Introducción a la Inteligencia Artificial
Proyecto sobre agentes de búsqueda
Prof. Carlos B. Ogando M.

PROYECTO FLOW FREE

HONESTIDAD ACADÉMICA

Como de costumbre, se aplica el código de honor estándar y la política de probidad académica. Las presentaciones isomórficas a (1) las que existen en cualquier lugar en línea, (2) las enviadas por sus compañeros de clase, o (3) las enviadas por los estudiantes en semestres anteriores serán consideradas plagio.

INSTRUCCIONES

En esta tarea, creará un agente para resolver el juego de **Flow Free**. Implementarás y compararás varios algoritmos de búsqueda y recopilarás algunas estadísticas relacionadas con su desempeño. Lea atentamente todas las secciones de las instrucciones:

I. Introducción

II. Revisión de algoritmos

III. Qué necesita enviar

IV. Lo que produce su programa

V. Información importante

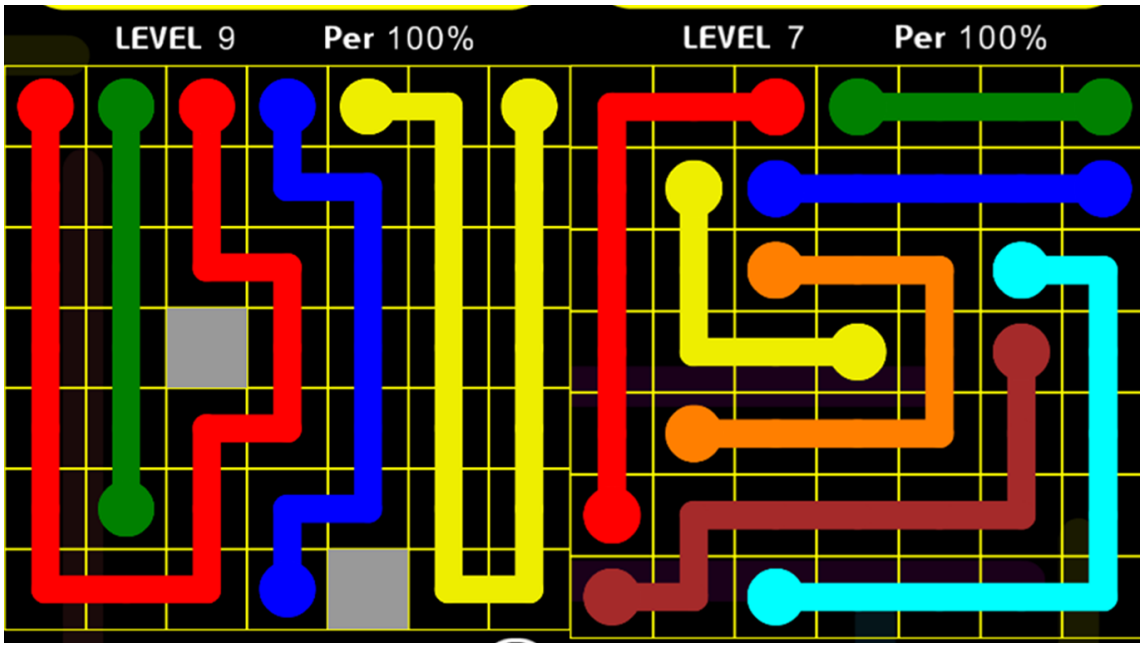
VI. Antes de terminar

I. Introducción

Flow Free presenta rompecabezas tipo numberlink, el clásico juego de hallar la ruta que conecte dos números. Cada rompecabezas tiene una cuadrícula de cuadrados con pares de puntos de colores que ocupan algunos de los cuadrados. El objetivo es conectar puntos del mismo color dibujando "tubos" entre ellos de modo que toda la cuadrícula esté ocupada por tubos. Sin embargo, los tubos no pueden cruzarse. La dificultad está determinada principalmente por el tamaño de la cuadrícula, que varía desde cuadrados de 5x5 (3 colores) hasta cuadrados de 15x15 (hasta 16 colores). Muchas cuadrículas están "abiertas" y algunas contienen "paredes" que se deben sortear. Cada vez que se completa un nivel, aparecerá una marca de verificación en el ícono de selección de nivel para indicar que el rompecabezas está resuelto, mientras que una estrella indica un juego "perfecto", en el que el jugador terminó el rompecabezas con la menor cantidad de movimientos necesarios. La aplicación también contiene paquetes pagos adicionales, así como un modo de contrarreloj.

En el siguiente enlace podemos ver un ejemplo de este juego:

https://www.xpgameplus.com/games/flowfree/index.html



II. Revisión de algoritmos

Recuerde de las conferencias que las búsquedas comienzan visitando el nodo raíz del árbol de búsqueda, dado por el estado inicial. Entre otros detalles contables, suceden tres cosas importantes en secuencia para visitar un nodo:

