Assignment #1

Assignment Overview

In this assignment you will implement search strategies for two classical AI problems:

- finding the best route between two cities; and
- playing the game of tic-tac-toe.

Background

This assignment will give you a chance to build working solutions ("apps") for two problems and demonstrate your knowledge of:

- **Uninformed search**, namely BFS and DFS
- **Heuristic search**, particularly the A* search algorithm
- **Adversarial search**, particularly MINMAX

See chapters 2 and 3 of the textbook for additional details.

Project Specification

This is a group assignment.
Students are encouraged (but not required) to work in groups of max 3 students.
Ideally, the group should be organized around three main tasks / duties:

- Design of the solution ("architect" role)
- Coding of the solution ("developer" role)
- Documentation of the solution ("reporter" role)

You are required to indicate in your report "who did what" and document the entire process, from sketching the original plans and dividing up the tasks all the way to polishing the interface, testing the solution, and preparing the report.

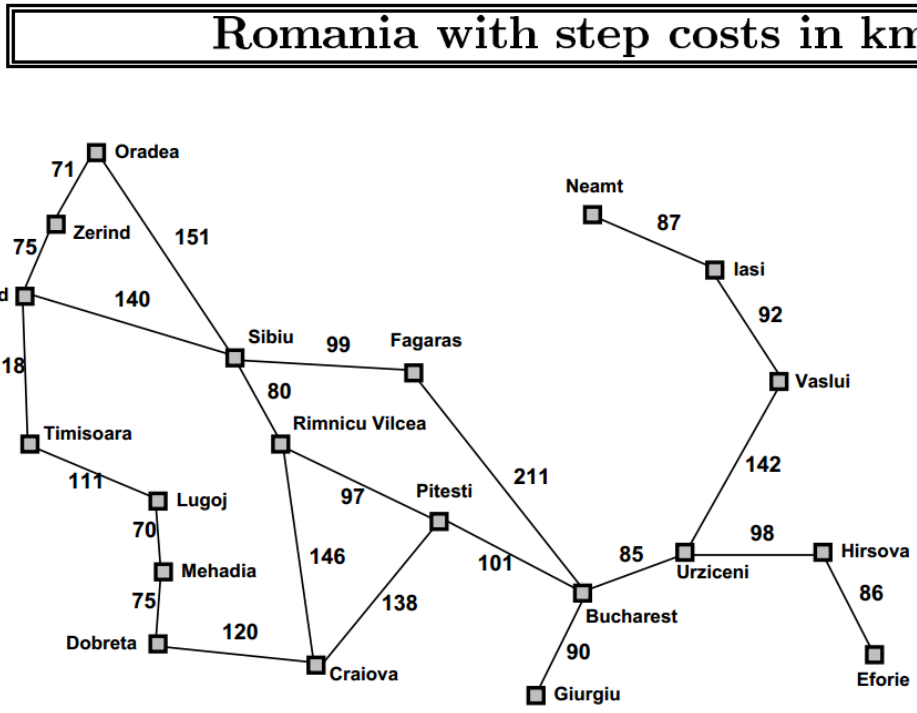
The basic functionality for each part is specified below.

Part 1 – Shortest route

In this part you will implement three solutions for the shortest route problem (DFS, BFS, and A*), using the map of Romania below[1] and the straight-line distance to the destination (Bucharest) as your heuristic function, h(n).

Your program ("**baseline solution**") will:

1. Print a brief (2-5 lines) message explaining the purpose of this "app".
2. Prompt for:
   - The city which the user plans to depart from (*remember that the destination is hard-coded as Bucharest*);
   - The choice of algorithm (A/B/D, for A*/BFS/DFS).
3. Process the input and run the corresponding algorithm.
4. Print the solution (name of the algorithm + itinerary + distances + total distance) and ask if the user wants to submit another city/algorithm or quit.



