AEDATA Lab 2

# Disjoint sets and connected components; the Traveling Salesman Problem

2022-23

Algoritmo y estructura de datos avanzadas, Lab 2

Daniel Varela Sánchez & Guillermo Martín-Coello Juárez

## I-C Questions about Disjoint sets and Connected Components

**1. We didn't realize it, but in our algorithm for CCs we are running through a list with repeated edges and with edges in which the vertices are repeated in different order. Will this affect the result of the algorithm? Why?**

Repeated edges and edges in which the vertices are repeated in different order will not affect the algorithm. This is due to the algorithm only considering when the nodes are on different subsets. If an edge has already been considered, when it is repeated the two vertices will be already in the same set, so the algorithm will not operate.

**2. Argue the correctness of the CCs algorithm, that is, argue that at the end of the algorithm the sets that we obtain do indeed represent the connected components of the graph.**

The algorithm will indeed return subsets representing the connected nodes of the CCs, but will fail at representing those connections. This means the edges of the graph will not be represented, because the disjoint set works with a tree, not a graph.

**3. The size of a graph is determined by the number n of nodes and the length l of the list of edges. Estimate, as a function of both parameters, the cost of the connected components algorithm implemented using disjoint sets. Justify your answer.**

The ccs algorithm contains a loop that gets executed L times. Inside that loop we find a union of two finds. We know that union has a cost of O(1) and find has a cost of O(log(n)). So we know that we execute the union of the two finds L times hence we obtain the following time complexity for the ccs algorithm:

Ccs complexity = O( L * log(n))

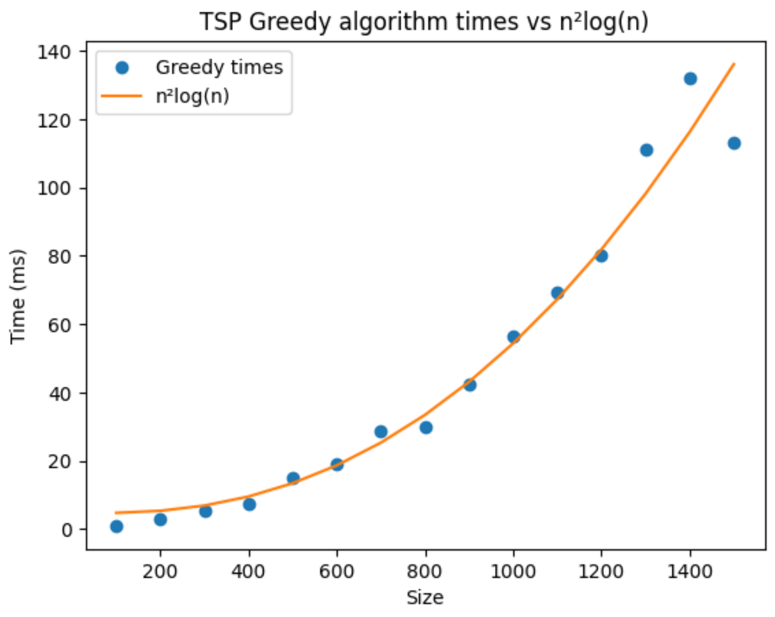
## II-B Questions about the Traveling Salesman Problem

**1. Estimate the cost of the greedy algorithm as a function of the number of cities n; justifying your question and show your reasoning. What would be the cost of applying exhaustive_greedy_tsp? How about that of repeated_greedy_tsp?**

The greedy algorithm will operate for each city in the graph, looking at each of the possible next steps to follow to choose the best one possible. This will result in a cost of $n^2$. The remaining of the code will do for each of them a search with cost log(n), ending up with a final cost of:

Greedy TSP Complexity $= O(n^2 * \log(n))$

To prove our hypothesis we plotted the times spent by the algorithm and compared them to the cost, resulting in the following graph:

