Memoria Práctica 1
Fundamentos de los Sistemas Inteligentes

Integrantes:

- Carlos Luis Miranda Hernández
- Adrián Robaina Mejías

Objetivos:

La meta de la Práctica 1 es completar el código base añadiendo la programación de los algoritmos de **Ramificación y Acotación** (simple y con subestimación). Una vez funcionales, el objetivo es poner a prueba y **contrastar los resultados** de los cuatro algoritmos.

Ramificación y Acotación:

Este algoritmo se emplea como estrategia de búsqueda para resolver problemas de optimización; en este contexto, su objetivo es hallar la ruta de coste mínimo entre una ciudad de origen y una de destino.

La implementación propuesta explora el grafo gestionando el coste acumulado (g) hacia cada nodo. Para garantizar la optimalidad, se incorpora un mecanismo de control de nodos visitados: si se alcanza un nodo previamente explorado mediante un camino más corto, se actualiza la información almacenada, descartando la ruta anterior más costosa.

La gestión de la frontera de búsqueda (*fringe*) se realiza mediante una **cola de prioridad**. Aunque existen diversas formas de implementación, se ha optado por insertar los nodos ordenados de menor a mayor coste acumulado.

Ramificación y Acotación con Subestimación:

Esta variante introduce el uso de una **heurística** (h) para guiar la búsqueda, definida aquí como la distancia en línea recta al objetivo. La diferencia fundamental radica en el criterio de ordenación de la cola de prioridad: en lugar de considerar únicamente el coste acumulado, se

utiliza una función de evaluación que suma dicho coste y la estimación heurística ($f = g + h$). Esto permite priorizar los caminos con mayor potencial de ser óptimos.

Partes Obligatorias

Nodos visitados y generados:

Para poder comparar los resultados, el código lleva la cuenta de dos tipos de nodos:

- **visited (Visitados):** Mide cuántos nodos analiza realmente el algoritmo. Este contador se incrementa cada vez que extraemos un elemento de la cola de prioridad (*fringe*) para procesarlo.
- **generated (Generados):** Mide cuántos nodos se crean en total en el árbol de búsqueda. Se incrementa cada vez que se expande un nodo, sumando todos sus nuevos hijos o sucesores al total acumulado.

Ruta encontrada:

La solución final se obtiene mediante el método proporcionado en el código base. Al analizar las rutas devueltas:

- La **Búsqueda en Profundidad (DFS)** tiende a devolver rutas mucho más largas (subóptimas), ya que este algoritmo no explora para encontrar el mejor camino, sino que se detiene en cuanto encuentra el primero disponible, sea bueno o malo.
- Por el contrario, los dos algoritmos de **Ramificación y Acotación** (con y sin subestimación) han encontrado la misma ruta. Al ser algoritmos óptimos, garantizan devolver siempre la mejor solución posible. La diferencia entre ambos no está en la ruta que dan (que es idéntica), sino en la eficiencia con la que la encuentran (menos nodos generados).

Coste Total:

Este valor representa la suma de los costes de las aristas del camino desde el origen hasta el objetivo.

- Los mejores resultados (el coste mínimo) se obtienen con los algoritmos de **Ramificación y Acotación**, cumpliendo con su propiedad de optimalidad.
- La **Búsqueda en Amplitud (BFS)** arroja valores cercanos, pero no siempre óptimos. Esto se debe a que BFS busca el camino con "menos ciudades intermedias" (menor profundidad), lo cual no siempre significa "menos kilómetros" (menor coste). Por

tanto, su resultado depende mucho de la estructura del mapa y no garantiza el coste mínimo.

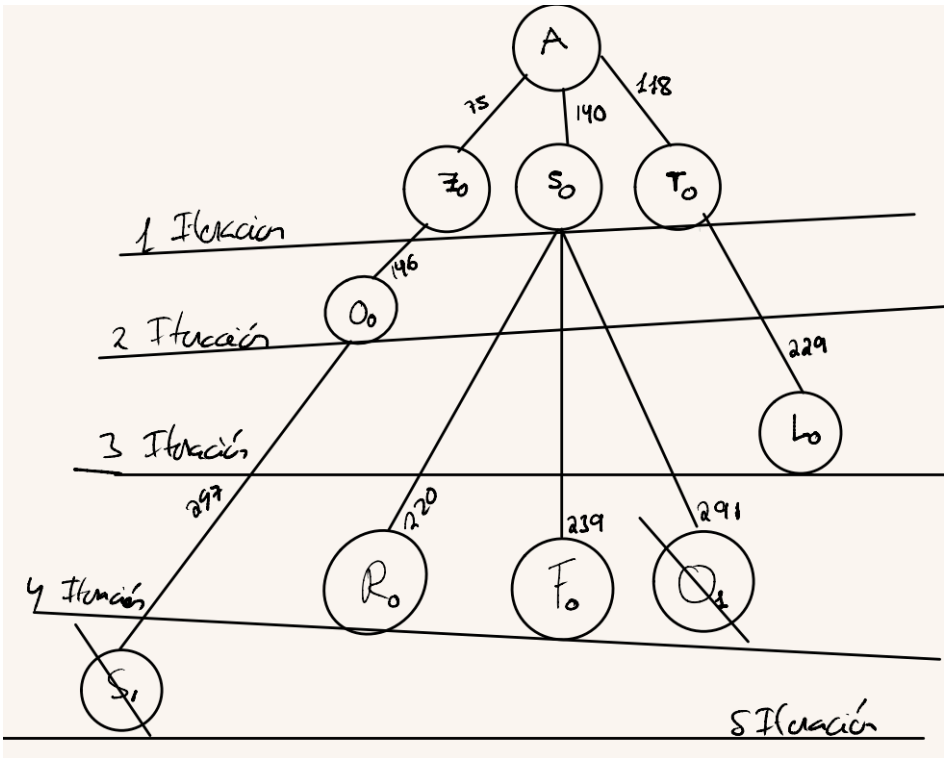
Partes Opcionales

Tiempo Ejecución:

Para medir la velocidad de los algoritmos hemos usado perf_counter() de la librería time. Básicamente, tomamos el tiempo justo antes de empezar y al terminar, y calculamos la diferencia.

Como estos problemas se resuelven muy rápido, el resultado en segundos nos daba números con demasiados ceros (tipo 0.000...), lo que hacía difícil compararlos. Por eso decidimos multiplicar ese valor por un millón y mostrarlo directamente en **microsegundos**. Así los datos son mucho más fáciles de leer y se ve mejor la diferencia real entre uno y otro.

Grafo de 5 Iteraciones:



Heurística Mala:

Una **heurística ineficiente** (ej: restar la distancia en lugar de sumarla) altera el orden de exploración. En consecuencia, al subestimar excesivamente, el algoritmo pierde su capacidad de guía. Aunque sí garantiza encontrar el óptimo, se comporta peor que una búsqueda ciega, generando y visitando una cantidad masiva de nodos innecesarios.