



Centro de Ciencias Básicas

Inteligencia Artificial

Laberinto Inteligente

Ingeniería en Computación Inteligente

Semestre 3° A

agosto-diciembre 2022

Integrantes:

Durón Láriz Miguel Ángel

Elías del Hoyo César Eduardo

López Flores Kandy Fabiola

Ortiz Quiroz Ángel David

Rivera Delgadillo Ximena

Sandoval Pérez José Luis

Saucedo Ortega Diego Emanuel

Torres Macías Carlos Daniel

ID

331992

262045

326912

261481

261261

261731

261230

244543

Docente: Doc. Alejandro Padilla Díaz

Fecha de entrega: 3 de diciembre del 2022

Contenido

Introducción.....	2
Desarrollo.....	4
Conclusión.....	28
Bibliografía	29

En este documento debemos poner **portada, desarrollo (imágenes y código), conclusiones particulares y de equipo.**

Introducción

Durante este proyecto, realizamos un laberinto inteligente el cual tenía como base el ejercicio del algoritmo genético. El objetivo de dicho problema era crear un algoritmo que encontrara un camino o dos, dentro de una matriz de 40×40 , que permitiera tener una salida al final de este. Las indicaciones del proyecto eran las siguientes:

- Creación de una matriz de 40×40 , la cual contendrá de 33 a 45% de 1s
- El laberinto será generado de manera aleatoria, valiéndose además de técnicas como la mutación, donde el único valor fijo será el punto de partida en la entrada $[0,0]$ de la matriz
- Generar mínimo dos caminos que lleven a la meta, para determinar el mejor de ambos, hacer uso de alguno de los métodos de búsqueda analizados en el curso.

Previo a la realización del proyecto, tuvimos que investigar un poco sobre los algoritmos genéticos (realizado en segundo parcial de este semestre) y la investigación de laberintos inteligentes.

Los algoritmos inteligentes son un tipo de programa informático que permite a los ordenadores realizar tareas que normalmente requieren inteligencia humana. Estos programas permiten que los ordenadores aprendan y se adapten a su entorno, reconozcan objetos y sonidos y procesen la información del mismo modo que lo hacen los seres humanos.

El laberinto inteligente permite encontrar una solución óptima mediante métodos de búsqueda y apoyándose con el algoritmo genético, el cual nos permitirá realizar mutaciones para encontrar un camino a la salida.

Para ello realizamos 2 algoritmos que nos permitieron comparar las diferencias que encontramos entre ambos programas y, a su vez, obtener una solución correcta.

Primer programa

Pasos para la realización del código:

- Es decir que nuestro laberinto siempre inicie de la esquina superior izquierda.

2. Después de generar la matriz aleatoria con nuestra puerta utilizamos el método A* para buscar si en este ya se encuentra un camino que una la primera fila con la última. En otras palabras, que recorra de nuestra puerta a alguna salida (1) de la última fila. De ser que no continuamos y mutamos.

3. Mutamos para ir generando un camino desde nuestra puerta hasta la última fila, para esto localizamos el último punto donde se genera un camino y a partir de esto mutamos un valor aleatorio inferior o derecho, para que de esta forma forcemos a que el camino se acerque a la última fila.

Guardamos las posiciones de los (0) mutados para compensar en la fila según el porcentaje que se haya seleccionado.

```

Laberinto de ida
[1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[2, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0]
[3, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1]
[4, 5, 6, 7, 8, 9, 10, 11, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 12, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1]
[0, 1, 1, 0, 1, 0, 0, 13, 14, 15, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 1, 1, 0, 0, 16, 17, 18, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1]
[1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 19, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0]
[1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 20, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1]
[0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 21, 22, 23, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 25, 26, 27, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0]
[0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 28, 29, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0]
[0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 30, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0]
[0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 31, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0]
[0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 32, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 33, 34, 35, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0]
[1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 36, 37, 38, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 39, 40, 41, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1]
[0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 42, 43, 44, 45, 46, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1]
[1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 47, 48, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1]
[1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 49, 50, 51, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 52, 0, 1, 0, 1, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 53, 0, 0, 0, 0, 1, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 54, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0]
[1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 55, 56, 0, 0, 1, 1, 0, 0, 0, 1]
[0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 57, 58, 59, 60, 61, 0, 0, 0, 0, 0]
[0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 62, 63, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 64, 65, 66, 0, 0]
[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 67, 0]
[0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 68, 1]
[0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 69, 0]
[0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 70, 0]
[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 71, 0]
[1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 72, 1]
[0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 80, 79, 78, 77, 0, 73, 1]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 81, 82, 0, 81, 82, 0, 76, 75, 74, 1]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 83, 0, 1, 1, 0, 0, 0]
[0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 84, 85, 0, 1, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 86, 1, 0, 0, 0, 0]
[1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 87, 0, 1, 1, 1, 0, 0, 0]

