Introducción a la Inteligencia Artificial
Proyecto sobre algoritmos de Búsqueda
Prof. Carlos B. Ogando M.

PROYECTO PUZZLE-8

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 **Puzzle-8**. 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

Debe entregar la solución a este proyecto en un proyecto de Google Collab donde se demuestre el trabajo colaborativo de cada integrante del equipo.

Tenga en cuenta que hay un código esqueleto disponible para su uso en Google Collab. Esto es completamente opcional de usar y puede modificarlo tanto como desee. También puede completar la tarea sin consultar el código esqueleto en absoluto.

I. Introducción

Una instancia del juego N-puzzle consiste en un tablero que contiene N = m ^ 2 - 1 (m = 3, 4, 5, ...) fichas móviles distintas, más un espacio vacío. Los mosaicos son números del conjunto {1,..., m ^ 2 - 1}. Para cualquier tablero de este tipo, el espacio vacío se puede intercambiar legalmente con cualquier ficha horizontal o verticalmente adyacente a él. En esta tarea, representaremos el espacio en blanco con el número 0 y nos centraremos en el caso m = 3 (rompecabezas de 8).

Dado un estado inicial del tablero, el problema de la búsqueda combinatoria es encontrar una secuencia de movimientos que haga la transición de este estado al

estado objetivo; es decir, la configuración con todos los mosaicos dispuestos en orden ascendente ⟨0, 1,..., m ^ 2 - 1⟩. El espacio de búsqueda es el conjunto de todos los estados posibles alcanzables desde el estado inicial.

El espacio en blanco puede intercambiarse con un componente en una de las cuatro direcciones {"Arriba", "Abajo", "Izquierda", "Derecha"}, un movimiento a la vez. El costo de pasar de una configuración de la placa a otra es el mismo e igual a uno. Por lo tanto, el costo total de la ruta es igual al número de movimientos realizados desde el estado inicial al estado objetivo.

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

Su trabajo en esta tarea es desarrollar una solución en Google Collab que resuelve cualquier tablero de 8 rompecabezas cuando se le da una configuración inicial arbitraria. El programa se ejecutará y se leerán dos entradas:

```
<método> <tablero>
```

El argumento del método será uno de los siguientes. Debe implementar los tres

`bfs` (búsqueda primero en amplitud)

`dfs` (búsqueda en profundidad)

`ast` (búsqueda A-Star)

El argumento del tablero será una lista separada por comas de enteros sin espacios. Por ejemplo, para utilizar la estrategia de búsqueda bread-first para resolver la placa

de entrada dada por la configuración inicial {0,8,7,6,5,4,3,2,1}, el programa se ejecutará así (con sin espacios entre comas):

```
bfs 0,8,7,6,5,4,3,2,1
```

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

`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

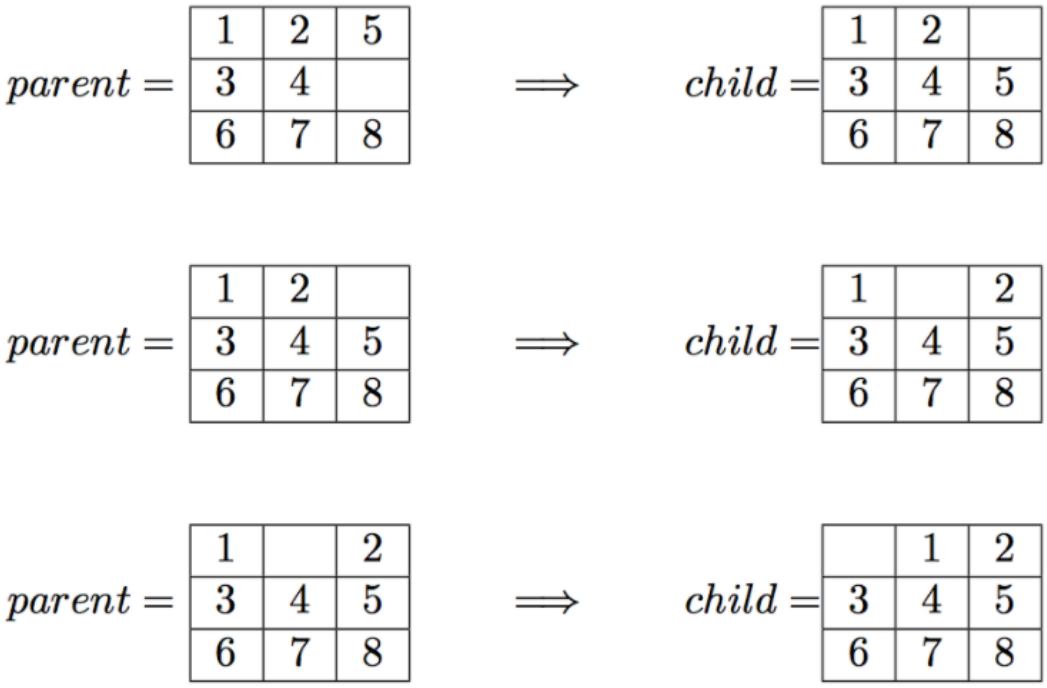
`max_ram_usage:` el uso máximo de RAM durante la vida útil del proceso.

