CS 101 Scaffolding a CS 2 Problem

Andrea Li, Carolyn Ruan

April 25 2025

1 Reference Solution

```java
public void printTours(String start, Map<String, Map<String, Double>> dists) {
    List<String> path = new ArrayList<>();
    Set<String> visited = new HashSet<>();

    path.add(start);
    visited.add(start);

    // TA note: use a public-private pair for recursion
    printTours(start, start, path, visited, 0.0, dists);
}

private void printTours(String start, String current, List<String> path,
    Set<String> visited, double currentDistance, Map<String, Map<String, Double>>
    dists) {
   // Base case: visited all the locations
    if (visited.size() == dists.size()) {
        // TA note: check that there is a path back to the start location
        if (dists.get(current).containsKey(start)) {
            double totalDistance = currentDistance + dists.get(current).get(start);

            // TA note: recursive backtracking, the path list must be unchanged
                after printing
            path.add(start);
            System.out.println(path + " (" + totalDistance + " mi)");
            path.remove(path.size() - 1);
        }

        return;
    }

   // Recursive step: visit all unvisited neighbors
    for (String neighbor : dists.get(current).keySet()) {
        if (!visited.contains(neighbor)) {
            visited.add(neighbor);
            path.add(neighbor);

            printTours(start, neighbor, path, visited, currentDistance +
                dists.get(current).get(neighbor), dists);

            // TA note: recursive backtracking, the path list and visited set must
                be unchanged
            path.remove(path.size() - 1);
            visited.remove(neighbor);
        }
    }
}
```

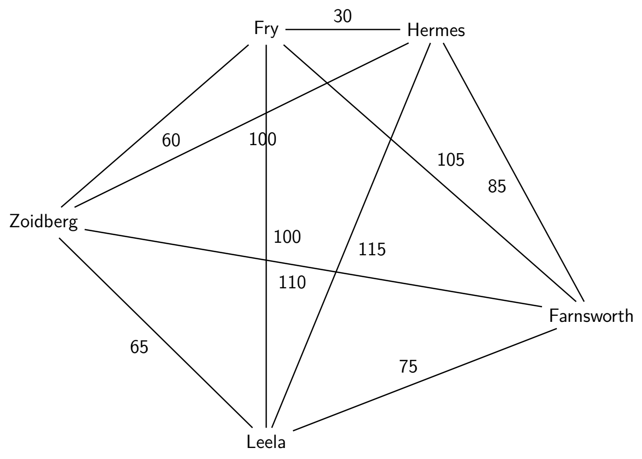
---

2 OH Hints

1. **Public/private pair:** Instead of writing everything in the public `printTours` method, you may write a recursive private helper method which overloads `printTours`, and call that helper function from the public `printTours` method.
2. **Visited set:** You can keep track of the nodes you've visited with a `Set`. Make sure to remove the node after visiting it, and keep track of the distance. It may be helpful to use the `.size()`, `.contains()` methods for Set.
3. **Base case/recursive step:** When can you return to the starting location to complete the tour? Which locations do you visit in the next step?

3 Scaffolded Problem

The *travelling salesperson problem* asks the question “what is the shortest route I can take that starts and ends in the same place and visits every location on my list exactly once?” For example, suppose I am Fry and I’d like to visit all my friends using the following routes:



One route Fry could take is:

Fry → Farnsworth → Leela → Hermes → Zoidberg → Fry

Note that this route visits each other person *exactly once* and starts and ends in the same place. Furthermore, the total distance of this route is $105 + 75 + 115 + 100 + 60 = 455$mi. Another route Fry could take is:

Fry → Leela → Zoidberg → Farnsworth → Hermes → Fry

This route also visits each person exactly once and starts and ends at Fry. However, the total distance of this route is $100 + 65 + 110 + 85 + 30 = 390$mi, which is significantly better than the first route. The goal of this problem is to identify all routes, along with how much distance they take. This allows us to select the best possible route (e.g., the one with the shortest distance).

public void printTours(String start, `Map<String, Map<String, Double>>` dists)

This method should use recursive backtracking to print all possible tours that visit each location in the input file exactly once, except `start` which it should visit at the very beginning and the very end. For the example tours mentioned above, this method would print the following two lines (among others...):

```
>> [Fry, Farnsworth, Leela, Hermes, Zoidberg, Fry] (455.0 mi)
>> [Fry, Leela, Zoidberg, Farnsworth, Hermes, Fry] (390.0 mi)
```

You may output the lines in any order, as long as they are all there. Do not round the distances.

Implementing `printTours` has been broken down into tasks to help you with the problem! Following these tasks is encouraged but not required.

Task 1: Write a public-private method pair for recursion. Initialize the appropriate variables, and think of the parameters you want in your recursive private method.

Task 2a: Write the base case. Hint: when can you return to the starting location?

Task 2b: Write the recursive step. Hint: what are the next locations to visit? Make sure to keep track of which locations you have visited and the path distance.

Task 3: Call your recursive private method from the public `printTours` method.