

PRÁCTICA 2

DARÍO MENESES HERNÁNDEZ

PLANIFICACIÓN DE RUTAS HOSPITALARIAS MEDIANTE ALGORITMOS DE BÚSQUEDA

https://github.com/dmh1006/Practica2_DarioMeneses2C

CONTENIDO

PLANIFICACIÓN DE RUTAS HOSPITALARIAS MEDIANTE ALGORITMOS DE BÚSQUEDA	1
INTRODUCCIÓN	2
ALGORÍTMOS IMPLEMENTADOS	2
Búsqueda en anchura (Breadth-First Search)	2
Búsqueda voraz (Greedy Best-First Search)	3
Algoritmo A*	3
MODELADO DEL PROBLEMA.....	3
CLASES DESARROLLADAS	4
Clase ProblemaRuta	4
Clase NodoBusqueda	4
Clase MetricasBusqueda	5
Clase ResultadoBusqueda.....	5
Clase PlanificadorTramos	5
DIAGRAMA DE CLASES	6
DESARROLLO DEL EJERCICIO 1	6
Planteamiento	6
Configuración del problema.....	6
Ejecución de los algoritmos	7
Obtención de resultados y métricas	7
VISUALIZACION DEL RECORRIDO.....	8
DESARROLLO DEL EJERCICIO 2	8
Planteamiento del ejercicio.....	8
Planificación por tramos	8
Resultados y análisis	9
Visor de ruta	10
MANUAL DE USO	10
CONCLUSIONES	11

INTRODUCCIÓN

En esta práctica se aborda el problema de la planificación de rutas en un entorno hospitalario mediante el uso de algoritmos de búsqueda, tomando como referencia un hospital modelado como una matriz bidimensional. Cada celda del mapa representa un tipo de estancia diferente (pasillos, consultas, quirófanos, unidades especiales, etc.), asociándose a cada una de ellas un coste de tránsito que refleja las restricciones y penalizaciones propias de un entorno sanitario real.

El objetivo principal de la práctica es analizar y comparar el comportamiento de distintos algoritmos de búsqueda, tanto no informados como informados, a la hora de encontrar rutas óptimas dentro del hospital. Para ello, se estudian no solo la longitud del camino obtenido, sino también el coste total de la ruta y el esfuerzo computacional requerido por cada algoritmo, medido en términos de nodos expandidos, tamaño de la frontera y tiempo de ejecución.

El problema planteado va más allá de una simple búsqueda del camino más corto, ya que el hospital presenta zonas con diferentes penalizaciones. Esto obliga a tratar el problema como uno de optimización, donde se pretende minimizar el coste total del recorrido evitando, en la medida de lo posible, atravesar áreas sensibles como quirófanos o unidades de cuidados intensivos, priorizando el uso de pasillos.

Todo el desarrollo se ha realizado en Python, siguiendo una estructura modular que separa claramente el modelado del problema, la implementación de los algoritmos de búsqueda y la ejecución de los distintos ejercicios propuestos. Esta organización permite reutilizar el mismo modelo del problema con diferentes estrategias de búsqueda y facilita la comparación objetiva de los resultados obtenidos.

ALGORÍTMOS IMPLEMENTADOS

Para la resolución del problema de planificación de rutas hospitalarias se han implementado y comparado tres algoritmos de búsqueda clásicos: búsqueda en anchura, búsqueda voraz y el algoritmo A*. Estos algoritmos presentan enfoques diferentes a la hora de explorar el espacio de estados, lo que permite analizar sus ventajas e inconvenientes en un entorno con costes heterogéneos.

BÚSQUEDA EN ANCHURA (BREADTH-FIRST SEARCH)

La búsqueda en anchura es un algoritmo no informado que explora el espacio de estados por niveles, expandiendo primero todos los nodos situados a la misma profundidad antes de avanzar a niveles más profundos. Su función de evaluación puede expresarse como:

$$f(n) = g(n)$$

donde $g(n)$ representa el número de pasos desde el estado inicial hasta el nodo actual.

