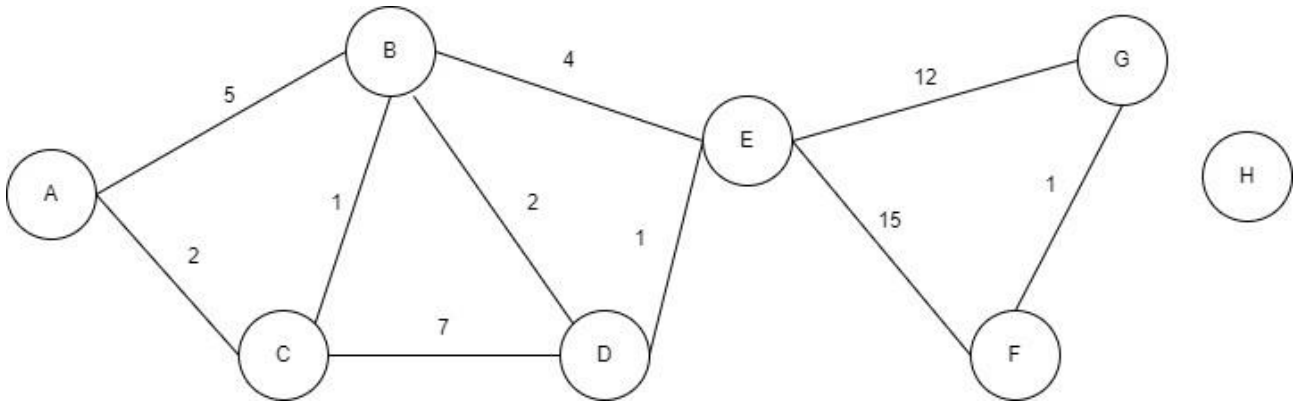


IABD – PROGRAMACIÓN DE INTELIGENCIA ARTIFICIAL PRÁCTICA
EVALUABLE 3.2-UT3 – búsquedas con a*

Dado el siguiente grafo, resuelve los siguientes ejercicios:



1. Sigue manualmente los pasos del algoritmo A* para llegar desde A hasta E, rellenando la siguiente tabla:

LISTA ABIERTA	LISTA CERRADA	NODO_SELECCIONADO	NODOS HIJOS
A	-	A	B, C
B, C	A	C	B, D
B, D	A, C	B	D, E
D, E	A, C, B	D	E
E	A, C, B, D	E	-

2. Crea un programa en Python para representar el grafo a través de la librería networkX. Elige las coordenadas apropiadas para que el gráfico se asemeje al mostrado arriba.
3. Crea un programa en Python para representar el grafo con las clases Node y Graph del módulo graph.py y codifica:

- a) Una función que muestre la estructura del grafo con matplotlib*
- b) Un programa que muestre:
 1. los vecinos del nodo B
 2. Si los nodos G y H están conectados
 3. Si los nodos D y E están conectados
 4. Número de nodos del grafo
 5. Lista de nodos del grafo

* Para mostrar el coste entre nodos puedes situar la etiqueta con el método annotate en el punto medio de las coordenadas de dos nodos:

El punto medio de (x1,y1) y (x2,y2) es (x1+x2)/2, (y1+y2)/2

4. Crea una función reset_graph(g) que borre los rastros de una búsqueda anterior, es decir, que elimine los padres de cada nodo, establezca a -1 su valor heurístico (heuristic_value) e infinito a su coste desde el inicio (distance_from_start).

5. Crea una función ruta(origen,destino,grafo) que utilizando el módulo AStar.py muestre la ruta entre el origen y el destino

