CS461 - Coding Project 1

Graph Traversal

By: Manuel Buffa

Intro

For this project we are given two files. The first has a list of cities with associated coordinates. The second has a list of adjacencies, city-to-city connections. The goal is to build a graph structure with the datasets, get user inputs for a starting point and ending point, and then find a route using various traversal methods, namely Breadth-First Search (BFS), Depth First Search (DFS), ID-DFS, Best-First Search, and A\* (A-star) Search.

Coding Process

Since I am still in the process of learning some aspects of Python fundamentals, I began by doing some searches for user input examples. Then I searched for graph implementations in Python. Some small modifications were necessary in the example that was provided. Being familiar with Pandas, I easily imported the .csv file as a Pandas dataframe. Modification to the .csv was required, so I added the labels to better accommodate the read\_csv() function. Next, I tried using NumPy for importing the adjacencies file. This worked initially but I had issues when adding graph edges.

At this point I had (mostly) successfully tested the functions described above, and I avoided using generative AI. Then I started running into problems. I needed to convert the underscores to species after reading from the files, and I needed to read from the .txt file in a better manner. This required assistance from our friend, ChatGPT. The conversation beginning is in the references\*. The initial suggestions were small and worked great for cleaning up the loops, converting the characters, and completing the graph structure build-out. Next, I started researching the traversal functions. This led me to the NetworkX library. This wonderful library has multiple traversals built-in and easy to use node/edge manipulation functions. To use these I needed to switch the entire structure to the NetworkX(nx) graph, therefore I started a new Notebook, v1, using the nx library. The initial Notebook is submitted as v0. At this point a fresh chat thread is needed, a link to this entire conversation is in the references (\*conversations w/ Chat can’t be shared when they have a user provided image, in this case a screenshot of the first entries in Adjacencies.txt).

After a bit more conversation in a fresh dialogue, and quite a bit more reading of the NetworkX documentation, the point-to-point searches are not specifically supported with built-in functions. I returned to v0 for ease of function creation. I copied user-made functions from GeeksforGeeks into a temporary Notebook, code\_blocks\_beta, that contained the functions designed in the dialogue with ChatGPT-o1. I used .txt files for tracking iterations of the program and for copying code for ChatGPT to help complete/debug. I started a new dialogue with ChatGPT-4o and sent the original program and the temporary functions. With this dialogue, I created yet another Notebook titled HWv2.

This entire dialogue is also in the references and includes the final correspondences that helped complete the program. Multiple files have been submitted with this document, including all four Notebooks. The creation/modification order is HWv0, then HWv1, then code\_block\_beta, and finally HWv2. After running multiple tests on the program, I needed to have a better way to represent the graph traversal. I tested a couple of plots, and then added a function that plots a path, if present.

Program Operation

Once the completed program had been debugged, I compared the paths that were found by each method from city to city. The calculation time and distance was produced for each method as well as a traversal plot for each path. These metrics worked well for comparing the different routes provided. The path plots are especially helpful in understanding the functionality of each method. A more detailed operational analysis of the program will be given during the submitted video presentation.

Note: I attempted to track the memory usage but couldn’t get a good measure of the differences in methods. I am sure I could get that problem solved with more time and experimentation, but the several attempts I tried didn’t work.

Below are some test results:

city\_1: Attica; city\_2: Salina

| Method | BFS | DFS | ID-DFS | Best-First | A\* |
| --- | --- | --- | --- | --- | --- |
| Path | 'Attica', 'Kiowa', 'Coldwater', 'Pratt', 'Hutchinson', 'McPherson', 'Salina' | 'Attica', 'Harper', 'Anthony', 'Argonia', 'Caldwell', 'Wellington', 'Oxford', 'Mayfield', 'Mulvane', 'Andover', 'Newton', 'El Dorado', 'Hillsboro', 'Lyons', 'Salina' | 'Attica', 'Kiowa', 'Coldwater', 'Pratt', 'Hutchinson', 'McPherson', 'Salina' | 'Attica', 'Harper', 'Anthony', 'Argonia', 'Rago', 'Viola', 'Sawyer', 'Pratt', 'Hutchinson', 'McPherson', 'Salina' | 'Attica', 'Harper', 'Anthony', 'Bluff City', 'Mayfield', 'Mulvane', 'Andover', 'Newton', 'McPherson', 'Salina' |
| Calc. Time | 0.000099 seconds | 0.000068 seconds | 0.000130 seconds | 0.000154 seconds | 0.000475 seconds |
| Dist. Travel | 360.01 km | 563.13 km | 360.01 km | 425.29 km | 287.03 km |