Este algoritmo garantiza encontrar una solución con el menor número de movimientos posibles, siempre que todas las acciones tengan el mismo coste. Sin embargo, no tiene en cuenta las penalizaciones asociadas a las distintas zonas del hospital, por lo que puede generar rutas con un coste total elevado al atravesar áreas altamente penalizadas. Además, su carácter exhaustivo

provoca un gran número de nodos expandidos y un mayor tiempo de ejecución, especialmente en mapas de gran tamaño.

BÚSQUEDA VORAZ (GREEDY BEST-FIRST SEARCH)

La búsqueda voraz es un algoritmo informado que utiliza una función heurística para estimar la cercanía de cada estado al objetivo. En este caso, se emplea la distancia Manhattan como heurística, definida como la suma de las diferencias absolutas entre las coordenadas del estado actual y el estado objetivo. Su función de evaluación viene dada por:

$$f(n) = h(n)$$

Este enfoque permite guiar la búsqueda de forma más directa hacia el objetivo, reduciendo significativamente el número de nodos expandidos y el tiempo de ejecución en comparación con la búsqueda en anchura. No obstante, al ignorar el coste acumulado del camino recorrido, el algoritmo puede seleccionar rutas que, aunque parezcan prometedoras según la heurística, atraviesen zonas con altas penalizaciones. Por este motivo, la búsqueda voraz no garantiza obtener la solución de menor coste total.

ALGORITMO A*

El algoritmo A* es un algoritmo informado que combina el coste acumulado del camino recorrido con una estimación heurística de la distancia al objetivo. Su función de evaluación se define como:

$$f(n) = g(n) + h(n)$$

Gracias a esta combinación, A* logra un equilibrio entre la exploración eficiente del espacio de estados y la minimización del coste total de la ruta. Al tener en cuenta las penalizaciones del entorno hospitalario, el algoritmo tiende a evitar zonas de alto coste cuando existen alternativas más favorables, obteniendo soluciones más óptimas que las proporcionadas por la búsqueda en anchura y la búsqueda voraz.

En el contexto de esta práctica, A* se presenta como el algoritmo más adecuado para la planificación de rutas en entornos con costes heterogéneos, ya que ofrece una buena relación entre calidad de la solución y eficiencia computacional.

MODELADO DEL PROBLEMA

El problema de planificación de rutas planteado en esta práctica se ha modelado como un problema clásico de búsqueda en espacios de estados, en el que el entorno hospitalario se representa mediante una matriz bidimensional. Cada celda de la matriz corresponde a una posición del hospital y se asocia a un tipo de estancia concreto, como pasillos, consultas, quirófanos o unidades especiales.

Cada estado del problema se define como una posición del agente dentro del mapa, representada mediante un par de coordenadas (*fila*, *columna*). El estado inicial corresponde a la posición de partida definida en cada ejercicio, mientras que el estado objetivo se establece como la celda de pasillo más cercana a la localización deseada, garantizando que todos los objetivos sean alcanzables por los algoritmos de búsqueda.

Las acciones disponibles desde cada estado consisten en desplazamientos a las cuatro direcciones cardinales (norte, sur, este y oeste). Para que una acción sea válida, el estado resultante debe encontrarse dentro de los límites del mapa y corresponder a una celda transitable, descartándose aquellas acciones que conduzcan a muros o zonas no accesibles.

El coste asociado a cada transición depende del tipo de celda a la que se accede. De este modo, el tránsito por pasillos tiene un coste reducido, mientras que el acceso a zonas sensibles del hospital presenta penalizaciones más elevadas. Esta asignación de costes permite reflejar de forma realista las restricciones del entorno y transforma el problema en uno de optimización, en el que no solo se busca alcanzar el objetivo, sino hacerlo minimizando el coste total del recorrido.

Para los algoritmos informados se emplea una heurística basada en la distancia Manhattan hasta el objetivo, calculada como la suma de las diferencias absolutas entre las coordenadas del estado actual y el estado objetivo. Esta heurística es admisible y consistente en este dominio, lo que garantiza un comportamiento correcto del algoritmo A* y permite guiar la búsqueda de forma eficiente sin sobreestimar el coste restante.

