MEMORIA DE ALGORITMIA PRACTICA 2

Hugo Torres y Luis Rodríguez.

1. Sin darnos cuenta, en nuestro algoritmo de encontrar CCs podemos pasar listas con ramas repetidas o donde los vértices coinciden con los de una que ya está aunque en orden inverso. ¿Afectará esto al resultado del algoritmo? ¿Por qué?

   No afecta, ya que el algoritmo coge al representante y no repite las conexiones, aunque sean inversas o se repitan.

2. Argumentar que nuestro algoritmo de encontrar componentes conexas es correcto, esto es, que a su final en los distintos subconjuntos disjuntos se encuentran los vértices de las distintas componentes del grafo dado.

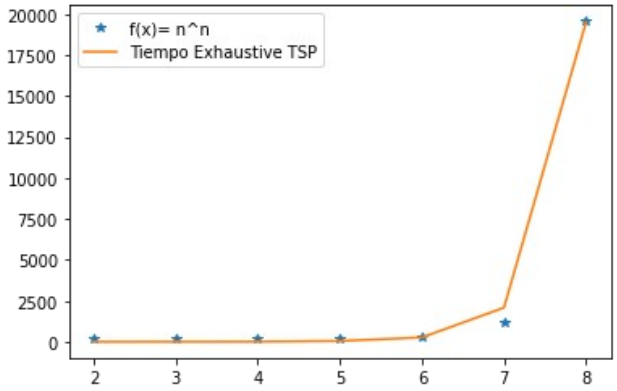
   En nuestro algoritmo se crea un diccionario usando como clave al representante de cada subconjunto usando el find, y de cada clave tiene como valores los vértices a los que está conectado. Despues llama a la función unión para que una los subconjuntos disjuntos, y por último se llama a la función de cd_2_dict para que haga un diccionario con los datos organizados anteriormente y lo devuelve la función ccs.

3. El tamaño de un grafo no dirigido viene determinado por el número n de nodos y la longitud de la lista l de ramas. Estimar razonadamente en función de ambos el coste del algoritmo de encontrar las componentes conexas mediante conjuntos disjuntos.

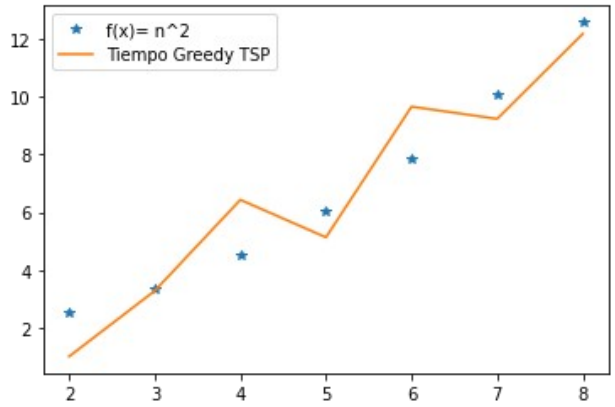
   La función css se compone de un bucle for en el rango de 1 a len(List), donde dentro se ejecutan siempre otros 2 finds. Find tiene un coste de O(log(n)), y aplicando lo aprendido en clase, el coste total del algoritmo css en función de L y N sería de O(L*Log(N)).

SEGUNDAS 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 ?



Esta es la gráfica del exhaustive que la hemos ajustado a N^N, viendo que tiene un coste mucho mayor en los casos más grandes de matrices.



Este es el greedy, los tiempos se ven que son mucho menores en los casos grandes de las funciones pero iguales casi en los casos pequeños. Esta función la hemos ajustado a N^2.

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

Un ejemplo de estos grafos sería:

```
[0    3   48   11]
[11   0   26   21]
[ 5   8    0   21]
[20  17  43    0 ]
```

Longitud usando la función greedy – 72
Longitud usando la función exhaustive - 59

El circuito que seguiría el greedy sería [0, 1, 3, 2, 0], mientras que el exhaustive seguiría el [0, 3, 1, 2, 0].