Ejemplo n.°1: BFS

Suponga que el programa se ejecuta para la búsqueda en amplitud de la siguiente manera:

```
bfs 1,2,5,3,4,0,6,7,8
```

Lo que debería conducir a la siguiente solución para la placa de entrada:



El archivo de salida contendrá exactamente las siguientes líneas (puede consultar el archivo de salida de ejemplo adjunto):

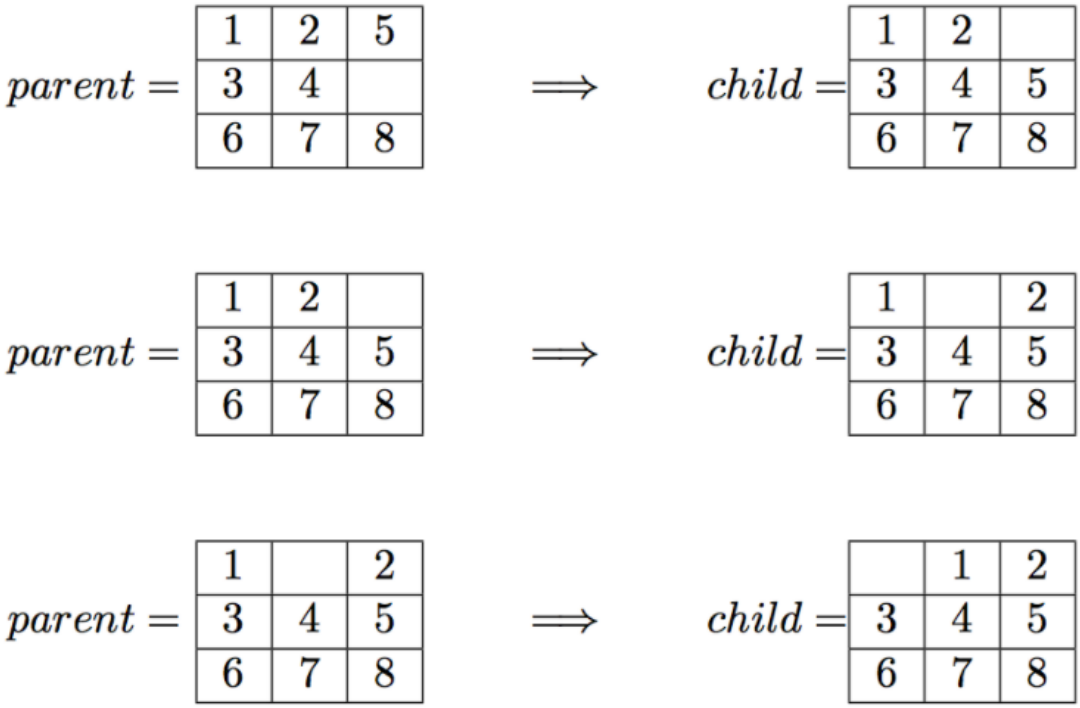
```
path_to_goal: ['Up', 'Left', 'Left']
cost_of_path: 3
nodos_expandidos: 10
profundidad_de_búsqueda: 3
max_search_depth: 4
running_time: 0.00188088
max_ram_usage: 0.07812500
```

