El recorrido del caballo

## Índice

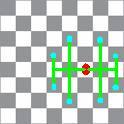
1. Análisis del problema
2. Solución
3. Explicación algoritmo y Pseudocódigo
4. Eficiencia Teórica
5. Caso Real
6. Instrucciones para compilar y ejecutar el código

# Análisis del problema

En esta práctica se nos presentaba el problema del recorrido del caballo. Dicho problema consta en hacer que la ficha del ajedrez del caballo pase por todas las casillas del tablero sin que pise dos veces una misma casilla y sin permitir que algún movimiento realizado por el mismo lo saque fuera de los límites del tablero. Para ello, se debe diseñar un algoritmo backtracking o Branch&Bound que solucione el problema.

En nuestro caso, nos hemos decantado por el algoritmo de backtracking por su sencillez especialmente y porque este problema por su naturaleza no es de optimización y aplicar la técnica de Branch and Bound no sería una tarea sencilla puesto que habría que plantear el problema como uno de optimización en el que definir una o varias cotas con las que ramificar el espacio de estados y podar aquellos nodos que no son viables para la exploración. Una posible forma de aplicar Branch and Bound sería intentado colocar en cada movimiento, el caballo en aquellas casillas en las que el número de casillas aisladas sea mínimo, es decir, intentando no dejar casillas sin ocupar a las que posteriormente el caballo no pueda acceder porque todas a su alrededor están ocupadas. Sin embargo, diseñar esa estimación quizás no sea viable, puesto que en cada nodo que se genere habría que llamar al procedimiento y recorrer en el peor de los casos toda la matriz para comprobar si dejamos casillas aisladas o no, lo que supone que generar un nodo nos costaría en el peor de los casos O(n^2), es decir podríamos ahorrar nodos de exploración pero no los suficientes como para compensar el coste que nos cuesta generarlos, haciendo que aplicar Branch and Bound con esa estimación sea más ineficiente que aplicar la técnica de BackTracking.

# 2. Diseño de la solución

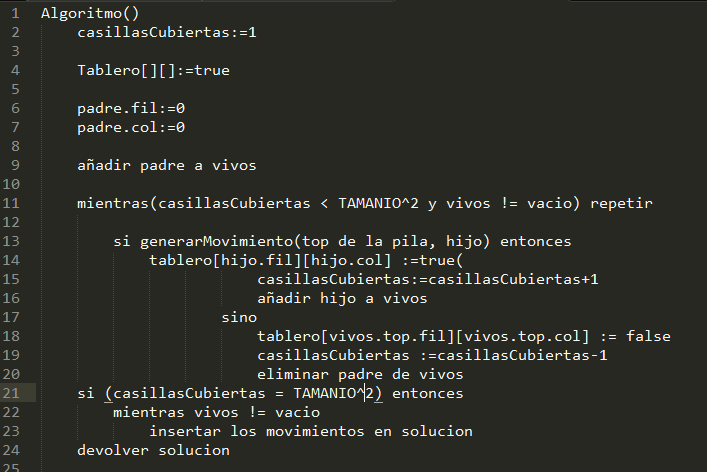


Para solventar este problema hemos optado por definir una matriz la cual representará el tablero de ajedrez, donde sus casillas pueden tener dos estados : visitado o no visitado. Hemos creado también un vector “movimientos” que almacena los ocho movimientos posibles que puede realizar el caballo en el tablero.

La solución la representaremos mediante un vector de tipo *Coordenada* que almacenará las coordenadas por las que va pasando el caballo desde la posición en la que parte hasta completar el tablero.

Para solucionar el problema aplicamos la técnica de Backtracking, partiendo desde una casilla inicial que estará incluida en el nodo inicial del árbol de búsqueda (el árbol de búsqueda no está implícito en el problema, y representa el conjunto de estados posibles del tablero generados a partir de la aplicación de alguno de los movimientos que puede realizar el caballo.) desde este punto vamos generando nodos si podemos, o retrocediendo a un nodo previo si no se pueden aplicar más movimientos al nodo que se está evaluando, intentando así hacer una especie de búsqueda en profundidad y hallar si es posible la secuencia de coordenadas que hay que tomar del tablero para obtener una solución.

# Explicación algoritmo y pseudocódigo.



**Componentes del algoritmo:**

**CasillasCubiertas**: Variable que cuenta el número de casillas del tablero visitadas por el caballo.

**Vivos**: Hemos usado una estrategia LIFO y por ese motivo la estructura de datos **vivos** es una pila, que almacena el conjunto de nodos del árbol de búsqueda por los que el caballo ha pasado, de tal forma que si el algoritmo encuentra una solución, esta estructura contiene todos los estados del tablero por los que el caballo ha tenido que pasar para llegar a la misma.

