Entrega final- Proyecto Análisis de Algoritmos

David Enrique Palacios García[1] Karen Sofia Coral Godoy[1]

[1]Departamento de Ingeniería de Sistemas, Pontificia Universidad Javeriana
Bogotá, Colombia
{david_palacios, corallg_ksofia}@javeriana.edu.co

4 de octubre de 2022

Resumen

En este documento se presenta la formalización del juego en automático de *FlowFree* correspondiente a la entrega final del proyecto del curso Análisis de Algoritmos.

Palabras clave: matriz, interfaz, formalización, caminos, puntos.

Índice

1. Introducción

FlowFree es un juego que consiste en una cuadrícula de cuadrados (matriz) con pares de puntos de colores que ocupan algunos de los cuadrados. El objetivo es conectar puntos del mismo color mediante caminos que no pueden cruzarse entre ellos. La dificultad está determinada principalmente por las dimensiones de la matriz, que va desde $5x5$ hasta $9x9$. En este documento se presenta la creación de un algoritmo que permita resolver automáticamente cada tablero, con el objetivo de mostrar: la formalización del problema (sección 3), idea de solución (sección 4), algoritmo planteado (sección 4.1.1) y análisis de la solución (sección 5) .

2. ¿Cómo jugar?

En el juego *FlowFree* tenemos una matriz que puede ser de distintos tamaños (para esta interfaz desde 5×5 hasta 9×9). En dichas casillas hay puntos de colores distribuidos en todo el tablero y se deben unir las parejas del mismo color. La cantidad de colores está determinada por el resultado de restarle a la dimensión de la matriz una unidad. Las uniones no se pueden cruzar y es necesario rellenar todas las casillas, es decir, no pueden quedar casillas vacías en el tablero.

2.1. Condiciones para ganar

Básicamente para que exista una victoria se debe cumplir con el objetivo del juego previamente descrito, unir todas las parejas de colores mediante caminos que no se cruzan, sin dejar casillas vacias en el tablero.

De manera específica, la victoria es alcanzada cuando se cumplen las siguientes condiciones:

1. Todos los caminos que unen las parejas de colores están completos, es decir, el número de caminos correctos es igual al número de colores del tablero
2. El porcentaje de llenado del tablero es igual al 100 %

2.2. Restricciones del juego

1. Únicamente se permiten realizar y deshacer movimientos horizontales o verticales (NO diagonales)
2. El número de colores es igual a la dimensión del tablero menos uno
3. Sólo están disponibles matrices de dimensiones desde 5×5 hasta 9×9 y se utilizaron los tableros originales del juego (para cada dimensión se utilizan 2 de estos, con el objetivo de asegurar que tienen solución).

3. Formalización del problema

Teniendo claro cómo se juega y el objetivo general del juego, es importante definirlo en términos de algoritmia y lógica. Cuando se juega un tablero de *FlowFree*, la intuición humana nos indica que la mejor manera de iniciar el llenado de un tablero es por medio de los bordes. ¿Por qué sucede esto? Es sencillo, porque estos nos impiden generar más caminos que por dentro de la matriz. Al existir campos no accesibles, tenemos menos espacio para movernos, por lo que es más sencillo llenar los bordes antes que el interior del tablero.

No obstante, una máquina no es capaz de asimilar esto de una manera tan intuitiva. Para esta, la solución exacta es probar todos los caminos posibles en el tablero, y si la sucesión de estos cumple la condición de victoria descrita previamente, entonces dicha sucesión de caminos es la que permite la victoria en el tablero.

Un humano puede iniciar un tablero de manera similar, llenando aleatoriamente todos los caminos posibles en el tablero, y, eventualmente, llegar a una solución. Esto es la búsqueda por fuerza bruta.

Se sabe que cada tablero diseñado tiene una única solución, por lo que, si una persona sigue esta estrategia, llegará un momento donde se canse y se aburra del juego, esta es la ventaja de las máquinas, no se cansan ni se aburren, se puede probar en repetida ocasiones posibles soluciones hasta que alguna cumpla la condición de victoria, siendo la clave para analizar el problema.

3.1. Definición del problema del llenado automático de "Flow Free"

Así, el problema del llenado automático se puede definir como:

3.1.1. Entradas del problema

1. Un tablero definido como $T \in N^{(r \times c)}$ donde $t_{ij} \in [0, 9]$ y t_{ij} representa cada color del tablero.

