# **Ejemplos y ejercicios de busqueda A\***

## **Ejemplo 1**

Imagina una cuadrícula 2D donde cada celda puede ser:

* Libre
* Ocupada por un obstáculo

Queremos encontrar el camino más corto desde un punto inicial (start) a un punto final (goal).

**Estructura de Datos**

1. **Nodo (Node)**: Representa una celda en la cuadrícula.

* position: Coordenadas (x, y)
* g: Costo desde el inicio hasta este nodo
* h: Heurística (estimación del costo desde este nodo hasta el objetivo)
* f: g + h
* parent: Nodo padre para reconstruir el camino

1. **Cola de Prioridad**: Utilizada para seleccionar el nodo con el menor valor f.
2. **Conjunto de Nodos Visitados**: Para evitar volver a visitar nodos.

Probar el código del archivo ejemplo1.py

**Detalle del Ejemplo**

1. **Inicialización**:
   * Se crea el nodo inicial con costo g=0, heurística h calculada mediante la distancia Manhattan, y se pone en la cola de prioridad (open\_list).
2. **Búsqueda**:
   * En cada iteración, se extrae el nodo con el menor costo f de la cola de prioridad.
   * Si el nodo actual es el objetivo, se reconstruye el camino hacia atrás desde el objetivo utilizando los nodos padres.
   * Se generan los vecinos del nodo actual (arriba, abajo, izquierda, derecha).
   * Para cada vecino, se calcula el costo g, la heurística h, y el costo total f.
   * Si el vecino no está en la lista de cerrados ni en la cola de prioridad con un costo f mayor, se agrega a la cola de prioridad.
3. **Resultado**:
   * El camino se reconstruye a partir de los nodos padres, si se encuentra un camino al objetivo.

Este ejemplo cubre un escenario básico y puede ser ampliado para manejar más casos, como el manejo de obstáculos dinámicos, distintos tipos de heurísticas, o cuadrículas tridimensionales.

## **Ejemplo 2**

Una aplicación que utiliza el algoritmo A\* para encontrar el camino más corto en un entorno 2D, pero con características adicionales como:

1. **Costos de Movimiento**: Diferentes tipos de terreno tienen diferentes costos.
2. **Costos de Movimiento Diagonales**: Permite moverse en diagonal con un costo adicional.
3. **Interfaz Gráfica**: Usa una interfaz gráfica para visualizar el entorno y el camino encontrado.

Usaremos Python con pygame para la interfaz gráfica y heapq para la cola de prioridad. Este ejemplo mostrará cómo el algoritmo A\* se puede aplicar a un entorno visual y cómo se puede gestionar el costo de movimiento.

Ver y ejecutar el código en pygame.py

1. **Configuración Inicial**:
   * Se define el tamaño de la ventana y la cuadrícula.
   * Los colores se definen para las celdas (blanco para libre, negro para obstáculos, verde para el camino).
2. **Nodo**:
   * La clase Node representa cada celda en la cuadrícula con atributos para los costos g, h y f, y un nodo padre para reconstruir el camino.
3. **Función Heurística**:
   * La función heuristic calcula la distancia Manhattan entre dos puntos.
4. **Algoritmo A**\*:
   * Se busca el camino más corto considerando las celdas libres y los obstáculos.
   * Se permite el movimiento en diagonal con un costo adicional (1.414, que es la raíz cuadrada de 2).
5. **Interfaz Gráfica**:
   * Usamos pygame para dibujar la cuadrícula y el camino encontrado.
   * draw\_grid visualiza el entorno y el camino en la ventana de pygame.
6. **Ejecución**:
   * La función main inicializa la ventana de pygame, configura la cuadrícula con algunos obstáculos, ejecuta el algoritmo A\* y actualiza la ventana en un bucle hasta que se cierra.

Este ejemplo proporciona una visión más completa y visual del algoritmo A\* en acción, y puede ser ampliado con más características según las necesidades de tu proyecto.

## **Ejemplo 3**

**Ejemplo con Heurística compuesta**

En este ejemplo, combinaremos A\* con una heurística compuesta que combina la distancia Manhattan (para la distancia rectilínea) y la distancia Euclidiana (para la distancia directa en línea recta) para mejorar el rendimiento en un entorno de búsqueda.

### Ejemplo de A\* con Heurística Compuesta

Ver y ejecutar archivo compuesta.py

**Detalles del Código**

1. **Heurísticas**:
   * heuristic\_manhattan(a, b): Calcula la distancia Manhattan, útil para un grid rectilíneo.
   * heuristic\_euclidean(a, b): Calcula la distancia Euclidiana, útil para considerar la distancia directa.
2. **Heurística Combinada**:
   * combined\_heuristic(a, b): Combina ambas heurísticas (Manhattan y Euclidiana) y utiliza el promedio para mejorar la estimación del costo.
3. A Modificado\*:
   * Usa la heurística combinada para calcular el costo estimado h de cada nodo.
   * La función astar realiza la búsqueda con la heurística compuesta para encontrar el camino más corto.
4. **Interfaz Gráfica**:
   * La visualización se realiza con pygame, mostrando la cuadrícula y el camino encontrado.

