Instructions

For the solution, you are free to use any language you are comfortable with, though we especially like to see solutions written in Scala. You may not use any external libraries to solve this problem; however, you may use external libraries or tools for building or testing purposes. Specifically, you may use ScalaTest, JUnit, Sbt or Maven to assist your development.

You should also include a brief explanation of your design and assumptions along with your code. The problem description might not be unambiguous therefore you might have to make assumptions.

The solution needs to process input and produces output. As a good citizen your solution reads input from stdin and writes to stdout. You should provide sufficient evidence that your solution is complete by, as a minimum, indicating that it works correctly against the supplied test data.

For the purpose of testing you are free to implement an adequate mechanism for feeding input into your solution (for example, using hard coded data within a unit test).

Generally you should aim at making it easy for the reviewer to understand your solution. Please keep it simple, useful, maintainable and testable.

We are not looking to see fully matured solution with optimised algorithms but for understanding of what kind of problems might occur. Focus on clean code, good abstractions, names, simple straightforward solution. DO NOT over-complicate or try to show off.

You might think about time-boxing it to 2 hours but this is up to you, we will fully understand if you'll share this with us as an assumption.

Public Transport Routing

You are asked to develop a new routing solution for Berlin's public transport system that lets people search for the quickest route between two given stations. Your customer is also asking for the possibility to find stations nearby a given station that can be reached within a given time.

The transport network is modeled as a directed graph with edges that represent a connection between the two stations and are labelled with the travel time in seconds. If a transport line can be used in two directions it is modeled as two distinct, opposing edges. The stations are identified with alphanumeric strings without special characters or white space, such as "A" or "B" or "BRANDENBURGERTOR".

To keep things simple the train schedule and changing times at stations will be ignored for the first release, i.e. assume there is always a train traveling along every edge.

Input

The routing program starts by reading the transport network graph. The first line of input is the number of edges in the graph. The edges are defined on a single line each in the form:

<source> -> <destination>: <travel time>

Where <source> and <destination> are station identifiers and <travel time> is a decimal number representing the travel time between source and destination in sec- onds.

After the graph has been defined the routing program is ready to process routing queries. Each query is followed by a newline.

There are two kinds of queries. A routing query which asks for the shortest route be- tween two stations takes the following form:

route <source> -> <destination>

A nearby query which asks for all the stations that can be reached from a given station within a given time looks as follows:

nearby <source>, <maximum travel time>

Output

The output for each query goes to a separate line.

The output for a routing query is the ->-separated route followed by a colon and the travel time:

<source> -> <waypoint1> -> ... <destination> : <travel time>

If no route exists an error message is printed.

Error: No route from <source> to <destination>

For nearby queries the output is a comma separated list of stations with the travel times ordered by ascending travel times.

<destination1>:<travel time>, <destinationation2>:<travel time>, ...

Test Input

8
A -> B: 240
A -> C: 70
A -> D: 120
C -> B: 60
D -> E: 480
C -> E: 240
B -> E: 210
E -> A: 300
route A -> B
nearby A, 130

Expected Output

A -> C -> B: 130
C: 70, D: 120, B: 130