Navmap reconstruction

Algorithms and Datastructures 2014-2015
Project 1

October 14, 2014

1 Background

In this exercise you will solve a computational problem from the space smuggling business.

Smuggler Spike travels from star to star each cycle to bring various illegitimate goods to the local population. As a smuggler, the use of Stargates is out of the question, as the authorities execute extensive cargo scans around such gates. Luckily, his spaceship is capable of entering wormholes, of which various numbers can be found around any star. These wormholes can act as bridges between two points by bending spacetime. Through these spacial anomalies instantaneous bi-directional travel is possible.

After an attack by space pirates, Spike's datacore was damaged, and he partially lost his precious navigational charts. The space pirates were chased away by Bounty Hunters, leaving Spike stranded near an obscure planet. He needs to reach a safe haven as fast as possible, without attracting any attention.

He only has his communications logs, from which his ship's computer has inferred the time required to travel to and from each star system. He also has the location of each wormhole. He now only needs to find out where each of these wormholes lead. Implement an algorithm to reconstruct Spike's navigational charts.

2 Problem description

Given N stars and for each pair of stars the shortest time required to travel between these stars, reconstruct the navigational charts. These charts consist of a list of N wormholes. (There are exactly as many wormholes as there are stars, as far as Spike knows) List for each wormholes the two stars it connects, and how much time is required to fly to such a wormhole, align, travel through it, and fly to the corresponding star. Spike has always travelled via the fastest route available.

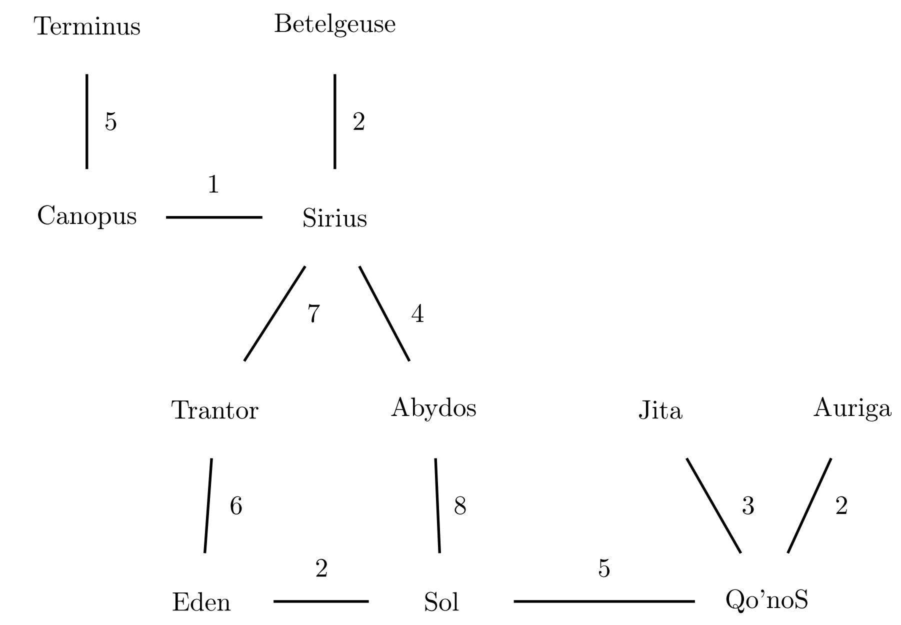

Figure 1: A navigational chart

$$\begin{array}{rl}
0. & \text{Terminus}\\
1. & \text{Betelgeuse}\\
2. & \text{Sirius}\\
3. & \text{Canopus}\\
4. & \text{Trantor}\\
5. & \text{Abydos}\\
6. & \text{Jita}\\
7. & \text{Auriga}\\
8. & \text{Eden}\\
9. & \text{Sol}\\
10. & \text{Qo'noS}
\end{array}
\qquad
\begin{pmatrix}
0 & 8 & 6 & 5 & 13 & 10 & 26 & 25 & 19 & 18 & 23\\
8 & 0 & 2 & 3 & 9 & 6 & 22 & 21 & 15 & 14 & 19\\
6 & 2 & 0 & 1 & 7 & 4 & 20 & 19 & 13 & 12 & 17\\
5 & 3 & 1 & 0 & 8 & 5 & 21 & 20 & 14 & 13 & 18\\
13 & 9 & 7 & 8 & 0 & 11 & 16 & 15 & 6 & 8 & 13\\
10 & 6 & 4 & 5 & 11 & 0 & 16 & 15 & 10 & 8 & 13\\
26 & 22 & 20 & 21 & 16 & 16 & 0 & 5 & 10 & 8 & 3\\
25 & 21 & 19 & 20 & 15 & 15 & 5 & 0 & 9 & 7 & 2\\
19 & 15 & 13 & 14 & 6 & 10 & 10 & 9 & 0 & 2 & 7\\
18 & 14 & 12 & 13 & 8 & 8 & 8 & 7 & 2 & 0 & 5\\
23 & 19 & 17 & 18 & 13 & 13 & 3 & 2 & 7 & 5 & 0
\end{pmatrix}$$

Figure 2: The corresponding navigational records

3 Interface

Write a program that accepts navigational records from Standard Input. The input begins with an integer N representing the number of stars *and* wormholes Spike knows. You may assume $N < 10000$. The number of stars is followed by a line with just a dash '-'. Next are N lines containing N integers, where on line i the jth integer T_{ij} represents the shortest time required to travel from star i to star j.

The output should be printed to Standard Output, beginning with the number of stars N.

Again a line with just a dash '-' follows. Next N lines follow, with on each line an inferred wormhole connection. A connection consists of two integer points i and j, identified by the index corresponding to their stars, and an integer W_{ij} representing the time required to travel to and from these stars using just this wormhole.

4 Assignment

Implement a solution to this problem, preferably in Java. If you use another language, make sure we can compile and run your code easily. You are only allowed to use the Standard Library corresponding to your selected language. On the website you will find a Java Eclipse framework in which you can implement your solution. This framework contains the necessary classes to read from Standard Input and print answers to Standard Output. You will also find a sizable set of examples and code to automatically test your solution in this framework.

Your solution will be graded based on the number of problems it can solve in a reasonable amount of time. If your solution generates incorrect results a serious amount of points will be deducted from your grade. You can check the correctness of your implementation using the tests provided, so please make sure you hand in a correct implementation.

Also hand in a report describing your implementation. Please include a complexity analysis of your solution. Mail your report and your source files to alexandra@cs.ru.nl before **8th of November, 23:59, Nijmegen time**.

5 Grading

The grading will be done by running several tests, followed by inspection of the code and reading the report. The report is worth 20 points, passing all tests will award you 65 points, the quality of the code 15 points.