CURTIN UNIVERSITY (CRICOS number: 00301J)

Department of Computing, Faculty of Engineering and Science

Data Structures and Algorithms (COMP1002)

PRACTICAL 2 – FILE I/O

AIMS

- To implement the loading and saving of the network from and to files.
- To serialize Network.

BEFORE THE PRACTICAL:

- Read this practical sheet fully before starting.
- If you're unsure on the details, revise the lecture notes.
- Download the sample network file sample.nt from Blackboard

ACTIVITY 1: READING IN A FILE

The network files for the assignment will be provided to you in XML. You will need to read these in before the building of the decision diagram can begin. The sample document shows the general format of a network file.

The network file is enclosed in the <NET> and </NET> tags. All information relating to the network should be found between these.

Each vertex is defined in a VERTEX tag that has components NAME, REL (reliability), COST and TYPE. The name and type strings are in double quotes and the reliability should be interpreted as a floating point number (either float or double, depending on your class choice). The cost is the cost in cents (US) per unit. A vertex can also have either SOURCE or TARGET as an attribute, which defines the communication requirement. We generally have the first vertex as the source vertex and the last be the target, but this isn't always the case and is actually a disadvantage in some cases so don't make that assumption.

Each edge is defined in an EDGE tag that has the same components as above but also has two additional fields, FROM and TO. Both of these fields are strings that detail the endpoints of the edge. Since our edges are assumed to be undirected the order of these two doesn't really matter, so you could theoretically switch the 'from' and 'to' endpoints if needed. Each of the FROM and TO strings should match the name of a vertex read in earlier. Note that the cost for edges represents the cost in cents (US) per meter of the component, and this cost can be zero for wireless connections.

The example details all vertices first, and then all edges. This is not necessarily required, but a vertex must be described before it can be used as the endpoint for an edge. In general, it's easier to list all vertices first.

In order to read in such a file, you should grab one line at a time and then tokenize it. The first token should tell you what sort of line you are processing and you'll then need to create the appropriate object and store it.

Because you are using arrays you may need to do some pre-processing. This means reading through the file once to count the number of items that you're going to store and then a second time to do the actual object creation and storing. Theoretically you could hope to buffer the read input but since some files will be large (tens of thousands of lines) this won't always work.

Note: The easiest way to do this is to close the file and then re-open it to start from the beginning again. A smarter way is to look up the use of FileChannel although it's also possible to use a RandomAccessFile. This is beyond the scope of the unit though, so you won't have marks deducted for just closing and re-opening.

Your task: Add functionality to your classes to enable them to read in a network. Make sure that you network class has a display method so you can test whether it has worked. You'll also need to decide whether to read in as part of the constructor method or whether to construct an empty object and then order it to read in data.

ACTIVITY 2: WRITING TO A FILE

In scientific work, it is always good to be clear on the benefit and cost of performing various actions. Sorting takes time, especially for large files. It is best not to have to do this every time you read in a network file. For this reason you will need to be able to write your sorted network to disk.

You could just write the network file to disk in the same XML format. This is very readable, and thus it's easier to see if you've made mistakes. You can fairly much take your methods for reading the data and transform them in the same way as discussed in the lecture.

Writing and reading in from XML is not efficient – it's even less efficient than using a CSV format and far less efficient than binary. For the assignment you will be tasked with being as efficient as possible, so consider creating a different file format to save sorted files (they should have the ".srt" extension rather than the ".nt" extension that unsorted network files have).

Your task: Consider the way to make the handling of sorted networks as efficient as possible. Next week you won't be marked on having implemented this, but you will be asked about your thoughts and you will be marked on efficiency in the assignment.

ACTIVITY 3: SERIALIZATION

Creating a read/write method for every class is quite time consuming. An alternative is to use Java serialization to have this done automatically.

Your task: Implement serialization for the network and use that to save and reload a network.

SUBMISSION DELIVERABLE:

Your three classes (Vertex, Edge and Network) are <u>due at the beginning of your next tutorial</u>. Also include any other relevant classes that you may have created. Please only submit the *.java files although if you have a Makefile, including that would be very welcome.

Your classes should include the ability to read in files like the sample file (I'll be testing it on a different file with the same format) and to read and write through serialization. Your files also need to be able to save themselves in some non-serial format that you have decided on, and if that format is different from the standard then you'll need to be able to read that in as well. I don't care about that format at this point, but you'll have to argue your choice of format in the assignment.

You do not need to submit your test harness if everything is fine. If there are doubts about some part of your program though, the test harness could be quite useful but it won't be available if you haven't submitted it. I will not allow you to bring code from your account during the testing since this adds uncertainty – we'll work with what you've submitted.

SUBMIT ELECTRONICALLY VIA BLACKBOARD, under the *Assessments* section.

If you finish early, use the rest of the practical to start the next worksheet, because that will be due later on.