**Padre**: Nodo inicial del árbol de búsqueda, su valor se corresponde con la casilla del tablero desde la que parte el caballo, en el pseudocódigo hemos puesto que se empieza desde la casilla (0,0).

**Hijo**: Nodo auxiliar que usaremos para guardar aquel nodo que estemos generando a partir de otro.

**Tablero**: Matriz que representa el tablero sobre el que el caballo se desplaza, si una casilla ha sido visitada, su valor es “true” y en caso contrario “false”.

**Solución**: vector en el que almacenamos las coordenadas del tablero por las que ha ido pasado el caballo partiendo desde el nodo inicial hasta completar todas las casillas del tablero (donde todas tiene un valor “true”).

**TAMANIO**: Tamaño de las filas/columnas del tablero.

**GenerarMovimiento(padre,hijo)**: Función a la que dándole un nodo (padre) genera si es posible otro nodo (hijo), como resultado de aplicar el siguiente movimiento aplicable al padre.

**Explicación del algoritmo:**

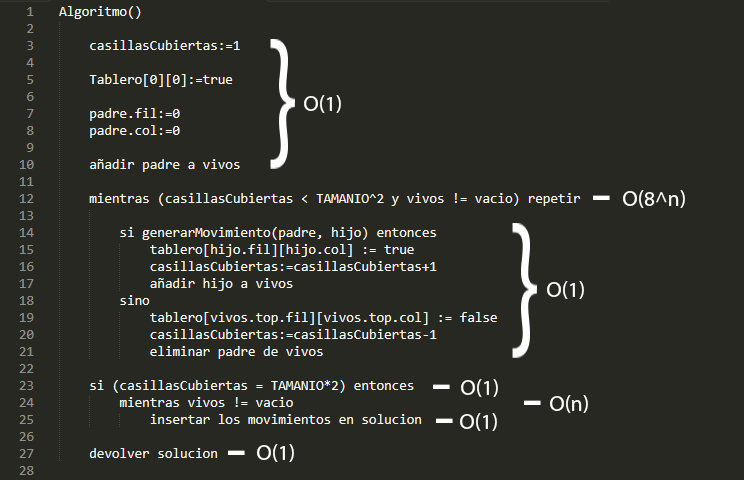
En primer lugar se agrega a la pila (**vivos**) el nodo inicial (identificado como **padre**) acto seguido el algoritmo realiza las siguientes acciones de forma iterativa, hasta obtener una solución (todas las casillas del tablero están a true) o no encontrar solución posible (la pila se vacía), llamaremos **nodo top** al nodo que se encuentra en la cabeza de la pila, es decir al final de la misma:

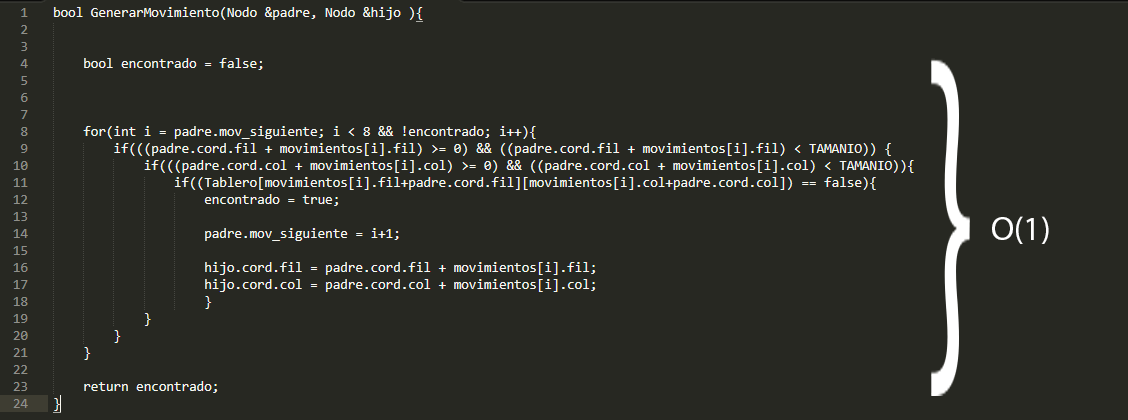
1. Si es posible aplicar el movimiento sobre el nodo top de la pila,se genera un hijo (almacenado en la variable **hijo**) con las coordenadas de la nueva posición del caballo y el siguiente movimiento pendiente de verificar si es aplicable o no. Además:
   1. La casilla del tablero con las coordenadas del nodo hijo generado (que se corresponde con la casilla hacia la que marcha el caballo) se marca como visitada (true).
   2. Se incrementa en uno el número de casillas visitadas por el caballo en el tablero.
   3. Se añade el nodo hijo generado a la pila, pasando a ser el nodo top.
2. si sobre el nodo top de la pila no se puede aplicar ningún movimiento del caballo entonces:
   1. La casilla del tablero con las coordenadas del nodo top se marca como no visitada (valor false).
   2. Se decrementa en uno el número de casillas visitadas por el caballo en el tablero.
   3. Se elimina el nodo top de la pila, puesto que a ese nodo no le quedan movimientos que aplicar, por tanto el nuevo nodo top pasa a ser el padre (si lo hay) del nodo eliminado (es decir el caballo **retrocede** al movimiento aplicado con anterioridad, si existe).

