Call Routing Project

This project is inspired by a real-world problem I solved at a telephony API company – let's call it *Teleo*. The primary task is to implement an international call routing system that finds the least cost route through multiple carriers. One of the goals of this project is to demonstrate the wide variety of solutions and compare the tradeoffs with each.

Background

Phone Numbers

Phone numbers are structured hierarchically and reveal the geography of call routing networks. International phone numbers begin with a '+' followed by the country code, area code, local code, and further groups that represent successively smaller locales. For example, U.S. phone numbers use the format 1-222-333-4444, while U.K. numbers look like +44-222-3333-4444, and Japanese numbers are written as +81-22-333-4444. Fortunately, phone numbers can be normalized into a standard format beginning with a '+' followed by only digits (no hyphens, dots, or spaces). Normalized U.S. numbers look like this: +12223334444. For this project, you will only need to use normalized numbers.

Routes

A route is a path through a carrier's phone network. Longer routes connect specific geographic regions while shorter routes reach larger regions. Hierarchical phone numbers allow routes to be easily represented with just a short phone number prefix. For example, +1415234 identifies a neighborhood in San Francisco, +1415 represents greater San Francisco, +1 covers the entire US, and +4420 covers most of London.

Carriers

Teleo has contracts with multiple telephone carriers which each specialize in different geographic regions, but they often overlap. Thus, their competitive advantage is that they can choose which carrier to route outgoing phone calls through to minimize costs. When finding a route to use in a single carrier's route list, the longest matching prefix must be used, as it's the most specific route. Some phone numbers may match prefixes in multiple carrier route lists, and in this case you may choose the least expensive one. In the rare case of identically priced routes, either is acceptable.

Data Files

Input Data

Input data comes in several plain text files. Some represent carrier route lists while others contain phone numbers of which you need to find the least cost route.

Carrier Routes

Each carrier's routes and their associated costs are given in a single file. Each line is a single route formatted as a comma-separated pair: a route's normalized prefix (starting with a '+'), then its cost in USD (a floating-point number). For example:

```
+1512,0.04
+1415,0.02
+1415234,0.03
+1415246,0.01
```

These files are named "carrier-routes-N.txt" where N is an integer. (Carrier 1, 2, 3, ...)

Using the carrier route list above, the cost of calling +14152345678 would be $0.03, as it matches with the prefixes +1415 and +1415234 (longest match), but not with +1415246.

Phone Numbers

The phone numbers you need to look up the route costs for are each normalized and given on separate lines of a file. For example:

```
+15124156620
+14152345678
+19876543210
```

These files are named "phone-numbers-N.txt" where N is an integer.

Output Data

After finding the least cost route for each phone number, write the number and its cost on a comma-separated line of a new text file. If there is no route for a number, write 0. For example, using the carrier route list and phone numbers given above, the output is:

```
+15124156620,0.04
+14152345678,0.03
+19876543210,0
```

Name these files "route-costs-N.txt" where N is the scenario number (see below).

Project

Scenarios

Each scenario below represents a particular situation with different dataset sizes and constraints based on a real-world problem setting. Consider how you might solve the problem in each scenario, accounting for both development time and actual runtime. Which scenarios simply need a "good enough" solution and which ones require more analysis and development time to handle the problem's underlying complexity? When should you use available tools and when should you build a solution from scratch? Sometimes the "best" solution may surprise you...

Scenario 1: One-time route cost check

You have a carrier route list with 100,000 (100K) entries (in arbitrary order) and a single phone number. How quickly can you find the cost of calling this number?

Scenario 2: List of route costs to check

You have a carrier route list with 100,000 (100K) entries (in arbitrary order) and a list of 1000 phone numbers. How can you operationalize the route cost lookup problem?

Scenario 3: Multiple long carrier route lists

You have 5 carrier route lists, each with 10,000,000 (10M) entries (in arbitrary order) and a list of 10,000 phone numbers. How can you speed up your route cost lookup solution to handle this larger dataset?

Scenario 4: High-throughput pricing API

You have 5 carrier route lists, each with 10,000,000 (10M) entries (in arbitrary order) and you want to create a web-service API to allow clients to price a call before it is initiated. How can you create an efficient route cost lookup solution that can handle high spikes of traffic (up to 10,000 requests per minute) without overloading your API servers?

Scenario 5: Changing carrier route lists

Consider the challenges imposed by carriers changing their route lists occasionally. How would you change a subset of the route cost data while the pricing API remains operational?