CLASES DESARROLLADAS

Con el objetivo de mantener una implementación modular, reutilizable y fácilmente extensible, el desarrollo de la práctica se ha estructurado en varias clases que separan claramente el modelado del problema, la representación de los nodos de búsqueda, la gestión de métricas y la obtención de resultados. Esta organización permite aplicar distintos algoritmos de búsqueda sobre el mismo modelo sin necesidad de modificar su formulación.

CLASE PROBLEMARUTA

La clase ProblemaRuta se encarga de definir formalmente el problema de búsqueda asociado a la navegación por el hospital. En ella se especifican los elementos fundamentales del problema: el estado inicial, el estado objetivo, la generación de sucesores, el cálculo del coste de las transiciones y la heurística utilizada por los algoritmos informados.

Cada estado representa una posición válida dentro del mapa del hospital. El método encargado de generar los sucesores evalúa todas las posibles acciones desde un estado dado, calculando el nuevo estado resultante y comprobando que se encuentre dentro de los límites del mapa y que sea transitable. Para cada sucesor válido se devuelve la acción realizada, el nuevo estado y el coste asociado a la transición, obtenido a partir de las penalizaciones definidas para cada tipo de celda.

Además, esta clase define la función heurística utilizada por los algoritmos voraz y A*, basada en la distancia Manhattan hasta el objetivo. Gracias a esta implementación, los algoritmos de búsqueda pueden operar de manera independiente del entorno concreto, reutilizando el mismo modelo del problema para distintos ejercicios y configuraciones.

CLASE NODOBUSQUEDA

La clase NodoBusqueda representa un nodo dentro del árbol de búsqueda generado por los algoritmos. Cada nodo encapsula un estado del problema, una referencia a su nodo padre y la acción que ha permitido alcanzar dicho estado. Esta información permite reconstruir el camino solución una vez alcanzado el estado objetivo, recorriendo la cadena de nodos desde el objetivo hasta el estado inicial.

Además, la clase almacena el coste acumulado $g(n)$ desde el estado inicial hasta el nodo actual, así como el valor heurístico $h(n)$ cuando corresponde. A partir de estos valores se calcula la función de evaluación $f(n)$, utilizada para priorizar la exploración de nodos en los algoritmos informados. La clase implementa también un criterio de comparación entre nodos basado en dicha función de evaluación, lo que permite su uso directo en estructuras de datos de prioridad como colas con heap.

CLASE METRICASBUSQUEDA

La clase MetricasBusqueda se utiliza para registrar y gestionar las métricas asociadas a la ejecución de los algoritmos de búsqueda. Durante cada ejecución se almacenan valores como el número de nodos expandidos, el tamaño máximo alcanzado por la frontera y el tiempo total de ejecución.

El uso de una clase específica para la gestión de métricas permite separar la lógica del algoritmo de la evaluación de su rendimiento, facilitando la posterior comparación objetiva entre las distintas estrategias de búsqueda. Estas métricas son fundamentales para justificar las diferencias observadas entre los algoritmos en términos de eficiencia y esfuerzo computacional.

CLASE RESULTADOBUSQUEDA

La clase ResultadoBusqueda actúa como un contenedor que encapsula el resultado final de la ejecución de un algoritmo de búsqueda. En ella se almacenan de forma estructurada datos como si se ha encontrado o no solución, el camino obtenido, el coste total de la ruta, la longitud del camino y las métricas asociadas a la búsqueda.

