UNIVERSIDAD AUTÓNOMA DE MADRID

ALGORITMOS Y ESTRUCTURAS DE DATOS AVANZADAS

Práctica 2: Conjuntos Disjuntos y Algoritmo de Kruskal. El Problema del Viajante

Sergio Hidalgo Gamborino
Miguel Ibáñez González

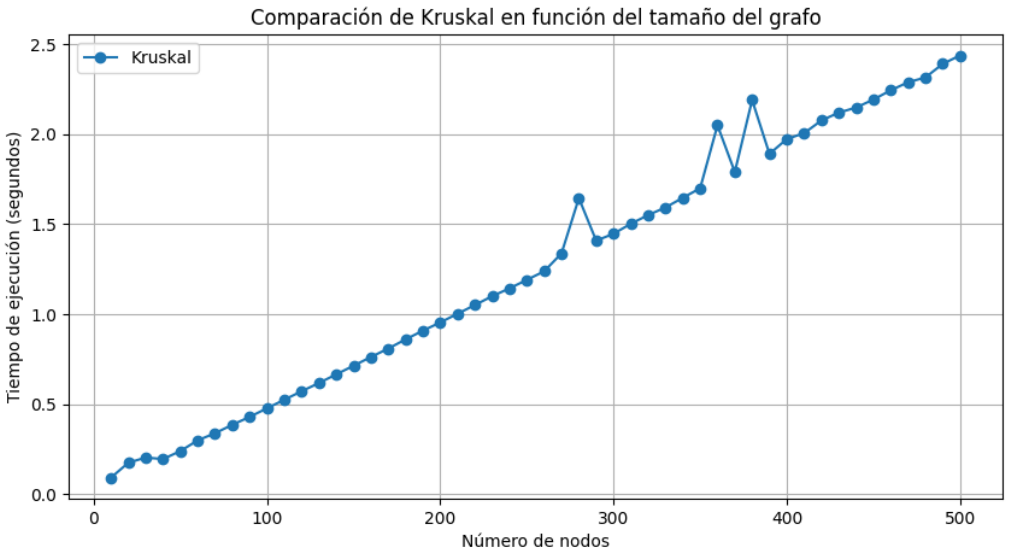
Grupo 1292
Pareja 05

II-C. Cuestiones sobre Kruskal Contestar razonadamente a las siguientes cuestiones, incluyendo gráficas si fuera preciso.

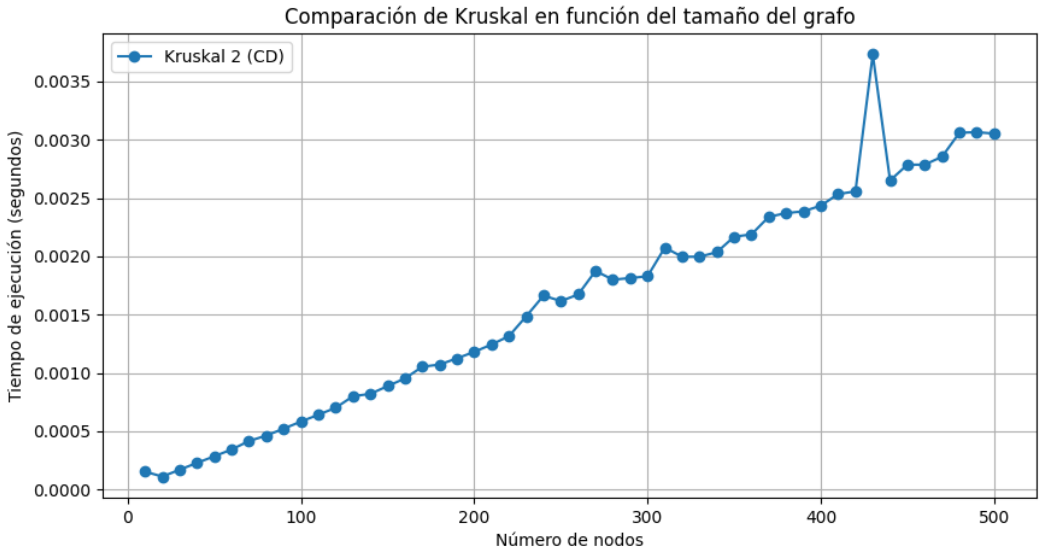
1. Discutir la aportación al coste teórico del algoritmo de Kruskal tanto de la gestión de la cola de prioridad como la del conjunto disjunto. Intentar llegar a la determinación individual de cada aportación. Contrastar la discusión anterior con las gráficas a elaborar mediante las funciones desarrolladas en la práctica.

> init_cd(): La creación del conjunto disjunto tiene un coste de O(|V|)
> create_pq(): La creación de la cola de prioridad tiene un coste de O(|E| log |V|)
> queue.get(): Obtener dentro del bucle el elemento de la cola d prioridad tiene un coste de O(|E| log |V|)
> La condicion dentro del bucle tiene un coste de O(|V|)
> Respecto a los finds, el coste es de O(|E| lg∗ |V|) que esencialmente es O(|E|)
>
> Por tanto, el costo teórico total del algoritmo de Kruskal es aproximadamente O(|E| lg |V|).

