Algorithms & Data Structures in Practice: Learning to Read Japanese

Summary: In this assignment, you will perform a basic market analysis for a proposed a language learning problem, and then implement a topological sort algorithm as a prototype solution.

1 Background

Learning Objectives:

- LO1: Identify the commercial market (if any) for a proposed problem. (EM@FSE K)
- LO2: Evaluate a proposed technical solution to estimate the commercial value that may be generated. (EM@FSE J)
- LO3: Develop a data structure for a directed graph ADT using an API description. (EM@FSE G)
- LO4: Develop a directed graph algorithm from a textual description using an API. (EM@FSE G)

Consider the following problem: how can an English speaker effectively learn Japanese? Unlike the English language, which uses the Roman alphabet to encode phonetics, many east Asian languages use Hanzi derived characters, which are symbols with semantics bound to them (to use the technical term: logographic character). Oftentimes, English speakers learning a language using logographic characters find themselves stumped by the apparently insurmountable problem of memorizing thousands of unique characters. They all look so different and yet so similar - can we hope to tell them apart or write them? 大変な問題！Wouldn't it be nice if we could use our knowledge of algorithms and data structures to somehow address this problem so that English speakers would have a better shot at learning Chinese or 日本語...?

The Foreign Service Institute (FSI) ranks Japanese and Mandarin as Category V, meaning they are exceptionally difficult for English speakers to learn, requiring around 2200 hours. However, both languages have seen growth in the number of learners over the past several years. Learning Japanese is difficult for many reasons, including:

- Requiring the acquisition of around 2000 characters to read/write native content. For example: "The cat walked on the computer's keyboard."translates as "猫はコンピューターのキーボードの上を歩いた。"In English, this sentence has 34 characters, with 18 unique. The Japanese representation has 22 characters, 18 unique. Of these, 3 (猫, 上, 歩) are Chinese derived Japanese characters ("Kanji"), 6 (コ, ン, ピュ, タ, キ, ボ, ド) are using the syllabary system for non-native words, and 4 (の, を, い, た) are using the syllabary system for native words.
- The differences in grammar between English and Japanese. In Japanese, word order is Subject-Object-Verb ("The cat running"), while English uses Subject-Verb-Object ("The running cat"). Learners often struggle with trying to understand sentences when the meanings they need aren't in the place they expect.
- The relative infrequency of speakers outside of Japan.

Often language learners choose other languages like Spanish because they are less intimidating (no special characters to learn) and have a more immediate payoff (Spanish is the second most spoken language in the United States).

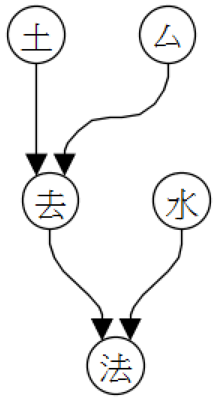


Figure 1: Dependency graph for 法.

Despite these roadblocks, Japanese is still a relatively popular language to try to learn. Reasons for learning Japanese very widely: to enable travel, for business purposes, and to enjoy popular culture. Often many people begin studying Japanese because of its uniqueness and value, but then stop after experiencing firsthand the difficulty of learning the language. A common reason is being overwhelmed by using Kanji characters. Learners see the years of practice that most people need to be able to successfully read or write and realize they don't have the time or motivation to get that far.

All of this represents a business opportunity: people wishing to succeed in learning Japanese will look for tools/technologies that can help them in achieving their language learning goals. If we can create a product that enables learning Japanese, we are building value for a customer, and that value that may be worth money. In most assignments on data structures & algorithms, the focus is on practicing a specific technical skill. However, it's important to recognize that these skills do not exist in a vacuum - there is no reason to know how to use a linked list, if there isn't any place where it should be it used, or a customer for whom it creates value. As a step towards contextualizing our technical skill knowledge, we will create a basic market assessment for Japanese learning software. Language learning software is, of course, an application area for the skills we have learned.

The homework involves two parts: 1) Assess the value of creating a solution to the problem. 2) Implement a partial algorithmic solution.

- **Market Analysis:** In the market assessment portion, you will work to identify the current market for language learning solutions, and the value of a new solution. You will create a report that first surveys the general Japanese language learning market in terms of software offerings and learner needs, and then dives down into evaluating a product that specifically helps to learn Kanji.

- **Algorithm Implementation:** Providing an application to help users learn Japanese is a very broad task. For the sake of being a reasonably sized assignment, we'll look at a specific aspect of learning Japanese that most learners find very difficult: memorizing the Kanji characters. In this part, you will practice applying your knowledge of graph data structures and algorithms to determine an optimal order to learn the logographic Kanji characters. In class, we introduced the idea of a *dependency graph* that represents how "tasks"need to be ordered. One interesting dependency graph that can be constructed is a "component"graph for Hanzi characters, or Hanzi derived characters (e.g., Kanji). 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"):

  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.) Note that in order to make use of the 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. To solve this problem, we can produce a *topological sort* of a graph. A topological sort is an ordered list of vertices in graph such that all dependencies are listed before their dependent. Anecdotally, it is said that people learning the characters in this special order can learn to write the basic 2000 characters in around three months - awesome!

This document is separated into four sections: Background, Market Analysis, Algorithm Implementation, and Submission. You have almost finished reading the Background section already. In Market Analysis, we will discuss what is expected of your market report. In Testing, we give some sample output for the program. Lastly, Submission discusses how your source code should be submitted on Canvas.

2 Market Analysis Report [20 points]

As a first step, we want to determine if the problem that we have proposed is indeed a relevant one to solve. To do this, we need to identify the market for the proposed problem. The market for us to investigate is: Japanese language learners interested in software solutions. Even if we find a wide market, we also need to know if it makes commercial sense to pursue it (even if a software product is potentially very useful, it doesn't matter if no one will pay for it). It might be that the market is saturated with existing products, making little sense to develop a new one.

