CS 4341 Individual Project Assignment 2 (100 Points)

Due: 11:59 p.m. on 09/22/2024

Project Objective: The goals of this assignment are to help you understand the agents and the search methods that we discuss in Week 3 and Week 4, respectively.

Note:

(1) *This project is to be done by **EACH STUDENT** individually. No help besides the textbook, materials, and the instructor/TA should be taken. Copying any answers or part of answers from other sources, including your classmates, will earn you a grade of zero.*

(2) *Your program must be developed and implemented in the PyCharm-like IDE, or 10% of the graded score is deducted. Please check and choose the one from here: https://realpython.com/python-ides-code-editors-guide/, as the suggestion. Note that we **DO NOT** use the Jupyter as the IDE.*

(3) *Assignments are accepted in their assigned Canvas drop box without penalty if they are received by 11:59PM EST on the due date, or 10% of the graded score is deducted for the late submission per day. Work submitted after one week of its original due date will not be accepted.*

Project Deliverables: Submit a **zip** file that includes the **.py files**, i.e., **SimpleProblemSolvingAgent.py, RomaniaCityApp.py**, and other **python files/folders (if needed)**, to complete your individual project assignment to Canvas.

In this project, we would like to develop a Simple Problem-Solving Agent to search the best path between any two cities in Romania shown in Figure 3.1 of your reference textbook, using the *Greedy Best-First Search*, *A* Search Algorithm*, *Hill-climbing*, and *Simulated Annealing* that we discuss in our lectures.

1. Develop and implement a **SimpleProblemSolvingAgent.py** (SPSA) class that you can use to instantiate a SPSA agent object to find the best path between any two Romania cities. You can review and investigate the **"SimpleProblemSolvingAgentProgram" class in search.py**, as the template and the starting point, provided from **https://github.com/aimacode/aima-python**. Note that you **CAN** develop your own class for this agent based upon Section 3.1 and 3.2 of your reference textbook if you prefer not to use the sample codes provided by **search.py**.

2. In the SPSA class, you need to develop and implement two informed searching algorithms: Greedy Best-First Search (i.e., **best_first_graph_search (problem, f)**) and A* Search (i.e., **astar_search (problem, h)**), and two local search algorithms: **hill_climbing(problem)** and **simulated_annealing(problem, schedule=exp_schedule())**, respectively. All of these functions, as the reference and the starting point, can be found in **search.py**. Note that you **CAN** develop your own functions for this agent based upon Figure 3.7, Section 3.5.2, Figure 4.2, and Figure 4.5 of your reference textbook, respectively, if you prefer not to use the sample codes provided by **search.py**.

3. The SPSA object will take a graph, i.e., the Romania map, as an input shown in Figure 3.1 of your reference textbook and the city coordinates, and any two cities in the map to find the best path between them using all the above algorithms, respectively. Both **romania_map** and **romania_map.locations** have been included in the **search.py**. Note that you **CAN** develop your own graph for this map based upon Figure 3.1 if you prefer not to use the sample codes provided by **search.py**.

4. Develop and implement a separate python App functional program called **RomaniaCityApp.py** using the given construct and do the following:

```
def main():
    pass

if __name__ == "__main__":
    main()
```

(a) Your app will prompt and ask a user to enter ***any two cities*** from the **romania_map**. If these two cities are the same or either one of them or both of them cannot be found in the Romania map, please ask the user to enter them again until the two cities are valid.

(b) Your app will then create a SPSA agent object from the **SimpleProblemSolvingAgent class** and search the best path between these two cities by using the **Greedy Best-First Search**, **A* Search**, **Hill Climbing** and **Simulated Annealing** algorithms, respectively. For each search algorithm used, the output should include (i) the search method name, (ii) the total cost of the path, and (iii) all the intermediate cities between them, including the start and the end cities so that you can see the difference among these four searching algorithms.

(c) At the end, your app will ask the user if they would like to find the best path between any two cities again. If yes, repeat (a) and (b). If not, terminate the program and then display **"Thank You for Using Our App"**.

Note:

- The following is the sample output. All the **red bold words** are entered by the user.
- You can assume that all the users' inputs are valid.
- You don't need to consider if there are any typos, case sensitive, etc., in the inputs.
- Each path generated includes a sequence of unique cities. That is, no cities are repeated in a path.

Here is the Sample Output:

Here are all the possible Romania cities that can be traveled:

['Arad', 'Bucharest', 'Craiova', 'Drobeta', 'Eforie', 'Fagaras', 'Giurgiu', 'Hirsova', 'Iasi', 'Lugoj', 'Mehadia', 'Neamt', 'Ora
'Pitesti', 'Rimnicu', 'Sibiu', 'Timisoara', 'Urziceni', 'Vaslui', 'Zerind']

Please enter the origin city: **Boston**

Could not find Boston, please try again: **Arad**

Please enter the destination city: **Arad**

The same city can't be both origin and destination. Please try again.

Please enter the origin city: **Arad**

Please enter the destination city: **Bucharest**

Greedy Best-First Search
Arad → Sibiu → Fagaras → Bucharest
Total Cost: 450

A* Search
Arad → Sibiu → Rimnicu → Pitesti → Bucharest
Total Cost: 418

Hill Climbing Search
Arad → Sibiu → Fagaras → Bucharest
Total Cost: 450

Simulated Annealing Search
Arad → Sibiu → Rimnicu → Craiova → Pitesti → Bucharest
Total Cost: 605

Would you like to find the best path between the other two cities? **yes**

Please enter the origin city: **Bucharest**
Please enter the destination city: **Arad**

Greedy Best-First Search
Bucharest → Fagaras → Sibiu → Arad
Total Cost: 450

A* Search
Bucharest → Pitesti → Rimnicu → Sibiu → Arad
Total Cost: 418

Hill Climbing Search
Bucharest → Fagaras → Sibiu → Arad
Total Cost: 450

Simulated Annealing Search
Bucharest → Pitesti → Craiova → Rimnicu → Sibiu → Oradea → Zerind → Arad
Total Cost: 762

Would you like to find the best path between the other two cities? **no**

Thank You for Using Our App

Grading Criteria: Your solutions must be complete and clear.

Checkpoints	Points Possible
(1) Proper Naming Conventions and Program Documentation on Your Codes	10 Points
(2) Executable Codes: ▪ SimpleProblemSolvingAgent.py ▪ RomaniaCityApp.py ▪ Other python files/folders (if needed)	15 Points
(3) Greedy Best-First Search() Implementation	10 Points
(4) A* Search() Implementation	10 Points
(5) Hill_Climbing() Implementation	10 Points
(6) Simulated_Annealing() Implementation	10 Points
(7) Prompt and ask a user to enter any two cities	10 Points
(8) In case of an error, e.g., same cities or no such cities, ask the user to enter them again until the two cities are valid	10 Points
(9) The output should include the results for the entered two cities (See the sample outputs). ▪ Test Cases: Arad and Bucharest, Bucharest and Craiova, and Craiova and Arad ▪ Search Method Name ▪ Total Cost between Two Cities ▪ All the Intermediate Cities Between Them ▪ Ask the Users If They Would Like to Find the Optimal Path Between Any Two Cities Again ▪ Terminate the Program and Then Display "Thank You for Using Our App." ▪ Your running output format generated from your program should be similar to the sample output's.	15 Points
Total	100 Points