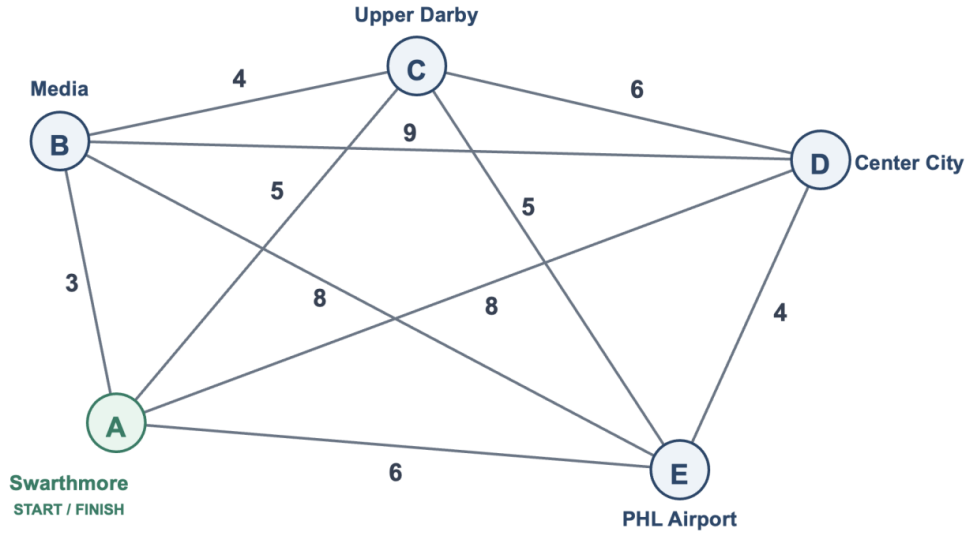
Lab 3

Solving the **T**raveling **S**alesperson **P**roblem

What is the TSP?

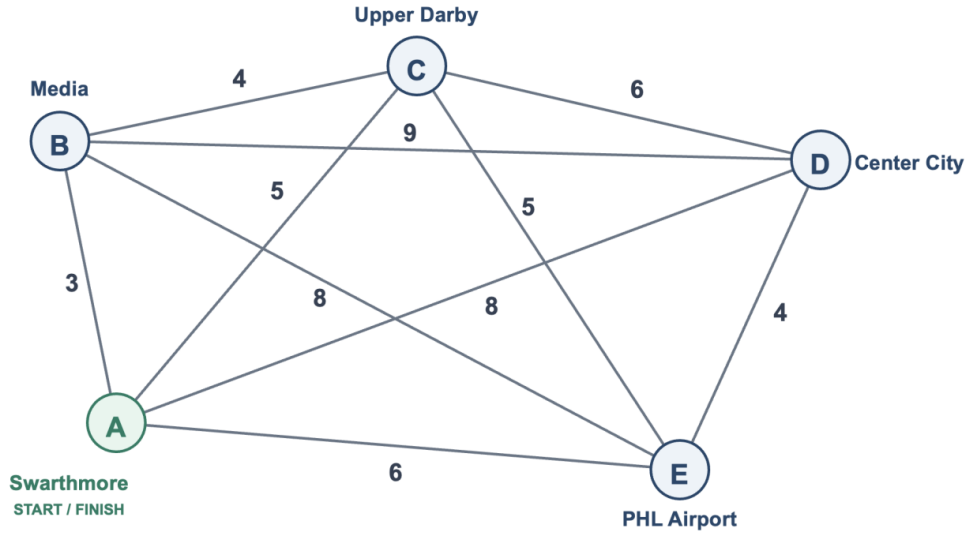
You need to organize a sales tour to pitch your new product to potential clients

- You want to visit *n* cities reducing your total travel distance as much as possible
- Every pair of cities has an an associated travel distance
- The goal is an order of cities that minimizes total travel distance, starting and ending at your home