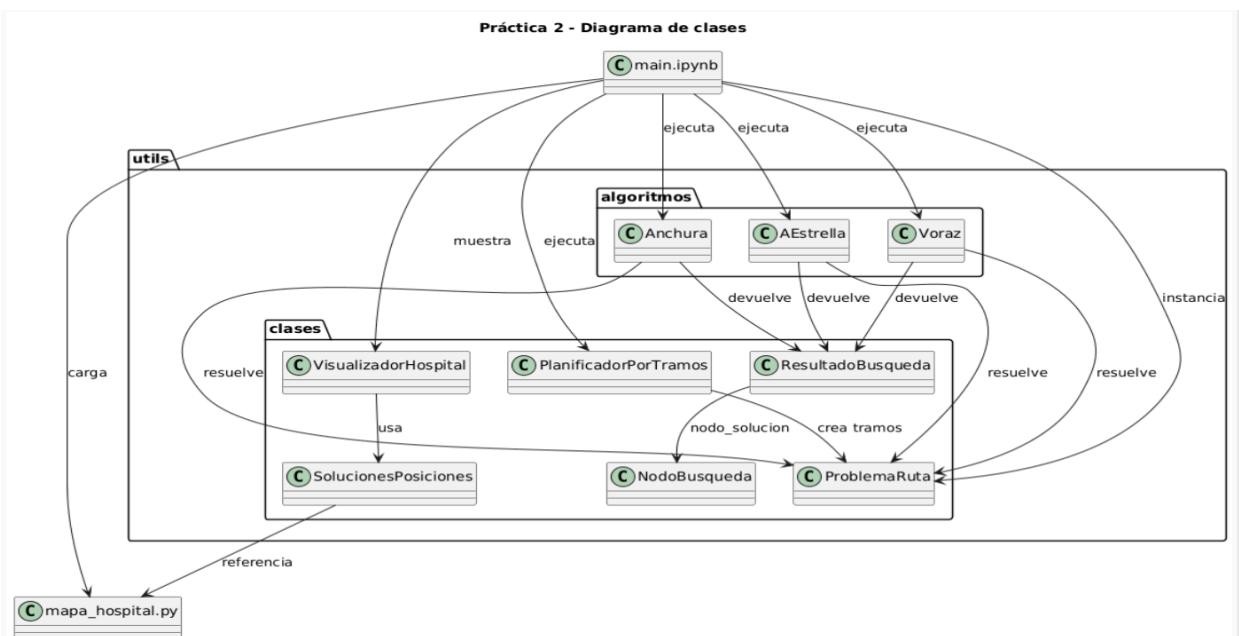
Gracias a esta estructura común, los resultados de los distintos algoritmos pueden tratarse y compararse de manera homogénea, independientemente de la estrategia utilizada para obtenerlos. Esto resulta especialmente útil a la hora de presentar los resultados en tablas comparativas y analizar el comportamiento de cada algoritmo.

CLASE PLANIFICADORTRAMOS

Para la resolución del segundo ejercicio, en el que es necesario visitar múltiples servicios del hospital en un orden determinado, se ha desarrollado la clase PlanificadorTramos. Esta clase permite dividir el problema global en una secuencia de tramos independientes, resolviendo cada uno de ellos mediante un algoritmo de búsqueda y concatenando posteriormente los caminos obtenidos.

En cada tramo, el estado inicial se corresponde con la posición final del tramo anterior, mientras que el objetivo se define como la celda de pasillo más cercana al siguiente servicio a visitar. Además de concatenar los caminos parciales, la clase se encarga de acumular el coste total y las métricas de todos los tramos, proporcionando un resultado global coherente. Esta aproximación permite abordar problemas de planificación complejos sin modificar los algoritmos de búsqueda subyacentes y garantiza una implementación modular y reutilizable.

DIAGRAMA DE CLASES



DESARROLLO DEL EJERCICIO 1

Ruta desde la celda de pasillo más cercana a D22 hasta la celda de pasillo más cercana a D1

PLANTEAMIENTO

El objetivo de este ejercicio es obtener una ruta dentro del hospital desde una posición inicial asociada a la zona **D22** hasta una posición objetivo-asociada a la zona **D1**, considerando que la navegación se realiza sobre celdas transitables del mapa y que cada tipo de celda tiene un coste de tránsito definido mediante penalizaciones.

El problema se formula como un problema de búsqueda sobre estados (*fila*, *columna*), en el que los sucesores se generan mediante movimientos en las cuatro direcciones cardinales y se descartan las transiciones a muros (celdas "M") o posiciones fuera de rango.