- Primero, **eliminamos** un nodo del conjunto de fronteras.
- En segundo lugar, **comparamos** el estado con el estado objetivo para determinar si se ha encontrado una solución.
- Finalmente, si el resultado de la verificación es negativo, **expandimos** el nodo. Para expandir un nodo dado, generamos nodos sucesores adyacentes al nodo actual y los agregamos al conjunto de fronteras. Tenga en cuenta que, si estos nodos sucesores ya están en la frontera, o ya han sido visitados, no deben volver a agregarse a la frontera.

Esto describe el ciclo de vida de una visita y es el orden básico de operaciones para los agentes de búsqueda en esta asignación: (1) eliminar, (2) verificar y (3) expandir. En esta tarea, implementaremos algoritmos como se describe aquí. Consulte las notas de la clase para obtener más detalles y revise el pseudocódigo de la clase antes de comenzar la tarea.

III. Qué necesita enviar

El objetivo del proyecto es crear el juego de **`Flow Free`** en Python **`jugable por consola`** e implementarlo tres algoritmos de IA que encuentren la solución de forma automática a los rompecabezas.

3.1. Requerimientos del juego:

1. Los niveles deben poder leerse a partir de txt. El formato es el siguiente:

a. Se indicará con un punto “.” Un espacio vacío de la matriz que representa el tablero.
b. Se indicará con la letra “R” (mayúscula), los puntos rojos a conectar.
c. Se indicará con la letra “A” (mayúscula), los puntos azules a conectar.
d. Se indicará con la letra “Y” (mayúscula), los puntos amarillos a conectar.
e. Se indicará con la letra “V” (mayúscula), los puntos verdes a conectar.
f. Se indicará con la letra “M” (mayúscula), los puntos magentas a conectar.
g. Se indicará con la letra “C” (mayúscula), los puntos cyan a conectar.
h. Para otros colores adicionales indicar la letra usted.
i. Sea n la cantidad de filas del tablero y m la cantidad de columnas. El txt tendrá n filas, cada una representando una fila del tablero, y cada fila m caracteres representando los espacios y objetos del tablero.
j. Por ejemplo, el siguiente nivel (5x6) se vería en el txt de la siguiente forma:

A	V	.	.	.	V
.	R	.	.	.	.
.	.	A	M	Y	.
M	.	.	.		R
C	.	.	C	.	Y

2. El tamaño de los niveles debe poder ser variable.
3. Un paso a nivel del juego representa colocar un camino en el grid.