```

4. Posteriormente volvemos a aplicar A* para localizar que nuestro camino sea el mejor y obtener al mismo tiempo el peso de este.

```

Camino optimo
[1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[2, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0]
[3, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1]
[4, 5, 6, 7, 8, 9, 10, 11, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 1, 0, 12, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1]
[0, 1, 1, 0, 1, 0, 0, 13, 14, 15, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 1, 1, 0, 0, 16, 17, 18, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1]
[1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 19, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0]
[1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 20, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1]
[0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 21, 22, 23, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 24, 25, 26, 27, 28, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0]
[0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 29, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 30, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0]
[0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 31, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0]
[0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 32, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1]
[0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 33, 34, 35, 36, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0]
[1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 37, 38, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 39, 40, 41, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1]
[0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 42, 43, 44, 45, 46, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1]
[1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 47, 48, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1]
[1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 49, 50, 51, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 52, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 53, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 54, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0]
[1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 55, 56, 0, 0, 1, 1, 0, 0, 0, 0, 1]
[0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 57, 58, 59, 60, 61, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 62, 63, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 64, 65, 66, 0, 0]
[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 67, 0]
[0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 68, 1]
[0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 69, 0]
[0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 70, 0]
[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 71, 0]
[1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 72, 1]
[0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 79, 78, 77, 0, 73, 1]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 80, 0, 76, 75, 74, 1]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 81, 0, 1, 1, 0, 0]
[0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 82, 83, 0, 1, 1, 0]
[0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 84, 1, 0, 0, 0]
[1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 85, 0, 1, 1, 1]

```

5. Imprimimos las coordenadas del camino desarrollado:

Caminito

```

[1,0]
[2,0]
[3,0]
[3,1]
[3,2]
[3,3]
[3,4]
[3,5]
[3,6]
[3,7]
[4,7]
[5,7]
[5,8]
[5,9]
[6,9]
[6,10]
[6,11]

```

[7,11]
[8,11]
[9,11]
[9,12]
[9,13]
[10,13]
[10,14]
[10,15]
[10,16]
[10,17]
[11,17]
[12,17]
[13,17]
[14,17]
[15,17]
[15,18]
[15,19]
[15,20]
[16,20]
[16,21]
[17,21]
[17,22]
[17,23]
[18,23]
[18,24]
[18,25]
[18,26]
[18,27]
[19,27]
[19,28]
[20,28]
[20,29]
[20,30]
[21,30]
[22,30]
[23,30]
[24,30]
[24,31]
[25,31]
[25,32]
[25,33]
[25,34]
[25,35]
[26,35]
[26,36]
[27,36]

[27,37]
[27,38]
[28,38]
[29,38]
[30,38]
[31,38]
[32,38]
[33,38]
[34,38]
[35,38]
[35,37]
[35,36]
[34,36]
[34,35]
[34,34]
[35,34]
[36,34]
[37,34]
[37,35]
[38,35]
[39,35]

6. Guardamos las coordenadas de la salida y repetimos el proceso de manera inversa con la condicional de que la salida de este nuevo camino sea la entrada [0][0] de la matriz.