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

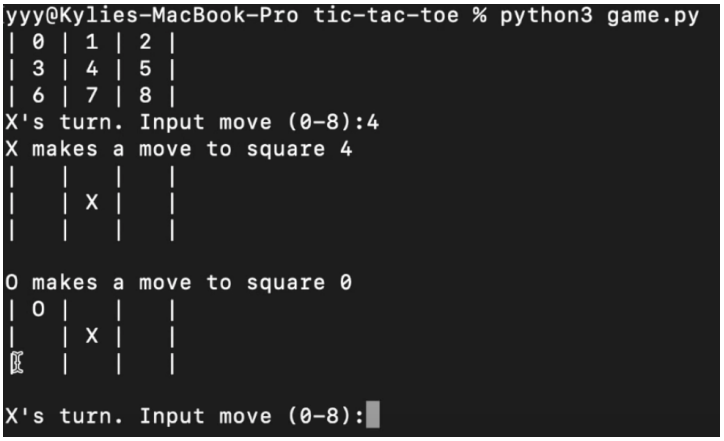
[1] Source: Russell and Norvig (2020)

Part 2 – Adversarial search

In this part you will implement a solution for the tic-tac-toe game using the MINMAX algorithm. There is no need to implement alpha-beta pruning. If you do, you will be eligible for (max 10) bonus points.

Your program ("**baseline solution**") will:

1. Print a brief (1-3 lines) message explaining the purpose of this "app".
2. Display the board. See screenshot below[2] for ideas.
3. Prompt the user to play (as 'X') by indicating numerically where they want to place their 'X' on the board.
4. Play (as 'O') the computer's move.
5. Repeat steps 3 and 4 until there is either a winner or the game ends in a tie.
6. Ask if the user wants to play another round or quit.



Deliverables

You must submit (via Canvas):

- **One** file **a1_FAUusername.py** (where "**FAUusername**" is the username of one of the team members); in my case the file would be called **a1_omarques.py**) that allows me to call the code for either solution.
    - This is your source code solution; be sure to include your names, date, assignment number and comments describing your code.
    - Only one submission per team is enough, if the names of the team members appear in the comments and report.
- (Optional) A **README.md file** with installation instructions, dependencies, etc.
- A **detailed report** (PDF, markdown, and/or HTML) with detailed "project notes" (describing what my TA and I cannot see by looking at your source code and/or running your program), screenshots, references, etc.
    - Examples: design decisions, documented limitations, future improvements, etc.
- **Screenshots of representative results** produced by your code.
    - Make sure to show input and output for **at least 6 runs of part 1** (2 cities x 3 algorithms)).
    - Make sure to show input and output for **at least 2 complete games of part 2.**

[2] Source: Kylie Ying https://youtu.be/8ext9G7xspg

Notes and Hints:

- Start by breaking the problems down into parts and solve smaller problems before producing the final solution.
- Use *good code* as starting point (and make sure to reference them accordingly in your report!)
    - Examples:
        - Code from the (GAIA) textbook (chapters 2 and 3)
        - Tic-tac-toe solution from Kylie Ying: https://youtu.be/8ext9G7xspg (video)
        - Official repository of sample solutions for Russell/Norvig's textbook (AIMA): https://github.com/aimacode/aima-python
- Try to handle special cases and prevent runtime errors to the best of your knowledge.
- **Don't overdo it!** You might want to try some of the bonus points options below but try not to risk breaking a good solution by adding bells and whistles to it.

Bonus opportunities:

If you successfully implement one or more of the bonus options below **without breaking the baseline solution** (*) you will earn extra points (max 10% each):

1. **Web-based solution**
    - If you implement an **elegant** web-based solution (e.g., using Flask[3]) for either problem, you will earn up to extra 5 points per problem.
2. **Jupyter notebook / Google Colab**
    - If you build a polished notebook (and make it available via Colab) with your solution (and report?), you will earn up to extra 10 points.
3. **Alpha-beta pruning option for tic-tac-toe game**
    - If you implement alpha-beta pruning correctly and demonstrate the difference between the modified algorithm and the baseline MINMAX, you will earn up to extra 10 points

(*)
A common mistake students make is to attempt bonus items before producing a "perfect 100" baseline solution.

Make no mistake: if your baseline solution doesn't meet **all** grading criteria (see rubric on Canvas), you are not eligible for bonus points. Period.

[3] https://flask.palletsprojects.com/en/2.0.x/