Introducción:

Para este trabajo se nos ha pedido que modifiquemos un código en python para añadir dos algoritmos de búsqueda para solventar el problema del viajero y los caminos en Rumanía además de añadir los costes de cada algoritmo y su tiempo de ejecución.

Para solventar este problema la opción que decidimos tomar fue modificar la llamada search ya establecida para tener en cuenta los nodos generados, los visitados, el coste y el tiempo además de implementar dos estructuras más utilizando los heaps como base para el desarrollo de los algoritmos de Branch and Bound y Branch and Bound aplicando heurística con subestimación.

Modificaciones Realizadas al código:

- **Run.py:**

Para modificar la visualización de los resultados, se optó por redefinir la forma en que se mostraban los datos obtenidos de la función **`search`**, dado que era necesario ampliar la información proporcionada. Además, se incorporó la posibilidad de probar y comparar distintos problemas con diferentes rutas. Para ello, se implementó la función **`run_search`**, que incluye un bucle **`for`** para iterar a través de la lista de problemas. Este enfoque permite separar los resultados por cada problema y método de búsqueda, y agregar la información adicional requerida, como el número de nodos generados, los nodos visitados, el coste total del camino y la duración de la búsqueda.

Se modificaron los prints para incluir estos nuevos detalles, de modo que, para cada combinación de problema y método de búsqueda, se pudiera observar toda la información relevante de manera organizada.

- **Modificación de la Función graph_search:**

En la función graph_search, se implementaron contadores para los nodos generados y los nodos visitados, los cuales se incrementan en los puntos correspondientes del algoritmo. Además, se utilizó la librería time para medir el tiempo de ejecución de la búsqueda, registrando el inicio y el final en nanosegundos.

La estructura de la función se mantiene igual, pero se añadió un contador nodes_generated que comienza en 1 (ya que el nodo inicial se genera antes de entrar al bucle), y un contador nodes_visited que se incrementa cada vez que se visita un nodo. Durante la ejecución, se contabilizan los nodos generados al expandir el nodo actual y se actualiza la lista de nodos en la franja (fringe).

El uso de la función time.perf_counter_ns() permite obtener mediciones precisas de tiempo en nanosegundos, lo que es útil para analizar el rendimiento de la búsqueda.

- Añadido de los Nuevos Métodos de Búsqueda:

Se implementaron dos variantes del algoritmo Branch and Bound para resolver el problema, una sin heurística y otra con heurística. Ambas variantes utilizan una cola de prioridad basada en la librería heapq, que garantiza que los nodos se extraigan en el orden adecuado, de acuerdo con su coste.

1. Búsqueda Branch and Bound sin Heurística:

La función branch_and_bound_priority se encarga de invocar la función graph_search, pasándole el problema y una cola de prioridad sin heurística. Los nodos se añaden y se extraen de la cola en función de su coste de camino, sin considerar ninguna función heurística para la priorización.

2. Búsqueda Branch and Bound con Heurística:

En la variante con heurística, la función branch_and_bound_heuristicpriority sigue el mismo esquema, pero ahora utiliza una cola de prioridad que también toma en cuenta el valor heurístico de cada nodo. Los nodos se ordenan en la cola según la suma de su coste de camino y el valor heurístico, lo que permite priorizar aquellos nodos que tienen un coste total estimado más bajo. Esto ayuda a dirigir la búsqueda de forma más eficiente hacia el objetivo.

La heurística utilizada en este caso es una función que calcula la distancia en línea recta entre el estado de un nodo y el objetivo. Esta función se proporciona dentro del código y se basa en las localizaciones de los nodos en el gráfico, utilizando la distancia euclidiana para estimar la proximidad al objetivo.

Modificación de la Clase Nodo para Usar Heaps:

Para gestionar correctamente los nodos en las colas de prioridad, se modificó la clase Node, añadiendo un método de comparación basado en el coste de camino. Esto permite ordenar los nodos en la estructura de datos heap, asegurando que siempre se procese el nodo con el menor coste y optimizando el rendimiento del algoritmo.