**Student Name <- AMEYA HUJARE**

**CWID: A20545367**

**CS 480 Spring 2024 Programming Assignment #01**

Due: **Sunday, February 18, 2024, 11:59 PM CST**

Points: **100**

**Instructions:**

1. Place **all your deliverables (as described below) into a single ZIP** file named:

LastName\_FirstName\_CS480\_Programming01.zip

1. Submit it to Blackboard Assignments section before the due date. **No late submissions will be accepted**.

**Objectives:**

1. (100 points) Implement and evaluate two informed search algorithms.

**Input data files:**

You are provided two CSV (comma separated values) files (see Programming Assignment #01 folder in Blackboard):

* driving.csv - with **driving distances** between state capitals.
* straightline.csv - with **straight line distances** between state capitals.

You **CANNOT** modify nor rename input data files. Rows and columns in those files represent individual state data (state labels/names are in the first row and column). Numerical data in both files is either:

* a non-negative integer corresponding to the distance between two state capitals,
* negative integer -1 indicating that there is no direct “road” (no edge on the graph below) between two state capitals.

**Deliverables:**

Your submission should include:

* Python code file(s). Your py file should be named:

cs480\_P01\_AXXXXXXXX.py

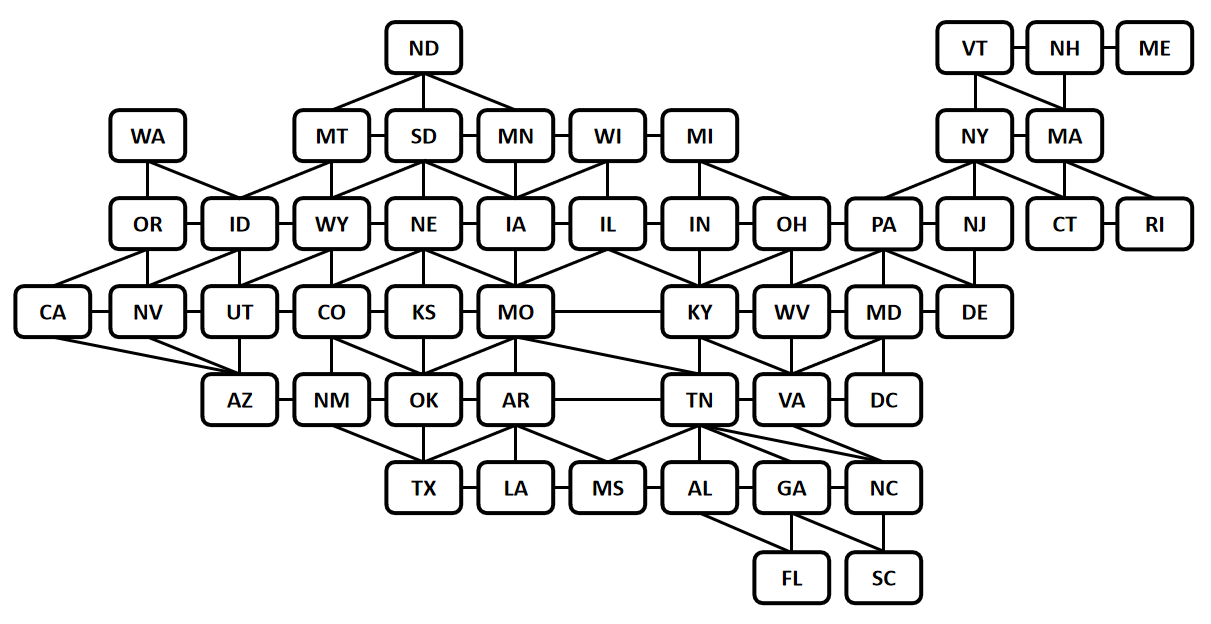
where AXXXXXXXX is your IIT A number (this is REQUIRED!). If your solution uses multiple files, makes sure that the main (the one that will be run to solve the problem) is named that way and others include your IIT A number in their names as well.

* this document with your results and conclusions. You should rename it to:

LastName\_FirstName\_CS480\_Programming01.doc

**Problem description:**

Consider the graph presented below (fig. 1). Each node represents a single state (or the District of Columbia (DC)). If two states are neighbors, there is an edge between them.



