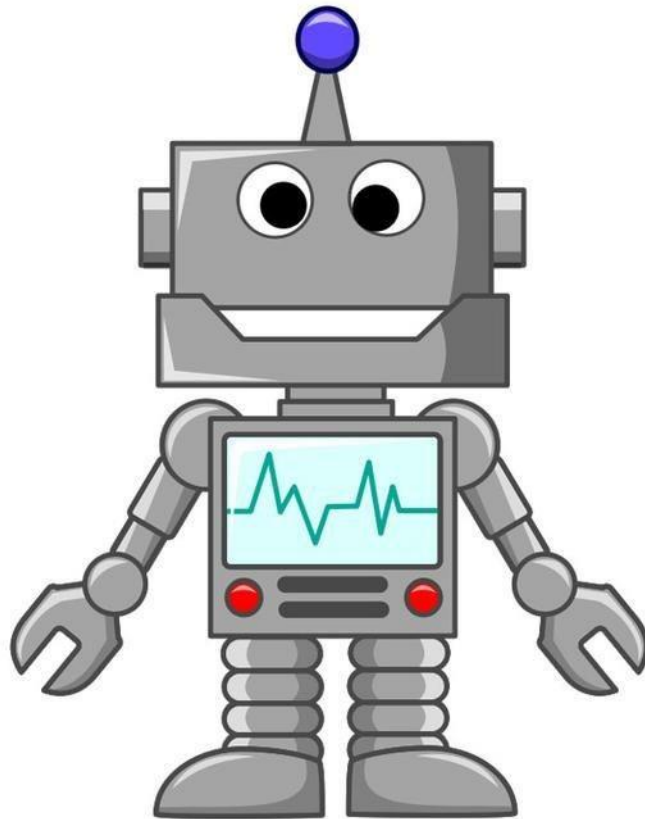


P R E D A



Laura Saltaren Andrade – 47586360G
lsaltaren1@alumno.uned.es

1.Respuesta a las cuestiones teóricas planteadas en este enunciado.

Coste Temporal y Espacial del Algoritmo:

Coste Temporal:

El algoritmo utiliza una búsqueda en profundidad mediante recursión para encontrar el tornillo.

En el peor caso, el algoritmo podría explorar cada celda una vez, ya que no hay garantía de que el camino más corto sea encontrado primero.

La complejidad temporal es, por lo tanto, $O(N \times M)$, donde N es el número de filas y M es el número de columnas.

Coste Espacial:

El algoritmo utiliza una pila (**Stack<Coordenada>**) para almacenar el camino.

En el peor caso, el tamaño de la pila podría ser $N \times M$, donde N y M son las dimensiones del edificio.

La complejidad espacial es $O(N \times M)$.

Basado en el peor escenario posible.

Otros Esquemas para Resolver el Problema:

Búsqueda en Anchura:

Podría ser más apropiada si se desea encontrar el camino más corto en lugar del primer camino.

Esta búsqueda garantiza encontrar el camino más corto en términos de número de movimientos.

Programación Dinámica:

Si hay múltiples caminos y se quiere optimizar en términos de alguna métrica (por ejemplo, minimizar la cantidad de movimientos o maximizar la distancia desde el tornillo), la programación dinámica podría ser una opción.

Se podría diseñar una función de valor que represente la "calidad" de un estado y utilizar técnicas de programación dinámica para encontrar la mejor secuencia de movimientos.

Algoritmos de Camino Mínimo:

Algoritmos como el algoritmo de Dijkstra o el algoritmo A* podrían ser útiles si se desea encontrar el camino más corto con eficiencia en términos de tiempo.

Estos algoritmos utilizan heurísticas y prioridades para explorar caminos prometedores primero.

2.Un ejemplo de ejecución para distintos tamaños del problema.

