Day 21: *Travelling Salesman Problem*

As the plane touched ground in Vancouver, the final flight in Dot's around-the-world journey came to a close. There would be no more airports, passport queues, or boarding passes. Dot was sad to see their trip end, but they couldn't help but feel a bit of relief. All that travelling had been exhausting! Dot was excited to get back home and stay grounded, at least for a little while. Dot opened up their passport and flipped through the pages on the way home. Previously bare, the pages were now filled with customs stamps from all over the world – Spain and Italy, Japan and Thailand, the UAE and Australia. Dot could hardly believe how many kilometres their trip had spanned across the globe: thousands and thousands!

Dot unlocked the front door of their apartment and rushed in, abandoning their luggage in the entryway as they ran to see their snake, Python. The serpent wriggled out from under its cave to greet Dot, curiously flicking its tongue. "Did you miss me, Python?" Dot asked, and the snake cocked its head in incomprehension. Dot fell into a comfy chair and let out an exhale. They sure were tired and jetlagged and were bound to sleep well that night. Though it was fun and exciting, the trip hadn't been as efficient as it could've been. Dot began to reflect on their travels and think about how they could've saved time by moving more efficiently. Can you help Dot one last time by computing the most optimal route they could've taken during their travels?

Tutorial

Welcome to the end of the 21-Day Data Challenge! Congratulations, you are on the final stretch!

In today's challenge, we'll take a look at the Travelling Salesman Problem, one of the most popularly used optimization problems. We will compute the most optimal route we could have taken to visit every stop during our trip.

```
import six
import sys
sys.modules['sklearn.externals.six'] = six
import mlrose
import numpy as np
```

Using the distance approach, the fitness function object can be initialized as follows:

```
# Create list of distances between pairs of cities
dist_list = [(0, 1, 3.1623), (0, 2, 4.1231), (0, 3, 5.8310), (0, 4, 4.2426), \
             (0, 5, 5.3852), (0, 6, 4.0000), (0, 7, 2.2361), (1, 2, 1.0000), \
             (1, 3, 2.8284), (1, 4, 2.0000), (1, 5, 4.1231), (1, 6, 4.2426), \
             (1, 7, 2.2361), (2, 3, 2.2361), (2, 4, 2.2361), (2, 5, 4.4721), \
             (2, 6, 5.0000), (2, 7, 3.1623), (3, 4, 2.0000), (3, 5, 3.6056), \
             (3, 6, 5.0990), (3, 7, 4.1231), (4, 5, 2.2361), (4, 6, 3.1623), \
             (4, 7, 2.2361), (5, 6, 2.2361), (5, 7, 3.1623), (6, 7, 2.2361)]

# Initialize fitness function object using dist_list
fitness_dists = mlrose.TravellingSales(distances = dist_list)
```

In our example, we want to solve a minimization problem of length 8. If we use the **fitness_coords** fitness function defined above, we can define an optimization problem object as follows:

```
problem_fit = mlrose.TSPOpt(length = 8, fitness_fn = fitness_dists,
                            maximize=False)
```

We will use the **genetic_alg** method to find the most optimal solution.

```
# Solve problem using the genetic algorithm
best_state, best_fitness = mlrose.genetic_alg(problem_fit, random_state = 2)

print('The best state found is: ', best_state)

print('The fitness at the best state is: ', best_fitness)
```

This was a demo example of how mlrose can solve a simple Travelling Salesman Problem. The tutorial was a shorter version of the one that can be found directly in the mlrose documentation. For a more detailed tutorial, you can visit the documentation here.

Challenge

In [2]:
```
import six
import sys
sys.modules['sklearn.externals.six'] = six
import mlrose
import numpy as np
```

In [3]:
```
city_mapper = {
    0:'Vancouver',
    1:'Toronto',
    2:'Munich',
    3:'London',
    4:'Barcelona',
    5:'Paris',
    6:'Florence',
    7:'Dubai',
    8:'Perth',
    9:'Melbourne',
    10:'New Zealand',
    11:'India',
    12:'Nepal',
    13:'Japan',
    14:'Thailand',
    15:'Hawaii',
    16:'Seattle'
}
```