Real-world applications

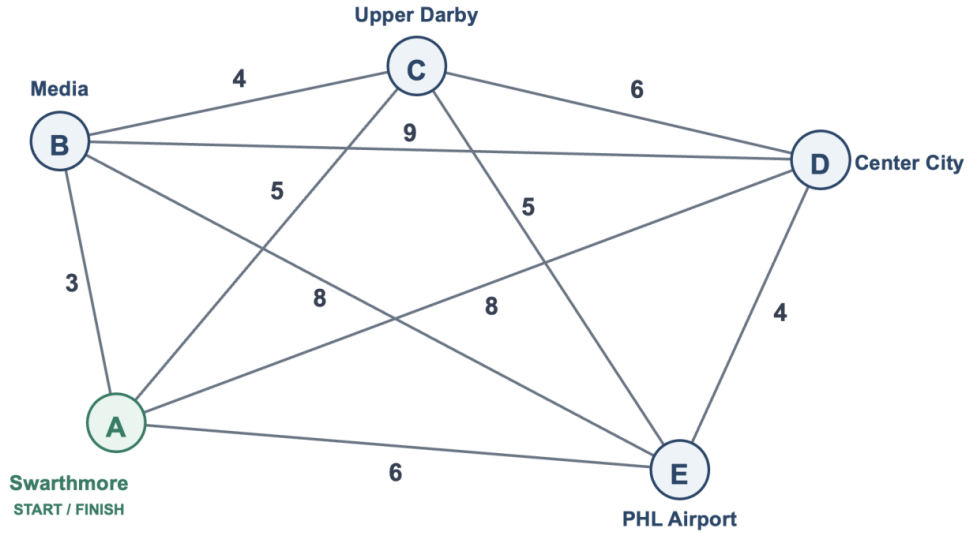
- **Logistics & Route Planning:** Package delivery (e.g., UPS, FedEx) and garbage collection routes.
- **Manufacturing:** Circuit board drilling (minimizing the movement of a drill head across thousands of hole locations).
- **DNA Sequencing:** Ordering DNA fragments to reconstruct genetic sequences.



Computational Complexity: Why is it Hard?

O(*N!) possible tours! Considered NP-Hard*

- 10 cities → ~200K tours
- 15 cities → 10^10
- 30 cities → more tours than atoms in the universe

