Introducción:

El laberinto del Pacman se modela como un **grafo implícito** $G = (V, E)$, donde V son las casillas transitables y $E \subseteq V \times V$ representa movimientos legales entre celdas adyacentes. Cada estado $s = (x, y)$ corresponde a una posición de Pacman, y getSuccessors(s) devuelve (s', a, c), donde s'es un sucesor, a la acción para alcanzarlo y c el coste del movimiento.

Q1.

En **DFS**, se utiliza una **pila** para los nodos pendientes y un conjunto $Visited$ para evitar ciclos. Se inicia con (s_0, π_0), donde π_0 es la trayectoria vacía. En cada iteración se extrae (s, π); si s es la meta, se devuelve π. De lo contrario, se apilan los sucesores no visitados $(s', \pi + [a])$y se marca s'como visitado.

Este proceso continúa hasta alcanzar el objetivo o hasta que la estructura de datos esté vacía, en cuyo caso no existe camino y se devuelve [].

Q2.

En **BFS**, se usa una **cola**, manteniendo la misma estructura (s, π) y $Visited$ que en el algoritmo DFS. Se extrae por orden de llegada y se expanden los sucesores no visitados $(s', \pi + [a])$, garantizando que la primera vez que se alcanza la meta se obtiene la **ruta más corta en número de pasos**.

Este proceso continúa hasta alcanzar el objetivo o que la estructura de datos esté vacía, en cuyo caso no existe camino y se devuelve [].

Q3.

En **UCS**, se utiliza una **cola de prioridad** para los nodos pendientes y un conjunto $Visited$ para evitar reexpansiones. Se inicia con $(s_0, \pi_0, 0)$, donde $\pi_0$es la trayectoria vacía y 0 el coste acumulado. En cada iteración se extrae el nodo con menor coste; si s es la meta, se devuelve π. De lo contrario, se expanden los sucesores no visitados $(s', \pi + [a], g(s) + w(s, s'))$y se actualiza su coste acumulado. Este proceso continúa hasta alcanzar el objetivo, garantizando que la ruta encontrada tenga **coste mínimo**. Si la cola se vacía sin hallar la meta, se devuelve [].

Q4.

En **A***, se utiliza una **cola de prioridad** para los nodos pendientes, donde la prioridad es $f(s) = g(s) + h(s)$, y un conjunto $Visited$ o un diccionario de costos para evitar reexpansiones innecesarias. Se inicia con $(s_0, \pi_0, 0)$, donde π_0 es la trayectoria vacía y 0 el coste acumulado.

En cada iteración se extrae el nodo con menor $f(s)$; si s es la meta, se devuelve π. De lo contrario, se expanden los sucesores no visitados o con un coste menor al conocido $(s', \pi + [a], g(s) + w(s, s'))$, y se actualiza su coste acumulado y prioridad $f(s') = g(s') + h(s')$.

Este proceso continúa hasta alcanzar el objetivo, garantizando que la ruta encontrada tenga coste mínimo si la heurística es admisible. Si la cola se vacía sin hallar la meta, se devuelve [].

Q5.

En el problema de las esquinas, se emplea un espacio de estados abstractos, donde cada estado se representa como la posición actual y las esquinas visitadas, evitando codificar información irrelevante. Se inicia con el vector (s_0, v_0) donde s_0 es la posición inicial del Pacman y $v_0 = (False, False, False, False)$ representa las esquinas visitadas. Posteriormente, en cada iteración, se extrae un nodo del frente de búsqueda y se verifica si todas las esquinas se han visitado. En caso de cumplirse, se devuelve la trayectoria π, en caso contrario, se expanden los sucesores legales que no han sido visitados, o si el costo acumulado es inferior al conocido $(s', \pi + [a], g(s) + 1)$. Para cada sucesor, se actualiza el estado de esquinas visitadas, y se añade a la estructura de datos de búsqueda con costo incremental 1. Este proceso sigue hasta que se alcance de meta, siendo este cuando Pacman ha visitado todas las esquinas. Si no alcanza la meta, devuelve un array vacío.

Q6.

La heurística calcula primero el conjunto de esquinas no visitadas R y, si está vacío, la consideramos trivial. Si aún quedan esquinas, se estima el coste mínimo para alcanzarlas simulando un recorrido "greedy", partiendo desde la posición actual $current$, busca la esquina $c \in R$, cuya distancia de Manhattan($d(current, c) = |x_{current} - x_c| + |y_{current} - y_c|$) sea mínima, se suma la distancia al total y se actualiza la posición actual con el nuevo valor. Se repite este suceso hasta que no queden esquinas sin visitar, obteniendo el valor final, el cual es la suma de todas las distancias mínimas sucesivas entre la posición actual y las esquinas más cercanas, lo que constituye una cota inferior del coste real del problema, garantizando una heurística admisible y consistente.

Q7

La heurística es calculada tomando la distancia más larga entre la posición actual del Pacman y cualquier comida restante. En caso de no haber comido, se devuelve 0 y en caso contrario, se emplea $mazeDistance$ para medir la distancia en el laberinto a cada comida y se devuelve el máximo valor, garantizando una cota inferior del coste real, haciendo que la heurística sea admisible y consistente.

Q8

La función encuentra un camino hasta la comida más cercana, generando un $AnyFoodSearchProblem$, cuyo objetivo se cumple al llegar a cualquier punto de la comida, y luego resolviendo el problema con BFS (Explicación del algoritmo en la pregunta Q2). De esta forma, el agente siempre llegará a la comida más próxima de forma rápida y eficiente, aunque no se garantiza que el camino total sea óptimo