city\_1: Bluff City; city\_2: Winfield

| Method | BFS | DFS | ID-DFS | Best-First | A\* |
| --- | --- | --- | --- | --- | --- |
| Path | 'Bluff City', 'South Haven', 'Mulvane', 'Andover', 'Winfield' | 'Bluff City', 'Mayfield', 'Mulvane', 'Andover', 'Newton', 'Emporia', 'Augusta', 'Winfield' | 'Bluff City', 'South Haven', 'Mulvane', 'Andover', 'Winfield' | 'Bluff City', 'South Haven', 'Mulvane', 'Andover', 'Winfield' | 'Bluff City', 'Mayfield', 'Mulvane', 'Andover', 'Winfield' |
| Calc. Time | 0.000105 seconds | 0.000165 seconds | 0.000127 seconds | 0.000140 seconds | 0.000459 seconds |
| Dist. Travel | 163.24 km | 397.57 km | 163.24 km | 163.24 km | 142.45 km |

Results and Conclusions

As discussed in class, the DFS can be unreliable due its the tendency to find the first path that works and keep that result. This can lead to a shorter search time but is consistently yielding the longest path of the methods. In both cases above, the A\* Search yielded the best route for distance but took the longest to calculate. For this graph, BFS and ID-DFS both yield the same paths but the ID-DFS takes longer to calculate. Best-First seems to be the least reliable overall, but that could be due to implementation of the heuristic. These results are confirmed further by the plotting of different paths. One of the more interesting paths can be found using Best-First Search. The heuristic approach of taking the shortest immediate path from the node can yield a strange route from start to end. More tests/demonstrations will be performed during the video presentation.

References:

Rachit Belwariar, A\* Search Algorithm, Updated: 30 July 2023, <https://www.geeksforgeeks.org/a-search-algorithm/>

Shambhavi Singh, Best First Search (Informed Search), Updated: 21 December 2023, <https://www.geeksforgeeks.org/best-first-search-informed-search/>