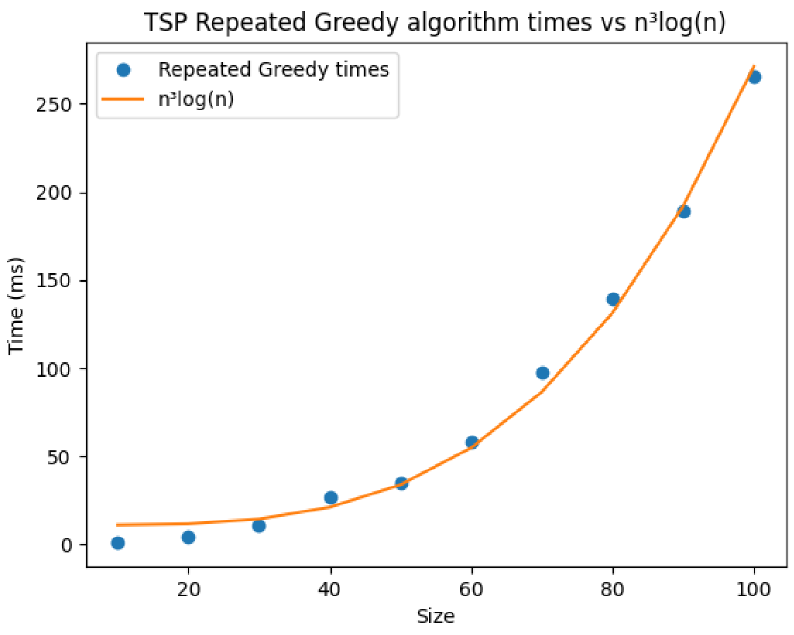


As we can observe, the graph shows a clear direct relation between the expected cost and the real time.

The cost of the repeated greedy algorithm, as it does the greedy tsp algorithm n times, is:

Repeated Greedy TSP Complexity $= O(n^3 * \log(n))$

This cost is proved in the following graph:

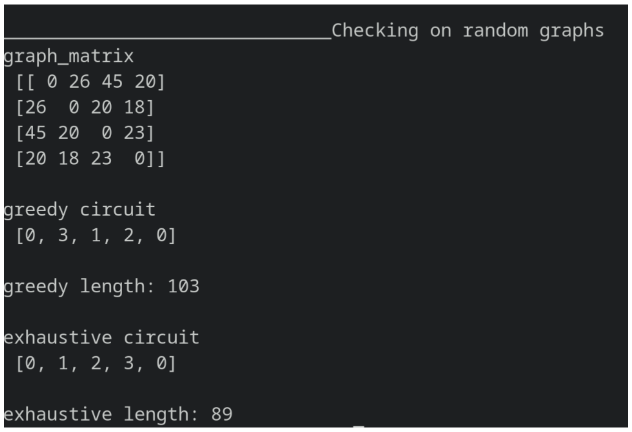


Finally, the cost of the exhaustive algorithm, as it uses a new way to achieve a faster execution with the same result, will cost:

Exhaustive Greedy TSP Complexity $= O( n! * n^2 * \log(n))$

**2. Using the code developed in this exercise, find a graph for which the greedy solution is not optimal.**

By executing the test several times we finally obtain an example of a case where the greedy solution is not the most optimal (we can say so because the exhaustive algorithm obtains a more efficient circuit). This happens because while the exhaustive algorithm tries all the possible circuits to find the most efficient, the greedy one in hopes of reducing the computation time, it always chooses the most efficient step in each case hence omitting possible better moves two steps ahead for example.



```
______________________________Checking on random graphs
graph_matrix
 [[ 0 26 45 20]
 [26  0 20 18]
 [45 20  0 23]
 [20 18 23  0]]

greedy circuit
 [0, 3, 1, 2, 0]

greedy length: 103

exhaustive circuit
 [0, 1, 2, 3, 0]

exhaustive length: 89
```

In this case for example the greedy algorithm chooses the third node in the first step because it is the one with less cost but the exhaustive algorithm chooses the first one that though it is less efficient for one step, it ends up with a more efficient circuit than the greedy algorithm.