Arizona State UniversitySER222: Data Structures & Algorithms

Lect. Ruben AcuñaRevised 10/29/2016 [Fall]

Determining Optimal Logographic Chapter Memorization Order

Summary: In this assignment, you will implement a topological sort algorithm as a way to reorder kanji (漢字; Japanese logographic characters described from Chinese) into an order optimal for memorization.

1 Background

In this assignment, you will practice applying your knowledge of graph data structures and algorithms to determine an optimal order to learn logographic characters. Consider this: one interesting dependency graph that can be constructed is a “component” graph for Hanzi characters, or Hanzi derived characters (e.g., Kanji). You see, complex characters are often built from simpler characters as components. This sub-structure is very useful! Components not only define a common appearance and stroke order but can indicate phonetics and/or semantics. Furthermore, there are considerably fewer components (hundreds) than actual characters (thousands). (Please note that we use the term component very generally here - it does not map to the traditional notion of *a radical*.) This sub-structure is particularly useful for people memorizing characters - instead of looking at each character as a monolithic block, one can memorize the individual components, and then reuse that knowledge for more complex characters. The following graph is an example of this for the character 法 ("method"):

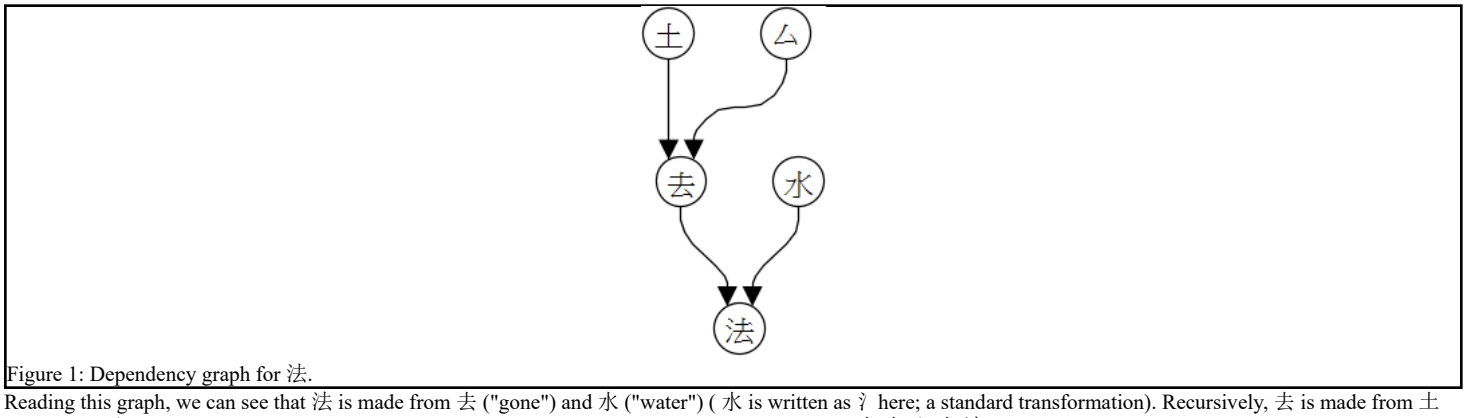

Figure 1: Dependency graph for 法.

Reading this graph, we can see that 法 is made from 去 ("gone") and 水 ("water") (水 is written as 氵 here; a standard transformation). Recursively, 去 is made from 土 ("soil") and 厶. Based on these dependencies, we would want to learn these characters in the order: 土 水 厶 去 法. If we do that, then instead of learning each stroke in 法, we just have to remember "method=water+gone", which tells us to write 氵 and 去. (For more information on these ideas, see Remembering the Kanji by James Heisig.) That said, in order to make use of this recursive structure, we would have to learn characters in an order such that we also see simpler characters before we see the more complex characters that are built out of them. Luckily, this is straightforward – this is exactly what a topological sort of a dependency graph will give us.

