**Prompts:**

**1) “Podrías explicarme cómo funciona la heurística en el algoritmo de A\*, para saber cómo el algoritmo entiende qué nodos seguir y cuáles no. Menciona a qué hace referencia con la distancia Manhattan.”**

**- Para qué sirvió:** Me permitió esclarecer las dudas que tenía respecto al algoritmo de A\*. Debido a que no tenía del todo claro a qué hacía referencia con la función heurística y cómo el algoritmo podía saber por dónde tenía que ir. Además, no comprendía a qué hacía referencia con el camino manhattan y cómo es que se relaciona con la heurística.

**- Qué fue lo aprendido:** La función heurística es una estimación de qué tan cerca se está del objetivo. Teniendo la idea central de decidir qué nodo explorar primero según un puntaje f(n). Se tiene la función f(n) = g(n) + h(n), siendo g(n) es el costo real desde el nodo inicial hasta el nodo n (es decir, lo que ya se recorrió), y h(n) la heurística (estimación del costo para llegar desde n hasta el objetivo). Básicamente, la heurística es la “intuición” del algoritmo sobre qué camino parece más prometedor. Si h(n) es pequeña, significa que el nodo n está estimado cerca del objetivo. Por otro lado, la distancia Manhattan es un tipo de heurística para tableros cuadrados (por ejemplo, un laberinto, un mapa en grilla). Los movimientos se limitan en líneas rectas horizontales o verticales, no en diagonal. Teniendo dos puntos, la función se calcula haciendo h(n) = |x1 – x2| + |y1 – y2|. Siendo |x1 – x2| el número de casillas que hay que moverse horizontalmente. Y |y1 – y2| es el número de casillas que hay que moverse verticalmente.

Para saber el camino que debe tomar A\*, primero mantiene una cola de prioridad de nodos a explorar, basada en f(n) = g(n) + h(n). Siempre explora el nodo con menor f(n). Luego, calcula g(n) a medida que avanza y actualiza h(n) según la heurística elegida (por ejemplo, la Manhattan). De esta forma, evita nodos que aumentarían demasiado f(n) y continúa hasta que el nodo objetivo se extrae de la cola, garantizando el camino más corto si la heurística es admisible.

**2) “Cómo logra el algoritmo de Bellman-Ford, detectar ciclos negativos y al mismo tiempo poder trabajar con pesos negativos. Explícame detalladamente el concepto de relajación de aristas y cómo se vincula con el algoritmo.”**

**- Para qué sirvió:** Permitió entender en profundidad la funcionalidad del algoritmo de Bellman – Ford, ya sea con su vinculación con los ciclos negativos y con los pesos negativos. Al mismo tiempo, ver qué diferencias posee si se lo compara con el algoritmo de Dijkstra.

**- Qué fue lo aprendido:** La relajación de aristas es la operación para propagar mejoras en las estimaciones de distancia desde la fuente. La relajación sólo propaga la información de que existe una mejor ruta. Repetir relajaciones en todo el grafo deja que las mejores distancias fluyan desde la fuente hacia todos los nodos.

Bellman – Ford usa la relajación como bloque de construcción y aplica relajación sobre todas las aristas repetidamente. Repite la operación de relajar todas las aristas V-1 veces, siendo V la cantidad de vértices. Si después de esas V-1 iteraciones aún existe alguna arista que puede relajarse, entonces hay un ciclo de peso negativo alcanzable desde la fuente. La razón de que sean V-1 iteraciones, radica en que cualquier camino simple (sin vértices repetidos) entre dos vértices contiene a lo sumo V-1 aristas. Si no hay ciclos negativos, la mejor distancia a cualquier nodo se alcanza con algún camino simple, por lo tanto, relajando todas las aristas V-1 veces cualquier mejora habrá llegado a su destino. Bellman – Ford detecta un ciclo negativo de la siguiente forma: Si existe un ciclo de suma negativa accesible desde la fuente, se puede dar vueltas a ese ciclo y reducir indefinidamente el costo total de la ruta alcanzando cada vértice del ciclo. Luego de V-1 iteraciones, si se vuelve a recorrer todas las aristas y se encuentra que alguna arista todavía permite una relajación, eso implica que hay una ruta que usa más de V-1 aristas que tiene menor coste, lo que implica que esa ruta contiene un ciclo negativo, debido a que los caminos simples no superan las V-1 aristas. Sin ciclos negativos cualquier camino óptimo de "s" a “v” es simple y usa <= V-1 aristas. Al relajar todas las aristas V-1 veces, la información de la ruta óptima ha tenido tiempo de propagarse a “v”. Por lo que después de V-1 rondas no hay más mejoras posibles.

Bellman – Ford funciona con pesos negativos porque no asume que una vez que se fija un nodo como “final” su distancia es definitiva. No sella distancias, continúa relajado y actualizado hasta converger (o detectar un ciclo negativo). Si se lo compara con Dijkstra, este último asume que cuando se extrae el nodo con menor “dist” de la cola de prioridad, esa distancia es la distancia definitiva mínima a ese nodo. Esto no es cierto si existen aristas negativas, puede ocurrir que más adelante aparezca una ruta más barata que llega a ese nodo pasando por otro con coste reducido por un peso negativo.

**3) “Podrías hacerme una comparación detallada entre los siguientes algoritmos: BFS, Dijkstra, Floyd-Warshall, Bellman-Ford, A\* y Prim/Kruskal. Ten en cuenta la complejidad utilizada por cada uno. El tipo de grafo con el que trabajan, ya sea dirigido, no dirigido, ponderado, no ponderado, utilizando pesos negativos.”**

**- Para qué sirvió:** Permitió ver de una forma clara y concisa las diferencias entre los algoritmos. De esta forma es más fácil entender las características de cada uno.

**- Qué fue lo aprendido:** BFS encuentra las distancias mínimas en grafos no ponderados (o con todos los pesos iguales a 1). También sirve para recorridos por capas o niveles. Funciona en grafos ya sean dirigidos o no dirigidos, siempre y cuando no sean ponderados. Su complejidad es O(V + E) usando lista de adyacencia. BFS sirve como base para otros algoritmos.

Dijkstra encuentra caminos mínimos desde un origen a todos los demás vértices. Funciona en grafos dirigidos o no dirigidos, de los cuales deben ser ponderados, sin embargo, no soporta pesos negativos. Su complejidad puede ser O(V2) con matriz y O((V + E) log V) con lista y cola de prioridad.

Bellman – Ford encuentra caminos mínimos desde un origen a todos los vértices, siendo capaz de detectar ciclos negativos. Funciona en grafos dirigidos o no dirigidos, de los cuales deben ser ponderados. Su complejidad es O(V \* E).

Floyd – Warshall: Encuentra caminos mínimos entre todos los pares de vértices. Funciona para grafos dirigidos o no dirigidos, de los cuales deben ser ponderados. También, soporta pesos negativos, pero no detecta ciclos negativos. Su complejidad es O(V3).

A\* encuentra el camino mínimo desde un origen hasta un destino específico. Funciona en grafos dirigidos o no dirigidos, de los cuales deben ser ponderados con pesos mayores o iguales a 0. No funciona con pesos negativos, dado que la heurística pierde su validez. Su complejidad es O(E log V).

Prim encuentra el árbol recubridor mínimo. Funciona para grafos no dirigidos, con ponderaciones mayores o iguales a 0. Su complejidad es O(V2) con matriz y O(E log V) con heap.

Kruskal encuentra un árbol recubridor mínimo (MST). Funciona para grafos no dirigidos y ponderados con pesos mayores o iguales a 0. Su complejidad es O(E log E).