Para archivo prueba.txt

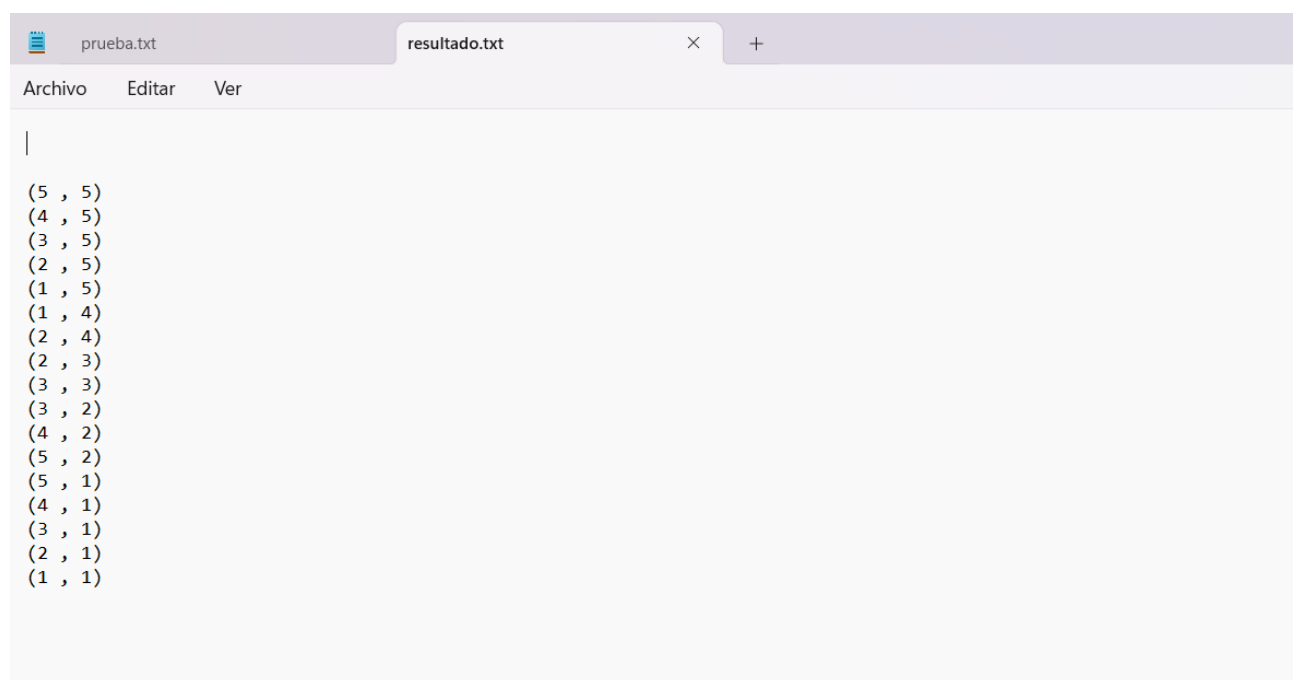
```
5
6
L L E L L E
L E L L L E
L L L E L L
L L E L L L
L L L L T L

C:\Users\lausa>java -jar robot.jar -t prueba.txt resultado.txt
Iniciando búsqueda con traza...

Se ha leído el archivo desde entrada
Explorando: (1, 1)
Explorando: (0, 1)
Fuera de los límites
Explorando: (2, 1)
Explorando: (1, 1)
Paso estrecho o ya visitado
Explorando: (3, 1)
Explorando: (2, 1)
Paso estrecho o ya visitado
Explorando: (4, 1)
Explorando: (3, 1)
Paso estrecho o ya visitado
Explorando: (5, 1)
Explorando: (4, 1)
Paso estrecho o ya visitado
Explorando: (6, 1)
Fuera de los límites
Explorando: (5, 2)
Explorando: (4, 2)
Explorando: (3, 2)
Explorando: (2, 2)
Paso estrecho o ya visitado
Explorando: (4, 2)
Paso estrecho o ya visitado
Explorando: (3, 3)
Explorando: (2, 3)
Explorando: (1, 3)
Paso estrecho o ya visitado
Explorando: (3, 3)
Paso estrecho o ya visitado
Explorando: (2, 4)
Explorando: (1, 4)
Explorando: (0, 4)
Fuera de los límites
Explorando: (2, 4)
Paso estrecho o ya visitado
Explorando: (1, 5)
Explorando: (0, 5)
Fuera de los límites
Explorando: (2, 5)
Explorando: (1, 5)
Paso estrecho o ya visitado
Explorando: (3, 5)
Explorando: (2, 5)
Paso estrecho o ya visitado
Explorando: (4, 5)
Explorando: (3, 5)
Paso estrecho o ya visitado
Explorando: (5, 5)
¡Tornillo encontrado!
```

```
¡Tornillo encontrado!  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Movimiento hacia atrás  
Se ha escrito en el archivo de salida
```