3.1.2. Salidas del problema

1. Una sucesión de caminos defnido como $C = \langle C_1, C_2, ..., C_n \rangle$ donde C_i es cada *Camino* que cumple con la condición de victoria.
2. Cada *Camino* se compone de 5 ítems
   a) $c \in [1, 9]$ refiriéndose al *color del camino*
   b) $i \in [0, r]$ *donde* r *es la cantidad de filas del tablero: fila inicial del camino*
   c) $j \in [0, c]$ *donde* c *es la cantidad columnas del tablero: columna inicial del camino*
   d) $k \in [0, r]$ *donde* r *es la cantidad de filas del tablero: fila final del camino*
   e) $l \in [0, c]$ *donde* c *es la cantidad columnas del tablero: columna final del camino*

4. Idea de solución

4.1. Fuerza bruta

Si se quiere generar la única solución existente en el tablero tenemos entonces un problema de clase NP y tipo *Decisión*. Se pretende decidir qué caminos elegir para completar un tablero, por lo que se justifica su tipo. Para su clase, es necesario definir la manera de solucionar este problema.

Como se conoce, cada tablero tiene una única solución, por lo que se debe realizar la combinación de todos los caminos posibles dado un tablero y sus dimensiones. Para evaluar cada una de las sucesiones de caminos, se debe crear mapas de prueba, donde se llenen según las instrucciones de cada camino. Si este mapa de prueba cumple con la condición de victoria, entonces esa sucesión de caminos es la solución exacta del problema. Lo anterior justifica la clase de problema a atacar, y explica de manera general la idea de solución.

4.1.1. Algoritmo solución

Teniendo en cuenta las entradas y salidas del problema, de manera general se plantearon las siguientes 4 funciones que buscan resolver el problema mediante fuerza bruta.

1. CrearCaminos: Crea todos los posibles caminos (estados) de un tablero, a partir de número de filas, columnas, y cantidad de colores disponibles.

Algoritmo 1 Crear caminos posibles en el tablero

```
1: procedure CrearCaminos(T)
2:     caminos ← Camino[]
3:     for i ← 0 to T.rows do
4:         for j ← 0 to T.cols do
5:             for k ← 0 to T.rows do
6:                 for l ← 0 to T.cols do
7:                     for color ← 0 to T.colors do
8:                         camino ← Camino(color,i,j,k,l)
9:                         caminos.add(camino)
10:                    end for
11:                end for
12:            end for
13:        end for
14:    end for
15:    return caminos
16: end procedure
```

2. EncontrarSolución: Encuentra la solución del tablero creando las combinaciones de todos los posibles estados (CrearCaminos) del tablero.

Algoritmo 2 Encontrar solución de tablero

```
1: procedure EncontrarSolución(T)
2:     caminos ← CrearCaminos(T)
3:     for r ← 1 to |caminos| + 1 do
4:         combinations ← Combinations(caminos,)
5:         for c ← 0 to |combinations| do
6:             comb ← combinations[c]
7:             if ValidarCamino(T, comb) then
8:                 return comb
9:             end if
10:        end for
11:    end for
12: end procedure
```

3. ValidarCamino: Valida si una combinatoria de caminos es aceptada como solución. Esto es, los caminos no son diagonales, no se solapan y están conectados.

Algoritmo 3 Validar combinatoria de camino

```
1: procedure ValidarCamino(T, caminos)
2:     tableroPrueba ← T.copy()
3:     for c ← 0 to |caminos| do
4:         camino ← caminos[c]
5:         color ← camino.color
6:         i ← camino.i
7:         j ← camino.j
8:         k ← camino.k
9:         l ← camino.l
10:        if i ≠ k ∧ j ≠ l then
11:            return False
12:        end if
13:        if tableroPrueba[i][j] ≠ 0 then
14:            return False
15:        end if
16:        if tableroPrueba[k][l] ≠ 0 then
17:            return False
18:        end if
19:        tableroPrueba[i][j] ← color
20:        tableroPrueba[k][l] ← color
21:    end for
22:    return validarTableroPrueba(tableroPrueba)
23: end procedure
```

4. ValidarTableroPrueba: Valida si un tablero de prueba cumple con la condición de que se encuentre 100 % lleno.

Algoritmo 4 Validar tablero de prueba

```
1: procedure ValidarTableroPrueba(tableroPrueba)
2:     for i ← 0 to tableroPrueba.rows do
3:         for j ← 0 to tableroPrueba.cols do
4:             if tableroPrueba[i][j] = 0 then
5:                 return False
6:             end if
7:         end for
8:     end for
9:     return True
10: end procedure
```

5. Análisis de solución

Como se aclaró anteriormente, se tiene un problema de clase NP y tipo *decision*, donde se encontró la solución mediante fuerza bruta. A la hora de probar los algoritmos, después de mucho tiempo de ejecución, nunca se llegó a completar el proceso a finalidad. Esto sucede por la naturaleza de la estrategia de fuerza bruta, pues es una solución combinatoria.