7. De esta forma finalmente obtenemos un segundo camino.

```
Laborinto de vuelta
```

```
[69, 68, 67, 66, 65, 64, 63, 62, 61, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 1, 1, 0, 60, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 59, 58, 57, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1]
[1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 56, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 55, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1]
[0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 54, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 53, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1]
[1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 52, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0]
[1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 51, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 50, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 49, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 48, 47, 46, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 45, 44, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0]
[0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 43, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0]
[0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 42, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 40, 41, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0]
[1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 38, 39, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 35, 36, 37, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1]
[0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 34, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1]
[1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 33, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0]
[1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 32, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 0, 0, 31, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 30, 29, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 28, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0]
[1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 27, 26, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 25, 24, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0]
[0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 23, 22, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 21, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 20, 19, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 18, 17, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1]
[0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 16, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1]
[0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 15, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 
```

8. Finalmente volvemos a emplear el A* para determinar el mejor camino encontrado e imprimirlo.

```

Camino optimo
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 1, 1, 0, 10, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 11, 12, 13, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 14, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 15, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 16, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 17, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 18, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 19, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 20, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 21, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 22, 23, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 24, 25, 26, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 27, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 30, 29, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 33, 32, 31, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 35, 34, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 36, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 37, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 38, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 39, 40, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 41, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 42, 43, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 44, 45, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 46, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 47, 48, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 49, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 50, 51, 52, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 54, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 55, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 56, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 57, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 60, 59, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 62, 61, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 63, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 64, 65, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 66, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

```

9. De esta forma ya tenemos un laberinto con dos caminos y un camino seleccionado como el mejor.

10. Finalmente, solo imprimimos el laberinto resultante de manera grafica para poder identificar todo con mayor claridad.



Capturas de código documentado de Python

```
Extension: Python Extension Pack  laberinto.py 1 ●
D: > 52449 > Documents > laberinto.py > imprimir
1  # -*- coding: cp1252 -*-
2  import wx #libreria utilizada para los elementos graficos que generan el laberinto final
3  import random
4  import copy
5
6  global matriz
7  matriz = []
8
9
10 #Almacena el primer camino, el cual se genera teniendo como punto de partida la coordenada [0][0]
11
12 class Node:
13     def __init__(self, coords, pre, costo):
14         self.coords = coords
15         self.pre = pre
16         self.costo = costo
17         self.estima = 39 - coords[0]
18
19     def __str__(self):
20         return '['+str(self.coords[0])+','+str(self.coords[1])+']'
21
22
23 #almacena el segundo camino generado, por ello el que tenga el costo, la coordenada
24 # en que ese encuentra, el valor anterior y se almacena a sí mismo
25 class Node2:
26     def __init__(self, coords, pre, costo):
27         self.coords = coords
28         self.pre = pre
29         self.costo = costo
30         self.estima = coords[0] + coords[1]
31
32     def __str__(self):
33         return '['+str(self.coords[0])+','+str(self.coords[1])+']'
34
35
36 #establece los valores aleatorios con los porcentajes que dió padilla
37 def inicializa():
38     indices = list(range(40))
39
40     for idx in range(40):
41         fila = []
```

```

41     fila = []
42     aleatorio = random.random()*12+33
43     unos = int(40*aleatorio*0.01)
44     lugares = random.sample(indices, unos)
45
46     for jdx in range(40):
47         if jdx in lugares:
48             fila.append(1)
49         else:
50             fila.append(0)
51
52     matriz.append(fila)
53
54
55     # Forzar [0][0] a ser 1
56     if matriz[0][0]==0:
57         matriz[0][0]=1
58
59     for idx in range(39,1,-1):
60         if matriz[0][idx] == 1:
61             matriz[0][idx] = 0
62             break
63
64     actual = Node((0,0),None,0)
65
66     return actual
67
68
69 def imprime(actual):
70     copia = copy.deepcopy(matriz)
71     caminito = [] #almacena las coordenadas de los caminos que fueron generados para posteriormente
72     #comparar y evitar que se generen dos caminos iguales
73     recorrer = actual
74
75     while(recorrer.pre):
76         caminito = [recorrer.coords] + caminito
77         recorrer = recorrer.pre
78
79     for n,coords in enumerate(caminito):
80         copia[coords[0]][coords[1]] = n + 2
81