In [4]:
```
# ditance between all cities in hours.
dist_list = [
    (0,1,4.0000),(0,2,10.0000),(0,3,9.3330),(0,4,13.0000),(0,5,9.7500),(0,6,12.5000),
    (0,7,17.5000),(0,8,22.0000),(0,9,18.7500),(0,10,16.7500),(0,11,24.0000),(0,12,22.00
    (0,14,25.0000),(0,15,6.0000),(0,16,0.5000),(1,2,8.7500),(1,3,6.7500),(1,4,10.0000),
    (1,5,7.2500),(1,6,10.0000),(1,7,12.7500),(1,8,29.0000),(1,9,25.0000),(1,10,24.0000)
    (1,12,19.0000),(1,13,16.0000),(1,14,22.0000),(1,15,10.0000),(1,16,5.0000),(2,3,2.00
```

```
    (2,5,1.5000),(2,6,1.2500),(2,7,6.0000),(2,8,18.5000),(2,9,21.5000),(2,10,30.0000),(
    (2,12,12.2500),(2,13,14.0000),(2,14,13.5000),(2,15,18.5000),(2,16,13.0000),(3,4,2.0
    (3,6,2.0000),(3,7,6.7500),(3,8,18.7500),(3,9,21.5000),(3,10,27.0000),(3,11,13.0000)
    (3,13,12.0000),(3,14,12.0000),(3,15,17.5000),(3,16,10.0000),(4,5,2.3000),(4,6,1.750
    (4,7,6.5000),(4,8,19.7500),(4,9,21.0000),(4,10,31.0000),(4,11,17.0000),(4,12,12.250
    (4,14,14.0000),(4,15,21.5000),(4,16,13.5000),(5,6,1.7500),(5,7,6.6600),(5,8,18.5000
    (5,9,21.7500),(5,10,30.0000),(5,11,16.0000),(5,12,12.0000),(5,13,12.0000),(5,14,13.
    (5,16,12.7500),(6,7,10.7500),(6,8,25.0000),(6,9,25.0000),(6,10,34.0000),(6,11,17.00
    (6,12,15.5000),(6,13,15.7500),(6,14,15.0000),(6,15,21.7500),(6,16,14.0000),(7,8,10.
    (7,9,13.4500),(7,10,21.7500),(7,11,7.8000),(7,12,3.7500),(7,13,9.5000),(7,14,6.0000
    (7,16,14.7500),(8,9,3.5000),(8,10,19.0000),(8,11,22.0000),(8,12,12.7500),(8,13,13.7
    (8,15,16.3300),(8,16,22.5000),(9,10,6.0000),(9,11,26.0000),(9,12,16.0000),(9,13,10.
    (9,15,10.2500),(9,16,19.5000),(10,11,33.0000),(10,12,20.4400),(10,13,15.0000),(10,1
    (10,16,19.5000),(11,12,5.0000),(11,13,17.7500),(11,14,9.2500),(11,15,36.0000),(11,1
    (12,14,10.3300),(12,15,24.0000),(12,16,24.0000),(13,14,10.5000),(13,15,6.7500),(13,
    (14,15,27.0000),(14,16,25.0000),(15,16,5.5000)
]
```

1. Use mlrose to find the most optimal route to visit all the places Dot visited throughout the challenge.

Note:

- In optimization algorithms, we can get different solutions when we choose different initial points. Therefore, it's important to use the following parameters in your **genetic_alg** function so your solution converges to the same result as ours:
  - `random_state` = 2
  - `mutation_prob` = 0.3
  - `max_attempts` = 100
- The solution might not start with index 0 but it creates an optimal "circle" so we can start from any city, in our case, Vancouver (index 0)

2. How much shorter is the optimal route?

In [5]:
```
# SOLUTION

# Initialize fitness function object using dist_list
fitness_function = mlrose.TravellingSales(distances = dist_list)
```

In [6]:
```
# Define optimization problem object
problem_fit = mlrose.TSPOpt(length = 17, fitness_fn = fitness_function, maximize=False)
```

In [7]:
```
best_state, best_fitness = mlrose.genetic_alg(problem_fit, random_state = 2, mutation_p
```

In [8]:
```
# the optimal route
best_state
```

Out[8]:
```
array([ 0, 10,  9, 13, 12, 11, 14,  5,  4,  6,  3,  1, 15,  8,  7,  2, 16])
```

In [9]:
```
# optimal travel time
best_fitness
```

Out[9]:
```
136.12
```

In [10]:
```
# our travel time
our_travel = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9 , 10, 11,12,13,14,15,16])
print(fitness_function.evaluate(our_travel))
```

```
139.38
```

In [11]:
```
139.38 - 136.12
```

Out[11]:
```
3.259999999999991
```