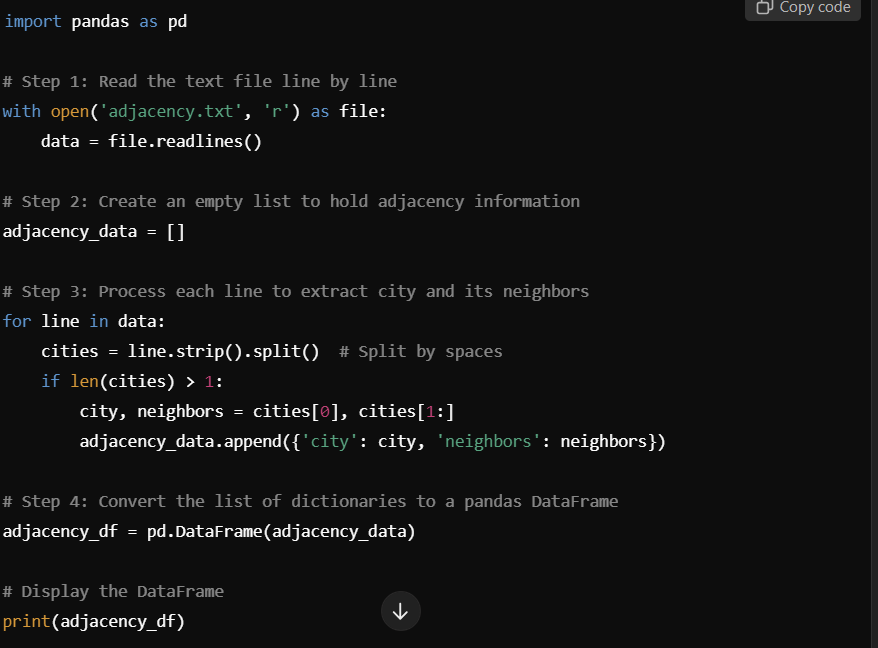
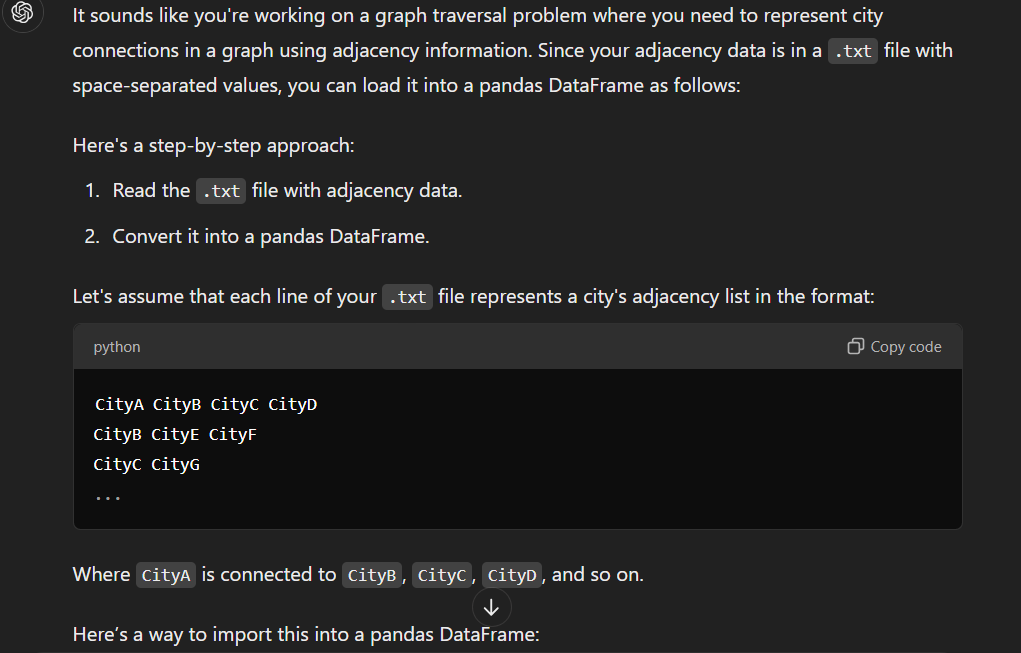
GeeksforGeeks, Iterative Deepening Search or Iterative Deepening Depth-First Search, Updated: 20 September 2024, <https://www.geeksforgeeks.org/iterative-deepening-searchids-iterative-deepening-depth-first-searchiddfs>

GeeksforGeeks, Depth-First Search or DFS for a Graph, Updated: 30 September 2024, <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>

GeeksforGeeks, Breadth-First Search or BFS for a Graph, Updated: 26 September 2024, <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

Bhatele, Shivam. “Graph Implementation in Python.” Experience Stack. April 20th 2023. <https://experiencestack.co/graph-implementation-in-python-916fc3b6a8a>

OpenAI, ChatGPT-4o-canvas {prompt: Hi Chat. I am working on a python graph traversal program. I have much of the core parts written. I have built a pandas dataframe from a .csv file with city\_name, latitude, and longitude. I have used the object to build my graph vertices. Now, I have a list of adjacencies (space separated) in a .txt file that I need to import into a dataframe.}



Entire chat threads are below.

OpenAI, ChatGPT-o1, dialogue #2, <https://chatgpt.com/share/6707e3da-2424-800b-8f0a-605db6510abc>

OpenAI, ChatGPT-4o, dialogue #3, <https://chatgpt.com/share/6707e42f-b8a8-800b-b116-b85a6b96b0f7>