12/11/2023

Algoritmia y Estructuras de Datos Avanzadas

Práctica - 2

G.1281 - P3

Joaquín Abad Díaz - Carlos García Santa

II-C. Cuestiones sobre Kruskal

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.

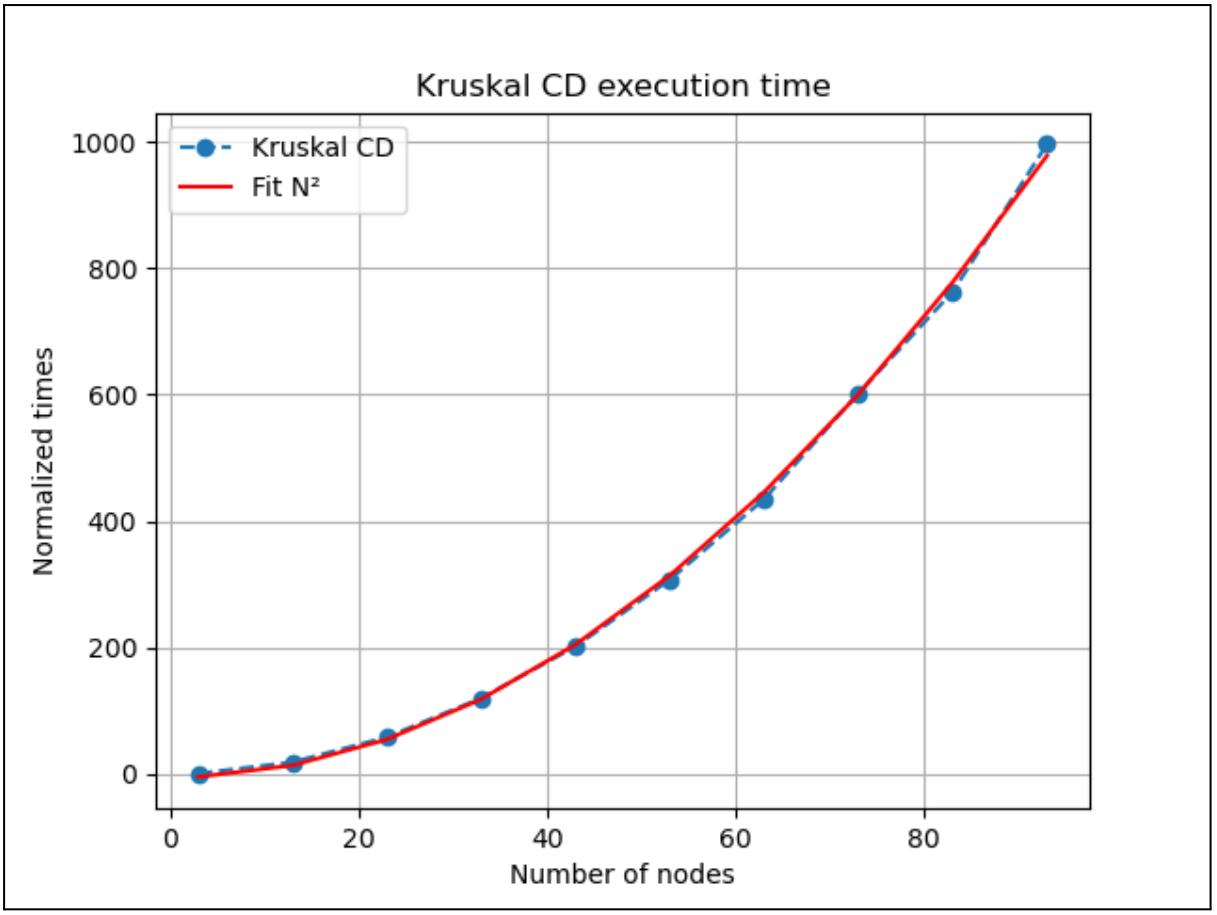
Contrastar la discusión anterior con las gráficas a elaborar mediante las funciones desarrolladas en la práctica.