Para archivo resultado.txt

A screenshot of a text editor window with two tabs: 'prueba.txt' and 'resultado.txt'. The 'resultado.txt' tab is active, showing a list of coordinate pairs. The menu bar includes 'Archivo', 'Editar', and 'Ver'.

```
(5 , 5)  
(4 , 5)  
(3 , 5)  
(2 , 5)  
(1 , 5)  
(1 , 4)  
(2 , 4)  
(2 , 3)  
(3 , 3)  
(3 , 2)  
(4 , 2)  
(5 , 2)  
(5 , 1)  
(4 , 1)  
(3 , 1)  
(2 , 1)  
(1 , 1)
```

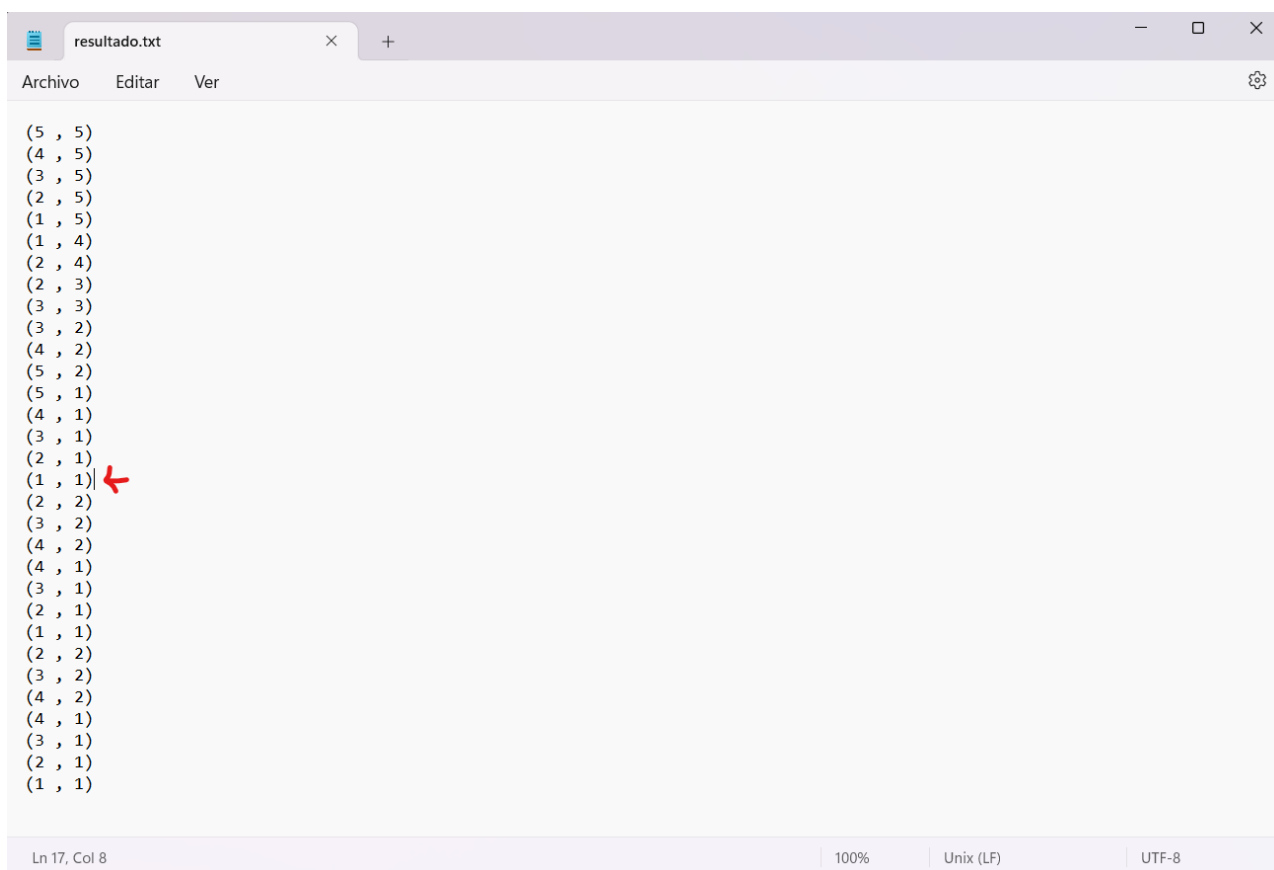
Para archivo prueba.txt

```
4  
3  
L L E  
L T L  
L L L  
L L E
```

```
C:\Users\lausa>java -jar robot.jar -t prueba.txt resultado.txt
Iniciando búsqueda con traza...
```

```
Se ha leído el archivo desde entrada
Explorando: (1, 1)
Explorando: (0, 1)
Fuera de los límites
Explorando: (2, 1)
Explorando: (1, 1)
Paso estrecho o ya visitado
Explorando: (3, 1)
Explorando: (2, 1)
Paso estrecho o ya visitado
Explorando: (4, 1)
Explorando: (3, 1)
Paso estrecho o ya visitado
Explorando: (5, 1)
Fuera de los límites
Explorando: (4, 2)
Explorando: (3, 2)
Explorando: (2, 2)
¡Tornillo encontrado!
Movimiento hacia atrás
Movimiento hacia atrás
Movimiento hacia atrás
Movimiento hacia atrás
Movimiento hacia atrás
Movimiento hacia atrás
Se ha escrito en el archivo de salida
```