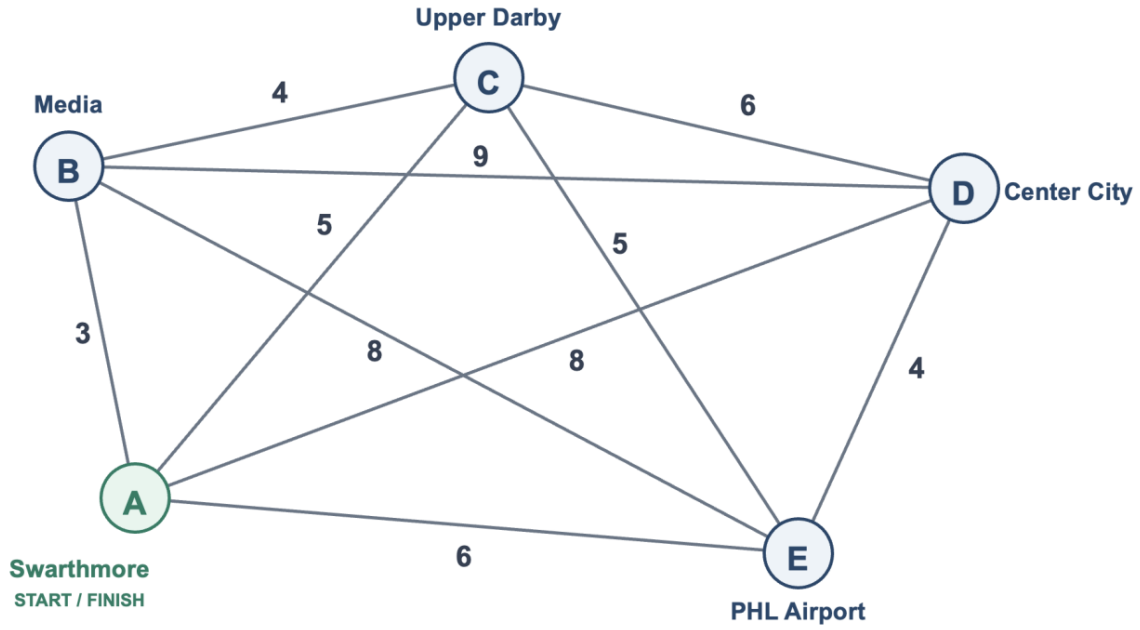


TSP as local search

1. State representation:

2. Objective function:

3. Successor function:



TSP as local search

1. State representation:

Each state is an ordering of cities to visit

A -> C -> D -> B -> E -> A

0 1 2 3 4 5

2. Objective function:

The total travel cost for that order

5 + 6 + 9 + 8 + 6 = 34

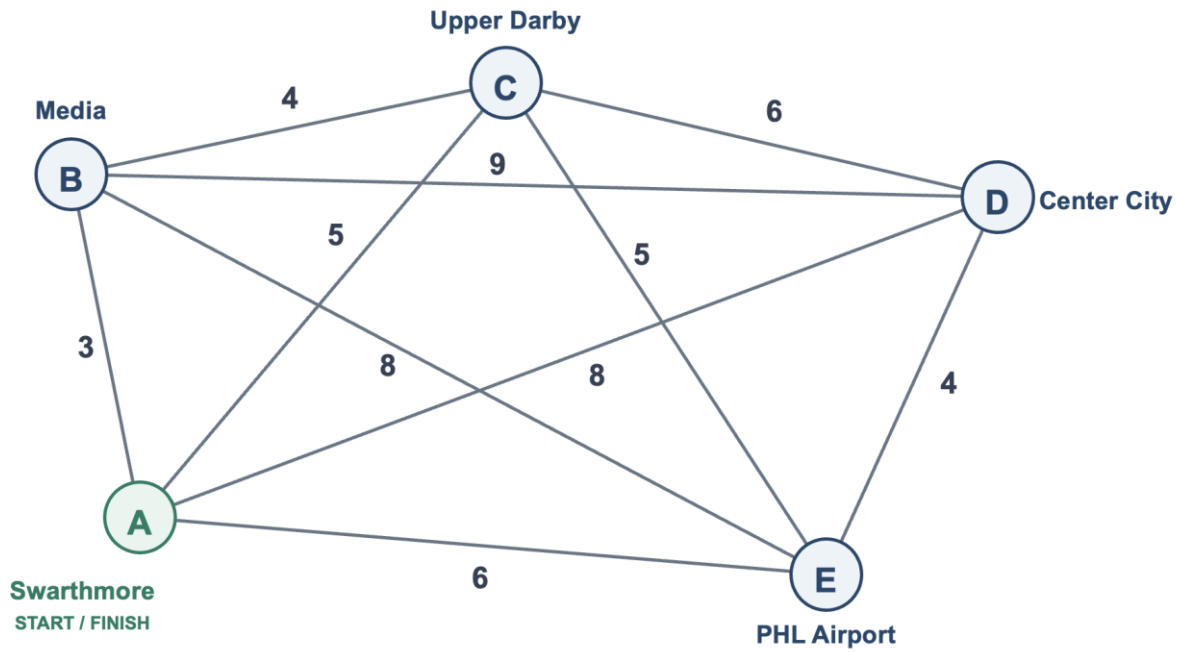
AC CD DB BE EA

3. Successor function:

Move a city to a new position in the order

A -> **E** -> C -> D -> B -> E -> A

Move E to 1



Your goal: Implement 4 algorithms for TSP

- Hill-climbing
- Simulated Annealing
- Beam Seach
- Stochastic Beam Search

Hints for implementation

#1 Use the provided code!

TSP Class:

- Functions for cost, distance between cities, random starting state, plotting and a list of all neighbors are provided!
- You need to implement: *best_neighbor, random_neighbor*

Tour class (a state in this problem)

- Function for moving a city in the order is provided

Search classes:

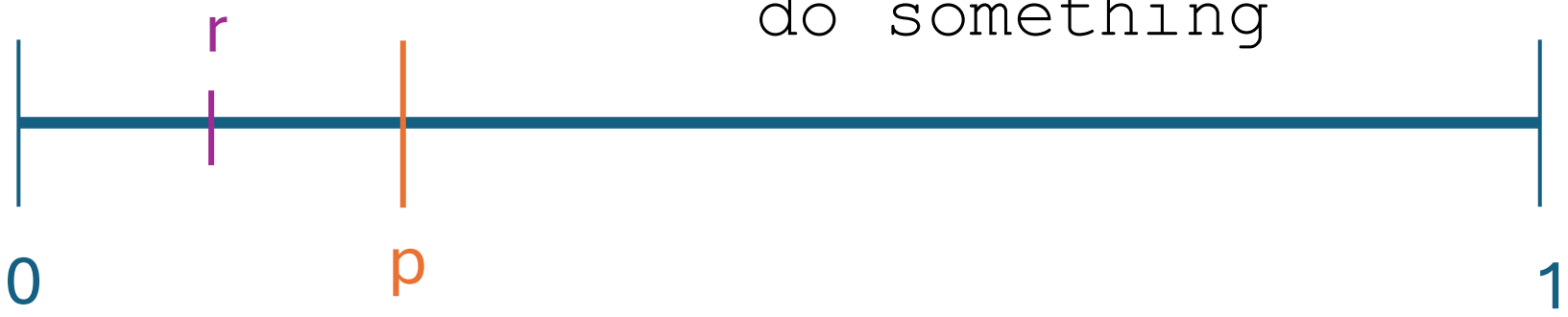
- These are the main things to need to implement!

#2 A random if statement

We want to do an action with probability *p (e.g. 0.35 or 35% chance)*

Python's built-in `random` function will give a random number between 0 and 1. Do our action if the number is < p.

```
r = random()
if r < p:
    do something
```



#3 A random choice

We want choose items from a list with some probabilities:

```
import numpy.random
items = ["a", "b", "c", "d", "e"]
indices = list(range(len(items)))
probs = [0.1, 0.1, 0.2, 0.3, 0.3]

choices = numpy.random.choice(indices, 3, p=probs)


chosen_items = items[choices]
```

Our list

The probability of choosing each item (should sum to 1!)

The random choice function

Operate on indices!