# 6th Semester

## Matrix Minimization

Design patterns group project

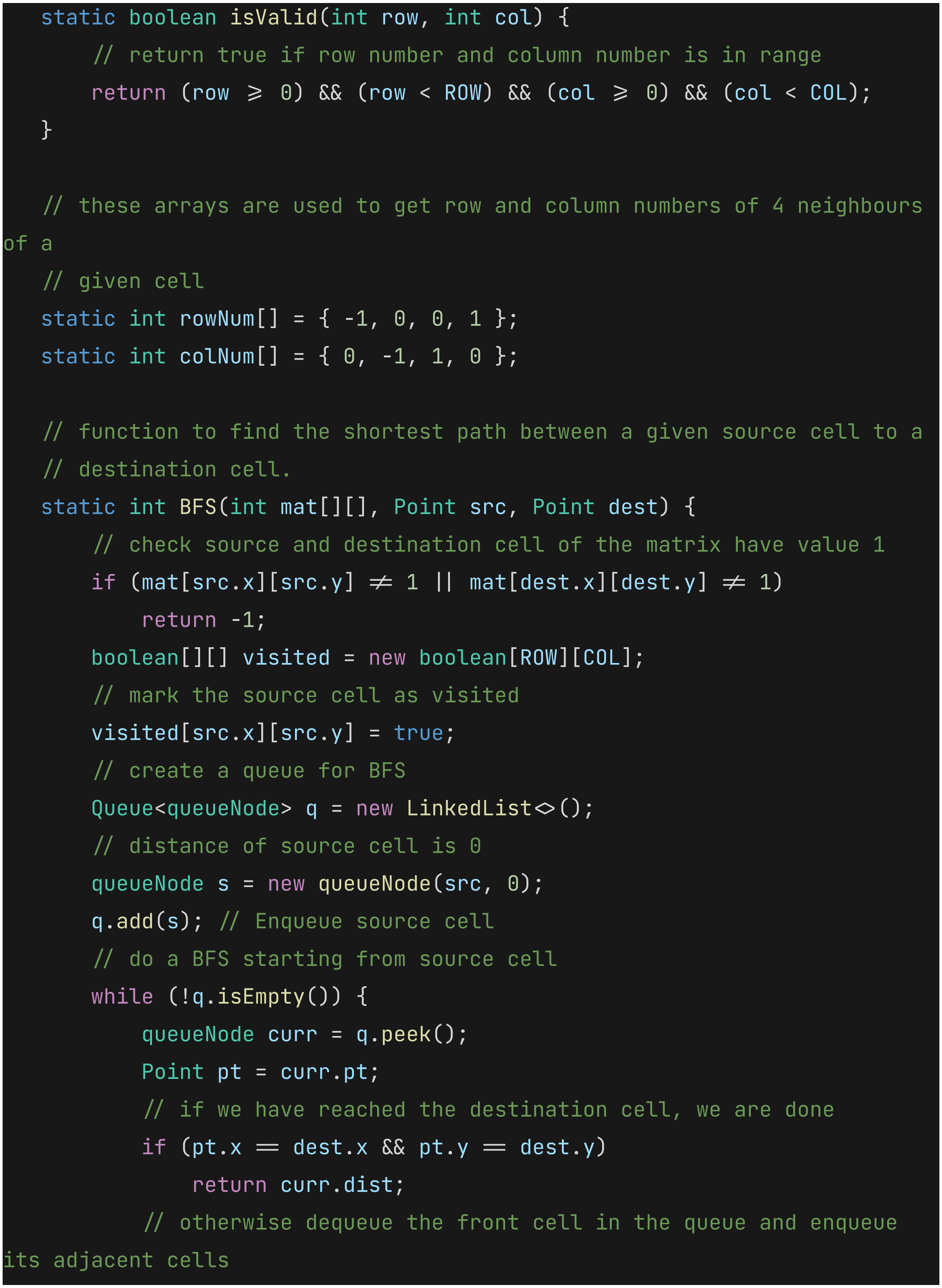
**Prepared By:**
Yash Sonalia
191001109011
BSc. I.T. Software Engineering & DevOps