Una vez finalizado el procedimiento anterior, comprobamos si hemos encontrado una solución, si es el caso, tenemos que la pila está completa con ***n*** nodos (siendo n el número de casillas del tablero), siendo estos los estados del tablero por los que ha pasado el caballo hasta completarlo. Por tanto almacenamos en un vector **solucion,** una por una las coordenadas registradas en cada nodo de la pila.

.

# Eficiencia teórica



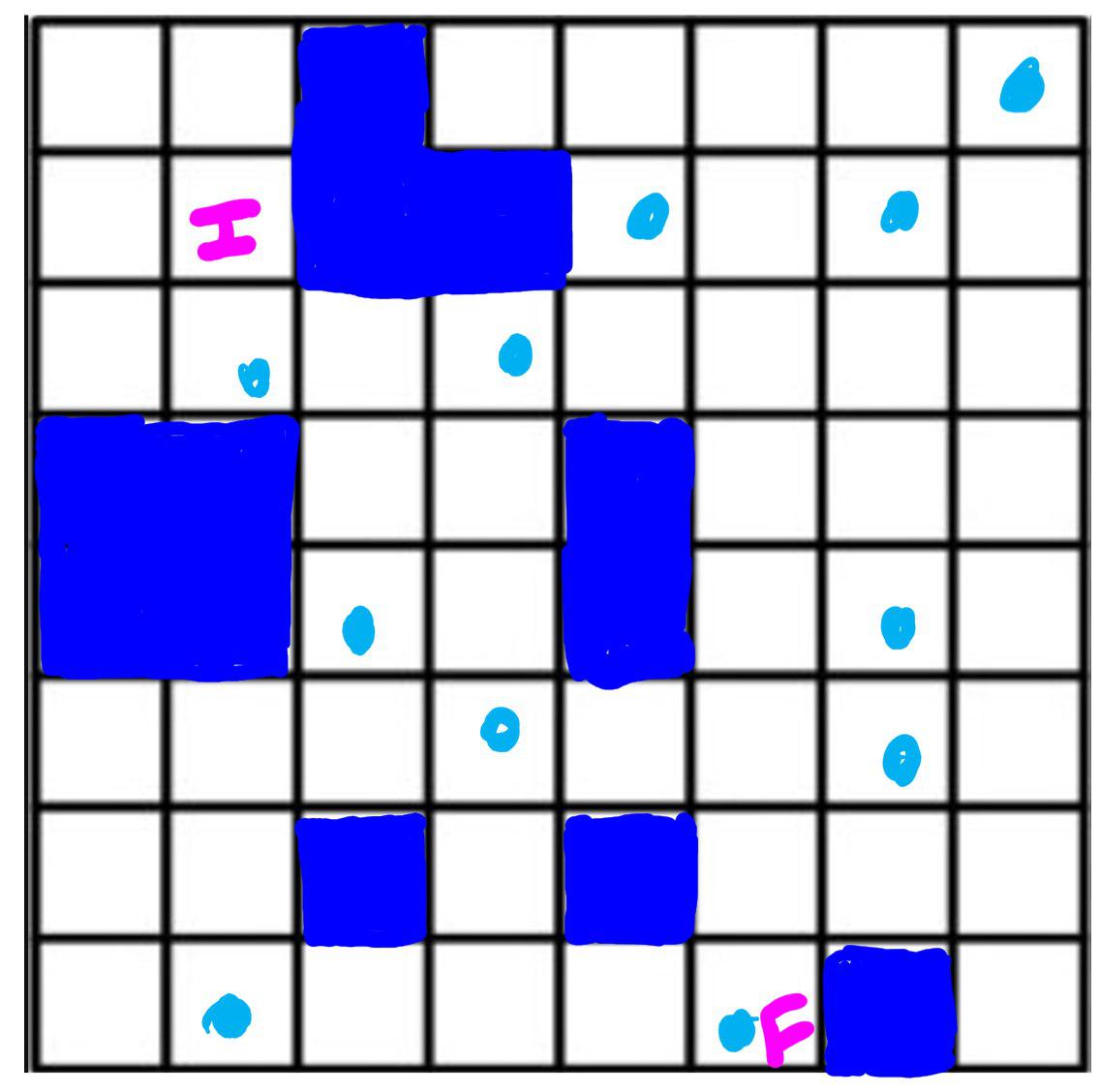
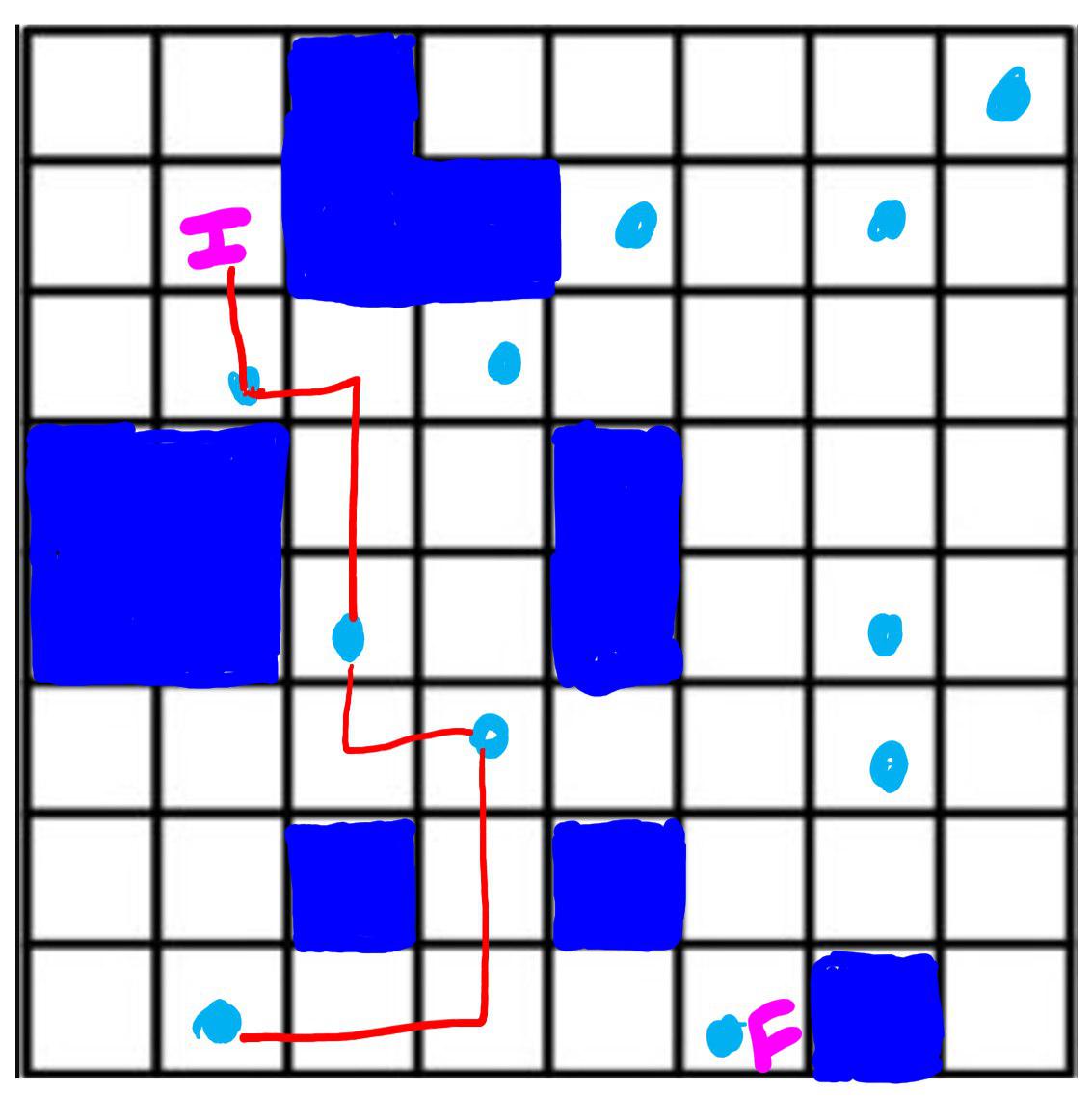


En este problema para calcular la eficiencia debemos fijarnos en la cantidad de nodos que tenemos que generar y analizar, puesto que el caballo puede realizar 8 movimientos posibles y las soluciones tienen un tamaño de ***n*** casillas, siendo ***n*** el tamaño del tablero, es decir el árbol de búsqueda tiene una profundidad de ***n***, en el peor de los casos, tendremos que analizar **8^n** nodos, tal y como hemos reflejado en las anteriores capturas, por tanto aplicando la regla de la suma obtenemos que la eficiencia sería *MAX(O(1),O(8^n),O(n))* que resulta en una complejidad en el peor de los casos de ***O(8^n)***