*Figure 1: A graph representing all 48 contiguous US states and District of Columbia.*

Assume that edge weights represent **driving distances between state capitals**.

Your task is to implement in Python two informed search algorithms:

* Greedy Best First Search algorithm, and
* A\* algorithm,

and apply them to find a path between two state capitals using provided data.

Your program should:

* Accept two (2) command line arguments corresponding to two states / state capitals (initial and goal states) so your code could be executed with

python cs480\_P01\_AXXXXXXXX.py GOAL INITIAL

where:

* + cs480\_P01\_AXXXXXXXX.py is your python code file name,
  + GOAL is the label/name of the initial state.
  + INITIAL is the label/name of the initial state,

Example:

python cs480\_P01\_A11111111.py WA TX

If the number of arguments provided is NOT two (none, one, or more than two), your program should display the following error message:

ERROR: Not enough or too many input arguments.

and exit.

* Load and process both input data files provided (assume that input data files are ALWAYS in the same folder as your code - this is REQUIRED!). Make sure your program is **flexible enough to accommodate different input data sets** (with a different graph of states and distances). Your submission will be tested using a different set of files!
* Run Greedy Best First Search and A\* algorithms searches to find a path between INITIAL and GOAL states and measure execution time (in seconds) for both methods.
* Report results on screen in the following format:

Last Name, First Name, AXXXXXXXX solution:

Initial state: INITIAL

Goal state: GOAL

Greedy Best First Search:

Solution: STATE1, STATE2, STATE3, …, STATEN-1, STATEN

Number of expanded nodes: AAAA

Number of stops on a path: X1

Execution time: T1 seconds

Complete path cost: Y1

A\* Search:

Solution: STATE1, STATE2, STATE3, …, STATEN-1, STATEN

Number of expanded nodes: AAAA

Number of stops on a path: X2

Execution time: T2 seconds

Complete path cost: Y2

where:

* + AXXXXXXXX is your IIT A number,
  + INITIAL is the label/name of the initial state,
  + GOAL is the label/name of the initial state,
  + AAAA is the number of expanded nodes (including the root node),
  + STATE1, STATE2, STATE3, …, STATEN-1, STATEN is a solution represented as a list of visited states (including INITIAL and GOAL states), for example: IL, IA, NE,

If no path is found replace appropriate information with:

Solution: NO SOLUTION FOUND

Number of stops on a path: 0

Execution time: T3 seconds

Complete path cost: 0

Pick INITIAL / GOAL state pair (with at least 5 states between them) and run both Greedy Best First and A\* algorithms to find the path between them. Repeat this search ten (10) times for each algorithm and calculate corresponding averages. Report your findings in the Table A below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **TABLE A: Results comparison** | | | | |
| Algorithm | Visited states | Number of visited states | Number of expanded nodes | Path cost | Average search time in seconds |
| Greedy Best First Search | ['NH', 'MA', 'NY', 'PA', 'WV', 'KY', 'TN', 'AL'] | 8 | 7 | 1584 miles | 0.015644 |
| A\* | ['NH', 'MA', 'NY', 'NJ', 'DE', 'MD', 'DC', 'VA', 'NC', 'GA', 'AL'] | 11 | 18 | 1352 miles | 0.002403 |

What are your conclusions? Which algorithm performed better? Was the optimal path found? Write a summary below

|  |
| --- |
| **Conclusions** |
| The A\* Search algorithm found the optimal path from NH to AL, as it produced a path with a lower cost (1352) compared to the Greedy Best First Search algorithm (1584).  **Greedy Best First Search 🡪** It expanded fewer nodes (7) compared to A\* Search (18), indicating a more efficient exploration in terms of node expansion. **However, GBFS did not find the optimal path.**  **A Search\*🡪** While it expanded more nodes, **A\* Search provided the optimal path with a lower cost.**  Both algorithms completed the search in the same amount of time, indicating that the difference in performance lies mainly in the search strategy rather than computational efficiency.  In summary, while Greedy Best First Search expanded fewer nodes, it did not find the optimal path. A\* Search, despite expanding more nodes, found the optimal path with a lower cost. **Therefore, A\* Search performed better in this scenario**. |