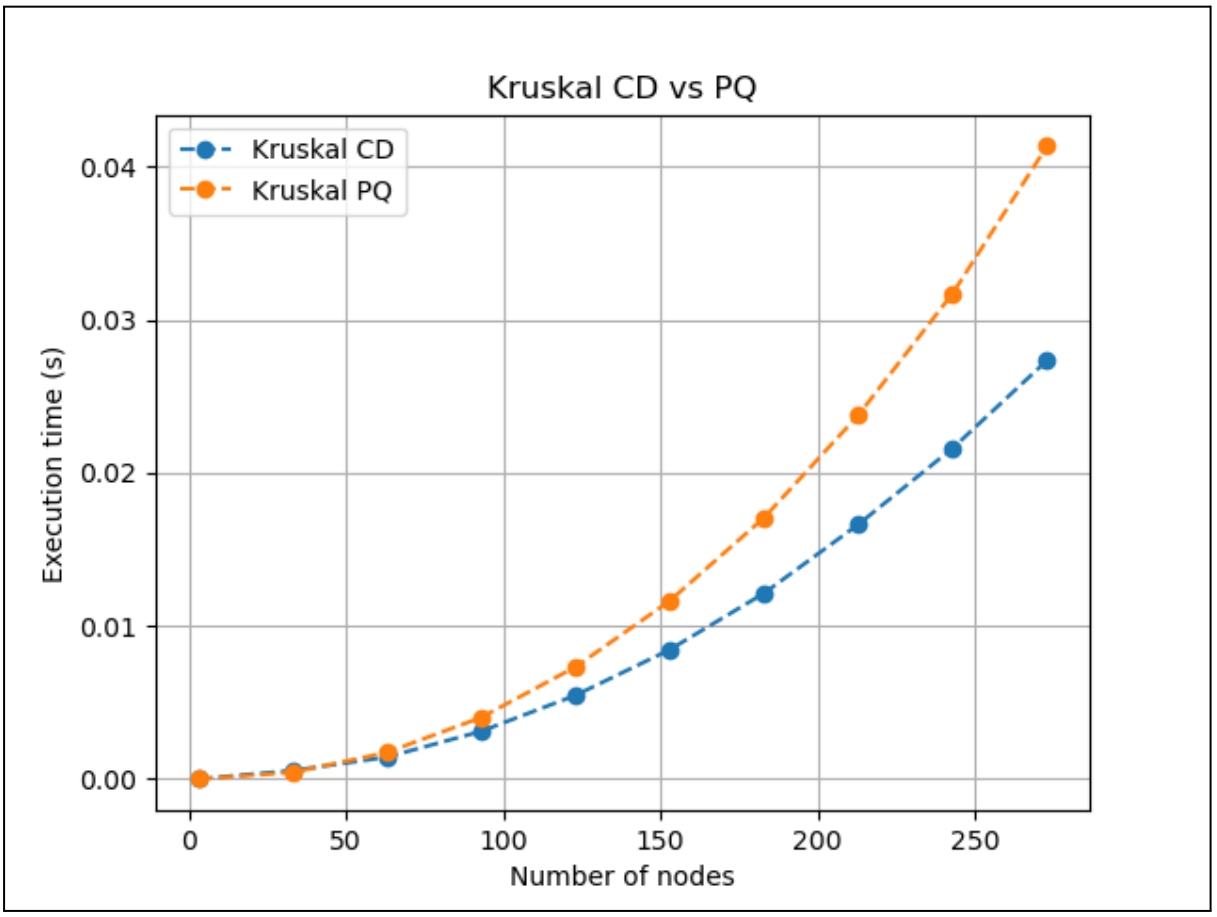
Las operaciones de extracción sobre una cola de prioridad tienen una complejidad de O(log E) donde E es el número de aristas introducidas en la cola de prioridad, como esta operación de extracción se realiza E veces la complejidad es O(E log E), pero dado que en un grafo completamente conectado $E = V^2$ (explicación en [1]) siendo V el número de nodos, la complejidad sería $O(V^2 \log V^2) = O(2V^2 \log V) \approx O(V^2 \log V)$.

[1] Cada vértice tiene aristas a todos los demás vértices menos a sí mismo, lo que resulta en $V-1$ aristas por vértice. Si multiplicamos esto por el número total de vértices, obtenemos $V \times (V-1) = V^2 - V$. Sin embargo, esto contaría cada arista dos veces, ya que una arista entre dos vértices A y B se contaría una vez para A y otra vez para B, por lo tanto, para obtener el número correcto de aristas únicas, debemos dividir este producto por 2.

Sin embargo, cuando V es grande (que es lo que necesitamos), el término V se vuelve insignificante en comparación con V^2, y la división por 2 puede pasarse por alto en una estimación aproximada, llevando a la idea de que E es del orden de V^2.

