CSC111 Project 2 Report: U of T Visual Course Map

Chris Cao, Ryan Fu, Vennise Ho

April 4, 2024

Introduction

Our main problem was that the current UofT academic calendar was hard to navigate and hard to look at the information, especially with all the brackets. Each course has its own individual requirements, and students must find their required courses on their own. Moreover, each course has its own individual prerequisites, corequisites, and exclusions. While students have access to the Art Sci Course Search and Degree Explorer, there are no other available resources that provide a visual guide for students to find the necessary requisites for their chosen courses. So for incoming students who aren't sure about what to pursue, finding the right courses for incoming students is daunting! Our project goal is to create a graph mapping courses to their prerequisites and report these to the user by telling them the different pathways to get to a course, what additional courses they need to complete a course's prerequisites, and provide a visual tree of the courses prerequisites.

Dataset Description

Arts & Science Course Data: This dataset is web-scraped data from the University of Toronto's Arts and Science Course Search. The data is formatted as follows:
Each entry contains:

- Course Name
- Course Description
- Course Hours
- Course Prerequisites
- Course Corequisites
- Course Distribution
- Course Exclusions
- Course Breadth Requirements

However, only the following are used:

- Course Name
- Course Prerequisites
- Course Corequisites

To webscrape our data, the following steps were taken:

1. We first imported the BeautifulSoup library to send http requests and scrape data.
2. Sending HTTP Requests: We sent HTTP requests to the Arts & Science Course Search server. This request fetches the HTML content of the web page.

3. Navigating the DOM: We navigated the document object model to locate the elements containing the data we wanted to scrape. Once we receive the HTML content, we parse it to extract the data we need. The following is done to identify the data:

   (a) Identifying course orders: Converting slashes and commas to be interpreted as "and," and "or," for course orders.

   (b) Identifying entry elements, such as:

      i. Course prerequisites
      ii. Course description
      iii. Course exclusions

   (c) Cleaning and Extracting Data: We parse through all the data to ensure that the produced data is relevant and clean. The following is done to clean and extract the data:

      i. Punctuation
      ii. English words
      iii. Non-course code words

   (d) Storing Data: We used Pandas to format our data to a Microsoft Excel sheet.

Computational Plan

1. Describe the kinds of data your project uses to represent your chosen domain, and how trees and/or graphs play a central role in this data representation.

   - In order to represent a courses' prerequisites, we used a hybrid of an abstract syntax tree and a graph. To do this, we created a new object - BoolOp or and/or quantifiers. These, along with courses, both inherit a general abstract expression class (much like in class AST's). Let's say a course, course A, requires 4 courses. It requires both B and C as well as one of D or E. Then, the prerequisites would be stored as:

     BoolOp('and', [B, C, BoolOp('or', [D, E])])

   - Where the first BoolOp parameter is and/or and the second is the list of courses or other BoolOps that need to be satisfied in some way.

2. Describe the major the computations your program performs, such as: building trees/graphs from a dataset or computation, data transformation/filtering/aggregation, computational models, and/or algorithms.

   - Building the AST - The program takes in a set of related courses and adds them to a function's prerequisites. By default, each course has an 'and' BoolOp and every time, a new element is added to this BoolOp. In the previous example, we would first add B, then add C, and then finally add the BoolOp('or', [D, E]). The prerequisite attribute only takes in immediate prerequisites. Prerequisites further down (such as a course needed for B in the previous example) are recursed onto in other functions.
   - Getting all Prereqs - Next, the program gives all pathways to meet a course requirement. In our previous example, getting all prereqs of A would yield [B, C, D, B, C, E], with 2 possible pathways, with each pathway stored as a set in a list. After, it cleans up the data by doing the following:
     - If a pathway has a course with a corequisite and the corequisite is not in the pathway, it is added
     - If the pathways has two courses which are exclusions of each other, the entire path is removed (as this is not possible)
     - If a pathway is redundant it is removed. For example, Our computation might yield B, C, D, E as a pathway but since a subset of that already satisfies the requirements, it is removed
   - Identifying key prereqs - After all prereqs are obtained, we give pathways important to the user by giving three parameters - Courses user has completed, courses the user wants to avoid, and credit limit.
     - Remove all courses the user has already taken from each pathway
     - If a pathway has a course the user wants to avoid (e.g. MAT137Y1 of course), then all pathways with that course are removed

- If a pathway is redundant it is removed, similar to before
- Finally the program sorts the pathways by credit and only returns the ones under the requested limit of credits.

