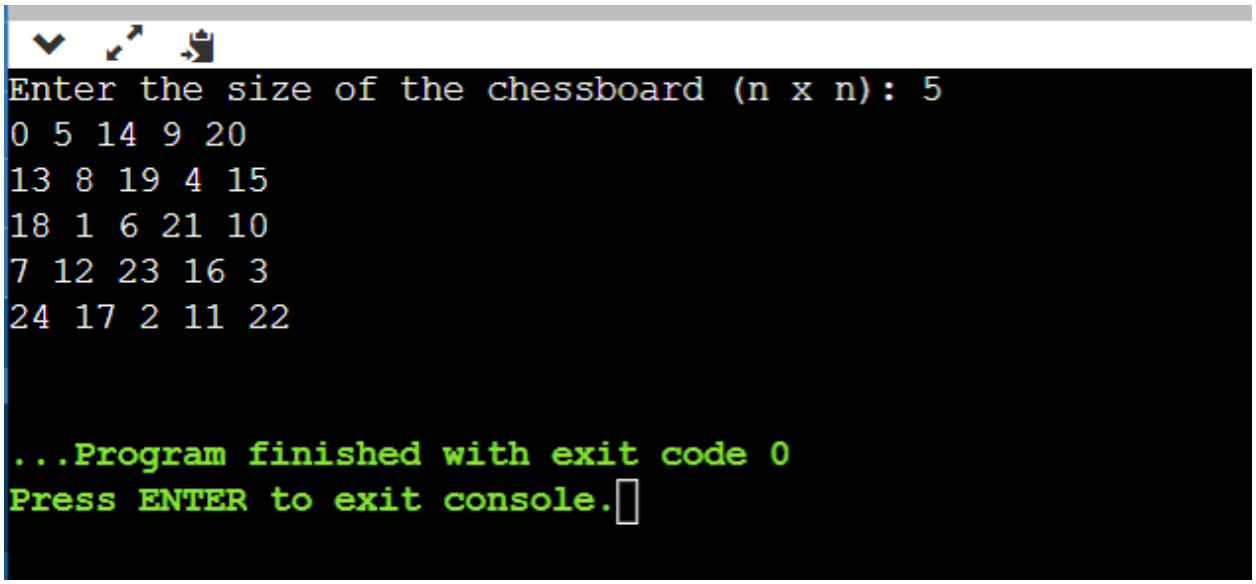
// Assignment : 5
// Name : Diptesh Anil Deore
// Roll No : TYCOA47

Code :

```
public class KnightTour {
    static int n;

    static boolean isSafe(int x, int y, int[][] board) {
        return (x >= 0 && y >= 0 && x < n && y < n && board[x][y] == -1);
    }

    static void printSolution(int n, int[][] board) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(board[i][j] + " ");
            }
            System.out.println();
        }
    }

    static boolean solveKT(int n) {
        int[][] board = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                board[i][j] = -1;
            }
        }
        int[] move_x = {2, 1, -1, -2, -2, -1, 1, 2};
        int[] move_y = {1, 2, 2, 1, -1, -2, -2, -1};

        board[0][0] = 0;
        int pos = 1;

        if (!solveKTUtil(n, board, 0, 0, move_x, move_y, pos)) {
            System.out.println("Solution does not exist");
            return false;
        } else {
            printSolution(n, board);
            return true;
        }
    }
```

```
    static boolean solveKTUtil(int n, int[][] board, int curr_x, int curr_y, int[] move_x, int[] move_y,
int pos) {
        if (pos == n * n) {
            return true;
        }

        for (int i = 0; i < 8; i++) {
            int new_x = curr_x + move_x[i];
            int new_y = curr_y + move_y[i];
            if (isSafe(new_x, new_y, board)) {
                board[new_x][new_y] = pos;
                if (solveKTUtil(n, board, new_x, new_y, move_x, move_y, pos + 1)) {
                    return true;
                }
                board[new_x][new_y] = -1;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        n = Integer.parseInt(System.console().readLine("Enter the size of the chessboard (n x n):
"));
        solveKT(n);
    }
}
```

Output :



