Program 1: Search Methods

The first program of the semester involves route-finding using multiple algorithms.

For this assignment, you'll be doing a small example of a large problem—route-finding.
Given a list of cities and their adjacencies—from city A, what cities are next to it—can a route be found from city A to city X?
As discussed in class I would like you to utilize your resources to rapidly implement a version of each of the major methods we've discussed (some are pending) in chapter 2.

This exercise will take the form of an experimental report detailing your findings of a comparison of each of the following methods:

1. undirected (blind) brute-force approaches
   - **breadth-first search**
   - **depth-first search**
   - **ID-DFS search**
2. Heuristic Approaches
   - **best-first search**
   - **A* search**

     (5 Total Methods)

You'll want to think about the initial conditions and how to check for a "valid" sequence and GOAL check at each step of the search process.
NOTE: It's not necessary to reinvent the wheel here, many implementations of these methods exist and you might even get a "fair" solution from an LLM. **(If you do use a generative model to help you code, please be sure to include the prompts you used as part of the generation process -- this isn't to grade your effort or use of the model, but simply good practice for source documentation in prompt-centric engineering!)** Of course, you are **absolutely free (and always encouraged)** to generate your own implementations of these systems, but please use the pseudocode and function descriptions from your text as a guideline.

Programming details:
• You can generate your program in C, C++, Python, C#, or Java. (If you want to use someother language I'm willing to discuss it but you'll need to come to me in advance.)

• You're given 2 data files:
◦ The first is a list of all the cities we know about—mostly small towns in southern Kansas —and the latitude and longitude of each.
# Names have been tweaked so that city names consisting of more than one word have an underscore rather than a space between the words (South_Haven rather than South Haven), to simplify input.

◦ A file (CSV) listing each town (pair) as a related adjacent node.
# be aware adjacency is symmetric: If A is adjacent to B, then B is adjacent to A. This may not be listed comprehensively if your method requires that bidirectional connections be explicitly

stated, you may need to generate additional pairs for the symmetrical connection to work. {That is, tell the program that it's possible to go from listed A as adjacent to B or listed B as adjacent to A.} --> Be sure to take this into account when setting up your program's data structures. If adjacency is listed in either direction, it should be considered present in both directions.

• Your program should:

- Ask the user for their starting and ending towns, making sure they're both towns in the database.
- Ask them then to select the search method they wish to use to find a route to the destination.
    - If that route exists, the program should then print the route the method found in order, from origin to destination.
    - *If you want to get fancy, you might see how the route generated looks as a map (either a map of connected states as we looked at in class, or as a projection in 2D space based on location and connectivity).*

      Note that your database is very limited, and many of the real-world roads are left out for simplicity.*

- Your program should also:
    - measure and print the total time needed to find the route (and include a time-out).
    - calculate and display the total distance (node to node) for the cities visited on the route selected.
    - (opt) determine the total memory used (scale of the arrays) to find the solution.
- Return to the search method selection and allow a new method to be selected for comparison.

**A real mapping application gives directions using much more complicated metrics, including variable road conditions, construction closure, fuel economy, and toll costs.*

Submission: Your submission will include:
1) A **link to the repo** -- Replit, GitHub, Jupiter or GitLab (or zip file of your project from your IDE)
2) A **video** *demonstrating* and *explaining* both the **CODE** and the requested **FUNCTIONALITY**.
You should accomplish this demonstration by selecting a set of start and end points that show off the operation of your code. [Suggestions on candidate search pairs will be provided later]
Please include a discussion of your observations of the difference in execution time for each method.
*You should **SPEAK** while showing the code, be sure to highlight how each of your functions are finding the route.*
This video should be **3-5** minutes in length [you can/should edit out those boring segments of the runtime] -- **please do not exceed 7 minutes** for your video.