```

```

D:\732449\Documents\laberinto.py / Inicializa
80     copia[coords[0]][coords[1]] = n + 2
81
82     for fila in copia:
83         print(fila)
84
85     #busca los adyacentes y ver si genera conexiones
86     def buscar(actual):
87         recorrer = actual
88         mayor = 0
89         regresar = actual
90
91         while(recorrer.pre):
92
93             if recorrer.coords[0] > mayor:
94                 regresar = recorrer
95                 mayor = recorrer.coords[0]
96
97             recorrer = recorrer.pre
98
99         while(actual != regresar):
100             actual = actual.pre
101
102         return actual
103
104
105     def buscarInv(actual):
106         recorrer = actual
107         menor = 1000
108         regresar = actual
109
110         while(recorrer.pre):
111
112             if recorrer.coords[0] < menor:
113                 regresar = recorrer
114                 menor = recorrer.coords[0]
115
116             recorrer = recorrer.pre
117
118         while(actual != regresar):
119             actual = actual.pre
120

```

```

D:\S2449\Documents> taberinto.py > Inicializa
118 while (actual != regresar):
119     actual = actual.pre
120
121 return actual
122
123
124 #compensa los valores y crea el camino final
125 def mutar(actual, pasados, candidatos):
126     actual = buscar(actual)
127
128     while actual.coords[0]<39 and matriz[actual.coords[0]+1][actual.coords[1]] == 1:
129         nuevo = Node((actual.coords[0]+1, actual.coords[1]), actual, actual.costo+1)
130         actual = nuevo
131     candis = []
132
133     if actual.coords[0]<39:
134         if matriz[actual.coords[0]+1][actual.coords[1]] == 0:
135             candis.append(Node((actual.coords[0]+1, actual.coords[1]), actual, actual.costo+1))
136
137     if actual.coords[1]<39:
138         if matriz[actual.coords[0]][actual.coords[1]+1] == 0:
139             candis.append(Node((actual.coords[0], actual.coords[1]+1), actual, actual.costo+1))
140
141     if candis:
142         nuevo = random.choice(candis)
143         compensar = []
144
145         for idx in range(40):
146             if matriz[nuevo.coords[0]][idx] == 1:
147                 repe = False
148                 for pasado in pasados:
149                     if pasado.coords == (nuevo.coords[0],idx):
150                         repe = True
151                         break
152                 if not repe:
153                     compensar.append(idx)
154         if compensar:
155             cero = random.choice(compensar)
156             matriz[nuevo.coords[0]][cero] = 0
157
158     matriz[nuevo.coords[0]][nuevo.coords[1]] = 1
159

```



```

157
158     matriz[nuevo.coords[0]][nuevo.coords[1]] = 1
159
160     actual = nuevo
161     pasados = pasados.union({actual})
162     candidatos = {actual}
163
164     return actual, pasados, candidatos
165
166 def mutarInv(actual, pasados, candidatos):
167     actual = buscarInv(actual)
168
169     while actual.coords[0]>0 and matriz[actual.coords[0]-1][actual.coords[1]] == 1:
170         nuevo = Node2((actual.coords[0]-1, actual.coords[1]), actual, actual.costo+1)
171         actual = nuevo
172     candis = []
173
174     if actual.coords[0]>0:
175         if matriz[actual.coords[0]-1][actual.coords[1]] == 0:
176             candis.append(Node2((actual.coords[0]-1, actual.coords[1]), actual, actual.costo+1))
177
178     if actual.coords[1]>0:
179         if matriz[actual.coords[0]][actual.coords[1]-1] == 0:
180             candis.append(Node2((actual.coords[0], actual.coords[1]-1), actual, actual.costo+1))
181
182     if candis:
183         nuevo = random.choice(candis)
184         compensar = []
185
186         for idx in range(40):
187             if matriz[nuevo.coords[0]][idx] == 1:
188                 repe = False
189                 for pasado in pasados:
190                     if pasado.coords == (nuevo.coords[0],idx):
191                         repe = True
192                         break
193                 if not repe:
194                     compensar.append(idx)
195         if compensar:
196             cero = random.choice(compensar)
197             if nuevo.coords[0] > 5 or cero > 0:
198                 matriz[nuevo.coords[0]][cero] = 0