This document is separated into four sections: Background, Requirements, Testing, and Submission. You have almost finished reading the Background section already. In Requirements, we will discuss what is expected of you in this homework. In Testing, we give some basic suggestions on how the map implementations can be tested. Lastly, Submission discusses how your source code should be submitted on BlackBoard.

2 Requirements [100 points, 20 extra credit]

In this assignment you will implement an editable graph data structure and a new algorithm for finding the topological sort of a graph. Our goal will be to load two data files, and print out a topological order for the characters that they list. Attached to this post are a base file and two interfaces. Further down the BlackBoard page, you'll find a PDF describing the Java implementation of hashtables (you may find it useful), and a visualization of the dependencies in the data files. The data is formatted as follows:

data-kanji.txt (UTF-8 formatted) stores nodes:

- Tab separated.
- # prefixes comment lines.
- Lines look like <characterID1> <character1>, which indicates that character1 can be represented as the number characterID1. IDs are just integers.

data-components.txt (ASCII formatted) stores edges:

- Tab separated.
- # prefixes comment lines
- Lines look like <character1ID> <character2ID>, which indicates that character1 is a component of character2.

In terms of programming, you will need to:

- Create a new class called BetterDiGraph that implements the EditableDiGraph interface. See the interface file for details. [40 points]
- Create a new class called IntuitiveTopological that implements the TopologicalSort interface. Use BetterDiGraph to store the graph. [40 points]
    - Instead of using DFS to find a topological sort, implement the following algorithm: "IntuitiveTopological". This algorithm works as follows: look at your graph, pick out a node with in-degree zero, add it to the topological ordering, and remove it from the graph. This process repeats until the graph is empty.
    - Make sure to check for cycles before trying to generate a topological sort of the graph!
- Complete the main method in LastNameMain.java. It should: [20 points]
    - Load data-kanji.txt, use it to populate a hashtable that maps IDs to characters, and add the IDs as nodes in the graph.
    - Load data-components.txt, and use it to add edges to the graph being built.
    - Create an IntuitiveTopological object, and use it to sort the graph.
    - Display the characters in the ordering. Note that topological sort will produce a list of a IDs - you'll need to take the IDs and uses them to look up the correct character in the hashtable you populated earlier.

- **Extra Credit:** add support for visualizing the graph. Most likely this will take the form of using a graph library such as GraphViz to render an image for the data you load. [20 points]
- If you find yourself using importing packages other than java.util.LinkedList, java.util.HashMap, java.util.NoSuchElementException, or java.io.*, please double check with your instructor that they may be used.
- Do not import any packages other than Queue or LinkedList in your map implementations.
- 頑張って!!

3 Testing

Below is a sample output for your program that contains the characters in the default order from the file, and the order resulting from a topological sort. Note that your topological sort may be in a slightly different order than is shown below. The key is that simpler characters are shown before the more complicate characters that are built from them.

```
run:
Original:
算法設計五目吾冒朋明唱早旦胆白千占卓見元句勺亘旬古昌口世日只言肌的昇百員水貞旧寸竹二十首胃品晶呂舌一博土月田旭朝去頁升専貝
Sorted:
五目見勺口吾占句日冒唱白旬昌只言設的水旧竹算二元十計早千卓古世寸品晶呂舌一旦亘百土月朋明胆肌田胃朝去法升昇専博貝員貞頁九旭
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 2: Sample output.

4 Submission

The submission for this assignment has only one part: a source code submission. It should be uploaded to the submission link on BlackBoard.

Writeup: Not required.

Source Code: Please zip your source code files together as "LastNameTS.zip" (e.g. "AcunaTS.zip"). It should contain only three files: LastNameMain.java, BetterDiGraph.java, and IntuitiveTopological.java. Be sure that you use the correct compression format - if you do not use the right format, we may be unable to your submission. Do not include your project files, or leave code in your files that is specific to your IDE/project.