Nilabja Dey
191001111014
BSc. I.T. Machine Learning

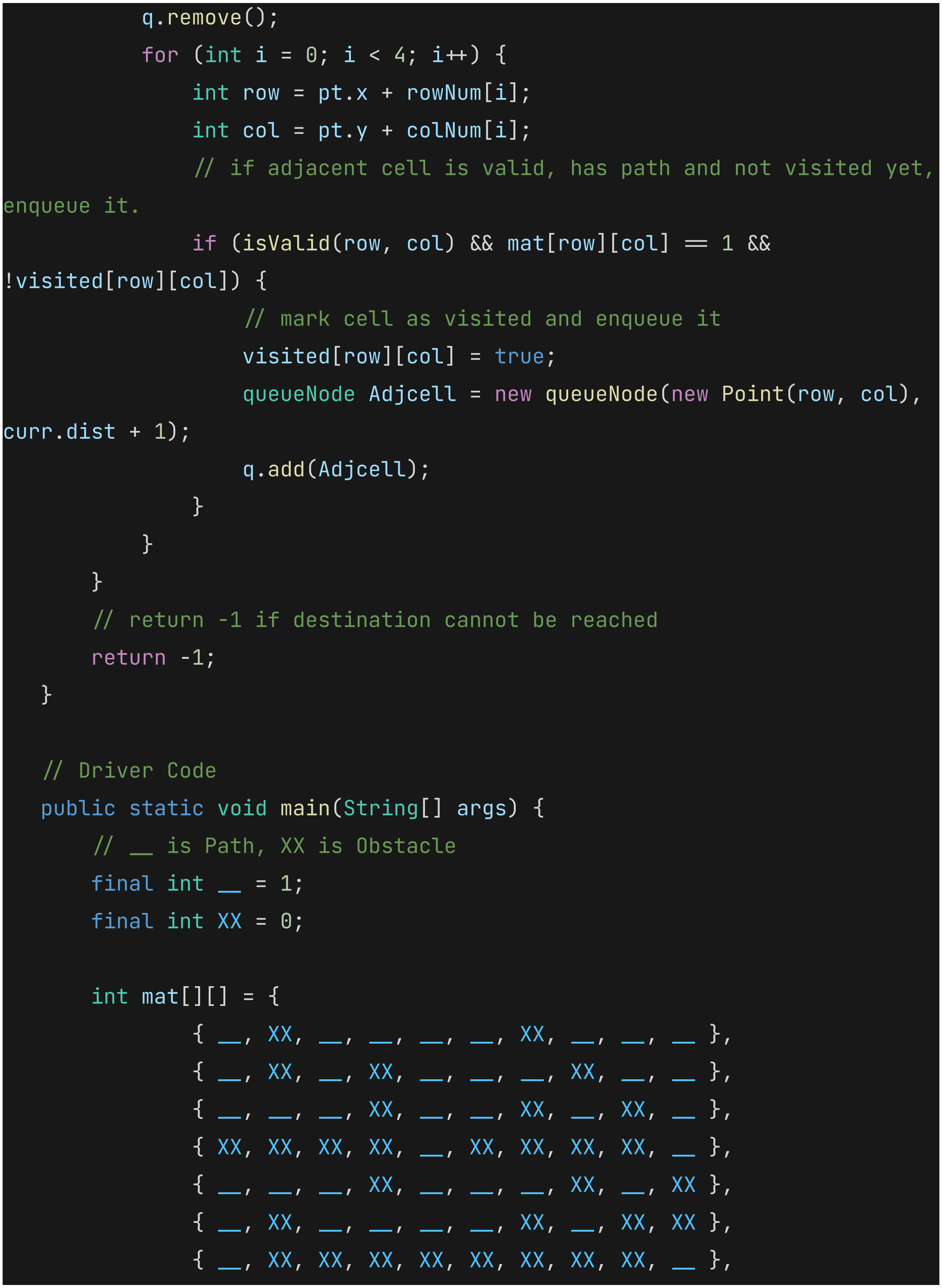
Anirudhya Paul
191001111008
BSc. I.T. Machine Learning