# 5. Caso real

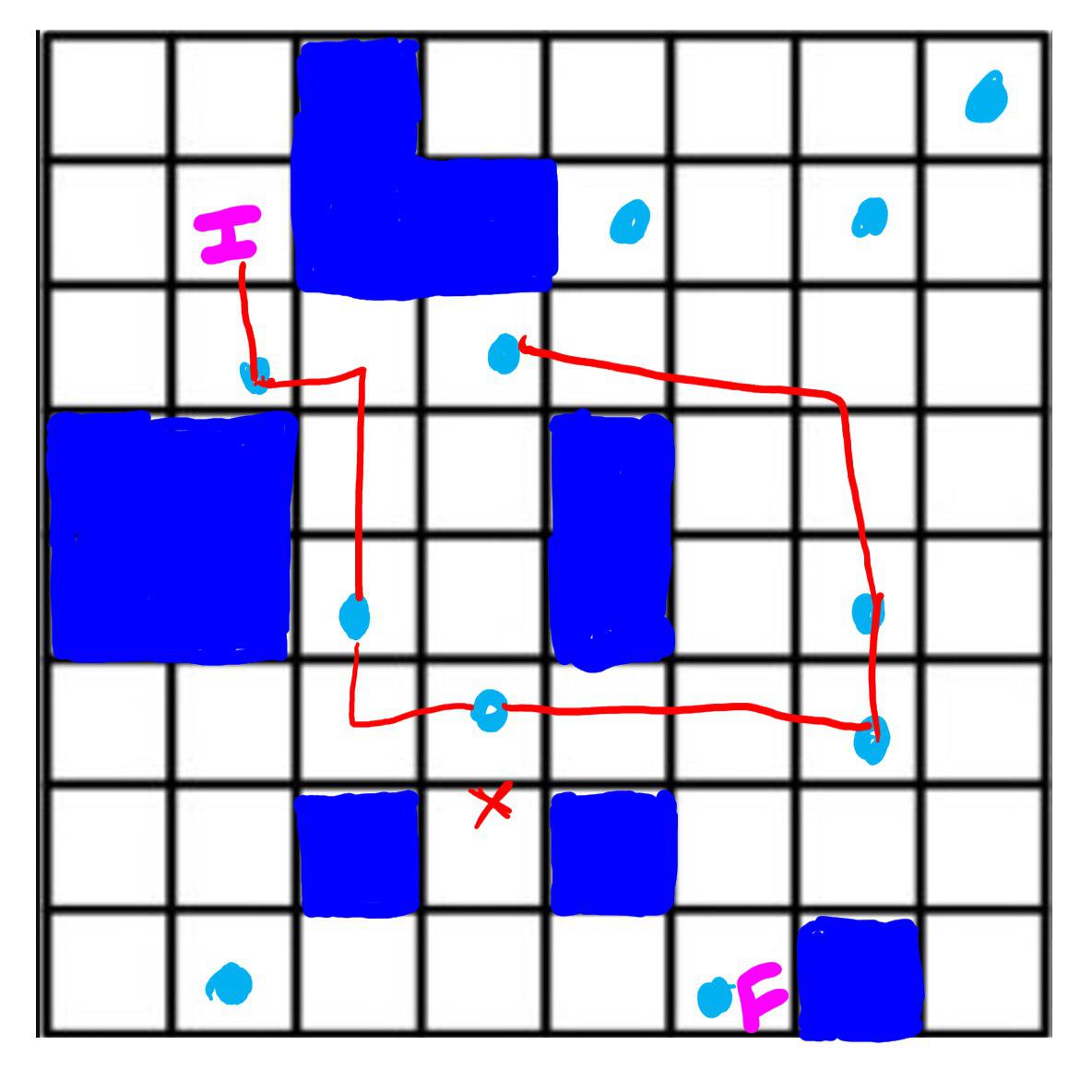
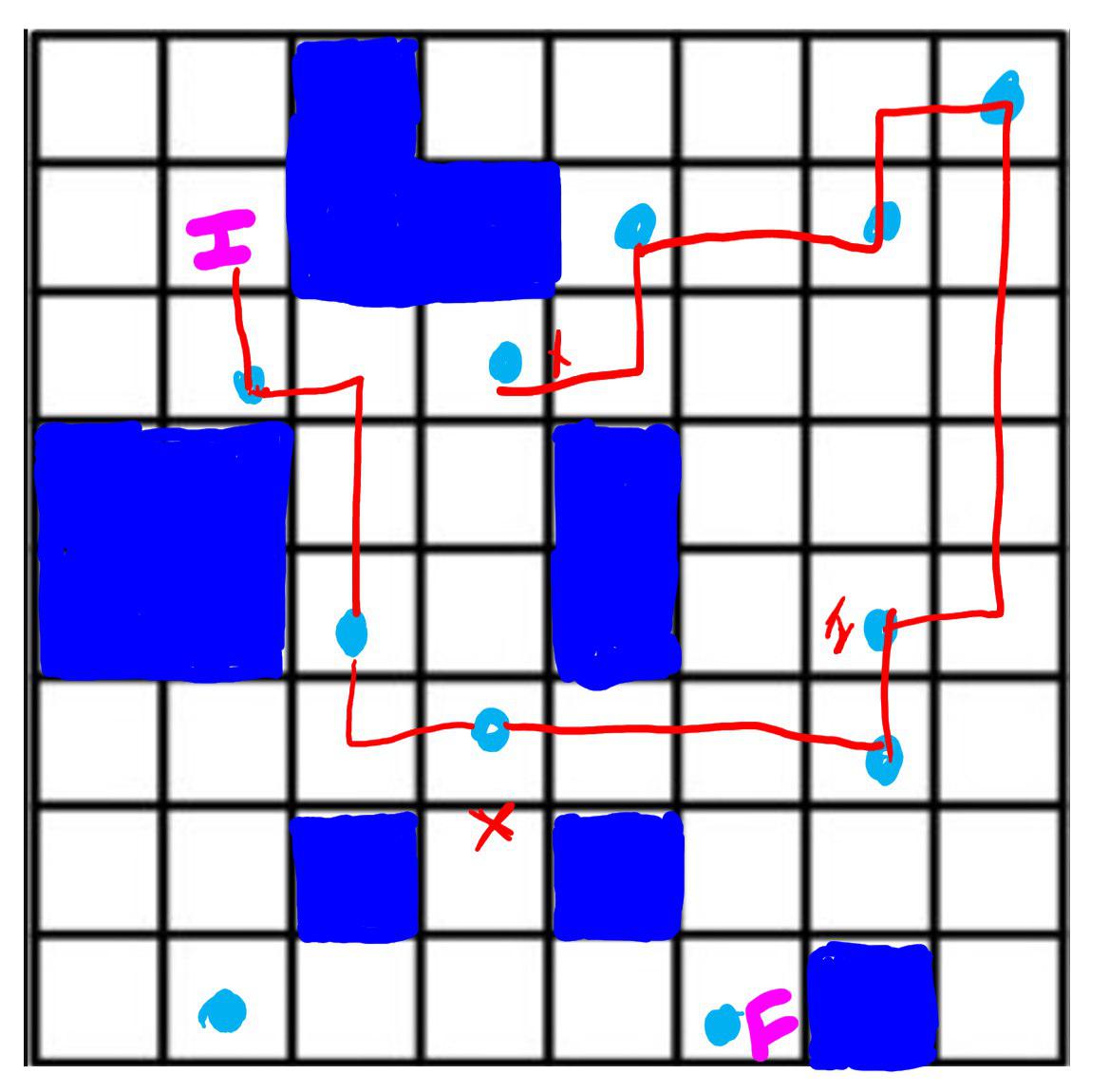
Como caso real hemos diseñado un nuevo juego que va por niveles y para pasar de nivel hay que recoger una serie de ítems que hay repartidos por el tablero pero no se puede pasar dos veces por una casilla. De esta forma tiene que hacer el camino del inicio al fin sin repetir casilla. Disponemos de un bichito que es el que se mueve por las casillas y va recogiendo migas de pan. Para subir de nivel debe recoger todas las migas de pan recordando el camino desde su inicio hasta el fin sin repetir ninguna casilla, pero puede hacer intentos para localizar la mejor de las soluciones y si no lo encuentra volver atrás a la parte donde empezó el camino sin salida para calcular otro camino que le lleve a la miga final que le hace subir de nivel.

