Distributed Tracing

One of the use cases for Instana is distributed tracing. The way this works is as follows, Instana provides its customers with agent software that the customers run on their servers. One of the things this agent software does is finding out if there are any microservices running on the server. If it finds such a microservice, it will start instrumenting that service and it starts tracking key performance indicators (KPI's) of the HTTP connections between this service and its neighboring services by looking at the HTTP library calls that service is performing. One of these KPI's the agent tracks is latency. Please note that latency is a uni-directional concept. When the latency of service A calling service B is 8ms, it is not implied that service B calling service A will also take 8ms as this depends on the underlying network infrastructure.

A trace is a series of one or more HTTP calls, starting in some service and ending in another service. You are tasked with building a program that gives Instana's customers information about the traces within their systems.

Input:
A directed graph where a node represents a microservice and an edge represents a connection between two microservices. The weight of the edge represents the average latency between those two services. A given connection will never appear more than once and for a given connection the starting and ending service will not be the same service.

Output:
For test input 1 through 5, if no such trace exists, output 'NO SUCH TRACE'. Otherwise, follow the trace as given; do not make any extra hops! For example the first problem asks for the total average latency of a trace originating in service A, service A makes a call to service B followed immediately by service B making a call to service C.

1. The average latency of the trace A-B-C.
2. The average latency of the trace A-D.
3. The average latency of the trace A-D-C.
4. The average latency of the trace A-E-B-C-D.
5. The average latency of the trace A-E-D.
6. The number of traces originating in service C and ending in service C with a maximum of 3 hops. In the sample data below there are two such traces: C-D-C (2 stops) and C-E-B-C (3 stops).
7. The number of traces originating in A and ending in C with exactly 4 hops. In the sample data below there are three such traces: A to C (via B, C, D); A to C (via D, C, D); and A to C (via D, E, B).
8. The length of the shortest trace (in terms of latency) between A and C.
9. The length of the shortest trace (in terms of latency) between B and B.

10. The number of different traces from C to C with an average latency of less than 30. In the same data, the traces are C-D-C, C-E-B-C, C-E-B-C-D-C, C-D-C-E-B-C, C-D-E-B-C, C-E-B-C-E-B-C, C-E-B-C-E-B-C-E-B-C.

Test Input:
For the test input, the microservices are named using the first few letters of the alphabet from A to E. A trace between 2 microservices (A to B) with a latency of 5 is represented as AB5.
Graph: AB5, BC4, CD8, DC8, DE6, AD5, CE2, EB3, AE7

Expected output:

1. 9
2. 5
3. 13
4. 22
5. NO SUCH TRACE
6. 2
7. 3
8. 9
9. 9
10. 7

Overall Expectation:
We ask you to provide a *working* program that reads the comma-separated input graph from a text file and prints out the 10 lines of expected output. Make sure that your solution is properly tested (ideally written test-first) and that documentation on how to run the program is provided.