```

```

197         if nuevo.coords[0] > 5 or cero > 0:
198             | matriz[nuevo.coords[0]][cero] = 0
199
200         matriz[nuevo.coords[0]][nuevo.coords[1]] = 1
201
202         actual = nuevo
203         pasados = pasados.union({actual})
204         candidatos = {actual}
205
206     return actual, pasados, candidatos
207
208
209 #solo busca el mejor
210 def estrella(actual, candidatos, pasados):
211     count = 0
212
213     while candidatos and actual.coords[0] < 39 and count < 10000:
214         count += 1
215         hijos = []
216
217         # Analizando vecinos de actual
218         # Agregamos a hijos los vecinos con 1
219
220         if actual.coords[1]>0:
221             | if matriz[actual.coords[0]][actual.coords[1]-1] == 1:
222                 | hijos.append(Node((actual.coords[0], actual.coords[1]-1), actual, actual.costo+1))
223         if actual.coords[1]<39:
224             | if matriz[actual.coords[0]][actual.coords[1]+1] == 1:
225                 | hijos.append(Node((actual.coords[0], actual.coords[1]+1), actual, actual.costo+1))
226         if actual.coords[0]>0:
227             | if matriz[actual.coords[0]-1][actual.coords[1]] == 1:
228                 | hijos.append(Node((actual.coords[0]-1, actual.coords[1]), actual, actual.costo+1))
229         if actual.coords[0]<39:
230             | if matriz[actual.coords[0]+1][actual.coords[1]] == 1:
231                 | hijos.append(Node((actual.coords[0]+1, actual.coords[1]), actual, actual.costo+1))
232
233         # Actual ya no es candidato
234         candidatos = candidatos - {actual}
235
236         # Si hubo hijos, vemos que no sean repetidos
237         for hijo in hijos:
238             |

```

```

235
236     # Si hubo hijos, vemos que no sean repetidos
237     for hijo in hijos:
238         repe = False
239         for pasado in pasados:
240             if pasado.coords == hijo.coords:
241                 repe = True
242                 break
243         if not repe:
244             candidatos = candidatos.union({hijo})
245
246     # Si hay candidatos, buscamos el mejor
247     if candidatos:
248         menor = 10000
249         for candi in candidatos:
250             if candi != actual.pre and candi.costo + candi.estima < menor:
251                 menor = candi.costo + candi.estima
252                 actual = candi
253
254     pasados = pasados.union({actual})
255
256     return actual, candidatos, pasados
257
258 def estrellaInv(actual, candidatos, pasados):
259     count = 0
260
261     while candidatos and (actual.coords[0] > 0 or actual.coords[1] > 0) and count < 100000:
262         count += 1
263         hijos = []
264
265         # Analizando vecinos de actual
266         # Agregamos a hijos los vecinos con 1
267
268         if actual.coords[1]>0:
269             if matriz[actual.coords[0]][actual.coords[1]-1] == 1:
270                 hijos.append(Node2((actual.coords[0], actual.coords[1]-1), actual, actual.costo+1))
271         if actual.coords[1]<39:
272             if matriz[actual.coords[0]][actual.coords[1]+1] == 1:
273                 hijos.append(Node2((actual.coords[0], actual.coords[1]+1), actual, actual.costo+1))
274         if actual.coords[0]>0:
275             if matriz[actual.coords[0]-1][actual.coords[1]] == 1:
276                 hijos.append(Node2((actual.coords[0]-1, actual.coords[1]), actual, actual.costo+1))

```

```

275         if matriz[actual.coords[0]-1][actual.coords[1]] == 1:
276             hijos.append(Node2((actual.coords[0]-1, actual.coords[1]), actual, actual.cost+1))
277     if actual.coords[0]<39:
278         if matriz[actual.coords[0]+1][actual.coords[1]] == 1:
279             hijos.append(Node2((actual.coords[0]+1, actual.coords[1]), actual, actual.cost+1))
280
281     # Actual ya no es candidato
282     candidatos = candidatos - {actual}
283
284     # Si hubo hijos, vemos que no sean repetidos
285     for hijo in hijos:
286         repe = False
287         for pasado in pasados:
288             if pasado.coords == hijo.coords:
289                 repe = True
290                 break
291         if not repe:
292             candidatos = candidatos.union({hijo})
293
294     # Si hay candidatos, buscamos el mejor
295     if candidatos:
296         menor = 10000
297         for candi in candidatos:
298             if candi != actual.pre and candi.cost+ 39 - candi.estima < menor:
299                 menor = candi.cost+ candi.estima
300                 actual = candi
301
302     pasados = pasados.union({actual})
303
304     return actual, candidatos, pasados
305 #se encarga de forzar el camino en caso de que no haya generado ya un camino en el paso inicial
306 def creaCamino(actual, candidatos, pasados):
307     count = 0
308
309     while actual.coords[0] < 39 and count < 10000:
310         count += 1
311         actual, candidatos, pasados = estrella(actual, candidatos, pasados)
312
313     # Si no hemos terminado, hacemos nuevo camino
314     if actual.coords[0] < 39:
315         actual, pasados, candidatos = mutar(actual, pasados, candidatos)

