaround Time: Total time taken from arrival to completion.

Turnaround Time=Completion Time−Arrival Time

Problem 1: Solution

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

class Process {
    int id;
    int burstTime;
    int waitingTime;
    int turnaroundTime;

    public Process(int id, int burstTime) {
        this.id = id;
        this.burstTime = burstTime;
    }
}

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

        System.out.println("Enter the number of processes:");
        int n = sc.nextInt();

        Process[] processes = new Process[n];

        System.out.println("Enter burst time for each process:");
        for (int i = 0; i < n; i++) {
            System.out.print("Process " + (i + 1) + ": ");
            int burst = sc.nextInt();
            processes[i] = new Process(i + 1, burst);
        }
        sc.close();
        Arrays.sort(processes, (p1, p2) -> Integer.compare(p1.burstTime,
p2.burstTime));

        int totalWaitingTime = 0;
        int totalTurnaroundTime = 0;
        int currentTime = 0;

        for (Process process : processes) {
            process.waitingTime = currentTime;
            process.turnaroundTime = process.waitingTime + process.burstTime;
            currentTime += process.burstTime;

            totalWaitingTime += process.waitingTime;
            totalTurnaroundTime += process.turnaroundTime;
        }
        System.out.println("\nProcess\tBurst Time\tWaiting Time\tTurnaround
Time");
        for (Process process : processes) {
            System.out.println(process.id + "\t\t" + process.burstTime + "\t\t"
+
                               process.waitingTime + "\t\t" +
process.turnaroundTime);
        }
        double avgWaitingTime = (double) totalWaitingTime / n;
        double avgTurnaroundTime = (double) totalTurnaroundTime / n;
        System.out.printf("\nAverage Waiting Time: %.2f\n", avgWaitingTime);
        System.out.printf("Average Turnaround Time: %.2f\n", avgTurnaroundTime);
    }
}
```

Problem 1: Output

```
Enter the number of processes:
4
Enter burst time for each process:
Process 1: 6
Process 2: 8
Process 3: 7
Process 4: 3

Process  Burst Time    Waiting Time   Turnaround Time
4             3             0             3
1             6             3             9
3             7             9             16
2             8             16            24

Average Waiting Time: 7.00
Average Turnaround Time: 13.00
```

Problem 1: **Implement Two-Person Zero-Sum Game using a Saddle Point (Pure Strategy)**

In Game Theory, a Two-Person Zero-Sum Game is a situation where:

Two players (Player A and Player B) play a game.

The gain of one player is the loss of the other.

The sum of gains and losses is always zero.

The goal is to find an optimal strategy for each player such that the expected payoff for both players is maximized or minimized depending on their role (row or column player).

A simple way to solve such games is by checking if a saddle point exists:

A saddle point is the element that is minimum in its row and maximum in its column.

If a saddle point exists, the game has a pure strategy solution and find it's solution Otherwise, find a solution using arithmetic/algebraic method

Input: 2 X 2 matrix

Output: Saddle point value / find the value of game (maximum winning)

Problem 1: Solution

```java
import java.util.Scanner;

public class Practical8 {

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

        int rows = 2;
        int cols = 2;
        int[][] matrix = new int[rows][cols];

        System.out.println("Enter the elements of the 2x2 payoff matrix:");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                System.out.print("Element [" + (i+1) + "][" + (j+1) + "]: ");
                matrix[i][j] = sc.nextInt();
            }
        }
        sc.close();

        findSaddlePoint(matrix);
    }

    static void findSaddlePoint(int[][] matrix) {
        int rows = matrix.length, cols = matrix[0].length;

        // Find row minima
        int[] rowMin = new int[rows];
        for (int i = 0; i < rows; i++) {
            rowMin[i] = matrix[i][0];
            for (int j = 1; j < cols; j++) {
                rowMin[i] = Math.min(rowMin[i], matrix[i][j]);
            }
        }

        // Find column maxima
        int[] colMax = new int[cols];
        for (int j = 0; j < cols; j++) {
            colMax[j] = matrix[0][j];
            for (int i = 1; i < rows; i++) {
                colMax[j] = Math.max(colMax[j], matrix[i][j]);
            }
        }