// Assignment : 5
// Name : Sangharsh Devtale
// Roll No : TYCOA56

Code :

```
public class KnightTour {
    static int n;

    static boolean isSafe(int x, int y, int[][] board) {
        return (x >= 0 && y >= 0 && x < n && y < n && board[x][y] == -1);
    }

    static void printSolution(int n, int[][] board) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                System.out.print(board[i][j] + " ");
            }
            System.out.println();
        }
    }

    static boolean solveKT(int n) {
        int[][] board = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                board[i][j] = -1;
            }
        }
        int[] move_x = {2, 1, -1, -2, -2, -1, 1, 2};
        int[] move_y = {1, 2, 2, 1, -1, -2, -2, -1};

        board[0][0] = 0;
        int pos = 1;

        if (!solveKTUtil(n, board, 0, 0, move_x, move_y, pos)) {
            System.out.println("Solution does not exist");
            return false;
        } else {
            printSolution(n, board);
            return true;
        }
    }
```

```
    static boolean solveKTUtil(int n, int[][] board, int curr_x, int curr_y, int[] move_x, int[] move_y,
int pos) {
        if (pos == n * n) {
            return true;
        }

        for (int i = 0; i < 8; i++) {
            int new_x = curr_x + move_x[i];
            int new_y = curr_y + move_y[i];
            if (isSafe(new_x, new_y, board)) {
                board[new_x][new_y] = pos;
                if (solveKTUtil(n, board, new_x, new_y, move_x, move_y, pos + 1)) {
                    return true;
                }
                board[new_x][new_y] = -1;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        n = Integer.parseInt(System.console().readLine("Enter the size of the chessboard (n x n):
"));
        solveKT(n);
    }
}
```

Output :

```
Enter the size of the chessboard (n x n): 8
0 59 38 33 30 17 8 63
37 34 31 60 9 62 29 16
58 1 36 39 32 27 18 7
35 48 41 26 61 10 15 28
42 57 2 49 40 23 6 19
47 50 45 54 25 20 11 14
56 43 52 3 22 13 24 5
51 46 55 44 53 4 21 12
```

// Assignment No : 4
// Name : Diptesh Anil Deore
// Roll No : TYCOA47

Code :

```
import java.util.*;


public class Dijkstra {

    //final static int INF = 2147483647;

private int dist[];
private Set<Integer> settled;
private PriorityQueue<Node> pq;
private int V;
List<List<Node> > adj;

// Constructor of this class
public Dijkstra(int V)
{

// This keyword refers to current object itself
this.V = V;
dist = new int[V];
settled = new HashSet<Integer>();
pq = new PriorityQueue<Node>(V, new Node());
}

// Method 1
// Dijkstra's Algorithm
public void dijkstra(List<List<Node> > adj, int src)
{
this.adj = adj;

for (int i = 0; i < V; i++)
dist[i] = Integer.MAX_VALUE;

// Add source node to the priority queue
pq.add(new Node(src, 0));

// Distance to the source is 0
dist[src] = 0;
```