```

```

314         if actual.coords[0] < 39:
315             actual, pasados, candidatos = mutar(actual, pasados, candidatos)
316
317     return actual, candidatos, pasados
318
319 def creaSegundo(actual, candidatos, pasados):
320     count = 0
321
322     while (actual.coords[0] > 0 or actual.coords[1] > 0) and count < 100000:
323         count += 1
324         actual, candidatos, pasados = estrellaInv(actual, candidatos, pasados)
325
326         # Si no hemos terminado, hacemos nuevo camino
327         if actual.coords[0] > 0 or actual.coords[1] > 0:
328             actual, pasados, candidatos = mutarInv(actual, pasados, candidatos)
329
330     return actual, candidatos, pasados
331
332
333 #genera los gráficos del laberinto final
334 class LabeGraph(wx.Frame):
335
336     def __init__(self, parent, title, style=wx.SYSTEM_MENU | wx.CLOSE_BOX | wx.CAPTION):
337         super(LabeGraph, self).__init__(parent, title=title)
338
339         self.InitUI()
340         self.Centre()
341
342
343     def InitUI(self):
344
345         vbox = wx.BoxSizer(wx.VERTICAL)
346         gs = wx.GridSizer(40, 40, 1, 1)
347         for idx in range(40):
348             for jdx in range(40):
349                 text = wx.StaticText(self, -1, "11")
350                 if matriz[idx][jdx]:
351                     text.SetBackgroundColour('white')
352                     text.SetForegroundColour('white')
353                 else:
354                     text.SetBackgroundColour('black')
355                     text.SetForegroundColour('black')

```

```

354         text.SetBackgroundColour('black')
355         text.SetForegroundColour('black')
356         gs.Add(text)
357
358
359         vbox.Add(gs, proportion=1, flag=wx.EXPAND)
360         self.SetSizer(vbox)
361
362
363     def main():
364
365         actual = inicializa()
366
367         print("\n*****\n")
368         print("    Laberinto inicial\n")
369         imprime(actual)
370
371         candidatos = {actual}
372         pasados = {actual}
373
374         actual, candidatos, pasados = creaCamino(actual, candidatos, pasados)
375
376         print("\n*****\n")
377         print("    Laberinto de ida \n")
378         imprime(actual)
379
380         actual = Node((0,0),None,0)
381         candidatos = {actual}
382         pasados = {actual}
383
384         actual, candidatos, pasados = estrella(actual, candidatos, pasados)
385         print("\n*****\n")
386         print("    Camino optimo\n")
387         imprime(actual)
388
389         caminito = []
390         recorrer = actual
391
392         while(recorrer.pre):
393             caminito = [recorrer] + caminito
394             recorrer = recorrer.pre
395

```

```

393     caminito = [recorrer] + caminito
394     recorrer = recorrer.pre
395
396     print ("Caminito")
397     for ele in caminito:
398         print (ele)
399
400     # # # Va pa atras
401     pasados = set()
402     for nodo in caminito:
403         if nodo.coords[0]>5 or nodo.coords[1]>5:
404             pasados = pasados.union({Node2(nodo.coords,None,0)})
405
406     candidatos = set()
407     for idx in range(40):
408         if idx != actual.coords[1] and matriz[39][idx] == 1:
409             candidatos = candidatos.union({Node2((39,idx),None,0)})
410
411     pasados = pasados.union(candidatos)
412     actual = random.choice(list(candidatos))
413
414     actual, candidatos, pasados = creaSegundo(actual, candidatos, pasados)
415
416     print("Reversa")
417     print(actual.coords, actual.costo, actual.estima, actual.pre.coords)
418     print("\n*****\n")
419     print("    Laberinto de vuelta\n")
420     imprime(actual)
421
422     actual = Node((0,0),None,0)
423     candidatos = {actual}
424     pasados = {actual}
425
426     actual, candidatos, pasados = estrella(actual, candidatos, pasados)
427     print("\n*****\n")
428     print("    Camino optimo\n")
429     imprime(actual)
430
431     caminito = []
432     recorrer = actual
433