        // Find max of row minima and min of column maxima
        int maxOfRowMin = rowMin[0];
        for (int i = 1; i < rows; i++) maxOfRowMin = Math.max(maxOfRowMin,
rowMin[i]);

        int minOfColMax = colMax[0];
        for (int j = 1; j < cols; j++) minOfColMax = Math.min(minOfColMax,
colMax[j]);

        if (maxOfRowMin == minOfColMax) {
            System.out.println("\nSaddle Point Found! Value of the Game = " +
maxOfRowMin);
        } else {
            System.out.println("\nNo Saddle Point. Solving using algebraic
method...");
            solve2x2(matrix);
        }
    }

    static void solve2x2(int[][] m) {
        int a = m[0][0], b = m[0][1], c = m[1][0], d = m[1][1];
        double denominator = (a - b - c + d);

        if (denominator == 0) {
            System.out.println("Game has no unique mixed strategy solution.");
            return;
        }

        double p1 = (double)(d - c) / denominator;
        double p2 = 1 - p1;

        double q1 = (double)(d - b) / denominator;
        double q2 = 1 - q1;

        double V = (a * d - b * c) / denominator;

        System.out.printf("Optimal Mixed Strategy for Player A: (%.2f, %.2f)%n",
p1, p2);
        System.out.printf("Optimal Mixed Strategy for Player B: (%.2f, %.2f)%n",
q1, q2);
        System.out.printf("Value of the Game = %.2f%n", V);
    }
}
```

Problem 1: Output

```
Enter the elements of the 2x2 payoff matrix:
Element [1][1]: 3
Element [1][2]: 2
Element [2][1]: 4
Element [2][2]: 1

Saddle Point Found! Value of the Game = 2
```

Problem 1: Implement a PERT-CPM based solution for following problem.

PERT (Program Evaluation Review Technique) and CPM (Critical Path Method) are used in project scheduling to: Estimate the minimum time required to complete a project, Identify the critical path, i.e., the longest path through the network which determines the project duration.

A project has 6 activities labeled A to F:

Activity	Duration	Predecessor
A	3	–
B	2	A
C	4	A
D	2	B, C
E	3	C
F	1	D, E

Write a program that calculates Earliest Start (ES), Earliest Finish (EF), Latest Start (LS), Latest Finish (LF), Slack Time, Critical Path

Problem 1: Solution

```java
import java.util.*;

public class Practical9_PERT_CPM {

    static class Activity {
        String id;
        int duration;
        List<String> preds = new ArrayList<>();
        List<String> succs = new ArrayList<>();

        int ES = 0, EF = 0, LS = Integer.MAX_VALUE, LF = Integer.MAX_VALUE,
slack = 0;

        Activity(String id, int duration) {
            this.id = id;
            this.duration = duration;
        }
    }

    public static void main(String[] args) {
        // Define the activities with their durations and predecessors
        Map<String, Activity> acts = new LinkedHashMap<>();
        acts.put("A", new Activity("A", 3));
        acts.put("B", new Activity("B", 2));
        acts.put("C", new Activity("C", 4));
        acts.put("D", new Activity("D", 2));
        acts.put("E", new Activity("E", 3));
        acts.put("F", new Activity("F", 1));

        acts.get("B").preds.add("A");
        acts.get("C").preds.add("A");
        acts.get("D").preds.add("B"); acts.get("D").preds.add("C");
        acts.get("E").preds.add("C");
        acts.get("F").preds.add("D"); acts.get("F").preds.add("E");

        for (Activity a : acts.values()) {
            for (String p : a.preds) {
                acts.get(p).succs.add(a.id);
            }
        }

        List<String> topo = topoSort(acts);
        if (topo == null) {
            System.out.println("Cycle detected in network; cannot perform
PERT/CPM.");
            return;
        }

        for (String id : topo) {
            Activity act = acts.get(id);
            if (act.preds.isEmpty()) {
                act.ES = 0;
            } else {
                int maxEF = 0;
                for (String p : act.preds) {
                    maxEF = Math.max(maxEF, acts.get(p).EF);
                }
                act.ES = maxEF;
            }
            act.EF = act.ES + act.duration;
        }

        int projectDuration = 0;
        for (Activity a : acts.values()) {
            projectDuration = Math.max(projectDuration, a.EF);
        }

        for (Activity a : acts.values()) {
            if (a.succs.isEmpty()) {
                a.LF = projectDuration;
                a.LS = a.LF - a.duration;
            }
        }