IABD – PROGRAMACIÓN DE INTELIGENCIA ARTIFICIAL PRÁCTICA
EVALUABLE 3.2-UT3 – búsquedas con a*

6. Realiza las siguientes búsquedas:
 - a) El camino más corto entre el nodo A y el nodo B
 - b) El camino más corto entre el nodo A y el nodo G
 - c) El camino más corto entre el nodo A y el nodo H
 - d) El camino más corto entre el nodo G y el nodo C
7. Realiza las modificaciones necesarias al módulo AStar.py para que, en cada iteración del algoritmo muestre:
 - La listas de nodos abiertos y cerrados
 - El nodo seleccionado
 - Los nodos hijos del nodo seleccionado
8. Realiza la búsqueda entre el nodo A* y el nodo E y compara los resultados del ejercicio 1 y la ejecución del caso A->E en el ejercicio 7. ¿Son iguales? ¿A qué se deben las diferencias?
 - Los resultados no son iguales en cuanto a la secuencia de nodos, pero sí en el coste total. En el ejercicio 7 selecciona y expande nodos estrictamente. Si dos nodos tienen el mismo coste, el algoritmo elige uno antes que otro en función del orden en que fueron añadidos, mientras que la búsqueda del ejercicio 1 es manual.
9. Conversa un poco con ChatGPT de OpenAI o similar y pregúntale:
 - a) Si se considera A* un algoritmo de inteligencia artificial
 - Sí, **A*** se considera un **algoritmo de inteligencia artificial** dentro del campo de **búsqueda informada**. Es utilizado en problemas de planificación y navegación, ya que combina las ventajas de la búsqueda de costo uniforme (Dijkstra) y la búsqueda heurística (Greedy). Se usa en videojuegos, robótica y sistemas de navegación, por lo que es ampliamente aplicado en IA.
 - b) Qué algoritmo de búsqueda utiliza Google Maps para encontrar rutas
 - Google Maps usa una combinación de algoritmos según el contexto:
Dijkstra para encontrar rutas más cortas en redes pequeñas.
A* en entornos grandes donde se necesita una heurística para acelerar la búsqueda.
 - **Bidirectional A*** (A* Bidireccional), que busca desde el punto de origen y destino simultáneamente para mejorar la eficiencia.
 - **Contracción jerárquica (CH)**, un método avanzado que acelera la búsqueda en mapas grandes.
Google Maps también optimiza rutas en tiempo real considerando tráfico y restricciones dinámicas.

c) Qué alternativas hay al algoritmo A*

Existen varias alternativas a A* dependiendo del problema a resolver:

- **Dijkstra**: Encuentra la ruta más corta, pero no usa heurística, por lo que es más lento en grandes mapas.
- **D (Dynamic A)****: Variante de A* que recalcula rutas en tiempo real, útil en robótica.
- **Jump Point Search (JPS)**: Optimización de A* en mapas con cuadrículas, útil en videojuegos.
- **Búsqueda IDA*** (Iterative Deepening A*): Usa menos memoria al aplicar búsqueda en profundidad con poda heurística.
- **Búsqueda Greedy (Voraz)**: Expande el nodo más cercano al objetivo, pero no siempre encuentra la ruta óptima.

Cada algoritmo tiene ventajas y desventajas según el contexto.

d) Para qué se usa el algoritmo de Dijkstra

El **algoritmo de Dijkstra** se usa para encontrar la ruta más corta en un grafo con pesos positivos. Se aplica en múltiples áreas, como:

- **Sistemas de navegación y mapas** (GPS, Google Maps).
- **Redes de telecomunicaciones** (optimización de tráfico de datos).
- **Planificación de rutas en logística y transporte**.
- **Sistemas de recomendación en redes sociales**.

Es un método fundamental en teoría de grafos y optimización.

e) Para qué se usa el algoritmo de Dijkstra en redes de comunicaciones

En **redes de comunicación** (Internet, telecomunicaciones), Dijkstra se usa para:

- **Enrutamiento de paquetes** en protocolos como **OSPF (Open Shortest Path First)** en redes IP.
- **Optimización de ancho de banda** en redes de fibra óptica y redes inalámbricas.
- **Gestión de tráfico en redes** para reducir la latencia y evitar congestiones.

Ayuda a encontrar las rutas más eficientes para transmitir datos a través de redes complejas.

f) Si se considera Dijkstra un algoritmo de inteligencia artificial

- No, **Dijkstra no es considerado un algoritmo de inteligencia artificial**, sino un **algoritmo de teoría de grafos y optimización**. Se usa para encontrar rutas óptimas, pero no tiene componentes de aprendizaje o toma de decisiones adaptativa, características esenciales de la IA. Sin embargo, Dijkstra **sí es usado dentro de sistemas de IA** en combinación con otros algoritmos para planificación de rutas y problemas de búsqueda.

10. Analiza la respuesta a la pregunta f y contesta: ¿Qué razón hay para que A* se considere inteligencia artificial y Dijkstra no?

- A* utiliza una función heurística para predecir qué caminos son mejores. No explora todo el grafo, sino que prioriza los caminos más prometedores. Dijkstra explora todos los caminos posibles hasta encontrar el más corto.