```

```

431     caminito = []
432     recorrer = actual
433
434     while(recorrer.pre):
435         |     caminito = [recorrer] + caminito
436         |     recorrer = recorrer.pre
437
438     print ("Caminito")
439     for ele in caminito:
440         |     print (ele)
441
442     app = wx.App()
443     ex = LabeGraph(None, title='Laberinto')
444     ex.Show()
445     app.MainLoop()
446
447
448
449     if __name__ == '__main__':
450         |     main()
451

```


Después de hacer el llenado de la matriz, pasamos a la función "hacerunhuecoenlapared". Esta función nos permitirá obtener huecos para poder seguir avanzando en el laberinto. Esto se realizó usando mutación. Se hizo lo siguiente:

Se obtuvo un porcentaje para tomar en cuenta la posición del hueco a realizar con los siguientes parámetros:

- 50% de probabilidad para mutar en la casilla de abajo
- 40% de probabilidad para mutar en la casilla de la derecha
- 10% de probabilidad para mutar en la casilla de la izquierda

Esto con el fin de obtener de manera aleatoria, mediante las probabilidades, un camino que no sea muy monótono y siempre en dirección hacia la derecha abajo. Sin embargo, no se usó un porcentaje de hacer hueco arriba, porque entonces el laberinto se haría tedioso y en ocasiones tendría a tardarse más.