        List<String> revTopo = new ArrayList<>(topo);
        Collections.reverse(revTopo);
        for (String id : revTopo) {
            Activity act = acts.get(id);
            if (!act.succs.isEmpty()) {
                int minLSofSucc = Integer.MAX_VALUE;
                for (String s : act.succs) {
                    minLSofSucc = Math.min(minLSofSucc, acts.get(s).LS);
                }
                act.LF = minLSofSucc;
                act.LS = act.LF - act.duration;
            }
            act.slack = act.LS - act.ES;
        }

        System.out.println("Activity | Dur | ES | EF | LS | LF | Slack");
        System.out.println("------------------------------------------------");
        for (Activity a : acts.values()) {
            System.out.printf("   %s     |  %2d | %2d | %2d | %2d | %2d
|  %d%n",
                    a.id, a.duration, a.ES, a.EF, a.LS, a.LF, a.slack);
        }

        System.out.println("\nProject Duration = " + projectDuration);

        List<List<String>> criticalPaths = findCriticalPaths(acts);
        if (criticalPaths.isEmpty()) {
            System.out.println("\nNo critical path found.");
        } else {
            System.out.println("\nCritical Path(s):");
            for (List<String> path : criticalPaths) {
                System.out.println(String.join(" -> ", path));
            }
        }
    }

    static List<String> topoSort(Map<String, Activity> acts) {
        Map<String, Integer> indeg = new HashMap<>();
        for (String id : acts.keySet()) indeg.put(id, 0);
        for (Activity a : acts.values()) {
            for (String s : a.succs) indeg.put(s, indeg.get(s) + 1);
        }
        Queue<String> q = new ArrayDeque<>();
        for (Map.Entry<String, Integer> e : indeg.entrySet()) {
            if (e.getValue() == 0) q.add(e.getKey());
        }
        List<String> order = new ArrayList<>();
        while (!q.isEmpty()) {
            String u = q.poll();
            order.add(u);
            for (String v : acts.get(u).succs) {
                indeg.put(v, indeg.get(v) - 1);
                if (indeg.get(v) == 0) q.add(v);
            }
        }
        if (order.size() != acts.size()) return null;
        return order;
    }

    static List<List<String>> findCriticalPaths(Map<String, Activity> acts) {
        List<String> starts = new ArrayList<>();
        for (Activity a : acts.values()) if (a.preds.isEmpty())
starts.add(a.id);

        List<List<String>> results = new ArrayList<>();
        for (String start : starts) {
            List<String> path = new ArrayList<>();
            dfsCritical(start, acts, path, results);
        }
        Set<String> seen = new HashSet<>();
        List<List<String>> unique = new ArrayList<>();
        for (List<String> p : results) {
            String key = String.join("->", p);
            if (!seen.contains(key)) { seen.add(key); unique.add(p); }
        }
        return unique;
    }

    static void dfsCritical(String curId, Map<String, Activity> acts,
List<String> path, List<List<String>> results) {
        Activity cur = acts.get(curId);
        if (cur.slack != 0) return;
        path.add(curId);

        boolean anyCriticalSucc = false;
        for (String s : cur.succs) {
            Activity succ = acts.get(s);
            if (succ.slack == 0 && cur.EF == succ.ES) {
                anyCriticalSucc = true;
                dfsCritical(s, acts, path, results);
            }
        }
        if (!anyCriticalSucc) {
            results.add(new ArrayList<>(path));
        }
        path.remove(path.size() - 1);
    }
}
```

Problem 1: Output

```
Activity | Dur | ES | EF | LS | LF | Slack
--------------------------------------------------
   A     |  3 |  0 |  3 |  0 |  3 |  0
   B     |  2 |  3 |  5 |  4 |  6 |  1
   C     |  4 |  3 |  7 |  3 |  7 |  0
   D     |  2 |  7 |  9 |  7 |  9 |  0
   E     |  3 |  7 | 10 |  8 | 11 |  1
   F     |  1 | 10 | 11 | 10 | 11 |  0

Project Duration = 11

Critical Path(s):
A -> C -> D -> F
```