Para archivo resultado.txt



```
resultado.txt
Archivo  Editar  Ver

(5 , 5)
(4 , 5)
(3 , 5)
(2 , 5)
(1 , 5)
(1 , 4)
(2 , 4)
(2 , 3)
(3 , 3)
(3 , 2)
(4 , 2)
(5 , 2)
(5 , 1)
(4 , 1)
(3 , 1)
(2 , 1)
(1 , 1)
(2 , 2)
(3 , 2)
(4 , 2)
(4 , 1)
(3 , 1)
(2 , 1)
(1 , 1)
(2 , 2)
(3 , 2)
(4 , 2)
(4 , 1)
(3 , 1)
(2 , 1)
(1 , 1)

Ln 17, Col 8  100%  Unix (LF)  UTF-8
```

Para archivo prueba.txt

```
4
4
L L E L
L E L L
L L L L
```

L L E L

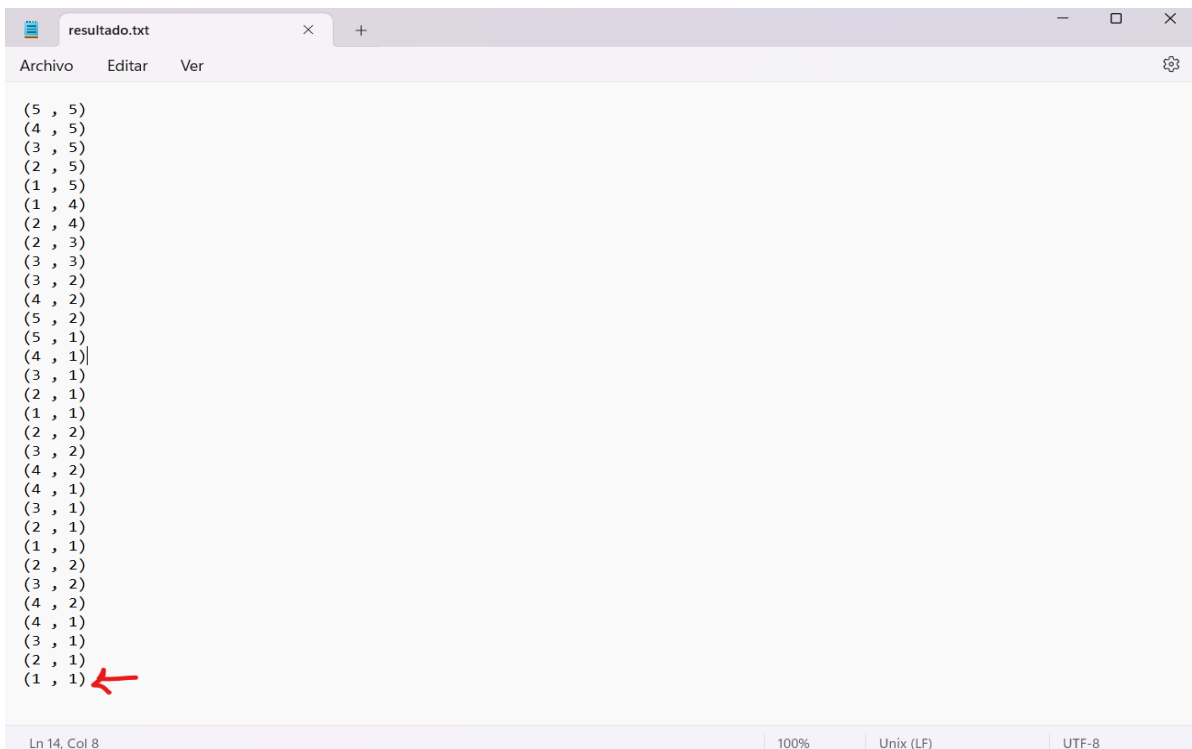
```
C:\Users\lausa>java -jar robot.jar -t prueba.txt resultado.txt
Iniciando búsqueda con traza...
```

```
Se ha leído el archivo desde entrada
```

```
Explorando: (1, 1)
Explorando: (0, 1)
Fuera de los límites
Explorando: (2, 1)
Explorando: (1, 1)
Paso estrecho o ya visitado
Explorando: (3, 1)
Explorando: (2, 1)
Paso estrecho o ya visitado
Explorando: (4, 1)
Explorando: (3, 1)
Paso estrecho o ya visitado
Explorando: (5, 1)
Fuera de los límites
Explorando: (4, 2)
Explorando: (3, 2)
Explorando: (2, 2)
Paso estrecho o ya visitado
Explorando: (4, 2)
Paso estrecho o ya visitado
Explorando: (3, 3)
Explorando: (2, 3)
Explorando: (1, 3)
Paso estrecho o ya visitado
Explorando: (3, 3)
Paso estrecho o ya visitado
Explorando: (2, 4)
Explorando: (1, 4)
Explorando: (0, 4)
Fuera de los límites
Explorando: (2, 4)
Paso estrecho o ya visitado
Explorando: (1, 5)