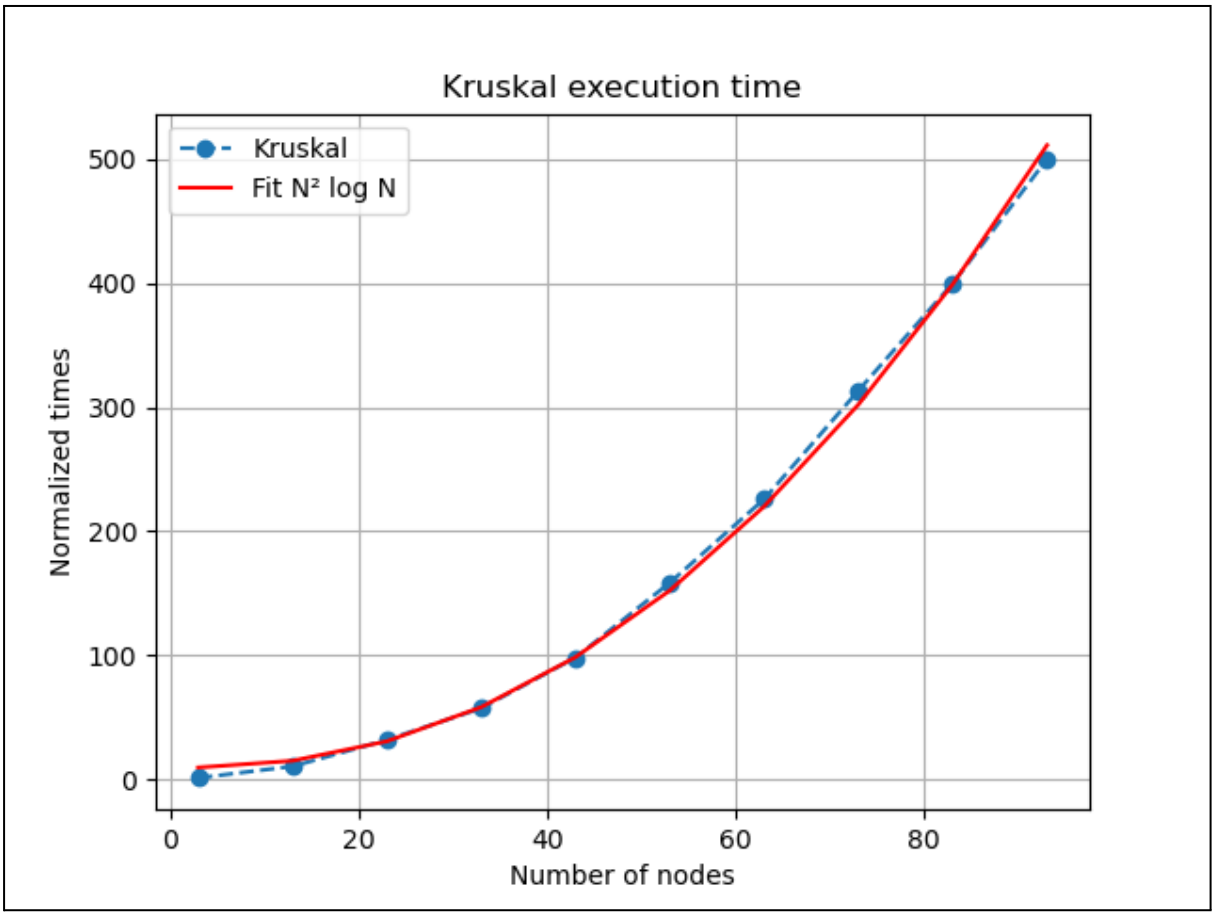
Las gestiones del conjunto disjunto, que corresponden a las funciones union y find, tienen respectivamente una complejidad de O(V) donde V es el número de elementos del conjunto disjunto que es igual al número de vértices del grafo, esto debe a que union tiene una complejidad constante O(1) que acumulada en el bucle es O(V); y O(E log* V) lo que esencialmente equivale a O(E), por la explicación anterior[1] esto es igual a $O(V^2)$. Resumiendo tendríamos $O(V^2) + O(V)$ equivalente a $O(V^2)$.