This is a research report: you should use your research skills to learn more about the market for Japanese learning software. Note that we aren't expecting a super fancy report - we haven't taught you how to do a formal market analysis, that will come later in your education - rather we want to start thinking about how we can create value for real customers using the technical skills we've been acquiring. Your report will be comprised of two sections that aim to answer the following:

1. What is the commercial market (if any) for software to aid Japanese learners? [10 points]
2. What commercial value would be generated by a software tool that aids learners in acquiring Kanji? [10 points]

Questions to consider:

- How popular is learning Japanese among those learning a language?
  - What does growth in this area look like?
  - What are some existing software products on the market that can help learning Japanese?
  - Is the market already saturated with Japanese learning products, or is there potential for growth?
- Do other products include functionality to quickly learn Kanji?
  - Using these other products as a baseline, what type of revenue stream (payment models, amounts, etc), might be possible?
    - (optional) Is this revenue stream sufficient to support a developer?
- (optional) Can you identify any gaps in the language learning software products you find?

(These particular questions are given as guidance. In future courses, you might start only with a general idea for a problem and then need to do all of the market research from scratch.)

Report requirements:

- The report must be “integrated”. Rather than explicitly answer the questions above, integrate the information you find into a full discussion.
- The report must include a header that contains your name, the class, and the assignment. No specific (e.g., MLA, APA) format is required although you are welcome to use one.
- The report must be professionally written, with minimal spelling or grammatical mistakes. Avoid casual wording, and try for something you could see sending to your boss as a preliminary assessment for a future project.
- The report must be single or double spaced and have references included at the end.
- The report should be about two pages - it may be shorter or longer though. Your goal is not to hit a page length but rather to communicate your findings.

3 Algorithm Implementation [60 points, 8 extra credit]

Once you’ve completed the analysis portion, it’s time to move on to programming. 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. Under the Symbol Table module’s readings page, you’ll find a PDF describing the Java implementation of hashtables (you may find it useful). Also below is 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>, e.g., “120 頑”, 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. These lines should be skipped similar to a line starting with // in Java.
- Lines look like <character1ID> <character2ID>, e.g., “92 73”, which indicates that character1 is a component of character2.

Hint: the first line of each data file will always be a comment listing the column names. If needed, you may skip over the first line of the two input files. This can help on some systems where Java will insert a special character into the first line to indicate it is a UTF-8 formatted file.

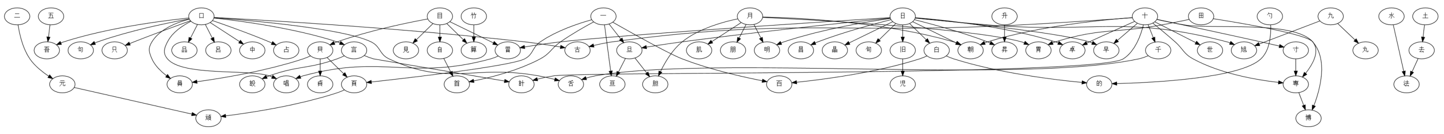
In terms of programming, you will need to:

- Create a new class called BetterDiGraph that implements the EditableDiGraph interface. See the interface file for details. [22 points]
  - The class should include a default constructor (i.e., one without any arguments that properly sets up an empty graph). Additional constructors may be created as well.
- Create a new class called IntuitiveTopological that implements the TopologicalSort interface. Use BetterDiGraph to store the graph. [20 points]
  - The constructor to IntuitiveTopological should take a BetterDiGraph typed object. Additional constructors may be created as well. A default constructor is not required.

  - Instead of using DFS to find a topological sort, implement the following algorithm: "Intuitive-Topological". 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: [18 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 that you generate. Most likely this will take the form of using a graph library such as GraphViz to render an image for the data you load. An example might look like the image below. [8 points]



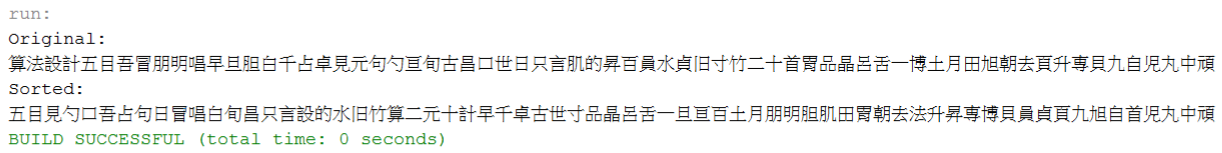
- If you find yourself adding import packages other than java.util.LinkedList, java.util.HashMap, java.util.NoSuchElementE or java.io.*, please double check with your instructor that they may be used.

- Beware that we may test on different data files than are provided as a sample. New data files will follow an identical and valid format.

- 頑張って!!

3.1 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 complicated 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九自児丸中頑
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自首児丸中頑
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 2: Sample output.

Testing your program at a more granular level is left to you. It is suggested that you developed a set of tests that assess the methods in the BetterDiGraph and IntuitiveTopological classes. As examples, see the assert based tests that were provided in the previous hashtable assignment. Tests will need to take into account the different behaviors that are described by the interfaces, and how different inputs may interact with them. For example, you will need to write tests for both a DAG and cyclic graph to check if IntuitiveTopological works properly.

4 Submission

The submission for this assignment has two parts: a market report (PDF), and a source code submission. Each one should be uploaded to the separate submission links on Canvas.

Writeup: Submit a PDF that includes your market report.

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. The classes must be in the default package. Be sure that you use the ZIP compression format - if you do not use the right format, we may be unable to grade your submission. Do not include your project files or any other files that are specific to your IDE.