3.2. Requerimientos de los algoritmos:

1. Se deben implementar tres algoritmos de búsqueda: DFS, BFS y AST. (Ver pseudocódigo de la PPT de agentes de búsqueda en la plataforma virtual).
2. Cada algoritmo debe estar implementado en una función/clase distinta.
3. El algoritmo de AST debe emplear mínimo 3 heurísticas para su función de coste.
4. La función de coste de heurística debe estar debidamente normalizada y ponderada.
5. El ejecutarse cualquier de los algoritmos se debe generar (preferiblemente en un txt tipo output.txt) las siguientes métricas se rendimiento:
   a. Lista de movimientos para resolver el acertijo ([Up, Down, Left, Right…]) (ruta).
   b. Número de movimientos (costo de la ruta).
   c. Cantidad de nodos expandidos.
   d. Profundidad de la solución.
   e. Máxima profundidad de la búsqueda.
   f. Tiempo de ejecución
   g. Máxima memoria RAM consumida durante la ejecución.

3.3. Requerimientos del análisis:

1. Para el entendimiento y el análisis de cada uno de los algoritmos se deberán crear los siguientes niveles:
    a. 4 niveles 5x5 fáciles con 4 colores.
    b. 4 niveles 7x7 medianos con 5 colores.
    c. 4 niveles 15x15 difíciles con 9 colores.
    d. Nota adicional: procurar que los niveles sean lo suficientemente variados en dificultad y que no sean triviales resolverlos.
2. Realizarán un benchmark entre los 3 algoritmos y sus configuraciones respectivas, en donde compararán por cada configuración:
    a. Métricas:
        i. Número de movimientos (costo de la ruta).
        ii. Cantidad de nodos expandidos.
        iii. Profundidad de la solución.
        iv. Máxima profundidad de la búsqueda.
        v. Tiempo de ejecución.
        vi. Máxima memoria RAM consumida durante la ejecución.
    b. Variables para comparar:
        i. DFS
        ii. BFS
        iii. AST con pesos iguales en sus heurísticas.
        iv. AST con 1er configuración de pesos.
        v. AST con 2nda configuración de pesos.
        vi. AST con 1 heurísticas.
        vii. AST con 2 heurísticas.
        viii. AST con 3 heurísticas.
        ix. AST con 4 heurísticas.
        x. AST con 5 heurísticas.
    c. Ejemplo de cómo se puede ver:

NIVEL X	Número de movimientos	Cantidad de nodos expandidos	Profundidad de la solución	...	Tiempo de ejecución	Máxima cantidad de memoria empelada
DFS						
BFS						
...						
AST con 4 heurísticas.						
AST con 5 heurísticas.						

Nota: No limitarse a este esquema. Pueden hacer una tabla nivel vs algoritmo bajo una sola métrica ordenado por dificultad.

Número de movimientos	Nivel 1	Nivel 2	Nivel 3	...	Nivel X	Nivel Y
DFS						
BFS						
...						
AST con 4 heurísticas.						
AST con 5 heurísticas.						

Pueden acompañar el análisis de los algoritmos con gráficas o cualquier otro elemento que consideren importante.

IV. Lo que produce su programa

Cuando se ejecute, su programa creará / escribirá en un archivo llamado `output.txt`, que contiene las siguientes estadísticas:

- `path_to_goal:` la secuencia de movimientos realizados para alcanzar la meta (ver formato en capítulo III)
- `cost_of_path:` el número de movimientos realizados para alcanzar la meta.
- `nodes_expanded:` el número de nodos que se han expandido
- `search_depth:` la profundidad dentro del árbol de búsqueda cuando se encuentra el nodo objetivo
- `max_search_depth:` la profundidad máxima del árbol de búsqueda durante la vida útil del algoritmo
- `running_time:` el tiempo de ejecución total de la instancia de búsqueda, expresado en segundos
- `max_ram_usage:` el uso máximo de RAM durante la vida útil del proceso.

4.1. Formato de path_to_goal

El formato de path_to_goal debe indicar en una matriz el rompecabezas resuelto.

Ejemplo de output.txt

Suponga que el programa se ejecuta para la búsqueda en amplitud con el nivel X.