Kruskal CD execution time
Kruskal CD
Fit N²
Normalized times
Number of nodes
0
200
400
600
800
1000
0
20
40
60
80

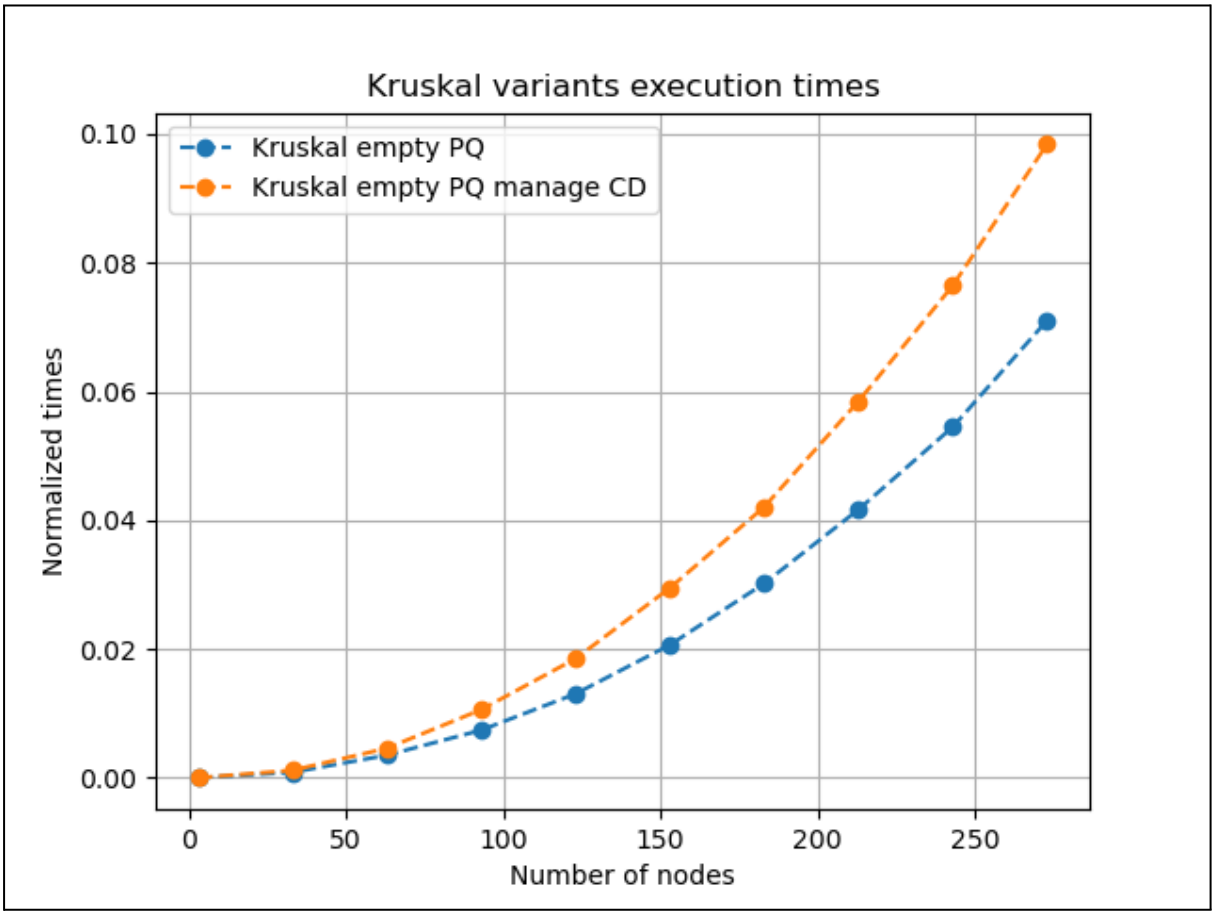

Kruskal CD vs PQ
Kruskal CD
Kruskal PQ
Execution time (s)
Number of nodes
0.00
0.01
0.02
0.03
0.04
0
50
100
150
200
250

Parece que los resultados empíricos coinciden con las afirmaciones teóricas, por tanto el aporte a kruskal del conjunto disjunto sería N^2 mientras que la de la cola de prioridad sería $N^2 \log N$, por lo tanto Kruskal estaría dominado por esta última complejidad de la PQ ya que $N^2 \log N > N^2$.