## Code

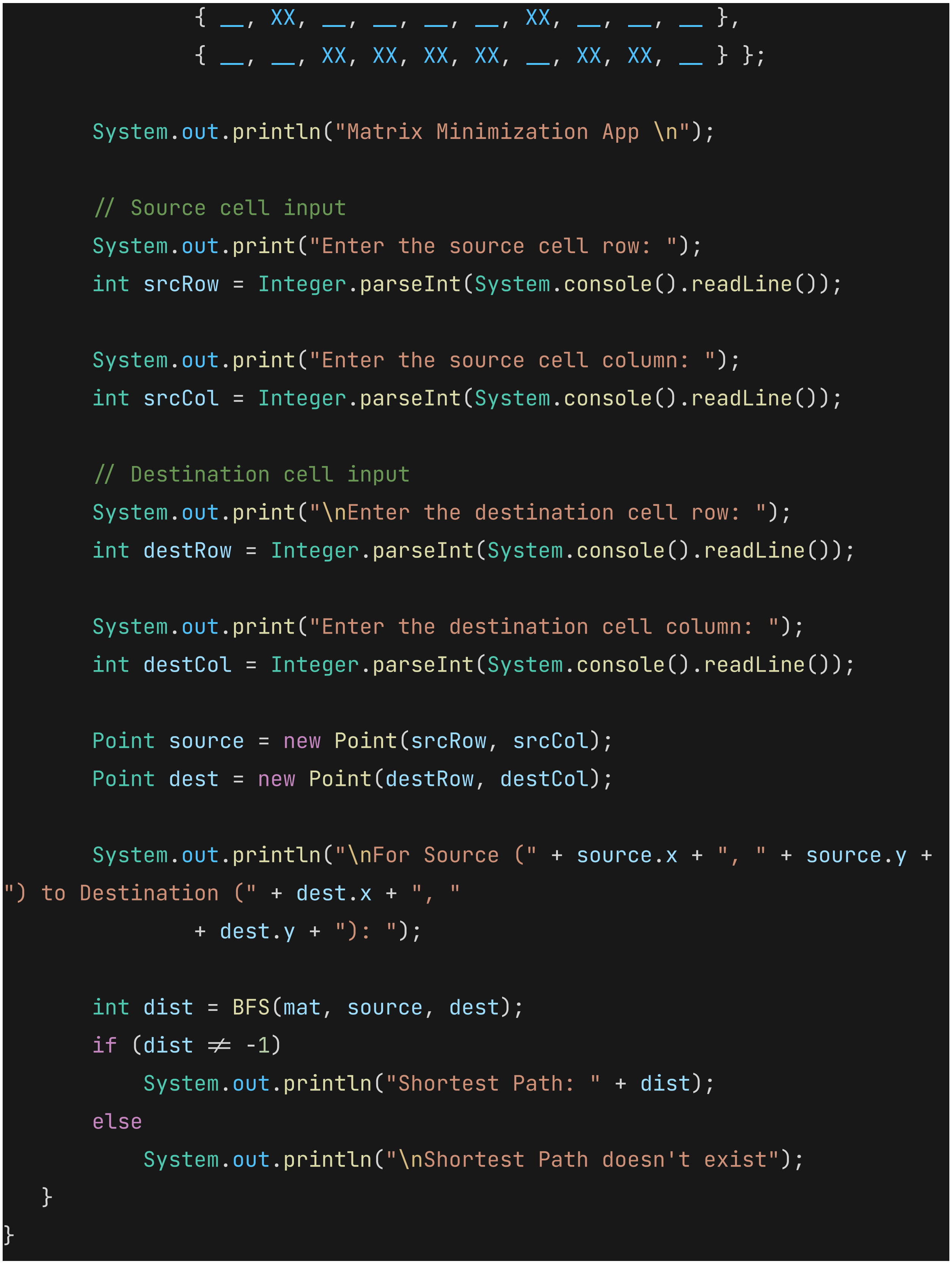
```
package src;

import java.util.LinkedList;
import java.util.Queue;

class MatrixOptimization {
    static int ROW = 9;
    static int COL = 10;

    // to store matrix cell coordinates
    static class Point {
        int x;
        int y;

        public Point(int x, int y) {
            this.x = x;
            this.y = y;
        }
    };

    // a Data Structure for queue used in BFS
    static class queueNode {
        Point pt; // the coordinates of a cell
        int dist; // cell's distance of from the source

        public queueNode(Point pt, int dist) {
            this.pt = pt;
            this.dist = dist;
        }
    };

    // check whether given cell (row, col) is a valid cell or not.
```



```
   static boolean isValid(int row, int col) {
       // return true if row number and column number is in range
       return (row >= 0) && (row < ROW) && (col >= 0) && (col < COL);
   }

   // these arrays are used to get row and column numbers of 4 neighbours
of a
   // given cell
   static int rowNum[] = { -1, 0, 0, 1 };
   static int colNum[] = { 0, -1, 1, 0 };

   // function to find the shortest path between a given source cell to a
   // destination cell.
   static int BFS(int mat[][], Point src, Point dest) {
       // check source and destination cell of the matrix have value 1
       if (mat[src.x][src.y] != 1 || mat[dest.x][dest.y] != 1)
           return -1;
       boolean[][] visited = new boolean[ROW][COL];
       // mark the source cell as visited
       visited[src.x][src.y] = true;
       // create a queue for BFS
       Queue<queueNode> q = new LinkedList<>();
       // distance of source cell is 0
       queueNode s = new queueNode(src, 0);
       q.add(s); // Enqueue source cell
       // do a BFS starting from source cell
       while (!q.isEmpty()) {
           queueNode curr = q.peek();
           Point pt = curr.pt;
           // if we have reached the destination cell, we are done
           if (pt.x == dest.x && pt.y == dest.y)
               return curr.dist;
           // otherwise dequeue the front cell in the queue and enqueue
its adjacent cells
```