11. Pídele a la IA que te muestre el algoritmo de A* y compáralo con el que has utilizado en la práctica.

```
from queue import PriorityQueue

class Node:
    def __init__(self, value, x, y):
        self.value = value
        self.x = x
        self.y = y
        self.g = float('inf') # Coste desde
# el inicio
        self.h = 0 # Heurística
        self.f = float('inf') # g + h
        self.parent = None # Nodo anterior
# en la ruta

    def __lt__(self, other):
        return self.f < other.f # Para
# ordenar en la cola de prioridad

def manhattan_distance(node1, node2):
    return abs(node1.x - node2.x) +
    abs(node1.y - node2.y)

def a_star(graph, start, goal):
    open_list = PriorityQueue()
    open_list.put((0, start))
    start.g = 0
    start.h = manhattan_distance(start,
    goal)
    start.f = start.h
    closed_list = set()

    while not open_list.empty():
        _, current = open_list.get()

        if current == goal:
            path = []
            while current:
                path.append(current.value)
                current = current.parent
            return path[::-1]

        closed_list.add(current)

        for neighbor, cost in
graph[current]:
            if neighbor in closed_list:
                continue

            tentative_g = current.g + cost
            if tentative_g < neighbor.g:
                neighbor.parent = current
                neighbor.g = tentative_g
                neighbor.h =
manhattan_distance(neighbor, goal)
                neighbor.f = neighbor.g +
neighbor.h
                open_list.put((neighbor.f,
neighbor))

    return None # No se encontró un camino
```

- En mi código, el manejo de los nodos abiertos puede ser diferente, y la estructura de datos puede no ser una cola de prioridad. En el algoritmo, los valores g, h y f se manejan dentro de cada nodo, mientras que en la práctica la forma de calcularlos puede variar.

12. Pídele a la IA que te compare el algoritmo A* y el de Dijkstra. Según el chatbot, ¿Cuál es más eficiente?

Característica	A*	Dijkstra
Estrategia	Búsqueda informada (usa heurística)	Búsqueda no informada (no usa heurística)
Función de coste	$f(n) = g(n) + h(n)$ (coste real + heurística)	Solo $g(n)$ (coste real desde el inicio)
Exploración de nodos	Prioriza los nodos con mejor estimación hacia el objetivo	Explora todos los nodos en orden de menor coste acumulado
Eficiencia	Más rápido en problemas grandes si la heurística es buena	Más lento porque revisa más nodos innecesarios
Aplicaciones	Juegos, planificación de rutas, robótica	Redes, telecomunicaciones, mapas

- A* es más eficiente en problemas grandes porque usa heurística para evitar expandir nodos innecesarios.

- Dijkstra revisa todos los nodos accesibles hasta el destino, por lo que es más lento en mapas grandes.
- Si la heurística de A es mala*, entonces A puede comportarse igual que Dijkstra*, perdiendo su ventaja.

13. Por último, pregúntale qué otros algoritmos de inteligencia artificial para búsquedas hay y comenta cuál te ha llamado más la atención.

- **Búsqueda en amplitud (BFS - Breadth First Search)**
 - Explora todos los nodos a un mismo nivel antes de profundizar.
 - Es útil para encontrar la ruta más corta en grafos no ponderados.
 - **Ejemplo:** Resolver un laberinto sin considerar pesos en los caminos.
- **Búsqueda en profundidad (DFS - Depth First Search)**
 - Explora un camino hasta el final antes de retroceder.
 - No garantiza la ruta más corta, pero es útil en problemas de exploración.
 - **Ejemplo:** Buscar una salida en un laberinto explorando caminos completos.
- **Búsqueda en profundidad iterativa (IDDFS - Iterative Deepening DFS)**
 - Combina BFS y DFS, usando profundidad limitada en cada iteración.
 - **Ejemplo:** Juegos de ajedrez donde hay que explorar muchas jugadas.
- **Búsqueda voraz (Greedy Best-First Search)**
 - Expande siempre el nodo más cercano al objetivo según una heurística.
 - Puede ser más rápida, pero no siempre encuentra la mejor solución.
 - **Ejemplo:** GPS que prioriza ir "hacia adelante" sin considerar si es el camino más corto.
- **Búsqueda con algoritmo de Monte Carlo (MCTS - Monte Carlo Tree Search)**
 - Se usa en inteligencia artificial para juegos.
 - Explora diferentes caminos aleatoriamente y aprende cuál es mejor con simulaciones.
 - **Ejemplo:** Juegos como ajedrez, Go o inteligencia artificial en videojuegos.
- **Algoritmos evolutivos y genéticos**
 - No buscan caminos, sino que "**evolucionan**" soluciones mediante selección natural.
 - **Ejemplo:** Optimización de rutas en logística y diseño de redes neuronales.

Me llamó la atención el algoritmo de Monte Carlo (MCTS) porque se usa en juegos de estrategia. En lugar de seguir reglas estrictas, prueba movimientos aleatorios y aprende cuál es mejor, lo que lo hace más "inteligente" en entornos complejos. También me pareció