**Beneficios de la Heurística Combinada**

La heurística combinada puede ofrecer un balance entre la precisión y la eficiencia de la búsqueda:

* **Heurística de Manhattan**: Proporciona una estimación simple basada en movimientos rectilíneos.
* **Heurística Euclidiana**: Proporciona una estimación más precisa para movimientos diagonales.

## **Ejercicio 4**

Realice alguno de los ejemplos de búsqueda con Heuristica combinada

El algoritmo A\* con heurísticas combinadas se puede aplicar a una variedad de problemas en diferentes dominios. Aquí hay algunos ejemplos de aplicaciones prácticas:

### 1. **Navegación en Juegos de Video**

**Problema**: Un personaje en un juego necesita encontrar el camino más corto desde su posición actual hasta un objetivo, evitando obstáculos y zonas peligrosas.

**Aplicación**:

* **Escenario**: Un juego de rol (RPG) en un entorno 2D con un mapa lleno de obstáculos, como paredes y enemigos.
* **Uso de A**\*: Utilizar A\* con heurística combinada para permitir que el personaje encuentre el camino más rápido y eficiente hacia el objetivo, teniendo en cuenta diferentes tipos de terreno (como pasillos estrechos o áreas peligrosas).
* **Beneficio**: Mejora la experiencia del jugador al proporcionar movimientos fluidos y naturales para el personaje en el entorno del juego.

### 2. **Robótica y Navegación Autónoma**

**Problema**: Un robot necesita navegar por un entorno desconocido, evitando obstáculos y buscando el objetivo de manera eficiente.

**Aplicación**:

* **Escenario**: Un robot de limpieza en una oficina o un robot de entrega en un almacén.
* **Uso de A**\*: Implementar A\* con una heurística combinada para planificar la ruta más corta y eficiente en un entorno con obstáculos dinámicos.
* **Beneficio**: Optimiza el tiempo de navegación y evita colisiones con obstáculos en tiempo real.

### 3. **Planificación de Rutas en Sistemas de Transporte**

**Problema**: Un sistema de transporte necesita encontrar la ruta más corta entre dos puntos en una red de carreteras o un sistema de transporte público.

**Aplicación**:

* **Escenario**: Un sistema de navegación para automóviles o una aplicación de transporte público que planifica rutas entre paradas.
* **Uso de A**\*: Utilizar A\* con una heurística combinada para encontrar rutas óptimas teniendo en cuenta diferentes factores como el tráfico, la distancia y el tiempo.
* **Beneficio**: Mejora la eficiencia del viaje y reduce el tiempo de viaje al encontrar las rutas más rápidas.

### 4. **Optimización de Recursos en Redes de Datos**

**Problema**: Optimizar el enrutamiento de datos a través de una red para minimizar el tiempo de transmisión y evitar congestionamientos.

**Aplicación**:

* **Escenario**: Un sistema de enrutamiento en una red de comunicaciones o en una red de computadoras.
* **Uso de A**\*: Implementar A\* con heurísticas para encontrar el camino más rápido a través de la red, teniendo en cuenta el ancho de banda y los tiempos de respuesta.
* **Beneficio**: Mejora la velocidad y la eficiencia de la transmisión de datos en la red.

### 5. **Optimización de Procesos en Manufactura**

**Problema**: Planificar y optimizar los movimientos de robots o sistemas automatizados en una línea de producción.

**Aplicación**:

* **Escenario**: Un sistema de manufactura automatizado que debe mover piezas a través de una serie de estaciones de trabajo.
* **Uso de A**\*: Utilizar A\* para planificar las rutas de los robots o transportadores dentro de la línea de producción, evitando colisiones y optimizando el flujo de trabajo.
* **Beneficio**: Aumenta la eficiencia de la producción y reduce el tiempo de ciclo.

### 6. **Resolución de Puzzles y Juegos de Estrategia**

**Problema**: Resolver puzzles complejos o juegos de estrategia que requieren encontrar el camino óptimo o una solución eficiente.

**Aplicación**:

* **Escenario**: Juegos de tablero como el Sudoku o problemas de laberintos complejos.
* **Uso de A**\*: Aplicar A\* con heurísticas para explorar diferentes configuraciones y encontrar soluciones óptimas en juegos o puzzles.
* **Beneficio**: Proporciona soluciones rápidas y eficientes a problemas complejos de manera automatizada.