Fuera de los límites
Explorando: (3, 4)
Explorando: (2, 4)
```

```
Paso estrecho o ya visitado
Explorando: (1, 5)
Fuera de los límites
Explorando: (3, 4)
Explorando: (2, 4)
Paso estrecho o ya visitado
Explorando: (4, 4)
Explorando: (3, 4)
Paso estrecho o ya visitado
Explorando: (5, 4)
Fuera de los límites
Explorando: (4, 5)
Fuera de los límites
Explorando: (3, 5)
Fuera de los límites
Explorando: (2, 5)
Fuera de los límites
Explorando: (4, 3)
Paso estrecho o ya visitado
Explorando: (3, 4)
Paso estrecho o ya visitado
Explorando: (5, 2)
Fuera de los límites
Explorando: (4, 3)
Paso estrecho o ya visitado
Explorando: (3, 2)
Paso estrecho o ya visitado
Explorando: (2, 2)
Paso estrecho o ya visitado
Explorando: (1, 2)
Explorando: (0, 2)
Fuera de los límites
Explorando: (2, 2)
Paso estrecho o ya visitado
Explorando: (1, 3)
Paso estrecho o ya visitado
No se encontro un camino hasta el tornillo
Se ha escrito en el archivo de salida
```

Para archivo resultado.txt



```
(5 , 5)
(4 , 5)
(3 , 5)
(2 , 5)
(1 , 5)
(1 , 4)
(2 , 4)
(2 , 3)
(3 , 3)
(3 , 2)
(4 , 2)
(5 , 2)
(5 , 1)
(4 , 1)|
(3 , 1)
(2 , 1)
(1 , 1)
(2 , 2)
(3 , 2)
(4 , 2)
(4 , 1)
(3 , 1)
(2 , 1)
(1 , 1)
(2 , 2)
(3 , 2)
(4 , 2)
(4 , 1)
(3 , 1)
(2 , 1)
(1 , 1)
(2 , 1)
(1 , 1) ←
```

Ln 14, Col 8 100% Unix (LF) UTF-8

3.Un listado del código fuente completo.