- Explain how your program reports the results of your computation in a visual and/or interactive way.
  - The program displays the results in a visual tree mapping courses to their prerequisites. It does this by the following
    (a) The Prereqs are converted to a Tree, with and/or and courses as a node
    (b) The Tree is run under the Reingold Tilford algorithm. This algorithm take sin a tree and returns the coordinates each node must be at such that
      * At each level of the tree, all does are at least 1 unit apart
      * All parents are centered to their children
    (c) The coordinates are then plotted on a scatter plot by Plotly and edges are added.
  - Explain how your program uses Python libraries to accomplish its tasks. Refer to specific functions, data types, and/or capabilities of the library that make it relevant for accomplishing these tasks.
    * plotly.graph_objects - This library is used to plot our tree in a visually appealing manner. The function used fig.add_trace. This function takes in two lists of numbers and plots their coordinates, with each index corresponding to the same point (e.g. if we have [1,2], [2, 3], it plots the points (1,2), (2,3)) as well as another list for text (course names and and/or operators) and the rest of its parameters for formatting.

How to obtain datasets and run our program

1. Install all requirements from requirements.txt
2. Run main.py
3. Download clean_data_v4.xlsx
4. EXAMPLE - Following the instructions in the console, type in:
   (a) display
   (b) MAT237Y1
5. Here, you should see a graph visualization of all the prerequisite courses required for MAT237. You can use the upper right corner to scroll and you may also zoom in the visualization (using a mouse is helpful). An example image is included on the last page.

Changes to Project Plan

We made a couple major changes to our project: removal of the degree class and dependence function from our graph. Web scraping and, especially, data cleaning took more time than we initially thought it would, leaving less time for other things. Originally, we planned on making a degree mapper, with the ability to map a degree to its requirements. However, web scraping degree requirements was infeasible because of the variability between different degree requirement formats. Further, adding prerequisites in the format we wanted was complicated and took lots of trial and error implementing the dependence function would complicate our function further.

Discussion

0.1 Obstacles Limitations

0.1.1 Webscraping

We faced two challenges when we initially scraped data into our excel sheet. Firstly, the amount of data that needed to be scraped led to our program facing long runtimes and computer crashes. Secondly, the webpages we scraped from often contained inconsistencies and errors in their HTML structure, which led to inaccurate or incomplete data extraction. To fix these issues, we had to implement several different solutions. Firstly, we optimized our

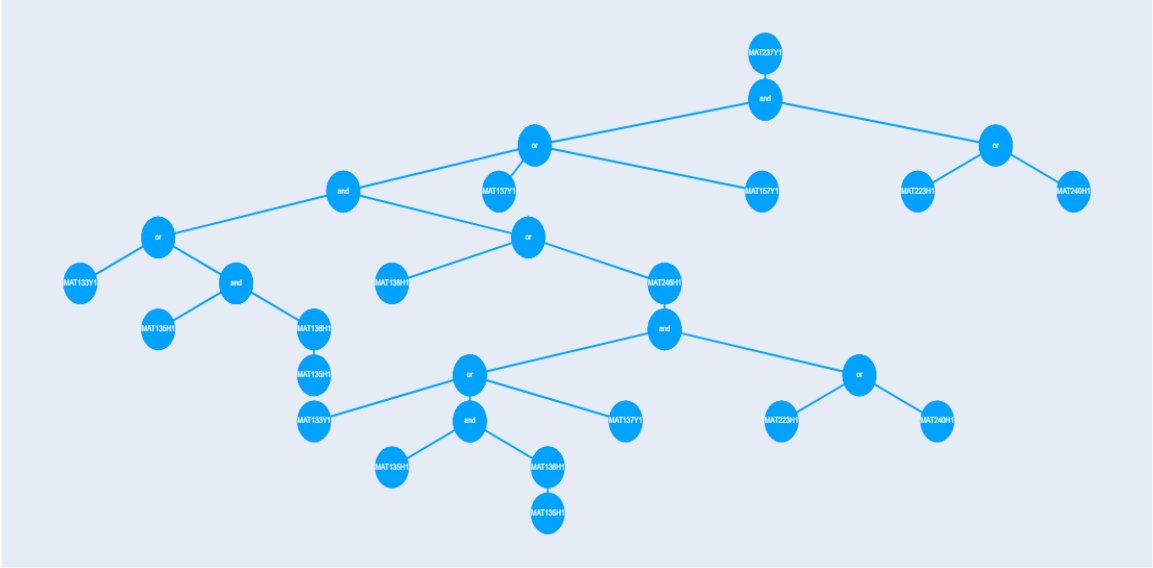

Figure 1: Example Plot

code through targeted scraping by using XPath selectors. This allowed us to more efficiently and accurately extract information compared to parsing entire HTML documents. Secondly, we used data validation techniques to ensure that our scraped data was accurate and reliable. For example, we ensured that the extracted prerequisite course codes were valid through checking that the extracted strings matched the format of a course code (XXX123H1).