El coste de la ruta se calcula sumando el coste de acceder a cada celda destino según la tabla de penalizaciones. De esta forma, el ejercicio permite comparar algoritmos que minimizan pasos frente a algoritmos que minimizan coste total.

CONFIGURACIÓN DEL PROBLEMA

Para resolver el ejercicio se crea una instancia de **ProblemaRuta** con:

- **Estado inicial**: coordenadas del pasillo seleccionado como punto de inicio asociado a D22.

- **Estado objetivo:** coordenadas del pasillo seleccionado como punto objetivo asociado a D1.
- **Mapa:** matriz del hospital.
- **Penalizaciones:** diccionario de costes por tipo de celda.

ProblemaRuta incorpora:

- sucesores(estado) para generar transiciones válidas.
- coste_estado(estado) para aplicar el coste de entrar en el estado destino (penalización).
- heuristica_estado(estado) con distancia Manhattan al objetivo para los algoritmos informados.

EJECUCIÓN DE LOS ALGORITMOS

Se ejecutan tres algoritmos sobre el mismo modelo del problema, asegurando una comparación homogénea gracias a ResultadoBusqueda.

6.3.1 Búsqueda en anchura (BFS)

Se aplica busqueda_anchura, que explora el espacio por niveles mediante una cola FIFO (deque). Durante la ejecución registra métricas y devuelve un ResultadoBusqueda.

6.3.2 Búsqueda voraz

Se aplica busqueda_voraz, priorizando la heurística estableciendo $f(n) = h(n)$. La selección del siguiente nodo se realiza extrayendo el mínimo f mediante recorrido de la estructura de abiertos.

6.3.3 Algoritmo A*

Se aplica busqueda_a_estrella, combinando coste real y heurística con $f(n) = g(n) + h(n)$. Para evitar expandir caminos peores hacia un mismo estado, mantiene un diccionario mejor_g con el mejor coste acumulado encontrado para cada estado. Al igual que en voraz, extrae el mínimo f recorriendo la lista de abiertos.

OBTENCIÓN DE RESULTADOS Y MÉTRICAS

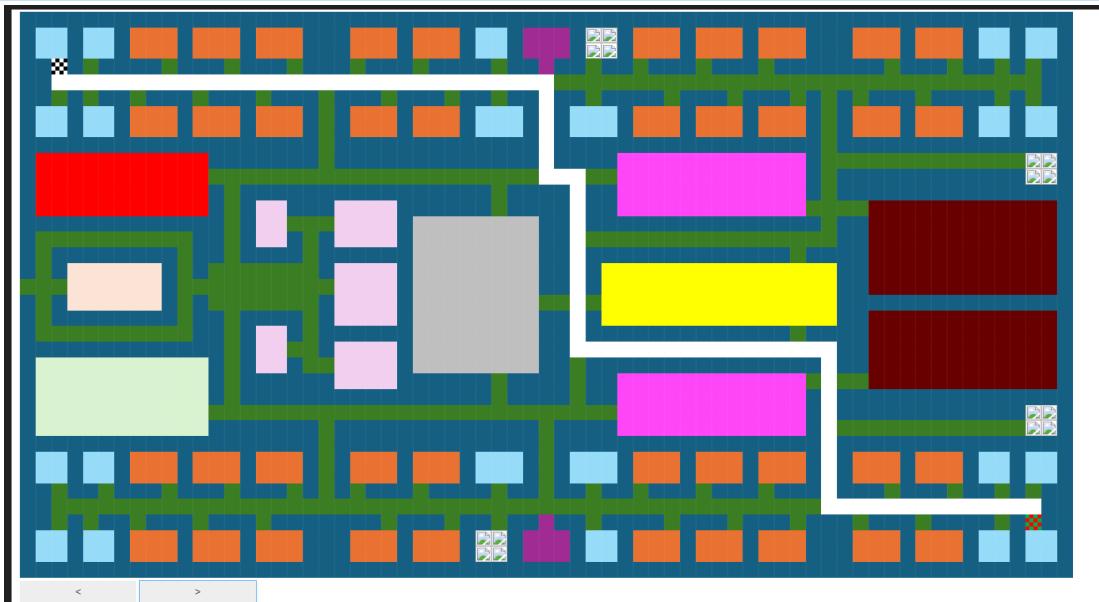
Todos los algoritmos retornan un ResultadoBusqueda, lo que permite:

- Recuperar el **camino** con get_camino().
- Recuperar el **coste total** con get_coste().
- Acceder a las **métricas**: nodos expandidos, frontera máxima y tiempo (ms).