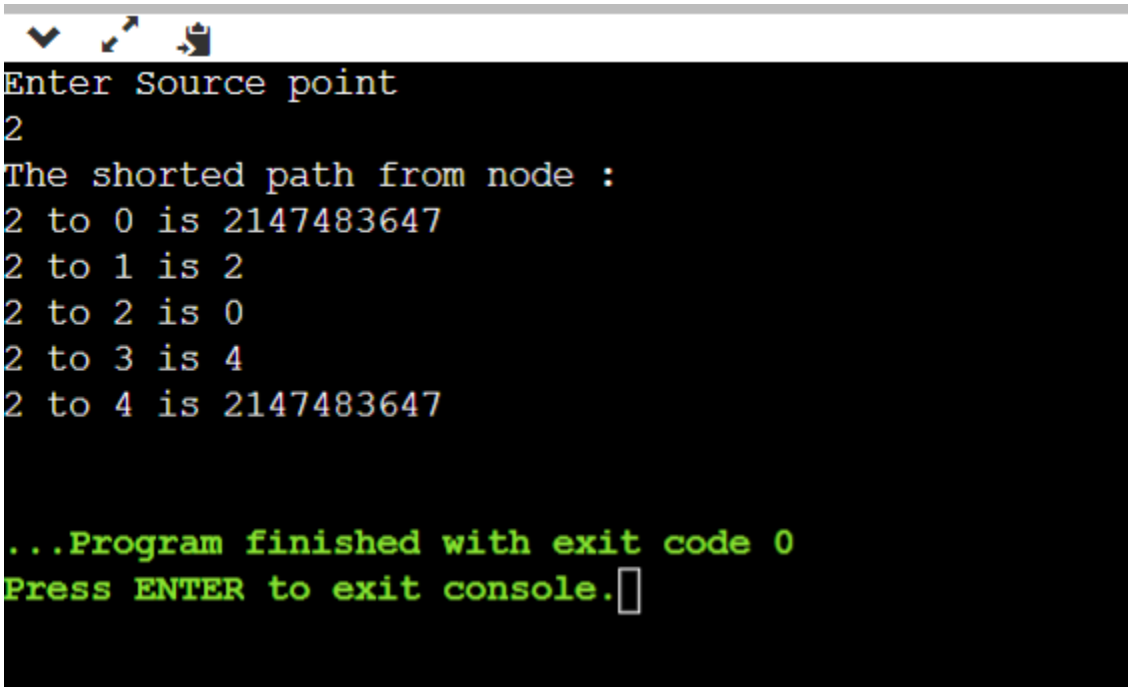
```
while (settled.size() != V) {

// Terminating condition check when
// the priority queue is empty, return
if (pq.isEmpty())
return;

// Removing the minimum distance node
// from the priority queue
int u = pq.remove().node;

// Adding the node whose distance is
// finalized
if (settled.contains(u))

// Continue keyword skips execution for
// following check
continue;

// We don't have to call e_Neighbors(u)
// if u is already present in the settled set.
settled.add(u);

e_Neighbours(u);
}
}

// Method 2
// To process all the neighbours
// of the passed node
private void e_Neighbours(int u)
{

int edgeDistance = -1;
int newDistance = -1;

// All the neighbors of v
for (int i = 0; i < adj.get(u).size(); i++) {
Node v = adj.get(u).get(i);

// If current node hasn't already been processed
if (!settled.contains(v.node)) {
edgeDistance = v.cost;
```

```
newDistance = dist[u] + edgeDistance;

// If new distance is cheaper in cost
if (newDistance < dist[v.node])
dist[v.node] = newDistance;

// Add the current node to the queue
pq.add(new Node(v.node, dist[v.node]));
}
}
}

// Main driver method
public static void main(String arg[])
{

int V = 5;
Scanner sc=new Scanner(System.in);
System.out.println("Enter Source point");
int source = sc.nextInt();

// Adjacency list representation of the
// connected edges by declaring List class object
// Declaring object of type List<Node>
List<List<Node> > adj
= new ArrayList<List<Node> >();

// Initialize list for every node
for (int i = 0; i < V; i++) {
List<Node> item = new ArrayList<Node>();
adj.add(item);
}

// Inputs for the GFG(dpq) graph
adj.get(0).add(new Node(1, 9));
adj.get(0).add(new Node(2, 6));
adj.get(0).add(new Node(3, 5));
adj.get(0).add(new Node(4, 3));

adj.get(2).add(new Node(1, 2));
adj.get(2).add(new Node(3, 4));

// Calculating the single source shortest path
Dijkstra dpq = new Dijkstra(V);
```

```
dpq.dijkstra(adj, source);

// Printing the shortest path to all the nodes
// from the source node
System.out.println("The shorted path from node :");

for (int i = 0; i < dpq.dist.length; i++)
System.out.println(source + " to " + i + " is "
+ dpq.dist[i]);
}
}

// Class 2
// Helper class implementing Comparator interface
// Representing a node in the graph
class Node implements Comparator<Node> {

// Member variables of this class
public int node;
public int cost;

// Constructors of this class

// Constructor 1
public Node() {}

// Constructor 2
public Node(int node, int cost)
{

// This keyword refers to current instance itself
this.node = node;
this.cost = cost;
}

// Method 1
@Override public int compare(Node node1, Node node2)
{

if (node1.cost < node2.cost)
return -1;

if (node1.cost > node2.cost)
return 1;
```

```
return 0;
}
}
```

Output :