0.1.2 Data Cleaning

One of the main challenges we faced was cleaning and parsing the data, particularly dealing with brackets. The original data (see output.xlsx) from the U of T ArtSci webpage for prerequisites varied widely, leading to issues when cleaning the web-scraped Excel file. Cleaning each row of prerequisite data was difficult due to extra brackets, slashes, commas, inconsistent spacing between course names, and missing courses. To address this, we developed a new algorithm specifically for data cleaning (data_cleaning.py), but handling brackets, especially nested ones, was complex and time-consuming. In the initial clean-up, removing empty brackets proved challenging to implement in code, requiring recursion and careful traversal of the string (see empty_brackets and empty_brackets_helper in data_cleaning.py). Parsing through each row of prerequisites in the cleaned data was also challenging due to the brackets, requiring yet another recursive function to manage nested brackets. Thus, we created a recursive function where we mutated the courses in each set of brackets before mutating the list as a whole. Coming up with this approach took a lot of time and experimentation and required extensive debugging to ensure the data was in the desired format.

0.2 Graph Visualization

A limitation we came across is the runtime and output of our visualization. Higher level courses have a lot more prerequisites and options for prerequisites, meaning when we are creating the visualization, the runtime increases almost exponentially. We also did not anticipate just how many ways we can get to a course. Take CSC263H1 for example. While most students reach this by taking STA237H1, it is actually possible to meet the stats requirement through STA255H1, a general stats course that can take in biostatistics or even economic statistics. As such, many courses including chemistry, biology, and economics appeared on the graph as it was possible to use these courses as requirements for CSC263H1.
Further, creating a tree including all these prerequisites and having them fit on a computer screen caused the

visualization to be extremely cluttered. This was partially resolved by adding a parameter that allowed the user to zoom in and zoom out as well as plotly built in pan function when the window is open (clocking the move arrow symbol at top right enters in pan mode allowing the user to move the tree around). As such, the display plot function works extremely well for low level courses, or higher level courses with only a few prerequisites.
Another obstacle was determining the coordinates for the tree plot. Very few libraries exist to install trees and they are extremely limited and hard to install. As such, we decided to create essentially our own algorithm that would take in a tree, determine the coordinate each node should be at, and plot it. In order to determine the coordinates, we used the Reingold Tilford algorithm. This algorithm assigns coordinates in a way that 1) ensures parents are centered over children and 2) no overlapping occurs between each node horizontally. The process of shifting the children and parents was incredibly complex and long as it required 3 functions, each function representing one step int he algorithm.

0.3 Results

The results of our computational exploration helped to answer one of the questions we posed since we removed the dependence function. We were able to answer the question of 'What pathways and prerequisites can I take to get to a given course'. This is clearly shown in our data visualization, especially in the lower level courses (ex. MAT237). We were also able to visually display this as a tree, especially for lower level courses. This solved our other problem of the academic calendar being extremely unreadable and allowed for a better, more visual, representation of a courses' prerequisites. In the future, we will be able to use our program to better plan our courses, especially the upper year ones that require a lot of

0.4 Next Steps

For next steps, we would like to continue implementing dependents and degrees, as in our original plan. We were also wondering if we could figure out the optimal path from a starting point (where some courses are already taken) and also figure out how on track someone is to their degree (like U of T's degree explorer) or an upper year course, including what courses they're missing to fulfill the requirements. Further, if we were able to web scrape the data, we could extend the project to other universities as well as expand it to a possible course recommendation system that gives the user good courses to take base don what they currently completed. This can be done by looking at what other students with similar course loads have taken (however, this would require a dataset with this information that is not readily available).

References

- Duke, J. (2022, December 7). How to crawl a web page with Scrapy and python 3. DigitalOcean. https://www.digitalocean.c to-crawl-a-web-page-with-scrapy-and-python-3
- GfG. "Python Web Scraping Tutorial." GeeksforGeeks, GeeksforGeeks, 11 Mar. 2024, www.geeksforgeeks.org/python-web-scraping-tutorial/.
- "Line and Scatter Plots." Plotly Python Graphing Library — Line and Scatter Plots, Plotly, plotly.com/python/line-and-scatter/.
- Pandas Documentation. Pandas, 2022, https://pandas.pydata.org/.
- "Plotly Express.scatter." Plotly Express.scatter, Plotly, plotly.com/python-api-reference/generated/plotly.express.scatter.ht
- "Reingold-Tilford Algorithm Explained with Walkthrough." Towards Data Science, Towards Data Science, 14 Sept. 2020, towardsdatascience.com/reingold-tilford-algorithm-explained-with-walkthrough-be5810e8ed93.
- Schafer, C. (2020, January 9). Python pandas tutorial (part 1): Getting started with data analysis - installation and loading data. YouTube. https://www.youtube.com/watch?v=ZyhVh-qRZPAlist=PL-osiE80TeTsWmV9i9c58mdDCSs
- Tree-plots. Tree-plots in Python. (n.d.). https://plotly.com/python/tree-plots/.
- University of Toronto. Academic Calendar, artsci.calendar.utoronto.ca/search-courses.
- "Beautiful Soup Documentation¶." Beautiful Soup Documentation - Beautiful Soup 4.4.0 Documentation, beautiful-soup-4.readthedocs.io/en/latest/. Accessed 4 Apr. 2024.