	Algoritmo	Hay solución	Coste	Pasos	Nodos expandidos	Frontera máx	Tiempo (ms)
0	Anchura (BFS)	True	8087	91	1329	36	8.054972
1	Voraz	True	8087	91	123	41	1.071692
2	A*	True	91	91	258	70	2.836227

En la Tabla del Ejercicio 1 se observa que BFS y Voraz obtienen la misma longitud de camino (91 pasos) y un coste elevado (8087), lo que indica que ambos encuentran una ruta corta en número de movimientos pero que atraviesa zonas con alta penalización. En contraste, A* mantiene la misma longitud del camino, pero reduce el coste a 91, evidenciando que prioriza trayectos por pasillos y evita celdas penalizadas al incorporar el coste acumulado en la función de evaluación. En términos de eficiencia, BFS expande muchos más nodos al no estar guiado, mientras que Voraz minimiza expansiones gracias a la heurística. A* presenta un equilibrio: expande más que Voraz para garantizar una solución de menor coste, pero sigue siendo significativamente más eficiente que BFS.

VISUALIZACION DEL RECORRIDO



DESARROLLO DEL EJERCICIO 2

Ruta del responsable de seguridad

PLANTEAMIENTO DEL EJERCICIO

En este ejercicio se plantea la planificación de una ruta que permita a un responsable de seguridad comprobar que todas las puertas de los diferentes servicios del hospital se encuentran cerradas. La ruta debe comenzar desde una puerta principal y recorrer los servicios en el orden indicado en el enunciado, pasando por todos ellos de forma secuencial.

El entorno hospitalario se modela del mismo modo que en el ejercicio anterior, representándose mediante una matriz bidimensional con distintos tipos de celdas y penalizaciones asociadas. Dado que los algoritmos de búsqueda implementados únicamente permiten desplazarse por celdas transitables, el problema se formula como una planificación de rutas sobre pasillos, garantizando que todos los objetivos sean alcanzables.

El reto principal de este ejercicio no es únicamente encontrar una ruta válida, sino hacerlo respetando estrictamente el orden de visita de los servicios y minimizando el coste total del recorrido, lo que convierte el problema en un caso de planificación multiobjetivo.

PLANIFICACIÓN POR TRAMOS

Para resolver el ejercicio se emplea una planificación por tramos, implementada mediante la clase PlanificadorTramos. En lugar de buscar una única ruta que cubra todos los servicios, el problema global se divide en una secuencia de subproblemas independientes, cada uno correspondiente a un tramo entre dos objetivos consecutivos.

En cada tramo se crea un nuevo problema de búsqueda utilizando la clase ProblemaRuta, donde:

- El estado inicial es la posición final del tramo anterior.
- El estado objetivo es el siguiente servicio a visitar, representado como una celda de pasillo alcanzable.
- Se reutiliza el mismo mapa del hospital y el mismo conjunto de penalizaciones.

Cada tramo se resuelve utilizando uno de los algoritmos de búsqueda implementados (búsqueda en anchura, búsqueda voraz o A*). Los caminos parciales obtenidos se concatenan para formar la ruta completa, evitando duplicar el estado inicial de cada tramo. De forma simultánea, se acumulan el coste total del recorrido y las métricas de búsqueda (nodos expandidos, frontera máxima y tiempo de ejecución), obteniéndose un resultado global coherente.

Esta estrategia permite mantener una implementación modular y reutilizable, ya que no es necesario modificar los algoritmos de búsqueda para abordar un problema con múltiples objetivos, y garantiza que el recorrido final respeta el orden de visita especificado en el enunciado.