Vamos a verlo con el ejemplo:

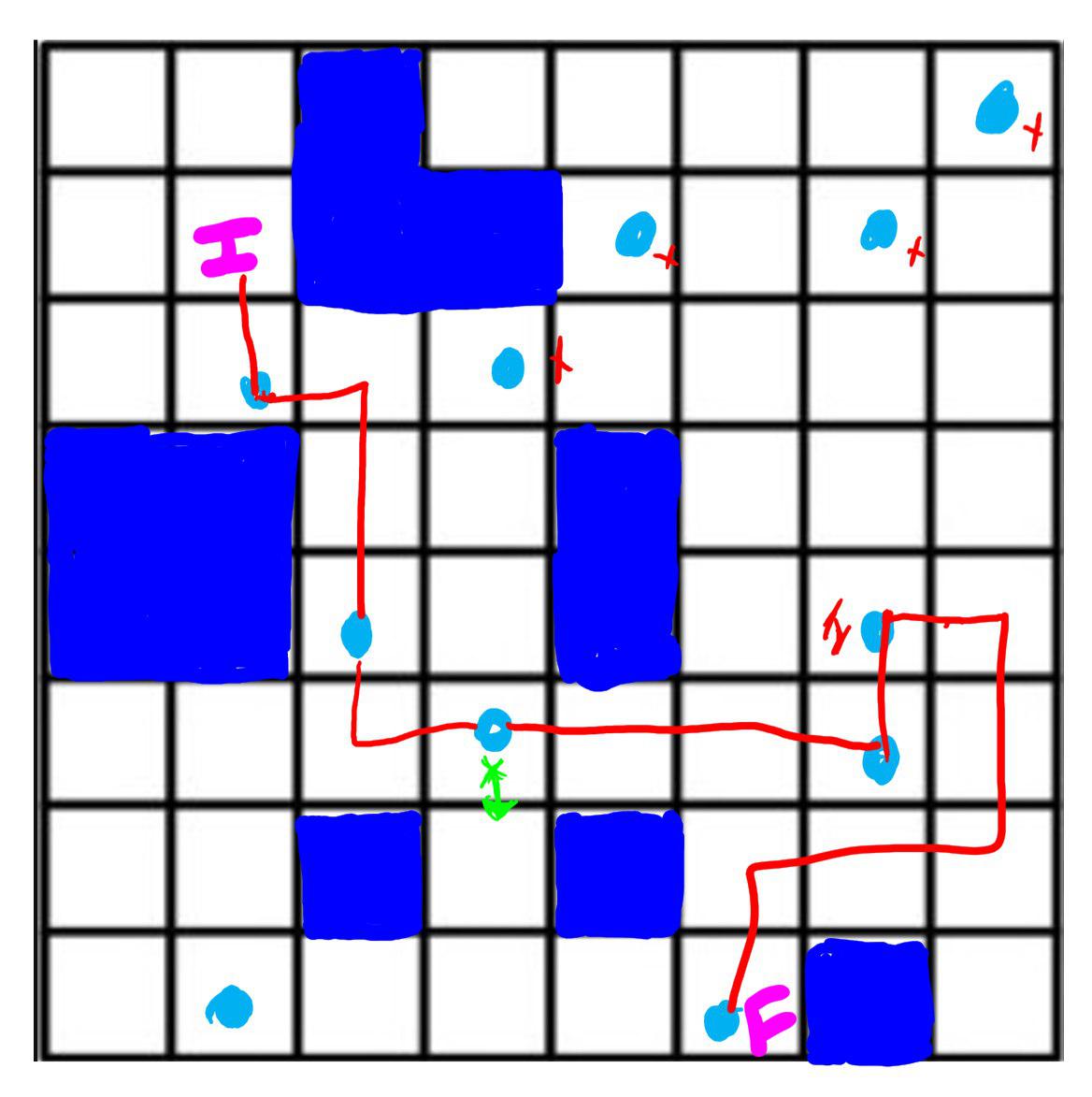
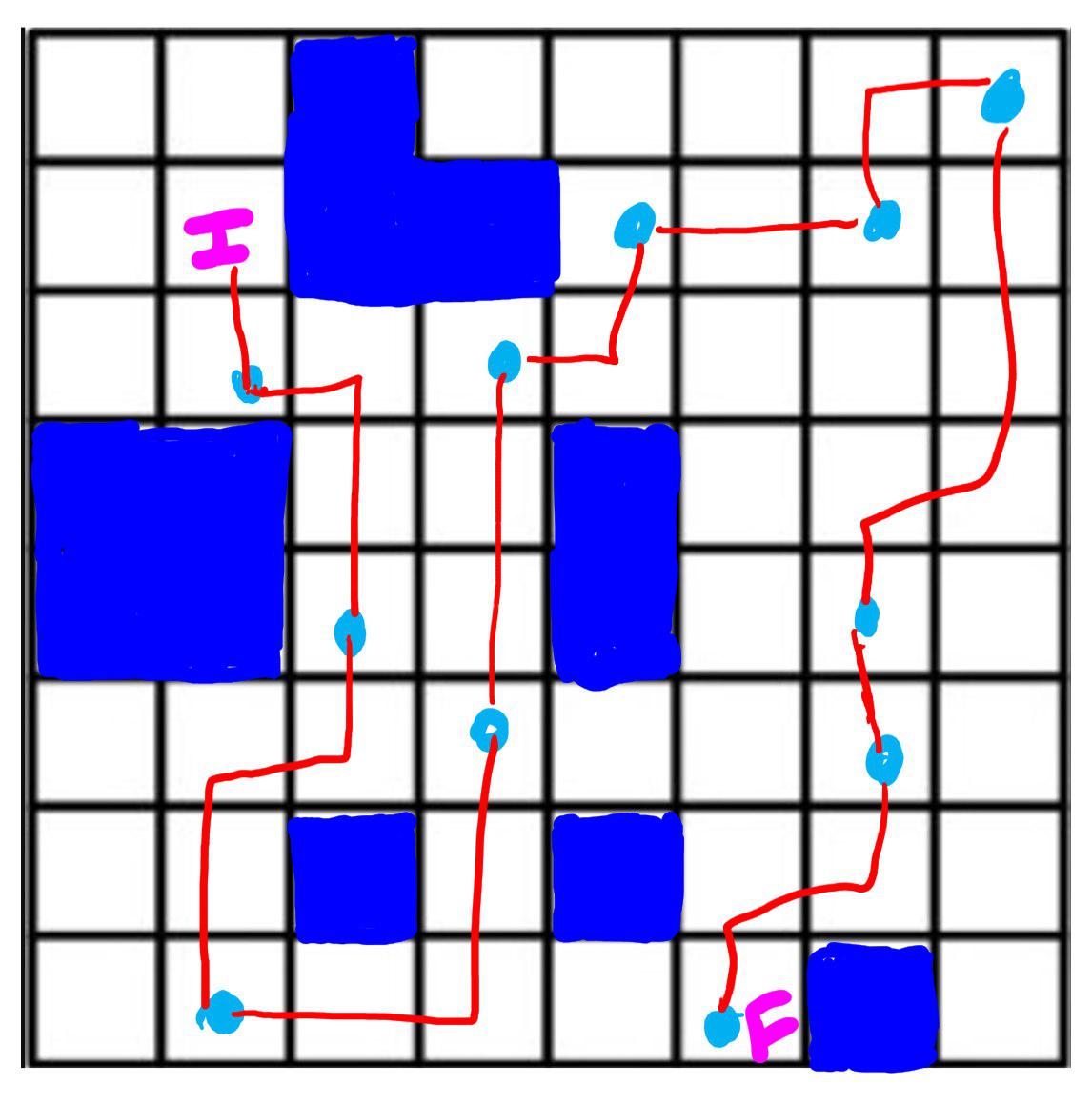
 