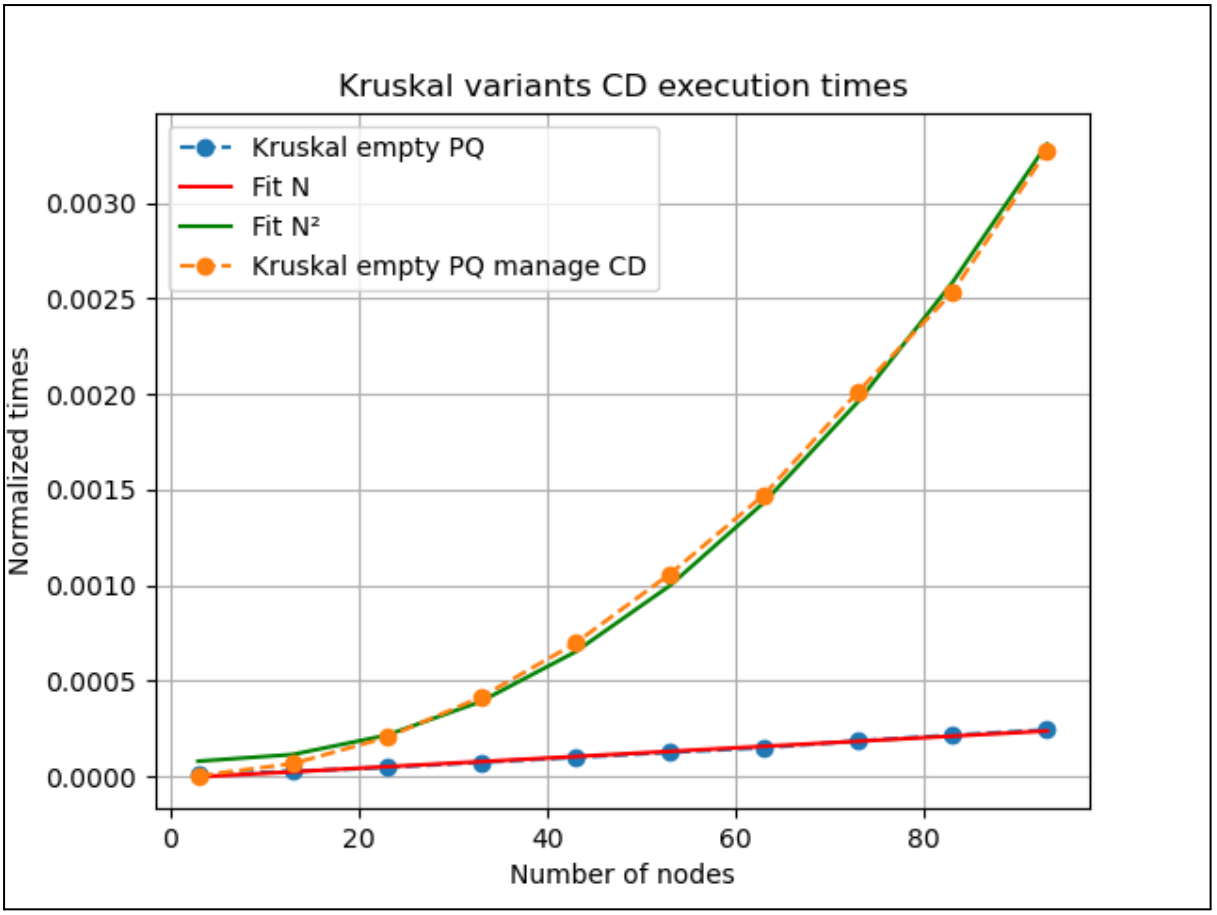
Por otra parte Kruskal tiene una complejidad O(E log V) donde E es el número de aristas y V es el número de vértices o nodos, como se ha desarrollado anteriormente en grafos completamente conectados $E = V^2$ (V número de vértices o nodos), por tanto tiene una complejidad $O(V^2 \log V)$.



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.



Según los tiempos de ejecución claramente la opción de dejar de procesar el conjunto disjunto una vez encontrado el AAM es mejor, esto tiene sentido ya que si evitamos operar el conjunto disjunto en vez de procesar todos los vértices pasamos de una complejidad $O(V^2)$ a una complejidad $O(V)$, mientras que en la otra opción se sigue teniendo $O(V^2)$.