Kruskal



Kruskal 2 (Conjuntos disjuntos)



El tiempo necesitado para los CD es mucho menor que el necesario para toda la función completa. Se asemeja bastante a O(N log N)

2. El algoritmo de Kruskal puede detectar que ya ha obtenido un árbol abarcador mínimo sin tener que procesar toda la cola de prioridad. Sin embargo, la misma debería vaciarse, para lo que podemos simplemente que nuestra función Kruskal ´ siga procesando la cola de prioridad hasta vaciarla (no se añadirán nuevas ramas al AAM) o bien simplemente dejar de procesar el CD y simplemente vaciar la cola mediante get .

¿Cual de las dos alternativas te parece más ventajosa computacionalmente? Argumenta tu respuesta analizando los costes de ejecución de ambas opciones.

Procesar hasta vaciar:

> Ventajas: Puede ser útil si hay operaciones adicionales que deben realizarse durante el proceso de extracción de elementos de la cola de prioridad. Si, por ejemplo, se necesita realizar alguna acción específica con los elementos antes de dejar de procesarlos, esta opción podría ser la adecuada.
>
> Desventajas: Puede llevar a un mayor tiempo de ejecución, especialmente si la cola de prioridad es grande y la operación de extracción es costosa.

Vaciar mediante get:

> Ventajas: Puede ser más eficiente pues se evita procesar elementos innecesarios. Esto puede ser útil si el algoritmo solo requiere los elementos sin necesidad de realizar acciones específicas con ellos.
>
> Desventajas: Si hay operaciones adicionales que deben realizarse, se deben llevar a cabo por separado después de vaciar la cola.

III-B. Cuestiones sobre la solución greedy de TSP
Contestar razonadamente a las siguientes cuestiones.

1. Estimar razonadamente en función del número de nodos del grafo el coste codicioso de resolver el TSP. ¿Cuál sería el coste de aplicar la función exhaustive_tsp ? ¿Y el de aplicar la función repeated_greedy_tsp ?

El coste de aplicar la función exhaustive_tsp es exponencial en función del número de nodos del grafo. La complejidad temporal es del orden de O(n!), lo que significa que el tiempo de ejecución aumenta rápidamente con el número de nodos.

greedy_tsp: El bucle while se ejecuta n veces, donde n es el número de nodos en el grafo representado por dist_m.
Dentro del bucle, hay otro bucle for que se ejecuta n veces. La complejidad de greedy_tsp es aproximadamente $O(n^2)$.

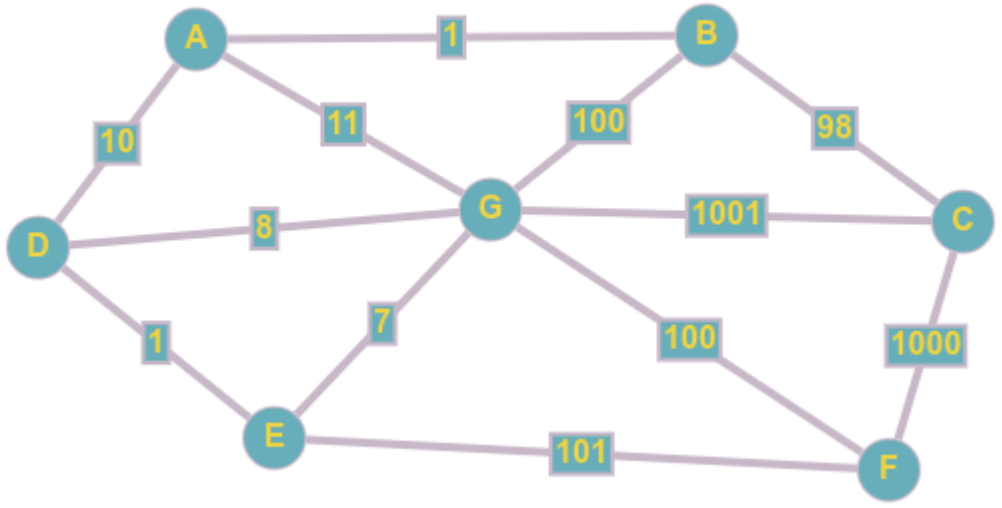
len_circuit: El bucle for se ejecuta n veces, donde n es la longitud del circuito. Las operaciones dentro del bucle son de tiempo constante. La complejidad de len_circuit es O(n).

repeated_greedy_tsp: Hay un bucle for que se ejecuta n veces, donde n es el número de nodos en el grafo representado por dist_m, La complejidad es O(n). Dentro de dicho bucle, se llama a greedy_tsp dos veces y len_circuit una vez, por lo que la complejidad es aproximadamente $O(n^2)$.

En el peor caso, se ejecuta greedy_tsp y len_circuit n veces cada una.
Por lo tanto, la complejidad total de repeated_greedy_tsp es aproximadamente $O(n^3)$.

2. A partir del código desarrollado en la práctica, encontrar algún ejemplo de grafo para el que la solución greedy del problema TSP no sea óptima.

Cualquier grafo cuyos caminos más cercanos de menor peso conduzcan a una solución que no sea óptima.



Camino según algoritmo: A-B-C-F-G-E-D coste: 1217
Un camino que es más óptimo que ese: D-E-F-G-C-B coste: 410