Ejemplo n. º 2: DFS

Suponga que el programa se ejecuta para la búsqueda en profundidad de la siguiente manera:

```
dfs 1,2,5,3,4,0,6,7,8
```

Lo que debería conducir a la siguiente solución para la placa de entrada:



El archivo de salida contendrá exactamente las siguientes líneas (puede consultar el archivo de salida de ejemplo adjunto):

```
path_to_goal: ['Arriba', 'Izquierda', 'Izquierda']
cost_of_path: 3
nodes_expanded: 181437
profundidad_de_búsqueda: 3
max_search_depth: 66125
running_time: 5.01608433
max_ram_usage: 4.23940217
```

Nota sobre la corrección

Por supuesto, los valores específicos para las variables running_time y max_ram_usage variarán mucho dependiendo de la máquina utilizada y los detalles de implementación específicos; no hay un valor "correcto" para buscar. Están destinados a permitirle comprobar las características de complejidad de tiempo y espacio de su código, y debería dedicar tiempo a hacerlo. Todas las demás variables, sin embargo, tendrán una y solo una respuesta correcta para cada algoritmo y placa inicial especificada en los casos de prueba de muestra. * Una buena manera de verificar la exactitud de su programa es recorrer pequeños ejemplos a mano, como el los de arriba.

* En general, para cualquier tablero inicial, para BFS y DFS hay una y solo una respuesta correcta. Sin embargo, para A *, su salida de nodes_expanded puede variar un poco, dependiendo de los detalles de implementación específicos. Estará bien siempre que su algoritmo cumpla con todas las especificaciones enumeradas en estas instrucciones.

V. Información importante

Por favor lea la siguiente información cuidadosamente. Dado que este es el primer proyecto de programación, proporcionamos 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 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. Para la elección de heurística, utilice la función de prioridad de Manhattan; es decir, la suma de las distancias de las fichas desde sus posiciones de meta. Tenga en cuenta que el espacio en blanco no se considera un mosaico real aquí.

2. Orden de visitas

En esta asignación, donde se debe hacer una elección arbitraria, siempre visitamos los nodos secundarios en el orden "UDLR"; es decir, ["Up", "Down", "Left", "Right"] en ese orden exacto. 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. En particular, habrá **cinco casos de prueba en total**, cada uno probado en los tres algoritmos, para un total de **15 pruebas distintas**. Si implementa su código con diseños razonables de estructuras de datos, su código resolverá los 15 casos de prueba en un minuto en total. 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

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. Su envío recibirá crédito completo siempre que su programa de conductor proporcione la información correcta.

VI. Antes de terminar

- **Asegúrese** de que su código pase al menos los cuatro casos de prueba de envío.
- **Asegúrese** de que sus algoritmos generen la solución correcta para una instancia arbitraria de un problema solucionable de 8 rompecabezas.
- **Asegúrese** de que su programa siempre finalice sin errores y en un período de tiempo razonable. Recibirá cero puntos si su programa no termina. Los tiempos de ejecución de más de uno o dos minutos pueden indicar un problema con su implementación. Si su implementación excede el límite de tiempo asignado (20 minutos para todos los casos de prueba), su calificación puede estar incompleta.
- **Asegúrese** de que la salida de su programa siga exactamente el formato especificado. En particular, para la ruta hacia la meta, use corchetes para rodear la lista de elementos, use comillas simples alrededor de cada elemento y escriba en mayúscula la primera letra de cada elemento. Redondea los números de coma flotante a 8 lugares después del decimal. No recibirá el crédito adecuado del evaluador si su formato difiere de los ejemplos proporcionados anteriormente.