Challenge: The Traveling Salesman Problem

In this lesson, you will solve a coding challenge on the famous traveling salesman problem.

We'll cover the following

- Problem statement
- Input
- Output
- Coding challenge

Problem statement

You are given a map that has n cities on it. All of the cities are connected directly to each other via distinct routes of variable lengths. You, being a salesman, have to travel to each of these cities on a business trip. But your company wants you to be very careful with the travel expenses and wants you to spend the least amount possible. Expenditure is a function of the distance traveled, so you want to minimize the total distance traveled. Find the length of the shortest path to travel all the cities.

Input

You will be given a 2-d matrix, `distances`, of size $n \times n$ denoting the distance between all the cities. To check the distance from i^{th} city to j^{th} city, you can simply do `distances[i][j]`. The matrix does not necessarily have to be symmetrical, i.e., `distances[i][j]` $\neq$ `distances[j][i]`. You can start from any city to travel around all the other cities until you come back to the same city you started traveling from. You should not visit any city twice other than the first city.

```
distances = [
      [0, 10, 20],
      [12, 0, 10],
      [19, 11, 0]
]
```

Output

Your algorithm should return the minimum distance traveled to traverse every city; each city other than the first. The chosen city should only be traveled to once.

```
TSP(distances) = 39
```

Let's look at the following visualization for a better understanding.

Look at this map of three cities, we have multiple ways to visit all the cities once starting from city 0, and then returning back to city 0 again

1 of 3

Path highlighted in blue is one, total distance of this path is 43; path = 0 -> 2 -> 1 -> 0

However, optimal path would be this one because its cost is 39; path = 0 -> 1 -> 2 -> 0

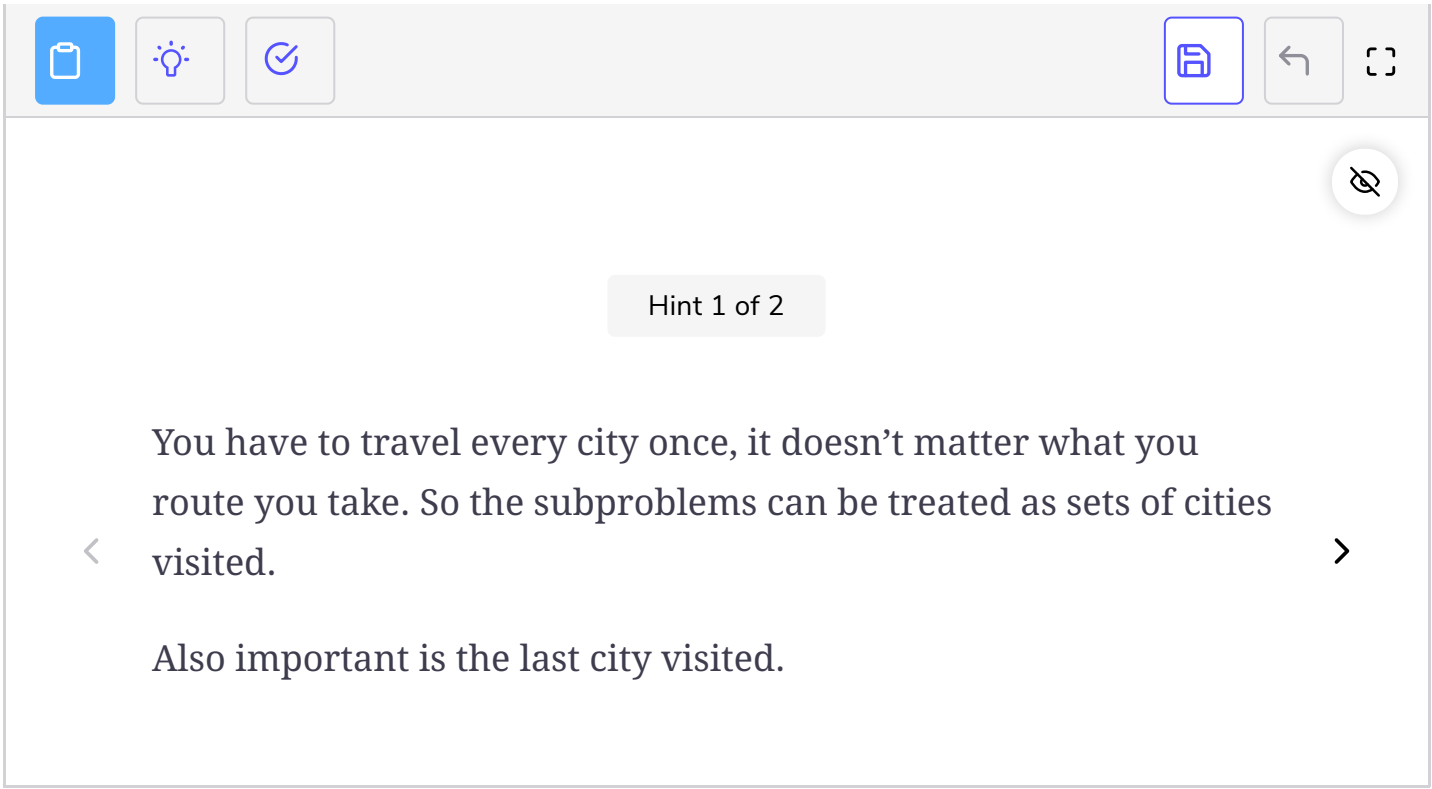
Coding challenge

This is a difficult problem, so let's start with a very basic example. Think about how many paths exist and how you can find the most optimal one amongst them. Once you have written a working algorithm, optimize it using dynamic programming.

```
import numpy as np

def TSP(distances):
  # write your code here
  return 0

stressTesting = True
```



In the next lesson, we will review some solutions to this problem.