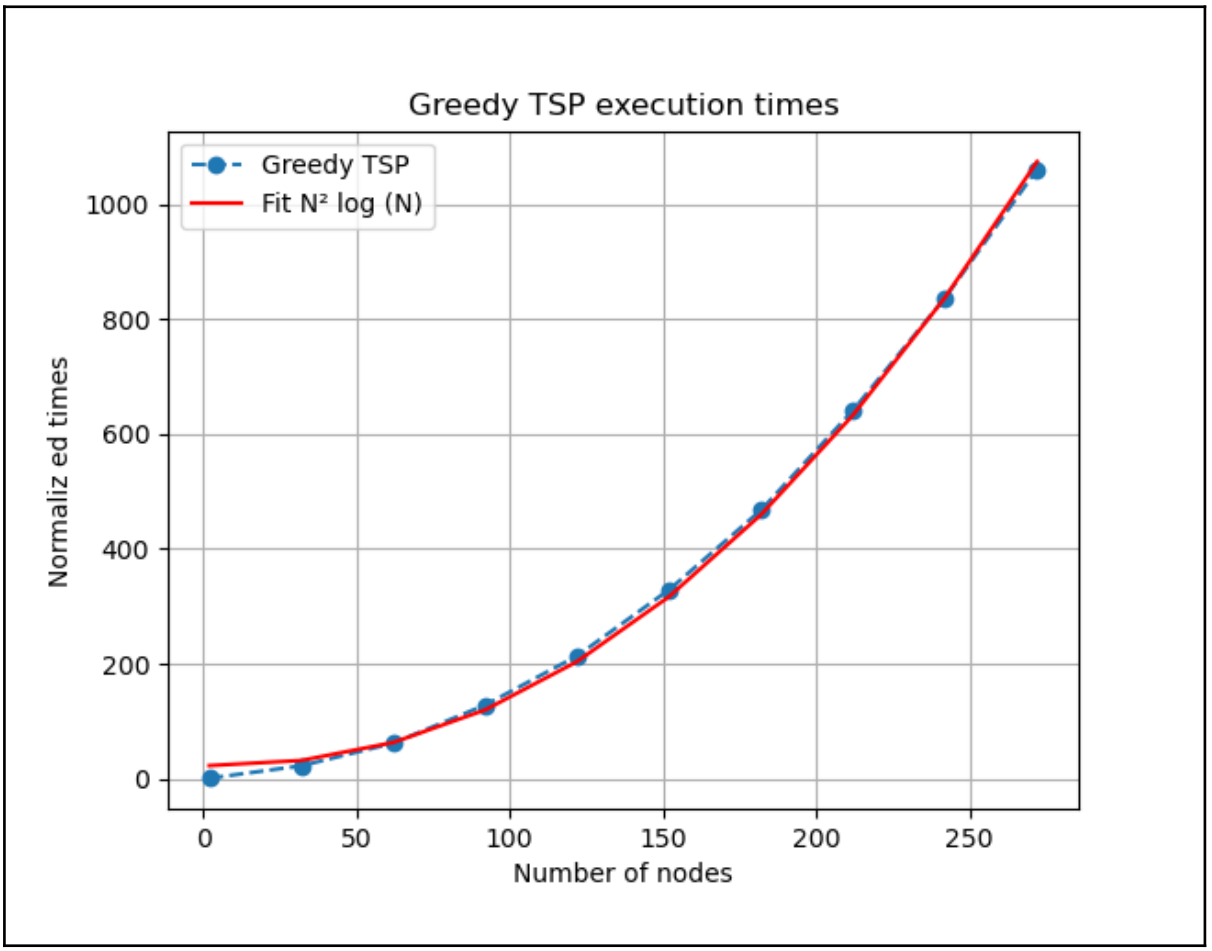


III-B. Cuestiones sobre la solución greedy de TSP.