El archivo de salida podría verse como las siguientes líneas:

```
path_to_goal: [['R', 'R', 'R', 'R'], ['A', 'A', 'A', 'A'], ['V',
'V', 'V', 'V'], ['Y', 'Y', 'Y', 'Y']]
cost_of_path: 4
nodos_expandidos: 29
profundidad_de_búsqueda: 4
max_search_depth: 5
running_time: 0.00188088
max_ram_usage: 0.07812500
```

V. Información importante

Por favor lea la siguiente información cuidadosamente. En esta sección proporcionaremos muchas sugerencias e instrucciones explícitas. Antes de publicar una pregunta aclaratoria en el panel de discusión, asegúrese de que su pregunta no esté ya respondida en las siguientes secciones.

1. Implementación

Implementará los siguientes tres algoritmos base como se demostró en la lección. En particular:

- **BFS**. Utilice una **cola explícita**, como se muestra en la lección.
- **DFS.** Utilice una **pila explícita**, como se muestra en la lección.
- **Búsqueda A-Star.** Utilice una **cola de prioridad**, como se muestra en la lección.

2. Orden de visitas

En esta asignación, donde se debe hacer una elección arbitraria, siempre visitamos los nodos secundarios bajo el siguiente esquema:

1. Primer criterio de desempate es el carro con l**a letra con el menor valor ASCII**.
2. Segundo criterio de desempate se toma la dirección **en orden UDLR.**

. Sobre el segundo criterio, específicamente:

- **BFS.** Poner en cola en orden **UDLR**; la eliminación de la cola da como resultado el orden UDLR.
- **DFS.** Empuje sobre la pila en orden **inverso-UDLR**; apareciendo resultados en orden UDLR.
- **Búsqueda A-Star.** Dado que está utilizando una cola de prioridad, ¿qué sucede cuando hay claves duplicadas? ¿Qué debe hacer para asegurarse de que los nodos se recuperen de la cola de prioridad en el orden deseado?

3. Pruebas de calificación y estrés

Calificaremos su proyecto ejecutando casos de prueba adicionales en su código. Si implementa su código con diseños razonables de estructuras de datos, su código resolverá los casos de prueba en un tiempo razonable. Usaremos una amplia variedad de entradas para realizar pruebas de estrés en sus algoritmos para verificar que la implementación sea correcta. Por lo tanto, le recomendamos que pruebe su propio código de forma exhaustiva.

No se preocupe por comprobar si hay placas de entrada mal formadas, incluidas placas de dimensiones no cuadradas o placas sin solución.

No se le calificará según los valores absolutos de su tiempo de ejecución o las estadísticas de uso de RAM. Los valores de estas estadísticas pueden variar

ampliamente según la máquina. Sin embargo, le recomendamos que los aproveche al probar su código. Intente ejecutar por lotes sus algoritmos en varias entradas y trazar sus resultados en un gráfico para obtener más información sobre las características de complejidad de espacio y tiempo de su código. **El hecho de que un algoritmo proporcione la ruta correcta hacia la meta no significa que se haya implementado correctamente.**

4. Consejos para empezar

Después de programar el juego jugable por consola con todas las reglas de lugar, comience escribiendo una clase para representar las estructuras de datos chequeables para la frontera (Queue, Stack y Priority Queue). Al comparar su código con un pseudocódigo, es posible que se le ocurra otra clase para organizar con elegancia aspectos específicos de su algoritmo de búsqueda.

No será calificado por su diseño, por lo que tiene la libertad de elegir entre sus paradigmas de programación favoritos. Los estudiantes han completado con éxito este proyecto utilizando un enfoque totalmente orientado a objetos, y otros lo han hecho con un enfoque puramente funcional.

VI. Antes de terminar

- **Asegúrese** de que sus algoritmos generen la solución correcta para una instancia arbitraria de un problema solucionable de Car Parking Puzzle.
- **Asegúrese** de que su programa siempre finalice sin errores y en un período de tiempo razonable.