Clase Principal:

Class Robot:

```
import java.util.List;
import java.util.Collections;
/**
 * Punto de entrada principal para la ejecución del programa.
 */
public class Robot
{
    /**
     * Realiza la lógica principal del programa. Verifica los argumentos
de entrada,
     * lee el edificio, resuelve el edificio y escribe los resultados.
     */
    public static void main(String[] args){

        // Verificar la cantidad correcta de argumentos
        if (args.length > 4) {
            FSalida.mostrarAyuda();
            return;
        }

        // Verificar si se proporciona el argumento -t para habilitar
trazas
        boolean trace = false;
        // boolean trace = args.length > 0 && args[0].equals("-t");
        int startIndex = 0;
        if (args.length > 0 && args[0].equals("-t")) {
            trace = true;
            startIndex = 1;
        }
        // Verificar la opción de ayuda
        if (args.length > startIndex && args[startIndex].equals("-h")) {
            FSalida.mostrarAyuda();
            return;
        }
        // Leer el nombre del archivo de entrada y salida
        String inputFile = args.length > startIndex &&
!args[startIndex].equals("-t") ? args[startIndex] : null;
        if (trace) {
            System.out.println("Iniciando búsqueda con traza...");
        }
        Edificio edificio;

        if(inputFile != null){
            // Leer desde el archivo de entrada
            System.out.print("\n Se ha leído el archivo desde entrada
\n");
        }
    }
}
```



```

        edificio = FEntrada.cargarEdificioDesdeArchivo(inputFile);
    }else{
        // Leer desde la entrada estándar
        edificio = FEntrada.leerEdificioDesdeConsola();
    }

    List<Coordenada> solucion = edificio.resolverEdificio(trace);
    Collections.reverse(solucion);
    String outputFile = (args.length > (startIndex + 1)) ?
args[startIndex + 1] : null;

    if(outputFile != null){
        // Escribir en el archivo de salida
        System.out.print("\n Se ha escrito en el archivo de salida
\n");
        FSalida.escribirSalida(outputFile, solucion);
    }else{
        // Imprimir la solución en la consola
        for(Coordenada coordenada: solucion){
            System.out.print("\n Se va a imprimir la solucion a
continucion: \n");
            System.out.print(coordenada);
        }
    }

}
}

```

Class Coordenada:

```

/**
 *Representa las coordenadas (x, y) en el espacio bidimensional.
 */
public class Coordenada
{
    private int x;
    private int y;

    /**
     * Constructor que inicializa las coordenadas.
     */
    public Coordenada(int x, int y)
    {
        this.x=x;
        this.y=y;
    }

    /**
     * Devuelve una representación en cadena de la coordenada en el
    formato (x, y).
     */
}

```

```

        public String toString(){
            return "("+x+" , "+y+")"+"\\n";
        }
    }
}

```

Class Edificio:

```

import java.util.List;
import java.util.ArrayList;
import java.util.Stack;

/**
 * Representa el edificio con pasillos libres (L), pasillos estrechos (E) y
 * la posición del tornillo (T).
 * Contiene métodos para resolver el edificio y encontrar el camino desde
 * la entrada hasta el tornillo.
 */
public class Edificio
{
    private char[][] datos;
    private boolean[][] visitados;
    private int filas;
    private int columnas;

    /**
     * Constructor que inicializa el edificio con sus dimensiones y
     datos.
     */
    public Edificio(int filas, int columnas, char[][] datos)
    {
        this.filas=filas;
        this.columnas=columnas;
        this.datos=datos;
        this.visitados=new boolean[filas][columnas];
    }

    /**
     * Método principal que inicia la búsqueda del tornillo y devuelve
     una lista de coordenadas
     * que representan el camino desde la entrada hasta el tornillo.
     */
    public List<Coordenada> resolverEdificio(boolean trace){
        List<Coordenada> solucion = new ArrayList<>();
        boolean exito = buscarTornillo(1, 1, solucion, trace);

        if(!exito){
            System.out.print("No se encontro un camino hasta el
tornillo");
        }

        return solucion;
    }
}
/**

