```
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;

public class Main
{

	Node[][] graph;
	Wall[] walls;

	private class Node
	{

		private ArrayList<Edge> adjList;

		private int column;
		private int row;
		private int layer;

		private boolean visited;

		public Node(int column, int row)
		{
			adjList = new ArrayList<Edge>();
			this.column = column;
			this.row = row;
			this.layer = 0;
			visited = false;
		}

	}

	/** Used to connect nodes to each other */
	private class Edge
	{

		private int to_column;
		private int to_row;

		public Edge(int to_column, int to_row)
		{

			this.to_column = to_column;
			this.to_row = to_row;

		}
```

```
}

private class Wall
{

	int start_column;
	int start_row;

	int end_column;
	int end_row;

	boolean isVertical = false;
	boolean isHorizontal = false;

	public Wall(int start_column, int start_row, int end_column, int end_row)
	{

		if(start_column == end_column)
			isVertical = true;
		else isHorizontal = true;

		this.start_row = start_row;
		this.start_column = start_column;

		this.end_row = end_row;
		this.end_column = end_column;

	}

}



/** Initialize the entire nodes that are in the graph */
private void initGraph(int num)
{

	graph = new Node[num][num];

	for(int i = 0; i < num; i++)
		for(int j = 0; j < num; j++)
			graph[i][j] = new Node(i, j);

}

private void initWalls(int num, Scanner sc)
{
        walls = new Wall[num];

        for(int i = 0; i < num; i++)
        {
            walls[i] = new Wall(sc.nextInt(), sc.nextInt(), sc.nextInt(), sc.nextInt());
        }

    }

    private void addEdge(int start_column, int start_row, int to_column, int to_row)
    {

        graph[start_column][start_row].adjList.add(new Edge(to_column, to_row));
        graph[to_column][to_row].adjList.add(new Edge(start_column, start_row));

    }

    private int bfs(int column, int row, int end_column, int end_row)
    {

        Queue<Node> queue = new LinkedList<Node>();
        Node current = null;

        for(int i = 0; i < graph.length; i++)
            for(Node node : graph[i])
                node.visited = false;

        graph[column][row].layer = 0;

        queue.add(graph[column][row]);

        while(!queue.isEmpty())
        {

            current = queue.peek();
            current.visited = true;

            for(Edge edge : current.adjList)
            {

                if(!graph[edge.to_column][edge.to_row].visited)
                {

                    queue.add(graph[edge.to_column][edge.to_row]);
                    graph[edge.to_column][edge.to_row].visited = true;
                    graph[edge.to_column][edge.to_row].layer = current.layer + 1;
                    }

                }

                queue.poll();

            }

            return graph[end_column][end_row].layer;

        }

        private void findPath(int start_column, int start_row, int end_column, int end_row)
        {

            Node bestNode = graph[start_column][start_row];
            while(bfs(bestNode.column, bestNode.row, end_column, end_row) != 0)
            {

                for(Edge edge : bestNode.adjList)
                {

                    int bestLayer = bfs(bestNode.column, bestNode.row, end_column, end_row);
                    int edgeLayer = bfs(edge.to_column, edge.to_row, end_column, end_row);
                    if(bestLayer > edgeLayer)
                    {

                        if(bestNode.column - edge.to_column == 0)
                        {
                            if(bestNode.row > edge.to_row)
                            {
                                System.out.print("N");
                            }
                            else System.out.print("S");
                        }
                        else
                        {

                            if(bestNode.column > edge.to_column)
                            {
                                System.out.print("W");
                            }
                            else System.out.print("E");

                        }

                        bestNode = graph[edge.to_column][edge.to_row];
```

```
					}
				}
			}

			System.out.println("");

		}

		private boolean canConnect(double start_column, double start_row, double to_column, double to_row)
		{

			boolean connection = true;

			for(int i = 0; i < walls.length; i++)
			{

				if(walls[i].isHorizontal)
				{
					if(start_column > walls[i].start_column && start_column < walls[i].end_column)
					{
						if(start_row > walls[i].start_row && to_row < walls[i].end_row || start_row <
walls[i].start_row && to_row > walls[i].end_row)
							connection = false;
					}
				}
				else
				{
					if(start_row > walls[i].start_row && start_row < walls[i].end_row)
					{
						if(start_column < walls[i].start_column && to_column >
walls[i].end_column || start_column > walls[i].start_column && to_column < walls[i].end_column)
							connection = false;
					}

				}

			}

			return connection;

		}

		public static void main(String[] args)
		{

			Main main = new Main();
			Scanner sc = new Scanner(System.in);
		int columnSize = 6;
		int rowSize = 6;
		int wallSize = 3;

		int start_column = sc.nextInt() - 1;
		int start_row = sc.nextInt() - 1 ;

		while(start_column != -1 || start_row != -1)
		{

			main.initGraph(columnSize);

			int end_column = sc.nextInt() - 1;
			int end_row = sc.nextInt() - 1;

			main.initWalls(wallSize, sc);

			for(int i = 0; i < rowSize; i++)
				for(int j = 0; j < columnSize; j++)
				{

					if(i > 0 && main.canConnect(j + 0.5, i + 0.5, j + 0.5, i - 1 + 0.5) &&
main.canConnect(j + 0.5, i - 1 + 0.5, j + 0.5, i + 0.5))
					{

						main.addEdge(j, i, j, i - 1);
						main.addEdge(j, i - 1, j, i);

					}

					if(main.canConnect(j + 0.5, i + 0.5, j + 1 + 0.5, i + 0.5) &&
main.canConnect(j + 1 + 0.5, i + 0.5, j + 0.5, i + 0.5) && j < columnSize - 1)
					{

						main.addEdge(j, i, j + 1, i);
						main.addEdge(j + 1, i, j, i);

					}
				}
			main.findPath(start_column, start_row, end_column, end_row);
			start_column = sc.nextInt() - 1;
			start_row = sc.nextInt() - 1;
		}
	}
}
```