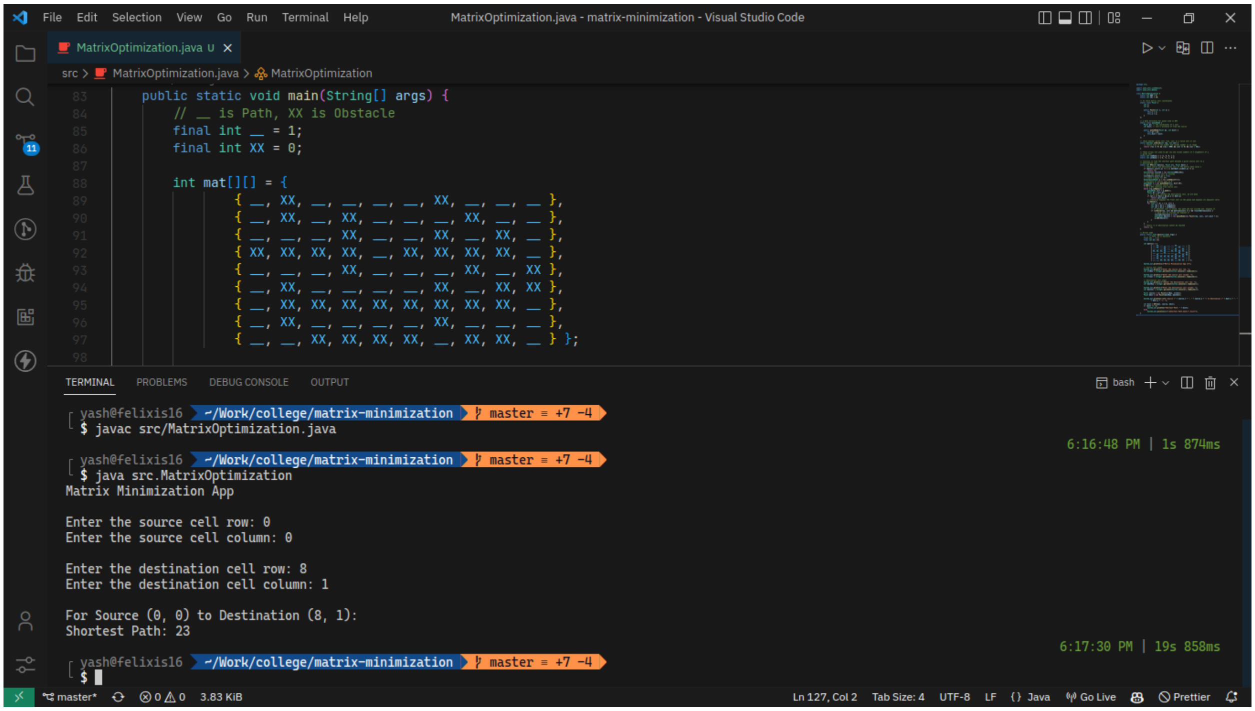


```
            q.remove();
            for (int i = 0; i < 4; i++) {
                int row = pt.x + rowNum[i];
                int col = pt.y + colNum[i];
                // if adjacent cell is valid, has path and not visited yet,
enqueue it.
                if (isValid(row, col) && mat[row][col] == 1 &&
!visited[row][col]) {
                    // mark cell as visited and enqueue it
                    visited[row][col] = true;
                    queueNode Adjcell = new queueNode(new Point(row, col),
curr.dist + 1);
                    q.add(Adjcell);
                }
            }
        }
        // return -1 if destination cannot be reached
        return -1;
    }

    // Driver Code
    public static void main(String[] args) {
        // __ is Path, XX is Obstacle
        final int __ = 1;
        final int XX = 0;

        int mat[][] = {
                { __, XX, __, __, __, __, XX, __, __, __ },
                { __, XX, __, XX, __, __, __, XX, __, __ },
                { __, __, __, XX, __, __, XX, __, XX, __ },
                { XX, XX, XX, XX, __, XX, XX, XX, XX, __ },
                { __, __, __, XX, __, __, __, XX, __, XX },
                { __, XX, __, __, __, __, XX, __, XX, XX },
                { __, XX, XX, XX, XX, XX, XX, XX, XX, __ },
```



```
                { __, XX, __, __, __, __, XX, __, __, __ },
                { __, __, XX, XX, XX, XX, __, XX, XX, __ } };

        System.out.println("Matrix Minimization App \n");

        // Source cell input
        System.out.print("Enter the source cell row: ");
        int srcRow = Integer.parseInt(System.console().readLine());

        System.out.print("Enter the source cell column: ");
        int srcCol = Integer.parseInt(System.console().readLine());

        // Destination cell input
        System.out.print("\nEnter the destination cell row: ");
        int destRow = Integer.parseInt(System.console().readLine());

        System.out.print("Enter the destination cell column: ");
        int destCol = Integer.parseInt(System.console().readLine());

        Point source = new Point(srcRow, srcCol);
        Point dest = new Point(destRow, destCol);

        System.out.println("\nFor Source (" + source.x + ", " + source.y +
") to Destination (" + dest.x + ", "
                + dest.y + "): ");

        int dist = BFS(mat, source, dest);
        if (dist != -1)
            System.out.println("Shortest Path: " + dist);
        else
            System.out.println("\nShortest Path doesn't exist");
    }
}
```

