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

# PREGUNTAS FRECUENTES SOBRE EL PROYECTO PUZZLE-8

## P. ¿Hay un código básico disponible para la tarea?

**R.** Sí, hay un código esqueleto disponible para su uso adjunto. Esto es completamente opcional de usar y puede modificarlo tanto como desee. También puede completar la tarea sin consultar el código esqueleto en absoluto.

## P. Mi algoritmo de búsqueda parece correcto, pero es demasiado lento. ¿Cómo puedo reducir su tiempo de ejecución?

**R.** El algoritmo de búsqueda es quizás uno de los mejores materiales de aprendizaje para la complejidad computacional y la idiosincrasia de Python. Hay cuatro pros y contras:

**1.** No almacene miembros de datos posiblemente grandes, como la ruta de la solución, en la clase de nodo del árbol de búsqueda. En cambio, reconsidere qué operación debería ser rápida.

> Explicación: *El almacenamiento de una ruta desde el nodo raíz en cada clase de nodo logra un tiempo de búsqueda de O (1) a expensas del tiempo de creación de O (n). Por ejemplo, si se visita el estado actual después de 60.000 estados intermedios, el estado actual debe asignar una lista de 60.000 elementos y cada uno de los estados secundarios debe asignar una lista de 60.001 elementos. Esto pronto consumiría la memoria física y, por lo general, el sistema operativo de su máquina mata el proceso de búsqueda.*

Una observación clave es que, en el caso del algoritmo de búsqueda, la operación de búsqueda de ruta se ejecuta solo una vez después de que finaliza la búsqueda. Por lo tanto, la operación de búsqueda está bien para ser lenta. Puede considerar otra estructura de datos que tenga O (n) tiempo de búsqueda para la ruta de la solución, pero requiera O (1) operaciones durante la búsqueda.

**2.** No se conforme con usar la lista como frontera. En su lugar, diseñe su clase Frontier que funcione más rápido.

> Explicación: *un cuello de botella importante de la clase de lista, deque o cola en Python es que su operación de prueba de membresía es O (n). La velocidad de prueba de pertenencia es fundamental para el algoritmo de búsqueda porque esa operación se ejecuta para cada estado secundario. Proponer el uso de estructuras de datos similares a listas es un buen primer paso, pero para*

*usarlas con un tiempo de ejecución razonable, es posible que necesite un truco más.*

Tenga en cuenta que el pseudocódigo en las diapositivas de las conferencias no refleja necesariamente los detalles de implementación (es decir, el tiempo y el espacio). Más bien, muestra conceptualmente las entradas, las órdenes de procesamiento y las salidas del algoritmo. Una de sus misiones en esta tarea es hacer realidad la "frontera" utilizando las primitivas de "bajo nivel" de Python.

**3.** No utilice la operación O (n) cuando tenga otra forma más rápida de hacer lo mismo.

Explicación: *Hablando en términos generales, si configura una operación O (n) debajo de su para vecino en el bucle de vecinos, es muy probable que su código exceda el límite de tiempo de calificación. En otras palabras, sucede que su código se ejecuta drásticamente más rápido cuando corrige solo una línea de su código.*

Por ejemplo, fusionar dos conjuntos y comprobar que un elemento está en el conjunto fusionado es una operación costosa, mientras que comprobar que un elemento está en uno de los dos conjuntos es una operación O (1).

**P. ¿Necesito optimizar mi algoritmo de búsqueda tanto como sea posible?**

**R.** No es necesario que exprima el rendimiento de su código mediante técnicas de optimización sofisticadas, como el desplazamiento de bits o la reducción del número de llamadas a funciones. A excepción de copy.deepcopy (), en la mayoría de sus opciones de diseño se tratan de elegir las mejores estructuras de datos en términos de complejidad de tiempo / espacio.