Al avanzar, si la casilla es un hueco, simplemente se desplazará hacia ese espacio. Pero si tenemos un muro, dicho muro será movido a un espacio situado en la misma fila, pero sin que este afecte al camino ya establecido hasta el momento, para entonces crear un hueco y seguir avanzando. Este procedimiento seguirá hasta llegar a la letra f, el final del laberinto. En cierta ocasión donde lleguemos al borde de la matriz y aún no encontremos la solución, se realizará en forma recta, desde el borde hasta el fin del laberinto para concluir más rápido la salida. Hasta aquí se ha encontrado la solución en base a los huecos realizados.

```
while(x!=fx || y!=fy){
    casilla[x][y].esCamino = true;

    if(casilla[x][y].simbolo == 'w'){
        //si el camino se topa con un muro mueve ese muro en la misma fila
        //debe revisar que no sea parte de un camino y que no sea un muro
        int mover = rand()%8;
        while(casilla[mover][y].simbolo == 'w' || casilla[mover][y].esCamino){ mover = rand()%8; }

        casilla[mover][y].simbolo = "w";//se mueve la casilla
    }
    //si la casilla es un muro
    casilla[x][y].simbolo = '.';

    int decision = rand()%10;

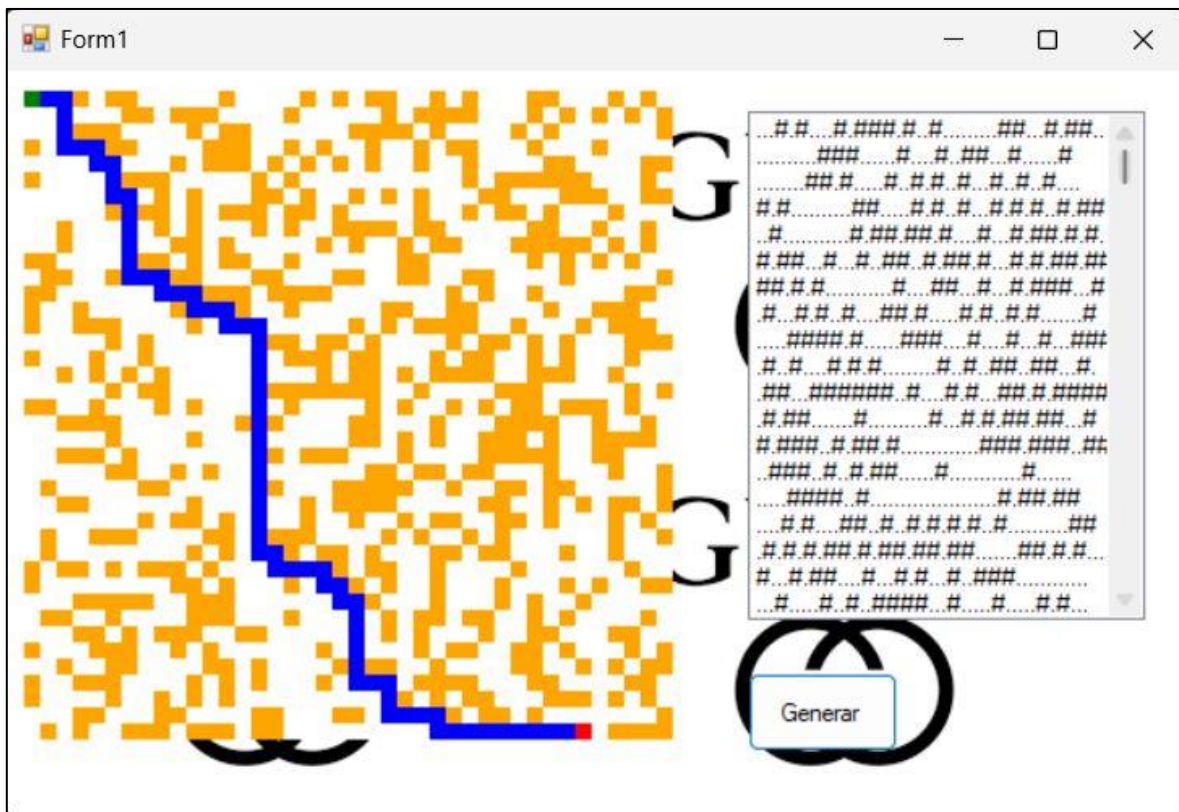
    if(x == fx){
        if(y>fy){ y--; }
        else{ y++; }
    }
    else{
        if(decision<=4 abajo && xN-1){ x++; } //hacia abajo
        else if(decision<=4 der && yN){ y++; } //preferiblemente a la derecha
        else if(y>0){ y--; } // a la izquierda
        else if(xN-1){ x++; } // si se llega a un limite mejor bajar
    }

}

} //despeja camino
```

[illegible]

26



Conclusión

Con este proyecto podemos concluir que los algoritmos de búsqueda tienen un sinfín de aplicaciones, en este proyecto se abordó darle solución a un laberinto, cuyo objetivo era encontrar la salida. Nos pareció interesante como es que cada algoritmo de búsqueda trabaja de una manera distinta, al hacer las comparaciones de ambos programas nos dimos dar cuenta que ambos son opuestos. Uno obtiene la solución óptima y otro obtiene la solución más rápida. Sin duda un proyecto muy divertido de realizar y gracias a la ayuda de los entornos gráficos todo fue más fácil y digerible.

Conclusión personal

Los laberintos siempre me han parecido interesantes, crear un laberinto fue de las cosas que más rescato de este semestre, me sorprendió como es que logre desarrollar mis capacidades para poder desarrollar un código que determinara la solución de un laberinto. Sin duda de mis proyectos favoritos.

Bibliografía

Difference between Breadth Search (BFS) and Deep Search (DFS). (2020, 25 mayo).

Encora. <https://www.encora.com/es/blog/dfs-vs-bfs>

Jain, S. (2021, 5 marzo). *The Insider's Guide to A* Algorithm in Python*. Python Pool.

<https://www.pythonpool.com/a-star-algorithm-python/>

Miembros de clase en C++ Variables y Métodos - Clases y Objetos en C++ (Práctica 1).

(s. f.). CodinGame. <https://www.codingame.com/playgrounds/50557/clases-y-objetos-en-c-practica-1/miembros-de-clase-en-c-variables-y-metodos>