# Output



File Edit Selection View Go Run Terminal Help
MatrixOptimization.java - matrix-minimization - Visual Studio Code
MatrixOptimization.java U
src > MatrixOptimization.java > MatrixOptimization
TERMINAL
PROBLEMS
DEBUG CONSOLE
OUTPUT
bash
master* 
0 0 3.83 KiB
Ln 127, Col 2 Tab Size: 4 UTF-8 LF {} Java Go Live Prettier

```java
    public static void main(String[] args) {
        // __ is Path, XX is Obstacle
        final int __ = 1;
        final int XX = 0;

        int mat[][] = {
                { __, XX, __, __, __, __, XX, __, __, __ },
                { __, XX, __, XX, __, __, __, XX, __, __ },
                { __, __, __, XX, __, __, XX, __, XX, __ },
                { XX, XX, XX, XX, __, XX, XX, XX, XX, __ },
                { __, __, __, XX, __, __, __, XX, __, XX },
                { __, XX, __, __, __, __, XX, __, XX, XX },
                { __, XX, XX, XX, XX, XX, XX, XX, XX, __ },
                { __, XX, __, __, __, __, XX, __, __, __ },
                { __, __, XX, XX, XX, XX, __, XX, XX, __ } };

```

```
[ yash@felixis16  ~/Work/college/matrix-minimization  master ≡ +7 -4
└ $ javac src/MatrixOptimization.java
                                                                    6:16:48 PM | 1s 874ms
[ yash@felixis16  ~/Work/college/matrix-minimization  master ≡ +7 -4
└ $ java src.MatrixOptimization
Matrix Minimization App

Enter the source cell row: 0
Enter the source cell column: 0

Enter the destination cell row: 8
Enter the destination cell column: 1

For Source (0, 0) to Destination (8, 1):
Shortest Path: 23
                                                                    6:17:30 PM | 19s 858ms
[ yash@felixis16  ~/Work/college/matrix-minimization  master ≡ +7 -4
└ $ 
```