RESULTADOS Y ANÁLISIS

Al ejecutar los distintos algoritmos sobre todos los tramos definidos, se obtienen rutas completas que recorren correctamente todos los servicios en el orden establecido. Sin embargo, el comportamiento de los algoritmos difiere de forma notable en términos de coste total y esfuerzo computacional.

La búsqueda en anchura consigue completar la ruta de seguridad respetando el orden de visita, pero a costa de un número muy elevado de nodos expandidos y un mayor tiempo de ejecución. Este comportamiento se debe a que el algoritmo no utiliza información heurística ni considera el coste de las celdas durante la exploración, lo que provoca una búsqueda exhaustiva en cada tramo.

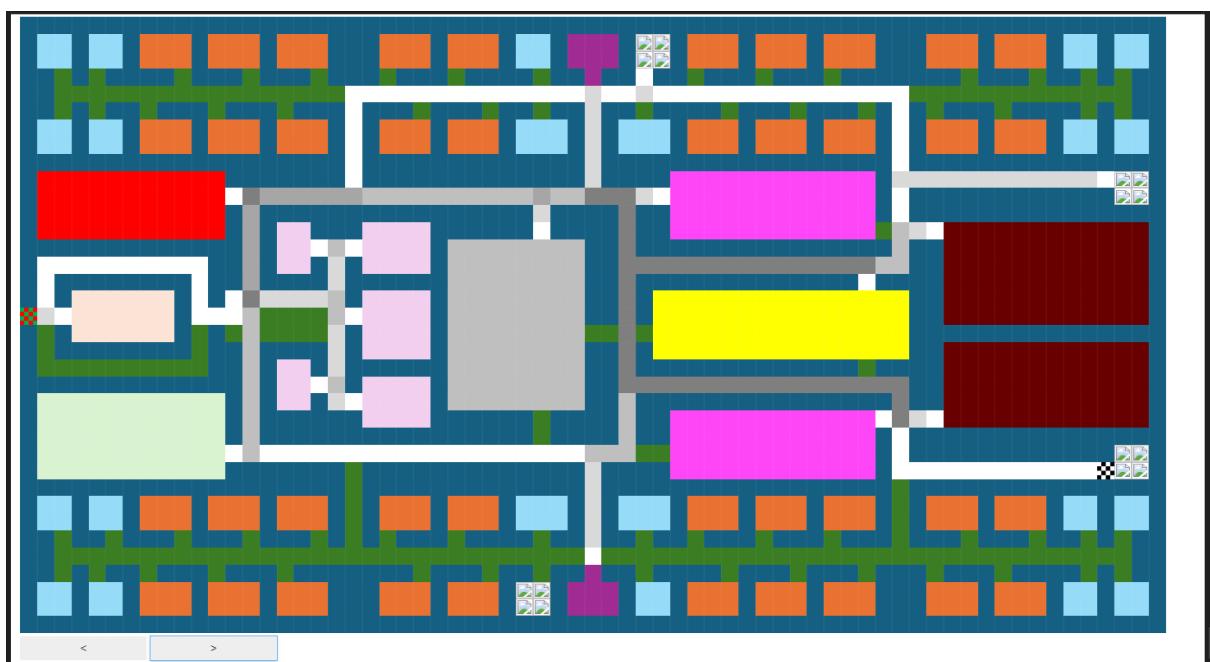
La búsqueda voraz reduce de manera significativa el número de nodos expandidos y el tiempo total de ejecución gracias al uso de la heurística, guiando la búsqueda de forma más directa hacia cada objetivo parcial. No obstante, al no tener en cuenta el coste acumulado del recorrido, el algoritmo atraviesa con frecuencia zonas altamente penalizadas, lo que se traduce en un coste total elevado para la ruta completa.

Por su parte, el algoritmo A* presenta nuevamente el comportamiento más equilibrado. Al combinar el coste real acumulado con la heurística, es capaz de evitar zonas de alto coste siempre que existen alternativas razonables, obteniendo rutas de menor coste total sin un incremento excesivo del esfuerzo computacional. Estos resultados confirman la idoneidad de A* para problemas de planificación multiobjetivo en entornos con penalizaciones heterogéneas.