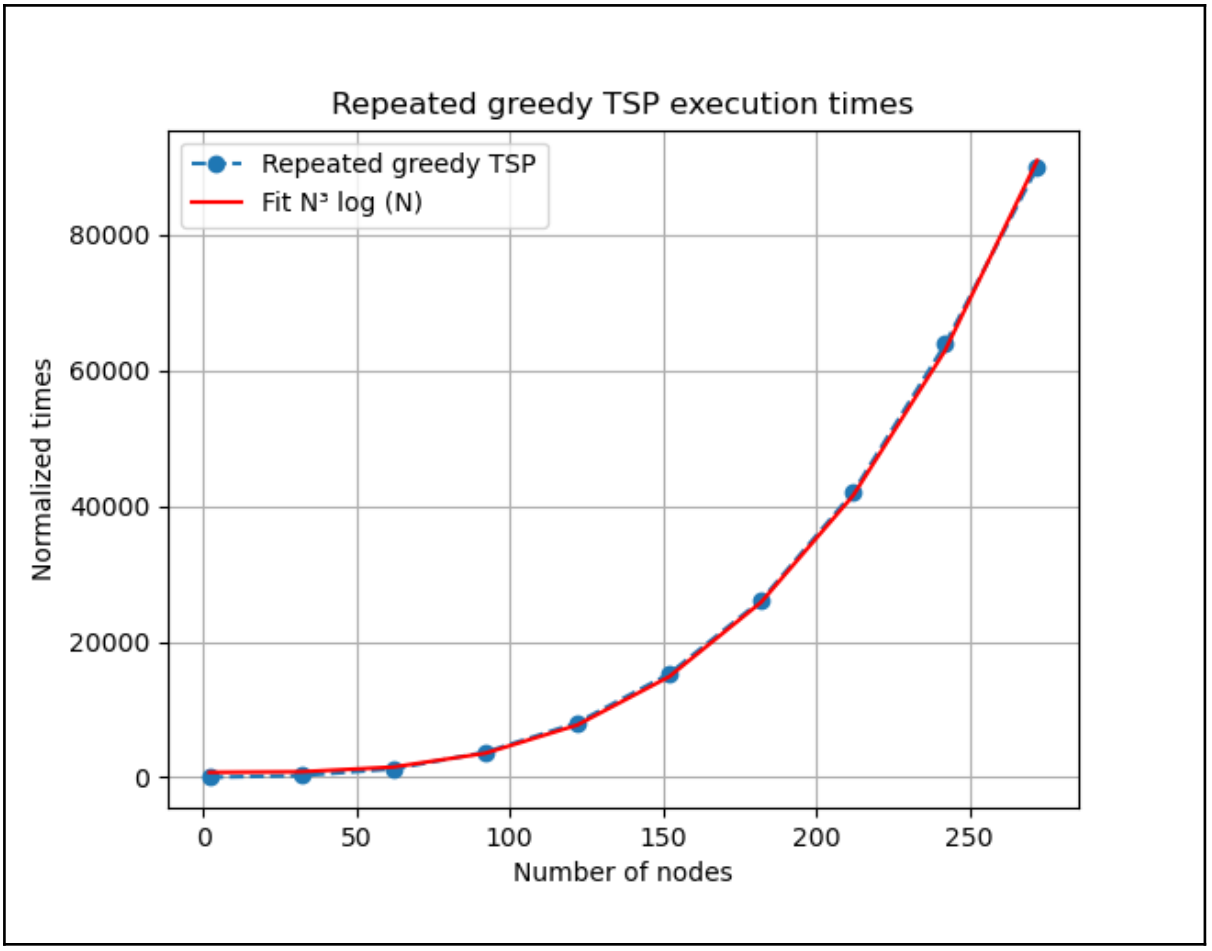
1. Estimar razonadamente en función del número de nodos del grafo el coste codicioso de resolver el TSP. ¿Cuál sería el coste de aplicar la función exhaustive_tsp ? ¿Y el de aplicar la función repeated_greedy_tsp?

Para estimar el coste de resolver el TSP con un enfoque codicioso podemos analizar la función y estimar la complejidad de la misma. Primero la función obtiene el número de ciudades a partir de la matriz de distancias lo cual es una operación de complejidad constante O(1), luego inicializa el circuito con la primera ciudad lo cual también tiene una complejidad constante O(1). Posteriormente tenemos un bucle que se ejecuta mientras la longitud del circuito sea menor que el número de ciudades, por ello, el bucle se ejecutará N veces, donde N es el número de ciudades; dentro de este bucle se utiliza np.argsort para ordenar las ciudades según su distancia a la ciudad actual, esta función tiene una complejidad O(N log N) en promedio, ya que se basa en algoritmos de ordenación como quicksort o mergesort; después de la ordenación se realiza una búsqueda lineal a través de las ciudades ordenadas para encontrar la primera que no ha sido visitada aún, esta operación, tiene una complejidad de O(N). Con todo esto, dado que el bucle se ejecuta N veces y dentro de cada iteración del