Como vemos la imagen inicial marca con la **I** la posición donde inicia nuestro bichito y con **F** la posición de la última miga que debe recoger para subir de nivel los círculos son el resto de migas que tiene que recoger sin repetir y las casillas sombreadas son obstáculos que no se pueden atravesar.

En la segunda imagen vemos como recorre un camino y ve que no puede seguir por ese camino para subir de nivel por lo tanto vuelve atrás a la última miga que recogió para explorar otro camino.

Una vez ha vuelto atrás sigue explorando otro camino recogiendo las migas que va encontrando, como se topa con que no puede seguir por ese camino para llegar a la solución realiza el mismo procedimiento que antes y vuelve atrás a la última miga que exploró para intentar llegar al final y subir de nivel

Como vemos en las imágenes, anteriores sigue realizando la misma operación de explorar y volver atrás cuando aún quedan caminos por explorar desde la última miga que se visitó antes de llegar a un camino sin salida.

En la imagen de la derecha podemos observar que tras explorar todos los caminos posibles desde la miga marcada con I no llegamos al final para subir de nivel, así que vuelve atrás a la última miga que exploró y aún dispone de caminos que se pueden explorar y desde ésta encuentra una solución para llegar al final y subir de nivel.

# 6.Instrucciones para compilar y ejecutar el programa

Para compilar hay que ejecutar la siguiente instrucción en la terminal:

*g++ -o caballo caballo.cpp*

Para ejecutar el programa basta con usar *./caballo.* Por defecto se ejecutará el algoritmo con un tablero de tamaño 8x8 y el caballo estará situado en la coordenada (0,0) del tablero. Para cambiar esto hay que modificar las constantes TAMANIO (que representa el tamaño de las filas/columnas del tablero) cordFilCab (es el número de fila sobre la que parte el caballo) y cordColCab (es el número de columna en el tablero sobre la que parte el caballo)

Estos valores deben ser valores enteros positivos y además cordColCab y cordFilcab deben de estar entre 0 y TAMANIO-1.

**Ejemplo de ejecución:**