	Algoritmo	Hay solución	Coste total	Pasos	Nodos expandidos	Frontera máx	Tiempo (ms)
0	Anchura (BFS)	True	74466	502	8239	43	57.531118
1	Voraz	True	87480	522	896	35	5.921125
2	A*	True	594	594	1162	44	8.713245

En el Ejercicio 2, al tratarse de una planificación multiobjetivo resuelta por tramos, las diferencias entre algoritmos se intensifican. La búsqueda en anchura completa la ruta pero requiere un esfuerzo computacional muy elevado (8239 nodos expandidos y 57.5 ms), debido a la ausencia de heurística y a la repetición de búsquedas en cada tramo. La búsqueda voraz reduce drásticamente el tiempo y los nodos expandidos al guiarse únicamente por la heurística, pero al ignorar el coste acumulado atraviesa con frecuencia zonas penalizadas, obteniendo el mayor coste total (87480). Por el contrario, A* combina el coste real con la heurística, logrando minimizar el coste total del recorrido (594), lo que indica que prioriza rutas por pasillos y evita zonas de alto coste; todo ello con un esfuerzo computacional moderado, superior al voraz pero muy inferior a BFS.

VISOR DE RUTA



MANUAL DE USO

El proyecto se ha desarrollado en Python y se organiza en distintos módulos que separan el modelado del problema, la implementación de los algoritmos de búsqueda, la gestión de métricas y la visualización de resultados. El punto de entrada principal es un notebook que permite ejecutar los ejercicios y analizar los resultados de forma interactiva.

Para ejecutar la práctica en un equipo local, es necesario clonar o descargar el repositorio del proyecto y situarse en el directorio raíz. Se recomienda crear y activar un entorno virtual de Python para aislar las dependencias del proyecto. Las librerías necesarias se encuentran especificadas en el archivo requirements.txt y pueden instalarse mediante el comando correspondiente del gestor de paquetes de Python.

Una vez configurado el entorno, se debe abrir el notebook principal y ejecutar sus celdas de forma secuencial. En primer lugar se cargan el mapa del hospital, las penalizaciones y las clases necesarias para el modelado del problema y la ejecución de los algoritmos. A continuación, se definen los parámetros de cada ejercicio, como el estado inicial, los objetivos y el orden de visita de los servicios.

El notebook ejecuta los distintos algoritmos de búsqueda sobre el mismo modelo del problema y almacena los resultados en estructuras comunes, lo que permite presentar los datos obtenidos mediante tablas comparativas. Además, el proyecto incluye un sistema de visualización que representa gráficamente los recorridos sobre el mapa del hospital y permite navegar paso a paso por la ruta obtenida, facilitando la comparación visual entre algoritmos.

CONCLUSIONES

En esta práctica se ha abordado el problema de la planificación de rutas hospitalarias mediante la aplicación y comparación de distintos algoritmos de búsqueda en un entorno con costes heterogéneos. A través de los dos ejercicios propuestos se ha podido analizar tanto la calidad de las rutas obtenidas como el esfuerzo computacional requerido por cada estrategia.

Los resultados muestran que la búsqueda en anchura no resulta adecuada para este tipo de entornos, ya que, aunque garantiza rutas con un número mínimo de pasos, ignora las penalizaciones del mapa y requiere expandir un gran número de nodos. La búsqueda voraz mejora notablemente la eficiencia gracias al uso de la heurística, pero al no considerar el coste acumulado puede generar soluciones rápidas pero poco óptimas en términos de coste total.

El algoritmo A* combina de forma equilibrada el coste real del recorrido con la información heurística, obteniendo rutas de menor coste y manteniendo una eficiencia razonable incluso en problemas de planificación multiobjetivo. Por este motivo, A* se presenta como la estrategia más adecuada para la planificación de rutas en entornos hospitalarios con penalizaciones, como el planteado en esta práctica.