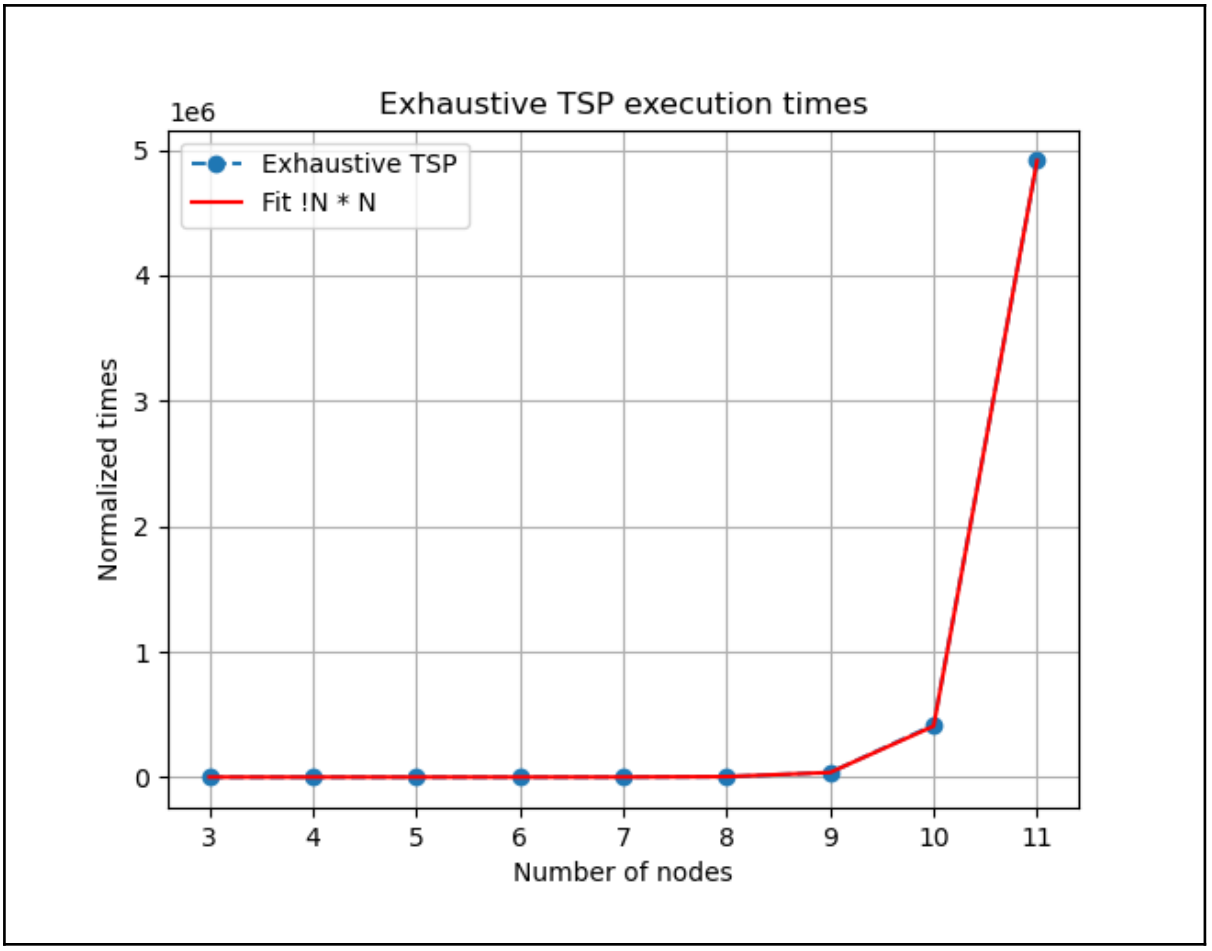
bucle se realiza un ordenamiento O(N log N) y una búsqueda lineal O(N), la complejidad temporal total de la función es O(N^2 log N).



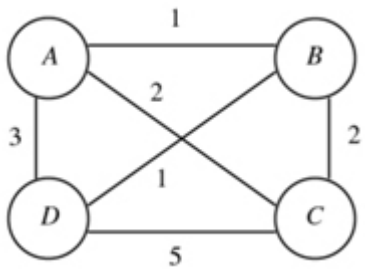
En el caso de repeated_greedy_tsp analizar la complejidad es sencillo ya que consiste en repetir el algoritmo codicioso anterior para cada nodo del grafo, por lo tanto, como tenemos que realizar N llamadas a la función codiciosa con complejidad O(N^2 log N) para cada nodo o ciudad la complejidad es simplemente O(N^3 log N).



La función exhaustiva itera sobre todas las permutaciones posibles de las ciudades, habiendo N! permutaciones posibles para N ciudades, para cada permutación, calcula la longitud del circuito sumando las distancias entre ciudades consecutivas, este cálculo es O(N) para cada permutación, ya que se necesita sumar las distancias para N pares de ciudades. Por lo tanto, la complejidad temporal total es O(N!×N), que es extremadamente alta incluso para valores moderadamente pequeños de N, por ello se ha graficado con un bajo número de nodos.



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.



Desde A, el circuito codicioso es ABDCA con una longitud de 9, mientras que ACBDA tiene una longitud de 8.