// Assignment No : 4
// Name :Sangharsh Devtale
// Roll No : TYCOA56

Code :

```
import java.util.*;


public class Dijkstra {

    //final static int INF = 2147483647;

private int dist[];
private Set<Integer> settled;
private PriorityQueue<Node> pq;
private int V;
List<List<Node> > adj;

// Constructor of this class
public Dijkstra(int V)
{

// This keyword refers to current object itself
this.V = V;
dist = new int[V];
settled = new HashSet<Integer>();
pq = new PriorityQueue<Node>(V, new Node());
}

// Method 1
// Dijkstra's Algorithm
public void dijkstra(List<List<Node> > adj, int src)
{
this.adj = adj;

for (int i = 0; i < V; i++)
dist[i] = Integer.MAX_VALUE;

// Add source node to the priority queue
pq.add(new Node(src, 0));

// Distance to the source is 0
dist[src] = 0;
```

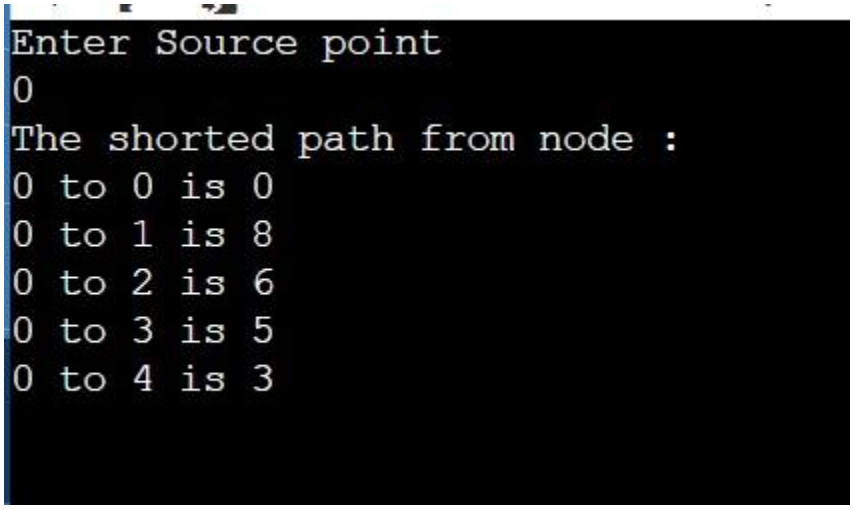
```
while (settled.size() != V) {

// Terminating condition check when
// the priority queue is empty, return
if (pq.isEmpty())
return;

// Removing the minimum distance node
// from the priority queue
int u = pq.remove().node;

// Adding the node whose distance is
// finalized
if (settled.contains(u))

// Continue keyword skips execution for
// following check
continue;

// We don't have to call e_Neighbors(u)
// if u is already present in the settled set.
settled.add(u);

e_Neighbours(u);
}
}

// Method 2
// To process all the neighbours
// of the passed node
private void e_Neighbours(int u)
{

int edgeDistance = -1;
int newDistance = -1;

// All the neighbors of v
for (int i = 0; i < adj.get(u).size(); i++) {
Node v = adj.get(u).get(i);

// If current node hasn't already been processed
if (!settled.contains(v.node)) {
edgeDistance = v.cost;
```

```
newDistance = dist[u] + edgeDistance;

// If new distance is cheaper in cost
if (newDistance < dist[v.node])
dist[v.node] = newDistance;

// Add the current node to the queue
pq.add(new Node(v.node, dist[v.node]));
}
}
}

// Main driver method
public static void main(String arg[])
{

int V = 5;
Scanner sc=new Scanner(System.in);
System.out.println("Enter Source point");
int source = sc.nextInt();

// Adjacency list representation of the
// connected edges by declaring List class object
// Declaring object of type List<Node>
List<List<Node> > adj
= new ArrayList<List<Node> >();

// Initialize list for every node
for (int i = 0; i < V; i++) {
List<Node> item = new ArrayList<Node>();
adj.add(item);
}

// Inputs for the GFG(dpq) graph
adj.get(0).add(new Node(1, 9));
adj.get(0).add(new Node(2, 6));
adj.get(0).add(new Node(3, 5));
adj.get(0).add(new Node(4, 3));

adj.get(2).add(new Node(1, 2));
adj.get(2).add(new Node(3, 4));

// Calculating the single source shortest path
Dijkstra dpq = new Dijkstra(V);
```

```
dpq.dijkstra(adj, source);

// Printing the shortest path to all the nodes
// from the source node
System.out.println("The shorted path from node :");

for (int i = 0; i < dpq.dist.length; i++)
System.out.println(source + " to " + i + " is "
+ dpq.dist[i]);
}
}

// Class 2
// Helper class implementing Comparator interface
// Representing a node in the graph
class Node implements Comparator<Node> {

// Member variables of this class
public int node;
public int cost;

// Constructors of this class

// Constructor 1
public Node() {}

// Constructor 2
public Node(int node, int cost)
{

// This keyword refers to current instance itself
this.node = node;
this.cost = cost;
}

// Method 1
@Override public int compare(Node node1, Node node2)
{

if (node1.cost < node2.cost)
return -1;

if (node1.cost > node2.cost)
return 1;
```

```
return 0;
}
}
```