```

* Método que inicia la búsqueda del tornillo utilizando un enfoque de vuelta atrás.

```
*/
private boolean buscarTornillo(int x, int y, List<Coordenada>
solucion, boolean trace){
    Stack<Coordenada> camino = new Stack<>();
    boolean exito = buscarRecursoivo(x, y, camino,trace);

    if(exito){
        while(!camino.isEmpty()){
            solucion.add(camino.pop());
        }
    }
    return exito;
}
/**
 * Método recursivo que realiza la búsqueda del tornillo. Se utiliza
en conjunto
 * con buscarTornillo.
 */
private boolean buscarRecursoivo(int x, int y, Stack<Coordenada>
camino, boolean trace){
    if (trace) {
        System.out.println("Explorando: (" + x + ", " + y + ")");
    }

    if(x<1 || x>filas || y<1 || y>columnas){
        if (trace) {
            System.out.println("Fuera de los límites");
        }
        return false;
    }
    if(datos[x-1][y-1] == 'T'){
        camino.push(new Coordenada(x,y));
        if (trace) {
            System.out.println("¡Tornillo encontrado!");
        }
        return true;
    }
    if(datos[x-1][y-1] == 'E' || visitados[x-1][y-1]){
        if (trace) {
            System.out.println("Paso estrecho o ya visitado");
        }
        return false;
    }
    visitados[x-1][y-1] = true;

    if(buscarRecursoivo(x-1, y, camino, trace) ||
        buscarRecursoivo(x+1, y, camino, trace) ||
        buscarRecursoivo(x, y+1, camino, trace)){
        camino.push(new Coordenada(x,y));
        if (trace) {
            System.out.println("Movimiento hacia atrás");
        }
    }
}
```

```

        return true;
    }
    return false;
}
}

```

Class FEntrada:

```

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.FileReader;

/**
 * Maneja la entrada de datos, ya sea desde un archivo o desde la consola.
 */

public class FEntrada
{
    /**
     * Lee el edificio desde un archivo y devuelve una instancia de la
     clase Edificio.
     */
    public static Edificio cargarEdificioDesdeArchivo(String
nombreArchivo) {
        try (BufferedReader br = new BufferedReader(new
FileReader(nombreArchivo))) {
            int filas = Integer.parseInt(br.readLine());
            int columnas = Integer.parseInt(br.readLine());

            char[][] datos = new char[filas][columnas];

            for (int i = 0; i < filas; i++) {
                String fila = br.readLine();
                datos[i] = fila.replaceAll("\\s", "").toCharArray();
            }

            return new Edificio(filas, columnas, datos);
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }
    /**
     * Lee el edificio desde la consola y devuelve una instancia de la
     clase Edificio.
     */
    public static Edificio leerEdificioDesdeConsola(){
        try(BufferedReader br = new BufferedReader(new
InputStreamReader(System.in))){
            System.out.print("1.Introduce el numero de filas: \n");
            System.out.print("2.Introduce el numero de columnas:\n");
            System.out.print("3.Introduce los datos del edificio
(L,E,T):\n");

```

```

        int filas=Integer.parseInt(br.readLine());
        int columnas=Integer.parseInt(br.readLine());
        char[][] datos = new char[filas][columnas];

        for(int i = 0; i<filas; i++){
            String fila = br.readLine();
            datos[i] = fila.replaceAll("\\s", "").toCharArray();
        }

        return new Edificio(filas, columnas, datos);
    }catch(IOException e){
        e.printStackTrace();
        return null;
    }
}
}

```

Class FSalida:

```

import java.io.FileWriter;
import java.io.IOException;
import java.util.List;

/**
 * Maneja la salida de resultados, ya sea imprimiendo en la consola o
 * escribiendo en un archivo.
 */
public class FSalida
{
    /**
     * Escribe las coordenadas de la solución en un archivo,
     * anexando al contenido existente si el archivo ya existe.
     */
    public static void escribirSalida(String outputFile, List<Coordenada>
solucion) {
        // El segundo parámetro indica que se debe anexar al archivo
        existente
        try (FileWriter writer = new FileWriter(outputFile, true)) {
            for (Coordenada coordenada : solucion) {
                writer.write(coordenada.toString());
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    /**
     * Objetivo: Muestra información de ayuda sobre la sintaxis del
     * programa y las opciones disponibles.
     */
    public static void mostrarAyuda(){
        System.out.println("SINTAXIS:java -jar CambioDinamica.jar [-t][-h]
[fichero entrada] [fichero salida]");
    }
}

```

```
        System.out.println("-t Traza el algoritmo");
        System.out.println("-h Muestra esta ayuda");
        System.out.println("[fichero entrada] Nombre del fichero de
entrada");
        System.out.println("[fichero salida] Nombre del fichero de
salida");
        System.exit(0);
    }
}
```