5.1. Análisis de complejidad

La complejidad algortítmica de este algoritmo es $O(2^n)$ donde $n = T.rows^2 \times T.cols^2 \times T.colors$, lo que lo hace, aunque preciso, sumamente tardío en ejecución. El hecho de tener que combinar todos los posibles estados del tablero, y verificar por cada uno de ellos si cumple con la condición de victoria, hace que el programa tenga que evaluar millones de operaciones para llegar a la solución.

Además, teniendo en cuenta que únicamente existe una solución exacta para el problema, hace que realmente sea dificíl para un ser humano esperar a que el programe arroje la solución esperada. Cabe resaltar, que para los tableros más sencillos (de 5×5) ni siquiera fue capaz de terminar en más 1 hora de ejecución, y no es difícil imaginar lo que tardaría con alguno más complejo.

6. Diagrama de clases

El juego *FloowFree* está compuesto por 4 clases definidas en el siguiente diagrama de clases (Ver figura 1) que trazan claramente la estructura del sistema que se intenta modelar mediante atributos, operaciones y relaciones entre clases.

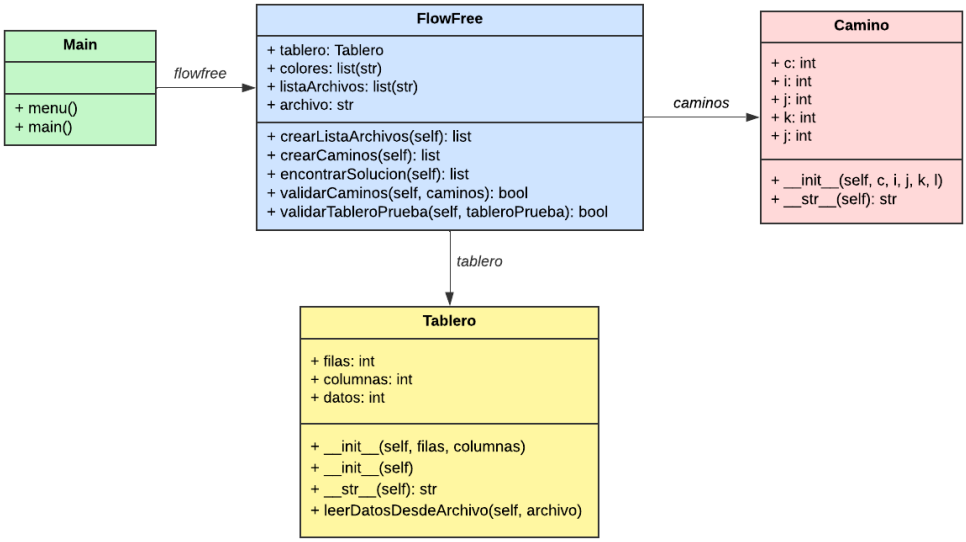

Figura 1: Diagrama de clases

6.1. Clase: Main

Es la clase que ejecuta el main del juego y llama a los métodos de las demás clases. Esta clase recibe la entrada del usuario que especifica el tablero que deseará solucionar y llama al método de la clase *FlowFree* que intenta encontrar la solución al tablero escogido.

6.2. Clase: Tablero

La clase *FlowFree* compone la lógica del juego, contiene el tablero de juego y sus atributos determinan e informan características importantes del juego como: colores disponibles. En cuanto a sus métodos, corresponden a la creación y validación de posibles caminos que conlleven la existencia de una victoria.

6.3. Clase: Camino

La clase *Camino* es muy sencilla, está compuesta por atributos que describen un camino como: color y coordenadas de inicio y fin, así mismo, cuenta con operaciones sencillas de inicialización y asignación del color en un camino.

6.4. Clase: Tablero

La clase *Tablero* define e inicializa un tablero de juego a partir de la lectura de archivos con las especificaciones del tablero elegido por el usuario.

Referencias

[1] Wikipedia, *Flow Free*, https://en.wikipedia.org/wiki/Flow_Free