Output :



// Assignment No : 6
// Name : Diptesh Anil Deore
// Roll No : TYCOA47

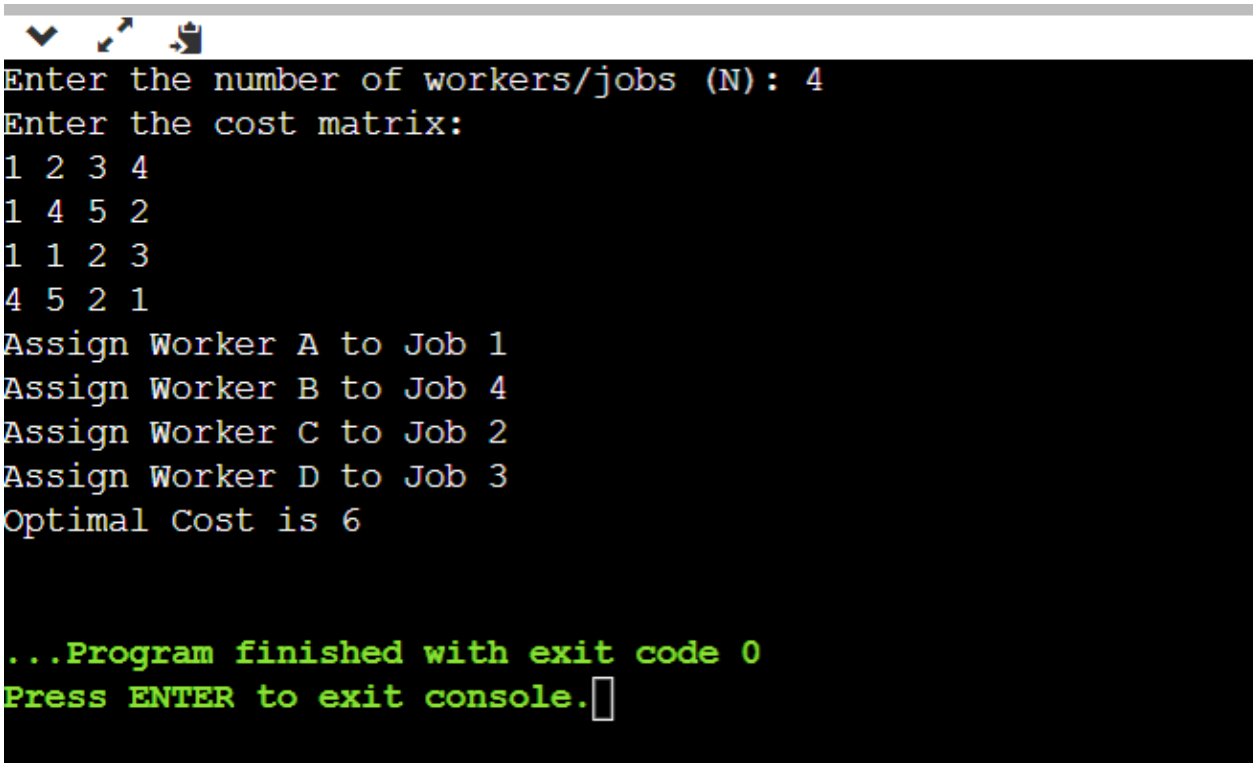
Code :

```
import java.util.PriorityQueue;
import java.util.Scanner;

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

        System.out.print("Enter the number of workers/jobs (N): ");
        int N = scanner.nextInt();

        int[][] costMatrix = new int[N][N];

        System.out.println("Enter the cost matrix:");
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                costMatrix[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Optimal Cost is " + findMinCost(costMatrix, N));
        scanner.close();
    }

    static class Node {
        Node parent;
        int pathCost;
        int cost;
        int workerID;
        int jobID;
        boolean[] assigned;

        Node(int x, int y, boolean[] assigned, Node parent, int N) {
            this.assigned = new boolean[N];
            for (int j = 0; j < N; j++) {
                this.assigned[j] = assigned[j];
            }
            if (x != -1 && y != -1) {
                this.assigned[y] = true;
```

```
        }
        this.parent = parent;
        this.workerID = x;
        this.jobID = y;
    }
}

static int calculateCost(int[][] costMatrix, int x, int y, boolean[] assigned, int N) {
    int cost = 0;
    boolean[] available = new boolean[N];
    for (int j = 0; j < N; j++) {
        available[j] = true;
    }

    for (int i = x + 1; i < N; i++) {
        int min = Integer.MAX_VALUE;
        int minIndex = -1;

        for (int j = 0; j < N; j++) {
            if (!assigned[j] && available[j] && costMatrix[i][j] < min) {
                minIndex = j;
                min = costMatrix[i][j];
            }
        }

        cost += min;
        available[minIndex] = false;
    }

    return cost;
}

static class NodeComparator implements java.util.Comparator<Node> {
    public int compare(Node lhs, Node rhs) {
        return lhs.cost - rhs.cost;
    }
}

static void printAssignments(Node min, int N) {
    if (min.parent == null) {
        return;
    }
    printAssignments(min.parent, N);
```

```
        System.out.println("Assign Worker " + (char) (min.workerID + 'A') + " to Job " + (min.jobID
+1));
    }

    static int findMinCost(int[][] costMatrix, int N) {
        PriorityQueue<Node> pq = new PriorityQueue<>(new NodeComparator());

        boolean[] assigned = new boolean[N];
        Node root = new Node(-1, -1, assigned, null, N);
        root.pathCost = root.cost = 0;
        root.workerID = -1;

        pq.add(root);

        while (!pq.isEmpty()) {
            Node min = pq.poll();
            int i = min.workerID + 1;
            if (i == N) {
                printAssignments(min, N);
                return min.cost;
            }

            for (int j = 0; j < N; j++) {
                if (!min.assigned[j]) {
                    Node child = new Node(i, j, min.assigned, min, N);
                    child.pathCost = min.pathCost + costMatrix[i][j];
                    child.cost = child.pathCost + calculateCost(costMatrix, i, j, child.assigned, N);
                    pq.add(child);
                }
            }
        }
        return -1;
    }
}
```

Output :


Enter the number of workers/jobs (N): 4
Enter the cost matrix:
1 2 3 4
1 4 5 2
1 1 2 3
4 5 2 1
Assign Worker A to Job 1
Assign Worker B to Job 4
Assign Worker C to Job 2
Assign Worker D to Job 3
Optimal Cost is 6
...Program finished with exit code 0
Press ENTER to exit console.

// Assignment No : 6
// Name : Sangharsh Devtale
// Roll No : TYCOA56

Code :

```
import java.util.PriorityQueue;
import java.util.Scanner;

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

        System.out.print("Enter the number of workers/jobs (N): ");
        int N = scanner.nextInt();

        int[][] costMatrix = new int[N][N];

        System.out.println("Enter the cost matrix:");
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                costMatrix[i][j] = scanner.nextInt();
            }
        }

        System.out.println("Optimal Cost is " + findMinCost(costMatrix, N));
        scanner.close();
    }

    static class Node {
        Node parent;
        int pathCost;
        int cost;
        int workerID;
        int jobID;
        boolean[] assigned;

        Node(int x, int y, boolean[] assigned, Node parent, int N) {
            this.assigned = new boolean[N];
            for (int j = 0; j < N; j++) {
                this.assigned[j] = assigned[j];
            }
            if (x != -1 && y != -1) {
                this.assigned[y] = true;
```

```
        }
        this.parent = parent;
        this.workerID = x;
        this.jobID = y;
    }
}

static int calculateCost(int[][] costMatrix, int x, int y, boolean[] assigned, int N) {
    int cost = 0;
    boolean[] available = new boolean[N];
    for (int j = 0; j < N; j++) {
        available[j] = true;
    }

    for (int i = x + 1; i < N; i++) {
        int min = Integer.MAX_VALUE;
        int minIndex = -1;

        for (int j = 0; j < N; j++) {
            if (!assigned[j] && available[j] && costMatrix[i][j] < min) {
                minIndex = j;
                min = costMatrix[i][j];
            }
        }

        cost += min;
        available[minIndex] = false;
    }

    return cost;
}

static class NodeComparator implements java.util.Comparator<Node> {
    public int compare(Node lhs, Node rhs) {
        return lhs.cost - rhs.cost;
    }
}

static void printAssignments(Node min, int N) {
    if (min.parent == null) {
        return;
    }
    printAssignments(min.parent, N);
```

```
        System.out.println("Assign Worker " + (char) (min.workerID + 'A') + " to Job " + (min.jobID
+1));
    }

    static int findMinCost(int[][] costMatrix, int N) {
        PriorityQueue<Node> pq = new PriorityQueue<>(new NodeComparator());

        boolean[] assigned = new boolean[N];
        Node root = new Node(-1, -1, assigned, null, N);
        root.pathCost = root.cost = 0;
        root.workerID = -1;

        pq.add(root);

        while (!pq.isEmpty()) {
            Node min = pq.poll();
            int i = min.workerID + 1;
            if (i == N) {
                printAssignments(min, N);
                return min.cost;
            }

            for (int j = 0; j < N; j++) {
                if (!min.assigned[j]) {
                    Node child = new Node(i, j, min.assigned, min, N);
                    child.pathCost = min.pathCost + costMatrix[i][j];
                    child.cost = child.pathCost + calculateCost(costMatrix, i, j, child.assigned, N);
                    pq.add(child);
                }
            }
        }
        return -1;
    }
}
```

Output :

```
Enter the number of workers/jobs (N): 4
Enter the cost matrix:
9
2
7
8
6
4
3
7
5
8
1
8
7
6
9
4
Assign Worker A to Job 2
Assign Worker B to Job 1
Assign Worker C to Job 3
Assign Worker D to Job 4
Optimal Cost is 13
```

// Name : Diptesh Deore
// Project Structure :

```
EmployeeManagementWebApp
├── src
│   └── com.example
│       ├── Employee.java
│       └── EmployeeServlet.java
├── WebContent
│   ├── WEB-INF
│   │   └── web.xml
│   └── index.jsp
```

// Employee.java

```
package com.example;

public class Employee {
    private int id;
    private String name;

    public Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }
}
```

// EmployeeServlet.java

```
package com.example;

import java.io.*;
```

```
import java.util.ArrayList;
import java.util.List;
import javax.servlet.*;
import javax.servlet.http.*;

public class EmployeeServlet extends HttpServlet {
    private List<Employee> employees = new ArrayList<>();

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        request.setAttribute("employees", employees);
        request.getRequestDispatcher("/index.jsp").forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String name = request.getParameter("name");
        String idStr = request.getParameter("id");

        if (name != null && idStr != null && !name.isEmpty() && !idStr.isEmpty()) {
            int id = Integer.parseInt(idStr);
            Employee newEmployee = new Employee(id, name);
            employees.add(newEmployee);
        }

        response.sendRedirect(request.getContextPath() + "/EmployeeServlet");
    }
}
```

```
// index.jsp
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <title>Employee Management</title>
</head>
<body>
    <h1>Employee Management</h1>
```

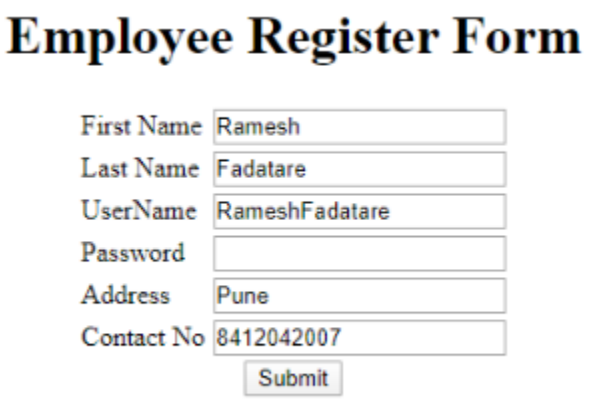
```
    <!-- Employee List -->
    <h2>Employee List</h2>
    <ul>
        <c:forEach items="${employees}" var="employee">
            <li>${employee.id}: ${employee.name}</li>
        </c:forEach>
    </ul>

    <!-- Add Employee Form -->
    <h2>Add Employee</h2>
    <form action="EmployeeServlet" method="post">
        <input type="text" name="id" placeholder="Employee ID">
        <input type="text" name="name" placeholder="Employee Name">
        <input type="submit" value="Add Employee">
    </form>
</body>
</html>
```

// web.xml

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <title>Employee Management</title>
</head>
<body>
    <h1>Employee Management</h1>

    <!-- Employee List -->
    <h2>Employee List</h2>
    <ul>
        <c:forEach items="${employees}" var="employee">
            <li>${employee.id}: ${employee.name}</li>
        </c:forEach>
    </ul>
```

```
    <!-- Add Employee Form -->
    <h2>Add Employee</h2>
    <form action="EmployeeServlet" method="post">
        <input type="text" name="id" placeholder="Employee ID">
        <input type="text" name="name" placeholder="Employee Name">
        <input type="submit" value="Add Employee">
    </form>
</body>
</html>
```

Output :



// Name : Sangharsh Devtale
// Project Structure :

```
EmployeeManagementWebApp
├── src
│   └── com.example
│       ├── Employee.java
│       └── EmployeeServlet.java
├── WebContent
│   ├── WEB-INF
│   │   └── web.xml
│   └── index.jsp
```

// Employee.java

```
package com.example;

public class Employee {
    private int id;
    private String name;

    public Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }
}
```

// EmployeeServlet.java

```
package com.example;

import java.io.*;
```

```
import java.util.ArrayList;
import java.util.List;
import javax.servlet.*;
import javax.servlet.http.*;

public class EmployeeServlet extends HttpServlet {
    private List<Employee> employees = new ArrayList<>();

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        request.setAttribute("employees", employees);
        request.getRequestDispatcher("/index.jsp").forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String name = request.getParameter("name");
        String idStr = request.getParameter("id");

        if (name != null && idStr != null && !name.isEmpty() && !idStr.isEmpty()) {
            int id = Integer.parseInt(idStr);
            Employee newEmployee = new Employee(id, name);
            employees.add(newEmployee);
        }

        response.sendRedirect(request.getContextPath() + "/EmployeeServlet");
    }
}
```

```
// index.jsp
```

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <title>Employee Management</title>
</head>
<body>
    <h1>Employee Management</h1>
```

```
    <!-- Employee List -->
    <h2>Employee List</h2>
    <ul>
        <c:forEach items="${employees}" var="employee">
            <li>${employee.id}: ${employee.name}</li>
        </c:forEach>
    </ul>

    <!-- Add Employee Form -->
    <h2>Add Employee</h2>
    <form action="EmployeeServlet" method="post">
        <input type="text" name="id" placeholder="Employee ID">
        <input type="text" name="name" placeholder="Employee Name">
        <input type="submit" value="Add Employee">
    </form>
</body>
</html>
```

// web.xml

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <title>Employee Management</title>
</head>
<body>
    <h1>Employee Management</h1>

    <!-- Employee List -->
    <h2>Employee List</h2>
    <ul>
        <c:forEach items="${employees}" var="employee">
            <li>${employee.id}: ${employee.name}</li>
        </c:forEach>
    </ul>
```

```
    <!-- Add Employee Form -->
    <h2>Add Employee</h2>
    <form action="EmployeeServlet" method="post">
        <input type="text" name="id" placeholder="Employee ID">
        <input type="text" name="name" placeholder="Employee Name">
        <input type="submit" value="Add Employee">
    </form>
</body>
</html>
```

Output :